UNIVERSIDADE DE SÃO PAULO

ESCOLA POLITÉCNICA

GUSTAVO RAHMILEVITZ

MICHEL FRIEDLANDER

SOFTWARE PARA ANÁLISE DE EMOÇÕES NA FACE.

São Paulo

2010

GUSTAVO RAHMILEVITZ
MICHEL FRIEDLANDER

SOFTWARE PARA ANÁLISE DE EMOÇÕES NA FACE

Trabalho apresentada à Escola Politécnica da Universidade de São Paulo para conclusão do curso de Engenharia Mecatrônica e obtenção do título de graduação.

Área de Concentração:
Engenharia Mecatrônica

Orientador: Prof. Doutor Marcos Ribeiro Pereira Barretto

São Paulo
2010

FICHA CATALOGRÁFICA

Rahmilevitz, Gustavo
Software para análise de emoções na face / G. Rahmilevitz; M. Friedlander. -- São Paulo, 2010.
122 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1. Emoções 2. Traços de personalidade (Características) 3. Comunicação não-verbal 4. Algoritmos para imagens 5. MATLAB I. Friedlander, Michel II. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos III. t.

AGRADECIMENTOS

Ao Prof Doutor Marcos Barretto pela orientação, pelo estímulo, pelas ideias e pelo apoio constante durante todo o projeto.

Aos nossos pais por sempre nos apoiarem até hoje em todas as iniciativas que tomamos.

E a todos nossos amigos e familiares que nesses últimos tempos nos “aguentaram” em nossas agonias e trabalhos sem fim.

RESUMO

Frequentemente desejamos conseguir olhar além do simples significado das palavras; o que a pessoa realmente quer dizer, o que ela esta sentindo e quão verdadeira está sendo? Expressões faciais universais para descrever emoções já foram identificadas (EKMAN, MATSUMOTO e FRIESSEN, 1982);(EKMAN, FRIESEN e HAGER, 2002) (incluindo – raiva, satisfação, nojo, medo, tristeza, surpresa e alegria – e podem de forma objetiva e confiável ser separadas umas das outras. Este trabalho tem como objetivo criar um programa completamente automatizado que analise os “action unit’s” (AU’s) faciais, baseados nos FACS de Paul Eckman, e determine qual a emoção que a pessoa em questão está sentindo e o grau de certeza dessa análise. Já existem muitos programas de detecção de emoção facial, no entanto todos se baseiam na comparação de estados emocionais macro presentes em fotos ou filmes. Sabendo que essas emoções prototípicas ocorrem com pouca frequência, preferimos estudar a comunicação presente na mudança de poucas e discretas características faciais. O programa deve analisar os AU’s descritos por características faciais permanentes (boca, sobrancelha) e transientes (rugas, sulcos) capturadas em fotos frontais que são então comparadas ao estado neutro. Um total de 21 AU’s (5 da face superior e 16 da face inferior) podem ser reconhecidos pelo programa depois que a face foi detectada e separada do resto da imagem. Esses AU’s são agrupados por região e analisados individualmente conforme as regiões são isoladas para melhor verificação. A biblioteca OpenCV foi utilizada para desenvolver algoritmos para a detecção da face e das características relevantes a análise de emoções. O núcleo do *software* é o método proposto por Viola e Jones, que apresenta boas taxas de detecção e baixa taxa de falsos positivos. Também apresentamos um algoritmo para o contorno das imagens para análise da posição da característica. Utilizamos para a tomada de decisão da emoção os algoritmos de Redes Neurais com aprendizado por retropropagação através do programa Matlab™ da MathWorks.

Palavras-chave: Detecção de face. Viola Jones. OpenCV. FACS. Action units.

ABSTRACT

It is frequent nowadays to want to look beyond the meaning of words; what does one truly mean, what is he feeling and how true is he being? Universal facial expressions to describe emotions have been identified (EKMAN, MATSUMOTO e FRIESSEN, 1982);(EKMAN, FRIESEN e HAGER, 2002) which include – anger, contempt, disgust, fear, sadness, surprise and happiness – and can be objectively and reliably distinguished one from another. The objective of this paper is to provide a fully automated program that analyses facial action units (AU's), based on Paul Eckman's Facial Acting Coding System (FACS), and determines the emotion felt by the person and the degree of certainty. Many programs of facial emotions already exist, however they are based on comparing the macro emotional state present on the photo or live feed. Knowing that these prototypic emotions occur rather infrequently we preferred to study the communication present in the change of one or a few discrete facial *features*. The program must analyze AU's described by permanent (mouth, eyebrows) and transient (wrinkles, furrows, bags) facial *features* captured in frontal-view pictures which are then compared to the neutral state. A total of 18 AU's (x upper face AU's and y lower face AU's) can be recognized by the program after the face has been located and separated from the rest of the picture. These units are grouped by region and analyzed distinctly as each section is isolated for better examination. We used the OpenCV library to develop algorithms to detect the face and other relevant *features* for the emotional analysis. The core of the software is the method proposed by Viola Jones that presents good detection indexes and low false positives. We also produced an algorithm which contours the image and shows its edges to facilitate the analysis of the exact location of the feature. We used neural networks algorithms with learning through the backpropagation algorithm to the emotions decision, and the software Matlab to the implementation.

Keywords: Face detection. Viola Jones. OpenCV. FACS. Action units.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Expressões faciais são uma das mais poderosas, naturais e imediatas formas para seres humanos comunicarem suas emoções e intenções. O rosto pode expressar emoções antes que as pessoas coloquem em palavras, ou até mesmo antes de perceber os seus próprios sentimentos.

Na última década, muitos progressos aconteceram para a criação de sistemas de computador para entender e usar essa forma natural de comunicação humana. A maioria dos sistemas tenta reconhecer um pequeno conjunto de expressões emocionais como a alegria, surpresa, raiva, tristeza, medo e aversão. Essa prática pode resultar do trabalho do Darwin, e mais recentemente do Ekman e do Friesen(1982), que propuseram que as emoções básicas possuem expressões faciais correspondentes. Porém, diariamente essas expressões são relativamente infrequentes. Em vez disso, emoções normalmente acontecem por mudanças sutis, em uma ou poucas características faciais discretas; como em um momento de raiva o aperto dos lábios, ou obliquamente baixar o canto dos lábios na hora da tristeza, e também levantar as sobrancelhas no momento de uma saudação. Para capturar essas sutilezas das emoções humanas e metalinguísticas da comunicação, o reconhecimento automático dessas pequenas mudanças é necessário.

Ekman e Friesen (2002) então acharam ser necessária a criação de padrões de codificação. O *Facial Action Coding System* (FACS) é o mais objetivo e compreensivo sistema de codificação nas ciências comportamentais. Esse descreve as expressões faciais em unidades de ação (*Action Units – AU's*). Das 44 AU's que eles definiram, 30 são anatomicamente relacionadas com a contração de músculos faciais específicos (12 AU's para a parte superior da face, e 18 AU's para a parte inferior), que podem corresponder a um músculo específico, ou a um grupo muscular, e são essencialmente fonemas faciais (que podem ser assimilados a formar expressões).

Os AU's podem ocorrer individualmente ou em combinação. Quando ocorrem em combinação elas podem ser aditivas, na qual a associação não muda a aparência do elemento; ou podem ser não aditivas, na qual essa aparência sim irá mudar. Mesmo por existindo um número baixo de AU's, mais de 7.000 combinações

podem ocorrer. FACS fornecem o poder necessário para descrever os detalhes da expressão facial.

Uma grande porcentagem da comunicação humana é não verbal, e entre esses sinais não verbais, grande parte está na forma de ações faciais. Um sistema que pudesse analisar as expressões faciais em tempo real sem intervenção humana, poderia ser aplicado em diferentes campos como: psicologia, computação afetiva e gráfica. Um sistema assim seria de extrema importância para uma máquina que é inteligente emocionalmente e socialmente, e que é esperada para lidar diretamente com o público.

Antigamente existiam diferentes técnicas de classificação para reconhecer unidades de ação e suas combinações. A maioria das tentativas de fazer uma análise automatizada, possuíam um pequeno conjunto de protótipos de expressões emocionais. Alguns exemplos desses trabalhos são:

- M. Suwa, N Sugie e K. Fugimora (1978), que lançaram um programa na tentativa de analizar as expressões a partir do acompanhamento de 20 pontos específicos em uma sequencia de imagens;

- K. Mase (1991) que manualmente selecionou regiões faciais que correspondiam a músculos da face e calculou o movimento dentro dessas regiões usando fluxo óptico;

- Y. Yacoob e S. L. Davis (1996) que também usaram o fluxo óptico, mas ao invés de usarem os grupos de músculos, utilizaram a região de superfície de características faciais (sobrancelhas, olhos, nariz e boca).

Neste trabalho, optamos por desenvolver um programa semi automatizado que analisa uma foto neutra e uma foto emotiva e informa qual a emoção mais provável descrita pelas características faciais. Usaremos os FACS(EKMAN, MATSUMOTO e FRIESSEN, 1982) como método de interpretação dos pontos específicos e pré-determinados. De acordo com Sayette (2001) podemos confiar na ocorrência e correta identificação dos AU’s para descrever as mudanças emocionais sentidas pelo individuo.

Até o presente momento, FACS são usados na analise de vídeos em câmera lenta e identificados por pessoas treinadas que os assistem repetidamente. Para descartar a interpretação humana das condições de interpretação, estudaremos métodos existentes para identificação de sequências gravadas e equacionaremos os possíveis movimentos para avaliar as diferenças em relação à posição neutra.

Primeiramente estudamos cuidadosamente os AU's do FACS (EKMAN, FRIESEN e HAGER, 2002) e avaliamos quais seriam os possíveis de ser identificados e analisados em imagens frontais, escolhemos 21 que satisfazem esses quesitos e independem do tempo de duração (esse ponto é importante por estarmos analisando fotografias e não vídeos). Desses 21 AU's, 5 são localizados na parte superior da face e 16 na parte inferior. Para localizar as regiões desejadas da foto, utilizaremos a biblioteca *OpenCV* e a metodologia de Viola Jones para treinar *HaarCascades* específicos para cada AU. Primeiramente localizamos a face na foto, e depois partes características de cada AU como: sobrancelha; boca; olhos e nariz.

Tendo a localização na foto das áreas desejadas, separamos apenas os contornos para limpar a imagem e conseguir matematicamente interpretar a presença/ausência do AU em questão. Esses AU's e sua determinada intensidade são combinados para descrever uma ou mais emoções presentes na fotografia.

Para a parte final da decisão de qual emoção que está sendo tratada, utilizamos redes neurais treinadas com o algoritmo de retropropagação. Utilizamos a caixa de ferramentas de redes neurais do Matlab para criar e treinar a rede e fazer os testes com outras imagens.

2 FACS

O FACS (EKMAN, FRIESEN e HAGER, 2002), sistema de codificação das ações faciais na tradução literal é utilizado para separar e analisar individualmente cada pequena alteração descrita na face. Os AU's são as peças usadas no quebra cabeça que ilustra a face humana, os movimentos descritos nos AU's são causados pela atuação de músculos, os principais podem ser encontrados na Figura 1 abaixo:

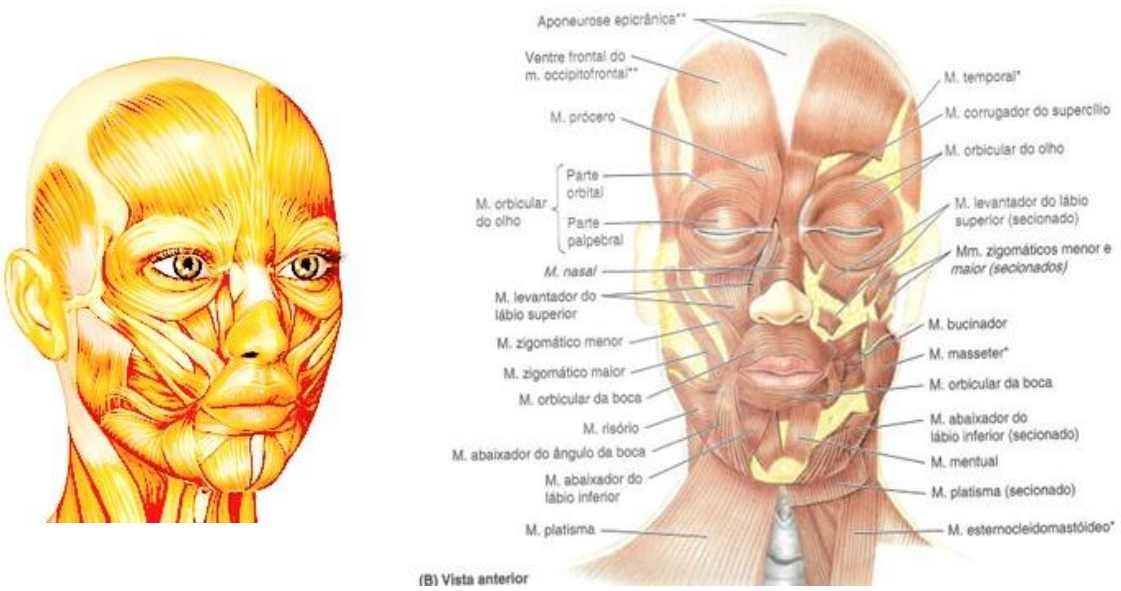

Figura 1 - Músculos da Face (Fonte: A - (ANÔNIMO) B - (MOORE e DALLEY, 2007)

Cada AU é cuidadosamente estudado, entendendo por completo qual o músculo causador do movimento, quais as principais aparências demonstradas na face e as variações possíveis entre diferentes pessoas.

Toda essa atividade facial é manifestada em discretos movimentos interpretados pela presença ou não dos AU's equivalentes, no entanto é preciso integrar toda essa informação novamente para poder avaliar o evento facial presente e então interpretá-lo com emoção. A tabela abaixo apresenta as seis emoções estudadas neste trabalho e quais os AU's que a caracterizam:

Tabela 1 – AU's representantes de emoções

Emoção	AU's Presentes	
	Típico	Variações aceitáveis
Surpresa	1+2+5B+26ou27	1+2+5B 1+2+26 1+2+27 5B+26 5B+27
Medo	1+2+4+5+20+25,26 ou 27	1+2+4+5 1+2+5 (25,26ou27 possíveis) 5+20 (25,26ou27 possíveis)
Alegria	6+12	
Tristeza	1+4+11+15B+25ou26 1+4+15+25ou26 6+15+25ou26	1+4+11 1+4+15B 1+4+15B+17 11+15B 11+17
Nojo	9 9+16+25ou26 9+17 10 10+16+25ou26 10+17	
Raiva	4+5+7+1+22+23+25ou26 4+5+7+10+23+25ou26 4+5+7+23+25ou26 4+5+7+17+23 4+5+7+17+24 4+5+7+23 4+5+7+24	Qualquer variação da combinação apresentada ao lado sem um dos seguintes AU's: 4, 5, 7 ou 10.

Fonte: (EKMAN, FRIESSEN e HAGER, 2002)

A seguir apresentamos uma descrição detalhada de cada uma das Unidades de Ação: quais suas características visíveis, como reconhecê-las e os músculos envolvidos em cada movimento.

2.1 ACTION UNIT 1

O maior músculo da face humana, o *Frontalis*, liga verticalmente o couro cabeludo à sobrancelha fixando todos os movimentos executados pela testa. A parte central (centralis) deste músculo é responsável realizar o deslocamento característico do AU1, que ajeita o canto interno da sobrancelha para cima formando uma figura obliqua.

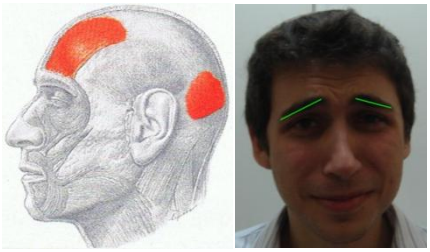

Figura 2 - A - Músculo Frontal (Fonte:(PUTZ e PABST, 2006)); B - AU1

Aparência característica do AU1:

- Seção interna da uma ou ambas as sobrancelhas e puxada para cima.

- Na maioria dos casos produz uma forma obliqua às sobrancelhas.

- Pode causar rugosidade na testa, no caso deste AU as rugas ficam limitadas apenas à área central da testa exibindo uma forma mais curva do que horizontal (centro mais elevado). Crianças e adolescentes podem não apresentar esse atributo e no caso de rugas permanentes, as mesmas podem apresentar profundidade acentuada.

- O ponto extremo externo das sobrancelhas pode também se mover um pouco, mas diferentemente do que ocorre com o AU2, nesse caso o movimento é em direção à linha central da face e não para cima.

2.2 ACTION UNIT 2

O mesmo músculo agente do AU descrito acima causa o movimento da parcela externa da sobrancelha, isso ocorre devido ao fato de que parte central e lateral tem ações separadas e independentes. No AU2 percebemos a contração das laterais da testa causando o levantamento da parte externa da sobrancelha formando uma figura curvada.

Figura 3 - AU2

Aparência característica do AU2:

- Seção externa de uma ou ambas as sobrancelhas é levantada.

- Produz na maioria dos casos uma forma arqueada aguda nas sobrancelhas.

- Estica para cima a parte lateral das pálpebras.

- Em muitos casos apresenta rugosidade distinta nas áreas laterais da testa em cima das sobrancelhas. Algumas pessoas podem também apresentar rugas na área central, porém estas são pouco profundas quando comparadas com as laterais.

- Novamente é necessário muito cuidado para não confundir o AU1 com o AU2. A parte interna da sobrancelha pode também se movimentar, mas nesse caso é um movimento puxado para fora e não para cima.

2.3 ACTION UNIT 4

Este AU é consequência da composição de ação de 3 músculos distintos da face superior: o primeiro deles, o *Depressor supercilii*, tem posicionamento obliquo à testa, emergindo perto da base do nariz e subindo para o exterior da testa logo acima da sobrancelha (este é o mais potente dos três e puxa as sobrancelhas em um movimento de união para baixo); o segundo músculo, o *Depressor glabellae*, também tem início próximo à base do nariz, mas sobe verticalmente com uma abertura radial; o último dos três músculos, o *Corrugator supercilii*, liga a parte interna da sobrancelha/testa ao canto do olho. Os músculos normalmente agem de forma integrada para proporcionar os elementos visíveis do AU4, mas dependendo da intensidade e particularidade do movimento, podemos perceber um pouco mais de participação de cada um deles.

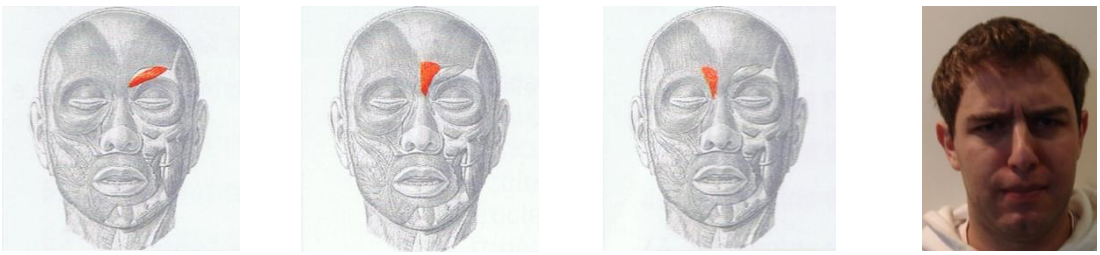

Figura 4 - A - Depressor supercilii; B – Depressor glabellaer; C – Corrugator supercilii (Fonte: (PUTZ e PABST, 2006)); D - AU4

Aparência característica do AU4:

- Abaixa uma ou ambas as sobrancelhas. Pode ser notado na parte interna, externa ou na sobrancelha como um todo.

- Empurra a pálpebra para baixo e pode estreitar a abertura do olho.

- Aproxima as sobrancelhas.

- Produz rugas horizontais entre as sobrancelhas. Em alguns poucos casos, essas rugas podem estar presentes em um ângulo de 45° ou uma combinação de horizontal e angular.

- Pode causar pequenas rugas na base do nariz.

- Pode produzir uma protuberância muscular descendo do centro da testa em direção ao canto interno da sobrancelha.

2.4 ACTION UNIT 6

Existe um músculo que circula a órbita ocular, *orbicularis oculi (pars orbitalis)*, esse músculo percorre toda a área interna e próxima às sobrancelhas e abaixo do sulco. O AU6 puxa toda a pele nesta zona de influencia em direção ao olho.

Figura 5 - A – Orbicularis oculi (Fonte:(PUTZ e PABST, 2006)); B - AU6

Aparência característica do AU6:

- Puxa a pele da têmpora e bochechas em direção ao olho e contrai a banda externa do músculo.

- Levanta o triângulo infra-orbital.

- Empurra a pele ao redor dos olhos em direção ao soquete ocular podendo estreitar a abertura dos olhos.

- Pode causar pés de galinha, estendo-se radialmente à partir do canto externo do olho.

- Aprofundo o sulco da pálpebra inferior.

- Pode abaixar a porção lateral das sobrancelhas.

2.5 ACTION UNIT 7

O músculo responsável pelo AU6 é também o responsável pela AU7, porém neste caso é a parte mais interna da coroa circular, *pars palpebralis*, (que percorre toda a área interna e próxima às pálpebras) que executa a ação. Quando contraído, ele puxa as pálpebras e toda a pele próxima para dentro em direção ao canto interno do olho.

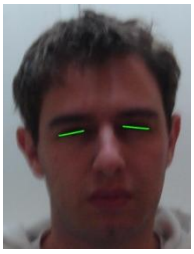

Figura 6 - AU7

Aparência característica do AU7:

- Estica as pálpebras.

- Estreita a abertura dos olhos.

- Normalmente é mais perceptível na pálpebra inferior do que na superior.

- Levanta a pálpebra inferior fazendo com que mais do olho seja coberto do que o normal.

- As pálpebras perdem parte de sua forma curva apresento desenho mais reto.

- Um pequeno inchaço é perceptível abaixo da pálpebra inferior.

- É possível que o AU7 seja percebido somente em um lado da face e AU6 na outra, nesse caso deve-se anotar a anomalia como AU6 bilateral.

2.6 ACTION UNIT 10

O AU10 é causado pelo músculo *Levator labii superioris.* Ele emerge no centro do triângulo infraorbital indo até a extremidade inferior do nariz. Neste AU a pele acima do lábio é erguida em direção à bochecha puxando o lábio superior.

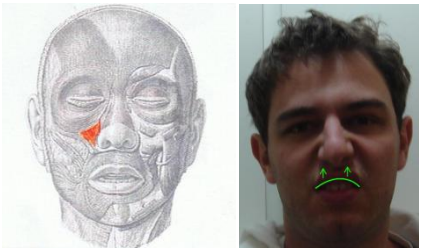

Figura 7 - A - Levator labii superioris (Fonte:(PUTZ e PABST, 2006)); B - AU10

Aparência característica do AU10:

- Ergue o lábio superior. A parte central do lábio tem um movimento mais acentuado do que os cantos.

- Causa o desenho de uma curva quadrática negativa nesta parte do lábio.

- Empurra o triângulo infraorbital para cima, podendo causar rugas.

- Pode inflar as bolsas laterais ao nariz.

- Alarga e eleva as asas das narinas.

- Em casos de movimento acentuado, os lábios podem separar.

2.7 ACTION UNIT 11

O *Zygomaticus minor*, músculo responsável pelo movimento deste AU emerge abaixo do osso malar e termina acima do lábio superior. A indefinição exata da separação dos músculos nessa área do rosto pode acarretar na presença de pequenas ações de outros músculos também.

Figura 8 - A - Zygomaticus minor (Fonte:(PUTZ e PABST, 2006)); B - AU11

Aparência característica do AU11:

- Puxa o lábio superior na diagonal em direção à orelha, o ponto puxado fica a um quarto da distância entre os cantos do lábio.

- Aprofundo o sulco nasal.

- Puxa a pele do sulco nasal para cima.

- Infla levemente a parte central do triângulo infraorbital.

2.8 ACTION UNIT 12

Este AU utiliza o mesmo músculo do AU anterior, porém com maior intensidade e por completo. Os cantos do lábio são puxados obliquamente em direção ao osso malar.

Figura 9 - A - Zygomaticus major (Fonte:(PUTZ e PABST, 2006)); B – AU12

Aparência característica do AU12:

- Dá à boca um formato de “U”.

- Aprofunda o sulco nasal, empurrando o lateralmente e para cima. A pele próxima a essa área tem o mesmo movimento podendo apresentar rugas.

- Triângulo infraorbital é erguido podendo apresentar demarcação dos sulcos.

- Um movimento forte pode apresentar: bolsas na pele inferior ao olho; pés de galinha; leve fechamento dos olhos; alargamento das narinas.

- Vale prestar atenção que as diferenças entre um AU12 forte e um AU6 são tênues.

2.9 ACTION UNIT 15

Causado pelo *Depressor anguli oris* que emerge na lateral do queixo e se estende até o canto do lábio. As laterais do lábio são puxadas para baixo.

Figura 10 - A - Depressor anguli oris (Fonte:(PUTZ e PABST, 2006)); B - AU15

Aparência característica do AU15:

- Muda a forma do lábio deixando-o angulando para baixo nas extremidades, e geralmente o lábio inferior está esticado horizontalmente.

- Produz rugas e aparência de bolsas logo abaixo do lábio inferior.

- Pode causar mudança na área do queixo.

2.10 ACTION UNIT 17

Causado pelo *Mentalis* que interliga o lábio inferior à área abaixo do queixo. Neste movimento a pele do queixo é esticada para cima empurrando o lábio.

Figura 11 - A – Mentalis (Fonte:(PUTZ e PABST, 2006)); B - AU17

Aparência característica do AU17:

- Empurra o queixo para cima.

- Lábio inferior pode ser erguido.

- Pode causar rugas conforme a pele do queixo é esticada e em alguns casos apresenta uma pequena depressão abaixo do lábio.

- Dá à boca a forma de um U invertido.

- Em movimentos acentuados a área logo abaixo do lábio pode sobressair.

2.11 ACTION UNIT 18

Os músculos responsáveis por este AU estão localizados acima do lábio superior (*Incisivii labii superioris*) e abaixo do lábio inferior (*Incisivii labii inferioris*). Quando contraídos, eles puxam os lábios em direção radial causando sobre-saliência dos mesmos.

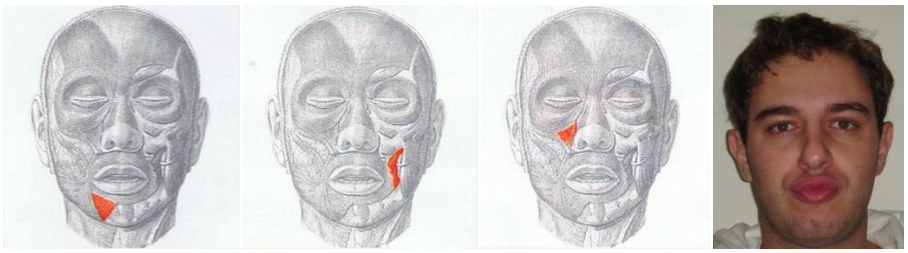

Figura 12 – A:Buccinator; B:Incisivii labii superioris; C:Incisivii labii inferioris (Fonte:(PUTZ e PABST, 2006)); D:AU18

Aparência característica do AU18:

- Empurra os lábios para fora e os puxa radialmente.

- Deixa a abertura oral menor e mais redonda.

- Produz rugas na pele acima e abaixo dos lábios.

- Pode ocasionalmente afetar somente um dos lábios, T18 (superior) B18 (inferior).

2.12 ACTION UNIT 19

Neste Action Unit é preciso ver a língua à frente da linha dos dentes. O critério de sobre-saliência da língua é necessário para diferenciar esse AU do perfil normal de comer, falar, etc.. Não basta ver a língua quando a boca estiver aberta.

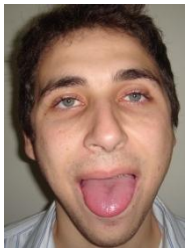

Figura 13 - AU19

Aparência característica do AU19:

- Pelo menos a ponta da língua deve estar visível e a frente dos dentes, na linha da parte vermelha dos lábios.

- Não existe distinção entre o quanto da língua está aparente ou sua posição quando fora da boca.

- Este AU normalmente é acompanhado pelos AU’s 25, 26 ou 27.

2.13 ACTION UNIT 20

Este AU tem ação principal do músculo *Risorius*, o canto do lábio é ligado à extremidade da mandíbula logo abaixo da orelha. Os lábios são esticados horizontalmente em direção às orelhas.

Figura 14 - A – Risorius (Fonte:(PUTZ e PABST, 2006)); B - AU20

Aparência característica do AU20:

- Apresenta movimento principal no eixo horizontal, porém as extremidades laterais do lábio podem ser elevadas ou rebaixadas.

- Alonga o perfil da boca.

- Afina os lábios.

- Parte da bochecha próxima ao lábio é esticada.

- Pode apresentar rugas nas áreas próximas ao canto do lábio.

- Estica a pele do queixo.

- Pode alargar as narinas.

2.14 ACTION UNIT 23

O *Orbicularis oris*, presente na órbita oral dentro dos lábios é responsável por apertar e afiná-los.

Figura 15 - A - Orbicularis oris (Fonte:(PUTZ e PABST, 2006)); B - AU23

Aparência característica do AU23:

- Aperta os lábios, deixando a parte avermelhada mais fina.

- Pode causar um enrolamento da parte vermelha do lábio, fazendo-o praticamente desaparecer.

- Pode aparentar sobre saliência dos lábios.

- Pode apresentar rugas acima do lábio e bolsas musculares abaixo. Rugas no queixo também podem ocorrer.

- Pode ocasionalmente afetar somente um dos lábios, T23 (superior) B23 (inferior).

2.15 ACTION UNITS 25, 26 & 27

Estes AU's são analisando em conjunto pois tratam do movimento da abertura da boca, separação dos lábios e dos dentes. O AU25 relaciona a distancio entre os lábios; AU26 especifica o movimento vertical da mandíbula; e o AU27 mede o movimento horizontal das laterais dos lábio quando este é aberto.

Aparência característica do AU25:

- Separação dos lábios.

- Exposição dos dentes e gengivas.

- Cavidade oral pode ser exposta, esta característica é dependente dos AU's 26 e 27

Aparência característica do AU26:

- Mandíbula é abaixada, inferindo uma possível separação dos dentes.

- Caso os lábios estejam separados é possível ver uma distância entre os dentes superiores e inferiores.

- Passa a impressão de que a mandíbula caiu, sem presença de força sendo executada (simples relaxamento do músculo)

Aparências características do AU27:

- A mandíbula é forçada para baixo.

- Abertura da boca ocorre até rebaixamento máximo da mandíbula, mudando a forma oval da boca de eixo longo horizontal para vertical.

- Aparente esticamento dos lábios.

- Achatamento das bochechas.

- É possível que ocorra sem separação dos lábios.

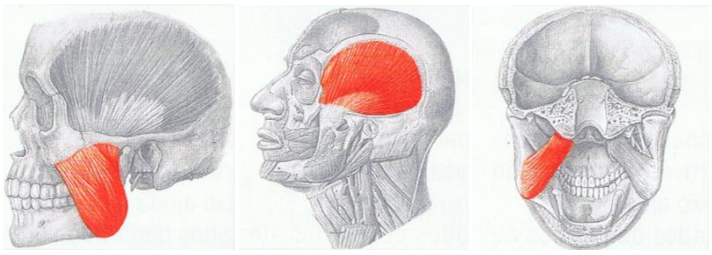

Figura 16 - A - Temporalis; B - Masseter; C – Pterygoids centralis (Fonte:(PUTZ e PABST, 2006))

Figura 17 - A - AU25 sem aparecimento da cavidade oral; B - AU25 com aparecimento da cavidade oral; C - AU26; D - AU27

2.16 ACTION UNIT 28

Envolve a ação do músculo *Orbicularis oris*, orbital à boca. Neste AU o(s) lábio(s) são puxados para dentro da boca.

Aparência característica do AU28:

- Partes vermelhas e adjacentes do(s) lábio(s) são sugados para dentro da boca cobrindo os dentes.

- Estica a pela acima e abaixo dos lábios conforme estes são puxados para dentro.

- Achata pele do queixo sem movê-lo para cima.

- Pode causar rugas nas laterais dos lábios.

- Pode ocasionalmente afetar somente um dos lábios, T28 (superior) B28 (inferior).

2.17 ACTION UNIT 29

O *Pterygoids lateralis* causa a protusão maxilar descrita neste AU.

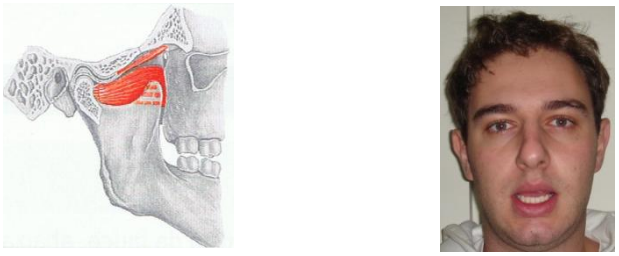

Figura 18 - A - Pterygoids lateralis (Fonte:(PUTZ e PABST, 2006)); B - AU29

Aparência característica do AU29:

- Maxilar inferior é empurrado para frente.

- Queixo aparente estar cravado para fora.

- Dentes inferiores ficam a frente dos superiores.

2.18 ACTION UNIT 30

Novamente o músculo *Pterygoids lateralis* move a mandíbula, mas dessa vez o movimento é lateral.

Figura 19 - AU30

Aparência característica do AU30:

- Queixo e lábio inferior são deslocados da linha média, em um movimento lateral.

- Quando a boa esta aberta (AU25, 26 ou 27) os dentes inferiores ficam fora de eixo em relação aos superiores. Quanto estão deslocados para a esquerda = L30, quando para a direita = R30.

2.19 ACTION UNIT 32

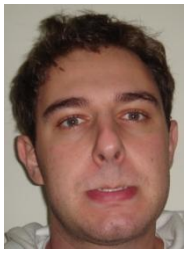

Figura 20 - AU32

Aparência característica do AU32:

- E possível ver uma linha de dentes (inferior ou superior) mordendo o lábio.

- Pode ocorrer com apenas uma pequena parte do lábio, e não ele inteiro.

3 MÉTODOS DE DETECÇÃO DE FACES

Os métodos de detecção faces são baseados em diversos tipos de informação, como intensidade de imagem em cinza, as arestas, diferença de cores, geometria da face, entre outros. Aqui explicaremos superficialmente alguns dos métodos desenvolvidos.

3.1 INTENSIDADE DE IMAGEM CINZA

3.1.1 Viola e Jones(2001)

O principal método de análise de imagem em cinza é o proposto por Viola & Jones (VIOLA e JONES, 2001) que é capaz de processar imagens de forma rápida e é o método mais utilizado e citado na literatura especializada. Ele é baseado em três inovações principais que serão explicadas na Seção4.1m que são:

- Imagem Integral – permite que todos os dados sejam extraídos em apenas uma varredura na imagem
- Classificadores utilizando AdaBoost – seleciona os *features* críticos da imagem.
- Combinação de classificadores em cascata – otimiza a utilização dos classificadores onde a probabilidade de se encontrar uma face é maior.

Os autores testaram o método e obtiveram em média 80% de taxa de detecção e 5% de taxa de erro.

3.1.2 Rowley(1996)

O método examina pequenas janelas da imagem e define se há uma face nesta janela ou não utilizando redes neurais. Os autores testaram a abordagem em um conjunto de 130 imagens e obtiveram uma média de 80% de detecção.

3.1.3 Sung e Poggio (1998)

Sung e Poggio apresentaram um algoritmo de detecção de faces em fundos complexos, através de um aprendizado baseado em modelos. Em cada posição da imagem, um vetor de características é calculado entre a imagem e o modelo. Baseado nestes vetores o classificador indica se há uma face ou não.

Os autores testaram o método em um conjunto de 324 imagens e obteve uma média 87% de detecção e 2,5% de falsos positivos.

3.2 ARESTAS

3.2.1 Wang e Tan(2000)

Wang e Tan desenvolveram um método baseado no formato da face, a imagem é realçada por seu histograma e em seguida é realizada a detecção por arestas utilizando um filtro. As arestas são então ligadas e o contorno da face é determinado utilizando a direção da ligação das arestas.

O algoritmo foi testado pelos autores com 50 imagens de fundo simples e 40 imagens de fundo complexo. No primeiro foram detectadas 100% das faces mas 16% em posição errada devido a inclinação do rosto e não houve falsos-positivos. No segundo conjunto foram detectadas 87,5% com 12,5% em posição erradas e 12,5% de falsos-positivos.

3.3 CORES

3.3.1 Cai e Goshtasby(1999)

Este método detecta faces através de transformações de cada cor em um nível de cinza e utilizando uma função de probabilidade para encontrar faces humanas na imagem.

Os autores testaram o método e obtiveram uma taxa de detecção de 87% e 8,7% de falsos-positivos

3.3.2 Hayuan & Yashida(1995)

Este método se baseia na teoria fuzzy, utilizando dois modelos. Um para descrever a cor do cabelo e outro para a cor da pele. Baseado em uma função de probabilidade para o cabelo e para a pele, tenta encontrar as faces nas imagens.

Os autores testaram o método com 233 faces e obtiveram 97% de taxa de detecção

3.4 GEOMETRIA

3.4.1 Jeng et al (1998)

Neste sistema localiza-se as possíveis coordenadas dos olhos e então busca-se um nariz, uma boca e sobrancelhas. Os autores testaram o método com 114 imagens e obtiveram 86% de taxa de detecção.

4 TRABALHO DESENVOLVIDO

A primeira parte do desenvolvimento de software para a analise da expressão facial e identificação da emoção sentida é a detecção da face e das características (boca, olhos, nariz, etc.). Existem diversos fatores que dificultam a localização da face, como barba, acessórios, cabelos sobre a face. Além de problemas como iluminação, tamanho e posição. Para o nosso trabalho tratamos de imagens simples com apenas uma face e bem iluminada para a detecção.

4.1 MÉTODO BASE

O método escolhido para desenvolvermos este trabalho foi a técnica desenvolvida por Paul Viola e Michael Jones(2001) devido a sua facilidade de uso, rapidez , possuir diversos sistemas que utilizam este método, além de ser muito citado na literatura especializada.

O método permite uma implementação rápida e robusta de objetos, o trabalho, segundo os autores, foi motivado pela possibilidade de detecção de faces. O método desenvolvido possui taxas de detecção de faces e de falso-positivos equivalentes aos melhores métodos publicados à época ((CAI e GOSHTASBY, 1999); (HJELMAS e LOW, 2001);(KAH e TOMASO, 1998)).

Existem três principais contribuições para a obtenção dos resultados descritos.

A primeira é a criação de uma nova representação de imagem chamada *Imagem Integral.* Ela permite uma rápida avaliação das características (*features*) para a detecção da face. Uma imagem integral pode ser obtida com uma única varredura da imagem original. A ideia inspirada no trabalho de Papageorgiou (1998), mas trabalhando sem a informação de intensidade da imagem pode ser computada

utilizando poucas operações por pixel, e uma vez feitas, qualquer feature do tipo Haar pode ser calculada em qualquer escala ou localização.

A segunda contribuição é o método de construção de um classificador selecionando um pequeno número de *features* importantes selecionadas a partir do algoritmo de treino AdaBoost. Isto se faz necessário, pois o número de *features* do tipo Haar é muito grande. Para garantir uma classificação rápida, o processo de aprendizagem deve excluir a grande maioria das *features* disponíveis e focar o processo em um pequeno conjunto de *features* críticas.

A terceira principal contribuição do método proposto por Viola e Jones(2001) é uma técnica para combinar sucessivamente classificadores complexos em uma estrutura de cascata aumentando a velocidade de detecção. Assim as avaliações complexas (mais demoradas) são executadas apenas em regiões que há uma probabilidade maior de se encontrar uma face. A medida de avaliação desta técnica é a taxa de falsos negativos, pois uma sub-imagem rejeitada nunca volta ao processo de classificação.

4.1.1 Imagem Integral

A imagem integral, base de todo o método proposto constitui em: dado um ponto (x,y), a imagem integral dele contém a somatória de todos os pixels acima e a esquerda do ponto x,y.

$$ii(x,y) = \sum_{x'=1}^{x} \sum_{y'=1}^{y} i(x',y')$$

Onde ii(x,y) é a Imagem integral do ponto (x,y) e i(x,y) é a imagem original. Ver figura abaixo.

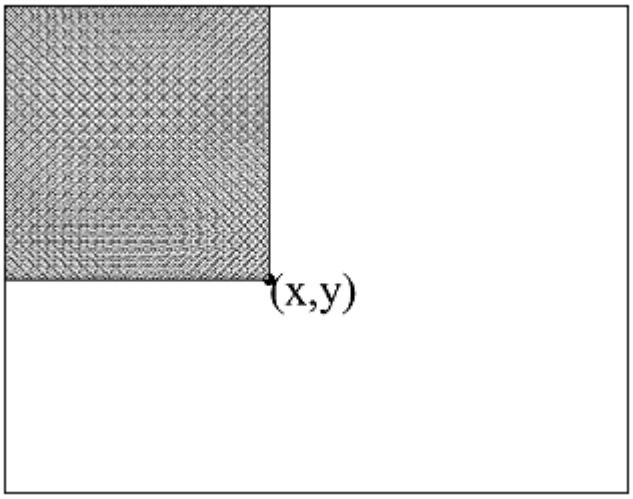

Figura 21 – valor da imagem integral no ponto (x,y) (Fonte: (VIOLA e JONES, 2001))

Utilizando as seguintes recorrências a imagem integral pode ser calculada com uma varredura sobre a imagem inteira.

$$s(x,y) = s(x,y-1) + i(x,y)$$

$$ii(x,y) = ii(x-1,y) + s(x,y)$$

Onde s(x,y) é a soma cumulativa da linha e s(x,-1) = ii(-1,y) = 0

Utilizando a imagem integral qualquer retângulo pode ser computado com o acesso a apenas quatro pontos da imagem como pode ser visto na figura abaixo, onde a soma dos pixels do retângulo D pode ser calculada como:

$$ii(1) - ii(2) - ii(3) + ii(4)$$

Pois ii(1) é a soma dos pixels da área A, ii(2) da área A+B, ii(3) da área A+C e ii(4) da área A+B+C+D.

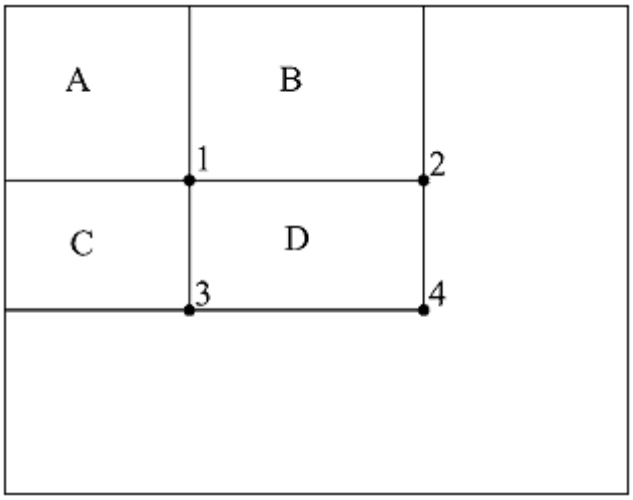

Figura 22 – Exemplo para cálculo da imagem integral no retângulo D (Fonte: (VIOLA e JONES, 2001))

4.1.2 Features

O procedimento de detecção classifica as imagens baseado nos valores dos *features* como citado acima. Existem muitos motivos para utilizá-los ao invés dos pixels diretamente. A principal razão é que os *features* podem servir para codificar especificamente um dado domínio, o que seria mais difícil utilizando a análise por pixels. E o segundo motivo mais importante é a rapidez com que se pode trabalhar com os *features*.

O valor de um feature de dois retângulos (partes A e B da Figura 23) é a diferença entre a soma dos pixels das duas regiões retangulares. As áreas tem o mesmo tamanho e formato e são adjacentes horizontal ou verticalmente. Para o *features* baseado em três retângulos (parte C da Figure 23) computa a soma dos dois retângulos exteriores e a diferença do retângulo central. E para o feature de quatro retângulos é computado a diferença da soma dos retângulos diagonais.

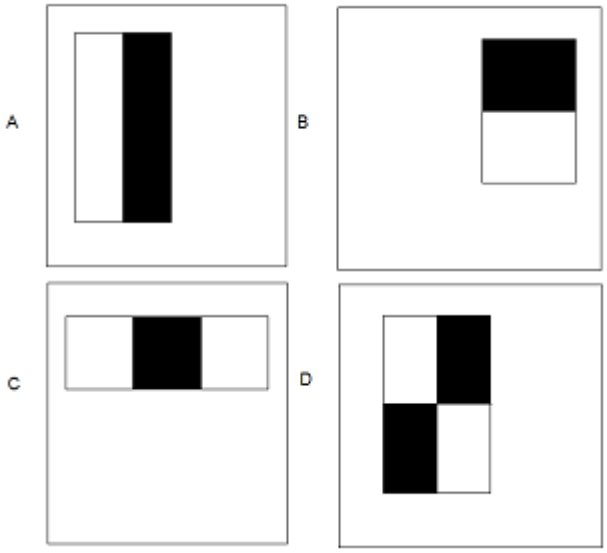

Figura 23 – *Features* retangulares para utilizados para detecção (Fonte: (VIOLA e JONES, 2001))

Os *features* apresentados são relativamente primitivos se comparados com filtros dirigíveis estudados por Freeman (1991), mas são mais eficientes para encontrar extremidades em figuras rígidas.

4.1.3 Aprendizado de Máquina

Dado um conjunto de *features* e um conjunto de imagens de faces e não faces para treinamento, qualquer procedimento de Aprendizado de Máquina pode ser utilizado para criar um classificador. No sistema proposto por Viola e Jones(2001), uma variante do AdaBoost é utilizada tanto para selecionar um pequeno conjunto de *features* e treinar o classificador enquanto que no AdaBoost original o algoritmo de aprendizado é utilizado para impulsionar a performance de classificação de um “classificador fraco”, combinando diversos classificadores fracos e criar um “classificador forte”.

O grande desafio do algoritmo é encontrar os *features* críticos para criar um classificador eficiente, pois há cerca de 180.000 *features* retangulares em imagens de 24x24 pixels em cada sub-imagem. Mesmo que cada característica possa ser processada rapidamente, computar integralmente todos os *features* é proibitivo.

Para atingir o objetivo de eficiência e encontrar os *features* críticos, o classificador fraco é desenvolvido para selecionar um único feature retangular que melhor separa entre exemplos positivos e negativos (menor erro). Para cada feature o classificador fraco determina o limite ótimo da função de classificação.

Portanto um classificador fraco $h_j(x)$ consiste em uma sub-imagem x, um feature f_j, um limite θ_j e uma paridade p_j que indica a direção da desigualdade.

$$h_j(x) = \begin{cases} 1 \; se \; p_j f_j < p_j \theta_j \\ 0 \; caso \; contrário \end{cases}$$

Na Figura 24 descrevemos em resumo o algoritmo de AdaBoost para seleção dos features de maior peso. Nele, um conjunto T de hipóteses (cada hipótese t_i criada com um único *feature*) avalia o conjunto de imagens. A saída do algoritmo é uma hipótese final criada a partir da combinação ponderada de T, onde o peso de t_i é inversamente proporcional ao seu erro de treinamento.

- Sendo T o conjunto de hipóteses a serem avaliadas onde cada t é construída utilizando um único *feature.*
- Dadas imagens de exemplo (x_1, y_1),...(x_n,y_n) onde $y = \begin{cases} 1\ para\ exemplos\ positivos \\ 0\ para\ exemplos\ negativos \end{cases}$
- Inicializar os pesos $w_{1,i} = \begin{cases} \frac{1}{2m}\ para\ y = 0 \\ \frac{1}{2l}\ para\ y = 1 \end{cases}$ onde

  $\begin{cases} m = número\ de\ exemplos\ não - face \\ l = número\ de\ exemplos\ face \end{cases}$
- Para t = 1...T:
  - Normalizar os pesos $w_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$, assim w_t é a distribuição de probabilidade
  - Para cada feature, j, treinar um classificador h_j restrito a um único feature.
    - Calcular o erro $e_j = \sum w_i \left| h_j(x_i) - y_i \right|$
  - Escolher o classificador h_t com o menor erro e_t
  - Atualizar os pesos

    $$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i} \text{ onde } \beta = \frac{e_t}{1-e_t} \text{ e}$$

    $$e_i = \begin{cases} 0, se\ x_i\ está\ classificado\ corretamente \\ 1, caso\ contrário \end{cases}$$
- O classificador forte é dado por:

  $h(x) = \begin{cases} 1, se\ \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2}\sum_{t=1}^{T} \alpha_t \\ 0, caso\ contrário \end{cases}$ onde $\alpha_t = \log(1/\beta_t)$

Figura 24: Algoritmo para construir um classificador forte

4.1.4 O Aprendizado em Cascata

Esta seção descreve o algoritmo para a construção de cascatas de classificadores que consegue uma maior performance de detecção do que o AdaBoost. O principio chave do algoritmo é que classificadores menores podem ser construídos para rejeitar muitas sub-imagens negativas enquanto detecta todas as possíveis instâncias de uma face (o limite do classificador impulsionado pode ser ajustado para que os falsos-negativos sejam próximos de zero). Classificadores simples são utilizados para rejeitar a maiorira das sub-imagens enquanto classificadores mais complexos são chamados para diminuir a taxa de falsos positivos.

Os níveis de cascata são construídos treinando classificadores utilizando o algoritmo apresentado na Figura 27, tal que estes são vários classificadores sequenciais, cada um deles com um número menor de classificadores fracos. Começando com um classificador forte de dois *features*, um filtro de face eficiente pode ser obtido ajustando o limite do classificador forte para minimizar falsos-negativos. O limite inicial, $\frac{1}{2}\sum_{t=1}^{T}\alpha_t$, é projetado para minimizar o erro no conjunto de treino. Um limite mais baixo rende taxas superiores de detecção e de falsos-positivos.

O desempenho de detecção de um classificador de dois *features* não é aceitável como um sistema de detecção de objetos. Entretanto o classificador pode diminuir significativamente o numero de sub-imagens que precisam de um processamento mais complexo com poucas operações

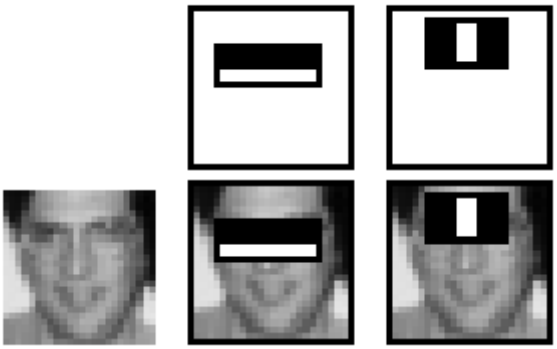

Figura 25: a e b – primeiras *features* do ada-boost, c – imagem a ser classificada, d e e – feature sobreposta a imagem. O primeiro feature mede a diferença de intensidade entre a região dos olhos e a região das bochechas superiores. O segundo feature compara a intensidade da região dos olhos em comparação com o nariz (Fonte: (VIOLA e JONES, 2001))

O processo de detecção tem a forma de uma árvore de decisão degenerada, ou cascata (ver Figura 26). Um resultado positivo do primeiro classificador dispara a avaliação pelo segundo classificador, que foi ajustado para obter altas taxas de detecção. Um resultado positivo no segundo classificador dispara o terceiro e assim por diante. Um resultado negativo em qualquer ponto, leva a rejeição da sub-imagem.

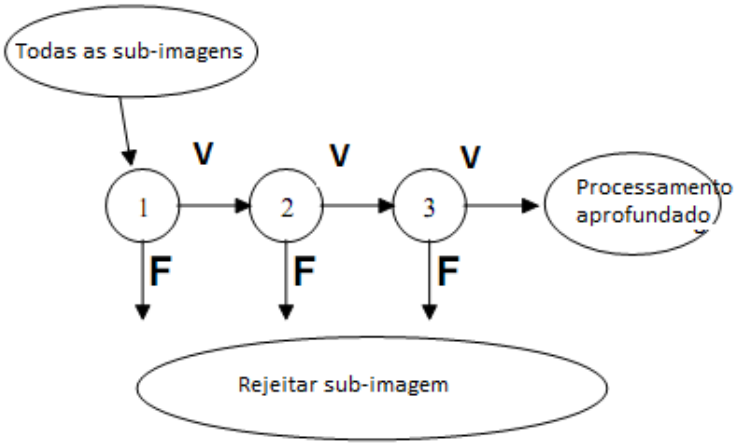

Figura 26 – Processo de detecção em cascata

A estrutura de cascata é construída com base num conjunto de detecção e de metas de desempenho. Dada uma cascata de classificadores, a taxa de falsos positivos da cascata é:

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

Onde F é a taxa de falsos-positivos da cascata de classificadores, K é o número de classificadores e f_i é a taxa de falso-positivo do i-ésimo classificador. A taxa de detecção é:

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

Onde D é a taxa de detecção da cascata de classificadores e d_i é a taxa de detecção do i-ésimo classificador.

Dados os objetivos concretos de taxa de detecção e falsos-positivos, é possível determinar as taxas de cada estágio do processo em cascata. Por exemplo para uma taxa de detecção de 90% pode ser obtida com um classificador de 10 estágios se cada estágio tem taxa de detecção de 0.99 ($0.99^{10} = 0.90$).

O número de *features* avaliadas ao processar imagens reais é necessariamente um processo probabilístico. Dada uma sub-imagem qualquer, ela progredirá na cascata, um classificador por vez, até que seja determinado se a imagem é uma face ou não. O comportamento esperado deste processo é determinado pela distribuição das imagens num conjunto de testes. A medida chave de cada classificador é a taxa de positivos (a proporção de imagens que estão marcadas como potenciais faces). O número de *features* que serão avaliados é dado por:

$$N = n_0 + \sum_{i=1}^{K} \left(n_i \prod_{j<i} p_j \right)$$

Onde N é o número esperado de *features* a serem avaliados, K é o número de classificadores, p_i é a taxa de positivos do i-ésimo classificador e n_i é o número de *features* no i-ésimo classificador.

O processo de treinamento envolve algumas considerações a serem feitas. Na maioria dos casos quanto mais *features* forem avaliadas, maior será a taxa de detecção e menor será a taxa de falsos-positivos, entretanto será necessário mais tempo para avaliar a imagem. A principio, pode-se definir um modelo de otimização no qual são considerados:

- O número de estágios de classificadores
- O número de *features* de cada estágio
- O limite de cada estágio

Para minimizar o número de esperado de *features* N, dado um objetivo F e D. Entretanto fazer esta otimização é um problema difícil.

Na prática um modelo muito simples é utilizado para produzir um classificador eficiente e efetivo. São selecionadas as taxas mínimas aceitáveis para f_i e d_i. Cada camada da cascata é treinada utilizando AdaBoost como explicado acima com o número de *features* usado, aumentando até termos os objetivos de taxa de detecção e falsos-positivos para aquele nível. As taxas são testadas testando o classificador num conjunto de imagens de validação. Se a taxa de falsos-positivos não tiver sido atingida, mais uma camada é adicionada a cascata. O algotmo está explicado mais detalhadamente na figura abaixo.

- Selecionar os valores de f, a máxima taxa de falsos-positivos por camada.
- Selecionar os valores de d, a taxa mínima de detecção por camada.
- Selecionar a taxa F_{obj} de falsos-positivos do classificador final.
- P = conjunto de exemplos positivos (face)
- N = conjunto de exemplos negativos (não-face)
- $F_o = 1$
- $D_o = 1$
- I = 0
- Enquanto $F_i > F_{obj}$
    - $i = i + 1$
    - $n_i = 0$
    - $F_i = F_{i-1}$
    - Enquanto $F_i > f * F_{i-1}$
        - $n_i = n_{i+1}$
        - Utilizar P e N para treinar o classificador n_i *features* com AdaBoost
        - Avaliar o classificador em cascata atual com o conjunto de validação para determinar F_i e D_i
        - Diminuir o limite para o i-ésimo classificador até que a taxa de detecção da cascata atual seja $d*D_{i-1}$
    - N = 0
    - Se $F_i > F_{obj}$
        - Avaliar o classificador em cascata atual no conjunto de imagens negaticas e inserir as detecções falsas no conjunto N

Figura 27: Algoritmo para construção de detector em cascata

5 SOFTWARE

5.1 OPEN CV

Foi utilizada a biblioteca OpenCV por ser uma biblioteca consagrada e de acesso livre. Possui diversos algoritmos implementados como exemplo além de vasta documentação oficial. É possível encontrar diversos fóruns de discussão na internet que auxiliaram o processo de aprendizagem da linguagem de programação.

OpenCV é uma biblioteca desenvolvida em código C e C++ e é multi-plataforma (pode ser utilizada em Linux, Windows e MacOS). Além disso, possui interface para desenvolvimento em outras linguagens como Python e Ruby. A principal meta do OpenCV é prover a infra-estrutura básica para o desenvolvimento de aplicações de Visão Computacional.

5.2 TRABALHO DESENVOLVIDO

Basicamente desenvolvemos aplicativos baseados na documentação do OpenCV para detecção da face e das características do rosto. Após isto, desenvolvemos um método para detectar o contorno destas características para análise futura das emoções.

5.2.1 Detecção

O algoritmo para detecção da face e das partes do rosto é o mesmo, a única diferença é o Haar Cascade utilizado em cada etapa para classificação das imagens. O código busca o feature desejado conforme o Haar Cascade utilizado e desenha um círculo em volta do objeto encontrado.

Utilizamos para testar o algoritmo e escolher o cascade diversas fotos com iluminação diferentes, com mais de um rosto e outros, entretanto os resultados foram satisfatórios apenas para fotos simples e bem iluminadas.

```
#define CV_NO_BACKWARD_COMPATIBILITY
#include "cv.h"
#include "highgui.h"

#include <iostream>
#include <cstdio>

#ifdef _EiC
#define WIN32
#endif

using namespace std;
using namespace cv;

void detectAndDraw( Mat& img,
                    CascadeClassifier& cascade, CascadeClassifier& nestedCascade,
                    double scale);

String cascadeName =
"../../data/haarcascades/haarcascade_frontalface_alt2.xml"; //Aqui se colocam os
HaarCascades desejados
String nestedCascadeName =
"../../data/haarcascades/haarcascade_frontalface_alt_tree.xml";

int main( int argc, const char** argv )
{
    CvCapture* capture = 0;
    Mat frame, frameCopy, image;
    const String scaleOpt = "--scale=";
    size_t scaleOptLen = scaleOpt.length();
    const String cascadeOpt = "--cascade=";
    size_t cascadeOptLen = cascadeOpt.length();
    const String nestedCascadeOpt = "--nested-cascade";
    size_t nestedCascadeOptLen = nestedCascadeOpt.length();
    String inputName;

    CascadeClassifier cascade, nestedCascade;
    double scale = 1;

    for( int i = 1; i < argc; i++ )
    {
        if( cascadeOpt.compare( 0, cascadeOptLen, argv[i], cascadeOptLen ) == 0 )
            cascadeName.assign( argv[i] + cascadeOptLen );
        else if( nestedCascadeOpt.compare( 0, nestedCascadeOptLen, argv[i],
nestedCascadeOptLen ) == 0 )
        {
            if( argv[i][nestedCascadeOpt.length()] == '=' )
                nestedCascadeName.assign( argv[i] + nestedCascadeOpt.length() + 1 );
            if( !nestedCascade.load( nestedCascadeName ) )
                cerr << "WARNING: Could not load classifier cascade for nested objects" <<
endl;
        }
        else if( scaleOpt.compare( 0, scaleOptLen, argv[i], scaleOptLen ) == 0 )
        {
            if( !sscanf( argv[i] + scaleOpt.length(), "%lf", &scale ) || scale < 1 )
                scale = 1;
        }
        else if( argv[i][0] == '-' )
        {
```

```
                cerr << "WARNING: Unknown option %s" << argv[i] << endl;
            }
            else
                inputName.assign( argv[i] );
        }

        if( !cascade.load( cascadeName ) )
        {
            cerr << "ERROR: Could not load classifier cascade" << endl;
            cerr << "Usage: facedetect [--cascade=\"<cascade_path>\"]\n"
                "   [--nested-cascade[=\"nested_cascade_path\"]]\n"
                "   [--scale[=<image scale>\n"
                "   [filename|camera_index]\n" ;
            return -1;
        }

        if( inputName.empty() || (isdigit(inputName.c_str()[0]) && inputName.c_str()[1] ==
'\0') )
            capture = cvCaptureFromCAM( inputName.empty() ? 0 : inputName.c_str()[0] -
'0' );
        else if( inputName.size() )
        {
            image = imread( inputName, 1 );
            if( image.empty() )
                capture = cvCaptureFromAVI( inputName.c_str() );
        }
        else
            image = imread( "lena.jpg", 1 );

        cvNamedWindow( "result", 1 );

        if( capture )
        {
            for(;;)
            {
                IplImage* iplImg = cvQueryFrame( capture );
                frame = iplImg;
                if( frame.empty() )
                    break;
                if( iplImg->origin == IPL_ORIGIN_TL )
                    frame.copyTo( frameCopy );
                else
                    flip( frame, frameCopy, 0 );

                detectAndDraw( frameCopy, cascade, nestedCascade, scale );

                if( waitKey( 10 ) >= 0 )
                    goto _cleanup_;
            }

            waitKey(0);
        _cleanup_:
            cvReleaseCapture( &capture );
        }
        else
        {
            if( !image.empty() )
            {
                detectAndDraw( image, cascade, nestedCascade, scale );
```

```
                waitKey(0);
            }
            else if( !inputName.empty() )
            {
                /* assume it is a text file containing the
                list of the image filenames to be processed - one per line */
                FILE* f = fopen( inputName.c_str(), "rt" );
                if( f )
                {
                    char buf[1000+1];
                    while( fgets( buf, 1000, f ) )
                    {
                        int len = (int)strlen(buf), c;
                        while( len > 0 && isspace(buf[len-1]) )
                            len--;
                        buf[len] = '\0';
                        cout << "file " << buf << endl;
                        image = imread( buf, 1 );
                        if( !image.empty() )
                        {
                            detectAndDraw( image, cascade, nestedCascade, scale );
                            c = waitKey(0);
                            if( c == 27 || c == 'q' || c == 'Q' )
                                break;
                        }
                    }
                    fclose(f);
                }
            }
        }

        cvDestroyWindow("result");

        return 0;
    }

    void detectAndDraw( Mat& img,
                        CascadeClassifier& cascade, CascadeClassifier& nestedCascade,
                        double scale)
    {
        int i = 0;
        double t = 0;
        vector<Rect> faces;
        const static Scalar colors[] =  { CV_RGB(0,0,255),
            CV_RGB(0,128,255),
            CV_RGB(0,255,255),
            CV_RGB(0,255,0),
            CV_RGB(255,128,0),
            CV_RGB(255,255,0),
            CV_RGB(255,0,0),
            CV_RGB(255,0,255)} ;
        Mat gray, smallImg( cvRound (img.rows/scale), cvRound(img.cols/scale),
CV_8UC1 );

        cvtColor( img, gray, CV_BGR2GRAY );
        resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
        equalizeHist( smallImg, smallImg );

        t = (double)cvGetTickCount();
```

```
            cascade.detectMultiScale( smallImg, faces,
                1.1, 2, 0
                //|CV_HAAR_FIND_BIGGEST_OBJECT
                //|CV_HAAR_DO_ROUGH_SEARCH
                |CV_HAAR_SCALE_IMAGE
                ,
                Size(30, 30) );
            t = (double)cvGetTickCount() - t;
            printf( "detection time = %g ms\n", t/((double)cvGetTickFrequency()*1000.) );
            for( vector<Rect>::const_iterator r = faces.begin(); r != faces.end(); r++, i++ )
            {
                Mat smallImgROI;
                vector<Rect> nestedObjects;
                Point center;
                Scalar color = colors[i%8];
                int radius;
                center.x = cvRound((r->x + r->width*0.5)*scale);
                center.y = cvRound((r->y + r->height*0.5)*scale);
                radius = cvRound((r->width + r->height)*0.25*scale);
                circle( img, center, radius, color, 3, 8, 0 );
                if( nestedCascade.empty() )
                    continue;
                smallImgROI = smallImg(*r);
                nestedCascade.detectMultiScale( smallImgROI, nestedObjects,
                    1.1, 2, 0
                    //|CV_HAAR_FIND_BIGGEST_OBJECT
                    //|CV_HAAR_DO_ROUGH_SEARCH
                    //|CV_HAAR_DO_CANNY_PRUNING
                    |CV_HAAR_SCALE_IMAGE
                    ,
                    Size(30, 30) );
                for( vector<Rect>::const_iterator nr = nestedObjects.begin(); nr !=
nestedObjects.end(); nr++ )
                {
                    center.x = cvRound((r->x + nr->x + nr->width*0.5)*scale);
                    center.y = cvRound((r->y + nr->y + nr->height*0.5)*scale);
            RADIUS = CVROUND((NR->WIDTH + NR->HEIGHT)*0.25*SCALE);
                    circle( img, center, radius, color, 3, 8, 0 );
                }
            }
            cv::imshow( "result", img );
        }
```

Figura 28: Código fonte para a detecção de *features*

5.2.1.1 Resultados

5.2.1.1.1 Face

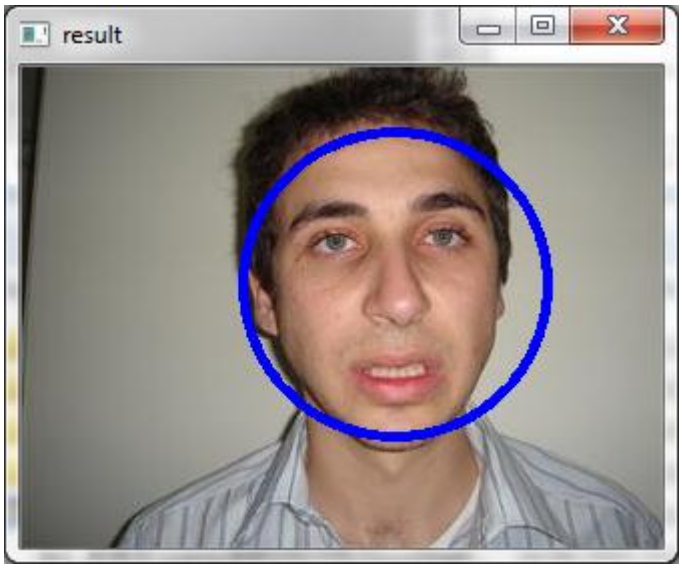

Figura 29 – Resultado para detecção da face

5.2.1.1.2 Olhos

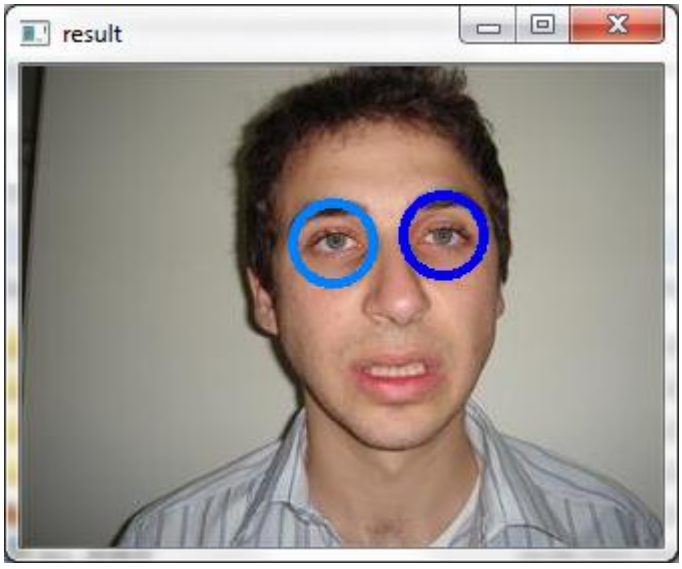

Figura 30 – Resultado para detecção dos olhos

5.2.1.1.3 Nariz

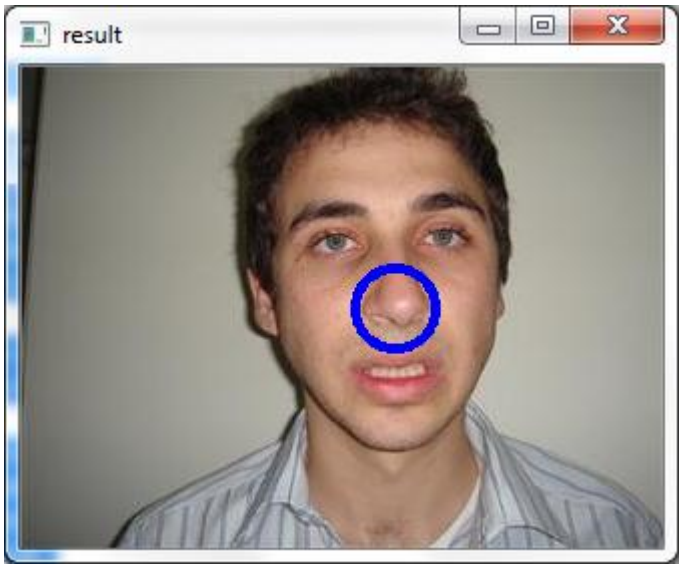

Figura 31 – Resultado para detecção do nariz

5.2.1.1.4 Boca

Figura 32 – Resultado para detecção da boca

5.2.2 Contorno

Utilizamos o algoritmo fornecido com a biblioteca OpenCV para encontrar os contornos da face para a análise da posição e angulação dos *features*.

```
#ifdef _CH_
#pragma package <opencv>
#endif

#define CV_NO_BACKWARD_COMPATIBILITY

#ifndef _EiC
#include "cv.h"
#include "highgui.h"
#endif

char wndname[] = "Edge";
char tbarname[] = "Threshold";
int edge_thresh = 1;

IplImage *image = 0, *cedge = 0, *gray = 0, *edge = 0;

// define a trackbar callback
void on_trackbar(int h)
{
    cvSmooth( gray, edge, CV_BLUR, 3, 3, 0, 0 );
    cvNot( gray, edge );

    // Run the edge detector on grayscale
    cvCanny(gray, edge, 100, 300, 3);

    cvZero( cedge );
    // copy edge points
    cvCopy( image, cedge, edge );

    cvShowImage(wndname, cedge);
}

int main( int argc, char** argv )
{
    char* filename = argc == 2 ? argv[1] : (char*)"fruits.jpg";

    if( (image = cvLoadImage( filename, 1)) == 0 )
        return -1;

    // Create the output image
```

```
    cedge = cvCreateImage(cvSize(image->width,image->height), IPL_DEPTH_8U, 3);

    // Convert to grayscale
    gray = cvCreateImage(cvSize(image->width,image->height), IPL_DEPTH_8U, 1);
    edge = cvCreateImage(cvSize(image->width,image->height), IPL_DEPTH_8U, 1);
    cvCvtColor(image, gray, CV_BGR2GRAY);

    // Create a window
    cvNamedWindow(wndname, 1);

    // create a toolbar
    // cvCreateTrackbar(tbarname, wndname, &edge_thresh, 100, on_trackbar);

    // Show the image
    on_trackbar(0);

    // Wait for a key stroke; the same function arranges events processing
    cvWaitKey(0);
    cvReleaseImage(&image);
    cvReleaseImage(&gray);
    cvReleaseImage(&edge);
    cvDestroyWindow(wndname);

    return 0;
}

#ifdef _EiC
main(1,"edge.c");
#endif
```

Figura 33: Algoritmo para detecção dos contornos

5.2.2.1 Resultados

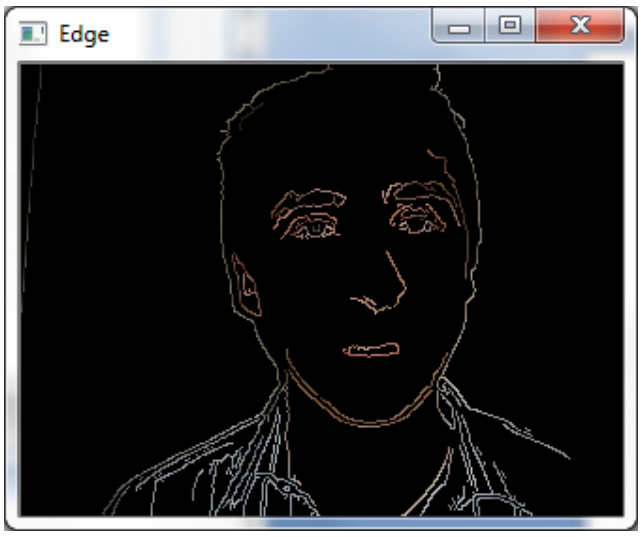

Figura 34 – Resultado para detecção dos contornos

5.3 TABELA DE RESULTADOS

A tabela abaixo apresenta os resultados obtidos com a aplicação dos Haar Cascades descritos acima de 100 imagens com face, olhos, nariz e boca nas condições consideradas adequadas a este trabalho e descritas no item **Erro! Fonte de referência não encontrada.**. E 200 imagens negativas.

Tabela 2 – Resultados dos Haar Cascade

Objeto	Taxa de Detecção	Taxa de Falsos-Positivos
Face	87%	19,5%
Olhos	61%	24,5%
Nariz	63%	32%
Boca	48%	29,5%

A base de fotos positivas foram fotos feitas para o trabalho quando detalhamos os AU's. As fotos negativas foram tiradas de banco de dados da internet (flickr.com)

6 TRATAMENTO DA IMAGEM EMOTIVA

O tratamento da imagem para obter os dados necessários para identificação da emoção expressa, ocorre em diversas fases. É um processo em serie que ocorre separadamente para cada área da face onde os Au's se caracterizam (boca, olhos e sobrancelhas), a saída de uma função adaptativa é a entrada da seguinte, tendo finalmente uma matriz com as coordenadas cartesianas dos 18 pontos desejados para a interpretação.

A figura abaixo ilustra esquematicamente a sequencia realizada sobre a imagem para garantir um resultado conforme esperado e possibilitar uma análise através da rede treinada em MatLab:

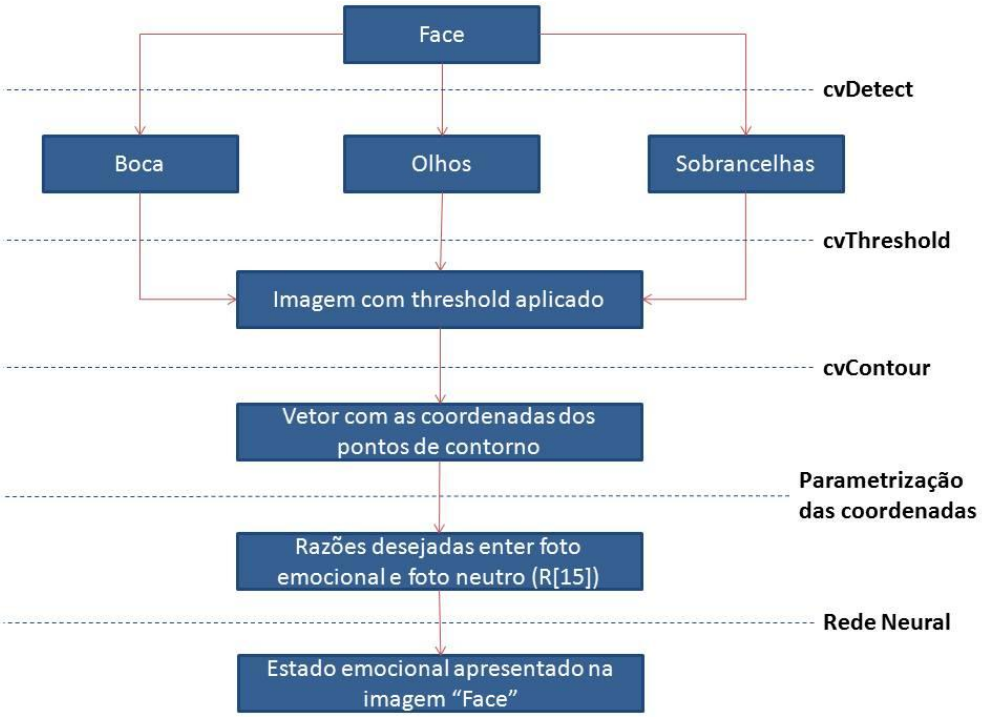

Figura 35 – Processo de tratamento das imagens

O programa de detecção de emoções tem como entrada duas imagens; a primeira apresentando o indivíduo em questão com aparência facial neutra, a segunda mostrando a emoção que deverá ser descrita. Nessa secção

desenvolveremos as características e necessidades de cada uma das funções aplicadas sobre a imagem emotiva resultando nas coordenadas desejadas.

A rede neural utilizada nesse programa precisa de dados coerentes e constantes relacionando a imagem emotiva à neutra, para isso é necessário identificar corretamente a posições de 18 pontos da face utilizados para parametrização das razões de entrada da rede, (as razões serão discutidas na secção 7). O tratamento que será descrito é preciso para que as coordenadas informadas sejam as mais corretas possível.

Devido a diferenças de iluminação, cor, profundidade e textura das partes do rosto optamos por considerar *thresholds* distintos em cada um dos casos, assim primeiramente detectamos a localização das áreas desejadas para então separá-las e processar cada uma separadamente. O método de deteçao de partes da face utilizado é o descrito previamente baseado na teoria de Viola-Jones(VIOLA e JONES, 2001).

6.1 LIMIAR

Com apenas a área em questão sendo analisada (saída da funçao cvDetect), utilizamos a aplicação de um *threshold* sobre a imagem para isolar os contornos desnecessários daqueles que descrevem os pontos extremos dos *features* analisados.

A função de *threshold* escolhida (Threshold Binary), elimina categoricamente qualquer pixel da imagem que possua valor menor que o determinado e torna máximo aqueles cujo valor é maior ou igual. As diferentes opções de *threshold* podem ser vistas na figura abaixo, esses gráficos ilustram o que ocorre com cada pixel dependendo de sua intensidade relaciona diretamente ao parâmetro de entrada.

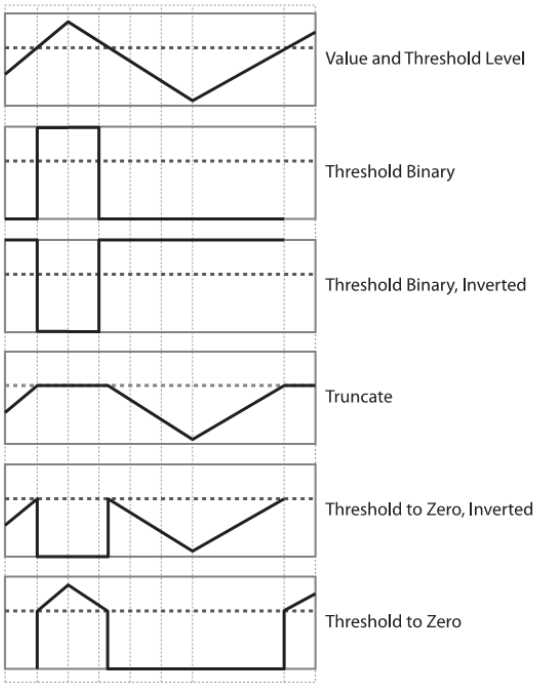

Figura 36 – Limiar FONTE: (BRADSKY e KAEHLER)

A matriz de entrada da função *cvThreshold*, deve ser uma que descreve a imagem em *grayscale*, isso significa que cada posição da matriz indica um pixel e o valor contido nessa posição descreve a grau de cinza daquele pixel, de zero a cem (preto a branco). A imagem apresentada abaixo é o resultado da função aplicada à uma foto da face inteira (figura abaixo). Como é possivel ver, temos uma imagem preta com linhas brancas que indicam os contornos que possuem pixels com o valor indicado pelo *threshold,* que pode ser alterado pelo trackbar acima da figura.

Figura 37 – Imagem com ajuste de Limiar

O *threshold* correto indica o melhor limite para separar cada característica do resto da face e por isso deve ser cuidadosamente escolhido pelo usuário individualmente para os *features,* ele será utilizado na próxima etapa para definição do vetor de coordenadas do contorno.

6.2 CONTOUR

Neste caso, a função *cvContour* necessita como entrada, além da subimagem o valor de threshold sobre o qual os contornos serão detectados. Com essas informações, é possível vetorizar as coordenadas dos pontos que compões cada contorno presente na imagem analisada.

```
contador = 0
maxnum = 0
for (coluna){
num = 0
 for (linha){
    if (pixel == branco ){
contornos [contador(num)][2*num]=coluna
contornos[contador(num)][2*num+1]=linha
contador(num) ++
num++}
 }
If(num>maxnum)
     maxnum = num
}
```

Figura 38 – Algoritmo para vetorização dos contornos

A figura acima apresenta um algoritmo simplificado do método utilizado para detecção da posição dos pixels, e salvá-los no vetor correspondente à cada linha desenhada na imagem. Varremos todos os pixels desejados, coluna-linha, e quando ele identificado como branco e, portanto faz parte da linha, guardamos sua coordenada (x,y) para depois analisar o AU presente. Naturalmente, existem mais de uma linha presentes na subimagem e relevantes à característica em análise, o algoritmo portanto permite guardarmos cada uma delas individualmente e também retorna o número total de linhas vetorizadas (maxnum).

Como resultado temos os contornos localizados, identificados abaixo em vermelho e azul (*in and out* – essa é uma separação entre os vetores para facilitar sua identificação, um contorno demonstra dentro de qual outro contorno eles se encontra e quais são os contornos que estão dentro dele), e as coordenadas informadas para que possamos posteriormente equacioná-las e decifrar a emoção expressa.

```
Contour #26
 2 elements:
    (110,472)
    (110,473)
Contour #27
 1 elements:
    (605,471)
Contour #28
 4 elements:
    (528,471)
    (529,472)
    (529,473)
    (529,472)
Contour #29
 6 elements:
    (490,471)
    (491,470)
    (492,471)
    (492,473)
    (491,474)
    (490,473)
Contour #30
 1 elements:
    (441,471)
Contour #31
 10 elements:
    (175,471)
    (173,473)
    (173,474)
    (171,476)
    (171,477)
    (170,478)
    (172,476)
    (172,475)
    (173,474)
    (173,473)
Contour #32
 4 elements:
    (105,471)
    (104,472)
    (104,473)
    (104,472)
Contour #33
 4 elements:
    (586,470)
```

Figura 39 – Resultado para detecção dos contornos e suas respectivas coordenadas

7 PARÂMETROS COMPARATIVOS PARA DETERMINAÇÃO EMOCIONAL

Para a determinação emocional, utilizamos 18 parâmetros intimamente ligados aos AU's apresentados previamente que descrevem as aparências faciais. Os parâmetros relacionam os principais features faciais como boca, olhos e sobrancelhas, sendo separados da seguinte forma:

Boca – 4 Parâmetros.

Olhos – 2 Parâmetros para cada.

Sobrancelhas – 3 Parâmetros para cada.

Relação Olho-Sobrancelha – 2 Parametros para cada.

Para o treinamento da base de dados e calibração da imagem neutra, o proprio usuario define os 18 pontos necessários. Isso é feito atraves de um programa que solicita ao usuario que clique sobre cada posição separadamente e na ordem correta descrita sobre a imagem. Esse método foi escolhido por permitir maior acuracidade dos pontos e controle do usuário. A figura abaixo demonstra a imgem já com os 18 pontos selecionados e suas coordenadas descritas na janela ao lado.

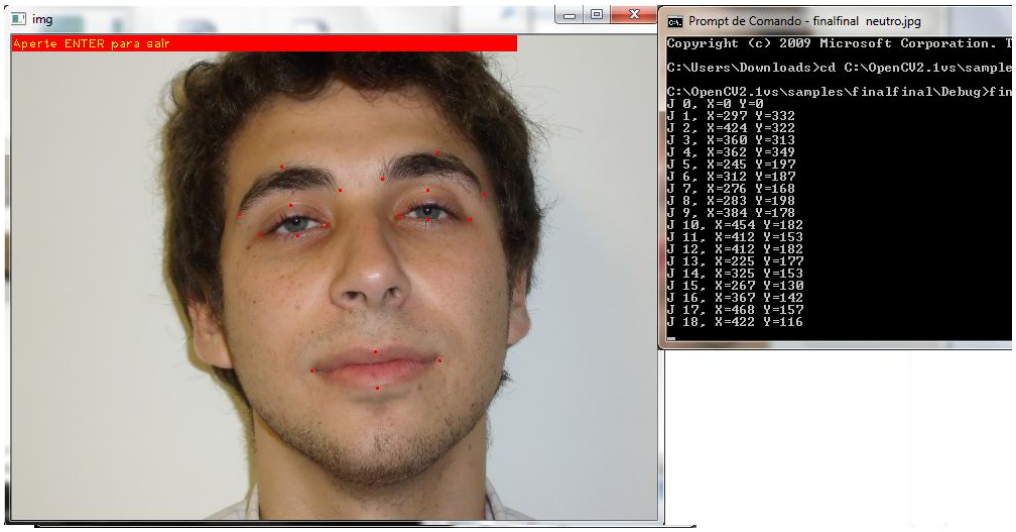

Figura 40 – Determinação manual dos pontos de interesse e suas respectivas coordenadas

7.1 DESCRIÇÃO DOS PARÂMETROS

A figura abaixo, ilustra as medições feitas para obtenção dos parâmetros. As medidas são absolutas e relacionam pontos extremos (figura 40) dos features principais.

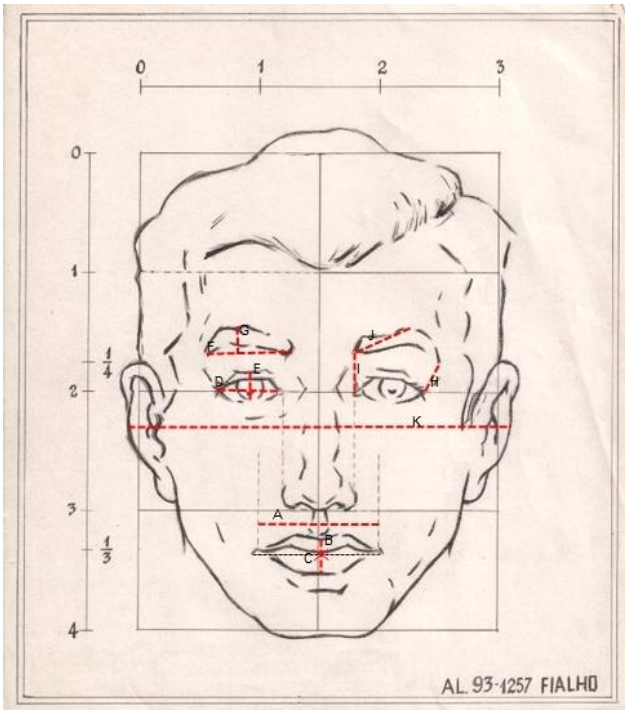

Figura 41 – Parâmetros de avaliação da emoção

(A) – Abertura horizontal da boca, distância entre os extremos dos cantos labiais.
(B) – Abertura vertical da boca, distância entre os extremos superior e inferior da boca.
(C) – Coordenadas do ponto onde as retas que unem os pontos de (A) e (B) se cruzam. Utilizamos isso para medir as distâncias horizontal e vertical deste ponto (centro) para o extremo superior e esquerdo da boca.
(D) – Abertura horizontal ocular, distância entre os extremos horizontais do olho.
(E) – Abertura vertical ocular, distância entre os extremos superior e inferior do olho.
(F) – Comprimento horizontal da sobrancelha, distância entre os extremos horizontais da sobrancelha.
(G) – Distância vertical da sobrancelha, distância vertical entre o ponto superior da sobrancelha e sua projeção na reta que une os pontos de (F).

(H) – Levantamento exterior da sobrancelha, distância entre o ponto exterior da sobrancelha e do olho.
(I) – Levantamento interior da sobrancelha, distância entre o ponto interior da sobrancelha e do olho.
(J) – Distância do ponto superior da sobrancelha, distância entre o ponto superior e o externo esquerdo da sobrancelha.

7.2 NORMALIZAÇÃO

O valor utilizado como entrada da rede neural é sempre a razão do valor medido na foto emotiva sobre o da foto neutra. Isso é importante para garantir uma coerência relativa entre os valores considerados, sendo sempre em relação à imagem neutra do indivíduo em questão. Optamos por uma normalização simples pois não existia necessidade de uniformizar os valores de outra forma.

Também foi considerado um parâmetro adicional (K), que mede a distância entre dois pontos da face que não sofram deformação pela manifestação da emoção. Esse parâmetro foi utilizado como fator de correção para alterações de distância e pequenos desvios de angulação entre as fotos neutra e emotiva.

8 REDES NEURAIS ARTIFICIAIS

Uma maneira eficiente de resolver problemas complexos é seguindo o lema “dividir para conquistar”. Um sistema complexo pode ser decomposto em diversos elementos mais simples para ser possível entendê-lo. Além disso, elementos mais simples também podem ser unidos para compor um sistema mais complexo. A aplicação de redes neurais é um dos jeitos de se fazer essa “decomposição”.

Uma rede é caracterizada pelos seguintes componentes: um conjunto de nós e conexões entre os nós. Os nós podem ser considerados unidades computacionais que recebem dados, processam e entregam dados de saída. Este processo pode ser bem simples como um função de soma, ou pode conter uma rede dentro do próprio nó. As conexões determinam o fluxo de informação entre os nós, podem ser unidirecionais ou bidirecionais.

A especificidade das Redes Neurais Artificiais é que elas veem os nós como neurônios artificiais. Um neurônio artificial é um modelo computacional inspirado nos neurônios reais. Neurônios biológicos recebem os sinais através de sinapses localizadas nos dendritos. Quando os sinais recebidos são fortes o suficiente (superam um certo limite pré-estabelecido), o neurônio é ativado e emite um sinal através do axônio. Este sinal pode ser enviado através de outra sinapse e pode ativar outros neurônios também

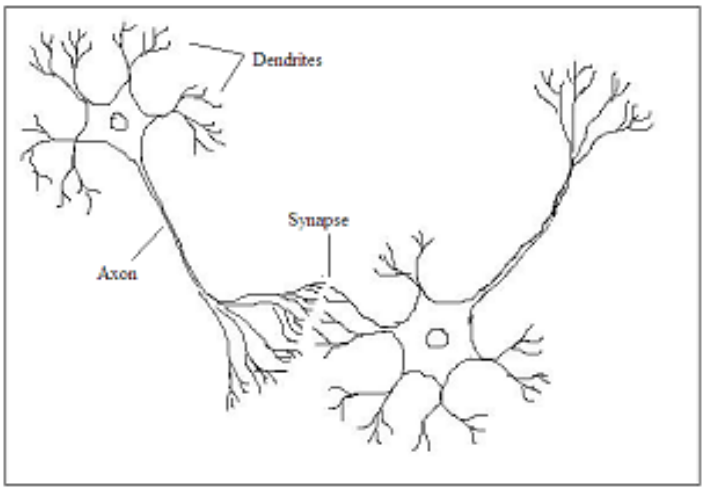

Figura 42 –Sinapse no sistema nervoso, FONTE: (PAKNIKAR, 2008)

A complexidade de neurônios reais é bastante abstrata quando modelamos neurônios artificiais. Estes, basicamente consistem de entradas (como sinapses), que são multiplicadas por pesos (força dos respectivos sinais) e depois computadas por uma função matemática que determina a ativação do neurônio. Outra função calcula a saída do neurônio artificial. Uma RNA combina neurônios artificiais para processar a informação.

Quanto maior o peso de um neurônio artificial, a entrada dele será multiplicada por um número mais forte. Pesos também podem ser negativos, pode-se dizer que o sinal é inibido com um peso negativo. Dependendo dos pesos, os cálculos de um neurônio serão diferentes. Ajustando os pesos de um neurônio artificial, podemos obter as saídas desejadas a partir das entradas fornecidas. Mas quando temos uma RNA de centenas de milhares de neurônios é bastante complexo e para isso existem os algoritmos de treino e aprendizado da rede neural que ajustam esses pesos para que dadas valores de entrada e saídas esperadas, a rede é montada com os respectivos pesos.

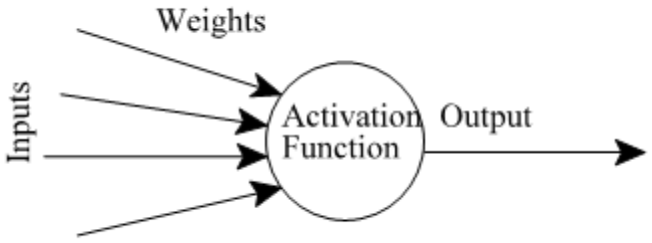

Figura 43 –Modelo típico de um neurônio artificial, FONTE: (PAKNIKAR, 2008)

8.1 O ALGORITMO DE APRENDIZADO – RETROPROPAGAÇÃO

O algoritmo de *backpropagtion* ou retropropagação é usado em redes neurais de camadas com alimentação “frontal”, isto significa que os neurônios artificiais estão organizados em camadas e enviam os seus sinais “para frente” e depois os erros são propagados “para trás”. A rede recebe os dados de entrada por neurônios na “camada de entrada” e a saída é dada pela “camada de saída”. Podem existir um ou mais “camadas escondidas”.

O algoritmo usa aprendizado supervisionado, ou seja, é provido ao algoritmo exemplos de entradas e as respectivas saídas desejadas que a rede deve devolver e assim o erro (diferença entre o valor esperado e o valor que a rede entregou como saída) é calculado. O treinamento começa com pesos randômicos e o objetivo é ajustar eles iterativamente para que o erro seja mínimo.

A função de ativação utilizada pelo algoritmo de retropropagação é simplesmente a soma ponderada das entradas x_i multiplicada pelos pesos w_{ji}:

$$A_j\left(\overline{x},\overline{w}\right)=\sum_{i=0}^{n}x_i w_{ji}$$

Podemos ver que a ativação depende apenas das entradas e dos pesos.

Se a função de saída for a função identidade (saída = ativação), o neurônio será chamado “linear”, mas o mais comum é utilizar a função sigmoide como saída:

$$O_j\left(\overline{x},\overline{w}\right)=\frac{1}{1+e^{A_j\left(\overline{x},\overline{w}\right)}}$$

A função sigmoide resulta em 1 para para grandes números positivos, 0,5 para 0 e -1 para grandes números negativos. Isto permite uma transmissão suave entre a saída alta e baixa do neurônio. Verifica-se que a saída depende exclusivamente da ativação, que por sua vez depende dos valores das entradas e seus respectivos pesos.

O objetivo do processo de treinamento é obter as saídas desejadas dadas certas entradas específicas. Já que o erro é a diferença entre a saída atual e a desejada, o erro depende dos pesos e portanto o algoritmo ajusta os pesos para minimizar o erro. Pode-se definir a função erro da saída de cada neurônio da seguinte forma:

$$E_j\left(\overline{x},\overline{w},d\right)=\left(O_j\left(\overline{x},\overline{w}\right)-d_j\right)^2$$

Tomamos o quadrado da difernça entre a saída obtida e a desejada para que seja sempre positivo. O erro da rede será simplesmente a soma destes erros:

$$E\left(\overline{x},\overline{w},\overline{d}\right)=\sum_j\left(O_j\left(\overline{x},\overline{w}\right)-d_j\right)^2$$

O algoritmo calcula em seguida a dependência dos erros para as entradas, saídas e pesos. E os pesos são ajustados usando o método gradiente descendente:

$$\Delta w_{ji}=-\eta\frac{\partial E}{\partial w_{ji}}$$

Esta fórmula pode ser interpretada da seguinte maneira: o ajuste de cada peso (Δw_{ji}) será o negativo da constante η multiplicada pela dependência do peso anterior no erro da rede. O tamanho do ajuste dependerá de η e da contribuição de cada peso para o erro da função. Isto é, se o peso contribui fortemente para o erro, o ajuste será maior do que para os pesos que contribuem menos para o erro. A equação acima é utilizada até que se ache pesos apropriados para minimizar o erro.

Portanto o objetivo do algoritmo é encontrar a derivada do E em função dos w_{ji}. Primeiramente calcula-se quanto o erro varia com a saída:

$$\frac{\partial E}{\partial O_j} = 2\left(O_j - d_j\right)$$

E em seguida calcula-se quanto que a saída depende da ativação (que depende dos pesos):

$$\frac{\partial O_j}{\partial w_{ji}} = \frac{\partial O_j}{\partial A_j}\frac{\partial A_j}{\partial w_{ji}} = O_j(1 - O_j)x_i$$

Obtemos então:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial O_j}\frac{\partial O_j}{\partial w_{ji}} = 2\left(O_j - d_j\right)O_j(1 - O_j)x_i$$

E o ajuste para cada peso é:

$$\Delta w_{ji} = -2\eta\left(O_j - d_j\right)O_j(1 - O_j)x_i$$

Pode-se utilizar esta última expressão para redes de duas camadas, entretanto para redes com mais camadas é necessário fazer algumas considerações. Se queremos ajustar os pesos (chamaremos de v_{ik}) de uma camada anterior, antes é necessário calcular como o erro depende não só do peso mas da entrada da camada anterior. Portanto é necessário alterar o x_i com w_{ji} nas equações acima. Também é necessário verificar como o erro depende do ajuste de v_{jk}, portanto:

$$\Delta v_{ik} = -\eta\frac{\partial E}{\partial v_{ik}} = -\eta\frac{\partial E}{\partial x_i}\frac{\partial x_i}{\partial v_{ik}}$$

Aonde:

$$\frac{\partial E}{\partial w_{ji}} = 2\left(O_j - d_j\right)O_j(1 - O_j)w_{ji}$$

E assumindo que há entradas u_k no neurônio com v_{ik}

$$\frac{\partial x_i}{\partial v_{ik}} = x_i(1 - x_i)v_{ik}$$

Para adicionar mais camadas, basta calcular como o erro depende das entradas e dos pesos da primeira camada.

8.2 TREINAMENTO DA REDE NO MATLAB®

Utilizamos o programa Matlab para treinar a rede neural utilizando o manual do Toolbox de Redes Neurais como guia (DEMUTH, BEALE e HAGAN, 2008)

Primeiramente tratamos a saída do programa em OpenCV no Excel para exportar ao Matlab

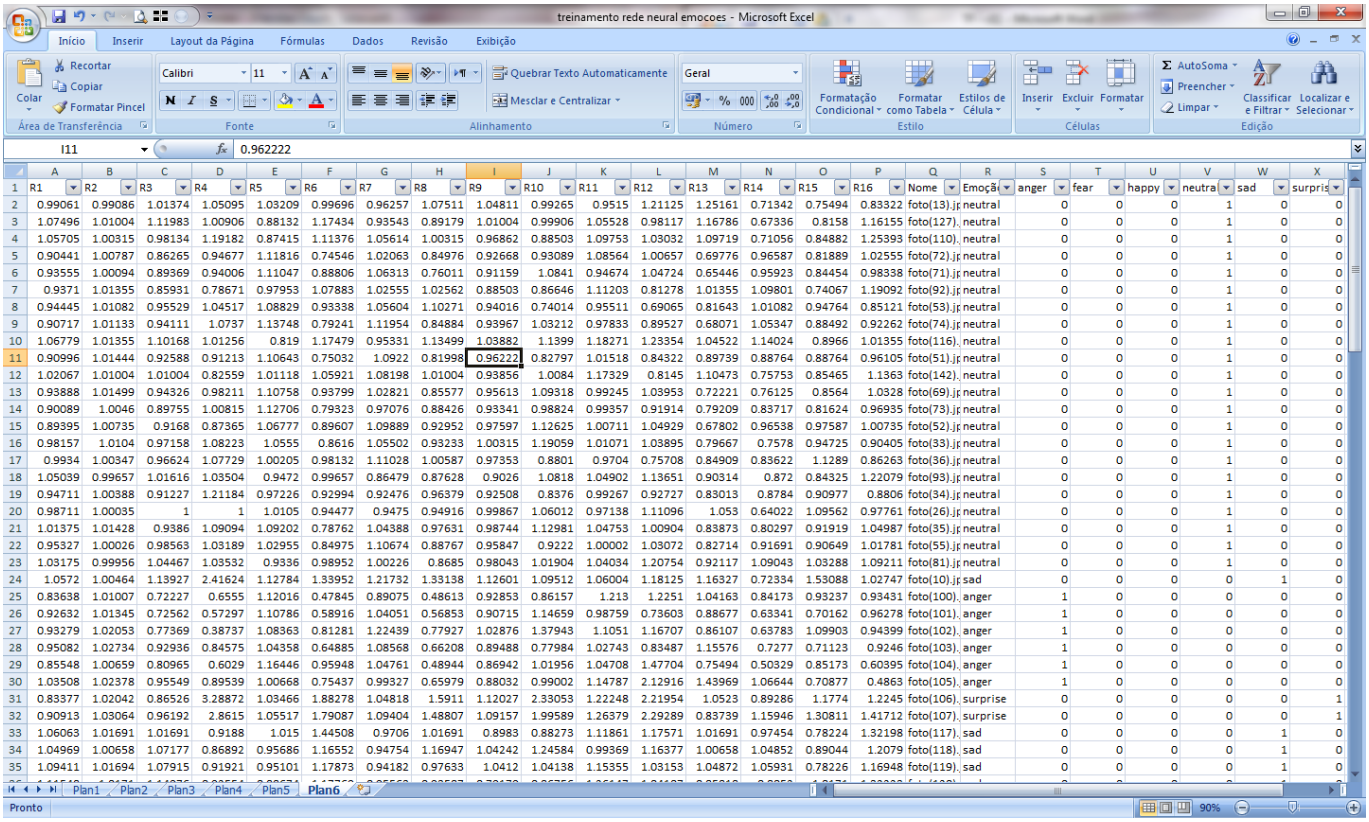

Figura 44 –Excel com os dados do OpenCV

No Matlab criamos duas matrizes: EmotionInputs com as colunas representando os parâmetros e as linhas as fotos lidas e EmotionTargets uma matriz onde as colunas representam as emoções tendo o valor “1” na emoção desejada e “0” nas outras colunas, as linhas também representam as fotos lidas.

Utilizamos o GUI para redes neurais de reconhecimento de padrões *nprtool:*

Basicamente temos que inputar as variáveis que representam os dados para criar a rede neural e escolher o número de Camadas Escondidas.

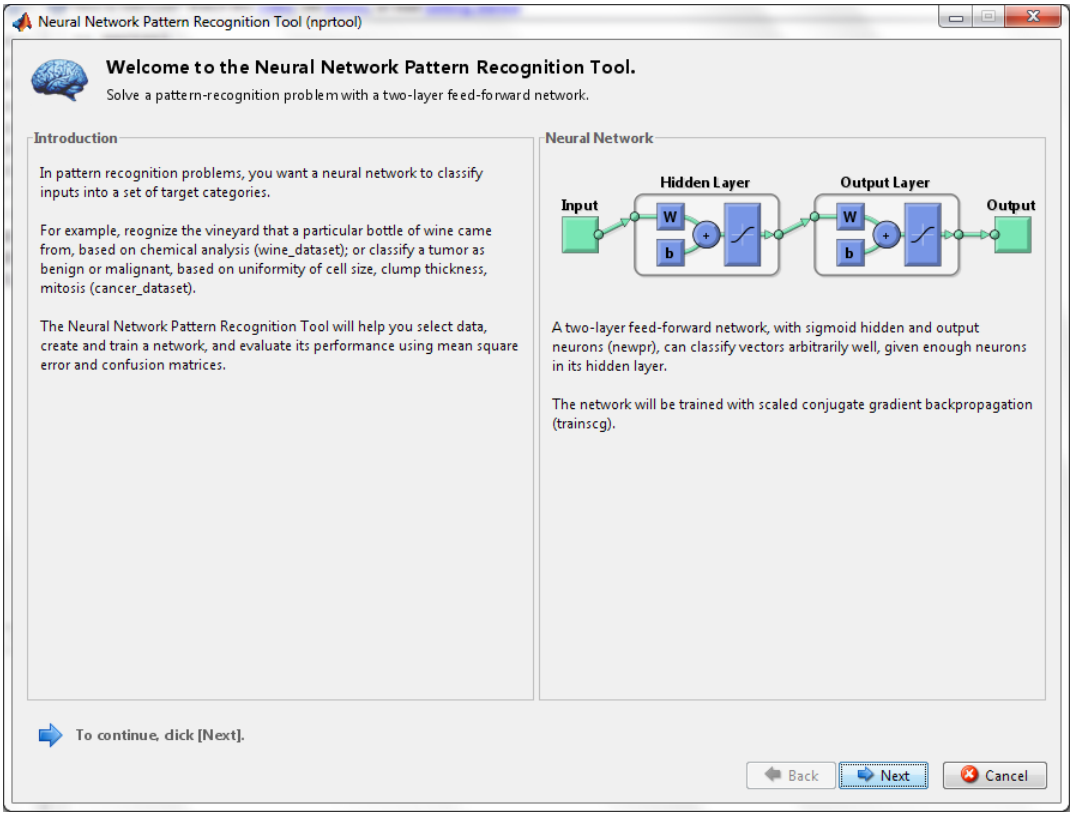

Figura 45 –Tela inicial da criação da rede

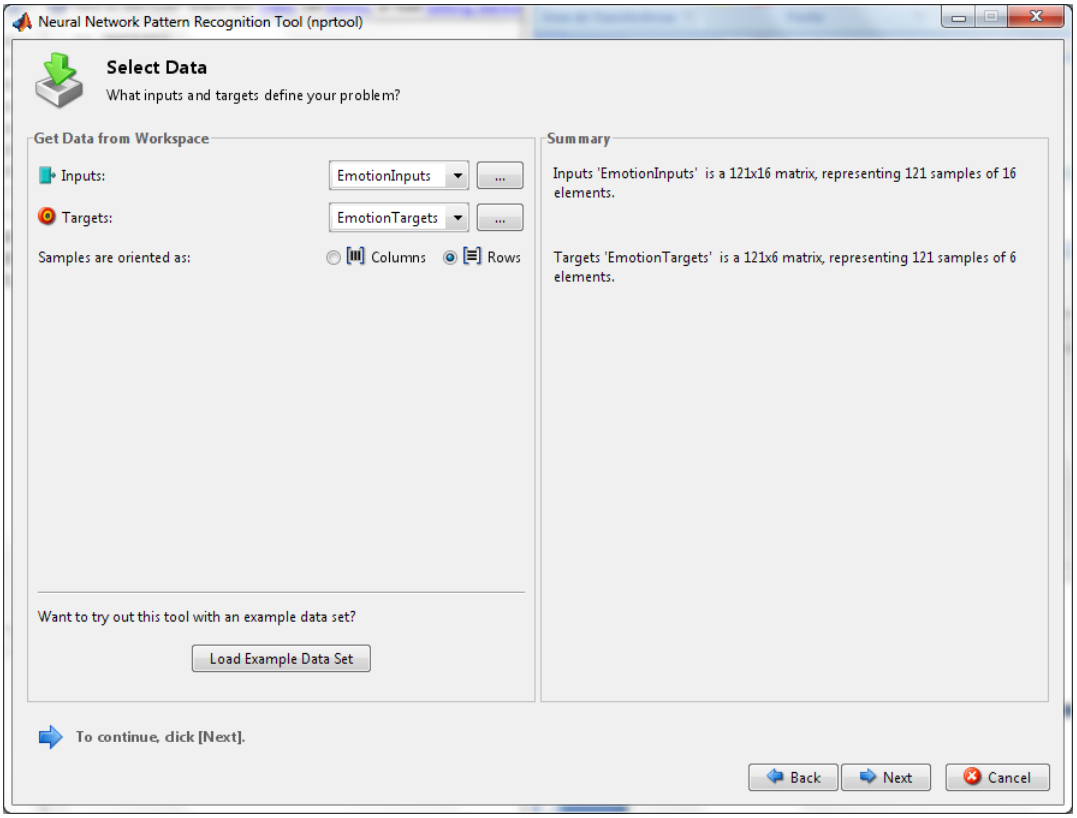

Figura 46 –Escolha das variáveis de entrada

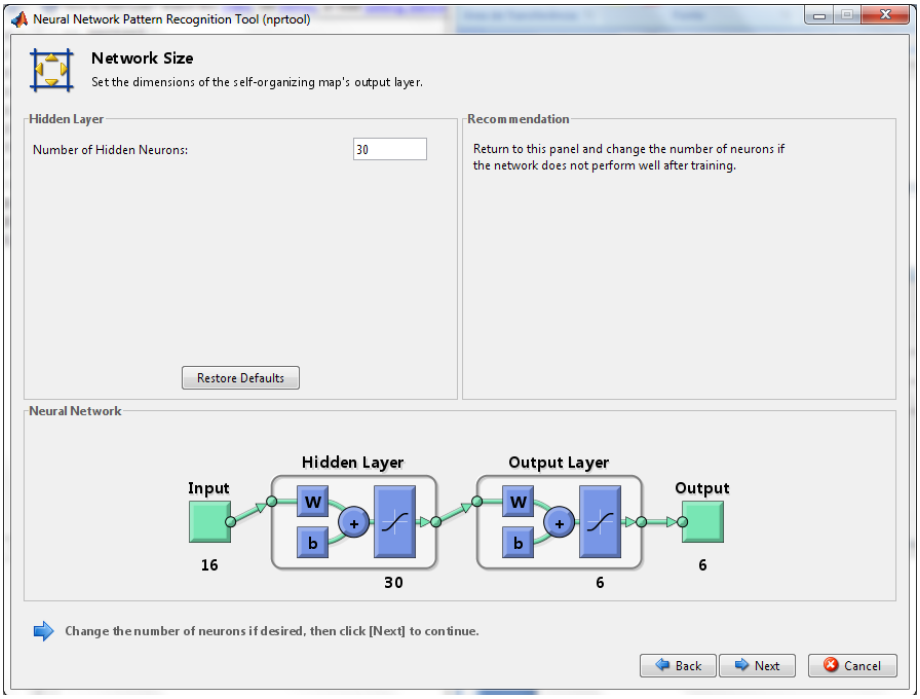

Figura 47 –Escolha do número de camadas escondidas.

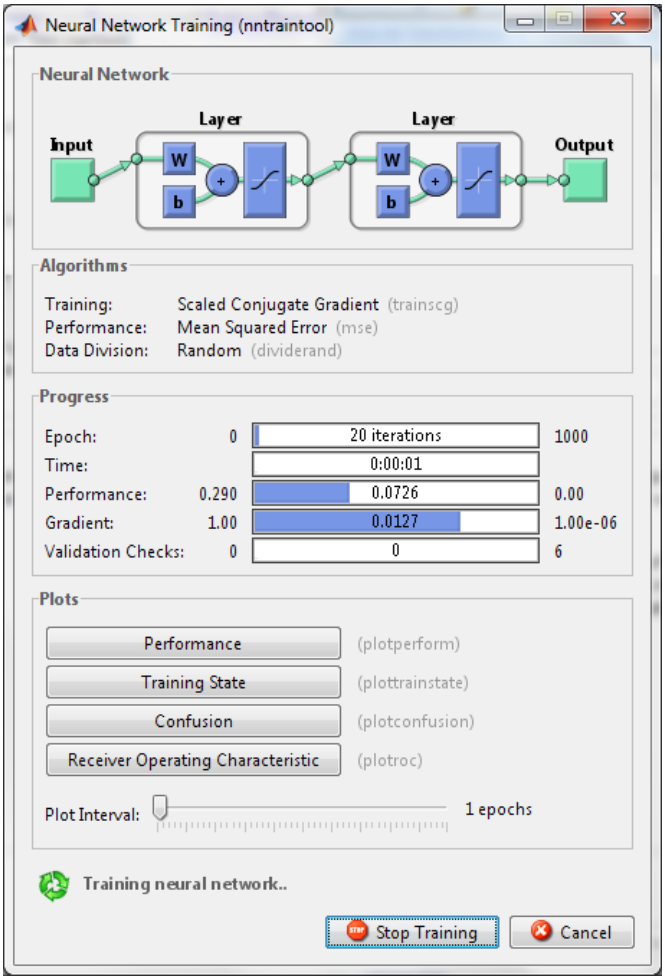

Figura 48 –Treinamento utilizando retropropagação

9 RESULTADOS FINAIS PARA DETECÇÃO DE EMOÇÃO

9.1 RESULTADO DA REDE NEURAL PARA A BASE DE DADOS DE TREINAMENTO

Após a criação da rede neural e feito o seu treinamento o Matlab testa todos os dados inputados para a rede para avaliar o erro da mesma.

Na figura abaixo as linhas das atrizes são as emoções de Saída da rede e as colunas são as emoções esperadas para os dados de entrada. Os quadrados verdes são os identificados corretamente e os vermelhos o inverso. A ordem das emoções é: Raiva, Medo, Felicidade, Neutro, Tristeza e Surpresa.

Podemos ver na figura que a média geral do erro das variáveis de entrada foi de 3,4% após algumas iterações de treinamento e escolha do número de camadas escondidas da rede neural. Vale ressaltar que os pontos para cálculo dos parâmetros caracterizadores de emoção nesta etapa de treinamento foram tomados manualmente, na próxima sessão apresentamos os resultados do programa final baseado na detecção dos features e nos seus contornos vetorizados.

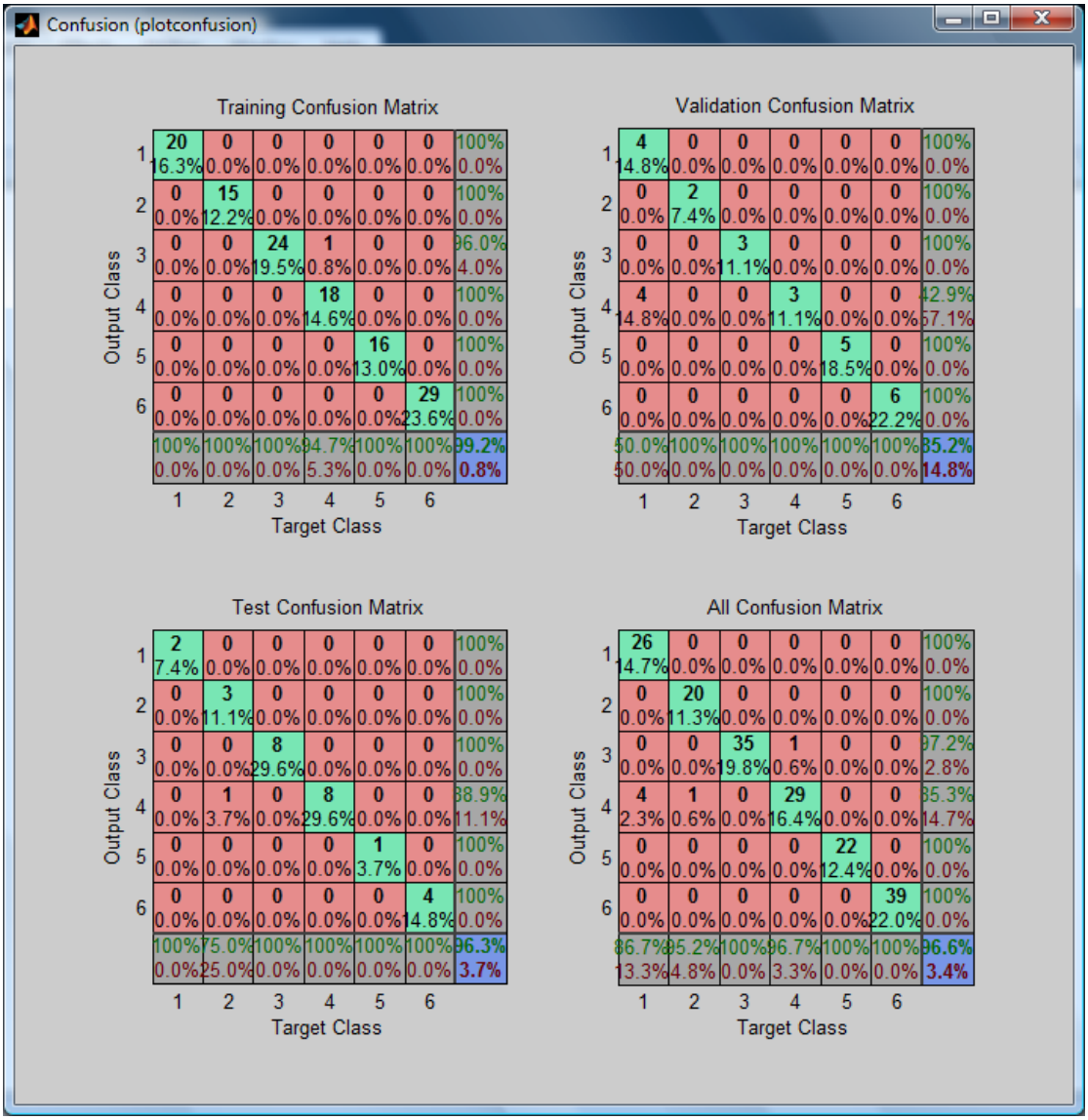

Figura 49 –Matrizes – "Confusion" que representam os acertos e erros da rede para cada conjunto de imagens

9.2 RESULTADOS PROGRAMA FINAL

A tabela abaixo mostra os resultados para os testes efetuados pelo programa final após a rede neural ter sido construída utilizando imagens do banco de dados Cohn-Kanade entretanto de um grupo diferente de fotos das utilizadas para o treinamento.

		Emoção Detectada pela rede Neural						
		Raiva	Medo	Feliz	Neutro	Triste	Surpreso	Total
Emoção Apresentada	Raiva	7			1	2		10
	Medo	2	5			1	2	10
	Feliz			8	1		1	10
	Neutro	2	1	1	4	2		10
	Triste	2			2	6		10
	Surpreso		2	1			7	10
Total		13	8	10	8	11	10	61,7%

Figura 50 –Matriz de resultados programa final

10 ANÁLISE DOS RESULTADOS

10.1 PARÂMETROS COMPARATIVOS PARA DETERMINAÇÃO EMOCIONAL

10.1.1 Abertura Horizontal da Boca

De acordo com o esperado é possível ver pela figura abaixo que o R1 pode ser usado para identificar parcialmente algumas emoções. Sua presença pode ser notada principalmente para felicidade, medo e tristeza, onde há uma abertura horizontal da boca, e para surpresa onde o efeito é o contrário.

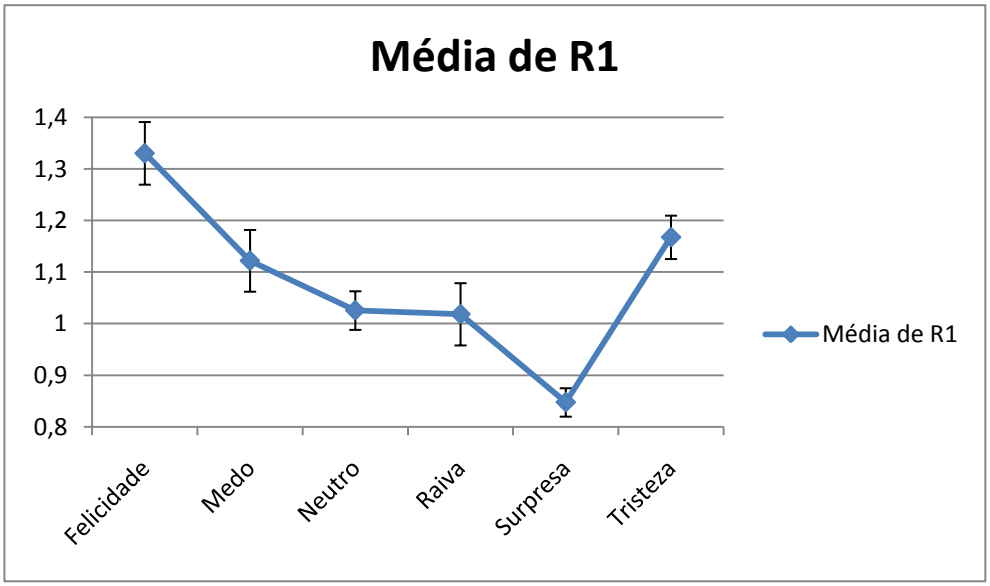

Figura 51 –Média e desvio padrão do R1 para cada emoção

10.1.2 Abertura Vertical da Boca

Para o parâmetro R2 o efeito é contrário ao do acima descrito, nesse caso uma grande alteração pode ser percebida para surpresa.

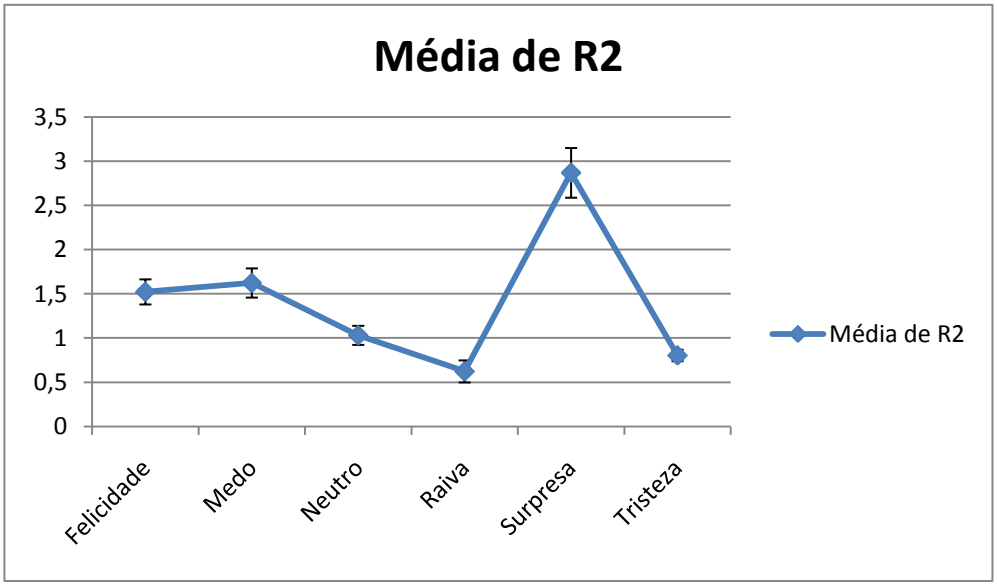

Figura 52 –Média e desvio padrão do R2 para cada emoção

10.1.3 Distância Horizontal do centro da boca

A distância horizontal do ponto central da boca, permite uma análise similar à percebida no R1 (abertura horizontal). Neste caso, novamente felicidade, medo e tristeza apresentam grandes distorções.

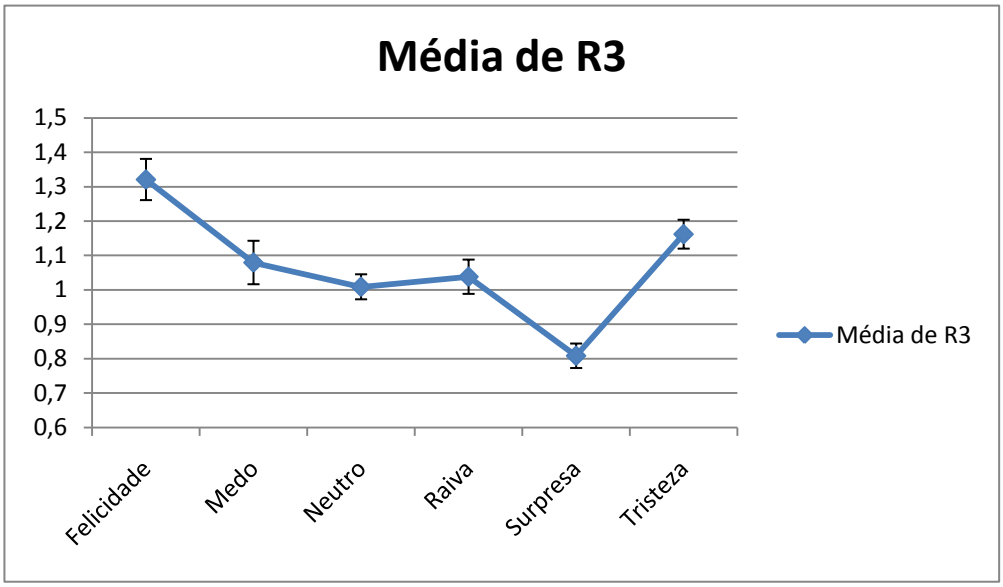

Figura 53 –Média e desvio padrão do R3 para cada emoção

10.1.4 Distância Vertical do centro da boca

O gráfico abaixo permite perceber que o parâmetro R4 apresenta um efeito combinado da abertura oral com o movimento elevatório das laterais da boca. Esse desvio é perceptível quando comparamos a distância vertical entre o centro da boca (onde a reta que une as extremidades horizontais cruza a reta das extremidades verticais) com o ponto labial superior. Assim temos um alto valor para surpresa (acentuado pela abertura da boca) e tristeza (causado pelo movimento para baixo dos cantos da boca), no caso da alegria o efeito é oposto.

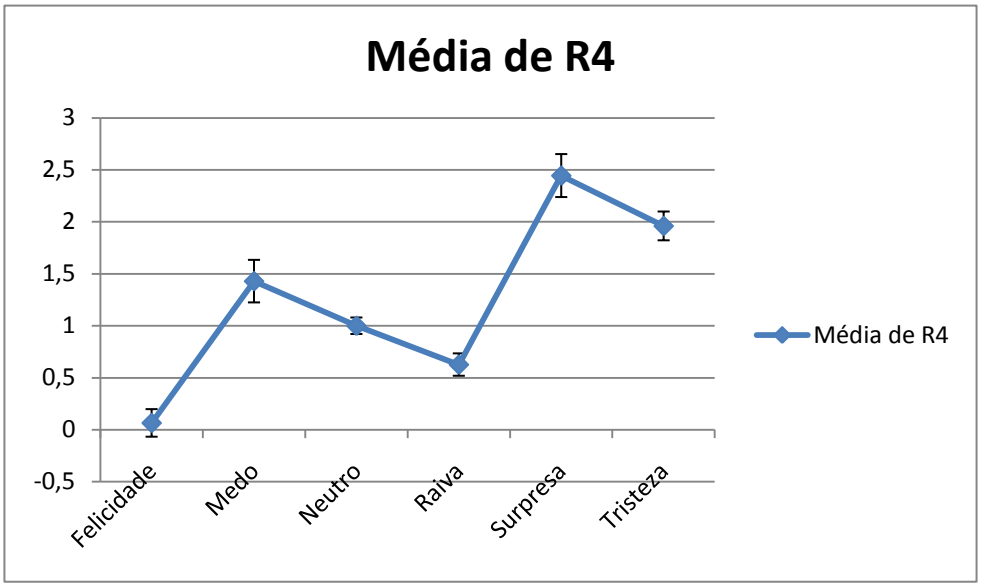

Figura 54 –Média e desvio padrão do R4 para cada emoção

10.1.5 Abertura horizontal ocular

Percebemos pelo gráfico abaixo que este parâmetro possui pouca influência na caracterização das emoções, sendo pouco necessário em trabalhos futuros.

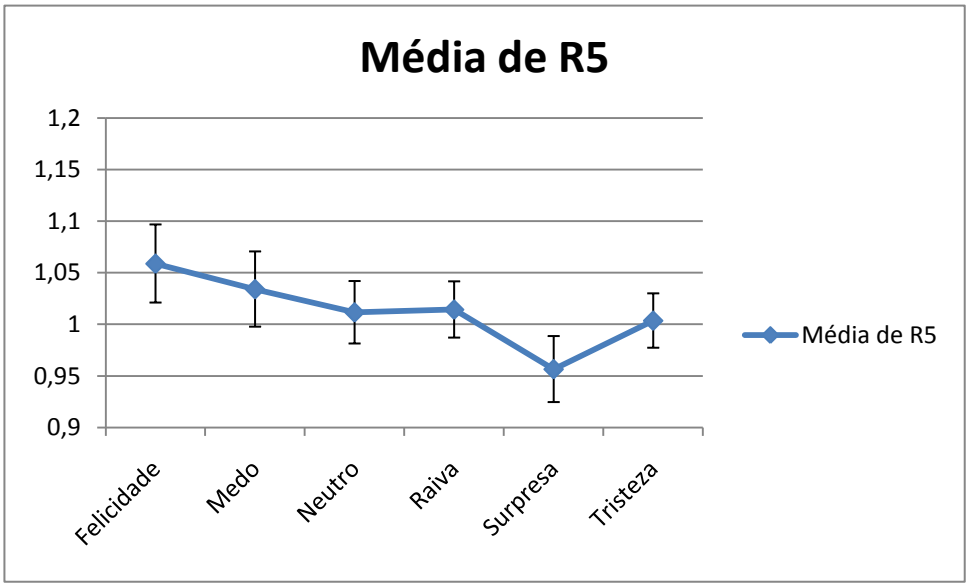

Figura 55 –Média e desvio padrão do R5 para cada emoção

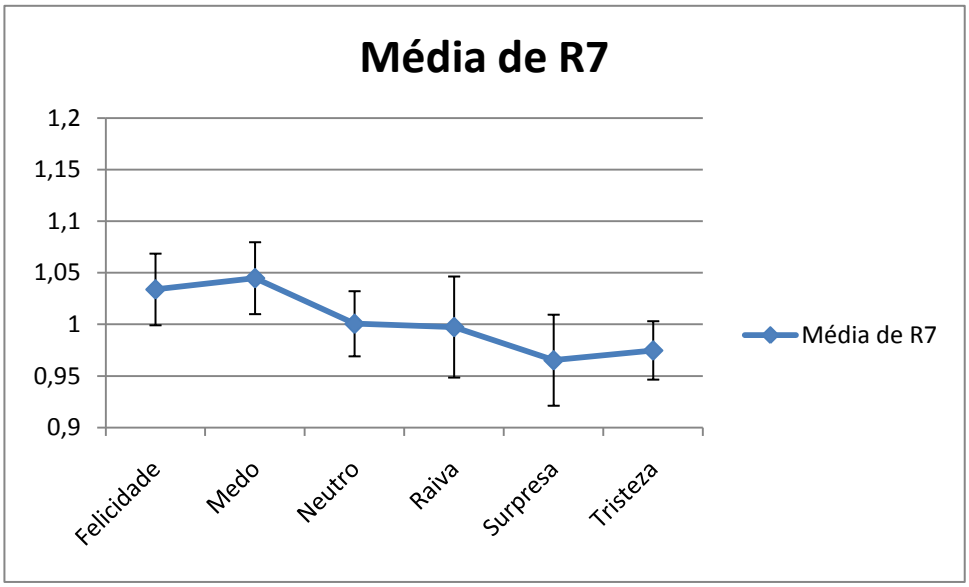

Figura 56 –Média e desvio padrão do R7 para cada emoção

10.1.6 Abertura vertical ocular

Para emoções como surpresa há uma abertura expressiva dos olhos e o efeito inverso para raiva.

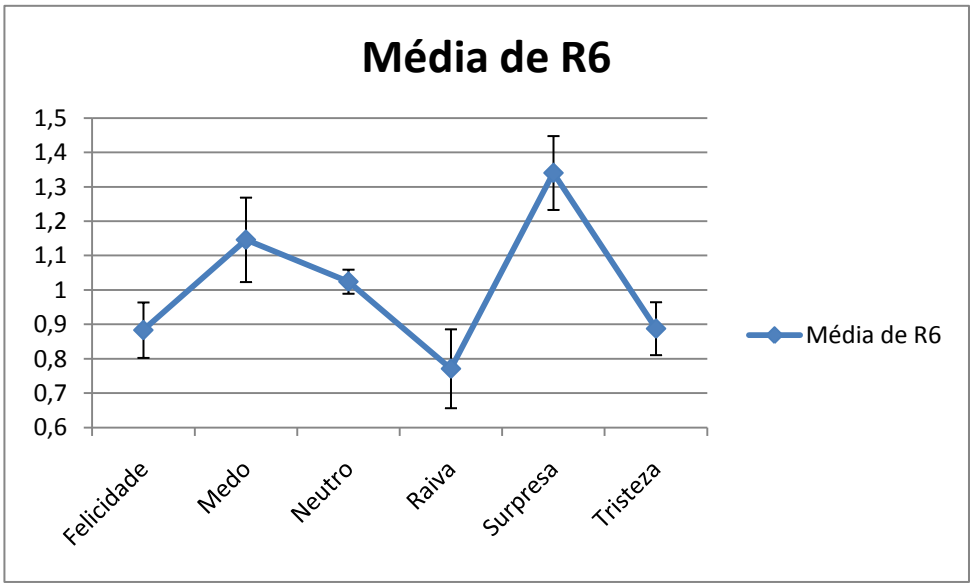

Figura 57 –Média e desvio padrão do R6 para cada emoção

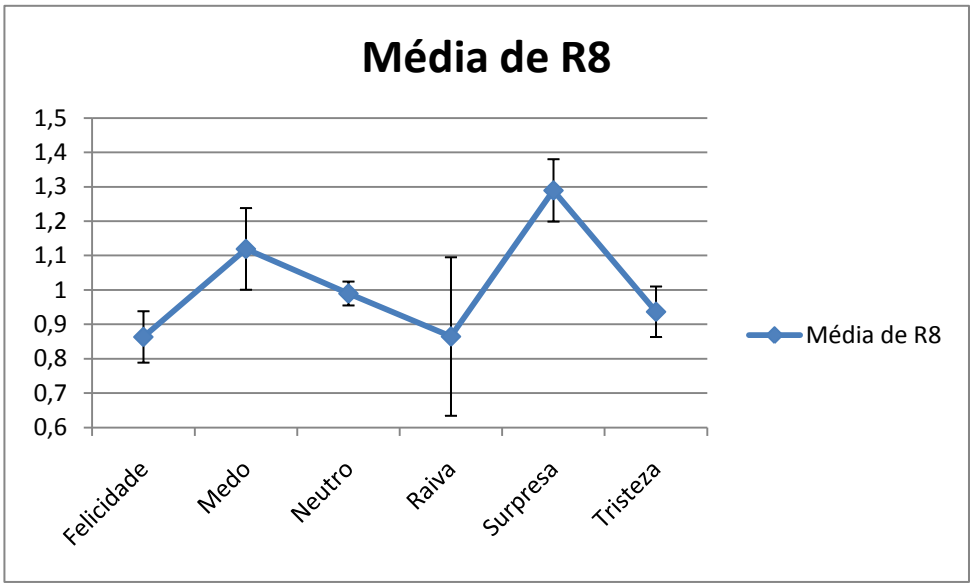

Figura 58 –Média e desvio padrão do R8 para cada emoção

10.1.7 Comprimento horizontal da sobrancelha

Novamente vemos pelos gráficos abaixo que estes parâmetros pouco influenciam na determinação da emoção.

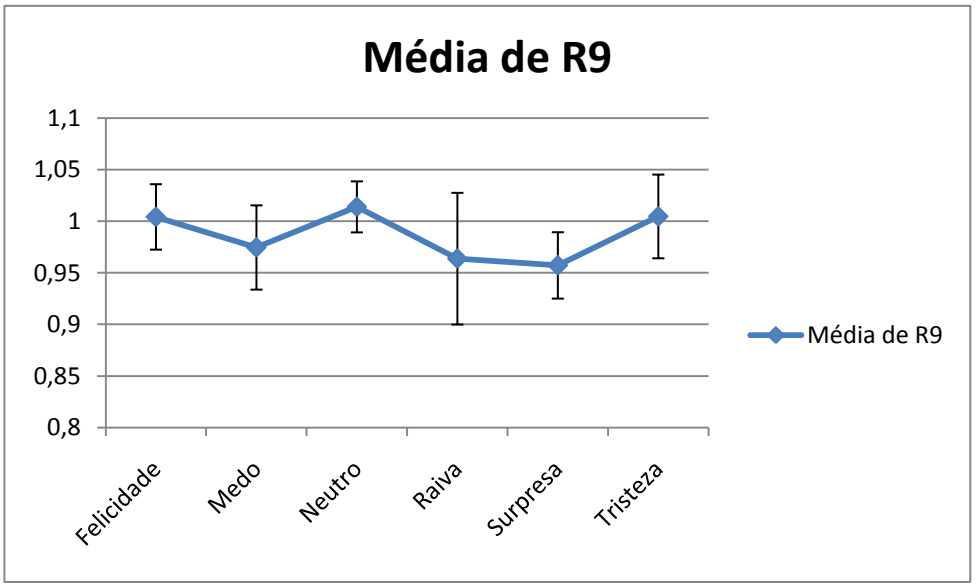

Figura 59 –Média e desvio padrão do R9 para cada emoção

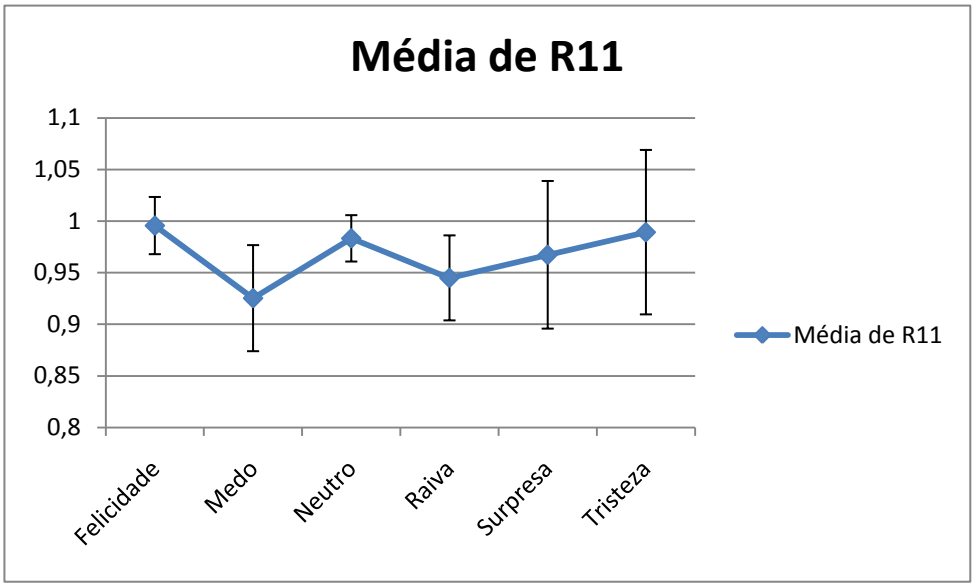

Figura 60 –Média e desvio padrão do R11 para cada emoção

10.1.8 Comprimento vertical da sobrancelha

Conforme esperado pela teoria dos FACS (EKMAN, FRIESEN e HAGER, 2002) o comprimento vertical da sobrancelha aumenta para surpresa e felicidade e diminui para raiva.

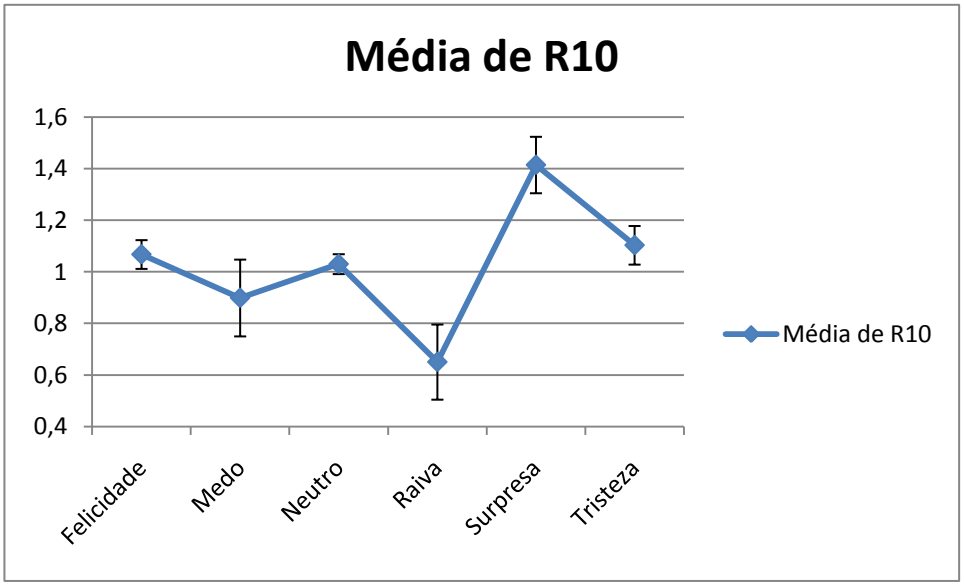

Figura 61 –Média e desvio padrão do R10 para cada emoção

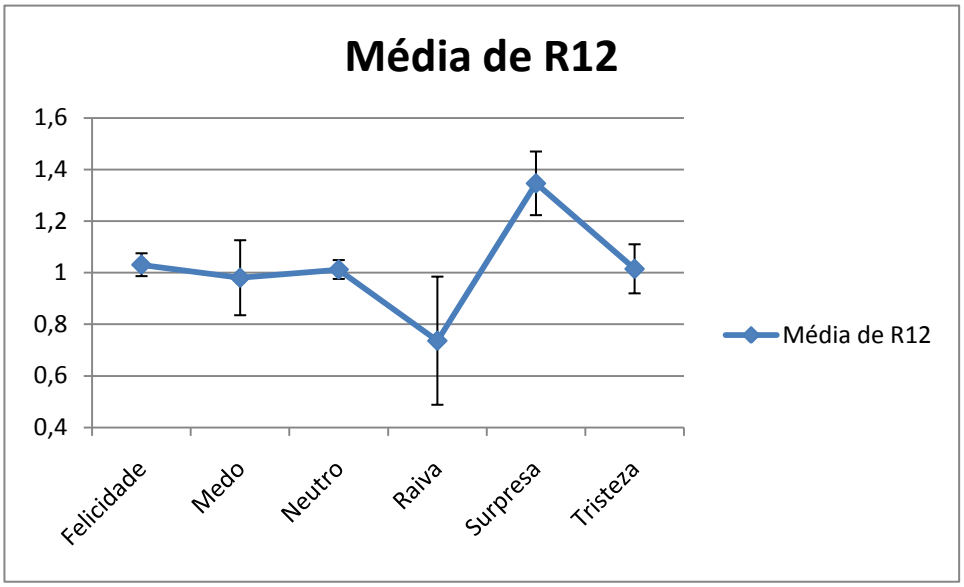

Figura 62 –Média e desvio padrão do R12 para cada emoção

10.1.9 Distância do ponto superior da sobrancelha

Há um deslocamento horizontal do ponto superior da sobrancelha para a raiva e para surpresa, nos outros casos o parâmetro permanece próximo à constante.

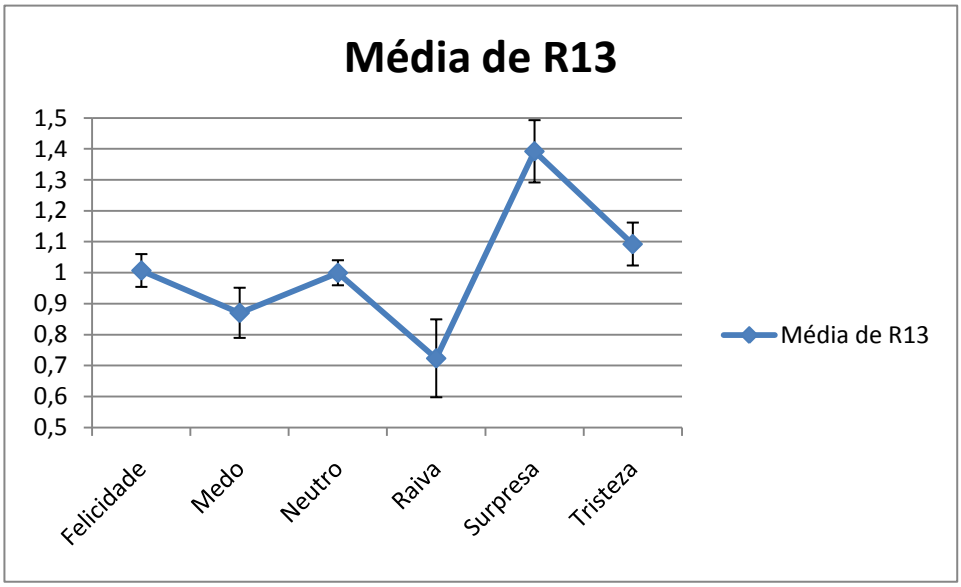

Figura 63 –Média e desvio padrão do R13 para cada emoção

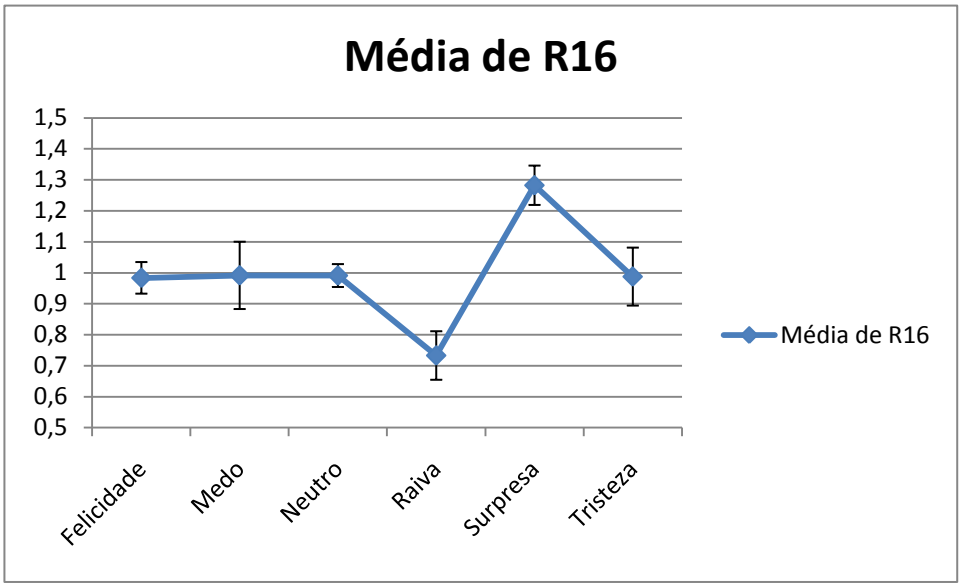

Figura 64 –Média e desvio padrão do R16 para cada emoção

10.1.10 Levantamento exterior da sobrancelha

Pode-se perceber que há um levantamento considerável da sobrancelha para surpresa.

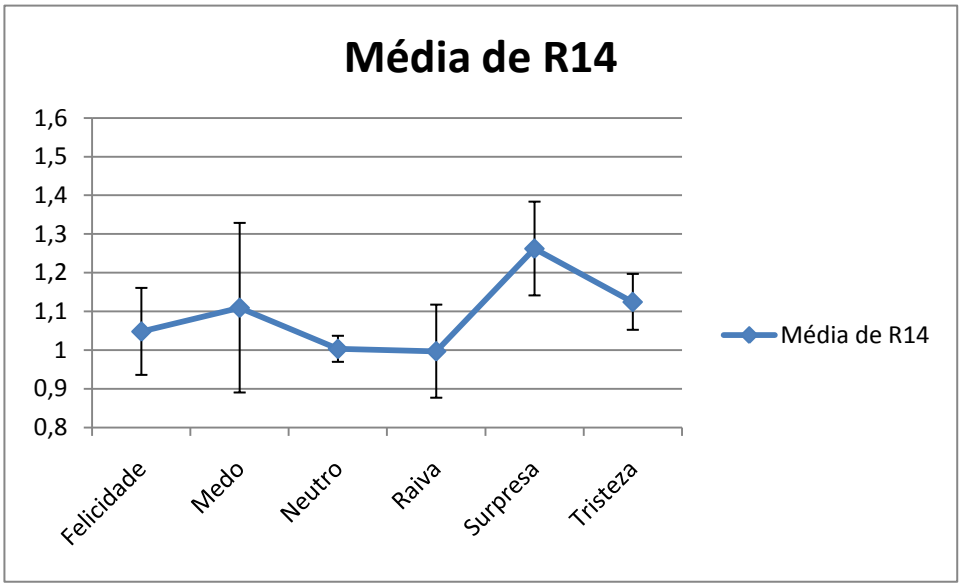

Figura 65 –Média e desvio padrão do R14 para cada emoção

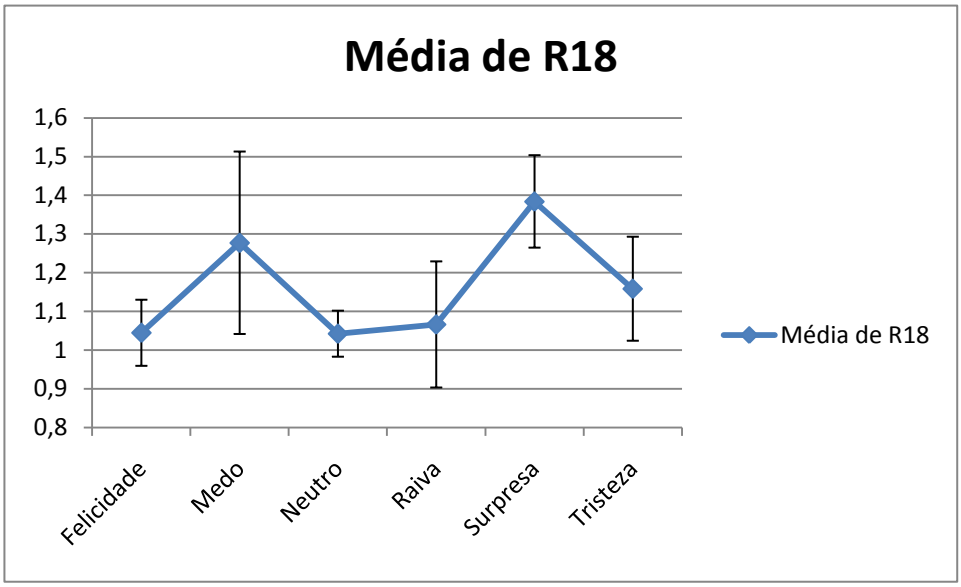

Figura 66 –Média e desvio padrão do R18 para cada emoção

10.1.11 Levantamento interior da sobrancelha

O efeito causado à parte interior da sobrancelha é similar àquele apresentado para a porção externa, no entanto existe também uma separação entre olho e sobrancelha para medo e tristeza e uma aproximação no caso de raiva.

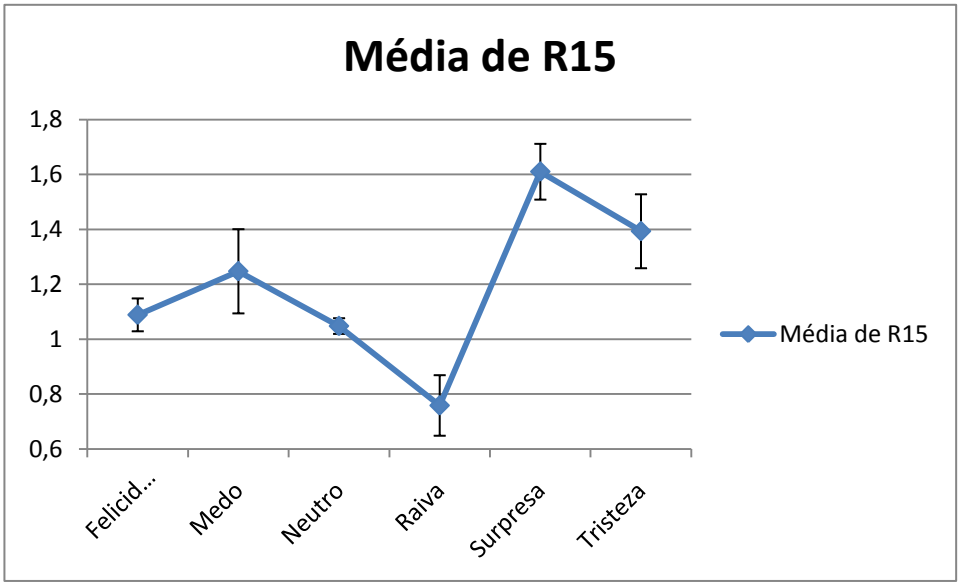

Figura 67 –Média e desvio padrão do R15 para cada emoção

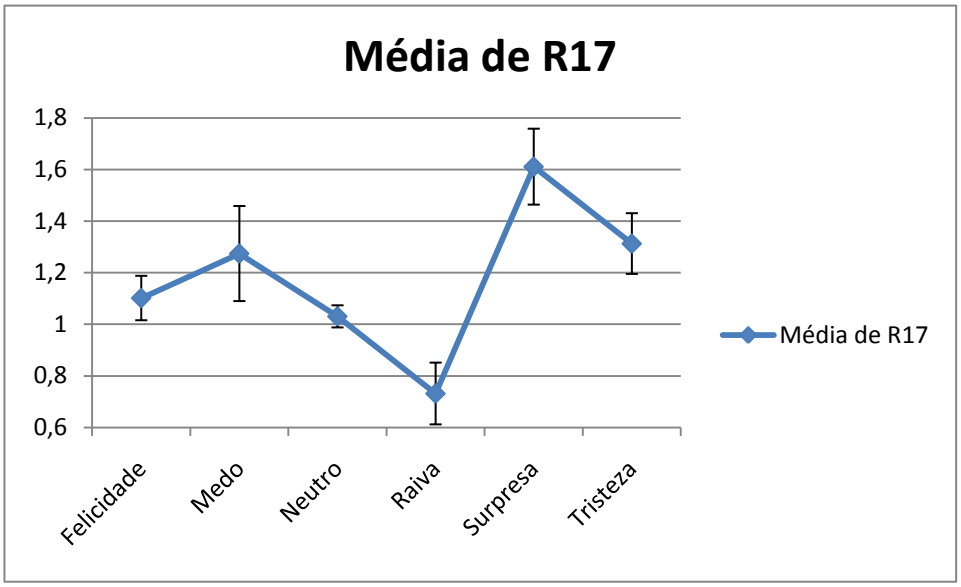

Figura 68 –Média e desvio padrão do R17 para cada emoção

10.2 ANÁLISE DOS RESULTADOS DE DETECÇÃO DA EMOÇÃO

Atingimos com o programa desenvolvido um nível satisfatório de reconhecimento emocional, aproximadamente 62% conforme visto na secção anterior. Este valor está próximo dos valores encontrados na literatura para softwares de semelhante atuação e é cerca de quatro vezes maior do que a escolhe arbitrária de uma das seis emoções, melhorando a probabilidade de acerto para um programa semiautomatizado.

Identificamos que grande parte dos erros é causada por condições específicas das imagens, como por exemplo luminosidade, que dificultam a correta vetorização dos contornos dos *features* necessários.

Conforme esperado as emoções que apresentam características únicas (entre os AU's considerados) tiveram maior índice de deteção correta. Esse é o caso de:

- Felicidade – parâmetro negativo ou nulo de distâcia vertical entre 'centro' da boca e ponto labial superior (80% de acerto).
- Surpresa – aumento significante dos parâmetros de distancia vertical da boca e entre olhos e sobrancelhas (70% de acerto).
- Medo – aumento combinado do tamanho da cavidade oral, horizontal e vertical (70% de acerto).

10.3 LIMITAÇÕES

O trabalho realizado possui algumas limitações que podem ser minimizadas em projetos futuros:

- Exigência de ajuste manual do *threshold* – Devido a importância do limite de *threshold* para correta determinação dos contornos optamos por realizar um ajuste manual do mesmo. Isto é um problema, pois impede a automação completa do software e pode interferir no resultado. Uma solução seria fazer o tratamento da luminosidade das fotos para cada parte do rosto e assim implementar detecção automática do *threshold* necessário para melhor aquisição dos contornos da face.
- Exigência de inputar duas fotos para a detecção – Esta é uma decisão de implementação do projeto visando melhores resultados de detecção da emoção.

- O programa perde acurácia caso as imagens apresentem grandes inclinações ou que o rosto esteja parcialmente coberto por acessórios ou cabelo.

11 CONCLUSÕES

O software proposto neste trabalho consegue efetivamente classifica fotos de faces humanas nas seis emoções tratadas (Raiva, Medo, Felicidade, Neutro, Tristeza e Surpresa). Considerando o número de emoções que escolhemos detectar e que a base de dados de testes é inteiramente diferente da base de dados de treinamento da rede neural a taxa de detecção das emoções é satisfatória comparando com os trabalhos disponíveis na literatura sobre o assunto.

Os dezoito parâmetros escolhidos são suficientes para classificar as fotos nas emoções selecionadas, possibilitando que qualquer individuo consiga classificar sua emoção exigindo no mínimo duas imagens sendo uma neutra e uma com a emoção a ser detectada.

A utilização de parâmetros comparativos entre imagem emotiva e neutral facilita a detecção da emoção em comparação com o método proposto inicialmente de analisar matematicamente os vetores dos contornos das faces. Entretanto exige que tenhamos as duas imagens disponíveis.

11.1 TRABALHOS FUTUROS

A principal mudança deve ser a automação completa do programa através do tratamento da luminosidade conforme descrito acima.

Também é bastante interessante incluir parâmetros binários para auxilias o reconhecimento da emoção, como por exemplo, se há enrugamento da testa, se os dentes estão visíveis, enrugamento da área próxima ao nariz, entre outros.

Outra importante melhoria seria a integração do software em C++ com o Matlab, podendo talvez implementar as redes neurais no ambiente C++.

Para reduzir a ambiguidade de alguns resultados apresentados, pode-se estudar a implementação de várias redes neurais em série com enfoques distintos e treinamentos específicos.

12 REFERÊNCIAS

ANÔNIMO. Bridgina Naturopathic - Facial SculptingNon-Surgical Face Lift. **Bridgina Naturopathic - Home**. Disponivel em: <http://bridgina.com/facial_sculpting>. Acesso em: 01 maio 2010.

BRADSKY, G.; KAEHLER, A. **Learning OpenCV**. [S.l.]: O'REILLY.

CAI, J.; GOSHTASBY, A. A. Detecting human faces in color images. **Image Vision Comput.**, 1999.

CHEN, Q.; WU, H.; YACHIDA, M. Face detection by fuzzy pattern matching. **Proceedings of the Fifth International Conference on Computer Vision**, 1995.

DEMUTH, H.; BEALE, M.; HAGAN, M. **Neural Network Toolbox™ User's Guide**. Natick: The MathWorks, Inc, 2008.

EKMAN, P.; FRIESEN, W. V.; HAGER, J. C. **Facial Action Coding System - The Manual**. Salt Lake City: Research Nexus, 2002.

EKMAN, P.; FRIESSEN, W. V.; HAGER, J. C. **Facial Action Coding System - Investigator's Guide**. 2ª Edição. ed. Salt Lake City: Research Nexus, 2002.

EKMAN, P.; MATSUMOTO, D.; FRIESSEN, W. V. Facil Expressions in Affective Disorders. In: EKMAN, P.; ROSENBERG, E. L. **What the Face Reveals - Basic and Applied Studies of Spontaneous Expressions Using Facial Action Coding System**. [S.l.]: [s.n.], 1982. Cap. 15, p. 331-342.

HJELMAS, E.; LOW, B. K. Face Detection: A survey. **Computer Vision and Image Understanding**, 2001.

JENG, S.-H. et al. Facial feature detection using geometrical face model: An efficient approach. **Pattern Recognition**, 1998.

KAH, S. K.; TOMASO, P. Example-based learning for view vased human face detection. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, 1998.

KOBAYASHI H., H. F. The Recognition of Basic Expressions by Neural Network. **Proceedings of International Joint Conference on Neural Network**, p. 460-466, 1991.

MASE, K. Recognition of Facial Expression from Optical Flow. **IEICE Trans**, v. E74, n. 10, p. 3474-3483, Outubro 1991.

MOORE, K. L.; DALLEY, A. F. **Anatomia orientada para clínica**. 5. ed. Rio de Janeiro: Guanabara Koogan, 2007.

PAKNIKAR, G. **Facial Image Based Expressions Classification System Using Comitee Neural Networks**. Akron: [s.n.], 2008.

PAPAGEORGIOU, C.; OREN, M.; POGGIO, T. **A genereal framework for object detection**. International Conference on Computer Vision. [S.l.]: [s.n.]. 1998.

PUTZ, R.; PABST, R. **Sobotta - Atlas de Anatomia Humana**. Tradução de Renate Putz. 22. ed. Rio de Janeiro: Editora Guanabara koogan S.A., v. 1, 2006. 76 p.

ROWLEY, H.; SHUMEET, B.; TAKEO, K. Neural network based face detection. **Computer Vision and Pattern Recognition**, 1996.

SAYETTE, M. A. et al. A Psychometric Evaluation of the Facial Action Coding System for Assessing Spontaneous Expression. **Journal of Nonverbal Behavior**, n. 25, p. 167-186, 2001.

SUWA, M.; SUGIE, N.; FUJIMORA, K. A Preliminary Note on Pattern Recognition of Human Emotional Expression. **Proc. Int'l Joint Conf. Pattern Recognition**, 1978. 408-410.

T. KANADE, J. F. C. Y. T. **Comprehensive Database for Facial Expression Analisys**. 4º IEEE International Conference on Automatic Face and Gesture Recognition. Grenoble: [s.n.]. 2000. p. 46-53.

VIEIRA, R. C.; ROISENBERG, M. Redes Neurais Artificiais: um breve tutorial. **Laboratório de Conexionismo e Ciências Cognitivas (L3C)**, Florianópolis.

VIOLA, P.; JONES, M. Rapid Object Detection using a Boosted Cascade of Simple Features, Vancouver, 2001.

WANG, J.; TAN, T. A new face detection method based on shape information. **Pattern Recognition Letter**, 2000.

YACOOB, Y.; DAVIS, L. S. Recognizing Human Facial Expression from Long Image Sequences Using Optical Flow. **IEEE Trans Pattern Analysis and Machine Inteligence**, v. 18, n. 6, p. 636-642, Junho 1996.

ZHANG, Z. Feature-base dacial expression recognition. **International Journal of Pattern Recognition and Artificial Intelligence**, p. 893-911, 1999.

13 ANEXO I – PERMISSÃO PARA UTILIZAR O DATABASE COHN-KANADE(T. KANADE, 2000)

AGREEMENT ON USE OF IMAGE DATA
Cohn-Kanade Facial Expression Database

I agree

- to cite Kanade, Cohn, &Tian (2000) in any paper of mine or my collaborators that makes any use of the database. The reference is:
  Kanade, T., Cohn, J. F., & Tian, Y. (2000). Comprehensive database for facial expression analysis. *Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition (FG'00)*, Grenoble, France, 46-53.
- to use the images for research purposes only.
- not to provide the images to second parties.
- if I reproduce images in electronic or print media, to use only those from the following subjects and include notice of copyright (©Jeffrey Cohn).

S52	S55	S74	S106	S111	S113
S121	S124	S125	S130	S132	

Signature: 

Name: MARCOS PEREIRA BORETO

Title: PROFESSOR

Institution: UNIVERSITY of SAO PAULO

Date: 14/SEP/2010

Email address: marcos.boreto@poli.usp.br

14 ANEXO II – PROGRAMA FINAL OPENCV

```
#include "cv.h"
#include "highgui.h"
#include "math.h"
#include <cstdlib>
#include <iostream>
#include <string>
#include <stdio.h>
#include <fstream>
#include <sstream>
#include <cstring>

using namespace std;

#define MAX_PARAM 19
#define CVX_RED         CV_RGB(0xff,0x00,0x00)
#define CVX_GREEN CV_RGB(0x00,0xff,0x00)
#define CVX_BLUE  CV_RGB(0x00,0x00,0xff)

int X[MAX_PARAM];
int Y[MAX_PARAM];
int J = 0;
int K = 2;

String cascadeName =
"../../data/haarcascades/boca.xml";
String nestedCascadeName =
"../../data/haarcascades/boca.xml";

IplImage*      g_image     = NULL;
IplImage*      g_gray      = NULL;
IplImage*      g_gray2     = NULL;
IplImage*      src         = NULL;
int            g_thresh    = 150;
CvMemStorage*  g_storage   = NULL;


 /* função que ativa o trackbar para escolha e aplicação do threshold
*/
 void on_trackbar(int) {
  if( g_storage==NULL ) {

    g_gray = cvCreateImage( cvGetSize(g_image), 8, 1 );
g_gray2 = cvCreateImage( cvSize(640,480), 8, 1 );
    g_storage = cvCreateMemStorage(0);
  } else {
    cvClearMemStorage( g_storage );
  }

  CvSeq* contours = 0;

  cvCvtColor( g_image, g_gray, CV_BGR2GRAY );

  cvThreshold( g_gray, g_gray, g_thresh, 255, CV_THRESH_BINARY );
  cvFindContours( g_gray, g_storage, &contours );
  cvZero( g_gray );
  cvResize(g_gray,g_gray2,CV_INTER_CUBIC);
```

```
        if( contours )
          cvDrawContours(
            g_gray2,
            contours,
            cvScalarAll(255),
            cvScalarAll(255),
            100
          );
        cvShowImage( "Contours", g_gray2 );
      }

       /* função para determinação do ponto máximo em um vetor */
       int maximumValue( int vetor[])
      {
       int length = 100000;  // tamanho máximo de um contorno
           int max = vetor[0];       // max recebe o primeiro elemento
       int pos = 0;

           for(int i = 1; i<length; i++)
           {
             if(vetor[i] > max){
                   max = vetor[i];
                   pos=i;
             }
           }
           return pos;                    // retorna a posição no vetor do
ponto maximo
      }

        /* função para determinação do ponto mínimo em um vetor */
        int minimumValue( int vetor[])
      {
       int length = 100000;
           int min = vetor[0];
       int pos = 0;

           for(int i = 1; i<length; i++)
           {
            if(vetor[i] < min && vetor[i]!=-1){
                  min = vetor[i];
                  pos = i;
             }
           }
           return pos;                    // retorna a posição no vetor do
ponto minimo
      }

        /* função que procura o valor mais proximo do pedido no vetorx, e
retorna a posição que possua o maior valor no vetory */
        int findMax( int vetorx[], int vetory[], int valor)
        {
        int length = 100000;
        int max = 0;
        int pos = -1;
        for( int i=0; i<length;i++){
              /*  valor  informado  dentro  do  intervalo  determinado  por
vetorx[i-1] e vetorx[i], e vetory[i] é maximo */
              if  (((vetorx[i-1]<valor  &&  vetorx[i]>=valor)  ||  (vetorx[i-
1]>valor && vetorx[i]<=valor)) && vetory[i]>max){
```

```
                        max=vetory[i];
                        pos=i;
                }
        }
        return pos;     // retorna posição do vetor em que o desejado
ocorre
        }

        /* função que procura o valor mais proximo do pedido no vetorx, e
retorna a posição que possua o menor valor no vetory */
        int findMin( int vetorx[], int vetory[], int valor)
        {
        int length = 100000;
        int min = 10000;
        int pos = -1;
        for( int i=0; i<length;i++){
                /* valor informado dentro do intervalo determinado por
vetorx[i-1] e vetorx[i], e vetory[i] é minimo */
                if (((vetorx[i-1]<valor && vetorx[i]>=valor) || (vetorx[i-
1]>valor && vetorx[i]<=valor)) && vetory[i]<min && vetory[i]!=-1){
                        min=vetory[i];
                        pos=i;
                }
        }
        return pos;     // retorna posição do vetor em que o desejado
ocorre
        }

        /* função que ativa o funcionamento do mouse sobre a imagem e
guarda as posições quando o botào direito é clicado */
        void mouseHandler(int event, int x, int y, int flags, void* param)
      {
          IplImage* img1;
          CvFont    font;
          //uchar*    ptr;
          char      label[60];

          img1 = (IplImage*) param;
         // img1 = cvCloneImage(img0);

          cvInitFont(&font, CV_FONT_HERSHEY_PLAIN, .8, .8, 0, 1, 8);
          // pede para clicar nos 18 pontos
          if (event == CV_EVENT_LBUTTONDOWN && J < MAX_PARAM)
          {
            switch (J) {
                  case 0:
                        sprintf(label,"Clique    no    ponto    esquero    da
boca");break;
                  case 1:
                        sprintf(label,"Clique    no    ponto    direito    da
boca");break;
                  case 2:
                        sprintf(label,   "Clique   no   ponto   superior   da
boca");break;
                  case 3:
                        sprintf(label,"Clique    no    ponto    inferior    da
boca");break;
                  case 4:
                        sprintf(label,"Clique  no  ponto  esquero  do  olho
esquerdo");break;
```

```
                    case 5:
                          sprintf(label,"Clique no ponto direito do olho
esquerdo");break;
                    case 6:
                          sprintf(label,"Clique no ponto superior do olho
esquero");break;
                    case 7:
                          sprintf(label,"Clique no ponto inferior do olho
esquerdo");break;
                    case 8:
                          sprintf(label,"Clique no ponto esquero do olho
direito");break;
                    case 9:
                          sprintf(label,"Clique no ponto direito do olho
direito");break;
                    case 10:
                          sprintf(label,"Clique no ponto superior do olho
direito");break;
                    case 11:
                          sprintf(label,"Clique no ponto inferior do olho
direito");break;
                    case 12:
                          sprintf(label,"Clique no ponto esquero da
sombrancelha esquerdo");break;
                    case 13:
                          sprintf(label,"Clique no ponto direito da
sombrancelha esquerdo");break;
                    case 14:
                          sprintf(label,"Clique no ponto superior da
sombrancelha esquero");break;
                    case 15:
                          sprintf(label,"Clique no ponto esquero do da
sombrancelha direito");break;
                    case 16:
                          sprintf(label,"Clique no ponto direito da
sombrancelha direito");break;
                    case 17:
                          sprintf(label,"Clique no ponto superior da
sombrancelha direito");break;
                    /* case 18:
                          sprintf(label,"Clique na orelha esquerda");break;
                    case 19:
                          sprintf(label,"Clique na orelha direita");break; */
                    default:
                          sprintf(label,"Aperte ENTER para sair");break;
              }
       // desenha um retangulo para escrever o texto
            cvRectangle(img1,cvPoint(0,0),cvPoint(500,15),CV_RGB(255,      0,
0),CV_FILLED,8, 0);
           cvPutText(img1,label,cvPoint(0, 12),&font,CV_RGB(255, 255, 0));
            /* read pixel */
              //ptr = cvPtr2D(img1, y, x, NULL);

              /*
               * display the BGR value
               */
       //cvShowImage("img", img1);
            // guarda e imprimi as coordenadas
              X[J]=x;
            Y[J]=y;
```

```
            printf("J %d, X=%d Y=%d \n",J, X[J],Y[J]);

            // desenha um retangulo vermelho pequeno sobre o ponto clicado
              cvRectangle(
                  img1,
                  cvPoint(x-1, y-1),
                  cvPoint(x+1, y+1),
                  CV_RGB(255, 0, 0),
                  CV_FILLED,
                  8, 0
              );

               // printf("\n\n a\n\n");
              cvShowImage("img", img1);
            J++;

        }
        }

          /* crop de 1 elipse */
           void CircleROI(int x, int y, int largura, int altura)
         {
             IplImage* res, * roi;

            /* usage: <prog_name> <image> */

            res = cvCreateImage(cvGetSize(src), 8, 3);
            roi = cvCreateImage(cvGetSize(src), 8, 1);

            /* prepare the 'ROI' image */
            cvZero(roi);

            /* Note that you can use any shape for the ROI
            cvCircle(
                roi,
                cvPoint(x, y),
                raio,
                CV_RGB(255, 255, 255),
                -1, 8, 0
            );
            CvBox2D box;
        box.center.x = 50;
        box.center.y =40;
        box.size.height = 20;
        box.size.width = 30;
            cvEllipseBox(roi,box,CV_RGB(255,255,255),-1,8,0);*/
        cvEllipse(roi,cvPoint(x,y),cvSize(largura,altura),0,0,360,CV_RGB(255,
255,255),-1,8,0);

            /* extract subimage */
            cvAnd(src, src, res, roi);


            /* 'restore' subimage */
            IplImage* roi_C3 = cvCreateImage(cvGetSize(src), 8, 3);
            cvMerge(roi, roi, roi, NULL, roi_C3);
            cvAnd(res, roi_C3, res, NULL);
        cvSaveImage("img2.jpg",res);

         }
```

```
         /* crop de 2 elipses */
         void CircleROI2(int x1, int y1, int largura1, int altura1,int x2,
int y2, int largura2, int altura2)
       {
           IplImage* res, * roi;

         /* usage: <prog_name> <image> */



         res = cvCreateImage(cvGetSize(src), 8, 3);
         roi = cvCreateImage(cvGetSize(src), 8, 1);

         /* prepare the 'ROI' image */
         cvZero(roi);

         /* Note that you can use any shape for the ROI
         cvCircle(
             roi,
             cvPoint(x, y),
             raio,
             CV_RGB(255, 255, 255),
             -1, 8, 0
         );
         CvBox2D box;
      box.center.x = 50;
      box.center.y =40;
      box.size.height = 20;
      box.size.width = 30;
         cvEllipseBox(roi,box,CV_RGB(255,255,255),-1,8,0);*/
      cvEllipse(roi,cvPoint(x1,y1),cvSize(largura1,altura1),0,0,360,CV_RGB(
255,255,255),-1,8,0);

cvEllipse(roi,cvPoint(x2,y2),cvSize(largura2,altura2),0,0,360,CV_RGB(255,25
5,255),-1,8,0);
         /* extract subimage */
         cvAnd(src, src, res, roi);


         /* 'restore' subimage */
         IplImage* roi_C3 = cvCreateImage(cvGetSize(src), 8, 3);
         cvMerge(roi, roi, roi, NULL, roi_C3);
         cvAnd(res, roi_C3, res, NULL);
      cvSaveImage("img2.jpg",res);

       }

        /* detecta de acordo com os cascades globais e guarda as posições
no vetor pos */
         void detect( int argc, const char** argv, int** pos )
      {
         CvCapture* capture = 0;
         Mat frame, frameCopy, image;
         const String scaleOpt = "--scale=";
         size_t scaleOptLen = scaleOpt.length();
         const String cascadeOpt = "--cascade=";
         size_t cascadeOptLen = cascadeOpt.length();
         const String nestedCascadeOpt = "--nested-cascade";
         size_t nestedCascadeOptLen = nestedCascadeOpt.length();
```

```
        String inputName;

        CascadeClassifier cascade, nestedCascade;
        double scale = 1;

        for( int i = 1; i < argc; i++ )
        {
            if(    cascadeOpt.compare(    0,    cascadeOptLen,    argv[i],
cascadeOptLen ) == 0 )
                cascadeName.assign( argv[i] + cascadeOptLen );
            else  if(  nestedCascadeOpt.compare(  0,  nestedCascadeOptLen,
argv[i], nestedCascadeOptLen ) == 0 )
            {
                if( argv[i][nestedCascadeOpt.length()] == '=' )
                    nestedCascadeName.assign(          argv[i]          +
nestedCascadeOpt.length() + 1 );
                if( !nestedCascade.load( nestedCascadeName ) )
                    cerr << "WARNING: Could not load classifier cascade
for nested objects" << endl;
            }
            else   if(   scaleOpt.compare(   0,   scaleOptLen,   argv[i],
scaleOptLen ) == 0 )
            {
                if( !sscanf( argv[i] + scaleOpt.length(), "%lf", &scale )
|| scale < 1 )
                    scale = 1;
            }
            else if( argv[i][0] == '-' )
            {
                cerr << "WARNING: Unknown option %s" << argv[i] << endl;
            }
            else
                inputName.assign( argv[i] );
        }

        if( !cascade.load( cascadeName ) )
        {
            cerr << "ERROR: Could not load classifier cascade" << endl;
            cerr << "Usage: facedetect [--cascade=\"<cascade_path>\"]\n"
                "   [--nested-cascade[=\"nested_cascade_path\"]]\n"
                "   [--scale[=<image scale>\n"
                "   [filename|camera_index]\n" ;
            return -1;
        }

        if(   inputName.empty()   ||   (isdigit(inputName.c_str()[0])   &&
inputName.c_str()[1] == '\0') )
            capture   =   cvCaptureFromCAM(   inputName.empty()   ?   0   :
inputName.c_str()[0] - '0' );
        else if( inputName.size() )
        {
            image = imread( inputName, 1 );
            if( image.empty() )
                capture = cvCaptureFromAVI( inputName.c_str() );
        }
        else
            image = imread( "lena.jpg", 1 );

        cvNamedWindow( "result", 1 );
```

```
        if( capture )
        {
            for(;;)
            {
                IplImage* iplImg = cvQueryFrame( capture );
                frame = iplImg;
                if( frame.empty() )
                    break;
                if( iplImg->origin == IPL_ORIGIN_TL )
                    frame.copyTo( frameCopy );
                else
                    flip( frame, frameCopy, 0 );

                detectAndDraw( frameCopy, cascade, nestedCascade, scale );

                if( waitKey( 10 ) >= 0 )
                    goto _cleanup_;
            }

            waitKey(0);
    _cleanup_:
            cvReleaseCapture( &capture );
        }
        else
        {
            if( !image.empty() )
            {
                detectAndDraw( image, cascade, nestedCascade, scale );
                waitKey(0);
            }
            else if( !inputName.empty() )
            {
                /* assume it is a text file containing the
                list of the image filenames to be processed - one per
line */
                FILE* f = fopen( inputName.c_str(), "rt" );
                if( f )
                {
                    char buf[1000+1];
                    while( fgets( buf, 1000, f ) )
                    {
                        int len = (int)strlen(buf), c;
                        while( len > 0 && isspace(buf[len-1]) )
                            len--;
                        buf[len] = '\0';
                        cout << "file " << buf << endl;
                        image = imread( buf, 1 );
                        if( !image.empty() )
                        {
                            detectAndDraw( image, cascade, nestedCascade,
scale );
                            c = waitKey(0);
                            if( c == 27 || c == 'q' || c == 'Q' )
                                break;
                        }
                    }
                    fclose(f);
                }
            }
        }
```

```
            cvDestroyWindow("result");
        }

        void detectAndDraw( Mat& img,
                            CascadeClassifier&   cascade,   CascadeClassifier&
nestedCascade,
                            double scale)
        {
            int i = 0;
            double t = 0;
            vector<Rect> faces;
            const static Scalar colors[] =  { CV_RGB(0,0,255),
                CV_RGB(0,128,255),
                CV_RGB(0,255,255),
                CV_RGB(0,255,0),
                CV_RGB(255,128,0),
                CV_RGB(255,255,0),
                CV_RGB(255,0,0),
                CV_RGB(255,0,255)} ;
            Mat      gray,      smallImg(      cvRound      (img.rows/scale),
cvRound(img.cols/scale), CV_8UC1 );

            cvtColor( img, gray, CV_BGR2GRAY );
            resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
            equalizeHist( smallImg, smallImg );

            t = (double)cvGetTickCount();
            cascade.detectMultiScale( smallImg, faces,
                1.1, 2, 0
                //|CV_HAAR_FIND_BIGGEST_OBJECT
                //|CV_HAAR_DO_ROUGH_SEARCH
                |CV_HAAR_SCALE_IMAGE
                ,
                Size(30, 30) );
            t = (double)cvGetTickCount() - t;
            printf(       "detection       time       =       %g       ms\n",
t/((double)cvGetTickFrequency()*1000.) );
            for(  vector<Rect>::const_iterator  r  =  faces.begin();  r  !=
faces.end(); r++, i++ )
            {
                Mat smallImgROI;
                vector<Rect> nestedObjects;
                Point center;
              Point up;
              Point down;
              up.x=150;
              up.y=130;
              down.x=220;
              down.y=180;
                Scalar color = colors[i%8];
                int radius;
                center.x = cvRound((r->x + r->width*0.5)*scale);
                center.y = cvRound((r->y + r->height*0.5)*scale);
                radius = cvRound((r->width + r->height)*0.25*scale);
              printf("x=%d\n y=%d\n r=%d\n",center.x ,center.y,radius);
              //circle( img, center, radius, color, 3, 8, 0 );
              rectangle(img, up, down, color, 1, 8, 0);
                if( nestedCascade.empty() )
                    continue;
```

```
                smallImgROI = smallImg(*r);
                nestedCascade.detectMultiScale( smallImgROI, nestedObjects,
                    1.1, 2, 0
                    //|CV_HAAR_FIND_BIGGEST_OBJECT
                    //|CV_HAAR_DO_ROUGH_SEARCH
                    //|CV_HAAR_DO_CANNY_PRUNING
                    |CV_HAAR_SCALE_IMAGE
                    ,
                    Size(30, 30) );
          /*/     for( vector<Rect>::const_iterator nr = nestedObjects.begin();
nr != nestedObjects.end(); nr++ )
                {
                    center.x = cvRound((r->x + nr->x + nr->width*0.5)*scale);
                    center.y = cvRound((r->y + nr->y + nr->height*0.5)*scale);
                    radius = cvRound((nr->width + nr->height)*0.25*scale);
                    circle( img, center, radius, color, 3, 8, 0 );
                }
               */ }
            // cv::imshow( "result", img );
        }

           int main(int argc, char** argv)
        {
            IplImage* imgneutra;
            IplImage* img_8uc1 = NULL;
        float M[2][MAX_PARAM];
        float N[2][MAX_PARAM];
        float R[MAX_PARAM];
        float m1;
        float m2;
        float n1;
        float n2;
        float xm;
        float xn;
        float ym;
        float yn;
        float r;
        M[0][0]=0;
        N[0][0]=0;
        M[1][0]=0;
        N[1][0]=0;
        const char nome[100] = "Sair";
        int flag = 0;
        fstream filestr;
        int emocao = 0;
        int Maior[2][100000];
        int Segunda[2][100000];
        int vetorx[100000];
        int vetory[100000];
        int maxelem=0;
        int pos;
        int cont;
        int meio;
            int posboca[4];
        int posolho1[4];
        int posolho2[4];
        int posobra1[4];
        int posobra2[4];
```

```
      /* comentários para a parte de definição dos pontos desejados foram
feitos somente para a parte da boca,
         porém o programa apenas repete a mesma lógica para as outras
regiões */
          printf("\n-------------------------------------------------------------
------------\n Insira a imagem neutra ou digite ''Sair'' \n --------------
-------------------------------------------------------------------------
\n");
      scanf("%s",nome);
      if (nome == "Sair"){
            printf("Saiu!");
            return 0;
            }
      else{
          /* load imagem */
          imgneutra= cvLoadImage(nome, 1);


          /* checa para ver se imagem existe */
          assert(imgneutra);

          cvNamedWindow("img", 1);
      J=0;
      /* chama função de seleção manual dos 18 pontos de interesse */
          cvSetMouseCallback("img", mouseHandler, (void*)imgneutra);

          cvWaitKey(0);

          cvDestroyAllWindows();
          cvReleaseImage(&imgneutra);
      /* Grava as posições da foto neutra na matriz M. (1 linha = x, 2
linha = y) */
      for(int i = 1; i<MAX_PARAM; i++){ //ignorar primeira posicao
            M[0][i]=float(X[i]);
            M[1][i]=float(Y[i]);
            printf("IMAGE NEUTRA: {X,Y(%d)=(%f; %f)\n",i,M[0][i],M[1][i]);
      }
       while(true){
      printf("\n-------------------------------------------------------------
---------\n Insira a imagem emotiva ou digite ''Sair'' \n -----------------
-------------------------------------------------------------------------\n");
      scanf("%s",nome);
      if (nome == "Sair"){
            printf("Saiu!");
            return 0;
            }
      else{

      for(int i=0;i<100000;i++){
        Maior[0][i]=Maior[1][i]=Segunda[0][i]=Segunda[1][i]=-1;
        }

        // BOCA

        /* localiza e guarda a posição da boca */
        detect(1, src, &posboca);

        IplImage* img1 = cvLoadImage(nome);
        cvNamedWindow(" tcc", CV_WINDOW_AUTOSIZE );
        cvShowImage(" tcc", img1 );
```

```
        src = img1;

        /* crop de uma elipse que circunda a boca */
        CircleROI(posboca[0],posboca[1],posboca[2],posboca[3]);

        g_image=cvLoadImage("img2.jpg");
        //cvSaveImage("img2.jpg",img2);
        cvResetImageROI(img1);
        cvNamedWindow("Contours", CV_WINDOW_AUTOSIZE );
        cvCreateTrackbar("Threshold","Contours",&g_thresh,255,on_trackbar);

        /* ativa o trackbar para definição do threshold desejado */
        on_trackbar(0);

        cvWaitKey();

        img_8uc1 = cvLoadImage( "img2.jpg", CV_LOAD_IMAGE_GRAYSCALE);
        CvSeq* maior;
        CvSeq* c;
        maxelem=0;
        maior = NULL;
        IplImage* img_edge = cvCreateImage( cvGetSize(img_8uc1), 8, 1 );
        IplImage* img_8uc3 = cvCreateImage( cvGetSize(img_8uc1), 8, 3 );
        /* aplica o threshold */
        cvThreshold( img_8uc1, img_edge, g_thresh, 255, CV_THRESH_BINARY );
        CvMemStorage* storage = cvCreateMemStorage();
        CvSeq* first_contour = NULL;
        /* localiza os contours da imagem já com o threshold definido e
aplicado */
        int Nc = cvFindContours(img_edge, storage, &first_contour );
        int n=0,k;
        //  printf("\n\nHit  any  key  to  draw  the  next  contour,  ESC  to
quit\n\n");
        // printf( "Total Contours Detected: %d\n", Nc );
        for( c=first_contour; c!=NULL; c=c->h_next ){
        //printf("\n a \n");

           /* passa por todos os contours e grava os dois maiores que sejam
in (=hole=azul) */
       for( int i=0; i<c->total; ++i ) {
            CvPoint* p = CV_GET_SEQ_ELEM( CvPoint, c, i );
            //printf("\n b \n");
            // printf("    (%d,%d)\n", p->x, p->y );
            if(c->total>=maxelem ){
                  if(c->flags==1117360652){
                  maxelem=c->total;
                  cont=n;
                  Segunda[0][i]=Maior[0][i];
                  Segunda[1][i]=Maior[1][i];
                  Maior[0][i]=p->x;
                  Maior[1][i]=p->y;
                  maior = c;
                  // printf("\n c \n");
                  }
            }
       }
            //if((k = cvWaitKey()&0x7F) == 27)
            //break;
           n++;
```

```
        }

           /* desenha uma linha azul em volta do contour selecionado =
maior */
           cvCvtColor( img_8uc1, img_8uc3, CV_GRAY2BGR );
           cvDrawContours(
              src,
              maior,
              CVX_RED,
              CVX_BLUE,
              0,
              2,
              8
           );
           printf("Contour #%d flag=%d\n", n,maior->flags );
        cvShowImage( argv[0], src );
        cvWaitKey();
           printf(" %d elements:\n", maior->total );

        // printf("Finished all contours. Hit key to finish\n"); */

        /* determina os 4 pontos desejados em volta da boca */
        for(int i=0; i<100000; i++){
        vetorx[i]=Maior[0][i];
        vetory[i]=Maior[1][i];
        }

        // ponto esquerdo da boca
        pos=minimumValue(vetorx);
        N[0][1]=Maior[0][pos];
        N[1][1]=Maior[1][pos];

        // ponto direito da boca
        pos=maximumValue(vetorx);
        N[0][2]=Maior[0][pos];
        N[1][2]=Maior[1][pos];

        // ponto superior da boca
        meio = (N[0][1]+N[0][2])/2;
        pos=findMin(vetorx, vetory, meio);
        N[0][3]=Maior[0][pos];
        N[1][3]=Maior[1][pos];

        // ponto superior da boca
        pos=findMax(vetorx, vetory, meio);
        N[0][4]=Maior[0][pos];
        N[1][4]=Maior[1][pos];

        cvCvtColor( img_8uc1, img_8uc3, CV_GRAY2BGR );
        cvShowImage( argv[0], img_8uc3 );
        cvDestroyWindow( argv[0] );
        cvReleaseImage( &img_8uc1 );
        cvReleaseImage( &img_8uc3 );
        cvReleaseImage( &img_edge );

        g_storage = NULL;

        //OLHOS

        String cascadeName =
```

```
        "../../data/haarcascades/olhoesquerdo.xml";
        String nestedCascadeName =
        "../../data/haarcascades/olhoesquerdo.xml";

          detect(1, src, &posolho1);

          String cascadeName =
        "../../data/haarcascades/olhodireito.xml";
        String nestedCascadeName =
        "../../data/haarcascades/olhodireito.xml";

          detect(1, src, &posolho2);

          for(int i=0;i<100000;i++){
          Maior[0][i]=Maior[1][i]=Segunda[0][i]=Segunda[1][i]=-1;
          }

          CircleROI2(posolho1[0],
posolho1[1],posolho1[2],posolho1[3],posolho2[0],posolho2[1],posolho2[2],pos
olho2[3]);
          g_image=cvLoadImage("img2.jpg");

          cvNamedWindow("Contours", CV_WINDOW_AUTOSIZE );

          cvCreateTrackbar("Threshold","Contours",&g_thresh,255,on_trackbar);

          on_trackbar(0);

          cvWaitKey();
          CircleROI(posolho1[0],posolho1[1],posolho1[2],posolho1[3]);
          img_8uc1 = cvLoadImage( "img2.jpg", CV_LOAD_IMAGE_GRAYSCALE);

          img_edge = cvCreateImage( cvGetSize(img_8uc1), 8, 1 );
          img_8uc3 = cvCreateImage( cvGetSize(img_8uc1), 8, 3 );
          cvThreshold( img_8uc1, img_edge, g_thresh, 255, CV_THRESH_BINARY );
          storage = cvCreateMemStorage();
          first_contour = NULL;
          Nc = cvFindContours(img_edge, storage, &first_contour );
          n=0;
          maxelem=0;
          maior = NULL;
          //  printf("\n\nHit  any  key  to  draw  the  next  contour,  ESC  to
quit\n\n");
          // printf( "Total Contours Detected: %d\n", Nc );
         for( c=first_contour; c!=NULL; c=c->h_next ){
          //printf("\n a \n");

         for( int i=0; i<c->total; ++i ) {
              CvPoint* p = CV_GET_SEQ_ELEM( CvPoint, c, i );
               //printf("\n b \n");
              // printf("    (%d,%d)\n", p->x, p->y );
              if(c->total>=maxelem ){
                    if(c->flags==1117360652){
                    maxelem=c->total;
                    cont=n;
                    Segunda[0][i]=Maior[0][i];
                    Segunda[1][i]=Maior[1][i];
                    Maior[0][i]=p->x;
                    Maior[1][i]=p->y;
                    maior = c;
```

```
            // printf("\n c \n");
            }
     }
}
     //if((k = cvWaitKey()&0x7F) == 27)
     //break;
    n++;

 }

    cvCvtColor( img_8uc1, img_8uc3, CV_GRAY2BGR );
    cvDrawContours(
       src,
       maior,
       CVX_RED,
       CVX_BLUE,
       0,
       2,
       8
    );
    printf("Contour #%d flag=%d\n", n,maior->flags );
cvShowImage( argv[0], src );
cvWaitKey();
    printf(" %d elements:\n", maior->total );

 for(int i=0; i<100000; i++){
 vetorx[i]=Maior[0][i];
 vetory[i]=Maior[1][i];
 }

 // ponto esquerdo do olho esquerdo
 pos=minimumValue(vetorx);
 N[0][5]=Maior[0][pos];
 N[1][5]=Maior[1][pos];

 // ponto direito do olho esquerdo
 pos=maximumValue(vetorx);
 N[0][6]=Maior[0][pos];
 N[1][6]=Maior[1][pos];

 // ponto superior do olho esquerdo
 meio = (N[0][5]+N[0][6])/2;
 pos=findMin(vetorx, vetory, meio);
 N[0][7]=Maior[0][pos];
 N[1][7]=Maior[1][pos];

 // ponto superior do olho esquerdo
 pos=findMax(vetorx, vetory, meio);
 N[0][8]=Maior[0][pos];
 N[1][8]=Maior[1][pos];

 cvCvtColor( img_8uc1, img_8uc3, CV_GRAY2BGR );
 cvShowImage( argv[0], img_8uc3 );
 // cvWaitKey(0);
 cvDestroyWindow( argv[0] );
 cvReleaseImage( &img_8uc1 );
 cvReleaseImage( &img_8uc3 );
 cvReleaseImage( &img_edge );

 g_storage = NULL;
```

```
         for(int i=0;i<100000;i++){
         Maior[0][i]=Maior[1][i]=Segunda[0][i]=Segunda[1][i]=-1;
         }

         CircleROI(posolho2[0],posolho2[1],posolho2[2],posolho2[3]);
         img_8uc1 = cvLoadImage( "img2.jpg", CV_LOAD_IMAGE_GRAYSCALE);

         img_edge = cvCreateImage( cvGetSize(img_8uc1), 8, 1 );
         img_8uc3 = cvCreateImage( cvGetSize(img_8uc1), 8, 3 );
         cvThreshold( img_8uc1, img_edge, g_thresh, 255, CV_THRESH_BINARY );
         storage = cvCreateMemStorage();
         first_contour = NULL;
         Nc = cvFindContours(img_edge, storage, &first_contour );
         n=0;
         maxelem=0;
         maior = NULL;
         //  printf("\n\nHit  any  key  to  draw  the  next  contour,  ESC  to
quit\n\n");
         // printf( "Total Contours Detected: %d\n", Nc );
        for( c=first_contour; c!=NULL; c=c->h_next ){
         //printf("\n a \n");

        for( int i=0; i<c->total; ++i ) {
             CvPoint* p = CV_GET_SEQ_ELEM( CvPoint, c, i );
              //printf("\n b \n");
             // printf("    (%d,%d)\n", p->x, p->y );
             if(c->total>=maxelem ){
                   if(c->flags==1117360652){
                   maxelem=c->total;
                   cont=n;
                   Segunda[0][i]=Maior[0][i];
                   Segunda[1][i]=Maior[1][i];
                   Maior[0][i]=p->x;
                   Maior[1][i]=p->y;
                   maior = c;
                   // printf("\n c \n");
                   }
              }
        }
             //if((k = cvWaitKey()&0x7F) == 27)
             //break;
            n++;

         }

            cvCvtColor( img_8uc1, img_8uc3, CV_GRAY2BGR );
            cvDrawContours(
               src,
               maior,
               CVX_RED,
               CVX_BLUE,
               0,
               2,
               8
            );
            printf("Contour #%d flag=%d\n", n,maior->flags );
        cvShowImage( argv[0], src );
        cvWaitKey();
            printf(" %d elements:\n", maior->total );
```

```
    for(int i=0; i<100000; i++){
    vetorx[i]=Maior[0][i];
    vetory[i]=Maior[1][i];
    }

    // ponto esquerdo do olho direito
    pos=minimumValue(vetorx);
    N[0][9]=Maior[0][pos];
    N[1][9]=Maior[1][pos];

    // ponto direito do olho direito
    pos=maximumValue(vetorx);
    N[0][10]=Maior[0][pos];
    N[1][10]=Maior[1][pos];

    // ponto superior do olho direito
    meio = (N[0][9]+N[0][10])/2;
    pos=findMin(vetorx, vetory, meio);
    N[0][11]=Maior[0][pos];
    N[1][11]=Maior[1][pos];

    // ponto superior do olho direito
    pos=findMax(vetorx, vetory, meio);
    N[0][12]=Maior[0][pos];
    N[1][12]=Maior[1][pos];

    cvCvtColor( img_8uc1, img_8uc3, CV_GRAY2BGR );
    cvShowImage( argv[0], img_8uc3 );
    // cvWaitKey(0);
    cvDestroyWindow( argv[0] );
    cvReleaseImage( &img_8uc1 );
    cvReleaseImage( &img_8uc3 );
    cvReleaseImage( &img_edge );

    g_storage = NULL;

    // SOBRANCELHA

    String cascadeName =
"../../data/haarcascades/sobrancelhaesquerda.xml";
String nestedCascadeName =
"../../data/haarcascades/sobrancelhaesquerda.xml";

    detect(1, src, &posobra1);

      String cascadeName =
"../../data/haarcascades/sobrancelhadireita.xml";
String nestedCascadeName =
"../../data/haarcascades/sobrancelhadireita.xml";

    detect(1, src, &posobra2);

    for(int i=0;i<100000;i++){
    Maior[0][i]=Maior[1][i]=Segunda[0][i]=Segunda[1][i]=-1;
    }


CircleROI2(posobra1[0],posobra1[1],posobra1[2],posobra1[3],posobra2[0],poso
bra2[1],posobra2[2],posobra2[3]);
```

```
        g_image=cvLoadImage("img2.jpg");

        cvNamedWindow("Contours", CV_WINDOW_AUTOSIZE );

        cvCreateTrackbar("Threshold","Contours",&g_thresh,255,on_trackbar);

        on_trackbar(0);

        cvWaitKey();
        CircleROI(posobra1[0],posobra1[1],posobra1[2],posobra1[3]);
        img_8uc1 = cvLoadImage( "img2.jpg", CV_LOAD_IMAGE_GRAYSCALE);

        img_edge = cvCreateImage( cvGetSize(img_8uc1), 8, 1 );
        img_8uc3 = cvCreateImage( cvGetSize(img_8uc1), 8, 3 );
        cvThreshold( img_8uc1, img_edge, g_thresh, 255, CV_THRESH_BINARY );
        storage = cvCreateMemStorage();
        first_contour = NULL;
        Nc = cvFindContours(img_edge, storage, &first_contour );
        n=0;
          maxelem=0;
        maior = NULL;
        //  printf("\n\nHit  any  key  to  draw  the  next  contour,  ESC  to
quit\n\n");
        // printf( "Total Contours Detected: %d\n", Nc );
        for( c=first_contour; c!=NULL; c=c->h_next ){
        //printf("\n a \n");

       for( int i=0; i<c->total; ++i ) {
            CvPoint* p = CV_GET_SEQ_ELEM( CvPoint, c, i );
             //printf("\n b \n");
            // printf("    (%d,%d)\n", p->x, p->y );
            if(c->total>=maxelem ){
                  if(c->flags==1117360652){
                  maxelem=c->total;
                  cont=n;
                  Segunda[0][i]=Maior[0][i];
                  Segunda[1][i]=Maior[1][i];
                  Maior[0][i]=p->x;
                  Maior[1][i]=p->y;
                  maior = c;
                  // printf("\n c \n");
                  }
             }
       }
            //if((k = cvWaitKey()&0x7F) == 27)
            //break;
           n++;

        }

           cvCvtColor( img_8uc1, img_8uc3, CV_GRAY2BGR );
           cvDrawContours(
              src,
              maior,
              CVX_RED,
              CVX_BLUE,
              0,
              2,
              8
           );
```

```
            printf("Contour #%d flag=%d\n", n,maior->flags );
        cvShowImage( argv[0], src );
        cvWaitKey();
            printf(" %d elements:\n", maior->total );
         for(int i=0; i<100000; i++){
         vetorx[i]=Maior[0][i];
         vetory[i]=Maior[1][i];
         }

         // ponto esquerdo da sobrancelha esquerda
         pos=minimumValue(vetorx);
         N[0][13]=Maior[0][pos];
         N[1][13]=Maior[1][pos];

         // ponto direito da sobrancelha esquerda
         pos=maximumValue(vetorx);
         N[0][14]=Maior[0][pos];
         N[1][14]=Maior[1][pos];

         // ponto superior da sobrancelha esquerda
         pos=minimumValue(vetory);
         N[0][15]=Maior[0][pos];
         N[1][15]=Maior[1][pos];

         cvCvtColor( img_8uc1, img_8uc3, CV_GRAY2BGR );
         cvShowImage( argv[0], img_8uc3 );
         // cvWaitKey(0);
         cvDestroyWindow( argv[0] );
         cvReleaseImage( &img_8uc1 );
         cvReleaseImage( &img_8uc3 );
         cvReleaseImage( &img_edge );

         for(int i=0;i<100000;i++){
         Maior[0][i]=Maior[1][i]=Segunda[0][i]=Segunda[1][i]=-1;
         }

         CircleROI(posobra2[0],posobra2[1],posobra2[2],posobra2[3]);
         img_8uc1 = cvLoadImage( "img2.jpg", CV_LOAD_IMAGE_GRAYSCALE);

         img_edge = cvCreateImage( cvGetSize(img_8uc1), 8, 1 );
         img_8uc3 = cvCreateImage( cvGetSize(img_8uc1), 8, 3 );
         cvThreshold( img_8uc1, img_edge, g_thresh, 255, CV_THRESH_BINARY );
         storage = cvCreateMemStorage();
         first_contour = NULL;
         Nc = cvFindContours(img_edge, storage, &first_contour );
         n=0;
           maxelem=0;
         maior = NULL;
         //  printf("\n\nHit  any  key  to  draw  the  next  contour,  ESC  to
quit\n\n");
         // printf( "Total Contours Detected: %d\n", Nc );
        for( c=first_contour; c!=NULL; c=c->h_next ){
         //printf("\n a \n");

        for( int i=0; i<c->total; ++i ) {
             CvPoint* p = CV_GET_SEQ_ELEM( CvPoint, c, i );
              //printf("\n b \n");
             // printf("    (%d,%d)\n", p->x, p->y );
             if(c->total>=maxelem ){
                   if(c->flags==1117360652){
```

```
            maxelem=c->total;
            cont=n;
            Segunda[0][i]=Maior[0][i];
            Segunda[1][i]=Maior[1][i];
            Maior[0][i]=p->x;
            Maior[1][i]=p->y;
            maior = c;
            // printf("\n c \n");
            }
      }
 }
      //if((k = cvWaitKey()&0x7F) == 27)
      //break;
     n++;

  }

     cvCvtColor( img_8uc1, img_8uc3, CV_GRAY2BGR );
     cvDrawContours(
        src,
        maior,
        CVX_RED,
        CVX_BLUE,
        0,
        2,
        8
     );
     printf("Contour #%d flag=%d\n", n,maior->flags );
 cvShowImage( argv[0], src );
 cvWaitKey();
     printf(" %d elements:\n", maior->total );
   for(int i=0; i<100000; i++){
  vetorx[i]=Maior[0][i];
  // if(vetorx[i]!=-1)
        // printf("(%d,", vetorx[i]);
  vetory[i]=Maior[1][i];
  // if(vetory[i]!=-1)
        // printf("(%d,", vetory[i]);
  }

  // ponto esquerdo da sobrancelha direita
  pos=minimumValue(vetorx);
N[0][16]=Maior[0][pos];
N[1][16]=Maior[1][pos];

  // ponto direito da sobrancelha direita
  pos=maximumValue(vetorx);
N[0][17]=Maior[0][pos];
N[1][17]=Maior[1][pos];

  // ponto superior da sobrancelha direita
  pos=minimumValue(vetory);
N[0][18]=Maior[0][pos];
N[1][18]=Maior[1][pos];

  cvCvtColor( img_8uc1, img_8uc3, CV_GRAY2BGR );
  cvShowImage( argv[0], img_8uc3 );
  // cvWaitKey(0);

  cvReleaseImage( &img_8uc1 );
```

```
        cvReleaseImage( &img_8uc3 );
        cvReleaseImage( &img_edge );
        cvReleaseImage( &img1 );
        cvDestroyWindow( "tcc" );
        cvDestroyWindow( "contours" );
        cvDestroyWindow( argv[0] );
        for(int i=1;i<19;i++)
        printf(" x = %f  y = %f\n",N[0][i],N[1][i]);


      //                    r=sqrt(pow(N[0][20]-N[0][19],2)+pow(N[1][20]-
N[1][19],2))/sqrt(pow(M[0][20]-M[0][19],2)+pow(M[1][20]-M[1][19],2));
      r=1;
      R[0]=r*sqrt(pow(N[0][2]-N[0][1],2)+pow(N[1][2]-
N[1][1],2))/sqrt(pow(M[0][2]-M[0][1],2)+pow(M[1][2]-M[1][1],2));          //
abertura horizontal da boca
      //                         a=sqrt(pow(N[0][2]-N[0][1],2)+pow(N[1][2]-
N[1][1],2))/sqrt(pow(M[0][2]-M[0][1],2)+pow(M[1][2]-M[1][1],2));
      R[1]=r*(N[1][4]-N[1][3])/(M[1][4]-M[1][3]); // abertura vertical da
boca
      m1=(M[1][2]-M[1][1])/(M[0][2]-M[0][1]);
      m2=(M[1][4]-M[1][3])/(M[0][4]-M[0][3]);
      xm=(M[1][1]-M[1][1]+m2*M[0][3]-m1*M[0][1])/(m2-m1);
      if(M[0][4]==M[0][3])
            xm = M[0][4];
      n1=(N[1][2]-N[1][1])/(N[0][2]-N[0][1]);
      n2=(N[1][4]-N[1][3])/(N[0][4]-N[0][3]);
      xn=(N[1][1]-N[1][1]+n2*N[0][3]-n1*N[0][1])/(n2-n1);
      if(N[0][4]==N[0][3])
            xn = N[0][4];
      ym=M[1][1]+m1*(xm-M[0][1]);
      yn=N[1][1]+n1*(xn-N[0][1]);
      R[2]=r*(xn-N[0][1])/(xm-M[0][1]);   //   distancia   horizontal   do
cruzamento dos vetores da boca
      R[3]=r*(yn-N[1][3])/(ym-M[1][3]); // distancia vertical do cruzamento
dos vetores da boca
      // printf("\n----------------------------------\n m1=%f \t m2=%f \n
n1=%f \t n2=%f \n xm=%f \t xn=%f \n ym=%f \t yn=%f \n R[2]=%f \t R[3]=%f\n-
----------------------------------\n",m1,m2,n1,n2,xm,xn,ym,yn,R[2],R[3]);
      R[4]=r*sqrt(pow(N[0][6]-N[0][5],2)+pow(N[1][6]-
N[1][5],2))/sqrt(pow(M[0][6]-M[0][5],2)+pow(M[1][6]-M[1][5],2));          //
abertura horizontal olho esquerdo
      R[5]=r*sqrt(pow(N[0][7]-N[0][8],2)+pow(N[1][7]-
N[1][8],2))/sqrt(pow(M[0][7]-M[0][8],2)+pow(M[1][7]-M[1][8],2));          //
abertura vertical olho esquerdo
      R[6]=r*sqrt(pow(N[0][10]-N[0][9],2)+pow(N[1][10]-
N[1][9],2))/sqrt(pow(M[0][10]-M[0][9],2)+pow(M[1][10]-M[1][9],2));        //
abertura horizontal olho direto
      R[7]=r*sqrt(pow(N[0][11]-N[0][12],2)+pow(N[1][11]-
N[1][12],2))/sqrt(pow(M[0][11]-M[0][12],2)+pow(M[1][11]-M[1][12],2));     //
abertura vertical olho direito
      R[8]=r*sqrt(pow(N[0][14]-N[0][13],2)+pow(N[1][14]-
N[1][13],2))/sqrt(pow(M[0][14]-M[0][13],2)+pow(M[1][14]-M[1][13],2));     //
distancia horizontal da sobrancelha esquerda
      R[9]=r*sqrt(pow(N[0][15]-(N[0][14]+N[0][13])/2,2)+pow(N[1][15]-
(N[1][14]+N[1][13])/2,2))/sqrt(pow(M[0][15]-
(M[0][14]+M[0][13])/2,2)+pow(M[1][15]-(M[1][14]+M[1][13])/2,2));          //
distancia vertical da sobrancelha esquerda
```

```
      R[10]=r*sqrt(pow(N[0][17]-N[0][16],2)+pow(N[1][17]-
N[1][16],2))/sqrt(pow(M[0][17]-M[0][16],2)+pow(M[1][17]-M[1][16],2));     //
distancia horizontal da sobrancelha direita
      R[11]=r*sqrt(pow(N[0][18]-(N[0][17]+N[0][16])/2,2)+pow(N[1][18]-
(N[1][17]+N[1][16])/2,2))/sqrt(pow(M[0][18]-
(M[0][17]+M[0][16])/2,2)+pow(M[1][18]-(M[1][17]+M[1][16])/2,2));          //
distancia vertical da sobrancelha direita
      R[12]=r*(N[0][15]-N[0][13])/(M[0][15]-M[0][13]);      //      distancia
horizontal do ponto mais alto da sobrancelha esquerda
      R[13]=r*(N[1][13]-N[1][5])/(M[1][13]-M[1][5]); // distancia vertical
entre canto esquerdo da sobrancelha e olho esquerdos
      R[14]=r*(N[1][14]-N[1][6])/(M[1][14]-M[1][6]); // distancia vertical
entre canto direito da sobrancelha e olho esquerdos
      R[15]=r*(N[0][18]-N[0][16])/(M[0][18]-M[0][16]);      //      distancia
horizontal do ponto mais alto da sobrancelha direita
      R[16]=r*(N[1][16]-N[1][9])/(M[1][16]-M[1][9]); // distancia vertical
entre canto esquerdo da sobrancelha e olho direitos
      R[17]=r*(N[1][17]-N[1][10])/(M[1][17]-M[1][10]);      //      distancia
vertical entre canto direito da sobrancelha e olho direitos

      filestr.open ("file.txt", fstream::in | fstream::out | fstream::app);
      filestr<<R[0]<<"\t"<<R[1]<<"\t"<<R[2]<<"\t"<<R[3]<<"\t"<<R[4]<<"\t"<<
R[5]<<"\t"<<R[6]<<"\t"<<R[7]<<"\t"<<R[8]<<"\t"<<R[9]<<"\t"<<R[10]<<"\t"<<R[
11]<<"\t"<<R[12]<<"\t"<<R[13]<<"\t"<<R[14]<<"\t"<<R[15]<<"\t"<<R[16]<<"\t"<
<R[17]<<"\t"<<nome<<endl;
      filestr.close();

      K++;
      }
         }
       }
          return 0;
      }
```

15 ANEXO III – PROGRAMA MATLAB

```
function [ ] = emotion(  )

load('emotionnetwork.mat');
strLabelFile='file.txt';
fid=fopen(strLabelFile);
% lê do arquivo file.txt todos os valores escritos
imageLabel=textscan(fid,'%s %s %s %s %s %s %s %s %s %s %s %s %s %s %s
 %s %s %s %s','whitespace','\t');
fclose(fid);
a=size(imageLabel{1});
 for k=1:a(1)
     for l=1:18
        %  transforma  os  valores  que  strings  em  float  e  guarda  na
matriz b
         aux = str2num(imageLabel{l}{k});
         b(l,1)=aux;
     end
     % aplica a rede neural sobre a matriz b
    c=sim(net,b);
    total = c(1,1)+c(2,1)+c(3,1)+c(4,1)+c(5,1)+c(6,1);
    imageLabel{19}{k}
    % imprimi para cada emoção, a porcentagem de presença
    disp(sprintf('Raiva = %f', 100*c(1,1)/total));
    disp(sprintf('Medo = %f', 100*c(2,1)/total));
    disp(sprintf('Felicidade = %f', 100*c(3,1)/total));
    disp(sprintf('Neutro = %f', 100*c(4,1)/total));
    disp(sprintf('Tristeza = %f', 100*c(5,1)/total));
    disp(sprintf('Surpresa = %f', 100*c(6,1)/total));
end


end
```