Julio Batista Silva

Explorando o algoritmo de Viola-Jones na detecção e reconhecimento facial

São Carlos, Brasil

2018

Julio Batista Silva

Explorando o algoritmo de Viola-Jones na detecção e reconhecimento facial

Trabalho de conclusão do curso de graduação em engenharia de computação submetido ao corpo docente do Departamento de Computação da Universidade Federal de São Carlos como parte dos requisitos necessários para obtenção do título de engenheiro de computação.

Universidade Federal de São Carlos – UFSCar

Departamento de Computação

Engenharia de Computação

Orientador: Prof. Dr. Alexandre Luis Magalhães Levada

São Carlos, Brasil

2018

Julio Batista Silva
Explorando o algoritmo de Viola-Jones na detecção e reconhecimento facial/ Julio Batista Silva. – São Carlos, Brasil, 2018-
90 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Alexandre Luis Magalhães Levada

Trabalho de Conclusão de Curso – Universidade Federal de São Carlos – UFSCar
Departamento de Computação
Engenharia de Computação, 2018.

1. Detecção facial. 2. Viola-Jones. I. Alexandre Luiz Magalhães Levada. II. Universidade Federal de São Carlos. III. Departamento de Computação. IV. Explorando o algoritmo de Viola-Jones na detecção e reconhecimento facial

CDU 02:141:005.7

Julio Batista Silva

Explorando o algoritmo de Viola-Jones na detecção e reconhecimento facial

Trabalho de conclusão do curso de graduação em engenharia de computação submetido ao corpo docente do Departamento de Computação da Universidade Federal de São Carlos como parte dos requisitos necessários para obtenção do título de engenheiro de computação.

Trabalho aprovado. São Carlos, Brasil, 12 de julho de 2018:

Prof. Dr. Alexandre Luis Magalhães Levada
Orientador

Prof. Dr. Fredy João Valente
Membro da banca

Profª. Drª. Marcela Xavier Ribeiro
Membro da banca

São Carlos, Brasil
2018

Dedico este trabalho aos meus pais, Cesar de Souza e Silva e Fátima Aparecida Batista Silva, por todo o amor, incentivo aos estudos e apoio, que foram fundamentais para a realização deste trabalho.

Agradecimentos

Agradeço aos meus pais, Cesar e Fátima, que sempre me apoiaram e me ajudaram durante a graduação.

À minha namorada, Louise Lobão, por todo o amor, carinho e companheirismo, que foram fundamentais para me manter motivado.

Ao meu orientador, Alexandre Levada, por sempre estar disposto a me ajudar respondendo meus e-mails de forma rápida e atenciosa com materiais e conselhos valiosos que me guiaram na escrita deste trabalho.

À Universidade Federal de São Carlos por oferecer recursos e acesso a grandes professores, que foram essenciais para a minha formação.

Aos meus veteranos, calouros e colegas de curso que, de alguma forma, contribuíram para a minha graduação.

Aos amigos que fiz durante o intercâmbio na Alemanha e à CAPES pela bolsa que me sustentou naquela que foi uma das melhores épocas da minha vida.

Ao meu gestor, Fillipe Silva, por, de certa forma, ter sugerido o tema do projeto e me ajudado a equilibrar meu tempo entre trabalho e estudo.

À Visagio por ter me colocado em contato com tanta gente boa e do bem que têm contribuído muito para o meu desenvolvimento pessoal e profissional.

Aos criadores do abnTEX por deixar este trabalho tão bonito e bem formatado.

Aos meus gatos, Amélia e Joaquim, que me fizeram companhia durante a escrita desta monografia.

"Ut imago est animi voltus sic indices oculi"
— Marcus Tullius Cicero

Resumo

As inúmeras aplicações das técnicas para detecção e reconhecimento facial têm chamado muita atenção de empresas e governos. Esse crescente interesse pelo assunto atraiu investimentos em pesquisas e resultou em progressos significantes no desenvolvimento de novos métodos, bibliotecas, produtos e serviços.

Apesar de muitas dessas ferramentas serem descritas como simples e utilizáveis sem a necessidade de conhecimentos prévios em visão computacional, um embasamento teórico permite escolher as tecnologias apropriadas e usá-las de forma otimizada, considerando suas capacidades e limitações.

Este trabalho introduz as áreas de detecção e reconhecimento facial através de uma extensa revisão bibliográfica, que fornece uma visão geral sobre inúmeros métodos encontrados na literatura e apresenta uma coletânea de recursos úteis ao treinamento de classificadores e validação dos algoritmos.

Também é feito um estudo mais aprofundado sobre Viola-Jones e Eigenfaces ao apresentar o projeto e a implementação de um sistema capaz de detectar e reconhecer faces construído através da combinação desses métodos. É mostrado que a taxa de detecção do Eigenfaces não é suficiente para o uso desejado e são propostas alternativas.

O primeiro módulo desse sistema é executado em um Raspberry Pi e é um exemplo de como aliar conhecimento teórico, bibliotecas open source, ferramentas comerciais e hardware para a criação de um produto lucrativo.

Palavras-chaves: Detecção facial. Viola-Jones. Reconhecimento facial. Eigenfaces. Raspberry Pi.

Abstract

The numerous applications of facial detection and recognition techniques have attracted much attention from companies and governments. This growing interest in the subject attracted investment in research and resulted in significant advances in the development of new methods, libraries, products and services.

Although many of these tools are described as simple and usable without the need for prior knowledge in computer vision, a theoretical basis allows one to choose the appropriate technologies and use them optimally, considering their capabilities and limitations.

This work introduces the areas of facial detection and recognition through an extensive bibliographic review, which provides an overview of the numerous methods found in the literature and presents a collection of useful resources for the training of classifiers and validation of algorithms.

Further study is also made on Viola-Jones and Eigenfaces while presenting the design and implementation of a system capable of detecting and recognizing faces constructed by combining these methods. It is shown that the detection rate of the Eigenfaces is not sufficient for the desired use and alternative solutions are proposed.

The first module of this system runs on a Raspberry Pi and is an example of how to combine theoretical knowledge, open source libraries, comercial tools and hardware in the creation of a profitable product.

Key-words: Facial detection. Viola-Jones. Facial recognition. Eigenfaces. Raspberry Pi.

Lista de ilustrações

Lista de códigos

Lista de Algoritmos

Lista de tabelas

Lista de abreviaturas e siglas

AdaBoost	Adaptive Boosting.
API	Application programming interface (interface de programação de aplicação).
BSD	Berkeley Software Distribution License.
CNN	Convolutional neural network (rede neural convolucional).
CSI	Camera Serial Interface.
DGPLVM	Discriminative Gaussian Process Latent Variable Model.
DNN	Deep neural networks (redes neurais profundas).
EQM	Erro quadrático médio.
FERET	Facial Recognition Technology.
FPR	False positive rate (raxa de falso-positivos).
FPS	Frames per second (quadros por segundo).
KLT	Karnuhen-Loève transform (transformada de Karhunen-Loève).
LDA	Linear Discriminant Analysis (análise discriminante linear).
LBPH	Local Binary Patterns Histograms.
LFW	Labeled Faces in the Wild.
MHz	Mega-hertz.
MP	Megapixel.
OpenCV	Open Source Computer Vision Library.
PCA	Principal Component Analysis (análise de componentes principais).
RAM	Random Access Memory.
SDRAM	Synchronous dynamic random-access memory (memória de acesso aleatório dinâmica síncrona).
SVD	Singular-Value Decomposition (decomposição em valores singulares)

Lista de símbolos

Γ	Letra grega Gama
Λ	Letra grega Lambda
Ψ	Letra grega Psi
θ	Letra grega minúscula teta
ζ	Letra grega minúscula zeta
$\in$	Pertence

Sumário

Introdução

A capacidade de identificar faces e suas emoções é um importante mecanismo neurológico para interações sociais que, em certo grau, está presente até mesmo em recém-nascidos (MORTON; JOHNSON, 1991; FANTZ, 1961).

Enquanto humanos possuem uma notável capacidade de reconhecer faces, o desenvolvimento de sistemas computacionais com capacidade similar é uma área de pesquisa em andamento há mais de cinco décadas (BLEDSOE, 1964; CHAN; BLEDSOE, 1965; BLEDSOE, 1966a; BLEDSOE, 1966b; BOYER; BOYER, 1991; KELLY, 1970; KANADE, 1973).

O desenvolvimento de modelos computacionais para reconhecimento facial são de interesse para diversas aplicações práticas como identificação criminal, sistemas de segurança, biometria, processamento de imagens e interação humano-computador.

Infelizmente, o desenvolvimento de um sistema computacional para reconhecimento facial automatizado é bastante difícil por diversos motivos. Faces são complexas, multidimensionais e expressivas (TURK; PENTLAND, 1991a) e as imagens podem sofrer variações em escala, localização, ponto de visão, iluminação e obstrução (CEN, 2016).

Motivação

O desenvolvimento de novos algoritmos de processamento de imagens acompanhado do maior acesso a câmeras digitais, hardwares com alto poder de processamento, bibliotecas para visão computacional, como a OpenCV, e APIs de análise de imagem, como o Google Vision, o Microsoft Cognitive Services, o Amazon Rekognition e o IBM Watson Visual Recognition, tornou viável a implantação de sistemas com detecção e reconhecimento facial em diversas empresas e produtos.

Nos últimos anos, lojas de varejo têm usado com sucesso as tecnologias de detecção e reconhecimento facial para segurança, personalização de serviço, marketing e análise de sentimento (ROBERTS, 2015; REPORTS, 2015; SALOMãO, 2018; NETO, 2017; NEWS, 2015).

Este trabalho explora um algoritmo de grande impacto na última década, o algoritmo Viola-Jones, e foi motivado pela implementação de um sistema de reconhecimento facial em uma rede de lojas de varejo na cidade do Rio de Janeiro.

Objetivos

O objetivo deste trabalho é fornecer uma base teórica e prática para a futura construção de um sistema de detecção e reconhecimento facial, que será utilizado em uma loja de varejo e, possivelmente, em outros estabelecimentos.

O sistema de detecção será executado em um computador de placa única com câmera, como o Raspberry Pi, que ficará instalado em um ponto estratégico da loja. Ele capturará fotos continuamente e deverá ser capaz de identificar quais fotos contém faces, extrair essas faces e enviá-las para o servidor no qual os algoritmos de reconhecimento facial serão executados.

O presente trabalho deverá identificar um algoritmo de detecção facial capaz de ser executado quase em tempo real nesse hardware com recursos limitados e também encontrar ou desenvolver uma implementação do algoritmo de detecção facial compatível com a arquitetura ARM.

Parte I

Detecção Facial

1 Detecção facial

O papel de um detector de faces é, dada uma imagem arbitrária, determinar se ela contém faces humanas ou não e, caso positivo, retornar a localização e dimensões de cada face (CEN, 2016).

A detecção de faces é utilizada em câmeras fotográficas para ajuste automático de foco, em softwares de imagens e redes sociais para marcação de pessoas e é uma etapa importante para o processo de reconhecimento facial. Antes de executar um algoritmo de reconhecimento facial, é de praxe realizar uma detecção facial a fim de concentrar os esforços do reconhecedor facial apenas nas áreas relevantes.

É preciso minimizar tanto a quantidade de faces não identificadas (falso-negativos) quanto objetos reconhecidos erroneamente como faces (falso-positivos) para obter uma performance aceitável. Para tanto, algoritmos de aprendizado de máquina podem ser muito úteis.

Diversas dificuldades influenciam na eficiência dos algoritmos, como ruídos, variação de iluminação, expressões faciais, imagem de fundo, orientação da cabeça, obstrução da face ou sobreposição de faces (SANTANA; ROCHA, 2015) e, nota-se que, para análise de streamings de vídeo, é fundamental que a detecção facial seja realizada em tempo real.

1.1 Métodos

Segundo Santana e Rocha (2015):

> As técnicas mais citadas para realizar a detecção de faces são: casamento de padrões que consiste na detecção por meio de comparações com formas geométricas, modelos estatísticos, modelos baseado em redes neurais, modelos baseados em tons de pele e o Viola-Jones.

Yang, Kriegman e Ahuja (2002) classificou os algoritmos de detecção e reconhecimento facial em quatro categorias:

1. **Métodos baseados em conhecimento (*knowledge-based*)** – Utilizam regras baseadas no conhecimento sobre características das faces humanas, como posição dos olhos e simetria. Este método possui a desvantagem de ser difícil derivar regras precisas que cubram todos os cenários.
2. **Métodos baseados em características (*feature based*)** – Procuram características invariantes que definam uma face, como regiões com cor de pele, formato dos lábios e bochechas. São bastante impactados por sombras e ruídos.

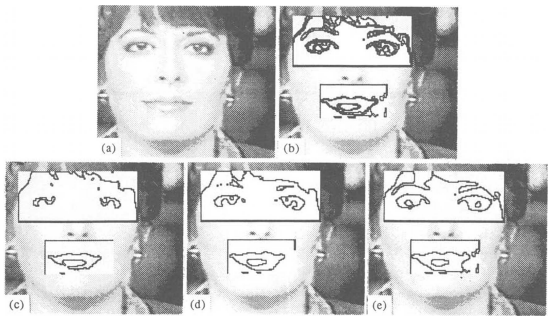
3. **Métodos de casamento de padrões (*template matching*)** – Utilizam templates predefinidos de faces, como uma silhueta ou um mapa de bordas. Este método não lida bem com variações de escala, poses, orientação e formatos de rosto.

4. **Métodos baseados na aparência (*appearance-based*)** – Utilizam algoritmos de aprendizado de máquina e bancos de imagens para treinar classificadores. Este é o método mais bem sucedido.

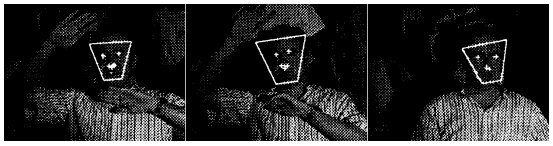
Tabela 1 – Categorias de métodos para detecção facial segundo Yang, Kriegman e Ahuja (2002)

Abordagem	Trabalhos representativos
Baseados em conhecimento	
	Yang e Huang (1994)
Características invariantes	
– Características faciais	Leung, Burl e Perona (1995)
	Yow e Cipolla (1997)
– Textura	Dai e Nakano (1996)
– Cor da pele	Yang e Waibel (1996)
	McKenna, Gong e Raja (1998)
– Múltiplas características	Kjeldsen e Kender (1996)
Casamento de padrões	
– Templates predefinidos	Craw, Tock e Bennett (1992)
– Templates deformáveis	Lanitis, Taylor e Cootes (1995)
Baseados na aparência	
– Eigenface	Turk e Pentland (1991a)
– Baseado em distribuição	Sung e Poggio (1998)
– Redes neurais	Rowley, Baluja e Kanade (1998)
– Máquina de vetores de suporte (SVM)	Osuna, Freund e Girosit (1997)
– Classificador Naive Bayes	Schneiderman e Kanade (1998)
– Modelo oculto de Markov (HMM)	Rajagopalan et al. (1998)
– Teoria da informação	Lew (1996)
	Collobert et al. (1996)

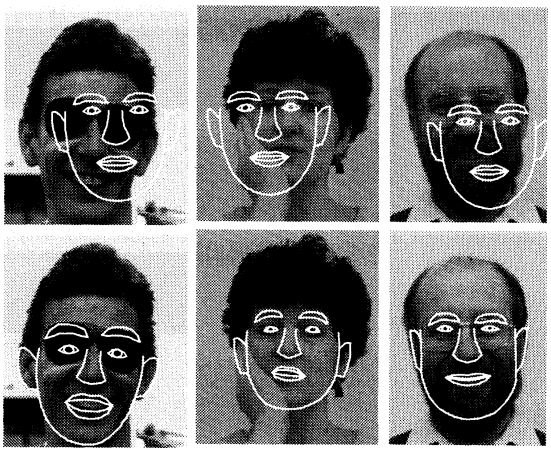
Figura 1 – Exemplos de métodos de detecção facial



(a) Baseado em conhecimento. Fonte: Yang e Huang (1994)



(b) Baseado em características invariantes. Fonte: Leung, Burl e Perona (1995)



(c) Casamento de padrões. Fonte: Lanitis, Taylor e Cootes (1995)

(d) Baseado na aparência. Fonte: Turk e Pentland (1991a)

2 O algoritmo de Viola e Jones

O algoritmo apresentado por Paul Viola e Michael J. Jones em 2001 (VIOLA; JONES, 2001; VIOLA; JONES, 2004) e revisado em 2003 (JONES; VIOLA, 2003) obteve grande sucesso pela sua relativa simplicidade, rápida execução e notável performance (JENSEN, 2008).

Ele encontra-se implementado na biblioteca OpenCV (BRADSKI; PISAREVSKY, 2000; VISION, 2018a) e no OpenBR (KLONTZ et al., 2013). Além de detectar faces, o algoritmo também pode ser treinado para detecção de outras partes do corpo (MUSTAFA; MIN; ZHU, 2014) e objetos diversos.

O processo de treino é lento, porém a detecção é bastante rápida. De acordo com Omaia (2009), "este foi o primeiro método de detecção de face em tempo real em vídeo, conseguindo processar até 15 quadros por segundo".

De acordo com Zhang e Zhang (2010) e Viola e Jones (2001), o algoritmo, na forma como proposto no artigo original, possui baixas taxas de detecção para faces de perfil ou inclinadas. Essa limitação foi tratada pelos autores na revisão de 2003 (JONES; VIOLA, 2003) e também por outros pesquisadores, porém essas variantes fogem do escopo deste trabalho.

O detector de faces de Viola-Jones possui quatro estágios: seleção de características Haar-like retangulares, criação da imagem integral, que permite o cálculo rápido das características, treino de classificadores por um algoritmo de aprendizado de máquina baseado no AdaBoost e, por fim, o uso de classificadores em cascata, que descarta as regiões de fundo para focar em áreas mais prováveis de conter uma face.

2.1 Características Haar-like retangulares

As características de Haar utilizadas por Viola-Jones, ilustradas na Figura 2, foram inspiradas nos trabalhos de Papageorgiou, Oren e Poggio (1998), que descreveram características construídas a partir de um conjunto de ondaletas de Haar (Haar wavelets).

Figura 2 – Características Haar-like

(a) (b) (c) (d)

As características Haar-like consistem em um ou mais retângulos adjacentes e são utilizadas para analisar locais específicos em uma janela de detecção. Cada característica

resulta em um único valor, calculado através da soma das intensidades dos pixels sob os retângulos brancos subtraída da soma dos valores sob os retângulos pretos, como na Equação 2.1:

$$\Delta = \frac{1}{n}\sum_{preto}^{n} I(x) - \frac{1}{n}\sum_{branco}^{n} I(x) \tag{2.1}$$

Quanto mais próximo de 1, mais semelhante é a imagem real da característica aplicada.

Figura 3 – Exemplo de característica Haar-like retangular de 4×4 pixels aplicada a uma imagem real. Pela Equação 2.1, $\Delta = 0,74 - 0,18 = 0,56$.

0	0	1	1
0	0	1	1
0	0	1	1
0	0	1	1

(a) Intensidades ideais

0.2	0.2	0.8	0.6
0.1	0.3	0.6	0.8
0.2	0.1	0.8	0.8
0.2	0.1	0.6	0.9

(b) Valores reais

Utilizar características é mais vantajoso do que trabalhar com cada pixel da imagem separadamente, pois é menos custoso computacionalmente e elas permitem reconhecer padrões determinados. A característica da Figura 2b pode ser utilizada para reconhecer a região dos olhos, que costuma ser mais escura do que a região das bochechas, e a característica 3 pode ser usada para reconhecer a região do nariz, como mostrado na Figura 5.

Existem alternativas mais sofisticadas às características Haar-like, como filtros orientáveis (steerable filters) (FREEMAN; ADELSON et al., 1991; GREENSPAN et al., 1994), porém, segundo Viola e Jones (2001), a eficiência de características retangulares fornece ampla compensação por sua flexibilidade limitada.

Em Lienhart e Maydt (2002) foram propostas novas características contendo rotações de 45°, mostradas na Figura 4. Segundo os autores, o uso dessas características reduziu em 10%, em média, o número de alarmes falsos. Messom e Barczak (2009) estenderam essa ideia para rotações de qualquer ângulo, ao custo dos erros de arredondamento.

Para uma imagem com resolução de 24×24 pixels, pode-se construir dezenas de milhares de características diferentes considerando todas as variações de tamanho e posição das características na Figura 2. Para calcular todas elas de forma eficiente, é usada uma representação intermediária da imagem, chamada imagem integral.

Figura 4 – Características propostas por Lienhart e Maydt

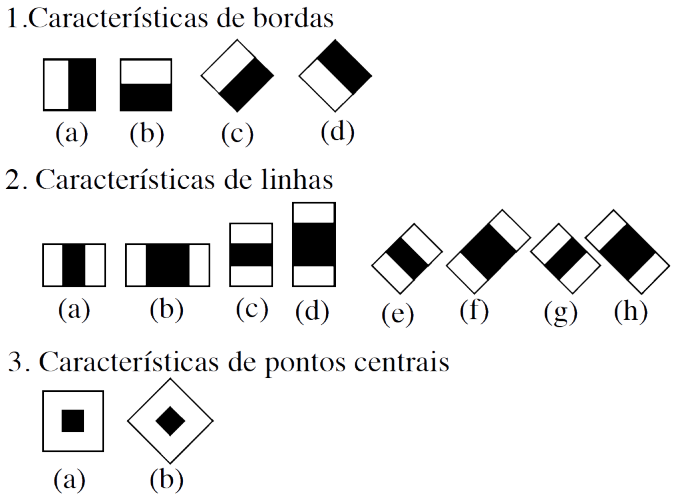


Figura 5 – Características usadas para detectar olhos e nariz

2.2 Imagem Integral

O algoritmo da imagem integral, proposto por Crow (1984) para mapeamento de texturas (mipmaps), é capaz de calcular rapidamente a soma dos valores em um subconjunto retangular de uma matriz.

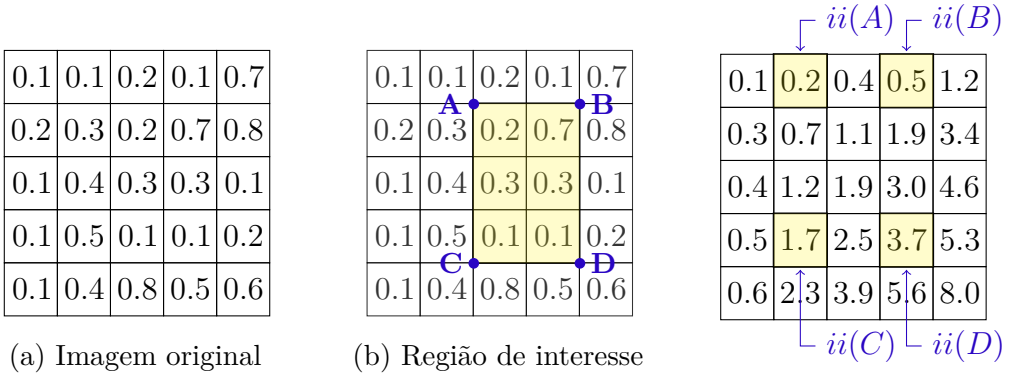
A imagem integral é uma tabela bidimensional do tamanho da imagem original, onde cada elemento equivale à soma de todos os níveis de cinza (intensidades) dos pixels à esquerda e acima do pixel atual, inclusive. Ela pode ser descrita pela Equação 2.2:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{2.2}$$

onde $ii(x, y)$ é a imagem integral e $i(x, y)$ é a imagem original.

Utilizando a imagem integral, a soma dos níveis de cinza de qualquer área retangular pode ser calculada em quatro referências à memória. A soma dos valores na região ABCD

Figura 6 – Imagem ilustrada como matriz de pixels



(a) Imagem original (b) Região de interesse (c) Imagem integral

da Figura 6 pode ser rapidamente calculada como mostrado na Equação 2.3:

$$\sum_{(x,y)\in ABCD} i(x,y) = ii(D) + ii(A) - (ii(B) + ii(C)) \tag{2.3}$$
$$= 3,7 + 0,2 - (0,5 + 1,7) = 1,7$$

O cálculo de características compostas por dois retângulos (Figura 2a e Figura 2b), requerem seis referências à memória, as características compostas por três retângulos (Figura 2c) requerem oito acessos e as compostas por quatro retângulos (Figura 2d) requerem nove acessos.

2.3 AdaBoost

Apesar do cálculo de cada característica ser rápido, calcular todo o conjunto de características é inviável. Experimentalmente, descobriu-se que um classificador eficiente pode ser formado combinando um pequeno subconjunto com apenas as características mais representativas. O algoritmo de Viola-Jones utiliza uma variante do AdaBoost para selecionar essas características e treinar o classificador.

O AdaBoost é um método de aprendizado de máquina, inventado por Yoav Freund e Robert Schapire (FREUND; SCHAPIRE, 1997), que combina de forma ponderada vários classificadores fracos com taxa de acerto acima de 50% para obter um classificador forte.

Um classificador fraco $h_j(x, f_j, p_j, \theta_j)$ consiste em uma característica f_j, um limite θ_j, que decide se x deve ser classificado como positivo (uma face) ou negativo (não face), e uma paridade p_j, que indica a direção da desigualdade:

$$h_j(x, f_j, p_j, \theta_j) = \begin{cases} 1 & \text{se } p_j f_j(x) < p_j \theta_j \\ 0 & \text{caso contrário} \end{cases} \tag{2.4}$$

onde x é uma janela de 24×24 pixels de uma imagem.

O algoritmo AdaBoost pode ser descrito pelo Algoritmo 1. O Anexo A contém uma implementação em Python.

Algoritmo 1: AdaBoost

Entrada: l imagens com faces $(x_i, y_i = 1), 1 \leq i \leq l$;
m imagens sem faces $(x_i, y_i = 0), l < i \leq n$;
T classificadores fracos

Saída: Um classificador forte $H(x)$

`// Inicia com uma distribuição uniforme de pesos`

para $i \leftarrow 1$ **até** n **faça**
  se x_i *tem face* **então**
    $w_{1,i} \leftarrow \frac{1}{2l}$
  senão
    $w_{1,i} \leftarrow \frac{1}{2m}$
  fim
fim

`// Cada iteração seleciona um único classificador fraco`

para $t \leftarrow 1$ **até** T **faça**
  `// Normaliza os pesos`
  $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,i}}$
  `// Seleciona o melhor classificador fraco`
  $h_t(x) \leftarrow$ classificador fraco com menor erro $\epsilon_t = \sum_i w_i |h_t(x_i, f_t, p_t, \theta_t) - y_i|$
  `// Atualiza os pesos` [1]
  se x_i *foi corretamente classificado* **então**
    $w_{t+1,i} \leftarrow w_{t,i}$
  senão
    $w_{t+1,i} \leftarrow w_{t,i} \frac{\epsilon_t}{1-\epsilon_t}$
  fim
fim

retorna

$$H(x) = \begin{cases} 1 & se \ \sum_{t=1}^{T} ln(\frac{1-\epsilon_t}{\epsilon_t}) h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} ln(\frac{1-\epsilon_t}{\epsilon_t}) \\ 0 & caso \ contrário \end{cases} \tag{2.5}$$

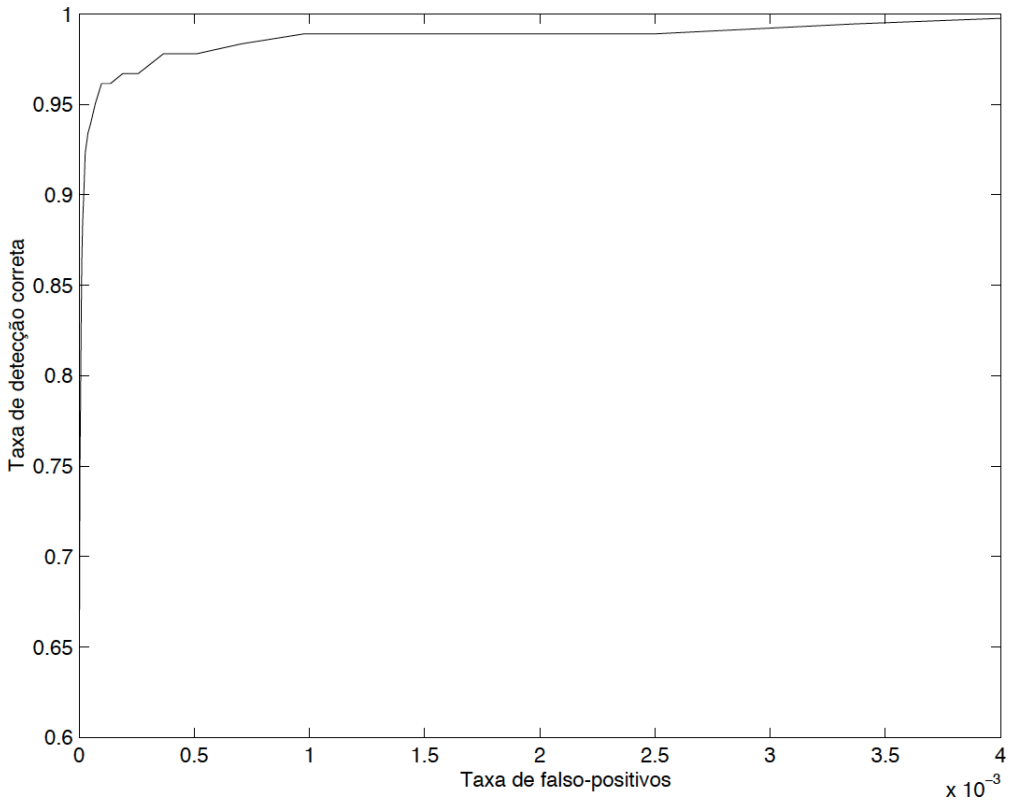
Em Viola e Jones (2004), foi construído um classificador a partir de 200 características. Para uma taxa de detecção de 95%, ele obteve uma taxa de falso-positivos de 1 em 14084 ($1,4 \times 10^{-4}$ FPR) em um conjunto de dados de teste, como mostra a Figura 7.

Apesar de interessante, esse resultado não é suficiente para muitas aplicações reais tanto pelo tempo gasto quanto pela taxa de falso-positivos.

O classificador processou todas as subjanelas de uma imagem com resolução 384×288 pixels em 0,7 segundos em um Intel Pentium III de 700 MHz. Esse valor equivale

[1] A lógica está invertida no artigo original. Esse erro foi replicado em diversos outros artigos. Esta versão está correta, pois faz o peso das imagens classificadas erroneamente aumentar.

Figura 7 – Curva COR para classificador de 200 características

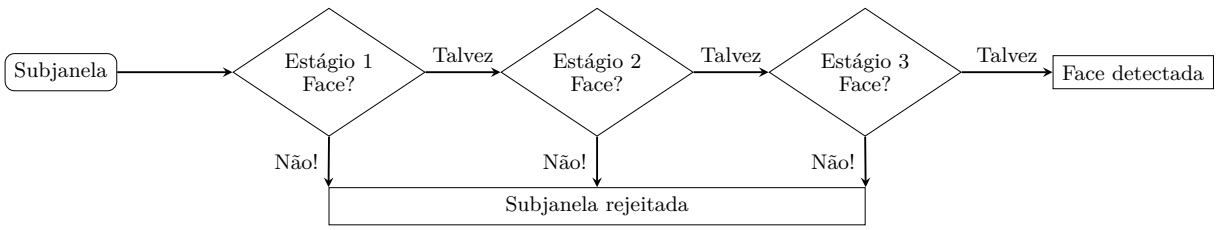


a 1,43 fps, o que não pode ser considerado tempo real.

Câmeras digitais tiram fotos com vários megapixels de resolução. Uma imagem de 1 MP possui 10^6 pixels, portanto a taxa de falso-positivos desejada nesse caso é de menos de 10^{-6}.

2.4 Classificadores em Cascata

Figura 8 – Classificadores em cascata



O classificador em cascata, também chamado de *cascata de atenção* por direcionar a atenção às áreas onde há maior suspeita de haver faces, consiste na concatenação de classificadores gradualmente mais complexos em uma estrutura de cascata.

Cada estágio da cascata contém um classificador forte construído pelo AdaBoost. A função de cada um deles é determinar se uma dada subjanela definitivamente não contém uma face ou talvez contenha uma face. Uma janela é imediatamente descartada se falhar em qualquer estágio. Uma representação do funcionamento da cascata pode ser vista na Figura 8.

Mesmo em fotos com várias pessoas, a maioria das subjanelas avaliadas não contém faces. A ideia chave é que colocar um classificador simples com alta taxa de detecção no início da cascata permite rejeitar as janelas negativas cedo e poupar muito processamento.

Viola e Jones conseguiram treinar um classificador forte, composto por apenas dois classificadores fracos, capaz de detectar 100% das faces com uma taxa de falso positivo de 50%. Desta forma, muitas subjanelas de segundo plano puderam ser descartadas com apenas 60 instruções do processador.

A Figura 9 apresenta uma comparação entre um classificador de duzentas características e um classificador em cascata com dez estágios de vinte características cada. O classificador em cascata obteve uma acurácia semelhante com um tempo de execução quase dez vezes menor.

Figura 9 – Curvas COR comparando classificador de 200 características com classificador em cascata com 10 estágios de 20 características cada

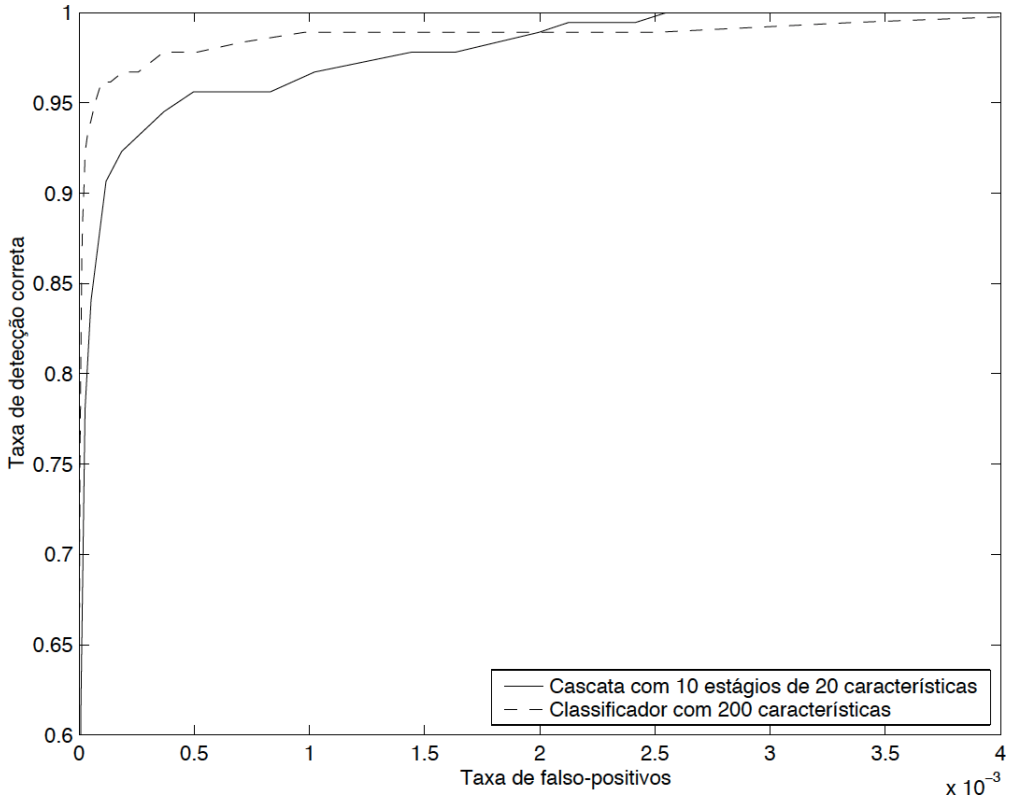
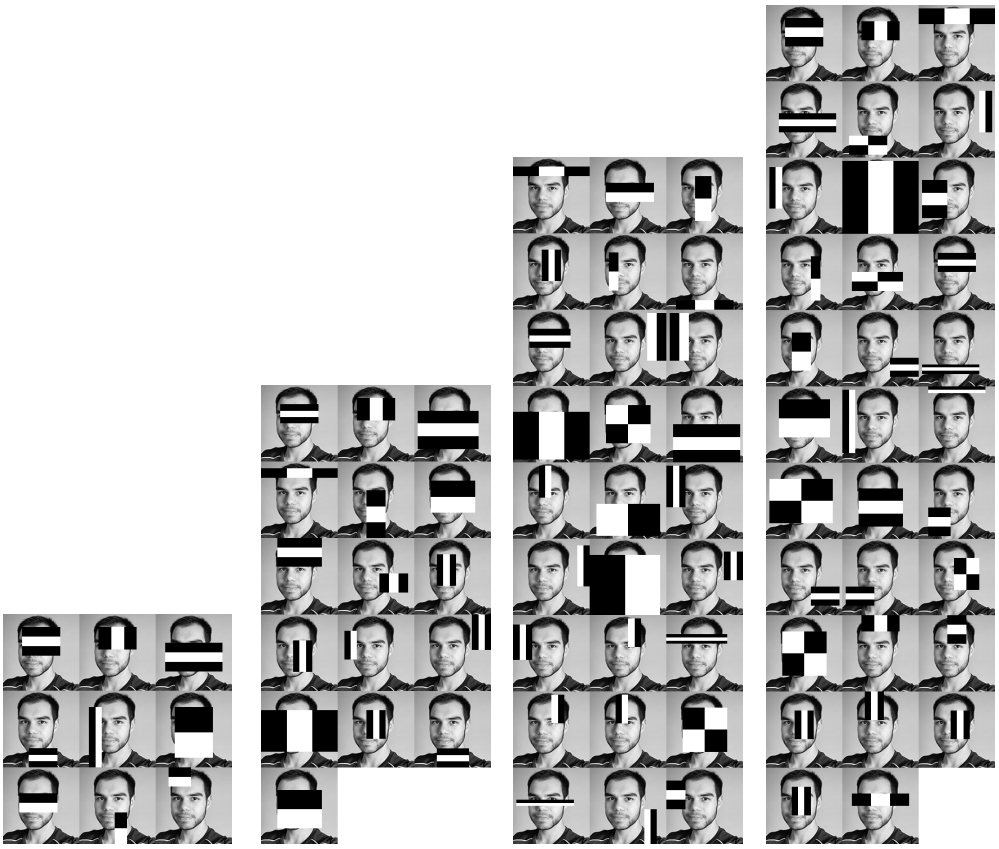


Figura 10 – Quatro estágios de um classificador em cascata



(a) Estágio 1 (b) Estágio 2 (c) Estágio 3 (d) Estágio 4

2.4.1 Treino da cascata de classificadores

A taxa de falso-positivos de uma cascata de classificadores é:

$$F = \prod_{i=1}^{K} f_i \tag{2.6}$$

onde F é a taxa de falso-positivos da cascata, K é o número de classificadores e f_i é a taxa de falso-positivos do i-ésimo classificador.

A taxa de detecção é:

$$D = \prod_{i=1}^{K} d_i \tag{2.7}$$

onde D é a taxa de detecção do classificador, K é o número de classificadores e d_i é a taxa de detecção do i-ésimo classificador.

Como explicado na seção 2.3, a taxa de falso-positivos desejada é da ordem de 10^{-6} para uma taxa de detecção acima de 90%. O número de estágios da cascata e o tamanho de cada estágio pode ser ajustado para alcançar este objetivo. Por exemplo, uma cascata de 10 estágios, cada um com 99% de detecção e um pouco menos de 30% de falso-positivos, é suficiente.

Algoritmo 2: Algoritmo de treino do detector em cascata

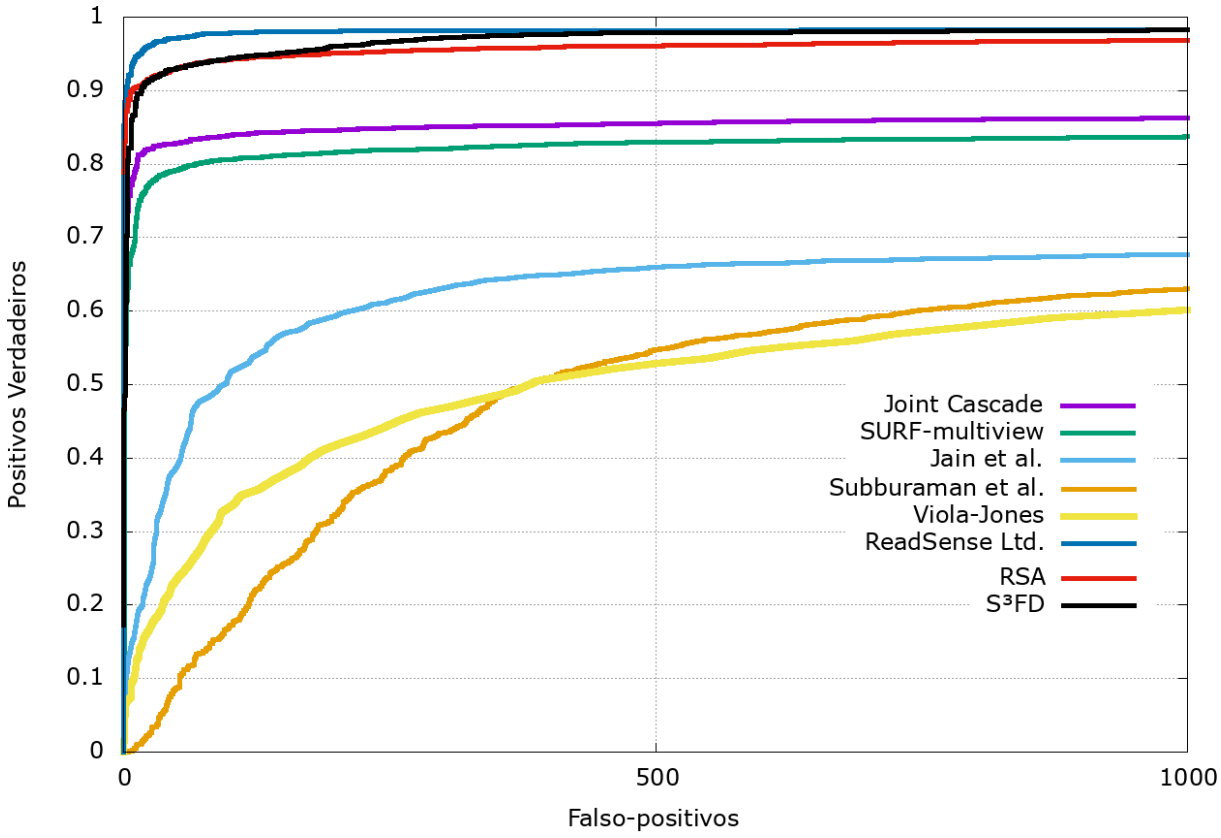
Entrada: P: imagens com faces
N: imagens sem faces
F_{meta}: taxa desejada de falso-positivos
f: taxa máxima de falso-positivos por estágio
d: taxa mínima de detecção por estágio

Saída: Detector de faces treinado

```
1  F_0 ← 1.0
2  D_0 ← 1.0
3  i ← 0 enquanto F_i > F_meta faça
4      i ← i + 1
5      n_i ← 0 // quantidade de características do i-ésimo estágio
6      F_i ← F_{i-1}
7      enquanto F_i > f × F_i1 faça
8          n_i ← n_{i-1}
9          * Usa P e N para treinar um classificador com n_i características usando
             AdaBoost.
10         * Avalia o classificador atual no conjunto de validação para determinar F_i e
             D_i.
11         * Diminui o limite para o i-ésimo classificador até o classificador em cascata
             atual obter uma taxa de detecção de pelo menos d × D_{i-1}
12     fim
13     N ← ∅
14     se F_i > F_meta então
15         Avalia o detector atual no conjunto de imagens sem faces e insere todas as
             detecções falsas ao conjunto N.
16     fim
17 fim
```

Classificadores com mais características obtém maiores taxas de detecção e menos falso-positivos, porém requerem maior tempo de cálculo.

Figura 11 – Curvas COR comparando vários algoritmos de detecção facial no FDDB benchmark (JAIN; LEARNED-MILLER, 2010)



3 Detecção facial com OpenCV

OpenCV (Open Source Computer Vision Library) é uma biblioteca multiplataforma de código aberto (licença BSD) voltada principalmente para aplicações de visão computacional em tempo real (KAEHLER; BRADSKI, 2016). Ela foi lançada oficialmente em 1999 pela Intel Corporation e continua em constante desenvolvimento. Apesar de ser escrita em C e C++, existem interfaces em diversas outras linguagens, como Python, Java e MATLAB.

3.1 Bases de faces para treino e testes

Existe uma grande quantidade de bases de imagens de faces disponíveis para treinar e testar algoritmos de detecção e reconhecimento facial, cada uma com características próprias (GRGIC; DELAC, 2013; HUANG et al., 2007; MOZAFFARI; BEHRAVAN, 2011). A Tabela 2 lista alguns deles.

Tabela 2 – Bases de imagens de faces

Bases de imagens	**Indivíduos**	**Imagens**	**Comentários**
AR Face Database (MARTINEZ, 1998)	126	4000	Pose frontal, expressão, iluminação, oclusão (óculos, cachecol)
AT&T Database (SAMARIA; HARTER, 1994)	40	400	Variação temporal, iluminação, expressão, óculos
BANCA database (BAILLY-BAILLIéRE et al., 2003)	208	2496	Voz e face, voltado para sistemas multimodais de verificação
BioID Face Database (JESORSKY; KIRCHBERG; FRISCHHOLZ, 2001)	23	1521	Tons de cinza, plano de fundo, iluminação, expressão, posição dos olhos
Caltech Faces (WEBER, 1995)	27	450	Iluminação, expressão, plano de fundo
Caltech 10000 Web Faces (ANGELOVA; ABU-MOSTAFAM; PERONA, 2005)	≈ 10000	10000	Grande variedade, características faciais anotadas
CAS-PEAL Face Database (GAO et al., 2008)	1040	99594	Expressão, acessórios, iluminação, poses múltiplas, chinês
CelebA (LIU et al., 2015)	10177	202599	Grande variedade, características faciais anotadas
Cohn-Kanade AU-Coded Facial Expression Database (COHN et al., 1999)	100	500 sequências	Sequência dinâmica de expressões faciais
Disguise Face Database (SINGH; VATSA; NOORE, 2009)	?	?	10 variações por indivíduo usando disfarces sintéticos (barba, etc)

Bases de imagens	**Indivíduos**	**Imagens**	**Comentários**
EQUINOX HID Face Database (SOCOLINSKY et al., 2001)	91	?	Imagens infra vermelho, frontal
Face Video Database da Sociedade Max Planck (KLEINER; WALLRAVEN; BÜLTHOFF, 2004)	?	246 sequências de vídeo	6 pontos de vista simultâneos, vídeo
Face Recognition Grand Challenge Databases (PHILLIPS et al., 2005)	> 466	> 50000 imagens e varreduras 3D	Bem grande, iluminação, expressão, plano de fundo, 3D, sequencias
FEI Face Database (JUNIOR; THOMAZ, 2006)	200	2800	Cor, fundo branco, expressão, rotação de cabeça
FERET Database (Color) (PHILLIPS et al., 1998)	1199	14126	Variação temporal, cor, expressão, pose e iluminação controladas
Georgia Tech Face Database (TECHNOLOGY, 1999)	50	750	Expressão, iluminação, escala, orientação
Indian Face Database (VIDIT; AMITABHA, 2002)	40	> 440	Frontal, Índia
Japanese Female Facial Expression (LYONS et al., 1998)	10	213	Emoções, indivíduos femininos, Japão
Labeled Faces in the Wild (HUANG et al., 2007)	5749	13233	Pose, iluminação, expressão, plano de fundo, diversidade
MIT-CBCL Face Recognition Database (WEYRAUCH et al., 2004)	10	> 2000	Imagens de modelos 3D, iluminação, pose, fundo
M2VTS Multimodel Face Database (Release 1.00) (SANCHEZ; MATAS; KITTLER, 1997)	37	185	Grandes mudanças de pose, indivíduos falando, óculos, mudanças no tempo
M2VTS, Extended, Univ. of Surrey, UK (MESSER et al., 1999)	295	1180 vídeos	Rotação de cabeça, indivíduos falando, modelos 3D, alta definição
NIST Mugshot ID (WATSON, 1994)	1573	3248	Imagens frontais e de perfil
PIE Database, CMU (SIM; BAKER; BSAT, 2002)	68	41368	Bem grande, pose, iluminação, expressão
Plastic Surgery Database (SINGH et al., 2010)	900	1800	Imagens antes e depois de diferentes tipos de cirurgia plástica
Psychological Image Collection at Stirling (PICS) (HANCOCK, 2004)	?	?	Voltado para experimentos de psicologia
Synthetic Face Disguise Database (SINGH; VATSA; NOORE, 2009)	100	4000	Faces sintéticas com disfarces variados
UCD Colour Face Image Database for Face Detection (SHARMA; REILLY, 2003)	≈ 299	299	Voltado para detecção, bastante variado, cor
UMIST Face Database (GRAHAM; ALLINSON, 1998)	20	564	Pose, gênero, raça, tons de cinza

Bases de imagens	**Indivíduos**	**Imagens**	**Comentários**
University of Essex, UK (SPACEK, 1996)	395	7900	Diversidade racial, óculos, barbas, universitários
University of Oulu Physics-Based Face Database (MARSZALEC et al., 2000)	125	> 2000	Iluminação bastante variada, óculos
VALID Database (FOX; O'MULLANE; REILLY, 2005)	106	530	Condições de escritório altamente variáveis
VidTIMIT Database (SANDERSON, 2008)	43	Múltiplos vídeos por pessoa	Vídeo, áudio, rotação de cabeça
Yale Face Database (BELHUMEUR; HESPANHA; KRIEGMAN, 1997)	15	165	Tons de cinza, expressões, óculos, iluminação
Yale Face Database B (GEORGHIADES; BELHUMEUR; KRIEGMAN, 2001)	10	5760	Pose, iluminação
Yale Face Database B Extended (LEE; HO; KRIEGMAN, 2005)	38	2452[1]	Nove poses e 64 condições de iluminação

Este trabalho utilizou quatro dessas bases para treinar e avaliar o detector facial:

FERET É uma base de faces criada por P. Jonathon Phillips e Harry Wechsler entre 1993 e 1996 como parte do programa FERET (Facial Recognition Technology), patrocinado pelo Departamento de Defesa dos Estados Unidos. A cópia utilizada neste trabalho foi obtida através de solicitação por email ao NIST (*National Institute of Standards and Technology*). Esta base contém 14126 imagens tiradas em condições controladas com algumas variações de ângulo e expressão. Alguns dos indivíduos foram fotografados múltiplas vezes em diferentes anos, o que permite estudar mudanças de aparência. Para o treino do classificador, foram selecionadas apenas 2409 fotos frontais.

FEI É uma base de faces criada pelo laboratório de processamento de imagens do Centro Universitário FEI em São Bernardo do Campo. Ela é composta por 14 imagens de 200 indivíduos (100 homens e 100 mulheres), totalizando 2800 fotografias. Todas as imagens são coloridas e com fundo branco. Para o treino do classificador, foram utilizadas 400 fotos frontais.

Caltech 10000 Web Faces É uma base que contém 10524 faces em 7092 imagens obtidas da internet. As coordenadas dos olhos, nariz e centro da boca estão anotadas em um arquivo, o que permitiu cortar as imagens para serem usadas no treinamento do classificador. Algumas imagens foram excluídas por não mostrar suficientemente o rosto, estarem muito rotacionadas ou não possuírem resolução suficiente.

[1] O site diz 16128 imagens de 28 indivíduos, porém o arquivo disponível para download contém mais imagens.

LFW (Labeled Faces in the Wild). É uma base de 13233 imagens coletadas da internet criado e distribuído pela universidade de Massachussetts para estudar reconhecimento facial irrestrito. Essa base foi utilizada apenas para testar o classificador.

3.2 Treino do Classificador

3.2.1 Imagens de treinamento

Para treinar um classificador, é preciso um conjunto de imagens positivas (imagens contendo faces) e um conjunto de imagens negativas (imagens de fundo).

Viola e Jones (2004) usaram 4916 imagens positivas e 9500 imagens negativas, posteriormente, Lienhart, Kuranov e Pisarevsky (2003) concluíram que, nas condições usadas por eles, utilizar mais do que 5000 imagens positivas e 3000 imagens negativas traz pouco benefício para o resultado do treinamento.

Imagens positivas podem ser obtidas de alguma base listada na seção 3.1. Se a base escolhida não possuir uma quantidade suficiente de exemplos, é possível aplicar transformações como rotação, inversão e alteração de intensidade às imagens existentes.

Após obter imagens positivas, é preciso identificar onde as faces estão localizadas utilizando um arquivo com formato específico, no qual cada linha contém o caminho da imagem, a quantidade de faces marcadas e coordenadas que descrevem retângulos em volta das faces.

O OpenCV possui uma ferramenta chamada `opencv_annotation` que permite marcar com o cursor do mouse as regiões contendo faces. O comando para executá-la é `opencv_annotation --annotations=/diretorio/anotacao.txt --images=/diretorio/das/imagens/`, como mostra a Figura 12.

Se as imagens contiverem apenas faces já cortadas, é possível gerar o arquivo de anotações com o comando `find positivas/ -name "*.jpg" -exec identify -format '%i 1 0 0 %w %h\n' {} \; > anotacoes.txt`.

Também é preciso listar as imagens negativas, o que pode ser feito com o comando `find negativas/ -name "*.jpg" > negativas.txt`.

3.2.2 Treinamento

A OpenCV também possui ferramentas para o treinamento do classificador. Primeiro é necessário criar um arquivo de vetor com o comando `opencv_createsamples`, passando a quantidade de exemplos, a largura e a altura como parâmetros. Exemplo: `opencv_createsamples -info anotacoes.txt -num 2400 -w 24 -h 24 -vec positivas_24x24.vec`.

Figura 12 – Ferramenta de anotação do OpenCV

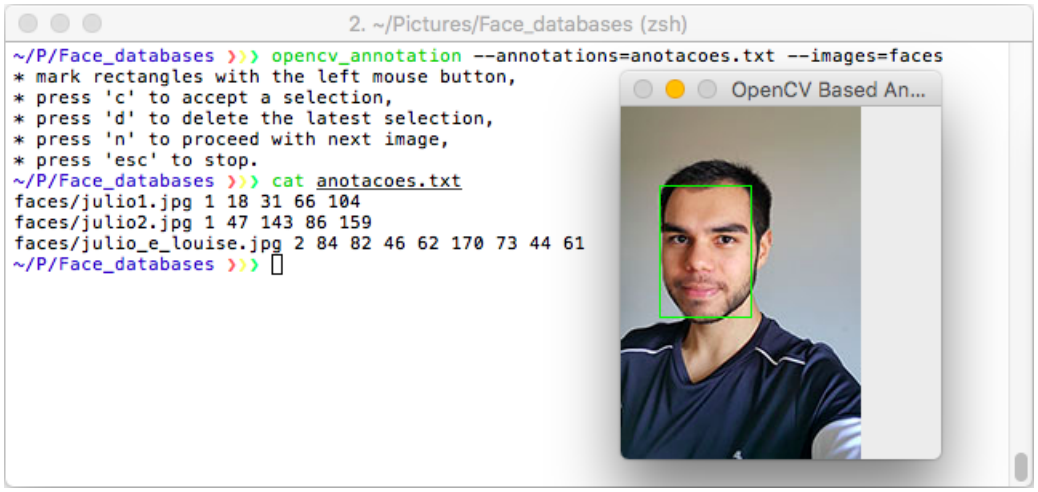


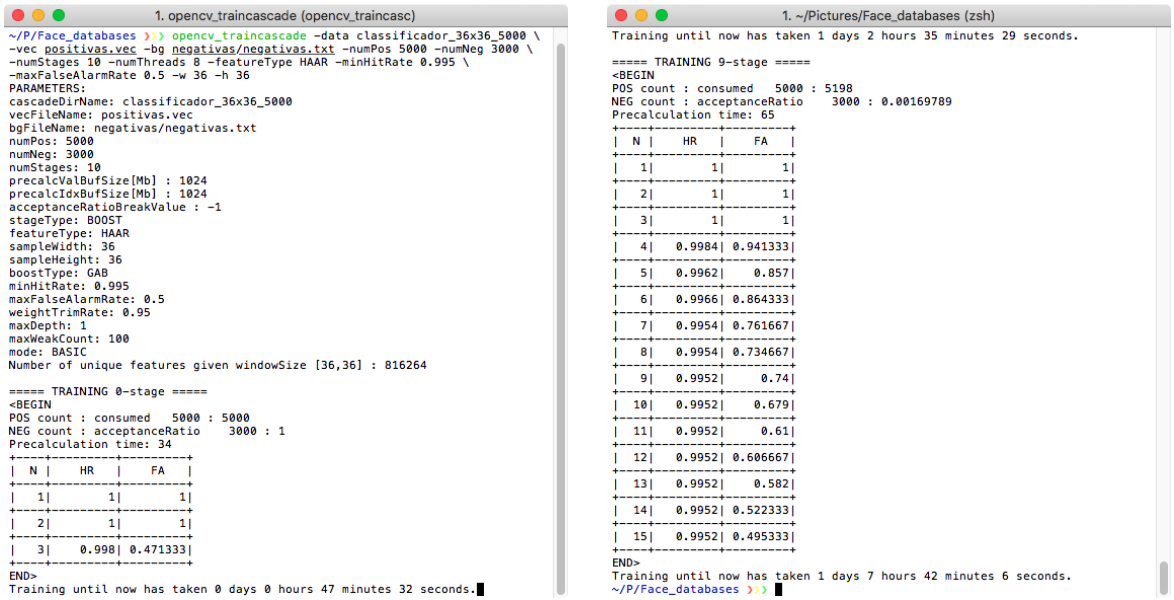
Figura 13 – Criação do vetor

Para visualizar as imagens inseridas no vetor, basta usar o mesmo comando, mas passando o vetor como parâmetro: `opencv_createsamples -vec positivas_24x24.vec -w 24 -h 24`.

Finalmente, o classificador em cascata é treinado com o comando `opencv_traincascade`, que recebe como parâmetros um diretório onde o classificador será salvo, o arquivo de vetor gerado anteriormente, a lista de imagens negativas, a quantidade de imagens positivas e negativas, o número e estágios da cascata e as dimensões das imagens no vetor. Exemplo: `opencv_traincascade -data classificador/ -vec positivas_24x24.vec -bg negativas.txt -numPos 2000 -numNeg 1000 -numStages 10 -w 24 -h 24`.

O treinamento pode demorar vários dias dependendo das especificações do computador e dos parâmetros passados para o comando `opencv_traincascade`. Como visto na

Figura 14 – Treinamento do classificador em cascata usando `opencv_traincascade`



(a) Início (b) Fim

Figura 14, o treinamento de um classificador em cascata de 10 estágios utilizando 5000 imagens positivas e 3000 imagens negativas com 36×36 pixels cada demorou 1 dia 7 horas e 42 minutos para ser concluído em um MacBook Air de 2013 com processador Intel Core i5 de 1,3 GHz e 4 GB de RAM.

3.3 Uso do classificador

Como resultado do treinamento, um arquivo chamado `cascade.xml` é criado no diretório especificado. Esse arquivo é o classificador em cascata treinado. Ele pode ser carregado com o método `cv2.CascadeClassifier` do OpenCV, como usado no Código 1, cuja execução pode ser vista na Figura 15 e na Figura 16.

Pela Figura 16, é possível perceber que o classificador em cascata foi capaz de detectar faces com alguma oclusão, levemente rotacionadas e com expressões, porém o número de falso-positivos foi consideravelmente alto. Este resultado sugere que o classificador deve ser treinado com mais imagens negativas, mais estágios e com um valor menor para o parâmetro maxFalseAlarmRate.

Figura 15 – Captura de tela do Código 1 em execução.

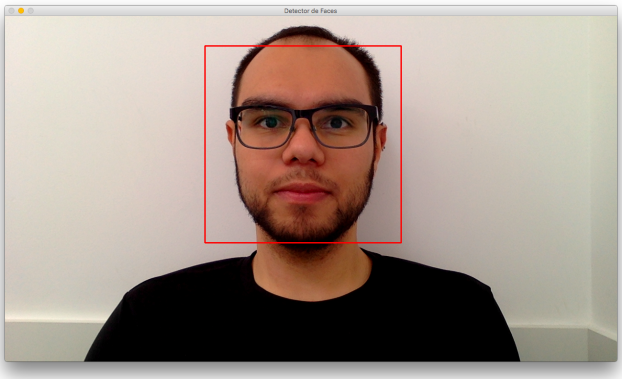
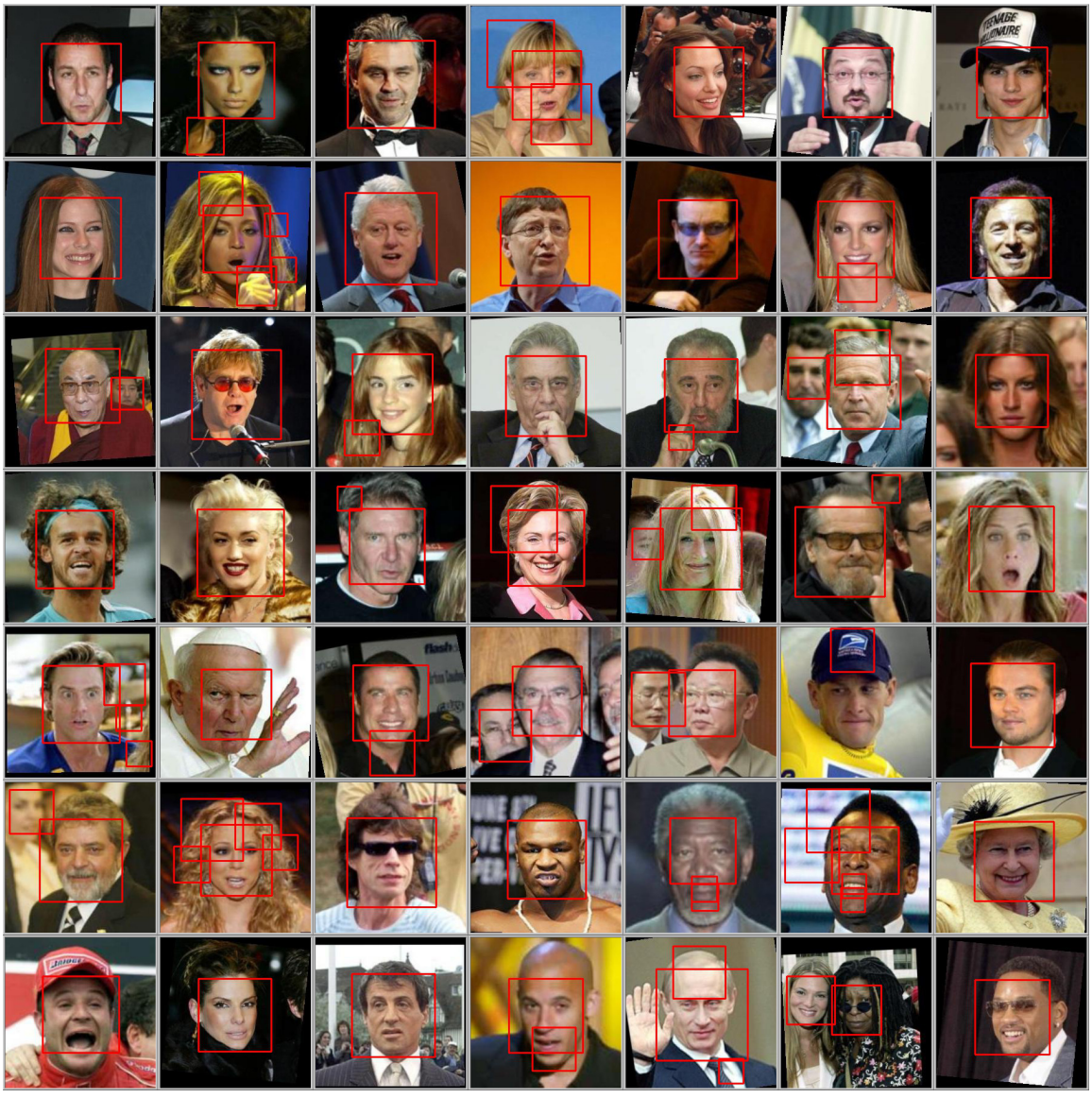


Figura 16 – Teste do classificador em cascata com imagens da Labeled Faces in the Wild



Parte II

Reconhecimento Facial

4 Reconhecimento facial

Um sistema de reconhecimento facial é uma tecnologia capaz de realizar identificação ou verificação automática de uma pessoa através de uma imagem ou vídeo.

Segundo Jafri e Arabnia (2009), essas duas tarefas principais do reconhecimento facial podem ser definidas como:

- **Verificação** (correspondência um-para-um): dada uma face e uma alegação de identidade, informa se a face realmente pertence ao indivíduo.
- **Identificação** (correspondência de um-para-muitos): compara a imagem de uma face desconhecida com todas as imagens em um banco de dados para determinar sua identidade.

Uma terceira tarefa também importante é a de agrupamento (clustering), que consiste em agrupar faces da mesma pessoa mesmo que essa pessoa não esteja cadastrada em um banco. (DHINGRA, 2017; SCHROFF; KALENICHENKO; PHILBIN, 2015).

São várias as aplicações práticas desses sistemas, como autenticação biométrica, vigilância, interação humano-computador e gerenciamento multimídia. (JAIN; LI, 2011).

O reconhecimento facial possui diversas vantagens sobre outras formas de biometria. Para validação de passaportes, por exemplo, Heitmeyer (2000) afirma que esse é o método preferido por ser rápido, não intrusivo e poder ser feito à distância sem necessidade de contato.

A tarefa de verificação facial em cenários cooperativos, nos quais as condições de iluminação e pose são controladas, já pode ser considerada um problema bem resolvido. Por outro lado, a identificação um-para-muitos, especialmente em cenários não cooperativos, como busca por pessoas desaparecidas ou procuradas pela polícia, ainda é uma tarefa desafiadora. A performance do reconhecimento é grandemente influenciada por variações nas condições de iluminação, ponto de vista, poses, expressão facial, envelhecimento, uso de disfarces e acessórios (LI; CHU; PAN, 2016; JAIN; LI, 2011).

Dentre os trabalhos pioneiros em reconhecimento facial automatizado, destacam-se a tese de doutorado de Takeo Kanade em 1973, os trabalhos sobre representação de faces em dimensionalidade reduzida usando PCA feitos por Kirby e Sirovich em 1987 e 1990 e os trabalhos sobre Eigenfaces de Turk e Pentland em 1991a.

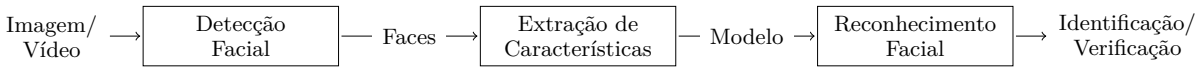
Outras técnicas relevantes utilizam discriminantes lineares (Fisherfaces LDA) (BELHUMEUR; HESPANHA; KRIEGMAN, 1997; ETEMAD; CHELLAPPA, 1996), local binary patterns (LBP) (OJALA; PIETIKAINEN; HARWOOD, 1994), filtros de Gabor

(LADES et al., 1993; WISKOTT et al., 1997; JúNIOR et al., 2016) e, mais recentemente, GaussianFace (DGP-LVM) (LU; TANG, 2015) e redes neurais profundas (DNN) (TAIGMAN et al., 2014; HE et al., 2015; SUN et al., 2014; SUN; WANG; TANG, 2014; SUN; WANG; TANG, 2015; ZHU et al., 2014; SCHROFF; KALENICHENKO; PHILBIN, 2015; ZHOU; CAO; YIN, 2015; YI et al., 2014; SUN et al., 2015; AMOS; LUDWICZUK; SATYANARAYANAN, 2016).

Redes neurais profundas podem ser consideradas o estado da arte em reconhecimento facial. Com taxas de detecção acima de 99,8% na LFW, esses métodos já ultrapassam a performance humana nesse teste (LEARNED-MILLER et al., 2016; KUMAR et al., 2009).

O processo de reconhecimento automático de faces envolve três etapas: detecção de faces, extração de características e identificação ou verificação, como ilustrado na Figura 17. Na fase de extração de características, é comum realizar uma normalização para alinhar as faces.

Figura 17 – Etapas do reconhecimento facial



Os algoritmos Eigenfaces, Fisherfaces e Local Binary Patterns Histograms (LBPH) estão implementados na biblioteca OpenCV (VISION, 2018b).

4.1 Extração de características

Imagens digitais são representadas por milhares de pixels que são codificados em arrays de alta dimensionalidade. Os métodos de extração de características procuram encontrar as informações mais relevantes das imagens originais e representá-las em um espaço de menor dimensionalidade.

Existem 256^{10000} formas diferentes de formar uma imagem em tons de cinza (8 bits) de 100×100 pixels, mas apenas uma pequena fração dessas imagens representam faces. Além disso, muitas características são comuns a todas as faces e não são úteis para diferenciá-las.

A redução da dimensionalidade do espaço permite eliminar essas redundâncias e características indesejadas. Ela pode ser realizada se o erro quadrático médio (EQM) ou a soma da variância dos elementos forem mínimas, o que é verdade para imagens de faces por elas compartilharem muitas similaridades.

Como explicado por Datta, Datta e Banerjee (2015), a ideia principal dos métodos de reconhecimento facial baseados em subespaço é encontrar vetores que melhor representam a distribuição de faces em todo o espaço da imagem. Esses vetores de dimensão reduzida definem o subespaço das faces. Após a linearização, o vetor médio entre todas as imagens

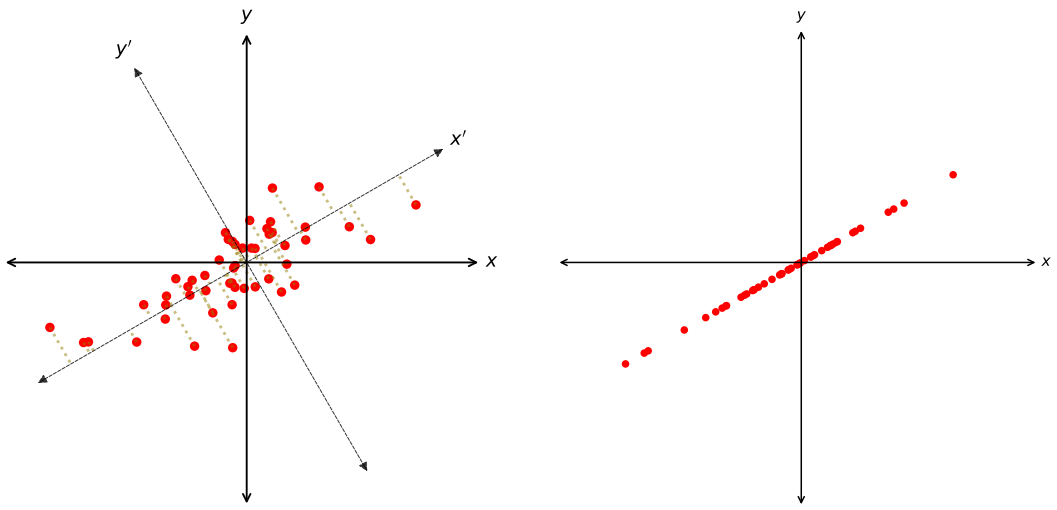
de faces é calculado e subtraído de cada vetor correspondente às faces originais. Depois, a matriz de covariância é calculada para extrair um número limitado de autovetores, correspondentes aos maiores autovalores. Esses autovetores, também chamados eigenfaces, representam uma base em um espaço de baixa dimensionalidade. Para testar uma imagem, sua eigenface é calculada e comparada com todas as faces no banco de dados.

Os principais métodos que utilizam subespaços para reconhecimento facial são o PCA e o LDA (WANG; TANG, 2004). O PCA seleciona as características que melhor representam a face e o LDA seleciona o subespaço que melhor discrimina classes de faces. Eles também podem ser utilizados em conjunto.

4.1.1 Análise de componentes principais (PCA)

A análise de componentes principais (em inglês, principal component analysis ou PCA) é uma técnica de redução de dimensionalidade inventada em 1901 por F.R.S. (1901). Em (GERBRANDS, 1981) foi concluído que PCA é o mesmo que a transformada de Karhunen-Loève (KLT) exceto por uma possível mudança na origem do sistema de coordenadas. Em (ANDREWS, 1975) foi alertado que, apesar de próximos, SVD (decomposição em valores singulares) não é o mesmo que PCA e KLT.

Figura 18 – Análise de Componentes Principais



(a) Distribuição original (b) Apenas o primeiro componente principal

Segundo (JOLLIFFE, 2002), a ideia central da análise de componentes principais (PCA) é reduzir a dimensionalidade de um conjunto de dados que consiste em um grande número de variáveis, mantendo o máximo possível da variação presente no conjunto de dados. Isto é alcançado através da transformação para um novo conjunto de variáveis, os componentes principais, que não são correlacionados e são ordenados de forma que os primeiros retenham a maior parte da variação presente em todas as variáveis originais.

O conceito do PCA está ilustrado na Figura 18. Os eixos x e y formam a base original e o eixo x' corresponde à direção de maior variância, ou seja, ao primeiro componente principal.

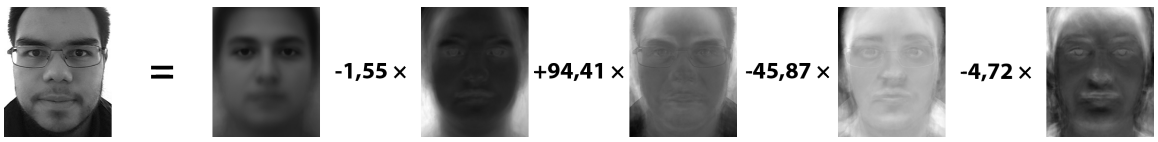
Matematicamente, dados n pontos em $\mathbb{R}^p$, a PCA consiste em escolher uma dimensão $k < p$ e encontrar um espaço afim de dimensão k onde a distância ao quadrado dos pontos às suas projeções ortogonais no espaço são minimizadas.

4.1.2 Eigenfaces

As eigenfaces são os componentes principais de uma distribuição de faces, ou seja, os autovetores da matriz de covariância de um conjunto de imagens de faces. Cada face pode ser representada como uma combinação linear de eigenfaces, como mostra a Figura 19. Elas também podem ser aproximadas usando apenas as eigenfaces que possuem os maiores autovalores e que, consequentemente, são responsáveis pela maior variância.

A ideia de usar componentes principais para representar faces humanas foi desenvolvida por Kirby e Sirovich (1987) e usada por (TURK; PENTLAND, 1991a) para detecção e reconhecimento facial. Muitos consideram essa técnica de reconhecimento facial baseada em aparência como a primeira a ser utilizada com sucesso na prática.

Figura 19 – Representação de uma face como combinação linear de eigenfaces



4.1.2.1 Algoritmo para cálculo das eigenfaces

O algoritmo para a obtenção das eigenfaces proposto por Turk e Pentland (1991b) pode ser formulado da seguinte forma:

1. Obter M imagens $I_1, I_2, I_3 \dots I_M$ com dimensão $N \times N$. As imagens devem estar centralizadas.

2. Representar cada imagem I_i como um vetor Γ_i de dimensão $N^2 \times 1$.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N1} & \cdots & a_{NN} \end{bmatrix}_{N \times N} \rightarrow \begin{bmatrix} a_{11} \\ \vdots \\ a_{1N} \\ \vdots \\ a_{2N} \\ \vdots \\ a_{NN} \end{bmatrix}_{N^2 \times 1} \tag{4.1}$$

3. Calcular o vetor Ψ correspondente à face média.

$$\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i \tag{4.2}$$

4. Subtrair a face média de cada vetor Γ_i para obter o conjunto de vetores Φ_i.

$$\Phi_i = \Gamma_i - \Psi \tag{4.3}$$

5. Encontrar a matriz C de covariância.

$$C = AA^T, \text{ onde } A = \begin{bmatrix}\Phi_1 & \Phi_2 & \dots & \Phi_3\end{bmatrix} \tag{4.4}$$

   C é uma matriz $N^2 \times N^2$ e A é uma matriz $N^2 \times M$.

6. Calcular os autovetores u_i de $C = AA^T$.

   Em vez desse cálculo, que resultaria em N^2 autovetores, calcular os autovetores v_i da matriz A^TA de dimensão $M \times M$.

$$\begin{aligned} & A^TAv_i = \mu_i v_i \\ \Rightarrow \quad & AA^TAv_i = \mu_i Av_i \\ \Rightarrow \quad & CAv_i = \mu_i Av_i \\ \Rightarrow \quad & u_i = Av_i \end{aligned} \tag{4.5}$$

   ou seja, Av_i são autovetores de $C = AA^T$. Os M autovalores de A^TA correspondem aos M maiores autovalores de AA^T.

7. Manter apenas os K autovetores, correspondentes aos K maiores autovalores.

Cada face Φ_i (face menos a média), correspondente a uma das M imagens de treinamento, possui uma representação vetorial Ω_i na base formada pelos K autovetores escolhidos, ou seja, pode ser representada pela seguinte combinação linear:

$$\Gamma_i - \Psi = \Phi_i = \sum_{j=1}^{K} w_j u_j, (w_j = u_j^T\Phi_i) \tag{4.6}$$

Cada peso w_j descreve a contribuição do autovetor u_j na representação da imagem. Eles formam o seguinte vetor Ω_i:

$$\Omega_i = \begin{bmatrix} w_1^i \\ w_2^i \\ \vdots \\ w_K^i \end{bmatrix}, i = 1, 2, \dots, M \tag{4.7}$$

4.1.2.2 Reconhecimento usando eigenfaces

O reconhecimento de uma face desconhecida Γ centralizada e com as mesmas dimensões das imagens de treinamento é feito através dos seguintes passos:

1. Normalização da imagem.
$$\Gamma : \Phi = \Gamma - \Psi \tag{4.8}$$

2. Projeção no autoespaço.
$$\hat{\Phi} = \sum_{i=1}^{K} w_i u_i, (w_i = u_i^T \Phi) \tag{4.9}$$

3. Representação de Φ como Ω:
$$\Omega = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \end{bmatrix} \tag{4.10}$$

4. Cálculo da distância (euclidiana) mínima $e_r = min_l \left\| \Omega - \Omega^l \right\|$.

5. Se a distância e_r foi menor que um limite T_r escolhido, então Γ é reconhecida como a face l do conjunto de treino.

Figura 20 – Eigenfaces

(a) Alguma faces da FEI Face Database (b) Face média (c) Eigenfaces

4.1.2.3 Implementação e avaliação do algoritmo

O Código 5 apresenta como realizar o reconhecimento facial com Eigenfaces utilizando a classe `EigenFaceRecognizer` da biblioteca OpenCV.

Foram realizados três testes para avaliar o desempenho do algoritmo com diferentes bases de faces. O primeiro utilizou a base da AT&T, um subconjunto da FERET database foi utilizado para o segundo teste e o terceiro foi feito com algumas imagens selecionadas da Extended Yale Face Database B.

No primeiro teste, foram utilizadas 9 fotos de cada um dos 40 indivíduos para o treino e uma foto de cada para a avaliação. Das 40 faces de teste, o algoritmo reconheceu 38 faces com sucesso e errou o reconhecimento de duas. A taxa de reconhecimento obtida foi de 95%, o que sugere que o método das eigenfaces tem bom resultado quando utilizado com imagens frontais de poucas pessoas em ambientes com iluminação controlada. Esse resultado pode ser visto na Figura 22.

No segundo teste, usando um subconjunto da base de faces FERET composto por 1542 imagens frontais de 865 pessoas para treino e uma foto de avaliação para cada pessoa, o algoritmo obteve 550 acertos e 315 erros, ou seja, uma taxa de apenas 63,6% de acertos mesmo com pose e iluminação controladas. Isso se deve, provavelmente, pelo grande número de indivíduos e pelo baixo número de imagens de treino por indivíduo.

No terceiro teste, foram utilizadas 49 imagens de cada um dos 38 indivíduos para treinamento e uma foto de cada para testar o reconhecimento. O algoritmo obteve 31 acertos e 7 erros, ou seja, 81,6% de acertos. Esse resultado está exposto na Figura 24.

O reconhecimento facial com eigenfaces é muito vulnerável a mudanças na iluminação e na orientação da face. Em (BAGGIO, 2012) são dadas algumas sugestões de pré-processamento, ilustradas na Figura 21, que ajudam a diminuir o impacto desses fatores:

- **Transformações geométricas**: redimensionar, rotacionar e transladar a imagem de forma que os olhos fiquem alinhados.
- ***Cropping***: remoção da testa, queixo, orelhas e fundo da imagem.
- **Equalização do histograma**: deve ser feito para ambos os lados da face para ajustar o brilho e o contraste em cada lado de forma independente.
- **Suavização usando filtros bilaterais**: esse processo reduz o ruído da imagem.
- **Máscara elíptica**: máscara que contorna a face a fim de remover o cabelo e o fundo da imagem.

Figura 21 – Pré-processamentos sugeridos por Baggio (2012) para melhorar a performance dos algoritmos de reconhecimento facial

4.1.2.4 Comparação com outros métodos

Além do Eigenfaces, os métodos Fisherfaces e Local Binary Patterns Histograms também estão implementados na biblioteca OpenCV. Para fins de comparação, os testes realizados com o Eigenfaces também foram realizados com esses dois métodos. Os resultados podem ser vistos na Tabela 4 [1].

Tabela 3 – Bancos de faces utilizados para testes

Banco de faces	**Indivíduos**	**Fotos de treino**
AT&T	40	360
FERET	865	1542
Extended Yale	38	1862

Tabela 4 – Comparação dos algoritmos de reconhecimento facial

Método	**Banco de faces**	**Acertos**	**Erros**	**Taxa de acertos**
Eigenfaces	AT&T	38	2	95,0%
	FERET	550	315	63,4%
	Extended Yale	31	7	81,6%
Fisherfaces	AT&T	39	1	97,5%
	FERET[1]	-	-	-
	Extended Yale	38	0	100%
LBPH	AT&T	40	40	100%
	FERET	786	81	90,7%
	Extended Yale	36	2	94,7%

Pelos resultados obtidos nos testes realizados neste trabalho e em outros artigos, pode-se concluir que o método de reconhecimento utilizando eigenfaces é bastante rudimentar quando comparado a métodos mais recentes. Seu estudo é importante por ter sido um pioneiro na área de reconhecimento e servido de base para outros algoritmos, porém seu desempenho deixa a desejar, especialmente em ambientes pouco controlados.

[1] Devido à pouca quantidade de imagens por indivíduo, o teste com a FERET não pode ser realizado com o Fisherfaces.

Figura 22 – Teste do reconhecimento usando Eigenfaces com a base de faces da AT&T. Em verde as detecções corretas, em vermelho as erradas

Figura 23 – Teste do reconhecimento usando Eigenfaces com a base de faces FERET. Em verde as detecções corretas, em vermelho as erradas

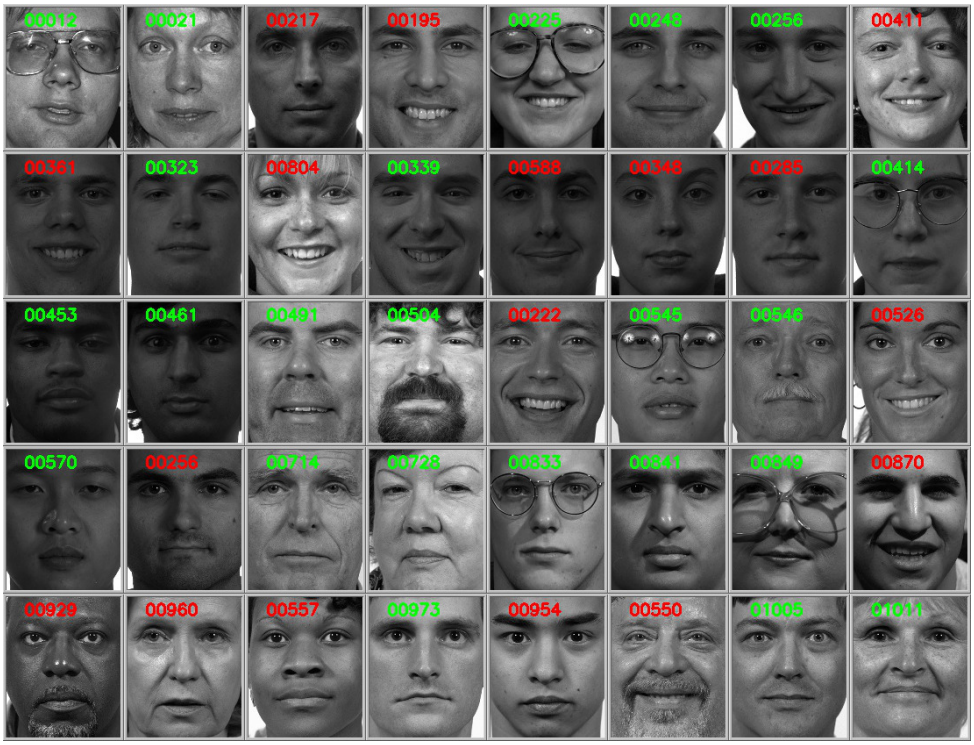
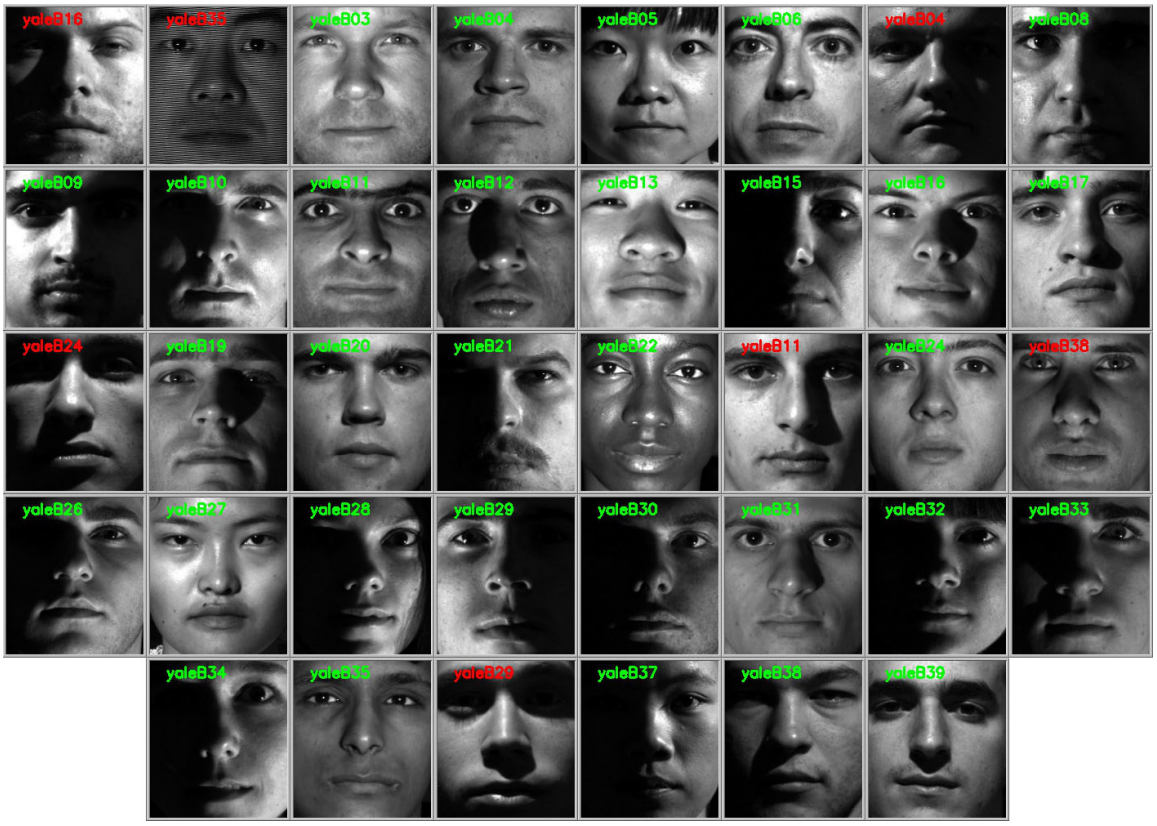


Figura 24 – Teste do reconhecimento usando Eigenfaces com o Extended Yale Face Database B. Em verde as detecções corretas, em vermelho as erradas



5 Projeto do sistema

O sistema proposto será composto por dois módulos. O primeiro corresponde à etapa de captura de fotos, detecção facial e upload das faces a serem processadas pelo segundo módulo, que, por sua vez, tenta verificar se a mesma face já foi analisada no passado e extrai características como idade, emoções e gênero.

5.1 Módulo 1

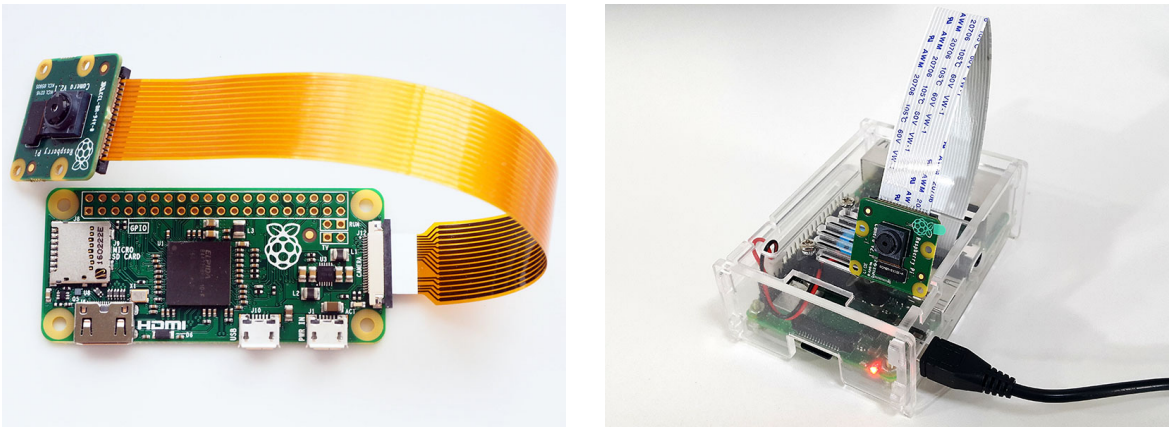
O módulo 1 é o foco deste trabalho. Sua principal função é capturar as fotos a serem analisadas no segundo módulo. Para minimizar o volume de dados trafegados pela rede e concentrar os esforços dos algoritmos de análise apenas nas áreas de interesse, o primeiro módulo não pode enviar todas as imagens capturadas, sendo necessário um pré-processamento que ignore as fotos que não contenham faces e também remova o fundo das fotos que contenham faces.

5.1.1 Raspberry Pi

Câmeras comuns não possuem poder computacional para realizar o pré-processamento desejado e computadores tradicionais são caros, grandes e muito pesados para esse propósito.

O meio termo ideal são os computadores de baixo custo em placa única, que podem ser posicionados de forma discreta e, apesar de limitados, são capazes de executar sistemas operacionais como o Linux ARM.

Figura 25 – Raspberry Pis com câmera



(a) Raspberry Pi Zero. Fonte: (UPTON, 2016) (b) Raspberry Pi 3 modelo B+ em um case de acrílico

O computador escolhido para este projeto foi o Raspberry Pi nos modelos Raspberry Pi Zero (Figura 25a) e Raspberry Pi 3 B+ (Figura 25b) devido ao seu baixo custo, alta disponibilidade e por possuir interface para câmera (CSI). O projeto completo custou menos de R$ 500,00.

O Raspberry Pi Zero possui um processador ARM single-core de 1 GHz e 512 MB de memória SDRAM, enquanto que o Raspberry Pi 3 modelo B+ possui um processador quad-core 64-bit de 1,4 GHz e 1 GB de memória SDRAM. O Raspberry Pi Zero não possui WiFi e Ethernet como o modelo 3 B+, porém é possível utilizar adaptadores USB para conexão de rede.

O sistema operacional escolhido para ser instalado no Raspberry Pi foi o Raspbian Stretch Lite, uma distribuição de Linux baseada no Debian otimizada especificamente para o Raspberry Pi.

5.1.2 Detecção facial

O programa para selecionar apenas as regiões com faces deve utilizar um algoritmo eficiente de detecção facial capaz de ser executado a pelo menos um quadro por segundo no hardware e sistema operacional escolhidos.

O algoritmo de detecção facial selecionado para o primeiro módulo foi o Viola-Jones, pois, como estudado em capítulos anteriores, ele é eficiente o suficiente para ser executado em tempo real e está implementado na biblioteca multiplataforma OpenCV.

O Código 2 apresenta a implementação do sistema de detecção facial utilizando a câmera do Raspberry Pi.

As faces extraídas pelo detector facial, podem ser salvas no cartão de memória para serem acessadas posteriormente ou enviadas pela rede para serem processadas imediatamente.

5.2 Módulo 2

O segundo módulo é o responsável por extrair informações das faces enviadas pelo primeiro módulo. Por limitação de armazenamento e processamento, foi decidido que esse módulo não será executado a partir do Raspberry Pi, mas sim em um servidor local ou na nuvem.

Além de reconhecer se uma face já foi analisada no passado, também é desejável estimar informações como idade, sentimento e gênero. Modificações do Eigenfaces podem ser utilizadas para todas essas tarefas (KEKRE; THEPADE; CHOPRA, 2010; CALDER et al., 2001; PANTIC; ROTHKRANTZ, 2000), porém a performance de métodos recentes,

muitos utilizando redes neurais, é consideravelmente maior (DUAN et al., 2018; DEHGHAN et al., 2017; GURNANI et al., 2018).

A performance da classificação de gênero de ferramentas comerciais foi avaliada em (BUOLAMWINI; GEBRU, 2018). As APIs de análise facial da Microsoft, IBM e Face++ (MICROSOFT, 2018; IBM, 2018; FACE++, 2018) se mostraram promissoras. A *Microsoft Face API* obteve o melhor resultado com uma taxa de acertos (TPR) de 93,7%.

A API da Microsoft também foi analisada por (DEHGHAN et al., 2017). Ela obteve uma acurácia de 61,3% para reconhecimento de emoções e 90,86% para reconhecimento de gênero. Os autores usaram um banco de imagens próprio para reconhecimento de emoções e o Adience benchmark para reconhecimento de gênero.

Pelos bons resultados apresentados, familiaridade com a Azure, preço e possibilidade de integração com outros serviços, a Face API da Microsoft foi escolhida para o segundo módulo.

Como pode ser visto na Figura 26, existe um modo gratuito que permite até 20 transações por minuto e 30000 transações por mês. Essa quantia é suficiente para aplicações pequenas. O pré-processamento realizado durante o primeiro módulo ajuda a reduzir o número de chamadas à API.

A Figura 27b mostra como utilizar a interface em Python da API para extrair informações de uma face. Os atributos desejados, como idade, gênero e emoção, são passados por parâmetro para o método `CF.face.detect`, que retorna um json com as informações.

A Figura 27a é a captura de tela de um aplicativo de exemplo que demonstra todas as funcionalidades da API. Os resultados foram bem satisfatórios pelos testes realizados.

Figura 26 – Criação das chaves de acesso à Microsoft Face API

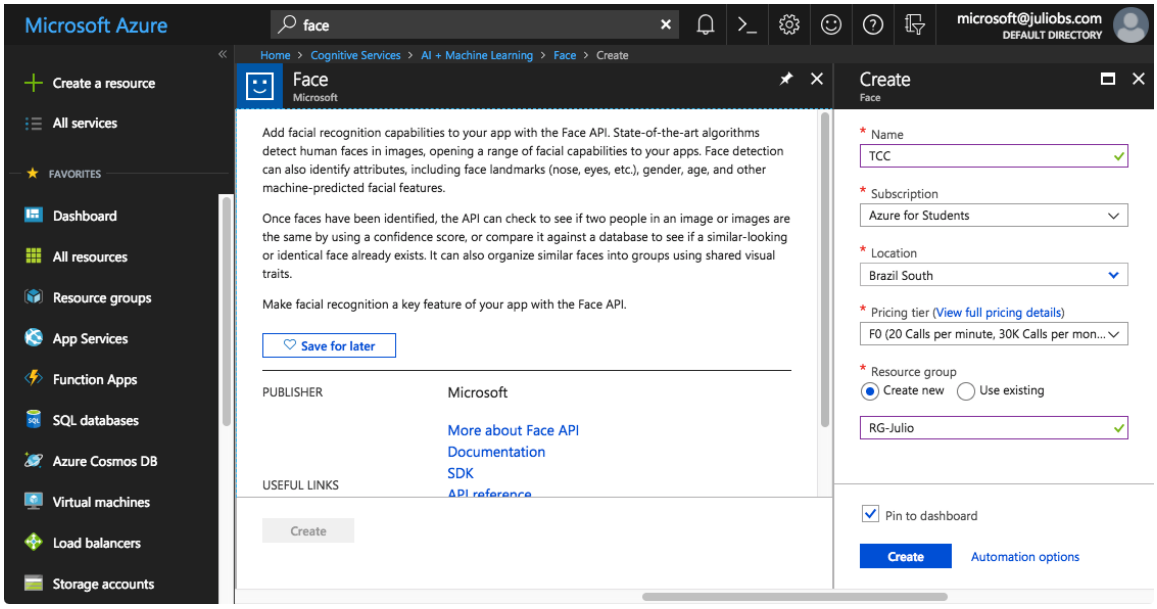
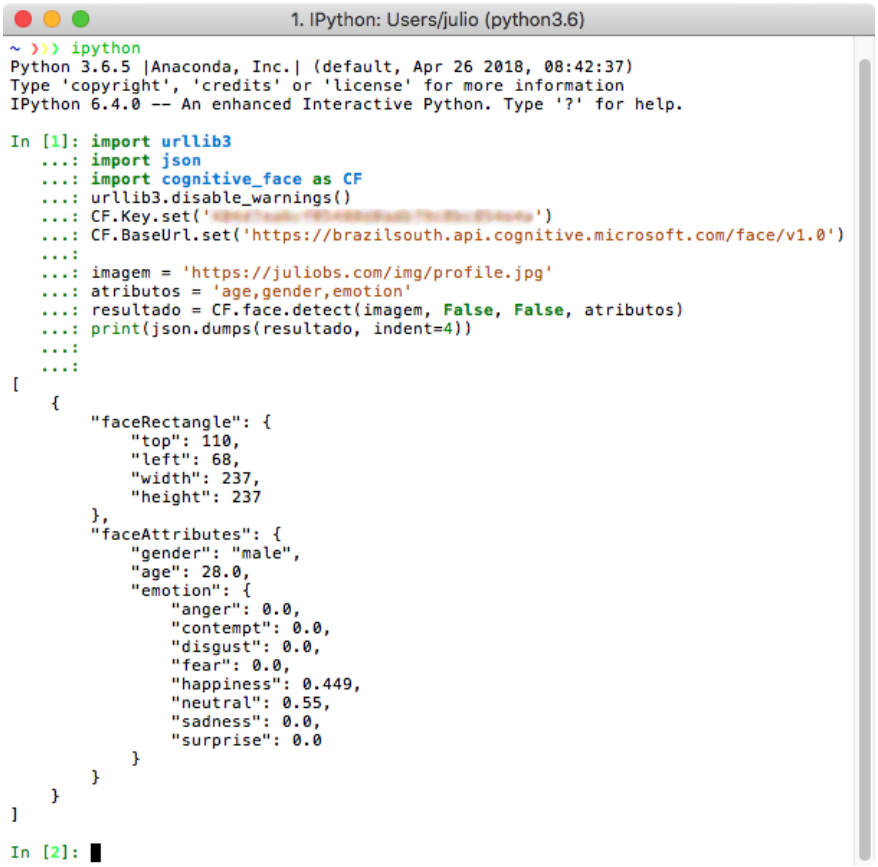
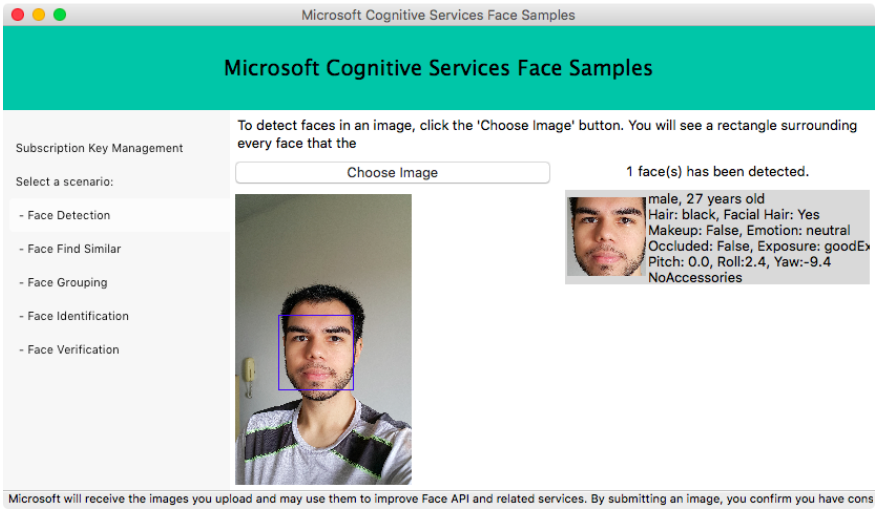


Figura 27 – Teste da Microsoft Face API



(a) Python SDK para a Microsoft Face API



(b) Aplicativo de exemplo

Conclusão

O presente trabalho apresentou uma visão ampla dos conceitos e métodos de detecção e reconhecimento facial e explicações detalhadas sobre os algoritmos Viola-Jones e Eigenfaces, incluindo a implementação de sistemas computacionais capazes de detectar e reconhecer faces utilizando esses algoritmos.

Para alcançar o objetivo de fornecer uma base teórica para a implantação de um sistema de detecção e reconhecimento de faces, foi necessária uma extensa pesquisa bibliográfica e a conclusão de cursos sobre visão computacional.

Foi possível mostrar a validade dos algoritmos Viola-Jones, Eigenfaces, Fisherfaces e LBPH utilizando a biblioteca OpenCV e comparar suas performances. Foi mostrado que a taxa de acertos do Eigenfaces é baixa quando comparada a outros métodos.

Ficou claro que reconhecimento facial irrestrito ainda é um desafio na área de processamento de imagens e precisa superar diversas dificuldades geradas por fatores como iluminação, baixa resolução, variação de pose, oclusão e expressão facial.

Foi mostrado que soluções comerciais, como a Face API da Microsoft, oferecem o estado da arte em diversas tarefas como detecção facial, reconhecimento facial, detecção de emoções e identificação de gênero.

Foi mostrado que o Raspberry Pi pode ser utilizado como uma solução barata de câmera inteligente capaz de detectar e pré-processar as faces de forma a reduzir o tráfego e as chamadas à API.

Sugestão de melhorias e trabalhos futuros

A principal dificuldade em detecção de faces com Viola-Jones é obter um classificador que tenha altas taxas de detecção e poucos falso-positivos sem comprometer a performance.

O classificador treinado na seção 3.2 retornou muitos falso-positivos. Para resolver esse problema, deve-se utilizar mais imagens positivas e negativas, que devem ser bem selecionadas, aumentar o número de estágios da cascata e diminuir a taxa máxima de falso-positivos (maxFalseAlarmRate).

O treinamento de um classificador em cascata com características Haar-like é muito demorado. Para agilizar o processo, ele pode ser treinado em uma arquitetura na nuvem como a Amazon EC2. Também é possível utilizar outros métodos como o Local Binary Patterns (LBP), que é mais rápido de treinar e possui uma acurácia comparável, o SURF cascade, que, além de ser mais rápido de treinar, possui maior acurácia de detecção (LI;

ZHANG, 2013; JAIN; LEARNED-MILLER, 2010) ou métodos mais recentes baseados em redes neurais, como o RSA (LIU et al., 2017) e o S^3FD (ZHANG et al., 2017).

Pela pesquisa bibliográfica, foi possível verificar que redes neurais são o estado da arte em reconhecimento facial e extração de características. Um trabalho futuro poderia estudar esse assunto e desenvolver uma solução que não dependa de ferramentas comerciais como a Face API da Microsoft. Algumas bibliotecas podem ser utilizadas no lugar da OpenCV como a *dlib* e a *OpenFace*.

Uma última sugestão é imprimir um case em 3D que comporte a câmera de forma discreta e mantenha o Raspberry Pi resfriado.

Referências

AMOS, B.; LUDWICZUK, B.; SATYANARAYANAN, M. *OpenFace: A general-purpose face recognition library with mobile applications*. [S.l.], 2016. Citado na página 42.

ANDREWS, H. Two-dimensional transforms. In: *Picture processing and digital filtering*. [S.l.]: Springer, 1975. p. 21–68. Citado na página 43.

ANGELOVA, A.; ABU-MOSTAFAM, Y.; PERONA, P. Pruning training sets for learning of object categories. In: IEEE. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. [S.l.], 2005. v. 1, p. 494–501. Citado na página 33.

BAGGIO, D. L. *Mastering OpenCV with practical computer vision projects*. [S.l.]: Packt Publishing Ltd, 2012. Citado 2 vezes nas páginas 47 e 48.

BAILLY-BAILLIéRE, E. et al. The banca database and evaluation protocol. In: SPRINGER. *International conference on Audio-and video-based biometric person authentication*. [S.l.], 2003. p. 625–638. Citado na página 33.

BELHUMEUR, P. N.; HESPANHA, J. P.; KRIEGMAN, D. J. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, v. 19, n. 7, p. 711–720, 1997. Citado 2 vezes nas páginas 35 e 41.

BLEDSOE, W. e. a. *Facial Recognition Project Report*. [S.l.], 1964. Citado na página 17.

BLEDSOE, W. W. Man-machine facial recognition. *Rep. PRi*, v. 22, 1966. Citado na página 17.

BLEDSOE, W. W. The model method in facial recognition. *Panoramic Research Inc., Palo Alto, CA, Rep. PR1*, v. 15, p. 47, 1966. Citado na página 17.

BOYER, A. O.; BOYER, R. S. A biographical sketch of w. w. bledsoe. In: _____. *Automated Reasoning: Essays in Honor of Woody Bledsoe*. Dordrecht: Springer Netherlands, 1991. p. 1–29. ISBN 978-94-011-3488-0. Citado na página 17.

BRADSKI, G. R.; PISAREVSKY, V. Intel's computer vision library: applications in calibration, stereo segmentation, tracking, gesture, face and object recognition. In: IEEE. *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*. [S.l.], 2000. v. 2, p. 796–797. Citado na página 23.

BUOLAMWINI, J.; GEBRU, T. Gender shades: Intersectional accuracy disparities in commercial gender classification. In: *Conference on Fairness, Accountability and Transparency*. [S.l.: s.n.], 2018. p. 77–91. Citado na página 53.

CALDER, A. J. et al. A principal component analysis of facial expressions. *Vision research*, Elsevier, v. 41, n. 9, p. 1179–1208, 2001. Citado na página 52.

CEN, K. Study of viola-jones real time face detector. *Stanford University*, 2016. Citado 2 vezes nas páginas 17 e 20.

CHAN, H.; BLEDSOE, W. A man-machine facial recognition system: some preliminary results. *Panoramic Research Inc., Palo Alto, CA, USA1965*, 1965. Citado na página 17.

COHN, J. F. et al. Automated face analysis by feature point tracking has high concurrent validity with manual facs coding. *Psychophysiology*, Cambridge University Press, v. 36, n. 1, p. 35–43, 1999. Citado na página 33.

COLLOBERT, M. et al. Listen: A system for locating and tracking individual speakers. In: IEEE. *Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference on*. [S.l.], 1996. p. 283–288. Citado na página 21.

CRAW, I.; TOCK, D.; BENNETT, A. Finding face features. In: SPRINGER. *European Conference on Computer Vision*. [S.l.], 1992. p. 92–96. Citado na página 21.

CROW, F. C. Summed-area tables for texture mapping. In: ACM. *ACM SIGGRAPH computer graphics*. [S.l.], 1984. v. 18, n. 3, p. 207–212. Citado na página 25.

DAI, Y.; NAKANO, Y. Face-texture model based on sgld and its application in face detection in a color scene. *Pattern recognition*, Elsevier, v. 29, n. 6, p. 1007–1017, 1996. Citado na página 21.

DATTA, A.; DATTA, M.; BANERJEE, P. *Face Detection and Recognition: Theory and Practice*. [S.l.]: CRC Press, 2015. ISBN 9781482226577. Citado 3 vezes nas páginas 10, 42 e 88.

DEHGHAN, A. et al. Dager: Deep age, gender and emotion recognition using convolutional neural network. *arXiv preprint arXiv:1702.04280*, 2017. Citado na página 53.

DHINGRA, A. Face identification and clustering. *arXiv preprint arXiv:1704.08328*, 2017. Citado na página 41.

DUAN, M. et al. A hybrid deep learning cnn–elm for age and gender classification. *Neurocomputing*, Elsevier, v. 275, p. 448–461, 2018. Citado na página 53.

ETEMAD, K.; CHELLAPPA, R. Face recognition using discriminant eigenvectors. In: IEEE. *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*. [S.l.], 1996. v. 4, p. 2148–2151. Citado na página 41.

FACE++. *Face++ API*. 2018. Disponível em: <https://www.faceplusplus.com/face-detection/>. Acesso em: 27 jun. 2018. Citado na página 53.

FANTZ, R. L. The origin of form perception. *Scientific American*, JSTOR, v. 204, n. 5, p. 66–73, 1961. Citado na página 17.

FOX, N. A.; O'MULLANE, B. A.; REILLY, R. B. The realistic multi-modal valid database and visual speaker identification comparison experiments. In: *5th International Conference on Audio-and Video-Based Biometric Person Authentication*. [S.l.: s.n.], 2005. Citado na página 35.

FREEMAN, W. T.; ADELSON, E. H. et al. The design and use of steerable filters. *IEEE Transactions on Pattern analysis and machine intelligence*, v. 13, n. 9, p. 891–906, 1991. Citado na página 24.

FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, Elsevier, v. 55, n. 1, p. 119–139, 1997. Citado na página 26.

F.R.S., K. P. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, Taylor & Francis, v. 2, n. 11, p. 559–572, 1901. Citado na página 43.

GAO, W. et al. The cas-peal large-scale chinese face database and baseline evaluations. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, IEEE, v. 38, n. 1, p. 149–161, 2008. Citado na página 33.

GEORGHIADES, A. S.; BELHUMEUR, P. N.; KRIEGMAN, D. J. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 23, n. 6, p. 643–660, 2001. Citado na página 35.

GERBRANDS, J. J. On the relationships between svd, klt and pca. *Pattern recognition*, Elsevier, v. 14, n. 1-6, p. 375–381, 1981. Citado na página 43.

GRAHAM, D. B.; ALLINSON, N. M. Characterising virtual eigensignatures for general purpose face recognition. In: *Face Recognition*. [S.l.]: Springer, 1998. p. 446–456. Citado na página 34.

GREENSPAN, H. et al. Overcomplete steerable pyramid filters and rotation invariance. *Proc. IEEE CVPR, 1994*, IEEE Computer Society Press, 1994. Citado na página 24.

GRGIC, M.; DELAC, K. Face recognition homepage. *Zagreb, Croatia*, v. 324, 2013. Disponível em: <http://www.face-rec.org/databases/>. Citado na página 33.

GURNANI, A. et al. Vegac: Visual saliency-based age, gender, and facial expression classification using convolutional neural networks. *arXiv preprint arXiv:1803.05719*, 2018. Citado na página 53.

HANCOCK, P. *Psychological image collection at stirling*. 2004. Disponível em: <http://pics.psych.stir.ac.uk/>. Citado na página 34.

HE, K. et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2015. p. 1026–1034. Citado na página 42.

HEITMEYER, R. Biometric identification promises fast and secure processing of airline passengers. *ICAO journal*, v. 55, n. 9, p. 10–11, 2000. Citado na página 41.

HUANG, G. B. et al. *Labeled faces in the wild: A database for studying face recognition in unconstrained environments*. [S.l.], 2007. Citado 2 vezes nas páginas 33 e 34.

IBM. *IBM Watson Visual Recognition*. 2018. Disponível em: <https://www.ibm.com/watson/services/visual-recognition/>. Acesso em: 27 jun. 2018. Citado na página 53.

JAFRI, R.; ARABNIA, H. R. A survey of face recognition techniques. *Jips*, v. 5, n. 2, p. 41–68, 2009. Citado na página 41.

JAIN, A. K.; LI, S. Z. *Handbook of face recognition*. [S.l.]: Springer, 2011. Citado na página 41.

JAIN, V.; LEARNED-MILLER, E. *FDDB: A Benchmark for Face Detection in Unconstrained Settings*. [S.l.], 2010. Citado 2 vezes nas páginas 32 e 57.

JENSEN, O. H. *Implementing the Viola-Jones face detection algorithm*. Dissertação (Mestrado) — Technical University of Denmark, DTU, DK-2800 Kgs. Lyngby, Denmark, 2008. Citado na página 23.

JESORSKY, O.; KIRCHBERG, K. J.; FRISCHHOLZ, R. W. Robust face detection using the hausdorff distance. In: SPRINGER. *International Conference on Audio-and Video-Based Biometric Person Authentication*. [S.l.], 2001. p. 90–95. Citado na página 33.

JOLLIFFE, I. T. *Principal Component Analysis*. [S.l.]: Springer Science & Business Media, 2002. Citado na página 43.

JONES, M.; VIOLA, P. Fast multi-view face detection. *Mitsubishi Electric Research Lab TR-20003-96*, v. 3, n. 14, p. 2, 2003. Citado na página 23.

JUNIOR, L. O.; THOMAZ, C. *Captura e alinhamento de imagens: um banco de faces brasileiro*. [S.l.], 2006. Citado na página 34.

JúNIOR, E. G. L. et al. Um robusto reconhecimento facial por filtro de gabor curvo e entropia. *Revista de Sistemas e Computação-RSC*, v. 6, n. 1, 2016. Citado na página 42.

KAEHLER, A.; BRADSKI, G. *Learning OpenCV 3: computer vision in C++ with the OpenCV library*. [S.l.]: "O'Reilly Media, Inc.", 2016. Citado na página 33.

KANADE, T. Picture processing by computer complex and recognition of human faces. *PhD Thesis, Kyoto University*, 1973. Citado 2 vezes nas páginas 17 e 41.

KEKRE, H.; THEPADE, S. D.; CHOPRA, T. Face and gender recognition using principal component analysis. *International Journal on Computer Science and Engineering*, v. 2, n. 4, p. 959–964, 2010. Citado na página 52.

KELLY, M. D. *Visual identification of people by computer*. [S.l.], 1970. Citado na página 17.

KIRBY, M.; SIROVICH, L. Low-dimensional procedure for the characterization of human faces. *Josa a*, Optical Society of America, v. 4, n. 3, p. 519–524, 1987. Citado 2 vezes nas páginas 41 e 44.

KIRBY, M.; SIROVICH, L. Application of the karhunen-loeve procedure for the characterization of human faces. *IEEE Transactions on Pattern analysis and Machine intelligence*, IEEE, v. 12, n. 1, p. 103–108, 1990. Citado na página 41.

KJELDSEN, R.; KENDER, J. Finding skin in color images. In: IEEE. *Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference on*. [S.l.], 1996. p. 312–317. Citado na página 21.

KLEINER, M.; WALLRAVEN, C.; BÜLTHOFF, H. H. The mpi videolab-a system for high quality synchronous recording of video and audio from multiple viewpoints. *MPI-Tech. Reports*, Citeseer, n. 123, 2004. Citado na página 34.

KLONTZ, J. C. et al. Open source biometric recognition. In: IEEE. *Biometrics: Theory, Applications and Systems (BTAS), 2013 IEEE Sixth International Conference on*. [S.l.], 2013. p. 1–8. Citado na página 23.

KUMAR, N. et al. Attribute and simile classifiers for face verification. In: IEEE. *Computer Vision, 2009 IEEE 12th International Conference on*. [S.l.], 2009. p. 365–372. Citado na página 42.

LADES, M. et al. Distortion invariant object recognition in the dynamic link architecture. *IEEE Transactions on computers*, IEEE, v. 42, n. 3, p. 300–311, 1993. Citado na página 42.

LANITIS, A.; TAYLOR, C. J.; COOTES, T. F. Automatic face identification system using flexible appearance models. *Image and vision computing*, Elsevier, v. 13, n. 5, p. 393–401, 1995. Citado 2 vezes nas páginas 21 e 22.

LEARNED-MILLER, E. et al. Labeled faces in the wild: A survey. In: *Advances in face detection and facial image analysis*. [S.l.]: Springer, 2016. p. 189–248. Citado na página 42.

LEE, K.; HO, J.; KRIEGMAN, D. Acquiring linear subspaces for face recognition under variable lighting. *IEEE Trans. Pattern Anal. Mach. Intelligence*, v. 27, n. 5, p. 684–698, 2005. Citado na página 35.

LEUNG, T. K.; BURL, M. C.; PERONA, P. Finding faces in cluttered scenes using random labeled graph matching. In: IEEE. *Computer Vision, 1995. Proceedings., Fifth International Conference on*. [S.l.], 1995. p. 637–644. Citado 2 vezes nas páginas 21 e 22.

LEW, M. S. Information theoretic view-based and modular face detection. In: IEEE. *Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference on*. [S.l.], 1996. p. 198–203. Citado na página 21.

LI, J.; ZHANG, Y. Learning surf cascade for fast and accurate object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2013. p. 3468–3475. Citado na página 57.

LI, J.-B.; CHU, S.-C.; PAN, J.-S. *Kernel Learning Algorithms for Face Recognition*. [S.l.]: Springer, 2016. Citado na página 41.

LIENHART, R.; KURANOV, A.; PISAREVSKY, V. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In: SPRINGER. *Joint Pattern Recognition Symposium*. [S.l.], 2003. p. 297–304. Citado na página 36.

LIENHART, R.; MAYDT, J. An extended set of haar-like features for rapid object detection. In: IEEE. *Image Processing. 2002. Proceedings. 2002 International Conference on*. [S.l.], 2002. v. 1, p. I–I. Citado na página 24.

LIU, Y. et al. Recurrent scale approximation for object detection in cnn. In: *IEEE international conference on computer vision*. [S.l.: s.n.], 2017. v. 5. Citado na página 57.

LIU, Z. et al. Deep learning face attributes in the wild. In: *Proceedings of International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2015. Citado na página 33.

LU, C.; TANG, X. Surpassing human-level face verification performance on lfw with gaussianface. In: *AAAI*. [S.l.: s.n.], 2015. p. 3811–3819. Citado na página 42.

LYONS, M. et al. Coding facial expressions with gabor wavelets. In: IEEE. *Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on*. [S.l.], 1998. p. 200–205. Citado na página 34.

MARSZALEC, E. A. et al. Physics-based face database for color research. *Journal of Electronic Imaging*, International Society for Optics and Photonics, v. 9, n. 1, p. 32–39, 2000. Citado na página 35.

MARTINEZ, A. M. The ar face database. *CVC Technical Report24*, 1998. Citado na página 33.

MCKENNA, S. J.; GONG, S.; RAJA, Y. Modelling facial colour and identity with gaussian mixtures. *Pattern recognition*, Elsevier, v. 31, n. 12, p. 1883–1892, 1998. Citado na página 21.

MESSER, K. et al. Xm2vtsdb: The extended m2vts database. In: *Second international conference on audio and video-based biometric person authentication*. [S.l.: s.n.], 1999. v. 964, p. 965–966. Citado na página 34.

MESSOM, C. H.; BARCZAK, A. L. Stream processing for fast and efficient rotated haar-like features using rotated integral images. *International journal of intelligent systems technologies and applications*, Inderscience Publishers, v. 7, n. 1, p. 40–57, 2009. Citado na página 24.

MICROSOFT. *Microsoft Face API*. 2018. Disponível em: <https://azure.microsoft.com/en-us/services/cognitive-services/face/>. Acesso em: 27 jun. 2018. Citado na página 53.

MORTON, J.; JOHNSON, M. H. Conspec and conlern: a two-process theory of infant face recognition. *Psychological review*, American Psychological Association, v. 98, n. 2, p. 164, 1991. Citado na página 17.

MOZAFFARI, S.; BEHRAVAN, H. Twins facial similarity impact on conventional face recognition systems. In: IEEE. *Electrical Engineering (ICEE), 2011 19th Iranian Conference on*. [S.l.], 2011. p. 1–6. Citado na página 33.

MUSTAFA, R.; MIN, Y.; ZHU, D. Obscenity detection using haar-like features and gentle adaboost classifier. *The Scientific World Journal*, Hindawi, v. 2014, 2014. Citado na página 23.

NETO, W. F. *Varejistas usam cada vez mais o reconhecimento facial para encontrar grandes compradores e... ladrões*. 2017. Disponível em: <http://reconhecimentofacial.com.br/2017/01/25/varejistas-usam-cada-vez-mais-o-reconhecimento-facial-para-encontrar-grandes-compradores-e-assaltantes/>. Acesso em: 27 jun. 2018. Citado na página 17.

NEWS, B. *Facewatch 'thief recognition' CCTV on trial in UK stores*. 2015. Disponível em: <https://www.bbc.com/news/technology-35111363>. Acesso em: 27 jun. 2018. Citado na página 17.

OJALA, T.; PIETIKAINEN, M.; HARWOOD, D. Performance evaluation of texture measures with classification based on kullback discrimination of distributions. In: IEEE. *Pattern Recognition, 1994. Vol. 1-Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on*. [S.l.], 1994. v. 1, p. 582–585. Citado na página 41.

OMAIA, D. e. a. *Um sistema para detecção e reconhecimento de face em vídeo utilizando a transformada cosseno discreta*. Dissertação (Mestrado) — Universidade Federal da Paraíba, 2009. Citado na página 23.

OSUNA, E.; FREUND, R.; GIROSIT, F. Training support vector machines: an application to face detection. In: IEEE. *Computer vision and pattern recognition, 1997. Proceedings., 1997 IEEE computer society conference on*. [S.l.], 1997. p. 130–136. Citado na página 21.

PANTIC, M.; ROTHKRANTZ, L. J. M. Automatic analysis of facial expressions: The state of the art. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, v. 22, n. 12, p. 1424–1445, 2000. Citado na página 52.

PAPAGEORGIOU, C. P.; OREN, M.; POGGIO, T. A general framework for object detection. In: IEEE. *Computer vision, 1998. sixth international conference on*. [S.l.], 1998. p. 555–562. Citado na página 23.

PHILLIPS, P. J. et al. Overview of the face recognition grand challenge. In: IEEE. *Computer vision and pattern recognition, 2005. CVPR 2005. IEEE computer society conference on*. [S.l.], 2005. v. 1, p. 947–954. Citado na página 34.

PHILLIPS, P. J. et al. The feret database and evaluation procedure for face-recognition algorithms. *Image and vision computing*, Elsevier, v. 16, n. 5, p. 295–306, 1998. Citado na página 34.

RAJAGOPALAN, A. N. et al. Finding faces in photographs. In: IEEE. *Computer Vision, 1998. Sixth International Conference on*. [S.l.], 1998. p. 640–645. Citado na página 21.

REPORTS, C. *Facial Recognition: Who's Tracking You in Public?* 2015. Disponível em: <https://www.consumerreports.org/privacy/facial-recognition-who-is-tracking-you-in-public1/>. Acesso em: 27 jun. 2018. Citado na página 17.

ROBERTS, J. J. *Walmart's Use of Sci-fi Tech To Spot Shoplifters Raises Privacy Questions*. 2015. Disponível em: <http://fortune.com/2015/11/09/wal-mart-facial-recognition/>. Acesso em: 27 jun. 2018. Citado na página 17.

ROWLEY, H. A.; BALUJA, S.; KANADE, T. Neural network-based face detection. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, v. 20, n. 1, p. 23–38, 1998. Citado na página 21.

SALOMãO, K. *Loja Pontofrio tem de vitrine virtual a reconhecimento facial*. 2018. Disponível em: <https://exame.abril.com.br/negocios/loja-pontofrio-tem-de-vitrine-virtual-a-reconhecimento-facial/>. Acesso em: 27 jun. 2018. Citado na página 17.

SAMARIA, F. S.; HARTER, A. C. Parameterisation of a stochastic model for human face identification. In: IEEE. *Applications of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on*. [S.l.], 1994. p. 138–142. Citado na página 33.

SANCHEZ, M. R.; MATAS, J.; KITTLER, J. Statistical chromaticity models for lip tracking with b-splines. In: SPRINGER. *International Conference on Audio-and Video-Based Biometric Person Authentication*. [S.l.], 1997. p. 69–76. Citado na página 34.

SANDERSON, C. *Biometric person recognition: Face, speech and fusion*. [S.l.]: VDM Publishing, 2008. v. 4. Citado na página 35.

SANTANA, L. M. Q. de; ROCHA, F. G. Processo de detecção facial utilizando viola; jones. *Interfaces Científicas-Exatas e Tecnológicas*, v. 1, n. 1, p. 35–40, 2015. Citado na página 20.

SCHNEIDERMAN, H.; KANADE, T. Probabilistic modeling of local appearance and spatial relationships for object recognition. In: IEEE. *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*. [S.l.], 1998. p. 45–51. Citado na página 21.

SCHROFF, F.; KALENICHENKO, D.; PHILBIN, J. Facenet: A unified embedding for face recognition and clustering. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2015. p. 815–823. Citado 2 vezes nas páginas 41 e 42.

SHARMA, P.; REILLY, R. B. A colour face image database for benchmarking of automatic face detection algorithms. In: IEEE. *Video/Image Processing and Multimedia Communications, 2003. 4th EURASIP Conference focused on*. [S.l.], 2003. v. 1, p. 423–428. Citado na página 34.

SIM, T.; BAKER, S.; BSAT, M. The cmu pose, illumination, and expression (pie) database. In: IEEE. *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*. [S.l.], 2002. p. 53–58. Citado na página 34.

SINGH, R. et al. Plastic surgery: A new dimension to face recognition. *IEEE Transactions on Information Forensics and Security*, IEEE, v. 5, n. 3, p. 441–448, 2010. Citado na página 34.

SINGH, R.; VATSA, M.; NOORE, A. Face recognition with disguise and single gallery images. *Image and Vision Computing*, Elsevier, v. 27, n. 3, p. 245–257, 2009. Citado 2 vezes nas páginas 33 e 34.

SOCOLINSKY, D. A. et al. Illumination invariant face recognition using thermal infrared imagery. In: IEEE. *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. [S.l.], 2001. v. 1, p. I–I. Citado na página 34.

SPACEK, L. *University of Essex collection of facial images*. 1996. Citado na página 35.

SUN, Y. et al. Deep learning face representation by joint identification-verification. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2014. p. 1988–1996. Citado na página 42.

SUN, Y. et al. Deepid3: Face recognition with very deep neural networks. *arXiv preprint arXiv:1502.00873*, 2015. Citado na página 42.

SUN, Y.; WANG, X.; TANG, X. Deep learning face representation from predicting 10,000 classes. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2014. p. 1891–1898. Citado na página 42.

SUN, Y.; WANG, X.; TANG, X. Deeply learned face representations are sparse, selective, and robust. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2015. p. 2892–2900. Citado na página 42.

SUNG, K.-K.; POGGIO, T. Example-based learning for view-based human face detection. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, v. 20, n. 1, p. 39–51, 1998. Citado na página 21.

TAIGMAN, Y. et al. Deepface: Closing the gap to human-level performance in face verification. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2014. p. 1701–1708. Citado na página 42.

TECHNOLOGY, G. I. of. *The Georgia Tech Face Database*. 1999. Disponível em: <http://www.anefian.com/research/face_reco.htm>. Acesso em: 27 jun. 2018. Citado na página 34.

TURK, M.; PENTLAND, A. Eigenfaces for recognition. *Journal of cognitive neuroscience*, MIT Press, v. 3, n. 1, p. 71–86, 1991. Citado 5 vezes nas páginas 17, 21, 22, 41 e 44.

TURK, M. A.; PENTLAND, A. P. Face recognition using eigenfaces. In: IEEE. *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*. [S.l.], 1991. p. 586–591. Citado na página 44.

UPTON, E. *Zero Grows a Camera Connector*. 2016. Disponível em: <https://www.raspberrypi.org/blog/zero-grows-camera-connector/>. Acesso em: 27 jun. 2018. Citado na página 51.

VIDIT, J.; AMITABHA, M. *The Indian face database*. 2002. Citado na página 34.

VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, v. 1, p. I–I, 2001. Citado 2 vezes nas páginas 23 e 24.

VIOLA, P.; JONES, M. J. Robust real-time face detection. *International Journal of Computer Vision*, Springer, v. 57, n. 2, p. 137–154, 2004. Citado 4 vezes nas páginas 23, 27, 29 e 36.

VISION, O. S. C. *OpenCV documentation: Object Detection Modules*. 2018. Disponível em: <https://docs.opencv.org/master/d5/d54/group___objdetect.html>. Acesso em: 27 jun. 2018. Citado na página 23.

VISION, O. S. C. *OpenCV documentation: Object Detection Modules*. 2018. Disponível em: <https://docs.opencv.org/master/da/d60/tutorial_face_main.html>. Acesso em: 27 jun. 2018. Citado na página 42.

WANG, X.; TANG, X. A unified framework for subspace face recognition. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, v. 26, n. 9, p. 1222–1228, 2004. Citado na página 43.

WATSON, C. I. *Nist mugshot identification database*. 1994. Citado na página 34.

WEBER, M. *Caltech face database*. Online, 1995. Disponível em: <http://www.vision.caltech.edu/html-files/archive.html>. Citado na página 33.

WEYRAUCH, B. et al. Component-based face recognition with 3d morphable models. In: IEEE. *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on*. [S.l.], 2004. p. 85–85. Citado na página 34.

WISKOTT, L. et al. Face recognition by elastic bunch graph matching. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, v. 19, n. 7, p. 775–779, 1997. Citado na página 42.

YANG, G.; HUANG, T. S. Human face detection in a complex background. *Pattern recognition*, Elsevier, v. 27, n. 1, p. 53–63, 1994. Citado 2 vezes nas páginas 21 e 22.

YANG, J.; WAIBEL, A. A real-time face tracker. In: IEEE. *Applications of Computer Vision, 1996. WACV'96., Proceedings 3rd IEEE Workshop on*. [S.l.], 1996. p. 142–147. Citado na página 21.

YANG, M.-H.; KRIEGMAN, D. J.; AHUJA, N. Detecting faces in images: A survey. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, v. 24, n. 1, p. 34–58, 2002. Citado 3 vezes nas páginas 12, 20 e 21.

YI, D. et al. Learning face representation from scratch. *arXiv preprint arXiv:1411.7923*, 2014. Citado na página 42.

YOW, K. C.; CIPOLLA, R. Feature-based human face detection. *Image and vision computing*, Elsevier, v. 15, n. 9, p. 713–735, 1997. Citado na página 21.

ZHANG, C.; ZHANG, Z. *A survey of recent advances in face detection*. [S.l.], 2010. Citado na página 23.

ZHANG, S. et al. S3fd: Single shot scale-invariant face detector. In: IEEE. *Computer Vision (ICCV), 2017 IEEE International Conference on*. [S.l.], 2017. p. 192–201. Citado na página 57.

ZHOU, E.; CAO, Z.; YIN, Q. Naive-deep face recognition: Touching the limit of lfw benchmark or not? *arXiv preprint arXiv:1501.04690*, 2015. Citado na página 42.

ZHU, Z. et al. Recover canonical-view faces in the wild with deep neural networks. *arXiv preprint arXiv:1404.3543*, 2014. Citado na página 42.

Apêndices

APÊNDICE A – Código do detector facial

Código 1 – Detector facial usando a biblioteca OpenCV

```
#!/usr/bin/env python3
# -*-*- encoding: utf-8 -*-*-
# Created: Sat, 26 May 2018 19:32:23 -0300

"""
Detector de faces usando a biblioteca OpenCV.

Uso: detector_facial.py [-c CLASSIFICADOR]

Argumentos opcionais:
  -c CLASSIFICADOR, --classificador CLASSIFICADOR
                        Caminho para classificador em cascata
"""

import os
import sys
import argparse
import logging
import cv2


__author__ = "Julio Batista Silva"
__copyright__ = "Copyright (c) 2018, Julio Batista Silva"
__license__ = "GPL v3"
__version__ = "1.0"
__email__ = "julio@juliobs.com"


SCRIPT_DIR = os.path.dirname(__file__)
SCRIPT_NAME = os.path.basename(__file__)
DEFAULT_CLASSIFIER = 'haarcascades/haarcascade_frontalface_default.xml'


def detecta_faces(imagem, classificador):
    ''' Retorna uma lista com as coordenadas das faces detectadas na imagem
    pelo classificador passado por parâmetro'''
    # Carrega o classificador em cascata passado por parâmetro
    classif = cv2.CascadeClassifier(classificador)

    # Cria cópia em tons de cinza da imagem
    cinza = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)

```

```
43      # Detecta as faces usando a imagem em tons de cinza
44      faces = classif.detectMultiScale(
45          cinza, scaleFactor=1.2, minNeighbors=6, minSize=(30, 30),
46          flags=cv2.cv.CV_HAAR_SCALE_IMAGE)
47
48      return faces
49
50
51  def main(ARGS):
52
53      for img_path in ARGS.imagens:
54          imagem = cv2.imread(img_path)
55
56          if imagem is None:
57              continue
58
59          faces = detecta_faces(imagem, ARGS.classificador)
60
61          if len(faces) > 0:
62              # Desenha um retângulo em volta das faces detectadas
63              for (x, y, w, h) in faces:
64                  cv2.rectangle(imagem, (x, y), (x + w, y + h), (0, 0, 255), 2)
65
66              # Grava imagem com o resultado da detecção
67              pasta = ARGS.saida if ARGS.saida else os.path.dirname(img_path)
68              nome = 'DETECT-' + os.path.basename(img_path)
69              cv2.imwrite(os.path.join(pasta, nome), imagem)
70
71      return 0
72
73
74  if __name__ == "__main__":
75      parser = argparse.ArgumentParser(description='Detecta faces na câmera.')
76      parser.add_argument('-c', '--classificador', type=str,
77                          default=DEFAULT_CLASSIFIER,
78                          help='Caminho para classificador em cascata')
79      parser.add_argument('-i', '--imagens', nargs='*',
80                          help='Caminho para imagem')
81      parser.add_argument('-s', '--saida',
82                          help='Caminho onde detecções serão salvas')
83      parser.add_argument('--versao', action='version',
84                          version='%(prog)s v' + __version__)
85      ARGS = parser.parse_args()
86
87      logging.basicConfig(
88          filename=os.path.join(SCRIPT_DIR, SCRIPT_NAME + '.log'),
89          format='%(asctime)s %(levelname)-8s %(message)s',
90          level=logging.INFO,
91          datefmt='%Y-%m-%d %H:%M:%S')
```

```
92
93      logging.debug("==================Iniciou o script=================")
94
95      try:
96          sys.exit(main())
97      except KeyboardInterrupt:
98          logging.debug("====================Interrompido====================")
99          sys.exit(0)
100     except SystemExit:
101         logging.debug("=================Finalizou o script================")
102     except BaseException:
103         logging.exception('Ocorreu um erro')
```

APÊNDICE B – Código para Raspberry Pi do detector facial usando OpenCV

Código 2 – Detector facial usando OpenCV e picamera

```
#!/usr/bin/env python2
# -*-*- encoding: utf-8 -*-*-
# Created: Thu, 14 Jun 2018 22:33:51 -0300

"""
Detector de faces usando a biblioteca OpenCV no Raspberry Pi.

Uso: detector_facial-raspberry.py [-c CLASSIFICADOR]

Argumentos opcionais:
  -c CLASSIFICADOR, --classificador CLASSIFICADOR
                        Caminho para classificador em cascata
  -d DIR_FOTOS, --dir_fotos DIR_FOTOS
                        Caminho do diretório onde as fotos serão salvas
"""

import os
import sys
import argparse
import logging
import time
import cv2
from picamera import PiCamera
from picamera.array import PiRGBArray


__author__ = "Julio Batista Silva"
__copyright__ = "Copyright (c) 2018, Julio Batista Silva"
__license__ = "GPL v3"
__version__ = "1.0"
__email__ = "julio@juliobs.com"


SCRIPT_DIR = os.path.dirname(__file__)
SCRIPT_NAME = os.path.basename(__file__)


DEFAULT_CLASSIFIER = 'haarcascades/haarcascade_frontalface_default.xml'
DEFAULT_OUTPUT_PATH = 'faces/'
```

```
40
41
42  def detecta_faces(imagem, classificador):
43      '''Retorna uma lista com a posição das faces detectadas usando o
44      classificador passado por parâmetro'''
45
46      # Carrega o classificador em cascata passado por parâmetro
47      cc = cv2.CascadeClassifier(classificador)
48      faces = cc.detectMultiScale(
49          imagem, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30),
50          flags=cv2.cv.CV_HAAR_SCALE_IMAGE)
51      return faces
52
53
54  def main():
55      '''Função principal'''
56
57      # Detecta faces até interrupção
58      with PiCamera(resolution=(640, 480), framerate=24) as camera:
59
60          # Inverte câmera
61          camera.rotation = 180
62
63          # Tempo para esquentar câmera
64          time.sleep(2)
65
66          img_buffer = PiRGBArray(camera, size=camera.resolution)
67
68          # Captura imagens
69          for frame in camera.capture_continuous(img_buffer, format="bgr"):
70              # Horário da captura
71              horario = time.strftime("%Y%m%d%H%M%S")
72
73              # Foto como um array
74              imagem = frame.array
75
76              # Cria cópia em tons de cinza da imagem
77              cinza = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)
78
79              # Detecta as faces usando a imagem em tons de cinza e o
80              # classificador
81              faces = detecta_faces(cinza, ARGS.classificador)
82
83              # Detectou alguma face?
84              if len(faces) > 0:
85                  logging.debug("================Face detectada================")
86
87                  # Contador de faces na imagem
88                  face_num = 0
```

```
89
90                  for (x, y, w, h) in faces:
91                      # Numera face
92                      face_num += 1
93
94                      # Corta face detectada
95                      face = imagem[y:y+h, x:x+w]
96
97                      # Salva imagem cortada
98                      cv2.imwrite(
99                          os.path.join(
100                             ARGS.dir_fotos,
101                             "{}_{}.jpg".format(horario, face_num)),
102                         face)
103
104             # Esvazia captura para reutilizar no próximo frame
105             img_buffer.truncate(0)
106
107             # Espera 1s antes de capturar outra foto
108             time.sleep(1)
109
110     return 0
111
112
113 if __name__ == "__main__":
114     PARSER = argparse.ArgumentParser(description='Detecta faces na câmera.')
115     PARSER.add_argument('-c', '--classificador', dest="classificador",
116                         default=DEFAULT_CLASSIFIER,
117                         help='Caminho para classificador em cascata')
118     PARSER.add_argument('-d', '--dir_fotos', dest="dir_fotos",
119                         default=DEFAULT_OUTPUT_PATH,
120                         help='Caminho do diretório onde as fotos serão salvas')
121     PARSER.add_argument('--versao', action='version',
122                         version='%(prog)s v' + __version__)
123     ARGS = PARSER.parse_args()
124
125     logging.basicConfig(
126         filename=os.path.join(SCRIPT_DIR, SCRIPT_NAME + '.log'),
127         format='%(asctime)s %(levelname)-8s %(message)s',
128         level=logging.DEBUG,
129         datefmt='%Y-%m-%d %H:%M:%S')
130
131     logging.debug("===============Iniciou o script==============")
132
133     try:
134         sys.exit(main())
135     except KeyboardInterrupt:
136         logging.debug("================Interrompido================")
137         sys.exit(0)
```

```
138     except SystemExit:
139         logging.debug("==============Finalizou o script==============")
140     except BaseException:
141         logging.exception('Ocorreu um erro')
```

APÊNDICE C – Código usado para gerar gráficos para ilustrar funcionamento do PCA

Código 3 – Código gerador dos gráficos da Figura 18

```python
#!/usr/bin/env python3
# -*-*- encoding: utf-8 -*-*-
# Created: Sat, 09 Jun 2018 16:54:20 -0300

"""
Gera gráficos para ilustrar a explicação sobre PCA
"""


import numpy as np
from matplotlib import pyplot as plt
from matplotlib2tikz import save as tikz_save


np.random.seed(2018)
r = 0.9

rotacao = np.pi / 6
s = np.sin(rotacao)
c = np.cos(rotacao)

X = np.random.normal(0, [0.3, 0.1], size=(50, 2)).T
R = np.array([[c, -s],
              [s, c]])
X = np.dot(R, X)

#### Figura 1 ####
fig = plt.figure(figsize=(4, 4), facecolor='w')
ax = plt.axes((0, 0, 1, 1), xticks=[], yticks=[], frameon=False)
ax.set_xlim(-1, 1)
ax.set_ylim(-1, 1)

# Eixos x e y
ax.annotate(r'$x$', (-r, 0), (r, 0),
            ha='center', va='center',
            arrowprops=dict(arrowstyle='<->', color='black', lw=1))
ax.annotate(r'$y$', (0, -r), (0, r),
            ha='center', va='center',
            arrowprops=dict(arrowstyle='<->', color='black', lw=1))
```

```
40
41  # Eixos x' e y'
42  ax.annotate(r'$x^\prime$', (-r * c, -r * s), (r * c, r * s),
43              ha='center', va='center',
44              arrowprops=dict(color='black', arrowstyle='<|-|>', alpha=0.8, lw=0.5,
                ↪  ls='--'))
45  ax.annotate(r'$y^\prime$', (r * s, -r * c), (-r * s, r * c),
46              ha='center', va='center',
47              arrowprops=dict(color='black', arrowstyle='<|-|>', alpha=0.8, lw=0.5,
                ↪  ls='--'))
48
49  # Pontos do gráfico de dispersão
50  ax.scatter(X[0], X[1], s=25, lw=0, c='red')
51
52  # Linhas das projeções
53  for v in (X.T):
54      vnorm = np.array([s, -c])
55      d = np.dot(v, vnorm)
56      v1 = v - d * vnorm
57      ax.plot([v[0], v1[0]], [v[1], v1[1]], color='xkcd:baby poop', alpha=0.5,
        ↪  ls='dotted')
58
59  plt.savefig('pca1.pdf')
60  #tikz_save("pca1.tex")
61  plt.show()
62
63  #### Figura 2 ####
64  fig2 = plt.figure(figsize=(5, 5), facecolor='w')
65  ax = plt.axes((0, 0, 1, 1), xticks=[], yticks=[], frameon=False)
66  ax.set_xlim(-1, 1)
67  ax.set_ylim(-1, 1)
68
69  # Eixos x e y
70  ax.annotate(r'$x$', (-r, 0), (r, 0),
71              ha='center', va='center',
72              arrowprops=dict(arrowstyle='<->', color='black', lw=1))
73  ax.annotate(r'$y$', (0, -r), (0, r),
74              ha='center', va='center',
75              arrowprops=dict(arrowstyle='<->', color='black', lw=1))
76
77  # Projeções
78  vnorm = np.array([c, s])
79  vnorm_row = vnorm.reshape((1, 2))
80  c_values = vnorm_row.dot(X)
81  proj = vnorm_row.T.dot(c_values)
82  ax.scatter(proj[0],proj[1], s=25, lw=0, c='red')
83
84  plt.savefig('pca2.pdf')
85  #tikz_save("pca2.tex")
```

```
86 plt.show()
```

APÊNDICE D – Código usado para obter imagens da face média e dos componentes principais

Código 4 – Ilustração

```
#!/usr/bin/env python3
# -*-*- encoding: utf-8 -*-*-
# Created: Sat, 23 Jun 2018 17:57:15 -0300

"""
Para ilustrar o eigenfaces
"""


import os
import sys
import argparse
import logging
import numpy as np
import cv2


__author__ = "Julio Batista Silva"
__copyright__ = "Copyright (c) 2018, Julio Batista Silva"
__license__ = "GPL v3"
__version__ = "1.0"
__email__ = "julio@juliobs.com"


SCRIPT_DIR = os.path.dirname(__file__)
SCRIPT_NAME = os.path.basename(__file__)


def normaliza(arr):
    return cv2.normalize(arr, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8UC3)


def main():
    '''Calcula Eigenfaces usando OpenCV'''

    # Lista de imagens
```

```
37      #from glob import glob
38      #arquivos = glob(f"{faces_dir}/*.jpg")
39      arquivos = ARGS.imagens
40      qtd_imgs = len(arquivos)
41
42      # Cria lista de imagens representadas como array de floats entre 0 e 1
43      imagens = []
44      for arquivo in arquivos:
45          img = cv2.imread(arquivo)
46          img = np.float32(img) / 255.0
47          imagens.append(img)
48
49      # Dimensões das imagens
50      dims = imagens[0].shape
51
52      # Passa todas as imagens para arrays de uma dimensão
53      fotos = np.zeros((qtd_imgs, dims[0] * dims[1] * dims[2]), dtype=np.float32)
54      for i in range(qtd_imgs):
55          fotos[i, :] = imagens[i].flatten()
56
57      # Calcula a média e os autovetores
58      #qtd_autov = min(200, qtd_imgs)
59      media, autovetores = cv2.PCACompute(
60          fotos, mean=None)
61          #fotos, mean=None, maxComponents=qtd_autov)
62
63      # Face média é o vetor médio em forma matricial
64      face_media = media.reshape(dims)
65      cv2.imwrite("face_media.png", face_media * 255)
66
67      # Eigenfaces são os autovetores em forma matricial
68      eigenfaces = []
69      for i, autovetor in enumerate(autovetores):
70          eigenface = autovetor.reshape(dims)
71          eigenfaces.append(eigenface)
72          cv2.imwrite(f"eigenface_{i:03}.png", normaliza(eigenface))
73
74      # Reconstrói uma faces a partir das eigenfaces
75      foto = np.float32(cv2.imread(ARGS.reconstruir)) / 255.0
76      face_norm = foto.flatten() - media
77      face_reco = face_media
78      coeficientes = []
79      for i, autovetor in enumerate(autovetores):
80          coef = np.dot(face_norm, autovetor)
81          coeficientes.append(coef.item())
82          face_reco = np.add(face_reco, eigenfaces[i] * coef)
83          cv2.imwrite(f"reconstruida_{i:03}.png", face_reco * 255)
84
85      print ("Coeficientes:")
```

```
86      for i, coef in enumerate(coeficientes):
87          print(f"{i:03}: {coef}")
88
89      return 0
90
91
92  if __name__ == "__main__":
93      PARSER = argparse.ArgumentParser(description='Ilustra PCA e Eigenfaces')
94      PARSER.add_argument('-i', '--imagens', nargs='*',
95                          help='Caminhos das imagens de entrada')
96      PARSER.add_argument('-r', '--reconstruir',
97                          help='Caminhos da imagem a ser reconstruida')
98      PARSER.add_argument('--version', action='version',
99                          version='%(prog)s v' + __version__)
100     ARGS = PARSER.parse_args()
101
102     logging.basicConfig(
103         filename=os.path.join(SCRIPT_DIR, SCRIPT_NAME + '.log'),
104         format='%(asctime)s %(levelname)-8s %(message)s',
105         level=logging.INFO,
106         datefmt='%Y-%m-%d %H:%M:%S')
107
108     logging.debug("==================Iniciou o script==================")
109
110     try:
111         sys.exit(main())
112     except KeyboardInterrupt:
113         logging.debug("====================Interrompido====================")
114         sys.exit(0)
115     except SystemExit:
116         logging.debug("=================Finalizou o script=================")
117     except BaseException:
118         logging.exception('Ocorreu um erro')
```

APÊNDICE E – Código do reconhecedor de faces por Eigenfaces usando OpenCV

Código 5 – Reconhecimento facial por Eigenfaces usando OpenCV

```
#!/usr/bin/env python3
# -*-*- encoding: utf-8 -*-*-
# Created: Sun, 17 Jun 2018 16:19:23 -0300

"""
Reconhece faces usando Eigenfaces

uso: eigenfaces.py [-c CLASSIFICADOR]
                   [-t DIR_TREINO]
                   [-i [IMAGENS [IMAGENS ...]]]
                   [-o DIR_SAIDA]
"""


import os
import sys
import argparse
import logging
import numpy as np
import cv2


__author__ = "Julio Batista Silva"
__copyright__ = "Copyright (c) 2018, Julio Batista Silva"
__license__ = "GPL v3"
__version__ = "1.0"
__email__ = "julio@juliobs.com"


SCRIPT_DIR = os.path.dirname(__file__)
SCRIPT_NAME = os.path.basename(__file__)
DEFAULT_CLASSIFIER = 'haarcascades/haarcascade_frontalface_default.xml'
DEFAULT_IMGS_PATH = '/Users/julio/Pictures/Face_databases/ATT'
VERDE = (0, 255, 0)
VERMELHO = (0, 0, 255)


def le_imagens_de_treino(path, tam=None):
    '''Lê imagens do diretório path, redimensiona para tam e retorna uma tupla
```

```
40      com três listas: imagens, ids e nomes'''
41
42      imagens, ids, nomes = [], [], []
43      qtd = 0
44
45      pessoas = [x for x in os.listdir(path)
46                 if os.path.isdir(os.path.join(path, x))]
47
48      for pessoa in pessoas:
49          for foto in [os.path.join(path, pessoa, x) for x in
50                       os.listdir(os.path.join(path, pessoa))]:
51              img = cv2.imread(foto, cv2.IMREAD_GRAYSCALE)
52
53              # Redimensiona imagem
54              if tam is not None:
55                  img = cv2.resize(img, tam)
56
57              imagens.append(np.asarray(img, dtype=np.uint8))
58              ids.append(qtd)
59          nomes.append(pessoa)
60          qtd += 1
61
62      return (imagens, ids, nomes)
63
64
65  def detecta_faces(imagem, classificador):
66      ''' Retorna uma lista com as coordenadas das faces detectadas na imagem
67      pelo classificador passado por parâmetro'''
68      # Carrega o classificador em cascata passado por parâmetro
69      classif = cv2.CascadeClassifier(classificador)
70
71      # Detecta as faces na imagem
72      faces = classif.detectMultiScale(
73          imagem, scaleFactor=1.2, minNeighbors=6, minSize=(30, 30))
74
75      return faces
76
77
78  def main():
79
80      # Obtém imagens de treino com ids e nomes
81      X, y, nomes = le_imagens_de_treino(ARGS.dir_treino, (200, 200))
82
83      # Treina o reconhecedor de faces
84      reconhecedor = cv2.face.EigenFaceRecognizer_create()
85      reconhecedor.train(np.asarray(X), np.asarray(y))
86
87      # Para estatística
88      acertos = 0
```

```
89      erros = 0
90      falhas = 0
91
92      for img_path in ARGS.imagens:
93          imagem = cv2.imread(img_path)
94
95          if imagem is None:
96              logging.debug(f"Não conseguiu abrir {img_path}.")
97              falhas += 1
98              continue
99
100         # Cria cópia em tons de cinza da imagem
101         cinza = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)
102
103         faces = detecta_faces(cinza, ARGS.classificador)
104         logging.debug(f"{len(faces)} faces detectadas.")
105
106         # Gabarito
107         nome_certo = os.path.split(os.path.split(img_path)[0])[1]
108
109         if len(faces) > 0:
110             for (x, y, w, h) in faces:
111                 # Corta e redimensiona face
112                 face = cinza[x:x + w, y:y + h]
113                 face = cv2.resize(
114                     face, (200, 200), interpolation=cv2.INTER_LINEAR)
115
116                 # Tenta reconhecer a face. Se conseguir, escreve nome na imagem
117                 try:
118                     # Confiança aceitável: abaixo de 5000
119                     num, confianca = reconhecedor.predict(face)
120                     nome = nomes[num]
121                     logging.debug(f"Reconhecido: {nome}, "
122                                   f"Confiança: {confianca:.2f}, "
123                                   f"Correto: {nome_certo}")
124
125                     # Desenha um retângulo em volta da face detectada
126                     cv2.rectangle(
127                         imagem, (x, y), (x + w, y + h), (255, 0, 0), 2)
128
129                     # Escreve nome da pessoa reconhecida.
130                     # Verde: correto, Vermelho: errado
131                     if nome == nome_certo:
132                         acertos += 1
133                         cv2.putText(imagem, nome, (x, y + 20),
134                                     cv2.FONT_HERSHEY_SIMPLEX, 0.6, VERDE, 2)
135                     else:
136                         cv2.putText(imagem, nome, (x, y + 20),
137                                     cv2.FONT_HERSHEY_SIMPLEX, 0.6, VERMELHO, 2)
```

```
138                         erros += 1
139
140                 except BaseException:
141                     logging.exception("Não reconheceu")
142                     falhas += 1
143
144                 # Grava imagem com o resultado da detecção
145                 pasta = ARGS.dir_saida if ARGS.dir_saida else os.path.dirname(
146                     img_path)
147                 nome = f"{nome_certo}-{nome}-"
148                        f"{os.path.splitext(os.path.basename(img_path))[0]}.jpg"
149                 cv2.imwrite(os.path.join(pasta, nome), imagem)
150                 logging.debug(f"Gravou em {os.path.join(pasta, nome)}.")
151
152     logging.info(f"Acertos: {acertos}, Erros: {erros}, Falhas: {falhas}")
153
154     return 0
155
156
157 if __name__ == "__main__":
158     parser = argparse.ArgumentParser(
159         description='Reconhece faces usando Eigenfaces')
160     parser.add_argument('-c', '--classificador',
161                         default=DEFAULT_CLASSIFIER,
162                         help='Caminho para classificador em cascata')
163     parser.add_argument('-t', '--dir_treino',
164                         default=DEFAULT_IMGS_PATH,
165                         help='Caminho da pasta com imagens de treino')
166     parser.add_argument('-i', '--imagens', nargs='*',
167                         help='Caminho para imagens a serem reconhecidas')
168     parser.add_argument('-o', '--dir_saida',
169                         help='Caminho onde imagens com reconhecimento serão salvas')
170     parser.add_argument('--version', action='version',
171                         version='%(prog)s v' + __version__)
172     ARGS = parser.parse_args()
173
174     logging.basicConfig(
175         filename=os.path.join(SCRIPT_DIR, SCRIPT_NAME + '.log'),
176         format='%(asctime)s %(levelname)-8s %(message)s',
177         level=logging.DEBUG,
178         datefmt='%Y-%m-%d %H:%M:%S')
179
180     logging.debug("==================Iniciou o script==================")
181
182     try:
183         sys.exit(main())
184     except KeyboardInterrupt:
185         logging.debug("====================Interrompido====================")
186         sys.exit(0)
```

```
187     except SystemExit:
188         logging.debug("==================Finalizou o script=================")
189     except BaseException:
190         logging.exception('Ocorreu um erro')
```

Anexos

ANEXO A – Código do AdaBoost

Código 6 – AdaBoost. Fonte: (DATTA; DATTA; BANERJEE, 2015)

```
from numpy import *

x = array([[0, 1], [1, 1], [2, 1], [3, -1], [4, -1],
       [5, -1], [6, 1], [7, 1], [8, 1], [9, -1]])
p = array([[0, 0.1], [1, 0.1], [2, 0.1], [3, 0.1], [4, 0.1],
       [5, 0.1], [6, 0.1], [7, 0.1], [8, 0.1], [9, 0.1]])
h_final = 0; Thres = zeros((4, 1)); alpha = zeros((4, 1))

for t in range(0, 3):
  err = zeros((9, 1))
  thr = array([0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5])

  for k in range(0, 9):
    if t == 2:
      h = sign(x[:, 0] - thr[k])
    else:
      h = sign(thr[k] - x[:, 0])

    for j in range(0, 10):
      if h[j] != x[j, 1]:
        err[k] = err[k] + p[j, 1]

  for l in range(0, 9):
    if err[l] == err.min():
      indx = l
      break
    Thres[t] = thr[l]

    if t == 2:
      h = sign(x[:, 0] - thr[l])
    else:
      h = sign(thr[l] - x[:, 0])

  alpha[t] = 0.5 * log((1 - err.min()) / err.min())
  q1 = exp(-alpha[t])
  q2 = exp(alpha[t])
  Zt = 2 * sqrt(err.min() * (1 - err.min()))

  for j in range(0, 10):
    if h[j] == x[j, 1]:
      p[j, 1] = (q1 * p[j, 1]) / Zt
    else:
```

```
43        p[j, 1] = (q2 * p[j, 1]) / Zt
44
45    f = alpha[t] * (h)
46    h_final = h_final + f
47
48  decision = sign(h_final)
49  print(decision)
```

Índice