Computação Gráfica
Processamento de Imagem
Contagem de Pessoas em uma Multidão

Douglas Fernando Raymundo 6791899
Antonio Carlos Mateus da Silva 8516031
Giovane Melo Andrade 8516688
Luís Felipe de Almeida da Silva 8516775

Esse projeto consiste na implementação de um algoritmo para contar quantas pessoas há em uma imagem. A classe principal do programa é **face_detect.py.** Para executar a detecção de pessoas basta:

```
$ cd <face_detecting_root>
$ python face_detect.py <image_name>
```

O parâmetro <face_detectin_root> é o diretório raiz do projeto onde se encontra o “face_detect.py” e <image_name> é o nome de qualquer imagem que você deseja aplicar o algoritmo que esteja no diretório raiz do projeto. Para executar é necessário ter o Python 2.7 ou maior instalado e o openCV 2.4 ou maior.

Na base do algoritmo temos um **Haar Classificador em Cascata** para identificar os rostos das pessoas na imagem. Ele é uma abordagem baseada em aprendizado de máquina que foi proposto por Paul Viola e Michael Jones no *paper* "Rapid Object Detection using a Boosted Cascade of Simple Features" de 2001.

Para treinar esse classificador é necessários passar dois tipos de conjunto de imagens para que ele possa extrair as *features* (características) desejadas: conjunto de imagens positivas, que contém o objeto que você quer reconhecer, e conjunto de imagens negativas, que não contêm os objetos que você quer reconhecer.

Em cima desses conjuntos de imagens, aplica-se kernels de convolução para extrair características das bordas, linhas e diagonais da imagens a partir das soma dos pixels.

No openCV, esse classificador é definido em um .xml e nativamente já há alguns classificadores treinados para determinadas tarefas. Na versão 2.4 do openCV, o classificador de faces já está disponível e podemos usá-los sem ter o arquivo na pasta do projeto. Porém, ele está contido na pasta do nosso projeto, caso a versão do openCV seja 3.+. Tendo o nome do arquivo do classificador é simples criar um classificador, no projeto:

```
# Create the haar cascade
# the path of .xml of Face CascadeClassifier
cascPath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascPath)
```

O classificador varre a imagem e retorna uma coleção de retângulos posicionados em cima de cada uma das faces detectadas. É necessário alguns parâmetros para executar o detector:

```
# Detect faces in the image
faces = faceCascade.detectMultiScale(
    gray,
    scaleFactor = scale,
    minNeighbors = neighbors,
    minSize = (minSize, minSize),
    maxSize = (maxSize, maxSize)
)
totalFacesDetected += len(faces)
```

- <gray> é a imagem (em escala de cinza porque é o suficiente para fazer a detecção e isso poupa processamento);
- <scale> é um fator de escala para detectar as diferentes faces que estão mais longe ou mais perto da câmera;
- <minNeighbors> é a quantidade mínima de faces que podem estar vizinhas de cada uma;
- <minSize> é o tamanho mínimo em altura e largura de cada face na imagem que se quer detectar;
- <maxSize> é o tamanho máximo em altura e largura de cada face na imagem que se quer detectar.

O programa aplica a detecção da imagem a cada vez que rotacionamos a imagem e rotacionamos a imagem em diferentes ângulos. Eles são definidos em uma lista para detectarmos faces em cabeças que estão inclinadas em diferentes ângulos.

```
totalFacesDetected = 0
#all the angles that the faces can be inclined
angles = range(40, -40, -5)
for rotate in angles:
    #rotate the image to detect inclined faces
    image = rotate_image(image, rotate)
    #Convert image to gray scale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

No projeto, utilizamos o detector com vários parâmetros diferentes em uma única execução do programa colocando-os fixos no código que podem ser modificados. Os três primeiros parâmetros referem-se aos parâmetros utilizados para aplicar o classificador em cascata em cada imagem e o último diz se será aplicada uma operação (definida em *multiply_v()* que será explicada mais a frente) para que a imagem recupere a qualidade original após a rotação. Segue o conjunto de parâmetros pre-definidos:

```
#face cascade params
params = [
#[neighbors, scale, minSize, keepQualityAfterRotation]
    [14, 1.5, 15, True],
    [14, 1.5, 15, False],
    [12, 1.5, 15, True],
    [12, 1.5, 15, False],
    [10, 1.5, 15, True],
    [10, 1.5, 15, False],
    [8, 1.5, 15, True],
    [8, 1.5, 15, False],
    [1, 1.5, 15, True],
    [1, 1.5, 15, False],
```

O programa gera, para cada um dos parâmetros no conjunto, uma imagem com quadrados pretos no lugar das faces detectadas e um .txt que contém o número de faces detectadas em cada iteração e quantas foram detectadas no total.

Nos nossos testes utilizamos duas imagens: “crowd0.jpg” e “crowdA.jpg”. Para a foto crowd0.jpg, o melhor parâmetro que conseguimos foi [neighbors = 8, scale = 1.1, minSize = 15, keepQualityAfterRotation = True]. Detectou-se apenas duas faces falsas, uma mais a direita da imagem e outra mais a esquerda:

Imagem: crowd0/crowd0_n=8_s=1.1_mins=15_maxs=100_kqar=True_total=39.jpg

Para a foto crowdA.jpg, o melhor parâmetro que conseguimos foi [neighbors = 8, scale = 1.1, minSize = 15, keepQualityAfterRotation = True]. O algoritmo só não detecta uma das pessoas a direita da imagem.

Imagem: crowdA/crowdA_n=8_s=1.1_mins=15_maxs=100_kqar=True_total=28.jpg

Os principais problemas encontrados foram:

- A perda de qualidade que a imagem tem após a rotação dela, que deixava as faces com traços menores mais difíceis de reconhecer em cada iteração, que foi resolvido com aplicação do método *multiply_v()* pós detecção de faces. Esse método aplica o *and* de duas imagens: uma é a imagem original; e outra a imagem com as faces detectadas e com um threshold que deixa branco todos os pixels que não são absolutamente pretos. Dessa forma “recuperamos” a qualidade dos pixels que não foram marcados com os retângulos pretos;
- As cabeças que estão inclinadas. O método de detecção do classificador não detecta esses rostos, portanto rotacionamos a imagem em diversos ângulos para melhorar a eficácia do algoritmo e marcamos as faces já detectadas com retângulos totalmente pretos para que eles não sejam detectados nas próximas rotações; e por último

- Enfrentamos o problema de que a rotação corta a imagem nas extremidades, por isso acrescentamos um *padding* à imagem para rotacionarmos livremente.

O código, as imagens de testes e os resultados podem ser encontrados no CustomFaceDetectin.zip que estão junto com esse relatório e no repositório público do github: https://github.com/luisfelipeas5/CustomFaceDetecting. Utilizamos como ajuda os tutorias presentes em:

- http://docs.opencv.org/master/d7/d8b/tutorial_py_face_detection.html#gsc.tab=0; e
- https://realpython.com/blog/python/face-recognition-with-python/