Trabalho Final PDI
Utilização de técnicas de detecção de faces em imagens de baixa resolução

Departamento de Computação e Automação - DCA/UFRN
Engenharia de Computação e Automação - DCA/UFRN
Aluno/Discente: Lukas Maximo Grilo Abreu Jardim
Professor/Docente: Agostinho Brito de Medeiros Jr.

July 4, 2018

Figure 1: Múltiplas faces em uma imagem.

1 Introdução

O Objetivo desse trabalho era demonstrar e adaptar um sistema de reconhecimento de faces para imagens em baixa resolução, com a utilização da técnica de Viola Jones, uma método que consiste em aplicar máscaras de filtros rentangulares (Haar Features) em uma imagem afim de se obter uma correspondência com um banco de dados definido previamente, definida a partir da combinação de vários destes filtros em um sistema de detecção que envolve técnicas de aprendizado de maquina.

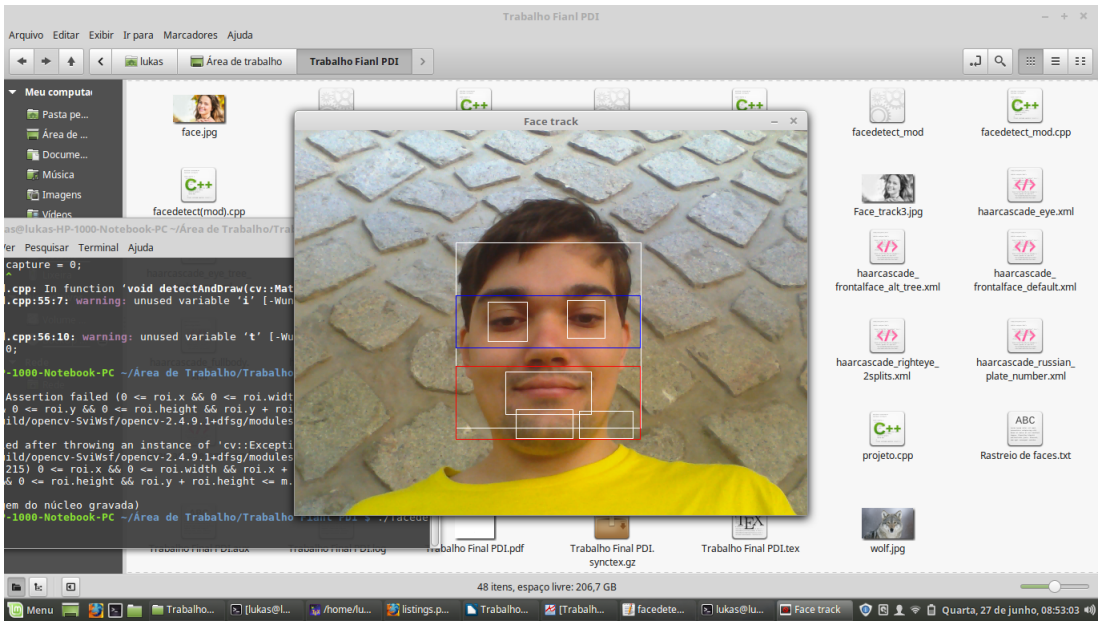

Figure 2: Os dois olhos foram detectados separadamente (em branco).

2 Experimentos

2.1 Aplicação da técnica em uma imagem de vídeo

Inicialmente, a técnica de reconhecimento foi aplicada em uma imagem de vídeo obtida através de uma câmera de computador onde ela procura catalogar e reconhecer vários elementos além do rosto da pessoa (no caso, os dois olhos e a boca), para melhor representação do método.

Figure 3: Uma Imagem em perfeitas condições.

2.2 detecção de faces em uma imagem

No segundo teste, o algoritmo de detecção foi aplicado em duas imagem em tons de cinza bem definidas, e da mesma maneira que foi executada na imagem de vídeo, a recognição foi realizada normalmente.
Nas imagens acima, que estão em boas condições, foram processadas por um algorítimo não muito diferentes do algorítmo de vídeo, mas que também processou a imagem para encontrar todas as faces possíveis, tantoque ele não detectou todas as faces direito na imagem superior.

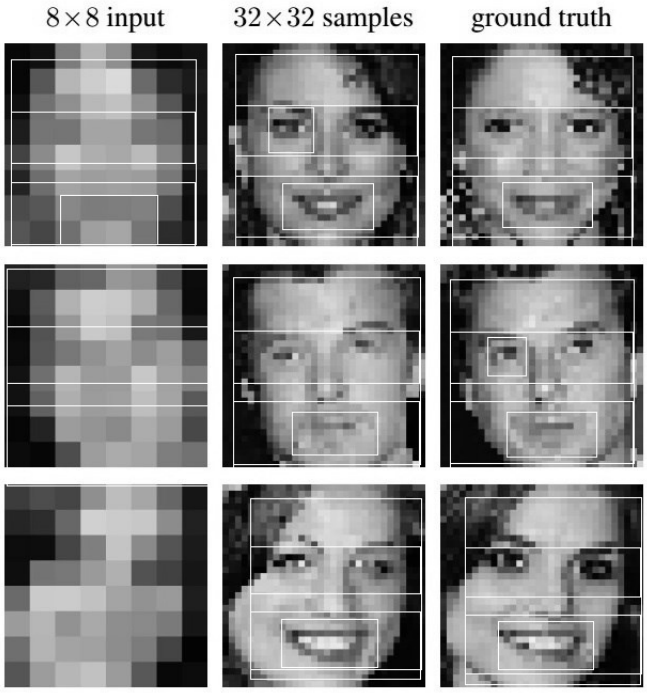

Figure 4: Imagem definitiva.

3 Teste Definitivo: Detecção de faces em uma imagem em baixa qualidade

Por último, e mais importante, o algoritmo foi utilizado em uma imagem que continha várias faces, e ao mesmo tempo, possuía diferentes graus de péssima qualidade de visualização. Mesmo assim, com a baixa qualidade da imagem, a partir de um certo grau limite de baixa resolução e de um tamanho da face mínimo, o algorítimo consegue encontrar as faces.

Para realizar múltiplas detecções, foi utilizado um vetor que armazena cada face ou elemento de face detectadas pelo algoritmo (cada elemento de cada vez), onde uma vez detectado uma face, a mesma face não vai ser detectada novamente.

4 Conclusão

Os códigos utilizados nesse trabalho, vide Apêndice, foram melhorados ou adaptados com o intuito de atender as exigências requeridas no mesmo. Os algorítmos utilizados para detectar as faces nas imagens foram adaptados para serem exibidos com maior visibilidade na detecção.

5 Apêndice

5.1 Códigos do Trabalho

5.1.1 facedetect modificado - (*facedetect_mod.cpp*)

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;

void detectAndDraw( Mat& img);

CascadeClassifier cascadeFace, cascadeMouth;

String cascadeFaceName  = "haarcascade_frontalface_alt.xml";
String cascadeMouthName = "haarcascade_mcs_mouth.xml";

int main( int argc, const char** argv ){
CvCapture* capture = 0;
Mat frame, frameCopy, image;
VideoCapture cap(0);
int key;

if( !cascadeFace.load( cascadeFaceName )) {
cerr << "ERRO: Nao carregou filtro em cascata facefrontal" << endl;
return -1;
}
if( !cascadeMouth.load( cascadeMouthName )) {
cerr << "ERRO: Nao carregou filtro em cascata mouth" << endl;
return -1;
}

//  cap.set(CV_CAP_PROP_FRAME_WIDTH, 320);
//  cap.set(CV_CAP_PROP_FRAME_HEIGHT, 240);

for(;;){
cap >> frame;
flip(frame, frameCopy, 1); // inverte a imagem horizontalmente
///imshow("image", frameCopy);
//cout << "foi\n";
detectAndDraw(frameCopy); // detecta

key = (char) waitKey(10);
if( key == 27 ) break;
}
return 0;
}

void detectAndDraw( Mat& img){
int i = 0;
double t = 0;
```

```
52  vector<Rect> faces;
53
54  Mat gray;
55  cascadeFace.detectMultiScale(img, // imagem para deteccao
56  faces, // vetor com os retangulos encontrados
57  1.1, // escala de multiresolucao
58  3, // numero de vizinhos que cada candidato a retangulo
59  // devera contemplar. evita multiplas deteccoes parecidas
60  // na mesma regiao
61  0 | CV_HAAR_FIND_BIGGEST_OBJECT, // parametros (normalmente nao
        usados)
62  Size(30, 30) ); // minimo tamanho para deteccao de um objeto
63
64  for( vector<Rect>::const_iterator r = faces.begin(); r != faces.end
        (); r++ ){
65  Mat imgROI;
66  vector<Rect> nestedObjects;
67
68  rectangle(img,
69  Point(r->x, r->y),
70  Point(r->x + r->width, r->y + r->height),
71  CV_RGB(255, 0, 0), 1, 8, 0);
72
73  if( cascadeMouth.empty() )
74  continue;
75
76  // posicao aproximada da boca em relacao a face...
77  Rect mouthROI = Rect(r->x, r->y + (r->height/1.5),
78  r->width, r->height/2.5);
79
80  rectangle(img, mouthROI, CV_RGB(255, 255, 0), 1, 8, 0);
81
82  imgROI = img(mouthROI);
83
84  cascadeMouth.detectMultiScale(
85  imgROI,
86  nestedObjects,
87  1.1,
88  2,
89  0 | CV_HAAR_FIND_BIGGEST_OBJECT,
90  Size(30, 30) );
91  // busca as bocas encontradas e desenha os retangulos
92  for( vector<Rect>::const_iterator nr = nestedObjects.begin(); nr !=
        nestedObjects.end(); nr++ ){
93  rectangle(img,
94  Point(r->x + nr->x, r->y + (r->height/1.5) + nr->y  ),
95  Point(r->x + nr->x + nr->width, r->y + (r->height/1.5) + nr->y + nr
        ->height),
96  CV_RGB(255, 0, 255), 1, 8, 0);
97  }
98  }
99  imshow("Face_track", img );
100 }
```

5.1.2 face01 - (*face*01.*cpp*)

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;

void detectAndDraw( Mat& img);

CascadeClassifier cascadeFace, cascadeMouth, cascadeEye;

String cascadeFaceName  = "haarcascade_frontalface_alt.xml";
String cascadeMouthName = "haarcascade_smile.xml";
String cascadeEyeName = "haarcascade_eye_tree_eyeglasses.xml";

int main( int argc, const char** argv ){
//CvCapture* capture = 0;
Mat image;
//VideoCapture cap(0);
int key;

if( !cascadeFace.load( cascadeFaceName )) {
cerr << "ERRO: Nao carregou filtro em cascata facefrontal" << endl;
return -1;
}
if( !cascadeMouth.load( cascadeMouthName )) {
cerr << "ERRO: Nao carregou filtro em cascata mouth" << endl;
return -1;
}
if( !cascadeEye.load( cascadeEyeName )) {
cerr << "ERRO: Nao carregou filtro em cascata olho" << endl;
return -1;
}

//  cap.set(CV_CAP_PROP_FRAME_WIDTH, 320);
//  cap.set(CV_CAP_PROP_FRAME_HEIGHT, 240);

for(;;){
//cap >> frame;
//flip(frame, frameCopy, 1); // inverte a imagem horizontalmente
///imshow("image", frameCopy);
//cout << "foi\n";
image = imread(argv[1],CV_LOAD_IMAGE_GRAYSCALE);
detectAndDraw(image); // detecta

key = (char) waitKey(10);
if( key == 27 ) break;
}
return 0;
}

void detectAndDraw( Mat& img){
```

```
56  int i = 0;
57  double t = 0;
58  vector<Rect> faces;
59
60  Mat gray;
61  cascadeFace.detectMultiScale(img, // imagem para deteccao
62  faces, // vetor com os retangulos encontrados
63  1.1, // escala de multiresolucao
64  3, // numero de vizinhos que cada candidato a retangulo
65  // devera contemplar. evita multiplas deteccoes parecidas
66  // na mesma regiao
67  0 | CV_HAAR_FIND_BIGGEST_OBJECT, // parametros (normalmente nao
        usados)
68  Size(30, 30) ); // minimo tamanho para deteccao de um objeto
69
70  for( vector<Rect>::const_iterator r = faces.begin(); r != faces.end
        (); r++ ){
71  Mat imgROI, imgROI2;
72  vector<Rect> nestedObjects, nestedObjects2;
73
74  rectangle(img,
75  Point(r->x, r->y),
76  Point(r->x + r->width, r->y + r->height),
77  CV_RGB(255, 255, 255), 1, 8, 0);
78
79  if( cascadeMouth.empty() )
80  continue;
81
82  // posicao aproximada da boca em relacao a face...
83  Rect mouthROI = Rect(r->x, r->y + (r->height/1.5),
84  r->width, r->height/2.5);
85
86
87  rectangle(img, mouthROI, CV_RGB(255, 255, 255), 1, 8, 0);
88
89  imgROI = img(mouthROI);
90
91  cascadeMouth.detectMultiScale(
92  imgROI,
93  nestedObjects,
94  1.1,
95  2,
96  0 | CV_HAAR_FIND_BIGGEST_OBJECT,
97  Size(30, 30) );
98
99  // busca os olhos e as bocas encontradas e desenha os retangulos
100 for( vector<Rect>::const_iterator nr = nestedObjects.begin(); nr !=
        nestedObjects.end(); nr++ ){
101 rectangle(img,
102 Point(r->x + nr->x, r->y + (r->height/1.5) + nr->y  ),
103 Point(r->x + nr->x + nr->width, r->y + (r->height/1.5) + nr->y + nr
        ->height),
104 CV_RGB(255, 255, 255), 1, 8, 0);
105 }
106 if( cascadeEye.empty() )
107 continue;
108 Rect eyeROI = Rect(r->x, r->y + (r->height/3.5),
```

```
109 r->width, r->height/3.5);
110 rectangle(img, eyeROI, CV_RGB(255, 255, 255), 1, 8, 0);
111
112 imgROI2 = img(eyeROI);
113 cascadeEye.detectMultiScale(
114 imgROI2,
115 nestedObjects,
116 1.1,
117 2,
118 0 | CV_HAAR_FIND_BIGGEST_OBJECT,
119 Size(30, 30) );
120 for( vector<Rect>::const_iterator nr = nestedObjects.begin(); nr !=
        nestedObjects.end(); nr++ ){
121 rectangle(img,
122 Point(r->x + nr->x, r->y + (r->height/3.5) + nr->y ),
123 Point(r->x + nr->x + nr->width, r->y + (r->height/3.5) + nr->y + nr
      ->height),
124 CV_RGB(255, 255, 255), 1, 8, 0);
125 }
126 }
127 imwrite("Face_track.jpg", img );
128 imshow("Face_track", img );
129 }
```