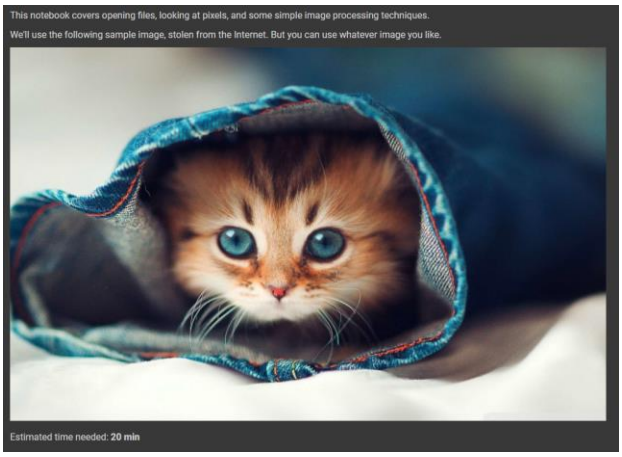
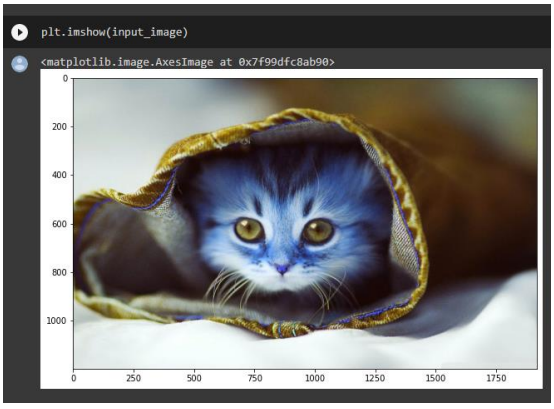
Abdul Moiz -19022

# Lab-1

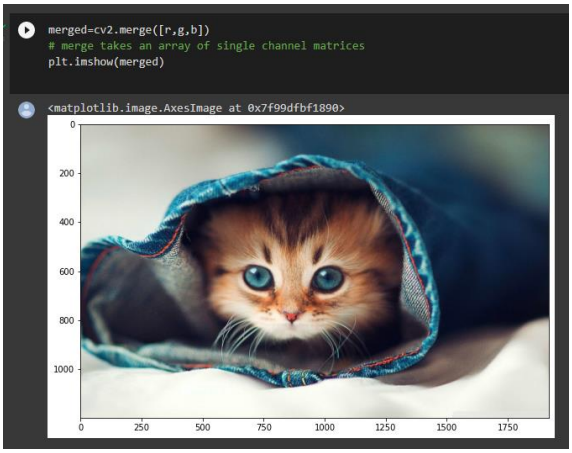
**2. Implement a simple python program with an image of your choice and demonstrate all the concepts (attach screenshots).**

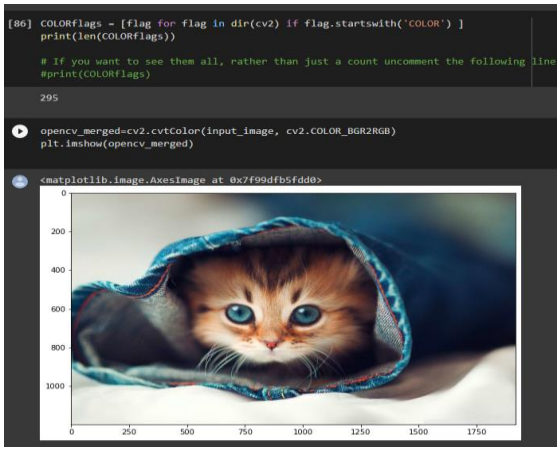










Abdul Moiz -19022

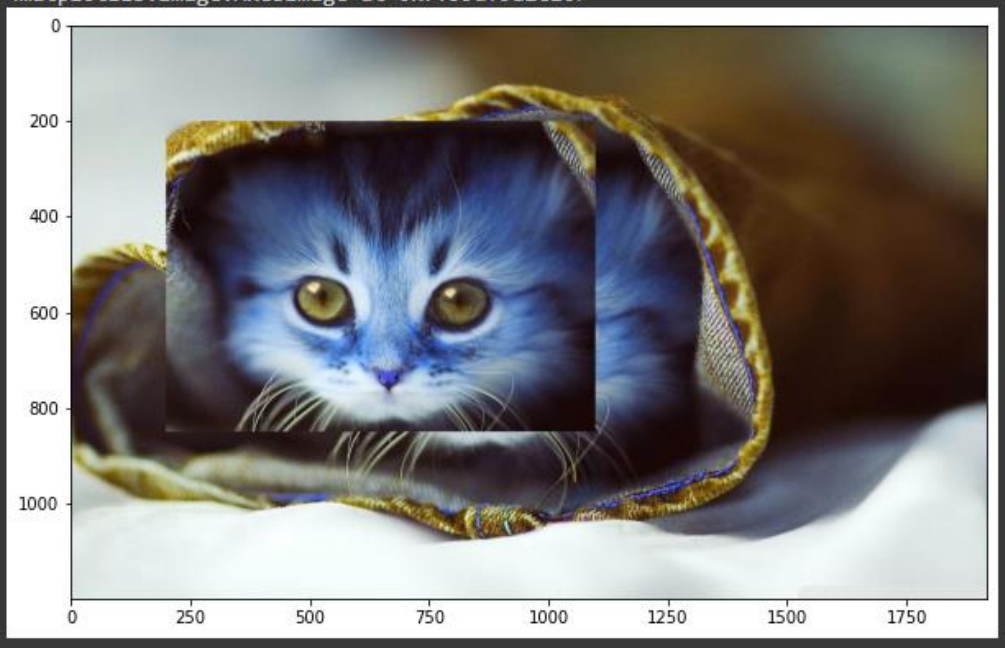
```
[88] pixel = input_image[100,100]
     print(pixel)

     [202 212 206]

[89] input_image[100,100] = [0,0,0]
     pixelnew = input_image[100,100]
     print(pixelnew)

     [0 0 0]
```

```
dogface = input_image[250:900, 400:1300]
plt.imshow(dogface)
```

```
<matplotlib.image.AxesImage at 0x7f99dfa29fd0>
```


```
fresh_image=cv2.imread('noidea.jpg') # it's either start with a fresh read of the image,
                                     # or end up with dogfaces on dogfaces on dogfaces
                                      # as you re-run parts of the notebook but not others...

fresh_image[200:200+dogface.shape[0], 200:200+dogface.shape[1]]=dogface
print(dogface.shape[0])
print(dogface.shape[1])
plt.imshow(fresh_image)
```

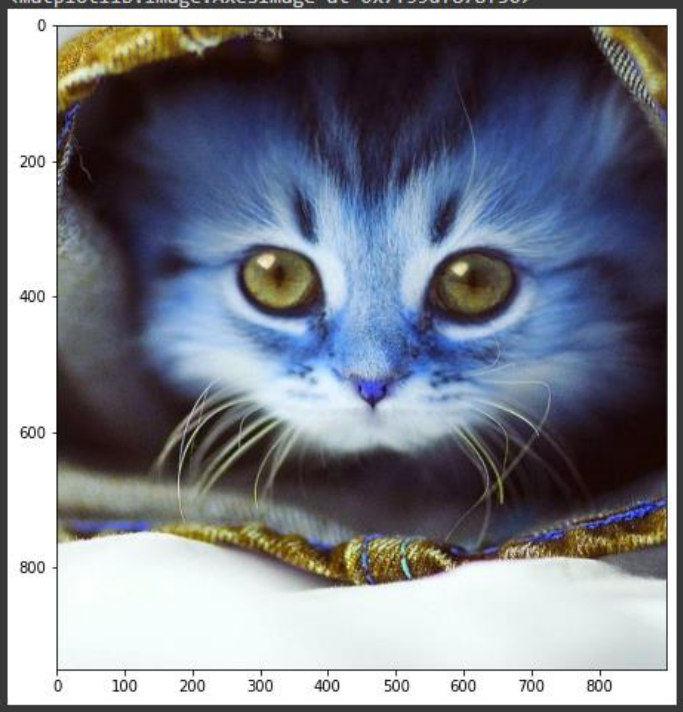
```
650
900
<matplotlib.image.AxesImage at 0x7f99df9a2c10>
```



Abdul Moiz -19022

```
freshim2 = cv2.imread("noidea.jpg")
crop = freshim2[250:1200, 400:1300]
plt.imshow(crop)
```

```
<matplotlib.image.AxesImage at 0x7f99df878f50>
```




```
hsvim=cv2.cvtColor(freshim2,cv2.COLOR_BGR2HSV)
bcrop =hsvim[250:1000, 400:1300, 1]
plt.imshow(bcrop, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f99df86f750>
```

Abdul Moiz -19022

# Lab-2

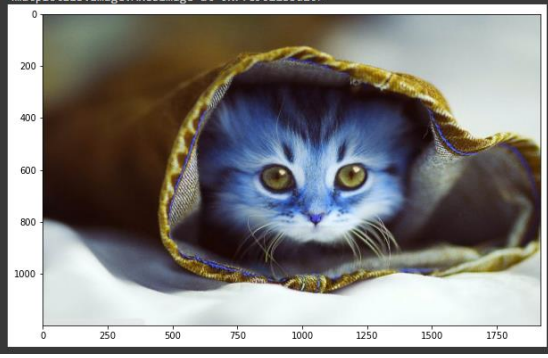
**1. Implement a simple python program with an image of your choice and demonstrate all the concepts (attach screenshots).**

```
flipped_code_0=cv2.flip(input_image,0) # vertical flip
plt.imshow(flipped_code_0)
```

```
<matplotlib.image.AxesImage at 0x7fe9622c7790>
```

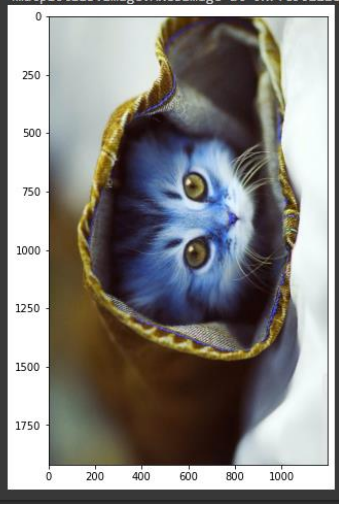
```
flipped_code_1=cv2.flip(input_image,1) # horizontal flip
plt.imshow(flipped_code_1)
```

```
<matplotlib.image.AxesImage at 0x7fe9622b3d10>
```



```
transposed=cv2.transpose(input_image)
plt.imshow(transposed)
```

```
<matplotlib.image.AxesImage at 0x7fe96222e110>
```



```
for i in range(0,3):
    min_value, max_value, min_location, max_location=cv2.minMaxLoc(input_image[:,:,i])
    print("min {} is at {}, and max {} is at {}".format(min_value, min_location, max_value, max_location))
```

```
min 0.0 is at (411, 395), and max 255.0 is at (441, 273)
min 0.0 is at (659, 271), and max 255.0 is at (431, 322)
min 0.0 is at (973, 138), and max 255.0 is at (1268, 346)
```

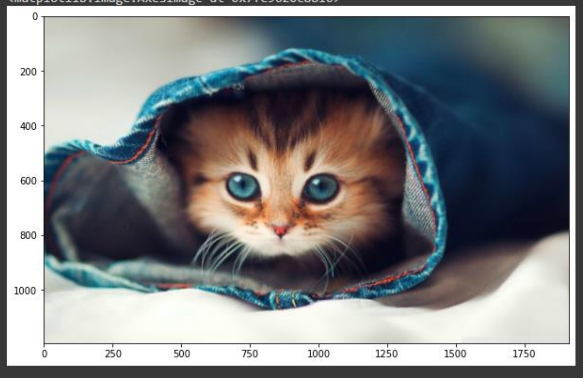
Abdul Moiz -19022

```
#First create an image the same size as our input
blank_image = np.zeros((input_image.shape), np.uint8)

blank_image[590:700,900:1100,1]=100; #give it a green square

new_image=cv2.add(blank_image,input_image) # add the two images together

plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7fe96233da90>
```

```
d=3
img_blur3 = cv2.GaussianBlur(input_image, (2*d+1, 2*d+1), -1)[d:-d,d:-d]

plt.imshow(cv2.cvtColor(img_blur3, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7fe9620c8810>
```

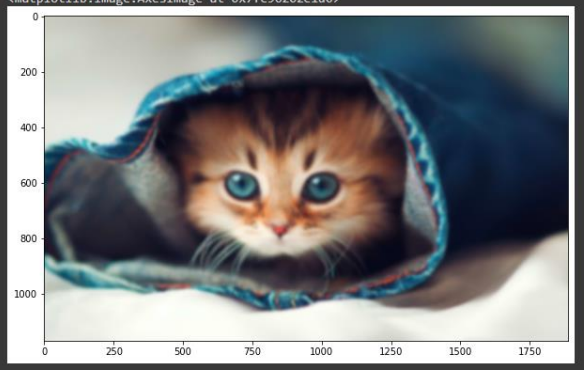


```
[36] d=5
     img_blur5 = cv2.GaussianBlur(input_image, (2*d+1, 2*d+1), -1)[d:-d,d:-d]

     plt.imshow(cv2.cvtColor(img_blur5, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7fe9620b5cd0>
```

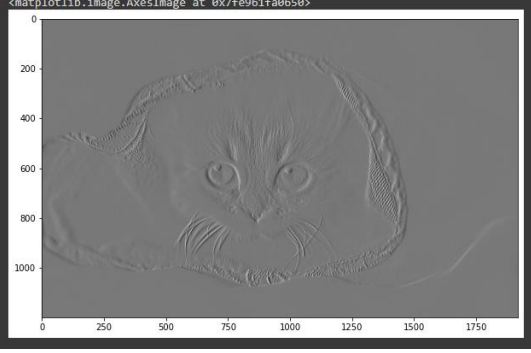
```
d=15
img_blur15 = cv2.GaussianBlur(input_image, (2*d+1, 2*d+1), -1)[d:-d,d:-d]

plt.imshow(cv2.cvtColor(img_blur15, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7fe96202e1d0>
```



```
sobelimage=cv2.cvtColor(input_image,cv2.COLOR_BGR2GRAY)

sobelx = cv2.Sobel(sobelimage,cv2.CV_64F,1,0,ksize=9)
sobely = cv2.Sobel(sobelimage,cv2.CV_64F,0,1,ksize=9)
plt.imshow(sobelx,cmap = 'gray')
# Sobel works in x and in y, change sobelx to sobely in the olt line above to see the d
```

```
<matplotlib.image.AxesImage at 0x7fe961fa0650>
```



```
th1=30
th2=60 # Canny recommends threshold 2 is 3 times threshold 1 - you could try experimenting with t
d=3 # gaussian blur

edgeresult=input_image.copy()
edgeresult = cv2.GaussianBlur(edgeresult, (2*d+1, 2*d+1), -1)[d:-d,d:-d]

gray = cv2.cvtColor(edgeresult, cv2.COLOR_BGR2GRAY)

edge = cv2.Canny(gray, th1, th2)

edgeresult[edge != 0] = (0, 255, 0) # this takes pixels in edgeresult where edge non-zero colours

plt.imshow(cv2.cvtColor(edgeresult, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7fe961f11610>
```

Abdul Moiz -19022

# Lab-3

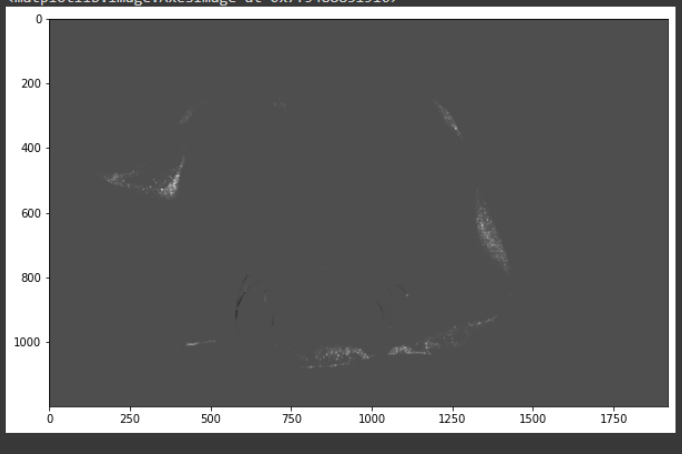
**1. Implement a simple python program with an image of your choice and demonstrate all the concepts (attach screenshots).**

```
harris_test=input_image.copy()
#greyscale it
gray = cv2.cvtColor(harris_test,cv2.COLOR_BGR2GRAY)

gray = np.float32(gray)
blocksize=4 #
kernel_size=3 # sobel kernel: must be odd and fairly small

# run the harris corner detector
dst = cv2.cornerHarris(gray,blocksize,kernel_size,0.05) # parameters are blocksize, Sobel parame

#result is dilated for marking the corners, this is visualisation related and just makes them bi
dst = cv2.dilate(dst,None)
#we then plot these on the input image for visualisation purposes, using bright red
harris_test[dst>0.01*dst.max()]=[0,0,255]
plt.imshow(cv2.cvtColor(harris_test, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f94888692d0>
```


```
plt.imshow(dst,cmap = 'gray')
```

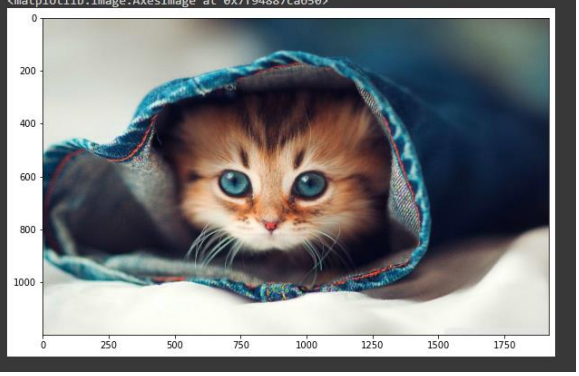
```
<matplotlib.image.AxesImage at 0x7f9488851910>
```



Abdul Moiz -19022

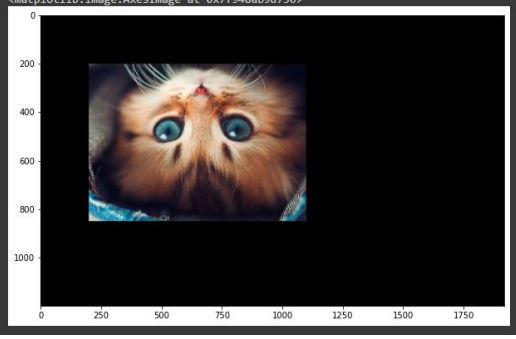
```
orbimg=input_image.copy()

orb = cv2.ORB_create()
# find the keypoints with ORB
kp = orb.detect(orbimg,None)
# compute the descriptors with ORB
kp, des = orb.compute(orbimg, kp)
# draw keypoints
cv2.drawKeypoints(orbimg,kp,orbimg)

plt.imshow(cv2.cvtColor(orbimg, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f94887ca650>
```



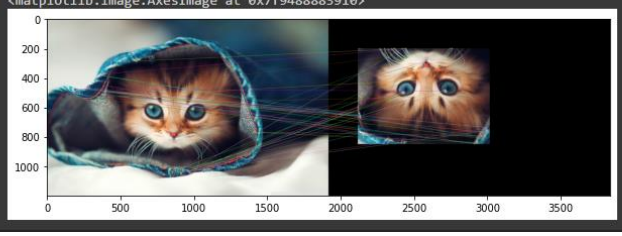

```
img2match=np.zeros(input_image.shape,np.uint8)
dogface=input_image[250:900, 400:1300] # copy out a bit
img2match[60:250,70:350]=[0,0,0] # blank that region
dogface=cv2.flip(dogface,0) #flip the copy
img2match[200:200+dogface.shape[0], 200:200+dogface.shape[1]]=dogface # paste it back so
plt.imshow(cv2.cvtColor(img2match, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f948ab9d750>
```




```
kp2 = orb.detect(img2match,None)
# compute the descriptors with ORB
kp2, des2 = orb.compute(img2match, kp2)
# create BFMatcher object: this is a Brute Force matching object
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
# Match descriptors.
matches = bf.match(des,des2)

# Sort them by distance between matches in feature space - so the best matches are first.
matches = sorted(matches, key = lambda x:x.distance)

# Draw first 50 matches.
oimg = cv2.drawMatches(orbimg,kp,img2match,kp2,matches[:50], orbimg)

plt.imshow(cv2.cvtColor(oimg, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f9488883910>
```



Abdul Moiz -19022

# Lab-4

**1. Implement a simple python program with an image of your choice and demonstrate all the concepts (attach screenshots).**

```
# Load the test image and create a greyscale copy of it to be used in the classifiers

base_image = cv2.imread('test.jpg')
grey = cv2.cvtColor(base_image, cv2.COLOR_BGR2GRAY)
plt.imshow(cv2.cvtColor(base_image, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f378fba7f90>
```

```
# this is a pre-trained face cascade
test_image = cv2.imread('test.jpg')
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
faces = face_cascade.detectMultiScale(grey, 1.3, 5)
for (x,y,w,h) in faces:
    cv2.rectangle(test_image,(x,y),(x+w,y+h),(255,0,0),2)
plt.imshow(cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f378fb996d0>
```

```
# this is a pre-trained face cascade
test_image = cv2.imread('test.jpg')
smile_cascade = cv2.CascadeClassifier('haarcascade_smile.xml')
smiles = smile_cascade.detectMultiScale(grey, 1.3, 20)
for (x,y,w,h) in smiles:
    cv2.rectangle(test_image,(x,y),(x+w,y+h),(0,255,0),2)
plt.imshow(cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f378fb07510>
```

```
# this is a pre-trained face cascade
test_image = cv2.imread('test.jpg')
for (x,y,w,h) in faces:
  for (x_s,y_s,w_s,h_s) in smiles:
    if( (x <= x_s) and (y <= y_s) and ( x+w >= x_s+w_s) and ( y+h >= y_s+h_s)):
      cv2.rectangle(test_image, (x_s,y_s),(x_s+w_s,y_s+h_s),(0,255,0),2)
plt.imshow(cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f378fa75650>
```

Abdul Moiz -19022

```
test_image = cv2.imread('test.jpg')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
eyes = eye_cascade.detectMultiScale(grey, 1.3, 1)
for (x,y,w,h) in eyes:
     cv2.rectangle(test_image,(x,y),(x+w,y+h),(255,255,255),2)
plt.imshow(cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f378fa54b10>
```


```
test_image = cv2.imread('test.jpg')
for (x,y,w,h) in faces:
  #cv2.rectangle(smile_faces_base_image,(x,y),(x+w,y+h),(255,0,0),2)
  for (x_s,y_s,w_s,h_s) in eyes:
    if( (x <= x_s) and (y <= y_s) and ( x+w >= x_s+w_s) and ( y+h >= y_s+h_s)):
      cv2.rectangle(test_image, (x_s,y_s),(x_s+w_s,y_s+h_s),(255,255,255),2)
plt.imshow(cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f378f9d6150>
```


```
test_image = cv2.imread('test.jpg')
for (x,y,w,h) in faces:
  cv2.rectangle(test_image,(x,y),(x+w,y+h),(255,0,0),2)
  for (x_s,y_s,w_s,h_s) in eyes:
    if( (x <= x_s) and (y <= y_s) and ( x+w >= x_s+w_s) and ( y+h >= y_s+h_s)):
      cv2.rectangle(test_image, (x_s,y_s),(x_s+w_s,y_s+h_s),(255,255,255),2)
  for (x_s,y_s,w_s,h_s) in smiles:
    if( (x <= x_s) and (y <= y_s) and ( x+w >= x_s+w_s) and ( y+h >= y_s+h_s)):
      cv2.rectangle(test_image, (x_s,y_s),(x_s+w_s,y_s+h_s),(0,255,0),2)
plt.imshow(cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f378f947190>
```