



UNIVERSIDADE FEDERAL DE OURO PRETO
ESCOLA DE MINAS
DEPARTAMENTO DE ENGENHARIA
DE CONTROLE E AUTOMAÇÃO



OSCAR FELICIO CANDIDO LONGUINHO

**APLICAÇÃO DE ALGORITMO SUPERVISIONADO NA CLASSIFICAÇÃO DE
PEÇAS**

Ouro Preto, 2019

OSCAR FELICIO CANDIDO LONGUINHO

**APLICAÇÃO DE ALGORITMO SUPERVISIONADO NA CLASSIFICAÇÃO DE
PEÇAS**

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para obtenção do Grau de Engenheiro de Controle e Automação.

Orientadora: Adrielle de Carvalho Santana

Ouro Preto

Escola de Minas – UFOP

Maio/2019

L858a Longuinho, Oscar Felício Cândido.
Aplicação de algoritmo supervisionado na classificação de peças [manuscrito] /
Oscar Felício Cândido Longuinho. - 2019.

87f.: il.: color; grafs; tabs.

Orientadora: Prof^a. MSc^a. Adrielle de Carvalho Santana.

Monografia (Graduação). Universidade Federal de Ouro Preto. Escola de Minas. Departamento de Engenharia de Controle e Automação e Técnicas Fundamentais.

1. Padrões - Reconhecimento. 2. Redes Neurais. 3. Perceptron Multicamadas. I. Santana, Adrielle de Carvalho. II. Universidade Federal de Ouro Preto. III. Título.

CDU: 681.5

Catálogo: ficha.sisbin@ufop.edu.br

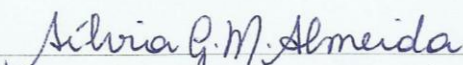
Monografia defendida e aprovada, em 24 de Maio de 2019, pela comissão avaliadora constituída pelos professores:



Profa. M.Sc. Adrielle de Carvalho Santana – Orientadora



Prof. Dr. Agnaldo José da Rocha Reis – Professor Convidado



Profa. Dra. Sílvia Graziella Moreira Almeida – Professora Convidada

RESUMO

Este trabalho estuda a utilização de uma rede neural na classificação de peças de acordo com sua cor e comprimento. Uma esteira transportadora foi desenvolvida para a execução do trabalho, nela está posicionado o sensor RGB, responsável pela obtenção dos dados das componentes vermelha, verde e azul da cor da peça e o sensor de barreira, responsável pela leitura do valor correspondente ao comprimento da peça. O controle da esteira e dos sensores é feito por meio de uma plataforma Arduino que é responsável também pelo controle do módulo de cartão SD onde são armazenados os dados de leitura dos sensores. Um conjunto de amostras é coletado e utilizado para o treinamento da rede, para isto, foi implementado um código utilizando o software MATLAB capaz de realizar o treinamento e execução de uma rede neural perceptron multicamadas. Foram realizados testes de validação cruzada para analisar qual estrutura da rede, ou seja, o número de camadas e o número de neurônios em cada camada, melhor se adaptava ao problema de classificação proposto. Em seguida os demais parâmetros da rede, tais como a taxa de aprendizagem e os critérios de parada para o treinamento, foram obtidos por meio de testes em que a eficiência da rede foi analisada, comparando o resultado de treinamentos realizados com parâmetros diferentes. Após a obtenção dos parâmetros da rede que melhor atendiam os requisitos do trabalho, foi realizado a coleta de dados para a validação da rede. Para o processo de validação foi desenvolvido um aplicativo classificador que recebe os dados de leitura dos sensores se comunicando com a plataforma Arduino por meio da porta serial de um computador. Este aplicativo executa a rede perceptron multicamadas utilizando os pesos sinápticos obtidos durante o treinamento realizado no MATLAB e exibe uma resposta visual com o resultado da classificação feita pela rede. Durante os testes realizados foi observada a necessidade de ajustes na estrutura física do trabalho para que a rede desenvolvida tivesse um desempenho adequado. Com os ajustes realizados a rede apresentou um resultado de classificação satisfatório com uma taxa de acertos de 81,6%, mostrando ser viável a utilização da rede perceptron multicamadas para a classificação das peças utilizando um sistema de baixo custo e com apenas duas características.

Palavras-chave: Reconhecimento de padrões, redes neurais, perceptron multicamadas.

ABSTRACT

This work studies the use of a neural network in the classification of pieces according to their color and length. A conveyor belt was developed to execute this work, in it is positioned the RGB sensor, responsible for obtaining the data of the red, green and blue components of the part color and the barrier sensor, responsible for reading the value corresponding to the part length. The control of the conveyor belt and the sensors is done by means of an Arduino platform that is also responsible for the control of the SD card module where the data of the reading from the sensors are stored. A set of samples is collected and used for the training of the network, for this, a code was implemented using the MATLAB program capable of performing the training and execution of a multi-layer perceptron neural network. Cross validation tests were performed to analyze which network structure, i.e., the number of layers and the number of neurons in each layer, was better adapted to the proposed classification problem. Then, the other parameters of the network, such as the learning rate and the stopping criteria for the training, were obtained by means of tests in which the network efficiency was analyzed, comparing the results of training with different parameters. After obtaining the parameters of the network that best met the requirements of the work, the data was collected for the validation of the network. For the validation process was developed a classifier application that receives the reading data from the sensors communicating with the Arduino platform through the serial port of a computer. This application executes the multilayer perceptron network using the synaptic weights obtained during the training performed in the MATLAB and displays a visual response with the result of the classification made by the network. During the tests, it was observed the need for adjustments in the physical structure of the work so that the developed network had an adequate performance. With the adjustments made the network presented a satisfactory classification result with a rate of 81,6%, showing that the use of the multilayer perceptron network is feasible for the classification of the parts using a low cost system and with only two characteristics.

Keywords: Pattern recognition, neural networks, multilayer perceptron.

LISTA DE FIGURAS

Figura 2.1 - Neurônio biológico	12
Figura 2.2 - Neurônio Artificial.....	13
Figura 2.3 - Rede Perceptron.....	14
Figura 2.4 - Fronteira de separação das classes em uma rede com duas entradas.....	16
Figura 2.5 - Rede perceptron multicamadas.....	18
Figura 2.6 - rede PMC.....	19
Figura 3.1 - Esquema de funcionamento do projeto	22
Figura 3.2 - Arduino Mega 2560	22
Figura 3.3 - Módulo Sensor TCS3200.....	24
Figura 3.4 - Diagrama de blocos funcionais.....	25
Figura 3.5 - Resistor Dependente da Luz.....	26
Figura 3.6 - Resistência vs. Intensidade da Luz	26
Figura 3.7 - Diodo LASER.....	27
Figura 3.8 - Sensor de barreira de luz	28
Figura 3.9 - Circuito receptor do sensor de barreira	29
Figura 3.10 - Peças plásticas representando classes	31
Figura 3.11 Vista lateral da esteira	31
Figura 3.12 - Suporte para sensores.....	32
Figura 3.13 - Esteira transportadora.....	33
Figura 3.14 - Sensor de Cores e Sensor de Barreira	36
Figura 3.15 - Cartão micro SD	37
Figura 3.16 - exemplo de dados coletados	37
Figura 3.17 - Fluxograma de funcionamento do sistema coletor de dados.....	38
Figura 3.18 - Grupo com dados dos sensores e identificação da classe	39
Figura 3.19 - Taxa de acerto das topologias candidatas	46
Figura 3.20 - Evolução do EQM com valor de taxa de aprendizagem igual a 0,1	48
Figura 3.21 - Evolução do EQM com valor de taxa de aprendizagem igual a 0,5	49
Figura 3.22 - Evolução do EQM com valor de taxa de aprendizagem igual a 1	49
Figura 3.23 - Tela do aplicativo de classificação	51
Figura 4.1 - Sensor com anteparo instalado	53

LISTA DE TABELAS

Tabela 2.1 Configuração da escala de frequência.....	24
Tabela 1.2 - Configuração dos filtros dos fotodiodos	25

LISTA DE ABREVIATURAS E SIGLAS

CSV	Character-separated values
EQM	Erro Quadrático Médio
LASER	Light Amplification by Stimulated Emission of Radiation
LDA	Linear discriminant analysis
LED	Light Emitter Diode
MDF	Medium Density Fiberboard
PMC	Perceptron Multicamadas
POO	Programação Orientada a Objetos
RAD	Rapid Application Development
RDL	Resistor Dependente da Luz
SD	Secure Digital
USB	Universal Serial Bus

Sumário

1. INTRODUÇÃO	11
1.1. Formulação do problema e motivação	11
1.2. Objetivo geral	13
1.3. Objetivos específicos	13
1.4. Estrutura do trabalho	14
2. BASE TEÓRICA	15
2.1. Redes Neurais Artificiais	15
2.2. Rede Perceptron	17
2.3. Perceptron Multicamadas	20
2.3.1. Ajustes dos pesos sinápticos em uma rede PMC	22
3. DESENVOLVIMENTO	24
3.1. Esquema geral	24
3.2. Materiais utilizados	25
3.2.1. Plataforma Arduino	25
3.2.2. Resistor Dependente da Luz (RDL)	29
3.2.3. Diodo Laser	30
3.2.4. Sensor de Barreira	30
3.2.5. Peças para classificação	33
3.2.6. Esteira transportadora	33
3.3. Coleta de Dados	36
3.3.1. Cor da peça	36
3.3.2. Comprimento da peça	37
3.3.3. Sistema coletor de dados	37
4. RESULTADOS	42
4.1. Treinamento da Rede Perceptron Multicamadas	42
4.1.1. Ajuste dos dados das amostras	42
4.1.2. Algoritmo Perceptron Multicamadas	44
4.2. Estrutura e parâmetros da rede PMC	48
4.2.1. Definição da estrutura da rede PMC	48
4.2.2. Taxa de Aprendizagem	51
4.3. Validação da Rede	54

4.3.1. Aplicativo para classificação das peças.....	55
4.4. Interferências externas e ajustes	57
4.5. Readequação da rede e resultados	58
5. CONCLUSOES E TRABALHOS FUTUROS	59
6. REFERÊNCIA BIBLIOGRÁFICA.....	61
APENDICE A - Classe Motor	64
APENDICE B – Classe “ColorSensor”	65
APÊNDICE C – Coletor de Dados	70
APÊNDICE D - Função de treinamento da rede PMC	74
APENDICE E – Script de validação cruzada	77
APENDICE F – Arquivo de configuração da rede	79
APENDICE G – Código fonte do aplicativo classificador.....	81

1. INTRODUÇÃO

Neste capítulo é feita a apresentação do problema e dos processos envolvidos e também são abordados os objetivos que se deseja alcançar com este trabalho.

1.1. Formulação do problema e motivação

Uma das etapas críticas da produção industrial é a inspeção de defeitos. Garantir que um produto atenda aos requisitos técnicos especificados é de fundamental importância no controle das etapas de produção impactando no custo da manufatura, na segurança dos processos, na qualidade final do produto e na satisfação do cliente a quem se destina o resultado final.

É cada vez mais comum o uso de técnicas de reconhecimento de padrões em aplicações industriais com fim de identificar características de peças, mas ainda em muitos casos os procedimentos de análises de defeitos ainda são executados de forma não ótima, seja pela complexidade do objeto analisado ou mesmo pela limitação ao acesso a um sistema automatizado que execute esta tarefa com a exatidão necessária.

Ao longo dos anos, com o avanço tecnológico e a maior acessibilidade aos recursos computacionais, tecnologias que anteriormente tinham acesso dificultado pelos elevados custos e pela complexidade de implementação têm se tornado mais disponíveis, e o uso de tais facilidades é cada vez mais frequente no desenvolvimento de soluções industriais que sejam capazes de verificar a existência de defeitos em objetos.

O processo de identificação de características físicas de um objeto é um procedimento essencial para que um sistema automatizado seja capaz de identificar defeitos em peças. Atualmente existem inúmeras alternativas que possibilitam a aquisição de dados que permitem extrair informações sobre as características de um objeto, mas desde a utilização de imagens de câmeras ao uso de sensores mais simples, é fator comum entre todas as soluções a necessidade de tratamento dos dados coletados a fim de tornar possível o reconhecimento dos padrões desejados.

Segundo BIANCHI (2006, p 21):

Um problema de reconhecimento de padrão consiste de uma tarefa de classificação ou categorização, onde as classes são definidas pelo projetista do sistema (classificação supervisionada) ou são “aprendidas” de acordo com a similaridade dos padrões (classificação não supervisionada).

Num problema de classificação supervisionada, o sistema recebe um conjunto de exemplos já classificados de acordo com critérios pré-estabelecidos e um algoritmo é responsável pela obtenção de uma função que consiga separar corretamente as classes de acordo com os exemplos dados. Esta fase do problema é chamada de treinamento e permite que o sistema aprenda a identificar as características como sendo ou não de uma determinada classe para que posteriormente, na fase de execução, ao se deparar com um novo item ainda não classificado, seja possível para o sistema identificá-lo como pertencente a uma das classes aprendidas.

Em um agrupamento não supervisionado um conjunto de dados é submetido à avaliação pelo sistema que utiliza um algoritmo para identificar itens com características similares e separá-los em conjuntos.

A diferença fundamental entre as duas estratégias é que enquanto na busca supervisionada as classes são definidas de acordo com as necessidades do projetista e o sistema se adequa às estas características, no agrupamento não supervisionado, apesar de geralmente o número de classes poderem ser determinadas, o sistema é que é responsável por especificar as características que definem ou não um grupo.

Levando em consideração que as características das peças a serem classificados são muito bem definidas, e deseja-se saber se o item avaliado está de acordo com parâmetros conhecidos, este trabalho pretende analisar algumas estratégias de reconhecimento supervisionado de padrões para executar esta tarefa.

De acordo com SILVA (2010,p 15), o processo de reconhecimento de padrões “[...] engloba três fases principais: a representação dos dados de entrada e sua mensuração; a extração das características e, por último, a classificação do objeto em estudo”.

A primeira fase diz respeito à obtenção dos dados do item a ser classificado pelos sensores e a representação deste da forma que o sistema consiga, na etapa seguinte, reconhecer as características lidas e por fim, realizar a tomada de decisão, ou seja, classificar o item como pertencente a uma das classes propostas.

A leitura realizada por alguns tipos de sensores podem muitas vezes resultar em valores que oscilam, havendo a necessidade de que o algoritmo utilizado seja capaz de generalizar estas oscilações.

Dentre as diversas possibilidades de implementação de algoritmos para reconhecimento de padrões, há de se destacar o uso de técnicas como a LDA (Linear discriminant analysis) que consiste em encontrar uma combinação linear que caracteriza ou separa as classes presentes em um conjunto de dados, Máquinas de vetor de suporte, que busca encontrar um hiperplano de separação dos dados por meio de análise estatística e de inteligência artificial como o caso das redes neurais artificiais. Segundo MUELLER (1996, p 35) “As redes neurais artificiais se fundamentam nos estudos sobre a estrutura do cérebro humano para tentar emular sua forma inteligente de processar informações”. HAYKIN (2001, p 28) define uma rede neural como:

[...] um processador maciçamente paralelamente distribuído constituído de unidades de processamentos simples, que têm a propensão natural para armazenar conhecimento experimental e torná-lo disponível para uso.

Ela se assemelha ao cérebro em dois aspectos:

- O conhecimento é adquirido pela rede a partir de seu ambiente através de um processo de aprendizagem;
- Forças de conexão entre neurônios, conhecidas como pesos sinápticos, são utilizadas para armazenar o conhecimento adquirido.

As redes neurais podem apresentar diversos modelos de arquitetura, que se diferenciam pela disposição dos neurônios em sua estrutura. MATICH (2001, p 27) descreve os parâmetros fundamentais de uma rede neural como sendo “[...] o número de camadas, o número de neurônios por camada, o grau de conectividade e o tipo de ligações entre os neurônios.”

Dentre as possíveis arquiteturas, para este trabalho, foi optado pelo estudo de variantes do algoritmo perceptron que segundo HAYKIN (2001, p 143) “[...]é a forma mais simples de uma rede neural usada para classificação de padrões linearmente separáveis [...]”. As variantes estudadas são algoritmos baseados no perceptron que têm melhor comportamento em conjunto de dados não linearmente separáveis.

1.2. Objetivo geral

Utilizar técnicas clássicas de reconhecimentos de padrões para implementar um algoritmo supervisionado com a finalidade de realizar a identificação das características de um lote de peças e classificá-las de acordo com parâmetros pré-estabelecidos baseados nas suas características físicas tais como cor e formato.

1.3. Objetivos específicos

- Desenvolver um código capaz de executar o algoritmo Perceptron Multicamadas;
- Construir um sistema de esteira e sensores capaz de coletar dados de peças que posteriormente serão analisadas pelo algoritmo da rede neural;

- Analisar o comportamento do Perceptron Multicamadas na classificação das peças apresentadas;
- Desenvolver um software que permita a visualização em tempo real das classificações feitas pela rede quando em modo de execução.

1.4. Estrutura do trabalho

Este trabalho foi dividido em cinco partes: Introdução, Base teórica, Desenvolvimento, Resultados e Conclusões e Trabalhos Futuros.

Na primeira parte é feita a introdução trabalho proposta assim como os objetivos que pretendiam ser alcançados.

A seguir é apresentado o embasamento teórico dos conceitos estudados no trabalho. São apresentadas as características de funcionamento do treinamento e execução da rede neural Perceptron Multicamadas.

Na terceira parte é detalhado o desenvolvimento do trabalho, assim como os materiais utilizados, os algoritmos implementados, o processo de coleta de dados.

Na quarta parte são apresentados os processos de definição dos parâmetros e treinamento da rede neural e analisados os resultados alcançados.

Por fim, na quinta parte, são apresentadas as conclusões do trabalho e sugeridas possíveis melhorias para futuras implementações semelhantes.

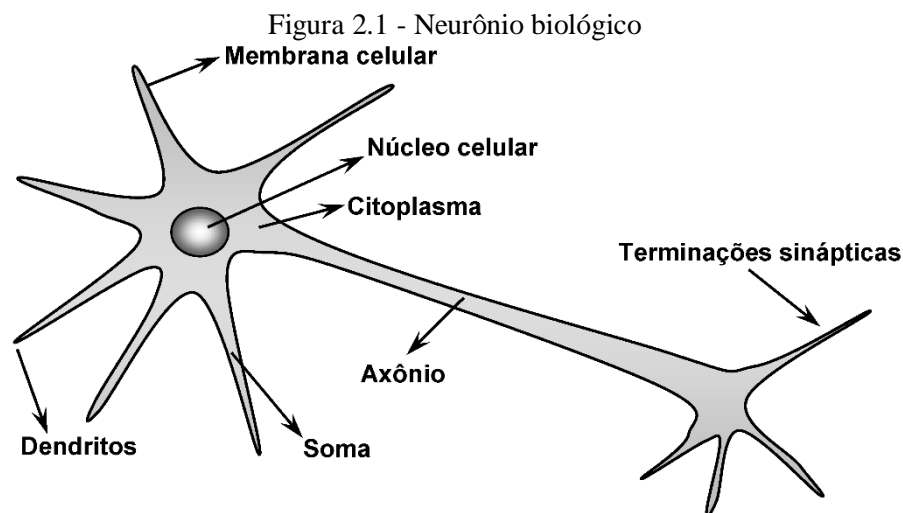
2. BASE TEÓRICA

Neste capítulo serão apresentados os embasamentos teóricos utilizados neste trabalho, englobando tanto o estudo dos algoritmos da rede neural quanto a especificação dos dispositivos utilizados na montagem do projeto.

2.1. Redes Neurais Artificiais

O neurônio é a célula básica do sistema nervoso humano e é basicamente responsável por processar impulsos nervosos.

A estrutura física do neurônio é dividida em três partes principais, dendritos, corpo celular e axônio. Os dendritos são responsáveis por receber os estímulos emitidos por outros neurônios, estes estímulos são processados pelo corpo celular que de acordo com um limiar poderá disparar um impulso através do axônio.



Fonte: Silva et AL., 2016

As terminações do axônio são chamadas sinapses, e permitem a comunicação com os dendritos de outros neurônios.

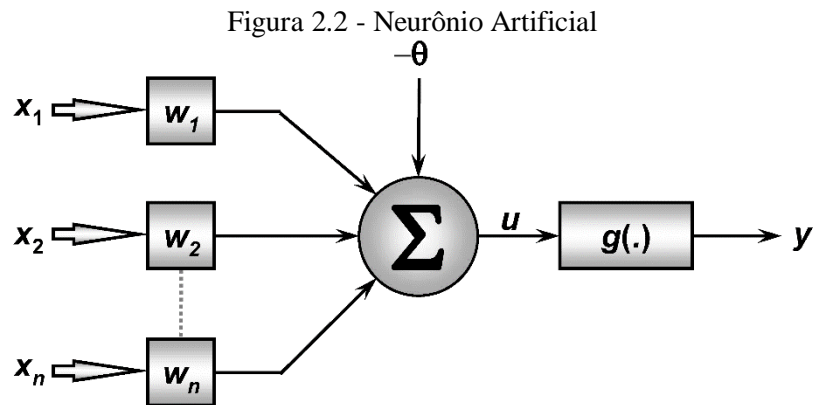
Um neurônio artificial é uma estrutura baseada no funcionamento do neurônio biológico, e procura aproximar o processamento computacional ao processamento do cérebro humano.

Segundo Silva, Spatti e Flauzino (2016, p 24) redes neurais artificiais são:

[...] modelos computacionais inspirados no sistema nervoso de seres vivos. Possuem a capacidade de aquisição e manutenção do conhecimento (baseado em informações) e podem ser definidas como um conjunto de unidades de

processamento, caracterizadas por neurônios artificiais, que são interligados por um grande número de interconexões[...]

O modelo de neurônio artificial mais comumente utilizado nas diversas arquiteturas de redes neurais existentes é o proposto por McCulloch & Pitts, que pode ser observado na figura 2.2.



Fonte: Silva et al., 2016

Fazendo um paralelo com o neurônio biológico, os estímulos recebidos são representados pelo conjunto $\{X_1, X_2, X_3, \dots, X_n\}$ que são ponderados pelos pesos sinápticos $\{W_1, W_2, W_3, \dots, W_n\}$, estes valores são agregados pela combinação linear $\{\Sigma\}$, e o valor deste somatório é comparado com o limiar de ativação $\{\Theta\}$, tendo como resultado o potencial de ativação $\{u\}$, neste potencial é aplicada a função de ativação $\{g(\cdot)\}$ que gera por fim a saída $\{y\}$ que é o resultado final do processamento do neurônio.

Dentre as características mais importantes das redes neurais artificiais estão a capacidade de aprendizado e a habilidade de generalização.

Uma rede neural é capaz de ao receber dados de amostras por meio de um método de treinamento, extrair informações dessas amostras aprendendo como os dados se relacionam entre si e armazenar este conhecimento.

Quando uma amostra ainda não analisada é apresentada a uma rede previamente treinada, esta tem a capacidade de compreender as informações contidas na amostra utilizando o conhecimento adquirido durante o treinamento e gerar uma saída compatível. Esta habilidade é chamada de generalização.

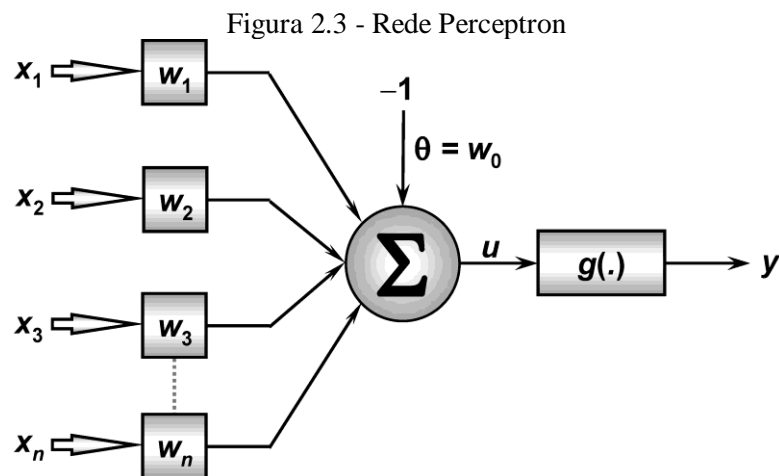
Estas características permitem a implementação de soluções em que estão presentes variáveis que sofrem influência de ruído e também naquelas em que as relações que representam as interações entre os dados são desconhecidas ou muito complexas.

Redes neurais têm se tornado cada vez mais comuns, sendo encontradas em diversas aplicações, desde análise e previsão de dados financeiros, diagnósticos médicos e análise de imagens de satélite a controle de eletrodomésticos, classificações de padrões de escrita e fala e aplicações em veículos autônomos.

2.2. Rede Perceptron

Em 1958, Inspirado pelos princípios biológicos do neurônio, Franck Rosenblatt desenvolveu o conceito da rede Perceptron, que é a configuração mais simples de uma rede neural e que tinha por objetivo receber sinais eletrônicos e processá-los com a intenção de reconhecer padrões geométricos de objetos.

A figura 2.3 representa uma rede perceptron formada por n entradas.



Fonte: Silva et al, 2016

Na rede mostrada na figura 2.3 os sinais de entrada X_i são multiplicados pelos pesos sinápticos W_i , posteriormente é realizado o somatório destes valores juntamente com o valor do limiar de ativação, neste caso representado pelo valor -1 multiplicado pelo peso W_0 . A seguir o resultado do somatório (u) é analisado por uma função de ativação $g(\cdot)$ resultando por fim na saída da rede (y).

Matematicamente, o processamento da rede perceptron pode ser representado da seguinte forma:

$$\begin{cases} u = \sum_{i=1}^n W_i \cdot X_i - \theta \\ y = g(u) \end{cases} \quad (2.1)$$

As funções de ativação geralmente utilizadas em uma rede perceptron são a função degrau ou a função degrau bipolar, de forma que a saída da rede pode apresentar somente dois valores possíveis, 0 e 1 no caso da função degrau ou -1 e 1 quando a função degrau bipolar está sendo utilizada.

Desta forma, fica claro que o algoritmo perceptron é capaz de classificar uma entrada apresentada em no máximo duas classes.

Utilizando a função degrau bipolar, a saída da rede (y) pode ser representada da seguinte forma:

$$y = \begin{cases} 1, & \text{se } \sum_{i=1}^n W_i \cdot X_i - \theta \geq 0 \\ -1, & \text{se } \sum_{i=1}^n W_i \cdot X_i - \theta < 0 \end{cases} \quad (2.2)$$

Em uma rede formada por apenas duas entradas (X_1 e X_2), as expressões possíveis para a saída (y) seriam:

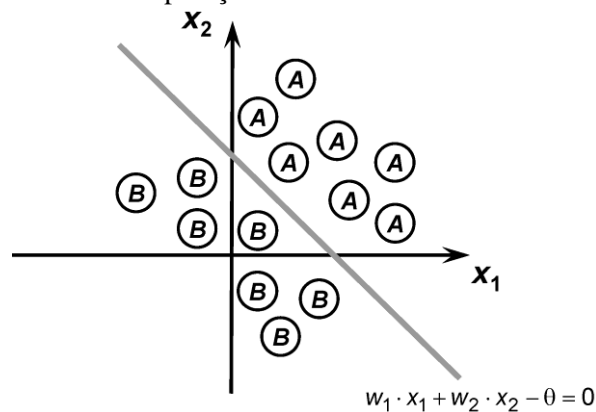
$$y = \begin{cases} 1, & \text{se } W_1 \cdot X_1 + W_2 \cdot X_2 \geq 0 \\ -1, & \text{se } W_1 \cdot X_1 + W_2 \cdot X_2 < 0 \end{cases} \quad (2.3)$$

Sendo assim, a fronteira entre as duas classes possíveis é definida pela reta representada pela equação :

$$W_1 \cdot X_1 + W_2 \cdot X_2 = 0 \quad (2.4)$$

Representada graficamente na figura 2.4.

Figura 2.4 - Fronteira de separação das classes em uma rede com duas entradas



Fonte: Silva et al., 2016

A análise da fronteira de classificação da rede perceptron demonstra que esta rede é capaz de identificar classes que sejam delimitadas por apenas uma linha (no caso da rede de duas entradas), ou mais amplamente, o algoritmo perceptron é capaz de classificar dados que sejam linearmente separáveis.

O processo de treinamento da rede consiste em apresentar um conjunto de amostras ao algoritmo, analisar as saídas apresentadas pela rede comparando-as com as saídas desejadas. Os pesos sinápticos e do limiar de ativação são ajustados até que a rede consiga classificar corretamente todas as amostras apresentadas.

As expressões que representam a atualização dos pesos sinápticos (W_i) e do limiar de ativação (θ) são as seguintes:

$$\begin{cases} W_i^{atual} = W_i^{anterior} + \eta \cdot (d^{(k)} - y) \cdot X_i^{(k)} \\ \theta^{atual} = \theta^{anterior} + \eta \cdot (d^{(k)} - y) \cdot (-1) \end{cases} \quad (2.5)$$

Onde η representa a taxa de aprendizagem, um valor utilizado para determinar a taxa com que os valores dos pesos podem variar de acordo com o erro encontrado entre o valor desejado para uma amostra k ($d^{(k)}$) e o valor da saída da rede (y). O valor $X_i^{(k)}$ representa o valor de uma amostra k para o neurônio i .

2.3. Perceptron Multicamadas

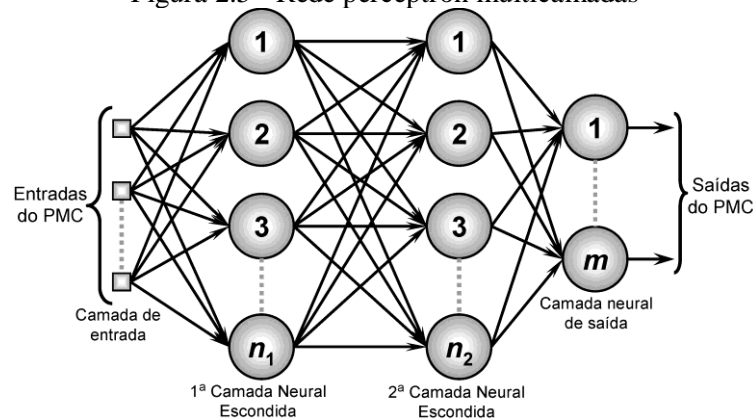
Perceptron Multicamadas (PMC) é uma configuração de rede neural em que os neurônios estão dispostos em pelo menos três camadas, uma camada de entrada composta pelos dados das amostras, uma camada de saída e uma ou mais camadas intermediárias chamadas de camadas ocultas. Diferente do perceptron simples, a rede PMC pode conter vários neurônios tanto nas camadas intermediárias quanto na camada de saída.

A rede PMC é uma das configurações mais versáteis para redes neurais, sendo empregadas não só em reconhecimento de padrões, mas também em aplicações de aproximação de funções, previsão de séries temporais e otimização de sistemas, dentre outras (SILVA et al., 2016). CARDOSO et al. (2010) apresentam o desenvolvimento de um sistema que utiliza uma rede PMC, capaz de executar comandos a partir do reconhecimento de voz. ADAMOWICZ, SAMPAIO e BARBOZA (2002) utilizam redes neurais PMC na tarefa de reconhecimento de padrões em análise de dados econômico-financeiros de empresas. NETO e FILHO (2013) demonstram o uso do algoritmo PMC no reconhecimento de placas de automóveis. AGUIAR (2010) descreve a utilização da rede PMC na detecção de padrões de vazamento em dutos.

Diferentemente da rede perceptron simples a perceptron multicamadas é capaz de classificar amostras presentes em conjuntos de dados não linearmente separáveis, o que amplia a aplicabilidade do algoritmo na tarefa de reconhecimento de padrões.

A camada de saída em uma rede PMC pode ser composta por mais de um neurônio, sendo possível que cada neurônio presente na camada de saída seja responsável por representar uma classe analisada pela rede, desta forma uma rede configurada para reconhecer m classes, terá sua camada de saída formada por m neurônios, além de permitir outras abordagens, como por exemplo, a representação binária, onde uma camada de saída compostas por m neurônios é capaz de representar até 2^m classes. Esta característica permite que as redes PMC possam classificar conjuntos de dados com inúmeras classes presentes.

Figura 2.5 - Rede perceptron multicamadas



Fonte: Silva et al, 2016

O processo de aprendizagem da rede PMC é baseado no algoritmo *backpropagation*. O processo de execução deste algoritmo é dividido em duas fases, a primeira chamada de *forward* ou avanço e a segunda chamada de *backward* ou retrocesso.

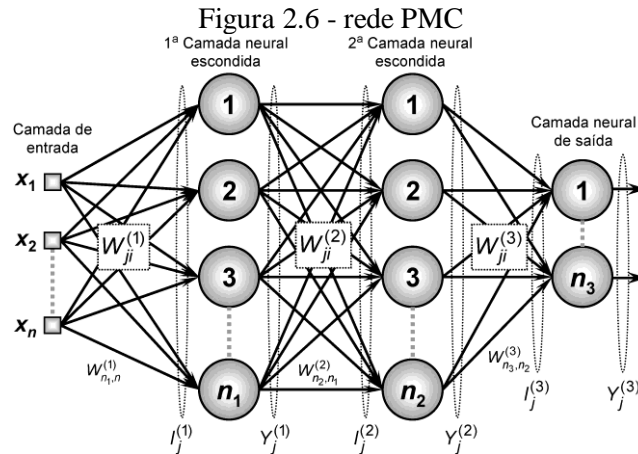
O processo de treinamento da rede PMC consiste na execução sucessiva das fases do algoritmo *backpropagation*, comparando os resultados obtidos com os desejados até que um determinado critério de parada seja alcançado. Este critério de parada está geralmente relacionado com o nível de erro entre as amostras analisadas e as respostas obtidas na saída da rede, mas também é comum que um número máximo de iterações também seja considerado como limite para a execução do algoritmo.

Na fase *forward* a rede PMC se comporta exatamente como a rede perceptron, onde os valores da amostra apresentada são ponderados por pesos, agrupados e é aplicada uma função de ativação. Nas camadas intermediárias as saídas da camada anterior funcionam como entradas para a seguinte, desta forma, os resultados do processamento são propagados camada por camada até a saída da rede.

Após a obtenção dos dados de saída com a execução da primeira fase, é iniciado o processo *backward*. Nesta etapa os pesos de cada camada são atualizados, iniciando-se a partir da camada de saída até a camada de entrada da rede. O processo é iniciado calculando-se o gradiente do erro na camada de saída com relação aos pesos sinápticos da camada anterior, e assim sucessivamente até que os pesos de todas as camadas sejam atualizados.

Ao final da execução das duas fases do algoritmo uma nova amostra é apresentada a rede, que executa novamente os processos até que o critério de parada definido seja alcançado.

2.3.1. Ajustes dos pesos sinápticos em uma rede PMC



Fonte: Silva et al, 2016

Na figura 2.6 é possível observar todas as variáveis que são utilizadas na execução da rede PMC. Na notação adotada, $W_{ji}^{(L)}$ representa a matriz de pesos da camada $\{L\}$ para cada neurônio $\{j\}$ nesta camada em relação a saída $\{i\}$ da camada anterior. A variável $I_j^{(L)}$ representa o vetor com as entradas já ponderadas para cada neurônio $\{j\}$ da camada $\{L\}$ e a variável $Y_j^{(L)}$ representa a saída de cada neurônio $\{j\}$ da camada $\{L\}$. O número de neurônios em cada camada $\{L\}$ é representado pela variável n_L .

Como citado anteriormente, os primeiros pesos ajustados são os da camada de saída, para tanto, o primeiro passo é calcular o gradiente local para cada neurônio na camada de saída por meio da equação 2.6:

$$\delta_j^{(saída)} = (d_j - Y_j^{(saída)}) \cdot g'(I_j^{(saída)}) \quad (2.6)$$

Na expressão 2.6 o erro entre o valor desejado (d_j) e o valor obtido na saída é multiplicado pela derivada da função de ativação ($g'(\cdot)$) aplicada no valor da entrada para o neurônio analisado, desta forma se faz necessário que a função de ativação em uma rede PMC seja diferenciável, geralmente são utilizadas como função de ativação nas redes PMC a função sigmóide (Equação 2.7) ou a função tangente hiperbólica (Equação 2.8) com suas respectivas derivadas (Equações 2.9 e 2.10).

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad (2.7)$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.8)$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \quad (2.9)$$

$$\tanh'(x) = 1 - \tanh^2(x) \quad (2.10)$$

Após o cálculo do gradiente local os pesos para a camada de saída são atualizados utilizando a equação 2.11, onde a variável η representa a taxa de aprendizagem da rede.

$$W_{ji}^{(saída)}(novo) = W_{ji}^{(saída)}(anterior) + \eta \cdot \delta_j^{(saída)} \cdot Y_i^{(camada anterior)} \quad (2.11)$$

Com os pesos da camada de saída atualizados os pesos das camadas intermediárias podem ser calculados, analogamente ao processo de atualização da camada de saída, o primeiro passo é calcular o gradiente local para cada neurônio da camada em estudo, utilizando a equação 2.12.

$$\delta_j^{(L)} = -\left(\sum_{k=1}^{n_{L+1}} \delta_k^{(L+1)} \cdot W_{kj}^{(L+1)}\right) \cdot g'(I_j^{(L)}) \quad (2.12)$$

E os pesos sinápticos da camada são atualizados utilizando a equação 2.13.

$$W_{ji}^{(L)}(novo) = W_{ji}^{(L)}(anterior) + \eta \cdot \delta_j^{(L)} \cdot Y_i^{(L-1)} \quad (2.13)$$

Por fim, são atualizados os pesos da camada de entrada. O gradiente local para os neurônios da camada de entrada são calculados da mesma forma que os das camadas intermediárias, por meio da equação 2.12. Para os pesos dos neurônios da camada de entrada a equação 2.14 é utilizada.

$$W_{ji}^{(1)}(\textit{novo}) = W_{ji}^{(1)}(\textit{anterior}) + \eta \cdot \delta_j^{(1)} \cdot x_i \quad (2.14)$$

3. DESENVOLVIMENTO

Neste capítulo serão abordadas as etapas de desenvolvimento do trabalho, detalhando os materiais e métodos utilizados para o estudo da rede neural Perceptron Multicamadas executando a tarefa de classificação de objetos.

3.1. Esquema geral

O sistema desenvolvido terá como objetivo classificar corretamente as peças apresentadas utilizando a rede PMC.

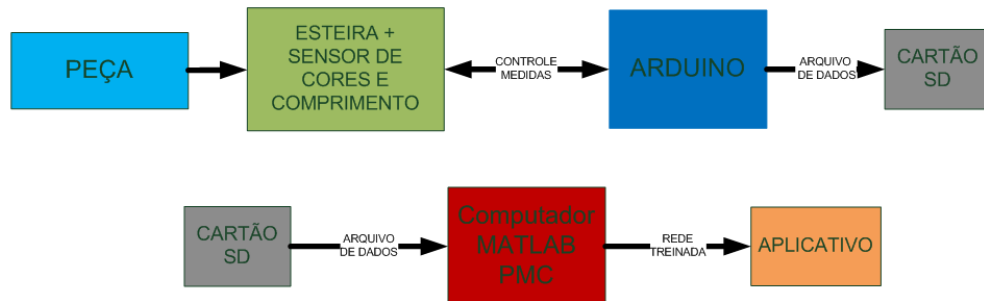
A peça a ser classificada deverá ser posicionada na esteira transportadora, que irá coletar os dados das cores e do comprimento da peça e transmitir tais dados para um computador que executará o processo de classificação utilizando a rede neural.

Para que esta tarefa seja possível, a rede precisa ser previamente treinada. Para este processo também será utilizada a esteira transportadora, onde amostras das diversas classes a serem treinadas serão analisadas pelos sensores de cor e tamanho. Os dados coletados serão armazenados para que posteriormente sejam fornecidos como exemplos para o processo de treinamento da rede.

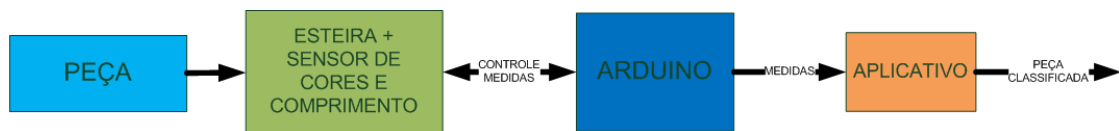
O esquema apresentado na figura 3.1 representa a sequência de tarefas a serem executadas durante o trabalho.

Figura 3.1 - Esquema de funcionamento do projeto

Processo para treinamento da rede neural



Processo para classificação de peças



3.2. Materiais utilizados

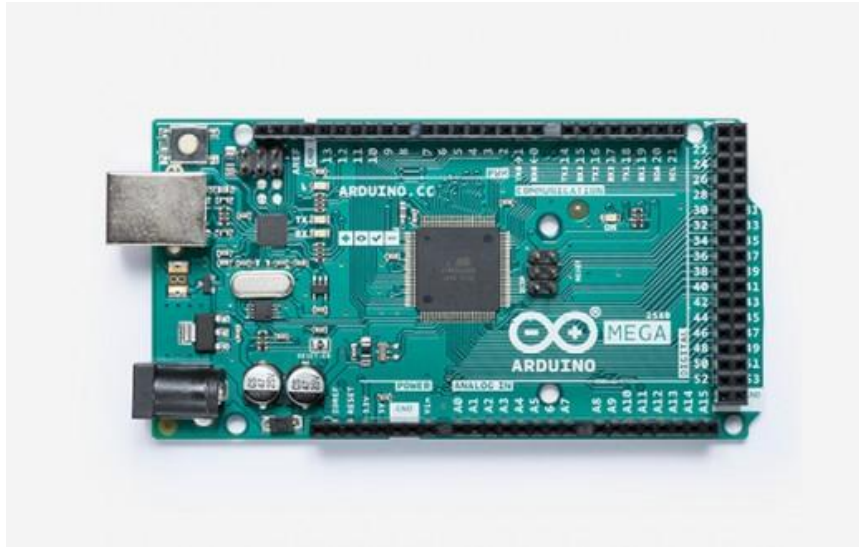
3.2.1. Plataforma Arduino

O Arduino é uma plataforma de prototipagem cujo desenvolvimento iniciou em 2005 na Itália e teve seu foco inicial direcionado a estudantes com baixa ou nenhuma fundamentação em eletrônica e programação.

A placa básica é chamada de Arduino UNO e é baseada no microcontrolador ATmega328P e possuindo 14 pinos de Entrada/Saídas digitais, dos quais 6 podem ser utilizadas como saídas PWM, além de 6 entradas analógicas.

Para projetos mais complexos há a possibilidade de utilização da placa Arduino Mega 2560, baseado no microcontrolador ATmega2560, ilustrado na figura 3.2, que possui 54 pinos de entrada ou saída digitais das quais 15 possuem funcionalidade PWM e 16 entradas analógicas, além de possuir mais capacidade de processamento.

Figura 3.2 - Arduino Mega 2560



Fonte:Arduino, 2018

A plataforma possibilita a utilização de módulos complementares em conjunto com o dispositivo básico chamados de “Shields”. Estes módulos adicionam funcionalidades ao sistema e permitem o desenvolvimento de projetos com as mais distintas funções, como exemplo a adição de portas Ethernet, entradas para cartões de memória interface de comunicação de fio, dentre outras.

A comunicação com o computador é feita por meio de uma porta USB, que permite a gravação dos programas desenvolvidos para a execução no microcontrolador.

A linguagem de desenvolvimento para o Arduino é baseada na linguagem C/C++ e pode ser implementada utilizando a Interface de Desenvolvimento de código livre denominada Software Arduino.

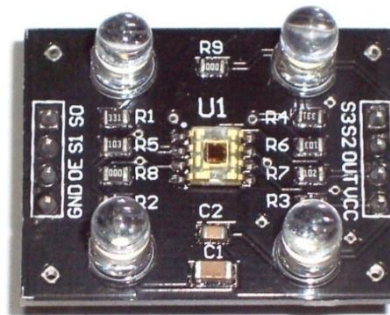
No trabalho a placa Arduino Mega 2560 é responsável pelo controle da esteira e da coleta dos dados por meio dos sensores utilizados.

Sensor de Cores TCS3200

O sensor de cor utilizado no trabalho é composto por um chip TAOS TCS3200 ilustrado na figura 3.3, capaz de medir e detectar uma vasta faixa de cores visíveis, convertendo um sinal luminoso em um sinal de frequência que pode ser lido por um microcontrolador, e quatro LEDs brancos, utilizados para iluminar o objeto que está sendo analisado.

O dispositivo trabalha com a leitura de uma matriz de fotodiodos formada por 8 linhas e 8 colunas divididos da seguinte forma: Dezesesseis fotodiodos com filtro vermelho, dezesesseis fotodiodos com filtro verde, dezesesseis fotodiodos com filtro azul e dezesesseis fotodiodos sem filtro.

Figura 3.3 - Módulo Sensor TCS3200



O módulo do sensor possui oito pinos, dois para a alimentação (GND e V_{dd}), dois para a seleção da escala de frequência da saída (S0 e S1), dois para a seleção do grupo de fotodiodo a ser lido (S2 e S3), um pino de saída da frequência lida (OUT) e um pino de habilitação (OE).

A frequência nominal de saída do sensor pode variar de 12 kHz a 20 kHz de acordo com o comprimento de onda e a intensidade de luz aplicada nos fotodiodos. O dispositivo permite que essa frequência de saída possa ser dividida de forma que se adéque melhor à aplicação desejada. Esta funcionalidade é acessada por meio dos pinos S0 e S1 e seguem a lógica de configuração listada na tabela 2.1.

Tabela 2.1 Configuração da escala de frequência

Nível lógico do pino		Escala da saída de frequência
S0	S1	
Baixo	Baixo	Desligado
Baixo	Alto	2%
Alto	Baixo	20%
Alto	Alto	100%

Fonte: Adaptado de TEXAS ADVANCED OPTOELETRONIC SOLUTIONS (2003)

A leitura da informação de cor é feita selecionando-se o grupo de diodo correspondente a componente de cor que se deseja observar, para isso os pinos S2 e S3 devem ser configurados de acordo com os parâmetros mostrados na tabela 2.2.

Tabela 2.2 - Configuração dos filtros dos fotodiodos

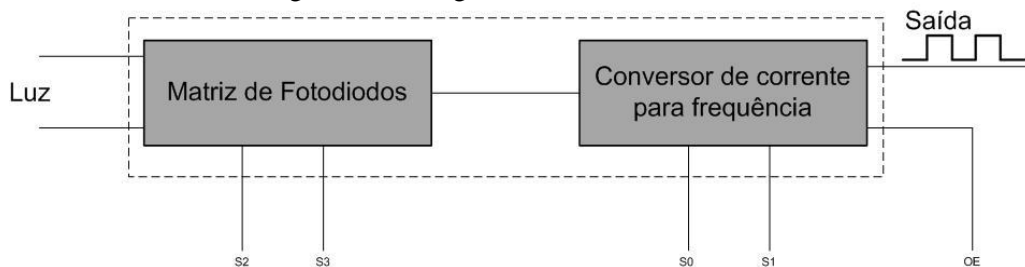
Nível lógico do pino		Tipo de fotodiodo selecionado
S1	S2	
Baixo	Baixo	Vermelho
Baixo	Alto	Azul
Alto	Baixo	Sem Filtro
Alto	Alto	Verde

Fonte : Adaptado de TEXAS ADVANCED OPTOELETRONIC SOLUTIONS (2003)

A frequência de saída dos canais vermelho, azul e verde representa uma proporção da respectiva cor em relação ao valor absoluto lido pelos fotodiodos sem filtro.

Sendo assim, o diagrama de blocos funcional pode ser representado na figura 3.4.

Figura 3.4 - Diagrama de blocos funcionais



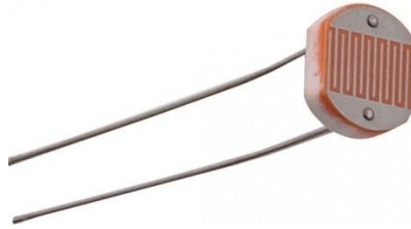
Fonte: Adaptado de TEXAS ADVANCED OPTOELETRONIC SOLUTIONS (2003)

No trabalho, o sensor TCS3200 tem a função de coletar os dados de cores que serão processados pela rede neural.

3.2.2. Resistor Dependente da Luz

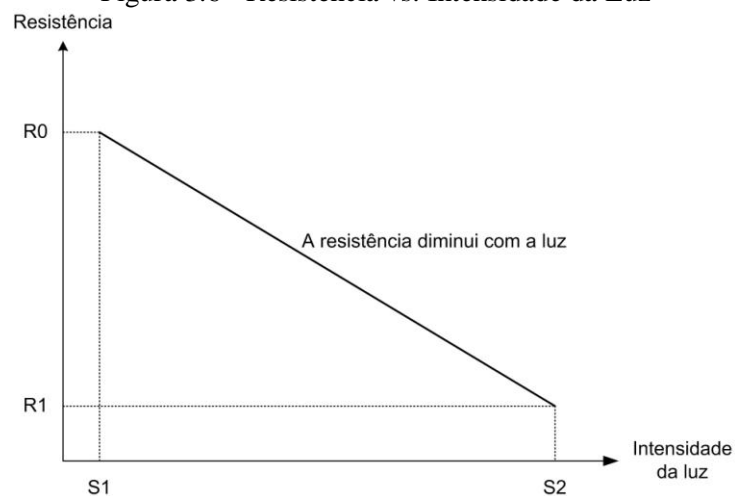
O Resistor Dependente da Luz (RDL) ilustrado na figura 3.5, é um dispositivo que é capaz de variar sua resistência elétrica de acordo com a intensidade de luz que incide sobre ele.

Figura 3.5 - Resistor Dependente da Luz



É comum que os dispositivos RDL sejam construídos utilizando as propriedades dos Sulfeto de Cádmio. Thomazini e Albuquerque (2007) explicam que o sulfeto de Cádmio “apresenta uma resistência extremamente elevada no escuro, da ordem de milhões de ohms. Essa resistência é diminuída para algumas centenas de milhares de ohms quando recebe iluminação direta”, desta forma fica claro que a resistência no RDL é inversamente proporcional à intensidade luminosa, esta característica é ilustrada na figura 3,6.

Figura 3.6 - Resistência vs. Intensidade da Luz



Fonte Adaptado de THOMAZINI e ALBUQUERQUE (2007)

3.2.3. Diodo Laser

A palavra LASER é um acrônimo de *Light Amplification by Stimulated Emission of Radiation*, ou em português, Amplificação da luz por emissão estimulada de radiação (SOUSA, 2011).

O diodo LASER, ilustrado na figura 3.7, é um componente eletrônico semiconductor, ele é formado por uma junção PN semelhante ao encontrado em um diodo de emissão de luz (LED) (DIODOS, 2018).

Figura 3.7 - Diodo LASER



No mercado existem diversas configurações disponíveis de diodos LASER com variadas opções de comprimento de onda e potência que podem ser empregados nas mais diversas aplicações desde a leitura de discos ópticos até a corte de peças de metal.

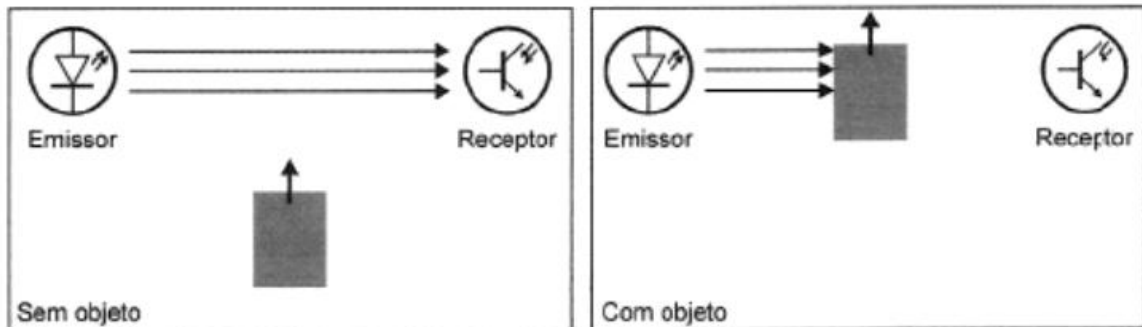
A característica fundamental do diodo de LASER aproveitada neste trabalho é sua capacidade de emitir um feixe concentrado de luz, sendo possível direcioná-lo para um ponto específico. Esta característica em conjunto com as propriedades do RDL possibilitou a construção do um sensor de presença utilizado no projeto, cujas características são descritas a seguir.

3.2.4. Sensor de Barreira

As características do RDL e do Diodo Laser permitem a utilização desses componentes na construção de um dispositivo capaz de detectar a presença de um objeto que se posicione entre a superfície sensível a luz do RDL e o feixe luminoso emitido pelo Diodo Laser.

Este tipo de dispositivo é chamado por Thomazini e Albuquerque (2007), de “Sensor óptico por detecção de barreira de luz”, e o define como um sensor que “possui o emissor e receptor montados em dispositivos separados. Ao serem alinhados, os dois componentes criam entre si uma barreira de luz. A presença de um objeto interrompendo essa barreira faz com que o sensor seja ativado”. O funcionamento deste sensor é ilustrado na figura 3.8.

Figura 3.8 - Sensor de barreira de luz



Fonte: THOMAZINI e ALBUQUERQUE (2007)

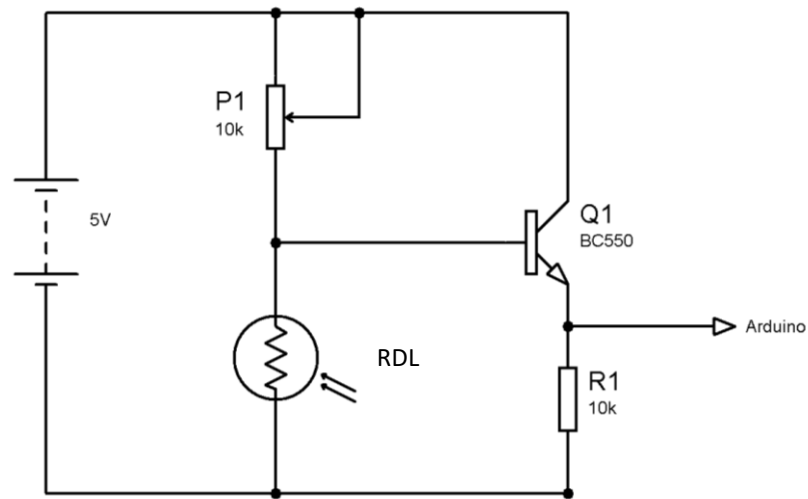
Desta forma, pode-se notar que o Diodo Laser funciona como o emissor e o RDL o receptor.

Ao se trabalhar com microcontroladores é interessante que determinados sensores disponibilizem saídas digitais, ou seja, que seja possível classificar o sinal de saída com uma lógica binária que o identifique como nível alto (1), ou nível baixo (0). No caso do sensor de barreira a ideia é que enquanto não haja objeto interrompendo o feixe um sinal de nível baixo seja apresentado ao microcontrolador e quando um objeto for detectado, o sinal de nível alto seja enviado.

A funcionalidade de envio da informação para o microcontrolador é executada pela área do sensor correspondente ao receptor, mas apenas o RDL não é capaz de realizar essa tarefa da forma que foi proposta, por isso se faz necessária a montagem de um circuito auxiliar, que permitirá a análise do sinal detectado pelo RDL de forma digital.

O circuito do sensor de barreira pode ser observado na figura 3.9.

Figura 3.9 - Circuito receptor do sensor de barreira



O princípio de funcionamento do circuito é baseado no divisor de tensão formado pelo potenciômetro P1 e o RDL, e o transistor BC550.

O BC550 é um transistor de junção NPN de propósito geral que pode operar em modo amplificar ou de chaveamento. Neste circuito o transistor funciona como uma chave que deve ser acionada quando a luminosidade que incide sobre o RDL está abaixo de uma determinada faixa.

Pelo esquema apresentado na figura 2.14 é possível observar que a tensão aplicada na base do transistor é a tensão presente sobre o RDL, ou seja, é proporcional à resistência do RDL. Como a resistência neste dispositivo é inversamente proporcional à luminosidade, quando o feixe do Diodo Laser estiver sobre o RDL, este possuirá uma baixa resistência, logo a tensão aplicada na base do transistor também será baixa e ele estará em modo de corte. Desta forma não haverá corrente sobre o resistor R1 e, portanto, a saída apresentará sinal lógico baixo.

Por outro lado, quando algum objeto interromper o feixe de luz que incide sobre o RDL a resistência do componente será aumentada, e conseqüentemente a tensão aplicada na base do transistor que passará para o modo de saturação, fazendo com que haja uma corrente passando pelo resistor R1, levando a saída para um nível lógico alto.

Como a resistência do RDL é afetada tanto pela luz emitida pelo Diodo Laser quando pela luz ambiente, o potenciômetro P1 permite o ajuste da faixa de luminosidade em que o sensor entra em operação compensando a interferência da luz ambiente no sensor.

No trabalho este sensor tem a função de identificar a presença da peça para que o sensor de cor possa fazer a leitura, e ainda, analisando o tempo que uma peça leva para cruzar totalmente o feixe de luz, é possível obter uma leitura correspondente ao comprimento da peça.

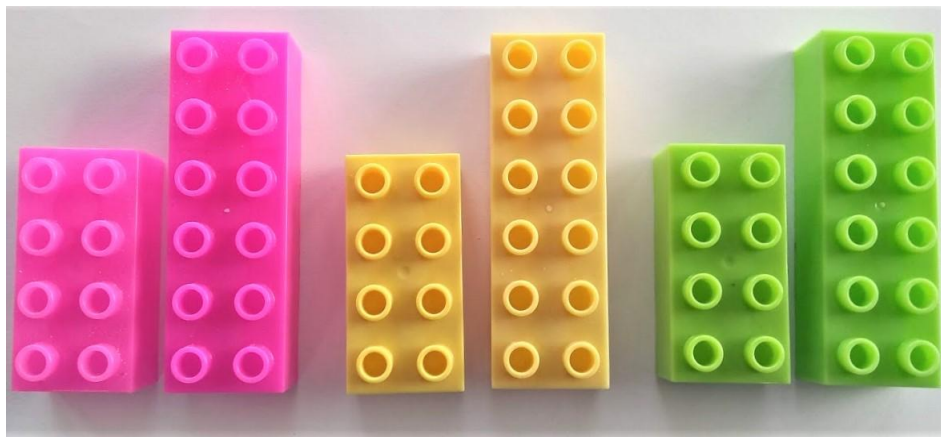
3.2.5. Peças para classificação

Por se tratar de um método de treinamento supervisionado, as classes a serem identificadas pela rede PMC devem ter suas características bem definidas. Tais características serão os parâmetros de entrada para a rede.

Para este trabalho foram escolhidos como parâmetros de diferenciação de classes a cor e o comprimento de objetos, para isso foram selecionadas peças plásticas para representar cada classe.

Foram selecionadas peças de três cores diferentes e dois tamanhos distintos totalizando seis classes possíveis, tais peças podem ser observadas na figura 3.10.

Figura 3.10 - Peças plásticas representando classes



3.2.6. Esteira transportadora

As esteiras transportadoras são equipamentos para indústrias feitos para manuseio, transporte e movimentação de cargas dentro de um determinado estabelecimento (KAUFMANN,2019).

Com o intuito de facilitar o processo de coleta de dados e classificação das peças foi realizado o projeto e a confecção de uma esteira transportadora capaz de acomodar de forma modular os sensores e atuadores utilizados no trabalho.

A estrutura da esteira foi projetada com o auxílio do software AutoDesk AutoCAD, onde foram realizados os desenhos que posteriormente serviram de molde para o corte a LASER das peças em placas de MDF. Detalhes construtivos da esteira podem ser observados nas figuras 3.11 em que é mostrada a vista lateral da esteira e na figura 3.12 em que é exibida a estrutura do suporte para sensor desenvolvido.

Figura 3.11 Vista lateral da esteira

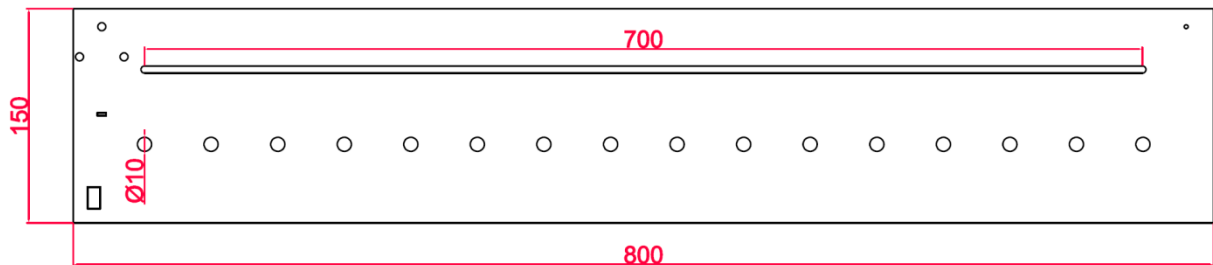
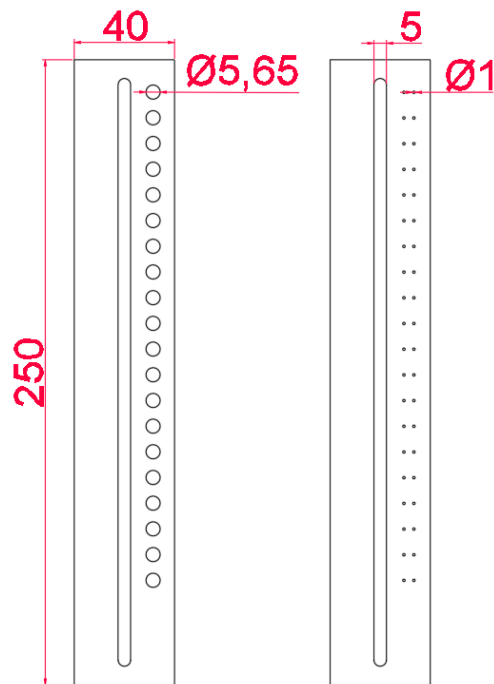


Figura 3.12 - Suporte para sensores



A correia transportadora da esteira foi recortada em lona de vinil, material flexível e de fácil manuseio geralmente usado na confecção de cartazes.

Para a movimentação do dispositivo foi utilizado um motor de corrente contínua de 5V acoplado a uma caixa de redução modelo 70168 do fabricante Tamiya. Este modelo de caixa de redução possibilita a conexão de dois motores e permite a configuração de engrenagens para uma redução de rotação de 58:1 ou 203:1. Neste projeto foi utilizado apenas um dos motores disponíveis na configuração de 58:1, que se mostrou suficiente para fornecer velocidade e torque adequados para o conjunto.

A figura 3.13 apresenta a estrutura da esteira montada.

Figura 3.13 - Esteira transportadora



3.3. Coleta de Dados

Conforme mencionado, neste trabalho serão analisadas peças plásticas cujas características de cor e comprimento, servirão de parâmetros de entrada para a rede. A seguir são descritos os processos executados para a coleta dos dados das características das peças.

3.3.1. Cor da peça

Conforme apresentado no item 3.2.2, o sensor TCS3200 utiliza uma matriz de fotodiodos portadores de filtros vermelhos, verdes e azuis, sendo assim capaz de identificar a participação de cada uma das cores primárias na formação da cor da peça. Desta forma para se obter os dados sobre a coloração da peça é necessário a realização da leitura em separado de cada componente de cor, resultando em três valores correspondentes aos canais vermelho, verde azul.

O sensor de cor funciona como um conversor de cor para frequência, ou seja, a leitura de um canal de cor é representada na saída como um pulso cuja frequência é proporcional à leitura de luz feita pelo fotodiodo. O dispositivo é capaz de gerar um pulso com a frequência máxima de 500 KHz para representar um componente de cor, mas é possível configurar o sensor para que a saída tenha pulsos com frequências máximas correspondentes a 2%, 20% ou 100% da frequência máxima disponível.

Para este trabalho, foram realizados testes com as frequências disponíveis, onde pode ser observado que a escala de frequência de 2% fornecia valores suficientemente significativos

para as leituras das peças a serem classificadas, resultando em pulsos com frequências entre 1.5 KHz e 3.3Khz aproximadamente.

3.3.2. Comprimento da peça

Para a coleta dos dados referentes ao comprimento de cada peça foi utilizado o sensor de barreira.

Como a velocidade de deslocamento da peça na esteira transportadora depende da velocidade do motor, como a tensão neste componente é mantida constante, a velocidade da esteira pode também ser considerada constante.

Com a utilização do sensor de barreira é possível determinar uma medida proporcional ao comprimento da peça calculando o tempo decorrido entre a interrupção do feixe de luz pela peça e a liberação deste feixe.

Para peças com tamanhos compatíveis a medida de tempo também será compatível, desta forma, a medida de tempo em milissegundos que o sensor de barreira fica ativado é suficiente para classificar uma peça de acordo com seu comprimento.

3.3.3. Sistema coletor de dados

Para a coleta dos dados das peças foi desenvolvido utilizando a plataforma Arduino um sistema com a finalidade de obter e armazenar as leituras referentes às cores e comprimentos das peças para que posteriormente tais dados fossem utilizados no treinamento da rede neural.

A primeira etapa no desenvolvimento do sistema consistiu em determinar os requisitos que deveriam ser cumpridos para a obtenção dos dados de forma otimizada para a execução do trabalho, desta forma, foi definido que o sistema deveria ser capaz de:

1. Controlar a movimentação da esteira transportadora
2. Identificar a presença de um objeto posicionado abaixo do sensor de cores
3. Obter os dados dos canais vermelho, verde e azul da cor da peça posicionada no sensor
4. Obter o tempo de interrupção do feixe de LASER pela peça
5. Armazenar os dados lidos de várias peças de forma que pudessem ser utilizados no treinamento da rede neural.

Para uma melhor organização do código fonte do sistema foi adotada a utilização de alguns conceitos de Programação Orientada a Objetos. Farinelli (2007) define programação orientada a objetos (POO) como sendo a proposta de “[...] representar o mais fielmente possível as situações do mundo real nos sistemas computacionais”. Seguindo esta abordagem foram adotados durante o desenvolvimento do sistema coletor de dados alguns conceitos relacionados a POO: o conceito de Objeto, que é a representação de um elemento do mundo real dentro do software; o conceito de abstração atribui ao objeto representado suas características e ações; e o conceito de Encapsulamento que determina que o código deve ser transparente para o usuário não sendo necessária a completa compreensão do código para que uma ação seja executada.

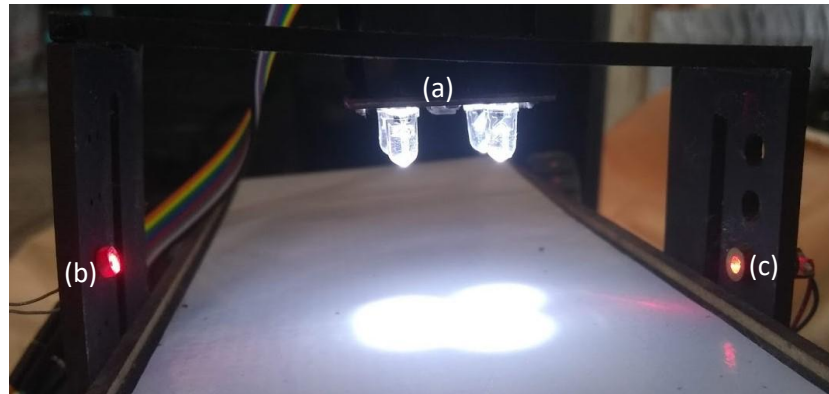
Desta forma, foram desenvolvidas classes para a representação do motor e dos sensores dentro do sistema.

Para o cumprimento do primeiro requisito do sistema foi utilizada a classe “Motor”, que tem como objetivo controlar o acionamento do motor que movimenta a esteira, para isso foram implementados os métodos “SetPWM”, responsável por configurar o valor relativo a modulação por largura de pulso que será fornecida pelo Arduino, “Liga” que aciona o motor e “Desliga” que interrompe o funcionamento do motor. Desta forma, o ato de ligar ou desligar a movimentação da esteira é realizada de forma mais clara e intuitiva com a utilização dos comandos “Liga” ou “Desliga” ao invés dos métodos convencionais de escrita em portas PWM do Arduino. O código que executa a classe “Motor” pode ser observado no apêndice A.

Para o segundo requisito foi utilizado o sensor de barreira, que montado com o auxílio do suporte para sensores desenvolvido, fica posicionado de forma que a peça ao se posicionar embaixo do sensor de cores interrompe o feixe do sensor de barreira, identificando assim que ela está na posição adequada para a leitura. A Montagem do sensor de barreira e do sensor de cores pode ser observada na figura 3.14.

Figura 3.14 - Sensor de Cores e Sensor de Barreira

(a) Sensor de Cores (b) receptor do sensor de barreira (c) Emissor do sensor de barreira



Com a detecção da peça posicionada abaixo do sensor, a movimentação da esteira é interrompida e é feita a aquisição dos dados correspondente a cor da peça. Para esta leitura foi desenvolvida a classe “ColorSensor” que possui os métodos “GetRed”, “GetGreen” e “GetBlue” que retornam os valores das leituras dos fotodiodos do sensor de cor com os filtros vermelho, verde e azul respectivamente satisfazendo assim o terceiro requisito do sistema. O apêndice B apresenta o código desenvolvido para a classe “ColorSensor”.

Após a leitura da cor da peça, a movimentação da esteira é retomada e o tempo decorrido até que a peça atravesse completamente o feixe do sensor de barreira é medido, atendendo assim o quarto requisito.

Por fim os dados coletados das cores da peça e seu respectivo comprimento precisam ser armazenados, para isto optou-se por utilizar um cartão de memória flash do tipo *Secure Digital* (SD). O cartão de SD é um padrão desenvolvido pela SD Association, fundada pela Panasonic, SanDisk e Toshiba em Janeiro de 2000, o padrão se tornou um dos mais populares para o armazenamento de dados em dispositivos portáteis principalmente nas versões mini SD e micro SD (CARTÃO SANDISK, 2019). A Figura 3.15 mostra um cartão do tipo micro SD.

Figura 3.15 - Cartão micro SD



Fonte: CARTÃO SANDISK

A gravação dos dados coletados no Arduino é feita utilizando um módulo controlador Ethernet, que além de possibilitar a comunicação por rede possui um slot para conexão de cartões SD. Neste trabalho as funcionalidades de comunicação de rede do módulo não foram utilizadas.

Os dados lidos dos sensores são gravados no cartão SD em um arquivo no formato CSV. “Os arquivos CSV (do inglês "Character-separated values" ou "valores separados por um delimitador") servem para armazenar dados tabulares (números e texto) em texto simples” (PIRES, 2015).

Desta forma, cada componente de cor corresponde a um campo no arquivo, seguido pelo valor de leitura do comprimento da peça. Cada linha do arquivo corresponde a uma peça amostrada, a figura 3.16 apresenta o formato dos dados gravados no arquivo.

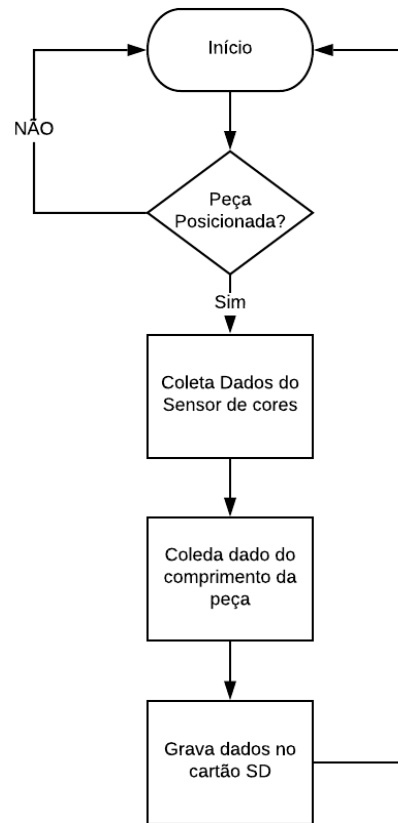
Figura 3.16 - exemplo de dados coletados

	A	B	C	D
1	292	459	460	699
2	271	449	438	698
3	271	452	444	634
4	270	449	453	549

Fonte: print screen da aplicação Microsoft Excel 2007

O fluxograma apresentado na figura 3.17 representa o funcionamento do sistema de coleta de dados, cujo código pode ser observado no apêndice C.

Figura 3.17 - Fluxograma de funcionamento do sistema coletor de dados



4. RESULTADOS

4.1. Treinamento da Rede Perceptron Multicamadas

Após a obtenção dos dados das peças plásticas que representarão as classes a serem selecionadas pela rede, deu-se início ao desenvolvimento dos algoritmos que executam o treinamento e validação da rede PMC.

A primeira etapa deste processo consistiu em ajustar o conjunto dados de forma que fosse possível apresentá-los como amostras para o algoritmo, logo em seguida o código capaz de executar o processo de treinamento da rede foi desenvolvido, os dados foram analisados pelo algoritmo e por fim foram realizados testes, adequando os parâmetros da rede de forma que fosse alcançado um resultado satisfatório.

A seguir são detalhadas as etapas que resultaram no treinamento da rede PMC.

4.1.1. Ajuste dos dados das amostras

Como demonstrado anteriormente, os dados coletados pelo sensor de cor e pelo sensor de barreira foram armazenados em um cartão SD em formato de arquivo CSV. Para que estes dados pudessem ser utilizados como amostras para o treinamento supervisionado, há a necessidade de identificar a qual classe um determinado grupo de dados pertence.

Para que fosse possível identificar a quais classes os dados são pertencentes, a leitura de cada peça foi feita de forma independente, resultando em um arquivo para cada classe, ou seja, foram obtidos seis arquivos, cada um contendo diversas leituras dos dados de uma das classes.

Durante o processo de treinamento, além dos dados da amostra, a rede precisa receber também a informação que descreva a resposta desejada para cada amostra analisada. A rede proposta deve ser capaz de classificar amostras como pertencente a uma das seis possíveis classes, dessa forma, neste trabalho foi adotado que a camada de saída da rede deve possuir seis neurônios, cada um representando uma classe. Sendo assim, para uma determinada amostra, apenas um neurônio na camada de saída deve ser ativado. Este estado foi

representado no arquivo de dados na forma de seis campos em que o que representa o neurônio ativo contém o valor “1” e os demais são preenchidos com o valor “0”. A figura 3.18 apresenta uma amostra contendo os dados lidos pelos sensores e os dados referentes à identificação da classe.

Figura 3.18 - Grupo com dados dos sensores e identificação da classe

Dados dos Sensores				Representação da Classe					
275	452	451	531	1	0	0	0	0	0

Após a inclusão da identificação das classes os seis arquivos foram agrupados em um único arquivo contendo todas as informações das amostras de todas as classes.

Segundo Silva et al. (2016) para melhorar o desempenho computacional do treinamento técnicas que normalizam os dados de entrada podem ser usados, ajustando os valores de acordo com os limites numéricos produzidos pela função de ativação utilizada. No caso deste trabalho a função utilizada deverá representar os resultados no intervalo entre 0 e 1, que foram os valores escolhidos para identificar uma amostra como não pertencente ou pertencente a uma determinada classe, desta forma, os dados das amostras contidos no arquivo foram ajustadas para que fossem compatíveis com estes valores.

Para normalizar os dados das amostras foram utilizados recursos do software Microsoft Excel, mais especificamente a função “PADRONIZAR” que segundo o manual do software “Retorna um valor normalizado de uma distribuição caracterizada por média e desvio padrão” (MICROSOFT, 2019). Os parâmetros que são passados para a função são o valor que se deseja normalizar, a média aritmética dos dados e o desvio padrão do conjunto de dados analisados, desta forma a sintaxe para a função é apresentada da seguinte forma: PADRONIZAR(x, média, desv_padrao) onde x representa o valor que deve ser padronizado. O valor retornado pela função pode ser obtido por meio da equação 3.1, onde Z é o valor padronizado, X o valor que deve ser padronizado, μ o valor da média e σ representa o valor do desvio padrão.

$$Z = \frac{X - \mu}{\sigma} \quad (3.1)$$

A função de normalização foi aplicada nos dados das leituras dos sensores de todas as amostras, como a função poderia retornar valores negativos, os dados finais correspondem ao módulo do valor retornado.

O resultado final dos ajustes dos dados foi um arquivo em formato CSV contendo dados normalizados de todas as amostras lidas pelos sensores e a respectiva representação da classe. Este arquivo é utilizado como entrada para rede nos processos de treinamento.

4.1.2. Algoritmo Perceptron Multicamadas

O desenvolvimento do algoritmo capaz de executar os procedimentos de treinamento e execução da rede PMC precisou levar em conta requisitos específicos necessários para a realização deste trabalho.

Neste ponto do trabalho ainda não se tem definidas as características construtivas da rede, apesar de já se ter especificado que a camada de saída da rede deve conter seis neurônios correspondentes a cada uma das possíveis classes, ainda não se tem estabelecidos o número total de camadas e conseqüentemente o número de neurônios presente em cada uma delas.

Desta forma o algoritmo foi desenvolvido com a capacidade de executar uma rede PMC composta por qualquer número de entradas, camadas e neurônios, permitindo que testes fossem realizados de forma ágil e diversificada.

O código foi desenvolvido utilizando o software MathWorks Matlab. A seguir são descritas as etapas de execução do algoritmo.

As dimensões da rede no código desenvolvido são informadas por meio de um vetor, em que o número de elementos representa o número e camadas da rede, e o valor de cada elemento do vetor corresponde ao número de neurônios em cada camada, sendo assim, um vetor com a configuração “[10 5]”, representa uma rede formada por duas camadas, sendo a primeira formada por dez neurônios e a segunda formada por cinco neurônios.

Os dados de entrada para a rede são lidos do arquivos de dados por meio do comando “csvread” do Matlab, que recebe como parâmetro o caminho e nome do arquivo a ser carregado e retorna uma matriz contendo os dados lidos que, no caso das amostras lidas, é

formado por dez colunas, correspondentes às leituras dos sensores e a identificação das classes, o número de linhas será equivalente ao número de amostras existentes no arquivo.

O script a seguir faz a divisão dos dados em duas variáveis, a primeira contendo apenas as quatro primeiras colunas da matriz, correspondente aos dados dos sensores e a segunda contendo as seis colunas restantes contendo os dados de identificação de cada amostra. Essas variáveis foram denominadas como “X” e “D” respectivamente. Em seguida para cada amostra presente na variável “X” é acrescentada mais um campo com o valor “1” que corresponde a entrada de limiar de ativação para camada de entrada.

O script foi configurado como arquivo de função do Matlab e recebe como parâmetros de entrada o vetor com as dimensões da rede, a matriz com os dados das amostras, a matriz contendo as classificações das amostras, a taxa de aprendizagem, o erro mínimo desejado e o número máximo de iterações permitidas. A função retorna uma célula contendo as matrizes dos pesos sinápticos, um vetor com a evolução do erro quadrático médio em cada iteração e o tempo total de execução do treinamento.

O apêndice D apresenta o código desenvolvido no Matlab, que inicialmente faz a inicialização da rede, nesta etapa são gerados pesos aleatórios para cada entrada dos neurônios em todas as camadas da rede. São inicializadas também as variáveis que receberão aos valores das entradas para cada neurônio e suas respectivas saídas.

A variável que armazena os pesos sinápticos dos neurônios é uma estrutura de dados do Matlab do tipo “cell” que pode ser representada como uma matriz de matrizes. No caso, a variável dos pesos será um vetor de matrizes em que cada elemento representa os pesos sinápticos das entradas dos neurônios em uma determinada camada. Sendo assim, cada matriz em um determinado elemento do vetor terá o número de colunas correspondente ao número de neurônios existentes na camada representada pelo elemento e o número de linhas será o número de neurônios presentes na camada anterior acrescido de um valor, correspondente aos pesos do limiar de ativação para esta camada. No caso da camada de entrada a matriz possuirá o número de linhas correspondente ao número de dados de entrada presentes na amostra. Como exemplo, em uma rede com duas camadas formadas por 10 e 5 neurônios respectivamente, em que os dados de entrada possuem 3 elementos a variável que representará os pesos sinápticos dos neurônios será formada por dois elementos, em que o primeiro será uma matriz 3×10 e o segundo uma matriz 11×5 .

Alguns parâmetros para o funcionamento do algoritmo devem ser informados pelo usuário, dentre eles o valor para a Taxa de Aprendizagem da rede. A taxa de aprendizagem é um parâmetro que determina o quão rápido a rede aprende, e geralmente é representada por um valor entre 0,1 e 1. Um valor muito baixo para a taxa de aprendizagem fará com que a rede leve um tempo maior para convergir, por outro lado um valor muito alto para este parâmetro pode fazer com que a rede oscile e não se estabilize no em um valor aceitável. No algoritmo desenvolvido a taxa de aprendizagem é uma variável representada pela letra “ η ”.

Outro parâmetro importante para a execução do código diz respeito às condições de parada para o algoritmo. Para este trabalho foram definidas duas possíveis condições de paradas para o processamento, a primeira diz respeito ao valor do erro encontrado entre as respostas dadas pela rede e as respostas desejadas, o segundo parâmetro correspondem ao número de épocas máximas que podem ser executadas, visando parar a execução caso a rede esteja levando um tempo excessivo para convergir.

Para medir o desempenho da rede frente ao erro apresentado pela saída em relação a desejada foi escolhida a função erro quadrático que é representada pela equação 3.2

$$E(k) = \frac{1}{2} \sum_j^n (d_j(k) - Y_j(k))^2 \quad (3.2)$$

Onde n representa o número de neurônios na camada de saída, $d_j(k)$ o valor desejado para a saída do neurônio j para a amostra k e $Y_j(k)$ o valor apresentado na saída do neurônio j para a amostra k .

Desta forma, o desempenho geral da rede pode ser medido pelo erro quadrático médio apresentado para um conjunto de amostras. O erro quadrático médio para um conjunto de amostras pode ser calculado pela equação 3.3 onde a variável p representa o número de amostras presentes no conjunto.

$$E_M = \frac{1}{p} \sum_{k=1}^p E(k) \quad (3.3)$$

Sendo assim, o critério de parada para o algoritmo é verificado comparando-se a variação do erro quadrático médio apresentado pela rede na iteração atual com o registrado na iteração anterior. O algoritmo executa iterações consecutivas enquanto a diferença absoluta entre os

erros anterior e atual seja maior que um parâmetro desejado. Esta condição é expressa matematicamente na equação 3.4

$$|E_M^{atual} - E_M^{anterior}| \geq \varepsilon \quad (3.4)$$

Onde ε é o erro mínimo desejado.

Com os parâmetros iniciais para a rede definidos, dá-se início ao processo de iterações que resultará no treinamento da rede.

O processo se inicia calculando o erro para o estado atual da rede. Para facilitar o cálculo e deixar o código mais inteligível, foi desenvolvida uma função que retorna o erro quadrático médio para um determinado conjunto de amostras analisados em relação aos pesos sinápticos apresentados.

Como citado anteriormente o processo de treinamento da rede PMC pode ser dividido em duas etapas, a primeira *forward* em que a resposta da rede é obtida considerando os pesos atuais e a segunda *backward* em que os pesos sinápticos são atualizados.

Desta forma, cada iteração do processo de treinamento da rede consiste em um primeiro momento obter a resposta da rede para uma determinada amostra, para isso foi desenvolvida a função “forward” no Matlab que processa uma dada amostra, fornecida em formato de um vetor, multiplicando-a pelo vetor de pesos sinápticos correspondente a cada neurônio na camada de entrada. Este resultado servirá como entrada para a camada seguinte que executará a mesma operação e assim consecutivamente até a camada de saída que apresentará a resposta da rede, concluindo a primeira etapa da iteração.

A segunda etapa se inicia com o cálculo para o valor da variável “delta” para a camada de saída, que representa o valor do gradiente local para esta camada em relação a resposta dada pela rede e o valor desejado.

Com a obtenção do valor da variável delta obtido para camada de saída, os pesos sinápticos dessa camada são atualizados.

Em seguida os gradientes locais das camadas intermediárias são calculados e os pesos sinápticos correspondentes atualizados levando em consideração as saídas das camadas posteriores, até a camada de entrada onde os valores fornecidos pela amostra são considerados no cálculo, finalizando a segunda e última etapa da iteração.

Este processo é repetido para cada uma das amostras contidas no conjunto de exemplos. Em seguida o erro quadrático médio é calculado novamente e comparado com o erro calculado no início da iteração, caso relação apresentada na equação 3.4 apresente valor verdadeiro é iniciada uma nova iteração continuamente até que a relação retorne um resultado falso ou que o número máximo de iterações seja alcançado.

4.2. Estrutura e parâmetros da rede PMC

Após o desenvolvimento do código capaz de executar o treinamento de uma rede PMC, foi necessário analisar as características construtivas da rede, ou seja, o número de camadas e o número de neurônios em cada uma delas.

Alem da estrutura também se faz necessário a configuração de outros parâmetros que podem interferir diretamente no desempenho da rede, tais como a taxa de aprendizagem e a mínima variação do erro quadrático médio entre duas iterações consecutivas desejada.

A seguir são descritos os procedimentos adotados para a obtenção de uma estrutura de rede PMC e seus respectivos parâmetros que pudessem cumprir de forma satisfatória as tarefas propostas neste trabalho.

4.2.1. Definição da estrutura da rede PMC

A estratégia adotada para a definição da melhor estrutura para a rede PMC necessária foi a utilização do método de validação cruzada.

Neste método a etapa inicial consiste em definir um conjunto de topologias diferentes para redes PMC intitulado de tipologias candidatas.

O propósito de um teste de validação cruzada é analisar o desempenho de cada topologia candidata quando a rede é aplicada em um conjunto diferente do utilizado durante a etapa de treinamento (SILVA et al.,2016).

Neste trabalho foi adotado o método de validação cruzada por amostragem aleatória, em que o conjunto total de amostras coletadas é dividido aleatoriamente em dois grupos: o primeiro, um grupo de testes, utilizado durante o processo de treinamento da rede; e o segundo grupo, denominado grupo de validação, que é responsável por avaliar o desempenho da rede treinada.

Para a execução dos testes de validação cruzada foi desenvolvido um novo script no Matlab capaz de utilizar a função de treinamento previamente desenvolvida para treinar um conjunto de topologias candidatas.

O código do script apresentado no apêndice E se inicia com a leitura dos dados das amostras e a separação dos grupos de teste e validação. A seguir são informadas as topologias das redes que serão testadas. Em seguida é realizado o processo de treinamento de cada uma das redes utilizando a função de treinamento.

Cada uma das redes avaliadas é utilizada para classificar os dados do conjunto de testes, e é feita a comparação das respostas dadas com as respectivas respostas desejadas, sendo possível calcular a taxa de acerto de cada uma das topologias candidatas.

Para esta etapa do trabalho foram coletadas cinquenta amostras de cada classe, totalizando trezentas amostras. Para o processo de validação cruzada foi estipulado que o grupo de testes seria composto por 60% dos dados, ou seja, 180 amostras, os 40% restante correspondente à 120 amostras foi utilizado no grupo de validação.

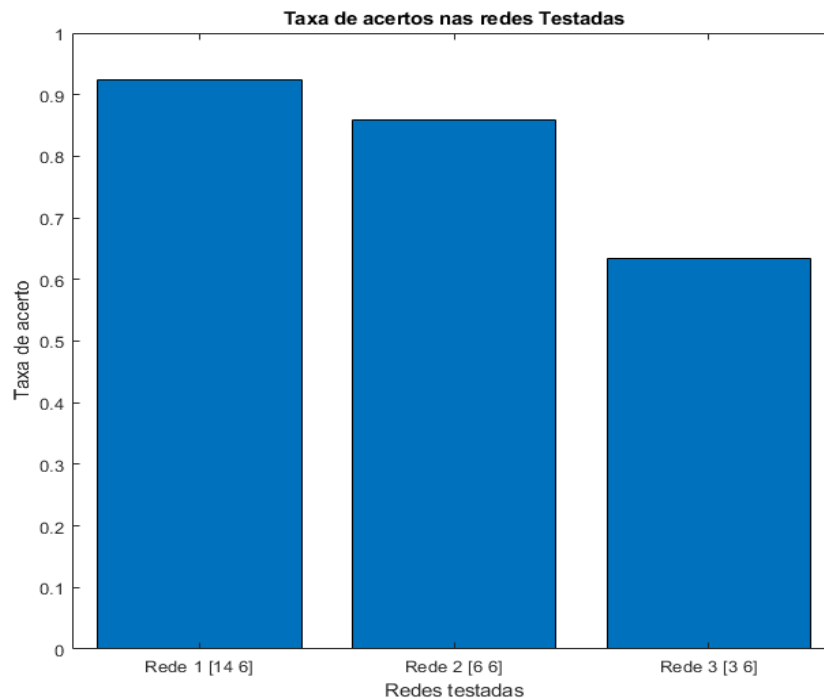
Em testes preliminares utilizando redes com números de camadas diferentes, foi observado que para o conjunto de dados utilizado, redes formadas por três camadas obtiveram resultados melhores durante o treinamento.

Desta forma, foram escolhidas a princípio três topologias candidatas formadas por duas camadas. A terceira camada obrigatoriamente deve ser composta por seis neurônios, já que deve ser equivalente ao número de classes presentes nas amostras, a segunda camada de neurônios foi configurada com 14, 6 e 3 neurônios em cada uma das topologias a serem

testadas. Para estes testes a taxa de aprendizagem foi fixada com o valor de 0.1, a variação mínima do erro quadrático médio em 10^{-8} e o número máximo de iterações em 10.000.

A figura 3.19 apresenta um gráfico onde pode ser observada a taxa de acerto apresentada por cada uma das três topologias testadas.

Figura 3.19 - Taxa de acerto das topologias candidatas



No gráfico é possível observar que a topologia composta por 14 neurônios na camada oculta, obteve uma taxa de acertos superior comparada às demais topologias, em valores percentuais a primeira topologia apresentou uma taxa de acertos de 92,5% enquanto a segunda apresentou 85,83% de acertos e a terceira uma taxa de 63,33% de peças classificadas corretamente.

Sendo assim, foi selecionada para os demais testes realizados neste trabalho a topologia formada por três camadas: a camada de entrada, onde os quatro valores das características das amostras são apresentados à rede, uma camada oculta composta por quatorze neurônios e seis neurônios na camada de saída.

4.2.2. Taxa de Aprendizagem

A taxa de aprendizagem, como já citado, influencia na velocidade com que a rede aprende mas pode alterar também o desempenho da rede.

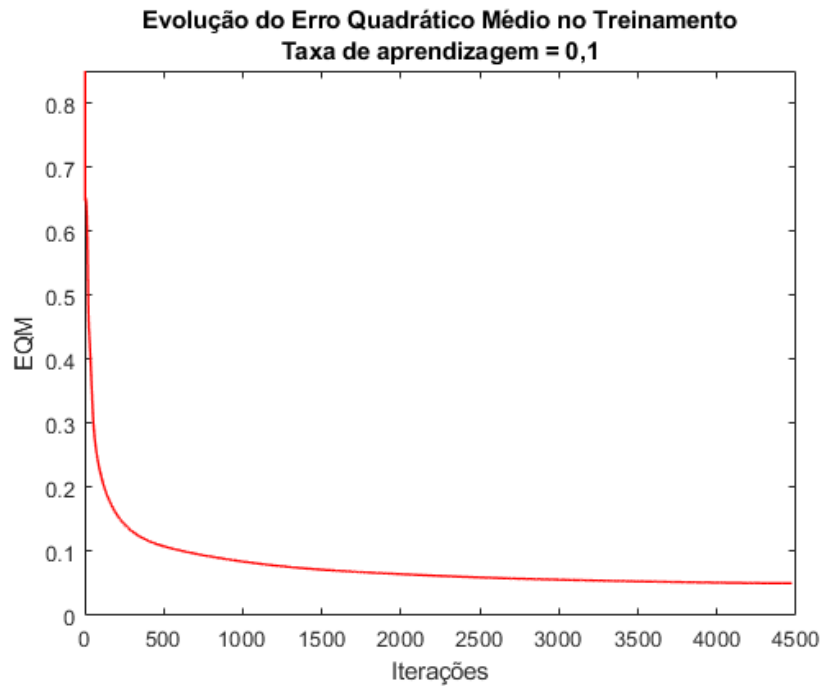
Com a topologia da rede definida foram realizados testes com taxas de aprendizagem distintas com o intuito de analisar o impacto deste parâmetro no processo de treinamento e consequentemente o desempenho da rede na classificação das peças.

Para estes testes a rede formada por 14 neurônios na camada oculta e 6 neurônios na camada de saída foi treinada repetidas vezes utilizando o mesmo conjunto de dados utilizado na validação cruzada. Neste teste os dados não foram separados em grupos, desta forma as trezentas amostras presentes no conjunto foram utilizadas para o treinamento da rede, estas mesmas amostras serviram para verificar a taxa de ajuste da rede.

Foram realizados treinamentos com as taxa de aprendizagem com valores de 0,1, 0,5 e 1. Os resultados foram avaliados quanto à evolução do erro quadrático médio, a taxa de acertos, o tempo de treinamento da rede e o erro ao final das iterações. A mínima variação do erro quadrático médio (EQM) entre duas iterações consecutivas foi configurado com o valor de 10^{-8} para todos os testes realizados e o número de iterações máximas foi configurado para dez mil.

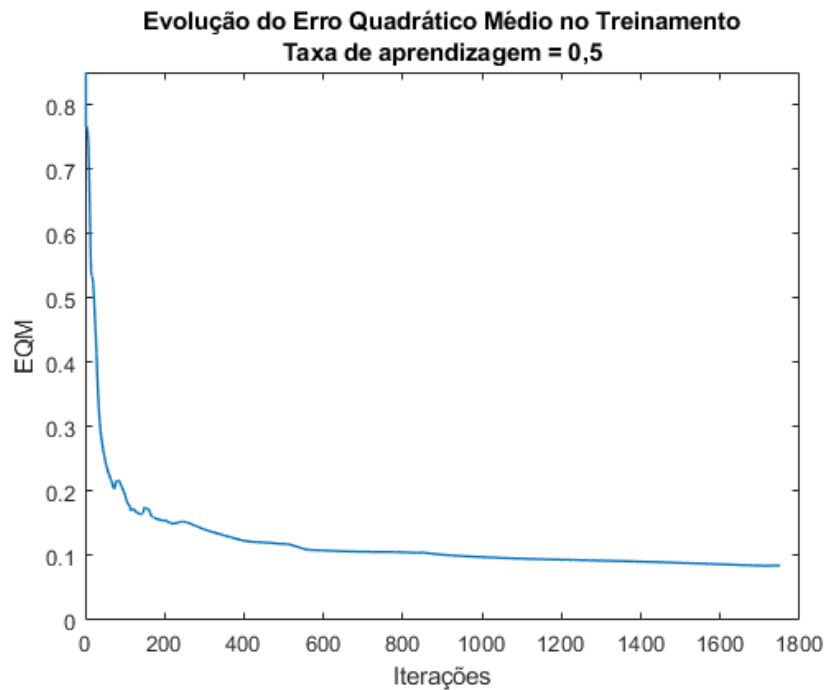
A figura 3.20 apresenta a evolução do EQM no treinamento executado com a taxa de aprendizagem com valor de 0,1. Neste teste a rede alcançou uma taxa de acertos de 93%, para isso foram executadas 4474 iterações em um tempo total de 757,19 segundos e o valor do erro ao final das iterações foi de 0,0503. No gráfico é possível observar que não houve oscilações no comportamento do EQM durante o treinamento, sendo este decrescente em todo o período.

Figura 3.20 - Evolução do EQM com valor de taxa de aprendizagem igual a 0,1



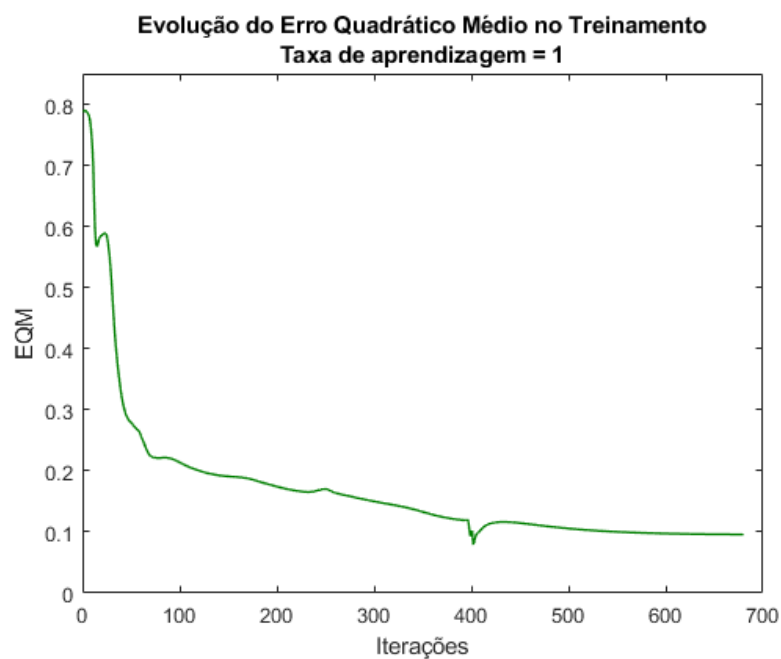
Na figura 3.21 pode ser observada a evolução do EQM no teste executado com taxa de aprendizagem de 0,5. Nestas configurações a rede apresentou uma taxa de acerto de 90%, foram executadas 1750 iterações em um tempo total de 198,14 segundos e o erro final encontrado foi de 0,084. Com a taxa de aprendizagem configurada com o valor de 0,5 já é possível observar oscilações nos valores dos erros. Em comparação com o teste realizado com a taxa de 0,1 houve uma queda considerável no número total de iterações, e consequentemente do tempo necessário para que a variação de erro mínima fosse alcançada, em contrapartida, houve uma deterioração do desempenho final da rede com a diminuição da taxa de acertos.

Figura 3.21 - Evolução do EQM com valor de taxa de aprendizagem igual a 0,5



A evolução do EQM na rede testada com taxa de aprendizagem com valor 1 é exibida na figura 3.22. Para estas condições a rede apresentou uma taxa de acerto de 88%, foram realizadas 680 iterações em 115,55 segundos, chegando a um erro final de 0,096. Neste caso as oscilações na evolução do erro são observáveis em vários pontos, novamente houve uma redução no número de iterações realizadas e mais uma vez houve deterioração da qualidade final comparada aos testes anteriores.

Figura 3.22 - Evolução do EQM com valor de taxa de aprendizagem igual a 1



Com os testes realizados ficou clara a influencia da taxa de aprendizagem na velocidade de convergência da rede, e também o possível impacto na qualidade do ajuste das respostas. Logo a definição deste parâmetro depende do tempo disponível para a obtenção dos resultados e da precisão dos resultados da rede na aplicação pretendida.

Neste trabalho um dos objetivos é fazer com que a rede classifique de forma correta o maior número de peças, para isso, é desejada uma taxa de acertos no treinamento o quão alta possível. Logo, a taxa de aprendizagem adotada é a que permite um melhor ajuste às amostras que, conforme os testes realizados, foi encontrado com o valor de taxa de aprendizagem igual a 0,1.

4.3. Validação da Rede

A partir dos testes executados no algoritmo da rede PMC foi possível definir a rede que teria melhores condições de atender os requisitos deste trabalho, desta forma, foi definida uma rede Perceptron Multicamadas com as seguintes configurações:

- Três camadas, sendo quatro valores na camada de entrada, quatorze neurônios na camada oculta e seis neurônios na segunda;
- Taxa de aprendizagem igual a 0,1;
- Variação mínima do Erro Quadrático média de 10^{-8} .
- Número máximo de iterações de 10.000.

No processo de validação da rede é executada somente a etapa *forward* do algoritmo PMC, resultado assim na classificação da peça de acordo com o modelo obtido no processo de treinamento.

Com a estrutura final da rede definida foi realizado o processo de coleta de dados para o treinamento e obtenção dos pesos sinápticos que serão utilizados no processo de validação.

Foram coletadas cinquenta amostras de cada uma das peças a ser classificada, totalizando trezentas amostras. Este conjunto de dados foi apresentado ao algoritmo de treinamento em que 60% das amostras foram utilizadas como grupo de testes e 40% como grupo de validação. Este treinamento resultou em uma rede que classificou corretamente 92,3% das amostras do grupo de validação.

A rede obtida foi selecionada para a validação em tempo real do classificador.

4.3.1. Aplicativo para classificação das peças

Para a execução do processo de validação em tempo real da rede e apresentação dos resultados foi desenvolvido um aplicativo capaz de receber os dados de leitura da peça feita pelos sensores presentes na esteira, executar a rede PMC e apresentar o resultado da classificação.

O aplicativo foi desenvolvido utilizando linguagem Delphi. O Delphi é uma linguagem de alto nível, compilada e fortemente tipada que suporta design estruturado e orientado a objetos. (EMBARCADERO TECHNOLOGIES, 2019)

Uma das vantagens da utilização da IDE de desenvolvimento Delphi é a possibilidade de utilização do conceito RAD (Rapid Application Development), que é um modelo de desenvolvimento que prioriza a criação rápida de protótipos, permitindo o desenvolvimento de aplicativos robustos de forma simplificada.

O funcionamento do aplicativo baseia-se em ler os parâmetros de uma rede PMC previamente treinada, executar a rede recebendo como entrada a leitura feita pelos sensores e fornecendo como saída uma representação visual da classe identificada. A figura 3.22 apresenta a tela do aplicativo de classificação desenvolvido.

Figura 3.23 - Tela do aplicativo de classificação



Os parâmetros da rede são passados para o aplicativo por meio de um arquivo no formato INI, que é um arquivo em texto utilizado para armazenar dados e configurações de um sistema (MACORATTI, 2019). Este formato de arquivo permite a separação dos dados em seções, entradas e valores. As seções identificam um grupo de entradas, que por sua vez armazenam valores que podem ser lidos pelo sistema.

O arquivo INI com as informações da rede a ser executada foi definido como tendo uma seção denominada “REDE”, contendo como entradas os itens “Camadas” que possui os valores de neurônios em cada camada da rede separados por vírgula, e “NivelAtivacao” que corresponde ao valor mínimo para que um neurônio seja considerado como ativado na camada de saída. Para cada uma das camadas da rede é definida uma seção no arquivo com o nome “CAMADA”, esta seção é composta pela entrada “Pesos” que armazena os dados dos pesos sinápticos de cada uma das entradas dos neurônios desta camada. Os valores de cada peso são separados por ponto e vírgula e o conjunto de pesos de cada neurônio é separado pelo caractere barra vertical. A última seção do arquivo é denominada “DADOS_TREINAMENTO” e possui as entradas “Media” e “Desvio” correspondentes à média e ao desvio padrão dos dados utilizados no treinamento da rede. Esta informação é utilizada para a normalização dos valores fornecidos pelos sensores. Um exemplo do formato de arquivo utilizado para fornecer os dados da rede pode ser observado no apêndice F.

O aplicativo se comunica com o Arduino por meio de uma interface serial, para isto o código utilizado para coleta de dados apresentado no apêndice C foi alterado de forma a enviar os dados coletados da peça pela porta serial do microcontrolador. Estes dados são lidos pelo aplicativo e normalizados utilizando a equação 3.1, em seguida é executada a etapa *forward* do algoritmo PMC, em que a entrada normalizada é apresentada à camada de entrada e propagada até a camada de saída utilizando o somatório das entradas ponderadas pelos pesos sinápticos e a ativação de cada neurônio fornecida pela função de ativação logística, resultado na ativação de um dos neurônios na camada de saída, analisada considerando o limiar de ativação fornecido como parâmetro no arquivo de configuração. Com o resultado da classificação o programa exibe o nome juntamente com uma figura representativa da peça identificada.

O código fonte do aplicativo desenvolvido pode ser observado no apêndice G.

4.4. Interferências externas e ajustes

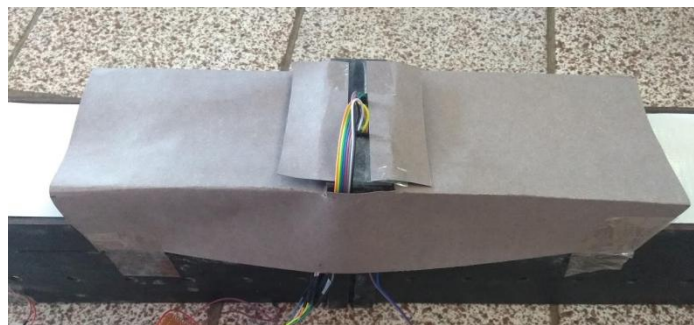
A partir dos dados obtidos com o treinamento da rede escolhida para a execução deste trabalho e utilizando o software desenvolvido para a classificação das peças foram realizados testes com o objetivo de verificar a capacidade da rede de analisar amostras apresentadas e classificá-las.

A metodologia utilizada consistiu em inserir na esteira em ordem aleatória peças pertencentes às classes propostas e verificar se a resposta apresentada era compatível com a esperada. Em cada teste realizado foram apresentadas vinte amostras de cada classe, totalizando 120 amostras por teste executado.

Nesta etapa foi possível observar os efeitos de variáveis externas na execução da rede e consequentemente na capacidade de generalização da rede neural.

Nos testes iniciais foi possível observar que a influência da iluminação ambiente nas leituras realizadas pelo sensor de cor era bastante significativa, ou seja, leituras realizadas na presença de luz natural apresentavam valores diferentes comparadas com leituras realizadas com iluminação artificial. Para minimizar esta variável foi instalado na esteira um anteparo com a finalidade de isolar a área de leitura do sensor de cor da influência da variação de luminosidade.

Figura 4.1 - Sensor com anteparo instalado



Outra característica construtiva que teve influência nos resultados encontrados foi a variação de velocidade do motor devido a defeitos mecânicos no equipamento utilizado. Foi observado que movimentações do cabo de alimentação do motor ocasionavam a variação da posição das escovas e por consequência dificultava o contato destas com as bobinas, ocasionando em variações abruptas na velocidade de rotação da esteira. Esta variável é relevante pois a leitura

do tamanho da peça é diretamente relacionada à velocidade com que ela se desloca sobre a esteira. A solução encontrada para contornar este problema foi a fixação do cabo de alimentação do motor, de forma que a região vulnerável ao deslocamento do cabo permanecesse fixa.

Outra observação importante sobre a leitura do comprimento da peça é a necessidade de um posicionamento correto da peça na esteira. A melhor condição de leitura dos dados é obtida quando a peça é posicionada no centro da largura da esteira, com seu comprimento alinhado perpendicularmente ao feixe do sensor de barreira. Pequenas variações deste posicionamento não afetaram os resultados das leituras, mas variações de posição, principalmente no ângulo de rotação da peça em relação ao sensor de barreira, podem ocasionar em alterações relevantes na leitura de comprimento.

4.5. Readequação da rede e resultados

Os ajustes realizados tornaram os parâmetros da rede utilizada obsoletos, tendo em vista que os dados foram coletados sem o anteparo do sensor e sujeito a variação de velocidade do motor. Sendo assim se fez necessária uma nova coleta de dados para o treinamento de uma rede que se adequasse às condições atuais. Foram novamente coletadas trezentas amostras das classes para um novo treinamento e validação.

Com a nova rede configurada foram repetidas as baterias de testes onde o sistema se comportou de forma satisfatória.

Foi realizada uma sequência de dez testes onde em cada um foram introduzidas no sistema 120 amostras, obtendo-se uma taxa média de acerto de 81,6%.

Esta taxa de acertos pode ser aumentada com melhorias na parte física do projeto, principalmente no que diz respeito à esteira transportadora, mecanismo com extrema importância tanto na etapa de coleta de dados para treinamento, quanto na etapa de leitura de dados para a classificação que, no caso deste trabalho, apresentou problemas que impactaram no resultado obtido.

5. CONCLUSOES E TRABALHOS FUTUROS

Este trabalho teve o objetivo de utilizar técnicas de reconhecimento de padrões na classificação de peças segundo suas características físicas. Foram selecionadas como características analisadas a cor do objeto e seu comprimento.

Para esta tarefa foi desenvolvida uma rede neural que executou o algoritmo Perceptron Multicamadas.

O código desenvolvido em Matlab para etapa de obtenção dos parâmetros e da seleção da melhor topologia de rede que atendesse os requisitos do trabalho funcionou perfeitamente, possibilitando a execução de testes de redes com topologias distintas, permitindo analisar de forma eficiente as diferenças entre as configurações escolhidas e assim encontrar os parâmetros que mais se adequassem ao problema apresentado.

O sensor de cores utilizado na coleta de dados se comportou de forma satisfatória. Vale salientar que as leituras dos componentes de cor podem sofrer interferência de vários fatores como posição da peça, iluminação ambiente e a capacidade de flexibilidade do objeto. Estas características corroboram para a afirmação da escolha de utilização da rede neural na solução do problema, tendo em vista que muito raramente as leituras dos dados de cores de peças da mesma classe serão idênticas, característica que é contornada pela capacidade de generalização da rede.

O sensor de barreira desenvolvido para detectar a presença da peça e fazer a leitura do dado correspondente ao comprimento do objeto também cumpriu com sucesso a tarefa. A escolha de utilização do diodo LASER em conjunto com o RDL se mostrou acertada, tendo em vista que estes componentes em nenhum momento apresentaram falhas. Assim como a leitura de cores a leitura do comprimento da peça é afetada por fatores externos, como a posição inicial da peça na esteira, e mais significativamente, variações na velocidade de movimentação do objeto, mais uma característica que depende da capacidade de generalização da rede para uma correta classificação.

Apesar da habilidade de generalização, variações extremas nas leituras dos sensores podem afetar negativamente no desempenho do algoritmo. Isto ficou claro durante os testes de desempenho em tempo real da rede, em que a interferência da luminosidade e as variações na velocidade do motor resultaram no deterioramento do desempenho da rede.

Os problemas encontrados durante a validação puderam ser contornados com soluções simples, tanto para a questão da luminosidade com a instalação de um anteparo, quanto para o defeito do motor que tinha seu funcionamento afetado pela movimentação do cabo de alimentação, resolvido com a melhor fixação deste.

De modo geral a o trabalho apresentou resultados que permitem concluir que a utilização redes neurais, em particular as redes PMC, são soluções viáveis para a classificação de padrões. A taxa de acerto de 81,6% pode ser considerada satisfatória, tendo em vista se tratar de um protótipo e a existência de possíveis interferências não detectadas.

O trabalho foi desenvolvido de forma que sua reprodutibilidade fosse de fácil execução e para projetos futuros fica a sugestão da utilização de métodos capazes de calcular a velocidade de deslocamento da esteira, tais como a utilização de encoders em conjunto com o sensor de barreira, permitindo uma melhora no cálculo do comprimento do objeto minimizando possíveis interferências causadas pela variação da velocidade da esteira, causadas não só por defeitos estruturais como no caso deste trabalho, mas também por características físicas da esteira como atrito entre as peças e possíveis variações de tensão no motor. Outra possível melhoria é utilização de captura de imagens dos objetos através de câmeras, podendo analisar, além das cores, dados bidimensionais das peças.

Melhorias também podem ser implementadas no código de treinamento da rede PMC, como a inclusão do termo momentum, evitando que o treinamento convirja para um mínimo local; uso de taxa de aprendizagem adaptativa, melhorando o tempo de convergência da rede; comparação simultânea da taxa de acerto no grupo de testes e validação durante o treinamento, tornando possível a verificação do início super especialização da rede, que ao ser detectado pode funcionar como critério de parada em substituição ao parâmetro erro mínimo, e ainda a utilização de matrizes de confusão, permitindo a análise do desempenho do classificador em cada uma das classes propostas.

6. REFERÊNCIA BIBLIOGRÁFICA

ADAMOWICZ, Elizabeth Cristina; SAMPAIO, Maria Eugênia de Carvalho e Silva;

BARBOZA, Angela Olandoski. Reconhecimento de padrões na análise econômico-financeira de empresas. *In: XXII ENCONTRO NACIONAL DE ENGENHARIA DE PRODUÇÃO*, 2002, Curitiba - PR. **Anais** [...]. [S. l.: s. n.], 2002.

AGUIAR, Fernando Guimarães. **Utilização de redes neurais artificiais para detecção de padrões de vazamento em dutos**. 2010. Dissertação (Mestrado em Térmica e Fluidos) - Escola de Engenharia de São Carlos, University of São Paulo, São Carlos, 2010. doi:10.11606/D.18.2010.tde-17012011-160008. Acesso em: 2019-02-26

ARDUINO. 2018. Disponível em: <<https://www.arduino.cc/>>. Acesso em: 25 out. 2018

BIANCHI, Marcelo Franceschi de. **Extração de características de imagens de faces humanas através de wavelets, PCA e IMPCA**, São Carlos, 2006.

CARDOSO, Sergio A. *et al.* SESAME: sistema de reconhecimento de comandos de voz utilizando PDS E RNA. *In: XVIII CONGRESSO BRASILEIRO DE AUTOMÁTICA*, 2010, Bonito-MS. **Anais** [...]. [S. l.: s. n.], 2010

CARTÃO SANDISK microSDHC. [S. l.: s. n.], 2019. Disponível em: <https://www.sandisk.com.br/home/memory-cards/microsd-cards/sandisk-microsd>. Acesso em: 18 fev. 2019.

DIODOS Laser. 2018. Disponível em: <<http://blog.novaeletronica.com.br/diodos-LASER/>>. Acesso em: 26 out. 2018.

EMBARCADERO TECHNOLOGIES. **Language Overview**. [S. l.], 2019. Disponível em: http://docwiki.embarcadero.com/RADStudio/Rio/en/Language_Overview. Acesso em: 19 abr. 2019.

FARINELLI, Fernanda. **Conceitos Básicos de Programação Orientada a Objetos**. [S. l.], 2007. Disponível em: http://jocivan.com.br/portal/wp-content/uploads/2015/02/APOSTILA_3.1.1.pdf. Acesso em: 18 fev. 2019.

HAYKIN, Simon. **Redes neurais: princípios e prática**. 2. Ed. Porto Alegre, 2001.

KAUFMANN. **Esteiras Transportadoras**. [S. l.], 2019. Disponível em: <https://www.kaufmann.com.br/esterias-transportadoras>. Acesso em: 14 fev. 2019.

MACORATTI, José Carlos. **O que é um arquivo INI**. [S. l.], 2019. Disponível em: http://www.macoratti.net/arq_ini.htm. Acesso em: 19 abr. 2019.

MAITICH, Damián Jorge. **Redes Neuronales: Conceptos Básicos y Aplicaciones**, Rosario, AR, 2001.

MICROSOFT. **PADRONIZAR (Função PADRONIZAR)**. [S. l.: s. n.], 2019. Disponível em: <https://support.office.com/pt-br/article/padronizar-fun%C3%A7%C3%A3o-padronizar-81d66554-2d54-40ec-ba83-6437108ee775>. Acesso em: 19 fev. 2019.

MUELLER, Alessandro, **Uma Aplicação de Redes neurais Artificiais na Previsão do Mercado Acionário**, Florianópolis, 1996.

NETO, Edson Cavalcanti ; FILHO, Pedro Pedrosa Rebouças. Técnicas de PDI e inteligência artificial aplicadas ao reconhecimento de placas de carro nos padrões brasileiros. *In: XI SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE*, 2013, Fortaleza-CE. **Anais [...]**. Fortaleza-CE: [s. n.], 2013.

PIRES, Marco Túlio. **Guia de Dados Abertos**. São Paulo: [s. n.], 2015. Disponível em: https://ceweb.br/media/docs/publicacoes/13/Guia_Dados_Abertos.pdf. Acesso em: 18 fev. 2019.

SILVA, Ivan Nunes da; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade. **Redes Neurais Artificiais Para Engenharia e Ciências Aplicadas: Fundamentos Teóricos e Aspectos Práticos**. 2ª. ed. São Paulo: Artliber, 2016. 431 p.

SILVA, Tiago Granja da. **Investigação Experimental de Técnicas de Reconhecimento de Padrões em Sistemas Automáticos**, Porto, 2010.

SOUSA, Carlos Alberto Gonçalves de. **Análise comparativa de tecnologias de baixo custo para reconhecimento de padrões em sistemas automáticos**. 2011. 115 p. Dissertação (Mestrado Integrado em Engenharia Mecânica Seção de Automação, Instrumentação e Controle) - Faculdade de Engenharia, Universidade do Porto, Porto, Portugal, 2011.

TEXAS ADVANCED OPTOELETRONIC SOLUTIONS. **TCS230 PROGRAMMABLE COLOR LIGHT-TOFREQUENCY CONVERTER**. Plano, TX USA: [s. n.], 2003.

THOMAZINI, Daniel; ALBUQUERQUE, Pedro Urbano Braga de. **Sensores Industriais: Fundamentos e Aplicações**. 4. ed. São Paulo: Editora Érica Ltda, 2007. 222 p.

APENDICE A - Classe Motor

```
Motor::Motor()
{
    //ctor
}
Motor::Motor(unsigned int Pino)
{
    SetPinoMotor(Pino);
}
void Motor::SetPinoMotor(unsigned int val)
{
    PinoMotor = val;
    pinMode(PinoMotor,OUTPUT);
}
void Motor::Liga()
{
    analogWrite(PinoMotor,PWM);
}
void Motor::Liga(unsigned int _PWM)
{
    analogWrite(PinoMotor,_PWM);
}
void Motor::Desliga()
{
    digitalWrite(PinoMotor,LOW);
}
Motor::~~Motor()
{
    //dctor
}
```


APENDICE B – Classe “ColorSensor”

```
/** CONSTRUCTORES **/
```

```
ColorSensor::ColorSensor()
```

```
{
```

```
}
```

```
ColorSensor::ColorSensor(unsigned int S0,unsigned int S1,unsigned int S2,unsigned int S3,unsigned int Out)
```

```
{
```

```
    SetpinS0(S0);
```

```
    SetpinS1(S1);
```

```
    SetpinS2(S2);
```

```
    SetpinS3(S3);
```

```
    SetpinOut(Out);
```

```
}
```

```
ColorSensor::ColorSensor(unsigned int S0,unsigned int S1,unsigned int S2,unsigned int S3,unsigned int Out,unsigned int Freq)
```

```
{
```

```
    SetpinS0(S0);
```

```
    SetpinS1(S1);
```

```
    SetpinS2(S2);
```

```
    SetpinS3(S3);
```

```
    SetpinOut(Out);
```

```
    SetOutFreq(Freq);
```

```
}
```

```
ColorSensor::ColorSensor(unsigned int OE,unsigned int S0,unsigned int S1,unsigned int S2,unsigned int S3,unsigned int Out,unsigned int Freq)
```

```
{
```

```
    SetPinOE(OE);
```

```
    SetpinS0(S0);
```

```
    SetpinS1(S1);
    SetpinS2(S2);
    SetpinS3(S3);
    SetpinOut(Out);
    SetOutFreq(Freq);
}

/** DESTRUTOR */

ColorSensor::~ColorSensor()
{
    //dtor
}

/** MÉTODOS */

void ColorSensor::SetEnable(bool val)
{
    if (val)
    {
        digitalWrite(pinOE,LOW);
    }
    else
    {
        digitalWrite(pinOE,HIGH);
    }
}

bool ColorSensor::GetEnable()
{
    if (digitalRead(pinOE)== LOW)
    {
        return true;
    }
    else
    {
```

```
        return false;
    }
}
void ColorSensor::SetPinOE(unsigned int val)//Configura pino OE
{
    pinOE = val;
    pinMode(pinOE,OUTPUT);
}
void ColorSensor::SetpinS0(unsigned int val)//Configura pino S0
{
    pinS0 = val;
    pinMode(pinS0,OUTPUT);
}
void ColorSensor::SetpinS1(unsigned int val)//Configura pino S1
{
    pinS1 = val;
    pinMode(pinS1,OUTPUT);
}
void ColorSensor::SetpinS2(unsigned int val)//Configura pino S2
{
    pinS2 = val;
    pinMode(pinS2,OUTPUT);
}
void ColorSensor::SetpinS3(unsigned int val)//Configura pino S3
{
    pinS3 = val;
    pinMode(pinS3,OUTPUT);
}
void ColorSensor::SetpinOut(unsigned int val)//Configura pino de Saída
{
    pinOut = val;
    pinMode(pinOut,INPUT);
}
void ColorSensor::SetOutFreq(unsigned int val)// Configura escala de frequência de saída
```

```

{
  digitalWrite(pinS0,LOW);
  digitalWrite(pinS1,LOW);
  switch(val)
  {
    case 2:
      digitalWrite(pinS0,LOW);
      digitalWrite(pinS1,HIGH);
      break;
    case 20:
      digitalWrite(pinS0,HIGH);
      digitalWrite(pinS1,LOW);
      break;
    case 100:
      digitalWrite(pinS0,HIGH);
      digitalWrite(pinS1,HIGH);
      break;
  }
}

/*****
*   LEITURA DA COR VERMELHA   *
*****/

unsigned int ColorSensor::GetRed(){
  digitalWrite(pinS2,LOW);
  digitalWrite(pinS3,LOW);
  Red = pulseIn(pinOut,LOW);
return Red;
}

```

```
/******  
*   LEITURA DA COR VERDE           *  
*****/
```

```
unsigned int ColorSensor::GetGreen(){  
    digitalWrite(pinS2,HIGH);  
    digitalWrite(pinS3,HIGH);  
    Green = pulseIn(pinOut,LOW);  
return Green;  
}
```

```
/******  
*   LEITURA DA COR AZUL           *  
*****/
```

```
unsigned int ColorSensor::GetBlue(){  
    digitalWrite(pinS2,LOW);  
    digitalWrite(pinS3,HIGH);  
    Blue = pulseIn(pinOut,LOW);  
return Blue;  
}
```

APÊNDICE C – Coletor de Dados

/*

CORES DOS CONECTORES SENSOR DE COR

PRETO GND

BRANCO OE

CINZA OUT

VERMELHO VCC

AZUL S2

VERDE S3

AMARELO S1

LARANJA S0

*/

#include <Arduino.h>

#include <SPI.h>

#include <SD.h>

#include "ColorSensor.h"

#include "Motor.h"

//Pinos do sensor de cor

#define S0 50

#define S1 51

#define S2 52

#define S3 53

#define Out 48

#define Freq 20

#define OE 49

//Pino motor

```
#define PinMotor 30

//Pino sensor presença

#define PinPresenca 2

//Variaveis globais

File Dados;//Arquivo de dados
const int chipSelect = 4;
//estrutura de cores

struct RGB{
    byte red;
    byte green;
    byte blue;
};

//Sensor de cor
ColorSensor Sensor(OE,S0,S1,S2,S3,Out,Freq);

//Motor
Motor Motor(PinMotor);

RGB ReadColor()
{
    RGB Color = {0,0,0};
    if (Sensor.GetEnable() == true)
    {
        Color.red = Sensor.GetRed();
        Color.green = Sensor.GetGreen();
        Color.blue = Sensor.GetBlue();
    }
    return Color;
}
```

```
}  
//captura dados da peça e grava no cartão SD  
void GetData()  
{  
  Motor.Desliga();  
  Serial.println("LENDO");  
  delay(2000);  
  Serial.print("Vermelho = ");  
  float Red = Sensor.GetRed();  
  Serial.print(Red);  
  Serial.print("\t");  
  Serial.print("Verde = ");  
  int Green = Sensor.GetGreen();  
  Serial.print(Green);  
  Serial.print("\t");  
  Serial.print("Azul = ");  
  int Blue = Sensor.GetBlue();  
  Serial.println(Blue);  
  Dados = SD.open("log.csv",FILE_WRITE);  
  if (Dados){  
    Dados.print(Red);  
    Dados.print(";");  
    Dados.print(Green);  
    Dados.print(";");  
    Dados.println(Blue);  
  }  
  Dados.close();  
}  
void setup()  
{  
  Serial.begin(9600);  
  //Configura Sensor de cor
```



```
    Sensor.SetEnable(true);

//Configura motor

Motor.Desliga();
Motor.SetPWM(255);

//Configura cartão SD

if (SD.begin(chipSelect)){
    Serial.println("Cartao SD pronto para uso");
} else {
    Serial.println("Falha ao inicia cartao SD");
    return;
};
}

void loop()
{
    Serial.println("Aguardando");
    Motor.Liga();
    delay(500);
}
```

APÊNDICE D - Função de treinamento da rede PMC

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               PERCEPTRON MULTICAMADAS                               %
%                               Oscar Felício Candido Longuinho                       %
%                               2018                                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%parâmentros

%net    - vetor com configuração da rede
%x      - matriz com características das amostras
%d      - matriz com respostas desejadas da rede
%lr     - Taxa de aprendizagem
%miner  - erro mínimo
%maxit  - número máximo de iterações

%retornos da função

%W      - Célula com os pesos para cada neurônio na rede
%Er     - Erro mínimo encontrado
%Time  - tempo de execução

function [W,Er,Time]= mlp(net,x,d,lr,miner,maxit)
%% Dados
tic
X = x;           %entradas
D = d;           %alvos

%% parâmetros

Net = net;           %neurônios por camada
layers = size(Net,2); %número de Camadas
NumEx = size(X,1);  %Número de Exemplos
NumIn = size(X,2);  %Número de Entradas
[W,I,Y] = initNet(Net,NumIn); %Inicia os parâmetros da rede
PrvW = W;           %pesos da iteração anterior
LrnRte = lr;        %Taxa de aprendizagem
count = 0;          %Contador de épocas
MaxCount = maxit;   %Número máximo de épocas
minErr = miner;     %erro mínimo;
EM = inf;           %erro
PrvError = 1;       %erro anterior
CurError = 0;      %erro atual
Delta = cell(1,layers); %Delta de Cada neurônio

%% Treinamento

while ((abs(EM - PrvError) > minErr ) && (count < MaxCount)) %Enquanto
a variação do erro for maior que o limite
    PrvError = CalcError(Net,X,D,W);
    for k = 1:NumEx %para cada exemplo
        %FORWARD
        [I,Y] = Forward(Net,X(k,:),W);

        %BACKWARD

```

```

        %cálculo do Delta da camada de saída

        Delta{layers} = (D(k,:) - Y{layers}) .*
(derivative(I{layers}'));

        %atualização dos pesos das camadas de saída

        for j=1:Net(layers)
            W{layers}(:,j) = W{layers}(:,j) +
LrnRte*Delta{layers}(j).*Y{layers-1};
        end

        %Cálculo do Delta das outras camadas
        for n = (layers-1):-1:1
            for j = 1:Net(n)
                Delta{n}(j,1) = -(W{n+1}(j,:) * Delta{n+1}).*
(derivative(I{n}(j)'));
            end
            for j=1:Net(n)
                if n>1 %se não for a camada de entrada
                    W{n}(:,j) = W{n}(:,j) + LrnRte*Delta{n}(j).*Y{n-1};
                else %se for a camada de entrada
                    W{n}(:,j) = W{n}(:,j) +
LrnRte*Delta{n}(j).*X(k,:);
                end
            end
        end
        end
        CurError = CalcError(Net,X,D,W);
        EM = CurError;
        count = count+1;
        Er(count) = CurError;
    end
    Time = toc;
end
%% Funções

% INICIA OS PARÂMETROS DA REDE

function [W,I,Y] = initNet(Net,NumIn)
    W = cell(1,size(Net,2));           % pesos sinápticos
    I = cell(1,size(Net,2));           % inputs dos neurônios
    Y = cell(1,size(Net,2));           % outputs dos neurônios
    W(1) = {rand(NumIn,Net(1))};
    I(1) = {zeros(Net(1),1)};
    Y(1) = {zeros(Net(1),1)};
    for i=2:size(Net,2)
        W(i) = {rand(Net(i-1)+1,Net(i))};
        I(i) = {zeros(Net(i),1)};
        Y(i) = {zeros(Net(i),1)};
    end
end

% FUNÇÃO DE ATIVAÇÃO

function y = active(inp)
    y = zeros(length(inp),1);

```

```

%função logística
for i=1:length(inp)
    y(i) = logsig(inp(i));
end

end

% DERIVADA DA FUNÇÃO DE ATIVAÇÃO

function dy = derivative(y)
    dy = zeros(length(y),1);
    %função logística
    for a=1:length(y)
        dy(a) = logsig(y(a)) * (1-logsig(y(a)));
    end
end

% FORWARD DA REDE

function [I,Y] = Forward(Net,X,W)
    I = cell(1,size(Net,2));
    Y = cell(1,size(Net,2));
    Layers = size(W,2);
    for i=1:Net(1) %para cada neurônio da primeira camada
        I{1}(i) = X * W{1}(:,i);
        Y{1} = active(I{1});
    end
    for i=2:Layers %para cada camada i
        Y{i-1} = vertcat(1,Y{i-1}); %acrescenta o Bias
        for j = 1:Net(i) %para cada neurônio j na camada
            I{i}(j) = Y{i-1}' * W{i}(:,j); %calcula o input do neurônio j
        end
        Y{i} = active(I{i});
    end
end

end

% CÁLCULA O ERRO QUADRÁTICO MÉDIO

function em = CalcError(Net,X,D,W)
    HistError = zeros(size(X,1),1); %histórico de erros
    for k=1:size(X,1)
        [I,Y] = Forward(Net,X(k,:),W);
        HistError(k) = (sum((D(k,:) - Y{size(Net,2)}) .^2))/2;
    end
    em = (sum(HistError))/size(X,1);
end

```

APENDICE E – Script de validação cruzada

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               PERCEPTRON MULTICAMADAS                               %
%                               VALIDAÇÃO CRUZADA                               %
%                               Oscar Felício Candido Longuinho                               %
%                               2018                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

close all;
clear;
clc;

```

```

%% Dados amostras

```

```

Data = csvread('dados_classes.csv');          %Dados dos exemplos
X = Data(:,1:4);                              %entradas
X = horzcat(ones(size(X,1),1),X);            %adiciona bias às entradas
D = Data(:,5:10);
%% parâmetros validação
nAmostras = size(X,1);                        %Total de amostras
txTestes = 0.6;                              %Taxa de amostras no grupo de teste
nClasses = 6;                                %Número de classes amostradas
%% seleção aleatórias de amostras para os grupos
inTestes = zeros(1,nAmostras);               %vetor cujos indices represenatam as
amostras 1 amostra selecionada 0 não selecionada;
inValid = ones(1,nAmostras);                 %vetor cujos indices represenatam as
amostras 1 amostra selecionada 0 não selecionada;
nTestes = nAmostras * txTestes;              %Número de amostras que deverão ser
selecionadas para o grupo de testes
nValid = nAmostras - nTestes;                %Número de amostras que deverão ser
selecionadas para o grupo de validação
cont = 0;
while cont < nTestes                          %Seleciona aleatoriamente as amostras
dos grupo de Testes
    I = randi([1 nAmostras]);
    if (inTestes(I) == 0)
        inTestes(I) = 1;
        cont = cont + 1;
    end
end
inValid = inValid - inTestes;                 %As amostras do grupo de validação são
as que não fazem parte do grupo de testes
cont = 1;
i = 1;
while cont <= nAmostras
    if inTestes(cont) == 1
        Xt(i,:) = X(cont,:);                  %Grupo de Testes
        Dt(i,:) = D(cont,:);
        i = i + 1;
    end
    cont = cont + 1;
end
cont = 1;
i = 1;
while cont <= nAmostras
    if inValid(cont) == 1

```

```

        Xv(i,:) = X(cont,:);           %Grupo de Validação
        Dv(i,:) = D(cont,:);
        i = i + 1;
    end
    cont = cont + 1;
end
%% configurações de redes a serem testadas
Net(1,:) = [14 6];
Net(2,:) = [6 6];
Net(3,:) = [3 6];
LrnRte = 0.1;
MinErr = 10e-8;
MaxIt = 50000;
%% validação cruzada
for I=1:size(Net,1)
    %% treinamento da rede
    [W,Er,Time]= mlp(Net(I,:),Xt,Dt,LrnRte,MinErr,MaxIt);
    %% validação
    Y = cell(1);
    for i=1:size(Xv,1)
        Y{i} = classifica(Net(I,:),Xv(i,:),W,"logsig");
    end

    %% Ajuste das respostas da validação
    Yadj = Y;
    for i=1:size(Y,2)
        for j = 1 :size(Y{1},1)
            if Y{i}(j) >= 0.5
                Yadj{i}(j) = 1;
            else
                Yadj{i}(j) = 0;
            end
        end
    end

    %% Comparação da saída ajustada da rede com o esperado

    Result = zeros(size(Dv,1),1);
    for i=1:size(Dv,1)
        if isequal(Dv(i,:),Yadj{i}')
            Result(i) = 1;
        else
            Result(i) = 0;
        end
    end

    Acertos = sum(Result);

    TxAcertos(I) = Acertos /size(Result,1);
end

```

APENDICE F – Arquivo de configuração da rede

[REDE]

Camadas = 14,6

NivelAtivacao = 0,5

[CAMADA1]

Pesos=-2,80488484646599;-0,988375764000241;-0,105810527032949;-
 0,581966085104258;4,43451324974498|0,211799356450271;4,72974172204965;-
 2,40649784584931;-2,09682110236787;-4,92979390402783|-0,778637577176549;-
 4,51598772758906;3,02290422180081;2,00393683076594;4,88131312800247|-
 0,236626625776649;4,35123122589266;-1,88683533611063;-1,93490653510643;-
 4,31058429949855|-2,75552972991354;0,179168272981607;-1,41509053815250;-
 1,60439468833492;3,59237097399274|3,45003116677335;-
 4,69578455144215;3,81979065723619;5,22100919312358;-4,85167733708243|-
 7,18366824468672;8,11636714148785;-4,58130462177119;-
 6,65230213968716;3,45228712247389|7,16676881317897;-
 7,52653741228506;3,17369594408241;6,12432529751782;-3,12256038643788|-
 1,88822390177825;5,90141070393587;-1,60312448654471;-1,51609132248906;-
 0,0401640277443501|2,96168777575819;-
 5,49819551745538;2,03150836490423;2,48865604903282;-
 0,934829326047231|2,05571501931188;3,32975294382162;2,36758238793342;2,575882989
 41175;-3,99651140453298|0,514576939394656;-8,87361828775369;0,0637010433409339;-
 1,92922134834834;5,39907246168973|-3,82265784469330;11,9857914897293;-
 0,864004842576792;3,07925166234189;-5,72376774329287|0,927049732074187;-
 9,01015554244423;9,52159009117113;5,79032691947145;2,42093501261894

[CAMADA2]

Pesos=-2,60533679695948;-3,82029283970204;4,50719493539261;-
 6,16836172973006;0,738590583201820;-3,94561645296911;6,05551124934903;-
 10,3325583032234;2,74501199750101;-0,0185120700088478;0,563637804423975;-
 0,256076267251529;-2,07297991106689;-0,618546858983982;-5,59938817383836|-
 5,33437151679016;-2,82074451718042;-2,55905773461686;-3,54295928298783;-
 1,65709245659550;-3,77738596726536;-3,06783612843409;3,43504619422190;-
 5,58117916826277;2,43797784408693;-4,41748181402556;-
 0,319871014813942;5,29807230402725;-8,38306767156316;14,3437979265657|-

1,84350157444818;-4,64118903149717;3,09913065615791;-
4,06308588144592;0,344704104099961;0,924252571625722;-
7,59555928977967;5,41700689252629;-5,20056776786295;1,23157263890686;-
5,06154800541397;0,769259138734813;-1,25892214436991;0,0893751661144890;-
3,64009056125139|-3,31102198974540;4,88015618457059;-
7,62694077177752;3,90370616962135;-4,04240539075305;2,89049427396068;-
2,56929191692486;2,79486847378567;-5,26692343894785;1,93597304254856;-
3,44144750731975;0,0999190642322666;-8,69795004512119;5,32462891182597;-
13,9510878797820|-1,14572867371461;-1,73173225668864;-
1,66609936490866;1,01456801240287;-1,49745658956695;-
4,85608902113363;3,77612759317020;-3,40015559819239;0,823561346351768;-
4,41299938664763;0,487073680088119;0,743089266135912;-1,86530035862289;-
1,55766017777358;2,38369329332414|-1,85713951558389;2,75765794494693;-
2,26188119813346;-0,802251969857014;-2,16069869731207;2,10128074521170;-
1,72051114326173;-5,68097370846919;1,21580613151041;-
2,56962275724008;0,893413767065153;-4,79716606449545;0,645076181828301;-
3,61462415423057;1,55665145633904

[DADOS_TREINAMENTO]

Media = 804,4342

Desvio = 322,5682

APENDICE G – Código fonte do aplicativo classificador.

```
unit uPrincipal;

interface

uses

  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes,
  Vcl.Graphics,
  Vcl.Controls,      Vcl.Forms,      Vcl.Dialogs,      Vcl.StdCtrls,      Vcl.Buttons,
  Vcl.ComCtrls,IniFiles,Math,
  Vcl.ExtCtrls,jpeg, AdPacket, OoMisc, AdPort;

type

TFrmClassifica = class(TForm)
  StatusBar1: TStatusBar;
  OpenFileDialog1: TOpenDialog;
  gpbDados: TGroupBox;
  GroupBox1: TGroupBox;
  lblNumCamadas: TLabel;
  GroupBox2: TGroupBox;
  LblNeucamadas: TLabel;
  Panel1: TPanel;
  Panel2: TPanel;
  Panel3: TPanel;
  SpeedButton1: TSpeedButton;
  Panel4: TPanel;
  Panel5: TPanel;
  Image1: TImage;
  ApdComPort1: TApdComPort;
  ApdDataPacket1: TApdDataPacket;
  ComboBox1: TComboBox;
  BitBtn1: TBitBtn;
  Label1: TLabel;
```

```

GroupBox3: TGroupBox;
GroupBox5: TGroupBox;
procedure ImportaRede(Arquivo:TiniFile);
function Classifica(Amostra:tarray<Extended>):Integer;
function StrArrayToFloatArray(AArray:Tarray<string>):Tarray<Extended>;
function MultiplicaArray(Array1:Tarray<Extended>;Array2:Tarray<Extended>):Extended;
function LogSig(x:Extended):Extended;
function
Normaliza(Dados:Tarray<Extended>;Media:Real;Desvio:Real):Tarray<Extended>;
procedure MostraClasse(aClasse:Integer);
procedure BitBtn2Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure SpeedButton1Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure ApdDataPacket1StringPacket(Sender: TObject; Data: AnsiString);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  FrmClassifica: TFrmClassifica;
  Rede :String;           //Configuração da rede :Neurônios por cada
  Camadas:Tarray<string>; //cada item cotem o numero de neurônios em cada camada
  NumCamadas:Integer;    //Número de Camadas
  NivelAtivacao : real;  //Nivel d eajuste da saída, acima do valor é considerado ativado
  Pesos: array of Tarray<string>; //Pesos sináptcos dos neurônios
  Classe : Integer;      //Classe identificada pela rede
  MediaDados :Real;     //Média dos dados utilizados no treinamento da rede
  DesvioPadrao: Real;   //Desvio padrão dos dados utilizados no treinamento da rede
implementation

{$R *.dfm}

```

```

procedure TFrmClassifica.ApdDataPacket1StringPacket(Sender: TObject;
  Data: AnsiString);
var
  AmostraStr:Tarray<string>;
  DataStr :String;
  Amostra : Tarray<Extended>;
begin
  DataStr := Data;
  DataStr := DataStr.Replace('[',');
  DataStr := DataStr.Replace(']',');
  DataStr := '1;' + DataStr;
  AmostraStr := DataStr.Split([';']);
  Amostra := StrArrayToFloatArray(AmostraStr);
  Amostra := Normaliza(Amostra,MediaDados,DesvioPadrao);
  Amostra[0] := 1;
  Classe := Classifica(Amostra);
  MostraClasse(Classe);
end;

```

```

procedure TFrmClassifica.BitBtn1Click(Sender: TObject);
begin
  ApdComPort1.ComNumber := ComboBox1.ItemIndex;
  ApdComPort1.Open := True;
end;

```

```

{*****
*
* Função Classifica
* Executa o Forward da rede
* Amostra: Vetor com os dados da amostra a ser classificada
* Retorno: Valor inteiro correspondente ao neurônio ativado na saída
*
*****}

```

```

function TFrmClassifica.Classifica(Amostra: Tarray<Extended>):Integer;
var
  I: Integer;
  PesoStr : Tarray<string>;
  PesoFlt : Tarray<Extended>;
  J: Integer;
  OutPuts : array of Tarray<Extended>;
  k: Integer;
begin
  SetLength(OutPuts,NumCamadas);
  for I := 0 to pred(NumCamadas) do
    begin
      SetLength(OutPuts[I],StrToInt(Camadas[I])+1);
      OutPuts[I][0] := 1;
      for J := 1 to StrToInt(Camadas[I]) do
        begin
          PesoStr := Pesos[I][J-1].Split([';']);
          PesoFlt := StrArrayToFloatArray(PesoStr);
          if I = 0 then
            OutPuts[I][J] := LogSig(MultiplicaArray(Amostra,PesoFlt))
          else
            OutPuts[I][J] := LogSig(MultiplicaArray(OutPuts[I-1],PesoFlt))
          end;
        end;
      end;
      OutPuts[I-1][0] := 0;
      for k := 1 to pred(Length(OutPuts[I-1])) do
        begin
          if Outputs[I-1][K] > NivelAtivacao then
            begin
              Result := k;
              exit;
            end;
          end;
        end;
      end;
    end;
  end;
end;

```

```
end;
```

```
procedure TFrmClassifica.FormCreate(Sender: TObject);
```

```
var
```

```
    Arquivo:TiniFile;
```

```
begin
```

```
    if FileExists(ExtractFilePath(Application.ExeName)+'Rede.ini') then
```

```
        begin
```

```
            Arquivo := TIniFile.Create(ExtractFilePath(Application.ExeName)+'Rede.ini');
```

```
            ImportaRede(Arquivo);
```

```
        end;
```

```
end;
```

```
{*****
*
* Procedure Importa Rede
* Importa os dados da rede a partir de um arquivo .ini
* Arquivo: Arquivo .ini contendo os parâmetros da rede
*
*****}
```

```
procedure TFrmClassifica.ImportaRede(Arquivo:TiniFile);
```

```
var
```

```
    PesoCamada:String;
```

```
    I: Integer;
```

```
begin
```

```
    Rede      := Arquivo.ReadString('REDE','Camadas','');
```

```
    MediaDados := StrToFloat(Arquivo.ReadString('DADOS_TREINAMENTO','Media','0'));
```

```
    DesvioPadrao :=
```

```
StrToFloat(Arquivo.ReadString('DADOS_TREINAMENTO','Desvio','1'));
```

```
    Camadas    := Rede.Split([';']);
```

```
    NivelAtivacao := StrToFloat(Arquivo.ReadString('REDE','NivelAtivacao',''));;
```

```
    NumCamadas := Length(Camadas);
```

```
    lblNumCamadas.Caption := IntToStr(NumCamadas);
```

```

LblNeucamadas.Caption := Rede;
SetLength(Pesos,NumCamadas);
for I := 0 to Pred(NumCamadas) do
begin
  PesoCamada := Arquivo.ReadString('CAMADA'+ IntToStr(I+1),'Pesos,");
  Pesos[I] := PesoCamada.Split([' ']);
end;
end;
procedure TFrmClassifica.MostraClasse(aClasse: Integer);
begin
  case aClasse of
    1: Lbl1.Caption := 'Rosa pequena ;
    2: Lbl1.Caption := 'Rosa grande';
    3: Lbl1.Caption := 'Amarela pequena';
    4: Lbl1.Caption := 'Amarela Grande';
    5: Lbl1.Caption := 'Verde pequena';
    6: Lbl1.Caption := 'Verde Grande';
  end;
end;
{*****
*
* Função LogSig
* Aplica a função logística
* x: Valor em que a função será aplicada
* Retorno: Função logística aplicada em x
*
*****}
function TFrmClassifica.LogSig(x: Extended): Extended;
begin
  Result := 1 / (1 + exp(-x));
end;

```

```

{*****
*
* Função MultiplicaArray
* Executa a multiplicação de dois vetores
* Array1: Primeiro vetor da multiplicação
* Array2: Segundo vetor da multiplicação
* Retorno: Resultado da multiplicação entre os dois vetore
*
*****}

```

```

function TFrmClassifica.MultiplicaArray(Array1, Array2: Tarray<Extended>): Extended;
var
  Sum :Extended;
  I: Integer;
begin
  Sum := 0;
  if Length(Array1) = Length(Array2) then
  begin
    for I := 0 to pred(Length(Array1)) do
    begin
      Sum := Sum + (Array1[I] * Array2[I]);
    end;
    Result := Sum;
  end
  else
  begin
    raise Exception.Create('Vetores incompatíveis');
  end;
end;

```

```

{*****
*
* Função Normaliza
* Normaliza os valores de um vetor de acordo com a média
* e desvio padrão dos dados de treinamento da rede
* Dados: Vetor a ser normalizado
* Retorno: Vetor de Float
*
*****}

```

```

function TFrmClassifica.Normaliza(Dados: Tarray<Extended>; Media,
  Desvio: Real): Tarray<Extended>;

```

```

var

```

```

  i: Integer;

```

```

begin

```

```

  if Desvio = 0 then

```

```

    begin

```

```

      raise Exception.Create('Desvio padrão inválido');

```

```

      exit;

```

```

    end;

```

```

  for i := 0 to pred(Length(Dados)) do

```

```

    begin

```

```

      Dados[i] := Abs((Dados[i] - Media)/Desvio);

```

```

    end;

```

```

  Result := Dados;

```

```

end;

```

```

procedure TFrmClassifica.SpeedButton1Click(Sender: TObject);

```

```

var

```

```

  Arquivo:TIniFile;

```

```

begin

```

```

  if OpenDialog1.Execute then

```

```

    begin

```



```

Arquivo := TIniFile.Create(OpenDialog1.FileName);
ImportaRede(Arquivo);
end;
end;
{*****}
*
* Função StrArrayToFloatArray
* Converte um vetor de Strings em um Vetor de Float
* AArray: Vetor a ser convertido
* Retorno: Vetor de Float
*
*****}

function TFrmClassifica.StrArrayToFloatArray(AArray: Tarray<string>): Tarray<Extended>;
var
  ArrayFloat : Tarray<Extended>;
  I: Integer;
begin
  SetLength(ArrayFloat,Length(AArray));
  for I := 0 to pred(Length(AArray)) do
  begin
    ArrayFloat[I] := StrToFloat(AArray[I]);
  end;

  Result := ArrayFloat;
end;

end.

```