UNIVERSIDADE DO VALE DO TAQUARI – UNIVATES

CURSO DE SISTEMAS DE INFORMAÇÃO

# IDENTIFICAÇÃO DE DOENÇAS NA SOJA UTILIZANDO INTELIGÊNCIA ARTIFICIAL POR MEIO DE ANÁLISE DE IMAGENS

Bruno Germano Neumann

Lajeado, junho de 2019

Bruno Germano Neumann

# IDENTIFICAÇÃO DE DOENÇAS NA SOJA UTILIZANDO INTELIGÊNCIA ARTIFICIAL POR MEIO DE ANÁLISE DE IMAGENS

Trabalho de Conclusão de Curso apresentado ao Centro de Ciências Exatas e Tecnológicas da Universidade do Vale do Taquari – Univates, como parte da exigência para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Evandro Franzen

Lajeado, junho de 2019

# RESUMO

O uso de Inteligência Artificial para resolução de problemas complexos vem ganhando força no mundo inteiro. A possibilidade de alcance de aplicação de algoritmos de Aprendizado de Máquina é vasta. Desse modo diversas áreas que antes não estavam conectadas com esse tipo de tecnologia começam a se beneficiar de suas competências. Uma dessas áreas é a agricultura, onde a necessidade de resolução de problemas no cultivo é fundamental para o sucesso da colheita. Especialmente na cultura de soja, onde é comum o aparecimento de doenças e pragas ao longo do seu cultivo, a identificação de qualquer irregularidade na planta é importante para se utilizar o corretivo adequado e garantir produtividade na lavoura. Este trabalho visou o reconhecimento de doenças na soja, a partir da análise de imagens da folha, utilizando algoritmos que implementam a arquitetura de uma Rede Neural Artificial (RNA). O uso desse tipo de algoritmo é apoiado pela literatura e por empresas que já o utilizam para tal fim. O trabalho envolveu o desenvolvimento de um software para o treinamento de várias imagens em uma RNA, implementada a partir de uma ferramenta pronta, em conjunto com uma interface de usuário para poder classificar as imagens. Foram realizados experimentos para validação da acurácia da ferramenta, dos quais obteve-se sucesso de acordo com as expectativas esperadas.

**Palavras-chave:** Inteligência Artificial. Aprendizado de Máquina. Rede Neural Artificial. Soja.

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE GRÁFICOS

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| AG | Algoritmo Genético |
| BC | Base de Conhecimento |
| CNN | Rede Neural Convolucional (*Convolutional Neural Network*) |
| GPU | Unidade de Processamento Gráfico (*Graphics Processing Unit*) |
| HTML | Linguagem de Marcação de Hipertexto (*Hypertext Markup Language*) |
| IA | Inteligência Artificial |
| JS | JavaScript |
| MCP | Modelo do neurônio artificial de McCulloch e Pitts |
| RC | Representação do Conhecimento |
| RNA | Rede Neural Artificial |
| SBC | Sistema Baseado em Conhecimento |
| SE | Sistema Especialista |
| TF | TensorFlow |
| VLSI | Very Large Scale Integration |

# SUMÁRIO

# 1 INTRODUÇÃO

Uma das características fundamentais para a evolução humana ter sido bem sucedida é a visão, uma habilidade que nos dá a capacidade de identificar o que vemos, determinar espaço, profundidade, diferenciar cores, reconhecer ambientes, ativar lembranças. A visão envia sinais elétricos para o cérebro, que é o responsável pela criação da imagem obtida no ambiente, juntamente com a propriedade de processamento de situações, análise de hipóteses e formulação de decisões em milésimos de segundo. Com um fantástico poder de processamento, através de seus bilhões de neurônios, junto com uma grande capacidade de armazenamento, o cérebro humano ainda é muito superior ao melhor supercomputador da atualidade.

Um dos aspectos pelos quais somos únicos são os sentidos, que nos dão a capacidade de envio de informação para o cérebro através de algumas fontes diferentes, como visão, audição, paladar, tato e olfato. Voltando à visão humana, na qual podemos absorver diversas informações e processá-las através do cérebro, em um computador essa entrada de informação se dá na forma de bits, uma passagem de dados que assume somente duas possibilidades, os números 0 ou 1, ou uma carga elétrica positiva ou negativa, assemelhando-se de certa forma ao nosso cérebro, que envia impulsos elétricos como forma de transporte de informações.

A aproximação entre homem e máquina sempre foi um assunto abordado pela ciência. Devido a sua complexidade, seria um desafio tecnológico construir um computador que se igualasse ao nosso cérebro, mas existem situações em que o computador sempre o vencerá, que por sua vez foram circunstâncias que levaram à construção do mesmo, situações que nada mais são do que a realização de tarefas,

uma grande quantidade delas. Computadores realizam, a partir de instruções dadas por um humano, tarefas com eficácia e exatidão, de forma melhor do que pessoas atualmente fazem. Com a capacidade de realização de inúmeras tarefas, e com o crescimento tecnológico como viés, a necessidade de trazer certo conhecimento ao computador como caráter decisório sempre foi uma questão abordada por pesquisadores. Para Rezende (2005), o conhecimento produz o uso claro da decisão, podendo ser representado por uma série de interpretações através de estruturas de dados, fornecendo os mecanismos para um sistema poder otimizar o alcance de um objetivo e resolvê-lo.

A partir dos anos 60, houve a necessidade de resolução de problemas complexos, não mais simples tarefas, mas uma série de técnicas heurísticas que seriam capazes de fazer com que o software pudesse manipular a informação e demonstrá-la na forma de conhecimento. Tendo como característica a solução de um problema restrito a uma área específica, trazendo consigo a manipulação do conhecimento de um profissional humano. Surgem os primeiros Sistemas Especialistas, traçando uma linha tênue com a Inteligência Artificial moderna.

A proposta de um Sistema Especialista (SE) é muito bem constituída e ganhou seu espaço comercial. A representação do conhecimento específico de um humano, buscando soluções a partir da sua base de dados, necessitam uma grande quantidade de conhecimento aplicado (ARTERO, 2009). Os SEs possuem objetivos concretos, que apoiam fortemente seu uso, além de servirem como um precioso arquivo de conhecimento, por valer-se da experiência de um especialista, podem servir também como um assistente ao próprio profissional, ou mesmo tornar apta uma pessoa que não entende do assunto a entregar uma resposta.

Existem certos fatores que tornaram os SEs difíceis de serem implementados como sendo sistemas para resolver problemas de forma definitiva, como a dependência de um ou vários especialistas, trazendo consigo um alto investimento devido ao uso intensivo de horas destes profissionais. A dificuldade de construção devido às várias formas de representação de conhecimento, podem levar a resultados frágeis e por sua vez à ineficiência de todo um projeto. Além de regras específicas, um especialista conta também com outros tipos de conhecimento, que

nem sempre podem ser abordados. Essas circunstâncias são conhecidas como sendo os principais problemas de um SE, que por sua vez, são mais comumente utilizados como assistentes na tomada de decisão, e acredita-se já terem atingido sua maturidade (REZENDE, 2005).

Durante a resolução de um problema, é bastante comum lidar com a incerteza, para nós mesmos é uma tarefa que exige afinco, normalmente sendo resolvido analisando ocasiões precedentes. Atualmente, para um sistema computacional, atendendo a deficiência dos SEs, o tratamento de incertezas é possível ao levar aprendizado automático para o computador.

A idéia de um computador poder aprender e ter inteligência autônoma não é nova, Alan Turing já havia sugerido essa possibilidade há muitos anos atrás, não de forma explicativa, como se um computador pudesse ter inteligência assemelhando-se à humana, mas sugerindo um jogo no qual se pudesse chegar a um consenso sobre essa questão. Em seu artigo *Computing Machinery And Intelligence,* de 1950, ele propôs a questão se máquinas poderiam de fato pensar, através do *Teste de Turing*[1].

Se um computador pode realizar uma grande quantidade de tarefas, adicionar uma habilidade que o faz trabalhar de forma autônoma e capaz de decidir problemas, tanto para absorver a entrada de dados quanto para dar a resposta final, lidando com incertezas, seria uma aproximação melhorada daquilo que os humanos podem fazer. Para ser concebido como inteligente, um computador deve ter determinadas características que o aproximem de uma pessoa inteligente, como o raciocínio, realizar inferências, resolver e prever problemas, armazenar conhecimento e aprender com ele, conseguir se comunicar, definir suas próprias ações e saber interpretar através de uma entrada de dados. Qualquer uma destas características aplicadas a um sistema computacional, já o torna uma máquina inteligente. Mas de modo em que essas são habilidades abstratas, implementar qualquer uma torna-se uma tarefa considerada não trivial (ARTERO, 2009).

---

1 O Teste de Turing consiste em um jogo de perguntas e respostas. O interlocutor realiza uma rodada de perguntas para um humano e para um computador, sem saber a identidade de cada um. Se ao final da série de perguntas e respostas o interlocutor não souber diferenciar com clareza quem é humano e quem é o computador, então pode-se dizer que o computador possui inteligência artificial (TURING, 1950).

Os avanços obtidos com o crescimento do poder computacional possibilitam resoluções de problemas através de uma extração de conhecimento em grandes quantidades de dados. No decorrer das últimas décadas, as instituições se depararam com isso, com um volume muito grande de informações em seus bancos de dados, a necessidade de extrair um significado sobre tudo aquilo era de extrema importância para a vitalidade da mesma, sobretudo para manter o aspecto competitivo.

Nos dias de hoje quem possui informação possui certo poder, já que o trabalho se tornou orientado pela informação e pelo conhecimento, mas essa informação não é trazida de forma corriqueira, para chegar no nível de interesse ela passa por um crivo trabalhoso e muito importante. Sem essa preparação não seria possível abstrair coerência em análises posteriores, a implementação de algoritmos computacionais anteriormente utilizados somente sob a forma teórica, abrem precedentes para decisões baseadas em evidências, a partir de análises sucessivas, a fim de encontrar padrões. Esse acúmulo de dados por meio de algoritmos, visando um conhecimento artificial, vem sendo chamado pelos pesquisadores de Aprendizado de Máquina.

Para a aquisição automática de conhecimento não existe um algoritmo capaz de efetuar a melhor análise para qualquer problema, é necessário uma compreensão de suas limitações (REZENDE, 2005). É importante notar que o Aprendizado de Máquina é possível ao entrar no processo de indução, caracterizado pelo raciocínio a partir de fatos históricos, experiências vindas de observações e exposições. A partir desses fatores, cria-se o processo cognitivo, a aquisição de conhecimento, que pode ser administrado por diversos recursos. Como demonstrativo, utilizando a cognição como ponto central, o cérebro biológico atua utilizando as mesmas ferramentas. Inspirado nisso, evidencia-se pela apresentação de um sistema de neurônios interconectados, o algoritmo de uma Rede Neural Artificial.

Uma Rede Neural Artificial (RNA) possui a capacidade de aprender e adaptar-se, além de conseguir organizar-se através de grupos de conhecimento obtidos com a experiência dos dados processados, tendo potencial de extrair conhecimento. Uma

RNA consiste em um conjunto de neurônios interconectados, que se comunicam através de sinapses, estes neurônios são ativados de acordo com a entrada de novas informações e podem se auto-ajustar, em um processo de treinamento. Segundo Braga, Carvalho e Ludermir (2011), alguns fatores foram os responsáveis pelo grande interesse no assunto de RNAs, como o avanço da tecnologia microeletrônica, que permitiu a simulação de como os neurônios trabalham fisicamente.

As RNAs possuem aplicações em diversas áreas econômicas, como indústria, agricultura e setor de serviços. A crescente utilização se deve ao fato de que elas desoneram pessoas de trabalhos complexos, diminuindo gastos e sobretudo erros humanos. Planos de saúde, por exemplo, podem usar as RNAs para diagnosticar o melhor plano para o paciente, levando em conta fatores atribuídos ao seu treinamento, o que elimina um trabalho humano demorado e sujeito a ser ineficiente para a empresa e para o paciente. A indústria pode utilizar RNAs para identificar inconformidades em matérias-primas, alertar sobre manutenções preventivas, servir como previsão de produção, analisar divergências entre um produto ideal e com falhas, garantindo mais qualidade.

A forma de aplicação que tornou as RNAs conhecidas no mundo inteiro, é o reconhecimento de imagens. Os carros autônomos da Google foram possíveis devido a capacidade de reconhecimento do ambiente ao redor do carro, como identificação de outros carros, caminhões, motos, bicicletas, pessoas, animais, etc. Outras empresas como Facebook, vem utilizando RNAs para reconhecimento de rostos de pessoas em fotos. Na medicina estuda-se o uso de reconhecimento de imagens com RNA para identificação de câncer de pele (ESTEVA et al, 2017). Na agricultura já existem sistemas para identificação de doenças em plantas para toda uma lavoura, as imagens são obtidas através de satélite e são processadas utilizando tecnologias que geram imagens em infravermelho para checagem completa do solo.

## 1.1 Definição do problema

O uso de Inteligência Artificial (IA) para ajudar (ou mesmo substituir) o trabalho humano não é uma decisão fácil de ser realizada, envolvendo em muitos casos um custo elevado e muito tempo para ser implementado. Na agricultura como já descrito, um sistema que utiliza visão computacional para analisar uma plantação e saber o estado da mesma, requer um alto investimento, o que torna a IA disponível somente sob o domínio de grandes empresas do agronegócio. Com o intuito de analisar a presença de doenças e pragas, existem acompanhamentos de agrônomos para o pequeno e médio produtor. Esses acompanhamentos podem gerar altos custos para o produtor, bem como estarem sujeitos à disponibilidade do profissional. As doenças que ocorrem em plantas são por muitas vezes tratadas incorretamente, justamente por não haver o devido conhecimento sobre o que é essa doença de fato. O diagnóstico inadequado pode resultar em um tratamento prejudicial à lavoura, através do uso generalizado de pesticidas incorretos para a resolução do problema, comprometendo a lavoura e podendo prejudicar o solo para próximas safras.

O Brasil é o segundo maior produtor mundial de soja, uma cultura fomentada pelo grande, médio e pequeno produtor. Entre outras características, os campos de soja são facilmente atingidos por centenas de doenças, entre fungos, bactérias e vírus, que comprometem lavouras inteiras (EMBRAPA, 2018). Apesar disso, ainda não há nenhum software, que utiliza IA, acessível ao pequeno e médio produtor rural, que possa auxiliá-lo no cultivo da soja.

## 1.2 Objetivos

Considerando os aspectos apresentados, que podem causar impactos econômicos em pequena e larga escala, o presente estudo teve como objetivo geral a utilização de Aprendizado de Máquina para avaliar as condições da soja a partir da

coleta de imagens das folhas desta planta, a fim de identificar corretamente pragas e doenças em desenvolvimento.

Os objetivos específicos são categorizados em:

- Pesquisar a bibliografia na área de Inteligência Artificial para apoiar a decisão quanto aos algoritmos e soluções mais adequadas para este estudo;
- Pesquisar a ferramenta computacional que auxilie no processo de implementação de uma RNA;
- Implementar um software que servirá de interface para os treinos, classificações, testes e análises, utilizando a biblioteca pesquisada;
- Treinar uma RNA com imagens suficientes para gerar uma base de dados consistente, com o intuito de obter o melhor resultado possível na resposta final;
- Avaliar a eficácia do software criado a partir da análises dos resultados.

## 1.3 Estrutura do trabalho

Este trabalho está estruturado em seis capítulos. Começando pelo capítulo de introdução, no qual são apresentados fundamentos sobre Inteligência Artificial, enfatizando a importância da utilização de redes neurais para reconhecimento de padrões, assim como a exposição do problema tratado neste trabalho e a justificativa para realizá-lo. No segundo capítulo encontra-se o referencial bibliográfico, onde é descrito o estado da arte na literatura, fundamentado pelos processos que existem e foram estudados ao longo dos anos para resolução de problemas com uso de IA. O terceiro capítulo apresenta a metodologia que será aplicada na resolução do problema proposto, acompanhado de trabalhos relacionados que utilizaram redes neurais para reconhecimento de imagens. O quarto capítulo trata do desenvolvimento do software utilizado para treinamento das imagens da soja e classificação das mesmas. O quinto capítulo mostra a

investigação dos resultados obtidos com a ferramenta criada, explorando resultados esperados. O sexto e último capítulo apresenta as conclusões do trabalho, a partir do desenvolvimento, complementando com propostas de melhorias no estudo e continuação para implementações futuras.

# 2 REFERENCIAL TEÓRICO

O referencial teórico deste trabalho teve como objetivo apresentar conceitos sobre Inteligência Artificial (IA), trazendo diversos fundamentos de autores sobre temas que explicam conceitos teóricos gerais sobre o assunto. A literatura abordada tratou desde formas de representação de conhecimento, até a forma de aquisição de conhecimento por meio de algoritmos que operam em Redes Neurais Artificiais.

O propósito da IA é a construção de padrões ou algoritmos que necessitam de máquinas para desempenhar trabalhos de compreensão e aprendizado, trabalhos que os humanos realizam de forma superior. Para tanto, um sistema de IA deve possuir a habilidade de armazenar conhecimento, empregar o conhecimento salvo na resolução de problemas e a partir da experiência obter conhecimento (SAGE, 1990 apud HAYKIN, 2008).

Existem três elementos que fundamentam um sistema de IA (SAGE, 1990 apud HAYKIN, 2008):

1. Representação: é o que diferencia os vários algoritmos usados em situações que exigem compreensão de estruturas simbólicas. A percepção simbólica da IA torna-a eficaz para o entendimento entre homem e máquina. O conhecimento pode ser referenciado em dois modelos essenciais para o entendimento da maior parte dos problemas, o declarativo e o procedimental. O declarativo é constituído por um grupo fixo de fatos, com alguns procedimentos para manuseá-los. A representação declarativa ainda tem a propriedade de que suas sentenças possuem significado próprio, o que

facilita o entendimento humano. A representação procedimental está ligada ao código executável que configura a interpretação do conhecimento.

2. Raciocínio: basicamente consiste na forma em que são solucionados os problemas. Para ser classificado em um sistema que utiliza raciocínio lógico, um sistema deve conter as seguintes condições (HAYKIN, 2008 apud FISCHLER e FIRSCHEIN, 1987):

- O sistema deve comunicar-se com uma grande quantidade de problemas;
- O sistema deve estabelecer conhecimento sobre informações explícitas e implícitas;
- O sistema deve ter uma ferramenta de controle que estabeleça as operações a serem utilizadas para resolver um problema.

3. Aprendizagem: o Aprendizado de Máquina contempla dois tipos distintos de manipulação da informação, o indutivo e o dedutivo. O processamento indutivo envolve exemplos e especificações a partir da informação de entrada, bem como da experiência. No processamento de dados dedutivo, são aplicadas regras para formar fatos exclusivos.

## 2.1 Representação do Conhecimento

A Representação do Conhecimento (RC), conforme Artero (2009), pode ser entendida como uma forma de determinar, reconhecer e corresponder a diversas questões, de forma apropriada. Para Davis, Shrobe e Szolovits (1993), a RC é algo que toma o lugar de algum objeto ou fato descrito cientificamente, de maneira que concede a uma entidade, humana ou não, a capacidade de interpretar por meio de pensamento, que é modelado de acordo com novos conhecimentos.

Existem determinadas características que formam uma RC (REZENDE, 2005):

- Interpretação facilitada ao ser humano;
- Abster-se do funcionamento interno do interpretador de conhecimento;
- Demonstrar solidez, permitindo seu uso mesmo não tratando todos os cenários;
- Uma mesma área de conhecimento deve ter vários modos subjetivos de entendimento, a fim de ser utilizado em várias circunstâncias.

Segundo Artero (2009), independente de ser pessoa ou máquina, a utilização do conhecimento requer algum tipo de exposição, para facilitar o manuseio por essas entidades. Algumas técnicas encontradas na literatura que são atribuídas à RC, servem como modelos propostos para resolução de problemas variados. Nas próximas seções, serão abordados alguns dos modelos mais comumente mencionados na literatura e utilizados em programas de computador.

### 2.1.1 Representação Lógica

A matemática, segundo Rezende (2005), é uma linguagem formal que pode possuir formas de inferência em sua linguagem de forma dedutiva. Uma dedução automática é a característica de um programa de computador de executar inferências conclusivas dentro das regras lógicas da matemática.

O tipo de Representação do Conhecimento mais simples, sendo também o mais utilizado, é a lógica proposicional, que assume um valor binário apenas, verdadeiro ou falso, por exemplo: “O gato está miando”, “A capital do Rio Grande do Sul é Porto Alegre”. Combinando essas proposições com operadores lógicos, do tipo E, OU (negação, condicional, bicondicional), são geradas regras de equivalência que suportam suas vantagens de utilização, devido a possibilidade de realizar análises automatizadas e por sua vez produzir novos conhecimentos (ARTERO, 2009).

Existem dois princípios na matemática que sustentam a representação lógica, segundo Artero (2009):

1. Princípio da não contradição: Uma proposição não pode ser verdadeira e falsa ao mesmo tempo.

2. Princípio do terceiro excluído: Uma proposição pode somente ser verdadeira ou falsa.

O cálculo proposicional é o mais simples de ser utilizado e o preferido durante uma formalização de compreensão para um programa de computador, possuindo potencial expressivo para a construção de um sistema baseado em conhecimento. Mas o problema desse método é que ele lida com as proposições somente como verdadeiras e falsas, não sendo possível ser aplicado em outros sistemas que requerem tratabilidade, ou seja, mesmo que uma lógica proposicional possa operar em caráter decisório um problema, ele é em termos gerais intratável. Um outro problema é que a decisão nem sempre é válida, podendo ter várias representações. Esses problemas podem ser amenizados a partir de um conjunto de sentenças lógicas declarativas, dentro de contextos especiais, que unem uma série de expressões positivas e negativas. Exemplo: uma resposta A é verdadeira somente se B e C forem verdadeiras também (REZENDE, 2005). Outra técnica bastante utilizada para reduzir o impacto da intratabilidade é a aplicação da Lógica de Predicados.

A Lógica de Predicados, ou Lógica de Primeira Ordem, consiste em declarações referente ao relacionamento de objetos, dando propriedade aos mesmos, permitindo a inclusão de fatos sobre um banco de dados de conhecimento e obtendo respostas a partir de eventos anteriores. Esse método é capaz de expor funcionalidades como variáveis, funções e quantificadores, estendendo a lógica proposicional, elevando o potencial representativo de um problema. São aplicadas regras de inferência na informação que possibilita a descrição de relação entre os objetos (ARTERO, 2009).

O método de cálculo de predicados faz a associação entre os objetos e suas propriedades. Por exemplo, armazenando a sentença em Português “gosta de torta” em uma variável A, permite-se criar uma condição de ¬A (negação), resultando na expressão “não gosta de torta”. A Lógica de Predicados permite extrair essa informação para determinar a característica do objeto sobre alguma outra entidade.

A função “gostar” poderia ter como parâmetros: “quem” e “o quê”. Por exemplo, a sentença “Maria gosta de torta” poderia ser expressa como uma função *G(Maria, Torta)*. Dessa forma foi estabelecido um relacionamento entre *Maria* e a *Torta*, podendo ser expressado a mesma ideia a partir de várias outras perspectivas (COPPIN, 2013).

### 2.1.2 Regras de Produção

Proposto pelo matemático Emil Post, em 1943, uma Regra de Produção é um modelo de Representação de Conhecimento muito utilizado na prática por ser considerado simples. O objetivo é converter o problema em um conjunto de estados, a partir de um estado inicial e final, trabalhado com regras de transição no formato SE <condição> ENTÃO <ação> (ARTERO, 2009).

Existem determinadas regras da Representação de Conhecimento que são resultados de combinações de expressões proposicionais. Um exemplo de uma regra poderia ser “se sair o sol o evento será mantido”, inferindo-se a ideia de que o evento será mantido, desde que o tempo esteja bom. Tais regras podem ser armazenadas em um banco de dados, implementadas em um programa de computador, sendo recuperadas pelo usuário através de interações com perguntas e respostas. Utilizando a sentença anterior, podem ser aplicadas regras adicionais a ela, por exemplo: “se sair o sol então vai estar quente”. Assim, através da aplicação de novos fatos, o programa pode concluir que “se sair o sol, estará quente e o evento será mantido”. A utilização de um Motor de Inferência na programação baseada em regras é capaz de manipular os fatos e produzir novos, podendo usar regras novas para chegar a uma conclusão (REZENDE, 2005).

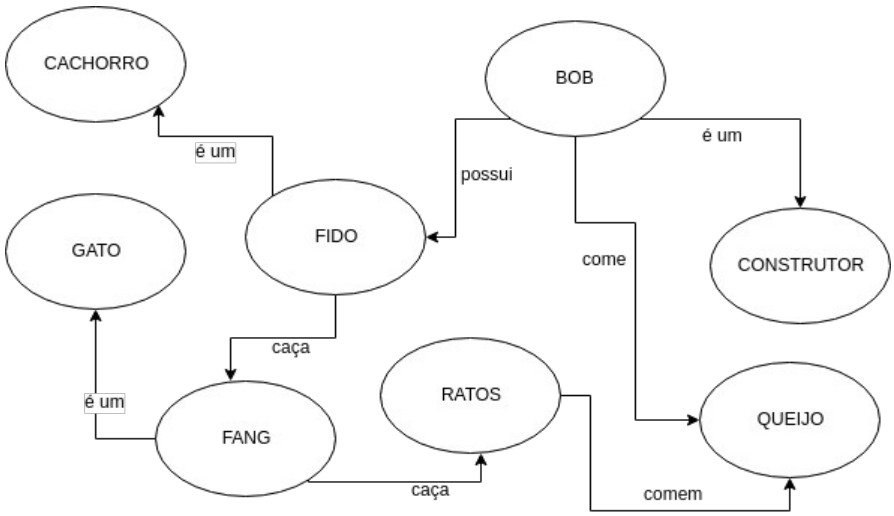
Em áreas em que o conhecimento não apresenta limites precisos e os fatos e conclusões podem ter autonomia variada, como a Medicina, as Regras de Produção podem se fazer presente. Existem certas vantagens de usar esse modelo, que incluem flexibilidade de alteração de módulos, naturalidade e regularidade. Isso se deve ao fato das regras serem peças independentes do conjunto, podendo ser

excluídas, alteradas ou adicionadas, bem como são escritas mantendo um padrão uniforme. Dessa forma, pessoas que não são aptas com a área de conhecimento também podem obter respostas. As desvantagens desse modelo de regra são: ausência de transparência e ineficiência, decorrentes respectivamente da dificuldade de abordagem total do conhecimento e a necessidade de combinação de muitas regras (ARTERO, 2009).

### 2.1.3 Redes Semânticas

Comum de ser utilizada em Inteligência Artificial, uma Rede Semântica consiste em um grafo com vértices, representados como objetos, unidos por arestas que representam uma relação entre estes objetos. A Figura 1 mostra um exemplo simples de Rede Semântica (COPPIN, 2013).

Figura 1 - Exemplo de uma Rede Semântica




Fonte: Adaptado pelo autor com base em Coppin (2013, p. 26).

As redes semânticas originalmente davam suporte à linguagem natural, esse estudo de processamento constituía bases fortes em razão da sua capacidade de representar conjuntos de conhecimento variados. Estima-se que o modelo de uma Rede Semântica seja o mais parecido com o que se imagina acontecer em um cérebro humano. A estrutura das redes são capazes de representar com

semelhança o modelo psicológico da memória associativa, podendo ter ligações entre os objetos com determinadas propriedades em comum (REZENDE, 2005).

Artero (2009), descreve essa relação em comum utilizando o exemplo de um pássaro e um avião, os dois possuem asas, mas o avião possui motor. Ao salvar as informações sem suas devidas propriedades pode resultar em um erro mínimo, bem relativo a um humano quando confunde um avião com um pássaro. Desta forma, quanto maior for a rede, quanto mais conexões ela tiver, menor será a chance de erro. Artero (2009) ainda descreve a hierarquia como uma desvantagem de uma Rede Semântica, a dificuldade de encontrar uma relação ideal no nó com suas conexões, definido pelo tempo.

Segundo Coppin (2013), o percurso sistemático que uma Rede Semântica faz, examinando cada nó da rede, pode muitas vezes não ser eficiente, sendo inadequado para determinadas situações por ter que sempre realizar um ciclo inteiro por esta rede. Para solucionar o problema de encontrar um caminho mais adequado utilizando menos conexões, não percorrendo caminhos cíclicos, existem as Árvores de Decisão.

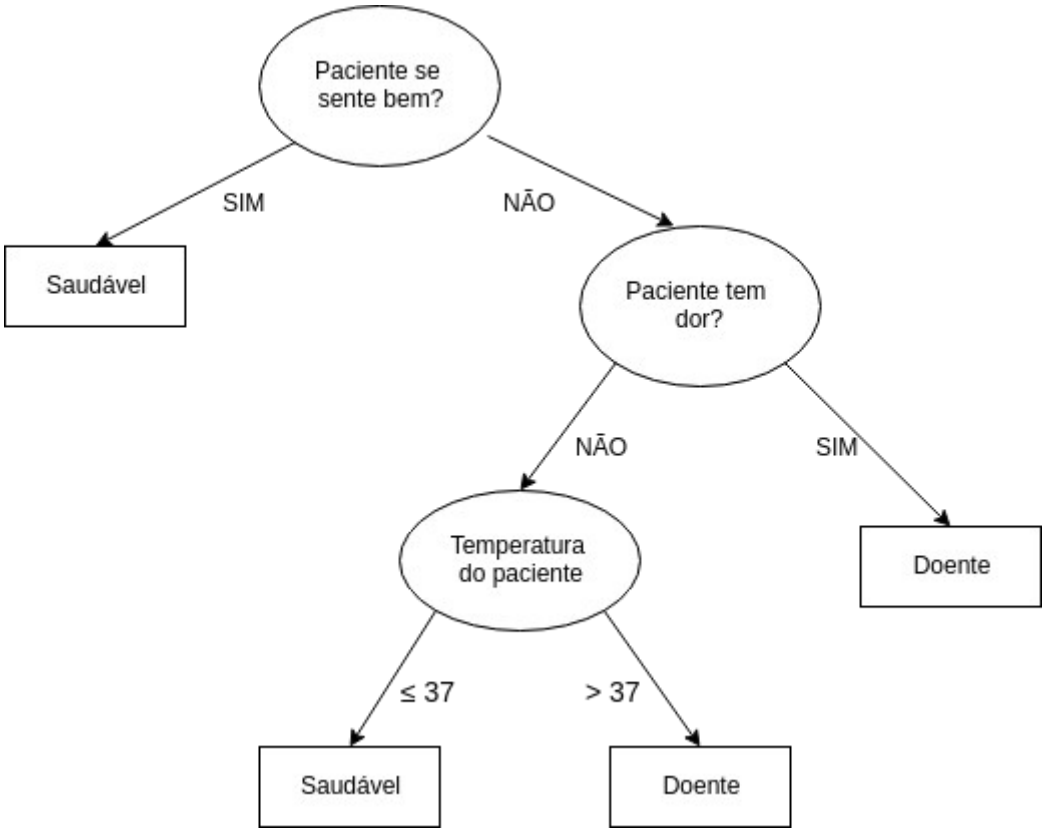
### 2.1.4 Árvores de Decisão

O processo de aprendizado por meio de indução de conhecimento, em ciência da computação, pode ser estabelecido a partir de uma estrutura de modelos, obtendo formas generalizadas sobre um problema aplicado (REZENDE, 2005). Segundo Artero (2009), para a representação de uma grande parte de conhecimento, o uso de uma Árvore de Decisão permite a visão de uma estrutura hierárquica de fácil compreensão. Vários algoritmos de busca utilizam o conceito de árvore, sendo bastante adotados em sistemas computacionais por serem eficientes.

Uma Árvore de Decisão pode ser representada por um grafo, com um *nó folha*, correspondente a uma classe, ou a um *nó de decisão*, consistindo em um teste sobre determinado objeto, estes testes podem ser expandidos em sub-árvores.

A Figura 2 ilustra um exemplo de árvore de decisão simples para diagnosticar se um paciente está doente (REZENDE, 2005).

Figura 2 - Árvore de Decisão para o diagnóstico de um paciente




Fonte: Adaptado pelo autor com base em Rezende (2005, p. 120).

Utilizando a técnica de divisão e conquista[2], as soluções encontradas em cada nó de uma Árvore de Decisão produzem a solução para um problema complexo maior. O que fortalece a eficiência deste método é a capacidade de expansão dos nós, iterando em cada problema recursivamente, possibilitando ajustes nos subespaços correspondentes, podendo realizar previsões para exemplos variados. Dessa forma, uma Árvore de Decisão aplica lógica booleana, atribuindo condições ao longo de todo o caminho entre a raiz e a folha, sendo que as divisões ocorrem nos ramos individuais. De acordo com a situação condicional, um ramo constitui uma regra e uma conclusão. Um conjunto de condições são testes

2 Divisão e conquista é uma técnica computacional que divide um problema maior em partes menores, sendo resolvido por completo de forma recursiva através da solução dos problemas menores.

que consideram operadores lógicos (=, >, <, ≥ etc.) bem como um valor de atributo (FACELI et al., 2011).

A Representação do Conhecimento utilizando uma estrutura de Árvore de Decisão é um modelo que cria condições apropriadas para a resolução de problemas, formando um importante instrumento para o armazenamento de conhecimento em um programa de computador, tanto para um Sistema Especialista, quanto para o Aprendizado de Máquina, que serão tratados respectivamente nas seções 2.2 (Sistemas Baseados em Conhecimento) e 2.3 (Aprendizado de Máquina).

### 2.1.5 Representação de Incertezas

A qualidade da informação é imprescindível ao definir um modelo de conhecimento representativo, uma ausência dessa qualidade torna o processo imperfeito, sendo que a literatura denomina isso como uma incerteza. Não restringindo apenas pelo princípio da incerteza, existe dentro dela, imperfeições que devem ser tratadas individualmente (BITTENCOURT, 2006). Por exemplo, se o problema dado for descobrir a que horas uma determinada loja abre, existem os possíveis cenários de informação:

- Perfeita: “A loja abre às 7h e 30min”;
- Imprecisa: “A loja abre entre 7h e 9h”;
- Incerta: “Eu acho que a loja abre às 7h, sem muita certeza”;
- Vaga: “A loja abre lá pelas 7h”;
- Probabilista: “É provável que a loja abra às 7h”;
- Possibilista: “É possível que a loja abra às 7h”;

- Inconsistente: “Pedro disse que a loja abre às 7h, mas Ana disse que abre às 8h”;

- Incompleta: “Eu não sei que horas abre a loja, mas normalmente as lojas da cidade abrem às 7h”;

- Ignorância total: “Eu não sei que horas abre a loja”.

Uma simples informação pode divergir em diversas situações: quando existe a precisão da informação, informação vinda de fontes conflitantes ou quando ela não existe. Para os humanos, mesmo com todas as incertezas colocadas nos exemplos acima, a resposta pode ser encontrada de forma adequada. Para um sistema, existem alguns modelos formais que tratam as incertezas (BITTENCOURT, 2006).

Nas próximas subseções, serão descritos alguns dos modelos mais comuns encontrados na literatura responsáveis por tratar imprecisões na informação, estes modelos estão profundamente conectados ao Aprendizado de Máquina detalhado na Seção 2.3.

#### 2.1.5.1 Lógica Nebulosa

Desenvolvida em 1965 por Lotfi Zadeh, a Lógica Nebulosa, também conhecida como Lógica Fuzzy, ou Lógica Difusa (estes dois últimos quando aplicado a um sistema computacional), caracteriza-se por investigar o problema da falta de informação, lidando com a imprecisão da solução de uma questão. É um modelo bem sucedido para o desenvolvimento de sistemas com controles de processos complexos, com manutenção facilitada e pouco esforço, sendo que requisitos sofisticados podem ser tratados com simples controladores, a partir da constatação que determinado modelo matemático está suscetível a incertezas (SANDRI; CORREA, 1999).

Constituído por uma base de regras, um controlador nebuloso é composto de um grupo de regras de produção (SE <condição> ENTÃO <ação>), que indicam

ações em várias faixas de valores, em que as variáveis que controlam o estado do problema podem assumir, essas faixas de valores são trabalhadas em conjuntos nebulosos que são chamados de termos linguísticos. Na definição dos termos linguísticos é onde está a maior dificuldade na criação de conjuntos nebulosos. Uma das formas de resolver esse problema é utilizar modelos chamados *neuro-fuzzy*, que possuem parâmetros ensinados com a apresentação de pares de entrada e saída esperada, inseridos em uma rede neural (aprofundado na Seção 2.3.3), internamente computados com operadores de intersecção e junção (SANDRI; CORREA, 1999).

#### 2.1.5.2 Algoritmo Genético

Uma outra maneira de aprendizado de regras com conjuntos nebulosos é a implementação de um Algoritmo Genético (AG). Este tipo de algoritmo computacional permite serem criadas estratégias de busca que se adaptam na medida em que forem avançando o conhecimento, normalmente relacionado de forma abstrata à evolução biológica. Comum de ser utilizado em problemas que requerem otimização de busca, um AG busca primariamente por uma solução que contenha o melhor valor possível (solução ótima), ao não encontrar é buscado então uma solução que seja adequada o suficiente (SANDRI; CORREA, 1999).

Segundo Rezende (2005), Algoritmos Genéticos são dotados da capacidade de determinar e percorrer o ambiente onde está o problema. A utilização de um AG consiste na utilização de técnicas oriundas da computação evolutiva, as quais trabalham paralelamente a partir de uma população de candidatos. Dessa forma, buscas são realizadas em diferentes áreas dentro de um intervalo de solução, sendo que essa operação possui potencial para atingir regiões promissoras dentro do ambiente de busca. Um AG se destaca pela busca eficiente de soluções ótimas em uma grande diversidade de problemas, superando métodos de busca e otimização tradicionais, onde quatro aspectos se destacam (REZENDE, 2005):

1. Não trabalham com parâmetros diretamente e sim com codificações inseridas sobre um conjunto de parâmetros;

2. Operam em população ao invés de um ponto exclusivo;

3. Empregam dados de custo ou recompensa para definir a importância de uma informação, ao invés de utilizar algum conhecimento secundário;

4. Trabalham com critérios de probabilidade ao invés de operar com transições determinísticas.

Rezende (2005) explica que o funcionamento de um AG começa com uma população aleatória de indivíduos, vistos como possíveis soluções para o problema. A população é avaliada durante a sequência de evolução e para cada indivíduo é atribuída uma nota, caracterizando-se pela sua capacidade de adaptação ao ambiente. Seguindo a mesma lógica da seleção natural proposta por Charles Darwin em 1859, os indivíduos mais adaptados são mantidos, enquanto os menos adaptados são desconsiderados.

Coppin (2013) explica um AG como uma população de cromossomos, em que cada um possui diferentes genes. Um AG então provoca a alteração desses cromossomos realizando cruzamentos e mutações a fim de criar uma nova geração de cromossomos melhor do que a anterior. O processo é repetido até o ponto de uma solução caracterizada como ótima ser encontrada.

#### 2.1.5.3 Lógica Probabilista

A teoria das probabilidades é utilizada para discussão e determinação de situações em que não se tem 100% de certeza. Os operadores lógicos não ajudam em hipóteses em que existem incertezas. Por exemplo, a expressão A→B, indica que se A for verdadeiro, B também será. Mas se não existe a certeza que A é verdadeiro, então esta expressão perde sua consistência (COPPIN, 2013).

O modelo probabilístico baseia-se no princípio de ordenação e é determinado a ser aplicado de forma ótima, fundamentado na hipótese de relevância independente entre um evento e outro. Apesar de depender da precisão das evidências probabilísticas, é um modelo que possui bom desempenho através desse comportamento ordenado (CARDOSO, 2004).

Segundo Bittencourt (2006), uma medida de probabilidade possui duas interpretações, a frequentista e a subjetiva. A frequentista é bastante utilizada em dados estatísticos, onde impõem-se limites de frequência das ocorrências, através de uma amostra de dados. A interpretação subjetiva é a mais comum de ser utilizada em Sistemas Baseados em Conhecimento. Representado por crenças em um determinado indivíduo, admitindo hipoteticamente que esse indivíduo abastece as probabilidades aos eventos empregados no modelo. É na interpretação subjetiva, caracterizado por redes de crença, que se enquadra o Teorema de Bayes, utilizado em situações em que é necessário um tratamento de imperfeições da informação.

#### 2.1.5.4 Teorema de Bayes

O Teorema de Bayes estabelece relação com causa e efeito, de maneira que, havendo a compreensão do efeito, revela-se a probabilidade de sua causas. O efeito disso é o que o torna influente para determinação de doenças e também o torna proveitoso para estabelecer resultados de medições das mesmas (LUGER, 2013).

Matemático e teólogo, Thomas Bayes criou no século XVIII o teorema que é amplamente utilizado até hoje em situações em que não existem certezas. Mais precisamente, ele calcula a probabilidade de uma proposição ser verdadeira ou de um evento acontecer. Esse teorema pode ser declarado da seguinte forma (COPPIN, 2013):

$$P(B \vee A) = \frac{P(A \vee B).P(B)}{P(A)}$$

Coppin (2013), descreve o exemplo de um diagnóstico médico para explicar a utilização do Teorema de Bayes. Ao estar gripada, uma pessoa geralmente tem 80% das vezes temperatura alta, assim, utilizando a letra A para representar “tenho temperatura alta” e B para “tenho gripe”, é expressa a primeira sentença:

$$P(B \vee A) = 0{,}8$$

O número decimal 0,8 é igual aos 80%, dado o fato de que a teoria da probabilidade delimita números entre 0 e 1. Ao utilizar A e B para representar conhecimento, essa informação tanto poderia ser uma hipótese quanto uma evidência. Supondo existir o conhecimento de que 1 em cada 10.000 pessoas podem contrair o vírus da gripe e que 1 em cada 1.000 fiquem com febre, as expressões em que podemos armazenar essas variáveis podem ser descritas da seguinte forma:

$$P(A) = 0{,}001$$

$$P(B) = 0{,}0001$$

A partir das variáveis do exemplo, pode-se formular a seguinte questão: Constatando que uma determinada pessoa está com febre, qual a probabilidade dela estar gripada? Utilizando o Teorema de Bayes, essa resposta é calculada da seguinte forma simplificada:

$$P(B \vee A) = \frac{P(A \vee B).P(B)}{P(A)} = \frac{0{,}8.0{,}0001}{0{,}001} = 0{,}8$$

O exemplo do diagnóstico médico mostra que se uma pessoa está com febre, não existe a certeza que ela está gripada. O Teorema de Bayes mostrou no exemplo que as chances de determinada pessoa estar gripada, sendo constatado apenas a temperatura alta, são de somente 8 em 100 (COPPIN, 2013).

## 2.2 Sistemas Baseados em Conhecimento

A integração de uma determinada área de conhecimento em um sistema computacional pode envolver a utilização de um ou vários modelos de Representação de Conhecimento, servindo como uma junção em um algoritmo sistemático, incitando um primeiro contato, a partir do modelo teórico representativo. Segundo Rezende (2005), um programa de computador baseado em conhecimento utiliza-o de forma explícita para resolução de problemas, constituindo a propriedade de manejar informações de forma inteligente.

Um Sistema Baseado em Conhecimento (SBC) é utilizado em casos em que não é requerido um profissional humano especialista, ou que o trabalho do mesmo ou de vários é inviável. A implementação necessita de um acervo considerável de informações, sendo que a quantidade e a consistência são importantes nesses casos.

De acordo com Rezende (2005), é necessário haver uma distinção de duas operações utilizadas no processo da construção de um SBC, a capacidade e a orientação do mesmo. A primeira operação consiste em determinar se com o conhecimento adquirido pode-se chegar a uma conclusão, ou se é passível de gerar novos conhecimentos. A resolução de problemas somente com o uso dessa capacidade não satisfaz, pois existe a necessidade de eficiência nos resultados. Para isso é utilizado a segunda operação, chamada de método para resolução de problemas, com o intuito de guiar corretamente o caminho a ser percorrido, considerando somente conclusões relevantes.

Um método para resolução de problemas também é uma classe de conhecimento, que nada mais é do que a habilidade de resolver o problema. Isso se dá por meio de pesquisas heurísticas, as quais contém regras aplicadas que ajudam na redução do trabalho. Para um SBC atuar de forma satisfatória, é fundamental haver qualidade na heurística construída, isso depende essencialmente da sabedoria externa induzida ao sistema. (REZENDE, 2005).

Um SBC assemelha-se com um sistema convencional, dado pela forma em que é construído. Um sistema de *Business Intelligence*, por exemplo, necessita de uma experiência inicial humana na sua construção, mas a metodologia aplicada em suas operações de análise podem ser moldadas de acordo com as necessidades da entidade que o utiliza, sem a necessidade de um consultor de negócios especialista. Dessa forma o conhecimento adquirido será de bom uso, caso as regras aplicadas sobre ele forem consistentes.

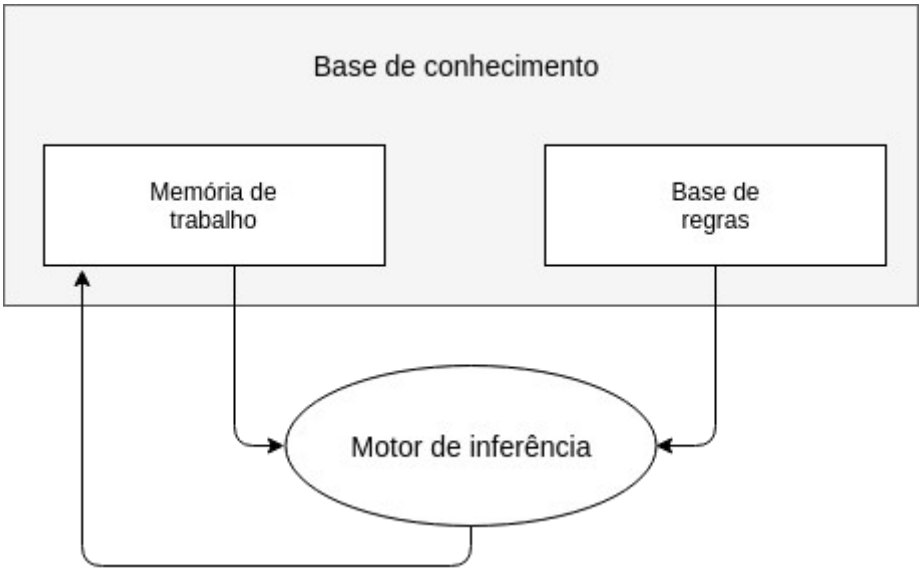
### 2.2.1 Sistemas Especialistas

Um Sistema Especialista (SE) atua em aplicações que requerem a manipulação de um domínio específico de conhecimento, contendo um grande escalonamento de especializações. Um SE pode ser considerado como um Sistema Baseado em Conhecimento que tem a função de resolver problemas que requerem a experiência de um profissional humano, necessitando de uma grande interação deste profissional (REZENDE, 2005).

Fernandes (2005), descreve que a utilização de um SE traz alguns benefícios, como: agilidade na resolução de problemas, fundamentação de uma base separada de conhecimento, garantia de estabilidade e flexibilidade. Além disso, um SE exige menos pessoas para operacionalizar o sistema, assim como previne a representação humana (não especialista) de regras condicionais.

De modo geral, conforme ilustrado na Figura 3, um SE denota de uma arquitetura de três módulos: a base de regras, a memória de trabalho e um motor de inferência. A base de conhecimento de um SE, área em que é retratada o conhecimento, é formada pela memória de trabalho e a base de regras. O instrumento de controle de um SE é o motor de inferência, responsável por aplicar as regras oriundas de informações salvas na memória de trabalho. O motor de inferência em si pode ser implementado com um SE de mais alto nível, para ponderar sobre a aplicação de regras de domínio, através de regras principais denominadas meta-regras (BITTENCOURT, 2006).

Figura 3 - Arquitetura de um Sistema Especialista




Fonte: Adaptado pelo autor com base em Bittencourt (2006, p. 261).

Segundo Medeiros (2018), existem cinco etapas para a construção de um SE:

1. Identificação e definição do domínio do problema: consulta de fontes de conhecimento, especialmente a consulta da experiência do especialista;

2. Aquisição de conhecimento: recolhimento e análise das fontes de conhecimento para produção de um documento que sirva de base para a produção do SE;

3. Organização do conhecimento: decorre de fatos e regras. Um fato apoia a retratação entre declarações de estruturas de conhecimento, normalmente uma proposição de afirmação ou negação. Exemplo: “O tempo está ensolarado”. Uma regra contém sequências lógicas, oriundas das Regras de Produção (descritas na Seção 2.1.2), podendo haver premissas e conclusões em seu formato e ter um fator de confiança ou de probabilidade. Exemplo: SE (temperatura <= 0) E (umidade > 70) E (tempo nublado) ENTÃO (possibilidade de nevar = alta);

4. Implementação: escolha de uma linguagem de programação para implementar o SE, ou escolha de uma ferramenta pronta, sendo que esta

última opção é a mais comum, devido à complexidade de implementar um SE e pelo fato de que um SE pronto já possui um motor de inferência;

5. Aplicação de testes e validação: análise de casos conhecidos através de testes na ferramenta, para validar a informação adquirida pela base de conhecimento.

#### 2.2.1.1 Base de Conhecimento

A sintetização do conhecimento de um especialista humano, constituído por regras e procedimentos para solução de problemas, é armazenada dentro de uma Base de Conhecimento (BC). O conhecimento dessa base é simples de ser alterado, devido à base estar separada da máquina de inferência (FERNANDES, 2005).

A Base de Conhecimento de um SE aplica algumas representações lógicas de conhecimento vistas na Seção 2.1, como Regras de Produção, Redes Semânticas, Árvores de Decisão, entre outras.

Artero (2009) explica que embora os modelos de Representação de Conhecimento possam ser aplicados em diversas regras distintas, a utilização de Regras de Produção é a lógica mais utilizada, por ser modular e uniforme. A modularidade aparece tanto no sentido em que cada regra representa um pedaço individual de conhecimento, concedendo propensão para inserção de novas regras, quanto na facilidade de alteração e exclusão de regras. A uniformidade se dá pela utilização de um padrão de representação de todas as regras, possibilitando pessoas que desconhecem o sistema aprender de forma facilitada o conteúdo das regras.

Bittencourt (2006) descreve que a base de regras dispõe de condições que retratam as perguntas para o conhecimento da memória de trabalho, essas perguntas podem envolver instâncias de variáveis, além de inferências. A memória de trabalho pode conter variados tipos de estruturas de dados, adequando-se aos

métodos que representam o conhecimento. As regras podem possuir sintaxe flexível, aproximada da linguagem natural ou mais formais.

A aquisição de conhecimento se torna a parte mais vulnerável de um SE, pois não pode balizar-se apenas de novas regras, mas de uma integração do conhecimento novo com o já existente. Uma solução para esse tipo de problema é a junção de entidades relacionadas, com o intuito de criar um *cluster* de regras de trabalho. A adoção de ponteiros para fazer ligações com elementos que possuem relação um com o outro também é uma forma de integrar a base (BITTENCOURT, 2006).

Um outro aspecto que torna a aquisição de conhecimento sensível é o tratamento de inconsistências, pois podem haver erros de aquisição em como o conhecimento foi abordado. Esses erros podem ser oriundos da respectiva natureza do conhecimento, ou produzidos pela interface de usuário. A classificação de regras de aquisição onde o conhecimento está estabelecido é uma das formas de tratar aquisições de conhecimento incoerentes. A avaliação de uma Base de Conhecimento de forma periódica pode também detectar falhas incorporadas durante o processo de aquisição (BITTENCOURT, 2006).

#### 2.2.1.2 Motor de Inferência

Responsável pelo processamento de perguntas do usuário em um Sistema Especialista, o Motor de Inferência, ou Máquina de Inferência, trabalha com os fatos armazenados na base de conhecimento, com o intuito de formular conclusões a partir dos caminhos estabelecidos na fase de implementação. Os processos inferidos na base, buscam criar novos conhecimentos de acordo com os fatos e suposições. Assim, de acordo com a situação entregue em um estado inicial, o Motor de Inferência transforma em um estado final com a situação desejada, trabalhando por meio de um conjunto de operadores. Esse processo que ocorre entre o estado inicial e o final remete-se exclusivamente a buscar uma sequência lógica de operadores, utilizando modelos de Representação de Conhecimento e

técnicas de buscas heurísticas. Em um ambiente casual, o Motor de Inferência aplica regras de produção que operam utilizando ligações, normalmente definidas de acordo com o objetivo específico a ser estabelecido (ARTERO, 2009).

## 2.3 Aprendizado de Máquina

Os Sistemas Especialistas determinam um período importante que os separam da Inteligência Artificial moderna. A habilidade de poder formular conclusões sobre determinado assunto é uma parte fundamental que diz respeito ao Aprendizado de Máquina. Mas a parte mais custosa de um SE sempre foi a alimentação da base de conhecimento, testes de averiguação e um planejamento complexo para torná-lo flexível o suficiente para adequar-se ao ambiente, sobretudo sob a influência externa de um especialista. A necessidade de um sistema computacional poder adquirir conhecimento sem o auxílio de um profissional de determinada área, aprendendo de forma autônoma, encontrando significado em meio a uma diversidade de aspectos que o levam a diversas aplicações, é o que torna o estudo do Aprendizado de Máquina atrativo. Atualmente quando há algum problema que exige busca por soluções complexas e demoradas, a IA torna-se presente, mostrando que é possível adquirir conceitos, compreensão e decisão sobre algum objetivo a que foi capacitada (REZENDE, 2005).

Dentre os paradigmas do Aprendizado de Máquina está a indução, um método de inferência lógica que possibilita a aquisição de conclusões genéricas a partir de um grupo especial de exemplos, formada pelo raciocínio oriundo de uma definição específica e generalizada. A inferência indutiva é uma das ferramentas essenciais usadas para extrair conhecimento novo e indicar eventos futuros, sendo o artifício mais utilizado pelo cérebro humano para derivação de novos conhecimentos. Realizado a partir do entendimento de exemplos obtidos por um processo externo, o aprendizado indutivo pode ser segmentado em supervisionado e não supervisionado (REZENDE, 2005).

### 2.3.1 Aprendizado Supervisionado

O Aprendizado Supervisionado é uma abordagem utilizada na IA que considera o fornecimento de dados por meio de um professor. Na fase de treino, são entregues ao algoritmo de aprendizado um grupo de exemplos de treinamento onde é conhecido o atributo de classe. Os exemplos são determinados por um vetor de valores de atributos e pelo rótulo da classe agregada. O algoritmo de indução tem como propósito produzir um classificador que estabeleça uma classe de novos exemplos passíveis de serem rotulados, isto é, que ainda não contenham um rótulo (REZENDE, 2005).

Em um algoritmo de rede neural (especificado na Seção 2.3.3), o Aprendizado Supervisionado é usado com o intuito de desenvolver a rede, esta por sua vez aprende ao alterar automaticamente os pesos contidos em suas conexões, classificando de modo objetivo as informações oriundas do treinamento. Desta forma, as redes neurais possuem um poder de generalização com altos índices de assertividade, através de uma ampla capacidade de entrada de dados (COPPIN, 2013).

### 2.3.2 Aprendizado Não Supervisionado

O Aprendizado Não Supervisionado caracteriza-se por instruir-se sem nenhuma mediação humana, utilizado em casos em que se desconhece o modo de classificação dos dados. Esse método pode aprender a classificar uma coleção de dados de entrada sem haver a atribuição de rótulos informados para a classe. Um modelo de exemplo utilizando um método de Aprendizado Não Supervisionado, seria a busca por documentos na Internet. O algoritmo poderia classificar informações semelhantes e prover, de forma automática, uma proposta de conteúdos distintos referidos nos documentos (COPPIN, 2013).

Em oposição ao treinamento supervisionado, que é indicado para trabalhar com classificação através de um conjunto de dados de preparação, o treinamento não supervisionado é mais adequado para tarefas de agrupamento, onde os registros são classificados a partir do comportamento das suas propriedades. Relacionando o Aprendizado Não Supervisionado com um algoritmo de rede neural, o modo como funciona esse treino é orientado a um modelo biológico, visto que quando uma pessoa aprende uma tarefa específica, regiões exclusivas do cérebro são ativadas, sendo acionadas quando é necessário reproduzir novamente a tarefa (ARTERO, 2009).

### 2.3.3 Redes Neurais Artificiais

Uma Rede Neural Artificial (RNA), também conhecida como *conexionismo* ou *sistema de processamento paralelo e distribuído*, constitui um sistema que em determinado ponto lembra a organização de um cérebro humano. Não sendo apoiado por regras, uma RNA cria uma possibilidade aprimorada em relação aos algoritmos tradicionais (BRAGA, CARVALHO e LUDERMIR, 2011).

Da forma que a organização estrutural de uma RNA assimila-se a um cérebro humano, o modo como são trabalhadas as informações por um computador o diferem bastante do mesmo. O cérebro humano é caracterizado por uma habilidade de organização de seus neurônios e a capacidade de realizar processamentos complexos, como reconhecimento de padrões, percepção e controle motor. Um exemplo de processamento complexo da informação é a visão humana, que guarnece uma reprodução do ambiente ao nosso redor, fornecendo informações para interagirmos nele. O cérebro efetua habitualmente funções de reconhecimento perceptivo, como o reconhecimento de um rosto conhecido em meio a outros desconhecidos (HAYKIN, 2008).

O aprendizado de uma RNA geralmente está ligado à adaptação de seus parâmetros, resultado de uma interação externa. Esse processo é interativo, de maneira que a rede desenvolve-se progressivamente na medida em que sofre

interações. O fundamento para a definição do desempenho que qualifica um modelo neural, bem como a ocasião de demarcação do fim de um treinamento, são convencionados pelos parâmetros do treino, como tamanho da rede, quantidade de interações e algoritmo classificador (REZENDE, 2005). Haykin (2008) explica que o algoritmo de aprendizagem de uma RNA empreende operações de conexão entre os neurônios para o processo de aquisição de conhecimento, chamadas de pesos sinápticos, cuja função é alterar os pesos sinápticos da rede para almejar um propósito esperado.

Segundo Haykin (2008), a utilização de uma RNA proporciona particularidades referente ao conhecimento, como elencado abaixo:

1. Não-linearidade: um neurônio artificial não-linear caracteriza a rede inteira da mesma forma, por utilizar a não-linearidade distribuída. Isso é significativo pois a forma em que são inseridos os dados de entrada geralmente não seguem um sentido linear, bem como a estratégia interna adotada pelo algoritmo de aprendizagem.

2. Mapeamento de entrada-saída: a partir de uma aprendizagem supervisionada ou não-supervisionada, acontece a alteração dos pesos sinápticos da rede, a partir de uma série de amostragens de treinamento previamente rotuladas. Cara interação dessas amostras contém um sinal de entrada exclusivo e uma saída esperada, sendo que a partir disso, são apresentados exemplos à rede para ajustes necessários nos pesos, com a finalidade de minimizar a distância para com a resposta esperada. Desta forma, a rede adquire conhecimento construindo um mapeamento de entrada-saída para o problema específico.

3. Adaptabilidade: Redes neurais possuem a habilidade de adaptar os seus pesos sinápticos quando ocorrem novas alterações de informações. Uma rede neural já treinada pode ser novamente treinada para operar em condições alteradas.

4. Resposta a Evidências: na classificação de padrões, uma rede neural pode ser organizada para prover conhecimento sobre a confiabilidade da sua decisão, ou “crença”.

5. Informação Contextual: a própria estrutura de uma rede neural e sua condição de ativação já representam a forma de conhecimento. Para cada atividade de um neurônio, todos os outros sofrem alterações, com isso a informação circunstancial é analisada de forma natural.

6. Tolerância a Falhas: caso um neurônio, ou alguma conexão com ele seja danificada, a qualidade do padrão salvo fica deficiente. Mas com a distribuição da informação pela rede, a degradação do desempenho é amena, usualmente não sendo passível de falhas gerais em toda a rede.

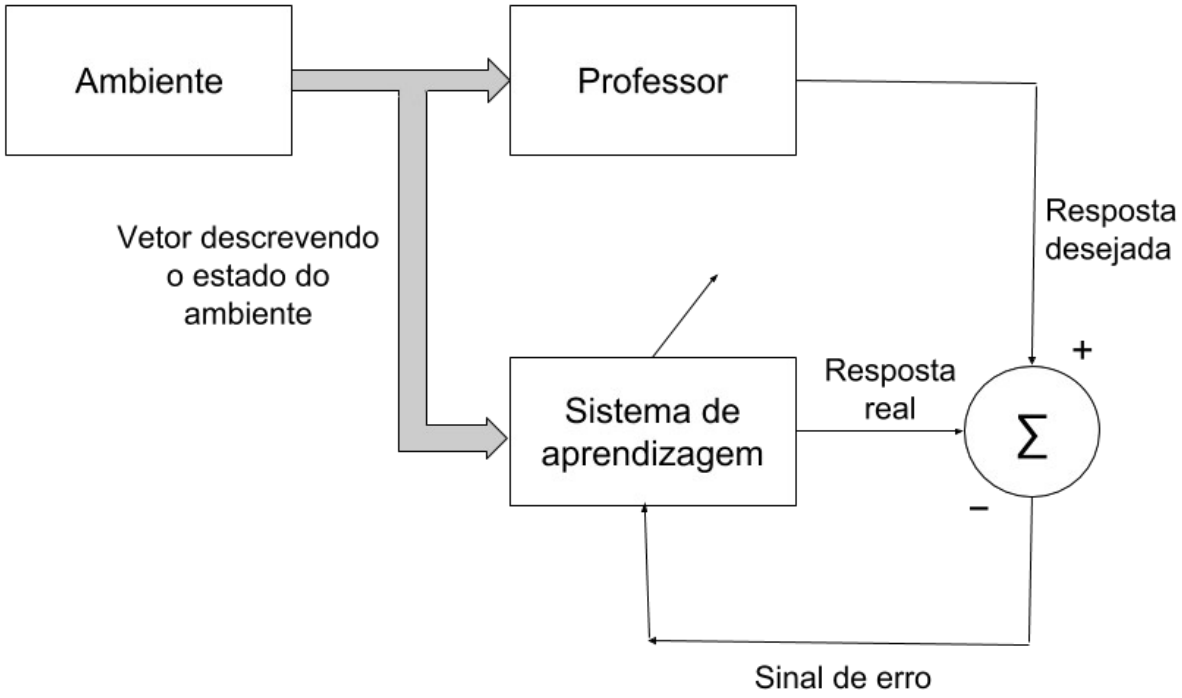
7. Implementação em VLSI: *Very Large Scale Integration* (VLSI) é uma tecnologia capaz de captar ações de fato complexas utilizando uma condição demasiadamente hierárquica. A capacidade de processamento em paralelo de uma rede neural faz com que ela tenha um ótimo desempenho, garantindo com isso uma capacidade de integração de tecnologias em grande escala.

8. Uniformidade de Análise e Projeto: o grande universo de aplicações que uma rede neural atende se dá pelo fato de que uma mesma notação pode ser usada em diversos domínios, possibilitando o compartilhamento de teorias e algoritmos de aprendizagem de áreas distintas. Os neurônios também apresentam um elemento comum para todas as formas de implementação de RNAs.

9. Analogia Neurobiológica: o processamento paralelo flexível a falhas, rápido e significativo de uma RNA com sua estrutura hierárquica formada internamente por neurônios artificiais, bem como a forma de comunicação por sinapses é o que determina a instigação para a analogia com o cérebro humano.

A aprendizagem supervisionada em uma RNA é ilustrada na Figura 4.

Figura 4 - Diagrama em blocos de um aprendizado com um professor




Fonte: Adaptado pelo autor com base em Haykin (2008, p. 88).

Conceitualmente, pode-se tratar o professor como a entidade que possui compreensão sobre o ambiente, sendo este apresentado por uma série de exemplos de entrada e saída. A partir do cenário em que a rede não conhece o ambiente, o professor e a rede são apresentados aos dados de treinamento (vetor de treino) vindos do ambiente. Em consequência do entendimento inicial do professor, ele dá à rede uma resposta esperada a partir do vetor de treinamento, esta resposta retrata a melhor possibilidade possível. Os parâmetros da rede são adequados a partir da ascendência conciliada entre a entrada e o sinal de erro, este por sua vez é o resultado da diferença entre a resposta pretendida e a resposta real da rede. O ajuste é executado de forma iterativa, com o propósito de fazer a rede neural emparelhar-se com o professor. Em uma hipótese em que o resultado seja ótimo, o conhecimento do professor é entregue à rede em forma completa de treinamento, de modo que, a partir do momento em que esse estado é atingido, pode-se isentar o professor e designar a rede a trabalhar por si própria (HAYKIN, 2008).

#### 2.3.3.1 Neurônio artificial

Criado por McCulloch e Pitts (1943), o neurônio artificial tem seu modelo teórico regularmente citado na literatura pela sigla MCP, nome de seus autores.

Segundo Coppin (2013), a modelagem de uma rede neural é relacionada ao cérebro humano, mas os neurônios artificiais possuem menos conexões que os biológicos, posto que atualmente não existem aplicações que implementam a mesma quantidade de neurônios que uma estrutura biológica.

Segundo Haykin (2008), existem três elementos que fundamentam o modelo de um neurônio artificial:

1. Uma série de sinapses ou *elos de conexão* que contenham um peso. Esse peso sináptico está incluído em um espaço que pode conter valores negativos e positivos.

2. Um somador que agrega os sinais de entrada, sendo ponderado pelas sinapses do neurônio.

3. Uma função de ativação que baliza a área de saída de um neurônio, para um valor finito.

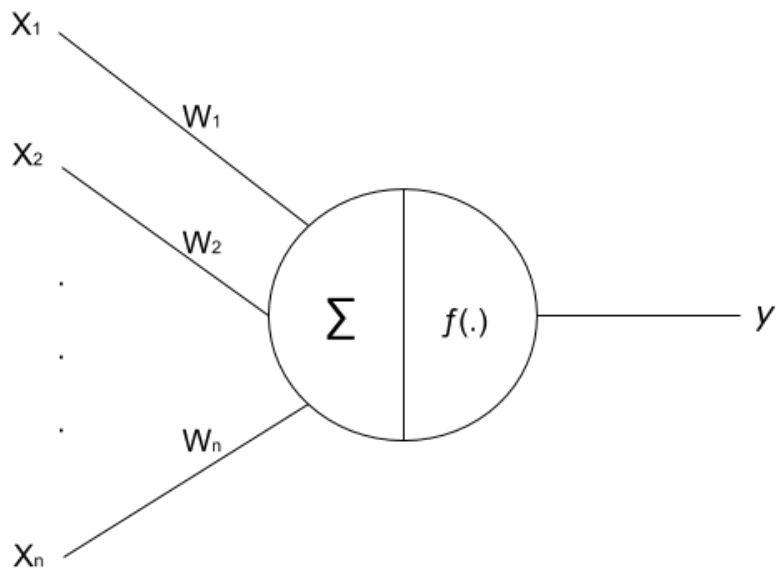
De acordo com Coppin (2013), para cada neurônio artificial de uma rede, também chamado de nó, é inferido um conjunto de entradas, a partir disto, é aplicada a operação chamada de função de ativação aos valores de entrada, cada neurônio possui uma função destas, originando o nível de ativação que é o valor de sua saída. A função degrau, ou função de limiar linear é uma das funções de ativação mais utilizadas, entre as diversas existentes.

A operação executada por uma função limiar linear é essencialmente a verificação da soma avaliada, obtida pelas entradas, com um valor de limiar. Se a soma exceder o limiar, é ocasionado a ativação da saída, senão a saída permanece desativada. Os pesos atribuídos aos neurônios podem ser diferentes, dado que, quanto mais complexos eles forem, maior é a complexidade no espaço de entrada,

para tanto, a flexibilidade que possui a função de ativação para resolução do problema é proporcional a essa complexidade. Caracterizada pela semelhança ao modelo MCP, uma função de limiar linear se restringe a variáveis booleanas, enquanto que a definição do neurônio MCP explica que essa função pode assumir qualquer valor real (BRAGA, CARVALHO e LUDERMIR, 2011).

A Figura 5 mostra um neurônio MCP, ou porta de limiar linear, onde a saída é resultado da aplicação da função de ativação $f(.)$ (BRAGA, CARVALHO e LUDERMIR, 2011).

Figura 5 - Neurônio MCP ou porta de limiar linear




Fonte: Adaptado pelo autor com base em Braga, Carvalho e Ludermir (2011, p. 24).

As funções limiar lineares restringem-se a solução de problemas que encontram-se separados linearmente, isto é, casos em que a solução é conquistada a partir da separação de partes para formalizar uma reta. A entrada para uma porta de limiar linear, conforme visto na Figura 9, é dividida em vetores *X*, dos quais são aplicados os pesos *W*, que levam para uma saída *y* o valor 1 ou 0. Os vetores X permanecem em lados opostos contra uma região de separação linear, os pontos que correspondem a *y* = 1 ficam acima da superfície e os pontos *y* = 0 ficam abaixo. Em ocasiões em que existem muitos valores possíveis de entrada no neurônio (valores grandes em *n*), as funções lineares não podem ser utilizadas, sendo neste

caso a utilização de uma função limiar quadrática a melhor opção, pois aumentam a capacidade computacional das mesmas (BRAGA, CARVALHO e LUDERMIR, 2011).

A inserção de um grande número de parâmetros dá maior flexibilidade em torno das possíveis soluções que podem ser aplicadas a uma função limiar. A correlação entre a quantidade de parâmetros e a capacidade computacional de uma RNA é um critério indispensável na definição da topologia da mesma. Uma rede com poucos parâmetros pode não ser flexível o suficiente para resolver determinado problema. No entanto, uma rede com uma grande quantidade de parâmetros tende a ser superdimensionada, gerando flexibilidade em excesso, o que pode levar a resultados fora da série de treinamento, causando hipóteses inconsistentes. Portanto deve existir um equilíbrio na implementação de um modelo, para não haver generalização fraca e ao mesmo tempo ser flexível (BRAGA, CARVALHO e LUDERMIR, 2011).

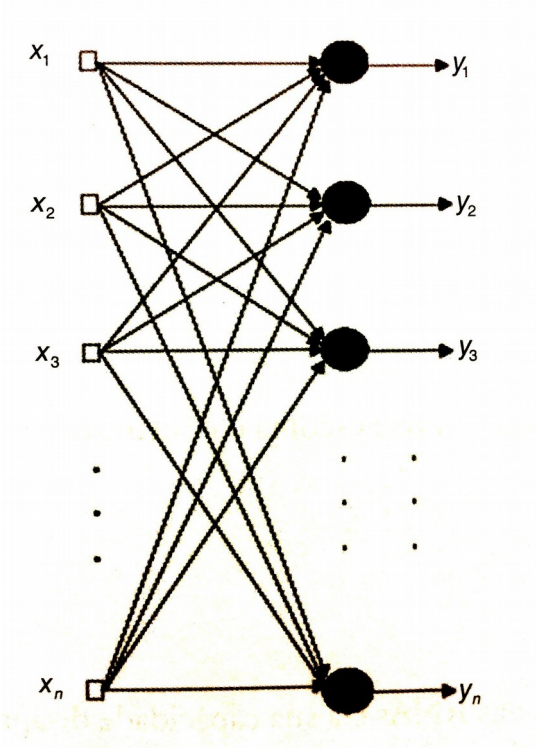
#### 2.3.3.2 Arquitetura de uma RNA

O algoritmo utilizado na aprendizagem de uma rede neural está conectado intrinsecamente com a forma em que os neurônios da rede estão estruturados. A arquitetura natural de uma rede neural transforma ela em um instrumento muito eficiente para a categorização adaptativa de padrões e processamento de sinais. Além disso, uma RNA pode ser construída para alterar os seus pesos sinápticos em tempo real (HAYKIN, 2008).

O modelo estrutural ilustrado de uma rede neural é de grande eficácia para o entendimento das operações que ocorrem dentro dela. A seguir serão apresentados algumas das arquiteturas praticáveis em RNAs.

A Figura 6 ilustra a abordagem mais simples de uma RNA, caracterizando-se por ser de uma única camada de neurônios, sendo alimentada exclusivamente para frente, modelo conhecido como *feedforward* (BRAGA, CARVALHO e LUDERMIR, 2011).

Figura 6 - Arquitetura *feedforward* de camada simples

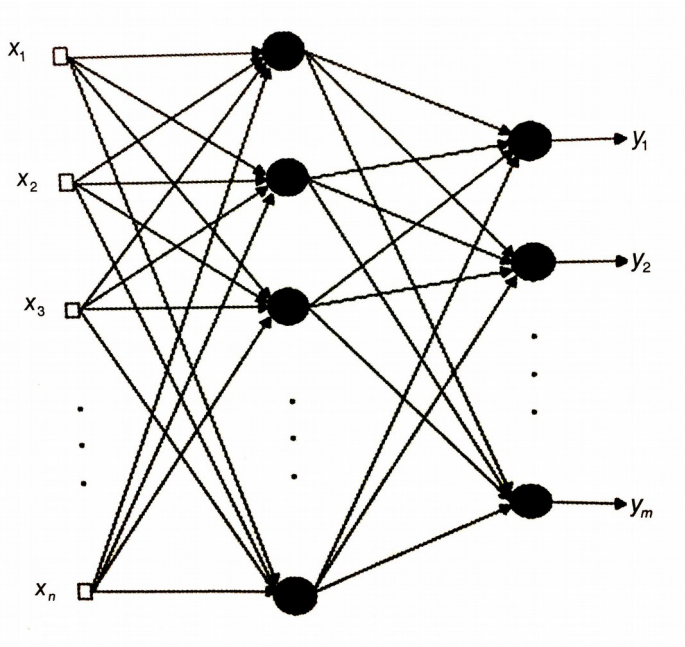



Fonte: Braga, Carvalho e Ludermir (2011, p. 11).

Segundo Braga, Carvalho e Ludermir (2011), estruturas simplificadas como a da Figura 6 são aptas para solucionar problemas multivariáveis com funções conjuntas. Por serem compostas de uma única camada, se restringem a cenários menos complexos.

A arquitetura ilustrada na Figura 7 mostra semelhança com a Figura 6, mas possui mais uma camada de neurônios. Isso dá à RNA uma eficiência de processamento e maior amplitude para usos contínuos (BRAGA, CARVALHO e LUDERMIR, 2011).

Figura 7 - Arquitetura *feedforward* de duas camadas

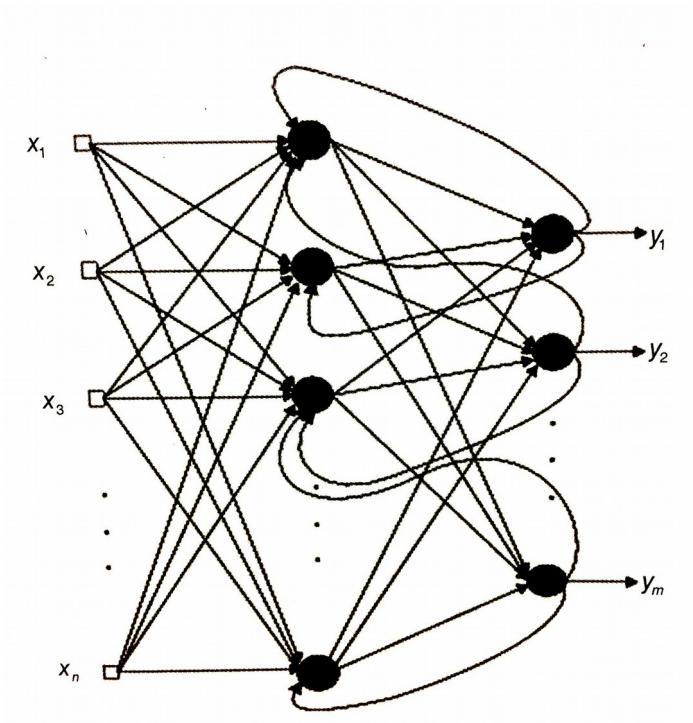



Fonte: Braga, Carvalho e Ludermir (2011, p. 11).

A arquitetura das Figuras 6 e 7 indicam um conceito estático, isto é, não apresentam reincidência internamente em suas estruturas, sendo direcionadas da entrada para a saída, sempre para frente. Nas Figuras 8 e 9, as conexões através dos neurônios de mesma categoria, ou entre a saída e camadas anteriores, são recorrentes (BRAGA, CARVALHO e LUDERMIR, 2011).

A Figura 8 ilustra um exemplo de arquitetura com conexões entre os neurônios recorrentes, onde a saída é consequência não somente da entrada processada nas camadas, mas também do atual estado reprocessado da rede. Essa construção geralmente é aplicada em resolução de problemas que envolvem predições de eventos futuros (BRAGA, CARVALHO e LUDERMIR, 2011).

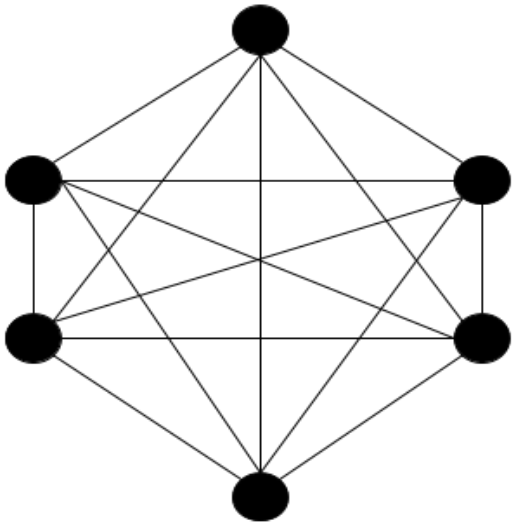
Figura 8 - Rede recorrente entre a saída e a camada central




Fonte: Braga, Carvalho e Ludermir (2011, p. 11).

A Figura 9 ilustra um exemplo de rede com recorrência auto-associativa, no qual a saída de cada um desses neurônios, a partir de um único nível, conecta-se com as entradas de todos os demais. Essa estrutura não apresenta entradas externas como visto nos outros modelos, sendo que o funcionamento é realizado pelo processo de atualização de estado dos neurônios, que se auto-associam (BRAGA, CARVALHO e LUDERMIR, 2011).

Figura 9 - Rede recorrente auto-associativa



Fonte: Adaptado pelo autor com base em Braga, Carvalho e Ludermir (2011, p. 11).

A escolha correta da disposição de uma rede neural para solucionar algum problema vai depender de alguns fatores, como: complexidade do problema, tamanho da entrada de dados, propriedades estáticas ou dinâmicas, conhecimento pressuposto do problema e representação da informação (BRAGA, CARVALHO e LUDERMIR, 2011).

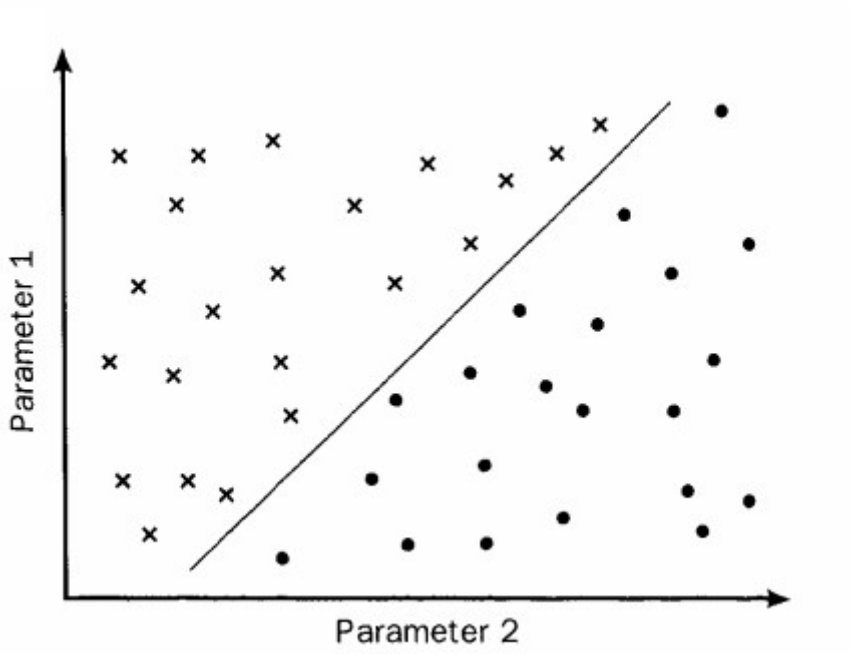
#### 2.3.3.3 A rede Perceptron

Proposto primeiramente por Rosenblatt em 1958, o Perceptron é um classificador binário composto por uma rede simples com somente um neurônio para classificar suas entradas para duas categorias, podendo ter qualquer valor de entrada. A entrada é organizada por uma grade, utilizada para representação de uma imagem, por exemplo, podendo realizar tarefas de reconhecimento simples (COPPIN, 2013). Segundo Artero (2009), a rede Perceptron é a estrutura de rede neural mais conhecida pois seu estudo é simplificado e possui uma carga histórica de literatura e pesquisa.

De acordo com Haykin (2008), o Perceptron é utilizado para classificação de padrões que apresentam-se em lados opostos em um hiperplano[3]. Essencialmente, caracteriza-se por um único neurônio com pesos sinápticos adaptáveis. Utilizado para ajustar parâmetros livres de uma RNA, o algoritmo surgiu em uma operação de aprendizagem feita por Rosenblatt entre os anos 1958 e 1962, para seu modelo de rede. Rosenblatt atestou que caso os vetores utilizados para treinamento do Perceptron são obtidos de duas classes separadas de forma linear, o algoritmo converge e dispõe a área decisória em um hiperplano de duas classes. O Perceptron de fato é elaborado com somente um neurônio, limitado a classificar padrões com duas hipóteses.

A Figura 10 mostra um exemplo de uma classificação linear com Perceptron ilustrado em um hiperplano, onde são classificados dois tipos de elementos.

Figura 10 - Exemplo de treinamento de duas classes com um Perceptron




Fonte: Cross, Harrison, Kennedy (1995, p. 1076).

Para a classificação com mais de duas classes, pode-se expandir a saída do Perceptron incluindo mais neurônios. Mas para o correto funcionamento da rede, as classes necessitam ficar separadas linearmente. O funcionamento elementar do

3 Na computação, um hiperplano separa uma superfície de características de modo robusto e eficiente. Essa competência é um aspecto importante quando aplicado com grandes quantidades de dados, como processamento de imagens e visão computacional (GONÇALVES, et al., 2007).

Perceptron serve como modelo lógico para qualquer tamanho de RNA (HAYKIN, 2008).

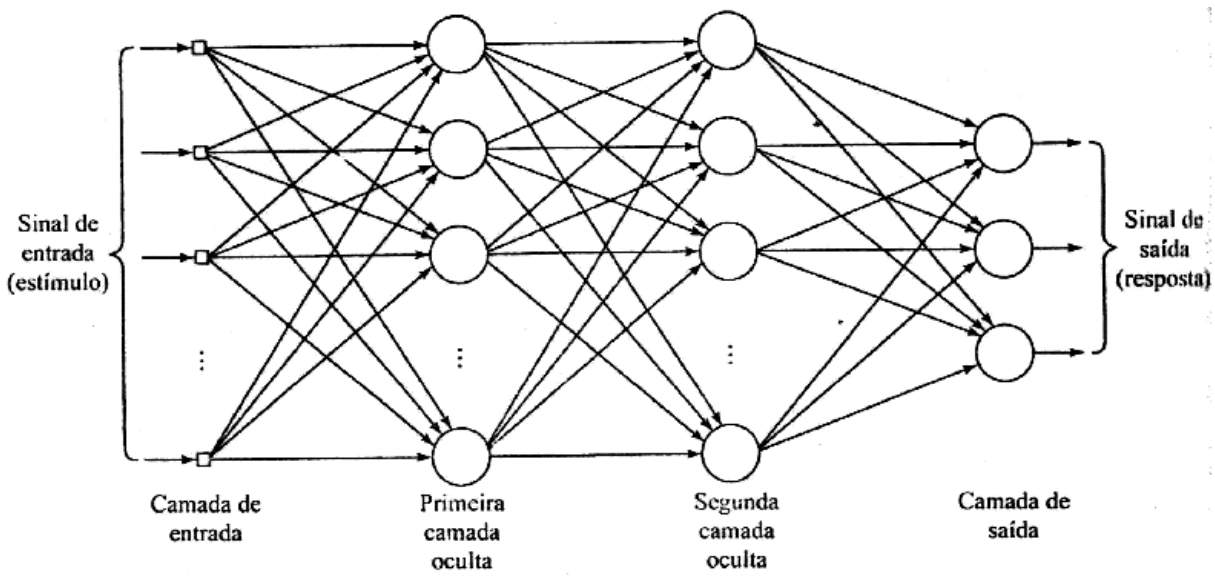
#### 2.3.3.4 Perceptron com múltiplas camadas

As redes de múltiplas camadas determinam uma classe importante para o estudo de RNAs, pois apresentam a generalização do Perceptron de uma única camada. Usualmente essas redes possuem uma série de unidades sensoriais que formam a camada de entrada, uma ou várias camadas intermediárias ocultas e uma camada de saída. Dado o sinal de entrada, ele é disseminado em todas as camadas da rede (HAYKIN, 2008).

Em redes Perceptron do tipo *feedforward*, a função das múltiplas camadas intermediárias é alterar o espaço de entrada de dados para uma apresentação acessível obtida pela camada de saída da rede. Desse modo, um determinado problema não-linear, pode ser solucionado em uma rede de mais camadas separando-o linearmente nessas camadas, gerando um novo arranjo dessa informação para a camada de saída (BRAGA, CARVALHO e LUDERMIR, 2011).

A Figura 11 mostra a arquitetura de uma rede Perceptron de múltiplas camadas, do tipo *feedforward*, contendo duas camadas ocultas e uma de saída. A imagem mostra que qualquer neurônio, em qualquer camada, é conectado a todos os outros da camada anterior/posterior.

Figura 11 - Perceptron de múltiplas camadas com duas camadas ocultas




Fonte: Haykin (2008, p. 186).

O fluxo do sinal propagado em uma rede de múltiplas camadas avança da esquerda para a direita, de camada em camada. Dois tipos de sinais são discernidos nesse modelo de rede (HAYKIN, 2008 apud PARKER, 1987):

- Sinal Funcional: consiste em um sinal de entrada (estímulo), disseminando-se de neurônio em neurônio, aparecendo no sinal de saída. É chamado de sinal funcional pois espera-se que seja realizada alguma função na camada de saída, assim como este mesmo sinal é modificado por cálculos a partir das suas entradas, a partir dos pesos associados em cada neurônio.

- Sinal de Erro: origina-se na saída de um neurônio, propagando-se para trás, camada por camada. É chamado de sinal de erro pois o cálculo estimativo do neurônio possui independentemente uma função que necessita de um erro.

De acordo com Haykin (2008), as camadas intermediárias são referenciadas como “ocultas” porque não fazem parte das camadas de entrada ou da saída da rede, a primeira camada oculta é populada pela entrada através de unidades sensoriais, a saída dessa primeira camada oculta é empregada na próxima. Os

neurônios ocultos contidos nessas camadas são estruturados para executar dois cálculos:

1. Cálculo funcional: explícito por uma função não-linear a partir do sinal de entrada, calculado com base nos pesos sinápticos ligados ao neurônio.

2. Cálculo de estimativa: cálculo também relacionado com os pesos dos neurônios, responsável por gerar o erro que é necessário para realizar a retropropagação através da rede.

Segundo Artero (2009), não existe a possibilidade de calcular o erro dos neurônios a partir das camadas ocultas de uma forma semelhante à camada de saída, pois na camada oculta ainda não é constituído o valor esperado. Assim sendo, é necessário adotar uma estratégia de propagação dos erros entregues na camada de saída de volta para as camadas ocultas para serem recalculados.

#### 2.3.3.5 O algoritmo de retropropagação (*backpropagation*)

Uma rede Perceptron de múltiplas camadas adquire conhecimento da mesma forma que o Perceptron simples. A distinção está no neurônio contido nas multicamadas, o qual é passível de ter inúmeros pesos ligados com camadas anteriores, tendo que ser ajustado de acordo com o erro oriundo dos dados de treino. O método de retropropagação, internacionalmente conhecido como o algoritmo de *backpropagation*, atribui responsabilidade nos diferentes pesos inferidos em cada neurônio, configurando um aprendizado por erro (COPPIN, 2013).

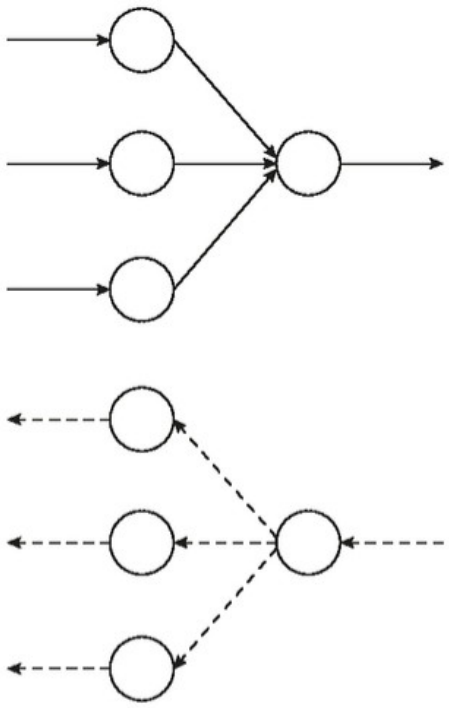
Artero (2009) explica que o algoritmo de *backpropagation* é o mais utilizado em redes com múltiplas camadas internas. O erro recebido na camada de saída é retro-propagado na rede direcionando-o para a camada de entrada.

O treinamento de uma rede que utiliza o algoritmo de *backpropagation* é do tipo supervisionado, no qual utiliza pares de entrada e saída para regular os pesos da rede por meio de um método de correção de erros. Esse treinamento acontece

em duas etapas, chamadas de *forward* e *backward*. A primeira é usada para deliberar a saída para determinado padrão de entrada, a segunda etapa usa a saída esperada entregue pela rede para ajustar os pesos de suas conexões (BRAGA, CARVALHO e LUDERMIR, 2011).

A Figura 12 ilustra o fluxo de processamento de um algoritmo *backpropagation*. Da entrada para a saída os dados seguem na etapa *forward*. Ainda na mesma rede os erros (*backward*) fazem o sentido inverso, da saída para a entrada.

Figura 12 - Representação do sinal *forward* e *backward*



Fonte: Medeiros (2018, p. 151).

Segundo Braga, Carvalho e Ludermir (2011), a fase *forward* abrange as seguintes etapas:

1. Um vetor de entrada X é inserido na camada de entrada da rede. As saídas obtidas nos neurônios da primeira camada intermediária $C_1$ são calculadas.

2. As saídas da camada oculta $C_1$ abastecem a próxima camada $C_2$. Da mesma forma as saídas da $C_2$ são calculadas, sendo o processo iterado até chegar na camada de saída $C_k$.

3. As saídas resultantes dos neurônios da camada $C_k$ são confrontadas com as saídas esperadas $Y_p$ a partir da entrada X. O erro proporcional a $Y_p$ - Y é calculado.

O objetivo da fase *forward* é alcançar o erro de saída depois da disseminação do sinal de entrada por todos os neurônios da rede. Já na fase *backward* acontecem os seguintes passos (BRAGA, CARVALHO e LUDERMIR, 2011):

1. Os erros encontrados na camada $C_k$ são usados para alterar os pesos dos neurônios desta camada, através de uma fórmula de gradiente descendente do erro.

2. Os erros da camada $C_k$ são propagados para a camada anterior $C_{k-1}$ usando os pesos das conexões dessas duas camadas, que por sua vez serão multiplicados pelos erros obtidos da camada de saída, provendo um valor de erro estimado para os neurônios da camada oculta, correspondente a uma proporção da influência entre os neurônios da camada $C_{k-1}$ com os erros da $C_k$.

3. Calculados os erros da camada $C_{k-1}$, os mesmos são empregados para modificações de seus pesos através do gradiente descendente, da mesma forma que o realizado no passo 1.

4. O procedimento é replicado até o ponto em que os pesos da camada $C_1$ sejam ajustados, finalizando desta forma a alteração dos pesos de todos os neurônios da rede.

### 2.3.3.6 Redes Neurais Convolucionais

Redes Neurais Convolucionais (*Convolutional Neural Networks*, CNNs) pertencem a um grupo de algoritmos constituídos com base em Redes Neurais Artificiais, que utilizam a convolução em no mínimo uma camada interior da rede (ARAÚJO et al, 2017). De acordo com Ferreira (2017), para ser considerada convolucional, uma rede neural deve possuir ao menos uma camada de convolução. Uma camada de convolução recebe uma entrada multidimensional, nomeado similarmente como tensor, empregando um conjunto de convoluções a partir de uma série de filtros.

Segundo ARAÚJO et al. (2017), as CNNs comprovaram ser eficientes em muitas tarefas, como reconhecimento de imagens, processamento de linguagem natural, reconhecimento de vídeos e sistemas de recomendação.

Algumas das vantagens na utilização de CNNs podem ser resumidas em (ARAÚJO et al, 2017):

- Capacidade de extração de aspectos importantes a partir de aprendizado de transformações.

- Precisam de um número menor de parâmetros de alterações em relação a redes inteiramente conectadas com uma mesma quantidade de camadas. Na medida em que uma camada interna não é conectada com todas as outras camadas seguintes, existe menos processamento para o ajuste de pesos, auxiliando no desempenho do treinamento.

Em RNAs clássicas, os neurônios são completamente conectados, capazes de receberem os pixels na entrada de uma imagem e gerar conhecimento das suas características. Mas isso requer um poder computacional grande. Considerando um exemplo de uma imagem colorida de 300x300 pixels, com 3 canais. Somente um neurônio depois da camada de entrada, conectado com todas as camadas anteriores, contém 270.000 parâmetros para serem aprendidos (300 x 300 x 3). Sabendo que um único neurônio é insuficiente para aprender todas as

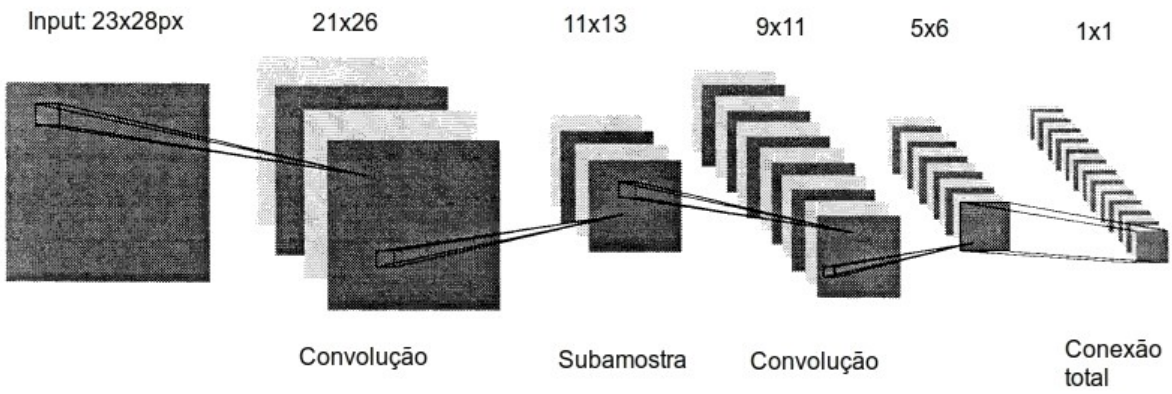
características de uma imagem, o número de parâmetros aumenta consideravelmente, tornando-se inviável treinar uma rede que tenha uma entrada com grandes dimensões. RNAs clássicas também não consideram a proximidade entre os pixels de entrada, eliminando a visão da disposição espacial de uma imagem (FERREIRA, 2017).

Uma CNN é do tipo *feedforward*, onde uma camada anterior provê recursos para a camada seguinte. Utiliza dois tipos de células em suas camadas da rede interna, simples e complexas, que são utilizadas para extração implícita de padrões visuais obtidos na entrada da rede, integradas em uma rede inteiramente conectada. Nas células complexas são extraídas características para a classificação de padrões (FERNANDES, 2013 apud LECUN et al, 1989, LECUN et al, 1998).

Através de campos receptivos locais conectados a determinadas regiões da rede, as CNNs, em sua arquitetura, compartilham pesos e características. Por meio de campos receptivos e compartilhamento de pesos, é possível a detecção de informações visuais essenciais em qualquer zona da entrada, como por exemplo bordas ou fins de linha em pixels de uma imagem (FERNANDES, 2013).

Segundo LAWRENCE et al. (1997), uma CNN refere-se a um conjunto de camadas, onde cada uma possui um ou vários planos. Imagens centradas e normalizadas entram na primeira camada. Toda unidade em um plano recebe a entrada de um vizinho menor referente a um plano anterior, conectando unidades a campos receptivos locais. Dessa forma, os pesos formados pelos campos receptivos em um plano são submetidos a serem iguais em todos os pontos restantes desse plano. A Figura 13 ilustra o modelo descrito, onde os planos são representados por dimensões de pixels do tipo RGB (*red, green, blue*).

Figura 13 - Uma representação típica de uma Rede Neural Convolucional




Fonte: Lawrence et al. (1997, p. 103).

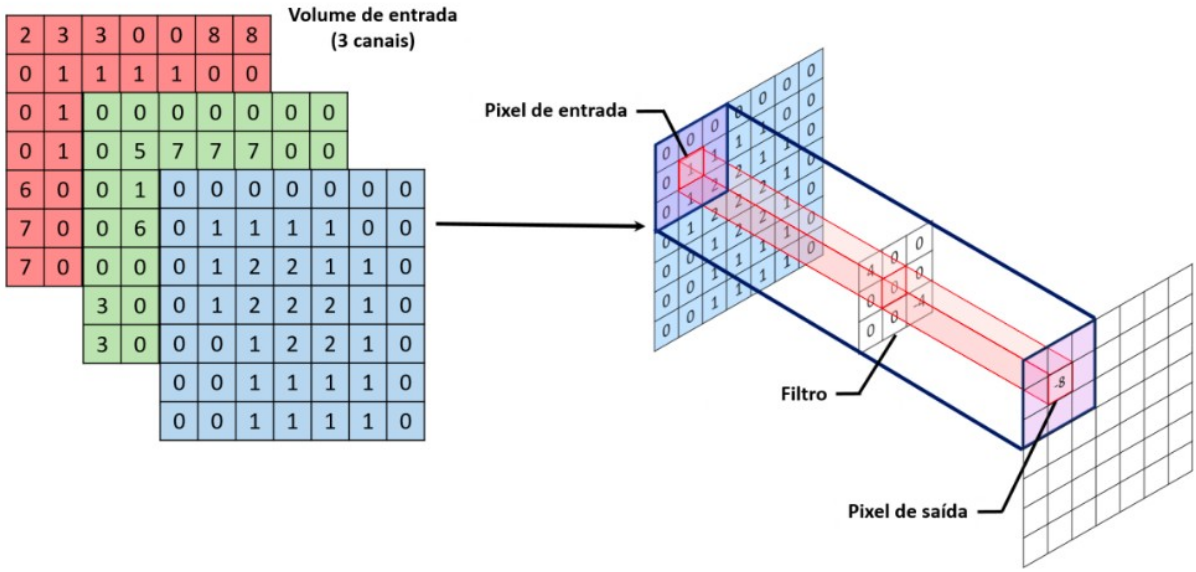
Cada plano individual pode ser visto como um mapa de recursos que possui um identificador fixo, envolvido em uma janela local escaneada a partir de camadas anteriores. Vários planos são assim utilizados em cada camada para serem apontados recursos com o intuito de observá-los. A partir do recurso detectado, a localização do mesmo deixa de ter importância. As camadas desses planos, onde são extraídos os recursos, são as camadas de convolução, ou camadas convolucionais. Cada camada é normalmente seguida por outra camada que realiza operações de média e subamostragem do local (LAWRENCE et al, 1997).

De acordo com Fernandes (2013), a partir da extração de um conjunto de características, a localização deixa de ser relevante, podendo eventualmente atravancar o reconhecimento. Ao contrário disso, o foco é situado na posição de cada característica sobre as restantes. Dessa forma, as camadas convolucionais seguintes se alternam com camadas de subamostragem, visando o crescimento da quantidade mapeada de características.

Em uma camada convolucional, cada filtro possui dimensão diminuída, estendendo-se por toda a profundidade no volume da entrada. No caso de imagens coloridas (RGB) que possuem 3 canais de entrada, o filtro da primeira camada convolucional terá 3 dimensões. Por exemplo, em uma camada de 5x5x3 (5 pixels de altura, 5 de largura e 3 de profundidade), os filtros serão ajustados durante o processo de treinamento da rede, para que os mesmos sejam ativados

automaticamente quando forem encontradas manchas ou bordas relevantes nos pixels. Essa relevância é mensurada de acordo com funções de otimização de amostras previamente classificadas (ARAÚJO et al, 2017 apud KARPATHY, 2017). A Figura 14 ilustra uma operação convolucional em uma imagem RGB.

Figura 14 - Convolução entre um filtro 3x3 com os pixels de entrada




Fonte: Araújo et al, (2017, p. 4).

Apesar de terem se mostrado eficientes e terem se tornado um novo padrão para visão computacional, CNNs precisam de uma grande quantidade de amostras com rótulos para chegar a um ponto satisfatório de classificação (ARAÚJO et al, 2017).

# 3 PROCEDIMENTOS METODOLÓGICOS

O presente trabalho teve como intuito descobrir o tipo de doença na planta da soja, a partir da análise da imagem da folha da planta, utilizando Inteligência Artificial, empreendido por meio de algoritmos construídos sob a estrutura de uma Rede Neural Artificial.

Com base na abordagem proposta, a presente pesquisa foi classificada como qualitativa. Segundo Ramos (2009), a abordagem qualitativa é utilizada em situações de estudo em que, por conta de sua complexidade, a sua quantificação torna-se difícil de ser aplicada. Não existindo uma hipótese a ser comprovada, seu uso é apropriado para o entendimento do evento estudado, caracterizado por uma conclusão indutiva sobre o assunto.

De forma conjectural, os métodos utilizados neste estudo pretenderam comprovar que doenças na soja podem ser diagnosticadas utilizando ferramentas de reconhecimento e classificação computacionais inteligentes. De acordo com a criação desse pressuposto e em conjunto com o objetivo geral, foi evidenciado o perfil exploratório descritivo deste trabalho.

Segundo Ramos (2009), a pesquisa exploratória institui um estágio inicial para uma pesquisa científica. O propósito é realizar uma aproximação com o problema estudado, a fim de fazer com que este problema seja visto de forma mais transparente. Ao adentrar na explicação de técnicas e estruturas trabalhadas, dando importância aos procedimentos aplicados com estudos de variáveis e com experimentos realizados, a pesquisa exploratória passa a ser complementada na

forma de um modelo descritivo. Para Matias-Pereira (2016), um estudo exploratório deve ser realizado considerando os esclarecimentos que comprovam seus princípios. Desse modo, durante os experimentos, a importância do estudo concentra-se no delineamento do problema, sendo que, durante a implementação do projeto a viabilidade de análise do problema formulado é avaliada.

Quanto ao tipo, classificou-se a presente pesquisa como experimental. De acordo com Ramos (2006), o tipo de pesquisa experimental é onde o evento estudado é reproduzido de forma controlada, geralmente feito por amostragem, com o propósito de encontrar os elementos que são produzidos por esse evento.

## 3.1 Etapa experimental

A organização e empreendimento deste trabalho iniciaram com a pesquisa bibliográfica sobre Inteligência Artificial. A partir da literatura foi possível constatar que o modelo de rede neural é o mais adequado para a classificação de imagens. Assim sendo, a biblioteca que melhor se adaptou ao propósito do trabalho foi utilizada. Foram aplicados testes de implementação, buscando a identificação de alguma imagem aleatória, a partir de treinamentos básicos (com poucas imagens) referenciados na documentação da mesma, objetivando a comprovação da eficácia do algoritmo contido na ferramenta.

A partir da definição de algumas amostras de folhas, com variados tipos de doenças, o ambiente de programação foi criado para a escrita do código fonte que utilizou o *framework* escolhido, o qual implementa em sua essência uma RNA.

Apoiado pela linguagem de programação JavaScript, foi construído um sistema projetado para utilização em um navegador de Internet (software web), que serviu de interface de usuário para leitura dos dados, a partir da importação de uma ou várias imagens. Os testes iniciais contiveram pequenas amostras de imagens, com o intuito de calibrar o aprendizado da rede e definir as melhores variáveis e

imagens para se trabalhar. As imagens de treino estavam nomeadas com o nome da doença, dessa forma o sistema pôde rotular a imagem com esse nome.

Com a definição de uma quantidade de imagens aceitável para treinamento na ferramenta, as imagens foram submetidas ao treinamento na RNA. Quanto ao aprendizado da rede, ele foi do tipo supervisionado. Segundo evidenciado na literatura, para uma análise eficaz da rede, quanto mais dados forem treinados, maior será a taxa de acerto na solução do problema. De acordo com isso, as amostras de imagens foram replicadas em várias outras, a partir de recortes de vários quadrados, com o auxílio de ferramentas de edição de imagens. Assim, a rede neural foi treinada com vários padrões dispersos de imagens contendo um mesmo significado, almejando uma melhor adaptação da rede a partir de exemplos distintos.

Os experimentos de classificação no software tiveram os seguintes objetivos: testar a acurácia dos resultados, testar a adição de conhecimento novo na rede neural e verificar os resultados a partir de testes com ajuste de parâmetros da rede.

## 3.2 Coleta de Dados

A partir da resolução da ferramenta de trabalho, foi consultado o portal na Internet da Embrapa, com o intuito de investigar as possíveis doenças e pragas que atingem as plantações de soja no Brasil. Após a escolha das doenças, foi iniciado a pesquisa de imagens das folhas de soja contendo essas doenças, no próprio portal da Embrapa e em repositórios de imagens da Internet, utilizando como fundamento as informações fornecidas pela empresa. As imagens pesquisadas contiveram um padrão de tamanho e foco na folha, a fim de facilitar a classificação das mesmas na fase de treinamento.

## 3.3 Análise de Dados

Na interface do software foi informado a quantidade de imagens processadas na base de treino e as taxas de perda a partir do treinamento iniciado. A partir desse ambiente, foi habilitado na tela do software o envio de uma imagem de uma folha de soja para ser classificada, onde foi realizada a identificação da doença. Nenhuma imagem dos testes de classificação foi treinada pela rede neural. Em tela foram mostradas as doenças previstas para a imagem testada, com destaque para o resultado com o valor de confiança mais alto.

Os resultados dos experimentos foram colocados em tabelas, contendo a sequência de testes. Nessa sequência de testes foi informado se o software retornou a resposta correta e os valores de confiança referentes à doença. Os resultados gerais obtidos com os experimentos foram colocados em gráficos, a partir dos quais foram obtidas visões mais amplas dos ciclos de testes para serem analisadas.

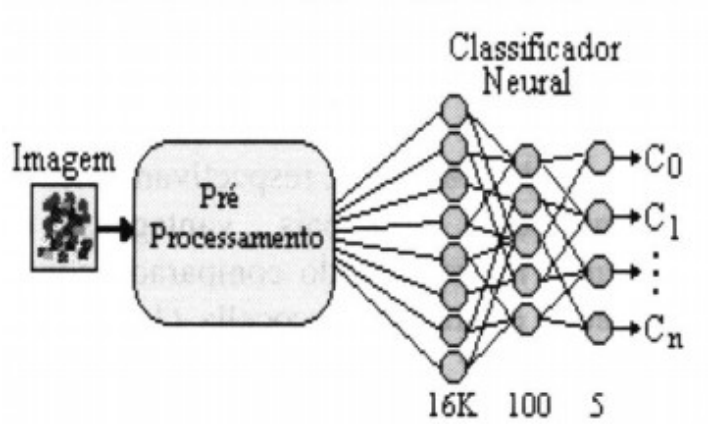
## 3.4 Trabalhos relacionados

Esta seção cita alguns trabalhos que utilizaram Aprendizado de Máquina por meio de RNAs para classificação e identificação de padrões em imagens. O intuito foi obter conhecimento para auxiliar na estruturação do trabalho proposto, uma vez que não foram encontrados trabalhos que retratavam o mesmo cenário atual apontado no problema de pesquisa.

Perelmuter et al. (1995), utilizou um algoritmo de *backpropagation* implementado na linguagem de programação C, para reconhecimento de imagens do tipo bidimensional. O objetivo do trabalho foi avaliar a eficácia das RNAs em comparação com modelos tradicionais de classificação, mais especificamente em algoritmos que implementam redes *Bayesianas* para classificação. Foi criado uma RNA com uma entrada com capacidade total de 16 KB de dados, uma camada oculta com 100 neurônios e uma camada de saída contendo uma unidade de

processamento para cada série de objetos classificada, resultando em 5 unidades de saída no total. A Figura 15 ilustra a arquitetura da rede criada.

Figura 15 - Arquitetura da RNA construída por Perelmuter et al. (1995)




Fonte: Perelmuter et al. (1995, p. 201).

Antes de importar as imagens para um treinamento utilizando um modelo de classificação inteligente, elas foram submetidas a um pré-processamento, para eliminar ruídos como alta ou baixa luminosidade e o plano de fundo das mesmas, este último sendo tratado com algoritmos de detecção de bordas em objetos. Ruídos na imagem são informações contidas dentro da matriz de pixels que não interessam, ou atrapalham o aprendizado do sistema. Perelmuter et al. (1995) propôs eliminar essa fase exaustiva de extração das características principais das imagens, por considerar uma RNA treinada apta a resolver ruídos de forma melhor que um classificador inteligente.

Segundo Perelmuter et al. (1995), os fundamentos de base para a implementação de um sistema de reconhecimento de imagens foram identificados. A classificação por meio de uma RNA identificou vantagens em relação a classificadores comuns, como a desobrigação de uma pré-seleção de imagens e de um extrator de características, já que os testes com imagens que não foram pré-processadas tiveram a mesma taxa de acerto em relação às que sofreram polimento. A possibilidade de utilização da mesma estrutura de RNA para identificação de qualquer conjunto de imagens bidimensionais também foi uma característica importante apresentada pelo autor.

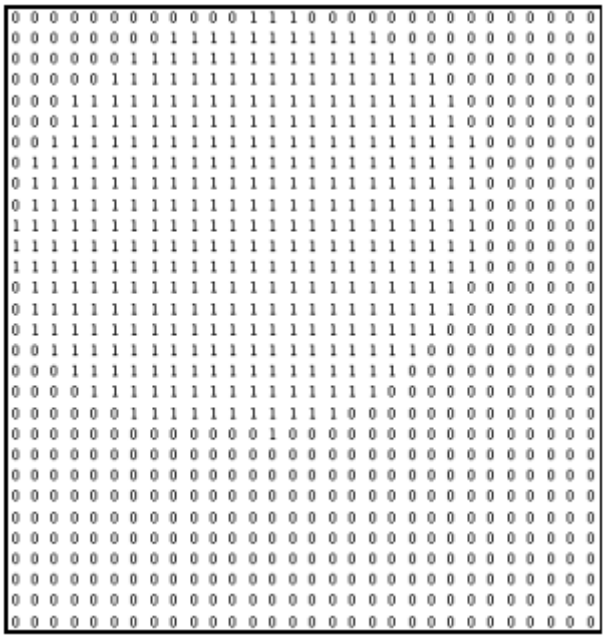
Khatchatourian e Padilha (2008) aplicaram RNAs para identificar variedades de soja a partir da forma e tamanho das sementes. Foram fotografadas imagens de uma amostra de sementes utilizando uma câmera digital, em uma superfície plana com luz fluorescente acima delas. Nessas imagens fotografadas foram aplicados filtros para as mesmas ficarem com tons cinzas, no modelo preto e branco. A partir desses filtros iniciais, foi realizada uma segmentação utilizando ferramentas computacionais de detecção de bordas em objetos de imagens, delineando as sementes e excluindo porções desnecessárias para a pesquisa.

Para a representação da informação, Khatchatourian e Padilha (2008) relacionaram um conjunto de pixels em cada semente de uma imagem, delimitando cada semente em uma matriz de tamanho igual ou inferior a 130 x 130 pixels. A Figura 16 ilustra uma imagem binária, já tratada por filtros, de uma semente de soja. A Figura 17 ilustra a representação da mesma imagem em uma matriz binária, onde o número 1 representa uma parte da semente e o número 0 somente o preenchimento do restante do retângulo.

Figura 16 - Imagem binária da semente de soja

Fonte: Khatchatourian e Padilha (2008, p. 765).

Figura 17 - Semente de soja da Figura 16 a partir de uma matriz 30 x 30

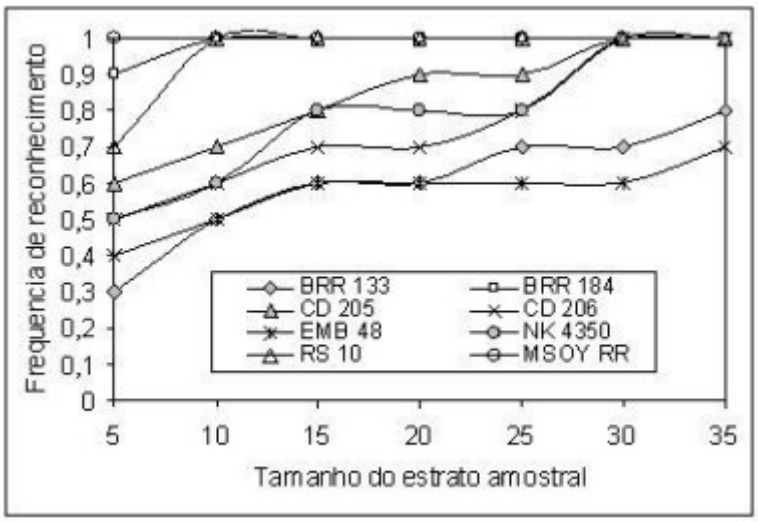


Fonte: Khatchatourian e Padilha (2008, p. 764).

Khatchatourian e Padilha (2008) utilizaram uma rede neural do tipo feedforward com o algoritmo *backpropagation*, com 16.900 componentes na camada de entrada (tamanho máximo de uma matriz 130 x 130), 3 camadas ocultas com 90, 70 e 60 neurônios respectivamente, finalizando com uma camada de saída contendo 8 neurônios. Foram treinadas 35 sementes de 8 marcas diferentes (correspondendo com o número de neurônios da camada de saída). O treinamento envolveu a propagação dos vetores (correspondentes à matriz binária) na camada de entrada, a retropropagação do erro e ajuste dos pesos, sendo necessárias 420 iterações para obter uma taxa de erro aceitável, conceituando a rede como treinada.

Para a validação da rede após o treino, Khatchatourian e Padilha (2008) utilizaram grupos de amostras de diferentes tamanhos com sementes de cada marca analisada. A Figura 18 mostra as conclusões do trabalho, onde ao ser incrementado o espaço amostral, a assertividade no reconhecimento da variedade das oito marcas de sementes analisadas aumenta consideravelmente.

Figura 18 - Influência da quantidade de amostras




Fonte: Khatchatourian e Padilha (2008, p. 768).

# 4 DESENVOLVIMENTO

Neste capítulo serão apresentadas as tecnologias utilizadas e o software desenvolvido para classificação de imagens. Testes de aplicação das tecnologias serão também descritos para avaliar a eficiência das mesmas.

A biblioteca de programação escolhida para auxiliar na obtenção de recursos de uma rede neural na construção do software foi a TensorFlow (TF). Criada pela empresa Google, atualmente é o *framework* de Inteligência Artificial mais popular utilizado pela comunidade científica e empresas privadas, possuindo uma comunidade extremamente ativa de programadores e entusiastas da tecnologia.

A escolha do TF se deu primeiramente por sua popularidade, na medida em que discussões em torno de sua utilização e aplicações crescem, se tornam acessíveis conteúdos que mostram suas habilidades na resolução de situações com uma abrangência de áreas significativa. Outro motivo pela escolha do TF, é que ele foi criado para ser compatível com várias linguagens de programação, principalmente JavaScript (JS), dando espaço para a criação de outros *frameworks* que se aplicam a finalidades específicas. Por último, um motivo fundamental que é o cerne deste trabalho, a classificação de imagens. O TF trabalha com algoritmos de redes neurais que são excelentes alternativas para o reconhecimento de padrões em imagens, pois são inúmeros os casos de uso com sucesso. O TF tem sido aplicado comercialmente em grandes empresas como Airbnb, Coca-Cola, Intel, Twitter, além da própria Google.

Nas seções seguintes, será descrito o processo de desenvolvimento do software criado para este trabalho, contendo explicações em torno do TF e suas tecnologias embarcadas.

## 4.1 TensorFlow

O TensorFlow (TF) é uma plataforma de programação de código aberto que auxilia na construção de aprendizado artificial profundo, utilizável em computadores pessoais, dispositivos móveis, navegadores web e servidores. Oferece múltiplos níveis de abstração, abrangendo uma grande variedade de solução de problemas e flexibilizando o Aprendizado de Máquina, envolvendo áreas como reconhecimento de voz, visão computacional, robótica, recuperação de informação, processamento de linguagem, extração de informações geográficas e descoberta de drogas computacionais. Conjuntamente, é versátil o suficiente para suportar experimentações e pesquisas de novos modelos em máquinas distintas (ABADI, MARTÍN et al, 2016).

Originalmente chamado de *DistBelief*, o TF foi criado pela equipe Google Brain, departamento da empresa Google responsável pela criação e pesquisas na área de IA, para explorar a otimização de redes neurais profundas. Essa equipe de desenvolvimento promove pesquisas que aprimoram o estado da arte e as aplicam em campo, com políticas de boas práticas que visam a acessibilidade de IA para qualquer pessoa. Atuam em áreas de grande potencial transformador e de impacto na sociedade, como saúde, segurança, energia, transporte, manufatura e entretenimento (Google AI, 2019).

TensorFlow, como sugere o nome, realiza computações em fluxo utilizando tensores. Um tensor é a generalização de matrizes e vetores para dimensões mais altas, ou seja, ele é um *array* multidimensional. Quando se utiliza essa biblioteca na programação, o tensor é o objeto principal do programa, representando uma computação parcialmente definida que retorna algum valor. Os programas desenvolvidos com essa ferramenta trabalham inicialmente construindo um grafo de

objetos com os tensores, detalhando a forma que cada um é calculado, com base em tensores disponíveis, executando partes deste grafo para alcançar os resultados desejados (TensorFlow, 2019b).

O TF permite treinamento e inferência em grande escala, dispondo de centenas de servidores remotos para treinamento rápido, executando modelos treinados para inferência em diversas plataformas. Utiliza um grafo de fluxo de dados para retratar a computação de um algoritmo e o estado em que o mesmo assume, inspirado por modelos de programação de alto nível. Em sistemas de fluxo de dados tradicionais, os vértices do grafo representam computação funcional e informações que não podem ser alteradas. No TF, o fluxo de dados permite que os vértices reproduzam cálculos que alteram o estado mutável do dado (ABADI, MARTÍN et al, 2015).

Treinar uma rede neural de alta precisão requer uma alta capacidade de computação, o TF dimensiona esses cálculos em um *cluster* de servidores habilitados para Unidades de Processamento Gráfico (*Graphics Processing Unit*, GPU). Com isso, as redes neurais profundas alcançaram desempenho inovador ao tratar de problemas relacionados à visão computacional, como por exemplo o reconhecimento de objetos em uma imagem. Essas tarefas aplicadas de forma eficiente são a chave para o funcionamento do software (ABADI, MARTÍN et al, 2016).

A programação com o TF é de baixo nível, ou seja, devem-se estabelecer parâmetros de otimização que estão relacionados não somente à arquitetura pretendida, bem como o conhecimento acerca dos algoritmos que são instanciados pelos tensores. Existem problemas complexos ligados a essa plataforma que precisam ser definidos de acordo com o caso de uso, como gerenciamento de memória, funções de ativação de neurônios artificiais e busca pelo melhor algoritmo otimizador. Para obter uma rede neural sofisticada para soluções que envolvem classificação de imagens e auxiliar na obtenção de resultados rápidos, o TF indica a utilização da biblioteca chamada *ml5.js* (ml5), a qual será utilizada para a implementação do software pretendido neste trabalho.

## 4.2 Biblioteca ml5

O ml5 é uma biblioteca que trabalha no topo do TensorFlow, isto é, utiliza-o nas suas camadas internas, servindo como um invólucro para ele. Essa ferramenta provê acesso a algoritmos de Aprendizado de Máquina profundo, para serem executados via navegador de Internet, utilizando apenas a linguagem JavaScript. A seguir serão apresentados alguns conceitos importantes utilizados no ml5, visando explicar a forma em que o mesmo opera.

*Datasets* são chamadas as cargas de dados que o ml5 trabalha. Cada ponto deste *Dataset* contém um atributo que é um dado específico, podendo ser do tipo texto ou numérico. Conhecidos como fatores, os dados são categóricos, ou seja, representam um conjunto claro de opções, como verdadeiro ou falso (ML5JS, 2019b).

Uma *Feature* no ml5 é um atributo relacionado ao problema, empregado no Aprendizado de Máquina. São propriedades brutas, podendo ser quantidades extraídas por uma forma de pré-processamento da informação. A extração de uma *Feature* é também chamada de engenharia de recursos, que normalmente requer um tipo de entendimento sobre o domínio. Um detalhe importante da aprendizagem em uma *Feature*, é que ela basicamente realiza a engenharia de recursos de forma automática, sem necessitar intervenção humana, o modelo aprende os recursos apropriados a partir dos dados brutos (ML5JS, 2019b).

Um *Model* é o algoritmo de Aprendizado de Máquina, caracterizado por ser um modelo matemático abstrato que procura descrever uma parte do mundo real. Para gerar um *Model*, a rede neural precisa ser treinada primeiro. Para validar esse *Model* por meio de treinamento supervisionado, o conjunto de dados é normalmente dividido em dois conjuntos aleatórios, o de treinamento e o de testes. O *Model* é feito a partir do conjunto de treino, sendo que o conjunto de testes é utilizado para fazer as previsões, contendo o respectivo rótulo e um número de predição entre 0 e 1. O que o algoritmo rotula em uma imagem está exclusivamente relacionado aos dados de treinamento (ML5JS, 2019b).

O ml5 fornece o acesso a um *Model* pré-treinado, criado a partir do treinamento em um banco de dados chamado ImageNet. Ele corresponde a um conjunto de dados de aproximadamente 15 milhões de imagens rotuladas, contendo aproximadamente 22 mil categorias. A coleta dessas imagens foi realizada na Internet e foram rotuladas por humanos a partir de ferramentas de colaboração coletivas (*crowdsourcing*) (KRIZHEVSKY SUTSKEVER e HINTON, 2012).

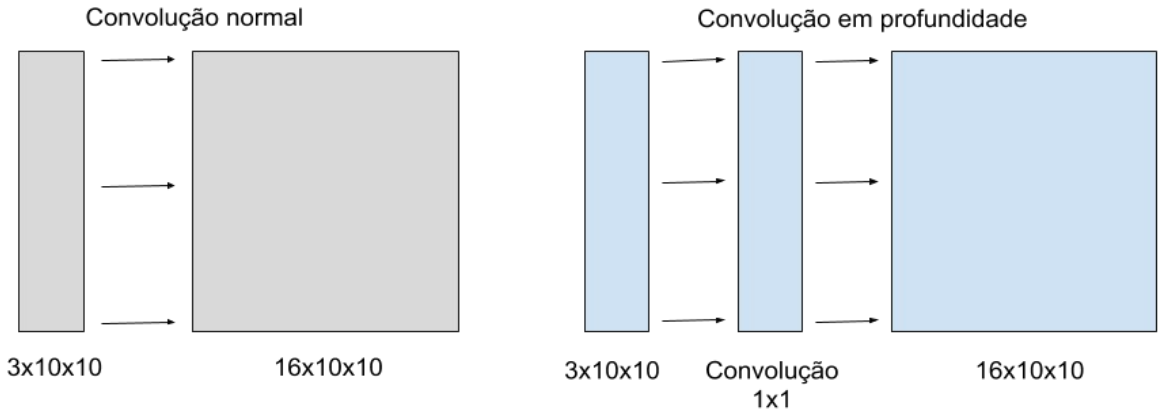
A grande maioria dos algoritmos atuais, que possuem a finalidade de reconhecer objetos, se retêm a um pequeno número de objetos regulares, como carros, pedestres e rostos. Isso se explica em razão da alta disponibilidade de imagens para categorias desse tipo. O ImageNet, de outro modo, possui uma grande quantidade de imagens contendo quase todas as classes de objetos, envolvendo também as inabituais (DENG, Jia et al, 2009). O ml5 herda do TensorFlow um algoritmo de Rede Neural Convolucional (CNN) chamado MobileNet.

### 4.2.1 MobileNet

De acordo com Howard et al. (2017), MobileNet é uma CNN (descrita na Seção 2.3.3.6) cujo modelo é apoiado em convoluções isoladas de acordo com sua profundidade, aplicando filtragens para cada canal de entrada, conhecidas também como convoluções pontuais. Esse tipo de filtro aplica uma convolução de 1x1 pixels para combinar as saídas de convolução em profundidade. Em redes convolucionais normais, os filtros combinam as entradas em um conjunto renovado com uma única etapa. A convolução isolada em profundidade separa em duas camadas, uma para a aplicação dos filtros e outra para as combinações. Isso tem o impacto de diminuir consideravelmente o tamanho do modelo computacional.

A Figura 19 exemplifica a diferença entre uma convolução de imagem padrão e uma em profundidade. No exemplo, a convolução em profundidade é realizada sobre cada canal de cor da entrada e é seguida por uma convolução 1x1 obtida através do estágio anterior, combinando-os em uma camada de saída (HOWARD et al, 2017).

Figura 19 - Convolução regular vs convolução em profundidade




Fonte: Adaptado pelo autor com base em Howard et al. (2017, p. 3).

Segundo Howard et al. (2017), a eficiência da convolução em profundidade é alta em comparação à convolução padrão, embora não combine os filtros de entrada para criar novos recursos. Para obter esses recursos, uma camada adicional é criada, da qual é calculado uma combinação da saída da convolução em profundidade através de uma convolução 1x1. O MobileNet realiza convoluções separáveis em profundidades de 3x3 pixels, utilizando entre 8 a 9 vezes menos computação, comparado a uma convolução padrão em uma mesma arquitetura de rede.

Ao todo, uma arquitetura convencional do MobileNet possui 30 camadas, fornecendo além de todas as características de uma RNA, camadas convolucionais normais, camadas com convolução em profundidade e camadas pontuais que duplicam o número de canais (HOWARD et al, 2017).

A Tabela 1 mostra o MobileNet em comparação a outros modelos de CNN que utilizam a mesma técnica de convolução em profundidade, em acurácia e quantidade de parâmetros necessários, a partir de amostras similares.

Tabela 1 - Comparação do MobileNet com modelos populares

| **Modelo (CNN)** | **Acurácia** | **Operações (milhões)** | **Parâmetros (milhões)** |
|---|---|---|---|
| MobileNet | 70,6% | 569 | 4,2 |

Continua

| Modelo (CNN) | Acurácia | Operações (milhões) | Parâmetros (milhões) |
|---|---|---|---|
| GoogleNet | 69,8% | 1550 | 6,8 |
| VGG16 | 71,5% | 15300 | 138 |

Fonte: Adaptado pelo autor com base em Howard et al. (2017, p. 6).

A Tabela 1 revela que o MobileNet possui uma precisão próxima do VGG16, sendo que é 32 vezes menor e executa 27 vezes menos operações computacionais. Em comparação ao GoogleNet, é menor, mais preciso e executa 2,5 vezes menos computações.

### 4.2.2 Testes com ml5

Para serem efetuados testes com o ml5, não é necessário rodar nenhum servidor ou outro tipo de serviço. Para iniciar com a programação da ferramenta, é necessário somente importar o script do ml5 em um arquivo HTML (Linguagem de Marcação de Hipertexto, do inglês *Hypertext Markup Language*), contendo algumas funções em JavaScript (JS) que irão chamar as funcionalidades da ferramenta.

Para iniciar uma classificação no ml5, utilizando o banco de imagens ImageNet, treinadas pela rede MobileNet, são necessárias 3 etapas:

1. Inicializar o MobileNet: Conecta no servidor remoto onde está a CNN do MobileNet através de uma nova instância do método classificador de imagens presente no ml5. É necessária uma conexão banda larga com a Internet nessa etapa.

2. Classificação: Assim que o MobileNet é carregado, ele já está pronto para classificar uma imagem. As imagens habilitadas para classificação podem ser de vários tipos e extensões. Nos testes iniciais deste trabalho, as imagens classificadas estavam dentro de tags HTML na própria página. Mas elas podem também estar no sistema de arquivos do computador (necessário um servidor HTTP) ou codificadas em *Base64*.

3. Obtenção dos resultados: O método classificador do ml5 retorna uma matriz de dados (chamado de *array* em JS), contendo as principais classes encontradas. Cada classe é um objeto que possui o rótulo da imagem encontrado e a confiança (número longo entre 0 e 1) sobre o mesmo.

O ml5 foi criado com uma propriedade importante em programas feitos em JS, a utilização de funções de retorno de chamadas, denominadas de *callbacks*. Com isso, cada função invocada aguarda a anterior para ser iniciada. Essa característica é presente naturalmente em outras linguagens, mas em navegadores de Internet o JS executa suas funções de modo não concorrente, isto é, ele não espera uma computação terminar para iniciar outra (modo assíncrono). Os *callbacks* existem para contornar isso, sendo essencial para o correto funcionamento do ml5, que depende sempre da finalização do processo anterior para executar o próximo.

Além de ter que estabelecer conexão com o servidor remoto (depende exclusivamente da velocidade de conexão com a Internet), existem funções que exigem concorrência com o processador local do computador que podem demorar para serem concluídas, como o treinamento de imagens novas, que será visto adiante.

A Figura 20 mostra o código fonte dos testes iniciais com o ml5, com comentários no código para especificar o que cada instrução realiza.

Figura 20 - Teste inicial de classificação de uma imagem com ml5

```
// A imagem estática para ser classificada
const image = document.getElementById('image');
// O rótulo do resultado em uma tag HTML
const result = document.getElementById('result');
// A probabilidade do resultado em uma tag HTML
const probability = document.getElementById('probability');

// Inicializa o método classificador de imagens do MobileNet
ml5.imageClassifier('MobileNet')
  // Executa a função de classificação instanciada do classificador
  .then(classifier => classifier.classify(image))
  // Obtém os resultados da classificação em um array
  .then(results => {
    result.innerText = results[0].label;
    probability.innerText = results[0].confidence.toFixed(4);
  });
```

Fonte: Do autor (2019).

Os resultados da classificação são trazidos em um *array* JS, ordenados em ordem decrescente segundo a confiança rotulada. No código fonte de exemplo foi exposto somente o resultado principal, ou seja, o de maior confiança. Esse resultado foi mostrado na página HTML criada.

O teste inicial classificou apenas uma imagem a partir do conhecimento prévio da rede MobileNet. Essa imagem foi colocada estaticamente na página HTML, sendo que para realizar testes com outras imagens foi necessário alterar o link ou baixar uma nova imagem e anexá-la ao projeto.

As Figuras 21, 22 e 23 mostram os resultados de testes feitos com diferentes classes de imagens. A Figura 21 mostra a classificação com uma imagem de um cachorro da raça labrador, o ml5 acertou a resposta com confiança de aproximadamente 77%.

Figura 21 - Teste unitário do ml5 com imagem de um cão da raça labrador


Fonte: Do autor (2019).

É importante observar que, o ml5 não só caracterizou a imagem como sendo um cachorro, mas também acertou a raça dele. Isso é decorrente do extenso treinamento existente no MobileNet a partir de milhões de imagens com milhares de classes obtidas via banco de imagens do ImageNet.

Na Figura 22, foi setado a imagem de um relógio de parede analógico. O ml5 apresentou o rótulo de um relógio analógico com confiança de aproximadamente 81%.

Figura 22 - Teste unitário do ml5 com imagem de um relógio de parede


Fonte: Do autor (2019).

A Figura 23 apresenta um resultado muito próximo de 100% de acerto ao classificar uma imagem de uma locomotiva a vapor.

Figura 23 - Teste unitário do ml5 com imagem de uma locomotiva a vapor


Fonte: Do autor (2019).

Classes comuns de imagens, como as apresentadas nas Figuras 21, 22 e 23, normalmente apresentam resultados corretos com confiança acima de 70%. Isso se dá pelo fato do MobileNet conter milhares de classes treinadas, sendo que a probabilidade de haver uma imagem buscada de forma aleatória de objetos comuns, animais ou meios de transporte, é bastante alta. Por isso um dos testes feitos com o ml5 incluiu a classificação de imagens incomuns, que possivelmente não foram treinadas pelo MobileNet, com a finalidade de observar os resultados, analisando a proximidade da resposta correta com as predições.

A Figura 24 refere-se a um teste de classificação de um objeto singular: um vaporizador de roupas. Para esse teste o código fonte foi modificado para trazer os 3 principais resultados em forma de *logs* no navegador, com o propósito de verificar todos os rótulos obtidos pelo ml5. Na figura foi ilustrado a maior confiança juntamente com os outros resultados.

Figura 24 - Teste unitário do ml5 com imagem de um vaporizador de roupas


Fonte: Do autor (2019).

Com 53% de confiança, o ml5 retornou que o objeto é um jarro de água (*water jug*). Com similares 13% de confiança (aproximados), foram identificados respectivamente um abridor de latas (*can opener*) e uma máquina de café expresso (*espresso maker*). Como esperado, o ml5 não acertou a predição do objeto da imagem, mas o importante desse teste foi verificar que os objetos mais próximos encontrados possuem características semelhantes.

Na Figura 25, foi aplicado o teste de classificação em um animal raro, um mamífero chamado ocapi.

Figura 25 - Teste unitário do ml5 com imagem de um ocapi


Fonte: Do autor (2019).

Analisando os resultados, com 55% de confiança, o ml5 trouxe um alazão (cavalo, *sorrel*). Com aproximadamente 12%, um cachorro da raça *whippet*, seguido de uma zebra, com 7% de confiança. Na Figura 25 foram ilustrados também a imagem dos 3 animais que foram retornados, para fins de comparação. Novamente o importante nesse teste foi observar que as 3 principais predições possuem características semelhantes com a imagem classificada. Esse teste foi essencial para perceber, a grosso modo, como o algoritmo de rede neural do ImageNet trabalha, visualizando a imagem por completo, vasculhando pixel a pixel, buscando pontos de similaridade específicos entre a imagem de entrada e o que já foi aprendido.

Os testes das Figuras 24 e 25 também foram importantes para perceber que o ml5 atende à demanda deste trabalho. Dado que para classificar doenças na folha da soja, é necessário analisar a imagem de entrada em sua totalidade, pois as doenças também possuem características individuais.

### 4.2.3 Aprendizado supervisionado no ml5

A classificação realizada pelo ml5 nos testes anteriores provém de uma base já treinada. Um dos objetivos deste trabalho foi treinar a rede neural a partir de um

aprendizado supervisionado. O ml5 viabiliza esse treinamento, utilizando o modelo pré-treinado oferecido e agregando o método conhecido como *featureExtractor*.

O *featureExtractor* permite que o modelo da rede neural do MobileNet seja treinado, retreinado, ou reutilizado para uma nova tarefa customizada, adicionando novos dados. Essa tarefa é também chamada de transferência de aprendizagem.

A Figura 26 mostra o método *featureExtractor*, com comentários relacionados aos seus parâmetros iniciais.

Figura 26 - Método *featureExtractor* do ml5

```
/**
 * @model O modelo de extração dos recursos aprendidos
 * @options Um objeto com opções customizadas para serem aplicadas na rede neural
 * @callback_ Função de retorno para executar quando o modelo estiver pronto
 */
ml5.featureExtractor("MobileNet", { numClasses: 2 }, function (loaded) {
    // Model carregado
});
```

Fonte: Do autor (2019).

Dentre as opções do método de transferência de aprendizagem, estão incluídas: versão do MobileNet a ser utilizada, número de camadas ocultas (100 é o padrão do ml5), número de *epochs*[4] (20 é o padrão do ml5) e número de classes para serem treinadas.

Ao final de cada ciclo de treinamento (*epoch*), a função de treino retorna um valor de perda. Diversos algoritmos de Aprendizado de Máquina possuem uma função de perda, também chamada de função de custo (aprofundada na Seção 2.3.3). Essa função é responsável por mensurar a performance do modelo de RNA para com o conjunto de treino. Quanto menor o valor retornado pela função de perda (valor ou taxa de perda), melhor é o desempenho do modelo (ML5JS, 2019a).

O ml5 permite o treinamento e classificação de imagens a partir de um vídeo. Para testar a funcionalidade de treinamento, foi criado um software de teste, habilitando a característica de capturar as imagens a partir da *webcam* do

4 Uma *epoch* refere-se a um ciclo completo de treino em uma rede neural, onde um determinado dado é percorrido para frente e para trás (*backpropagation*) uma única vez na rede (MIKOLOV et al, 2010).

computador. Esse software consistiu em adicionar e treinar duas classes de imagens, a *webcam* foi ativada para capturar 20 imagens de cada uma das duas classes (uma caneca e um copo). A partir da captura das imagens, elas foram treinadas com o método *featureExtractor*, ilustrado na Figura 26. A classificação no programa foi feita por meio da aproximação do objeto com a *webcam*.

A Figura 27 mostra o teste aplicado com duas classes de imagens, em dois momentos distintos. No primeiro momento, foi aproximado a imagem da caneca à *webcam*, no segundo momento a imagem de um copo. Em ambos os casos, houveram acertos nas predições dos objetos com confiança próxima de 100%.

Figura 27 - Classificação de duas classes de imagens a partir de uma *webcam*


Fonte: Do autor (2019).

## 4.3 Imagens de doenças na soja

A procura por imagens de doenças na folha na soja ocorreu por meio de pesquisas na Internet. Apoiado pelo manual de identificação de doenças na soja da Embrapa (Empresa Brasileira de Pesquisa Agropecuária), foram identificadas 8 doenças comuns em solo brasileiro. São doenças causadas por fungos, bactérias e viroses, provenientes de clima úmido e quente (HENNING et al, 2014).

A Figura 28 ilustra um exemplo de cada uma das 8 classes de doenças que foram utilizadas no trabalho de treinamento supervisionado da RNA. Da esquerda para a direita na imagem, estão as doenças: ferrugem, míldio, oídio, olho de rã, mancha parda, mancha alvo, crestamento bacteriano e virose.

Figura 28 - Doenças na soja

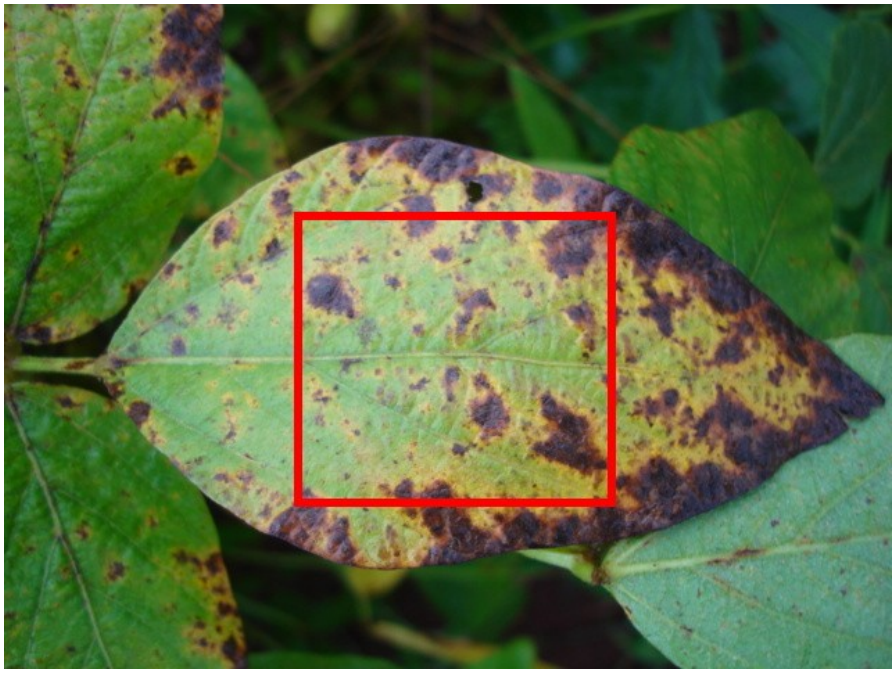
Fonte: Adaptado pelo autor com base em HENNING et al. (2014, p. 13-61).

Nos testes iniciais realizados com a ferramenta ml5, foram utilizadas imagens de objetos e animais. Devido suas características particulares, essas imagens podem ser classificadas com uma acurácia extremamente estável pelo software. Como visto anteriormente, o problema deste trabalho não possui o objetivo de reconhecer um objeto em uma imagem, mas sim uma doença, caracterizada por uma irregularidade ímpar na folha da soja. Essas irregularidades estão presentes (como visto na Figura 28), em forma de manchas intrínsecas na folha. Em outros termos, já sabe-se que o objeto se trata de uma folha de soja, o aspecto que torna o estudo diferente dos testes iniciais, é o foco nas manchas dessas folhas.

Para assegurar o foco nas manchas das folhas de soja, bem como ter a possibilidade de expansão do tamanho da base de imagens, foram capturados recortes de 120x120 pixels das imagens das doenças. Isso preveniu que imagens com o segundo plano diferentes pudessem comprometer a obtenção das características exclusivas de cada doença, pois como constatado nas Figuras 24 e 25, a CNN do MobileNet examina a imagem por completo, salvando o conhecimento de pontos importantes dela. Por exemplo: se em uma determinada imagem de uma doença A, houver um fundo com um céu azul marcante, similar ao da doença B, a rede poderia salvar essa característica desnecessária e determinar que existem similaridades entre as duas imagens durante a fase de treinamento, prejudicando o prognóstico posterior.

Cada imagem foi salva com o nome de sua respectiva doença, por exemplo: *ferrugem-1.png*, *ferrugem-2.png,* isso permitiu a identificação das mesmas na fase de treinamento da rede neural (Seção 4.3). A Figura 29 transmite a concepção de como foram extraídos os pontos de interesse nas imagens em quadrados de 120x120 pixels.

Figura 29 - Extração de pontos de interesse para treinamento



Fonte: Do autor (2019).

Devido a inexistência de uma quantidade satisfatória de imagens na Internet, para cada imagem buscada, foram geradas aproximadamente 8 novas imagens com recortes feitos a partir da original. Todas as imagens novas criadas foram rotacionadas em 45° por 3 vezes, sendo salvo em uma nova imagem a cada rotação.

## 4.4 Implementação do Software

O TensorFlow é implementado pelo ml5 em linguagem JavaScript (JS). Com isso o software criado para usar o ml5 deve ser preferencialmente desenvolvido

utilizando tecnologias web modernas, para ser executado em navegadores em suas últimas versões.

Para o desenvolvimento do software, foi utilizado a biblioteca *Meteor.js*, que cria um ambiente de desenvolvimento para utilização da linguagem de servidor *Node.js*. O Node executa código JS no lado do servidor, orientado a eventos ocorridos de forma assíncrona. A execução do código no servidor foi necessária na implementação do software, pois ela permitiu que fossem lidas informações de arquivos da máquina em que está sendo executado o Node. Por padrões de segurança, é inevitavelmente bloqueado pelo navegador de Internet a leitura de arquivos locais utilizando somente código JS sem estar hospedado em um servidor.

Com o ambiente estipulado, é o servidor Node que gerencia o uso de CPU e memória RAM do computador, tirando a carga de trabalho do navegador local. A interface do programa foi criada considerando todos os passos em que o servidor e o usuário necessitam realizar. Os requisitos de software fundamentais para a realização deste trabalho foram:

- Carregar o modelo de RNA do MobileNet.
- Adicionar as imagens para treinamento.
- Treinar a RNA com as imagens adicionadas.
- Carregar uma imagem para teste.
- Recortar a imagem sob o foco da área de interesse selecionada pelo usuário.
- Classificar a imagem, mostrando os principais resultados com seu respectivo percentual de confiança sobre o mesmo.

O programa inicia conectando-se ao MobileNet e carregando o modelo de rede com os parâmetros iniciais. O parâmetro alterado no software foi a quantidade de classes, que muda ao longo dos testes e análises (Capítulo 5).

Após o carregamento inicial do MobileNet, o sistema adiciona automaticamente as imagens das doenças na soja. Em tela são mostradas todas as imagens da base, com a quantidade total delas. Essas imagens estão presentes em uma pasta do servidor local, já preparadas com tamanhos iguais de 120x120 pixels. As imagens foram salvas com o nome da doença a que referem-se, esse mesmo nome foi utilizado para extrair o rótulo da imagem adicionada. Por exemplo: nas imagens nomeadas como *olho_de_ra-23.png*, *mancha_parda-39.png*, *ferrugem-15.png*, foram extraídos respectivamente os rótulos: *olho_de_ra*, *mancha_parda* e *ferrugem*.

Logo após a adição de todas as imagens, o usuário fica habilitado para clicar no botão de treinamento, criado especialmente para dar início a esta etapa. Utilizando o método *featureExtractor* do ml5 (descrito na Seção 4.2.3), inicia-se o treinamento das imagens adicionadas. O valor de perda (Seção 4.2.3) é impresso na tela a cada final de ciclo de treinamento (*epoch*). Essa etapa refere-se ao aprendizado supervisionado presente no trabalho, onde foram adicionados na rede neural os dados rotulados de forma manual a partir do conhecimento prévio (manual de identificação de doenças na soja da EMBRAPA) que obteve-se das imagens.

Figura 30 - Imagens treinadas


Fonte: Do autor (2019).

Concluída a fase de treinamento, fica disponível ao usuário a opção de selecionar alguma imagem da máquina local, para ser carregada no navegador. Assim que a imagem é carregada, o software mostra a imagem na tela e executa a função para recorte de um ponto da mesma. Essa função de recorte permite que o usuário posicione um quadrado de 120x120 pixels em um ponto específico da imagem, mirando em um ponto de interesse a ser classificado. Assim que o ponto específico da imagem é escolhido, fica habilitado para o usuário recortar e classificar a imagem, a partir de um botão na tela (Figura 31).

Figura 31 - Recorte da imagem focando a área de interesse


Fonte: Do autor (2019).

O software chama a função de classificação do ml5, trazendo o vetor de objetos, contendo o rótulo encontrado e a confiança sobre ele. Os resultados são mostrados em tela, mediante uma tabela, contendo duas colunas. A primeira com o

nome da doença e a segunda com o percentual de confiança previsto para a mesma. A tabela fica ordenada pelo percentual de confiança em ordem decrescente, com destaque em fonte maior para a maior predição (Figura 32).

Figura 32 - Classificação da doença presente na imagem da folha

| Doença | Confiança |
|---|---|
| ferrugem | 99.98741149902344 % |
| mancha_parda | 0.012588337995111942 % |

Fonte: Do autor (2019).

Os Apêndices A, B, C e D ilustram trechos de código referente ao software implementado, contendo as funções para instanciação da rede neural MobileNet, adição das imagens, treinamento e classificação.

# 5 ANÁLISES E RESULTADOS

Este capítulo tem o propósito de apresentar os resultados de classificações realizadas em imagens de doenças na soja, utilizando o software implementado (descrito na Seção 4.4).

Para cada conjunto de testes, foi feita uma análise dos resultados, tendo como objetivo avaliar o software criado e detectar possíveis alterações em sua estrutura, buscando melhorias. A acurácia foi o ponto central onde concentraram-se os testes, tendo em vista a busca pela melhor confiança possível.

Os experimentos de classificação foram aplicados com imagens aleatórias, extraídas da Internet, correspondentes às oito classes de doenças pesquisadas (Seção 4.3). Para cada tipo de doença, foram buscadas 5 imagens diferentes, para ser testada a classificação retornada pelo software. O primeiro experimento envolveu o treinamento com todas as 8 classes de doenças na soja pesquisadas. A base continha 606 imagens, com aproximadamente 75 imagens para cada doença.

Os resultados foram adicionados em uma tabela, contendo os registros de todos os testes. A Tabela 2 apresenta os resultados do experimento com 8 classes de doenças. As colunas com o número do teste representam cada uma das 5 imagens em que foi testada a classificação. Essas imagens são correspondentes às doenças descritas na primeira coluna. Para cada teste foi verificado se o software retornou a doença esperada, marcado na tabela com "A" (acerto) ou "E" (erro). O erro significa que o resultado principal (aquele com maior percentual de confiança), estava errado, de acordo com a doença da imagem testada. Como o ml5 retorna a

confiança para todas as classes configuradas da rede, foi mostrado também o percentual de confiança em que a doença correta foi entregue.

A coluna referente à taxa de acerto faz uma média aritmética simples a partir da quantidade de acertos obtidos. Para cada acerto, foi também guardada a informação do percentual de confiança obtido na classificação da doença, essa informação foi usada na coluna de média de confiança dos acertos, onde foi aplicada uma média aritmética simples somente nos testes em que os resultados foram corretos.

A última coluna, média de confiança geral, corresponde à média aritmética simples de todos os percentuais de confiança encontrados, correspondentes à imagem que seria a correta.

Tabela 2 - Testes de classificação com 8 classes de imagens

| **Doença** | **Acerto/Confiança** | | | | | **Taxa de acerto** | **Média confiança dos acertos** | **Média confiança geral** |
|---|---|---|---|---|---|---|---|---|
| | **Teste 1** | **Teste 2** | **Teste 3** | **Teste 4** | **Teste 5** | | | |
| Ferrugem | A/90,28 | A/85,86 | A/85,88 | A/97,03 | A/87,82 | 100% | 89,37% | 89,37% |
| Crestamento bacteriano | E/13,11 | E/38,53 | A/71,83 | E/0,98 | E/0,65 | 20% | 71,83% | 25,02% |
| Mancha alvo | E/2,20 | E/0 | E/2,09 | E/0,04 | E/8,43 | 0% | - | 2,55% |
| Mancha parda | E/2,18 | E/0,08 | A/39,17 | E/2,89 | E/4,64 | 20% | 39,17% | 9,79% |
| Míldio | E/30,02 | A/84 | A/86,86 | A/61,15 | A/98,37 | 80% | 82,59% | 72,08% |
| Oídio | A/91,02 | A/98,09 | E/11,44 | A/95,37 | E/26,92 | 60% | 94,83% | 64,57% |
| Olho de rã | A/93,40 | A/97,22 | A/66,78 | E/0,51 | A/60,11 | 80% | 79,38% | 63,60% |
| Virose | E/10,80 | E/2,17 | E/0,79 | E/5,23 | A/61,68 | 20% | 61,68% | 16,13% |
| **Média geral** | | | | | | **47,50%** | **64,86%** | **42,89%** |

Fonte: Do autor (2019).

Os resultados exibidos na Tabela 2 mostram que, a taxa de acerto na classificação da doença correta foi de 47,50%. A média de confiança desses acertos foi de 64,86% e a média da confiança na predição da doença que seria a correta foi de apenas 42,89%.

Para verificar o motivo que deixou os resultados de tal maneira abaixo do esperado, foram analisados tanto os testes que baixaram consideravelmente o valor de confiança, quanto os que elevaram a média geral com valores corretos.

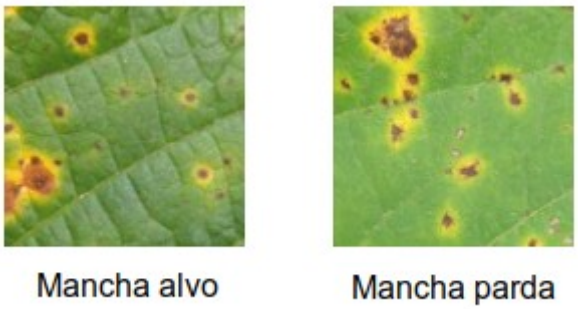
A primeira doença (ferrugem), foi a única que obteve sucesso em todos os testes. Além de ter acertado a classificação das cinco imagens testadas, o software retornou uma média geral de confiança relativamente alta. Isso aconteceu porque as imagens da ferrugem na soja possuem características altamente representativas, de modo que nenhuma das outras doenças presentes na base possuem alguma similaridade com as manchas na folha que ela possui. Essa propriedade de acerto com alta convicção, em imagens singulares, esteve presente nos testes iniciais da ferramenta ml5, como o exemplo mostrado na Figura 23.

As doenças que declinaram a média geral dos testes da Tabela 2, foram: crestamento bacteriano, mancha alvo, mancha parda, oídio e virose. Elas correspondem a mais de 60% da base de imagens treinada com oito classes. Sendo que a taxa média de predição correta dessas doenças ficou em apenas 24%.

Na doença mancha alvo, o software não acertou nenhum dos cinco testes e também resultou em uma confiança geral de apenas 2,55%. Na doença mancha parda houve somente um acerto, mas obteve uma certeza nele de apenas 39,17%, ficando com 9,79% na média geral.

Explorando especificamente o teste número 2 da doença mancha alvo, que obteve 0% (0,0056%) de confiança na doença que seria a correta, notou-se que houve uma certeza muito alta que a doença correta seria a mancha parda (96,90%). Para avaliar esse resultado, a Figura 33 mostra a mesma imagem utilizada no teste da Tabela 2, referente à doença mancha alvo (trecho recortado em um quadrado de 120x120 pixels), em comparação com uma imagem da doença mancha parda, retirada da base treinada, retornada em primeiro lugar pelo software.

Figura 33 - Doenças mancha alvo e mancha parda




Fonte: Do autor (2019).

Analisando a Figura 33, é notória a similaridade nos padrões das manchas entre as duas imagens. Uma pessoa sem o devido conhecimento poderia facilmente confundir os dois tipos de manchas e associá-las incorretamente.

Na base das 606 imagens treinadas, haviam 76 imagens relativas à mancha alvo. Verificou-se que nenhuma dessas imagens apresentava características similares às imagens testadas para essa doença. Por outro lado, as imagens dos testes continham muito mais similaridade para com as imagens de mancha parda treinadas. Tendo em vista essa falta de treino para mais amostras de manchas alvos, não pode-se afirmar que o software falhou na entrega dos resultados, pois houve uma entrega de resposta caracterizada pela homogeneidade entre as duas doenças. Essa visão para padrões pertinentes de uma imagem foi constatada também nos testes iniciais do ml5, nas Figuras 24 e 25.

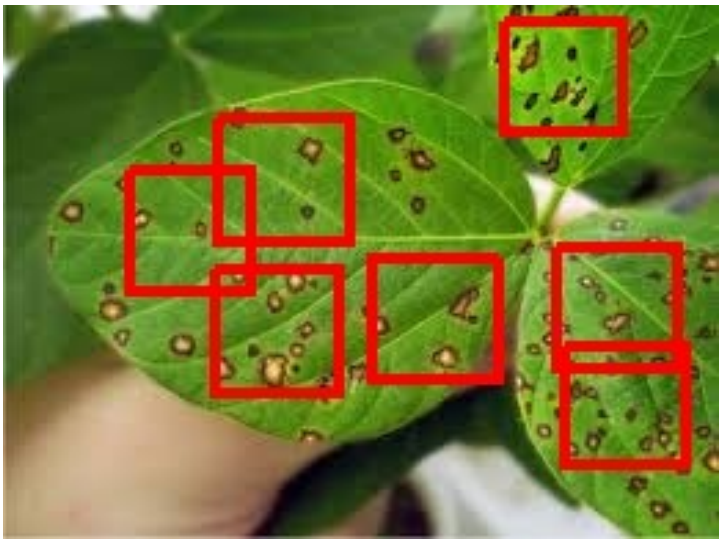
Para melhorar e testar a evolução das respostas do software criado, baseando-se nas inconsistências encontradas na Tabela 2, foram elaboradas novas sequências de testes, contendo somente duas classes de doenças. O intuito dos próximos testes foi confirmar se o tamanho da base de treino é crucial para a acurácia da ferramenta. As imagens de folha da soja escolhidas para os próximos testes foram as que continham as doenças mancha parda e mancha alvo, que por sua vez foram as que tiveram os resultados mais baixos mostrados na Tabela 2.

A base de imagens de treinamento do teste seguinte, com duas classes de imagens continha inicialmente 100 imagens, metade para a doença mancha parda e

outra metade para a mancha alvo. O objetivo foi iniciar com uma base pequena e ir incrementando com novas imagens de treino, até duplicar a quantidade para cada classe.

Para cada sequência nova de testes, foram adicionadas novas imagens na base, seguindo o mesmo conceito já aplicado, a partir da extração de quadrados exatos de 120x120 pixels de uma imagem original. Mas como o objetivo foi incrementar a quantidade visando duplicar a base, a busca por novas imagens na Internet não seria suficiente. Em virtude disso foram extraídas diversas outras imagens de uma original, como ilustrado na Figura 34.

Figura 34 - Extração de vários pontos de interesse em uma imagem original



Fonte: Do autor (2019).

A Tabela 3 mostra os resultados dos testes aplicados com 5 imagens para cada doença, progredindo a quantidade de imagens treinadas, até chegar a uma base de 100 imagens para mancha parda e 100 para mancha alvo.

Tabela 3 - Testes de classificação incremental com 2 classes

| **100 imagens treinadas** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **Acerto/Confiança** | | | | | | | |
| **Doença** | **Teste 1** | **Teste 2** | **Teste 3** | **Teste 4** | **Teste 5** | **Taxa de acerto** | **Média confiança dos acertos** | **Média confiança geral** |

Continua

| 100 imagens treinadas | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Acerto/Confiança | | | | | Taxa de acerto | Média confiança dos acertos | Média confiança geral |
| Doença | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | | | |
| Mancha alvo | E/33,15 | E/5,32 | E/2,21 | E/8,02 | A/76,09 | 20% | 76,09% | 24,95% |
| Mancha parda | E/42,53 | E/45,57 | A/93,91 | E/20,21 | E/38,58 | 20% | 93,91% | 48,16% |
| **Média geral** | | | | | | **20%** | **85%** | **36,56%** |
| **120 imagens treinadas** | | | | | | | | |
| | Acerto/Confiança | | | | | Taxa de acerto | Média confiança dos acertos | Média confiança geral |
| Doença | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | | | |
| Mancha alvo | A/54,32 | E/8,11 | E/5,40 | E/4,97 | A/63,61 | 40% | 58,97% | 27,28% |
| Mancha parda | E/33,36 | E/12,31 | A/98,44 | E/24,69 | E/24,78 | 20% | 98,44% | 38,72% |
| **Média geral** | | | | | | **30%** | **78,71%** | **33%** |
| **140 imagens treinadas** | | | | | | | | |
| | Acerto/Confiança | | | | | Taxa de acerto | Média confiança dos acertos | Média confiança geral |
| Doença | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | | | |
| Mancha alvo | E/49,23 | E/20,70 | E/21,86 | E/2,98 | A/50,69 | 20% | 50,69% | 29,09% |
| Mancha parda | A/53,27 | E/15,93 | A/92,13 | E/29,03 | E/39,36 | 40% | 72,70% | 45,94% |
| **Média geral** | | | | | | **30%** | **61,70%** | **37,52%** |
| **160 imagens treinadas** | | | | | | | | |
| | Acerto/Confiança | | | | | Taxa de acerto | Média confiança dos acertos | Média confiança geral |
| Doença | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | | | |
| Mancha alvo | A/60,20 | E/45,92 | E/41,71 | E/38,71 | A/64,89 | 40% | 62,55% | 50,29% |
| Mancha parda | A/59,46 | E/44,89 | A/95,98 | E/40,19 | E/48,03 | 40% | 77,72% | 57,71% |
| **Média geral** | | | | | | **40%** | **70,14%** | **54%** |
| **180 imagens treinadas** | | | | | | | | |

Continua

| 180 imagens treinadas | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Acerto/Confiança | | | | | | | |
| Doença | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Taxa de acerto | Média confiança dos acertos | Média confiança geral |
| Mancha alvo | A/51,88 | A/70,10 | E/48,42 | E/31,99 | E/45,14 | 40% | 60,99% | 49,51% |
| Mancha parda | A/75,22 | E/43,54 | A/99,70 | E/30,03 | A/52,79 | 60% | 75,90% | 60,26 |
| **Média geral** | | | | | | **50%** | **68,45%** | **54,89%** |
| **200 imagens treinadas** | | | | | | | | |
| | Acerto/Confiança | | | | | | | |
| Doença | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Taxa de acerto | Média confiança dos acertos | Média confiança geral |
| Mancha alvo | A/64,17 | A/91,64 | E/45,63 | A/50,59 | A/78,26 | 80% | 71,17% | 66,06% |
| Mancha parda | A/74,41 | E/40,23 | A/98,84 | E/48,01 | A/56,48 | 60% | 76,58% | 63,59% |
| **Média geral** | | | | | | **70%** | **73,88%** | **64,83%** |

Fonte: Do autor (2019).

No primeiro teste, com a base de 100 imagens, a melhoria nos resultados de classificação em comparação com os da Tabela 2, atribuiu-se pelo fato de existirem somente duas classes para o programa classificar, ou seja, a concorrência para a predição junto com outras classes foi bem menor. Outros aspectos importantes dos testes incrementais com 2 classes são descritos a seguir.

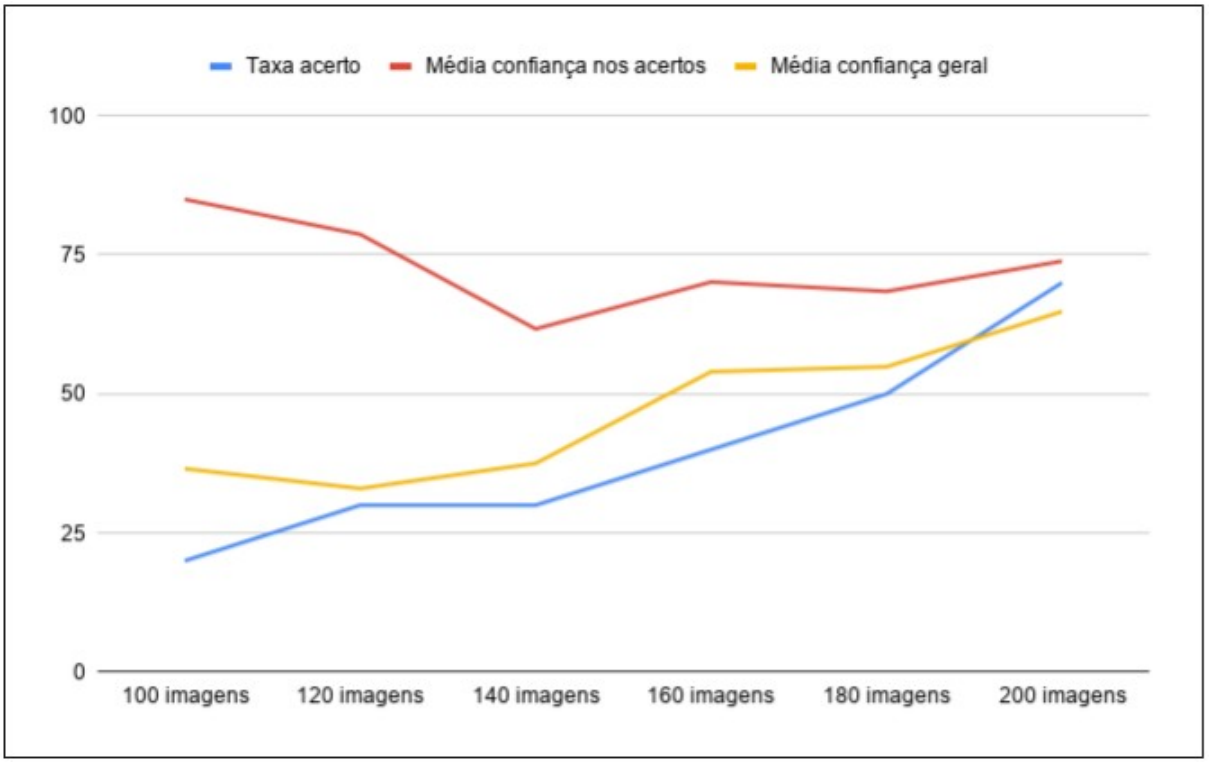
O percentual de confiança que decai entre um teste com mais imagens e eleva-se com menos imagens, visto em alguns testes, pode ser explicado tanto pelo fato de existirem novos conhecimentos aplicados, como pelo enquadramento realizado em partes distintas da imagem durante o teste.

Imagens rotuladas incorretamente podem gerar confusão em uma RNA. Mas na hipótese surgida pelos testes anteriores, o conhecimento novo adicionado a cada ciclo de testes pode ter gerado essa desordem. Isso acontece se novas imagens treinadas da doença mancha parda forem muito parecidas com a da mancha alvo. Naturalmente a confiança que a rede possuía no ciclo anterior de testes, com a mesma imagem, sofre uma queda.

O foco em uma parte específica da imagem, dado pelo usuário na parte de interesse a ser analisada, pode ter impactado também em alguns resultados onde viu-se algum declínio de confiança. Na medida em que uma imagem é posta para ser classificada, a posição do enquadramento do software (exemplo na Figura 31) dificilmente é a mesma do teste prévio. Desta forma, podem haver diferenças consideráveis na certeza da resposta entregue pelo software entre um ciclo de testes e outro.

Para obter uma melhor compreensão sobre o prognóstico pressuposto através dos testes incrementais da Tabela 3, as médias gerais de cada ciclo foram aplicadas no Gráfico 1.

Gráfico 1 - Resultados dos testes de classificação incremental com 2 classes




Fonte: Do autor (2019).

A média de confiança nos acertos decaiu nos testes de 120 e 140 imagens, voltando a subir levemente a partir dos testes com 160 imagens. Uma vez que o resultado de um teste passou de incorreto para correto, esse valor de acerto foi adicionado na média de confiança, sendo que esse valor não correspondeu a um

valor alto, logicamente, devido a ele ter ascendido de um teste prévio incorreto. Por outro lado, a taxa de acerto sobe na medida em que novos testes foram efetuados.

A assertividade do software cresceu consideravelmente (linha azul do Gráfico 1) na medida em que foi adicionado conhecimento novo na rede neural. Isso também foi especificado na literatura do Capítulo 2 deste trabalho, na Seção 2.3.3, onde descreveu-se que uma das principais características de uma RNA é a adaptabilidade através de novos estímulos de conhecimento.

De acordo com Haykin (2008), a característica de importante relevância para o funcionamento de uma rede neural, é o aprendizado através do ambiente em que foi introduzida, aprimorando seu desempenho a partir do conhecimento adquirido. A rede aprende através de ajustes, tornando-se instruída após seguidas iterações durante o processo de aprendizagem.

Embora as taxas de confiança tenham decaído, devida a ascensão lenta dos acertos em testes individuais, o percentual médio de confiança geral subiu significativamente. Do ciclo de testes de 100 imagens para o de 200 imagens, a confiança média cresceu 177,32%.

O Gráfico 1 sintetiza a hipótese do experimento realizado, onde esperava-se uma melhora nos valores de confiança entregues pelo software, na medida em que novas imagens fossem treinadas.

Outro experimento realizado envolveu os ajustes de parâmetros da rede neural MobileNet, existentes na biblioteca ml5 e passíveis de serem alterados. Esses parâmetros são também conhecidos como hiperparâmetros, que alteram o comportamento do modelo da rede neural. Os hiperparâmetros alterados nos próximos testes foram: *epochs*, quantidade de camadas ocultas e taxa de aprendizado. O valor da taxa de aprendizado é aplicada na fórmula de gradiente descendente da RNA, mencionada na Seção 2.3.3.5.

A taxa de aprendizado é um hiperparâmetro de escolha significativa para o MobileNet. Caso a taxa de aprendizado for muito alta, a perda média resultante ficará também alta. Uma taxa de aprendizado ideal é usualmente próxima ao fator

de 2, correspondente à maior taxa que não causa discordâncias nos critérios de treinamento. Um exemplo disso é iniciar o treinamento com uma taxa mais alta, caso hajam divergências, baixa-se o valor gradativamente, até o ponto em que mais nenhuma divergência seja observada (BENGIO, 2012).

Os valores iniciais que o ml5 atribui aos hiperparâmetros são:

- Número de *epochs*: 20
- Quantidade de camadas ocultas: 100
- Taxa de aprendizado: 0.0001

O experimento de ajuste desses parâmetros visou obter a melhor configuração para o determinado problema proposto, baseando-se na acuracidade das respostas. Outrossim, a hipótese de desconfiguração parcial ou total da rede neural, a partir de inconsistências nos resultados, também foi observada.

Para esse teste foram escolhidas 3 classes de imagens de doenças na soja: ferrugem, mancha parda e olho de rã, com uma base total de 336 imagens (112 imagens para cada doença). A escolha se deu pela característica distinta de manchas dessas doenças, eliminando problemas de classificação decorrentes à similaridade entre as classes, bem como falta de imagens suficientes para resultados assertivos, resultando em classificações iniciais corretas. Dessa forma, pôde-se focar no objetivo do experimento, que foi avaliar a mudança de comportamento do software a partir de ajustes em suas configurações iniciais.

Seguindo a mesma metodologia de testes das Tabelas 2 e 3, para cada doença foram testadas cinco imagens, caracterizadas pela mesma classe, porém distintas uma da outra. Como o objetivo foi medir a diferença dos resultados diante da alteração dos parâmetros iniciais do ml5, o software foi modificado para não efetuar o recorte da imagem a ser classificada. A ideia foi evitar que a posição do enquadramento entre um teste e outro pudesse influenciar nos resultados. Portanto, todas as imagens de teste foram previamente recortadas com o tamanho de 120x120 pixels, em uma posição fixa, focalizando a mancha característica da

doença. Assim, assegurou-se que em cada teste de ajuste de parâmetros fossem utilizadas as mesmas imagens.

A listagem a seguir descreve as configurações aplicadas para cada teste, sendo que o primeiro teste consiste no padrão utilizado pelo software:

- Experimento 1: *epochs*: 20, camadas ocultas: 100, taxa de aprendizado: 0.0001.
- Experimento 2: *epochs*: 25, camadas ocultas: 120, taxa de aprendizado: 0.00025.
- Experimento 3: *epochs*: 30, camadas ocultas: 140, taxa de aprendizado: 0.0005.
- Experimento 4: *epochs*: 35, camadas ocultas: 160, taxa de aprendizado: 0.00075.
- Experimento 5: *epochs*: 40, camadas ocultas: 180, taxa de aprendizado: 0.001.
- Experimento 6: *epochs*: 45, camadas ocultas: 200, taxa de aprendizado: 0.0025.

A Tabela 4 apresenta os resultados do experimento de alteração dos hiperparâmetros.

Tabela 4 - Testes com 3 classes de imagens com alteração de parâmetros

| **Experimento 1: epochs: 20, camadas ocultas: 100, taxa de aprendizado: 0.0001** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **Acerto/Confiança** | | | | | | | |
| **Doença** | **Teste 1** | **Teste 2** | **Teste 3** | **Teste 4** | **Teste 5** | **Taxa de acerto** | **Média confiança dos acertos** | **Média confiança geral** |
| Ferrugem | A/99,76 | A/80,35 | A/96,82 | A/94,70 | A/99,41 | 100% | 94,21% | 94,21% |
| Mancha parda | A/78,10 | A/99,86 | A/96,22 | A/95,53 | A/99,79 | 100% | 93,90% | 93,90% |
| Olho de rã | A/89,65 | A/99,71 | A/98,73 | A/80,15 | A/97,56 | 100% | 93,16% | 93,16% |
| **Média geral** | | | | | | **100%** | **93,76%** | **93,76%** |

Continua

| Experimento 2: epochs: 25, camadas ocultas: 120, taxa de aprendizado: 0.00025 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Acerto/Confiança | | | | | | | |
| Doença | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Taxa de acerto | Média confiança dos acertos | Média confiança geral |
| Ferrugem | A/99,99 | A/99,73 | A/99,99 | A/99,99 | A/99,99 | 100% | 99,34% | 99,34% |
| Mancha parda | A/98,03 | A/99,67 | A/99,94 | A/99,99 | A/99,90 | 100% | 99,50% | 99,50% |
| Olho de rã | A/93,22 | A/99,99 | A/99,99 | A/98,18 | A/99,98 | 100% | 98,27% | 98,27% |
| **Média geral** | | | | | | **100%** | **99,04%** | **99,04%** |
| **Experimento 3: epochs: 30, camadas ocultas: 140, taxa de aprendizado: 0.0005** | | | | | | | | |
| | Acerto/Confiança | | | | | | | |
| Doença | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Taxa de acerto | Média confiança dos acertos | Média confiança geral |
| Ferrugem | A/65,02 | A/70,64 | A/99,95 | A/83,39 | A/99,73 | 100% | 83,75% | 83,75% |
| Mancha parda | A/99,93 | A/100 | A/100 | A/99,99 | A/100 | 100% | 99,98% | 99,98% |
| Olho de rã | A/79,69 | A/99,99 | A/100 | A/78,43 | A/99,87 | 100% | 91,60% | 91,60% |
| **Média geral** | | | | | | **100%** | **91,78%** | **91,78%** |
| **Experimento 4: epochs: 35, camadas ocultas: 160, taxa de aprendizado: 0.00075** | | | | | | | | |
| | Acerto/Confiança | | | | | | | |
| Doença | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Taxa de acerto | Média confiança dos acertos | Média confiança geral |
| Ferrugem | A/99,97 | A/83,31 | A/55,63 | A/99,99 | A/99,98 | 100% | 87,78% | 87,78% |
| Mancha parda | A/99,97 | A/99,99 | A/99,98 | A/100 | A/100 | 100% | 99,99% | 99,99% |
| Olho de rã | A/99,93 | A/100 | A/100 | A/99,97 | A/100 | 100% | 99,98% | 99,98% |
| **Média geral** | | | | | | **100%** | **95,92%** | **95,92%** |
| **Experimento 5: epochs: 40, camadas ocultas: 180, taxa de aprendizado: 0.001** | | | | | | | | |
| | Acerto/Confiança | | | | | | | |
| Doença | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Taxa de acerto | Média confiança dos acertos | Média confiança geral |
| Ferrugem | A/77,15 | E/0,05 | A/99,98 | A/96,13 | A/100 | 80% | 93,32% | 74,66% |
| Mancha parda | A/100 | A/99,99 | A/100 | A/100 | A/100 | 100% | 100,00% | 100,00% |
| Olho de rã | E/0,01 | A/100 | A/100 | E/28,69 | A/100 | 60% | 100,00% | 65,74% |
| **Média geral** | | | | | | **80%** | **97,77%** | **80,13%** |
| **Experimento 6: epochs: 45, camadas ocultas: 200, taxa de aprendizado: 0.0025** | | | | | | | | |

Continua

| Experimento 6: epochs: 45, camadas ocultas: 200, taxa de aprendizado: 0.0025 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Acerto/Confiança | | | | | | | |
| **Doença** | **Teste 1** | **Teste 2** | **Teste 3** | **Teste 4** | **Teste 5** | **Taxa de acerto** | **Média confiança dos acertos** | **Média confiança geral** |
| Ferrugem | A/100 | E/9,79 | E/2,95 | A/100 | A/100 | 60% | 100,00% | 62,55% |
| Mancha parda | E/0 | E/0 | E/0 | E/6,87 | A/100 | 20% | 100,00% | 21,37% |
| Olho de rã | E/6,07 | A/100 | A/100 | E/0 | A/97,87 | 60% | 99,29% | 60,79% |
| **Média geral** | | | | | | **46,67%** | **99,76%** | **48,24%** |

Fonte: Do autor (2019).

Os resultados do experimento de alteração dos parâmetros do ml5, apresentados na Tabela 4, mostram desequilíbrios significativos. Para cada ciclo novo de testes notou-se que houveram mudanças de comportamento no valor de confiança entregue.

No experimento 2 houve uma alta considerável nos valores de confiança, comparado ao experimento 1. Embora a resposta tenha sido a correta, os valores para as outras imagens que concorriam com uma resposta, praticamente zeraram. A partir do primeiro experimento, a confiança para a resposta certa foi desproporcional para todos os testes.

No experimento 3, o valor de confiança desigual para os testes das doenças mancha parda e olho de rã começou a ficar mais evidente, repetindo-se no experimento 4.

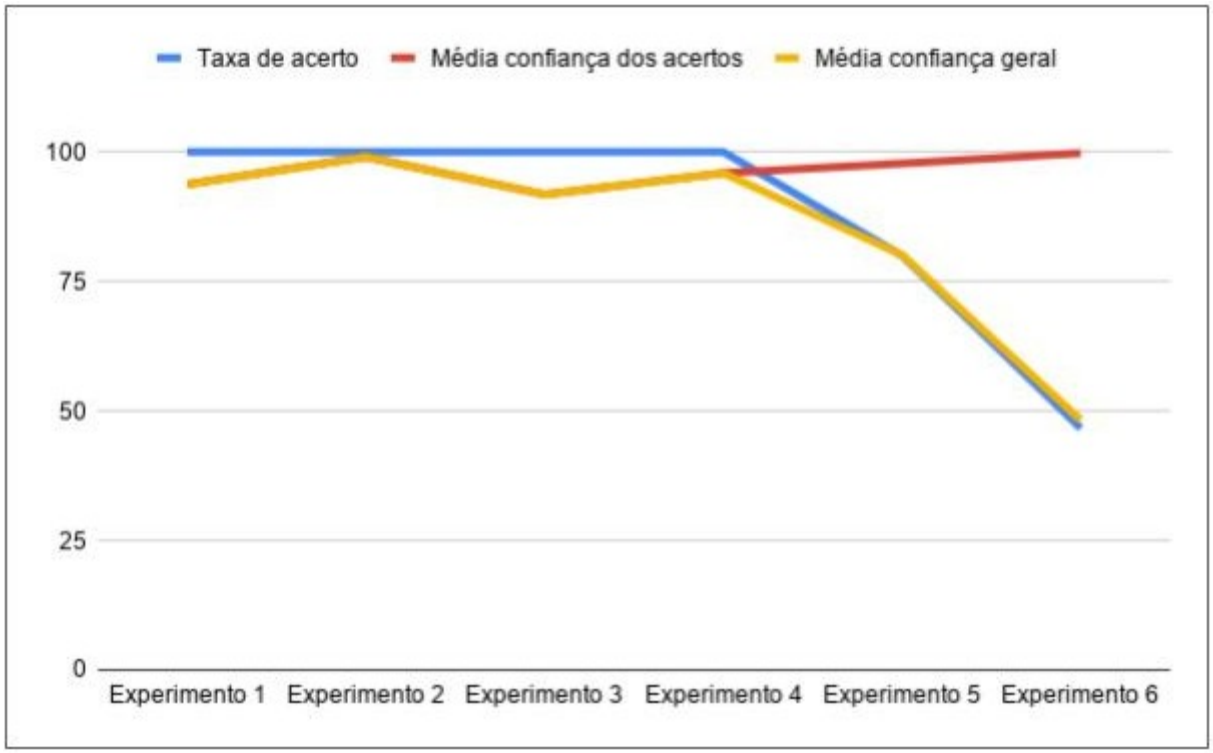
No experimento 5 ocorreram os primeiros erros dos ciclos de testes. Esses erros foram acompanhados de uma confiança desproporcionalmente baixa, considerando os testes vizinhos.

A hipótese de desconfiguração total da rede neural MobileNet foi comprovada no experimento 6, onde ocorreram vários erros e valores de confiança muito baixos e outros muito altos. Constatou-se que quando o resultado era o correto, o valor de certeza era alto em demasia, posto que, nas respostas incorretas, os valores de confiança ficaram praticamente insignificantes.

A escolha incorreta de configurações aplicadas em uma RNA também foi evidenciada na literatura, na Seção 2.3.3.1, onde explicou-se que os níveis dos parâmetros podem levar a um superdimensionamento na rede, flexibilizando-a excessivamente e gerando hipóteses inválidas (BRAGA, CARVALHO e LUDERMIR, 2011).

O Gráfico 2 ilustra os resultados obtidos nos experimentos da Tabela 4. Nele pôde-se observar o momento em que ocorre a desconfiguração completa da rede neural do software nos experimentos 5 e 6.

Gráfico 2 - Resultados dos testes de alteração nos parâmetros do software




Fonte: Do autor (2019).

O experimento de ajuste dos hiperparâmetros da rede neural MobileNet, permitido pelo ml5 a partir de opções customizadas na ferramenta, revelou que as configurações iniciais da biblioteca são adequadas para casos parecidos com o problema proposto, no qual haviam 3 classes de imagens, com pouco mais de 100 imagens cada.

Observou-se que o ajuste dos hiperparâmetros deve ser realizado com cautela, realizando rotinas de testes para verificar a acurácia e coerência dos resultados. Para a maioria dos casos, como o experimento realizado apresentado na Tabela 3, as configurações iniciais da ferramenta devem ser mantidas. Sendo necessário somente acrescentar mais conhecimento na rede neural para aumentar a acurácia dos resultados.

Considerando as oito doenças na soja encontradas nas pesquisas, os resultados do primeiro experimento (Tabela 2) não foram assertivos. Devido à ausência de uma quantidade adequada de imagens na Internet, não foi possível criar uma base de imagens grande o suficiente para classificar corretamente todas as 8 classes. Esse experimento demonstrou um comportamento que lembra muito o processo cognitivo humano, em que algumas imagens foram confundidas com outras muito semelhantes.

Os resultados dos testes da Tabela 3 mostraram que é possível ter uma boa acurácia, quando adicionadas novas imagens na rede neural. Para obter o mesmo cenário para o treinamento de uma rede contendo oito classes, a quantidade de imagens de treino teria que, no mínimo dobrar, para ser suficientemente capaz de retornar resultados melhores.

No que refere-se às configurações padronizadas da ferramenta, o experimento de ajuste nos parâmetros da rede neural MobileNet apresentou uma ótima acurácia. Isso ocorreu pelo fato das três classes testadas serem oriundas de imagens com características distintas, facilitando o aprendizado e a classificação. Já as alterações nas configurações iniciais da ferramenta, apontaram diversas previsões inválidas, destacando a importância de realização de vários testes de ajustes para cada treinamento, visando a uniformidade da rede neural.

# 6 CONCLUSÕES

O presente trabalho teve como objetivo principal utilizar Inteligência Artificial para reconhecer doenças na soja a partir de imagens de folhas da planta. Para tal fim, implementou-se um software que instancia uma rede neural através da biblioteca ml5 (invólucro para a plataforma TensorFlow), possibilitando adicionar conhecimento a ela através de aprendizagem supervisionada.

De forma geral, baseando-se na quantidade de recursos de imagens, no ambiente proposto e nos resultados obtidos, pode-se afirmar que foi possível identificar doenças na soja a partir do software criado. Quanto à acuracidade desses resultados, existem ressalvas alusivas exclusivamente ao treinamento adequado da rede neural.

Conforme apresentado nas análises dos experimentos, quanto maior for a quantidade de classes distintas de imagens presentes em um treinamento, maior deve ser a quantidade total dessas imagens. As análises mostraram que para imagens com padrões próprios, o software teve uma acurácia excelente. Mas quando imagens com características semelhantes foram adicionadas ao treinamento, ocorreram confusões nas previsões.

Foi constatado através de experimento, que quando acrescentou-se mais conhecimento na rede neural, as previsões começaram a melhorar gradativamente. Em outras palavras, quanto maior for a base de imagens de treinamento, melhor serão os resultados.

Outra constatação evidenciada através de experimentos, foi que uma rede neural com mais neurônios e que efetua mais iterações, não significa que trará melhores resultados, visto que o excesso de flexibilidade da mesma causa inconsistências severas nos resultados de classificação. Verificou-se que o nível de parâmetros iniciais da rede deve ser ajustado de acordo com o estudo empreendido, sendo que esses ajustes devem ser testados de acordo com a alteração do tamanho da base de treino.

## 6.1 Trabalhos futuros

O estudo realizado mostrou que existem algumas melhorias necessárias para aumentar a precisão e aplicabilidade do software desenvolvido. A seguir são listadas algumas propostas de avanços futuros para uma continuidade deste trabalho.

- Aumentar a base de imagens para treinamento: com a insuficiência de imagens na Internet para realizar um treinamento adequado, é necessário a saída de campo para obtenção de imagens em sua origem. Isso requer, além da busca por lavouras de soja que estejam sendo atingidas por doenças, o auxílio de um agrônomo para a identificação da mesma, para rotular corretamente as imagens que serão capturadas.

- Banco de dados: para evitar repetir o treinamento em cada utilização do software, salvar o conhecimento adquirido pela rede neural em um banco de dados possibilitaria acessar o software já pronto para uso.

- Automatizar o processo de coleta de imagens: a fase de extração de pequenas imagens, contendo áreas de interesse relativas à doença, mostrou-se custosa em relação ao tempo gasto. Propõe-se para trabalhos futuros a automatização desse processo, por meio de bibliotecas computacionais, para remover o fundo das imagens ou partes irrelevantes de forma automática.

- Criar aplicativo móvel para validação em campo: um aplicativo móvel facilitaria não só a identificação de doenças na própria planta, mas também a possibilidade de adicionar novas imagens na base.

# REFERÊNCIAS

ABADI, MARTÍN et al. **Tensorflow: A system for large-scale machine learning**. 12th Symposium on Operating Systems Design and Implementation. p. 265-283, 2016.

ABADI, MARTÍN et al. **Tensorflow: Large-scale machine learning on heterogeneous distributed systems**. arXiv 2016, 2015.

ARAÚJO, Flávio HD et al. **Redes Neurais Convolucionais com Tensorflow: Teoria e Prática**. SOCIEDADE BRASILEIRA DE COMPUTAÇÃO. III Escola Regional de Informática do Piauí. Livro Anais-Artigos e Minicursos, v. 1, p. 382-406, 2017.

ARTERO, Almir Olivette. **Inteligência Artificial: Teórica e Prática**. São Paulo: Livraria da Física, 2009.

BENGIO, Yoshua. **Practical recommendations for gradient-based training of deep architectures**. **Neural networks: Tricks of the trade**. Springer, Berlin, Heidelberg, p. 437-478, 2012.

BITTENCOURT, Guilherme. **Inteligência Artificial: ferramentas e teorias**. **3. ed**. Florianópolis: Ed. da UFSC, 2006.

BRAGA, Antônio de Pádua; CARVALHO, André Carlos Ponce de Leon Ferreira de; LUDERMIR, Teresa Bernarda. **Redes Neurais Artificiais: Teoria e Aplicações. 2. ed**. Rio de Janeiro: LTC, 2011.

CARDOSO, Olinda Nogueira Paes. **Recuperação de Informação**. INFOCOMP, v. 2, n. 1, p. 33-38, 2004.

COPPIN, Ben. **Inteligência Artificial**. Rio de Janeiro: LTC, 2013.

CROSS, Simon S.; HARRISON, Robert F.; KENNEDY, R. Lee. **Introduction to neural networks**. The Lancet, v. 346, n. 8982, p. 1075-1079, 1995.

DAVIS, Randall; SHROBE, Howard; SZOLOVITS, Peter. **What is a knowledge representation?. AI magazine**, v. 14, n. 1, p. 17, 1993.

DENG, Jia et al. **Imagenet: A large-scale hierarchical image database**. 2009 IEEE conference on computer vision and pattern recognition. IEEE, p. 248-255, 2009.

EMBRAPA, Empresa Brasileira de Pesquisa Agropecuária. **Soja em números (safra 2017/2018)**. Disponível em: <https://www.embrapa.br/web/portal/soja/cultivos/soja1/dados-economicos>. Acesso em 20 de Agosto de 2018.

ESTEVA, Andre et al. **Dermatologist-level classification of skin cancer with deep neural networks**. Nature, v. 542, n. 7639, p. 115, 2017.

FACELI, Katti; LORENA, Ana Carolina; GAMA, João; CARVALHO, André Carlos Ponce de Leon Ferreira de. **Inteligência Artificial : uma abordagem de aprendizado de máquina**. Rio de Janeiro: LTC, 2011.

FERNANDES, Anita Maria da Rocha. **Inteligência Artificial: noções gerais**. Florianópolis: VisualBooks, 2005.

FERNANDES, Bruno José Torres. **Redes neurais com extração implícita de características para reconhecimento de padrões visuais**. Tese de Doutorado. Centro de Informática, Pernambuco: Universidade Federal de Pernambuco, 2013.

FERREIRA, Arthur Emídio Teixeira. **Estimação do ângulo de direção por vídeo para veículos autônomos utilizando redes neurais convolucionais multicanais**. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação), Brasília: Universidade de Brasília, 2017.

GONÇALVES, Wesley Nunes et al. **Técnicas de segmentação baseadas em subtração de fundo e modelos de cores: Um estudo comparativo**. XXVIII CILAMCE-Iberian Latin American Congress on Computational Methods in Engineering, p. 13-15, 2007.

GOOGLE AI. **About Google AI Team**. Disponível em: <https://ai.google/about/>. Acesso em 20 de Março de 2019.

HAYKIN, Simon. **Redes Neurais: princípios e prática. 2. ed**. Porto Alegre: Bookman, 2008.

HENNING, Ademir Assis et al. **Manual de identificação de doenças de soja**. Embrapa Soja-Documentos (INFOTECA-E), 2014.

HOWARD, Andrew G. et al. **Mobilenets: Efficient convolutional neural networks for mobile vision applications**. arXiv, 2017.

KHATCHATOURIAN, Oleg; PADILHA, Fábio RR. **Reconhecimento de variedades de soja por meio do processamento de imagens digitais usando redes neurais artificiais**. Eng. Agric. Jaboticabal, v. 28, n. 4, p. 759-69, 2008.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. **Imagenet classification with deep convolutional neural networks**. Advances in neural information processing systems, p. 1097-1105, 2012.

LAWRENCE, Steve et al. **Face recognition: A convolutional neural-network approach**. IEEE transactions on neural networks, v. 8, n. 1, p. 98-113, 1997.

LUGER, George F. **Inteligência Artificial. 6. ed**. São Paulo: Pearson Education do Brasil, 2013.

MATIAS-PEREIRA, José. **Manual de metodologia da pesquisa científica**. 4. ed. São Paulo: Atlas, 2016.

MEDEIROS, Luciano Frontino de. **Inteligência artificial aplicada: uma abordagem introdutória**. Curitiba: InterSaberes, 2018.

MIKOLOV, Tomáš et al. **Recurrent neural network based language model**. Eleventh annual conference of the international speech communication association. 2010.

ML5JS. **Feature Extractor.** Disponível em: <https://ml5js.org/reference/api-FeatureExtractor>. Acesso em 6 de Abril de 2019.

ML5JS. **Getting Started**. Disponível em: <https://ml5js.org/getting-started>. Acesso em 5 de Abril de 2019.

PERELMUTER, Guy et al. **Reconhecimento de imagens bidimensionais utilizando Redes Neurais Artificiais**. Anais do VIII SIBGRAPI, p. 197-203, 1995.

RAMOS, Albenides. **Metodologia da pesquisa científica: como uma monografia pode abrir o horizonte do conhecimento**. São Paulo: Atlas, 2009.

REZENDE, Solange Oliveira. **Sistemas Inteligentes: Fundamentos e Aplicações**. Barueri: Manole, 2005.

SANDRI, Sandra; CORREA, Cláudio. **Lógica nebulosa**. Instituto Tecnológico da Aeronáutica–ITA, V Escola de Redes Neurais, pp. C073-c090. São José dos Campos, 1999.

TENSORFLOW. **Image Classification**. Disponível em: <https://www.tensorflow.org/lite/models/image_classification/overview>. Acesso em 5 de Abril de 2019.

TENSORFLOW. **Tensors**. Disponível em: <https://www.tensorflow.org/guide/tensors>. Acesso em 5 de Abril de 2019.

TENSORFLOW. **Why TensorFlow**. Disponível em: <https://www.tensorflow.org/about>. Acesso em 5 de Abril de 2019.

TURING, Alan M. Computing machinery and intelligence (1950). **The Essential Turing: The Ideas that Gave Birth to the Computer Age. Ed. B. Jack Copeland. Oxford: Oxford UP**, p. 433-64, 2004.

# APÊNDICES

APÊNDICE A – Inicialização do MobileNet

```
initMl5(images, instance) {
        if (images && images.length > 0) {
            // network options
            let options = {
                numClasses: NUM_CLASSES,
                batchSize: images.length
            };
            // Load MobileNet
            featureExtractor = ml5.featureExtractor('MobileNet', options, modelLoaded);

            function modelLoaded() {
                instance.state.set('status', 'Model carregado!');
                setTimeout(() => {
                    // Starting featureExtractor method for classification
                    classifier = featureExtractor.classification();
                    // Starting adding images
                    networkController.prototype.addImages(images, instance);
                }, 2000);
            };
        }
    };
```

Fonte: Do autor (2019).

APÊNDICE B – Adição das imagens para treinamento

```
addImages(images, instance) {
        let index = 0;
        let addImage = () => {
            if (images[index]) {
                let image = images[index];
                // Get image label to training
                let label = image.substring(0, image.indexOf('-'));
                // Instantiate new Image object
                let img = `<img id="${image}"
                src="/images/${FOLDER}/${image}"
                title="${label}"
                class="margin-right-5">`;
                $("#containerImages").append(img);
                let imageElement = document.getElementById(`${image}`);
                // Wait for image load
                imageElement.onload = () => {
                    classifier.addImage(imageElement, label, () => {
                        instance.state.set('processingImage',
                        `Adicionando imagem ${index + 1}: ${image}`);
                        index++;
                        setTimeout(() => { addImage() }, 10);
                    });
                };
            } else {
                instance.state.set('status', 'Imagens adicionadas!');
                instance.state.set('label', `${images.length} imagens para treinar..`);
                $("#btnStartTraining").removeClass("hide");
            }
        };

        addImage();
    };
```

Fonte: Do autor (2019).

## APÊNDICE C – Treinamento das imagens

```
startTraining(instance) {
    classifier.train(function (lossValue) {
        if (lossValue == null) {
            instance.state.set('status', 'Treinamento completo!');
            instance.state.set('label', 'Escolha uma imagem para classificar');
            instance.state.set('processingImage', '');
            $("#labelErrorTax").text("");
            $("#containerImageToPredict").removeClass("hide");
            $("#trainingPanel").addClass("hide");
        } else {
            $("#labelErrorTax").text("Valor de perda: "+lossValue);
        }
    });
};
```

Fonte: Do autor (2019).

## APÊNDICE D – Classificação de uma imagem carregada

```
classify(instance) {
    let image = document.getElementById('imageToPredict');
    classifier.classify(image, (err, results) => {
        let array = [];
        results.forEach((element, index) => {
            array.push({
                label: element.label,
                confidence: `${element.confidence * 100} %`
            });
            if(index===(results.length-1)) {
                instance.state.set('results', array);
            }
        });
    });
};
```

Fonte: Do autor (2019).