Image and Video Analysis Exam

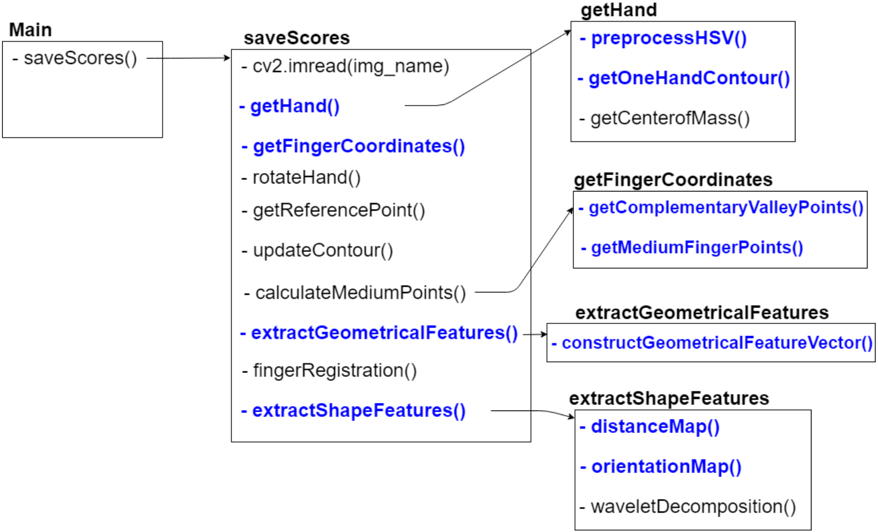
Identity verification through hand shape and geometry

Emanuele Vivoli
Francesca Del Lungo

University of Florence, Italy

May 2019

Code structure



Exercises for Thursday Lab

- preprocessing (with step by step images)
- find fingers (contour, convex hull, defects, finger points)
- construct 21 geometrical features from 7 distances
- distance and orientation map
- test all with your hands

Lab exercise 1

Image preprocessing

Preprocessing

Steps:

1. BGR image;
2. reshape image (cv2.resize);
3. changing color space to HSV;
4. median blur with kernel size 3 to each HSV channel;
5. segmentation of image through Gaussian mixtures (with 2 gaussian);
6. open with (10 x 10) ellipse kernel;

Preprocessing

The function to be created is:

img_bin = preprocessingHSV (img_bgr)

Preprocessing

```
def preprocessingHSV(img_bgr):

    img = cv2.resize(img_bgr, None, fx=0.5, fy=0.5)

    img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    b,g,r = cv2.split(img)
    shape = b.shape

    # Smooth the three color channels one by one
    b = cv2.medianBlur(b,5)
    g = cv2.medianBlur(g,5)
    r = cv2.medianBlur(r,5)
```

Preprocessing

```
num_clusters = 2
# Warning: X is 3xNum_pixels. To fit the kmeans model X.T should be used
X = np.array([b.reshape(-1), g.reshape(-1), r.reshape(-1)])
gmm=GaussianMixture(n_components=num_clusters,
                    covariance_type='full',
                    init_params='kmeans',
                    max_iter=300, n_init=4, random_state=10)
gmm.fit(X.T)

Y = gmm.predict(X.T)

mask_img = copy.deepcopy(b.reshape(-1))

unique, counts = np.unique(Y, return_counts=True)
dic = dict(zip(unique, counts))
```

Preprocessing

```python
    # define foreground and background
    if dic[0] > dic[1]:
        mask_img[ Y==0 ] = 0
        mask_img[ Y==1 ] = 1
    else:
        mask_img[ Y==0 ] = 1
        mask_img[ Y==1 ] = 0

    mask_img = mask_img.reshape(shape)

    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(10, 10))
    img_bin = cv2.morphologyEx(mask_img, cv2.MORPH_OPEN, kernel)

    return img_bin
```

Lab exercise 2

Find finger, valley, complementary and medium points

Finger, valley, complementary and medium points

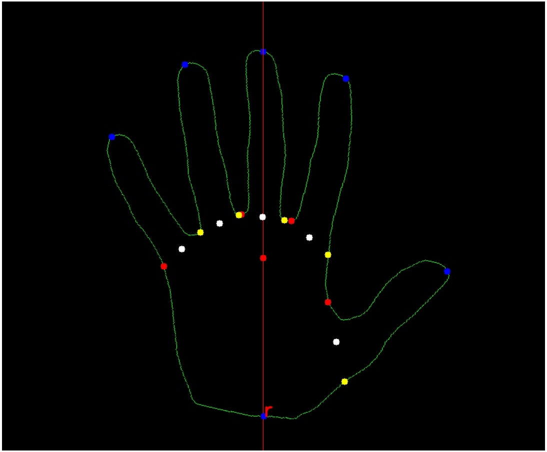


Figure: finger points, valley points, complementary valley points and white medium points

Find finger, valley, complementary and medium points

Steps:

1. find contour of the hand mask;
2. find convex hull;
3. find convexity defects;
4. select only the 4 most important defects;
5. define finger points and valley points as peaks and valleys (pay attention to the finger points);
6. find reference point r with the given function *getReferencePoint*;
7. rotate hand mask with the given function *rotateHand*;
8. find complementary valley points;
9. calculate medium points.

Find fingers (find contours) I

Open CV has the function:
image, contours, hierarchy = cv2.findContours(img_binary, mode, method)
that retrieves contours from a binary image (source image is not modified).
Parameters:

- img_binary: binary image;
- mode: mode of the contour retrieval algorithm;
- method: contour approximation method.

Returns:

- image: binary image (same as input);
- contours: detected contours. Each contour is stored as a vector of points;
- hierarchy: containing information about the image topology;

Find fingers (find contours) II

Example of different contour approximation methods:

cv2.CHAIN_APPROX_NONE
cv2.CHAIN_APPROX_SIMPLE

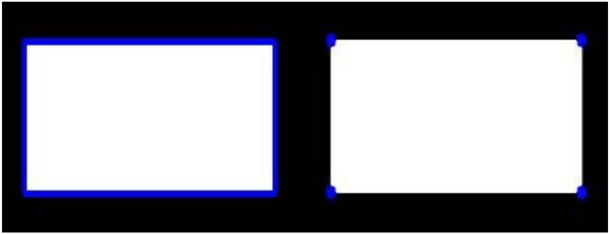


Figure: Result with NONE method on the left, SIMPLE on the right.

Find fingers

cv2.findContours function returns a list of contour elements of all the contours found in the image:

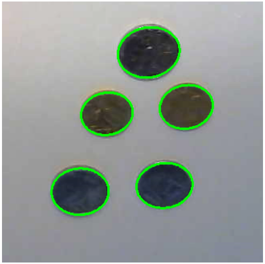


Figure: Example of found contours

So, to find the contour that defines the hand mask, you have to select the contour with the biggest area.

Find fingers

You can do it with the Open CV function:

area = cv2.contourArea(contour)

Parameters:

- area: area of the selected contour;
- contour: component of the contour whose area to be calculated.

Find fingers

The functions to be created are:

contour = getOneHandContour (img_binary)
that takes the binary image and gives as a result the contour of the hand.

hand_mask, contour, center_of_mass, None = getHand (img_bgr)
that performs the *preprocessingHSV()* function, than uses *getOneHandContour()* to get the hand contour and cv2.fillPoly() to create the hand mask and finally find the center of mass.

Find contours (code)

```
def getOneHandContour(img_binary):
    # find contours of processed image
    contours_bin, hierarchy_bin = cv2.findContours(img_binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    # we have more than one contours in some image
    # than we need to consider only hand contour, it is
    # the biggest one
    area_max = 0
    index_area_max = -1
```

Find contours (code)

```
    # select biggest contour
    for i in range(len(contours_bin)):

        # get area of ith contour
        area_i = cv2.contourArea(contours_bin[i])

        # check if dimension of area is bigger than the
 biggest
        if area_max < area_i:
            index_area_max = i
            area_max = area_i

    return contours_bin[index_area_max]
```

Find centroid (code)

```
def getHand(img_bgr):

    # preprocessing with gaussian mixture
    img_binary = preprocessingHSV(img_bgr)

    # create an empty black image
    hand_mask = np.zeros((img_binary.shape[0],
    img_binary.shape[1]), np.uint8)

    #select contour with biggest area
    contour = getOneHandContour(img_binary)

    # create polylines from contour points, and draw if
     filled in image
    cv2.fillPoly(hand_mask, pts =[contour], color=(255)
    )
    img_binary = hand_mask.copy()
```

Find centroid (code)

```python
    # create a binary mask, where img is white(255) put 1, else let 0
    img_binary[img_binary == 255] = 1

    # skimage regionprops need labeled mask (binary mask)
    # and return properties object with pixels property like
    # centroid, ellipse around pixels, ecc...
    properties = regionprops(img_binary, coordinates='xy')

    # get center of mass of pixels (also called centroid)
    center_of_mass = properties[0].centroid[::-1]

    return hand_mask, contour, center_of_mass, None
```

Find fingers (find convex hull)

OpenCV function convexHull finds the convex hull of a point set.

hull=cv2.convexHull(points, clockwise=True, returnPoints = False)

Parameters:

- points: input 2D point set;
- clockwise: orientation flag. If it is true, the output convex hull is oriented clockwise. The assumed coordinate system has its X axis pointing to the right, and its Y axis pointing upwards. (We use clockwise orientation with opposite coordinate direction of y axis);
- returnPoints: when the flag is true, the function returns convex hull *points*. Otherwise, it returns *indices* of the convex hull points.

Returns:

- hull: output convex hull (numpy.ndarray);

Find fingers (find convexity defects)

Find the convexity defects of a contour consists on finding internal convex hull points (belonging to contour) that have the biggest distance to convex hull margin.

defects = cv2.convexityDefects(contour, hull) Parameters:

- contour: input contour;
- hull: convex hull obtained using convexHull() function.

Returns:

- defects: output vector of convexity defects. Each convexity defect is represented as 4-element integer vector. (start_index, end_index, farthest_pt_index, fixpt_depth), where indices are 0-based indices in the original contour of the convexity defect beginning, end and the farthest point,

Find convexity defects

The function to be created is:

defects = getImportantDefect (contour , hull)

that finds convexity defects and than the most important defects: the ones with maximum distance from their farthest point. Finally reorder the 4 defects found in clockwise direction starting from little point.

Find convexity defects (code)

```python
def getImportantDefect(cnt, hull):

        # find convexityDefects
        defects = cv2.convexityDefects(cnt, hull)


        # first defect become the last one
        defects = np.concatenate((defects[1::],[defects[0]]))

        # find segments at maximum distance from the relative depth points (d), these correspond to the segments between the fingers (fingertips)
        defects = [[list(elem[0][:]), i] for i, elem in enumerate(defects)]
```

Filter convexity defects (code)

```
    # sort in descending order elements of defects list.
    defects.sort(key = lambda x: x[0][3], reverse=True)

    # consider only the 4 segments
    defects = defects[:4]

    defects.sort(key = lambda x: x[1], reverse=True)

    defects = [ elem[0] for elem in defects]

    return defects
```

Find fingers

The function to be created is:

finger_points , valley_points ,fingers_indexes , valley_indexes = getFingerCoordinates(contour , img_binary)

that finds the convex hull and than use *getImportantDefect()* to select the considered defects. From these defines valley points and finger points.

Find contours and centroid (code)

```python
def getFingerCoordinates(cnt, img_binary):

        # convex hull of the binary image
        hull = cv2.convexHull(cnt, clockwise=True, returnPoints = False)


        # obtain the 4 important defects from contour and hull of hand image
        defects = getImportantDefect(cnt, hull)


        # initialize empty lists
        all_fingers_indexes = []
        valley_points = []
        valley_indexes = []

        font = cv2.FONT_HERSHEY_SIMPLEX
```

Find contours and centroid (code)

```
    for i in range(len(defects)):
        # get indexes of important points from defects
        s,e,f,d = defects[i]

        # update coordinates of valley points, that are points between each pair of fingers
        valley_points.append(list(cnt[f][0]))
        valley_indexes.append(f)

        # get points
        start = tuple(cnt[s][0])
        end = tuple(cnt[e][0])
        far = tuple(cnt[f][0])

        all_fingers_indexes.append(e)
        all_fingers_indexes.append(s)
```

Find contours and centroid (code)

```
    # find one representative point for each fingertips (if a fingertips had two points the final point is calculated as the middlepoint)
    fingers_indexes = findFingerIndexesSimple(len(cnt), all_fingers_indexes)

    # print('contour: ',len(cnt))
    # print('fin_idx: ',fingers_indexes)
    finger_points = cnt[fingers_indexes]

    return finger_points, valley_points, fingers_indexes, valley_indexes
```

Find fingers (rotate hand mask)

GetAngle function returns the angle between segment from point p to point m and the vertical axis (y) passed by m point.

phi = getAngle(p, m)

Parameters:

- p: first point considered, given as (x,y) coordinates;
- m: second point considered, given as (x,y) coordinates.

Returns:

- phi: angle (rad) returns angle in the 1 and 4 quadrant: [-$\frac{\pi}{2}$, $\frac{\pi}{2}$];

Find fingers (rotate hand mask)

hand_mask_rotated, finger_points, valley_points, contour, center_of_mass = rotateHand(shape, contour, angle, center_of_mass, fingers_indexes, valley_indexes)

Parameters:

- shape: shape of the hand mask;
- contour: contour that has to be updated;
- angle: angle that defines the rotation (rad);
- center_of_mass, fingers_indexes, valley_indexes: indexes and coordinates that have to be updated.

Returns:

- hand_mask_rotated: new hand mask with rotated contours;
- finger_points, valley_points: new (x,y) coordinates of each finger and valley point respectively;
- center_of_mass: new (x,y) coordinates of the center of mass;

Find fingers (get reference point)

Coordinates of reference point are calculated as the point of intersection between the line passing by center of mass and middle finger point and the contour on the opposite side of the hand.

r_point, r_index = getReferencePoint(contour, fingers_indexes, center_of_mass)

Parameters:

- contour: contour of the hand mask;
- fingers_indexes: contour indexes of the 5 finger points;
- center_of_mass: (x,y) coordinates of the center of mass.

Returns:

- r_point: (x,y) coordinates of reference point;
- r_index: contour index of the reference point;

Find fingers

The functions to be created are:

complementary_valley_indexes, valley_indexes = getComplementaryValleyPoints(cnt_length, valley_indexes, fingers_indexes)

that finds complementary valley points in counter-clockwise order.

medium_points = getMediumFingerPoint(valley_points, complementary_valley_points)

that calculates the coordinates of the medium point of each finger as the middle point between valley and complementary valley.

Find complementary valleys (code)

```python
def getComplementaryValleyPoints(cnt_length, valley_indexes, fingers_indexes):

    valley_indexes.insert(0, valley_indexes[0])

    complementary_valley_indexes = [( 2 * p_index - v_index ) % cnt_length for v_index, p_index in zip(valley_indexes[::-1], fingers_indexes[::-1])]

    app = complementary_valley_indexes[-1]
    complementary_valley_indexes[-1] = valley_indexes[0]
    valley_indexes[0] = app
    complementary_valley_indexes = complementary_valley_indexes[::-1]
```

Find complementary valleys (code)

```
    all_valley = []
    all_valley.append(valley_indexes[0])
    for x, o in zip(valley_indexes[1:], complementary_valley_indexes[:-1]):
        app = [x, o]
        app.sort(key = lambda x: x, reverse= True)
        all_valley.append(app[0])
        all_valley.append(app[1])

    all_valley.append(complementary_valley_indexes[-1])

    valley_indexes = all_valley[0::2]
    complementary_valley_indexes = all_valley[1::2]

    return complementary_valley_indexes, valley_indexes
```

Find medium points (code)

```python
def getMediumFingerPoint(valley_points, complementary_valley_points):
    medium_points = []

    for v_point, c_v_point in zip(valley_points, complementary_valley_points):
        app = [v_point, c_v_point]
        medium_points.append(np.mean(app, axis = 0).tolist())

    return medium_points
```

Lab exercise 3

Construct geometrical features

Geometrical features (distance calculation)

The geometrical features of one hand image are based on the 5 distances calculated between the finger point and medium point of each finger. Together with these 5 distances we need two other ones in order to have geometrical features of one hand image.

The two other distances are:

- distance between medium point of little finger and medium point of index finger
- distance between medium point of index finger and medium point of thumb finger

Geometrical features (distance calculation) II

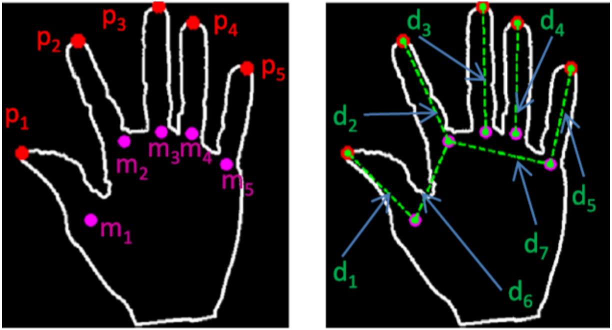


Figure: Distances calculated between finger points and medium points and the other two distances based on palm size. Pay attention: we are working with right hands (always with the palm upside), not left ones.

Geometrical features (build them)

Geometrical features are calculated as distances ratio between one distance and all the following ones.

The first 5 distances are calculated starting from thumb finger (that is the last element of finger_points).

From 7 distances we get 21 geometrical features.

Geometrical features

The functions to be created are:

geom_feature = constructGeometricalFeatureVector(distances)

that calculates, for all the distances, the difference between the distance considered and all the subsequent ones.

distances, geom_feature = extractGeometricalFeatures(finger_points, medium_points)

that creates distances between each finger point and medium point and then uses *constructGeometricalFeatureVector* function to calculate the vector of geometrical features.

Geometrical features (code)

```python
def extractGeometricalFeatures(finger_points, medium_points):

    distances = []
    # distances are calculated starting from thumb finger
    for i in range(4,-1, -1):
        # print(i, finger_points[i][0], medium_points[i])
        dist = math.hypot(finger_points[i][0][0] - medium_points[i][0][0], finger_points[i][0][1] - medium_points[i][0][1]) # Linear distance
        distances.append(dist)
```

Geometrical features (code)

```
    # dist_6 : distance between 3-medium_point (index finger) and 4-medium_point (thumb finger)
    x_med_point_3, y_med_point_3 = medium_points[3][0][0], medium_points[3][0][1]
    x_med_point_4, y_med_point_4 = medium_points[4][0][0], medium_points[4][0][1]

    dist_6 = math.hypot(x_med_point_3 - x_med_point_4, y_med_point_3- y_med_point_4)
    distances.append(dist_6)

    # dist_7 : distance between 3-medium_point (index finger) and 0-medium_point (little finger)
    x_med_point_3, y_med_point_3 = medium_points[3][0][0], medium_points[3][0][1]
    x_med_point_0, y_med_point_0 = medium_points[0][0][0], medium_points[0][0][1]
```

Geometrical features (code)

```
    dist_7 = math.hypot(x_med_point_3 - x_med_point_0,
    y_med_point_3- y_med_point_0)
    distances.append(dist_7)

    # calculate geometrical feature vector
    geom_feature = constructGeometricalFeatureVector(
    distances)

    return distances, geom_feature
```

Geometrical features (code)

```python
def constructGeometricalFeatureVector(distances):

    geom_features = []
    for i in range(len(distances)):
        for j in range(i+1, len(distances)):
            geom_features.append(distances[i]/distances[j])

    return geom_features
```

Lab exercise 4

Construction of Distance and Orientation maps

Fingers registration I

First of all, to generate distance and orientation maps, fingers have to be aligned: all hands must have the same fingers in the same orientation.

This can be done with *fingerRegistration* function:

Fingers registration II

contour_updated = fingerRegistration(contour, center_of_mass, p_list, m_list, c_list, v_list)

Parameters:

- contour: original contour of the hand mask;
- center_of_mass: center of mass (x,y) coordinates;
- p_list: list of finger points coordinates;
- m_list: list of medium points coordinates;
- c_list: list of indexes of complementary valley points of the 5 finger points;
- v_list: list of indexes of valley points of the 5 finger points.

Returns:

- contour_updated: updated contour with fingers in the right orientation;

Distance map

To generate a distance map you have to calculate the distance from each point of the contour and the reference point r, in clockwise direction.

$$dp(i) = \sqrt{(x'_r - x'_{{b_r}_{\mathrm{cw}_{(\mathrm{i})}}})^2 + (y'_r - y'_{{b_r}_{\mathrm{cw}_{(\mathrm{i})}}})^2} \tag{1}$$

Orientaton map

To generate a orientation map you have to calculate the angle created between the line passing from each point of the contour and the reference point r.

And the horizontal axis is taken as zero angle, having the vertical lowest point as 90 positive degrees, and upper extreme point as -90 degree.

$$op(i) = 90 + tan^{-1}\left(\frac{y'_r - y'_{b_r \text{cw}_{(\text{i})}}}{x'_r - x'_{b_r \text{cw}_{(\text{i})}} + \sigma}\right) \tag{2}$$

Distance and Orientation map

numpy.arctan2(x1, x2 [, ...])
Element-wise arc tangent of x1/x2 choosing the quadrant correctly.
Parameters:

- x1 : array_like; real-valued y-coordinates.
- x2 : array_like; real-valued x-coordinates.
- ...

Returns:

- angle : ndarray; Array of angles in radians, in the range [-pi, pi]. This is a scalar if both x1 and x2 are scalars.

Distance and Orientation map

numpy.rad2deg(x [, ...])
Convert angles from radians to degrees.
Parameters:

- x : array_like; Angle in radians.

Returns:

- y : ndarray; The corresponding angle in degrees. This is a scalar if x is a scalar.

Geometrical features

The functions to be created are:

distance_map = distanceMap (contour , r_idx)
that calculates for each point of the contour the distance between that point and the reference point whose index is r_idx. Than the list of distances has to be normalized to have distances in [0,1].

orientation_map = orientationMap (contour , r_idx)
that calculates, for each point of the contour, the angle (given in radians) between the line passing by the considered point and the reference point r and the horizontal axis.

Distance map (code)

```
def distanceMap(cnt, r_idx):

    distance_map = []
    for point in cnt:
        d_value = math.sqrt((cnt[r_idx][0][1] - point[0][1])**2 + (cnt[r_idx][0][0] - point[0][0])**2)
        distance_map.append(d_value)

    return distance_map
```

Orientation map (code)

```
def orientationMap(cnt, r_idx):
    orientation_map = []

    for i in range(len(cnt)):
        point = cnt[(r_idx + i)%len(cnt)]
        if cnt[r_idx][0][1] < point[0][1]:
            point[0][1] = cnt[r_idx][0][1]

        o_value = np.arctan2([cnt[r_idx][0][1] - point[0][1]],[cnt[r_idx][0][0] - point[0][0]])

        orientation_map.append(np.rad2deg(o_value))

    return orientation_map
```

Lab exercise 5

Test all with your hands

Test all with your hands

Steps:

1. remove from the *./hands/dataset* folder all the images except the ones of 5/6 people;
2. do the same thing with the images in the *./tests/hands/dataset* folder (if a person is in the test folder it must be also in the training folder);
3. take 6 pictures of your hand with your smartphone (as shown in the picture below);
4. put 5 of them in the *./hands/dataset* folder (these will be part of our database);
5. put the sixth image in the test folder.

Test all with your hands (how to take pictures)

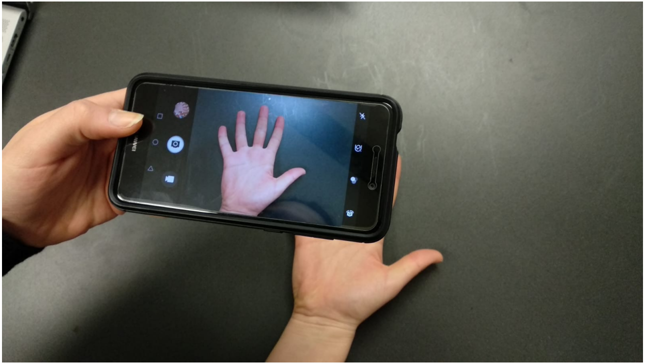
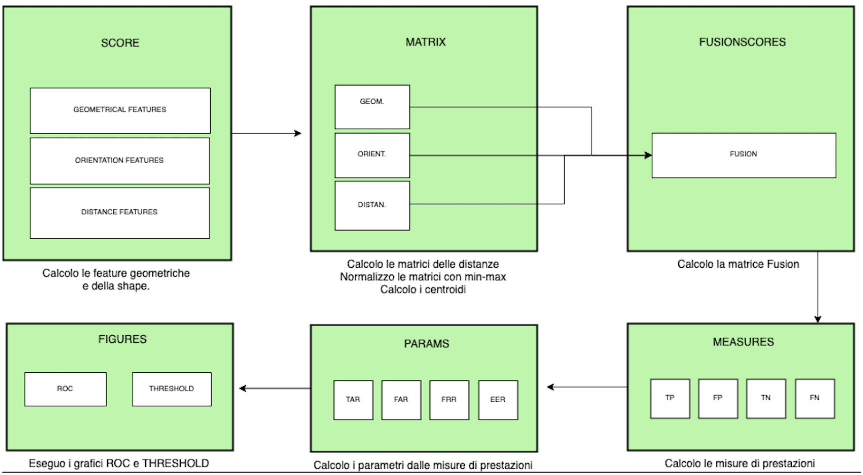


Figure: Take your photos horizontally, stop at the wrist, do not include the arm. The background can be light or dark.

Remaining code diagram



Thanks for your attention!