Assignment 1

Batuhan Faik Derinbay derinbay18@itu.edu.tr
150180705

Part 1: Dancing Alone

In this part, a video of a cat dancing in the Vice City Malibu Club is aimed to be obtained. To start, the background image ("`Malibu.jpg`") and a frame of the cat ("`cat_0.png`") is read using the `imread` function of the Python OpenCV library and then displayed with the help of `matplotlib pyplot` library. After displaying the pictures, the background image is resized using the `resize` function of the OpenCV library in order the match the height of the cat frames.

Matching the Height of the Background to the Cat

```
background_height = background.shape[0]
background_width = background.shape[1]
cat_height = cat_0.shape[0]
ratio = cat_height/background_height

background = cv2.resize(background, (int(background_width*ratio), ↩
    cat_height))
```

Once the background image is at the same height as the cat frames, green backgrounds of the cat frames need to be eliminated. To do that, green and red channels of the frames are separated. Then green pixel indices with a value less then 180 and red pixel indices with a value more then 150 are located. Using these indices the cat without a green background is obtained and placed on top of the background, obtaining a `new_frame`.

Removing the Green Background from the Cat Frames and Placing the Cat on the Background Image

```
cat_0_green = cat_0[:,:,1]
cat_0_red = cat_0[:,:,0]

foreground = np.logical_or(cat_0_green < 180, cat_0_red > 150)
nonzero_x, nonzero_y = np.nonzero(foreground)
nonzero_cat = cat_0[nonzero_x, nonzero_y, :]

new_frame = background.copy()
new_frame[nonzero_x, nonzero_y, :] = nonzero_cat
```

After successfully placing the cat on top of the background, in order to render a video this process has to be done for many frames to come. Therefore, a list containing directories of all the cat frames ("`cat_frames`") is created and sorted using a human sort algorithm (The sorting algorithm is adopted from "https://nedbatchelder.com/blog/200712/human_sorting.html").

Reading all the Cat Frame Directories and Sorting Them

```
import re
def atoi(text):
    return int(text) if text.isdigit() else text
# The human sort algorithm
def natural_keys(text):
    return [atoi(c) for c in re.split(r'(\d+)', text)]
path = "cat"
cat_frames = [os.path.join(path, f) for f in os.listdir(path) if os.↩
    path.isfile(os.path.join(path, f))]
cat_frames.sort(key=natural_keys)
```

Finally, all the cat frames in the `cat_frames` list is read in order, previously explained transformations are applied and the resulting `new_frame` is appended to the `new_cat_frames` list. To render a video file with these new frames named "`part1_video.mp4`", `ImageSequenceClip` and `write_videofile` functions of the `moviepy` library is used. Also, by using the `AudioFileClip` and the `set_audio` functions of the same library, the "`selfcontrol_part.wav`" audio file is added to the clip.

Rendering the Video

```
clip = mpy.ImageSequenceClip(new_cat_frames, fps=25)
audio = mpy.AudioFileClip("selfcontrol_part.wav").set_duration(clip.↩
    duration)
clip = clip.set_audio(audioclip=audio)
clip.write_videofile("part1_video.mp4", codec="libx264")
```

Part 2: Dancing with Myself

In this part, the cat needs to be mirrored and placed on the right side of the background. To achieve that `flip` function of the `numpy` library is used and the frame is flipped on the vertical (indexed "1") axis. Then, distance from the rightmost pixel of the cat frame to the rightmost pixel of the background is calculated ("`horizontal_shift`"). This distance is then used to move (shift) the cat frame on the X (horizontal) axis.

Mirroring the Cat

```
cat_0_mirrored = np.flip(cat_0, 1)
```

Placing the Mirrored Cat on the Right Side of the Background

```
new_frame[nonzero_x, nonzero_y + horizontal_shift, :] = nonzero_cat
```

Finally, the video is rendered using the same method as aforementioned.

Part 3: Dancing with My Shadow

In this part, by using point-wise operations the brightness of the cat on the right is decreased according to a previously defined transform mapping. In order to change the brightness of the cat, a function named `change_brightness` which takes an image, converts it to HSV, changes the third channel (brightness) according to input parameter and returns the manipulated image in BGR is defined.

Function to Change the Brightness of an Image

```
def change_brightness(image, brightness=0):
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV).astype("int16")
    hsv[:,:,2] += brightness
    hsv[:,:,2] = hsv[:,:,2].clip(min=0, max=255)
    return cv2.cvtColor(hsv.astype("uint8"), cv2.COLOR_HSV2BGR)
```

After displaying the right cat that is 39% darker (`brightness = -100` and $100*100/256 = 39.06$) than its original, all the new cat frames with different brightness values are appended to the `new_cat_frames` list to ensure trying different parameters. As can be seen on the output "`part3_video.mp4`" video file, the brightness of the cat on the right decreases with each frame. This effect is achieved via the following code snippet.

Linearly Decreasing Brightness Between Each Cat Frame

```
for idx, frame in enumerate(cat_frames):
    # ...
    brightness = -idx
    # ...
    cat_dark = change_brightness(cat_dark, brightness)
    # ...
```

Finally, the video is rendered using the same method as aforementioned.

Part 4: Dancing with My Friend

In this part, histogram matching between the cat on the right and a target image is applied to the video frames. The target image is chosen as the background image which is "`Malibu.jpg`".

In order to do histogram matching, histograms of the both images needs to be obtained. For this, two functions with the names `get_cat_hist` and `get_hist` are defined respectively.

Since only the foreground of the cat frames need to be addressed during the histogram calculation, the algorithm goes through every pixel of the image in each channel which takes quite a bit of time and processing power. Even though it can be improved, it is sufficient enough for this use case. It should also be noted that rendering videos that use this function will take some time to complete. Nonetheless, `get_hist` function uses `numpy`'s built-in `histogram` function, therefore is significantly faster.

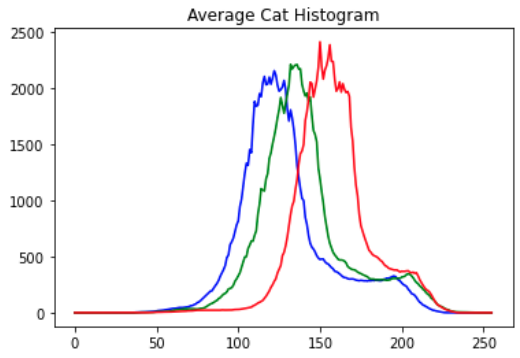
get_cat_hist Function

```
1  def get_cat_hist(image):
2      fg = np.logical_or(image[:,:,1] < 180, image[:,:,0] > 150)
3      nz_x, nz_y = np.nonzero(fg)
4      nz_cat = image[nz_x, nz_y, :]
5
6      hist = np.zeros([256, nz_cat.shape[1]]).astype("uint32")
7      for color in range(nz_cat.shape[1]):
8          for pixel in range(nz_cat.shape[0]):
9              hist[nz_cat[pixel, color], color] += 1
10     return hist
```

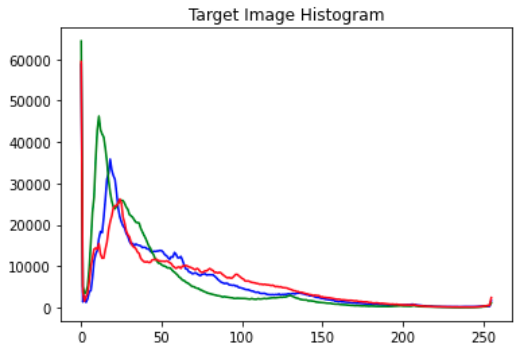
get_hist Function

```
1  def get_hist(image):
2      hist = np.empty([256, image.shape[-1]])
3      for idx in range(hist.shape[-1]):
4          hist_, bins = np.histogram(image[:,:,idx], 256, [0, 256])
5          hist[:, idx] = hist_
6      return hist, bins
```

After obtaining the average of the cat histograms by appending histograms of the all cat frames and calculating the mean of it, histograms of the cat and the target image are as follows (Figure 1i, 1ii).



(i) Average Histogram of Cat Frames



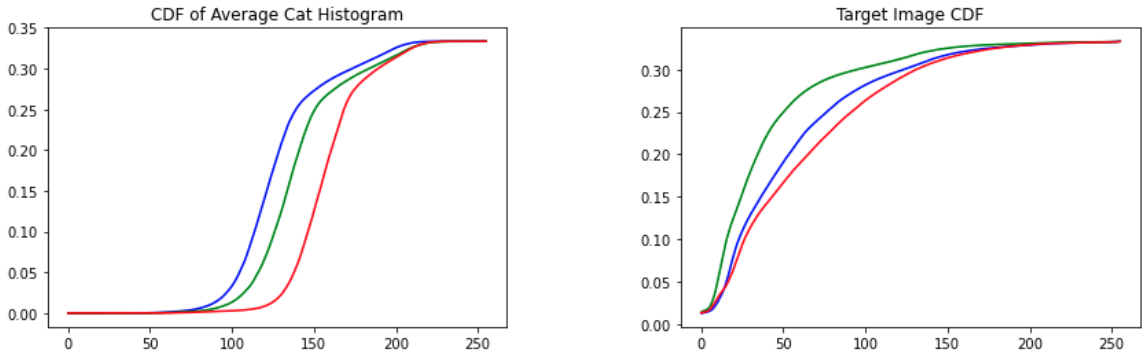
(ii) Histogram of the Target Image

Figure 1: Histograms

Before creating the lookup table (LUT), a function to calculate the CDF of histograms ("`cdf`") is defined. Then LUT (Figure 3) is created using the `compose_lut` function. The algorithm that composes the LUT is the same with the one that is provided in the lecture slides with a minor correction of the matrix indices.

pdf cdf and compose_lut Functions

```
1  def pdf(hist):
2      return hist / np.sum(hist)
3
4  def cdf(hist):
5      return np.cumsum(pdf(hist), 0)
6
7  def compose_lut(image_hist, target_hist):
8      img_cdf = cdf(image_hist)
9      trg_cdf = cdf(target_hist)
10     lut = np.empty([256,image_hist.shape[-1]])
11
12     for color in range(image_hist.shape[-1]):
13         g_trg = 0
14         for g in range(256):
15             while trg_cdf[g_trg, color] < img_cdf[g, color] and g_trg ↩
                   < 255:
16                 g_trg += 1
17             lut[g, color] = g_trg
18     return lut
```



(i) CDF of Cat Frames

(ii) CDF of the Target Image

Figure 2: Cumulative Distributions

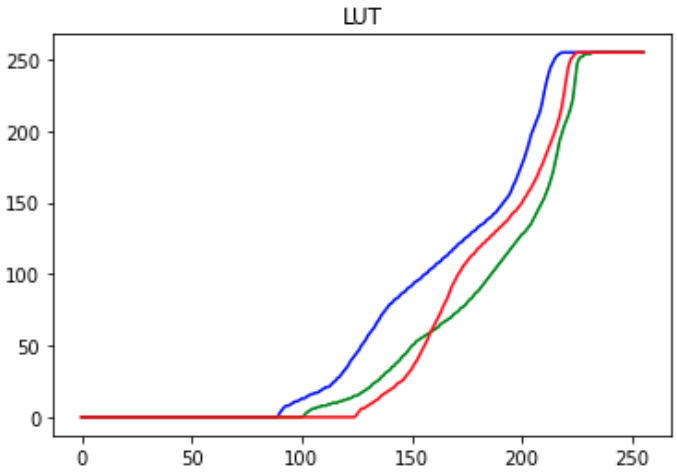


Figure 3: The Lookup Table

After composing the LUT, histogram matching between the cat on the right and the target image is done and an example frame is displayed.

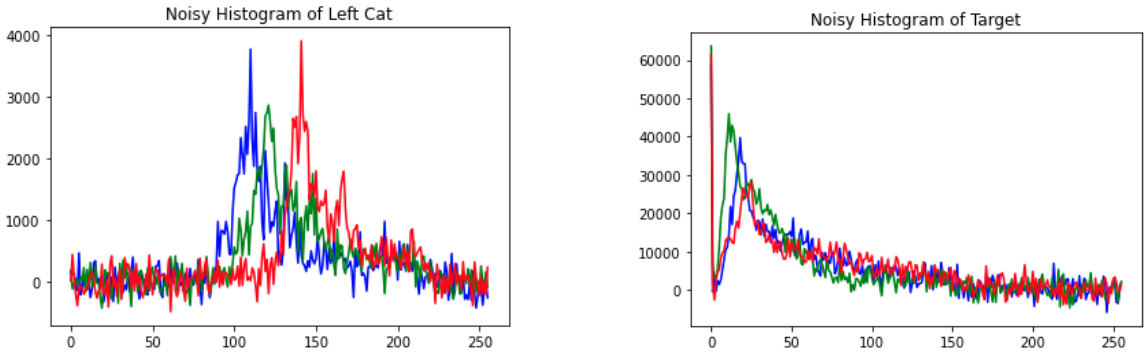
Histogram Matching

```
# ...
hist_matched_cat = np.empty(nz_cat_mirrored.shape)
for color in range(3):
    hist_matched_cat[:, color] = lut[nz_cat_mirrored[:, color], color]
```

Finally, the video is rendered using the same method as aforementioned.

Part 5: Disco Dancing

In this part, random noise added to the histogram of the current cat frame (Figure 4i for the cat on the left) and the target background image (Figure 4ii for the cat on the right) using `numpy`'s `random.normal` function with medians of 200 and 5000 respectively in order to achieve flashing frames (the disco effect). Please note that the addition of random noise and the target images for the cat on the left changes every frame. Therefore Figures 4i and 4ii are representative. Moreover, processing the frames of this part's video takes a substantial amount of time.



(i) Noisy Histogram of the Cat

(ii) Noisy Histogram of the Background

Figure 4: Noisy Histograms

Adding Noise to Target Histograms

```
1 l_hist = get_cat_hist(left_cat)
2 noisy_l_hist = l_hist + np.random.normal(0, 200, l_hist.shape)
3
4 target_hist, bins = get_hist(target_image)
5 r_hist = get_cat_hist(right_cat)
6 noisy_trg_hist = target_hist + np.random.normal(0, 2000, target_hist.↩
    shape)
```

Finally, the video is rendered using the same method as aforementioned.