**Task №1. Autocontrast**

Read an image from img\_1.png. Print the image and its histogram. Убидетесь используя гистограмму что изображение можно улучшить. Apply linear brightness alignment to each pixel of the image using formula:

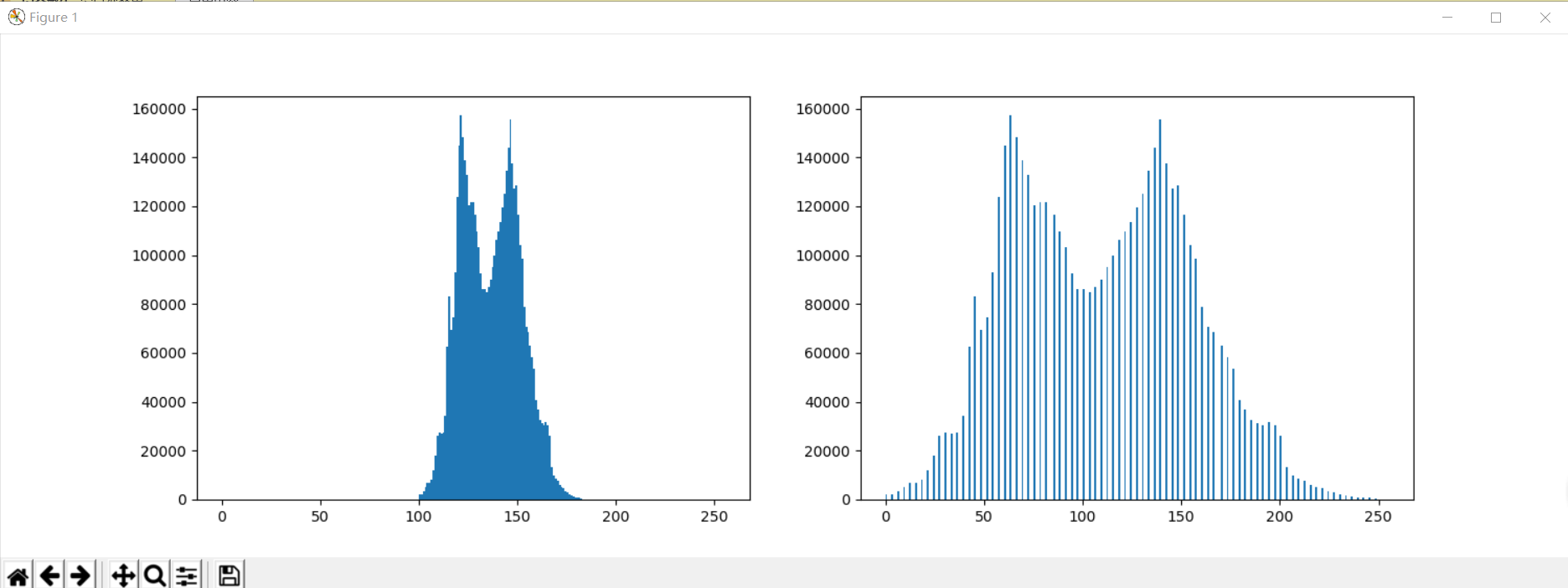


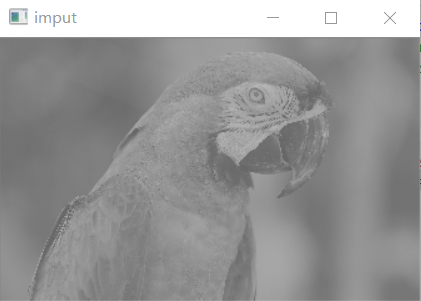
Wher x is the value of the pixel,x(min)is the minimum value of image,m(max)is the maximum value of image anf f(x) is the new value of the pixel.

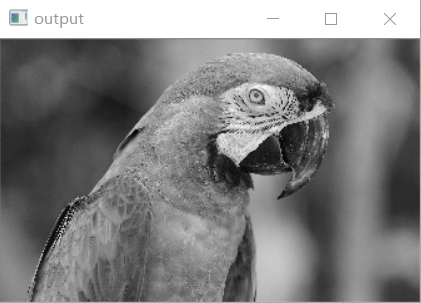
Show the resulting image and its histogram. Remember about the image type, formula above can be applied to byte image. If you use float image in range from 0 to 1 you must use formula above:



**import** cv2 **as** cv  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
src=cv.imread(**"D:/picture2/img\_1.png"**)  
src =np.float32(src)  
h,w=src.shape[:2]  
cv.imshow(**"imput"**,cv.resize(src,(w//5,h//5),interpolation=cv.INTER\_LINEAR).astype(np.uint8))  
src2=((src-src.min())/(src.max()-src.min()))\*255  
cv.imshow(**"output"**,cv.resize(src2,(w//5,h//5),interpolation=cv.INTER\_LINEAR).astype(np.uint8))  
plt.figure(figsize=(15,5))  
plt.subplot(1,2,1)  
plt.hist(src.ravel(), bins = 255, range=[0, 255])  
plt.subplot(1,2,2)  
plt.hist(src2.ravel(), bins = 255, range=[0, 255])  
plt.show()  
cv.waitKey(0)  
cv.destroyAllWindows()







**Task №2. Steady autocontrast**

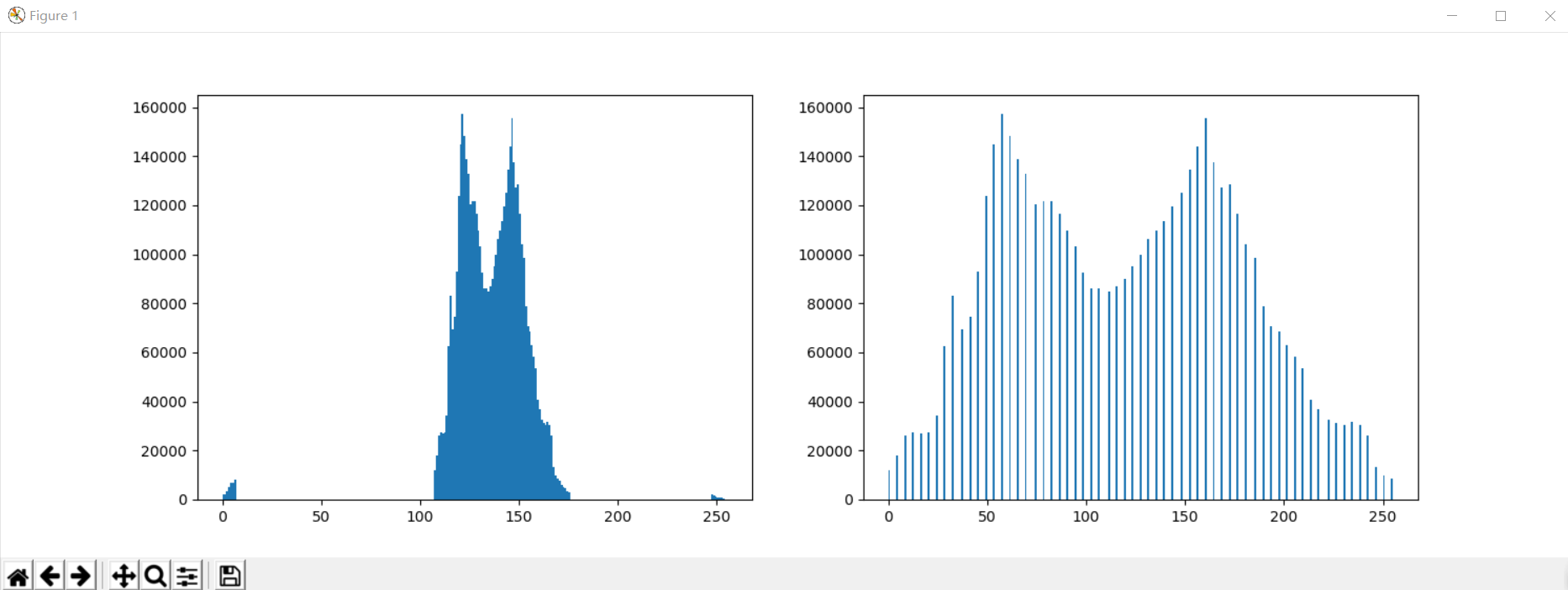
Load an image from the img\_2 file.png. Write a function that implements a stable autocontrast of the image.The steady autocontrast ignores some number of darkest and lightest pixels of the image.

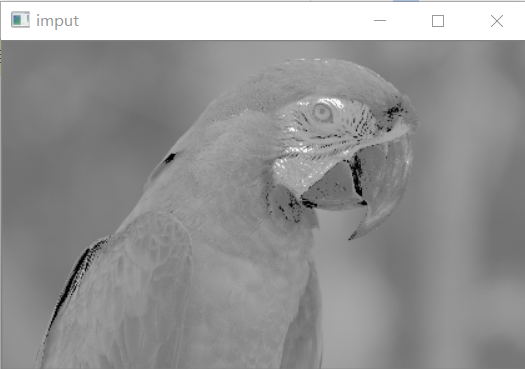
The input of the function is the image and two parameters in percentage: the number of the darkest pixels to be discarded and the number of the lightest pixels to be discarded.

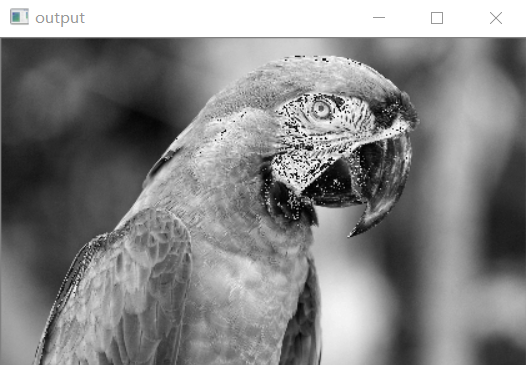
**import** cv2 **as** cv  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
src=cv.imread(**"D:/picture2/img\_2.png"**)  
src\_f =np.float32(src)  
h,w=src.shape[:2]  
cv.imshow(**"imput"**,cv.resize(src,(w//4,h//4),interpolation=cv.INTER\_LINEAR))  
hist=np.zeros([256],dtype=np.int32)  
**for** row **in** range(h):  
 **for** col **in** range(w):  
 pv=src[row,col]  
 hist[pv]+=1  
**for** i **in** range(256):  
 **if** (np.sum(hist[:i+1])/(h\*w))>=0.01:  
 left\_val = i-1  
 **break  
for** j **in** range(256):  
 **if** (np.sum(hist[:j+1])/(h\*w))>=0.99:  
 right\_val = j+1  
 **break**print(left\_val)  
print(right\_val)  
src2=((src\_f-left\_val)/(right\_val-left\_val))\*255  
cv.imshow(**"output"**,cv.resize(src2,(w//4,h//4),interpolation=cv.INTER\_LINEAR).astype(np.uint8))  
plt.figure(figsize=(15,5))  
plt.subplot(1,2,1)  
plt.hist(src.ravel(), bins = 255, range=[0, 255])  
plt.subplot(1,2,2)  
plt.hist(src2.ravel(), bins = 255, range=[0, 255])  
plt.show()  
cv.waitKey(0)  
cv.destroyAllWindows()

107

169



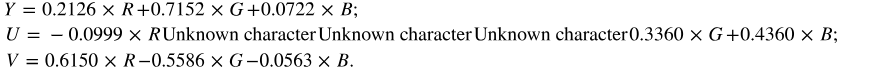




**Task number №3. Autocontrast of color images**

Read the color image from img\_3.png. Apply stable autocontrast to it:

1. Turn the image into a real number between 0 and 1.
2. Translate the image into YUV space according to the formulas:



Find the maximum and minimum for the steady autocontrast with dropping 5% of the lightest and 5% of the darkest pixels.

Apply the linear stretching of the Y channel using the formula



Clip the Y channel values from 0 to 1.

Convert the image in the RGB space according to the formulas:

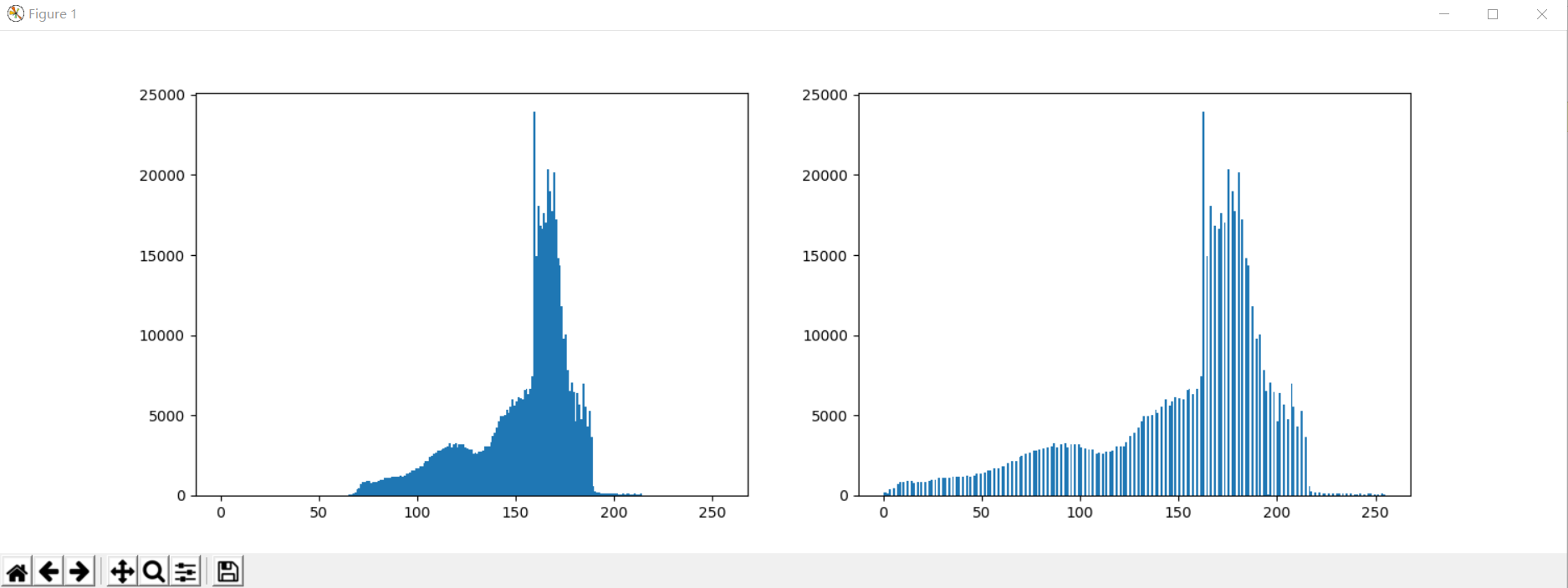


1. Clip the image values from 0 to 1.
2. Convert the image into integers from 0 to 255.

**import** cv2 **as** cv  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
RGB=cv.imread(**"D:/picture2/img\_3.png"**)  
*# src\_f =np.float32(src)  
# bgr=cv.split(src\_f)*h,w=RGB.shape[:2]  
cv.imshow(**"input"**,cv.resize(RGB,(w//2,h//2),interpolation=cv.INTER\_LINEAR))  
YUV=cv.cvtColor(RGB,cv.COLOR\_BGR2YUV)  
YUV=cv.split(YUV)  
plt.figure(figsize=(15,5))  
plt.subplot(1,2,1)  
plt.hist(YUV[0].ravel(), bins = 255, range=[0, 255])  
hist=np.zeros([256],dtype=np.int32)  
**for** row **in** range(h):  
 **for** col **in** range(w):  
 pv=YUV[0][row,col]  
 hist[pv]+=1  
**for** i **in** range(256):  
 **if** (np.sum(hist[:i+1])/(h\*w))>=0.0005:  
 left\_val = i-1  
 **break  
for** j **in** range(256):  
 **if** (np.sum(hist[:j+1])/(h\*w))>=0.9995:  
 right\_val = j+1  
 **break**print(left\_val)  
print(right\_val)  
YUV[0]=((YUV[0]-left\_val)/(right\_val-left\_val))\*255  
YUV[0]=YUV[0].astype(np.uint8)  
YUV2=cv.merge(YUV)  
BGR=cv.cvtColor(YUV2,cv.COLOR\_YUV2BGR)  
cv.imshow(**"output"**,cv.resize(BGR,(w//2,h//2),interpolation=cv.INTER\_LINEAR).astype(np.uint8))  
plt.subplot(1,2,2)  
plt.hist(YUV[0].ravel(), bins = 255, range=[0, 255])  
plt.show()  
cv.waitKey(0)  
cv.destroyAllWindows()

67

211

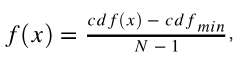






**Practical lesson №4. Equalize histogram. (Additional task)**

Read the image img\_2.png. Apply histogram equalization to it so that the cumulative distribution function (cdf) will be linear. The formula for converting image pixels:



Where cdf- the cumulative distribution function,N- the number of image pixels,cdf(min)- the minimum value of cdf not equal 0.

Hint: cdf equals the cumulative sum of the columns of the image histogram (np.cumsum)

Show the image before and after the histogram equalization. Show histograms of both images and their cdf.

**from** **skimage.data** **import** moon

img = moon()

plt.figure(figsize = (10, 5))

plt.subplot(121)

imshow(img)

plt.subplot(122)

h = plt.hist(img.ravel(), bins = 256, range=(0,255))[0];

cdf = np.cumsum(h)

plt.plot(range(len(h)), cdf / cdf.max() \* h.max());

*#https://blog.csdn.net/m0\_37477175/article/details/83108084***import** cv2 **as** cv  
**import** numpy **as** np  
**from** matplotlib **import** pyplot **as** plt  
img=cv.imread(**"D:/picture2/img\_2.png"**)  
h,w=img.shape[:2]  
hist = np.histogram(img.ravel(),256,[0,256])[0]  
cdf = hist.cumsum()  
cdf\_normalized = cdf \* hist.max()/ cdf.max()  
cdf\_m = np.ma.masked\_equal(cdf,0)  
cdf\_m = (cdf\_m - cdf\_m.min())\*255/(cdf\_m.max()-cdf\_m.min())  
cdf = np.ma.filled(cdf\_m,0).astype(**'uint8'**)  
img2 = cdf[img]  
h1,w1=img2.shape[:2]  
hist2,bins2 = np.histogram(img2.ravel(),256,[0,256])  
cdf2= hist2.cumsum()  
cdf2\_normalized2 = cdf2 \* hist2.max()/ cdf2.max()  
plt.figure(num=**'img'**)  
plt.plot(cdf\_normalized, color = **'r'**)  
plt.hist(img.ravel(),256,[0,256], color = **'g'**)  
plt.xlim([0,256])  
plt.legend((**'cdf'**,**'histogram'**))  
plt.figure(num=**'img2'**)  
plt.plot(cdf2\_normalized2, color = **'r'**)  
plt.hist(img2.ravel(),256,[0,256], color = **'g'**)  
plt.xlim([0,256])  
plt.legend((**'cdf2'**,**'histogram2'**))  
plt.show()  
cv.imshow(**"intput"**,cv.resize(img,(w//3,h//3),interpolation=cv.INTER\_LINEAR))  
cv.imshow(**"output"**,cv.resize(img2,(w1//3,h1//3),interpolation=cv.INTER\_LINEAR))  
cv.waitKey(0)  
cv.destroyAllWindows()

