**Detecção de Faces com a biblioteca OpenCV**

Uma breve introdução intuitiva a biblioteca OpenCV com uma “pitada” de hands-on utilizando um classificador já treinado.

[](https://dataml.com.br/deteccao-de-faces-com-a-biblioteca-opencv/)

[Matheus](https://dataml.com.br/author/matheus-de-paula/)

18 Dez, 2020 - 6 min

Foto em preto e branco de homens

Descrição gerada automaticamente com confiança média

[**Facebook**](https://www.facebook.com/sharer/sharer.php?u=https://dataml.com.br/deteccao-de-faces-com-a-biblioteca-opencv/&display=popup&ref=plugin&src=share_button)[**LinkedIn**](https://www.linkedin.com/shareArticle?mini=true&url=https://dataml.com.br/deteccao-de-faces-com-a-biblioteca-opencv/)[**Tweet**](https://twitter.com/share?url=https://dataml.com.br/deteccao-de-faces-com-a-biblioteca-opencv/&text=Detec%C3%A7%C3%A3o%20de%20Faces%20com%20a%20biblioteca%20OpenCV)

Neste artigo:

* Breve introdução a OpenCV
* O classificador e o processo cascata
* Principais parâmetros do classificador
* Identificação de faces com a OpenCV
* Identificação de face com a OpenCV via webcam

Breve introdução a OpenCV

A biblioteca OpenCV (*Open Source Computer Vision*), é a mais popular utilizada na linguagem Python, quando estamos falando de detecção de faces além de outras distintas aplicações no dia-a-dia, tais como: controle de tráfegos, tamanho de um objeto, alarmes e segurança pública em geral, controle de quantidade de pessoas em determinado ambiente, etc.

Para que possamos introduzir essa ferramenta mágica, precisamos primeiramente saber e compreender os tipos de detecção ou reconhecimento facial que existem:

* **Geométrica:** a detecção geométrica é a mais comum utilizada para o reconhecimento facial, basicamente são considerados diversos “marcos” para que se possa ocorrer o reconhecimento a partir de dados como a distância entre o nariz e os olhos, por exemplo. Ademais, vale dizer que esse tipo é bastante robusto mesmo com pouca iluminação no local de análise.
* **EigenFaces:** essa técnica basicamente utiliza PCA (*Principal Component Analysis*). Em síntese, nesse método cada face é representada por uma combinação linear entre de diversas “EigenFaces”.
* **Abordagens de Machine Learning:** onde utilizamos o “pipeline” padrão de qualquer projeto de dados, dividindo o dataset em treino/teste e subtemos uma série de imagens para realizar a classificação das imagens.
* **Cascade (Viola-Jones):** que é a mais comum para a detecção de imagens em geral.

Ao longo desse artigo, daremos enfase ao método Cascade (Viola-Jones), onde será possível compreender melhor sua implementação além de seu funcionamento.

O classificador e o processo cascata

Em tese, para a utilização do algoritmo começamos carregando em nosso código diversas imagens contendo faces de seres humanos (imagens positivas) e um segundo conjunto de imagens que vão conter diversas “não faces”, tais como: objetos, animais, vale colocar até foto de Et’s se preferir (imagens negativas).

Após essa etapa, é realizado o treinamento com esses conjuntos de dados utilizando o algoritmo AdaBoost que realiza a seleção do melhor conjunto de características através dos pixels (unidade mínima) da imagem, removendo imagens desnecessárias e combinando vários classificadores fracos. No nosso site temos um [artigo](https://dataml.com.br/extreme-gradient-boosting-xgboosting-um-overview-geral/) onde ensinamos de forma breve a poderosa técnica de boosting, entretanto no presente não daremos ênfase ao funcionamento desse algoritmo.

Por fim, é aplicado para a nova imagem diversas “subjanelas” ou subdivisões com o objetivo de identificar a face na nova imagem.

Na imagem abaixo, pode-se observar de forma clara um “pepiline” de como é realizado a classificador cascade utilizando a famosa abordagem supervisionada (quando fornecemos ao modelo dados para identificar os padrões).

Diagrama

Descrição gerada automaticamente com confiança médiaClassificador Cascade

O processo Haar Cascade em síntese vai funcionar a partir dos seguintes passos (3 passos principais):

1. **Deslizar** pela imagem em torno dos pixels presentes na mesma.
2. **Realizar uma média** dos valores dos pixels que estão presentes nas áreas brancas e pretas.
3. **Fazer a diferença entre as áreas** (haar), sendo que caso for menor que um valor limite (limiar) a característica vai coincidir.

Na figura abaixo fica mais claro o esquemático realizado pelo algoritmo, onde ao longo dos classificadores (C1, C2, C3, CN) caso não exista determinada característica já é descartada a detecção e assim por diante a partir desse processo de “cascata”.

Diagrama

Descrição gerada automaticamentePipeline do modelo “cascata”

Assim, caso o algoritmo passe de todas as características irá ocorrer a detecção da face.

Principais parâmetros do classificador

Para o classificador temos três parâmetros principais que podem ser configurados com o objetivo principal de melhorar a detecção, são eles:

**Scale Factor**

O parâmetro scale factor, como o próprio nome já diz se refere ao fator de escala que adotamos para imagem.

Esse parâmetro vai basicamente “controlar” a detecção a partir de uma escala adotada para detectar por exemplo pessoas que estão mais afastadas de uma determinada câmera do que outras na própria imagem.

Vale dizer, que quanto menor o valor adotado mais lenta será a detecção, justamente por estar em uma escala menor.

**Min Neighbors**

Esse parâmetro vai ter como objetivo “detectar” o mínimo de candidatos possíveis que determinada face pode ter de acordo com a distância entre as mesmas. Assim, vale dizer que basicamente teremos 2 condições:

* **Valores mais altos:** proporcionam menos detecções, porém com maior qualidade. Nesse caso, podemos imaginar que pessoas que estão mais distantes necessitarão de valores mais altos de distância.
* **Valores mais baixos:** proporcionam mais detecções pois identifica vários candidatos, caso os mesmos estejam próximos. De uma forma mais simples, pessoas que estão mais próximas não necessitam de uma distância tão grande para terem suas faces detectadas.

**Min Size**

Esse parâmetro especifica o menor objeto a ser reconhecido. O valor default (padrão) que vem configurado na biblioteca é 30×30 pixels, isso quer dizer que se a face for menor que essa dimensão o algoritmo não consegue reconhecer.

Identificação de faces com a OpenCV

Agora que vimos as principais aplicações e aprendemos de forma intuitiva o funcionamento do algoritmo e do Haar Cascades vamos utilizar a biblioteca OpenCV com um classificador de faces já treinado com o objetivo de detectar as faces da maneira mais precisa possível.

Assim, caso não tenha a biblioteca instalada primeiramente deve-se acessar o anaconda prompt no terminal de comando e digitar o seguinte comando:

pip install opencv-python

Basicamente para o nosso exemplo prático utilizaremos um arquivo XML com um classificador já treinado para detecção de faces que pode ser adquirido nesse [link](http://alereimondo.no-ip.org/OpenCV/34/).

Vamos analisar uma foto que eu tirei no Rio de Janeiro juntamente a um ponto turístico clássico do Brasil, o Cristo Redentor.

Será que o algoritmo vai ser capaz de identificar somente o meu rosto e não o do Cristo?

É o que veremos por agora…

Basicamente iremos importar a biblioteca, ler a imagem, converter para a escala de cinza e utilizar o classificador já treinado para ver quantas faces estão sendo detectadas na foto.

## Carregando a OpenCV

**import** cv2

## Carregando o classificador já treinado

classificador = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

## Lendo a imagem

imagem = cv2.imread('cristo.jpg')

## Convertendo para cinza

imagemcinza = cv2.cvtColor(imagem, cv2.COLOR\_BGR2GRAY)

## Detectando com o classificador já treinado

facesdetectadas = classificador.detectMultiScale(imagemcinza,scaleFactor=1.1, minNeighbors=8,minSize=(30,30)) # parâmetros principais

## Número de Faces Detectadas

print(len(facesdetectadas))



Fica evidente dizer então que o Haar Cascade está identificando somente a minha face (provavelmente). Assim, por agora vamos plotar a imagem e realizando a detecção a partir de retângulos vermelhos.

## Observando a matriz e detectando as faces com retângulos

**for** (x, y, l, a) **in** facesdetectadas: ## eixos x e y, largura (l) e altura (a)

## (0,0,255) valores RGB

## 2 representa a largura da borda a ser pintada

cv2.rectangle(imagem, (x,y), (x+l, y+a), (0, 0, 255), 2) ## retângulo

## Mostrando a imagem

cv2.imshow("Faces Encontradas", imagem)

cv2.waitKey()

Uma imagem contendo pessoa, no interior, homem, em pé

Descrição gerada automaticamenteFace detectada pelo classificador

Como pode ser observada a minha face foi detectada e a do Cristo Redentor não (como esperado).

Caso quiséssemos tentar detectar a imagem da mulher ao fundo que está olhando para o lado, seria necessário variar os parâmetros com o objetivo principal de detectar sua face (lembrando que o algoritmo pode não ser preciso e acabar não conseguindo).

Identificação de faces com a OpenCV via webcam

Para a detecção da face em tempo real via webcam, realizamos um processo similar ao código realizado para a detecção em imagens.

Basicamente carregaremos o nosso classificador face (arquivo XML) e iniciaremos a câmera automaticamente. Dentro de um loop while iremos manter o vídeo ligado e utilizar o classificador de maneira similar a feita anteriormente e como condição de saída do vídeo colocaremos a tecla ‘s’.

## Captura de vídeo

video = cv2.VideoCapture(0)

## Classificador

classificadorFace = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

## Loop

**while** **True**:

## mantendo conectada a webcam e os frames

conectado, frame = video.read()

## transformando em escala de cinza

frameCinza = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

## detectando a face com o classificador treinado xml

facesDetectadas = classificadorFace.detectMultiScale(frameCinza,minSize=(20,20))

**for** (x,y,l,a) **in** facesDetectadas:

cv2.rectangle(frame, (x,y), (x+l, y+a), (0,0,255), 2)

cv2.imshow('Video',frame)

## Fechar janela do video

**if** cv2.waitKey(1) == ord('s'):

**break**

## Liberar a captura

video.release()

## Liberar memória

cv2.destroyAllWindows()

Uma imagem contendo no interior, pessoa, vestindo, segurando

Descrição gerada automaticamenteFace detectada em tempo real via webcam

Assim, fica evidente dizer que o classificador está funcionando perfeitamente, obviamente pode acontecer de pessoas mais longe não serem detectadas e para que ocorra a detecção seja necessário realizar o ajuste dos parâmetros apresentados.

Resumo

Nesse artigo foram apresentados de forma sucinta as principais abordagens para realizar detecções e reconhecimentos com o uso da biblioteca OpenCV. Foi dado ênfase ao classificador Haar Cascades que é o processo mais comum para a detecção em imagens.

Além disso, foi possível ver com uma “pitada” de hands-on como é intuitivo a utilização de classificadores já prontos e como eles podem ser eficientes para tarefas do dia-a-dia como sistemas de segurança, por exemplo.

Referências

[1] VIOLA, Paul; JONES, Michael J. Robust real-time face detection. **International journal of computer vision**, v. 57, n. 2, p. 137-154, 2004.

[2] “Simple Face Detection in Python”, Towards Data Science. Acesso: <[link](https://towardsdatascience.com/simple-face-detection-in-python-1fcda0ea648e)>

Eai, curtiu as detecções e o artigo? Deixe seu feedback abaixo ou a sua dúvida se tiver e até a próxima 😉