LUIZ FILIPE ZENICOLA BRAGA

SISTEMAS DE RECONHECIMENTO FACIAL

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase em
Eletrônica

ORIENTADOR: Prof. Dr. Ivan Nunes da Silva

São Carlos
2013

Braga, Luiz Filipe Zenicola
B813s Sistemas de Reconhecimento Facial / Luiz Filipe Zenicola Braga; orientador Ivan Nunes da Silva. São Carlos, 2013.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2013.

1. Linear Discriminant Analysis. 2. Principal Component Analysis. 3. Independent Component Analysis. 4. Viola-Jones. 5. AdaBoost. 6. Reconhecimento Facial. I. Título.

FOLHA DE APROVAÇÃO

Nome: Luiz Filipe Zenicola Braga

Título: "Sistemas de Reconhecimento Facial"

Trabalho de Conclusão de Curso defendido e aprovado em 11 / 06 / 2013,

com NOTA 9,0 (Nove, Zero), *pela Comissão Julgadora:*

Prof. Dr. Ivan Nunes da Silva (orientador)
SEL/EESC/USP

M.Sc. Oureste Elias Batista - (Doutorando - SEL/EESC/USP)

M.Sc. Silas Franco dos Reis Alves - (Doutorando - SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

Dedico este trabalho aos meus pais, à minha irmã e à minha namorada.

Agradecimentos

Agradeço a Deus por ter me permitido chegar onde estou. Agradeço à minha mãe por sempre estar por perto me apoiando, e ao meu pai, que mesmo morando longe, sempre se fez presente. Agradeço à minha irmã pelo companheirismo. Agradeço à minha namorada por sempre me motivar e me apoiar.

Agradeço à Universidade de São Paulo por ter me dado o privilégio de estudar em uma excelente instituição de ensino, oferecendo sempre professores capacitados e laboratórios de qualidade. Agradeço em especial ao Prof. Dr. Ivan Nunes da Silva por ter sido meu orientador neste projeto e me permitir escolher um tema que fosse de meu interesse.

Sumário

Lista de Figuras

Lista de Siglas

ICA	*Independent Component Analysis*
FERET	*Face Recognition Technology*
GB	*Gigabyte*
LDA	*Linear discriminant analysis*
PCA	*Principal Component Analysis*
RAM	*Random Access Memory*

Resumo

Este trabalho apresenta o estudo, desenvolvimento e avaliação de um sistema de reconhecimento facial automatizado, que recebe como entrada uma imagem e como saída retorna uma resposta positiva ou negativa no caso de haver ou não, na imagem, faces previamente cadastradas em um banco de dados. O projeto do sistema consiste em três partes: detecção facial, extração de características e reconhecimento. A detecção facial foi implementada usando-se o algoritmo de Viola-Jones, que tem como base o treinamento de classificadores fortes agregando-se diversos classificadores fracos. Para a extração de características, foi utilizada a análise discriminante, que reduz a dimensionalidade do espaço enquanto preserva o máximo possível de informação discriminatória, e para o reconhecimento foi utilizada uma métrica baseada em distância entre projeções.

Palavras-chave: Linear Discriminant Analysis, Principal Component Analysis, Independent Component Analysis, Viola-Jones, AdaBoost, Reconhecimento Facial.

Abstract

This work presents the study, development and evaluation of an automatic facial recognition system, that receives an image as input and returns a positive or negative response as output, depending on whether or not there are known faces, previously added to the database, on the image.

The system design consists of three parts: face detection, feature extraction and recognition. Face detection has been implemented using the Viola-Jones algorithm, which is based on training strong classifiers by aggregating various weak classifiers. Feature extraction has been conducted by using discriminant analysis, which reduces the dimensionality of space while preserving the discriminatory information as much as possible. In conclusion, recognition was performed by applying a metric based on distance between projections.

Keywords: Linear Discriminant Analysis, Principal Component Analysis, Independent Component Analysis, Viola-Jones, AdaBoost, Facial Recognition.

1 Introdução

Os primeiros trabalhos sobre reconhecimento facial no campo da engenharia começaram por volta de 1960 [1], mas somente na década seguinte que foram propostos sistemas automatizados de reconhecimento facial [2]. Desde então, várias linhas de pesquisa sobre o assunto iniciaram-se em diversos campos da ciência, permitindo assim, que novas tecnologias incorporassem o processo e o tornassem cada vez mais robusto e eficaz. No entanto, mesmo com mais de 50 anos de pesquisas, ainda existem barreiras científicas que dificultam o reconhecimento facial automatizado em certas condições. Por exemplo, dependendo do tipo de sistema utilizado, a taxa de erro aumenta significativamente quando a iluminação ambiente não é muito favorável, ou quando a face da pessoa a ser reconhecida é captada de perfil e não frontalmente, ou ainda, se as expressões faciais não forem neutras.

1.1 Motivação

Ao longo dos anos, diversos algoritmos de reconhecimento facial foram desenvolvidos, mas ainda não há nenhum que reconheça faces com a mesma eficiência que o ser humano reconhece, isto é, que reconheça faces em qualquer ambiente, vistas de qualquer ângulo, e não importando a expressão facial. Porém, em condições controladas, existem sistemas que superam o desempenho humano, sendo até mesmo capazes de diferenciar gêmeos monozigóticos [3].

A vantagem do reconhecimento facial sobre outros métodos biométricos é que o reconhecimento não é intrusivo e pode ser feito sem que o individuo interaja de forma direta com o sistema. Por exemplo, é possível captar a imagem de um rosto através de uma câmera de segurança e permitir que autoridades identifiquem, de forma automatizada, possíveis suspeitos sem que eles saibam que estão sendo identificados. O *FBI* (*Federal Bureau of Investigation*) utiliza sistema de reconhecimento facial há pelo menos três anos e já obteve êxito [4].

1.2 Objetivos

O objetivo deste trabalho é desenvolver e avaliar um sistema de reconhecimento facial automatizado utilizando métodos de detecção facial e extração de características.

1.3 Organização do trabalho

Este trabalho está organizado da seguinte forma:

1. Introdução: Apresenta um histórico sobre sistemas de reconhecimento facial, bem como suas aplicações, vantagens e dificuldades tecnológicas envolvidas no processo. Também são apresentados os objetivos e organização deste trabalho.
2. Revisão Bibliográfica: Nesta parte, o sistema de reconhecimento facial é abordado de forma teórica e são apresentados diferentes métodos e conceitos envolvendo cada etapa do processo e as vantagens e desvantagens de cada um deles.
3. Implementação do Sistema: Descreve todos os passos necessários para implementar um sistema de reconhecimento facial totalmente automatizado que vai desde a detecção até o reconhecimento facial.
4. Resultados: Os resultados do sistema implementado no capítulo 3 são apresentados e discutidos.
5. Conclusões: São apresentadas as conclusões sobre o projeto.

2 Revisão Bibliográfica

Este capítulo tem como objetivo descrever as etapas do processo de reconhecimento facial, que podem ser divididas nos seguintes grupos: (1) Detecção de faces, (2) Extração de características e Representação da Face, (3) Reconhecimento e Verificação. Um sistema de reconhecimento facial totalmente automatizado é capaz de receber como entrada uma imagem ou vídeo, identificar as faces presentes, caracterizá-las matematicamente, compará-las com outras previamente cadastradas em um banco de dados e caso haja alguma correspondência, informar, como saída, qual face do banco de dados condiz à imagem de entrada. A Figura 2-1 ilustra um típico sistema de reconhecimento facial. Adicionalmente, a imagem de entrada pode ser pré-processada para eliminar ruídos e reduzir a taxa de erro do sistema.

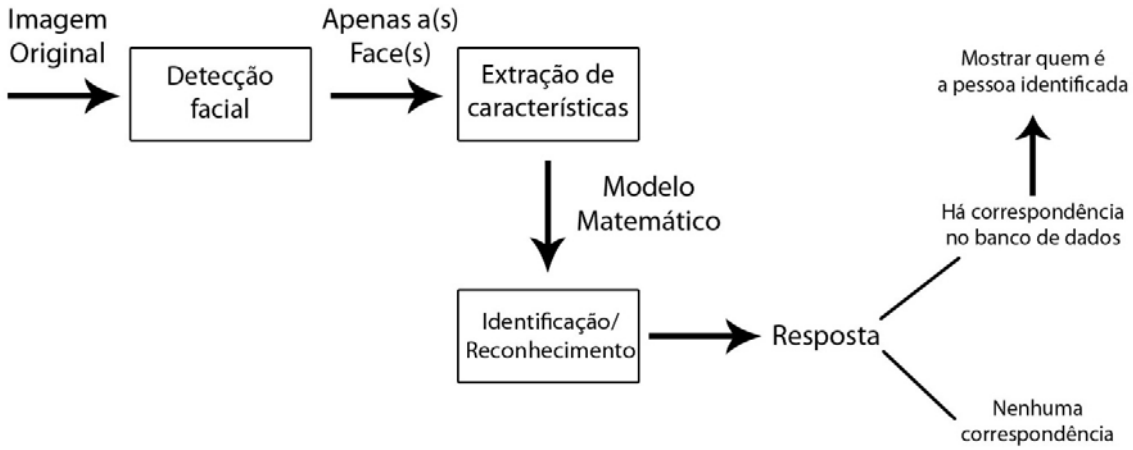

Figura 2-1: Diagrama de um Sistema de Reconhecimento Facial.

2.1 Detecção facial

O primeiro passo em um sistema de reconhecimento facial é a detecção de faces em uma imagem. Essa etapa é importante porque elimina da imagem informações desnecessárias. Se o algoritmo encontra uma ou mais faces, estas são extraídas da imagem original de modo que sejam analisadas separadamente. É importante salientar que caso a entrada do sistema seja uma sequência de vídeo, a informação temporal também é levada em conta e neste caso, além disso, é crítico que o algoritmo seja rápido, uma vez que a detecção é em tempo real.

Muitos dos algoritmos de detecção facial precisam ser treinados exaustivamente antes de obterem um resultado satisfatório. Esses algoritmos necessitam ser treinados tanto com uma série de imagens de várias faces distintas como também com uma série de imagens de objetos que não são faces. Pode-se então tratar o problema de detecção facial utilizando reconhecimento de padrões de duas classes, onde uma classe corresponde à faces e a outra classe à tudo o que não é face. A dificuldade da detecção facial em imagens se dá porque, a princípio, não se sabe de antemão em que região da imagem podem existir faces e em quais escalas elas estão, e também pelo fato de alguns objetos ou a combinação deles se assemelharem a faces quando analisados em baixa resolução. É possível que a detecção de faces e extração de características faciais sejam realizadas simultaneamente, dependendo do tipo de algoritmo usado.

Duas medidas são importantes para avaliar a qualidade do algoritmo: A quantidade de objetos que foram incorretamente identificados como face (falso positivo) e a quantidade de faces que não foram identificadas (falso negativo). Idealmente, o algoritmo teria ambos os valores nulos.

A seguir serão apresentados alguns métodos computacionais para a detecção de faces em imagens.

2.1.1 Algoritmo de Viola-Jones

Paul Viola e *Michael Jones* propuseram em 2001 [5] uma abordagem para detecção de objetos em imagens que se baseia em três conceitos: integral de imagem, treinamento de classificadores usando *boosting* e o uso de classificadores em cascata. Embora o algoritmo possa ser treinado para reconhecer qualquer objeto, a motivação principal da abordagem de *Viola* e *Jones* foi o reconhecimento facial. O ponto forte deste algoritmo é a rapidez com que é executado.

A integral de imagem, também conhecida como tabela de soma de áreas, é um algoritmo, proposto por *Frank Crow* em 1984, que permite avaliar eficientemente a soma dos valores dos pixels (intensidade dos níveis de cinza) de uma área retangular em uma sub-região da imagem. A equação (1) indica como calcular a integral de imagem em uma determinada coordenada:

$$ii(x,y) = \sum_{\substack{x' \le x \\ y' \le y}} i(x',y') \qquad (1)$$

onde $ii(x,y)$ é a integral da imagem nas coordenadas do pixel (x,y) e $i(x,y)$ é a imagem original. Pode-se ver que a integral da imagem na coordenada (x,y) é a soma dos valores dos pixels acima de y e à esquerda de x, inclusive x e y (supondo que a origem do sistema de coordenadas está localizada no canto superior esquerdo da imagem). A tabela de soma pode então ser computada para todos os pixels em uma única varredura como mostra a equação (2):

$$ii(x,y) = i(x,y) + ii(x-1,y) + ii(x,y-1) - ii(x-1,y-1) \quad (2)$$

Define-se $(x,-1) = 0$ e $(-1,y) = 0$ para contornar os casos nos quais as coordenadas dos pixels estão fora dos limites da imagem.

Desse modo, encontra-se facilmente a soma de área em qualquer região retangular da imagem. Portanto, dada uma região retangular ABCD de uma imagem (Figura 2-2), a soma das intensidades dos pixels nessa área pode ser calculada como:

$$\sum_{(x,y)\in ABCD} i(x,y) = ii(A) + ii(D) - ii(B) - ii(C) \quad (3)$$

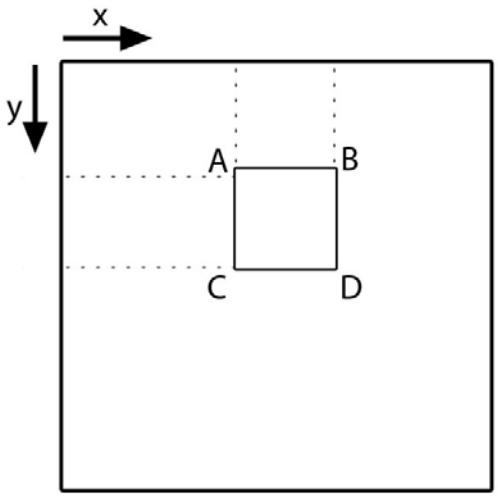

Figura 2-2: Região ABCD em uma matriz de pixels.

A partir da integral de imagem é possível identificar padrões utilizando características *Haar-like*, que são máscaras retangulares nas quais os valores dos pixels de uma região são subtraídos dos valores dos pixels de outra região, representando uma diferença de intensidade luminosa entre áreas da imagem. A Figura 2-3 mostra quatro

possíveis tipos de características-base que podem ser usadas, e que são calculadas subtraindo-se a soma dos valores dos pixels da região branca, da soma dos valores dos pixels da região preta. Para calcular a característica *A* da figura 2-3, são necessárias oito consultas à tabela de soma de áreas (integral de imagem), respectivamente para os oito pontos indicados.

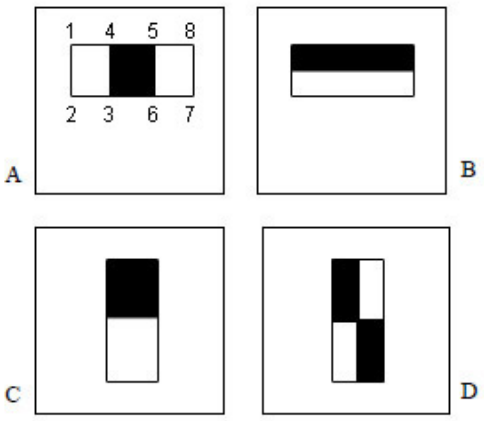

Figura 2-3: Representação visual de quatro tipos de características *Haar-like*.

Cada tipo de característica pode ajudar a reconhecer um determinado padrão, principalmente quando combinados em cascata. Por exemplo, a característica B da figura 2-3 permite identificar uma área na imagem onde há uma diferença de intensidade significativa entre a parte superior e a parte inferior de uma região. Essa característica pode ser aplicada no processo de detecção de faces, uma vez que frequentemente a região dos olhos é mais escura do que a região das bochechas. Obviamente existem outros padrões com o mesmo perfil e que não são faces, daí a necessidade de combinar várias características para refinar a busca. A resolução base da máscara usada no algoritmo é de 24x24 pixels. Existem mais de 100.000 possíveis características *Haar-like* distintas se considerarmos diferentes posições e tamanhos destas dentro da janela.

O segundo passo no algoritmo de Viola-Jones é o treinamento de classificadores. Dado um conjunto de características deve-se treinar o sistema com imagens positivas (faces) e imagens negativas (tudo menos faces). Para isso deve-se usar um algoritmo de treinamento que aprenda funções de classificação. Uma opção é utilizar um algoritmo de aprendizagem que use o método *Boosting*, que consiste em encontrar um classificador de alta precisão combinando-se muitos classificadores “fracos”, onde cada um desses classificadores fracos possui uma precisão média com uma taxa de acertos de pelo

menos 51%. Neste trabalho será abordado o algoritmo de aprendizagem de máquina denominado *AdaBoost*.

AdaBoost (Adaptative Boosting) é um algoritmo que tem como premissa construir um classificador "forte" como uma combinação linear de vários classificadores fracos. A equação (4) expressa essa ideia:

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{4}$$

A função $h_t(x)$ representa classificadores fracos, e pode assumir valores 0 ou 1, respectivamente para exemplos negativos e positivos e x representa uma janela, tipicamente de 24x24. Mais especificamente, um classificador fraco pode ser expresso em função da característica (f), de um *threshold* (θ) e de uma polaridade (p) para indicar a direção da desigualdade, como mostra a equação abaixo:

$$h(x, f, p, \theta) = \begin{cases} 1, & \text{se } pf(x) < p\theta \\ 0, & caso\ contrário \end{cases} \tag{5}$$

O peso de cada classificador fraco é definido por α_t. O classificador forte é então dado pela função $H(x)$:

$$H(x) = \begin{cases} 1, & f(x) \geq \frac{1}{2}\sum \alpha_t \\ 0, & caso\ contrário \end{cases} \tag{6}$$

O algoritmo *AdaBoost* pode ser usado tanto para que este escolha quais características são mais adequadas, como também para treinar classificadores com estas características escolhidas. A Figura 2-4 [5] mostra algumas características selecionadas pelo algoritmo para identificar faces.

Figura 2-4: Características selecionadas pelo algoritmo *AdaBoost* [5].

A última etapa consiste em combinar classificadores fortes em cascata de modo a processar eficientemente regiões da imagem em busca de um padrão. Cada estágio na cascata aplica um classificador mais específico e complexo do que o anterior, de modo que o algoritmo rejeite rapidamente regiões que sejam muito distintas da característica procurada e termine o processo de procura neste caso, evitando que os estágios posteriores sejam executados desnecessariamente. Isso faz com que muitos dos cenários e panos de fundo sejam descartados nos primeiros estágios e apenas faces e outros objetos semelhantes a faces sejam analisados mais exaustivamente. A Figura 2-5 ilustra o processo.

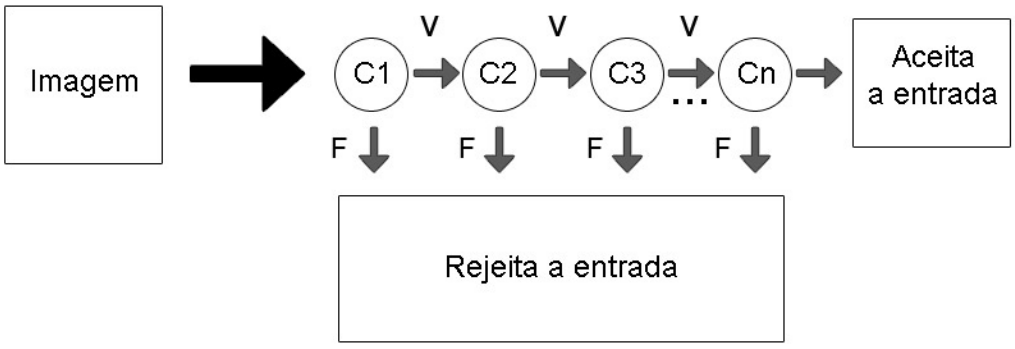

Figura 2-5: Cascata de classificadores.

O algoritmo de Viola-Jones, se bem treinado, tem uma boa precisão, e consegue detectar a maioria das faces que são captadas frontalmente, além de apresentar um baixo número de falsos positivos. No entanto, o algoritmo falha quando tenta detectar faces de perfil, ou quando os olhos estão cobertos (usando-se óculos de sol, por exemplo), ou

ainda, quando a iluminação ambiente é desfavorável. É possível fazer detecções em múltipla escala mudando a escala da imagem ou mudando a escala da janela de detecção. A Figura 2-6 ilustra como as janelas de detecção varrem a imagem em busca dos padrões definidos pelas características do tipo *Haar-like*, lembrando que em que cada região, vários classificadores são aplicados, e que a detecção só é considerada positiva caso a região seja classificada positivamente para todos os classificadores da cascata.

Figura 2-6: Varredura de uma imagem em busca de padrões baseado na diferença entre áreas claras e escuras de uma região.

2.1.2 Outras Abordagens

Existem diversas outras abordagens que tentam solucionar o problema de detecção facial. Algumas dessas soluções utilizam variações das ideias propostas por Viola e Jones. De um modo geral, as abordagens de detecção facial podem ser divididas

em três grupos: Métodos baseados em conhecimento (*Knowledge-Based*), Métodos baseados em modelos (*Template-Based*) e Métodos baseados em Aparência (*Appearance-Based*). Existem também métodos baseados em características invariantes (*Feature invariant*), que visam encontrar estruturas na face humana que são invariantes à iluminação, ângulo e pose.

Métodos *Knowledge-Based* procuram descrever os padrões da face usando regras baseadas no conhecimento humano, como por exemplo, que uma face humana típica tem dois olhos, um nariz e uma boca. A partir dessas regras, são estabelecidas relações entre as características que as definem, como posições e distâncias relativas e contrastes. Os pesquisadores *Yang* e *Huang* apresentaram um trabalho em 1994 que usa este método [6]. Eles propuseram um sistema que consiste em três níveis de regras, no qual no primeiro nível as regras são mais genéricas e no último nível as regras se baseiam em detalhes faciais mais específicos, como é possível observar na Figura 2-7. As regras dos níveis 1 e 2 são aplicadas numa versão de resolução reduzida da imagem, que pode ser obtida aplicando filtros de média. As regras do nível 3 são aplicadas usando-se detectores de borda para identificar contornos da boca e dos olhos (Figura 2-8 [6]). O principal objetivo do nível 3 é eliminar falsos positivos; por exemplo, se existem muitas bordas horizontais na região de uma suposta boca, possivelmente não é uma boca, ou se existem muitas bordas verticais na região de um suposto olho, possivelmente não é um olho. Nos testes experimentais realizados por Yang e Huang, o sistema identificou corretamente 50 de 60 faces (~ 83% de acerto) e, além disso, ainda apresentou falsos positivos em 28 imagens.

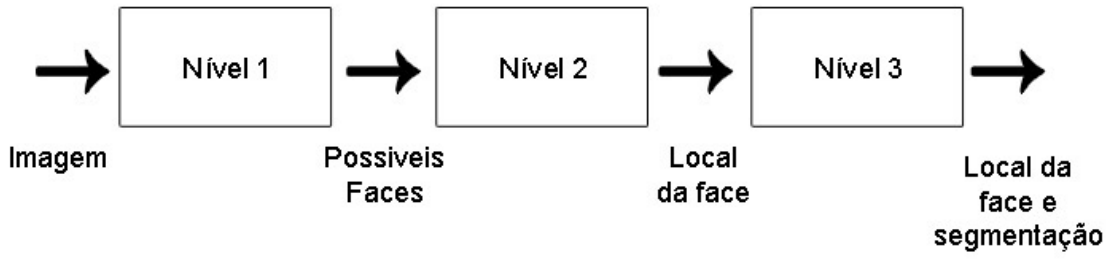

Figura 2-7: Diagrama de bloco de um sistema usando método *Knowledge-Based*.

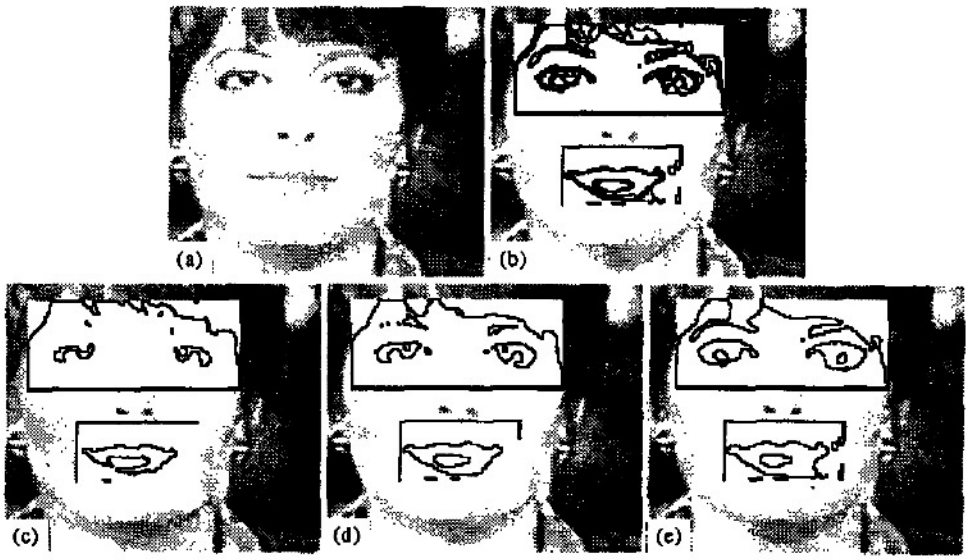

Figura 2-8: Detectores de borda com diferentes *thresholds* usados no nível 3 de um sistema *Knowledge-Based* [6].

A dificuldade dessa abordagem é traduzir o conhecimento humano em regras bem definidas. Se o método opta por regras mais rígidas, pode falhar ao tentar detectar faces em certas circunstâncias, mas se as regras são muito gerais, podem existir muitos falsos positivos.

Métodos *Template-Based* se diferenciam dos *Knowledge-Based* porque não procuram padrões baseados em uma série de regras escolhidas manualmente, mas tentam representar a face parametricamente através de pontos de controle que se deformam a fim de encontrar na imagem um padrão, alinhando o modelo numa possível face. Esses pontos de controle do modelo são escolhidos de modo a representar as principais características da face. A Figura 2-9 [7] mostra o processo iterativo de alinhamento e deformação dos pontos de controle para encontrar uma face na imagem. A desvantagem desses métodos é que geralmente o modelo deve ser inicializado próximo à região onde haja possíveis faces para que a detecção seja bem sucedida.

Por último, métodos *Appearance-Based* têm como base aprender características sobre a face humana utilizando algoritmos de aprendizagem de máquina que são treinados com uma vasta quantidade de imagens tanto de faces quanto de outros objetos. De modo resumido, esses métodos utilizam análises estatísticas a fim de encontrar características discriminantes entre faces e não-faces. As características aprendidas são modelos de distribuição. O algoritmo de Viola-Jones enquadra-se nesta categoria.

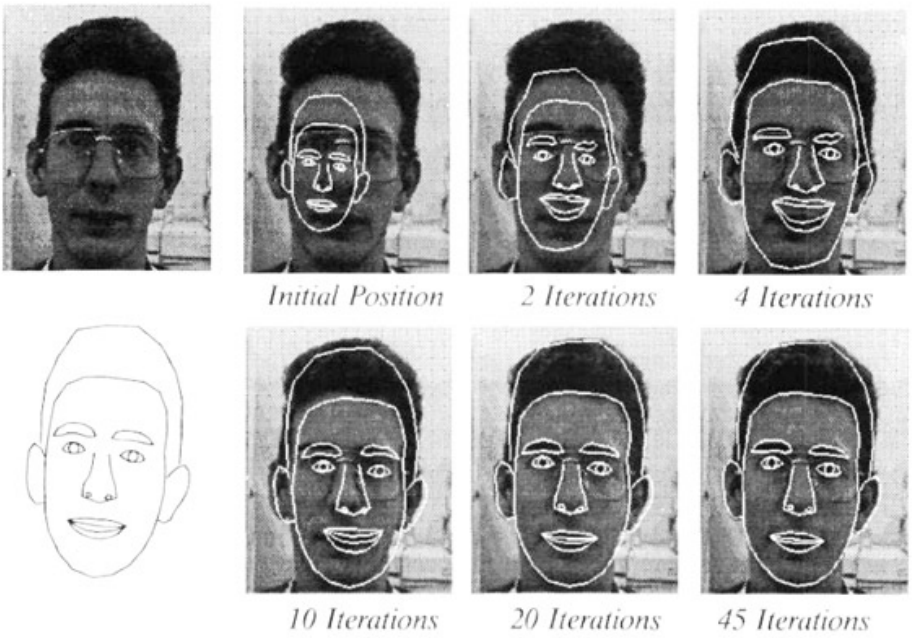

Figura 2-9: *Método Templated-Based* [7].

2.1.3 Considerações finais sobre métodos de detecção facial

A maior dificuldade dos algoritmos de detecção facial é encontrar classificadores, regras ou modelos que consigam identificar faces vistas de qualquer ângulo, com qualquer expressão facial, sob qualquer iluminação. Ainda há muito que se melhorar nos algoritmos de detecção facial. Como *Jain* e *Miller* observaram em [8], “a detecção facial em condições totalmente não-controladas ainda é uma tarefa muito desafiadora”. Porém, como já foi dito anteriormente, os classificadores funcionam muito bem na maioria dos casos em que a face é vista frontalmente ou com uma variação angular razoável, portanto mesmo com as dificuldades apresentadas, ainda é possível obter um bom resultado em determinadas aplicações. Uma prova disso é que já existem diversos softwares e serviços on-line que utilizam detectores faciais como parte de seus produtos e apresentam um bom desempenho. Por exemplo, desde 2008 o *Google Picasa* utiliza um sistema de detecção facial para ajudar seus usuários a marcarem amigos nas fotos de seus álbuns.

2.2 Extração de características

A segunda etapa em um sistema de reconhecimento facial é a extração de características. O objetivo é localizar regiões da imagem que contenham características significativas. Essas regiões podem ser globais ou locais e podem ser distinguidas por texturas, formas, intensidades, propriedades estatísticas e outros. De um modo geral, tenta-se extrair um conjunto compacto de características geométricas interpessoais ou características fotométricas da face. Diferentemente da detecção facial, que categoriza todas as faces numa única classe, na extração de características tenta-se encontrar fatores discriminantes que permitam que cada uma das faces possa ser diferenciada matematicamente das demais. A solução para o problema de extração de características se baseia na redução da dimensionalidade de um espaço. Isso porque as faces em geral compartilham características intrínsecas e possuem regularidades estatísticas, o que faz com que o espaço da face seja menor do que o espaço da imagem de entrada. Assim é possível mapear as características da face em um subespaço usando-se vetores. Os métodos de extração de características abordados neste trabalho necessitam de um conjunto de dados (faces), ou seja, funcionam de modo a representar variações de uma face em relação às demais do banco de dados e não para caracterizar as faces independentemente.

2.2.1 Análise de Componentes Principais (PCA)

A Análise de Componentes Principais (*Principal Component Analysis* ou simplesmente *PCA*) é uma técnica matemática de redução de dimensionalidade baseada em extrair componentes principais de um espaço multidimensional. Esta técnica é usada em reconhecimento de padrões para eliminar redundâncias de informação e ainda assim manter as principais características de um padrão. Há uma grande semelhança entre a Análise de Componentes Principais e a transformada de *Karhunen–Loève* (KLT), que foi originada no contexto de processamento de sinais. A Figura 2-10 ilustra a ideia da análise aplicada a um conjunto de dados em duas dimensões: o eixo Δ_1 corresponde à direção de máxima variância, que é escolhida como a primeira componente principal, e a segunda componente principal é determinada pela ortogonalidade em relação à primeira. Em um espaço de maior dimensão, o processo de análise e seleção de componentes continuaria, observando-se a variância das projeções.

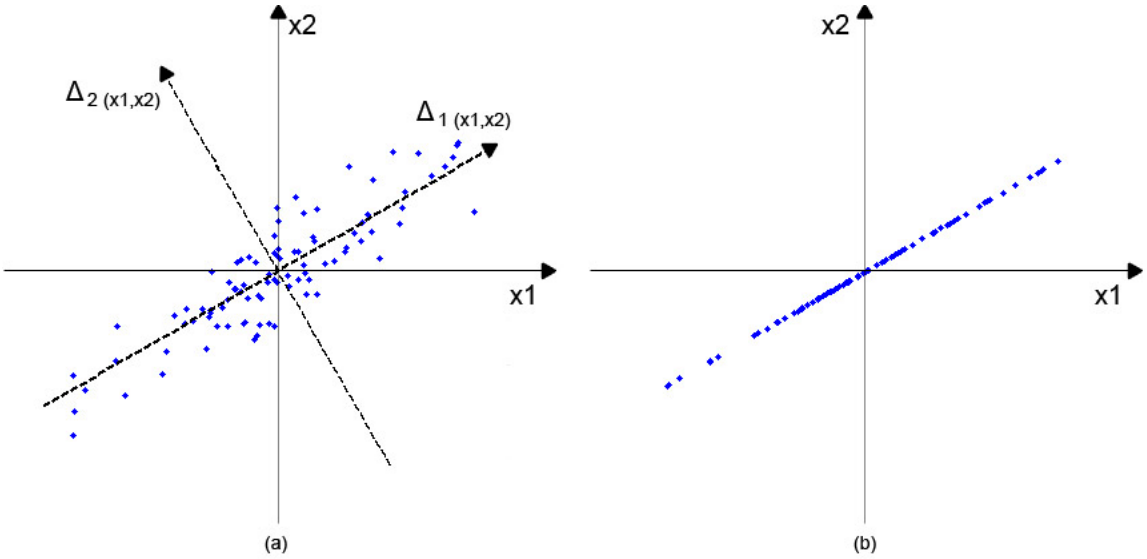

Figura 2-10: (a) Análise de Componentes Principais aplicada a um conjunto de dados, onde as linhas tracejada são as componentes principais; (b) projeção desses dados usando apenas a primeira componente principal.

Uma imagem de *M* linhas por *N* colunas de pixels pode ser considerada como um ponto em um espaço de dimensão $M \times N$, ou também como um vetor de dimensão $M \times N$. Então, uma imagem de 64 x 64 pixels pode ser representada por um vetor de 4096 dimensões ou, equivalentemente, por um ponto em um espaço de 4096 dimensões, onde cada pixel da imagem é uma dimensão e as intensidades dos pixels definem a posição da imagem neste espaço. Uma coletânea de imagens mapeia uma coleção de pontos nesse espaço de alta dimensão. Esses pontos são considerados amostras de uma distribuição de probabilidade. Como as faces humanas, apesar de distintas, possuem muitas características comuns, imagens de faces não são distribuídas de forma totalmente aleatória neste imenso espaço, possibilitando assim que sejam descritas por um subespaço de menor dimensão. A finalidade é encontrar vetores que melhor representam a distribuição de faces no espaço da imagem. Nesta análise, assume-se como sendo gaussiana a distribuição das faces nas imagens, e mesmo que haja um pequeno desvio em relação à suposição, o desempenho do algoritmo não é significativamente afetado.

As componentes principais são obtidas projetando-se os vetores das imagens das faces no espaço gerado pelos autovetores da matriz de covariância do conjunto de imagens. Esses autovetores representam um conjunto de características que, juntas,

descrevem variações entre diferentes imagens de faces, e recebem o nome de *eigenfaces* [9].

Em 1986, *Kirby* e *Sirovich* propuseram uma forma de caracterizar faces humanas usando o método em questão [10], e em 1991, *Turk* e *Pentland* apresentaram uma variação desse método, porém não apenas para caracterizar faces, mas para classificá-las de modo a permitir o reconhecimento automatizado. O algoritmo da Análise de Componentes Principais aplicado ao reconhecimento facial proposto por *Turk* e *Pentland* [9] pode ser formulado da seguinte maneira:

Seja $x_1, x_2, \dots, x_M$ um conjunto de imagens normalizadas de faces de diferentes pessoas. Cada imagem é transformada em um vetor de dimensão N, onde N é a quantidade de pixels da imagem. Então a média do conjunto (face média) é definida por $\overline{x} = \frac{1}{M}\sum_{n=1}^{M} x_n$. A figura 2-11 mostra uma face média calculada com 20 faces distintas. A diferença de cada imagem em relação à média e dada pelo vetor $\Delta_i = x_i - \overline{x}$. Esse conjunto de vetores é sujeito à análise de componentes principais, que procura um conjunto de *M* vetores ortonormais, u_n, que melhor descrevem o conjunto de dados. O *k-ésimo* vetor, u_k, é escolhido de modo que $\lambda_k = \frac{1}{M}\sum_{n=1}^{M}\left(u_k^T \Delta_n\right)^2$ seja máximo, sujeito a

$$u_l^T u_k = \delta_{lk} = \begin{cases} 1 & \text{se } l = k \\ 0 & caso\ contrário \end{cases} \tag{7}$$

Os vetores u_k e os escalares λ_k são respectivamente os autovetores e os autovalores da matriz de covariância

$$C = \frac{1}{M}\sum_{n=1}^{M} \Delta_n \Delta_n^T = AA^T \tag{8}$$

onde a matriz $A = [\Delta_1 \quad \Delta_2 \quad \dots \quad \Delta_M]$. A matriz C é de ordem N^2 por N^2, então, calcular N^2 autovetores e autovalores requer um esforço computacional elevado. Por exemplo, para uma imagem de 256x256 pixels seriam mais de 65.000 autovetores e autovalores. Mas se a quantidade de imagens de faces do conjunto for menor do que a dimensão do espaço da imagem ($M < N^2$), existirão apenas $M - 1$ autovetores relevantes (ao invés de N^2). Os demais autovetores serão associados a autovalores nulos. Seguindo essa análise, constrói-se a matriz $L = A^T A$, de ordem M por M, onde $L_{mn} = \Delta_m^T \Delta_n$ e calculam-se os M autovetores, v_i, da matriz L. Com esses vetores determina-se uma combinação linear das M imagens de faces do banco de dados, formando os *eigenfaces*, u_l.

$$u_l = \sum_{k=1}^{M} v_{lk}\,\Delta_k, \qquad l = 1, \ldots, M \tag{9}$$

Os autovalores permitem que os autovetores sejam ranqueados de acordo com sua importância em caracterizar variações entre as faces. A figura 2-12 mostra os *eigenfaces* calculados usando-se as mesmas 20 faces da figura 2-11.

Figura 2-11: Imagens de 20 faces (*The Database of Faces - AT&T Laboratories Cambridge*) e a face média obtida com esse conjunto de faces.

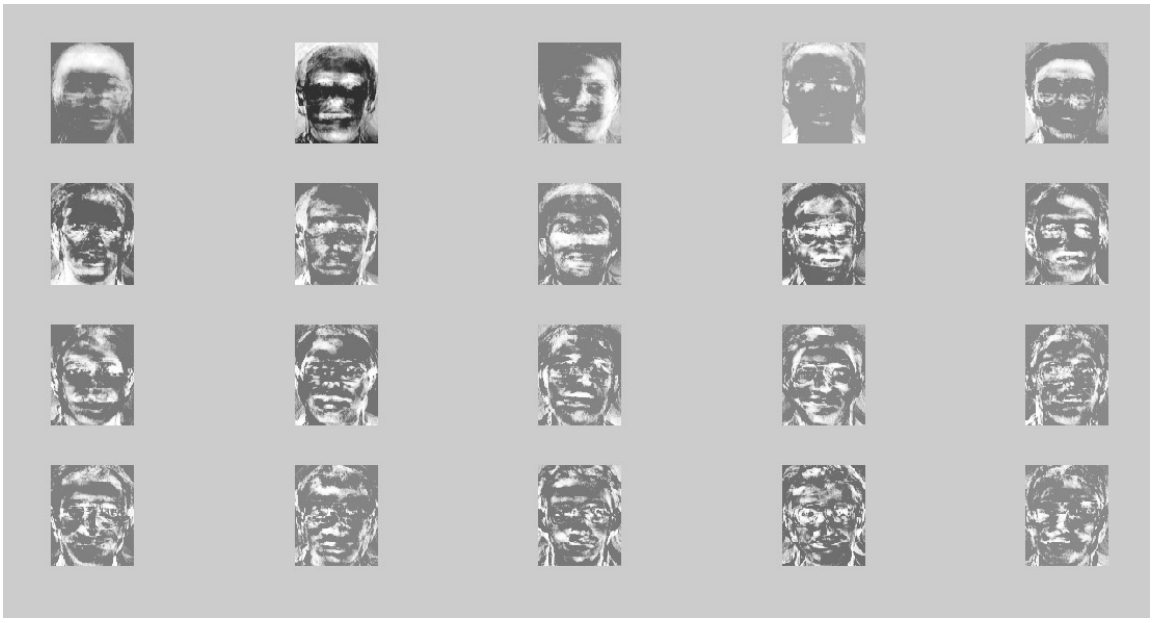

Figura 2-12: 20 *Eigenfaces* calculadas a partir das faces da figura 2-11.

Esse método é fácil de ser implementado e diminui consideravelmente os esforços computacionais porque reduz a ordem da matriz de interesse, possibilitando melhor eficiência e menor espaço de armazenamento. No entanto, uma das desvantagens é que cada vez que uma nova face é adicionada ao banco de dados, é necessário recalcular todos os autovetores novamente. Outro problema dessa abordagem é que a análise em si não garante um bom poder discriminatório, uma vez que as componentes são encontradas com base na direção de maior variância do conjunto e não de maior discriminabilidade. Por isso, pode-se dizer que a Análise de Componentes Principais caracteriza bem a face, mas não necessariamente funciona como um bom classificador porque nem sempre as dimensões de maior variância são as mais relevantes para discriminar diferentes identidades [22].

2.2.2 Análise Discriminante Linear (LDA)

A análise discriminante linear (*Linear Discriminant Analysis* ou *LDA*) é um método estatístico que visa reduzir a dimensionalidade do espaço enquanto preserva o máximo possível de informação discriminatória. Cada componente facial possui certo poder discriminatório que permite que sejam inferidas características como idade, etnia e sexo, e que possibilita, também, que uma pessoa seja distinguida de outra. Diferentemente do método *PCA* que seleciona as características que melhor representam a face, o *LDA* seleciona o subespaço que melhor discrimina classes de faces, o que faz com que o método não seja muito sensível à variação de iluminação das imagens, mas seja mais sensível à variação de identidade das pessoas. Essa técnica procura direções na qual as classes são melhores separadas, como mostra a Figura 2-13. Embora não seja um requisito, é recomendado que o banco de dados de faces possua várias imagens de cada uma das pessoas (por exemplo, com diferentes expressões faciais, diferentes configurações da iluminação ambiente). Desse modo é possível definir melhor as classes (identidades) e obter um melhor resultado comparado ao *PCA*.

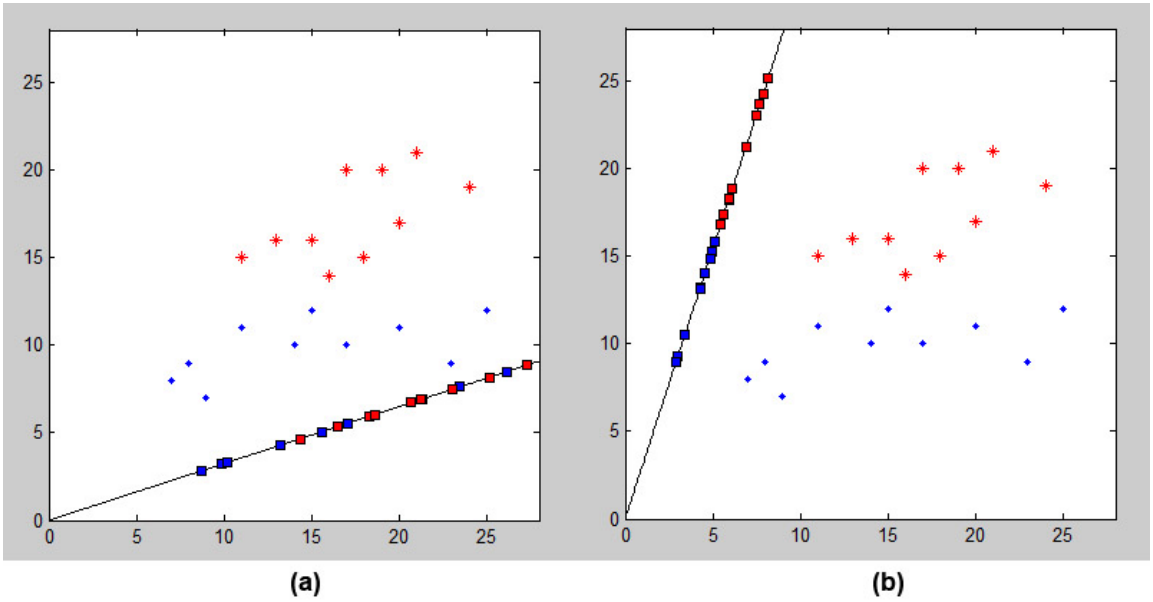

Figura 2-13: (a) Pontos misturados quando projetados numa reta; (b) pontos separados quando projetados em outra reta, indicando um bom classificador.

Dois conceitos são importantes no *LDA*: dispersão dentro da própria classe e dispersão entre classes. O primeiro conceito se refere às características intrapessoais e representa variações de aparência na mesma pessoa devido à diferença na iluminação ambiente, à expressões faciais distintas ou à obstrução parcial do rosto. O segundo conceito representa diferença na aparência relacionada à diferença de identidade. A matriz de dispersão S_w, e a matriz de dispersão S_b, se baseiam respectivamente nos dois conceitos apresentados e são expressas por:

$$S_w = \sum_{j=1}^{C} \sum_{i=1}^{N_j} (x_i^j - \mu_j)(x_i^j - \mu_j)^T \tag{10}$$

$$S_b = \sum_{j=1}^{C} (\mu_j - \mu)(\mu_j - \mu)^T \tag{11}$$

onde x_i^j é a i-ésima amostra (imagem) da classe j, μ_j é a média da classe j, C é o número de classes (número de pessoas diferentes no banco de dados), N_j é o número de amostras da classe j e μ é a média de todas as classes. O subespaço criado pelo LDA é formado pelo conjunto de vetores discriminantes $W = [W1 \quad W2 \quad \dots \quad W_n]$ que satisfaça

$$W = \text{argmax} = \left| \frac{W^T S_b W}{W^T S_w W} \right| \tag{12}$$

Quando as imagens das faces forem projetadas nos vetores W, as imagens que forem da mesma pessoa (classe) deverão ser distribuídas perto uma das outras, e esse *cluster* por sua vez, deverá ser projetado o mais distante possível do *cluster,* obtido ao projetarem-se imagens de outras pessoas, obtendo assim um bom poder discriminatório, como mostra a Figura 2-14 [11]. Isso é alcançado ao minimizar $W^T S_w W$ e maximizar $W^T S_b W$. Portanto W pode ser construído pelos autovetores de $S_w^{-1} S_b$, que são conhecidos como *fisherfaces* no contexto do reconhecimento facial, devido à relação do *LDA* com o Discriminante Linear de Fisher. Como a matriz S_w, na maioria das vezes, não é invertível, é necessário solucionar o problema usando algum artifício matemático. Esse problema acontece porque geralmente o número de imagens é muito menor do que a dimensionalidade do espaço dos dados, expressa pela quantidade de pixels da imagem, criando assim matrizes degeneradas. Duas possíveis soluções são: usar o método da pseudo-inversa, ou aplicar a análise em um subespaço ao invés de aplicá-la no espaço completo. Por exemplo, pode-se primeiro reduzir a dimensionalidade usando-se o *PCA* e então, no subespaço obtido, aplica-se o *LDA* para encontrar as direções mais discriminantes. É importante notar que neste caso o *PCA* é aplicado apenas para contornar o problema da singularidade da matriz, removendo o espaço-nulo. Porém, algumas pesquisas [12], [13] mostraram que o espaço nulo da matriz S_w, eliminado pelo *PCA*, pode conter informação discriminante importante se a projeção da matriz S_b não for zero nessa direção, mas o espaço-nulo de S_b pode ser descartado sem prejuízo. Uma eficiente alternativa é usar métodos baseados no *LDA-direto* (*D-LDA*), que tem como premissa solucionar o problema da matriz singular sem usar o *PCA* para reduzir o espaço. Em suma, os métodos *D-LDA* se baseiam em descartar o espaço-nulo da matriz S_b através de diagonalização, depois projetar e fatorar a matriz S_w no subespaço de S_b e por último, analisar os autovetores e autovalores para obter a solução. A escolha de qual dessas soluções usar depende das condições do sistema, como número de pessoas no banco de dados, quantidade de imagens por pessoa e qualidade das imagens.

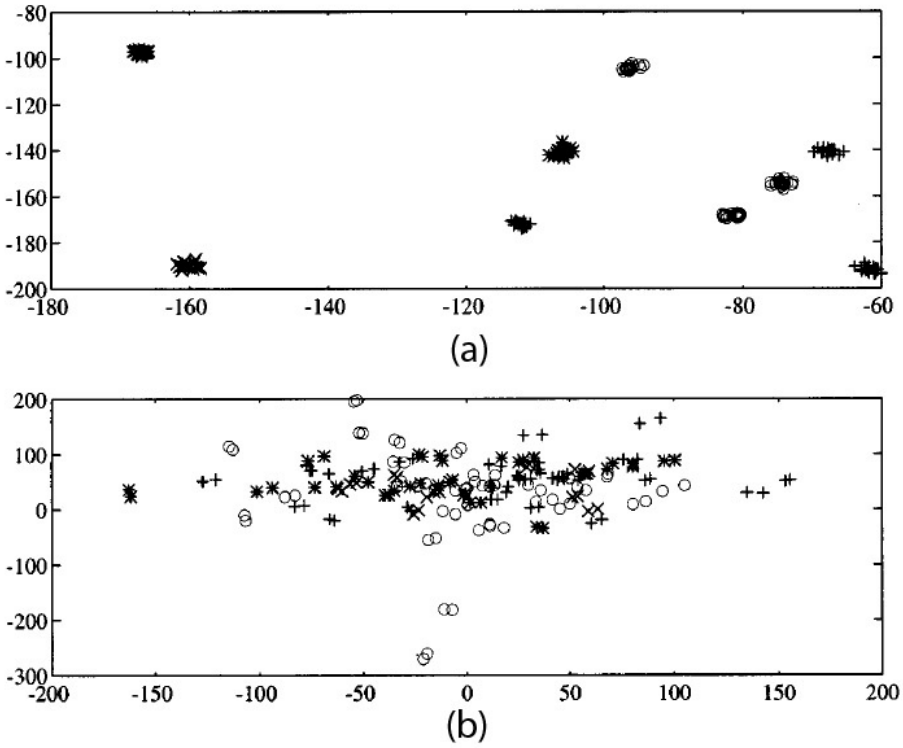

Figura 2-14: (a) LDA aplicado a um conjunto de dados: faces da mesma pessoa são projetadas próximas umas das outras criando *clusters* que facilitam a classificação com base em identidade. (b) PCA aplicado ao mesmo conjunto de dados: faces de pessoas diferentes projetadas muito próximas, dificultando a classificação [11].

2.2.3 Análise de Componentes Independentes (ICA)

A Análise de Componentes Independentes (*Independent Component Analysis* ou *ICA*) é um método de análise estatística que se baseia na "separação cega de sinais" (*blind source separation*) e é uma generalização do *PCA*. O objetivo da análise é separar sinais de diferentes fontes, todos misturados, em componentes independentes. A análise pode ser aplicada em ondas sonoras, imagens digitais, indicadores econômicos e outros tipos de dados. O exemplo clássico para ilustrar o problema da separação de sinais é conhecido como *cocktail party problem*: várias pessoas estão conversando em um salão de festa e, além disso, há música ambiente; pergunta-se então, é possível separar cada uma das vozes e também a música da mistura sonora? Embora o cérebro humano consiga separar as fontes facilmente e identificar as diferentes origens, a tarefa não é tão simples quando se tenta o mesmo no contexto de processamento digital de sinais.

O *cocktail party problem* pode ser solucionado usando-se o *ICA*, desde que o número de microfones no ambiente seja pelo menos igual ao número total das fontes sonoras. Por exemplo, se em uma sala existem quatro pessoas, todas elas falando ao mesmo tempo, e mais uma música ambiente, então, se houver pelo menos cinco microfones colocados em lugares aleatórios da sala, é possível separar os cinco sinais analisando-se a saída dos cinco microfones. A Figura 2-15 ilustra o processo.

O *ICA* diferencia-se do *PCA* porque procura por componentes não-gaussianas que sejam estatisticamente independentes. Enquanto o *PCA* minimiza a covariância do conjunto de amostras (dependência de segunda ordem), o *ICA* minimiza também dependências de ordem mais elevada. A Figura 2-16 mostra uma comparação entre o *ICA* e o *PCA* em duas dimensões. A premissa básica do *ICA* é que as variáveis de interesse são independentes, ou seja, se duas ou mais variáveis são independentes entre si, então o valor de uma não diz nada a respeito do valor das outras e, portanto, não há qualquer correlação entre elas. Em situações práticas, é inviável encontrar uma representação na qual as variáveis são realmente independentes, mas é possível encontrar componentes que são o mais independente possível entre si.

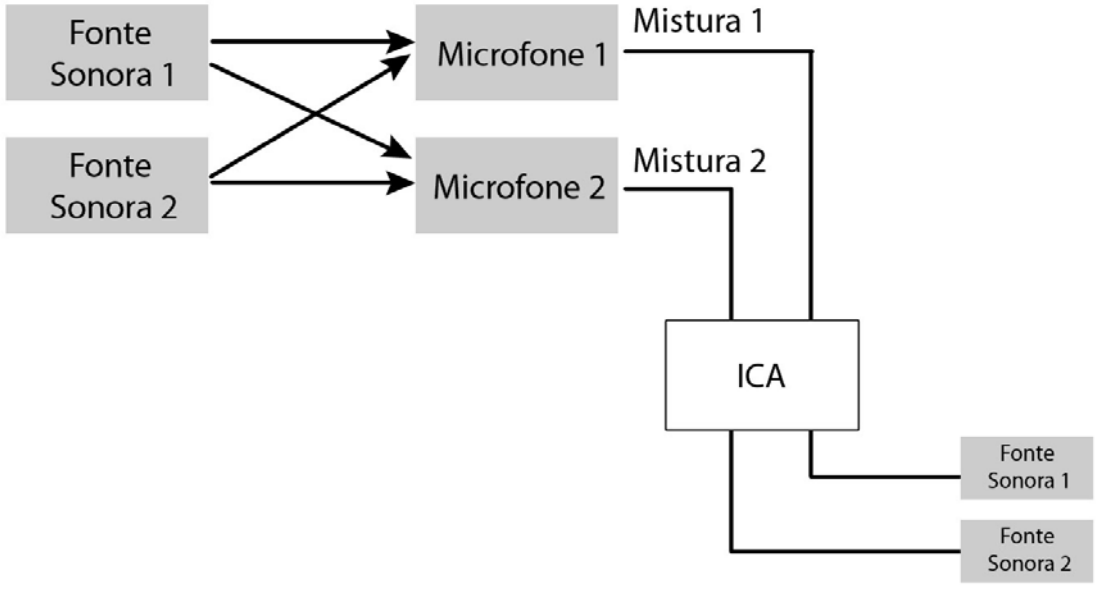

Figura 2-15: Duas fontes sonoras são gravadas simultaneamente por dois microfones distintos, de modo que a saída de cada microfone seja uma mistura das duas fontes. A partir dessas duas misturas, o *ICA* consegue isolar as duas fontes.

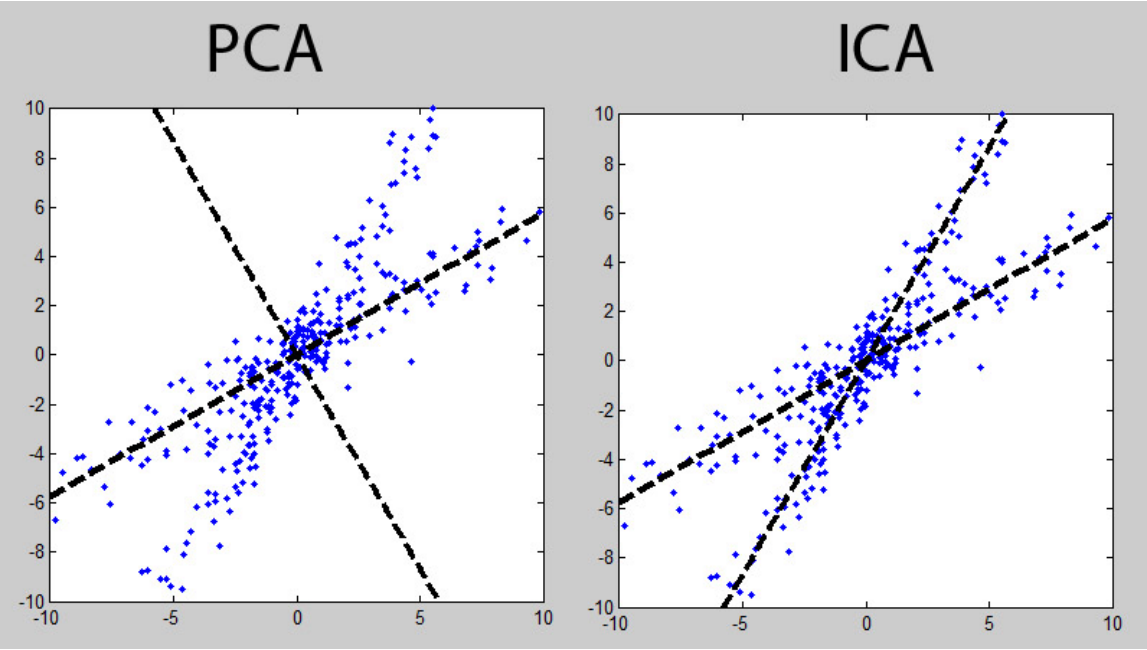

Figura 2-16: Comparação entre o *PCA* e o *ICA*: No PCA as componentes são ortogonais e no ICA são independentes.

O objetivo do *ICA* é decompor um dado sinal em uma combinação linear de sinais desconhecidos e independentes. Então, seja s o vetor de sinais desconhecidos e x o vetor das misturas desses sinais. Se A é a matriz desconhecida, responsável pela mistura dos sinais (matriz de "mixagem"), então $x = As$. Supõe-se que os sinais desconhecidos são independentes e que a matriz A é invertível. Baseando-se nessas suposições, o *ICA* tenta encontrar a matriz A ou a matriz de separação W tal que $u = Wx = WAs$.

No contexto do reconhecimento facial, existem fundamentalmente duas arquiteturas para aplicar o *ICA*, conforme especificado em [14]. Na arquitetura I, as imagens de entrada das faces são consideradas como uma mistura linear de imagens-base estatisticamente independentes combinadas por uma matriz de mixagem desconhecida. O algoritmo estima então os pesos da matriz de separação W (figura 2-17). Nessa arquitetura, as imagens de faces são as variáveis e os valores dos pixels fornecem informações e observações a respeito dessas variáveis. Na arquitetura II, a abordagem é inversa: os pixels são as variáveis e as imagens são observações a cerca dessas variáveis. A separação de componentes, neste caso, é feita nos pixels e não mais no espaço da face como na arquitetura I. De acordo com os testes realizados em [14], a arquitetura I obteve melhores resultados do que a arquitetura II. As bases derivadas na arquitetura I refletem propriedades locais da face enquanto as bases derivadas na

arquitetura II apresentam propriedades mais globais da face, como é possível observar na figura 2-18 [14].

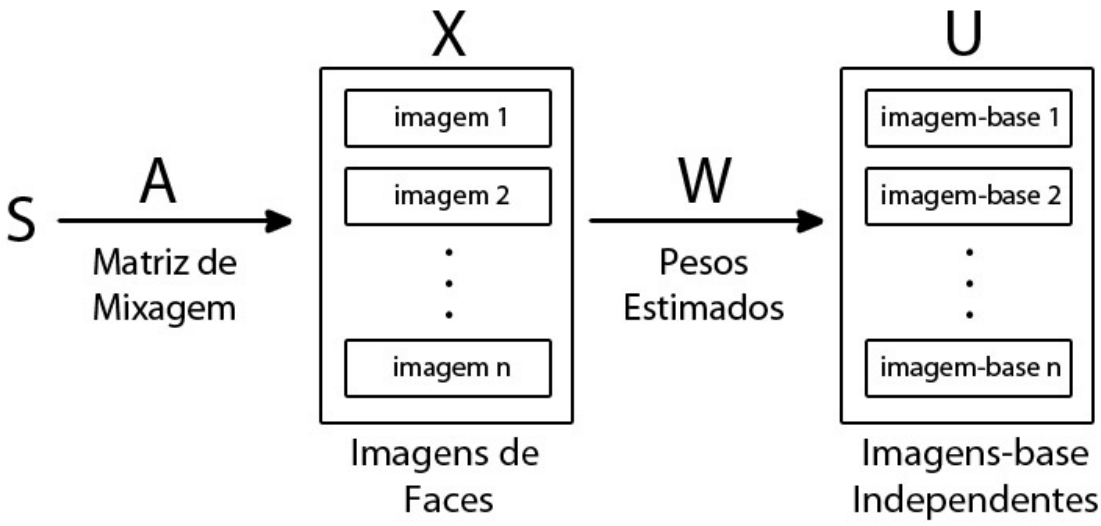

Figura 2-17: *ICA* aplicado para extrair características de faces como imagens-base independentes.

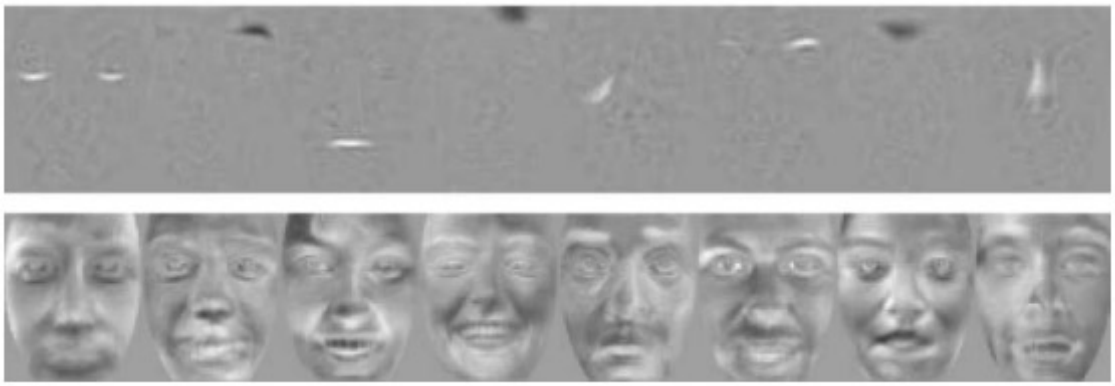

Figura 2-18: Arquitetura I (topo): bases refletem características mais locais; Arquitetura II: bases refletem características mais globais [14].

2.2.4 Outros Métodos

Os três métodos apresentados anteriormente são classificados como holísticos, porque consideram toda a região da face no processo de caracterização. Existem também métodos baseados em estruturas, no qual os olhos, o nariz e a boca são extraídos da imagem e são caracterizados de acordo com suas formas, distâncias relativas e propriedades estatísticas. Um exemplo de um método baseado em estruturas pode ser

encontrado em [15], que usa Modelo Oculto de Markov. Existem também modelos híbridos, como em [16], que combinam métodos holísticos e método baseados em estruturas. Evidentemente, cada método possui vantagens e desvantagens, e pode ser influenciado mais ou menos por fatores externos.

2.3 Reconhecimento

A última parte do sistema consiste em avaliar se as características faciais de entrada, representadas por um modelo matemático, condizem com a de alguma face já previamente cadastrada no banco de dados. É interessante notar que ao contrário do problema de detecção facial, que pode ser considerado como uma classificação de duas classes (faces e não-faces), o de reconhecimento facial é um problema de classificação de muitas classes (uma face contra todas as outras). O reconhecimento pode ser aplicado de dois modos: verificação (autenticação) ou identificação. Na verificação, o sistema valida ou não a hipótese da pessoa ser quem diz; por exemplo, um indivíduo está portando um passaporte e o apresenta como sua identidade. O sistema então valida se aquele indivíduo é o mesmo do passaporte. A verificação é um processo de correspondência de um para um, onde a face de entrada é testada somente contra uma face do banco de dados. Já na identificação, o processo de correspondência é de um para vários, porque compara a face de entrada com todas do banco de dados.

Existem diversas abordagens para o problema de classificação de faces. Muitos desses métodos propõem projetar a imagem de entrada no subespaço de características, criado pelas imagens do banco de dados e então calcular a distância desta imagem em relação às outras. Na extração de características é criada uma base de vetores, que define o espaço da face, e cada face do banco de dados é representada como uma combinação linear desses vetores. A tarefa do reconhecimento é dizer se uma imagem de entrada não pertencente ao banco de dados (imagem-teste) é matematicamente próxima o suficiente de alguma do banco de dados. A dificuldade se dá em estabelecer critérios de rejeição e aceitação para que não haja falso-positivo ou falso-negativo.

O mais importante é escolher uma métrica confiável para estimar a similaridade entre as faces. Os métodos mais simples empregam classificação de distância baseada no vizinho mais próximo, que usam distância, em geral euclidiana, para estimar a proximidade das imagens no subespaço da face. Métodos mais sofisticados se baseiam em densidades de probabilidade, e redes neurais.

3 Implementação do Sistema

O objetivo desta parte do trabalho é implementar um sistema de reconhecimento facial totalmente automatizado. O projeto do sistema completo envolve as três etapas descritas no capítulo anterior: Detecção facial, Extração de Características e Reconhecimento. Neste capítulo serão detalhados os métodos escolhidos e as ferramentas usadas para o desenvolvimento de cada uma das etapas.

3.1 Plataforma de desenvolvimento

A plataforma de desenvolvimento escolhida para elaborar este projeto foi o *MATLAB* (*MATrix LABoratory*). A principal razão que motivou tal escolha foi pelo fato do software em questão oferecer uma vasta biblioteca de funções matemáticas e algoritmos numéricos, além de um *toolbox* próprio para processar imagens (disponível somente em algumas versões). Outro ponto forte é a simplicidade da linguagem, que faz com que seja possível desenvolver programas complexos em um curto espaço de tempo e de forma intuitiva. Além disso, oferece boa portabilidade, sendo compatível com *Windows*, *Linux*, *Mac OS X*, e também com sistemas móveis como *Android* e *IOS*.

3.2 Banco de faces

Para testar e avaliar um sistema de reconhecimento facial é imprescindível que se tenha um banco de dados com imagens de várias pessoas, não apenas sozinhas, mas em grupos e em diferentes configurações de iluminação e pose. É recomendável que cada pessoa cadastrada no sistema, possua pelo menos duas imagens, pois não pode haver interseção entre as imagens-teste (usadas para avaliar a qualidade do sistema) e a imagens usadas para o cadastro. E a imagem-teste deve ser, obviamente, diferente da imagem cadastrada, seja pelo ângulo da câmera em relação à face, ou pela iluminação ambiente, ou expressão facial, etc. Dependendo do tipo de extrator de características usado, pode ser desejado mais de duas imagens diferentes por pessoa para o cadastro no banco de dados, além da imagem-teste.

Neste trabalho foram usados bancos com imagens de grupo de pessoas (para testar o detector de faces) e bancos com uma face por imagem. Os seguintes bancos de imagens de faces foram usados:

- *The Database of Faces* (*AT&T Laboratories Cambridge*): Criado na década de 90, contém imagens de faces de 40 pessoas. Cada pessoa foi fotografada 10 vezes, em diferentes ângulos e com diferentes expressões faciais. Cada imagem apresenta 256 níveis de cinza por pixel e possuem dimensões de 92x112 pixels.
- *The Color FERET Database* [17] [18]: Banco de faces criado pelos pesquisadores *Harry Wechsler* e *P. Jonathon Phillips* na década de 90, patrocinado por *DOD Counterdrug Technology Development Program Office*. Possui mais de 14.000 imagens coloridas de 1199 pessoas vistas de diferentes ângulos e com diferentes configurações de iluminação ambiente.
- *Faces in the Wild* [19]: Compilação com mais de 30.000 imagens de faces coletadas de artigos jornalísticos.

3.3 Detecção facial

Este módulo do sistema recebe como entrada uma imagem que pode ou não conter faces humanas, e como saída retorna imagens somente das faces, caso estas existam. O algoritmo escolhido foi o de Viola-Jones por sua eficiência e baixa taxa de erros. O maior desafio deste método é treinar o sistema exaustivamente com uma vasta compilação de exemplos positivos (faces) e exemplos negativos (não-faces). É importante que essa coleção de imagens de faces abranja pessoas de diferentes idades, etnias, com barba, sem barba, com diferentes estilos de cabelo e diferentes expressões faciais.

A implementação desta parte do projeto foi divida em três submódulos: cálculo das características *Haar-like*, treinamento dos classificadores usando *AdaBoost* e Varredura da imagem de entrada em busca dos padrões treinados. Os códigos desenvolvidos nesta parte projeto podem ser encontrados no Apêndice A.

3.3.1 Características *Haar-like*

Cada característica (ou padrão) representa uma diferença de intensidades entre áreas da imagem. O número total de características possíveis para uma dada janela depende de quantas bases são usadas. Foram escolhidas três bases (figura 3-1) que, de acordo com relatórios de pesquisa, obtiveram um bom desempenho, e a janela usada foi de 24x24 pixels (dimensão das imagens de treinamento). Portanto existem 114.000 diferentes características nessa janela considerando-se as três bases escolhidas. Isso porque as características podem sofrer alterações de escala e posição dentro da janela. É importante notar a diferença entre a característica em si e seu respectivo valor quando

computado em uma região da imagem: A característica é apenas uma máscara que pode sofrer alteração de posição e escala dentro de uma região; quando essa máscara é aplicada em uma área da imagem, computa-se então seu valor, e é o valor associado à característica que permite diferenciar regiões. O algoritmo para o cálculo dos valores de todas as características *Haar-like,* para as bases escolhidas, em uma dada imagem, pode ser descrito da seguinte maneira:

- Calcular a Integral da Imagem para todos os pixels, onde a Integral no ponto (x, y) é igual a soma dos valores dos pixels menores ou iguais a x e y.
- Definir a base da característica *Haar-like* em altura e largura, onde estas definem quantas divisões existem em cada eixo. Por exemplo, no caso da característica 1 da figura 3-1, seria largura 3 e altura 1, pois existem duas retas verticais dividindo o retângulo, e nenhuma reta horizontal.
- Para cada mudança de escala da base, deslocar a característica ao longo de toda a imagem.
- Para cada posição da característica dentro da imagem, calcula-se o valor das diferenças de intensidades baseado no padrão da base.
- Repetir o processo, a partir do segundo passo, para cada base escolhida.
- Repetir o processo para todos os exemplos positivos e negativos

Algoritmo 3-1: Cálculo das características *Haar-like*

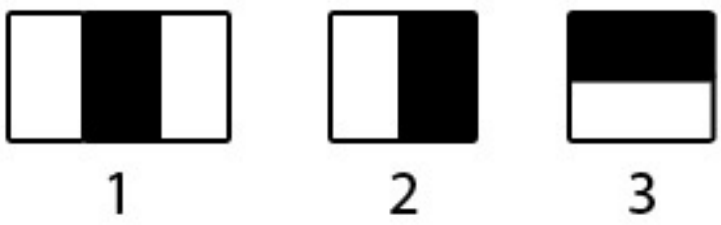

Figura 3-1: Bases *Haar-like* usadas no projeto

A partir desse algoritmo, derivaram-se todas as características possíveis numa imagem de 24x24 pixels, usando-se essas três bases, e cada uma dessas características foi então calculada para cada um dos exemplos positivos e negativos. Neste passo, não foi escolhida nenhum característica ótima, portanto todas são consideradas

classificadores fracos. O próximo passo é analisar o conjunto de todas as características e escolher aquelas que possuem o menor erro de classificação para então combiná-las em um classificador forte.

3.3.2 Treinamento usando *AdaBoost*

O algoritmo de aprendizagem de máquina desenvolvido neste projeto foi baseado em [5], e tem como premissa usar uma série de classificadores fracos para formar um classificador forte, e então cascatear vários estágios de classificadores fortes. O algoritmo abaixo descreve o processo de formação do classificador forte:

- Seja um conjunto de exemplos $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, onde x_i são as imagens, y_i é 0 para exemplos negativos e 1 para exemplos positivos, e n é o número total de exemplos
- Inicializar os pesos $w_{1,i} = \frac{1}{2m}$ onde m é o número total de exemplos da categoria. Por exemplo, se tiverem 100 exemplos positivos e 50 negativos, os pesos iniciais dos exemplos positivos e negativos serão respectivamente $\frac{1}{2\times100}$ e $\frac{1}{2\times50}$.
- Para t=1 até T (número de classificadores fracos) faça:
  1. Normalizar os pesos $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$
  2. Selecionar o melhor classificador fraco de acordo com o seguinte erro ponderado:
     $$\epsilon_t = \min_{f,p,\theta}(\sum_i w_i |h(x_i, f, p, \theta) - y_i|)$$
  3. Definir $h_t(x) = h(x, f_t, p_t, \theta_t)$, onde f_t, p_t e θ_t são minimizadores de ϵ_t. f_t é uma característica *Haar-like*.
  4. Atualizar os pesos:
     $$w_{t+1,i} = w_{t,i}\beta_t^{\,1-e_i}$$
     onde $e_i = 0$ se o exemplo foi classificado corretamente ou $e_i = 1$ caso contrário, e $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$
- O classificador forte é então dado por:

$$C(x) = \begin{cases} 1 & \text{se } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{caso contrário} \end{cases}$$

onde $\alpha_t = \log \frac{1}{\beta_t}$

Algoritmo 3-2: Treinamento de classificadores utilizando *AdaBoost*

A partir desse algoritmo cria-se um classificador forte combinando-se *T* classificadores fracos. Para criar a cascata de classificadores fortes basta repetir o processo para uma diferente combinação de imagens (ou pelo menos mudando parte dos exemplos negativos) até obter o número de estágios desejado. A dificuldade do algoritmo se dá em encontrar o *threshold* (θ) e a característica *Haar-like* (f) que, simultaneamente, minimizam o erro de classificação ϵ_t, ou seja, selecionar o melhor classificador fraco (passo 2 do *loop*). A estratégia usada para solucionar o problema da seleção pode ser descrita pelo seguinte algoritmo:

- Definir T^-, T^+ como a soma dos pesos atuais respectivamente dos exemplos negativos e positivos.
- Atribuir à variável *min_err* um valor muito elevado.
- Para c=1 até C (número de características totais)
  1. Selecionar o valor da característica c para todos os exemplos positivos e negativos.
  2. Ordenar os exemplos em ordem crescente de acordo com esses valores.
  3. Calcular S^- e S^+ como sendo a soma cumulativa dos pesos negativos e positivos respectivamente.
  4. Definir $e1 = S^+ - S^- + T^-$ e $e_2 = S^- - S^+ + T^+$.
  5. Entre todos os exemplos selecionar $\text{erro} = \min(e_1, e_2)$.
  6. Se *erro* < *min_err*, então *min_err* = *erro* e guarda-se o índice da característica com o erro mínimo.

Algoritmo 3-3: Escolha do melhor classificador fraco

O número de classificadores fracos escolhido para cada estágio foi 20, e o número de estágios escolhido na cascata foi 5. Para o treinamento, foram usados cerca de 1000 exemplos positivos e 2000 exemplos negativos, todas as imagens com dimensões de 24x24, e em escala de cinza. As imagens usadas no treinamento podem ser encontradas

em [21]. Para o treinamento de cada estágio da cascata, criou-se um subconjunto aleatório com metade dos exemplos negativos. O tempo total para treinar os classificadores foi de aproximadamente 25 minutos em um computador com processador *Intel Core i7* e com *6GB* de memória *RAM*.

Os parâmetros das melhores características selecionadas foram salvos em um arquivo, uma vez que é inviável fazer o treinamento cada vez que se queira detectar faces em uma imagem.

3.3.3 Detecção em múltipla escala

Com os classificadores já treinados, o próximo passo é implementar um módulo que busque na imagem de entrada os padrões desejados e identifique as regiões classificadas corretamente.

Para que a detecção facial seja feita em múltipla escala, é necessário redimensionar a imagem ou a janela de detecção iterativamente. Como a janela base dos padrões é de 24x24 pixels, qualquer face na imagem que seja muito maior do que esse tamanho, não será detectada corretamente. Optou-se por reduzir a escala da imagem ao invés de ampliar a escala da janela, porque este último acaba tornando o algoritmo mais lento. Foi usada uma técnica chamada Pirâmide de Imagens (*Image Pyramid*) na qual, a partir da imagem original, é criada uma coleção de imagens com escala reduzida. Para cada escala, a janela de detecção varre a imagem inteira (figura 3-2). O algoritmo abaixo detalha o processo de detecção:

- Escolher um fator de escala para criar a pirâmide e os incrementos de posição nos dois eixos para o deslocamento da janela.
- Enquanto a área da imagem for maior do que a área da janela de detecção, reduzir a escala.
- Para cada escala da imagem, calcular a integral de imagem para todos os pixels.
- Deslocar a janela de detecção ao longo de toda a imagem.
- Para cada posição da janela na sub-região da imagem:
    1. Para cada estágio do classificador forte aplicar todos os classificadores fracos.
    2. Se a classificação foi positiva ao final de um estágio, prosseguir para o próximo, caso contrário, terminar a classificação para essa posição e deslocar a janela para a próxima posição.

3. Se a classificação foi positiva para todos os estágios, salvar as coordenadas da região.

- Ao final do processo de varredura em todas as escalas, analisar as regiões classificadas corretamente e agrupar as sobreposições.

Algoritmo 3-4: Detecção em múltipla escala

O último passo do algoritmo é importante porque como a varredura é feita em múltiplas escalas, frequentemente quando há faces na imagem, a mesma região classifica positivamente várias vezes criando sobreposições. Além disso, outro fator que causa *overlaps* é o deslocamento da janela de detecção ao longo da imagem. Dependendo de quão pequeno seja o incremento do deslocamento, uma mesma face pode estar presente em mais de uma janela ao longo da detecção. O método usado para agrupar as sobreposições foi encontrar as intersecções das regiões classificadas positivamente (duas de cada vez) e caso a área de intersecção seja maior do que 1/3 do menor retângulo, então o menor retângulo é descartado e o maior permanece.

3.4 Extração de Características e Reconhecimento

Neste módulo do projeto, o objetivo é desenvolver um algoritmo capaz de identificar com confiabilidade, nas imagens de entrada, faces cadastradas no banco de dados. É importante que o sistema seja o máximo possível invariante às condições ambientes (e.g: iluminação) e às expressões faciais. Os códigos em MATLAB desenvolvidos neste módulo podem ser encontrados no Apêndice B.

Este módulo foi divido em duas funções: Criar e Testar. A primeira função cria um conjunto de parâmetros que definem matematicamente as imagens do banco de dados. A segunda função recebe como entrada uma imagem qualquer (não pertencente ao banco de dados) e analisa a proximidade desta com as do banco de dados através dos parâmetros matemáticos.

O método escolhido para a extração de característica foi o *LDA* porque este oferece um bom poder discriminatório e apresenta um bom desempenho em termos de tempo de execução. A ideia central é reduzir a dimensionalidade do espaço, dada pelas dimensões da mesma, para o número de indivíduos cadastrados no banco de dados. O algoritmo a seguir indica quais passos foram usados neste projeto para o desenvolvimento de um extrator de características baseado em *LDA* a partir de um banco de imagens de faces.

- Seja N o número total de imagens a serem analisadas e M o número de indivíduos.
- Normalizar as N imagens para que tenham a mesma largura W e altura H.
- Transformar as imagens em um vetor-coluna de dimensões $(WH) \times 1$.
- Seja x_i a i-ésima imagem vetorizada e C o número de classes (cada classe representa um indivíduo)
- Calcular a média de todas as imagens (μ) e as médias das classes $(\mu_1, \dots, \mu_M)$
- Criar uma matriz A de ordem $WH \times M$, na qual cada coluna é dada por $\mu_j - \mu$, para j de 1 até M.
- Cria-se outra matriz B de ordem $WH \times N$ na qual cada coluna é dada por $x_i - \mu_j$, para i de 1 até M, onde μ_j é a média da classe a qual pertence a imagem vetorizada x_i.
- Sejam S_b e S_w respectivamente as matrizes de dispersão entre classes e dentro da própria classe. Então $Sb = A \times A^T$ e $S_w = B \times B^T$, ambas de ordem $WH \times WH$.
- Definir as matrizes auxiliares $C = A^T \times A$ e $D = B^T \times B$, ambas de ordem $M \times M$.
- Seja V a matriz dos autovetores de C.
- Ordenar em ordem decrescente os autovetoers da matriz V de acordo com os autovalores.
- Encontra-se então M autovetores de S_b, que são dados por $A \times V$.
- Cria-se uma matriz diagonal D com os autovalores de $A \times V$.
- Calcular a matriz $Z = (A \times V) \times D^{-1/2}$.
- Definir a matriz auxiliar $E = Z^T \times B$.
- Calcular os autovetores de $E \times E^T$ e formar uma matriz U com esses vetores.
- O subespaço final é então dado por $T = Z \times U$.
- Projeta-se a média de cada classe (cada coluna da matriz A) nesse subespaço.

Algoritmo 3-5: Análise Linear Discriminante

Depois que todas as variáveis forem calculadas e as imagens do banco de dados forem projetadas no subespaço dos autovetores, são salvos em um arquivo a matriz de projeção, as matrizes T e A, e os escalares W, H, N. Com isso, as imagens do banco de dados não são mais necessárias no processo de reconhecimento. Porém, cada vez que forem adicionados novos indivíduos ao banco de dados, é necessário computar novamente todas as matrizes.

Para o reconhecimento, foi utilizada uma métrica baseada na distância euclianda. Primeiro, normaliza-se a imagem-teste para que tenha as mesmas dimensões das imagens que foram usadas para criar o subespaço. Depois transforma-se a imagem-teste em um vetor coluna, subtrai-se o vetor A e então projeta-se a mesma no subespaço T. Calcula-se a norma da diferença entre a projeção da imagem-teste e cada projeção salva no banco de dados. A projeção que originou a menor norma é a mais próxima da imagem-teste, mas não quer dizer necessariamente que há uma correspondência positiva, uma vez que só por ser a imagem mais próxima não significa ser próxima o suficiente, daí a necessidade de se estabelecer um critério de aceitação. O critério de aceitação adotado neste projeto baseia-se em analisar a razão entre a menor distância e a segunda menor distância. Se esta razão for muito pequena, a chance da imagem-teste corresponder a mesma classe da imagem de menor distância é elevada.

4 Resultados

Esta parte do trabalho apresenta os resultados obtidos nos testes dos módulos do sistema de reconhecimento e também descreve como os testes foram realizados. Os códigos desenvolvidos para realizar os testes podem ser encontrados no Apêndice C.

4.1 Teste do detector facial

Depois que o algoritmo do detector facial foi desenvolvido, foram usadas diversas imagens para testar a eficiência e a acurácia do detector. As imagens usadas para o teste foram retiradas do banco de imagens *Faces in the Wild* [19]. O módulo de detector recebe como parâmetros: a imagem de entrada, uma escala inicial, um fator de escala e incrementos de posição da janela de detecção (de quanto em quanto a janela se desloca nos eixos x e y). Para que fosse possível visualizar as detecções, foi desenvolvida uma pequena rotina que desenha na imagem os retângulos dados pelas coordenadas da saída do detector. Apesar da maioria das imagens testadas terem obtido um resultado satisfatório, algumas apresentaram falsos positivos e falsos negativos. A Figura 4-1 mostra os resultados do teste que obtiveram melhor desempenho, e a Figura 4-2 mostra os casos problemáticos, que apresentaram falsos positivos e/ou falsos negativos.

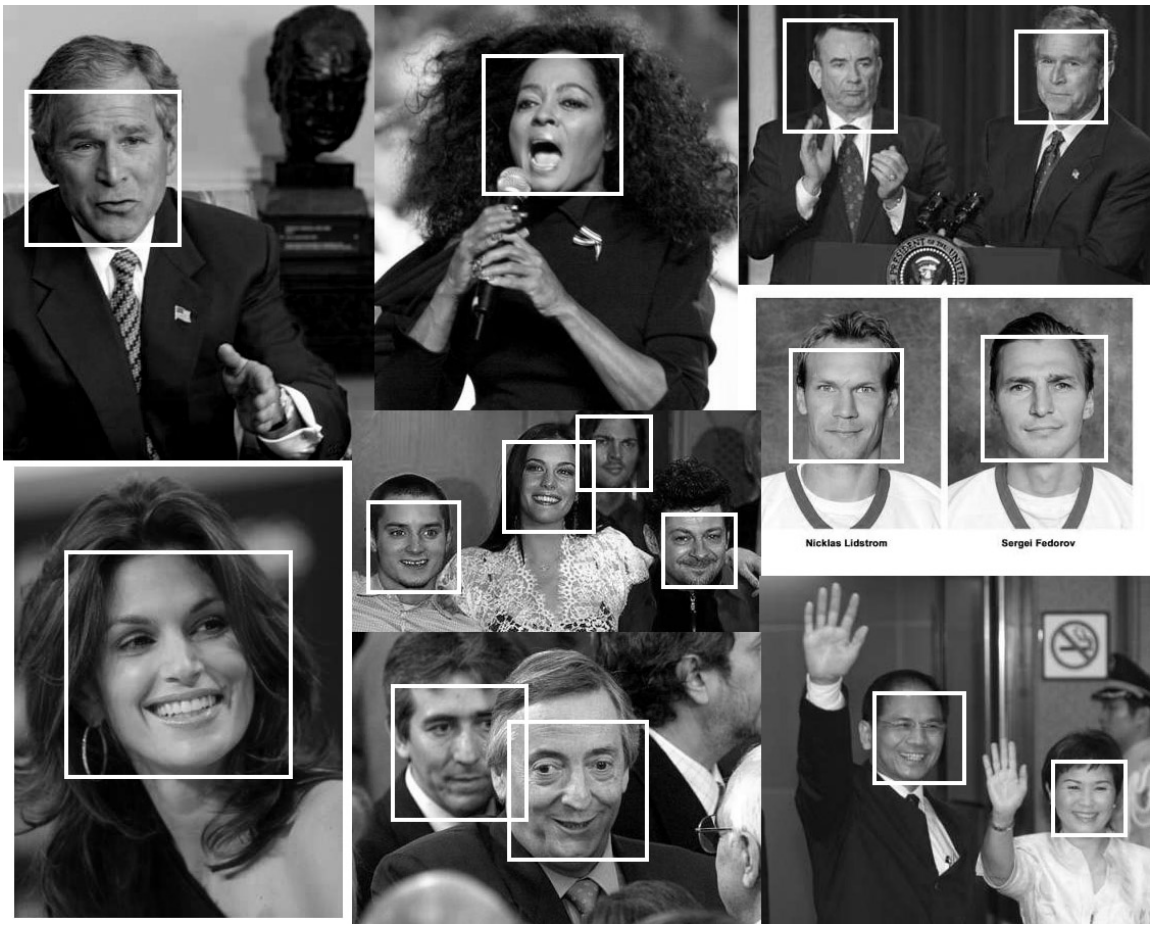

Figura 4-1: Resultados que obtiveram melhor desempenho.

Observou-se que o resultado da detecção é significantemente afetado pelo fator de escala e pelo incremento de deslocamento da janela de detecção. Por exemplos, algumas imagens, para um determinado fator de escala, apresentaram resultados errôneos, e quando o fator foi alterado, o resultado foi correto. O mesmo ocorreu para os incrementos de descolamento da janela de detecção. Após diversos testes, chegou-se a conclusão que um fator de escala de 0,97 e incrementos de 2 pixels para cada um dos eixos ofereceram um bom resultado em geral. A escala inicial escolhida foi de 0,5 para poupar esforços computacionais. Em média, levou-se 10 segundos para detectar faces em uma imagem de dimensões 410 x270 pixels. Com a escala inicial em 1,0, o tempo médio foi de 50 segundos e as detecções foram praticamente idênticas às da outra escala. Uma possível solução para reduzir o tempo de detecção é criar a cascata com menos filtros nos estágios iniciais para que a maioria das regiões negativas seja descartada de início.

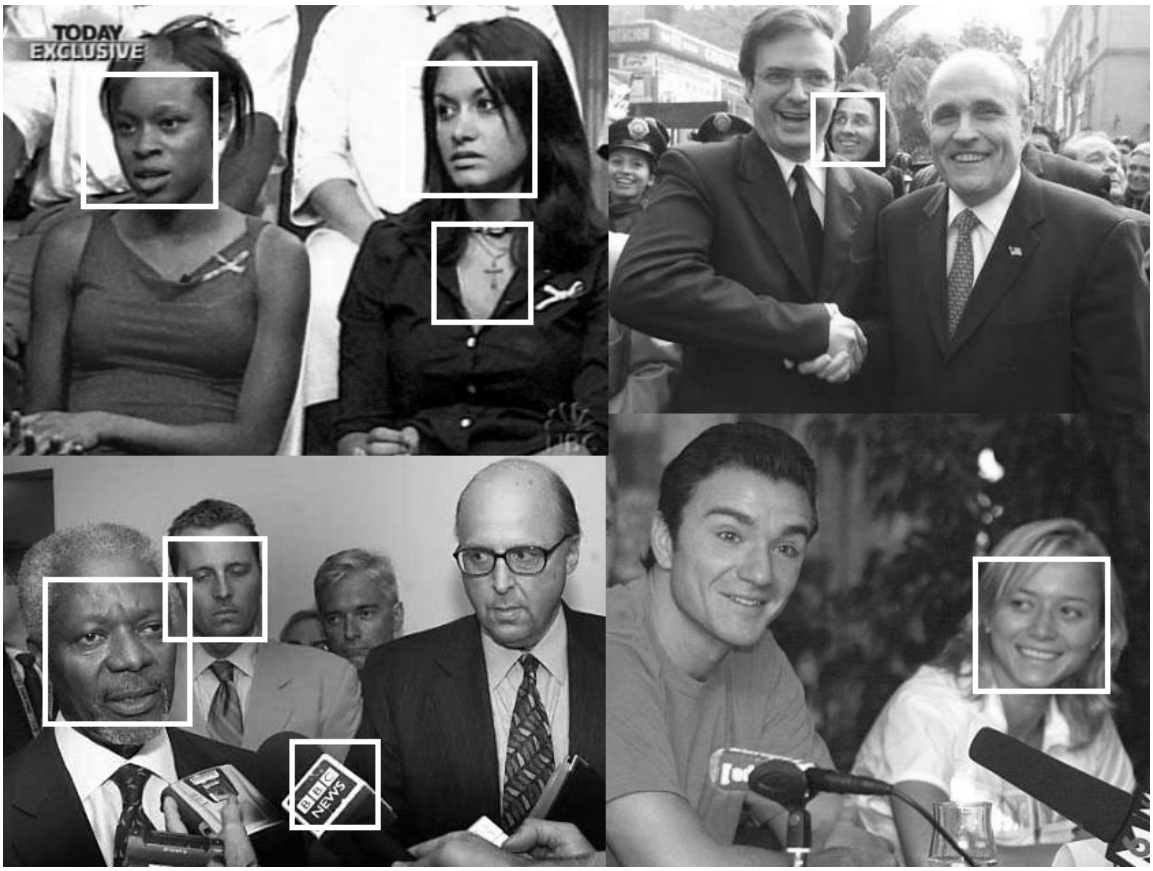

Figura 4-2: Casos problemáticos com falsos positivos e falsos negativos.

Uma possível causa para a presença de falsos positivos e falsos negativos nos resultados é que o treinamento dos classificadores precisaria de mais exemplos positivos e negativos. Neste projeto foram usados cerca de 1.000 exemplos positivos e 2.000 exemplos negativos. Os autores *Viola* e *Jones* informaram em uma versão atualizada de seu artigo [20] que eles utilizaram aproximadamente 5.000 exemplos positivos e 10.000 exemplos negativos para treinar os classificadores.

4.2 Teste da Extração de Características e Reconhecimento

Para testar o módulo de extração de características e reconhecimento foi usado o banco de imagens *The Database of Faces*. É de suma importância que as imagens-teste sejam distintas das imagens que serão cadastradas no banco de dados. Seguindo este critério, foram separadas 5 imagens de cada um dos 40 individuos para o cadastro e mais 5 diferentes para serem testadas, totalizando 200 imagens para cadastro e 200 para testes.

Tanto para o cálculo dos parâmetros quanto para a execução dos testes, é suposto que as imagens de entrada são apenas faces. Isto é importante para garantir o desempenho deste módulo.

O tempo total gasto pelo algoritmo para analisar as 200 imagens de dimensões 92x112 pixels, montar um subespaço de dimensão igual a 40, e projetar as médias das classes nesse subespaço, foi de 1,6 segundos. A matriz de projeção possui dimensões 40x40 e cada coluna representa a projeção de um indivíduo. Portanto o segundo índice da matriz identifica o número do individuo. Para facilitar o controle sobre os indivíduos, o algoritmo gera no final da execução uma lista com uma foto do individuo e o seu número de identificação (índice), como mostra a Figura 4-3.

Figura 4-3: Imagem gerada pelo algoritmo de caracterização para auxiliar o controle do banco de dados.

Ao final da caracterização, as variáveis de interesse são salvas em um arquivo do *MATLAB* para serem usadas pelo submódulo de teste.

A próxima etapa é usar o conjunto de imagens-teste para avaliar a precisão deste módulo. A métrica usada para avaliar a proximidade das projeções baseia-se na norma da diferença das projeções. Essa norma é então elevada a uma potência arbitrária para maximizar as distâncias. Define-se a acurácia como sendo $\alpha = 1 - \frac{m1}{m2}$ onde m_1 é a menor distância e $m2$ a segunda menor distância. Define-se então o critério de aceitação como

$\alpha > \mathrm{k}$, onde *k* é uma constante arbitrária que varia de 0 a 1. O módulo de teste recebe como entrada uma imagem desconhecida e como saída retorna valor zero caso não haja correspondência (rejeição), ou o número do índice do individuo caso a acurácia seja maior do que *k*. Quanto maior o valor de *k,* mais seletivo o sistema é, evitando falsos-positivos, ao custo de rejeitar mais imagens, inclusive aquelas que potencialmente seriam classificadas corretamente para um *k* menor.

Para testar a robustez do sistema, foram realizados quatro tipos de testes. O primeiro teste consiste em utilizar como entrada as imagens-teste sem nenhuma modificação, apenas para avaliar a tolerância do sistema quanto à variação de pose, de expressão facial do indivíduo e de iluminação ambiente. O segundo teste consiste em adicionar ruído nas imagens para avaliar à sensibilidade quanto à qualidade da imagem. O terceiro teste consiste em cortar parte da imagem e/ou inserir tarjas, para avaliar tolerância quanto à obstrução parcial. E o quarto teste, avalia o sistema com faces de indivíduos não pertencentes ao banco de dados.

No primeiro teste, para $\mathrm{k} = 0{,}9$, das 200 imagens-teste usadas, 130 foram classificadas corretamente, 1 foi classificada incorretamente e 69 foram rejeitadas ($\alpha < \mathrm{k}$). Para $\mathrm{k} = 0{,}7$, 156 imagens foram classificadas corretamente, 5 foram classificadas incorretamente e 39 foram rejeitadas. A Figura 4-4 mostra um gráfico relacionando k com as classificações corretas, incorretas e as rejeições. Pode-se observar que quanto maior o α, menor é a taxa de erros, mas em compensação, mais imagens são rejeitadas.

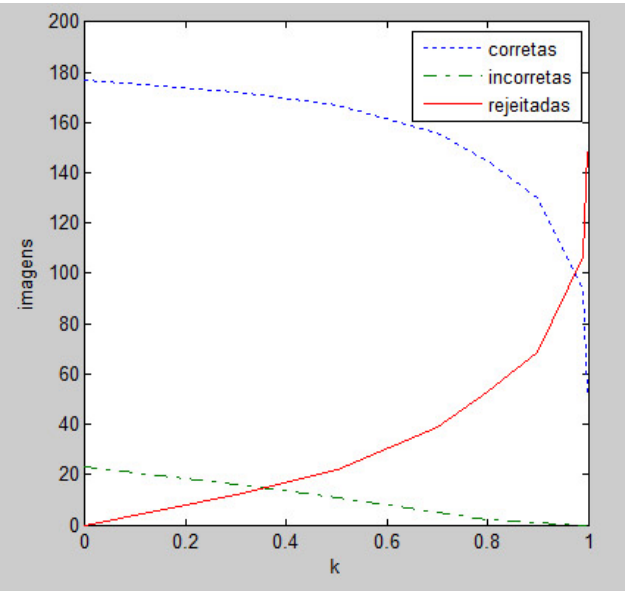

Figura 4-4: Gráfico relacionando *k* com as classificações das imagens-teste.

No segundo teste, foram adicionados diferentes tipos de ruído às imagens-teste. O primeiro tipo foi o ruído gaussiano, que alterou muito pouco o resultado obtido anteriormente: para $k = 0{,}9$, 129 imagens foram classificadas corretamente, 1 foi classificada incorretamente 70 foram rejeitadas. Para $k = 0{,}7$, o resultado foi idêntico ao teste realizado sem a adição de ruído. Foram adicionados também ruído "*salt and pepper*" (pixels brancos e pretos aleatoriamente adicionados na imagem) e ruído de *Poisson* que também não afetaram muito o desempenho, obtendo resultados quase que idênticos ao gaussiano. Isso permite dizer que o sistema é tolerante aos tipos de ruídos testados.

Na primeira parte do terceiro teste foram adicionadas programaticamente tarjas pretas nas imagens-teste como mostra a figura 4-5. A finalidade deste teste é avaliar a tolerância quanto a partes da face que não estão visíveis. As tarjas adicionadas têm dimensões de 30x10 pixels, ocupando cerca de 3% da área total da imagem e posicionadas abaixo da região do nariz. Com esta tarja, para $k = 0{,}9$, 111 imagens foram classificadas corretamente, nenhuma incorretamente, e 89 foram rejeitadas. Na segunda parte, as tarjas adicionadas têm dimensões de 20x40 pixels, cobrindo aproximadamente 7,7% da área da imagem e foram posicionadas na região dos olhos. Para esta tarja e $k = 0{,}9$, 47 imagens foram classificadas corretamente, 1 incorretamente e 152 foram rejeitadas. Um interessante ponto a se observar é que a cor da tarja influenciou os resultados. Mudando-se a cor da tarja de preto para branco, 69 imagens foram classificadas corretamente ao invés de 47. A figura 4-7 mostra um gráfico relacionando o número de classificações corretas, incorretas e rejeições, para $k = 0{,}9$, em função da cor da tarja, que foi variada do preto ao branco (0 a 255). Quando a cor da tarja assume um valor próximo de 135, o número de classificações corretas atinge seu máximo (130) e se iguala ao número de classificações corretas para o teste realizado sem a tarja. Isso indica que, se ignorarmos a cor da tarja, e considerarmos apenas o efeito da obstrução, conclui-se que o sistema consegue identificar indivíduos mesmo que a região dos olhos não esteja visível. Pode-se então dizer que, nas condições testadas, o sistema foi tolerante à obstrução parcial da face. Na terceira parte, as imagens foram cortadas pelas laterais, como mostra a figura 4-8. Este teste obteve um desempenho insatisfatório: Para $k = 0{,}9$, 18 imagens foram classificadas corretamente, 13 foram classificadas incorretamente e 169 foram rejeitadas. A razão para isso é porque quando as imagens são cortadas, e são então projetadas no subespaço das características, pode haver um desalinhamento das regiões de interesse. Por exemplo, a posição da região dos olhos da imagem-teste,

quando redimensionada para a projeção no subespaço das características, pode não ser coincidente com a posição da região dos olhos da imagem do banco de dados, mas com a região da testa ou do nariz. Uma sugestão para contornar este problema é fazer um alinhamento dos olhos e da boca antes da imagem ser projetada, garantindo assim que as regiões de interesse, quando projetadas, sejam sempre coincidentes. Isso pode ser feito treinando-se classificadores com o algoritmo de *Viola-Jones* para que reconheçam olhos e boca.

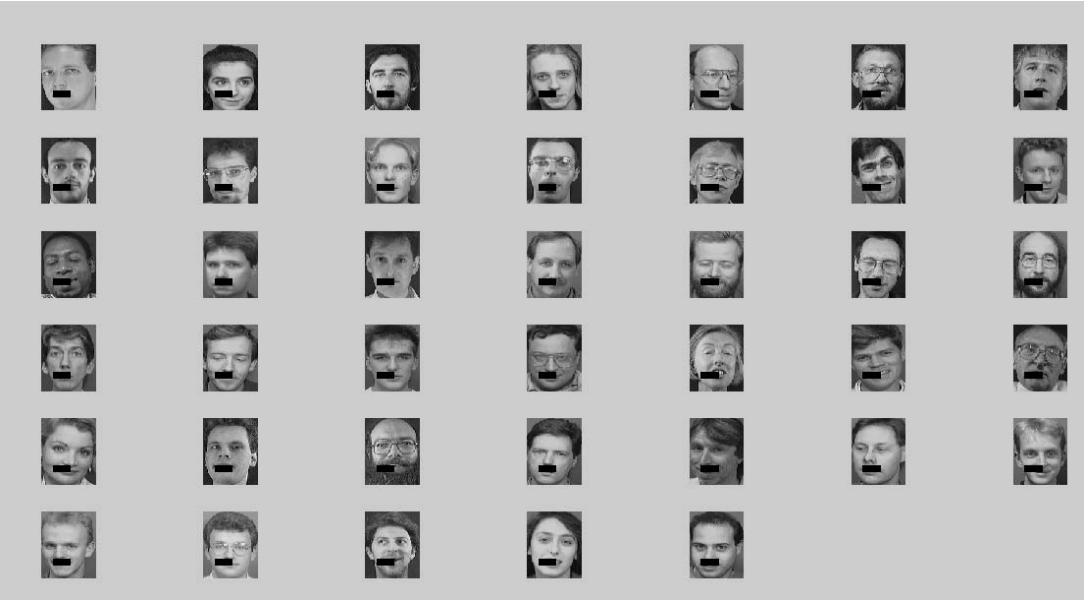

Figura 4-5: Imagens-teste com obstrução parcial do rosto (3% da área da imagem)

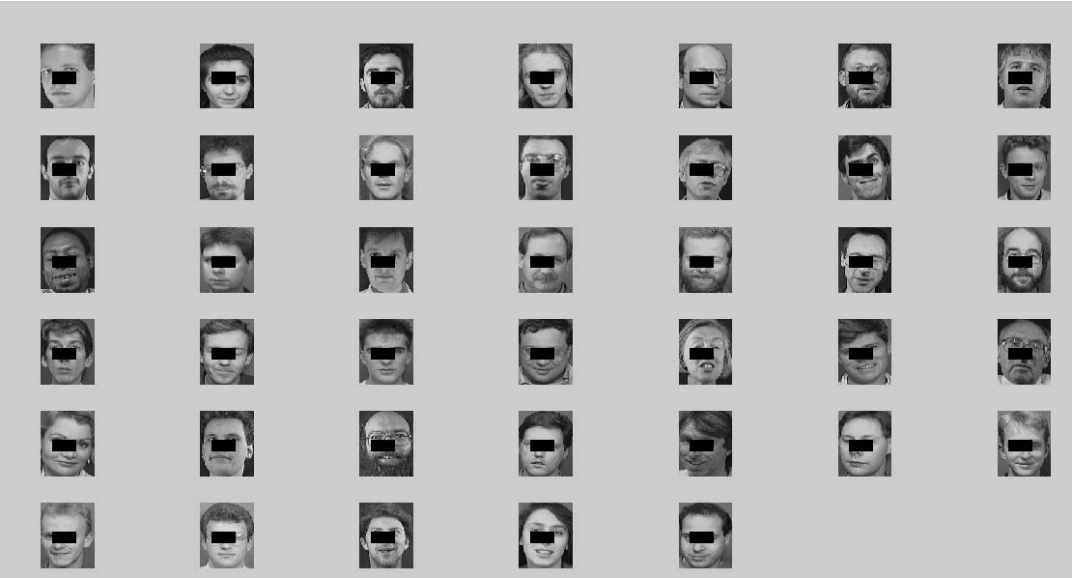

Figura 4-6: Imagens-teste com obstrução parcial do rosto (7,7% da área da imagem)

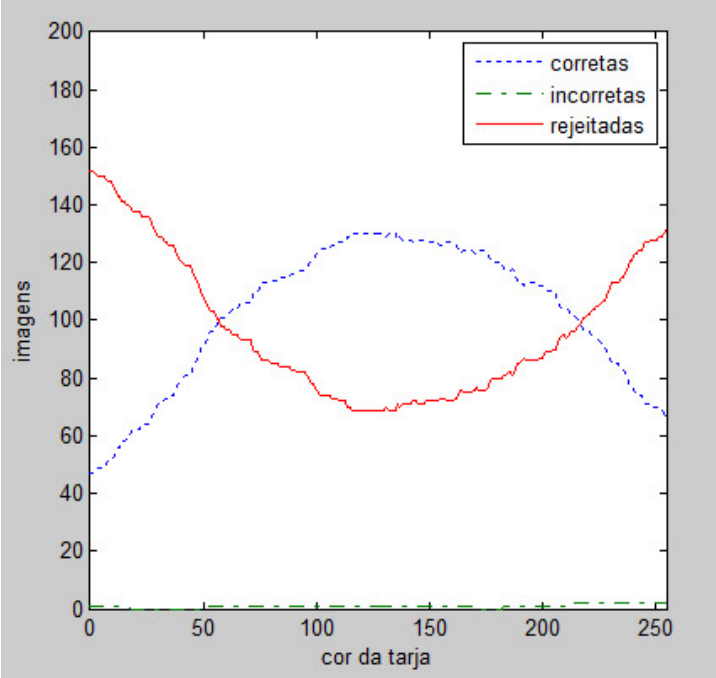

Figura 4-7: Gráfico relacionando o resultado das classificações com a cor da tarja adicionada na imagem (figura 4-6).

Figura 4-8: Imagens-teste após serem cortadas

O quarto teste utilizou imagens de 130 indivíduos não pertencentes ao banco de dados para testar se haveria algum falso-positivo. Essas imagens foram retiradas do *The Color FERET Database*. Para $\mathrm{k} = 0{,}9$, das 130 imagens usadas, o sistema rejeitou todas, não havendo nenhum falso positivo. Para $k = 0{,}7$, o sistema rejeitou 111, e 19 foram falsos positivos.

Com base nos resultados dos quatro testes realizados, pode-se concluir que o sistema de reconhecimento, em geral, tem uma baixa taxa de erros, para $\mathrm{k} > 0{,}9$ e na maioria das condições testadas, apresentou resultados satisfatórios, reconhecendo indivíduos mesmo com a região dos olhos e a região da boca coberta. O maior problema encontrado é devido ao não-alinhamento das regiões de interesse, que acabam gerando falsos positivos e evitando que as faces sejam classificadas corretamente.

5 Conclusões

Este trabalho apresentou um estudo sobre sistemas de reconhecimento facial automatizados, cobrindo desde a parte teórica até os passos necessários para o desenvolvimento de um programa computacional baseado nos métodos estudados. Foi verificado que a implementação de um sistema de reconhecimento facial exige algoritmos robustos, capazes de lidar com variações de pose do indivíduo, ruídos na imagem de entrada e diferentes configurações de iluminação ambiente. O sistema desenvolvido consistiu de um módulo de detecção facial, outro de extração de características e outro de reconhecimento. O módulo de detecção facial foi baseado no algoritmo de *Viola-Jones* e apresentou resultado satisfatório. O módulo de extração de característica, baseado na análise linear discriminante, mostrou-se eficiente ao calcular o subespaço de características, levando 1,6 segundos para analisar 200 imagens. Das 200 imagens-teste, o módulo de reconhecimento, no melhor caso, identificou 130 imagens corretamente, 1 incorretamente e o restante foi rejeitado pelo critério de aceitação. Além disso, mostrou-se tolerante a ruído e conseguiu identificar indivíduos mesmo com a região da boca ou dos olhos cobertos. Em contrapartida, quando as regiões de interesse não estavam alinhadas, a taxa de erros aumentou significativamente.

5.1 Sugestões para trabalhos futuros

Para melhorar o desempenho do sistema, recomenda-se treinar os classificadores do detector facial com pelo menos 5,000 exemplos para cada uma das categorias, usar menos classificadores fracos na cascata nos estágios iniciais e mais nos estágios finais, e usar pelo menos 10 estágios na cascata. Com o aumento dos exemplos, o sistema aumenta o número de classificações corretas e com menos classificadores fracos nos estágios iniciais, o algoritmo é executado mais rápido. O número de estágios na cascata ajuda a reduzir falsos positivos sem comprometer muito o tempo de execução.

Para os módulos de extração de características e de reconhecimento, uma sugestão é alinhar a região dos olhos e da boca antes de efetuar a extração, e também antes de projetar a imagem-teste no subespaço das características. Assim, é esperado que se reduza a taxa de erros. Esse alinhamento pode ser feito treinando-se classificadores com o algoritmo de Viola-Jones para reconhecer as regiões de interesse.

Referências Bibliográficas

[1] M. Bledsoe. The model method in facial recognition. Technical Report PRI 15, Panoramic Research Inc., Palo Alto, CA, 1964.

[2] M. Kelly. Visual identification of people by computer. Technical Report AI 130,Stanford, CA, 1970.

[3] W. Zhao; R. Chellappa. Face Processing: Advanced Modeling and Methods. Academic Press, Inc., Orlando, FL, USA, 2005

[4] USA TODAY. FBI uses facial-recognition technology on DMV photos. USA TODAY, 2009. Disponível em: http://usatoday30.usatoday.com/tech/news/2009-10-13-fbi-dmv-facial-recognition_N.htm

[5] P. Viola; M. Jones. Rapid object detection using a boosted cascade of simple features. In Proc. Of CVPR, 2001

[6] G. Yang; T. S. Huang, “Human Face Detection in Complex Background”. Pattern Recognition, vol. 27, no. 1, 1994.

[7] T.F.Cootes, C.J.Taylor, A.Lanitis, Active Shape Models: Evaluation of a Multi-Resolution Method for Improving Image Search, in Proc. British Machine Vision Conference, 1994

[8] V. Jain; E. Learned-Miller. FDDB: A benchmark for face detection in unconstrained settings. Technical report, University of Massachusetts, Amherst, 2010

[9] M. Turk; A. P. Pentland, “Eigenfaces for recognition,” J. Cognitive Neuroscience, 1991.

[10] L. Sirovich; M. Kirby, “Low dimensional procedure for the characterization of human faces,” Journal of the Optical Society of America 4, No. 3, pp. 519–524, 1987.

[11] K. Etemad; R. Chellappa, “Discriminant analysis for recognition of human face images,” J. Opt. Soc. Amer. A: Opt. Image Sci. Vis., vol. 14, no. 8, pp. 1724–1733, 1997.

[12] Li-Fen Chen, Hong-Yuan Mark Liao, Ming-Tat Ko, JaChen Lin, and Gwo-Jong Yu, “A new LDA-based face recognition system which can solve the small sample size problem,” Pattern Recognition, vol. 33, pp. 1713–1726, 2000.

[13] H. Yu; J. Yang, “A direct LDA algorithm for high-dimensional data with application to face recognition,” Pattern Recognition, vol. 34, pp. 2067–2070, 2001.

[14] Draper, B.A., Baek, K., Bartlett, M.S., Beveridge, J.R.: Recognizing faces with PCA and ICA. Comput. Vis. Image Underst. 91(1–2), 115–137, 2003

[15] F. Samaria; S.Young. HMMbased architecture for face identification. Image and Vision Computing, 12:537–583, 1994.

[16] A. Lanitis; C. Taylor; T. Cootes. Automatic face identification system using flexible appearance models. Image and Vision Computing, 13:393–401, 1995.

[17] P.J. Phillips, H. Wechsler, J. Huang, P. Rauss, "The FERET database and evaluation procedure for face recognition algorithms," Image and Vision Computing J, Vol. 16, No. 5, pp. 295-306, 1998.

[18] P.J. Phillips, H. Moon, S.A. Rizvi, P.J. Rauss, "The FERET Evaluation Methodology for Face Recognition Algorithms," IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 22, pp. 1090-1104, 2000.

[19] Tamara L. Berg, Alexander C. Berg, Jaety Edwards, David A. Forsyth, Neural Information Processing Systems (NIPS), 2004

[20] P. Viola; M. Jones, “Robust Real-Time Face Detection”, International Journal of Computer Vision 57(2), 137-154, 2004.

[21] R. Pless, Project 3: Faces!. Disponível em: http://www.cs.wustl.edu/~pless/559/Projects/faceTrainingData.zip

[22] H. Gulati, D. Aggarwal, A. Verma, P. Sandhu, Face Recognition using Hybrid Histogram & Eigen value Approach, International Journal of Research in Engineering and Technology (IJRET), Vol. 1, No. 1, pp. 65

Apêndice A – Códigos em MATLAB (Detecção Facial)

Integral da Imagem:

```
function [ii] = IntegralImage(image)
%Entrada --> image: Imagem em escala de cinza
%Saída   --> Matriz 2D com os valores da Integral para cada pixel da
imagem

[W, H]=size(image);
ii = zeros(W,H);

for y=1:H
   for x=1:W
     ii(x,y) = image(x,y);
     if (x > 1)
        ii(x,y) = ii(x,y) + ii(x-1,y);
     end
     if (y > 1)
        ii(x,y) = ii(x,y) + ii(x,y-1);
     end
     if (y > 1) && (x > 1)
        ii(x,y) = ii(x,y) - ii(x-1,y-1);
     end
   end
end
end

function Val = SRect(iim,x1,y1,w,h)
%Entrada --> iim: Integral da Imagem para todos os pixels (Matriz de
%                 mesma dimensão da imagem)
%           x1,x2: Coordenadas do ponto superior esquerdo do retângulo
%           w,h:   largura e altura do retângulo
%Saída --> Soma das intensidades dos pixels na região especificada.
    x2 = x1+w;
    y2 = y1;
    x3 = x1;
    y3 = y1+h;
    x4 = x1+w;
    y4 = y1+h;
    Val = iim(y1,x1)+iim(y4,x4)-iim(y2,x2)-iim(y3,x3);
end
```

Características *Haar-like*:

```
function ValHaar = CalcHaar(iim,x,y,w,h,t)
%Entrada --> iim: Integral da Imagem para todos os pixels (Matriz de
%                 mesma dimensão da imagem)
%            x,y: posição do pixel superior e esquerdo do retângulo de
%                 característica
%            h,w: altura e largura do retângulo
%            t: tipo da base
%Saída   --> Valor da característica definida pelos parâmetros de entrada

    switch t

       %Base tipo 1 (Ver figura 3-1 da monografia)
        case 1

            ValHaar = SRect(iim,x+w,y,w,h) - (SRect(iim,x,y,w,h) +...
                      SRect(iim,x+2*w,y,w,h));
       %Base tipo 2 (Ver figura 3-1 da monografia)
        case 2

            ValHaar = SRect(iim,x+w,y,w,h) - SRect(iim,x,y,w,h);
       %Base tipo 3 (Ver figura 3-1 da monografia)
        case 3

            ValHaar = SRect(iim,x,y+h,w,h)-SRect(iim,x,y,w,h);

    end
end


function VHaar = CalcAllHaar(iim,Mascaras)
%Calcula o valor das características de Haar
%Entrada --> iim: Integral da Imagem em todos os pixels (Matriz de mesma
%                 dimensão da imagem)
%            Mascaras: Matriz com as características de Haar
%Saída   --> Vetor cotendo o valor calculado de cada característica



    %VHaar = zeros(size(Mascaras),1);

    for i = 1:size(Mascaras)
      temp = Mascaras(i,:);
      VHaar(i) = CalcHaar(iim,temp(1),temp(2),temp(3),temp(4),temp(5));
    end
end


function MHaar = MascaraHaar(ImageW,ImageH)
%Calcula todas as possíveis características/máscaras para as
%3 bases escolhidas para uma imagem de ImageWxImageH
%Entrada --> iim: Integral da Imagem para todos os pixels (Matriz de
mesma
```

```
%                 dimensão da imagem)
%Saída   --> Vetor com todas as máscaras possíveis

   %Bases [Largura Altura Tipo]
    bases = [3 1 1;2 1 2;1 2 3];

    %Índice da Característica
    i = 1;

    %Pré-aloca a matriz de características
    %Se a imagem inteira fosse considerada,
    %seriam 114.000 características
    %mas por questões de desempenhos
    %as bordas da imageme a escala
    %inicial da base foram deconsideradas,
    %sobrando apenas 59367 características
    MHaar = zeros(59367,5);

    for j = 1:size(bases,1) %Tipo da base
        for w1 = 2:floor(ImageW/bases(j,1)) % Redimensiona a
característica na horizontal
            for h1 = 2:floor(ImageH/bases(j,2)) % Redimensiona a
característica na vertical
                for x = 2:ImageW-w1*bases(j,1) % Desloca a característica
ao longo o eixo x
                    for y = 2:ImageH-h1*bases(j,2) % Desloca a
característica ao longo o eixo y
                        MHaar(i,:) = [x y w1 h1 j];
                        i=i+1;
                    end
                end
            end
        end
    end

end
```

AdaBoost:

```
function Classificadores = AdaBoost(Faces,NFaces,Mascaras,n_class_fracos)
%Entrada --> Faces, NFaces: Características Calculadas para exemplos
%                           positivos e negativos
%            Mascaras: Todas as máscaras possíveis
%            n_cascatas: número de cascatas no classificador forte
%Saída   --> Parâmetros das melhores caracteristicas selecionadas

%Número de características
dados.num_c = size(Mascaras, 1);
%Número de exemplos positivos e negativos
dados.num_ep = size(Faces, 1);
dados.num_en = size(NFaces, 1);

ntotal = dados.num_ep+dados.num_en;
```

```
%Todas os valores das características (positivas e negativas)
dados.pos_neg = [Faces ; NFaces];

%Pesos iniciais
dados.wi = [ones(dados.num_ep, 1)/(2*dados.num_ep); ones(dados.num_en,
1)/(2*dados.num_en)];

%y=0 ou 1 para exemplos negativos e positivos respectivamente
dados.yi = [ones(dados.num_ep, 1); zeros(dados.num_en, 1)];

%inicializa os coeficientes e matrizes de parâmetros
alphas = zeros(n_class_fracos, 1);
cfraco_params = zeros(n_class_fracos,4);
h_res = zeros(ntotal,n_class_fracos);

for t=1:n_class_fracos
    %Normaliza os pesos
    dados.wi = dados.wi ./ sum(dados.wi);
    %Encontra o melhor classificador fraco para o peso atual
    res = ClassificadorFraco(dados);

    beta = res.erro /(1-res.erro);
    %Atualiza os pesos
    dados.wi = dados.wi.*beta.^(1-xor(dados.yi,res.class_result));

    alphas(t) = log(1/beta);

    %Resultado da classificação do classificador fraco
    h_res(:,t) = res.class_result;

    %Guarda os parâmetros
    cfraco_params(t,1) = res.indice;
    cfraco_params(t,2) = res.threshold;
    cfraco_params(t,3) = res.polaridade;
    cfraco_params(t,4) = res.erro;

    disp(sprintf('classificador fraco %d selecionado', t));
end

%Escolhe um treshold para o classificador forte, neste caso
%é a média dos valores dos exemplos positivos
cforte = h_res*alphas;
threshold = median(cforte(1:dados.num_ep));

Classificadores.alphas = alphas;
Classificadores.cfraco = cfraco_params;
Classificadores.threshold = threshold;

end


function resultado = ClassificadorFraco(dados)
%Procura a característica com o menor erro de classificação
```

```
%Entrada --> dados = dados.pos_neg: Características de Haar calculadas
%                                   para exemplos positivos e negaitvos
%                    dados.num_ep, num_en: Número de exemplos positivos
%                                          e negativos
%                    dados.num_c: Número total de Características de Haar
%                    dados.wi: pesos de cada exemplo
%                    dados.yi: 0 para exemplos negativos e 1 para
%                              positivos
%Saída   --> Threshold, Polaridade, Índice e Erro da melhor
%            característica
%            e Resultado da classificação com a caracteristica escolhida
%            para os outros exemplos


ntotal = dados.num_ep+dados.num_en;

%Inicializa as saídas
resultado.erro = inf;
resultado.indice = 1;
resultado.threshold = 0;
resultado.polaridade = 1;

%Soma dos pesos negativos
Tneg = sum(dados.wi(dados.num_ep+1:ntotal,1));
%Soma dos pesos positivos
Tpos = sum(dados.wi(1:dados.num_ep,1));

%Varre todas as características em busca daquela com o menor erro de
%classificação
for i = 1:dados.num_c

    %Seleciona uma característica
    haar_c = dados.pos_neg(:,i);
    %Ordena as características de acordo com seus valores
    [shaar, shaar_i] = sort(haar_c);
    %Ordena os pesos e os 'ys' de acordo com a ordenação anterior
    sw = dados.wi(shaar_i);
    sy = dados.yi(shaar_i);

    %Soma cumulativa dos pesos positivos e negativos
    Spos = cumsum(sw.*sy);
    Sneg = cumsum(sw) - Spos;

    %Erros dos exemplos positivos e negativos
    err_p = Spos+Tneg-Sneg;
    err_n = Sneg+Tpos-Spos;

    %Escolhe o valor que melhor separa as duas classes
    err = min(err_p, err_n);
    [err, ind] = min(err);
    result = zeros(ntotal,1);

    %Classifica os exemplos com o threshold escolhido
    if err_p(ind)<=err_n(ind)
        result(ind+1:end) = 1;
```

```
        result(shaar_i) = result;
        p = -1;
    else
        result(1:ind) = 1;
        result(shaar_i) = result;
        p = 1;
    end
    %Se o erro for o menor encontrado, guarda a característica atual
    if err < resultado.erro
        resultado.erro = err;
        resultado.threshold = shaar(ind);
        resultado.polaridade = p;
        resultado.indice = i;
        resultado.class_result = result;
    end

end
end
```

Treinamento:

```
close all
clear all

tic

%Assume-se que todas as imagens tenham o mesmo tamanho
img_w = 24;
img_h = 24;

%Diretório dos exemplos positivos de faces
diretorio_faces = '../banco de imagens/faces-nfaces/faces/f1/';

%Diretório dos exemplos negativos de faces
diretorio_nfaces = '../banco de imagens/faces-nfaces/nfaces/f1/';

%Extensão do arquivo
ext = '*.GIF';

%Cria lista dos arquivos
lista_exemplos_faces = dir(strcat(diretorio_faces,ext));
lista_exemplos_nfaces = dir(strcat(diretorio_nfaces,ext));

%Todas as possíveis características
if (exist('MHaar.mat','file') ~= 2)
    MHaar = MascaraHaar(img_w,img_h);
    save('MHaar.mat', 'MHaar');
    disp('Características Haar computadas');
else
    load('MHaar.mat');
    disp('Características Haar carregadas');
end

HFaces = zeros(size(lista_exemplos_faces,1),size(MHaar,1));
HNFaces = zeros(size(lista_exemplos_nfaces,1),size(MHaar,1));
```

```
%Ler todas as imagens dos diretórios e calcular em cada uma delas o
%valor de todas as características
for i=1:size(lista_exemplos_faces,1)
    A = imread([diretorio_faces  lista_exemplos_faces(i).name]);
    %Converte para escala de cinza, caso a imagem seja colorida
    if size(A,3)>1
        A = rgb2gray(A);
    end
    A = double(A);
    HFaces(i,:) = CalcAllHaar(IntegralImage(A),MHaar);
end
disp('Características Haar calculadas para exemplos positivos');
for i=1:size(lista_exemplos_nfaces,1)
    A = imread([diretorio_nfaces  lista_exemplos_nfaces(i).name]);
     %Converte para escala de cinza, caso a imagem seja colorida
    if size(A,3)>1
        A = rgb2gray(A);
    end
    A = double(A);
    HNFaces(i,:) = CalcAllHaar(IntegralImage(A),MHaar);
end
disp('Características Haar calculadas para exemplos negativos');

numero_class_fracos = 20;
numero_cascata = 5;
disp('Iniciando o treinamento...');
for i=1:numero_cascata
    HNFaces2 =
HNFaces(randperm(size(HNFaces,1),floor(size(HNFaces,1)/2)),:);
    C(i) = AdaBoost(HFaces,HNFaces2,MHaar,numero_class_fracos);
    disp(sprintf('Estágio %d completo', i));
end
disp('Treinamento finalizado!');
save('Parametros.mat', 'C');
disp('Parâmetros salvos');

toc
```

Detecção em Múltipla Escala:

```
function faces_merged = MergeDetections(faces)
%Agrupa regiões com overlaps
%Entrada --> faces: Vetor com as coordenadas das regiões com faces
%Saída   --> faces_merged: Vetor com as regiões depois de excluir os
%overlaps

num_faces = size(faces,1);
count = 1;
exclude_ind = 0;
%Se o vetor tiver mais de duas regiões de face, verificar se há overlaps
if (num_faces >=2)
    %Calcula as intersecções dos retângulos
    intersec = rectint(faces, faces);
```

```
    %Varre todas as possíveis combinações dos retângulos
    for i=1:num_faces
        area1 = faces(i,3)^2;
        for j=i+1:num_faces
            area2 = faces(j,3)^2;
            if area1>area2
                ind = j;
                a = area2;
            else
                ind = i;
                a = area1;
            end
            area_i = intersec(i,j);
            %Se a área de intersecção for maior do que 1/3 do menor
            %retângulo, então agrupe as regiões.
            if (area_i > a/3)
                exclude_ind(count) = ind;
                count = count+1;
            end
        end
    end
    if (exclude_ind ~= 0)
        faces_merged = removerows(faces,'ind',exclude_ind);
    else
        faces_merged = faces;
    end
else
    faces_merged = faces;
end

end

function faces_final = Detectar(image,i_scale,step_s,step_x,step_y)
%Varre a imagem em busca dos padrões
%Entrada --> Classificador = Parâmetros das características treinadas
%                    image: Imagem na qual será feita a varredura
%                    step_s: Fator de escala
%                    MHaar: Conjunto de todas as características/máscaras
%                    de Haar
%Saída   --> Coordenas dos retângulos definidos pelas
%            regiões que classificaram positivamente

%Carrega as características treinadas
load('Parametros.mat');
load('MHaar.mat');

Classificador = C;

image = double(image);

%Dimensões da janela de varredura
haar_w=24; haar_h=24;

%Inicializa a saída
```

```
faces_final = [];

%Número de estágios dos classificadores
n_cascatas = size(Classificador,2);

%Incremento de posição da janela de varredura
%step_x = 2;
%step_y = 2;

%Escala inicial
scale=i_scale;
%Escala mínima
min_scale = 0.1;

%Varrer a imagem até que a escala mínima seja atingida.
while scale>=min_scale
    %Redimensionar a imagem
    image_resized = imresize(image, scale);
    w = size(image_resized,2);
    h = size(image_resized,1);
    %Integral da Imagem para todos os pixels
    ii_imr = IntegralImage(image_resized);

    faces = [];

    %Desloca a janela de varredura em incrementos de step_x no eixo x
    %e step_y no eixo y
    for i=2:step_x:w-haar_w
        for j=2:step_y:h-haar_h
            face_found = true;
            for cascata=1:n_cascatas
                %Lê os parâmetros do classificador para o estágio atual
                threshold = Classificador(cascata).threshold;
                alphas = Classificador(cascata).alphas';
                findexes = Classificador(cascata).cfraco(:,1);
                theta = Classificador(cascata).cfraco(:,2);
                p = Classificador(cascata).cfraco(:,3);
                p_m_theta = p.*theta;

                %Busca as máscaras para os classificadores fracos
                fc = zeros(size(findexes,1),5);
                for i2=1:size(findexes,1)
                    fc(i2,:) = MHaar(findexes(i2),:);
                end

                r = 0;
                %Aplica o classificador na região da imagem
                for k=1:size(fc,1)
                    fs =
CalcHaar(ii_imr,fc(k,1)+i,fc(k,2)+j,fc(k,3),fc(k,4),fc(k,5));
                    r = r + alphas(k)*(p(k).*fs<p_m_theta(k));
                end
                %Se o valor computado for menor do que o threshold, a
                %região não classificou positivamente para o estágio
atual
```

```
                %portanto essa região não passará pelos próximos estágios
                if r<threshold
                    face_found = false;
                    break;
                end
            end
            %Se a região classificou positivamente para todos os
estágios,
            %guarda as coordenadas do retângulo da região atual
            if (face_found == true)
                faces = [faces; i,j,haar_w,haar_h];
            end
        end

    end
    %Se uma ou mais regiões foram classificadas positivamente na escala
    %atual, guarda as coordenadas com a devida correção de escala
    if size(faces,1)> 0
        faces_final = [faces_final; faces/scale];
    end
    scale = scale*1/step_s;
end
%Se existirem regiões positivas, agrupar os possíveis overlaps
if isempty(faces_final) == false
    faces_final = MergeDetections(faces_final);
end
```

Apêndice B – Códigos em MATLAB (*LDA*)

Criar:

```
clear all
close all

tic

diretorio_faces = '../banco de imagens/ORL/orl_faces/';

ext = '*.PGM';

%Número de imagens por pessoa
n_imagens_pessoa = 5;

%Cada pessoa tem um diretório separado, com várias imagens
%Todas as Imagens devem ter as mesmas dimensões
temp = dir(diretorio_faces);
isub = [temp(:).isdir];
lista_diretorios = {temp(isub).name}';
lista_diretorios(ismember(lista_diretorios,{'.','..'})) = [];

n_pessoas = size(lista_diretorios,1);

ntotal = n_imagens_pessoa*n_pessoas;

f1 = figure(1);

%Lê as imagens
for i=1:n_pessoas
    lista_imagens =
dir(strcat(diretorio_faces,lista_diretorios{i},'\',ext));
    for j=1:n_imagens_pessoa
        A = imread([diretorio_faces lista_diretorios{i} '\'
lista_imagens(j).name]);
        %Mostra a imagem de cada uma das pessoas cadastradas
        if (j==1)
            subplot(ceil(sqrt(n_pessoas)),ceil(sqrt(n_pessoas)),i)
            imshow(A)
            title(['Pessoa ' int2str(i)],'fontsize',12)
        end

        %Converte para escala de cinza, caso a imagem seja colorida
        if size(A,3)>1
            A = rgb2gray(A);
        end
        [H, W] = size(A);
        A = double(A);
        %Transoforma a imagem em um vetor coluna
        A_vec = reshape(A',W*H,1);
        Faces(:,i,j) = A_vec;
```

```
    end
end
%Ajusta para que a imagem com todas as faces seja exibida em tela cheia
screen_size = get(0, 'ScreenSize');
set(f1, 'Position', [0 0 screen_size(3) screen_size(4) ] );

%Média da face de cada classe
meanFace1 = squeeze(mean(Faces,3));
%Média da face de todas as classes
meanFace2 = mean(meanFace1,2);

%Média geral é subtraída da média de cada classe
M = zeros(W*H,n_pessoas);
for i=1:n_pessoas
    M(:,i) = meanFace1(:,i)-meanFace2;
end

%Média da classe é subtraída de cada imagem de acordo com suas classes
M2 = zeros(W*H,ntotal);
count = 1;
for i=1:n_pessoas
    for j=1:n_imagens_pessoa
        M2(:,count) = squeeze(Faces(:,i,j))-meanFace1(:,i);
        count = count + 1;
    end
end

%Matriz auxiliar
MA = (M'*M)/n_pessoas;

%Autovetores e autovalores de MA (MA*V = V*D)
[V, D] = eig(MA);
%Ordena os autovalores em ordem decrescente
D1 = sort(D,2,'descend');
D1 = D1(:,1);
[B, IX] = sort(D1,'descend');
%Ordena os autovetores de acordo com a ordenação anterior
V = V(:,IX);
%Autovetores de SB (SB = Matriz de dispersão entre classes)
A_SB = M*V;
%Autovalores de SB (Matriz diagonal)
D = D(IX,IX);

%Matriz Auxiliar para calcular SW(Matriz de dispersão dentro da classe)
%indiretamente
Z=A_SB*D^(-0.5);

MA2 = Z'*M2;
[U,Uval] = eig(MA2*MA2');

Wv = Z*U;
for i=1:n_pessoas
    Wv(:,i) = Wv(:,i)/norm(Wv(:,i));
end
```

```
for i=1:n_pessoas
    Caracteristicas(:,i) = Wv'*M(:,i);
end

save('LDA_DB.mat',
'Wv','meanFace2','n_pessoas','W','H','Caracteristicas');

toc
```

Reconhecer:

```
function I = LDA_Testar(img)
I = 0;
if (exist('LDA_DB.mat','file') == 2)
    load('LDA_DB.mat');
    img =  imresize(img, [H W]);
    A = double(img);
    TestImg = reshape(A',W*H,1);
    Caracteristica_Teste = Wv'*(TestImg-meanFace2);

    for i=1:n_pessoas
        distancia(i) = norm(Caracteristica_Teste-
squeeze(Caracteristicas(:,i)))^10;
    end
    [C, Ind] = sort(distancia);
%Critério de aceitação
    alpha = 1 - C(1)/C(2);
    if (alpha > 0.9)
        I = Ind(1);
    end

else
    disp('Banco de dados não encontrado!');
end

end
```

Apêndice C – Códigos em MATLAB (Rotinas de Teste)

Mostra Detecções:

```
function MostrarDetec(image, faces)
%Mostra na imagem de entrada, as regiões classificadas corretamente
%Entrada --> image: Imagem Original
%            faces: Vetor com as coordenadas das regiões com faces
figure;
imshow(image);

k = size(faces,1);

hold on;
for i=1:k
   rectangle('Position', faces(i,:),'LineWidth',3,'EdgeColor', 'w');
end
hold off;
```

Testa Detecção em uma imagem:

```
close all

tic

im = imread('../banco de
imagens\umass\originalPics\2003\01\31\big\img_355.jpg');

%Todas as possíveis características
if (exist('MHaar.mat','file') ~= 2)
    MHaar = MascaraHaar(img_w,img_h);
    save('MHaar.mat', 'MHaar');
    disp('Características Haar computadas');
else
    load('MHaar.mat');
    disp('Características Haar carregadas');
end

%Carrega as características treinadas
load('Param_final2.mat');

im2 = im;

if size(im2,3)>1
        im2 = rgb2gray(im2);
end

[H,W] = size(im2);

faces_rect = Detectar(im2,1, 0.97,2,2);
```

```
toc
MostrarDetec(im, faces_rect);
```

Testa o *LDA*:

```
%Testar
tic
diretorio_faces = '../banco de imagens/ORL/orl_faces/';

ext = '*.PGM';

n_imagens_pessoa = 5;
n_pessoas = 40;

temp = dir(diretorio_faces);
isub = [temp(:).isdir];
lista_diretorios = {temp(isub).name}';
lista_diretorios(ismember(lista_diretorios,{'.','..'})) = [];

fp = 0; %classificações incorretas
nr = 0; % rejeições
c = 0; %classificações corretas

for i=1:n_pessoas
        lista_imagens =
dir(strcat(diretorio_faces,lista_diretorios{i},'\',ext));
        for j=n_imagens_pessoa+1:size(lista_imagens,1)
            A = imread([diretorio_faces lista_diretorios{i} '\'
lista_imagens(j).name]);
            Index = LDA_Testar(A);
            if (Index == 0)
                nr = nr + 1;
            else
                if Index ~=i
                   fp = fp + 1;
                else
                    c = c+1;
                end
            end
        end
    end
toc
```