Universidade Federal de Santa Catarina
Centro de Blumenau
Departamento de Engenharia de
Controle e Automação e Computação

Fellipe Eduardo Gonçalves da Silva

Sistema de identificação biométrica baseado em extração de minúcias e redes neurais artificiais

Blumenau
2019

Fellipe Eduardo Gonçalves da Silva

Sistema de identificação biométrica baseado em extração de minúcias e redes neurais artificiais

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como parte dos requisitos necessários para a obtenção do Título de Engenheiro de Controle e Automação.
Orientador: Prof. Dra. Janaína Gonçalves Guimarães

Universidade Federal de Santa Catarina
Centro de Blumenau
Departamento de Engenharia de
Controle e Automação e Computação

Blumenau
2019

Fellipe Eduardo Gonçalves da Silva

Sistema de identificação biométrica baseado em extração de minúcias e redes neurais artificiais

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como requisito parcial para a obtenção do título de Engenheiro de Controle e Automação.

Comissão Examinadora

Prof. Dra. Janaína Gonçalves Guimarães
Universidade Federal de Santa Catarina
Orientador

Prof. Dr. Ciro André Pitz
Universidade Federal de Santa Catarina

Prof. Dr. Marcos Vinicius Matsuo
Universidade Federal de Santa Catarina

Blumenau, 26 de junho de 2019

Dedico este trabalho aos meu pais, aos meus irmãos,
meus amigos e todos aqueles que, de alguma forma,
auxiliaram para a concretização desta etapa.

Agradecimentos

Agradeço principalmente aos meus pais, Carmem e Marcos, que sempre me apoiaram e investiram na minha educação e evolução. Agradeço aos meus irmãos, me ensinaram o valor da família e me deram suporte para finalização desta etapa.

Agradeço aos meus amigos, principalmente ao Caique, Bianca, Ciro, Eduardo, Nikolas e Célio. Eles me permitiram compartilhar grandes momentos durante esses anos de graduação.

Agradeço aos professores da UFSC que se dedicaram a passar adiante o seu conhecimento. Agradeço, principalmente, a Professora Janaina, que me orientou e me deu suporte em cada fase desse projeto e me guiou para à conclusão do curso.

"Somewhere, something incredible is waiting to be known."
(Carl Sagan)

Resumo

O objetivo deste projeto foi a implementação de um sistema de reconhecimento de impressões digitais baseado no método extração de minúcias e utilizando uma rede neural artificial do tipo Perceptron para a classificação dos padrões. As imagens de entrada foram pré-processadas para obter a definição das linhas dactilares e extrair os pontos característicos de cada impressão digital. As técnicas de processamento dos dados e arquitetura da rede neural foram implementadas através do MATLAB, que possui bibliotecas e funções especializadas para a aplicação deste trabalho.

O sistema foi implementado e integrado por meio de um algoritmo descrito em *Python* e realizou o reconhecimento dos indivíduos previamente cadastrados. Para obtenção do resultado se mostrou importante escolher adequadamente técnicas de pré-processamento das imagens de entrada do sistema, e a forma de manipulação dos dados de entrada para o treinamento da rede neural.

A arquitetura final da rede foi definida como Perceptron multicamadas, com 10 neurônios na camada oculta. Durante o treinamento apresentou 95.1% de acertos, e com 75% de acertos no reconhecimento de indivíduos no sistema total. Além disso, o projeto apresentou 10% de falsas aceitações, possibilidade de falso reconhecimento.

Palavras-Chave: 1. Sistemas Biométricos. 2. Impressão digital. 3. Extração de Minúcias. 4. Redes Neurais Artificais. 5. Perceptron multicamadas.

Abstract

The objective of this project was to implement a system of fingerprint recognition based in minutiae extraction method using Perceptron model of artificial neural network for pattern classification. The data processing and neural netwokrk architecture techniques were implemented through MATLAB, which has specialized libraries and functions for the application of this project.

The system was deployed and integrated by a *Python* algorithm and accomplished the recognition of previously registered individuals. To obtain the result of this project was important to choose properly techniques of pre-processing the input images of the system, and the scheme of manipulation the input data for the training of neural networks.

The final network architecture was defined as multilayer Perceptron, with 10 neurons in the hidden layer. During the training it presented 95.1% of correct answers, and with 75% of correct answers in the recognition of individuals in the total system. In addition, the project presented 10% of false acceptances, possibility of false recognition.

Keywords: 1. Biometric Systems. 2. Fingerprint. 3. Minutiae extraction. 4. Artificial Neural Networks. 5. Multilayer Perceptron.

Lista de figuras

Lista de Siglas e Abreviaturas

MLP	*Multi Layer Perceptron*
API	*Automated Personal Identification*
RNA	*Redes Neurais Artificiais*
RNN	*Redes Neurais Naturais*
FTIR	*Frustated Total Internal Reflection*
BMP	*BitMap*
ID	*Indivíduo*
LM	*Levenberg-Marquardt*
RP	*Resilient Backpropagation*

Sumário

1 Introdução

O termo Biometria, no início do século XX, denominava métodos estatísticos aplicados na biologia evolutiva. As suas pesquisas eram relacionadas à variação interna e externa de grandes populações, a compreensão estatística do fenômeno de massa ou fatores efetivos na evolução humana [1].

A partir de 1980 o termo Biometria, que tinha o significado de buscar conhecimento biológico por métodos quantitativos, passou a englobar o conceito de identificação pessoal automatizada, ou *Automated Personal Identification* (API). O campo de atuação da API engloba tecnologias para diferenciar características individuais dos seres humanos, como por exemplo, reconhecimento de voz, retina, impressões digitais, face, assinatura, comprimento dos dedos, entre outros. Com essa mudança, o indivíduo que antes não tinha importância, era apenas tratado como um dado entre toda a linha evolutiva, passa a ser reconhecido como um ser único e distinguível [2].

Na atualidade, em uma era da informação, a sociedade vem se tornando um sistema eletronicamente conectado. Nessa sociedade avançou o desenvolvimento do tecnologias que facilitam a vida das pessoas no cotidiano. Dentre esses sistemas se destacam processamento digital de sinais, internet, transações financeiras realizadas por redes de computadores, entre outros fatores. Nesse contexto, houve a necessidade da criação de uma identidade que representasse cada indivíduo de forma definida e circunscrita.

Dentre as áreas de estudo da Biometria, as impressões digitais são amplamente utilizadas no reconhecimento de indivíduos através de mecanismos de identificação e autentificação. A sua análise foi realizada por muito tempo através da inspeção visual. Nos últimos anos, com a evolução dos sistemas microprocessados e embarcados, a biometria baseada em impressões digitais foi automatizada, fornecendo respostas rápidas, precisas e seguras [3].

O reconhecimento e classificação de padrões de uma impressão digital é realizado através de duas principais técnicas. A principal característica da biometria digital são os pontos locais, denominados de minúcias. A outra técnica de reconhecimento é obtida através da análise global da impressão digital, combinando as minúcias são encontrados os pontos singulares.

Para a extração das minúcias, a visão computacional apresenta como a melhor alternativa, pois oferece técnicas de pré-processamento capazes de realçar as características das linhas dactilares e possibilita obter as informações necessárias que compões as minúcias.

Redes Neurais Artificiais (RNA) são técnicas computacionais que adquirem conhecimento utilizando modelos matemáticos inspirados na estrutura neural dos organismos inteligentes [4]. Um dos modelos de RNAs mais utilizados é o Perceptron Multicama-

das, do inglês *Multi Layer Perceptron* (MLP), cujo uma das principais aplicações é o reconhecimento e classificação de padrões através de classes previamente definidas.

A utilização das redes neurais para identificação biométrica possibilita a identificação do indivíduo mesmo quando apresenta leves cicatrizes ou pequenas modificações de suas características. Além disso, a partir do treinamento da rede, pode ser definido o reconhecimento para diferentes posicionamentos da impressão obtida pelo sensor.

1.1 Objetivos

1.1.1 Objetivo Geral

Este trabalho tem como objetivo a criação do protótipo de um sistema de reconhecimento de impressões digitais baseado no método de extração das minúcias, capaz de classificar um conjunto de diferentes pessoas utilizando uma rede neural artificial multicamadas, do tipo Perceptron. Para a extração dos pontos locais será utilizando método de pré-processamento utilizando técnicas de visão computacional.

1.1.2 Objetivos específicos

Para a execução do objetivo principal, foram definidos os seguintes objetivos específicos:

- revisar os métodos de padronização e classificação de impressões digitais;
- implementar técnicas de tratamento para o pré-processamento das imagens;
- desenvolver o método de extração das minúcias;
- encontrar arquitetura da rede neural para a aplicação do trabalho;
- implementar o sistema de reconhecimento biométrico.

2 Revisão de Literatura

2.1 Redes Neurais

O cérebro pode ser classificado como um computador altamente complexo, não-linear e paralelo. Ele tem a capacidade de organizar seus constituintes estruturais de forma a realizar processamento rapidamente [5]. As principais unidades do cérebro são os neurônios, e é por meio dessas células que os dados são transmitidos e processados.

Diversos pesquisadores tentaram simular processos do cérebro, com o intuito de reproduzir situações previamente ocorridas. A partir dessas pesquisas, diversos processos como aprendizagem por experiência, classificação e reconhecimento de padrões já foram estudados e desenvolvidos nesse sentido.

Inspirado no sistema nervoso biológico dos seres vivos, em 1943, Warren McCulloch e Walter Pitts criaram um modelo computacional de um neurônio artificial, surgindo então o primeiro neurônio artificial. Posteriormente, obteve-se um sistema com vários neurônios interconectados, sendo chamado de Redes Neurais Artificiais [6].

As redes neurais são modelos computacionais de um circuito cerebral, sendo uma estrutura de processamento composta por um número de neurônios artificiais interconectados, em que cada neurônio apresenta um comportamento específico, determinado por sua função de transferência, interconexões, e possivelmente pelas entradas externas [7].

2.1.1 Inspiração Biológica

O sistema nervoso está presente em todos os organismos multicelulares, variando a sua complexidade e organização. A partir dele, seres como lesmas, insetos, humanos, entre outros, conseguem adaptar seu comportamento e armazenar informações.

A descoberta do italiano Camillo Golgi, em 1875, foi um dos primeiros passos da neuroanatomia [8]. A partir do acaso, foi encontrado um método em que é possível isolar individualmente os neurônios, e assim observá-los. Entretanto, Santiago Ramón y Cajal revelou a comunicação entre as células pela sinapse, e a interconexão entre neurônios, introduzindo a idéia dos neurônios como constituintes estruturais do cérebro.

Em termos de implementação física, os neurônios são 5 vezes mais lentos se comparados a portas lógicas digitais implementadas em silício. Entretanto, o cérebro humano possui uma grande quantidade de neurônios, com conexões massivas entre si. O cérebro humano, possui cerca de 10 bilhões de neurônios, onde são feitas 60 trilhões de sinapses ou conexões. Desse modo, a eficiência energética do cérebro é muito superior aos melhores computadores em uso atualmente [5, 9].

O neurônio é formado por três partes principais: a soma, também chamada de corpo celular, onde deriva ramificações denominadas de dentritos, e outra ramificação extensa chamada de axônio. A extremidade dos axônios é composta pelos nervos terminais, onde é realizada a transmissão das informações para outros neurônios, definida como sinapse [10].

O corpo celular e os dentritos constituem a superfície de entrada do neurônio e axônio à superfície referente a saída do fluxo de informação. Pode ser analisado as partes do neurônio e o fluxo da informação, por meio de setas, na Figura 1.

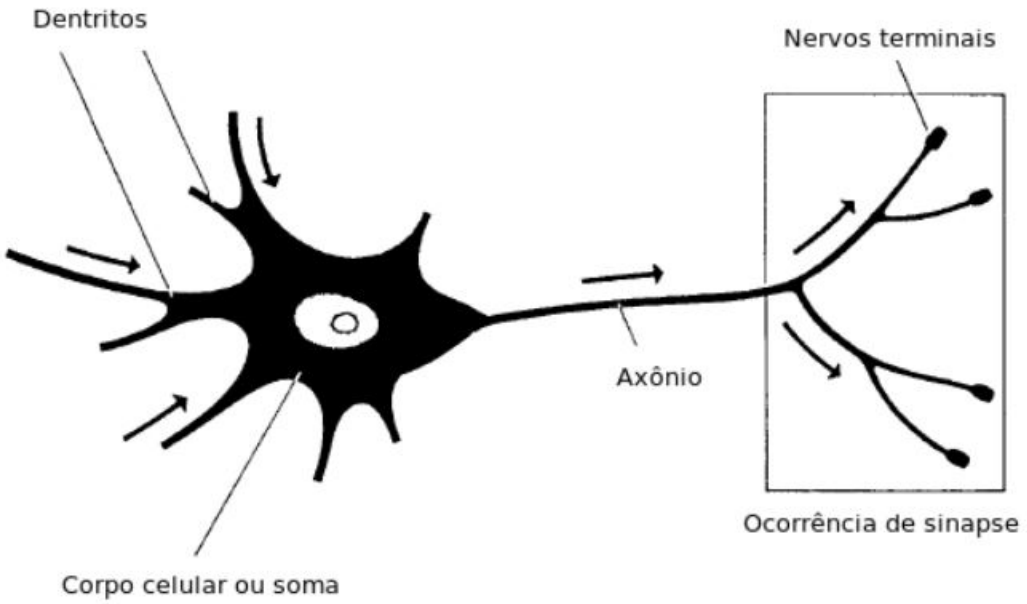

Figura 1 – Estrutura de um neurônio biológico [6].

As sinapses são elementos estruturais e funcionais que permeiam as interações entre os neurônios, possuindo o papel fundamental na memorização da informação. A cada sinapse ativada, o número de neurotransmissores liberados aumenta, expandindo a conexão entre os dois neurônios. Esse processo é denominado de facilitação, inspirado na Lei de Hebb: *"A intensidade de uma conexão sináptica entre dois neurônios aumenta quando os dois neurônio estão excitados simutaneamente"*. Sendo então, a representação base para o desenvolvimento do algoritmo de aprendizagem de uma rede neural artificial (RNA) [4].

2.1.2 Redes Neurais Artificiais

A rede neural é constituída por um agrupamento de neurônios interligados, estimulando uns aos outros e produzindo um sistema maior capaz de armazenar conhecimento por meio de padrões previamente apresentados, e assim, encontrar uma solução para novas situações, desconhecidas [6].

O modelo das redes neurais artificiais (RNAs) foi inspirado nos neurônios biológicos e nos sistemas nervosos. Porém, é válido resaltar que RNA está distante das redes neurais naturais (RNNs).

A estrutura de uma RNA mais difundida foi proposta por McCulloch e Pitts (1943), denominada como Perceptron multicamadas. Os impulsos elétricos descendentes de outro neurônio são denominados sinais de entrada (x_j), em que os sinais serão processados de maneira diferente, e essa medida é representada através dos *pesos sinápticos*. Os pesos sinápticos são representados por w_{kj}, onde k é referente ao índice do neurônio e j ao terminal de entrada da sinapse [6].

O corpo celular, ou a soma, é representado por uma combinação linear, uma soma ponderada referente aos sinais de entradas pelas respectivas sinapses do neurônio, e o estágio de ativação, ou função restritiva [$\varphi(.)$], em que restringe a amplitude de saída de um neurônio. O axônio é representado pela saída (y_k) como resposta da função de ativação. O modelo inclui também um *bias* aplicado externamente (b_k), podendo aumentar ou diminuir o sinal de entrada da função de ativação [5].

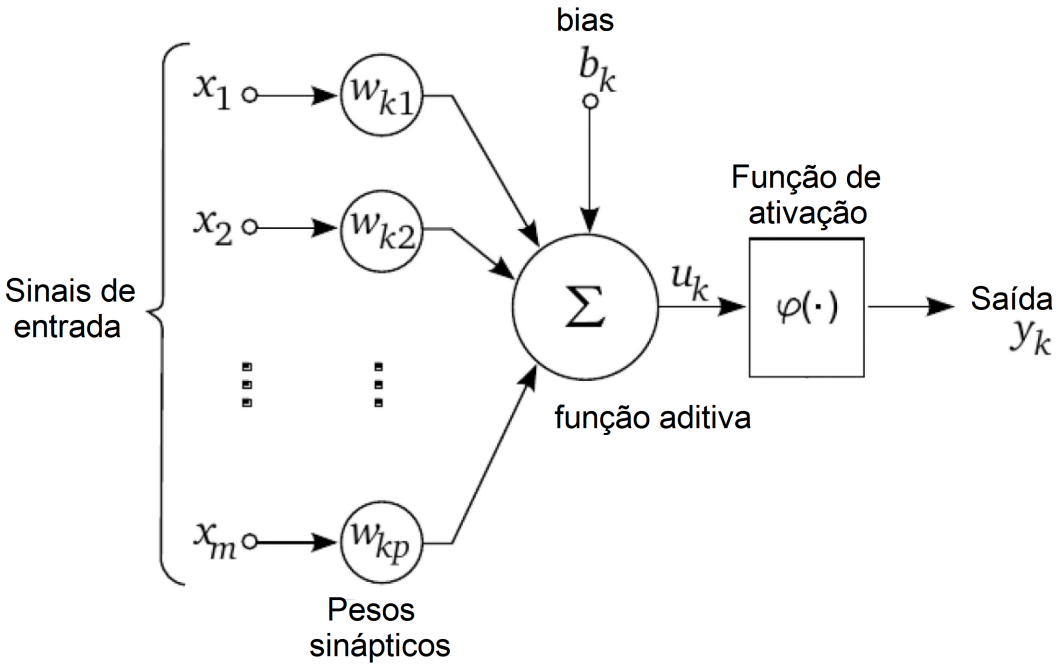

Figura 2 – Modelo de um neurônio proposto por McCulloch e Pitts.

O neurônio artificial pode ser representado , em termos matemáticos, como:

$$u_k = \sum_{j=1}^{m} w_{kj} \cdot x_j \tag{2.1}$$

no qual m é o número de entradas no neurônio k, e adiante a função de ativação:

$$y_k = \varphi(u_k) \tag{2.2}$$

Para definir o tipo de rede a ser implantada, deve ser estabelecido a topologia, a função de ativação e o processo de aprendizagem a ser adotado. Essas variáveis podem ser alteradas de acordo com as especificações da aplicação da rede neural.

2.1.2.1 Topologia

A rede neural pode ser considerada como um sistema dinâmico complexo, ou seja, uma rede de sistemas interconectados. A representação de uma rede pode ser realizada utilizando modelo de grafos orientados, onde há um fluxo de sinal através dos ramos que são interligados em determinados pontos, os nós.

A partir desse modelo, temos três classes de topologias, as redes diretas, as redes recorrentes e as redes competitivas.

a) Redes Diretas (*Feed-Forward*)

As redes diretas são denominadas pela ausência de ciclos nos grafos. Essa rede é frequentemente representada em camadas, sendo chamada de *redes em camadas* [4]. O fluxo de informação é da camada de entrada para a camada de saída. As camadas que não pertencem a camada de entrada ou saída são chamadas de camadas ocultas, ou *hidden layers*. O exemplo de suas variações pode ser vista na Figura 3.

Segundo Haykin [5], essa topologia é segmentada em rede com camada única e rede com múltiplas camadas. A rede com camada única se refere a camada de saída de nós computacionais, não é considerado a camada de entrada pelo fato de não ser realizado nenhuma computação. A rede com múltiplas camadas possui camadas ocultas, capazes de intervir no processamento entre a camada de entrada e a de saída, além da rede ser capaz de extrair estatística de ordem elevada [5].

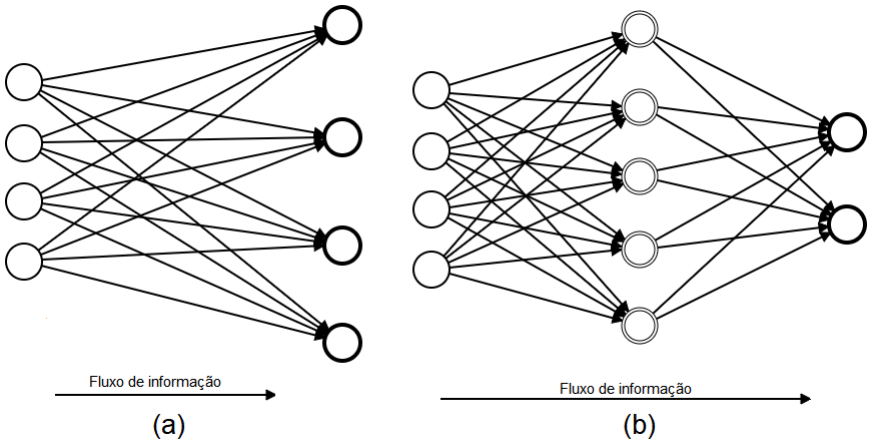

Figura 3 – Exemplo de Rede Neural, sendo em (a) única camada, e em (b), múltiplas camadas.

b) Redes Recorrentes (*Feed-Backward*)

A rede neural recorrente apresenta pelo menos um laço de realimentação, em que o sinal da saída de um nó é aplicada como entrada no próprio neurônio e/ou em camadas anteriores. Pode ser analisado o fluxograma na Figura 4.

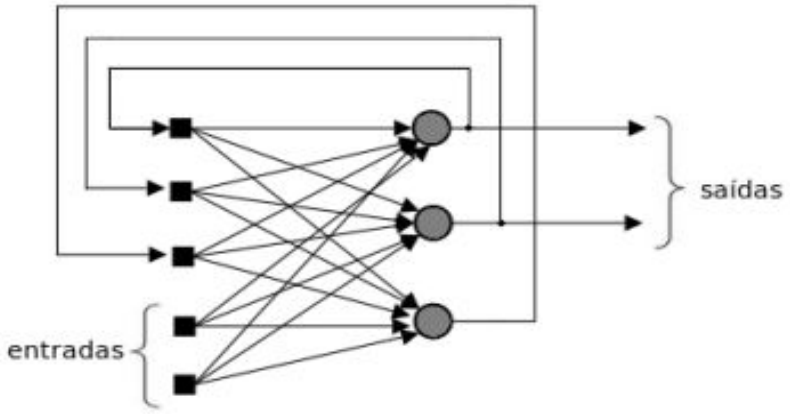

Figura 4 – Exemplo de Rede Neural com Realimentação [6].

A presença dos ciclos tem um grande impacto na capacidade de aprendizagem da rede e no seu desempenho. Os laços de realimentação envolvem o uso de elementos de atraso unitário, que resulta em um comportamento dinâmico não linear.

c) Redes Competitivas

As redes competitivas, ou também chamada de redes simétricas, é um caso particular das redes com ciclos. É utilizado simetria na realimentação, afim de assegurar estabilidade do transitório da rede. Os neurônios são forçados a competir entre si, devido a similaridade entre os neurônios, e somente o neurônio "vencedor"será ativado a cada iteração [6].

2.1.2.2 Funções de Ativação

As funções de ativação, de forma geral, decidem se um neurônio deve ser ativado ou permanecer inativo, se a informação deve ser considerada para a rede ou ignorada, tornando uma variável extremamente importante para a rede neural.

Pode-se definir a função de ativação como o efeito que a entrada e o estado atual de ativação exercem na decisão do próximo estado de ativação do neurônio.

As principais funções de ativação são: binária, linear, sigmóide e tangente hiperbólica.

a) Função Binária

A função binária é caracterizada pela existência de um limiar. Caso o valor da saída esteja acima do limite especificado, deve ativar o neurônio, caso contrário ele permanece desativado.

Essa função é utilizada para escolhas entre sim e não, uma escolha de única classe. Desse modo, a função não pode ser utilizada para classificação de várias saídas. O funcionamento é baseado nas equações:

$$f(x) = 1 \quad \forall\ x > 0 \tag{2.3}$$

$$f(x) = 0 \quad \forall\ x <= 0 \tag{2.4}$$

onde a saída possui valor igual a 1 para todo x maior que zero, e valor igual a 0 para todo x menor ou igual a zero.

b) Função Linear

A função linear é uma transformação linear da entrada. Com um fator de multiplicação igual a 1 (um), a saída linear repete o sinal de entrada do neurônio. A função é determinada pela equação:

$$f(x) = a \cdot x \tag{2.5}$$

em que a representa o coeficiente, e quando igual a 1 torna a função como identidade. Ao variar o valor do seu coeficiente pode obter a variação da inclinação da reta, onde é analisado na Figura 5.

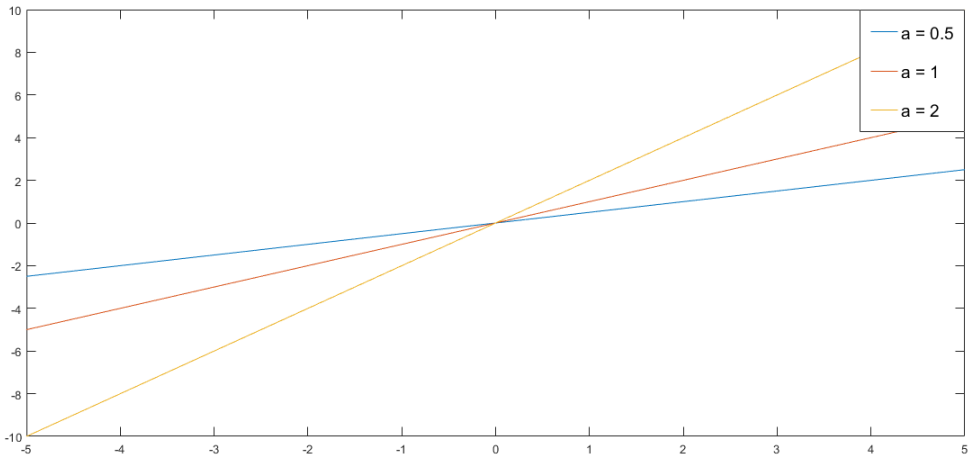

Figura 5 – Função Linar.

c) Função Sigmóide

A função de ativação sigmóide é determinada pela variação entre 0 e 1, em um formato S. A função resulta em valores extremos, o que torna desejável em classificação de valores. Além disso, temos pelo fato da função não ser simétrica na origem, os valores da saída são sempre positivos.

A origem desta função é na limitação do intervalo de variação da derivada da função, pois existe um efeito de saturação. Podemos analisar a função pela equação:

$$f(x) = \frac{1}{1 + e^{-a \cdot x}} \tag{2.6}$$

onde a representa o coeficiente responsável pela variação da inclinação da curva. Para demonstrar a variação do coeficiente, pode ser visto na 6 para valores de a igual a $0.5, 1 e 2$.

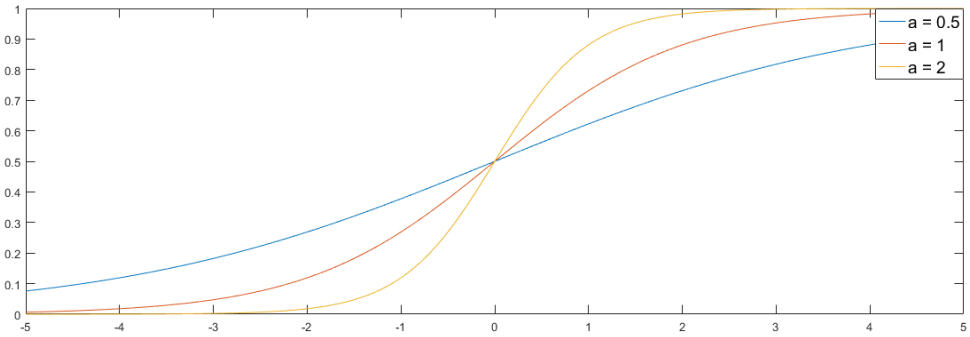

Figura 6 – Função Sigmóide.

d) Função Tangente Hiperbólica

Ao contrário da função sigmóide que limita os valores de ativação entre o intervalo (0,1), a função hiperbólica preserva a forma sigmoidal da função logística, assumindo valores positivos e negativos. A função *tanh* segue a expressão dada na equação:

$$f(x) = \frac{e^{a \cdot x} - e^{-a \cdot x}}{e^{a \cdot x} + e^{-a \cdot x}} = tanh(a \cdot x) \tag{2.7}$$

onde a representa o coeficiente da equação, permitindo a variação da inclinação da curva. Assim como a função sigmóide, foi definido valores para a igual a $0.5, 1 e 2$, que pode ser visto na Figura 7.

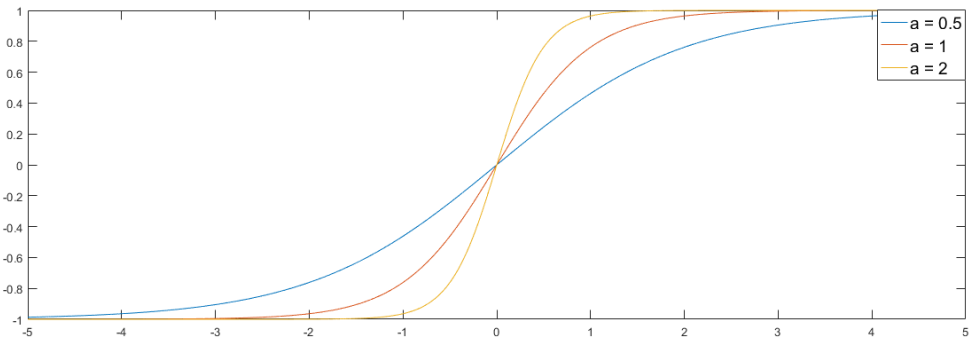

Figura 7 – Função Tangente Hiperbólica.

2.1.3 Redes Perceptron

A rede Perceptron é altamente utilizada para classificação de padrões. Ela consiste em um neurônio simples com pesos sinápticos manipuláveis e *bias*. De acordo com Rosenblatt(1958), os padrões de treinamento perceptron são extraídos a partir de duas classes linearmente separáveis, de forma que sua resposta converge na forma de um hiperplano entre as duas classes.

O Perceptron de único neurônio possui limitação na classificação de apenas duas classes de padrão. Desse modo, é necessário expandir a camada de computação, acrescentando neurônios na camada oculta, para realizar classificações com mais de duas classes [5].

A rede Perceptron de múltiplas camadas é conhecida como MLP, *multilayer perceptron*, que consiste em redes diretas que possuem no mínimo um neurônio entre a camada de entrada e saída, a existência da camada oculta. Essa camada oculta é responsável pela captura da não-linearidade dos dados [6].

As redes MLP não possuem ligações de neurônios da mesma camada, todos os neurônios são conectados apenas aos neurônios da camada seguinte. Além disso, temos que as redes perceptron não apresenta realimentação. O exemplo da rede pode ser vista na Figura 8.

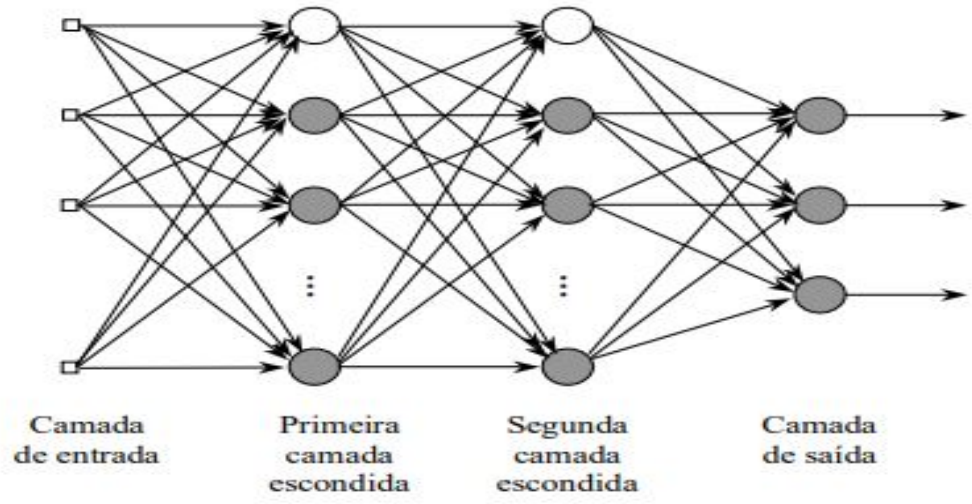

Figura 8 – Arquitetura de uma Rede Neural MLP com 2 Camadas Ocultas [11].

2.1.3.1 Aprendizagem

Em uma rede MLP, o algoritmo de treinamento mais popular é a retropropagação, *Backpropagation*, baseado na técnica de aprendizagem por correção de erros.

A aprendizagem por correção de erros é realizada a partir de um treinamento supervisionado. Essa técnica pode ser encontrada como regra delta ou de Widrow-Hoff [12], em que a variação do peso sináptico de um neurônio é proporcional ao produto entre o sinal de erro e o sinal de entrada da sinapse. Desse modo, o erro da rede decresce gradualmente.

O algoritmo de retropropagação consiste em três passos: propagação dos padrões de entrada para o treinamento, retropropagação do erro associado e os ajustes dos pesos sinápticos. De forma a representar os passos do algoritmo, pode ser vizualizado na Figura 9 o exemplo de um fluxograma.

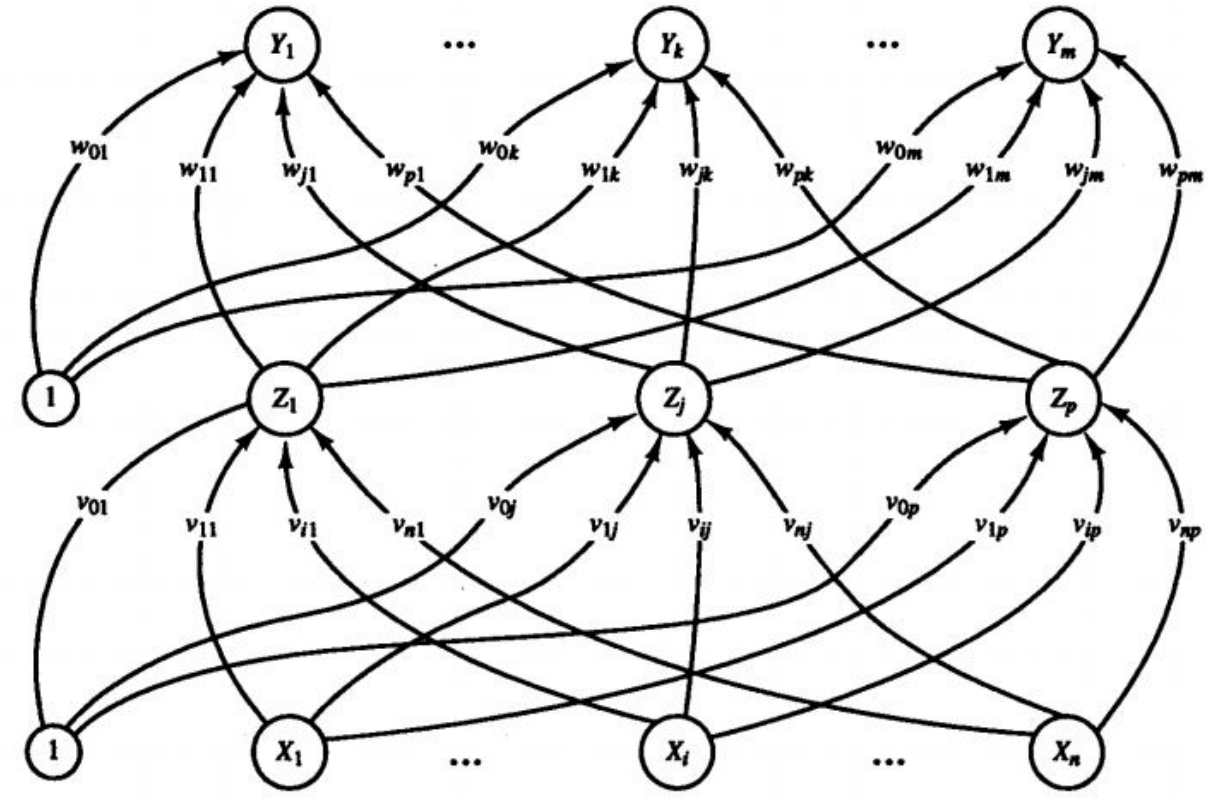

Figura 9 – Retropropagação de uma Rede Neural com 1 Camada Oculta [5].

Seguindo com o fluxograma da Figura 9, o primeiro passo é referente ao de propagação, onde os valores dos neurônios de entrada (X_i) são introduzidos aos neurônios da camada oculta (Z_p), que em seguida serão aplicados na camada de saída (Y_k). Cada unidade de saída computa sua ativação (y_k) formando a resposta da rede para os padrões de entrada. Nesse passo os pesos sinápticos são fixos [5, 6].

No treinamento, o erro associado de cada unidade da camada de saída é obtido pela diferença entre o sinal de ativação y_k com valor do desejado t_k. A partir desse erro é calculado o fator δ_k φ (k = 1,...,m). O valor de δ_k é utilizado para transmitir o erro do neurônio de saída (Y_k) de volta para os neurônios das camadas anteriores. Além disso, o fator é utilizado para atualizar os pesos entre a saída e a camada oculta. Dessa mesma forma, calculamos o fator δ_j φ ($j = 1, .., p$) para cada neurônio da camada oculta Z_j.

Depois do fator δ ser determinado, os pesos sinápticos de todas as camadas devem ser ajustados simultaneamente. O ajuste do peso w_{jk} (do neurônio oculto Z_j para o neurônio de saída Y_k) é baseado no fator δ_k e a ativação z_j do neurônio oculto Z_j. O ajuste do peso ν_{ij} (do neurônio de entrada X_i para o neurônio oculto Z_j) é baseado no fator δ_j e a ativação x_i do neurônio de entrada [13].

2.1.4 Aplicações

O número de diferentes redes neurais presentes na literatura e considerando suas variações em relação à topologia e seu algoritmo de aprendizagem, as RNA são aplicadas em uma diversidade de áreas de conhecimento. De acordo com Laurene Fausett (1993) [13], temos alguns exemplos de áreas que utilizam ou pode ser utilizadas RNA:

a) Processamento de Sinais

A principal aplicação de redes neurais no processamento de sinais é a diminuição do ruído em linhas telefônicas. A rede neural utilizada para essa função é chamada de *ADALINE*. Em uma era de comunição de longa distância, é essencial um processamento capaz de cancelar sinais ruidosos de forma adaptativa.

Nas linhas telefônicas, a rede neural foi utilizada de forma a cancelar o efeito de eco. Foi adicionado, em ambas as direções, um filtro adaptativo, em que a diferença da saída do comunicador com a saída do filtro é o erro, que é ajustado na rede neural.

b) Controle de Processos

A área de controle industrial vem utilizando redes neurais no auxílio de controles adaptativos. Desse modo, o sistema é capaz de adaptar um sinal de controle, de acordo com a mudança nas condições determinadas pelo controlador garantindo que o sistema seja capaz de atingir a trajetória de referência.

c) Reconhecimento de Padrões

A classificação e reconhecimento de padrões é a categoria com maior aplicação de redes neurais artificiais. Uma das aplicações é o desenvolvimento deste trabalho, onde é possível o reconhecimento de padrões biométricos. Além disso, pode ser aplicado para reconhecimento de caracteres, conseguindo transcrever textos contidos em imagens.

A rede mais utilizada para reconhecimento de padrões é a Multilayer Perceptron, utilizando o algoritmo de treinamento Backpropagation. Apesar de ser a mais popular, é normal ter costumizações nas redes apresentadas para melhoria de performance.

d) Medicina

Um exemplo de rede neural no campo medicinal é referente a o treinamento de uma rede neural com memória auto associativa. Desse modo, é capaz de armazenar uma grande quantidade de diagnósticos médicos, em que cada um inclui as informações referentes a sintomas, método de diagnósticos e tratamentos de casos específicos. Após o treinamento, a rede era capaz de encontrar o melhor diagnóstico e tratamento para a série de sintomas especificados.

e) Produção da Fala

Em 1986, Sejnowski e Rosenberg, produziram a *NETtalk*, em que consiste na correção de fonética das pronuncias de palavras e letras, dependendo do contexto inserido. A rede neural informa a pronúncia de frases em diferente regras gramaticais, definindo questionamento, finalização de frases, afirmação, negação, entre outros.

f) Clusterização

A clusterização é um avanço da classificação de padrões, em que as classes a serem classificadas não possuem uma quantidade e características definidas. Desse modo, é necessário uma rede que utilize algoritmo de aprendizagem não-supervisionada. Uma das redes conhecidas para clusterização é a Rede de Kohonen [6].

g) Negócios

Uma prática conhecida no mundo dos negócios, é referente à ação de bolsas de valores. A partir de redes neurais, é utilizadas estatísticas para encontrar os riscos de vendas e compras de ações. Operadores humanos tomam decisões a partir de dados marginais das ações, enquanto as redes neurais conseguem arriscar em decisões a partir de dados passados, obtendo o histórico da empresa.

2.2 Impressão Digital

A Biometria pode ser definida como a análise matemática dos dados biológicos, ou a aplicação dos métodos de estatística quantitativa em características biológicas, que pode ser armazenada com um alto grau de confiança e que permite realizar a identificação do individuo. Com o crescimento da centralização de informações há a necessidade da criação de uma identidade que é individual, definida e circunscrita [3, 14].

A eficiência de um sistema biométrico é atribuída aos seguintes fatores: universalidade - a existência do mesmo conjunto de características em todas as pessoas; unicidade - não pode haver duplicidade de características em pessoas distintas; permanência - a característica obtida não pode sofrer mutação naturalmente; e critério quantitativo - a característica obtida deve ser medida de forma quantitativa. Um sistema biométrico pode se basear em diversas características do corpo humano, como irís, voz e impressão digital [15].

No caso da impressão digital, por exemplo, sua formação ocorre até o sétimo mês de gestação. Após esse período não sofrem qualquer tipo de mutação, com exceção de cortes obtidos através de acidentes. Além disso, a impressão digital é uma particularidade única de cada indivíduo. Pesquisas referentes a dactiloscopia abordam maneiras de quantificar os parâmetros biométricos a partir de singularidades e/ou mínucias de cada individuo.

Desse modo, o reconhecimento por impressão digital é considerada a forma mais atrativa para identificação biométrica [14, 15].

2.2.1 História

A primeira pesquisa relacionada a identificação de indivíduos através de padrões contidos nas impressões digitais foi realizado por Fracis Galton, em 1888. O pesquisador identificou pontos característicos formados a partir das linhas dactilares da pele, em que as cristas, regiões mais acentuadas, convergem sobre outra crista ou é finalizada [3].

Apenas em 1905 foi realizado a classificação de indivíduos a partir da impressão digital. Foi definido, por Edward Henry, os pontos singulares como a localização em que as linhas formadas pela crista tem o formato delta, circular ou esperial - onde se origina o núcleo. Desse modo, o padrão biométrico pode ser definido pela informação obtida dos pontos característicos, encontrados por Galton, e pelos pontos singulares desenvolvidos por Henry, determinando a unicidade do indivíduo [3, 16].

2.2.2 Pontos Singulares

Os pontos singulares são denominados de detalhes globais, sendo geralmente as primeiras características analisadas em uma impressão digital. As singularidades são padrões gerados a partir das zonas em que as riscas e vales deixam de estar em paralelo, ou nas localizações de variação da orientação de forma abrupta [14].

O principal ponto singular é o núcleo, correspondente ao ponto central da impressão digital. O núcleo usualmente é classificado pela presença de *loop*, em que as linhas dactilares circundam o núcleo, ou do *espiral*, quando apresenta a volta entorno do núcleo. Essas definições podem ser vistas no exemplo da Figura 10. Em casos que o núcleo não apresenta um ponto central definido é encontrado a partir da região interna da linha de maior curvatura [3].

A classificação das impressões digitais são realizadas a partir da quantidade de singularidades delta, que são pontos formados por uma bifurcação ou uma brusca convergência de duas linhas paralelas [3].

A partir dos pontos singulares das impressões digitais, Edward Henry desenvolveu 5 classes: arco plano, arco angular, presilha interna, presilha externa e verticilo [3, 16]. Os seus exemplos podem ser vistos na Figura 11.

a) Arco Plano

O arco plano é caracterizado pela ausência de deltas. Além disso, é formado por linhas dactilares arqueadas, com formação e término em lados opostos.

b) Arco Angular

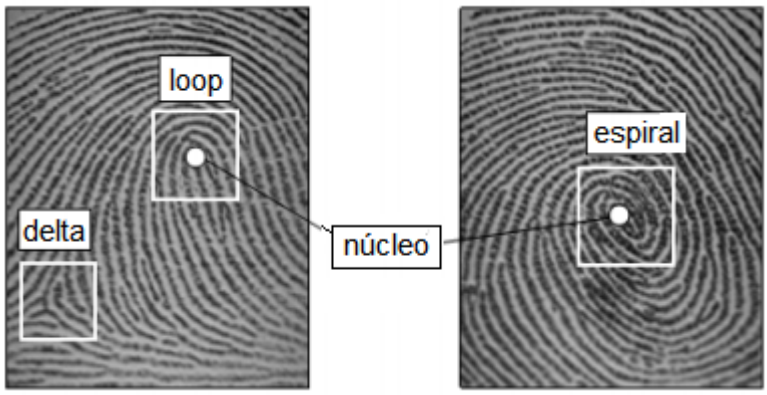

Figura 10 – Exemplos de Singularidade [14]

O centro da impressão digital é formado por um formato de tenda, com elevação acentuada. A classe do arco angular possui características semelhantes ao arco plano.

c) Presilha Interna

A presilha interna é caracterizada pela presença da singularidade delta à direita do núcleo, e com as linhas curvadas concêntrica à esquerda do núcleo.

d) Presilha Externa

A classe em questão é formada por linhas dactilares à direita do núcleo, realizando o formato espiral na impressão digital, com tendência a retornar para local de formação. Além disso, é formado pela presença de um delta à esquerda do núcleo.

e) Verticilo

A classe verticilo é composta por duas singularidades deltas, em lados distintos do núcleo. As cristas internas ao delta possui um formado curvado ou oval, concêntrico ao núcleo da impressão digital.

Figura 11 – Exemplos das classes de Henry [17]

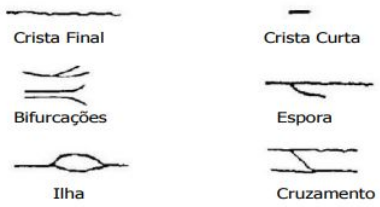

Figura 12 – Caracterização da classificação de Minúcias [17]

2.2.3 Minúcias

As minúcias são chamadas de pontos locais ou características de Galton, pontos gerados a partir pertubações presentes nas riscas. Essas pertubações resultam em terminações ou bifurcações das linhas papilares [3, 14].

Existem 2 tipos principais de minúcias, bifurcação e crista final, e 4 tipos compostos, sendo ilha, cruzamento, esporas e cristas curtas. Pode ser visto os exemplos nas Figuras 12 e 13 [3, 18].

- crista final: também chamada de terminação, ela é caracterizada pelo término de uma crista;
- bifurcação: A bifurcação é obtida a partir do união ou ligação entre duas cristas paralelas;
- ilha: também conhecida como lagos, são formadas pela união entre duas bifurcações, em que há uma divisão da crista e em seguida o reencontro entre elas;
- cruzamento: também denominada como ponte, a minúcia é o ponto de coincidência entre duas ou mais bifurcações;
- espora: a minúcia é formada a partir de uma combinação entre bifurcação e uma terminação;
- crista curta: esta minúcia é caracterizada por duas terminações com um pequeno comprimento.

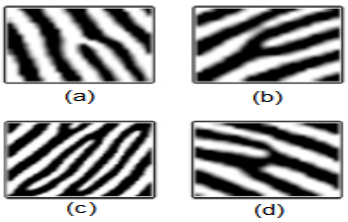

Figura 13 – Exemplos de Minúcias. Em (a), crista final; (b) bifurcação; (c) lagos; e (d) cruzamento [3]

3 Metodologia

Neste capítulo será demonstrada a metodologia utilizada no desenvolvimento do projeto. Desse modo, o capítulo é organizado em seções: na seção 3.1 será apresentado o desenvolvimento do banco de dados, na seção 3.2 o pré-processamento das imagens de impressões digitais, na seção 3.3 a extração de minúcias, na seção 3.4 a arquitetura e treinamento da rede neural e, por fim, na seção 3.5 a implementação do sistema.

3.1 Banco de Dados

O banco de dados é formado através da aquisição de 8 amostras da mesma impressão digital, alterando o seu posicionamento e a angulação do dedo correspondente.

Foi utilizado um sensor ótico FTIR (*Frustated Total Internal Reflection*), em que um prisma de vidro é utilizado, com uma fonte de luz e um sensor para criar a impressão digital. Seu funcionamento pode ser visualizado na Figura 14.

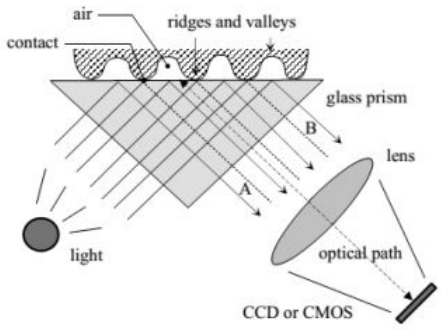

Figura 14 – Funcionamento de um Sensor Ótico FTIR [14]

O modelo utilizado foi o FPM10A que dispõe de uma interface e bibliotecas para conexão com microprocessador Arduíno.

Foram obtidas 8 amostras do mesmo dedo de 15 pessoas distintas. Todas as imagens foram coletadas especificamente para esse trabalho.

3.2 Pré-processamento da Imagem

O processo de tratamento da imagem e da detecção de minúcias foi realizado a partir de algoritmo implementado no software MATLAB. Além disso, foi utilizado conteúdo de duas bibliotecas de visão computacional, disponibilizada pelo Peter Corke [19], e por Lin Hong, Yifei Wan e A. Jain [15], e trechos de Athi Narayanan para extração das minúcias [20].

Os estágios utilizados neste trabalho para o pré-processamento da impressão digital podem ser vistos na Figura 15, e segue sua descrição detalhada está nos itens subsequentes.

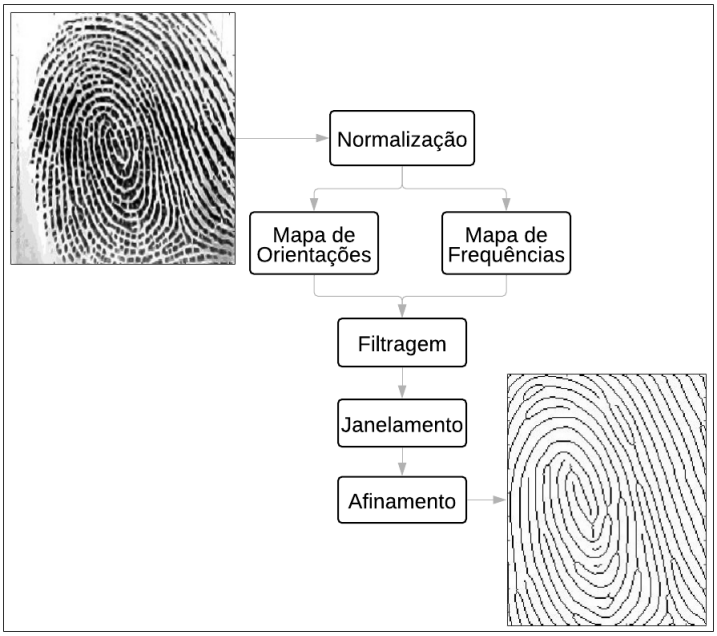

Figura 15 – Estágios do pré-processamento de impressões digitais

3.2.1 Processo de normalização

Uma imagem adquirida pelo sensor pode apresentar variações de tonalidades das linhas dactilares, apresentando regiões mais claras e outras mais escuras. Dessa forma, temos o processo de normalização para definir um nível de tonalidade padrão, e corrigir pixels que sofreram algum tipo de ruído.

No processo de normalização, a intensidade de cada *pixel* é substituída pelo valor médio e a variância dos *pixels* de toda a imagem. Além disso, nesse trabalho foi utilizado o processo de segmentação, que elimina os *pixels* com alta variância dos demais, a fim de retirar manchas ou pontos ruídosos [3].

Na figura 20 podemos identificar a variação da imagem de entrada para o resultado do processo de normalização.

Figura 16 – Exemplos do processo de normalização, em (a) a imagem de entrada, e em (b) a imagem normalizada.

3.2.2 Estimação da orientação e frequência

Uma impressão digital pode ser vista como uma superfície do tipo senoidal, com as cristas sendo os valores de pico dessa senóide. A partir da imagem normalizada, podemos encontrar o padrão da crista com a aproximação de uma onda cossenoidal:

$$w(x, y) = Acos[2\pi f_o(xcos\theta + ysin\theta)] \tag{3.1}$$

onde x e y representam as posições, A a amplitude, f_0 a frequência e θ a orientação da onda. A partir da transformada de Fourier e adotando $\hat{u}$ e $\hat{v}$ como máximo valor de magnitude do espectro, podemos separar o cosseno nos seguintes parâmetros:

$$\hat{A} = |W(\hat{u}, \hat{v})| \tag{3.2}$$

$$\hat{\theta} = arctan(\frac{\hat{v}}{\hat{u}}) \tag{3.3}$$

$$\hat{f}_o = \sqrt{\hat{u}^2 + \hat{v}^2} \tag{3.4}$$

Desse modo, é possível encontrar a estimação da orientação e frequência das cristas de uma impressão digital.

a) Mapa de orientação

O processo de estimação da orientação das riscas consiste no cálculo do ângulo θ do *pixel* seguindo o eixo x, onde pode obter valores de 0º a 179º. Esse valor pode ser encontrado a partir do cálculo do gradiente, a derivada parcial da imagem da impressão digital. Desse modo, podemos reescrever a equação 3.3 como

$$\theta(u, v) = tan^{-1}[\frac{\partial v(u, v)}{\partial u(u, v)}] \tag{3.5}$$

A fase dos componentes representa a direção da variação máxima de intensidade, e apresenta a orientação das riscas, pode ser visto como exemplo o resultado na Figura 17.

De modo a evitar ambiguidade da direção da risca obtida na equação 3.5 foi utilizada para o cálculo a conhecida função arco tangente (arctan2), que assume a analisa o sinal de cada etapa, e torna possível a identificação correta da direção da fase obtida.

b) Mapa da frequência de cristas

O mapeamento da frequência consiste em encontrar a frequência de variação dos espaçamentos entre cristas e vales de uma impressão digital, de modo a definir o limiar para encontrar os picos.

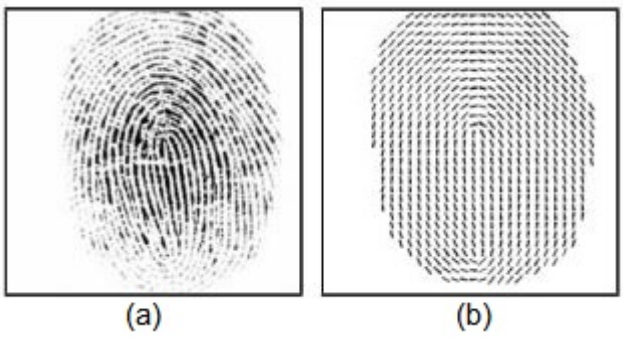

Figura 17 – Exemplo da estimação de orientação, em (a) a imagem digital de entrada, e em (b) a orientação obtida [15].

O cálculo geralmente é realizado por janelas da imagem orientada, como mostra a Figura 18. Para o cálculo da frequência é construído um vetor, tamanho W, em que cada elemento desse vetor é formado pela soma dos N pixels. A partir desse vetor podemos calcular a frequência de um sinal [3]. Como exemplo temos a Figura 18, onde podemos temos seu cálculo a partir da expressão:

$$f_{ij} = \frac{4}{L_1 + L_2 + L_3 + L_4} \tag{3.6}$$

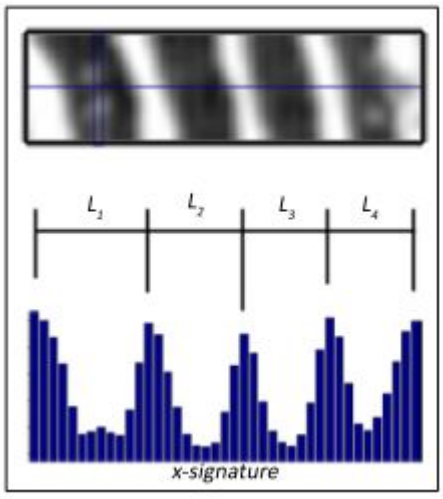

Figura 18 – Cálculo da frequência de cristas [3]

onde f representa a frequência do *pixel* na posição de índices i e j, e L_1, L_2, L_3 e L_4 são a distância entre dois picos consecutivos.

Desse modo, conseguimos calcular a frequência média estimada de uma determinada janela, com interações das janelas vizinhas, o que permite a sua estimação em casos de perturbações ou descontinuidades da imagem.

3.2.3 Filtragem

O processo de filtragem busca destacar as características em uma determinada orientação. Sendo assim, é aplicado um filtro de Gabor na imagem da impressão digital onde será destacado a diferença entre uma crista e um vale. Esse processo facilita a binarização da imagem, que será essencial para o processo de afinamento.

Para filtragem é necessário possuir os parâmetros obtidos na estimação da orientação e frequência. Desse modo, temos que o filtro é dinâmico para a imagem de entrada.

3.2.4 Redimensionamento

Neste trabalho, foi necessário realizar uma diminuição do tamanho da imagem de entrada. O sensor adotado apresentou facilidade em reconhecimento de sombras e manchas, que se tornam ruídos para a impressão digital.

A partir da imagem filtrada, e com os parâmetros descritos anteriormente encontrados, é obtido o centro da impressão digital, que possui a maior quantidade de informações, e pontos característicos.

3.2.5 Afinamento

Esse processo pode ser denominado extração das riscas, e busca definir o desenho da impressão digital. Essa técnica consiste na remoção dos *pixels* indesejados e obtenção do esqueleto da impressão digital, mantendo a sua estrutura básica.

O processo de afinamento pode gerar o efeito de serrilhamento, e formar degraus nas linhas da imagem resultante, o que pode interferir na extração das minúcias, apresentando falsas características.

3.3 Extração das Minúcias

A extração das minúcias é o passo subsequente do processo de afinamento, então é necessária uma imagem binária para esta etapa. Foi considerado a vizinhança de 8 *pixels*, então uma janela de 3x3, onde o cálculo é realizado para o *pixel* central. Desse modo, podemos obter a minúcia (Mn) do *pixel* pela equação 3.7.

$$Mn(p) = \frac{1}{2}\sum_{i=1}^{8} |Im(p_{imod8}) - Im(p_{i-1})| \tag{3.7}$$

onde p representa o *pixel* e Im(p) o valor da imagem no determinado *pixel*, e *mod* representa o módulo. A partir da equação, temos as seguintes relações:

- Mn(p) = 1, o *pixel* corresponde a uma terminação;

- Mn(p) = 2, o *pixel* é referente a um ponto da risca;
- Mn(p) = 3, o *pixel* trata-se de uma bifurcação;
- Mn(p) > 3, o *pixel* é indefinido, podendo ser uma minúcia composta.

O exemplo do método aplicado pode ser visto na Figura 19.

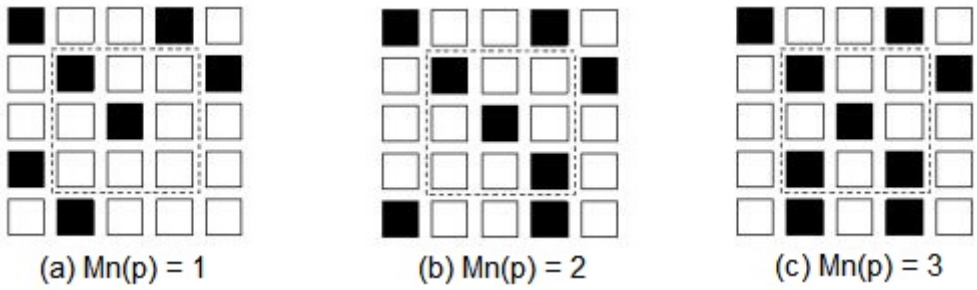

Figura 19 – Método de identificação de Minúcias, (a) terminação, (b) ponto intermédio, (c) bifurcação [14]

3.3.1 Falsa Minúcia

Durante o afinamento da impressão digital podem ocorrer falhas, como uma terminação errada, ou a presença de uma bifurcação pelo serrilhamento de uma risca comum. Desse modo, foi necessário incluir no algoritmo do trabalho um processo que elimina falsas minúcias.

O processo de eliminação de falsas minúcias consiste na eliminação dos pontos encontrados que possuem minúcias na distância de 3 *pixels*. Desse modo, são desconsideradas terminações e bifurcações muito próximas, aumentando a confiabilidade dos dados para identificação da impressão digital.

3.4 Arquitetura e treinamento da rede neural

A primeira etapa para a construção da rede neural é a padronização dos dados de entrada. A informação de entrada deste trabalho é o mapeamento das minúcias encontradas da impressão digital. Dessa maneira, foram criadas matrizes quadradas de tamanho fixo para a entrada da rede baseadas nos dados extraídos das minúcias.

Note que é inviável a utilização da imagem original pela complexidade de informação e pelo alto esforço computacional envolvido. Assim, foi necessário reduzir a informação para uma matriz com menor quantidade de informações, limitando a valores binários.

O objetivo do processamento da rede neural é, a partir da imagem de entrada, tomar a decisão da classificação da impressão digital. A sua configuração pode ser obtida de forma empírica, porém respeitando as seguintes etapas:

- tipo de rede neural;
- topologia de rede (número de camadas ocultas e sua quantidade de neurônios);
- definição do algoritmo de treinamento e função de ativação.

Em um sistema de classificação de padrões é utilizado uma rede supervisionada, onde a rede considera as saídas desejadas no seu treinamento. Neste trabalho foi determinada a rede neural MLP, que permite a solução para um sistema com inúmeras entradas e saídas.

O número de camadas ocultas e a quantidade de neurônios por camadas atua diretamente no desempenho e resultado da rede neural, e seus valores estão ligados diretamente ao sistema que está sendo aplicado. O valor abaixo do ideal torna a rede com baixa porcentagem de acerto, devido a baixa situações mapeadas. O valor acima do ideal, além de produzir um esforço computacional desnecessário, a rede de decisão fica atrelada a características específicas de treinamento, o que abaixa o índice de generalização, causando muitos erros. Dessa forma, é necessário testar e encontrar os parâmetros que possuam o melhor desempenho para o sistema que está sendo aplicado.

3.5 Implementação do sistema

A implementação do sistema consiste na integração de todas as etapas descrita anteriormente e a construção do seu protótipo. Dessa forma, foi realizado a separação dos códigos em 3 etapas.

A primeira etapa é referente a aquisição de dados. A imagem digital é produzida a partir do sensor e transferida ao sistema com comunicação serial. Dessa forma foi criado o código em linguagem C para transferir os dados serial em formato *BitMap* (BMP), produzindo a imagem. Essa etapa foi realizada a partir do *software* da IDE do Arduino que possui integração com o sensor utilizado no trabalho.

Em seguida é necessário a comunicação entre o sistema de aquisição com o sistema de processamento da imagem e rede neural. Uma linguagem que permite programação em linguagem C e consegue a integração com outros *softwares* é o *Python*. Assim, foi criado um sistema de integração entre a aquisição dos dados do Arduino com as funções produzidas no MATLAB.

A última parte do sistema consiste na lógica de processamento da imagem e funcionamento da rede neural. A plataforma escolhida para desenvolver essa última parte foi o MATLAB, que é um *software* de alto desempenho e fácil manipulação. Nesse processo foi criado a lógica para preparar a imagem obtida do sensor e obter a resposta da rede neural.

O protótipo para o teste do sistema se trata de uma plataforma fixa, em que possui o sensor de biometria digital e o microprocessador para aquisição de dados. Além disso, o

sistema é ligado com um microcomputador para o processamento e interface do usuário. Neste trabalho foi utilizado o sensor FPM10A, que possui um sistema ótico, e o Arduino Uno.

4 Resultados e análise de dados

Neste capítulo serão mostrados os resultados das etapas de desenvolvimento do projeto. O capítulo seguirá com as seguintes divisões: na seção 4.1 será apresentado o resultado das etapas de pré-processamento das imagens digitais; na seção 4.2 mostra a saída da extração das minúcias; na seção 4.3 é descrita a arquitetura final da rede neural e finaliza na seção 4.4 com os resultados da implementação do sistema.

4.1 Resultados do pré-processamento

Primeiramente foi desenvolvido um programa de linguagem C para transferir os dados serial em formato *BitMap* (BMP), produzindo as imagens. O código está no Apêndice A. A partir da aquisição de imagens, que foram capturadas usando dimensões de 256x288 *pixels*, a primeira etapa de tratamento é a normalização. Esse processo altera o contraste da imagem, o que intensifica as riscas da impressão digital. O resultado pode ser conferido na Figura 20.

Figura 20 – Resultado do processo de normalização

Durante o processo de normalização foi observado que algumas imagens intensificaram também características indesejadas, como marcações de aquisições anteriores ou sujeiras, o que dificulta na identificação ou torna a amostra inválida. Essa característica é devido a fácil marcação na superfície de vidro do sensor. Essa análise pode ser vista na Figura 21.

Com a imagem normalizada é possível realizar a estimação de orientação e frequência da impressão digital. A orientação busca indicar a trajetória e direção das riscas da impressão digital, o seu resultado pode ser visto na Figura 22. Combinando a orientação obtida, a frequência das riscas e aplicando filtros é encontrada a máscara da impressão digital, mostrada na Figura 23, que será a base para o processo de afinamento.

Um fato importante é a eliminação dos ruídos da imagem previamente a esse processo. Os detalhes indesejados são identificados como características da impressão digital, e

Figura 21 – Resultado de amostras com ruídos, em (a) a amostra da digital com sujeira, em (b) o processo normalizado, em (c) a amostra da impressão com marcações, em (c) o resultado normalizado

Figura 22 – Resultado da estimação de orientação das riscas da impressão digital

Figura 23 – Resultado da máscara da impressão digital

nesse processo será considerado como uma risca, o que retorna uma informação errada no sistema.

Pelo motivo das bordas das imagens apresentarem marcações e/ou imperfeições, neste trabalho foi adotado o método de redimensionamento. Nesse processo a informação é concentrada no núcleo da impressão digital, reduzindo a imagem para o tamanho de 211x166 *pixels*. Essa redução é vista na Figura 24.

A partir da definição das riscas é realizado o processo de afinamento. Esse método tem como objetivo remover os *pixels* redundantes das linhas que formam as cristas e obter

Figura 24 – Centralização das riscas da impressão digital

como resultado o esqueleto da imagem. O resultado desse processamento é evidenciado na Figura 25.

Figura 25 – Resultado do processo de afinamento

4.2 Extração das minúcias

A etapa de extração de minúcias é responsável pela identificação das características da impressão digital de um determinado indivíduo (ID). Neste trabalho foi concentrado a extração das minúcias de terminação e bifurcação das riscas. Essas características possuem maior quantidade e menor complexidade comparada as demais.

O processo resultante do afinamento das impressões digitais possui falhas, apresentando minúcias espúrias. Isso ocorre devido ao fato da existência de cicatrizes e falhas na pele, provocando interrupções das riscas, e/ou a qualidade da imagem e o pré-processamento não foram suficientes para determinar um bom afinamento.

Para minimizar o efeito da falha do processamento, foi realizado um algoritmo para eliminar as falsas minúcias. Nesse processo, pode ocorrer de desconsiderar minúcias verdadeiras pelo fato de imperfeições ao seu redor. Nesse processo consiste na desconsideração da minúcia do *pixel* central, caso exista outra dentro da janela de 3x3.

Na figura 26 é visto o resultado desse processo, sendo em (a) a sua extração sem o eliminação das falsas minúcias, e em (b) o efeito do algoritmo.

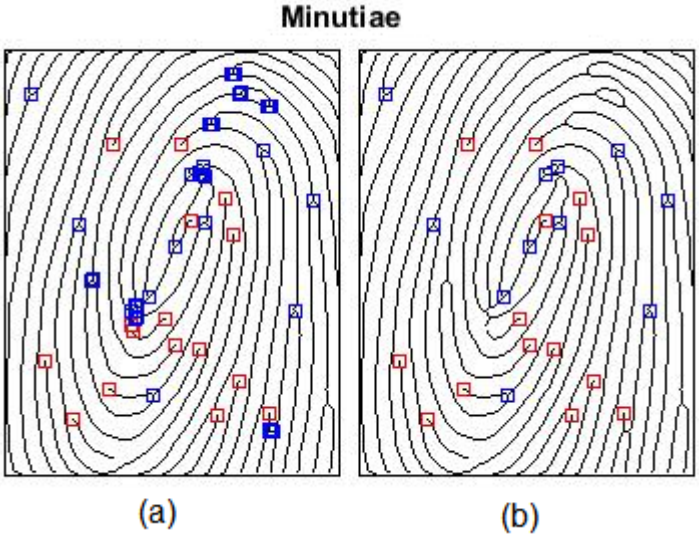

Figura 26 – Resultado da extração de minúcias, em (a) algoritmo de extração de minúcias, e em (b) algoritmo de eliminação de falsas minúcias

Note que os quadrados marcados em vermelho representam os pontos característicos de terminação, e os quadrados marcados em azul são as minúcias de bifurcação. A rotina desenvolvida em MATLAB para pré-processamento e extração das minúcias está no Apêndice B deste trabalho.

4.3 Definição da arquitetura e treinamento da Rede Neural

A partir da extração das minúcias foram padronizadas as informações em uma matriz quadrada de tamanho fixo 15x15. As coordenadas das características são divididas e inseridas proporcionalmente nessas matrizes. Desse modo, a matriz é iniciada com valores igual a 0, representando a cor preta, e os pontos brancos, de valor 1, são referentes as respectivas minúcias encontradas para o ID. A matriz de entrada da rede neural pode ser vista na Figura 27.

Considerando a aplicação do sistema do projeto, foi decidido a arquitetura da Rede Neural como uma rede MLP, que possui facilidade para identificação de padrões.

A quantidade de neurônios da camada oculta seu método de treinamento foram obtidos de forma empírica. Foi necessário realizar ajustes para encontrar o melhor resultado para o sistema proposto. Utilizando o *software* MATLAB e a *toolbox* Machine Learning, foi identificado que os métodos de Levenberg-Marquardt (LM) e o *Resilient Backpropagation*

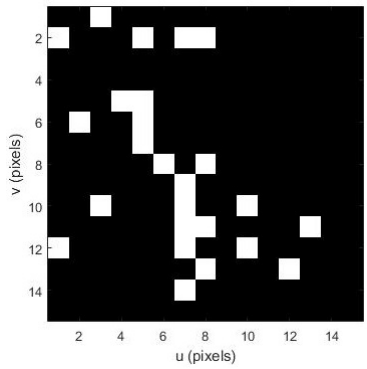

Figura 27 – Matriz de entrada da Rede Neural

(RP) apresentaram os melhores resultados. De acordo com a performance e eficiência da rede foi adotado a técnica LM para o treinamento neste trabalho.

Com o método de treinamento determinado, a configuração de somente uma camada oculta, com 10 neurônios de processamento, apresentou o melhor desempenho.

Para resolução da rede foi utilizado a métrica *Confusion Matrix*. Nessa avaliação são utilizadas quatro matrizes de confusão para a rede neural treinada, cada uma representa uma diretriz diferente. A primeira matriz *Training Confusion Matrix* representa o resultado do treinamento da rede; a matriz *Validation Confusion Matrix* diz respeito ao resultado da validação dos dados; a matriz *Test Confusion Matrix* representa o resultado dos testes para o treinamento; e a matriz *Overall Confusion Matrix* possui o resultado total final da rede, com a soma dos resultados obtidos nas matrizes anteriores.

Com o banco de dados criado, foi determinado que dos 15 ID's distintos, 12 ID's foram usados para treinamento, ou seja, o sistema foi treinado para reconhecer apernas os 12 ID's cadastrados. Além disso, foram utilizados os outros 3 ID's e as amostras restantes para testes. Desta forma, a matriz de saída da rede corresponde a um vetor de tamanho 12, em que cada unidade corresponde a um neurônio de saída da rede.

A configuração descrita foi criada no *software* MATLAB e foi obtido a sua forma esquemática, que pode ser observada pela Figura 28.

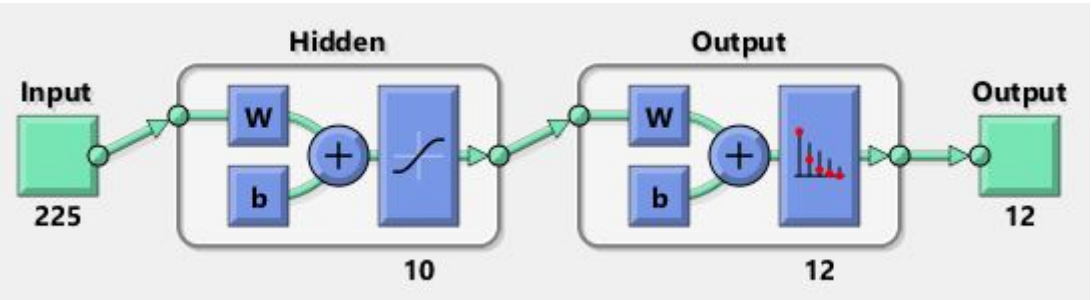

Figura 28 – Esquemático da Rede Neural adotada

A Figura 29, representando a matriz de confusão da Rede Neural, indica que o treinamento da rede resultou em um acerto de 95.1%. Apesar de um número alto durante o

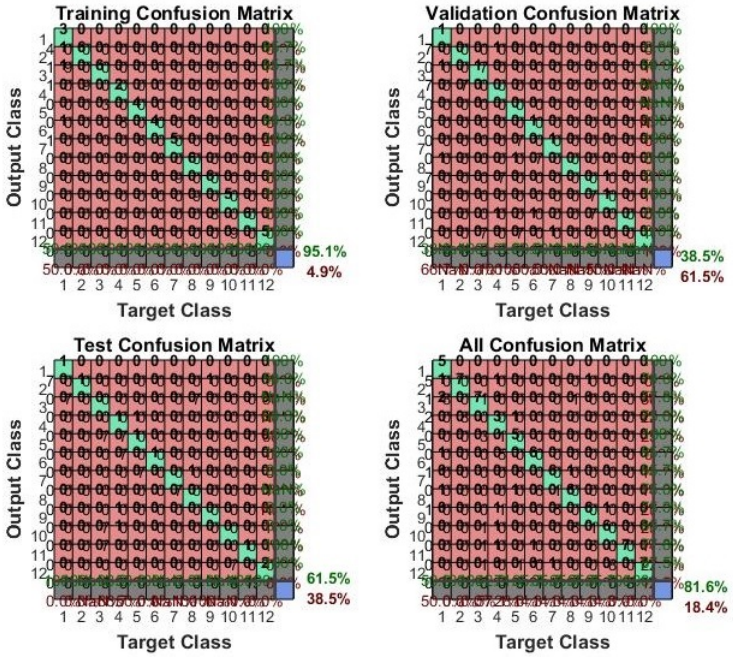

Figura 29 – Matriz de confusão da Rede Neural

treinamento, pode perceber que durante os testes de treinamento foi obtido apenas 61.5%, e na validação 38.5% de acerto.

O resultado dessas variáveis foi devido a utilização de amostras dos ID's não treinados para a validação e testes. Dessa maneira, durante o teste e validação, as amostras cujo não são encontrados os valores de saída retornam aumentam o número de confusão, abaixando o nível de acerto.

Além disso, temos que apesar do alto reconhecimento de padrões da Rede Neural, há falhas na identificação do indivíduo quando possui uma variação pronunciada de posicionamento e angulação da impressão digital.

4.4 Sistema Implementado

O sistema completo foi implementado no *software* baseado em linguagem *Python*. A partir dele foi integrado o sistema de aquisição de dados com o sistema de processamento. Desse modo, a partir da aquisição da imagem pelo sensor FPM10A, o sistema realiza o processamento e retorna o ID da digital correspondente. Caso a impressão digital não esteja cadastrada o sistema retorna valor nulo, igual a zero.

A primeira etapa dos testes realizados foi realizada com as imagens obtidas do banco de dados. A partir dessas imagens foi executado o processamento do sistema e foi alcançado o acerto próximo do testes do treinamento, o total de 60% dos IDs correspondentes.

Para esse reconhecimento da impressão digital foi adotado a certeza de 85% dos dados processados. Apesar de ser um número alto, sendo necessário a equivalência de 85% de

minúcias correspondentes, o resultado do sistema com o banco de dados obtidos apresentou um alto número de falsas aceitações, representando 9.887% de um total de 120 amostras. Apesar de ser um número baixo, em um sistema de identificação de indivíduos se torna crítico, isso representa a possibilidade de duplicidade dos dados no trabalho proposto, o que pode inviabilizar a utilização em um sistema real.

Esse resultado é derivado do sistema de redução da informação, ao reduzir as informações da imagem de 211x166 para uma de 15x15, há pontos característicos que são sobrepostos aos outros. Além disso, se torna necessário aumentar o número de amostras que não pertencem ao reconhecimento do sistema.

Além das imagens do banco de dados, foi realizado testes com o protótipo do sistema. Dessa maneira, é gerado uma imagem inédita ao sistema e é obtido através do processamento a equivalência da impressão digital. Nesta etapa foi obtido o acerto de 75% da ID correspondente e não foi observado a detecção de falsa aceitação, a partir de 35 amostras. Os códigos do sistema implementado em *Python* estão no Apêndice C.

5 Conclusões

O objetivo deste trabalho foi o de reconhecer indivíduos previamente cadastrados utilizando o método de extração das minúcias por intermédio de redes neurais artificiais e de técnicas de visão computacional, realizando pré-processamento das imagens. A aquisição das imagens foi realizada a partir da comunicação serial do Arduino com o sensor FPM10A, o processamento das imagens e aplicação da rede neural foi realizado pelo MATLAB que possui diferentes bibliotecas disponíveis. A integração dos dois *softwares* foi realizado a partir da programação em Python.

As etapas de pré-processamento das imagens seguiram procedimentos padrões normalmente utilizados para o reconhecimento de biometria digital através de cálculos estatísticos, e apresentaram um resultado positivo. A qualidade do sensor influenciou os resultados do processamento. Desta forma, a etapa de normalização se mostrou a principal do processo, com a alteração de amostras do mesmo indivíduo obteve resultados diferentes. O processo de afinamento apresentou pontos de serrilhamento, com características semelhantes às minúcias, porém, apesar desse efeito o algoritmo de eliminação de falsas minúcias foi eficiente no seu funcionamento.

A rede neural foi capaz de reconhecer o indivíduos mesmo com alta variação de posicionamento e ângulo das amostras. A rede utilizada para o processamento foi a Perceptron multicamadas com 10 neurônios na camada oculta, apresentando uma taxa de 95.1% de acerto durante o seu treinamento. A implementação total do sistema obteve um acerto de 75% do reconhecimento dos indivíduos. Entretanto, apesar da alta taxa de acerto do sistema o algoritmo apresentou cerca de 10% de falta aceitação, o que torna uma característica crítica do seu funcionamento.

Como perspectivas futuras busca-se aprimorar o algoritmo de pré-processamento das impressões digitais. A etapa de ajustes das imagens utilizando técnicas de visão computacional revelou ser o processo mais importante para o reconhecimento dos padrões. Desta forma, é necessário aperfeiçoar a etapa de normalização e afinamento do algoritmo. Outro aspecto importante seria aumentar a confiabilidade e robustez da rede, alterando a matriz de entrada da rede neural de forma a não permitir duplicidade ou sobreposição de dados, tornando a matriz do indivíduo única ao sistema.

Além disso, é interessante pensar na implementação física em algum cenário de atuação, como entradas de estabelecimentos, aeroportos, codificação de cofres ou celulares, entre outros. Para isso é necessário aumentar a quantidade de informações e indivíduos no banco de dados e lidar com as complexidades consequentes desse aumento.

Referências Bibliográficas

1 NORTON, B. Karl pearson and statistics: The social origins of scientific innovation. In: SAGE PUBLICATIONS. *Social Studies of Science*. [S.l.], 1978. v. 8, p. 3–34.

2 JAIN ANIL K., R. A. A. N. K. *Introduction to Biometrics*. [S.l.]: Springer, 2011.

3 CASTRO, T.

Identificação de Impressões Digitais Baseada na Extração de Minucias — Universidade Federal de Juiz de Fora, 2008.

4 BARRETO, J. Introdução às redes neurais artificiais. In: *Laboratorio de Conexionismo e Ciências Cognitivas, Universidade Federal de Santa Catarina (UFSC)*. [S.l.: s.n.], 2002.

5 HAYKIN, S. *Neural networks: a comprehensive foundation*. 2nd. ed. [S.l.]: Prentice Hall, 1999.

6 GONcALVES, A. R. *Fundamentos e Aplicações de Técnicas de Aprendizado de Máquina*. Bachelor's Thesis — Department of Computer Science, State University of Londrina, Londrina, Brazil, 2008.

7 MCCULLOCH, W. S.; PITTS, W. The bullentin of mathematical biophysics. v. 5, p. 115–133, 1943.

8 HUBEL, D. H. *The brain*. [S.l.]: Freeman, San Francisco, 1979. 2-14 p.

9 GM, S. *The Synaptic Organization of the Brain*. [S.l.]: Oxford University Press, New York, 1974.

10 ARBIB, M. A. *The Handbook of Brain Theory and Neural Networks*. 2. ed. [S.l.]: MIT Press, 2002.

11 NUNES, L. *Analise e Sintese de Estrategias de Aprendizado para Redes Neurais Artificiais*. Master's Thesis — Departamento de Engenharia de Computação e Automação Industrial. Universidade Estadual de Campinas - Unicamp, 1998.

12 WIDROW, B.; HOFF, J. M. E. Adaptative switching circuits. In: *IREWESCON Converntion Record*. [S.l.: s.n.], 1960. p. 96–104.

13 FAUSSET, L. *Fundamentals of Neural Networks: Architectures, Algorithms and Applications*. 1. ed. [S.l.]: Pearson, 1993.

14 LEITAO, R.

Sistema de de Impressões Digitais Baseadoo em FPGA — Instituto Superior de Engenharia de Lisboa, 2014.

15 HONG L., W. Y. J. A. K. Fingerprint image enhancement: Algorithms and performance evaluation. In: IEEE. *Transactions on Pattern Analysis and Machine Intelligence*. [S.l.], 1998. v. 20, p. 777–789.

16 HENRY, E. R. *Classification and Uses of Fingerprints*. [S.l.]: Wyman and Sons Ltda, 1905.

17 MAZI, R.; JúNIOR, A. Identificação biometrica atraves da impressão digital usando redes neurais artificiais. *XIV ENCITA*, 2008.

18 ZAMBERLAM ALEXANDRA, B. M. F. P. A. n. L. B. P.

Um Estudo do Processo de Reconhecimento de Individuos pelo Metodo da Impressão Digital — Centro Universitário Feevale Novo Hamburgo.

19 CORKE, P. *Robotics, Vision and Control: Fundamental Algotithms in MATLAB*. 1. ed. [S.l.]: Springer Verlang NY, 2011.

20 NARAYANAN, A. Program for fingerprint minutiae extraction. In: . [S.l.: s.n.], 2011.

A Código para aquisição das imagens

A.1 Aquisição via Arduino

Segue o código para transferir a imagem obtida do sensor FPM10A para o computador:

```
#include <SoftwareSerial.h>
#include <FPM.h>

/* Send fingerprint image to PC in 4-bit BMP format */

/*  pin #2 is IN from sensor (GREEN wire)
    pin #3 is OUT from arduino  (WHITE/YELLOW wire)
*/
SoftwareSerial fserial(2, 3);

FPM finger(&fserial);
FPM_System_Params params;

void setup()
{
    Serial.begin(57600);
    Serial.println("SEND IMAGE TO PC test");
    fserial.begin(57600);

    if (finger.begin()) {
        finger.readParams(&params);
        Serial.println("Found fingerprint sensor!");
        Serial.print("Capacity: "); Serial.println(params.capacity);
        Serial.print("Packet length: "); Serial.println(FPM::packet_lengths
    [params.packet_len]);
    }
    else {
        Serial.println("Did not find fingerprint sensor :(");
        while (1) yield();
    }
}

void loop() {
    stream_image();
    while (1) yield();
}

```

```
37 void stream_image(void) {
38     if (!set_packet_len_128()) {
39         Serial.println("Could not set packet length");
40         return;
41     }
42 
43     delay(100);
44 
45     int16_t p = -1;
46     Serial.println("Waiting for a finger...");
47     while (p != FPM_OK) {
48         p = finger.getImage();
49         switch (p) {
50             case FPM_OK:
51                 Serial.println("Image taken");
52                 break;
53             case FPM_NOFINGER:
54                 break;
55             case FPM_PACKETRECIEVEERR:
56                 Serial.println("Communication error");
57                 break;
58             case FPM_IMAGEFAIL:
59                 Serial.println("Imaging error");
60                 break;
61             default:
62                 Serial.println("Unknown error");
63                 break;
64         }
65         yield();
66     }
67 
68     p = finger.downImage();
69     switch (p) {
70         case FPM_OK:
71             Serial.println("Starting image stream...");
72             break;
73         case FPM_PACKETRECIEVEERR:
74             Serial.println("Communication error");
75             return;
76         case FPM_UPLOADFAIL:
77             Serial.println("Cannot transfer the image");
78             return;
79     }
80 
81     /* header to indicate start of image stream to PC */
82     Serial.write('\t');
83     bool read_finished;
```

```
84      int16_t count = 0;
85
86      while (true) {
87          bool ret = finger.readRaw(FPM_OUTPUT_TO_STREAM, &Serial, &
        read_finished);
88          if (ret) {
89              count++;
90              if (read_finished)
91                  break;
92          }
93          else {
94              Serial.print("\r\nError receiving packet ");
95              Serial.println(count);
96              return;
97          }
98          yield();
99      }
100
101     Serial.println();
102     Serial.print(count * FPM::packet_lengths[params.packet_len]); Serial.
        println(" bytes read.");
103     Serial.println("Image stream complete.");
104 }
105
106 bool set_packet_len_128(void) {
107     uint8_t param = FPM_SETPARAM_PACKET_LEN; // Example
108     uint8_t value = FPM_PLEN_128;
109     int16_t p = finger.setParam(param, value);
110     switch (p) {
111         case FPM_OK:
112             Serial.println("Packet length set to 128 bytes");
113             break;
114         case FPM_PACKETRECIEVEERR:
115             Serial.println("Comms error");
116             break;
117         case FPM_INVALIDREG:
118             Serial.println("Invalid settings!");
119             break;
120         default:
121             Serial.println("Unknown error");
122     }
123
124     return (p == FPM_OK);
125 }
```

A.2 Aquisição via Python

De forma a simplificar a obtenção dos dados, segue o código para aquisição para um específico diretório:

```
# Written by Brian Ejike (2017)
# Adapted by Fellipe E G Silva (2019)

import serial, time

WIDTH = 256
HEIGHT = 288
READ_LEN = int(WIDTH * HEIGHT / 2)

DEPTH = 8
HEADER_SZ = 54

portSettings = ['', 0]

print("----------Extract Fingerprint Image----------")
print()

# assemble bmp header for a grayscale image
def assembleHeader(width, height, depth, cTable=False):
    header = bytearray(HEADER_SZ)
    header[0:2] = b'BM'    # bmp signature
    byte_width = int((depth*width + 31) / 32) * 4
    if cTable:
        header[2:6] = ((byte_width * height) + (2**depth)*4 + HEADER_SZ).
    to_bytes(4, byteorder='little') #file size
    else:
        header[2:6] = ((byte_width * height) + HEADER_SZ).to_bytes(4,
    byteorder='little') #file size
    #header[6:10] = (0).to_bytes(4, byteorder='little')
    if cTable:
        header[10:14] = ((2**depth) * 4 + HEADER_SZ).to_bytes(4, byteorder=
    'little') #offset
    else:
        header[10:14] = (HEADER_SZ).to_bytes(4, byteorder='little') #offset

    header[14:18] = (40).to_bytes(4, byteorder='little')    #file header
    size
    header[18:22] = width.to_bytes(4, byteorder='little') #width
    header[22:26] = (-height).to_bytes(4, byteorder='little', signed=True)
    #height
    header[26:28] = (1).to_bytes(2, byteorder='little') #no of planes
    header[28:30] = depth.to_bytes(2, byteorder='little') #depth
```

```
38      #header[30:34] = (0).to_bytes(4, byteorder='little')
39      header[34:38] = (byte_width * height).to_bytes(4, byteorder='little') #
        image size
40      header[38:42] = (1).to_bytes(4, byteorder='little') #resolution
41      header[42:46] = (1).to_bytes(4, byteorder='little')
42      #header[46:50] = (0).to_bytes(4, byteorder='little')
43      #header[50:54] = (0).to_bytes(4, byteorder='little')
44      return header
45
46  def options():
47      print("Options:")
48      print("\tPress 1 to enter serial port settings")
49      print("\tPress 2 to scan a fingerprint and save the image")
50      print("\tPress 4 to exit")
51      print()
52      choice = input(">> ")
53      print()
54      return choice
55
56  def getSettings():
57      portSettings[0] = input("Enter Arduino serial port number: ")
58      portSettings[1] = int(input('Enter serial port baud rate: '))
59      print()
60
61  def getPrint():
62      '''
63      First enter the port settings with menu option 1:
64      >>> Enter Arduino serial port number: COM13
65      >>> Enter serial port baud rate: 57600
66
67      Then enter the filename of the image with menu option 2:
68      >>> Enter filename/path of output file (without extension): myprints
69      Found fingerprint sensor!
70      .
71      .
72      .
73      (Here you communicate with the Arduino and follow instructions)
74      .
75      .
76      .
77      Extracting image...saved as <filename>.bmp
78
79      '''
80      out = open(input("Enter filename/path of output file (without extension
        ): ")+'.bmp', 'wb')
81      # assemble and write the BMP header to the file
82      out.write(assembleHeader(WIDTH, HEIGHT, DEPTH, True))
```

```
83      for i in range(256):
84          # write the colour palette
85          out.write(i.to_bytes(1,byteorder='little') * 4)
86      try:
87          # open the port; timeout is 1 sec; also resets the arduino
88          ser = serial.Serial(portSettings[0], portSettings[1], timeout=1)
89      except Exception:
90          print('Invalid port settings!')
91          print()
92          out.close()
93          return
94      while ser.isOpen():
95          try:
96              # assumes everything recved at first is printable ascii
97              curr = ser.read().decode()
98              # based on the image_to_pc sketch, \t indicates start of the
    stream
99              if curr != '\t':
100                 # print the debug messages from arduino running the
    image_to_pc sketch
101                 # print(curr, end='')
102                 continue
103             for i in range(READ_LEN): # start recving image
104                 byte = ser.read()
105                 # if we get nothing after the 1 sec timeout period
106                 if not byte:
107                     print("Timeout!")
108                     out.close()  # close port and file
109                     ser.close()
110                     return False
111                 # make each nibble a high nibble
112                 out.write((byte[0] & 0xf0).to_bytes(1, byteorder='little'))
113                 out.write(((byte[0] & 0x0f) << 4).to_bytes(1, byteorder='
    little'))
114
115             out.close()  # close file
116             print('Image saved as', out.name)
117
118             # read anything that's left and print
119             left = ser.read(100)
120             print(left.decode('ascii', errors='ignore'))
121             ser.close()
122
123             print()
124             return True
125         except Exception as e:
126             print("ERROR: ", e)
```

```
127             out.close()
128             ser.close()
129             return False
130         except KeyboardInterrupt:
131             print("Closing port.")
132             out.close()
133             ser.close()
134             return False
135 
136 while True:
137     chose = options()
138     if chose == "4":
139         break
140     elif chose == '1':
141         getSettings()
142     elif chose == "2":
143         res = getPrint()
144         if not res:
145             print("Image extraction failed!")
146         continue
```

B Código de tratamento das imagens

B.1 Pré-processamento

As etapas de pré-processamento das imagens foram separadas por funções.

a) Função Principal

```
function [Out sz] = preprocessing (dig)
% Developer Fellipe E G Silva
%% start
blksze = 16; thresh = 0.1;
[normim, mask, maskind] = ridgesegment(dig, blksze, thresh);
H = 1/25*ones(5,5);
LEN = 2;
THETA = 2;
PSF = fspecial('motion',LEN,THETA); % create PSF
%figure; imshow(normim)

%% Get orientation
[orientim, reliability] = ridgeorient(normim, 1, 5, 5);
%plotridgeorient(orientim, 8, normim, 2)
%% Get Frequency
[freq,medfreq] = ridgefreq(normim, mask, orientim, 16, 5, 5,15);
freq = medfreq.*mask;
newin = ridgefilter(normim,orientim, freq, 0.5, 0.5, 0);
binim = newin>0;
%idisp(binim.*mask.*(reliability>0.5),'new')

%% Get Small Image
An_im = binim.*-1;
An_im = An_im(50:260,60:225);

%% Thin Image
thin_image=~bwmorph(An_im,'thin',150);
%idisp(thin_image,'new')

%% Extract Minutia
s=size(thin_image);
N=3;%window size
n=(N-1)/2;
r=s(1)+2*n;
```

```
c=s(2)+2*n;
double temp(r,c);

%%
temp=zeros(r,c);bifurcation=zeros(r,c);ridge=zeros(r,c);
temp((n+1):(end-n),(n+1):(end-n))=thin_image(:,:);
outImg=zeros(r,c,3);%For Display
outImg(:,:,1) = temp .* 255;
outImg(:,:,2) = temp .* 255;
outImg(:,:,3) = temp .* 255;
%%
for x=(n+1+10):(s(1)+n-10)
    for y=(n+1+10):(s(2)+n-10)
        e=1;
        for k=x-n:x+n
            f=1;
            for l=y-n:y+n
                mat(e,f)=temp(k,l);
                f=f+1;
            end
            e=e+1;
        end;
         if(mat(2,2)==0)
            ridge(x,y)=sum(sum(~mat));
            bifurcation(x,y)=sum(sum(~mat));
         end
    end;
end;
%% RIDGE END FINDING
[ridge_x ridge_y]=find(ridge==2);
len=length(ridge_x);
t = 0;
ridge_x2 = [];
ridge_y2 = [];

for i = 1:len
    for j = 1:len
        if (ridge_x(i) >= ridge_x(j) - 3 && ridge_x(i) <= ridge_x(j) + 3)
   && (i ~= j)
            if (ridge_y(i) >= ridge_y(j) - 3 && ridge_y(i) <= ridge_y(j) +
    3) && (i ~= j)
                t = 1;
            end
        end
    end
    if t == 0
        ridge_x2 = [ridge_x2; ridge_x(i)];
```

```
80          ridge_y2 = [ridge_y2; ridge_y(i)];
81      end
82      t = 0;
83  end
84  len = length(ridge_x2);
85  %For Display
86  for i=1:len
87      outImg((ridge_x2(i)-3):(ridge_x2(i)+3),(ridge_y2(i)-3),2:3)=0;
88      outImg((ridge_x2(i)-3):(ridge_x2(i)+3),(ridge_y2(i)+3),2:3)=0;
89      outImg((ridge_x2(i)-3),(ridge_y2(i)-3):(ridge_y2(i)+3),2:3)=0;
90      outImg((ridge_x2(i)+3),(ridge_y2(i)-3):(ridge_y2(i)+3),2:3)=0;
91
92      outImg((ridge_x2(i)-3):(ridge_x2(i)+3),(ridge_y2(i)-3),1)=255;
93      outImg((ridge_x2(i)-3):(ridge_x2(i)+3),(ridge_y2(i)+3),1)=255;
94      outImg((ridge_x2(i)-3),(ridge_y2(i)-3):(ridge_y2(i)+3),1)=255;
95      outImg((ridge_x2(i)+3),(ridge_y2(i)-3):(ridge_y2(i)+3),1)=255;
96  end
97  %BIFURCATION FINDING
98  [bifurcation_x bifurcation_y]=find(bifurcation==4);
99  len=length(bifurcation_x);
100 t = 0;
101 bifurcation_x2 = [];
102 bifurcation_y2 = [];
103 for i = 1:len
104     for j = 1:len
105         if (bifurcation_x(i) >= bifurcation_x(j) - 3 && bifurcation_x(i) <=
     bifurcation_x(j) + 3) && (i ~= j)
106             if (bifurcation_y(i) >= bifurcation_y(j) - 3 && bifurcation_y(i
    ) <= bifurcation_y(j) + 3) && (i ~= j)
107                 t = 1;
108             end
109         end
110     end
111     if t == 0
112         bifurcation_x2 = [bifurcation_x2; bifurcation_x(i)];
113         bifurcation_y2 = [bifurcation_y2; bifurcation_y(i)];
114     end
115     t = 0;
116 end
117
118 len = length(bifurcation_x2);
119 %For Display
120 for i=1:len
121     outImg((bifurcation_x2(i)-3):(bifurcation_x2(i)+3),(bifurcation_y2(i)
    -3),1:2)=0;
122     outImg((bifurcation_x2(i)-3):(bifurcation_x2(i)+3),(bifurcation_y2(i)
    +3),1:2)=0;
```

```
123     outImg((bifurcation_x2(i)-3),(bifurcation_y2(i)-3):(bifurcation_y2(i)
    +3),1:2)=0;
124     outImg((bifurcation_x2(i)+3),(bifurcation_y2(i)-3):(bifurcation_y2(i)
    +3),1:2)=0;
125
126     outImg((bifurcation_x2(i)-3):(bifurcation_x2(i)+3),(bifurcation_y2(i)
    -3),3)=255;
127     outImg((bifurcation_x2(i)-3):(bifurcation_x2(i)+3),(bifurcation_y2(i)
    +3),3)=255;
128     outImg((bifurcation_x2(i)-3),(bifurcation_y2(i)-3):(bifurcation_y2(i)
    +3),3)=255;
129     outImg((bifurcation_x2(i)+3),(bifurcation_y2(i)-3):(bifurcation_y2(i)
    +3),3)=255;
130 end
131 %figure;imshow(outImg);title('Minutiae');
132
133 %% Get Singularity
134 An_or = orientim;
135 An_or = An_or(50:260,60:225);
136 [si,sj]=size(An_or);
137 soma = 0;
138 loop_x = []; loop_y = [];
139 delta_x = []; delta_y = [];
140 whorl_x = []; whorl_y = [];
141 for i = 1:si
142     for j = 1:sj
143         if(i+1 <= si && i-1 > 0 && j+1 <= sj && j-1 > 0)
144             O = [An_or(i-1,j-1) An_or(i-1,j) An_or(i-1,j+1) An_or(i,j-1)
    An_or(i,j+1) An_or(i+1,j-1) An_or(i+1,j) An_or(i+1,j+1)];
145             for k = 1:8
146                 if(k~=8)
147                     tht = O(k+1)-O(k);
148                 end
149                 if(k==8)
150                     tht = O(1)-O(k);
151                 end
152                 if(tht > pi/2)
153                     dlt = tht - pi;
154                 end
155                 if(tht <= pi/2 && tht >= -pi/2)
156                     dlt = tht;
157                 end
158                 if(tht < -pi/2)
159                     dlt = tht + pi;
160                 end
161                 soma = soma + dlt;
162             end
```

```
163             PI = (1/pi)*soma;
164             if (PI == -1)
165                 delta_x = [delta_x i];
166                 delta_y = [delta_y j];
167             end
168             if (PI == 1)
169                 loop_x = [loop_x i];
170                 loop_y = [loop_y j];
171             end
172             if (PI == 2)
173                 whorl_x = [whorl_x i];
174                 whorl_y = [whorl_y j];
175             end
176         end
177         soma = 0;
178     end
179 end
180
181 %% Display singularity
182 % figure;idisp(normim(50:260,60:225))
183 % len = length(loop_x)
184 % for m = 1:len
185 %     plot_box('centre',[loop_y(m) loop_x(m)],'size',5);
186 % end
187 % len = length(delta_x)
188 % for n = 1:len
189 %     plot_circle([delta_y(n) delta_x(n)],5);
190 % end
191
192 %% Output
193 Out = [bifurcation_x2' ridge_x2' loop_x delta_x whorl_x;
194        bifurcation_y2' ridge_y2' loop_y delta_y whorl_y];
195 Out = Out';
196 sz = size(An_im);
```

b) Função Normalização

```
1 % RIDGESEGMENT - Normalises fingerprint image and segments ridge region
2 %
3 % Function identifies ridge regions of a fingerprint image and returns a
4 % mask identifying this region.  It also normalises the intesity values of
5 % the image so that the ridge regions have zero mean, unit standard
6 % deviation.
7 %
8 % This function breaks the image up into blocks of size blksze x blksze and
9 % evaluates the standard deviation in each region.  If the standard
10 % deviation is above the threshold it is deemed part of the fingerprint.
```

```
% Note that the image is normalised to have zero mean, unit standard
% deviation prior to performing this process so that the threshold you
% specify is relative to a unit standard deviation.
%
% Usage:   [normim, mask, maskind] = ridgesegment(im, blksze, thresh)
%
% Arguments:   im      - Fingerprint image to be segmented.
%              blksze  - Block size over which the the standard
%                        deviation is determined (try a value of 16).
%              thresh  - Threshold of standard deviation to decide if a
%                        block is a ridge region (Try a value 0.1 - 0.2)
%
% Returns:     normim  - Image where the ridge regions are renormalised to
%                        have zero mean, unit standard deviation.
%              mask    - Mask indicating ridge-like regions of the image,
%                        0 for non ridge regions, 1 for ridge regions.
%              maskind - Vector of indices of locations within the mask.
%
% Suggested values for a 500dpi fingerprint image:
%
%   [normim, mask, maskind] = ridgesegment(im, 16, 0.1)
%
% See also: RIDGEORIENT, RIDGEFREQ, RIDGEFILTER

% Peter Kovesi
% School of Computer Science & Software Engineering
% The University of Western Australia
% pk at csse uwa edu au
% http://www.csse.uwa.edu.au/~pk
%
% January 2005


function [normim, mask, maskind] = ridgesegment(im, blksze, thresh)

    im = normalise(im,0,1);  % normalise to have zero mean, unit std dev

    fun = inline('std(x(:))*ones(size(x))');

    stddevim = blkproc(im, [blksze blksze], fun);

    mask = stddevim > thresh;
    maskind = find(mask);

    % Renormalise image so that the *ridge regions* have zero mean, unit
    % standard deviation.
    im = im - mean(im(maskind));
```

```
58     normim = im/std(im(maskind));
```

c) Função Orientação

```
1  % RIDGEORIENT - Estimates the local orientation of ridges in a fingerprint
2  %
3  % Usage:  [orientim, reliability, coherence] = ridgeorientation(im,
       gradientsigma,...
4  %                                              blocksigma, ...
5  %                                              orientsmoothsigma)
6  %
7  % Arguments:  im                - A normalised input image.
8  %             gradientsigma     - Sigma of the derivative of Gaussian
9  %                                 used to compute image gradients.
10 %             blocksigma        - Sigma of the Gaussian weighting used to
11 %                                 sum the gradient moments.
12 %             orientsmoothsigma - Sigma of the Gaussian used to smooth
13 %                                 the final orientation vector field.
14 %                                 Optional: if omitted it defaults to 0
15 %
16 % Returns:    orientim          - The orientation image in radians.
17 %                                 Orientation values are +ve clockwise
18 %                                 and give the direction *along* the
19 %                                 ridges.
20 %             reliability       - Measure of the reliability of the
21 %                                 orientation measure.  This is a value
22 %                                 between 0 and 1. I think a value above
23 %                                 about 0.5 can be considered 'reliable'.
24 %                                 reliability = 1 - Imin./(Imax+.001);
25 %             coherence         - A measure of the degree to which the
       local
26 %                                 area is oriented.
27 %                                 coherence = ((Imax-Imin)./(Imax+Imin))
       .^2;
28 %
29 % With a fingerprint image at a 'standard' resolution of 500dpi suggested
30 % parameter values might be:
31 %
32 %   [orientim, reliability] = ridgeorient(im, 1, 3, 3);
33 %
34 % See also: RIDGESEGMENT, RIDGEFREQ, RIDGEFILTER
35
36 % May 2003      Original version by Raymond Thai,
37 % January 2005  Reworked by Peter Kovesi
38 % October 2011  Added coherence computation and orientsmoothsigma
39 %               made optional
40 % April 2018    Change computation of gradients to use DERIAVTIVE5
```

```
41 %
42 % Peter Kovesi
43 % peterkovesi.com
44
45
46 function [orientim, reliability, coherence] = ...
47             ridgeorient(im, gradientsigma, blocksigma, orientsmoothsigma)
48
49     if ~exist('orientsmoothsigma', 'var'), orientsmoothsigma = 0; end
50     [rows,cols] = size(im);
51
52     % Calculate image gradients.
53     [Gx, Gy] = derivative5(gaussfilt(im, gradientsigma), 'x', 'y');
54
55     % Estimate the local ridge orientation at each point by finding the
56     % principal axis of variation in the image gradients.
57     Gxx = Gx.^2;       % Covariance data for the image gradients
58     Gxy = Gx.*Gy;
59     Gyy = Gy.^2;
60
61     % Now smooth the covariance data to perform a weighted summation of the
62     % data.
63     sze = fix(6*blocksigma);   if ~mod(sze,2); sze = sze+1; end
64     f = fspecial('gaussian', sze, blocksigma);
65     Gxx = filter2(f, Gxx);
66     Gxy = 2*filter2(f, Gxy);
67     Gyy = filter2(f, Gyy);
68
69     % Analytic solution of principal direction
70     denom = sqrt(Gxy.^2 + (Gxx - Gyy).^2) + eps;
71     sin2theta = Gxy./denom;            % Sine and cosine of doubled angles
72     cos2theta = (Gxx-Gyy)./denom;
73
74     if orientsmoothsigma
75         sze = fix(6*orientsmoothsigma);   if ~mod(sze,2); sze = sze+1; end
76         f = fspecial('gaussian', sze, orientsmoothsigma);
77         cos2theta = filter2(f, cos2theta); % Smoothed sine and cosine of
78         sin2theta = filter2(f, sin2theta); % doubled angles
79     end
80
81     orientim = pi/2 + atan2(sin2theta,cos2theta)/2;
82
83     % Calculate 'reliability' of orientation data.  Here we calculate the
       area
84     % moment about the orientation axis found (this will be the minimum
       moment)
85     % and an axis perpendicular (which will be the maximum moment).  The
```

```
86      % reliability measure is given by 1.0-min_moment/max_moment. The
        reasoning
87      % being that if the ratio of the minimum to maximum moments is close to
         one
88      % we have little orientation information.
89
90      Imin = (Gyy+Gxx)/2 - (Gxx-Gyy).*cos2theta/2 - Gxy.*sin2theta/2;
91      Imax = Gyy+Gxx - Imin;
92
93      reliability = 1 - Imin./(Imax+.001);
94      coherence = ((Imax-Imin)./(Imax+Imin)).^2;
95
96      % Finally mask reliability to exclude regions where the denominator
97      % in the orientation calculation above was small. Here I have set
98      % the value to 0.001, adjust this if you feel the need
99      reliability = reliability.*(denom>.001);
```

d) Função Frequência

```
1  % RIDGEFREQ - Calculates a ridge frequency image
2  %
3  % Function to estimate the fingerprint ridge frequency across a
4  % fingerprint image. This is done by considering blocks of the image and
5  % determining a ridgecount within each block by a call to FREQEST.
6  %
7  % Usage:
8  %  [freqim, medianfreq] =  ridgefreq(im, mask, orientim, blksze, windsze,
       ...
9  %                                    minWaveLength, maxWaveLength)
10 %
11 % Arguments:
12 %         im       - Image to be processed.
13 %         mask     - Mask defining ridge regions (obtained from
       RIDGESEGMENT)
14 %         orientim - Ridge orientation image (obtained from RIDGORIENT)
15 %         blksze   - Size of image block to use (say 32)
16 %         windsze  - Window length used to identify peaks. This should be
17 %                    an odd integer, say 3 or 5.
18 %         minWaveLength,  maxWaveLength - Minimum and maximum ridge
19 %                      wavelengths, in pixels, considered acceptable.
20 %
21 % Returns:
22 %         freqim     - An image  the same size as im with  values set to
23 %                      the estimated ridge spatial frequency within each
24 %                      image block.  If a  ridge frequency cannot be
25 %                      found within a block, or cannot be found within the
26 %                      limits set by min and max Wavlength freqim is set
```

```
27 %                       to zeros within that block.
28 %         medianfreq - Median frequency value evaluated over all the
29 %                       valid regions of the image.
30 %
31 % Suggested parameters for a 500dpi fingerprint image
32 %   [freqim, medianfreq] = ridgefreq(im,orientim, 32, 5, 5, 15);
33 %
34 % I seem to find that the median frequency value is more useful as an
35 % input to RIDGEFILTER than the more detailed freqim.  This is possibly
36 % due to deficiencies in FREQEST.
37 %
38 % See also: RIDGEORIENT, FREQEST, RIDGESEGMENT
39
40 % Reference:
41 % Hong, L., Wan, Y., and Jain, A. K. Fingerprint image enhancement:
42 % Algorithm and performance evaluation. IEEE Transactions on Pattern
43 % Analysis and Machine Intelligence 20, 8 (1998), 777 789.
44
45 % Peter Kovesi
46 % School of Computer Science & Software Engineering
47 % The University of Western Australia
48 % pk at csse uwa edu au
49 % http://www.csse.uwa.edu.au/~pk
50 %
51 % January 2005
52
53 function [freq, medianfreq] = ridgefreq(im, mask, orient, blksze, windsze,
       ...
54                                         minWaveLength, maxWaveLength)
55
56     [rows, cols] = size(im);
57     freq = zeros(size(im));
58
59     for r = 1:blksze:rows-blksze
60         for c = 1:blksze:cols-blksze
61           blkim = im(r:r+blksze-1, c:c+blksze-1);
62           blkor = orient(r:r+blksze-1, c:c+blksze-1);
63
64           freq(r:r+blksze-1,c:c+blksze-1) =  ...
65               freqest(blkim, blkor, windsze, minWaveLength, maxWaveLength);
66         end
67     end
68
69     % Mask out frequencies calculated for non ridge regions
70     freq = freq.*mask;
71
72     % Find median freqency over all the valid regions of the image.
```

```
73      medianfreq = median(freq(find(freq>0)));
```

B.2 Manipulação para entrada da Rede Neural

O mapeamento dos dados da impressão digital foi realizado pelo código:

```
1  function [Im] = neuralnetwork_mapping(caract, sz, tm)
2  % Developer Fellipe E G Silva (2019)
3
4  %% Calculate mapping factor
5  Im = zeros(tm,tm);
6  rz_x = tm/sz(1);
7  rz_y = tm/sz(2);
8  caract2 = caract;
9  len = length(caract);
10 %% Calculate new positions
11 for i=1:len
12     caract2(i,1) = round(caract(i,1)*rz_x);
13     caract2(i,2) = round(caract(i,2)*rz_y);
14     if(round(caract(i,1)*rz_x)==0)
15         caract2(i,1) = 1;
16     end
17     if (round(caract(i,2)*rz_y)==0)
18         caract2(i,2) = 1;
19     end
20 end
21 %% Creating output image
22 len2 = length(caract2);
23 for i=1:len2
24     Im(caract2(i,1),caract2(i,2)) = 1;
25 end
```

C Código para implementação do sistema

O código de implementação do sistema foi realizado em Python e necessita das funções implementadas no MATLAB. Segue o código de implementação:

```
import matlab.engine
def getID():
    '''
    First enter the port settings with menu option 1:
    >>> Enter Arduino serial port number: COM13
    >>> Enter serial port baud rate: 57600
    '''

    out = open(input("Enter with temp Path\Fingerprint File")+'.bmp', 'wb')
    # assemble and write the BMP header to the file
    out.write(assembleHeader(WIDTH, HEIGHT, DEPTH, True))
    for i in range(256):
        # write the colour palette
        out.write(i.to_bytes(1,byteorder='little') * 4)
    try:
        # open the port; timeout is 1 sec; also resets the arduino
        ser = serial.Serial(portSettings[0], portSettings[1], timeout=1)
    except Exception:
        print('Invalid port settings!')
        print()
        out.close()
        return
    while ser.isOpen():
        try:
            # assumes everything recved at first is printable ascii
            curr = ser.read().decode()
            # based on the image_to_pc sketch, \t indicates start of the
    stream
            if curr != '\t':
                # print the debug messages from arduino running the
    image_to_pc sketch
                # print(curr, end='')
                continue
            for i in range(READ_LEN): # start recving image
                byte = ser.read()
                # if we get nothing after the 1 sec timeout period
                if not byte:
                    print("Timeout!")
```

```
37                      out.close()  # close port and file
38                      ser.close()
39                      return False
40                  # make each nibble a high nibble
41                  out.write((byte[0] & 0xf0).to_bytes(1, byteorder='little'))
42                  out.write(((byte[0] & 0x0f) << 4).to_bytes(1, byteorder='
    little'))
43
44              out.close()  # close file
45              print('Image saved as', out.name)
46
47              # read anything that's left and print
48              left = ser.read(100)
49              print(left.decode('ascii', errors='ignore'))
50              ser.close()
51
52              print()
53              eng = matlab.engine.start_matlab()
54              ID = eng.Get_ID('Temp File\Fingerprint.bpm')
55
56              print(ID)
57              return True
58          except Exception as e:
59              print("ERROR: ", e)
60              out.close()
61              ser.close()
62              return False
63          except KeyboardInterrupt:
64              print("Closing port.")
65              out.close()
66              ser.close()
67              return False
```