IDENTIFICAÇÃO FACIAL COM HARDWARE DE BAIXO CUSTO UTILIZANDO HISTOGRAMAS DE PADRÕES BINÁRIOS LOCAIS

Lucas Rubian Schatz[1], Fernanda Moreira Behrend Corrêa[2], Ricardo Antonello[3]
[1]Instituto Federal Catarinense, Campus Luzerna / lucasschatz@gmail.com
[2]Instituto Federal Catarinense, Campus Luzerna / fernanda.behrend1203@gmail.com
[3]Instituto Federal Catarinense, Campus Luzerna / ricardo.antonello@ifc.edu.br

Resumo: ***Neste trabalho foi utilizada a linguagem Python juntamente com a biblioteca de visão computacional OpenCV para construir um sistema biométrico de identificação de faces. O sistema realiza a identificação on-line através de um streaming de vídeo capturando os frames e analisando-os isoladamente. O hardware utilizado para captura foi uma webcam comum com resolução de 1 megapixel 720p além de um computador padrão PC para processamento. Através da biblioteca OpenCV foi utilizado o algoritmo "Histogramas de padrões binários locais" explicitado no referencial teórico deste trabalho. Apesar deste algoritmo ser mais resistente a mudanças na iluminação do ambiente é importante frisar a necessidade de cadastramento de pelo menos 10 imagens para garantir a performance na identificação biométrica acima de 90%. O sistema desenvolvido, após o cadastramento da primeira imagem da face, cria um 'id' único para aquele rosto e a cada nova identificação da mesma face no sistema refaz o treinamento. Este procedimento faz o acerto inicial subir de próximo de 60% com uma face cadastrada para mais de 90% após 10 identificações com o treinamento aprimorado. Não foi utilizada a tecnologia de captura de imagens com sensores de profundidade o que certamente resultaria em uma maior precisão mas aumentariam o custo do sistema. Foi opção dos autores utilizarem o hardware de baixo custo. Por isso, é importante considerar a iluminação do ambiente já que os resultados demonstram que a iluminação estável é um fator chave para uma grande taxa de acerto. A troca da direção da iluminação do ambiente ou a mudança do posicionamento da câmera afetam a performance do sistema conforme descrito em detalhes nos resultados obtidos. Conclui-se que é possível utilizar um hardware de baixo custo para identificação biométrica em ambientes internos com iluminação e posicionamento da câmera estáveis com taxas superiores a 90% de acerto.***

Palavras-Chave: ***Visão computacional, Identificação facial, Histogramas de padrões binários locais.***

1. INTRODUÇÃO

O principal objetivo de tal projeto é, com uma câmera, reconhecer o rosto das pessoas já cadastradas e adicionar novos rostos ao banco de dados para que sejam identificados também. A linguagem *Python* (PYTHON, 2017), em sua versão 3, e a biblioteca de imagens *OpenCV* (ROSEBROCK, 2015), também na versão 3, serão as principais ferramentas de trabalho para o desenvolvimento do projeto proposto, ainda levando em consideração que o sistema final será apresentado em uma aplicação desktop.

2. REFERENCIAL TEÓRICO

2.1. Linguagem de programação *Python*

A linguagem de programação Python foi originada por Guido Van Rossum em 1991 (PYTHON, 2017), com o intuito de ser eficiente e de fácil interpretação.

2.2. Biblioteca *OpenCV*

> OpenCV é uma popular biblioteca de visão computacional criada pela Intel no ano de 1999. É uma biblioteca multiplataforma com foco no processamento de imagens em tempo real

além de incluir implementações para algoritmos de visão computacional bem recente. Em 2008 Willow Garage encarregou-se do suporte e o OpenCV, na versão 2.3.1, agora possui uma interface de programação para C, C++, Python e Android. O OpenCV é lançado sob uma licença BSD, consequentemente é usando em projetos acadêmicos e produtos comerciais. Em sua versão 2.4, o OpenCV possui a classe FaceRecognizer para reconhecimento de rostos, onde se pode começar a experimentar o reconhecimento facial imediatamente (OPENCV, 2017b).

2.3. Reconhecimento facial utilizando Haar Cascade

Segundo a documentação do OpenCV intitulada *Face Detection using Haar Cascades:*

> Detecção de objetos utilizando *Haar Cascade* é um método efetivo de detecção de objetos proposto por Paul Viola e Michael Jones em seu artigo *Rapid Object Detection using a Boosted Cascade of Simple Features* publicado em 2001. É uma abordagem baseada em aprendizado de máquina onde uma função em cascata é treinada a partir de muitas imagens positivas e negativas. Dessa forma é usado para detectar objetos em outras imagens.
> Para auxiliar no desenvolvimento do projeto e obtenção de conhecimento, o livro *Pratical Python and OpenCV* foi disponibilizado pelo professor. Todos os exemplos encontrados no livro utilizam o *Haar Cascade* para detecção de objetos (OPENCV, 2017a).

2.4. Oracle VM VirtualBox

O *VirtualBox* (TECHTUDO, 2017) é um programa multiplataforma desenvolvido pela empresa *Oracle* com o intuito de criar máquinas virtuais capazes de emular sistemas operacionais (*Linux*, *Windows*, *iOS*, *Android*, etc.). Durante os procedimentos práticos uma imagem Linux pré-configurada com a biblioteca *OpenCV* e outros componentes foi utilizada por meio da máquina virtual.

2.5. Padrões binários locais

Esse tipo de padrão nada mais é do que as características locais retiradas como olhos, nariz e boca (AMARAL; THOMAZ, 2017). Como foi utilizado hardware de baixo custo, alguns aspectos devem ser cuidados para que a detecção seja o mais eficiente possível como: iluminação, a posição do rosto, interferência de acessórios (óculos, bonés, barba, etc.), rostos parcialmente cobertos.

2. DESENVOLVIMENTO

Basicamente, todo o projeto foi desenvolvido em uma máquina virtual Linux (TECHTUDO, 2017) com a biblioteca *OpenCV* já configurada conforme Rosebrock (2017). Desta forma o código poderia ser implementado num bloco de notas normal ou através de um software próprio para o desenvolvimento do mesmo. O aplicativo terminal seria o meio do código entrar em execução.

Para pôr em prática qualquer código com extensão ".py" que acesse a biblioteca OpenCV, é necessário executar uma linha de código no terminal para iniciar a o ambiente virtual onde está instalada a linguagem Python e a biblioteca OpenCV. Portanto, foi executado o seguinte comando: "*source start_py3cv3.sh"* onde o arquivo "start_py3cv3.sh" possui a configuração de acesso ao ambiente virtual.

Após iniciar o ambiente virtual conforme definimos acima, acessamos o diretório que contém o arquivo ".py" com o código a executar. Para executar o arquivo acionamos o seguinte comando: "*python detectarosto.py"* que contém o nome do arquivo a executar.

O código presente no arquivo "*detectarosto.py"* identifica os rostos já treinados e conhecidos pelo banco de dados, mostrando na tela o número e o nome correspondente ao rosto juntamente com o quadrado desenhado onde a face é reconhecida. Caso o rosto não seja reconhecido, podemos treinar aquele rosto para que possa ser identificado como já cadastrado pelo banco.

Tabela 1 – Código de detecção de rosto

	Código	Descrição
1	from math import atan2, atan, degrees, pi import datetime import operator from treinaUtil import *	Importa biblioteca matemática e operadores, manipulador de data e hora, e funções criadas na rede neural.

2	class Posicao:
	  def __init__(self, x, y, w ,h):
	  self.x=x
	  self.y=y
	  self.w=w
	  self.h=h
	  def hittest(self, posicao, stop=False):
	  if self.x <= (posicao.x+posicao.w//2) <= self.x+self.w:
	  if self.y <= (posicao.y + posicao.h // 2) <= self.y + self.h:
	  return True
	  if(not stop):
	  return posicao.hittest(self, stop=True)
	  return False
	  def center(self):
	  return (self.x+self.w//2, self.y+self.h//2)
	  def pos(self):
	  return (self.x, self.y)
	  def module(self, posicao):
	  vect=np.asarray(self.center())-np.asarray(posicao.center())
	  return np.sqrt(vect.dot(vect))
		Declaração da classe *Posicao*. Classe que vai intervir no momento de reconhecer se há alguma face em determinada posição da imagem.
		A função *init* será por onde a classe começará.
		A função *hittest* retornará o valor da posição.
		A função *center* será aplicada para no conteúdo central da posição.
		A função *pos* retornará a posição.
		A função *module* irá calcular o vetor de posição centralizada e retornar a raiz do vetor.

3	class Rosto:
	  def __init__(self, existe, id, conf, posicao, nome=""):
	  self.existe = existe
	  self.id=id
	  self.conf = conf
	  self.posicao=posicao
	  self.hora=datetime.datetime.now()
	  self.nome=nome
	  def hittest(self, rosto, stop=False):
	  return self.posicao.hittest(rosto.posicao)
	  def idade(self):
	  r=datetime.datetime.now()-self.hora
	  #r.total_seconds()
	  return r
		Declaração da classe *Rosto*.
		A função *init* será por onde a classe '*rosto'* será iniciada. Os selfs são usados para indicar que os argumentos serão eles mesmos quando chamados.
		A função *hittest* retornará a posição do rosto.

4	rosto = Rosto(False, -1, 0, 0, "") rostosNoCache={}	Utilização da classe *Rosto* dentro de uma variável denominada *rosto* onde: *self=False, existe=-1, id=0,*

		conf=0, posição=0, nome=””.

5	def ensure_dir(file_path):
	  directory = os.path.dirname(file_path)
	  if not os.path.exists(directory):
	  os.makedirs(directory)
		Função que verifica se o diretório existe. Caso não exista, cria um novo diretório. Essa função será utilizada para armazenar as imagens capturadas dos rostos dentro de pastas.

6	path="./faces/"
	ensure_dir(path)
	maxId=preparaArquivos(path)
	print("Max id:",maxId)
	#quit()
	arqRedeNeural="trainingData_LBPHF.yml"
	if not loadNeuralNetwork(arqRedeNeural):
	  treinaImagens("faces/")
	  salvaNeuralNetwork(arqRedeNeural)
		Esse bloco define o caminho em que se encontra o diretório correspondente à ID do rosto. Ele carrega o arquivo da rede neural para verificar se o rosto já é conhecido. Caso o rosto já não esteja treinado, ele treina o rosto da pessoa, armazena os arquivos na rede neural e a salva.

7	cap = cv2.VideoCapture(0)
	xmls = [
	  ["xml/haarcascade_frontalface_alt.xml", (255, 255, 0), "alt"]
	  # ["xml/haarcascade_frontalface_alt2.xml", (255,0,255), "alt2"],
	  # ["xml/haarcascade_frontalface_default.xml", (0,255,255), "default"],
	  # ["xml/haarcascade_frontalface_alt_tree.xml", (127,127,0), "tree"],
	  # ["xml/haarcascade_frontalcatface.xml", (127,0,127), "catface"],
	  # ["xml/haarcascade_frontalcatface_extended.xml", (0,127,127), "catextended"]
	]
	cascades = []
		Aqui é pega a lista de rostos detectados. A variável *cap* é responsável por capturar o vídeo.
		Os arquivos *.xml* também são chamados juntamente com parâmetros e agrupados em uma *tupla (lista)*. O primeiro é o nome do arquivo, o segundo é relacionado a cor e o terceiro funciona como abreviação do nome dos arquivos para utilizar de forma mais prática no decorrer do desenvolvimento do algoritmo. A variável *cascades* também é uma *tupla*.

8	for xml in xmls:
	  xml.append(cv2.CascadeClassifier(xml[0]))
	  # xml.append("oi")
	# dfSmile=cv2.CascadeClassifier("xml/haarcascade_smile.xml")
	dfSmile = cv2.CascadeClassifier("xml/mouth.xml")
	# dfEyes=cv2.CascadeClassifier('xml/ambosOlhos.xml')
	dfEyes = cv2.CascadeClassifier('xml/haarcascade_eye_tree_eyeglasses.xml')
	dfEyeLeft = cv2.CascadeClassifier("xml/olhoLeft.xml")
	dfEyeRight = cv2.CascadeClassifier("xml/olhoRight.xml")
		O laço de repetição *for* será responsável por adicionar os arquivos *.xml* dentro da tupla *xmls*, onde cada arquivo estará alocado dentro de uma variável (*dfSmile*, *dfEyes*, etc.).

9	# print(xmls)
	inverter = False
	treina=False
	blur = True
	nomeNaFace=False
	xEye, yEye, wEye, hEye = 0, 0, 0, 0
	green = (0, 255, 0)
		Aqui algumas variáveis são declaradas: *inverter*, *treina*, *blur*, *green*, etc.

10	def escreveNome(img, text, pos, fontScale=0.8, thickness=2, color=(0, 255, 0)):
	  cv2.putText(img, text, pos, cv2.FONT_HERSHEY_SIMPLEX, fontScale, color, thickness)
	def pegaTamanhoNome(text, fontScale=0.8, thickness=2):
	  return cv2.getTextSize(text, cv2.FONT_HERSHEY_SIMPLEX, fontScale, thickness)
		A função *escreveNome* será utilizada para escrever o nome correspondente à ID do rosto que está sendo detectado.
		A função *pegaTamanhoNome* pega o tamanho do nome

		correspondente à ID do rosto detectado.
11	while (True):	Todo o código dentro de *while(True)* será executado até que o programa pare de executar.

12	ret, img = cap.read()
	img[:, :] = img[:, ::-1]
	mascara = np.zeros(img.shape[:2], dtype="uint8")
	
	iPB = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
	relatorio = "";
	dellist=[]
	for id in rostosNoCache.keys():
	if rostosNoCache[id].idade().total_seconds() > 3:
	dellist.append(id)
	for id in dellist:
	del rostosNoCache[id]
		Aqui a imagem é capturada frame por frame. A variável *img[:,:]* será responsável por espelhar a imagem.
		A variável *iPB* será responsável por converter a imagem em escala de cinza.
		A partir da variável *relatorio* será criado o detector de faces.
		O laço *for* irá adicionar o rosto na tupla *dellist*, caso esse rosto ainda não tenha sido cadastrado.

13	for tupla in xmls:
	# df = cv2.CascadeClassifier('xml/haarcascade_frontalface_alt2.xml')
	df = tupla[3]
	faces = df.detectMultiScale(iPB,
	scaleFactor=1.05, minNeighbors=7,
	minSize=(30, 30),
	flags=cv2.CASCADE_SCALE_IMAGE)
	# tupla[3] = len(faces)
	relatorio = relatorio + " {}: {}".format(tupla[2], len(faces))
		Aqui é executada a detecção de rosto onde, na variável *faces* é definido a escala de procura de rostos, quantos retângulos precisam ser detectados para afirmar que é um rosto e o tamanho mínimo de contorno do rosto.

14	for (x, y, w, h) in faces:
	posicao=Posicao(x,y,w,h)
	forcaTreino=False
	# cv2.rectangle(img, (x, y), (x + w, y + h), tupla[1], 7)
	cv2.rectangle(mascara, (x, y), (x + w, y + h), 255, -1)
	size = (w, h)
	center = (w // 2, h // 2)
	pos=(x,y)
	
	roi_gray = iPB[y:y + h, x:x + w]
	roi_color = img[y:y + h, x:x + w]
	roi_color2 = roi_color.copy()
	
	roi_eyes = []
	if (inverter):
	roi_color[:, :] = roi_color[:, ::-1]
	
	eyes = dfEyes.detectMultiScale(
	roi_gray,
	scaleFactor=1.01,
	minNeighbors=posicao.w//10, #minNeighbors=22,
	minSize=(posicao.w // 15, posicao.w // 15), #minSize=(25, 25),
	flags=cv2.CASCADE_SCALE_IMAGE
	)
		Aqui desenha retângulos amarelos na imagem original (colorida). Pega posição, tamanho, centro. Utiliza a conversão em escala de cinza e o colorido, além de criar uma cópia do colorido.
		Cria a variável *eyes*, que será utilizada da mesma forma que a variável *faces*.

15	if (len(eyes) >= 1):
	#print("achou", len(eyes), "olhos")
	for (xEye, yEye, wEye, hEye) in eyes:
	roi_eyes = roi_color[yEye:yEye + hEye, xEye:xEye + wEye].copy()
	if not blur:
		Esse bloco representa um contador. Ele verifica quantos olhos há na imagem e desenha retângulos onde os encontra.

```
                #                   cv2.rectangle(roi_eyes, (0, 0), (0+wEye,
0+hEye), (0, 255, 0), 1)
                #               else:
                cv2.rectangle(roi_color, (xEye, yEye), (xEye + wEye, yEye +
hEye), (0, 255, 0), 1)
```

16

```
smile = dfSmile.detectMultiScale(
            roi_gray,
            scaleFactor=1.7,
            minNeighbors=22,
            minSize=(25, 25),
            flags=cv2.CASCADE_SCALE_IMAGE
        )
```

A variável *smile* detectará se a pessoa está sorrindo.

17

```
roi_mouth = []
        resized = roi_color
        #print("Achou", len(smile), "sorriso(s)!")
for (x, y, w, h) in smile:
            cv2.rectangle(roi_color, (x, y), (x + w, y + h), (255, 0, 0), 1)
            roi_mouth = roi_color[y:y + h, x:x + w].copy()
```

Desenha um retângulo onde a boca é detectada.

18

```
if (blur):
            roi_colorBlur = cv2.medianBlur(roi_color, 23)
            if len(smile) == 1:
                roi_colorBlur[y:y + h, x:x + w] = roi_mouth
            # if len(eyes)>=1:
            for (x, y, w, h) in eyes:
                cv2.rectangle(roi_color, (x, y), (x + w, y + h), (0, 255, 0), 1)
                roi_colorBlur[y:y + h, x:x + w] = roi_color[y:y + h, x:x + w]
            roi_color[:, :] = roi_colorBlur[:, :]
```

Aqui é utilizado o *blur* para borrar a imagem. Existem três filtros: *GaussianBlur*, *medianBlur* e *bilateral Filter A*. Nesse caso está sendo utilizado o *medianBlur*.

19

```
if len(eyes) == 2:
            x1 = eyes[0][0]
            x2 = eyes[1][0]
            y1 = eyes[0][1]
            y2 = eyes[1][1]
            dx = x2 - x1
            dy = y2 - y1
            rads = atan2(-dy, dx)  # -pi a +pi
            rads = 0
            if (dy != 0):
                rads = atan(float(dx) / float(dy))
            else:
                rads = pi / 2
            # rads %= pi/2
            degs = degrees(rads % (pi))
            # if degs>180:
            #    degs-=180
            #print("Angulo: ", degs, "Rad: ", rads, "Rad%(pi/2): ", rads % (pi))
            M = cv2.getRotationMatrix2D(center, 90 + (degs * -1), 1.0)

            cv2.line(roi_color, (eyes[0][0] + eyes[0][2] // 2, eyes[0][1] +
eyes[0][3] // 2),
                     (eyes[1][0] + eyes[1][2] // 2, eyes[1][1] + eyes[1][3] // 2), green)

            rotated = cv2.warpAffine(roi_color2, M, size)
            r = 200.0 / rotated.shape[1]
            dim = (200, int(rotated.shape[0] * r))
            resized = cv2.resize(rotated, dim, interpolation=cv2.INTER_AREA)
            cv2.imshow("rotated", resized)

        (achou, label_predicted, conf) = encontraRosto(resized)
```

Aqui é verificado se existem dois olhos no rosto.
O ângulo dos olhos é ajustado.
São desenhados retângulos na posição dos olhos.
Define M como a matriz de rotação, sendo (centro da imagem, ângulo, escala) e *rotated* pega a imagem *image* rotacionada conforme a matriz M.

```
            cadAnt = temIdCadastrada("faces/", label_predicted)
if (achou):
            print("Rosto encontrado, ID {}".format(label_predicted))
            if nomeNaFace:
               escreveNome(img, "{} - {}".format(label_predicted, cadAnt[2]),
pos)
            rosto=Rosto(True,label_predicted,conf, posicao, cadAnt[2])
            if rosto.id in rostosNoCache:
               if
rostosNoCache[rosto.id].posicao.module(rosto.posicao)>rostosNoCache[rosto.i
d].posicao.w/10:
            rostosNoCache[rosto.id]=rosto


         else:
20          print("Rosto nao encontrado, ID {}, treinar nova
ID".format(label_predicted))
            for lRosto in rostosNoCache:
               if posicao.hittest(rostosNoCache[lRosto].posicao):
                  (achou2, label_predicted2, conf2) = encontraRosto(resized,
tratado=False)
                  print("achou!", rostosNoCache[lRosto].id, label_predicted2,
conf2)
                  if label_predicted2==rostosNoCache[lRosto].id and conf2<90:
                     achou = True
                     label_predicted=label_predicted2
                     forcaTreino=True
                     cadAnt = temIdCadastrada("faces/", label_predicted)
file = ""
```

Caso o rosto que tenha sido detectado já esteja armazenado na rede neural, o bloco imprimirá que o rosto foi encontrado juntamente com a ID correspondente.
Se o rosto ainda não foi cadastrado, imprimirá que o rosto não foi encontrado e treinará uma ID para ele.

```
if ((treina and (not achou or 10 < conf < 40) or(not achou and conf > 100) or
forcaTreino) and len(eyes) == 2):
            forcaTreino=False
            #achou=True
            #label_predicted=0

            if not achou:
               file = "";
               if(label_predicted!=-1):
                  for p in range(maxId,1000):
21                   maxId=max(maxId, p)
                     if temIdCadastrada("faces/",p)[0]:
                        continue
                     newDir = "faces/{}.nome/".format(p)
                     prefix=newDir + "pessoa{}.".format(p)
                     os.mkdir(newDir)
                     if (not (os.path.isfile(prefix+"0.bmp"))):
                        file = achaId(prefix)
                        print("Novo cadastro: {}".format(p))
                        label_predicted=p
                        break
```

Esse bloco realiza o cadastro de novos rostos não encontrados no banco de dados (*if not achou*), cria uma pasta com a nota ID (*for p in range(maxId,1000)*), e já retorna o nome do arquivo onde será salvo o frame do rosto (*file = achaId(prefix)*).

```
else:
            lastSeq=pegaLastSeqUsada(cadAnt[1])
            if cadAnt[2]=="nome":
               #prefix = "faces/pessoa{}.".format(label_predicted)
22             prefix = cadAnt[1]+"/pessoa{}.".format(label_predicted)
            else:
               prefix = "faces/pessoa{}.".format(label_predicted)
               lastSeq+=1
            file = achaId(prefix, lastSeq)
```

Outro porém aplicado aqui seria no caso de haver um cadastro novo. Se isso, então as imagens são salvas diretamente na pasta da pessoa. Se o cadastro for antigo, a pasta da pessoa é mantida e as imagens são salvas no diretório raiz.

```
            print("Atualizando cadastro: {}".format(label_predicted))
        if(file!=""):
            resized2 = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)
            cv2.imwrite(file, resized2)
        if(label_predicted!=-1):
            atualizaNovoRosto(resized, label_predicted)
```

23

```
img_mask = cv2.bitwise_and(iPB, iPB, mask=mascara)
    cv2.imshow("mascara", img_mask)

    # smiles = dfSmile.detectMultiScale(img,
    #                 scaleFactor = 1.05, minNeighbors = 7,
    #                                         minSize = (30,30), flags =
cv2.CASCADE_SCALE_IMAGE)
    # for (x, y, w, h) in smiles:
    #    cv2.rectangle(img, (x, y), (x + w, y + h), tupla[1], 7)

  #cv2.putText(img, relatorio, (10, 10), cv2.FONT_HERSHEY_SIMPLEX,
0.5, 255)
    # cv2.imshow(str(len(faces))+' face(s) encontrada(s).', img)
```

A máscara aplicada na imagem faz um *and* entre a cor das duas imagens e se tiver uma máscara, examina só o que coincide com branco. Logo após exibe a imagem e o título da janela exibe o número de faces.

24

```
k=0
  maxX=0
  newMax=0
  offsetX, offsetY=10,20
  for i in sorted(rostosNoCache.items(), key=operator.itemgetter(0)):
#r=rostosNoCache[i]
    r=i[1]
    p=r.posicao
    x,y,w,h = p.x, p.y, p.w, p.h
    cv2.rectangle(img, (x, y), (x + w, y + h), (255, 255, 255), 1)
    if not nomeNaFace:
      text="{} - {}".format(r.id, r.nome)
      pos=pegaTamanhoNome(text)[0]
      x=offsetX+maxX
      y=offsetY+(pos[1]+5)*k
      if y>img.shape[0]-offsetY:
        k=0
        maxX=newMax
        x = offsetX + maxX
        y = offsetY * 2 + pos[1] * (k)

      escreveNome(img, text, (x, y) )
      cv2.line(img, (x+pos[0], y), p.pos(), (0,0,0), thickness=5,
lineType=8, shift=0)
      newMax=max(newMax,x+pos[0])

      k+=1
```

Esse bloco é responsável por escrever os nomes das pessoas no canto superior esquerdo da tela, um abaixo do outro. A outra opção é escrever logo acima dos rostos.
Para usar essa opção tem a tecla de atalho *n* que alterna a posição dos nomes.
Tem um *offset* para cada nome ficar em uma linha.

25

```
menu="i-Inverter|" \
      "b-Borrar Rosto|" \
      "t-Treinar IA|" \
      "n-intercalar posicao dos nomes|" \
      "s-salvar rostos|" \
      "q-Sair"

  pos = pegaTamanhoNome(menu, fontScale=0.4, thickness=1)[0]
  x = img.shape[1]-pos[0]
  y = img.shape[0] - pos[1]
  escreveNome(img, menu, (x, y), fontScale=0.4, thickness=1, color=(100,
255, 100))
```

Aqui são definidos os ‘atalhos de teclado’, onde: *i* inverte a imagem, *b* borra o rosto, *t* treina o rosto da pessoa na rede neural, *n* altera a posição dos nomes, *s* salva os rostos detectados, e *q* sai da aplicação.

	cv2.imshow(' face(s) encontrada(s).', img)	

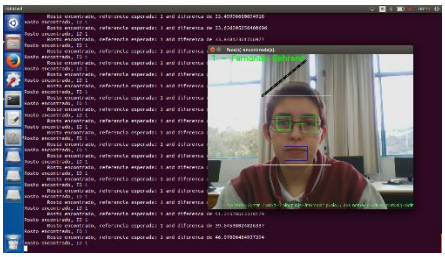
26	key = chr(cv2.waitKey(1) & 0xFF)
	if (key == 'q'):
	break
	elif (key == 'i'):
	inverter = not inverter
	elif (key == 'b'):
	blur = not blur
	elif (key == 't'):
	treina = not treina
	elif (key == 'n'):
	nomeNaFace = not nomeNaFace
	elif (key == 's'):
	ensure_dir('faces/')
	for i in range(100):
	if (not (os.path.isfile('faces/pessoa5.' + str(i) + '.bmp'))):
	resized = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)
	cv2.imwrite("faces/pessoa5." + str(i) + '.bmp', resized)
	break
		Aqui são definidos os atalhos de teclado.
		A letra *q* é usada para sair do sistema.
		A letra *i* inverte os rostos no eixo x.
		A letra *b* embaça os rostos.
		A letra *t* treina novos rostos (pode aumentar drasticamente o tamanho do banco de dados).
		A letra *n* alterna a posição do *label* dos nomes no canto superior esquerdo, ou acima do rosto detectado.

27	salvaNeuralNetwork(arqRedeNeural); cap.release() cv2.destroyAllWindows()	Salva a rede neural. Quando tudo estiver pronto, libera a captura.

Fonte: Os autores

Tudo, na prática, pode ser feito através do código mostrado na Tabela 1. Tanto a captura dos rostos para salvar no banco de dados quanto a detecção dos mesmos pode ser feita através desse código. A aplicação desse código seria bem simples e universal, podendo ser utilizado em academias, instituições de ensino para atribuir presença aos estudantes, etc.

Figura 1- Sistema em execução



Fonte: Os autores (2017)

3. RESULTADOS E DISCUSSÕES

Segundo os testes realizados após o término do programa, a detecção facial desenvolvida tem entre 90% e 95% de precisão. Os problemas encontrados para que ele não seja 100% eficaz, são ligados a iluminação e a pose que a pessoa faz na frente da câmera, pois, dependendo de como o rosto está disposto, o sistema as vezes não consegue reconhecê-lo mesmo já tendo sido cadastrado no banco, fazendo com que um novo diretório seja criado com as fotos dessa pessoa.

4. CONSIDERAÇÕES FINAIS

Desta forma podemos observar que o sistema para reconhecimento de rostos apresenta grande praticidade. Para cadastrar os rostos basta apenas ficar na frente de uma câmera e apertar a letra ‘t’, quando o software estiver em execução, para que sua face seja cadastrada. No momento em que o indivíduo volte a passar na frente dessa câmera com o programa sendo executado, seu rosto, caso cadastrado, será reconhecido e uma imagem será salva no diretório raiz mesmo já tendo um diretório para si e este sendo mantido salvo. Caso a pessoa não seja reconhecida pelo sistema, seu rosto poderá ser treinado e as imagens correspondentes serão salvas em uma outra pasta nomeada como *id.nome*. Por fim, gostaríamos de agradecer o IFC Luzerna por financiar este projeto via edital de fomento à pesquisa 12/2016.

REFERÊNCIAS BIBLIOGRÁFICAS

AMARAL, Vagner do; THOMAZ, Carlos Eduardo. **Extração e Comparação de Características Locais e Globais para o Reconhecimento Automático de Imagens de Faces.** Disponível em: <http://fei.edu.br/~cet/VagnerAmaral_WVC2012.pdf>. Acesso em: 04 ago. 2017.

OPENCV. **FACE Detection using Haar Cascades.** Disponível em: <http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html>. Acesso em: 14 jul. 2017a.

OPENCV. **FACE Recognition with OpenCV.** Disponível em: <http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html>. Acesso em: 09 jun. 2017b.

PYTHON: **O que é? Por que usar?** Disponível em: <http://pyscience-brasil.wikidot.com/python:python-oq-e-pq>. Acesso em: 29 mar. 2017.

RASPBERRY PI. **O que é o Raspberry Pi?** Disponível em: <http://bodgarage.repofy.com/?p=439>. Acesso em: 12 jul. 2017.

ROSEBROCK, Adrian. **Install OpenCV 3.0 and Python 3.4+ on Ubuntu.** 2015. Disponível em: <http://www.pyimagesearch.com/2015/07/20/install-opencv-3-0-and-python-3-4-on-ubuntu/>. Acesso em: 05 ago. 2017.

TECHTUDO. **VirtualBox: baixe e instale outros sistemas operacionais.** 2016. Disponível em: <http://www.techtudo.com.br/tudo-sobre/virtualbox.html>. Acesso em: 04 ago. 2017.