UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
ESCOLA POLITÉCNICA
DEPARTAMENTO DE ELETRÔNICA E DE COMPUTAÇÃO

Reconhecimento de padrões de calibração em estereofotogrametria através de redes neurais

Autor:

Juliana Calixto Acchar

Orientador:

Prof. Julio Cesar Boscher Torres, D.Sc.

Co-orientador:

Prof. José Gabriel Rodriguez Carneiro Gomes, Ph.D.

Examinador:

Profa. Mariane Rembold Petraglia, Ph.D.

DEL
JUNHO DE 2008

Dedicatória

A Deus, meus pais, minha irmã, meu irmão e meu tio Jorge

Agradecimentos

À minha família inteira, pelo amor, aprendizado, apoio e conselhos que me fizeram alcançar este objetivo.

Aos meus amigos de classe, agradeço pelo companheirismo e pelas incontáveis ajudas. Em especial ao meu grande amigo Rafael, por ser muito prestativo e me apoiar sempre.

Aos meus amigos de sempre, que me incentivaram e presenciaram toda esta minha trajetória.

Ao Rubens, meu grande companheiro de classe e de vida, essencial em todas as minhas conquistas.

Ao meu professor de Física do Ensino Médio, Orlandino Otávio, agradeço por sua capacidade de ensinar além de fórmulas e conceitos, com atitudes que me fizeram acreditar que somos bons o suficiente para o que quisermos tentar.

Ao meu pai, também engenheiro e mestre Roberto Acchar, agradeço pela importância que sempre deu aos estudos, se tornando o meu maior exemplo.

Aos professores do Departamento de Engenharia Eletrônica e de Computação da UFRJ, fundamentais para a minha formação profissional.

Em especial, ao meu orientador Julio Boscher Torres, agradeço pela paciência, clareza em ensinar e competência para conduzir um projeto, por se mostrar presente e solícito durante todo o decorrer do trabalho. Ao meu co-orientador José Gabriel Gomes, agradeço pela ajuda e pelo excelente trabalho que desempenha na universidade.

Resumo

Neste trabalho desenvolveu-se um sistema de reconhecimento de padrões utilizando redes neurais. Os padrões a serem reconhecidos são alvos pré-definidos utilizados no processo de calibração dos sistemas de dimensionamento por estereofotogrametria e de reconstrução de cenas 2D e 3D.

O processo de calibração desses sistemas requer a ação humana de relacionar o pixel central de cada alvo à sua posição espacial real, relativa a um ponto de espaço. Este é um processo demorado, que requer precisão e paciência do apurador, que está sujeito a erros devido à repetição de apurações e de similaridade dos alvos. A associação equívoca de um alvo invalida toda a calibração. Como forma de evitar tais erros e automatizar o processo de calibração, foi projetada um rede neural capaz de identificar, em uma imagem, pixels centrais dos alvos, para que uma rotina possa, automaticamente, associá-los aos pontos do objeto de calibração.

A rede projetada apresenta um desempenho muito bom em seu treinamento, considerando um pré-processamento com Análise de Componentes Principais (PCA) e um pós-processamento de busca de centros de classes por "clusterização".

Palavras-Chave

- Processamento de Imagens
- Rede Neural
- Análise de Componentes Principais (PCA)
- “Clusterização”
- Estereofotogrametria

Sumário

Índice de Figuras

Índice de Tabelas

Glossário e Abreviaturas

Neurônios

Unidades de processamento interconectadas e designadas para a solução de um problema determinado. As operações realizadas pelos neurônios baseiam-se no processamento apenas de dados locais, isto é, apenas nas entradas recebidas por suas conexões.

Pixel

Pixel é a menor unidade em uma imagem digital. Um conjunto de milhares de pixels forma uma imagem inteira. O termo megapixel é muito utilizado em câmeras digitais.

JPEG

JPEG (Joint Photographic Experts Group) é um método de compactação de imagens com perdas. Os algoritmos são baseados na transformada DCT (Discrete Cosine Transformation), para descartar as partes menos significativas da imagem em termos de como ela é percebida pelo olho humano. Como é uma compactação com perdas, em cada edição haverá perda de dados. É o formato mais utilizado, atualmente, em câmeras digitais devido ao seu alto poder de compressão e conseqüente redução de espaço para armazenamento.

BMP

BMP (bitmap) é o formato de imagem de bitmap padrão do Windows em computadores compatíveis com esse sistema operacional e com o DOS. Não há compressão com perdas e se baseia no mapeamento de pixels.

Capítulo 1

Introdução

1.1 Motivação

O processo de estereofotogrametria, bastante empregado em reconstrução de cenas a partir de imagens e dimensionamento de objetos, se utiliza dos parâmetros de calibração das câmeras para determinar seu resultado. Para realizar a calibração de câmeras é necessário mapear pixels existentes na cena tridimensional em coordenadas de duas dimensões. Para automatizar esse processo de mapeamento e torná-lo sem erro, o emprego da técnica de redes neurais se mostra a melhor solução.

1.2 Objetivo

Este trabalho possui o objetivo de implementar uma rede neural backpropagation que reconheça padrões específicos em uma imagem e extraia informações do posicionamento dos alvos que são desejados. Utilizando redes neurais, o processo de extração das informações de tais imagens se torna automático e também permite identificar padrões tão complexos quanto os reconhecidos pelos seres humanos. Este assunto é de extrema importância em diversas áreas da engenharia, porém será voltado para o processo de automatização de calibração de câmeras em estereofotogrametria.

A etapa de implementação consiste na elaboração de rotinas computacionais capazes de simular os modelos matemáticos estudados previamente. É necessário capturar as imagens contendo os padrões que devem ser reconhecidos pela rede. Essas imagens devem ser convertidas para tons de cinza e a partir daí, separar os alvos (padrões que devem ser reconhecidos) e os não-alvos em janelas de tamanho pré-determinado (imagens transformadas em matrizes). A rotina computacional do

treinamento *Backpropagation* consiste na "apresentação" das matrizes contendo os alvos e não-alvos para a rede neural. A rede deve ser capaz de generalizar os exemplos e apresentar uma baixa (em torno de 10%) taxa de erros. Com a rede treinada, é realizado o teste. Este se baseia na carga de uma imagem e na varredura à procura de alvos. Ao fim, devem ser mostradas as coordenadas do centro dos alvos presentes nessa imagem.

Ao final do trabalho, pretende-se avaliar o desempenho da rede neural e suas vantagens. Além disso, pode-se generalizar essa rede para que ela reconheça padrões diferentes dos apresentados na calibração de câmeras em estereofotogrametria e ela se torne uma importante ferramenta para inúmeras aplicações.

1.3 Metodologia

Para que o objetivo de determinar os alvos na figura contendo um *grid* (Figura 2) seja alcançado, devem ser seguidas algumas etapas. Todas elas devem ser executadas com bastante cuidado a fim de obter o menor erro possível nos resultados finais. Além disso, para automatizar alguns procedimentos, foram criadas rotinas, explicadas a seguir.

Foram realizados testes variando os dados de entrada para verificar a precisão do treinamento e da execução da rede neural, a fim de obter uma unidade de medida de qualidade.

O processo é dividido, em linhas gerais, nas etapas mostradas na Figura 1. Esse fluxograma nos traz a visão geral dos pré-requisitos de cada procedimento efetuado neste trabalho.

Figura 1 – Visão geral do processo

Capítulo 2

Fundamentos Teóricos

2.1. O Processo de Calibração

Dois sistemas desenvolvidos no PADS (Laboratório de Processamento Digital e Analógico de sinais da UFRJ) necessitam de um processo de calibração seja para estereofotogrametria ou para reconstruir objeto em cenas 2D e 3D.

Neste processo, diversos alvos – pixels centrais de uma figura padrão – precisam ser associados a pontos do espaço real tridimensional (x,y,z). Cada alvo corresponde a um ponto no espaço.

De posse de uma lista de pontos associados (mapeamento das coordenadas da imagem para coordenadas do espaço 3D), uma rotina não-linear de minimização de erro quadrático atualiza um conjunto de parâmetros utilizados para realizar a projeção do espaço 3D (real) no espaço de imagem (2D), como pode ser visto detalhadamente na tese "Método Robusto para a Calibração de Câmeras em Estereofotogrametria" do autor Lenildo Carqueija Silva.[1]

A aquisição e tratamento dos dados de entrada é o primeiro passo para posterior treinamento da rede. Para isso, foram coletadas fotos que possuem um *grid* posicionado, como na Figura 2. As mesmas imagens com rotações de 30 e 60 graus também foram consideradas para coleta de novos dados de treinamento da rede, como o exemplo da Figura 3. É importante ressaltar que com uma maior variedade de fotos do *grid* é possível melhorar a etapa de treinamento da rede neural, já que sua lógica baseia-se em exemplos "aprendidos" previamente. Pode-se, então, notar que foram escolhidas imagens bem distintas do *grid* ao comparar a Figura 2 com a Figura 4.

O Grid é um objeto (formando eixos ortogonais, como mostrado na Figura 2)

contendo um determinado quadriculado que destina-se a localizar pontos de referência para posterior calibração de câmeras.

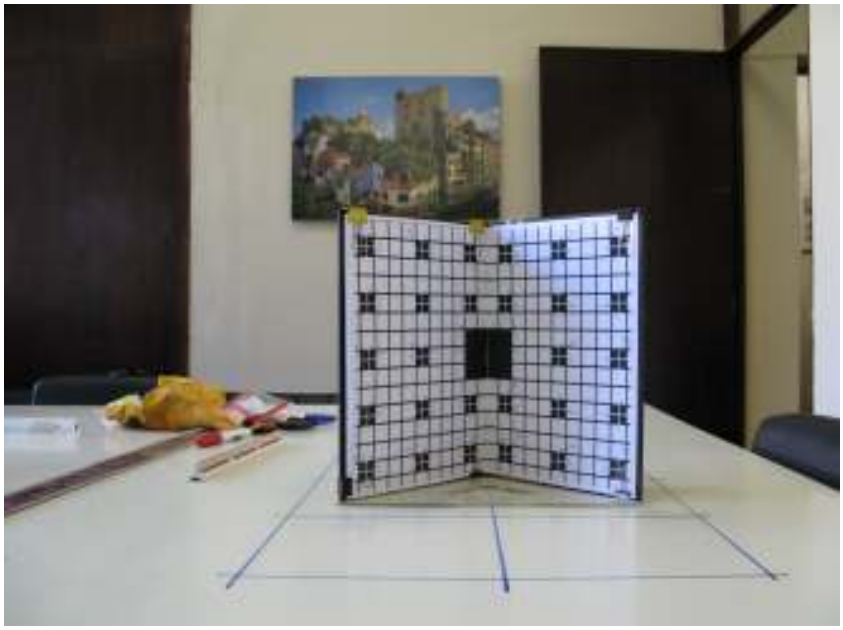

Figura 2 - Exemplo de imagem com grid posicionado

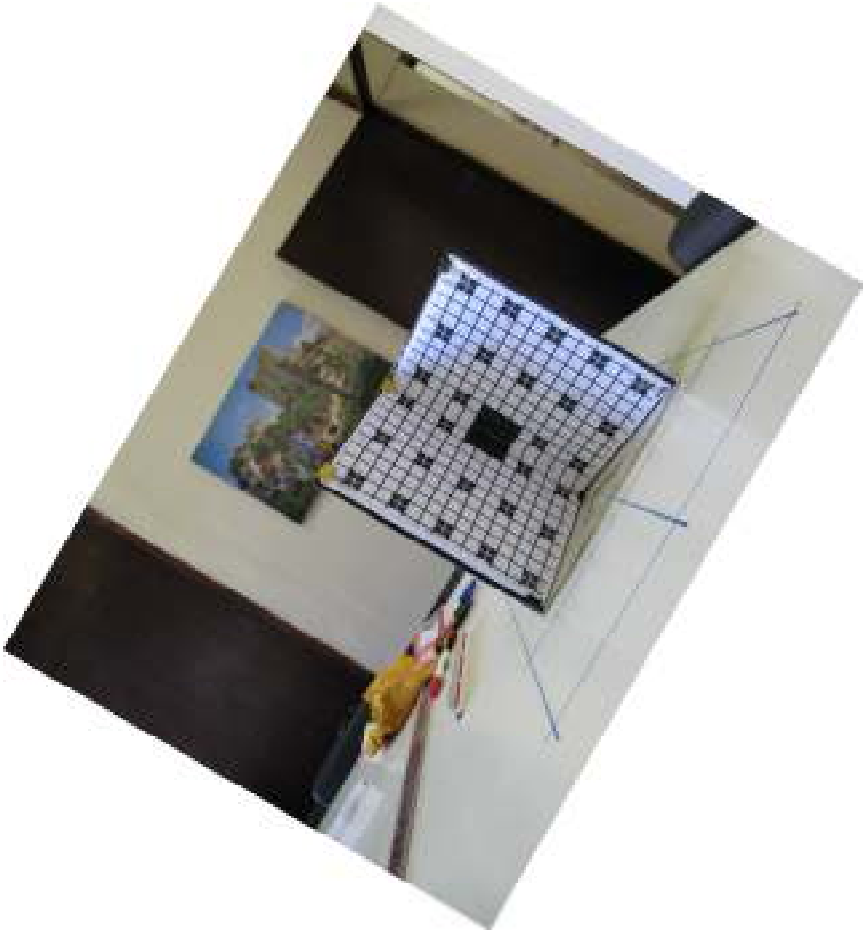

Figura 3 - Exemplo da imagem com grid posicionado rotacionada em 60 graus

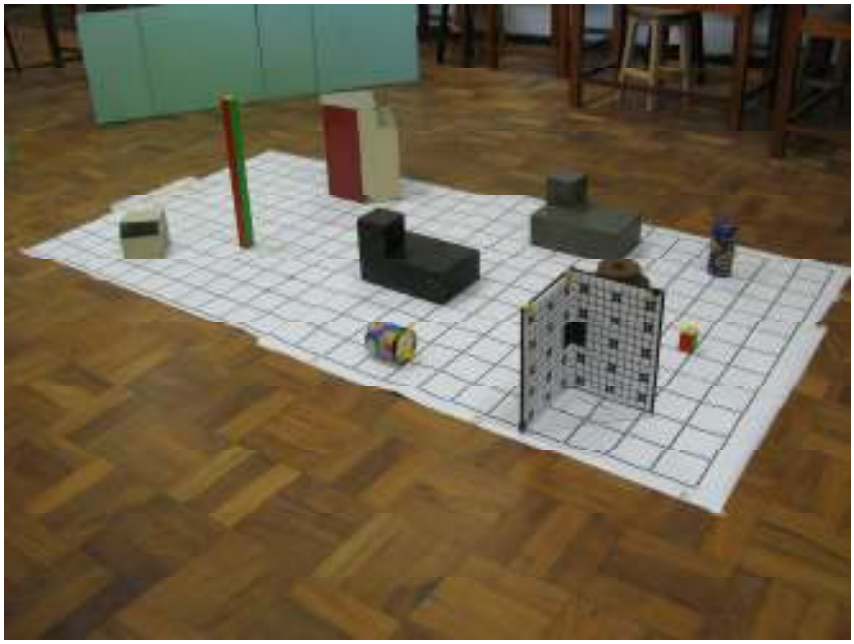

Figura 4 - Mais um exemplo de imagem com grid posicionado

Nesta etapa ocorre a preparação dos dados para o treinamento da rede neural. O objetivo do projeto é identificar as coordenadas de pontos específicos do grid e para isso, são escolhidos, manualmente, alvos na figura com o grid. Essa figura será mapeada como uma matriz em que haverá elementos alvo ou não-alvo e que serão estes dois os parâmetros de entrada na rotina de treinamento. Nas Figuras 5 e 6 há exemplos de imagens que terão seus pontos mapeados em alvos ou não-alvos e formatados em matriz.

Figura 5 - Exemplos de alvos

Elementos alvos são padrões de imagem que a rede neural deverá reconhecer. Para isso, são coletadas diversas imagens que reproduzem um mesmo tipo de alvo para um correto treinamento da rede, como mostrado na Figura 5.

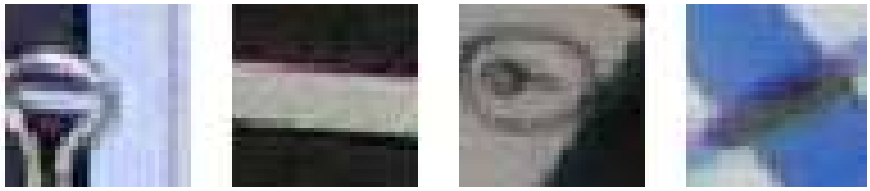

Figura 6 – Exemplos de não-alvos

Elementos não-alvos são partes de imagens utilizadas no treinamento da rede neural, como exemplos do que não deve ser identificado como alvo. Alguns elementos não-alvos podem ser identificados na Figura 6.

Para um funcionamento adequado, os dados devem ser separados em: dados para treinamento e dados para testes. Com isso, há a garantia de que há generalização no momento do treinamento, bom desempenho com dados distintos no momento do teste simulando condições diversas e prevenção de apresentação de tendências por parte da rede neural.

Com o intuito de automatizar esse processo, foi desenvolvida a rotina *"converte_imagens"* (diagrama de fluxo mostrado na Figura 7). Sua função principal é mapear, em forma de matriz, as janelas das imagens consideradas alvos ou não-alvos. Estas janelas possuem 29x29 pixels, pois foi um tamanho padrão testado, que compreendia os alvos integralmente quando a grade estava próxima (60 cm) e distante (100 cm).

Primeiramente, há conversão da imagem para níveis de cinza e depois, cada imagem de 29x29 pixels é transformada em um vetor de 1x841. Os vetores de cada imagem são acumulados em uma matriz de alvos (se for alvo) ou em uma matriz de não-alvos (se não for alvo), ambos pré-definidos pelo apurador. Portanto, as saídas dessa rotina são duas matrizes, compostas por elementos alvos e outra por elementos não-alvos. Essas matrizes são compostas por vetores, cada um equivalente a uma imagem (dimensão de 1x841). Na Tabela 1 são mostrados os parâmetros de entrada e saída da rotina *"converte_imagens"*. O código-fonte desta função está mostrado no Apêndice A.

Figura 7 - Diagrama de fluxo da rotina de tratamento de dados

É importante lembrar que todas as imagens passíveis de serem entrada desta rotina, devem ser convertidas do formato de compressão JPEG (utilizado pela maioria das câmeras digitais, atualmente) para o formato BMP, no qual não existe perda de informação. (A perda já ocorreu no JPEG).[2]

Tabela 1 – Parâmetros de entrada e saída da rotina de conversão de imagem

converte_imagens	
Entrada de dados	1. Nome dos arquivos BMP (imagem) contendo alvos; 2. Nome dos arquivos BMP (imagem) contendo não-alvos.
Saída de dados	1. Matriz (conversão da imagem para matriz) contendo os alvos (cada linha representa uma imagem); 2. Matriz (conversão da imagem para matriz) contendo os não-alvos (cada linha representa uma imagem).

2.2. Redes Neurais

Redes neurais são modelos matemáticos de processamento de dados que se baseiam em sistemas nervosos de organismos inteligentes que adquirem conhecimento por aprendizado. Essas técnicas são compostas de centenas a milhares de unidades de processamento, chamadas neurônios, altamente interconectadas para resolverem um problema específico.

Tais neurônios, artificiais, são geralmente conectados por canais de comunicação que estão associados a determinado peso. Essas unidades fazem operações apenas sobre seus dados locais, que são entradas recebidas pelas suas conexões. O comportamento inteligente de uma rede neural artificial vem das interações entre os neurônios, responsáveis pelo processamento da rede.

Na figura 8 podemos observar a estrutura básica de um neurônio. Consiste em várias entradas (representadas por X_1 até X_n), uma porta de entrada para treinamento, uma porta de seleção para escolher se o neurônio irá treinar ou processar realmente as entradas, pesos (representados por W_1 até W_n) e uma porta de saída.

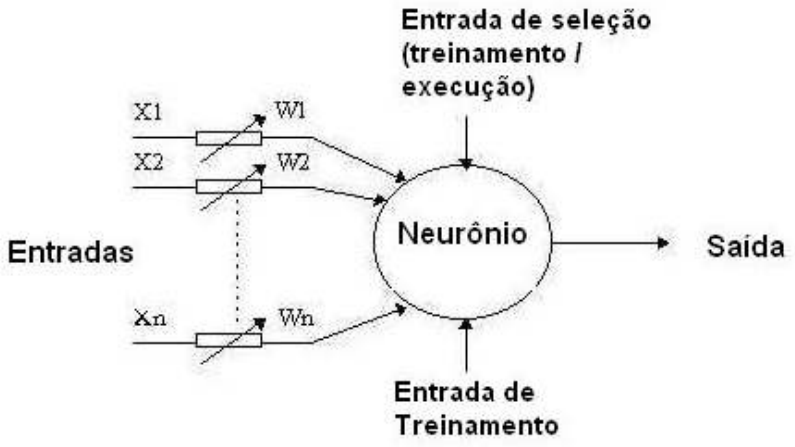

Figura 8 – Estrutura de um neurônio

Historicamente esta técnica foi criada em 1943, por McCulloch e Pitts, que tinham a intenção de criar uma máquina inspirada no cérebro humano. Em 1949, Hebb propôs uma lei de aprendizagem específica para as sinapses dos neurônios. A construção do primeiro neuro computador, denominado Snark, por Mavin Minsky aconteceu no decorrer dos anos 50. O Snark operava com sucesso a partir de um ponto de partida técnico, ajustando seus pesos automaticamente, entretanto, ele nunca executou qualquer função complexa de processamento de informação. A partir dos anos 80, muitos pesquisadores começaram a explorar esse ramo da tecnologia. Institutos de pesquisa sobre esse assunto foram criados e foi realizada a primeira conferência sobre redes neurais, em São Francisco, 1987.

As técnicas de redes neurais podem ser utilizadas em diversas aplicações: Funções de aproximação incluindo predição de séries temporais, classificação incluindo reconhecimento de padrões ou de novos elementos em um cenário, processamento de dados em geral, dentre outras. Separando em grandes áreas, podem ser enumeradas algumas utilizações:

- Sistemas de identificação e controle (controle de veículos, processos);
- Tomadas de decisões;
- Diagnósticos médicos;
- Aplicações financeiras;
- Reconhecimento de movimentos (de objetos ou gestos de pessoais);
- Reconhecimento de padrões (sistemas de radar, identificação de faces ou objetos).

Enquanto a computação convencional utiliza métodos de aproximação e um conjunto de instruções para resolver determinado problema, a técnica de redes neurais se baseia em exemplos previamente conhecidos. Para isso, é necessário que haja um treinamento dessa rede. Algumas vantagens de se utilizar redes neurais ao invés de técnicas convencionais são:

- Flexibilidade (variando de aplicação para aplicação);
- Adaptação (aprendizado a cada execução);
- Tolerância à variação das informações a serem processadas.

Para o perfeito funcionamento de uma rede neural é necessário que ela seja

treinada. Para isso são usados algoritmos com regras bem definidas para a solução do problema, normalmente ajustando os pesos associados aos neurônios. Existem diversos algoritmos de treinamento de redes neurais, porém, neste projeto foi implementado um treinamento supervisionado, isto é, um agente externo indica à rede a resposta desejada para o padrão de entrada, utilizando o algoritmo Backpropagation.[3]

2.3. Análise de Componentes Principais

A análise de componentes principais (**Principal Components Analysis**), também chamada de Transformada de Karhunen-Loève, é um método de ordenação que se baseia no cálculo dos autovalores e respectivos autovetores de uma matriz de variâncias-covariâncias ou de uma matriz de coeficientes de correlação de variáveis.

Esta é uma técnica de transformação de variáveis onde, primeiramente, são calculados os autovalores e autovetores de uma matriz de correlações de variáveis. O primeiro autovalor determinado corresponde à maior porcentagem da variabilidade total e assim sucessivamente. Com isto, é possível reduzir drasticamente as informações linearmente dependentes e redundantes na entrada da rede neural, agilizando sua performance.[4]

2.4. Clusterização

Clusterização é a classificação não-supervisionada de dados, formando agrupamentos ou clusters. Ela representa uma das principais etapas de processos de análise de dados e envolve a organização de um conjunto de padrões (usualmente representados na forma de vetores de atributos ou pontos em um espaço multidimensional – espaço de atributos) em clusters, de acordo com alguma medida de similaridade.

Intuitivamente, padrões pertencentes a um dado cluster devem ser mais "similares" entre si do que em relação a padrões pertencentes a outros clusters. No caso específico deste projeto, podemos dizer que o objetivo da *clusterização* é obter uma coordenada média a partir de uma "nuvem" de pontos (pixels).

Capítulo 3

Projeto do Sistema de Identificação de Alvos

3.1. Seleção do Conjunto de Dados

O conjunto de dados deve ser dividido em duas categorias: treinamento e validação. Os dados de treinamento são usados para ajustar os parâmetros da rede neural e podem ser usados diversas vezes (repetidamente) durante o período de treinamento. O conjunto de teste ou validação não é apresentado para treinamento e serve para avaliar o desempenho da rede com relação a dados nunca antes processados. Dentro de cada categoria devem-se selecionar dois subconjuntos de dados: alvos e não alvos.

Os alvos e não alvos são imagens pré-definidas pelo apurador, processadas para níveis de cinza e linearizadas para que possam ser tratadas computacionalmente. Neste projeto, definiu-se como alvo janelas de 29x29 pixels que contivessem como centro os cruzamentos das linhas ortogonais brancas de cada *grid*. E os não-alvos foram definidos como quaisquer janelas de 29x29 pixels dentro da imagem, que não contivessem como centro os cruzamentos das linhas ortogonais brancas do *grid*.

3.2. Redução do Espaço de Entrada

Como o vetor de entrada da rede é a linearização dos pixels das janelas (alvos ou não-alvos), o tempo de processamento será elevado e haverá muita redundância de dados. Foi utilizado, então, o PCA (Análise de Componentes Principais) para minimizar estas redundâncias e melhorar o desempenho da rede.

Estas redundâncias podem ser percebidas através do gráfico das componentes em ordem crescente e cumulativa, mostrada na Figura 9 abaixo.

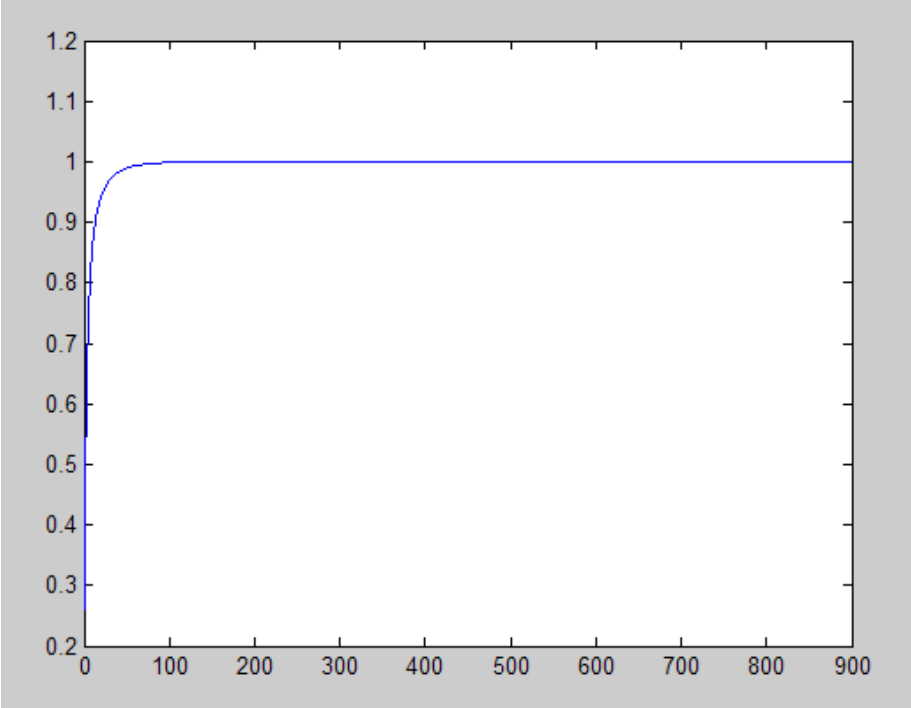

Figura 9 – Gráfico das componentes em ordem crescente e cumulativa

3.2.1. Critério de Seleção dos Componentes

Verificou-se, através do gráfico de PCA da Figura 9, que 90% da energia é mantida se as primeiras 14 componentes forem utilizadas. Assim, passamos de 841 para 14 elementos o vetor de entrada da rede, obtendo velocidade de processamento maior e melhor resultado no treinamento da rede. (Taxa de erro de treinamento= 0,1%)

3.3. Rede Neural

3.3.1. Topologia

A rede neural utilizada possui um treinamento seqüencial com 2 camadas (conforme ilustrado na Figura 10), taxa de aprendizado de 0,01, momento constante igual a 0 e um erro quadrático instantâneo dado pela expressão erro = [erro(e*e)/2] . Com a Análise de Componentes Principais, a rede passou a ter 14 entradas (ao invés de 841) e uma saída, a qual é o resultado do processo de treinamento.

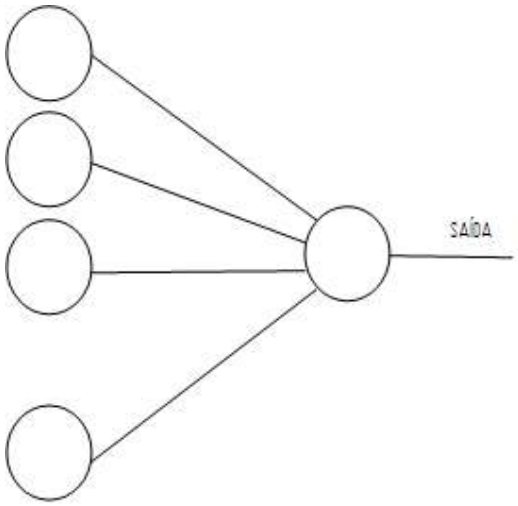

Figura 10 – Topologia da rede neural Backpropagation utilizada

3.3.2. Treinamento da Rede Neural

A rede neural desenvolvida utiliza o algoritmo Backpropagation para o treinamento, que consiste em dois passos. No primeiro, um padrão é apresentado às unidades da camada de entrada e, a partir desta camada, os neurônios calculam sua resposta que é produzida na camada de saída. O erro é calculado e, no segundo passo, este é propagado a partir da camada de saída até a camada de entrada, e os pesos das conexões das unidades das camadas internas vão sendo modificados.

Após a rede ter sido treinada adequadamente, a execução da rede, em linhas gerais, consiste em:

- Sinais são apresentados à entrada;
- Cada sinal é multiplicado por um peso, que indica a sua influência na saída do neurônio;
- É feita a soma ponderada dos sinais, que produz um nível de atividade;
- Se este nível de atividade exceder limite (*threshold*), o neurônio produz uma determinada resposta de saída.

Um exemplo, para ilustrar o funcionamento desta técnica para reconhecimento de padrões, está descrito a seguir. Neste exemplo, a rede deve ser treinada para identificar a letra "T" ou a letra "H". Na figura 11, são mostradas as saídas que foram especificadas de acordo com a entrada.

Figura 11 - Saídas esperadas em uma rede neural para reconhecimento de padrões

Observa-se, de acordo com a Figura 12, que dependendo da entrada, a rede neural resulta na saída esperada. Os sinais de entrada nem sempre serão idênticos ao padrão que é desejado (mostrado nas duas entradas da simulação da figura 12), o que faz com que a rede busque um formato que mais se aproxime dos padrões que ela reconhece. O que também pode ocorrer é a entrada não possuir nenhum padrão definido, logo não reconhecido pela rede neural. Esta situação resulta em uma saída aleatória, como mostrada na figura 13.

Figura 12 – Resultados esperados em simulações de reconhecimento de padrões

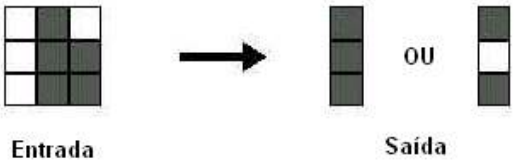

Figura 13 - Saída indefinida em simulação de reconhecimento de padrões

Apesar de ser relativamente fácil de ser projetada, uma rede usando backpropagation pode ter limitações, tais como: dificuldade de obtenção da lógica de execução da rede para alcançar o resultado final e alto tempo de treinamento, necessitando de muitos ciclos para serem obtidas taxas de erros aceitáveis à aplicação.

Para executar o treinamento corretamente, a etapa de coleta e tratamento de dados é pré-requisito. Para realizar a configuração e treinamento da rede, são utilizadas as matrizes de saída da etapa anterior. São duas matrizes: uma composta pelos elementos alvos e outra pelos não-alvos.

O tempo de treinamento de uma rede utilizando Backpropagation é variável, dependendo de diversos fatores para que ela alcance resultados com erro aceitável. Para que ela esteja bem treinada, é necessário levar em consideração a taxa de erro médio e a capacidade de generalização da rede. Quando a rede apresentar um ponto ótimo entre esses dois fatores, ela pode ser considerada bem treinada.

A rotina de treinamento "*treinarede*" implementada neste projeto tem como características principais o passo fixo de 0,01 e apresenta um treinamento seqüencial que tem como critério de decisão para avaliar se é alvo ou não é alvo o valor 0, ou

seja, no início do treinamento são atribuídos valores iguais a 1 para os alvos e valores iguais a -1 para os não-alvos. Na saída da rede, será considerado alvo, os vetores que possuírem valores acima de 0.

Após o treinamento e teste da rede neural implementada pela rotina *rede02* (rotina implementada antes da "*treinarede"*), foi percebido que os índices de erro apresentados no resultado final eram muito altos (por volta de 50%). A rotina *rede02* foi desenvolvida para utilizar toda a informação da imagem em seu treinamento. Uma nova rotina de treinamento, *treinarede,* foi especificada para utilizar apenas as informações relevantes e eliminar as redundâncias. Para isso, foi utilizada a técnica de Análise de Componentes Principais (PCA), que reduziu significativamente o erro da rede de 60% para 0,01% e que, na rede ótima, que será explicada no item 4.2 deste relatório, reduziu o número de componentes para treinamento de 841 (29 x 29) para 14 componentes.

Esta rotina possui as seguintes entradas e saídas, conforme mostrado na Tabela 2.

Tabela 2 – Rotina para treinamento da rede neural

treinarede	
Entrada de dados	1. Matriz com as imagens convertidas de alvos; 2. Matriz com as imagens convertidas de não-alvos;
Saída de dados	1. Gráfico mostrando a evolução do erro no treinamento da rede; 2. Variável "L" que é utilizada na execução da rede.

Para avaliação do treinamento da rede, é gerado um gráfico que nos permite avaliar a evolução do erro de avaliação por parte da rede, ou seja, a sua curva de aprendizagem, mostrada na Figura 14. É importante ressaltar que a rede foi elaborada de modo que, a cada treinamento, ela gere um gráfico de erro diferente.

A variável “L” (uma das saídas da rede neural) é um conjunto de dados que contém os pesos e as polarizações que foram atualizadas no processo de treinamento da rede.

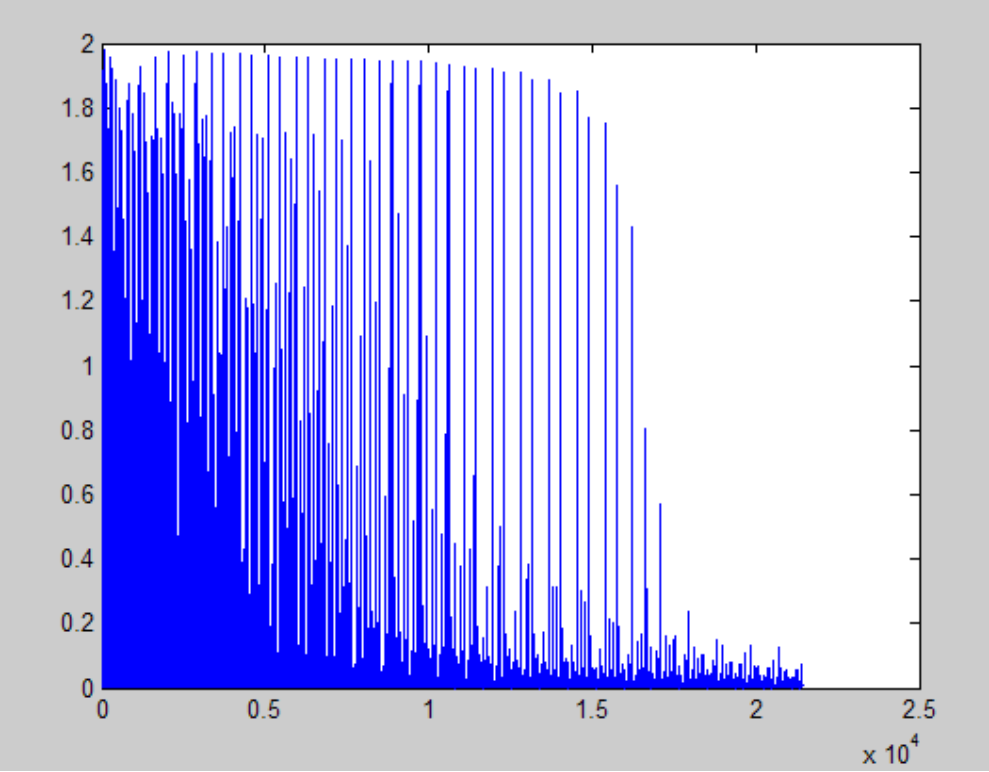

Figura 14 – Curva de aprendizagem de uma rede neural

3.4. Testes da Rede Neural

Após a etapa de treinamento da rede neural, é necessário realizar testes para verificar seu funcionamento. Para isso, podem ser utilizados dois tipos de dados de entrada para o referido teste: imagens utilizadas na etapa de treinamento e imagens diferentes das utilizadas no treinamento da rede.

Esses conjuntos de dados podem ser considerados suficientes para uma

análise do comportamento da rede. Os dados que não tiverem sido utilizados durante o treinamento servem para nos mostrar o nível de generalização da rede, isto é, o quanto a rede consegue identificar o padrão ao qual ela foi treinada com imagens nunca antes "observadas" por ela.

Para a realização dos testes, foi desenvolvida uma rotina computacional nomeada *acha_alvos05* (diagrama de fluxo na Figura 15) que possui as entradas e saídas mostradas na Tabela 3 e seu código-fonte descrito no Apêndice A.

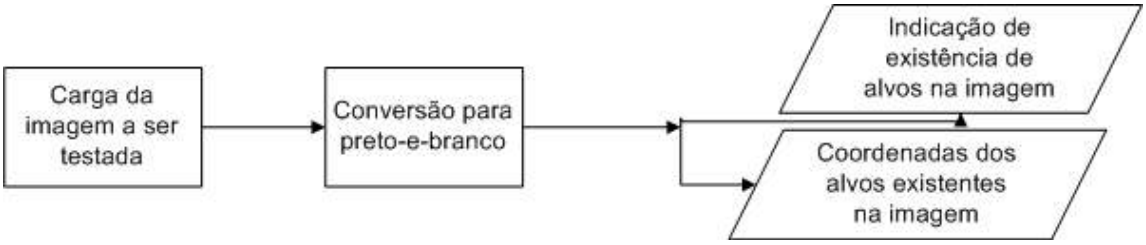

Figura 15 - Diagrama de fluxo de testes da rede neural

Inicialmente, a imagem que será testada deve ser carregada e convertida para preto e branco. Após isso, deve acontecer uma varredura em janelas de 29x29 pixels com deslocamento horizontal e vertical de 1 pixel de cada vez para encontrar os alvos e não-alvos. O resultado apresentado é uma matriz contendo as coordenadas dos pixels centrais dos alvos determinados pela rede. Estes pixels são mostrados na imagem com a cor vermelha.

Tabela 3 – Entradas e saída da rotina de testes da rede neural

acha_alvos05	
Entrada de dados	1. Imagem, formato BMP, para os testes.
Saída de dados	1. Indicação da presença ou não de alvos na imagem de teste, com os pixels centrais dos alvos pintados de vermelho. 2. Coordenadas dos pixels centrais dos alvos determinados pela rede.

3.5. Pós- Processamento

O resultado apresentado pela rotina *acha_alvos05* mostrou necessário um pós-processamento, pois a saída encontrada foi uma "nuvem de pixels" em torno de cada pixel central determinado como alvo, como ilustra a Figura 16. Para que cada alvo estivesse relacionado a somente um par de coordenadas, foi utilizada a função *[C]=subclust (X,RADII),* onde C é o centro das classes (coordenadas finais dos alvos), X é o par de coordenadas geradas na saída da rotina *acha_alvos05* e RADII é o raio da classe, que pode variar de 0 a 1 (mostrados na Tabela 4). Através da matriz de alvos gerada pela função *acha_alvos05*, a rotina *subclust* separa as coordenadas em grupos (cada grupo correspondente a um alvo distinto) e, por final, seleciona apenas um par de coordenadas de cada grupo, sendo este determinado de alvo final.

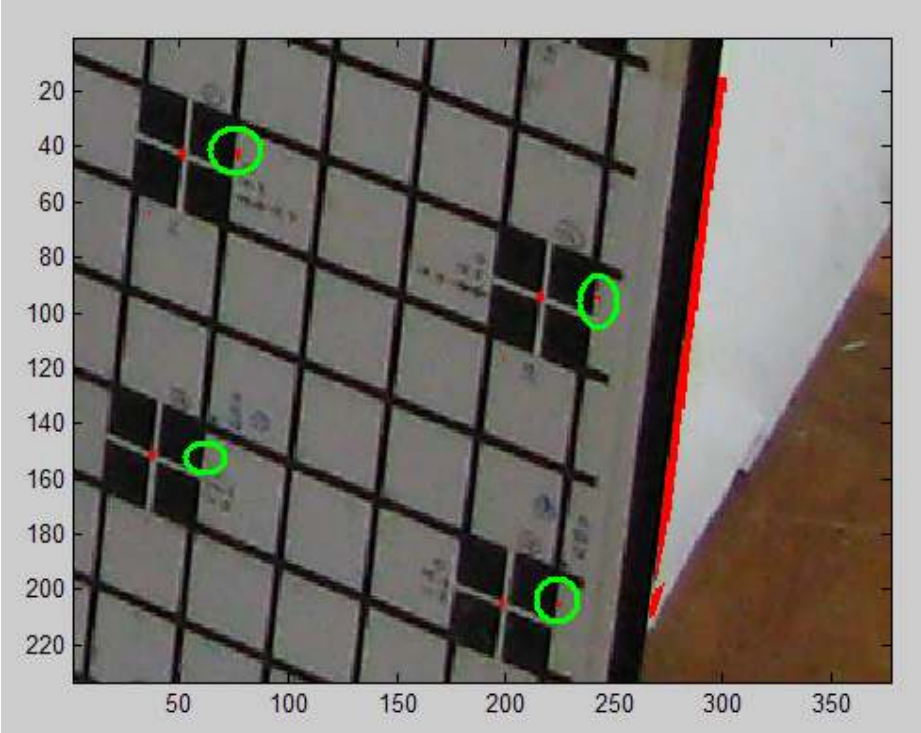

Figura 16 – Exemplo da "nuvem de pixels" gerada no grid

Pode-se perceber na Figura 16 que a rede identificou como alvos, pixels completamente diferentes dos elementos alvo treinados (facilmente observados no canto direito da Figura 18), o que foi interpretado como um mau treinamento da rede neural. Houve também a identificação de falsos alvos (circulados com um anel verde) praticamente no mesmo lugar, se comparados alvo a alvo. Diante destes problemas, para o teste final, foram acrescentados mais dados de treinamento (alvos e não alvos) na entrada da rotina *converte_imagens* e a função *subclust* foi utilizada.

Tabela 4 – Entradas e saídas da rotina de pós- processamento subclust

[C]= subclust (X, RADII)	
Entrada de dados	1. Matriz de alvos (saída da rotina *acha_alvos*) identificada como X 2. Raio da classe (variável de 0 a 1) declarada como 0,2.
Saída de dados	3. Matriz C, com o par de coordenadas de cada pixel identificado como alvo de cada subgrupo.

O valor de RADII igual a 0,2 foi empírico, pois após serem feitos diversos testes, notou-se um melhor resultado na escolha dos centros das classes para este valor.

Capítulo 4

Testes e Resultados

4.1. Redes Treinadas e Resultados

Os testes se destinaram a utilizar treinamentos distintos da rede neural, chamados de L1, L2, L3 e L. Cada vez que a rotina *treinarede* é processada, ela gera um L diferente, devido ao comando "randn('state', sum(100*clock)) ", pois cada rede é treinada a partir de pesos cujos valores iniciais são aleatórios e variam a cada nova inicialização, utilizando uma semente randômica, gerada computacionalmente pela função randn( ). Analisando o gráfico de erro de cada L, juntamente com os testes aplicados a um conjunto de imagens, define-se o L mais eficaz. Os primeiros testes apresentavam erros baixos, porém, a análise das figuras obtidas indicava que não se tratava de um bom treinamento a ponto de ser funcional para qualquer imagem. A fim de corrigir este problema, foram acrescentados mais alvos e não-alvos às entradas da rotina *converte_imagens,* sendo que estes novos elementos foram retirados de imagens de treinamento e não de imagens de teste. Estes alvos e não-alvos foram cuidadosamente escolhidos, de modo que pudesse esclarecer possíveis dúvidas da rede no momento do seu treinamento. Desta maneira, foi obtido o L final, que apresentou erro quadrático instantâneo abaixo de 0,1% e resultados satisfatórios para qualquer imagem de *grid* testado.

4.1.1. Primeiro Teste – Rede Neural Treinada L1

No teste em que foi utilizada a rede neural treinada L1, aparentemente, percebe-se que o erro diminui bastante, quase tendendo a zero no decorrer do tempo (Figura 17). Porém, de acordo com os testes realizados nas figuras 18, 19, 20, 21, 22, 23 e 24 foi constatado que isso não significou o bom treinamento da rede. Nota-se uma

grande quantidade de pixels em vermelho, marcados como alvo, em regiões bastante distintas dos verdadeiros alvos.

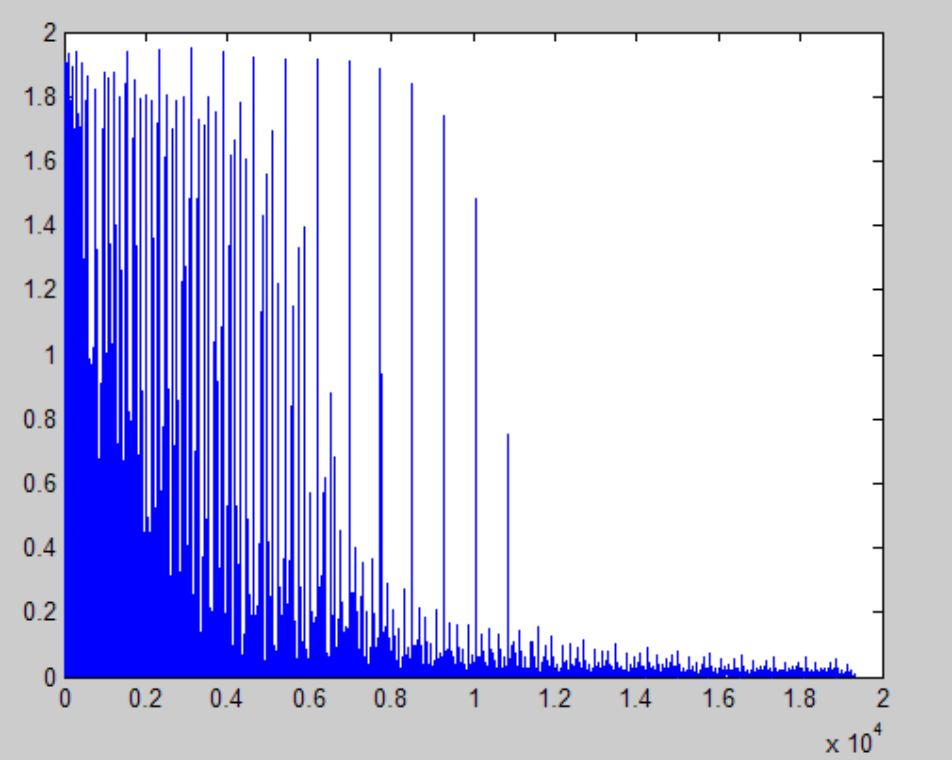

Figura 17 – Erro da rede L1 ao longo do tempo de treinamento.

Esta rede L1, com o gráfico de erro mostrado acima, resultou nos gráficos de teste mostrados nas Figuras 18 a 21.

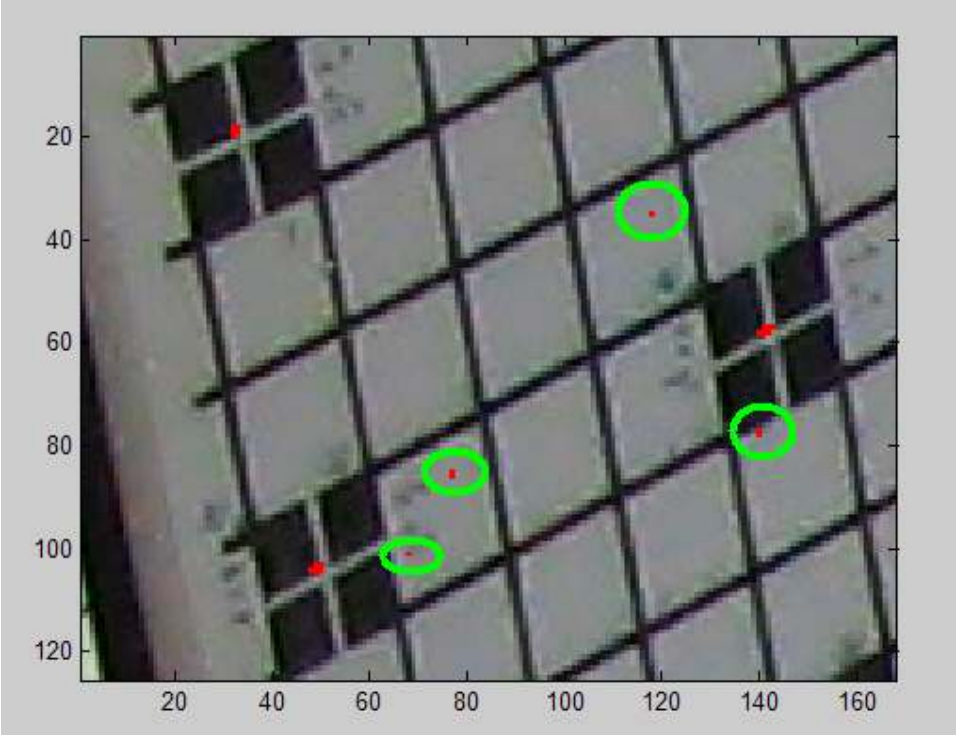

Figura 18 – Exemplo 1 dos alvos encontrados após o treinamento da rede L1

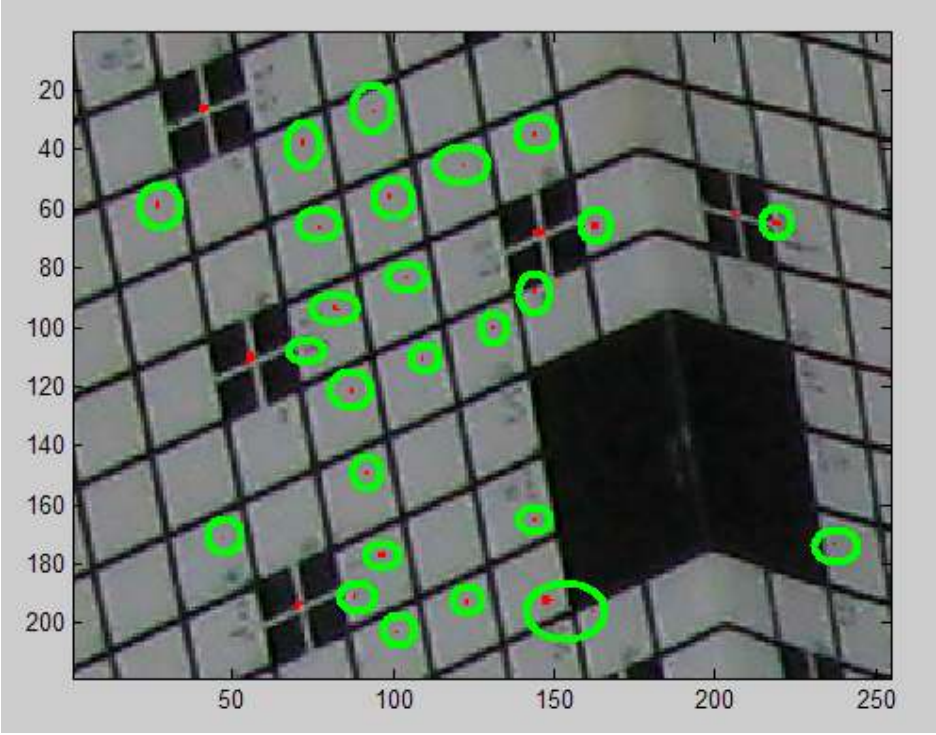

Figura 19 - Exemplo 2 dos alvos encontrados após o treinamento da rede L1

Nas figuras 18 e 19 acima, pode-se notar, circulados em verde, os pixels que foram erroneamente identificados como alvo. Quanto maior o número de círculos verdes, mais erros ocorreram no treinamento da rede. Houve também o acerto de todos os alvos reais.

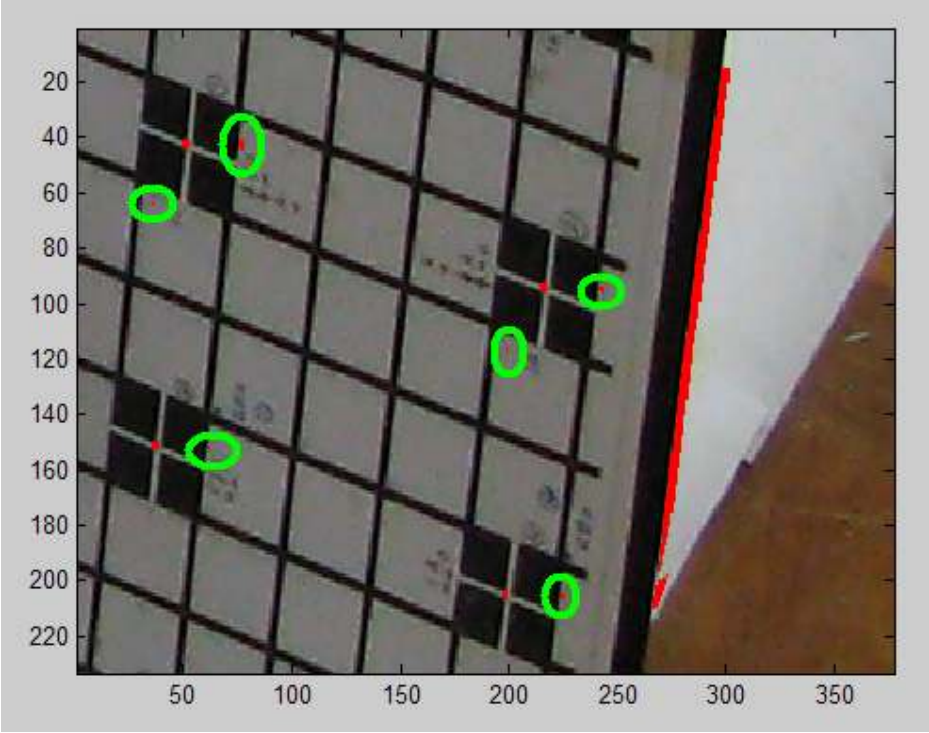

Figura 20 - Exemplo 3 dos alvos encontrados após o treinamento da rede L1

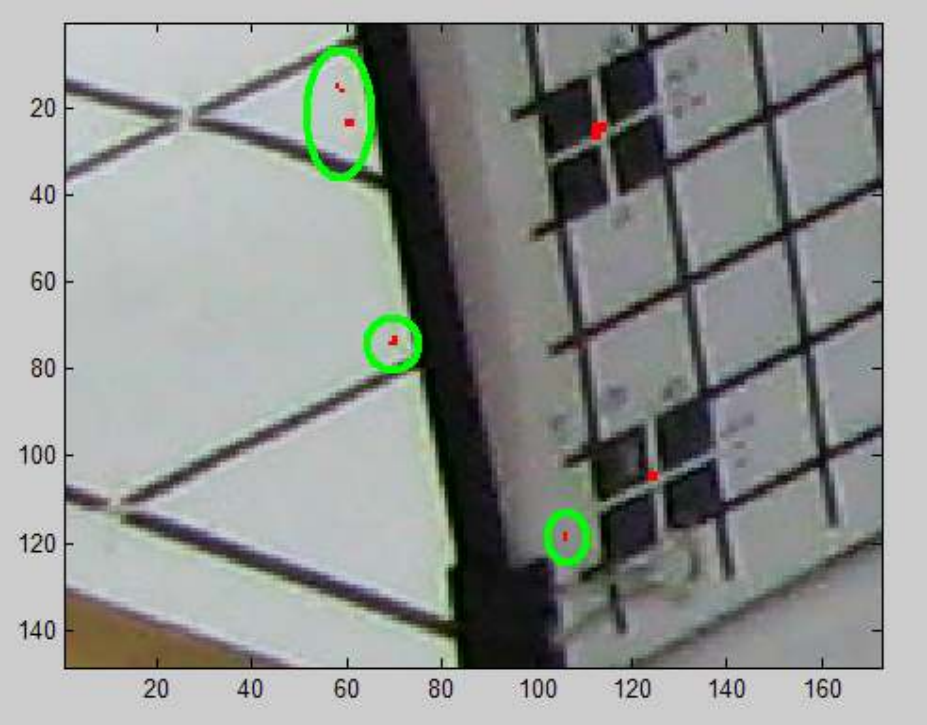

Figura 21 - Exemplo 4 dos alvos encontrados após o treinamento da rede L1

Nas figuras 20 e 21 há uma semelhança em relação aos pixels que foram pintados de vermelho (identificados como alvos). Nota-se uma grande concentração destes pixels nas regiões de extremidade dos grids, onde há a troca do preto para o branco. Além disto, poucos foram os alvos falsos.

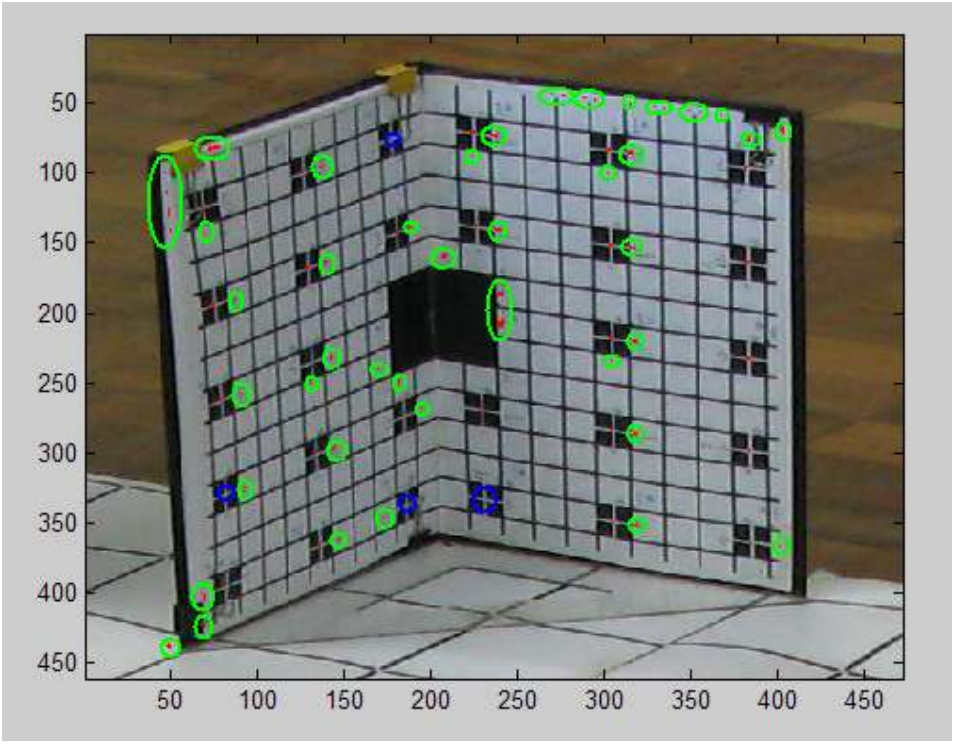

Figura 22 - Exemplo 5 dos alvos encontrados após o treinamento da rede L1

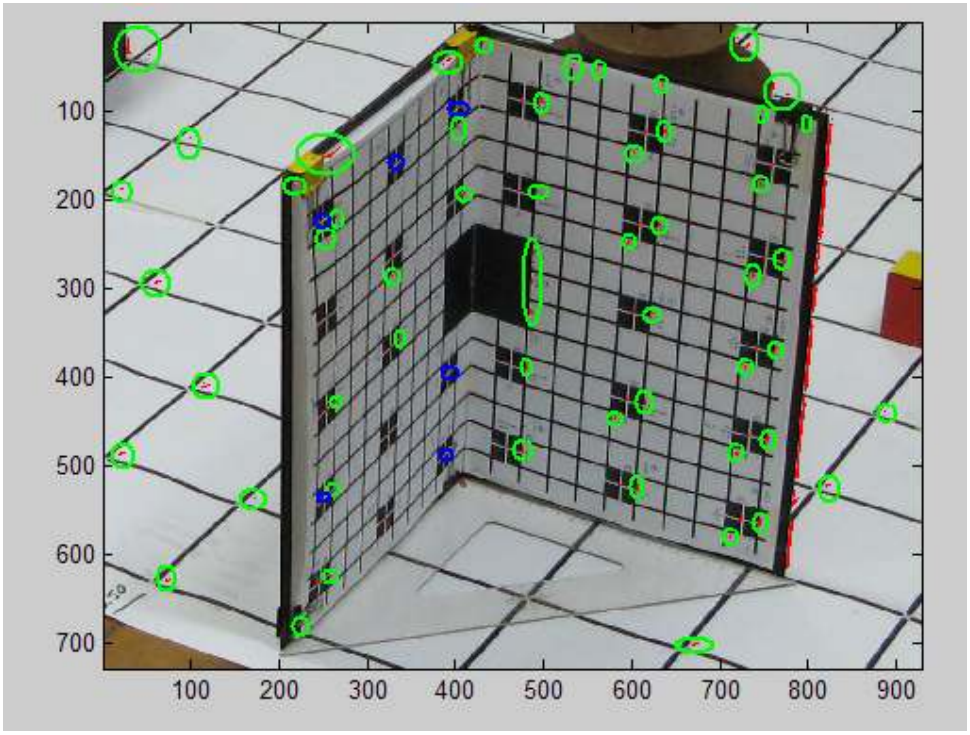

Figura 23 - Exemplo 6 dos alvos encontrados após o treinamento da rede L1

As figuras 22, 23 e 24 contêm todo o grid na imagem, portanto, para um maior número de alvos, houve também mais concentração deles, erroneamente encontrados (circulados em verde). Como pode ser observado, os círculos azuis representam lugares do grid onde o alvo deveria ser reconhecido e não foi. Na parte esquerda do grid da figura 24, portanto, não foi reconhecido nenhum alvo, enquanto que, na parte direita, todos foram encontrados. Para esta rede específica, os alvos significativamente inclinados não foram bem treinados.

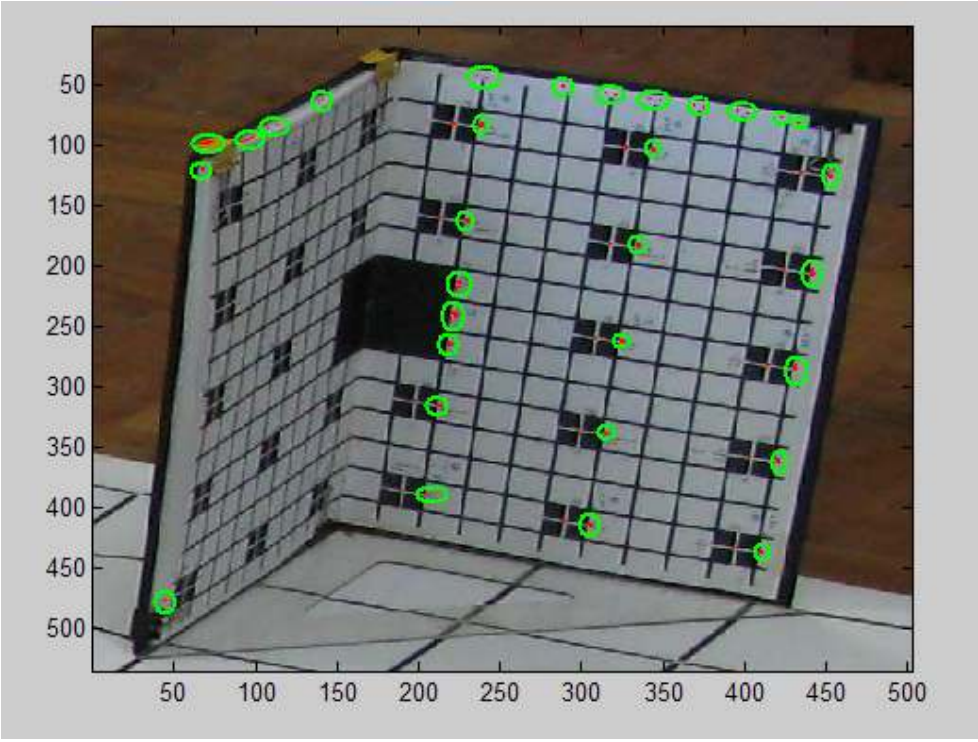

Figura 24 - Exemplo 7 dos alvos encontrados após o treinamento da rede L1

4.1.2. Segundo Teste – Rede neural treinada L2

Para este novo teste, a rede foi processada novamente e foi gerado o L2. Aparentemente, também se percebe que o erro tende à zero no decorrer do tempo (Figura 25), porém, o resultado ainda não foi considerado aceitável, pois apesar de melhores resultados em relação à rede L1 (ver através das Figuras 26 a 32), os testes da rede L2 ainda apresentam diversos alvos bem distintos dos alvos treinados.

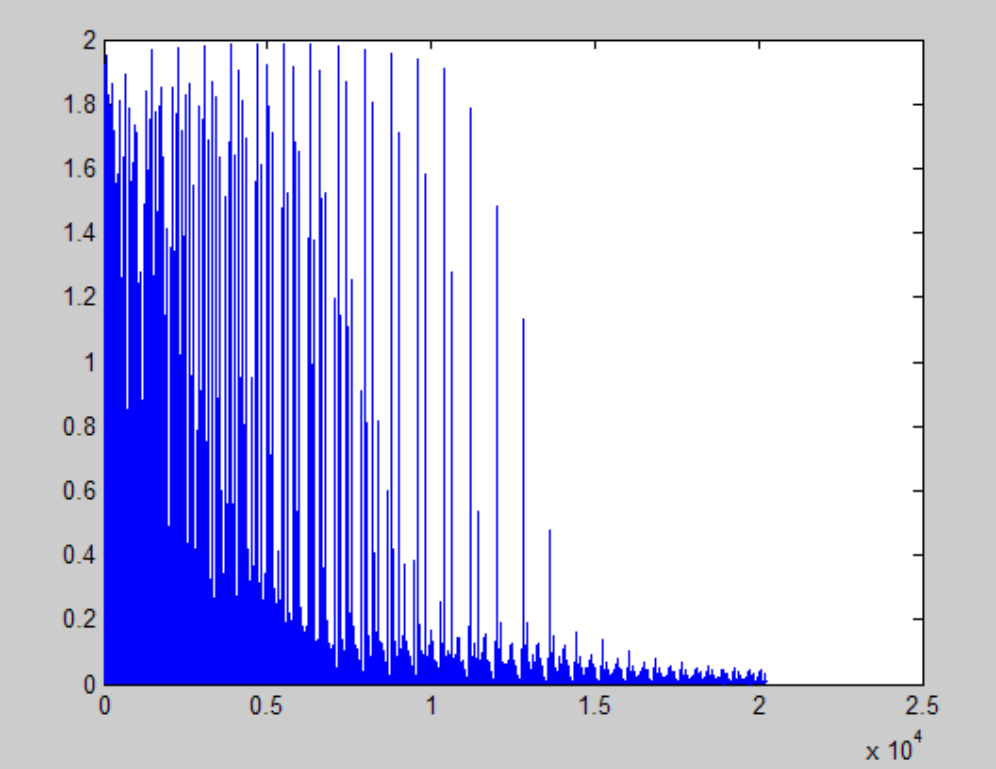

Figura 25- Erro da rede L2 ao longo do tempo de treinamento

A nova rede obtida apresentou os gráficos de teste mostrados nas Figuras 26 a 32.

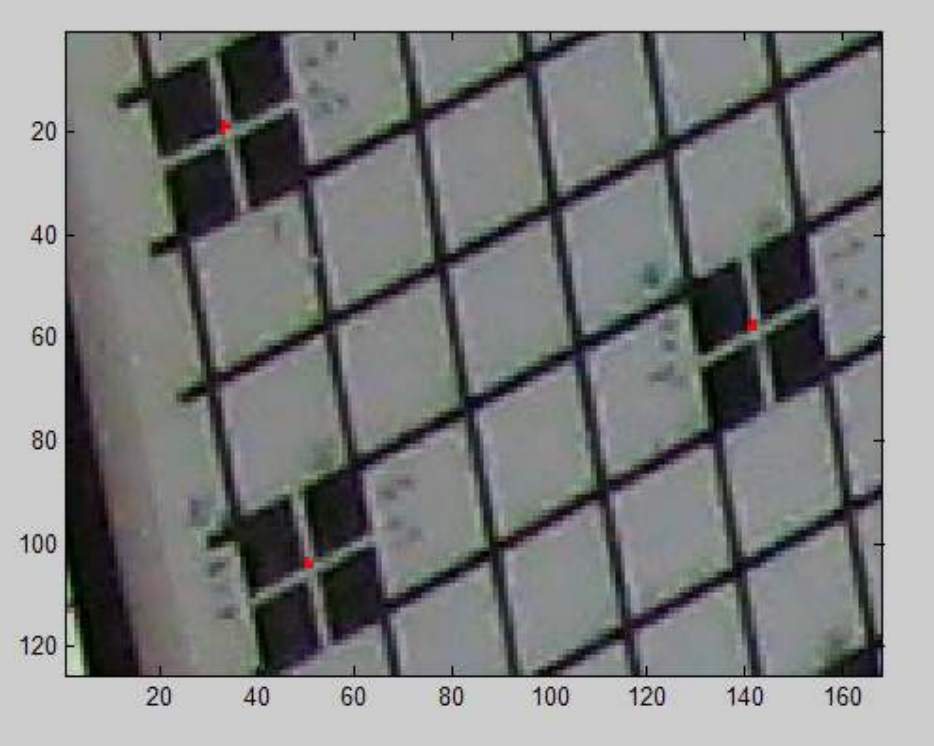

Figura 26 - Exemplo 1 dos alvos encontrados após o treinamento da rede L2

A Figura 26 apresentou um bom resultado, onde nenhum alvo falso foi identificado e todos os alvos reais foram encontrados.

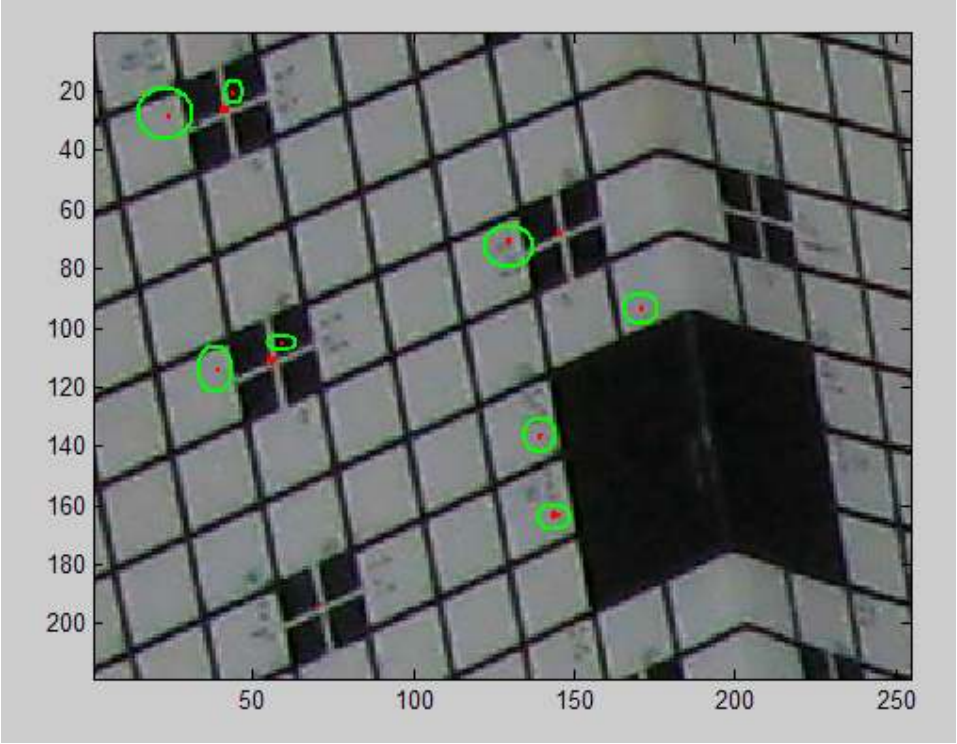

Figura 27 - Exemplo 2 dos alvos encontrados após o treinamento da rede L2

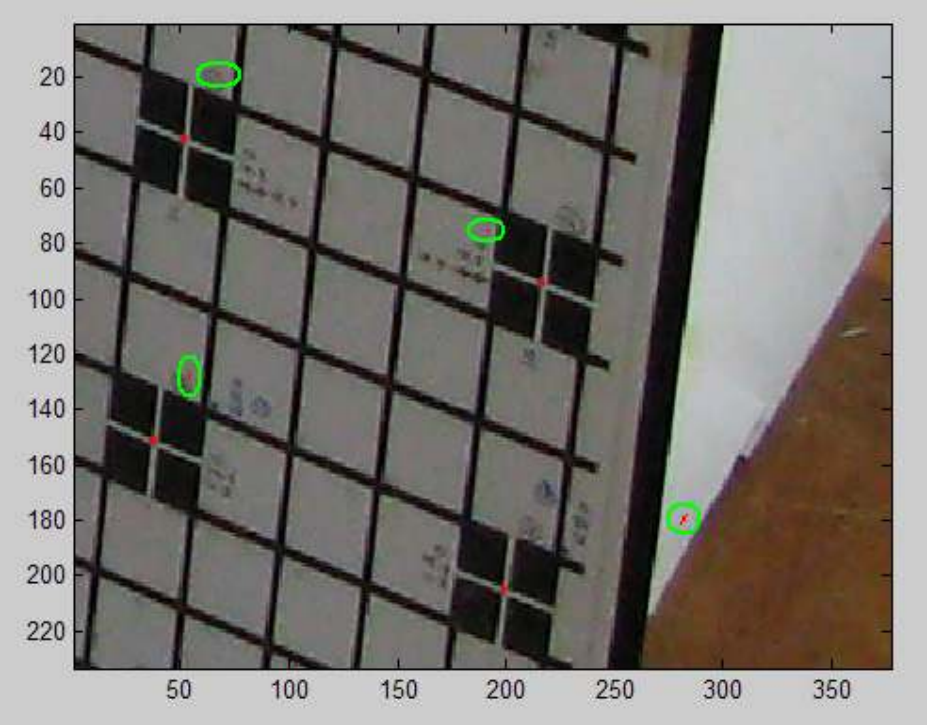

Figura 28 - Exemplo 3 dos alvos encontrados após o treinamento da rede L2

As Figuras 27 e 28 apresentam menor número de alvos falsos identificados, se comparados com as Figuras 19 e 20 da rede anterior (as figuras são idênticas). Já a Figura 29 apresentou resultado menos satisfatório em relação à Figura 21, o que dificultou a conclusão sobre qual seria a melhor rede.

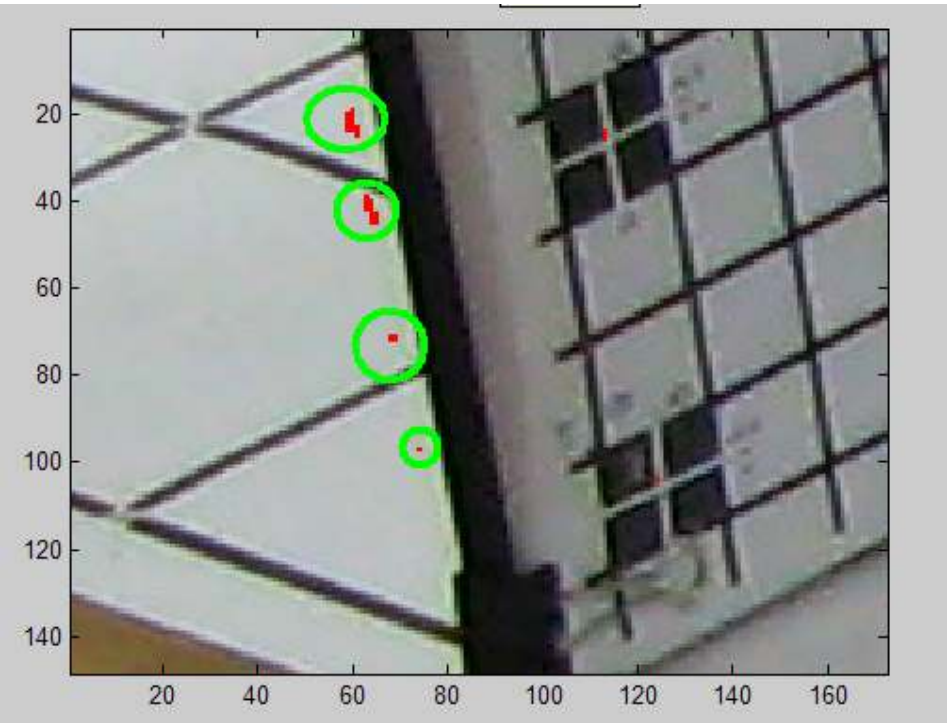

Figura 29 - Exemplo 4 dos alvos encontrados após o treinamento da rede L2

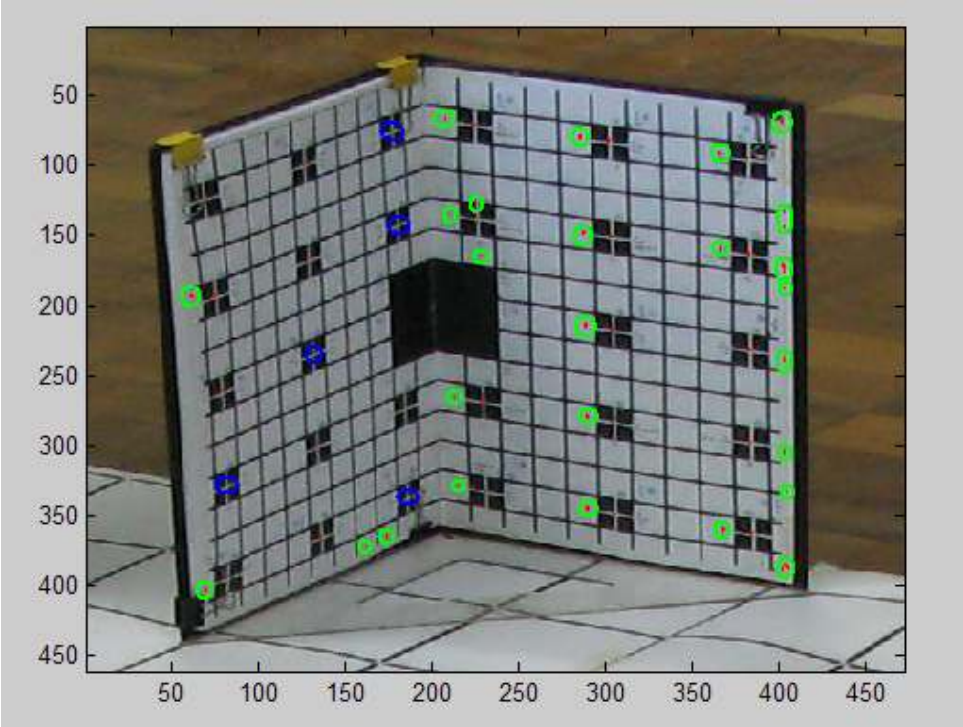

Figura 30 - Exemplo 5 dos alvos encontrados após o treinamento da rede L2

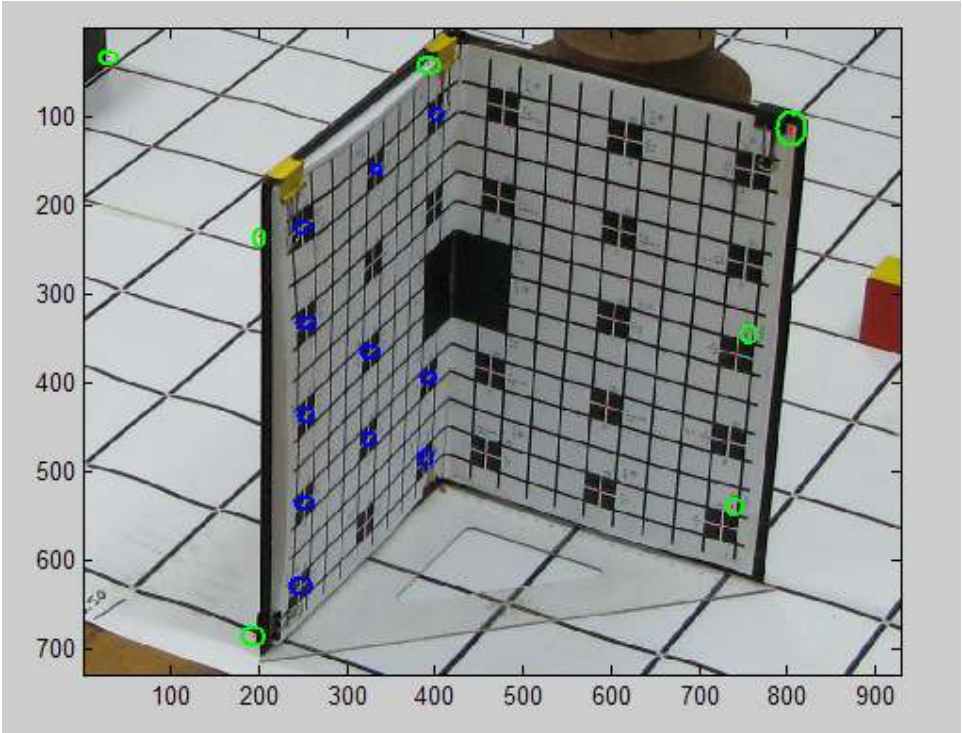

Figura 31 - Exemplo 6 dos alvos encontrados após o treinamento da rede L2

Assim como a rede L1 anterior, a rede L2 mostrou que, para imagens com o grid inteiro contigo na imagem (Figuras 30, 31 e 32), conforme se aumenta a inclinação dos alvos, mais dificilmente eles são identificados (alvos circulados de azul). Na Figura 32, nenhum alvo foi encontrado na parte esquerda do *grid*.

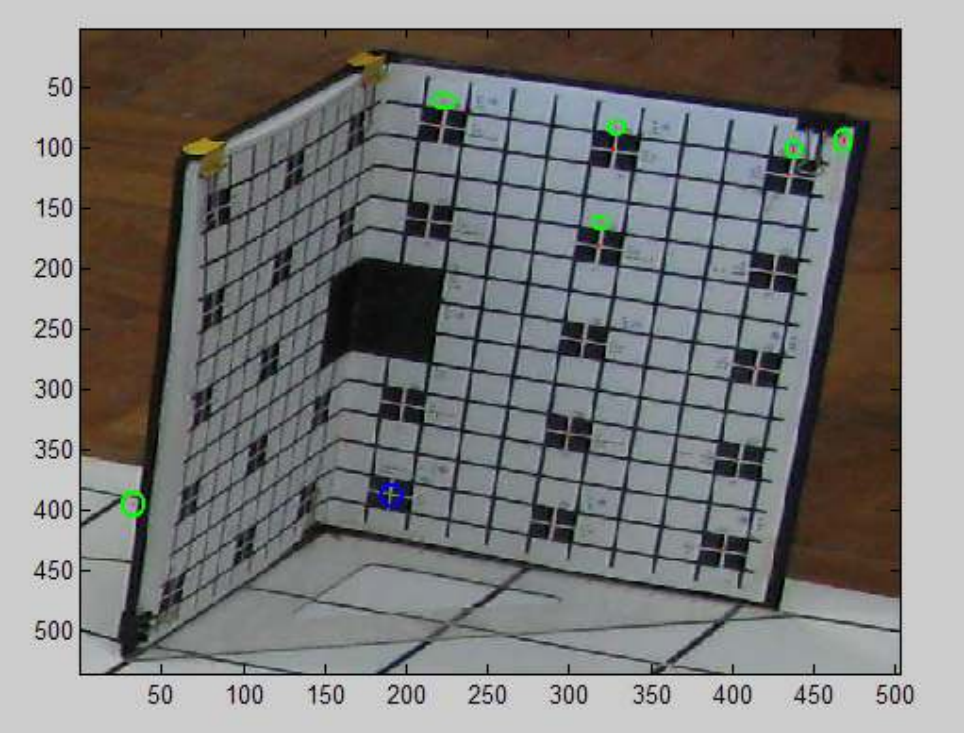

Figura 32 - Exemplo 7 dos alvos encontrados após o treinamento da rede L2

4.1.3. Terceiro Teste – Rede Neural Treinada L3

A rede neural treinada L3 é mais um exemplo de que nem sempre um erro tendendo a zero (Figura 33) significa o bom treinamento da rede. Ela, possivelmente, pode aprender errado (devido aos dados iniciais apresentados para este respectivo treinamento), porém, se for entendido que houve o aprendizado por parte da rede, esta apresentará o erro considerado baixo. O artifício utilizado para confirmar se a rede é funcional caracteriza-se por testes em diversos *grids,* como mostram as Figuras 34 a 40.

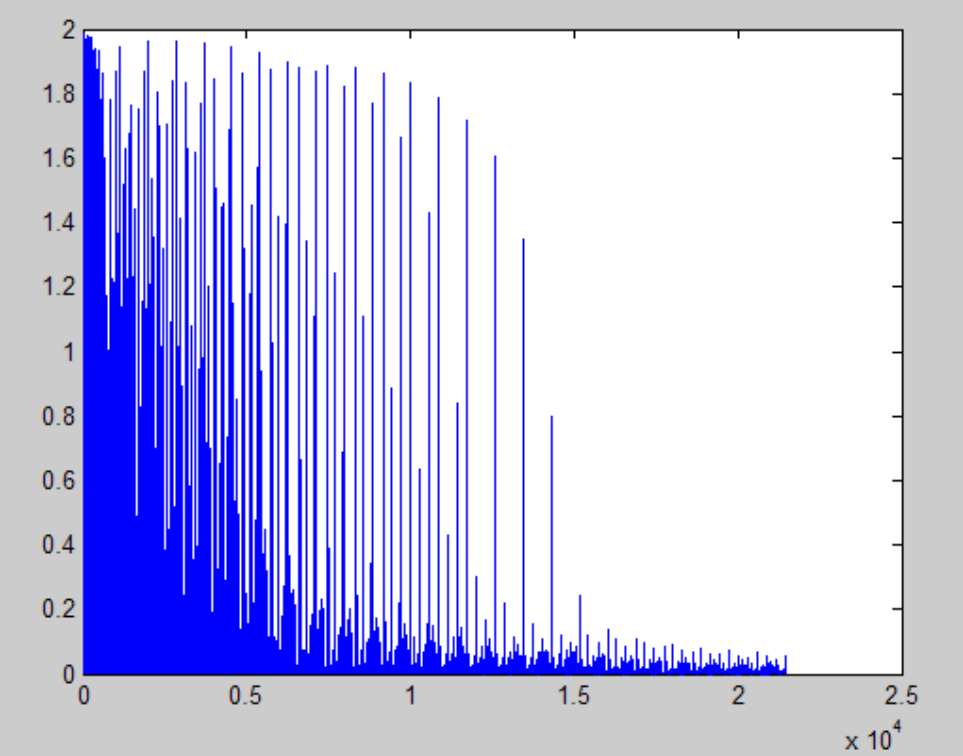

Figura 33 - Erro da rede L3 ao longo do tempo de treinamento

Na Figura 34 observa-se que um alvo não foi identificado, o mesmo que foi encontrado em todas as redes anteriormente testadas. O mesmo ocorre na figura 37.

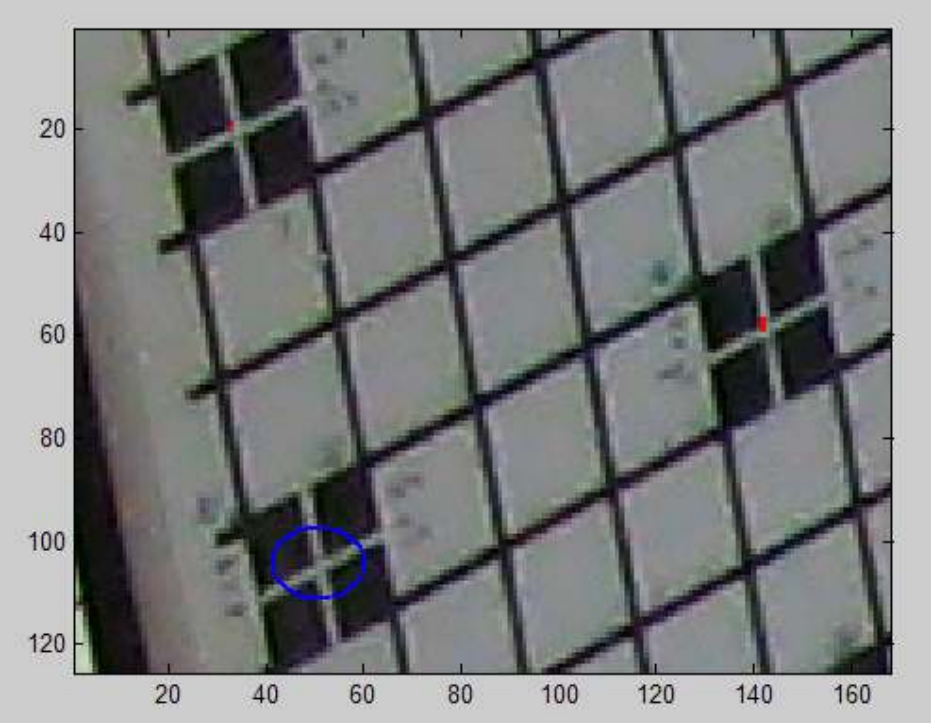

Figura 34 - Exemplo 1 dos alvos encontrados após o treinamento da rede L3

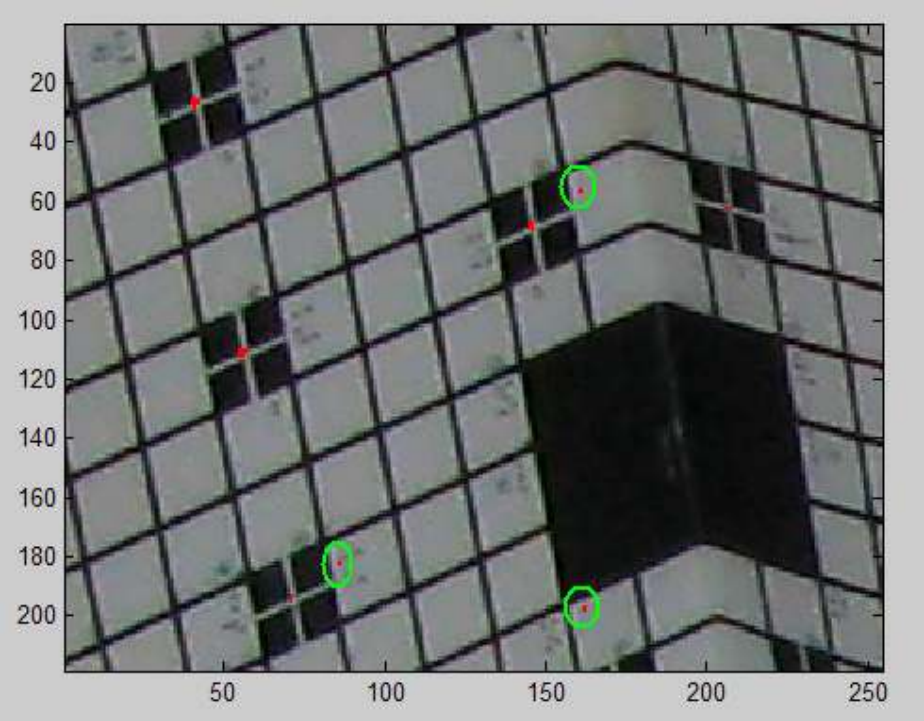

Figura 35 - Exemplo 2 dos alvos encontrados após o treinamento da rede L3

Poucos alvos falsos foram identificados na Figura 35 e nenhum destes foi encontrado na Figura 36. Nota-se, também, que em todas as imagens os alvos reais não contêm apenas um pixel, e sim, uma "nuvem de pixels", daí a importância do pós-processamento.

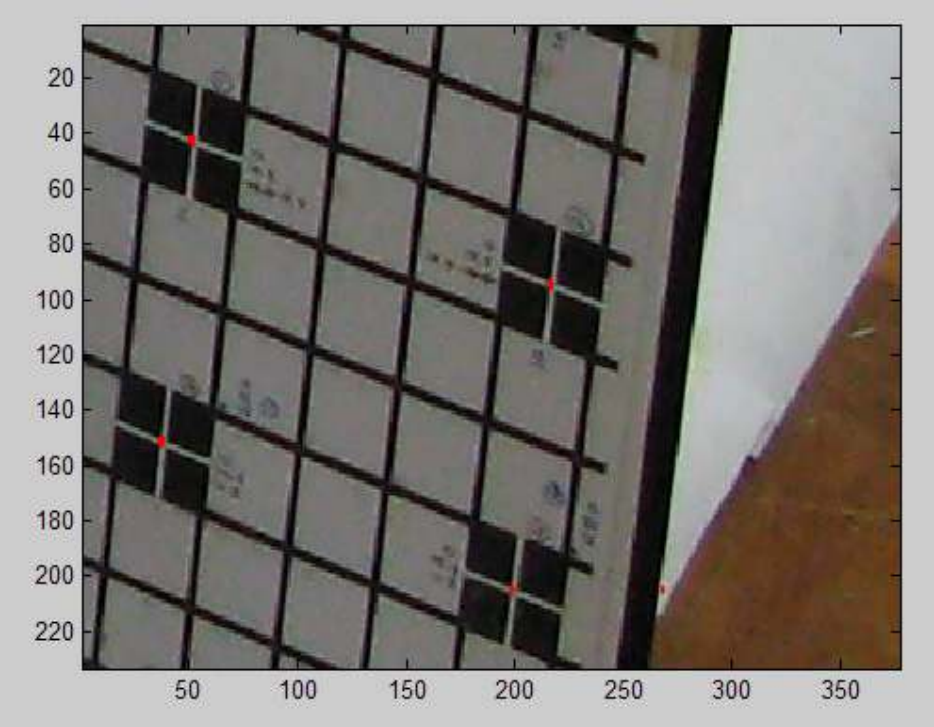

Figura 36 - Exemplo 3 dos alvos encontrados após o treinamento da rede L3

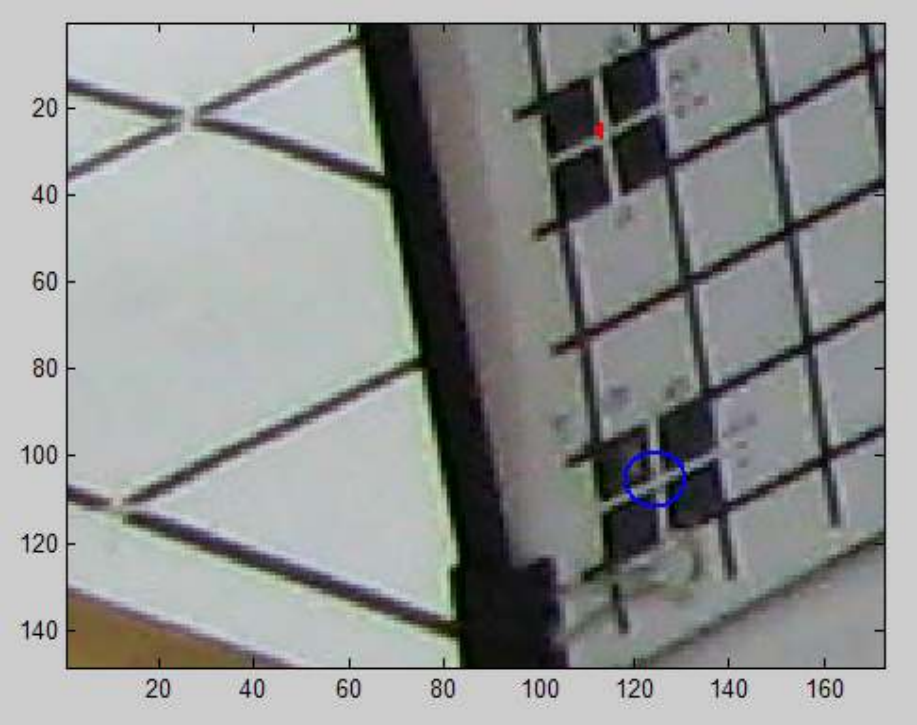

Figura 37 - Exemplo 4 dos alvos encontrados após o treinamento da rede L3

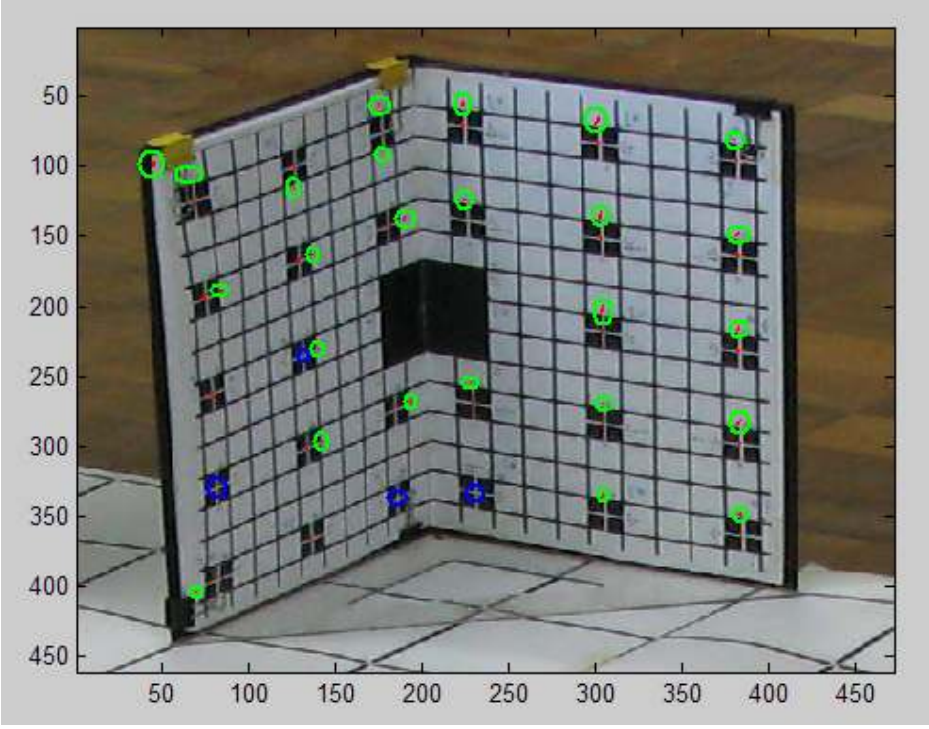

Figura 38 - Exemplo 5 dos alvos encontrados após o treinamento da rede L3

Nas Figuras 38 e 39, houve muitos alvos erroneamente encontrados (circulados de verde). Porém, na Figura 38, percebe-se uma relação entre estes alvos, pois se encontram sempre no mesmo lugar (analisando cada quadrado preto do *grid*). Já na Figura 39, muitos alvos foram identificados fora do *grid*.

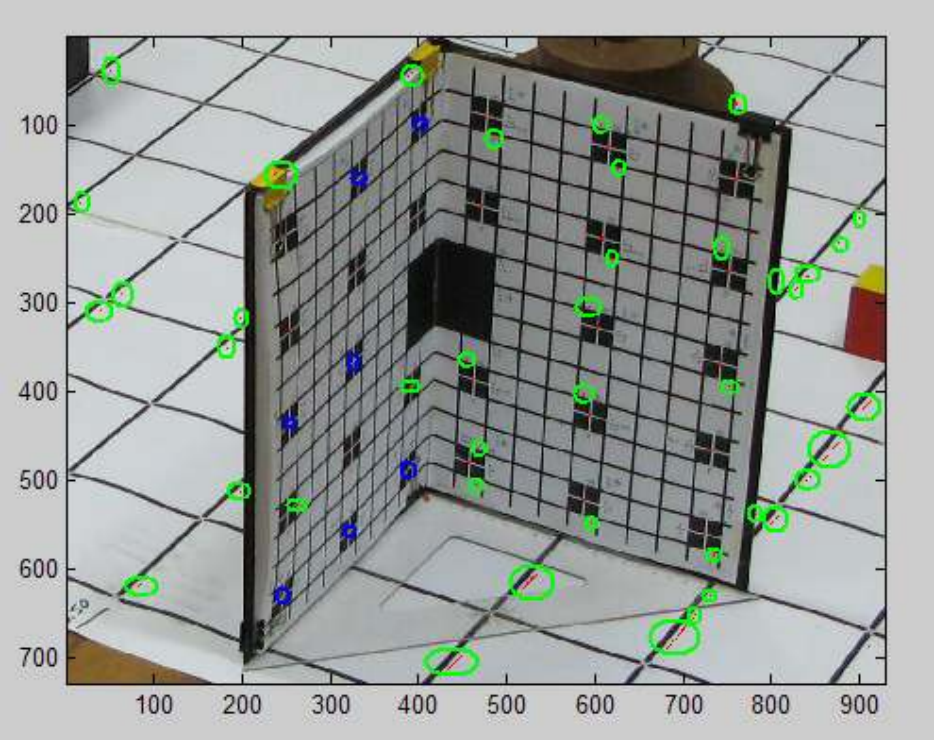

Figura 39 - Exemplo 6 dos alvos encontrados após o treinamento da rede L3

A Figura 40 mostra que poucos foram os alvos falsos encontrados, porém, nenhum alvo real foi encontrado na parte esquerda do *grid*, enquanto todos foram encontrados na parte direita.

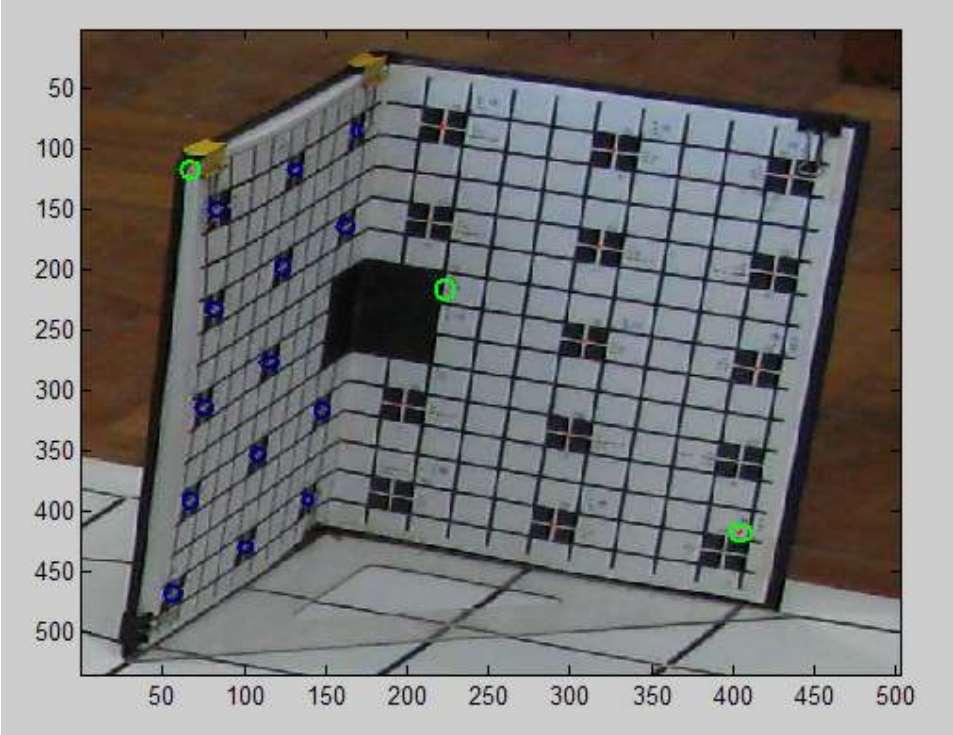

Figura 40 - Exemplo 7 dos alvos encontrados após o treinamento da rede L3

4.2. Avaliação do Desempenho da Rede Ótima

A fim de otimizar os resultados anteriormente obtidos, foram acrescentados novos dados de entrada na rotina *converte_imagens*. Estes novos dados foram não-alvos escolhidos de acordo com a análise dos erros cometidos nos testes das redes anteriores e alguns exemplos são mostrados na Figura 41.

Figura 41 – Exemplos de não-alvos adicionados para treinamento da rede ótima

Novas matrizes de alvos e não-alvos treinaram a rede neural e geraram o L com erro indicado na Figura 42. Este foi o melhor L obtido após dezenas de tentativas de treinamento e essa eficácia pode ser comprovada com as Figuras 43 a 49 que seguem abaixo. Alguns pixels marcados em vermelho, que foram erroneamente identificados, são erros aceitáveis desde que não sejam muito populosos, uma vez que o computador não possui raciocínio próprio, e a diferença entre os alvos e não alvos são apresentados através de algumas imagens apenas.

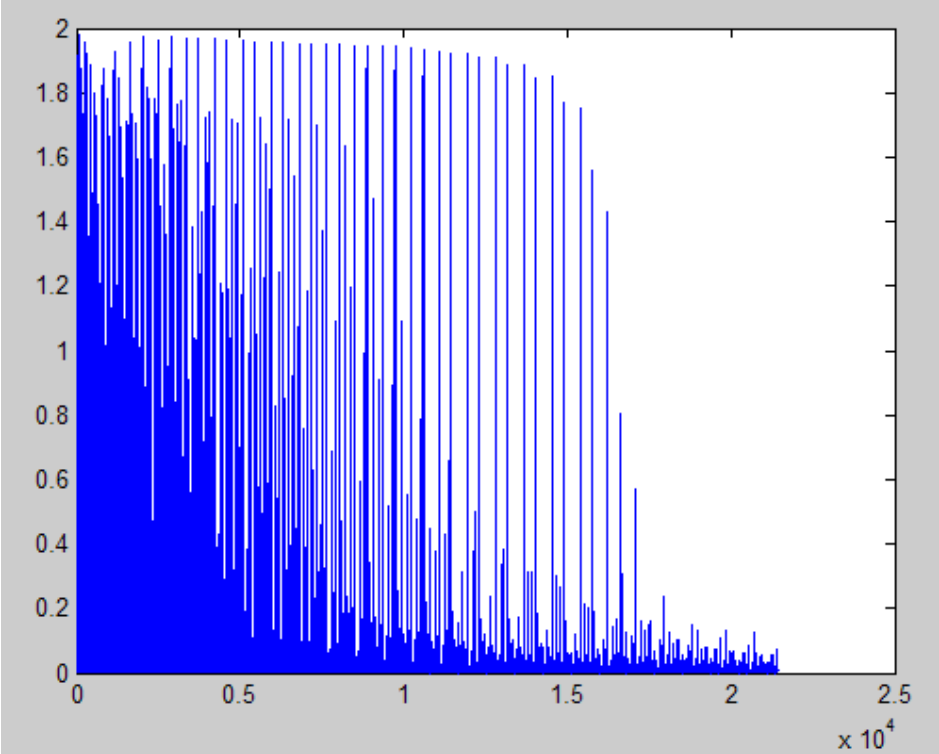

Figura 42 - Erro da rede final L ao longo do tempo de treinamento

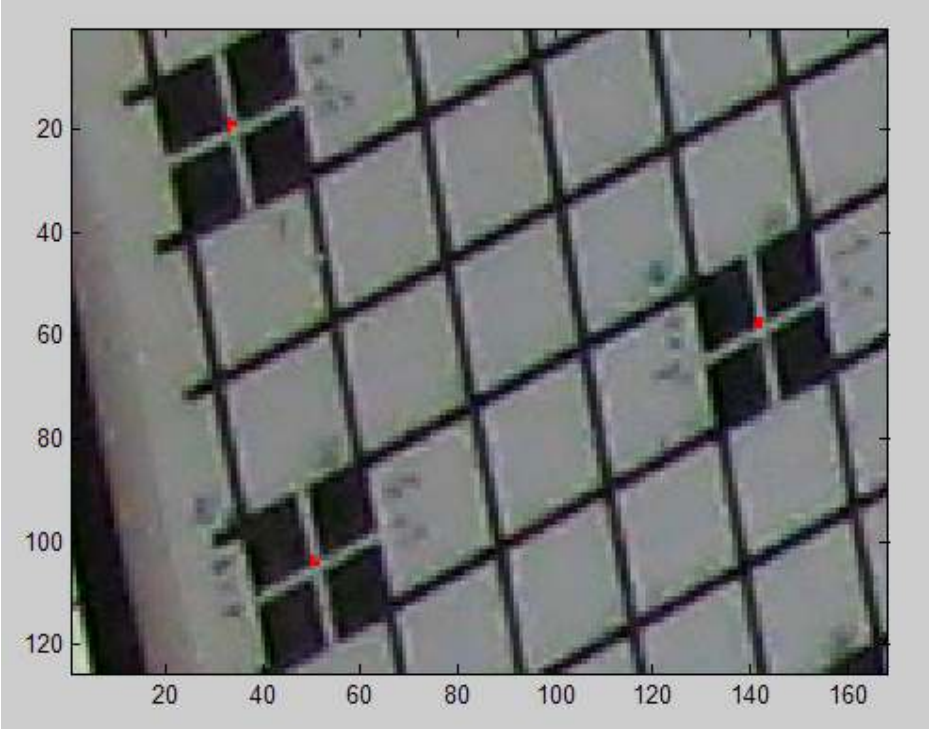

Figura 43 - Exemplo 1 dos alvos encontrados após o treinamento da rede L

Nas Figuras 43, 45 e 46 nenhum alvo falso foi identificado, sendo necessário somente o pós-processamento para determinar o pixel central de cada “nuvem de pixels” encontrada.

A Figura 44 também apresentou resultado satisfatório, uma vez que todos os alvos reais foram encontrados e apenas dois grupos de falsos alvos ocuparam o *grid.*

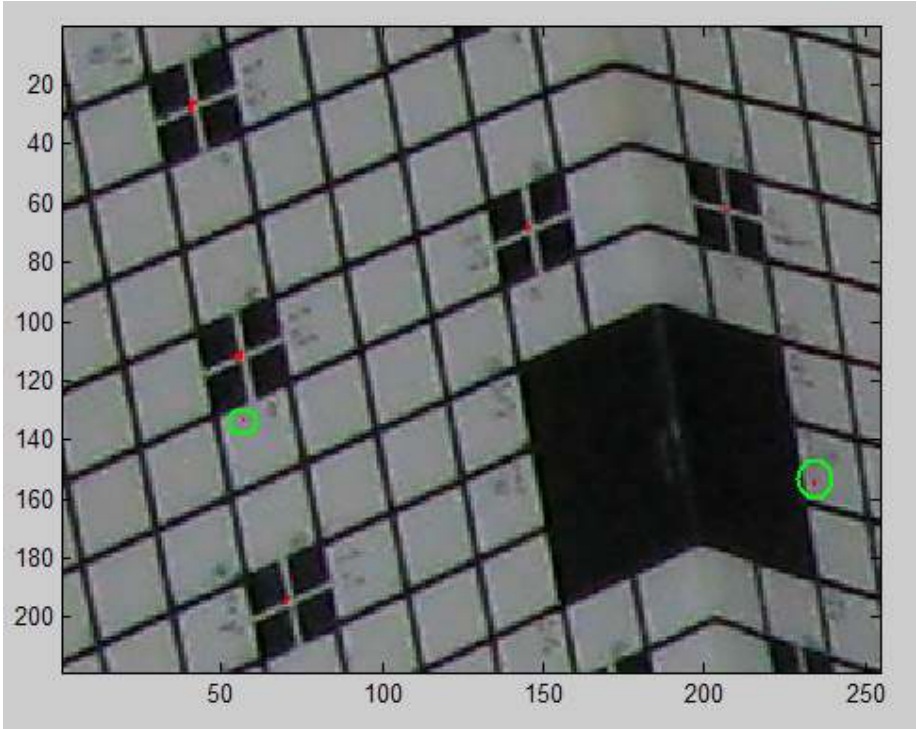

Figura 44 - Exemplo 2 dos alvos encontrados após o treinamento da rede L

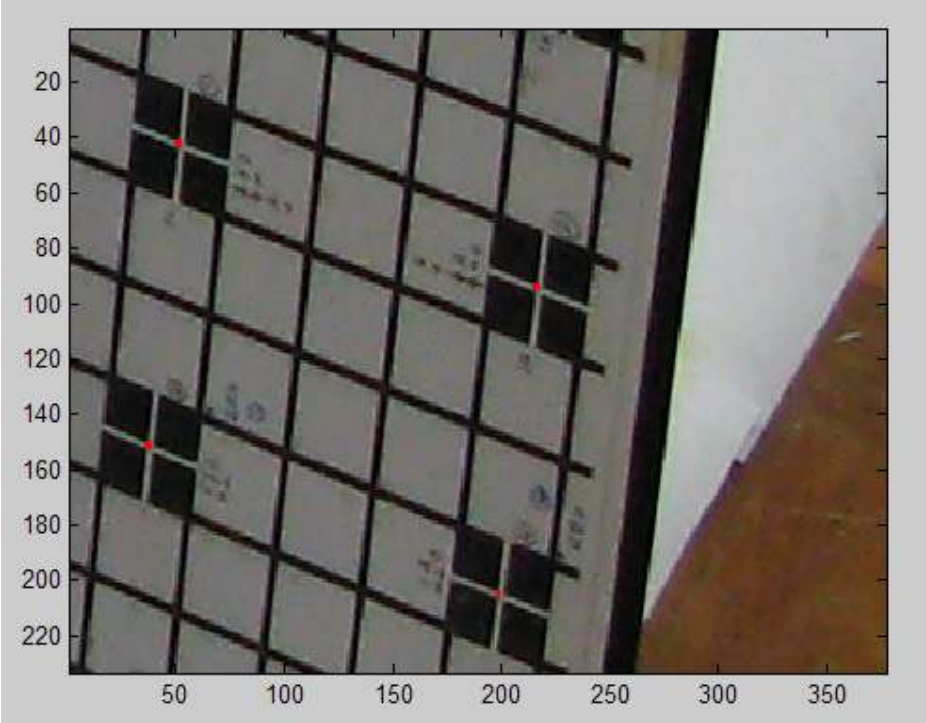

Figura 45 - Exemplo 3 dos alvos encontrados após o treinamento da rede L

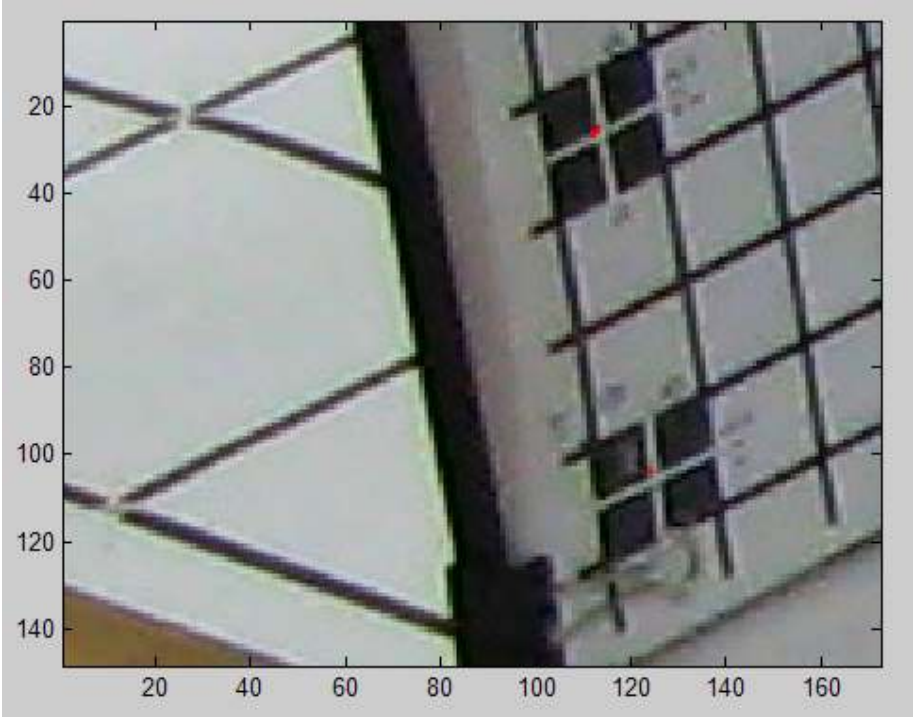

Figura 46 - Exemplo 4 dos alvos encontrados após o treinamento da rede L

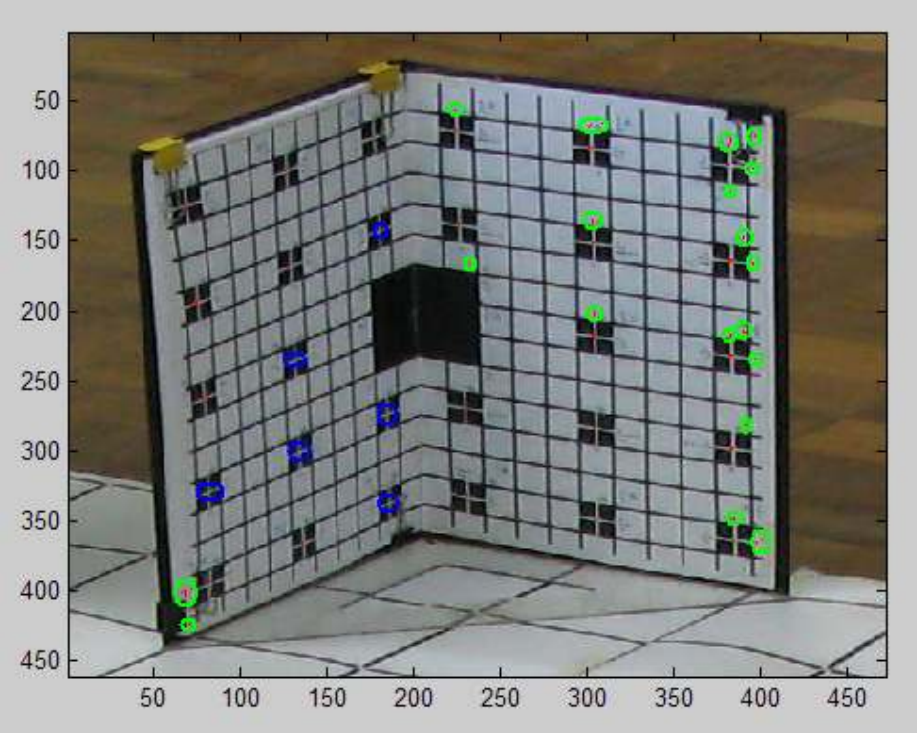

Figura 47 - Exemplo 5 dos alvos encontrados após o treinamento da rede L

As Figuras 47 e 48 também não identificaram todos os alvos da parte esquerda do *grid* (circulados de azul), mesmo após o acréscimo de dados de entrada para o treinamento desta rede. Porém, poucos alvos falsos (circulados de verde) foram identificados.

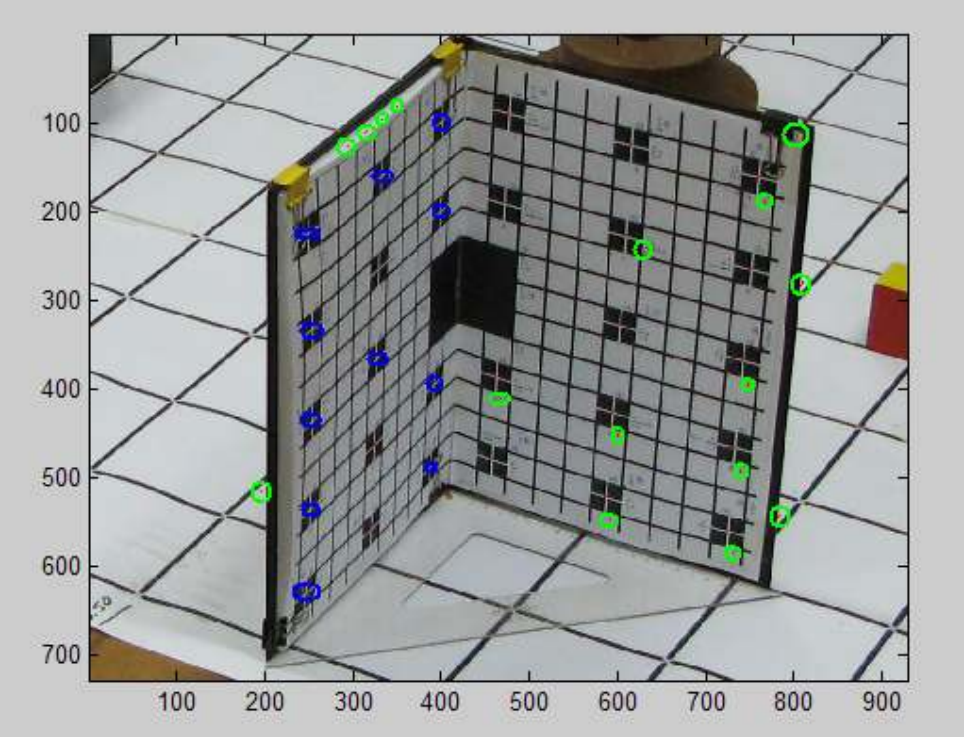

Figura 48 - Exemplo 6 dos alvos encontrados após o treinamento da rede L

Similar à Figura 48, a Figura 49 apresenta poucos alvos falsos identificados, porém, não identificou alvos na parte esquerda do *grid,* mostrando que para alvos inclinados, a rede não foi eficiente.

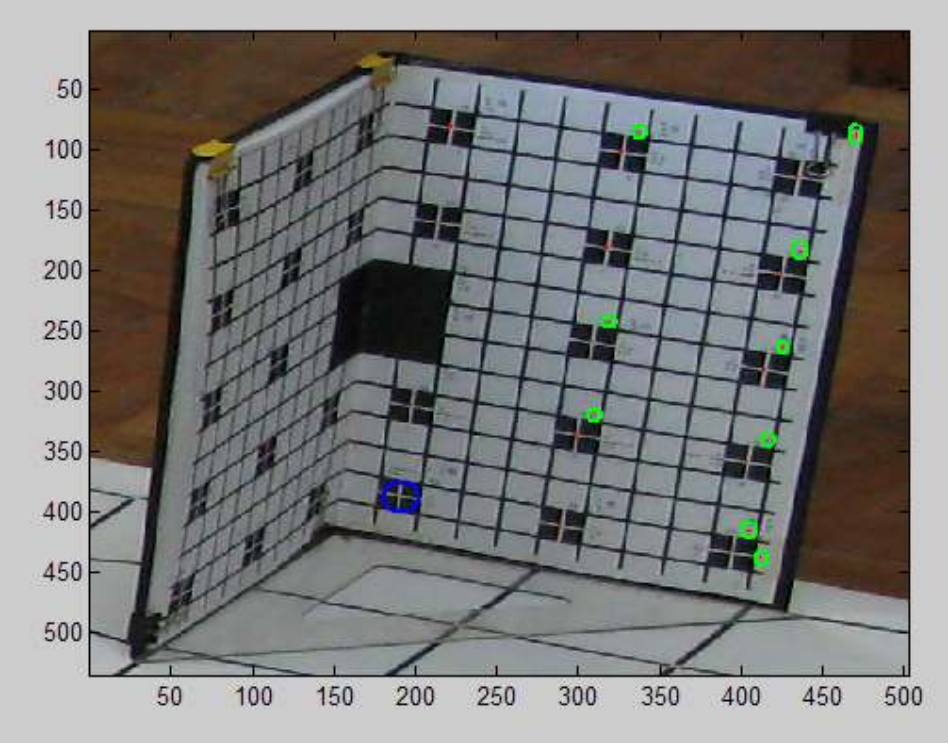

Figura 49 - Exemplo 7 dos alvos encontrados após o treinamento da rede L

Analisando todas as Figuras, esta rede ótima apresentou melhores resultados após a inserção de dados de entrada para seu treinamento. Notou-se, também, que um erro quadrático alto implica em um mau treinamento da rede, porém, um erro quadrático baixo nem sempre significa um bom treinamento da mesma.

Após a finalização do processamento (onde a rotina *acha_alvos05* exibe os gráficos com alvos pintados de vermelho), foi necessário utilizar a função do Matlab "[C]=subclust(X,RADII)" seguida da rotina *mostra_C,* para que seja exibido somente um pixel pintado de cada nuvem de alvos anteriormente identificada. Logo, os alvos foram identificados com um pixel pintado de verde nas imagens (Figuras 50, 51 e 52) do *grid* e suas coordenadas foram explicitadas na tela do Matlab e mostradas nas Tabelas 5, 6 e 7.

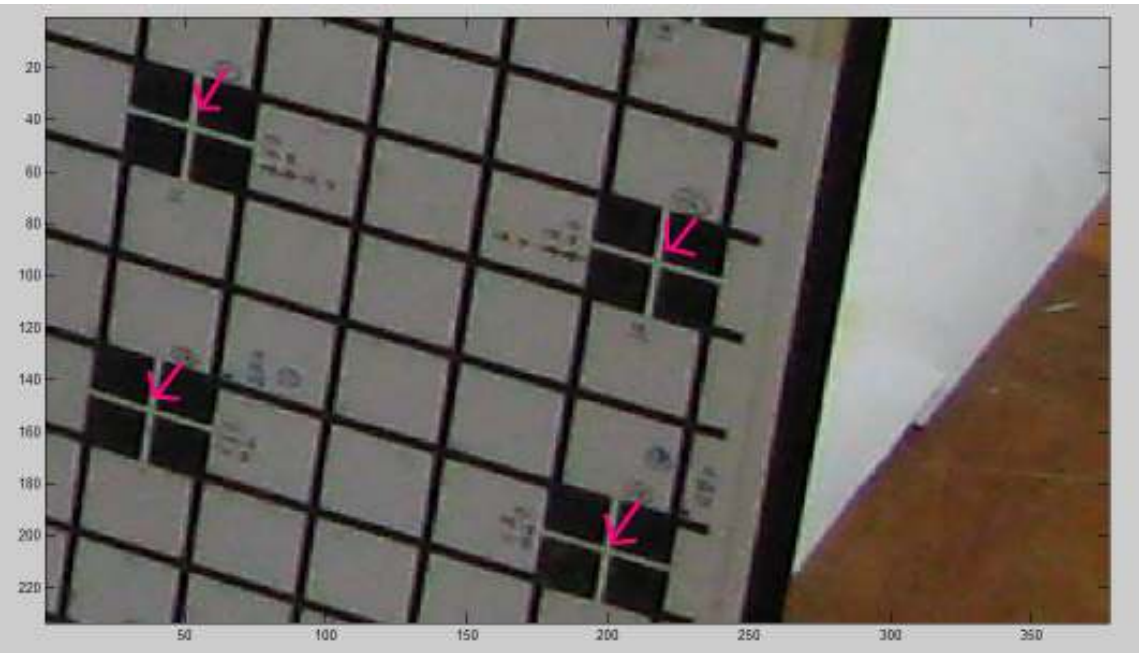

Figura 50 – Exemplo 1 do grid após a clusterização com seus alvos marcados na cor verde

A Figura 50 apresenta apenas 1 pixel para cada alvo encontrado, pois a rotina *subclust* os selecionou de cada agrupamento. Estes 4 pixels resultantes de todo o processamento têm suas coordenadas explicitadas na Tabela 5.

Tabela 5 – Coordenadas dos pixels verdes da Figura 51, identificados como alvos

Coordenada X	**Coordenada Y**
216	94
37	151
51	42
198	205

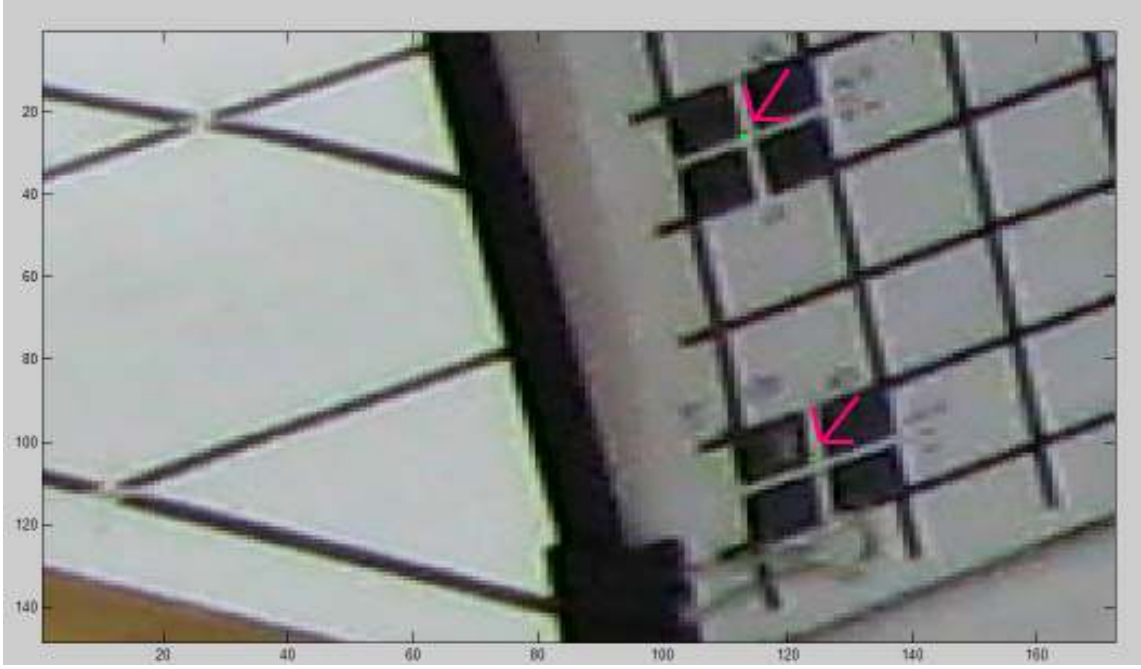

Figura 51 – Exemplo 2 do grid após a clusterização com seus alvos marcados na cor verde

A Figura 51 apresenta 2 pixels resultantes do processamento, que são os pixels centrais de cada "nuvem" anteriormente encontrada. A Tabela 6 mostra as coordenadas destes alvos.

Tabela 6 - Coordenadas dos pixels verdes da Figura 52, identificados como alvos

Coordenada X	**Coordenada Y**
113	26
124	104

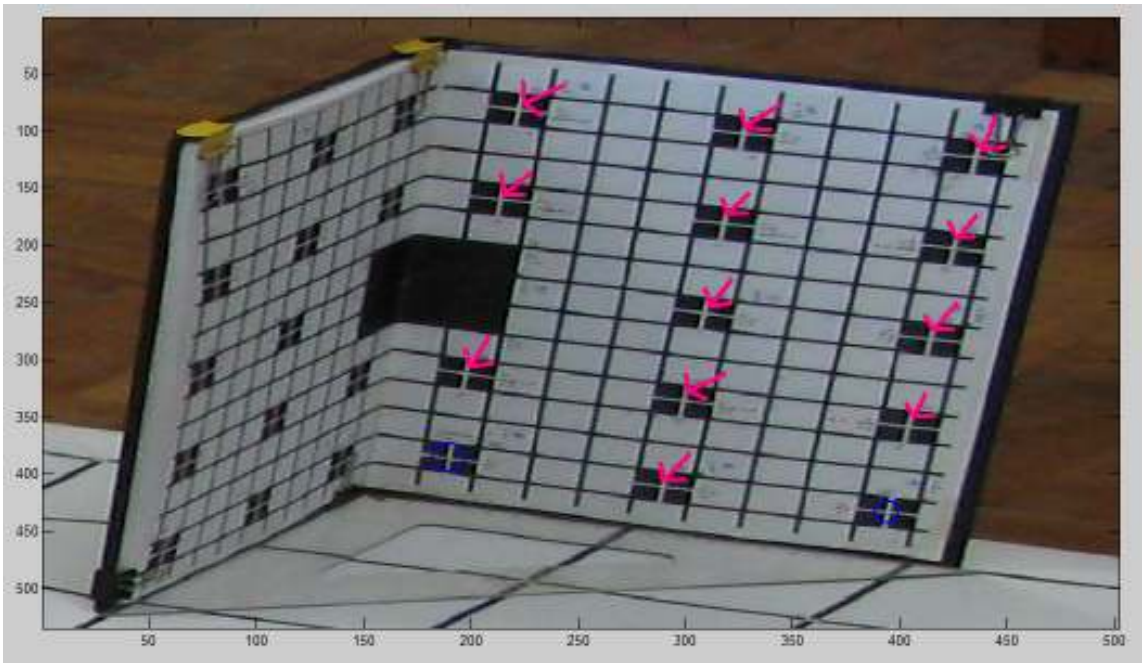

Figura 52 - Exemplo 3 do grid após a clusterização com seus alvos marcados na cor verde

A Figura 52 é mais um exemplo da rotina *subclust* aplicada. Dos 14 alvos encontrados, apenas 2 alvos não estão em suas posições centrais e, portanto, são falsos. Como não foi encontrado nenhum grupo de pixels na parte esquerda do grid, dos 28 alvos que deveriam ser encontrados (número total de alvos no grid), apenas 14 coordenadas foram identificadas. Estas são apresentadas na Tabela 7.

Tabela 7 - Coordenadas dos pixels verdes da Figura 53, identificados como alvos

Coordenada X	Coordenada Y
425	201
404	416
414	280
318	179
221	81
327	100
309	258
435	121
404	357
299	335
197	312
212	160
289	411
471	86

Capítulo 5

Conclusões

Os objetivos principais do projeto eram analisar e implementar uma rede neural para reconhecer padrões em imagens. Na etapa de desenvolvimento do algoritmo buscou-se generalizar o máximo possível, para que esta aplicação pudesse ser utilizada em diversos processos de reconhecimento de padrões em imagens. Para que a rede funcionasse corretamente foi empregado o treinamento através do algoritmo Backpropagation. Para realizar os testes, foram adotadas imagens de *grids* de calibração de câmeras utilizadas em processos de estereofotogrametria. A partir dessas imagens, as informações das coordenadas horizontais e verticais dos alvos deviam ser determinadas.

No teste preliminar realizado, todas as etapas descritas na metodologia foram executadas. O resultado final foi a localização de "nuvens" de pixels em torno dos pixels centrais dos elementos alvo. Além disso, alguns elementos foram erroneamente reconhecidos pela rede, fazendo com que fosse necessária a implantação de técnicas de pós-processamento.

A partir dos resultados do primeiro teste, foi vista a necessidade da implantação de mecanismos que atuassem na redução da dispersão dos resultados obtidos. Analisou-se que determinadas componentes se repetiam e influenciavam negativamente, portanto, foi utilizada a técnica de PCA (Principal Component Analysis) para reduzir as informações linearmente dependentes e redundantes e assim melhorar o resultado final. Além disso, o resultado final emitido pela rede nos apresentava conjuntos de coordenadas de pixels em torno dos alvos das imagens processadas. Como o objetivo é determinar a coordenada do pixel central de um alvo, foi implementada uma função de clusterização (subclust), que se baseou no agrupamento (através de médias) de coordenadas de pixels "próximos".

Nos testes utilizando as técnicas de pós-processamento descritas acima, notou-se que com o mesmo banco de imagens para treinamento, os erros da rede se mostraram baixos (abaixo de 1%), porém, o resultado final obtido a cada treinamento foi diferente. Os testes então foram realizados com várias imagens e diversos resultados de treinamentos, sendo

assim determinada uma configuração que se adequava melhor ao problema proposto. Utilizando esta configuração ótima, o reconhecimento dos pixels centrais dos alvos desejados se mostrou bastante eficiente, ocorrendo poucos erros.

Para a medição do treinamento da rede, chegou-se a conclusão de que somente o erro resultante do treinamento não pode ser utilizado como um indicador de qualidade. Observou-se que a rede treinada com erros acima de 20% possui um rendimento inaceitável para a aplicação que desejamos, porém, quando a rede foi treinada com erros da ordem de 0,1% foram observados resultados próximos do ideal, aceitáveis e ruins. A partir disso, podemos dizer que cada variável gerada em treinamentos distintos e empregada posteriormente na execução da rede, possui uma característica distinta e se adapta melhor ou pior à imagem à que ela se propõe a solucionar. Para tornar este projeto em um sistema robusto, devemos aumentar sua confiabilidade. Uma alternativa que pode ser implantada é o uso de algoritmos genéticos adaptados para atuarem em conjunto com a rede neural.

Este trabalho apresentou resultados bastante expressivos, dependendo da variável resultante do treinamento, e foi concluído que este processo realmente pode ser adotado na calibração de câmeras em estereofotogrametria, o que automatizaria uma etapa trabalhosa (determinação das coordenadas dos alvos, normalmente, realizada manualmente) e também fruto de muitos erros. Além disso, este processo pode ser utilizado em muitas outras aplicações na área da eletrônica, reduzindo custos de projeto, tanto em termos de mão-de-obra quanto em termos tecnológicos, já que este se mostrou bastante eficiente, utilizando poucos recursos computacionais.

Para trabalhos futuros, sugere-se implementar uma rotina de correspondência da lista de alvos identificados com a posição espacial real do alvo na grande de calibração.

Sugere-se, também, utilizar imagens com alvos diferentes dos que foram treinados neste projeto e que possuam tamanhos variáveis (dependendo da distância do alvo para a câmera) para que eles possam ser treinados por uma gama maior de imagens.

Referências Bibliográficas

[1] Silva, L.C., "Método Robusto para Calibração de Câmeras em Estereofotogrametria", Tese, COPPE-UFRJ, 2003.

[2] Matsumoto, E. Y., "Matlab 7 - Fundamentos", Ed. Erica, 2000.

[3] "Neural Networks" - http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#What%20is%20a%20Neural%20Network – 16/04/2008.

[4] Sistema para Análises Estatísticas – http://www.ufv.br/saeg/saeg44.htm - 20/05/2008.

Apêndice A

Código-fonte do algoritmo de conversão de imagens

```
function converte_imagens(nome_out_alvo,nome_out_nao_alvo)

listaAlvos = strcat('../Imagens/ALVOS/',ls('../Imagens/ALVOS/*.bmp'));
%strcat para concatenar o caminho das imagens, pois estao em pastas
diferentes
listaNaoAlvos = strcat ('../Imagens/NAO-ALVOS/',ls('../Imagens/NAO-
ALVOS/*.bmp'));

matriz = [];

for n=1:size(listaAlvos,1)
    % carregar a imagem da listaAlvos
    I = imread(listaAlvos(n,:));
    % converte p/ preto e branco
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [w, h, p]=size(I);                                  % funcao
conversao_cinza
    Idouble = double(I); %para fazer os calculos
    for i=1:w
        for j=1:h

Ipb(i,j)=0.299*Idouble(i,j,1)+0.587*Idouble(i,j,2)+0.114*Idouble(i,j,3);
        end
    end                                                 % END funcao
conversao_cinza
    %verficacao da conversao em nivel de cinza
    %figure;
    subplot(1,2,1);
    imagesc(I);
    title('Imagem RGB');
    subplot(1,2,2);
    imagesc(Ipb);
    colormap('gray');
    title('Mesma imagem em nivel de cinza');
    %pause;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    % converter para vetor
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    for i = 1:29                                      % funcao monta vetor
        for j = 1:29
            vetor((i-1)*29+j) = Ipb(i,j);
        end
    end                                               % END monta vetor
    %figure;
    plot(vetor,'k');
    title('Vetor da imagem');
    ylabel('nivel de cinza');
    xlabel('numero da amostra');
    %pause;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % acumular em uma matrix
    matrizAlvos(n,:) = vetor;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Para Matriz de Nao-
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Alvos

for n=1:size(listaNaoAlvos,1)
    % carregar a imagem da listaNaoAlvos
    I = imread(listaNaoAlvos(n,:));
    % converte p/ preto e branco
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [w, h, p]=size(I);                                % funcao
conversao_cinza
    Idouble = double(I); %para fazer os calculos
    for i=1:w
        for j=1:h

Ipb(i,j)=0.299*Idouble(i,j,1)+0.587*Idouble(i,j,2)+0.114*Idouble(i,j,3);
        end
    end                                               % END funcao
conversao_cinza
    %verficacao da conversao em nivel de cinza
    %figure;
    subplot(1,2,1);
    imagesc(I);
    title('Imagem RGB');
    subplot(1,2,2);
    imagesc(Ipb);
    colormap('gray');
    title('Mesma imagem em nivel de cinza');
    %pause;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % converter para vetor
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    for i = 1:29                                      % funcao monta vetor
        for j = 1:29
            vetor((i-1)*29+j) = Ipb(i,j);
        end
    end                                               % END monta vetor
    %figure;
```

```
    plot(vetor,'k');
    title('Vetor da Imagem');
    ylabel('nivel de cinza');
    xlabel('numero da amostra');
    %pause;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % acumular em uma matrix
    matrizNaoAlvos(n,:) = vetor;
end

%%%%%%%%%%%%

save(nome_out_alvo,'matrizAlvos')
save(nome_out_nao_alvo,'matrizNaoAlvos')  %para gravar em 2 arquivos
```

Código-fonte do algoritmo da rede neural “rede02”

```
close all; clear all;

load teste_a
load teste_na

num_alvos = size(matrizAlvos);
num_nao_alvos = size(matrizNaoAlvos);


% [A] Data
% substituir pelos seus dados (X e o t):
NUM_DADOS = num_alvos(1) + num_nao_alvos(1);
Input_size = num_alvos(2);
rand('state',0)
X = [matrizAlvos ; matrizNaoAlvos]';

%Matriz X de dados (normalizar entre -1 e 1 a matriz de entrada):
X = X - min(min(X));
X = X/max(max(abs(X)));
X = X*2 - 1;


%Saída desejada da rede (target)
t = [ones(1,num_alvos(1)) -ones(1,num_nao_alvos(1))];

% [A1] Randomize Data Order
randn('state',0);
aux = [X ; t ; randn(1,size(X,2))]';
aux = sortrows(aux,size(aux,2))';
X = aux(1:Input_size,:);
t = aux(Input_size+1,:);
clear aux;

% [B] Network Init

% [B1] Parameters
K = 2; % Number of Layers
eta = 1; % Learning Rate Initial Value
Delta = 1e-6; % Stop Criterion
N = size(X,2); % Number of Input Vectors
P = 1;
E = 4*P; % Number of Feed-Forward Iterations per Epoch
alpha = 0.99; % Learning Rate Decay Factor
mu = 0.65; % Momentum constant

% [B2] Layers
% inicializacao dos pesos da rede
L(1).W = rand(Input_size,Input_size)-0.5;
L(1).b = rand(Input_size,1)-0.5;
L(2).W = rand(1,Input_size)-0.5;
L(2).b = rand(1,1)-0.5;
```

```
for k=1:K,
    L(k).vb = zeros(size(L(k).b));
    L(k).vW = zeros(size(L(k).W));
end;

% [C] Batch Error Backpropagation Training

n=1; i=1; fim=0;
while not(fim),

    for k=1:K,
        L(k).db = zeros(size(L(k).b));
        L(k).dW = zeros(size(L(k).W));
    end;
    J(i) = 0;
    for ep=1:E,

        % [C1] Feed-Forward

        L(1).x = X(:,n);
        for k = 1:K,
            L(k).u = L(k).W*L(k).x + L(k).b;
            L(k).o = tanh(L(k).u);
            L(k+1).x = L(k).o;
        end;
        e = t(n) - L(K).o;
        J(i) = J(i) + (e'*e)/2;

        % [C2] Error Backpropagation

        L(K+1).alpha = e; L(K+1).W = eye(length(e));
        for k = fliplr(1:K),
            L(k).M = eye(length(L(k).o)) - diag(L(k).o)^2;
            L(k).alpha = L(k).M*L(k+1).W'*L(k+1).alpha;
            L(k).db = L(k).db + L(k).alpha;
            L(k).dW = L(k).dW + kron(L(k).x',L(k).alpha);
        end;
        n = n+1; if n>N, n=1; end;

    end;

    % [C3] Updates

    for k = 1:K,
        L(k).vb = eta*L(k).db + mu*L(k).vb;
        L(k).b = L(k).b + L(k).vb;
        L(k).vW = eta*L(k).dW + mu*L(k).vW;
        L(k).W = L(k).W + L(k).vW;
    end;
    J(i) = J(i)/E;

    % [C4] Stop criterion

    if (i>1),
       if (abs(J(i)-J(i-1))/(J(i)+ 1e-9) < Delta)|(i>5000),
           fim = 1;
       end;
    end;
```

```
    if not(fim)
        i = i+1; if n>N, n=1; end; eta = eta*alpha;
    end;

end;

% [D] Test - Processamento pela rede

hold on;
for n = 1:size(X,2),
    L(1).x = X(:,n);
    for k = 1:K,
        L(k).u = L(k).W*L(k).x + L(k).b;
        L(k).o = tanh(L(k).u);
        L(k+1).x = L(k).o;
    end;
    tfinal(n) = L(K).o;
end;


plot(sort(tfinal),'-x')
figure; plot(J);
```

Código-fonte do algoritmo da rede neural "treinarede"

```
function L = treinarede(nome_arquivo_alvo,nome_arquivo_nao_alvo)

load(nome_arquivo_alvo)
load(nome_arquivo_nao_alvo)

num_alvos = size(matrizAlvos);
num_nao_alvos = size(matrizNaoAlvos);


% [A] Data
% substituir pelos seus dados (X e o t):
NUM_DADOS = num_alvos(1) + num_nao_alvos(1);
rand('state',0)
Xorig = [matrizAlvos ; matrizNaoAlvos]';
Media = mean(Xorig,2);
Xorig = Xorig - repmat(Media,1,NUM_DADOS);

% Extração das componentes principais -
R=Xorig*Xorig'/NUM_DADOS;
[V,D] = eig(R);
V = fliplr(V);
D = flipud(diag(D));
energ = cumsum(D)/sum(D);
i = find(energ >= 0.9);
N_comp = i(1);
for n=1:N_comp
    V90(:,n) = V(:,n)/norm(V(:,n));
end
X = V90'*Xorig;
%[Vecs,Vals,Psi] = pc_evectors(Xorig,N_comp);
%X = Vecs(:,1:N_comp)'*(Xorig - repmat(Psi,1,NUM_DADOS));  % projeção de X
no novo espaço
Input_size = N_comp;

% Normalizacao
STD = std(X,[],2);

%Saída desejada da rede (target)
torig = [ones(1,num_alvos(1)) -ones(1,num_nao_alvos(1))];

% [A1] Randomize Data Order
randn('state',sum(100*clock))
%randn('state',0);
aux = [X ; torig ; randn(1,size(X,2))]';
aux = sortrows(aux,size(aux,2))';
X = aux(1:Input_size,:);
t = aux(Input_size+1,:);
clear aux;
```

```
% [B] Network Init

% [B1] Parameters
K = 2; % Number of Layers
eta = 1e-2; % Learning Rate Initial Value
Delta = 1e-6; % Stop Criterion
N = size(X,2); % Number of Input Vectors
P = 25;
alpha = 1; % Learning Rate Decay Factor
mu = 0; % Momentum constant

% [B2] Layers
% inicializacao dos pesos da rede
L(1).W = rand(Input_size,Input_size)-0.5;
L(1).b = rand(Input_size,1)-0.5;
L(2).W = rand(1,Input_size)-0.5;
L(2).b = rand(1,1)-0.5;
L(1).STD = STD;
%L(1).Vect = Vecs(:,1:N_comp);
L(1).Vect = V90;
L(1).Mean = Media;

for k=1:K,
    L(k).vb = zeros(size(L(k).b));
    L(k).vW = zeros(size(L(k).W));
end;

% [C] Sequential Error Backpropagation Training
hh=0;
kk=0;
J=0;
Jtotal = 0;
erro = [];
for p=1:P
    for n=1:N

        % [C1] Feed-Forward
        L(1).x = X(:,n)./L(1).STD;
        for k = 1:K,
            L(k).u = L(k).W*L(k).x + L(k).b;
            L(k).o = tanh(L(k).u);
            L(k+1).x = L(k).o;
        end;
        e = t(n) - L(K).o;
        %[t(n) L(K).o e]
        erro = [ erro (e'*e)/2];
        %Jtotal = Jtotal + (e'*e)/2;
        %J = [J Jtotal/(kk+1)];

        % [C2] Error Backpropagation

        L(K+1).alpha = e;
        L(K+1).W = eye(length(e));
        for k = fliplr(1:K),
            L(k).M = eye(length(L(k).o)) - diag(L(k).o)^2;
            L(k).alpha = L(k).M*L(k+1).W'*L(k+1).alpha;
            L(k).db = L(k).alpha;
            L(k).dW = kron(L(k).x',L(k).alpha);
        end;
```

```
        % [C3] Updates

        for k = 1:K,
            L(k).vb = eta*L(k).db + mu*L(k).vb;
            L(k).b = L(k).b + L(k).vb;
            L(k).vW = eta*L(k).dW + mu*L(k).vW;
            L(k).W = L(k).W + L(k).vW;
        end;

        % [C4] Stop criterion

        %     if (i>1),
        %         if (abs(J(i)-J(i-1))/(J(i)+1e-10) < Delta)
        %             fim = 1
        %         elseif (i > 10000)
        %             fim = 2
        %         end;
        %     end;
        %     if not(fim)
        eta = eta*alpha;
        %     end;
        hh = hh+1;
        kk = kk+1;
        if(hh> 100)
            hh = 0;
            kk/(N*P)
        end
    end;
end


% [D] Test - Processamento pela rede

plot(erro);
certo = 0;
for n = 1:size(X,2),
       L(1).x = (L(1).Vect'*Xorig(:,n))./L(1).STD;
    for k = 1:K,
        L(k).u = L(k).W*L(k).x + L(k).b;
        L(k).o = tanh(L(k).u);
        L(k+1).x = L(k).o;
    end;
    tfinal(n) = L(K).o;

    if(tfinal(n) > 0)
        s=sprintf('alvo = %1.2f',tfinal(n));
        %title(s)
    else
        s=sprintf('NAO alvo = %1.2f',tfinal(n));
        %title (s)
    end
end;
```

Código-fonte do algoritmo que localiza os alvos identificados pela rede na imagem e plota a figura

```
nome_da_imagem = 'IMAGEM_X';

 Grid = strcat('../Imagens/Copias Imagens
Principais/',nome_da_imagem,'.bmp');

% %for n=1:size(Grid,1) % se fosse para carregar mais de uma imagem
%      % carregar a imagem da Grid
      I = imread(Grid(1,:));
      % converte p/ preto e branco
%      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      [w, h, p]=size(I);                                    % funcao
conversao_cinza
      Idouble = double(I); %para fazer os calculos
      for i=1:w
          for j=1:h

Ipb(i,j)=0.299*Idouble(i,j,1)+0.587*Idouble(i,j,2)+0.114*Idouble(i,j,3);
          end
      end                                                   % END funcao
K=2;
     lado=29; %tamanho da matriz de janelamento
     n=1;
     alvo=[ ];

    for  i= 1:(size(Ipb,1)-lado) %for i=1:(61- lado)
       for  j= 1:(size(Ipb,2)-lado) %for  j=2:40
             Win=Ipb(i:i+(lado-1),j:j+(lado-1));

             vWin = monta_vetor(Win); % pega a imagem 29x29 uma a uma e
monta em um vetor de 1X841
             L(1).x = (L(1).Vect'*(vWin'-L(1).Mean))./L(1).STD; % analisa se
é alvo
             for k = 1:K,
                  L(k).u = L(k).W*L(k).x + L(k).b;
                  L(k).o = tanh(L(k).u);
                  L(k+1).x = L(k).o;
             end;
             tfinal = L(K).o;

             if (tfinal > 0.9) % se for alvo
                 alvo = [alvo;j+(lado-1)/2 i+(lado-1)/2]
%alvo(n,:)=[i+(lado-1)/2 j+(lado-1)/2];
                 %n=n+1;
             else
                 nalvo = 0;
             end

         end
    end
```

```
matriz_imagem = imread(Grid);
matriz_pontos = alvo;  % Colocar todos os pontos que deseja pintar
espessura_linha = 1;    % pinta so o pixel se quiser pintar ao redor do
pixel tb é so aumentar o numer 2, 3, 4 ...
r = 255;        % coordenada da parte vermelha
g = 0;
b = 0;



if(espessura_linha < 13)

    if ((espessura_linha > 0)&&(espessura_linha/(round(espessura_linha))))
        num_niveis = espessura_linha - 1;
        nlin_m_pontos = size(matriz_pontos,1);   % Numero de linhas da
matriz de pontos

        for k = 1 : nlin_m_pontos
            u = matriz_pontos(k,1);
            v = matriz_pontos(k,2);

            for cont_v = num_niveis*(-1):num_niveis

                for cont_u = num_niveis*(-1):num_niveis
                    if (((u+cont_u)>=1) &&
((v+cont_v)>=1)&&((u+cont_u)<=(size(matriz_imagem,2))) &&
((v+cont_v)<=(size(matriz_imagem,1))))
                        x=(u + cont_u);
                        y=(v + cont_v);

                        matriz_imagem(y,x,1) = r;
                        matriz_imagem(y,x,2) = g;
                        matriz_imagem(y,x,3) = b;
                    end
                    cont_u = cont_u + 1;
                end

                cont_v = cont_v +1;
            end

            k=k+1;
        end
    else
        % Mensagem de erro caso o valor da linha seja qualquer valor
        % diferente do permitido
       % msgbox('O valor da espessura da linha deve ser um inteiro
positivo. Ex: 1,2,3,...', 'Erro - Valor para espessura da Linha', 'error');
    end

else
    % Mensagem de erro caso o valor da linha seja maior do que 23, que
    % corresponderia 11 niveis de espessura a mais do que o pixel
    % original
    % No texto está 21 para termos uma margem de tolerancia.
    %msgbox('O valor maximo para a espessura da linha é de 12', 'Erro -
Espessura Máxima da Linha', 'error');

end
```

```
    matriz_imagem_s = matriz_imagem;

%matriz_imagem_s = pinta (matriz_imagem, matriz_pontos, espessura_linha,r,
g, b)


image(matriz_imagem_s);      % Usado so para ver a imagem

%imwrite(matriz_imagem_s, 'nome_do_arq_de_imagem_novo.bmp','bmp');  Grava
Imagem

%end

save alvo.mat
```

Código-fonte do algoritmo “mostra_C”

```
matriz_imagem = imread(Grid);
matriz_pontos = C;  % Colocar todos os pontos que deseja pintar
espessura_linha = 1;    % pinta so o pixel se quiser pintar ao redor do
pixel tb é so aumentar o numer 2, 3, 4 ...
r = 0;      % coordenada da parte vermelha
g = 255;
b = 0;



if(espessura_linha < 13)

    if ((espessura_linha > 0)&&(espessura_linha/(round(espessura_linha))))
        num_niveis = espessura_linha - 1;
        nlin_m_pontos = size(matriz_pontos,1);   % Numero de linhas da
matriz de pontos

        for k = 1 : nlin_m_pontos
            u = matriz_pontos(k,1);
            v = matriz_pontos(k,2);

            for cont_v = num_niveis*(-1):num_niveis

                for cont_u = num_niveis*(-1):num_niveis
                    if (((u+cont_u)>=1) &&
((v+cont_v)>=1)&&((u+cont_u)<=(size(matriz_imagem,2))) &&
((v+cont_v)<=(size(matriz_imagem,1))))
                        x=(u + cont_u);
                        y=(v + cont_v);

                        matriz_imagem(y,x,1) = r;
                        matriz_imagem(y,x,2) = g;
                        matriz_imagem(y,x,3) = b;
                    end
                    cont_u = cont_u + 1;
                end

                cont_v = cont_v +1;
            end

            k=k+1;
        end
    else
        % Mensagem de erro caso o valor da linha seja qualquer valor
        % diferente do permitido
       % msgbox('O valor da espessura da linha deve ser um inteiro
positivo. Ex: 1,2,3,...', 'Erro - Valor para espessura da Linha', 'error');
    end

else
    % Mensagem de erro caso o valor da linha seja maior do que 23, que
    % corresponderia 11 niveis de espessura a mais do que o pixel
    % original
    % No texto está 21 para termos uma margem de tolerancia.
```

```
    %msgbox('O valor maximo para a espessura da linha é de 12', 'Erro -
Espessura Máxima da Linha', 'error');

end


    matriz_imagem_s = matriz_imagem;

%matriz_imagem_s = pinta (matriz_imagem, matriz_pontos, espessura_linha,r,
g, b)


image(matriz_imagem_s);      % Usado so para ver a imagem

%imwrite(matriz_imagem_s, 'nome_do_arq_de_imagem_novo.bmp','bmp');  Grava
Imagem

%end
```