BLG453E: Computer Vision Assignment 1

Uğur Ali Kaplan
150170042

Question 1

For this one, I have followed the instructions in the homework.

My code:

```
# Reading files
data_path = os.path.join(os.curdir, "HW1_material")
background = cv2.imread(os.path.join(data_path, "Malibu.jpg"))

background_height, background_width, _ = background.shape
ratio = 360 / background_height

# Resize background and protect aspect ratio
background = cv2.resize(background, (int(background_width * ratio), 360))

cat_path = os.path.join(data_path, "cat") # Path to the cat images

# Place the cat values onto background
image = cv2.imread(os.path.join(cat_path, "cat_5.png"))
image_g_channel = image[:, :, 1]
image_r_channel = image[:, :, 0]

foreground = np.logical_or(image_g_channel < 180, image_r_channel > 150)
nonzero_x, nonzero_y = np.nonzero(foreground)

nonzero_cat_values = image[nonzero_x, nonzero_y, :]

new_frame = background.copy()
new_frame[nonzero_x, nonzero_y, :] = nonzero_cat_values
new_frame = new_frame[:, :, [2, 1, 0]]

# Create the video file
images_list = list()
num_images = len(os.listdir(cat_path)) # Find the number of cat images in the
folder
for num in range(num_images):
    image = cv2.imread(os.path.join(cat_path, "cat_" + str(num) + ".png"))
    image_g_channel = image[:, :, 1]
    image_r_channel = image[:, :, 0]

    foreground = np.logical_or(image_g_channel < 180, image_r_channel > 150)
    nonzero_x, nonzero_y = np.nonzero(foreground)

    nonzero_cat_values = image[nonzero_x, nonzero_y, :]

    new_frame = background.copy()
    new_frame[nonzero_x, nonzero_y, :] = nonzero_cat_values
    new_frame = new_frame[:, :, [2, 1, 0]]
```

```
    images_list.append(new_frame)

clip = mpy.ImageSequenceClip(images_list, fps=25)
audio = mpy.AudioFileClip(os.path.join(data_path,
"selfcontrol_part.wav")).set_duration(clip.duration)
clip = clip.set_audio(audioclip = audio)
clip.write_videofile("part1_video.mp4", codec="libx264")
```

My output is the file called `part1_video.mp4`.

Question 2

In this question, I have mirrored the image by reversing the values in the rows. Then, since I want my second cat to be aligned to the right, I have calculated how much of a shift is required and finally, I have overwritten the coordinates for both the image and the mirrored image.

```
1. Reverse the image by reversing the order of values in rows
2. Calculate the shift required to align the mirrored image to the right
3. Overwrite the calculated coordinates in the background
```

```
# We have to shift the mirrored image as image width is less than background
width
translate = background.shape[1] - cv2.imread(os.path.join(cat_path,
"cat_5.png")).shape[1]

# Do the operations for all frames and create the video

images_list = list()
num_images = len(os.listdir(cat_path))
for num in range(num_images):
    image = cv2.imread(os.path.join(cat_path, "cat_" + str(num) + ".png"))
    m_image = image[:, ::-1, :]

    image_g_channel = image[:, :, 1]
    image_r_channel = image[:, :, 0]
    m_image_g_channel = m_image[:, :, 1]
    m_image_r_channel = m_image[:, :, 0]

    foreground = np.logical_or(image_g_channel < 180, image_r_channel > 150)
    m_foreground = np.logical_or(m_image_g_channel < 180, m_image_r_channel >
150)

    nonzero_x, nonzero_y = np.nonzero(foreground)
    m_nonzero_x, m_nonzero_y = np.nonzero(m_foreground)

    nonzero_cat_values = image[nonzero_x, nonzero_y, :]
    m_nonzero_cat_values = m_image[m_nonzero_x, m_nonzero_y, :]

    new_frame = background.copy()
    new_frame[nonzero_x, nonzero_y, :] = nonzero_cat_values
    new_frame[m_nonzero_x, m_nonzero_y + translate, :] = m_nonzero_cat_values
    new_frame = new_frame[:, :, [2, 1, 0]]
    images_list.append(new_frame)

clip = mpy.ImageSequenceClip(images_list, fps=25)
```

```
audio = mpy.AudioFileClip(os.path.join(data_path,
"selfcontrol_part.wav")).set_duration(clip.duration)
clip = clip.set_audio(audioclip = audio)
clip.write_videofile("part2_video.mp4", codec="libx264")
```

Output file is `part2_video.mp4`.

Question 3

To make the cat dance with its shadow, I have written the following function:

```
def make_darker(x, dec_intensity):
    """
    Takes an input image x, with 3 channels and represented as RGB.
    Decreases the intensity according to provided argument, dec_intensity.
    If the intensity of a pixel is less than dec_intensity, its new value is
mapped to 0.
    """
    x = x - [dec_intensity, dec_intensity, dec_intensity]
    x[x < 0] = 0
    return x
```

This function needs a 3-channel image and how much of a density decrease is required as inputs. Then, if any of the values became negative through this operation, it makes them zero.

To try different decrease values, I have used a for loop.

```
# Create multiple videos with different darkened cats
for dec in [20, 50, 100, 150, 200, 250]:
    images_list = list()
    num_images = len(os.listdir(cat_path))
    for num in range(num_images):
        image = cv2.imread(os.path.join(cat_path, "cat_" + str(num) + ".png"))
        m_image = image[:, ::-1, :]

        image_g_channel = image[:, :, 1]
        image_r_channel = image[:, :, 0]
        m_image_g_channel = m_image[:, :, 1]
        m_image_r_channel = m_image[:, :, 0]

        foreground = np.logical_or(image_g_channel < 180, image_r_channel >
150)
        m_foreground = np.logical_or(m_image_g_channel < 180, m_image_r_channel
> 150)

        nonzero_x, nonzero_y = np.nonzero(foreground)
        m_nonzero_x, m_nonzero_y = np.nonzero(m_foreground)

        nonzero_cat_values = image[nonzero_x, nonzero_y, :]
        m_nonzero_cat_values = m_image[m_nonzero_x, m_nonzero_y, :]

        new_frame = background.copy()
        new_frame[nonzero_x, nonzero_y, :] = nonzero_cat_values
        m_nonzero_cat_values = make_darker(m_nonzero_cat_values, dec)
        new_frame[m_nonzero_x, m_nonzero_y + translate, :] =
m_nonzero_cat_values
```

```
        new_frame = new_frame[:, :, [2, 1, 0]]
        images_list.append(new_frame)

    clip = mpy.ImageSequenceClip(images_list, fps=25)
    audio = mpy.AudioFileClip(os.path.join(data_path,
"selfcontrol_part.wav")).set_duration(clip.duration)
    clip = clip.set_audio(audioclip = audio)
    clip.write_videofile("part3_video_decreaseby" + str(dec) + ".mp4",
codec="libx264")
```

Cat got darker and darker as `dec_intensity` increased as expected. For this exercise, I got six output files in the format `part3_video_decrease_by_dec_intensity.mp4` where `dec_intensity` was $20, 50, 100, 150, 200, 250$.

Question 4

For this question, I have picked the "Starry Night" by Van Gogh as my target image.

```
target_image = cv2.imread(os.path.join(data_path, "StarryNight.jpg"))
```

I have also written the following helper functions:

```
def histogram(image):
    _, _, channel = image.shape

    hist = np.zeros((256, 1, channel), dtype=np.uint64)

    for g in range(256):
        hist[g, 0, ...] = np.sum(np.sum(image == g, 0), 0)
```

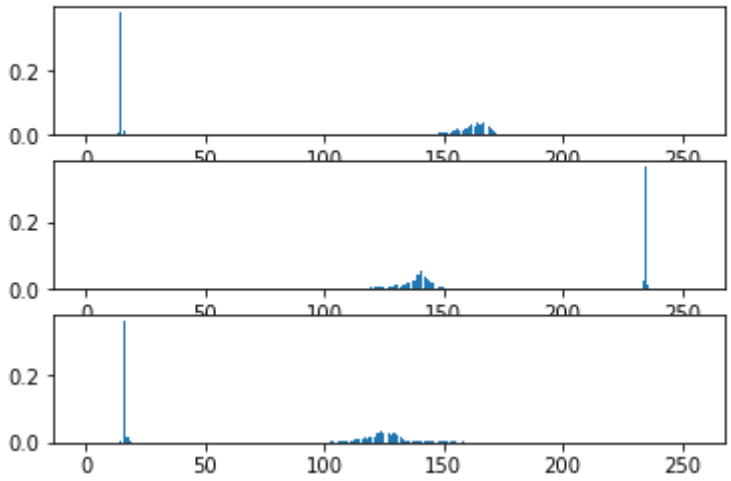
```
    return hist

def hist_to_pdf(hist):
    hist = hist.astype(np.float32)
    hist[:, 0, 2] = hist[:, 0, 2] / hist[:, 0, 2].sum(0)
    hist[:, 0, 1] = hist[:, 0, 1] / hist[:, 0, 1].sum(0)
    hist[:, 0, 0] = hist[:, 0, 0] / hist[:, 0, 0].sum(0)
    return hist

def get_pdf(image):
    hist = histogram(image).astype(np.float32)
    return hist_to_pdf(hist)

def pdf_to_cdf(pdf):
    cdf = np.zeros(pdf.shape)
    for channel in range(cdf.shape[-1]):
        cdf[:, :, channel] = np.cumsum(pdf[:, :, channel], axis=0)

    return cdf

def histogram_matching(image_cdf, target_cdf):

    lut = np.zeros(image_cdf.shape)

    for channel in range(lut.shape[-1]):
        g_target = 0
        for g_image in range(256):
            while ((g_target < 255) and \
                   (image_cdf[g_image, 0, channel] < 1) and \
                   target_cdf[g_target, 0, channel] < image_cdf[g_image, 0,
channel]):
                g_target += 1
            lut[g_image, 0, channel] = g_target

    return lut
```

Then, I have calculated the CDF and PDF of cat images. Here are some of my results:

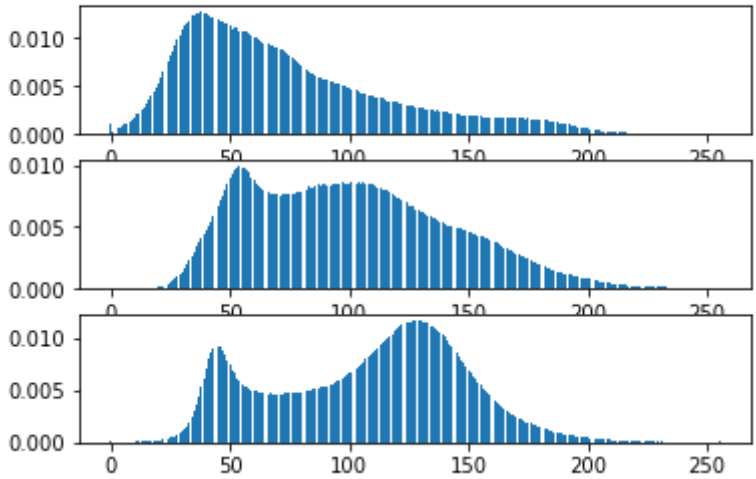
PDFs

In the PDF of the average cat, I have seen pixel intensities clustering around certain points. I think this is related to the green screen in the images, as I have also confirmed my results with the help of an image editing program.

Normalized Average Cat Red, Green and Blue Channels in Order

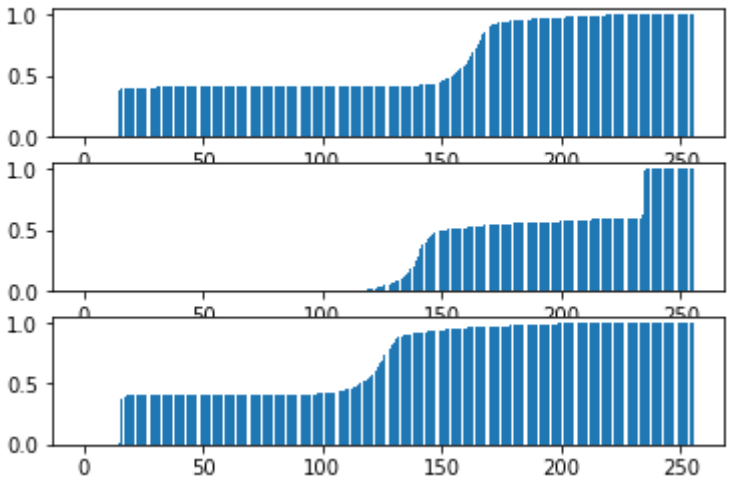


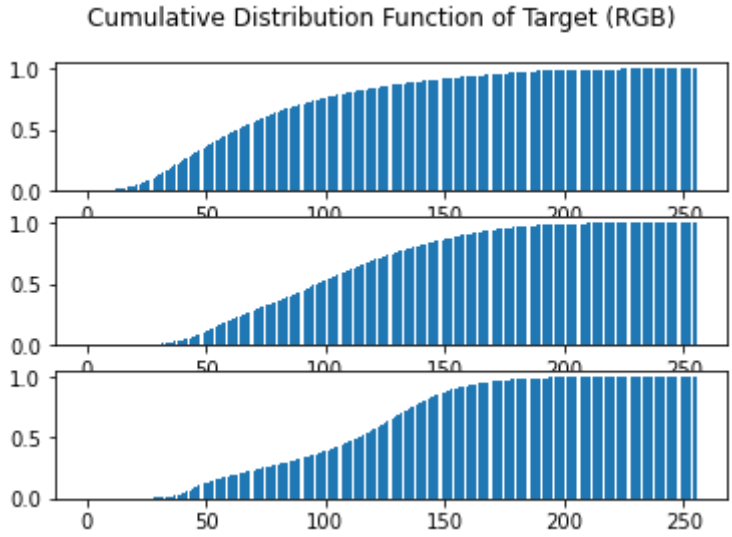
Target Image PDFs (RGB)



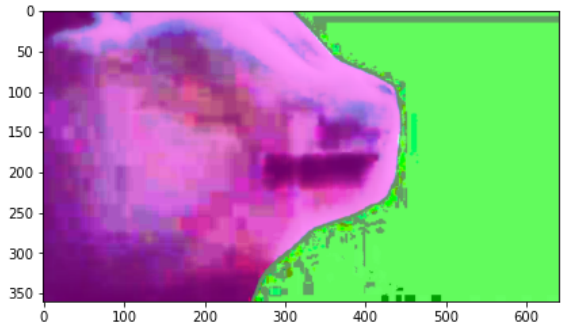
CDFs

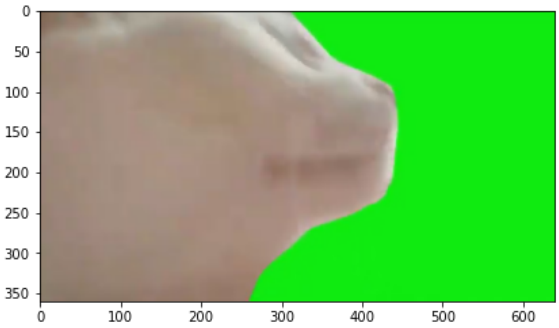
Cumulative Distribution Function of Average Cat (RGB)

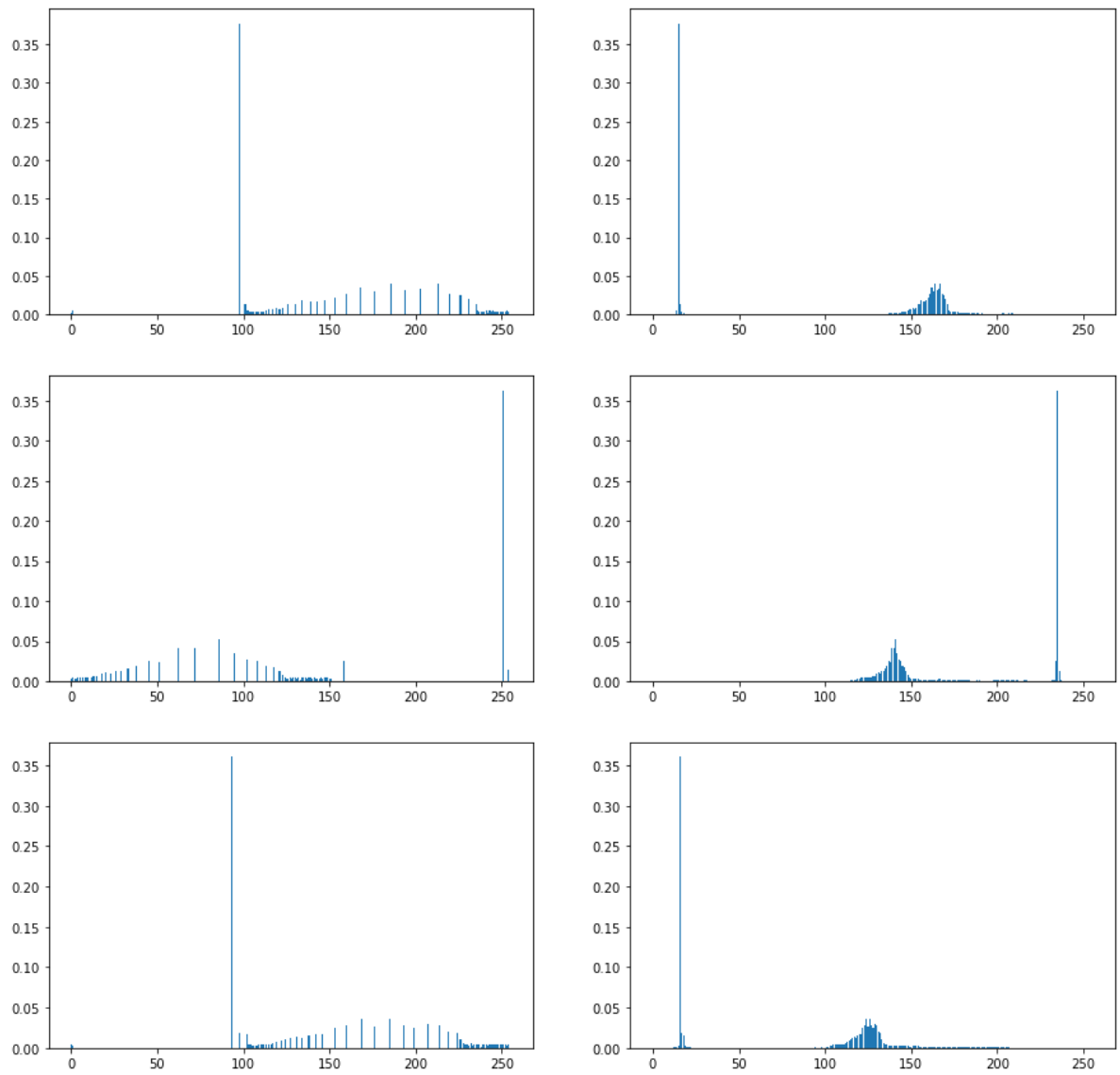




Histogram Equalization

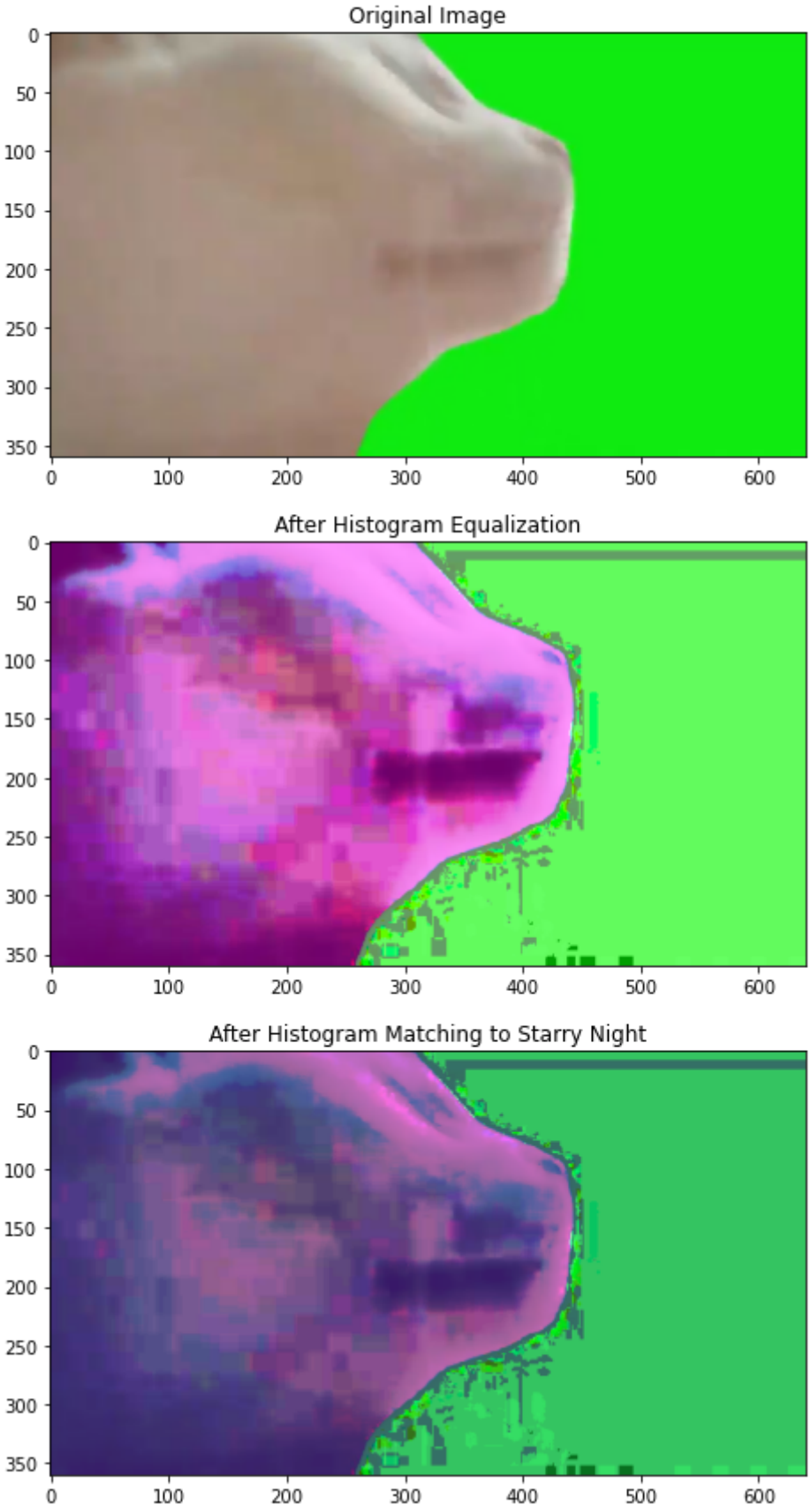






Histogram Matching

Comparison of Cat Images



Video

Using these operations, I have created the video with the following code:

```
images_list = list()
num_images = len(os.listdir(cat_path))
for num in range(num_images):
    image = cv2.imread(os.path.join(cat_path, "cat_" + str(num) + ".png"))
    m_image = image[:, ::-1, :]

    image_g_channel = image[:, :, 1]
    image_r_channel = image[:, :, 0]
    m_image_g_channel = m_image[:, :, 1]
    m_image_r_channel = m_image[:, :, 0]

    foreground = np.logical_or(image_g_channel < 180, image_r_channel > 150)
    m_foreground = np.logical_or(m_image_g_channel < 180, m_image_r_channel >
150)

    nonzero_x, nonzero_y = np.nonzero(foreground)
    m_nonzero_x, m_nonzero_y = np.nonzero(m_foreground)

    nonzero_cat_values = image[nonzero_x, nonzero_y, :]
    m_nonzero_cat_values = m_image[m_nonzero_x, m_nonzero_y, :]

    new_frame = background.copy()
    new_frame[nonzero_x, nonzero_y, :] = nonzero_cat_values
    m_nonzero_cat_values[:, 2] = np.uint8(mapping[:, 0, 2]
[m_nonzero_cat_values[:, 2]])
    m_nonzero_cat_values[:, 1] = np.uint8(mapping[:, 0, 1]
[m_nonzero_cat_values[:, 1]])
    m_nonzero_cat_values[:, 0] = np.uint8(mapping[:, 0, 0]
[m_nonzero_cat_values[:, 0]])
    new_frame[m_nonzero_x, m_nonzero_y + translate, :] = m_nonzero_cat_values
    new_frame = new_frame[:, :, [2, 1, 0]]
    images_list.append(new_frame)

clip = mpy.ImageSequenceClip(images_list, fps=25)
audio = mpy.AudioFileClip(os.path.join(data_path,
"selfcontrol_part.wav")).set_duration(clip.duration)
clip = clip.set_audio(audioclip = audio)
clip.write_videofile("part4.mp4", codec="libx264")
```

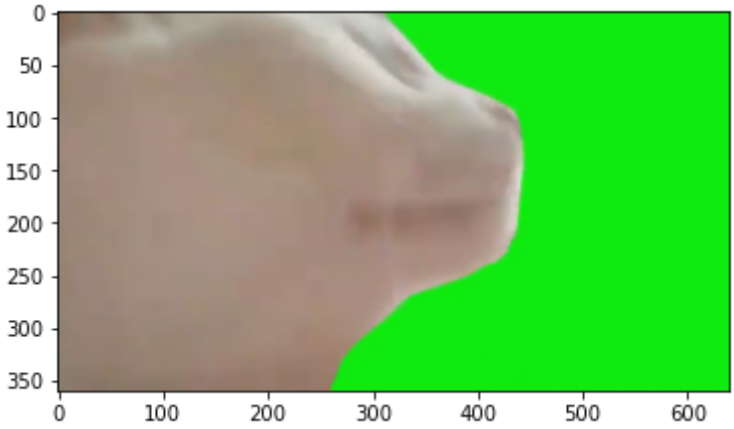
My output for this part is `part4.mp4`.

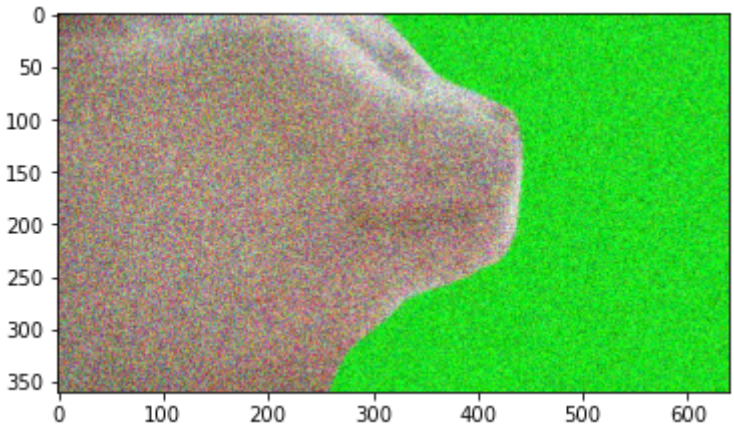
Question 5

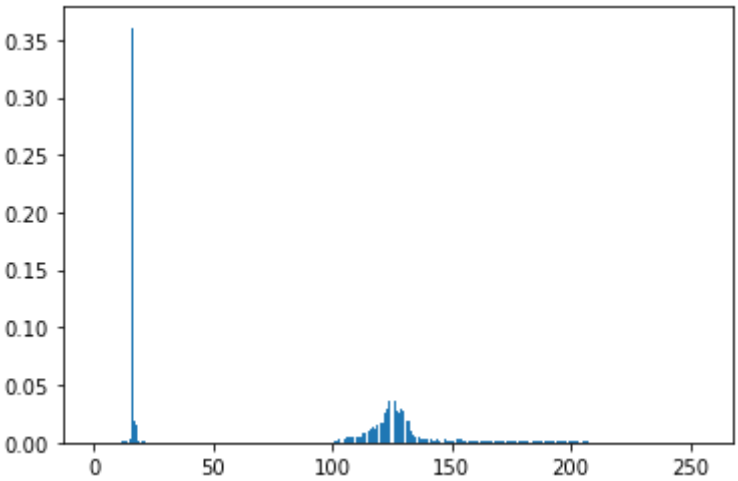
For this question, I have written a `perturb` helper function as most of the necessary parts were written for the previous question.

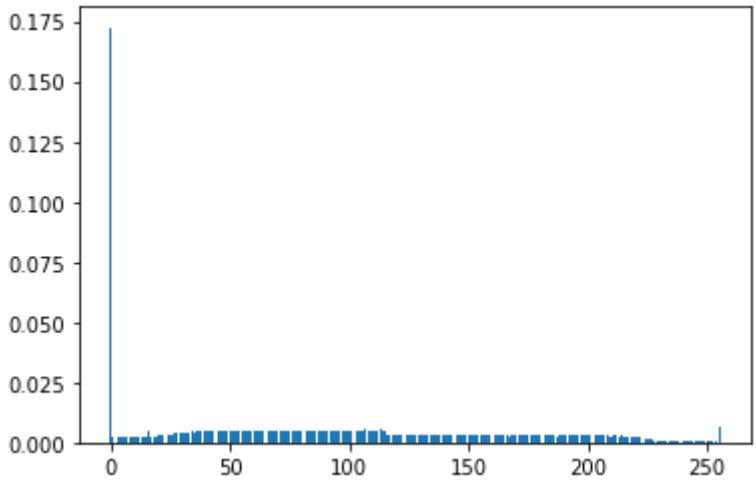
```
def perturb(image):
    image = image.astype(np.int64)
    noise = ((np.random.rand(*image.shape) * 200) - 100).astype(np.int64)
    image += noise
    image[image>255] = 255
    image[image<0] = 0
    image = image.astype(np.uint8)
    return image
```

As an example, this function produces the following:









Here is my code:

```
images_list = list()
num_images = len(os.listdir(cat_path))
for num in range(num_images):
    image = cv2.imread(os.path.join(cat_path, "cat_" + str(num) + ".png"))
    m_image = image[:, ::-1, :]

    image_g_channel = image[:, :, 1]
    image_r_channel = image[:, :, 0]
    m_image_g_channel = m_image[:, :, 1]
    m_image_r_channel = m_image[:, :, 0]

    foreground = np.logical_or(image_g_channel < 180, image_r_channel > 150)
    m_foreground = np.logical_or(m_image_g_channel < 180, m_image_r_channel >
150)

    # Equalization

    # 1. Perturb the image on the left
    image_p = perturb(image)

    # 2. Calculate the pdf of the perturbed cat on the left
    image_p_pdf = get_pdf(image_p)

    # 3. Calculate the pdf of the clean cat on the left
    image_pdf = get_pdf(image)

    # 4. Calculate the pdf of the cat on the right
    m_image_pdf = get_pdf(m_image)

    # 5. Randomly perturb the target image (Starry Night)
    target_p = perturb(target_image)

    # 6. Calculate the pdf of the perturbed target image
    target_p_pdf = get_pdf(target_p)

    # 7. Create cdfs
    image_p_cdf = pdf_to_cdf(image_p_pdf)
    image_cdf = pdf_to_cdf(image_pdf)
    m_image_cdf = pdf_to_cdf(m_image_pdf)
    target_p_cdf = pdf_to_cdf(target_p_pdf)
```

```
    # 8. LUT for Histogram matching of the cat on the left
    left_map = histogram_matching(image_cdf, image_p_cdf)

    # 9. LUT for Histogram matching of the cat on the right
    right_map = histogram_matching(m_image_cdf, target_p_cdf)

    nonzero_x, nonzero_y = np.nonzero(foreground)
    m_nonzero_x, m_nonzero_y = np.nonzero(m_foreground)

    nonzero_cat_values = image[nonzero_x, nonzero_y, :]
    m_nonzero_cat_values = m_image[m_nonzero_x, m_nonzero_y, :]

    # 10. New values for the cat on the left
    nonzero_cat_values[:, 2] = np.uint8(left_map[:, 0, 2][nonzero_cat_values[:,
2]])
    nonzero_cat_values[:, 1] = np.uint8(left_map[:, 0, 1][nonzero_cat_values[:,
1]])
    nonzero_cat_values[:, 0] = np.uint8(left_map[:, 0, 0][nonzero_cat_values[:,
0]])

    # 11. New values for the cat on the right
    m_nonzero_cat_values[:, 2] = np.uint8(right_map[:, 0, 2]
[m_nonzero_cat_values[:, 2]])
    m_nonzero_cat_values[:, 1] = np.uint8(right_map[:, 0, 1]
[m_nonzero_cat_values[:, 1]])
    m_nonzero_cat_values[:, 0] = np.uint8(right_map[:, 0, 0]
[m_nonzero_cat_values[:, 0]])

    new_frame = background.copy()
    new_frame[nonzero_x, nonzero_y, :] = nonzero_cat_values
    new_frame[m_nonzero_x, m_nonzero_y + translate, :] = m_nonzero_cat_values
    new_frame = new_frame[:, :, [2, 1, 0]]
    images_list.append(new_frame)
    print(f"{num + 1}/{num_images} Complete!")

clip = mpy.ImageSequenceClip(images_list, fps=25)
audio = mpy.AudioFileClip(os.path.join(data_path,
"selfcontrol_part.wav")).set_duration(clip.duration)
clip = clip.set_audio(audioclip = audio)
clip.write_videofile("part5.mp4", codec="libx264")
```

However, this did not produce a disco effect. Maybe my noise was too small to make a significant change for every frame. My output is `part5_old.mp4`.

To get the disco effect in my mind, I have changed my perturb function a little bit and now it allows us to specify the amount of noise that can be added to the original image. Also, instead of clipping, I am letting the intensities to overflow.

```
def perturb(image, r = 150):
    image = image.astype(np.int64)
    noise = ((np.random.rand(*image.shape) * r) - r // 2).astype(np.int64)
    image += noise
    image = image.astype(np.uint8)
    return image
```

And now, I am also randomizing the perturbation process by randomly generating the parameter `r`. Specifically, I have changed two parts in my video generating loop:

```
# 1. Perturb the image on the left
image_p = perturb(image, r=np.random.rand() * 100)

# 5. Randomly perturb the target image (Starry Night)
target_p = perturb(target_image, r=np.random.rand() * 300)
```

And this output is `part5.mp4`.