



Face Anonymization

- 14 Kushal Shah
- 16 Nazish Shaikh
- 17 Sidra Shaikh
- 50 Atharva Tayade
- 53 Pratik Thorave



Steps

- Face Detection
- Facial landmarks detection
- Finding face border
- Approximating nonlinear operations
- Finding triangles for image transformation
- Blending images together
- Video Stabilization



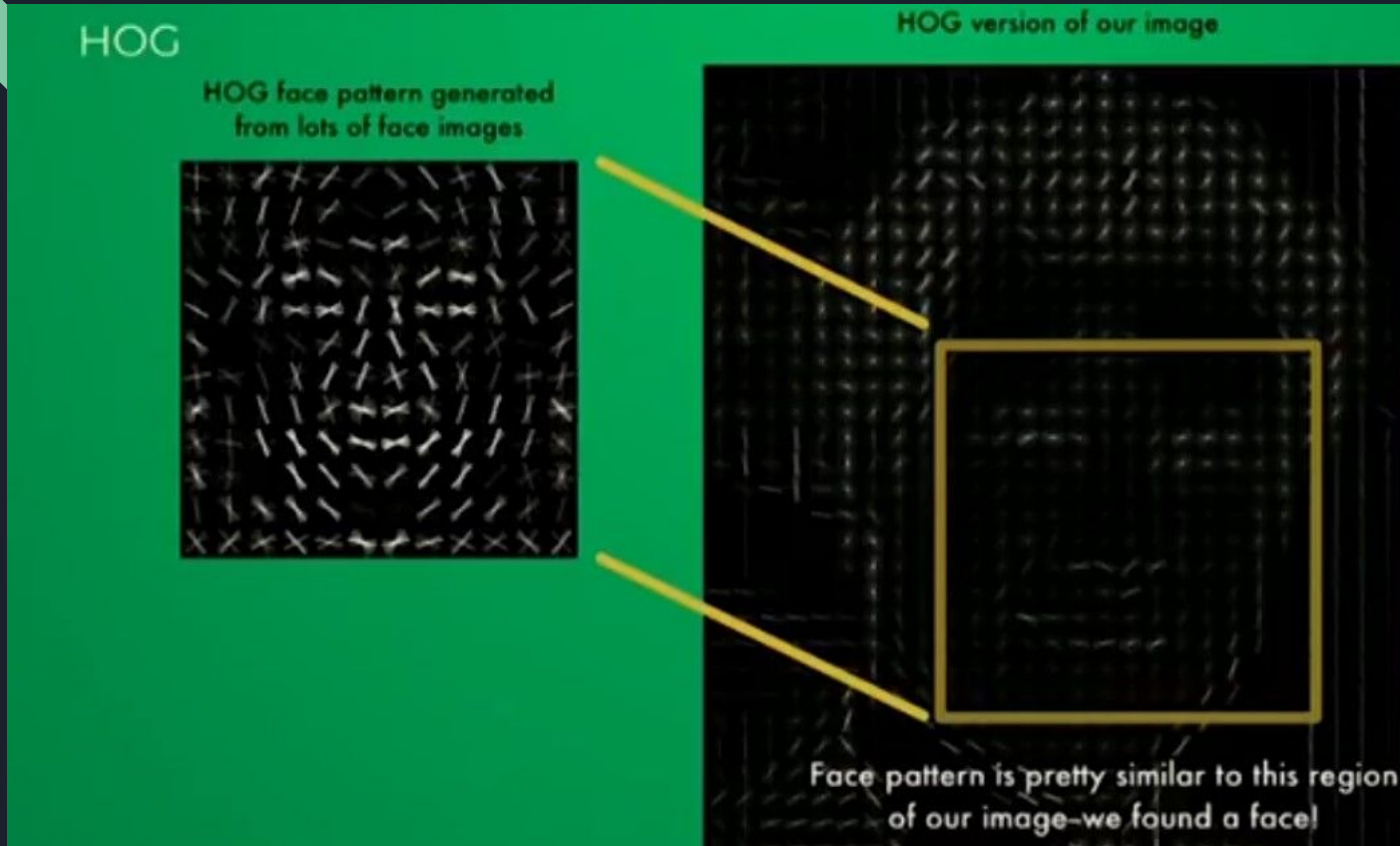
What is Face Anonymization?

Anonymize: remove identifying particulars or details from something for certain purposes.

Face anonymization refers to anonymizing faces to keep identity of the person's face confidential.

HOG

The Histogram of Oriented Gradients(HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection.





1. Face Detection :



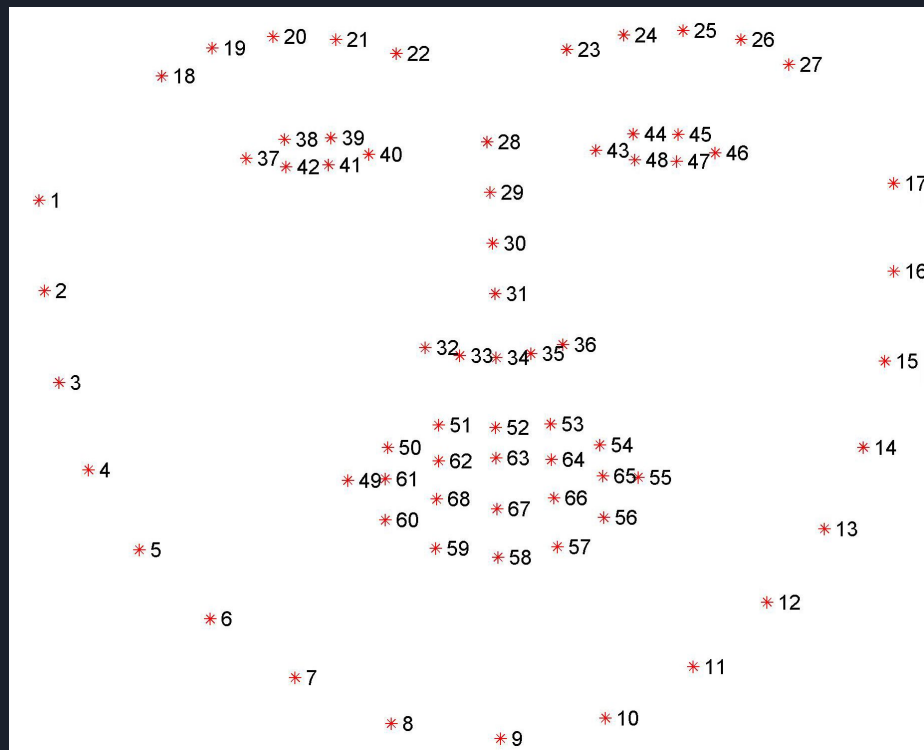
```
import cv2
import dlib

detector = dlib.get_frontal_face_detector()

img = cv2.imread('image.jpg')

rectangles = detector(img)
```

2. Face Landmarks Detection



- dlib Library
- 68 facial landmarks

Face Landmarks Detection

One millisecond Face Alignment



```
import cv2
import dlib

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

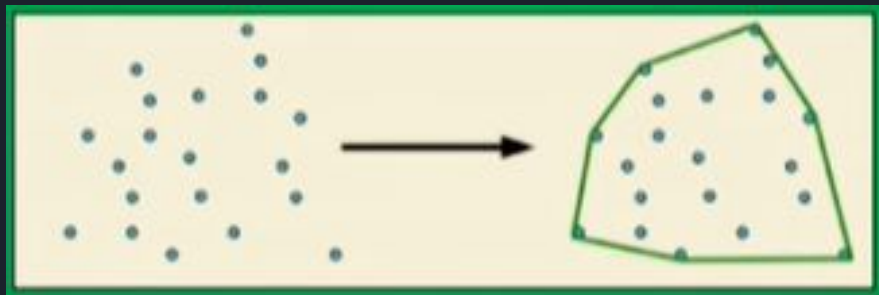
img = cv2.imread('image.jpg')

rectangles = detector(img)

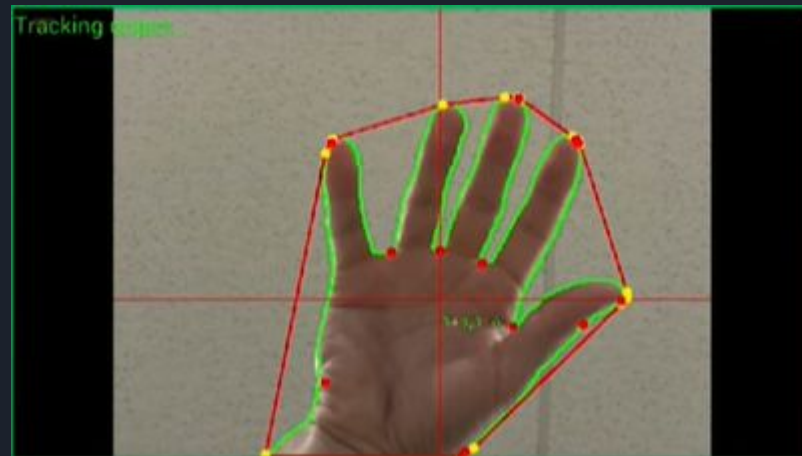
face = max(rectangles, key=lambda r: r.area())
landmarks = predictor(img, face)
```

3. Find face border

Convex hull



Convex hull vs Contour





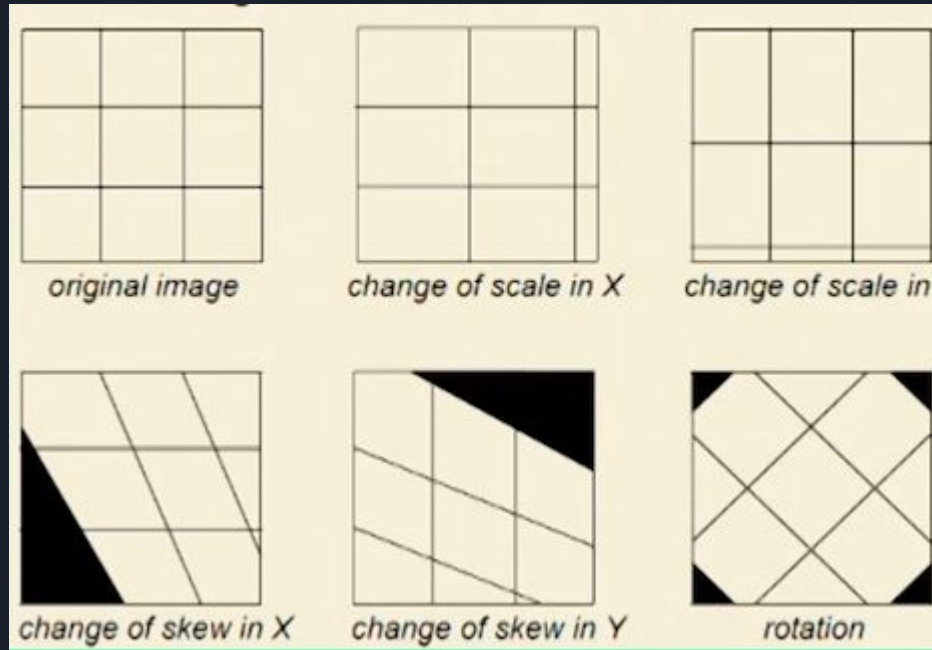
Face Border

Convex hull

```
...  
  
landmarks = predictor(img, face)  
  
points = [(p.x, p.y) for p in landmarks.parts()]  
  
hull = cv2.convexHull(np.array(points), returnPoints=False)
```

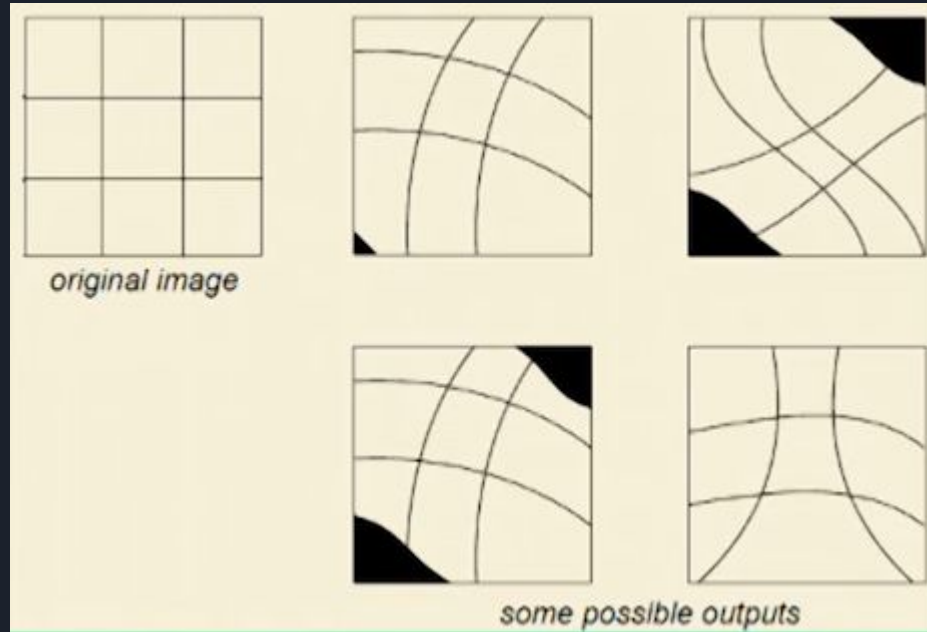
Approximating non-linear operations with linear ones

- Affine Transformation




Approximating non-linear operations with linear ones

- Non-Linear Transformation



We'll be using triangles.



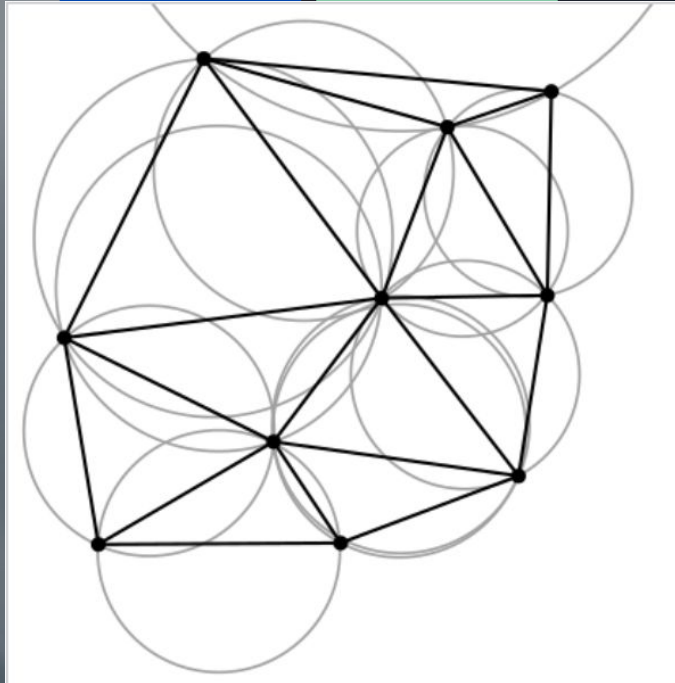
Approximating non-linear operations with linear ones

There is no such operation in OpenCV. So to implement -

Steps:

1. Get coordinates of input triangle and output triangle.
2. Find bounding rectangles around those triangles with `cv2.boundingRect()`.
3. Crop out these bounding rectangles out of the image.
4. Create a mask with 1.0 inside triangle, and 0.0 outside.
5. `getAffineTransform()` returns matrix of how to transform from triangle 1 to triangle 2
6. `warpAffine()` uses this matrix to transform bounding rect 1 to bounding rect 2
7. Multiply output of `warpAffine` with mask (inside triangle (1.0) stays, outside (0.0) is out)
8. Put the warped fragment back into image.

4. Finding Triangles



Find Delaunay Triangulation on first image

Find corresponding triangles on second image

Transform every triangle from first image to second image

```
...  
mouth_points = [[60], [61], [62], [63], [64], [65], [66], [67]]  
  
hull = np.concatenate([hull, mouth_points_indexes])  
  
rect = (0, 0, img.shape[1], img.shape[0])  
subdiv = cv2.Subdiv2D(rect)  
  
subdiv.insert(hull)  
  
triangles = subdiv.getTriangleList()
```



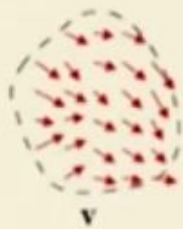
5. Blending two images

Seamless Poisson cloning

- Given vector field \mathbf{v} (pasted gradient), find the

$\min_f \iint_{\Omega} |\nabla f - \mathbf{v}|^2$ with $f|_{\partial\Omega} = f^*|_{\partial\Omega}$ optimize:

Pasted gradient



Mask



In this method, we don't copy the values of pixels, just copy the gradient.

```
dest = cv2.imread("image1.jpg")

source = cv2.imread("image2.jpg")

mask = np.zeros(source.shape[:2], dtype=np.float32)
rect = [(23, 25), (55, 112)]
mask = np.uint8(cv2.rectangle(mask, *rect, (1.0, 1.0, 1.0), -1) * 255)

center = 641, 395

cloned = cv2.seamlessClone(source, dest, mask, center,
cv2.MIXED_CLONE)
```




6. Stabilization

- Stability is one of the most important part.

To avoid this shakiness Optical Flow with Lucas-Kanade Method is used.

Equation:-

$$(u,v) = (dx/dt, dy/dt)$$

Assumptions:-

- 1) Brightness Constancy Assumptions .
- 2) Additional Assumption :- Lucas-Kanade Method.

Eg:-





Code for Optical Flow :-

```
hull_next, *_ = cv2.calcOpticalFlowPyrLK(img_gray_prev, img_gray, hull_prev, hull)
```



THANK YOU