Report--Face Recognition

Contents

1. Given Tasks 1

1.1. Problem description 1

1.2. Details 1

2. Knowledge Preparation 2

3. Understanding of the Given Codes..5

3.1. Face_Detection 5

3.2. Smile_Detection..8

3.3. Real_Time_Detection 12

4. Simplification of the Codes 14

1. Given Tasks

1.1. Problem Description

Give you a picture of a person, could you tell me whether he/she is smiling? Please make your computer to learn smile detection.

Is she smiling now? Let your computer answer this question.

1.2. Details

1.Face detection. (5’)

Using any face detector tool to detect faces in those images.

2.Smile classification. (15’)

Using face as input, train a model by yourself, and test your model. Choose the best one for classification. Here, you can choose HOG or LBP or both as features, and you could also use other features if you like, for example, SIFT. You can use SVM as classification algorithm, and you could also use other algorithms if you like. We encourage different ways to solve the same problems. We also encourage some groups to compare different ways. Smile classification is the most important one for this project. We will test your model.

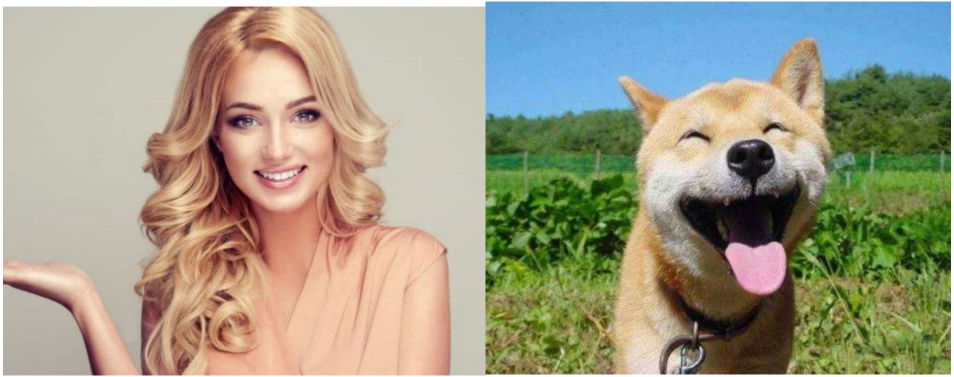
3. Your own real-time Smile Detection application using camera (5')

After you have trained your own model, we hope you can use it. For example, could you please open your camera on your computer and detect whether you are smiling real-time?

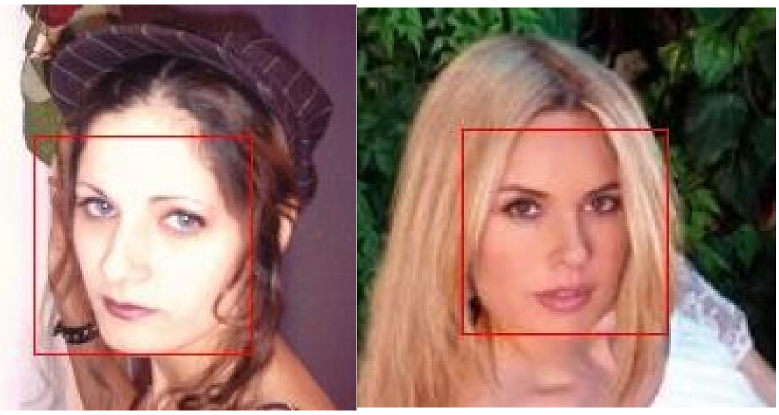
2. Knowledge Preparation

To solve the given tasks, we need to know some basic concepts:

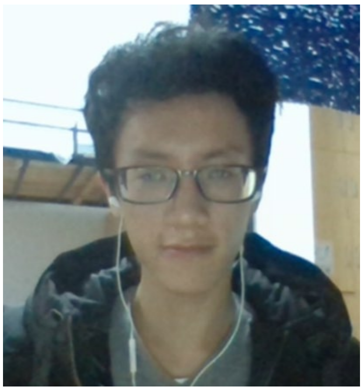
How to detect the first picture contains a human face and the second one doesn't?



After we find the human face, how to determine whether he or she is smiling or not?



How can we use the computer camera to solve the similar problems?



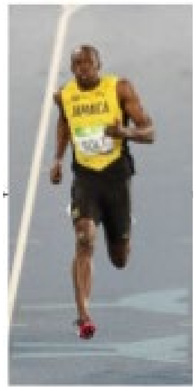
With the help of our kind teacher and teaching assistants, we know what we should know: to get the feature of a face and a smile, then we can calculate the feature of the given picture and determine what we want.

Basic tool: OpenCV

OpenCV, a cross-platform computer vision library, is worldwide used in the area of image processing and computer vision. This is the fundamental tool we use to solve the tasks.

Chosen feature: Histogram of Oriented Gradients(HOG)

To get the feature, a feature descriptor like HOG simplifies the image by extracting useful information and throwing away extraneous information. Let us consider a picture with the ratio 1:2.

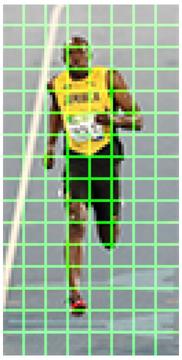


Step1, to calculate a HOG descriptor, we need to calculate the horizontal and vertical gradients; after all, we want to calculate the histogram of gradients. This is easily achieved by filtering the image with the following kernels.

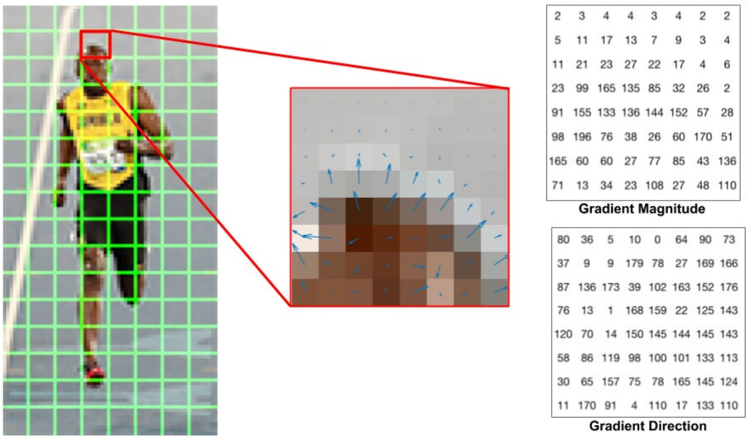
-1	0	1

-1
0
1

Step2, calculate Histogram of Gradients in 8×8 cells like the following picture.



Exactly 8×8 cells is great enough. The way to calculate is mentioned bellow.



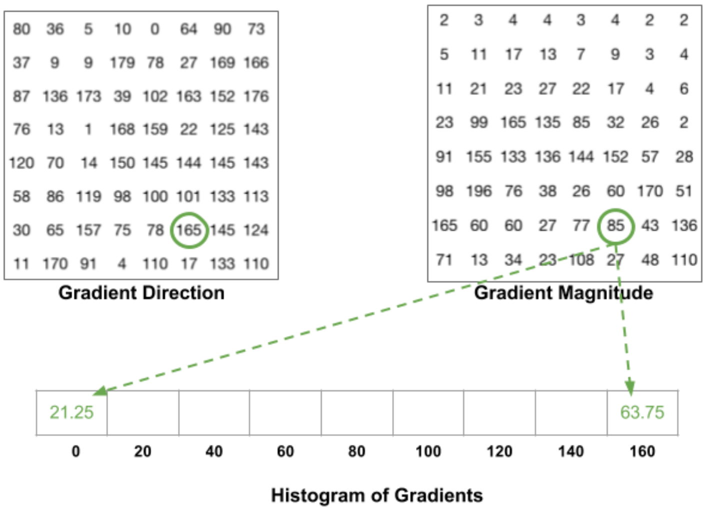
2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

From step1 we get the gradient magnitude and gradient direction of each cell. Then we calculate Histogram of Gradients to the certain angles 0, 20, 40 ··· 160 on average.



80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

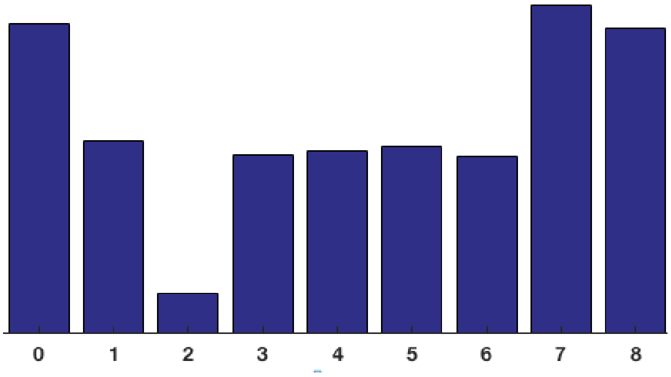
2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

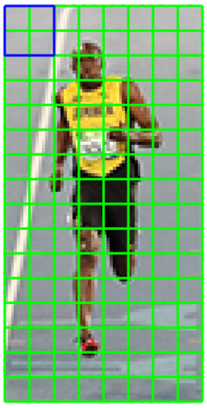
0	20	40	60	80	100	120	140	160
21.25								63.75

Histogram of Gradients

The contributions of all the pixels in the 8×8 cells are added up to create the 9-bin histogram. For the patch above, it looks like this.



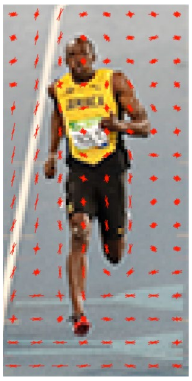
Step3, we do the 16×16 block normalization, for instance, make an RGB color vector [128, 64, 32] to a normalized form [0.87, 0.43, 0.22], by means of which, they are not affected by lighting variations.



Step4, calculate the final vector. A 16×16 block has 4 histograms which can be concatenated to form a 36 x 1 element vector and it can be normalized just the way a 3×1 vector is normalized. To calculate the final feature vector for the entire image patch, the 36×1 vectors are concatenated into one giant vector. What is the size of this vector ? Let us calculate:

1.How many positions of the 16×16 blocks do we have ? There are 7 horizontal and 15 vertical positions making a total of 7 x 15 = 105 positions.

2.Each 16×16 block is represented by a 36×1 vector. So when we concatenate them all into one giant vector we obtain a 36×105 =3780 dimensional vector.



3. Understanding of the Given Codes

3.1. Face_Detection

The annotation before the codes has generally explained the work we do, using OpenCV classifier haarcascade_frontalface_default.xml to detect faces. Let us discuss them at large.

Line

```
img = cv2.imread(image_name)
```

Loads an image from a file.

The function imread loads an image from the specified file and returns it. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format), the function returns an empty matrix (Mat::data==NULL).

Line

```
face_cascade =
cv2.CascadeClassifier("trained_files/haarcascade_frontalface_default.xml")
```

OpenCV already contains many pre-trained classifiers for face, eyes, smile etc. Those XML files are stored in opencv/data/haarcascades/ folder. Load the face detector with OpenCV XML classifierse. Then load our input image (or video) in grayscale mode.

Line

```
if img.ndim == 3:
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

If the dimension of the img is 3, we convert it to gray, on account of the fact that grayscale maps can speed up recognition.

The function cvtColor() converts an image from one color space to another. COLOR_BGR2GRAY convert between RGB/BGR and grayscale.

Line

```
faces = face_cascade.detectMultiScale(gray, 1.2, 5)
```

Detects objects of different sizes in the input image. The detected objects are returned as a list of rectangles.

Parameters

image Matrix of the type CV_8U containing an image where objects are detected.

objects Vector of rectangles where each rectangle contains the detected object, the rectangles may be partially outside the original image.

scaleFactor Parameter specifying how much the image size is reduced at each image scale.

minNeighbors Parameter specifying how many neighbors each candidate rectangle should have to retain it.

flags Parameter with the same meaning for an old cascade as in the function cvHaarDetectObjects. It is not used for a new cascade.

minSize Minimum possible object size. Objects smaller than that are ignored.

maxSize Maximum possible object size. Objects larger than that are ignored. If `maxSize == minSize` model is evaluated on single scale.

Return : 4 numbers: lower left coordinate x, y; width and height. These together form the area of face we detected.

The non-mentioned parameters are defaults.

MinNeighbors is 3 in default.

Flags can be CV_HAAR_DO_CANNY_PRUNING (not default), then Canny edge detection will be used to exclude areas with too many or too few edges, so these areas are not usually where the face is(we won’t use this one).

Line

```
img = Image.open(image_name)
```

To load an image from a file, use the open() function in the Image module.

If successful, this function returns an Image object. You can now use instance attributes to examine the file contents.

Line

```
draw_instance = ImageDraw.Draw(img)
```

The ImageDraw module provides simple 2D graphics for Image objects. You can use this module to create new images, annotate or retouch existing images, and to generate graphics on the fly for web use.

Line

```
draw_instance.rectangle((x1,y1,x2,y2), outline=(255))
```

Draws a rectangle. A 4-tuple of integers whose order and function is currently undocumented.

“Outline” can be “fill”(the color will fill the whole rectangle, we won’t use it). There are 3 colors RGB to define the color of the lines.

Line

```
img.save('the_frame_of_faces/'+image_name.split('/')[1])
```

Serializes this object to a given filename.

Line

```
if not os.path.exists('the_frame_of_faces'):
```

Return True if path refers to an existing path or an open file descriptor. Returns False for broken symbolic links. On some platforms, this function may return False if permission is not granted to execute os.stat() on the requested file, even if the path physically exists.

Line

```
os.mkdir('the_frame_of_faces')
```

Create a directory. If dir_fd is not None, it should be a file descriptor open to a directory, and path should be relative; path will then be relative to that directory. dir_fd may not be implemented on your platform. If it is unavailable, using it will raise a NotImplementedError.

Line

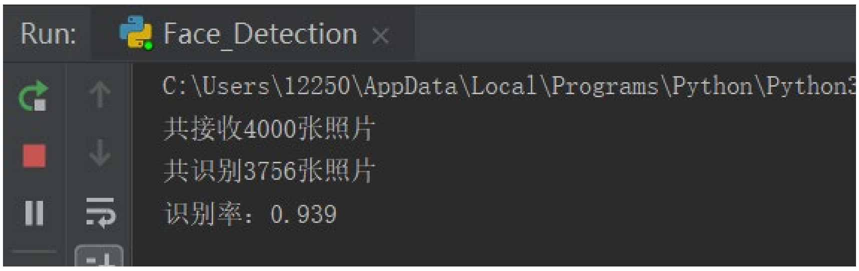
```
faces.close()
```

Close the stream.

Returns: An image object.

Raises: IOError – If the parser failed to parse the image file either because it cannot be identified or cannot be decoded.

Generally, we get the face recognition rate.



```
C:\Users\12250\AppData\Local\Programs\Python\Python3
共接收4000张照片
共识别3756张照片
识别率：0.939
```

3.2. Smile_Detection

As mentioned before, we choose HOG to detect smiles.

Line

```
wideth = image.shape[1]
heigh = image.shape[0]
```

Image.shape[] function will return the width and height, namely, the number of pixel points.

Line

```
hist=np.array([])
```

For example, you can create an array from a regular Python list or tuple using the array function. The type of the resulting array is deduced from the type of the elements in the sequences.

Line

```
lbp1 = local_binary_pattern(image[row*(i//block):row*((i//block)+1),column*(i % block):column*((i % block)+1)], 8, 1,    'default')
```

Gray scale and rotation invariant LBP (Local Binary Patterns). LBP is an invariant descriptor that can be used for texture classification.

image : (N, M) array, Graylevel image.

P : int, Number of circularly symmetric neighbour set points (quantization of the angular space).

R : float, Radius of circle (spatial resolution of the operator).

method : {‘default’, ‘ror’, ‘uniform’, ‘var’}

Method to determine the pattern.

- ‘default’: original local binary pattern which is gray scale but not rotation invariant.
- ‘ror’: extension of default implementation which is gray scale and rotation invariant.
- ‘uniform’: improved rotation invariance with uniform patterns and finer quantization of the angular space which is gray scale and rotation invariant.
- ‘nri_uniform’: non rotation-invariant uniform patterns variant which is only gray scale invariant.
- ‘var’: rotation invariant variance measures of the contrast of local image texture which is rotation but not gray scale invariant.

Returns: output : (N, M) array LBP image.

Line

```
hist1, _ = np.histogram(lbp1, normed=True, bins=256, range=(0, 256))
```

In the new edition, we change “normed” to density.

Compute the histogram of a set of data.

Parameters:

a : ***array_like***

Input data. The histogram is computed over the flattened array.

bins : ***int or sequence of scalars or str, optional***

If *bins* is an int, it defines the number of equal-width bins in the given range (10, by default). If *bins* is a sequence, it defines a monotonically increasing array of bin edges, including the rightmost edge, allowing for non-uniform bin widths.

New in version 1.11.0.

If *bins* is a string, it defines the method used to calculate the optimal bin width, as defined by histogram_bin_edges.

range : *(float, float), optional*

The lower and upper range of the bins. If not provided, range is simply (a.min(), a.max()). Values outside the range are ignored. The first element of the range must be less than or equal to the second. *range* affects the automatic bin computation as well. While bin width is computed to be optimal based on the actual data within *range*, the bin count will fill the entire range including portions containing no data.

normed : *bool, optional*

Deprecated since version 1.6.0.

This is equivalent to the *density* argument, but produces incorrect results for unequal bin widths. It should not be used.

Changed in version 1.15.0: DeprecationWarnings are actually emitted.

weights : *array_like, optional*

An array of weights, of the same shape as *a*. Each value in *a* only contributes its associated weight towards the bin count (instead of 1). If *density* is True, the weights are normalized, so that the integral of the density over the range remains 1.

density : *bool, optional*

If False, the result will contain the number of samples in each bin. If True, the result is the value of the probability *density* function at the bin, normalized such that the *integral* over the range is 1. Note that the sum of the histogram values will not be equal to 1 unless bins of unity width are chosen; it is not a probability *mass* function.

Overrides the normed keyword if given.

Returns : Return the bin edges (length(hist)+1).

Line

```
hist=np.concatenate((hist,hist1))
```

Join a sequence of arrays along an existing axis.

Parameters:

a1, a2, … : *sequence of array_like*

The arrays must have the same shape, except in the dimension corresponding to *axis* (the first, by default).

axis : *int, optional*

The axis along which the arrays will be joined. If axis is None, arrays are flattened before use. Default is 0.

out : *ndarray, optional*

If provided, the destination to place the result. The shape must be correct, matching

that of what concatenate would have returned if no out argument were specified.
Returns : **res** : ***ndarray*** The concatenated array.

Line

```
svc = SVC(kernel='linear', degree=2, gamma=1, coef0=0)
```

C-Support Vector Classification.

The implementation is based on libsvm. The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to dataset with more than a couple of 10000 samples.

The multiclass support is handled according to a one-vs-one scheme.

Parameters:

kernel : string, optional (default=’ rbf’)

Specifies the kernel type to be used in the algorithm. It must be one of ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable. If none is given, ‘rbf’ will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n_samples, n_samples).

degree : int, optional (default=3)

Degree of the polynomial kernel function (‘poly’). Ignored by all other kernels.

gamma : float, optional (default=’ auto’)

Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’.

Current default is ‘auto’ which uses 1 / n_features, if gamma='scale' is passed then it uses 1 / (n_features * X.std()) as value of gamma. The current default of gamma, ‘auto’, will change to ‘scale’ in version 0.22. ‘auto_deprecated’, a deprecated version of ‘auto’ is used as a default indicating that no explicit value of gamma was passed.

coef0 : float, optional (default=0.0)

Independent term in kernel function. It is only significant in ‘poly’ and ‘sigmoid’.

Note that choosing the right kernel is quite hard, we just understand how SVM do without too much further learning.

Line

```
svc.fit(train_histogram,train_label)
```

Fit the SVM model according to the given training data.

Parameters:

X : *{array-like, sparse matrix}, shape (n_samples, n_features)*

Training vectors, where n_samples is the number of samples and n_features is the number of features. For kernel=” precomputed”, the expected shape of X is (n_samples, n_samples).

y : *array-like, shape (n_samples,)*

Target values (class labels in classification, real numbers in regression)

sample_weight : array-like, shape (n_samples,)

Per-sample weights. Rescale C per sample. Higher weights force the classifier to put more emphasis on these points.

Returns : self : object

Line

```
predict_result=svc.predict(test_histogram)
```

Perform classification on samples in X.

For an one-class model, +1 or -1 is returned.

Parameters:

X : *{array-like, sparse matrix}, shape (n_samples, n_features)*

For kernel=" precomputed" , the expected shape of X is [n_samples_test, n_samples_train]

Returns: y_pred : *array, shape (n_samples,)*

Class labels for samples in X.

Line

```
train_label=np.vstack((label,train_label))
```

Stack arrays in sequence vertically (row wise).

This is equivalent to concatenation along the first axis after 1-D arrays of shape (N,) have been reshaped to (1,N). Rebuilds arrays divided by vsplit.

This function makes most sense for arrays with up to 3 dimensions. For instance, for pixel-data with a height (first axis), width (second axis), and r/g/b channels (third axis). The functions concatenate, stack and block provide more general stacking and concatenation operations.

Parameters:

tup : *sequence of ndarrays*

The arrays must have the same shape along all but the first axis. 1-D arrays must have the same length.

Returns: stacked : *ndarray*

The array formed by stacking the given arrays, will be at least 2-D.

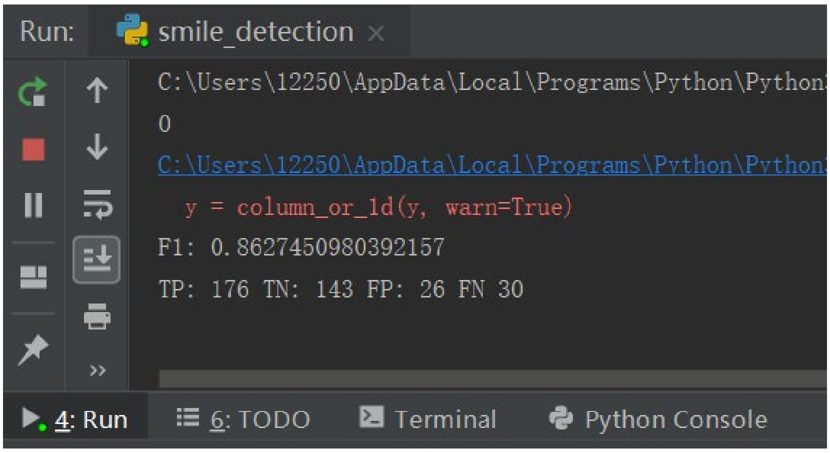
Line

Then we calculate the F1(recognition rate) 10 times, for each time we select one group to be the test set. Therefore every group are contained.

		Ground truth	
		Positive	Negative
Predicted results	Positive	True positive (TP)	False positive (FP)
	Negative	False negative (FN)	True Negative (TN)

$$F_1 = 2 \cdot \frac{\mathrm{PPV} \cdot \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

Generally, we get the smile recognition rate F1.



```
C:\Users\12250\AppData\Local\Programs\Python\Python
0
C:\Users\12250\AppData\Local\Programs\Python\Python
  y = column_or_1d(y, warn=True)
F1: 0.8627450980392157
TP: 176 TN: 143 FP: 26 FN 30
```

3.3. Real_Time_Detection

These codes will execute out computer camera and detect whether we are smiling or not. We use the built-in classifier to detect faces and smiles.

Line

```
cv2.putText(img, 'Best Detecting Area', (190, 320), 4, 0.6, (255, 0, 0), 1, cv2.LINE_AA)
```

Draws a text string.

The function putText renders the specified text string in the image. Symbols that cannot be rendered using the specified font are replaced by "Tofu" or non-drawn.

Parameters

img	Image.
text	Text string to be drawn.
org	Bottom-left/Top-left corner of the text string in the image.
fontHeight	Drawing font size by pixel unit.
color	Text color.
thickness	Thickness of the lines used to draw a text when negative, the glyph is filled. Otherwise, the glyph is drawn with this thickness.
line_type	Line type. See the line for details.
bottomLeftOrigin	When true, the image data origin is at the bottom-left corner. Otherwise, it is at the top-left corner.

Line

```
cap=cv2.VideoCapture(0)
```

Opens a video file or a capturing device or an IP video stream for video capturing with API Preference.

Line

```
width=int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)+0.5)
```

Get returns the specified VideoCapture property.

Returns : Value for the specified property. Value 0 is returned when querying a property that is not supported by the backend used by the VideoCapture instance.

CAP_PROP_FRAME_WIDTH :Width of the frames in the video stream.

Line

```
height=int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)+0.5)
```

Height of the frames in the video stream.

Line

```
fourcc=cv2.VideoWriter_fourcc('M','P','4','V')
```

The constructors/functions initialize video writers.On Windows FFMPEG or MSWF or DSHOW is used.

Line

```
out=cv2.VideoWriter('output.mp4',fourcc,20.0,(width,height))
```

This is an overloaded member function.

Parameters

filename	Name of the output video file.
fourcc	4-character code of codec used to compress the frames. For example, **VideoWriter::fourcc**('P','I','M','1') is a MPEG-1 codec, **VideoWriter::fourcc**('M','J','P','G') is a motion-jpeg codec etc. List of codes can be obtained at Video Codecs by FOURCC page. FFMPEG backend with MP4 container natively uses other values as fourcc code: see ObjectType, so you may receive a warning message from OpenCV about fourcc code conversion.
fps	Framerate of the created video stream.
frameSize	Size of the video frames.
isColor	If it is not zero, the encoder will expect and encode color frames, otherwise it will work with grayscale frames (the flag is currently supported on Windows only).

Line

```
out.write(frame)
```

Writes the next video frame.

Parameters

image	The written frame. In general, color images are expected in BGR format.

The function/method writes the specified image to video file. It must have the same size as has been specified when opening the video writer.

Line

```
cv2.imshow('Real-Time Smile Detect', frame)
```

Displays an image in the specified window.

Line

```
if cv2.waitKey(1) & 0xFF <= 126:
```

Waits for a pressed key.

The function waitKey waits for a key event infinitely (when delay⩽0) or for delay milliseconds, when it is positive. Since the OS has a minimum time between switching threads, the function will not wait exactly delay ms, it will wait at least delay ms, depending on what else is running on your computer at that time. It returns the code of the pressed key or -1 if no key was pressed before the specified time had elapsed.

0xFF is any key and no key is larger than 126.

Line

```
cap.release()
```

The method is automatically called by subsequent VideoWriter::open and by the VideoWriter destructor.

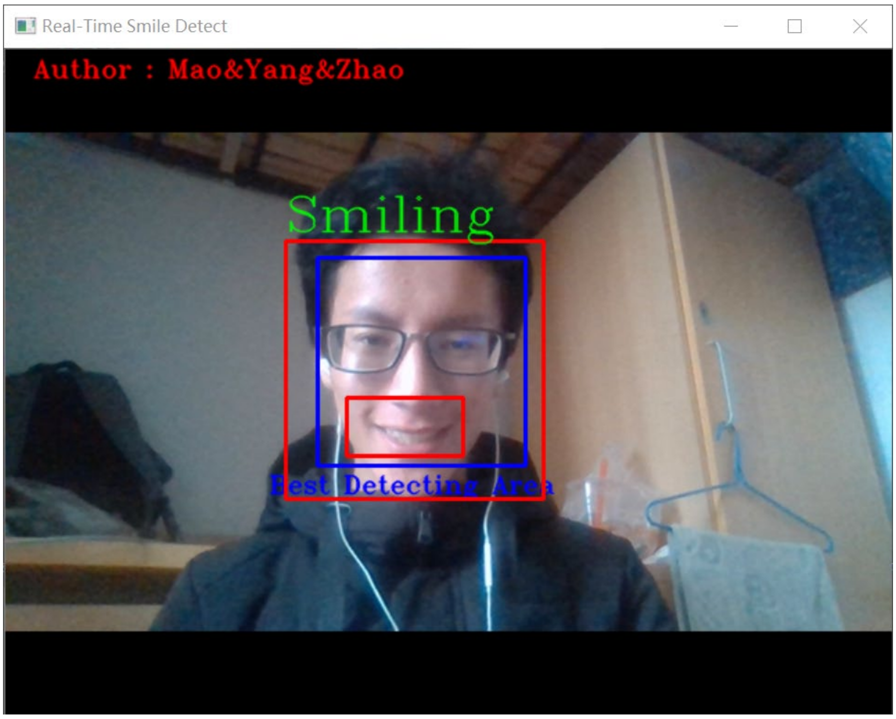
Line

```
cv2.destroyAllWindows()
```

Destroys all of the HighGUI windows.

The function destroyAllWindows destroys all of the opened HighGUI windows.

Generally, we attain the results below.



4. Simplification of the codes

What if a picture get two or more faces? The original codes will let the F1 greater than fact. So we add some codes to solve this problem.

```
if len(faces) > 1:
        count = count - len(faces) + 1
```

The results are showing below.

In Face_Detection

```
faces = face_cascade.detectMultiScale(gray, 1.2, 5)
```

scale	1.1	1.1	1.1	1.2	1.2

minsize	4	5	6	4	5
rate	0.9725	0.9655	0.96	0.94225	0.93
scale	1.2	1.3	1.3	1.3	
minsize	6	4	5	6	
rate	0.9155	0.903	0.881	0.85975	

So scale = 1.1, minsize = 4 is the best(although the running time may be longer).

In Smile_Detection

```
block = 5
```

```
block = 5                    block = 4                    block =3


0                            0                            0
F1: 0.8704156479217604       F1: 0.8456057007125891       F1: 0.8190045248868778
TP: 178 TN: 144 FP: 24 FN 29 TP: 178 TN: 132 FP: 24 FN 41 TP: 181 TN: 114 FP: 21 FN 59
1                            1                            1
F1: 0.8740740740740741       F1: 0.8502415458937198       F1: 0.801909307875895
TP: 177 TN: 140 FP: 25 FN 26 TP: 176 TN: 130 FP: 26 FN 36 TP: 168 TN: 117 FP: 34 FN 49
2                            2                            2
F1: 0.8704156479217604       F1: 0.8660287081339713       F1: 0.8373205741626795
TP: 178 TN: 129 FP: 25 FN 28 TP: 181 TN: 123 FP: 22 FN 34 TP: 175 TN: 117 FP: 28 FN 40
3                            3                            3
F1: 0.8329177057356608       F1: 0.851063829787234        F1: 0.7972350230414746
TP: 167 TN: 133 FP: 37 FN 30 TP: 180 TN: 124 FP: 24 FN 39 TP: 173 TN: 106 FP: 31 FN 57
4                            4                            4
F1: 0.8626506024096385       F1: 0.8792270531400966       F1: 0.8484848484848485
TP: 179 TN: 137 FP: 25 FN 32 TP: 182 TN: 141 FP: 22 FN 28 TP: 182 TN: 126 FP: 22 FN 43
5                            5                            5
F1: 0.9046454767726161       F1: 0.8699763593380615       F1: 0.8413793103448276
TP: 185 TN: 151 FP: 19 FN 20 TP: 184 TN: 136 FP: 20 FN 35 TP: 183 TN: 123 FP: 21 FN 48
6                            6                            6
F1: 0.8735083532219571       F1: 0.8775981524249422       F1: 0.8243243243243243
TP: 183 TN: 140 FP: 22 FN 31 TP: 190 TN: 133 FP: 15 FN 38 TP: 183 TN: 115 FP: 22 FN 56
7                            7                            7
F1: 0.8823529411764706       F1: 0.9                      F1: 0.8705882352941177
TP: 180 TN: 149 FP: 25 FN 23 TP: 189 TN: 146 FP: 16 FN 26 TP: 185 TN: 137 FP: 20 FN 35
8                            8                            8
F1: 0.8801955990220048       F1: 0.8765133171912833       F1: 0.8221153846153846
TP: 180 TN: 144 FP: 21 FN 28 TP: 181 TN: 141 FP: 20 FN 31 TP: 171 TN: 128 FP: 30 FN 44
9                            9                            9
F1: 0.8693467336683417       F1: 0.848780487804878        F1: 0.8058252427184466
TP: 173 TN: 148 FP: 28 FN 24 TP: 174 TN: 137 FP: 27 FN 35 TP: 166 TN: 127 FP: 35 FN 45
```

So block = 5 is the best.

```
lbp1 = local_binary_pattern(
    image[row * (i // block):row * ((i // block) + 1), column * (i % block):column *
((i % block) + 1)], 8, 2,
```

P:8 R:1   P:8 R:2   P:16 R:2

```
0
F1: 0.8704156479217604
TP: 178 TN: 144 FP: 24 FN 29
1
F1: 0.8740740740740741
TP: 177 TN: 140 FP: 25 FN 26
2
F1: 0.8704156479217604
TP: 178 TN: 129 FP: 25 FN 28
3
F1: 0.8329177057356608
TP: 167 TN: 133 FP: 37 FN 30
4
F1: 0.8626506024096385
TP: 179 TN: 137 FP: 25 FN 32
5
F1: 0.9046454767726161
TP: 185 TN: 151 FP: 19 FN 20
6
F1: 0.8735083532219571
TP: 183 TN: 140 FP: 22 FN 31
7
F1: 0.8823529411764706
TP: 180 TN: 149 FP: 25 FN 23
8
F1: 0.8801955990220048
TP: 180 TN: 144 FP: 21 FN 28
9
F1: 0.8693467336683417
TP: 173 TN: 148 FP: 28 FN 24
```

```
0
F1: 0.8731707317073171
TP: 179 TN: 144 FP: 23 FN 29
1
F1: 0.8790123456790123
TP: 178 TN: 141 FP: 24 FN 25
2
F1: 0.8706467661691543
TP: 175 TN: 133 FP: 28 FN 24
3
F1: 0.8308457711442786
TP: 167 TN: 132 FP: 37 FN 31
4
F1: 0.8722891566265061
TP: 181 TN: 139 FP: 23 FN 30
5
F1: 0.909952606635071
TP: 192 TN: 145 FP: 12 FN 26
6
F1: 0.8888888888888888
TP: 180 TN: 151 FP: 25 FN 20
7
F1: 0.8970588235294118
TP: 183 TN: 152 FP: 22 FN 20
8
F1: 0.8866995073891626
TP: 180 TN: 147 FP: 21 FN 25
9
F1: 0.87
TP: 174 TN: 147 FP: 27 FN 25
```

```
0
F1: 0.8186274509803921
TP: 167 TN: 134 FP: 39 FN 35
1
F1: 0.8329297820823245
TP: 172 TN: 127 FP: 39 FN 30
2
F1: 0.8312958435207825
TP: 170 TN: 121 FP: 36 FN 33
3
F1: 0.8068459657701712
TP: 165 TN: 123 FP: 40 FN 39
4
F1: 0.8077858880778589
TP: 166 TN: 128 FP: 41 FN 38
5
F1: 0.8428571428571429
TP: 177 TN: 132 FP: 39 FN 27
6
F1: 0.8262910798122066
TP: 176 TN: 126 FP: 45 FN 29
7
F1: 0.8516746411483254
TP: 178 TN: 137 FP: 35 FN 27
8
F1: 0.85
TP: 170 TN: 143 FP: 29 FN 31
9
F1: 0.8137254901960784
TP: 166 TN: 131 FP: 41 FN 35
```

So P=8, R=2 is the best(very close to 8, 1).

Other simplification has nothing to do with the results. We retain the Chinese annotations and the places where we change are in the codes. Other txt results are also submitted. The followings show some special simplification.

In Face_Detection

```
if len(faces) >= 2:   # 加上让多脸情况计数正确
    count = count - len(faces) + 1
```

In Smile_Detection

```
train_hist = lbp("train_and_test/0/file0010.jpg")
train_label = np.array([1])   # 初始化
for abc in range(10):
    if abc != time:   # 不用写两遍
        face = open('train_and_test/' + str(abc) + '/faces.txt', 'r')
```

```
face = open('train_and_test/' + str(time) + '/faces.txt', 'r')   # 打开测试集
test_hist = lbp("train_and_test/" + str(time) + "/" + "file0010.jpg")
test_label = np.array([1])  # 初始化
while True:
    ……（省略）
    if num != 0 and num != 1:   # 根据初始化修改
        test_label = np.vstack((label, test_label))
        test_hist  =  np.vstack((lbp("train_and_test/"  +  str(time)  +  "/"  +  content),
test_hist))
```

Reference documentation:

https://docs.opencv.org/master/index.html
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.htm
https://pillow.readthedocs.io/en/5.3.x/index.html
https://www.numpy.org/devdocs/genindex.html
https://scikit-learn.org/stable/index.html
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Team members:

石润晗 曲博文