UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

JEAN DANIEL PRESTES MASSUCATTO

# APLICAÇÃO DE CONCEITOS DE REDES NEURAIS CONVOLUCIONAIS NA CLASSIFICAÇÃO DE IMAGENS DE FOLHAS

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2018

JEAN DANIEL PRESTES MASSUCATTO

# APLICAÇÃO DE CONCEITOS DE REDES NEURAIS CONVOLUCIONAIS NA CLASSIFICAÇÃO DE IMAGENS DE FOLHAS

Trabalho de Conclusão de Curso 1, apresentado à disciplina de Trabalho de Conclusão de Curso 1, do Curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná - UTFPR, Câmpus Pato Branco, como requisito parcial para obtenção do título de Engenheiro de Computação.

Orientador: Prof. Dr. Dalcimar Casanova

PATO BRANCO

2018

**Ministério da Educação**
**Universidade Tecnológica Federal do Paraná**
**Câmpus Pato Branco**
**Departamento Acadêmico de Informática**
**Curso de Engenharia de Computação**

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

## TERMO DE APROVAÇÃO

Às 16 horas do dia 05 de julho de 2018, na sala V007, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, reuniu-se a banca examinadora composta pelos professores Dalcimar Casanova (orientador), Ives Rene Venturini Pola e Pablo Gauterio Cavalcanti para avaliar o trabalho de conclusão de curso com o título **Aplicação de conceitos de redes neurais convolucionais na classificação de imagens de folhas**, do aluno **Jean Daniel Prestes Massucatto**, matrícula 01260430, do curso de Engenharia de Computação. Após a apresentação o candidato foi arguido pela banca examinadora. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Dalcimar Casanova
Orientador (UTFPR)

_____
Ives Rene Venturini Póla
(UTFPR)

_____
Pablo Gauterio Cavalcanti
(UTFPR)

_____
Profa. Beatriz Terezinha Borsoi
Coordenador de TCC

_____
Prof. Pablo Gauterio Cavalcanti
Coordenador do Curso de
Engenharia de Computação

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

*"You must understand that there is more than one path to the top of the mountain."*

Miyamoto Musashi

## RESUMO

MASSUCATTO, Jean Daniel Prestes. Aplicação de conceitos de Redes Neurais Convolucionais na classificação de imagens de folhas. 2018. 68f. Trabalho de Conclusão de Curso de bacharelado em Engenharia de Computação - Universidade Tecnológica Federal do Paraná. Pato Branco, 2018.

Este trabalho propõe um algoritmo para solucionar um dos problemas enfrentados por taxonomistas, a classificação de folhas, um processo que até então é feito manualmente e emprega muito tempo para sua realização, no qual a taxa de sucesso depende simplesmente da capacidade profissional do taxonomista. Com o avanço da tecnologia e surgimento do aprendizado profundo ou *deep learning*, as redes neurais convolucionais foram foco de grandes estudos na área de classificação de imagens. Atualmente, diversos pesquisadores dedicam seu tempo em desenvolver algoritmos para classificação de imagens foliares baseados nos conceitos intrínsecos as redes neurais convolucionais. A rede neural desenvolvida neste trabalho foi treinada e testada com um banco de imagens disponível pelo projeto PlantCLEF no ano de 2016 entre outros.

**Palavras-chave**: Redes Neurais. Redes Neurais Convolucionais. Aprendizado Profundo. Classificação de Folhas.

## ABSTRACT

MASSUCATTO, Jean Daniel Prestes. Application of concepts of Convolutional Neural Networks in classification of leaves. 2018. 68f. Trabalho de Conclusão de Curso de bacharelado em Engenharia de Computação - Universidade Tecnológica Federal do Paraná. Pato Branco, 2018.

This work proposes an algorithm to solve one of the problems faced by taxonomists, the classification of leaves, a process that is done manually and takes a lot of time to accomplish, where the success rate depends simply on the professional capacity of the taxonomist. With the advance of technology and the emergence of deep learning, the convolutional neural networks were the focus of large studies in the area of image classification. Nowadays several researchers dedicate their time in developing algorithms for classification of foliar images based on the intrinsic concepts of convolutional neural networks. The developed neural network in this work was trained and tested on an image dataset available by the PlantCLEF project in the year 2016 among others.



**Keywords**: Neural Networks. Convolutional Neural Networks. Deep Learning. Leaves Classification.

## LISTA DE FIGURAS

**SUMÁRIO**

# 1 INTRODUÇÃO

Redes Neurais Artificiais (*Artificial Neural Networks* - ANNs) comumente chamadas de Redes Neurais tem seu estudo baseado no comportamento dos neurônios do cérebro humano e consistem em abordar problemas relacionados à Inteligência Artificial. Pode-se dizer que o cérebro humano é um computador altamente complexo, não-linear e paralelo com a capacidade de organizar seus constituintes estruturais muito mais rapidamente que o mais rápido computador digital hoje existente (HAYKIN, 2004).

Assimilando a ideia do cérebro humano, uma rede neural é baseada em uma coleção de unidades de processamento conectadas entre si chamadas de neurônios artificiais, os quais inicialmente possuíam objetivos de solucionar problemas da mesma maneira que um cérebro faria.

Os estudos sobre redes neurais artificiais têm seu início em meados de 1940, com o trabalho de McCulloch e Pitts (1943), que demonstraram que redes neurais artificiais poderiam calcular funções aritméticas e lógicas.

A primeira prática para redes neurais artificiais surgiu no final dos anos 50, com a invenção da arquitetura perceptron e seu algoritmo de aprendizado por Frank Rosenblatt (ROSENBLATT, 1958), embora com pouco desempenho de processamento. Quase vinte anos depois surge o algoritmo de *backpropagation*, que foi responsável por revigorar os estudos das redes neurais e mantê-los em evolução nas últimas décadas (FERREIRA, 2017).

Na abordagem de uma rede *Perceptron* ou rede neural artificial, tarefas maiores são desmembradas em tarefas menores precisamente definidas e que em conjunto apresentam a solução do problema. Porém, não é ensinado à máquina como resolver o problema, o aprendizado consiste na observação de dados, descobrindo sua própria solução para o problema em questão.

Em 2006 houve um grande salto para os estudos sobre redes neurais, a descoberta das técnicas de aprendizagem nas chamadas redes neurais profundas, conhecidas como *Deep Learning*, obteve resultados bastante relevantes em muitos problemas importantes como visão computacional (SRINIVAS et al., 2016), reconhecimento de fala (HINTON et al., 2012) e processamento de linguagem natural (GOLD-

BERG, 2016). Além disso, atualmente, redes neurais e *Deep Learning* fornecem as melhores soluções para muitos problemas no reconhecimento de imagens.

*Deep Learning* é, em sua essência, uma rede neural artificial, porém a diferença básica entre essas duas redes é o número de camadas, ou seja, uma Rede Neural Profunda (Deep Neural Network - DNN). Uma DNN é uma ANN com múltiplas camadas ocultas entre as camadas de entrada e saída (BENGIO et al., 2009). As redes neurais de aprendizado profundo realizam o aprendizado de suas características de forma hierárquica: nos níveis mais altos da hierarquia estão as características de combinação de mais baixo nível. Para realizar uma classificação com uma taxa considerável de acertos, as redes neurais necessitam de uma grande quantidade de dados de entrada e camadas escondidas cada vez mais complexas.

Dentro do universo de redes neurais profundas ou Deep Learning, encontram-se as Redes Neurais Convolucionais (Convolutional Neural Networks - CNNs) ou também chamadas de ConvNets, redes que se inspiram no funcionamento do córtex visual, tendo sua arquitetura adaptada para explorar a correlação espacial existente em imagens naturais.

As CNNs tiveram seu estado da arte alcançado em várias tarefas de reconhecimento de imagem e um dos seus modelos pioneiros foi proposto por LeCun (LECUN et al., 1990), com a CNN organizada em dois tipos de camadas, camadas de convolução e camadas de sub-amostra. Atualmente a arquitetura de uma ConvNet pode ser composta de diversas maneiras, mas em sua forma básica é dividida em estágios como: camadas de convolução, camadas de agrupamento e camadas totalmente conectadas.

Em 2012, a arquitetura proposta por Krizhevsky et al. (2012), foi considerada o marco na utilização de Redes Neurais Convolucionais, sendo obtidos resultados promissores no *ImageNet Large-Scale Visual Recognition Challenge* (ILSVRC). O ILSVRC é uma competição anual de visão computacional na qual os modelos participantes são submetidos à prova para realizar tarefas de classificação, localização, detecção, etc.

Karen Simonyan e Andrew Zisserman da *University of Oxford* (SIMONYAN; ZISSERMAN, 2014), foram responsáveis pelos melhores resultados em 2014 no ILSVRC, com um modelo de CNN de 19 camadas, que usavam filtros 3x3 com passo e preenchimento de 1, juntamente com camadas de *maxpooling* de 2x2 com passo 2.

Em 2015, a GoogLeNet lançou seu modelo chamado *Inception* (SZEGEDY et

al., 2015). Trata-se de uma CNN que desviou da abordagem tradicional de empilhar camadas convolutivas e camadas de agrupamento, em uma estrutura sequencial. A característica principal desta arquitetura é a melhor utilização dos recursos de computação dentro da rede. Essa arquitetura foi elaborada aumentando a profundidade e largura da rede, mantendo o custo computacional constante, usando uma CNN de 22 camadas.

As arquiteturas citadas mostram que não existe uma arquitetura especifica para uma CNN, e não existe um número mínimo de camadas escondidas necessárias para diferenciar uma rede neural artificial comum de uma rede neural convolucional. As CNNs apresentam inúmeras áreas de aplicação, entre elas destaca-se a área de reconhecimento e identificação de imagens, com novos modelos e arquiteturas sendo avaliados e classificados a cada ano no ILSVRC.

## 1.1 OBJETIVOS

### 1.1.1 OBJETIVO GERAL

Desenvolver e testar uma metodologia para classificação de espécies vegetais tendo como base conceitos de Processamento de Imagens, *Deep Learning* e Redes Neurais Convolucionais.

### 1.1.2 OBJETIVOS ESPECÍFICOS

- Análise da base de dados foliar disponibilizadas pelo projeto *ImageCLEF* e outras;
- Analisar os algoritmos de *Deep Learning* já existentes, comparando-os e entendendo suas diferenças;
- Definir e implementar o algoritmo que será utilizado para a classificação das imagens;
- Utilizar métricas (Ex: *Accuracy*), para análise de desempenho na classificação da arquitetura.

## 1.2 JUSTIFICATIVA

A taxonomia é uma das áreas de pesquisa da biologia que tem como objetivo identificar, descrever e classificar a diversidade de seres vivos. A taxonomia vegetal é a ciência responsável pela correta organização, classificação e nomenclatura das espécies de plantas. Alguns estudos estimam que há 298.000 espécies na flora mundial, das quais apenas 215.644 estão catalogadas e nomeadas pelos taxonomistas (MORA et al., 2011).

O método tradicional para classificação e identificação de espécies é treinando taxonomistas que irão examinar e atribuir as respectivas classes taxonômicas das plantas. Entretanto, o número de profissionais taxonomistas é escasso e a imensa variedade de espécies existentes no mundo acaba tornando difícil o trabalho a ser realizado, um taxonomista pode chegar a conhecer algumas centenas de espécimes, um número relativamente baixo se comparado ao número de espécies existentes. Essa situação revela a necessidade de ferramentas capazes de auxiliar nas tarefas taxonômicas, como os herbários vegetais.

Herbários vegetais acabaram tornando-se imprescindíveis no trabalho de taxonomistas e, podem ser vistos como repositórios de plantas, estruturados de maneira a conter todas as informações taxonômicas referente a determinada espécie. Muitas vezes, a fim de melhorar o acesso e com a alta qualidade das maquinas digitais atuais, esses herbários tendem a ser digitalizados, formando um banco de dados com imagens e informações da espécie, data, local e assim por diante. O processo manual de taxonomia e identificação morfológica das plantas é basicamente feito por meio destes herbários Bridson et al. (1998), onde as características morfológicas e anatômicas de órgãos vegetativos e reprodutivos de diferentes espécies podem ser estudadas e analisadas por observação (LEENHOUTS et al., 1968), sendo sua identificação realizada através de comparação com espécimes previamente catalogadas (DEWOLF, 1968), e a taxa de sucesso na identificação e classificação das plantas depende basicamente da experiência do especialista. Apesar da existência dos herbários, todo o trabalho de identificação e caracterização continua sendo feito manualmente (VICENTE et al., 1999), sendo um trabalho difícil, o qual demanda muito tempo para classificação da folha em vista ao imenso número de espécimes. É neste contexto que diversos pesquisadores do mundo inteiro vêm empregando seu tempo desenvolvendo soluções em processamento de imagens para analise automática de imagens.

É neste contexto que o projeto visa trabalhar, melhorando o tempo de classi-

ficação e taxa de acertos baseando-se na utilização de conceitos e técnicas de *Deep Learning* para processamento de imagens, acelerando assim o processo de catalogação das plantas e evitando possíveis erros humanos. O projeto utilizará uma base de dados para treinamento e testes fornecidas pelo projeto do *ImageCLEF*.

Atualmente, diversos pesquisadores do mundo inteiro vêm empregando seu tempo desenvolvendo soluções em processamento de imagens para analise automática de imagens, podendo testar e comparar suas arquiteturas e algoritmos em competições de visão computacional como ILSVRC ou ImageCLEF, onde são disponibilizados *datasets* contendo imagens para treinamento e testes, até um ranking das melhores taxas de acerto e algoritmo utilizado.

## 1.3 ESTRUTURA DO TRABALHO

Este trabalho está organizado da seguinte forma: O Capítulo 2 aborda um breve resumo sobre taxonomia vegetal, e alguns dos principais métodos para análise de folhas. O Capítulo 3 apresenta uma introdução sobre Redes Neurais e sua evolução até as redes *Multilayer Perceptron*. Em seguida o Capítulo 4 começa com uma abordagem sobre o que são Redes Neurais Convolucionais (CNNs) e, em seguida, descreve como são compostas as principais partes de uma CNN, e para finalizar o Capítulo 5 apresenta os bancos de imagens que serão utilizados no decorrer do projeto, seja o de gatos e cachorros para obtenção de resultados preliminares e o banco de imagens foliar disponível pelo projeto do (PLANTCLEF, 2016), seguido do Capítulo 6 que apresenta alguns dos resultados obtidos na realização de testes preliminares realizados em uma CNN básica.

# 2 TAXONOMIA VEGETAL

Em muitos casos, espécies de plantas podem ser identificadas e classificadas por meio de características derivadas do formato da folha ou flor e da ramificação de suas veias. O estudo da morfometria da forma da planta tem sido utilizada há muitos anos. Folhas e flores são objetos não rígidos, levando a uma variedade de deformações, podendo resultar na perda de informações úteis durante o processo de obtenção das imagens digitais de boa qualidade.

Alguns termos que devem ser levados em consideração tratando-se de taxonomia vegetal, como a "classificação", que para os botânicos pode ser definida como o processo de agrupamento com base na similaridade, afim de mensurar características como espécies ou gêneros (STUESSY, 2006), e para cientistas da computação como a atribuição de um exemplo individual a um número finito de categorias no caso, as categorias de folhas presentes no herbário. Plantas também podem ser divididas em reinos, divisão, classe, ordem, família, gênero e espécie, dificultando o modo de aquisição de características para distinção entre essas subdivisões, portanto qualquer sistema apresentado que esteja preocupado com a distinção entre diferentes grupos de plantas deve estar ciente da grande intra-classe e pequena variação entre essas classes. Por exemplo, as folhas mostradas na Figura 1 - são folhas de jabuticaba, laranja, tangerina e manga e todas elas pertencem a divisão *Magnoliophyta* e classe *Magnoliopsida*. Porém, das quatro folhas citadas a única que não pertence à mesma ordem e família é a folha de jabuticaba, embora aos olhos humanos a folha que mais difere entre essas quatro seja a folha de manga.

Classificar plantas entre um imenso número de grupos é certamente mais complexo e tipicamente é necessário um número muito maior de dados para alcançar um desempenho satisfatório. Mesmo que em um estudo uma classificação seja restrita a um único gênero, este gênero pode conter muitas espécies, cada qual abrangerá uma variação entre suas populações constituintes. Portanto, diferentes características se fazem necessárias para categorizar diferentes plantas.

**Figura 1 – Amostras foliares de jabuticabeira, mangueira, tangerineira e laranjeira**
**Fonte: Autoria própria (2017).**

## 2.1 MÉTODOS PARA ANÁLISE DE FOLHAS

Existem muitos aspectos em uma estrutura foliar e sua aparência que podem ser usados por profissionais botânicos na pesquisa por morfologia das plantas, dentre eles os principais aspectos e características são: formas de contorno bidimensionais de uma folha, a estrutura de venação e as margens de uma folha.

### 2.1.1 A FORMA DA FOLHA

A forma da folha é um dos aspectos mais fáceis de ser extraído automaticamente. Em casos em que a folha é impressa contra um fundo branco, técnicas simples de limiar podem ser aplicadas para separar o fundo da folha, e o contorno então pode ser encontrado simplesmente isolando os *pixels* da folha que definem a borda com o fundo. A Figura 2 mostra a estrutura e partes de uma folha.

Uma das técnicas mais comuns para análise de formas aplicadas a folhas são os descritores elípticos de Fourier (Elliptic Fourier Descriptors - EFDs) (KUHL; GIARDINA, 1982), aonde a folha é analisada no domínio da frequência ao invés do domínio espacial. Um conjunto definido de harmônicas de Fourier é calculado para o contorno, cada um dos quais apresentando seus coeficientes. Este conjunto de coeficientes formam os descritores de Fourier, com um maior número de harmônicos fornecendo descrições mais precisas como mostrado na figura 3. Uma das principais vantagens dos descritores de Fourier está na possibilidade de poder reconstruir a

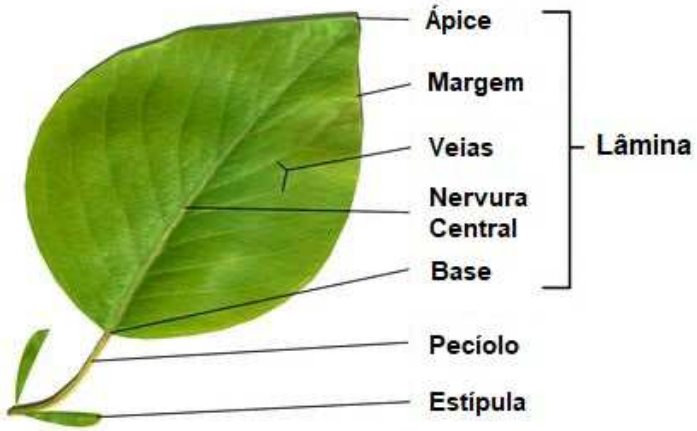


**Figura 2 – Partes da composição de uma folha**
**Fonte: Bailey (2016, p.s.n.)**

forma da folha a partir destes descritores.

McLellan e Endler (1998) comprovaram que a análise por descritores elípticos de Fourier, comparada a outros métodos obteve uma boa discriminação entre diversos grupos de folhas. Eles também apontam que poucos marcos são facilmente identificáveis na maioria das folhas, exceto talvez aqueles que tenham lóbulos regulares. Du et al. (2005) obtiveram sucesso ao combinar descritores elípticos de Fourier com uma rede neural de função radial para identificar espécies de plantas a partir de imagens de folhas.

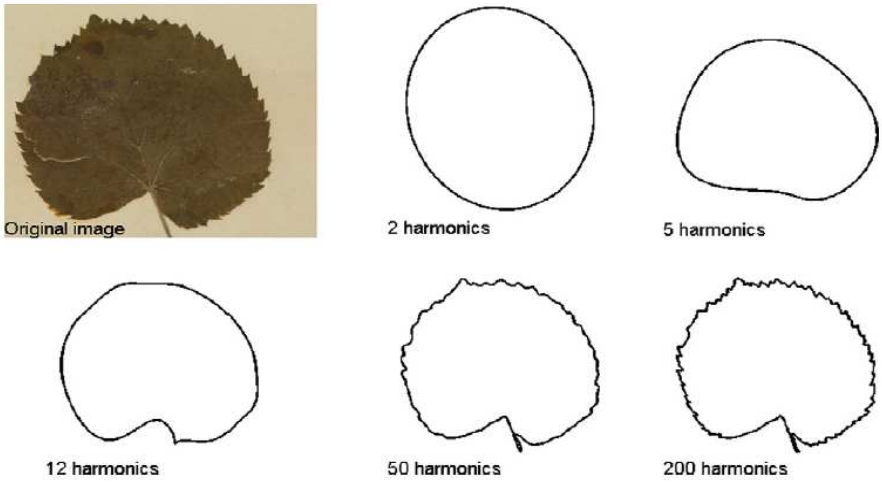


**Figura 3 – Um exemplo EFDs. Quanto mais harmônicos mais a imagem é preservada**
**Fonte: Cope et al. (2010)**

Outra técnica a ser aplicada relacionada a forma de folhas são as assinaturas de contorno. Um contorno de assinatura para a forma é uma sequência de valores

calculados em pontos ao redor do contorno da folha, começando em algum ponto de início e traçando o contorno em sentido horário ou anti-horário. Um dos dados diretos que se pode obter dessa análise é a distância do centróide de contorno, a qual consiste em uma sequência de distâncias entre o centro da folha e os pontos tomados no contorno. O objetivo de obter tais assinaturas de contorno é representar a forma como um vetor independente de orientação e localização, e como reforço também é possível após obtido o vetor de contorno aplicar uma normalização para que o mesmo também seja independente de escala. Embora diversos métodos usem assinaturas de contorno, uma das maiores dificuldades para esses métodos baseados em limites é o problema da auto intersecção, ou seja, quando uma parte da folha se sobrepõe a outra parte da mesma folha podendo resultar em erros ao traçar o esboço da folha. Uma das principais folhas afetadas por esse problema, são as folhas do tipo *lobed leaves* demonstrada na figura 4.

**Figura 4 – Folha lobada**
**Fonte: Gardnerdy (2016, p.s.n.)**

Uma tentativa para superar esse problema foi proposta por Mokhtarian e Abbasi (2004), aonde assumiram que as áreas mais escuras da folha representavam regiões onde a sobreposição ocorreu, e baseando-se nisso, utilizaram o método de espaço de escada de curvatura para assim tentarem extrair os contornos verdadeiros, embora o método só funcione com folhas finas e retro iluminadas, onde a luz consiga passar pela folha para criar as ditas áreas mais escuras.

Por último, pode-se citar ainda se tratando da forma das folhas os métodos de *Landmarks* e medidas lineares, conexão de polígonos e dimensões fractais.

*Landmarks* são conhecidos como pontos de referência. Biologicamente é um ponto definível em um organismo, o qual pode ser sensivelmente comparado com um organismo relacionado. Normalmente, tais marcos requerem conhecimento especializado e especifico do domínio para escolher um conjunto de referência, e comumente são escolhidos a partir de máximos locais ou mínimos de um limite conforme (BOOKSTEIN, 1986). No entanto um número de limitações existe ao aplicar métodos como este, como a dificuldade de extração automática, em que o ápice da folha pode ser difícil de distinguir da ponta de um lóbulo. Além disso, mesmo o comprimento de uma folha pode ser difícil de medir se a folha for assimétrica e a veia principal não se alinha com o eixo primário. Por estas razões, estudos envolvendo *landmarks* e medidas lineares muitas vezes envolvem a extração manual de dados por botânicos especialistas.

Conexão de polígonos e dimensão de fractais, envolvem um numero real capaz de representar o quão completamente uma forma enche o espaço dimensional ao qual pertence, isso pode fornecer uma medida útil da complexidade de uma forma, podendo ser considerada uma característica de entrada para um classificador. Um dos principais métodos para calcular dimensões fractais é o método de Minkowski-Bouligand devido a sua precisão e também por possuir uma versão de múltipla escala.

McLellan e Endler (1998) mostraram que a dimensão fractal tende a estar altamente correlacionada com a relação perímetro/área, sugerindo que é um benefício limitado. Dada a grande variedade de formas possíveis para uma folha, caracterizar esta forma por qualquer medida de complexidade única pode descartar muitas informações úteis, sugerindo que medidas de dimensão fractal possam ser úteis em combinação com outros recursos. Du et al. (2006) criaram representações poligonais de folhas e usaram estas para realizar as comparações.

### 2.1.2 EXTRAÇÃO E ANÁLISE DA VENAÇÃO DAS FOLHAS

A extração e análise de como as veias da folha estão estruturadas também são alguns dos aspectos mais estudados. Como mostra a Figura 5, existem diversos tipos de estruturas de venação. As veias de uma folha fornecem uma estrutura do mecanismo de transporte de água, minerais, e açúcares etc. Embora os detalhes finos, veias secundárias, possam variar em uma folha, encontrar um padrão nas veias

maiores com o tratamento certo pode ajudar a identificar uma planta.

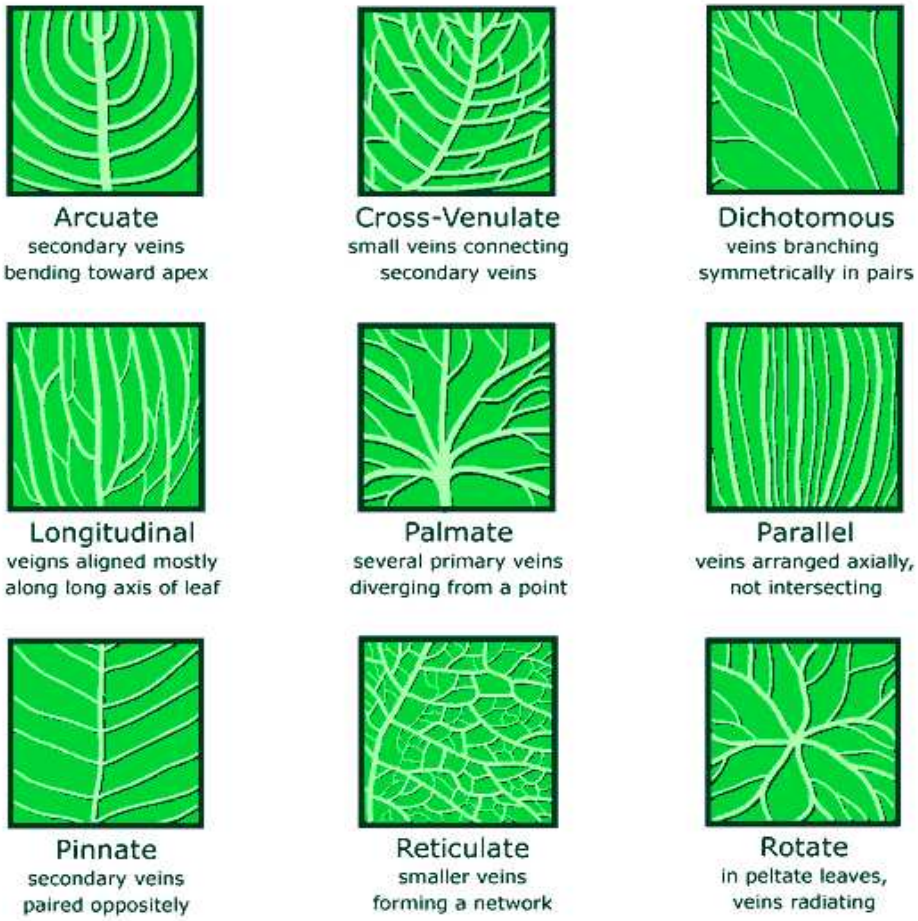


**Figura 5 – Alguns dos principais tipos de venação**
**Fonte: LIVEEARTH (2009, p.s.n.)**

Cope et al. (2010) usaram algoritmos genéticos robustos em classificadores para identificação de *pixels* de veias. Obtendo uma taxa aceitável de acertos ao reconhecer a venação primária e secundária, extraíram com êxito a venação de uma pequena imagem da folha utilizando Análise de Componentes Independentes (ICA). Embora, quando o mesmo teste foi realizado na folha inteira, os resultados não foram melhores que os algoritmos de detecção de bordas. Um dos melhores resultados na utilização da estrutura de venação da folha foi alcançado por Fu e Chi (2006) usando uma abordagem combinada de *thresholding* e redes neurais. Para conseguir obter tal resultado as folhas fotografadas foram expostas a um banco de luz fluorescente para melhorar a visualização da venação.

### 2.1.3 ANÁLISE DA MARGEM DAS FOLHAS

A margem das folhas, ou seja, a borda externa da lâmina, pode conter porções serrilhadas, distintas dos lóbulos, maiores e mais suaves, entre outras formas como mostra a Figura 6. Apesar de ser uma característica útil para os botânicos poderem descrever as folhas, a margem tem um uso muito pequeno na análise automatizada de folhas, porque os dentes serrilhados não estão presentes em todas as espécies de plantas. Na verdade, foi afirmado que nenhum algoritmo de computador pode detectar com segurança os dentes das folhas (ROYER et al., 2005).

No entanto, os dentes são uma característica muito importante de várias espécies, pois de acordo com o tamanho e o número de dentes pode-se estimar o clima e os padrões de crescimento (ROYER; WILF, 2006), e podem também ser utilizados para fazer inferências sobre climas pré-históricos fazendo-se uso de folhas fossilizadas (ELLIS et al., 2009).

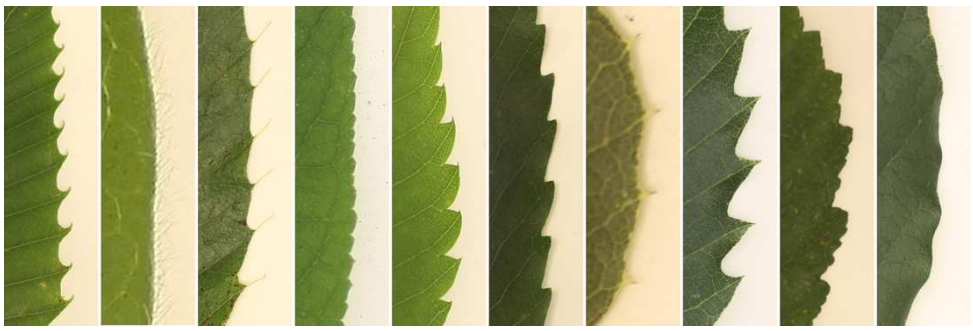

**Figura 6 – Diferentes variedades de margem de folhas**
**Fonte: Cope et al. (2010)**

Estudos usando a margem da folha normalmente são combinados com outras características e medidas. Uma possibilidade para melhorar esse tipo de análise, é a combinação com a análise de venação, pois os dentes muitas vezes têm veias pequenas correndo para suas pontas. Margens que estão quebradas ou danificadas por insetos podem parecer superficialmente como dentes, mas são menos propensos a ter os mesmos padrões de veias.

### 2.1.4 ANÁLISE DA TEXTURA DAS FOLHAS

A textura é um dos principais atributos visuais para métodos e aplicações que envolvem análise de imagens e visão computacional. A textura representa o aspecto de uma superfície, e em imagens digitais, características as quais podem ser descritas por variações locais em valores de *pixels* que se repetem de maneira regular

ou aleatória ao longo do objeto ou imagem. Portanto a textura pode ser considerada um agrupamento de similaridade em uma imagem (ROSENFELD; KAK, 1982), e as propriedades destes agrupamentos locais podem causar efeitos perceptivos como uniformidade, densidade, aspereza, regularidade, linearidade, frequência, fase, direcionalidade, suavidade entre outros efeitos (LEVINE, 1985). Dentre os métodos conhecidos para análise computacional de textura, estão os descritores de Haralick que fazem uso de Matrizes de Coocorrência.

A Figura 7 representa cinco máscaras em tons de cinza retiradas de uma única folha de jabuticabeira para análise de textura. Coocorrência, na sua forma geral, pode ser especificada por uma matriz de frequências relativas $P(i, j, d, \theta)$ na qual dois elementos de textura vizinhos, separados por uma distância $d$ em uma orientação que ocorrem na imagem, um com propriedade $i$ e outro com propriedade $j$ (SCHWARTZ; PEDRINI, 2003). Inicialmente foram propostos 14 medidas de textura baseadas nos dados retirados da matriz de coocorrência.

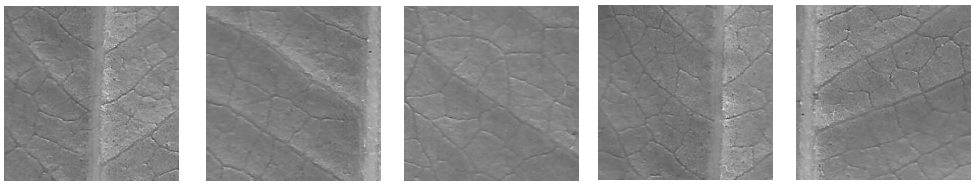

**Figura 7 – Amostras de máscaras para análise de textura por Haralick**
**Fonte: Autoria própria (2017).**

Entre outros métodos estão os de Casanova et al. (2009), que usaram uma série de filtros *Gabor* em um conjunto de dados maior, calculando a energia para a resposta de cada filtro aplicado, obtendo resultados razoáveis, enquanto Liu et al. (2009) apresentaram um método baseado em transformadas *Wavelet* e máquina de vetores de suporte (SVM) obtendo resultados bastante razoáveis.

# 3 REDES NEURAIS

Redes Neurais Artificiais (RNA) são modelos computacionais inspirados em reproduzir o modelo do cérebro humano, obtendo capacidade de adquirir, armazenar e utilizar conhecimento experimental.

Segundo Haykin (2004) redes neurais podem ser definidas como sendo um processador maciçamente paralelo e distribuído constituído de unidades de processamento simples, que tem a propensão natural para armazenar conhecimento experimental e torná-lo disponível para uso. Assemelhando-se ao cérebro, no qual o conhecimento é adquirido pela rede em um processo de aprendizado e as fortes ligações entre neurônios, conhecidos como pesos sinápticos, são utilizados para armazenar o conhecimento.

## 3.1 MODELO DE UM NEURÔNIO

Os neurônios são os blocos básicos de construção em uma rede neural seja ela uma Rede Neural Biológica (RNB) ou uma RNA. Na Figura 8 pode-se observar de maneira resumida as diferentes partes de um neurônio biológico. Nota-se que as dendrites são responsáveis por receber o sinal de outros neurônios, conduzindo-os até o corpo da célula, o qual irá somar todos os sinais de entrada recebidos pelas dendrites. Após a somatória dos sinais de entrada atingir um limiar, o sinal viaja através do axônio para atingir outros neurônios.

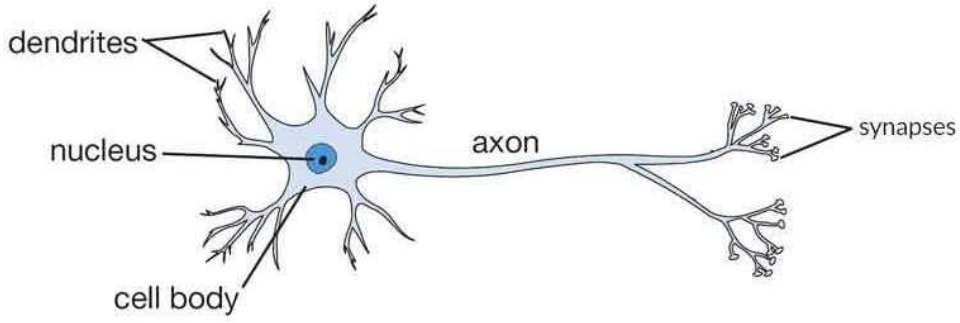


**Figura 8 – Componentes da estrutura de um neurônio biológico**
**Fonte: Jagreet (2017)**

É importante notar que um neurônio sozinho é basicamente inútil, pois o po-

der está na quantidade de neurônios interligados entre si, capazes de construir uma rede para solucionar determinado problema. A Figura 9 apresenta o modelo de uma unidade de processamento, ou seja, um neurônio, responsável por constituir a base de uma RNA.

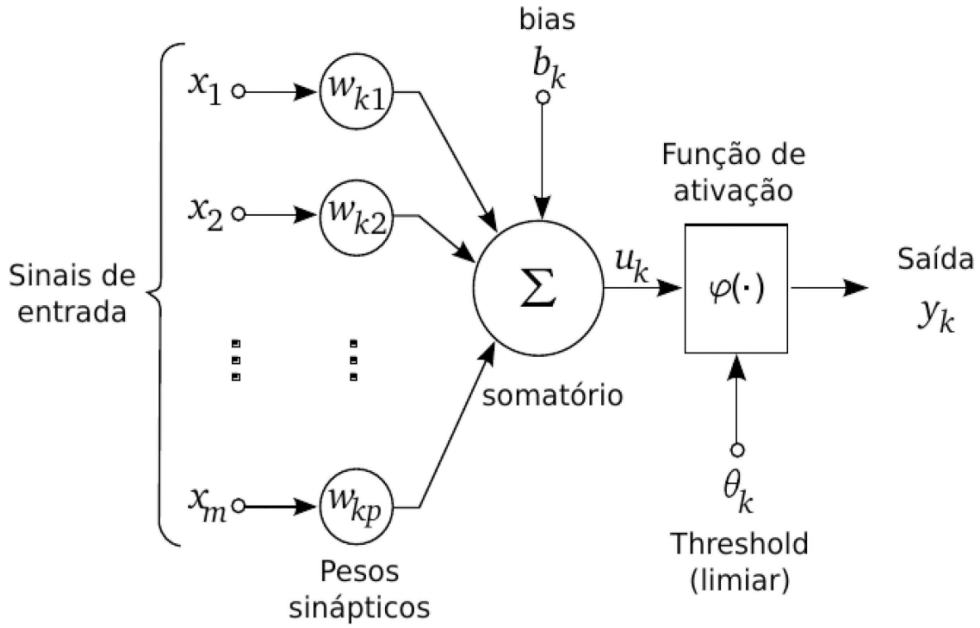


**Figura 9 – Modelo simplificado de um neurônio com $x_m$ entradas**
**Fonte: Adaptado de Haykin (2004)**

O neurônio da Figura 9 é composto por três principais partes, assim como o neurônio biológico citado na Figura 8:

(i) Um conjunto de sinapses, cada uma caracterizada por um peso próprio. Especificamente, um sinal $x_j$ na entrada da sinapse $j$ conectada ao neurônio $k$ é multiplicado pelo peso sináptico $w_{kj}$, no qual o primeiro subíndice se refere ao neurônio e o segundo se refere ao terminal de entrada da sinapse à qual o peso se refere. Por analogia à RNB, esse processo de multiplicação do sinal de entrada $x_1$ e continuação do sinal até o corpo da célula caracteriza o funcionamento das dendrites.

(ii) Um somador que por analogia à RNB faz parte do corpo da célula, responsável por somar os sinais de entrada, ponderados pelas respectivas sinapses do neurônio.

(iii) Uma função de ativação, que também constitui parte da célula em uma RNB, tem como objetivo restringir a amplitude da saída de um neurônio.

O modelo da Figura 9 inclui a utilização de um *bias* aplicado externamente,

representado por $b_k$, com a finalidade de aumentar ou diminuir a entrada da função de ativação, dependendo se ele é positivo ou negativo, respectivamente.

Em termos matemáticos, o sinal $u_k$ que serve de entrada para a função de ativação pode ser descrito como:

$$u_k = \sum_{j=1}^{m} w_{kj} * x_j \tag{1}$$

E a saída de um neurônio dada por $y_k$:

$$y_k = \varphi(u_k + b_k) \tag{2}$$

Onde pode-se notar na Figura 9 que $x_1, x_2, ..., x_m$ são os sinais de entrada e $w_{k1}, w_{k2}, ..., w_{km}$ são os respectivos pesos do neurônio $k$. Para simplificar a Equação (2), e estabelecer a relação potencial de ativação $v_k$ do neurônio $k$ temos que $v_k = u_k + b_k$, portanto :

$$y_k = \varphi(v_k) \tag{3}$$

## 3.2 FUNÇÕES DE ATIVAÇÃO

A função de ativação é responsável pelo processamento de cada neurônio. A escolha de uma determinada função de ativação de uma rede neural é baseada nos dados de entrada da rede neural e uma determinada entrada funciona melhor com determinada função. Segundo Haykin (2004) as funções de ativação estão divididas em três tipos básicos:

- Função de limiar, ou função *Heaviside*. O primeiro modelo de neurônio a utilizar essa função ficou conhecido como neurônio de McCulloch e Pitts (1943). Neste modelo, a saída de um neurônio assume valor $1$ se o campo local induzido também chamado de potencial de ativação é positivo, e $0$ caso contrário.

$$\varphi(v) = \begin{cases} 1 \text{ se } v \geq 0 \\ 0 \text{ se } v < 0 \end{cases} \tag{4}$$

- Função de ativação linear por partes, é uma função real definida nos números reais ou em um segmento, cujo gráfico é composto de seções em linha ret

$$\varphi(v) = \begin{cases} 1, & v \geq 0,5 \\ v, & 0,5 > v > -0,5 \\ 0, & v \leq -0,5 \end{cases} \tag{5}$$

- Função de ativação sigmóide é, de longe, a forma mais usada em RNAs. É definida como uma função estritamente crescente que exibe um balanceamento adequado entre comportamento linear e não-linear. Um exemplo de função sigmóide é a função logística definida por:

$$\varphi(v) = \frac{1}{1 + e^{-av}} \tag{6}$$

onde $a$ é o parâmetro de inclinação da função sigmóide, ou seja variando-se o parâmetro $a$, obtêm-se sigmóides com diferentes inclinações.

Uma representação gráfica das funções citadas é mostrada na Figura 10.

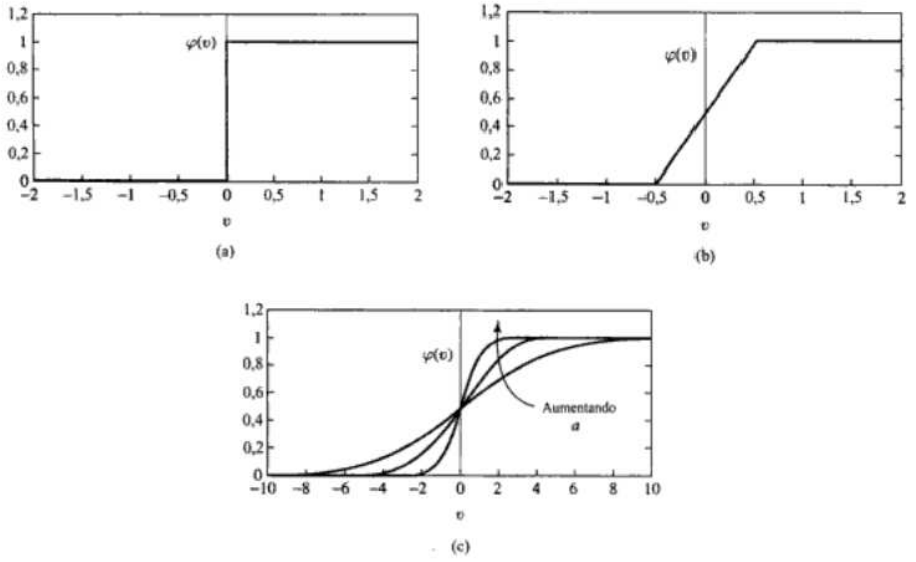


**Figura 10 – (a)Função de limiar (b)Função linear por partes (c)Função sigmóide**
**Fonte: Adaptado de Haykin (2004)**

## 3.3 REDES *PERCEPTRON*

Uma rede neural com uma camada somente é chamada de *single layer feed-forward neural network*, também conhecida como *Perceptron*. Rosenblatt (1958) foi responsável por propor o modelo *Perceptron* como primeiro modelo para aprendizagem supervisionada, ou seja, sua ideia era criar uma rede que pudesse aprender e se

ajustar sozinha.

A simplicidade da rede *Perceptron* está no fato de ser constituída por apenas uma camada neural. Por ser a forma mais simples de uma rede neural, essa arquitetura é usada usada para classificação de padrões ditos linearmente separáveis (HAYKIN, 2004), no qual as classes possíveis de classificação variam de acordo com o número de neurônios encontrados na camada de saída.

Porém, um dos problemas que a rede *Perceptron* não consegue tratar é o problema do $OU$ exclusivo como mostrado por (MINSKY; PAPERT, 1969). O motivo que torna o problema do $OU$ exclusivo impossível de ser resolvido é que as duas classes que precisam ser distinguidas não são linearmente separáveis como pode ser visto no exemplo da Figura 11, na qual não se pode traçar uma linha reta que separe em duas dimensões os pontos pretos dos pontos brancos.

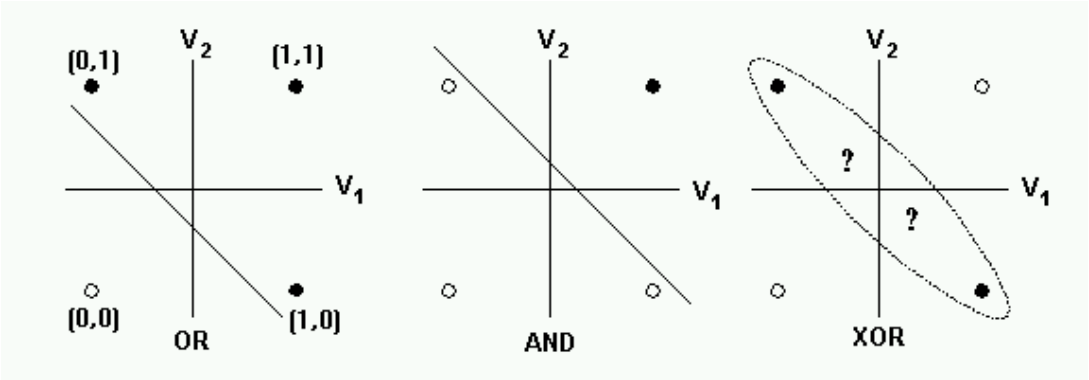


**Figura 11 – Problema não-linear em uma Rede *Perceptron***
**Fonte: Beeman (2001)**

## 3.4 REDES *PERCEPTRON* MULTICAMADAS

As Redes *Perceptron* Multicamadas (*Multilayer Perceptron* - MLP), são redes do tipo *feedforward* caracterizadas por conter mais de uma camada intermediária, ou também chamadas de camadas escondidas (*hidden layers*), situadas entre a camada de entrada e a respectiva camada neural de saída (NUNES et al., 2010). A Figura 12 apresenta a estrutura em grafo de uma arquitetura MLP totalmente conectada e com duas camadas escondidas.

Com a adição das camadas extras, a rede MLP engloba todos os problemas solucionados pela rede *Perceptron* clássica, como também os problemas de não-linearidade pelas quais as redes Perceptron eram limitadas.

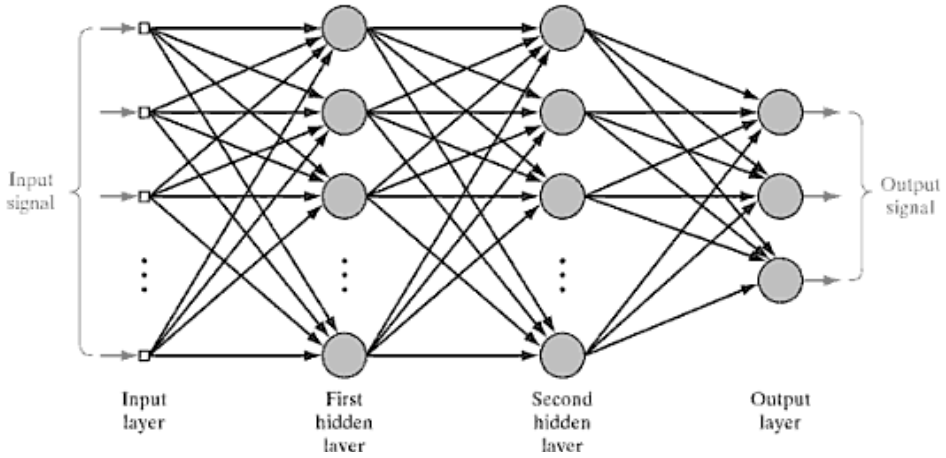


**Figura 12 – Rede MLP com duas camadas escondidas**
**Fonte: Adaptado de Haykin (2004)**

### 3.4.1 BACKPROPAGATION

As redes MLP utilizam um algoritmo de aprendizado supervisionado para realizar seu treinamento chamado *backpropagation*. O objetivo do *backpropagation* é otimizar os pesos para que a rede neural possa aprender a mapear corretamente as entradas arbitrárias para as respectivas saídas. O aprendizado ocorre em duas etapas, a fase *forward* e a fase *backward*.

Na fase *forward*, os sinais de entrada percorrem toda a rede em direção a camada de saída sendo multiplicados pelos seus respectivos pesos, os quais são iniciados aleatoriamente com valores entre 0 e 1. Para cada neurônio somam-se os produtos das entradas com seu conjunto correspondente de pesos como mostra a Equação (7):

$$net_j = \sum_{i=1}^{n} x_i w_{ji} + \theta_i \tag{7}$$

Sendo que a constante $n$ representa o número de conexões na entrada do neurônio $j$, $w_{ji}$ representa o peso da conexão entre a entrada $x_i$ e o nerônio $j$ e $\theta_i$ é o peso do *bias* (CASANOVA, 2008).

Ao resultado dessa somatória é aplicada a função de ativação, isto ocorre até que um sinal de saída seja apresentado na última camada. O sinal de saída pode ser dado pela Equação (8).

$$y_i = \varphi(net_j) \tag{8}$$

Tendo calculado a saída da rede, deve-se calcular o erro de saída da rede para então dar inicio a fase de retro-propagação, a fase *backward*. Uma das maneiras de calcular o erro é dado pela Equação (9).

$$E_{total} = \frac{1}{2}(target - output)^2 \tag{9}$$

Na qual a variável *output* representa a saída da rede MLP, ou seja a saída de toda a CNN e a variável *target* representa a resposta desejada, o rotulo de saída a ser alcançado pela rede.

Calculado o erro, tem-se início então a fase *backward*, o erro é retro-propagado a fim de atualizar o peso das suas sinapses, para que ocorra a aproximação dos valores *target* e *output* minimizando assim o erro de cada neurônio de saída e da rede como um todo.

O objetivo nesta fase portanto é minimizar o erro variando o peso. O que pode ser determinado pela Equação (10), sendo $\eta$ denominado como taxa de aprendizagem. A taxa de aprendizado é um hiperparâmetro que controla o quanto os pesos da rede serão ajustados, $\eta$ deve ser um valor positivo entre 0 e 1. A escolha desse valor implica na velocidade com que a rede irá convergir, se o valor da taxa de aprendizado for muito grande, o algoritmo excederá o custo global mínimo e se for muito pequeno, o algoritmo necessitará de mais épocas até a convergência, o que pode tornar o aprendizado lento.

$$\Delta w_{ij} = -\eta \frac{\partial E_{total}}{\partial net_j} \tag{10}$$

Porém, o erro dado por $\partial E_{total}$ depende da saída do neurônio $net_j$, o qual depende do peso das sinapses dada por $\partial w_{ij}$, portanto faz-se necessária a utilização da regra da cadeia, podendo expandir a Equação (10) em:

$$\Delta w_{ij} = -\eta \frac{\partial E_{total}}{\partial y_j} \frac{\partial y_j}{net_j} \frac{\partial net_j}{\partial w_{ij}} \tag{11}$$

A Figura 13 exemplifica o comportamento da Equação (11). Onde $out_{01}$ equivale a $y_i$ e $w5$ representa uma sinapse dada por $w_{ij}$. Portanto o $\partial E_{total}$ será derivado em relação a saída do neurônio $y_i$ que por sua vez será derivado em relação a saída da função de ativação dada por $net_j$ que por fim será derivada em relação ao peso do neurônio dado por $w_{ij}$.

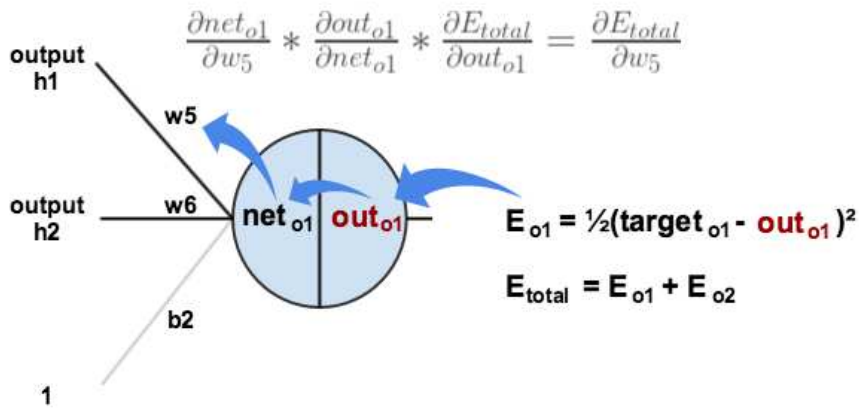


**Figura 13 – Aplicação do comportamento do *backpropagation***
**Fonte: Adaptado de Mazur (2015)**

De maneira simplificada, a Equação (11) pode ser escrita como sendo:

$$\Delta w_{ij} = -\eta e_j x_i \tag{12}$$

Onde $x_i$ é o resultado da derivada $\frac{\partial net_j}{\partial wij}$, e $e_j$ o erro, que depende em qual camada o neurônio esta localizado.

Se $j \ni$ a camada de saída, tem-se:

$$e_j = F'(net_j)(target_j - output_j) \tag{13}$$

E se $j \ni$ a camada escondida:

$$e_j = F'(net_j) \sum_k (target_j - output_j) \tag{14}$$

# 4 DEEP LEARNING

A partir do conceito de *Deep Learning* novas propostas foram desenvolvidas, originando as chamadas redes neurais profundas, como as Redes Neurais Convolucionais (CNNs).

## 4.1 REDES NEURAIS CONVOLUCIONAIS

As CNNs, também conhecidas como ConvNets, fazem parte de uma área do aprendizado profundo que teve sua inspiração baseada no córtex visual, voltada para o desenvolvimento do campo de visão computacional, para classificação de imagens.

O processo de classificação consiste em dada uma imagem de entrada, a saída seja a classe a qual a imagem de entrada pertence ou a probabilidade de pertencer a determinada classe. Para humanos, a tarefa de classificar imagens é relativamente fácil, e também uma das primeiras habilidades a ser aprendida, sem mencionar que mantém seu desenvolvimento mesmo após tornarem-se adultos, permitindo que se possa identificar e classificar uma imagem ou objeto sem pensar duas vezes. Entretanto para uma máquina essa tarefa não é tão fácil.

Quando um computador vê uma imagem, nada mais é do que uma matriz de pixels variando seus valores entre 0 a 255, possuindo três camadas quando tratando-se de imagens RGB, uma camada para cada canal de cor e podendo variar o tamanho desta matriz de acordo com o tamanho da imagem de entrada. Para uma CNN, uma imagem com H linhas, W colunas e três canais (R,G e B respectivamente) também podem ser descrita como sendo um tensor de ordem 3, ou ordem 2 se a imagem estiver em escalas de cinza (WU, 2017).

Os tensores são objetos geométricos que descrevem relações lineares entre vetores geométricos, escalares e outros tensores, ou seja, um tensor consiste em um conjunto de valores primitivos moldados em uma matriz de qualquer número de dimensões. Em termos simples, um tensor é apenas um *array* multidimensional. Um tensor pode consistir de um único número, caso em que é referido como um tensor de ordem zero, ou simplesmente um escalar. Um escalar pode ser considerado como uma matriz de dimensão zero (igual à ordem do tensor).

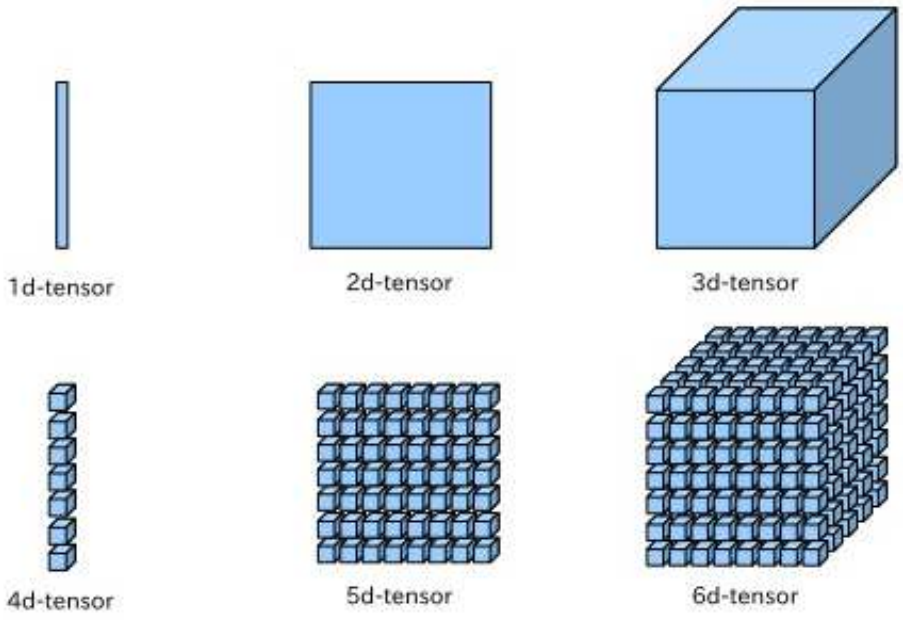


**Figura 14 – Ordem dos tensores**
**Fonte: Akshansh (2017, p.s.n.)**

Em uma arquitetura CNN, a imagem de entrada percorre uma série de processamentos (cada um desses processamentos pode ser dito como uma camada) a fim de obter uma saída que possa permitir a classificação da imagem. Existem diversos tipos de camadas em uma CNN, entre elas as camadas mais básicas são: camadas de convolução, camadas de agrupamento (*pooling*) e camadas totalmente conectadas. Os primeiros estágios presentes em uma rede neural convolucional, ou seja, seus primeiros *layers* convolucionais e de *pooling*, são tidos como os mais importantes da rede e são responsáveis pela maior parte do processamento computacional (BRANDAO et al., 2005).

Segundo (WU, 2017), a última camada é a camada de perda. Supondo $t$ como objetivo correspondente a entrada $x^i$, então o custo ou a função perda pode ser usada para medir a discrepância entre a predição $x^l$ da CNN e o objetivo $t$. Por exemplo, uma simples função de perda pode ser descrita como:

$$z = \frac{1}{2}||t - x^l||^2 \tag{15}$$

A equação (15) é comumente utilizada para problemas de regressão. Para problemas envolvendo classificação de imagens, a função perda frequentemente utilizada é a *cross-entropy* dada por:

$$H(p,q) = -\sum_{x} p(x) \log q(x) \tag{16}$$

A entropia cruzada é comumente usada para quantificar a diferença entre duas distribuições de probabilidade. Normalmente, a distribuição "verdadeira"dada por $p$ e a distribuição prevista pelo modelo dada $q$.

## 4.2 CAMADAS CONVOLUCIONAIS

Para compreender plenamente o funcionamento das redes neurais convolucionais é necessário primeiro ter uma noção sobre operações de convolução.

Uma convolução discreta ou simplesmente convolução é uma operação fundamental nas redes neurais convolucionais, assim sendo de obrigatória importância entender como essa operação funciona. Para explicar o funcionamento da operação de convolução pode-se supor a existência de dois vetores $x$ e $w$ e $y$ denotado como sendo a convolução entre eles, ou seja:

$$y = x * w \tag{17}$$

Onde o vetor $x$ é a entrada da camada convolucional, muitas vezes chamado de sinal e $w$ o filtro de características ou *kernel*. Deve-se manter atenção que a operação indicada na Equação 17 pelo símbolo $*$ não indica o operador de multiplicação escalar como usando em linguagens de programação, e sim a operação de convolução entre os dois vetores $x$ e $w$. A convolução em sua forma geral pode ser descrita da seguinte maneira:

$$y = x * w : y[i] = \sum_{k=-\infty}^{+\infty} x[i-k]w[k] \tag{18}$$

Porem é necessário esclarecer dois conceitos básicos: os índices -$\infty$ até +$\infty$ para o vetor $x$. Em aplicações de *Machine Learning* trabalha-se sempre com vetores de características finitos. Por exemplo, se o vetor de entrada $x$ possui 10 características, terá seus índices de maneira tal a ser 0,1,2,...,9, então os índices -$\infty$:-1 e +$\infty$:10 estão fora dos limites do sinal $x$. Além disso, para calcular corretamente a soma assume-se que os vetores $x$ e $w$ estão preenchidos com zeros. Resultando em um vetor de saída $y$ de tamanho infinito preenchido com inúmeros zeros.

Como essa abordagem não é útil na prática, o sinal $x$ é preenchido com um

número finito de zeros. Esse processo denomina-se *Zero-padding* ou *Padding*. Um exemplo desta abordagem aplicado a um vetor unidimensional pode ser visto na Figura 15. No qual o *padding*, denotado por $p$ é igual a 2.

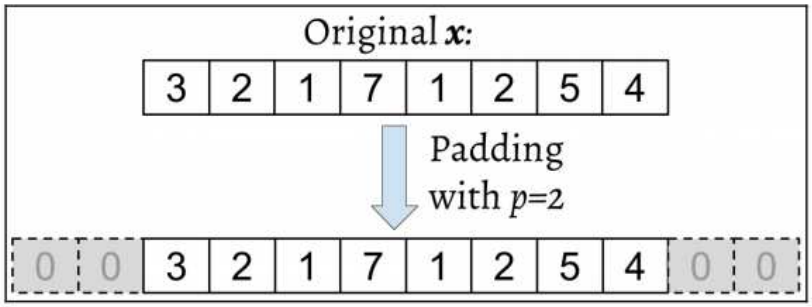


**Figura 15 – *Padding* de tamanho 2**
**Fonte: Raschka e Mirjalili (2017)**

Os vetores de entrada $x$ e filtro $w$ possuem tamanho $n$ e $m$ respectivamente, sendo $m$<$n$. Além disso o preenchimento do vetor $\mathbf{x}^p$ possui tamanho $n + 2p$. Então, a fórmula para calcular a convolução pode ser transformada da seguinte maneira:

$$y = x * w \rightarrow y[i] = \sum_{k=0}^{k=m-1} x^p[i + m - k]w[k] \tag{19}$$

O segundo conceito é referende ao indexamento de $x$ com $i + m - k$. É importante notar que $x$ e $w$ são indexados em diferentes direções neste somatório, portanto pode-se inverter um desses vetores após realizado o procedimento de preenchimento para então computar o produto entre eles. Seguindo o exemplo da Figura 16, no qual inverte-se o filtro $w$ e calcula-se o produto entre eles com um deslocamento de dois índices. O procedimento continua até que todo o sinal de entrada seja percorrido.

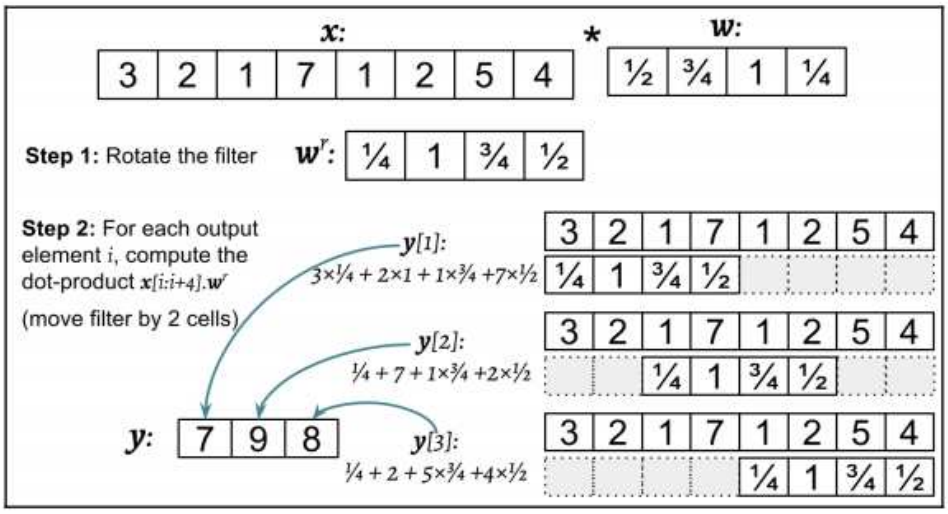


**Figura 16 – Convolução unidimensional**
**Fonte: Raschka e Mirjalili (2017)**

Analisando atentamente, nota-se que o preenchimento no exemplo é zero, além disso o deslocamento do filtro em relação ao sinal de entrada $x$ de duas unidades, recebe o nome de *stride*, considerado um dos hiperparâmetros da convolução. O *stride*, obrigatoriamente precisa ser positivo e menor do que o tamanho do vetor de entrada.

Agora, com o conceito das operações de convolução, parte-se para a abordagem das camadas convolucionais. As camadas convolucionais são compostas por um número "$n$"de filtros, com pesos previamente inicializados que serão ajustados através do método de *backpropagation* conforme a rede aprende a identificar regiões significantes a fim de minimizar a função custo.

Filtros geralmente são matrizes pequenas como, por exemplo, matrizes 5x5 nos 3 canais (R, G e B) de cor, composta por valores reais que serão convoluídos com os dados de entrada para obter um mapa de características, equivalente ao $y$ do exemplo anterior. Estes mapas indicam regiões na qual as características extraídas pelos filtros se encontram.

Os filtros, também chamados de campos receptivos ou detectores de características aplicados na camada de convolução, possuem dois parâmetros principais, importados da convolução unidimensional de dois vetores, que podem mudar o comportamento de cada camada. Como mostrado na Figura 17, os parâmetros que podem ser pré-determinados são o preenchimento da imagem e o passo com o qual o filtro irá percorrer toda a imagem de entrada.

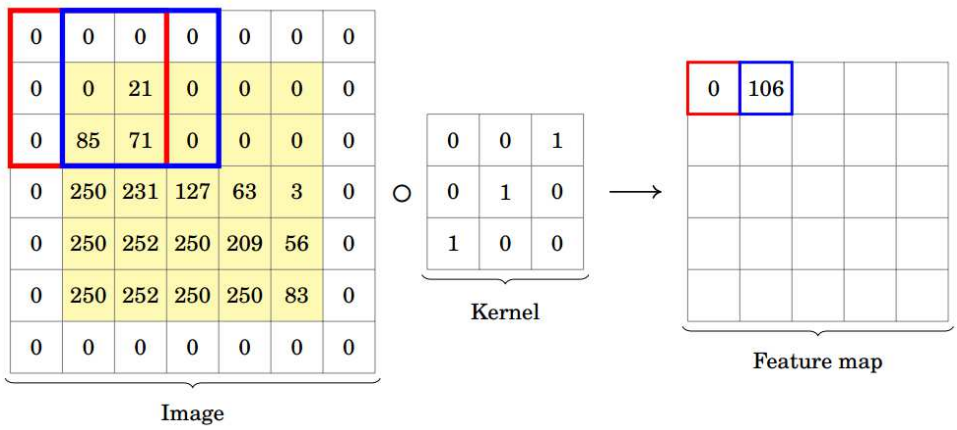


**Figura 17 – Camada convolucional parcial, utilizando um filtro 3x3**
**Fonte: Pavlovsky (2017)**

O tamanho da matriz resultante da camada de convolução será também a entrada da próxima camada da CNN pode ser obtido através da equação (21):

$$O = \left\lfloor \frac{(n + 2p - m)}{S} + 1 \right\rfloor \tag{20}$$

Onde:

- $O$ é o tamanho da matriz de saída em linhas $x$ colunas.
- $n$ é o tamanho da imagem de entrada em linhas $x$ colunas.
- $m$ é o tamanho do filtro.
- $p$ é o preenchimento.
- $S$ é o *stride*/passo.

E a função piso denotada por $\lfloor . \rfloor$ retorna o maior inteiro igual ou menor que a entrada. Exemplo:

$$floor(1.77) = \lfloor 1.77 \rfloor = 1 \tag{21}$$

A aplicação dos detectores de características pode levar a perda de algumas informações, pois a matriz resultante apresenta um número menor de valores como mostra a Equação (21), mas ao mesmo tempo o proposito é detectar certas características, as partes da imagem mais significantes. Portanto, muitos mapas de características são criados na primeira camada de convolução a fim de preservar um maior

número de informação. De forma simplificada, a camada de convolução serve para informar a rede aonde estão localizadas as características importantes da imagem.

Ainda na camada de convolução, é comum encontrar uma sub-camada de ativação conectada aos mapas de características. Nas literaturas recentes, para CNNs vêm sendo usado três possíveis tipos de função de ativação, sendo elas a função *sigmoid*, que transforma valores de entrada em um intervalo de 0 a 1; a função de unidade linear retificada (ReLU), que transforma valores negativos em 0 e mantém os valores positivos sem nenhuma modificação e ReLU parametrizado (PReLU), que mapeia os maiores valores negativos em valores menores reduzindo a inclinação da função de mapeamento (KUO, 2016). Dentre as funções de ativação citadas, a função ReLU é a mais usada para CNNS, que simplesmente aplica a função:

$$y_{i,j,d} = max(0, x^{l}_{i,j,d}) \tag{22}$$

em cada elemento resultante da matriz de convolução. Onde $y_{i,j,d}$ é o valor do pixel no mapa de características na posição $i, j$ no canal $d$.

Ao realizar uma operação matemática como a convolução, aplicando detectores de características para gerar mapas de ativação, arrisca-se criar uma saída linear, portanto a função da camada ReLU conectada à camada de convolução é evitar que isso aconteça, adicionando uma não linearidade para a saída da convolução. A Figura 18 exemplifica de que forma é composta toda a camada convolucional.

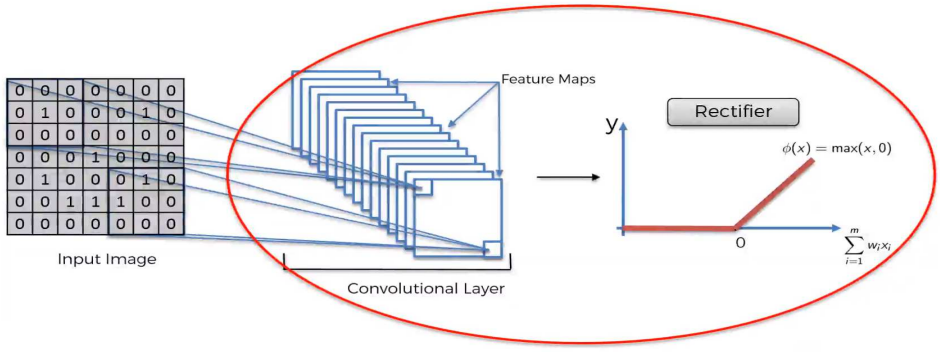


**Figura 18 – Exemplo genérico de camada convolucional completa**
**Fonte: SuperDataScience (2017)**

## 4.3 CAMADAS DE AGRUPAMENTO (*POOLING*)

A segunda camada em uma CNN é a camada de *pooling*. As camadas de *pooling* são uma forma de *down-sampling*. Esta técnica é utilizada com o objetivo de

reduzir o tamanho espacial das matrizes resultantes da convolução, reduzindo assim os parâmetros a serem aprendidos na rede, contribuindo para o controle de *overfitting*. Tipicamente uma camada de *pooling* é como um filtro de tamanho e passo previamente determinado que computa qual é o máximo local de uma determinada região do mapa de atributos. A Figura 19 apresenta uma camada de *pooling* de tamanho $2$x$2$ com um passo de $2$.

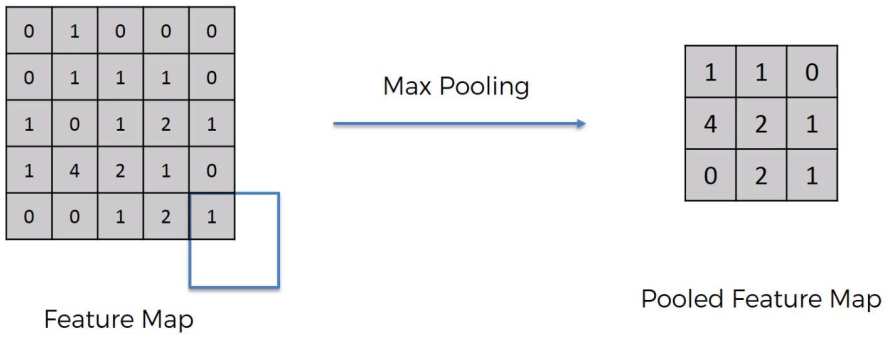


**Figura 19 – Camada de *pooling*, utilizando um kernel 2x2**
**Fonte: (SUPERDATASCIENCE, 2017)**

A operação de *pooling* não requer que nenhum parâmetro seja passado, assumimos que $H$ divide $H^l$, onde $H^l$ é a altura da matriz do *layer* atual e $W$ divide $W^l$, onde $W^l$ é a largura da matriz *layer* atual, ou seja a saída da camada de convolução é a entrada para a camada de *pooling*, a saída do *pooling* irá resultar em um tensor de ordem 3 de tamanho $H^{l+1}$x$W^l$x$D^{l+1}$ sendo:

$$H^{l+1} = \frac{H^l}{H}, \qquad W^{l+1} = \frac{W^l}{W}, \qquad D^{l+1} = D^l \tag{23}$$

Existem dois tipos de operações de *pooling* amplamente usadas: *max pooling* e *mean-pooling* (também conhecida como *average pooling*). A função *max pooling* parte da análise do maior valor de uma região (WU, 2017), pois os valores máximos no mapa de caracteristicas representam aonde estão localizadas as caracteristicas, e ao aplicar a camada de *pooling* uma porcentagem de informação que não é característica é descartada. A matriz (imagem) resultante pode ser chamado como mapa de características agrupados (*pooled feature map*). A função *average-pooling* por sua vez é responsável por computar o valor médio de uma determinada região limitada pelo filtro.

Modelos de arquitetura CNN normalmente possuem inúmeros estágios de

convolução, não-linearidade e camadas de *pooling* seguidos por mais camadas de convolução e camadas totalmente conectadas.

## 4.4 CAMADAS TOTALMENTE CONECTADAS (*FULLY CONNECTED*)

Inicialmente, antes de chegar até as camadas totalmente conectadas é necessário realizar a conversão das matrizes de caracteristicas resultantes da camada de *Pooling* em um vetor coluna o qual servirá de entrada para a ANN clássica como mostrado na Figura 20.

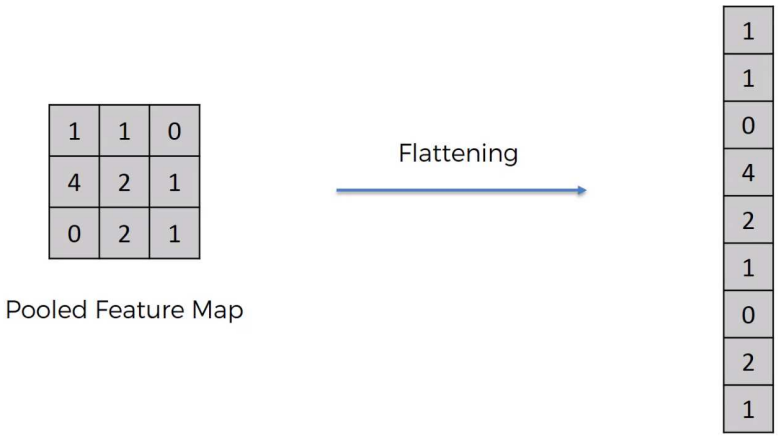


**Figura 20 – Operação de *flattening*, dado um mapa de características**
**Fonte: SuperDataScience (2017)**

As camadas totalmente conectadas, tipicamente são utilizadas como camadas finais, onde diferente das camadas convolucionais, todos os pesos são totalmente conectados com a camada anterior.

Aqui, toda uma rede neural artificial clássica, ou seja, toda uma ANN é adicionada ao final da CNN, na qual cada elemento do vetor coluna criado pela camada de *flattening* se comporta como uma entrada para uma ANN clássica, para em seguida serem ligados às camadas escondidas totalmente conectadas até a camada final, como mostra a Figura 21.

O propósito de uma ANN após a uma operação de *flattening* é combinar as características em mais atributos, para melhorar o desempenho de previsão das classes. O erro é calculado e propagado de volta como em uma ANN comum e alguns parâmetros como: os pesos da ANN (sinapses) e os filtros (detectores de características) são ajustados para ajudar a otimizar o desempenho. Inicialmente, os filtros procuram por determinadas características, mas tais características podem não estar

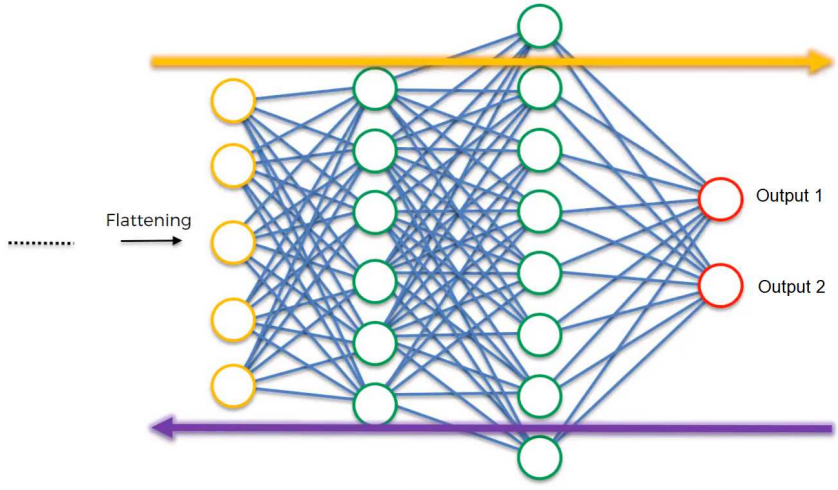


**Figura 21 – Camada totalmente conectada, é uma ANN clássica**
**Fonte: SuperDataScience (2017)**

corretas, então os filtros são ajustados para procurarem por novas características.

Para finalizar, na Figura 22 pode-se observar uma Rede Neural Convolucional completa desde sua imagem de entrada como a classe de saída, que no exemplo mostrado possui somente 2 classes.

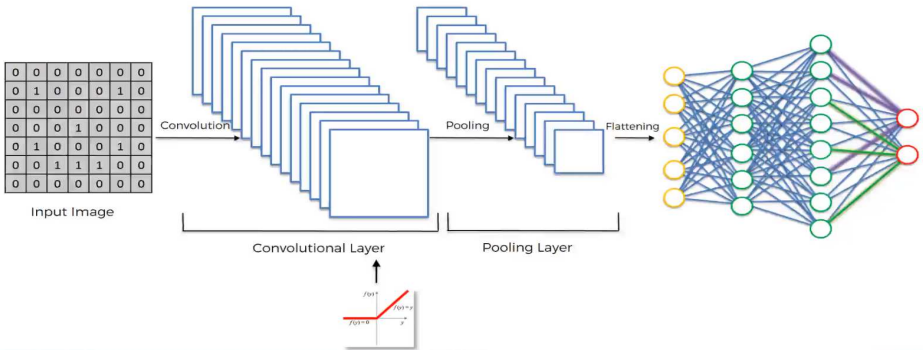


**Figura 22 – Arquitetura simplificada de uma Rede Neural Convolucional**
**Fonte: SuperDataScience (2017)**

## 4.5 MODELO VGGNET

Com as redes neurais convolucionais se tornando cada vez mais comuns no campo da visão computacional, várias tentativas foram feitas para melhorar a arquitetura original de Krizhevsky et al. (2012) em uma tentativa de obter melhor precisão.

A rede VGGNet foi proposta e desenvolvida por Simonyan e Zisserman (2014), recebendo este nome devido ao departamento da Universidade de Oxford chamado *Visual Geometry Group*. A rede VGG focava em uma abordagem relacionada a pro-

fundidade da rede, ou seja, aumentando o número de camadas escondidas presentes na rede. Seu desenvolvimento e implementação são derivados do framework Caffe (JIA et al., 2013), mas contém um número significativo de mudanças permitindo a execução do algoritmo em múltiplas $GPUs$ instaladas em um único sistema. Atualmente a rede VGG16 serve como um modelo pré-treinado utilizado como base para o desenvolvimento de outras arquiteturas.

Um modelo pré-treinado significa dizer que a rede foi previamente treinada em um conjunto de dados e contém os pesos e *biases* que representam as características de qualquer conjunto de dados em que foi treinado. As características aprendidas frequentemente podem ser transferidas para dados diferentes. Por exemplo, um modelo treinado em um grande conjunto de dados de imagens de aves conterá características aprendidas, como bordas ou linhas horizontais, nos quais podem ser transferidos para o conjunto de dados a ser treinado.

A arquitetura VGG, mostrada na Figura 23, demonstra que a rede foi treinada com diferentes números de camadas. Inicialmente começando com a arquitetura da coluna A, contendo 11 camadas (oito camadas de convolução e três camadas totalmente conectadas) e terminando na coluna E com 19 camadas (dezesseis camadas de convolução e três camadas totalmente conectadas).

Inicialmente durante a fase de treinamento todas as imagens, ou seja, as entradas da rede, foram setadas em um tamanho fixo de 224x224 pixels nos 3 canais RGB. A única parte referente ao pré-processamento realizado foi a subtração do valor médio RGB computado no conjunto de treinamento, de cada pixel da imagem em todos os canais.

A imagem então é passada através de uma pilha de camadas de convolução, onde são utilizados filtros de características, também chamados de filtros receptivos, de tamanho 3x3 (O qual é o menor tamanho necessário para capturar noções de direcionamento, ou seja, esquerda/direita, cima/baixo e centro).

O deslocamento dos filtros características na camada de convolução foi fixado em 1 pixel e o preenchimento espacial da camada convolucional de entrada é tal que a resolução espacial é preservada após a convolução, isto é, o preenchimento é de 1 pixel para cada camada de convolução 3x3.

O fase de agrupamento espacial é realizada ao final de cada um dos 5 blocos estipulados na VGG16, logo após a última camada de convolução de cada bloco. O agrupamento utilizado pela VGG166 é o *MaxPooling*, realizado em uma janela de

| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 × 224 RGB image) | | | | | |
| conv3-64 | conv3-64<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

**Figura 23 – Configuração das redes neurais VGGNet testadas**
**Fonte: Simonyan e Zisserman (2014)**

tamanho 2x2, com um descolamento (*stride*) de 2 pixels.

Uma pilha de camadas convolucionais (as quais possuem diferentes profundidade, ou seja um diferente número de camadas como mostrado na Figura 23) é seguida por três camadas totalmente conectadas (FC): As primeiras duas camadas possuem 4096 canais cada uma, ou seja 4096 neurônios de entrada, significando 4096 características diferentes da imagem de entrada. A última camada contém o número de neurônios suficiente para representar distintamente cada classe do conjunto de dados (No caso do ILSVRC 1000 neurônios de entrada), utilizando a função de ativação *softmax*. Em uma VGGNet todas as camadas escondidas utilizam como função de ativação, a função *ReLU*.

A arquitetura VGGNet apresenta filtros de características muito menores do que as apresentadas anteriormente por Krizhevsky et al. (2012), o qual utilizava filtros de tamanho 11x11 com um deslocamento de 4 pixels ou 7x7 com deslocamento de 2 pixels. A rede VGG como dito anteriormente utiliza filtros 3x3 com *stride* 1, porém é facilmente visível que uma pilha de duas camadas de convolução 3x3 sem um agrupamento espacial tem um filtro característico efetivo de tamanho 5x5 assim como uma pilha com três camadas convolucionais apresenta um filtro efetivo de tamanho 7x7.

A escolha apresentada pelo tamanho do filtro, consiste na possibilidade de adicionar função de ativação *ReLU* em cada uma das camadas de convolução tornando a função de decisão mais descritiva. A Figura 24 exemplifica uma macro da arquitetura VGGNet.

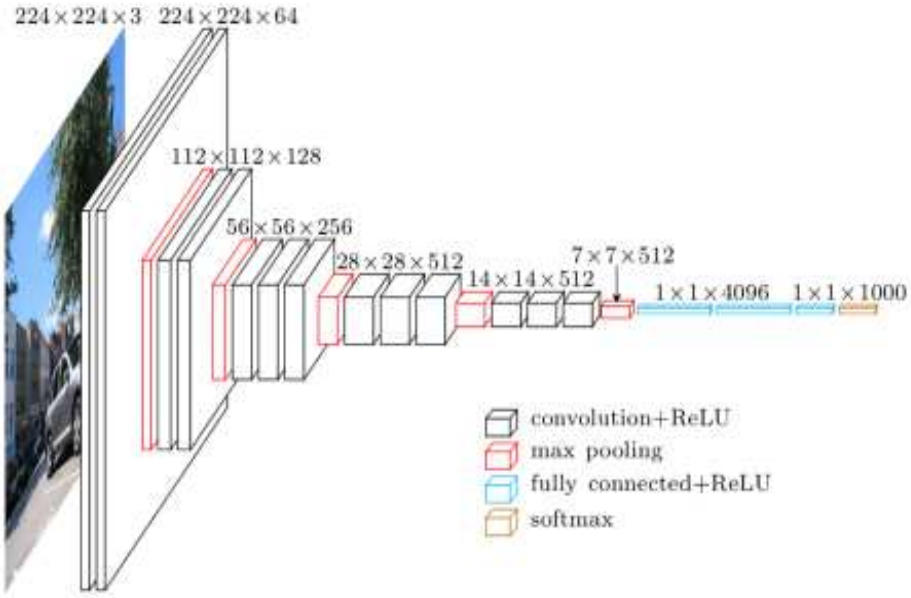


**Figura 24 – Macro da arquitetura neural VGGNet**
**Fonte: FROSSARD (2016, p.s.n.)**

## 4.6 REGULARIZAÇÃO *DROPOUT* PARA REDES NEURAIS

A capacidade de uma rede neural é baseada no nível de complexidade da função que a mesma precisa aprender. Redes neurais pequenas, ou seja, redes com um número relativamente pequeno de parâmetros tem uma baixa capacidade de aprendizado, uma baixa complexidade de resolução, portanto estarão mais propensas a sofrerem *underfitting*, ou seja, um problema no qual a rede não é capaz de aprender a estrutura dos dados mais complexados, resultando assim em um desempenho fraco.

Por outro lado, as redes grandes, com um numero alto de parâmetros, podem sofrer com *overfitting*, ou seja, a rede irá memorizar os dados de treinamento, resultando em um desempenho extremamente bom durante a fase de treinamento, porém apresentará um desempenho fraco em relação ao conjunto de testes.

Uma maneira de abordar estes problemas, é construir uma rede relativamente grande, um pouco maior que o necessário para que se obtenha um bom desempenho na fase de treinamento. Então, com o objetivo de prevenir o *overfitting* pode-se aplicar

um ou vários esquemas de regularização para que se obtenha um bom desempenho de generalização em dados novos.

Nos últimos anos, uma nova técnica de regularização tem se destacado, chamada *Dropout* (RASCHKA; MIRJALILI, 2017). O *Dropout* é uma técnica em que neurônios selecionados aleatoriamente são ignorados durante o treinamento. Eles são descartados aleatoriamente. Isso significa que sua contribuição para a ativação de neurônios é temporariamente removida na fase *forward* e quaisquer atualizações de peso não são aplicadas ao neurônio na fase *backward*.

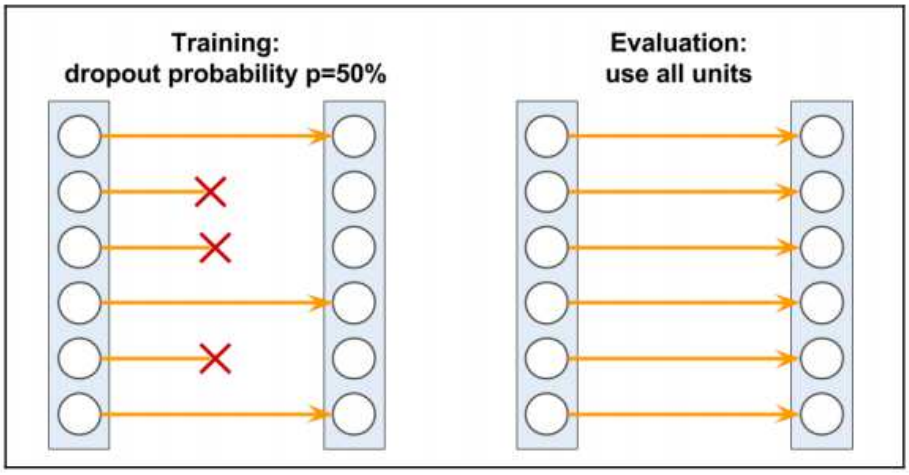


**Figura 25 – *Dropout* com $p = 0.5$**
**Fonte: Raschka e Mirjalili (2017)**

*Dropout* é, em sua maioria das vezes, aplicado em neurônios da camada escondida ou camadas de mais alto nível. Durante o treinamento uma fração dos neurônios é aleatoriamente descartada baseada em uma probabilidade configurada pelo usuário, comumente sendo uma escolha de $p = 0.5$.

O efeito desse abandono aleatório força a rede a aprender uma representação redundante dos dados. Portanto, a rede não pode depender de uma ativação de qualquer conjunto de unidades ocultas, pois elas podem ser desativadas a qualquer momento durante o treinamento sendo forçadas a aprender padrões mais gerais e robustos a partir dos dados (RASCHKA; MIRJALILI, 2017).

Esse abandono aleatório pode efetivamente impedir o *overfitting*. A Figura 25 a seguir mostra um exemplo de aplicação do descarte de neurônios com probabilidade $p = 0.5$ durante a fase de treinamento, assim metade dos neurônios torna-se inativa aleatoriamente. No entanto, durante a fase de testes e validação, todos os neurônios contribuem para computar as pré-ativações da próxima camada.

# 5 MEDIDAS DE DESEMPENHO

## 5.1 MATRIZ DE CONFUSÃO

No campo de aprendizado de máquina e aprendizado profundo, especificamente em problemas relacionados à classificação estatística, uma matriz de confusão também chamada de matriz de erro ou matriz de contingência, é uma tabela que permite a visualização do desempenho realizado pelo algoritmo. Basicamente esta tabela valida o aprendizado supervisionado, comparando sua base de testes com a base de treinamento previamente aprendida e indica o quanto de acerto e erro o algoritmo obteve (NOGARE, 2015).

Cada linha da matriz representa as instâncias em uma classe prevista, enquanto que cada coluna representa as instâncias em uma classe real. Uma matriz de contingência é um tipo de tabela em um formato de matriz que exibe a distribuição de frequência das variáveis. A matriz de confusão é um tipo especial das tabelas de contingências, é uma matriz quadrada com duas dimensões (valores reais e valores previstos) e um conjunto idêntico de classes em ambas as dimensões, na qual o tamanho da matriz será definida pela número de classes existentes na classificação.

Na Figura 26, tem-se um exemplo de matriz de confusão na qual uma base de testes contêm um total de 27 amostras, das quais 8 são gatos, 6 cachorros e 13 coelhos.

| | | Classes | | |
|---|---|---|---|---|
| | | Gato | Cachorrro | Coelho |
| Classes Previstas | Gato | 5 | 2 | 0 |
| | Cachorrro | 3 | 3 | 2 |
| | Coelho | 0 | 1 | 11 |

**Figura 26 – Matriz de confusão para 3 classes**
**Fonte: Autoria própria 2017.**

A diagonal principal representa todos os acertos corretos que o sistema obteve, enquanto que a soma de cada coluna representa o numero total de amostras

representada pela classe da coluna.

Para a área de análises preditivas, a tabela de confusão é uma tabela 2x2 a qual apresenta o numero de falsos positivos, falsos negativos, verdadeiros positivos e verdadeiros negativos, permitindo uma análise mais detalhada, do que a simples proporção de classificações corretas também chamada de *accuracy*.

A *accuracy* é um parâmetro estatístico simples derivado da matriz de confusão, que é dado pela divisão da soma dos valores da diagonal principal dessa matriz pela quantidade total de amostras usadas para o cálculo da mesma (CASANOVA, 2013). Dada a matriz de confusão M, a acurácia de um classificador pode ser calculada por:

$$Accuracy = \frac{\sum_{i=1}^{r} m_{ii}}{n} \tag{24}$$

onde r é o número de linhas e colunas da matriz de confusão, $m_{ii}$ é a contagem das observações da linha $i$ e coluna $i$ e $n$ é o número total de observações.

Porém *accuracy* deve ser usada em *datasets* com a mesma proporção de exemplos para cada classe, pois em problemas com classes desproporcionais, ela causa uma falsa impressão de bom desempenho. Por exemplo, num dataset em que 80% dos exemplos pertençam a uma classe, só de classificar todos os exemplos naquela classe já se atinge uma precisão de 80%, mesmo que todos os exemplos da outra classe estejam classificados incorretamente (FILHO, 2015).

Como mostra o exemplo da Figura 27, a qual é uma continuação do exemplo de classificação entre gatos, cachorros e coelhos. A tabela de confusão será montada de forma a conter as amostras selecionadas como valor verdadeiro da classe escolhida para análise e a soma de todas as outras amostras como valor falso.

| | | Classes | |
|---|---|---|---|
| | | Gato | Cachorrro |
| Classes Previstas | Gato | 5 Positivos Verdadeiros | 2 Falsos Positivos |
| | Não Gato | 3 Falsos Negativos | 17 Verdadeiros Negativos |

**Figura 27 – Exemplo de tabela de confusão**
**Fonte: Autoria própria 2017.**

Após a criação da tabela de confusão, uma nova *accuracy* para a classe $x$

pode ser calculada a partir da equação:

$$Accuracy = \frac{\sum VerdadeirosPositivos + \sum VerdadeirosNegativos}{\sum TotaldeAmostras} \tag{25}$$

E pode-se calcular também a taxa de erro total através da equação:

$$E_r = \frac{FN + FP}{Num.deAmostras} \tag{26}$$

## 5.2 VALIDAÇÃO CRUZADA

Validação cruzada é uma técnica frequentemente utilizada para avaliar a capacidade de generalização de um modelo a partir de um conjunto de dados (KOHAVI et al., 1995), ou seja, determinar como os resultados de uma análise serão generalizados para o conjunto de dados independentes.

O principal conceito por trás das técnicas de validação cruzada está em dividir o conjunto total de amostras em dois subconjuntos mutuamente exclusivos, os subconjuntos de treinamento e teste.

Existem diversas maneiras de fazer esta divisão dos conjuntos, sendo as principais formas o método *holdout*, *k-fold* e *leave-one-out*.

### 5.2.1 MÉTODO *HOLDOUT*

O método *holdout cross-validation* é um dos métodos mais comuns para estimar a generalização de desempenho em modelos de *machine learning* (RASCHKA; MIRJALILI, 2017). Usando o método *holdout*, o *dataset* é inicialmente dividido em dois sub-conjuntos de treinamento e teste mutualmente exclusivos de forma aleatória. Não existe um tamanho fixo estipulado para a divisão desses dois conjuntos embora seja comumente usado 1/3 para testes e o restante para treinamento(KOHAVI et al., 1995).

Uma melhor maneira de usar o método *holdout* para a seleção do modelo é separa-lo em três partes: conjunto de treinamento, conjunto de validação e conjunto de testes. O conjunto de treinamento será usado para treinar diferentes modelos, e o desempenho em cima do conjunto de validação será utilizado para a seleção do modelo. A vantagem de se possuir um conjunto de testes que o modelo ainda não visualizou durante os passos do treinamento e seleção do modelo é que podemos

obter uma estimativa menos tendenciosa dessa capacidade de generalizar para novos dados. A Figura 28 ilustra o conceito do método *holdout cross-validation*, no qual usa-se um *set* de treinamento repetidas vezes para avaliar o desempenho do modelo usando diferentes parâmetros.

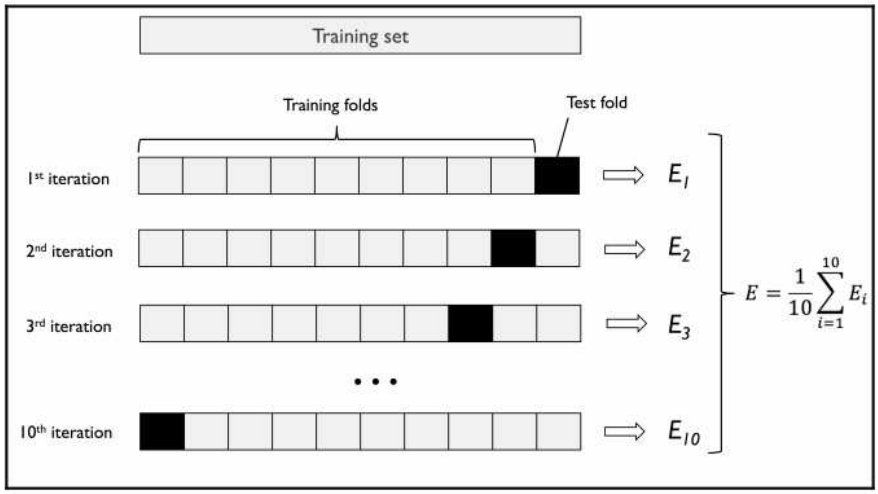


**Figura 28 – *K-fold cross-validation***
**Fonte: Raschka e Mirjalili (2017)**

Umas das desvantagens de se utilizar o método *holdout* é que a estimativa de desempenho pode ser muito sensível a como os dados de treinamento são particionados em conjunto de treinamento e conjunto de validação. A estimativa irá variar para diferentes amostras dos dados. Existem métodos para melhorar esta estimativa, tais como repetir este método $k$ vezes em $k$ sub-conjuntos dos dados de treinamento.

### 5.2.2 MÉTODO *K-FOLD*

No método de validação cruzada utilizando *K-fold*, o conjunto total de amostras é particionado em $K$ subconjuntos mutuamente excludentes e de tamanhos aproximadamente iguais. Dos $K$ subconjuntos criados um é retirado para ser usado como o conjunto de testes na avaliação de desempenho, e os restantes $K - 1$ são utilizados como dados de treinamento. O processo de validação cruzada é então repetido $K$ vezes, onde cada subconjunto é utilizado exatamente uma vez como conjunto de testes. Os $K$ resultados obtidos são então combinados para produzir uma única estimação de erro (CASANOVA, 2008).

Normalmente usa-se *k-fold cross-validation* para ajuste do modelo, ou seja, encontrar os valores ideais de hiperparâmetro que produzam um desempenho de ge-

neralização satisfatório (RASCHKA; MIRJALILI, 2017).

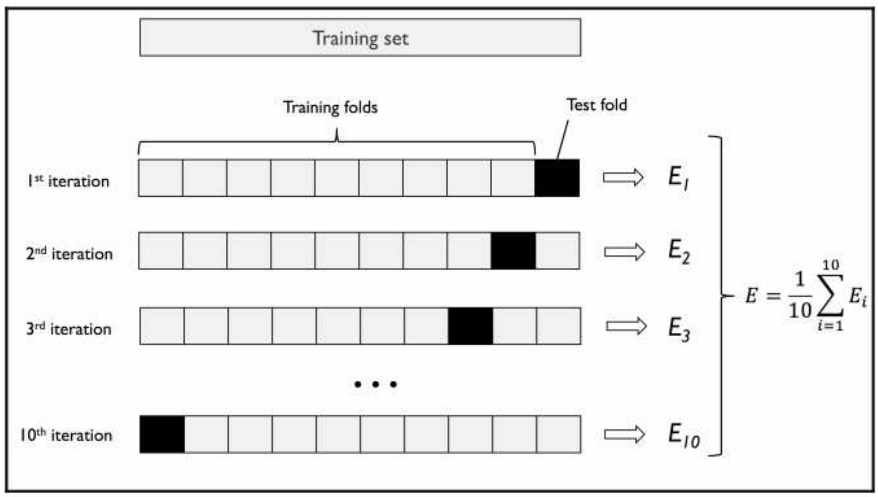


**Figura 29 – *Holdout cross-validation***
**Fonte: Raschka e Mirjalili (2017)**

### 5.2.3 MÉTODO *LEAVE-ONE-OUT*

O método *leave-one-out* é uma variação do método *k-fold*, o qual utiliza apenas uma amostra como teste e o restante das amostras como treinamento. Porém este processo é repetido $n$ vezes, onde $n$ é o número total de amostras, ou seja, o processo será executado ao menos uma vez com cada amostra sendo utilizada como teste.

Embora existam diversos métodos para separar os conjuntos de validação cruzada a forma de calcular a precisão obtida permanece sempre a mesma, como segue a equação (27):

$$A_{cf} = \frac{1}{v}\sum_{i=1}^{v}(y_i - \hat{y}_i) \tag{27}$$

onde $v$ é o número de dados de validação, $y$ o resultado real e $\hat{y}$ o valor de saída da rede.

# 6 BASES DE DADOS

## 6.1 GATOS E CACHORROS

O banco de dados original Asirra, é composto por imagens de cachorros e gatos provenientes de duas redes sociais dedicadas a coleção e discussões sobre imagens de animais de estimação, CATSTER e DOGSTER, de grupos do Flickr e também a partir do Google imagens. O banco de imagens original consiste em 25.000 imagens.

Porém, para a análise e aplicação no desenvolvimento deste trabalho foram utilizadas um total de 4.000 imagens de cachorros para treinamento da rede. Imagens em sua maioria contendo uma considerável quantidade de ruído, como pessoas e outros objetos que não possuem importância para a rede a ser treinada. A figura 30 mostra alguns exemplos de imagens encontradas dentro do conjunto de dados.

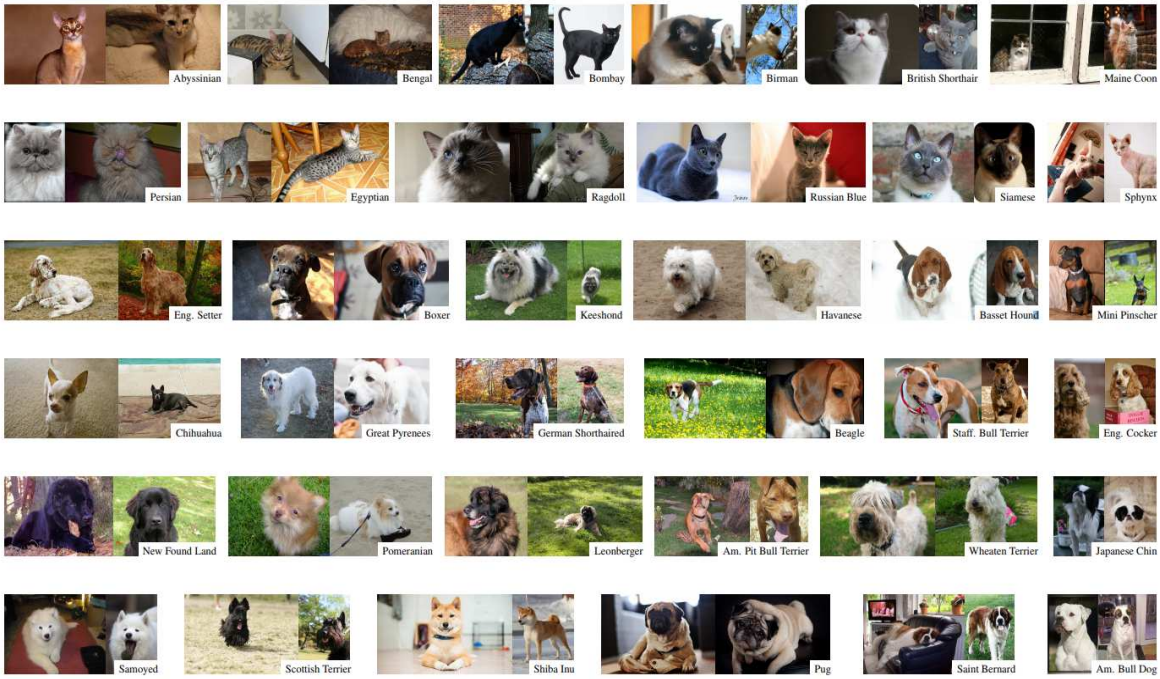


**Figura 30 – Duas imagens por raça são mostradas, ilustrando a variabilidade dos dados**
**Fonte: Parkhi et al. (2012)**

Além das imagens de cachorros, o conjunto possui outras 4.000 imagens para treinamento de gatos seguindo o mesmo estilo das imagens descritas para o conjunto

de cachorros.

O conjunto de testes contabiliza um total de 2.000 imagens, distribuídas igualmente entre as duas espécies de animais apresentadas no treinamento. As imagens sejam elas do conjunto de treinamento ou testes não apresentam um tamanho de entrada fixo, portanto foi necessário a utilização de uma função *resize*, ou seja, um redimensionamento das imagens em uma diferente proporção, antes de serem utlizadas como entradas para a rede neural convolucional.

## 6.2 PLANTCLEF 2016

Os dados do PlantCLEF (2016) a serem analisados podem ser encontrados no site para download. São amostras de imagens submetidas por usuários da aplicação mobile *Pl@ntNet* a qual está disponível para *iPhone* e *Android* e conta com a participação de centenas de usuários ativos, sendo submetidas milhares de imagens por dia.

A Figura 31 apresenta algumas amostras de imagem, as quais contêm um alto nível de ruido, objetos desconhecidos e até mesmo plantas que não estão contidas no banco de dados de treinamento. Dificultando assim assim a classificação por parte da rede, sendo necessária uma rede mais complexa e com um maior número de camadas.

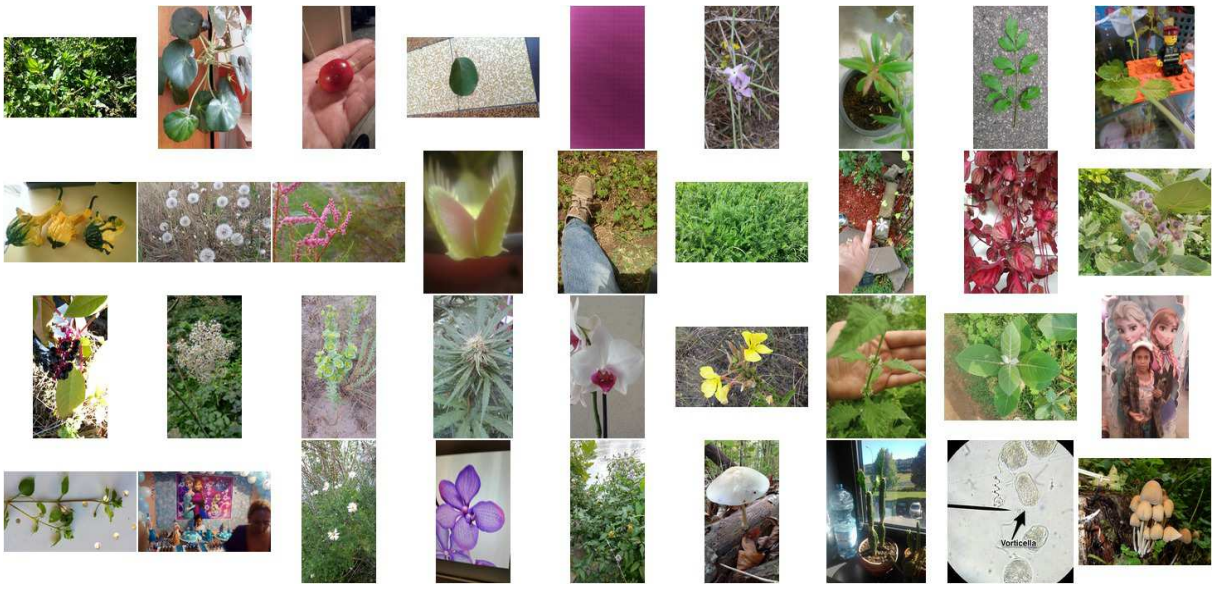


**Figura 31 – Amostras do banco de imagens PlantCLEF**
**Fonte: PlantCLEF (2016)**

O conjunto de imagens para treinamento em 2016 foi composto por 91.758

imagens. As quais foram divididas em 70% para realização do treinamento e 30% para realização de testes.

## 6.3 IMAGENS DE FOLHAS

O banco de dados é composto por um total de 800 amostras, divididas em 40 classes diferentes (aqui denominadas por $C01, C02, C03, ..., C40$), totalizando portanto 20 amostras por classe.

As amostras foram coletadas *in vivo* e cada folha foi submetida a um processo de limpeza para retirada de impurezas (tais como poeira, sedimentos, etc.), ajustada para que o seu eixo central (linha que une as extremidades basal e apical) ficasse na posição vertical e digitalizada individualmente com o auxilio de um escâner de mesa a uma resolução de 1200dpi (*pontos por polegada*). Para cada imagem de folha coletada foi definido um padrão para a construção do nome do respectivo arquivo de imagem. O nome ficou definido como $cXX\_WWW.png$, sendo XX um número composto por dois dígitos denotando o número da classe/espécie, WWW um número composto por três dígitos identificando o número da folha/amostra da referida classe/espécie Casanova (2008).

Nenhum procedimento de normalização extra foi adotado uma vez que o escâner possui condições de iluminação e resolução controladas. Utilizou-se armazenagem com compressão *lossless*, que viabiliza a compressão dos dados sem perda de informações.

## 6.4 IMAGENS DE TEXTURAS

Utilizando como base as mesmas imagens pertencentes a Seção 6.3, apresentadas na Figura 32 a seguir. Foram extraídas aleatoriamente pequenos quadrados da imagem original, um total de 10 imagens de tamanho 128x128 *pixels* de cada uma das folhas, com o objetivo de representar parcialmente a textura em diferentes pontos da folha. Totalizando assim um novo banco de imagens com 8.000 amostras.

**Figura 32 – As 40 classes utilizadas pela CNN**
**Fonte: Casanova (2008)**

# 7 RESULTADOS

A seguir serão apresentados os resultados obtidos nos *datasets* do Capítulo 6. Importante ressaltar que todos os códigos e execuções foram realizadas utilizando *Keras*. Uma API de alto nível para redes neurais capaz de rodar em cima de bibliotecas como *Tensorflow* e *Theano*.

## 7.1 RESULTADOS DO *DATASET* DE GATOS E CACHORROS

A fim de obter resultados preliminares e treinar uma rede convolucional básica de classificação binária, foram utilizadas as imagens do *dataset* Asirra. Composto por imagens como as apresentadas pela Figura 30.

A Rede Neural Convolucional construída para obter resultados preliminares, consiste de uma camada convolucional, uma camada de *pooling*, uma operação de *flattening* e por último uma camada totalmente conectada.

A camada convolucional é composta por 32 filtros (detectores de características) de tamanho $3$x$3$ aplicados em imagens de entrada redimensionadas para um tamanho de $64$x$64$x$3$, e para evitar a linearidade na imagem a função de ativação *ReLU* foi utilizada.

```
1 [...]
2 # Layer 1 - Convolution
3 classifier.add(Convolution2D(32,
4                                (3,3),
5                                input_shape=(64, 64, 3),
6                                activation = 'relu'))
7 [...]
```

A camada de *pooling*, utilizando a função de máximo local para extrair o maior valor da região delimitada pelo tamanho dos filtros configurados em $2$x$2$, com o objetivo de evidenciar a característica detectada pelo filtro.

Após passar pelas camadas iniciais básicas, aplica-se uma operação de *flattening* transformando assim as matrizes em um vetor coluna na qual cada posição do

vetor irá representar uma das entradas da MLP a seguir.

```
[...]
# Layer 2 - Max Pooling e Flatten
classifier.add(MaxPooling2D(pool_size = (2,2)))
classifier.add(Flatten())
[...]
```

E finalmente para terminar a CNN, a camada totalmente conectada, na qual o vetor criado pela operação de *flattening* serão as entradas de uma MLP tradicional, composta de uma camada escondida de $128$ neurônios utilizando a função de ativação *ReLU* e a camada de saída com um único neurônio utilizando a função de ativação sigmoide. A utilização de um único neurônio na saída se refere a possibilidade de existir apenas duas classes no problema, ou seja, a saída será ou gato ou cachorro.

```
[...]
# Layer 4 - FulL Conection
classifier.add(Dense(output_dim = 128,
                     activation = 'relu')) #hidden layer
classifier.add(Dense(output_dim = 1,
                     activation = 'sigmoid')) #output layer
[...]
```

Por último, após criado e compilado o modelo, a CNN foi executado durante $25$ épocas, no intuito de aumentar a precisão dos resultados.

Uma época significa que um conjunto de dados é passado pela fase *forward* e pela *backward* através da rede neural somente uma vez, ou seja, um ciclo de treinamento completo.

```
[...]
# Keras 2
classifier.fit_generator(training_set,
                         steps_per_epoch=8000,
                         epochs=25,
                         validation_data=test_set,
                         validation_steps=2000)
```

A métrica utilizada para medir os resultados foi a precisão (*accuracy*) com que a rede identificava a classe correta do animal. Para as imagens de treinamento a *ac-*

*curacy* foi de $99,56\%$ enquanto que para as imagens de teste a *accuracy* apresentada foi $76,02\%$. A adição de mais camadas convolucionais e número de filtros, assim como um tamanho de imagens de entrada maiores e a adição de neurônios na ANN clássica, podem implicar redução da variação de *accuracy* encontrada entre os resultados de treinamento e teste.

## 7.2 RESULTADOS DO IMAGECLEF 2016

Inicialmente o objetivo era construir uma rede neural convolucional de estrutura similar a uma arquitetura VGG16. Para isso as imagens do *dataset* do ImageCLEF precisaram ser redimensionadas para um tamanho de 224x244 *pixels*, o mesmo tamanho utilizado em uma rede VGG16.

Porém o fato do *dataset* apresentar um grande número de imagens, a transformação para o tamanho desejado demandou uma grande quantidade de memória RAM.

Portanto optou-se pela possibilidade de realizar as execuções dos algoritmos em um servidor da Google Cloud. O qual disponibiliza $U\$$300,00 dólares para a realização de testes. Existem duas opcões para realizar a execução de um algoritmo de *deep learning* na Google Cloud. Primeiro pode-se usar as APIs específicas da GC, disponíveis para o uso em redes neurais ou, como foi optado no trabalho, a criação de uma máquina virtual.

Para a criação de uma máquina virtual no GC, escolhe-se o servidor da localidade desejada, na qual a máquina ficará hospedada, e suas configurações de memória RAM, quantidade de processadores e placa de vídeo. Conforme as especificações da máquina aumentam o custo por ela consequentemente é acrescentado também. Portanto optou-se pela criação de uma máquina virutal inicialmente com 50GB de memória RAM e 16 *threads* e sem GPU, para reduzir os custos até que todas as imagens estivessem armazenadas e os códigos prontos para serem rodados.

A transferência das imagens para o servidor da GC ocupou cerca de 34 horas para sua finalização. Após todas as imagens transferidas, executou-se o código responsável pelo redimensionamento e transformação em um conjunto de matrizes.

Na primeira tentativa, o máquina virtual parou sua execução devido a falta de memória RAM. Portanto uma nova aquisição de memória foi realizada. Deixando a máquina com 16 *threads* e 128GB de mémoria. A nova execução do código levou aproximadamente 10 horas, para que todas as imagens fossem convertidas e arma-

zenadas.

A arquitetura dos *datasets* disponíveis pelo ImageCLEF, consiste em uma imagem, seguida por um arquivo XML (*Extensible Markup Language*) de mesmo nome. Utilizando a biblioteca *Pandas*, foram extraídos desses arquivos os rótulos de cada imagem e diversos outros dados.

Após aquisição dos rótulos e formatação correta das imagens, retorna-se a etapa inicial de configuração da máquina para realizar a adição da quantidade de GPUs desejadas. Porém, para que uma GPU seja adiciona, alguns critérios devem ser respeitados como por exemplo: $X$ número da GPU $Y$ só pode ser adicionada em um máquina com um número $Z$ de processadores e um número $W$ de memória. Além disso, todos os requisitos devem estar disponíveis na região na qual máquina foi alugada.

Portanto não foi possível executar a rede convolucional criada na máquina virtual que estava sendo utilizada desde o início.

Os problemas de escalabilidade não foram os únicos. Com o objetivo de testar o funcionamento da VGG16 foram realizados testes de classificação binária, selecionando apenas duas classes do *dataset* apresentado no Capítulo 6.3 e notou-se que a rede não estava aprendendo.

## 7.3 RESULTADOS DO *DATASET* DE FOLHAS E TEXTURAS

### 7.3.1 TODAS AS FOLHAS EM UMA REDE VGG16

A execução desta rede foi realizada em uma máquina física localizada nas dependências da universidade executando um código utilizando todas as camadas descritas pela VGG16. Todo o conjunto de imagens foi transformando em um conjunto de matrizes e carregado na memória RAM. Em seguida foi criado um vetor de rótulos para representação de cada uma das imagens.

Após a preparação dos *labels* e imagens, foi necessária a aplicação de uma função responsável por realizar o *split* dos dados. Resultando em uma separação de 70% dos dados para treinamento e 30% para testes.

Realizada a construção da rede convolucional contendo os 5 blocos presentes em uma rede VGG16, o algoritmo foi executado por um número de 200 épocas, resultando em um valor de *accuracy* inferior a 3%. A realização desse processo pode confirmar que embora a rede executada seja semelhante a uma VGG, notou-se que a

rede não foi capaz de realizar as classificações adequadas, resultando em uma rede que não convergia a uma boa classificação.

Existem inúmeros motivos a serem analisados na tentativa de encontrar o motivo da rede não ter aprendido. Por exemplo o fato das imagens conterem uma grande quantidade de *pixels* em branco ou inicialmente a falta de um processamento prévio das imagens antes que pudessem ser usadas pela CNN.

### 7.3.2 TODAS AS TEXTURAS EM UMA REDE CONVOLUCIONAL BÁSICA

A fim de identificar se o problema estava relacionado ao *dataset* ou a rede criada, foi criado um novo banco de imagens. Partindo das imagens contidas no Capítulo 6.3 foram extraídos pequenos fragmentos de textura de tamanho $128$x$128$, totalizando 8000 diferentes imagens de entrada para a rede.

Procurando por um melhor desempenho de toda a rede, foi encontrada uma nova maneira de fornecer as imagens como entrada para a ConvNet. A utilização de uma função da API *Keras* chamada *ImageDataGenerator*.

A função permite que possíveis definições possam ser configuradas para preparação das imagens, exigindo somente que as imagens estejam dispostas seguindo uma estrutura imposta pela biblioteca. Na qual, um diretório deve conter outros dois diretórios referentes as imagens de treinamento e testes que, por sua vez devem conter um diretório para cada classe disponível no banco de imagens.

Além disso o processo de treinamento das imagens realizados por *batches* agora com a utilização da função, realiza o carregamento e transformação da imagem automaticamente de acordo com o tamanho do *batch*, carregando na memória somente as imagens utilizadas pelo *batch* específico. Evitando assim o problema encontrado na execução do processo anterior o qual demandava uma enorme quantidade de memória.

Para iniciar a criação do modelo para obtenção de novos resultados, foi utilizada a mesma rede convolucional dos primeiros testes realizados, com os gatos e cachorros. Porém algumas modificações foram necessárias em relação a taxa de aprendizado e o número de classes, de uma classificação binária para uma classificação categórica com 40 classes distintas.

A Figura 33 apresenta um resumo com os dados da arquitetura criada. Na qual encontram-se presentes duas camadas convolucionais e uma pequena rede MLP com 40 neurônios de saída representando o número de classes disponíveis no banco

de imagens.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 61, 61, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 28800)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               3686528
_________________________________________________________________
dense_2 (Dense)              (None, 40)                5160
=================================================================
Total params: 3,701,832
Trainable params: 3,701,832
Non-trainable params: 0
```

**Figura 33 – Resumo rede convolucional básica**
**Fonte: Autoria própria (2018).**

A rede foi executada durante um total de 8 épocas, o que durou por um período de aproximadamente 4:30 horas, cerca de 32 minutos por época como mostra a Figura 34.

```
Epoch 1/8
6400/6400 [==============================] - 2096s 328ms/step - loss: 1.0909 - acc: 0.6603 - val_loss: 1.1846 -
val_acc: 0.6587
Epoch 2/8
6400/6400 [==============================] - 2068s 323ms/step - loss: 0.3786 - acc: 0.8733 - val_loss: 1.6269 -
val_acc: 0.6450
Epoch 3/8
6400/6400 [==============================] - 2065s 323ms/step - loss: 0.2087 - acc: 0.9309 - val_loss: 1.8380 -
val_acc: 0.6538
Epoch 4/8
6400/6400 [==============================] - 2091s 327ms/step - loss: 0.1458 - acc: 0.9524 - val_loss: 1.9876 -
val_acc: 0.6631
Epoch 5/8
6400/6400 [==============================] - 2106s 329ms/step - loss: 0.1145 - acc: 0.9638 - val_loss: 2.4137 -
val_acc: 0.6531
Epoch 6/8
6400/6400 [==============================] - 2109s 329ms/step - loss: 0.0996 - acc: 0.9691 - val_loss: 2.2954 -
val_acc: 0.6806
Epoch 7/8
6400/6400 [==============================] - 2113s 330ms/step - loss: 0.0893 - acc: 0.9727 - val_loss: 2.2932 -
val_acc: 0.6769
Epoch 8/8
6400/6400 [==============================] - 2181s 341ms/step - loss: 0.0776 - acc: 0.9770 - val_loss: 2.6115 -
val_acc: 0.6687
```

**Figura 34 – Execução rede convolucional básica com 40 classes**
**Fonte: Autoria própria (2018).**

Em um primeiro momento, quando executada a rede apresentou problemas em relação a taxa de perda como pode ser visto na Figura 35. A fim de evitar com que ocorresse novamente este erro a taxa de aprendizado da rede foi alterada para um valor dez vezes menor.

```
Epoch 1/50
6400/6400 [==============================] - 2117s 331ms/step - loss: 1.0445 - acc: 0.6732 - val_loss: 1.4727 - val_acc: 0.6356
Epoch 2/50
6400/6400 [==============================] - 2108s 329ms/step - loss: 0.3193 - acc: 0.8958 - val_loss: 1.9294 - val_acc: 0.6425
Epoch 3/50
6400/6400 [==============================] - 2112s 330ms/step - loss: 0.1904 - acc: 0.9384 - val_loss: 2.3625 - val_acc: 0.6319
Epoch 4/50
6400/6400 [==============================] - 2143s 335ms/step - loss: 0.1417 - acc: 0.9547 - val_loss: 2.3949 - val_acc: 0.6456
Epoch 5/50
6400/6400 [==============================] - 2108s 329ms/step - loss: 0.1152 - acc: 0.9647 - val_loss: 2.4829 - val_acc: 0.6456
Epoch 6/50
6400/6400 [==============================] - 2105s 329ms/step - loss: 0.1035 - acc: 0.9689 - val_loss: 2.9758 - val_acc: 0.6106
Epoch 7/50
6400/6400 [==============================] - 2109s 330ms/step - loss: 0.0916 - acc: 0.9728 - val_loss: 2.7260 - val_acc: 0.6650
Epoch 8/50
6400/6400 [==============================] - 2107s 329ms/step - loss: 0.0842 - acc: 0.9754 - val_loss: 3.1057 - val_acc: 0.6288
Epoch 9/50
6400/6400 [==============================] - 2091s 327ms/step - loss: nan - acc: 0.1579 - val_loss: nan - val_acc: 0.0250
Epoch 10/50
6400/6400 [==============================] - 2088s 327ms/step - loss: nan - acc: 0.0250 - val_loss: nan - val_acc: 0.0250
Epoch 11/50
6400/6400 [==============================] - 2087s 327ms/step - loss: nan - acc: 0.0250 - val_loss: nan - val_acc: 0.0250
```

**Figura 35 – Primeira execução da rede básica com todas as texturas**
**Fonte: Autoria própria (2018).**

Pode-se notar que após alterado o valor da taxa de aprendizado e mesmo executando por um pequeno período de tempo a *accuracy* alcançou valores bastante satisfatórios. Aproximadamente 98% durante a fase de treinamento e 67% durante a fase de testes, como pode ser visto na Figuras 36.

Porém os valores de perda para a fase de testes ainda não apresentaram bons resultados, como pode ser visto na Figura 37.

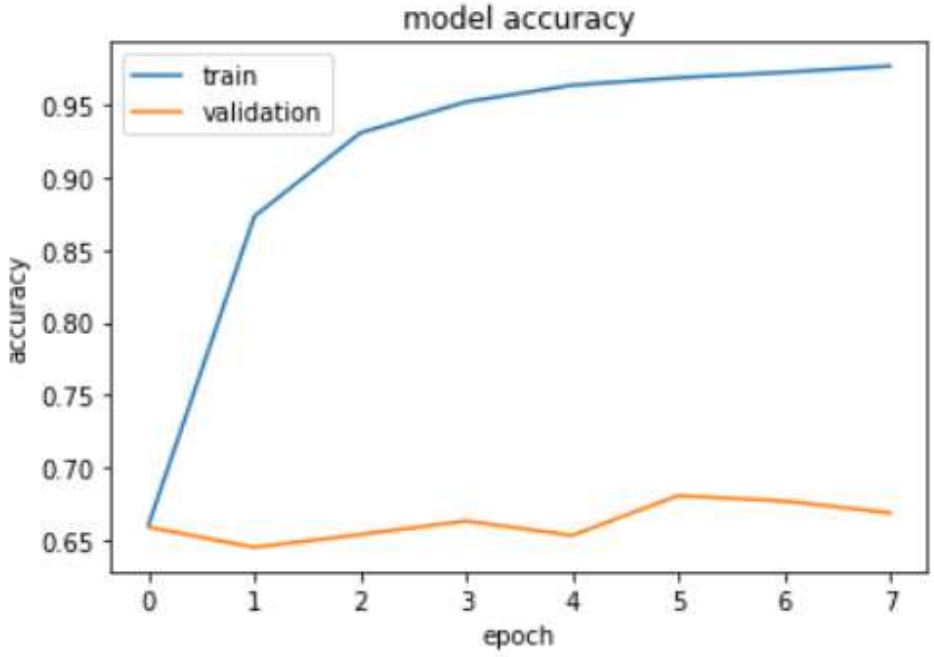


**Figura 36 – Taxas de *accuracy* para treinamento e teste (CNN básica)**
**Fonte: Autoria própria (2018).**

Embora os valores obtidos pela rede alcançarem bons resultados, eles ainda são inferiores aos resultados apresentados por Casanova (2008), com uma taxa de

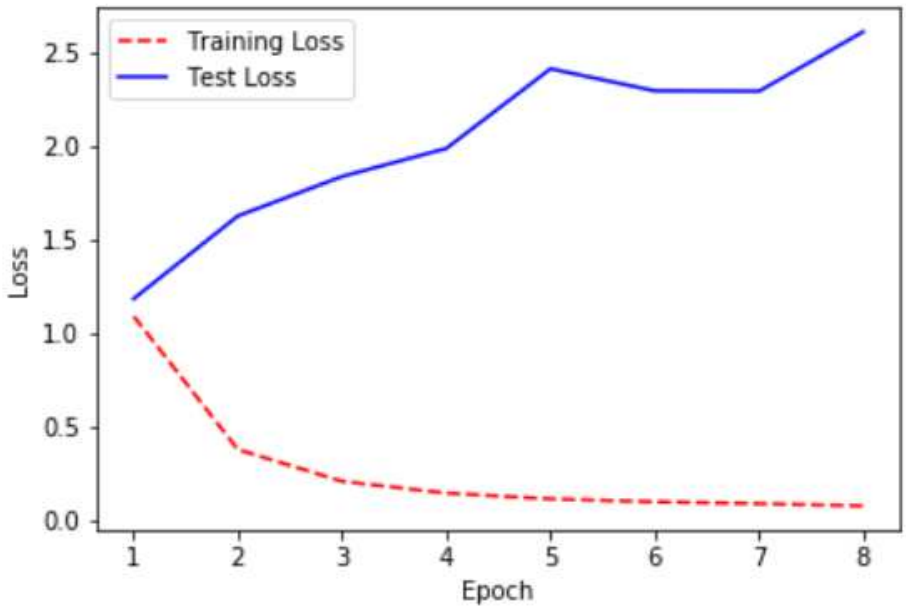


**Figura 37 – Taxas de perda para treinamento e teste (CNN básica)**
**Fonte: Autoria própria (2018).**

*accuracy* de 81,52% utilizando técnicas de visão computacional e *machine learning*.

Os resultados foram satisfatórios se levada em consideração a baixa complexidade da rede se comparada a complexidade das texturas extraídas do banco de imagens das folhas.

Portanto a fim de obter melhores resultados com o mesmo *dataset*, uma nova rede foi implementada, encaminhando-se para a arquitetura de uma rede VGG 16.

Para esta nova rede foram acrescentados 3 dos 5 blocos presentes em uma rede VGG16, adicionando assim um maior número de camadas convolucionais, funçoes de *Padding* e *Pooling*. A Figura 38 mostra o resultante da nova rede implementada.

Importante ressaltar o aumento no número dos parâmetros da primeira para a segunda rede. Enquanto que a primeira rede apresentada na Figura 33 possui 3.701.832 parâmetros de treinamento, a rede apresentada na Figura 38 apresenta 26.856.552 parâmetros de treinamento, aproximadamente 625% a mais de parâmetros variáveis.

As Figuras 39 e 40 apresentadas demonstram os resultados obtidos pela nova rede implementada.

Pode-se analisar que, aumentando a complexidade da rede, os resultados obtidos foram consideravelmente melhores, seja para a *accuracy* durante a fase de

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 126, 126, 64)      1792
_________________________________________________________________
zero_padding2d_1 (ZeroPaddin (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 126, 126, 64)      36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 63, 63, 64)        0
_________________________________________________________________
zero_padding2d_2 (ZeroPaddin (None, 65, 65, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 63, 63, 128)       73856
_________________________________________________________________
zero_padding2d_3 (ZeroPaddin (None, 65, 65, 128)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 30, 30, 128)       147584
_________________________________________________________________
zero_padding2d_4 (ZeroPaddin (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 30, 30, 256)       295168
_________________________________________________________________
zero_padding2d_5 (ZeroPaddin (None, 32, 32, 256)       0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 256)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 14, 14, 256)       590080
_________________________________________________________________
flatten_1 (Flatten)          (None, 50176)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               25690624
_________________________________________________________________
dense_2 (Dense)              (None, 40)                20520
=================================================================
Total params: 26,856,552
Trainable params: 26,856,552
Non-trainable params: 0
```

**Figura 38 – CNN com 3 dos 5 blocos presentes em uma rede VGG16**
**Fonte: Autoria própria (2018).**

treinamento, aproximadamente 98% como para a *accuracy* na fase de testes de aproximadamente 72%.

Embora apresentando melhoras também relacionadas à taxa de perda da rede a partir da Figura 40 pode-se observar sugestões de *overfitting* a partir da época 20, quando as duas curvas de treinamento e teste começam a se distanciar em números cada vez maiores.

Com os resultados obtidos nesta ultima etapa de testes, acredita-se que seguindo o mesmo modo de implementação e uma disponibilidade de recursos e tempo maior, a rede possa ser expandida para uma rede VGG16, obtendo resultados que

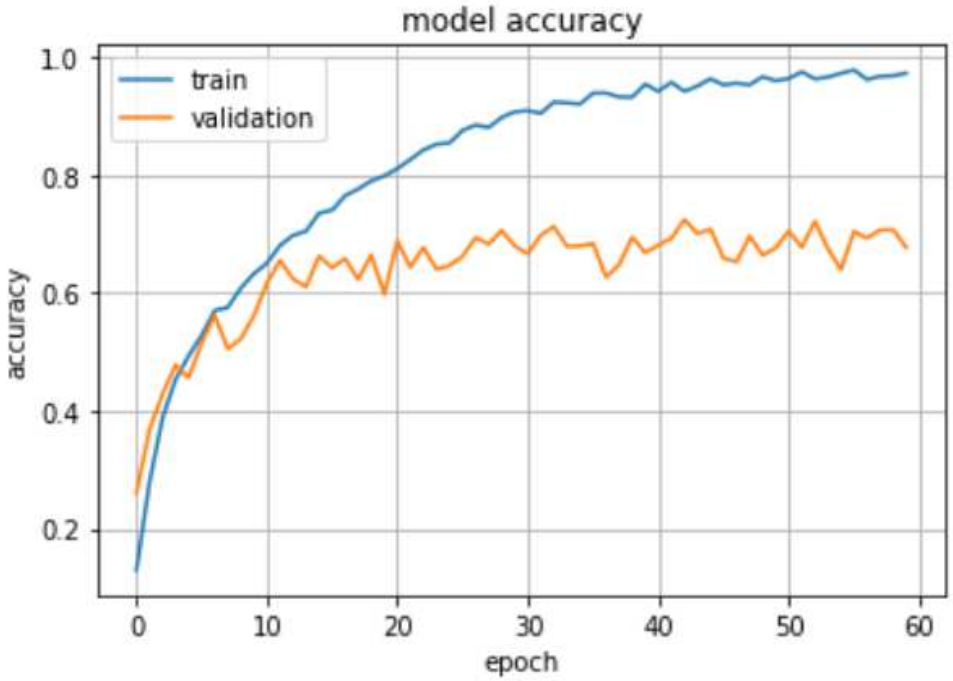


**Figura 39 – Taxa de *accuracy* para treinamento e teste de uma CNN com 3/5 Blocos de uma rede VGG16**
**Fonte: Autoria própria (2018).**

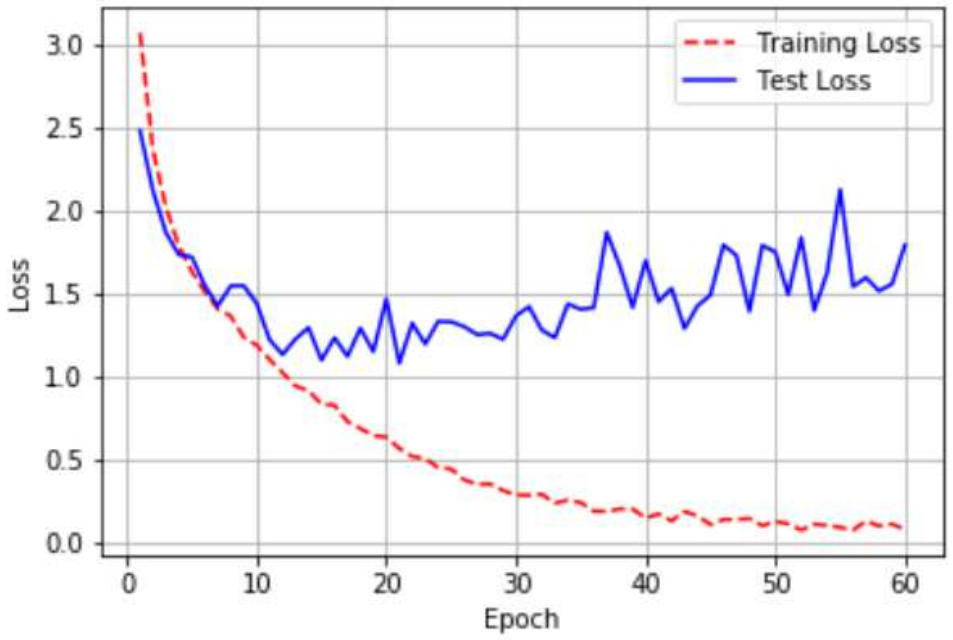


**Figura 40 – Taxa de perda para treinamento e teste de uma CNN com 3/5 Blocos de uma rede VGG16**
**Fonte: Autoria própria (2018).**

possam ser ainda mais satisfatórios.

# 8 CONCLUSÕES

Melhores investigações precisam ser realizadas para adaptar a rede VGG16 ao problema do ImageCLEF. Embora existam artigos que já aplicam esta rede em particular ao problema, tivemos vários desafios desde recursos, tempo e erros inesperados na plataforma a qual estava sendo inicialmente executada.

Também deve-se se considerar uma melhor investigação de possíveis técnicas de regularização que possam ser empregadas a fim de evitar problemas como *overfitting*. Embora os resultados obtidos não sugiram overfitting em todos os casos acredita-se que os problemas matemáticos obtidos (Ex: nan) possam ser oriundos de uma má configuração da taxa de aprendizado e/ou uma convergência precoce da rede. Maiores investigações a respeito dos parâmetros da rede tais como, *dropout*, técnicas de aprendizado, taxa de aprendizado, numero de camadas precisam ser analisadas mais detalhadamente.

Embora os resultados obtidos não tenham sido os esperados acredita-se que as redes neurais convolucionais sejam uma excelente técnica para reconhecimento de imagens, visto todos os resultados relatados na literatura recente. deve-se observar que, na base de dados de texturas, a taxa de acerto obtida pelo modelo de Casanova (2008) obteve 81.52% de *accuracy* e nosso resultado é de aproximadamente 70% de *accuracy* mesmo sem nenhum tipo de ajuste de hiperparâmetros necessários.

O desenvolvimento de projetos multidisciplinares, como nesse trabalho sobre identificação de espécies vegetais, demanda além dos conhecimentos naturais sobre computação, o estudo e entendimento do problema biológico que está sendo abordado. Para este projeto foi necessário realizar uma revisão bibliográfica sobre assuntos relacionados a taxonomia de espécies vegetais.

Entre os benefícios e aplicações da identificação vegetal temos o conhecimento da flora com credibilidade cientifica, auxilio para o manejo sustentável dos recursos florestais, recomposição das informações sobre a flora original de uma área, atualmente em processo de degradação ou extinção e o subsídio para áreas da Botânica e áreas do conhecimentos afins.

# REFERÊNCIAS


AKSHANSH, J. **Getting started with TensorFlow: A Brief Introduction**. 2017, p.s.n. Disponível em <https://blog.knoldus.com/2017/11/19/getting-started-with-tensorflow-a-brief-introduction/>. Acessado em: 27 jun. 2018.

BAILEY, R. **Plant leaves and leaf anatomy**. 2016, p.s.n. Disponível em <https://www.thoughtco.com/plant-leaves-and-leaf-anatomy-373618>. Acessado em: 19 out. 2017.

BEEMAN, D. Multi-layer perceptrons (feed-forward nets), gradient descent, and back propagation. **University of Colorado**, 2001.

BENGIO, Y. et al. Learning deep architectures for ai. **Foundations and trends® in Machine Learning**, Now Publishers, Inc., v. 2, n. 1, p. 1–127, 2009.

BOOKSTEIN, F. L. Size and shape spaces for landmark data in two dimensions. **Statistical science**, JSTOR, p. 181–222, 1986.

BRANDAO, A. S. et al. Redes neurais artificiais aplicadas ao reconhecimento de comandos de voz. **Trabalho de Conclusão de Curso. Engenharia Elétrica, Universidade Federal de Viçosa–UFV–2005**, 2005.

BRIDSON, D. M. et al. Book. **The Herbarium handbook**. 3rd ed. ed. Kew : Royal Botanic Gardens, 1998. Disponível em: <http://trove.nla.gov.au/work/6140598>.

CASANOVA, D. **Identificação de espécies vegatais por meio da análise de textura foliar**. Tese (Doutorado) — Universidade de São Paulo, 2008.

CASANOVA, D. **Redes complexas em visão computacional com aplicações em bioinformática**. Tese (Doutorado) — Universidade de São Paulo, 2013.

CASANOVA, D.; JUNIOR, J. J. de M. S.; BRUNO, O. M. Plant leaf identification using gabor wavelets. **International Journal of Imaging Systems and Technology**, Wiley Online Library, v. 19, n. 3, p. 236–243, 2009.

CATSTER. **Catster**. Disponível em <https://www.catster.com/>. Acessado em: 27 jun. 2018.

COPE, J. S. et al. The extraction of venation from leaf images by evolved vein classifiers and ant colony algorithms. In: SPRINGER. **International Conference on Advanced Concepts for Intelligent Vision Systems**. [S.l.], 2010. p. 135–144.

DEWOLF, G. P. Notes on making an herbarium. **Arnoldia**, JSTOR, v. 28, n. 8-9, p. 69–111, 1968.

DOGSTER. **Dogster**. Disponível em <https://www.dogster.com/>. Acessado em: 27 jun. 2018.

DU, J. et al. Shape recognition based on radial basis probabilistic neural network and application to plant species identification. **Advances in Neural Networks–ISNN 2005**, Springer, p. 811–811, 2005.

DU, J.-X. et al. Computer-aided plant species identification (capsi) based on leaf shape matching technique. **Transactions of the Institute of Measurement and Control**, Sage Publications Sage CA: Thousand Oaks, CA, v. 28, n. 3, p. 275–285, 2006.

ELLIS, B. et al. **Manual of leaf architecture**. [S.l.]: Cornell University Press Ithaca, NY, 2009. v. 190.

FERREIRA, A. d. S. **Redes Neurais Convolucionais Profundas na Detecção de Plantas Daninhas em Lavoura de Soja**. Dissertação (Mestrado), 2017.

FILHO, M. As métricas mais populares para avaliar modelos de machine learning. 2015. Disponível em: <http://mariofilho.com/as-metricas-mais-populares-para-avaliar-modelos-de-machine-learning>.

FROSSARD, D. **VGG in TensorFlow**. 2016, p.s.n. Disponível em <https://www.cs.toronto.edu/~frossard/post/vgg16/>. Acessado em: 12 jun. 2018.

FU, H.; CHI, Z. Combined thresholding and neural network approach for vein pattern extraction from leaf images. **IEE Proceedings-Vision, Image and Signal Processing**, IET, v. 153, n. 6, p. 881–892, 2006.

GARDNERDY. **Common Types of Shrubs and Their Identification**. 2016, p.s.n. Disponível em <https://gardenerdy.com/shrub-identification-by-leaf>.

GOLDBERG, Y. A primer on neural network models for natural language processing. **J. Artif. Intell. Res.(JAIR)**, v. 57, p. 345–420, 2016.

HAYKIN, S. A comprehensive foundation. **Neural Networks**, v. 2, n. 2004, p. 41, 2004.

HINTON, G. et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. **IEEE Signal Processing Magazine**, IEEE, v. 29, n. 6, p. 82–97, 2012.

JAGREET. **Overview of Artificial Neural Networks and its Applications**. 2017. Disponível em <https://www.xenonstack.com/blog/overview-of-artificial-neural-networks-and-its-applications>. Acessado em: 21 out. 2017.

JIA, Y. et al. An open source convolutional architecture for fast feature embedding. In: **Proceedings of the 22nd ACM International Conference on Multimedia (Orlando, Florida, USA)**. [S.l.: s.n.], 2013. p. 675–678.

KOHAVI, R. et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: STANFORD, CA. **Ijcai**. [S.l.], 1995. v. 14, n. 2, p. 1137–1145.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2012. p. 1097–1105.

KUHL, F. P.; GIARDINA, C. R. Elliptic fourier features of a closed contour. **Computer graphics and image processing**, Elsevier, v. 18, n. 3, p. 236–258, 1982.

KUO, C.-C. J. Understanding convolutional neural networks with a mathematical model. **Journal of Visual Communication and Image Representation**, Elsevier, v. 41, p. 406–413, 2016.

LECUN, Y. et al. Handwritten digit recognition with a back-propagation network. In: **Advances in neural information processing systems**. [S.l.: s.n.], 1990. p. 396–404.

LEENHOUTS, P. W. et al. Book. **A guide to the practice of herbarium taxonomy**. Utrecht, Netherlands : International Bureau for Plant Taxonomy e Nomenclature, 1968. Disponível em: <http://trove.nla.gov.au/work/21525732>.

LEVINE, M. D. **Vision in man and machine**. [S.l.]: McGraw-Hill College, 1985.

LIU, J.; ZHANG, S.; DENG, S. A method of plant classification based on wavelet transforms and support vector machines. In: SPRINGER. **International Conference on Intelligent Computing**. [S.l.], 2009. p. 253–260.

LIVEEARTH, I. H. C. **Venation**. 2009, p.s.n. Disponível em <http://indiahomeclub.com/botanical_garden/bot_leaf_venation.php>. Acessado em: 19 out. 2017.

MAZUR, M. **Step by step Backpropagation Example**. 2015. <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>. Accessed: 23 nov. 2017.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943.

MCLELLAN, T.; ENDLER, J. A. The relative success of some methods for measuring and describing the shape of complex objects. **Systematic Biology**, Society of Systematic Zoology, v. 47, n. 2, p. 264–281, 1998.

MINSKY, M.; PAPERT, S. A step toward the understanding of information processes.(book reviews: Perceptrons. an introduction to computational geometry). **Science**, v. 165, p. 780–782, 1969.

MOKHTARIAN, F.; ABBASI, S. Matching shapes with self-intersections: Application to leaf classification. **IEEE Transactions on Image Processing**, IEEE, v. 13, n. 5, p. 653–661, 2004.

MORA, C. et al. How many species are there on earth and in the ocean? **PLOS Biology**, Public Library of Science, v. 9, n. 8, p. 1–8, 08 2011. Disponível em: <https://doi.org/10.1371/journal.pbio.1001127>.

NOGARE, D. Azure machine learning matriz de confusão. 2015. Disponível em: <http://www.diegonogare.net/2015/01/azure-machine-learning-matriz-de-confusao-parte-4>.

NUNES, I.; DANILO, S.; FLAUZINO, R. **Redes Neurais Artificiais para engenharia e ciências aplicadas**. [S.l.]: ArtLiber, 2010.

PARKHI, O. M. et al. Cats and dogs. In: IEEE. **Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on**. [S.l.], 2012. p. 3498–3505.

PAVLOVSKY, V. **Introduction To Convolutional Neural Networks**. 2017. Disponível em <https://www.vaetas.cz/blog/intro-convolutional-neural-networks>. Acessado em: 05 nov. 2017.

PLANTCLEF. 2016. <http://www.imageclef.org/lifeclef/2016/plant>. Accessed: 07 nov. 2017.

RASCHKA, S.; MIRJALILI, V. **Python Machine Learning**. [S.l.]: Packt Publishing Ltd, 2017.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. **Psychological review**, American Psychological Association, v. 65, n. 6, p. 386, 1958.

ROSENFELD, A.; KAK, A. Digital picture processing-volume 1, volume 2. **Computer Science and Applied Mathematics, New York: Academic Press, 1982, 2nd ed.**, 1982.

ROYER, D. L.; WILF, P. Why do toothed leaves correlate with cold climates? gas exchange at leaf margins provides new insights into a classic paleotemperature proxy. **International Journal of Plant Sciences**, The University of Chicago Press, v. 167, n. 1, p. 11–18, 2006.

ROYER, D. L. et al. Correlations of climate and plant ecology to leaf size and shape: potential proxies for the fossil record. **American Journal of Botany**, Botanical Soc America, v. 92, n. 7, p. 1141–1151, 2005.

SCHWARTZ, W. R.; PEDRINI, H. Método para classificação de imagens baseada em matrizes de coocorrência utilizando características de textura. **III Colóquio Brasileiro de Ciências Geodésicas, CuritibaPR, Brasil**, p. 1, 2003.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **arXiv preprint arXiv:1409.1556**, 2014.

SRINIVAS, S. et al. A taxonomy of deep convolutional neural nets for computer vision. **arXiv preprint arXiv:1601.06615**, 2016.

STUESSY, T. **Principles and practice of plant taxonomy**. [S.l.]: Taxonomy and plant conservation: the cornerstone of the conservation and the sustainable use of plants. Cambridge: Cambridge University Press, 2006.

SUPERDATASCIENCE. 2017. <https://www.superdatascience.com/deep-learning>. Accessed: 07 nov. 2017.

SZEGEDY, C. et al. Going deeper with convolutions. In: **The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2015.

VICENTE, A. C. A. et al. do c. 1999 florula fanerogamica das restingas do estado do para. ilha de algodoal. 1. familia turneraceae ap de candolle. **Bol. Mus. Paraense Emilio Goeldi, Bot**, v. 15, n. 2, p. 173–198, 1999.

WU, J. Introduction to convolutional neural networks. **National Key Lab for Novel Software Technology Nanjing University, China**, 2017.