8,0 (oito)

UNIVERSIDADE DE SÃO PAULO
ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

RECONHECIMENTO DE IMPRESSÕES DIGITAIS

REDES NEURAIS X EXTRAÇÃO DE CARACTERÍSTICAS

ELABORADO POR:
MARCELO FURTADO DE ARAGÃO
MARTIM FILIPE DE CAZULA E CONSTANTINO

ORIENTADOR:
JUN OKAMOTO JR.

SÃO PAULO
1999

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

DEPARTAMENTO DE ENGENHARIA MECÂNICA

RECONHECIMENTO DE IMPRESSÕES DIGITAIS

REDES NEURAIS X EXTRAÇÃO DE CARACTERÍSTICAS

DISSERTAÇÃO APRESENTADA À ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO PARA OBTENÇÃO DO TÍTULO DE GRADUAÇÃO EM ENGENHARIA

MARCELO FURTADO DE ARAGÃO

N^{O}USP 1232943

MARTIM FILIPE DE CAZULA E CONSTANTINO

N^{O}USP1509447

PROF. ORIENTADOR:

JUN OKAMOTO JR.

SÃO PAULO

1999

FICHA CATALOGRÁFICA

ARAGÃO, Marcelo F. ;CONSTANTINO, Martim F. C., **Reconhecimento de Impressões Digitais. Redes Neurais X Extração de Características,** pp, 1999

AOS MEUS FILHOS, IAGO E VICTOR, POR SUA IMPORTÂNCIA NA MINHA VIDA.

AGRADECIMENTOS

Aos nossos pais, por todo apoio e incentivo ao longo desses difíceis anos de faculdade.

À minha esposa Danielly, por todo carinho e dedicação durante o tempo em que estamos juntos.

Às cervejadas, por terem proporcionado bons momentos de alegria.

À minha namorada Denise, por sempre me apoiar nas horas difíceis.

Aos meus amigos de faculdade, pelos quase sempre bons momentos.

Ao CAM, onde passei boa parte do meu tempo durante os últimos anos.

Índice

Resumo

Esse projeto Mecatrônico visa o reconhecimento de impressões digitais utilizando-se dois métodos, um com redes neurais e o outro através da extração de características das impressões. A idéia é no final comparar os resultados obtidos pelos dois métodos chegando a conclusões a respeito de qual deles é o mais apropriado para fazer esse tipo de reconhecimento.

Nesse projeto as impressões serão coletadas através de um scaner e então processadas pelos dois métodos, vale observar aqui que ambos os métodos foram programados em C++ Builder..

O método de redes neurais visa encontrar padrões no campo de orientação das impressões digitais e através desses padrões poder realizar um reconhecimento positivo do indivíduo.

Já a extração de características visa encontrar pontos que caracterizam aquela determinada impressão e através desses pontos identificar se é o indivíduo ou não.

Esse projeto foi escolhido devido ao seu imenso campo de implantação e a carência de segurança que existe em muitos sistemas atualmente, melhores exemplos serão dados a seguir.

1. Introdução ao Reconhecimento de Impressões Digitais

1.1. Histórico

A impressão digital é uma característica humana conhecida há muito tempo. Desenhos representativos da mesma já foram encontrados até em cavernas pré-históricas. Falando-se em tempos mais recentes, por volta do ano 1856, um inglês chamado Sir William Hershel começou a utilizar impressões digitais em contratos. Ele era um administrador na Índia, e estava muito desapontado com os nativos, pois esses não costumavam honrar seus contratos só porque os haviam assinado. Nesse caso, porém, a utilização de impressões digitais como forma de identificação não tinha nada de científico, tinha sim um tom supersticioso. Hershel acreditava que por deixar uma marca de seus corpos nos contratos, os nativos deixariam de renegar os mesmos.

Foi nesse mesmo século (séc. XIX) que cientistas começaram a se interessar por impressões digitais, mas muito antes disso, as sociedades estavam interessadas em métodos que pudessem identificar transgressores. Por exemplo, os romanos costumavam tatuar mercenários, para que suas tropas soubessem com que tipo de pessoa estavam lidando; alguns países costumavam cortar uma das mãos de um ladrão, como uma forma de penalizá-

lo por furtar objetos. Mas conforme o mundo foi se tornando mais civilizado, essas práticas foram sendo abolidas.

No começo do século XIX, Alphonse Bertillon desenvolveu o primeiro método científico para a identificação de indivíduos. Esse método, chamado Bertillonage, consistia da medida de várias características do corpo de uma pessoa, como o tamanho do pé, largura da cabeça, cúbito (distância entre a ponta do dedo indicador e o nariz, com o braço esticado para o lado, na altura do ombro). Associado a essas medidas, havia uma série de outras características, como cor de olhos e pele, fotografias de frente e lado e nome do indivíduo, o que compunha a ficha da pessoa.

Esse método se espalhou rapidamente, com várias regiões do mundo o utilizando. Mas então, começaram a surgir alguns problemas. Conforme cresciam o número de pessoas registradas, mais difícil era de localizar alguém, pois o número grande de fichas atrasava o processo. Outro problema era a padronização das medidas. As formas de se medir podiam variar de localidade para localidade, o que podia gerar confusões. Mas esses dois problemas podiam ser contornados de certa forma, mas um terceiro problema aconteceu e decretou de vez o fim do uso desse método: as medidas não eram únicas para cada indivíduo. No ano de 1903, um homem chamado Will West foi preso. Embora afirmava que era a sua primeira vez na prisão, o registro da polícia não confirmava isso. Havia um registro com o nome de Willian West, com medidas praticamente idênticas e fotografias muito parecidas. Will West teria uma pena maior por ser considerado reincidente, não fosse um pequeno detalhe: Willian

West ainda estava preso, e não poderia ser Will West. Um exame de suas impressões digitais constatou que realmente não eram a mesma pessoa.

Voltando à Índia, Sir Willian Hershel, depois de ter registrado um grande número de impressões digitais, percebeu que essas eram diferentes para cada pessoa, e que, sendo assim, poderiam ser utilizadas para identificação. E percebeu mais, as impressões não mudavam durante a vida, o que ocorria com outras características físicas. Outros pesquisadores, como Dr Henry Faulds e Sir Francis Galton chegaram à mesma conclusão.

Porém, ainda existia o problema do grande número de registros, o que atrasava bastante um processo de identificação. Era preciso inventar um método de classificação, que dividisse as impressões em algumas classes, para agilizar o processo. Henry Faulds inventou um método para essa classificação, usado até hoje, enquanto Francis Galton conseguiu provar que as impressões digitais não mudam durante a vida e que são únicas para cada indivíduo: a chance de duas idênticas é de 1 em 64 bilhões.

1.2. Introdução

A identificação pessoal na atualidade tornou-se um problema crítico, por isso ela deve ser feita através de um processo que propicie o máximo de confiança. Um exemplo dessa necessidade é controle do acesso de pessoas a certos privilégios e facilidades em computadores, dado que esse acesso feito pela pessoa errada pode levar essa pessoa a ter informações confidenciais de empresas, bancos etc. Só para reforçar a importância da identificação pessoal atualmente aqui estão alguns números:

- Em torno de um bilhão de dólares em benefícios de previdência social são roubados nos Estados Unidos todo ano através de golpes utilizando falsas identidades;
- A MasterCard estima que a falsificação e roubo de cartões de credito leva 450 milhões de dólares todo ano;
- 1 bilhão de dólares em ligações de celulares roubados são feitos todo ano ;
- O departamento de Imigração e Nacionalização dos EUA estima que 3000 imigrantes ilegais entram pela fronteira com México

Como mostrado, o prejuízo com problemas de identificação são muito grandes. Por exemplo, se houvesse um método confiável è barato de confirmação da identidade do dono de um cartão de identificação, poderia diminuir a perda com fraudes e roubos em até 3 bilhões anualmente. Por isso o interesse em uma

forma confiável e barata de identificação em áreas como a comercial, a financeira , a civil etc... tem aumentado a cada dia.

Há dois tipos de abordagem para o problema da identificação de indivíduos: Verificação e Reconhecimento.

A Verificação trata basicamente de confirmar ou negar a identidade de uma pessoa, enquanto o Reconhecimento se preocupa em estabelecer a identidade do indivíduo.

Normalmente, tanto na Verificação como o Reconhecimento, uma pessoa é identificada de duas maneiras:

- Através de algum objeto que o identifique, uma chave por exemplo;
- Através de alguma coisa que só você saiba, uma senha por exemplo.

Outra abordagem para a identificação pessoal, e que será visada nesse trabalho, é baseada no reconhecimento de características da pessoa. As características podem ser tanto físicas, como impressão digital, formato da mão etc. como de outros tipos, voz, assinatura, etc. A esse tipo de abordagem é dado o nome de biométrica. Como as características biológicas não podem ser esquecidas, como senhas, e não podem ser facilmente falsificadas, como chaves, esse tipo de abordagem é considerado o mais confiável método para a identificação pessoal.

O advento das redes de computadores mais velozes possibilitou o surgimento de novas oportunidades para utilização desse tipo de tecnologia, visto que o processamento e recebimento de dados pode ser feito através dessas redes mais velozes.

O padrão de fluxo de vales e cristas existente na palma das mãos é chamado de impressão da palma. A formação dessa impressão depende das condições iniciais da mesoderme embrionária, por isso cada indivíduo possui uma impressão diferente do outro. Logo usar a impressão digital para identificação, visto que ela faz parte dessa impressão da palma, é bastante confiável porque cada um possui a sua, mesmo que elas sejam gêmeos idênticos suas impressões digitais serão diferentes.

A impressão digital é uma das mais naturais tecnologia biométrica de identificação e é considerada uma prova legitima em tribunais do mundo inteiro, por isso elas são utilizadas nas divisões criminais das policias pelo mundo todo como uma forma de pegar os criminosos. Mas agora o uso de impressões digitais está aumentado entre as pessoas comuns e áreas comerciais, esse aumento é devido a melhoria do desempenho dos leitores de impressões e do processamento das imagens adquiridas, logo a analise das impressões digitais tornou-se o tipo de reconhecimento biométrico mais comum.

1.3. Aplicação

Muitas são as aplicações para o reconhecimento de indivíduos através de suas impressões digitais, como já mencionado no item 1.2. O nosso trabalho visa reconhecer uma pessoa como cadastrada, ou não, num pequeno banco de dados. Esse reconhecimento será realizado por verificação de uma a uma das impressões registradas.

O trabalho consiste de duas abordagens diferentes para o processo de reconhecimento. A primeira abordagem, explicitada no capítulo 2 desse trabalho, trata de extração de características das impressões, enquanto a segunda abordagem, contida no capítulo 3, utiliza redes neurais para identificação dos parâmetros das digitais.

1.4. Representação da Impressão Digital

Uma imagem de uma impressão digital é formada por cristas e vales, sendo a primeira os filetes mais altos e o segundo as depressões existentes entre elas.

As características importantes na representação de uma digital são basicamente três:

- Pontos singulares;
- Campo de orientação;
- Minúcias da digital.

1.4.1 Pontos Singulares

Pontos Singulares são pontos especiais numa impressão digital. Pontos onde há uma curvatura bastante acentuada são chamados de "core" , e regiões onde há a formação de uma espécie de triângulo, ou melhor, um delta (como os dos rios) são chamadas de "delta"(ver figura 1.1). Esses pontos são geralmente utilizados na classificação de uma impressão digital. A classificação de uma impressão digital é útil quando se trabalha com bancos de dados muito grandes, pois divide as impressões digitais em algumas classes, agilizando o processo de comparação. Esses pontos podem ser também utilizados como pontos de referência, quando do alinhamento das imagens na comparação.

Figura 1.1 – Minúcia(O), Delta(Δ) e Core(□)

1.4.2. Campo de Orientação

Campo de Orientação é o conjunto de direções que os pontos característicos da impressão digital possui, ou seja, cada ponto característico da digital está localizado em cima de um vale ou de uma crista, sendo que tanto um quanto outro possuem uma trajetória e a direção dessa trajetória nesses pontos especiais é que formam esse campo de orientação. A figura 1.2 mostra o que é um campo de orientação.

Esse campo também é de fundamental importância para a identificação porque, diferentemente do caso anterior, ele serve como característica de comparação entre as digitais em processo.

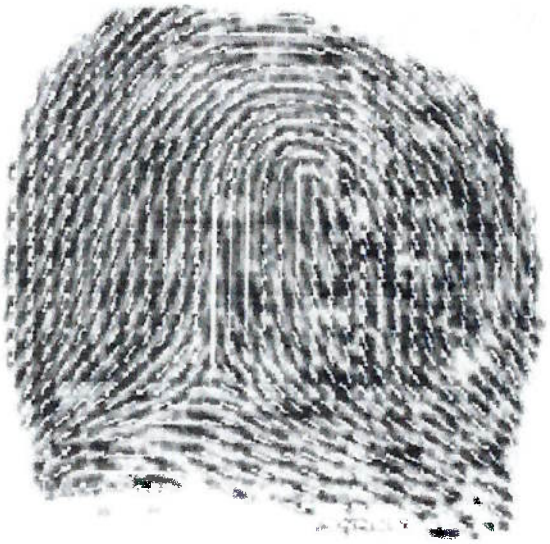


Figura 1.2 - Campo de Orientação

1.4.3. Minúcias da Digital

As minúcias de uma digital são as principais características da mesma, e o que elas são na verdade é o encontro de duas cristas ou a criação de uma crista. A fig. 1.3 ilustra melhor o que são esses pontos. A comparação da existência ou não dessas minúcias é a base do processo de identificação das digitais.

Figura 1.3 - Tipos de Minúcias

1.5. Processo de Preparação da Imagem

Há uma série de procedimentos necessários para realizar o reconhecimento de uma impressão digital, tanto pelo método de extração de características como pelo de redes neurais. Mas a princípio deve ser feita uma preparação da imagem da impressão, como ilustra o esquema da figura 1.4.

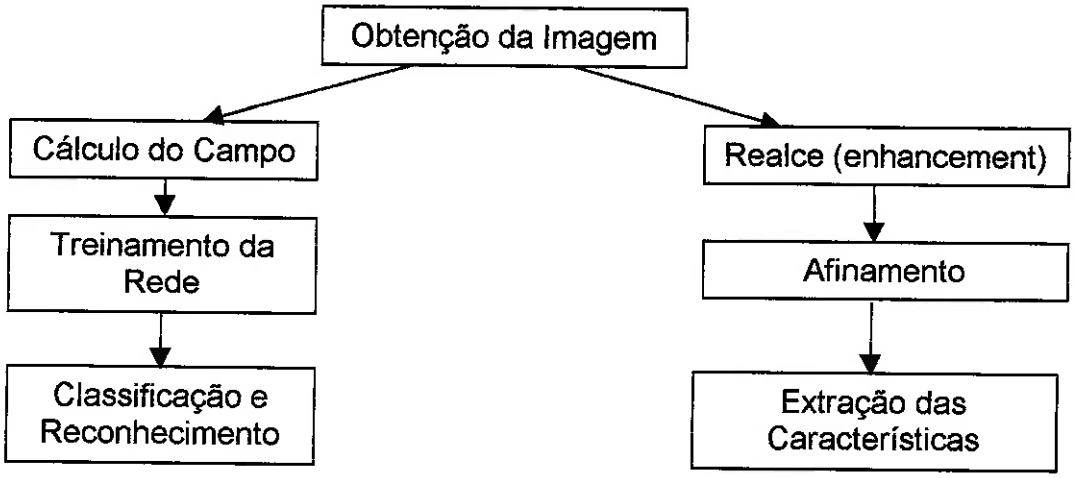


Figura 1.4: Sequencia de passos para o reconhecimento

Quanto a obtenção da imagem, ele se dará pelo método tradicional: utilizando uma folha de panel e tinta. Uma vez coletada a imagem, ele será *escaneada* e gravada em um arquivo. Essas imagens tem tamanho 512x512 pixels, e resolução 480 dpi.

2. Método da Extração de Características (Implementação)

Essa parte do projeto trata da implementação do software que realizará a identificação por meio de extração das características. Alguns procedimentos descritos aqui são comuns a ambas as abordagens do projeto, e não serão repetidos na próxima parte. Os passos da implementação seguem o diagrama representado no item 1.5.

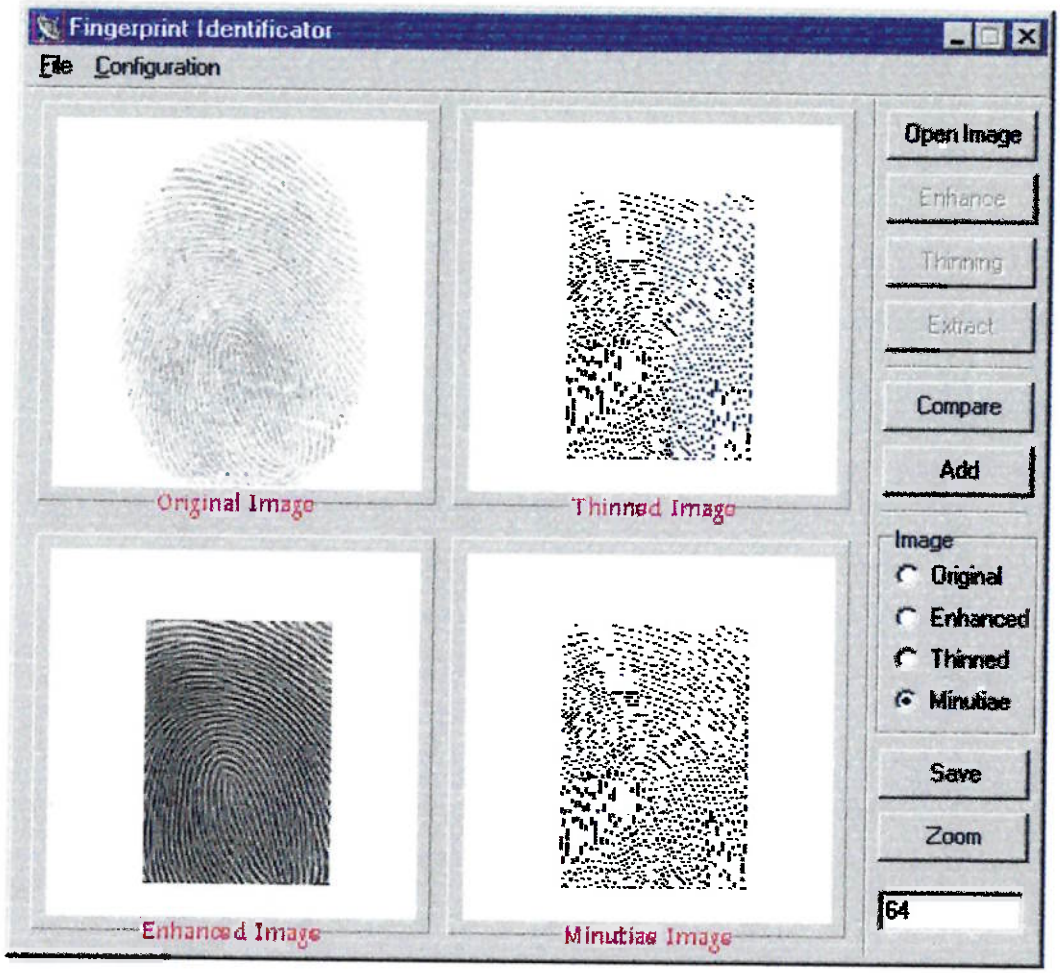


Figura 2.1 - Tela do Programa

2.1. Conversão de Imagem

Uma vez que se obtém a imagem através de um *scanner*, no formato especificado (ver item 1.5), devemos convertê-la em uma matriz de números inteiros. Para apresentar os resultados das diversas operações na tela do programa, é necessário também uma conversão de uma matriz de inteiros para uma imagem.

As imagens das impressões devem estar em tons de cinza. Ou seja, têm pixels variando entre 0 (preto) e 255 (branco). Porém, embora obtidas em tons de cinza, o programa interpreta as cores da imagem com o sistema RGB (0x000000 para preto e 0xFFFFFF para branco). Para os tons de cinza, as três cores (Red Green Blue) tem mesma intensidade.

Na conversão Imagem→ Inteiros, devemos transformar um número de 6 dígitos hexadecimais em um de apenas 2. Uma vez que a intensidade para as três cores do RGB é igual, basta pegar uma delas:

$$\text{Cor (tons de cinza)} = \text{Cor (RGB)} \;\&\; 0xFF$$

Na conversão Inteiros→Imagem, devemos fazer a operação inversa. Basta pegar a intensidade e aplicá-la nas três cores do RGB.

$$\text{Cor (RGB)} = \text{Cor (tons de cinza)} * (1+256+256^2)$$

$$\text{Cor (RGB)} = \text{Cor (tons de cinza)} * 65793$$

2.2. Realce (Enhancement)

O enhancement é um procedimento de grande utilidade no processo de reconhecimento. Conforme a qualidade da imagem obtida, a não realização de um processo de enhancement pode levar a erros graves [1,3], como:

- Criação de um grande número de minúcias aleatórias
- Perda de boa parte das minúcias verdadeiras
- Erro na localização (posição e orientação) das minúcias

2.2.1. Notação

Para o completo entendimento do processo de enhancement, é necessária uma lista de notações [3] que serão utilizadas nos próximos itens.

Chamaremos de I a imagem de uma impressão digital em tons de cinza, e a definiremos como uma matriz quadrada (NxN) onde $\mathrm{I}(i,j)$ representa a intensidade do pixel da i-ésima linha e j-ésima coluna.

A **média** e a **variância** de I são definidas pelas seguintes equações :

$$M(\mathrm{I}) = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\mathrm{I}(i,j)$$

$$VAR(\mathrm{I}) = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}(\mathrm{I}(i,j) - M(\mathrm{I}))^2$$

A **imagem de orientação**, O , é definida como uma matriz NxN, onde O(i,j) representa a orientação local da crista no ponto (i,j). No nosso trabalho, a imagem será dividida em blocos WxW e será calculado apenas um valor de orientação para cada bloco.

Por fim, a **imagem de frequência**, f, também é uma matriz NxN, com f(i,j) representando a frequência local da crista. Esse valor é definido como a frequência das cristas e vales numa certa vizinhança, na direção perpendicular à direção de orientação. Novamente, utilizaremos cálculos em blocos WxW.

2.2.2. Algoritmo

O processo de enhancement é dividido em várias etapas [3]:

- Normalização
- Estimativa do Campo de Orientação
- Estimativa das Frequências Locais
- Filtragem

2.2.2.1 Normalização

Dada a definição de média e variância de I, podemos definir a imagem normalizada **G**(i,j) como:

$$G(i,j) = M_0 + \sqrt{\frac{VAR_0(I(i,j)-M)^2}{VAR}} \quad , \text{ caso I(i,j) > M}$$

$$G(i,j) = M_0 - \sqrt{\frac{VAR_0(I(i,j)-M)^2}{VAR}} \quad , \text{ caso contrário}$$

O processo de normalização determina um novo valor tanto para a média quanto para a variância. Não objetiva uma melhoria na imagem, e sim, reduz a variação dos tons de cinza ao longo da imagem, o que auxilia todos os processos subsequentes.

A figura 2.2 mostra uma imagem normal e uma normalizada. Os valores utilizados para média(M_o) e variância(VAR_o) são, respectivamente, 128 e 4000.

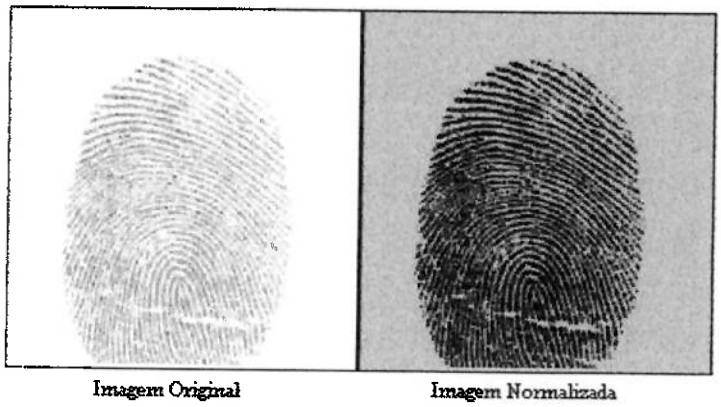


Figura 2.2 – Imagem Original e Normalizada

2.2.2.2. Estimativa do Campo de Orientação

A obtenção do campo de orientação (ou imagem de orientação) se dá a partir da imagem normalizada **G**. Os principais passos para a obtenção desse campo são [3]:

1) Dividir **G** em blocos de tamanho WxW (16x16)

2) Calcular os gradientes (∂) de cada pixel, nas direções x e y.

Estimar o campo de orientação em cada bloco utilizando as seguintes equações:

$$V_x(i,j) = \sum_{u=i-W/2}^{i+W/2} \sum_{v=j-W/2}^{j+W/2} 2\partial_x(u,v)\partial_y(u,v)$$

$$V_y(i,j) = \sum_{u=i-W/2}^{i+W/2} \sum_{v=j-W/2}^{j+W/2} \left(\partial_x^2(u,v) - \partial_y^2(u,v)\right)$$

$$\theta(i,j) = \frac{1}{2}\tan^{-1}\left(\frac{V_x(i,j)}{V_y(i,j)}\right)$$

Matematicamente, isso representa a direção ortogonal à direção dominante do Espectro de Fourier dentro de cada bloco.

4) Devido à presença de ruídos, minúcias, etc..., a estimativa do campo de orientação nem sempre é muito correta. Uma vez que o campo não apresenta variações muito bruscas em regiões onde não há minúcias ou ruídos, um filtro passa-baixas pode ser utilizado para modificar as orientações incorretas.

Para realizar essa filtragem, devemos converter a imagem de orientação num vetor contínuo, segundo segue abaixo:

$$\Phi_x(i,j) = \cos(2\theta(i,j))$$

$$\Phi_y(i,j) = \text{sen}(2\theta(i,j))$$

Com essas componentes do vetor do campo, podemos aplicar a filtragem:

$$\Phi_x'(i,j) = \sum_{u=-W_\Phi/2}^{W_\Phi/2} \sum_{v=-W_\Phi/2}^{W_\Phi/2} W(u,v)\Phi_x(i-uw, j-vw)$$

$$\Phi_y'(i,j) = \sum_{u=-W_\Phi/2}^{W_\Phi/2} \sum_{v=-W_\Phi/2}^{W_\Phi/2} W(u,v)\Phi_y(i-uw, j-vw)$$

E, finalmente, recalcular o campo de orientação:

$$O(i,j) = \frac{1}{2}\tan^{-1}\left(\frac{\Phi_x'}{\Phi_y'}\right)$$

O cálculo dos gradientes pode ser efetuado de diversas maneiras. Duas maneiras foram propostas nesse trabalho: Operadores Simples e de Sobel.

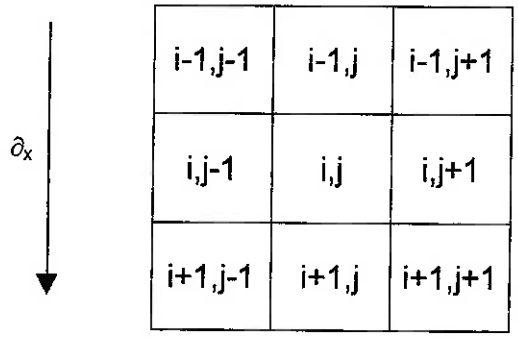
Os **Operadores Simples** se baseiam numa diferença simples entre um pixel adjacente e o pixel onde se calcula o gradiente. As equações seguem abaixo, notando-se que a direção "z" nada mais é que a direção oposta a "x", isto é, z=-x. O motivo da utilização desse outro eixo será explicado mais adiante.

$$\partial_x = G(i+1,j) - G(i,j)$$

$$\partial_y = G(i,j+1) - G(i,j)$$

$$\partial_z = G(i-1,j) - G(i,j)$$

Os **Operadores de Sobel** também se baseiam em diferenças, porém, nesse caso, se faz a diferença entre 3 pixels a frente do pixel no qual se quer calcular o gradiente e 3 pixels atrás.



$$\partial_x = G(i+1, j-1) + 2G(i+1, j) + G(i+1, j+1) - G(i-1, j-1) - 2G(i-1, j) - G(i-1, j+1)$$
$$\partial_y = G(i-1, j+1) + 2G(i, j+1) + G(i+1, j+1) - G(i-1, j-1) - 2G(i, j-1) - G(i+1, j-1)$$
$$\partial_z = G(i-1, j-1) + 2G(i-1, j) + G(i-1, j+1) - G(i+1, j-1) - 2G(i+1, j) - G(i+1, j+1)$$

O cálculo do campo de orientação através do uso de V_x e V_y é válido, desde que se tome certos cuidados. Por exemplo, quando V_y for muito menor que V_x o resultado da divisão tenderia a infinito, o que é inviável em termos de programação. Além disso, desejamos obter valores de ângulo entre 0° e 180°, não interessando valores negativos. É necessário, então, converter direções dadas por ângulos negativos em direções com ângulo positivo correspondente.

Afim de realizar uma análise mais profunda a respeito do cálculo do campo de rotação, devemos primeiramente entender como se comportam as variáveis V_x e V_y ao longo das possíveis direções.

Figura 2.3 – Imagem Original e Gradientes Simples e de Sobel

A variável V_y representa a diferença entre gradientes na direção x e y. Por exemplo, a 0° há muitos pixels com gradiente em x, mas sem gradientes em y. Portanto, para esse ângulo, V_y atinge valor máximo. Conforme vai se inclinando a direção, vão surgindo gradientes na direção y, o que reduz o valor de V_y. Quando a direção atinge 45°, os gradientes nas direções x e y se igualam, resultando em V_y nulo. Com 90°, atinge-se o valor mínimo. O raciocínio prossegue idêntico até 180° .

A variável V_x é mais complexa de se tratar, e traz complicações na implementação do programa. Quando os pixels só apresentam gradientes em uma única direção (casos do 0° e 90°), V_x é nulo. Conforme vai se aumentando o ângulo a partir de 0°, vão surgindo alguns pixels com gradientes nas duas direções. Além disso, esses gradientes têm mesmo sinal, o que resulta em V_x positivo. No ângulo de 45°, V_x atinge seu valor máximo, pois todos os pixels com gradiente em x também possuem gradiente em y, e vice-versa.

Os problemas surgem a partir do ângulo de 90°. Pegue o exemplo de uma linha de espessura de 1 pixel e ângulo 135°. Os únicos pixels que apresentam gradientes nos dois sentidos são os pixels que compõem a linha. E como esses gradientes têm mesmo sinal, gera V_x positivo. Porém, se a linha tiver espessura de 2 pixels (de mesma intensidade), nenhum pixel terá gradientes nos dois sentidos, o que resulta em V_x nulo. Nesses dois casos, apesar das linhas terem mesma inclinação, teríamos resultados diferentes.

Porém, as linhas de uma impressão não são formadas por pixels de mesma intensidade, e essa hipótese será utilizada nos raciocínios seguintes. Nesses casos, teríamos alguns pixels (aqueles que se localizam no centro das cristas)

com gradientes de mesmo sinal e outros pixels (em maior quantidade, nas bordas das cristas) com gradientes de sinal contrário. Teríamos, então, V_x negativo. De fato, alguns testes com o programa indicam que V_x é realmente negativo para ângulos maiores que 90°.

A figura 2.4 mostra um esquema do comportamento das variáveis V_x e V_y ao longo dos diversos ângulos:

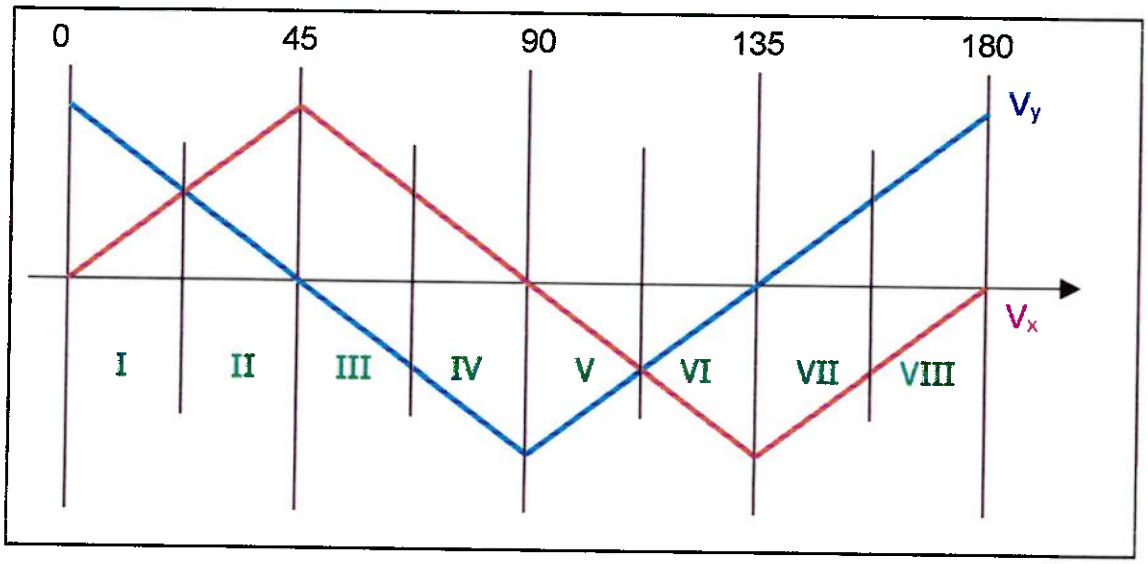


Figura 2.4 – Comportamento das Variáveis Vx e Vy

Devemos, entretanto, prestar atenção num detalhe. Quando calculamos o ângulo para uma direção de 45°, temos um certo V_x que só recebe valores positivos. No cálculo de 135°, V_x recebe valores negativos, mas também pode receber alguns positivos, o que pode gerar imprecisão no cálculo desse ângulo. Para corrigir esse problema, e tentar extrair somente os valores negativos, utilizamos o eixo z, definido anteriormente. A figura 2.5 mostra como funciona a utilização desse eixo.

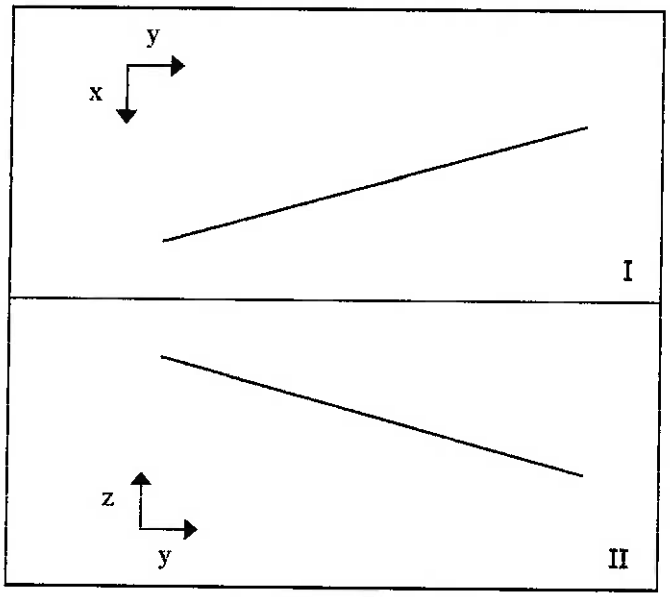


Figura 2.5 – Utilização do eixo Z

O quadro I mostra o cálculo com os eixos normais, de uma reta com inclinação menor que 90°. Para uma inclinação maior que 90°, utilizamos o eixo z ao invés de x. Nota-se que o quadro II nada mais é que a imagem espelhada do quadro I. Se fizermos os cálculos para ambos os quadros, obteremos o mesmo resultado. Isso elimina a obtenção de valores positivos e negativos para V_x, tornando o processo mais preciso.

Como podemos perceber (ver fig. 2.4) , cada um dos 8 trechos de ângulo tem sua peculiaridade no cálculo, e requerem uma adaptação da fórmula.

Trecho I – Utiliza-se a equação normalmente, sem alterações.

$$\theta(i,j) = \frac{1}{2}\tan^{-1}\left(\frac{V_x(i,j)}{V_y(i,j)}\right)$$

Trecho II – A diferença entre esse trecho e o anterior é que nesse, V_x se torna maior que V_y. Quando essa diferença for muito grande, o resultado tende a infinito, o que não é viável em termos computacionais. Para solucionar esse problema, podemos inverter V_x e V_y no cálculo do ângulo. Porém, isso resultaria não no ângulo correto, e sim no seu complemento para 90°. Basta, então, subtrair de 90° o ângulo obtido.

$$\theta(i,j)=\frac{1}{2}\left(90^o-\tan^{-1}\left(\frac{V_y(i,j)}{V_x(i,j)}\right)\right)$$

Trecho III – Nesse trecho, assim como no próximo, a relação entre V_x e V_y é negativa, o que gera um ângulo também negativo. Devemos transformar esse ângulo no seu correspondente positivo, bastando somar 180°. Como nesse trecho V_x é maior que V_y, também é necessário fazer a inversão como no trecho II.

$$\theta(i,j)=\frac{1}{2}\left(180^o+\left(-90^o-\tan^{-1}\left(\frac{V_y(i,j)}{V_x(i,j)}\right)\right)\right)$$

Trecho IV – Como já mencionado no trecho III, basta corrigir o ângulo negativo.

$$\theta(i,j)=\frac{1}{2}\left(180^o+\tan^{-1}\left(\frac{V_x(i,j)}{V_y(i,j)}\right)\right)$$

Trecho V – Esse trecho, assim como os próximos, é identificado por ter um V_x negativo. Porém, V_x não é utilizado nos cálculos, sendo substituído por V_z. Como essa substituição gera um efeito de espelhamento, o cálculo nesse trecho é igual ao trecho IV. Porém, o ângulo obtido não é o correto, e sim o complementar para 180°.

$$\theta(i,j) = 180^o - \left(\frac{1}{2}\left(180^o + \tan^{-1}\left(\frac{V_z(i,j)}{V_y(i,j)}\right)\right)\right)$$

Trecho VI – Utilizando raciocínio análogo ao do trecho V, esse trecho é igual ao trecho III.

$$\theta(i,j) = 180^o - \left(\frac{1}{2}\left(180^o + \left(-90^o - \tan^{-1}\left(\frac{V_y(i,j)}{V_z(i,j)}\right)\right)\right)\right)$$

Trecho VII – Idem, mas referente ao trecho II.

$$\theta(i,j) = 180^o - \left(\frac{1}{2}\left(90^o - \tan^{-1}\left(\frac{V_y(i,j)}{V_z(i,j)}\right)\right)\right)$$

Trecho VIII – Idem, mas referente ao trecho I.

$$\theta(i,j) = 180^o - \left(\frac{1}{2}\tan^{-1}\left(\frac{V_z(i,j)}{V_y(i,j)}\right)\right)$$

Uma vez calculados todos os ângulos (ainda sem a filtragem), podemos representá-los na própria impressão digital, como vemos na figura 2.6.

Figura 2.6 – Campo de Orientação

O próximo passo é filtrar o campo de orientação. Considerando que numa região onde não há singularidades a orientação das cristas não muda bruscamente, aplicarei um filtro para eliminar possíveis mudanças bruscas. O filtro é baseado numa média ponderada das orientações ao redor do ponto que se deseja corrigir [3]. Porém, calcular média de ângulos não é uma tarefa fácil, principalmente quando esses representam direções. Por exemplo, a direção 179° e a direção 1° são muito próximas, tendo como média 0°, e não 90°, que seria o resultado de uma operação aritmética.

Para resolver o problema, os valores de ângulos são dobrados (179° passa para 358°, e 1° para 2°, e portanto, ficando mais próximos) e são divididos em valores de seno e cosseno. No exemplo utilizado, teríamos valores de cosseno iguais para 2° e 358°, e valores de seno iguais em módulo, mais de sinais trocados. Portanto, a média do seno seria 0, e a do cosseno, um valor próximo de 1. A tangente teria, então, valor nulo, e o ângulo correspondente seria o 0°, como desejado.

Uma vez que se trata de uma média ponderada, devemos atribuir pesos para o filtro. A princípio, com base empírica, determinei os seguintes pesos:

1	1	2	1	1
1	6	8	6	1
2	8	10	8	2
1	6	8	6	1
1	1	2	1	1

Figura 2.7 – Filtro do Campo

Porém, alguns pontos particulares apresentavam resultados ruins após o uso do filtro. Esses pontos estão situados na região dos pontos *core*, que, conforme explicado anteriormente, são os pontos que apresentam inversão na orientação próxima de 180°. Nessas regiões, os ângulos mudam de forma tão brusca que não faz sentido algum calcular uma média. Seria preciso, então, desenvolver um processo que identificasse as regiões onde não fizesse sentido calcular a média, e então, não mexer na orientação já calculada para essa região. Aqui se assume um risco, pois pode-se estar deixando de filtrar uma região. Porém,

considero risco maior (o que os testes confirmaram) calcular uma média que poderia assumir qualquer valor, e que, portanto, dificilmente seria o valor correto.

Além disso, tal processo deveria manter a filtragem para as regiões consideradas boas. Para isso, dividi o filtro (5X5) em 4 filtros menores (3X3), de tal forma que o filtro resultante da sobreposição (parcial) desses filtros fosse igual ao filtro obtido anteriormente (fig. 2.7).

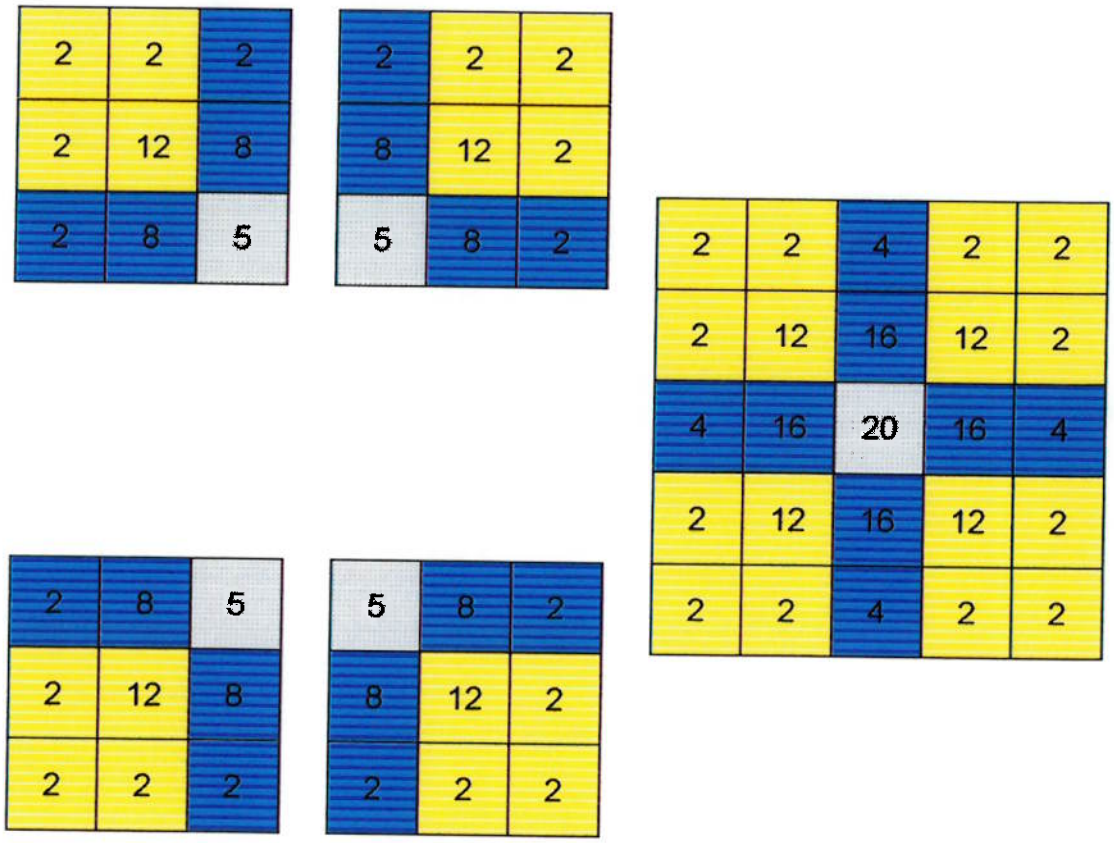


Figura 2.8 – Filtros Parciais e Final

A idéia de utilizar 4 filtros distintos é determinar se faz sentido calcular a média para uma tal região. Cada um desses filtros leva ao cálculo de uma média parcial. Numa região normal, as médias dos 4 filtros devem ser próximas, e

numa região irregular, as médias são bem distintas. Para determinar se uma região é irregular ou não, compara-se as médias parciais obtidas, e determina-se a maior diferença de valores entre elas. Se essa diferença for maior que 60°, a região é considerada ruim. Deve-se notar que o valor máximo dessa diferença é 90°. Por exemplo, a diferença entre as direções 10° e 170° não é 160°, e sim 20°, pois a direção 170° é igual à direção –10°.

No que se refere ao cálculo das médias, sejam elas parciais ou total, devemos novamente ter muita atenção. Deve-se realizar a soma ponderada dos senos e cossenos dos ângulos, e com base nesse resultado, obter a tangente média e por conseguinte a orientação média. Porém, a maneira de calcular essa média varia conforme os valores de somax (soma dos senos) e somay (soma dos cossenos), como vemos abaixo:

		Somax > 0	Somax < 0
Somay > 0	|Somax| maior que |Somay|	$\frac{1}{2}\left(\frac{\pi}{2} - tg^{-1}\left(\frac{sy}{sx}\right)\right)$	$\frac{1}{2}\left(\frac{3\pi}{2} - tg^{-1}\left(\frac{sy}{sx}\right)\right)$
	|Somax| menor que |Somay|	$\frac{1}{2}\left(tg^{-1}\left(\frac{sx}{sy}\right)\right)$	$\frac{1}{2}\left(2\pi + tg^{-1}\left(\frac{sx}{sy}\right)\right)$
Somay < 0	|Somax| maior que |Somay|	$\frac{1}{2}\left(\frac{\pi}{2} - tg^{-1}\left(\frac{sy}{sx}\right)\right)$	$\frac{1}{2}\left(\frac{3\pi}{2} - tg^{-1}\left(\frac{sy}{sx}\right)\right)$
	|Somax| menor que |Somay|	$\frac{1}{2}\left(\pi + tg^{-1}\left(\frac{sx}{sy}\right)\right)$	$\frac{1}{2}\left(\pi + tg^{-1}\left(\frac{sx}{sy}\right)\right)$

A figura abaixo (fig. 2.9) mostra o campo de orientação após a filtragem, e podemos compará-lo com o obtido anteriormente (fig. 2.6).

Figura 2.9 – Campo de Orientação após Filtragem

2.2.2.3. Estimativa das Frequências Locais

Numa região da impressão onde não há nenhuma singularidade, os tons de cinza das cristas e vales formam uma curva aproximadamente senoidal, na direção normal à orientação das cristas, conforme ilustra a figura 2.10.

O cálculo das frequências locais é feito a partir do seguinte procedimento [3]:

1) Dividir **G** em blocos de tamanho WxW (16x16)

2) Para cada bloco, centrado nas coordenadas (i, j), definir uma janela orientada de tamanho LxW (32x16) (ver fig. 10)

3) Para cada janela, calcular o vetor X (*x-signature*), X[0], X[1],...,X[L-1], onde:

$$X[k]=\frac{1}{W}\sum_{d=0}^{W-1}G(u,v)\ , \qquad k=0,1,2,...,L-1$$

$$u=i+\left(\frac{W}{2}-d\right)sen\theta+\left(k-\frac{L}{2}\right)cos\theta$$
$$v=j+\left(d-\frac{W}{2}\right)cos\theta+\left(k-\frac{L}{2}\right)sen\theta$$

Então, a partir do vetor X, calcular a distância $\tau(i, j)$ entre os picos, sendo que a frequência será o valor inverso desta distância $\Omega(i, j) = 1/\tau(i, j)$. No caso de não ser possível determinar picos consecutivos, atribuir o valor –1 para a frequência.

4) Para os pontos para os quais a frequência foi atribuída –1, é preciso interpolar um valor, baseado nos valores vizinhos. Essa interpolação é feita a partir das seguintes equações:

$$\Omega'(i,j)=\frac{\sum_{u=-\varpi/2}^{\varpi/2}\sum_{v=-\varpi/2}^{\varpi/2}W_g(u,v)\mu(\Omega(i-uw,j-vw))}{\sum_{u=-\varpi/2}^{\varpi/2}\sum_{v=-\varpi/2}^{\varpi/2}W_g(u,v)\delta(\Omega(i-uw,j-vw))}$$

$$\mu(x)=\begin{cases}0,se & x\leq 0\\ x,se & x>0\end{cases}$$

$$\delta(x)=\begin{cases}0,se & x\leq 0\\ 1,se & x>0\end{cases}$$

Onde:

W_g é um kernel Gaussiano, com média e variância 0 e 9, respectivamente.

$\varpi = 7$ é o tamanho do kernel.

5) Como as frequências variam suavemente numa impressão digital, é necessário usar um filtro passa-baixa para remover valores com variação brusca.

$$F(i,j) = \sum_{u=-\alpha/2}^{\alpha/2} \sum_{v=-\alpha/2}^{\alpha/2} Wl(u,v)\Omega'(i-uw, j-vw)$$

Onde Wl é o filtro e α=7 o seu tamanho

O cálculo do vetor X não apresenta problemas, mas determinar um valor de frequência a partir dele é muito difícil. Para localizar picos numa curva (a curva é representada pelo vetor X), é preciso primeiramente definir o que é um pico. No meu projeto, pico é todo ponto da curva cujos dois próximos pontos sejam menores do que eles, e decrescentes, e os dois anteriores também, mas crescentes. Um vale é definido analogamente. Uma vez determinados os picos, o período do sinal é determinado pela distância média entre picos adjacentes.

Entretanto, novamente é necessário cuidado no cálculo dos períodos. Se a distância entre um par de picos adjacentes for muito diferente da distância de outro par, o sinal provavelmente não será uma boa senóide, e deve ser descartado. No caso de só existir um par de picos, não há como fazer essa verificação, mas se nesse caso, a distância entre o primeiro pico e o próximo vale for muito diferente da distância desse vale até o segundo pico, novamente

devemos desconsiderar esse sinal. As curvas abaixo ilustram bem os conceitos envolvidos nesse cálculo.

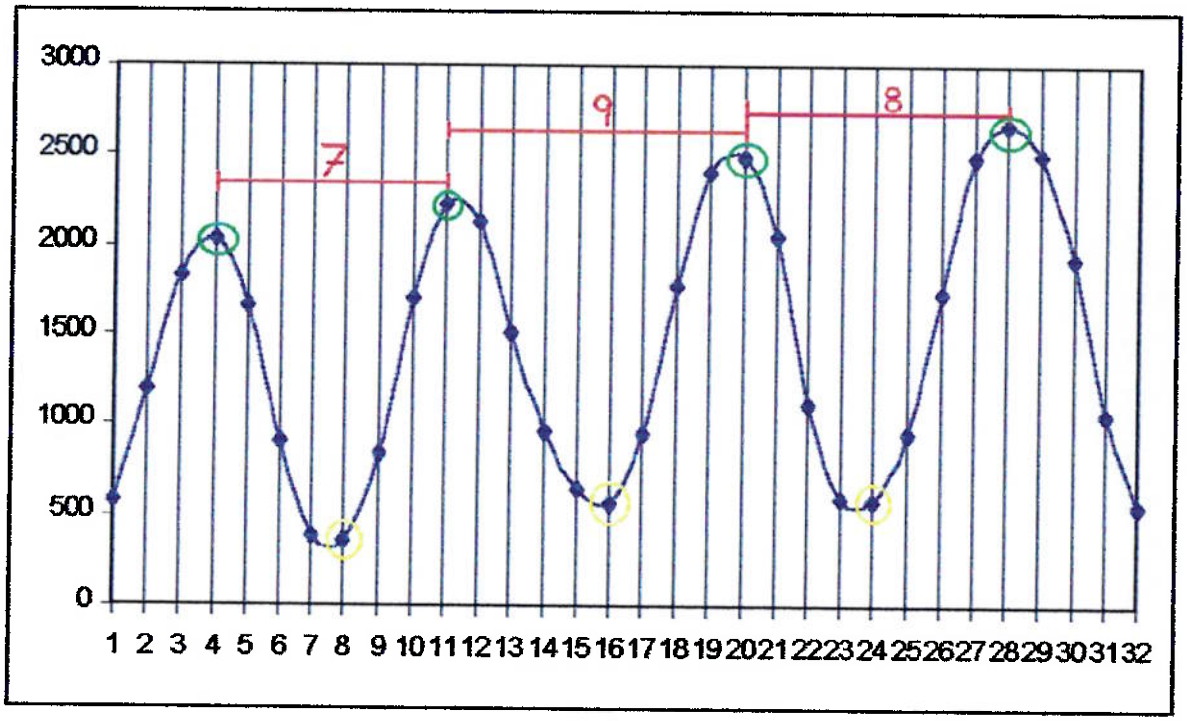


Figura 2.10 – Exemplo de Curva Senoidal

Na curva acima (fig. 2.10), os picos estão circulados em verde, e os vales, em amarelo. A distância entre os picos apresenta diferença (9-7=2) menor do que o limite imposto no programa (limite obtido empiricamente, igual à 3). Para a distância entre picos e vales, vale a mesma coisa. Portanto, podemos considerar a curva acima aproveitável, com período 8.

Na curva da fig. 2.11, a distância entre pico e vale varia muito (13-3=10>3), além de existir dois vales seguidos. Visualmente é fácil dizer que não se trata de uma curva senoidal, mas são necessários alguns parâmetros para que o programa conclua isso, e é isso que torna esse cálculo difícil.

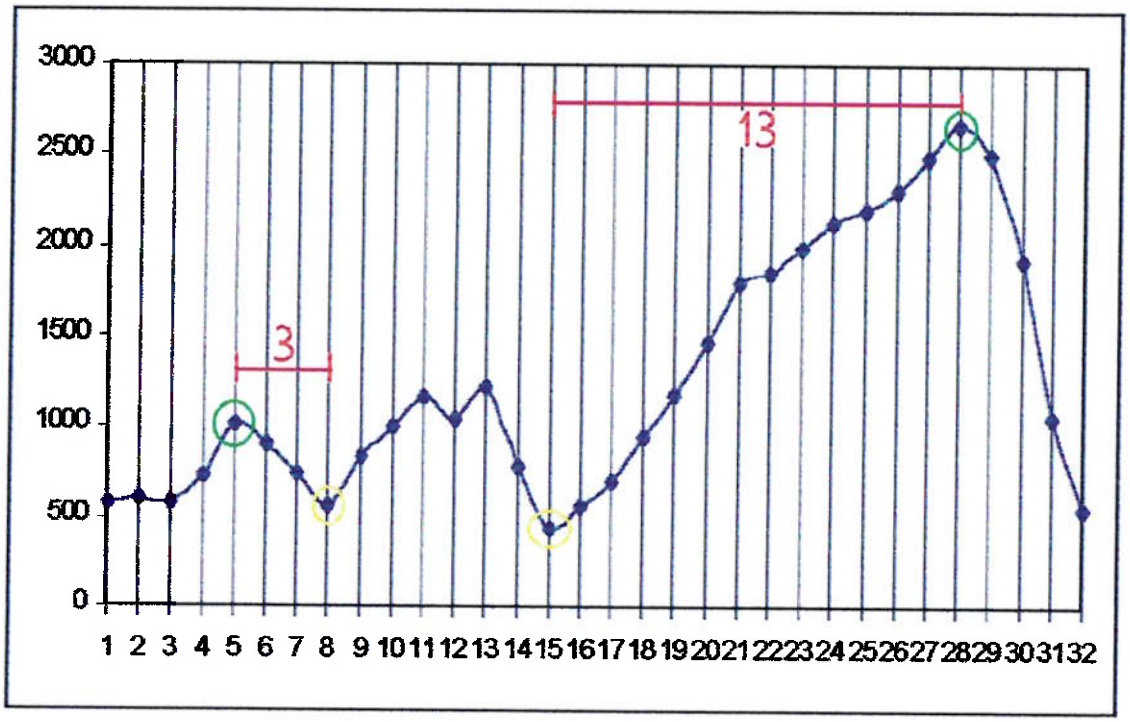


Figura 2.11 – Exemplo de Curva Não-Senoidal

Uma vez calculados os períodos das curvas, e obtidas as frequências, é necessário interpolar valores para os pontos onde não foi possível calcular um valor. Para tal, utilizamos o kernel Gaussiano. O kernel nesse caso é uma região extraída de uma curva de Gauss de domínio bi-dimensional. Os parâmetros dessa curva são média 0 (curva centrada no ponto (0,0)) e variância 9 (determina a "espessura" da curva). Como os valores dessa curva são pequenos, na implementação do programa multipliquei-os por uma constante, de forma que o kernel resultante ficou:

4.9	6.5	7.6	8.1	7.6	6.5	4.9
6.5	8.5	10.1	10.7	10.1	8.5	6.5
7.6	10.1	11.9	12.6	11.9	10.1	7.6
8.1	10.7	12.6	13.3	12.6	10.7	8.1
7.6	10.1	11.9	12.6	11.9	10.1	7.6
6.5	8.5	10.1	10.7	10.1	8.5	6.5
4.9	6.5	7.6	8.1	7.6	6.5	4.9

É necessário então, realizar uma filtragem dos valores calculados. O filtro utilizado tem os seguintes parâmetros:

1	2	3	4	3	2	1
2	5	6	7	6	5	2
3	6	8	12	8	6	3
4	7	12	20	12	7	4
3	6	8	12	8	6	3
2	5	6	7	6	5	2
1	2	3	4	3	2	1

As próximas imagens (fig. 2.12) representam as frequências calculadas antes da interpolação, após interpolação mas sem filtragem e após filtragem. Pontos brancos têm valor de frequência –1. Quanto mais claro for o ponto, maior sua frequência.

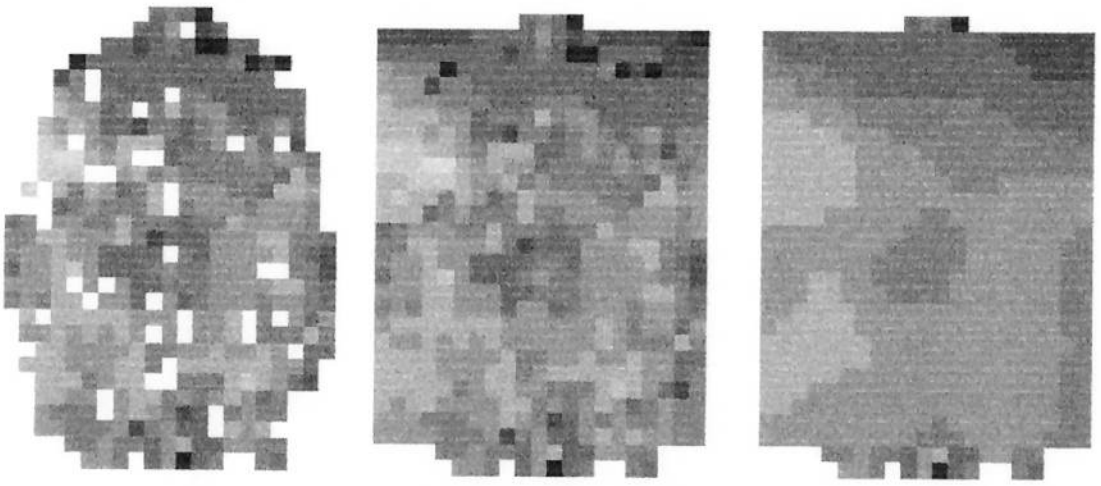


Figura 2.12 – Frequências (a) Antes da Interpolação (b) Após Interpolação (c) Após Filtragem

2.2.2.4. Filtragem

O próximo passo no processo de Enhancement é a aplicação dos filtros de Gabor [1,2,3]. Esse tipo de filtro depende da orientação e frequências locais, motivo pelo qual elas foram calculadas.

A idéia básica por trás desses filtros é a seguinte: Sabendo-se a orientação local em uma região, divide-se a mesma em faixas paralelas, no sentido da orientação. Soma-se as intensidades nessas faixas, de forma ponderada. A equação que pondera essa faixa é:

$$exp(-\frac{EO^2}{2\sigma^2})$$

EO é a posição de um ponto da faixa, na direção de orientação;

σ é uma constante Gaussiana. (σ=4, valor escolhido arbitrariamente).

Então, devemos somar os valores obtidos para cada faixa, também de forma ponderada. Nesse caso, o ponderação é feita por:

$$cos(2\pi fEF)$$

EF é a posição da faixa na direção da frequência, isto é, na direção normal à orientação.

A figura abaixo (fig. 2.13) ilustra essas variáveis e as equações de ponderação.

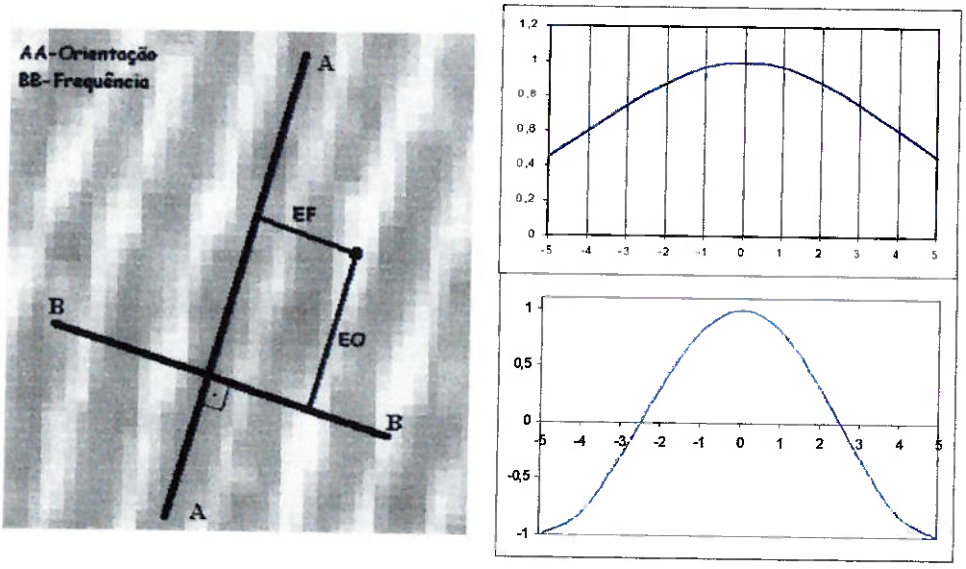


Figura 2.13 – Eixos EO e EF e Curvas de Ponderação

O resultado final dessas operações é o que chamamos de filtros de Gabor.

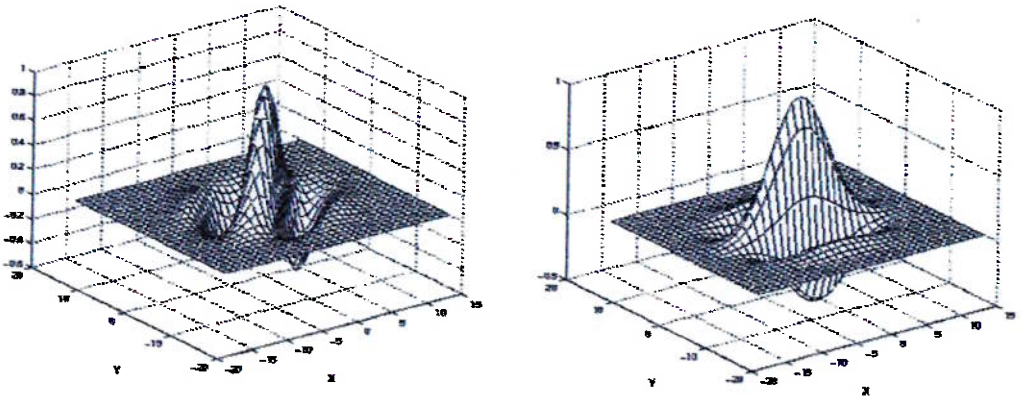


Figura 2.14 – Filtros de Gabor

O grande problema desse filtro é que ele não retorna um valor dentro da faixa de interesse (0 a 255). Isso se deve ao fato de que nesse filtro alguns coeficientes assumem valores negativos, o que inviabilizaria dividir a somatória ponderada pela somatória de pesos. Porém, algumas associações podem ser feitas, e a partir delas bolar um método para adequar os valores obtidos à faixa de interesse. Quando o centro do filtro corresponde ao centro de uma crista, as faixas centrais têm somatória baixa e as laterais, alta. Mas como o peso das faixas laterais é negativo, o valor da somatória é um valor negativo. Para o caso contrário, com o centro do filtro coincidindo com o centro de um vale, a somatória assume um valor positivo. As outras posições assumem valores intermediários. Então, relacionando o valor máximo encontrado com o valor 255, e o mínimo com 0, obtemos uma transformação linear entre os valores da somatória e os da faixa de interesse.

Surgem, entretanto, dois problemas. Se o filtro for aplicado em uma região da imagem onde não há impressão, ou seja, uma região branca, o valor encontrado para a somatória será aleatório, fora do esperado. Pode ser que o valor seja muito maior que o máximo encontrado na impressão, o que acarretaria num sério problema: esse novo máximo seria associado ao valor 255, e os pixels brancos da impressão, com valor de somatória bem menor, tenderiam a valores baixos, e seriam confundidos com pixels pretos. Para solucionar esse problema, basta garantir a aplicação do filtro somente numa região retangular contida na impressão digital.

O outro problema é um pouco mais grave, e necessitou uma maior elaboração para ser resolvido. O filtro deve ter tamanho constante, ou seja, ter sempre o mesmo número de faixas (11) e mesmo número de pixels por faixa (11). Mas

quando esse filtro é aplicado em regiões de frequências diferentes, o valor da somatória obtido, para uma mesma condição (por exemplo, aplicação no centro da crista), é muito diferente. Contudo, esse valor deveria ser igual ou muito próximo, pois se trata de uma mesma condição. Nesse caso, centros de cristas podem se tornar pixels pretos (como deveriam ser), cinzas ou até brancos. O esquema abaixo (fig. 2.15) ilustra a causa desse problema.

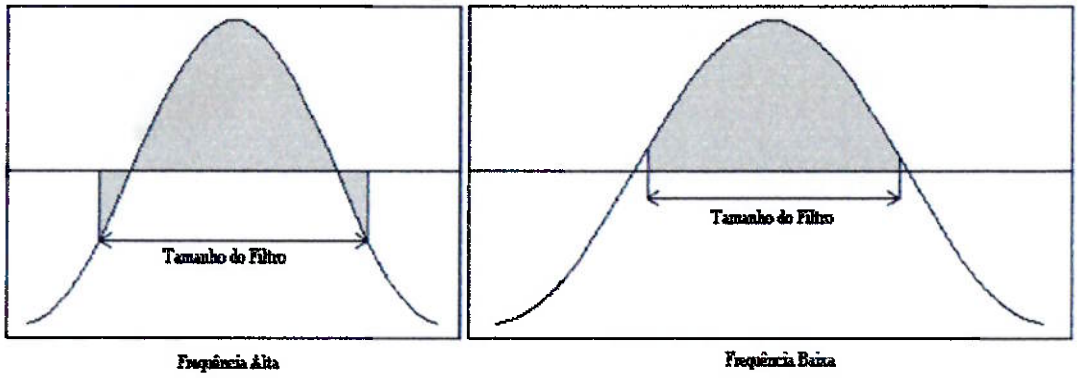


Figura 2.15 – Esquema do Problema devido à diferença de Frequências

Seria necessário que o filtro abrangesse sempre uma mesma faixa, independente da frequência. A faixa escolhida é de um período. Contudo, como não podemos alterar o número de faixas conforme muda a frequência, é preciso espaçá-las para adequá-las a cada caso.

Uma vez corrigidos esses dois problemas, o resultado (imagem melhorada) se mostrou muito bom, como ilustra a figura 2.16.

Figura 2.16 – Imagens Originais e Melhoradas

2.3. Thinning

A partir da imagem melhorada, o processo de afinamento (thinning) pode ser realizado sem que ocorram muitos problemas.

O processo de afinamento é baseado nas hipóteses de que as intensidades das cristas e vales formam uma onda senoidal na direção normal à orientação e de que os pixels centrais de uma crista são os de menor intensidade, ou seja, são os mais negros [1]. A idéia do método é, então, encontrar esses pixels centrais e apagar todo o resto da imagem. Dessa forma, poderíamos representar a impressão digital por cristas de espessura 1 pixel, o que tornaria muito simples a extração das características.

Embora simples, o método não é perfeito, tampouco o é a impressão melhorada. Portanto, algumas correções devem ser feitas para melhorar a imagem afinada.

Nas regiões onde a orientação é próxima de 45° ou 135°, as cristas apresentam espessura dupla, como ilustra a figura 2.17. É necessário, então, um simples algoritmo para remover os pixels indesejáveis.

Figura 2.17 – Problema no Afinamento

O outro problema consiste no aparecimento de muitos pixels brancos "cortando" as cristas afinadas. Esses espaços não significam que ali é um final de crista, e sim um pequeno defeito, mas muito comum. Para evitar o surgimento de muitas minúcias aleatórias, criei um algoritmo que detecta esses pequenos defeitos e os corrige, ou seja, une novamente os dois pedaços de crista e apaga pixels indesejáveis que venham a aparecer.

Figura 2.18 – Problemas No Afinamento

2.4. Extração de Características

Por fim, chegamos na extração de características em si. O processo novamente é simples, mas devemos ficar atentos com os possíveis (e são muitos) problemas.

Para detectar as características basta varrer a imagem afinada, e para cada pixel de crista, averiguar o número de pixels vizinhos. Um final de crista só tem um vizinho, uma bifurcação tem três e um pixel normal tem dois.

Embora a primeira extração seja banal, ela contém muitos prováveis erros, que listo, juntamente com a solução proposta, em seguida:

1) *Bifurcações muito próximas:* Muitas vezes, o que seria uma única bifurcação é representada por mais de uma, como ilustra a figura 2.19. A solução é reduzir a apenas uma um conjunto de bifurcações adjacentes.

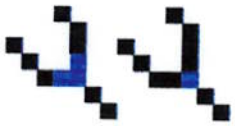


Figura 2.19 – Bifurcações Próximas

2) *Bifurcação muito próxima de um final de crista:* Defeitos no processo podem gerar "falsas bifurcações". Essas são facilmente detectadas pois um de seus ramos tem tamanho muito pequeno (uma vez que há um final de crista próximo). A solução é desconsiderar ambas as minúcias (ver figura 2.20).

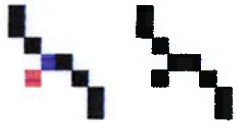


Figura 2.20 – Bifurcação e Final

3) *Dois finais de crista muito próximos, isolados de outras minúcias:* Embora o processo de afinamento elimine lacunas de tamanho 1 pixel em uma crista, quando o espaço em branco é um pouco maior (2 ou mais pixels), o processo de extração assume que encontrou duas minúcias, o que é errado. A solução é novamente desconsiderar ambas.

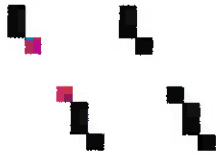


Figura 2.21 – Dois Finais

4) *Três finais de crista muito próximos, isolados de outras minúcias:* Algumas vezes uma bifurcação fica apagada, mas os seus três ramos tem finais próximos. Para corrigir esse problema, basta substituir as três minúcias por uma só, posicionada no centro geométrico delas.

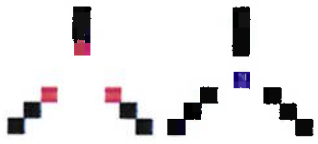


Figura 2.22 – Três Finais

5) *Várias cristas numa mesma região*: É muito difícil dizer o que causa esse tipo de erro, e é mais difícil ainda de corrigi-lo. Adotei solução análoga ao caso anterior, substituindo todas por uma só. É claro que podemos estar perdendo minúcias genuínas, mas é mais provável que estejamos eliminando algumas falsas.

2.5. Comparação com o Banco de Dados

Visualmente, a extração de características apresenta um bom resultado. Comparando duas impressões distintas de uma mesma pessoa, verificamos que a grande maioria das minúcias coincidem, como observamos nas figuras 2.23 e 2.24.

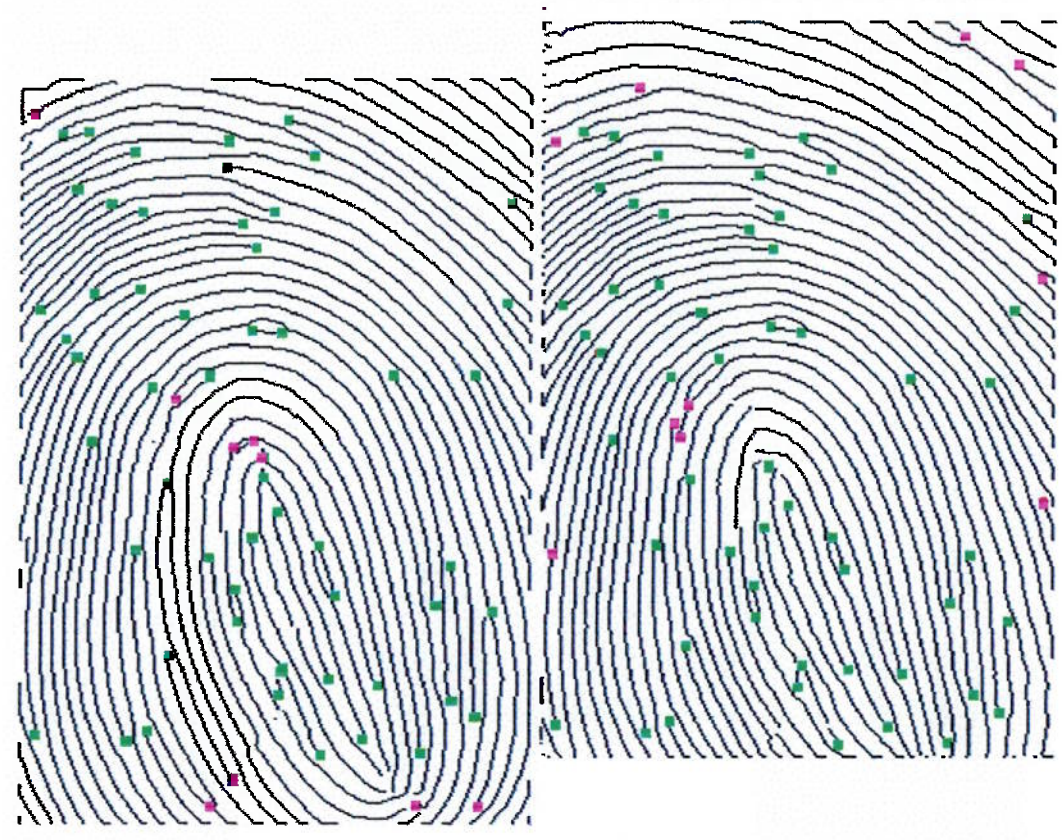


Figura 2.23 – Comparação entre as Minúcias de Duas Impressões Digitais

A grande dificuldade, entretanto, é desenvolver um algoritmo que perceba essa semelhança. Como a posição da impressão digital dentro da imagem varia de

coleta para coleta, uma comparação direta das posições seria inviável. Portanto, temos somente outras duas maneiras possíveis para efetuar a comparação: utilizar um ponto comum de referência ou se basear na posição relativa das minúcias entre si.

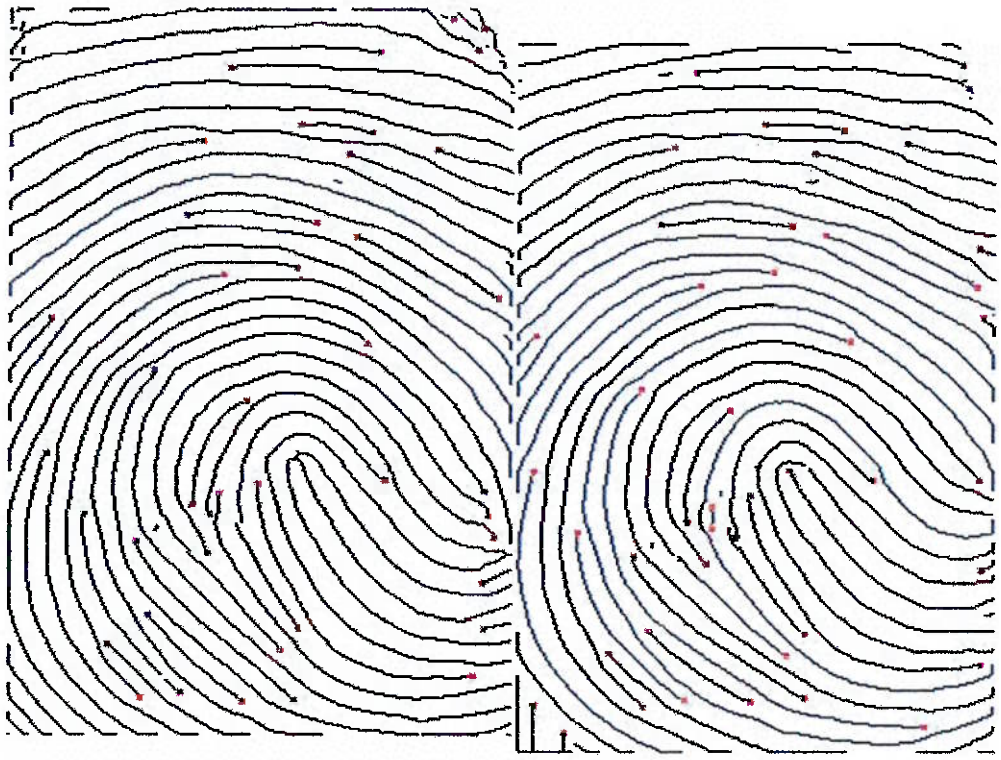


Figura 2.24 – Comparação entre as Minúcias de Duas Impressões Digitais

A primeira opção seria computacionalmente mais simples. Os dados de posição e orientação das minúcias, em relação a um ponto de referência seriam armazenados no banco de dados e a comparação seria direta (minúcia a minúcia). Porém, esse método depende muito de encontrar um ponto de referência confiável. A idéia inicial era utilizar os pontos singulares da impressão (por exemplo, o ponto *core*). Mas a localização desse ponto está sujeita a erros de razoável grandeza. Somado aos erros de localização das minúcias, teríamos uma imprecisão muito grande, e para vencer essa

imprecisão, deveríamos adotar uma tolerância também bastante grande. Com isso, seria muito comum confundir pessoas, e o programa perderia muito em robustez.

A solução, então, estaria em desenvolver um método para relacionar as minúcias entre si. Uma opção seria escolher uma minúcia aleatoriamente, e ligá-la a minúcia mais próxima, e assim sucessivamente até fechar o ciclo. Comparar-se-ia o tamanho do caminho obtido e outras propriedades, como ângulos dos trechos, etc... Nesse caso, uma minúcia falsa em uma impressão poderia desviar completamente o caminho original, o que certamente inviabiliza esse procedimento. Outra opção seria escolher uma minúcia para ser o ponto de referência. É lógico que seria muito difícil escolher a mesma minúcia nas duas imagens que estão sendo comparadas. Poderíamos então utilizar mais de um ponto de referência, e se em alguma combinação de referências o "encaixe" das minúcias fosse adequado, a comparação seria positiva. Os problemas nesse caso são que o processo seria computacionalmente exaustivo, e que, embora visualmente as minúcias pareçam equivalentes, as distâncias entre elas variam razoavelmente (devido à deformação linear na coleta da digital), e voltaríamos na questão das grandes tolerâncias.

Embora não tenha desenvolvido um algoritmo que resolva essa questão, acredito que a resposta esteja em encontrar padrões nas minúcias. Por exemplo, olhando a figura 2.23, poderíamos detectar um quadrilátero de minúcias na região superior central, e assim por diante.

3. Método de Redes Neurais

3.1.Introdução

Para realizar o reconhecimento das impressões digitais vai-se utilizar redes neurais. As redes neurais são uma espécie de esquema que tenta imitar o funcionamento do cérebro humano. O cérebro humano é um computador com uma complexidade imensa, não possui linearidade e possui processamento paralelo [4].

Desde o nascimento o cérebro possui a capacidade de educar os neurônios, criando o que se pode chamar de "experiência". Desse mesmo modo funcionam as redes neurais elas são programadas para adquirir essa "experiência" durante um processo de treinamento de tal forma que para uma determinada entrada se tenha uma determinada saída desejada.

Essa "experiência" fica guardada nas sinápses, no caso do cérebro, e nos pesos existentes entre as unidades de processamentos, no caso da rede neural (neurônios).

O processo para treinamento dos pesos sinápticos é chamado de algoritmo de treinamento, e ele deve gerar uma matriz de pesos de tal forma que a rede sempre forneça uma saída satisfatória dado uma certa entrada.

Utilizar as redes neurais nos dá algumas vantagens, entre elas [4]:

- ☺ Capacidade de análise de sistemas não lineares;
- ☺ Mapeamento das entradas e saídas num processo de aprendizagem;
- ☺ Adaptabilidade a variações no sistema;
- ☺ Robustez do sistema(resistência a erros);
- ☺ Processamento paralelo simultâneo;

As redes neurais podem ser configuradas de várias formas, mas todas possuem algumas características em comum, essas características serão resumidas à seguir.

A redes neurais são compostas por neurônios e pesos, fazendo um paralelismo com o funcionamento dos neurônios biológicos.

Esses neurônios recebem um conjunto de entradas multiplicadas pelos respectivos pesos, essas entradas já ponderados são somadas para gerar a saída "primaria" do neurônio, essa saída servirá de entrada para a função de ativação da rede. O esquema a seguir demonstra esse funcionamento:

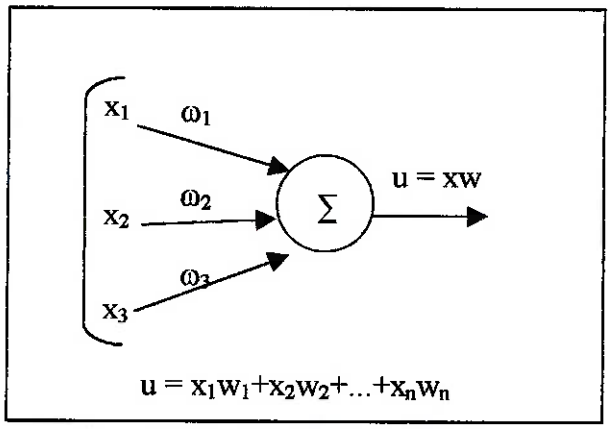


Figura 3.1 : Neurônio Artificial

Os demais detalhes sobre a rede neural em si e seu processo de treinamento serão dados nas seções posteriores.

Antes de falarmos da rede neural em si vamos falar um pouco dos pré processamentos necessários para que a rede possa trabalhar com a imagem da impressão digital e faça sobre ela qualquer tipo de analise, esses pré processamentos são descritos a seguir.

3.2. Processo de Classificação com Redes Neurais

A idéia é fazer com que a rede neural primeiro classifica as impressões digitais , elas serão divididas em cinco classes Arch, Tended Arch, Left Arch, Right Loop and Whorl, após essa classificação os valores gerados pela rede de verão ser usados para fazer o reconhecimento em si. A figura 1 abaixo mostra as cinco possíveis classes [3].

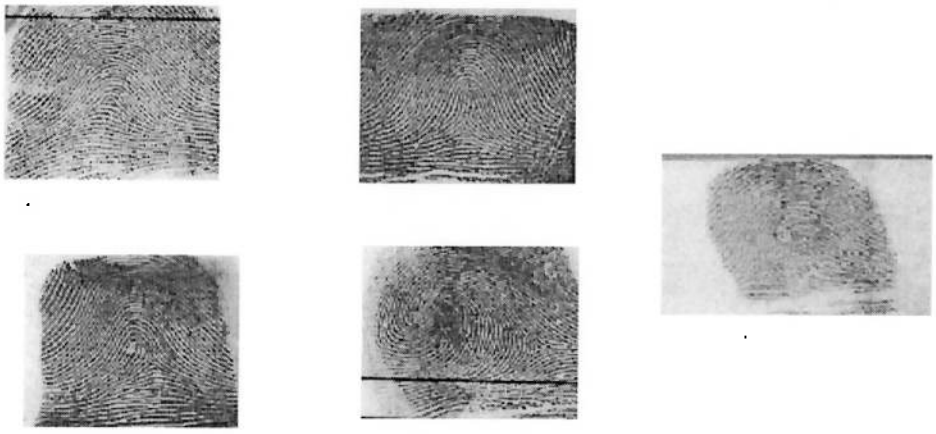


Figura 3.2: Classes de Impressões

Diferentemente do outro método mostrado nesse trabalho, aqui não vamos comparar as minúcias das impressões, ou seja, comparar fins e bifurcações de cristas, o processo aqui será diferente. O processo de classificação das impressões está representado na figura 2, onde os componentes ovais representam a entrada e a saída, os retângulos representam os componentes de processamento e as flechas o fluxo de informação. Os componentes não representam necessariamente a programas separados, eles são meramente

uma separação em blocos conceituais. A imagem da impressão digital é uma matriz de 512 x 512 e 8-bits de graus de cinza [5].

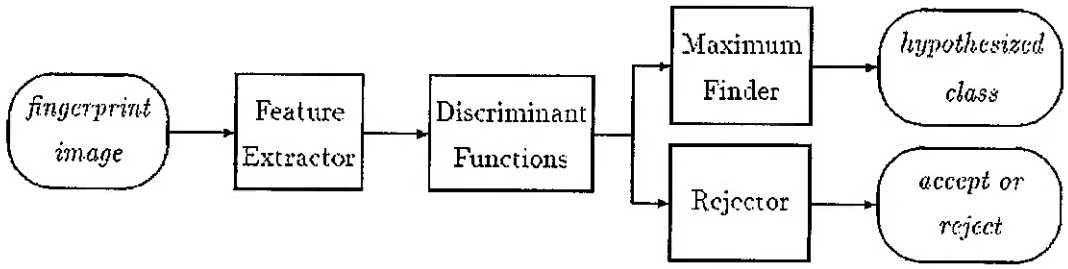


Figura 3.3: Seqüência Conceituai de Passos

O componente de "Extração de Características" reduz a digital de 262.144 bytes em um vetor de características, uma pequena lista de números que ocupa apenas 448 bytes. Este vetor de características tem como pretensão representar, de uma forma compacta, as características relevantes para a classificação das impressões.

O próximo componente é o "Banco de Funções Discriminantes", existe cinco funções discriminantes, uma para cada classe. Cada uma produz um número real, o qual tende a ser grande se a impressão digital for a correspondente daquela função.

Os cincos valores produzidos pelas funções discriminantes são mandados para os dois componentes finais, o "Encontrar o Máximo" e o "Rejeitor". O Encontrar

o Máximo meramente acha o maior dos valores que foi gerado pelas funções e diz a partir desse valor qual seria a classe da impressão digital. O componente Rejeitor serve com avaliador se essa classificação é válida ou não, ele possui um erro permitido, se o erro entre o valor máximo gerado e o valor máximo de referência for maior que esse error base a classificação é rejeitada. Essa rejeição significa que o sistema de classificação se recusa a determinar uma classe, porque ele não pode ter a certeza suficiente para fazer o mesmo. Agora vamos descrever um pouco sobre cada um dos componentes desses processos.

3.3. Extrator de Características

Esse componente recebe como entrada um pequeno vetor numérico de características, esse vetor é gerado a partir da impressão digital da seguinte forma.

3.3.1. Campo de Orientação

Agora vamos calcular o campo de orientação da impressão digital, este campo de orientação é calculado baseado em um método chamado de "cristas-vales", esse método baseia-se em um binarizador que trabalha da seguinte forma. Para cada pixel C, a soma dos slits s_i, i=1...8, são calculados, onde cada s_i é a soma dos valores de todos os pixels nomeados i na figura 4. O binarizador converte os níveis de cinza em preto e branco através da combinação dos métodos de "Threshoulding Local" e "Comparação de Slit" [5].

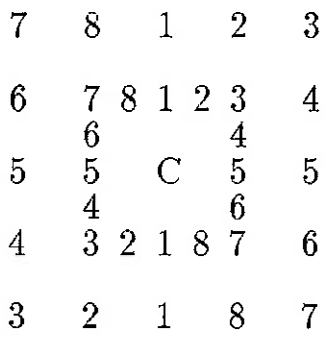


Figura 3.4 : Numeração dos Pontos vizinhos

A formula do "Threshoulding Local" coloca o pixel para branco quando a entrada C excede a média das intensidades dos pontos pertencentes aos slits, logo a formula fica:

$$C > \frac{1}{32}\sum_{i=1}^{8} s_i$$

A fórmula de 'Comparação dos Slits ' coloca o pixel para branco se a média do mínimo e máximo slits excede a soma de todos os slits, logo a fórmula fica:

$$\frac{1}{2}(s_{\min} + s_{\max}) > \frac{1}{8}\sum_{i=1}^{8} s_i$$

Combinando as duas equações chega-se a equação que se segue:

$$4C + s_{\max} + s_{\min} > \frac{3}{8}\sum_{i=1}^{8} s_i$$

Para encontrar a direção das cristas e vales é apenas uma extensão do binarizador. Após binarizar, a direção da crista em ponto individual é considerado como a direção que os slits escolhem de uma maneira natural, ou seja, da menor soma de slits para o caso daqueles pontos binarizados para

preto e da máxima soma para os pontos binarizados para branco. Para gerar um melhor efeito no calculo, calcula-se as direções em janelas de 16x16, ou seja, na verdade uma média das direções naquela região.

O angulo da crista θ em uma localização é definido como sendo 0° se a crista for horizontal, aumentando até 180° a medida que se gira no sentido horário e revertendo para zero quando a crista se torna horizontal novamente ($0^{\circ} < \theta < 180^{\circ}$). Quando a direção de uma crista em pixel é calculada, na verdade não se tem o ângulo θ mas sim um vetor de direções ($\cos 2\theta, \operatorname{sen} 2\theta$). Calcular a média dos ângulos pode produzir resultados absurdos: a média entre 1° e 179°, cada um próximo da horizontal, produz um angulo de 90°, que não é um resultado nada aceitável. Quando se faz a média de senos e cosenos também não se obtém bons resultados, logo por isso utilizamos para calcular as médias os senos e cosenos de 2θ.

Calculando-se as direções em janelas de 16x16, gera-se 840 direções que é na verdade a saída desse programa, ou seja, o campo de orientação em si. A figura 5 mostra os maiores 112 autovalores, sua rápida queda nesse gráfico mostra que a informação gerada por essa transformada são concentradas nas primeiras características.

3.4. Classificador

Agora daremos uma breve introdução do classificador em si. O classificador a ser utilizado será uma rede neural Feedforward do tipo MLP(Multi-Layer Pecption) com três camadas (contando a entrada como uma camada). Se faz conveniente aqui definir a seguinte notação [5]:

$N^{(i)}$ = número de nós na camada i^{th}(i =0,1,2)

$f(x) = 1/(1+e^{-x})$ = função sigmoid

$b_i^{(k)}$ = peso bias do nó i^{th} da camada k^{th}

$w_{ij}^{(k)}$ = peso da conexão do nó i^{th} da camada k^{th} ao nó j^{th} da camada (k-1)th (k=1,2;$1 \le i \le N^{(k-1)}$)

$x = (x_1,....,x_n)^T$ = vetor de características

Vale observar que

$N^{(0)}$ = número de nós de entrada = n = número de características

$N^{(1)}$ = números de nós ocultos;

$N^{(2)}$ = números de nós de saída = N = número de classes.

Daí sai que a função discriminante que representa essa rede fica:

$$D_i(x) = f(b_i^{(2)} + \sum_{j=1}^{N^{(1)}} w_{ij}^{(2)} f(b_j^{(1)} + \sum_{k=1}^{N(0)} w_{jk}^{(1)} x_k$$

Para treinar os pesos da rede o procedimento a ser usada não foi determinado ainda,

visto que ele é de suma importancia para a boa resposta da rede e deve ser otimizado para que não leve muito tempo de processamento.

3.5. Funções de Ativação

Depois que foi dado na seção anterior uma breve introdução sobre que tipo de rede será utilizada vamos entrar em mais detalhes sobre a rede neural em si e seus componentes [7].

Chama-se função de Ativação a função que pega a saída u gerada pelo neurônio e normaliza ela, a saída dessa função (y) será considerada a saída do neurônio.

Essa função pode ser por exemplo uma função linear do tipo:

$$y = \varphi(u) = k \cdot u$$

onde k é uma constante ou uma função *threshold* do tipo,

$$y = \varphi(u) = \begin{cases} 1 & \text{se u} > \text{x} \\ 0 & \text{se u} < \text{x} \end{cases}$$

onde x é um *threshold* .

Porém esses tipos de funções não possuem respostas satisfatórias, quando se trata de utiliza-las em redes neurais, o problema com essas funções é que não possuem um limite, ou seja elas variam de acordo com a variação de u, o que não é bom pois pode ir se acumulando com as interações até que sature a rede.

Logo o ideal é utilizar uma função que limite o valor da saída y. A função que normalmente é utilizada é função sigmoidal:

$$y = \varphi(u) = \frac{1}{1+e^{-u}} \qquad (1)$$

o gráfico dessa função é o seguinte:

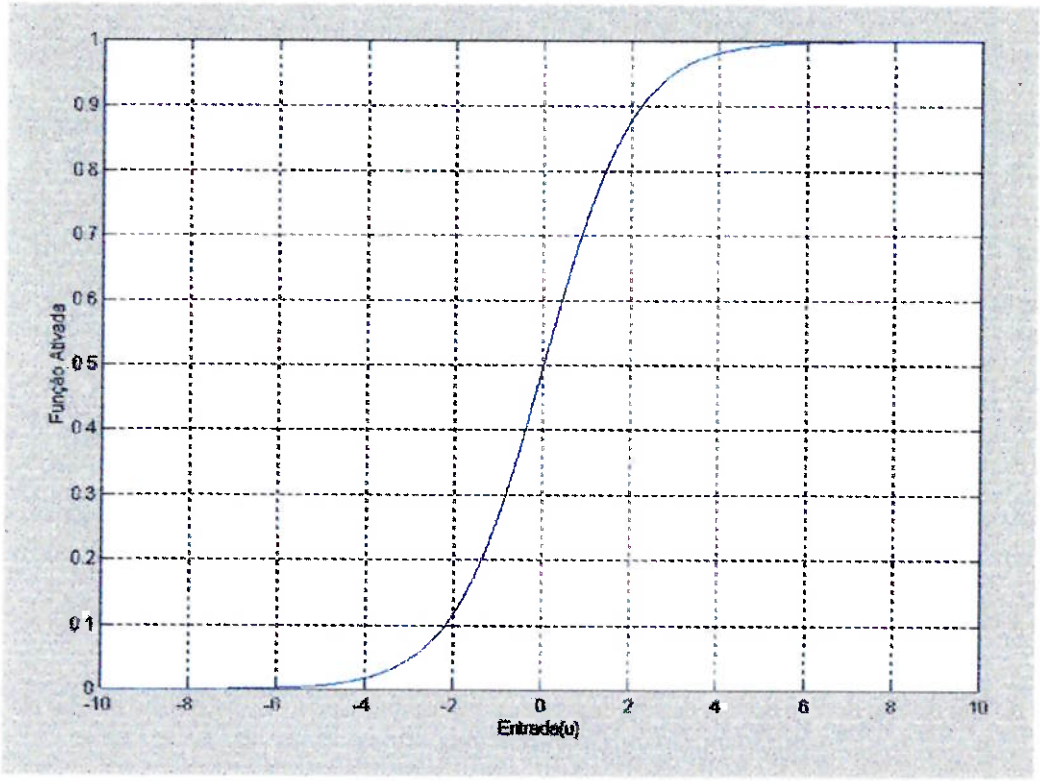


Figura 3.5: Gráfico da função de ativação do tipo sigmaoidal

Como o gráfico pode mostra a saída dessa função varia de 0 à 1, para qualquer faixa de variação da entrada, logo ela pode ser utilizada em uma rede porque não vai acumular os valores de iteração a medida que o processamento

avança. Outro ponto importante que essa função tem, e faz com que ela seja ideal para utilização nesses casos, é que para valores próximos de zero da entrada tem-se valores de saída diferentes de zero.

Logo o esquema do neurônio fica agora:

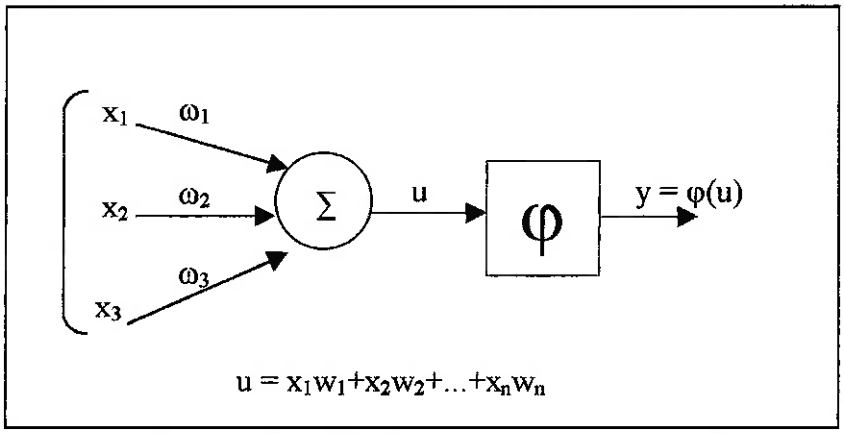


Figura 3.6: Neurônio Artificial com a Função de Ativação

3.6. Número de Camadas

3.6.1 Redes de Camada Única

Esse tipo de rede é mais simples, é composta apenas de nós de entrada , que distribuem as mesmas, e de neurônios de saída. Vale aqui comentar que os nós de entrada não são neurônios em si. Assim como mostra o esquema a seguir [7].

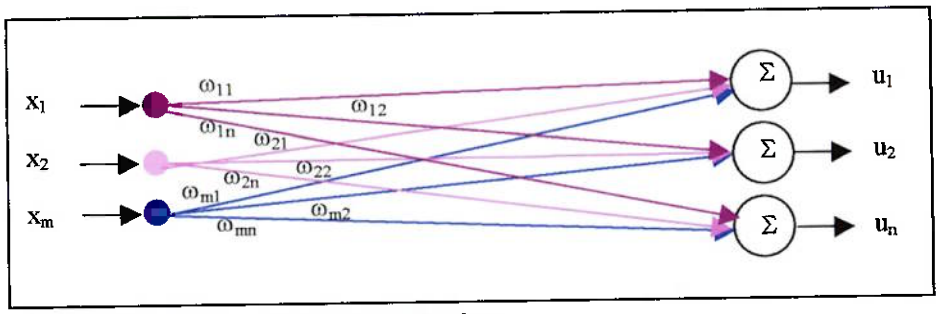


Figura 3.7: Rede de Camada Simples

Para esse caso os neurônios recebem os valores das entradas multiplicadas pelos respectivos pesos e as soma, o valor dessa soma é então jogada na função de ativação para gerar a saída da rede. Os pesos formam uma matriz m x n, onde m é o número de entradas e n é número de neurônios.

3.6.2 Redes Neurais de Camadas Múltiplas

Nesse tipo de rede existem pelo menos duas camadas de neurônios, por isso são mais complexas e geralmente possuem maiores capacidades computacionais. Para caso desse trabalho será usada uma rede desse tipo, com duas camadas de neurônios, sendo uma delas escondida..

Um esquema da rede que vai ser utilizada é mostrada no esquema a seguir, como mostra a figura ela terá 1024 entradas e apenas cinco neurônios de saída, sendo que a camada interna terá os seguintes números possíveis 15,20 e 25.

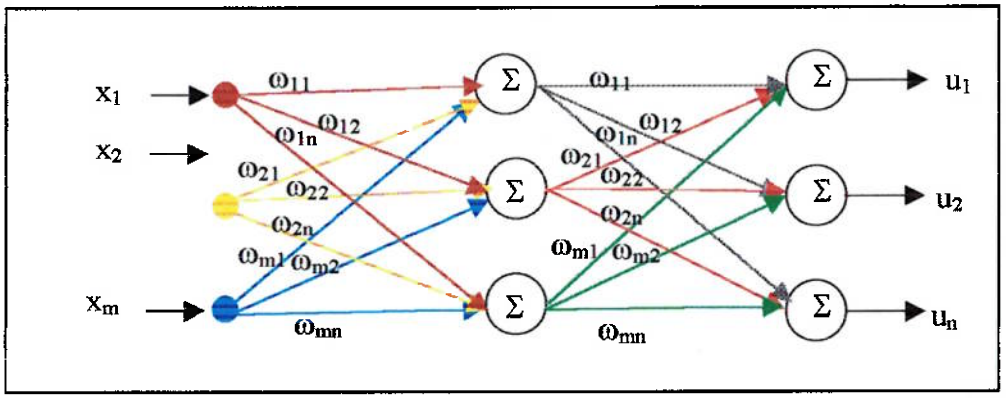


Figura 3.8: Rede de Camada Múltiplas

Já que a saída de uma camada é a entrada de outra , como pode ser visto no esquema, a função de ativação está presente entre duas camadas e essa é toda diferença entre esse tipo de rede e o de camada simples como pode ser provado facilmente.

Caso não houvesse a função de ativação a saída da rede seria calculada multiplicando as entradas por duas matrizes de pesos seqüencialmente, ou seja a saída seria:

$$y = (uw_1)w_2;$$

Como a multiplicação de matrizes é associativa é equação acima pode ser escrita da seguinte forma:

$$y = u(w_1w_2);$$

Logo seria como se as entradas tivessem passadas por uma rede de camada simples, por tanto a função de Ativação é que faz a diferença entre os dois tipos.

3.7. Treinamento de Redes Neurais

Agora vai-se falar na característica mais importante das redes neurais artificiais, a capacidade de aprendizagem. Esse processo é realizado através do treinamento dos pesos, com já mencionado anteriormente.

O treinamento consiste de uma seqüência de passos [6]:

1. Fornece-se o vetor de entradas para a rede e faz-se com que ela calcula a saída;

2. Essa saída é comparada com a saída desejada e gera-se um erro que será propagado para trás, recalculando-se os pesos de forma que a saída passe a ser a desejada.;

3. Repete-se esse processo até que os pesos estejam de acordo, para o calculo como desejado.

3.7.1 Treinamento Backpropagation

O treinamento de redes neurais multicamadas foi durante muito tempo um problema, porque não havia um processo sistemático para o treinamento, mas como as redes de camadas simples começaram a se mostrar limitadas, teve-se que se desenvolver um algoritmo que realizasse essa tarefa, daí surgiu o Backpropagation [6].

Vai-se falar do método desenvolvido por Rumelhart, Hilton e Willians (1986). Esse método tenta minimizar o erro entre o vetor de saída gerado pela rede o vetor de saída desejado.

A rede é inicializada com pesos randômicos de valores pequenos, como a nossa rede vai usar a função de ativação sigmoidal, a equação para calculo da saída final de cada neurônio, com exceção da camada de entrada que apenas passa o vetor de entrada, multiplicado pelos respectivos erros, para a primeira camada, fica sendo:

$$Y_j(\omega,h,u) = \frac{1}{1+\exp\{-(\Sigma_{i=1}^{N}(\omega_{ji}y_i)+h_j)\}} \qquad (2)$$

onde ω_{ji} entre o neurônio j, para qual está sendo calculada a saída, e o neurônio i da camada anterior, N é o número total de neurônios da camada anterior, y_i é a saída ativada do neurônio i da camada anterior, h_j é o chamado bias da unidade j, ele pode ser entendido como um outro peso e para o nosso caso será considerado como zero.

3.7.2 Ajuste dos Pesos da Camada de Saída

Primeiramente deve-se calcular o erro entre a saída desejada e a saída gerada pela rede, que fica:

$$e_j = d_j - y_j \qquad (3)$$

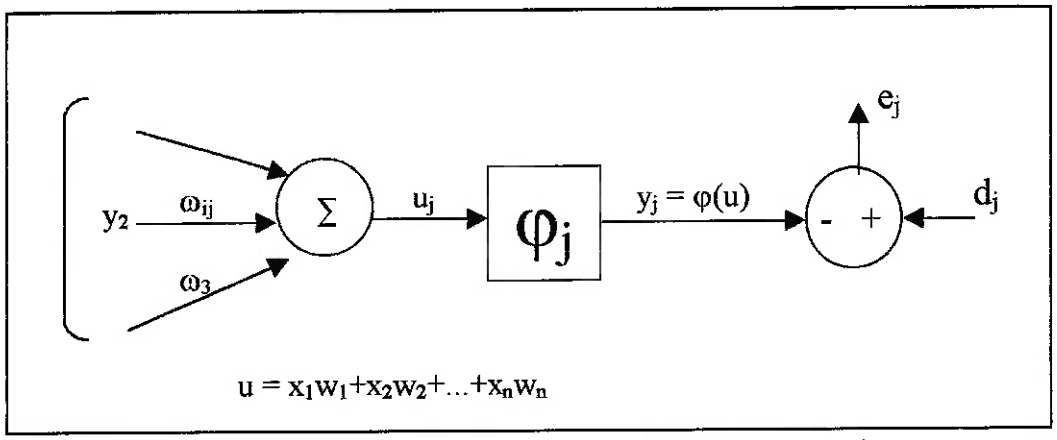


Figura 3.9: Esquema para o Ajuste de Pesos da Camada de Saída

Agora deve-se definir o erro quadrático global, que nada mais é do que a soma dos erros quadráticos de cada neurônio da camada de saída, ou seja:

$$\xi = \frac{1}{2}\sum_j e_j^{\,2} \qquad (4)$$

A atividade interna do neurônio antes ser ativado é :

$$u_j = \sum_i \omega_{ij} \cdot y_i \qquad (5)$$

De acordo com a função de ativação sabe-se que:

$$y_j = \varphi(u_j)$$

O que o algoritmo vai fazer é calcular uma correção $\Delta\omega_{ij}$ para cada ω_{ij} de tal forma que o novo peso, depois de corrigido, será:

$$\omega_{ij}(n+1) = \omega_{ij}(n) + \Delta\omega_{ij}$$

onde n é o índice da alteração que está sendo feita.

Porém sabe-se que a correção $\Delta\omega_{ij}$ é proporcional ao gradiente (regra delta), logo tem-se:

$$\Delta\omega_{ij} = \eta \cdot \frac{\partial\xi}{\partial\omega_{ij}}; \qquad (6)$$

A constante η determina a taxa de aprendizagem e chamada de coeficiente de aprendizagem.

Logo tem-se que calcular o valor do gradiente, esse gradiente pode ser expresso através da seguinte expressão:

$$\frac{\partial \xi}{\partial \omega_{ij}} = \frac{\partial \xi}{\partial e_j} \cdot \frac{\partial \xi}{\partial y_j} \cdot \frac{\partial \xi}{\partial u_j} \cdot \frac{\partial \xi}{\partial \omega_{ij}} \qquad (7)$$

Agora vai-se calcular cada um dos componentes desse gradiente. O primeiro deles pode ser deduzido a partir da equação (4) e fica:

$$\frac{\partial \xi}{\partial e_j} = e_j; \qquad (8)$$

O segundo componente pode ser deduzido a partir da equação (3) e (4):

$$\frac{\partial e_j}{\partial y_j} = -1 \qquad (9)$$

Derivando-se a equação da Função da Ativação chega-se na equação para o terceiro componente:

$$\frac{\partial y_j}{\partial u_j} = \varphi_j'(u_j); \qquad (10)$$

Derivando-se a equação (5) chega-se na equação do quarto componente:

$$\frac{\partial u_j}{\partial \omega_{ij}} = y_i \qquad (11)$$

Agora substituindo na equação (7) a equação dos componentes encontradas, tem-se:

$$\frac{\partial \xi}{\partial \omega_{ij}} = -e_j \cdot \varphi_j'(u_j) \cdot y_i \qquad (12)$$

substituindo agora na equação (6) do gradiente, pode encontrar uma equação para a correção:

$$\Delta \omega_{ij} = -e_j \cdot \varphi_j'(u_j) \cdot y_i \qquad (13)$$

A função de ativação que vai-se usar na rede a sigmoidal, logo:

$$\varphi_j'(u_j) = y_j(1 - y_j); \qquad (14)$$

Definindo δ_j como um gradiente local igual a:

$$\delta_j = e_j \cdot y_j(1 - y_j); \qquad (15)$$

Logo substituindo essa equação na equação (13) fica:

$$\Delta\omega_{ij} = -\eta\delta_{ij} y_i \qquad (16)$$

3.7.3 Treinamento dos Pesos da Camada Escondida

O problema de treinamento dos pesos dessa camada é que ela não possui informações sobre os erros da saída, logo não se pode fazer o mesmo processo do caso anterior. Logo o método de Backpropagation vai propagar o erro da camada de saída para essa camada de tal forma a calcular os novos pesos. Agora vai-se trabalhar em cima do seguinte esquema [6]:

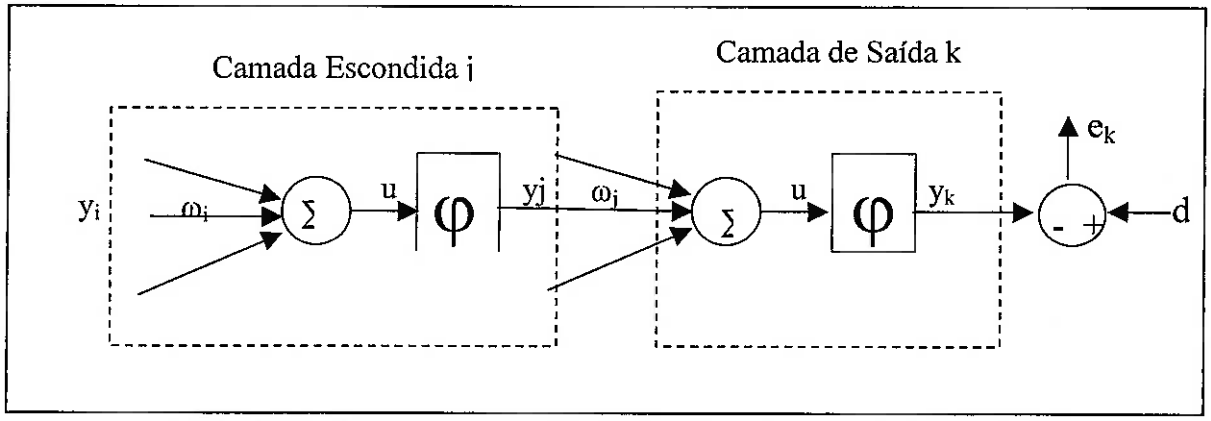


Figura 3.10: Esquema para o Ajuste de Pesos da Camada Escondida

O gradiente local para um neurônio escondido j é dado por :

$$\delta_j = -\frac{\partial \xi}{\partial y_j} \cdot \frac{\partial y_j}{\partial u_j} \qquad (17)$$

Que também pode ser escrita da seguinte forma:

$$\delta_j = -\frac{\partial \xi}{\partial y_j} \cdot \varphi'_j(u_j) \qquad (18)$$

Substituindo o índice novo, de acordo com o esquema da figura 6, na equação do erro quadrático (4), tem-se:

$$\xi = \frac{1}{2}\sum_k e_k^{\,2} \qquad (20)$$

Derivando-se essa equação tem-se:

$$\frac{\partial \xi}{\partial y_j} = \sum_k e_k \cdot \frac{\partial e_k}{\partial u_k} \cdot \frac{\partial u_k}{\partial y_j} \qquad (21)$$

Observando a figura 6, pode-se notar que:

$$e_k = d_k - y_k = d_k - \varphi_k(u_k); \qquad (22)$$

Daí deduz-se que:

$$\frac{\partial e_k}{\partial u_k} = -\varphi_k'(u_k) \qquad (23)$$

Mudando os índices da equação de atividade interna (5) para k, fica:

$$u_k = \sum_j \omega_{jk} \cdot y_k \qquad (24)$$

Derivando-se novamente essa equação, tem-se:

$$\frac{\partial u_k}{\partial y_j} = \omega_{ij} \qquad (25)$$

Substituindo-se as equações (25),(24) e (23) na equação (21) e depois substituindo essa equação na equação (18) chega-se à:

$$\delta_j = \varphi'_j(u_j)\sum_k \delta_k \cdot \omega_{jk} \qquad (26)$$

Logo pode-se resumir o processo de treinamento como sendo baseado em dois pontos básicos:

1. A correção $\Delta\omega_{ij}$ aplicada ao peso que conecta o neurônio i com o neurônio j, sendo definido pela Regra Delta;
2. Correção do erro = Coeficiente de Aprendizagem x Gradiente Local Sinal de Entrada

O gradiente Local δ_j vai depender da camada em que estamos recalculando os pesos

- Se o neurônio de analise (j) for da camada de saída, δ_j é igual ao produto da derivada da função de Ativação e o erro do neurônio j
- Se o neurônio j pertencer a camada escondida então o gradiente será igual ao produto entre a derivada da função de Ativação e a somatória dos produtos entre os pesos e os gradientes da camada posterior.

3.7.4 Melhora da velocidade

O método descrito acima pode ser um pouco lento, para melhorar essa performance Rumrlhart, Hilton e Willians desenvolveram, além do método descrito, o que eles chamaram de *Coeficiente de Momento*, ele consiste em

adicionar um termo no calculo da correção dos pesos, sendo essa nova parcela proporcional a correção anterior, logo a equação fica;

$$\Delta\omega_{ij}(n+1) = -\eta\delta_j y_i + \alpha\Delta\omega_{ij}(n)$$

Para o caso desse trabalho vai-se utilizar α igual a 1, visto que de acordo com os testes feitos esse valor foi o mais indicado e o que obteve o melhores resultados.

3.8. Descrição do Programa

Agora vamos fazer a descrição do funcionamento do algoritmo do programa, quais são suas interligações e a para que servem os objetos utilizados, no final.

Basicamente o programa tem cinco ações Open, Binarizar, Classificar, Treinar a Rede e Cadastrar.

3.8.1 Open

Este comando serve apenas para abrir a impressão digital a ser analisada e classificada.

3.8.2 Binarizar

Este comando torna a imagem da impressão digital, capturada em tons de cinza pelo scaner, uma imagem binarizada, ou seja transforma os tons de cinza em preto e branco facilitando assim a analise.

O método utilizado foi descrito anteriormente nas seções XXXX .

3.8.3 Classificar

Este comando serve para classificar a impressão digital em si. O processo que ele realiza está descrito abaixo.

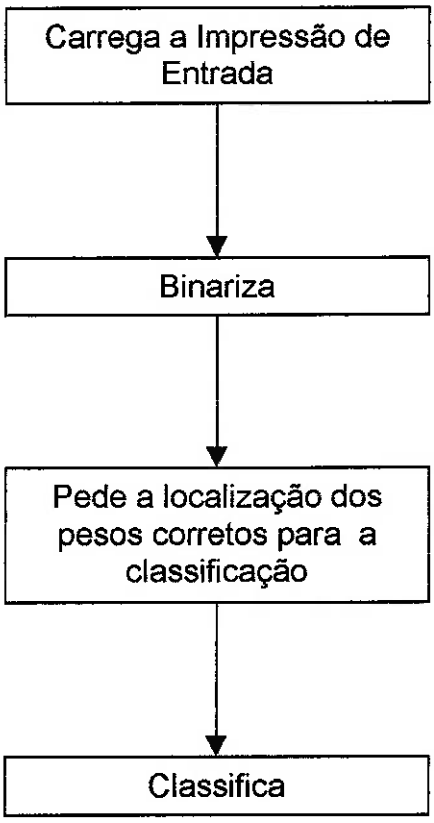


Figura 3.11: Diagrama de blocos do comando Classificar

3.8.4 Treinar Rede

Este comando serve para treinar os pesos da rede neural, o processo está descrito no esquema abaixo, o algoritmo de treinamento usado é o de Backpropagation, explicado anteriormente.

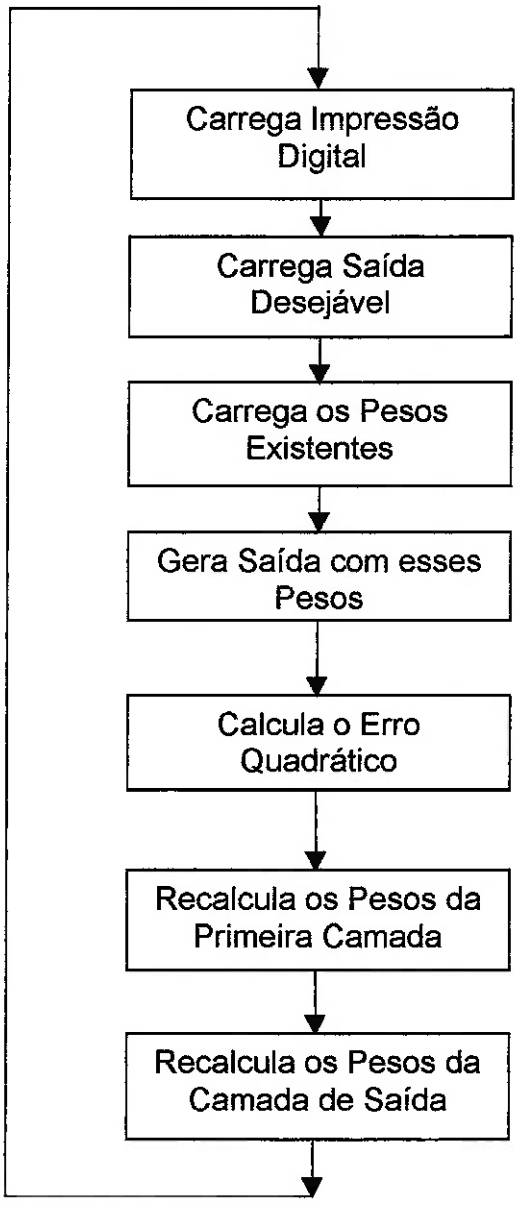


Figura 3.12: Diagrama de Blocos do comando Treinar Rede

3.8.5. Cadastrar

Esse comando apenas jogo em um arquivo de saídas desejadas predeterminado pelo programa os valores de saída colocados nos campos a direita chamados de "Saidas Desejadas", da seguinte forma:

1. 10000 se a nova impressão for do Whrol;
2. 01000 se for Arch
3. 00100 se for Tended Arch
4. 00010 se Loop para esquerda e;
5. 00001 se Loop para a direita.

3.9 Arquitetura do Programa

Agora será descrito a hierarquia entre as diversas funções e objetos existentes no programa:

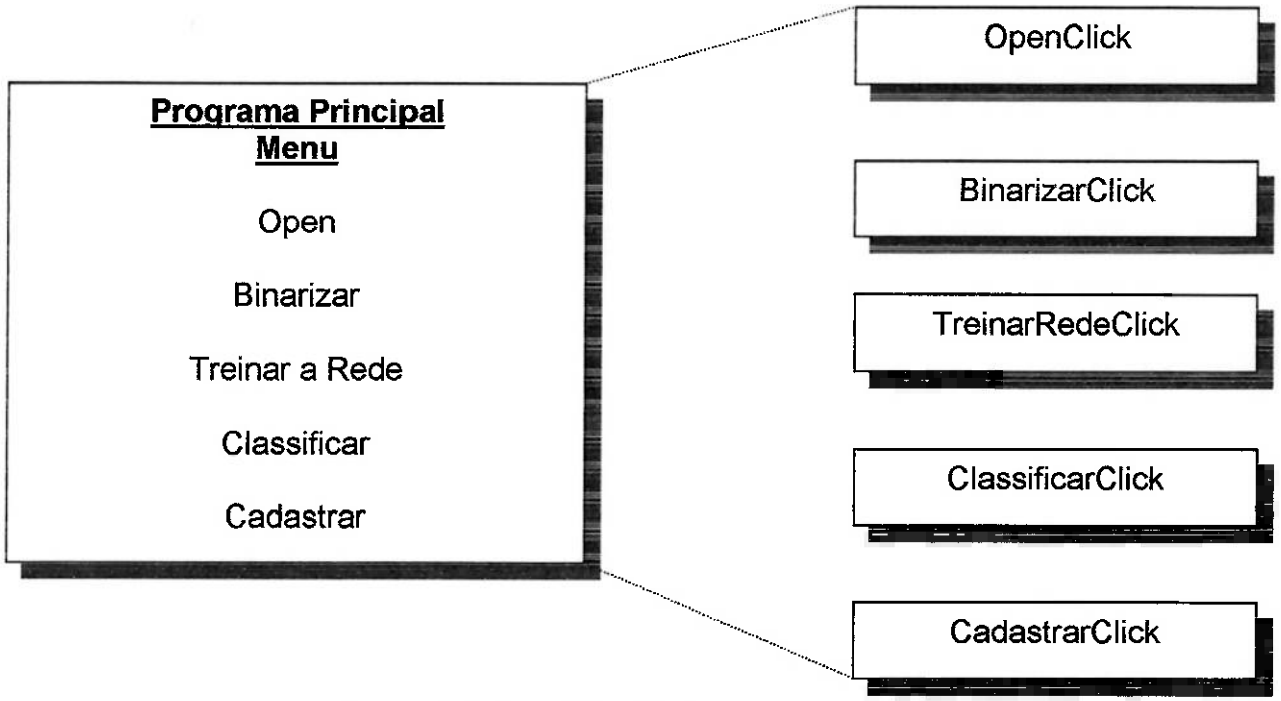


Figura 3.13 - Arquitetura do Programa Principal

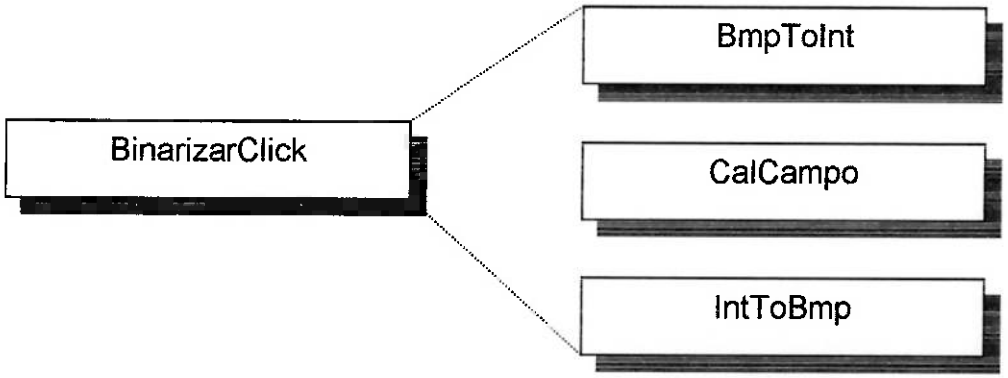


Figura 3.14 - Programa Principal: Rotina BinarizarClick

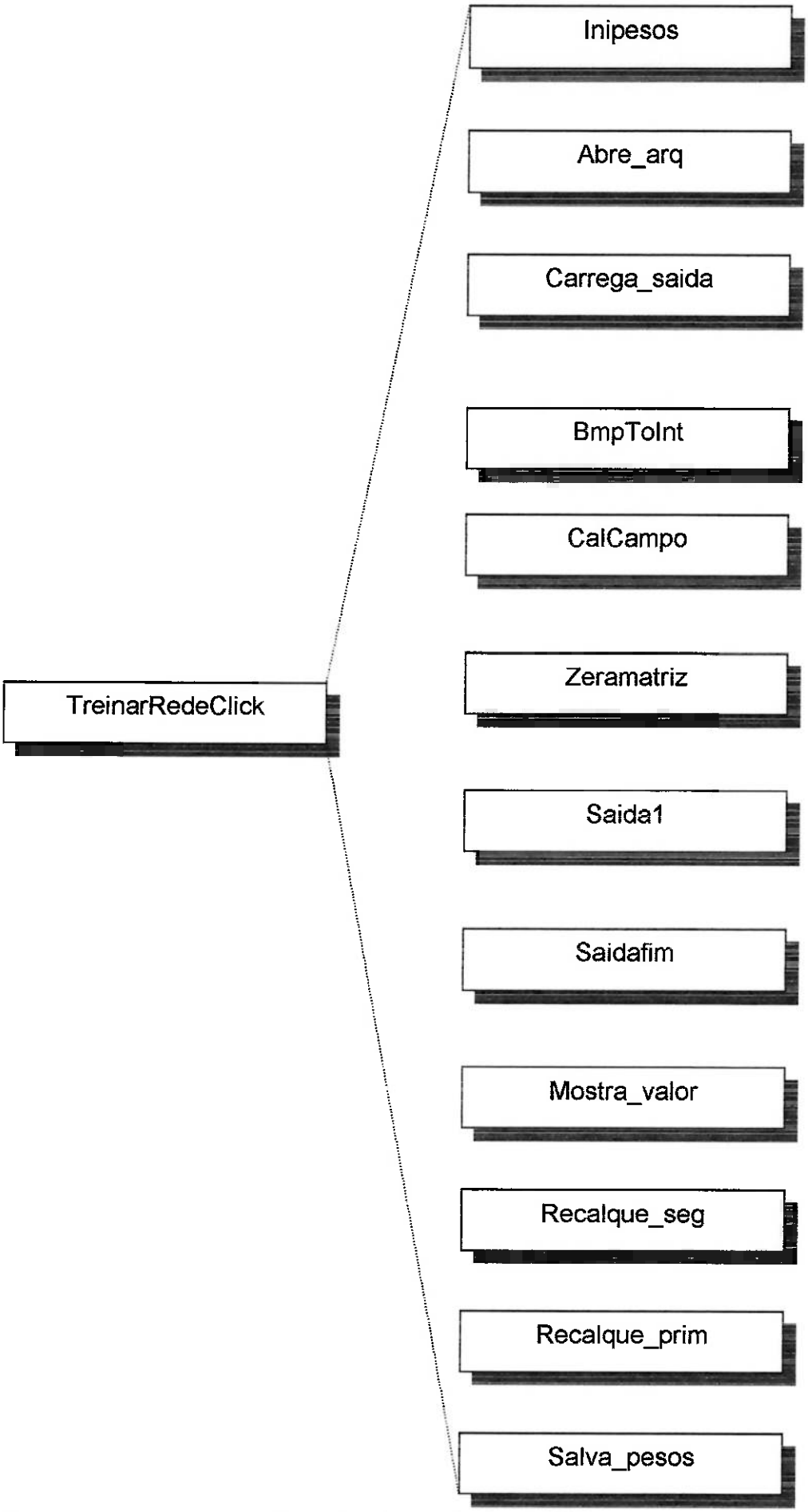


Figura 3.15 - Programa Principal: Rotina TreinaRedeClick

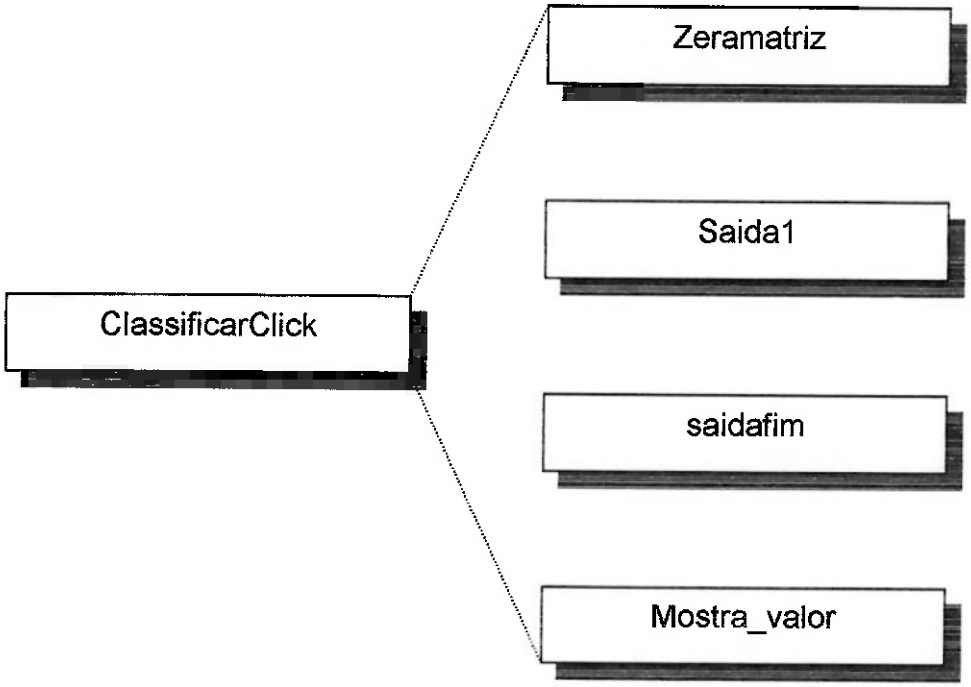


Figura 3.16 - Programa Principal : Rotina ClassificarClick

3.10. Testes e Resultados

Nesta seção vamos falar a respeito dos resultados obtidos com o programa, antes de começar vale fazer a seguinte observação, não se conseguiu chegar ao reconhecimento em si da impressão digital de forma satisfatória.

Isso ocorreu porque como a rede neural possui como entrada os ângulos (campo de orientação) dos pontos das cristas das impressões digitais ocorre o seguinte problema, a digital quando é cadastrada sua imagem é captada em uma certa posição, porém se da próxima vez que ela for captada não estiver na mesma posição da primeira o valor do ângulo em cada ponto relativamente a anterior ira mudar completamente, embora seu campo de orientação na se altere muito porque o campo de orientação representa um sentido de rotação, que não importa a posição que seja colocado o dedo na hora da captação, será o mesmo.

Portanto, para se fazer o reconhecimento fica muito dificil por que os valores gerados pela rede serão muito diferentes em cada caso, embora o meior continue sendo o mesmo. Depois de isso posto vamos aos resultados em si do programa.

3.10.1. Treinamento da Rede

O treinamento da rede é o processo mais demorado que o programa possui, em média ele demora 3 segundos por par de impressão digital se realizado em um computador com processador Pentium II 400Mhz de frequência de clock. A figura 3.17 abaixo mostra a tela do programa durante o treinamento.

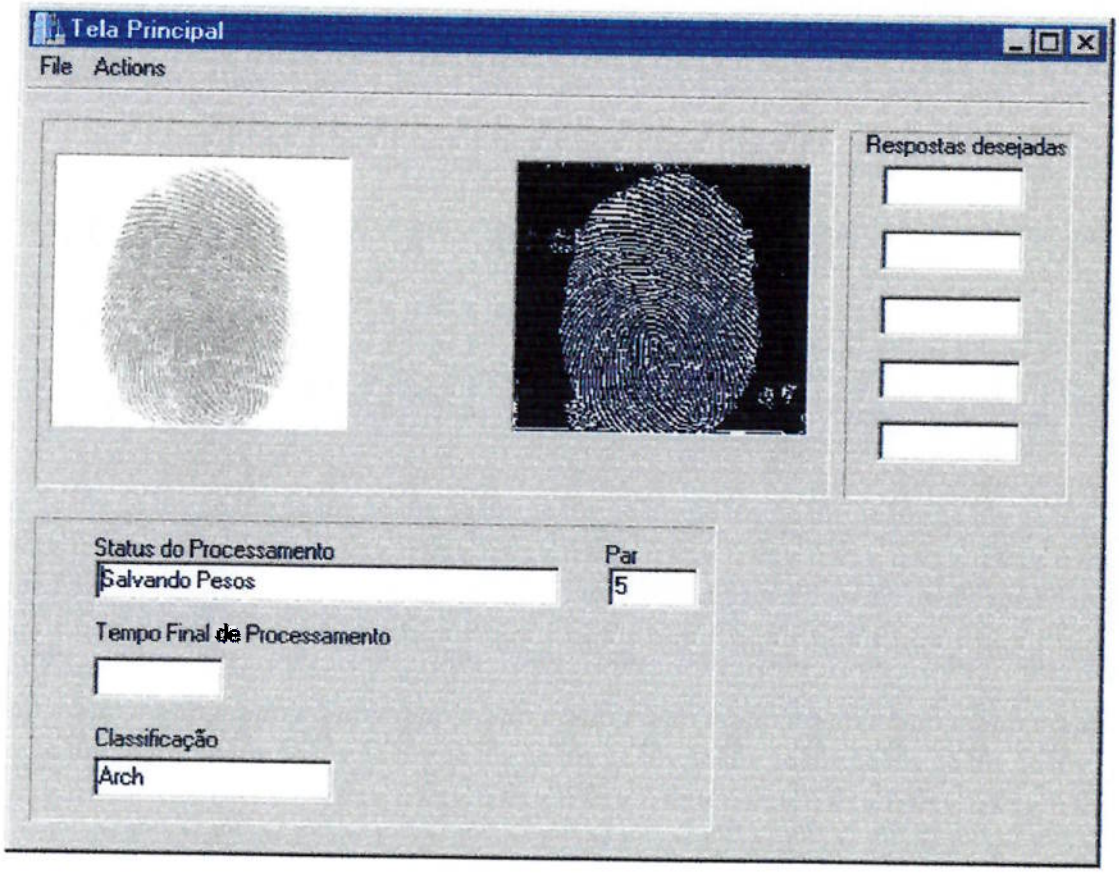


Figura 3.17: Processo de Treinamento

Como mostra a figura nesse momento o treinamento está sendo finalizado, pois no quadrado "Status do Processamento" está escrito Salvando Pesos, ou seja os pesos já foram recalculados estão sendo armazenados. O quadrado onde tem escrito "Par" corresponde ao par de treinamento que está sendo usado no momento para treinamento, no que tem escrito "Classificação" corresponde a classificação do mesmo dentro daquelas cinco classes já mencionadas.

Quando a opção "Treinar Rede" é escolhida no menu "Actions" a primeira tela que aparecerá será a mostrada na figura 3.18

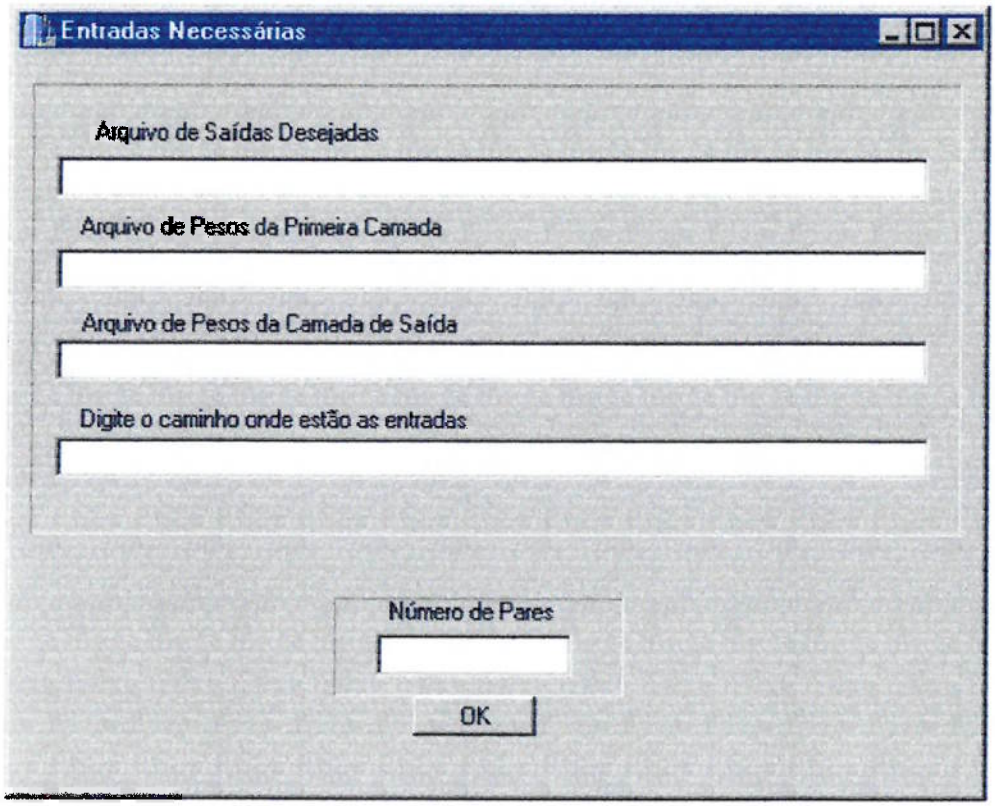


Figura 3.18: Entradas para treinamento da rede

Nessa tela deve ser digitado as entradas que serão utilizadas pelo programa para a realização do treinamento. A primeira delas é o arquivo de saídas desejadas, onde deve estar armazenadas todas as saídas desejadas que serão usadas durante para o treinamento, depois deve ser digitado o arquivo de pesos da primeira e segunda camada, se esses arquivos ainda não existirem deve ser digitado o caminho e arquivo onde se deseja criar esses arquivos, depois deve se digitar o caminho para o diretório onde estão armazenadas as impressões digitais que serão usadas, nesse diretórios os arquivos com as impressões digitais devem ser chamados de "firger1.bmp" ,

"firger3.bmp", etc, pois esse serão os nomes procurados pelo programa para realizar o treinamento.

3.10.2 Classificar

O processo de classificação feito por esse programa é bem eficiente tanto no que diz respeito a tempo como qualidade de acerto. Após a rede treinada, um computador, com as mesmas características mencionadas para o caso do treinamento, leva em média 0,25 segundos para realizar a classificação. A figura 3.19 a seguir mostra a tela do programa quando a classificação já foi realizada.

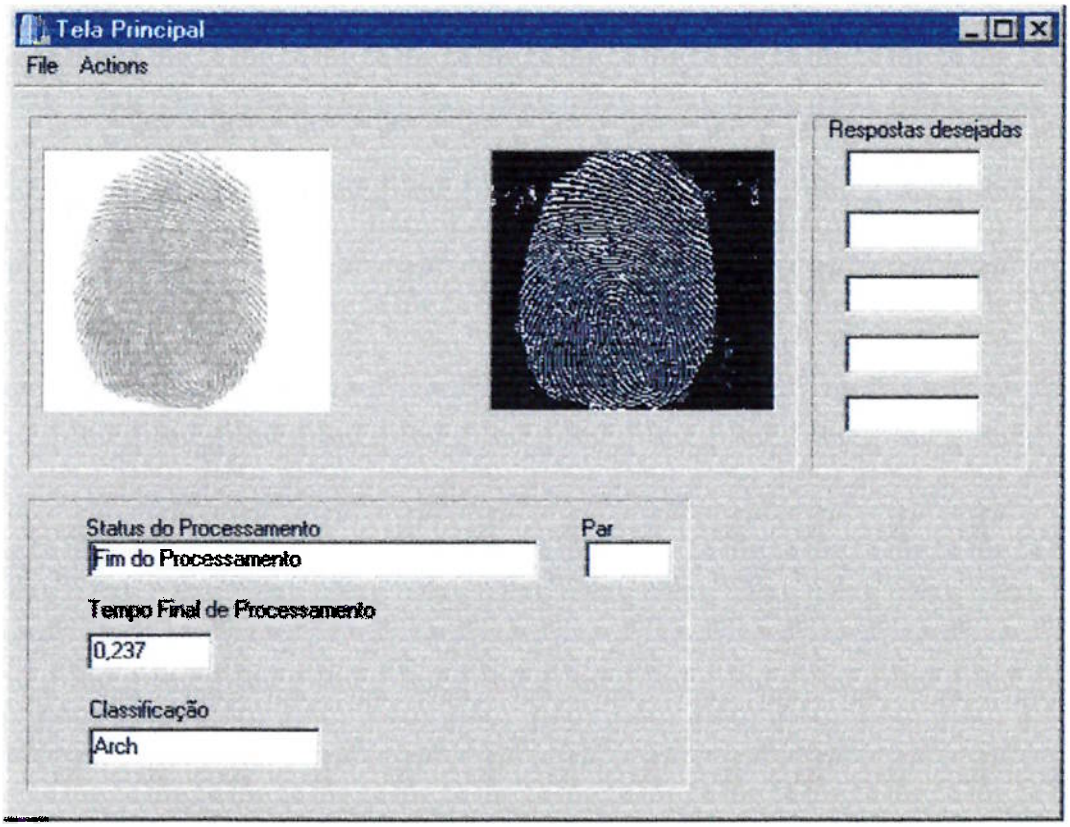


Figura 3.19: Tela final do processo de classificação

Quando a opção "Classificar " é escolhida no menu "Actions", primeiramente se terá que escolher qual digital se quer classificar, depois da escolha feita a tela mostrada na figura 3.20 aparecerá.

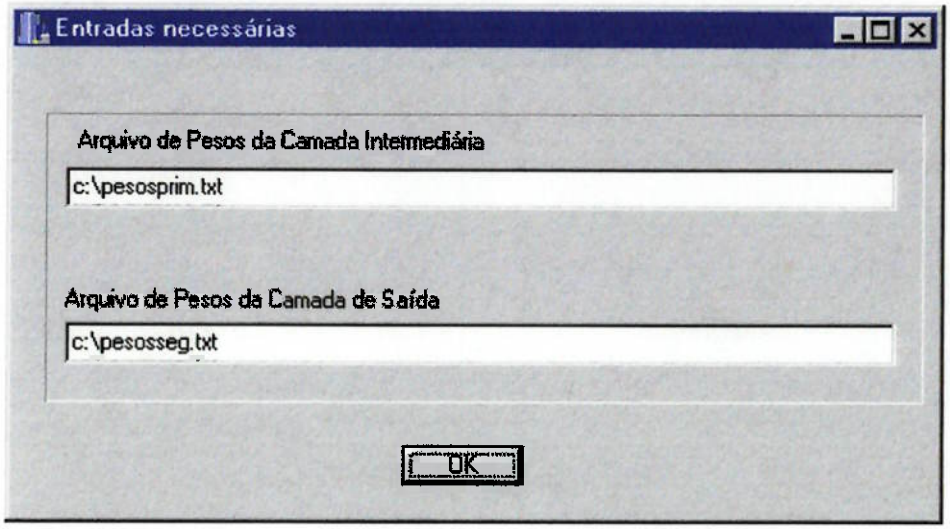


Figura 3.20: Entradas necessarias à classificação

Nesta tela deve ser digitado os arquivos de pesos da primeira e segunda camada com os quais o programa classificará a impressão digital escolhida anteriormente.

4. Análise dos resultados

4.1. Extração de Características

Como um todo, podemos dizer que os resultados obtidos pelo procedimento de extração de características foram satisfatórios. Devemos notar que há uma diferença entre extrair as minúcias e compará-las com um banco de dados. Essa última será tratada mais adiante. Nessa análise, constarão apenas os fatos relativos a extração em si.

O procedimento de extração é formado por diversos procedimentos em série. Como cada procedimento depende do resultado do procedimento anterior, erros em uma dada parte do programa se propagam e causam erros nas demais partes. Portanto, cada parte do programa deve ser otimizada, para que o próximo passo possa ocorrer sem problemas.

O primeiro cuidado a ser tomado está na etapa de ***normalização*** (item 2.2.2.1). Deve ser escolhido um valor para a média de intensidade e variância, e essa escolha é arbitrária. Embora nesse caso uma pequena diferença na escolha não cause muita diferença nos próximos processos, existem valores que gerariam grandes erros no próprio processo de normalização, inviabilizando o resto do programa. Por exemplo, escolhendo uma média mais próxima de 0, o resultado da normalização geraria pixels com valores for a da faixa 0-255. A cor desse pixel não mais seria um tom de cinza, e sim uma cor aleatória.

Outra parte muito importante é o ***filtro do campo de orientação*** (item 2.2.2.2). Os valores do filtro são completamente empíricos, e forma obtidos pelo método de teste e observação. Uma grande vantagem desse filtro é que os resultados são facilmente observáveis. O cuidado aqui deve ser tomado em dois casos. Se o peso central for muito alto, o ângulo que se está filtrando predominará entre os outros de tal forma que o filtro não faria sentido, pois erros não seriam bem corrigidos. Por outro lado, ponderar mais os outros pesos pode tornar os ângulos muito homogêneos, perdendo algumas particularidades importantes. É necessário determinar valores no meio termo.

O ***cálculo das frequências*** (item 2.2.2.3) a partir do vetor X também não é simples. É preciso determinar com precisão quais são os picos e verificar se são razoáveis. Novamente, a visualização das curvas (vetor X) é uma ferramenta de grande auxílio na escolha dos parâmetros que determinam se uma curva é ou não senoidal. A desvantagem nessa parte é que não é fácil de se visualizar se o resultado final (campo de frequências) está bom.

A ***filtragem da frequência*** (item 2.2.2.3) tem problemas semelhantes ao filtro do campo de orientação. Aparentemente, diferenças nesses filtros não gerariam mudanças consideráveis, mas pequenas mudanças geram grandes diferenças na imagem melhorada, e, consequentemente, nas minúcias encontradas. Para obter valores adequados para esses filtros, deve-se testar várias opções e observar o resultado final para diversas imagens. Porém, mesmo assim não conseguiríamos determinar o valor ótimo.

A questão do ***afinamento*** (item 2.3) e da ***extração de características*** (item 2.4) é um pouco mais delicada. Ambos os processos são simples, mais

apresentam resultados ruins. É necessário fazer um grande número de correções, para erros determinados empiricamente (novamente, a observação dos resultados em diversas imagens é necessária). Os dois processos em questão são intimamente relacionados. Se o resultado do afinamento fosse perfeito, não seriam necessárias correções no processo de extração. Por isso, é interessante melhorar ao máximo o processo de afinamento para que o processo de extração seja mais preciso.

4.2. Redes Neurais

Os resultados obtidos pelo método de rede neurais são bastante satisfatórios no que diz respeito à classificação das impressões digitais. Já quando se fala no reconhecimento em si esse processo não é muito eficiente.

O grande problema que ele sofre é que dificilmente o indivíduo colocará seu dedo na mesma posição que colocou quando foi cadastrado pela rede neural. Esse posicionamento diferenciado vai gerar um campo de orientação bem diferente do que aquele gerado quando o cadastro foi realizado. Logo, os cinco valores gerados pela rede, que teoricamente seriam usados para comparar as digitais, com a mudança da posição terão valores totalmente diferentes, o que impossibilitará a comparação das digitais.

Uma pergunta que pode surgir é por que funciona para classificar se o campo de orientação muda completamente. A resposta é simples, para classificar o que é mais importante não é a variação pontual do campo mas sim como ele varia ao longo da impressão digital. Então os valores gerados serão totalmente diferentes, porém o maior continuará sendo o mesmo e consequentemente a classificação também.

Este problema poderia ser evitado se de alguma forma fosse garantido que o indivíduo sempre colocasse o dedo na mesma posição que da primeira vez, isso talvez não fosse muito difícil pode-se por exemplo restringir o espaço para que a pessoa coloque o dedo e esse espaço restrito tenha a forma de um dedo, para que assim a pessoa saiba exatamente onde colocar o dedo e em que posição.

5. Comparação entre os Métodos

Os dois métodos encontram sérios problemas na hora de fazer a comparação em si, porém a origem desses problemas e suas respectivas soluções são de complexidades bem diferentes. O caso da rede neural é que a análise da impressão não é repetitiva, ou seja, é difícil garantir que mesmo sendo a mesma pessoa a colocar o dedo para se fazer a leitura, ela coloque na mesma posição. Já na impressão digital o grande problema é fazer um referenciamento das minúcias em impressões digitais diferentes da mesma pessoa. Os dois problemas possuem algumas possíveis soluções sugeridas na seção anterior.

Um ponto importantíssimo, e provavelmente mais relevante, é a qualidade de reconhecimento das impressões. Essa qualidade é bem diferente entre os mesmos; o método de extração de características é bem mais preciso no que diz respeito ao reconhecimento, caso fosse achada uma maneira de referenciar as minúcias encontradas nas digitais, do que o método de rede neurais, onde esse reconhecimento pode não ser tão preciso, visto que ele analisa uma característica mais geral da impressão, que é o campo de orientação.

Outro ponto importante que deve ser comparado aqui é o tempo que cada um levaria para fazer o reconhecimento em si. Esse tempo para o caso da rede seria bem pequeno, provavelmente apenas um pouco maior que o de classificação, visto que, quando a rede classifica, ele já gera os parâmetros de

comparação. Daí para frente seria só procurar no banco de dados alguma impressão que possuísse parâmetros similares.

Para o caso da extração de características o tempo de reconhecimento seria bem maior que o tempo gasto pela rede, por que além do processo de identificação das minúcias ser bem mais demorado que o processo de classificação da rede, depois de identificado as minúcias teriam que passar por algum processo de equiparação para poderem ser comparadas e então a impressão ser reconhecida.

Outro ponto que deve ser considerado é que, embora o tempo de reconhecimento da rede seja bem menor que o do outro método, o tempo de treinamento da rede é inconvenientemente grande, como já mencionado o programa leva em média 3 segundos por "par" de treinamento, o que é recomendado é que ela seja treinada com algo em torno de 4000 pares, o que levaria à um tempo médio de 3,3 horas para realizar cada treinamento, além disso o ideal é que para um certo número de entradas novas esse treinamento seja refeito para que o resultado da rede continue coerente.

Logo, no que diz respeito a tempo, enquanto a rede depois de treinada é bem mais eficiente que o método de extração de característica, tem a desvantagem de seu treinamento ser muito demorado.

Por fim vale aqui analisar a facilidade de implementação, embora não seja um ponto tão relevante, a rede neural é bem mais facilmente implementada que o outro método. O grande problema da implementação do método de extração de características, é a necessidade de analisar vários casos especiais, o que faz sua implementação ser bem especifica para cada tipo de análise necessária.

6. Conclusões

A conclusão mais importante que se pode fazer sobre esse trabalho é que os métodos são bem distintos quanto à sua finalidade, e, dessa forma, não deveriam estar sendo comparados. A idéia é juntar os dois programas num só, pois as vantagens de um complementam as deficiências do outro.

Da maneira como foi proposto o trabalho, os softwares deveriam reconhecer um indivíduo pela sua digital. Concluímos que o melhor método para isso é o de extração de características, pois pode realmente reconhecer uma digital. O método das redes neurais tem uma grande dificuldade nesse quesito, mas ele é muito útil para suprir uma das deficiências do outro método: velocidade de processamento. Por causa do procedimento de *enhancement*, a extração fica um pouco lenta. Para algumas aplicações o processo ficaria inviável, a menos que se construísse um processador dedicado a essa tarefa. Se somarmos a essa demora a demora causada pela varredura de todo o banco de dados, o uso do programa ficaria ainda mais restrito. Então, se utilizarmos o método de classificação realizado pelas redes neurais, o banco de dados seria subdividido, agilizando enormemente o processo.

Para que o programa realize o que foi proposto com boa eficiência, devemos então interligar os dois métodos e desenvolver um procedimento eficiente de comparação das minúcias extraídas com as do banco de dados. Embora já tenhamos comentado sobre as dificuldades de efetuar essa comparação,

outros métodos podem ser testados. Basear a comparação em dados estatísticos de posições de minúcias (histogramas) parece ser uma saída adequada.

Com o programa integrado e eficiente, surgem inúmeras aplicações para o software. Os departamentos de polícia podem utilizá-lo no reconhecimento de criminosos, coletando impressões no local do crime e buscando a identificação do criminoso num vasto banco de dados. Aqui, a velocidade de processamento na extração não é o ponto crucial, e sim o tempo de busca no banco de dados. A classificação se torna, então, fundamental.

Existem ainda diversos outros tipos de aplicação, que na sua maioria envolvem banco de dados menores, mas que requerem uma maior velocidade de processamento. Exemplos seriam permissões de acesso a ambientes por meio de impressões, acesso a internet ou ao próprio computador, substituição de senhas e chaves, etc... Em alguns desses casos, a segurança é fundamental, e dessa forma, a extração deverá ser feita da maneira mais precisa possível, o que, logicamente, levaria mais tempo. Para minimizar esse problema, deveríamos utilizar um processador específico para esse fim, como mencionado anteriormente.

É fácil de perceber que as possibilidades para esse tipo de programa são enormes. E apesar de cada aplicação necessitar de características específicas para o seu funcionamento, o programa base não precisa ser alterado.

7. Referências Bibliográficas

[1] JAIN, Anil; PANKANTI, Sharath; **Fingerprint Classification and Matching,** 1998

[2] JAIN, Anil; PRABHAKAR, Salil; HONG, Lin,; **A multichanel approach to fingerprint classification.**

[3] JAIN, Anil; HONG, Lin; WAN, Yifei; **Fingerprint Image Enhancement: Algorithm and Performance Evaluation**

[4] HERTZ, John; A. Krogh; R. G. Palmer; **Intrduction to theory of neural computation.** Addison – Wesley, 1991.

[5] A.K.Jain;S.Pankanti, **The image and Video Processing Handbook**, 1999

[6] WILSON, C. L.; CANDELA, G. T.; WATSON, C. I.; **Neural network fingerprint classification,** 1993 .

[7] WEI, Chang C.; **Aplicação de redes neurais para reconhecimento de sinais acústicos,** 1998.

Apêndice

A.1 – Listagem do Método de Extração de Características

```
#ifndef _funcoes_h
#define _funcoes_h

#define TAM 512     // Tamanho da imagem
#define MASK 16     // Tamanho dos blocos de subdivisão da imagem
#define Mo 200      // Média desejada para a intensidade
#define VARo 2000   // Variância desejada para a intensidade
#define VMIN 1000   //Valor abaixo do qual não se calcula o campo de orientação

#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <dos.h>

#include "c:\Poli\Trabalho de Formatura\Programa\gradientes.h"

const float pi=3.141593;
struct gradxy {         // Variável que guarda os valores dos
int x,y,z;              // gradientes na direção x, y e z.
};                      // A direção z nada mais é que o oposto
gradxy grad[TAM][TAM]; // da direção x. Isto é, z=-x.

double rot[TAM/MASK][TAM/MASK];  // Guarda o angulo médio de um bloco
double freq[TAM/MASK][TAM/MASK]; //Guarda a frequencia média de um bloco
double auxrot[TAM/MASK][TAM/MASK];// Variável auxiliar para 'rot'
double somax=0, somay=0;
int lii, lsi, lij, lsj;  // Limites da impressão digital
int liix,lsix,lijx,lsjx;
double final[TAM][TAM];
clock_t start, end;
int tempo;
int ceni, cenj;
struct minucia{
   int posx,posy;
   double ang;
   bool end;
};
minucia lista[200];
int viz[100][7];
int cont=0;

//---------------------------------------------------------------------------
// Função: BmpToInt
// Finalidade: Recebe uma imagem, que deve ter o tamanho de TAMxTAM pixels, e
// constrói uma matriz de mesmo tamanho com os valores inteiros (0 a 255) que
//representam os tons de cinza dos pixels
//---------------------------------------------------------------------------
void BmpToInt(TImage *imagem,int matriz[TAM][TAM]){

   for (int i=0;i<TAM;i++)
   for (int j=0;j<TAM;j++)
      matriz[i][j]=imagem->Canvas->Pixels[j][i]&0xFF;
}
```

```
//---------------------------------------------------------------------------
// Função: IntToBmp
// Finalidade: Exatamente o processo contrário da função anterior, BmpToInt.
//---------------------------------------------------------------------------
void IntToBmp(TImage *image, int matriz[TAM][TAM]){

   int h,w;
   h=image->Height;
   w=image->Width;
   image->Visible=false;
   image->Height=TAM;
      image->Width=TAM;
   for(int i=0;i<TAM;i++)
   for(int j=0;j<TAM;j++){
      if(matriz[i][j]>255) matriz[i][j]=255;
      if(matriz[i][j]<0) matriz[i][j]=0;
      image->Canvas->Pixels[j][i]=(TColor)(matriz[i][j]*65793);
   }
   image->Height=h;
   image->Width=w;
   image->Visible=true;
}

//---------------------------------------------------------------------------
// Função: peso
// Finalidade: Atribui um peso para uma determinada posição do filtro do Campo
// de Orientação (ver explicação no item 2.2.2.2)
//---------------------------------------------------------------------------
int peso (int i, int j, int u, int v){

   int dist,out;
   dist=pow(i-u,2)+pow(j-v,2);//representa a distância ao quadrado de um pixel
   switch(dist){              //(u,v) até o pixel (i,j)
      case 0:{out=5;break;}
      case 1:{out=8;break;}
      case 2:{out=12;break;}
      case 4:{out=2;break;}
      case 5:{out=-2;break;}
      case 8:{out=2;break;}
   }
   return(out);
}

//---------------------------------------------------------------------------
// Função: Normalizacao
// Finalidade: Padroniza o tom e o contraste das imagens
//---------------------------------------------------------------------------
void Normalizacao (int matriz[TAM][TAM]){

   int M = 0;
   int VAR = 0;

   for (int i=0;i<TAM;i++)      // Cálculo da média das intensidades
   for (int j=0;j<TAM;j++)
      M+=matriz[i][j];
   M/=pow(TAM,2);
```

```
    for (int i=0;i<TAM;i++)     // Cálculo da variância das intensidades
    for (int j=0;j<TAM;j++)
       VAR+=pow((matriz[i][j]-M),2);
    VAR/=pow(TAM,2);

    for (int i=0;i<TAM;i++)     // Normalização
    for (int j=0;j<TAM;j++){
       if (matriz[i][j]>M)
             matriz[i][j]=Mo+sqrt(pow((matriz[i][j]-M),2)*VARo/VAR);
       else
             matriz[i][j]=Mo-sqrt(pow((matriz[i][j]-M),2)*VARo/VAR);
    }
}

//-------------------------------------------------------------------------
// Função: CampoDeOrientacao
// Finalidade: Primeiro cálculo do campo de orientação
//-------------------------------------------------------------------------
void CampoDeOrientacao(int matriz[TAM][TAM], double rot[TAM/MASK][TAM/MASK]){

   double vx=0, vy=0, vz=0;

   for (int i=0;i<TAM;i++)       // Cálculo dos gradientes
   for (int j=0;j<TAM;j++){
      grad[i][j].x=Gradiente('x',matriz,i,j);
      grad[i][j].y=Gradiente('y',matriz,i,j);
      grad[i][j].z=Gradiente('z',matriz,i,j);
   }

   for (int u=0;u<TAM/MASK;u++)  // Varredura de bloco em bloco
   for (int v=0;v<TAM/MASK;v++){
      for (int i=0;i<MASK;i++)  // Varredura de pixel em pixel,
      for (int j=0;j<MASK;j++){ // dentro de cada bloco
         vx+=2*grad[i+MASK*u][j+MASK*v].x*grad[i+MASK*u][j+MASK*v].y;
         vy+=pow(grad[i+MASK*u][j+MASK*v].x,2)-
             pow(grad[i+MASK*u][j+MASK*v].y,2);
         vz+=2*grad[i+MASK*u][j+MASK*v].z*grad[i+MASK*u][j+MASK*v].y;
      }

      if (abs(vx)+abs(vy)>VMIN){
            if (vx>=0){           // Cálculo dos ângulos em cada bloco.
                  if (vy>=0){       // Maiores explicações no item 2.2.2.2
                  if (vx<-vy){
                  rot[u][v]=0.5*atan(vx/vy);
               }
               else{
                  rot[u][v]=0.5*(1.57079633-atan(vy/vx));
               }
            }
            else{
                  if (vx<-abs(vy)){
                  rot[u][v]=0.5*(3.14159265+atan(vx/vy));
               }
               else{
                  rot[u][v]=0.5*(3.14159265+(-1.57079633-atan(vy/vx)));
               }
            }
         }
            else{
                  if (vy<=0){
```

```
                    if (abs(vx)<=abs(vy)){
                    rot[u][v]=3.14159265-(0.5*(3.14159265+atan(vz/vy)));
                }
                else{
                    rot[u][v]=3.14159265-(0.5*(3.14159265+(-1.57079633-
                    atan(vy/vz))));
                }
            }
            else{
                    if (abs(vx)<=vy){
                    rot[u][v]=3.14159265-(0.5*atan(vz/vy));
                }
                else{
                    rot[u][v]=3.14159265-(0.5*(1.57079633-atan(vy/vz)));
                }
            }
            }
        }
        else rot[u][v]=10000;
        vy=0; vx=0; vz=0;
    }
}

//-------------------------------------------------------------------------
// Função: Media
// Finalidade: Calcula o valor do ângulo médio de uma região
//-------------------------------------------------------------------------
double Media(double somax, double somay){
   double aux;
      if((somax==0)&&(somay==0)) aux=0;
      else{                                  //Novamente, item 2.2.2.2
      if (somax>=0){
            if (somay>=0){
            if (abs(somax)>=abs(somay)){
                  aux=0.5*(1.570796-atan(somay/somax));
            }
            else{
                  aux=0.5*atan(somax/somay);
            }
         }
         else{
            if (abs(somax)>=abs(somay)){
                  aux=0.5*(1.570796-atan(somay/somax));
            }
            else{
                  aux=0.5*(3.141593+atan(somax/somay));
            }
         }
      }
      else{
            if (somay>=0){
            if (abs(somax)>=abs(somay)){
                  aux=0.5*(4.712389-atan(somay/somax));
            }
            else{
                  aux=0.5*(6.283185+atan(somax/somay));
            }
         }
         else{
            if (abs(somax)>=abs(somay)){
```

```
                aux=0.5*(4.712389-atan(somay/somax));
            }
            else{
                aux=0.5*(3.141593+atan(somax/somay));
            }
        }
    }
    }
    return(aux);
}

//-------------------------------------------------------------------------
// Função: Verifica
// Finalidade: Dado quatro valores de ângulos adjascentes, verifica se há uma
// diferença maior do que 70 graus entre dois deles
//-------------------------------------------------------------------------
bool Verifica(double se, double sd, double ie, double id){

   bool ok;
   double max=0;
   double aux;
   double vet[4];
   vet[0]=se*180/pi;   //Armazena os quatro ângulos na forma de graus.
   vet[1]=sd*180/pi;
   vet[2]=ie*180/pi;
   vet[3]=id*180/pi;
   for (int i=0;i<3;i++){
      for (int j=i+1;j<4;j++){
         aux=abs(vet[i]-vet[j]);
         if (aux>90) aux=180-aux;
         if (aux>max) max=aux;  //max guarda a maior diferença encontrada
      }
   }
   if (max>70) ok=false;  // Caso a diferença máxima seja menor que 70 graus
   else ok=true;          // a função retorna true

   return(ok);
}
//-------------------------------------------------------------------------
// Função: FiltrarCampo
// Finalidade: Cálculo melhorado do campo de orientação
//-------------------------------------------------------------------------

void FiltrarCampo(double rot[TAM/MASK][TAM/MASK]){

   double se, sd, ie, id;  // Guardam o valor médio dos ângulos nas regiões
                           // Superior Esquerda, Superior Direita, Inferior
   for (int i=2;i<TAM/MASK-2;i++)   //Esquerda e Inferior Direita
   for (int j=2;j<TAM/MASK-2;j++){
      somax=0; somay=0;
      for (int u=i-2;u<=i;u++)
      for (int v=j-2;v<=j;v++){
         if (rot[u][v]!=10000){
            somax+=sin(2*rot[u][v])*peso(i,j,u,v);
            somay+=cos(2*rot[u][v])*peso(i,j,u,v);
         }
      }
      se=Media(somax,somay);
      somax=0; somay=0;
      for (int u=i-2;u<=i;u++)
```

```
        for (int v=j;v<=j+2;v++){
           if (rot[u][v]!=10000){
              somax+=sin(2*rot[u][v])*peso(i,j,u,v);
              somay+=cos(2*rot[u][v])*peso(i,j,u,v);
           }
        }
        sd=Media(somax,somay);
        somax=0; somay=0;
        for (int u=i;u<=i+2;u++)
        for (int v=j-2;v<=j;v++){
           if (rot[u][v]!=10000){
              somax+=sin(2*rot[u][v])*peso(i,j,u,v);
              somay+=cos(2*rot[u][v])*peso(i,j,u,v);
           }
        }
        ie=Media(somax,somay);
        somax=0; somay=0;
        for (int u=i;u<=i+2;u++)
        for (int v=j;v<=j+2;v++){
           if (rot[u][v]!=10000){
              somax+=sin(2*rot[u][v])*peso(i,j,u,v);
              somay+=cos(2*rot[u][v])*peso(i,j,u,v);
           }
        }
        id=Media(somax,somay);
        if(Verifica(se,sd,ie,id)){  // Se não houver problemas com os ângulos,
              somax=sin(2*se)+sin(2*sd)+sin(2*ie)+sin(2*id);// calcula média final
              somay=cos(2*se)+cos(2*sd)+cos(2*ie)+cos(2*id);
              auxrot[i][j]=Media(somax,somay);
        }
        else auxrot[i][j]=rot[i][j];  // Se não, mantém o valor anterior


     }
     for (int i=2;i<TAM/MASK-2;i++)
     for (int j=2;j<TAM/MASK-2;j++){
        rot[i][j]=auxrot[i][j];    // Atualiza a matriz rot
     }
}

//-------------------------------------------------------------------------
// Função: MostrarAngulo
// Finalidade: Desenha ângulos na imagem (função somente utilizada durante o
// desenvolvimento do software, como forma de verificar visualmente o resultado
// do campo de orientação)
//-------------------------------------------------------------------------

void MostrarAngulo(int matriz[TAM][TAM], double auxrot[TAM/MASK][TAM/MASK]){

   int a,b,c;
   double d;

   for (int u=0;u<TAM/MASK;u++)
   for (int v=0;v<TAM/MASK;v++){
      a=u*MASK+MASK/2-1;
      b=v*MASK+MASK/2-1;

      if ((auxrot[u][v]>0.785398)&&(auxrot[u][v]<2.356194)){
         int j=b;
         for (int i=a;i<(u+1)*MASK-1;i++){
```

```
                matriz[i][j]-255;
                d=tan(1.570796-auxrot[u][v])*(i-a+1);
                if (d<0) d-=0.5;
                else d+=0.5;
                c=d;
                j=b-c;
            }
            j=b;
            for (int i=a;i>=u*MASK;i--){
               matriz[i][j]=255;
               d=tan(1.570796-auxrot[u][v])*(a-i+1);
               if (d<0) d-=0.5;
               else d+=0.5;
               c=d;
               j=b+c;
            }
         }
         else{
               int i=a;
               for (int j=b;j<(v+1)*MASK-1;j++){
               matriz[i][j]-255;
               d=tan(auxrot[u][v])*(j-b+1);
               if (d<0) d-=0.5;
               else d+=0.5;
               c=d;
               i=a-c;
            }
            i=a;
            for (int j=b;j>=v*MASK;j--){
               matriz[i][j]=255;
               d=tan(auxrot[u][v])*(b-j+1);
               if (d<0) d-=0.5;
               else d+=0.5;
               c=d;
               i=a+c;
            }
         }
      }
   }

//-------------------------------------------------------------------------------
// Função: Calc_Freq
// Finalidade: Calcular a frequencia senoidal de x (na verdade, retorna o
// período de x)
//-------------------------------------------------------------------------------
float Calc_Freq(int x[MASK*2]){

   int aux;
   float resultado=0;
      struct pontos{   //Guarda supostos picos e vales (pos) e o tamanho (seq)
      int seq,pos;
   };
   struct picos{
      int ant,prox,pos;
   };
   int valor; // Indica se a sequência de pontos é decrescente ou crescente
   pontos a[MASK*2];
   picos b[MASK];
   int i=0;  // Número de supostos picos e vales
   int j=0;  // Número de picos reais
```

```
    if (x[1]>x[0]) valor=1;
    else valor=-1;
    for (int k=2;k<32;k++){
       if (valor<0){  // Se a sequência for decrescente e o ponto for inferior ao
       if (x[k]<x[k-1]) valor-=1;// anterior, aumentar o tamanho da sequencia
          else{
              a[i].seq=valor; // Se não, terminar a sequência negativa, começar
              a[i].pos=k-1;   // uma positiva, mas guardar o tamanho da sequência
              valor=1;        // e a posição final da mesma
              i++;
          }
       }
       else{ //valor>0      Análogo ao caso a cima
             if (x[k]>x[k-1]) valor+=1;
          else{
             a[i].seq=valor;
             a[i].pos=k-1;
             valor=-1;
             i++;
          }
       }
    }
    a[i].seq=valor;      // Guarda o último ponto também
    a[i].pos=MASK*2-1;
    for (int k=0;k<i;k++){
       if ((a[k].seq>=2)&&(a[k+1].seq<=-2)){  // Detecta e armazena picos
             b[j].ant=a[k].seq;
          b[j].prox=abs(a[k+1].seq);
          b[j].pos=a[k].pos;
          j++;
       }
    }
    if (j<2) resultado=-1; // Não se pode obter frequência com menos de 2 picos
    else{  // Ver item 2.2.2.3 para maiores detalhes
       for (int k=1;k<j;k++) if (abs(b[k].ant-b[k-1].prox)>3) resultado=-1;
       if (j>2) for (int k=2;k<j;k++) if (abs(b[k].pos-2*b[k-1].pos+b[k-
       2].pos)>3) resultado=-1;
    }
    if (resultado==0){
       resultado=float(b[j-1].pos-b[0].pos)/(j-1);
       aux=int((MASK*2-3)/resultado);// Dado um período, qual é o mínimo de picos
       if(j<aux) resultado=-1;      // esperado. Se o número obtido for menor que
       if(resultado>=15) resultado=-1;// esse mínimo, não se deve considerar.
       if(resultado<=3) resultado=-1;
    }
    return(resultado);
}

//-------------------------------------------------------------------------
// Função: Interpolar
// Finalidade: Interpola valores de frequência
//-------------------------------------------------------------------------
void Interpolar(double freq[TAM/MASK][TAM/MASK], float kernel[7][7]){

   bool sair = false;
   double freq1[TAM/MASK][TAM/MASK]; //Matriz auxiliar de frequências
   int i=3;                          //Cálculo dos limites da impressão
   int j;
   do{
      for(j=0;j<TAM/MASK;j++)
```

```
            if(freq[i][j]!=-1){
            sair=true;
            lii=i;
         }
      i++;
   }while(!sair);
   i=TAM/MASK-4;
   sair=false;
   do{
      for(j=0;j<TAM/MASK;j++)
            if(freq[i][j]!=-1){
            sair=true;
            lsi=i;
         }
      i--;
   }while(!sair);
   j=3;
   sair=false;
   do{
      for(i=0;i<TAM/MASK;i++)
            if(freq[i][j]!=-1){
            sair=true;
            lij=j;
         }
      j++;
   }while(!sair);
   j=TAM/MASK-4;
   sair=false;
   do{
      for(i=0;i<TAM/MASK;i++)
            if(freq[i][j]!=-1){
            sair=true;
            lsj=j;
         }
      j--;
   }while(!sair);

   for (i=lii;i<=lsi;i++)
   for (j=lij;j<=lsj;j++){
      freq1[i][j]=freq[i][j];
   }

   for (i=lii;i<=lsi;i++)
   for (j=lij;j<=lsj;j++){
      if (freq[i][j]==-1){
         double soma1=0;
         double soma2=0;
            for (int u=i-3;u<=i+3;u++)
         for (int v=j-3;v<=j+3;v++){
            if(freq[u][v]!=-1){    // Interpolação
                  soma1+=kernel[u-i+3][v-j+3]*freq[u][v];
                  soma2+=kernel[u-i+3][v-j+3];
            }
         }
         if (soma2!=0) freq1[i][j]=soma1/soma2;
         else freq1[i][j]=freq[i][j];
      }
   }
   for (i=lii;i<=lsi;i++)
   for (j=lij;j<=lsj;j++){
```

```
        freq[i][j]=freq1[i][j];
     }
  }
  //-------------------------------------------------------------------------
  // Função: Filtrar_Freq
  // Finalidade: Eliminar ruídos nas frequências calculadas
  //-------------------------------------------------------------------------
  void Filtrar_Freq(double freq[TAM/MASK][TAM/MASK]){

     int peso, dist;
        double freq1[TAM/MASK][TAM/MASK];

     for(int i=0;i<TAM/MASK;i++)
     for(int j=0;j<TAM/MASK;j++){
        freq1[i][j]=freq[i][j];
     }

     for(int i=lii;i<=lsi;i++)     // Varre a impressão, dentro dos limites obtidos
     for(int j=lij;j<=lsj;j++){    // anteriormente.
        double soma=0;
        int div=0;
        for(int u=i-3;u<=i+3;u++)
        for(int v=j-3;v<=j+3;v++){
              if ((u<lii)||(u>lsi)||(v<lij)||(v>lsj)) peso=0;
           else {
              dist=pow(i-u,2)+pow(j-v,2); // Atribui o peso correspondente à
              switch(dist){               // do ponto
              case 0:{peso=20;break;}
              case 1:{peso=10;break;}
              case 2:{peso=6;break;}
              case 4:{peso=5;break;}
              case 5:{peso=4;break;}
              case 8:{peso=3;break;}
              case 9:{peso=2;break;}
              case 10:{peso=1;break;}
              case 13:{peso=1;break;}
              case 18:{peso=1;break;}
              }
           }
           div+=peso;
           soma+=peso*freq1[u][v];
        }
        freq[i][j]=soma/div;
     }
  }

  //-------------------------------------------------------------------------
  // Função: Frequencia
  // Finalidade: Calcular as frequências da impressão
  //-------------------------------------------------------------------------
  void Frequencia(int matriz[TAM][TAM], double rot[TAM/MASK][TAM/MASK], double
  freq[TAM/MASK][TAM/MASK]){

     int u,v;
     int x[MASK*2];
     float kernel[7][7];
     int aux;
     for (int i=0;i<7;i++)
     for (int j=0;j<7;j++){          // Preenche os coeficientes do kernel
        aux=pow(3-i,2)+pow(3-j,2);
```

```
        switch(aux){
        case 0:{kernel[i][j]=13.3;break;}
        case 1:{kernel[i][j]=12.6;break;}
        case 2:{kernel[i][j]=11.9;break;}
        case 4:{kernel[i][j]=10.7;break;}
        case 5:{kernel[i][j]=10.1;break;}
        case 8:{kernel[i][j]=8.5;break;}
        case 9:{kernel[i][j]=8.1;break;}
        case 10:{kernel[i][j]=7.6;break;}
        case 13:{kernel[i][j]=6.5;break;}
        case 18:{kernel[i][j]=4.9;break;}
        }
    }
    for (int i=0;i<TAM/MASK;i++){
       freq[i][0]=-1;              // "Zera" as bordas
       freq[i][TAM/MASK-1]=-1;
       freq[0][i]=-1;
       freq[TAM/MASK-1][i]=-1;
    }
       for (int i=1;i<TAM/MASK-1;i++)
    for (int j=1;j<TAM/MASK-1;j++){
       for (int k=0;k<MASK*2;k++){
          x[k]=0;
             for (int d=0;d<MASK;d++){
             u=0.5+i*MASK+MASK/2-1+(MASK/2-d)*sin(rot[i][j])+(k-
             MASK)*cos(rot[i][j]);
             v=0.5+j*MASK+MASK/2-1+(d-MASK/2)*cos(rot[i][j])+(k-
             MASK)*sin(rot[i][j]);
             x[k]+=matriz[u][v]; // Calcula o vetor X, na orientação correta
          }
       }
       freq[i][j]=1/Calc_Freq(x);
    }
    Interpolar(freq,kernel);
    Filtrar_Freq(freq);
}

//-------------------------------------------------------------------------
// Função: Gabor
// Finalidade: Filtra a imagem
//-------------------------------------------------------------------------
void Gabor(int matriz[TAM][TAM], double rot[TAM/MASK][TAM/MASK], double
freq[TAM/MASK][TAM/MASK]){

   const int G=5;  // Variável relacionada ao tamanho do filtro de Gabor (2*G+1)
   double soma;
   float sigma=4;
   double min=100000;
   double max=-100000;
   int ci,cj;
   double c,gab;
   double c1,c2,c3,c4,aux;

   lsix=((lsi+lii)/2)*0.15+lsi*0.85; // Reduz os limites da impressão obtidos
   lsjx=((lsj+lij)/2)*0.2+lsj*0.8;   // anteriormente, para garantir a aplicação
   liix=((lsi+lii)/2)*0.25+lii*0.75; // do filtro sempre dentro da impressão
   lijx=((lsj+lij)/2)*0.2+lij*0.8;
```

```
    for (int i=liix+G;i<lsix-G;i++)
    for (int j=lijx+G;j<lsjx-G;j++){
       ci=i/MASK;
       cj=j/MASK;
       soma=0;

       for (int EF=0;EF<=G;EF++){
             c=cos(pi*EF/G);
          for (int EO=0;EO<=G;EO++){
             gab=exp(-0.5*(pow(EO,2)+pow(EF/(2*G*freq[ci][cj]),2))/pow(sigma,2));
             aux=0;
             // Devido à dupla simetria do filtro, o cálculo dos coeficientes
             // é feito de quatro em quatro.
             c1=-EO*sin(rot[ci][cj]);
             c2=(EF/(2*G*freq[ci][cj]))*cos(rot[ci][cj]);
             c3=EO*cos(rot[ci][cj]);
             c4=(EF/(2*G*freq[ci][cj]))*sin(rot[ci][cj]);
             if((EO!=0)&&(EF!=0)){
                aux+=matriz[int(i+c1-c2+0.5)][int(j+c3-c4+0.5)]+matriz[int(i-
c1+c2+0.5)][int(j-c3+c4+0.5)]+matriz[int(i-c1-c2+0.5)][int(j-c3-c4+0.5)];
             }
             else{
                   if(EO!=0){
                   aux+=matriz[int(i-c1+0.5)][int(j-c3+0.5)];
                }
                if(EF!=0){
                   aux+=matriz[int(i-c2+0.5)][int(j-c4+0.5)];
                }
             }
             soma+=(aux+matriz[int(i+c1+c2+0.5)][int(j+c3+c4+0.5)])*gab*c;
          }

       }
       final[i][j]=soma;
    }
    for (int i=liix+G;i<lsix-G;i++)  // Loop cuja finalidade é determinar os
    for (int j=lijx+G;j<lsjx-G;j++){ // valores mínimo e máximo da somatória
       if (final[i][j]<min) min=final[i][j];
       if (final[i][j]>max) max=final[i][j];
    }
    for (int i=0;i<TAM;i++)
    for (int j=0;j<TAM;j++)
       matriz[i][j]=254;

    for (int i=liix+G;i<lsix-G;i++)  // Loop cuja finalidade é transformar,
    for (int j=lijx+G;j<lsjx-G;j++){ // linearmente, os valores da somatória em
       matriz[i][j]=int(255*(final[i][j]-min)/(max-min));// valores de 0 a 255
    }
}


//---------------------------------------------------------------------------
// Função: Enhancement
// Finalidade: Melhorar a imagem da impressão digital
//---------------------------------------------------------------------------
void Enhancement(int matriz[TAM][TAM]){

   Normalizacao(matriz);

   CampoDeOrientacao(matriz,rot);
```

```
    FiltrarCampo(rot);

    Frequencia(matriz,rot,freq);

    int n;            //Recálculo dos limites da impressão
    int x=-1;
    int y;
    do{
       n=0;
       x++;
       for (y=0;y<TAM;y++)  if (matriz[x][y]<150) n++;
    }while(n<10);
    lii=x;
    x=TAM;
    do{
       n=0;
       x--;
       for (y=0;y<TAM;y++)  if (matriz[x][y]<150) n++;
    }while(n<10);
    lsi=x;
    y=-1;
    do{
       n=0;
       y++;
       for (x=0;x<TAM;x++)  if (matriz[x][y]<150) n++;
    }while(n<10);
    lij=y;
    y=TAM;
    do{
       n=0;
       y--;
       for (x=0;x<TAM;x++)  if (matriz[x][y]<150) n++;
    }while(n<10);
    lsj=y;

    start=clock();
    Gabor(matriz,rot,freq);
    end=clock();
    tempo=(end - start) / CLK_TCK;   //Mede o tempo de aplicação do filtro

      //MostrarAngulo(matriz,rot);

}

//-------------------------------------------------------------------------
// Função: Thinning
// Finalidade: Afinar a imagem
//-------------------------------------------------------------------------
void Thinning(int matriz[TAM][TAM], double rot[TAM/MASK][TAM/MASK]){

      int u,v,ci,cj,min,xmin,n,a,b;

      for (int i=liix+5;i<lsix-5;i++)
      for (int j=lijx+5;j<lsjx-5;j++){
      min=10000;
      ci=i/MASK;
      cj=j/MASK;
      for (int x=-2;x<=2;x++){  // Varre uma vizinhança do ponto, na direção
            u=int(i+x*cos(rot[ci][cj])+0.5);// normal à orientação, e detecta o
```

```
      v=int(j+x*sin(rot[ci][cj])+0.5);// ponto de menor valor.
      if (matriz[u][v]<min){
         min=matriz[u][v];
         xmin=x;
      }
   }
   if (xmin==0) final[i][j]=1; // Caso o ponto de menor valor seja o próprio
   else final[i][j]=254;// ponto, ele se torna um ponto da impressão afinada
}
for (int i=liix+5;i<lsix-5;i++)
for (int j=lijx+5;j<lsjx-5;j++){
   matriz[i][j]=int(final[i][j]);
}
for (int i=liix+5;i<lsix-5;i++)  // Remove pixels indesejados (ver item 2.3)
for (int j=lijx+5;j<lsjx-5;j++){
   if (matriz[i][j]==1){
      if((matriz[i-1][j]==1)||(matriz[i+1][j]==1)){
         if((matriz[i][j-1]==1)||(matriz[i][j+1]==1)) matriz[i][j]=254;
      }
   }
}
for (int i=liix+5;i<lsix-5;i++)
for (int j=lijx+5;j<lsjx-5;j++){
   if(matriz[i][j]==1){
      n=0;
         for(u=i-1;u<=i+1;u++)
      for(v=j-1;v<=j+1;v++){
         if((matriz[u][v]==1)&&((u!=i)||(v!=j))){
               n++;
            a=u-i; b=v-j;
         }
      }
      if (n==1){
         if (a==0){
               if((matriz[i-1][j-2*b]==1)||(matriz[i][j-
               2*b]==1)||(matriz[i+1][j-2*b]==1)){
               matriz[i][j-b]=1;
               if ((matriz[i][j-2*b]==1)&&((matriz[i-1][j-
               2*b]==1)||(matriz[i+1][j-2*b]==1))) matriz[i][j-2*b]=254;
            }
            else
            if((matriz[i+2][j-b]==1)||(matriz[i+2][j-2*b]==1)){
               matriz[i+1][j-b]=1;
               if ((matriz[i+2][j-b]==1)&&((matriz[i+2][j-
               2*b]==1)||(matriz[i+2][j]==1))) matriz[i+2][j-b]=254;
            }
            else
            if((matriz[i-2][j-b]==1)||(matriz[i-2][j-2*b]==1)){
               matriz[i-1][j-b]=1;
               if ((matriz[i-2][j-b]==1)&&((matriz[i-2][j-
               2*b]==1)||(matriz[i-2][j]==1))) matriz[i-2][j-b]=254;
            }
         }
         if (b==0){
               if((matriz[i-2*a][j-1]==1)||(matriz[i-2*a][j]==1)||(matriz[i-
               2*a][j+1]==1)){
               matriz[i-a][j]=1;
               if ((matriz[i-2*a][j]==1)&&((matriz[i-2*a][j-
               1]==1)||(matriz[i-2*a][j+1]==1))) matriz[i-2*a][j]=254;
            }
```

```
    else
    if((matriz[i-a][j+2]==1)||(matriz[i-2*a][j+2]==1)){
       matriz[i-a][j+1]=1;
        if ((matriz[i-a][j+2]==1)&&((matriz[i-
       2*a][j+2]==1)||(matriz[i][j+2]==1))) matriz[i-a][j+2]=254;
    }
    else
    if((matriz[i-a][j-2]==1)||(matriz[i-2*a][j-2]==1)){
       matriz[i-a][j-1]=1;
        if ((matriz[i-a][j-2]==1)&&((matriz[i-2*a][j-
       2]==1)||(matriz[i][j-2]==1))) matriz[i-a][j-2]=254;
    }

}
if (a*b!=0){
      if((matriz[i-2*a][j-2*b]==1)||(matriz[i-2*a][j-
      b]==1)||(matriz[i-a][j-2*b]==1)){
      matriz[i-a][j-b]=1;
      if (matriz[i-2*a][j-2*b]==1){
            if (matriz[i-a][j-2*b]==1){
            n=0;
            for(u=i-a-1;u<=i-a+1;u++)
                        for(v=j-2*b-1;v<=j-2*b+1;v++){
                  if (matriz[u][v]==1) n++;
            }
            if (n==3) matriz[i-a][j-2*b]=254;
         }
         if (matriz[i-2*a][j-b]==1){
            n=0;
            for(u=i-2*a-1;u<=i-2*a+1;u++)
                        for(v=j-b-1;v<=j-b+1;v++){
                  if (matriz[u][v]==1) n++;
            }
            if (n==3) matriz[i-2*a][j-b]=254;
         }
      }
      else{
            if(matriz[i-a][j-2*b]==1){
            n=0;
            for(u=i-a-1;u<=i-a+1;u++)
                        for(v=j-2*b-1;v<=j-2*b+1;v++){
                  if (matriz[u][v]==1) n++;
            }
            if(matriz[i-2*a][j-b]==1) n--;
            if ((n==3)&&(matriz[i][j-2*b]==1)) matriz[i-a][j-
            2*b]=254;
            if (n==2) matriz[i-a][j-2*b]=254;
         }
         if(matriz[i-2*a][j-b]==1){
            n=0;
            for(u=i-2*a-1;u<=i-2*a+1;u++)
                        for(v=j-b-1;v<=j-b+1;v++){
                  if (matriz[u][v]==1) n++;
            }
            if(matriz[i-a][j-2*b]==1) n--;
            if ((n==3)&&(matriz[i-2*a][j]==1)) matriz[i-2*a][j-
            b]=254;
            if (n==2) matriz[i-2*a][j-b]=254;
         }
```

```
                    }
                }
                else
                if((matriz[i][j-2*b]==1)||(matriz[i+a][j-2*b]==1)){
                   matriz[i][j-b]=1;
                    if((matriz[i][j-2*b]==1)&&(matriz[i+a][j-2*b]==1)){
                          if(matriz[i-a][j-3*b]==254) matriz[i][j-2*b]=254;
                       if((matriz[i+2*a][j-b]==254)&&(matriz[i+2*a][j-
            2*b]==254)&&(matriz[i+2*a][j-3*b]==254)) matriz[i+a][j-2*b]=254;
                    }

                }
                else
                if((matriz[i-2*a][j]==1)||(matriz[i-2*a][j+b]==1)){
                   matriz[i-a][j]=1;
                    if((matriz[i-2*a][j]==1)&&(matriz[i-2*a][j+b]==1)){
                          if(matriz[i-3*a][j-b]==254) matriz[i-2*a][j]=254;
                       if((matriz[i-a][j+2*b]==254)&&(matriz[i-
2*a][j+2*b]==254)&&(matriz[i-3*a][j+2*b]==254)) matriz[i-2*a][j+b]=254;
                    }
                }
            }
         }
      }
   }
}

//---------------------------------------------------------------------------
// Função: Remover
// Finalidade: Remover um elemento da lista de minúcias
//---------------------------------------------------------------------------
void Remover(int removido){

   for (int i=removido;i<cont-1;i++){
      lista[i].posx=lista[i+1].posx;
      lista[i].posy=lista[i+1].posy;
      lista[i].end=lista[i+1].end;
      lista[i].ang=lista[i+1].ang;
   }
}

//---------------------------------------------------------------------------
// Função: EliminarVizinhos
// Finalidade: Transforma um conjunto de minúcias próximas (provavelmente
// falsas) em uma única minúcia
//---------------------------------------------------------------------------
void EliminarVizinhos(int pos,int &gx,int &gy,int &num){
      for (int i=1;i<=viz[pos][0];i++){
      if (viz[viz[pos][i]][6]==0){
         num++;
            viz[viz[pos][i]][6]=2;
         gx+=lista[viz[pos][i]].posx;
         gy+=lista[viz[pos][i]].posy;
         EliminarVizinhos(viz[pos][i],gx,gy,num);
      }
   }
}
```

```
//---------------------------------------------------------------------------
// Função: Extract
// Finalidade: Constrói a lista de minúcias da imagem
//---------------------------------------------------------------------------
void Extract(int matriz[TAM][TAM], double rot[TAM/MASK][TAM/MASK]){

   int n;
   int dist;
   int gx;
   int gy;
   int aii,aif,aji,ajf;
   cont=0;
      for (int i=liix+10;i<lsix-10;i++)
      for (int j=lijx+10;j<lsjx-10;j++){
      if (matriz[i][j]==1){
         n=0;
            for (int u=i-1;u<=i+1;u++)
         for (int v=j-1;v<=j+1;v++)
            if (matriz[u][v]==1) n++;
         if (n==2){ // Minúcias de final de crista
            lista[cont].posx=i;
            lista[cont].posy=j;
            lista[cont].ang=rot[(int)(i/MASK)][(int)(j/MASK)];
            lista[cont].end=true;
            cont++;
         }
         if (n==4){ // Minúcias de bifurcação
            lista[cont].posx=i;
            lista[cont].posy=j;
            lista[cont].ang=rot[int(i/MASK)][int(j/MASK)];
            lista[cont].end=false;
            cont++;
         }
      }
   }
   for (int i=0;i<cont;i++){  // Elimina bifurcações próximas
      if (lista[i].end==false){
            for (int j=i+1;j<cont;j++){
            if (lista[j].end==false){
                  dist=pow(lista[i].posx-lista[j].posx,2)+pow(lista[i].posy-
                  lista[j].posy,2);
                  if (dist<9){
                        Remover(j);
                  cont--;
                  j--;
                  }
            }
         }
      }
   }
   for (int i=0;i<cont;i++){ // Elimina bifurcação próxima de final de crista
      if (lista[i].end==false){
            for (int j=0;j<cont;j++){
            dist=pow(lista[i].posx-lista[j].posx,2)+pow(lista[i].posy-
            lista[j].posy,2);
            if ((dist<=2)&&(i!=j)){
                  if (j>i){
                  Remover(j);
                  Remover(i);
               }
```

```
                else{
                   Remover(i);
                   Remover(j);
                }
                cont-=2;
                j=cont;
                i--;
             }
          }
       }
    }
    for (int i=0;i<cont;i++){  // Constrói lista de vizinhança (registra quantas
       viz[i][0]=0;   // e quem são as minúcias próximas de uma dada minúcia)
       viz[i][6]=0;
       aii=lista[i].posx-8;
       aif=lista[i].posx+8;
       aji=lista[i].posy-8;
       ajf=lista[i].posy+8;
       for (int j=0;j<cont;j++){

       if((lista[j].posx>aii)&&(lista[j].posx<aif)&&(lista[j].posy>aji)&&(lista[j
       ].posy<ajf)&&(i!=j)){
             dist=pow(lista[i].posx-lista[j].posx,2)+pow(lista[i].posy-
             lista[j].posy,2);
             if(dist<=50){
                   viz[i][0]+=1;
                viz[i][viz[i][0]]=j;
             }
          }
       }
       if(viz[i][0]==0) viz[i][6]=1;
    }
    for (int i=0;i<cont;i++){    // Elimina dois finais de crista próximos
       if (viz[i][0]==1){
             if(viz[viz[i][1]][0]==1){
             viz[i][6]=2;
             viz[viz[i][1]][6]=2;
          }
       }
    }

    for (int i=0;i<cont;i++){  // Transforma três finais em uma única minúcia
       if ((viz[i][6]==0)&&(viz[i][0]==2)){
             if ((viz[viz[i][1]][0]==2)||(viz[viz[i][2]][0]==2)){
             viz[i][6]=3;
             viz[viz[i][1]][6]=2;
             viz[viz[i][2]][6]=2;

gx=int(float((lista[i].posx+lista[viz[i][1]].posx+lista[viz[i][2]].posx)/3)+0.5)
;

gy=int(float((lista[i].posy+lista[viz[i][1]].posy+lista[viz[i][2]].posy)/3)+0.5)
;
             lista[i].posx=gx;
             lista[i].posy=gy;
          }
       }
    }

    for(int i=0;i<cont;i++){ // Remove conjuntos de minúcias
```

```
        if (viz[i][6]==0){
            int num=1;
               viz[i][6]=4;
            gx=lista[i].posx;
            gy=lista[i].posy;
            EliminarVizinhos(i,gx,gy,num);
            lista[i].posx=int(float(gx/num)+0.5);
            lista[i].posy=int(float(gy/num)+0.5);
        }
    }

    for (int i=cont-1;i>=0;i--){  // Atualiza a lista de minúcias
        if (viz[i][6]==2){
               Remover(i);
            cont--;
        }
    }


}

#endif
```

```
#ifndef _gradientes_h
#define _gradientes_h

// Essas variáveis determinam qual método será utilizado para o
// cálculo do gradiente
bool SobelCheck=false;
bool SimpleCheck=true;

//-------------------------------------------------------------
// Função: Sobel
// Finalidade: Calcula o gradiente, tanto em 'x' como em 'y',
// de um ponto de uma matriz
//-------------------------------------------------------------
int GradSobel (char escolha, int matriz[TAM][TAM], int i, int j){

      int resultado;

    if (i==0) i++;       // O método de Sobel não funciona nas
    if (i==TAM-1) i--;   // bordas. Por isso, resolvi considerar
    if (j==0) j++;       // que os gradientes dos pontos das bordas
    if (j==TAM-1) j--;   // são iguais aos seus adjacentes.

    switch (escolha){
    case 'x':{resultado=matriz[i+1][j-1]+matriz[i+1][j+1]+2*matriz[i+1][j]
       -matriz[i-1][j-1]-matriz[i-1][j+1]-2*matriz[i-1][j];break;}
    case 'y':{resultado=matriz[i-1][j+1]+matriz[i+1][j+1]+2*matriz[i][j+1]
       -matriz[i-1][j-1]-matriz[i+1][j-1]-2*matriz[i][j-1];break;}
    case 'z':{resultado=matriz[i+1][j-1]+matriz[i+1][j+1]+2*matriz[i+1][j]
       -matriz[i-1][j-1]-matriz[i-1][j+1]-2*matriz[i-1][j];
        resultado*=-1;  break;}
    }

      return(resultado);
}

//-------------------------------------------------------------
// Função: GradSimples
// Finalidade: Calcula o gradiente, tanto em 'x' como em 'y',
// de um ponto de uma matriz
//-------------------------------------------------------------
int GradSimples (char escolha, int matriz[TAM][TAM], int i, int j){

      int resultado;

    if (escolha=='z'){
      if (i==0) i++; // Não funciona na borda horizontal de cima
        resultado=matriz[i-1][j]-matriz[i][j];
    }
    else{
      if (i==TAM-1) i--; //Idem para a horizontal de baixo
      if (j==TAM-1) j--; //idem para a vertical da direita

      if (escolha=='x'){
                resultado=matriz[i+1][j]-matriz[i][j];
      }
      else{
                resultado=matriz[i][j+1]-matriz[i][j];
```

```
        }
      }
      return(resultado);
}

//-------------------------------------------------------------
// Função: Gradiente
// Finalidade: Calcula o gradiente, tanto em 'x' como em 'y',
// de um ponto de uma matriz, utilizando um dos métodos acima,
// de acordo com configuração do usuário
//-------------------------------------------------------------
int Gradiente (char escolha, int matriz[TAM][TAM], int i, int j){

    int resultado;

      if (SobelCheck) resultado=GradSobel(escolha,matriz,i,j);
    if (SimpleCheck) resultado=GradSimples(escolha,matriz,i,j);

    return(resultado);
}

#endif
```

A.2 – Rede Neural

A.2.1. Arquivo.h

```
//---------------------------------------------------------------------------

#ifndef _arquivos_h
#define _arquivos_h

#include <string.h>
#include <stdio.h>
#include <stdlib>
#include <time.h>
#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "red.h"

// nome dos arquivos de entrada do programa
char  nome_arq1[50],nome_arq2[50],arq_saida[50],arq_entrada[50];


//---------------------------------------------------------------------------
// Rotina que carrega as impressões digitais durante o treinamneto
//---------------------------------------------------------------------------
void abre_arq(Timage *imagem1 ,int num){

 char arq1[100];
 char par[1];
 Form3->Edit5->GetTextBuf(arq_entrada,50);
 itoa(num,par,10);
 strcat(arq_entrada,"finger");

 strcat(arq_entrada,par);
 strcat(arq_entrada,".bmp");
    imagem1->Picture->LoadFromFile(arq_entrada);


}

//---------------------------------------------------------------------------
//  Função que carrega as saídas desejadas
//---------------------------------------------------------------------------
void carrega_saida(FILE *nome_arq4, int *saidas_des1){

 int i;
 for(i=0;i<5;i++)
  fscanf(nome_arq4,"%1d",&saidas_des1[i]);
 }

//---------------------------------------------------------------------------
//  Função que salva em arquivos os pesos calculados pelo treinamento
//---------------------------------------------------------------------------

void salva_pesos(void){

FILE *pesosprim;
FILE *pesosseg;
int i,j;
pesosprim = fopen("c:\\pesosprim.txt","w");
```

```
   pesosseg = fopen("c:\\pesosseg.txt","w");

   for(i=0;i<1024;i++)
            for(j=0;j<15;j++)
             fprintf(pesosprim,"%f\n",pesos_prim[i][j]);
   fclose(pesosprim);
   for(i=0;i<15;i++)
            for(j=0;j<5;j++)
       fprintf(pesosseg,"%f\n",pesos_seg[i][j]);
   fclose(pesosseg);

   }


   //---------------------------------------------------------------------------
   // Função que carrega o pesos corretos para a matriz
   //---------------------------------------------------------------------------
   int carrega_pesos(void){
      int i,j;
      FILE *pesosprim;
      FILE *pesosseg;

      if((pesosprim = fopen(nome_arq1,"r"))== NULL){
       Form2->Edit1->Text = "Arquivo Inexistente";
       Form3->Edit3->Text = "Arquivo Inexistente";
       Form3->Edit3->Update();
       return 0;
       }
      if((pesosseg  = fopen(nome_arq2,"r"))== NULL){
       Form2->Edit2->Text = "Arquivo Inexistente";
       Form3->Edit4->Text = "Arquivo Inexistente";

       return 0;
      }
      for(i=0;i<1024;i++)
      for(j=0;j<15;j++)
            fscanf(pesosprim,"%lf\n",&pesos_prim[i][j]);
      fclose(pesosprim);
            for(i=0;i<15;i++)
      for(j=0;j<5;j++)
            fscanf(pesosseg,"%lf\n",&pesos_seg[i][j]);
      fclose(pesosseg);
      return 1;
   }

   //---------------------------------------------------------------------------
   // Rotina que cadastra uma nova saída desejada no arquivo de saídas desejadas
   //---------------------------------------------------------------------------
   void cad(void){

    int des1,des2,des3,des4,des5;
    FILE *saidas_corretas;

    des1 = StrToInt(Form1->clas1->Text);
    des2 = StrToInt(Form1->clas2->Text);
    des3 = StrToInt(Form1->clas3->Text);
    des4 = StrToInt(Form1->clas4->Text);
    des5 = StrToInt(Form1->clas5->Text);
    saidas_corretas = fopen("c:\\temp\\piau\\saidasdes.txt","a");
```

```
  fprintf(saidas_corretas,"%d",des1);
  fprintf(saidas_corretas,"%d",des2);
  fprintf(saidas_corretas,"%d",des3);
  fprintf(saidas_corretas,"%d",des4);
  fprintf(saidas_corretas,"%d\n",des5);
  fclose(saidas_corretas);
 }


 #endif
 //---------------------------------------------------------------------------
```

A.2.2. Red.h

```
 //---------------------------------------------------------------------------

 #ifndef _transf_h
 #define _transf_h

 #define tam 32
 #include <math.h>


 // Variaveis dos pesos das camadas
 double pesos_prim[1024][15],pesos_seg[15][5];
 //Variaveis com a entrada e saidas das camadas
 double vet_ent[1024],saida2[15],saida3[5],saida[5];


 //---------------------------------------------------------------------------
 // Função que calcula o vetor de saída da camada escondida
 //---------------------------------------------------------------------------
 void saida1(double saida2_temp[15],double entrada[1024]){

 int i,j;

  for (i=0;i<15;i++)
    for(j=0;j<1024;j++)
          saida2_temp[i] += (entrada[j] * (pesos_prim[j][i]));

  for (i=0;i<1024;i++)
          saida2_temp[i] = 1/(1+exp(-saida2_temp[i]));
 }

 //---------------------------------------------------------------------------
 // Função que calcula a saída final da rede
 //---------------------------------------------------------------------------

 void saidafim(double saida2_temp[15],double saida3_temp[5],double saida[5]){

 int i,j;

  for (i=0;i<5;i++)
    for(j=0;j<15;j++)
          saida3[i] += saida2[j] * (pesos_seg[j][i]);
```

```
    for(i=0;i<5;i++)
            saida[i] = 1/(1+exp(-saida3[i]));

  }

  //---------------------------------------------------------------------------
  // Rotina que mostra na tela a classificação da impressão digital
  //---------------------------------------------------------------------------
  void mostra_valor(TEdit *clas1,TEdit *clas2,TEdit *clas3,TEdit *clas4,TEdit *clas5){

    double maximo=0;
    int classe,i;

            for (i=0;i<5;i++)
     if (saida[i]> maximo){
            classe = i;
       maximo = saida[i];
     }
     switch (classe){

     case 0:
                     Form1->Edit4->Text = "Whrol";
          break;
     case 1:
          Form1->Edit4->Text = "Arch";
          break;
     case 2:
                     Form1->Edit4->Text = "Tended Arch";
          break;
             case 3:
                     Form1->Edit4->Text = "Loop Direito";
          break;
    case 4:
                     Form1->Edit4->Text = "Loop Esquerdo";
          break;

    }
}


#endif

//*****************************Fim da Rede Neural em Si************************
```

A.2.3. Trein.h

```
//---------------------------------------------------------------------------

#ifndef _trein_h
#define _trein_h

#include <math.h>
#include "arquivos.h"
```

```
#define eta 0.8
#define alfa 1

// Variaveis com as correções dos pesos
double delta_prim[1024][15],delta_seg[15][5];


//-------------------------------------------------------------------------
// Função que zera as matrizes de saida das camadas para recomeço do processo
//-------------------------------------------------------------------------

void zeramatriz(void){

int i;

for(i=0;i<20;i++)
        saida2[1]=0;
for(i=0;i<5;i++){
        saida3[i]=0;
   saida[i]=0;
}

}


//-------------------------------------------------------------------------
// função que inicializa os pesos da rede
//-------------------------------------------------------------------------
void inipesos(void){

int i,j;

for (i=0;i<15;i++)
        for (j=0;j<1024;j++){
   pesos_prim[j][i] = (1+random(5))*0.1;
   delta_prim[j][i] = 0;
   }
for (i=0;i<5;i++)
        for (j=0;j<15;j++){
   pesos_seg[j][i] = (1+random(5))*0.1;
   delta_seg[j][i] = 0;
   }

}


//-------------------------------------------------------------------------
//Função que recalcula os pesos entre a camada de saída e a escondida
//-------------------------------------------------------------------------

void recalque_seg(int saidas_des1[5]){

int i,j;


for(i=0;i<5;i++)
        for(j=0;j<15;j++){
                delta_seg[j][i]=eta*(saidas_des1[i]-saida[i])*(saida[i]*(1-saida[i])*saida2[jj] +
alfa*delta_seg[j][i];
```

```
                pesos_seg[j][i] += delta_seg[i][j];
    }

}

//-------------------------------------------------------------------
// Função que recalcula os pesos entre a camada escondida e a camada de Entrada
//-------------------------------------------------------------------

void recalque_prim(int saidas_des1[5]){

double soma;
int i,j,k;


for(i=0;i<15;i++)
for(j=0;j<1024;j++){
        for(k=0,soma=0;k<5;k++)
                soma+= (saidas_des1[k]-saida[k])*pesos_seg[k][i];
   soma *=saida2[i]*(1-saida2[i]);
   delta_prim[j][i] = eta* vet_ent[j]*(1-vet_ent[j])*soma + alfa*delta_prim[j][i];
   pesos_prim[j][i]+=delta_prim[j][i];

}
}


#endif
/*******************************Fim do Treinamento em Si************************/
```

A.2.4. Ima.h

```
//-------------------------------------------------------------------

#ifndef _funcoes_h
#define _funcoes_h

#define TAM 512
#define MASK 16
#include <math.h>



//-----------------------------------------------------------
// Função: BmpToInt
// Finalidade: Recebe uma imagem, que deve ter o tamanho de
// TAMxTAM pixels, e constrói uma matriz de mesmo tamanho com
// os valores inteiros (0 a 255) que representam os tons de
// cinza dos pixels
//-----------------------------------------------------------
double aux[TAM][TAM];
double aux1[32][32];
int matriz[TAM][TAM];
struct angulo{
double s,c;
};
```

```
angulo ang[TAM][TAM];

//---------------------------------------------------------------------------
// Rotina que transforma a imagem de tons de cinza para uma matriz de inteiros
//---------------------------------------------------------------------------

void BmpToInt(TImage *imagem,int matriz[TAM][TAM]){
        for (int i=0;i<TAM;i++)
        for (int j=0;j<TAM;j++)
        matriz[i][j]=imagem->Canvas->Pixels[j][i]&0xFF;
}

//---------------------------------------------------------------------------
// Função: IntToBmp
// Finalidade: Exatamente o processo contrário da função ante-
// rior, BmpToInt.
//---------------------------------------------------------------------------
void IntToBmp(TImage *image, int matriz[TAM][TAM]){
   int h,w;
   h=image->Height;
   w=image->Width;
   image->Visible=false;
   image->Height=TAM;
        image->Width=TAM;
        for(int i=0;i<TAM;i++)
        for (int j=0;j<TAM;j++)
        image->Canvas->Pixels[j][i]=matriz[i][j]*65793;
   image->Height=h;
   image->Width=w;
   image->Visible=true;
}

//---------------------------------------------------------------------------
// Função que calcula o campo de orientação
// das cristas e dos vales
//---------------------------------------------------------------------------

void CalCampo(int matriz[TAM][TAM]){


   int s[9];
   int kmin;
   int kmax;
   int smax=0;
   int smin=1021;
   int soma=0;
   angulo media;
   media.c=0;
   media.s=0;
   for(int i=4;i<TAM-4;i++)
   for(int j=4;j<TAM-4;j++){
        s[1]=matriz[i-4][j]+matriz[i-2][j]+matriz[i+2][j]+matriz[i+4][j];
     s[2]=matriz[i-4][j+2]+matriz[i-2][j+1]+matriz[i+2][j-1]+matriz[i+4][j-2];
     s[3]=matriz[i-4][j+4]+matriz[i-2][j+2]+matriz[i+2][j-2]+matriz[i+4][j-4];
     s[4]=matriz[i-2][j+4]+matriz[i-1][j+2]+matriz[i+1][j-2]+matriz[i+2][j-4];
     s[5]=matriz[i][j-4]+matriz[i][j-2]+matriz[i][j+2]+matriz[i][j+4];
     s[6]=matriz[i-2][j-4]+matriz[i-1][j-2]+matriz[i+1][j+2]+matriz[i+2][j+4];
     s[7]=matriz[i-4][j-4]+matriz[i-2][j-2]+matriz[i+2][j+2]+matriz[i+4][j+4];
     s[8]=matriz[i-4][j-2]+matriz[i-2][j-1]+matriz[i+2][j+1]+matriz[i+4][j+2];
```

```
      for(int k=1;k<=8;k++){
          soma+=s[k];
          if (s[k]>smax) {smax=s[k]; kmax=k;}
         if (s[k]<smin) {smin=s[k]; kmin=k;}
      }
      if ((4*matriz[i][j]+smax+smin)>3*soma/8){
      aux[i][j]=255;
      switch (kmax) {
      case 1: {ang[i][j].s=sin(3.14159);
                    ang[i][j].c=cos(3.14159);break;}
      case 2: {ang[i][j].s=sin(2*3.14159/3);
                    ang[i][j].c=cos(2*3.14159/3);break;}
      case 3: {ang[i][j].s=sin(2*3.14159/4);
                    ang[i][j].c=cos(2*3.14159/4);break;}
      case 4: {ang[i][j].s=sin(2*3.14159/6);
                    ang[i][j].s=sin(2*3.14159/6);break;}
      case 5: {ang[i][j].s=0;
                    ang[i][j].c=1;    break;}
      case 6: {ang[i][j].s=sin(10*3.14159/6);
            ang[i][j].c=cos(10*3.14159/6);break;}
      case 7: {ang[i][j].s=sin(6*3.14159/4);
                    ang[i][j].c=cos(6*3.14159/4);break;}
      case 8: {ang[i][j].s=sin(4*3.14159/3);
                    ang[i][j].c=cos(4*3.14159/3);break;}
      }
      }
      else{
      aux[i][j]=0;
      switch (kmin) {
          case 1: {ang[i][j].s=sin(3.14159);
                    ang[i][j].c=cos(3.14159);break;}
      case 2: {ang[i][j].s=sin(2*3.14159/3);
                    ang[i][j].c=cos(2*3.14159/3);break;}
      case 3: {ang[i][j].s=sin(2*3.14159/4);
                    ang[i][j].c=cos(2*3.14159/4);break;}
      case 4: {ang[i][j].s=sin(2*3.14159/6);
                    ang[i][j].s=sin(2*3.14159/6);break;}
      case 5: {ang[i][j].s=0;
                    ang[i][j].c=1;    break;}
      case 6: {ang[i][j].s=sin(10*3.14159/6);
            ang[i][j].c=cos(10*3.14159/6);break;}
      case 7: {ang[i][j].s=sin(6*3.14159/4);
                    ang[i][j].c=cos(6*3.14159/4);break;}
      case 8: {ang[i][j].s=sin(4*3.14159/3);
                    ang[i][j].c=cos(4*3.14159/3);break;}

      }
      }
      soma=0; smax=0; smin=1021;
}
for(int i=4;i<TAM-4;i++)
for(int j=4;j<TAM-4;j++)
        matriz[i][j]=aux[i][j];
for (int u=1;u<TAM/MASK-1;u++)
for (int v=1;v<TAM/MASK-1;v++){
        for (int i=0;i<MASK;i++)
     for (int j=0;j<MASK;j++){
      media.c+=ang[i+u*MASK][j+v*MASK].c;
      media.s+=ang[i+u*MASK][j+v*MASK].s;
```

```
        }
        media.c/=pow(MASK,2);
        media.s/=pow(MASK,2);
        if (media.s>0)
          aux1[u-1][v-1]=acos(media.c)/2;
        else
           aux1[u-1][v-1]= 3.1415 - acos(media.c)/2;
        media.c=0;
        media.s=0;
      }
      for(int i=0;i<TAM/MASK-2;i++)
      for(int j=0;j<TAM/MASK-2;j++);



//----------End of Orientation Field-------------
}


#endif
//---------------------------------------------------------------------------
```

A.2.5. Programa Principal

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "arquivos.h" /* includes que carregam as varias bliotecas de funções*/
#include "cadastro.h" /* usadas pelo programa */
#include "ima.h"
#include "red.h"
#include "trein.h"
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <stdlib>
#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"



//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
```

```
{
}

//---------------------------------------------------------------------------
// Rotina que carrega um arquivo do tipo bmp
//---------------------------------------------------------------------------
void __fastcall TForm1::OpenClick(TObject *Sender)
{
        if (OpenDialog1->Execute())
        Image1->Picture->LoadFromFile(OpenDialog1->FileName);
}

//---------------------------------------------------------------------------
// Rotina que binariza uma certa impressão digital
//---------------------------------------------------------------------------
void __fastcall TForm1::BinarizarClick(TObject *Sender)
{
  BmpToInt(Image1,matriz);
  CalCampo(matriz);
  IntToBmp(Image2,matriz);
}

//---------------------------------------------------------------------------
// Rotina que treina a rede
//---------------------------------------------------------------------------
void __fastcall TForm1::TreinarRedeClick(TObject *Sender)
{
 Edit1->Text = "Processando";
 Edit1->Update();
 Form3->Enabled=true;     // Form onde serão digitadas as entradas necessárias
 Form3->Visible=true;
}

//---------------------------------------------------------------------------
// Rotina que classfica uma impressão digital
//---------------------------------------------------------------------------
void __fastcall TForm1::ClassificarClick(TObject *Sender)
{
  Edit1->Text = "Processando";
  Edit1->Update();
  if (OpenDialog1->Execute())
        Image1->Picture->LoadFromFile(OpenDialog1->FileName);
  Image1->Visible=true;
        BmpToInt(Image1,matriz);
        CalCampo(matriz);
        IntToBmp(Image2,matriz);

        Form2->Enabled=true;  // Form onde serão digitadas as entradas necessárias
 Form2->Visible=true;

}

//---------------------------------------------------------------------------
// Rotina que cadastra uma nova impressão
//---------------------------------------------------------------------------
void __fastcall TForm1::CadastrarClick(TObject *Sender)
{
 cad();
}
```

```
//---------------------------------------------------------------------------
// Rotina que termina o treinamento da rede
//---------------------------------------------------------------------------
void __fastcall TForm1::Treinar_rede1Click(TObject *Sender)
{

        FILE *nome_arq3;
        double err[5],errqua=1;
  int i,j,par,total,saidas_des[5];
  clock_t start;
  par = 0;


  start = clock();
  inipesos();
  salva_pesos();

  Form3->Edit1->GetTextBuf(arq_saida,50); // Leitura dos nomes dos
  Form3->Edit3->GetTextBuf(nome_arq1,50); // arquivos necessarios ao
  Form3->Edit4->GetTextBuf(nome_arq2,50); // treinamento
  total = StrToInt(Form3->Edit2->Text);



  if((nome_arq3 = fopen(arq_saida,"r"))== NULL){ // Verificação da existencia
    Form3->Edit1->Text = "Arquivo Inexistente";  // do arquivo
        return;
  }
  else{

     while((errqua>0.5) & (par<=total)){  // Loop de treinamento
         rewind(nome_arq3);
     if(carrega_pesos()==0)
        return;
         for(par=1;par<=total;par++){
     abre_arq(Image1,par);
        carrega_saida(nome_arq3,saidas_des);
     Form3->Enabled=false;
                          Form3->Visible=false;
        BmpToInt(Image1,matriz);
        CalCampo(matriz);
        IntToBmp(Image2,matriz);
        Image1->Update();
     Image2->Update();
     Edit3->Text = IntToStr(par);
     Edit3->Update();
     for(i=0;i<32;i++)
        for(j=0;j<32;j++)
                vet_ent[j+i*32] = aux1[i][j];
     zeramatriz();
     saida1(saida2,vet_ent);
     saidafim(saida2,saida3,saida);
     mostra_valor(clas1,clas2,clas3,clas4,clas5);
     clas1->Update();
     clas2->Update();
     clas3->Update();
     clas4->Update();
     clas5->Update();
```

```
        for(i=0;i<5;i++)
           err[i]=saidas_des[i]-saida[i];
        errqua = (pow(err[1],2)+pow(err[2],2)+pow(err[3],2)+pow(err[4],2)+pow(err[5],2))/2;

        if(errqua>0.5){
          recalque_seg(saidas_des);
          recalque_prim(saidas_des);
          salva_pesos();
          }

       }
    }
    }
    fclose(nome_arq3);
    Edit1->Text= "Salvando Pesos";
    Edit1->Update();
    salva_pesos();
    Edit1->Text= "Fim do Treinamento";
    Image1->Visible=false;
    Image2->Visible=false;
    Edit2->Text = (clock()-start)/CLK_TCK;  // Calculo do tempo de processamento
    Edit3->Text = "";
    Edit4->Text = "";
}

//---------------------------------------------------------------------------
// Rotina que finaliza a classificação
//---------------------------------------------------------------------------
void __fastcall TForm1::Classificar1Click(TObject *Sender)
{
        int i,j;
    clock_t comeco;

    comeco = clock();
    for (i=0;i<32;i++)
        for(j=0;j<32;j++)
        vet_ent[j+i*32] = aux1[i][j];
    Form2->Edit1->GetTextBuf(nome_arq1,50);
    Form2->Edit2->GetTextBuf(nome_arq2,50);
    if(carrega_pesos()==0)
        return;
    else{
       Form2->Enabled=false;
       Form2->Visible=false;
       zeramatriz();
       saida1(saida2,vet_ent);
       saidafim(saida2,saida3,saida);
       mostra_valor(clas1,clas2,clas3,clas4,clas5);

       }
    Edit1->Text = "Fim do Processamento";
    Edit1->Update();
    Edit2->Text = ((clock()-comeco)/CLK_TCK); //Calcula o tempo de processamento

}
//---------------------------------------------------------------------------
```