

# Instituto Federal de Educação, Ciência e Tecnologia de Alagoas



## Emotion Recognition

Computação Gráfica

Alunos	Djalma Júnior Djanilson Alves
Professor	Leonardo Medeiros

Maceió, 28 de Dezembro de 2017

# Índice

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Objetivos</b>	<b>1</b>
<b>3</b>	<b>Tecnologias utilizadas</b>	<b>1</b>
3.1	OpenCV + DLib + SKLearn . . . . .	1
3.2	CK Dataset . . . . .	2
<b>4</b>	<b>Procedimento</b>	<b>2</b>
4.1	Organizando os dados . . . . .	2
4.2	Preparandos as imagens . . . . .	3
4.3	Extraindo características das faces . . . . .	6
<b>5</b>	<b>Rodando o experimento</b>	<b>6</b>
5.1	Acuracidade . . . . .	8
<b>6</b>	<b>Análise dos resultados e conclusão</b>	<b>8</b>
<b>7</b>	<b>Figuras</b>	<b>10</b>
<b>8</b>	<b>Referências</b>	<b>12</b>

# 1 Introdução

Reconhecimento de emoções é sem duvida algo que, para um humano normal é rápido, mas nem sempre tão simples, porém uma tarefa difícil para maquina. A fim de solidificar o conhecimento adquirido no estudo de visão computacional propomos a presente pesquisa que consiste em construir uma solução que possa classificar um banco de imagens baseadas nas emoções expressadas em cada imagem. E por fim avaliar a acuracidade da solução em relação ao resultado de sua classificação.

## 2 Objetivos

O presente trabalho teve como objetivo principal realizar uma pesquisa exploratória sobre métodos de conhecimento de emoções, a fim de solidificar os conhecimentos adquiridos no estudo de computação gráfica, mais especificamente no assunto de visão computacional. Para isso os objetivos específicos a serem alcançados foram: Elaborar uma solução algorítmica que, dado um banco de imagens, ele possa reconhecer as emoções para cada imagem no banco. Viabilizar um banco de imagens ou adquirir um disponibilizado por outros autores. Classificar, por meio da solução proposta, as imagens agrupando-as por emoção. Por fim, disponibilizar a metodologia utilizada, bem como os resultados obtidos por meio de um artigo acadêmico.

## 3 Tecnologias utilizadas

### 3.1 OpenCV + DLib + SKLearn

O OpenCV é uma biblioteca de visão computacional de Código Aberto. O OpenCV foi construído para fornecer uma infra-estrutura comum para aplicações de visão por computador e acelerar o uso da percepção da máquina. Esta biblioteca possui algumas classes de reconhecimento de face do qual usaremos o **FisherFace**.

**Dlib** também é uma biblioteca para reconhecimento de faces cuja técnica consiste em marcar o rosto com pontos indicando as regiões da face (como olhos, boca, nariz, etc).

Essas duas bibliotecas serão utilizadas para o reconhecimento de emoções e comparadas para verificar qual delas chegam mais próximo do resultado desejado. Usaremos o dlib em conjunto com um algoritmo de aprendizado de máquina do **sci-kit learn**.

## 3.2 CK Dataset

O conjunto de dados **Cohn-Kanade** será utilizado como base para o treinamento dos classificadores (no decorrer do artigo utilizaremos apenas *dataset* para nos referir a ele). Esse conjunto de dados consiste da 593 sequências de imagens da face de 153 sujeitos onde cada sequências vai do rosto neutro até o pico da expressão.

# 4 Procedimento

## 4.1 Organizando os dados

Inicialmente, foi necessário organizar o *dataset* para que os resultados possam ser mais fáceis de serem interpretados. No diretório do projeto foram criadas duas pastas chamadas *emotions* e *images*. Foi criado um diretório chamado *emotions* onde dados contendo os arquivos *.txt* (S005, S010, etc.) foram depositados, já as imagens em um diretório chamado *images*. Também foi criada uma pasta chamada *sorted* para armazenar as imagens de emoções ordenadas pelo nome correspondente as emoções (*neutral*, *anger*, etc.).

No arquivo *readme*, os autores do *dataset* mencionam que apenas um subconjunto (327 do 593) das sequências realmente contém as emoções. Cada sequência de imagens consiste na formação de uma expressão emocional, começando com um rosto neutro e terminando com a emoção. Então, a partir de cada sequência de imagens, extraímos duas imagens: uma neutra (a primeira imagem) e uma com a expressão emocional (a última) como foi explicado anteriormente. Para ajudar a fazer essa separação, foi necessário um pequeno script:

---

```
1 import glob
2 import os
3 from shutil import copyfile
4
5 def make_dir(src):
6     folder = os.sep.join(src.split(os.sep)[0:-1])
7
8     if not os.path.exists(folder):
9         os.makedirs(folder)
10
11 def copy_img(src, dst):
12     make_dir(dst)
```

```

13     copyfile(src, dst)
14
15     # Define emotion order
16     emotions = [
17         "neutral", "anger", "contempt", "disgust",
18         "fear", "happy", "sadness", "surprise"
19     ]
20
21     # Returns a list of all folders with participant numbers
22     participants = glob.glob(os.path.join("emotions", "*"))
23
24     for x in participants:
25         # Store current participant number
26         part = x.split(os.sep)[1]
27
28         # Store list of sessions for current participant
29         for sessions in glob.glob(os.path.join(x, "*")):
30             for files in glob.glob(os.path.join(sessions, "*")):
31                 session = files.split(os.sep)[2]
32                 file = open(files, 'r')
33
34                 # Emotions are encoded as a float, readline as float,
35                 # then convert to integer.
36                 emotion = int(float(file.readline()))
37
38                 # Get path for first and last image in sequence
39                 src_neutral = glob.glob(os.path.join("images", part, session, "*"))[0]
40                 src_emotion = glob.glob(os.path.join("images", part, session, "*"))[-1]
41
42                 # Generate path to put neutral and emotion images
43                 dst_neutral = os.path.join("sorted", "neutral", src_neutral.split(os.sep)[-1])
44                 dst_emotion = os.path.join("sorted", emotions[emotion], src_emotion.split(os.sep)[-1])
45
46                 copy_img(src_neutral, dst_neutral)
47                 copy_img(src_emotion, dst_emotion)

```

---

## 4.2 Preparandos as imagens

Para que os classificadores funcionem melhor, as imagens precisam estar no mesmo tamanho e ter apenas um rosto nelas. Os passos para isso são:

a) encontrar o rosto em cada imagem, b) converter em escala de cinza, c) recortar, redimensionar e salvá-las em outro diretório.

Para automatizar a busca da face, foi utilizado um filtro HAAR do OpenCV. O OpenCV fornece 4 classificadores pré-treinados, por isso, para ter certeza de que a busca da face será eficaz, todos os filtros foram utilizados em sequência e, assim que uma face for encontrada, descartamos os demais resultados.

Essa extração das faces se dá com o seguinte *script*:

---

```
1  import cv2
2  import glob
3  import os
4
5  faceDet_one = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
6  faceDet_two = cv2.CascadeClassifier("haarcascade_frontalface_alt2.xml")
7  faceDet_three = cv2.CascadeClassifier("haarcascade_frontalface_alt.xml")
8  faceDet_four = cv2.CascadeClassifier("haarcascade_frontalface_alt_tree.xml")
9
10 detect_options = {
11     "scaleFactor": 1.1,
12     "minNeighbors": 15,
13     "minSize": (5, 5),
14     "flags": cv2.CASCADE_SCALE_IMAGE
15 }
16
17 emotions = [
18     "neutral", "anger", "contempt", "disgust",
19     "fear", "happy", "sadness", "surprise"
20 ]
21
22 def make_dir(src):
23     folder = os.sep.join(src.split(os.sep)[0:-1])
24
25     if not os.path.exists(folder):
26         os.makedirs(folder)
27
28 def detect_faces(emotion):
29     # Get list of all images with emotion
30     files = glob.glob(os.path.join("sorted", emotion, "*"))
31
```

```

32     filenamenumber = 0
33
34     for f in files:
35         frame = cv2.imread(f)
36         gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
37
38         # Detect face using 4 different classifiers
39         face_one = faceDet_one.detectMultiScale(gray, **detect_options)
40         face_two = faceDet_two.detectMultiScale(gray, **detect_options)
41         face_three = faceDet_three.detectMultiScale(gray, **detect_options)
42         face_four = faceDet_four.detectMultiScale(gray, **detect_options)
43
44         # Go over detected faces, stop at first detected face
45         if len(face_one) == 1:
46             facefeatures = face_one
47         elif len(face_two) == 1:
48             facefeatures = face_two
49         elif len(face_three) == 1:
50             facefeatures = face_three
51         elif len(face_four) == 1:
52             facefeatures = face_four
53         else:
54             facefeatures = []
55
56         # Get coordinates and size of rectangle containing face
57         for (x, y, w, h) in facefeatures:
58             print "face found in file: %s" % f
59
60             # Cut the frame to size
61             gray = gray[y:y + h, x:x + w]
62
63             try:
64                 # Resize face so all images have same size
65                 out = cv2.resize(gray, (350, 350))
66                 filename = os.path.join("dataset", emotion, "%s.jpg" % filenamenumber)
67                 make_dir(filename)
68                 cv2.imwrite(filename, out)
69             except Exception:
70                 print "error"
71                 pass
72

```

```
73     filenumber += 1
74
75     for emotion in emotions:
76         detect_faces(emotion)
```

---

O último passo foi filtrar do nosso *dataset* as imagens neutras, pois como a maioria dos participantes tem várias amostras de expressões, temos algumas imagens repetidas da mesma pessoa. Neste caso, o trabalho de remoção foi manual.

### 4.3 Extraindo características das faces

Para treinar nosso *dataset* usando dlib, foi necessário extrair algumas características da imagem para que o classificador trabalhe corretamente. A forma que foi feita essa extração da informação foi obtendo as coordenadas do ponto central do rosto e, em seguida, obter a posição de todos os pontos relativos a este ponto central, ao total o Dlib mapeia a face com 68 pontos (figura 1). Isso se faz necessário para evitar que o classificador trabalhe com o posicionamento absoluto dos pontos, visto que a pessoa pode apresentar a mesma expressão, porém os pontos estão em posições diferentes da tela. As faces também podem estar inclinadas, o que pode confundir o classificador. Foi feita uma correção simples para amenizar esse problema que consiste em calcular a correção do deslocamento da face tomando a ponta do nariz e encontrando o ângulo que o nariz faz em relação à imagem, e assim encontrar o deslocamento angular que precisamos aplicar.

## 5 Rodando o experimento

O código principal do programa é o seguinte:

---

```
1  import cv2
2  import imutils
3  import helpers
4  import numpy as np
5
6  from imutils.video import VideoStream
7  from imutils import face_utils
8  from sklearn.svm import SVC
9
```



```

10 fishface = cv2.face.createFisherFaceRecognizer()
11 # fishface = cv2.face.FisherFaceRecognizer_create()
12 classifier = SVC(kernel='linear', probability=True, tol=1e-3)
13
14 print("making sets...")
15 land_data, land_labels, fish_data, fish_labels = helpers.make_sets()
16
17 print("training SVM linear classifier...")
18 classifier.fit(np.array(land_data), np.array(land_labels))
19
20 print("training fisher face classifier...")
21 fishface.train(fish_data, np.asarray(fish_labels))
22
23 vs = VideoStream().start()
24
25 while True:
26     frame = imutils.resize(vs.read(), width=400)
27     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
28     rects = helpers.detector(gray, 0)
29     landmarks = helpers.get_landmarks(gray)
30     face = helpers.detect_and_resize_face(gray)
31     thumb = cv2.cvtColor(cv2.resize(face, (100, 100)), cv2.COLOR_GRAY2RGB)
32
33     frame[-100:, -100:] = thumb
34
35     try:
36         idx, conf = fishface.predict(face)
37         cv2.putText(frame, helpers.emotions[idx], (15, 30),
38             cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)
39     except Exception:
40         pass
41
42     try:
43         idx = classifier.predict([landmarks])[0]
44         cv2.putText(frame, helpers.emotions[idx], (15, 80),
45             cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
46     except Exception:
47         pass
48
49     # loop over the face detections
50     for rect in rects:

```

```

51     # determine the facial landmarks for the face region, then
52     # convert the facial landmark (x, y)-coordinates to a NumPy array
53     shape = helpers.predictor(gray, rect)
54     shape = face_utils.shape_to_np(shape)
55
56     # loop over the (x, y)-coordinates for the facial landmarks
57     # and draw them on the image
58     for (x, y) in shape:
59         cv2.circle(frame, (x, y), 1, (0, 0, 255), -1)
60
61     # show the frame
62     cv2.imshow("Frame", frame)
63     key = cv2.waitKey(1) & 0xFF
64
65     # if the `q` key was pressed, break from the loop
66     if key == ord("q"):
67         break
68
69     # do a bit of cleanup
70     cv2.destroyAllWindows()
71     vs.stop()

```

---

## 5.1 Acuracidade

Para obter a acuracidade dos algoritmos, foi obtido aleatoriamente 80% das imagens e os 20% restantes foram classificadas para efeito de comparação. Esse processo foi repetido 10 vezes. Os scripts `accur_fishface.py` e `accur_landmarks.py` são utilizados para fazer esses cálculos.

A precisão do reconhecimento usando a técnica de fishface obteve uma média de 69.9% (figura 2). Utilizando landmarks, 71,5% (figura 3).

## 6 Análise dos resultados e conclusão

A primeira coisa a se notar é que temos poucos exemplos de emoções. Isso faz com que os classificadores não tenham uma precisão muito boa e informe incorretamente a emoção obtida pela câmera. Usar um conjunto de dados maior provavelmente aumentará bastante a detecção. Além do mais, o *dataset* não condiz exatamente com uma emoção que as pessoas fazem no dia a dia. Algumas dessas imagens chegam até a ser cômicas.

É claro que o reconhecimento de emoção é uma tarefa complexa, mais ainda quando só usa imagens. Mesmo para nós humanos, isso é difícil porque o reconhecimento correto de uma emoção facial muitas vezes depende do contexto dentro do qual a emoção se origina e se expressa.

O repositório desse projeto pode ser encontrado em <https://github.com/djalmajr/emotion-recognition>

## 7 Figuras

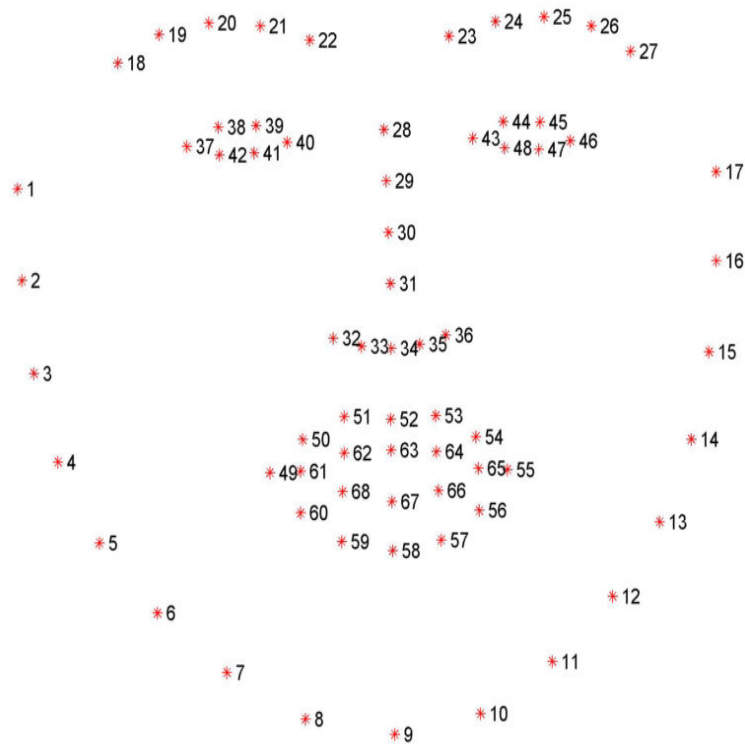


Figura 1: Mapeamento facial de 68 pontos da biblioteca Dlib

```

size of training set is: 413 images
predicting classification set
got 68 percent correct!
training fisher face classifier
size of training set is: 413 images
predicting classification set
got 68 percent correct!
training fisher face classifier
size of training set is: 413 images
predicting classification set
got 73 percent correct!
training fisher face classifier
size of training set is: 413 images
predicting classification set
got 64 percent correct!

~

end score: 69.9 percent correct!

```

Figura 2: Precisão utilizando fishface

```

no face detected on this one
working on contempt
working on disgust
working on fear
no face detected on this one
working on happy
no face detected on this one
no face detected on this one
working on sadness
no face detected on this one
working on surprise
no face detected on this one
training SVM linear 9
getting accuracies 9
linear: 0.756302521008
Mean value lin svm: 0.715880444035

```

Figura 3: Precisão utilizando landmarks

## 8 Referências

van Gent, P. (2016). Emotion Recognition With Python, OpenCV and a Face Dataset. Retrieved from: <http://www.paulvangent.com/2016/04/01/emotion-recognition-with-python-opencv-and-a-face-dataset/>

van Gent, P. (2016). Emotion Recognition With Python, OpenCV and a Face Dataset. A tech blog about fun things with Python and embedded electronics. Retrieved from: <http://www.paulvangent.com/2016/04/01/emotion-recognition-with-python-opencv-and-a-face-dataset/>

Adrian Rosebrock, (2017). Facial landmarks with dlib, OpenCV, and Python. Retrieved from: <https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/>