



**UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO  
CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO**

Lenivaldo Ribeiro de Souza

**ALGORITMO PARA RECONHECIMENTO E  
ACOMPANHAMENTO DE TRAJETÓRIA DE PADRÕES EM  
VÍDEOS**

Juazeiro - BA

2011

**UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO  
CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO**

Lenivaldo Ribeiro de Souza

**ALGORITMO PARA RECONHECIMENTO E  
ACOMPANHAMENTO DE TRAJETÓRIA DE PADRÕES EM  
VÍDEOS**

Trabalho de Conclusão de Curso apresentado  
a Universidade Federal do Vale do São  
Francisco – UNIVASF, campus de Juazeiro,  
como requisito parcial para obtenção do título  
de Engenheiro da Computação.

Orientador: Brauliro Gonçalves Leal.  
Co-orientador: Eduard Montgomery Meira  
Costa.

Juazeiro - BA

2011

S729a Souza, Lenivaldo Ribeiro de.  
Algoritmo para reconhecimento e acompanhamento de  
trajetória de padrões em vídeos / Lenivaldo Ribeiro de Souza. --  
Juazeiro, 2011.  
XI; 78f. : il. 29 cm.

Trabalho de Conclusão de Curso (Graduação em  
Engenharia da Computação) - Universidade Federal do Vale do  
São Francisco, Campus Juazeiro, Juazeiro-BA, 2011.  
Orientador (a): Prof.(a) Dr. Brauliro Gonçalves Leal.

1. Reconhecimento e acompanhamento de trajetória de  
padrões em vídeos. 2. Redes Neurais Artificiais. I. Título. II.  
Universidade Federal do Vale do São Francisco.

CDD 006.4

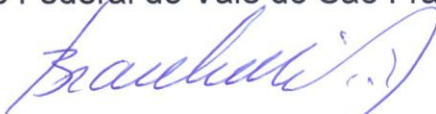
**UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO  
CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO**

**FOLHA DE APROVAÇÃO**

Lenivaldo Ribeiro de Souza

**ALGORITMO PARA RECONHECIMENTO E ACOMPANHAMENTO DE  
TRAJETÓRIA DE PADRÕES EM IMAGENS MÓVEIS**

Trabalho de Conclusão de Curso apresentado como requisito parcial  
para obtenção do título de Engenheiro da Computação, pela  
Universidade Federal do Vale do São Francisco – UNIVASF.



---

Brauliro Gonçalves Leal, Doutor, Professor da UNIVASF.



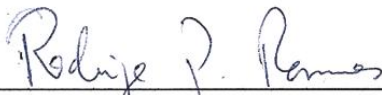
---

Eduard Montgomery Meira Costa, Doutor, Professor da UNIVASF.



---

Marcelo Santos Linder, Mestre, Professor da UNIVASF.



---

Rodrigo Pereira Ramos, Doutor, Professor da UNIVASF.

Aprovado pelo Colegiado de Engenharia da Computação em:

29 / 06 / 2011

Dedico essa grande conquista a minha família pela grande força proporcionada ao longo da minha vida acadêmica e a todos aqueles que de alguma forma acreditaram na minha capacidade de superação.

## **AGRADECIMENTOS**

Agradeço acima de tudo a Deus, pois sem ele nada disso seria possível.

À minha família pelo apoio moral e financeiro.

Ao orientador e co-orientador desse projeto de pesquisa.

À todos os professores de graduação.

## RESUMO

Este projeto visa o desenvolvimento de um algoritmo para reconhecimento e acompanhamento de trajetória de padrões em vídeos. Foram utilizadas técnicas de processamento digital de imagens, de reconhecimento de padrões a partir de Redes Neurais Artificiais (RNAs) e de detecção de trajetória. No que diz respeito à detecção de trajetória foi utilizado o algoritmo de rastreamento de cor conhecido como *Camshift*. Para a implementação dos algoritmos, foi utilizada como ferramenta computacional de apoio a biblioteca de programação OpenCV (Open Source Computer Vision). Portanto, por meio da escolha de uma linguagem de programação foi possível desenvolver um algoritmo que efetua a etapa de pré-processamento de imagens, o reconhecimento de um padrão nela contido através de Redes Neurais Artificiais e o acompanhamento desse objeto em uma sequência sucessiva de imagens.

Para tanto, inicialmente foi realizado um estudo de estruturas de geração de vídeos digitais, redes neurais artificiais e aplicações de reconhecimento de padrões em vídeos, estimadores e programação em C/C++. Após a instalação e configuração das ferramentas de programação, foram elaborados exemplos para testá-las. Em seguida, efetuou-se a análise e definição dos procedimentos a serem utilizados para estruturação dos algoritmos para, finalmente, implementá-los utilizando linguagem de programação C/C++.

**Palavras-chave:** Redes Neurais. Padrões. OpenCV. Rastreamento. Reconhecimento.

## **ABSTRACT**

This project aims to develop an algorithm for trajectory tracking and recognition of patterns in videos. Techniques were used for digital image processing, pattern recognition from Artificial Neural Networks (ANN) and detection path. With respect to the detection path was used tracking algorithm with known as color Camshift. To implement the algorithms was be used as a computational tool to support library programming OpenCV (Open Source Computer Vision). Therefore, by choosing a programming language was possible to develop an algorithm that performs the step of pre-processing of images, the recognition of a pattern within it through Artificial Neural Networks and monitoring of this object in a sequence of successive images.

For this purpose it was originally performed a study of structure generation digital video, artificial neural networks and pattern recognition applications in videos, estimators and programming in C/C++. After installation and configuration of the programming tools, examples were designed to test them. Then we performed the analysis and definition of procedures to be used for the structuring of algorithms to finally implement them using the programming language C/C++.

**Keywords:** Neural Networks. Standards. OpenCV. Trace. Recognition.



## LISTA DE FIGURAS

<i>Figura 1. Representação numérica de uma imagem ampliada de 10x10 pixels. ....</i>	<i>19</i>
<i>Figura 2. Elementos do processo de análise da imagem (Gonzalez and Woods, 2000). ....</i>	<i>20</i>
<i>Figura 3. Representação artificial de um neurônio natural (Cordeiro, 2002). ....</i>	<i>26</i>
<i>Figura 4. Funções de ativação comumente usadas. ....</i>	<i>27</i>
<i>Figura 5. Arquitetura de uma rede neural (Silva e Leal, 2002). ....</i>	<i>28</i>
<i>Figura 6. Esquema do funcionamento do Algoritmo Backpropagation. ....</i>	<i>30</i>
<i>Figura 7. Superfície de erro para um treinamento usando backpropagation (Braga et al, 2000). ....</i>	<i>31</i>
<i>Figura 8. Exemplo de rede MLP. ....</i>	<i>32</i>
<i>Figura 9. Problemas linearmente e não-linearmente separáveis. ....</i>	<i>33</i>
<i>Figura 10. Seqüência de quadros de um vídeo mostrando a movimentação de um círculo. ....</i>	<i>43</i>
<i>Figura 11. Fluxograma com as etapas de captura de quadros de vídeo ....</i>	<i>44</i>
<i>Figura 12. Tela de saída de vídeo capturado de uma câmera ....</i>	<i>45</i>
<i>Figura 13. Imagens resultantes do algoritmo de processamento de imagens ....</i>	<i>49</i>
<i>Figura 14. Arquivo com saídas do processamento do quadro 0.....</i>	<i>49</i>
<i>Figura 15. Trecho de código mostrando função de treinamento ....</i>	<i>51</i>
<i>Figura 16. Exemplo de arquivo de configurações e pesos de uma RNA ....</i>	<i>52</i>
<i>Figura 17. Esquema de blocos para o de reconhecimento de padrões ....</i>	<i>53</i>
<i>Figura 18. Algoritmo Cam Shift (INTEL Corporation,2009). ....</i>	<i>55</i>
<i>Figura 19. Menu principal.....</i>	<i>56</i>
<i>Figura 20. Menu treinar RNA.....</i>	<i>56</i>
<i>Figura 21. Menu treinar círculo ....</i>	<i>57</i>
<i>Figura 22. Exemplo de arquivo txt com entradas para treinamento.....</i>	<i>57</i>
<i>Figura 23. Sub-menu reconhecer um padrão ....</i>	<i>57</i>
<i>Figura 24. Reconhecer padrão por vídeo.....</i>	<i>58</i>
<i>Figura 25. Tela para escolha de vídeos.....</i>	<i>59</i>
<i>Figura 26. Reconhecer padrão por WebCam.....</i>	<i>59</i>
<i>Figura 27. Menu acompanhar um padrão.....</i>	<i>60</i>
<i>Figura 28. Acompanhar um padrão por arquivo de vídeo.....</i>	<i>60</i>
<i>Figura 29. Acompanhar um padrão por WebCam.....</i>	<i>61</i>
<i>Figura 30. Exemplo de tela de acompanhamento de padrão.....</i>	<i>61</i>
<i>Figura 31. Etapas resumidas do algoritmo.....</i>	<i>62</i>
<i>Figura 32. Resultados do processamento de quadros ....</i>	<i>62</i>
<i>Figura 33. Resultados para etapa de rastreamento.....</i>	<i>63</i>

## **LISTA DE ABREVIATURAS E SIGLAS**

AVI – Audio Video Interleave

FPS – Frames Per Second

MLP – Multi-layer Perceptron

OpenCV – Open Source Computer Vision

RNA – Rede Neural Artificial

RGB – Red, Green and Blue

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO .....</b>	<b>12</b>
<b>2</b>	<b>JUSTIFICATIVA .....</b>	<b>15</b>
<b>3</b>	<b>OBJETIVOS .....</b>	<b>17</b>
<b>4</b>	<b>REVISÃO BIBLIOGRÁFICA .....</b>	<b>18</b>
4.1	PROCESSAMENTO DE IMAGENS DIGITAIS .....	18
4.1.1	Aquisição de imagem .....	20
4.1.2	Pré-processamento .....	20
4.1.3	Segmentação .....	21
4.1.4	Representação e descrição .....	21
4.1.5	Reconhecimento .....	22
4.1.6	Interpretação.....	22
4.2	FORMATO DE VÍDEO AVI.....	23
4.3	REDES NEURAIS ARTIFICIAIS .....	25
4.3.1	Algoritmo <i>Backpropagation</i> .....	30
4.3.2	Rede MLP ( <i>Multilayer Perceptron</i> ) .....	32
4.4	RECONHECIMENTO DE PADRÕES.....	33
4.5	RASTREAMENTO .....	34
4.6	BIBLIOTECA OPENCV .....	36
<b>5</b>	<b>MATERIAIS E MÉTODOS .....</b>	<b>39</b>
5.1	DELINEAMENTO METODOLÓGICO.....	39
<b>6</b>	<b>RESULTADOS OBTIDOS .....</b>	<b>44</b>
6.1	ETAPA DE PROCESSAMENTO DAS IMAGENS .....	46
6.2	ETAPA DE TREINAMENTO .....	50
6.3	ETAPA DE RECONHECIMENTO .....	52
6.4	ETAPA DE RASTREAMENTO .....	53
6.5	COMO UTILIZAR OS ALGORITMOS .....	55
6.6	DISCUSSÃO DOS RESULTADOS .....	61
<b>7</b>	<b>CONCLUSÃO.....</b>	<b>65</b>
<b>8</b>	<b>REFERÊNCIAS.....</b>	<b>66</b>
	<b>ANEXOS.....</b>	<b>69</b>

## 1 Introdução

O processamento de imagens ganha novo papel na sociedade com a popularização de equipamentos capazes de gerar, armazenar e transmitir vídeos. Dessa forma, a utilização de métodos para obter informações automaticamente de vídeos e de seqüência de imagens está se tornando cada vez mais comum. As informações extraídas de um vídeo podem ser utilizadas de diversas formas como a geração de estatísticas, classificação, reconhecimento de pessoas ou objetos.

O reconhecimento de padrões faz parte do nosso contato diário. O simples fato de ler esse texto já envolve o reconhecimento de cada símbolo como sendo uma letra do alfabeto, que por sua vez em conjunto são reconhecidos como palavras que apresentam algum significado. Enfim, reconhecer o rosto de uma pessoa, distinguir um cachorro de um gato, compreender a fala, ler as mais diversas caligrafias e até mesmo interpretar um exame de eletrocardiografia, tudo isso é reconhecimento de padrões (Henrique, 2003). Atualmente, com o desenvolvimento das áreas relacionadas à visão computacional, psicologia, neurociências e inteligência artificial, o campo de reconhecimento automático de objetos tem procurado se aproximar cada vez mais do sistema visual humano.

Já existe uma boa quantidade de trabalhos desenvolvidos relacionados com o reconhecimento de padrões e suas aplicações, tais como: identificação através de impressões digitais e análise da íris (Steiner, 1995); análise de imagens aeroespaciais (Perelmuter et al., 1995); investigação da qualidade do papel industrial (Steiner, 1995); análise de caracteres manuscritos (Prado, 1975); análise de eletrocardiogramas (Mascarenhas, 1987); detecção da Bola em Vídeos de Futebol (Guimarães); reconhecimento em tempo real de agentes autônomos em Futebol de robôs (Schwartz et al., 2003); reconhecimento e identificação de cromossomos (Todesco, 1995).

Alguns exemplos de trabalhos correlatos são os anais apresentados no VII WORKSHOP DE VISÃO COMPUTACIONAL (2011): Sistema de Detecção e Indexação Automática de Objetos em Vídeos Digitais, Reconhecimento Multibiométrico de Pessoas em Imagens de Vídeo para Monitoramento Visual de

Ambientes, e Modelagem Contextual de Padrões Aplicada ao Monitoramento de Eventos em Vídeo.

A palavra “reconhecimento” tem sido usada largamente na literatura para significar entendimento, interpretação, classificação, cognição e outras tarefas inerentemente humanas.

Entende-se por padrão as propriedades que possibilitam o agrupamento de objetos semelhantes dentro de uma determinada classe ou categoria, mediante a interpretação de dados de entrada, que permitam a extração das características relevantes desses objetos (Tou e González, 1981). Assim, reconhecimento de padrões pode ser definido como sendo um procedimento em que se busca a identificação de certas estruturas nos dados de entrada em comparação a estruturas conhecidas e sua posterior classificação dentro de categorias.

Para resolver problemas relacionados ao reconhecimento de padrões, a abordagem comumente utilizada é dividir o problema em dois módulos subseqüentes: a) de extração de características; e b) de classificação. Tanto para o primeiro como para o segundo módulo já existem várias técnicas desenvolvidas que vão desde técnicas estatísticas até técnicas de Inteligência Artificial (Artur, 1999).

Na maioria dos casos, os métodos de reconhecimento necessitam de um pré-processamento da imagem antes do passo final de reconhecimento em si. Portanto, é importante contar com o auxílio de ferramentas computacionais tais como OpenCV para simplificação do problema.

Acompanhar a trajetória de um padrão em vídeos significa rastreá-lo, isto é, reconhecê-lo em uma seqüência de imagens. Tendo em vista que a busca em cada imagem de uma seqüência sem o uso de qualquer conhecimento específico é relativamente lenta, torna-se necessário um conhecimento sobre o movimento do objeto que está sendo rastreado para minimizar a busca entre as imagens em uma seqüência.

A Inteligência Artificial é um dos campos da ciência da computação que se ocupa da automação do comportamento inteligente. As redes neurais constituem um dos ramos da inteligência artificial mais bem sucedidos no reconhecimento de padrão, cujo campo de aplicação é bastante amplo. De acordo com Braga et. al

(2000), a solução através de Redes Neurais Artificiais (RNAs) é bastante atrativa. Sendo que o procedimento usual na solução de um problema passa inicialmente por uma etapa de aprendizagem, em que um conjunto de exemplos é apresentado à rede, a qual extrai automaticamente as características necessárias para representar a informação fornecida. Esse é, sem dúvidas, o atrativo principal da solução de problemas através de RNAs.

Portanto, este projeto apresenta um algoritmo para reconhecimento e acompanhamento de trajetória de padrões em vídeos. Para tanto, serão utilizadas as vantagens proporcionadas pelas redes neurais artificiais para o reconhecimento de padrões e de algoritmos de rastreamento, tais como *Camshift* e *Meanshift*, para o monitoramento de trajetória.

O escopo desse trabalho de conclusão de curso compreenderá objetos de geometria simples (círculos, retângulos e triângulos) e monocromáticos presentes em arquivos de vídeos com cenas bidimensionais de fundo monocromático (com cor diferente da do objeto a ser identificado) e com luminosidade suficiente para a sua identificação e acompanhamento de sua trajetória. O objeto deverá estar presente completamente na cena. O ambiente será do tipo dinâmico onde apenas o objeto observado se desloca no vídeo, sendo que esse deslocamento poderá também ocorrer em decorrência da movimentação da câmera.

## 2 Justificativa

Optou-se pela técnica de reconhecimento utilizando Redes Neurais Artificiais porque elas constituem um dos ramos da Inteligência Artificial mais bem sucedidos no reconhecimento de padrões, com amplo campo de aplicação. O procedimento usual na solução de um problema passa inicialmente por uma etapa de aprendizagem, em que um conjunto de exemplos é apresentado à rede, a qual extrai automaticamente as características necessárias para representar a informação fornecida. Apesar de alguns autores verem essa etapa como sendo lenta e computacionalmente dispendiosa, ela possibilita a vantagem de uma execução rápida posteriormente, isto é, depois de devidamente treinada a rede é capaz de reconhecer rapidamente um padrão. Esse é, sem dúvidas, o atrativo principal da solução de problemas através de RNAs.

A utilização de métodos para obter informações automaticamente de vídeos e de seqüência de imagens está se tornando cada vez mais comum. A localização de objetos em uma imagem e acompanhamento de seu deslocamento numa seqüência de imagens são tarefas de interesse teórico e prático, visto que a maioria dos algoritmos já desenvolvidos se aplica a imagens estáticas. Aplicações de reconhecimento e rastreamento de padrões e objetos tem se difundido ultimamente, principalmente no ramo de controle, automação e vigilância.

A grande quantidade de dados aliada com a rapidez exigida para que as novas informações sejam geradas e distribuídas abre caminho para o processamento automático desses dados. As informações extraídas da seqüência de vídeos são utilizadas das mais variadas formas. A geração de estatísticas, classificação, reconhecimento de pessoas ou objetos são alguns exemplos de aplicações apresentadas pelos algoritmos.

Dentre os fatores motivadores para o desenvolvimento desse projeto está a enorme quantidade de aplicações para a localização de objetos em uma imagem e acompanhamento de seu deslocamento numa seqüência de imagens. Dentre as diversas utilidades, pode-se destacar:

- Aplicações na medicina para análise de radiografias, diagnóstico de doenças, identificação de células, simulação de funções cerebrais;
- Monitoramento de objetos a partir de imagens geradas por uma câmera;
- Aplicações na área de Robótica;
- Caracterização de rochas, prospecção mineral, sensoriamento remoto;
- Detecção de alvos, classificação de sinais de radar;
- Efeitos especiais, animação;
- Visão computacional, controle de manipuladores, análise de situações;
- Simulação de processos químicos, processamento de sinal, diagnóstico de falhas;
- Análise de texturas e fases.

Portanto, os resultados a serem obtidos no desenvolvimento do presente projeto abrangem um amplo contexto, podendo ser aplicado em várias áreas e setores da atividade humana tais como: segurança pública e privada, medicina, indústria, engenharia, entre outras possibilidades. Em sistemas bélicos, é de fundamental importância o correto rastreamento de possíveis alvos. Na indústria, utilizam-se tais sistemas para verificação da qualidade de produtos bem como para acompanhamento dos mesmos na linha de produção. No ramo de segurança, a detecção e rastreamento de pessoas podem ajudar a identificar a presença de indivíduos não autorizados.



### 3 Objetivos

Este projeto teve como objetivo geral propor um algoritmo para reconhecimento e acompanhamento de trajetória de padrões em vídeos, através do uso de técnicas para extração de características, para reconhecimento e para monitoramento de um determinado objeto em arquivos de vídeo. Para tanto, obteve-se um contexto de similaridade dessas características para o treinamento de redes neurais artificiais para auxiliar na tarefa de detecção do padrão pretendido.

Partindo-se desse objetivo geral, estabeleceram-se os seguintes objetivos específicos:

- Desenvolver um algoritmo para capturar e aplicar operações de processamento de imagem em um determinado quadro ou seqüências de quadros de um vídeo;
- Efetuar reconhecimento de um padrão em uma seqüência de quadros utilizando rede neural;
- Implementar um algoritmo para acompanhar a trajetória de um determinado padrão em uma seqüência de quadros de um vídeo.

Ao final deste trabalho, obteve-se uma experiência prática com as técnicas de reconhecimento e acompanhamento de trajetória de objetos em arquivos de vídeo, contribuindo então para a ampliação de conhecimentos profissionais e desenvolvimento de capacidades de criatividade e pesquisa. Os objetos ou padrões mencionados anteriormente consistem de figuras geométricas simples (círculos, retângulos e triângulos) presentes em seqüências de quadros de arquivos de vídeos.

## 4 Revisão bibliográfica

### 4.1 Processamento de imagens digitais

Segundo Maria (2000), entende-se por Processamento Digital de Imagens a manipulação de uma imagem por computador de modo que a entrada e a saída do processo sejam imagens. O objetivo de se utilizar essa técnica é melhorar o aspecto visual de certas feições estruturais para o analista humano e fornecer outros subsídios para a sua interpretação, inclusive gerando produtos que possam ser posteriormente submetidos a outros processamentos.

A visão computacional procura imitar o comportamento da visão humana e, portanto, também possui como entrada uma imagem, no entanto, a saída é uma interpretação da imagem completa ou de parte dela.

De acordo com Gonzalez and Woods (2000), uma imagem pode ser definida como uma função  $f(x, y)$ , onde o valor nas coordenadas espaciais  $x$  e  $y$  corresponde ao brilho (intensidade) da imagem nessa coordenada.

Para que uma imagem seja representada em um computador, é necessária a sua digitalização tanto no domínio espacial como no das amplitudes. Dessa forma, uma imagem digital é a representação numérica e discreta de um objeto, ou especificamente, é uma função quantificada e amostrada, de duas dimensões, geradas por meios ópticos, disposta em uma grade padrão, retangular, igualmente espaçada, quantificada em iguais intervalos de amplitude. Assim, conforme Cordeiro (2002), uma imagem digital é um vetor retangular bidimensional de amostras de valores quantificados.

A menor unidade que constitui uma imagem digital é denominada *picture element* (pixel). Um pixel é a representação numérica da luminosidade de um ponto da imagem. A Figura 1 ilustra a representação numérica de uma imagem ampliada de 10 X 10 pixels.

Tipicamente, cada ponto de uma imagem é decomposto em uma tripla de cores e cada proporção relativa é transformada em valores numéricos que permitem

que eles sejam recuperados. No modelo conhecido como RGB, por exemplo, a imagem é decomposta nas cores vermelho (R-red), verde (G-Green) e azul (B-Blue).

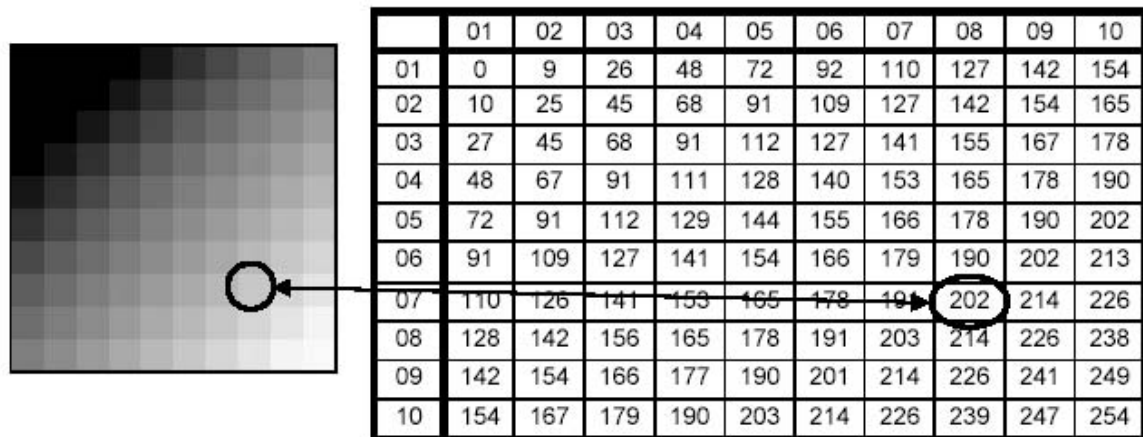


Figura 1. Representação numérica de uma imagem ampliada de 10x10 pixels.

O modelo de cores RGB é um modelo aditivo no qual o vermelho, o verde e o azul são combinados de várias maneiras para reproduzir outras cores. Cada uma pode variar entre o mínimo (completamente escuro) e máximo (completamente intenso). Geralmente, a partição é de 8 bits para cada uma das 3 cores, dando um alcance de 256 possíveis valores, ou intensidades, para cada tom.

O ato de processar uma imagem consiste em aplicar sobre ela transformações sucessivas com o objetivo de extrair mais facilmente a informação nela contida de modo que o resultado final seja mais adequado que a imagem original para uma aplicação específica.

De acordo com Gonzalez and Woods (2000), as técnicas em análise de imagem podem ser divididas em três áreas básicas: processamento de baixo nível, processamento de nível intermediário e processamento de alto nível. A Figura 2 mostra os processos de cada uma dessas áreas.

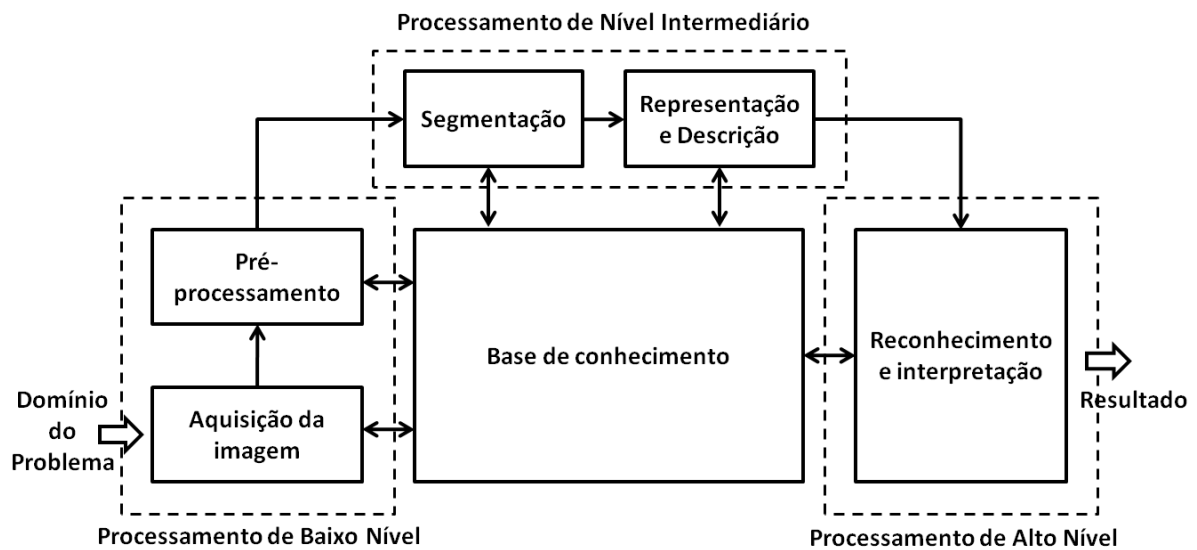


Figura 2. Elementos do processo de análise da imagem (Gonzalez and Woods, 2000).

Nas seções seguintes, serão apresentados alguns dos passos para o processamento digital de imagens com enfoque no reconhecimento de imagens, tais como: aquisição de imagem, pré-processamento, segmentação, representação e descrição, reconhecimento e interpretação.

#### 4.1.1 Aquisição de imagem

É o primeiro passo do processo. Segundo Gonzalez and Woods (2000), requer apenas um sensor de imagens e a capacidade para digitalizar o sinal produzido pelo sensor. Portanto, são necessários dois elementos: um aparelho físico que é sensível à faixa espectral de energia eletromagnética e que gera um sinal elétrico de saída e um digitalizador, que converte o sinal elétrico capturado na sua forma digital.

#### 4.1.2 Pré-processamento

Após a aquisição e digitalização da imagem, o próximo passo é o pré-processamento que visa melhorar a imagem, aumentando as chances de sucesso dos processos seguintes (Gonzalez and Woods, 2000). Nesta etapa, são utilizadas técnicas para aumento de contraste, remoção de ruídos, realce e normalização, com

o objetivo de converter os padrões para uma forma que possibilite uma simplificação do posterior processo de reconhecimento.

As técnicas utilizadas nessa etapa envolvem duas categorias principais: métodos que operam no domínio espacial e métodos que operam no domínio da frequência. O domínio espacial refere-se ao próprio plano da imagem, e as técnicas nesta categoria são baseadas na manipulação direta dos pixels de uma imagem. As técnicas de processamento no domínio da frequência se baseiam na modificação da transformada de Fourier de uma imagem.

Algumas das técnicas que podem ser aplicadas nessa etapa são: suavização espacial (homogeneização de pixels através de filtros tais como passa-baixa e passa - alta), realce de bordas e detecção de bordas.

#### **4.1.3 Segmentação**

De uma forma geral, a segmentação consiste em subdividir uma imagem de entrada em suas partes constituintes ou objetos. Cada uma destas partes é uniforme e homogênea com respeito a algumas propriedades da imagem, como por exemplo, cor e textura. Algoritmos de segmentação para imagens monocromáticas são geralmente baseados em duas propriedades básicas de valores em escala de cinza: descontinuidade e similaridade. Na descontinuidade, o particionamento da imagem é baseado no subconjunto de pontos de um objeto que o separa do restante da imagem. Na similaridade, a segmentação é baseada nas técnicas de limiarização, crescimento por regiões, união e divisão de regiões (Gonzalez and Woods, 2000).

#### **4.1.4 Representação e descrição**

Geralmente, a saída do estágio de segmentação são dados brutos de pixel. Neste caso, pode ser necessário converter os dados para uma forma conveniente, possibilitando o processamento por computador. Dois tipos de representação podem ser utilizados: representação limite (focaliza características da forma) ou representação regional (focaliza as propriedades refletivas). Escolher a

representação é apenas parte da solução para a transformação de dados brutos em uma forma conveniente para o processamento computacional subsequente.

Um método para descrever os dados de tal forma que as características de interesse sejam realçadas também deve ser utilizado. Descrição, também chamada de seleção de característica, lida com a extração de características que resultam em diferenciar uma classe de objetos de outra (Gonzalez and Woods, 2000).

#### **4.1.5 Reconhecimento**

Depois de feita a separação de classes de objetos com características semelhantes, o próximo passo é identificar o que cada uma dessas classes representa, e assim identificá-las com um respectivo valor.

As técnicas de reconhecimento de padrões existentes atualmente podem ser divididas em três abordagens: estatística, estrutural e neural. A abordagem estatística consiste em extrair das imagens um conjunto de medidas de características (na forma de n-uplas ou vetores) e aplicar métodos estatísticos para separar as classes. Na abordagem estrutural, os padrões são representados em uma forma simbólica (tais como strings e árvores), e os métodos de reconhecimento são baseados em casamento de símbolos ou em modelos que tratam padrões de símbolos como sentenças, a partir de uma linguagem artificial. Na abordagem neural, o reconhecimento é realizado utilizando-se Redes Neurais Artificiais.

Alguns autores consideram o reconhecimento via redes neurais como sendo um tipo particular de reconhecimento estatístico, já que as características também estão na forma de n-uplas ou vetores e existe uma equivalência entre alguns modelos de redes neurais e técnicas estatísticas fundamentais.

#### **4.1.6 Interpretação**

Para Gonzalez and Woods (2000), interpretação envolve a fixação de significado a um grupo de objetos reconhecidos, ou seja, o interesse está em dar significados à imagem.

Interpretar uma imagem computadorizada é um processo extremamente complexo. As dificuldades aparecem tanto pela quantidade de dados a serem processados como também pela falta de ferramentas de processamento fundamental para receber os dados iniciais a fim de gerar os resultados desejados (detalhamento do conteúdo da imagem).

## **4.2 Formato de vídeo AVI**

Após definidos alguns conceitos sobre imagens digitais e apresentados os passos para processamento de imagens, serão destacadas, em seguida, algumas informações sobre o padrão de vídeo utilizado durante o desenvolvimento desse trabalho de conclusão de curso.

Um vídeo consiste numa sucessão de imagens em um determinado intervalo de tempo. O olho humano tem como característica ser capaz de distinguir cerca de 20 imagens por segundo. Assim, fixando mais de 20 imagens por segundo, é possível superar a capacidade do olho e induzi-lo a ver uma imagem animada. Dessa forma, caracteriza-se a fluidez de um vídeo pelo número de imagens por segundo ou FPS (*Frames per Second*).

Um arquivo de vídeo é um formato de arquivo de computador que pode conter vários tipos de dados, comprimido por meios estáveis utilizando *codecs* de vídeo e/ou áudio que são aplicativos responsáveis por fazer a codificação e decodificação de determinados arquivos de mídia. Formatos mais simples podem conter diferentes tipos de *codecs* de áudio, enquanto os mais avançados podem suportar múltiplas transmissões de áudio e vídeo, legendas, informações sobre capítulos e metadados, além da informação que é necessária para sincronização entre várias transmissões.

O padrão AVI (Audio Video Interleave), cuja extensão oficial é “.avi”, trata-se de um formato criado pela empresa Microsoft. De acordo com a Microsoft Corporation (1997) esse formato é um caso especial de arquivo RIFF (Resource Interchange File Format – Formato de arquivo com intercâmbio de recursos) capaz de encapsular áudio e vídeo.

É possível guardar no formato AVI uma faixa de vídeo codificada em um codec qualquer e nessa mesma faixa é possível associar um áudio em MP3. Nele, podemos na realidade armazenar vários tipos de informação diferentes tais como várias seqüências de vídeo e som. Cada seqüência é chamada de "Stream" e pode estar compactada com qualquer método de compressão desde que esse método (ou codec) esteja instalado no sistema operacional. Assim pode ocorrer do vídeo ser compactado em um sistema e não conseguir ser reproduzido em outro pela falta do codec.

Um vídeo do tipo AVI é formado por várias imagens. Cada imagem é chamada *quadro* e a quantidade de imagens projetadas por segundo é chamada *cadência*, medida em FPS. Quanto mais quadros por segundo o vídeo tiver, melhor, pois mais realista será a imagem. Vídeos normalmente trabalham com a mesma cadência da TV, que é de 30 quadros por segundo.

É possível conhecer o número de bytes transferidos por unidade de tempo de um determinado vídeo multiplicando-se o tamanho de uma imagem (frame) pelo número de imagens por segundo (ou FPS). Dessa forma, uma maneira de diminuir o tamanho do vídeo é justamente diminuindo a quantidade de quadros por segundo. O tamanho do vídeo diminui, mas sua qualidade também: há "quebras de quadro", isto é, os movimentos no vídeo ficam "truncados", menos realistas.

Para diminuir o tamanho do vídeo, é usada uma técnica de compressão de imagem, que funciona removendo das imagens informações que já foram projetadas. Esta técnica economiza uma quantidade enorme de espaço, já que somente o primeiro quadro precisa estar completo, os demais só têm o que é diferente do quadro anterior. Esses quadros incompletos são chamados *quadros delta* (delta frames).

Tendo em vista a grande popularidade desse padrão de vídeo e que a biblioteca OpenCV oferece suporte somente para esse formato em particular, ele será escolhido para o desenvolvimento do projeto. Para tanto, foram utilizados arquivos de vídeo no formato AVI (40 x 30), com duração entre 10 e 14 segundos, 12 quadros/s e tamanho entre 14 e 25 KB, para treinamento e validação da rede neural.



### 4.3 Redes Neurais Artificiais

Partindo-se do conhecimento das características e do princípio de funcionamento de um arquivo de vídeo, é possível escolher uma ferramenta capaz de assimilar algum tipo de informação presente em uma seqüência de imagens e poder realizar o reconhecimento de um determinado tipo de padrão.

Um dos ramos da inteligência artificial que vem se desenvolvendo bastante ultimamente é o de redes neurais (Cordeiro, 2002). De acordo com Artur (1999), seu crescente uso se deve à capacidade de fazer suposições mais delicadas a respeito da distribuição dos dados de entrada do que métodos estatísticos tradicionais e à capacidade de auxiliar na resolução de problemas de natureza não-linear.

Segundo Haykin (2001), “uma rede neural é um processador maciçamente paralelamente distribuído constituído de unidades de processamento simples, que têm a propensão natural para armazenar conhecimento experimental e torná-lo disponível para uso (...)”.

Segundo Ludwig e Costa (2007) as RNAs são, provavelmente, a mais antiga técnica de Inteligência Artificial em uso. A era moderna das redes neurais começou, segundo Braga et. al (2000), com o trabalho pioneiro de McCulloch e Walter Pitts em 1943. McCulloch foi um psiquiatra e neuroanatomista por treinamento; passou cerca de 20 anos refletindo sobre a representação de um evento no sistema nervoso. Pitts foi um prodígio matemático que se associou a McCulloch em 1942. A intenção era fazer uma analogia entre neurônios biológicos e circuitos eletrônicos.

De acordo com Haykin (2001), no artigo escrito em 1943, os autores McCulloch e Pitts descrevem um cálculo lógico das redes neurais que unificava os estudos da neurofisiologia e da lógica matemática. Com um número suficiente de neurônios artificiais e com conexões sinápticas ajustadas apropriadamente e operando de forma síncrona, os dois autores mostraram que uma rede assim construída realizaria, a princípio, a computação de qualquer função computável. Este era um resultado muito significativo e com ele é geralmente aceito o nascimento das disciplinas de redes neurais e inteligência artificial.

Uma RNA é caracterizada por uma arquitetura, um método de aprendizado ou treinamento, uma função de ativação e um *bias*. Informalmente ela é um sistema composto por vários neurônios. Estes neurônios estão ligados por conexões, chamadas conexões sinápticas. Alguns neurônios recebem excitações do exterior e são chamados neurônios de entrada e correspondem aos neurônios dos órgãos dos sentidos. Outros têm suas respostas usadas para alterar, de alguma forma, o mundo exterior e são chamados neurônios de saída e correspondem aos motoneurônios que são os neurônios biológicos que excitam os músculos. Os neurônios que não são nem entrada nem saída são conhecidos como neurônios internos. Estes neurônios internos à rede têm grande importância e são conhecidos por alguns como ocultos.

Um neurônio artificial é uma estrutura relativamente simples que responde a estímulos de outros neurônios conectados a ele. Para Cordeiro (2002), essa arquitetura dá às redes neurais características marcantes de intenso paralelismo e robustez. A Figura 3 mostra a representação artificial de um neurônio natural.

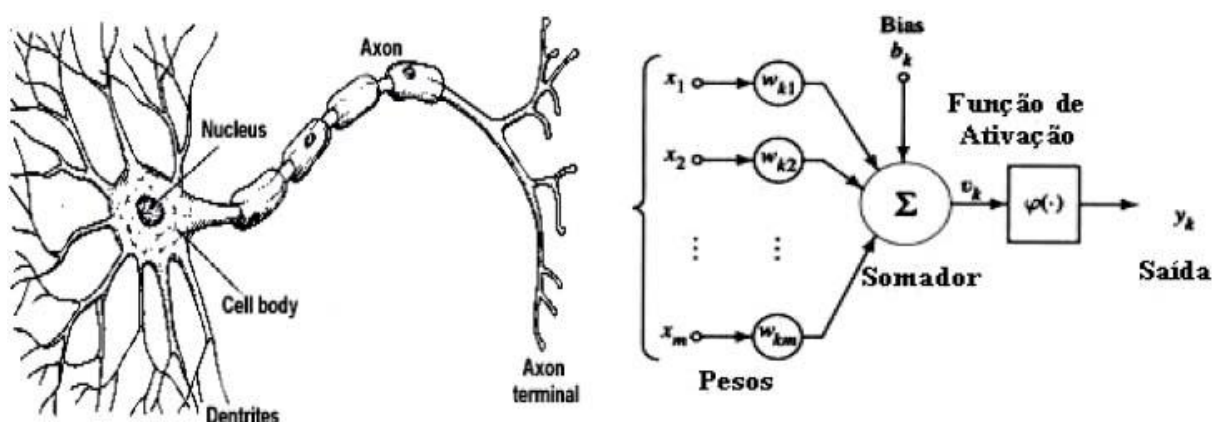


Figura 3. Representação artificial de um neurônio natural (Cordeiro, 2002).

Pela observação da Figura 3, percebe-se que um neurônio artificial é composto, basicamente, por um conjunto de entradas  $x_1, x_2, \dots, x_m$ , uma saída  $y_k$ , uma função responsável pelo cálculo da entrada efetiva para o neurônio ( $\Sigma$ ) denominada somador, uma função de ativação  $\varphi(\cdot)$ , os pesos e o *bias*. As entradas  $x_1, x_2, \dots, x_m$  representam os sinais vindos de outros neurônios. Os pesos  $w_{k1}, w_{k2}, \dots$ ,

$w_{km}$  representam os pesos sinápticos das conexões entre os neurônios da camada anterior e da camada  $k$ .

O neurônio pode ainda apresentar um *bias* ( $b_k$ ) que tem o efeito de aumentar ou diminuir a entrada líquida da função de ativação, ou seja, o termo *bias* age como um peso extra, estímulo, nas conexões das unidades cuja entrada é sempre um.

Matematicamente, um neurônio  $k$  seria descrito pelas seguintes equações:

$$u_k = \sum_{j=1}^m w_{kj} \cdot x_j \quad (1)$$

$$v_k = \sum_{j=1}^m w_{kj} \cdot x_j + b_k \quad (2)$$

$$y_k = \varphi(u_k + b_k) = \varphi(v_k) \quad (3)$$

Uma vez que a entrada para um determinado neurônio é calculada, o valor resultante é comparado a um limiar (valor estipulado) que, uma vez atingido, propaga a saída para os neurônios da camada seguinte. Nesse processo, a função de ativação é muito importante, pois ela é usada para restringir a amplitude de saída de um neurônio. Tipicamente, o intervalo normalizado da amplitude de saída de um neurônio é escrito como o intervalo unitário fechado  $[0,1]$  ou alternativamente  $[-1,1]$  (Haykin, 2001). A não linearidade do neurônio freqüentemente é introduzida na função de ativação. A Figura 4 mostra algumas funções identificadas, respectivamente, como: função rampa, em degraus e tangente hiperbólica.

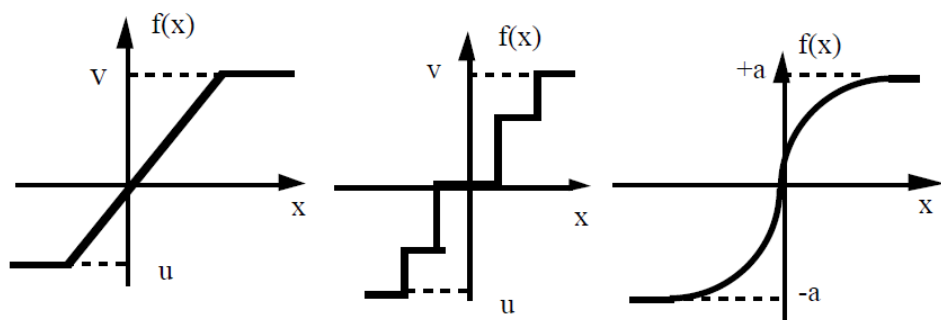


Figura 4. Funções de ativação comumente usadas.

A arquitetura de uma RNA é definida pela maneira com a qual os neurônios estão estruturados. Em geral, pode-se identificar três classes de arquiteturas de rede fundamentalmente diferentes (Haykin, 2001): Redes Alimentadas Adiante com

Camada Única, Redes Alimentadas Diretamente com Múltiplas Camadas e Redes Recorrentes (possui pelo menos um laço e uma camada de nós).

No caso mais comum, os neurônios são separados em camadas (conforme mostra a Figura 5), de modo que aqueles pertencentes a uma dada camada possam conectar-se unicamente aos da camada imediatamente anterior e emitem seu sinal exclusivamente aos neurônios da camada imediatamente posterior.

Para o reconhecimento de padrões, o treinamento da rede é realizado corrigindo os pesos nas conexões para se estabelecer as relações entre as características e classes que promovam a melhor discriminação possível entre os padrões de classes diferentes. Assim, ao ser apresentado à rede um novo padrão, esta indicará a classe que melhor o representa na camada de saída (Artur, 1999).

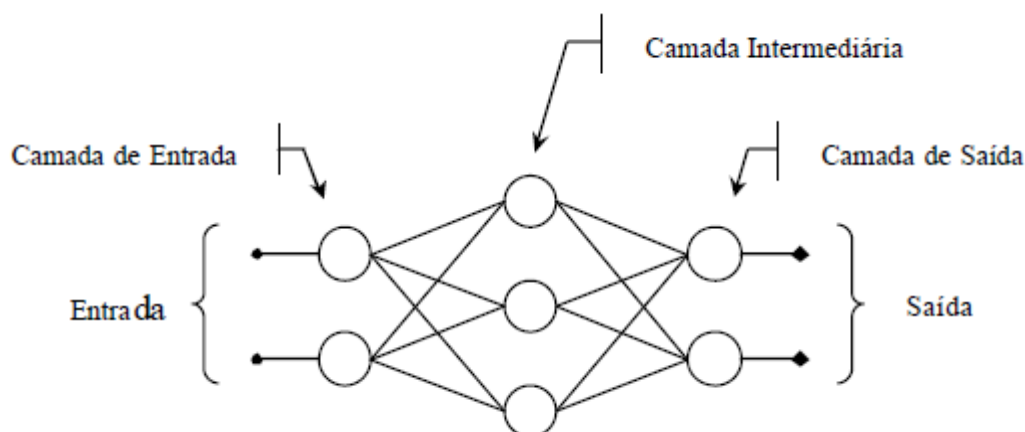


Figura 5. Arquitetura de uma rede neural (Silva e Leal, 2002).

O conhecimento numa rede neural artificial não está armazenado em locais determinados. De acordo com Carlos (1994), o conhecimento está armazenado na topologia e nos pesos. Portanto, os pesos representam o conhecimento da rede e determinam a ponderação que determinada entrada possui. Quanto mais estimulada uma conexão sináptica, mais freqüente a atualização do respectivo peso e, conseqüentemente, maior a influência desta entrada na produção da saída.

O processo de aprendizado de uma rede neural artificial acontece, basicamente, de duas formas: aprendizado supervisionado e aprendizado não-supervisionado. O tipo de aprendizagem é determinado pela maneira com a qual a modificação dos parâmetros ocorre (Haykin, 2001).

No aprendizado não-supervisionado, nenhum conjunto de pares de entrada e saída é apresentado à rede. Neste caso são fornecidas condições para realizar uma medida, independente da tarefa ou da qualidade da representação, cuja rede deve aprender.

Durante o aprendizado supervisionado, pares de entrada e saída são apresentados à rede. Esses pares consistem em um conjunto de entradas e um conjunto com as saídas desejadas para cada entrada. Quando uma entrada é apresentada à rede, uma saída será produzida e posteriormente comparada com a saída desejada. Se a saída produzida for diferente da saída desejada, um ajuste de pesos sinápticos deverá ser realizado de forma que a rede armazene o conhecimento desejado e, conseqüentemente, reflita o aprendizado das funções para as quais ela foi projetada. Portanto, o aprendizado consiste em um processo iterativo de atualização dos pesos sinápticos e, por meio desse processo, uma RNA é capaz de aprender e generalizar.

O ajuste dos pesos sinápticos é realizado de forma a minimizar a diferença entre a saída produzida e a saída desejada até que, para determinada entrada, a respectiva saída seja calculada pela rede. O cálculo abaixo visa somar ao peso atual o erro gerado pela rede e, dessa forma, corrigir o valor do peso. Existem inúmeros cálculos para este fim. No exemplo a seguir, será utilizada a Regra Delta, descrita abaixo:

$$w_i(n + 1) = w_i(n) + \Delta_i$$

$$\Delta_i = c \cdot g \cdot x_i$$

Onde:

- $w_i(n + 1)$ : Novo peso
- $w_i(n)$ : peso atual
- $\Delta_i$ : correção associada à entrada  $i$
- $c$ : taxa de aprendizado (normalmente é 1 quando a rede opera com valores binários)
- $g$  = saída desejada – saída obtida

De acordo com Ludwig e Costa (2007), uma das desvantagens das redes neurais consiste no fato delas serem, normalmente, uma “caixa preta”. Dessa forma, é impossível justificar como uma rede chegou a um determinado resultado ou, no caso de uma rede com múltiplas camadas, saber qual a importância de um peso sináptico para certo resultado. No entanto, é possível descobrir se a rede está funcionando corretamente pela análise do erro médio quadrático apresentado ao se introduzir os dados para validação.

#### 4.3.1 Algoritmo *Backpropagation*

Para que uma rede neural seja capaz de fazer o reconhecimento de padrões, inicialmente a estrutura da rede é treinada, ou seja, adquire conhecimento, para poder identificar os padrões almejados.

Um algoritmo de treinamento bastante conhecido na literatura através do qual as redes neurais podem adquirir conhecimento, é o chamado *backpropagation*.

Ele é constituído por duas etapas: a propagação e a retropropagação de um conjunto de sinais através da rede. A propagação consiste na aquisição dos dados pela camada de entrada e sua propagação por toda rede, produzindo uma saída. A saída da rede neural é comparada com a saída desejada e um valor de erro é calculado. A partir daí começa o processo de retropropagação, esse erro é propagado de volta à rede neural e usado para ajustar os pesos buscando reduzir o erro a cada iteração para que o resultado aproxime-se cada vez mais da saída desejada. A Figura 6 ilustra o esquema de funcionamento desse algoritmo.

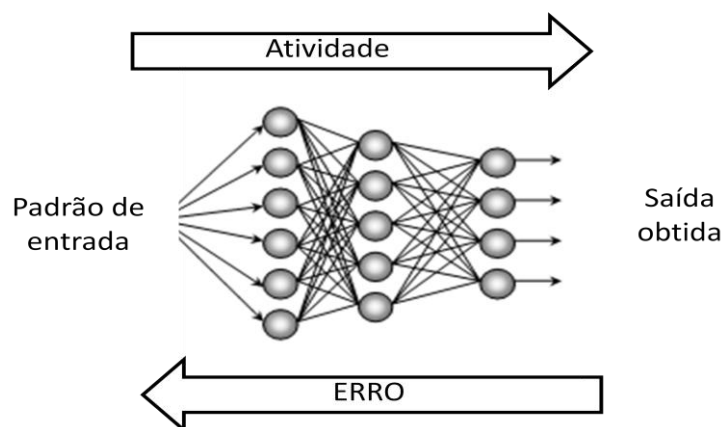


Figura 6. Esquema do funcionamento do Algoritmo Backpropagation.

As redes que utilizam *backpropagation* trabalham com a regra delta generalizada, uma variação da regra delta, apropriada para redes com múltiplas camadas.

A superfície de erro para um treinamento usando esse algoritmo pode não ser tão simples, como a ilustrada na Figura 7. Nestes casos, devem ser utilizadas redes com camadas intermediárias. Ainda assim, as redes ficam sujeitas aos problemas de procedimentos *hill-climbing*, ou seja, ao problema de mínimos locais. A Figura 7 também ilustra o processo de redução dos erros utilizando o *backpropagation*. O erro cometido pela rede vai sendo progressivamente diminuído, podendo chegar ao mínimo global. Esta figura ilustra um caso simples de uma rede com apenas um neurônio e duas conexões. Os valores dos pesos da rede definem a coordenada de um ponto da superfície de erro. O erro produzido pela rede para cada combinação de valores de pesos é dado pela altura da superfície naquele ponto. Assim, quanto mais alto for o ponto, maior o erro produzido pela rede.

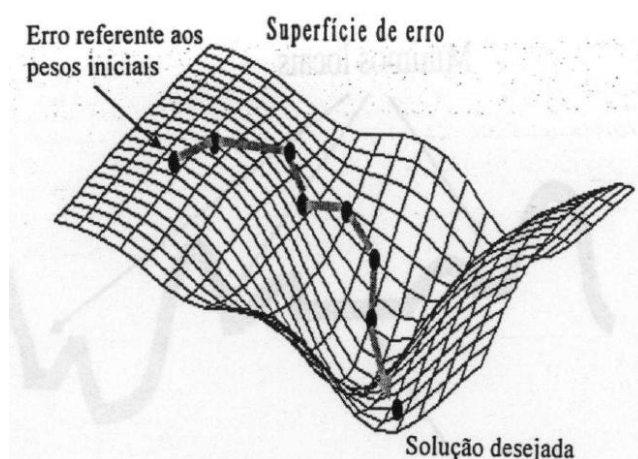


Figura 7. Superfície de erro para um treinamento usando backpropagation (Braga et al, 2000).

A regra delta generalizada funciona quando são utilizadas na rede unidades com uma função de ativação semi-linear, que é uma função diferenciável e não decrescente. Uma função de ativação amplamente utilizada, nestes casos, é a função sigmóide que apresenta um gráfico em forma de S e é definida pela equação:

$$f(x) = (1 + e^{-\lambda x})^{-1}.$$

### 4.3.2 Rede MLP (*Multilayer Perceptron*)

Segundo Braga et. al (2000), MLP é um tipo de rede neural artificial que apresenta pelo menos uma camada intermediária ou escondida. Para treinar esse tipo de rede utiliza-se o algoritmo *backpropagation*. A Figura 8 apresenta um exemplo de rede MLP.

O *Multilayer Perceptron* foi concebido para resolver problemas complexos, os quais não poderiam ser resolvidos pelo modelo de neurônio básico. Nesta classe de problemas, um único *perceptron* (tipo de RNA mais simples para classificação de padrões que possui apenas um neurônio) ou uma combinação das saídas de alguns perceptrons poderia realizar determinadas operações, porém, seria incapaz de aprendê-las. Para isto, são necessárias mais conexões, que só existem em uma rede de perceptrons dispostos em camadas. Os neurônios internos são de suma importância na rede neural, pois se provou que sem estes se torna impossível a resolução de problemas linearmente não separáveis, ou seja, aqueles que não podem ser satisfeitos utilizando uma reta como fronteira de decisão (veja Figura 9).

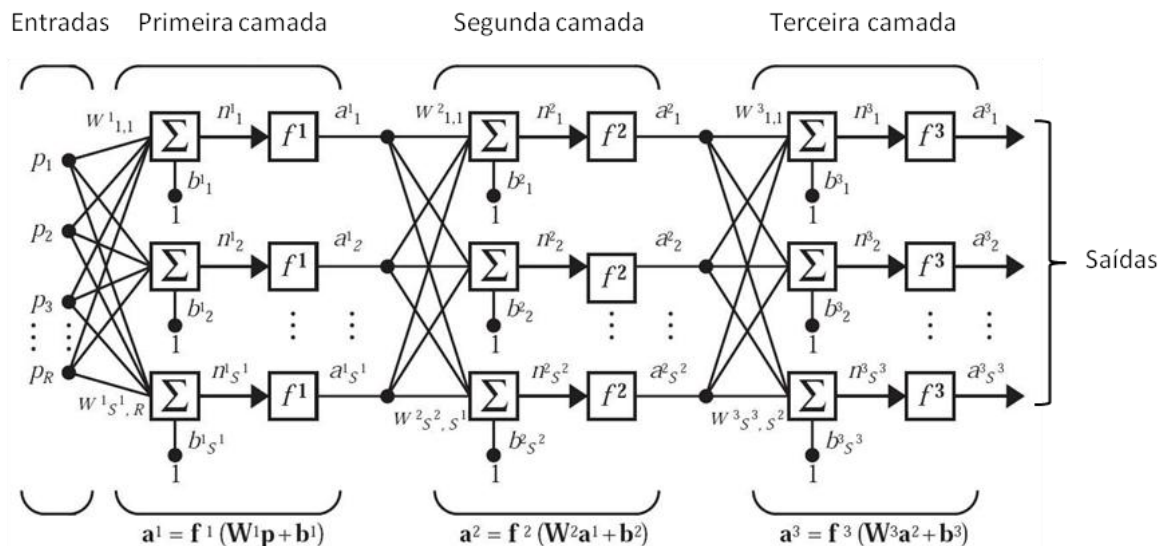


Figura 8. Exemplo de rede MLP.



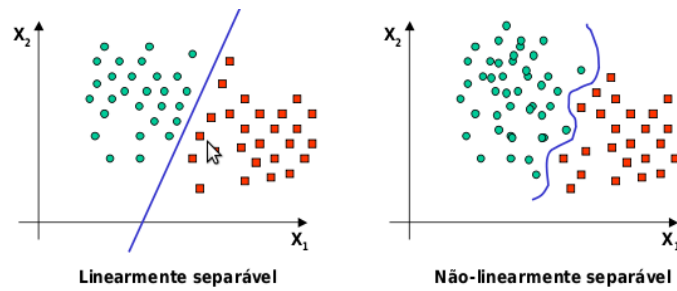


Figura 9. Problemas linearmente e não-linearmente separáveis.

#### 4.4 Reconhecimento de Padrões

A palavra reconhecer significa conhecer de novo, e isto implica num processo onde existe algum conhecimento prévio e algum tipo de armazenamento do conhecimento sobre o objeto a ser reconhecido. Esta é a parte onde os sistemas de visão possuem uma intersecção com a área de inteligência artificial. Portanto, para fazer o reconhecimento, um sistema de visão necessita de uma base de conhecimento dos objetos a serem reconhecidos. Esta base de conhecimento pode ser implementada diretamente no código ou pode ser aprendida a partir de um conjunto de amostras dos objetos a serem reconhecidos utilizando técnicas de aprendizado de máquina.

Os seres humanos são bons reconhecedores de padrões. Ao receberem dados à sua volta através dos sentidos, eles são capazes de reconhecer a fonte dos dados e, freqüentemente, isso acontece quase que imediatamente e praticamente sem esforço aparente. Pode-se, por exemplo, reconhecer um rosto familiar de uma pessoa muito embora esta pessoa tenha envelhecido desde o último encontro, identificar uma pessoa familiar pela sua voz ao telefone, apesar de uma conexão ruim. Enfim, os humanos são capazes de reconhecer um padrão através de um processo de aprendizagem; e assim acontece com as RNAs.

Um objeto pode ser definido por mais de um padrão (textura, forma, cor, dimensões, etc.) e o reconhecimento individual de cada um destes padrões pode facilitar o reconhecimento do objeto como um todo. Segundo Marengoni e Stringhini (2009), as técnicas de reconhecimento de padrões podem ser divididas em dois grandes grupos: estruturais e as de teoria de decisão. Nas estruturais, os padrões

são descritos de forma simbólica e a estrutura é a forma como estes padrões se relacionam. Nas técnicas, que utilizam teoria de decisão e englobam abordagens estatísticas e neurais, os padrões são descritos por propriedades quantitativas e deve-se decidir se o objeto possui ou não estas propriedades. É possível ainda utilizar ambas as técnicas, como por exemplo, no processo de reconhecimento de faces.

Não é uma tarefa fácil encontrar técnicas estruturais prontas em bibliotecas para linguagens de programação, uma vez que estas dependem da estrutura de cada objeto. Para alguns objetos específicos, porém, é possível encontrar pacotes prontos. As técnicas baseadas em teoria de decisão são mais gerais e podem ser adaptadas a diferentes tipos de objetos.

De acordo com Haykin (2001), o reconhecimento de padrões pode ser definido formalmente como o “processo pelo qual um padrão/sinal recebido é atribuído a uma classe dentre um número predeterminado de classes (categorias)”.

Para realizar o reconhecimento de padrões, uma rede passa inicialmente por uma seção de treinamento, como já foi mencionado. No caso do treinamento supervisionado, um conjunto de padrões de entrada junto com a categoria à qual cada padrão particular pertence são apresentados repetidamente à ela. Após o treinamento, apresenta-se a rede um padrão que não foi visto antes mas que pertence à mesma população de padrões utilizada para treinar a rede. A rede torna-se capaz de identificar a classe daquele padrão particular por causa da informação que ela extraiu dos dados de treinamento.

À proporção de padrões corretamente classificados do conjunto total de padrões de teste é dado o nome taxa de acerto. A proporção de padrões incorretamente classificados dá origem à taxa de erro.

## **4.5 Rastreamento**

Após efetuar o reconhecimento de um padrão em uma sequência de quadros de um vídeo utilizando rede neural, é possível também acompanhar a sua

trajetória nessa seqüência. Dessa forma, o rastreamento consiste num processo de reconhecer um determinado padrão em uma seqüência de imagens ou quadros.

De acordo com Marengoni e Stringhini (2009), as técnicas de rastreamento podem ser aplicadas nas mais diversas áreas, desde sistemas de segurança (vigilância) até o uso em sistemas de interface homem-máquina. Além disso, no openCV, tais técnicas incluem dois componentes principais: identificação de objetos e modelagem da trajetória.

É possível encontrar métodos para se estimar a posição de um objeto quadro a quadro, indo de filtros Kalman, até processos com filtros de partículas. Algoritmos estimadores são aqueles capazes de estimar, por exemplo, a localização de uma pessoa ou objeto em movimento numa seqüência de vídeo. Segundo Marengoni e Stringhini (2009), esta tarefa é dividida, basicamente, em duas fases: a) predição – baseada num conhecimento prévio de dados da imagem e b) correção – que usa novas medidas para apurar a predição realizada anteriormente.

A técnica mais conhecida para isto é o filtro de Kalman. Conforme WELCH et al. (2006), ele resolve o problema de estimar o estado  $x \in \mathbb{R}^m$  de um processo discreto que é governado por uma função não-determinística linear (Equação 4) e pela medição de  $z \in \mathbb{R}^m$  (Equação 5).

$$x_t = A \cdot x_{t-1} + B \cdot u_t + w_t \quad (4)$$

$$z_t = H \cdot x_t + v_t \quad (5)$$

O vetor  $u$  armazena entradas para controle externo sobre o sistema,  $B$  é uma matriz que relaciona estas entradas para a mudança de estado. Pelo fato de observar apenas o sistema a ser previsto,  $u$  é sempre igual a zero e a Equação 4 possui apenas o primeiro e o último termo diferentes de zero. O valor de  $A$  representa a relação entre o estado anterior, medido, e o estado previsto. Na equação 5,  $H$  é uma matriz  $m \times n$  representando a relação entre o estado previsto ( $x_t$ ) e o valor medido ( $z_t$ ). As variáveis aleatórias  $w_t$  e  $v_t$  são chamadas, respectivamente, de ruído de processo e ruído de medição.

Conforme WELCH et al. (2006), o filtro utiliza um método de controle retroalimentado, ou seja, ele estima o processo em algum momento e obtém a

retroalimentação medindo o sistema ruidoso. Assim pode-se dizer que ele funciona em um processo de duas etapas: uma de previsão e outra de correção.

Na etapa de previsão, o próximo estado do filtro é previsto a partir do estado atual. Essa previsão é conhecida como estado *a priori*. Depois de prever o próximo estado, ele realiza a correção. Nessa etapa, o valor medido é incorporado ao sistema para melhorar o estado *a priori*, criando um estado aprimorado *a posteriori*. A diferença entre o previsto e o medido é utilizada para aperfeiçoar as variáveis do filtro, diminuindo, assim, o erro de predição.

Além desta técnica, existem também algumas funções no OpenCV que são utilizadas para rastreamento, baseadas em cálculos estatísticos. Essas funções estão presentes nos algoritmos de rastreamento *Mean Shift* e *Cam Shift*.

*Cam Shift* (*Continuously Adaptive Mean-SHIFT*) é um algoritmo desenvolvido para o rastreamento de cor, possibilitando também o rastreamento de faces. É baseado numa técnica estatística onde se busca o pico entre distribuições de probabilidade. Ele utiliza como base o algoritmo *Mean Shift* (média por deslocamento). O *Mean Shift* foi adaptado no *Cam Shift* para tratar a mudança dinâmica das distribuições de probabilidade das cores numa sequência de vídeo.

#### 4.6 Biblioteca OpenCV

É possível encontrar ferramentas e bibliotecas para auxiliar na execução das tarefas de reconhecimento e estimação de trajetória, discutidas anteriormente.

Desenvolvida inicialmente pela Intel Corporation, OpenCV (Open Source Computer Vision) é uma biblioteca de programação, de código aberto, que implementa uma variedade de ferramentas de interpretação de imagens. Ela foi idealizada com o objetivo de tornar a visão computacional acessível a usuários e programadores em áreas tais como a interação humano-computador em tempo real e a robótica.

O OpenCV possui módulos de Processamento de Imagens e Vídeo I/O, Estrutura de dados, Álgebra Linear, GUI Básica com sistema de janelas independentes, Controle de mouse e teclado, além de mais de 350 algoritmos de

Visão Computacional como filtros de imagem, calibração de câmera, reconhecimento de objetos e análise estrutural.

Segundo Bradski e Kaehler (2008), a biblioteca OpenCV está escrita em C e C++ e pode ser executada nos sistemas operacionais Linux, Windows e Mac OS X, podendo tirar proveito de processadores com múltiplos núcleos. O Anexo A apresenta os procedimentos para ajuste do ambiente OpenCV para os sistemas operacionais Linux e Windows. Existe um desenvolvimento ativo em interfaces para dar suporte a programadores das linguagens Python, Ruby, Matlab, e outras linguagens.

O pacote OpenCV está disponível gratuitamente na Internet por meio do endereço eletrônico: <http://sourceforge.net/projects/opencvlibrary>. No pacote da versão 2.0, estão disponíveis o código fonte da biblioteca, os executáveis (binários) otimizados para os processadores Intel, a documentação e exemplos em C/C++ e python.

As principais bibliotecas do OpenCV são:

- “cv.h” : possui as principais funcionalidades de processamento de imagem e algoritmos de Visão Computacional.
- “cvaux.h”: possui algoritmos de Visão que ainda estão em fase experimental.
- “cxcore.h”: possui estruturas básicas e algoritmos, suporte a XML, funções de desenho.
- “highgui.h”: possui funções ligadas a interface de usuário, imagem e entrada e saída de vídeos.
- “ml.h”: possui métodos de *Machine Learning* (aprendizagem de máquina), *clustering*, classificação e análise de dados.

Um programa OpenCV, ao ser executado, invoca automaticamente uma DLL (Dynamic Linked Library) que detecta o tipo de processador e carrega, por sua vez, a DLL otimizada para este. Juntamente com o pacote OpenCV, é oferecida a

biblioteca IPL (Image Processing Library), da qual a OpenCV depende parcialmente, além de documentação e um conjunto de códigos exemplos.

A biblioteca está dividida em cinco grupos de funções: Processamento de imagens; Análise estrutural; Análise de movimento e rastreamento de objetos; Reconhecimento de padrões e Calibração de câmera e reconstrução 3D.

Portanto, o OpenCV foi de fundamental importância para o desenvolvimento deste projeto pois ofereceu suporte para o tratamento de Vídeos, para RNAs do tipo MLP e, ainda, forneceu um algoritmo para rastreamento conhecido como *Cam Shift*.

## **5 Materiais e métodos**

Os materiais utilizados no desenvolvimento deste projeto foram: câmera digital, arquivos de vídeo, livros, artigos, computador, simuladores e ferramentas de programação. Estes materiais encontram-se disponíveis na internet e nos laboratórios de hardware e de informática da UNIVASF.

Os materiais para estudos foram basicamente: livros, apostilas, tutoriais, artigos, trabalhos de conclusão de curso e simuladores. Estes materiais foram obtidos através do orientador, na biblioteca da própria universidade e também obtidos pela internet.

Os conhecimentos requeridos para o desenvolvimento do projeto foram referentes a linguagens de programação tais como C/C++ (Costa, 2007), processamento de imagens, redes neurais e estimadores. Conhecimentos estes que foram adquiridos através das disciplinas oferecidas pela UNIVASF, através do orientador e também através de livros, artigos e minicursos.

### **5.1 Delineamento Metodológico**

O projeto foi desenvolvido durante 12 meses. As atividades desenvolvidas estão relacionadas abaixo e seguiram o cronograma da Tabela 1. Para o caso específico deste projeto, foram seguidos os seguintes passos:

1. Estudo de estruturas de geração de vídeos digitais, redes neurais artificiais e aplicações de reconhecimento de padrões em vídeos, programação em C/C++;
2. Download, instalação e estudo da biblioteca OpenCV;
3. Elaboração de exemplos e testes com a biblioteca OpenCV. Nesta etapa foram desenvolvidos algoritmos em C/C++ utilizando a biblioteca OpenCV voltados para o projeto tais como rotinas para ler um arquivo de vídeo, capturar um quadro, processar o conteúdo de um quadro, entre outras. Essas rotinas foram usadas posteriormente para implementação dos

algoritmos de processamento de imagem, reconhecimento de padrões e acompanhamento de trajetória;

4. Análise e definição dos procedimentos utilizados para estruturação dos algoritmos. Nessa etapa foi feito um planejamento de como os algoritmos seriam implementados e como os módulos estariam ligados entre si.
5. Desenvolvimento em linguagem C/C++ utilizando a biblioteca OpenCV dos algoritmos para capturar e aplicar operações de processamento de imagem para um determinado quadro ou seqüências de quadros de um vídeo, efetuar reconhecimento de um padrão em uma seqüência de quadros utilizando rede neural, e para acompanhar a trajetória de um determinado padrão em uma seqüência de quadros de um vídeo;
6. Realização de testes e validação;
7. Documentação das atividades desenvolvidas;
8. Efetuar correções sugeridas pela banca examinadora;

Tabela 1. Cronograma de Execução do Projeto

	Atividade							
Mês	1	2	3	4	5	6	7	8
Agosto/2010	X						X	
Setembro/2010	X	X					X	
Outubro/2010			X	X			X	
Novembro/2010			X	X			X	
Dezembro/2010					X		X	
Janeiro/2011					X		X	
Fevereiro/2011					X	X	X	
Março/2011					X	X	X	
Abril/2011					X	X	X	
Maio/2011					X	X	X	
Junho/2011					X	X	X	
Julho/2011						X	X	X

As atividades acima listadas foram executadas durante dois períodos ou duas disciplinas: TCCI e TCII. O período do TCCI consistiu na fase inicial para o aprofundamento teórico do tema a ser trabalhado e preparação para posterior



codificação dos algoritmos no TCCII. A defesa do TCCI foi fundamental para o desenvolvimento do sistema, uma vez que nesta fase foi definido o escopo do trabalho e as formas como os algoritmos seriam implementados.

As atividades de 1 a 4 consistiram em uma preparação para o desenvolvimento dos algoritmos mencionados na atividade 5. Na atividade 1, foi efetuado um estudo e levantamento bibliográfico dos conhecimentos necessários para desenvolvimento do projeto. Para tanto foi realizado estudo de livros, tutoriais, apostilas, artigos referentes aos temas: visão computacional, redes neurais, OpenCV, reconhecimento de padrões e linguagem de programação C/C++.

Em seguida, na atividade 2, houve uma preparação do ambiente operacional, por meio de download e instalação das ferramentas necessárias para a implementação dos algoritmos. Para execução das etapas posteriores foi necessário o download, instalação e configuração das seguintes ferramentas de software, tanto no computador do laboratório 10 da UNIVASF como no computador pessoal, necessárias para execução do projeto: DEV-C++, biblioteca OpenCV e FormatFactory 2.50 (programa para conversão de vídeos).

Na etapa posterior, foi realizado o estudo e execução de exemplos de programas utilizando a biblioteca OpenCV que acompanham a ferramenta e estão disponíveis para execução após sua devida instalação e configuração. Além disso, foram desenvolvidos algoritmos em C/C++ utilizando a biblioteca OpenCV tais como rotinas para ler um arquivo de vídeo, capturar um quadro, processar o conteúdo de um quadro, treinar uma rede, localizar um padrão em um quadro, entre outras. Essas rotinas foram usadas posteriormente para implementação dos algoritmos de processamento de imagem, reconhecimento de padrões e acompanhamento de trajetória.

O passo 4 consistiu então em planejar como os exemplos desenvolvidos até o momento da fase de implementação do sistema seriam utilizados, como os algoritmos estariam interligados, como seria realizado o treinamento, reconhecimento e acompanhamento dos padrões em arquivos de vídeo. Durante esse planejamento, decidiu-se que o sistema seria desenvolvido na plataforma operacional Windows (também compatível com o Sistema Operacional Linux desde

que devidamente configurado), utilizando-se a ferramenta de programação DEV-C++, sendo dividido em partes para facilitar a implementação e correção de eventuais erros. Essa ferramenta de programação pode ser baixada gratuitamente no site oficial: <http://www.bloodshed.net/>. A rede neural escolhida foi a do tipo *Multilayer Perceptron* (MLP) por se adequar melhor à resolução do problema. Para efetuar o treinamento da rede, foi utilizado o algoritmo *backpropagation*.

A linguagem escolhida para implementação do sistema foi C++. Dentre as principais características que motivaram tal escolha estão: código multiplataforma, paradigma de orientação à objeto, poder de processamento e manter um padrão com o projeto já realizado.

Na etapa 5, desenvolveu-se o sistema propriamente dito. O primeiro passo consistiu na aquisição de arquivos de vídeo e posterior captura dos quadros dos mesmos. Foi então desenvolvido um algoritmo para processamento dos quadros capturados. Em seguida, foi esquematizado o algoritmo da RNA do tipo MLP para posterior treinamento com arquivos de vídeo. Utilizou-se a RNA fornecida pelo próprio OpenCV.

Para tanto, foram utilizados 4 arquivos de vídeo no formato AVI (40 x 30), para cada tipo de padrão, com duração entre 10 e 14 segundos, 12 quadros/s e tamanho entre 14 e 25 KB, para treinamento e validação da rede neural. Gravou-se também arquivos no mesmo formato, 8 arquivos de vídeo para cada tipo de padrão, com duração entre 25 e 36 segundos, 12 quadros/s e tamanho entre 32 e 66 KB, para testar o funcionamento do algoritmo de reconhecimento e acompanhamento de padrões. A Figura 10 ilustra uma seqüência de quadros obtidos de um vídeo com o padrão círculo.

Após realizar o treinamento da RNA para o reconhecimento de um determinado padrão, foi possível esquematizar o algoritmo para detecção de trajetória, utilizando uma função para rastreamento conhecida como *CamShift* fornecida pela biblioteca OpenCV.

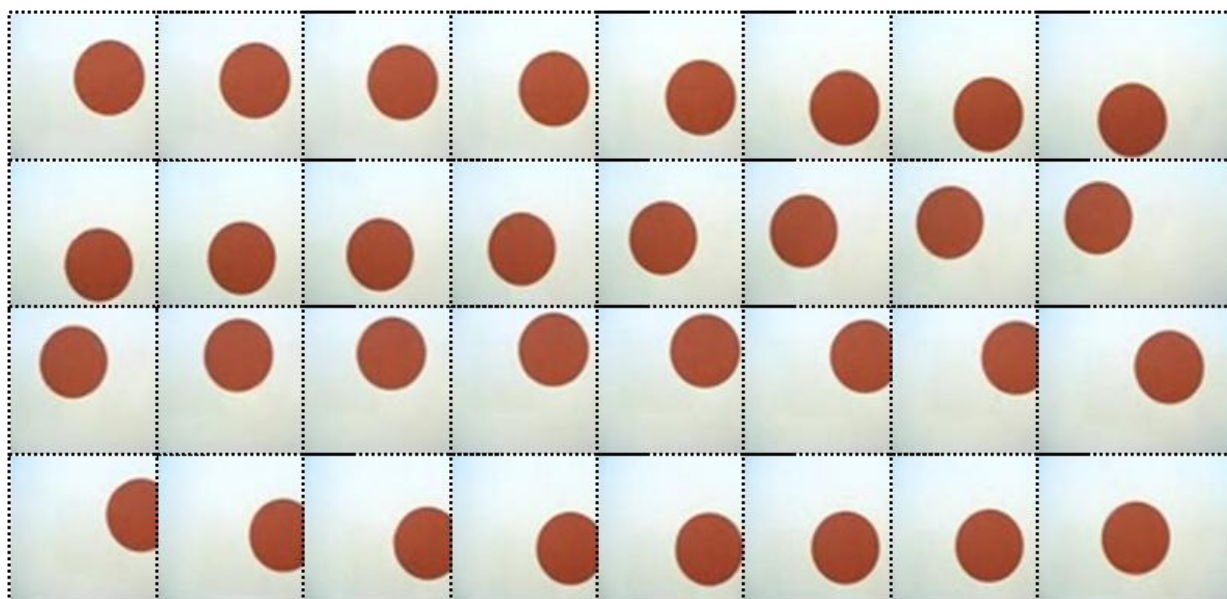


Figura 10. Seqüência de quadros de um vídeo mostrando a movimentação de um círculo.

A gravação de vídeos foi realizada utilizando uma câmera digital Kodak Easyshare C143 utilizando-se uma resolução de 640 x 480. Para diminuir a resolução dos vídeos para 40 x 30 utilizou-se o software gratuito de conversão conhecido como FormatFactory 2.50.

Após a implementação de cada algoritmo, realizava-se a atividade 6, ou seja, eram efetuados testes para verificar o funcionamento e validar o que foi desenvolvido. Da mesma forma, ao longo de todas as etapas de desenvolvimento do sistema foi executada a atividade 7, isto é, todos os procedimentos realizados foram documentados para a produção de relatórios bimestrais.

Foi reservado também um tempo, conforme mostra a atividade 8, para realizar eventuais correções sugeridas pela banca examinadora deste Trabalho de Conclusão de Curso.

## 6 Resultados obtidos

O primeiro desafio a ser vencido consistiu em aprender a lidar com arquivos de vídeo, desde a sua captura até a reprodução, separação e processamento de quadros. Pelo fato de lidar muito bem com imagens, a biblioteca OpenCV oferece uma gama de funções que também se aplicam ao processamento de quadros de um vídeo.

Para a captura dos quadros de um vídeo no formato AVI e a sua respectiva execução, é necessário conhecer algumas das funcionalidades que a ferramenta OpenCV proporciona. Antes do início da captura de quadros do arquivo de vídeo, é necessário informar para o algoritmo o caminho completo de localização do arquivo no computador. Além disso, é preciso declarar uma variável do tipo *IplImage* para armazenar temporariamente os quadros individuais do arquivo de vídeo, e uma variável do tipo *CvCapture* que consiste numa estrutura para carregar o vídeo.

Para a execução do vídeo, é necessária a criação de uma janela. Na Figura 11, é apresentado um fluxograma com as etapas de captura de quadros num arquivo de vídeo.

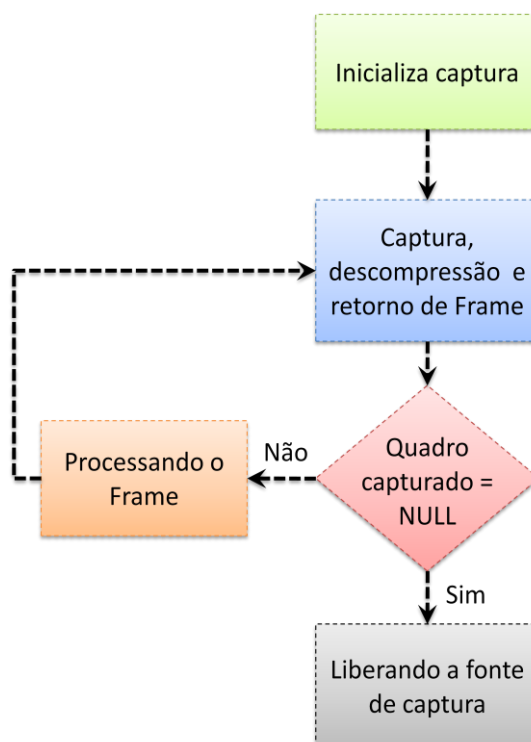


Figura 11. Fluxograma com as etapas de captura de quadros de vídeo

Na primeira etapa do fluxograma anterior, inicializa-se a captura por meio da chamada da função `cvCreateFileCapture(char*)` que retorna uma estrutura para captura de vídeo denominada `CvCapture` e recebe como parâmetro o caminho completo de localização do arquivo no computador. Caso ocorra alguma falha durante a execução da função `cvCreateFileCapture(char*)`, a captura não poderá continuar. Uma falha pode acontecer caso o caminho do arquivo de vídeo seja informado incorretamente ou caso o arquivo não exista. Não ocorrendo nenhum erro, o programa continua a execução normalmente. O programa entra em um laço de repetição que só é finalizado quando não é possível mais capturar um próximo quadro do vídeo, liberando-se então a fonte de captura. Em cada execução do laço de repetição ocorre a captura de um novo quadro por meio da função `cvQueryFrame(CvCapture*)` e a sua exibição em uma janela por meio da função `cvShowImage()`. A função `cvQueryFrame(CvCapture*)` retorna um quadro capturado da variável `Cvcapture`. É possível ainda a execução de algum tipo de processamento sobre o quadro capturado. Portanto, é possível várias execuções das etapas 2 (Captura, descompressão e retorno de Frame) e 3 (Processando o Frame) do fluxograma da Figura 11.

Além de permitir a aplicação de operações de processamento às imagens contidas em arquivos de vídeo no formato AVI, a biblioteca OpenCV também oferece a possibilidade de capturar um vídeo diretamente de uma câmera filmadora, conforme ilustra a Figura 12.

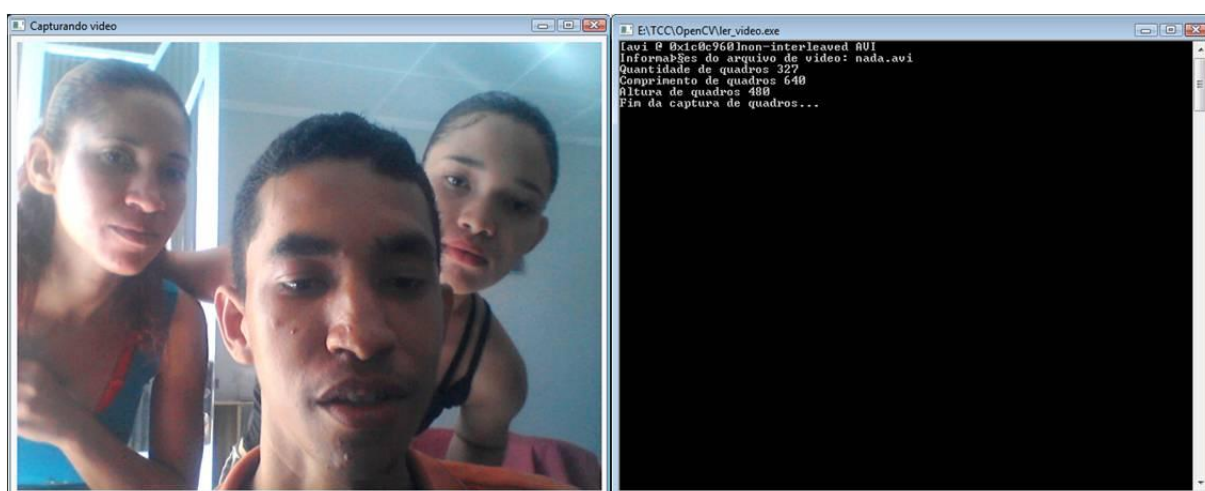


Figura 12. Tela de saída de vídeo capturado de uma câmera

Os procedimentos para captura de vídeo a partir de uma câmera seguem as mesmas etapas do fluxograma da Figura 11, o que muda é que a fonte de captura agora é uma câmera, ao invés de um arquivo de vídeo. Para definir uma câmera como fonte de captura utiliza-se a função *cvCaptureFromCAM(int)*. O parâmetro desta função é um número inteiro (aceita valores de 0 a 10) que se refere ao índice da câmera que será utilizada para a captura de imagem, ou seja, é possível capturar imagem de várias câmeras simultaneamente, chamando a função com os índices respectivos. No caso deste projeto, utilizou-se apenas uma câmera, portanto utilizou-se o valor do parâmetro como sendo 0 (zero). Caso aconteça alguma falha e a captura não possa ser inicializada, devido a seleção da câmera errada ou caso ela esteja desligada, uma mensagem de erro é então exibida na tela e o programa é então finalizado.

Depois do passo anterior e após ter inicializado uma janela, para visualizar o resultado, o programa entra em um laço de repetição que só é finalizado quando não é possível mais capturar um próximo quadro do vídeo.

## 6.1 Etapa de processamento das imagens

Após a captura de cada quadro de um determinado vídeo foi aplicado um conjunto de transformações em cada um desses quadros com o objetivo de facilitar o trabalho da RNA durante a etapa de treinamento e reconhecimento.

Podem ser aplicados diversos tipos de operações de processamento sobre uma determinada imagem. No caso deste trabalho, foram aplicadas operações de transformação em tons de cinza, filtro *gaussiano*, binarização e detecção de bordas por meio do filtro de *sobel*. Esta última foi bastante importante para o desenvolvimento do algoritmo, pois gera como resultado apenas o contorno que define a forma principal do objeto permitindo a economia de recursos computacionais uma vez que não será necessário analisar todo o contexto da imagem.

A transformação em tons de cinza foi realizada para cada quadro original capturado do vídeo de entrada, utilizando-se a função *cvCvtColor(quadro, cinza, CV\_BGR2GRAY)* que consiste basicamente em tornar os valores das coordenadas

RGB da imagem de origem todos com o mesmo valor de acordo com um limiar estabelecido. Essa função recebe como parâmetros uma imagem de entrada, outra de saída e um código inteiro que identifica o tipo de conversão de cor que será executado, e possui retorno do tipo *void*. No caso do OpenCV, a transformação de cada pixel ocorre da seguinte forma:  $Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$ , onde Y é saída para um determinado pixel da imagem.

Após a transformação em tons de cinza, os quadros resultantes passam por um filtro *gaussiano* para a correção e remoção de ruídos. O filtro *gaussiano* é basicamente uma operação de convolução, utilizada para “borrar” uma imagem digital com o objetivo de remover detalhes e ruídos. Ele utiliza dois parâmetros: a dimensão da janela e o desvio padrão máximo. O desvio padrão está relacionado com o quanto a imagem será suavizada (quanto maior mais a imagem é suavizada), a janela define a influência que os pixels vizinhos terão sobre um determinado ponto. Para a sua execução utilizou-se a função *cvSmooth(cinza, filtro, CV\_GAUSSIAN, 3, 3, 0, 0)*, onde se define, respectivamente, a imagem de entrada, a imagem de saída, coordenadas da janela e desvio padrão. Utilizou-se uma janela 3 X 3 e um desvio padrão igual a 0 pelo fato de não “borrar” muito a imagem.

Uma distribuição Gaussiana com média zero e desvio padrão  $\sigma$  em duas dimensões é descrita em termos de direções perpendiculares  $X$  e  $Y$  pela equação abaixo, onde  $X$  e  $Y$  representam as posições dos pixels na janela e  $G(x, y)$  retorna o valor a ser colocado na posição  $(x, y)$  da janela.

$$G(x, y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Depois dos quadros passarem pelo filtro *gaussiano*, aplicou-se o algoritmo de *Canny*. Este último foi utilizado para efetuar a detecção de contornos e a conseqüente binarização dos quadros. Numa etapa inicial, a imagem de entrada é ligeiramente desfocada para reduzir o efeito de pixels ruidosos. Em seguida, acontece o cálculo do gradiente da intensidade de cada pixel na imagem, retornando assim a direção da maior variação de claro para escuro e a quantidade de variação nessa direção.

No algoritmo de *Canny*, as primeiras derivadas são calculadas em x e y e então combinadas em quatro derivadas direcionais. Para o cálculo dessas derivadas utiliza-se o operador *Sobel* definido pela função *cvSobel(source, dest, xorder, yorder, mask)* que retorna um tipo *void* e recebe como parâmetro a imagem de entrada, a imagem de saída, ordem da derivada x, ordem da derivada y e tamanho da janela. Os pontos onde estas derivadas direcionais são máximos locais são pixels candidatos para a montagem de contornos. Esses contornos são formados através da aplicação de um limiar de histerese para os pixels. Isto significa que existem dois limiares, um superior e um inferior. Se um pixel tem um gradiente maior do que o limite superior, então é aceito como um pixel de borda, se um pixel está abaixo do limite inferior, é rejeitado. Se o gradiente do pixel está entre os limiares, será aceito somente se estiver conectado a um pixel que está acima do limite de altura.

Para a execução do algoritmo de *Canny*, utilizou-se a função *cvCanny(filtro, filtro\_sobel, sobel\_limiar, sobel\_limiar\*8, 3)*, onde se define, respectivamente, a imagem de entrada em tons de cinza, a imagem de saída binarizada, limiar 1, limiar 2 e parâmetro de abertura para operador *Sobel*.

A Figura 13 ilustra na prática alguns dos procedimentos descritos anteriormente, aplicados na etapa de processamento dos quadros de cada um dos vídeos de entrada. As telas da primeira coluna são os quadros originais, nos quadros da segunda foi aplicado transformação em tons de cinza, nos da terceira foi aplicado filtro Gaussiano e nos da última foi aplicado filtro *Sobel*.

Após a aplicação do algoritmo de *Canny*, os valores dos pixels de cada quadro do vídeo são armazenados em arquivos no formato txt, para posterior treinamento da RNA. Esses arquivos são nomeados de acordo com a ordem de numeração do quadro no vídeo, a Figura 14 mostra um exemplo de saída gerada para o quadro 0 de um arquivo de vídeo. Neste arquivo, o valor da linha 1 e coluna 1 indica o tamanho do vetor de pixels da imagem, resultante da multiplicação do seu comprimento pela sua largura (o que corresponde ao número de linhas do arquivo), o valor da linha 1 e coluna 2 é a identificação do tipo de padrão presente no quadro (o uso desse valor será entendido posteriormente). Os valores de identificação do tipo de padrão foram estabelecidos conforme mostra a Tabela 2.



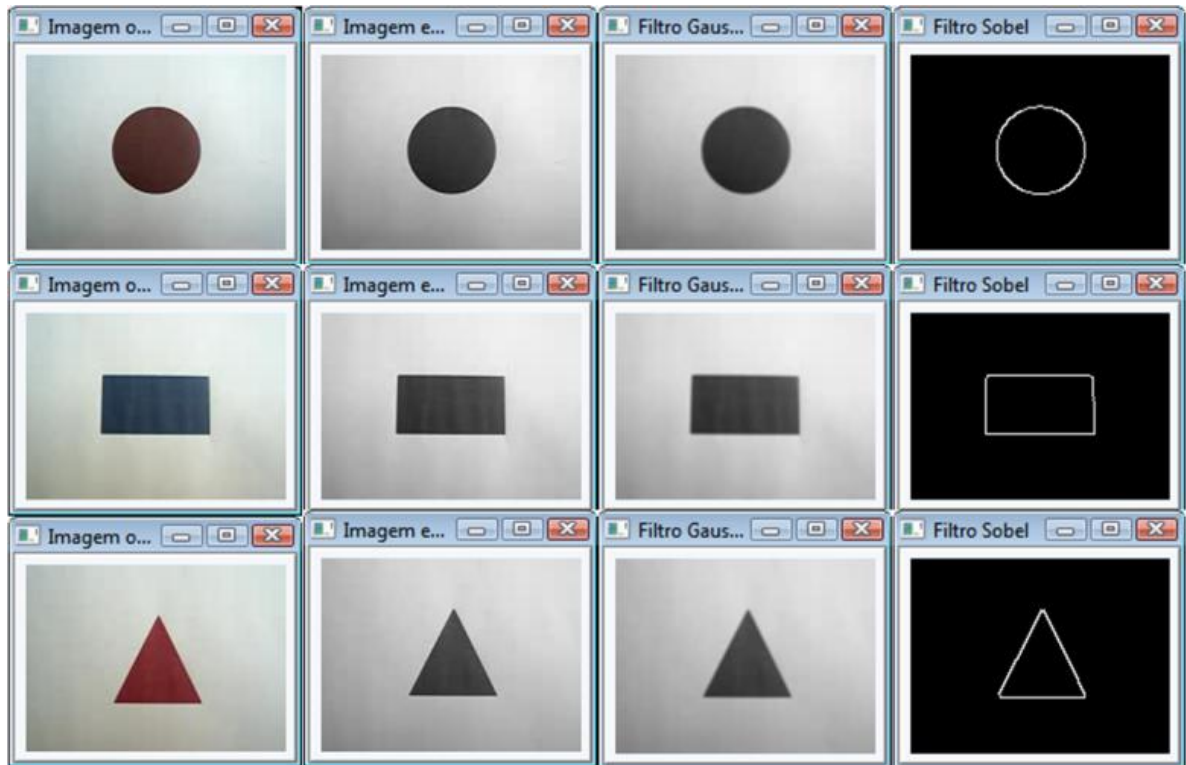


Figura 13. Imagens resultantes do algoritmo de processamento de imagens

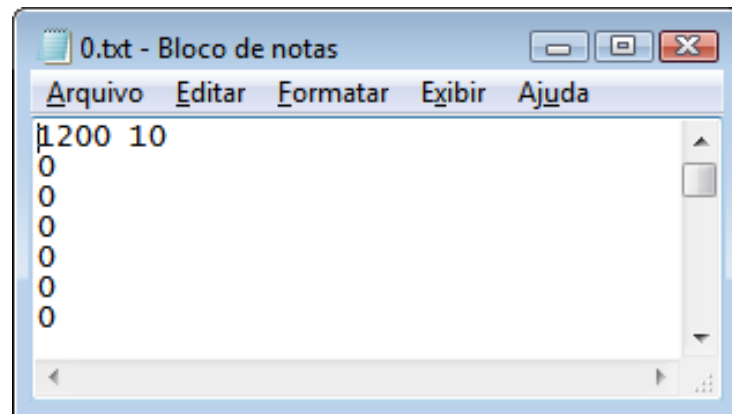


Figura 14. Arquivo com saídas do processamento do quadro 0

Tabela 2. Códigos de identificação das classes de imagem

Padrão	Saída desejada
Círculo	10
Retângulo	20
Triângulo	30

## 6.2 Etapa de treinamento

Após devidamente processados, os quadros estão prontos para o treinamento de uma RNA para posterior reconhecimento. Nessa etapa, utilizou-se a RNA fornecida pelo próprio OpenCV. No entanto, antes de chamar a função do OpenCV para treinamento da rede, foi necessário configurar os parâmetros da rede e preparar os dados para serem submetidos.

Criou-se um arquivo da classe *CvANN\_MLP* denominado de *mlp* que consiste em uma estrutura para a criação de uma RNA do tipo MLP. Antes da criação da RNA, foi definida a sua arquitetura, que consistiu de 4 camadas (uma camada de entrada com 1200 neurônios, duas camadas ocultas com 20 neurônios e uma camada de saída com 1 neurônio). A quantidade de neurônios na camada de entrada corresponde ao tamanho da imagem de entrada. A quantidade de camadas ocultas com seus respectivos neurônios não é fixa e pode ser alterada a qualquer momento no código para refinamento dos resultados. A camada de saída, por meio de seu neurônio, corresponde à classificação que será dada pela rede.

Essas configurações foram então salvas em uma matriz do tipo *CvMat* denominada de *camadas\_rede*. Os dados resultantes da etapa de processamento de imagens (entradas e saída, salvas em arquivo do tipo ilustrado na Figura 14) foram então carregados, respectivamente, para as matrizes denominadas no programa como *dados* e *respostas*. Foi criada ainda uma matriz chamada *saída\_rede* para armazenar as respostas produzidas pela RNA. Dessa forma, foi criada uma RNA para cada vídeo de entrada. O código completo da função de treinamento desenvolvida em linguagem C/C++ encontra-se no Anexo B.

Após o carregamento dos dados, a RNA definida pela variável *mlp* é então treinada por meio da chamada da função *mlp.train()*, que está detalhada no trecho de código da Figura 15. Dentre os parâmetros que ela recebe estão as matrizes de entrada, critérios de finalização (definidos como sendo o número máximo de iterações ou a precisão), número máximo de iterações (10000), precisão (0.000001), algoritmo de treinamento (BACKPROP - *backpropagation*), taxa de aprendizagem (0.01) e momento (0.01). A precisão consiste na diferença entre a saída obtida e a saída esperada. O momento é um valor constante entre 0 e 1 que influencia na

velocidade de alteração dos pesos, acelerando-a ou desacelerando-a. A taxa de aprendizagem também é um valor entre 0 e 1 que influencia na mudança dos pesos de cada neurônio e, conseqüentemente, no tempo de treinamento da rede, isto é, quando próximo de 0 esse tempo é mais longo, contudo, quando próximo de 1 ele é mais curto. A escolha dos valores dos parâmetros foi realizada de forma experimental, isto é, com base em testes de variados valores e análises das mudanças nas respostas.

```

118     mlp.train(
119         dados, //vetor de entradas
120         respostas, //vetor de saidas
121         saida_rede, //vetor de pesos
122         0, //vetor opcional mostrando colunas de entrada e de saida
123         CvANN_MLP_TrainParams( cvTermCriteria( //criterio de finalizacao
124             CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, /*CV_TERMCRIT_ITER*/
125             10000, //numero maximo de iteracoes
126             0.000001 // precisao necessaria
127         ),
128         CvANN_MLP_TrainParams::BACKPROP, /*CvANN_MLP_TrainParams::RPROP*/
129         0.01,
130         0.01
131     )
132 );

```

Figura 15. Trecho de código mostrando função de treinamento

A execução da função é encerrada ao atingir um dos critérios de finalização. Após o treinamento, os dados resultantes são salvos em um arquivo “.rna” por meio da chamada *mlp.save(nome\_file)*. Esse arquivo contém os pesos ajustados pelo algoritmo *backpropagation* na etapa de treinamento e outras informações relativas à rede tais como: número de camadas e de neurônios, parâmetros de treino, valores mínimos e máximos, critérios de aprendizagem, taxa de aprendizagem e momento, método de treinamento, entre outras. A Figura 16 ilustra um exemplo desse tipo de arquivo.

Para verificar o desempenho da rede chama-se a função *mlp.predict(file\_dados, file\_saida\_rede)* para que ela utilize os pesos obtidos para calcular as saídas a partir dos próprios dados de entrada utilizados no treinamento. A saída gerada pela rede é então comparada com a saída que se espera. Caso o erro gerado seja maior que o erro estabelecido, a rede então é novamente treinada por meio da função *mlp.train()* e os resultados são novamente salvos e atualizados.



um determinado quadro do vídeo de entrada e gerar um arquivo txt com os dados resultantes da operação de processamento, o arquivo com os dados de treinamento de uma RNA escolhida pelo usuário é então carregado para que possa estimar a saída para aquele quadro. Se a saída estimada estiver com uma determinada precisão estabelecida, então o padrão foi reconhecido.

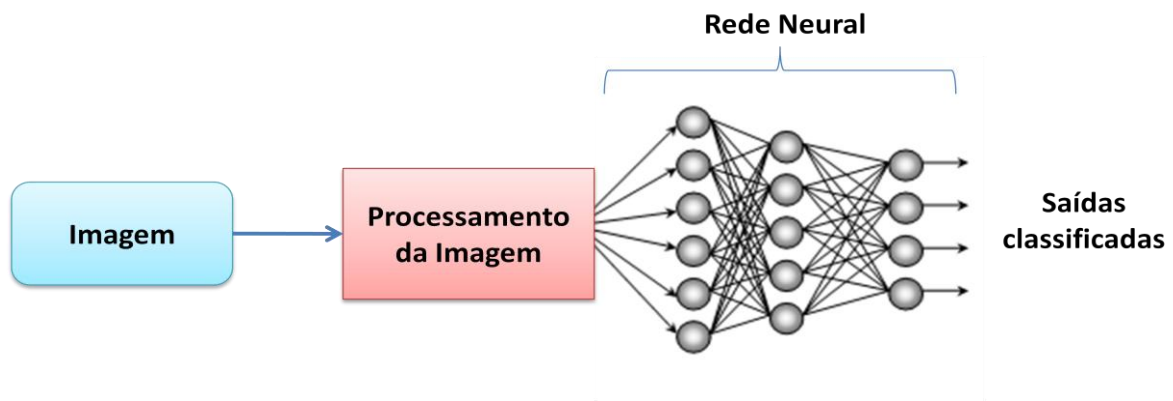


Figura 17. Esquema de blocos para o de reconhecimento de padrões

#### 6.4 Etapa de rastreamento

Após o reconhecimento de um determinado padrão, é possível rastreá-lo numa seqüência de quadros de um vídeo, caso ele ainda se faça presente. Neste trabalho foi utilizado o algoritmo de rastreamento conhecido como *Cam Shift* (*Continuously Adaptive Mean-SHIFT*). Ele foi desenvolvido para o rastreamento de cor. Baseia-se em uma técnica estatística onde se busca o pico entre distribuições de probabilidade.

Para o seu funcionamento, é necessário fornecer a área onde se encontra o padrão que será rastreado. Portanto, neste trabalho, o rastreamento acontece de duas formas. Na primeira forma, após o padrão ter sido reconhecido pela RNA, utiliza-se uma função denominada *definir\_area()*, implementada durante a realização deste trabalho, para encontrar as coordenadas do retângulo que melhor circunscreve o padrão detectado. A função utiliza técnicas de detecção de contornos e retorna o retângulo que melhor se adéqua a um determinado contorno. O retângulo encontrado é então fornecido como informação inicial para o algoritmo

*Cam Shift* que então será responsável por continuar rastreando o padrão nos próximos quadros. A segunda forma de utilização consiste em o usuário selecionar com o *mouse* o retângulo que melhor circunscreve o padrão a ser rastreado.

O *Cam Shift* é utilizado na função *cvCamShift( const CvArr\* prob image, CvRect window, CvTermCriteria criteria, CvConnectedComp\* comp, CvBox2D\* box=NULL)* da biblioteca OpenCV para o acompanhamento de objetos em uma seqüência de quadros. Essa função retorna um inteiro que indica o número de operações feitas internamente pelo *Mean Shift*. Ela recebe, respectivamente, como parâmetros: projeção de fundo do objeto histograma, janela de pesquisa inicial, critério para finalizar a pesquisa, estrutura resultante que contém as coordenadas da janela de pesquisa convergente e caixa circunscrita para o objeto. A base desse algoritmo é outro algoritmo conhecido como *Mean Shift*. A Figura 18 mostra o funcionamento do algoritmo *Cam Shift*, onde a parte cinza corresponde à contribuição do algoritmo *Mean Shift*.

Os passos executados pelo *Cam Shift* podem ser descritos resumidamente da seguinte forma:

1. Determina a região para cálculo da distribuição de probabilidades para toda a imagem;
2. Escolhe a posição inicial para a janela de busca em duas dimensões (2D) necessária para o algoritmo *Mean Shift*;
3. Calcula a distribuição de probabilidades de cor na região centrada na localização da janela de busca, um pouco maior que o tamanho da janela de busca do *Mean Shift*;
4. Executa o algoritmo *Mean Shift* para achar o centro da janela de busca.
5. Para o próximo quadro do vídeo, centraliza a janela de busca na localização média encontrada no passo 4. Retorna ao passo 3.

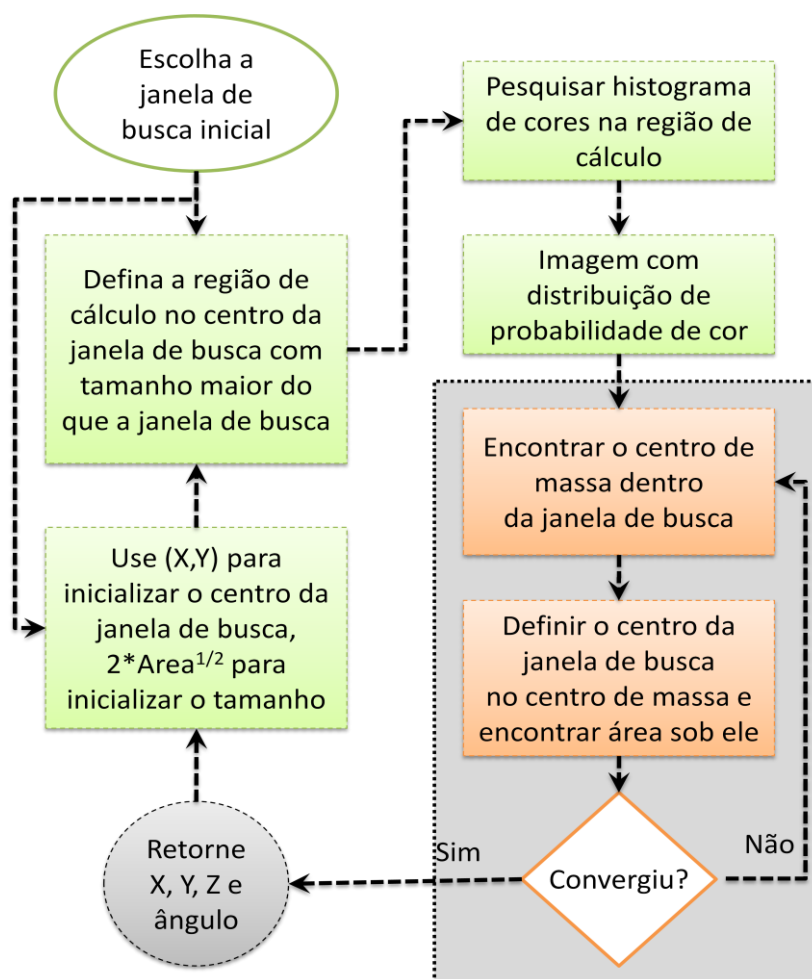


Figura 18. Algoritmo Cam Shift (INTEL Corporation,2009).

## 6.5 Como utilizar os algoritmos

Os algoritmos são executados por meio de uma interface simples que possibilita a escolha dos procedimentos desejados pelo usuário. A Figura 19 ilustra o menu principal do sistema.

Conforme ilustra a Figura 19, o sistema possibilita treinar uma RNA para um determinado tipo de padrão, reconhecer um padrão a partir de um vídeo e acompanhar o deslocamento de um padrão ao longo de um vídeo.

Ao escolher a opção 1 do menu principal, visualiza-se uma tela seguinte conforme mostra a Figura 20. É possível treinar uma RNA para reconhecer os seguintes tipos de padrões: círculo, retângulo ou triângulo.

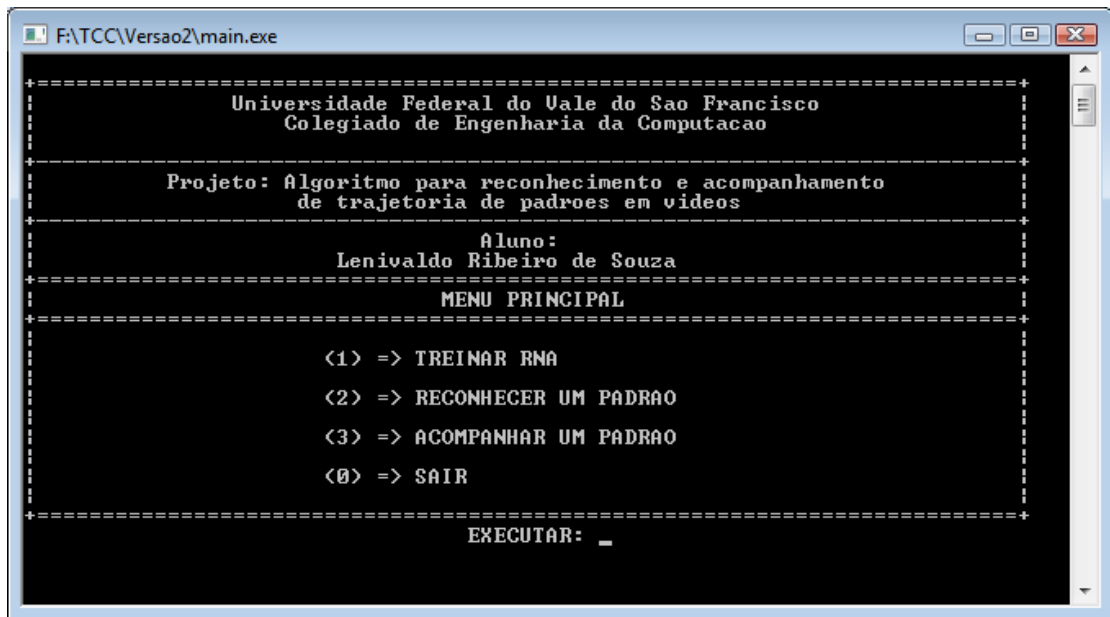


Figura 19. Menu principal



Figura 20. Menu treinar RNA

Após escolher um das opções da Figura 20, aparece a informação de que o caminho completo dos vídeos para serem usados durante o treinamento da RNA deve ser informado em um arquivo no formato txt, conforme ilustram as Figuras 21 e 22. Após essa informação, basta o usuário pressionar Enter para que o processo de



treinamento seja iniciado. Dessa forma, uma nova janela será aberta com a execução de cada um dos vídeos informados no arquivo txt.



Figura 21. Menu treinar circulo

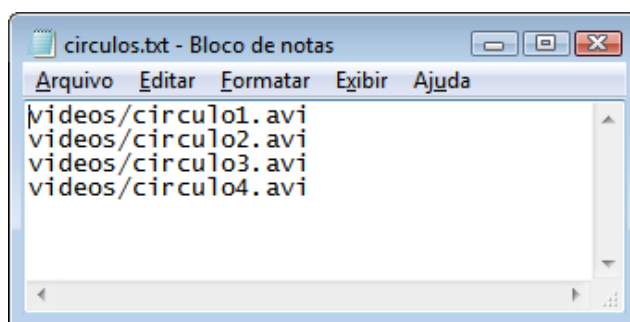


Figura 22. Exemplo de arquivo txt com entradas para treinamento

Ao escolher a opção 2 do menu principal, uma tela com as opções de reconhecimento de padrões será aberta, conforme mostra a Figura 23.



Figura 23. Sub-menu reconhecer um padrão

É possível reconhecer um padrão a partir de um arquivo de vídeo armazenado no computador ou dispositivo de memória externa ou ainda por meio de uma câmera conectada e configurada no computador. Caso seja escolhida a opção 1 da Figura 23, será solicitado que o usuário informe o tipo de padrão a ser reconhecido para que o respectivo arquivo com os dados do treinamento produzido pela RNA, conforme mostra a Figura 24. Em seguida deverá ser escolhido um dentre oito vídeos com o padrão a ser reconhecido, conforme ilustra a Figura 25. Mas caso a opção 2 seja a escolhida, após escolher o tipo de padrão a ser reconhecido da mesma forma que ilustra a Figura 24, basta que o usuário deixe a câmera ligada antes de continuar (Figura 26). Dessa forma, uma nova janela será aberta com a execução do vídeo informado pelo usuário para o seu reconhecimento pela RNA e posterior rastreamento.



Figura 24. Reconhecer padrão por vídeo

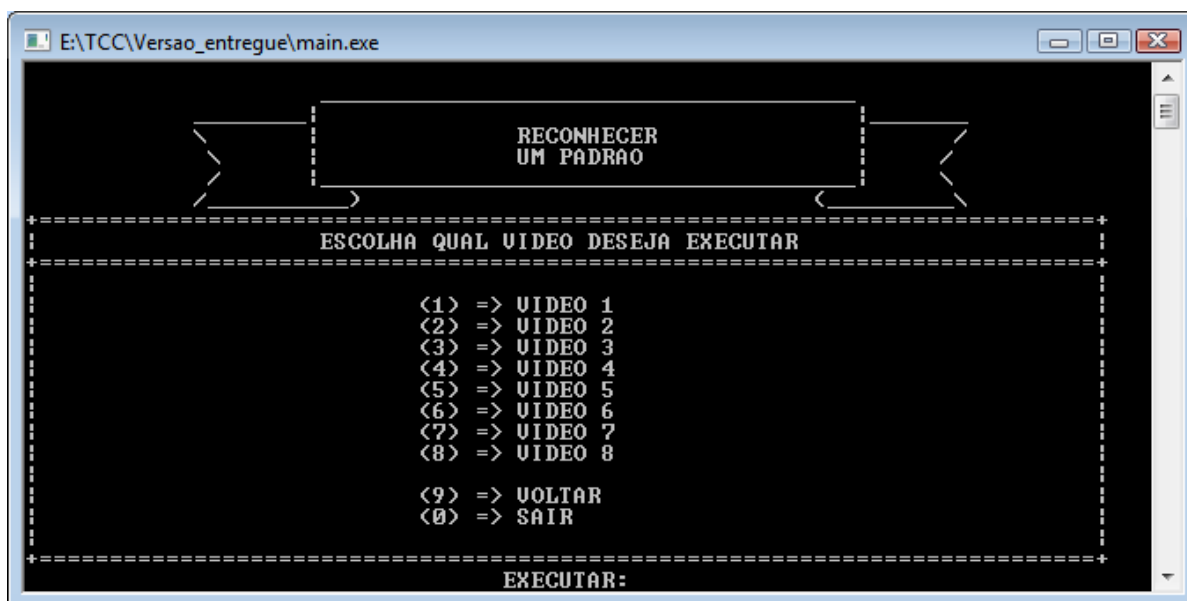


Figura 25. Tela para escolha de vídeos

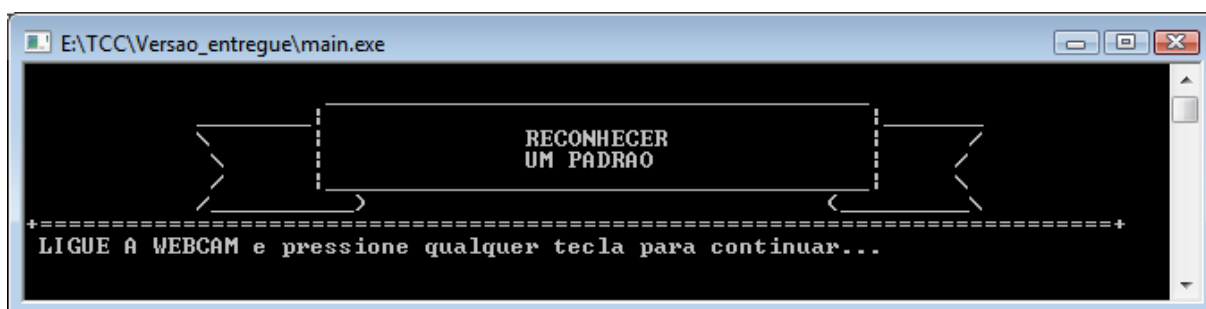


Figura 26. Reconhecer padrão por WebCam

Por fim, o sistema também possibilita o rastreamento ou acompanhamento de um padrão em um arquivo de vídeo, conforme mostra a Figura 27. Caso seja escolhida a opção 1 da Figura 27, será solicitado que o usuário informe o caminho do arquivo de vídeo com o padrão a ser reconhecido, conforme ilustra a Figura 28. Mas caso a opção 2 seja a escolhida, basta o usuário ligar a câmera antes de continuar (Figura 29). Após isso o programa abrirá uma nova janela contendo a execução do vídeo, onde o usuário deverá selecionar com o *mouse* a área de interesse que contém o padrão que deseja ser rastreado. O algoritmo então irá desenhar uma elipse verde em torno do padrão selecionado ao longo de cada quadro do vídeo (Figura 30). No caso do usuário escolher a opção 3 do menu

principal (Figura 19), o rastreamento do padrão acontece de forma independente do reconhecimento pela rede neural, tanto é que o usuário precisará selecionar com o mouse o padrão que deseja acompanhar, dessa forma utilizou-se vídeos para teste com padrões mistos, conforme ilustra a Figura 30. Além disso, é possível ainda rastrear uma face ou qualquer outro padrão de um vídeo gerado de uma câmera conectada ao computador.



Figura 27. Menu acompanhar um padrão



Figura 28. Acompanhar um padrão por arquivo de vídeo

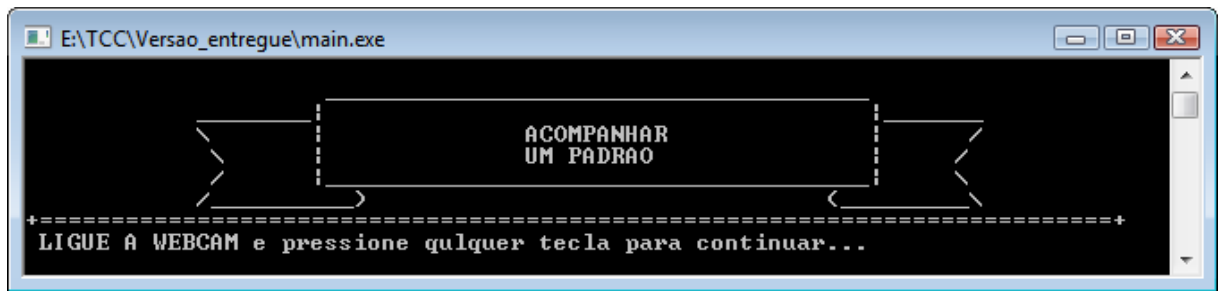


Figura 29. Acompanhar um padrão por WebCam

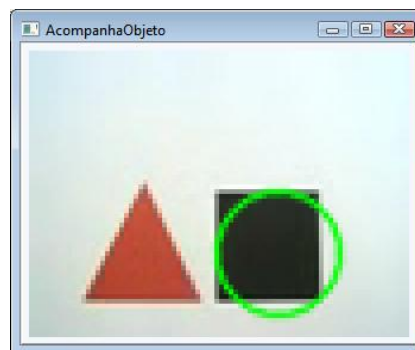


Figura 30. Exemplo de tela de acompanhamento de padrão

Nas partes do programa onde usuário é solicitado a fornecer o caminho no computador para um arquivo de vídeo de entrada, caso o caminho esteja incorreto ou o arquivo informado não exista o programa retorna um erro, informando que o arquivo não pôde ser aberto.

## 6.6 Discussão dos resultados

Pretendeu-se, ao longo do desenvolvimento do projeto, obter um algoritmo para capturar e aplicar operações de processamento de imagem sobre um determinado quadro ou seqüências de quadros de um vídeo, efetuar reconhecimento de um padrão em uma seqüência de quadros utilizando rede neural e acompanhar a trajetória do padrão nessa seqüência.

A Figura 31 ilustra, em alto nível e de forma resumida, as etapas do sistema desenvolvido. A Figura 32 mostra o resultado da aplicação de processamento de

imagem nos quadros de um vídeo. A Tabela 3 mostra os resultados da etapa de reconhecimento. A Figura 33 mostra os resultados da etapa de rastreamento.

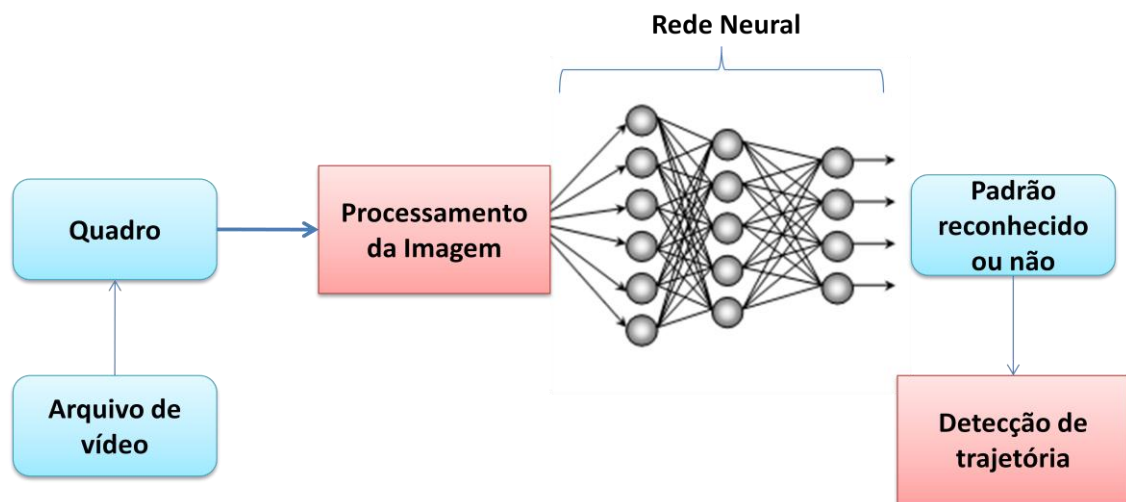


Figura 31. Etapas resumidas do algoritmo

A parte de processamento de imagem demonstrou-se bastante eficaz, para todos os vídeos utilizados, sendo o contorno de cada padrão detectado de forma precisa, conforme pode ser verificado nas imagens da Figura 32.

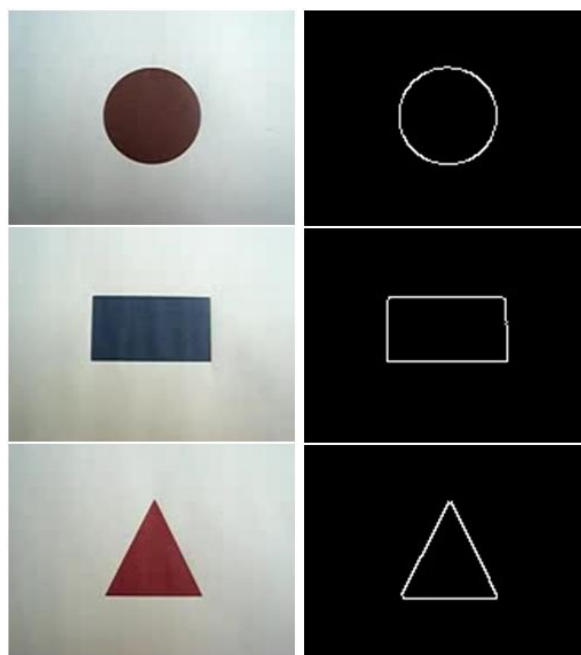


Figura 32. Resultados do processamento de quadros

Tabela 3. Resultados obtidos na etapa de reconhecimento

Padrão	Vídeos testados	Vídeos classificados corretamente	Erro máximo cometido
<b>Círculo</b>	8	8	12%
<b>Retângulo</b>	8	8	8%
<b>Triângulo</b>	8	8	1,1%

Na Tabela 3, o erro máximo cometido corresponde a percentagem da diferença entre o valor absoluto da diferença entre a saída máxima gerada pela rede e a saída desejada, ou seja:

$$erro_{maximo} = |saida_{rede} - saida_{desejada}| * 100 / saida_{desejada}$$

Quando o erro cometido pela rede for menor que o erro máximo admitido para cada padrão, definido como 15%, então se conclui que o vídeo foi classificado corretamente.

Na apresentação das imagens à rede, foi verificada a necessidade de imagens de boa qualidade, pois muitos fatores podem influenciar no resultado, como por exemplo, a iluminação, o fundo, ruídos, entre outros. Embora a tarefa do sistema de visão humano de analisar e reconhecer informações visuais pareça trivial, esta pode ser muito complexa quando executada por sistemas de visão artificial, tanto mais quanto se permite um conjunto pequeno de restrições sobre o sistema de aquisição de imagens, por exemplo, variações bruscas de iluminação na imagem adquirida.

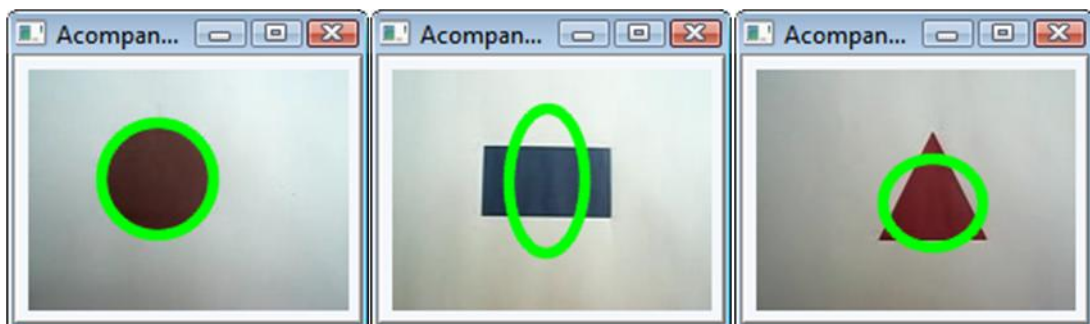


Figura 33. Resultados para etapa de rastreamento.

A parte de reconhecimento pode ser melhorada por meio do aumento do nível de precisão do reconhecimento sem diminuir muito o tamanho dos quadros do vídeo, pois na medida em que se diminui o tamanho do vídeo a qualidade da imagem fica comprometida e dificulta o nível de precisão da rede.

Apesar de o algoritmo ser desenvolvido para padrões de baixa complexidade (círculo, retângulo e triângulo), os resultados poderão ser generalizados futuramente para padrões de alta complexidade.

Durante a realização do projeto, foi possível obter um algoritmo para capturar e aplicar operações de processamento de imagem para um determinado quadro ou seqüências de quadros de um vídeo, reconhecer os padrões para os quais a RNA foi treinada, mas apenas em vídeos contendo somente o padrão, e acompanhar a trajetória do padrão ao longo de uma seqüência de vídeos. Embora o reconhecimento tenha sido de forma limitada, todas as etapas planejadas inicialmente neste trabalho foram executadas.



## 7 Conclusão

Por meio do estudo teórico e do levantamento bibliográfico realizado, foi possível perceber a viabilidade da utilização das redes neurais artificiais na tarefa de reconhecimento de padrões em imagens. Além disso, algoritmos para reconhecimento e acompanhamento de trajetória de padrões em vídeos são de interesse tanto teórico quanto prático.

A importância desse projeto se deve ao fato de propor um sistema de visão que pode ser melhorado e expandido para utilização em aplicações reais. Além disso, os resultados deste projeto poderão ser compartilhados com a comunidade científico-acadêmica por meio da divulgação em congressos, bem como à sociedade em geral através de palestras. No contexto técnico, os resultados deste projeto poderão servir de base para o desenvolvimento de futuros projetos relacionados com a mesma linha.

A biblioteca OpenCV ofereceu uma grande ajuda no desenvolvimento do projeto, ao oferecer módulos e soluções prontas e em desenvolvimento, além de possuir uma boa documentação e um livro como excelente referência, no entanto, algumas limitações dificultaram a realização do projeto tais como: máximo de 1000 ciclos para treinamento da rede e limitação do tamanho de dados de entrada para treinamento. Essas limitações tiveram um peso na precisão do reconhecimento pela RNA. Dentre outras dificuldades encontradas pode-se citar também: dificuldade para encontrar uma estratégia eficaz para submeter os dados dos quadros do vídeo para o treinamento da RNA, dominar os comandos e funções da biblioteca OpenCV pois são muitos e alguns possuem elevada complexidade.

Portanto, como sugestão de melhorias a serem efetuadas em trabalhos futuros pode-se citar: redimensionamento dos quadros de um vídeo utilizando técnicas de redução, adaptação para reconhecimento de outros tipos de padrões e melhoria na interface com o usuário.

Sem dúvidas, a realização deste trabalho contribuiu para aquisição por parte do discente de uma gama de conhecimentos a respeito da área definida, não somente conhecimentos teóricos como também práticos.

## 8 Referências

- ARTUR, J. S. **Reconhecimento de Padrões Usando Indexação Recursiva**. Universidade Federais de Santa Catarina, 1999.
- BARRETO, Jorge M. **Introdução às Redes Neurais Artificiais**. In: V Escola Regional de Informática. Sociedade Brasileira de Computação. Florianópolis, 1997.
- BRADSKI, G; KAEHLER, Adrian. **Learning OpenCV**. O'Reilly, 2008.
- BRAGA, Antônio de Pádua; LUDERMIR, Teresa Bernarda; CARVALHO, André C. P. de Leon Ferreira. **Redes Neurais Artificiais: teoria e aplicações**. Editora LTC: Rio de Janeiro, 2000.
- CARLOS, J. F. P. **Aplicação de Redes Neurais no Processamento Digital de Imagens**. Universidade Federal de Minas Gerais, 1994.
- CASTLEMAN, Kenneth R. **Digital Image Processing**. Upper Saddle River: Prentice Hall, Inc. 1996.
- CORDEIRO, F. M. **Reconhecimento e Classificação de Padrões de Imagens de Núcleos Linfócitos do Sangue Periférico Humano com a Utilização de Redes Neurais Artificiais**. Universidade Federal de Santa Catarina, 2002.
- COSTA, Eduard M. M. **Programando com C Simples e Prático**. Editora Alta Books: Rio de Janeiro, 2006.
- GONZALEZ, R. C. WOODS, R. E. **Processamento de Imagens Digitais**. EDITORA EDGARD BLÜCHER LTDA: São Paulo, 2000.
- GUIMARÃES, Luiz Felipe Sá Leitão. **Deteção da Bola em Vídeos de Futebol**. Universidade Federal de Pernambuco, Centro de Informática.
- HAYKIN, S. **Redes Neurais: Princípios e prática**. McMaster University, Hamilton, Ontário, Canadá, 2001.

- HENRIQUE, J. B. R. **Desenvolvimento de uma técnica de reconhecimento de padrões baseada em distância**. Universidade Federal de Santa Catarina, 2003.
- INTEL Corporation. **OpenCV documentation, reference manual**. 2009.
- LUDWIG, Oswaldo Jr., e COSTA, Eduard M. M. **Redes Neurais: Fundamentos e Aplicações com Programas em C**. Editora Ciência Moderna: Rio de Janeiro, 2007.
- LUGER, G. F. **Inteligência Artificial: estruturas e estratégias para a resolução de problemas complexos**. University of New Mexico at Albuquerque, 2004.
- MARENGONI, Maurício; STRINGHINI, Denise. **Tutorial: Introdução à Visão Computacional usando OpenCV**. Universidade Presbiteriana Mackenzie, 2009.
- MARIA, L. G. F. **Processamento Digital de Imagens**. INPE, Junho de 2000.
- MASCARENHAS, N. D. A. **Breve Introdução ao Reconhecimento Estatístico de Padrões**. 39ª Reunião Anual da SBPC. 1987.
- MICROSOFT CORPORATION. **DV Data in the AVI File Format**. 1997. Disponível em: <http://download.microsoft.com/download/1/6/1/161ba512-40e2-4cc9-843a-923143f3456c/DVAVSPEC.RTF>. Acesso em: 18 de Janeiro de 2011.
- PERELMUTER, G.; CARRERA, E. V.; VELLASCO, M.; PACHECO, A. **Reconhecimento de Imagens Bidimensionais Utilizando Redes Neurais Artificiais**. Anais do VII SIBGRAPI, 1995.
- PRADO, A. Jr.; ELFES, A. **Um Projeto em Reconhecimento de Padrões de Forma**. Monografia de Graduação, ITA. São José dos Campos-SP, 1975.
- SCHWARTZ, William Robson ET AL. **Reconhecimento em tempo real de agentes autônomos em futebol de robôs**. Universidade Federal do Paraná: Curitiba, 2003.

- SILVA, Anderson F.; LEAL, Brauliro L.; COSTA, Luiz Cláudio. **OpenNet: Software Livre para Estudo e Prática de Redes Neurais**. Universidade Federal de Viçosa, 2002.
- STEINER, M. T. A. **Uma Metodologia para o Reconhecimento de Padrões Multivariados com Resposta Dicotômica**. Tese de Doutorado. Florianópolis-SC, 1995.
- TODESCO, J. L. **Reconhecimento de Padrões usando Rede Neuronal Artificial com uma Função de Base Radial: uma aplicação na classificação de cromossomos humanos**. Tese de Doutorado. Florianópolis-SC, 1995.
- TOU, J. T.; Gonzalez, R. C. **Pattern Recognition Principles**. Addison-Wesley Publishing Company: Massachusetts, 1981.
- VII WORKSHOP DE VISÃO COMPUTACIONAL. **Anais do Evento**. Universidade Federal do Paraná. Curitiba: 2011. Disponível em: [http://www.wvc2011.ufpr.br/anais\\_wvc\\_2011.pdf](http://www.wvc2011.ufpr.br/anais_wvc_2011.pdf). Acesso em: 08 de Julho de 2011.
- WELCH, Greg; BISHOP, Gary. **An Introduction to the kalman filter**. Disponível em: <<http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html>>. Acesso em: 17 de Novembro de 2010.

## ANEXOS

### ANEXO A. Ajuste do ambiente do OpenCV

- **Download e instalação no Ambiente Linux**

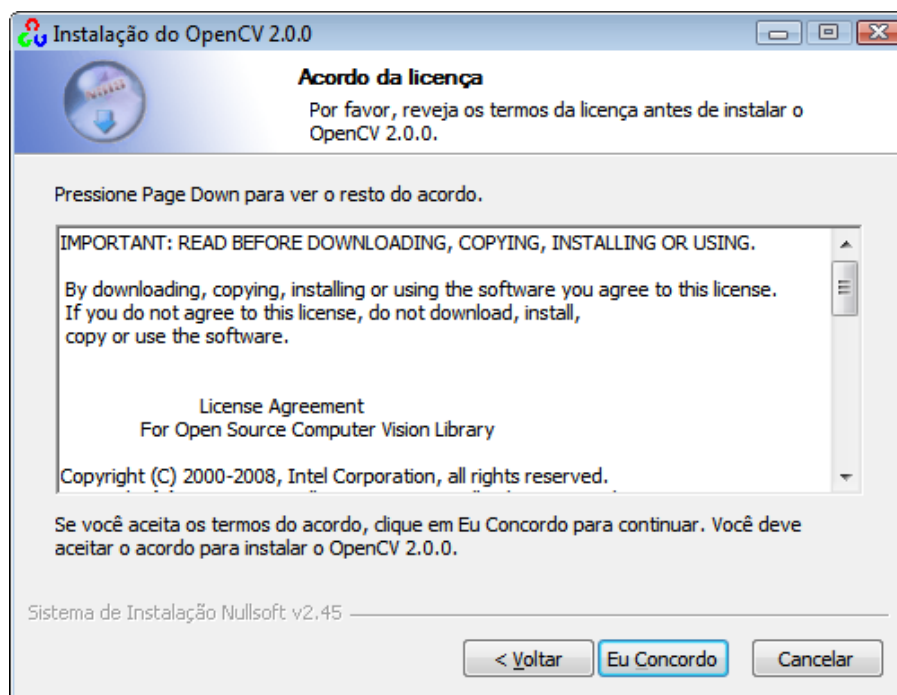
1. Efetue o download do pacote OpenCV formato *tar.gz* por meio do endereço eletrônico: <http://sourceforge.net/projects/opencvlibrary>;
2. Entrar no terminal do Linux e efetuar *login* de super-usuário por meio do comando “sudo su”.
3. Para descompactar o arquivo, deve-se redirecionar o terminal por meio do comando “cd” para o diretório onde foi feito o download do arquivo *tar.gz* do OpenCV e, na janela aberta, digitar os seguintes comandos (supondo que o pacote que você tenha baixado seja: *opencv-0.9.7.tar.gz*):
  - `tar -xzf opencv-0.9.7.tar.gz`
  - `cd opencv-0.9.7`
  - `./configure && make && make install`
4. Fazer os ajustes para o compilador do linux G++:
  - Ir ao diretório “home” e abrir o arquivo *.bashrc* (para o ambiente bash).
  - Acrescentar a seguinte linha ao arquivo:
    - `alias gcv="g++ -I/usr/local/include/opencv -lcxcore -lcvaux -lhighgui"`

- **Download e instalação no Ambiente Windows**

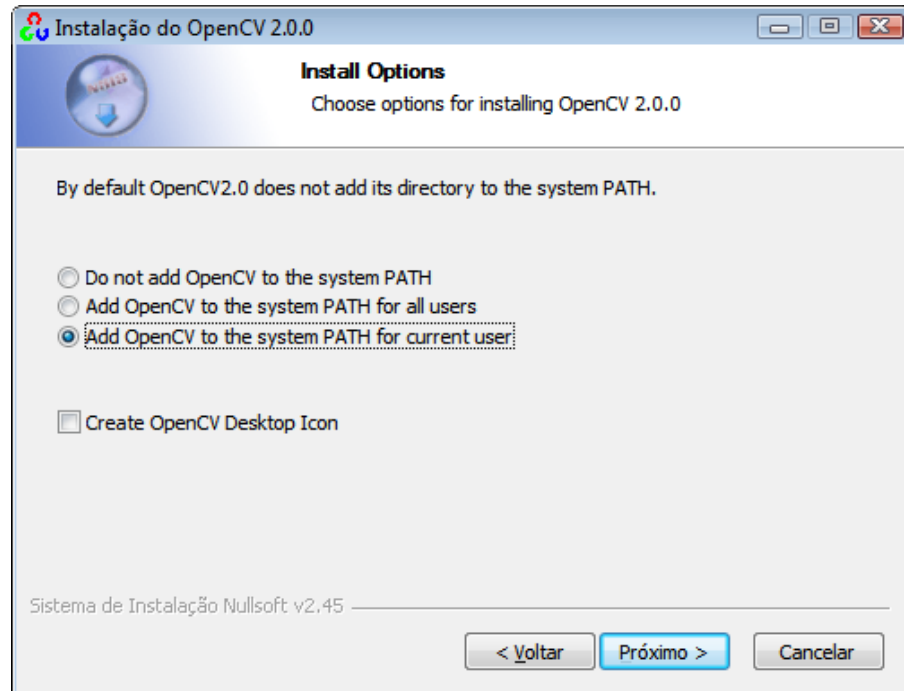
1. Efetue o download do pacote OpenCV 2.0 por meio do endereço eletrônico: <http://sourceforge.net/projects/opencvlibrary>;
2. Após descompactar clique no arquivo executável e abrirá uma tela semelhante a tela abaixo;



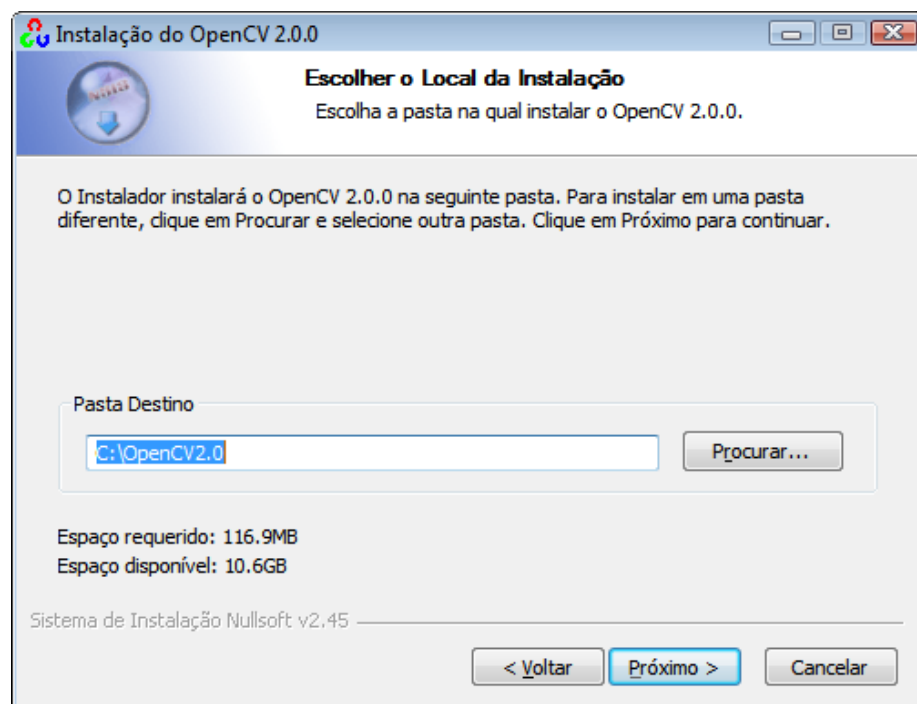
3. Na tela anterior clique em “Próximo” para continuar;



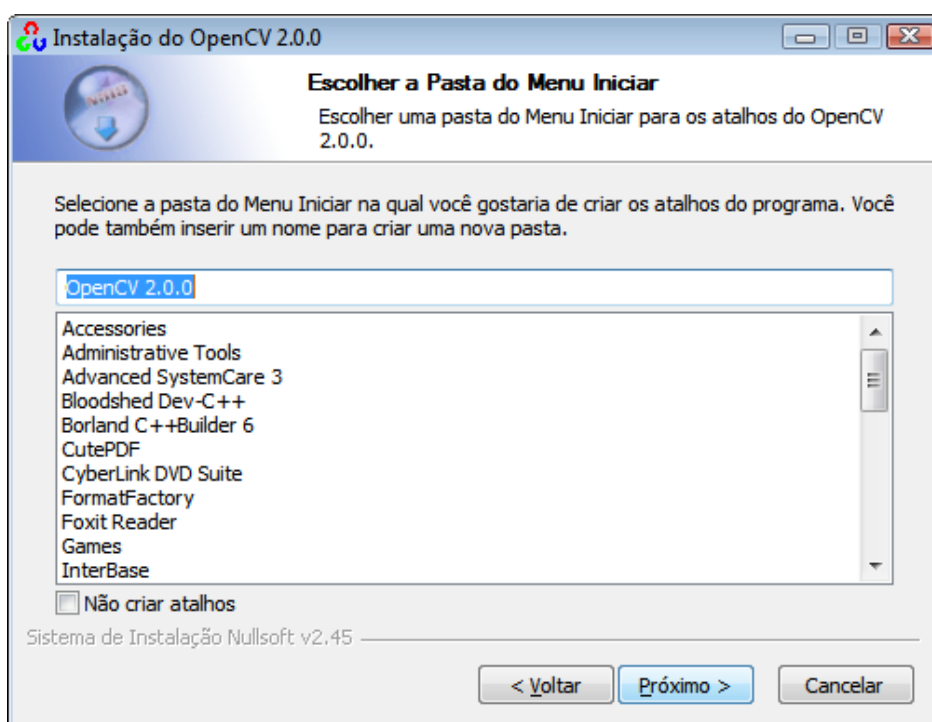
4. Após ler o acordo de licença clique em “Eu concordo”;



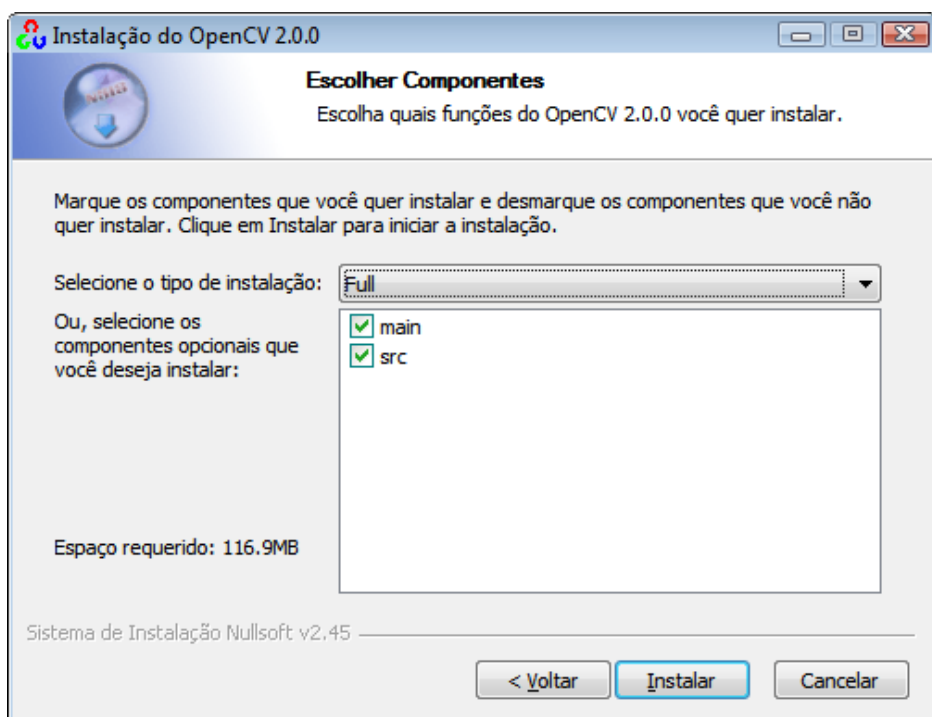
5. Na tela anterior, marque a terceira opção “Add OpenCV to the system PATH for current user” e clique em “Próximo”;



6. Escolha a pasta de destino como sendo a padrão: C:\OpenCV2.0. Em seguida, clique em “Próximo”;



7. Escolha a pasta do menu iniciar e clique em “Próximo”;



8. Escolha o tipo de instalação “full” e clique em “instalar”.



- **Configurando Windows para execução de arquivos do OpenCV:**

1. No ícone “*Meu Computador*” clique com o botão direito do mouse e selecione *Propriedades*.
2. Clique em “*Avançado*” e após no botão “*Variáveis do Ambiente*” na parte inferior da janela.
3. Nas “*Variáveis do usuário*” localize a variável PATH, acrescente o caminho para a biblioteca OpenCV (o padrão é: C:\OpenCV2.0\bin).
4. Adicione o caminho para a biblioteca de interface (o padrão é C:\OpenCV2.0\include\opencv). Lembre de separá-las com “;”.
5. OBS: a PATH pode ser atualizada automaticamente na instalação, mas é preciso verificar. Talvez, seja preciso reiniciar.

- **Para compilar utilizando o Dev-C++:**

1. Abrir o Bloodshed Dev-C++ (versão testada: 4.9.9.2).
2. Para “linkar” os programas do Dev C++ com as bibliotecas do OpenCV:
  - i. Clicar em “*Ferramentas*” e selecionar “*Opções do compilador*”. O sistema deve abrir uma janela com a aba “*Compilador*” selecionada.
  - ii. Marcar a caixa “*Adicionar estes comandos à linha de comando do linker*” e copiar os seguintes comandos:
    - C:/OpenCV2.0/lib/libcv200.dll.a
    - C:/OpenCV2.0/lib/libcvaux200.dll.a
    - C:/OpenCV2.0/lib/libcxcore200.dll.a
    - C:/OpenCV2.0/lib/libcxts200.dll.a
    - C:/OpenCV2.0/lib/libhighgui200.dll.a
    - C:/OpenCV2.0/lib/libml200.dll.a
3. Na aba “*Diretórios*”, em “*Binários*” adicionar o caminho:
  - C:\OpenCV2.0\bin
4. Em “*C Includes*” adicionar o caminho:
  - C:\OpenCV2.0\include\opencv
5. Em “*C++ Includes*” adicionar o mesmo caminho acima.

Em “*Bibliotecas*” adicionar o caminho: C:\OpenCV2.0\lib. Dê “OK”.

## ANEXO B. Código detalhado da função de treinamento da RNA

```

void executarRNA(int num_quadros, int cod, int num)
{
    CvANN_MLP mlp; //RNA
    char arquivo1[100],arquivo2[100],arquivo3[100], buffer[10];
    char buffer2[10];

    FILE *f, *fp;
    int n_amostras = 0; //quantidade de entradas para treinamento
    int i, j; //controlar linhas e colunas dos arquivos
    int col_entrada = max_amostras; //quantidade de colunas com dados de entrada

    int num_entrada,repete; //quantidade de entradas
    if(num_quadros>max_entradas) num_entrada = max_entradas;
    else num_entrada = num_quadros;

    int num_saida = num_entrada; //quantidade de saidas
    int col_saida=1; //quantidade de colunas com dados de saida
    double in[num_entrada][col_entrada], out[num_saida],resultado[num_saida];

    printf(" Executando RNA...\n");

    //Criando as matrizes
    //Dados de entrada. Matriz de ordem (num_entrada x col_entrada)
    CvMat *dados = cvCreateMat(num_entrada, col_entrada, CV_64FC1);

    //Dados de saida. Matriz de ordem (num_saida x col_saida)
    CvMat *respostas = cvCreateMat(num_saida, col_saida, CV_64FC1);

    //Armazenara a Saida produzida pela rede
    CvMat *saida_rede = cvCreateMat(num_saida, col_saida, CV_64FC1);

    //Matriz de representacao da RNA com 4 camadas
    CvMat *camadas_rede = cvCreateMat(4, 1, CV_32SC1);

    //Definindo o numero de neuronios em cada camada da RNA
    /*      Camada 1: 1200 neuronios (entradas)
           Camada 2: 20 neuronios (camada oculta)
           Camada 3: 20 neuronios (camada oculta)
           Camada 4: 1 neuronios (1 saida)
    */
    int camadas []= { col_entrada, 20, 20, col_saida } ;

    if(cod==1)
    {
        strcpy(arquivo2,"treinamento/CirculoNet");
    }
}

```

```

        strcpy(arquivo3,"resultados/CirculoNet");
    }
    else if(cod==2)
    {
        strcpy(arquivo2,"treinamento/RetanguloNet");
        strcpy(arquivo3,"resultados/RetanguloNet");
    }
    else
    {
        strcpy(arquivo2,"treinamento/TrianguloNet");
        strcpy(arquivo3,"resultados/TrianguloNet");
    }

    itoa(num,buffer2,10);
    strcat(arquivo3,buffer2);
    strcat(arquivo2,buffer2);
    strcat(arquivo2,".rna");
    strcat(arquivo3,".txt");

    for(i=0;i<num_entrada;i++)
    {
        strcpy(arquivo1,"arquivos/");
        itoa(i,buffer,10); // convertendo inteiro em string
        strcat(arquivo1,buffer);
        strcat(arquivo1,".txt");

        //Lendo um arquivo com dados para treinamento
        if((f=fopen(arquivo1,"r"))==NULL)
        {
            fprintf(stderr," ERRO: arquivo %s nao foi aberto!!!\n",arquivo1);
            return;
        }

        //Obtendo numero de amostras
        fscanf(f, "%d %lf\n", &n_amostras, &out[i]);
        col_entrada = n_amostras;

        //Lendo e Reunindo os dados de treinamento
        printf(" Lendo dados de quadro %d...\n",i);

        for (j=0; j<col_entrada; j++)
            fscanf(f,"%lf\n",&in[i][j]);

        fclose(f); //fechando arquivo
    }

    //inicializando matrizes
    cvInitMatHeader(dados, num_entrada, col_entrada, CV_64FC1, in);

```

```

cvInitMatHeader(respostas, num_saida, col_saida, CV_64FC1, out);
cvInitMatHeader(camadas_rede, 1, 4, CV_32SC1, camadas);

//Verifica se o arquivo de treinamento da rede jah existe
if((fp=fopen(arquivo2,"r"))==NULL)
    mlp.create(camadas_rede); //Criando a RNA
else
{
    mlp.load(arquivo2);
    fclose(fp); //fechando arquivo
}

do
{
    printf(" Ciclo %d...\n",i++);
    mlp.train(
        dados, //vetor de entradas
        respostas, //vetor de saidas
        saida_rede, //vetor de pesos
        0, //vetor opicional mostrando colunas de entrada e de saida
        CvANN_MLP_TrainParams( cvTermCriteria( //criterio de finalizacao
            CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, /*CV_TERMCRIT_ITER*/
            10000, //numero maximo de iteracoes
            0.000001 // precisao necessaria
        ),
        CvANN_MLP_TrainParams::BACKPROP,
/*CvANN_MLP_TrainParams::RPROP*/
        0.01,
        0.01
    )
);

    // Salvar classificador para o arquivo, se necessário
    mlp.save( arquivo2 );

    // Testando a rede treinada
    mlp.predict(dados,saida_rede);

    for(j=0, repete=0;j<num_entrada;j++)
    {
        resultado[j] = CV_MAT_ELEM(*saida_rede,double,j,0);
        if(abs(resultado[j]-out[j])>erro_treino)
        {
            repete = 1;
            break;
        }
    }
}

```

```

} while(repete);

//Lendo um arquivo com dados para treinamento
if((fp=fopen(arquivo3,"w"))==NULL)
{
    fprintf(stderr," ERRO: arquivo %s nao foi aberto!!!\n",arquivo3);
    return;
}
for(i=0;i<num_entrada;i++)
    fprintf(fp, "%.5lf\n", resultado[i]);

fclose(fp); //fechando arquivo

// desalocando recursos
cvReleaseMat(&dados);
cvReleaseMat(&respostas);
cvReleaseMat(&saida_rede);
cvReleaseMat(&camadas_rede);
printf(" Fim de execucao da RNA...\n");
}

```

## ANEXO C. Código detalhado da função de reconhecimento de padrões

```

double loadRNA(char *file_treino)
{
    CvANN_MLP mlp; //RNA
    FILE *f, *fp;
    int n_amostras = 0; //quantidade de amostras para treinamento
    int i, j; //controlar linhas e colunas dos arquivos

    int col_entrada; //quantidade de colunas com dados de entrada
    int num_entrada=1; //quantidade de entradas
    int num_saida=1; //quantidade de saidas
    int col_saida=1; //quantidade de colunas com dados de saida
    double in[max_amostras],resultado;

    printf(" Executando RNA...\n");

    //Lendo um arquivo com dados para treinamento
    if((f=fopen("arquivos/EntradaRede.txt","r"))==NULL)
    {
        fprintf(stderr," ERRO: arquivo 'arquivos/EntradaRede.txt' "
            "nao foi aberto!!!\n");
        return 0;
    }
}

```

```

//Obtendo numero de amostras
fscanf(f, "%d\n", &n_amostras);
col_entrada = n_amostras;

//Criando as matrizes
//Dados de entrada. Matriz de ordem (num_entrada x col_entrada)
CvMat *dados = cvCreateMat(num_entrada, col_entrada, CV_64FC1);

//Armazenar a Saida produzida pela rede
CvMat *saida_rede = cvCreateMat(num_saida, col_saida, CV_64FC1);

//Lendo e Reunindo os dados de treinamento
for (i=0; i<col_entrada; i++)
    fscanf(f, "%lf\n", &in[i]);

fclose(f); //fechando arquivo

//Carregando arquivo de treinamento
mlp.load(file_treino);

//Executando a rede treinada
mlp.predict(dados, saida_rede);

resultado = CV_MAT_ELEM(*saida_rede, double, 0, 0);

// desalocando recursos
cvReleaseMat(&dados);
cvReleaseMat(&saida_rede);

printf(" Fim de execucao da RNA...\n");
return resultado;
}

```