-۱

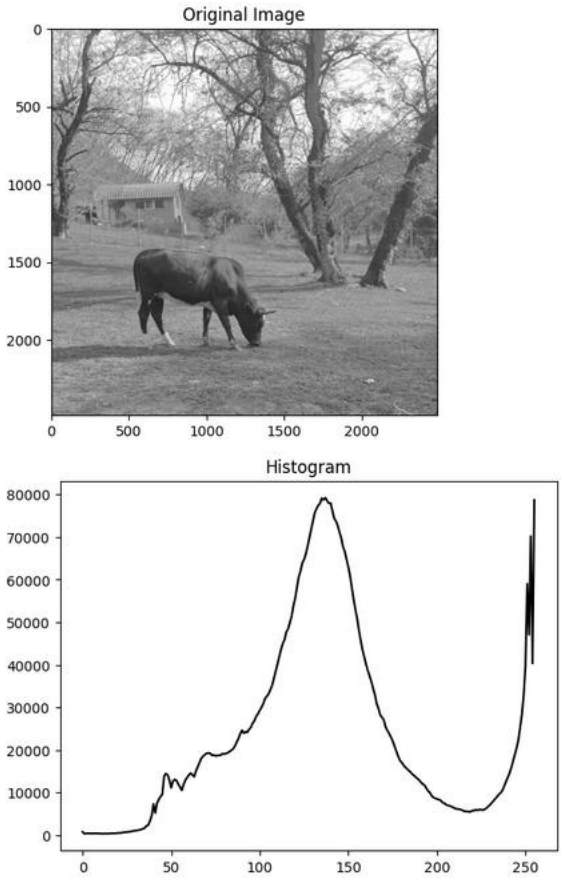
```
# Download the image from the URL
urllib.request.urlretrieve("https://drive.google.com/uc?export=download&id=1LKSbeIL0rk20PLPyXzu-QbXoLpr7mZTe", "sample.jpeg")

image = cv2.imread('sample.jpeg', cv2.IMREAD_GRAYSCALE)

# Display the original image
plt.figure(figsize=(5, 5))
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.show()
```

با ریکوئست زدن به لینک داده شده تصویر را دانلود میکنیم، میخوانیم و در هروجی نمایش میدهیم.

الف)

```
# a - Describe contrast level by checking the image's histogram
hist = cv2.calcHist([image], [0], None, [256], [0, 256])
plt.plot(hist, color='k')
plt.title('Histogram')
plt.show()
```

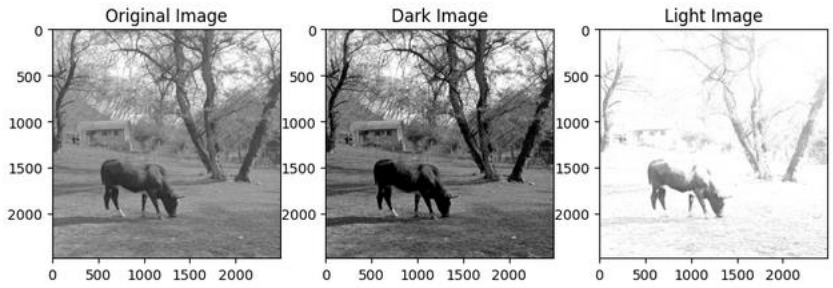
با استفاده از cv2 هیستوگرام تصویر را محاسبه میکنیم.



ب) برای پیاده سازی هیستوگرام sliding به مقدار الفا و بتا احتیاج داریم. مقادیر پیکسل تصویر جدید به صورت «alpha * pixel_value + beta» محاسبه می‌شود. مقادیر پیکسل به دست آمده در محدوده «[۰، ۲۵۵]» کات می شوند تا اطمینان حاصل شود که مقادیر شدت معتبر برای یک تصویر ۸ بیتی دارند. سپس تصویر به دست آمده به فرمت عدد صحیح بدون علامت ۸ بیتی تبدیل می شود.

برای روشن کردن تصویر هیستوگرام به سمت چپ شیفت داده میشود و برای تیره تر کردن به سمت راست.

```
# b - Implement histogram sliding
def histogram_sliding_manual(image, alpha, beta):
    new_image = np.clip(alpha * image + beta, 0, 255).astype(np.uint8)
    return new_image

light_image = histogram_sliding_manual(image, 1.5, 100)
dark_image = histogram_sliding_manual(image, 1.5, -100)

plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.subplot(1, 3, 2)
plt.imshow(dark_image, cmap='gray')
plt.title('Dark Image')
plt.subplot(1, 3, 3)
plt.imshow(light_image, cmap='gray')
plt.title('Light Image')
plt.show()
```




ث)

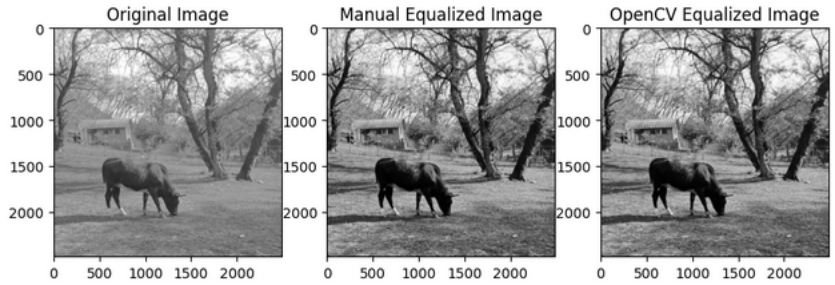
در اینجا باید فرمول هیستوگرام خواسته شده را پیاده سازی کنیم. این هیستوگرام تصویر ورودی را با استفاده از تابع «هیستوگرام» NumPy محاسبه می کند. هیستوگرام با مسطح کردن تصویر در یک آرایه ۱ بعدی و تقسیم مقادیر شدت به ۲۵۶ محاسبه می شود. دو مقدار بر میگرداند که صرفا هیستوگرام ان به درد ما میخورد.

سپس توزیع جمعی CDF هیستوگرام را محاسبه و نرمال میکنیم تا حداکثر مقدار آن برابر با حداکثر مقدار هیستوگرام باشد. حال این مقدار نباید ۰ داشته باشد تا تقسیم بر صفر نداشته باشیم. برای همین ۰ هارا مسک میکنیم و مقیاس مقادیر مسک شده را بین ۰ تا ۲۵۵ قرار میدهیم. در اخر هم مقادیر مسک شده در CDF مقیاس شده را با ۰ پر می کنیم و آرایه حاصل را به یک فرمت عدد صحیح بدون علامت ۸ بیتی تبدیل می کنیم. (این کار به این دلیل انجام می‌شود که CDF می‌تواند مقادیر ۰ داشته باشد، که باعث ایجاد خطای تقسیم بر صفر هنگام مقیاس‌بندی CDF به محدوده «[۰، ۲۵۵]» می‌شود. با پوشاندن مقادیر ۰

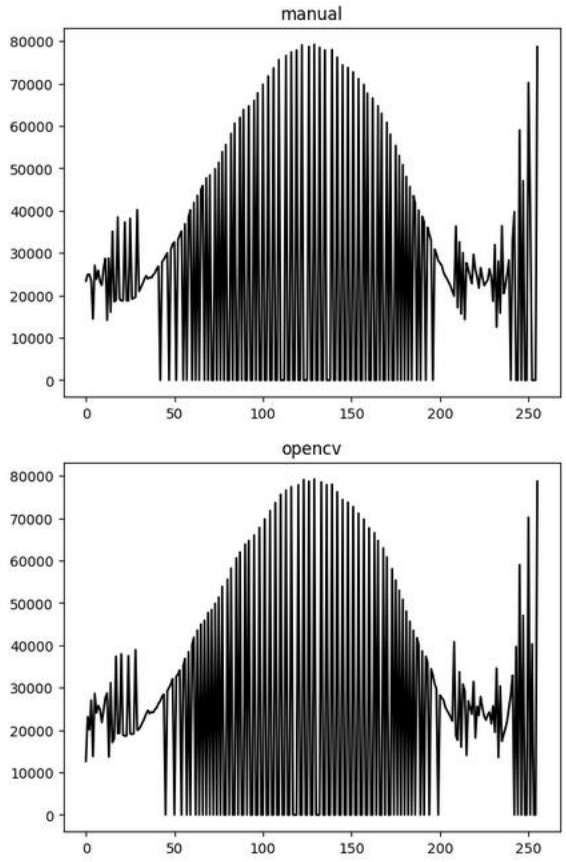
می توان عملیات مقیاس بندی را بدون خطا انجام داد. سپس آرایه پوشانده شده با مقادیر ۰ پر شده و به ترتیب با استفاده از توابع «filled» و «atype» به یک قالب عدد صحیح بدون علامت ۸ بیتی تبدیل می شود.)

در نهایت هم با استفاده از CDF برای نگاشت مقادیر intensity تصویر ورودی به مقادیر intensity جدید، histogram equalization transformation را به تصویر ورودی اعمال می کنیم.

```
#c - Implement histogram equalization and apply it to the image
def histogram_equalization_manual(image):
    histogram, bins = np.histogram(image.flatten(), 256, [0, 256])
    cdf = histogram.cumsum()
    cdf_normalized = cdf * histogram.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')
    equalized_image = cdf[image]
    return equalized_image

# Apply histogram equalization
equalized_image_manual = histogram_equalization_manual(image)

# Using OpenCV for histogram equalization
equalized_image_opencv = cv2.equalizeHist(image)

# Compare the two outputs
plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.subplot(1, 3, 2)
plt.imshow(equalized_image_manual, cmap='gray')
plt.title('Manual Equalized Image')
plt.subplot(1, 3, 3)
plt.imshow(equalized_image_opencv, cmap='gray')
plt.title('OpenCV Equalized Image')
plt.show()
```




سپس با استفاده از OpenCV هم همینکار را میکنیم و خروجی هارا به منظور مقایسه چاپ میکنیم. برای مقایسه بهتر مدل پیاده سازی شده هیستوگرام هر سه تصویر را  هم چاپ میکنیم.




همانطور که مسخص است به خوبی پیاده سازی انجام شده است.

۲-

برای حل این سوال از ایده الگوریتم dfs استفاده میکنیم. و با جستجو در تصویر و روی پیکسل ها قسمت هایی که چند پیکسل تقریبا سفید کنار هم هستند را شناسایی و میشماریم سپس مختصات مرکز این نقاط را محاسبه و داخل لیست قرار میدهیم که بتوانیم در خروجی چاپ نماییم.

توضیح کد:

```
# Read in the file
with open('input.txt', 'r') as file:
    filedata = file.read()

# Remove ' '
filedata = filedata.replace(', ', ',')

# Write again
with open('input.txt', 'w') as file:
    file.write(filedata)

# Seperate R and G and B
with open("input.txt", "r") as file:
    h, w, expected = map(int, file.readline().split())
    r_list, g_list, b_list = [], [], []
    for _ in range(h):
        r_row, g_row, b_row = zip(*[map(int, p.strip("()").split(",")) for p in file.readline().split()])
        r_list.append(list(r_row))
        g_list.append(list(g_row))
        b_list.append(list(b_row))
```

فایل مورد نظر را میخوانیم (نام فایل مطابق نام فایل مورد نظر باشد) برای اینکه بتوانیم راحت مقادیر سه کانال را جدا کنیم فاصله بعد از , را حذف میکنیم. فایل را مجدد میخوانیم و مقادیر طول و عرض تصویر و مقدار خواسته شده را در خط اول میخوانیم. سپس مقادیر هر سه کانال را به صورت مجزا در یک فابل قرار میدهیم.

```
threshold = 5
detected, stars = detect_stars(w, h, threshold, r_list, g_list, b_list)
while detected < expected and threshold > 0:
  threshold -= 1
  detect_stars(w, h, threshold, r_list, g_list, b_list)

print(len(stars))
for star in stars:
    print(f"{star[1]} {star[0]}")
```

مقدار ترشولد مقداری است که نشان میدهد اگر به این مقدار پیکسل به اندازه کافی روشن کنار هم باشند یعنی یک ستاره داریم. سپس تابع تشخیص فراخوانی میشود که در ادامه توضیح خواهیم داد و در نهایت اگر تعداد ستاره های تشخیث داده شده از تعداد ستاره های مورد انتظار کمتر باشد مقدار این ترشولد کم میشود تا مدل حساس تر عمل کند. در اخر هم در خروجی تعداد تشخیص داده شده و مختصات مرکز هر ستاره چاپ میشود.

```python
# function for detect stars
def detect_stars(w, h, threshold, r_list, g_list, b_list):
    visited = [[False] * w for _ in range(h)]

    # check color of a pixel
    def is_valid(x, y, r, g, b):
        return r > 240 and g > 240 and b > 240

    def dfs(x, y, star_points):
        stack = [(x, y)]
        while stack:
            x, y = stack.pop()
            if not (0 <= x < w and 0 <= y < h and not visited[y][x] and is_valid(x, y, r_list[y][x], g_list[y][x], b_list[y][x])):
                continue
            visited[y][x] = True
            star_points.append((x, y))
            for dx in range(-1, 2):
                for dy in range(-1, 2):
                    if dx == 0 and dy == 0:
                        continue
                    stack.append((x + dx, y + dy))
```

```python
stars = []
for y in range(h):
    for x in range(w):
        r, g, b = r_list[y][x], g_list[y][x], b_list[y][x]
        if not visited[y][x] and is_valid(x, y, r, g, b):
            star_points = []
            dfs(x, y, star_points)
            if len(star_points) >= threshold:
                stars.append((sum(p[0] for p in star_points) // len(star_points),
                              sum(p[1] for p in star_points) // len(star_points)))

return len(stars) , stars
```

این تابع ابعاد تصویر، مقدار ترشولد و لیست هر سه کانال رنگی را دریافت میکند. ابتدا یک لیست به اندازه تصویر برای مشخص کردن دیده شده یا نشده بودن هر پیکسل از تصویر درست میکنیم که طبیعتا درابتدا همه دیده نشده اند.

یک لیست برای ستاره ها در نظر میگیریم. روی تصویر پیمایش انجام میدهیم. و برای هر پیکسل مثادیر کانال های ان را مشخص میکنیم. اگر پیکسل به اندازه کافی سفید بود  تا کنون ویزیت نشده بود (با توجه به الگوریتم dfs که در ادامه توضیح داده میشود) الگوریتم dfs را اجرا میکنیم تا برای هر ستاره پیکسل های آن مشخص شوند. و اگر تعداد پیکسل های ستاره های مسخص شده از مقدار ترشولد بیشتر بود مسانگین مختصات پیکسل ها به عنوان مرکز ستاره مشخص میشود.

توضیح توابع:

یک تابع برای بررسی سفید بودن یا نبودن نقاط لازم داریم. یک پیکسل در صورتی سفید در نظر گرفته می شود که مقادیر قرمز، سبز و آبی آن همه بیشتر از ۲۴۰ باشد.این مقدار میتواند در حساسیت مدل تاثیرگذار باشد.

DFS: این تابع با انجام یک جستجوی عمقی (DFS) روی تصویر برای یافتن مناطق متصل از پیکسل‌ها که به اندازه کافی سفید هستند تا ستاره در نظر گرفته شوند، کار می‌کند. هنگامی که یک ناحیه متصل از پیکسل های سفید پیدا شد، مرکز منطقه به لیست ستارگان شناسایی شده اضافه می شود. توضیح عملکرد الگوریتم نیز به این صورت است که یک پشته در نظر میگیریم که در ابتدا شامل اولین پیکسل است. در یک حلقه پیکسل کنونی را مشخص و از استک خارج میکنیم و شرط قرار میدهیم. که آیا پیکسل فعلی در محدوده تصویر است، قبلاً بازدید نشده است و به اندازه کافی سفید است که به عنوان یک ستاره در نظر گرفته شود. اگر هر یک از این شرایط برآورده نشد، حلقه به تکرار بعدی ادامه میدهد. پیکسل کنونی ویزیت میشود و به لیست نقاط ستاره اضافه میشود و به سراغ پیکسل های اطراف ان میرویم.

برای فایل input داده شده ۱۲ ستاره تشخیص داده شد که شامل خروجی های داده شده در فایل output هم میشود.

```
12
86 33
130 89
141 158
144 151
153 156
170 56
206 117
214 21
218 155
260 155
271 233
296 224
```

یک راه حل دیگر این سوال حذف نویز از تصویر و مشخص کردن تعداد مناطق سفید باقی مانده است.