Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

# Reconhecimento de Caracteres de Placa Veicular Usando Redes Neurais

Autor:

_____

Allan Almeida Dieguez

Orientador:

_____

Prof. Mariane Rembold Petraglia, Ph. D.

Examinador:

_____

Prof. Sergio Barbosa Villas-Boas, Ph.D

Examinador:

_____

Prof. Flávio Luís de Mello, D.Sc.

DEL

Fevereiro de 2010

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica – Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro – RJ CEP 21949-900

## DEDICATÓRIA

Dedico este trabalho ao povo brasileiro que contribuiu de forma significativa à minha formação e estada nesta Universidade. Este projeto é uma pequena forma de retribuir o investimento e confiança em mim depositados.

## AGRADECIMENTO

À Deus, pela constante orientação e pelos citados neste texto.

Aos meus pais, José Paulo e Neide, pelo esforço e dedicação despendidos na minha criação, e também por todo o apoio psicológico, financeiro e moral na minha formação.

Aos meus irmãos, pelo apoio e amizade em todas as etapas da minha vida.

Aos meus avós, pelo carinho e pelo exemplo de vida para todos à sua volta.

À minha namorada, Lucienne, pelo amor, companheirismo e paciência durante todo o tempo em que me assistiu lutar para finalizar esta importante etapa.

À Maria, que faz o melhor café do Rio de Janeiro.

Aos meus amigos e colegas da Engenharia, que acompanharam de perto todo o esforço e luta para chegar até aqui, compartilhando os bons e maus momentos durante esses longos anos de curso.

Ao professor Thomé, pela iniciação nos estudos em Inteligência Computacional e pelas importantes lições tanto na área acadêmica quanto no tratamento pessoal e profissional.

À professora Mariane, pela orientação e pela confiança em mim depositada durante a elaboração deste trabalho.

## RESUMO

Este trabalho apresenta duas técnicas de extração de descritores para o reconhecimento de caracteres de placa de automóvel usando uma rede neural do tipo MLP. A primeira técnica usa o mapa de bits puro, junto com as projeções vertical e horizontal do mesmo, como características da imagem do caractere a serem aprendidas pela rede neural. A segunda extrai momentos bidimensionais sobre o mapa de bits, em uma tentativa de representar a imagem do caractere por um conjunto de medidas estatísticas sobre a mesma. As duas soluções são comparadas em termos de desempenho total e por classe. Também é analisada a facilidade de aprendizado pela rede neural, em termos de tempo e convergência no treinamento.



Palavras-Chave: Redes Neurais Artificiais, Inteligência Computacional, Reconhecimento Automático de Caracteres, OCR.

## ABSTRACT

This paper presents two descriptor extraction techniques for recognition of license plate characters using a MLP type neural network. The first technique uses the bitmap itself as the character image's characteristics, along with it's vertical and horizontal projections, to be used in the neural network's learning. The second one extracts bidimensional moments from the bitmap, in an attempt to represent the character image by a set of statistical measures. These two solutions are compared in terms of total and by class performance. Also, the neural network's learning difficulty in training is analyzed by measures of time and convergence rates.



Keywords: Artificial Neural Networks, Computational Intelligence, Automatic Character Recognition, OCR.

## SIGLAS

DCC – Departamento de Ciência da Computação
GDA – *Gradient Descent Backpropagation*
LabIC – Laboratório de Inteligência Computacional
LMS – *Least Mean Square*
MLP – *Multi Layer Perceptron*
OCR – *Optical Character Recognition*
PCA – *Principal Component Analysis*
RNA – Rede Neural Artificial
RP – *Resilient Backpropagation*
SSE – *Sum Squared Error*
UFRJ – Universidade Federal do Rio de Janeiro

# Sumário

# Lista de Figuras

# Lista de Tabelas

# 1 Introdução

## 1.1 Tema

O tema deste trabalho é o estudo de técnicas de reconhecimento automático em uma imagem digital de um caractere de uma placa de automóvel, usando redes neurais artificiais do tipo MLP (*Multi Layer Perceptron*). Serão apresentados dois métodos distintos de extração de descritores, que analisam a imagem do caractere na forma de mapa de bits em tons de cinza. O primeiro utiliza o valor de cada pixel da imagem como descritor, assim como as projeções horizontal e vertical da mesma. O segundo é baseado no cálculo de momentos bidimensionais sobre o mapa de bits. Comparando estes dois métodos e analisando-os em termos de desempenho e robustez, será possível verificar a capacidade de generalização de uma rede neural artificial, assim como a solucionabilidade deste complexo problema.

## 1.2 Delimitação

O objeto de estudo é a classificação de imagens de caracteres de placas de automóveis, já segmentadas da imagem original da placa; a segmentação da imagem e o pré-processamento ao qual esta é submetida não fazem parte do escopo do trabalho. Sendo assim, as imagens de caractere usadas neste trabalho já foram segmentadas da placa e também já passaram por um pré-processamento, que realiza uma limpeza da imagem e a redimensiona para um tamanho padrão. Desta forma, serão analisadas imagens de dimensões iguais, com fundo branco e informação relevante em tons de cinza escuros, sendo também requisito que esta imagem contenha sempre somente um caractere.

## 1.3 Justificativa

A evolução da tecnologia automatizou muitas tarefas consideradas degradantes para o ser humano. Tarefas como inspeção de uma linha de montagem, que nos tempos

de Ford era feita apenas com os olhos humanos do inspetor, tornaram-se muito mais eficientes e rápidas com o uso de máquinas especializadas que utilizam visão computacional.

A visão computacional utiliza várias técnicas de software para classificar imagens recebidas de uma câmera ou sensor similar, muitas delas não determinísticas. Como em muitas outras áreas, nem sempre existem algoritmos que possam ser concebidos pela mente humana, devido à complexidade exagerada de alguns problemas, para sistematizar as ações a serem tomadas mediante a apresentação de imagens relativas ao problema em questão. Algumas vezes deve-se recorrer a soluções em que a máquina aprenda a resolver o problema ao generalizar uma variedade de exemplos a ela apresentados; temos como principal representante deste tipo de solucionador a rede neural artificial. Neste trabalho, será usada a rede neural do tipo *MLP*, uma ferramenta poderosa em problemas de classificação onde não existem algoritmos conhecidos capazes de resolvê-los.

A motivação do trabalho é a automatização do reconhecimento de placas veiculares, colaborando com a construção de sistemas de segurança e rastreamento de veículos, como controles de acesso a condomínios e estacionamentos. Outra motivação é a contribuição deste estudo para a área de visão computacional, cujas possibilidades de aplicação são ainda muito vastas e pouco exploradas.

## 1.4 Objetivos

O objetivo geral é construir um *OCR* robusto e eficiente que possa reconhecer caracteres de placa veicular, e que possa trabalhar com uma margem aceitável de ruído nas imagens. Desta forma, têm-se como objetivos específicos: (1) relacionar um conjunto de imagens grande e variado o suficiente para que seja representativo dos diversos ambientes onde foram capturadas as imagens de veículos; (2) elaborar dois tipos de extração de descritores: um mais simples, baseado no mapa de bits puro, e outro mais complexo, baseado em momentos bidimensionais; (3) treinar redes neurais com diferentes configurações para comparar as duas soluções em termos de eficiência e robustez.

## 1.5 Metodologia

A primeira etapa neste estudo é a preparação da massa de dados que será usada. Serão usados dados provenientes de um banco de imagens do sistema Kapta, um sistema de reconhecimento de placas de veículo totalmente concebido e desenvolvido no âmbito do LabIC - Laboratório de Inteligência Computacional da UFRJ - Universidade Federal do Rio de Janeiro. As imagens de caractere são segmentadas a partir de imagens de placa e depois analisadas manualmente, com a ajuda de programas construídos especificamente para isso, descartando as imagens ilegíveis e estragadas e corrigindo as marcadas erradamente. Essa marcação é relativa à qual caractere corresponde à imagem, e é feita automaticamente pelo programa que segmenta a imagem de placa. As imagens que sobram recebem um tratamento de forma a terem fundo branco e informação relevante em tons de cinza escuros. As imagens então são redimensionadas para que sua maior dimensão fique padronizada, pois os métodos de extração de descritores dependem desse tipo de padronização. A imagem final tem dimensão de 16 pixels de altura por 16 de largura, sendo que a informação relevante possui 14 pixels na sua maior dimensão; a imagem continua proporcional àquela da qual foi gerada, mas é colocada no centro de um quadrado branco, também por motivos de padronização.

Com a massa de dados devidamente tratada, podemos extrair seus descritores para usar no treinamento das redes neurais. Neste trabalho são usadas duas extrações bem diferentes entre si, nomeadas ***Mapa de Bits*** e ***Momentos***, explicadas a seguir:

***Mapa de Bits***: usa como descritores o próprio mapa de bits e as projeções horizontal e vertical. O mapa de bits é armazenado no vetor de descritores, de forma que os 16 pixels da primeira linha do mapa de bits são os 16 primeiros valores do vetor de descritores, os 16 pixels da segunda linha são os próximos 16 valores e assim por diante. Ocupados os 256 primeiros valores no vetor de descritores, são calculadas as projeções horizontal e vertical, cada uma contendo 16 valores, completando o vetor de descritores, com 288 valores no total.

***Momentos***: calcula uma série de momentos bidimensionais sobre o mapa de bits, sendo que cada momento calculado é um valor do vetor de descritores.

São gerados, assim, os dados necessários para os treinamentos das redes neurais. A partir disso, redes de variadas configurações serão treinadas e posteriormente testadas. As saídas da rede serão analisadas por dois métodos: um mais básico, *Winner*

*Takes All*, que analisa o maior valor de saída da rede, e um mais elaborado, que analisa as matrizes de confusão geradas nos testes.

As duas abordagens serão então comparadas por meio dos resultados obtidos em toda a massa de dados disponível e também por localidade de onde foram extraídas as imagens. Da comparação entre as duas abordagens será concluída a eficiência de generalização da rede neural proposta.

## 1.6 Descrição

No Capítulo 2 será apresentado o sistema Kapta e o seu funcionamento, com uma breve descrição das etapas de segmentação e reconhecimento de imagens de caracteres de placa veicular.

O Capítulo 3 dará o embasamento teórico deste trabalho, onde os fundamentos de Redes Neurais Artificiais serão explicados, com ênfase nos classificadores baseados em MLP.

A metodologia usada será apresentada no Capítulo 4, desde a preparação do banco de imagens e extração dos descritores até a construção das Redes Neurais. Também serão apresentadas as metodologias de análise das saídas das Redes Neurais, analisando as saídas das RNAs e também fazendo a análise das suas matrizes de confusão.

O Capítulo 5 dedica-se à análise dos resultados obtidos pelas Redes Neurais, comparando os métodos de extração de descritores das imagens e também os métodos de análise das saídas das Redes Neurais.

Finalmente, a conclusão e propostas para trabalhos futuros serão mostradas no Capítulo 6.

# 2 O Sistema Kapta

## 2.1 Introdução

O presente trabalho apresenta uma solução de reconhecimento de caracteres de placas de automóveis para o módulo de OCR do sistema Kapta. Assim, o conhecimento do seu funcionamento ajudará a compreensão do escopo de funcionamento do OCR proposto, da forma como as imagens do caractere são obtidas até o objetivo do reconhecimento automático da placa pelo sistema.

O sistema Kapta foi desenvolvido no Laboratório de Projetos do DCC, e tem como objetivo principal o controle do fluxo de entrada e saída da Cidade Universitária, reconhecendo automaticamente a placa do veículo que passar por cada um dos portões monitorados. Totalmente desenvolvido por alunos do Departamento de Ciência da Computação e da Engenharia Eletrônica e de Computação, ele faz parte do Programa de Segurança da UFRJ.

Seu histórico de desenvolvimento e seus primeiros casos de uso podem ser encontrados na dissertação de mestrado de Bruno Guingo [4], onde foram desenvolvidos os primeiros métodos de reconhecimento do sistema usando RNAs, e no artigo subseqüente [5], onde o sistema é referenciado pela primeira vez pelo nome Kapta (Lilenbaum [3] apud Guingo [5]).

Inicialmente o sistema objetiva apenas analisar o fluxo de veículos no Campus, fornecendo informações sobre quem entra ou sai e quando o fez, facilitando o trabalho de rastreamento de veículos, o que permite ações mais eficientes contra roubo de veículos, por exemplo. Em uma próxima etapa, o sistema servirá como controle de acesso, permitindo a entrada automática dos veículos cadastrados e barrando os não cadastrados, que serão abordados por seguranças que estão alocados nas cabines; este esquema só funcionará nos horários em que a Cidade Universitária estiver com os portões fechados, voltando ao seu estado de analisador nos horário normais de funcionamento.

## 2.2 Princípios de Funcionamento

### 2.2.1 Etapas do processamento

O sistema Kapta usa alguns módulos básicos na sua operação: a Captura, o OCR e o Banco de Dados. A Captura aciona a câmera e envia a imagem do veículo para o módulo do OCR, que processa a imagem e retorna o resultado do reconhecimento da placa. Então, esse resultado é comparado com os veículos cadastrados no Banco de Dados para permitir o acesso ou abordar o motorista.

O módulo de OCR é dividido em três partes principais: a Localização, a Segmentação e o Reconhecimento. A Localização procura, na imagem do veículo, a placa para reconhecer seus caracteres. A imagem da placa, recortada da imagem principal, é processada pela Segmentação, que extrai as imagens dos caracteres e as envia para o Reconhecimento. Este, por sua vez, classifica as imagens e retorna o resultado da placa na forma de uma cadeia de caracteres reconhecidos. Nas seções a seguir, serão explicadas com mais detalhes todas as etapas.

### 2.2.2 Detecção do veículo e captura da imagem

A Captura utiliza um módulo de software, operando na máquina dentro da cabine de segurança onde está instalado o sistema, que é acionado por um sensor indutivo enterrado no asfalto. Ao ser acionado, o módulo inicia a captura de imagens das câmeras instaladas em postes, que fotografam o veículo. Essas imagens são então enviadas ao módulo de OCR, para reconhecimento da placa do veículo. Na imagem mostrada na figura 2.1 pode-se ver a marca no asfalto do sensor. A imagem mostra o veículo no momento que ele atravessa o laço indutivo, disparando a Captura.


Figura 2.1 - Imagem de veículo em cima do sensor, obtido pela Captura.
Fonte: Sistema Kapta, 2007.

Para evitar problemas de enquadramento, a Captura tira três fotos cada vez que um veículo passa no sensor. Essa redundância também é necessária para o cálculo da confiabilidade do reconhecimento da placa, que usa uma estimativa a partir dos três resultados obtidos pelo OCR.

### 2.2.3 Localização da placa

A localização da placa na imagem é feita totalmente por meio de processamento de imagens. Segundo Ricardo Lilenbaum [3], no seu trabalho final de graduação, para localizar a placa são utilizados algoritmos de detecção das bordas dos caracteres, como o Canny. O algoritmo da Localização procura, a partir da imagem de bordas, pixels adjacentes que formem conjuntos parecidos com bordas de caracteres, levando em conta o tamanho da placa na imagem, parâmetro proveniente da configuração do OCR.

Ainda segundo Ricardo [3], a busca pela região da placa é realizada mais de uma vez para cada foto, assim produzindo várias regiões candidatas. Atualmente, três regiões são buscadas na varredura. Ao final dessa etapa, essas regiões são ordenadas pela suas probabilidades de conter a placa e a eleita como a mais provável é passada para a próxima etapa.

A Localização pode ser atrapalhada por vários fatores, como a presença de mais de uma placa na imagem, grau de inclinação da placa, placa incompleta cortada pelos limites da foto, inscrições alfanuméricas na lataria do veículo, sombras e clarões. Na figura 2.2 pode-se ver a imagem de uma placa extraída pela Localização a partir da imagem da figura 2.1.

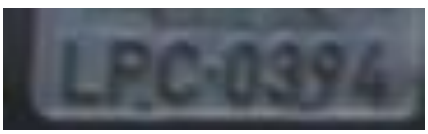


Figura 2.2 - Imagem da placa extraída pela Localização.
Fonte: Sistema Kapta, 2007.

### 2.2.4 Segmentação dos caracteres

A Segmentação usa processamento de imagens para separar as imagens de

caracteres da imagem da placa. Segundo Ricardo [3], a região da placa é transformada em uma imagem em tons de cinza e uma transformação de maximização de contraste é aplicada. A seguir, a imagem tem seus grupos conexos definidos e com base nos tamanhos e posições relativas desses grupos os caracteres são encontrados e extraídos da imagem; cada caractere é salvo em uma imagem de 16x16 pixels (Ricardo [3] apud Rodrigues [10]), de fundo branco e informação relevante em tons de cinza escuros, com uma moldura branca de um pixel de espessura. As imagens dos caracteres segmentados da placa da figura 2.2 estão mostradas na figura 2.3, a seguir.

Figura 2.3 - Imagem dos caracteres extraídos pela Segmentação.

Fonte: Sistema Kapta, 2007.

A padronização do tamanho da imagem dos caracteres é devida ao fato de que o classificador usado inicialmente no sistema Kapta era baseado na extração de descritores do *MapaDeBits*, desenvolvido por Bruno Guingo [4]. Essa extração é totalmente dependente da disposição da imagem, pois usa como descritores todos os pixels da imagem, mais as projeções horizontal e vertical da mesma. Como o vetor de descritores deve sempre ter o mesmo tamanho, se procede desta maneira.

### 2.2.5 Reconhecimento dos caracteres

Os caracteres separados em imagens estão prontos para serem reconhecidos pelo módulo do Reconhecimento. Os três primeiros caracteres, representando as letras da placa, são processados por uma RNA classificadora de letras, enquanto os quatro restantes são classificados por uma de dígitos.

O módulo de Reconhecimento classifica as imagens, para cada uma atribuindo uma classe de caractere estimada pela RNA. Essa informação, junto com outras de confiabilidade do reconhecimento, é enviada para o sistema principal, que a utiliza nas várias aplicações às quais o sistema se propõe, seja armazenando no banco de dados ou verificando no mesmo se a placa é cadastrada.

# 3 As Redes Neurais Artificiais

## 3.1 Introdução

As redes neurais artificiais são muito utilizadas para solucionar problemas em que, por serem muito complexos, não existem algoritmos de bom desempenho capazes de resolvê-los. Esse é o caso do reconhecimento automático de caracteres, que encontra nesta área de estudo ferramentas competentes o suficiente para gerar soluções que utilizam o aprendizado automático dos padrões do problema para a resolução do mesmo.

Neste capítulo serão abordados alguns conceitos básicos de redes neurais, da concepção do neurônio até as diferentes arquiteturas usadas, explicando alguns paradigmas do aprendizado e casos de uso em projetos. Em seguida, será explicada a arquitetura MLP, muito utilizada em classificadores, e o conceito do algoritmo *Backpropagation* usado para o treinamento da RNA.

## 3.2 A Rede Neural Artificial

### 3.2.1 O conceito da Rede Neural Artificial

As redes neurais artificiais são estruturas computacionais radicalmente diferentes dos demais modelos convencionais, inspiradas no modelo biológico do cérebro humano, uma máquina de processamento de informações altamente complexa, não-linear e paralela (Haykin [1]). São compostas por unidades de processamento mais simples chamadas neurônios (um paralelo com o neurônio biológico), dispostas em uma ou mais camadas altamente interconectadas e processando em paralelo. A capacidade mais interessante de uma rede neural artificial é a de, assim como o cérebro humano, aprender através de exemplos e acumular o conhecimento na forma de pesos sinápticos para processamento posterior, sendo uma ferramenta poderosa na área de reconhecimento de padrões.

Segundo Haykin [1], o enorme poder computacional de uma rede neural é

devido a dois fatores: sua estrutura massivamente paralela e distribuída e sua capacidade de generalização por aprendizado. Por generalização entende-se a sua capacidade de, após ser treinada com diversos exemplos, relacionar corretamente entradas nunca apresentadas à rede com as saídas corretas; isso indica que a rede neural extrai informações adicionais dos exemplos apresentados, não se resumindo a um mapeamento linear de entrada-saída. Com isso também se pode dizer que a rede neural gera seu próprio conhecimento.

Os principais tipos de problemas que podem ser solucionadas por uma RNA são a classificação, a categorização (ou *clustering*), o reconhecimento de padrões e a previsão. A classificação busca por uma função que consiga relacionar elementos a uma classe, dentre um conjunto finito e pré-definido. A construção do modelo segundo esta estratégia pressupõe o conhecimento prévio das possíveis classes e a correta classificação dos exemplos usados na modelagem. A classificação automática de caracteres é uma aplicação dessa categoria de problema, em que se deseja relacionar a imagem da letra ou dígito com a classe correspondente.

A categorização consiste na busca de similaridades entre os dados que permita definir um conjunto finito de classes ou categorias que os contenha e os descreva, sem haver um prévio conhecimento sobre o número de classes possíveis nem a possível pertinência dos exemplos usados na modelagem. Este tipo de solução é muito usado em *data mining* para separar categorias similares de dados em um conjunto.

O reconhecimento de padrões consiste na associação do padrão corrente ao padrão representativo de uma e somente uma entidade previamente armazenada em memória. Exemplos de aplicações podem ser encontrados em reconhecedores biométricos como os de face, voz, digitais e retina.

A previsão é muito utilizada em séries temporais, para prever o comportamento ou valor futuro com base em valores anteriores dos elementos da série ou de outros não pertencentes, porém relacionados, a ela. A previsão da Bolsa de Valores e a climática são exemplos de problemas em que se pode usar redes neurais de previsão temporal.

### 3.2.2 Modelagem computacional do neurônio

O neurônio biológico pode ser simplificado de forma a ser representado em três partes distintas: o corpo celular, os dendritos e o axônio. Os impulsos nervosos, também

chamados sinapses, são recebidos pelos dendritos, que são as extensões menores da célula. Esses impulsos são enviados ao corpo celular, que os processa e envia um novo sinal (gerado a partir dos impulsos recebidos) pelo axônio, que transmite esse novo impulso para os dendritos adjacentes de outros neurônios. A modelagem matemática do neurônio surgiu a partir dessa idéia de representação do mesmo como um grafo direcionado (Haykin [1]), onde os dendritos são as *m* entradas e o axônio a única saída. A figura 3.1 mostra o esquema simplificado do neurônio biológico.

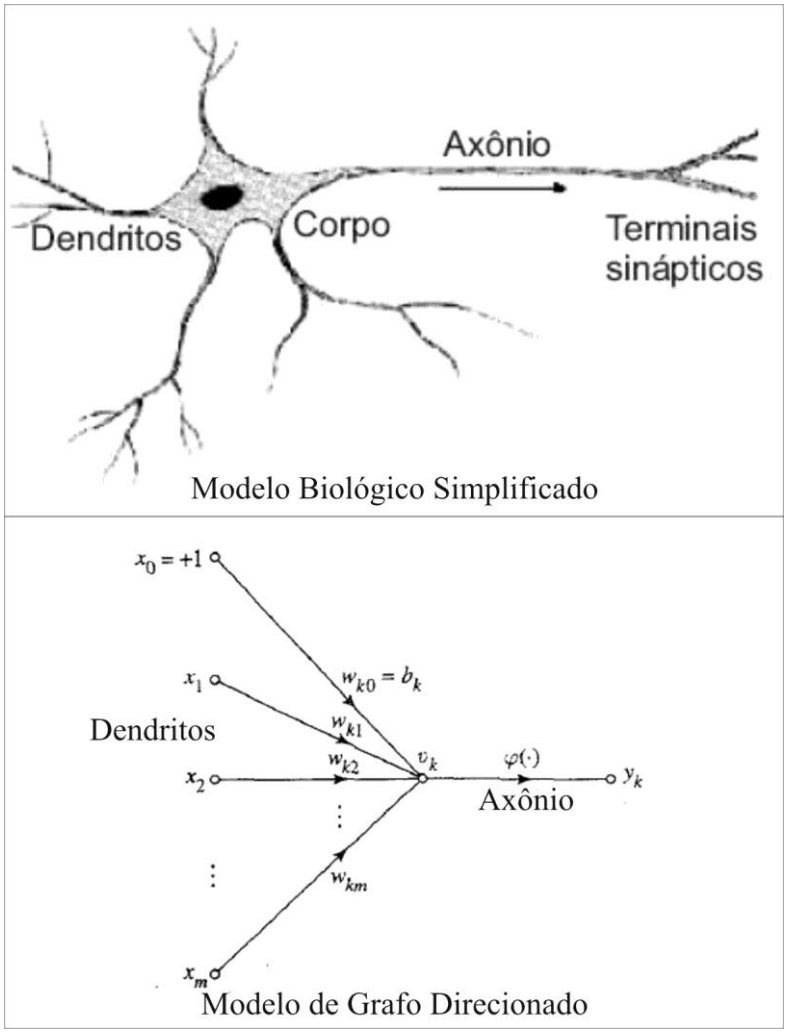


Figura 3.1 - Esquema simplificado do neurônio e seu grafo equivalente.
Fonte: Haykin [1], p.30, modificado.

O primeiro modelo matemático de neurônio é de 1944, surgindo no trabalho pioneiro de McCulloch e Pitts, *A Logical Calculus of the Ideas Immanent in Nervous Activity* [11]. O neurônio de McCulloch-Pitts foi concebido como uma estrutura lógica simples, com diversas entradas, ponderadas por pesos numéricos que representam a força da sinapse de cada dendrito, e uma saída binária que responda 0 ou 1, de acordo

com o seu estado de ativação.

O princípio de funcionamento é simples: o neurônio deve receber um vetor de valores binários de tamanho ***m***, um para cada entrada; tais valores serão a partir de agora denominados $x_k$. Cada entrada do neurônio é ponderada por um valor de peso sináptico, denominados $w_k$, que definem a força da sinapse naquela entrada. A função de ativação é calculada pela soma das entradas ponderada pelos pesos; o valor calculado então é comparado com um valor interno de limiar (ou *threshold*) do neurônio: se for maior que este limite, a saída será **1**, caso contrário será **0**. A equação 3.1 mostra o funcionamento da ativação do neurônio de McCulloch-Pitts.

$$Saída = \begin{cases} 1, & \text{se} \quad \sum_{k=1}^{m} x_k . w_k \geq Limiar \\ 0, & \text{se} \quad \sum_{k=1}^{m} x_k . w_k < Limiar \end{cases} \tag{3.1}$$

A natureza binária do neurônio de McCulloch-Pitts teve origem no modelo da época no qual o neurônio biológico seguia um comportamento "tudo-ou-nada" [1], sendo essa a principal causa da sua limitação. Os pesos também eram fixos, não podendo ser ajustados por um treinamento, portanto não existia aprendizado.

Ainda assim, foi comprovado que uma rede neural usando um número suficiente de neurônios de McCulloch-Pitts é capaz de calcular qualquer função computável [11]. De fato, uma rede com neurônios desse tipo podia ser treinada para solucionar problemas lógicos de primeira ordem como "*or"*, "*and"* e "*not*"; as funções "*xor*" e "*xnor*", entretanto, não eram capazes de serem obtidas.

Em 1949, Hebb publicou o livro *The Organization of Behavior* [12], com algumas modificações essenciais no modelo de McCulloch-Pitts. Hebb propôs a teoria de que o peso (ou força) das sinapses em um neurônio biológico seria modificado de acordo com o processo de aprendizado, postulando que a efetividade da sinapse entre dois neurônios aumenta cada vez que esses neurônios se comunicam por ela.

Com esse avanço no estudo das redes neurais, em 1958 Rosemblat propôs, no seu livro *Principles of Neurodynamics* [13], o modelo de neurônio *perceptron*. Esse modelo apresentava um método inovador de aprendizagem supervisionada, onde os pesos do neurônio seriam agora modificados por um processo em que exemplos do problema seriam responsáveis pelo ajuste dos pesos de cada entrada, a partir do

resultado da ativação comparado com uma saída esperada.

O processo de aprendizado postulado permitiu que se usassem entradas diferentes da binária, mas as saídas do neurônio continuavam iguais. O fato de as saídas de um neurônio só fornecerem informações de ativado ou desativado limitava as redes neurais a uma camada de neurônios apenas. Além disso, o *perceptron* apenas poderia classificar problemas com classes linearmente separáveis. Assim, foram introduzidas funções de ativação que pudessem assumir valores diferentes de zero e um, como as lineares de *rampa* e *linear pura*, além das funções não-lineares como a *gaussiana* e as *sigmoidais* e da função *degrau*.

A função de ativação *linear pura* foi concebida para problemas em que a saída pode assumir quaisquer valores entre menos infinito e mais infinito, sendo muito usada para estimadores temporais e ajuste de curvas. Já *rampa* foi a primeira a ser usada em classificadores, pois permitia valores reais entre os valores de saturação, sendo uma combinação das funções *linear* e *degrau*. Esta última é um adaptação da função binária usada nos primeiros *perceptrons*, podendo ser ajustada para faixas de valores diversas. Os gráficos destas funções estão mostrados na figura 3.2.

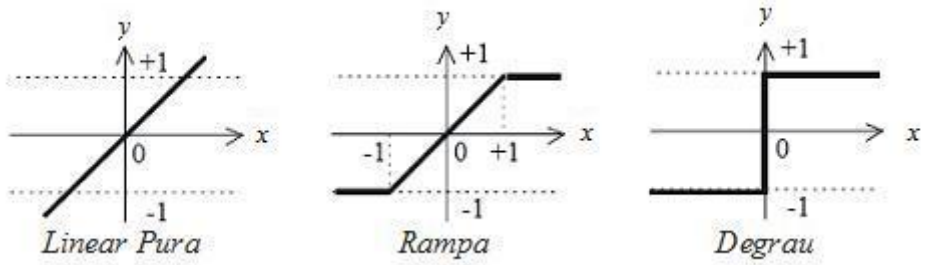


Figura 3.2 - Funções de ativação *linear pura*, *rampa* e *degrau*.
Fonte: *Matlab Neural Network Toolbox ™ 6 User's Guide* [8], p.14-17, modificado.

A equação (3.2) é relativa à função *linear*, onde $\alpha$ é o coeficiente angular, ou a tangente do ângulo entre a reta e o eixo horizontal ($x$).

$$y = \alpha.x \tag{3.2}$$

A função *degrau* é representada pela equação (3.3). Essa função polariza as saídas maiores ou iguais a zero em $+N$ , e as menores em $-N$.

$$y = \begin{cases} +\,\mathrm{N}, & \text{se} \quad \mathrm{x} \geq 0 \\ -\,\mathrm{N}, & \text{se} \quad \mathrm{x} < 0 \end{cases} \tag{3.3}$$

A função *rampa*, representada pela equação (3.4), combina as funções *linear* e *degrau* da seguinte maneira: se ***x*** for maior que um limiar ***+L***, então *y* é polarizado em ***+N***; no caso de ***x*** ser menor que um limiar ***-L***, então ***y*** é polarizado em ***–N***. entre os valores ***–L*** e ***+L***, a função se comporta como a *linear*.

$$y = \begin{cases} +\,\mathrm{N}, & \text{se} \quad \mathrm{x} > +\,\mathrm{L} \\ -\,\mathrm{N}, & \text{se} \quad \mathrm{x} < -\,\mathrm{L} \\ \alpha.\mathrm{x}, & \text{caso contrário} \end{cases} \tag{3.4}$$

As funções de ativação *gaussiana* e as *sigmoidais* são usadas quase que exclusivamente em classificadores. Por serem não-lineares, possuem uma forte capacidade de separação de grupos. Os gráficos destas funções estão mostrados na figura 3.3.

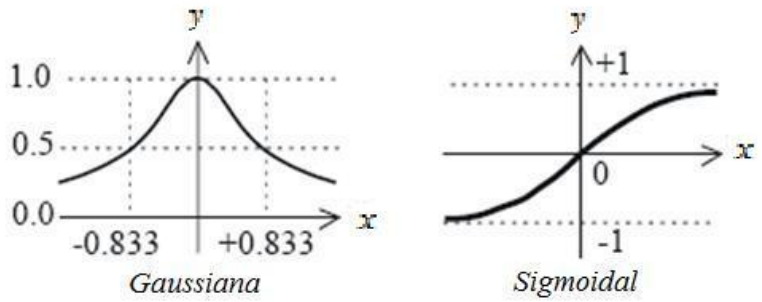


Figura 3.3 - Funções de ativação *gaussiana* e *sigmoidal*.
Fonte: *Matlab Neural Network Toolbox ™ 6 User's Guide* [8], p.14-17, modificado.

A equação (3.5) é relativa à função *gaussiana*, também conhecida como *distribuição normal*. Essa função é muito conhecida no estudo de probabilidades, e é descrita pelos seus parâmetros de média $\mu$ (posição central da *gaussiana*, onde está seu valor máximo) e o desvio padrão $\sigma$ (que define a abertura).

$$y = e^{-\frac{(x-\mu)^2}{2.\sigma^2}} \tag{3.5}$$

A família de curvas da função *sigmoidal* é a mais usada para funções de ativação em neurônios, pois possui um comportamento ao mesmo tempo linear e não-linear. As funções mais utilizadas são a *logística-sigmóide*, que possui valores entre 0 e 1, e a *tangente-sigmóide*, com valores entre -1 e 1. A equação genérica da *sigmoidal* está mostrada na equação (3.6), onde o valor $\lambda$ representa a inclinação da porção linear da curva; se $\lambda$ tender ao infinito, a função *sigmoidal* torna-se a função *degrau*.

$$y = \frac{1}{1 + e^{-\lambda . x}} \tag{3.6}$$

Com essas novas funções de ativação, o *perceptron* pôde ser usado em redes neurais artificiais com mais de uma camada, pois o uso de funções deriváveis tornaria possível a criação do algoritmo *backpropagation* em 1986 por Rosemblat, que publicou em 1992 o livro *Backpropagation: Theory, Architectures and Applications* [15].

O *perceptron*, como é usado até a atualidade, pode ser representado como na figura 3.4, sendo que $\varphi$ representa a função de ativação e $w_0$ representa o *bias* do neurônio, valor que permite uma maior maleabilidade no processo de aprendizado, sendo tratado como um peso (que também é ajustado) que entra no somatório ponderado das entradas com os outros pesos.

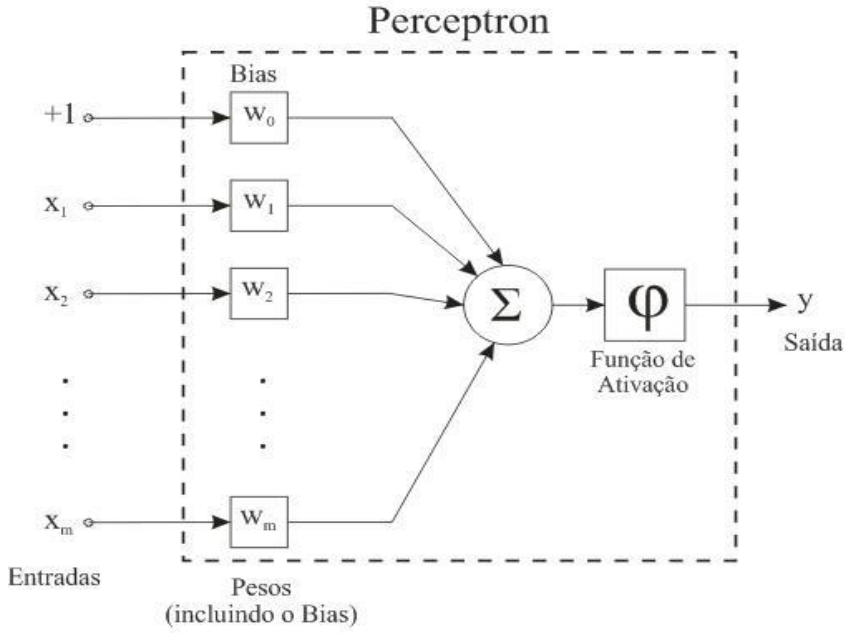


Figura 3.4 - Diagrama de blocos do *perceptron*.

Assim, para calcular a saída ***y*** de um neurônio de ***m*** entradas $x_k$, usa-se a equação (3.7):

$$y = \varphi\left(\sum_{k=1}^{m} w_k x_k + b_k\right) \tag{3.7}$$

### 3.2.3 Paradigmas de Aprendizado

Uma das principais vantagens em se usar redes neurais artificiais é a capacidade de aprendizado da mesma, que permite encontrar soluções em problemas nos quais a capacidade lógica de um algoritmo sozinha não consegue resolver. O aprendizado pode ser definido como a adaptação do conhecimento da rede para que esta responda de maneira apropriada aos estímulos do ambiente.

Assim, primeiramente deve-se definir o que é o conhecimento de uma rede neural artificial. A equação (3.7) mostra como uma saída de um neurônio pode ser calculada a partir das suas entradas, que são os estímulos do ambiente para este neurônio. Observa-se que os pesos sinápticos definem a maneira como esses estímulos serão usados no cálculo da saída. Os pesos sinápticos seriam, na comparação com o neurônio biológico, a força ou consistência da sinapse no dendrito ou entrada correspondente, que é modificada a partir dos estímulos externos de forma a modelar o comportamento do neurônio, consistindo nisto o aprendizado. Assim, o conhecimento de um neurônio pode ser definido como os pesos sinápticos de cada entrada do mesmo, que são modificados com a apresentação de exemplos com o objetivo de responder corretamente a esses estímulos.

Existe uma grande variedade de algoritmos de aprendizado, que basicamente diferem uns dos outros pela maneira que os pesos dos neurônios são ajustados. Uma divisão primária dos algoritmos pode ser feita em aprendizado supervisionado e não-supervisionado.

O aprendizado supervisionado utiliza um conjunto de entradas e saídas esperadas, denominado massa de treino. Os parâmetros da rede neural são ajustados pelo coeficiente de erro, calculado pela diferença entre a saída gerada pelo processamento das entradas e a saída esperada. Assim, iterativamente, a rede neural processa as entradas e compara a saída obtida com a esperada, ajustando os pesos até

que o treinamento seja interrompido. A interrupção do treinamento se dá por diferentes condições, sendo este um importante estudo na construção de soluções usando redes neurais. Na Seção 4.4.2 são apresentados alguns métodos de interrupção do algoritmo de treinamento usados neste projeto.

O principal representante do aprendizado supervisionado é o algoritmo de treinamento baseado em erros, ou *delta-rule*, que é uma das bases do algoritmo *Backpropagation*. O *Backpropagation* é o algoritmo de treinamento usado nas redes neurais tipo MLP, e será tratado com mais detalhes na Seção 3.3.2.

No aprendizado não supervisionado, a informação de saída esperada, se existir, não é usada para o ajuste de pesos por um módulo supervisor, sendo necessário o uso de outros métodos para direcionar o treinamento das redes neurais. Em geral, esses métodos empregam o uso de métricas estatísticas obtidas em cima do ambiente de aplicação do problema. Esse tipo de treinamento é muito utilizado em redes auto-organizáveis e em algumas redes temporais.

Alguns algoritmos de aprendizado não supervisionado são o Hebbiano (mais antiga e famosa de todas as regras de aprendizado) e o Competitivo, muitas vezes chamado *winner-takes-all* (não deve ser confundido com o algoritmo homônimo de análise das saídas de uma rede neural). Esses algoritmos atuam fortalecendo as sinapses dos neurônios que apresentam as melhores respostas a uma dada entrada, utilizando para isso métodos heurísticos. Esses algoritmos não serão tratados neste trabalho, por estarem fora do escopo de classificadores, sendo uma boa referência para estudos posteriores o Capítulo 2 do Haykin [1].

### 3.2.4 Arquiteturas de Redes Neurais

A arquitetura de uma rede neural pode ser classificada quanto ao número de camadas, o tipo de conexão entre as camadas e o tipo de conectividade entre os neurônios. A variação desses parâmetros influencia no tipo de aprendizado e também nos tipos de problemas que a rede neural pode resolver,

Em relação ao número de camadas, as redes podem ter uma única camada ou múltiplas. Cada camada é formada por um número de neurônios em paralelo, possuindo um número de entradas qualquer e um número de saídas igual à quantidade de neurônios. No caso de múltiplas camadas, as saídas dos neurônios são usadas como

entrada da próxima.

Uma rede multicamadas é composta por uma camada de saída e uma ou mais camadas escondidas. Segundo Haykin [1], ao se adicionar uma ou mais camadas escondidas, a rede neural consegue extrair informações de ordem superior, de forma que, quanto mais camadas, mais informações não triviais se pode extrair do ambiente. George Cybenko publicou em 1988 um trabalho [16] no qual dizia que uma rede com uma camada escondida possui a capacidade de aproximar qualquer função contínua, enquanto uma com duas ou mais consegue implementar qualquer função.

Uma rede neural é completamente conectada quando todas as entradas de uma camada são conectadas a todos os neurônios da mesma, em todas as camadas. Se uma ou mais conexões não existirem, a rede é classificada como parcialmente conectada. Pode ocorrer durante o treinamento que um ou mais pesos de um neurônio sejam colocados em zero; isso na prática anula aquela conexão, pois a multiplicação de uma entrada por esse peso sináptico será sempre zero. Ainda assim, a conectividade é referente à arquitetura inicial da rede neural, ou seja, uma rede que tenha zerado uma sinapse não passa a ser parcialmente conectada na classificação usual.

Os tipos de conexão definem se a rede é cíclica ou acíclica. Uma rede acíclica possui uma propriedade chamada *feedforward*, que significa que a informação é sempre passada pelas camadas em direção à camada de saída. As redes do tipo *Single Layer Perceptron* e as *Multi Layer Perceptron* (*MLP*) são do tipo *feedforward*. A figura 3.5 mostra uma rede *MLP* de três camadas (duas escondidas e a de saída).

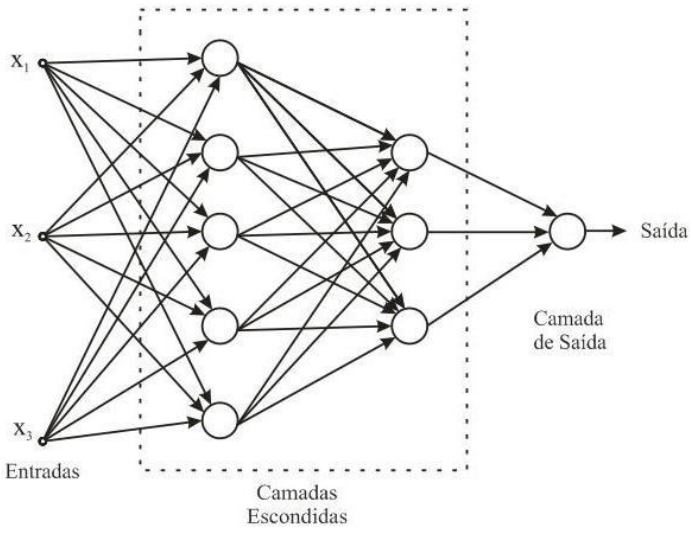


Figura 3.5 - Rede neural de três camadas do tipo *Multi Layer Perceptron*.

A rede cíclica, ou recorrente, possui realimentação (*feedback*) entre as camadas, sendo muito usada para problemas de previsão temporal. A realimentação pode ocorrer entre duas camadas escondidas, entre a camada de saída e a de entrada, entre a camada de saída e uma escondida ou entre uma camada escondida e a entrada. Nessas realimentações geralmente empregam o uso de *delays* (ou atrasos), na forma de janelas temporais. Por exemplo, a informação da saída que é realimentada para a entrada no instante t deve ser aquela do instante t-1. No modelo equivalente usando transformada Z, o atraso temporal t-1 é representado como $z^{-1}$.

As redes neurais de Elman e Jordan são as mais conhecidas redes com realimentação. A rede de Elman possui realimentação da camada escondida para a entrada, enquanto na rede de Jordan a realimentação é da saída para a entrada. Na figura 3.6 é mostrada uma rede de Elman com duas camadas e janela de atraso no tempo $z^{-1}$.

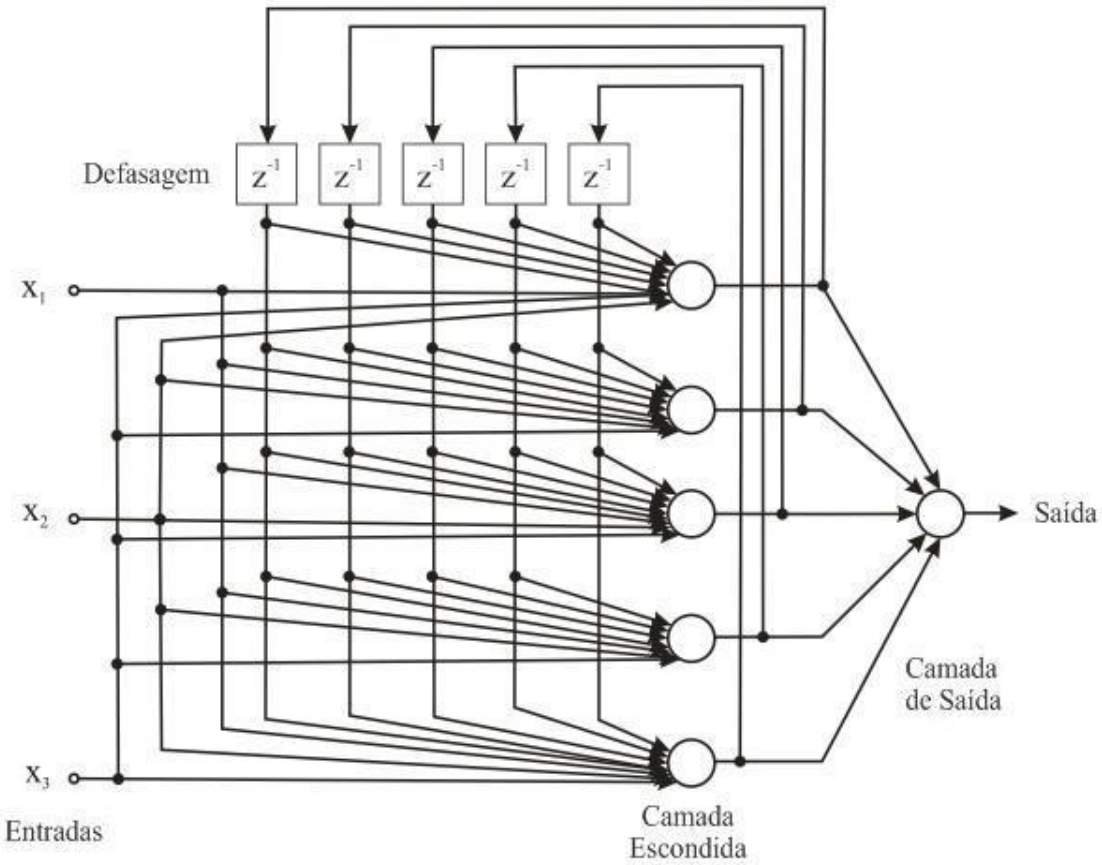


Figura 3.6 - Rede neural de duas camadas do tipo Elman.

## 3.3 O Multi Layer Perceptron

### 3.3.1 Topologia do MPL

O *Multi Layer Perceptron* (MLP) pode ser descrito como uma rede neural com múltiplas camadas, acíclica e completamente conectada, sendo uma das arquiteturas mais utilizadas em problemas de classificação. Segundo Haykin [1], uma rede MLP tem três características distintivas:

1. O modelo de cada neurônio da rede MLP deve conter uma função de ativação não-linear e diferenciável em todo o intervalo de aplicação, como é o caso das *sigmoidais*. A presença da não-linearidade é importante, pois caso a rede fosse composta apenas de funções lineares, a relação de entrada-saída poderia ser simplificada para uma rede de camada única (*Single Layer Perceptron*);
2. A rede contém um ou mais neurônios escondidos que não fazem parte das camadas de entrada ou de saída da rede. Estes neurônios permitem à rede aprender tarefas complexas, extraindo progressivamente mais informações significativas dos padrões de entrada;
3. A rede deve exibir um alto grau de conectividade (recomendável que seja totalmente conectada). A arquitetura da rede deve ser definida em número de neurônios, pois o próprio algoritmo de treinamento se encarrega de eliminar as conexões desnecessárias ou nocivas à rede.

A arquitetura MLP é uma estrutura de processamento paralelo muito poderosa e com grande capacidade de aprendizado. Ao mesmo tempo, sua característica não-linear e altamente conectada torna a análise teórica do seu funcionamento muito difícil. Também a presença de camadas de neurônios escondidas torna o processo de aprendizado difícil de visualizar, pois o processamento não-linear das entradas pela camada escondida não é mapeado diretamente na saída.

Muitos tipos de problemas podem ser resolvidos com a arquitetura MLP, de classificadores a rastreadores de função, sendo que para cada tipo uma solução específica deve ser concebida. Para rastrear uma função com valores reais, por exemplo,

a função de ativação dos neurônios da camada de saída deve ser a *linear pura*, discutida na Seção 3.2.2. Para classificadores, o melhor tipo de arquitetura da MLP é aquele que tem uma saída para cada classe, usando como função de ativação a *logística-sigmóide*; a saída da rede neural fica parecida com um vetor de probabilidades para cada classe, podendo ser analisada de diversas formas. Essa arquitetura de classificadores foi implementada neste trabalho, como se pode ver na Seção 4.3.3.

### 3.3.2 Treinamento usando *Backpropagation*

O processo de aprendizado da MLP é o algoritmo *Backpropagation*, baseado no aprendizado por correção de erros (*delta-rule*). O algoritmo consegue com sucesso propagar a informação de erro da camada de saída para as camadas escondidas, podendo ser visto como uma generalização do filtro adaptativo *least-mean-square* (LMS).

Basicamente, o *Backpropagation* consiste em duas etapas de processamento: o *forward pass* e o *backward pass*. O primeiro é equivalente ao *feedforward*, em que as entradas são processadas pelas camadas em seqüência, e a saída é obtida na última camada. A saída obtida é então subtraída da saída esperada (*target*), gerando a informação de erro (*error signal*). A informação de erro é então propagada na direção contrária, ajustando os pesos no processo, vindo daí o nome *Backpropagation*. Essa propagação do erro é feita usando a generalização da *delta-rule*, que usa derivadas parciais para calcular os ajustes nos valores dos pesos de camada. Por isso, as funções de ativação dos neurônios devem ser sempre deriváveis, pois o algoritmo seria impossível de aplicar em funções que não o fossem, limitando as redes a apenas uma camada.

Os passos do algoritmo Backpropagation para uma iteração de treinamento são:

1. As saídas são calculadas processando as entradas pelas camadas da rede, no *forward pass*;
2. O erro relativo à saída é calculado pela subtração desta com a saída esperada;
3. A energia do erro é calculada sobre os erros de todas as saídas, resultando no erro da rede para a entrada atual, iniciando o *backward pass*. Dependendo do tipo de treinamento, esse erro é calculado por

elemento apresentado, ajustando os pesos da rede para cada entrada (treinamento incremental), ou então fazendo a média aritmética de todos os erros, ajustando os pesos da rede após processar todas as entradas (treinamento em lotes ou *batch*);

4. Calcula-se o gradiente da energia do erro em relação aos pesos da camada de saída, atualizando esses pesos em seguida. Os pesos são ajustados proporcionalmente a um valor chamado passo de aprendizado, que define o quanto se pode variar o peso a cada iteração. Este processo é análogo ao método LMS (*least-mean-square*) usado no treinamento por correção de erros em redes de um única camada;
5. É calculado o erro em relação à saída da última camada escondida com a camada de saída, usando um cálculo de retropropagação ponderada do erro de cada neurônio da camada de saída;
6. As demais camadas escondidas são ajustadas seguindo o passo 5, onde a camada de saída é representada pela camada posterior à atual.

O algoritmo continua as iterações até que uma condição de parada seja alcançada. Essa condição de parada pode ser número máximo de iterações alcançado, erro mínimo atingido ou a verificação de falha na convergência da curva de erros do treinamento. Alguns métodos de parada de treinamento usados neste trabalho são discutidos na Seção 4.4.2.

Para a formulação matemática do Backpropagation, algumas notações devem ser fixadas. Considerando uma rede MLP genérica, a contagem de camadas é feita a partir da saída para a entrada. Assim, a camada 0 é relativa aos neurônios de saída, a 1 é relativa à primeira camada escondida, a 2 à segunda e assim por diante. Os parâmetros sinápticos de um neurônio podem ser definidos como a seguir:

$\mathbf{y}^{\mathbf{k}}_{\mathbf{i}}$ - saída do neurônio **i** da camada **k**;
$\mathbf{w}^{\mathbf{k}}_{\mathbf{ij}}$ - peso sináptico de posição **j** do i-ésimo neurônio da k-ésima camada.

A saída do neurônio é calculada pela equação (3.7), sendo que as entradas do neurônio de uma camada são as saídas de todos os neurônios da camada anterior. Sendo assim, as saídas de todos os neurônios de uma camada são conectadas a todos os pesos da camada seguinte. Na figura 3.7 é mostrada a conexão entre dois neurônios, um da

primeira camada escondida (notação da saída para a entrada) e outro pertencente à de saída.

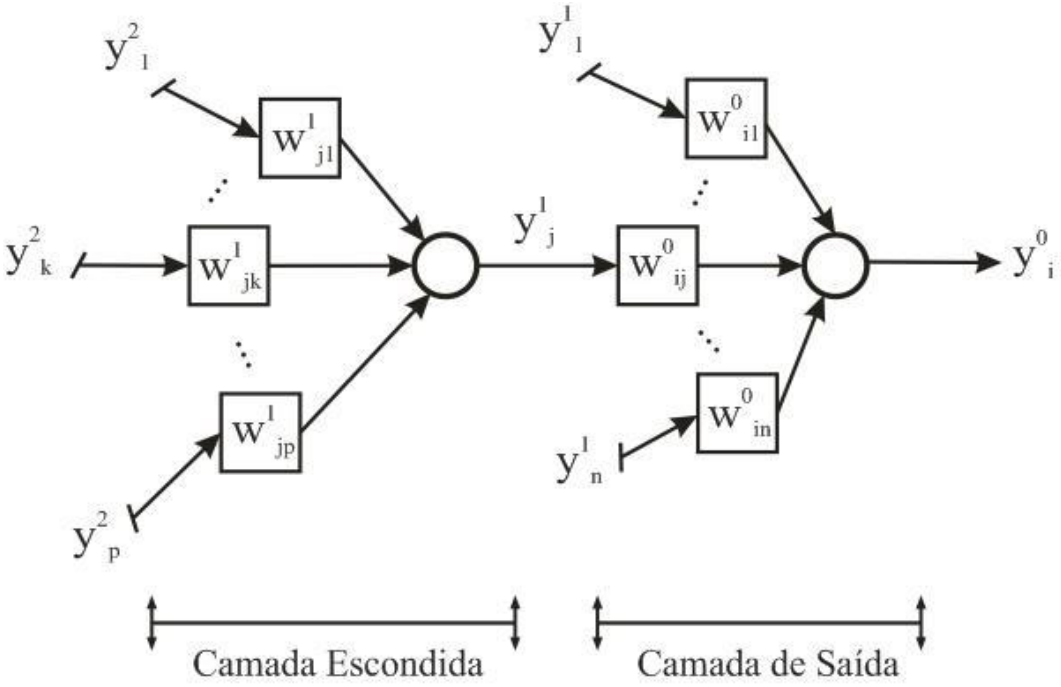


Figura 3.7 - Neurônios das camadas de saída e escondida em uma rede MLP.

O primeiro passo do algoritmo é o cálculo do erro quadrático. Em uma iteração **t** do treinamento, sendo $\mathbf{y^0_i(t)}$ o conjunto de saídas obtido no *forward pass* para o i-ésimo neurônio de saída e $\mathbf{y_{di}(t)}$ o conjunto de saídas esperadas, calcula-se o erro quadrático da saída $\mathbf{E^0_i(t)}$ com a equação (3.8).

$$E^0{}_i(t) = \frac{\left(y_{di}(t) - y^0{}_i(t)\right)^2}{2} \tag{3.8}$$

Com o valor do erro quadrático, pode-se calcular o gradiente usado no ajuste do peso $\mathbf{w^0_{ij}(t)}$ com a equação (3.9), usando também o valor de entrada para este peso $\mathbf{y^{1j}(t)}$**.** A derivada de $\mathbf{y^0_i}$ mostrada na equação é a derivada da função de ativação do neurônio em relação ao somatório das entradas ponderadas pelos pesos.

$$\nabla E^0{}_i(t) = \frac{\partial E^0{}_i(t)}{\partial w^0{}_{ij}(t)} = \left(y_{di}(t) - y^0{}_i(t)\right) \times \left(y^0{}_i(t)\right)' \times y^1{}_j(t) \tag{3.9}$$

Assim, o cálculo do valor atualizado do peso $\mathbf{w^0_{ij}(t+1)}$ pode ser calculado pela equação (3.10), onde $\boldsymbol{\eta}$ é a taxa (ou passo) de aprendizado.

$$w^0{}_{ij}(t+1) = w^0{}_{ij}(t) - \eta \times \nabla E^0{}_i(t) \tag{3.10}$$

Para calcular os pesos dos neurônios da camada escondida, é necessário calcular um novo gradiente a partir do erro quadrático correspondente à essa camada. O cálculo do novo erro quadrático é definido pela equação (3.11).

$$E^1 j(t) = \frac{\left( \sum_{i=1}^{m} E^0{}_i(t) \right)^2}{2} \tag{3.11}$$

O gradiente então é calculado pela equação (3.12).

$$\nabla E^1{}_j(t) = \frac{\partial E^1{}_j(t)}{\partial w^1{}_{jk}(t)} = \left( \sum_{i=1}^{m} E^0{}_i(t) \right) \times \left( y^1{}_j(t) \right)' \times y^2{}_k(t) \tag{3.12}$$

Assim, o novo peso sináptico $\mathbf{w^1_{jk}(t+1)}$ pode ser calculado pela equação (3.13).

$$w^1{}_{jk}(t+1) = w^1{}_{jk}(t) - \eta \times \nabla E^1{}_j(t) \tag{3.13}$$

Uma análise mais completa da matemática envolvida neste algoritmo pode ser obtida na Seção 4.3 do Haykin [1] e também no trabalho de Rumelhart *et al* [15]. O exemplo aqui exposto pode ser aplicado em um número qualquer de camadas, desde que sejam compostas por neurônios com funções de ativação diferenciáveis.

### 3.3.3 Estratégias de treinamento

O treinamento de uma MLP não garante que a rede irá aprender corretamente a resolver o problema, sendo necessários alguns cuidados para elevar a eficiência do algoritmo. Serão apresentadas a seguir algumas estratégias que poderão auxiliar a

construção de soluções usando MLPs.

Uma primeira escolha a se fazer é se o ajuste de pesos da RNA será feito a cada elemento apresentado, no chamado treinamento incremental, ou depois de apresentar toda a massa de dados, no treinamento por lotes. A vantagem do treinamento incremental é o baixo custo de memória utilizado, já que o algoritmo não acumula informações sobre outros elementos além do atual. Se o conjunto de treinamento for grande e redundante, a convergência usualmente é rápida; caso contrário, o algoritmo pode nunca convergir. A estabilidade do algoritmo só é garantida se o passo de aprendizado for pequeno, pois cada elemento que passa pela rede a altera de forma a acertar aquela classe. Assim, se a rede mudar radicalmente de um elemento para outro, nunca será encontrada uma solução genérica para o problema.

O treinamento por lotes usa mais memória, pois antes de alterar a estrutura da rede devem ser apresentados todos os exemplos da massa de treino. Para massas de dados grandes e redundantes, este algoritmo apresenta lentidão, pois pouca informação poderá ser extraída. Em compensação, é muito estável, pois qualquer ajuste levará em conta toda a massa de dados, sendo atribuição da mesma a convergência da curva de erros. Existem soluções híbridas, que treinam a rede com blocos da massa de dados, combinando vantagens e desvantagens de ambas.

O conjunto de dados disponível deve ser equilibrado, apresentando quantidades semelhantes de elementos de cada classe. Dependendo da quantidade de exemplos disponível, deve-se usar de 50% a 70% dessa massa somente para treino e o restante somente para testes. No caso de se usar validação, como neste trabalho, o recomendável é que se use 50% da massa total de dados para treino, 30% para testes e 20% para validação.

A validação é um método em que se testa a rede a cada ciclo de treinamento, sendo usada preferencialmente em treinos por lotes. Depois de ajustados os pesos, a arquitetura da rede é fixada e a massa de dados de validação é passada por ela, testando sua performance. Caso a performance da rede nesses dados piore com os ciclos, indicando a não convergência da curva de erros, pode indicar que a rede está decorando os dados de treino e com isso a generalização do problema está sendo comprometida.

A perda de generalização da rede é chamada *overfitting*. Um dos métodos para ajudar a rede a evitar *overfitting*, além daquele de melhorar o critério de parada do treinamento, é inserir ruído na massa de dados, pois assim o trabalho de memorização dos elementos de treino é dificultado.

A configuração do número de neurônios da rede também influencia no treinamento. Usar poucos neurônios na camada escondida pode causar o *underfitting*, que é a impossibilidade do aprendizado do problema, causando ausência de convergência da curva de erros. Outro problema é a superpopulação de neurônios, que causa o *overfitting*. Assim, o tamanho da camada escondida deve estar em uma faixa de operação entre 10% do número de elementos da massa de treino e metade do número de descritores utilizados em cada exemplo. Como não existe uma metodologia comprovada de arquitetura de RNA, o indicado é treinar várias redes com diferentes números de neurônios na camada escondida, dentro da faixa proposta.

Um problema muito grave que pode ocorrer no treinamento é a saturação dos pesos sinápticos, que impedem o algoritmo de aprendizado de ajustá-los eficientemente. Assim, os pesos devem ser inicializados com valores perto de zero e os descritores da massa de dados devem ser normalizados com algoritmos como o Z-Score e o PCA.

Finalmente, existem vários algoritmos de *Backpropagation* que melhoram a performance do aprendizado da rede ao usar passos de aprendizado dinâmicos e adaptativos, como o *Resilient Backpropagation* (RP) e o *Gradient Descent Backpropagation* (GDA). O uso desses algoritmos, em conjunto com algumas das soluções propostas nessa seção, aumenta as chances da rede chegar a um bom nível de generalização do problema, ficando dependente apenas de que o conjunto de descritores escolhido represente competentemente a massa de dados.

# 4 Metodologia

## 4.1 Introdução

Neste capítulo serão descritos os passos do desenvolvimento do reconhecedor de caracteres de placa veicular, da análise inicial do problema até a configuração das redes neurais, passando pela preparação da massa de dados e pela estrutura dos ensaios.

## 4.2 Análise do problema

### 4.2.1 Separação em letras e números

Um OCR de caracteres de placa de automóveis deve identificar, através da análise computacional da imagem apresentada, a qual das 36 classes de caracteres (10 números e 26 letras) a mesma pertence.

O problema apresentado pode ser simplificado ao separar as classes em letras e números, evitando confusões como diferenciar o caractere relativo ao número “zero” daquele que representa a letra “O”. Isso também auxilia no treinamento da RNA, pois um problema com 36 classes a serem relacionadas é mais complexo do que um com 10 ou 26 classes.

Além disso, muitos países adotam placas de automóveis com posições fixas para letras e números. Por exemplo, no Brasil os caracteres das placas são três letras seguidas por quatro números enquanto na Argentina e no Paraguai são três letras seguidas por três números. Nesse caso, não é necessário descobrir à qual classe de caracteres a imagem pertence, pois esta já chega rotulada como letra ou número. Sendo assim, a imagem é direcionada a uma RNA especificamente treinada para letras ou números.

### 4.2.2 Estrutura dos ensaios

O projeto do OCR é estruturado em ensaios, combinando o tipo de extração de descritores e o tipo de caracteres (letras ou números). Estes ensaios foram estruturados

em pastas e desenvolvidos a partir de scripts em *Matlab*. A estrutura de pastas do projeto está mostrada na figura 4.1:


Figura 4.1 - Estrutura de pastas dos ensaios.

A pasta **Comuns** armazena os scripts de *Matlab* comuns a todos os ensaios, como os algoritmos de separação de dados e os de criação, treino e teste de redes neurais. Estes scripts serão mais detalhados na próxima seção, e seus códigos estarão disponíveis no Capítulo 8 (Apêndices).

A pasta **Dados** contém os arquivos **.mat**, formato nativo do *Matlab* para armazenagem de variáveis, relativos às imagens usadas nos ensaios, além dos bancos de descritores usado em cada um destes. Mais detalhes sobre as estruturas de dados usadas no projeto estão na Seção 4.3.

Na pasta **Ensaios** estão duas pastas: **MapaDeBits** e **Momentos**. O nome das pastas é relacionado com o tipo de extração de descritores usado pelos ensaios nelas contidos. Cada pasta contém mais duas pastas: **Letras** e **Dígitos**, contendo respectivamente os ensaios de letras e números. Nestas pastas estão *scripts* de configuração e os dados gerados durante a execução dos ensaios: alguns arquivos **.mat** relativos aos treinamentos e um arquivo **.xls** de *Excel* contendo as performances dos treinamentos e as matrizes de confusão extraídas das cinco melhores RNAs treinadas.

### 4.2.3 Desenvolvimento dos scripts de Matlab

Os scripts de *Matlab* desenvolvidos para os ensaios têm como principal característica serem autônomos o máximo possível do tempo, salvando sempre que possível as variáveis mais importantes de cada etapa nas pastas do ensaio. Assim, pode-

se reiniciar o ensaio do ponto em que foi parado sem perdas de resultados, que podem ser causadas por erros no sistema operacional, no *Matlab* ou qualquer outro fator externo. Os scripts podem ser classificados como tratamento de dados, construção de RNAs, de suporte e de configuração.

Os scripts de tratamento de dados contêm funções que vão desde a separação das imagens do banco fornecido até a extração e pré-processamento dos descritores das mesmas imagens, estando todos eles armazenados na pasta **Comuns**. São eles: **SepararBancoImagens**, **ExtrairBancoDescritores**, **ExtrairMapaDeBits** e **ExtrairMomentos**. Os dois últimos são específicos para os tipos de extração de descritores de cada ensaio, e são chamados pelo **ExtrairBancoDescritores**. Mais adiante serão abordados os detalhes destas funções, e seus códigos serão fornecidos separadamente junto com o material digital desta monografia.

Os scripts de construção de RNAs são responsáveis pela criação, treinamento e teste das redes neurais do projeto. São eles: **CriarRedes**, **TreinarRedes** e **TestarRedes**. O **CriarRedes** cria o arquivo **redes_criadas.mat**, contendo, respectivamente, todas as redes antes dos treinos e as já treinadas. O **TreinarRedes**, como o nome já diz, treina todas as redes criadas com as massas de dados separadas, salvando no arquivo **redes_treinadas.mat**. O script **TestarRedes** testa as redes e salva os melhores resultados nos arquivos **melhor_rede.mat** e em um arquivo de *Excel* nomeado dinamicamente de acordo com o ensaio ao qual pertence.

Os scripts de suporte são funções criadas para facilitar alguns processamentos. São eles: **imoments**, **pca**, **Char2Index** e **Index2Char**. O primeiro é proveniente do toolbox de *Matlab* **Machine Vision** [6], usado para calcular os momentos bidimensionais de uma dada imagem; foram feitas algumas modificações para calcular momentos invariantes à escala e à rotação. O script **pca** também vem de um toolbox, a **Dimension Reduction Toolbox** [7], e é usado para descorrelacionar os valores e também para reduzir a dimensão do vetor de descritores da imagem. Os scripts **Char2Index** e **Index2Char** são, respectivamente, para traduzir um caractere para seu índice (por exemplo, para a entrada sendo o caractere “A”, a saída será 1) e para fazer o caminho inverso; foram criados com o objetivo de facilitar essa conversão de representações para as classes.

Finalmente, temos os scripts de configuração, os únicos que são replicados dentro das pastas dos diferentes ensaios. O único representante dessa classe de script é o **Config**, que salva uma estrutura de nome **cfg** no arquivo **config_vars.mat**, contendo

todas as características que são diferentes em cada ensaio e também algumas que foram deliberadamente colocadas iguais, como o endereço do diretório contendo os dados de imagens e as configurações de RNAs. Existem três grupos de variáveis de configuração nesta estrutura: relativas aos dados das imagens, à extração de descritores e à arquitetura da RNA. As variáveis de configuração relativas aos dados das imagens serão comentadas na seção de preparação da massa de dados, enquanto as relativas à extração de descritores serão tratadas cada uma na seção correspondente ao tipo de abordagem de extração do ensaio. As variáveis de configuração relativas à arquitetura da RNA serão tratadas na Seção 4.4.

## 4.3 Preparação da massa de dados

### 4.3.1 O banco de imagens

Os dados utilizados neste projeto são provenientes de um banco de imagens do sistema Kapta, que possui alguns milhares de imagens de caracteres provenientes de onze locais diferentes. As tabelas 4.1, 4.2 e 4.3 mostram os locais de origem e a distribuição de imagens de caracteres por classe e local. Cada classe de caractere possui por volta de 600 imagens.

| **Identificador** | **Nome do Local de Origem** |
|---|---|
| L1 | Acesso Estacionamento (externo) SP |
| L2 | Avenida SP |
| L3 | Acesso Estacionamento (coberto) RJ 01 |
| L4 | Acesso Estacionamento (coberto) RJ 02 |
| L5 | Acesso UFRJ - Hospital |
| L6 | Acesso UFRJ - Linha Amarela |
| L7 | Acesso UFRJ - Prefeitura |
| L8 | Linha Vermelha (noturno) |
| L9 | Linha Vermelha (diurno) |
| L10 | Acesso Condomínio RJ 01 |
| L11 | Acesso Condomínio RJ 02 |

Tabela 4.1 - Locais de origem e seus identificadores.

| Local / Classe | L1 | L2 | L3 | L4 | L5 | L6 | L7 | L8 | L9 | L10 | L11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 52 | 51 | 50 | 54 | 52 | 61 | 79 | 53 | 50 | 52 | 50 |
| **1** | 55 | 55 | 54 | 55 | 55 | 59 | 60 | 54 | 54 | 54 | 55 |
| **2** | 51 | 49 | 50 | 54 | 51 | 68 | 86 | 50 | 49 | 51 | 50 |
| **3** | 51 | 48 | 48 | 49 | 51 | 63 | 102 | 48 | 48 | 48 | 48 |
| **4** | 52 | 55 | 52 | 54 | 52 | 59 | 74 | 52 | 52 | 54 | 52 |
| **5** | 50 | 51 | 49 | 52 | 50 | 72 | 85 | 50 | 49 | 49 | 49 |
| **6** | 43 | 46 | 42 | 44 | 42 | 79 | 135 | 44 | 42 | 42 | 46 |
| **7** | 54 | 52 | 51 | 55 | 52 | 55 | 79 | 51 | 51 | 53 | 53 |
| **8** | 36 | 43 | 42 | 36 | 36 | 69 | 184 | 40 | 37 | 40 | 37 |
| **9** | 54 | 57 | 49 | 48 | 46 | 69 | 88 | 47 | 50 | 49 | 48 |

Tabela 4.2 - Distribuição de imagens de números por classe e local de origem.

| Local / Classe | L1 | L2 | L3 | L4 | L5 | L6 | L7 | L8 | L9 | L10 | L11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **A** | 111 | 25 | 22 | 61 | 101 | 35 | 103 | 60 | 22 | 19 | 18 |
| **B** | 103 | 35 | 62 | 52 | 59 | 38 | 76 | 57 | 24 | 42 | 31 |
| **C** | 74 | 61 | 63 | 59 | 62 | 31 | 62 | 60 | 27 | 30 | 51 |
| **D** | 138 | 19 | 100 | 39 | 88 | 17 | 104 | 19 | 21 | 7 | 23 |
| **E** | 158 | 31 | 42 | 34 | 148 | 18 | 100 | 17 | 13 | 3 | 12 |
| **F** | 281 | 11 | 19 | 16 | 111 | 3 | 93 | 22 | 11 | 8 | 0 |
| **G** | 152 | 14 | 22 | 41 | 108 | 19 | 141 | 35 | 9 | 13 | 22 |
| **H** | 182 | 12 | 36 | 18 | 85 | 21 | 155 | 35 | 14 | 7 | 10 |
| **I** | 229 | 13 | 37 | 12 | 76 | 17 | 122 | 41 | 10 | 5 | 13 |
| **J** | 114 | 14 | 24 | 40 | 110 | 20 | 111 | 47 | 23 | 10 | 62 |
| **K** | 120 | 44 | 26 | 43 | 62 | 43 | 79 | 46 | 44 | 27 | 44 |
| **L** | 62 | 54 | 21 | 54 | 62 | 52 | 66 | 53 | 52 | 53 | 52 |
| **M** | 133 | 14 | 27 | 33 | 76 | 40 | 94 | 72 | 33 | 16 | 38 |
| **N** | 108 | 62 | 18 | 55 | 58 | 45 | 70 | 59 | 37 | 16 | 47 |
| **O** | 135 | 43 | 28 | 42 | 50 | 42 | 76 | 41 | 39 | 41 | 42 |
| **P** | 115 | 30 | 22 | 82 | 89 | 32 | 96 | 43 | 15 | 21 | 31 |
| **Q** | 245 | 28 | 18 | 28 | 31 | 28 | 110 | 26 | 18 | 22 | 22 |
| **R** | 144 | 13 | 11 | 42 | 116 | 23 | 129 | 58 | 13 | 4 | 24 |
| **S** | 105 | 22 | 15 | 51 | 96 | 31 | 104 | 70 | 14 | 5 | 63 |
| **T** | 126 | 37 | 14 | 60 | 117 | 36 | 126 | 32 | 12 | 11 | 6 |
| **U** | 91 | 27 | 22 | 74 | 74 | 46 | 81 | 73 | 41 | 12 | 37 |
| **V** | 123 | 23 | 7 | 46 | 110 | 46 | 113 | 39 | 22 | 41 | 5 |
| **W** | 169 | 11 | 12 | 53 | 86 | 16 | 141 | 37 | 19 | 8 | 23 |
| **X** | 130 | 17 | 49 | 47 | 115 | 12 | 127 | 49 | 17 | 3 | 11 |
| **Y** | 131 | 8 | 13 | 48 | 112 | 38 | 128 | 62 | 23 | 8 | 4 |
| **Z** | 99 | 14 | 15 | 81 | 83 | 51 | 91 | 81 | 27 | 10 | 26 |

Tabela 4.3 - Distribuição de imagens de letras por classe e local de origem.

A distribuição de imagens por classe e local de origem pode ser vista na forma de gráfico de barras mostrado na figura 4.2.

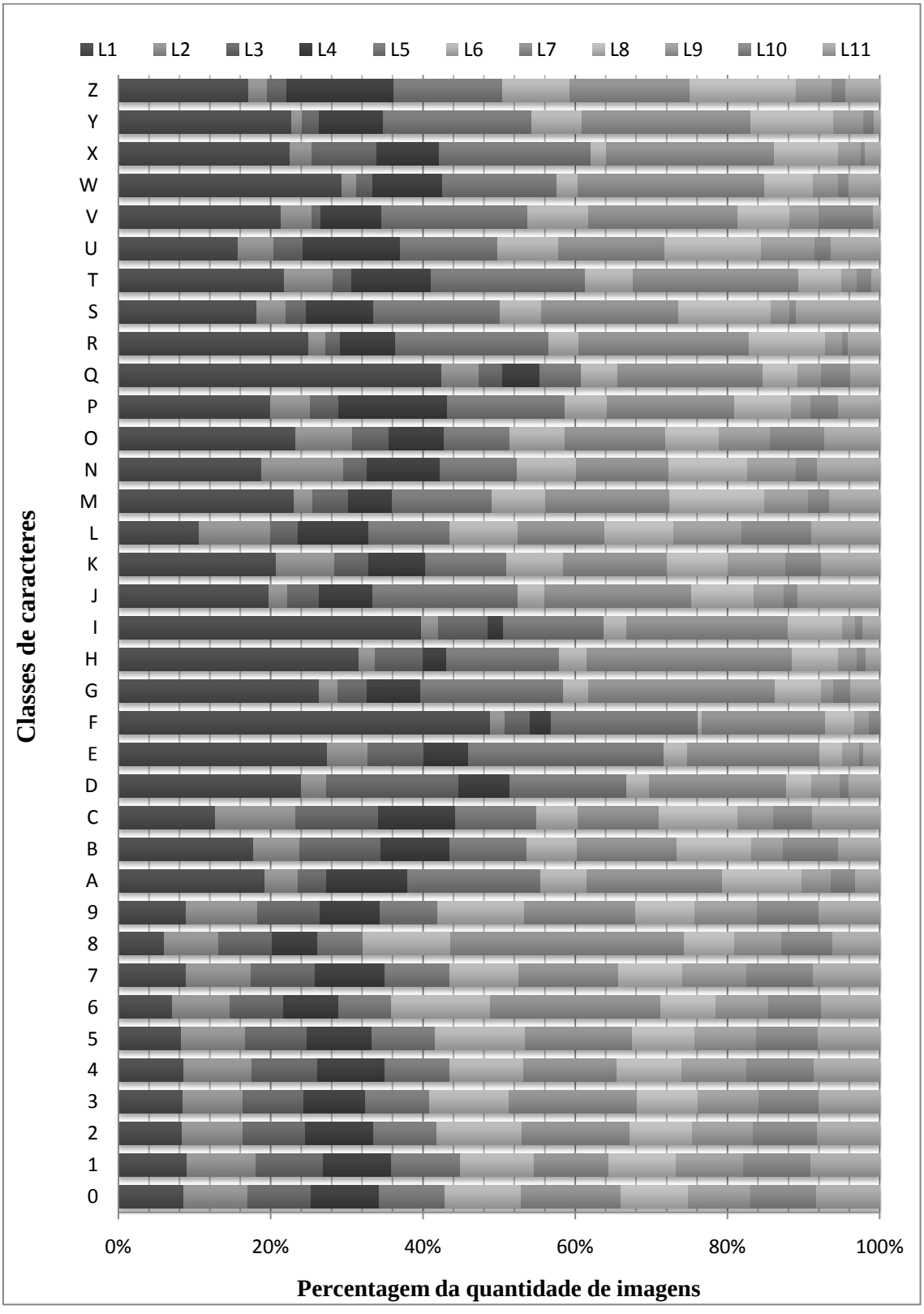


Figura 4.2 - Distribuição de imagens por classe e local de origem.

O banco de imagens do sistema Kapta usado em todos os ensaios foi construído a partir de um conjunto grande de imagens pré-processadas, salvas em dois arquivos: **Letras_Bmps.mat** e **Digitos_Bmps.mat**. Cada arquivo possui uma matriz de células, que é um tipo padrão de estrutura de dados do *Matlab* que permite diversos tipos de variáveis, sendo o número de linhas igual ao de locais de origem e o número colunas igual ao de classes de caracteres. Assim, as dimensões da matriz de células em **Letras_Bmps.mat** é 11x26, enquanto em **Digitos_Bmps.mat** é 11x10. A matriz de células é armazenada em uma variável padrão de nome **bmps_matrix**; cada célula da matriz contém um vetor de variáveis do tipo *struct*, contendo cada uma informações sobre a imagem da qual foi extraída.

O comprimento de cada vetor é igual ao número nos campos das tabelas mostradas nessa seção; por exemplo, a célula de coordenada (1,1) na **bmps_matrix** contida no arquivo **Letras_Bmps.mat**, referente à classe de caractere "A" de local de origem "L1", contém um vetor com 111 *structs* referentes às imagens de caracteres da mesma classe e do mesmo local.

Cada *struct* possui quatro variáveis internas: **bmp_data**, **target_char**, **target** e **local**. A variável **bmp_data** é uma matriz de dimensões 16x16, que representa uma imagem em tons de cinza; dessa forma, cada pixel de informação dessa imagem é representado por um valor, relativo à sua luminância, entre 0 e 255. As variáveis **target_char** e **target** representam, respectivamente, o caractere e o índice da classe à qual pertence a imagem. O local de origem é definido pelo índice contido na variável **local**.

### 4.3.2 Separando as massas de treino, teste e validação

Para que a avaliação das soluções propostas de OCR seja coerente, deve-se partir de uma massa de dados idêntica para cada ensaio. Assim, a mesma massa de dados para o ensaio **MapaDeBits/Letras** será usada para o ensaio **Momentos/Letras**, da mesma forma que os ensaios **MapaDeBits/Digitos** e **Momentos/Digitos** compartilham o mesmo conjunto de dados.

O projeto das RNAs requisita três massas distintas de dados: massa de treino, de teste e de validação. A proporção utilizada em todo o projeto é de 50% dos dados para treino, 20% para validação e os 30% restantes para teste. Essa proporção é respeitada

para cada dupla classe de caractere e local de origem da matriz de dados **bmps_matrix**, sendo que cada classe dos vetores de treino, teste e validação será composta pela percentagem especificada de imagens em cada local de origem. Por exemplo, a massa de treino da classe "A" será composta por 50% das imagens do local "L1", mais 50% das imagens do local "L12" e assim por diante, totalizando cerca de 300 imagens.

Usando o script de *Matlab* **SepararBancoImagens** com as entradas sendo a variável **bmps_matrix** e um vetor de percentagens **perf** igual a [0.5, 0.2, 0.3], temos como saída a *struct* **bmps_puros**, contendo as três variáveis **treino**, **testes** e **valida**, cada qual contendo um vetor com as imagens utilizadas, respectivamente, para treino, teste e validação.

A etapa final do script mistura aleatoriamente, dentro de cada vetor, a ordem das imagens; isto é necessário para que, durante o treinamento da RNA, a ordem de apresentação das imagens da massa de treino não afete o desempenho desta etapa. As variáveis **perf** e **bmps_puros** são então salvas nos arquivos de onde se carregou a variável **bmps_matrix**: **Letras_Bmps.mat** ou **Digitos_Bmps.mat**. As dimensões das massas de dados salvas em **bmps_puros** estão mostradas na tabela 4.4.

| **Tipo de massa de dados** | **Treino** | **Validação** | **Teste** |
|---|---|---|---|
| **Percentagem** | 50% | 20% | 30% |
| **Dígitos (Total)** | 3.052 | 1.211 | 1.794 |
| **Dígitos (Média por classe)** | 305,20 | 121,10 | 179,40 |
| **Letras (Total)** | 7.566 | 2.992 | 4.434 |
| **Letras (Média por classe)** | 291,00 | 115,08 | 170,54 |

Tabela 4.4 - Distribuição de imagens por tipo de massa de dados.

### 4.3.3 Construção do banco de descritores

A construção do banco de descritores se dá em três etapas: extração de descritores, normalização do banco extraído e aplicação do algoritmo PCA no banco normalizado. O script usado nesta etapa é o **ExtrairBancoDescritores**, que recebe como entrada o banco de imagens e o arquivo de configuração de cada ensaio, gerando como saída o banco de descritores.

Na extração dos descritores, cada imagem do banco de imagens é processada com o algoritmo específico do ensaio, gerando um vetor coluna de números reais (os descritores). Assim, ao final da extração dos descritores, serão criadas três matrizes,

relativas às massas de treino, teste e validação, com número de linhas igual ao número de descritores (cada ensaio tem seu próprio número de descritores) e número de colunas igual à quantidade de imagens.

Nessa mesma etapa, as saídas esperadas usadas no treinamento (também conhecidas como *targets*) são extraídas a cada análise de componente do banco de imagens. Para representar a saída da rede neural, foi usada a metodologia proposta por Simon Haykin [1] na Seção 4.8 de seu livro: uma representação estatística da probabilidade de um elemento analisado pela RNA pertencer às classes do problema. Usa-se para isso um número de saídas igual ao número de classes, cada uma podendo apresentar valores reais entre 0 e 1. Os valores entre 0 e 1 representam a probabilidade do elemento que gerou a saída pertencer às classes em cada posição. As saídas esperadas ficam no formato de um vetor coluna contendo zeros em todas as posições a não ser na posição relativa à classe real do elemento, ou seja, 100% de probabilidade em ser da classe real e 0% de ser das outras. Assim, são geradas mais três matrizes relativas a treino, teste e validação, com o número de linhas igual ao número de classes (10 linhas para números e 26 para letras) e número de colunas igual ao número de elementos da massa de dados. Na tabela 4.5 são mostrados os descritores e saídas esperadas dos cinco primeiros elementos das imagens de caracteres mostradas na figura 4.3, provenientes da massa de treino, usando o ensaio do tipo Momentos.

| **Vetores de Descritores** | | | | | |
|---|---|---|---|---|---|
| **Posição no vetor** | **Treino(1)** | **Treino(2)** | **Treino(3)** | **Treino(4)** | **Treino(5)** |
| **1** | 1,89 | 3,37 | 2,86 | 2,40 | 2,23 |
| **2** | 2,35 | 5,16 | 4,00 | 2,79 | 2,27 |
| **3** | 2,09 | 3,28 | 4,06 | 3,24 | 2,35 |
| **4** | 1,45 | 1,31 | 1,58 | 2,27 | 1,79 |
| **5** | 4,16 | 9,11 | 11,10 | 10,72 | 6,49 |
| **6** | 9,78 | 47,01 | 44,38 | 29,89 | 14,75 |
| **7** | 7,85 | 30,67 | 31,78 | 25,78 | 14,50 |
| **8** | 17,76 | 153,44 | 124,22 | 71,27 | 31,11 |
| **9** | 27,30 | 265,76 | 222,93 | 111,49 | 40,20 |
| **…** | … | … | … | … | … |
| **83** | 0,88 | 7,73 | 7,68 | 2,67 | 1,91 |
| **84** | 10,35 | 30,75 | 64,54 | 31,93 | 3,14 |

Tabela 4.5 - Amostra de descritores das imagens de caracteres da Figura 4.3.

| Vetores de Targets | | | | | | |
|---|---|---|---|---|---|---|
| **Posição** | **Classe** | **Treino(1)** | **Treino(2)** | **Treino(3)** | **Treino(4)** | **Treino(5)** |
| **1** | **"0"** | 0 | 0 | 0 | 0 | 0 |
| **2** | **"1"** | 0 | 0 | 0 | 0 | 0 |
| **3** | **"2"** | 0 | 0 | 0 | 0 | 0 |
| **4** | **"3"** | 0 | 0 | 0 | **1** | 0 |
| **5** | **"4"** | 0 | 0 | 0 | 0 | 0 |
| **6** | **"5"** | 0 | 0 | 0 | 0 | 0 |
| **7** | **"6"** | 0 | **1** | **1** | 0 | 0 |
| **8** | **"7"** | **1** | 0 | 0 | 0 | **1** |
| **9** | **"8"** | 0 | 0 | 0 | 0 | 0 |
| **10** | **"9"** | 0 | 0 | 0 | 0 | 0 |

Tabela 4.6 - Amostra de saídas esperadas das imagens de caracteres da Figura 4.3.

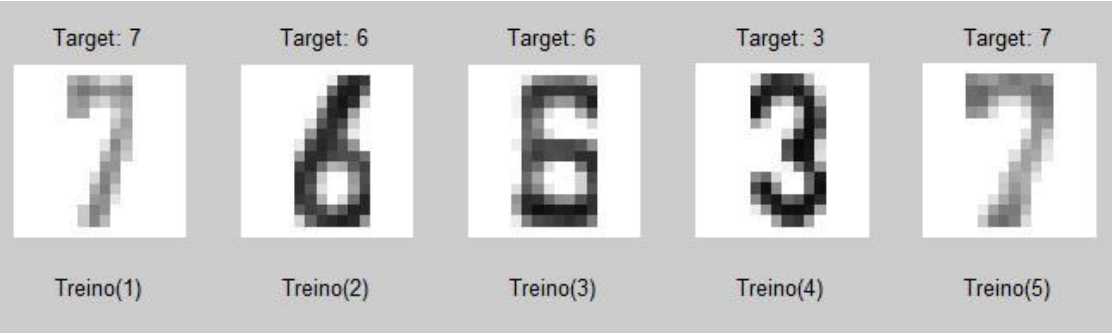


Figura 4.3 - Cinco primeiras imagens da massa de treino de números.

A etapa a seguir é a normalização da massa de descritores. O processo de normalização da massa de dados que será analisada pela rede neural é fortemente recomendado por Simon Haykin [1] na Seção 4.6. Esta etapa é de extrema importância, pois ela dimensiona os dados de entrada da RNA de forma que esta não tenha problemas com *outliers* (pontos fora da curva), ou descritores cujos valores são muito diferentes dos outros, que influenciam negativamente (podendo inclusive impedir) o treinamento da RNA.

A normalização utilizada neste projeto é a chamada Z-Score, ou Standard-Score. O Z-Score aplicado a uma massa de dados tem por objetivo zerar a média e tornar o desvio padrão igual a 1. Primeiro são calculados a média ($\mu$) e o desvio padrão ($\sigma$) da massa (X), para depois aplicar a equação (4.1), obtendo uma massa de dados normalizada (Z). A equação (4.1) está mostrada a seguir.

$$Z = \frac{X - \mu}{\sigma} \tag{4.1}$$

Assim, a massa de treino é normalizada usando a função *prestd* do *Matlab*. Essa função faz a normalização Z-Score em cada linha da massa de treino e calcula os vetores coluna de média e desvio padrão, ambos com número de elementos igual ao número de descritores. Cada valor destes vetores representa, respectivamente, a média e o desvio padrão de todos os elementos da linha correspondente da matriz de descritores da massa de treino. Os vetores são então usados para normalizar as massas de validação e de teste, aplicando a equação (4.1).

Após esta etapa, aplica-se o algoritmo PCA (*Principal Componente Analysis*), descrito com detalhes no Capítulo 8 do livro de Haykin [1], usando a função de *Matlab* **pca** criada por Peyré [7] em sua *Dimensional Reduction Toolbox*, que permite um controle mais acurado do tamanho final do vetor de descritores. O PCA tem como objetivo descorrelacionar e reduzir a dimensão do vetor de descritores, embora nem sempre esta redução seja necessária ou até possível sem perdas nos treinos.

Aplicando a função **pca** de Peyré [7] sobre a massa de treino normalizada, obtém-se a nova massa de treino e a matriz de transformação gerada pelo algoritmo, **transmat**, que deve ser aplicada sobre as massas de teste e validação normalizadas ao se multiplicar cada uma pela **transmat**.

Assim, cada ensaio define cinco tamanhos de entrada de dados a serem usados na função **pca**, gerando ao final do processo de separação do banco de descritores três massas de dados (treino, teste e validação) representadas cada uma por cinco matrizes de dimensões diferentes (na dimensão das linhas, ou número de descritores). Os tamanhos dos vetores de descritores são calculados pela equação (4.2), onde ***T*** é o vetor de tamanhos após o PCA e ***N*** é o tamanho original do vetor de descritores.

$$\overline{T} = \left\lceil N; \frac{9}{10}N; \frac{8}{10}N; \frac{7}{10}N; \frac{6}{10}N \right\rceil \tag{4.2}$$

Assim, do banco de descritores puros da Tabela 4.5 seriam gerados cinco bancos processados com PCA, cujos vetores de descritores teriam tamanhos 84, 76, 68, 59 e 51. O primeiro vetor tem exatamente as mesmas dimensões do banco puro; o que mudou foi que agora os descritores estão em uma base ortogonal e com mesma dimensão, estando,

portanto, descorrelacionados. Esse tipo de uso do PCA garante que não haverão perdas de informação após o processo, o que pode acontecer ao se eliminar dimensões de uma massa de dados.

O script **ExtrairBancoDescritores** salva um arquivo **.mat** nomeado de acordo com o ensaio (por exemplo, o ensaio Momentos com letras tem o nome **Banco_Momentos_Letras_bmps.mat**) contendo todos os bancos extraídos. O banco puro de descritores fica armazenado na variável **banco_raw**, que contém três matrizes: **treino**, **testes** e **valida**. O banco normalizado é armazenado em **banco_norm**, que contém, além das três matrizes de descritores normalizados, os vetores de média **meanp** e de desvio padrão **stdp**. O banco de descritores processados com PCA fica em **banco_pca**; as matrizes de treino, teste e validação têm o mesmo nome que nos outros bancos, mas cada uma armazena cinco matrizes com os descritores nos diferentes tamanhos que a função **pca** gera. Além dessas variáveis, também armazena na variável **transmat** cinco matrizes de transformação para produzir dados a partir de uma matriz de descritores normalizada.

## 4.4 Construção das RNAs

### 4.4.1 Estrutura geral das RNAs nos ensaios

A etapa de treinamento das RNAs foi construída de forma a avaliar o desempenho dos ensaios sob os seguintes aspectos: tipo de extração dos descritores, redução do número de descritores por PCA e tamanho da camada escondida da RNA. Essas características são únicas para cada ensaio; todas as outras configurações das RNAs são idênticas em quaisquer circunstâncias, para que não sejam levadas em conta na hora de avaliar as performances dos ensaios.

Baseado nisso, para cada ensaio foram criadas cinqüenta redes neurais, na forma de uma matriz de RNAs de tamanho 5x10. As cinco linhas são grupos de RNAs que compartilham, em cada linha, o mesmo número de entradas, baseado no tamanho do vetor de descritores processado com PCA que será usado no grupo. As dez colunas são relativas ao número de neurônios na camada escondida, que é compartilhado por todas as RNAs de cada coluna.

A configuração do número de neurônios na camada escondida é feita de acordo

com cada ensaio baseado no tamanho da entrada (ainda sem ter sido reduzida pelo PCA) e da saída relativos às RNAs. A primeira coluna de RNAs possui tamanho da camada escondida igual à média aritmética do número de entradas com o número de saídas; na segunda coluna, esse número é aumentado em dez neurônios, e assim por diante até que a décima coluna tenha noventa neurônios a mais que a primeira.

### 4.4.2 Arquitetura comum das RNAs

As configurações gerais das RNAs estão no arquivo de configuração de cada ensaio, para automatizar o cálculo das dimensões explicadas anteriormente, mas fora isso são as mesmas para todos os ensaios. Na tabela 4.7 estão os valores das configurações das RNAs, que serão explicados em seguida.

| Tipo Param. | Parâmetro | Valor | Observação |
|---|---|---|---|
| **Arquitetura** | ***ativacao*** | *tansig (cmd 1)*<br>*logsig (cmd 2)* | Funções de ativação das camadas escondida e de saída. |
| | ***tam_saida*** | *26 (Letras)*<br>*10 (Dígitos)* | Número de neurônios da camada de saída |
| | ***tipos_redes*** | *N,*<br>*N+10,*<br>*N+20,*<br>*...*<br>*N+90* | Vetor com os números de neurônios da camada escondida; **N** é a média aritmética do número de entradas com o número de saídas da RNA. |
| **Treinamento** | ***performFcn*** | *sse* | Função que calcula o erro relativo à toda a masa de dados em uma época. |
| | ***trainFcn*** | *trainrp* | Algoritmo de treinamento *backpropagation* da RNA usado no ensaio. |
| **Condições de parada do treino** | ***epochs*** | *1000* | Número máximo de ciclos ou épocas que a rede irá treinar. |
| | ***goal*** | *0.01* | Erro mínimo calculado sobre uma época com a função **performFcn** que, se atingido, pára o treinamento. |
| | ***max_fail*** | *50* | Número máximo de épocas que será permitida a curva de validação subir. |

Tabela 4.7 - Parâmetros de configuração comuns às RNAs de todos os ensaios.

Na configuração da arquitetura das redes, já foi visto anteriormente como são

dimensionadas as camadas escondida e de saída. As funções de ativação usadas em cada uma são a função de tangente hiperbólica, ou *tangente -sigmóide* (no *Matlab* nomeada **tansig**), na camada escondida, e a *logística-sigmóide* (no *Matlab* nomeada **logsig**) na camada de saída.

A função **tansig** é preferida por sua característica anti-simétrica, que segundo Haykin [1], nas seções 4.6 e 4.11, torna o treinamento mais veloz em termos de números de iterações necessários para convergência. Essa função, de característica não-linear, pode ter valores entre -1 e 1, e é representada pela equação (4.3) e pela figura 4.4:

$$a(n) = \frac{2}{1+e^{-2n}} - 1 \tag{4.3}$$

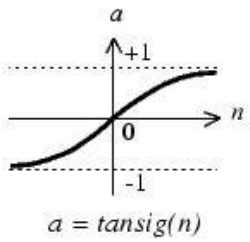


Figura 4.4 - Gráfico da função *tangente sigmóide*.

A função **logsig** usada na camada de saída tem valores entre 0 e 1, e é usada aqui pela característica estatística das saídas esperadas (os *targets*), como já foi explicado anteriormente. A equação e gráfico da **logsig** estão representados pela equação (4.4) e pela figura 4.5:

$$a(n) = \frac{1}{1+e^{-n}} \tag{4.4}$$

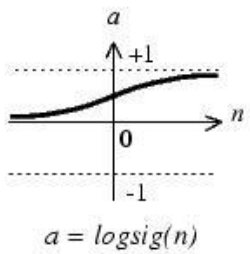


Figura 4.5 - Gráfico da função *sigmoidal*.

O treinamento das RNAs foi definido em lotes, ou *batch*. Outra recomendação de Haykin [1] é que esse tipo de treinamento use toda a massa de treino para calcular o erro que será utilizado para corrigir os pesos da rede. Neste projeto foi usado o algoritmo SSE (*Sum Squared Error*, ou soma dos quadrados) para calcular este valor.

A função de treinamento usada nas RNAs, que é o algoritmo que ajusta os pesos da rede a cada época, foi a **trainrp** (*Resilient Backpropagation*). Esta função usa passos diferentes de acordo com a direção de crescimento da curva de erro: se a curva apresentar uma direção constante, o passo de aprendizado aumenta um valor de incremento, caso contrário ele diminui outro valor de decremento. Este algoritmo de treinamento apresenta bons índices de velocidade e convergência, sendo indicado para problemas de classificadores com uma grande massa de dados. Mais detalhes sobre este e outros algoritmos usados neste projeto podem ser encontrados em *Matlab Neural Networks Toolbox* $^{TM}$ *User's Guide* [8].

As condições de parada do treinamento de uma RNA devem ser definidas de forma que não haja vício na massa de treino, também chamado *overfitting*, onde a rede treinada se torna uma especialista nos dados usados no treino, atingindo performances altíssimas, enquanto para dados fora dessa massa ela apresente um desmpenho muito inferior. Assim, foram configuradas três condições de parada: por épocas, por erro mínimo atingido e por validação.

A parada por épocas é a mais comum de todas: caso o número de iterações de treinamento chegue ao limite configurado, a rede pára de treinar. Dependendo de como estiver o treinamento, pode ser vantajoso para a rede continuar treinando até atingir o erro mínimo, assim como pode ficar muito tempo treinando sem que haja algum ganho de performance; portanto o número de épocas deve ser balanceado para que a rede treine rapidamente e ainda assim atinja a melhor performance possível.

A outra condição, quando o erro mínimo é atingido, seria a condição ideal, pois indica que a performance máxima da rede para esta configuração foi atingida. De qualquer forma, isso não garante que a rede tenha aprendido, ou generalizado, o problema e sim que ela decorou a massa de treino, indicando *overfitting*. A metodologia usada no projeto coloca um erro mínimo praticamente inatingível, bem próximo de zero, para que a rede tenha a chance de parar o treinamento por validação, e ainda em um melhor caso, atingir uma performance de praticamente cem por cento na massa de treino.

A parada por validação usa uma massa de dados auxiliar para testar a rede a

cada época, sem influenciar no ajuste dos pesos dos neurônios. O funcionamento ocorre da seguinte maneira: a cada época, toda a massa de treino é passada pela RNA, calculando o erro com o algoritmo escolhido (neste projeto, o SSE) e ajustando com este valor os pesos dos neurônios com o algoritmo *backpropagation* (**trainrp** no projeto). Depois disso, a rede é congelada, isto é, os pesos dos neurônios não podem mais ser atualizados até a próxima época. Passa-se então pela rede a massa de validação e, com o mesmo algoritmo de cálculo de erro, encontra-se o valor da performance da rede para a massa que acabou de passar. Este valor é comparado com todos os valores calculados nas épocas anteriores para a mesma massa de validação; caso o valor seja maior do que os anteriores, a contagem de erros é incrementada. Quando a contagem de erros atinge um limite pré-estabelecido (na variável **max_fail**), a rede pára de treinar.

Na figura 4.6, pode-se verificar a janela do gráfico de treinamento de uma das redes do projeto.

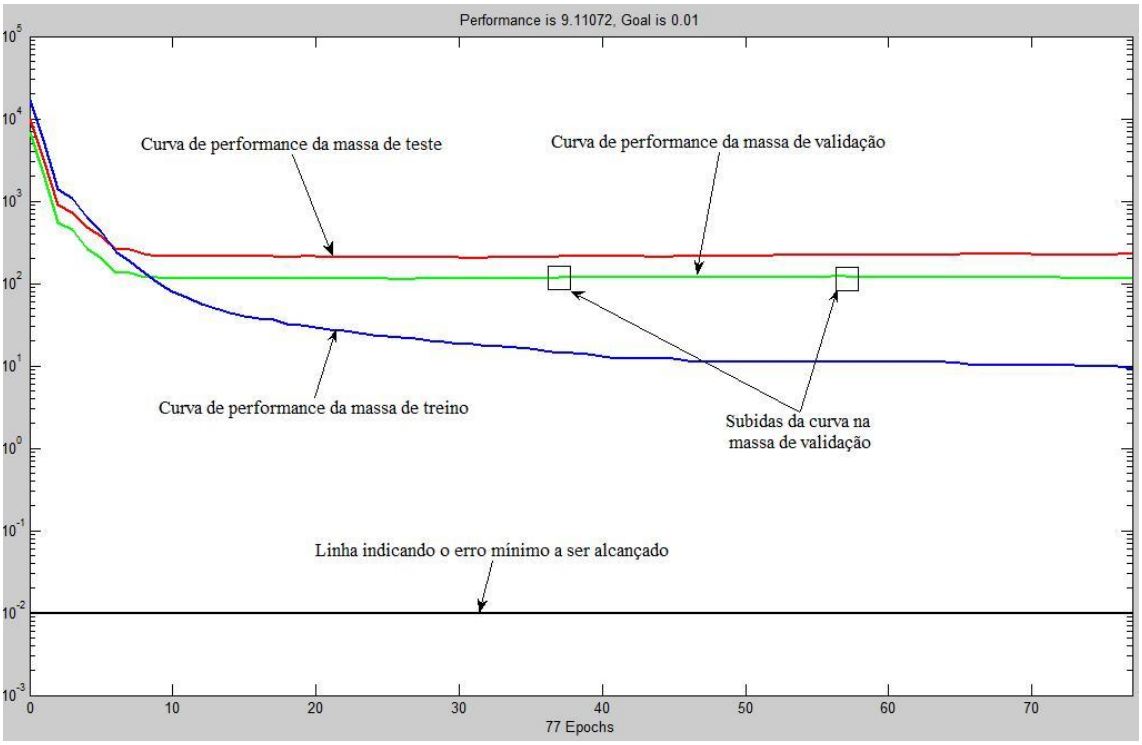


Figura 4.6 - Gráfico do treinamento de uma RNA.

Nota-se que a curva de performance da massa de treino sequer chegou perto do erro mínimo, e que o número de épocas também não chegou ao seu limite. A curva relativa à massa de validação apresenta algumas subidas, ou mudanças de sinal em relação aos valores anteriores; essas subidas são contabilizadas como erro e atingiram o

valor de **max_fail**, parando a rede por validação.

### 4.4.3 Metodologias de avaliação das RNAs

Após o treinamento de todas as redes em um ensaio, usa-se a massa de testes e de validação para testar as performances de cada RNA treinada. O uso das duas massas juntas é perfeitamente justificado pelo fato de que a única influência da massa de validação no treino da RNA é a parada do próprio, não influenciando no ajuste dos pesos sinápticos.

Os testes são feitos usando o algoritmo *Winner Takes All*, que analisa o vetor de saída da RNA após esta processar determinada entrada e retorna um índice relativo à posição do maior valor deste vetor. Apesar de ser simples, o algoritmo é muito eficiente na hora de avaliar o desempenho geral de uma rede neural, sendo amplamente utilizado em classificadores com muitas classes.

O algoritmo *Winner Takes All* utiliza o pressuposto otimista que o treinamento da rede neural tende a polarizar as suas saídas de acordo com as saídas esperadas. No caso do classificador de caracteres, em que o número de saídas é igual ao número de classes e a saída esperada é o valor máximo (um) na posição da classe e o valor mínimo (zero) no restante, espera-se que o valor de saída da RNA seja o maior possível na classe estimada e o mínimo possível nas outras.

Assim, o algoritmo escolhe o maior valor dentre as saídas da rede neural, sem se importar se há algum outro valor próximo a ele. Por exemplo, caso a resposta da rede possua um valor 0.997 correspondente à terceira linha e na sexta linha esse valor for 0.996, o *Winner Takes All* retorna o índice 3 como o da classe reconhecida.

Dos melhores resultados de performance em cada ensaio, calculados com o *Winner Takes All*, são extraídas as matrizes de confusão. Estas são matrizes quadradas, cuja dimensão é igual ao número de classes de um problema(10x10 para números e 26x26 para letras), onde as linhas representam as classes reais e as colunas as classes estimadas pela RNA.

As matrizes de confusão têm propriedades interessantes, que podem dar informações valiosas sobre o desempenho de uma RNA. Por exemplo, a soma dos elementos de uma linha é igual ao número total de elementos cuja classe real é representada por esta linha. Já os elementos da diagonal da matriz representam o

número de acertos para cada classe. Podemos então calcular o desempenho da RNA para uma determinada classe ao dividir o elemento da diagonal pela soma da linha correspondente. Por exemplo, se para todos os elementos de classe “A”, em 99% das vezes a RNA acertou, o desempenho da RNA para a classe “A” é de 99%.

Outra informação relevante é a acurácia da RNA para determinada classe, calculada pela divisão do elemento da diagonal pela soma dos elementos da coluna correspondente. A acurácia da RNA mede o quanto a rede acerta quando ela estima que dado elemento pertence à determinada classe. Por exemplo, se dentre todas as vezes que a RNA estimou elementos como pertencentes à classe “A”, ela acertou 98% das vezes, podemos dizer que a acurácia desta RNA para a classe “A” é de 98%.

## 4.5 Extração dos Descritores

### 4.5.1 Tipos de descritores

Os descritores de uma imagem são o conjunto de valores extraídos que podem ser usados para o reconhecimento automático da mesma. A representação mais comum de uma imagem é o mapa de bits, onde é representada por uma matriz de pixels, o elemento primário da imagem.

Na classificação de imagens de caracteres, a forma mais comum de representação da imagem é em tons de cinza, onde cada pixel é corresponde a um valor de luminância entre preto, o valor mais baixo (zero), e branco, o mais alto. Dependendo da disponibilidade de memória para armazenamento, o valor do pixel pode ser representado como um *byte* (8 *bits*) de informação, resultando em valores de 0 a 255, ou como um número de ponto flutuante de dupla precisão (*double*), permitindo um conjunto muito maior de valores entre 0 e 1. Neste projeto, o banco de imagens fornece mapas de bits no primeiro formato (inteiros entre 0 e 255), mas durante a extração de descritores esses mapas de bits são passados para o formato *double* entre 0 e 1, pois desta forma a precisão dos números torna os cálculos mais eficientes.

Os descritores de imagens, segundo Gonzalez [2], podem ser classificados em dois tipos principais: características externas, ou de fronteira, e características internas ou regionais, que considera os pixels que compõe a região.

Os descritores de fronteira são usados para se obter informações de formato da

imagem analisada, concentrando-se nas bordas externas e conectividade dos pixels. Assim, a imagem geralmente é binarizada, ou seja, os pixels acima de certo valor limiar passam a ser brancos e todos os outros pretos, pois nesta abordagem não são consideradas relevantes as informações de luminância dos pixels. Sobre a imagem binarizada são então calculados valores que representem a fronteira da mesma. Gonzalez [2] descreve alguns dos principais métodos de extração de descritores de fronteira no Capítulo 11, nas Seções 11.1 e 11.2.

Os descritores internos, ou regionais, podem ser classificados em três tipos principais: simples, topológicos e de textura. Os descritores simples, como diz o nome, são medidas básicas sobre o mapa de bits, como média e mediana dos níveis de cinza, valores mínimos e máximos, área (número de pixels que não são fundo) e também o próprio mapa de bits. Os topológicos são medidas das propriedades de uma figura que não são alteradas por deformações, como o número de buracos e de componentes conexos; a exemplo dos descritores de fronteira também estes são calculados sobre uma imagem binarizada. Os descritores regionais de textura são os mais complexos, fazendo uso da análise de histogramas dos níveis de cinza, de análise do espaço da freqüência da imagem e também usando análise de momentos bidimensionais, que fundamenta um dos ensaios deste projeto. Uma explicação mais detalhada sobre descritores regionais pode ser encontrada na Seção 11.3 do Gonzalez [2].

### 4.5.2 Proposta dos ensaios

O projeto do OCR foi dividido em ensaios com o objetivo de testar duas extrações de descritores com abordagens fundamentalmente diferentes: uma mais simples, baseada na análise matricial da imagem do caractere decomposto em pixels, e outra usando a complexidade estatística dos momentos bidimensionais.

A primeira abordagem, denominada *MapaDeBits*, parte do princípio simplificado da análise pixel a pixel, em que a máquina lê a seqüência de valores como um ser humano lê um texto letra a letra, compreendendo o contexto à medida que conecta as letras em palavras. Da mesma maneira, espera-se que a máquina compreenda os conjuntos de pixels e aprenda o que cada conjunto significa, ou seja, a classe à qual pertence. O aprendizado da máquina ainda é reforçado com a apresentação de projeções verticais e horizontais, que fornecem informações sobre o formato da imagem.

A segunda abordagem, *Momentos*, parte do princípio de que representações estatísticas das imagens são mais robustas, pois não dependem tanto da disposição dos pixels. Assim, os diferentes tipos de imagens pertencentes a uma mesma classe devem possuir características estatísticas próximas umas das outras e preferencialmente muito diferentes daquelas pertencentes às outras classes.

Nas próximas subseções serão descritos os fundamentos de cada tipo de ensaio, com detalhes sobre as configurações e abordagens de análise da imagem e extração dos descritores que as caracterizam.

### 4.5.3 Ensaios usando *MapaDeBits*

A imagem fornecida pelo banco do sistema Kapta vem em um formato padrão de tamanho 16x16, fundo branco e caractere em tons de cinza escuros. O caractere está colocado dentro deste quadrado branco de forma que sempre fique uma moldura de um pixel à sua volta. Como essa informação seria irrelevante para a RNA, por ser comum a todas as imagens, ela certamente atrapalharia o aprendizado, sendo então o primeiro passo eliminar esta moldura.

A imagem agora tem dimensão 14x14, sendo que muitas de suas colunas devem possuir grandes porções de fundo branco sobrando; não podemos, entretanto, eliminar esses pixels, pois existem imagens largas que ocupam alguns destes pixels com informação relevante. A imagem, com pixels com valores inteiros de zero (equivalente à cor preta) a 255 (branco), é passada para a representação de números reais (double) entre 0 e 1 (a nova cor branca). A imagem também tem o valor de seus pixels invertido usando a função **imcomplement**, que torna os valores escuros em seus complementos claros e vice-versa.

O passo inicial da extração de descritores é armazenar os valores de todos os pixels desta imagem no vetor de descritores, ocupando as primeiras 196 posições deste vetor com os valores de cada pixel linha a linha.

O passo seguinte é calcular as projeções horizontal e vertical da imagem. Nesta etapa, a imagem é binarizada com o algoritmo de Otsu [9]. Este algoritmo calcula o limiar de binarização automaticamente a partir de uma análise estatística do histograma da imagem em tons de cinza, e é muito utilizado para segmentação de elementos em imagens, tanto por sua facilidade de uso (é automático) quanto pela eficiência.

Depois serão calculadas as somas de todos os elementos de cada linha, gerando a projeção horizontal com 14 elementos, e também a soma por colunas, que gera a projeção vertical também com 14 elementos. A extração de descritores usada neste ensaio pode ser considerada mista, pois o mapa de bits usado como vetor de características é a forma mais simples de descritor regional, enquanto as projeções são explicitamente uma técnica de extração de descritores de fronteira.

O vetor de descritores então terá tamanho igual a 224. Com o algoritmo PCA, são geradas as massas cujos vetores de descritores possuem tamanhos 224, 202, 180, 157 e 135.

As etapas do processamento e a representação gráfica das projeções de uma imagem do caractere “U” são mostradas na figura 4.7.

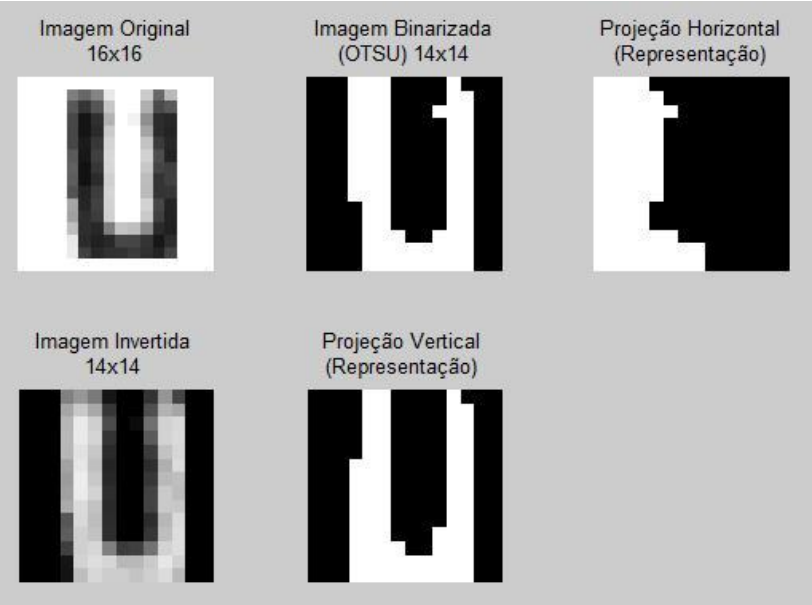


Figura 4.7 - Representação gráfica da extração *MapaDeBits*.

### 4.5.4 Ensaios usando *Momentos*

Os momentos bidimensionais são valores estatísticos calculados sobre o mapa de bits. Neste trabalho serão considerados quatro tipos de momentos bidimensionais: os puros, os centrais, os normalizados e os invariantes. A explicação teórica completa sobre os métodos de cálculo de momentos bidimensionais está no Apêndice 8.1; nesta seção apenas alguns aspectos básicos, necessários à compreensão deste ensaio, serão cobertos.

A ordem de um momento é calculada pela soma dos índices do mesmo. Por exemplo, o momento puro $M_{23}$ é um momento de quinta ordem (2+3=5). Os momentos usados neste trabalho são de até terceira ordem, sendo descartados os momentos centrais e normalizados de ordem zero ($\mu_{00}$ e $\eta_{00}$), por serem sempre iguais ao momento puro $M_{00}$, e os de ordem um do mesmo grupo ($\mu_{01}$, $\mu_{10}$, $\eta_{01}$ e $\eta_{10}$), por serem sempre iguais a zero. Os sete momentos invariantes ($\phi_1$ a $\phi_7$) serão usados sem restrição.

Alguns descritores derivados dos momentos também serão utilizados: os centróides $x_c$ e $y_c$, que são as coordenadas do centro de massa da imagem, e os valores relativos ao tamanho dos eixos da elipse circunscrita à imagem, $a_e$ e $b_e$, e o ângulo $\theta_e$ desta mesma elipse em relação ao eixo horizontal. A tabela 4.8 mostra todos os momentos usados neste ensaio.

| Tipo de Momentos | Momentos usados | Observação |
|---|---|---|
| **Momentos Puros** | $M_{00}$, $M_{01}$, $M_{10}$, $M_{11}$, $M_{02}$, $M_{20}$, $M_{12}$, $M_{21}$, $M_{03}$, $M_{30}$ | - |
| **Momentos Centrais** | $\mu_{11}$, $\mu_{02}$, $\mu_{20}$, $\mu_{12}$, $\mu_{21}$, $\mu_{03}$, $\mu_{31}$ | Invariantes à Translação. |
| **Momentos Normalizados** | $\eta_{11}$, $\eta_{02}$, $\eta_{20}$, $\eta_{12}$, $\eta_{21}$, $\eta_{03}$, $\eta_{32}$ | Invariantes à Translação e Escala. |
| **Momentos Invariantes** | $\phi_1$, $\phi_2$, $\phi_3$, $\phi_4$, $\phi_5$, $\phi_6$, $\phi_7$ | Invariantes à Translação, Escala e Rotação. |
| **Outros** | $x_c$, $y_c$, $a_e$, $b_e$, $\theta_e$ | Calculados a partir dos Momentos Puros e Centrais |

Tabela 4.8 - Todos os descritores disponíveis no ensaio *Momentos*.

A extração de descritores usando momentos tem uma primeira vantagem em relação à anterior: não existe a necessidade de se usar imagens exatamente do mesmo tamanho, pois sobre o mapa de bits serão calculados valores estatísticos. Assim, pode-se eliminar colunas e linhas sem informação, ou seja, contendo apenas informação de fundo.

Partindo da imagem recortada do fundo branco, o mesmo passo de inverter as cores e transformar os pixels para números reais entre zero e um é aplicado. A inversão dos níveis de cinza neste ensaio é interessante, pois os valores dos momentos são calculados sobre a informação relevante, sendo vantagem a cor do fundo da imagem ser

o preto (zero) por diminuir os valores dos descritores.

A primeira abordagem tentada foi a de calcular todos os momentos disponíveis sobre a imagem resultante. Para verificar se essa seria uma boa solução, foi elaborado um ensaio investigativo para comparar com a primeira RNA treinada no ensaio *MapaDeBits*. Foi criada assim uma RNA para sondar o desempenho desta solução, mas o resultado do treinamento foi muito abaixo (cerca de dez pontos percentuais abaixo) do modelo de rede equivalente nos ensaios *MapaDeBits*. Em outros ensaios investigativos, foram eliminados alguns grupos de momentos usados nas extrações, mas nenhuma solução chegou ao nível do desempenho do *MapaDeBits*.

Foi cogitado que o cálculo de momentos sobre a imagem dividida em janelas pudesse melhorar o desempenho do treinamento. Assim, a imagem foi dividida em quatro pedaços de mesmo tamanho, e todos os momentos extraídos sobre cada sub-imagem. De fato o desempenho foi o esperado: o desempenho do ensaio *Momentos* ficou muito próximo à do *MapaDeBits*, chegando a atingir a mesma performance deste. A figura 4.8 mostra a divisão em janelas da imagem antes da extração de descritores:

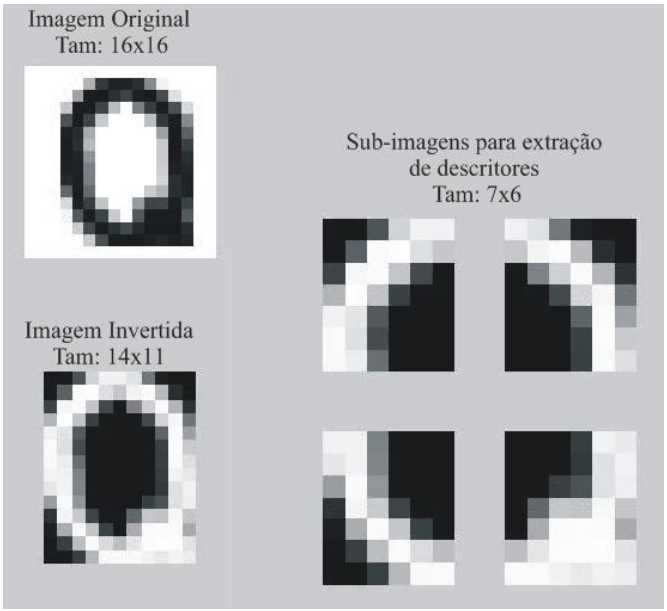


Figura 4.8 - Representação gráfica da extração *Momentos*.

Um último ensaio investigativo foi feito, usando os momentos de melhor desempenho na solução com a imagem inteira. Foi verificado que, com o novo conjunto de momentos, a performance chegou a ser melhor que a melhor solução *MapaDeBits*.

Assim, em um ensaio completo com a mesma estrutura usada nos ensaios *MapaDeBits*, usando uma matriz de 50 RNAs com diferentes configurações, foram

treinadas as redes dos ensaios de Letras e Dígitos com a nova extração definida pelos ensaios investigativos. A melhor configuração de momentos usados na extração está na tabela 4.9:

| **Tipo de Momentos** | **Momentos usados** | **Observação** |
|---|---|---|
| **Momentos Puros** | $M_{00}$, $M_{01}$, $M_{10}$, $M_{11}$, $M_{02}$, $M_{20}$, $M_{12}$, $M_{21}$, $M_{03}$, $M_{30}$ | - |
| **Momentos Centrais** | $\mu_{11}$, $\mu_{02}$, $\mu_{20}$, $\mu_{12}$, $\mu_{21}$, $\mu_{03}$, $\mu_{31}$ | Invariantes à Translação. |
| **Outros** | $x_c$, $y_c$, $a_e$, $b_e$ | Calculados a partir dos Momentos Puros e Centrais |

Tabela 4.9 - Descritores usados na arquitetura final do ensaio *Momentos*.

A extração final de descritores extrai, de cada sub-imagem gerada a partir da imagem invertida e recortada do fundo, os momentos da tabela 4.9 na seguinte ordem: $x_c$, $y_c$, $a_e$, $b_e$ e os momentos puros e centrais na ordem que estão mostrados. É gerado assim um vetor de descritores de tamanho 84, relativo a quatro conjuntos com 21 momentos cada.

O vetor de descritores é então tornado igual ao seu módulo, usando a função de *Matlab* **abs**, para eliminar quaisquer números complexos e negativos do conjunto. Em seguida é verificado para detectar presença de *NaN*s (*Not a Number*), decorrentes de cálculos errados como divisão por zero, e substitui-se todos os valores encontrados pelo valor padrão **-1**.

Com o algoritmo PCA, são geradas as massas cujos vetores de descritores possuem tamanhos 84, 76, 68, 59 e 51. Nota-se que, apesar de o número de descritores ser muito reduzido em comparação com a solução *MapaDeBits*, o desempenho desta solução alcançou uma performance melhor, como será mostrado no próximo capítulo.

# 5 Análise dos Resultados

## 5.1 Introdução

Neste capítulo serão apresentados os resultados dos treinamentos das RNAs em cada ensaio. As massas de dados usadas para testar as RNAs são as de treino e validação, que correspondem a 50% de toda a massa disponível. O algoritmo usado para medir a performance de cada rede é o *Winner Takes All*, que verifica qual a maior saída da RNA e retorna a classe correspondente a essa saída.

Em seguida, serão analisadas as matrizes de confusão dos melhores treinamentos de cada ensaio, verificando a performance e a acurácia por classe de caractere. Serão também comentadas as classes que possuem o maior número de confusões e suas influências no treinamento das redes neurais.

## 5.2 Resultados da abordagem usando *MapaDeBits*

### 5.2.1 Performance geral das redes

Nas tabelas 5.1 e 5.2, são mostradas as performances das cinco melhores RNAs treinadas neste ensaio, para Letras e para Dígitos, com informações sobre o tempo de treinamento e arquitetura das mesmas.

| Neurônios na camada escondida | Tamanho da entrada após o PCA | Acertos na massa de treino | Acertos na massa de teste+validação | Tempo de treino | Épocas usadas no Treino |
|---|---|---|---|---|---|
| 178 | 135 | 99,207% | 95,044% | 03:33:33 | 94 |
| 228 | 135 | 98,943% | 94,923% | 04:08:53 | 108 |
| 218 | 135 | 98,414% | 94,762% | 03:06:34 | 82 |
| 138 | 157 | 98,625% | 94,748% | 02:25:01 | 78 |
| 148 | 135 | 98,890% | 94,721% | 02:12:43 | 89 |

Tabela 5.1 - Resultados do treino das cinco melhores RNAs em *MapaDeBits*-Letras.

| Neurônios na camada escondida | Tamanho da entrada após o PCA | Acertos na massa de treino | Acertos na massa de teste+validação | Tempo de treino | Épocas usadas no Treino |
|---|---|---|---|---|---|
| 192 | 180 | 99,345% | 95,840% | 02:04:06 | 98 |
| 172 | 180 | 98,296% | 95,674% | 01:18:09 | 62 |
| 182 | 157 | 99,410% | 95,674% | 01:17:17 | 89 |
| 182 | 135 | 99,246% | 95,574% | 01:20:52 | 94 |
| 162 | 157 | 99,279% | 95,507% | 01:33:02 | 81 |

Tabela 5.2 - Resultados do treino das cinco melhores RNAs em *MapaDeBits*-Dígitos.

Pode-se observar que o número de épocas ficou entre 62 e 108, bem abaixo do valor limite de 1000 épocas, indicando que os treinamentos foram interrompidos por validação.

A avaliação do desempenho das redes foi feita sobre as massas de teste e validação, que representam juntas 50% do total de imagens, sem levar em conta a performance sobre a massa de treino. Assim, mesmo que uma rede tenha um melhor desempenho em toda a massa de dados disponível, ela só será considerada a melhor se possuir o melhor acerto com os dados que não foram usados para modificar seus pesos, pois essa performance é a que indica a capacidade de generalização de uma RNA.

Na tabelas 5.3 serão mostradas as performances do melhor treinamento de cada solução por local de origem.

| | Dígitos | | Letras | |
|---|---|---|---|---|
| **Local** | **Treino** | **Teste + Validação** | **Treino** | **Teste + Validação** |
| **L1** | 98,381% | 96,761% | 98,934% | 92,424% |
| **L2** | 99,200% | 96,000% | 99,701% | 95,808% |
| **L3** | 98,760% | 99,587% | 99,183% | 92,371% |
| **L4** | 99,197% | 97,189% | 99,167% | 97,833% |
| **L5** | 98,760% | 97,107% | 99,297% | 97,627% |
| **L6** | 99,690% | 90,712% | 98,985% | 95,939% |
| **L7** | 99,793% | 91,529% | 99,404% | 93,373% |
| **L8** | 99,177% | 97,942% | 99,507% | 97,537% |
| **L9** | 100,000% | 98,326% | 99,315% | 98,630% |
| **L10** | 99,180% | 97,541% | 99,065% | 92,523% |
| **L11** | 100,000% | 97,521% | 98,867% | 96,884% |

Tabela 5.3 - Resultados por local da melhor RNA nos ensaios de Letras e Dígitos.

A performance nos locais variou entre 90% e 100%, sendo considerados de

baixa performance aqueles com acertos abaixo de 95%. Os locais de pior performance em Letras foram o L1, um estacionamento externo de São Paulo, o L3 e o L10, ambos estacionamentos cobertos no Rio de Janeiro, e o L7, que é o acesso à UFRJ pela Prefeitura. Em Dígitos, apenas L6, outro acesso à UFRJ pela Linha Amarela, e L7 ficaram com performances abaixo de 95%. Dada a variedade de tipos de ambientes, pode-se dizer que o OCR é robusto para imagens provenientes de diferentes condições de captura, sejam eles internos, externos, de trânsito rápido ou controlado, com luz natural ou artificial, durante o dia e durante a noite.

### 5.2.2 Análise das Matrizes de Confusão

As matrizes de confusão foram extraídas a partir das estimativas da melhor RNA em cima das massas de teste e validação. Nas figuras 5.1 e 5.2 serão mostradas as matrizes de confusão relativas às melhores performances para Letras e Dígitos, respectivamente. As linhas representam os valores reais (ou targets) e as colunas representam os valores estimados pela melhor RNA de cada ensaio. Podemos com essas matrizes calcular a performance e a acurácia por classe de caracteres. Os valores estão mostrados nas tabelas 5.4 e 5.5.

| | 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 | 279 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| B | 0 | 262 | 0 | 7 | 3 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 2 | 0 | 2 | 0 | 1 | 3 | 1 | 0 | 0 | 2 | 0 | 1 | 0 | 0 |
| C | 1 | 0 | 271 | 1 | 2 | 0 | 1 | 0 | 1 | 0 | 1 | 3 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| D | 0 | 1 | 1 | 239 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 24 | 0 | 12 | 1 | 0 | 0 | 3 | 1 | 0 | 1 | 0 | 0 |
| E | 0 | 0 | 0 | 0 | 281 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| F | 0 | 1 | 0 | 0 | 2 | 280 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| G | 0 | 0 | 3 | 1 | 0 | 0 | 267 | 0 | 0 | 0 | 2 | 0 | 0 | 1 | 5 | 0 | 3 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| H | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 271 | 0 | 1 | 2 | 1 | 3 | 1 | 0 | 1 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| I | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 280 | 2 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| J | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 284 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| K | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 270 | 1 | 4 | 2 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 4 | 0 | 0 | 1 |
| L | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 4 | 0 | 0 | 282 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| M | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 4 | 0 | 261 | 5 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 2 | 8 | 0 | 1 | 1 |
| N | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 3 | 0 | 7 | 268 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| O | 0 | 1 | 3 | 23 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 1 | 0 | 235 | 0 | 20 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| P | 0 | 2 | 0 | 1 | 0 | 4 | 0 | 2 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 273 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Q | 0 | 0 | 3 | 4 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 14 | 0 | 256 | 1 | 0 | 0 | 1 | 3 | 0 | 1 | 0 | 0 |
| R | 0 | 2 | 0 | 1 | 0 | 1 | 0 | 2 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 4 | 0 | 273 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| S | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 282 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| T | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 283 | 0 | 0 | 0 | 0 | 0 | 0 |
| U | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 2 | 1 | 2 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 276 | 0 | 0 | 0 | 0 | 0 |
| V | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 2 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 4 | 272 | 0 | 0 | 1 | 0 |
| W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 279 | 0 | 0 | 0 |
| X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 280 | 1 | 0 |
| Y | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 1 | 1 | 277 | 0 |
| Z | 0 | 0 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 277 |

Figura 5.1 - Matriz de Confusão de Letras.

| Classe | Performance | Acurácia | Classe | Performance | Acurácia |
|---|---|---|---|---|---|
| **A** | 97,895% | 98,936% | **N** | 94,035% | 96,057% |
| **B** | 91,289% | 96,324% | **O** | 81,882% | 83,039% |
| **C** | 94,425% | 95,423% | **P** | 95,790% | 96,809% |
| **D** | 84,155% | 84,155% | **Q** | 89,199% | 85,050% |
| **E** | 98,252% | 96,897% | **R** | 95,455% | 96,466% |
| **F** | 98,592% | 96,886% | **S** | 98,947% | 97,241% |
| **G** | 93,684% | 98,524% | **T** | 98,606% | 98,951% |
| **H** | 95,088% | 95,423% | **U** | 96,504% | 95,502% |
| **I** | 98,592% | 96,220% | **V** | 95,775% | 95,775% |
| **J** | 99,301% | 96,928% | **W** | 98,239% | 94,576% |
| **K** | 94,077% | 92,784% | **X** | 98,592% | 97,561% |
| **L** | 97,578% | 96,907% | **Y** | 96,853% | 98,577% |
| **M** | 91,259% | 91,579% | **Z** | 97,193% | 99,283% |

Tabela 5.4 - Performance e Acurácia por classe ensaios - Letras.

Uma análise primária das performances do ensaio Letras demonstra que três classes de caracteres estão muito abaixo do valor médio, com valores abaixo de 90%: **D**, **O** e **Q**. Olhando na matriz de confusão, pode-se ver que 24 imagens de classe **D** são estimadas como **O** e 12 como **Q**, enquanto 23 imagens da classe **O** são reconhecidas como **D** e 12 como **Q**. Das imagens pertencentes à classe **Q**, apenas 4 são confundidas com **D**, mas 14 são confundidas com **O**. Pode-se notar que essas três classes são responsáveis por 89 imagens classificadas erradamente, ou seja, mais de 1% de toda a massa de teste e validação juntas. Somente resolvendo essa confusão, a performance geral do sistema para letras subiria de 95% para 96%.

Outras confusões menores podem ser observadas: **H**, **M**, **N** e **W** fazem parte de uma confusão responsável por 33 imagens classificadas erradas, sendo a segunda maior confusão verificada; pequenas confusões como **I** com **L** e **X** com **K** respondem cada uma por cerca de cinco imagens essas confusões. Todas as confusões menores juntas respondem por cerca de metade das imagens erradas na maior confusão (**DOQ**), correspondendo a cerca de meio por cento da massa total usada no teste.

Ainda assim, uma solução que resolvesse essas confusões elevaria a taxa de acertos do OCR de letras para cerca de 96,5%. Nota-se que as confusões são sempre entre caracteres cujas imagens são bem parecidas, o que sugere que um descritor baseado unicamente no mapa de bits, com todas as variações nos pixels que imagens de uma mesma classe apresenta, raramente conseguirá solucionar essas questões por si.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 289 | 1 | 0 | 3 | 3 | 0 | 2 | 0 | 2 | 0 |
| 1 | 0 | 299 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 1 | 291 | 2 | 0 | 0 | 2 | 3 | 1 | 1 |
| 3 | 1 | 1 | 1 | 289 | 1 | 1 | 2 | 2 | 0 | 2 |
| 4 | 0 | 1 | 1 | 0 | 300 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 0 | 1 | 2 | 0 | 280 | 14 | 0 | 2 | 1 |
| 6 | 0 | 0 | 0 | 1 | 7 | 5 | 281 | 0 | 6 | 1 |
| 7 | 0 | 1 | 1 | 2 | 1 | 0 | 1 | 292 | 0 | 1 |
| 8 | 4 | 1 | 2 | 0 | 0 | 0 | 3 | 0 | 282 | 6 |
| 9 | 3 | 0 | 2 | 7 | 2 | 3 | 1 | 0 | 5 | 277 |

Figura 5.2 - Matriz de Confusão de Dígitos.

| Classe | Performance | Acurácia | Classe | Performance | Acurácia |
|---|---|---|---|---|---|
| **0** | 96,333% | 96,980% | **5** | 93,333% | 96,552% |
| **1** | 99,007% | 98,033% | **6** | 93,356% | 91,234% |
| **2** | 96,358% | 97,000% | **7** | 97,659% | 98,317% |
| **3** | 96,333% | 94,444% | **8** | 94,631% | 94,631% |
| **4** | 99,010% | 95,541% | **9** | 92,333% | 95,848% |

Tabela 5.5 - Performance e Acurácia por classe ensaios - Dígitos.

Analisando as performances do ensaio Dígitos, vemos que quatro classes estão abaixo da performance geral: **5**, **6**, **8** e **9**. A classe **9** é a de pior performance, respondendo por 23 imagens mal classificadas, só não sendo confundida com as classes **1** e **7**. A classe **5**, segunda pior, se confunde somente com a classe **6** em 14 imagens, mas com outras classes em apenas mais 5. A classe **6** se confunde com a **5**, com a **4** e com a **8** em 18 imagens, enquanto a **8** tem suas maiores confusões com as classes **0**, **6** e **9**, totalizando 13 imagens.

A grande confusão verificada é entre as classes **5** e **6**, responsáveis por 19 imagens mal classificadas. Outras duas grande confusões são entre **6** e **8** (9 imagens) e entre **8** e **9** (11 imagens). Juntas, as confusões entre essas quatro classes respondem por 49 imagens, que em um conjunto de cerca de três mil elementos correspondem a 0,61%. Assim, uma solução que resolvesse essas confusões elevaria a taxa de acertos do OCR de letras para cerca de 96,5%.

## 5.3 Resultados da abordagem usando *Momentos*

### 5.3.1 Resultados preliminares de concepção da extração

Diferente das RNAs do ensaio *MapaDeBits*, cuja extração de descritores foi concebida por Bruno Guingo [4] em sua dissertação de mestrado, não existia uma solução de extração de descritores baseada em momentos pré-concebida para este trabalho. Assim, alguns ensaios investigativos foram feito a fim de validar a solução.

Foi tomada como base a solução do ensaio *MapaDeBits* que não usava o algoritmo PCA para redimensionar o vetor de descritores, mas apenas para os descorrelacionar. Para o tamanho da camada escondida, foi usada a configuração inicial da estrutura de ensaios, que define o número de neurônios como a média aritmética do número de entradas com o número de saídas.

Foram feitos seis ensaios investigativos com a estrutura da RNA fixa, modificando apenas a extração dos descritores. Nas tabelas 5.6 e 5.7 estão as performances obtidas nesses ensaios sobre as massas de teste e validação, com informações sobre a arquitetura da RNA usada, lembrando que o tamanho da entrada com e sem PCA é o mesmo. Os valores de referência de performance dos ensaios *MapaDeBits* são 95,044% para Letras e 95,840% para Dígitos .

| Ensaio | Acertos | Extração de Descritores | | Dimensões RNA | |
|---|---|---|---|---|---|
| | | Momentos | Janelas | Tam. Entrada | Camada Escond. |
| 1 | 80,912% | $x_c, y_c, a_e, b_e, \theta_e, \{M_{xy}\}, \{\mu_{xy}\}, \{\eta_{xy}\}, \{\phi_{xy}\}$ | Não | 36 | 31 |
| 2 | 83,078% | $x_c, y_c, a_e, b_e, \theta_e, \{M_{xy}\}, \{\mu_{xy}\}, \{\eta_{xy}\}$ | Não | 29 | 28 |
| 3 | 85,554% | $x_c, y_c, a_e, b_e, \theta_e, \{M_{xy}\}, \{\mu_{xy}\}$ | Não | 22 | 24 |
| 4 | 82,939% | $x_c, y_c, a_e, b_e, \theta_e, \{M_{xy}\}$ | Não | 15 | 21 |
| 5 | 86,301% | $x_c, y_c, a_e, b_e, \{M_{xy}\}, \{\mu_{xy}\}$ | Não | 21 | 24 |
| 6 | 94,964% | $x_c, y_c, a_e, b_e, \theta_e, \{M_{xy}\}, \{\mu_{xy}\}, \{\eta_{xy}\}, \{\phi_{xy}\}$ | Sim | 144 | 85 |
| 7 | 95,637% | $x_c, y_c, a_e, b_e, \{M_{xy}\}, \{\mu_{xy}\}$ | Sim | 84 | 55 |

Tabela 5.6 - Performances dos ensaios preliminares - Letras.

| Ensaio | Acertos | Extração de Descritores | | Dimensões RNA | |
|---|---|---|---|---|---|
| | | Momentos | Janelas | Tam. Entrada | Camada Escond. |
| 1 | 82,655% | $x_c, y_c, a_e, b_e, \theta_e, \{M_{xy}\}, \{\mu_{xy}\}, \{\eta_{xy}\}, \{\phi_{xy}\}$ | Não | 36 | 23 |
| 2 | 85,909% | $x_c, y_c, a_e, b_e, \theta_e, \{M_{xy}\}, \{\mu_{xy}\}, \{\eta_{xy}\}$ | Não | 29 | 20 |
| 3 | 87,111% | $x_c, y_c, a_e, b_e, \theta_e, \{M_{xy}\}, \{\mu_{xy}\}$ | Não | 22 | 16 |
| 4 | 85,670% | $x_c, y_c, a_e, b_e, \theta_e, \{M_{xy}\}$ | Não | 15 | 13 |
| 5 | 89,301% | $x_c, y_c, a_e, b_e, \{M_{xy}\}, \{\mu_{xy}\}$ | Não | 21 | 16 |
| 6 | 94,775% | $x_c, y_c, a_e, b_e, \theta_e, \{M_{xy}\}, \{\mu_{xy}\}, \{\eta_{xy}\}, \{\phi_{xy}\}$ | Sim | 144 | 77 |
| 7 | 95,973% | $x_c, y_c, a_e, b_e, \{M_{xy}\}, \{\mu_{xy}\}$ | Sim | 84 | 47 |

Tabela 5.7 - Performances dos ensaios preliminares - Dígitos.

Pode-se observar que nos ensaios 7 de Letras e Dígitos o desempenho foi maior do que os valores de referência de performance dos ensaios *MapaDeBits*. A extração de descritores do ensaio 7 foi então escolhida para o treinamento dos ensaios *Momentos*, usando o janelamento da imagem e extraindo sobre cada sub-imagem resultante os momentos centrais e puros, assim como as centróides e eixos da elipse circunscrita.

### 5.3.2 Performance geral das redes

Nas tabelas 5.8 e 5.9 são mostradas as performances das cinco melhores RNAs treinadas neste ensaio para Letras e Dígitos.

| Neurônios na camada escondida | Tamanho da entrada após o PCA | Acertos na massa de treino | Acertos na massa de teste+validação | Tempo de treino | Épocas usadas no Treino |
|---|---|---|---|---|---|
| 88 | 84 | 98,335% | 95,879% | 01:14:03 | 84 |
| 118 | 76 | 98,718% | 95,879% | 01:32:17 | 85 |
| 158 | 84 | 98,586% | 95,879% | 02:12:02 | 84 |
| 148 | 84 | 98,797% | 95,866% | 02:06:03 | 86 |
| 118 | 84 | 98,718% | 95,825% | 01:58:59 | 91 |

Tabela 5.8 - Resultados do treino das cinco melhores RNAs em *Momentos*-Letras.

| Neurônios na camada escondida | Tamanho da entrada após o PCA | Acertos na massa de treino | Acertos na massa de teste+validação | Tempo de treino | Épocas usadas no Treino |
|---|---|---|---|---|---|
| 132 | 84 | 99,541% | 96,173% | 01:33:48 | 82 |
| 62 | 84 | 99,836% | 96,106% | 00:41:41 | 106 |
| 82 | 76 | 99,771% | 96,073% | 01:23:24 | 95 |
| 122 | 76 | 99,607% | 96,073% | 01:23:48 | 84 |
| 102 | 84 | 99,672% | 96,040% | 00:50:19 | 94 |

Tabela 5.9 - Resultados do treino das cinco melhores RNAs em *Momentos*-Dígitos.

Pode-se observar que o número de épocas ficou entre 82 e 106, praticamente a mesma faixa do ensaio *MapaDeBits*; isso indica também que as redes pararam o treinamento por validação. O diferencial está no tempo de treinamento, que na maioria foi bem menor que o tempo do *MapaDeBits*, especialmente em Letras, onde o tempo foi quase a metade em todos os testes.

Na tabela 5.3 serão mostradas as performances do melhor treinamento em Letras e Dígitos, por local de origem.

| | Dígitos | | Letras | |
|---|---|---|---|---|
| **Local** | **Treino** | **Teste + Validação** | **Treino** | **Teste + Validação** |
| **L1** | 99,595% | 95,951% | 98,485% | 93,715% |
| **L2** | 99,600% | 97,200% | 99,102% | 95,808% |
| **L3** | 99,587% | 99,587% | 97,820% | 93,188% |
| **L4** | 99,598% | 96,787% | 97,833% | 99,167% |
| **L5** | 100% | 97,934% | 98,330% | 97,627% |
| **L6** | 99,690% | 90,093% | 97,716% | 96,954% |
| **L7** | 99,174% | 92,355% | 98,436% | 94,639% |
| **L8** | 100% | 98,354% | 98,851% | 98,030% |
| **L9** | 99,6% | 98,326% | 97,945% | 98,973% |
| **L10** | 100% | 97,951% | 98,131% | 93,925% |
| **L11** | 98,3% | 99,174% | 98,017% | 96,884% |

Tabela 5.10 - Resultados por local da melhor RNA nos ensaios de Letras e Dígitos.

Verificando os locais onde a performance foi abaixo de 95%, pode-se ver que são exatamente os mesmos destacados no ensaio *MapaDeBits*: **L1**, **L3**, **L7** e **L10** em Letras e **L6** e **L7** em Dígitos. A não ser pela performance de **L6** em Dígitos, que ficou menor em *Momentos* por 0,62%, os demais locais tiveram desempenho superior aos dos

ensaios *MapaDeBits* por volta de 1%.

### 5.3.3 Análise das Matrizes de Confusão

A matriz de confusão do ensaio de Letras, na figura 5.3, será analisada a seguir, com o auxílio da tabela 5.11:

| | 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 | 281 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| B | 1 | 269 | 0 | 2 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 3 | 0 | 2 | 2 | 2 | 0 | 0 | 0 | 2 | 0 | 0 | 0 |
| C | 0 | 0 | 273 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 5 | 0 | 0 | 3 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| D | 0 | 0 | 1 | 246 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 29 | 0 | 6 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| E | 0 | 0 | 0 | 0 | 279 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 3 | 0 | 0 | 0 |
| F | 0 | 0 | 1 | 0 | 3 | 280 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| G | 0 | 1 | 4 | 2 | 0 | 0 | 273 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 273 | 0 | 0 | 1 | 0 | 5 | 2 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| I | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 279 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| J | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 284 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| K | 1 | 1 | 2 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 274 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 3 | 0 | 1 | 0 |
| L | 1 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 9 | 0 | 1 | 274 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| M | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 5 | 0 | 0 | 3 | 0 | 257 | 7 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 5 | 0 | 1 | 0 |
| N | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 1 | 0 | 5 | 275 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| O | 0 | 0 | 6 | 15 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 247 | 1 | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| P | 0 | 1 | 0 | 0 | 0 | 6 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 276 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Q | 0 | 4 | 2 | 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 13 | 0 | 263 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 0 | 280 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| S | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 284 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| T | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 282 | 0 | 0 | 0 | 0 | 0 | 0 |
| U | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 278 | 0 | 0 | 0 | 0 | 0 |
| V | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 278 | 2 | 0 | 1 | 0 |
| W | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 279 | 0 | 0 | 0 |
| X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 280 | 0 | 0 |
| Y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 2 | 0 | 1 | 1 | 0 | 0 | 0 | 3 | 0 | 0 | 276 | 0 |
| Z | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 280 |

Figura 5.3 - Matriz de Confusão de Letras.

| Classe | Performance | Acurácia | Classe | Performance | Acurácia |
|---|---|---|---|---|---|
| **A** | 98,597% | 97,569% | **N** | 96,491% | 95,819% |
| **B** | 93,728% | 96,416% | **O** | 86,063% | 81,250% |
| **C** | 95,122% | 93,174% | **P** | 96,842% | 98,571% |
| **D** | 86,620% | 90,441% | **Q** | 91,638% | 90,378% |
| **E** | 97,552% | 96,875% | **R** | 97,902% | 97,902% |
| **F** | 98,592% | 96,552% | **S** | 99,649% | 98,270% |
| **G** | 95,790% | 98,201% | **T** | 98,258% | 98,258% |
| **H** | 95,790% | 94,138% | **U** | 97,203% | 99,642% |
| **I** | 98,239% | 96,207% | **V** | 97,887% | 98,233% |
| **J** | 99,301% | 99,301% | **W** | 98,239% | 94,257% |
| **K** | 95,470% | 96,479% | **X** | 98,592% | 99,291% |
| **L** | 94,810% | 94,810% | **Y** | 96,504% | 98,925% |
| **M** | 89,860% | 93,796% | **Z** | 98,246% | 99,291% |

Tabela 5.11 - Performance e Acurácia por classe ensaios - Letras.

Analisando as performances, verificou-se que quatro classes possuem este valor abaixo de 90%: **D**, **O** e **M**. As classes **B**, **L** e **Q** são, em seguida, as com pior desempenho, variando entre 91% e 94%.

No ensaio *MapaDeBits* de Letras, **D** e **O**, juntamente com **Q** (aqui com performance de 91,64%), formavam um grupo de confusão com o maior número de elementos, respondendo por 1% da massa total. Neste ensaio, somente a confusão destes caracteres entre si é responsável por 72 imagens erradas, contra as 89 no ensaio *MapaDeBits*, resultando em um ganho de 17 imagens corretamente classificadas.

A segunda maior confusão continua sendo entre as classes **M**, **N**, **W** e **H**, respondendo por 26 imagens classificadas erradamente. As outras confusões menores, somadas com a segunda maior, ainda respondem por pouco mais da metade das imagens da confusão principal, como é também o caso do ensaio *MapaDeBits*. Ainda assim, nota-se um ganho em desempenho que reflete em um resultado acima de 96% da massa total de teste.

A seguir, será analisada a matriz de confusão do ensaio de Dígitos, mostrada na figura 5.4, com o auxílio da tabela 5.12:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 287 | 1 | 0 | 2 | 0 | 0 | 3 | 0 | 4 | 3 |
| 1 | 0 | 298 | 2 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 291 | 2 | 0 | 0 | 2 | 1 | 0 | 5 |
| 3 | 0 | 2 | 2 | 290 | 0 | 0 | 2 | 0 | 3 | 1 |
| 4 | 1 | 0 | 0 | 0 | 298 | 0 | 4 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 1 | 286 | 7 | 0 | 2 | 2 |
| 6 | 0 | 0 | 0 | 0 | 3 | 6 | 289 | 0 | 2 | 1 |
| 7 | 0 | 3 | 2 | 0 | 0 | 1 | 1 | 290 | 1 | 1 |
| 8 | 3 | 0 | 0 | 0 | 0 | 0 | 5 | 1 | 284 | 5 |
| 9 | 2 | 0 | 2 | 4 | 0 | 3 | 2 | 2 | 8 | 277 |

Figura 5.4 - Matriz de Confusão de Dígitos.

| Classe | Performance | Acurácia | Classe | Performance | Acurácia |
|---|---|---|---|---|---|
| **0** | 96,333% | 96,980% | **5** | 93,333% | 96,552% |
| **1** | 99,007% | 98,033% | **6** | 93,356% | 91,234% |
| **2** | 96,358% | 97,000% | **7** | 97,659% | 98,317% |
| **3** | 96,333% | 94,444% | **8** | 94,631% | 94,631% |
| **4** | 99,010% | 95,541% | **9** | 92,333% | 95,848% |

Tabela 5.12 - Performance e Acurácia por classe ensaios - Dígitos.

No ensaio de Dígitos, verifica-se que as mesmas quatro classes de caracteres estão com performance abaixo de 95%: **5**, **6**, **8** e **9**. Juntas, essas confusões respondem

por 43 imagens erradas, 6 a menos que a solução *MapaDeBits*. Ainda assim, pode-se observar que existem menos confusões menores na matriz de confusão, indicando um classificador mais eficiente.

## 5.4 Comparação dos ensaios

De modo geral, as soluções usando a extração *Momentos* foram melhores que os ensaios *MapaDeBits*, tanto pelo desempenho quanto pelo treinamento mais rápido. Além disso, os ensaios *Momentos* exigem menor poder computacional, pois as estruturas das RNAs possuem bem menos neurônios e o número de entradas é significativamente menor.

Ainda assim, as soluções obtiveram uma taxa de acertos razoavelmente alta, entre 95% e 96%, sobre uma quantidade significativa de imagens. As confusões entre as classes de caracteres se mostraram responsáveis por uma boa parte dos erros de classificação, sendo um ponto a ser trabalhado em soluções de OCR futuras.

Na tabela 5.13 serão apresentados alguns dados sobre os testes em cima de todas as RNAs treinadas em cada ensaio, como média e desvio padrão das performances nos testes e do tempo e número de épocas no treinamento.

| | | **MapaDeBits** | | **Momentos** | |
|---|---|---|---|---|---|
| | | **Letras** | **Dígitos** | **Letras** | **Dígitos** |
| **Performance do teste** | **Média** | 94,395% | 95,191% | 95,453% | 95,555% |
| | **Mínimo** | 93,711% | 94,709% | 94,964% | 94,775% |
| | **Máximo** | 95,044% | 95,840% | 95,879% | 96,173% |
| | **Std** | 0,274% | 0,263% | 0,275% | 0,356% |
| **Tempo de treino** | **Média** | 02:13:50 | 01:20:27 | 01:06:27 | 00:57:32 |
| | **Mínimo** | 01:21:17 | 00:56:55 | 01:00:17 | 00:27:23 |
| | **Máximo** | 04:29:53 | 01:29:50 | 01:55:27 | 01:39:05 |
| | **Std** | 01:01:38 | 00:16:01 | 00:26:33 | 00:19:16 |
| **Épocas do treino** | **Média** | 88,30 | 73,16 | 90,72 | 87.22 |
| | **Mínimo** | 68 | 60 | 76 | 71 |
| | **Máximo** | 184 | 98 | 114 | 136 |
| | **Std** | 18,938 | 8,837 | 8,345 | 15,944 |

Tabela 5.13 - Comparação estatística entre *MapaDeBits* e *Momentos*.

# 6 Conclusões

Este estudo apresenta uma metodologia de construção de OCR para caracteres de placa de automóvel usando redes neurais artificiais. Foram usadas imagens de placas do Rio de Janeiro e de São Paulo, provenientes de onze locais diferentes, entre estacionamentos cobertos e externo, avenidas de trânsito rápido e acessos de condomínio e da própria Cidade Universitária, em condições de luminosidade controladas (no caso de estacionamentos cobertos) e ao ar livre, tanto de dia quanto à noite.

O estudo comparou duas soluções de extração de descritores das imagens de caracteres de placa, uma baseada no mapa de bits puro e outra baseada em momentos bidimensionais. Foram treinadas e testadas duzentas redes neurais, com o objetivo de comparar as duas extrações e também atingir a melhor performance de acertos possível.

Este trabalho ainda propôs uma metodologia de construção de ensaios de classificadores de imagens de caractere usando RNA que analisa diferentes extrações de descritores e compara seus resultados de maneira quantitativa e qualitativa, usando massas de dados consistentes e comparando as performances do ponto de vista da estimativa do classificador, com o conceito de acurácia por classe sendo a confiabilidade de cada saída do mesmo, com a performance do ponto de vista da saída esperada, usada para medir a percentagem de acertos bruta.

O objetivo de construir um OCR robusto foi atingido, com uma performance final de 96,173% no reconhecimento de dígitos e de 95,879% no reconhecimento de letras. Os acertos por locais ficaram entre 90,093% e 99,587% para dígitos e entre 93,188% e 99,167% para letras. Os números de melhor desempenho apresentados são relativos à melhor solução de extração de descritores, *Momentos*.

Foram verificadas as confusões mais comuns entre as classes de caracteres, que se resolvidas resultariam em um aumento de pelo menos um ponto percentual em ambos as soluções de dígitos e de letras.

Um trabalho futuro interessante seria um método de extração de descritores diferenciado para os caracteres pertencentes às classes de maior confusão, podendo a partir do momento que se identifica que a imagem de caractere pertence à uma dessas classes, usar uma RNA especialmente treinada para isto.

Outra solução interessante seria usar comitês de redes neurais para uma solução

com vários tipos de extrações de descritores, em que cada solução acerte caracteres que a outra erre. O desenvolvimento de um comitê de RNAs com as duas soluções propostas neste trabalho poderia resultar em um aumento de performance, visto que a diferença das matrizes de confusão obtidas nos testes é bem significativa.

O estudo de um algoritmo de tratamento das saídas da RNA mais completo do que o *Winner Takes All*, usando as estatísticas geradas nos testes deste estudo, seria também uma expansão futura dos conceitos aqui apresentados; uma solução que empregue as informações extraídas das matrizes de confusão combinadas às saídas da rede neural aumenta o aproveitamento das informações estatísticas que esse tipo de saída de um classificador oferece.

Ainda seria um trabalho futuro interessante um estudo mais profundo da extração usando momentos bidimensionais, variando a composição dos momentos e também os tipos de janelamento da imagem, visto que foi esta solução que alavancou a performance dos ensaios *Momentos.*

Ainda na questão dos descritores, há muito espaço para testes de extração de descritores usados para treinar RNAs, com uma vasta gama de características de uma imagem que não foram abordadas neste trabalho, que podem ser vistas no Capítulo 11 do Gonzalez [2]. Dentre elas, o uso de momentos sobre os histogramas dos tons de cinza e as características do espaço da freqüência são descritores que foram pouco estudados no campo do reconhecimento de caracteres.

# 7 Bibliografia


[1] HAYKIN, Simon, "*Neural Networks: A Comprehensive Foundation*", Prentice Hall, 1999.

[2] GONZALEZ, Rafael C., "*Digital Image Processing Using Matlab*", Segunda Edição, Gatesmark Publishing, 2009.

[3] LILENBAUM, Ricardo C., *O Problema da Localização de Placas de Veículos – Análise de uma Abordagem via Redes Neurais*, Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Universidade Federal do Rio de Janeiro, Rio de Janeiro, Novembro 2006.

[4] GUINGO, Bruno C., *Reconhecimento Automático de Placas de Veículos Automotores*, Dissertação (Mestrado em Informática) - Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2003.

[5] GUINGO, Bruno C.; STIEBLER, Guilherme M.; THOMÉ, Antonio C. G., *Kapta – Um Sistema de Reconhecimento Automático de Placas de Veículos Baseado nas Técnicas de Redes Neurais e Processamento de Imagens*, Congresso Brasileiro de Tecnologia da Informação e Comunicação – SUCESU 2004, Florianópolis, 2004.

[6] CORCKE, Peter I., *Machine Vision Toolbox*, IEEE Robotics and Automation Magazine, 12(4), pp 16-25, November 2005.

[7] PEYRÉ, Gabriel, *Dimension Reduction Toolbox*, 2006.

[8] DEMUTH, Howard; BEALE, Mark; HAGAN, Martin, *Matlab Neural Network Toolbox ™ 6 User's Guide*, The MathWorks, 2007.

[9] OTSU, Nobuyuki, A *threshold selection method from gray-level histograms*, Systems, Man and Cybernetics, IEEE Transactions, Volume 9, Issue 1, pp 62 – 66, Janeiro 1979.

[10] RODRIGUES, Roberto J., *Segmentação e Extração de Características para Reconhecimento Automático de Caracteres – Estudo e Propostas*, Dissertação (Mestrado em Informática) - Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2003.

[11] MCCULLOCH, Warren S.; PITTS, Walter, *A Logical Calculus of the Ideas Immanent in Nervous Activity*, Bulletin of Mathematical Biophysics, 1943.

[12] HEBB, Donald O., *The Organization of Behavior*. New York, Wiley, 1949

[13] ROSENBLATT, Frank, *Principles of Neurodynamics*. New York, Spartan Books, 1959.

[14] HOPFIELD, John J., *Neural Networks and Physical Systems with Emergent Collectives Computational Abilities*, Proceedings of the National Academy of Sciences, 1982.

[15] RUMELHART, David E.; DURBIN, Richard; GOLDEN, Richard; CHAUVIN, Yves , *Backpropagation: Theory, Architectures and Applications*, Hillsdale, New Jersey, Erlbaum, 1992.

[16] CYBENKO, George, *Continuos Valued Neural Network with two Hidden Layers are Sufficient*, Technical Report, Departament of Computer Science, Tufts University, 1988.

[17] HU, Ming K., *Visual Pattern Recognition by Moment Invariants*, IRE Trans. Info. Theory, vol. IT-8, pp.179-187, 1962.

# 8 Apêndice

## 8.1 Momentos Bidimensionais

Momentos são medidas de distribuição de informação, usadas para descrever uma função em termos de suas propriedades estatísticas. Para calcular medidas estatísticas em funções de duas variáveis, são usados os momentos bidimensionais.

### 8.1.1 Momentos puros e centrais

Seja uma função contínua f(x,y), o momento bidimensional de ordem p+q será escrito como na equação (8.1).

$$M_{pq} = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} x^p . y^q . f(x,y).\mathrm{dx}.\mathrm{dy} \tag{8.1}$$

Uma imagem digital em tons de cinza pode ser considerada uma função discreta bidimensional I(x,y). Assim, cada valor I(x,y) representa a luminância do pixel situado na linha x e na coluna y. O momento puro de ordem p+q da imagem I pode ser calculado como na equação (8.2).

$$M_{pq} = \sum_x \sum_y x^p . y^q . I(x,y) \tag{8.2}$$

Usando os momentos puros de ordem zero e de primeira ordem, pode-se calcular as centróides (centro de massa da imagem), como mostrado na equação (8.3).

$$\begin{cases} M_{00} = \sum_x \sum_y I(x,y) \\ M_{10} = \sum_x \sum_y x.I(x,y) \\ M_{01} = \sum_x \sum_y y.I(x,y) \end{cases} \Rightarrow \begin{cases} \mathrm{x_c} = \dfrac{M_{10}}{M_{00}} \\ y_c = \dfrac{M_{01}}{M_{00}} \end{cases} \tag{8.3}$$

Obtidos esses valores, os momentos centrais de ordem p+q podem ser calculados com a equação (8.4).

$$\mu_{pq} = \sum_x \sum_y (x - x_c)^p . (y - y_c)^q . I(x, y) \tag{8.4}$$

Os momentos centrais calculados sobre uma imagem são invariantes à translação da mesma. Os momentos centrais podem também ser calculados usando os valores dos momentos puros. O cálculo dos momentos centrais de até terceira ordem pode ser visto no conjunto de equações (8.5).

$$\begin{aligned}
&\mu_{00} = M_{00} \\
&\mu_{10} = 0 \\
&\mu_{01} = 0 \\
&\mu_{11} = M_{11} - x_c . M_{01} = M_{11} - y_c . M_{10} \\
&\mu_{20} = M_{20} - x_c . M_{10} \\
&\mu_{02} = M_{20} - x_c . M_{10} \\
&\mu_{21} = M_{21} - 2..x_c . M_{11} - y_c . M_{20} + 2.x_c{}^2 . M_{01} \\
&\mu_{12} = M_{12} - 2.y_c . M_{11} - x_c . M_{02} + 2.y_c{}^2 . M_{10} \\
&\mu_{30} = M_{30} - 3.x_c . M_{20} + x_c{}^2 . M_{10} \\
&\mu_{03} = M_{03} - 3.y_c . M_{02} + y_c{}^2 . M_{01}
\end{aligned} \tag{8.5}$$

### 8.1.2 Algumas propriedades da imagem

Com os momentos puros e centrais de até segunda ordem podem ser calculadas algumas propriedades interessantes da imagem. Já foi visto o cálculo do centro de massa da imagem na equação (8.3), que é usado para calcular os momentos centrais. O momento de ordem zero é igual à área da imagem, se essa for binária, ou ao somatório da luminância de todos os pixels. Assim, o cálculo das três primeiras propriedades da imagem pode ser visto no conjunto de equações (8.6).

$$\text{Área} = M_{00}; \qquad x_c = \frac{M_{10}}{M_{00}}; \qquad y_c = \frac{M_{01}}{M_{00}} \tag{8.6}$$

Com os momentos centrais, pode-se calcular a matriz de covariância da imagem, para em seguida calcular os autovalores desta matriz. O cálculo da matriz de covariância e dos autovalores da mesma é feito pela equação (8.7).

$$Cov = \begin{bmatrix} \frac{\mu_{20}}{\mu_{00}} & \frac{\mu_{11}}{\mu_{00}} \\ \frac{\mu_{11}}{\mu_{00}} & \frac{\mu_{02}}{\mu_{00}} \end{bmatrix} \Rightarrow \begin{cases} \lambda_1 = \dfrac{\mu_{02} + \mu_{20} + \sqrt{4.\mu_{11}^{\ 2} + (\mu_{20} - \mu_{02})^2}}{2.\mu_{00}} \\ \lambda_2 = \dfrac{\mu_{02} + \mu_{20} - \sqrt{4.\mu_{11}^{\ 2} + (\mu_{20} - \mu_{02})^2}}{2.\mu_{00}} \end{cases} \tag{8.7}$$

Os autovalores podem ser usados para se obter informações da orientação da imagem, definindo uma elipse circunscrita na mesma. Assim, os eixos $\boldsymbol{a_e}$ e $\boldsymbol{b_e}$ e o ângulo $\boldsymbol{\theta_e}$ do eixo maior da elipse com a horizontal são definidos pelos autovalores da matriz de covariância pela equação (8.8).

$$\begin{aligned} \mathrm{a}_{\mathrm{e}} &= 2.\sqrt{\lambda_1} \\ b_e &= 2.\sqrt{\lambda_2} \\ \theta_e &= \frac{1}{2}\tan^{-1}\left(\frac{2\mu_{11}^{\ 2}}{\mu_{20} - \mu_{02}}\right) \end{aligned} \tag{8.8}$$

### 8.1.3 Momentos normalizados e invariantes

Os momentos normalizados são invariantes à translação e à escala, podendo ser calculados a partir dos momentos centrais, usando a equação (8.9).

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\ 1+\frac{p+q}{2}}} \tag{8.9}$$

A partir dos momentos normalizados podem ser calculados os sete momentos invariantes de Hu [17], que além de invariantes à translação e à escala, são também invariantes à rotação. O cálculo dos momentos invariantes está mostrado no conjunto de equações (8.10).

$$\Phi_1 = \eta_{20} + \eta_{02}$$

$$\Phi_2 = \left(\eta_{20} - \eta_{02}\right)^2 + \left(2.\eta_{11}\right)^2$$

$$\Phi_3 = (\eta_{30} - 3.\eta_{12})^2 + (3.\eta_{21} - \eta_{03})^2$$

$$\Phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$

$$\Phi_5 = (\eta_{30} - 3.\eta_{12}).(\eta_{30} + \eta_{12}).[(\eta_{30} + \eta_{12})^2 - 3.(\eta_{21} + \eta_{03})^2] + \quad (8.10)$$
$$(3.\eta_{21} - \eta_{03}).(\eta_{21} + \eta_{03}).[3.(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

$$\Phi_6 = (\eta_{20} - \eta_{02}).[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 + 4.\eta_{11}(\eta_{30} + \eta_{12}).(\eta_{21} + \eta_{03})]$$

$$\Phi_7 = (3\eta_{21} - \eta_{03}).(\eta_{30} + \eta_{12}).[(\eta_{30} + \eta_{12})^2 - 3.(\eta_{21} + \eta_{03})^2] -$$
$$(\eta_{30} - 3.\eta_{12}).(\eta_{21} + \eta_{03}).[3.(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2].$$