UNIVERSIDADE FEDERAL DO PAMPA

ALEXANDRE AMARAL MOREIRA

RECONHECIMENTO DE PLACAS DE VEÍCULOS ATRAVÉS DA APLICAÇÃO DE TÉCNICAS DE PROCESSAMENTO DE IMAGENS E REDES NEURAIS ARTIFICIAIS

Bagé
2013

ALEXANDRE AMARAL MOREIRA

RECONHECIMENTO DE PLACAS DE VEÍCULOS ATRAVÉS DA APLICAÇÃO DE TÉCNICAS DE PROCESSAMENTO DE IMAGENS E REDES NEURAIS ARTIFICIAIS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal do Pampa, como requisito para obtenção do título de Engenheiro de Computação.

Orientador: Prof. Dr. Sandro da Silva Camargo

Bagé

2013

ALEXANDRE AMARAL MOREIRA

RECONHECIMENTO DE PLACAS DE VEÍCULOS ATRAVÉS DA APLICAÇÃO DE TÉCNICAS DE PROCESSAMENTO DE IMAGENS E REDES NEURAIS ARTIFICIAIS

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 19 de outubro de 2013.

Banca examinadora:

_____

Prof. Dr. Sandro da Silva Camargo
Orientador
UNIPAMPA

_____

Prof. MSc. Bruno Silveira Neves
UNIPAMPA

_____

Prof. MSc. Carlos Michel Betemps
UNIPAMPA

AGRADECIMENTOS

Ao Prof. Dr. Sandro da Silva Camargo, por sua orientação e paciência durante o período de realização deste trabalho.

Aos professores, minha gratidão por todo aprendizado e pela forma de conduzir o curso em todas as etapas.

A todos os colegas e amigos que estiveram ao meu lado durante o curso, por todas as risadas, brincadeiras e momentos de descontração que ajudavam a diminuir a tensão das provas e finais de semestres difíceis. Em especial, agradeço a Patrícia Silva Domingues, por toda ajuda dentro e fora da universidade.

Ao Centro Brasileiro de Pesquisas Físicas, pela disponibilização das amostras de imagens que permitiram a conclusão do trabalho.

A todas as pessoas que, direta ou indiretamente contribuíram para a realização deste trabalho.

“Não há nada melhor do que as adversidades. Cada derrota, cada mágoa, cada perda, contém sua própria semente, sua própria lição de como melhorar seu desempenho na próxima vez.”

Malcom X

RESUMO

As imagens digitais estão cada vez mais presentes na vida das pessoas, podendo ser utilizadas para trabalho ou lazer. Com a popularização da internet, sobretudo das redes sociais, tornou-se cada vez mais prático armazenar e compartilhar imagens com pessoas de diferentes partes mundo. Todas essas facilidades propiciadas pelas imagens digitais fizeram crescer uma necessidade de mecanismos de automação e tomada de decisão acerca desse assunto. Indústrias de componentes eletrônicos já utilizam o processamento de imagens digitais para verificar defeitos placas de circuitos eletrônicos, e diferentes áreas, como por exemplo, a medicina, conta com softwares que ajudam a detectar tumores ou outro tipo de doenças analisando imagens radiográficas. Com o intuito de suprir essa necessidade, focando em um determinado problema, este trabalho tem por objetivo desenvolver um software capaz de detectar, extrair e reconhecer caracteres alfanuméricos contidos em placas de automóveis. Em um primeiro momento, será apresentado o que já foi pesquisado e desenvolvido ao longo do trabalho, citando alguns conceitos e características sobre imagens digitais e redes neurais. Em seguida, serão comentadas as técnicas utilizadas para a implementação do software proposto, obedecendo aos passos para o reconhecimento de imagens. E por fim, serão apresentados os treinamentos e resultados que serviram para buscar o melhor modelo a ser utilizado para o reconhecimento de letras e números.

Palavras-chave: Placas de Automóveis, Imagem Digital, Redes Neurais.

ABSTRACT

Digital images are increasingly present in people's lives and can be used for work or leisure. With the popularization of the Internet, especially social networks, has become increasingly practical to store and share images with people from different parts of the world. All these facilities afforded by digital images have increased a need for mechanisms of automation and decision making about this subject. Electronic component industries already use digital image processing to be defective electronic circuit boards, and different areas, such as medicine, it uses software features that help detect tumors or other diseases by analyzing radiographic images. In order to meet this need, focusing on a particular problem, this paper aims to develop software able to detect, extract and recognize alphanumeric characters contained in vehicles plates. At first, you will see what has been researched and developed over the work, citing some concepts and characteristics of digital images and neural networks. Then will be commented the techniques used to implement the proposed software, following the steps for image recognition. Finally, we will present the results and training that served to seek the best model to be used for the recognition of letters and numbers.

Keywords: Digital Image, Neural Networks, Vehicles Plates.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CMYK – Sistema de cores (*Cyan*, *Magenta*, *Yellow*, *Key*)

FANN – *Fast Artificial Neural Network*

MLP – *MultLayer Perceptron*

OpenCV – *Open Source Computer Vision*

RNA – Rede Neural Artificial

RGB – Sistema de cores (*Red*, *Green*, *Blue*)

SUMÁRIO

1 INTRODUÇÃO

As imagens digitais vêm ganhando uma grande importância nos últimos tempos, isso se deve principalmente aos avanços tecnológicos que proporcionam ganhos em capacidade de processamento e armazenamento em dispositivos cada vez menores e mais robustos. Nos dias de hoje, muitos dispositivos eletrônicos contêm uma câmera digital integrada, como por exemplo, celulares e *tablets*. Aliados a essa facilidade, também é importante salientar que devido ao crescimento e popularização da internet, sobretudo das redes sociais, tornou-se muito prático o armazenamento e a distribuição de imagens digitais, podendo ser compartilhadas por diversas pessoas e sem precisar se preocupar com questões de deterioração. Esta realidade em nada lembra aqueles antigos álbuns de fotografias de décadas passadas.

Diversas outras áreas como entretenimento, publicidade, educação, indústria e até medicina se beneficiam das vantagens proporcionadas pelas imagens digitais. Todas as facilidades oriundas da imagem digital ajudaram a tornar essa uma área interdisciplinar, utilizando conceitos de informática, física e eletrônica, sendo cada vez mais utilizadas em pesquisas e soluções de problemas que necessitam de uma capacidade de automação e também uma tomada de decisão.

As informações dentro de uma imagem podem ser divididas em dois grandes grupos: texto e conteúdo. O primeiro grupo é mais direcionado para a área de banco de dados, onde as informações são armazenadas em formato textual, como por exemplo, busca por metadados de uma imagem. Já o segundo grupo é voltado para quem trabalha com o processamento de imagens, visão computacional, reconhecimento de padrões e áreas afins, que estão preocupados em descrever a imagem com os objetos contidos nela (ERPEN, 2004).

Dentro da área da visão computacional, existem diversas técnicas de processamento de imagem que permitem tratar as imagens como filtro de média que eliminam os ruídos nas regiões ao redor da borda de um objeto, a binarização, que muda uma imagem deixando apenas com as cores branco e preto, além dos filtros morfológicos, erosão e dilatação, que alteram a escalas dos objetos em uma imagem. Além dessas técnicas de processamento de imagens, outras tantas podem ser usadas dependendo do problema em questão.

A divisão dos elementos de uma imagem permite que sejam tratados apenas os objetos de interesse em uma imagem, isso reduz o trabalho de ferramentas responsáveis por reconhecer estes objetos. Nesse contexto, as redes neurais artificiais surgem como uma solução computacional eficaz para aplicações que necessitam de uma tomada de decisão. Uma RNA busca simular a capacidade cognitiva do ser humano, por meio de uma aprendizagem de modelos apresentado anteriormente visando uma boa capacidade de generalização. A apresentação de modelos é chamada de treinamento de uma rede neural, onde é informado os estímulos de entrada e a resposta esperada na saída. Dentre as diversas vantagens no uso de RNA, destaca-se a capacidade de não permitir que variações como escala e rotação de um modelo altere o valor de saída.

1.1 Objetivo do Trabalho

Este trabalho tem por objetivo desenvolver um software capaz de detectar, extrair e reconhecer os caracteres alfanuméricos contido em placas de automóveis.

Como objetivos específicos podem ser citados os seguintes itens:

- Manipular as imagens realçando as áreas de interesse dentro da mesma, para que nas etapas mais avançadas o reconhecimento obtenha êxito maior;
- Dividir a imagem em objetos com o intuito de focar o processamento apenas nas regiões que interessam à resolução do problema;
- Extrair informações dos objetos de interesse de forma que seja possível diferenciá-los dentro de um grupo de objetos de mesmo tipo;
- Treinar redes neurais, apresentando amostras de imagens contendo letras e números para que sejam capazes de reconhecer os caracteres alfanuméricos das placas de automóveis;

1.2 Estrutura do Trabalho

O trabalho está estruturado da seguinte forma:

No capítulo 1 serão apresentados uma visão geral do trabalho e os objetivos do mesmo. No capítulo 2 será mostrada a definição de imagem digital, além de

descrever os tipos de imagens digitais, as etapas de reconhecimento de padrões envolvidas no processo de reconhecimento de imagens e descritores de forma que são empregados para classificação de objetos. No capítulo 3 será abordado sobre redes neurais, citando suas principais características e a definição sobre sua estrutura básica na forma matemática, a topologia das redes neurais e o processo de aprendizagem. No capitulo 4 são citadas as etapas e as técnicas empregadas para localizar, segmentar e identificar os caracteres alfanuméricos das placas de veículos contidas nas imagens. O capitulo 5 apresentará os resultados que foram obtidos nos testes para a generalização dos modelos de redes neurais. E, por fim, no capitulo 6 será descrita a conclusão deste trabalho.

2 IMAGEM DIGITAL

Neste capitulo serão abordados alguns fundamentos sobre imagem digital. Inicialmente é apresentada uma definição de imagem digital e posteriormente algumas etapas do processamento de imagens no domínio da frequência. Segundo (ERPEN, 2004), de acordo com o problema a ser abordado, diferentes soluções devem ser empregadas, tornando importante o conhecimento sobre imagens digitais.

2.1 Introdução

De acordo com (GONZALEZ; WOODS, 2006), uma imagem digital pode ser definida como uma função bidimensional *f(x,y)* onde x e y são coordenadas espaciais e a amplitude de *f* em qualquer ponto (x,y) do plano representa o brilho da imagem (intensidade). Quando x, y e amplitude de *f* possuem valores finitos e discretos podemos dizer que a imagem é digital. As imagens digitais são formadas por pequenos elementos que representam a luminosidade em cada ponto da matriz, chamados de pixels (*picture elements*).

O ser humano é capaz de identificar a cor de um objeto devido ao reflexo da luz incidente nesse objeto. Para descrever uma imagem em uma forma matemática, podemos representar a função $f(x,y)$,denominada a fração de luz que vai ser transmitida ou refletida no ponto (x,y), como o produto entre a iluminância $i(x,y)$, que exprime a quantidade de luz que incide sobre o objeto, e as propriedades de refletância própria do objeto, representadas pela função $r(x,y)$ (FILHO; NETO, 1999).

$$f(x,y) = i(x,y) \cdot r(x,y) \quad (1)$$

2.2 Tipos de Imagem Digital

As imagens digitais são formadas por conjuntos de pixels. O pixel é o menor elemento de uma imagem e possui um valor associado a uma cor. Cabe salientar

que a quantidade de pixels de uma imagem varia de acordo com a resolução ou tipo de imagem digital.

2.2.1 Imagem Binária

São imagens onde os pixels só podem ter valor 0 (preto) ou 1(branco). Este tipo de imagem é usada principalmente em processamento de textos, pois, neste caso, só estamos interessados nas letras (preto) e no fundo (branco) (LIBERMAN, 1997).

Outro fator positivo desse tipo de imagem é o baixo consumo de armazenamento em disco, pelo fato de que cada pixel ocupa apenas 1 bit.

2.2.2 Imagem Monocromática

São imagens onde os pixels assumem entre 0 e N representando a intensidade do cinza, onde zero representa intensidade nula, cor preto, e o N representa a máxima intensidade de cinza, cor branco, valores intermediários representam tons de cinza. Geralmente N+1 é uma potência de dois, a forma mais comum utilizada é a de 256 tons que utilizam 8 bits por pixel de armazenamento.

2.2.3 Imagem Colorida

É um tipo de imagem multi-espectral que é formada por mistura de cores, dependendo do padrão utilizado, os mais conhecidos são o RGB (vermelho (*red*), verde (*green*), azul (*blue*)) e o CMYK(ciano (*cyan*), magenta (*magenta*), amarelo (*yellow*) e preto(que é a cor base do padrão, chave = *key*)). Um pixel representa a intensidade de cada banda imagem, assim, as imagens coloridas no padrão RGB são constituídas de 24 bits por pixel sendo 8 bits para representar o vermelho, 8 bits para representar o verde e 8 bits para representar o azul. Ocorrem casos onde é usado 1 byte para representar a transparência, assim a imagem fica constituída de 32 bits.

Com a combinação das três cores, pode-se formar uma imagem com até 16 milhões de cores e tonalidades distintas. Este número é perfeitamente adequado para representar a realidade sem perda de detalhes e qualidade em relação às cores, pois está acima da capacidade do olho humano em distinguir cores e tonalidades (CAMPOS, 2001).

2.3 Etapas do Reconhecimento de Padrões em Imagens

Os sistemas que utilizam o reconhecimento de padrões, de maneira geral, seguem as cinco etapas apresentadas na figura 1. Cada etapa contém funções e técnicas específicas que visam reduzir dimensões de busca por objetos e a redundância de dados. Um sistema de reconhecimento deve ser capaz de reconhecer um objeto independentemente de sua orientação, tamanho e posição na imagem (ERPEN, 2004).

Figura 1- Etapas de reconhecimento de uma imagem

Fonte: Adaptado de FILHO, O. M.; NETO, H. V. Processamento Digital de Imagens, 1999.

2.3.1 Aquisição da Imagem

Consiste em converter uma cena tridimensional real em uma imagem bidimensional ou sequência de imagens estáticas. Isso é feito através de sensores capazes de capturar os sinais luminosos emitidos ou refletidos por um objeto.

Podem ser citados diversos equipamentos com a finalidade de capturar imagens, como por exemplo: câmera digital, scanner, webcams, entre outros. Esses equipamentos possuem sensores que capturam as imagens de acordo com uma determinada taxa de amostragem que determina a resolução da imagem (SILVA, 2005).

2.3.2 Pré-processamento

É a etapa seguinte à aquisição da imagem com o objetivo de corrigir imperfeições e defeitos inseridos na imagem durante a etapa anterior.

Para correção das imagens são utilizados métodos específicos, como por exemplo, filtragem de ruídos e aumento de contraste, visando facilitar a extração de características e identificação dos objetos. A realização desta etapa aumenta as chances de sucesso dos processos seguintes (ERPEN, 2004).

2.3.3 Segmentação

A segmentação consiste em subdividir uma imagem em regiões e objetos que possuam propriedades em comum. O nível de detalhe que a subdivisão é feita depende do problema a ser resolvido.

Muitos algoritmos de segmentação são baseados em uma das duas propriedades básicas: descontinuidade e similaridade. Na primeira, é realizado um particionamento da imagem baseado na mudança abrupta na intensidade, como ocorre nas bordas. Na segunda categoria, os algoritmos são baseados no particionamento de uma imagem em regiões que são similares de acordo com o conjunto de critérios como técnicas de limiarização, crescimento por regiões, união e divisão de regiões (GONZALEZ; WOODS, 2006).

2.3.4 Extração de Características

A extração de feições tem por objetivo diminuir a redundância dos dados mantendo grande parte da informação útil para a discriminação entre as classes de objetos.

Após a imagem ter sido dividida em regiões de acordo com alguma similaridade é necessário organizar essas regiões de forma que se possa extrair alguma informação pertinente. Nessa etapa, podem-se citar duas formas de representação dos dados, por fronteiras (externa) ou por regiões completas (interna).

A representação de dados por fronteira é adequada quando a área de interesse é baseada em características externas, como por exemplo, as bordas. Já a representação por regiões é adequada quando o interesse está baseado em características internas (os pixels que compõem a região), tais como parâmetros estatísticos, cor e textura. Entretanto, dependendo da tarefa a ser realizada, as duas formas de representação podem ser utilizadas simultaneamente (ERPEN,2004).

2.3.5 Reconhecimento e Interpretação

A ultima etapa do reconhecimento de imagens pode ser considerado uma etapa de alto nível, pois diferentemente das etapas anteriores onde se empregava conjuntos de formulas teóricas, na etapa de classificação são necessários mecanismos de tomada de decisão (CAMPOS, 2001).

De acordo com (ERPEN, 2004), podemos considerar que a fase de classificação consiste em reconhecer um objeto, uma forma ou, de modo geral, interpretar as características particulares da imagem e assim decidir a que classe o padrão pertence. Um exemplo prático de classificação seria verificar em um alfabeto se um pequeno conjunto de símbolos é válido.

3 REDES NEURAIS ARTIFICIAIS

Este capítulo abordará sobre redes neurais artificiais. Primeiramente será apresentada uma introdução que trata de como surgiram as redes neurais e as semelhanças com o cérebro humano. A segunda etapa mostrará uma definição matemática de um neurônio empregado em uma rede neural, explicando suas características. Em seguida mostrará as principais arquiteturas utilizadas nas construções de redes neurais e, por fim, será abordado o processo de aprendizagem que ocorre nas redes neurais artificiais.

3.1 Introdução

Os trabalhos envolvendo redes neurais artificiais se tornaram cada vez mais frequentes pela capacidade de criar sistemas inteligentes imitando a inteligência humana, simplificando a estrutura do sistema nervoso, responsável pelas funções de inteligência (raciocínio, integração de ideias, etc.). O principal órgão para o funcionamento deste sistema é o cérebro e possui como seu elemento básico o neurônio (LIBERMAN, 1997).

O cérebro é um sistema de processamento de informação altamente complexo capaz de realizar processamento (visão e controle motor, por exemplo) mais rápido do que um computador digital moderno. Ainda, o cérebro tem a capacidade de se adaptar ao meio ambiente através da experiência adquirida com o tempo (HAYKIN, 2001).

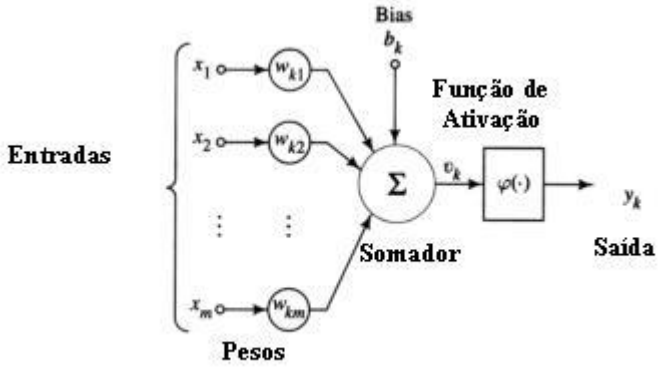
Para Braga, Carvalho e Ludemir (2000), “as redes neurais artificiais (RNAs) são modelos matemáticos que se assemelham às estruturas neurais biológicas e que tem capacidade computacional adquirida por meio de aprendizado e generalização”. O aprendizado, ou treinamento, é a fase onde os dados são apresentados a rede, para que esta ajuste seus parâmetros de entrada. Esta é uma etapa fundamental, pois a partir dela a RNA começa a se adaptar às características intrínsecas de um problema, buscando por meio de melhoras gradativas, uma boa capacidade de resposta para o maior número de situações possíveis. Por sua vez, a generalização permite que a RNA dê uma resposta coerente para dados que não

foram apresentados a ela na fase de aprendizado. É esperado que a rede neural apresente uma capacidade de generalização, independente de isso ter sido levado em consideração na etapa anterior.

3.2 Modelo Matemático de um Neurônio

Segundo Haykin (2001), “Um neurônio é uma unidade de processamento de informação que é fundamental para a operação de uma rede neural.”. Na figura abaixo é mostrado um modelo de um neurônio artificial introduzido por McCulloch e Pitts (1943).

Figura 2: Modelo de neurônio artificial.



Fonte: (HAYKIN, 2001, p. 36)

Pela figura 2, podemos destacar os três elementos básicos de um modelo neuronal:

1- Um conjunto de sinapses (entradas) ou elos de conexão contendo um peso próprio. Cada sinal de entrada (estímulo) x_j da sinapse j do neurônio k é multiplicado por um peso sináptico w_{kj}.
2- Um somador para os sinais de entrada, ponderados pelos respectivos pesos sinápticos.
3- Uma função de ativação, também chamada de função restritiva, que restringe a amplitude de saída do neurônio. Normalmente a amplitude de saída do

neurônio está contida em um intervalo fechado [0,1] ou alternativamente [-1,1].

Existe uma variação do modelo de neurônio apresentado onde o bias, representado por b_k, é aplicado como um peso neural. O bias tem o efeito de aumentar ou diminuir a entrada liquida da função de ativação.

Matematicamente, podemos descrever o campo local induzido e a saída de um neurônio k, respectivamente, utilizando as seguintes equações:

$$u_k = \sum_{j=1}^{m} w_{kj}\, x_j + b_k \tag{2}$$

$$y_k = \varphi(u_k) \tag{3}$$

onde u_k é a saída do somador dos sinais de entrada ponderados, φ é a função de ativação e y_k é a saída do neurônio.

A função de ativação $\varphi(u)$, define a saída de um neurônio em termos do campo local induzido (também chamado de potencial de ativação) por u. As classificações das funções de ativação serão descritas adiante.

3.2.1 Tipos de Funções de Ativação

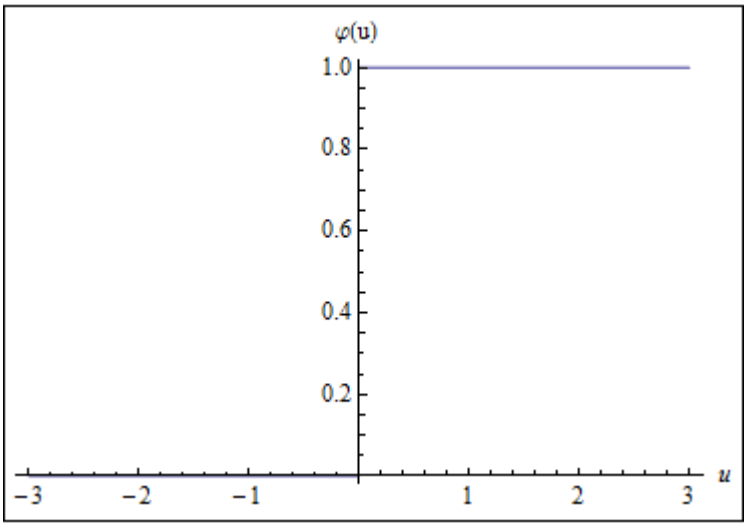
Haykin(2001) descreve em seu livro “Redes Neurais” que as funções de ativação podem ser classificadas como: função de limiar, função linear por partes ou função sigmóide.

O neurônio que emprega na sua saída uma função de limiar (degrau) é chamado de modelo McCulloch-Pitts, em reconhecimento ao trabalho de McCulloch e Pitts (1943). Neste modelo a função de ativação tem o valor 1 (um) quando o campo local induzido é positivo e o valor 0 (zero) quando o campo local induzido é negativo, seguindo a descrição da propriedade tudo-ou-nada do modelo McCulloch-Pitts. A equação 4 mostra a forma binária da função degrau, já a equação 5

apresenta a forma bipolar da mesma e os gráficos destas funções são apresentados nas figuras 3 e 4, respectivamente.

$$\varphi(u) = \begin{cases} 1 & se\ u \geq 0 \\ 0 & se\ u < 0 \end{cases} \tag{4}$$

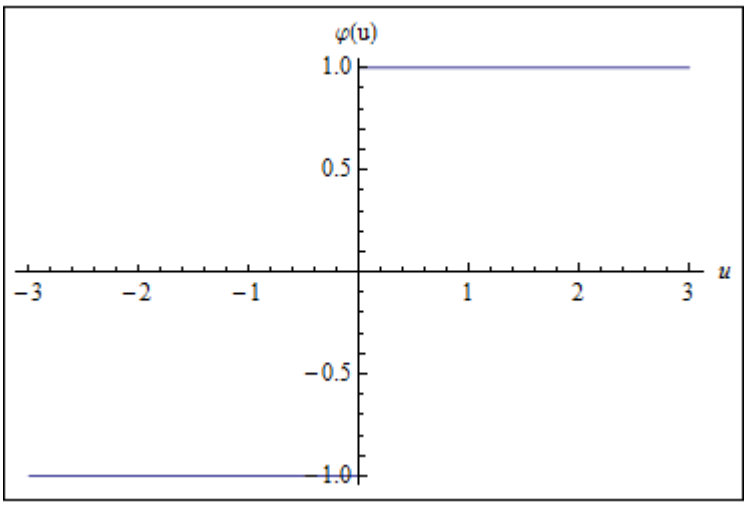
Figura 3: Gráfico da forma binária da função de limiar.



Fonte: Próprio autor.

$$\varphi(u) = \begin{cases} 1 & se\ u \geq 0 \\ 0 & se\ u = 0 \\ -1 & se\ u < 0 \end{cases} \tag{5}$$

Figura 4: Gráfico da forma bipolar da função de limiar.

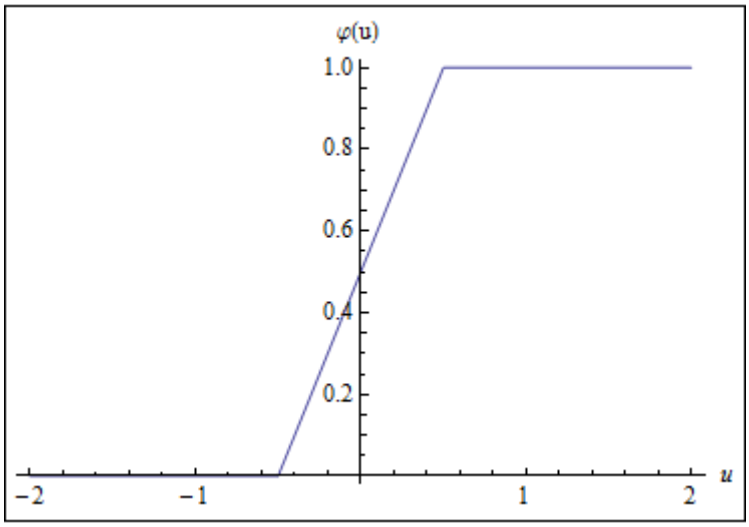


Fonte: Próprio autor.

A função linear por partes é um tipo de função que permite que a função de ativação seja continua e assuma qualquer valor real. A equação 6 descreve este tipo de função e a figura 5 apresenta o gráfico da função:

$$\varphi(u) = \begin{cases} 1, & se\ u \geq +\frac{1}{2} \\ u + \frac{1}{2}, & -\frac{1}{2} < u < +\frac{1}{2} \\ 0, & se\ u \leq -\frac{1}{2} \end{cases} \tag{6}$$

Figura 5: Gráfico da função linear binária.

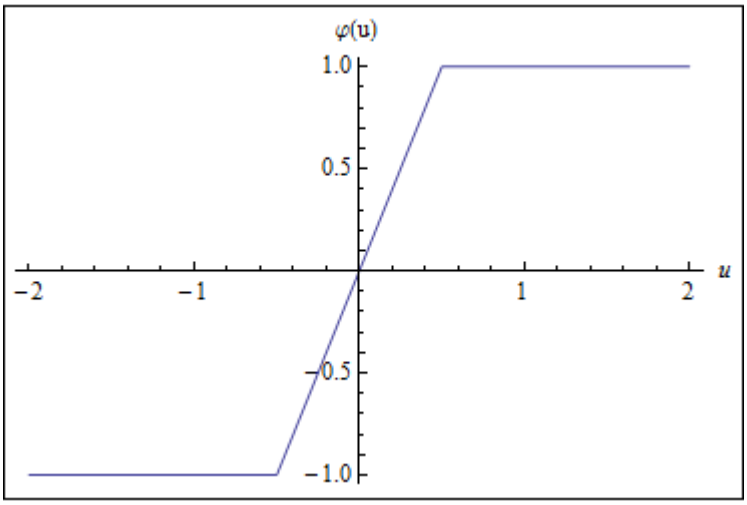


Fonte: Próprio autor.

A forma bipolar da função linear é mostrada na equação 7 abaixo e no gráfico 6 a seguir

$$\varphi(u) = \begin{cases} 1, & se\ u \geq +\frac{1}{2} \\ 2u, & -\frac{1}{2} < u < +\frac{1}{2} \\ -1, & se\ u \leq -\frac{1}{2} \end{cases} \quad (7)$$

Figura 6: Gráfico da função linear bipolar.



Fonte: Próprio autor.

Duas situações que podem ocorrer gerando formas especiais da função linear por partes:

1. O surgimento de um combinador linear, para o caso a região linear não entrar em saturação.
2. Quando o fator de amplificação da região tender ao infinito, esta função pode ser reduzida a uma função de limiar.

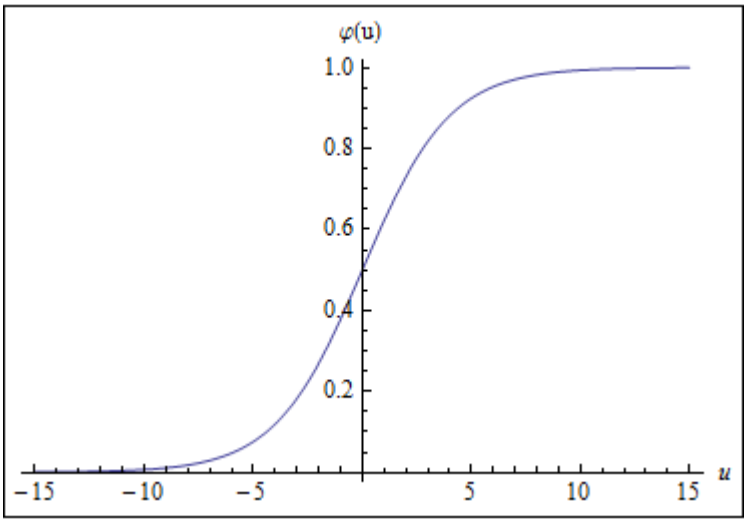
Por fim, a função sigmóide é a função de ativação mais utilizada na construção de redes neurais, seu gráfico possui a forma de um S e se caracteriza por ser uma função crescente que apresenta um balanceamento adequado entre os comportamentos lineares e não-lineares. Esta função pode ser utilizada na forma de função logística, quando se deseja utilizar um intervalo de valores binários, e função

tangente hiperbólica, para intervalos bipolares. A função logística é definida pela expressão:

$$\varphi(u) = \frac{1}{1+ e^{-au}} \quad (8)$$

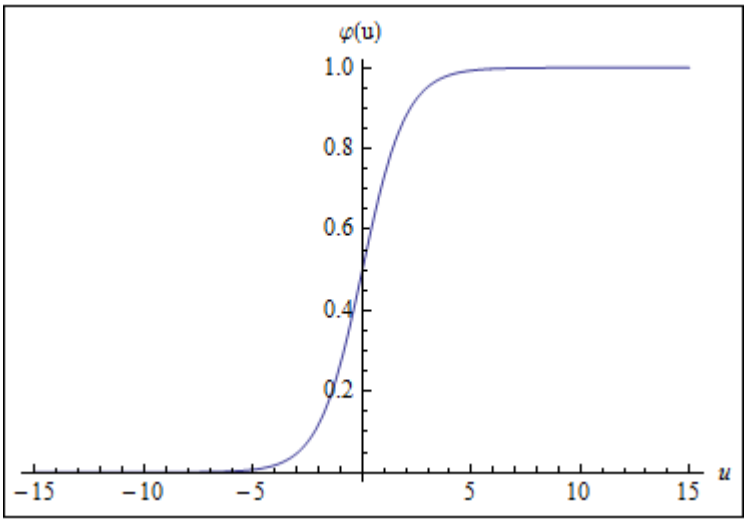
onde a é o parâmetro de inclinação da função logística. Conforme o parâmetro de inclinação varia, diferentes funções sigmóides são obtidas, de acordo com os gráficos das figuras 7, 8 e 9:

Figura 7: Gráfico da função logística (a = 0,5).



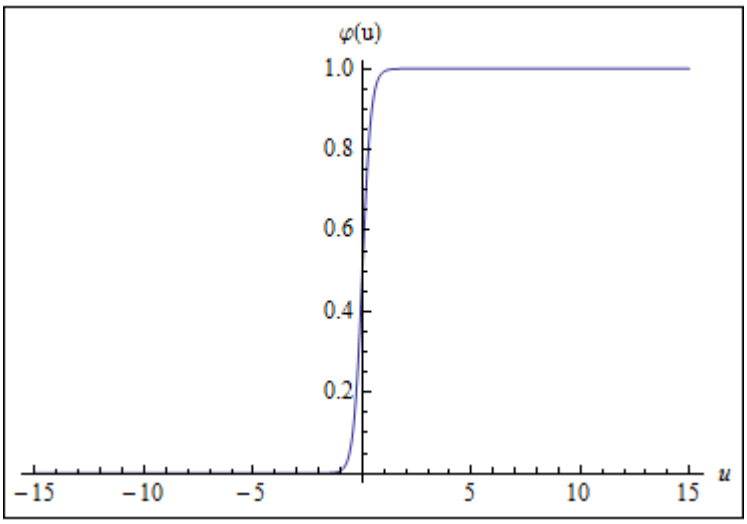
Fonte: Próprio autor.

Figura 8: Gráfico da função logística (a = 1,0).



Fonte: Próprio autor.

Figura 9: Gráfico da função logística (a = 5,0).



Fonte: Próprio autor.

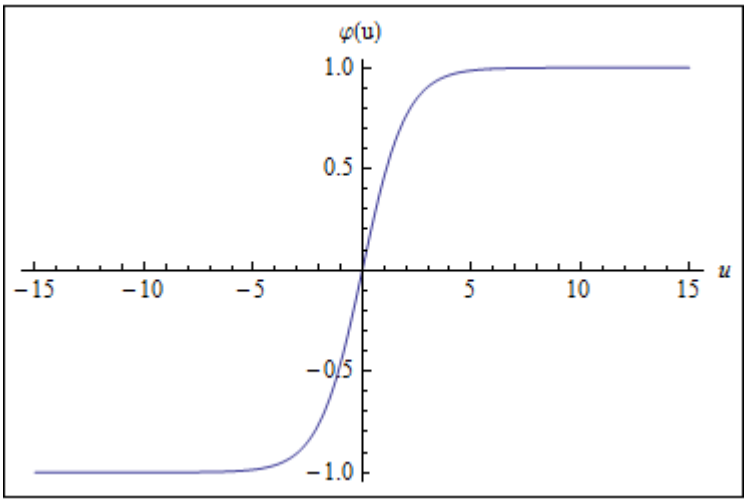
Pela figura 9 percebesse que quando a se aproxima do infinito, a função logística se torna uma função de limiar. Cabe ressaltar que, enquanto a função de limiar assume o valor de 0 ou 1, a função logística assume um valor continuo no intervalo de 0 e 1, outro ponto importante é que a função sigmóide é diferenciável enquanto a função degrau não é.

Dependendo do problema, pode ser desejável obter um valor bipolar para a função de ativação, utilizamos o correspondente para a função sigmóide que é a função tangente hiperbólica, descrita na seguinte equação:

$$\varphi(u) = \frac{e^{at} - e^{-at}}{e^{at} + e^{-at}} \tag{9}$$

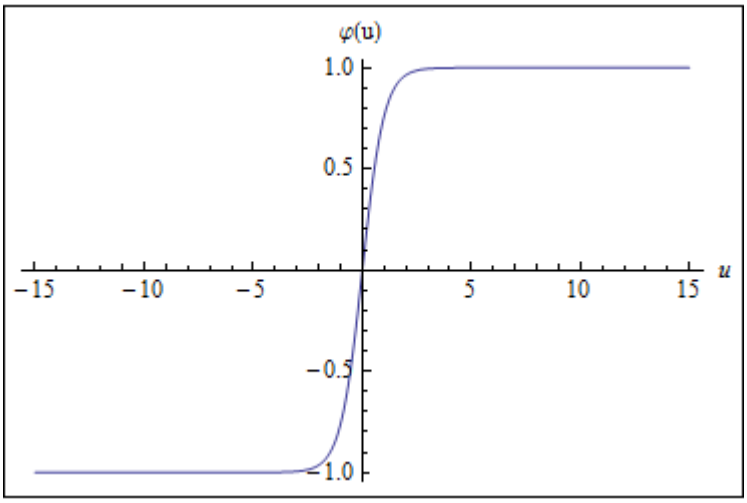
Os gráficos para a função tangente hiperbólica são mostrados nas figuras 10, 11 e 12.

Figura 10: Gráfico da função tangente hiperbólica (a = 0,5).



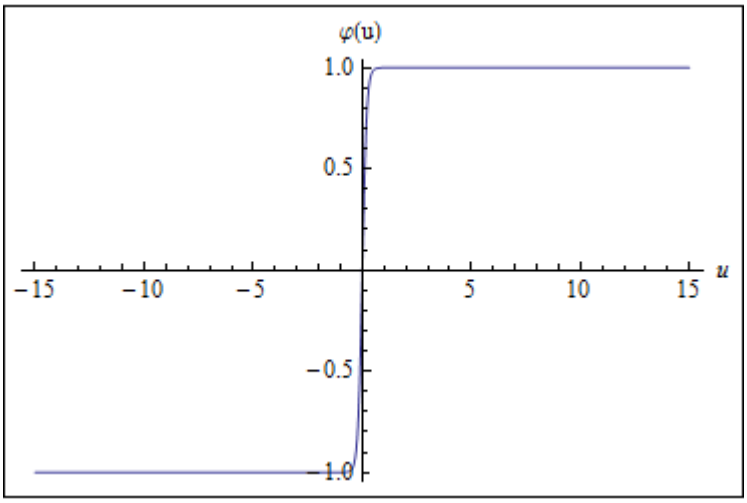
Fonte: Próprio autor.

Figura 11: Gráfico da função tangente hiperbólica (a = 1,0).



Fonte: Próprio autor.

Figura 12: Gráfico da função tangente hiperbólica (a = 5,0).



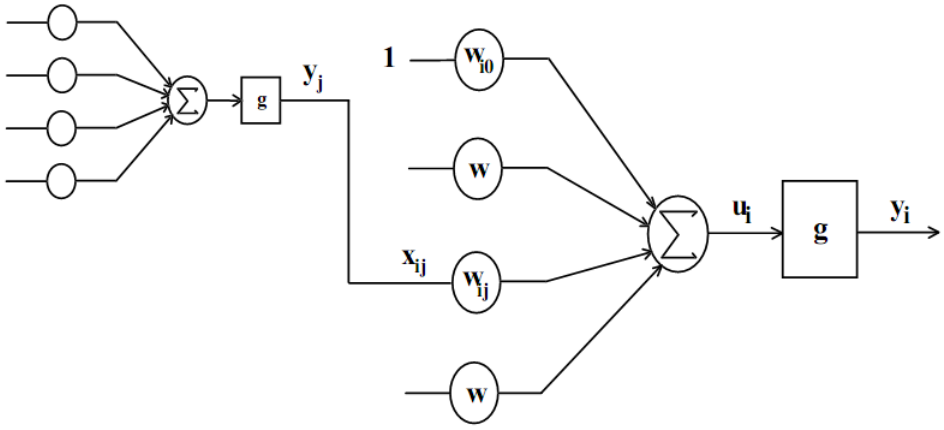
Fonte: Próprio autor.

3.3 Arquiteturas de Rede

A arquitetura utilizada em uma rede neural está relacionada com o algoritmo de aprendizagem que será empregado no treinamento da rede. Existem diversos algoritmos para treinamento de redes neurais e eles serão citados mais adiante, porém, este trabalho se deterá apenas no algoritmo que foi utilizado para o desenvolvimento prático deste.

As RNAs utilizam arquiteturas padronizadas, projetadas para solucionar algumas classes de problemas, por meio das conexões entre neurônios artificiais que levam à geração de sinapses e à construção de redes neurais artificiais (CASTRO, ZUBEN; [20--]).

Figura 13: Conexão entre neurônios artificiais.



Fonte: (CASTRO; ZUBEN, [20--]).

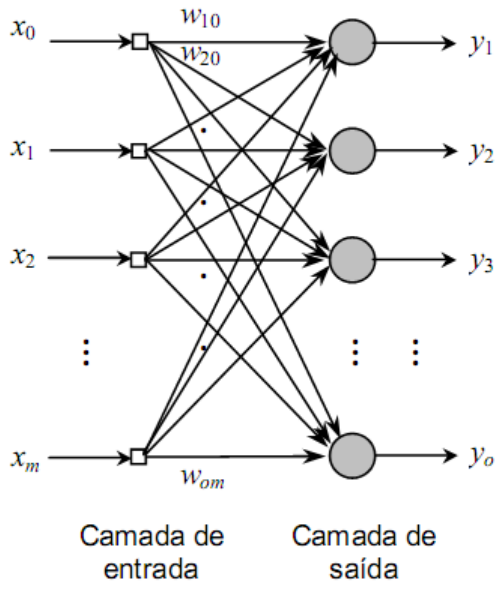
As arquiteturas de redes neurais podem ser classificadas em três tipos: redes *feedforward* de uma única camada, redes *feedforward* de múltiplas camadas, e redes recorrentes.

3.3.1 Redes Feedforward de Uma Única Camada

Um RNA é composto por basicamente três camadas, camada de entrada de nós de fonte de dados, a camada intermediaria e a camada de saída. A forma mais simples é a rede *feedforward* de uma única camada, onde os nós de fonte são projetados em direção a camada de saída, sentido positivo, não havendo retorno do sinal, em outras palavras, esta rede é dita alimentada adiante ou acíclica. (HAYKIN, 2001)

A figura 14 apresenta um modelo deste tipo de rede, a denominação “única camada” refere-se ao fato do processamento ser realizado apenas na camada de saída.

Figura 14: Exemplo de rede *feedforward* de uma única camada.



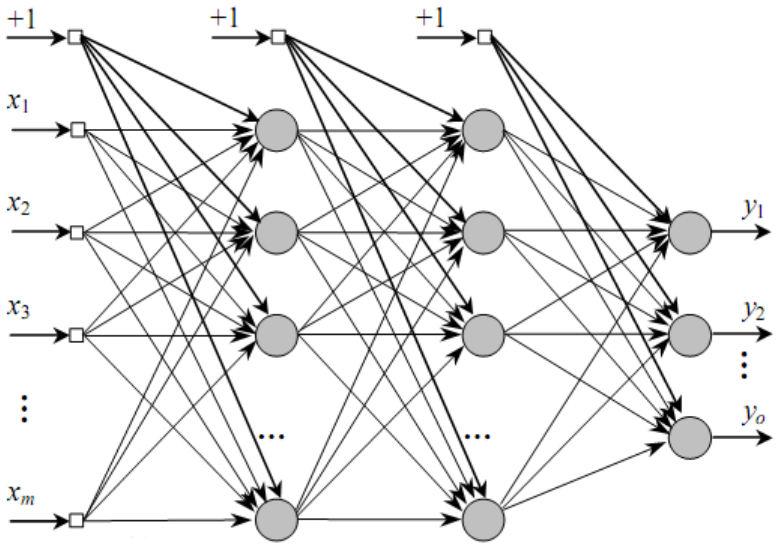
Fonte: (CASTRO; ZUBEN, [20--]).

3.3.2 Redes Feedforward de Múltiplas Camadas

As RNAs de múltiplas camadas se diferem da primeira por apresentarem uma camada ou mais camadas intermediarias, que são chamadas de camadas ocultas. A intenção de adicionar camadas ocultas tem por objetivo aumentar a capacidade de processamento da rede, o que pode ser muito vantajoso dependendo da quantidade de dados que estão presentes na entrada da rede. (HAYKIN, 2001)

As respostas geradas pela camada oculta são utilizadas como entrada para os neurônios da camada de saída, conforme figura 15. O algoritmo de treinamento para esse tipo de RNA envolve uma comparação da saída gerada na ultima camada com um valor desejado para a saída e posteriormente, a retropropagação deste erro (CASTRO; ZUBEN, [20--]).

Figura 15: Exemplo de rede *feedforward* de múltiplas camadas.



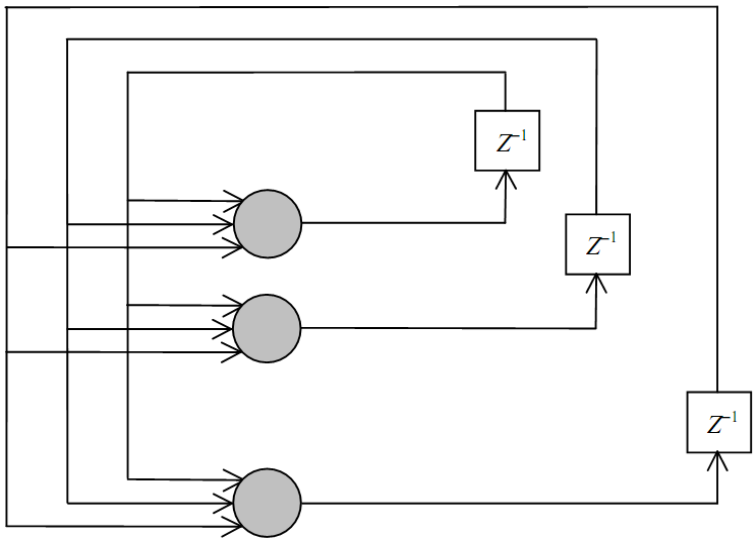
Fonte: (CASTRO; ZUBEN, [20--]).

3.3.3 Redes Recorrentes

As redes recorrentes são um tipo de rede que propaga no sentido positivo que se caracteriza por possuir pelo menos um laço de realimentação que é utilizado como fonte para os outros neurônios da rede. Sua estrutura pode conter apenas a camada de saída como também uma camada oculta para o processamento.

Os laços de realimentação apresentam elementos de atraso unitário (representados por z^{-1}). A utilização dos laços implica em um impacto profundo na capacidade de aprendizagem e no desempenho da rede.

Figura 16: Exemplo de rede recorrente.



Fonte: (CASTRO; ZUBEN, [20--]).

3.4 Processo de Aprendizagem

A aprendizagem em uma rede neural artificial é realizada por meio de exemplos. Estes exemplos são utilizados como referencia para que o algoritmo de aprendizagem ajuste os parâmetros da rede. A mudança dos parâmetros da rede acarreta em uma mudança no comportamento da rede neural, a fim de buscar um melhor desempenho de forma gradual. Conforme os exemplos são apresentados, é natural que a adaptação dos parâmetros leve a convergência para uma solução, sendo que o critério de convergência varia conforme o algoritmo e o paradigma de aprendizado (KARRER et al., 2005).

Existem basicamente dois paradigmas no processo de aprendizagem: o aprendizado com professor, também chamado de aprendizado supervisionado, e o aprendizado sem professor, ou aprendizado não-supervisionado. No aprendizado supervisionado, em virtude do conhecimento prévio, o professor é capaz de fornecer à rede a resposta desejada para o conjunto de treinamento. Os parâmetros da rede são ajustados pela combinação dos estímulos da rede com o sinal de erro, sendo este definido como a diferença entre a resposta desejada, ou seja, a resposta fornecida pelo professor e a resposta da rede aos estímulos.

Por outro lado, na aprendizagem sem um professor, como o próprio nome diz, não há conhecimento prévio da resposta ideal para o conjunto de treinamento. Para esse paradigma existem duas subdivisões: aprendizagem por reforço e a aprendizagem não-supervisionada. Na aprendizagem por reforço é feito um mapeamento entrada-saída contínuo com o ambiente visando à minimização de um índice escalar de desempenho. E na aprendizagem não-supervisionada é realizada uma medida independente da tarefa realizada da qualidade de representação que a rede deve aprender e esta medida é utilizada para o ajuste dos parâmetros (HAYKIN, 2001).

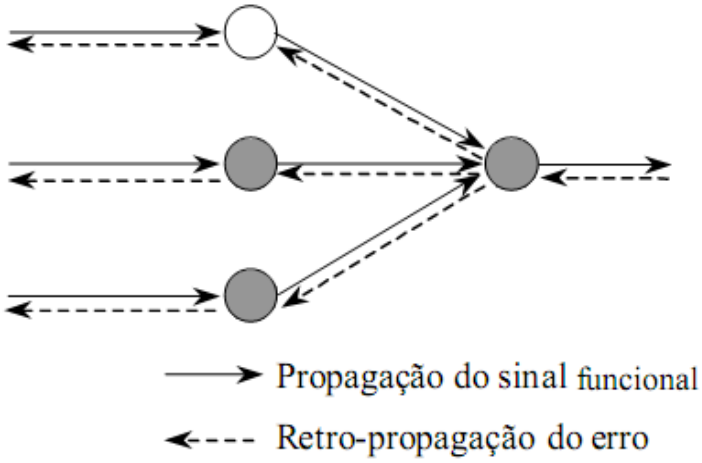
3.5 O Backpropagation

O *backpropagation* é um algoritmo utilizado para treinamento supervisionado de redes neurais do tipo *MultLayer Perceptron* (MLP). É um dos algoritmos de treinamento mais utilizados devido sua grande capacidade de generalização e adaptação aos mais diversos tipos de modelos (LIBERMAN, 1997). O treinamento com esse algoritmo consiste basicamente em apresentar um modelo à entrada da rede, propagar o sinal até a saída, retornar o erro da saída de volta para entrada para ajustar os pesos sinápticos.

Segundo Silva (1998), existem dois tipos de sinais apresentados nessa rede:

- Sinal funcional: estimulo da entrada da rede que é propagado positivamente pelos neurônios chegando à saída do RNA;
- Sinal de erro: é o sinal que é retro propagado até a entrada, com a função de ajuste dos pesos, gerado pela diferença entre a saída desejada e sinal de saída da rede.

Figura 17: Fluxo do sinal funcional e sinal de erro através de um neurônio.



Fonte: (Silva, 1998).

Em Haykin (2001), quando o neurônio esta localizado na saída da rede é possível calcular o sinal de erro associado a este neurônio e a fórmula para ajuste dos pesos dos neurônios de saída é dado por:

$$\delta_j(n) = e_j(n) \,.\, \varphi'_j(u_j(n)) \tag{10}$$

Onde: δ_j : gradiente para o neurônio de saída j

e_j : erro do neurônio de saída j

φ'_j : derivada da função de ativação do neurônio de saída j

u_j : campo local induzindo no neurônio de saída j

Para um neurônio localizado na camada oculta não existe uma resposta desejada especifica. Nesse ponto o sinal de erro é obtido de forma recursiva, levando em consideração o sinal de erro de todos os neurônios com os quais o neurônio oculto está diretamente ligado. Utilizando o gradiente calculado para os neurônios da camada de saída, obtemos a seguinte formula da retro propagação:

$$\delta_j(n) = \varphi'_j\left(u_j(n)\right)\sum_k \delta_k(n)\,\omega_{kj}(n) \quad (11)$$

Onde: j: refere-se à camada oculta.

k: refere-se à camada de saída.

ω_{kj}: pesos sinápticos conectados da camada oculta a camada de saída da rede.

A correção dos pesos que conecta um neurônio i ao neurônio j utiliza a regra delta, dada pela formula:

$$\Delta\omega_{ji}(n) = \eta\,\delta_j(n)\,y_i(n) \quad (12)$$

Onde: $\Delta\omega_{ji}$: correção dos pesos

y_i: sinal de entrada do neurônio j

η: taxa de aprendizagem

A taxa de aprendizagem η tem a função de controlar a velocidade da aprendizagem. Um valor baixo para a taxa de aprendizagem pode tornar o aprendizado mais lento, pois as variações dos pesos sinápticos serão mais lentas, enquanto que um valor alto acelera o aprendizado, mas pode causar instabilidade na rede devido a grandes modificações nos pesos sinápticos.

4 METODOLOGIA

Com o intuito de focar o trabalho para ser utilizado em um ambiente específico, tomaremos como estudo de caso um determinado local que utilize o software para restringir a entrada de veículos não autorizados. Com isso, tomamos como base um cenário onde a câmera esteja disposta na frente do veículo, a altura de um metro do chão, com a iluminação mínima para a captura da imagem e o veículo não esteja em movimento.

O software será projetado para reconhecer os caracteres de placas de identificação de automóveis que seguem as normas do Código Brasileiro de Trânsito, que determina que os veículos devam ser identificados contendo 7 (sete) caracteres alfanuméricos individualizados, sendo o primeiro grupo composto por 3(três) letras e o segundo grupo composto por 4 (quatro) algarismos.

O software proposto foi desenvolvido utilizando a linguagem de programação C++ e as bibliotecas OpenCV(*Open Source Computer Vision*) e FANN(*Fast Artificial Neural Network*). O OpenCV é uma biblioteca de uso livre e contém diversos algoritmos otimizados para as análises de imagem e vídeo. OpenCV foi originalmente desenvolvida pela Intel como uma iniciativa para pesquisas na área de visão computacional (LAGANIÈRE, 2011). A FANN é também uma biblioteca *open source* para o desenvolvimento de redes neurais artificiais multicamadas em C/C++ que possui diversas características a serem empregadas em variados problemas que requerem uma tomada de decisão.

Para validar as técnicas empregadas neste trabalho, a aquisição das imagens se deu por meio da utilização de uma amostra de 600 imagens, do tipo monocromática e com dimensões de 640x240 pixels, representando as condições reais encontradas diariamente, disponibilizadas pelo Projeto de Reconhecimento de Placas de Veículos Brasileiros do Centro Brasileiro de Pesquisas Físicas. Embora as imagens tenham sido disponibilizadas para a comunidade acadêmica que tenha interesse em pesquisas sobre reconhecimento de imagens, cerca de 72 imagens não puderam ser utilizadas por motivos que variam da falta de iluminação adequada no local da placa até a deterioração física da mesma, tornando inviável o reconhecimento até mesmo para um observador humano, algumas imagens podem ser vistas nas figuras 18a e 18b:

Figura 18: Imagens não utilizadas por (a) deterioração e (b) falta de iluminação adequada.

(a) (b)

Fonte: Disponibilizadas pelo Projeto de Reconhecimento de Placas de Veículos Brasileiros do Centro Brasileiro de Pesquisas Físicas.

Adiante serão mostradas as etapas que constituem o processo de reconhecimento de imagens e as técnicas utilizadas nas mesmas.

4.1 Pré-processamento

A partir dessa etapa a imagem começará a ser tratada para facilitar o processo de análise e tomada de decisão. O pré-processamento é um passo importante, pois através deste é possível reduzir ou eliminar ruído, melhorar a informação sobre características significantes e eliminar informações desnecessárias. Para o tratamento das imagens foi utilizado nesta etapa o filtro gaussiano através da operação de convolução, binarização e dilatação de imagens.

4.1.1 Filtro Gaussiano

O filtro gaussiano tem a função de reduzir a amplitude nos pontos em que a imagem apresenta maiores variações, geralmente nas bordas de um objeto, com a finalidade de suavização da imagem, diminuindo as variações rápidas de frequências. Uma maneira simples de alcançar este objetivo é substituir cada pixel pelo valor médio dos pixels ao seu redor (LAGANIÈRE, 2011). O filtro gaussiano é dado pela forma matricial com dimensões 3x3 descrita na equação 13:

$$h(x,y) = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}, \text{ onde x e y são as dimensões da imagem.} \quad (13)$$

A aplicação desse filtro sobre a imagem se dá por meio de uma operação chamada convolução. A convolução utiliza o conceito de máscaras ou janela de convolução que possuem dimensões 3x3, 5x5 ou 7x7, sendo estas as mais utilizadas, o processo se dá através de uma multiplicação elemento a elemento entre a máscara e uma sub-área da imagem, de mesmo tamanho que a janela de convolução, o resultado de cada uma destas multiplicações é somado e o resultado final desta soma é então colocado na posição central da máscara da imagem resultante na saída (FLORES, 2012).

4.1.2 Binarização

Depois de aplicado o filtro gaussiano, o passo seguinte é a binarização da imagem. Conforme descrito anteriormente uma imagem binária possui apenas duas cores, o preto representado pelo valor 0 (zero) e o branco representado pelo valor 1 (um). A binarização é responsável por destacar as áreas de interesse dentro da imagem, no caso os caracteres da placa.

Para transformar uma imagem monocromática em binária é necessário determinar um limiar (*threshold*), onde os pixels com valores menor que o *threshold* são igualados a zero (cor preto) e os pixels com valor maior, são igualados a um (cor branco). Embora em um primeiro momento a definição do limiar pareça trivial, a escolha de um valor adequado não é uma tarefa simples, pois o valor que pode ser bom para uma imagem pode ser ruim para outras, conforme figuras 19 e 20.

Figura 19: Boa aproximação de limiar (*threshold* = 145).

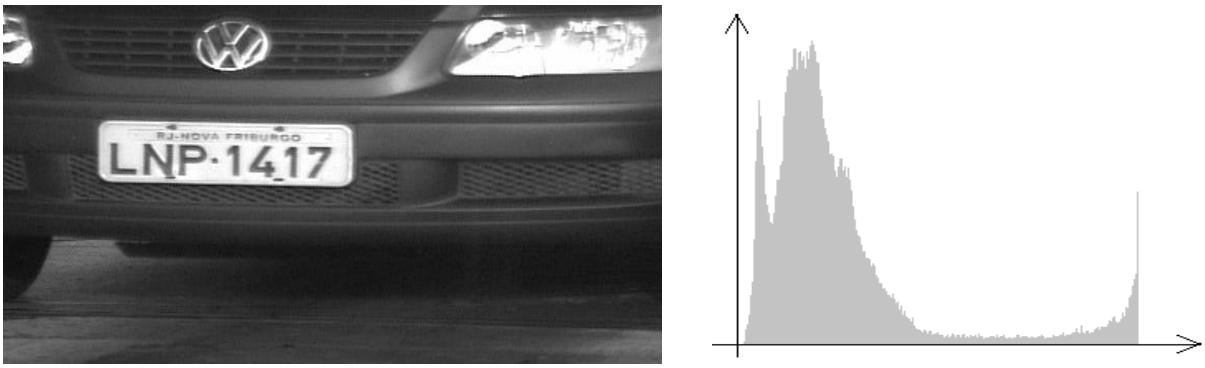
Fonte: Próprio autor.

Figura 20: Péssima aproximação de limiar (*threshold* = 95).

Fonte: Próprio autor.

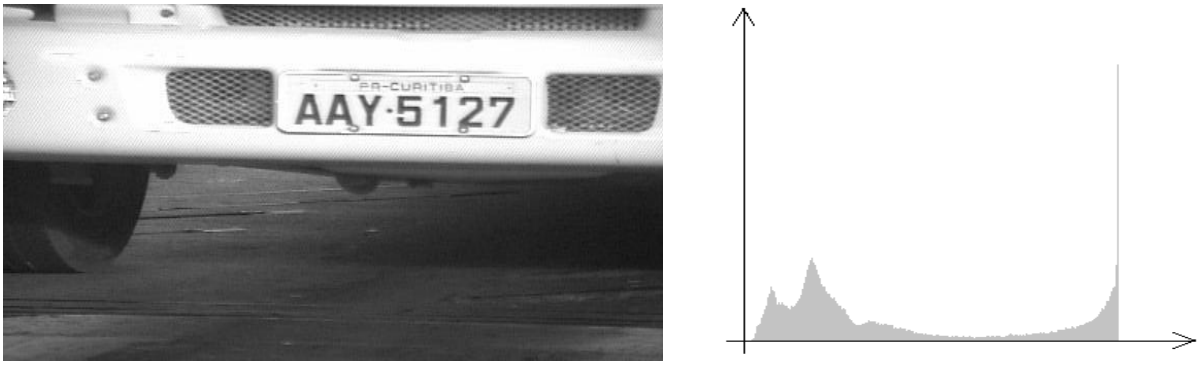
Uma técnica que garante uma boa aproximação é o método de Otsu, este método consiste na criação de um histograma de intensidade de pixels da imagem. O histograma de uma imagem representa a quantidade de pixels de cada tom de cinza presente na imagem, geralmente o histograma é representado por um gráfico de barras onde é possível verificar características da imagem como o brilho e o nível de contraste da imagem (SHAPIRO; STOCKMAN, 2001).

Figura 21: Imagem do veículo de cor escura e histograma da imagem.



Fonte: Próprio Autor.

Figura 22: Imagem do veículo de cor claro e histograma da imagem.



Fonte: Próprio Autor.

A abordagem por histograma utilizada assume que há valores médios para o fundo da imagem e o objeto. Ikeda (2011) apresenta em seu trabalho o seguinte pseudocódigo que descreve o funcionamento do método:

Considere as seguintes somas parciais, para uma intensidade k que variando de 1 até L, onde L é total de intensidades representados no histograma. Usando as somas parciais:

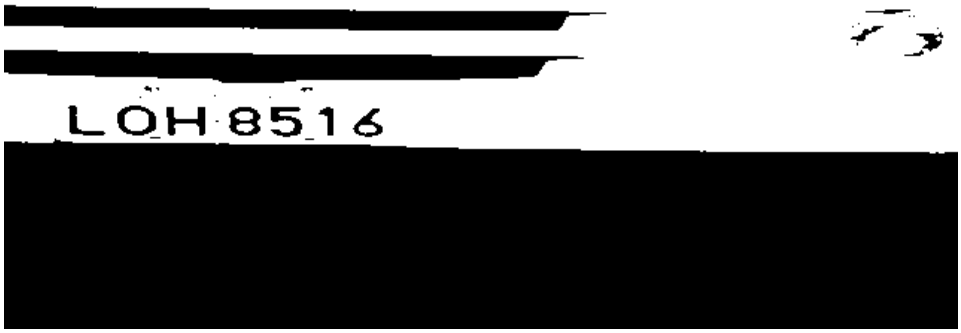
$$\omega(k) = \sum_{i=1}^{k} p_i \quad (14), \qquad \mu(k) = \sum_{i=1}^{k} i \cdot p_i \quad (15)$$

A soma total e a soma da variância dos grupos, respectivamente:

$$\mu_{\mathrm{T}} = \mu(L) = \sum_{i=1}^{L} i \cdot p_i \quad (16), \qquad \sigma_B^2 = \frac{[\,\mu_{\mathrm{T}} \cdot \omega(k) - \mu(k)]^2}{\omega(k) \cdot [1 - \omega(k)]^2} \quad (17)$$

1. Obtenha o histograma da imagem.
2. Divida cada frequência pelo número total de pixels da imagem, obtendo as probabilidades individuais das frequências (normalização do histograma).
3. Obtenha as somas parciais acumuladas ω e μ de cada intensidade.
4. Para cada intensidade k, calcule $\sigma_B^2(\mathrm{k})$. Se este for o maior valor encontrado, atualize o limiar k* com o valor de k.
5. Para cada pixel, se a intensidade do pixel for menor que ou igual a k*, ele será considerado texto; caso contrário, fundo.

Figura 23: Exemplo de binarização utilizando o método de Otsu.



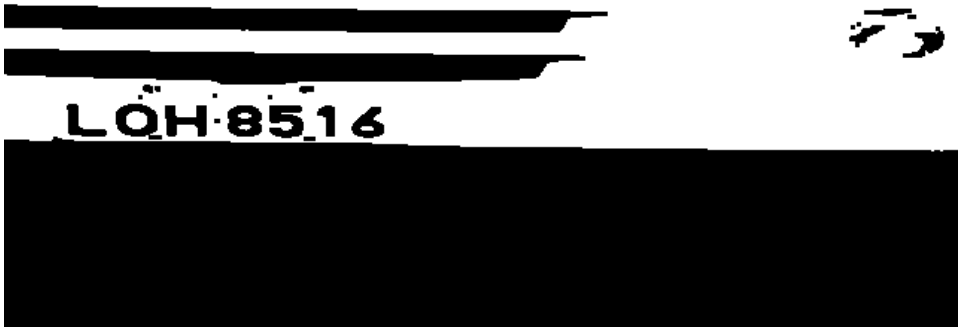
Fonte: Próprio Autor.

Conforme testes realizados, essa técnica tem se mostrado bastante eficaz e seus resultados satisfatórios quando empregado em imagens contendo iluminação uniforme e diferentes objetos em tons de cinza.

4.1.3 Dilatação

A dilatação tem por objetivo, como próprio nome diz aumentar as dimensões dos objetos na imagem. Algumas amostras apresentaram um afinamento nos traços dos caracteres e outros alguns pequenos *pixels* que surgiram do processo de binarização. A dilatação fez com que esses pequenos *pixels* desaparecessem e os traços dos caracteres sejam realçados, salientando ainda mais suas bordas.

Figura 24: Exemplo de imagem dilatada.



Fonte: Próprio Autor.

Mesmo depois da dilatação é possível notar que existem pequenas áreas de pixels de cor branca que não são de interesse para o reconhecimento de caracteres. Para tentar minimizar a quantidade dessas áreas foi realizado um preenchimento com o valor 0 (cor preto) em objetos com menos de quatro *pixels* de largura e três *pixels* de altura. É importante minimizar a ocorrência dessas áreas para que o processo de localização da placa, descrito a seguir, não seja prejudicado.

4.2 Segmentação

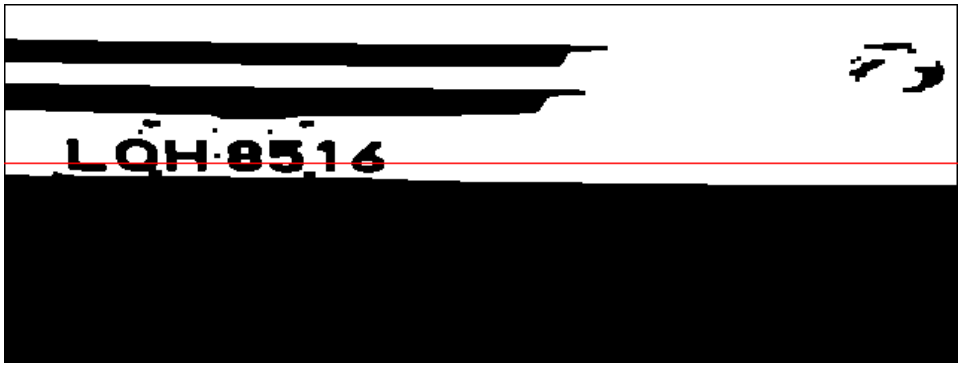
Após a realização do pré-processamento, já é possível obter uma imagem binária com seus elementos principais destacados e suas bordas suavizadas. Nesta seção serão abordadas as etapas de segmentação das regiões de interesse na imagem, descrevendo a forma utilizada para localizar a placa do veículo e extrair o conteúdo desejado, que para este trabalho são os caracteres alfanuméricos da imagem.

4.2.1 Localização da Placa

Para a realização desta etapa foi utilizado como base o algoritmo de análise de variação tonal proposto por Souza et al.(2006). Segundo ele, devido à placa de um automóvel ser constituída por um fundo com cor diferente dos caracteres forma-se uma grande diferença tonal nas bordas dos dígitos permitindo ao algoritmo de análise por variação tonal identificar estas áreas e classificá-las como uma possível área de localização da placa.

Dessa forma, o algoritmo realiza uma busca vertical na imagem para localizar a linha que possui a maior variação tonal, considerando que essa linha constitui alguma área pertencente à placa do veículo. Encontrada essa linha, já é possível ter uma noção da posição da placa.

Figura 25: Linha com maior variação tonal.

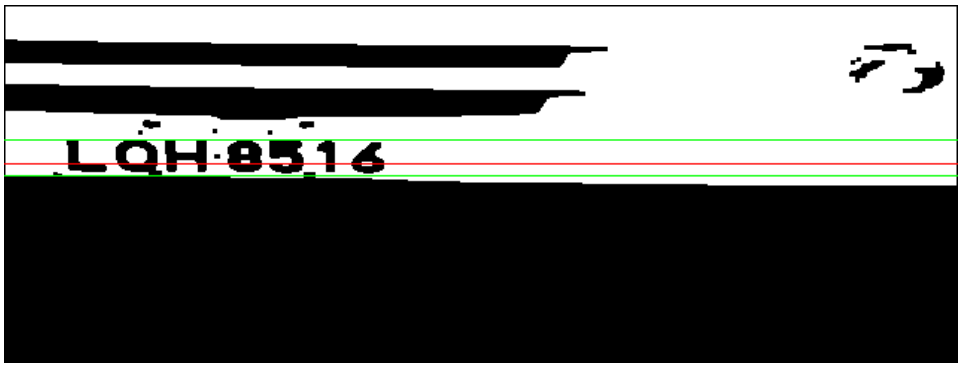


Fonte: Próprio Autor.

4.2.2 Localização e Segmentação dos Caracteres

A partir da linha de maior variação são feitas duas novas varreduras verticais, ambas começando da linha encontrada, porém a primeira varredura em direção ao topo da imagem e outra em direção a parte inferior, visando encontrar uma linha da imagem onde a variação tonal seja menor que a quantidade de variação tonal encontrada na linha central menos 14. O número 14 foi definido devido ao número de caracteres contido em uma placa (sete) sendo que para cada caractere encontrado há no mínimo duas variações tonais. Isso aumenta a garantia de que os caracteres estarão contidos na zona limitada verticalmente por essas duas linhas.

Figura 26: Linha central (vermelho) e linhas superior e inferior (verde).



Fonte: Próprio Autor.

Para um melhor ajuste da área de segmentação dos caracteres, foi calculado o valor médio entre a linha superior e a linha central e também o valor médio entre a

linha central e a linha inferior. Essas linhas servirão como referencia, pois elas passam pelos caracteres e ajudarão a definir melhor a posição de cada um destes. Uma terceira varredura é realizada, agora sobre a linha central e as novas linhas médias calculadas com a função de capturar os locais onde há variações do valor um para zero (onde a imagem muda da cor branco para cor preto), pelo fato de que cada uma dessas variações pode representar a mudança do fundo branco, que caracteriza o fundo da placa binarizada, para um objeto preto que representa um caractere. Como último passo, foi realizada uma comparação entre os valores capturados para a linha central e as linhas médias. Os valores capturados que são comuns, ou próximos, a duas linhas ou mais representam um caractere. Os valores seguintes a serem considerados devem estar em uma posição maior que a do ultimo valor definido como um caractere mais a largura mínima estabelecida.

Por meio dos testes realizados, ficaram definidas as dimensões de imagem contendo um caractere alfanumérico segmentado de 24 *pixels* de altura e 28 *pixels* de largura, pelo fato de que podemos inserir qualquer caractere das imagens contidas no conjunto de dados nessas dimensões.

Figura 27: Imagem destacando os caracteres localizados.

Fonte: Próprio Autor.

4.3 Extração de Características

Os caracteres segmentados na etapa anterior servirão para criar um conjunto de dados que será utilizado para o treinamento da rede neural artificial.

A RNA terá como estímulos de entrada o valor de cada *pixel* da imagem, por essa razão todas as imagens foram segmentadas com as mesmas dimensões (24 x 28 *pixels*), totalizando 672 entradas com valores binários.

Como o reconhecimento de um modo geral trabalha com dois grupos, letras e números, optou-se por criar duas RNAs onde ambas possuem o mesmo número de entradas e serão diferenciadas pelo número de neurônios nas demais camadas, onde o número de neurônios para a camada será definido por meio de testes, que serão mostrados mais adiante neste trabalho, enquanto a RNA que faz o reconhecimento dos números possui 10 saídas (valores de 0 a 9) e a que faz reconhecimento das letras possui 26 saídas (valores de A a Z).

4.4 Reconhecimento e Interpretação

O reconhecimento e interpretação das imagens utilizaram redes neurais artificiais do tipo *feedforward* contendo três camadas: entrada, oculta e saída. Ambas as redes utilizam como função de ativação a função tangente hiperbólica nas camadas oculta e de saída. Para o treinamento das redes neurais foi utilizado a algoritmo de treinamento *backpropagation*.

Cada uma das redes neurais utilizadas possui um conjunto de dados para seu treinamento e teste. Para a RNA que reconhece os números, foram utilizadas 40 amostras de cada algarismo numérico de zero a nove, totalizando 400 imagens neste conjunto de dados e para formar o conjunto de dados para o reconhecimento de letras foram utilizados 470 imagens, distribuídas conforme a tabela 1:

Tabela 1: Distribuição das amostras utilizadas no conjunto de dados das letras.

A	20	N	21
B	21	O	19
C	20	P	18
D	17	Q	16
E	20	R	20
F	20	S	16
G	17	T	19
H	18	U	15
I	19	V	17
J	18	W	11
K	20	X	16
L	21	Y	15
M	19	Z	17

A definição do melhor modelo a ser empregado se deu por meio de testes que consistiam em encontrar o número de neurônios ideal da camada oculta. Como o número de neurônios da camada de entrada e saída foram definidas na estrutura do problema, determinando as dimensões da imagem que será segmentada e nas possíveis respostas do reconhecimento dos caracteres alfanuméricos, respectivamente, os modelos de RNA se diferenciaram pelo número de neurônios da camada oculta. Diversos modelos com valores de 5 a 100 neurônios, variando ao passo de 5, foram analisados, com as amostras de ambos os conjuntos divididos da seguinte forma:

Tabela 2: Divisão dos conjuntos de dados (letras e números).

Subconjunto	% do Conjunto de Dados
Estimação	70
Validação	15
Teste	15

O conjunto de estimação foi utilizado no treinamento da rede e o conjunto de validação teve o propósito de validar o treinamento. A apresentação de todos os elementos do conjunto de estimação a rede chamamos de época. Para cada época apresentada à rede, os pesos sinápticos são ajustados propagando os valores da saída até as primeiras camadas da rede. Após o ajuste, a rede opera de modo *forward*, ou seja, da entrada até a saída sem realizar ajuste nos pesos, utilizando o subconjunto de validação e calculando o erro global médio da rede. Consideramos

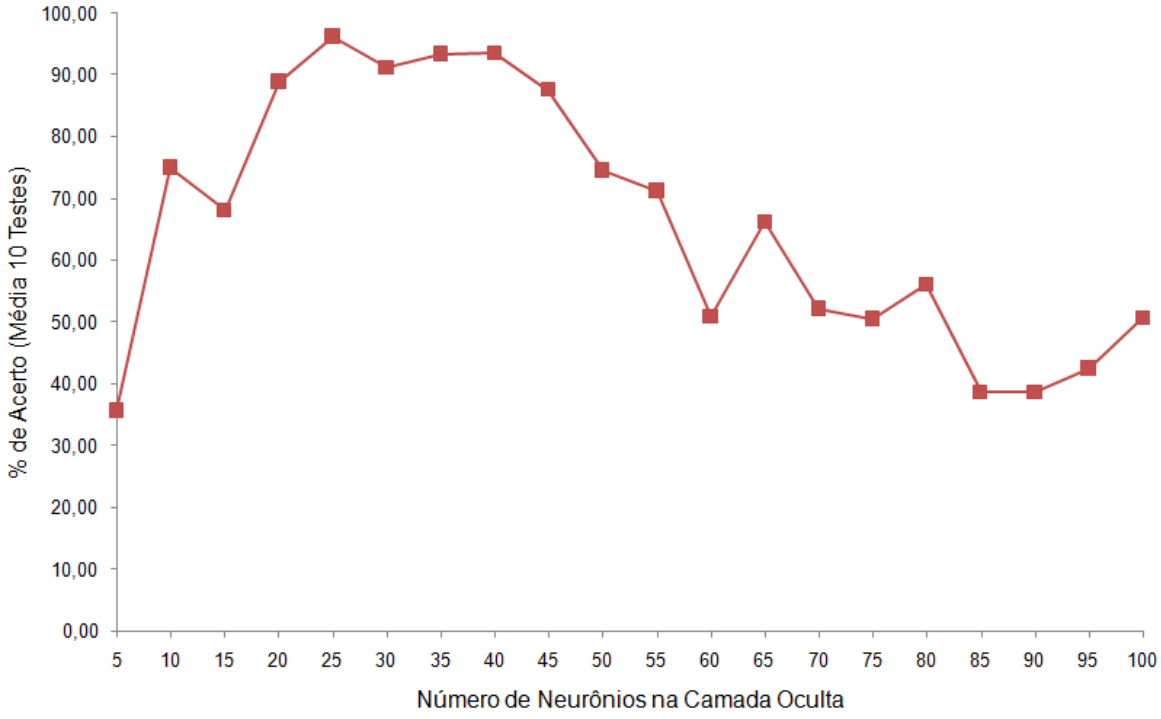
um acerto quando o valor da saída da rede é equivalente ao valor apresentado previamente como saída ideal, para o estímulo na entrada da rede.

Conforme as épocas são apresentadas, o erro global médio para o conjunto de treinamento decresce continuamente enquanto que o erro para o conjunto de validação decresce até um valor mínimo antes começar a crescer, esse ponto de erro mínimo é considerado o ponto ideal de parada do treinamento, essa técnica é conhecida como validação cruzada com parada antecipada.

5 RESULTADOS

Os resultados apresentados nesta seção foram obtidos conforme os acertos gerados na saída da rede utilizando como estímulos para as redes neurais as amostras do subconjunto de teste. Estes valores estão contidos nos gráficos das figuras 28 e 29, esses resultados levaram em conta a média de 10 testes realizados com cada modelo.

Figura 28: Resultado dos Testes da Rede Neural para Números.



Fonte: Próprio Autor.

Para a rede neural que faz o reconhecimento dos números, uma alta média de acertos pode ser obtida com 20 a 40 neurônios na camada oculta, contudo para o problema proposto, o valor ideal calculado é de 25 neurônios, apresentando uma média de acertos de mais de 96%.

Com o valor de neurônios da camada oculta definido para rede neural dos números, um novo teste foi apresentado buscando visualizar o desempenho do modelo. Novamente todo o processo já descrito, desde o ajuste dos pesos com o

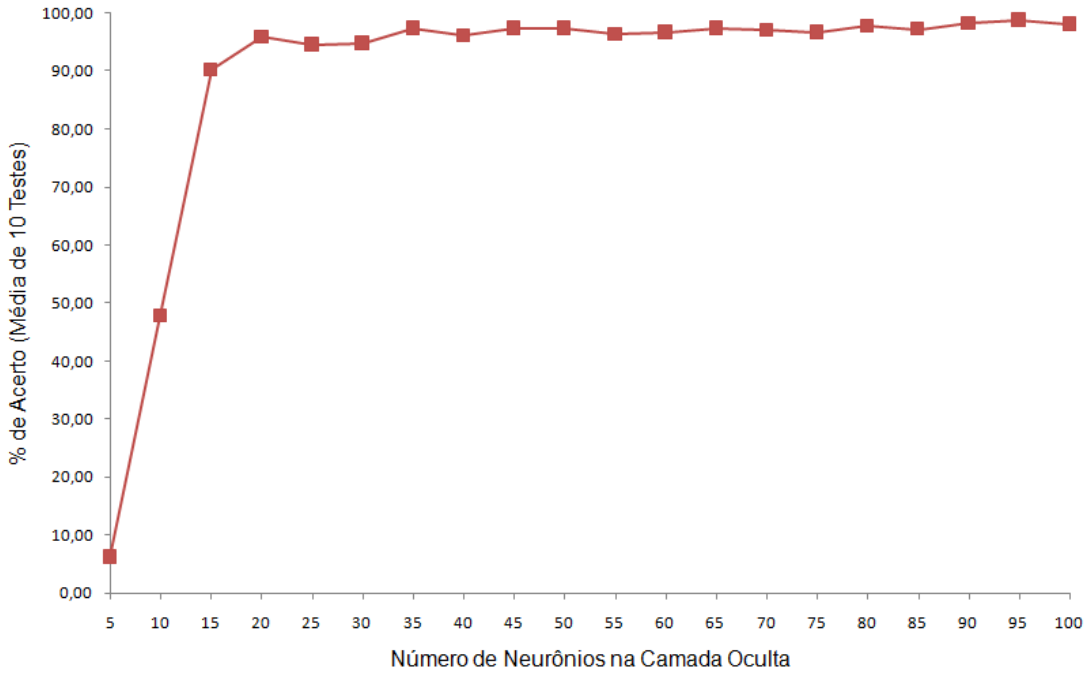
conjunto de estimação, passando pela validação do modelo de RNA, até a contagem de acertos sobre o conjunto de testes. Este processo foi repetido 10 vezes e as médias dos resultados deste teste estão destacadas em uma tabela chamada de matriz de confusão. As colunas da matriz representam os resultados previstos enquanto as linhas mostram as saídas reais da rede. Este tipo de tabela é muito utilizado para visualizar o desempenho do algoritmo e verificar se ele está distinguindo corretamente as classes.

Tabela 3: Matriz de confusão para o modelo ideal de reconhecimento dos números.

	0	1	2	3	4	5	6	7	8	9
0	4,9	0	0	0,3	0	0,1	0	0	1,3	0
1	0	5,2	0	0	0	0	0	0	0	0
2	0	0	5,9	0,1	0	0	0	0,4	0	0
3	0	0	0	5,2	0	0	0	0	0	0
4	0	0	0	0	5,2	0	0	0	0	0
5	0	0	0	0	0	5,2	0	0	0	0
6	0	0	0	0	0,5	0	7,6	0	0	0
7	0	0	0,7	0	0	0	0	6,4	0	0
8	0,4	0	0,1	0	0	0	0	0	4,4	0,2
9	0	0	0,1	0,3	0	0	0	0,1	0,3	5,1

Analisando os resultados obtidos para a RNA que reconhece as letras, observa-se uma média de acertos alta a partir de 15 neurônios, conforme o gráfico da figura 29:

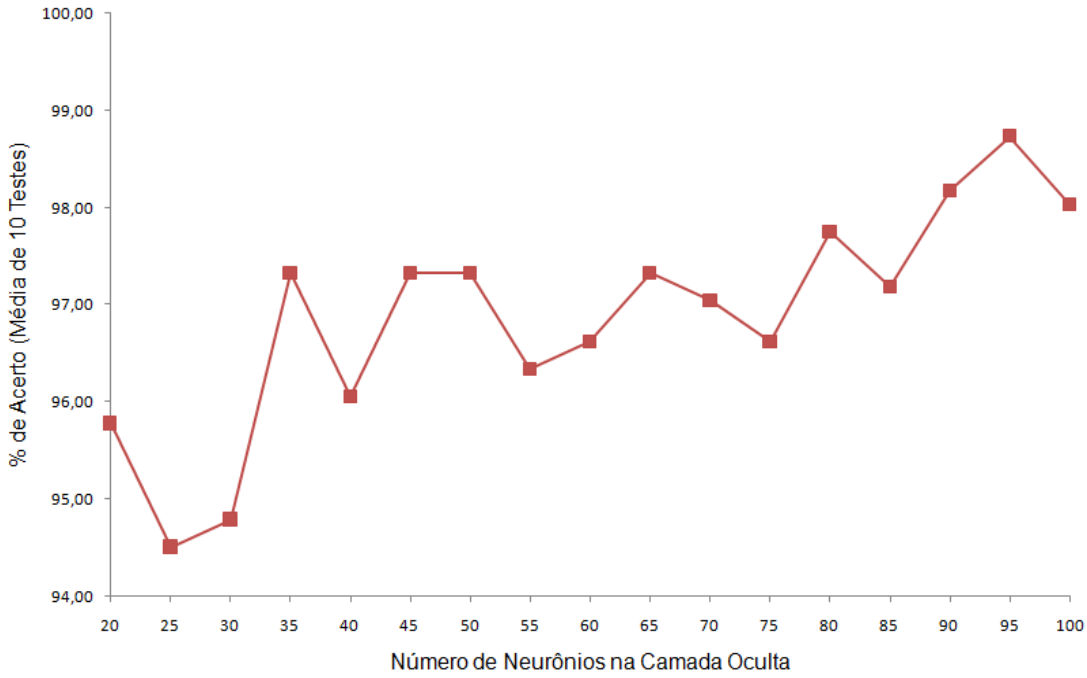
Figura 29: Resultado dos Testes da Rede Neural para Letras.



Fonte: Próprio Autor.

Como os resultados possuem valores muito próximos, o gráfico da figura 30 apresentará os mesmo valores com maiores detalhes:

Figura 30: Resultados dos Testes mais Detalhados para a RNA das Letras.



Fonte: Próprio Autor.

Com 95 neurônios na camada oculta, a rede neural de letras obteve um desempenho de generalização de aproximadamente 99%. Para o teste de visualização do desempenho do modelo ideal calculado para o reconhecimento das letras, foram coletados os resultados para a matriz de confusão da tabela 4.

Tabela 4: Matriz de confusão para o modelo ideal de reconhecimento das letras.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	2,9	0	0	0	0	0	0	0	0	0	0	0	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	2,5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	3,8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	2,5	0	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	2,1	0,2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	2,2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	3,0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	2,8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	2,5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	2,7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	2,8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	3,8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0,1	0	0	0	0	3,5	0	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	4,0	0	0	0	0,1	0	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3,3	0	0,6	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3,1	0	0	0	0	0	0	0	0	0	0
Q	0	0	0	0,2	0	0	0	0	0	0	0	0	0	0	0,2	0	2,0	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,1	0	2,6	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2,1	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0,1	0	0	0	0	0	0	0	0	0	0	3,5	0	0	0	0	0	0
U	0	0	0	0	0	0	0	0	0	0,2	0	0	0	0	0	0	0	0	0	0	1,2	0	0	0	0	0
V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2,6	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1,2	0	0	0
X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2,5	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1,6	0
Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2,2

6 CONCLUSÕES

Ao final deste trabalho conclui-se que o problema de reconhecimento de caracteres contidos em placas de automóveis pode ser resolvido empregando processamento digital de imagens e redes neurais, para a manipulação da imagem e o reconhecimento, respectivamente. Contudo, outros pontos também devem ser levados em consideração.

A binarização da imagem, aplicando o método de Otsu, foi eficaz em imagens apresentando veículos particulares (fundo da placa de cor cinza e caracteres de cor preta), porém para veículos de aluguel (táxis, ônibus e vans, fundo da placa de cor vermelha e caracteres de cor branca) foi ineficaz, pois o método parte do principio que o fundo da imagem, neste caso o fundo da placa, terá a cor branca, tornando impossível de distinguir o código da placa.

Algumas imagens das placas disponibilizadas possuíam caracteres de diferentes fontes e escalas e algumas com inclinação devido ao ângulo em que foram registradas. Porém, as redes neurais foram invariantes a essas deformações.

Por último, a definição dos parâmetros de uma rede neural e o tamanho do conjunto de dados têm importância significativa no seu tempo de treinamento, porém, nos resultados não houve diferença significativa.

REFERÊNCIAS

ARAUJO, L. M., OLIVEIRA, F. L. Estudo dos Descritores de Características Primitivas. In: ENCONTRO DE COMPUTAÇÃO E INFORMÁTICA DO TOCANTINS, 2011, Palmas. p. 169-178. Disponível em http://ulbra-to.br/ encoinfo/artigos/2011/Estudo_dos_descritores_de_caracteristicas_primitivas.pdf

BRAGA, A. de P.; LUDEMIR, T. B.; CARVALHO, A. C. P. de L. F. Redes Neurais Artificiais: Teoria e Aplicação. Rio de Janeiro: LTC, 2000.

CAMPOS, T. d. J. Reconhecimento de Caracteres Alfanuméricos de Placas em Imagens de Veículos. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, 2001.

CASTRO, Leandro N. de; ZUBEN, Fernando J. Von. Redes Neurais Artificiais. Campinas: DCA/FEEC/Unicamp, [20--]. 96 slides: color. Acompanha texto.

CONCI, A.; MONTEIRO, L. H. Reconhecimento de Placas de Veículos por imagem. Rio de Janeiro, 2004.

ERPEN, L. R. C. Reconhecimento de Padrões em Imagens por Descritores de Forma. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, 2004.

FILHO, O. M.; NETO, H. V. Processamento Digital de Imagens. Rio de Janeiro: Brasport, 1999.

FLORES, E. S., Processamento Digital de Imagens na Identificação de Variedades de Soja. Monografia – Universidade Federal do Pampa, Bagé, 2012.

GONZALEZ, R. C.; WOODS, R. E. Digital Image Processing. 3. ed. Upper Saddle River, NJ USA: Prentice-Hall, Inc., 2006. ISBN 013168728X.

HAYKIN, S. Redes Neurais: Princípios e prática. 2ª edição. Editora Bookman, Porto Alegre 2001.

IKEDA, D. T., Seleção Não-Supervisionada de Métodos de Binarização para Documentos Históricos. Monografia - Universidade de São Paulo, São Paulo, 2011.

KARRER, D.; CAMEIRA, R. F.; VASQUES, A. S.; BENZECRY, M. de A. Redes Neurais Artificiais: Conceitos e Aplicações. In: PROFUNDÃO – IX ENCONTRO DE ENGENHARIA DE PRODUÇÃO DA UFRJ. Rio de Janeiro, 2005.

LAGANIÈRE, R. OpenCV 2 Computer Vision Application Programming Cookbook. [S.l.]: Packt Publishing, 2011. ISBN 1849513244.

LIBERMAN, F. Classificação de Imagens Digitais por Textura Usando Redes Neurais. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, 1997.

Manual de Referência do Fast Artificial Neural Network Library. Disponível em: < http://leenissen.dk/fann/wp/help/>. Acessado em: 02 de ago. 2013.

MILANO, D. de; HONORATO, L. B. Visão Computacional. São Paulo, 2010.

MONTABONE, S. Beginning Digital Image Processing: Using Free Tools for Photographers. 1st. ed. Berkely, CA, USA: Apress, 2010.

NASCIMENTO, M. C. Detecção de objetos em imagens. Recife, janeiro de 2007.

REHEM A., TRINDADE F. H. V. Técnicas de Visão Computacional para Rastreamento de Olhar em Vídeos. Publicado em 03/02/2009. Disponível em: <http://almerindo.devin.com.br/index.php?

option=com_content&view=article&id=78%3Atecnicas-de-computacao-visual-pararastreamento-de-olharemvideos&catid=43%3Atrabalhos-dealunos&Itemid=86&showall=1> . Acesso em: 04 de out. 2011.

SANTOS, A. P. de O. Desenvolvimento de Descritores de Imagem para Reconhecimento de Padrões de Plantas Invasoras (Folhas Largas e Folhas Estreitas). Dissertação (Mestrado) — Universidade Federal de São Carlos, 2009.

SHAPIRO, L. G.; STOCKMAN, G. Computer Vision. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN 0130307963.

SILVA, J. de A., GONÇALVES, W. N., MACHADO, B. B., Pistori H., SOUZA, A. S. de, Comparação de Descritores de Formas no Reconhecimento de Objetos. Universidade Católica Dom Bosco. Campo Grande – MS, 2007.

SILVA, J. R. P., Representação de Objetos Através de Descritores de Contorno e Classificação por Funções Discriminantes Lineares por Partes. Dissertação (Mestrado) —Universidade Estadual de Campinas,1996.

SILVA, L. N. de C. Análise e Síntese de Estratégias de Aprendizado Para Redes Neurais Artificiais. Dissertação (Mestrado) — Universidade Estadual de Campinas, Setembro de 1998.

SILVA, R. R. da. Reconhecimento de Imagens Digitais Utilizando Redes Neurais Artificiais. Dissertação (Mestrado) — Universidade Federal de Lavras, 2005.

SOUZA, C. A. de Q. et al. Reconhecimento de Placas de Automóveis Através de Câmeras IP. São Paulo, 2006.

TRINDADE, F. Técnicas de Visão Computacional para Rastreamento de Olhar em Vídeos. 2009.

APÊNDICE A – CÓDIGOS-FONTE

```
/*****************************************************************************************************************************
***
***                                          Codigo main.cpp
***        codigo de inicializacao do programa, chamada das bibliotecas até salvar os caracteres para o
***        reconhecimento
***
*****************************************************************************************************************************/

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/features2d/features2d.hpp>

#include <iostream>
#include "Histogram1D.hpp"
#include "Find_plate.hpp"

// 0 bit preto; 255 bit branco

void area_objetos(cv::Mat &image);

int main(int argc,char *argv[]) {

        cv::Mat original_image,image,eroded;

        if(argc==2){ //nome da imagem passada por parametro
                original_image= cv::imread(argv[1]); // carrega a imagem

                image= cv::imread(argv[1],
                CV_LOAD_IMAGE_GRAYSCALE); // carrega a imagem transformando em uma matriz de nivel
                                                        // de cinza
        }else{
                std::cout<<"IMAGEM DE ENTRADA NAO DECLARADA\n
                        O PROGRAMA SERA ENCERRADO"<<std::endl;
                return -1;
        }

        if(!image.data ) {
                std::cout<<"O PROGRAMA SERA ENCERRADO"<<std::endl;
                return -1;
        }

        cv::Mat result;
        cv::Mat result2;
        Histogram1D h;

        cv::GaussianBlur(image,result,cv::Size(3,3),1.0); //aplicacao de filtro gaussiano 3x3

        int limiar = h.getThreshold(image);
        std::cout<<limiar<<std::endl;

        //a imagem e binarizada utilizando o valor da funcao "getThreshold" como limiar

        cv::threshold(result, result2,limiar, 255, cv::THRESH_BINARY);

        cv::erode(result2,eroded,cv::Mat());

        result2=eroded;

        Find_plate fp(result2);

        fp.area_objetos(result2);
```

```
	fp.getPlate(original_image); // segmenta e salva os caracteres em um diretorio

	return 0;
}



/*************************************************************************************************************************
***
***                                     Codigo Find_plate.hpp
***	Busca a linha com maior variacao tonal, dimensoes da placa e caracteres, redimensionamento da area
***	de segmentacao e salva as imagens para serem reconhecidas pela rede.
***
*************************************************************************************************************************/


#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <stdlib.h>
#include <iostream>
#include <sstream>

// MAIOR CARACTER ALT: 24 LARG: 28

#define LARGURA 28
//os numeros possuem uma largura diferente
//por isso, a busca por eles exige uma largura diferente

#define ALTURA 24

class Find_plate{
	private:
		cv::Mat imgcp;
		int main_line;
		int botton_line;
		int upper_line;
		int left_line;
		int right_line;


		std::vector<int> upper_changes;
		std::vector<int> botton_changes;
		std::vector<int> chars_main_line;

		int repets,repets_bottom,repets_upper;
		std::vector<int> largura_chars;
		std::vector<int> dim_carac;
		int count_img;

	public:
		Find_plate(const cv::Mat &image){
			imgcp=image.clone(); // manipula feita com uma copia da imagem
			count_img=0;
		}

		void getMainLine(){
			int var_num=0;
			int previous_var_num=0;
			int cur_lin=0;
			uchar fb;
			std::vector<int> changes;

			for(int i=0;i<=imgcp.rows;i++){		// varredura vertical, da primeira ate o numero total
								// de linhas
				fb=imgcp.at<uchar>(i,0);	// 1.o bit da linha, nesse bit contem a cor do pixel
								// para comparar com os demais bits da linha
```

```
                var_num =0;
                for(int j=1;j<imgcp.cols-1;j++){
                        if(imgcp.at<uchar>(i,j)!=fb){
                        // cada vez que o pixel for diferente do bit em fb
                                fb=imgcp.at<uchar>(i,j);
                                //e apontado uma variacao e o fb tem o valor alterado
                                var_num++;
                                if(fb == 0)
                                        changes.push_back(j);
                        }
                }

                if(var_num>previous_var_num){
                // cada vez que uma linha apresenta maior variacao o numero da linha e
                // salva como candidata a linha com maior variacao

                        cur_lin = i;
                        chars_main_line=changes;
                        previous_var_num = var_num;
                }
                changes.clear();
        }
        std::cout <<cur_lin<<std::endl;
        main_line= cur_lin;
        repets = previous_var_num;
}


void getBottonLine(){
        uchar fb;
        int var_num;
/* para encontrar a margem inferior, o principio e o mesmo,
a diferenca e que agora e procurada uma linha sem variacao de tom*/

        for(int i=main_line+1;i<=imgcp.rows;i++){
/* a busca e vertical, comecando da linha encontrada em "getMainLine"ate o numero de linha
da imagem */
                fb=imgcp.at<uchar>(i,0);
                var_num =0;
                for(int j=1;j<imgcp.cols;j++){//(row,col)
                        if(imgcp.at<uchar>(i,j)!=fb){
                                fb=imgcp.at<uchar>(i,j);
                                var_num++;
                        }
                }
                if(abs(var_num - repets)>14){
                        repets_bottom=var_num;
                        botton_line = i;
                        break;
                }
        }

}


void getUpperLine(){
/* A "getBottonLine" e a "getUpperLine" funcionam da mesma forma,
buscando uma linha que nao tenha variacao porem nesta funcao a busca
e a partir da linha encontrada por "getMainLine" até o topo da imagem*/

        uchar fb;
        int var_num;

        for(int i=main_line-1;i>=0;i--){
                fb=imgcp.at<uchar>(i,0);
                var_num =0;
                for(int j=1;j<imgcp.cols;j++){
```

```
                        if(imgcp.at<uchar>(i,j)!=fb){
                                fb=imgcp.at<uchar>(i,j);
                                var_num++;
                        }
                }

                if(abs(var_num - repets)>14){
                        repets_upper=var_num;
                        upper_line = i;
                        break;
                }
        }
}

static int menor(int a, int b){
        if(a<b)    return a;
        else return b;
}

void Resize_segments(){

/* De posse das margens de cada elemento essa funcao serve apenas para adequa-los as
dimensoes corretas para o reconhecimento pela rede */

        int size_chars = botton_line - upper_line;
        int res;

        // redimensiona a altura

        if(size_chars>24){
                res=size_chars - 24;
                if(res%2 != 0){
                        upper_line+= int(res/2);
                        botton_line-=int(res/2)-1;
                }else{
                        upper_line+= int(res/2);
                        botton_line-=int(res/2);
                }

        }else if(size_chars<24){
                res=24 - size_chars;
                if(res%2 != 0){
                        upper_line-= int(res/2);
                        botton_line+=int(res/2)+1;
                }else{
                        upper_line-= int(res/2);
                        botton_line+=int(res/2);
                }

        }

        // redimensiona a largura

        int width_char,lwidth, rwidth;

        for (std::vector<int>::iterator it = largura_chars.begin() ; it != largura_chars.end();
        it=it+2){

                lwidth=*it;
                rwidth=*(it+1);

                width_char = rwidth - lwidth;

                if(width_char>28){
                        res=width_char - 28;
                        if(res%2 != 0){
                                lwidth+= int(res/2);
```

```
                                        rwidth-=int(res/2)-1;
                                }else{
                                        lwidth+= int(res/2);
                                        rwidth-=int(res/2);
                                }

                        }else if(size_chars<28){
                                res=28 - width_char;
                                if(res%2 != 0){
                                        lwidth-= int(res/2);
                                        rwidth+=int(res/2)+1;
                                }else{
                                        lwidth-= int(res/2);
                                        rwidth+=int(res/2);
                                }

                        }

/* cada par salvo no vetor dim_carac respresenta as coordenadas do ponto superior e
esquerdo,respectivamente, de cada caracter */

                        dim_carac.push_back(upper_line);
                        dim_carac.push_back(lwidth);
                }

}

void area_objetos(cv::Mat &image){  //elimina areas de pixels brancos
        uchar t, pixref;

        std::vector<std::vector<int> > alturas;
        std::vector<std::vector<int> > larguras;

        std::vector<int> aux;
        std::vector<int> alguma_largura;
        std::vector<int> alguma_altura;

        //ajuste de alturas

        for(int i=0;i<image.cols;i++){

                image.at<uchar>(0,i)=0;

                //marcadas como escura para obter ponto de referencia
                image.at<uchar>(image.rows-1,i)=0;
                pixref=255;

                for(int j=0;j<image.rows;j++){//largura
                        if(image.at<uchar>(j,i)==255 && pixref==255){
                                aux.push_back(j);
                                pixref=0;
                        }else if(i==image.cols-1 || (image.at<uchar>(j,i)==0 && pixref==0)){
                                aux.push_back(j);
                                pixref=255;
                        }
                }
                alturas.push_back(aux);
                aux.clear();
        }

        for (int j=0;j<image.cols-1;j++){
                alguma_altura=alturas[j];
                for (int i=1;i<alguma_altura.size();i=i+2){

                // preenche as areas com alturas menor que 3;
```

```
                                        if((alguma_altura[i]-alguma_altura[i-1])<3){
                                                cv::line(image,cv::Point(j,alguma_altura[i-
                                        1]),cv::Point(j,alguma_altura[i]-1),cv::Scalar(0));
                                        }
                                }
                        }

                        //ajuste de larguras

                        for(int i=0;i<image.rows-1;i++){

                                image.at<uchar>(i,0)=0;

                                //marcadas como escura para obter ponto de referencia
                                image.at<uchar>(i,image.cols-1)=0 ;
                                pixref=255;

                                for(int j=0;j<image.cols;j++){//largura
                                        if(image.at<uchar>(i,j)==255 && pixref==255){
                                                aux.push_back(j);
                                                pixref=0;
                                        }else if(i==image.rows-1 || (image.at<uchar>(i,j)==0 && pixref==0)){
                                                aux.push_back(j);
                                                pixref=255;
                                        }
                                }
                                larguras.push_back(aux);
                                aux.clear();
                        }

                        for (int j=0;j<image.rows-1;j++){
                                alguma_largura=larguras[j];
                                for (int i=1;i<alguma_largura.size();i=i+2){
                                        // preenche as areas com largura menor que 5;
                                        if((alguma_largura[i]-alguma_largura[i-1])<4){
                                cv::line(image,cv::Point(alguma_largura[i-1],j),cv::Point(alguma_largura[i]-
1,j),cv::Scalar(0));
                                        }
                                }
                        }

                }

                std::string IntToString ( )
                {
                        ++count_img;
                        std::ostringstream oss;
                         oss<< count_img;
                         return oss.str();
                }

                void getPlate(cv::Mat &original){

                        cv::Mat newimage(24,28, CV_8UC3,cv::Scalar(255,255,255)); // nova imagem
                        std::string nameimg;

                        // busca todas as margens da placa e...
                        getMainLine();
                        getBottonLine();
                        getUpperLine();

                        Resize_segments();

                        for (std::vector<int>::iterator it = dim_carac.begin() ; it !=  dim_carac.end(); it=it+2){
                                cout<<"largura: "<<*(it+1)<< "  ate  "<< *(it+1)+LARGURA<<"
("<<*(it+1)+LARGURA - *(it+1)<<" )"<<endl;
```

```
                                        cout<<"largura: "<<*it<< "  ate  "<< *it+ALTURA<<" ("<<(*it+ALTURA) - *it<<"
)"<<endl;

                                        for(int i = *(it+1); i< *(it+1)+LARGURA; i++){
                                                for(int j = *it;j < (*it+ALTURA) ;j++){

                                                newimage.at<cv::Vec3b>(j-*it,i-
*(it+1))[0]=original.at<cv::Vec3b>(j,i)[0];
                                                newimage.at<cv::Vec3b>(j-*it,i-
*(it+1))[1]=original.at<cv::Vec3b>(j,i)[1];
                                                newimage.at<cv::Vec3b>(j-*it,i-
*(it+1))[2]=original.at<cv::Vec3b>(j,i)[2];
                                                }
                                        }
                                        nameimg="chars/"+IntToString()+".png";
                                        cv::imwrite(nameimg, newimage);

                                }
                        }
};


/*************************************************************************************************************************************
***
***                                     Codigo Histogram1D.hpp
***                     criacao, manipulacao de histograma e calculo do threshold
***
*************************************************************************************************************************************/


#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/features2d/features2d.hpp>
#include <cstdlib>
#include<math.h>

class Histogram1D {
        private:
                int histSize[1]; // numero de niveis
                float hranges[2]; // min e max valores de pixel
                const float* ranges[1];
                int channels[1];
                long int num_pix;
                float prob_intens[256];//armazena as probabilidades de cada intensidade
                float sum_u[256];
                float sum_omega[256];

        public:
                Histogram1D() {
                        // parametros default do histograma
                        histSize[0]= 256;
                        hranges[0]= 0.0;
                        hranges[1]= 255.0;
                        ranges[0]= hranges;
                        channels[0]= 0; // por ser uma imagem monocromatica, e usado 1 canal
                }

                cv::MatND getHistogram(const cv::Mat &image) {
                        cv::MatND hist;
                        // calculo do histograma
                        cv::calcHist(&image,
                        1,// histograma de 1 imagem
                        channels, // canais usados
                        cv::Mat(), // mascara usada para o histograma
                        hist,// histograma resultante
                        1,
```

```
		histSize, // numero de niveis de cinza
		ranges// variacao do valor dos pixels
		);
		return hist;
	}

	// calculo e imagem do histograma calculado
	cv::Mat getHistogramImage(const cv::Mat &image){
		cv::MatND hist= getHistogram(image);
		// busca o maior e menor valor de nivel de cinza
		double maxVal=0;
		double minVal=0;
		cv::minMaxLoc(hist, &minVal, &maxVal, 0, 0);
		// imagem do histograma
		cv::Mat histImg(histSize[0], histSize[0],CV_8U,cv::Scalar(255));
		int hpt = static_cast<int>(0.9*histSize[0]);

		// desenha um linha entre dois pontos
		for( int h = 0; h < histSize[0]; h++ ) {
			float binVal = hist.at<float>(h);
			int intensity = static_cast<int>(binVal*hpt/maxVal);
			cv::line(histImg,cv::Point(h,histSize[0]),
			cv::Point(h,histSize[0]-intensity),
			cv::Scalar::all(0));
		}
		return histImg;
	}


	int getThreshold(const cv::Mat &image){

		cv::MatND hist= getHistogram(image);

		num_pix=image.cols*image.rows;

		int index; // threshold
		float sum_uT;
		float sigma_quad=0;
		float thres=0;

		sum_u[0]=0;
		prob_intens[0]=hist.at<float>(0)/num_pix;
		sum_omega[0]=prob_intens[0];

		for(int i=1; i<256;i++){
			//probabilidade de cada intensidade ocorrer na imagem
			prob_intens[i]=hist.at<float>(i)/num_pix;

			//soma parcial u
			sum_u[i]= (prob_intens[i]*i)+sum_u[i-1];

			//somatorio das probabilidades
			sum_omega[i]=prob_intens[i]+sum_omega[i-1];
		}

		sum_uT = sum_u[255];

		for(int i=1; i<256;i++){

			sigma_quad=std::pow((sum_uT*sum_omega[i]-
sum_u[i]),2)/(sum_omega[i]*(1-sum_omega[i]));

			if(sigma_quad>=thres){
				thres=sigma_quad;
				index=i;
			}
		}
```

```
                return index;
            }

};

/*****************************************************************************************************************************
***                                 Codigo rede_neural.hpp
***                             codigo de treinamento e teste das redes neurais
***
***          OBS.: A MESMA BIBLIOTECA FOI UTILIZADA PARA AMBAS
***         AS REDES O QUE AS DIFERENCIA E O TAMANHO DO CONJUNTO DE
***         DADOS, E PROPORCIONALMENTE O TAMANHO DOS SUBCONJUNTOS
***         DE ESTIMACAO, VALIDACAO E TESTE
*****************************************************************************************************************************/

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include "/usr/local/include/fann.h"

#include <iostream>
#include <vector>
#include <cmath>
#include <cstdlib>
#include <string>
#include <fstream>

using namespace std;

string conj_dados[400]={...};  //conjunto de dados com numeros

class Rede_neural{
    private:
        int nos_entrada; //construtor, quantidade de nos de entrada -- input
        int nos_ocultos;
        int nos_saida;
        int imagens_por_epoca;
        fann_type entrada[672];
        fann_type saida_desejada[10];

        vector<int> estimacao, validacao, teste;

    public:
        Rede_neural(int ne, int no, int ns){
            nos_entrada = ne;
            nos_ocultos = no;
            nos_saida = ns;
        }

        int index(char l){
            switch(l){
                case '0': return 0; break;
                case '1': return 1; break;
                case '2': return 2; break;
                case '3': return 3; break;
                case '4': return 4; break;
                case '5': return 5; break;
                case '6': return 6; break;
                case '7': return 7; break;
                case '8': return 8; break;
                case '9': return 9; break;
            }

        }
```

```
void Carrega_dados_treino(string quadro){

	cv::Mat image;
	std::vector<cv::Mat> frames;
	ofstream escreve;

	//leitura da imagem e armazenamento na matriz
	image = cv::imread(quadro);

	//por se tratar de uma imagem binaria, sera processado apenas um camada

	cv::split(image,frames);
	image=frames[0]; //agora a imagem tem apenas uma camada

	escreve.open("treino.data",ofstream::app);

	for(int i=0;i<24;i++){
		for(int j=0;j<28;j++){
			if(image.at<uchar>(i,j)%254 == 1)
				escreve <<"1 ";
			else
				escreve <<"-1 ";
		}
	}
	escreve <<"\n";

	//escreve as saidas
	for (int cont=0; cont < nos_saida; cont++){
		if(index(quadro.at(0)) == cont)
			escreve<<"1 ";
		else
			escreve<<"-1 ";
	}
	escreve <<"\n";

	escreve.close();

}


bool Contido_conjunto(int num){
	for (int i=0; i<estimacao.size(); i++){
		if (estimacao.at(i) == num) return true;
	}

	for (int i=0; i<validacao.size(); i++){
		if (validacao.at(i) == num) return true;
	}

	for (int i=0; i<teste.size(); i++){
		if (teste.at(i) == num) return true;
	}
	return false;
}

void Cria_arquivo_treino(void){

	ofstream escreve;
	int indice;

	escreve.open("treino.data");//cria o arquivo permitindo a escrita

	//parametros de configuracao do arquivo de treinamento

	escreve<<"280"<<" "<<nos_entrada<<" "<<nos_saida<<"\n"; //para conj de 400

	escreve.close();
```

```
	srand (time(NULL));

	//70% de 686 = 280 para o conjunto de estimacao

	for(int h = 0; h < 280; h++){

		do{
			indice=rand()%400;
		}while(Contido_conjunto(indice));

		estimacao.push_back(indice);

		Carrega_dados_treino(conj_dados[indice]);
	}
}


void Carrega_teste(string quadro){

	cv::Mat image;
	std::vector<cv::Mat> frames;

	//leitura da imagem e armazenamento na matriz
	image = cv::imread(quadro);

	//por se tratar de uma imagem binaria, sera processado apenas um camada
	cv::split(image,frames);
	image=frames[0]; //agora a imagem tem apenas uma camada

	for(int i=0;i<24;i++){
		for(int j=0;j<28;j++){
			if(image.at<uchar>(i,j)%254 == 1)
				entrada[(i*28)+j] = 1;
			else
				entrada[(i*28)+j] = -1;
		}
	}

	for (int cont=0; cont < nos_saida; cont++){
		if(index(quadro.at(0)) == cont)
			saida_desejada[cont] = 1;
		else
			saida_desejada[cont] =-1;
	}
}

void treino_validacaocruzada(){
	struct fann *ann;
	struct fann *ann_ant;
	struct fann_train_data *estima;
	fann_type *calc_out;
	int indice;
	float acum=0;

	int conjunto_validacao[60]; // para conj de 400

	float erro_ant=100;
	float erro_treinamento=1;
	float erro_validacao = 45;

	//cria a rede: (numero de camadas, nos entradas, nos ocultos, nos saida)
	ann = fann_create_standard(3,nos_entrada,nos_ocultos,nos_saida);

	/*define com funcao de ativacao para ambas as camadas oculta e saida, a funcao de
	ativação*/
	fann_set_activation_function_hidden(ann, FANN_SIGMOID_SYMMETRIC);
```

```
	fann_set_activation_function_output(ann, FANN_SIGMOID_SYMMETRIC);
	fann_set_training_algorithm(ann,FANN_TRAIN_BATCH);
	fann_set_learning_rate(ann,0.5);
	fann_set_learning_momentum(ann,0.3);

	//estrutura para treino da rede
	Cria_arquivo_treino();

	srand (time(NULL));

	//400 * 15% = 60 para o conjunto de validacao
	for(int h = 0; h < 60;h++){
		do{
			indice=rand()%400;
		}while(Contido_conjunto(indice));

		validacao.push_back(indice);
		conjunto_validacao[h] = indice;

	}

	int cont_epoca = 0;
	estima = fann_read_train_from_file("treino.data");

	while(erro_validacao <= erro_ant && erro_treinamento > 0.00001){
		erro_ant = erro_validacao;
		erro_treinamento = fann_train_epoch(ann, estima);
		erro_validacao = 0;

		for(int h = 0; h < 60;h++){

			acum=0;
			Carrega_teste(conj_dados[conjunto_validacao[h]]);
			calc_out = fann_run(ann, entrada);

			for(int i = 0;i < nos_saida;i++){
				acum+=0.25* pow((saida_desejada[i]-calc_out[i]),2);
			}

			erro_validacao+= acum/60;
		}

		if(cont_epoca % 100 == 0)
			cout<<"Epoca ["<<cont_epoca<<"]"<<"Erro de estimacao:
			"<<erro_treinamento<<"\tErro validacao: "<<erro_validacao<<endl;

		cont_epoca++;
	}

	fann_destroy_train(estima);

	cout<<"Epoca ["<<cont_epoca<<"]"<<"Erro de treinamento:
	"<<erro_treinamento<<"\tErro validacao: "<<erro_validacao<<endl;

	fann_save(ann, "rede_neural.net");
	fann_destroy(ann);
}


int maior_indice(fann_type *calc_out){
	int maior = 0;

	for (int cont=1; cont < nos_saida; cont++){
		if(calc_out[maior] < calc_out[cont])
			maior = cont;
	}
```

```
		return maior;
	}

	float Teste(void){
		int indice, maior;
		fann_type *calc_out;
		cv::Mat image;
		std::vector<cv::Mat> frames;
		float acerto=0.0;
		struct fann *ann = fann_create_from_file("rede_neural.net");

		srand (time(NULL));

		for(int h = 0; h < 60;h++){
			do{
				indice=rand()%400;
			}while(Contido_conjunto(indice));

			teste.push_back(indice);

			image = cv::imread(conj_dados[indice]);

			//por se tratar de uma imagem binaria, sera processado apenas um camada
			cv::split(image,frames);
			image=frames[0]; //agora a imagem tem apenas uma camada

			for(int i=0;i<24;i++){
				for(int j=0;j<28;j++){
					if(image.at<uchar>(i,j)%254 == 1)
						entrada[(i*28)+j] = 1;
					else
						entrada[(i*28)+j] = -1;
				}
			}


			calc_out = fann_run(ann, entrada);
			maior = maior_indice(calc_out);

			if(index(conj_dados[indice].at(0)) == maior){
				acerto++;
			}

		}
		return (acerto/60)*100;
	}
};
```