Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

Redes Neurais Artificiais e Reconhecimento de Caracteres Numéricos em Imagens

Eric Lau de Oliveira

Assis/SP

2018

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

Redes Neurais Artificiais e Reconhecimento de Caracteres Numéricos em Imagens

Eric Lau de Oliveira

Trabalho de conclusão de Curso
Apresentado ao Instituto de Ensino Superior
De Assis, como requisito do Curso de Graduação
Em Análise e Desenvolvimento de Sistemas

Orientador: Dr. Luiz Carlos Begosso
Orientado: Eric Lau de Oliveira

Assis/SP

2018

FICHA CATALOGRÁFICA

O48r OLIVEIRA, Eric Lau de
Redes neurais artificiais e reconhecimento de caracteres numéri-
cos em imagens / Eric Lau de Oliveira. – Assis, 2018.

105p.

Trabalho de conclusão do curso (Análise e Desenvolvimento de
Sistemas). – Fundação Educacional do Município de Assis-FEMA

Orientador: Dr. Luiz Carlos Begosso

1.Redes neurais 2.Padrões-imagens

CDD 006.32

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

Redes Neurais Artificiais e Reconhecimento de Caracteres Numéricos em Imagens

Eric Lau de Oliveira

Trabalho de conclusão de Curso

Apresentado ao Instituto de Ensino Superior

De Assis, como requisito do Curso de Graduação

Em Análise e Desenvolvimento de Sistemas

Orientador:_____

Dr. Luiz Carlos Begosso

Analisador:_____

Dedicatória

A todos aqueles que tem vontade de aprender.

Agradecimentos

Agradeço primeiramente a Deus.

A minha família que me incentivou a fazer o curso.

Aos amigos que eu fiz durante o período, em especial a Andrezza Lima Aragão que acreditou em mim mais do que eu mesmo.

Ao corpo de professores que sempre estiveram dispostos a me ensinar e a responder minhas perguntas, em especial ao professor Luiz Carlos Begosso, que me ensinou a base da programação e a professora Marisa Atsuko Nitto que me fez apaixonar por matemática.

A todos aqueles que me apoiaram nessa caminhada.

RESUMO

A principal característica dos seres-humanos é sua capacidade de aprender e para este fim o núcleo deste aprendizado está no cérebro, os chamados neurônios. As redes neurais artificiais são simulações matemáticas e estatísticas deste componente biológico e nos últimos anos tem sido estudada e aprimorada cada vez mais para o aprendizado de máquina, buscando a automatização de tarefas e maior velocidade de reconhecimento de padrões. Este trabalho tem como objetivo desenvolver uma rede neural artificial que realize o reconhecimento de padrões em uma imagem.

Palavras chaves: Redes Neurais Artificiais; Aprendizado de Máquina; Reconhecimento de padrões.

ABSTRACT

The main characteristic of human beings is their ability to learn and to this end the core of this learning is in the brain, the so-called neurons. Artificial neural networks are mathematical and statistical simulations of this biological component and in recent years it has been studied and improved more and more for machine learning, seeking the automation of tasks and greater speed of pattern recognition. This work aims to develop an artificial neural network that performs pattern recognition in an image.

Keywords: Artificial Neural Networks; Machine Learning; Pattern Recognition.

Sumário

1. Introdução

Redes neurais artificiais são uma simulação matemática do cérebro humano. Através de algoritmos é possível descobrir padrões presentes nos mais diversos aspectos da vida dos seres-humanos, animais e o universo como um todo.

Padrão é um termo utilizado ao longo de todo o trabalho e, por esse motivo, é importante defini-lo. O senso comum define padrão como qualquer objeto ou formato a ser usado ou imitado como modelo ou protótipo (Michaelis, 2018).

Existem diversos tipos de tarefas repetitivas presentes no planeta feitas por seres humanos, que poderiam ser substituídas por uma máquina. Hoje em dia essa técnica já é utilizada por diversas empresas como no reconhecimento de *spams* em e-mails, reconhecimento e classificação de pessoas ou animais em imagens, carros autônomos, e é possível prever padrões de doenças onde uma rede neural, após treinada, consegue prever com base nos dados fornecidos a ela uma probabilidade de que determinada pessoa possa desenvolver certa doença.

Este trabalho tem por objetivo estudar, e aprender como funcionam esses computadores capazes de aprenderem sozinhos através de algoritmos e cálculos matemáticos, utilizando também técnicas de processamento de imagens.

No algoritmo tradicional é necessário entender todo o problema para que o programador consiga criar um algoritmo seguindo uma série de passos que ele irá impor para a máquina executar. Seguindo esta ideia, quando um novo conceito for apresentado, o programador teria que adicionar mais uma cláusula ao código para que o computador consiga lidar com o novo problema. Ou seja, novamente o ser humano precisa executar uma tarefa que seria mais interessante ser automatizada. Para melhorar isso é necessário fazer o computador aprender sozinho.

Para entender uma rede neural artificial é necessário ter um pouco de conhecimento da base onde ela foi inspirada: o cérebro.

Haykin (2001) cita em sua obra que o cérebro humano tem um valor aproximado de 10 bilhões de neurônios no córtex e 60 trilhões de sinapses ou conexões. As sinapses são a forma de comunicação entre os neurônios, permitindo que a informação se propague saindo dos terminais axônio de um neurônio e conectando aos dendritos de outro, dessa forma o neurônio é estimulado a aprender.

> A maioria dos neurônios codifica suas saídas como uma série de pulsos breves de tensão. Estes pulsos, usualmente conhecidos como potenciais de ação ou impulsos, originam-se no corpo celular dos neurônios, ou perto dele, e então se propagam através dos neurônios individuais a velocidade e amplitude constantes (Haykin, 2001, pág. 33).

A figura 1, representa a estrutura de um neurônio biológico ao qual o neurônio artificial é baseado.

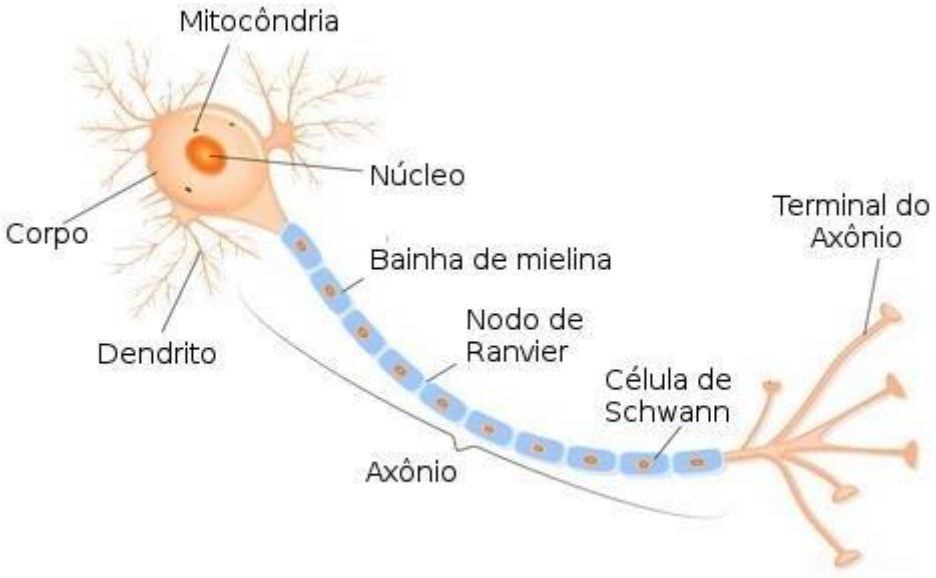

Figura 1 – Neurônio Biológico.

Fonte: https://medium.com/@avinicius.adorno/redes-neurais-artificiais-5b65a43614a0

O neurônio artificial e quase que uma cópia do neurônio biológico onde as entradas que seriam os dados apresentados a ele para que o mesmo aprenda, serão multiplicados pelos seus respectivos pesos, que seriam as sinapses dos neurônios biológico. São os pesos que carregam o conhecimento do neurônio artificial, eles serão corrigidos até que atinjam um valor específico para que seja possível prever com uma alta taxa de acerto o problema proposto a rede neural artificial no início. A multiplicação de todas as entradas e seus respectivos pesos serão ponderados e o resultado será atribuído a uma função de ativação que pode ou não ser o resultado final da rede neural artificial. Caso o resultado não seja satisfatório, ele é utilizado para treinar a rede neural novamente, passando por um algoritmo de correção dos pesos e repetindo o processo de aprendizagem desde o começo.

De acordo com Haykin (2001) este modelo artificial conta com um *bias*, que tem o efeito de aumentar ou diminuir a entrada líquida da função de ativação dependendo se ele é positivo ou negativo.

A figura 2, apresenta um modelo de rede neural artificial linear com uma camada de entrada representada pelos valores X, e para cada entrada é atribuído um peso representado pelo W.

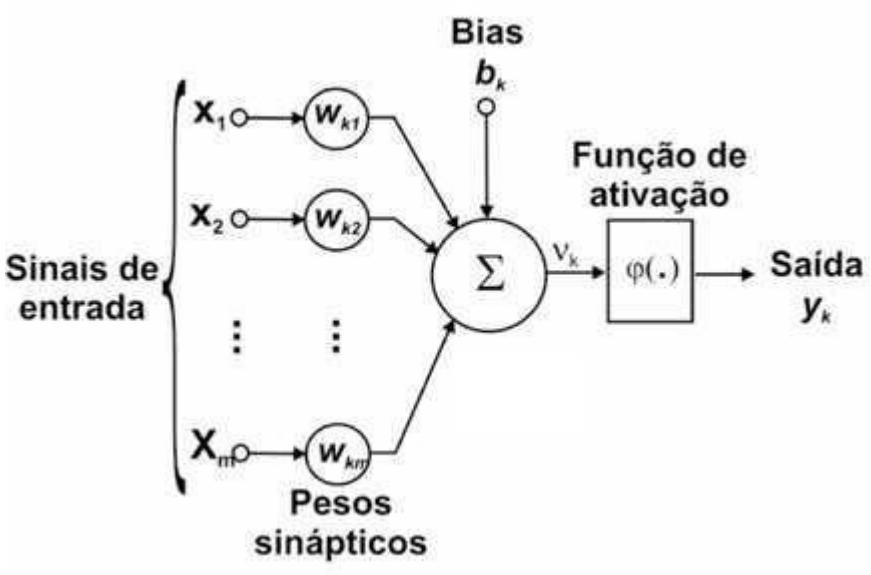

Figura 2, Modelo de um neurônio artificial linear.

Fonte: https://www.researchgate.net/figure/Figura-23-Exemplo-de-neuronio-artificial-Adaptado-de-Haykin-2004_fig3_201549563

As entradas de uma rede são padronizadas de acordo com a função de ativação escolhida.

Existem diversos tipos de modelos de redes neurais como, Perceptron Simples, Perceptron Multicamadas e os Mapas Auto Organizáveis de Kohonen. E cada uma delas lida melhor com certos tipos de problemas.

Para criar uma rede neural é preciso estudar o problema antes de implementá-la. Estes problemas podem ser lineares, geralmente os mais fáceis de resolver e não muito interessantes para redes neurais, e os problemas não lineares onde elas provam seu verdadeiro valor.

Problemas lineares geralmente podem ser classificados e separados por uma reta num plano Cartesiano, onde por exemplo, haja um grupo de cães e gatos e estes não estejam misturados. Seria possível com uma nova amostra de um cão ou de um gato classifica-los nesses dois grupos, entretanto, caso um deles invadisse o grupo um do outro, o algoritmo linear não conseguiria classificá-los.

Um exemplo clássico da programação são os operadores lógicos AND (E) e OR (OU) onde a porta AND só aceita como verdadeira quando todas as entradas são verdadeiras, já a porta OR só responde falso caso todas as entradas sejam falsas. É basicamente uma resposta binária. Já o operador lógico XOR conhecido como “ou” exclusivo é um problema não linear, pois como veremos na figura 3 não é possível traçar uma reta para classificar esta porta lógica. A porta XOR só responde verdadeiro quando apenas uma das entradas é verdadeira.

A figura 3, apresenta no plano cartesiano os operadores lógicos AND, OR mostrando que são linearmente separáveis e o operador XOR sendo não linear, ou seja, não é possível separá-los com uma reta.

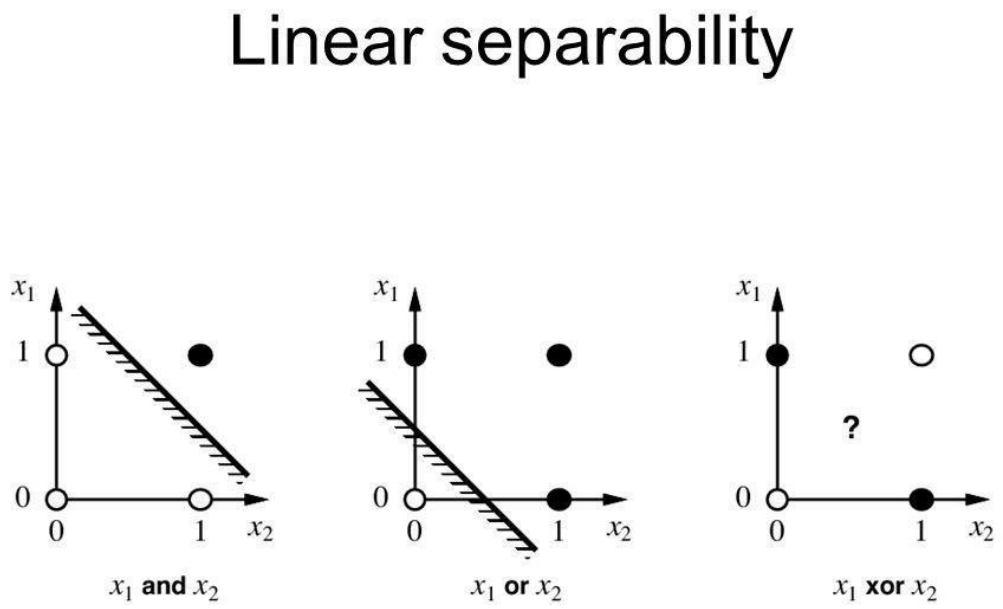

Figura 3. Problemas lineares e não lineares.

Fonte: https://towardsdatascience.com/radial-basis-functions-neural-networks-all-we-need-to-know-9a88cc053448

A figura 4, é um exemplo simples de um neurônio programado utilizando a linguagem Java para a representação dos conceitos utilizados acima. Como é um código apenas para demonstração não foi colocado uma saída desejada para o treinamento do neurônio artificial, portanto o resultado apresentado pode não fazer sentido, mas será melhor explicado no decorrer deste trabalho. É possível perceber que as entradas foram padronizadas com valores entre 0 e 1 e a função de ativação escolhida consegue lidar muito bem com esse tipo entrada. O resultado final nunca excederá os limites de 0 e 1.

```java
public class Neuronio {

    public static void main(String[] args) {

        double x[] = {0.35, 0.57, 0};
        double w[] = new double[x.length];
        double b = (double) Math.random();
        double u = 0.0;
        for(int i = 0; i < x.length; i++) {
            w[i] = (double) Math.random();
            u += (x[i] * w[i]);
        }
        u += b;
        double y = 1 / (1 + Math.exp(-u));
        System.out.printf("Saida: %.8f\n", y);

    }

}
```

Figura 4. Neurônio artificial programado em Java.

Fonte: Próprio autor.

Neste trabalho as redes neurais foram desenvolvidas para lidar com padrões de caracteres numéricos presentes em imagens.

Para isso é necessário entender um pouco de processamento de imagens.

Uma imagem pode ser definida como uma função bidimensional, F (x, y), onde x e y são as coordenadas, e a amplitude de F em qualquer posição (x, y) é chamada de intensidade de cinza. Uma imagem digital é composta por uma infinidade de elementos conhecidos como pixels e cada um possui uma localização e valores específicos (GONZALEZ e WOODS, 2010).

A partir das técnicas de processamento de imagens é possível facilitar o treinamento da rede neural utilizando alguns algoritmos para normalizar a imagem, como o algoritmo de Limiarização de imagens (imagem binária), ou seja, os valores de seus pixels se tornam apenas 0 (preto) e 1 (branco). Segundo Pettersen (2018), ao utilizar esse tipo de modelo a quantidade de neurônios de entrada necessários para a rede é reduzido. Este modelo é mais indicado para reconhecimento de caracteres.

A figura 5, apresenta uma imagem binária.

Figura 5. Imagem binária e matriz da imagem representada por 1 como branco e 0 como preto. (Fonte: Próprio autor).

A matriz binária mostrada na figura 5, seria colocada num vetor, e dado como uma amostra para o neurônio artificial, também seria associada a essa entrada uma saída desejada que vai definir a resposta final do neurônio.

Para o tratamento destas imagens foi desenvolvida uma interface utilizando a linguagem Java e o ambiente de desenvolvimento Eclipse IDE. Esta interface será capaz de capturar imagens presentes no sistema operacional, e guarda-las no banco de dados MySQL que utiliza o modelo relacional, ao fazer isso também será gravado uma cópia da imagem para cada filtro utilizado no programa. As amostras geradas a partir das imagens serão gravadas num banco de dados não relacional chamado MongoDB, assim como os modelos de redes neurais artificiais programadas. Os algoritmos de tratamento das imagens foram feitos com base em códigos já prontos e adaptados para que funcionassem junto a interface criada. Os autores originais foram referenciados dento do código do programa e no final deste trabalho.

1.1 Objetivo

O presente trabalho tem como objetivo estudar modelos de redes neurais artificiais e entender como funcionam os algoritmos criados para que uma máquina possa pensar, e assim baseado em um desses modelos desenvolver uma rede neural que reconheça padrões de caracteres numéricos presentes em imagens.

Este estudo também procura facilitar o entendimento deste assunto que aparentemente parece ser tão complexo, sendo que alguns modelos usados são simples e fáceis de executar.

O objetivo principal é ganhar, e gerar conhecimento, tirando os obstáculos e toda a burocracia de cálculos matemáticos, expressões complexas e gráficos de difícil

entendimento. Espera-se que, ao atingir o maior número de pessoas interessadas sobre o assunto, o presente trabalho possa servir como referência a todos os interessados nesta área.

1.2 Justificativa

Os algoritmos de aprendizagem de máquina têm sido as principais fontes de automatização de tarefas nos últimos anos e o estudo nessa área tem crescido exponencialmente, pois, agora os computadores possuem um poder de processamento que permite a execução desses algoritmos junto a grande quantidade de dados disponível.

As redes neurais artificiais vêm acompanhadas de fundamentos matemáticos e estatísticos muito fortes, o que pode acabar afastando que pessoas se interessem em estudar o assunto, sendo que existem algoritmos que são de fácil entendimento podendo ser a porta de entrada para esta ferramenta que pode ser usada nas mais diversas áreas.

O presente trabalho tem como objetivo enaltecer o campo da tecnologia com novas pesquisas que podem ajudar no desenvolvimento de aplicações colocando em prática os conhecimentos adquiridos nele.

As aplicações desenvolvidas após a compreensão desta ferramenta são estendidas nos mais diversos campos, ou seja, a tecnologia agora não apenas facilita a vida das pessoas como também está começando a evoluir sozinha, a aprender como resolver problemas por si só.

Os fundamentos desta tecnologia estão acompanhados com a evolução dos processadores, e da computação, ao contrário de muitas tecnologias que são lançadas ou modificadas todos os anos, pois, são algoritmos que utilizam de todo o poder de processamento das máquinas, e quanto mais neurônios em uma rede neural, melhor será o resultado final, porém isso torna o treinamento mais lento, já que geralmente é trabalhado com variáveis multidimensionais e quando isso é usado junto ao processamento de imagens os valores tendem a ficar ainda mais grandes pela quantidade de pixels e cores presentes na imagem.

1.3 Público Alvo

Técnicas de aprendizagem de máquina têm sido exploradas cada vez mais nos últimos anos. Empresas como a Uber, uma empresa americana que criou um aplicativo onde motoristas se cadastram para trabalhar como “taxistas”. O aplicativo fornece aos usuários a localização destes motoristas podendo avalia-los tornando o processo mais rápido e amigável que um táxi convencional. A Uber está investindo em aprendizagem de máquina para lidar com a grande quantidade de dados que trafegam e não param de crescer em sua empresa (ENGINEERING, 2017). Ela está investindo na automatização de seus veículos, ou seja, se ela elimina a parte do dinheiro repassado a seus motoristas, significa

que ela recebe o valor total de seus clientes. Isso pode levar a questões éticas e políticas, mas isto é outra história.

Para que um carro possa ser dirigido por uma máquina é necessário um número gigante de variáveis e padrões que apenas algoritmos de aprendizagem poderia lidar. Como reconhecimentos de rostos, pessoas, placas, faróis, animais, dentre outras coisas.

O Google tem usado aprendizagem de máquina para ajudar a NASA a encontrar sistemas planetários e descobrir se existe vida nesses sistemas (CNBC, 2018).

Trazendo para um ambiente mais próximo, já existem técnicas de aprendizagem sendo usadas a algum tempo, como na detecção de spams em e-mails, ou seja, onde há um padrão e possível trabalhar em cima disso.

A automatização de tarefas pode melhorar o funcionamento de empresas, cortar custos, e ainda prever informações. Não é atoa que empresas tão grandes quanto as citadas acima estão investindo nisso.

O Google dispõe de um framework, uma ferramenta gratuita de redes neurais usado para reconhecimento de padrões chamado Tensorflow. Essa ferramenta é de código aberto e está disponível para estudo em seu site (HOLMES, 2015).

Na área médica já é possível identificar padrões de manchas presentes em tomografias acelerando o descobrimento de certos tipos de doenças. Em consequência disso os médicos podem usar o tempo que passariam analisando essas imagens para auxiliar no tratamento de seus pacientes.

O Watson é um software criado pela IBM, uma das principais empresas de tecnologia do mundo, que utiliza aprendizagem de máquina. Ele é um sistema de programação cognitiva podendo ser alimentado com informações novas todas os dias e aprender com isso (D'EGMONT, 2016). Este software está sendo usado nas mais diversas áreas como saúde, medicina, educação e até judicial como descreve AGRELA (2017) em seu artigo, onde esta ferramenta foi contratada por advogados brasileiros para lidar com uma grande quantidade de processos. Em 2013 a IBM faturou cerca de 100 milhões com este supercomputador (FELITTI, 2014).

1.4 Motivação

Uma das motivações para a realização desse trabalho é a de aprender a desenvolver algoritmos de aprendizagem de máquina; trabalhar com imagens utilizando técnicas de programação.

A Inteligência Artificial pode melhorar a vida das pessoas, diminuindo riscos de acidentes de trânsito, pois, uma máquina controlando um carro tem menos possibilidades de erro do que um ser um humano.

Focar a atenção dos médicos em seus pacientes e até mesmo fazendo com que ele possa atender mais pacientes em vez de estar analisando uma tomografia, trabalho este, que poderia ser executado por uma rede neural.

2. Planejamento do Projeto

Para o desenvolvimento deste projeto, as atividades foram separadas em partes. Primeiro um mapa mental foi construído para organizar as ideias principais. A Estrutura Analítica do Projeto (EAP) que compreende cada passo dado desde o início ao fim do projeto também foi elaborada. Finalmente, um cronograma para determinar o limite de tempo em que cada parte será desenvolvida até ser entregue.

A documentação do projeto é importante para que o conhecimento dele não exista apenas na mente do criador, mas também para que outras pessoas possam compreende-la e estuda-la caso seja necessário futuramente.

Uma Interface foi criada em Linguagem Java para capturar e filtrar as imagens que serão usadas como entrada e saída para a rede neural.

Após isso foi escolhido um modelo de redes neurais para trabalhar com o reconhecimento de padrões em caracteres numéricos presentes em imagens.

2.1 Mapa Mental

O mapa mental é uma ferramenta criada para exibir graficamente as ideias principais de um projeto, além de mostrar as ralações que existem entre cada uma delas e a partir disso definir estratégias para alcançar os objetivos que pretendem ser desenvolvidos.

A Figura 6 ilustra o Mapa Mental com as ideias principais desenvolvidas no projeto.

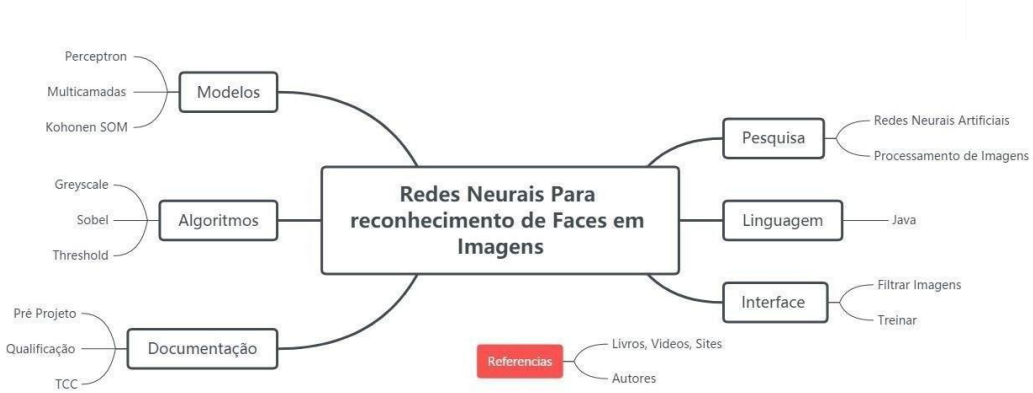

Figura 6. Mapa Mental.

Fonte: Próprio autor.

2.3 Estrutura Analítica do Projeto (EAP)

A EAP é uma sigla para Estrutura Analítica do Projeto, que tem como objetivo dividir o projeto em partes menores, tonando essas partes mais fáceis de serem gerenciadas.

A criação da estrutura analítica do projeto se dá a partir da identificação das principais tarefas e com a subdivisão delas em sistema menores e subprodutos finais.

A EAP é uma decomposição hierárquica do escopo total do projeto a fim de alcançar os objetivos do projeto e criar as entregas requeridas (PMBOK, 5ª Edição).

Na figura 7 EAP, mostra hierarquicamente cada passo do projeto que será desenvolvida.

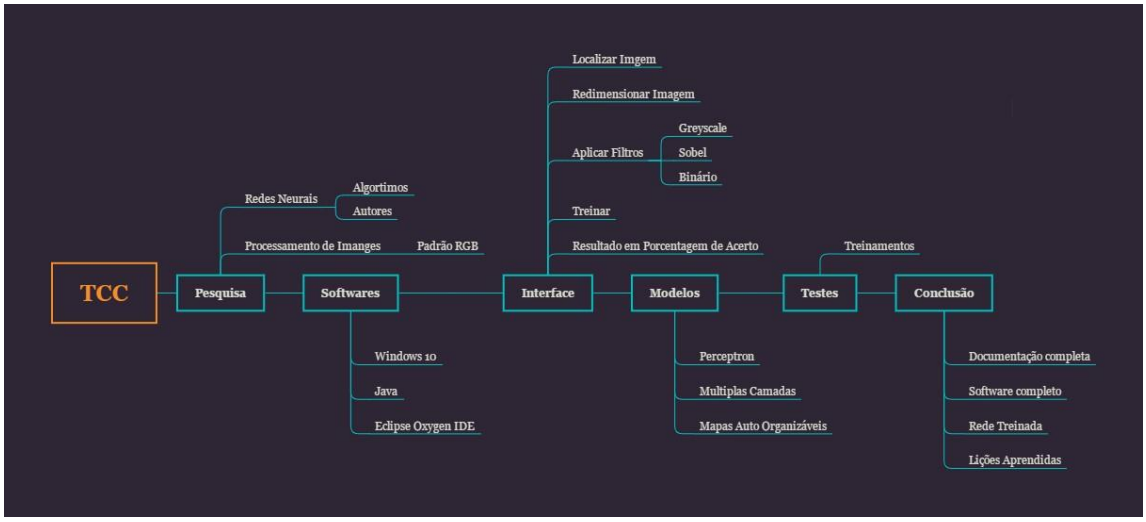

Figura 7. Estrutura Analítica do Projeto (EAP).

Fonte: Próprio autor.

2.4 Cronograma

O cronograma é o processo de análise de sequências das atividades, durações, recursos necessários e restrições do cronograma visando criar o modelo do cronograma do projeto, (PMBOK, 5ª Edição).

É uma matriz que revela graficamente para cada item da EAP, em uma escala de tempo, o período que deve ser realizado. No cronograma está todo o trabalho

que precisa ser feito, quais os prazos que precisam ser cumpridos para que o trabalho seja realizado.

Abaixo, a figura 8 Cronograma, mostra o tempo das atividades que pretendem ser executadas, organizando cada atividade para que tudo fique dentro da data estipulada.

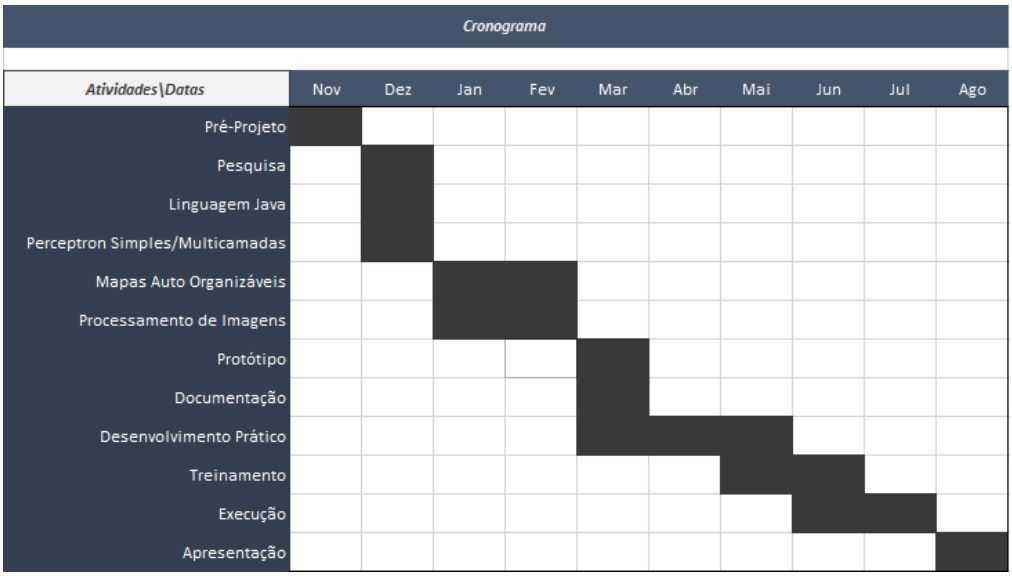

Figura 8. Cronograma do projeto.

Fonte: Próprio autor.

2.5 Especificações do Sistema

Para o desenvolvimento do projeto foi escolhido a Linguagem Java para manipulação de imagens e a criação do modelo de Rede Neural Artificial. Foram utilizados os artefatos da UML como diagramas e conceitos de Programação Orientada a Objetos para que o programa tenha não apenas uma visibilidade clara do código, mas que também ele possa ser reaproveitado por outras funcionalidades do programa.

2.6 UML

A UML (Linguagem de Modelagem Unificada), é uma linguagem visual para documentação de projeto e padrões de software, podendo ser aplicada em várias áreas diferentes e pode documentar e transmitir qualquer coisa. Ela possui nove tipos de diagramas que são usados para documentar e modelar diversos tipos de aspectos.

2.7 Diagrama de Caso de Uso

Este diagrama documenta o que o sistema faz do ponto de vista do usuário. Ele descreve as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema.

A Figura 9 Diagrama de Caso de Uso, permite visualizar as requisições e as principais funcionalidades que o sistema visa desenvolver.

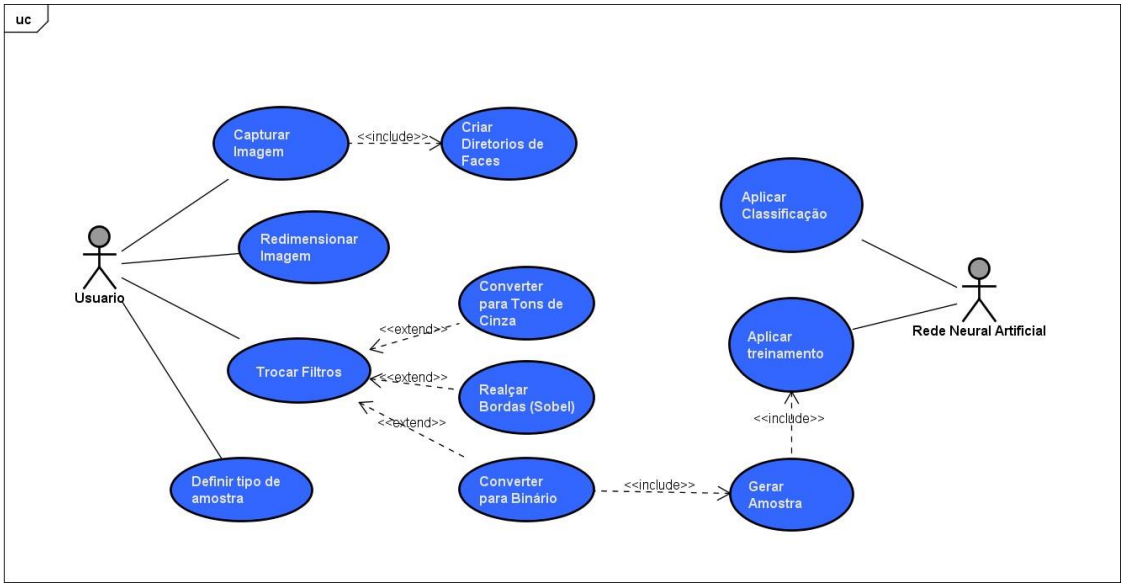

Figura 9. Diagrama de Caso de Uso.

Fonte: Próprio autor.

2.8 Diagrama de Classes

Um diagrama de classes é uma representação da estrutura e relações das classes que servem de modelo para objetos, ele pode ser baseado no diagrama ER (Entidade Relacionamento).

Na Figura 10 Diagrama de Classes, tem por objetivo mostrar a planta do sistema a ser programado e suas conexões e dependências entre as classes.

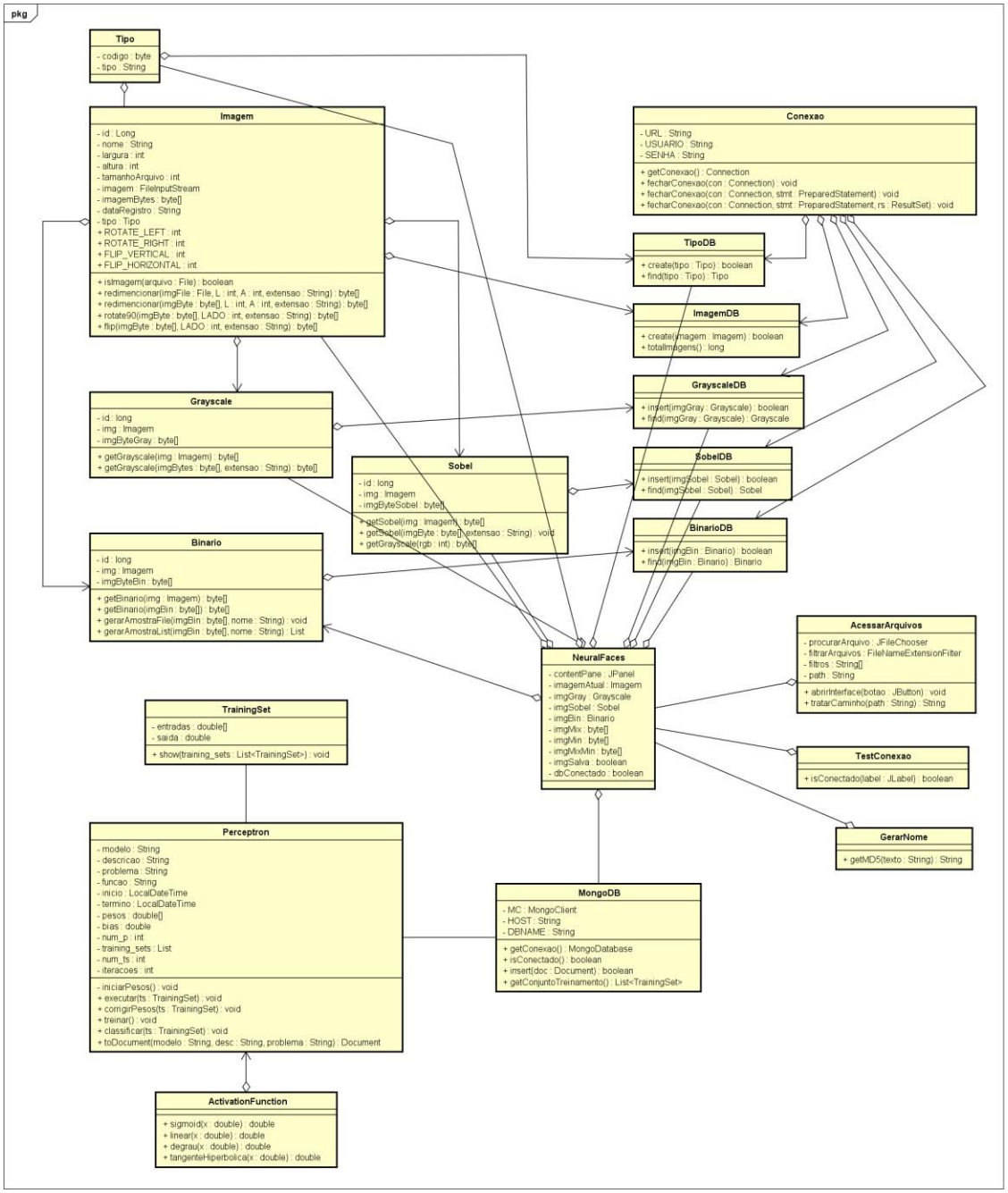

Figura 10. Diagrama de Classes.

Fonte: Próprio autor.

2.9 Diagrama Entidade Relacionamento

O Diagrama Entidade Relacionamento é uma representação gráfica da estrutura que será criada para o banco de dados para o armazenamento dos objetos. Em geral, este modelo representa de forma abstrata a estrutura que possuirá o banco de dados da aplicação.

Abaixo, a Figura 11 Diagrama Entidade Relacionamento, é utilizado para determinar as relações entre os objetos que serão definidos e guardados no banco.

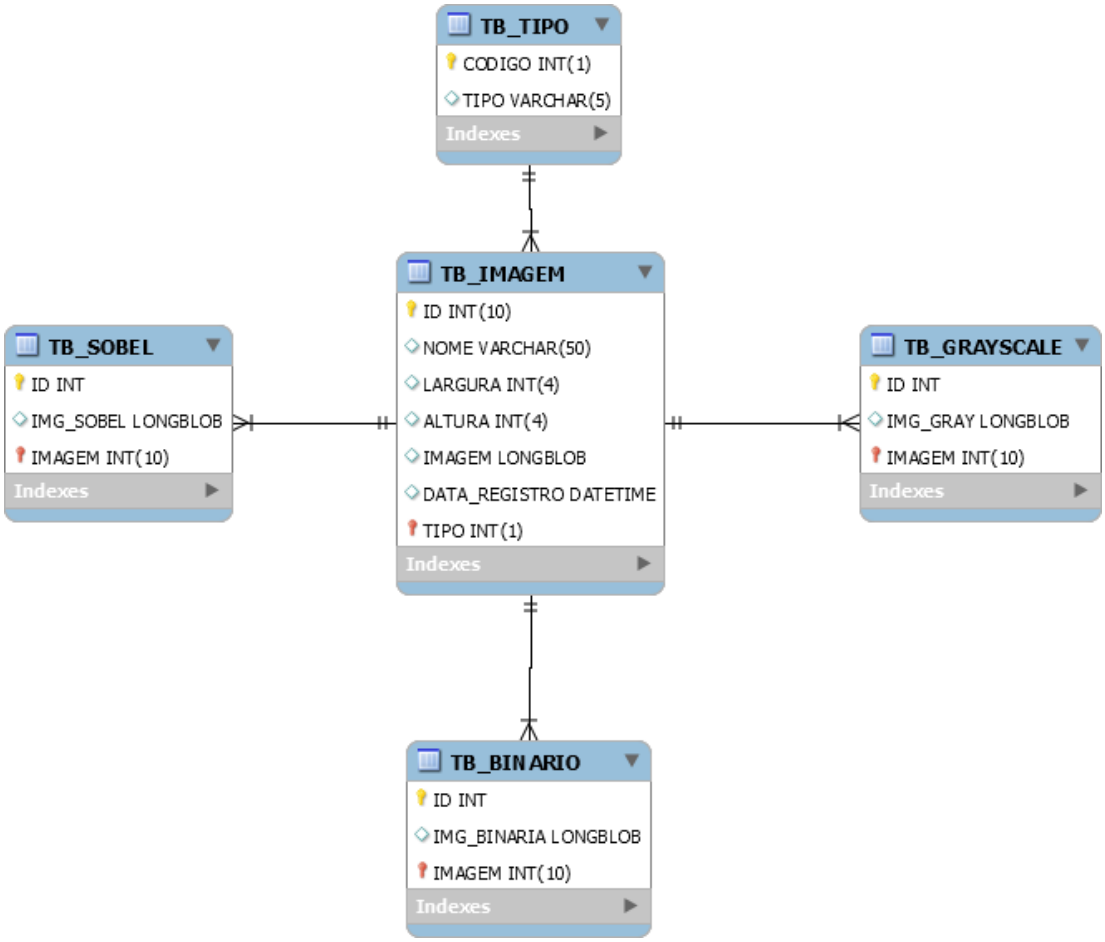

Figura 11. Diagrama Entidade Relacionamento.

Fonte: Próprio autor.

3. Tecnologias previstas

Nesta parte será descrito todas as tecnologias que foram usadas para o desenvolvimento do projeto, desde a parte conceitual até a parte codificada.

3.1 Astah

É uma ferramenta paga feito em java para a criação de diagramas baseados nos conceitos da UML. Porém existe uma versão de comunidade gratuita para estudantes.

3.2 XMind

Ferramenta paga para a criação de mapas mentais e EAPs. Possui uma versão gratuita para teste.

3.3 Java

Java é uma linguagem compilada baseada em C++, financiada pela Sun Microsystems em 1991, e criada por James Gosling, um dos funcionários da empresa que deu a ela o nome Oak (Carvalho) em homenagem a um carvalho que ele sempre via da janela do seu escritório. Mais tarde o nome foi modificado para Java pois já havia uma linguagem com o nome que Gosling havia dado (DEITEL, 2010).

Hoje em dia o Java é mantido pela Oracle, que levou a linguagem de brinde ao comprar a Sun Microsystems, porém, Java é uma linguagem de código aberto, disponível para a comunidade de desenvolvedores gratuitamente.

Um dos principais atrativos do Java é a possibilidade de criar programas que possam ser executados em qualquer sistema operacional. Para que ela consiga fazer essa comunicação é necessário que o SO tenha instalado o JRE para que ele execute um programa feito em Java.

Java é uma linguagem naturalmente Orientada a Objetos, fornecendo melhor desempenho do programa e reutilização do código além de classes já fornecidas por ela.

3.4 Eclipse IDE

É um ambiente de desenvolvimento integrado para ajudar na manutenção do código. Ajuda no processo de compilação e geração de executáveis além atalhos para classes, métodos e atributos, acelerando o processo de desenvolvimento da aplicação.

O Eclipse foi desenvolvido pela IBM que doou o projeto para a comunidade como software livre (Código fonte aberto). Pode ser baixado gratuitamente, tendo suporte a Java dentre outras linguagens.

3.5 MySQL

É um SGBD (Sistema Gerenciador de Banco de Dados) gratuito de fácil utilização e disponível gratuitamente além de ser código aberto. Este banco e baseado no modelo relacional que fornece melhor organização, segurança e integridade das informações.

Um banco de dados pode guardar todo o tipo de informação como textos, números, datas e até arquivos como imagens entre outros. Este modelo e composto de tabelas que são os locais onde ficam o conjunto de dados específico e dentro dessas tabelas estão as linhas que separam as informações referente a cada elemento que as utilize. Nestas linhas existem as colunas que são os atributos ou títulos dados a informação gravada naquela linha que a à identifica. Uma tabela pode ter várias linhas e essas linhas podem ter várias colunas, todas elas com informações relacionadas ao tema em que a tabela foi criada. Por exemplo numa tabela chamada imagem tem-se os atributos id que identifica a imagem como única, o nome, a largura e a altura da imagem. Neste contexto os nomes das colunas estão relacionados com a tabela pois uma imagem possui estes atributos, e não seria ideal se nessa tabela estivesse uma coluna chamada música. A não ser que essa tabela estive relacionada como capa do álbum de algum cantor. Mesmo assim o campo *música* seria mais apropriado ser colocado na tabela que fosse mais relacionada ao tema dele como cantor ou músicas preferidas. É seguindo está logica que se dá o modelo relacional, cada tabela e criada sob um contexto e todos elas são inter-relacionadas.

De acordo com Carvalho (2018), uma das principais razões do modelo relacional ser bastante utilizado e na facilidade de modificação nas estruturas das tabelas.

3.6 MongoDB

O nome Mongo vem da expressão inglesa *Humongous* (Anderson, 2015), que descreve algo como algo gigantesco, avantajado ou monstruoso (Collins, 2018).

É um banco de dados não relacional orientado a documentos. Documentos podem ter todo o tipo de informação, ao contrário do modelo relacional cada documento pode possuir um tipo de informação que outro não possua. As informações no modelo relacional precisam seguir à risca como as tabelas foram montadas em sua concepção já no modelo não relacional ou NoSQL (Not Only SQL – Não Apenas SQL) Anderson (2018), cada documento tem sua própria informação não utilizando memoria extra para informações que não necessitem serem gravadas para determinado documento.

O MongoDB foi utilizado neste projeto para salvar os modelos de redes neurais que serão programadas com o resultado do treinamento, já que o mesmo leva algum tempo para executar. Assim é mais fácil recuperar os dados da rede neural artificial já treinada e assim classificar novas entradas. O modelo não-relacional serve perfeitamente para este propósito pois cada modelo de rede neural artificial segue suas próprias regras podendo um modelo possuir campos ou atributos diferentes de outros.

4. Redes Neurais Artificiais

A base das redes neurais artificiais se dá pelas entradas, os pesos, a função de integração ou soma que pondera os valores das entradas com os pesos e a função de ativação que irá determinar se a saída é ou não o resultado esperado.

4.1 Entradas

As entradas da rede neural artificial determinam-se de acordo com o problema que será colocado para que ela aprenda. Geralmente os valores são determinados entre 0 e 1 ou entre -1 e 1 dependendo do problema escolhido. É possível trabalhar com valores maiores, porém, é necessário utilizar uma função de ativação que cubra esses valores.

4.2 Pesos

Os pesos são vetores ou matrizes correspondentes ao número de entradas do neurônio. Para cada entrada existe um peso, geralmente inicializados randomicamente. O Processo de aprendizagem das redes neurais artificiais está na correção dos pesos.

A correção dos pesos se dá calculando a diferença da saída desejada com a saída atual. Este processo pode variar de acordo com algoritmo de aprendizagem. Os pesos é onde fica armazenado o conhecimento da rede (GAGO, 2015).

4.3 Soma (Integração)

A soma ou função de integração recebe a multiplicação de todas as entradas com seus respectivos pesos.

4.4 Bias

O Bias é uma entrada com valor constante. Acelera o processo de treinamento da rede. Sem o *bias* o processo de aprendizagem da rede neural pode nunca acontecer (GAGO, 2015).

Segundo Haykin (2001), o Bias tem o efeito de aumentar ou diminuir a entrada líquida da função de ativação, dependendo se ele é positivo ou negativo.

De acordo com COUTO (2017), o bias tem a função mover o sigmóide sem deixar que ele se contorça.

4.5 Funções de Ativação

A função de ativação define a saída do neurônio, podendo ser ou não resposta final da rede.

Haykin (2001), descreve que a função de ativação restringe o intervalo permissível de amplitude do sinal de saída a um valor finito.

Alguns tipos básicos a seguir:

4.5.1 Função Linear

- É uma função que não altera a saída do neurônio.
- Representação: *f(x) = x*.

4.5.2 Função Degrau

- Conhecida também como função de Heaviside. Haykin (2001) se refere a ela como função de limiar.
- Esta função retorna 1 se a saída for maior ou igual a zero, ou, zero se a saída for menor que zero. É possível que essas definições se modifiquem dependendo do problema em que se usa esta função.
- Representação: *f(x) = (1 se x >= 0), (0 se x < 0).*

A Figura 12, mostra graficamente a função degrau.

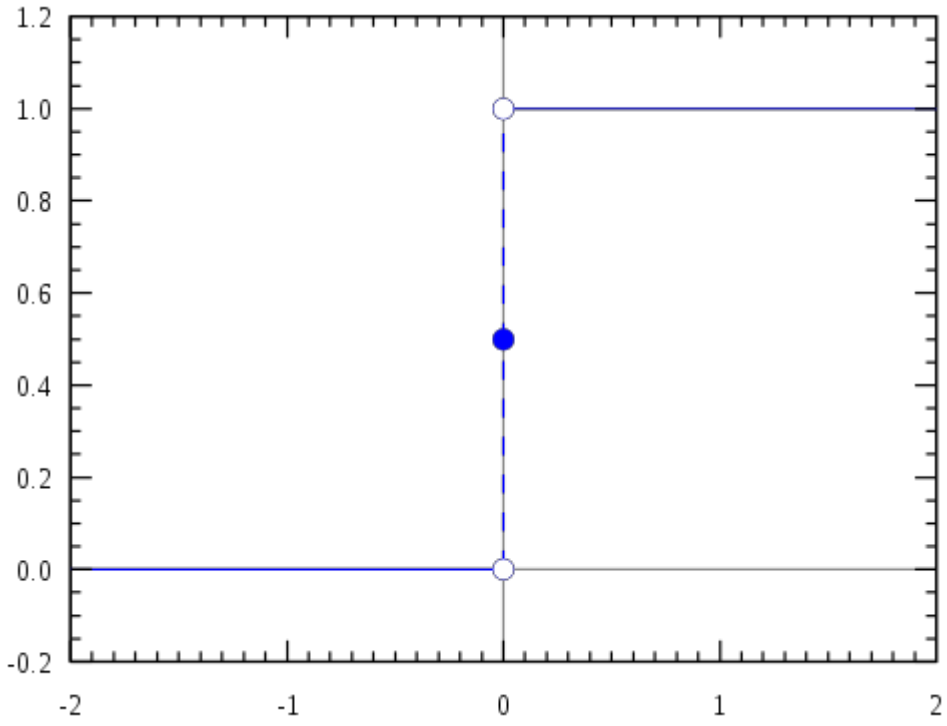

Figura 12. Função degrau (heaviside).

Fonte: https://pt.wikipedia.org/wiki/Fun%C3%A7%C3%A3o_de_Heaviside#/media/File:Dirac_distribution_CDF.svg

4.5.3 Sigmóide

- Esta função é usada em redes neurais que produzam apenas saídas positivas. Ela recebe um valor como entrada e normaliza entre 0 e 1.
- Segundo Haykin (2001) a função sigmóide, cujo gráfico tem a forma de um “S”, é a função mais utilizada na construção de uma rede neural artificial. É uma função estritamente crescente adequada para comportamentos Linear e não Linear.

- Representação: *f(x) = 1 / (1 + exp(-x)).*

A figura 13, mostra a representação da função sigmodal no plano estabilizando valores de 0 a 1.

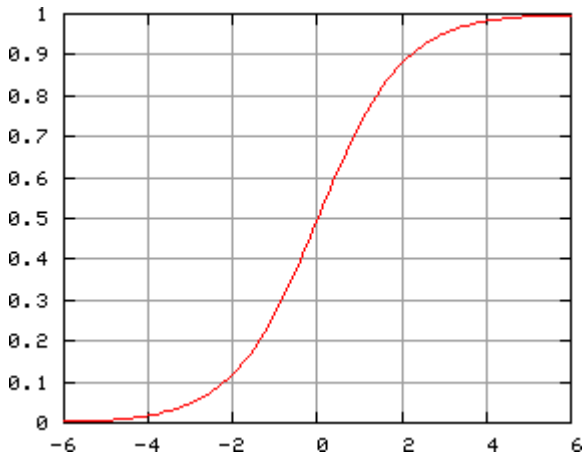

Figura 13. Função de Sigmóide.

Fonte: https://upload.wikimedia.org/wikipedia/commons/a/ac/Logistic-curve.png

4.5.4 Tangente Hiperbólica

- A função tangente hiperbólica permite que a função de ativação tipo sigmóide assuma valores negativos de acordo com Haykin (2001).
- Esta função é muito comum em redes neurais em que as saídas precisam ser entre -1 e 1.
- Representação: *f(x) = tanh(x).*

A Figura 14, apresenta graficamente a função tangente hiperbólica no plano normalizando os valores de -1 a 1.

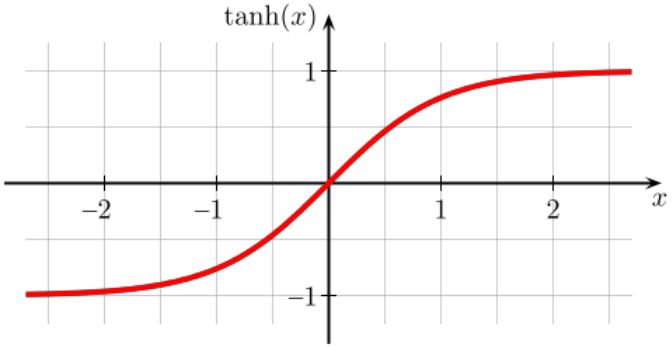

Figura 14. Tangente hiperbólica.

Fonte: https://upload.wikimedia.org/wikipedia/commons/8/87/Hyperbolic_Tangent.svg

O apêndice B, apresenta o código fonte com as funções de ativação que foram utilizadas no trabalho.

4.6 Histórico

O desenvolvimento das redes neurais artificiais aconteceu após a publicação de um artigo “A Logical Calculus of the Ideas Immament in Nervous Activity" em 1943, pelo psiquiatra e neuroanatomista Warren McCulloch e o matemático Walter Pitts. O trabalho de McCulloch e Pitts se concentra muito mais em descrever um modelo artificial do que em técnicas de aprendizado (Braga, Ludermir e Carvalho, 2000).

Neste artigo McCulloch e Pitts descrevem o cálculo lógico das redes neurais que unificava os estudos da neurofisiologia e a lógica matemática, (Haykin, 2001).

Em 1949 Donald Hebb mostrou como o processo de aprendizagem das redes neurais é conseguida através dos pesos de entrada dos neurônios e em 1958, Frank Rosenblatt desenvolveu um dos modelos de redes neurais mais básicos para resolver problemas lineares, o Perceptron Simples (Braga, Ludermir e Carvalho, 2000).

Em 1969, Minsky e Papert mostraram que o Perceptron não era capaz de executar algumas tarefas já que ele só resolve problemas linearmente separáveis, Braga, Ludermir e Carvalho (2000) ainda descrevem que por causa disso nos anos 70, o estudo das redes neurais conexionistas ficou adormecida devido a repercussão do trabalho de Minsky e Papert.

Em meados dos anos 80 houve uma explosão de interesse nas RNA por causa do rápido avanço das tecnologias, que permitia aumentar a quantidade de neurônios e camadas (Braga, Ludermir e Carvalho 2000).

Em 1982 Teuvo Kohonen publicou um artigo sobre os Mapas Auto-Organizáveis, utilizando uma rede bidimensional ou unidimensional (Braga, Ludermir e Carvalho 2000).

4.7 Tipos de Redes Neurais

Uma rede neural é escolhida de acordo com o problema que será proposto a ela. Para isto existe diferentes tipos. Antes de entrar mais a dentro nos modelos, há duas formas de aprendizagem que podem ser construídas para que uma RNA aprenda. Aprendizagem supervisionada e a não supervisionada.

4.8 Aprendizagem Supervisionada

Como diz o nome, na aprendizagem supervisionada, é necessária que um supervisor (programador) forneça as entradas e as saídas para a rede neural e verifique as saídas produzidas por ela, modificando os parâmetros até que ela forneça os resultados desejados. O objetivo é encontrar uma ligação entre os pares de entrada e saída fornecidos (Braga, Ludermir, Carvalho 2000).

4.9 Aprendizagem Não Supervisionada

Na aprendizagem não supervisionada, não é necessário um supervisor (programador) para ajustar os parâmetros da rede, pois apenas as entradas são fornecidas e a aprendizagem irá depender do algoritmo utilizado.

Este tipo de aprendizagem só se torna possível quando os dados de entrada são redundantes pois sem isto seria impossível encontrar quaisquer padrões ou características da rede (Braga, Ludermir e Carvalho 2000).

5. Modelos de Redes Neurais

Existem diferentes tipos de redes neurais como as redes recorrentes, as redes de Hopfield e a CNN conhecida como redes neurais convolucionais atualmente a melhor rede lidar com classificação de imagens.

Neste trabalho foram pesquisados os modelos Perceptron Simples ao qual foi implementado para o reconhecimento de padrões de caracteres numéricos. O Perceptron de múltiplas camadas e os mapas auto organizáveis de Kohonen. A seguir será apresentado um pouco melhor as características de cada uma delas.

5.1 Perceptron simples

Perceptron simples é uma rede criada por Rosenblatt (1958) que propôs o primeiro modelo para treinamento supervisionado (Braga, Ludermir e Carvalho 2000).

A rede neural Perceptron é a rede mais básica usada para classificar padrões linearmente separáveis. Basicamente ela consiste de um único neurônio com pesos sinápticos ajustáveis e bias, Haykin (2001).

Na Figura 15 Perceptron simples, mostra a representação em grafo do modelo Perceptron Simples.

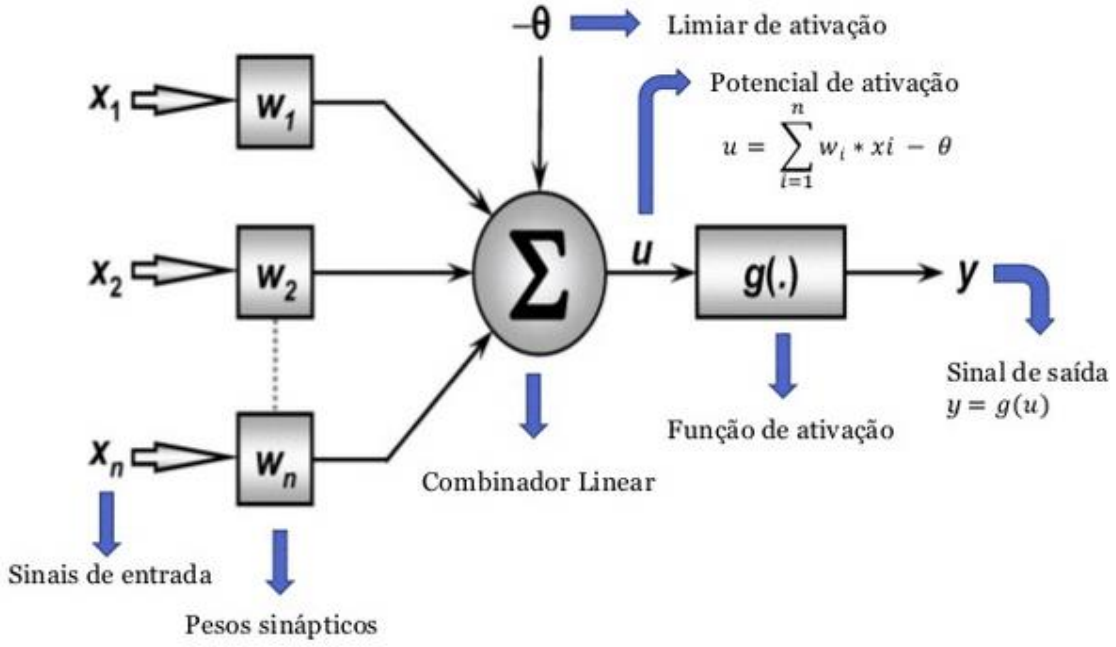

Figura 15. Perceptron simples.

Fonte: https://www.embarcados.com.br/wp-content/uploads/2016/09/Perceptron-01.png

O Apêndice A apresenta o código fonte para o modelo de rede neural perceptron simples.

5.2 Perceptron Multicamadas

Esta rede é usada para resolver problemas mais complexos não linearmente separáveis e consiste da camada de entrada, uma ou mais camadas intermediárias e uma camada de saída. O treinamento desta rede se dá de forma supervisionada utilizando um algoritmo conhecido como algoritmo de retro propagação de erro (Back-Propagation). Este algoritmo é baseado na regra por correção de erro, Haykin (2001).

A Figura 16 Perceptron Multicamadas, é representada por grafos, com uma camada de entrada (input layer), duas camadas intermediárias chamadas de camadas ocultas (hidden layer) e uma camada de saída (output layer).

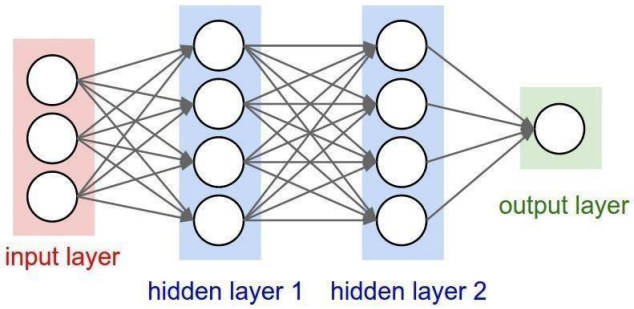

Figura 16. Perceptron multicamadas.

Fonte: https://cdn-images-1.medium.com/max/800/1*Z3zHoX1nhK6Rsmd4yNPdsg.jpeg

De acordo com Braga, Ludermir e Carvalho (2000), para treinar este tipo de rede foi proposto o método baseado no gradiente descendente. A função de ativação precisa ser contínua, diferenciável e não-decrescente, informando os erros produzidos pela saída da rede para as camadas anteriores com maior precisão possível. A função de ativação mais utilizada é a função sigmodal logística (Sigmóide).

5.3 Mapas Auto Organizáveis (Kohonen SOM)

Este é um tipo de rede que utiliza o processo de aprendizagem não supervisionada. Para o treinamento desse tipo de rede neural não é necessário apresentar os valores de saída da rede. Ao invés disso é o algoritmo que irá procurar as semelhanças entre os padrões de entradas e agrupar padrões parecidos sem conhecer as classes de antemão.

A rede SOM (self organizing maps – mapas auto organizáveis) é uma rede neural de duas camadas que aceita padrões de N-dimensões como entrada e os mapeia para um conjunto de neurônios de saída.

Uma das formas de treinamento desta rede está no aprendizado competitivo. Os neurônios de saída competem entre si pelo direito de atualizar seus pesos, (Braga, Ludermir e Carvalho, 2000).

A Figura 17 Mapas auto organizáveis, apresenta a camada de entrada e a camada de saída classificando as saídas produzidas pela rede.

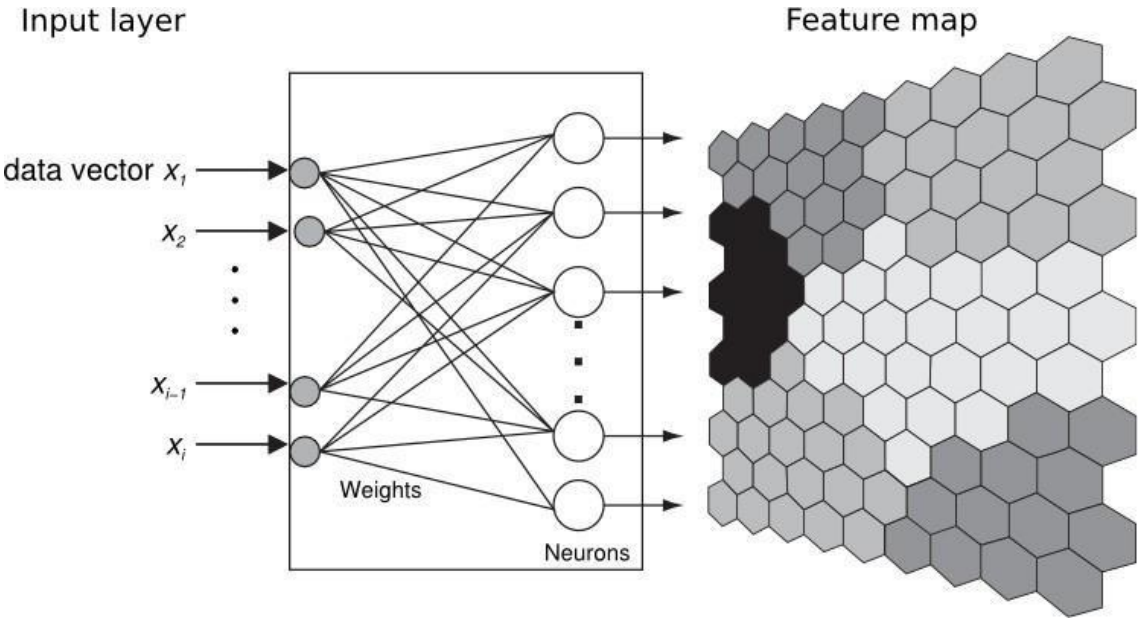

Figura 17. Mapas auto organizáveis.

Fonte: https://i.stack.imgur.com/FNaEm.png

5.4 Explorando o Perceptron Simples

Neste trabalho, para a construção do modelo Perceptron foi utilizado a linguagem de programação Java dentro do ambiente de desenvolvimento Eclipse IDE.

O problema escolhido para o primeiro teste foi um dos mais comuns: o “ou” lógico.

O “ou” lógico na programação define que se as duas entradas são verdadeiras então o resultado é verdadeiro, se uma entrada é verdade e outra não, o resultado é verdadeiro, e é somente falso quando todas as entradas são falsas.

A figura 18, mostra a tabela verdade do “ou” lógico computacional;

p	q	p v q
1	1	1
0	1	1
1	0	1
0	0	0

Figura 18. Tabela verdade “ou” lógico.
Fonte: Próprio autor.

Com esses dados foi possível estabelecer as entradas e as saídas desejadas da rede neural Perceptron.

As entradas são as duas primeiras colunas da matriz tabela verdade e a saída desejada é a terceira coluna que é o resultado da logica da linha correspondente a ela.

Com isso em mente a primeira classe criada no Java para mapear essas entradas e a saída foi a “TrainingSet” que do inglês significa como um conjunto de treinamento.

A figura 19, mostra a classe TrainingSet que será responsável por mapear as entradas e saídas do Perceptron. O atributo “entradas” é um vetor pois cada linha da matriz da tabela da verdade possui dois valores de entrada (p, q) e um de saída que é o resultado do cálculo lógico (p $\vee$ q), o código fonte completo da classe se encontra no apêndice C.

```
public class TrainingSet {

    private double[] entradas;
    private double saida;

    public TrainingSet(double[] entradas, double saida) {
        this.entradas = entradas;
        this.saida = saida;
    }
```

Figura 19. Classe TrainingSet (Conjunto de Treinamento).

Fonte: Próprio autor.

Este problema é linearmente separável, retornando resultados binários de 0 ou 1. Entendendo isto é possível perceber que os resultados não passarão de 0 e 1 e existe uma função de ativação para lidar com esse tipo de problema. A função escolhida foi a Sigmóide.

A figura 20, apresenta a Classe “ActivationFunction” que do inglês significa Função de Ativação para abrigar as funções de ativação que serão utilizadas.

```
public class ActivationFunction {

    public static double sigmoid( double x ) {
        return 1 / (1 + Math.exp(-x));
    }

    public static double linear( double x ) {
        return x;
    }

    public static double heaviside ( double x ) {

        if( x >= 0) {
            return 1;
        }

        return 0;
    }

    public static double tangenteHiperbolica( double x ) {
        return Math.tanh(x);
    }
```

Figura 20. Classe “ActivationFunction” (função de sigmóide destacada no retângulo vermelho). Fonte: Próprio autor.

Agora que as entradas e as funções de ativação foram preparadas já é possível construir a Classe principal que será a rede neural artificial Perceptron.

O classe possui dois atributos principais, o primeiro: os pesos, que também é um vetor como as entradas, entretanto, deve se pensar que como são quatro linhas com duas colunas de entradas, então os pesos também são equivalentes e daria 8 pesos finais. Após muitos testes foi descoberto que são, apenas 2 pesos que irão passar pelas 4 linhas da matriz pois se cada linha possuir seus respectivos pesos apenas as entradas que correspondem aquela linha específica irão retornar o resultado desejado. Logo a rede neural se torna inútil, porque ela deve retornar o valor desejado independente de qual entrada for a escolhida. Logo os pesos devem ter o tamanho de apenas uma das linhas de entradas e neste caso é apenas 2.

Existe também um atributo Bias que é essencial para o treinamento da rede neural.

Figura 21, Classe Perceptron (atributos).

```
public class Perceptron {

    private double[] pesos;
    private double bias;

    private int NP; // numero de pesos

    private List<TrainingSet> conjuntoTreinamento;

    private int NTS; // numero de amostras de treinamento

    private int iteracoes;
```

Figura 21. Classe Perceptron (atributos).

Fonte: Próprio autor.

A figura 22, apresenta o construtor da Classe que recebe um conjunto de treinamento, o número de iterações que será utilizado no treinamento e também inicializa os pesos através de um método privado.

```
public Perceptron(List<TrainingSet> training_sets, int i) {

    System.out.println("Perceptron Simples");

    this.conjuntoTreinamento = training_sets;

    this.NTS = training_sets.size();
    this.NP = training_sets.get(0).getEntradas().length;

    this.pesos = new double[this.NP];

    iniciarPesos();

    this.iteracoes = i;

    System.out.println("Iniciando treinamento...");

}
```

Figura 22. Classe Perceptron (Construtor).

Fonte: Próprio autor.

Os pesos são iniciados randomicamente, pois, como eles serão corrigidos no treinamento não é necessário determinar um valor inicial. Segundo GAGO (2015) não é recomendável que se inicialize os pesos com 0.

Seguindo a lógica do problema escolhido para a rede os pesos são inicializados com valores entre e 0 e 1 com uma função da biblioteca java, e aproveitando, o bias também é inicializado randomicamente.

A figura 23, mostra o método privado que inicializa os pesos e o bias com valores randômicos entre 0 e 1.

```
private void iniciarPesos() {
    this.bias = Math.random();
    for(int i = 0; i < this.NP; i++) {
        this.pesos[i] = Math.random();
    }
}
```

Figura 23. Perceptron (método para inicializar os pesos e o bias).

Fonte: Próprio autor.

Após os pesos inicializados é o momento de ponderar as entradas pelos pesos através do método de integração retratada na figura 24. Este método recebe uma linha por vez do conjunto de treinamento.

```
public double executar(TrainingSet ts) {
    double soma = 0.0;
    for(int i = 0; i < this.NP; i++) {
        soma += ts.getEntradas()[i] * this.pesos[i];
    }
    soma += this.bias;
    return ActivationFunction.sigmoid(soma);
}
```

Figura 24. Perceptron (método executar).

Fonte: Próprio autor.

O método executar é o responsável por determinar a saída final da rede neural, entretanto, este resultado pode não ser o desejado pois os pesos ainda não foram corrigidos. A correção dos pesos se dá pela diferença da saída desejada com a saída atual que é obtida através do método executar, o valor obtido é multiplicado pela entrada correspondente ao peso e somado a ele. Para isso foi criado o próximo método que corrige os pesos mostrado na figura 25.

```
private void corrigirPesos(TrainingSet ts) {
    double y = executar(ts);
    for(int i = 0; i < this.NP; i++) {
        this.pesos[i] += (ts.getSaida() - y) * ts.getEntradas()[i];
    }
    this.bias += (ts.getSaida() - y);
}
```

Figura 25. Perceptron (método para corrigir os pesos).

Fonte: Próprio autor.

Para que a rede neural seja treinada adequadamente é necessário determinar um número de iterações e repetir todo o processo de integração das entradas com os pesos, até a saída produzida pela função de ativação e a correção dos pesos. Todo o processo é conhecido como época e em cada época os pesos são corrigidos para todas as entradas fornecidas no conjunto de treinamento.

A figura 26, mostra a função de treinamento.

```
public void treinar() {

    int epocas = 0;

    do {

        for(int i = 0; i < this.NTS; i++) {

            corrigirPesos(this.conjuntoTreinamento.get(i));

            double saida = executar(this.conjuntoTreinamento.get(i));
            double alvo = this.conjuntoTreinamento.get(i).getSaida();

            System.out.printf("Saida: %.8f -> Alvo: %.1f \n", saida, alvo);
        }

        epocas++;
    } while (epocas < this.iteracoes);

    System.out.println("Treinamento completo!");
}
```

Figura 26. Perceptron (método de treinamento).

Fonte: Próprio autor.

Um método adicional foi criado para classificar os valores que forem fornecidos após o treinamento. Ao observar o operador ternário calculando se a saída é menor que 0.5 conclui-se que o resultado tem que ser maior que 50% para ser verdadeiro senão este é falso, ou seja, se for esperado que o resultado seja verdadeiro com a saída desejada 1 e o resultado obtido é abaixo de 0.5 significa que o neurônio não está treinado ou, há alguma falha na programação.

Figura 27, método de classificação.

```
public String classificar(TrainingSet ts) {

    String y = "";

    double soma = 0.0;
    for(int i = 0; i < this.NP; i++) {

        soma += (ts.getEntradas()[i] * this.pesos[i]);

    }
    soma += this.bias;

    y = (ActivationFunction.sigmoid(soma) < 0.5)? "FALSE" : "TRUE";

    return y;

}
```

Figura 27. Perceptron (método de classificação).

Fonte: Próprio autor.

5.5 Perceptron – Fase de Testes

Agora é o momento de observar se a rede neural realmente funciona. Para isso é necessário informar o conjunto de treinamento da rede mostrado na figura 28, é possível observar que o conjunto de treinamento está idêntico a tabela verdade do “ou” lógico destacado no retângulo em vermelho.

```
public class PerceptronTest {

    public static void main(String[] args) {

        List<TrainingSet> training_sets = new ArrayList<>();

        training_sets.add(new TrainingSet(new double[]{ 0, 0}, 0));
        training_sets.add(new TrainingSet(new double[]{ 0, 1}, 1));
        training_sets.add(new TrainingSet(new double[]{ 1, 0}, 1));
        training_sets.add(new TrainingSet(new double[]{ 1, 1}, 1));
```

Figura 28. Conjunto de treinamento (“ou” lógico).

Fonte: Próprio autor.

Um objeto da classe Perceptron é instanciado para começar o treinamento e é definido o valor de 1000 iterações para treinar a rede neural, o código fonte completo da classe de execução encontra-se no apêndice D.

O valor de 1000 épocas e determinado mas não precisa ser o valor definitivo para todas as redes neurais. Este valor pode ser aumentado ou diminuído de acordo com a estrutura da rede e o resultado do treinamento.

A figura 29, mostra a rede neural Perceptron instanciada recebendo o conjunto de treinamento e sendo treinada por mil épocas.

```
Perceptron net = new Perceptron(training_sets, 1000);
net.treinar();
```

Figura 29. Perceptron (treinamento).

Fonte: Próprio autor.

Ao começar o treinamento é possível observar que os valores iniciais são aleatórios sendo que o valor de saída é o resultado produzido pela função de ativação e o alvo é o valor desejado que se esperava que o neurônio artificial produzisse.
Os valores de saída ainda não estão próximos aos valores desejados, principalmente a entrada (0, 0) em que a saída precisa ser 0.

A figura 30, mostra as três primeiras épocas de treinamento do neurônio artificial.

```
Epoca 1
Saida: 0,54479403 -> Alvo: 0,0
Saida: 0,82225136 -> Alvo: 1,0
Saida: 0,77893702 -> Alvo: 1,0
Saida: 0,93111869 -> Alvo: 1,0

Epoca 2
Saida: 0,54988532 -> Alvo: 0,0
Saida: 0,85503136 -> Alvo: 1,0
Saida: 0,81642727 -> Alvo: 1,0
Saida: 0,95372896 -> Alvo: 1,0

Epoca 3
Saida: 0,51595434 -> Alvo: 0,0
Saida: 0,86500031 -> Alvo: 1,0
Saida: 0,83255989 -> Alvo: 1,0
Saida: 0,96542630 -> Alvo: 1,0
```

Figura 30. Três primeiras épocas de treinamento.

Fonte: Próprio autor.

Com 100 épocas os valores já estão mais próximos das saídas desejadas.

A figura 31, treinamento após 100 épocas.

```
Epoca 100
Saida: 0,04802122 -> Alvo: 0,0
Saida: 0,98096372 -> Alvo: 1,0
Saida: 0,98062028 -> Alvo: 1,0
Saida: 0,99998027 -> Alvo: 1,0
```

Figura 31. Treinamento após 100 épocas.

Fonte: Próprio autor.

A figura 32, mostra o resultado final do treinamento.

```
Epoca 1000
Saida: 0,00499494 -> Alvo: 0,0
Saida: 0,99800359 -> Alvo: 1,0
Saida: 0,99799965 -> Alvo: 1,0
Saida: 0,99999998 -> Alvo: 1,0
```

Figura 32. Treinamento completo com sucesso.

Fonte: Próprio autor.

Após o treinamento já é possível verificar se o neurônio responde corretamente para uma entrada escolhida pelo usuário.

Foi instanciado um novo objeto TrainingSet e para este foi informado os valores 0 e 1. Agora que a rede neural esta treinada o valor de saída não importa e vai ser ignorado pelo método de classificação. Para essas entradas o resultado final precisa ser True (Verdadeiro) pois pela tabela verdade 0 ou 1 é igual a 1.

A figura 33, apresenta as entradas e o resultado final classificado com sucesso.

```
22
23
24          System.out.println("Classificando...");
25
26          System.out.println("Resultado: " +
27                  net.classificar(new TrainingSet(new double[] {0, 1}, 0)
28                          ));
29
30      }
31  }
32
```

```
Console
<terminated> PerceptronTest (1) [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (30 de jun de 2018 00:51:48)
Perceptron Simples
Iniciando treinamento...
Treinamento completo!
Classificando...
Resultado: TRUE
```

Figura 33. Classificação.

Para prever o impacto do Bias no treinamento do neurônio artificial, a soma do bias foi retirado do método.

A primeira entrada não foi treinada e o valor ficou estabilizado em 0.5, isso acontece por causa da função de sigmóide, que como foi explicado anteriormente, o bias tem o papel de movimentar o resultado da função sigmóide para frente e para trás sem deixar que ele se contorça.

A figura 34, mostra os resultados do treinamento do neurônio artificial sem o bias.

```
Epoca 1000
Saida: 0,50000000 -> Alvo: 0,0
Saida: 0,99900589 -> Alvo: 1,0
Saida: 0,99900546 -> Alvo: 1,0
Saida: 0,99999901 -> Alvo: 1,0
```

Figura 34. Treinamento sem o Bias falhou.

Com o neurônio treinado ele é salvo no banco de dados não relacional MongoDB, para que assim os valores dos pesos corrigidos possam ser utilizados para uma próxima classificação sem precisar do processo de treinamento novamente. O apêndice F mostra o código fonte de como foi persistido os dados no MongoDB.

A figura 35, mostra o modelo de rede neural com os pesos treinados salvo no banco dados MongoDB.

Prompt de Comando - mongo

```
> db.rna.find().pretty();
{
        "_id" : ObjectId("5b37013e1f04751928bd9f43"),
        "modelo" : "Perceptron Simples",
        "descricao" : "Resolve problemas Lineares",
        "problema" : "OR Lógico",
        "training_sets" : [
                {
                        "entradas" : [
                                0,
                                0
                        ],
                        "saida" : 0
                },
                {
                        "entradas" : [
                                0,
                                1
                        ],
                        "saida" : 1
                },
                {
                        "entradas" : [
                                1,
                                0
                        ],
                        "saida" : 1
                },
                {
                        "entradas" : [
                                1,
                                1
                        ],
                        "saida" : 1
                }
        ],
        "pesos" : [
                11.504794262205 98,
                11.508755070533104
        ],
        "iteracoes" : 1000
}
>
```

Figura 35. Perceptron salvo no MongoDB.

Fonte: Próprio autor.

5.6 Perceptron – Prevendo valores

O problema anterior foi um dos mais simples de se resolver, inclusive é possível modificar o conjunto de treinamento e utilizar a lógica do AND (E lógico) que o neurônio irá classificar corretamente. Entretanto os valores deste problema já foram definidos no início e não era necessário que o neurônio previsse o resultado.

Para que essa mesma rede neural conseguisse prever e classificar entradas independente dos valores do conjunto de treinamento passado a ela, foi implementado um plano cartesiano onde foram distribuídos alguns pontos e entre eles é traçado uma reta onde foi definido para a classe A os pontos verdes que estão acima da reta, e para a classe B, os azuis, que estão abaixo da reta.

A figura 36, mostra o plano cartesiano definindo as classes A e B.

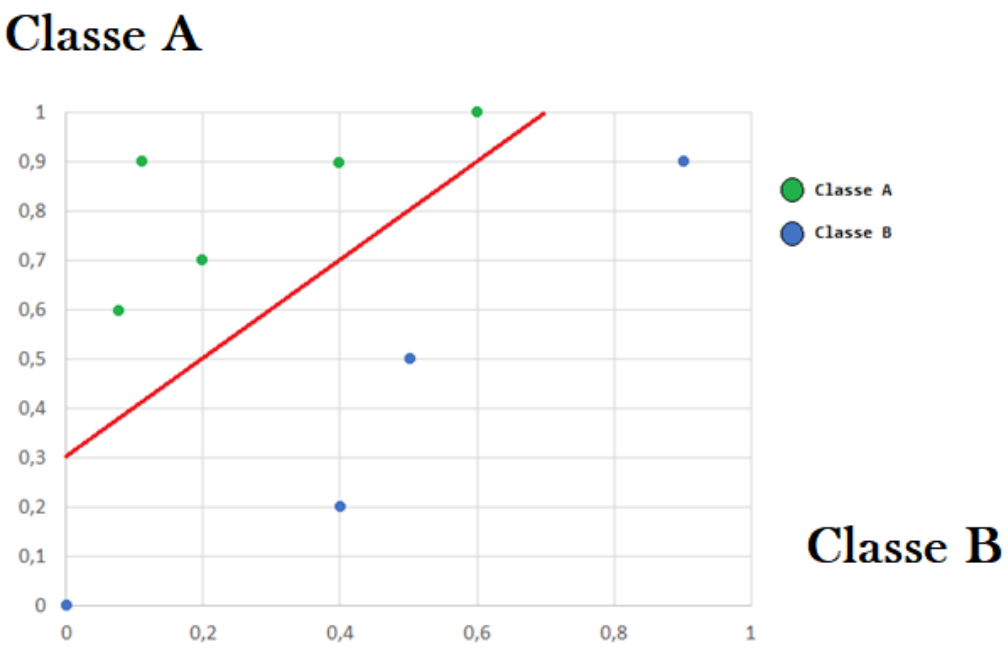

Figura 36. Plano Cartesiano.

Fonte: Próprio autor.

Alguns pontos presentes no plano, foram passados como amostras para o neurônio artificial, o mesmo teria que prever os pontos que não estavam presentes nas amostras e classifica-los corretamente, à qual classe eles pertenciam.

A figura 37, mostra os pontos selecionados para serem usados como amostras do Perceptron. Pontos para a classe A: (x = 0.08, y = 0.6), (x = 0.13, y = 0.9), (x = 0.2, y = 0.7), (x = 0.4, y = 0.9). Classe B: (x = 0.4, y = 0.2), (x = 0.5, y = 0.5).

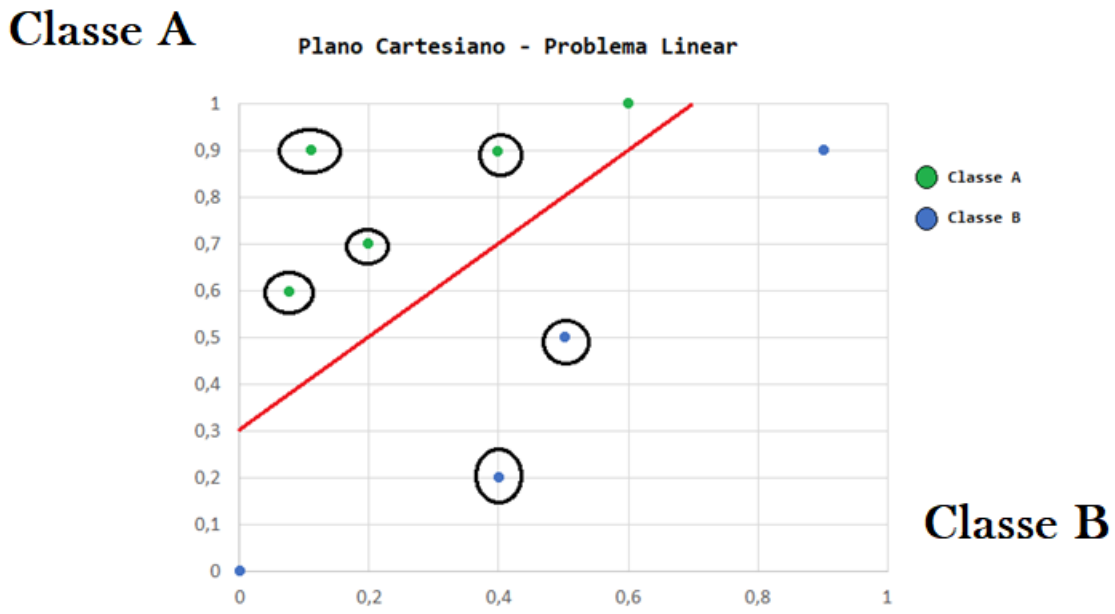

Figura 37. Pontos selecionados como amostra.

Fonte: Próprio autor.

É importante notar que esse plano trata de valores entre 0 e 1 por isso a função de sigmóide foi usada novamente.

A figura 38 representa os pontos como amostras da rede neural.

```
List<TrainingSet> training_sets = new ArrayList<>();

// Pontos da Classe A
training_sets.add(new TrainingSet(new double[]{ 0.08, 0.6}, 1));
training_sets.add(new TrainingSet(new double[]{ 0.13, 0.9}, 1));
training_sets.add(new TrainingSet(new double[]{ 0.2, 0.7},  1));
training_sets.add(new TrainingSet(new double[]{ 0.4, 0.9},  1));


// Pontos da Classe B
training_sets.add(new TrainingSet(new double[]{ 0.4, 0.2},  0));
training_sets.add(new TrainingSet(new double[]{ 0.5, 0.5},  0));
```

Figura 38. Plano Cartesiano (amostras de treinamento).

Fonte: Próprio autor.

Após o treinamento um novo ponto foi passado para o neurônio para que ele conseguisse classificar corretamente o ponto na classe A ou B.

A figura 39, mostra o resultado da classificação para o ponto (x = 0.9, y = 0.9) que não estava no conjunto de amostras, mas é possível ver no plano cartesiano que ele pertence a classe B.

```
24        Perceptron net = new Perceptron(training_sets, 1000);
25        net.treinar();
26
27
28        System.out.println("Classificando...");
29
30        double y = net.executar(new TrainingSet(new double[] {0.9, 0.9}, -1));
31
32        String result = (y < 0.5)? "B" : "A";
33
34        System.out.println("Classe " + result);
35
```

```
Console
<terminated> PerceptronTest (1) [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (1 de jul de 2018 00:20:54)
Perceptron Simples
Iniciando treinamento...
Treinamento completo!
Classificando...
Classe B
```

Figura 39. O Perceptron classificou corretamente.

Fonte: Próprio autor.

A Figura 40 apresenta o resultado do treinamento após mil épocas.

```
Epoca 1000
Saida: 0,99669337 -> Alvo: 1,0
Saida: 0,99996456 -> Alvo: 1,0
Saida: 0,99475829 -> Alvo: 1,0
Saida: 0,99385167 -> Alvo: 1,0
Saida: 0,00042932 -> Alvo: 0,0
Saida: 0,01487131 -> Alvo: 0,0
```

Figura 40. Amostras treinadas após mil épocas.

Fonte: Próprio autor.

Na figura 41, um laço “for” foi criado para percorrer todos os pontos do plano cartesiano e classificar os pontos das duas classes.

```
Perceptron net = new Perceptron(training_sets, 1000);
net.treinar();


System.out.println("Classificando...");


for(double x = 0.0; x <= 1.0; x += 0.1) {
    for(double y = 0.0; y <= 1.0; y += 0.1) {

        TrainingSet teste = new TrainingSet(new double[] {x, y}, -1);

        double r = net.executar(teste);
        String result = (r < 0.5)? "B":"A";
        System.out.printf("(x:%.2f, y:%.2f) => %s \n", x, y, result);

    }
}
```

Figura 41. Classificando todos os pontos do plano.

Fonte: Próprio autor.

Resultado do treinamento com 1000 épocas e a classificação de todos os pontos do plano cartesiano na figura 42:

```
Classificando...
0       (x:0,00, y:0,00) => B
1       (x:0,00, y:0,10) => B
2       (x:0,00, y:0,20) => B
3       (x:0,00, y:0,30) => A
4       (x:0,00, y:0,40) => A
5       (x:0,00, y:0,50) => A
6       (x:0,00, y:0,60) => A
7       (x:0,00, y:0,70) => A
8       (x:0,00, y:0,80) => A
9       (x:0,00, y:0,90) => A
10      (x:0,00, y:1,00) => A
11      (x:0,10, y:0,00) => B
12      (x:0,10, y:0,10) => B
13      (x:0,10, y:0,20) => B
14      (x:0,10, y:0,30) => B
15      (x:0,10, y:0,40) => A
16      (x:0,10, y:0,50) => A
17      (x:0,10, y:0,60) => A
18      (x:0,10, y:0,70) => A
19      (x:0,10, y:0,80) => A
20      (x:0,10, y:0,90) => A
21      (x:0,10, y:1,00) => A
22      (x:0,20, y:0,00) => B
23      (x:0,20, y:0,10) => B
24      (x:0,20, y:0,20) => B
25      (x:0,20, y:0,30) => B
26      (x:0,20, y:0,40) => B
27      (x:0,20, y:0,50) => A
28      (x:0,20, y:0,60) => A
29      (x:0,20, y:0,70) => A
30      (x:0,20, y:0,80) => A
31      (x:0,20, y:0,90) => A
32      (x:0,20, y:1,00) => A
33      (x:0,30, y:0,00) => B
34      (x:0,30, y:0,10) => B
35      (x:0,30, y:0,20) => B
36      (x:0,30, y:0,30) => B
37      (x:0,30, y:0,40) => B
38      (x:0,30, y:0,50) => B
39      (x:0,30, y:0,60) => A
40      (x:0,30, y:0,70) => A
40      (x:0,30, y:0,70) => A
41      (x:0,30, y:0,80) => A
42      (x:0,30, y:0,90) => A
43      (x:0,30, y:1,00) => A
44      (x:0,40, y:0,00) => B
45      (x:0,40, y:0,10) => B
46      (x:0,40, y:0,20) => B
47      (x:0,40, y:0,30) => B
48      (x:0,40, y:0,40) => B
49      (x:0,40, y:0,50) => B
50      (x:0,40, y:0,60) => B
51      (x:0,40, y:0,70) => A
52      (x:0,40, y:0,80) => A
53      (x:0,40, y:0,90) => A
54      (x:0,40, y:1,00) => A
55      (x:0,50, y:0,00) => B
56      (x:0,50, y:0,10) => B
57      (x:0,50, y:0,20) => B
58      (x:0,50, y:0,30) => B
59      (x:0,50, y:0,40) => B
60      (x:0,50, y:0,50) => B
61      (x:0,50, y:0,60) => B
62      (x:0,50, y:0,70) => B
63      (x:0,50, y:0,80) => A
64      (x:0,50, y:0,90) => A
65      (x:0,50, y:1,00) => A
66      (x:0,60, y:0,00) => B
67      (x:0,60, y:0,10) => B
68      (x:0,60, y:0,20) => B
69      (x:0,60, y:0,30) => B
70      (x:0,60, y:0,40) => B
71      (x:0,60, y:0,50) => B
72      (x:0,60, y:0,60) => B
73      (x:0,60, y:0,70) => B
74      (x:0,60, y:0,80) => B
75      (x:0,60, y:0,90) => A
76      (x:0,60, y:1,00) => A
77      (x:0,70, y:0,00) => B
78      (x:0,70, y:0,10) => B
79      (x:0,70, y:0,20) => B
80      (x:0,70, y:0,30) => B
80      (x:0,70, y:0,30) => B
81      (x:0,70, y:0,40) => B
82      (x:0,70, y:0,50) => B
83      (x:0,70, y:0,60) => B
84      (x:0,70, y:0,70) => B
85      (x:0,70, y:0,80) => B
86      (x:0,70, y:0,90) => B
87      (x:0,70, y:1,00) => A
88      (x:0,80, y:0,00) => B
89      (x:0,80, y:0,10) => B
90      (x:0,80, y:0,20) => B
91      (x:0,80, y:0,30) => B
92      (x:0,80, y:0,40) => B
93      (x:0,80, y:0,50) => B
94      (x:0,80, y:0,60) => B
95      (x:0,80, y:0,70) => B
96      (x:0,80, y:0,80) => B
97      (x:0,80, y:0,90) => B
98      (x:0,80, y:1,00) => B
99      (x:0,90, y:0,00) => B
100     (x:0,90, y:0,10) => B
101     (x:0,90, y:0,20) => B
102     (x:0,90, y:0,30) => B
103     (x:0,90, y:0,40) => B
104     (x:0,90, y:0,50) => B
105     (x:0,90, y:0,60) => B
106     (x:0,90, y:0,70) => B
107     (x:0,90, y:0,80) => B
108     (x:0,90, y:0,90) => B
109     (x:0,90, y:1,00) => B
110     (x:1,00, y:0,00) => B
111     (x:1,00, y:0,10) => B
112     (x:1,00, y:0,20) => B
113     (x:1,00, y:0,30) => B
114     (x:1,00, y:0,40) => B
115     (x:1,00, y:0,50) => B
116     (x:1,00, y:0,60) => B
117     (x:1,00, y:0,70) => B
118     (x:1,00, y:0,80) => B
119     (x:1,00, y:0,90) => B
120     (x:1,00, y:1,00) => B
```

Figura 42. Perceptron classificando as classes.

Fonte: Próprio autor.

Ao comparar com o plano cartesiano a maioria dos pontos foram classificados corretamente. Porém quanto mais próximo da reta for o ponto mais ambíguo e difícil é, para a rede neural artificial classifica-lo. Para visualizar este problema será fornecido um ponto próximo a reta mostrado na figura 43, onde x = 0.4 e y = 0.65, que pertencem à classe B.

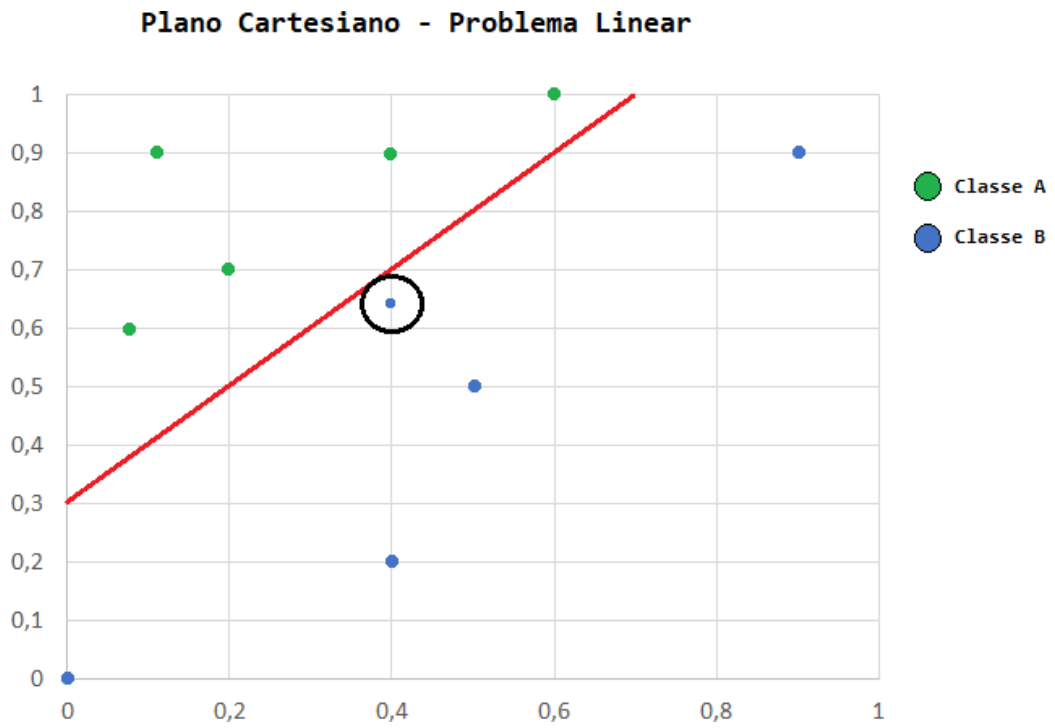

Figura 43. Pontos ambíguos.

Fonte: Próprio autor.

O resultado apresentado na figura 44, mostra que o Perceptron não conseguiu classificar corretamente o ponto.

```
23
24          Perceptron net = new Perceptron(training_sets, 1000);
25          net.treinar();
26
27
28          System.out.println("Classificando...");
29
30          TrainingSet test = new TrainingSet(new double[] {0.4, 0.65}, -1);
31
32          String result = (net.executar(test) < 0.5) ? "B" : "A";
33
34          System.out.println("Pertence a classe " + result);
```

```
Console
<terminated> PerceptronTest (1) [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (1 de jul de 2018 01:06:36)
Perceptron Simples
Iniciando treinamento...
Treinamento completo!
Classificando...
Pertence a classe A
```

Figura 44. Problema classificação ambígua.

Fonte: Próprio autor.

Para resolver este problema foi preciso melhorar o treinamento fornecendo mais amostras para a rede. Foi fornecida mais uma amostra para tentar corrigir este problema com um ponto mais próximo da reta (x = 0.6, y = 0.85) que será associado para a classe B.

A figura 45, mostra que a modificação obteve êxito.

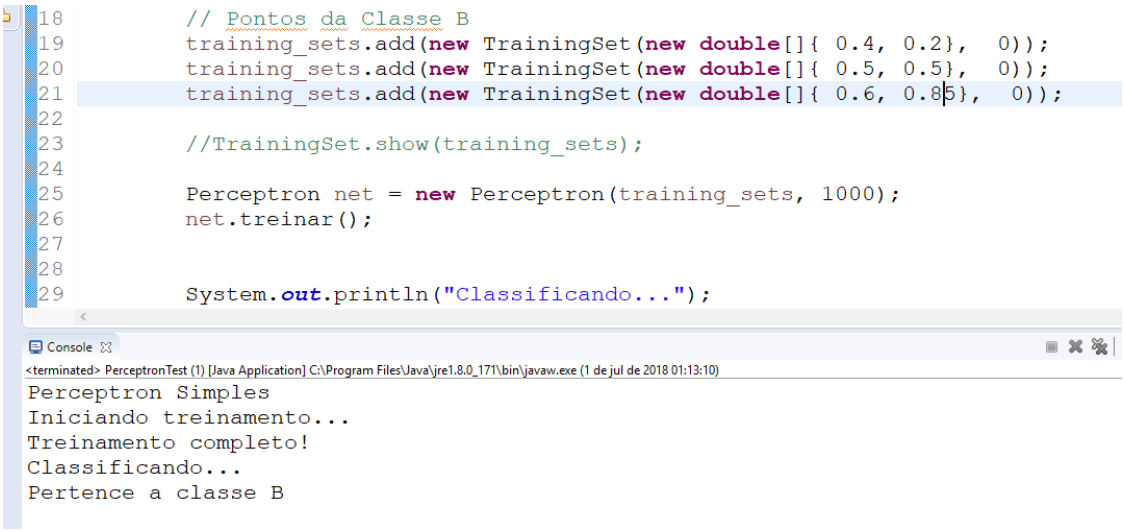

Figura 45. Perceptron melhorado.

Fonte: Próprio autor.

Após o treinamento com a nova amostra o Perceptron conseguiu classificar o ponto corretamente. O apêndice E mostra o código fonte completo utilizado para estes testes.

Estes foram os testes iniciais e os problemas encontrados durante o estudo e a pesquisa da rede neural artificial Perceptron.

Antes de passar para o próximo teste é necessário entender um pouco de visão computacional e processamento de imagens.

6. Processamento de Imagens

Como neste projeto foram apresentadas imagens para que as Redes Neurais Artificiais reconheçam padrões de caracteres numéricos presentes em imagens é necessário entender como o computador “enxerga” uma imagem.

O processamento de imagens digitais abrange uma ampla escala de hardware, software e fundamentos teóricos (GONZALEZ e WOODS, 2013).

6.1 Matriz e Imagem

Uma matriz é uma estrutura matemática representada por linhas e colunas, como se fosse uma tabela, m(i, j). Cada linha e coluna possuem um valor que irão delimitar as posições de cada bloco dessa tabela. Se tratando de imagens esses “blocos” são chamados de “Pixel”.

A figura 46, mostra a estrutura de uma matriz.

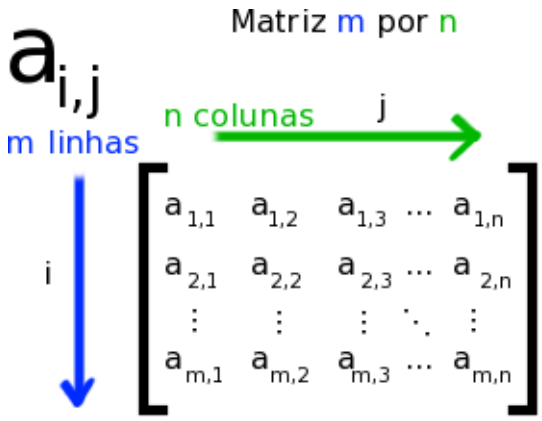

Figura 46. Matriz.

Fonte: https://upload.wikimedia.org/wikipedia/commons/d/d8/Matriz_organizacao.png

Nas palavras de GONZALEZ e WOODS (2013), uma imagem digital pode ser considerada com uma matriz cujo os índices de linhas e de colunas identificam um ponto na imagem, e o correspondente valor do elemento da matriz identifica o nível de cinza naquele ponto.

6.2 Pixel

A palavra Pixel vem de uma junção de palavras “Picture” e “Element” que numa tradução livre significa “Elemento de Imagem” (GONZALEZ e WOODS, 2013). O pixel é um elemento encontrado tanto em imagens quando em dispositivos eletrônicos. Ele é o menor ponto de uma imagem. Quanto mais uma imagem é aproximada, mais é possível identificar os mínimos detalhes dos pixels. A cor de um pixel é representada por valores RGB.

A figura 47, destaca os pixels de uma imagem e mostra seus valores RGB.

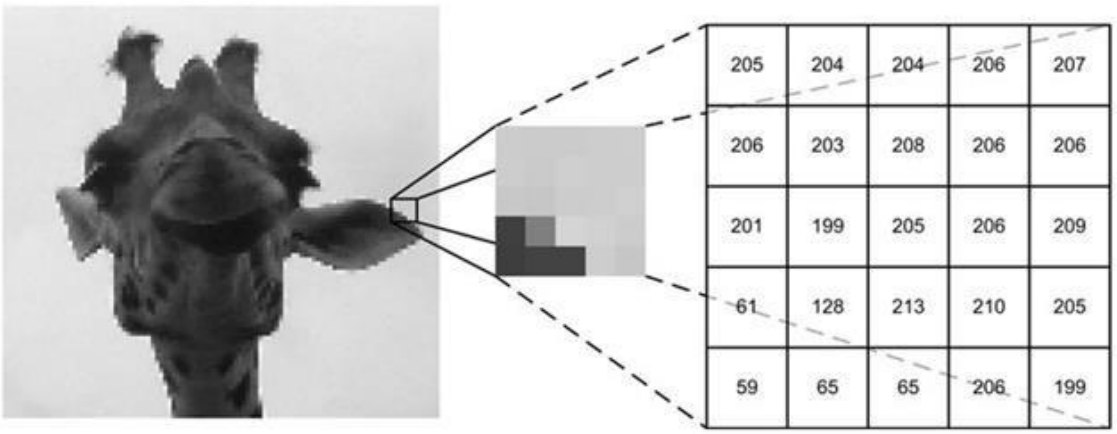

Figura 47. Pixels de uma imagem.

Fonte: http://what-when-how.com/wp-content/uploads/2012/07/tmp26dc133_thumb.png

As imagens digitais sempre são exibidas com a utilização de pixels. Elas são convertidas em pixels para que os monitores possam reproduzi-las, (JORDÃO, 2011).

6.3 RGB

No modelo RGB (Red, Green, Blue – Vermelho, Verde, Azul), cada cor aparece em componentes primários de vermelho, verde e azul, (GONZALEZ e WOODS, 2010).

Considerando uma imagem RGB na qual cada uma das imagens, vermelha verde e azul, seja uma imagem de 8 bits, GONZALEZ e WOODS (2013) explica que, cada pixel de cores RGB corresponde a um trio de valores que tem uma profundidade de 24 bits, significa que cada tonalidade dessa possui um byte, ou seja, cada tonalidade tem de 0 a 255 possibilidades de cores diferentes sendo que 0 é preto e 255 é branco. Ao juntar as três tonalidades que formam o padrão RGB é possível representar milhares de cores diferentes. Observa-se a quantidade de possibilidades fazendo o cálculo $(2^8)^3 = 16.777.216$.

A figura 48, mostra como estão dispostas as cores no padrão RGB em um cubo.

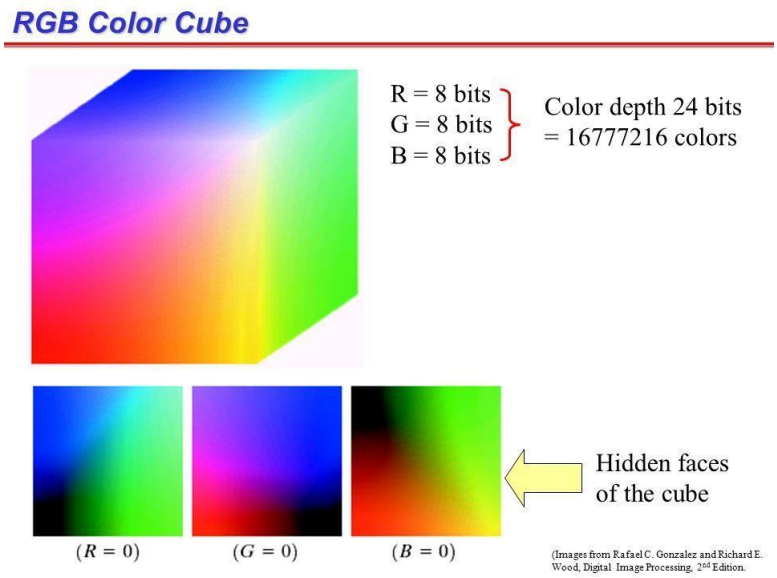

Figura 48. Cubo de cores RGB.

Fonte: https://slideplayer.com/slide/7577824/24/images/11/RGB+Color+Cube+R+=+8+bits+Color+depth+24+bits+G+=+8+bits.jpg

Este sistema de cores é usado por diversos objetos eletrônicos como televisões, monitores, câmeras entre outros.

Os apêndices H e I, apresentam os códigos fontes para lidar com as imagens.

6.4 Segmentação de imagens

Para que uma imagem possa ser enviada para a rede neural é necessário tratá-las, normalizando os valores RGB de cada pixel sem perder o conteúdo principal da imagem.

Alguns algoritmos foram desenvolvidos para fazer essa manipulação.

Para lidar com esses algoritmos uma interface foi criada no ambiente de desenvolvimento Eclipse em linguagem Java.

Os algoritmos utilizados foram adaptados de outros autores refenciados dentro do programa e no final do trabalho.

6.4.1 Tons de Cinza

Um tom de cinza é obtido quando os valores de vermelho, verde e azul do pixel são todos iguais, ou seja, a oscilação dos valores dos pixels é reduzida com valores mais constantes.

A Figura 49, Tons de Cinza, mostra a interface gráfica tratando a imagem que foi selecionada do sistema operacional e sendo convertida para tons de cinza.

Figura 49. Imagem em tons de cinza.

Fonte: Andrezza Lima Aragão

6.4.2 Sobel

O algoritmo de Sobel é uma técnica usada para realçar as bordas ou contornos de uma imagem. Porém não é o suficiente para usar como entrada de uma Rede Neural Artificial (SILVA, 2015).

Existem diferentes métodos para detecção de bordas em imagens. De acordo com CABRAL (2018), o filtro de Sobel provavelmente é o mais utilizado no processamento de imagens, mas, existem também o de Prewitt, Roberts e Canny, apontado como melhor método, porém exige maior esforço computacional.

A Figura 50 Imagem com bordas realçadas, mostra a interface gráfica tratando a imagem e destacando as bordas existentes nela, através do filtro de Sobel.

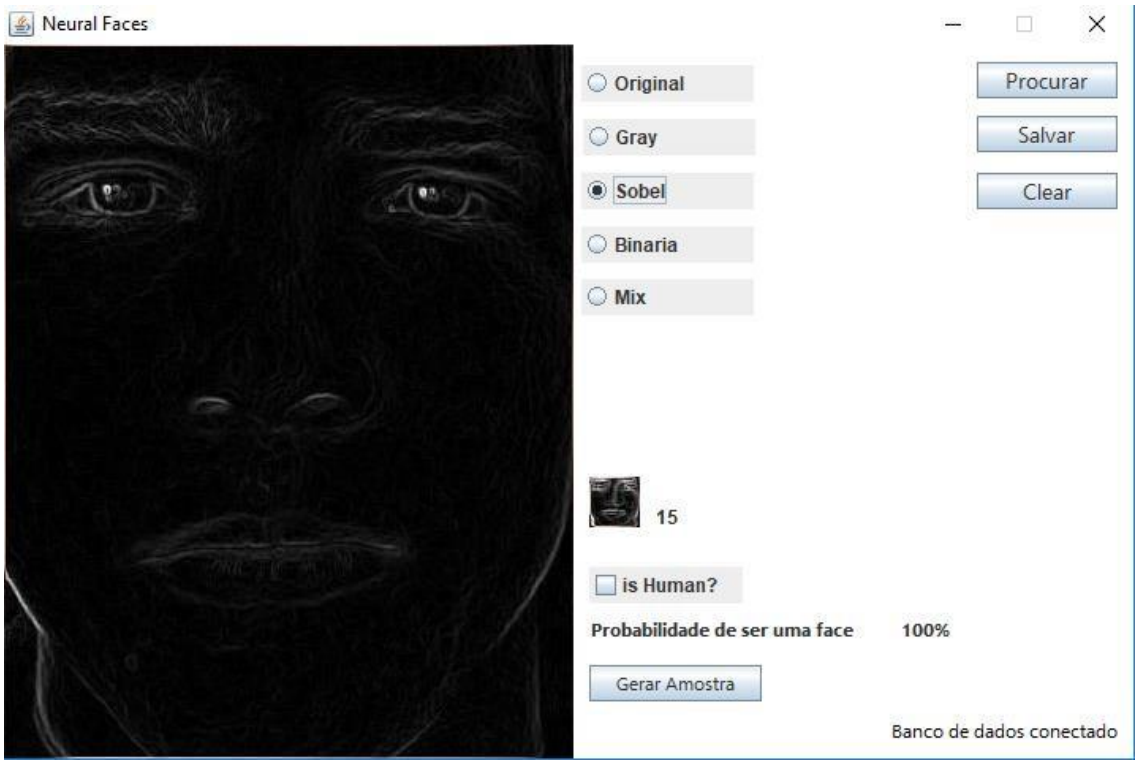

Figura 50. Imagem com filtro de Sobel.

Fonte: Felipe Andrade.

6.4.3 Limiarização (Binário)

Este algoritmo, busca a total transformação da imagem em preto e branco, ou seja 0 e 1. Tornando os valores dos pixels mais atrativos para as entradas da rede neural.

Esta técnica é conhecida no processamento de imagens como Limiarização. GONZALEZ e WOODS (2013) descrevem que a Limiarização pode ser vista como uma operação que envolve teste de uma função T da forma:

$$T = T[x, y, p(x, y), f(x, y)]$$

Sendo que f(x, y) é o nível de cinza do ponto e p(x, y) aponta alguma prioridade local desse ponto, como exemplo, o nível de cinza média de uma vizinhança centrada em (x, y). Uma imagem limiarizada g(x, y) é definida como:

$$g(x, y) = (1 \text{ se } f(x, y) > T) \text{ ou } (0 \text{ se } f(x, y) <= T)$$

A Figura 51 Imagem Binária, mostra a interface gráfica tratando a imagem que foi selecionada do sistema operacional e a transformando em uma matriz de 0 e 1.

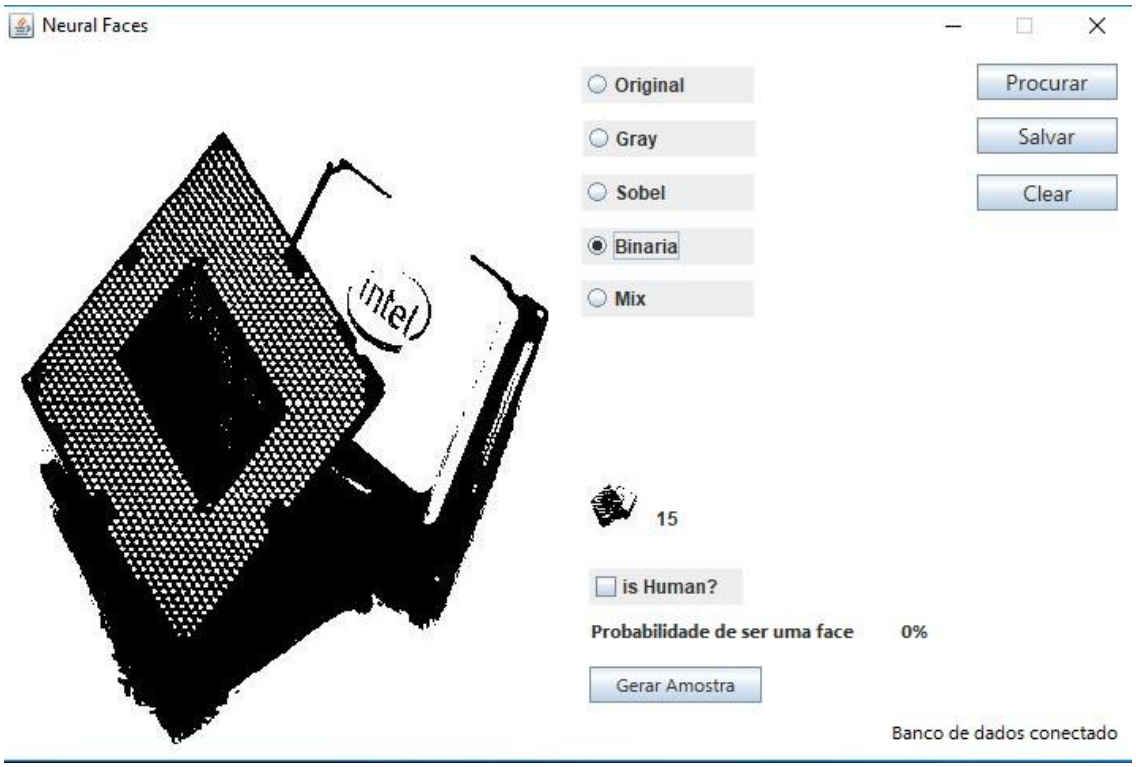

Figura 51. Imagem Binária.

Fonte: http://s2.glbimg.com/gpqskLwZ660ZwM1MsI5TR2qx0-c=/0x0:700x466/695x463/s.glbimg.com/po/tt2/f/original/2014/06/07/extreme_large1.jpg

O apêndice G, apresenta o código fonte com as funções que convertem imagens coloridas para petro e branco.

7. Reconhecimento de padrões em Imagens

Uma imagem pode possuir milhares de características diferentes não apenas pelo objeto existente nela como também as texturas, as cores, pelos, sombras, luminosidade, entre outras peculiaridades. Acarretando em um número considerável de variáveis para lidar com elas. Porém, um conjunto de imagens agrupadas sob um contexto específico pode apresentar padrões existentes em todas elas que podem servir para identificar novas imagens que não estavam presentes neste conjunto específico, mas que possuem características ou padrões que seja possível classifica-la neste mesmo grupo.

> O reconhecimento de padrões é formalmente definido como o processo pelo qual um padrão/sinal recebido é atribuído a uma classe dentre um número predeterminado de classes (Haykin, 2001, página 92).

Para alcançar o objetivo deste trabalho e passar a imagem como entrada para a rede neural foi necessário utilizar de técnicas de segmentação de imagens, tornando uma imagem colorida em preto e branco.

Primeiro foram coletadas 99 imagens de caracteres numéricos da internet com valores de 0 a 9, sendo 42 imagens coloridas. Após isso foi realizado o recorte das imagens igualando suas dimensões para que a altura e largura possuíssem tamanhos idênticos. Não foi necessário deixar todas as imagens com a mesma resolução pois dentro do sistema já existia um algoritmo que deixaria a imagem com o tamanho 32 por 32, ou seja, a altura e a largura terão 32 pixels de cada lado, porém foi necessário deixar a imagem quadrada para que ela não se deformasse quando fosse lida pelo algoritmo.

Feito o tratamento manual da imagem, é necessário fazer o tratamento computacional que ficou encarregado de diminuir os valores dos pixels para 0 e 1 tornando a imagem binária.

A figura 52. Tratamento da imagem, apresenta a transformação da imagem colorida numa matriz de 0’s e 1’s. Na imagem em questão os valores pretos que seriam 0 de acordo com o padrão RGB foram modificados para 1(branco) pois a parte preta é a parte interessante que irá servir como entrada para a rede neural, pois é nela onde está o caractere numérico.

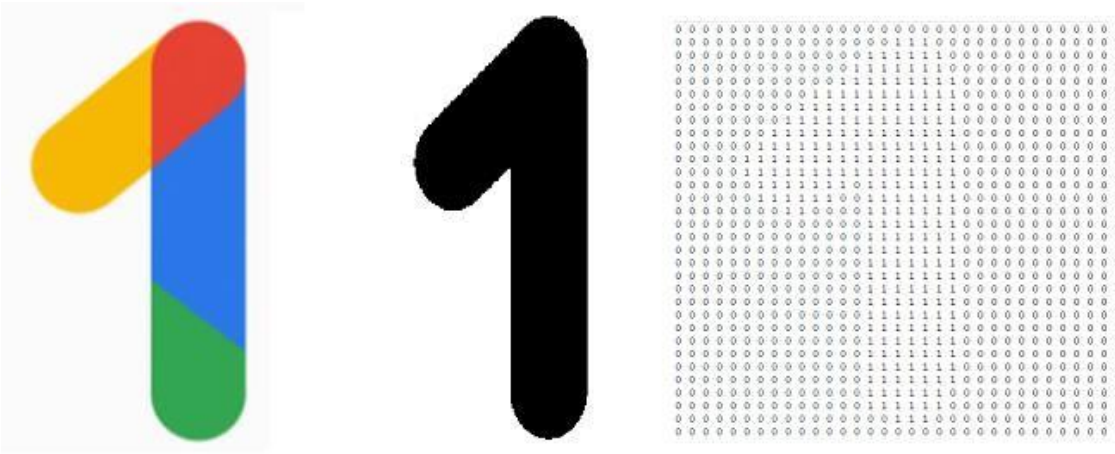

Figura 52. Tratamento de imagem.

Fonte: Próprio autor.

Doze imagens do caractere numérico 1 foram introduzidas como entradas para a rede neural que ficaria responsável por identificar se era ou não este valor. Também foram fornecidas mais 9 imagens cada uma representando uma valor diferente de 0 a 9.

A figura 53, apresenta as amostras fornecidas para que a rede neural conseguisse identificar o padrão deste número.

Figura 53. Amostras do caractere numérico 1.

Fonte: Próprio autor.

A figura 54, Amostras de caracteres numéricos diferentes, apresenta as outras amostras que serviram para que a rede separasse o numero 1 dos outros números que não eram ele.

Figura 54. Amostras de caracteres numéricos diferentes de 1.

Fonte: Próprio autor.

As amostras foram fornecidas para o sistema que ficaria encarregado de redimensionar a imagem numa matriz de 32 por 32 pixels com valores apenas de 0's e 1's. Sendo que o 1 fica nos pontos onde existe o caractere numérico.

A matriz foi colocada num vetor e fornecido como entrada para a rede neural.

Foi utilizado o mesmo algoritmo feito anteriormente nos outros testes para ver como a rede neural respondia.

Após o treinamento, os pesos e o *bias* foram salvos no banco de dados NoSQL MongoDB, para que, próximas imagens que não estavam presentes nas amostras do conjunto de treinamento pudessem ser classificadas. O código fonte para salvar os resultados e recuperá-los do banco para calcular novas entradas encontram-se nos apêndices M e N.

A seguir será o mostrado o método usado para reconhecer o padrão numérico na imagem.

7.1 Algoritmo para salvar as amostras

Início

Coleta das imagens.

Tratamento das imagens manualmente.

Processamento das imagens.

Segmentação das imagens.

Salvar imagens no banco de dados MySQL.

Ler matriz de 0 e 1.

Se imagem igual a 1

marque a opção para identifica-la como 1, ou seja: verdadeiro.

Senão

Desmarque a opção para identificar a imagem como 0: falso .

Salvar as matrizes binárias como amostras no MongoDB.

Fim.

7.2 Algoritmo de treinamento

Início

Ler os dados salvos como amostras.

Iniciar treinamento da rede neural a partir das amostras.

Salvar os dados da rede neural no MongoDB.

Fim.

7.3 Algoritmo de reconhecimento de padrões

Início

Ler os dados da rede neural treinada.

Ler a nova imagem.

Segmentação da imagem.

Classificação da imagem.

Fim

7.4 Resultados da Rede Neural

Concluído o treinamento, as outras imagens não utilizadas como amostras foram lidas pelo programa para ver se a rede neural conseguia separar os valores que não eram o caractere numérico 1.

De 78 das imagens restantes usadas, a rede neural errou apenas 6, sendo 4 imagens do número 4 e duas do número 7 que por sinal os dois números possuem certas características próximas ao número 1 dependendo da imagem.

O apêndice J, mostra o código fonte com as funções para acessar as imagens localizadas no sistema operacional.

A figura 55, mostra imagens com o caractere número 1 que não participaram do treinamento sendo reconhecidas pela rede neural.

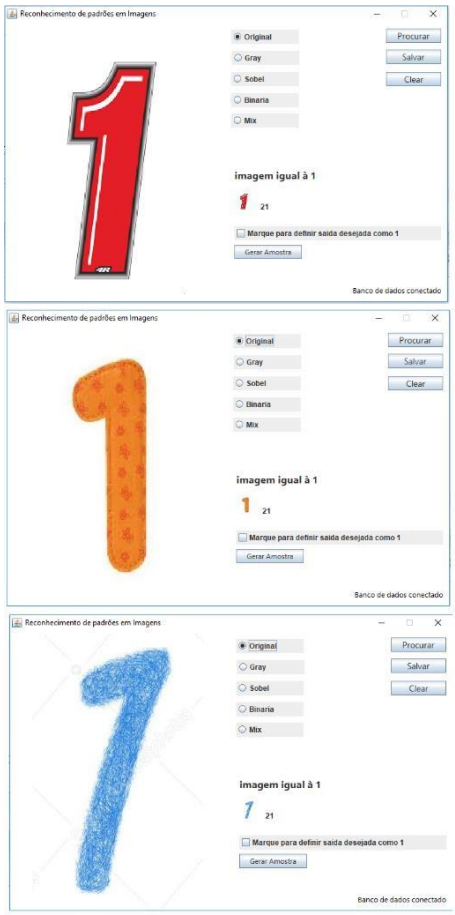

Figura 55. Reconhecendo caractere numérico 1.

Fonte: Próprio autor.

Na figura 56, mostra alguns dos números que não representam o caractere numérico 1.

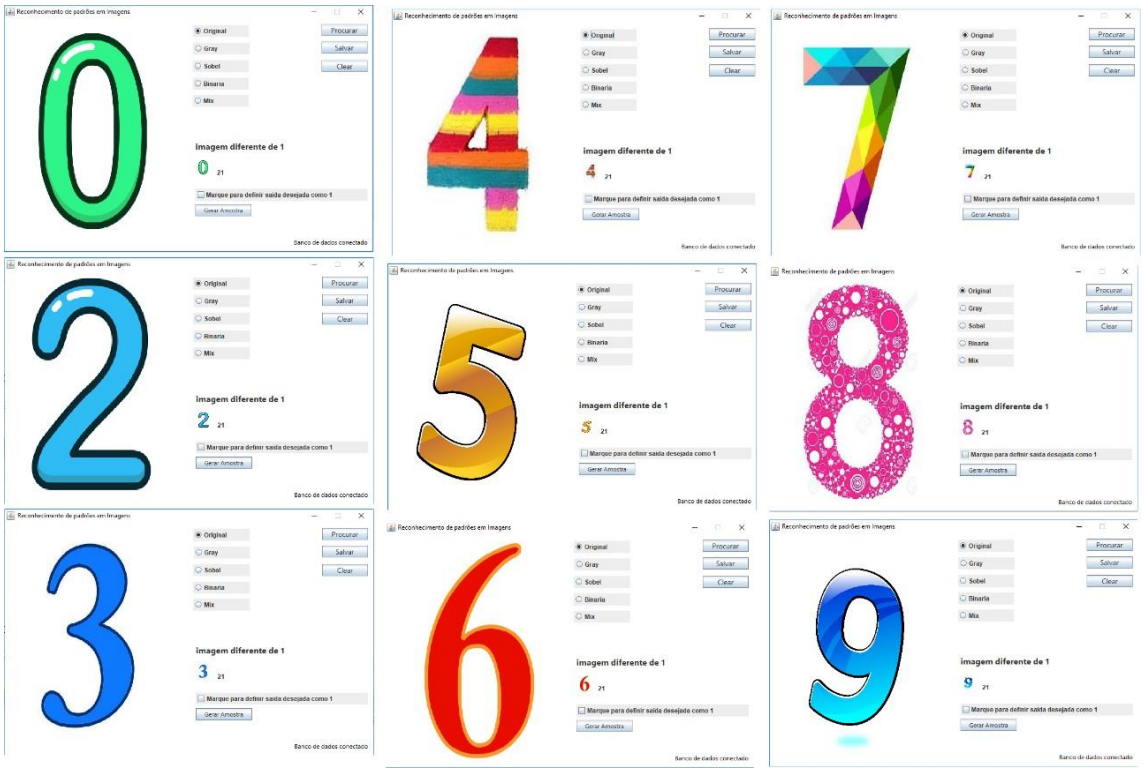

Figura 56. Caracteres numéricos diferentes de 1.

Fonte: Próprio autor.

A rede neural obteve 94% de precisão no reconhecimento de padrões do caractere 1, considerando o universo amostral apresentado a ela de 99 imagens.

A figura 57, mostra algumas falhas de reconhecimento de padrões.

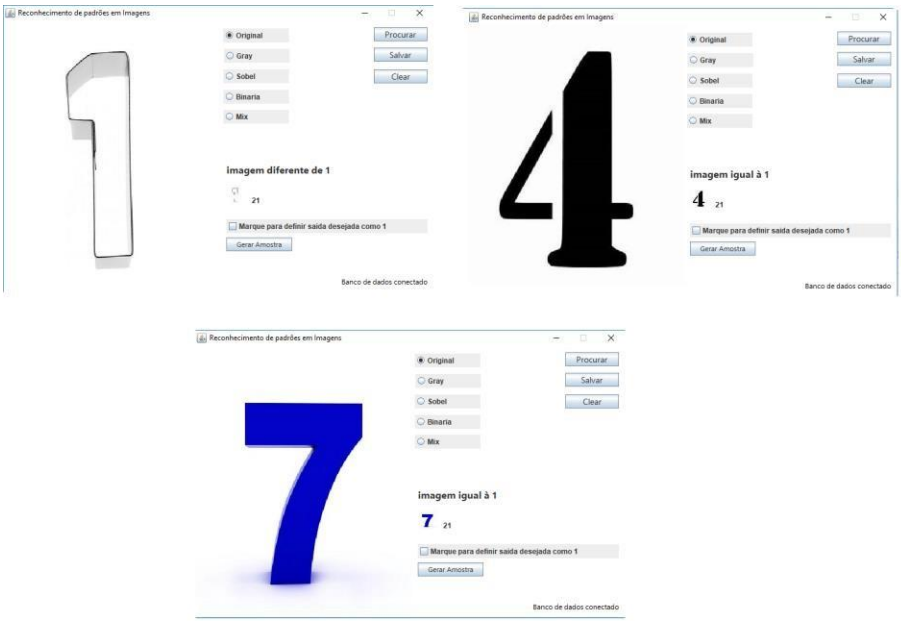

Figura 57. Falhas no reconhecimento de padrões.

Fonte: Próprio autor.

As falhas poderiam ser corrigidas passando-as como entradas para o treinamento da rede, e assim aumentar a precisão e diminuir os erros.

A figura 58, resultados da rede neural, apresenta os dados detalhados obtidos da rede neural artificial Perceptron no reconhecimento de padrões do caractere numérico 1.

Reconhecimento do caractere numérico 1

Número testado	Qtde.	Imagens acertadas	Porcetagem de acerto
0	6	6	100%
2	6	6	100%
3	9	9	100%
4	9	5	60%
5	9	9	100%
6	9	9	100%
7	12	10	80%
8	14	14	100%
9	13	13	100%

Total de Imagens	99
Amostras	21
Usadas para teste	78
Total Falhas	6
Total acerto	72

Figura 58. Resultados da Rede Neural.

Fonte: Próprio autor.

O caractere numérico 1 foi excluído da tabela pois os mesmo foram usados para o treinamento da rede.

Foram baixadas mais quatro imagens do numero 1 para serem testadas. A rede neural reconheceu 3 imagens corretamente e errou apenas uma. Os apêndices K e L, mostram o código fonte da rede neural esponsável por tratar os padrões da imagem e as amostras fornecidas como entradas.

Apesar de parecerem muito complexas e as vezes em algumas redes neurais é impossível saber como a rede consegue acertar o resultado correto de tão grande que é quantidade de camadas e neurônios entrelaçados. Ainda assim, uma ferramenta poderosa como esta, foi possível de ser implementada. Mesmo sendo a mais simples delas, deu para se ter uma ideia do que as redes neurais são capazes.

8. Sugestões

Com base no modelo apresentado sugere-se criar um neurônio para cada valor numérico. Em seguida, espera-se que o programa preveja o número apresentado e não apenas separar um valor dos demais.

Sugere-se também, aproveitar o conhecimento adquirido para estudar novos modelos de redes neurais que sejam não lineares, como Perceptron de Múltiplas Camadas e os mapas auto organizáveis de Teuvo Kohonen.

9. Conclusão

Para que uma rede neural artificial aprenda é necessário compreender o problema e pensar numa forma de como passa-lo para a RNA, ou seja, as entradas da rede.

É necessário que as entradas sejam normalizadas, pois computadores entendem apenas números, ou melhor, 0’s e 1’s.

Ao entender o problema será necessário verificar qual modelo de RNA será o melhor para apresentar os resultados desejados.

Para que a RNA enxergue uma imagem é necessário reduzir os valores RGB de cada pixel, pois são esses valores que serão passados como entrada, já que se fossem passados os valores reais, eles seriam grandes demais pois cada tonalidade RGB possuem valores de 0 a 255.

As imagens serão tratadas para reduzir os valores dos pixels, utilizando algoritmos de tratamento de imagens para deixar a imagem em tons de cinza, reduzindo drasticamente a oscilação de valores baixo ou altos demais, o algoritmo de Sobel para realçar as bordas da imagem e por fim o algoritmo para binarizar a imagem e transforma-la em 0 e 1.

Após a fase de testes é possível concluir que a rede neural é capaz de aprender de acordo com o número de amostras apresentado a ela. Quanto mais amostras, maior será sua precisão.

Existe a chance de a rede neural falhar pois ela não traz o resultado100% exato, o que ela faz é prever novas entradas com uma alta taxa de porcentagem de acerto de acordo com os dados apresentados a ela. Para que essa porcentagem aumente é necessário modificar a estrutura da rede para que ela melhore.

Referências

ABOELNASR, Mostafa. Threshold Dithering java Source Code. Limiarização em Java. Youtube. 1min54seg. 5 de janeiro de 2018. <https://www.youtube.com/watch?v=aoLcKUYVq-M> Acesso em: 31 de janeiro de 2018.

AGRELA, Lucas. Inteligência artificial da IBM já ajuda advogados brasileiros. 2017. Disponível em <https://exame.abril.com.br/tecnologia/inteligencia-artificial-da-ibm-ja-ajuda-advogados-brasileiros/> Acesso em: 26 de julho de 2018.

ANDERSON, Cristiano. MongoDB e NoSQL, tudo o que você precisa saber. Youtube. 15 de agostos de 2015. 53min44seg. Disponível em < https://www.youtube.com/watch?v=uQGEY0KnAjg> Acesso em: 27 de julho de 2018.

BRAGA, Antônio de Paduá, LUDERMIR, Teresa Bernarda, CARVALHO, André Carlos Ponce de Leon Ferreira. Redes Neurais Artificiais Teoria e aplicações. Livros Técnicos e Científicos Editora. 2000.

CABRAL, Eduardo L. L. PMR2560 – Visão Computacional Detecção de Bordas. 36 páginas. Apresentação de PowerPoint. 2018.
COUTO, Gabriel. YouTube. 9 de maio, 2017. 44min36seg. Disponível em: < https://www.youtube.com/watch?v=2sNBvlaImLc>. Acesso em 5 de dezembro, 2017.

CARVALHO, Vinicius. MySQL comece com o principal banco de dados opensource do Mercado. Ed. Casa do Código. São Paulo – SP. 2018.
CNBC. NASA and this Google employee are using AI to find new planets, with an eye towards finding alien life. Disponível em: <https://www.cnbc.com/2017/12/20/nasa-and-google-are-using-ai-to-find-new-planets.html>. Acesso em: 4 de março de 2018.

COLLINS. Humongous. Disponível em < https://www.collinsdictionary.com/pt/dictionary/english/humongous> Acesso em: 27 de julho de 2018.

D'EGMONT, Tahiana. IBM Watson. O que é Watson? Plataforma cognitiva? Inteligência artificial? Um robô? 16 de dezembro de 2016. Disponível em < https://www.ibm.com/blogs/digital-transformation/br-pt/o-que-e-watson-plataforma-cognitiva-inteligencia-artificial-robo/> Acesso em: 26 de julho de 2018.

DEITEL, Paul, Deitel, Harvey. Java como programar. 8. Ed. Tradução de Edson Furmankiewicz. São Paulo. 2010.

ENGINEERING, Uber. Criando transporte mais confiável com machine learning e IA na Uber. 30 de novembro de 2017. Disponível em: <https://imasters.com.br/desenvolvimento/criando-transporte-mais-confiavel-com-machine-learning-e-ia-na-uber> Acesso: 26 de julho de 2018.

FELITTI, Guilherme. Como a IBM pretende ganhar dinheiro com o supercomputador Watson. 10 de março de 2014. Disponível em <https://epocanegocios.globo.com/Tecnologia/noticia/2014/03/como-ibm-pretende-ganhar-dinheiro-com-o-supercomputador-watson.html> Acesso em: 26 de julho de 2018.

GAGO, Everton. YouTube. 8 ago, 2015. 1Hr10min41s. Disponível em: < https://www.youtube.com/watch?v=xZYTj6MXYLk> Acesso em: 30 dez, 2017.

GONZALEZ, Rafael C., WOODS, Richard E. Processamento Digital de Imagens. Ed. 3ª. Tradução de: Cristina Yamagami e Leonardo Piamonte. Editora Pearson. 2010.

GONZALEZ, Rafael C, WOODS, Richard E. Processamento de imagens Digitais. Tradução de Roberto Marcondes Cesar Junior e Luciano da Fontoura Costa. Editora Blucher. 2013.

HAYKIN, S. Redes Neurais: Princípios e prática. 2. Ed. Tradução de Paulo Martins Angel. Porto Alegre: Editora Bookman, 2001.

HOLMES, Thiago. Tensor Flow – Inteligência Artificial opensource desenvolvida pela Google. 24 de novembro de 2015. Disponível em <https://linuxcentro.com.br/news/tensor-flow-inteligencia-artificial-opensource-desenvolvida-pela-google/> Acesso em: 26 de julho de 2018.

INSTITUTE, Project Management. Um guia do conhecimento em gerenciamento de projetos, 5ed. Brasport, 2014.

JORDÃO, fabio. Pixel: O que você precisa saber sobre ele? Disponível em: https://www.tecmundo.com.br/pixel/7529-pixel-o-que-voce-precisa-saber-sobre-ele-.htm. Acesso em: 11 mar 2018.

PAULO, João. YouTube. 30 set, 2016. 30min20s. https://www.youtube.com/watch?v=wLqOGPnoqdk> Acesso em 20 jan, 2018.

PETERSEN, Michael Egmont. Reconhecimento de Imagens com Redes Neurais. Tradução de: Nicholas Braga. Disponível em: http://neuroph.sourceforge.net/image_recognition_portuguese.html>. Acesso em: 10 de março, 2018.

MICHAELIS. Dicionário Online. Disponível em <https://michaelis.uol.com.br/moderno-portugues/busca/portugues-brasileiro/padr%C3%A3o/> Acesso em: 25 de julho de 2018.

SILVA, Gabriel Souza da. Redes Neurais Artificiais para a identificação de objetos contidos em imagens. 2015. 84 páginas. Trabalho de Conclusão de Curso. Fundação Educacional do Munícipio de Assis, São Paulo, 2017.

Apêndice A - Modelo de rede neural Perceptron Simples.

```
package br.com.eric.rna;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

import org.bson.Document;

public class Perceptron {

	private String modelo;
	private String descricao;
	private String problema;
	private String nameFunction;

	private LocalDateTime inicio;
	private LocalDateTime termino;

	private double[] pesos;
	private double bias;

	private int NP; // numero de pesos

	private List<TrainingSet> conjuntoTreinamento;

	private int NTS; // numero de amostras de treinamento

	private int iteracoes;

	public Perceptron(List<TrainingSet> training_sets, int i) {

		System.out.println("Perceptron Simples");

		this.conjuntoTreinamento = training_sets;

		this.NTS = training_sets.size();
		this.NP = training_sets.get(0).getEntradas().length;

		this.pesos = new double[this.NP];

		iniciarPesos();

		this.iteracoes = i;

		this.nameFunction = "Sigmoid";

		System.out.println("Iniciando treinamento...");

	}

	private void iniciarPesos() {
		this.bias = Math.random();
		for(int i = 0; i < this.NP; i++) {
			this.pesos[i] = Math.random();
		}
	}

	public double executar(TrainingSet ts) {
		double soma = 0.0;
		for(int i = 0; i < this.NP; i++) {
```

```
                soma += ts.getEntradas()[i] * this.pesos[i];
            }
            soma += this.bias;
            return ActivationFunction.sigmoid(soma);
      }

      private void corrigirPesos(TrainingSet ts) {
            double y = executar(ts);
            for(int i = 0; i < this.NP; i++) {
                  this.pesos[i] += (ts.getSaida() - y) *
ts.getEntradas()[i];
            }
            this.bias += (ts.getSaida() - y);
      }

      public void treinar() {

            int epocas = 0;

            this.inicio = LocalDateTime.now();

            do {

                  for(int i = 0; i < this.NTS; i++) {

                        corrigirPesos(this.conjuntoTreinamento.get(i));

                        double saida =
executar(this.conjuntoTreinamento.get(i));
                        double alvo =
this.conjuntoTreinamento.get(i).getSaida();

                        System.out.printf("Saida: %.8f -> Alvo: %.1f
\n", saida, alvo);
                  }

                  epocas++;
            } while (epocas < this.iteracoes);


            this.termino = LocalDateTime.now();

            System.out.println("Treinamento completo!");
      }

      public String classificar(TrainingSet ts) {

            String y = "";

            double soma = 0.0;
            for(int i = 0; i < this.NP; i++) {

                  soma += (ts.getEntradas()[i] * this.pesos[i]);

            }
            soma += this.bias;

            y = (ActivationFunction.sigmoid(soma) < 0.5)? "FALSE" :
"TRUE";

            return y;
```

```
	}

	public Document ToDocument(String modelo, String desc, String
problema) {

		this.modelo = modelo;
		this.descricao = desc;
		this.problema = problema;

		Document doc = new Document("modelo", this.modelo);
		doc.append("descricao", this.descricao);
		doc.append("problema", this.problema);
		doc.append("bias", this.bias);

		List<Document> ts = new ArrayList<>();

		for(int i = 0; i < this.NTS; i++) {

			List<Double> e = new ArrayList<>();

			for(int j = 0; j < this.NP; j++) {


	e.add(this.conjuntoTreinamento.get(i).getEntradas()[j]);

			}

			Document d = new Document("entradas", e)
					.append("saida",
this.conjuntoTreinamento.get(i).
							getSaida());

			ts.add(d);
		}

		doc.append("training_sets", ts);

		List<Double> p = new ArrayList<>();

		for(int i = 0; i < this.NP; i++) {
			p.add(this.pesos[i]);
		}

		doc.append("pesos", p);

		doc.append("funcao", this.nameFunction);

		doc.append("inicio", "" + this.inicio);

		doc.append("iteracoes", this.iteracoes);

		doc.append("termino", "" + this.termino);

		return doc;
	}

	public String getModelo() {
		return modelo;
	}

	public void setModelo(String modelo) {
		this.modelo = modelo;
```

```
}

public String getDescricao() {
	return descricao;
}

public void setDescricao(String descricao) {
	this.descricao = descricao;
}

public String getProblema() {
	return problema;
}

public void setProblema(String problema) {
	this.problema = problema;
}

public void setNameFunction(String nameFunction) {
	this.nameFunction = nameFunction;
}

public String getNameFunction() {
	return this.nameFunction;
}

public LocalDateTime getInicio() {
	return inicio;
}

public void setInicio(LocalDateTime inicio) {
	this.inicio = inicio;
}

public LocalDateTime getTermino() {
	return termino;
}

public void setTermino(LocalDateTime termino) {
	this.termino = termino;
}

public double[] getPesos() {
	return pesos;
}

public void setPesos(double[] pesos) {
	this.pesos = pesos;
}

public double getBias() {
	return bias;
}

public void setBias(double bias) {
	this.bias = bias;
}

public int getNP() {
	return NP;
}
```

```
	public void setNP(int nP) {
		NP = nP;
	}

	public List<TrainingSet> getConjuntoTreinamento() {
		return conjuntoTreinamento;
	}

	public void setConjuntoTreinamento(List<TrainingSet>
conjuntoTreinamento) {
		this.conjuntoTreinamento = conjuntoTreinamento;
	}

	public int getNTS() {
		return NTS;
	}

	public void setNTS(int nTS) {
		NTS = nTS;
	}

	public int getIteracoes() {
		return iteracoes;
	}

	public void setIteracoes(int iteracoes) {
		this.iteracoes = iteracoes;
	}


}
```

Apêndice B - Classe que guarda as funções de ativação.

```
public class ActivationFunction {

	public static double sigmoid( double x ) {
		return 1 / (1 + Math.exp(-x));
	}

	public static double linear( double x ) {
		return x;
	}

	public static double heaviside ( double x ) {

		if( x >= 0) {
			return 1;
		}

		return 0;
	}

	public static double tangenteHiperbolica( double x ) {
		return Math.tanh(x);
	}


}
```

Apêndice C - Classe para tratar do conjunto de treinamento (Amostras de entrada).

```
import java.util.List;

public class TrainingSet {

	private double[] entradas;
	private double saida;

	public TrainingSet(double[] entradas, double saida) {
		this.entradas = entradas;
		this.saida = saida;
	}

	public double[] getEntradas() {
		return entradas;
	}
	public void setEntradas(double[] entradas) {
		this.entradas = entradas;
	}
	public double getSaida() {
		return saida;
	}
	public void setSaida(double saida) {
		this.saida = saida;
	}

	public static void show(List<TrainingSet> training_sets) {

		System.out.println("Conjunto de Treinamento: \n");

		for(int i = 0; i < training_sets.size(); i++) {

			final int T =
training_sets.get(i).getEntradas().length;

			System.out.print("Entradas: ");

			for(int j = 0; j < T; j++) {

				System.out.print(training_sets.get(i).
					getEntradas()[j] + ", ");

			}

			System.out.println(" Saida: " +
training_sets.get(i).getSaida());
		}

		System.out.println("\n");
	}

}
```

Apêndice D - Executando o Perceptron para o treinamento do “OU” lógico.

```
import java.util.ArrayList;
import java.util.List;

import br.com.eric.mongodb.MongoDB;
import br.com.eric.rna.Perceptron;
import br.com.eric.rna.TrainingSet;

public class OrLogico {


	public static void main(String[] args) {

		List<TrainingSet> training_sets = new ArrayList<>();

		training_sets.add(new TrainingSet(new double[]{ 0, 0}, 0));
		training_sets.add(new TrainingSet(new double[]{ 0, 1}, 1));
		training_sets.add(new TrainingSet(new double[]{ 1, 0}, 1));
		training_sets.add(new TrainingSet(new double[]{ 1, 1}, 1));

		TrainingSet.show(training_sets);

		Perceptron net = new Perceptron(training_sets, 1000);
		net.treinar();

		System.out.println("Classificando...");
		for(int i = 0; i < training_sets.size(); i++) {

	System.out.println(net.classificar(training_sets.get(i)));
		}

		net.setModelo("Perceptron Simples");
		net.setDescricao("Resolve problemas lineares");
		net.setProblema("OR Logico");
		System.out.println(net.ToDocument(net.getModelo(),
					net.getDescricao(),
					net.getProblema()).toJson());

		MongoDB.insert(net.ToDocument(net.getModelo(),
				net.getDescricao(),
				net.getProblema()), "rna");
	}
}
```

Apêndice E - Classe de execução de testes com os valores do Plano Cartesiano.

```
public class PlanoCartesiano {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		List<TrainingSet> training_sets = new ArrayList<>();

		// Classe A
		training_sets.add(new TrainingSet(new double[]{ 0.08, 0.6},
1));
		training_sets.add(new TrainingSet(new double[]{ 0.12, 0.9},
1));
		training_sets.add(new TrainingSet(new double[]{ 0.2, 0.7},
1));
		training_sets.add(new TrainingSet(new double[]{ 0.4, 0.9},
1));

		// Classe B
		training_sets.add(new TrainingSet(new double[]{ 0.4, 0.2},
0));
		training_sets.add(new TrainingSet(new double[]{ 0.5, 0.5},
0));

		TrainingSet.show(training_sets);

		Perceptron net = new Perceptron(training_sets, 1000);
		net.treinar();

		System.out.println("Classificando...");

		TrainingSet t1 = new TrainingSet(new double[] {0.9, 0.9},
1);
		TrainingSet t2 = new TrainingSet(new double[] {0.3, 0.5},
0);

		String A = (net.executar(t1) < 0.5)? "B" : "A";
		//System.out.println(net.executar(t1));
		String B = (net.executar(t2) < 0.5)? "B" : "A";

		System.out.println("Classe: " + A);
		System.out.println("Classe: " + B);

		net.setModelo("Perceptron Simples");
		net.setDescricao("Resolve problemas lineares");
		net.setProblema("Plano Cartesiano - Classes A e B");
		System.out.println(net.ToDocument(net.getModelo(),
				net.getDescricao(),
				net.getProblema()).toJson());

		MongoDB.insert(net.ToDocument(net.getModelo(),
				net.getDescricao(),
				net.getProblema()), "rna");

	}


}
```

Apêndice F - Classe para persistir os dados no banco de dados MongoDB.

```
/*
 * Site do MongoDB:
 *
 *    https://www.mongodb.com/
 *
 * Pós Instalação:
 *
 *  1° - Criar um diretorio no C: chamado "data"
 *  2° - Acessar o diretorio de instalação do MongoDb
 *  3° - Executar o "mongod.exe" que executa o servidor
 *  4° - Executar o "mongo.exe" que executa o cliente
 *
 * Download do mongo java driver:
 *
 *    https://mongodb.github.io/mongo-java-driver/
 *
 * Driver adicionado ao diretorio lib
 *
 * Apontar o build path para o driver
 *
 * Let's Code...
 *
 * */

import java.util.ArrayList;
import java.util.List;

import org.bson.Document;

import com.mongodb.MongoClient;
import com.mongodb.MongoException;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;

import br.com.eric.recogimages.Conjunto;

public class MongoDB {

	private static MongoClient mc;
	public static final String DBNAME = "redeneural";

	public static MongoDatabase getConexao() {

		mc = new MongoClient("localhost", 27017);

		MongoDatabase db = null;

		try {

			db = mc.getDatabase(DBNAME);

		} catch (MongoException ex) {

			System.out.println("Erro: " + ex.getMessage());

		}

		return db;
	}
```

```
	public static boolean isConectado() {

		boolean conectado = false;

		mc = new MongoClient("localhost", 27017);

		MongoDatabase db = null;

		try {

			db = mc.getDatabase(DBNAME);

			System.out.println("MongoDB conectado! " +
db.getName());

			MongoCollection<Document> colecao =
db.getCollection("amostras");

			System.out.println("Coleção " +
colecao.getNamespace());

			conectado = true;

		} catch (MongoException ex) {

			System.out.println("Erro: " + ex.getMessage());

		} finally {
			mc.close();
		}

		return conectado;
	}

	public static boolean insert(Document doc, String col) {

		MongoDatabase db = getConexao();

		if(isConectado()) {

			try {

				MongoCollection<Document> colecao =
db.getCollection(col);

				colecao.insertOne(doc);

				System.out.println("Novo documento
adicionado!");

				return true;

			} catch (MongoException ex) {

				System.out.println("Erro: " + ex.getMessage());

			}
		} else {
			System.out.println("MongoDB não está conectado...");
		}
```

```
			return false;
		}

		@SuppressWarnings("unchecked")
		public static boolean isAmostraDuplicada(List<Integer> amostraBin)
{

			MongoDatabase db = getConexao();

			MongoCollection<Document> colecao =
db.getCollection("amostras");

			Document busca = new Document("amostra", amostraBin);

			MongoCursor<Document> iterador =
colecao.find(busca).iterator();

			try {

				List<Integer> amostraEncontrada = new ArrayList<>();

				while(iterador.hasNext()) {

					Document doc = iterador.next();

					amostraEncontrada = doc.get("amostra",
List.class);

					if(amostraBin.equals(amostraEncontrada)) {
						return true;
					}
				}

			} catch (MongoException e) {
				e.printStackTrace();
			}

			return false;
		}

		@SuppressWarnings("unchecked")
		public static List<Conjunto> getConjunto() {

			List<Conjunto> conjunto = new ArrayList<>();

			MongoDatabase db = getConexao();

			MongoCollection<Document> colecao =
db.getCollection("amostras");

			MongoCursor<Document> iterador = colecao.find().iterator();

			try {

				// verifica se a coleção nao esta vazia
				if (iterador.hasNext()) {

					while(iterador.hasNext()) {

						Document doc = iterador.next();
						Conjunto amostra = new Conjunto();
```

```
                                amostra.setEntradas(doc.get("amostra",
List.class));

      amostra.setSaida(doc.getInteger("saida_desejada"));

                                conjunto.add(amostra);
                          }

                    } else {

                          conjunto = null;

                    }

              } catch (MongoException ex) {

                    System.out.println("Erro: " + ex.getMessage());
                    ex.printStackTrace();

              } finally {
                    iterador.close();
              }

              return conjunto;
      }
}
```

Apêndice G - Classe para tratar as imagens convertando para binário.

```
import static java.awt.image.BufferedImage.TYPE_INT_RGB;

import java.awt.Color;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;

import javax.imageio.ImageIO;

import br.com.eric.model.Imagem;

/*
 *  Algoritmo de Limiarização
 *
 *  Autor Original: https://www.youtube.com/watch?v=aoLcKUYVq-M
 *
 * */

public class Binario {

	private Long id;
	private Imagem img;
	private byte[] imgByteBin;

	public Binario () {

	}

	public Binario (Imagem img) {
		this.img = img;
		this.imgByteBin = getBinario(this.img);
	}

	public byte[] getBinario(Imagem img) {

		byte[] imBin = null;

		BufferedImage imagem = null;
		ByteArrayOutputStream baos = new ByteArrayOutputStream();

		Color cor = null;

		try {

			InputStream in = new
ByteArrayInputStream(img.getImagemBytes());

			imagem = ImageIO.read(in);

		BufferedImage newImage = new
BufferedImage(imagem.getWidth(), imagem.getHeight(), TYPE_INT_RGB);

		//System.out.printf("Image => Largura: %d, Altura: %d \n",
buff.getWidth(), buff.getHeight());
```

```
            double media = 0;

            for(int row = 0; row < imagem.getWidth(); row++) {
                for(int col = 0; col < imagem.getHeight(); col++) {

                    cor = new Color(imagem.getRGB(row, col));

                    int r = cor.getRed();
                    int g = cor.getGreen();
                    int b = cor.getBlue();

                    media += (r * 0.21f + g * 0.71f + b * 0.07f) / 255;

                }
            }

            media /= (imagem.getHeight() * imagem.getWidth());

            for(int row = 0; row < imagem.getWidth(); row++) {
                for(int col = 0; col < imagem.getHeight(); col++) {

                    cor = new Color(imagem.getRGB(row, col));

                    int r = cor.getRed();
                    int g = cor.getGreen();
                    int b = cor.getBlue();

                    double lum = (r * 0.21f + g * 0.71f + b * 0.07f) /
255;

                    if (lum <= media) {

                        newImage.setRGB(row, col, 0x000000);

                    } else {

                        newImage.setRGB(row, col, 0xFFFFFF);

                    }
                }
            }

              ImageIO.write(newImage, img.getTipo().getTipo(), baos);

              imBin = baos.toByteArray();

              baos.close();

            } catch (IOException ex) {
                ex.printStackTrace();
            }

            return imBin;

        }

        public static byte[] getBinario(byte[] imgBytes, String tipo) {

            byte[] imBin = null;

            BufferedImage imagem = null;
```

```
            ByteArrayOutputStream baos = new ByteArrayOutputStream();

            Color cor = null;

            try {

                  InputStream in = new ByteArrayInputStream(imgBytes);

                  imagem = ImageIO.read(in);

            BufferedImage newImage = new
BufferedImage(imagem.getWidth(), imagem.getHeight(), TYPE_INT_RGB);

            //System.out.printf("Image => Largura: %d, Altura: %d \n",
buff.getWidth(), buff.getHeight());

            double media = 0;

            for(int row = 0; row < imagem.getWidth(); row++) {
                for(int col = 0; col < imagem.getHeight(); col++) {

                    cor = new Color(imagem.getRGB(row, col));

                    int r = cor.getRed();
                    int g = cor.getGreen();
                    int b = cor.getBlue();

                    media += (r * 0.21f + g * 0.71f + b * 0.07f) / 255;

                }
            }

            media /= (imagem.getHeight() * imagem.getWidth());

            for(int row = 0; row < imagem.getWidth(); row++) {
                for(int col = 0; col < imagem.getHeight(); col++) {

                    cor = new Color(imagem.getRGB(row, col));

                    int r = cor.getRed();
                    int g = cor.getGreen();
                    int b = cor.getBlue();

                    double lum = (r * 0.21f + g * 0.71f + b * 0.07f) /
255;

                    if (lum <= media) {

                        newImage.setRGB(row, col, 0x000000);

                    } else {

                        newImage.setRGB(row, col, 0xFFFFFF);

                    }
                }
            }

              ImageIO.write(newImage, tipo, baos);

              imBin = baos.toByteArray();
```

```
            baos.close();

        } catch (IOException ex) {
            ex.printStackTrace();
        }

        return imBin;

    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Imagem getImg() {
        return img;
    }

    public void setImg(Imagem img) {
        this.img = img;
    }

    public byte[] getImgByteBin() {
        return imgByteBin;
    }

    public void setImgByteBin(byte[] imgByteBin) {
        this.imgByteBin = imgByteBin;
    }

    public static void gerarAmostraFile(byte[] img, String nome) {

        BufferedImage imagem = null;
        //ByteArrayOutputStream baos = new ByteArrayOutputStream();

        Color cor = null;

        try {

            InputStream in = new ByteArrayInputStream(img);

            imagem = ImageIO.read(in);

        //BufferedImage newImage = new
BufferedImage(imagem.getWidth(), imagem.getHeight(), TYPE_INT_RGB);

        //System.out.printf("Image => Largura: %d, Altura: %d \n",
buff.getWidth(), buff.getHeight());

        double media = 0;

        for(int row = 0; row < imagem.getWidth(); row++) {
            for(int col = 0; col < imagem.getHeight(); col++) {

                cor = new Color(imagem.getRGB(row, col));

                int r = cor.getRed();
                int g = cor.getGreen();
```

```
                int b = cor.getBlue();

                media += (r * 0.21f + g * 0.71f + b * 0.07f) / 255;

            }
        }

        media /= (imagem.getHeight() * imagem.getWidth());


        String caminho = "C:\\Amostras\\" + nome + ".txt";

        PrintWriter amostraFile = new PrintWriter(caminho);

        for(int row = 0; row < imagem.getWidth(); row++) {
            for(int col = 0; col < imagem.getHeight(); col++) {

                cor = new Color(imagem.getRGB(row, col));

                int r = cor.getRed();
                int g = cor.getGreen();
                int b = cor.getBlue();

                double lum = (r * 0.21f + g * 0.71f + b * 0.07f) /
255;

                if (lum <= media) {

                    //newImage.setRGB(row, col, 0x000000);
                    amostraFile.print(1 + " ");

                } else {

                    //newImage.setRGB(row, col, 0xFFFFFF);
                    amostraFile.print(0 + " ");

                }
            }
            amostraFile.println("");
        }

          //ImageIO.write(newImage, img.getTipo().getTipo(), baos);

          //imBin = baos.toByteArray();

          //baos.close();

        amostraFile.close();
        System.out.println("Amostra gerada com sucesso!");

        } catch (IOException ex) {
              ex.printStackTrace();
        }
    }

    public static List<Integer> gerarAmostraArray(byte[] img, String
nome) {

        BufferedImage imagem = null;

        Color cor = null;
```

```
            List<Integer> arList = new ArrayList<>();


            try {

                  InputStream in = new ByteArrayInputStream(img);

                  imagem = ImageIO.read(in);

            //BufferedImage newImage = new
BufferedImage(imagem.getWidth(), imagem.getHeight(), TYPE_INT_RGB);

            //System.out.printf("Image => Largura: %d, Altura: %d \n",
buff.getWidth(), buff.getHeight());

            double media = 0;

            for(int row = 0; row < imagem.getWidth(); row++) {
                for(int col = 0; col < imagem.getHeight(); col++) {

                    cor = new Color(imagem.getRGB(row, col));

                    int r = cor.getRed();
                    int g = cor.getGreen();
                    int b = cor.getBlue();

                    media += (r * 0.21f + g * 0.71f + b * 0.07f) / 255;

                }
            }

            media /= (imagem.getHeight() * imagem.getWidth());


            for(int row = 0; row < imagem.getWidth(); row++) {
                for(int col = 0; col < imagem.getHeight(); col++) {

                    cor = new Color(imagem.getRGB(row, col));

                    int r = cor.getRed();
                    int g = cor.getGreen();
                    int b = cor.getBlue();

                    double lum = (r * 0.21f + g * 0.71f + b * 0.07f) /
255;

                    if (lum <= media) {

                        //newImage.setRGB(row, col, 0x000000);
                        arList.add(0);

                    } else {

                        //newImage.setRGB(row, col, 0xFFFFFF);
                        arList.add(1);

                    }
                }

            }

           System.out.println("Array de amostra gerada com sucesso!");
```

```
            } catch (IOException ex) {
                  ex.printStackTrace();
            }

            return arList;
      }
}
```

Apêndice H - Classe para manipular os objetos de Imagens.

```
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;

import javax.imageio.ImageIO;
import javax.swing.JOptionPane;

public class Imagem {

	private Long id;
	private String nome;
	private int largura;
	private int altura;
	private int tamanhoArquivo;
	private FileInputStream imagem;
	private byte[] imagemBytes;
	private String dataRegistro;
	private Tipo tipo;

	public Imagem() {
		//this.imagemBytes = new byte[1024];
	}

	public Imagem(File imagem, String nome, Tipo tipo) {

		this.nome = nome;
		this.tipo = tipo;
		this.tamanhoArquivo = (int)imagem.length();

		try {

			if(this.isImagem(imagem)) {
				this.imagem = new FileInputStream(imagem);

			}

		} catch(Exception ex) {
			ex.printStackTrace();
		}
	}

	public boolean isImagem(File arquivo) {
		try {

			BufferedImage imagem = ImageIO.read(arquivo);

			this.largura = imagem.getWidth();
			this.altura = imagem.getHeight();

			return true;

		} catch (IOException ex) {
			System.err.println("Falha ao ler o arquivo.\nErro: " +
					ex.getMessage());
		}
```

```
        return false;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public int getLargura() {
        return largura;
    }

    public void setLargura(int largura) {
        this.largura = largura;
    }

    public int getAltura() {
        return altura;
    }

    public void setAltura(int altura) {
        this.altura = altura;
    }

    public int getTamanhoArquivo() {
        return tamanhoArquivo;
    }

    public void setTamanhoArquivo(int tamanhoArquivo) {
        this.tamanhoArquivo = tamanhoArquivo;
    }

    public FileInputStream getImagem() {
        return this.imagem;
    }

    public void setImagem(FileInputStream imagem) {
        this.imagem = imagem;
    }

    public byte[] getImagemBytes() {
        return imagemBytes;
    }

    public void setImagemBytes(byte[] imagemBytes) {
        this.imagemBytes = imagemBytes;
    }

    public String getDataRegistro() {
        return dataRegistro;
    }
```

```
	public void setDataRegistro(String dataRegistro) {
		this.dataRegistro = dataRegistro;
	}

	public Tipo getTipo() {
		return tipo;
	}

	public void setTipo(Tipo tipo) {
		this.tipo = tipo;
	}

	public void mostre(Imagem img) {
		JOptionPane.showMessageDialog(null, "Imagem Status\n" +
				"\nId: " + img.getId() +
				"\nNome: " + img.getNome() +
				"\nLargura: " + img.getLargura() +
				"\nAltura: " + img.getAltura() +
				"\nData Registro: " + img.getDataRegistro() +
				"\nTipo: " + img.getTipo().getCodigo()
				);
	}

	@Override
	public String toString() {

		return "{\n" +
				 "\t\"nome\"    :\t\"" + this.getNome() + "\",\n" +
				 "\t\"altura\"  :\t" + this.getAltura() + ",\n" +
				 "\t\"largura\" :\t" + this.getLargura() + ",\n" +
				 "\t\"tipo\"    :\t\"" + this.getTipo().getTipo() +
"\"\n" +
				 "}\n";

	}

	public static byte[] redimencionar(File imgFile, final int L,
final int A, String extensao) {

		byte[] imgBin = null;

		try {

			BufferedImage imagemOriginal = ImageIO.read(imgFile);
			BufferedImage novaImagem = new BufferedImage(L, A,
imagemOriginal.getType());

			Graphics2D g2d = novaImagem.createGraphics();

			g2d.drawImage(imagemOriginal, 0, 0, L, A, null);

			g2d.dispose();

			//JOptionPane.showMessageDialog(null, "Deu certo...");
			ByteArrayOutputStream baos = new
ByteArrayOutputStream();

			ImageIO.write(novaImagem, extensao, baos);
			baos.flush();

			imgBin = baos.toByteArray();
```

```
                baos.close();

            } catch (IOException ex) {
                ex.printStackTrace();
            }
            return imgBin;
        }

        public static byte[] redimencionar(byte[] img, final int L, final
int A, String extensao) {

            byte[] imgByte = null;

            try {

                InputStream in = new ByteArrayInputStream(img);

                BufferedImage imagemOriginal = ImageIO.read(in);
                BufferedImage novaImagem = new BufferedImage(L, A,
imagemOriginal.getType());

                Graphics2D g2d = novaImagem.createGraphics();

                g2d.drawImage(imagemOriginal, 0, 0, L, A, null);

                g2d.dispose();

                //JOptionPane.showMessageDialog(null, "Deu certo...");
                ByteArrayOutputStream baos = new
ByteArrayOutputStream();

                ImageIO.write(novaImagem, extensao, baos);
                baos.flush();

                imgByte = baos.toByteArray();

                baos.close();

            } catch (IOException ex) {
                ex.printStackTrace();
            }
            return imgByte;
        }

        /*
         * Rotação de Imagem
         *
         * Autor Original: Zoran Davidović
         *
         * Video: https://www.youtube.com/watch?v=RLHG1dR3TsI
         *
         * */

        public static final int ROTATE_LEFT = 1;
        public static final int ROTATE_RIGHT = -1;

        public static byte[] rotate90(byte[] imgBin, final int LADO,
String tipo) {

            byte[] imgRot = null;
```

```
            BufferedImage imagem = null;

            ByteArrayOutputStream baos = new ByteArrayOutputStream();

            try {

                  InputStream in = new ByteArrayInputStream(imgBin);

                  imagem = ImageIO.read(in);

                  int L = imagem.getWidth();
                  int A = imagem.getHeight();

                  BufferedImage rotated = new BufferedImage(A, L,
imagem.getType());

                  for(int y = 0; y < A; y++) {

                        for(int x = 0; x < L; x++) {

                              switch(LADO) {

                              case ROTATE_LEFT :
                                    rotated.setRGB(y, (L - 1) - x,
imagem.getRGB(x, y));
                                    break;
                              case ROTATE_RIGHT :
                                    rotated.setRGB((A - 1) - y, x,
imagem.getRGB(x, y));

                              } // fim switch

                        }

                  }

              ImageIO.write(rotated, tipo, baos);

              imgRot = baos.toByteArray();

              baos.close();

            } catch (IOException ex) {

                  ex.printStackTrace();

            }

            return imgRot;
      }


      /*
       * Flipping de Imagem
       *
       * Autor Original: Zoran Davidović
       *
       * Video: https://www.youtube.com/watch?v=RLHG1dR3TsI
       *
       * */

      public static final int FLIP_VERTICAL = 1;
```

```
	public static final int FLIP_HORIZONTAL = -1;

	public static byte[] flip (byte[] imgBin, final int LADO, String
tipo) {

		byte[] imgByte = null;

		ByteArrayOutputStream baos = new ByteArrayOutputStream();

		try {

			InputStream in = new ByteArrayInputStream(imgBin);

			BufferedImage imagem = ImageIO.read(in);

			int L = imagem.getWidth();
			int A = imagem.getHeight();

			BufferedImage flipped = new BufferedImage(L, A,
BufferedImage.TYPE_INT_RGB);

			for(int y = 0; y < A; y++) {

				for(int x = 0; x < L; x++) {

					switch(LADO) {

					case FLIP_HORIZONTAL:
						flipped.setRGB((L - 1) - x, y,
imagem.getRGB(x, y));
						break;
					case FLIP_VERTICAL:
						flipped.setRGB(x, (A - 1) - y,
imagem.getRGB(x, y));
						break;

					} // fim switch


				}
			}

			ImageIO.write(flipped, tipo, baos);

			imgByte = baos.toByteArray();

			baos.close();

		} catch (IOException ex) {
			ex.printStackTrace();
		}

		return imgByte;


}
```

Apêndice I - Classe para tratar as extensões da imagem (Ex: jpg, png).

```
public class Tipo {

	private Byte codigo;
	private String tipo;

	public Byte getCodigo() {
		return codigo;
	}
	public void setCodigo(byte i) {
		this.codigo = i;
	}
	public String getTipo() {
		return tipo;
	}
	public void setTipo(String tipo) {
		this.tipo = tipo;
	}


}
```

Apêndice J - Classe para acessar o sistema de arquivos.

```
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileNameExtensionFilter;

public class AcessarArquivos {

	// Classe para procurar arquivos no Sistema
	private final JFileChooser procurarArquivo;

	// Classe que filtra o tipo de arquivo que será procurado
	private FileNameExtensionFilter filtrarArquivos;

	public String[] filtros;

	public String path;

	public AcessarArquivos() {

		this.procurarArquivo = new JFileChooser();

		this.filtros = new String[3];
		this.filtros[0] = "png";
		this.filtros[1] = "jpg";
		this.filtros[2] = "jpeg";

		this.path = "";

		this.filtrarArquivos = new
FileNameExtensionFilter("Imagens", "png", "jpg", "jpeg");

		// Só ira permitir a selecão dos arquivos especificados
		this.procurarArquivo.setFileFilter(filtrarArquivos);

	}

	public AcessarArquivos(String titulo, String[] filtros) {

		this.procurarArquivo = new JFileChooser();

		this.filtros = filtros;

		this.filtrarArquivos = new FileNameExtensionFilter( titulo ,
this.getFiltros() );

		this.procurarArquivo.setFileFilter(filtrarArquivos);
	}

	// Abre uma interface pré programada para acessar o sistema de
arquivos
	public void abrirInterface(JButton botao) {

		try {

			this.getProcurarArquivo().showOpenDialog(botao);


	this.setPath(this.getProcurarArquivo().getSelectedFile().getAbsolu
tePath());

			this.setPath(tratarCaminho(this.getPath()));
```

```
            } catch(Exception ex) {
                  System.err.println("Falha a acessar o sistema de
arquivos. Erro: " +
                              ex.getMessage());
            }

      }

      private String tratarCaminho(String path) {
            String p = "";
            for(int i = 0; i < path.length(); i++) {

                  if(path.charAt(i) == '\\') {
                        p += path.charAt(i);
                        p += "\\";

                  }else {
                        p += path.charAt(i);
                  }
            }
            return p;
      }

      // Getters e Setters

      public FileNameExtensionFilter getFiltrarArquivos() {
            return filtrarArquivos;
      }

      public void setFiltrarArquivos(FileNameExtensionFilter
filtrarArquivos) {
            this.filtrarArquivos = filtrarArquivos;
      }

      public JFileChooser getProcurarArquivo() {
            return procurarArquivo;
      }


      public String[] getFiltros() {
            return filtros;
      }


      public void setFiltros(String[] filtros) {
            this.filtros = filtros;
      }

      public void setPath(String path) {
            this.path = path;
      }

      public String getPath() {
            return this.path;
      }

}
```

Apêndice K - Classe da rede neural para reconhecer padrões de caracteres em imagem.

```
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

import org.bson.Document;

import br.com.eric.rna.ActivationFunction;

public class RecoImagem {

	public String descricao;

	private List<Double> pesos;
	private Double bias;

	private Integer tamanho;

	public RecoImagem (List<Conjunto> amostras, String descricao) {

		this.tamanho = amostras.get(0).getEntradas().size();

		this.descricao = descricao;

		iniciarPesos();

		this.bias = Math.random();
		System.out.println("Bias: " + this.bias );

	}

	private void iniciarPesos() {

		this.pesos = new ArrayList<>();
		System.out.println("Inicialidando pesos...");
		System.out.print("[ ");
		for(int i = 0; i < this.tamanho; i++) {

			this.pesos.add((Double) Math.random());

			System.out.printf("%.2f, ", this.pesos.get(i));
		}
		System.out.println(" ]");
		System.out.println(this.pesos.size() + " pesos inicialidos
com sucesso!");
	}

	public double executar(Conjunto amostra) {

		double soma = 0.0;
		for(int i = 0; i < amostra.getEntradas().size(); i++) {
			soma += ((double) amostra.getEntradas().get(i) *
this.pesos.get(i) );
		}

		soma += this.bias;

		return ActivationFunction.sigmoid(soma);
	}
```

```
	public void treinar(Conjunto amostra) {

		double saidaAtual = executar(amostra);

		for(int i = 0; i < this.pesos.size(); i++) {

			double correcao = ((double)amostra.getSaida() -
saidaAtual) * amostra.getEntradas().get(i);

			this.pesos.set(i, this.pesos.get(i) + correcao);

		}
		this.bias += ((double)amostra.getSaida() - saidaAtual);

	}


	public List<Double> getPesos() {
		return this.pesos;
	}

	public Double getBias() {
		return this.bias;
	}

	public Document ToDocument(List<Conjunto> amostras, boolean
treinamento, LocalDateTime inicio, LocalDateTime termino, int epocas) {

		String modelo = "Perceptron";

		String problema = "Problema linear - Reconhecimento de
numeros";

		Document doc = new Document("modelo", modelo);
		doc.append("descricao", this.descricao);
		doc.append("problema", problema);
		doc.append("treinamento", treinamento);
		doc.append("inicio", "" + inicio);
		doc.append("termino","" + termino);
		doc.append("amostras", amostras.size());

		doc.append("pesos", this.getPesos());
		doc.append("bias", this.getBias());

		doc.append("iteracoes", epocas);

		return doc;

	}

}
```

Apêndice L - Classe para tratar do conjunto de amostras salvo no banco de dados MongoDB.

```
import java.util.List;

public class Conjunto {

	public List<Integer> entradas;
	public Integer saida;

	public Conjunto() {

	}

	public Conjunto(List<Integer> entradas, Integer saida) {
		super();
		this.entradas = entradas;
		this.saida = saida;
	}

	public List<Integer> getEntradas() {
		return entradas;
	}

	public void setEntradas(List<Integer> entradas) {
		this.entradas = entradas;
	}

	public Integer getSaida() {
		return saida;
	}

	public void setSaida(Integer saida) {
		this.saida = saida;
	}

	@Override
	public String toString() {

		String formatJson = "{\n" +
                                        "\t\"entradas\" :\t";

		for(int i = 0; i < this.getEntradas().size(); i++) {


			formatJson += this.getEntradas().get(i) + ", ";

		}

		formatJson += "\n\t\"saida\"    :\t" + this.getSaida() +
"\n}";

		return formatJson;
	}

}
```

Apêndice M - Classe para executar a rede neural de reconhecimento de padrões de caracteres os dados no MongoDB.

```
import java.io.PrintWriter;
import java.time.LocalDateTime;
import java.util.List;

import br.com.eric.mongodb.MongoDB;

public class SalvarRedeNeural {


	public static void main(String[] args) {

		String descricao = "Reconhecer numero 1";

		String file = "reconhecer-numero-1.txt";

		String path = "C:\\Amostras\\net\\" + file;

		try {

			PrintWriter arq = new PrintWriter(path);

			List<Conjunto> amostras = MongoDB.getConjunto();

			if(amostras != null) {

				RecoImagem net = new RecoImagem(amostras,
descricao);

				int count = 0;

				LocalDateTime in = LocalDateTime.now();

				do {

					for(int i = 0; i < amostras.size(); i++) {

						net.treinar(amostras.get(i));

						/*
						System.out.printf("Saida: %.8f ->
Alvo: %d\n",

	net.executar(amostras.get(i)),

	amostras.get(i).getSaida()); */
						arq.printf("Saida: %.8f -> Alvo:
%d",

	net.executar(amostras.get(i)),

	amostras.get(i).getSaida());
						arq.println("\n");

					}

					count++;

				} while ( count < 1000 );
```

```
                        arq.close();
                        LocalDateTime te = LocalDateTime.now();
                        for(int i = 0; i < amostras.size(); i++) {
                              System.out.println("Amostra " + (i + 1));
      System.out.println(amostras.get(i).toString());
                        }
                        System.out.println("Total: " + amostras.size() +
" amostras");
                        MongoDB.insert(net.ToDocument(amostras, true,
in, te, count), "rna");
                  } else {
                        System.out.println("\namostras não existem...");
                  }
            } catch (Exception ex) {
                  System.out.println("Erro: " + ex.getMessage());
                  ex.printStackTrace();
            }
      }
}
```

Apêndice N - Classe que recupera os dados da rede neural treinada para reconhecer padrões de novas entradas de imagens.

```
import java.util.List;

import org.bson.Document;

import com.mongodb.MongoException;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;

import br.com.eric.mongodb.MongoDB;
import br.com.eric.rna.ActivationFunction;

public class NeuronioTreinado {

	public Double bias;
	public List<Double> pesos;

	@SuppressWarnings("unchecked")
	public NeuronioTreinado(String descricao) {


		try {

			MongoDatabase db = MongoDB.getConexao();

			MongoCollection<Document> colecao =
db.getCollection("rna");

			Document buscar = new Document("descricao",
descricao);

			MongoCursor<Document> iterador =
colecao.find(buscar).iterator();

			while(iterador.hasNext()) {

				Document doc = iterador.next();

				this.setBias(doc.getDouble("bias"));
				this.setPesos(doc.get("pesos",
List.class));
			}

			System.out.println(formatJson());

		} catch (MongoException e) {
			System.out.println("Infelizmente ocorreu um erro." +
				"\nErro: " + e.getMessage());
		}

	}

	public double run(List<Integer> entradas) {

		double soma = -1.0;

		if(entradas.size() == this.getPesos().size()) {
```

```
                        soma = 0.0D;
                        for(int i = 0; i < this.getPesos().size(); i++) {
                              soma += (double)entradas.get(i) *
this.getPesos().get(i);
                        }
                        soma += getBias();
                        soma = ActivationFunction.sigmoid(soma);
                  }

                  return soma;
            }



            public Double getBias() {
                  return bias;
            }

            public void setBias(Double bias) {
                  this.bias = bias;
            }

            public List<Double> getPesos() {
                  return pesos;
            }

            public void setPesos(List<Double> pesos) {
                  this.pesos = pesos;
            }

            public String formatJson() {
                  String json = "Perceptron Treinando \n{";

                  if(this.getPesos() != null) {

                        json += "\t\"bias\":\t" + this.getBias() + "\n";
                        json += "\t\"pesos\":[\n";
                        for(int i = 0; i < this.getPesos().size(); i++) {

                              if(i < this.getPesos().size() - 1) {
                                    json += "\t\t" + this.getPesos().get(i) +
",\n";
                              }else {
                                    json += "\t\t" + this.getPesos().get(i) +
"\n";
                              }

                        }
                        json += "\t]\n";

                  } else {
                        json += "null";
                  }

                  json += "}";

                  return json;
            }

}
```