1.

- **روش اقلیدسی :** **الف)** برای حل سوال یک رنگ هدف که در اینجا قرمز می باشد را با استفاده از آرایه تعریف می کنیم :

```
target_color = np.array([0, 0, 255])
```

فاصله اقلیدسی بین هر پیکسل از تصویر و رنگ هدف را با استفاده از فرمول فاصله ی اقلیدسی محاسبه کرده و در distances ذخیره میکنیم :

$$D(z,a) = \|z-a\|$$
$$= [(z-a)^T(z-a)]^{\frac{1}{2}}$$
$$= [(z_R-a_R)^2 + (z_G-a_G)^2 + (z_B-a_B)^2]^{\frac{1}{2}}$$

```
distances = np.sqrt(np.sum((img - target_color) ** 2, axis=2))
```

مقادیر مختلفی را برای threshold تعریف میکنیم : (قسمت ب سوال)

```
thresholds = [80, 100, 150, 200]
```

سپس برای استخراج بخش قرمز تصویر از یک حلقه ی for استفاده می کنیم که هر پیکسلی که مقدارش از مقدار threshold بیشتر باشد را برابر 255 و بقیه را 0 قرار می دهد. سپس با استفاده از cv2.bitwise_and(img, img, mask=mask)، ماسک را با تصویر اصلی (img) and بیتی می کنیم تا بخش های قرمز تصویر را استخراج کنیم. نتیجه را در متغیر red_segment ذخیره می کنیم در نهایت، red_segment به لیست red_segments اضافه می‌شود. این لیست شامل بخش های قرمز تصویر با مقادیر threshold های مختلف خواهد بود.

```
red_segments = []
for th in thresholds:
    mask = np.where(distances <= th, 255, 0).astype(np.uint8)
    red_segment = cv2.bitwise_and(img, img, mask=mask)
    red_segments.append(red_segment)
```

تصویر را به سطوح خاکستری تبدیل می‌کنیم و با استفاده از ماتریس gray_image، رنگ خاکستری را به تصویر اصلی اضافه می‌کنیم.

```
gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray_image = cv2.cvtColor(gray_image, cv2.COLOR_GRAY2BGR)
```

در نهایت بخش های قرمز و تصویر خاکستری را ترکیب می‌کنیم و در لیست results ذخیره می‌کنیم.

```
results = []
for red_segment in red_segments:
    result = cv2.add(red_segment, gray_image)
    results.append(result)
```

کد کلی این بخش به صورت زیر است :

```python
# Read the image
img = cv2.imread("drive/MyDrive/DIP_EXC4/Q1/Q1_img.jpg")

# Define the target color(Red) for segmentation
target_color = np.array([0, 0, 255])

# Calculate Euclidean distance between each pixel and the target color
distances = np.sqrt(np.sum((img - target_color) ** 2, axis=2))

# Define threshold values
thresholds = [80, 100, 150, 200]

# Apply the masks to the image to get the red segments
red_segments = []
for th in thresholds:
    mask = np.where(distances <= th, 255, 0).astype(np.uint8)
    red_segment = cv2.bitwise_and(img, img, mask=mask)
    red_segments.append(red_segment)

# Convert the rest of the image to gray
gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray_image = cv2.cvtColor(gray_image, cv2.COLOR_GRAY2BGR)

# Combine the red segments and gray image
results = []
for red_segment in red_segments:
    result = cv2.add(red_segment, gray_image)
    results.append(result)

result1 = np.concatenate((results[0], results[1]), axis=1)
result2 = np.concatenate((results[2], results[3]), axis=1)
```

جدا کردن قسمت قرمز تصویر با روش اقلیدسی و مقادیر آستانه چپ بالا به راست پایین : 80، 100، 150، 200.

ب) با توجه به خروجی با مقادیر آستانه مختلف، مقدار آستانه ی 150 در روش اقلیدسی برای این تصویر بهتر از بقیه مقادیر می باشد، چرا که مقدار آستانه ی 50 هنوز به طور کامل قسمت های قرمز را استخراج نکرده و مقدار آستانه ی 200 هم بخش هایی از تصویر را رنگی کرده که کمی قرمزی داشتن ولی مورد نیاز ما نبوده و تصویر خروجی مطلوبی ارایه نکرده است.

- **روش مکعب** : **الف)** کد روش مکعب نیز مانند کد روش اقلیدسی می باشد با این تفاوت که قدر مطلق تفاوت بین هر پیکسل از تصویر و رنگ هدف را محاسبه کرده و در متغیر diff ذخیره می کنیم :

```
diff = np.abs(img.astype(np.int32) - target_color.astype(np.int32))
```

سپس با استفاده از np.max(diff, axis=2)، بزرگترین تفاوت را بین کانال‌های رنگی محاسبه می‌کنیم و در متغیر max_diff ذخیره می کنیم.

```
max_diff = np.max(diff, axis=2)
```

مقادیر مختلفی را برای threshold تعریف میکنیم : (قسمت ب سوال)

```
thresholds = [100, 150, 200, 255]
```

در هر مرحله از حلقه for، با استفاده از عبارت np.where(max_diff <= th/2, 255, 0)، یک ماسک ایجاد می‌شود. این ماسک مقادیری که بیشتر از نصف مقدار آستانه (th/2) نیستند را برابر 255 قرار می‌دهد و بقیه مقادیر را برابر 0 قرار می‌دهد. با استفاده از .astype(np.uint8)، نوع داده ماسک به uint8 تغییر می‌کند.

```
for th in thresholds:
    mask = np.where(max_diff <= th/2, 255, 0).astype(np.uint8)
```

و بقیه مراحل استخراج بخش قرمز تصویر و ترکیب آن با تصویر خاکستری مانند بخش قبل می باشد.

کد کلی این بخش به صورت زیر است :

```
# Calculate the absolute differences between each channel and the target color
diff = np.abs(img.astype(np.int32) - target_color.astype(np.int32))

# Calculate the maximum difference along the channel axis
max_diff = np.max(diff, axis=2)

# Define threshold values
thresholds = [100, 150, 200, 255]

# Apply the masks to the image to get the red segments
red_segments = []
for th in thresholds:
    mask = np.where(max_diff <= th/2, 255, 0).astype(np.uint8)
    red_segment = cv2.bitwise_and(img, img, mask=mask)
    red_segments.append(red_segment)

# Combine the red segments and gray image
results = []
for red_segment in red_segments:
    result = cv2.add(red_segment, gray_image)
    results.append(result)

result1 = np.concatenate((results[0], results[1]), axis=1)
result2 = np.concatenate((results[2], results[3]), axis=1)
```

جدا کردن قسمت قرمز تصویر با روش اقلیدسی و مقادیر آستانه چپ بالا به راست پایین : 100, 150, 200, 255.

ب) با توجه به خروجی با مقادیر آستانه مختلف، مقدار آستانه ی 200 در روش اقلیدسی برای این تصویر بهتر از بقیه مقادیر می باشد، چرا که مقدار آستانه ی 100 هنوز به طور کامل قسمت های قرمز را استخراج نکرده و مقدار آستانه ی255 هم بخش هایی از تصویر را رنگی کرده که کمی قرمزی داشتن ولی مورد نیاز ما نبوده و تصویر خروجی مطلوبی ارایه نکرده است.

2. روش Intensity Segmentation :

برای حل این سوال ابتدا یک تصویر خالی با فرمت RGB ایجاد می‌شود که اندازه آن با اندازه تصویر ورودی برابر است :

```
height, width = img.shape
colorized_img = np.zeros((height, width, 3), dtype=np.uint8)
```

بازه‌های segmentation را تعریف می‌کنیم. این بازه‌ها شامل اعدادی هستند که مشخص می‌کنند کدام بخش از سطوح خاکستری تصویر باید به چه رنگی تبدیل شود و در متغیر segments ذخیره می‌کنیم :

```
segments = [(0, 50), (51, 100), (101, 150), (150, 200), (201, 255)]
```

با استفاده از تابع plt.get_cmap نقشه رنگی با نام 'viridis' را دریافت می‌کنیم و در متغیر colormap ذخیره می‌کنیم :

```
colormap = plt.get_cmap('viridis')
```

تعداد بازه‌های segmentation را با استفاده از تابع len برابر با تعداد بازه‌های تعریف شده در segments محاسبه می‌کنیم و در متغیر num_segments ذخیره می‌کنیم :

```
num_segments = len(segments)
```

با استفاده از یک حلقه for به طول num_segments، رنگ متناظر با هر بازه را از نقشه رنگی دریافت می‌کنیم و در متغیر colors ذخیره می‌کنیم :

```
colors = [colormap(i / num_segments) for i in range(num_segments)]
```

سپس با استفاده از دو حلقه for به ترتیب برای هر پیکسل از تصویر سطح خاکستری، رنگ متناظر را بر اساس بازه‌های segmentation مشخص می‌کنیم و به آن پیکسل در تصویر رنگی اختصاص می‌دهیم. این کار با استفاده از مقایسه پیکسل با محدوده‌های رنگی در segments و انتخاب متناظر با آن محدوده رنگ انجام می‌شود:

```
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        pixel_value = img[i, j]
        for k in range(num_segments):
            if segments[k][0] <= pixel_value <= segments[k][1]:
                colorized_img[i, j] = (int(colors[k][0] * 255), int(colors[k][1] * 255), int(colors[k][2] * 255))
                break
```

کد کلی این بخش به صورت زیر است :

```
# Read the image
img = cv2.imread("drive/MyDrive/DIP_EXC4/Q2/Q2_img.jpg", 0)

# Create a blank RGB image
height, width = img.shape
colorized_img = np.zeros((height, width, 3), dtype=np.uint8)

# Define the segment ranges
segments = [(0, 50), (51, 100), (101, 150), (150, 200), (201, 255)]

# Obtain colormap colors
colormap = plt.get_cmap('viridis')
num_segments = len(segments)
colors = [colormap(i / num_segments) for i in range(num_segments)]

# Iterate through each pixel of the grayscale image and assign color
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        pixel_value = img[i, j]

        # Check the pixel value against the color ranges
        for k in range(num_segments):
            if segments[k][0] <= pixel_value <= segments[k][1]:
                # Assign the corresponding color from the colormap
                colorized_img[i, j] = (int(colors[k][0] * 255), int(colors[k][1] * 255), int(colors[k][2] * 255))
                break
```

خروجی تصویر رنگی با روش Intensity Segmentation :

روش Transformation:

برای حل این سوال ابتدا تابع transform_func را تعریف می‌کنیم که سه ورودی دریافت می‌کند: تصویر (image) و دو پارامتر a و b.

```
def transform_func(image, a, b):
```

با استفاده از تابع np.zeros یک تصویر خالی با ابعاد مشابه تصویر ورودی و فرمت RGB ایجاد می کنیم :

```
  colored_image = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
```

مقادیر کانال های رنگی آبی(B) و سبز (G) را برابر با تصویر ورودی قرار می‌دهیم، به این صورت که هر دو کانال با داده های تصویر سطح خاکستری یکسان مقداردهی می‌شوند.

```
  colored_image[:, :, 0] = img
  colored_image[:, :, 1] = img
```

در کانال قرمز (R)، از فرمول تبدیل **a * image + b** برای اعمال تبدیل استفاده می‌کنیم. این تبدیل بر اساس پارامترهای a و b تعریف شده است. برای این که هدف سوال قرمز شدن تاج خروس بود، برای کانال قرمز تابع تبدیل را نوشتیم . این تبدیل باعث تغییر رنگ های تصویر می‌شود :

```
  colored_image[:, :, 2] = a * image + b
```

در این مثال، پارامتر های a و b را به طور تصادفی آزمایش کردیم و مقدار های زیر برای قرمز بودن تاج خروس بهتر از بقیه مقادیر بود :

```
# Transformation parameter
a = 1.8
b = 80
# Colorize the image
transform_func(img, a, b)
```

کد کلی این بخش به صورت زیر است :

```
# Show the original image
print("Original image :\n")
cv2_imshow(img)

def transform_func(image, a, b):

    # Apply the transformation formula to colorize the image
    colored_image = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
    colored_image[:, :, 0] = img
    colored_image[:, :, 1] = img
    colored_image[:, :, 2] = a * image + b # Because the rooster's crown should be red

    # Show the colorized image
    print("\nColorized image with Transformation method :\n")
    cv2_imshow(colored_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Transformation parameter
a = 1.8
b = 80

# Colorize the image
transform_func(img, a, b)
```

خروجی تصویر رنگی با روش Transformation :