
COGS 260: Assignment 1

Kriti Aggarwal (PID: A53214465)
Department of Computer Science and Engineering
University of California, San Diego
San Diego, CA 92093
kriti@eng.ucsd.edu

Abstract

This assignment serves as an introduction to the field of image recognition. The tasks begin with a basic read and write operations on a image and move towards more sophisticated tasks. After the image is read we look into various image filtering techniques, namely mean, gaussian and median filters. After we have a de-noised image, we work towards enhancing the image, by using and comparing techniques of histogram equalization and adaptive histogram equalization. The final task of this assignment paper discusses on three famous techniques of edge detection, sobel, canny and the most recent structured forests.



Figure 1: Sample color image



Figure 2: Sample grayscale image

Description of the methods and Experimental Results

Basic Image Operations

We used open cv in python to run all the experiments for this assignment.

- Image read and write
Read any image from the dataset in the resource folder (data/img) and write it in RGB and Grayscale.



Figure 3: Images after applying blur filters of various sizes

We first loaded the image using open cv. For writing the image as RGB, we first split the image in BGR color channels and then merged them. We noticed that the channels in the image are obtained as BGR by opencv. Hence, we converted its channels from BGR to RGB so that the images could be plotted by the matplotlib library. `cv2.cvtColor(img, cv2.COLOR_BGR2RGB)`

For converting the truecolor image RGB to the grayscale intensity image, hue and saturation are eliminated while the luminance is retained. Here are the sample color and grayscale images. The loaded grayscale and colored images are shown in the figure 2 and 1

- Image Smoothing

For this experiment, we took average and Gaussian filters of various sizes and performed smoothing over the images. For average smoothing, we used filters of size 3,5,7 and 9. We found that as the size of the filters increased, the blurriness in the image was enhanced. This follows our intuition since average blur filter is implemented by convolving the image matrix with a kernel. For every pixel, the value of the pixel in the kernel size window is



Figure 4: Results of Gaussian smoothing of various filter size

averaged. As the size of the filter is increased, the blurriness of the image increases as the averaging is done over a bigger window. The results of the blur filter is shown in figure 3.

While in Gaussian smoothing, the value of sigma governs the degree of smoothing, and eventually how the edges are preserved. We experimented with various values of filters: 3,5,7 and 9. We found that the blurriness in the image increases as filter size increases as is the case of average smoothing. The results of the gaussian filter is shown in figure 4

We also experimented with the variance in Gaussian filter. We used the filter size 5x5 and variance parameter as 0 and 10. We found that as the variance is increased, the blurriness of the image increases. This is because as the sigma becomes larger, more variance is allowed around the mean and as the sigma becomes smaller, the less variance is allowed around the mean. That is if the gaussian filter has a small sigma it has the steepest peak, so more weights will be focused in the center and the less around it. The results of this experiment are shown in figure 5.

One major observation in this experiment was the difference in the level of smoothing for the same filter size. We observed that the Gaussian smoothing preserves the image edges

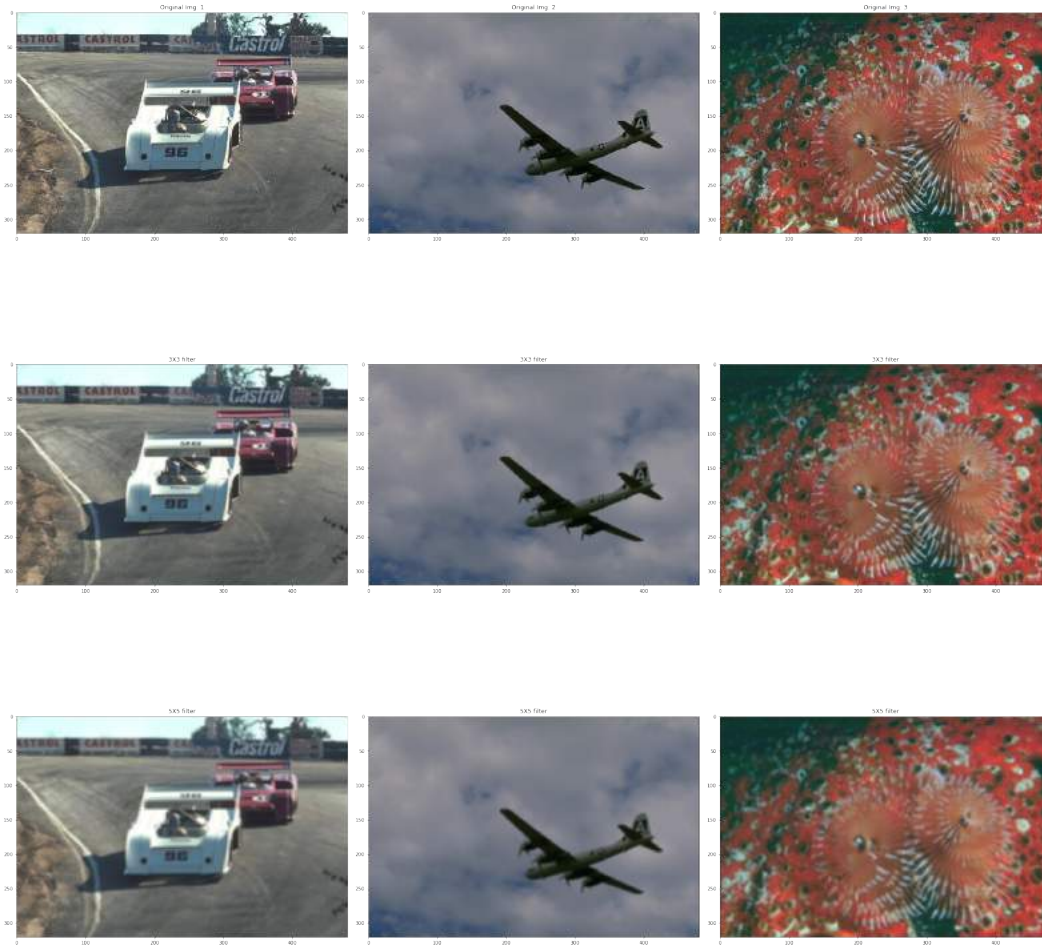


Figure 5: Images after changing the variance in Gaussian filters from 0 to 10 with 5x5 kernel

more than average smoothing. This can be attributed to the fact that Gaussian smoothing is a weighted average as opposed to a simple average as in the case of the average smoothing. In Gaussian smoothing, middle pixel in a window gets the highest weight to vote for the output pixel. Hence, Gaussian smoothing produces better results than average smoothing for the same filter size.

- Image denoising

In this experiment, we applied median filter of sizes 3,5,7,9 and 11. We found that denoising the image has a tradeoff. Since, as the denoising of the image is done at the expense of degrading the quality of the image. We found that different size of filters gave good results in different images. In the first and fourth image, the noise was the maximum and of bigger size than the rest of the images. Therefore, the filter size required to remove the noise to a sufficient level in these images was larger(11x11) than the other images. But as the size of the filter is increased, the quality of the image gets degraded. For image 2 and 3 a smaller filter size gave good results for this image (3x3). For the last image, a filter size of 5x5 gave the best results. By this experiment, we conclude that the size of the filter that works best depends on how heavily noised the image is. The results of median filtering are shown in figure 6.

We performed another experiment with median filter by using a small filter size multiple times and comparing it with a filter of a larger size. In this experiment, we found that performing median filtering multiple times with a small filter preserves the image quality better than with a large size filter. Though in the case of image 1 and 4, a lot of noise still

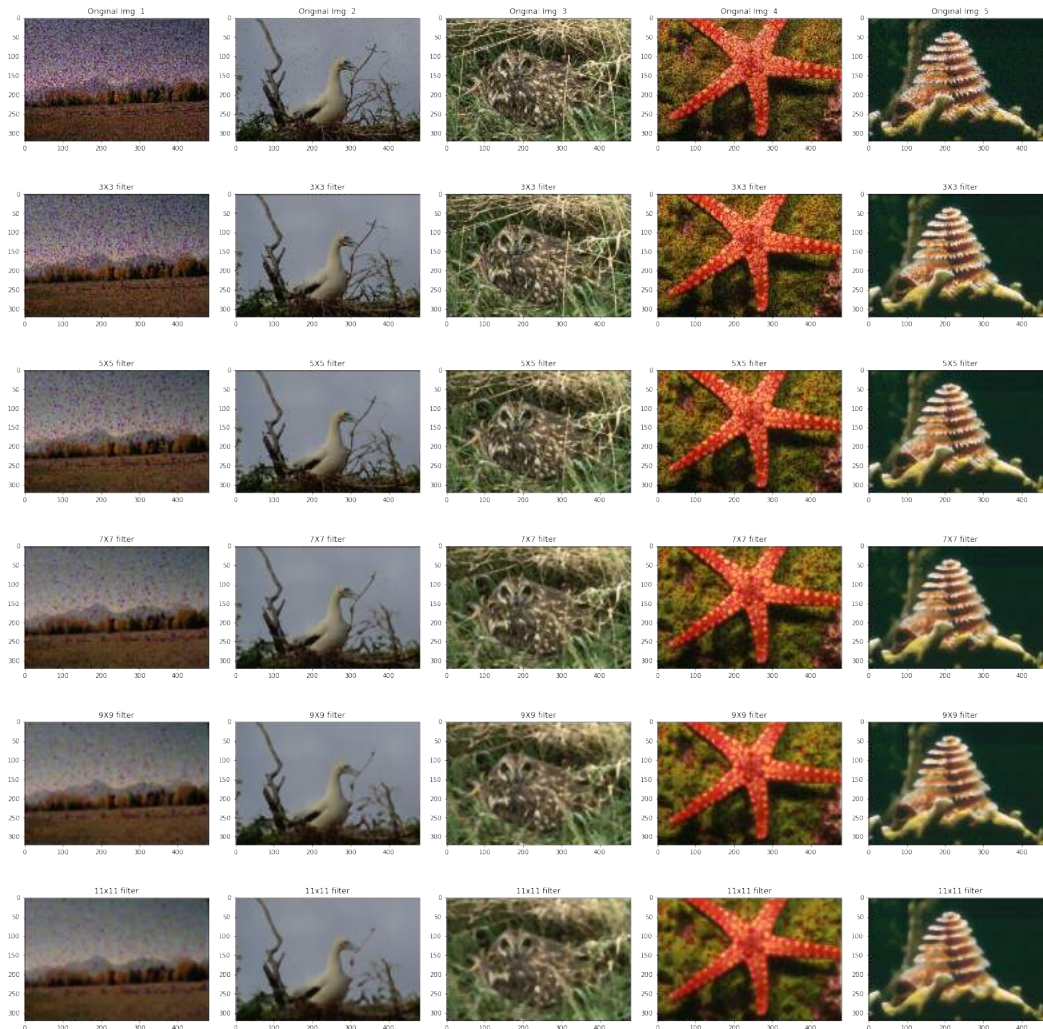


Figure 6: Results after applying median filter in various sizes

remains even after applying the 3x3 filter 3 times. The corresponding results are shown in figures 7 and 8.

However, in general the median filter of size 7x7 worked best for all the images.

Image Enhancement

- Histogram Histogram is similar to a graph or plot, which gives us an overall idea about the intensity distribution of an image. It is a plot with pixel values (ranging from 0 to 255, not always) in X-axis and corresponding number of pixels in the image on Y-axis. It helps in building an understanding of the images by giving us the intuition about the intensity, brightness and saturation.

The color histogram shows the proportion of the various colors in the image. In the first image, the proportion of all the three colors is almost equal, while in the second image blue is higher in some places and red is higher in some other areas of the image. In the third image there is a very high portion of the image that is red as is shown by the histogram. Also, the areas containing high proportions of red and blue are segregated as visualized in the histogram as shown in figure 9.

We changed the number of bins and got the results as shown in figures 10 and 11.

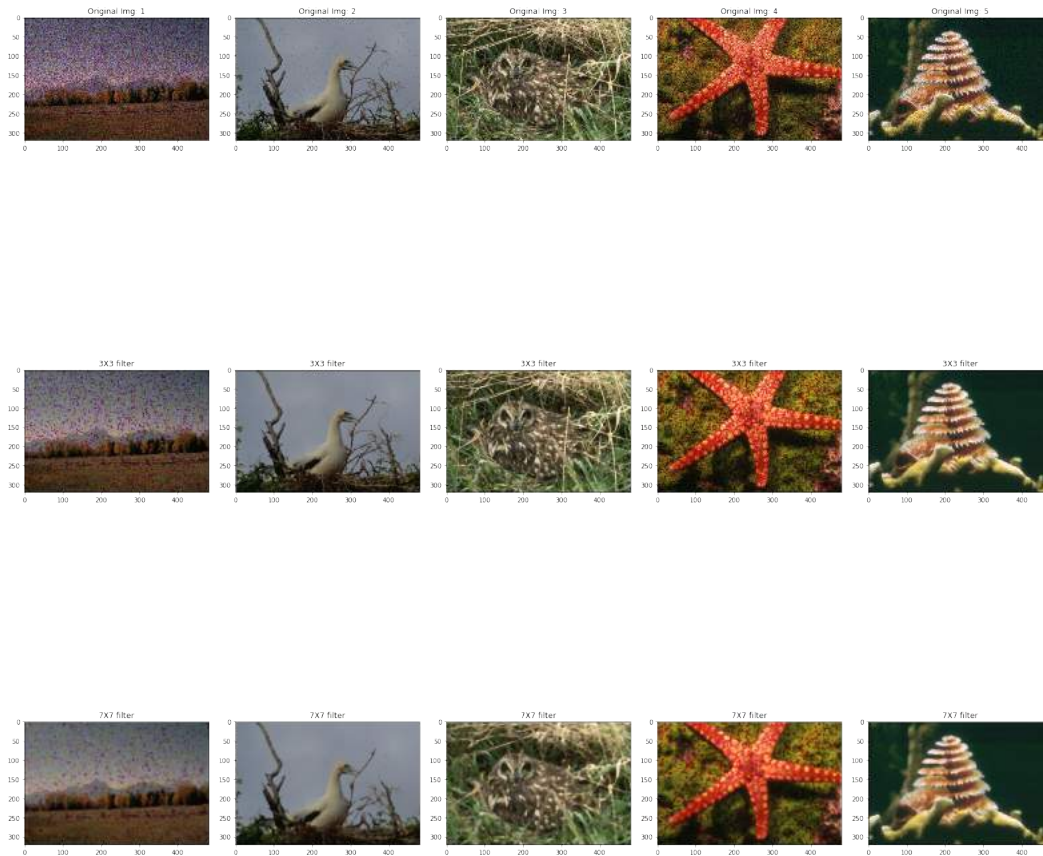


Figure 7: Results after applying median filter 3x3 2 times vs filter 7x7

- **Global Histogram Equalization** If an image has pixels confined to a specific range of values, stretching such intensity histogram would provide a better quality image. Histogram equalization is one such technique of tuning the contrast in the image. By looking at the histogram, we can see the tonal(intensity) variation in the image. By performing equalization, the intensity of the pixels is equalized and the image has similar contrast in the complete image.

For this experiment, we extracted the intensity value of the image and spread them in the equalized the intensity values through out the image. As can be seen in the figure 12, the most frequent intensity values are effectively spread through out the image after the image equalization is done. In all the images the low and high intensity values are more pronounced after equalization as the image is brighter overall. But in the second image, the aeroplane is darkened, so is the sky which has resulted in a overall dark image than the original image. This shows that this method performs worse when only a part of the image is dark/bright. Since, in this image only certain parts of the image is dark but the algorithm darkens the complete image.

- **Adaptive Histogram Equalization (CLAHE)**

Adaptive method computes several histograms, each corresponding to a distinct section of the image, and uses them to redistribute the lightness values of the image. Hence, it is suitable for improving the local contrast and enhancing the edges in each region of an image. The results for adaptive equalization is shown in figure 13.

The results obtained were far better than global histogram equalization. This is because this method takes into account the local lightness and intensity as opposed to the complete image.

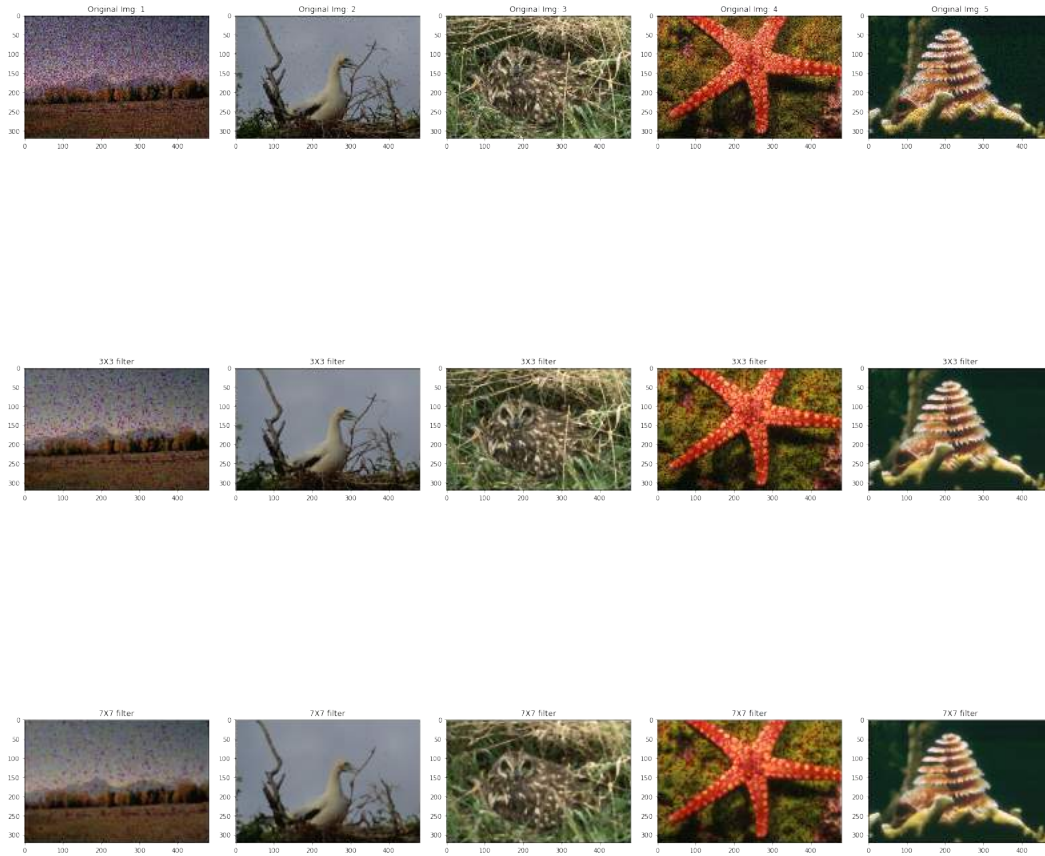


Figure 8: Results after applying median filter 3x3 3 times vs filter 7x7

We performed several experiments with this method. In this experiment, we played with the various grid sizes of the image for which a local histogram is computed. Here are the results for the various grid sizes as shown in figures ??,15,16,17, and 18. We used multiple of 2 as grid sizes starting from size 4x4 to 64x64. We found that the images got better as the grid size was increased with a peak at grid size 32x32. After that the quality of the images worsened with further increase in grid size.

In another experiment, we experimented with different clip sizes, while keeping the grid size fixed to 8x8: [1,1.5,2.3,3,5] as shown in figures 19,20,21,22 and 23. This clip size determines the thresholding of the intensities. Hence, as the size of the clip sizes increases, pixels with higher intensity are allowed and the image gets more and more brighter. The image obtained is best for clip size 2.3, after which the image gets worsened.

Edge detection

- Sobel filter In this experiment, we implemented the sobel filter and ran it on the test data set. We calculated the gradients in x and y axis and then took second norm of the same. We experimented with various sizes of the sobel operator and found that the filter of size 1x1 works the best. As the size of the filter is increased, the accuracy goes down. This can be attributed to the fact that the gradient direction is better captured in a smaller kernel window.

For filter size 5x5, the accuracy is 0 for mostly all the images. For filter size 3x3, the accuracy of all the images got worse except for the last image.

Also, the accuracy is calculated by thresholding first and then comparing the same pixels in the ground truth and the output image. We think that a better evaluation strategy can be

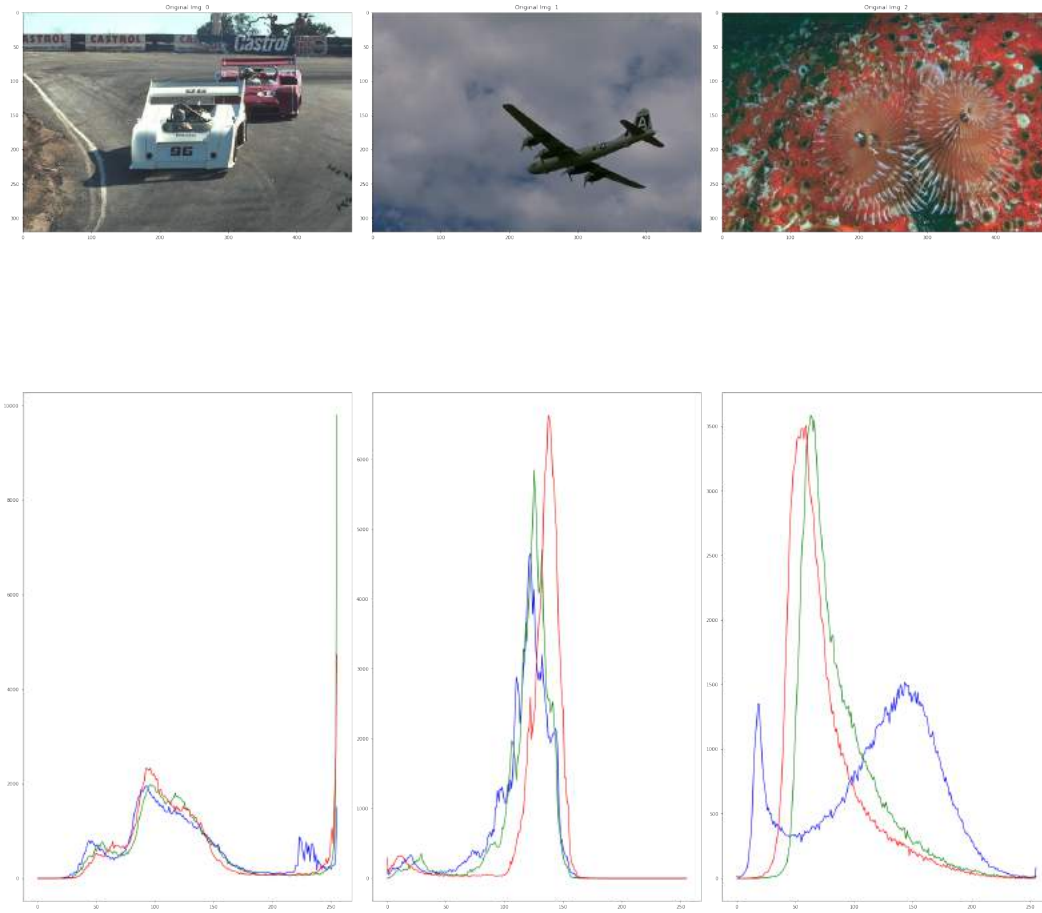


Figure 9: RGB histogram with bins 256

used to capture the accuracy of edge detection methods. The results for the sobel operator are shown in figures 24 and 25.

- Canny operator For canny operator we used the opencv utility for computing the edges using canny operator. We experimented with various values of upper and lower threshold for the canny operator and found that the values 200 for the lower threshold and 225 for the higher threshold gave the best results. The results have been shown in this figure 26.

We experimented with 3 threshold types: median, tight and wide. For median threshold, we calculated a median of all the pixel intensities for an image and calculated the lower threshold as $\text{int}(\max(0, (1.0 - \text{sigma}) * v))$ and the higher threshold $\text{int}(\min(255, (1.0 + \text{sigma}) * v))$. But this threshold resulted in an accuracy of 0 for many images.

For wide and tight thresholds, we experimented with various range of thresholds. As the name suggests, wide threshold is one with with greater separation between the two ends of threshold, and tight thresholds being the opposite case. The observation made was that tight threshold worked better for our data set. The threshold parameter which lead to optimal results was defined in the range [200,225] as shown in figure 27.

- Structured Forests Structured forests is a learning approach to solve this problem of edge detection. This is basically structured learning approach. [3]Structured learning addresses the problem of learning a mapping where the input or output space may be arbitrarily complex representing strings, sequences, graphs, object pose, bounding boxes etc. This approach uses random forests to predict a structured segmentation mask from a larger image patch.

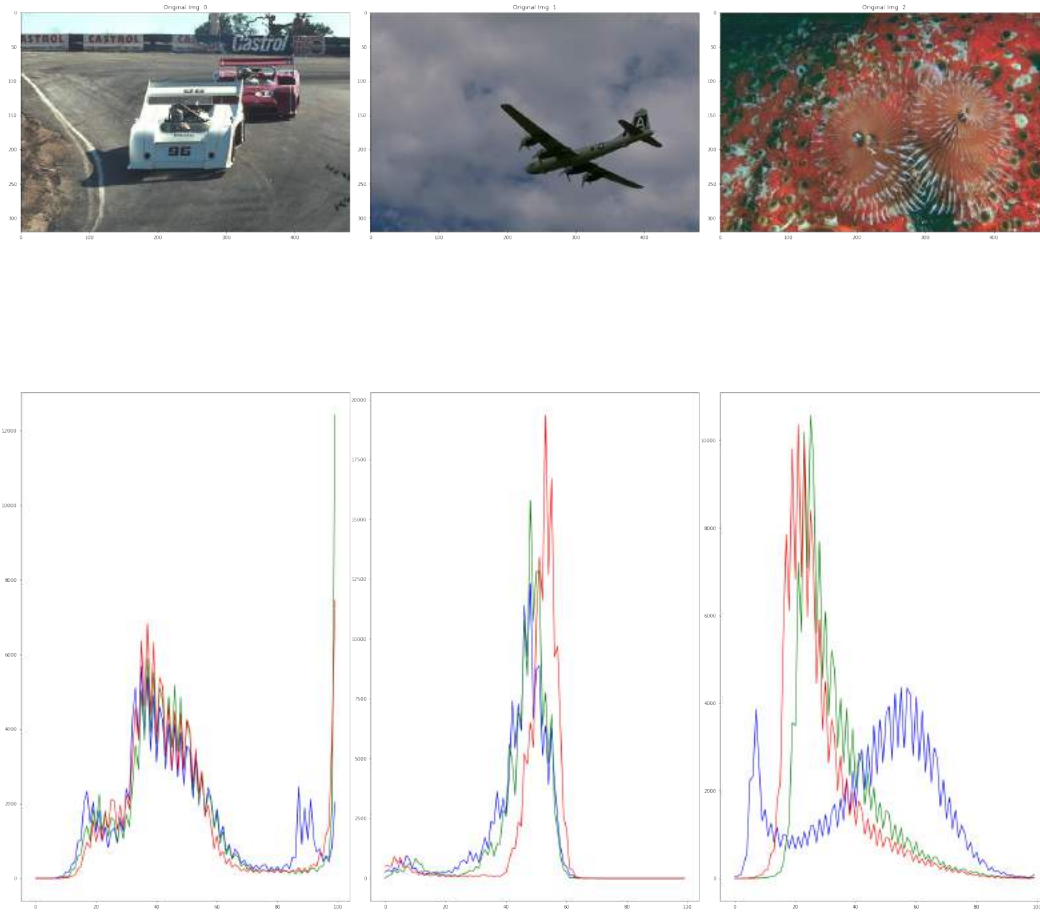


Figure 10: RGB histogram with bins 100

We found a python implementation of this algorithm which is trained on BSDS500 dataset. There was pretrained model with a few pluggable parameters such as *fid*s : *featureindicesforeachnode*, *cd*s : *indicesofchildrenforeachnode*, *end*s : *begin/endofedgepointsforeachnode*, *edge*s : *edgepointsforeachnode*, *nseg* : *numberofsegmentationsforeachnode*, *segs* : *segmentationmapforeachnode* etc. Here are the results obtained by this algorithm as shown in figure 29. Figure 28 shows output for each of the test dataset image. This algorithm was able to detect the edges with much higher accuracy than sobel and canny, especially for cases when the aforementioned algorithms performed very poorly.

We experimented by changing some of these features but there was not much change in the accuracy obtained. Also, the time taken for training the model was close to 11 hours which restricted experimenting extensively with the parameters. Here are the results of running this algorithm for detecting the edges in test dataset.

We also experimented with multi-scale detection, but that too failed to increase the accuracy on the test dataset.

1 References

- [1] docs.opencv.org.
 [2] [https : //en.wikipedia.org/wiki/Histogram_equalization](https://en.wikipedia.org/wiki/Histogram_equalization) [3] [https : //pdollar.github.io/files/papers/DollarPAMI15edges.pdf](https://pdollar.github.io/files/papers/DollarPAMI15edges.pdf)

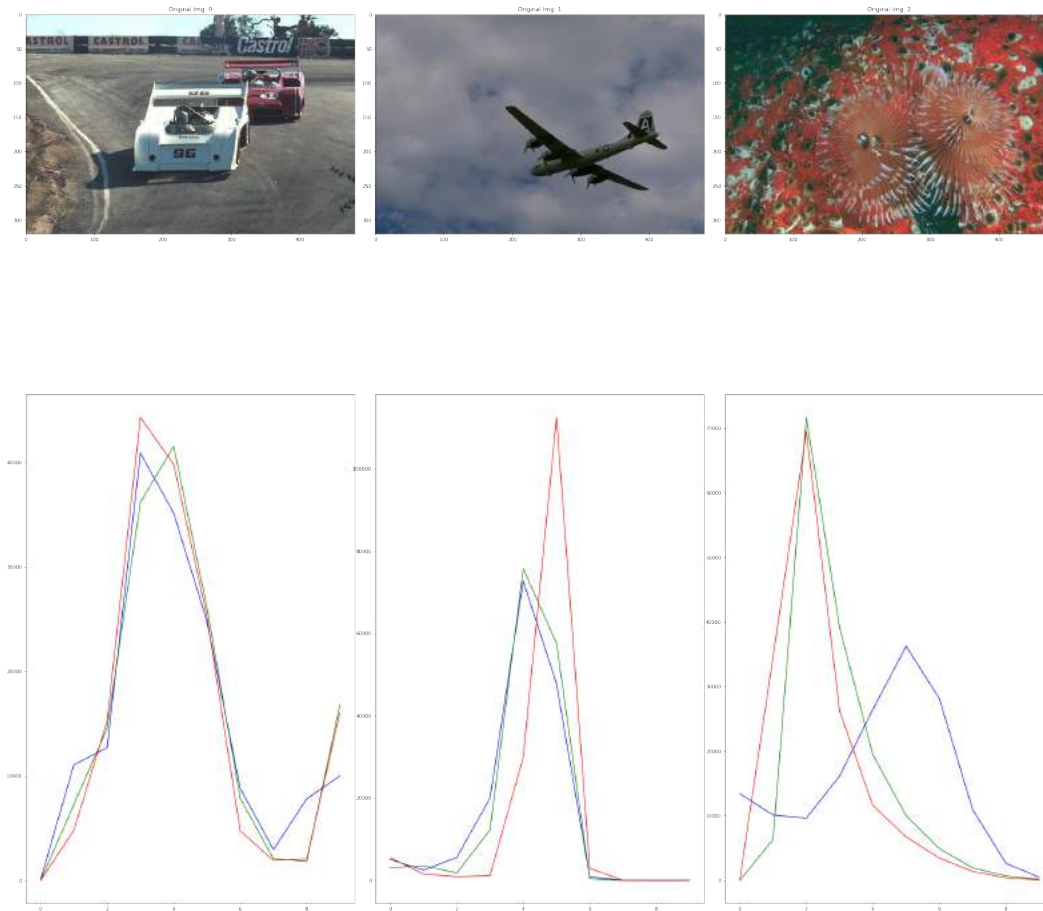


Figure 11: RGB histogram with bins 10

2 Appendix

```
# coding: utf-8
```

```
# In[2]:
```

```
get_ipython().magic('matplotlib inline')
```

```
#1a)Read any image from the dataset in the resource folder (data/img) and write it in
```

```
# In[3]:
```

```
import numpy as np
from matplotlib import pyplot as plt
import cv2
from IPython.display import Image
```

```
# In[4]:
```

```
def imgRead(path):
    img = cv2.imread(path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

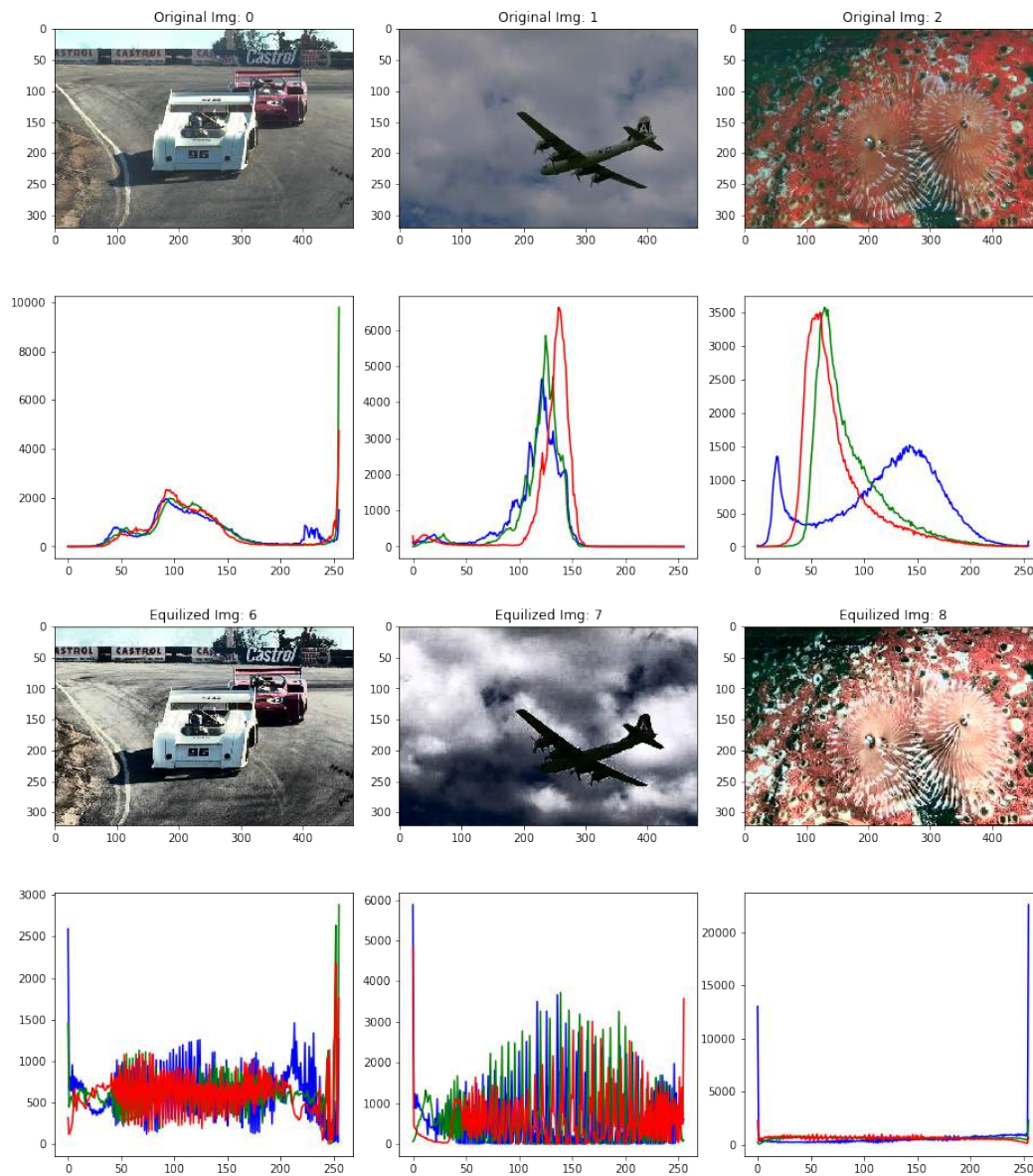


Figure 12: Global Histogram Equalization

```

return img
img1 = imread('../ass/Project1/data/img/21077.jpg')

# In[5]:

# Second arg for the type of the image, 1: color, 0: grayscale, -1: no change
gray_image = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
cv2.imwrite('../ass/Project1/output/car_gray.png',gray_image)

# In[6]:

# Grayscale file

```

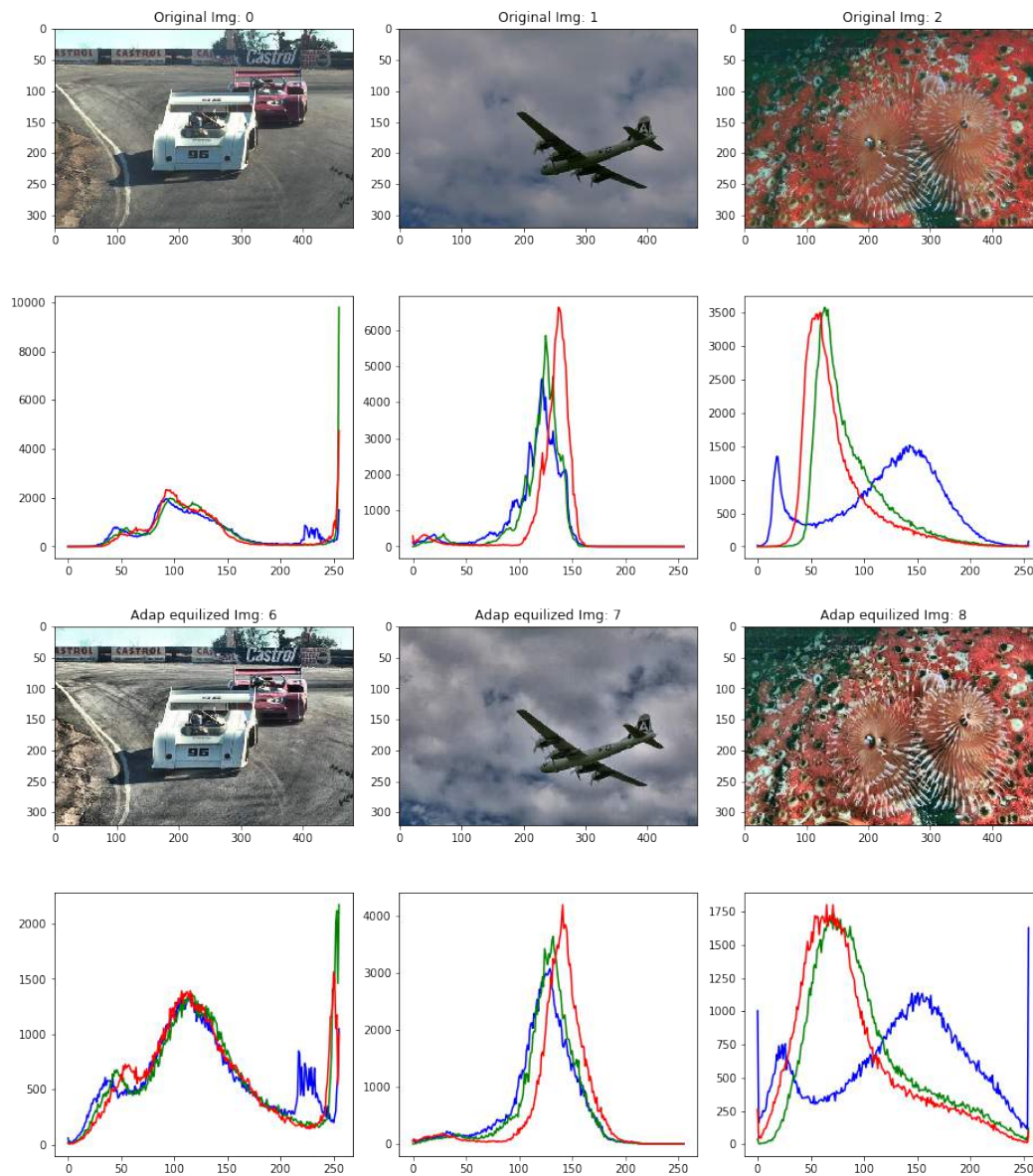


Figure 13: Adaptive Histogram Equalization (CLAHE)

```
Image(filename='../ass/Project1/output/car_gray.png')
```

```
# In[7]:
```

```
# split the b,g,r channels
b,g,r = cv2.split(img1)
img1 = cv2.merge((b,g,r))
```

```
# In[8]:
```

```
#plt.imshow(img)
def writeImg(path,img):
```

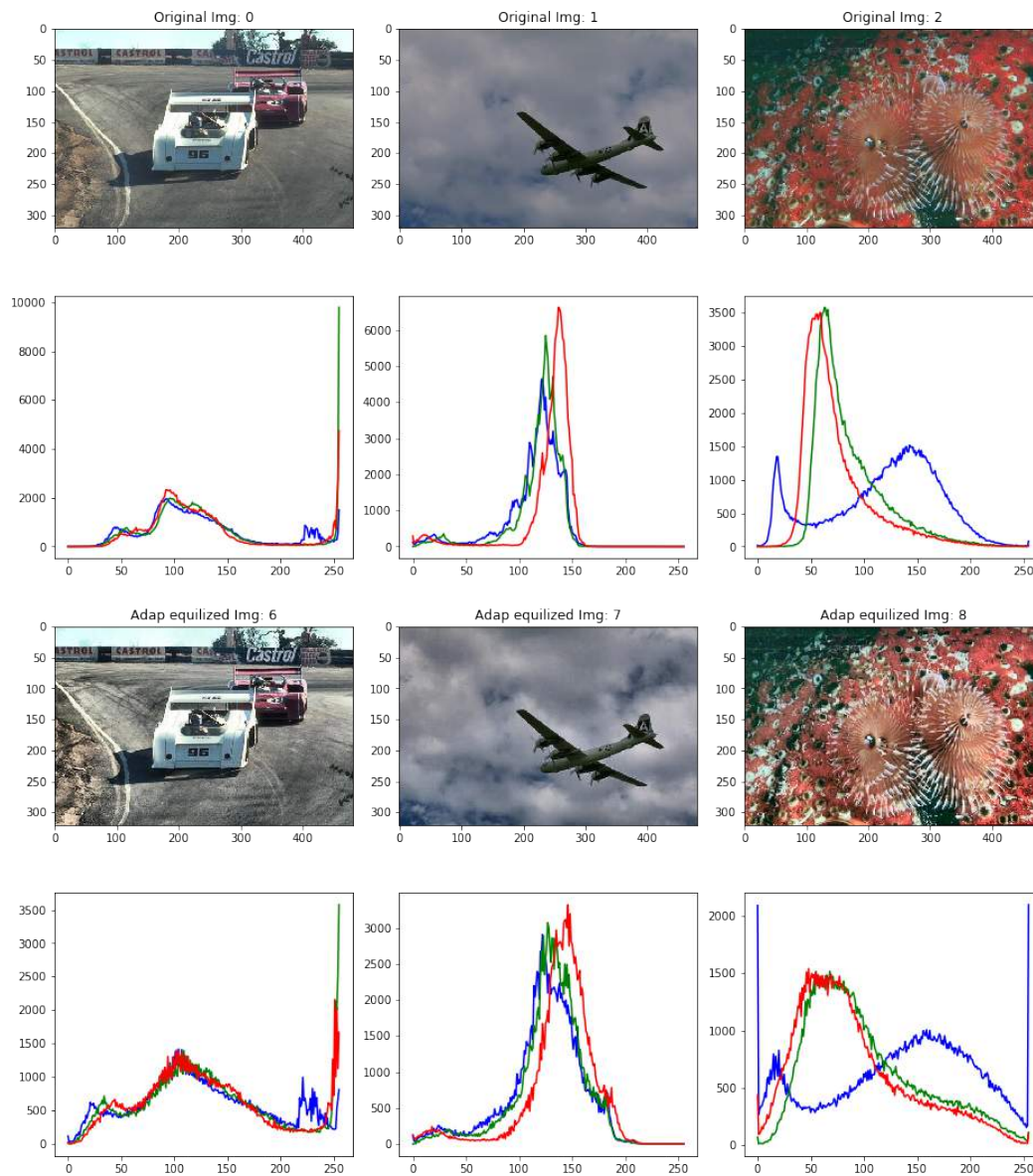



Figure 14: Adaptive Histogram Equalization (CLAHE) with grid size 4x4

```
cv2.imwrite(path,img)

# In[9]:

plt.imshow(img1)

# # Image blurring

# In[10]:

#loading more images
img1 = imgRead('../ass/Project1/data/img/21077.jpg')
```

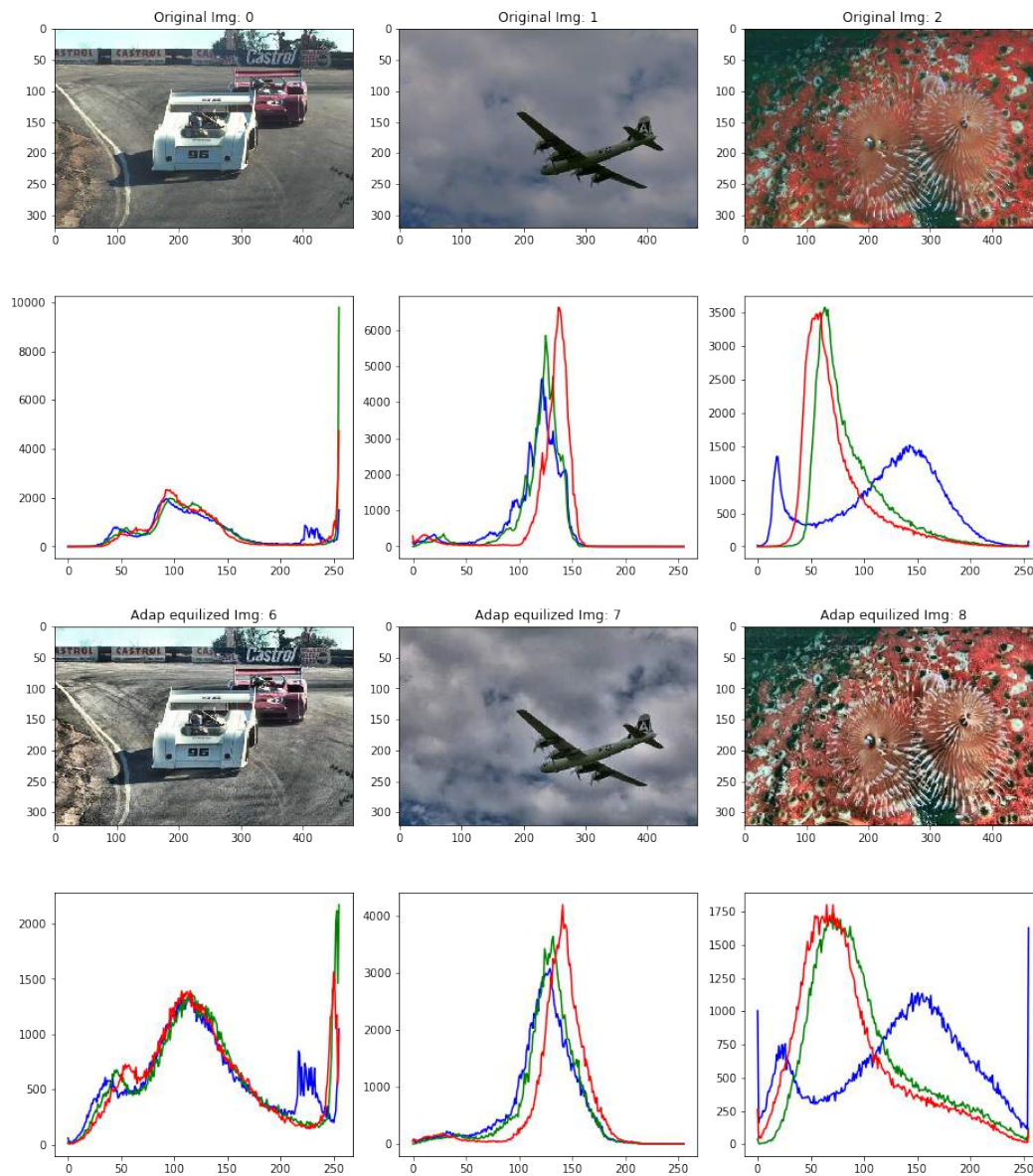


Figure 15: Adaptive Histogram Equalization (CLAHE) with grid size 8x8

```
print img1
img2 = imgRead('../ass/Project1/data/img/3096.jpg')
img3 = imgRead('../ass/Project1/data/img/12084.jpg')

# In[11]:

#blurring filters
fig, plots = plt.subplots(5, 3, figsize=(30,30))
plots = plots.flatten()
fig.tight_layout()

for i in range(0,3):
    plots[i].imshow(eval("img" + str(i+1)))
```

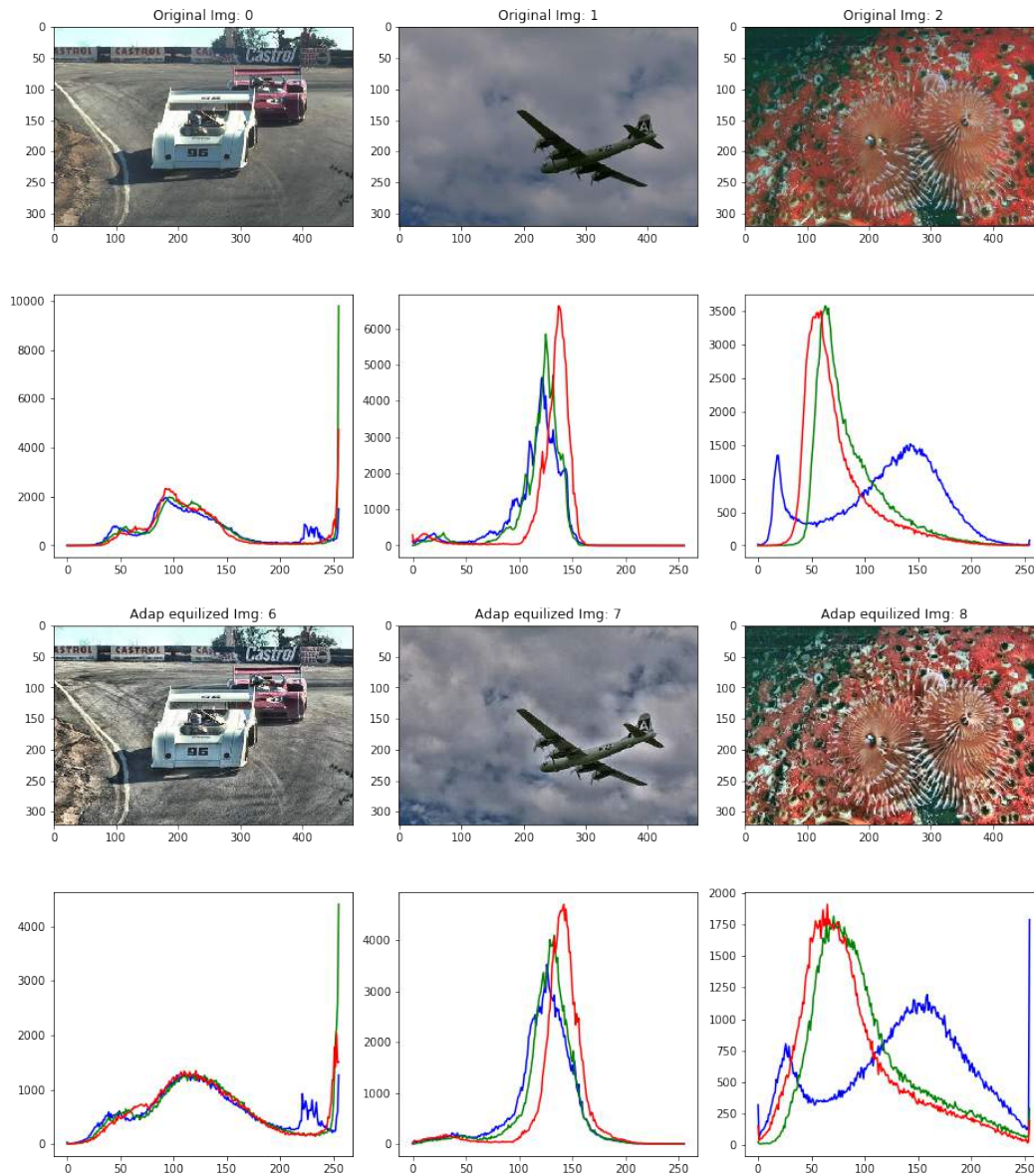


Figure 16: Adaptive Histogram Equalization (CLAHE) with grid size 16x16

```

title = 'Original Img: %s' % (i+1)
plots[i].set_title(title)
for i in range(3,6):
    j = (i+1) - 3
    plots[i].imshow(cv2.blur(eval("img" + str(j)), (3,3)))
    title = '3X3 filter'
    plots[i].set_title(title)
for i in range(6,9):
    j = (i+1) - 6
    plots[i].imshow(cv2.blur(eval("img" + str(j)), (5,5)))
    title = '5X5 filter'
    plots[i].set_title(title)
for i in range(9,12):
    j = (i+1) - 9

```

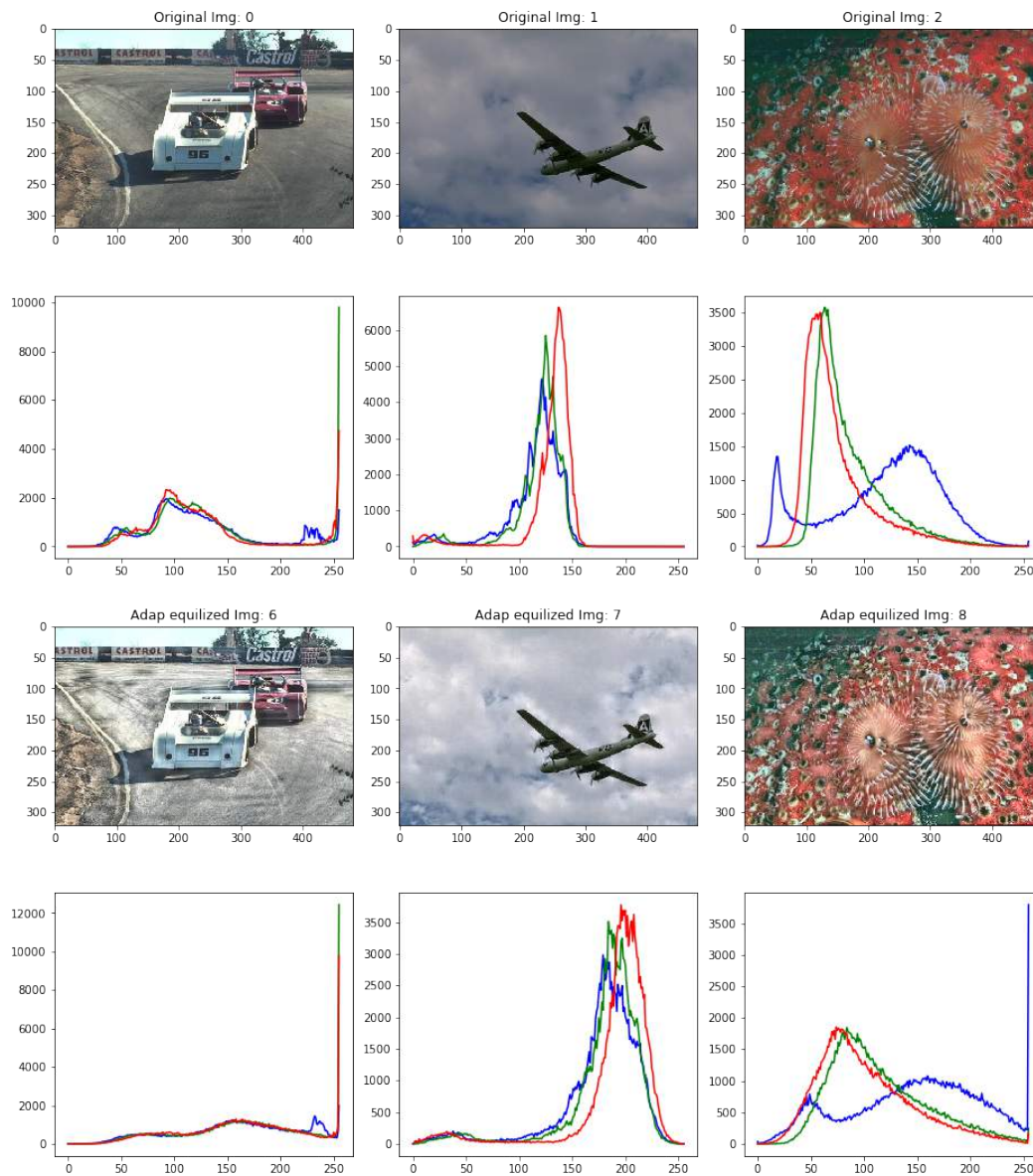


Figure 17: Adaptive Histogram Equalization (CLAHE) with grid size 32x32

```

plots[i].imshow(cv2.blur(eval("img" + str(j)), (7,7)))
title = '7X7 filter'
plots[i].set_title(title)
for i in range(12,15):
    j = (i+1) - 12
    plots[i].imshow(cv2.blur(eval("img" + str(j)), (9,9)))
    title = '9X9 filter'
    plots[i].set_title(title)
# Fine-tune figure; make subplots farther from each other.
fig.subplots_adjust(hspace=0.3)

```

In[25]:

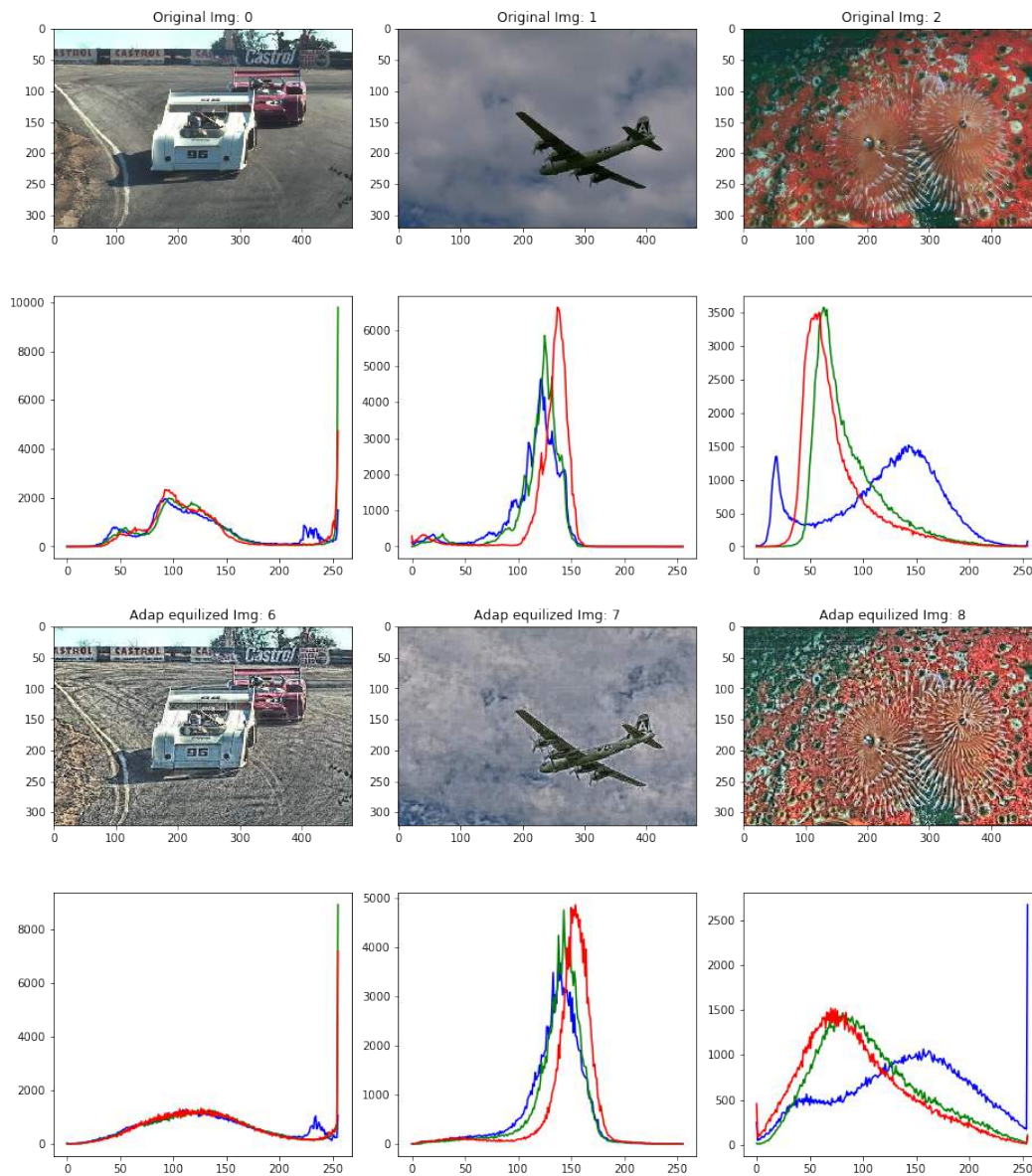


Figure 18: Adaptive Histogram Equalization (CLAHE) with grid size 64x64

```
# Gaussian filter
# blur: filter size 5x5
#blurring filters
fig, plots = plt.subplots(3, 3, figsize=(30,30))
plots = plots.flatten()
fig.tight_layout()

for i in range(0,3):
    plots[i].imshow(eval("img" + str(i+1)))
    title = 'Original Img: %s' % (i+1)
    plots[i].set_title(title)
for i in range(3,6):
    j = (i+1) - 3
    plots[i].imshow(cv2.GaussianBlur(eval("img" + str(j)), (5,5),0))
```

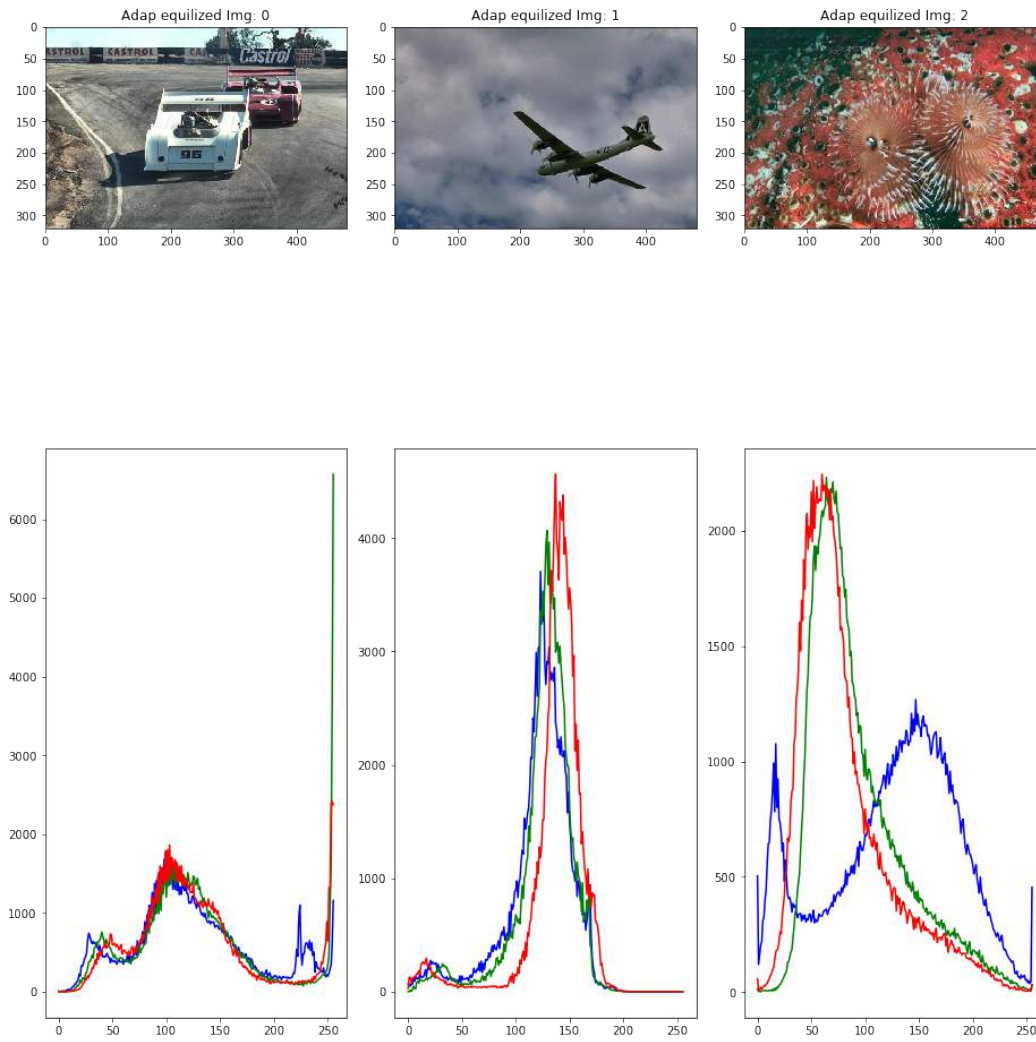


Figure 19: Adaptive Histogram Equalization (CLAHE) with clip size 1

```

title = '3X3 filter'
plots[i].set_title(title)
for i in range(6,9):
    j = (i+1) - 6
    plots[i].imshow(cv2.GaussianBlur(eval("img" + str(j)), (5,5),10))
    title = '5X5 filter'
    plots[i].set_title(title)
# for i in range(9,12):
#     j = (i+1) - 9
#     plots[i].imshow(cv2.GaussianBlur(eval("img" + str(j)), (7,7),0))
#     title = '7X7 filter'
#     plots[i].set_title(title)
# for i in range(12,15):
#     j = (i+1) - 12
#     plots[i].imshow(cv2.GaussianBlur(eval("img" + str(j)), (9,9),0))
#     title = '9X9 filter'
#     plots[i].set_title(title)

```

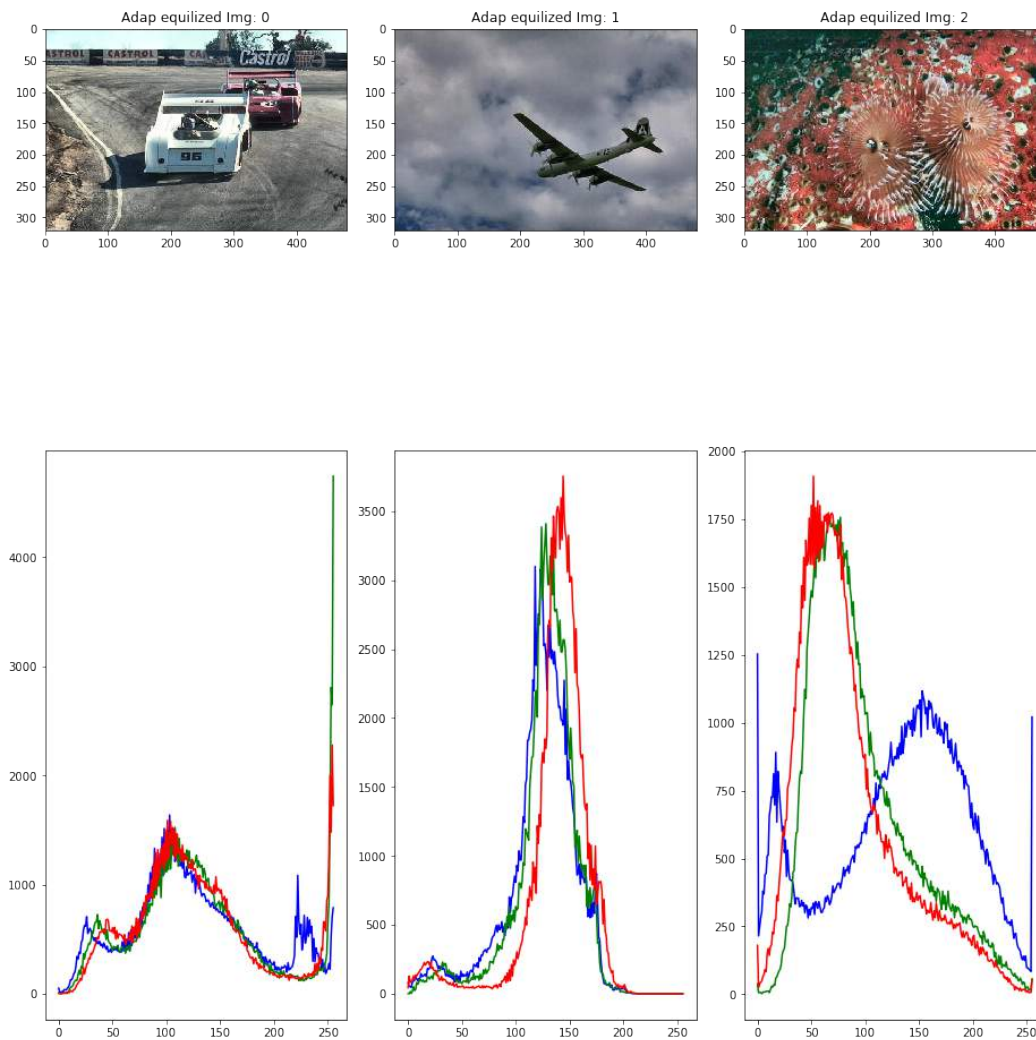


Figure 20: Adaptive Histogram Equalization (CLAHE) with clip size 1.5

```
# # Image denoising

# Here, the function cv2.medianBlur() takes median of all the pixels under kernel area

# In[27]:

# Loading all images in snp folder
import os
rootdir = './...'
images = []
for subdir, dirs, files in os.walk(rootdir):
    for file in files:
        #print os.path.join(subdir, file)
        filepath = subdir + os.sep + file
        if subdir.endswith('snp') and filepath.endswith('.jpg'):
            #loading more images
            img = imread(filepath)
            if img is not None:
                images.append(img)
```

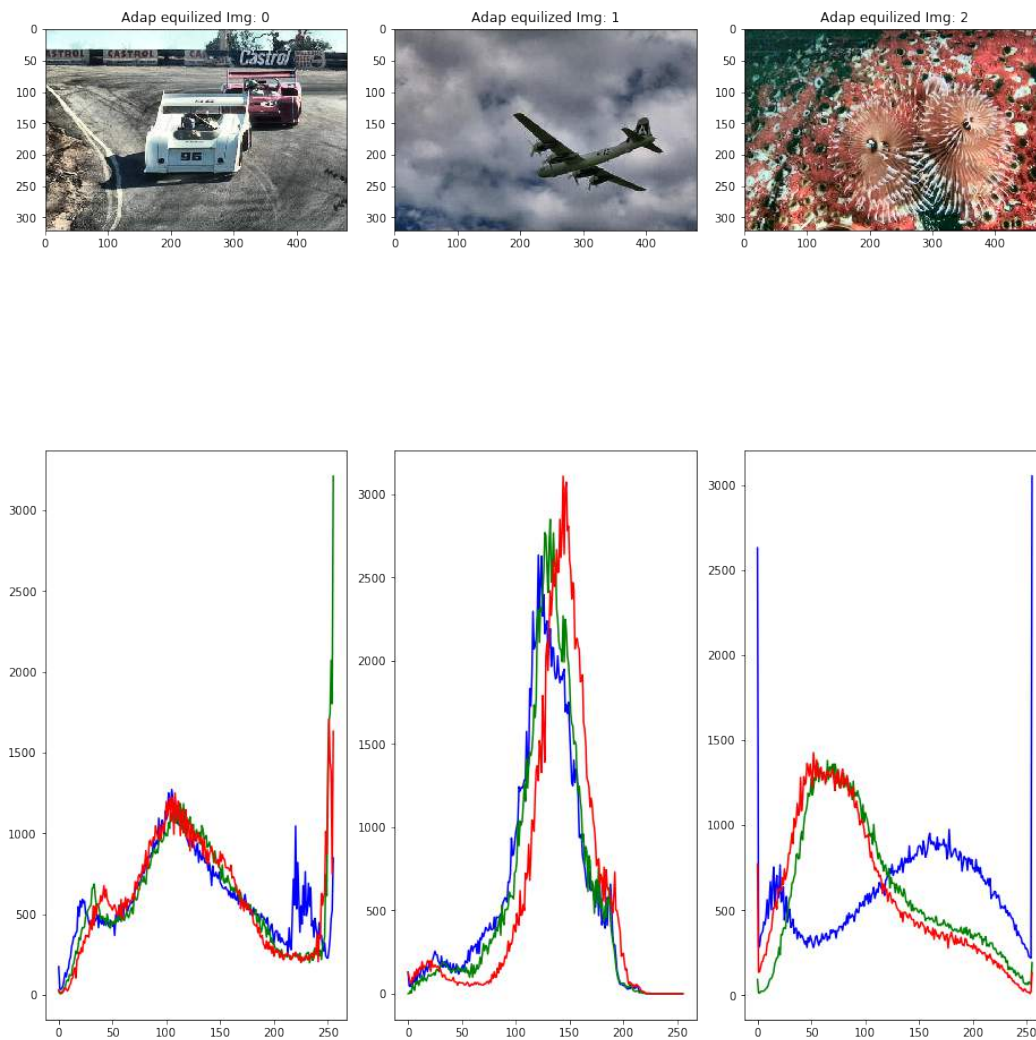


Figure 21: Adaptive Histogram Equalization (CLAHE) with clip size 2.3

```
print (filepath)

# In[17]:

# Denoising filter
fig, plots = plt.subplots(6, 5, figsize=(23,23))
plots = plots.flatten()
fig.tight_layout()

for i in range(0,5):
    plots[i].imshow(images[i])
    title = 'Original Img: %s' % (i+1)
    plots[i].set_title(title)
for i in range(5,10):
    j = (i) - 5
    plots[i].imshow(cv2.medianBlur(images[j],3))
    title = '3X3 filter'
    plots[i].set_title(title)
```

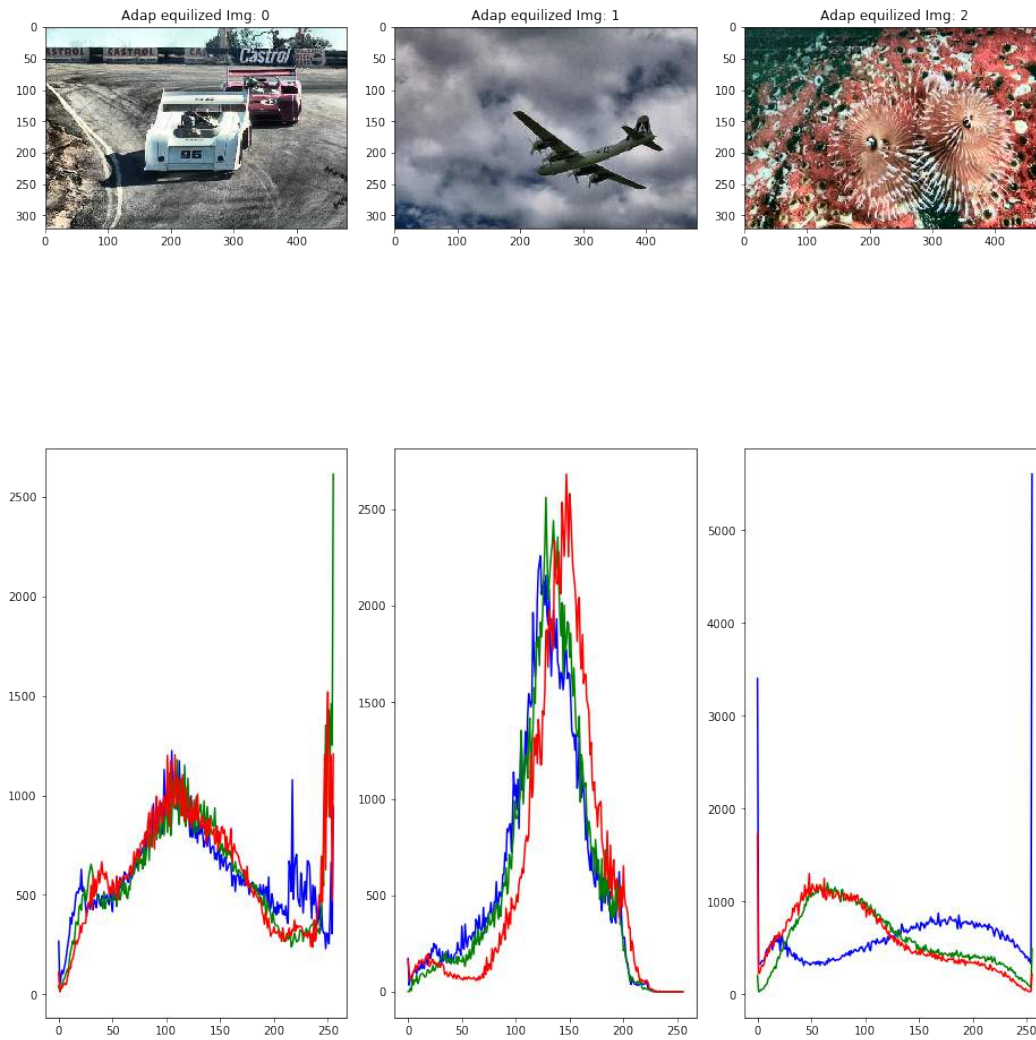



Figure 22: Adaptive Histogram Equalization (CLAHE) with clip size 3

```

for i in range(10,15):
    j = (i) - 10
    plots[i].imshow(cv2.medianBlur(images[j],5))
    title = '5X5 filter'
    plots[i].set_title(title)
for i in range(15,20):
    j = (i) - 15
    plots[i].imshow(cv2.medianBlur(images[j],7))
    title = '7X7 filter'
    plots[i].set_title(title)
for i in range(20,25):
    j = (i) - 20
    plots[i].imshow(cv2.medianBlur(images[j],9))
    title = '9X9 filter'
    plots[i].set_title(title)
for i in range(25,30):
    j = (i) - 25
    plots[i].imshow(cv2.medianBlur(images[j],11))
    title = '11x11 filter'

```

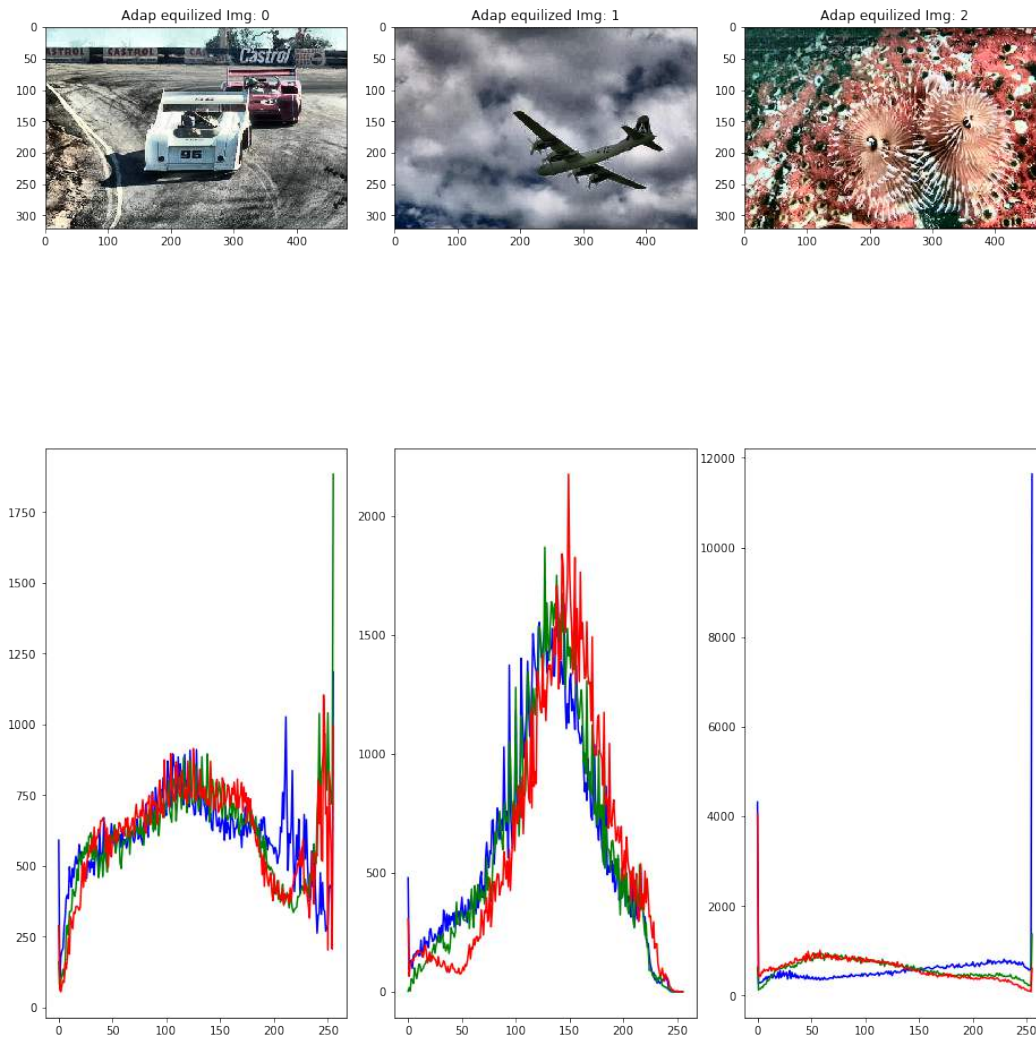


Figure 23: Adaptive Histogram Equalization (CLAHE) with clip size 5

```

plots[i].set_title(title)

# In[30]:

# Denoising filter
fig, plots = plt.subplots(3, 5, figsize=(23,23))
plots = plots.flatten()
fig.tight_layout()

for i in range(0,5):
    plots[i].imshow(images[i])
    title = 'Original Img: %s' % (i+1)
    plots[i].set_title(title)
for i in range(5,10):
    j = (i) - 5
    plots[i].imshow(cv2.medianBlur(cv2.medianBlur(cv2.medianBlur(images[j],3),3),3))
    title = '3X3 filter'
    plots[i].set_title(title)

```

```

for i in range(10,15):
    j = (i) - 10
    plots[i].imshow(cv2.medianBlur(images[j],7))
    title = '7X7 filter'
    plots[i].set_title(title)

# # Histogram
#
# So what is histogram ? You can consider histogram as a graph or plot, which gives y
#
# It is just another way of understanding the image. By looking at the histogram of a

# Lets identify some parts of the histogram:
#
# dims: The number of parameters you want to collect data of. In our example, dims =
# bins: It is the number of subdivisions in each dim. In our example, bins = 16
# range: The limits for the values to be measured. In this case: range = [0,255]
# So now we use cv2.calcHist() function to find the histogram. Let's familiarize with it
#
# cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])
# images : it is the source image of type uint8 or float32. it should be given in square brackets
# channels : it is also given in square brackets. It is the index of channel for which histogram is to be calculated
# mask : mask image. To find histogram of full image, it is given as "None". But if you want to find histogram of a specific region, you can pass a mask image.
# histSize : this represents our BIN count. Need to be given in square brackets. For 2D arrays, it is given as [bins for each dimension]
# ranges : this is our RANGE. Normally, it is [0,256].

# In[34]:

fig, plots = plt.subplots(2, 3, figsize=(30,30))
color = ('b','g','r')
plots = plots.flatten()
fig.tight_layout()

#plt.xlim([0,256])
for i in range(0,3):
    plots[i].imshow(eval("img"+str(i+1)))
    title = 'Original Img: %s' % (i)
    plots[i].set_title(title)
for i in range(3,6):
    j = (i+1) - 3
    for k,col in enumerate(color):
        histr = cv2.calcHist([eval("img"+str(j))], [k], None, [255], [0,256])
        plots[i].plot(histr,color = col)
    #plots[i].show()

# In[20]:

#Global Histogram Equalization
color = ('b','g','r')
fig, plots = plt.subplots(4, 3, figsize=(13,15))
plots = plots.flatten()
fig.tight_layout()
#original image
for i in range(0,3):
    plots[i].imshow(eval("img"+str(i+1)))
    title = 'Original Img: %s' % (i)
    plots[i].set_title(title)

```

```

    for k,col in enumerate(color):
        histr = cv2.calcHist([eval("img"+str(i+1))],[k],None,[256],[0,256])
        plots[i+3].plot(histr,color = col)

for i in range(6,9):
    j = (i+1) - 6
    img = cv2.cvtColor(eval("img"+str(j)), cv2.COLOR_BGR2YUV)
    y,u,v = cv2.split(img)
    y = cv2.equalizeHist(y)
    img = cv2.merge([y,u,v])
    img = cv2.cvtColor(img, cv2.COLOR_YUV2BGR)
    title = 'Equilized Img: %s' % (i)
    plots[i].set_title(title)
    plots[i].imshow(img)

    for k,col in enumerate(color):
        histr = cv2.calcHist([img],[k],None,[256],[0,256])
        plots[i+3].plot(histr,color = col)

# In[46]:

#Adaptive Histogram Equalization
size = [4,8,16,32,64]
for s in size:
    clahe_obj = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(s,s))
    color = ('b','g','r')
    fig, plots = plt.subplots(4, 3, figsize=(13,15))
    plots = plots.flatten()
    fig.tight_layout()
    #original image
    for i in range(0,3):
        plots[i].imshow(eval("img"+str(i+1)))
        title = 'Original Img: %s' % (i)
        plots[i].set_title(title)
        for k,col in enumerate(color):
            histr = cv2.calcHist([eval("img"+str(i+1))],[k],None,[256],[0,256])
            plots[i+3].plot(histr,color = col)

    for i in range(6,9):
        j = (i+1) - 6
        img = cv2.cvtColor(eval("img"+str(j)), cv2.COLOR_RGB2YUV)
        y,u,v = cv2.split(img)
        y = clahe_obj.apply(y)
        img = cv2.merge([y,u,v])
        img = cv2.cvtColor(img, cv2.COLOR_YUV2RGB)
        title = 'Adap equilized Img: %s' % (i)
        plots[i].set_title(title)
        plots[i].imshow(img)

        for k,col in enumerate(color):
            histr = cv2.calcHist([img],[k],None,[256],[0,256])
            plots[i+3].plot(histr,color = col)

# In[45]:

#Adaptive Histogram Equalization

```



```

clip = [1,1.5,2.3,3,5]
for c in clip:
    clahe_obj = cv2.createCLAHE(clipLimit=c, tileGridSize=(s,s))
    color = ('b','g','r')
    fig, plots = plt.subplots(2, 3, figsize=(13,15))
    plots = plots.flatten()
    fig.tight_layout()

    for i in range(0,3):
        j = i+1
        img = cv2.cvtColor(eval("img"+str(j)), cv2.COLOR_RGB2YUV)
        y,u,v = cv2.split(img)
        y = clahe_obj.apply(y)
        img = cv2.merge([y,u,v])
        img = cv2.cvtColor(img, cv2.COLOR_YUV2RGB)
        title = 'Adap equilized Img: %s' % (i)
        plots[i].set_title(title)
        plots[i].imshow(img)

        for k,col in enumerate(color):
            histr = cv2.calcHist([img],[k],None,[256],[0,256])
            plots[i+3].plot(histr,color = col)

# In[22]:

# Loading all images in data folder
import os
rootdir = '..\..'
images = []
imageNames = []
for subdir, dirs, files in os.walk(rootdir):
    for file in files:
        #print (file)
        filepath = subdir + os.sep + file
        if subdir.endswith('img') and filepath.endswith(".jpg"):
            #loading more images
            img = imgRead(filepath)
            if img is not None:
                images.append(img)
                imageNames.append(file)
                print (filepath)

# In[54]:

import evaluate
#Canny Edge Detector
fig, plots = plt.subplots(10, 2, figsize=(20,20))
plots = plots.flatten()
fig.tight_layout()
mean = 0
cnt = 0
#original image
for i in range(0,10):
    plots[cnt].set_title(title)
    plots[cnt].imshow(images[i])
    cnt = cnt + 1
    title = 'Original Img: %s' % (i)

```

```

gt = ".\\ground_truth\\" + imageNames[i].replace('.jpg', '.bmp')
edges = cv2.Canny(images[i],140,210)
cv2.imwrite('temp.jpg', edges)
im = cv2.imread('temp.jpg')
accuracy = evaluate.evaluate('temp.jpg', gt)
mean += accuracy
acc = round(accuracy,2)
print(imageNames[i] + " " + str(accuracy))
plots[cnt].imshow(edges, cmap='Greys_r')
plots[cnt].set_title("Canny accuracy: %s" % (acc))
cnt = cnt + 1

# In[60]:

import evaluate
sigma = 0.33
#Canny Edge Detector
fig, plots = plt.subplots(10, 2, figsize=(20,20))
plots = plots.flatten()
fig.tight_layout()
mean = 0
cnt = 0

#original image
for i in range(0,10):
    plots[cnt].set_title(title)
    plots[cnt].imshow(images[i])
    cnt = cnt + 1
    title = 'Original Img: %s' % (i)
    gt = ".\\ground_truth\\" + imageNames[i].replace('.jpg', '.bmp')
    # v = np.median(images[i])
    # lower = int(max(0, (1.0 - sigma) * v))
    # upper = int(min(255, (1.0 + sigma) * v))
    edges = cv2.Canny(images[i],200,225)
    cv2.imwrite('temp.jpg', edges)
    im = cv2.imread('temp.jpg')
    accuracy = evaluate.evaluate('temp.jpg', gt)
    mean += accuracy
    acc = round(accuracy,2)
    print(imageNames[i] + " " + str(accuracy))
    plots[cnt].imshow(edges, cmap='Greys_r')
    plots[cnt].set_title("Canny accuracy: %s" % (acc))
    cnt = cnt + 1

# In[52]:

import evaluate

#Canny Edge Detector
# fig, plots = plt.subplots(10, 2, figsize=(30,30))
# plots = plots.flatten()
# fig.tight_layout()
mean = 0
cnt = 0
high = [100,140,200,250,300,350]
#original image
for h in high:

```

```

    for i in range(0,10):
#         plots[cnt].set_title(title)
#         plots[cnt].imshow(images[i])
        cnt = cnt + 1
        title = 'Original Img: %s' % (i)
        gt = ".\\ground_truth\\" + imageNames[i].replace('.jpg', '.bmp')
        # Otsu's thresholding
        #         gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        #         ret2,th2 = cv2.threshold(gray,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
        #         print(ret2)
        edges = cv2.Canny(images[i],0.5*h,h)
        cv2.imwrite('temp.jpg', edges)
        im = cv2.imread('temp.jpg')
        accuracy = evaluate.evaluate('temp.jpg', gt)
        mean += accuracy
        acc = round(accuracy,2)
        print(imageNames[i] + " " + str(accuracy))
#         plots[cnt].imshow(edges, cmap='Greys_r')
#         plots[cnt].set_title("Canny accuracy: %s" % (acc))
        cnt = cnt + 1

# In[ ]:

# Loading all images in data folder in grayscale format
import os
rootdir = '..\..'
images = []
imageNames = []
for subdir, dirs, files in os.walk(rootdir):
    for file in files:
        #print (file)
        filepath = subdir + os.sep + file
        if subdir.endswith('img') and filepath.endswith(".jpg"):
            #loading more images
            img = cv2.imread(filepath)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            if img is not None:
                images.append(img)
                imageNames.append(file)
                print (filepath)

# In[ ]:

##### Sobel Edge Detector
from scipy import ndimage
import numpy as np
cnt = 0
fig, plots = plt.subplots(10, 2, figsize=(30,30))
plots = plots.flatten()
fig.tight_layout()
mean = 0
#original image

for i in range(0,10):
    plots[cnt].set_title(title)
    plots[cnt].imshow(images[i], cmap = 'gray')
    cnt = cnt + 1

```

```

title = 'Original Img: %s' % (i)
sobelx = cv2.Sobel(images[i],cv2.CV_64F,1,0,ksize=1)
sobely = cv2.Sobel(images[i],cv2.CV_64F,0,1,ksize=1)
gt = ".\\ground_truth\\" + imageNames[i].replace('.jpg','.bmp')
sx2 = np.square(sobelx)
sy2 = np.square(sobely)
edges = np.sqrt(sx2+sy2)
cv2.imwrite('temp.jpg', edges)
im = cv2.imread('temp.jpg')
accuracy = evaluate.evaluate('temp.jpg', gt)
mean += accuracy
acc = round(accuracy,2)
print(imageNames[i] + " " + str(accuracy))
plots[cnt].imshow(im, cmap='gray')
plots[cnt].set_title("Sobel accuracy: %s" % (acc))
cnt = cnt + 1

# In[ ]:

#Sobel Edge Detector
from scipy import ndimage
import numpy as np
cnt = 0
fig, plots = plt.subplots(10, 2, figsize=(30,30))
plots = plots.flatten()
fig.tight_layout()
mean = 0
#original image

for i in range(0,10):
    plots[cnt].set_title(title)
    plots[cnt].imshow(images[i], cmap = 'gray')
    cnt = cnt + 1
    title = 'Original Img: %s' % (i)
    sobelx = cv2.Sobel(images[i],cv2.CV_64F,1,0,ksize=5)
    sobely = cv2.Sobel(images[i],cv2.CV_64F,0,1,ksize=5)
    gt = ".\\ground_truth\\" + imageNames[i].replace('.jpg','.bmp')
    sx2 = np.square(sobelx)
    sy2 = np.square(sobely)
    edges = np.sqrt(sx2+sy2)
    cv2.imwrite('temp.jpg', edges)
    im = cv2.imread('temp.jpg')
    accuracy = evaluate.evaluate('temp.jpg', gt)
    mean += accuracy
    acc = round(accuracy,2)
    print(imageNames[i] + " " + str(accuracy))
    plots[cnt].imshow(im, cmap='gray')
    plots[cnt].set_title("Sobel accuracy: %s" % (acc))
    cnt = cnt + 1

# In[16]:

#Sobel Edge Detector
from scipy import ndimage
import numpy as np
cnt = 0
fig, plots = plt.subplots(10, 2, figsize=(30,30))

```



```

plots = plots.flatten()
fig.tight_layout()
mean = 0
#original image

for i in range(0,10):
    plots[cnt].set_title(title)
    plots[cnt].imshow(images[i], cmap = 'gray')
    cnt = cnt + 1
    title = 'Original Img: %s' % (i)
    sobelx = cv2.Sobel(images[i],cv2.CV_64F,1,0,ksize=3)
    sobely = cv2.Sobel(images[i],cv2.CV_64F,0,1,ksize=3)
    gt = ".\\ground_truth\\" + imageNames[i].replace('.jpg',''.bmp')
    sx2 = np.square(sobelx)
    sy2 = np.square(sobely)
    edges = np.sqrt(sx2+sy2)
    cv2.imwrite('temp.jpg', edges)
    im = cv2.imread('temp.jpg')
    accuracy = evaluate.evaluate('temp.jpg', gt)
    mean += accuracy
    acc = round(accuracy,2)
    print(imageNames[i] + " " + str(accuracy))
    plots[cnt].imshow(im, cmap='gray')
    plots[cnt].set_title("Sobel accuracy: %s" % (acc))
    cnt = cnt + 1

```

In[36]:

Loading all images in data folder

```

import os
rootdir = '..\..'
images = []
imageNames = []
for subdir, dirs, files in os.walk(rootdir):
    for file in files:
        #print (file)
        filepath = subdir + os.sep + file
        if subdir.endswith('img') and filepath.endswith(".jpg"):
            #loading more images
            img = imgRead(filepath)
            if img is not None:
                images.append(img)
                imageNames.append(file)
            print (filepath)

```

In[37]:

#SStructured Forests Edge Detector

```

from scipy import ndimage
import numpy as np
cnt = 0
fig, plots = plt.subplots(10, 2, figsize=(30,30))
plots = plots.flatten()
fig.tight_layout()
mean = 0
#original image

```

```

for i in range(0,10):
    plots[cnt].set_title(title)
    plots[cnt].imshow(images[i])
    cnt = cnt + 1
    title = 'Original Img: %s' % (i)
    gt = ".\\ground_truth\\" + imageNames[i].replace('.jpg','.bmp')
    output = ".\\output\\" + imageNames[i].replace('.jpg','.png')
    im = imgRead(output)
    accuracy = evaluate.evaluate(output, gt)
    mean += accuracy
    acc = round(accuracy,2)
    print(imageNames[i] + " " + str(accuracy))
    plots[cnt].imshow(im)
    plots[cnt].set_title("Structed Forests accuracy: %s" % (acc))
    cnt = cnt + 1

# In[ ]:

```



Figure 24: Results of sobel 1x1 filter on test images











Image	Filter 1x1 accuracy	Filter 3x3 accuracy	Filter 5x5 accuracy
	0.84	0.85	0.017
	0.83	0.51	0.0
	0.66	0.0613	0.0
	0.73	0.63	0.0
	0.75	0.039	0.0
	0.72	0.57	0.0
	0.71	0.422	0.0
	0.52	0.26	0.0
	0.19	0.0	0.0
	0.65	0.74	0.0

Figure 25: Accuracy of sobel with different filter sizes

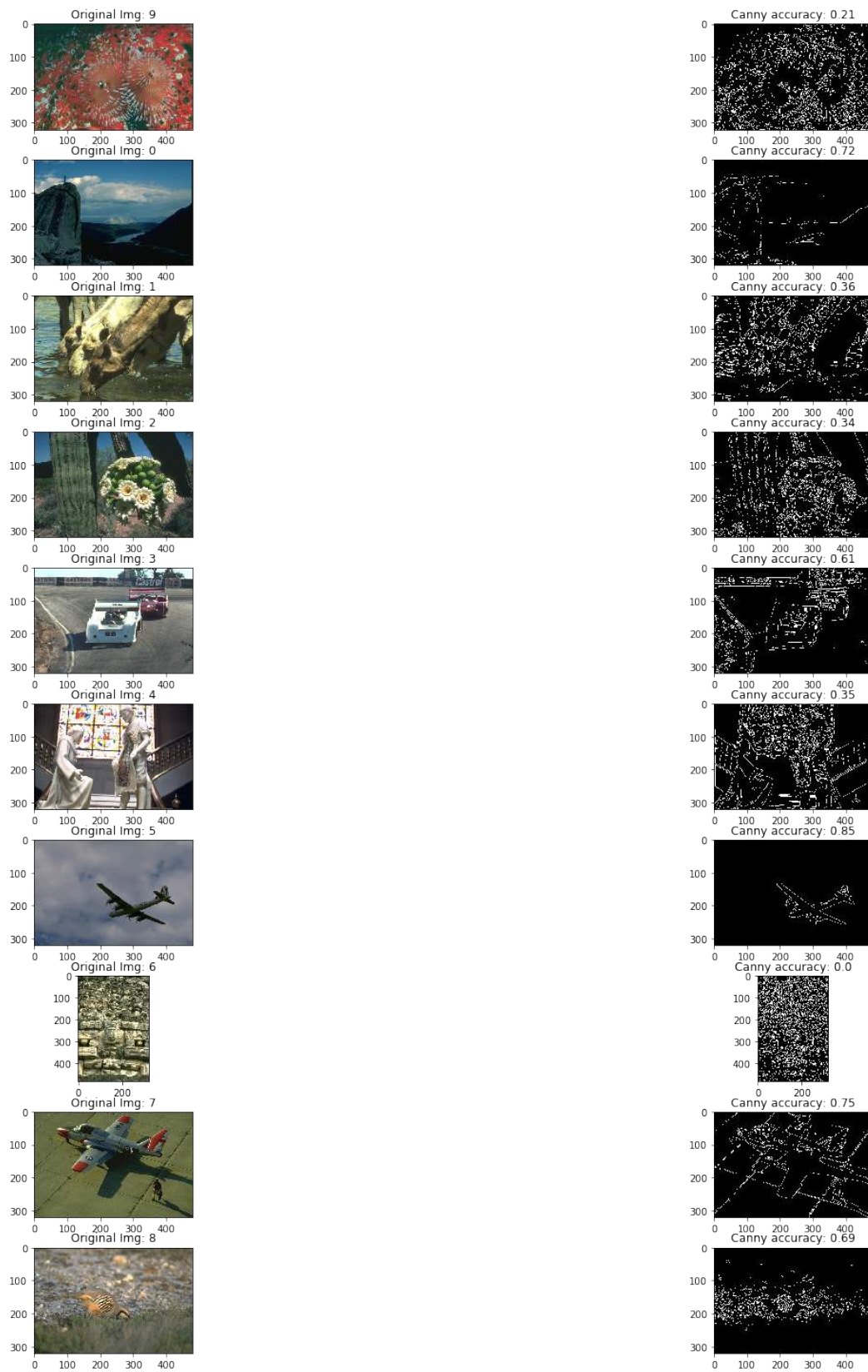
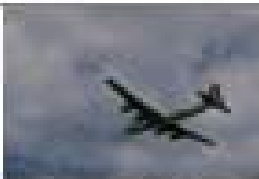

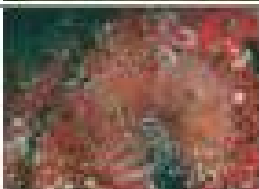
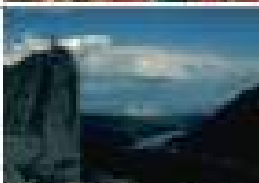
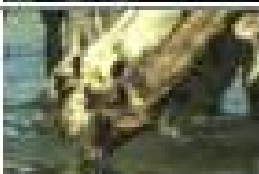
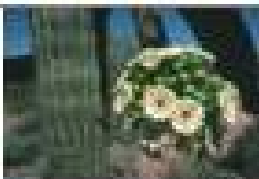

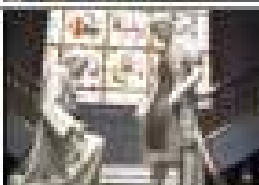



Figure 26: Edge detection using Canny operator

Image	Canny With threshold(200,225]
	0.85
	0.72
	0.27
	0.73
	0.42
	0.4
	0.62
	0.36
	0.0

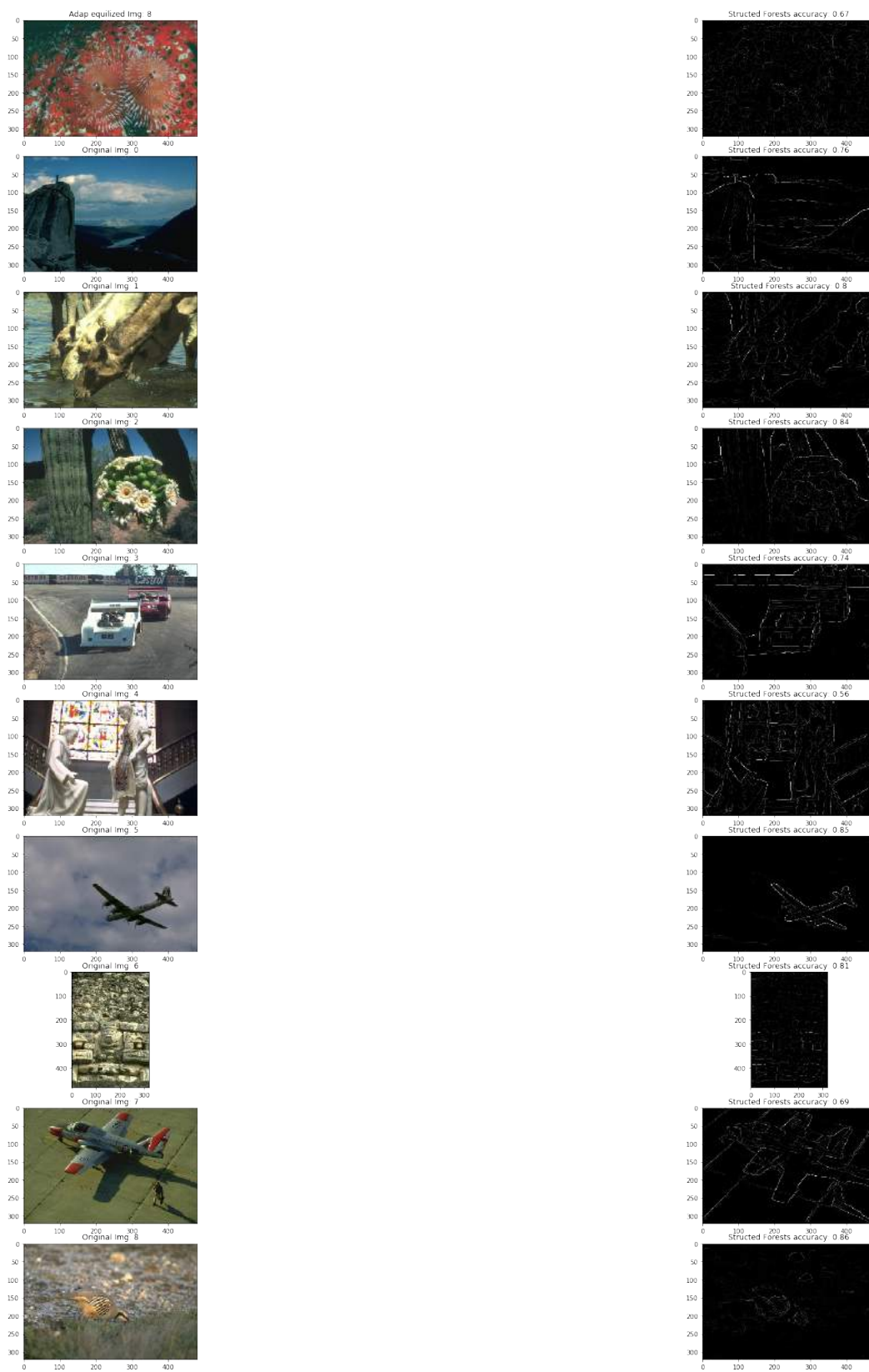


Figure 28: Edge detection using Structured Forest



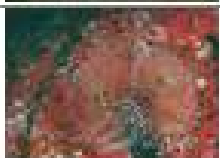

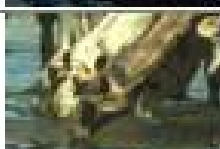



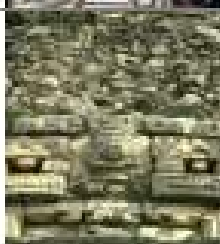

Image	Structured Forests
	0.85
	0.86
	0.67
	0.76
	0.8
	0.84
	0.74
	0.56
	0.81
	0.69

Figure 29: Accuracy using Structured Forest