UNIVERSIDADE FEDERAL FLUMINENSE

ESCOLA DE ENGENHARIA

MESTRADO EM ENGENHARIA DE TELECOMUNICAÇÕES

FERNANDO OTÁVIO GOMES DA FONSECA

DETECTOR DE FACES UTILIZANDO FILTROS DE CARACTERÍSTICAS

Niterói
2016

FERNANDO OTÁVIO GOMES DA FONSECA

DETECTOR DE FACES UTILIZANDO FILTROS DE CARACTERÍSTICAS

Dissertação de Mestrado apresentada ao Programa de Pós Graduação em Engenharia Elétrica e de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para a obtenção do Grau de Mestre em Engenharia Elétrica e de Telecomunicações. Área de Concentração: Sistemas de Telecomunicações.

Orientador: Prof. MURILO BRESCIANI DE CARVALHO

Niterói
Março/2016

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

F676 Fonseca, Fernando Otávio Gomes da
Detector de faces utilizando filtros de características / Fernando Otávio Gomes da Fonseca. – Niterói, RJ : [s.n.], 2016.
111 f.

Dissertação (Mestrado em Engenharia Elétrica e de Telecomunicações) - Universidade Federal Fluminense, 2016.
Orientador: Murilo Bresciani de Carvalho.

1. Processamento digital de imagem. 2. Detecção de face. 3. Aprendizado de máquina. 4. Inteligência artificial. 5. Sistema de telecomunicação. I. Título.

CDD 006.42

Fernando Otávio Gomes da Fonseca

"Detector de Faces utilizando Filtros de Características"

Dissertação de Mestrado apresentada ao Programa de Pós Graduação em Engenharia Elétrica e de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para a obtenção do Grau de Mestre em Engenharia Elétrica e de Telecomunicações. Área de Concentração: Sistemas de Telecomunicações.

BANCA EXAMINADORA

Murilo B. de Carvalho

Prof. MURILO BRESCIANI DE CARVALHO - Orientador, D.Sc. UFF

Carla Liberal Pagliari

Profª. Carla Liberal Pagliari, Ph.D., IME

Tadeu Nagashima Ferreira

Prof. Tadeu Nagashima Ferreira, D.Sc., UFF

Niterói
Março/2016

Dedicatória

Dedico este trabalho primeiramente a Deus, que me ilumina em todos os momentos, a minha mãe Izaura e meu pai Fernando, que estão sempre em meu coração, e a minha esposa Sandra, fiel companheira e amor da minha vida.

Agradecimentos

Agradeço a Deus pela vida e família que me presenteou.

Agradeço a minha mãe Izaura e meu pai Fernando pela formação de caráter ético e fraterno que me proporcionaram.

Agradeço a minha esposa Sandra pelo carinho, paciência e estímulo para toda a minha vida.

Agradeço ao meu orientador prof. Murilo B. de Carvalho por acreditar no meu trabalho, mesmo depois de muitos anos distante da sala de aula.

Agradeço aos professores, funcionários e colegas da UFF por terem me acolhido de volta a Universidade com gentileza e sorrisos sinceros.

Agradeço às pessoas que passaram pela minha vida e que mesmo sem saber, contribuíram para o meu crescimento pessoal, profissional e principalmente como Ser Humano.

Resumo

O presente trabalho visa estudar e comparar 2 métodos de detecção de faces em imagens, a fim de averiguar a eficiência e eficácia dos mesmos, propondo melhorias nos processos avaliados. O método de detecção de caraterísticas em imagens proposto por Viola e Jones é ainda uma referência na detecção de faces. Neste trabalho serão avaliadas propostas de melhorias nesse processo e comparados resultados quando utilizadas redes neurais mais modernas para o treinamento da base de dados. Realizamos simulações computacionais desenvolvidas em Matlab para obtenção dos resultados do comportamento dos sistemas e ao final do trabalho apresentamos as conclusões e sugestões de projetos futuros.

Palavras chave: Aprendizado de Máquina, Detecção de Faces, Detecção de Características.

Abstract

This work aims to study and compare two methods of face detection in images, in order to verify their efficiency and effectiveness, proposing improvements in such processes. The feature detection method in images proposed by Viola and Jones is also a reference in detecting faces. In this work improvement proposals will be evaluated in that process and compared results when used more modern neural networks for the training database. We performed computer simulations developed in Matlab to obtain the results on systems behavior. At the end of the work, we present the conclusions and suggestions for future projects.

Keywords: Machine Learning, Face Detection, Feature Detection.

Sumário

Índice de Ilustrações

Capítulo I

1. Introdução

A obtenção instantânea de informações a partir das imagens é um dos desafios na área de processamento digital de imagens. Esse interesse é ainda maior quando se trata da identificação automática de faces, e demanda por diversos estudos e pesquisas nas áreas acadêmicas e comerciais.

Diversas aplicações demandam estudos para detecção da imagem da face, tais como biometria, vigilância, segurança e entretenimento. Como exemplo, a biometria consiste na aplicação de métodos matemáticos e de estatística quantitativa a fatores biológicos. A vantagem da utilização da face como meio de identificação biométrica, em comparação ao reconhecimento da íris ou das digitais, é que esse método é menos invasivo, não sendo necessários equipamentos muito sofisticados, além de câmeras digitais, para a obtenção da imagem.

Imagens digitais são a representação da imagem real após o processo de aquisição, que transforma a cena real em uma imagem analógica, e do processo de digitalização, que é composto por uma amostragem que define a distribuição espacial dos pixels, e por uma quantização que define os níveis de amplitude da intensidade do sinal (vide Figura 1).

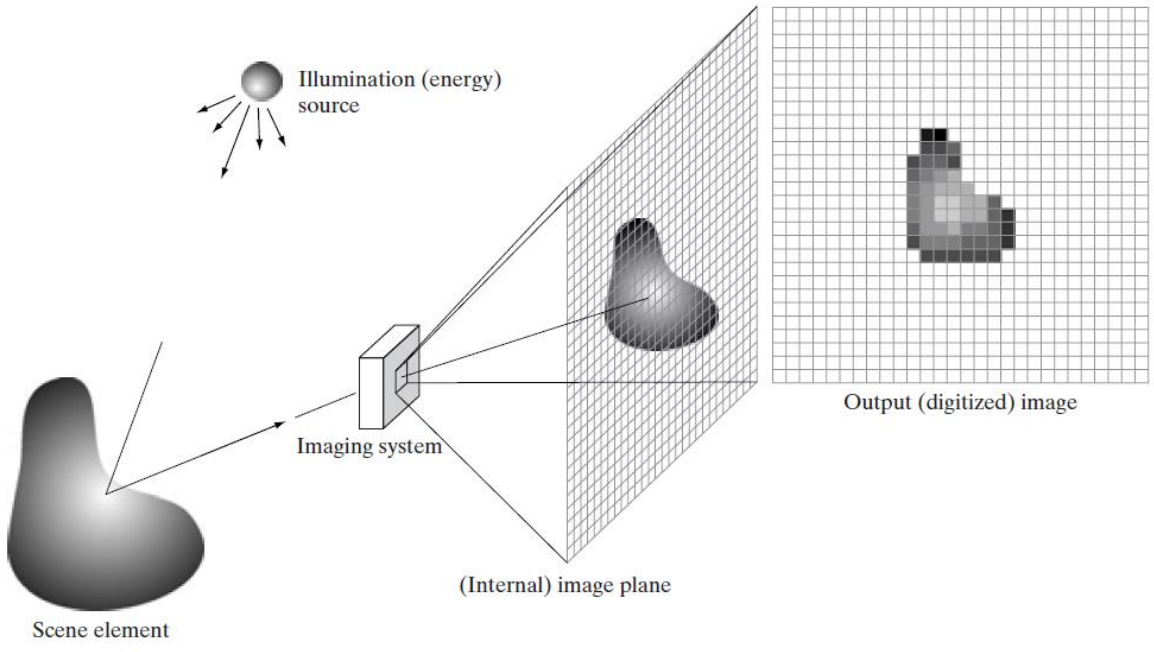

Figura 1 - Exemplo de aquisição e digitalização de uma imagem [14].

A resolução das imagens é caracterizada por dois parâmetros independentes: a resolução espectral e a resolução espacial. A resolução espectral está relacionada ao número de amostras tomadas no espectro. A resolução espacial está relacionada ao número de linhas e colunas que foram amostradas na aquisição da imagem.

Neste estudo, trabalharemos com imagens monocromáticas, que são representadas pela intensidade luminosa em qualquer ponto de suas coordenadas espaciais, uma dimensão espectral e os valores dos pixels representam 256 níveis de intensidade de cinza no intervalo [0, 255] para uma codificação de 8 bits.

Um sistema de detecção de faces pretende reproduzir uma das capacidades naturais dos seres humanos: reconhecer as características de uma face em diferentes ambientes e associá-las a informações já armazenadas na memória. Entretanto, o que é natural nos seres humanos é um desafio complexo para os computadores, pois existem diferentes fatores, tais como: iluminação, sombras, oclusões, distância e posição da câmera, que interferem no reconhecimento das imagens.

Na literatura encontramos diversas abordagens empregadas na detecção de faces. Zafeiriou et al [1] examinam os recentes avanços nas técnicas de detecção de face, e categorizam dois esquemas gerais: modelos rígidos e modelos deformáveis. Nos modelos rígidos são utilizadas máquinas de aprendizado sobre o rosto completo, enquanto que os modelos deformáveis descrevem o rosto por suas partes (olhos, boca, nariz e etc) e buscam esses padrões nas imagens em detecção.

Também Stan e Anil [2] definem duas abordagens básicas para a identificação da face. A primeira utiliza a extração de vetores característicos de partes básicas de uma face, tais como: olhos, nariz, boca e queixo. A segunda abordagem utiliza conceitos da teoria da informação, e descreve uma face a partir de informações de toda imagem da face.

Em nosso trabalho utilizaremos modelos rígidos que buscam avaliar o rosto completo, mas utilizaremos filtros de características para identificar partes básicas de uma face, tais como as regiões dos olhos, nariz e boca.

Conforme o tipo de aplicação, a eficiência do sistema terá diferentes requisitos. Por exemplo, em um sistema que analisa vídeos de crimes capturados por câmeras de vigilância, o tempo de execução do sistema não é crucial, mas a precisão do resultado é muito importante, ou seja, um tempo de resposta mais prolongado é aceitável, desde que o resultado seja eficaz. Em contrapartida, o resultado em tempo real é essencial em sistemas de controle de acesso, pois há um usuário na outra ponta a espera de uma resposta.

Um sistema computacional de reconhecimento de face busca responder a algumas das seguintes questões: A imagem na entrada possui uma face? As faces detectadas na imagem são novas ou já são “conhecidas” pelo sistema? De quem são as faces que aparecem na imagem? A imagem de entrada contém a face esperada pelo sistema?

A entrada de um sistema de detecção de face é uma imagem com uma ou mais faces, e, a saída, a(s) face(s) detectada(s) e/ou sua identificação / verificação a partir de um banco de dados. Dessa forma, os diferentes parâmetros e requisitos de cada sistema geram restrições que irão definir a sua forma de implementação.

O presente trabalho foi planejado com dois objetivos principais: pesquisa e melhoria do método de detecção de faces proposto por Viola-Jones [3]; e pesquisa e desenvolvimento de um método de detecção de faces utilizando filtros na entrada de uma rede neural baseada na máquina SVM.

Esta tese está organizada da seguinte maneira: no capítulo 2, detalhamos os fundamentos do método de detecção de faces proposto por Viola-Jones; no capítulo 3 apresentamos uma breve introdução às máquinas de aprendizado como base para o detalhamento da teoria das máquinas de vetor de suporte (SVM). No capítulo 4, mostramos as implementações e testes realizados no método de detecção de faces Viola-Jones, as modificações propostas e os resultados das simulações de detecção com a utilização do SVM. No capítulo 5 encontram-se as conclusões e sugestões para trabalhos futuros.

Capítulo II

2. Detecção de faces com o método Viola-Jones

2.1. O Método Viola-Jones

Em 2001, Paul Viola e Michael Jones [3] propuseram um método de detecção de objetos com alta taxa de precisão e baixo custo computacional em comparação com os métodos da época. Mesmo com o passar dos anos o método de detecção de Viola e Jones ainda tem sido muito utilizado em estudos recentes.

O método considera três componentes básicos: a integral de imagem para calcular filtros de características de Haar, um classificador simples responsável por identificar as características e limiares de detecção mais relevantes, e uma combinação de estágios desses classificadores para permitir o ajuste de desempenho e resultados adequados.

Embora o algoritmo possa ser treinado para reconhecer qualquer objeto, a motivação principal da abordagem de Viola e Jones [3] foi o reconhecimento facial. O ponto forte deste algoritmo é a rapidez com que é executado, embora envolva um elevado custo computacional na fase de treinamento.

Em contrapartida o método de detecção de faces de Viola-Jones [3] possui as seguintes deficiências observadas desde a sua publicação: é pouco confiável em rostos com inclinação ou de perfil, com oclusões em partes do rosto e em situações especiais de iluminação da imagem em que o rosto é mais escuro que o fundo.

2.2. Utilização de Características

O método de detecção de faces de Viola-Jones [3] classifica as imagens com base nos valores de características simples.

A principal razão para utilizar características e não pixels diretamente é que as características podem agir para codificar o conhecimento do contexto e da finalidade, que é difícil de aprender quando usamos uma quantidade finita de dados de treinamento. Há também uma segunda motivação crítica para o uso das características: o sistema opera muito mais rápido que um sistema baseado em pixels.

As características usadas por Viola-Jones [3] são uma reminiscência de funções de base Haar, que foram utilizados por Papageorgiou *et al.* [4] em 1998. Foram usados três tipos de características (veja a Figura 2): duas características com dois retângulos

mostradas em (A) e (B), uma característica de três retângulos mostrada em (C), e uma característica de quatro retângulos mostrada em (D).

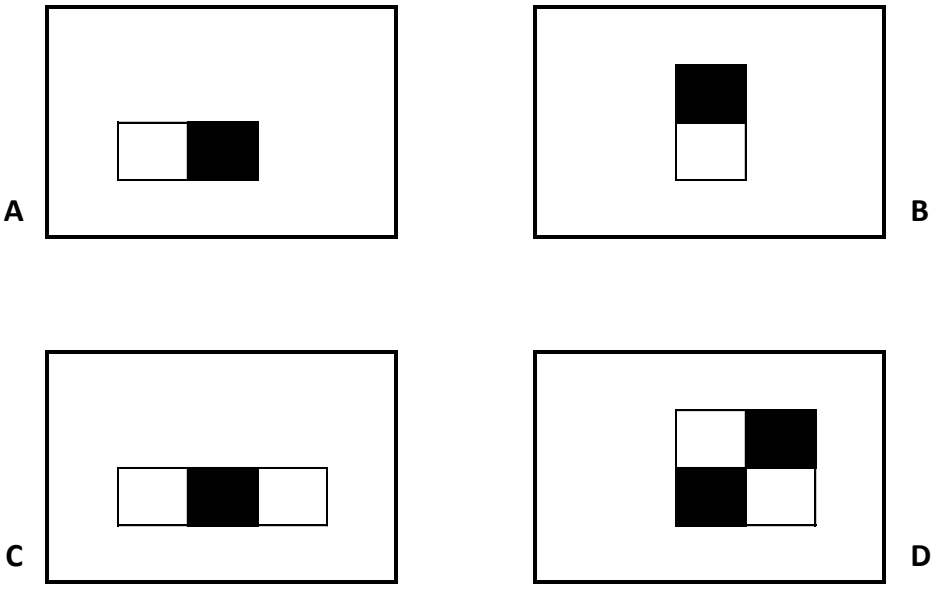

Figura 2 - Características retangulares Haar like utilizadas

As regiões retangulares têm o mesmo tamanho e forma e são horizontalmente ou verticalmente adjacentes. O valor de uma característica é calculado pela soma dos pixels que se encontram dentro dos retângulos brancos e subtraídos da soma de pixels nos retângulos pretos. (Figura 2).

Cada característica pode ajudar a reconhecer determinados padrões, principalmente quando combinados em sequência. Por exemplo, a característica B da Figura 2 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 a região dos olhos é mais escura do que a região da face em torno do nariz (Figura 3).

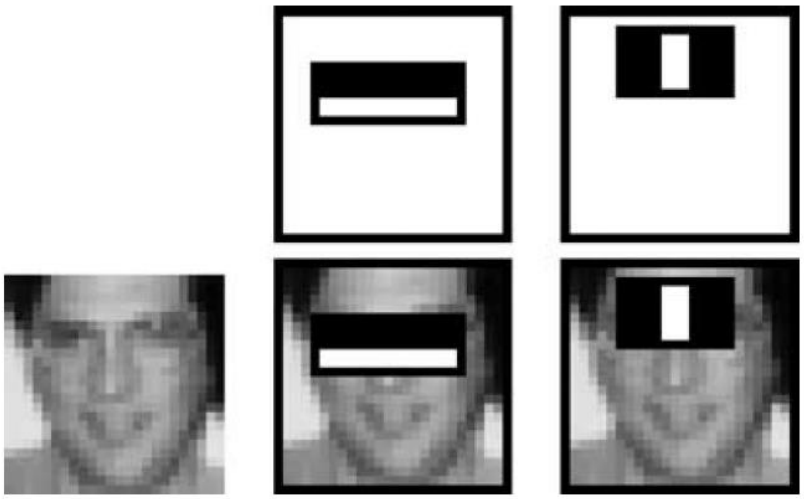

Figura 3 - Exemplo de aplicação de um característica como elemento da face

Duas características Haar são mostradas na linha superior da Figura 3 e, em seguida, cobrem um rosto típico de treinamento na linha de baixo. A primeira característica mede a diferença na intensidade entre a região dos olhos e a região superior das bochechas. Essa característica aproveita a observação de que a região do olho é frequentemente mais escura do que a das faces. A segunda característica compara as intensidades nas regiões dos olhos, mais escuras, contra a intensidade da parte superior do nariz, mais clara.

Existem, é claro, 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. Com isso, são possíveis mais de 180.000 características Haar-like distintas, se considerarmos os quatro modelos propostos por Viola-Jones [3]; e as diferentes posições e tamanhos de cada modelo dentro da janela. A Figura 4 exemplifica como isso ocorre.

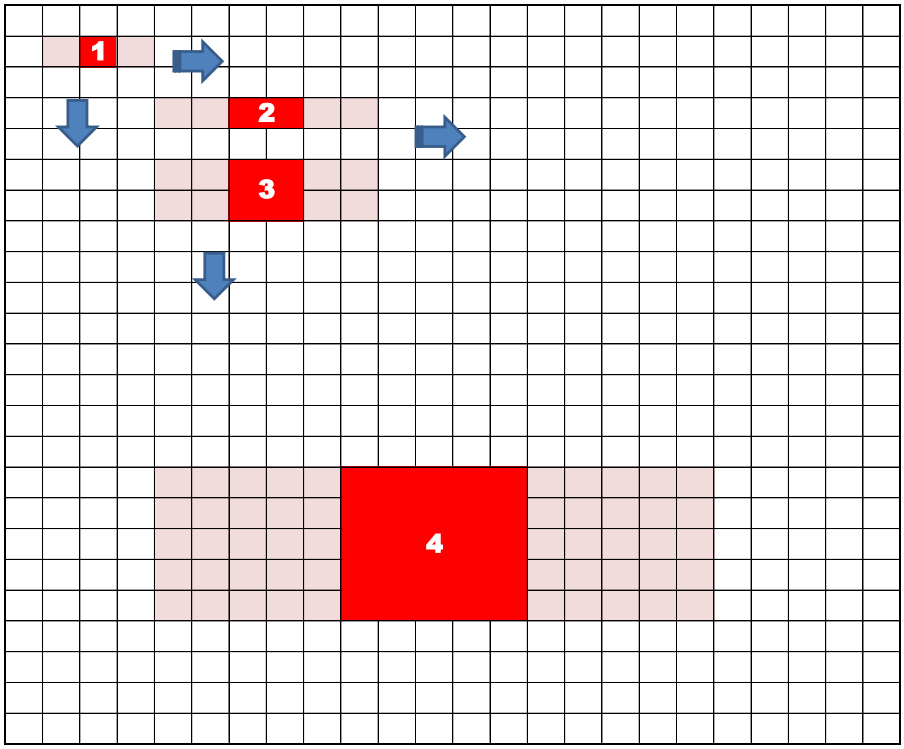

Figura 4 - Modelo Haar dentro de uma janela de 24x24 pixels

Na Figura 4 vemos uma característica do tipo A, conforme identificada na Figura 2, em tamanho mínimo, 1 pixel de altura e 3 pixels de comprimento, identificada no diagrama com o número 1. Essa característica pode deslizar pixel a pixel da esquerda para direita e de cima para baixo até passar por todos os pixels da janela de 24x24. Se a dimensão completa da janela for utilizada teremos 22 vezes 24, ou 528 possíveis posições para a característica de número 1 ocupar dentro da janela.

Entretanto, a característica do tipo A pode ser aumentada na largura, na altura, ou em ambas, como podemos ver nos exemplos numerados como 2, 3 e 4 da Figura 4. Somadas todas a possibilidades chegamos no total de 180.000 características possíveis numa janela base de 24x24 pixels.

2.3. Integral de Imagem

As características retangulares podem ser computadas muito rapidamente usando uma representação intermediária para a imagem que nós chamamos Integral da Imagem.

A integral de imagem, também conhecida como tabela de soma de áreas, é utilizada em um algoritmo, proposto por Frank Crow [5] em 1984, que permite avaliar eficientemente a soma dos valores dos pixels (intensidade dos níveis de cinza) de uma região retangular da imagem. A equação (1) define o valor da integral de imagem na posição (x y).

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

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 na Figura 5, 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).

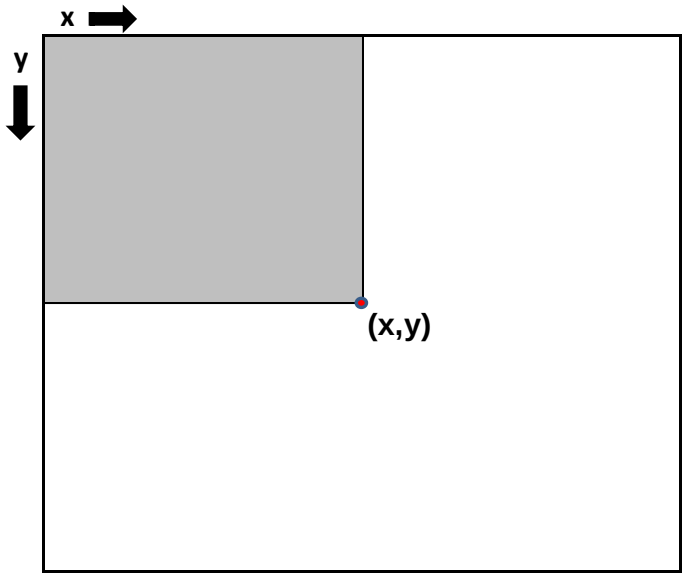

Figura 5 - O valor da Integral da Imagem num ponto (x,y)

Usando a Integral da Imagem qualquer soma retangular pode ser calculada em uma matriz de quatro referências (ver Figura 6).

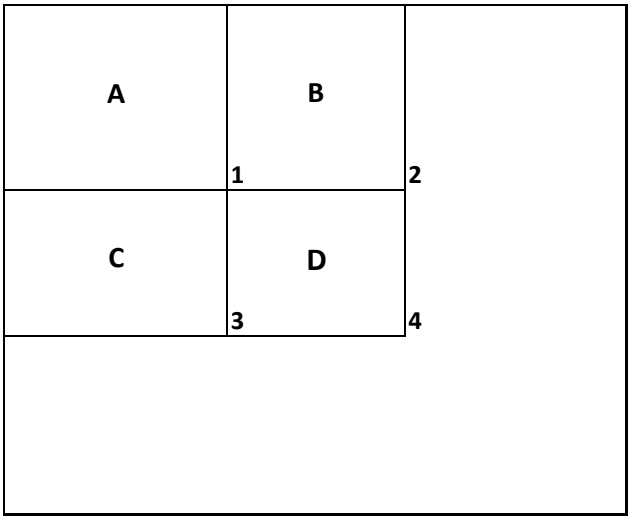

Figura 6 - Soma de valores dos pixels na região D.

Pode-se verificar que a soma dos pixels dentro retângulo D pode ser calculada com quatro referências de matriz, que são os vértices do retângulo formado por 1234. O valor da Integral da Imagem na localização 1 é a soma dos pixels no retângulo A. O valor na localização 2 é A + B, no local 3 é A + C, e na posição 4 é A + B + C + D. Logo, a soma dentro de D pode ser calculada como:

$$\sum_{(x,y)\in 1234} i(x,y) = ii(x_1,y_1) + ii(x_4,y_4) - ii(x_2,y_2) - ii(x_3,y_3) \qquad (2)$$

Dessa forma, Viola-Jones [3] propuseram utilizar a integral da imagem para facilitar o cálculo da soma dos valores dos pixels em qualquer região retangular de uma imagem, e com isso identificar os padrões das características Haar-like, que são utilizadas no treinamento dos classificadores.

2.4. Classificador

O segundo passo no algoritmo de Viola-Jones [3] é o treinamento de classificadores. Dado um conjunto de características deve-se treinar o sistema com imagens positivas (faces) e imagens negativas (sem faces). Deve-se usar um algoritmo de treinamento que aprenda funções de classificação. A opção proposta por Viola-Jones [3] é utilizar um algoritmo de aprendizagem que use o método Adaboost, que consiste em encontrar um

classificador de alta precisão agregando-se muitos classificadores denominados “fracos”, pois cada um deles possui uma precisão média com uma taxa de acertos próxima de 51%.

Dentre os modelos de aprendizado de máquina supervisionado, aparecem os comitês de máquinas [6]. Um comitê de máquinas representa a combinação de mais de uma máquina de aprendizado na produção de uma solução computacional única para um determinado problema.

O boosting ou reforço, proposto por Shapire e Freund [7], funciona em iterações. A cada etapa, um classificador simples utiliza uma diferente versão do conjunto de amostras de treinamento. As diferentes versões do conjunto de treinamento são obtidas através da variação do peso associado a cada um dos exemplos.

Segundo esta proposta, os conjuntos de treinamento são gerados considerando a contribuição dos mesmos para o erro de treinamento dos componentes já treinados, isto é, caso uma amostra não tenha sido corretamente classificada pelos componentes anteriores, a probabilidade de escolha desta aumenta em relação às demais amostras. Dessa forma, esta amostra terá uma chance maior de ser escolhida para compor o conjunto de dados do próximo componente a ser gerado.

Após um número de iterações, o boosting combina os diversos classificadores parciais, gerando um classificador único, que, possivelmente, possui um desempenho melhor que o dos classificadores parciais.

O AdaBoost (Adaptative Boosting) é um algoritmo, baseado em Boosting, que tem como premissa construir um classificador "forte" como uma combinação linear de vários classificadores fracos. Os diversos classificadores, que compõem o seu comitê, são gerados sequencialmente, favorecendo as amostras do conjunto de treinamento classificados incorretamente pelos classificadores anteriores.

A equação a seguir expressa essa ideia:

$$f(x) = \sum_{t=1}^{T} a_t h_t(x) \qquad (3)$$

A função $h_t(x)$ representa classificadores fracos, e pode assumir valores 0 ou 1 (-1, 1 em determinados casos), 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:

$$h(x, f, p, \emptyset) = \begin{cases} 1, & se\ pf(x) < p\emptyset \\ 0, & caso\ contrário \end{cases} \qquad (4)$$

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.

2.5. Cascata de classificadores

A última etapa consiste em combinar classificadores fortes em estágios, 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 fundo de planos sejam descartados nos primeiros estágios, e apenas faces e outros objetos semelhantes a faces sejam analisados mais exaustivamente (Figura 7).

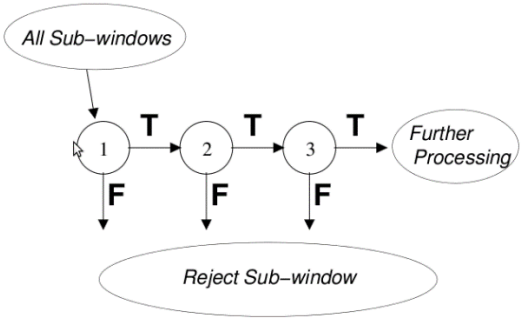

Figura 7 - Cascata de Classificadores Viola-Jones

A quantidade de estágios de uma cascata está relacionada com a taxa de detecção correta de objetos e o registro de falsos positivos. De acordo com a teoria do Adaboost, a taxa de detecção de uma cascata de classificadores é:

$$D = \prod_{i=1}^{K} d_i \qquad (5)$$

Onde D é a taxa detecção da cascata de classificadores, K é o número de estágios e d_i é a taxa de detecção de cada estágio. Do mesmo modo, a taxa de falsos positivos é dada por:

$$F = \prod_{i=1}^{K} f_i \qquad (6)$$

Onde F é a taxa falsos positivos da cascata de classificadores, K é o número de estágios e f_i é a taxa de falsos positivos de cada estágio.

2.6. Detector em Múltipla Escala

Após o treinamento dos classificadores, é possível implementar uma busca na imagem de entrada dos padrões desejados e que permita identificar as regiões classificadas corretamente como faces.

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. Portanto é necessário redimensionar a imagem ou a janela de detecção iterativamente, até que todas as possíveis regiões da imagem de entrada sejam testadas em busca de uma face.

Capítulo III

3. Aprendizado de Máquina e SVM

O Aprendizado de Máquina tem sido utilizado com sucesso em muitos problemas envolvendo o reconhecimento de padrões. Como o reconhecimento facial pode ser tratado como um problema de classificação binária, diversos tipos de redes neurais têm sido propostos com essa finalidade.

As redes neurais nasceram do estudo realizado por McCulloch e Pitts [8] em 1943, que unificaram estudos de neurofisiologia e da lógica matemática para mostrar que uma rede de neurônios com conexões ajustadas poderia realizar a computação de qualquer função computável [6]. Com o desenvolvimento das pesquisas sobre o aprendizado das máquinas, percebeu-se, nas décadas seguintes, que poderiam fazer uso desse conhecimento para desenvolver algoritmos computacionais mais eficientes para tratar questões nas áreas de:

- Prognóstico de mercados financeiros;
- Reconhecimento óptico de caracteres (OCR)
- Controle de processos;
- Previsões climáticas,
- Identificação de fraudes nos sistemas financeiros.
- Sistema de conhecimento de sintomas e diagnósticos médicos
- Análise e processamento de sinais (voz, imagens e vídeo)
- Robótica;
- Classificação e Reconhecimento de Padrões;

Redes neurais artificiais ou máquinas de aprendizagem são processadores capazes de aprender através da experiência e, então, utilizar o conhecimento adquirido em situações novas no mesmo escopo de sua aprendizagem. As redes neurais foram criadas com base no modelo de aprendizado humano, que inclui os neurônios e as suas transmissões sinápticas, bem como as propriedades de plasticidade e adaptabilidade [6].

O modelo de uma rede neural artificial é composto por neurônios (unidades de processamento), pesos sinápticos (fatores de multiplicação dos sinais de entrada), somadores para reunir os sinais de entrada e as funções de ativação, que restringem a amplitude do sinal de saída [6].

Um agrupamento é uma coleção de objetos que são similares uns aos outros (de acordo com algum critério de similaridade pré-definido) e dissimilares a objetos pertencentes a outros grupos.

O ser humano se depara com processos de agrupamento a todo o momento, seja no momento de arrumar o seu guarda-roupa, quando cria um círculo de amizades, ou quando torce por um time de futebol. Para o cérebro humano é bastante simples realizar tais agrupamentos, mas para que uma máquina realize tal classificação são necessárias diversas etapas, tais como:

- Seleção de atributos
- Definição dos critérios de agrupamento
- Utilização de um método de agrupamento
- Verificação e Interpretação dos resultados

As redes neurais são classificadas e divididas em máquinas supervisionadas e sem supervisão.

No aprendizado supervisionado, todos os exemplos de treinamento são rotulados, isto é, para cada entrada no treinamento da rede neural existe uma saída correspondente. Durante o treinamento, a máquina “aprende” com a repetição de entradas e saídas a responder numa situação futura sem a rotulação.

No entanto, existem situações em que não é possível rotular previamente os dados recebidos, seja devido ao volume dos dados que exigiria um custo muito elevado para identificar todos os dados, ou simplesmente por que, inicialmente, não sabemos como classificar esses conjuntos de dados.

O principal objetivo do aprendizado não supervisionado é identificar a organização dos padrões existentes nos dados através de agrupamentos consistentes. Com isso, é possível definir semelhanças e diferenças entre os padrões, e até tirar conclusões sobre os mesmos.

A seguir vamos estudar com mais detalhes alguns métodos utilizados na classificação de objetos, baseados em máquinas de aprendizado supervisionado.

3.1. Perceptron Multicamadas (MLP)

3.1.1. Rede Neural

O cérebro humano é um poderoso sistema de neurônios interligados que pode resolver uma grande variedade de problemas relacionados ao pensar, falar, relembrar, sentir e aprender, e que motiva muitos cientistas para tentar modelar sua operação. O cérebro é um sistema de processamento de informação altamente complexo, não-linear fazendo cálculos em paralelo.

As redes neurais artificiais tentam simular o cérebro humano, modelando a maneira como o cérebro realiza uma tarefa particular ou função de interesse. As redes neurais empregam uma interligação de células computacionais simples denominadas de "neurônios" ou unidades de processamento. Haykin [6] oferece a seguinte definição de uma rede neural:

Uma rede neural é um processador maciçamente paralelo e distribuído, de unidades de processamento simples, que têm a propensão natural para armazenar conhecimento experimental e torná-lo disponível para o uso. Ela se assemelha ao cérebro em dois aspectos:

1. O conhecimento é adquirido pela rede, a partir de seu ambiente, através de um processo de aprendizagem.

2. Forças de conexão entre neurônios conhecidas como pesos sinápticos, são utilizadas para armazenar o conhecimento adquirido.

O processo de aprendizagem é chamado de algoritmo de aprendizagem, e sua função é, através de sucessivas iterações, modificar os pesos sinápticos da rede de forma a alcançar um objetivo de projeto desejado com o menor erro possível estipulado.

3.1.2. Modelo de um Neurônio

No cérebro, um neurônio é uma unidade de processamento de informação que é fundamental para a operação de uma rede neural. Cada neurônio recebe e combina sinais de muitos outros neurônios e produz sinais para o axônio conduzir certas ações, chamados de sinapses. Juntos, os neurônios formam uma grande rede, denominada de rede neural [6].

Os neurônios recebem continuamente impulsos nas sinapses de seus dendritos vindos de milhares de outras células. Os impulsos geram ondas de corrente elétrica (excitatória ou inibitória) através do corpo da célula até a uma zona chamada a zona de disparo, no começo do axônio. Os processos nos neurônios conduzem a impulsos nervosos

(reações físico-químicas) para o corpo e do corpo para a célula nervosa. O diagrama de um neurônio fisiológico é apresentado na Figura 8.

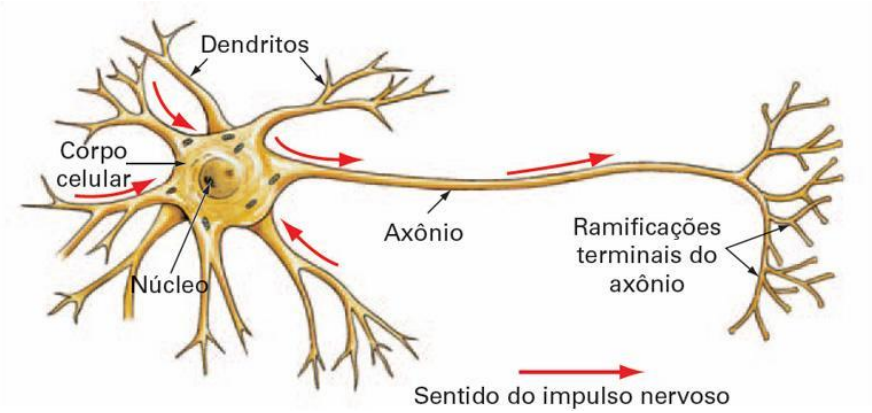

Figura 8 - Estrutura de um neurônio fisiológico [62]

O modelo de neurônio usado pelas redes neurais artificiais está baseado na concepção do neurônio biológico apresentada acima: ele é uma unidade que recebe muitas entradas, integra-as segundo alguma regra ou somador e fornece uma saída que é dada por uma função de ativação (vide Figura 9).

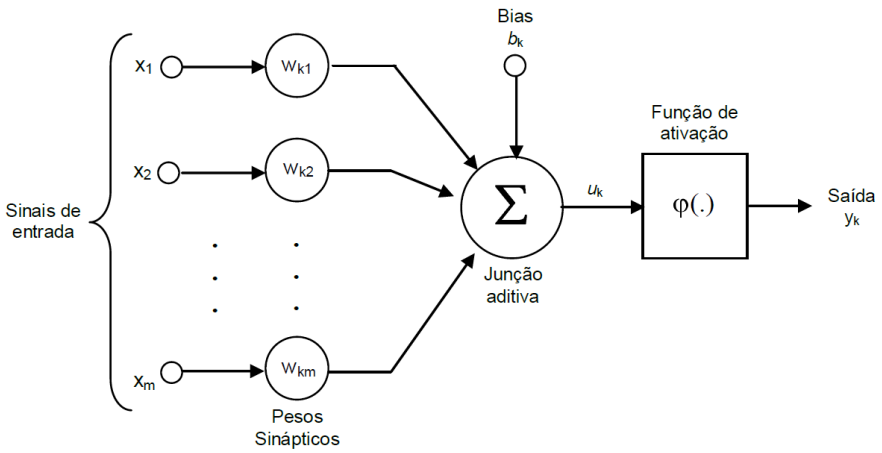

Figura 9 - Modelo de um neurônio artificial [6]

No modelo proposto, o neurônio é formado por um conjunto de sinais de entrada (as sinapses), cada uma associada a um peso individual. Dessa forma, cada sinal X_i na entrada da sinapse conectada ao neurônio é multiplicado pelo peso sináptico Wk_i. O somador combina os sinais de entrada, ponderados pelos respectivos pesos em cada neurônio, na forma de um combinador linear.

A função de ativação é usada para limitar a amplitude do sinal de saída, e combinada com o 'bias' aplicado faz com que o sinal seja aumentado ou diminuído. A função de ativação, denotada por $\emptyset(\ldots)$, define o valor de saída de um neurônio em termos do nível de atividade de sua entrada. Pode-se identificar três tipos básicos de funções de ativação: função de limiar, função linear por partes e função sigmoide [6]. Segue a descrição matemática do neurônio:

$$Y_k = \emptyset(U_k + b_k) \qquad (7)$$

Onde:

$$U_k = \sum_{i=1}^{n} Wk_i X_i \qquad (8)$$

3.1.3. O perceptron MLP

O perceptron multicamadas MLP (multilayer perceptron) é uma rede do tipo feedforward, e caracterizado por:

- Uma ou mais camadas ocultas;
- Neurônios da camada oculta com função de ativação não-linear;
- Elevado grau de conectividade.

A rede perceptron multicamadas consiste de uma camada de entrada (sinapses), uma ou mais camadas ocultas e uma camada de saída como pode ser visto na Figura 10.

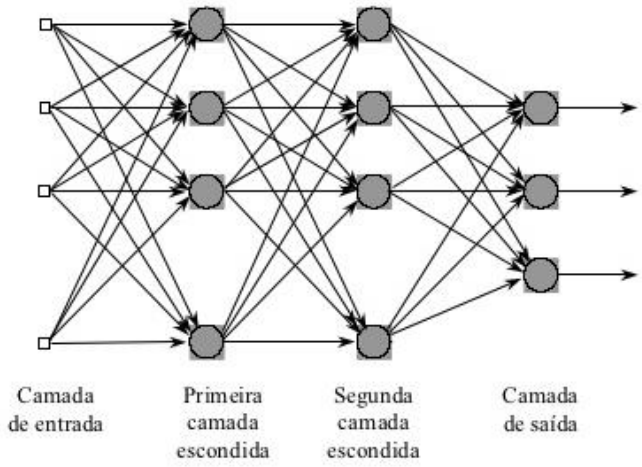

Figura 10 - Modelo do perceptron MLP

O treinamento do MLP é realizado de forma supervisionada com o algoritmo de retro propagação de erro. Este algoritmo baseia-se na regra de aprendizagem por correção de erro [6].

O sinal de entrada propaga-se para frente através da rede, camada por camada, até o neurônio de saída. O neurônio de saída gera um sinal de erro que se propaga para trás (camada por camada) através da rede.

Como característica da rede MLP, os neurônios das camadas intermediárias (ocultas) e, em alguns casos, os da camada de saída possuem uma função de ativação não-linear do tipo sigmoidal (função logística ou tangente hiperbólica).

3.1.4. Treinamento backpropagation

A rede MLP utiliza para o seu treinamento o algoritmo de retro propagação (backpropagation). Este algoritmo é constituído por dois passos principais: o passo para frente (forward) do sinal de entrada; e o passo trás (backward) do erro sinalizado pela saída. Durante o processo de propagação, os pesos sinápticos da rede não se alteram. No passo backward os pesos sinápticos são todos ajustados de acordo com a regra de correção de erro. Os pesos sinápticos da rede são atualizados para que a resposta gerada pela rede aproxime-se da resposta desejada [6].

O processo é repetido diversas vezes durante o treinamento, até que para todas as saídas e padrões de treinamento, o erro seja menor do que o especificado. O principal objetivo do processo de treinamento é minimizar o sinal de erro entre a resposta desejada e a resposta atual produzida pela rede.

Algoritmo de retro propagação do erro pode ser expresso matematicamente:

- Regra de atualização – descida em gradiente:

$$\underline{W}(k+1) = \underline{W}(k) - n\Delta\underline{W}(k) \qquad (9)$$

- Treinamento sequencial

$$E_{med}(\underline{W}) = \frac{1}{N}\sum_{k=1}^{N} E(k) \qquad (10)$$

onde:

$$E(k) = \frac{1}{2}\sum_{j=1}^{m}\left[e_j(k)\right]^2 \qquad (11)$$

3.2. Máquinas de Vetor de Suporte (SVM)

Os fundamentos de SVM são provenientes da Teoria de Aprendizagem Estatística desenvolvida inicialmente pelo pesquisador russo Vladmir Vapnik [9]. Vapnik idealizou o princípio indutivo de Minimização do Risco Estrutural. Este princípio busca minimizar o erro do conjunto de treinamento (risco empírico), juntamente com o erro do conjunto de teste.

O treinamento de SVM envolve a otimização de uma função quadrática convexa, que é um problema de Otimização Matemática. O SVM considera poucos parâmetros livres que precisam ser ajustados pelo usuário e não há uma dependência explícita da dimensão do espaço de entrada do problema. Assim, o SVM pode ser útil em problemas com um grande número de entradas.

O SVM pode ser aplicado ao Reconhecimento de Padrões, Regressão, Extração de Características, e especialmente na Classificação Binária. Num contexto de classificação binária por exemplo a ideia principal da SVM é construir um hiperplano como superfície de separação ótima entre exemplos positivos e exemplos negativos [6].

3.2.1. Otimização Matemática

O treinamento de SVM envolve a resolução de um problema de Otimização Matemática [10]. A compreensão dessa teoria é importante no funcionamento do processo de aprendizagem do algoritmo SVM.

A Otimização é um ramo da Matemática que envolve a solução de classes de problemas compostos por funções que devem ser escolhidas para minimizar ou maximizar uma certa função custo, sujeita a certas restrições.

Os problemas de Otimização são definidos por um conjunto de variáveis ou parâmetros independentes e incluem condições ou restrições que definem valores aceitáveis das variáveis. A solução de um problema de otimização é um conjunto de valores atribuídos a estas variáveis que satisfaz as restrições e minimiza/maximiza a função de custo

Os problemas de otimização podem ser divididos em Programação Linear e Programação Não-linear. A seguir são apresentadas algumas definições relacionadas às funções e à natureza das restrições.

Programação Linear: um programa linear é um problema de otimização em que a função objetivo e todas as demais funções de restrição são lineares. Os problemas de

Programação Linear buscam a distribuição eficiente de recursos limitados para atender um determinado objetivo, em geral, maximizar lucros ou minimizar custos.

Programação Não-Linear: quando a função objetivo e/ou as funções de restrição são não-lineares. Em grande parte das aplicações do dia-a-dia, modelos lineares refletem apenas aproximações dos modelos reais. Fenômenos físicos ou econômicos são geralmente melhor representados por modelos não lineares.

Na maioria dos casos um problema complexo, como por exemplo, a política de produção detalhada de uma corporação gigante, ou o planejamento de uma grande agência governamental, ou mesmo a concepção de um dispositivo complexo não pode ser tratado diretamente na sua totalidade pela representação de todas as escolhas possíveis. Em vez disso, tal complexidade tem de ser decomposta em subproblemas que em separado têm limitações impostas para restringir o seu âmbito de aplicação.

Programação Quadrática: quando a função objetivo é quadrática e as restrições são lineares.

Programação Quadrática Convexa: trata de problemas em que a função objetivo é convexa e quadrática. Uma função é considerada convexa: se a linha que conecta dois pontos na função nunca se estende abaixo da mesma. Uma função convexa é garantida estar livre de ótimos locais distintos O SVM considera problemas em que as restrições são lineares, a função objetivo é convexa e quadrática.

Para realizar a procura de máximos e mínimos condicionados, como nos problemas quadráticos convexos pode-se utilizar o método de Lagrange. Esse método é usado para solucionar o treinamento de SVM.

3.2.2. Classificador de Margem Máxima

O modelo mais simples de SVM é chamado de Classificador de Margem Máxima. Ele trabalha apenas com dados linearmente separáveis, ficando restrito, portanto, a poucas aplicações práticas. Apesar dessa limitação, o Classificador de Margem Máxima apresenta propriedades importantes e é a pedra fundamental para a formulação de SVMs mais sofisticadas.

Com os SVMs tentamos encontrar uma fronteira de decisão que maximiza a margem ou a distância que separa amostras positivas das amostras negativas. A Figura 11 mostra um espaço de características linearmente separável para um conjunto de treinamento bidimensional.

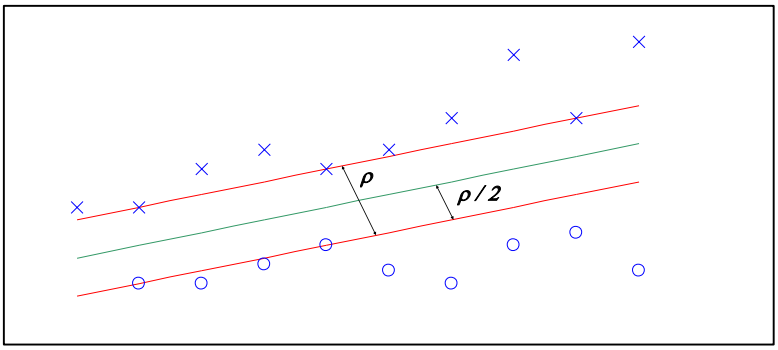

Figura 11 - Classificador de Margem Máxima

Um hiperplano é considerado de Margem Máxima (ou de Separação Ótima) se separa um conjunto de vetores sem erro e a distância entre os vetores (das classes opostas) mais próximos ao hiperplano é máxima [9]. Para o caso linearmente separável, o algoritmo de SVM tem como objetivo encontrar esse hiperplano.

Considerando que existem infinitos planos que separam as amostras positivas das negativas, a função que define como será classificada uma nova amostra desconhecida será:

$$f(u) = w.u + b \ > 0 \qquad (12)$$

Considerando que x_+ e x_- são as amostras positivas e negativas, respectivamente, podemos definir que:

$$f(x_+) = w.x_+ + b > 0 \qquad (13)$$

$$f(x_-) = w.x_- + b \ > 0 \qquad (14)$$

Logo a largura da margem de separação é:

$$w.(x_1 - \ x_2) = 2 \qquad (15)$$

Dividindo pela extensão de w:

$$\frac{w}{\|w\|}.(x_1 - \ x_2) = \frac{2}{\|w\|} \qquad (16)$$

Dessa forma, é necessário maximizar a expressão:

$$\frac{1}{\|w\|} \qquad (17)$$

O que é equivalente a minimizar a expressão:

$$\frac{1}{2}\|w\|^2 \qquad (18)$$

Será utilizado o método de Lagrange para minimizar a expressão. O método de Lagrange utiliza uma multiplicador α (alfa). A expressão fica da seguinte forma:

$$L = \frac{1}{2}\|w\|^2 - \sum \alpha_i [y_i(\overline{w}.\bar{x}_i + b) - 1] \qquad (19)$$

$$\frac{\partial L}{\partial w} = \overline{w} - \sum a_i y_i \overline{x}_i = 0 \rightarrow \overline{w} = \sum a_i y_i \overline{x}_i \qquad (20)$$

$$\frac{\partial L}{\partial b} = -\sum a_i y_i = 0 \rightarrow \sum a_i y_i = 0 \qquad (21)$$

O resultado anterior mostra que a classificação de um novo ponto depende apenas da soma linear do produto das amostras pelo ponto desconhecido, ou seja:

$$f(u) = w.u + b = (\sum a_i.y_i.x_i.u) + b \qquad (22)$$

Esta é a solução para amostras espacialmente separáveis, mas no mundo real este é caso pouco frequente. Como classificar as amostras quando não há uma separação espacial entre elas?

Para tanto utiliza-se uma função que mapeie as amostras em uma outra dimensão, onde as amostras positivas e negativas sejam separáveis.

3.2.3. Funções Kernel

As representações Kernel trabalham com a projeção dos dados em um espaço de características com alta dimensão para permitir a classificação em espaços não-linearmente separáveis. Trata-se de uma estratégia de pré-processamento que envolve mudar a representação dos dados.

Esse passo é equivalente ao mapeamento do espaço de entrada em um novo espaço chamado espaço de características que são as funções Kernel. A Figura 12 apresenta um mapeamento de um espaço de entrada linearmente inseparável, para um espaço de características de maior dimensão, onde os dados podem ser separados linearmente. É importante observar que ambos os gráficos representam espaços bidimensionais por razões puramente didáticas.

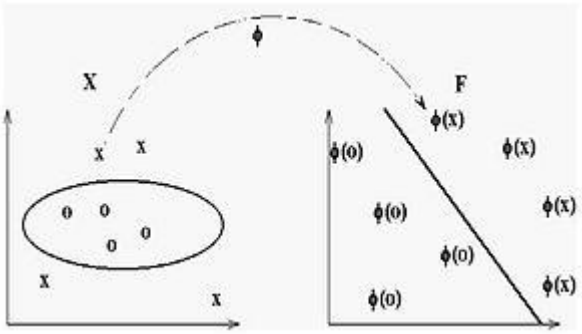

Figura 12 - Exemplo de função kernel

Suponha o problema de otimização quadrática visto anteriormente:

$$Max = \sum a_i - \frac{1}{2}\sum\sum \alpha_i\alpha_j y_i y_j x_i x_j \qquad (23)$$

Sujeito a:

$$\sum \alpha_i y_i \ e\ 0 \ \leq a \ \leq C \qquad (24)$$

De acordo com o teorema de Cover: "A probabilidade de um problema ser linearmente separável é maior em espaços de maior dimensionalidade. " Em outras palavras, embora a dimensão do espaço aumente, a complexidade diminui, porque a classificação, que no espaço de entrada só era possível utilizando superfícies de decisão não lineares, no espaço de características, pode ser feita apenas com um simples hiperplano (superfície de decisão linear).

Considera-se a função $\varphi(\overline{x}) \rightarrow x$. Logo a expressão anterior pode ser reescrita:

$$Max = \sum a_i - \frac{1}{2}\sum\sum \alpha_i\alpha_j y_i y_j \varphi(x_i)\varphi(x_j) \qquad (25)$$

$$K(x_i, x_j) = \varphi(x_i)\varphi(x_j) \qquad (26)$$

$K(x_i, x_j)$ é o núcleo do produto interno (Kernel).

1. Kernel gaussiano:

$$K(x_i, x_j) = e^{-\frac{1}{2\sigma^2}\|x_i - x_j\|^2} \qquad (27)$$

2. Kernel polinomial:

$$K(x_i, x_j) = (x_i{}^t . x_j + 1)^p \qquad (28)$$

3. Kernel sigmoidal:

$$K(x_i, x_j) = tanh(\beta_0 {x_i}^t x_j + \beta_1) \qquad (29)$$

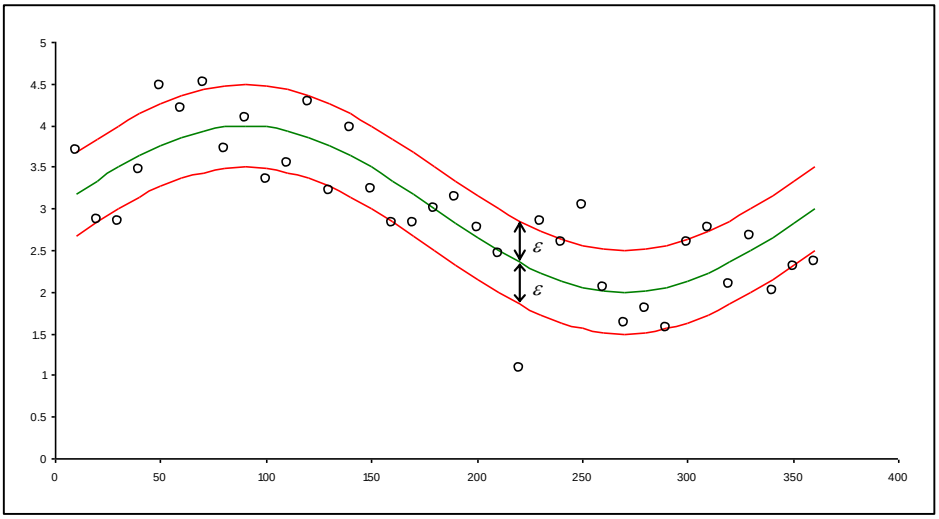

Figura 13 - Exemplos de Vetores de Suporte

Capítulo IV

4. Implementações e Testes

4.1. Metodologia dos Testes

No escopo de pesquisa e melhoria do método de Viola-Jones [3] foram considerados:

- Pesquisa e reprodução de um algoritmo de detecção de faces, baseado no método Viola-Jones [3], medição e avaliação dos resultados;
- Introdução de modelos de características estendidos, melhoria nos estágios de treinamento, medição e avaliação dos resultados alcançados;

No treinamento do método de detecção de faces em rede SVM foram considerados os seguintes tipos de entradas:

- Filtros de Eigenfaces [11];
- Filtros de características Haar-like estendidos (modelos usados no primeiro objeto da pesquisa).

Um critério importante na avaliação das detecções é a existência de falsos positivos. Os falsos positivos são regiões da imagem que não são faces humanas, mas que são identificadas positivamente pelos detectores. Portanto os falsos positivos são considerados problemas nos resultados das detecções.

A seguir serão descritos o ambiente de desenvolvimento e base de imagens utilizados.

4.1.1. Utilização do Matlab

O Matlab é uma ferramenta completa de utilização gráfica e matemática com uma linguagem de alto nível e um ambiente interativo voltado para a computação numérica. O software trabalha com matrizes e inclui bibliotecas de funções específicas, entre elas as relacionadas ao processamento de imagens e redes neurais [12].

Os arquivos de imagem podem ser manipulados através do “Image Processing ToolBox”, que inclui as funções tais como: imread e imreduce, utilizadas neste projeto.

A imagens lidas pelo Matlab são armazenadas em matrizes de pixels, permitindo a manipulação matemática das mesmas através de funções específicas disponíveis. A matriz

de pixels tem o mesmo número de linhas e colunas da imagem de entrada e o valor de cada posição na mesma representa a cor de um pixel.

As versões mais recentes do Matlab, a partir da R2012a, disponibilizam funções que implementam o algoritmo de detecção de faces Viola-Jones. Neste estudo, utilizamos a versão R2010a, e dessa forma, implementamos toda a codificação do método e suas alterações.

As Redes Neurais no Matlab são criadas e manipuladas através do Neural Network ToolBox, que inclui diversas funções de aprendizado e simulação. Foi utilizada ainda uma biblioteca para treinamento de máquina de vetor de suporte (SVM): o libsvm [13] que possui performance superior às funções SVM nativas do toolbox do Matlab. Durante o treinamento do SVM foram avaliados diferentes valores dos parâmetros de regularização e do Kernel e escolhidos os de melhor resposta nas amostras de validação.

As redes criadas no Matlab podem ser configuradas quanto ao número de camadas escondidas, tipo de função e número de épocas do treinamento, erro final de aprendizado, taxa de treinamento e formato de saída. O treinamento das redes com aprendizado supervisionado é feito utilizando a função train ou svmtrain, conforme o tipo de treinamento utilizado.

Nas redes de retro propagação do erro, os neurônios de entrada e neurônios na camada oculta usaram as funções de ativação do tipo ‘tansig’ (função sigmodal) que permite uma ótima separação entre as entradas (-1 ou 1).

4.1.2. Base de Dados

Em um projeto de detecção de faces, como é o foco deste estudo, é fundamental a utilização de uma base de dados adequada que seja padronizada e esteja disponível para utilização em estudos posteriores que ratifiquem, complementem e ampliem os resultados do estudo atual.

As imagens usadas no treinamento foram retiradas de sites acadêmicos reconhecidos e utilizados amplamente em pesquisas de imagens:

- O repositório da Washington University in St. Louis (WUSTL) composto de 4.916 amostras positivas (com faces humanas) e 7.960 amostras negativas (sem faces). Este conjunto de imagens possui dimensões 24x24 pixels, no formato ‘gif’ e em escala de cinza. [14]

- O CBCL Face Database - MIT Center For Biological and Computation Learning composto de 180 imagens de teste, sendo 50 imagens inclinadas a 45º, no formato ‘gif’ e em escala de cinza. [15]
- O Carnegie Mellon University (CMU) - Vision & Autonomous System Center, the Robotics Institute composto de 472 amostras positivas (com faces humanas) e 23.573 amostras negativas (sem faces). Este conjunto de imagens possui dimensões 19x19 pixels, no formato ‘pgm’ e em escala de cinza. [16]
- Base de imagens do Computer Vision Research Projects da Universidade de Essex da Inglaterra, composta de 7.900 imagens de rostos de homens e mulheres de diferentes idades com oclusão de óculos e barba, no formato 24bit colorido JPEG [17].

Para o treinamento dos classificadores baseados em Viola-Jones [3], foram utilizadas imagens do banco WUSTL [14] pois estão no formato 24x24, proposto pelo método. Em cada estágio do treinamento foram usadas 100% das amostras positivas e uma distribuição ponderada das amostras negativas, de forma a não viciar o treinamento entre os diferentes estágios.

Para a validação do algoritmo foi utilizado um conjunto de 500 imagens de faces [17] e 100 imagens sem faces diferentes das imagens usadas para o treinamento (validação quantitativa), e para o teste manual e visual (validação qualitativa) foram utilizadas 20 imagens do MIT [15] com inúmeras pessoas presentes nas mesmas e em poses com inclinação.

4.2. Resultados da Implementação do Método Viola-Jones

O objetivo desta fase de testes é validar o algoritmo implementado baseado nos 3 principais conceitos do método Viola Jones de detecção de faces (detalhado no Capítulo 2):

- Filtros de modelos de características Haar;
- Classificador forte como um conjunto de classificadores fracos;
- Treinamento em cascata de classificadores.

Foram implementadas diferentes configurações de treinamento, de forma a avaliar a influência de diferentes características do modelo estudado:

- Quantidade de amostras positivas e negativas;
- Número de modelos Haar-like;
- Quantidade de Características baseadas nos modelos escolhidos.

4.2.1. Escolha do conjunto de características para treinamento

O número de modelos Haar-like (vide Figura 14) foi avaliado também. Os modelos A, B, C e D possuem alguma relação com formas presentes no rosto humano, em que existem zonas mais claras ou escuras na face (boca, olhos, nariz e testa). Foram treinadas configurações com duas características (A e B), três (A, B e C) e quatro características (A, B, C e D).

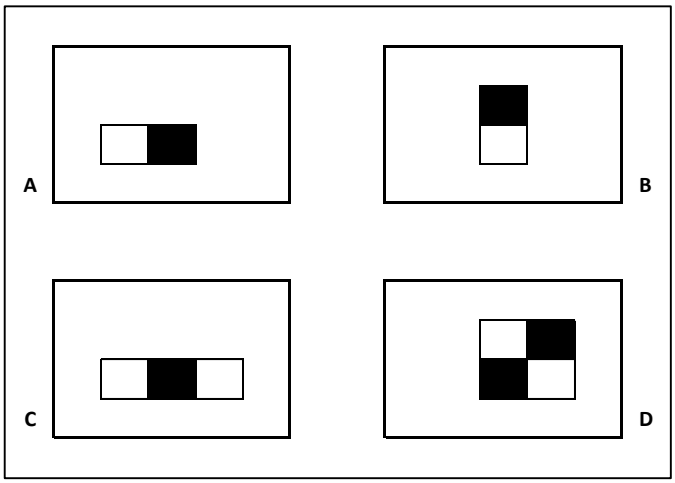

Figura 14 - Modelos de Características Haar-like originais

Conforme foi visto na seção 2.2, se considerarmos que todos os 24x24 pixels da janela básica podem ser utilizados, o total de combinações de tipos, tamanhos e posições das características pode chegar a 180.000.

Cada configuração adotada possui influência no desempenho dos detectores, mas também no tempo de duração dos treinamentos, que podem variar de alguns segundos para várias horas, dias ou semanas.

A questão de moldura não utilizável na janela base, por exemplo, é um fator a ser considerado. Pode-se verificar que há quase sempre um espaço entre a face e a borda da imagem. Se considerarmos o espaço na borda da imagem de 1 ou 2 pixels (moldura) que não contém características podemos reduzir a quantidade de combinações e configurações possíveis.

Dessa forma, foi feita uma seleção das características utilizadas que envolveu fatores como utilização de uma moldura na janela básica, espaçamento entre as características na horizontal e na vertical e a escala de crescimentos das características dentro da janela. A Figura 15 exemplifica as possibilidades de escolha das características.

Com isso, antes de prosseguir com a avaliação do método de detecção precisamos definir quantas e quais características serão utilizadas nos testes e quais os critérios para tal. Os critérios para escolha das características foram:

- Sem moldura, moldura com espessura de 1 ou 2 pixels;
- Espaçamento horizontal entre as características de 0, 1 ou 2 pixels;
- Espaçamento vertical entre as características de 0, 1 ou 2 pixels;
- Crescimento da escala das características de 1, 2 ou 3 vezes.

Assim, realizamos diversos testes com a geração de conjuntos de características de acordo com combinações dos critérios acima, que foram treinadas e avaliadas quanto às taxas de detecção obtidas por cada conjunto.

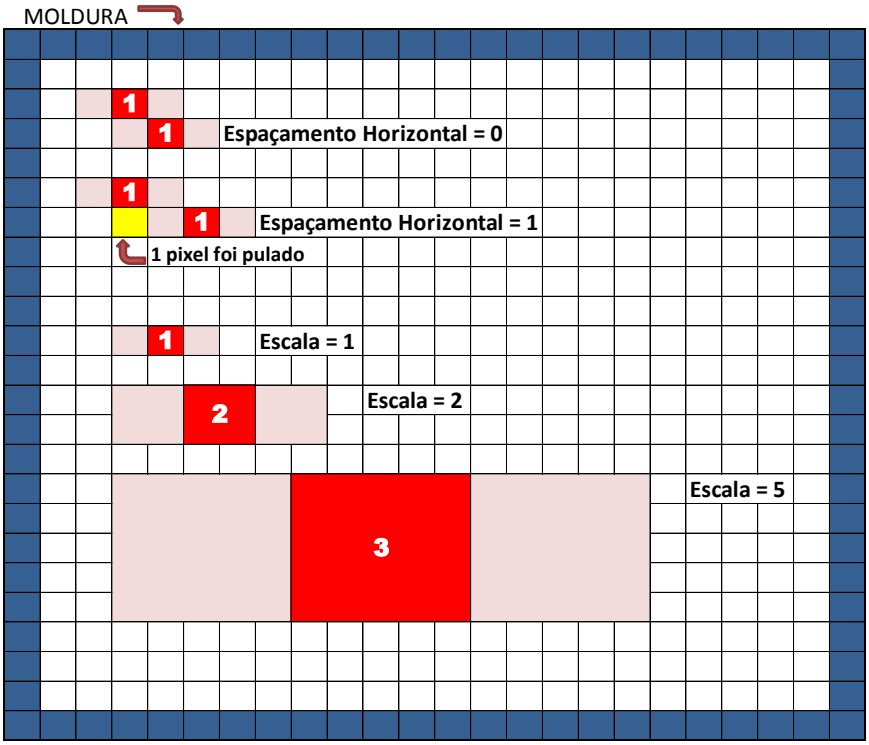

Figura 15 - Exemplos de Características na Janela de 24x24 pixels

Após as diversas simulações, optou-se por utilizar configurações com totais entre 2.000 e 5.000 características que mostraram uma boa relação tempo de treinamento e alta taxa de detecção. A configuração de varredura da janela de 24x24 pixels foi feita a cada 2 pixels na horizontal e na vertical, com uma moldura não utilizável de 2 pixels e um fator x2 de crescimento da escala das características.

4.2.2. Algoritmo de treinamento Adaboost

Foi implementado um módulo de aprendizado de máquina baseado no Adaboost, que utiliza uma série de classificadores fracos formando um classificador forte. A cascata de classificadores é planejada para que os estágios iniciais permitam um maior número de falsos positivos que serão descartados nos estágios seguintes.

A seguir o algoritmo do modulo de treinamento utilizado:

1. Monta a máscara dos modelos de características. É gerado o conjunto de todas as características possível dentro da janela básica (no caso 24x24 pixels). Esta máscara é também utilizada na detecção.
2. Monta o conjunto de características de faces. Lê todos os arquivos de imagens e extrai as características haar-like. (fa = total de amostras de faces do treinamento).
3. Monta conjunto de características de não faces. Lê todos os arquivos de imagens e extrai as características haar-like. (nf = total de amostras de não faces do treinamento).
4. Para cada estágio do classificador forte:

4.1 Inicializa os pesos:

$$w_{fa,i} = \frac{1}{2\ x\ fa} \qquad (30)$$

$$w_{nf,i} = \frac{1}{2\ x\ nf} \qquad (31)$$

4.2 Para cada classificador

4.2.1 Normaliza os pesos:

$$w_{fa,i} = \frac{w_{fa,i}}{\sum_{j=1}^{fa} w_{fa,j}} \qquad (32)$$

$$w_{nf,i} = \frac{w_{nf,i}}{\sum_{j=1}^{nf} w_{nf,j}} \qquad (33)$$

4.2.2 Seleciona o melhor classificador fraco (h_t), que é o que produz o menor erro:

$$\epsilon_i = \sum_i w_i |h_i(x_i) - y_i| \qquad (34)$$

4.2.3 Atualiza os pesos:

$$w_{t+1,i} = w_{t,i} \frac{\epsilon_t}{(1-\epsilon_t)}^{(1-e_i)} \quad (35)$$

Onde: e_i = 0 se x_i classificou corretamente e e_i = 1, caso contrário.

4.3 O classificador forte neste estágio será:

$$h(x) = \begin{cases} 1\,, se\ \sum_{t=1}^{T} \alpha_t . h_t\,(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0\,, caso\ contrário \end{cases} \quad (36)$$

Onde:

$$\alpha_t = log \frac{(1-\epsilon_t)}{\epsilon_t} \quad (37)$$

A arquitetura da cascata de classificadores foi obtida através de sucessivas etapas de treinamento. Após cada etapa de treinamento foram medidas as taxas de detecção e falsos positivos e acrescentados os estágios necessários no final da cascata até que as taxas desejadas foram alcançadas.

Após este processo, a configuração da máquina de treinamento ficou com 7 estágios de treinamento e cada estágio do classificador foi constituído das seguintes quantidades de classificadores fracos (vide Tabela 1):

Tabela 1 - Distribuição de classificadores nos estágios

Estágio 1	Estágio 2	Estágio 3	Estágio 4	Estágio 5	Estágio 6	Estágio 7
2	10	25	25	50	50	50

A escolhas das quantidades acima foi baseada nos resultados de Viola-Jones [3], e considerando que os estágios iniciais devem ser mais simples e os estágios finais mais complexos. Cada configuração de treinamento foi salva para utilização nos detectores.

A quantidade de amostras é um fator decisivo dos resultados. A recomendação de Viola-Jones [3] é que sejam utilizadas aproximadamente 5.000 imagens positivas e 10.000 negativas, mas não todas em todos os estágios. Verificou-se que se o mesmo conjunto de amostras for utilizado em todos os estágios não haverá variação nos resultados dos

classificadores, pois os thresholds e características escolhidos em cada estágio manterão o mesmo padrão.

Assim, o treinamento foi realizado com as mesmas 4.916 imagens positivas e com 2.000 amostras negativas em cada estágio. As amostras negativas foram escolhidas aleatoriamente em cada estágio dentro do universo de 7.960 existentes.

Tabela 2 - Detalhes das configurações de treinamento Viola-Jones [3] iniciais

Configuração de Treinamento	1	2	3
Amostras positivas	4916	4916	4916
Amostras negativas	7960	7960	7960
Modelos Utilizados	A e B	A, B e C	A, B, C, D
Total de Características	1969	3619	4244
Tempo de Treinamento	303s	527s	626s

Após o treinamento passou-se a fase de teste de detecção.

4.2.3. Avaliação da Detecção com os modelos A B C e D

Conforme citado anteriormente, foram feitas avaliações em um bloco de amostras utilizadas no treinamento para estatística quantitativa, e ainda testes visuais em imagens de teste novas que não fizeram parte do treinamento.

Foi construído um detector em múltipla escala, pois sem isso qualquer face nas imagens de teste com dimensões maiores que a escala base de treinamento de 24x24 pixels não seria identificada.

A detecção multiescala pode ser realizada de duas formas: através do redimensionamento da janela original do treinamento (24x24) para alcançar as resoluções reais de cada imagem do teste, ou através da redução da resolução da imagem de teste em etapas até um limite mínimo. Foram testadas as duas formas e por apresentar menor tempo de resposta total, optou-se pela redução da resolução da imagem de teste.

Dessa forma, foi criado um esquema para redimensionar a imagem de testes iterativamente. Em cada incremento de escala, a imagem é subdividida em N regiões onde será feito o teste de face ou não face. Isso é feito da escala inicial 1 até a escala máxima de 10 vezes a inicial.

A seguir o algoritmo do modulo de detecção multiescala utilizado:

1. Define o valor da Escala = 1 e da Escala final = Escala / 10;
2. Executa o loop enquanto Escala > Escala final;
   2.1. Redimensiona a imagem original para imagem atual;
   2.2. Calcula todas as regiões de 24x24 pixels possíveis na imagem atual.
   2.3. Calcula a Integral da Imagem de cada região e Monta o conjunto de características para cada região;
   2.4. Avalia cada janela pelos estágios do classificador:
   - Se o valor computado for menor do que o threshold do classificador, a região não classificou positivamente para o estágio atual e, portanto, essa região não passará pelos próximos estágios.
   - Se valor computado for maior que o threshold do classificador, passa para o próximo estágio.

   2.5. As regiões que chegarem ao final dos estágios serão salvas;
   2.6. Incrementa a escala.
3. Ao final verifica se há regiões sobrepostas e elimina as redundantes.

Foram testados 2 fatores de redução/aumento de escala: 1,0/0,97 e 1,0/0,92. Não foram identificadas diferenças no resultado da detecção em função dos fatores de escala utilizados. Sendo assim, foi utilizada o fator de 0,92, pois é aproximadamente 2,5 vezes mais rápido que a utilização do fator de 0,97.

Nos testes quantitativos foram utilizadas 500 imagens positivas e 100 imagens negativas para análise em bloco. O resultado da avaliação quantitativa sobre as configurações de treinamento descritas no item anterior foi o seguinte:

Tabela 3 - Resultado das detecções nas configurações Viola-Jones [3] iniciais

Configuração de Treinamento	**1**	**2**	**3**
% acerto sobre amostras positivas	84%	79%	78%
% acerto sobre amostras negativas (Falsos Positivos)	84%	97%	99%
% acerto total	84%	82%	82%

Os percentuais de acerto ficaram dentro da margem dos resultados de Viola-Jones (entre 76% e 93%). Este resultado indica que não é uma quantidade muito grande de características que importa, mas a qualidade das características utilizada.

A seguir foi feita uma avaliação qualitativa utilizando imagens do MIT [15]. Algumas imagens incluem várias faces, e em poses com inclinação de 45º e pouca iluminação. Como exemplo, são apresentados resultados do teste de detecção com a imagem bttf301.gif, com resolução de 610x395 pixels.

A configuração 1 apresentou o melhor índice de faces corretas detectadas, mas algumas detecções de falsos positivos foram identificadas. Veja um exemplo na Figura 16. Pode-se verificar que o rosto do ator Michael J. Fox não foi detectado. Isso pode estar relacionado com a deficiência do método Viola-Jones [3] em tratar faces pouco iluminadas ou mais escuras em relação ao fundo, que é caso da situação do rosto de Fox na Figura 16.

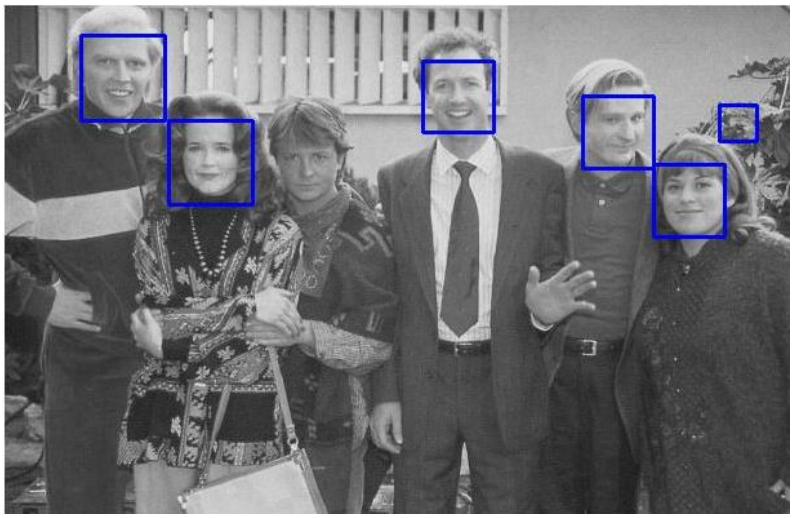

Figura 16 - Exemplo de Detecção com a Configuração 1.

Com a mesma imagem bttf301.gif, podemos analisar que as configurações 2 e 3 apresentaram menos faces detectadas, mas uma taxa menor de falsos positivos, especialmente para a Configuração 3. Veja um exemplo na Figura 17. Verifica-se que há uma relação direta entre o aumento na taxa de detecção com o crescimento na taxa de falsos positivos. Entretanto, neste caso a taxa de detecção foi abaixo do esperado: apenas 2 faces em 6 existentes na figura.

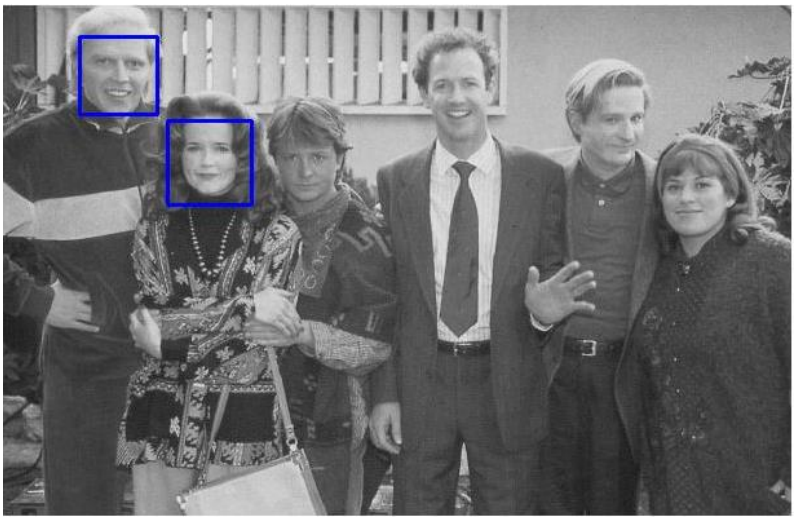

Figura 17 - Exemplo de Detecção com a Configuração 3.

4.3. Modificações Propostas ao Viola Jones

De acordo com as medições da seção anterior, existe uma oportunidade de melhoria no percentual de faces detectadas corretamente, visto que a taxa está em 84%. Além disso, o desempenho do método de Viola-Jones piora quando as faces a serem detectadas tem pouca iluminação, estão inclinadas ou sofrem oclusão [3].

A partir de deduções empíricas foi verificada a possibilidade de criação de outros modelos de características, além dos quatro propostos originalmente, com o objetivo de melhoria nas taxas de detecção de faces. Após observação dos modelos originais de Viola-Jones e dos resultados da primeira parte deste estudo vimos a possiblidade de criação dos modelos estendidos apresentados na Figura 18.

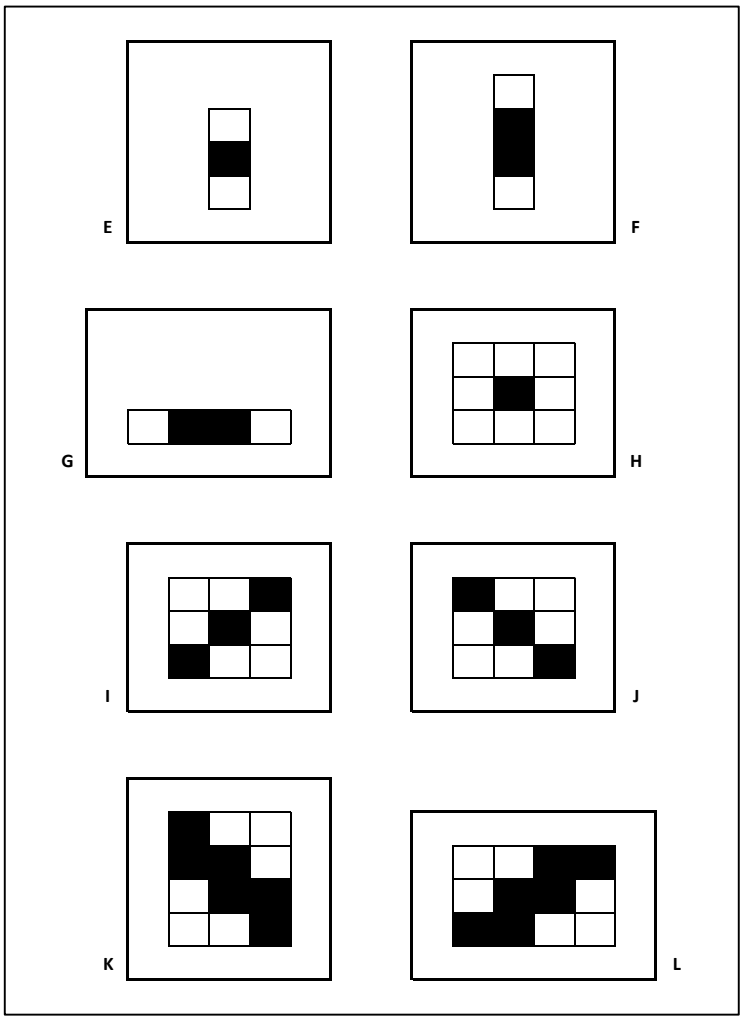

Figura 18 - Modelos Haar-like propostos

A premissa na criação dos modelos estendidos foi manter relação com as formas do rosto, mas acrescentar características que reflitam a oclusão ou inclinação nas faces a

serem treinadas. Os modelos E, F, G e H complementam os originais no sentido de ressaltar pontos de boca (G), olhos (E ou G), nariz (F), e consideram objetos como bigode (G) e óculos (E e G). Os modelos I, J, K e L têm motivação direta na inclinação dos modelos em ambos os sentidos.

A ideia de modelos estendidos já foi abordada por outros estudos, como por exemplo por Lienhart e Maydt [18], que propuseram novos modelos de características obtidas após rotacionar em 45º alguns modelos básicos. Eles também alteraram a fórmula de cálculo da Integral da Imagem, pois após a rotação parte da característica fica abaixo ou a direita do ponto base do cálculo da soma de pixels definido pela fórmula original (vide Figura 19).

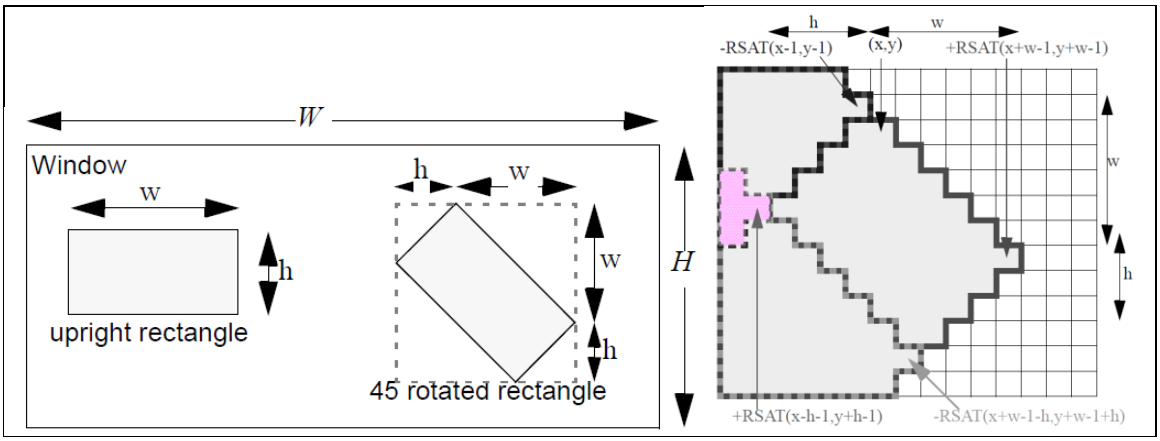

Figura 19 - Exemplo de características com giro de 45º [18].

Diferentemente, no estudo atual, os novos modelos estendidos são obtidos pela utilização de retângulos sem giro e combinados entre si para obter formas inclinadas, e assim mantendo o mesmo cálculo para a integral de imagem (vide Figura 20).

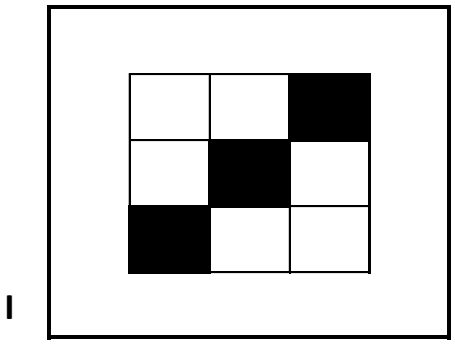

Figura 20 - Exemplo da formação do modelo I usado neste estudo.

Lienhart e Maydt [18] treinaram e testaram dois sistemas de detecção de faces: um com as características básicas e outro com o conjunto de características haar-like estendidas. Em média, a taxa de falsos positivos foi cerca de 10% menor para as características estendidas, e a taxa de sucesso se manteve no mesmo nível em ambos os casos. Conforme conclusão dos autores os resultados sugerem que, embora o crescimento do conjunto de características aumente a complexidade do processo de aprendizagem, há uma recompensa pelos ganhos no conhecimento do comportamento do processo que a máquina adquiriu.

4.3.1. Implementação das Modificações Propostas

Os seguintes conjuntos de modelos estendidos foram testados:

Tabela 4 - Detalhes das configurações de treinamento com as características propostas

Configuração de Treinamento	4	5	6	7	8
Amostras positivas	4916	4916	4916	4916	4916
Amostras negativas	7960	7960	7960	7960	7960
Modelos Utilizados	A, B, C, D, E, F, G	A, B, C, D, E, F, G, H, I, J	A, B, C, D, E, F, G, H, I, J, K, L	A, B, C, E, F, G, H, I, J, K, L	A, B, C, E, F, G
Total de Características	5267	5591	5654	5029	4642
Tempo de Treinamento	768s	833s	825s	1191s	1084s

Foi desenvolvida e testada uma proposta de configuração da máquina de treinamento em ‘boosting’ diferente da proposta de Viola-Jones [3], onde foram utilizados 3 estágios de treinamento e cada estágio do classificador foi constituído das seguintes quantidades de classificadores fracos:

Tabela 5 - Distribuição de classificadores nas configurações propostas

Estágio 1	Estágio 2	Estágio 3
3	25	150

A configuração acima foi obtida após sucessivas tentativas de melhoria. Notou-se que ao diminuir a quantidade total de estágios foi melhorada a taxa de acertos positivos, mas houve um aumento de falsos positivos. Para resolver o problema chegamos na configuração de 3 estágios com 150 classificadores no último estágio para reduzir os falsos positivos.

A seguir são apresentados os resultados dos testes com as modificações propostas.

4.3.2. Avaliação do Desempenho com as Modificações Propostas

Seguindo a metodologia adotada, foram feitas avaliações no mesmo conjunto de amostras utilizadas anteriormente para estatística quantitativa. Os resultados estão apresentados nas Tabelas 6 e 7.

Tabela 6 - Resultados nas detecções com as características estendidas propostas (de 4 a 8)

Configuração de Treinamento	1	2	3	4	5	6	7	8
% acerto sobre amostras positivas	84%	79%	78%	78%	78%	81%	77%	87%
% acerto sobre amostras negativas (Falsos Positivos)	84%	97%	99%	97%	97%	97%	97%	87%
% acerto total	84%	82%	82%	82%	82%	83%	81%	87%

Nota-se que os resultados ao utilizar os modelos estendidos se mantêm para quase todas as configurações, mas há uma melhoria nos resultados da configuração 8.

Para a proposta de distribuição de estágios da tabela 5 foram testadas todas as configurações de características Haar-like (originais e estendidas). Na tabela 7 são apresentados os resultados das configurações que tiveram o melhor desempenho:

Tabela 7 - Resultados nas detecções com a proposta de distribuição de estágios do Adaboost

Configuração de Treinamento	**9**	**10**	**11**
Modelos Utilizados	A, B e C	A, B, C, D, E, F, G, H, I, J	A, B, C, E, F, G
% acerto sobre amostras positivas	91%	87%	89%
% acerto sobre amostras negativas (Falsos Positivos)	91%	91%	91%
% acerto total	91%	87%	89%

Para a avaliação visual qualitativa foram escolhidas as configurações de treinamento 8, 9, 10 e 11 que tiveram os melhores resultados nos testes quantitativos.

A configuração 8 apresentou maior taxa de detecção, e passou a identificar faces inclinadas que não foram identificadas pelas configurações anteriores. Na Figura 21 apresentamos o resultado da detecção de faces utilizando a configuração 8 nas imagens bm6290a.gif e am5438b.gif, com resoluções de 234x313 pixels e 250x361, respectivamente. Essas imagens possuem faces com inclinação de 45º aproximadamente e foram corretamente detectadas.

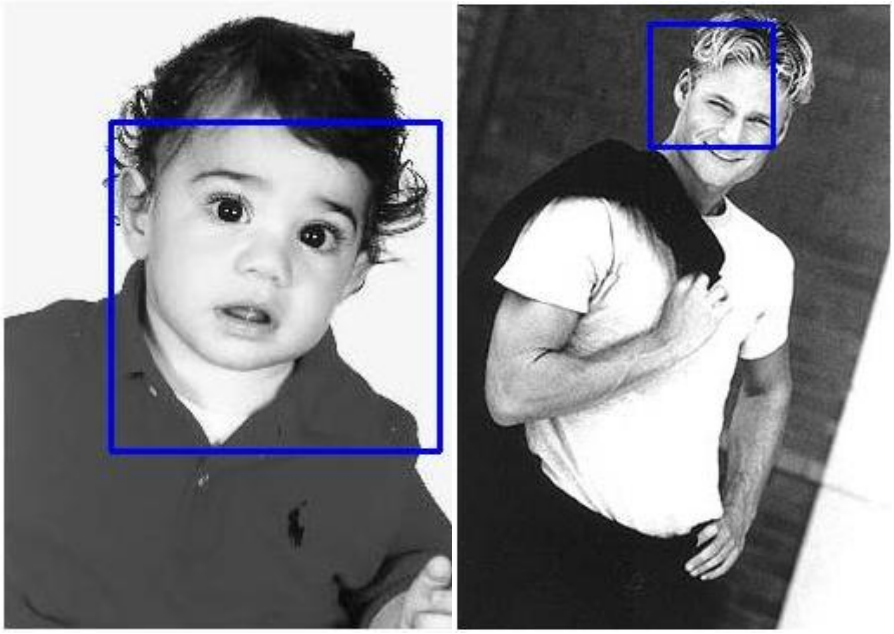

Figura 21 - Exemplo de Detecção com a Configuração 8.

Entretanto também foram identificados mais casos de falsos positivos. A Figura 22 é o resultado da utilização da configuração 8 na imagem newsradio.gif, com resolução 500x500 pixels. A detecção foi positiva em sete dos oito rostos da imagem, mas também foram identificadas 3 regiões que não correspondem a faces.

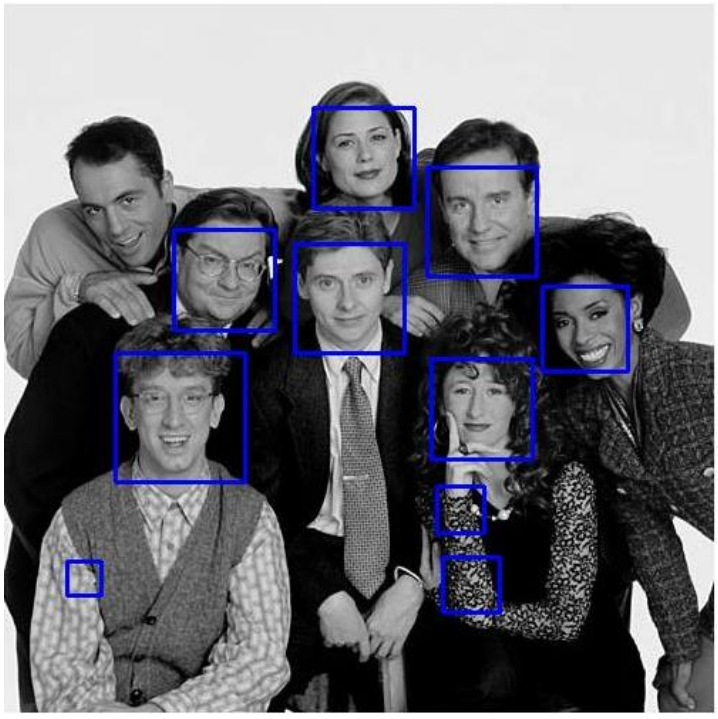

Figura 22 - Exemplo de Detecção com a Configuração 8.

A configuração 9, apesar do seu desempenho elevado nos testes quantitativos, não possui modelos estendidos, e com isso não detecta faces com inclinação. Seu desempenho em imagens sem oclusão e inclinação é bastante eficiente, mas apresenta elevado número de falsos positivos. Isso está demonstrada na Figura 23 que apresenta o resultado da detecção de faces com a configuração 9 na imagem er.gif na resolução 500x500 pixels.

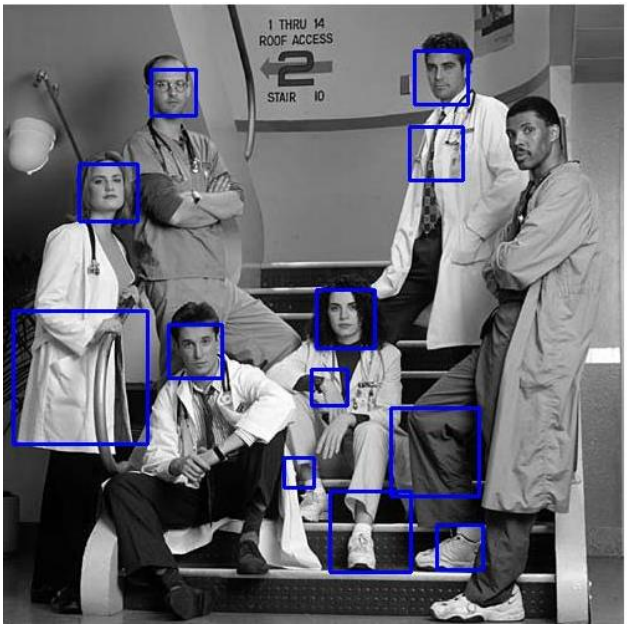

Figura 23 - Exemplo de Detecção com a Configuração 9.

A Configuração 10 considera as melhorias de modelos estendidos e redefinição dos estágios de treinamento. Na Figura 24 é apresentado um exemplo de detecção de faces com a configuração 10 na imagem addams-family.gif na resolução 864x890 pixels. Apesar de identificar faces ainda não capturadas pelas configurações anteriores, ainda possui alguns itens com falsos positivos.

No resultado desta detecção houve um falso positivo no menino Pugsley interessante. Devido ao padrão listrado da sua camiseta, o sistema de detecção considerou a grande incidência de padrões claros e escuros como partes do rosto. Em casos similares, as características induzem aos falsos positivos.

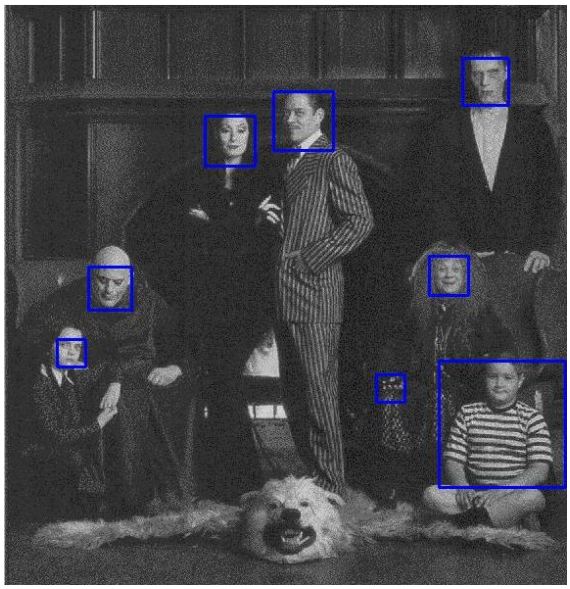

Figura 24 - Exemplo de Detecção com a Configuração 10.

Na Configuração 11 houve, também, uma melhoria na taxa de detecção de imagens inclinadas. Isto pode ser verificado na Figura 25, com o resultado da detecção de faces da configuração 11 na imagem jprc.gif na resolução 480x640 pixels. Essa imagem é um exemplo importante, pois possui várias pessoas deitadas no chão formando um círculo com as cabeças, ou seja, em ângulos variados, mas também algumas com oclusões de rosto como barba e óculos.

Foram detectados dois rostos sem oclusão, e novamente podemos identificar o padrão xadrez na camisa de uma das pessoas que gerou falsos positivos no detector.

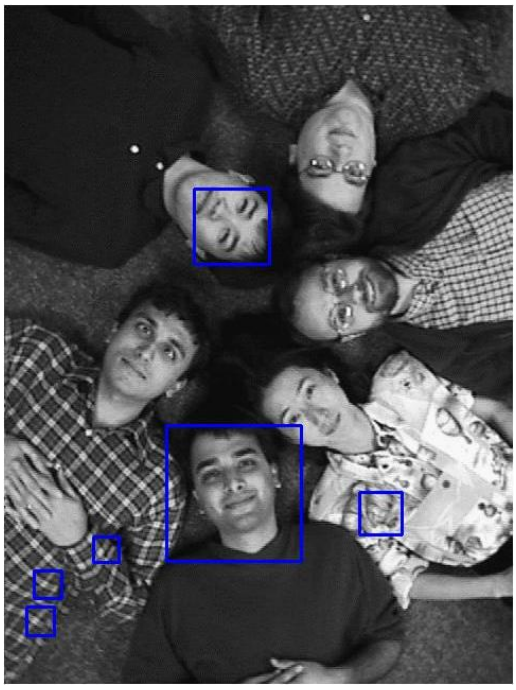

Figura 25 - Exemplo de Detecção com a Configuração 11.

4.4. Detector de Faces usando SVM

Foi implementado no Matlab o algoritmo que implementa uma máquina de aprendizado baseada no SVM (ver descrição teórica no Capítulo 4). O código utiliza uma biblioteca libsvm [13] que permite um desempenho melhorado (em termos de tempo de resposta) em relação a biblioteca nativa do Matlab.

Para escolha dos melhores parâmetros do Kernel foi elaborado um esquema de treinamento seguido de validação dos parâmetros com índices máximos de acerto. Os parâmetros do Kernel (Tipo, C e gama) foram avaliados entre 0 e 4 para Tipo (0 = linear, 1 = polinomial, 2 = gaussiana, 3 = sigmoidal – tanh e 4 = calculado libsvm), 1 e 1.000.000 para o C e 0,001 e 1 para o gama.

Na avaliação e escolha do melhor modelo de rede SVM foi utilizado um esquema de validação cruzada, composto de:

- Um conjunto de treinamento: utilizado para treinar a rede SVM e com 80% das amostras totais;
- Um conjunto de validação: utilizado para avaliar se a rede está generalizando de forma satisfatória e com 20% do total de amostras.

Foram utilizadas 2 bases de imagens de treinamento para o SVM (vide item 4.1.2):

- WUSTL (Washington University in St. Louis) [14] com imagens 24x24 pixels;
- CMU (Carnegie Mellon University) [16] com imagens 19x19 pixels.

Na entrada das máquinas de aprendizado foram testadas as imagens utilizando dois tipos de filtros:

- Os modelos de características Haar-like estendidos usados no método Viola-Jones [3]. Foram escolhidas as configurações que apresentaram melhores resultados na detecção de faces (vide seção anterior): 3 modelos Haar básicos e 3 modelos estendidos, e 2 modelos Haar básicos e 8 modelos estendidos.
- Um vetor de características obtido pelas eigenfaces sobre um conjunto de imagens de faces e não faces

As Eigenfaces são um conjunto de autovetores utilizados no reconhecimento facial humano, desenvolvido originalmente por Kirby e Sirovich [11] e utilizados ainda por Turk e Pentland [19] na classificação de faces. Os autovetores são derivados a partir da matriz de covariância de distribuição de probabilidade sobre o vetor de imagens de faces.

O objetivo é encontrar os vetores que melhor representam as imagens de faces, dentro do espaço de imagens. Estes vetores (eigenfaces) são denominados de auto faces devido à semelhança que possuem com imagens de faces.

As imagens de face são projetadas no subespaço e agrupadas. De forma similar, no treinamento de não faces, as imagens são projetadas no mesmo subespaço e agrupadas. Conforme descrito por Yang et al [20], as imagens de face não apresentam grandes mudanças quando projetadas no espaço de auto faces, mas quando uma imagem de não-face é projetada, ela mostra-se completamente diferente.

Os próprios eigenfaces formam o conjunto de base de todas as imagens utilizadas para a construção da matriz covariância. Isso reduz a dimensão das imagens, permitindo um conjunto menor de características para representar as imagens de treinamento originais.

A seguir apresentamos a tabela 8 com as configurações de treinamento utilizadas:

Tabela 8 - Configurações de Treinamento SVM

Configuração	1	2	3	4	5	6
Filtro de Entrada	Modelos Haar (3 Haar e 3 estendidos)	Modelos Haar (2 Haar e 8 estendidos)	Modelos Haar (3 Haar e 3 estendidos)	Modelos Haar (2 Haar e 8 estendidos)	Eigenfaces	Eigenfaces
Base de Faces	CMU	CMU	WUSTL	WUSTL	CMU	WUSTL
Faces para treinamento	1200	1200	3440	3440	1200	3440
Não faces para treinamento	2200	2200	5576	5576	2200	5576
Tipo Kernel	2	2	2	2	2	2
Kernel C	10	10	10	10	1000	1000
Kernel Gamma	0,001	0,001	0,001	0,001	0,001	0,001
Tempo decorrido	566s	424s	933s	868s	1078s	7320s

Os tempos de treinamento dos filtros Haar foram aproximadamente os mesmos quando comparados com o treinamento com Adaboost para quantidades semelhantes. No

caso do dos filtros de Eigenfaces, o treinamento foi mais demorado, especificamente na configuração 6. Não é objetivo deste estudo avaliar tempos de resposta dos algoritmos, pois são característica do ambiente de desenvolvimento Matlab.

4.4.1. Avaliação da Detecção com SVM

Seguindo a mesma metodologia adotada, foram feitas avaliações das configurações de treinamento de forma quantitativa e qualitativa.

Para avaliação quantitativa foram reservadas 10% do total de amostras disponíveis, e que não foram utilizadas para treinamento, para montar o conjunto de amostras utilizadas para os testes. Os resultados dos testes são apresentados na Tabela 9 a seguir:

Tabela 9 - Resultados nas detecções com as máquinas de aprendizado SVM

Configuração	1	2	3	4	5	6
Filtro de Entrada	Modelos Haar (3 Haar e 3 estendidos)	Modelos Haar (2 Haar e 8 estendidos)	Modelos Haar (3 Haar e 3 estendidos)	Modelos Haar (2 Haar e 8 estendidos)	Eigenfaces	Eigenfaces
Imagens com Faces	472	472	496	496	472	496
Imagens sem faces	400	400	795	795	400	795
% Acertos	54%	54%	98%	97%	55%	83%

Os resultados das configurações 1 e 2 foram muito abaixo do esperado e pode ser devido a condições específicas da base de imagens. Os resultados das configurações 3 e 4 foram os melhores e foram superiores aos obtidos com o método Viola-Jones [3].

Vamos verificar os resultados qualitativos (visuais) quando usada a mesma base de dados de teste do item 4.3.2. Devido aos resultados prévios, os testes qualitativos ficarão restritos às configurações 3, 4 e 6.

A configuração 6 não mostrou resultados aceitáveis. Apresentou muitos falsos positivos e pouca coerência nas faces detectadas. Além disso, apresentou muita lentidão, visto que a máquina SVM demora para apresentar o resultado. Na Figura 26 são apresentados exemplos de detecção SVM na configuração 6 com as figuras married2.gif e trekcolr.gif, com resoluções de 172x201 pixels e 160x100 pixels, respectivamente.

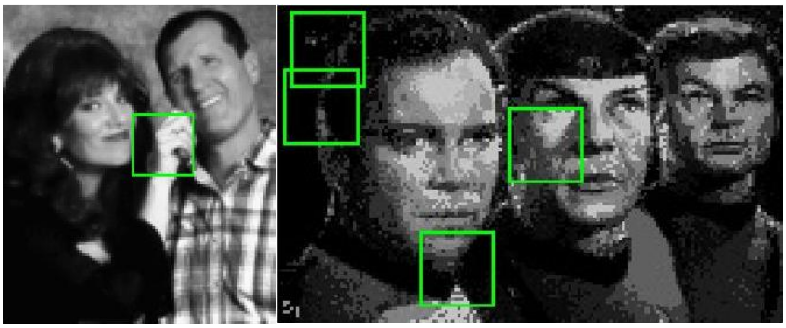

Figura 26 - Exemplos de Detecção utilizando SVM com a configuração 6.

As configurações 3 e 4 tiveram desempenho bastante similar. Houve detecção de diversas faces (70% do total), mas diversos falsos positivos foram incluídos. Isto está ilustrado nas figuras 27 e 28, que apresentam exemplos de detecção com o SVM nas imagens newsradio.gif e class57.gif, nas resoluções 500x500 pixels.e 1280x1024 pixels, respectivamente.

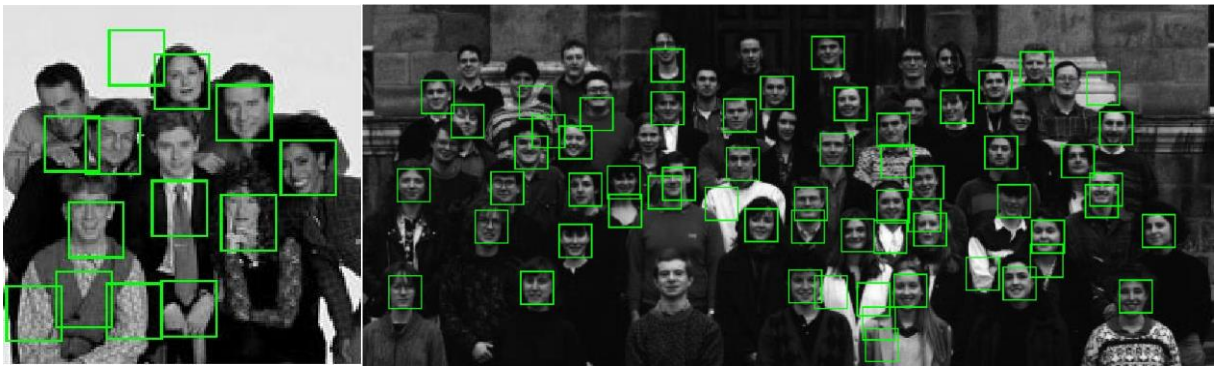

Figura 27 - Exemplo de Detecção utilizando a máquina SVM com a configuração 3.

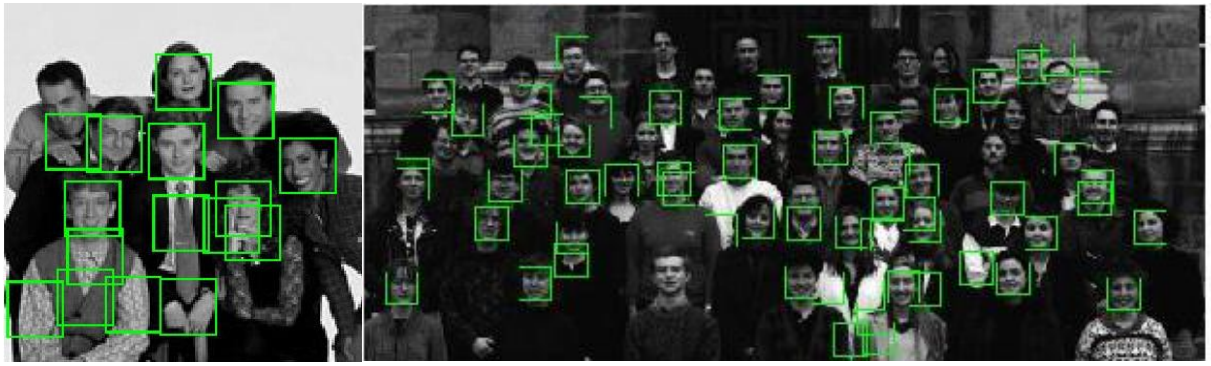

Figura 28 - Exemplo de detecção utilizando a máquina SVM com a configuração 4.

Como referência para comparação, a Figura 29 ilustra o resultado obtido para a detecção nas mesmas imagens das figuras 27 e 28, mas utilizando o método Viola-Jones [3]. Neste último, os resultados são mais precisos com mais rostos identificados corretamente e menos falsos positivos.

Figura 29 - Exemplo de Detecção com a Configuração 8 do Viola-Jones.

Capítulo V

5. Conclusões e Sugestões para Trabalhos Futuros

5.1. Conclusões

A detecção de faces é um instrumento valioso em diversas áreas tais como a segurança, comunicações e divertimento. Conforme foi citado no início deste trabalho e comprovado durante o seu desenvolvimento, a escolha de uma técnica de detecção de faces adequada está diretamente ligada a natureza dos dados e ao tipo de aplicação. Assim sendo, o conhecimento dos detalhes relacionados às imagens e à técnica empregados é de extrema importância para uma boa utilização dessas em aplicações humanas.

A maior dificuldade encontrada no desenvolvimento da detecção de faces é adequar os métodos aos diferentes níveis de iluminação, tonalidades de pele, e situações de oclusão (óculos escuros e barba). Neste ponto a utilização de uma rede neural é fundamental para treinar os classificadores com grandes quantidades de amostras de imagens contendo pessoas de diferentes etnias, em diversas poses e sob condições de iluminação diferentes, e ainda por técnicas de filtragem que destacam características nas imagens de entrada.

Foram desenvolvidos algoritmos no Matlab que implementam detectores de faces baseados em máquinas de aprendizado utilizando métodos de Viola-Jones com filtros de características estendidas e SVM (vetores de suporte) e que utilizam na entrada filtros de características Haar-like ou eigenfaces. O código dos algoritmos utilizados encontra-se transcrito no Apêndice B deste documento.

Aplicando-se a metodologia citada e junto com os algoritmos de detecção de faces implementados foram obtidos resultados satisfatórios (vide Tabela 10).

Tabela 10 - Comparativo dos detectores de faces estudados.

	1	2	3
Método	Detector em Boosting com filtro de entrada com Modelos Haar estendidos baseado em Viola-Jones	Detector em máquina SVM com filtro de entrada com Modelos Haar estendidos	Detector em máquina SVM com filtro de entrada com Eigenfaces
% Acertos	89%	98%	83%
Avaliação Visual	Ótima relação entre faces e falsos positivos detectados. Conseguiu detectar faces com inclinação.	Boa detecção de faces, mas com alto índice de falsos positivos.	Resultado de detecção fraco.

Além dos resultados obtidos, as contribuições deste trabalho foram as melhorias e desenvolvimentos implementados, que devem ser destacados:

- Criação de novos modelos de filtros de características que aumentaram a capacidade de detecção do classificador original, inspirado pelas pesquisas e recomendações de Viola-Jones [3]. Com os modelos de características estendidos foi possível aumentar a taxa de detecção geral para 87% e especificamente melhorar a detecção de faces com inclinação.

- Em paralelo uma nova proposta de distribuição dos estágios de treinamento do Adaboost apresentou uma melhoria na detecção geral em 10%.

- Desenvolvimento de um detector de faces com máquina SVM que utiliza modelos de filtros de características baseados em Haar.

5.2. Sugestões de Trabalhos Futuros

Para continuidade e ampliação deste trabalho no futuro, sugerem-se os seguintes pontos de análise:

1- Nova pesquisa sobre os estágios de boosting e quantidades de classificadores. Está demonstrado que essas quantidades influenciam na quantidade de faces e falsos positivos detectados.

2- Foi identificado ao longo do desenvolvimento da máquina SVM que ela não se comportou da mesma forma em imagens diferentes da escala do treinamento. Uma possível solução seria a utilização de bases de imagens de treinamento customizadas para diferentes escalas. É preciso avaliar o custo-benefício dessa solução.

3- Implementar o método de detecção Viola-Jones em uma Unidade de Processamento Gráfico - GPU, utilizando a linguagem Cuda. Devido às características de computação em paralelo da GPU, tanto o treinamento quanto a detecção podem possivelmente ser acelerados se executados na GPU.

6. Referências:

[1] Zafeiriou, Stefanos; Zhang, Cha e Zhang, Zhengyou - A Survey on Face Detection in the wild: past, present and future Original Research Article Computer Vision and Image Understanding, In Press, Accepted Manuscript, Available online 18 April 2015

[2] Stan Z. Li; Anil K. J. “Handbook of Face Recognition, 2nd Edition”, Springer, 2011.

[3] Viola, P.; Jones, M. - Rapid object detection using a boosted cascade of simple features. In: IEEE. Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 - IEEE Computer Society Conference on. [S.l.], 2001. v. 1, p. I–511.

[4] Papageorgiou, C., Oren, M., and Poggio, - A general framework for object detection. – 1998 - In International Conference on Computer Vision.

[5] Crow, F. - Summed-area tables for texture mapping. – 1984 - In Proceedings of SIGGRAPH, 18(3):207–212.

[6] S. O. Haykin; - Redes Neurais: Princípios e Práticas - 2ª. Edição, Bookman, 2001;

[7] Freund, Y. and Schapire, R.E. - A decision-theoretic generalization of on-line learning and an application to boosting. In Computational Learning Theory: Eurocolt 95, Springer-Verlag, - 1995 - pp. 23–37.

[8] Warren S. McCulloch, and Walter Pitts - "A logical calculus of the ideas immanent in nervous activity" - 1943 - in the Bulletin of Mathematical Biophysics 5:115-133.

[9] Vapnik, V. N. - The Nature of Statistical Learning Theory. Springer Verlag, New York, 2nd edition - 1999.

[10] Mattera, D., Palmieri, F., and Haykin, S. - An explicit algorithm for training support vector machines. IEEE Signal Processing Letters, 6(9):243–245 - 1999.

[11] Kirby. M.; Sirovich. L. - "Applicatíon of the Karhunen-Loeve procedure for the characterizatiori of human faces". IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12. pp.103-108 – janeiro de 1990.

[12] The Mathworks - http://www.mathworks.com/help/index.html - visitado em 08/04/2015.

[13] LIBSVM -- A Library for Support Vector Machines - Chih-Chung Chang and Chih-Jen Lin - http://www.csie.ntu.edu.tw/~cjlin/libsvm/ - visitado em 13/05/2015.

[14] Washington University in St. Louis - http://www.cs.wustl.edu – visitado em 21/09/2015.

[15] CBCL Face Database - MIT Center For Biological and Computation Learning - http://cbcl.mit.edu/software-datasets/ - visitado em 30/11/2015.

[16] Carnegie Mellon University - Vision & Autonomous System Center, the Robotics Institute - http://vasc.ri.cmu.edu/idb/html/face/index.html - visitado em 30/11/2015.

[17] Computer Vision Research Projects - University of Essex - http://cswww.essex.ac.uk/mv/allfaces/ - visitado em 06/04/2015

[18] Lienhart, R., Maydt, J.: An extended set of Haar-like features for rapid object detection. In: IEEE ICIP 2002, vol. 1, pp. 900–903 – 2002.

[19] Turk, M. e Pentland, A. - Face recognition using eigenfaces. In IEEE Conference on Computer Vision and Pattern Recognition – 1991 - pp. 586–591.

[20] Yang. M. H.: Kriegman. D. J.: Ahuja. N. (2002). "Detecting Faces in Images: A Survey. IEEE Transactions on Pattern Analysis and Machine Inteligence. Vol 24, no. 1.

7. Apêndice A – Programas Desenvolvidos

7.1. Treinamento Adaboost com features Haar

```
function iim = integralImagem(imagem)
% Função que calcula a integral da imagem
%
% Entrada - imagem: imagem 2D
%
% Saída   - iim: matriz com os valores da integral para cada pixel
da imagem
%
% Autor: Fernando Otávio Fonseca
% Data: 01/11/2015

[lin,col] = size(imagem);
iim = zeros(lin,col);

% O Matlab referencia a imagem como linha x coluna (Im(lin,col))
% Por isso, neste caso y controla as colunas e x controla as linhas
for y=1:col
    for x=1:lin
        iim(x,y) = imagem(x,y);
        if (y > 1) && (x > 1)
            iim(x,y) = iim(x,y) + iim(x-1,y) + iim(x,y-1) - iim(x-
1,y-1);
        else if (x > 1)
              iim(x,y) = iim(x,y) + iim(x-1,y);
              else if (y > 1)
                     iim(x,y) = iim(x,y) + iim(x,y-1);
                end
            end
        end
    end
end

% ############################################################
```

```
function carac_haar = determinaHaar(iim,mhaar)
% Calcula o valor das características de Haar
%
% Entrada - iim: Matriz da mesma dimensão da imagem com os valores
Integral
%                da Imagem para todos os pixels
%           mhaar: Matriz com o mapa das características de Haar
%                  mhaar(1): coordenada x
%                  mhaar(2): coordenada y
%                  mhaar(3): largura do retângulo
%                  mhaar(4): altura do retângulo
%                  mhaar(5): tipo de feature haar (define o cálculo)
%
% Saída   - Vetor contendo o valor calculado de cada característica
Haar
%
% Autor: Fernando Otávio Fonseca
% Data: 01/11/2015

% carac_haar = zeros(length(mhaar),1);
carac_haar = zeros(length(mhaar(:,1)),1);



for i = 1:size(mhaar)
    switch mhaar(i,5)
        % Molde tipo 1 - Três retângulos na horizontal, soma os dois
da ponta e subtrai o retangulo do centro
        case 1
            carac_haar(i)                                          =
somaRetangulo(iim,mhaar(i,1),mhaar(i,2),mhaar(i,3),mhaar(i,4))     -
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),mhaar(i,2),mhaar(i,3),mhaa
r(i,4))                                                            +
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),mhaar(i,2),mhaar(i,3),mh
aar(i,4));


        % Molde tipo 2 = Dois retângulos na vertical, soma do
primeiro menos a soma do segundo
        case 2
            carac_haar(i)                                          =
somaRetangulo(iim,mhaar(i,1),mhaar(i,2),mhaar(i,3),mhaar(i,4))     -
```

```
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+mhaar(i,4)),mhaar(i,3),mhaa
r(i,4));


        % Molde tipo 3 - Dois retângulos na horizontal, soma do
primeiro menos a soma do segundo
        case 3
            carac_haar(i)                                          =
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),mhaar(i,2),mhaar(i,3),mhaa
r(i,4))                                                            -
somaRetangulo(iim,mhaar(i,1),mhaar(i,2),mhaar(i,3),mhaar(i,4));


        % Molde tipo 4 - Quatro retângulos, dois na horizontal e
dois na vertical, soma de dois retângulos numa diagonal subtraído da
soma dos retângulos da outra diagonal
        case 4
            carac_haar(i)                                          =
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),mhaar(i,2),mhaar(i,3),mhaa
r(i,4))                                                            -
somaRetangulo(iim,mhaar(i,1),mhaar(i,2),mhaar(i,3),mhaar(i,4))     +
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+mhaar(i,4)),mhaar(i,3),mhaa
r(i,4))                                                            -
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),(mhaar(i,2)+mhaar(i,4)),mh
aar(i,3),mhaar(i,4));


        % Molde tipo 5 - Três retângulos na vertical, soma os dois
da extremidade e subtrai o retangulo do centro
        case 5
            carac_haar(i)                                          =
somaRetangulo(iim,mhaar(i,1),mhaar(i,2),mhaar(i,3),mhaar(i,4))     -
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+mhaar(i,4)),mhaar(i,3),mhaa
r(i,4))                                                            +
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+2*mhaar(i,4)),mhaar(i,3),mh
aar(i,4));


        % Molde tipo 6 - Quatro retângulos na vertical, soma os dois
da extremidade e subtrai os dois retangulos do interior
        case 6
            carac_haar(i)                                          =
somaRetangulo(iim,mhaar(i,1),mhaar(i,2),mhaar(i,3),mhaar(i,4))     -
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+mhaar(i,4)),mhaar(i,3),mhaa
r(i,4))                                                            -
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+2*mhaar(i,4)),mhaar(i,3),mh
aar(i,4))                                                          +
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+3*mhaar(i,4)),mhaar(i,3),mh
aar(i,4));
```

```
        % Molde tipo 7 - Quatro retângulos na horizontal, soma os
dois da extremidade e subtrai os dois retangulos do interior
        case 7
            carac_haar(i)                                                 =
somaRetangulo(iim,mhaar(i,1),mhaar(i,2),mhaar(i,3),mhaar(i,4))      -
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),mhaar(i,2),mhaar(i,3),mhaa
r(i,4))                                                               -
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),mhaar(i,2),mhaar(i,3),mh
aar(i,4))                                                             +
somaRetangulo(iim,(mhaar(i,1)+3*mhaar(i,3)),mhaar(i,2),mhaar(i,3),mh
aar(i,4));


        % Molde tipo 8 - Nove retângulos formando um quadrado, soma
os oito retângulos da periferia e subtrai o retangulo do centro
        case 8
            carac_haar(i)                                                 =
somaRetangulo(iim,mhaar(i,1),mhaar(i,2),mhaar(i,3),mhaar(i,4))      +
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),mhaar(i,2),mhaar(i,3),mhaa
r(i,4))                                                               +
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),mhaar(i,2),mhaar(i,3),mh
aar(i,4))                                                             +
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+mhaar(i,4)),mhaar(i,3),mhaa
r(i,4))                                                               -
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),(mhaar(i,2)+mhaar(i,4)),mh
aar(i,3),mhaar(i,4))                                                  +
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),(mhaar(i,2)+mhaar(i,4)),
mhaar(i,3),mhaar(i,4))                                                +
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+2*mhaar(i,4)),mhaar(i,3),mh
aar(i,4))                                                             +
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),(mhaar(i,2)+2*mhaar(i,4)),
mhaar(i,3),mhaar(i,4))                                                +
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),(mhaar(i,2)+2*mhaar(i,4)
),mhaar(i,3),mhaar(i,4));


        % Molde tipo 9 - Nove retângulos formando um quadrado, soma
os seis retângulos acima e abaixo da diagonal e subtrai os
retangulos da diagonal a 45°
        case 9
            carac_haar(i)                          =                      -
somaRetangulo(iim,mhaar(i,1),mhaar(i,2),mhaar(i,3),mhaar(i,4))      +
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),mhaar(i,2),mhaar(i,3),mhaa
r(i,4))                                                               +
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),mhaar(i,2),mhaar(i,3),mh
aar(i,4))                                                             +
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+mhaar(i,4)),mhaar(i,3),mhaa
r(i,4))                                                               -
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),(mhaar(i,2)+mhaar(i,4)),mh
aar(i,3),mhaar(i,4))                                                  +
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),(mhaar(i,2)+mhaar(i,4)),
mhaar(i,3),mhaar(i,4))                                                +
```

```
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+2*mhaar(i,4)),mhaar(i,3),mh
aar(i,4))                                                          +
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),(mhaar(i,2)+2*mhaar(i,4)),
mhaar(i,3),mhaar(i,4))                                             -
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),(mhaar(i,2)+2*mhaar(i,4)
),mhaar(i,3),mhaar(i,4));


        % Molde tipo 10 - Nove retângulos formando um quadrado, soma
os seis retângulos acima e abaixo da diagonal e subtrai os
retangulos da diagonal a -45°

        case 10

            carac_haar(i)                        =                 +
somaRetangulo(iim,mhaar(i,1),mhaar(i,2),mhaar(i,3),mhaar(i,4))     +
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),mhaar(i,2),mhaar(i,3),mhaa
r(i,4))                                                            -
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),mhaar(i,2),mhaar(i,3),mh
aar(i,4))                                                          +
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+mhaar(i,4)),mhaar(i,3),mhaa
r(i,4))                                                            -
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),(mhaar(i,2)+mhaar(i,4)),mh
aar(i,3),mhaar(i,4))                                               +
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),(mhaar(i,2)+mhaar(i,4)),
mhaar(i,3),mhaar(i,4))                                             -
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+2*mhaar(i,4)),mhaar(i,3),mh
aar(i,4))                                                          +
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),(mhaar(i,2)+2*mhaar(i,4)),
mhaar(i,3),mhaar(i,4))                                             +
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),(mhaar(i,2)+2*mhaar(i,4)
),mhaar(i,3),mhaar(i,4));


        % Molde tipo 11 - Doze retângulos formando um retangulo de
base 3 e altura 4, soma os retângulos acima e abaixo da diagonal e
subtrai os retangulos da diagonal dupla a -45°

        case 11

            carac_haar(i)                                          =
somaRetangulo(iim,mhaar(i,1),mhaar(i,2),mhaar(i,3),mhaar(i,4))     +
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),mhaar(i,2),mhaar(i,3),mhaa
r(i,4))                                                            -
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),mhaar(i,2),mhaar(i,3),mh
aar(i,4))                                                          +
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+mhaar(i,4)),mhaar(i,3),mhaa
r(i,4))                                                            -
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),(mhaar(i,2)+mhaar(i,4)),mh
aar(i,3),mhaar(i,4))                                               -
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),(mhaar(i,2)+mhaar(i,4)),
mhaar(i,3),mhaar(i,4))                                             -
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+2*mhaar(i,4)),mhaar(i,3),mh
aar(i,4))                                                          -
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),(mhaar(i,2)+2*mhaar(i,4)),
mhaar(i,3),mhaar(i,4))                                             +
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),(mhaar(i,2)+2*mhaar(i,4)
```

```
),mhaar(i,3),mhaar(i,4))                                                -
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+3*mhaar(i,4)),mhaar(i,3),mh
aar(i,4))                                                               +
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),(mhaar(i,2)+3*mhaar(i,4)),
mhaar(i,3),mhaar(i,4))                                                  +
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),(mhaar(i,2)+3*mhaar(i,4)
),mhaar(i,3),mhaar(i,4));


        % Molde tipo 12 - Doze retângulos formando um retangulo de
base 4 e altura 3, soma os retângulos acima e abaixo da diagonal e
subtrai os retangulos da diagonal dupla a 45°
        case 12
            carac_haar(i)                       =                   -
somaRetangulo(iim,mhaar(i,1),mhaar(i,2),mhaar(i,3),mhaar(i,4))      -
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),mhaar(i,2),mhaar(i,3),mhaa
r(i,4))                                                                 +
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),mhaar(i,2),mhaar(i,3),mh
aar(i,4))                                                               +
somaRetangulo(iim,(mhaar(i,1)+3*mhaar(i,3)),mhaar(i,2),mhaar(i,3),mh
aar(i,4))                                                               +
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+mhaar(i,4)),mhaar(i,3),mhaa
r(i,4))                                                                 -
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),(mhaar(i,2)+mhaar(i,4)),mh
aar(i,3),mhaar(i,4))                                                    -
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),(mhaar(i,2)+mhaar(i,4)),
mhaar(i,3),mhaar(i,4))                                                  +
somaRetangulo(iim,(mhaar(i,1)+3*mhaar(i,3)),(mhaar(i,2)+mhaar(i,4)),
mhaar(i,3),mhaar(i,4))                                                  +
somaRetangulo(iim,mhaar(i,1),(mhaar(i,2)+2*mhaar(i,4)),mhaar(i,3),mh
aar(i,4))                                                               +
somaRetangulo(iim,(mhaar(i,1)+mhaar(i,3)),(mhaar(i,2)+2*mhaar(i,4)),
mhaar(i,3),mhaar(i,4))                                                  -
somaRetangulo(iim,(mhaar(i,1)+2*mhaar(i,3)),(mhaar(i,2)+2*mhaar(i,4)
),mhaar(i,3),mhaar(i,4))                                                -
somaRetangulo(iim,(mhaar(i,1)+3*mhaar(i,3)),(mhaar(i,2)+2*mhaar(i,4)
),mhaar(i,3),mhaar(i,4));


    end
end


% ###########################################################


function soma = somaRetangulo(iim,x,y,largura,altura)
% Calcula a soma dos pixels do retângulo
%
% Entrada - iim: Matriz da mesma dimensão da imagem com os valores
Integral
%                da Imagem para todos os pixels
```

```
%           x,y: Coordenadas do ponto superior esquerdo do retângulo
%           largura, altura do retângulo
%
% Saída   - Soma das intensidades dos pixels na região especificada.
%
% Autor: Fernando Otávio Fonseca
% Data: 01/11/2015

soma = iim(y,x) + iim(y + altura,x + largura) - iim(y,x + largura) -
iim(y + altura,x);

% ############################################################

function chaar = montaCaracteristicas(diretorio, ext, imlarg, imalt,
mascara)
% Calcula o conjunto de características de Haar
%
% Entrada - listarq: listagem de arquivos
%           ext: extenção dos arquivos de imagem
%           imlarg: largura padrão das imasgens
%           imalt: altura padrão das imagens
%           mascara: máscara das características Haar selecionadas
%
% Saída   - Matriz com as características do conjunto de imagens
%
% Autor: Fernando Otávio Fonseca
% Data: 10/11/2015

% Cria lista dos arquivos
listarq = dir(strcat(diretorio,ext));

% Inicializa os conjuntos de características
chaar = zeros(size(listarq,1),size(mascara,1));

% Calcula o valor das características para as amostras do diretório
for i=1:size(listarq,1)
    im = imread([diretorio  listarq(i).name]);
```

```
    % Garante que a escala seja padronizada
    % im = imresize(im, [imlarg imalt]);

    % Converte para escala de cinza, caso a imagem seja colorida
    if size(im,3)>1
        im = rgb2gray(im);
    end
    im = double(im);
    chaar(i,:) = determinaHaar(integralImagem(im),mascara);
end

% ############################################################
```

function mascara = montaMascara(larg, alt, escala, passohor, passover, borda)

```
% Calcula as máscaras das características de Haar
%
% Máscaras para as bases escolhidas
% Entrada - larg, alt: dimensões da imagem
%                    escala: passo de crescimento da escala das características (1, 2, 3, 4...))
%           passohor, passover: espaçamento entre caracterísitcas na
%                                    horizontal e vertical (1, 2, 3, 4...)
%
% Saida   - Vetor com as características possiveis
%
% Autor: Fernando Otávio Fonseca
% Data: 01/11/2015
%

% Define os moldes das características [Largura Altura Tipo]
% moldes = [3 1 1;1 2 2]; % Usando 2 modelos de Haar
% moldes = [3 1 1;1 2 2;2 1 3]; % Usando 3 modelos de Haar
% moldes = [3 1 1;1 2 2;2 1 3;2 2 4]; % Usando 4 modelos de Haar
% moldes = [3 1 1;1 2 2;2 1 3;1 3 5]; % Usando 3 modelos de Haar e 1 modelo estendido
% moldes = [3 1 1;1 2 2;2 1 3;1 3 5;1 4 6;4 1 7]; % Usando 3 modelos de Haar e 3 modelos estendidos
```

```
% moldes = [3 1 1;1 2 2;2 1 3;2 2 4;1 3 5;1 4 6;4 1 7]; % Usando 4 modelos de Haar e 3 modelos estendidos
% moldes = [3 1 1;1 2 2;2 1 3;1 3 5;1 4 6;4 1 7;3 3 8]; % Usando 3 modelos de Haar e 4 modelos estendidos
% moldes = [3 1 1;1 2 2;2 1 3;1 3 5;1 4 6;4 1 7;3 3 8;3 3 9;3 3 10]; % Usando 3 modelos de Haar e 6 modelos estendidos
% moldes = [3 1 1;1 2 2;2 1 3;2 2 4;1 3 5;1 4 6;4 1 7;3 3 8;3 3 9;3 3 10]; % Usando 4 modelos de Haar e 6 modelos estendidos
% moldes = [3 1 1;1 2 2;2 1 3;1 3 5;1 4 6;4 1 7;3 3 8;3 3 9;3 3 10;3 4 11;4 3 12]; % Usando 3 modelos de Haar e 8 modelos estendidos
% moldes = [3 1 1;1 2 2;2 1 3;2 2 4;1 3 5;1 4 6;4 1 7;3 3 8;3 3 9;3 3 10;3 4 11;4 3 12]; % Usando 4 modelos de Haar e 8 modelos estendidos
moldes = [3 1 1;1 2 2;1 3 5;1 4 6;4 1 7;3 3 8;3 3 9;3 3 10;3 4 11;4 3 12]; % Usando 2 modelos de Haar e 8 modelos estendidos

% Pré-aloca a matriz temporária de características.
temp = zeros (50000,5);

% Inicializa o contador das Características
ct1 = 0;

for m1=1:size(moldes,1)
    % Para cada tipo da base faz
    for cp1=moldes(m1,1):escala:floor(larg/moldes(m1,1))
        % Dimensiona a característica na horizontal
        for at1=moldes(m1,2):escala:floor(alt/moldes(m1,2))
            % Dimensiona a característica na vertical
            for x1=1+borda:passohor:larg+1 - cp1*moldes(m1,1) -borda
                % Desloca a característica ao longo do eixo x sem borda
                for y1=2:passover:alt+1 - at1*moldes(m1,2) -borda
                    % Desloca a característica ao longo o eixo y sem borda
                    ct1 = ct1 + 1;
                    temp(ct1,:) = [x1 y1 cp1 at1 moldes(m1,3)];
                end
            end
        end
    end
end
```

```
% Armazena o vetor de saída
mascara = temp(1:ct1,:);

% ###########################################################

% Programa de Treinamento de Detecção de Faces baseado no Viola-Jones
%
% Autor: Fernando Otávio Fonseca
% Data: 15/11/2015
%
close('all');
clc;
clear;

tic

% Cria a Máscara das Características
imlarg = 24;
imalt = 24;
escala = 2; % Define o crescimento da escala das características
passohor = 2; % Define o espaçamento entre características na horizontal
passover = 2; % Define o espaçamento entre características na vertical
borda = 2; % Se usa borda = 0; se exclui a borda do cálculo = 1
mascara = montaMascara(imlarg,imalt,escala,passohor,passover,borda);

% Diretório das amostras positivas de faces e negativas sem faces
dirfaces = 'C:\Users\Fernando\Documents\MATLAB\Detector Faces VJ\TreinaFaces\';
dirnfaces = 'C:\Users\Fernando\Documents\MATLAB\Detector Faces VJ\TreinaNFaces\';

% Extensão do arquivo
ext = '*.GIF';

carga = 1;
```

```
if carga == 1
    load('faces.mat');
    load('nonfaces.mat');
    disp ('Base de Faces e Não Faces Carregadas');
else
    disp ('Montando Base de Faces e Não Faces...');
    % Calcula o valor das características para as amostras positivas
    hfaces = montaCaracteristicas(dirfaces,ext,imlarg,imalt,mascara);
    disp ('Características criadas para amostras positivas');
    save('faces.mat', 'hfaces', '-v7.3');

    % Calcula o valor das características para as amostras negativas
    hnfaces = montaCaracteristicas(dirnfaces,ext,imlarg,imalt,mascara);
    disp ('Características criadas para amostras negativas');
    save('nonfaces.mat', 'hnfaces', '-v7.3');
end

% numfolhas = [2 10 25 25 50 50 50]; % Número de folhas por estágio
% numfolhas = [3 10 20 30 50 55 75 90 100]; % Número de folhas por estágio
% numfolhas = [3 16 21 39 33 44 50 51 56 71 80 103 111 102 135 137 140 169 177 182 211 213]; % Número de folhas por estágio HaarCascade
% numfolhas = [2 3 3]; % Teste do passa tudo
% numfolhas = [2 3 3 25 25 25]; % Teste do passa médio???
% numfolhas = [2 3 3 25 100]; % Teste do passa pouco
numfolhas = [3 25 150]; % Teste do passa pouco modificado
estagios = size(numfolhas,2); % Número de estágios do classificador
disp('Iniciando o treinamento...');
indnfaces = [500 1500 2500];
% indnfaces = [400 800 1100 1100 1500 1500 1500];
fprintf('Total de amostras de faces : %d.\n',size(hfaces,1));
fprintf('Total de amostras de não faces : %d.\n',size(hnfaces,1));
fprintf('Distribuição de não faces pelos estágios : %d.\n',indnfaces);
ctnfaces = 1;
```

```
for i=1:estagios
    % Distribui as imagens em cada estágio
     hnfaces2 = hnfaces(ctnfaces:ctnfaces+indnfaces(i)-1,:);
     % Aciona o classificador forte de cada estágio
    c(i) = classificador(hfaces,hnfaces2,mascara,numfolhas(i));
    fprintf('Estágio %d completo.\n',i);
    estrutura.class = c(i);
    save(strcat('estrutura',int2str(i),'.mat'), 'estrutura');
    fprintf('Configurações salvas em %s.\n',datestr(now));
    % fprintf('Próximo estágio usará %d amostras de não faces.
\n',size(indnfaces,1));
    % hnfaces2 = hnfaces2(indnfaces,:);
    ctnfaces = ctnfaces + indnfaces(i);
end
estrutura.mascara = mascara;
estrutura.class = c;
disp('Treinamento finalizado!');
fprintf('Tamanho da Mascara de Características: %d.
\n',size(mascara,1));
save('estrutura.mat', 'estrutura');
disp('Configuração do Treinamento Salva.');

toc
```

7.2. Detecção Multiescala Adaboost com features Haar

```
function regioes = multiEscala(im, escalainicial, passoescala, passox, passoy)

% Busca os padrões de face na imm variando a escala
%
% Entrada - im: imagem na qual será feita a varredura
%           escalainicial: escala inicial da imm
%           passoescala: fator de redução na escala
%           passox,passoy: passos de coordenadas
% Saída    - Posições dos retângulos das regiões que classificaram positivas
%
% Autor: Fernando Otávio Fonseca
% Data: 23/11/2015
%

% Carrega as características treinadas
load('estrutura.mat');
mascara = estrutura.mascara; % Máscara de características
classificador = estrutura.class; % Parâmetros do classificador salvos no treinamento

im = double(im);

% Dimensões da janela de varredura
larg = 24;
alt = 24;

% Inicializa a saída
regioes = [];

% Número de estágios dos classificadores
numestagios = size(classificador,2);

% Incremento de posição da janela de varredura
% passox = 2;
```

```
% passoy = 2;

% Escala inicial
escala=escalainicial;
%Escala minima
min_escala = 0.1;
% min_escala = 1;

% Percorre a imagem até a escala mínima
while escala>=min_escala
    % Redimensionar a imagem
    imre = imresize(im,escala);
    lre = size(imre,2);
    alre = size(imre,1);

    % Integral da imm para todos os pixels
    ii_imr = integralImagem(imre);

    faces = [];

    % Movimenta a janela usando os passos definidos
    for i=2:passox:lre-larg
        for j=2:passoy:alre-alt
            acheiface = true;
            for estagio=1:numestagios
                % Carrega os indicadores do estágio atual
                threshold = classificador(estagio).threshold;
                alfa = classificador(estagio).alfa';
                indices = classificador(estagio).folhas(:,1);
                folha_thres = classificador(estagio).folhas(:,2);
                polar = classificador(estagio).folhas(:,3);
                folha_thres_polar = polar.*folha_thres;

                % Carrega as máscaras dos classificadores
                fc = zeros (size(indices,1),5);
                for i2=1:size(indices,1)
                    fc(i2,:) = mascara(indices(i2),:);
```

```
                end

                r = 0;
                % Aplica o classificador na região da imagem
                for k=1:size(fc,1)
                    % fs = calculaHaar(ii_imr, fc(k, 1)+i, fc(k,
2)+j , fc(k, 3) , fc(k, 4) , fc(k,5));
                    temp = fc(k,:);
                    temp(1) = temp(1)+i;
                    temp(2) = temp(2)+j;
                    fs = determinaHaar(ii_imr,temp);
                    r               =               r               +
alfa(k)*(polar(k).*fs<folha_thres_polar(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
                    acheiface = false;
                    break;
                end
            end
            % Se a região classificou positivamente para todos os
estágios,
            % guarda as coordenadas do retângulo da região atual
            if (acheiface == true)
                faces = [faces; i,j,larg,alt];
            end
        end
    end

    % Se uma ou mais regiões foram classificadas positivamente na
escala
    % atual, guarda as coordenadas com a devida correção de escala
    fprintf('Verificação da Escala %d concluída.\n',escala);
    if size (faces,1)> 0
        regioes = [regioes; faces/escala];
```

```
        fprintf('Encontrada %d regiões.\n',size(faces,1));
    end
    % escala = escala*1/passoescala;
    escala = escala*passoescala;
end

% ###########################################################

function regioes = detectaRegioes(iim,estrutura,options)

% Controla a detecção em escalas diferentes
%
% Entrada - im: imagem na qual será feita a varredura
%           estrutura: estrutura do treinamento
%           options: opções de escala%
% Saída    - Posições dos retângulos das regiões que classificaram
positivas
%
% Autor: Fernando Otávio Fonseca
% Data: 26/11/2015
%

% Calcula escala mais inferior
[iimlin,iimcol] = size(iim);
imbcol = 24; % largura dos blocos de treinamento
imblin = 24; % altura dos blocos de treinamento

escalhor = iimcol/imbcol;
escalvert = iimlin/imblin;
if(escalvert < escalhor),
    escalaini =  escalvert;
else
    escalaini = escalhor;
end

% Inicializa o vetor das coordenadas das regiões identificadas [x y
largura altura]
regioes = zeros(100,4);
```

```
n=0;

% Calcula o máximo de iterações de busca nesta escala
itt = ceil(log(1/escalaini)/log(options.ScaleUpdate));

% Percorre todas as escalas da imagem
for i=1:itt
    % Calcula a escala desta passagem
    escala = escalaini*options.ScaleUpdate^(i-1);

    % Calcula o tamanho da janela que vai decrescendo
    larg = floor(imbcol*escala);
    alt = floor(imblin*escala);

    % Espaço de pesquisa na imagem
    passo = floor(max(escala,2));

    % Cria vetores com as coordenadas da imagem possíveis na escala
atual
    [x,y]    =    ndgrid(0:passo:(iimcol-larg),0:passo:(iimlin-alt));
x=x(:); y=y(:);
    %  [x,y]  =  ndgrid(0:passo:(iimcol-larg-1),0:passo:(iimlin-alt-
1)); x=x(:); y=y(:);

    % Se não houver coordenadas no passo atual, continua para uma
nova
    % escala
    if(isempty(x))
        % Mostra a escala e o número de regiões detectadas
        if(options.Verbose)
            disp(['Escala : ' num2str(escala) ' regiões detectadas :
' num2str(n) ])
        end
        continue;
    end

    % Busca regiões na imagem para a escala presente
    [x,y] = escalaUnica( x, y, escala, iim, larg, alt, estrutura);
```

```
    % Verifica se existem coordenadas de busca
    for k=1:length(x);
        n = n + 1;
        regioes(n,:) = [x(k) y(k) larg alt];
    end

    % Mostra a escala e o número de regiões detectadas
    if(options.Verbose)
        disp(['Escala : ' num2str(escala) ' regiões detectadas : ' num2str(n) ])
    end
end

% Elimina as linhas a mais da matriz de regiões de acordo com o que foi detectado
regioes = regioes(1:n,:);

% ###########################################################

function [x,y] = escalaUnica( x, y, escala, iim, larg, alt, estrutura)

% Executa a detecção em escala única
%
% Entrada - im: imagem na qual será feita a varredura
%           estrutura: estrutura do treinamento
%           options: opções de escala%
% Saída   - Coordenadas de x e y que passaram por todos os estágios
%
% Autor: Fernando Otávio Fonseca
% Data: 26/11/2015
%

% Armazena as caracterísitcas treinadas
mascara = estrutura.mascara; % Máscara de características
classificador = estrutura.class; % Parâmetros do classificador salvos no treinamento

% acheiface = true;
```

```
for estagio=1:size(classificador,2)
    % Carrega os indicadores do estágio atual
    threshold = classificador(estagio).threshold;
    alfa = classificador(estagio).alfa';
    indices = classificador(estagio).folhas(:,1);
    folha_thres = classificador(estagio).folhas(:,2);
    polar = classificador(estagio).folhas(:,3);
    folha_thres_polar = polar.*folha_thres;

    % Carrega as máscaras dos classificadores
    fc = mascara(indices,:);
    % &&&&&&&&&&&&&&&&&&& VAMOS MELHORAR &&&&&&&&&&&&&&&&
    % fc = zeros (size(indices,1),5);
    % for i2=1:size(indices,1)
    %     fc(i2,:) = mascara(indices(i2),:);
    % end

    rsoma = zeros(size(x)); % Armazena a soma dos classificadores
    folha = zeros(size(x)); % Vetor para armazenar os índices de
folhas
    % Aplica o classificador na região da imagem
    for k=1:size(fc,1)
        % fs = calculaHaar(ii_imr, fc(k, 1)+i, fc(k, 2)+j , fc(k, 3)
, fc(k, 4) , fc(k,5));
        folhas = fc(folha+k,:);
        folhas(:,1:4) = floor(folhas(:,1:4)*escala);
        folhas(:,1) = floor(folhas(:,1)+x);
        folhas(:,2) = floor(folhas(:,2)+y);

        % Valida as dimensões geradas
        % erroX = folhas(:,1) > size(iim,2);
        % if ~isempty(erroX==1)
        %         disp('Erro nas dimensões de coluna calculadas.
Coordenadas ignoradas.');
        %    folhas = folhas(~erroX);
        % end
        % erroY = folhas(:,2) > size(iim,1);
        % if ~isempty(erroY==1)
```

```
        %         disp('Erro nas dimensões de linha calculadas.
Coordenadas ignoradas.');
        %     folhas = folhas(~erroY);
        % end
        fs = determinaHaar(iim,folhas);
        rsoma = rsoma + alfa(k)*(polar(k).*fs<folha_thres_polar(k));
    end

    % Se o valor computado for menor do que o threshold do estagio
atual,
    % a região não classificou positivamente para o estágio atual
    naopassou = rsoma < threshold;
    % Portanto essa região não passará pelos próximos estágios
    x = x(~naopassou);
    y = y(~naopassou);
    % E se todos as coordenadas falham passa para o próximo estágio
    if(isempty(x))
        % acheiface = false;
        break;
    end
end

% ###########################################################

function resultado = classificadorBase(amostras)

% Determina a característica com menor erro nos classificadore fraco
%
% Entrada - amostras = amostras.valores: Conjunto de valores das
Características
%                                          de Haar calculadas para
exemplos
%                                       positivos e negaitvos
%                         amostras.numamp, numamn: Número de exemplos
positivos
%                                          e negativos
%                       amostras.wi: pesos de cada amostra
%                          amostras.yi: 0 para exemplos negativos e 1
para
```

```
%                                   positivos
% Saída    - Threshold, Polaridade, índice e Erro da melhor
%            característica e Resultado da classificação com a
%            característica escolhida para os outros exemplos
%
% Autor: Fernando Otávio Fonseca
% Data: 01/11/2015

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

% Varre todas as caracteristicas em busca daquela com o menor erro
de
% classificação
for i = 1:amostras.numch

    chaar = amostras.valores(:,i);

    % Ordena as características de acordo com seus valores
    [chordenado, indchaar] = sort(chaar);

    % Ordena os pesos e os 'ys' de acordo com a ordenação anterior
    sw = amostras.wi(indchaar);
    sy = amostras.yi(indchaar);

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

    % Erros dos exemplos positivos e negativos
    erropos = somapesopos + amostras.twneg - somapesoneg;
    erroneg = somapesoneg + amostras.twpos - somapesopos;

    % Escolhe o valor que melhor separa as duas classes
    erro = min(erropos, erroneg);
```

```
    [erro, ind] = min(erro);
    result = zeros(amostras.numamp + amostras.numamn,1);

    % Classifica os exemplos com o threshold escolhido
    if erropos(ind) <= erroneg(ind)
        result(ind + 1:end) = 1;
        result(indchaar) = result;
        p = -1;
    else
        result(1:ind) = 1;
        result(indchaar) = result;
        p = 1;
    end

    % Se o erro for o menor encontrado, guarda a característica
atual
    if erro < resultado.erro
        resultado.erro = erro;
        resultado.threshold = chordenado(ind);
        resultado.polaridade = p;
        resultado.indice = i;
        resultado.class_result = result;
    end

end



% ############################################################

function regioes = detectaRegioes(iim,estrutura,options)

% Controla a detecção em escalas diferentes
%
% Entrada - im: imagem na qual será feita a varredura
%           estrutura: estrutura do treinamento
%           options: opções de escala%
% Saída    - Posições dos retângulos das regiões que classificaram
positivas
```

```
%
% Autor: Fernando Otávio Fonseca
% Data: 26/11/2015
%

% Calcula escala mais inferior
[iimlin,iimcol] = size(iim);
imbcol = 24; % largura dos blocos de treinamento
imblin = 24; % altura dos blocos de treinamento

escalhor = iimcol/imbcol;
escalvert = iimlin/imblin;
if(escalvert < escalhor),
    escalaini =  escalvert;
else
    escalaini = escalhor;
end

% Inicializa o vetor das coordenadas das regiões identificadas [x y
largura altura]
regioes = zeros(100,4);
n=0;

% Calcula o máximo de iterações de busca nesta escala
itt = ceil(log(1/escalaini)/log(options.ScaleUpdate));

% Percorre todas as escalas da imagem
for i=1:itt
    % Calcula a escala desta passagem
    escala = escalaini*options.ScaleUpdate^(i-1);

    % Calcula o tamanho da janela que vai decrescendo
    larg = floor(imbcol*escala);
    alt = floor(imblin*escala);

    % Espaço de pesquisa na imagem
    passo = floor(max(escala,2));
```

```
    % Cria vetores com as coordenadas da imagem possíveis na escala
atual
    [x,y]    =    ndgrid(0:passo:(iimcol-larg),0:passo:(iimlin-alt));
x=x(:); y=y(:);
    %  [x,y]  =  ndgrid(0:passo:(iimcol-larg-1),0:passo:(iimlin-alt-
1)); x=x(:); y=y(:);


    % Se não houver coordenadas no passo atual, continua para uma
nova
    % escala
    if(isempty(x))
        % Mostra a escala e o número de regiões detectadas
        if(options.Verbose)
            disp(['Escala : ' num2str(escala) ' regiões detectadas :
' num2str(n) ])
        end
        continue;
    end


    % Busca regiões na imagem para a escala presente
    [x,y] = escalaUnica( x, y, escala, iim, larg, alt, estrutura);


    % Verifica se existem coordenadas de busca
    for k=1:length(x);
        n = n + 1;
        regioes(n,:) = [x(k) y(k) larg alt];
    end


    % Mostra a escala e o número de regiões detectadas
    if(options.Verbose)
        disp(['Escala : ' num2str(escala) ' regiões detectadas : '
num2str(n) ])
    end
end


% Elimina as linhas a mais da matriz de regiões de acordo com o que
foi detectado
regioes = regioes(1:n,:);
```

7.3. Treinamento com máquina SVM e filtros de características

```
% Programa de Treinamento de Detecção de Faces usando SVM
%
% Autor: Fernando Otávio Fonseca
% Data: 19/01/2016
%
% Utiliza partes do programa exemplo do prof. Vitor Hugo
%
% Incluída variação 't' (tipo de kernel) na avaliação dos modelos de
% treinamento em 23/01/2016.

close('all');
clc;
clear;
tic

% Cria a Máscara das Características
janela = 24;
imlarg = 24;
imalt = 24;
escala = 2; % Define o crescimento da escala das características
passohor = 2; % Define o espaçamento entre características na
horizontal
passover = 2; % Define o espaçamento entre características na
vertical
borda = 1; % Se usa borda = 0; se exclui a borda do cálculo = 1
mascara = montaMascara(imlarg,imalt,escala,passohor,passover,borda);

% Definição dos diretórios de localização das imagens (treinamento,
% validação, teste e padrões

% PaternPath = uigetdir('C:\Trab\Mestrado\UFF\Aprendizado de
Máquina\Trabalho Aprendizado\PaternDatabase', 'Defina o caminho da
base de fotos Padronizadas' );
TrainFacesPath =
uigetdir('C:\Users\Fernando\Documents\MATLAB\Detector Faces
VJ\TreinaFaces', 'Defina o caminho da base de imagens de faces de
Treinamento' );
```

```
TrainFacesPath = strcat(TrainFacesPath,'\');
TrainNFacesPath = uigetdir('C:\Users\Fernando\Documents\MATLAB\Detector Faces VJ\TreinaNFaces', 'Defina o caminho da base de imagens sem faces de Treinamento' );
TrainNFacesPath = strcat(TrainNFacesPath,'\');
ValidFacesPath = uigetdir('C:\Users\Fernando\Documents\MATLAB\Detector Faces VJ\ValidaFaces', 'Defina o caminho da base de imagesn de faces de Validação');
ValidFacesPath = strcat(ValidFacesPath,'\');
ValidNFacesPath = uigetdir('C:\Users\Fernando\Documents\MATLAB\Detector Faces VJ\ValidaNFaces', 'Defina o caminho da base de imagens sem faces de Validação');
ValidNFacesPath = strcat(ValidNFacesPath,'\');

% Extensão do arquivo
ext = '*.GIF';
% ext = '*.png';
% ext = '*.pgm';

disp ('Base de Faces e Não Faces em preparação...');
% Calcula o valor das características para as amostras positivas
matrizFaces = montaCaracteristicas(TrainFacesPath,ext,imlarg,imalt,mascara);
eT = size(matrizFaces,1);
% Na codificação da saídas são usados dois bits (face e não face)
saidaTreina = ones(eT,2);
% O primeiro bit é aquele com informação da face
% Como é melhor trabalhar com -1 do que 0, o bit com 0 receberá -1
saidaTreina(:,2) = -1;
disp ('Características de treinamento criadas para amostras positivas');
% save('faces.mat', 'hfaces','saidaTreina','-v7.3');

% Imagens de Não Faces
matrizNFaces = montaCaracteristicas(TrainNFacesPath,ext,imlarg,imalt,mascara);
eT = size(matrizNFaces,1) + eT;
% O segundo bit é aquele com informação da não face
```

```
saidaT = ones(size(matrizNFaces,1),2);
saidaT(:,1) = -1;
saidaTreina = [saidaTreina ; saidaT];
disp ('Características de treinamento criadas para amostras negativas');

% Chama a função que monta a matriz de pixels das imagens de validação
% Imagens de Faces
matrizFacesValida = montaCaracteristicas(ValidFacesPath,ext,imlarg,imalt,mascara);
eV = size(matrizFacesValida,1);
% Monta a matriz de saída
saidaValida = ones(eV,2);
saidaValida(:,2) = -1;
disp ('Características de validação criadas para amostras positivas');

% Imagens de Não Faces
matrizNFacesValida = montaCaracteristicas(ValidNFacesPath,ext,imlarg,imalt,mascara);
eV = size(matrizNFacesValida,1) + eV;
% Monta a matriz de saída para validação
saidaV = ones(size(matrizNFacesValida,1),2);
saidaV(:,1) = -1;
saidaValida = [saidaValida ; saidaV];
disp ('Características de validação criadas para amostras negativas');

% Junta a Matrizes de Treinamento e Validação em uma única matriz
matrizPixels = [matrizFaces;matrizNFaces;matrizFacesValida;matrizNFacesValida];

% Constroi as entradas de treinamento e validação
% matrizPixels = matrizPixels';

[Entrada_Normalizada,Limites_Entradas] = mapminmax(double(matrizPixels')); % Normaliza as entradas
Entrada_Normalizada = Entrada_Normalizada';
```

```
entradasT = Entrada_Normalizada(1:eT,:); % As entradas do treinamento são eT primeiras linhas
entradasV = Entrada_Normalizada(eT+1:eT+eV,:); % As entradas da validação são eV últimas linhas

[numero_de_padroesT,numero_de_entradas] = size(entradasT);

[numero_de_padroesV,numero_de_entradas_valida] = size(entradasV);

save('faceSVM.mat',
'entradasT','entradasT','saidaTreina','saidaValida','Limites_Entradas','-v7.3');

% Treinamento e simulação com Máquina de Vetor Suporte
% Especificacao dos limites de variacao para os parametros

C_minimo = 1;
C_maximo = 1000000;

gamma_minimo = 0.001;
gamma_maximo = 1;

t_minimo = 0;
t_maximo = 4;

t = t_minimo;

validador = struct([]);
taxa_de_acertos = zeros(100,1);
cont = 1;

while t <= t_maximo

    C = C_minimo;

    % Treinamento e Teste com 'MLP' kernel (valores de kernel na matriz instância treinamento), t=4
     if t==4
          entradasT = [(1:eT)' entradasT*entradasT'];
```

```
            entradasV = [(1:eV)' entradasV*entradasV'];
        end

    while C <= C_maximo

        gamma = gamma_minimo;

        while gamma <= gamma_maximo

            for i = 1:2

                % Definicao de alguns parametros da biblioteca de
SVM. Maiores
                % detalhes podem ser encontrados no link
                % http://www.csie.ntu.edu.tw/~cjlin/libsvm/

                options = ['-s 0 -t ' num2str(t) ' -g '
num2str(gamma) ' -c ' num2str(C)];
                modelo{cont}{i}                                  =
svmtrain(saidaTreina(:,i),entradasT,options);

            end

            fprintf('Valor do kernel (t): %d \n',t);
            validador(cont).kernel = t;
            fprintf('Valor do C : %d \n',C);
            validador(cont).C = C;
            fprintf('Valor do gamma : %6.3f \n',gamma);
            validador(cont).gamma = gamma;

            % Avaliacao do modelo para o conjunto de treinamento

            [porcentagem_de_acertos_treino(cont)]                =
avalia_modelo_SVM(modelo{cont},entradasT,saidaTreina);

            % Avaliacao do modelo para o conjunto de validacao

            [taxa_de_acertos(cont)]                              =
avalia_modelo_SVM(modelo{cont},entradasV,saidaValida);
```

```
            fprintf('Porcentagem de acertos de Validação : %6.2f
\n',taxa_de_acertos(cont));

            gamma = gamma*10;
            cont = cont + 1;

        end

        C = C*10;

    end

    t = t + 1;

end

% Escolha da melhor estrutura baseado no erro para o conjunto de
validacao

[taxaSV,indice] = max(taxa_de_acertos);

SVM = modelo{indice};

disp('Treinamento SVM finalizado!');
fprintf('Tamanho da Mascara de Características: %d.
\n',size(mascara,1));
fprintf('Porcentagem de acertos de Validação máxima : %6.2f
\n',taxa_de_acertos(indice));
fprintf('Valor do kernel (t): %d \n',validador(indice).kernel);
fprintf('Valor do C : %d \n',validador(indice).C);
fprintf('Valor do gamma : %6.3f \n',validador(indice).gamma);
save('estruturaSVMHaar.mat',                                    'SVM',
'Limites_Entradas','mascara','janela');
disp('Configuração do Treinamento Salva.');
toc

% ###########################################################
```

```
% Programa de Treinamento de Detecção de Faces usando eigenfaces treinadas com SVM
%
% Autor: Fernando Otávio Fonseca
% Data: 25/01/2016
%

close('all');
clc;
clear;
tic

% Definição dos diretórios de localização das imagens (treinamento,
% validação, teste e padrões

% PaternPath = uigetdir('C:\Trab\Mestrado\UFF\Aprendizado de Máquina\Trabalho Aprendizado\PaternDatabase', 'Defina o caminho da base de fotos Padronizadas' );
TrainFacesPath = uigetdir('C:\Users\Fernando\Documents\MATLAB\Detector Faces VJ\TreinaFaces', 'Defina o caminho da base de imagens de faces de Treinamento' );
TrainFacesPath = strcat(TrainFacesPath,'\');
TrainNFacesPath = uigetdir('C:\Users\Fernando\Documents\MATLAB\Detector Faces VJ\TreinaNFaces', 'Defina o caminho da base de imagens sem faces de Treinamento' );
TrainNFacesPath = strcat(TrainNFacesPath,'\');
ValidFacesPath = uigetdir('C:\Users\Fernando\Documents\MATLAB\Detector Faces VJ\ValidaFaces', 'Defina o caminho da base de imagesn de faces de Validação');
ValidFacesPath = strcat(ValidFacesPath,'\');
ValidNFacesPath = uigetdir('C:\Users\Fernando\Documents\MATLAB\Detector Faces VJ\ValidaNFaces', 'Defina o caminho da base de imagens sem faces de Validação');
ValidNFacesPath = strcat(ValidNFacesPath,'\');

% Extensão do arquivo
ext = '*.gif';
% ext = '*.png';
```

```
% ext = '*.pgm';

% Quantidade de linhas das imagens
linha = 24;
coluna = 24;
tamimag = linha*coluna;

% Chama a função que monta a matriz autovalores das imagens de treinamento
[total_faces, media, eigenfaces, vetor_carac] = montaEigenfaces(TrainFacesPath,TrainNFacesPath,ext,tamimag);

eT = size(vetor_carac,2);

% Na codificação da saídas são usados dois bits (face e não face)
saidaTreina = ones(eT,2);
% O primeiro bit é aquele com informação da face
% Como é melhor trabalhar com -1 do que 0, o bit com 0 receberá -1
% Imagens de Faces
saidaTreina(1:total_faces,2) = -1;
% Imagens de Não Faces
saidaTreina(total_faces+1:eT,1) = -1;

% Monta a matriz autovalores das imagens de validação
% Imagens de Faces
listarq = dir(strcat(ValidFacesPath,ext));
tot_arq_fac = size(listarq,1);

mat_autoval = [];
for i = 1 : tot_arq_fac

    % Extrai as caracterísitcas Eigen das imagens uma a uma
    [imgproj] = extraiEigenfaces(eigenfaces,media,tamimag,listarq(i).name,ValidFacesPath);

    % Adiciona a matriz de pixels
    mat_autoval = [mat_autoval imgproj];
```

```
end

% Imagens de Não Faces
listarq = dir(strcat(ValidNFacesPath,ext));
tot_arq_nfac = size(listarq,1);

for i = 1 : tot_arq_nfac

     % Extrai as caracterísitcas Eigen das imagens uma a uma
    [imgproj] = extraiEigenfaces(eigenfaces,media,tamimag,listarq(i).name,ValidNFacesPath);

    % Adiciona a matriz de pixels
    mat_autoval = [mat_autoval imgproj];

end

% Define entradas de validação
eV = size(mat_autoval,2);

% Monta a matriz de saída
saidaValida = ones(eV,2);
% Imagens de Faces
saidaValida(1:tot_arq_fac,2) = -1;
% Imagens de Não Faces
saidaValida(tot_arq_fac+1:eV,1) = -1;

% Junta a Matrizes de Treinamento e Validação em uma única matriz
matrizPixels = [vetor_carac mat_autoval];

[Entrada_Normalizada,Limites_Entradas] = mapminmax(matrizPixels); % Normaliza as entradas
Entrada_Normalizada = Entrada_Normalizada';

entradasT = Entrada_Normalizada(1:eT,:); % As entradas do treinamento são eT primeiras linhas
entradasV = Entrada_Normalizada(eT+1:eT+eV,:); % As entradas da validação são eV últimas linhas
```

```
% Treinamento e simulação com Máquina de Vetor Suporte
% Especificacao dos limites de variacao para os parametros

C_minimo = 100;
C_maximo = 1000000;

gamma_minimo = 0.001;
gamma_maximo = 1;

t_minimo = 0;
t_maximo = 4;

t = t_minimo;

validador = struct([]);
taxa_de_acertos = zeros(100,1);
cont = 1;

while t <= t_maximo

    C = C_minimo;

     % Treinamento e Teste com 'MLP' kernel (valores de kernel na
matriz instância treinamento), t=4
     if t==4
          entradasT = [(1:eT)' entradasT*entradasT'];
          entradasV = [(1:eV)' entradasV*entradasV'];
     end

    while C <= C_maximo

        gamma = gamma_minimo;

        while gamma <= gamma_maximo

            for i = 1:2
```

```
                % Definicao de alguns parametros da biblioteca de
SVM. Maiores
                % detalhes podem ser encontrados no link
                % http://www.csie.ntu.edu.tw/~cjlin/libsvm/

                options = ['-s 0 -t ' num2str(t) ' -g '
num2str(gamma) ' -c ' num2str(C)];
                modelo{cont}{i}                                  =
svmtrain(saidaTreina(:,i),entradasT,options);

            end

            fprintf('Valor do kernel (t): %d \n',t);
            validador(cont).kernel = t;
            fprintf('Valor do C : %d \n',C);
            validador(cont).C = C;
            fprintf('Valor do gamma : %6.3f \n',gamma);
            validador(cont).gamma = gamma;

            % Avaliacao do modelo para o conjunto de treinamento

            [porcentagem_de_acertos_treino(cont)]                =
avalia_modelo_SVM(modelo{cont},entradasT,saidaTreina);

            % Avaliacao do modelo para o conjunto de validacao

            [taxa_de_acertos(cont)]                              =
avalia_modelo_SVM(modelo{cont},entradasV,saidaValida);

            fprintf('Porcentagem de acertos de Validação : %6.2f
\n',taxa_de_acertos(cont));

            gamma = gamma*10;
            cont = cont + 1;

        end

        C = C*10;
```

```
    end

    t = t + 1;

end

% Escolha da melhor estrutura baseado no erro para o conjunto de
validacao

[taxaSV,indice] = max(taxa_de_acertos);

SVM = modelo{indice};

disp('Treinamento SVM finalizado!');
fprintf('Porcentagem de acertos de Validação máxima : %6.2f
\n',taxa_de_acertos(indice));
fprintf('Valor do kernel (t): %d \n',validador(indice).kernel);
fprintf('Valor do C : %d \n',validador(indice).C);
fprintf('Valor do gamma : %6.3f \n',validador(indice).gamma);
save('treinaSVMEigen.mat',                                    'SVM',
'Limites_Entradas','eigenfaces','media','tamimag','linha','coluna');
disp('Configuração do Treinamento Salva.');
toc



% ############################################################

function [imgproj] = extraiEigenfaces(eigenfaces, media, tamimag,
nome, diretorio)

% Processa e extrai o vetor de características Eigen de uma dada
imagem
%
% Autor: Fernando Otávio Fonseca
% Data: 23/01/2016
%
% Entradas: diretorio (local onde está a imagem)
%           nome (do arquivo da imagem)
```

```
%           tamimag - lin*col (tamanhos da imagem)
%           media - das imagens de treinamento
%           eigenfaces
%
% Saída:    imgproj - Vetor de características Eigen

if strcmp(diretorio,'matriz')
    img = nome;
else
    img = imread([diretorio  nome]);
    % Converte para escala de cinza, caso a imagem seja colorida
    if size(img,3)>1
        img = rgb2gray(img);
    end
end
% Vetoriza a imagem 2D em 1D
vecimg = reshape(img',tamimag,1);

% Calcula a diferença em relação ao centro
difg = double(vecimg) - media;

% Vetor de características da imagem
imgproj = eigenfaces'*difg;

% ############################################################
```

function [total_faces, media, eigenfaces, vetor_carac] = montaEigenfaces(dirfaces, dirnonfaces, ext, tamimag)

```
% Função que gera matriz de características das Eigenfaces
%
% Autor: Fernando Otávio Fonseca
% Data: 23/01/2016
%
% Entradas: dirfaces
%           dirnonfaces
%           ext - extensão dos arquivos de imagens
%
```

```
% Saídas:   vetor_carac
%           eigenfaces
%           media
%           tot_arq_fac

% Cria lista dos arquivos de treinamento de Faces e Não faces
listarq = dir(strcat(dirfaces,ext));
tot_arq_fac = size(listarq,1);

% Constroi a matriz de imagens
matpix = [];
for i = 1 : tot_arq_fac

     % Lê imagem como matriz e converte para tom de cinza se
necessário
    im = imread([dirfaces  listarq(i).name]);
    if size(im,3)>1
        im = rgb2gray(im);
    end

    % Transforma a imagem 2D em vetor de 1D
    vec = reshape(im',tamimag,1);
    % Adiciona a matriz de pixels
    matpix = [matpix vec];

end

% Cria lista dos arquivos de treinamento de Não Faces
listarq = dir(strcat(dirnonfaces,ext));
tot_arq_non = size(listarq,1);

% Constroi a matriz de imagens
for i = 1 : tot_arq_non

     % Lê imagem como matriz e converte para tom de cinza se
necessário
    im = imread([dirnonfaces  listarq(i).name]);
    if size(im,3)>1
```

```
        im = rgb2gray(im);
    end

    % Transforma a imagem 2D em vetor de 1D
     vec = reshape(im',tamimag,1);
    % Adiciona a matriz de pixels
    matpix = [matpix vec];

end

% Calcula a média das imagens
% Média da imagem: media = (1/P)*sum(Tj's)     (j = 1 : P)
media = mean(matpix,2);
totalarq = tot_arq_fac + tot_arq_non;

% Calculando o desvio de cada imagem em relação a média
A = zeros(size(media,1),totalarq);
for i = 1 : totalarq
    dife = double(matpix(:,i)) - media;
    A(:,i) = dife;
end

% L é o substituto da matriz covariância C = A * A'.
L = A'*A;
% As colunas de V são os autovetores de L, e D é uma matriz diagonal
com
% os eigenvalues correspondentes, onde LV = VD ou L = VDV^(-1).
[V D] = eig(L);

% Classificando e eliminando eigenvalues
eig_vec = [];
% Vamos identificar o limite das faces e não faces
for i = 1 : tot_arq_fac
    if( D(i,i)>1 )
        eig_vec = [eig_vec V(:,i)];
    end
end
total_faces = size(eig_vec,2);
```

```
for i = tot_arq_fac+1 : tot_arq_fac+tot_arq_non
    if( D(i,i)>1 )
        eig_vec = [eig_vec V(:,i)];
    end
end

% Usar todos os eigenvalues
total_faces = size(V,2)/2;
eig_vec = V;
% Autovetores da matriz covariância C (ou chamados "Eigenfaces") podem ser
% recuperados a partir de autovectores de L.
eigenfaces = A * eig_vec;

% Projeção do vetor de imagens no espaço - Matriz de Características
vetor_carac = [];
tamanho = size(eigenfaces,2);
for i = 1 : tamanho
    temp = eigenfaces'*A(:,i);
    vetor_carac = [vetor_carac temp];
end

% ############################################################
```

function [porcentagem_de_acertos] = avalia_modelo_SVM(SVM, Entrada, Saida_Desejada)

```
% Funcao para avaliacao de uma Rede Neural treinada utilizando o toolbox de
% Redes Neurais do Matlab e da biblioteca de SVM.
% Sao considerados conjuntos de treinamento e validacao.
% Desenvolvido por Vitor Hugo Ferreira, 19/06/2008
% Alterado por Fernando Otávio, 23/01/2016, para considerar valores
% espelhados nas saídas estimadas.

% Rede - rede neural treinada utilizando a funcao train
% Entrada - matriz de dados de dimensao N x n, onde N responde
% pelo numero de padroes e n pelo numero de entradas
```

```
% Saida_Desejada - matriz de dimensao N x 1, onde N responde pelo
numero
% de padroes e uma saida
% porcentagem_de_acertos - porcentagem de acertos para o conjunto de
dados

[numero_de_padroes,numero_de_classes] = size(Saida_Desejada);

% Avaliacao do modelo para o conjunto de treinamento

for i = 1:numero_de_classes

    [Saida_Estimada(:,i),           Erro,           lixo]           =
svmpredict(Saida_Desejada(:,i), Entrada, SVM{i});

end

% Diferenca = sum(Saida_Estimada - Saida_Desejada,2);
Dif = Saida_Estimada - Saida_Desejada;
Diferenca = Dif(:,1) - Dif(:,2);

acertos = find(Diferenca == 0);
numero_de_acertos = max(size(acertos));
porcentagem_de_acertos = numero_de_acertos/numero_de_padroes*100;
```

7.4. Detecção com máquina SVM e filtros de características

```
% Teste da Detecção de Faces usando SVM
%
% Autor: Fernando Otávio Fonseca
% Data: 04/01/2016
%

close('all');
clc;
clear;

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

% Cria lista dos arquivos
% Teste com imagens novas
%  diretorio  =  'C:\Users\Fernando\Documents\MATLAB\Detector  Faces
VJ\Imagens Teste\';
diretorio   =   'C:\Users\Fernando\Documents\MATLAB\Detector   Faces
VJ\Teste MIT\test-low\';
%  diretorio  =  'C:\Users\Fernando\Documents\MATLAB\Detector  Faces
VJ\TesteLote\';
%  diretorio  =  'C:\Users\Fernando\Documents\MATLAB\Detector  Faces
VJ\TesteNFaces\';
%  diretorio  =  'C:\Users\Fernando\Documents\MATLAB\Detector  Faces
VJ\Base CMU\test\face\';

% Extensões das imagens
ext = '*.jpg';
% ext = '*.pgm';

listarq = dir(strcat(diretorio,ext));

% Quantidade de linhas e colunas das imagens treinadas
linha = 24;
coluna = 24;
larg = coluna-1; % Largura total menos um
alt = linha-1; % Altura total menos um
```

```
pixels = linha*coluna; % Usado para reshape

% for k=50:65
for k=1:size(listarq,1)

    % im = imread([diretorio  'im1.jpg']);
    im = imread([diretorio  listarq(k).name]);

    % Converte para escala de cinza, caso a imagem seja colorida
    if size(im,3)>1
        im = rgb2gray(im);
    end

    % im = imresize(im,[33 33]);

    im2 = double(im);

    % Inicializa os controles de escala
    escala = 1; % Define a escala inicial
    min_escala = 1;
    % min_escala = 0.1;
    passoescala = 0.92;

    % Inicializa a saída
    regioes = [];


    while escala>=min_escala
        tic
        % Redimensionar a imagem
        imre = imresize(im2,escala);
        lre = size(imre,2);
        alre = size(imre,1);
        passox = floor(16*escala);
        passoy = floor(16*escala);
```

```
        faces = [];

        % Movimenta a janela usando os passos definidos
        for i=2:passox:lre-larg-1
            for j=2:passoy:alre-alt-1

                % Extrai a região do teste e prepara a entrada
                janela = imre(j:j+alt,i:i+larg);
                entradaTeste                                        =
determinaHaar(integralImagem(janela),mascara);
                eTeste                                              =
mapminmax('apply',entradaTeste,Limites_Entradas);
                entradaTeste = eTeste';

                % Avalia a janela no modelo treinado
                numero_de_saidas = 2;
                saidaTeste = ones(1,2);
                saidaTeste(:,2) = -1;
                for v = 1:numero_de_saidas
                    [Saida_Estimada(:,v),      Erro,      lixo]      =
svmpredict(saidaTeste(:,v), entradaTeste, SVM{v}, '-q');
                end
                saidaTeste = Saida_Estimada;

                if saidaTeste(1,1)>0 && saidaTeste(1,2)<0
                    faces = [faces; i,j,larg+1,alt+1];
                end
            end
        end
        % Se uma ou mais regiões foram classificadas positivamente
na escala
        % atual, guarda as coordenadas com a devida correção de
escala
        fprintf('Verificação da Escala %d concluída.\n',escala);
        toc
        if size (faces,1)> 0
            regioes = [regioes; faces/escala];
            fprintf('Encontrada %d regiões.\n',size(faces,1));
        end
```

```
    % escala = escala*1/passoescala;
    escala = escala*passoescala;
    end

    % Mostra imagem original
    figure,imshow(im),hold on;
    % Desenha os retangulos identificados
    for i=1:size(regioes,1);
        rectangle('Position',
regioes(i,:),'LineWidth',2.5,'EdgeColor','b');
    end

end


% ##########################################################

function [x,y] = avaliaRegioeSVM( x, y, img, alt, larg,
Limites_Entradas, SVM, numero_de_classes, tipo)

% Funcao para avaliacao de uma região da imagem com uma Rede Neural
% treinada utilizando o toolbox de Redes Neurais do Matlab e da
biblioteca de SVM.
% Desenvolvido por Fernando Otávio em 27/01/2016

% Entradas:  img - matriz de pixels da imagem em avaliação
%              x, y - vetores de pontos na imagem que identificam o
canto
%                   superior esquerdo das regiões a serem avaliadas
pela
%                  máquina SVM
%            alt, larg - dimensões das regiões da imagem
%                  imblin, imbcol - dimensões das imagens base de
treinamento
%            Limites_Entradas - referência para a função mapminmax
%            SVM - estrutura da rede neural treinada
%            numero de classes - das saídas
%            tipo - dos filtros das imagens de treinamento
%
```

```
% Saídas:     x, y - vetores dos pontos que classificaram positivos
após
%                   consulta na máquina de SVM

% Cria a matriz de pixels para a entrada
matrizEntrada = [];
for i=1:length(x);

    regiao = img(y:y+alt,x:x+larg);
    regiao = imresize(regiao,[tipo.linha tipo.linha]);
    if strcmp(tipo.nome,'SVM')
        retorno = reshape(regiao',tipo.tamimag,1);
    elseif strcmp(tipo.nome,'SVMHaar')
        retorno                                                    =
determinaHaar(integralImagem(regiao),tipo.mascara);
    else
        retorno                                                    =
extraiEigenfaces(tipo.eigenfaces,tipo.media,tipo.tamimag,regiao,'mat
riz');
    end
    matrizEntrada = [matrizEntrada retorno];
end

Entrada = mapminmax('apply',matrizEntrada,Limites_Entradas);
Entrada = Entrada';
Saida_Desejada = ones(length(x),numero_de_classes);
Saida_Desejada(:,2) = -1;
Saida_Estimada = ones(length(x),numero_de_classes);

for i = 1:numero_de_classes

    [Saida_Estimada(:,i),  ~,  ~]  =  svmpredict(Saida_Desejada(:,i),
Entrada, SVM{i});

end

Dif = Saida_Estimada - Saida_Desejada;
Diferenca = Dif(:,1) - Dif(:,2);
```

```
acertos = find(Diferenca == 0);
x = x(acertos);
y = y(acertos);
```