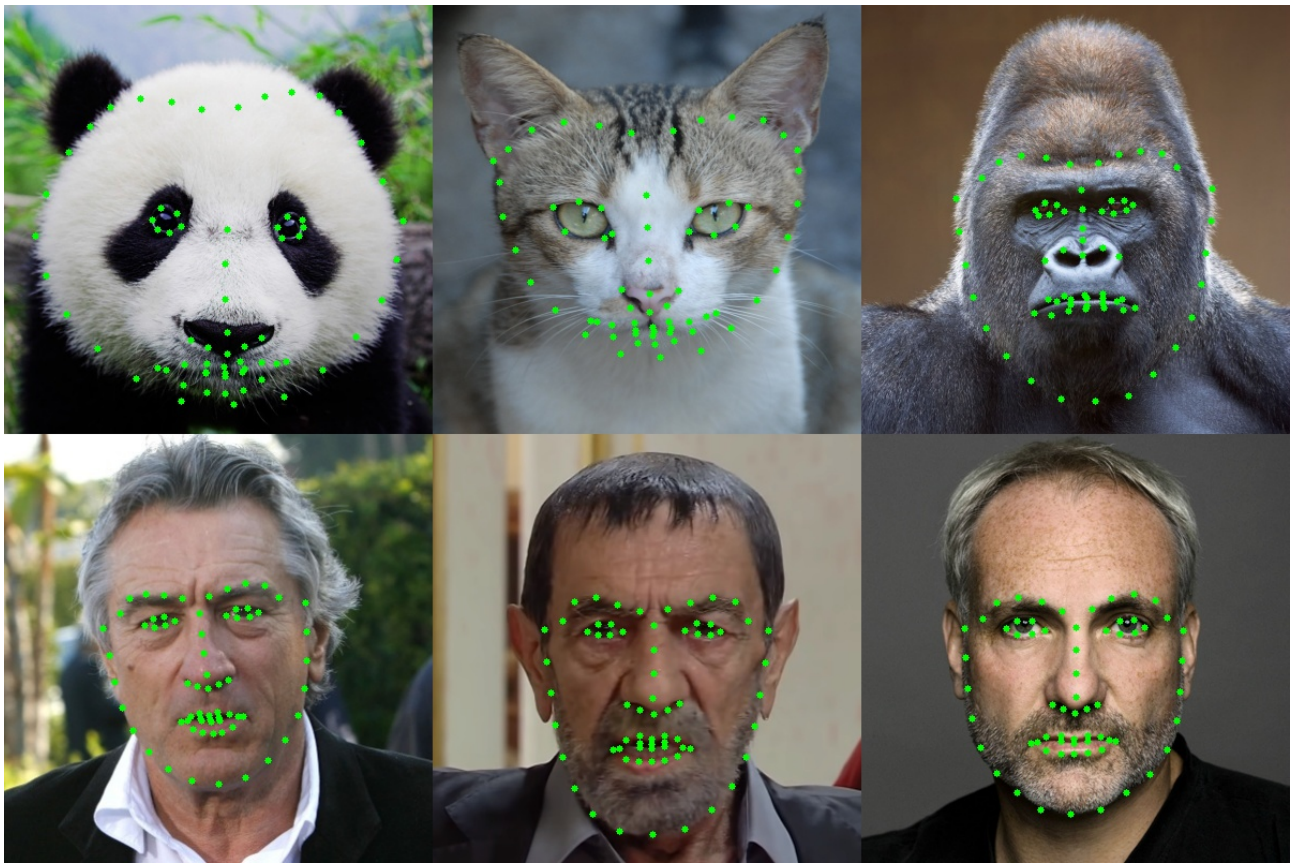
BLG 453E - Computer Vision
Homework II
Ekin Çelebi - 150160114

Part 1:

In this part of the homework we were required to show the landmark points of both humans and animals. For the humans we use a face detector model and for the animals we were given .npy files.

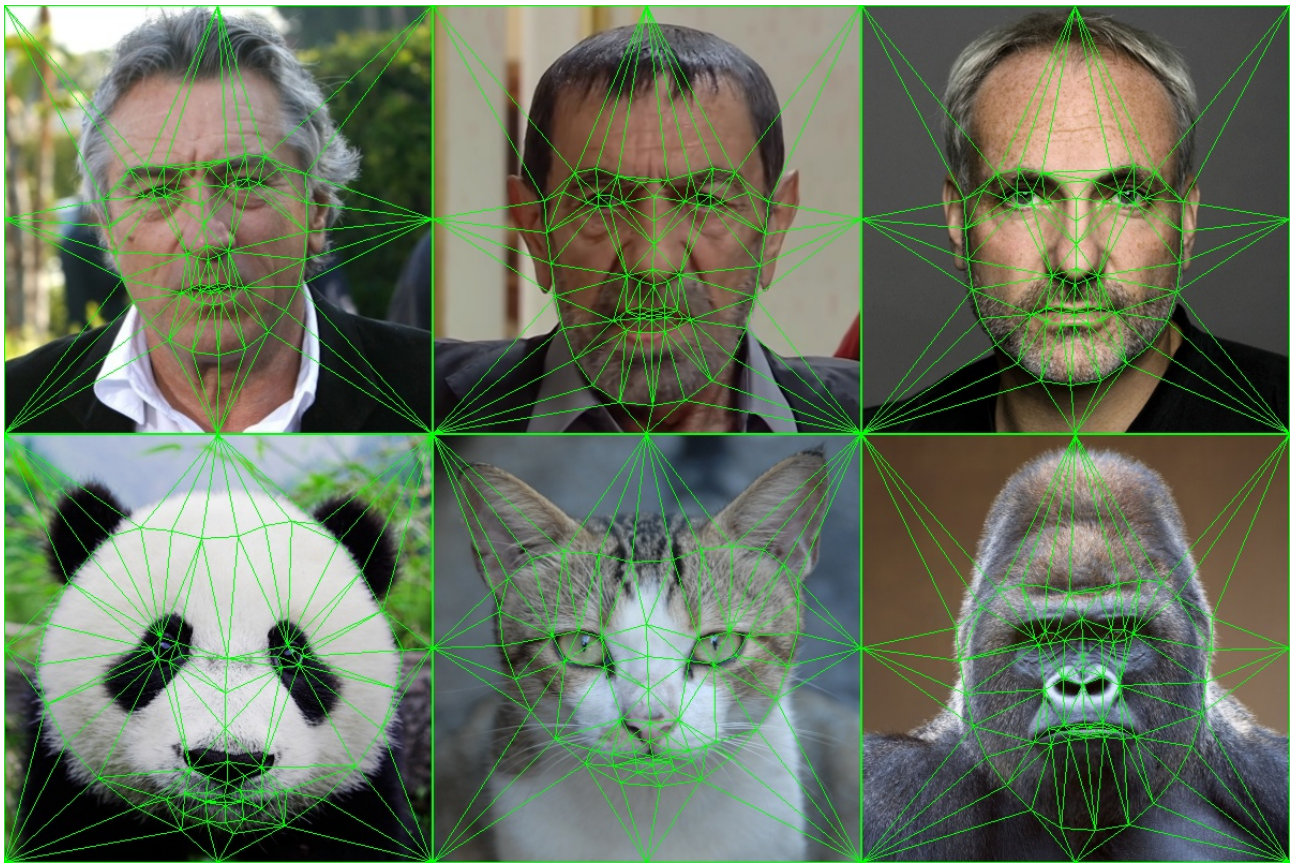
After the implementation I get following results on the faces.



Part 2:

In this part we were required to use Delaunay Triangulation. For this task we should use subdiv.
We would use the triangulation result for the next task. To able able to do the morphing task properly we have obtain two triangulation. The first triangulation is just a normal triangulation. To calculate the second images triangles we had to consider the corner id's of the first images then sort the second image accordingly.

After the implementation I get following results on the faces.



```
Transforms[i] = calc_transform(source, target) #(A)
```

A) In order to apply transformation at every iteration, a transformation matrix has to be determined considering the triangles.

```
for t in np.arange(0, 1.0001, 0.02): #(B)
    print("processing:\t", t*100, "%")
    morphs.append(image_morph(image, cat_img, img1_triangles,
    img2_triangles, Transforms, t)[:, :, ::-1])
```

B) The loop generates a smooth transition from source image to target image. “t” is given as an interpolation parameter to “image_morph()”. Inside the loop blended frames are appended into the “morphs” list. After morphing is done, the frames generates a video sequence

```
homogeneous = np.array([triangle[::2],   \
                        triangle[1::2],  \
                        [1, 1, 1]]) #(C)
```

C) Each triangle consist of x1, y1, x2,y2, x3,y3 values.The function creates rows for x and y values and adds a bottom row with ones to create a 3x3 square matrix.

```
Mtx = np.array([np.concatenate((source[0], np.zeros(3))), \
                np.concatenate((np.zeros(3), source[0])), \
                np.concatenate((source[1], np.zeros(3))), \
                np.concatenate((np.zeros(3), source[1])), \
                np.concatenate((source[2], np.zeros(3))), \
                np.concatenate((np.zeros(3), source[2]))]) #(D)

coefs = np.matmul(np.linalg.pinv(Mtx), target) #(E)

Transform = np.array([coefs[:3], coefs[3:], [0, 0, 1]]) #(F)
```

D) The set of suitable control points are determined. Following matrix is obtained and it will later be used for calculating image transformations.

$$\begin{bmatrix} x1 & y1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x1 & y1 & 1 \\ x2 & y2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x2 & y2 & 1 \\ x3 & y3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x3 & y3 & 1 \end{bmatrix}$$

E) From the control points transform coefficients are determined. Which has a size of 6x1.

F) We want to find a weighted average of the intensity values of the 4 neighboring points to copy the intensities to the target image. Creates a 3x3 transformation matrix from the coefficients matrix.

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{21} & C_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

G) We want to find a weighted average of the intensity values of the 4 neighboring points to copy the intensities to the destination. Here the weights of the 4 corners are determined.

```
    return np.uint8(np.round(top_left + top_right + bot_left + bot_right))
#(H)
```

H) The weighted average of the intensity values are returned. This averaging smoothens the transition.

```
    homo_inter_tri = (1 - t)*make_homogeneous(triangles1[i]) + t*
make_homogeneous(triangles2[i]) #(I)

    polygon_mask = np.zeros(image1.shape[:2], dtype=np.uint8)
    cv2.fillPoly(polygon_mask, [np.int32(np.round(homo_inter_tri[1::-1,
:].T))], color=255) #(J)

    seg = np.where(polygon_mask == 255) #(K)

    mask_points = np.vstack((seg[0], seg[1], np.ones(len(seg[0]))))#(L)
```

I) The outer loop iterates over triangles of both images This step takes the weighted average of the triangles of the same index.The target triangle gets more weight as the interpolation parameter increases.

J) A mask of the given image shape is created. The corner points of the homo_inter_triangle creates a triangle. The cells that fall inside the triangular area are assigned a value of 255.

K) Zero values are filtered

L) seg variable is the tuple that holds x and y values of the triangular masked area. At this step the tuple another row of ones. Briefly the masked triangular area form a matrice.

```
inter_tri = homo_inter_tri[:2].flatten(order="F") #(M)
```

M) homo_inter_tri matrix is vectorized and takes the form of (x1,y1,x2,y2,x3,y3)

$$[x_1 \quad y_1 \quad x_2 \quad y_2 \quad x_3 \quad y_3]$$

```
    mapped_to_img1 = np.matmul(inter_to_img1, mask_points)[:-1] #(N)
    mapped_to_img2 = np.matmul(inter_to_img2, mask_points)[:-1]

    inter_image_1[seg[0], seg[1], :] = vectorised_Bilinear(
mapped_to_img1, image1, inter_image_1.shape) #(O)
    inter_image_2[seg[0], seg[1], :] = vectorised_Bilinear(
mapped_to_img2, image2, inter_image_2.shape)

result = (1 - t)*inter_image_1 + t*inter_image_2 #(P)
```

N) Just before this step intermediate mapping functions are calculated. We need to bring our images to the same intermediate locations, that is the reason we warp the images. And here we calculate our coordinate parameters for warping.

O) At this point warping happens. We do not want double image effect instead we need blended images. By warping our points are aligned.

P) Take the weighted combinations of the images, with an interpolation parameter. In other words, the images are blended.