UNIVERSIDADE FEDERAL DA PARAÍBA

Rodrigo Rocha Amorim

ESTUDO DE TÉCNICAS DE VISÃO COMPUTACIONAL E REDES NEURAIS APLICADAS A DETECÇÃO DE VEÍCULOS

CENTRO DE ENERGIAS ALTERNATIVAS E RENOVÁVEIS
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, 2019

Rodrigo Rocha Amorim

ESTUDO DE TÉCNICAS DE VISÃO COMPUTACIONAL E REDES NEURAIS APLICADAS A DETECÇÃO DE VEÍCULOS

Trabalho de Conclusão de Curso submetido ao Departamentode Engenharia Elétrica da Universidade Federal da Paraíba como parte dos requisitos necessários para a obtenção do título de Engenheiro Eletricista .

Orientador: Dr. Alexandro José Virgínio dos Santos.

Maio de 2019

CENTRO DE ENERGIAS ALTERNATIVAS E RENOVÁVEIS
UNIVERSIDADE FEDERAL DA PARAÍBA

Trabalho de Conclusão de Curso de Engenharia Elétrica intitulado ESTUDO DE TÉCNICAS DE VISÃO COMPUTACIONAL E REDES NEURAIS APLICADAS A DETECÇÃO DE VEÍCULOS de autoria de Rodrigo Rocha Amorim, aprovada pela banca examinadora constituída pelos seguintes professores:

Dr. Alexandro José Virgínio dos Santos
Orientador

Dr. José Mauricio Ramos de Souza Neto
Avaliador

Dr. Juan Moises Mauricio Villanueva
Avaliador

João Pessoa, 2019

Centro de Energias Alternativas e Renováveis
Universidade Federal da Paraíba

Catalogação na publicação
Seção de Catalogação e Classificação

A524e Amorim, Rodrigo Rocha.
Estudo de técnicas de Visão Computacional e Redes Neurais aplicadas a Detecção de Veículos. / Rodrigo Rocha Amorim. - João Pessoa, 2019.
65 f. : il.

Orientação: Alexandro José Virgínio dos Santos dos Santos.
TCC (Especialização) - UFPB/CEAR.

1. Processamento digital de imagens. 2. Visão computacional. 3. Redes neurais convolucionais. 4. Python. I. dos Santos, Alexandro José Virgínio dos Santos. II. Título.

UFPB/BC

RESUMO

Esse trabalhofaz os estudos de diversas técnicas de visão computacional e processamento de imagens com o intuito de aplicações dessas técnicas para a detecção de veículos numa via de tráfego. Além disso são realizadas os estudos um pouco mais aprofundados de redes neurais utilizando a arquitetura de redes neurais convolucionais, para a criação, treinamento e validação da rede, para obter dos resultados na mesma aplicação. O trabalho focou na utilização da linguagem *python* já que a mesma, tem sido bastante utilizada principalmente nas áreas que são focos desse trabalho, contendo bibliotecas para auxiliar a implementação dos algoritmos, como *OpenCV* utilizada nas aplicações de PDI e visão computacional e a biblioteca *tensorflow* bastante utilizada no treinamento e validação de redes neurais. Para a realização do treinamento da rede neural, foram utilizados conjuntos de dados da *COCO - dataset* e o conjunto de dados da *Stanford*. Por fim foram realizadas comparações entre esses métodos desenvolvidos, vale resaltar que os resultados obtidos são inferiores as técnicas mais utilizadas ultimamente, todavia tendo bastante importância para futuras aplicações.

Palavras-chave: processamento digital de imagens. visão computacional. rede neural convolucional. python.

ABSTRACT

This work has as objective the studies of several techniques of computer vision and digital image processing with the purpose of applications for the detection of vehicles in a road of traffic. In addition, was build a more in-depth study of neural networks (ANN) using the convoltional neural network architecture (CNN), for the creation, training and validation of the network, was performed to obtain the results in the same application. The work focused on the use of the python programming language since it has been used mainly in the areas that are the focus on this work, because containing libraries to help improve and implement the algorithms, such as OpenCV, that was used in PID and computer vision applications and the tensorflow library widely used in the training and validation of neural networks. For the training of the neural network, we used the dataset of the COCO dataset and the Stanford dataset. Finally, comparisons are made between these methods to characterize the work obtained, it is worth notice that the results obtained are inferior to the most techniques used lately, however having a lot of importance for future applications.

Key-words: vision computer. python. digital image processing. convolutional neural network.

LISTA DE FIGURAS

LISTA DE TABELAS

Sumário

1 Introdução

Nos ultimos anos, com o avanço da capacidade computacinal dos processadores, surgiram variás áreas de conhecimento para a resolução de problemas que até pouco tempo atrás, eram impossíveis de se resolver pelos computadores. Uma dessas áreas é a área de processamento digital de imagens, que é definida como a área de estudos que permite aprimorar informações em imagens para interpretação humana e analisar informações extraídas de uma cena por computador (MARQUES FILHO e VIEIRA NETO, 1999).

A área de processamento digital de imagens (PDI), vem apresentando expressivo crescimento, já que suas aplicações permeiam quase todos os ramos da atividade humana, como por exemplo, na medicina o uso de imagens no diagnóstico médico, utilizando técnicas que auxiliam na interpretação de imagens produzidas por equipamentos como o Raios X e outras imagens biomédicas. Já na biologia a capacidade de processar automaticamente imagens obtidas de microscópios, por exemplo contando o número de células de um certo tipo presentes em uma imagem, facilitando assim a execução de tarefas laboratoriais com alto grau de precisão e repetibilidade (MARQUES FILHO e VIEIRA NETO, 1999). Essas técnicas também podem ser empregadas no processamento e na interpretação automática de imagens captadas por satélites, auxiliando em áreas de Geografia, Sensoriamento Remoto, Geoprocessamento e Meteorologia, além de aplicações nas áreas de engenharia, como o uso de robôs dotados de visão artificial para fazer o controle de qualidade em uma linha de produção, e o uso de imagens no sistema de segurança para reconhecimental facial, dentre outros.

A visão computacional, é uma ramificação da área de processamento digital de imagens, baseada em inteligência artificial (campo de estudo referentes as técnicas computacionais que simulam a capacidade humana de raciocinar), que tem como característica a extração e reconhecimento de características de uma imagem. A visão computacional é a forma como o computador entende a imagem, sendo utilizados as técnicas de processamento de imagens para poder fazer o reconhecimento e ter a tomada de decisão sobre os objetos ou características de interesse na imagem (BALLARD, BROWN, 1982).

Nessa última década, a utilização de visão computacional cresceu muito com o advento dos poderios computacionais, com isso, também houve um crescimento no estudo de técnicas para a utilização da mesma, ultimamente tem se popularizado, o estudo na área de aprendizado de máquina (*Machine Learning*) , que é uma sub-área da inteligência artificial, que consiste em utilizar métodos computacionais para predizer e resolver problemas derivados da interpretação de conhecimentos já adquiridos, podendo ser definido como o campo de estudo que dá aos computadores a habilidade de aprender sem serem explicitamente programados (PHIL, 2013), essa área abrange uma grande variedade de técnicas para a solução de problemas complexos, tais como as técnicas de regressão, técnicas base-

adas em árvores de decisão e os algoritmos inteligêntes, como as Redes Neurais e a lógica *fuzzy*.

Desde sua concepção, os algoritmos de aprendizado de máquina, têm contribuido para o avanço de diversas áreas do conhecimento. Hoje em dia, a visão computacional foi a que mais se beneficiou com o surgimento desses métodos, principalmente pelo grande uso das redes neurais, sendo um dos campos que vêm ganhando grande importância, tanto pela forma que é desenvolvida, como pela forma que as informações estão sendo expostas.

As Redes Neurais Artificiais podem funcionar como modelos preditivos que descrevem a relação funcional entre as variáveis de entrada e variáveis de saída de um sistema. As Redes Neurais Artificiais (RNAs) têm várias vantagens sobre os modelos fenomenológicos tradicionais ou modelos empíricos. RNAs desenvolvem um mapeamento das variáveis de entrada e saída, que podem ser usados para predizer parâmetros de saída do sistema (SINGH, 2009).

Como mencionado anteriormente, as Redes Neurais Artificiais (RNA) são bastante utilizadas nos problemas de visão computacional já que agem de forma mais robustas nas aplicações em relação ao uso das técnicas de PDI em geral. A RNA que mais se destaca no ambiente de visão computacional é a CNN(Convolucional Neural Network), que tem como característica a utilizações de camada de filtros convolucionais para identificar e reduzir as áreas de interesse da imagem de entrada, resultando em uma redução do custo computacional além de ser bastente utilizada por ser invariante a rotações.

O reconhecimento de objetos continua sendo um dos grandes desafios fundamentais dos sistemas de visão computacional. Em geral, muito dos esforços estão concentrados na classificação de categorias de objetos em 2D, porém, hoje em dia, há um constante crescimento na utilização de formas 3D, fazendo necessário o desenvolvimento de novos métodos para classifica-los (MIYAZAKI, 2017). O reconhecimento de véiculos no trânsito ou em uma via de tráfego abrangem uma variedade de aplicações, tais como o reconhecimento de veículos furtados, a utilização dos dados para estudos e melhoria na mobilidade urbana, dentre outros. Por isso, se faz necessário a utilização de várias técnicas relacionadas a visão computacional para tentar tornar mais robustos e menos sucepctíveis a erros os algoritmos para essas determinadas aplicações.

1.1 Objetivo geral

Esse trabalho tem como características principais a utilização de técnicas de processamento digital de imagens e visão computacional como estudo de caso para o reconhecimento de veículos em vias de tráfego, como também, a utilização de redes neurais convolucionais, como comparação para a melhor precisão dentre eles.

1.2 Objetivos específicos

- Programar códigos na linguagem de programação *python* utilizando técnicas de processamento digital de imagens e visão computacional visando a detecção de veículos em vias de tráfego.
- Programar códigos com a finalidade de criar, treinar e verificar uma rede neural convolucional com o intuito da detecção de veículos em vias de tráfegos.
- Comparar os resultados das técnicas utilizadas.

1.3 Estrutura do Trabalho

A organização do trabalho consiste nos seguintes capítulos:

- O capítulo 2, descreve os princípios fundamentais do processamento de imagens, como também, algumas das técnicas de visão computacinal.
- O capítulo 3, descreve as características das redes neurais e as etapas de uma rede neural convolucional.
- O capítulo 4, descreve as ferramentas utilizadas para a aplicação desse trabalho.
- O capítulo 5, demonstra a implentação dos algorítimos utilizando as técnicas de PDI e visão computacional, como também a implementação da rede neural.
- O capítulo 6, descreve a comparação dos resultados.
- Por fim, o capítulo 7, traz as conclusões do referente trabalho.

2 PDI e Visão Computacional

O estudo da visão computacional, pode ser exemplificado como o estudo das técnicas de PDI que tem como finalidade a extração e o reconhecimento de certas características de interesse, portanto, para a aplicação da mesma, nesse contexto, é necessário a utilização das técnicas de PDI. As técnicas de processamento digital de imagens são técnicas que fazem transformações na imagem para melhor adequá-la para tarefas posteriores, portanto, é necessário ter conhecimentos básicos de uma imagem digitial, como também de suas operações. Nesta seção serão introduzidos esses princípios, como também, as etapas de um sistema de visão computacional.

2.1 Representação de Imagens Digitais

Uma imagem digital monocromática pode ser definida como uma função de intensidade luminosa bidimensional, $f(x, y)$, em que x e y são coordenadas espaciais e o valor da função f nessas coordenadas equivale a intensidade da imagem naquele ponto. Os valores de $f(x, y)$ tem como características serem positivos e finitos.

A geração de imagens é caracterizada por dois componentes: a iluminação $i(x, y)$, que representa a quantidade de luz incidente sobre o ponto (x, y) e a reflectância, $r(x, y)$, que representa a quantidade de luz refletida pelo ponto (x, y) (MARQUES FILHO e VIEIRA NETO, 1999). A função $f(x, y)$ é formada pela multiplicação desses dois termos:

$$f(x, y) = i(x, y) \cdot r(x, y) \tag{2.1}$$

A reflectância é limitada entre o valor 0, que equivale a absorção total da luz e 1 que equivale a reflectância total. A natureza do componente de iluminação é determinada pela fonte de iluminação, enquanto a reflectância é determinada pelas características dos objetos da cena (GONZALEZ e WOODS, 2006).

A Intensidade de uma imagem monocromática f nas coordenadas $f(x, y)$ será denominada nível de cinza ou tom de cinza (L) da imagem naquele ponto (MARQUES FILHO e VIEIRA NETO, 1999). Este valor estará no intervalo:

$$L_{min} \leq L \leq L_{max}$$

Para uma imagem digital colorida, a representação é feita de forma semelhante. Considerando o sistema de cores RGB (*Red Green Blue*), a função pode ser vista como um vetor, $f(x, y) = (f_R(x, y), f_G(x, y), f_B(x, y))$, onde cada componente com o índice mostrado representa as intensidades das cores vermelho, verde e azul, respectivamente. Portanto, uma imagem colorida pode ser vista como uma composição de três imagens monocromáticas(GONZALEZ e WOODS, 2006).

2.2 Etapas do sistema de visão computacional

Em um sistema de visão computacional, são definidos um conjunto de etapas para facilitar e reproduzir um melhor desempenho para aplicações. Dentre elas, estão: aquisição de imagens, pré-processamento, segmentação, representação e descrição, reconhecimento e intepretação.

Na Aquisição de imagens é feita a aquisição propriamente dita da imagem através de um sensor ou um dispositivo, transformando em um formato adequado para leitura da imagem. Os dispositivos mais usados para tal tarefa são: câmeras, *scanners*, satélites. Na etapa de pré-processamento são utilizados técnicas para correção de contraste ou brilho, técnicas para atenuação de ruídos e suavização, tudo isso visando melhorar a qualidade da imagem.

Na etapa de segmentação, a imagem será particionada nos objetos de interesse que a compõe, ou seja, as regiões de interesse contidas na imagem. Serão indentificadas as descontinuidades de intensidade(bordas) ou similaridades (regiões) na imagem. Na representação e descrição é realizada a extração de características de cada região obtida da segmentação. Essas informações são representadas por descritores para cada atributo, por exemplo, cor ou textura, sendo necessárias para diferenciar cada região em uma classe de objetos. Em geral, esses atributos são descritos numericamente, constituindo um vetor de características (GONZALEZ e WOODS, 2006).

O reconhecimento, como o próprio nome diz, atribui um idetificador aos objetos da imagem, enquanto o processo de interpretação irá atribuir um significado a esses objetos reconhecidos de acordo com a aplicação.

2.3 Propriedades da Imagem Digital

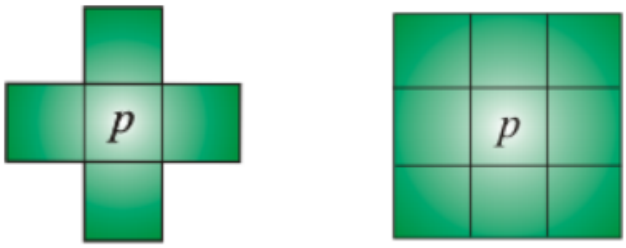
Em uma imagem digital os elementos básicos que as compões são chamados de elemento fundamental da imagem ou pixel (*pictures elements*). Uma imagem digital é uma imagem $f(x, y)$ discretizada tanto espacialmente quanto em amplitude. Portanto, uma imagem digital pode ser vista como uma matriz cujas linhas e colunas identificam um ponto na imagem, cujo valor corresponde ao nível de cinza da imagem naquele ponto (MARQUES FILHO e VIEIRA NETO, 1999). Nessa sub seção será demonstrado algumas das principais relações entre pixels numa imagem digital.

2.3.1 Vizinhança

Cada pixel pertencente a imagem é expresso por $f(x, y)$ e possui quatro vizinhos, horizontais e verticais na coordenada (x, y). Essas coordenadas são representados por: $(x + 1, y), (x - 1, y), (x, y + 1), (x, y - 1)$.

Esses pixels formam um conjunto denominado vizinhança-4 de p, designada por $N_4(p)$, cada pixel está separado por uma unidade de distância de (x, y). Se p estiver na borda da imagem, ele deixará de ter alguns vizinhos. Unindo os vizinhos diagonais com o $N_4(p)$ têm-se a chamada vizinhança-8, designada $N_8(p)$ (GONZALEZ e WOODS, 2000). Na figura 1 é mostrado esses dois tipos de vizinhanças.

Figura 1: Conceitos de Vizinhança.



(a) Vizinhança $N_4(p)$. (b) Vizinhança $N_8(p)$.

Fonte: (GONZALEZ e WOODS, 2000)

2.3.2 Conectividade

A conectividade entre pixels é um importante conceito usado para estabelecer limites de objetos e componentes de regiões em uma imagem. Dois pixels p_1 e p_2 são ditos conectados quando satisfazem dois critérios. O primeiro diz que os pixels devem ser adjacentes, ou seja, devem ser vizinhos-de-8. O segundo diz que os pixels devem atender a algum critério de classificação, digamos, seus níveis de cinza devem ser iguais (GONZALEZ e WOODS, 2000). Por exemplo, em uma imagem binária, onde os pixels podem assumir os valores 0 e 1, dois pixels podem ser 4-vizinhos, mas somente serão considerados 4-conectados se possuírem o mesmo valor.

2.3.3 Adjacência

Um pixel p_1 é adjacente a um pixel p_2 se eles forem conectados. Há tantos critérios de adjacência quantos são os critérios de conectividade. Dois subconjuntos de imagens, S_1 e S_2, são adjacentes se algum pixel em S_1 é adjacente a algum pixel em S_2 (MARQUES FILHO e VIEIRA NETO, 1999).

2.3.4 Operações Lógicas e Aritméticas

Uma imagem digital é vista como uma matriz de inteiros, e portanto, pode ser manipulada numericamente utilizando operações lógicas e aritméticas. Essas operações

podem ser descritas pela equação 2.2.

$$X \Theta Y = Z \tag{2.2}$$

Onde X e Y podem ser tanto matriz como escalares. Z é uma matriz(imagem) e Θ significa um operador aritmético ou lógico binário .

As operações aritméticas podem ser divididas em adição, subtração, multiplicação e divisão. Na operação de adição o resultado é a soma de duas matrizes ou de uma matriz por um escalar, tendo como um dos principais resultados a normalização do brilho da imagem e a remoção de ruídos. Já a subtração é o resultado da subtração da intensidade X por Y, se Y for um escalar positivo, o resultado será uma versão mais escura de X, o decréscimo de intensidade será o próprio valor de Y, essa técnica é bastante utilizada para detecção de diferenças entre duas imagens, geralmente adquiridas de forma consecutivas da mesma cena. Já a multiplicação e a divisão são aplicadas para calibração e normalização do brilho, respectivamente (MARQUES FILHO e VIEIRA NETO, 1999).

As operações lógicas ou booleanas (AND,OR,XOR,NOT), podem ser aplicadas nas imagens, essas operações podem ser aplicadas em imagens de qualquer número de níveis de cinza, porém, o melhor resultado se dá quando são aplicadas em imagens binárias.

2.4 Operações Morfológicas

A morfologia matemática, elaborada inicialmente por Georges Matheron e Jean Serra (SERRA, 1982), concentra seus esforços no estudo da estrutura geométrica das entidades presentes em uma imagem. A morfologia matemática pode ser aplicada em várias áreas de processamento e análise de imagens, com objetivos tão distintos como realce, filtragem, segmentação, deteção de bordas, dentre outras. O principio básico da morfologia matemática consiste em extrair as informações relativas à geometria e à topologia de um conjunto desconhecido (uma imagem), pela transformação através de outro conjunto completamente definido, chamado elemento estruturante. Portanto, a base da morfologia matemática é a teoria de conjuntos. Nos seguintes tópicos serão discutidas as operações de dilatação, erosão, abertura e fechamento.

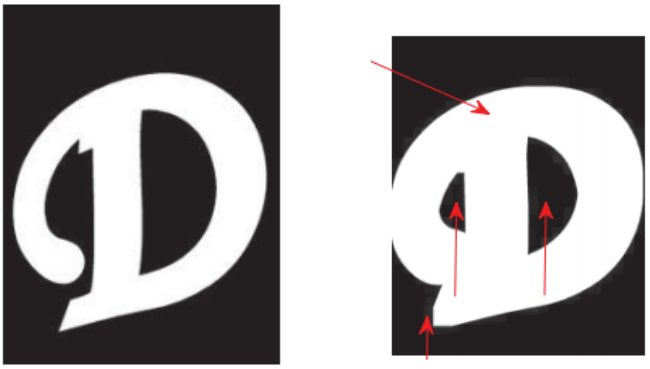
2.4.1 Dilatação

Sejam A e B conjuntos no espaço Z^2 e seja $\oslash$ o conjunto vazio. A dilatação de A por B, denotada $A \oplus B$, é definida como:

$$A \oplus B = x | (\widehat{B})_x \cap A \neq \oslash \tag{2.3}$$

A operação de dilatação tem como objetivo fazer o objeto expandir de tamanho, onde esse aumento depende do elemento estruturante, ou seja, a dilatação adiciona pixels aos contornos dos elementos, esse processo é feito aumentando o número de pixels com valor 1 da região de primeiro plano(*foreground*) e reduzindo o número de pixels com valor zero (*background*). Essa técnica é bastante utilizada para preencher os buracos (pixels ausentes) em um objeto contínuo se assemelhando a técnica de suavização (filtro passa-baixa), já que adicionar pixels no limite do objeto afeta a intensidade nesse local, resultando no efeito de desfocagem (RAVI; KHAN, 2013). A figura 2 apresenta uma aplicação de ditalação.

Figura 2: Aplicação da técnica de Dilatação.



(a) Imagem Original.

(b) Imagem com a operação de dilatação.

Fonte: (RAVI; KHAN, 2013)

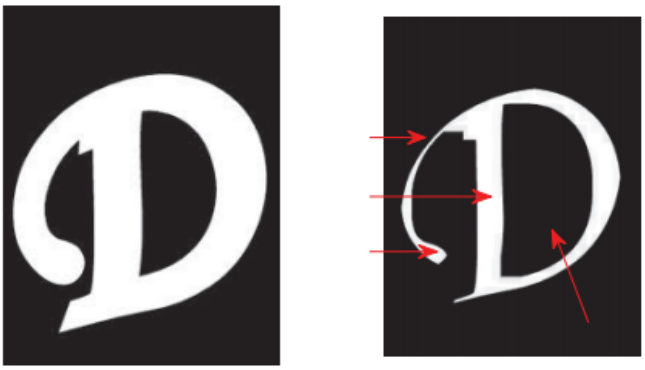
2.4.2 Erosão

Sejam A e B conjuntos no espaço Z_2. A erosão de A por B, denotada $A \ominus B$, é definida como:

$$A \ominus B = x|(\widehat{B})_x \subseteq A \tag{2.4}$$

A operação de erosão é o complemento da operação de dilatação em relação ao efeito da operação, isto é, a erosão faz com que o objeto encolha seu tamanho. Esta operação resulta em perda de pixels na borda do objeto. O processo de erosão aumenta o número de pixels com valor zero e encolhe o número de pixels com valor um. Essa operação é bastante utilizada para remoção de conexões ruidosas entre dois objetos (RAVI; KHAN, 2013). A figura 3 mostra uma aplicação de erosão.

Figura 3: Aplicação da técnica de Erosão.



(a) Imagem Original.

(b) Imagem com a operação de erosão.

Fonte: (RAVI; KHAN, 2013)

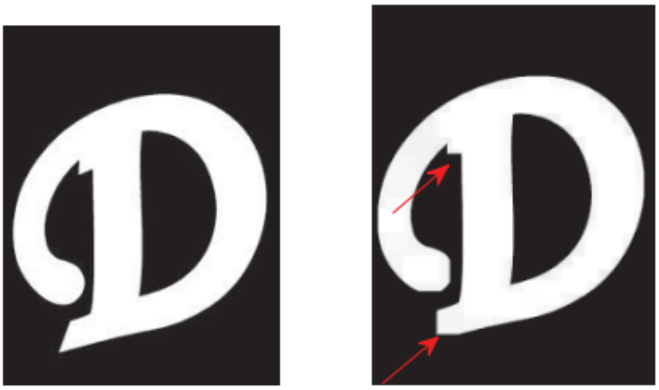
2.4.3 Abertura

A abertura de um conjunto A por um elemento estruturante B, denotada $A \circ B$, é definida como:

$$A \circ B = (A \ominus B) \oplus B \tag{2.5}$$

O que representa dizer que a abertura de A por B é simplismente a erosão de A por B seguida da dilatação do resultado por B. A abertura em geral suaviza o contorno de uma imagem, quebra istmos estreitos e elimina proeminências delgadas (extensões menores presentes no objeto) (MARQUES FILHO e VIEIRA NETO, 1999). A figura 4 mostra uma aplicação de abertura.

Figura 4: Aplicação da técnica de Abertura.



(a) Imagem Original.

(b) Imagem com a operação de abertura.

Fonte: (RAVI; KHAN, 2013)

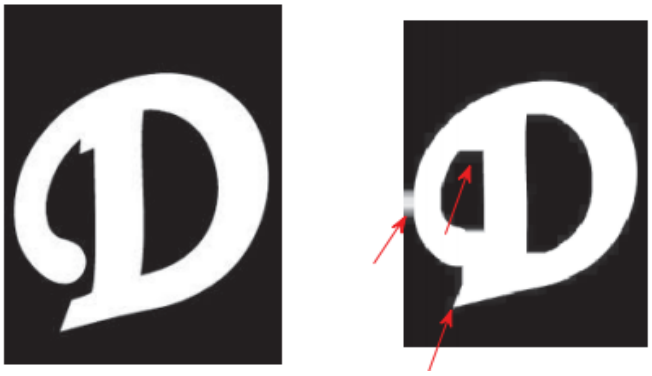
2.4.4 Fechamento

O fechamento do conjunto A pelo elemento estruturante B, denotado $A \bullet B$, é definida como:

$$A \bullet B = (A \oplus B) \ominus B \tag{2.6}$$

O que representa dizer que é a dilatação de A por B seguida da erosão do resultado pelo mesmo elemento estruturante B. O fechamento, por sua vez, funde pequenas quebras e alarga golfos estreitos, elimina pequenos orifícios e preenche *gaps* no contorno (MARQUES FILHO e VIEIRA NETO, 1999). A figura 5 mostra uma aplicação de fechamento.

Figura 5: Aplicação da técnica de Fechamento.



(a) Imagem Original.

(b) Imagem com a operação de fechamento.

Fonte: (RAVI; KHAN, 2013)

2.5 Histograma

O histograma de uma imagem é simplesmente um conjunto de números indicando o percentual de pixels naquela imagem que apresentam um determinado nível de cinza. Estes valores são normalmente representados por um gráfico de barras que fornece para cada nível de cinza o número (ou o percentual) de pixels correspondentes na imagem. Através da visualização do histograma de uma imagem obtemos uma indicação de sua qualidade quanto ao nível de contraste e quanto ao seu brilho médio, ou seja, se a imagem é predominantemente clara ou escura (MARQUES FILHO e VIEIRA NETO, 1999). Cada elemento desse conjunto é calculado pela equação 2.7.

$$p_r(r_k) = \frac{n_k}{n} \tag{2.7}$$

onde:

$0 \leq r_k \leq 1$

k = 0, 1, ..., L-1, onde L é o número de níveis de cinza da imagem digitalizada.

n = número total de pixels na imagem.

$p_r(r_k)$ = probabilidade do k-ésimo nível de cinza.

n_k = número de pixels cujo nível de cinza corresponde a k.

Para uma melhor correção e melhoria no contraste das imagens , pode-se destacar a técnica de equalização de histograma, que consiste em redistribuir os valores de tons de cinza dos pixels de uma imagem, de modo a obter um histograma uniforme, no qual o

número(percentual) de pixels de qualquer nível de cinza é praticamente o mesmo. Para isso se utiliza uma função auxiliar chamada de função de transformação. Para a equalização de histograma, a função de transformação mais comumente utilizada é a função de distribuição acumulada da distribuição de probabilidade original (MARQUES FILHO e VIEIRA NETO, 1999). , que pode ser demonstrada pela equação 2.8.

$$s_k = \sum_{j=0}^{k} \frac{n_j}{n} = \sum_{j=0}^{k} p_r(r_j) \tag{2.8}$$

Vale salientar que a função de equalização de histograma é utilizada com os valores de cinza da imagem normalizados no intervalo entre 0 e 1.

2.6 Limiarização

O princípio da limiarização consiste em separar as regiões de uma imagem quando esta apresenta duas classes (o fundo e o objeto). Devido ao fato da limiarização produzir uma imagem binária à saída, o processo também é denominado, muitas vezes, binarização. A forma mais simples de limiarização consiste na bipartição do histograma, convertendo os pixels cujo tom de cinza é maior ou igual a um certo valor de limiar (T) em brancos e os demais em pretos (MARQUES FILHO e VIEIRA NETO, 1999). Matematicamente, a operação de limiarização pode ser descrita como uma técnica na qual uma imagem de entrada $f(x, y)$ com N níveis de cinza produz uma imagem de saída $g(x, y)$, que geralmente apresenta dois níveis:

$$g(x,y) = \begin{cases} 1, & \text{se } f(x,y) \geq T \\ 0, & \text{se } f(x,y) < T \end{cases} \tag{2.9}$$

onde os pixels que recebem o valor 1 correspondem aos objetos e os pixels com o valor 0 correspondem ao fundo (background) e T é um valor de tom de cinza pré-definido, ao qual denominamos limiar. Além da limiarização através de um valor pré-definido pelo usuário, também existe na literatura a limiarização adaptativa, que consiste em utilizar varios valores de limiar, cada qual, específico para uma determinada região da imagem, onde esses valores são calculados computacionalmente utilizando janelas(matrizes) onde é computado a média dos valores dos pixels.

2.7 Filtros Espaciais

As técnicas de filtragem no domínio espacial são aquelas que atuam diretamente sobre a matriz de pixels que é a imagem digitalizada. Logo, as funções de processamento

de imagens no domínio espacial podem ser expressas como:

$$g(x, y) = T[f(x, y)] \tag{2.10}$$

onde:

$g(x, y)$ é a imagem processada.

$f(x, y)$ é a imagem original.

T é um operador conhecido como máscara espacial.

O uso de máscaras espaciais no processamento de imagens é normalmente denominado filtragem espacial, essas máscaras agem através da operação de convolução, que consiste no deslizamento das mesmas pela imagem original, computando o valor médio proveniente dessas máscaras em cada pixel central da imagem de entrada. Essas operações são amplamente utilizadas no processamento de imagens, uma seleção apropriada dessas máscaras torna possível uma vasta gama de operações, tais como redução de ruído, afinamento e detecção de características da imagem. Os principais filtros são postos a seguir:

2.7.1 Filtros de Aguçamento

Os filtros de aguçamento se assemelham aos filtros passa-altas no domínio da frequência, que tem como funcionalidade atenuar as componentes de frequências mais baixas e enfatizar as componentes de frequência mais altas. O principal objetivo das técnicas de aguçamento, é destacar os detalhes finos na imagem, com o intuito de melhorar a condição da imagem para detecções de linhas, curvas e bordas. Define-se borda como a fronteira entre duas regiões cujos níveis de cinza predominantes são razoavelmente diferentes (PRATT, 1991).

Para a detecção de bordas, aplicam-se geralmente dois tipos de filtros espaciais, o filtro baseado no gradiente da função de luminosidade $I(x, y)$ da imagem, e os filtros baseados no laplaciano de $I(x, y)$ (GONZALEZ e WOODS, 2000). Para a utilização desses filtros, são utilizados as máscaras de convolução, que geralmente são matrizes 3x3 ou 5x5. O operador Laplaciano é definido pela equação 2.11.

$$\bigtriangledown^2 f(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{2.11}$$

Uma das características do operador laplaciano é a insensibilidade a rotação, podendo realçar ou detectar bordas em qualquer direção, porém, seu uso se torna bastante restrito pela sua grande susceptibilidade ao ruído (MARQUES FILHO e VIEIRA NETO, 1999).

Por outro lado, o método mais utilizado são os filtros baseado no gradiente, onde é calculado a derivada de uma imagem discreta com a finalidade de encontrar as intensidades luminosas e a direção das bordas em uma determinada posição da imagem. O gradiente $f(x, y)$ em um certo ponto (x, y) é definido pela equação 2.12.

$$\nabla f = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \\ \dfrac{\partial f}{\partial y} \end{bmatrix} \tag{2.12}$$

No PDI as derivadas de primeira ordem são implementadas utilizando-se a magnitude do gradiente, já o ângulo ou a fase do vetor gradiente é medido em relação ao eixo x, e sempre dá a direção do crescimento da função. A magnitude e a fase do gradiente são calculados a partir das equações 2.13 e 2.14, respectivamente.

$$mag(\nabla f) = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{2.13}$$

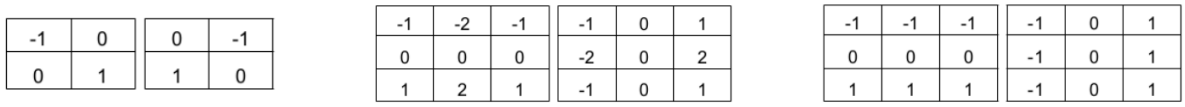
$$\alpha(x, y) = tg^{-1} \begin{bmatrix} \dfrac{\partial f}{\partial y} \\ \\ \dfrac{\partial f}{\partial x} \end{bmatrix} \tag{2.14}$$

Por serem gradiente esses filtros costumam ser utilizados em suas formas unidimensionais, isto é, buscam transições em apenas um dos eixos x ou y. Os operadores de gradiente mais conhecidos são os de Roberts, Sobel, Prewitt, e Canny.

O método de Roberts foi um dos primeiros a serem utilizados e consiste em aproximar o gradiente de uma imagem através de diferenciação discreta, que é obtida calculando a soma dos quadrados das diferenças entre os pixels adjacentes na diagonal, apresentando como resultado uma imagem com baixos valores de cinza em regiões de pouco contraste, e em uma região com contrastes bem definidos, altos valores de cinza (BARELLI, 2018).

O método de Sobel é o método que realça linhas verticais e horizontais mais escuras que o fundo, sem realçar pontos isolados. Semelhante ao Sobel, Prewitt localiza as bordas usando uma aproximação da derivada e realça as bordas no qual o gradiente da imagem é máximo (BARELLI, 2018). Na imagem 6 é apresentado as máscaras desses gradientes na direção vertical e horizontal, respectivamente.

Figura 6: Máscaras para o cálculo dos gradientes



(a) Operador de Roberts. (b) Operador de Sobel. (c) Operador de Prewitt.

O método de Canny inicialmente proposto em 1986, tinha como características a detecção de borda com baixa taxa de erro, onde a borda detectada pelo operador deve localizar com precisão o centro da borda e ter uma única resposta para os cantos de bordas encontrados, foi realizada utilizando a primeira derivada de uma gaussiana (CANNY, 1986).

De forma resumida, o operador de Canny é dividido em quatro partes: primeiro é aplicado um filtro gaussiano para suavizar a imagem e remover o ruído, depois é encontrado os gradientes de intensidade da imagem, é aplicado o operador de Sobel em ambas as direções para determinar potenciais bordas, e por fim, é aplicado um processo de histerese para verificar se o *pixel* faz parte de uma borda "forte"suprimindo todas as outras bordas ou que são fracas e não conectadas a bordas fortes (BARELLI, 2018).

2.7.2 Filtros de Suavização

Os filtros de Suavização se assemelham ao filtro passa-baixas no domínio da frequência, resultando numa resposta em que atenuam ou eliminam as componentes de alta frequência, suavizando a imagem, ou seja, provocando um leve borramento na mesma, como consequência, esse tipo de filtro geralmente altera as formas das bordas de uma imagem.

A suavização de imagens no domínio espacial baseia-se no uso de máscaras de convolução adequadas para o objetivo em questão, normalmente o borramento da imagem (para eliminar detalhes que não são de interesse para as etapas subseqüentes do processamento) ou a remoção de ruídos nela presentes (MARQUES FILHO e VIEIRA NETO, 1999). Os quatro filtros mais conhecidos são os filtros de média, mediana, gaussiano e o bilateral.

No filtro de média o valor do pixel central da máscara é igual a média dos seus vizinhos, já o filtro de mediana é um filtro de suavização não linear que ordena os pixels central e seus vizinhos e extrai o valor da mediana, o filtro da mediana é semelhante à média, mas ela despreza os valores muito altos ou muito baixos que podem distorcer o resultado (Barelli, 2018).

De forma similar, o filtro gaussiano aplica ao pixel central a média ponderada dos pixels vizinhos em que os pesos atribuídos a cada elemento da vizinhança pertencem a

uma distribuição gaussiana centrada no pixel central. O filtro bilateral utiliza-se de dois filtros gaussianos, um para a distância espacial ao redor do pixel central e seu vizinho e o outro filtro que leva em consideração a diferença de intensidade entre os pixels. O filtro bilateral é bastante utilizado por conseguir, além da remoção de ruídos e suavização da imagem, manter o formato das bordas (BARELLI, 2018).

2.8 *Haar Cascade*

Uma técnica bastante utilizada para detecção de objetos, é o *Haar Cascade*, essa técnica se baseia no algoritmo criado por Paul Viola e Michael J. Jones apresentada em 2001 no artigo *Rapid Object Detection using a Boosted Cascade of Simple Features*. Essa técnica é baseada no uso de características ao invés do uso de *pixels* diretamente. Subdividido em três partes, o algoritmo utiliza uma representação da imagem em um espaço de características, essas características são obtidas a partir do remanescente das aplicações sucessivas da transformada *wavelet* com a base de *Haar*, sendo o grande responsável pelo aumento do ganho computacional pois as operações são realizadas sobre retângulos ao invés de operar sobre *pixels* (DUARTE, 2009)

É utilizado um classificador baseado no método *AdaBoost* que é um algoritmo de boosting. O boosting é uma técnica de aprendizado de máquina que combina diversos classificadores fracos com o objetivos de melhorar a acurácia geral. O algoritmo mais famoso baseado da classe de boosting é o AdaBoost. Por fim a utilização de combinações de classificadores com o nível de detalhe e complexidade crescente. A abordagem em cascata permite que os métodos apresentados sejam aplicados de maneira mais inteligente, contribuindo para uma alta taxa de detecção (DUARTE, 2009).

3 Redes Neurais Artificias

Durante muito tempo, o estudo do cérebro humano tem demonstrado bastante curiosidade e atraído a atenção de vários cientistas nas mais diversas áreas de conhecimento, buscando compreender os mecanismos responsáveis por comportamentos tão variados e complexos e como consequência modelar e construir sistemas artificais capazes de reproduzir algumas funções semelhantes a do cérebro. O estudo na área de Redes Neurais Artificiais(RNAs), tem sido motivado pelo fato do cérebro humano processar informações de uma forma inteiramente diferente de um computador convencional.

A busca por um modelo computacional que simule o funcionamento das células do cérebro data dos anos 40, com o trabalho de de McCulloch e Pitts (1943). O entusiasmo pela pesquisa neste campo cresceu durante os anos 50 e 60. Nesse período, Rosenblatt (1958) propôs um método inovador de aprendizagem para as redes neurais artificiais denominado percepton. Até 1969, muitos trabalhos foram realizados utilizando o percepton como modelo. No final dos anos 60, Minsky e Pappert (1969) publicam um livro no qual apresentam importantes limitações do perceptron. As dificuldades metodológicas e tecnológicas, juntamente com os ataques extremamente pessimistas de Papert e Minsky, fizeram com que as pesquisas arrefecessem nos anos seguintes. Durante os anos 70, a pesquisa contava apenas com um número ínfimo de cientistas. Porém, durante os anos 80, o entusiasmo ressurge graças a avanços metodológicos importantes e ao aumento dos recursos computacionais disponíveis (HAYKIN, 2007).

O objetivo principal do estudo de RNAs é motivado pela solução de problemas complexos de maneira eficiente, baseado no processamento do cérebro. Uma rede neural é um processador que trabalha de forma paralela, possuindo a capacidade de armazenar determinados conhecimentos e de torna-los disponíveis para uso.

As redes neurais artificais atualmente possuem aplicações nas mais diversas áreas de conhecimento, muito disso se dá pela grande expansão nos últimos anos, já que tem como uma das suas principais características a habilidade de aprender a partir de dados de entrada sem a necessidade de um "professor".

As RNAs são bastante utilizadas em tarefas que incluem classificação (decidir a qual grupo pertence uma determinada entrada), Reconhecimento de Padrões (identificar padrões em entradas), predição (predizer algo a partir de certas características como, por exemplo, identificar doenças a partir de sintomas, causas a partir de efeitos), otimização (buscar o melhor resultado) e filtragem de ruído (separar partes irrelevantes de um sinal) (LUGER, 2004).

As vantagens do uso de RNAs se dá na sua capacidade de processar em paralelo e na sua habilidade de aprender e portanto generalizar. A generalização se refere a rede

neural produzir saídas adequadas para entradas que não estavam presentes durante o treinamento. Essas vantagens de processamento de informação tornam possíveis para as redes neurais solucinar problemas complexos que hoje em dia são extremamente complicados de resolver (HAYKIN, 2007).

Daqui em diante será retratado um pouco sobre as principais características das redes neurais, incluindo sua natureza biológica, o desenvolvimento do neurônio artificial, a arquitetura e o paradigma de aprendizagem.

3.1 Redes Neurais Biológicas

O Sistema Nervoso é uma rede formada por um número muito grande de células, com a capacidade de receber, transmitir, elaborar e armazenar informações através de impulsos eletroquímicos. Onde recebe informações diversas sobre mudanças no meio externo, relacionando o indivíduo com o ambiente, iniciando e regulando as respostas adequadas. O cérebro humano é considerado um sistema de processamento de informações extremamente complexo, não linear e paralelo, capaz de organizar suas estruturas, conhecidas como neurônios, para realizar processamentos a uma velocidade tão alta que nenhum computador existente consegue alcançar (HAYKIN, 2007).

Os neurônios são células nervosas que possuem uma morfologia bastante complexa com a propriedade de enviar e receber estímulos. De forma resumida, os neurônios tem como seus elementos principais: o corpo celular, que representa a parte central da célula nervosa onde é responsável pelo processamento dos sinais recebidos de outros neurônios, os dentritos que são prolongamentos numerosos com a função de receber estímulos de outros neurônios, o axônio que é responsável na condução de impulsos que transmitem informações dos neurônios para outras células, e por fim as sinapses que são os pontos de contatos entre os axiônios de um neurônio com os dentritos de outro neurônio e é a partir dela que os sinais são transmitidos de um neurônio para outro.

As redes neurais Artificiais se assemelham as redes neurais biológicas tentando reproduzir algumas de suas funções, já que são baseadas em processamento paralelo e distribuído onde se comunicam através de sinapses. De forma análoga ao cérebro, o conhecimento da rede é adquirido pelo processo de aprendizagem, onde a intensidade da conexões (pesos sinápticos) entre as unidades de processamento (neurônios) são utilizadas para armazenar o conhecimento adquirido pela rede(GLENDA, 2007).

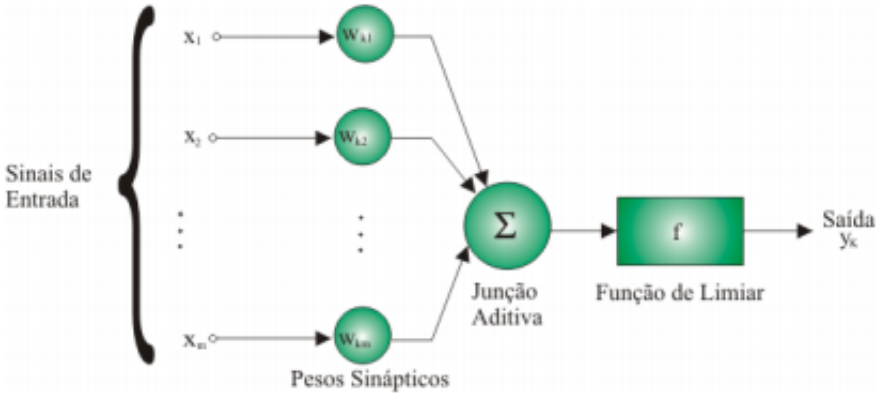
3.2 O Neurônio Artificial

Do mesmo modo que o cérebro humano é composto por diversos neurônios, uma RNA é composta por unidades que simulam o funcionamento dos neurônios, estas unida-

des são chamadas de neurônios artificiais. Os neurônios artificiais são redes hierárquicas conectadas, com as saídas de alguns neurônios sendo as entradas para os outros.

O primeiro modelo matemático de um neurônio foi derivado das propriedades fisiológicas dos neurônios biológicos e de suas conexões. Este modelo foi proposto por McCulloch e Pitts em 1943 e interpreta o neurônio como sendo um circuito binário (HAYKIN, 2001), como pode ser visto na figura 7.

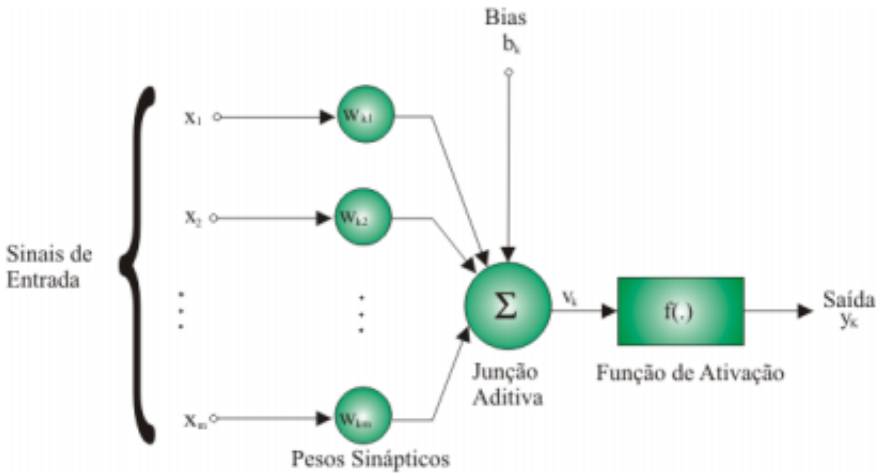
Figura 7: Modelo de McCulloch e Pitts.



Fonte: (HAYKIN, 2001)

Neste modelo as entradas x_m do neurônio k, que também são binárias são combinadas com os respectivos pesos w_{km}, através de uma soma ponderada $\sum w_{km}x_m$ para gerar a entrada da função de limiar, ou função de ativação. Se a soma ponderada das entradas de um neurônio ultrapassar um determinado limiar, então a saída toma um determinado valor, se não ultrapassar toma o valor zero. (McCULLOCH and PITTS, 1943). O modelo de McCulloch e Pitts foi generalizado, resultando no modelo apresentado na figura 8.

Figura 8: Representação do Neurônio Artificial generalizado.



Fonte: (HAYKIN, 2001)

O modelo generalizado do neurônio é composto por quatro unidades: Os sinais de entradas x_m que são os dados vindos do ambiente ou de outros neurônios, um conjunto de sinapses, cada uma caracterizada por um peso w_{km} próprio que representa o conhecimento adquirido pela RNA, uma junção adtiva que tem como intuito fazer a soma ponderada das entradas para calcular o nível de ativação dos neurônios, uma função de ativação f que limita os valores de saídas para um intervalo finito, e uma entrada denominada de Bias b_k que é usada para aumentar (se o valor for positivo) ou diminuir (se o valor for negativo) a entrada da função de ativação causando um deslocamento no potencial de ativação do neurônio. O neurônio k do modelo generalizado pode ser descrito matematicamente pelas equações abaixo:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{3.1}$$

$$v_k = u_k + b_k \tag{3.2}$$

$$y_k = f(v_k) \tag{3.3}$$

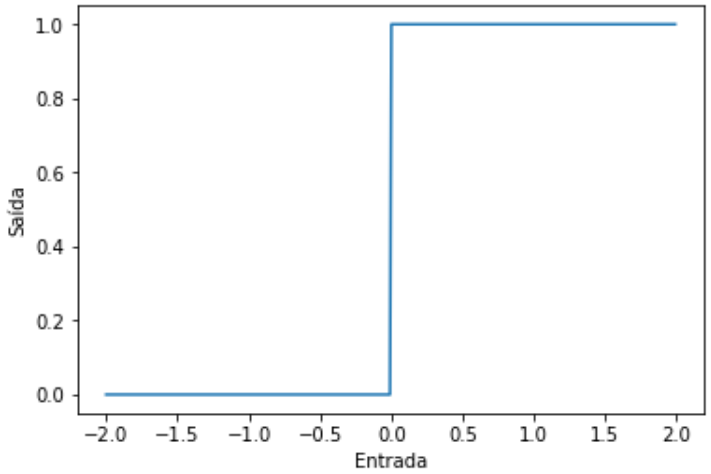
Um neurônio artificial pode utilizar diferentes tipos de funções de ativação, dentre elas as mais básicas são: função de limiar, função linear por partes e a função sigmóide (HAYKIN, 2001).

A função de limiar, representada pela equação abaixo, é usada no modelo de Mc-Culloch e Pitts e descreve a propriedade deste neurônio. De acordo com essa função y_k

o neurônio será igual a 1 se o valor de ativação for maior que 0, e será 0 caso o valor de ativação v for menor que 0, simulando uma função degrau. A figura 9 representa o gráfico desta função.

$$f(v) = \begin{cases} 1, & \text{se } v \geq 0 \\ 0, & \text{se } v < 0 \end{cases} \tag{3.4}$$

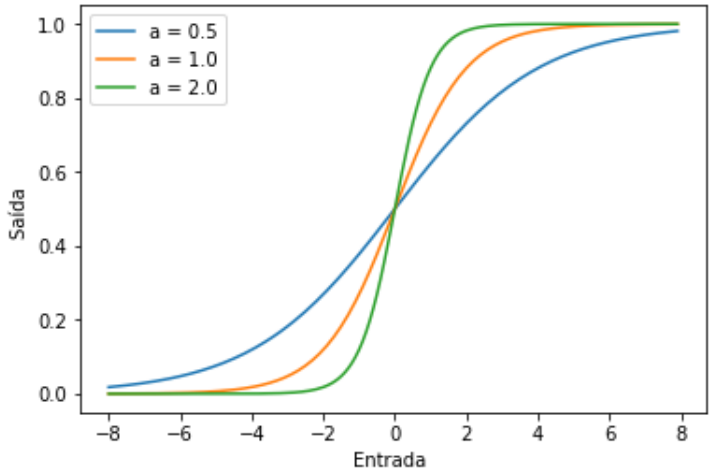
Figura 9: Gráfico da Função de Limiar.



Já na função de ativação sigmóide, cujo o gráfico tem a forma de S, admitirá valores de saída entre 0 e 1. Um exemplo da função sigmóide é a função logística, definida pela equação abaixo e apresentada pela figura 10. Nesta função a variável a, conhecida como parâmetro de achatamento, determina a inclinação da função, podendo ser negativa ou positiva.

$$f(v) = \frac{1}{1 + e^{-av}} \tag{3.5}$$

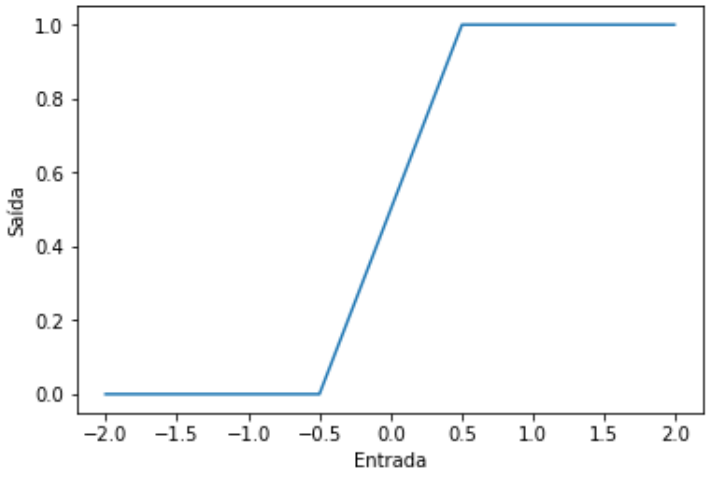
Figura 10: Gráfico da Função de Ativação Sigmóide.



Já na função linear por partes, a saída do neurônio será igual ao valor de ativação v se o valor estiver dentro de um certo intervalo, mas, se o valor v for maior ou igual ao valor máximo do intervalo a saída será 1 e se for menor ou igual ao valor mínimo do intervalo a saída será 0. A equação abaixo é um exemplo de função linear por partes, a figura 11 representa o gráfico dessa equação.

$$f(v) = \begin{cases} 1, & \text{se } v \geq +\frac{1}{2} \\ v + \frac{1}{2}, & \text{se } -\frac{1}{2} < v < \frac{1}{2} \\ 0, & \text{se } v \leq -\frac{1}{2} \end{cases} \tag{3.6}$$

Figura 11: Gráfico da Função Linear por Partes.

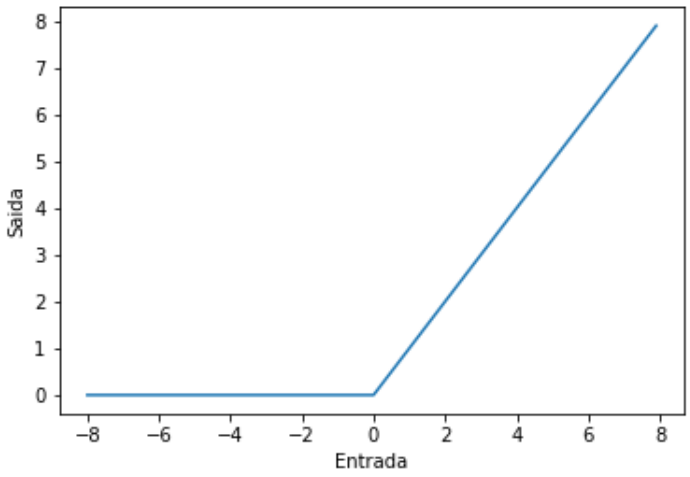


Um outro tipo de função linear por partes, é a função de ativação linear retificada (Relu), é o tipo de função mais utilizada no contexto de aprendizado de máquina e

aprendizado profundo, já que tem como característica a não utilização de expoente, sendo assim, mais eficiente que a função sigmóide. Sendo representada pela equação 3.7, a figura 12 representa o gráfico dessa equação.

$$f(v) = max(0, v) \tag{3.7}$$

Figura 12: Gráfico da Função Linear Retificada.



A escolha da função de ativação dependerá de como a RNA será executada, já que o neurônio é a unidade básica para o processamento da rede e deve ser modelado para atender os requisitos de determinada aplicação, vale salientar que uma Rede Neural possui mais de um neurônio, no qual devem ser conectados de uma forma a garantir o melhor desempenho. A forma de conexão entre os neurônios é definida pela arquitetura da rede, que é um dos elementos principais para a criação de uma RNA eficiente.

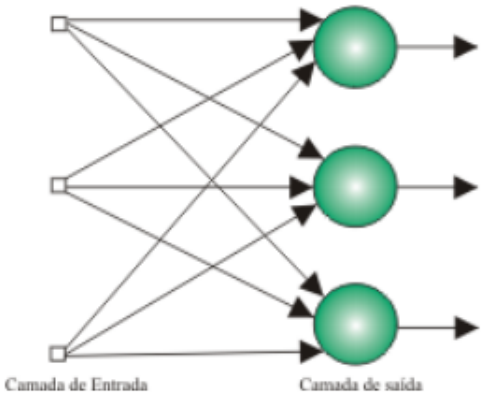
3.3 Arquitetura das Redes Neurais

A arquitetura de uma rede neural determina o modo como os neurônios estão organizados. Está associada com o algoritmo de aprendizagem usado para treinar a rede, com sua aplicação. A estrutura utilizada, exerce uma forte influência no desempenho da rede, de tal forma que uma escolha errônea pode acarretar falhas ou até mesmo a incapacidade de resolver dado problema. Pode ser diferenciada de acordo com o número de camadas e a disposição das conexões (GLENDA, 2007).

Geralmente as RNAs possuem uma camada de entrada, usada para receber os sinais ou dados externos, zero ou mais camadas intermediárias, conhecidas como camadas escondidas, e uma camada de saída que é responsável por emitir uma resposta para um certo dado de entrada (HAYKIN, 2001). Tem como principais exemplos de RNAs

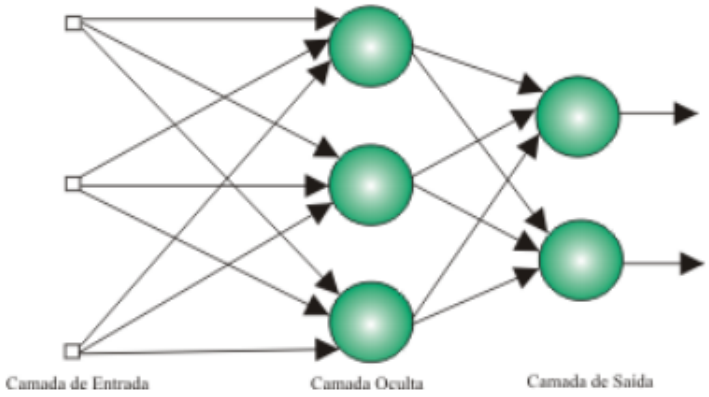
a Perceptron de Camada Única e a Perceptron multicamada (MLP) exemplificadas nas figuras 13 e 14, respectivamente.

Figura 13: Rede Neural conectada com camada única.



Fonte: (HAYKIN, 2001)

Figura 14: Rede Neural conectada com uma camada oculta.



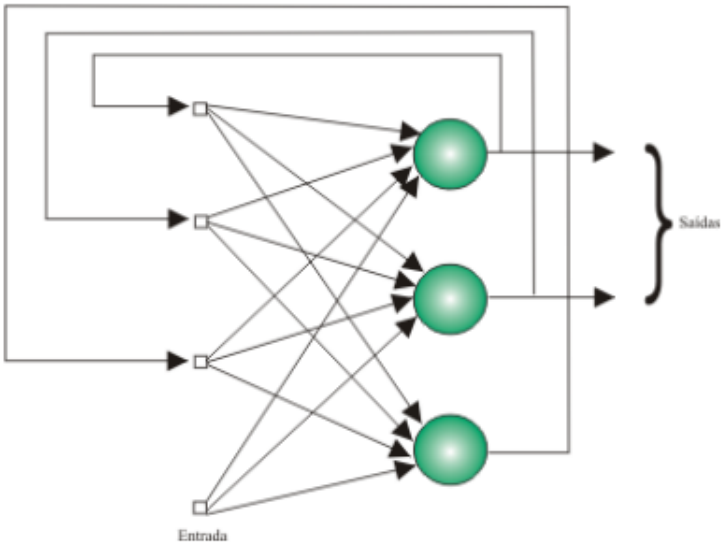
Fonte: (HAYKIN, 2001)

De acordo com as formas de conexões, as RNAs podem ser denominadas de completamente conectadas, ou seja, quando cada neurônio de uma camada se conecta com todos os neurônios da camada posterior, ou parcialmente Conectadas, quando nem todos os neurônios da rede estão conectados (MARTINELLI, 1999). De acordo com o tipo de conexão as RNAs podem ser classificadas em redes alimentadas adiantes (*feed-forward*) e as redes recorrentes(*feedback*).

Nas Redes *feed-forward*, cada neurônio de uma camada de neurônios recebe suas entradas da camada anterior e envia seus sinais de saída para os neurônios da camada

posterior, como exemplo, pode se destacar a rede neural convolucional, bastante utilizada no aprendizado profundo, sendo o tipo de rede utilizada neste projeto onde será discutida em uma seção posterior. Nas redes recorrentes pelo menos um de seus neurônios apresenta conexões sinápticas que enviam informações para alguma camada anterior a sua, de forma que seus impulsos podem realimentar a rede e possivelmente suas próprias entradas (ÓSORIO and VIEIRA, 1999), como pode ser visto na figura 15.

Figura 15: Rede Neural Recorrente.



Fonte: (HAYKIN, 2001)

A arquitetura de uma rede também pode ser classificada de acordo com a sua evolução no decorrer de sua utilização e do seu aprendizado. Baseado nestes critérios tem-se as redes estáticas e as redes dinâmicas (ÓSORIO and VIEIRA, 1999). Nas redes estáticas a quantidade de neurônios e as formas de conexão não se alteram durante o aprendizado da rede, as únicas mudanças que ocorrem são os ajustes dos pesos sinápticos. Já nas redes dinâmicas a quantidade de neurônios e conexões pode variar no decorrer do aprendizado da rede. Essa variação pode ser do tipo generativa (criação de novas conexões) ou destrutiva (eliminação de conexões).

3.4 Paradigmas de Aprendizagem

O paradigma de aprendizagem determina a forma em que a rede neural se relaciona com o ambiente, sendo divididas em: aprendizagem supervisionada, aprendizagem por reforço e aprendizagem não supervisionada. Na aprendizagem supervisionada, para o treinamento são fornecidos amostras de dados tanto de entrada como de saída, onde o aprendizado é realizado ajustando os pesos das conexões da rede com a intenção de

minimizar o erro(ÓSORIO and VIEIRA, 2009). O erro é a diferença entre a saída obtida pela rede e a saída desejada e é calculado passo a passo, interativamente, até obter um valor satisfatório para a saída específica.

Na aprendizagem por reforço, o mapeamento de entradas e saídas é feito gradativamente através da interação com o ambiente, porém, não tendo indicações precisas sobre o resultado da rede, impossibilitando assim, o cálculo do erro (HAYKIN, 2001). Já na aprendizagem não supervisionada não se usa exemplos de entradas e saída para treinar a rede. Esta aprende sozinha, sem um supervisor externo para indicar os erros. Este tipo de rede desenvolve a habilidade de extrair características relevantes, a partir das entradas fornecidas, e formar representações internas que codifiquem essas características, criando assim, novas classes automaticamente (ÓSORIO and VIEIRA, 1999).

3.5 Algoritmos de Aprendizagem

Uma das principais características das redes neurais é sua capacidade de aprender e melhorar seu desempenho gradativamente a partir da aprendizagem. Uma rede neural aprende acerca do seu ambiente através de um processo interativo de ajustes aplicados a seus pesos sinápticos e seus níveis de bias, tornando mais instruída sobre o seu ambiente após cada interação (HAYKIN, 2001).

Algoritmo de aprendizagem é um conjunto previamente estabelecido de regras bem definidas para a solução de um problema de aprendizagem. Existem diferentes tipos de aprendizagem, onde são diferenciados pela forma como é formulado o ajuste dos pesos sinápticos de um neurônio, cada qual oferencendo vantagens específicas (HAYKIN, 2001). Os aprendizados podem ser classificados em: aprendizagem Competitiva, aprendizagem de Boltzmann, aprendizagem Hebbiana, aprendizagem baseado em memória e o aprendizado por correção de erro. Para a aplicação desse projeto, foi utilizada a aprendizagem por correção de erro, que será descrita a seguir.

3.5.1 Aprendizado por Correção de Erro

Nesta forma de aprendizado, o ajuste dos pesos sinápticos são realizados para se obter o menor erro possível. Dado um neurônio k, onde sua saída é representada por $y_k(n)$, onde n representa o passo de um processo interativo no ajuste dos pesos sinápticos, o erro é calculado comparando a saída $y_k(n)$ com uma resposta desejada ou saída alvo, representada por $d_k(n)$, como apresentado na equação 3.8.

$$e_k(n) = d_k(n) - y_k(n) \tag{3.8}$$

No decorrer do aprendizado os erros são calculados iterativamente até a rede al-

cançar um estado onde os pesos sinápticos estão estabilizados de acordo com um valor previamente definido. Os pesos são ajustados visando aproximar o valor de saída obtido pela rede ao valor desejado (MENDES and OLIVEIRA 2006). Essa aproximação é alcançada através através de uma função de custo ou índice de desempenho, definidos em termos do erro, como:

$$\xi(n) = \frac{1}{2}e_k^2(n) \tag{3.9}$$

Onde, $\xi(n)$ é o valor instântaneo da energia do erro.

A minimização da função de custo resulta na regra de aprendizagem normalmente conhecida como regra delta, que pode ser representada da seguinte maneira:

$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n) \tag{3.10}$$

Onde $w_{kj}(n)$ representa o valor do peso sináptico $w_{kj}(n)$ do neurônio k excitado por um elemento $x_j(n)$e η é uma constante positiva que determina a taxa de aprendizado, quanto menor for essa taxa de aprendizado, menor serão as variações dos pesos sinápticos, de uma iteração para outra , e mais suave será a trajetória no espaços de pesos, consequentemente, uma trajetória lenta. Por outro lado, uma taxa de aprendizado muito grande, acelera o aprendizado podendo até acarretar em grandes modificações tornando a rede instável (Haykin, 2001). Após o cálculo do ajuste, o valor atualizado do peso sináptico é dado por:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n) \tag{3.11}$$

Onde o primeiro e o segundo termo da equação, se refere ao valor novo e antigo do peso sináptico, respectivamente, e o terceiro elemento indica a correção aplicada ao determinado peso.

O objetivo do processo de aprendizagem é ajustar os parâmetros livres da rede para minimizar a energia média do erro, que pode ser apresentada pela equação 3.12.

$$\xi_{med} = \frac{1}{N}\sum_{n=1}^{N}\xi(n) \tag{3.12}$$

onde N representa o número total de exemplos contidos em um dado conjunto de treinamento e ξ_{med} representa a função de custo como uma medida de desempenho de aprendizagem para esse conjunto. Essa minimização é feita calculando a função gradiente, ou seja, a derivada da função do custo, que pode ser representada pela equação 3.13.

$$\delta(n) = -\frac{\partial \xi(n)}{\partial v_j(n)} = e_j(n) f_j'(v_j(n)) \tag{3.13}$$

Onde o gradiente $\delta(n)$ para o neurônio de saída j corresponde ao produto do sinal de erro $e_j(n)$ pela derivada da função de ativação associada ao neurônio de saida j e o sinal negativo representa a descida do gradiente no espaço dos pesos.

Um dos algoritmos de aprendizagem mais popularmente conhecidos é o algoritmo de retropropagação de erro (*error back propagation*) onde o mesmo é baseado na regra de aprendizagem por correção de erro, sendo assim, um dos pilares para a construção de uma rede neural que será descrito a seguir.

3.5.2 Algoritmo de Retropropagação de Erro

Esse tipo de algoritmo consiste em dois passos através das diferentes camadas da rede: um passo para a frente, chamado de propagação, e um passo para trás, chamado de retropropagação. No passo para frente, seria o processo habitual feito por uma rede neural do tipo perceptrons de múltiplas camadas, um padrão de atividade (vetor de entrada) é aplicado aos nós sensoriais da rede e seu efeito se propaga através da rede, camada por camada, onde um conjunto de saídas é produzido como resposta real da rede. Durante esse passo, todos os pesos sinápticos são fixos. Já no passo para trás, os pesos sinápticos são todos ajustados de acordo com a regra de correção de erro. Sendo assim, a resposta real da rede é subtraída de uma resposta desejada, ou resposta alvo, para produzir um sinal de erro. Este sinal de erro é então propagado através da rede, contra a direção das conexões sinápticas. Os pesos sinápticos são ajustados para fazer com que a resposta real da rede se mova para mais perto da resposta desejada, em um sentido estatístico (HAYKIN, 2001).

Como explicado antes, o algoritmo se baseia na minimização do erro através da regra delta (equação 3.10), ou seja, o erro será minimizado a partir dos cálculos dos gradientes, para então, ter os valores dos pesos sinápticos atualizados. Para o caso do algoritmo de retropropagação, o cálculo do gradiente se divide para os dois tipos de passos, já que no caso do passo para frente, o erro é calculado para o neurônio na camada de saída, e no passo para trás, é efetuado o cálculo para o neurônio j estando na camada oculta, as equações 3.14 e 3.15 representam o cálculo dos respectivos gradientes, respectivamente.

$$\delta_k = e_k(n) f'_k(v_k(n)) \tag{3.14}$$

$$\delta_j(n) = f'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \tag{3.15}$$

Onde o fator $f'_j(v_j(n))$ depende apenas da função de ativação associado ao neurônio oculto j. O somatório sobre k depende de dois conjuntos de termos. O primeiro conjunto,

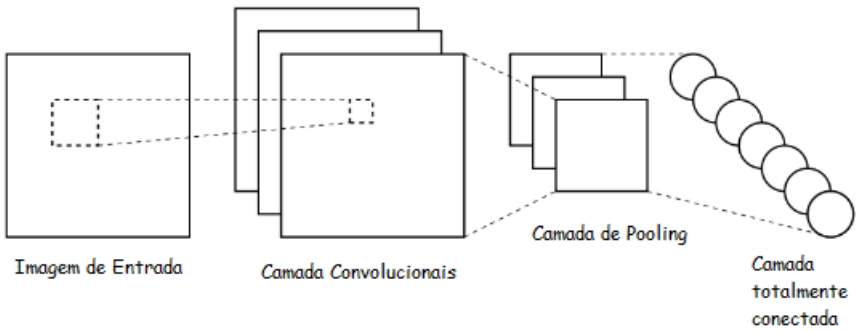
os gradientes $\delta_k(n)$, necessita do conhecimento dos sinais de erros $e_k(n)$ para todos os neurônios que se encontram na camada imediatamente à direita do neurônio oculto j, ou seja, a camada de saída e que estão diretamente conectados ao neurônio j, já o segundo conjunto de termos, os $w_{kj}(n)$ consiste nos pesos sinápticos associados com estas conexões (HAYKIN, 2001).

3.6 Rede Neural Convolucional

As Redes Neurais Convolucionais (*Convolucional Neural Network - CNN*) são arquiteturas do tipo perceptron de múltiplas camadas extremamente utilizadas no aprendizado profundo pois tem como características a sub divisão dos dados para tentar extrair as características de cada conjunto. São bastante utilizadas em áreas como processamento de fala, segmentação e classificação de imagens e análise de vídeos (SIMONOVSKY; KOMODAKIS, 2017), já que tem como característica importante a sua invariância a escala, translação e outras transformações, podendo assim, reconhecer padrões de forma mais robusta.

As Redes Neurais Convolucionais são divididas em três partes, a camada convolucional, responsável por extrair características, a camada de *polling* que tem a finalidade de reduzir a dimensão da rede, e a camada totalmente conectada, que liga todas as saídas da camada anterior e determina a saída da rede através de funções de ativação, a CNN pode ser representada pela figura 16.

Figura 16: Representação de uma Rede Neural Convolucional



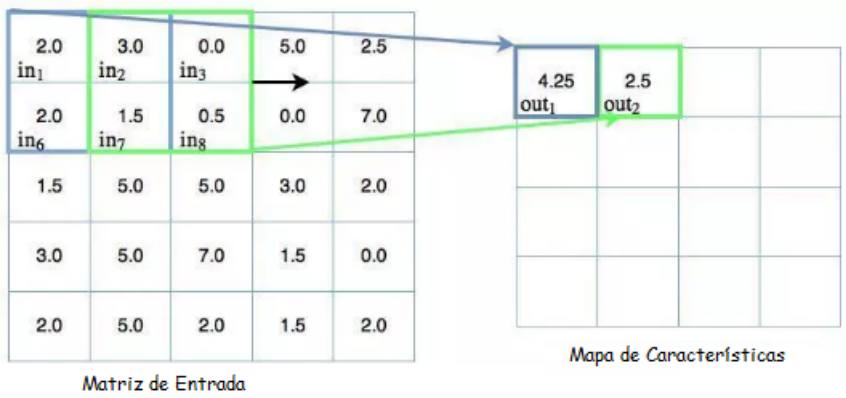
Fonte: Imagem adaptada do site:
https://www.vaetas.cz/posts/intro-convolutional-neural-networks/

3.6.1 Camadas Convolucionais

As camadas convolucionais são conjuntos de filtros móveis que percorrem sequencialmente os dados de entrada resultando em matrizes menores chamadas de mapas de

características, como mostrado na figura 17.

Figura 17: Representação da camada convolucional.



Fonte: imagem adaptada do site:
https://adventuresinmachinelearning.com/convolutional-neural-networks-tutorial-tensorflow/

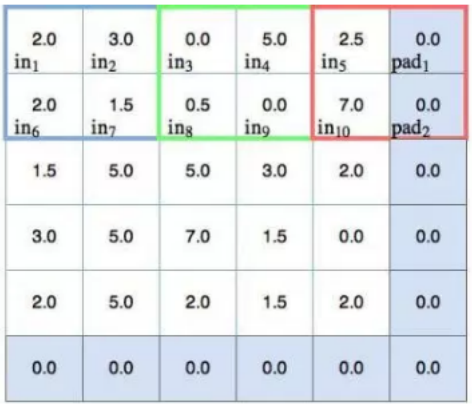
Na figura os filtros detém de uma matriz 2x2, onde os valores do vetor de entrada são somados e posteriormente multiplicados por um peso constante resultando em uma matriz de valores menor, denominada mapa de características.

Durante o processo de treinamento, esses filtros são ajustados automaticamente para que sejam ativados na presença de características relevantes, como detecção de bordas (KARPATHY, 2017). Em cada camada convolucional vários filtros são utilizados, e os mapas de características produzidos são empilhados formando uma matriz 3D para uma imagem 2D (monocromática) ou uma matriz 4D para imagem 3D (colorida). Além disso, cada camada convolucional varia as suas dimensões conforme os dados percorrem a CNN. Estas variações ocorrem devido a mudança de dois parâmetros no decorrer do processo: o passo dos filtros, mais conhecidos como *stride* e o preenchimento na camada que sofrerá a convolução, denominado *padding* (MIYAZAKI, 2017).

Os *stride* são os passos percorridos pelos filtros na matriz, passando de pixel por pixel numa imagem ou de posição por posição em uma matriz. As dimensões da matriz, largura e altura, dependem do tamanho do passo, se o passo for igual a 1, é mantido as dimensões da matriz, se o passo for igual a 2, a saída possuirá metade do tamanho da entrada.

O (*padding*) é o preenchimento das bordas da entrada em relação aos filtros, a forma mais comum utilizada é conhecida por *same padding* onde as fronteiras são preenchidas com o valor 0, de modo que não interfere no resultado do filtro, matendo as dimensões desejáveis. A figura 18 mostra o processo.

Figura 18: Preenchimento das bordas (*Padding.*)



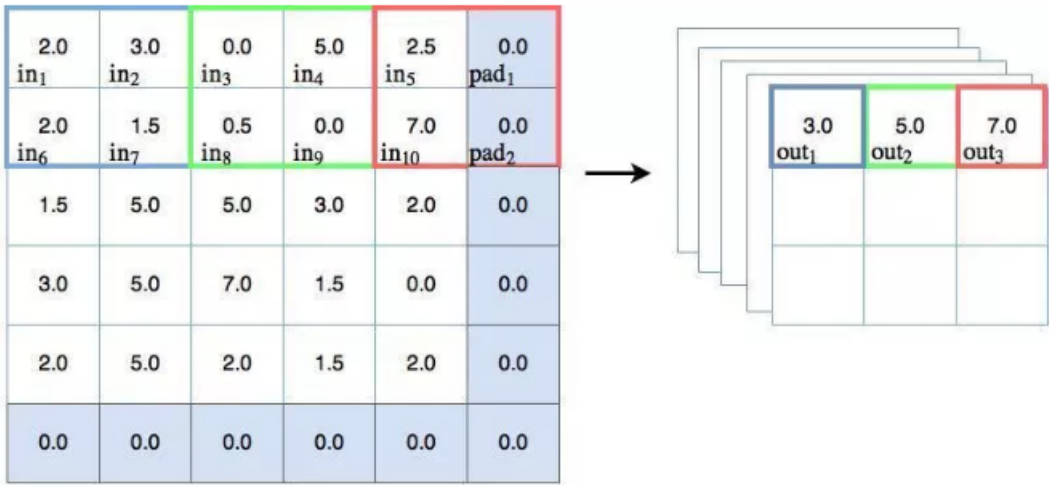
Fonte: imagem adaptada do site: https://adventuresinmachinelearning.com/convolutional-neural-networks-tutorial-tensorflow/

Como é possível comparar com a imagem 17, a imagem 18 preencheu as fronteiras com o valor zero para tornar o passo do filtro igualitário.

3.6.2 Camada de Pooling

A camada de *polling* é utilizada após a camada convolucional com o objetivo de reduzir a dimensão da camada de entrada, com a finalidade de diminuir o custo computacional evitando o problema de *overfitting* (quando o treinamento se torna muito complexo para novos vetores de entrada). O método mais utilizado é o *max polling*, onde consiste na redução das dimensões da camada pegando o máximo valor da região. Implicando na eliminação de valores desprezíveis, criando uma invariância a pequenas mudanças e distorções locais (ARAÚJO; CARNEIRO; SILVA, 2017). Um exemplo é demonstrado na figura 19.

Figura 19: Representação da camada de *polling*.



Fonte: imagem adaptada do site:
https://adventuresinmachinelearning.com/convolutional-neural-networks-tutorial-tensorflow/

3.6.3 Camada Totalmente Conectada

As camadas totalmente conectadas (*Fully Connected Layer - FCL*) são camadas onde todos os neurônios da camada anterior estão conectados com cada neurônio desta camada. O propósito dessas camadas é fazer a classificação em relação ao objeto de interesse, ou seja, coloca-se um classificador de uma rede neural convencional no final de um detector de objetos já treinado e nessa camada utiliza a função de ativação para predizer a classe do objeto(MIYAZAKI, 2017).

4 Ferramentas de Software

As principais ferramentas de software que serão utilizadas para a realização do projeto são: a linguagem de programação *Python*, a biblioteca de processamento de imagens e visão computacional *Opencv* e a biblioteca *Tensorflow*, utilizada para o treinamento e teste da rede neural.

4.1 Linguagem de Programação Python

A linguagem de programação Python, é uma linguagem de alto nível, interpretada, imperativa e orientada a objetos. Foi criada em 1991, pelo holandês Guido Van Rossum com a finalidade de ser uma ferramenta robusta e capaz de realizar tarefas complexas, e ao mesmo tempo ser de fácil manipulação e aprendizado (Borges, 2014). A linguagem foi projetada com a filosofia de enfatizar a importância do esforço do programador sobre o esforço computacional, priorizando a legibilidade do código sobre a velocidade ou expressividade.

Com a característica marcante de um ambiente de desenvolvimento mais legível e limpo, pela falta de regras e sintaxes comparando-a com a linguagem C, é muito utilizada como meio de inserção de pessoas no mundo da programação, onde a pequena quantidade de imposições de sintaxe ajuda ao estudante a adquirir uma noção geral de como é programar, e estender esta noção para outras linguagens (BOGDANCHIKOV; ZHAPAROV; SULIYEV, 2013).

Por essas características, a linguagem se tornou bastante popular, e vem sendo bastante utilizada nos últimos anos, estando entre as 5 linguagens mais utilizadas no mundo segundo a pesquisa do *Stack Overflow (2018)*. Por sua facilidade, os programadores ao redor do mundo começaram a desenvolver pacotes ou bibliotecas que ampliaram bastante a capacidade da linguagem, tornando mais prático para o programador final, como por exemplo, a biblioteca *Tensorflow* que através de processamento paralelo tornou mais prático e popular a utilização do aprendizado de máquinas, com ênfase nas redes neurais profundas *(Deep Learning)*.

4.2 OpenCV

A biblioteca *OpenCV (Open Source Computer Visio Libray)* é uma biblioteca de código aberto com funções de programação e aplicações de visão computacional e PDI em tempo real, originalmente desenvolvida pela *Intel* em 2000, tem suporte para linguagens de programação como: C/C++, *Python*, *Java* e *Visual Basic* podendo ser executada nos mais diversos sistemas operacionais, como *Windows*, *Linux*, *Android* e *Mac*.

Entre as aplicações da biblioteca, estão filtros de imagem, segmentação, identificação e reconhecimento de objetos, reconhecimento de face e gestos, reconhecimento de bordas e movimento, dentre outros. A facilidade na utilização e a diversidade dos algoritmos foi o motivo pela escolha da biblioteca para a realização deste projeto.

4.3 TensorFlow

A biblioteca *TensorFlow* é uma biblioteca de código aberto com uma ampla utilização nas aplicações de aprendizado de máquinas (*Machine Learning*) e principalmente, na construção e treinamento de modelos de aprendizado profundo (*Deep Learning*). Sua operação é baseada em grafos de fluxos de dados, onde esses dados são geralmente arrays multidimensionais, chamada de tensores. Foi desenvolvida pela Google em 2015, tendo o suporte para diversas linguagens de programação, tais como: *Python*, *C++/C#*, *Java*, *Go*, dentre outras, e tem como diferencial a possibilidade de execução em múltiplas CPUs e GPUs, tornando uma boa opção para tarefas complexas, tais como o treinamento de redes neurais convolucionais e recorrentes, sendo esse o motivo pela sua utilização nesse projeto.

4.4 Base de Dados

Com o intuito do reconhecer carros, foi utilizado para o treinamento da rede neural o conjunto de dados de *Stanford* do projeto FGComp 2013, que consistia na representação e categorização de objetos 3D, com ênfase em carros. Essa base de dados, contém 16185 imagens de 196 modelos de carros diferente. Além disso, foi utilizada outro conjunto de dados para o treinamento das imagens chamadas negativas (que não contém o objeto a ser detectado), para isso foi utilizado o conjunto de dados da COCO (*Comon Objects in Context*), que detém milhares de imagens de vários tipos de objetos.

5 Implementação dos algoritmos

Nesta seção serão postas e descritas as implementações dos algoritmos utilizados baseados nas técnicas de visão computacional, como também, na aplicação da rede neural com o objetivo da detecção de veículos. Para realizar essa aplicação, foram utilizados a linguagem de programação *Python*, e as bibliotecas de visão computacinal *Opencv* para a realização das etapas de visão computacionale e a biblioteca *tensorflow* para a criação, modelamento e treinamento da rede neural. Para o projeto foram utilizados vídeos onde a gravação consiste na captura das imagens de uma estrada, ou seja, de uma via de tráfego. Para a rede neural, a metodologia consiste no treinamento e validação de uma rede neural convolucional para o reconhecimento de carros, a partir da utilização do banco de dados de *Stanford* FGComp 2013.

5.1 Desenvolvimento dos algoritmos de Visão Computacional

O objetivo dessa seção é demonstrar como foi realizado a implementação do algoritmo para a detecção de veículos em uma via de tráfego utilizando algumas das técnicas demonstradas até aqui. Como o aplicação consiste na detecção em um cenário real, pode haver várias complicações em termos do processamento da imagem, tal quais, uma grande variação da iluminação, variações climáticas, as sombras dos veículos, além de alguns tipos de ruídos, portanto, é necessário tentar tornar o algoritmo mais robusto a fim de reduzir essas influências.

Na implementação do algoritmo é inicialmente posto as bibliotecas utilizadas na aplicação, como também, a leitura e o armazenamento do vídeo. Após a realização dessa etapa, é aplicado a etapa do processamento inicial, é convertido os frames coloridos do vídeo para a escala de cinza, afim de ter melhor rendimento nas técnicas que serão utilizadas, como também reduzir o processamento do computador, além disso, foi utilizado um filtro de suavização para a remoção de ruídos, para aplicação foi implementado o filtro de suavização Gaussiana com um kernel de tamanho 7x7, representado pela função *cv2.GaussianBlur*, que teve um melhor rendimento na atênuação do ruído digital, dentre os filtros testados. A figura 20 representa a etapa de pré-processamento.

Figura 20: Código referente a etapa de pré-processamento.

```
import numpy as np  # biblioteca utililizada para manipulação de vetores e matrizes
import cv2   # importação da OPENCV - biblioteca de visão computacional
import matplotlib.pyplot as plt # biblioteca utilizada para vizualização de imagens

video = cv2.VideoCapture('CarTraffic.mp4')  #  video utilizado para o experimento
fgbg = cv2.createBackgroundSubtractorMOG2(detectShadows = False)

while True:
    ret, frame = video.read()  # Ler o video e grava os frames
    frame1 = frame.copy()

    # Etapa de processamento Inicial

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) # Converte os frames para escala de cinza
    blur = cv2.GaussianBlur(gray, (7, 7), 0)   # Aplica a Suavização para redução dos ruídos - kernel 7x7
```

A próxima etapa é a de segmentação, aqui é onde será feito a separação do objeto de interesse que no caso, são os carros (*foreground*), para as demais partes da imagem (*background*), para isso, foi utilizado uma função chamada de *cv2.createBackgroundSubtractor MOG2*,essa função tem como finalidade extrair o objeto em movimento do plano de fundo (*background*) estático, tendo como resposta uma imagem com o plano de fundo preto e os objetos com um tom de cinza alto, ou branco para uma imagem binarizada. Nessa etapa ainda é utilizado a operação de limiarização, para tentar reduzir, ao máximo, as sombras dos carros. Para isso foi utilizado a função *cv2.threshold*, essa função requer quatro parâmetros: a imagem a ser utilizada a aplicação de limiarização, o valor de limiar, o valor máximo do *pixel* e o tipo de limiarização utilizada. Para essa aplicação utilizou a operação *cv2.THRESHBINARY* que se assemelha a equação 2.9, além disso utilizou um limiar bastante alto em relação ao valor máximo, na intenção de detectar só as sombras, para assim elimina-las na imagem. A figura 21 representa o código da etapa de segmentação.

Figura 21: Código referente a etapa de segmentação.

```
# Etapa de Segmentação

mask = fgbg.apply(blur)  # aplica o background subtractor
value, threshold = cv2.threshold(mask, 240, 255, cv2.THRESH_BINARY)  # Utilização do threshold para diminuir as sombras
```

Depois de ter retirado o ruído, e diminuido a influência das sombras, é realizado operações morfológicas com a finalidade de tornar mais perceptível a detecção do objeto desejado. Foram utilizadas três operações, primeiramente foi utilizado a operação de fechamento, para a eliminação de pequenos orifícios e preenchimento dos *gaps*, compactando assim, o objeto a ser detectado. Após a operação de fechamento, foi realizado a de abertura, que consiste em eliminar as proeminências delgadas e suavizar o contorno da imagem, além da remoção de ruídos, e por fim, foi aplicado a dilatação, para preencher os buracos com pixels brancos, além de aumentar a imagem em si, facilitando ainda mais a

detecção do objeto. Para a utilização das funções morfológicas, é necessário a utilização de um *kernel*, para aplicação foi utilizado o *MORPH_CROSS* que consiste em uma matriz 5x5 em formato de cruz, comumente utilizado para esse tipo de aplicação. Esse *kernel* pode ser criado apartir de uma função proveniente da biblioteca *OpenCV* chamada de *getStructuringElement*, onde é especificado o tipo de elemento estruturante, como também o tamanho do *kernel*. Na figura 22 é mostrado o código referente as operações morfológicas.

Figura 22: Código referente a aplicação das operações morfológicas.

```
# Etapas Morfológicas

kernel = cv2.getStructuringElement(cv2.MORPH_CROSS, (5, 5))  # Kernel utilizado na aplicação das operações morfológicas
fechamento = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel, iterations=1) # Aplicação do fechamento ( 1 iteração)
abertura = cv2.morphologyEx(fechamento, cv2.MORPH_OPEN, kernel, iterations=1) # Aplicação da abertura
dilatacao = cv2.dilate(abertura, kernel)  # aplica a operação de dilatação
```

Já para a etapa de extração de características foi utilizado a função do *cv2.findContours* que consiste na procura e obtenção dos contornos externos. Essa função, exige três parâmetros: a imagem que será realizado a operação, o tipo de contorno, que para esse caso foi utilizado o contorno externo, e o terceiro parâmetro se refere a forma em que será aproximado o contorno achado, no caso, foi utilizado a opção *cv2.CHAIN_APPROX_SIMPLE*, que comprime os segmentos horizontais, verticais e diagonais deixando apenas, os pontos finais. Para finalizar é necessário selecionar esses contornos, ou seja, os objetos selecionados, então foi utilizado a função *cv2.boundingRect*, onde ela constrói uma caixa delimitadora limitando o objeto. Como, mesmo depois de todo processamento feito até aqui, a função seleciona vários contornos além dos carros propriamente dito, foi delimitado uma condição para se utilizar a caixa delimitadora, foi utilizado uma área mínima para tal aplicação, por fim, é desenhado o retângulo para poder ser vizualizado a detecção na imagem apartir da função *cv2.Rectangle*, onde nessa função, são postas as dimensões para o desenho do quadrado, como também a cor e a espessura. Essa etapa é demonstrada na figura 23.

Figura 23: Código referente a etapa de extração.

```
# Utilização da função FindContours para achar os contornos externos
contornos, hier = cv2.findContours(dilatacao.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

for contorno in contornos:
    if cv2.contourArea(contorno) < 1500:
        continue
    (x, y, w, h) = cv2.boundingRect(contorno)  # cria a caixa delimitadora
    cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 3) # desenha o retângulo no frame
```

Outra forma da obtenção da detecção de veículos, é a utilização do *HaarCascade*, essa técnica utilizada consiste no treinamento de um classificador, que nada mais é que combinação de determinadas características (características *Haar*) do objeto de interesse.

O algoritmo implementado é bastante simples, já que não foi realizado o treinamento desse classificador, sendo utilizado um classificador já pronto. Para o algoritmo é utilizado uma função do *OpenCV* que consiste em ler os dados desse treinamento, ou seja, faz a leitura do classificador. Essa função é demonstrada por *cv2.CascadeClassifier*. Outra função utilizada é a *detectMultiScale* que tem como finalidade determinar os parâmetros do classificador, que são: *scaleFactor*, que especifica quanto o tamanho da imagem é reduzido em cada escala de imagem e o *minSize* que especifica a área do menor objeto a ser reconhecido, esses parâmetros são bastante específicos, podendo variar de aplicação para aplicação, no caso desse algoritmo, foi encontrado um resultado razoável para os valores de *scaleFactor* e *minSize* iguais a 1.05 e 15 por 15, respectivamente. Vale salientar que quanto menor o valor de *minSize* maior é o processamento computacional. A figura 24 mostra essa implementação.

Figura 24: Código utilizando o *HaarCascade*.

```
# Utilização do Haar Cascade
cascade = 'cars.xml'  # algoritmo de treinamento utilizado
car_cascade = cv2.CascadeClassifier(cascade)    # utilização do classificador
image_haar = car_cascade.detectMultiScale(gray, scaleFactor = 1.05, minSize = (15,15)) # parâmetros da função
for (x, y, w, h) in image_haar:   # for utilizado para desenhar o retângulo
    cv2.rectangle(frame1, (x, y), (x+w,y+h), (0, 255, 255), 3)
```

Nas figuras 25, 26, 27, 28 são mostrado as etapas de processamento utilizada até aqui:

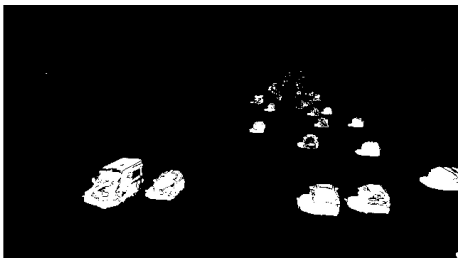
Figura 25: Etapa de processamento inicial .

(a) Imagem Original.

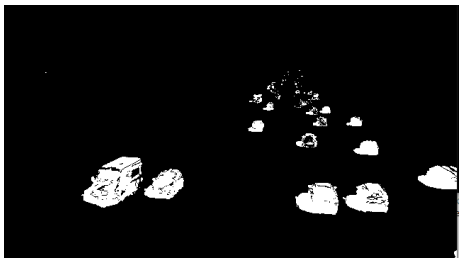
(b) Imagem em escala de cinza.

(c) Imagem com a aplicação do filtro de suavização.

Figura 26: Etapa de segmentação.

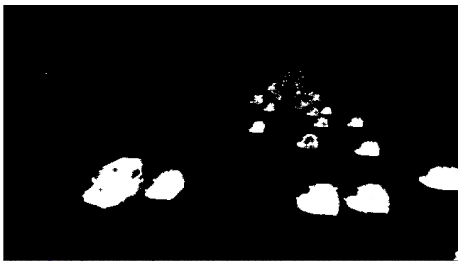


(a) Imagem com a aplicação do *createBackgroundSubtractorMOG2*



(b) Imagem com a aplicação do *threshold*.

Figura 27: Etapa de aplicação das operações morfológicas.



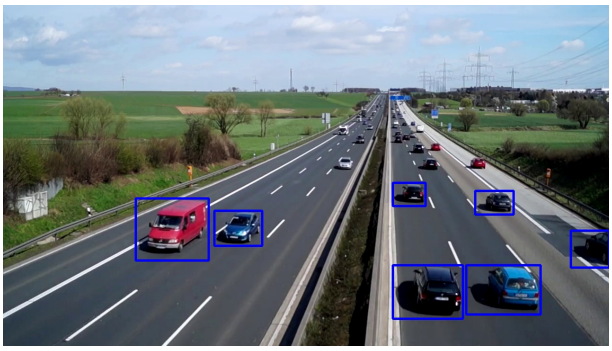
(a) Imagem com a aplicação da operação de fechamento.

(b) Imagem com a aplicação da operação de abertura.

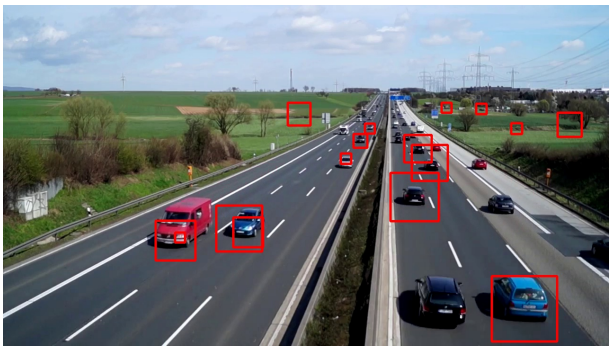
(c) Imagem com a aplicação da operação de dilatação.

Por fim é demonstrado a parte final, ou seja, a etapa de extração, do reconhecimento em si do objeto desejado utilizando as técnicas de visão computacional discutidas até aqui. Além disso é demonstrado de forma paralela, a aplicação do classificador do *HaarCascade*.

Figura 28: Etapa de extração.



(a) Imagem com a aplicação da função *findContours*



(b) Imagem com a aplicação do *HaarCascade*.

5.2 Implementação da Rede Neural Convolucional

Como dito antes, foi realizado a implementação e o treinamento de um rede neural convolucional para a detecção de carros, ou seja, será feito uma rede neural para identificar se em uma certa imagem existe ou não um carro, depois disso será adequado essa rede para a utilização dos frames . O algoritmo será dividido em algumas partes: será feito primeiramente a separação e a devida modificação nas imagens para que estejam aptas para o treinamento, será implementado todas as etapas da CNN e por fim será feito o treinamento. A figura 29 mostra a implementação das bibliotecas utilizadas no programa.

Figura 29: Implementação das bibliotecas utilizadas.

```
# Importação das bibliotecas utilizadas no projeto

import cv2   # importação da OPENCV - biblioteca de visão computacional
import numpy as np  # biblioteca utililizada para manipulação de vetores e matrizes
import matplotlib.pyplot as plt # biblioteca utilizada para vizualização de imagens
import os # biblioteca utilizada para manipulação de funções comuns do sistema operacional
import tensorflow as tf # biblioteca utilizada para o treinamento da rede neural convolucional
import random  # biblioteca utilizada para utilizar numeros aleatórios
```

Após a implementação das bibliotecas é necessário padronizar as imagens para um tamanho específico, foram utilizado 10000 imagens para o treinamento, 5000 imagens

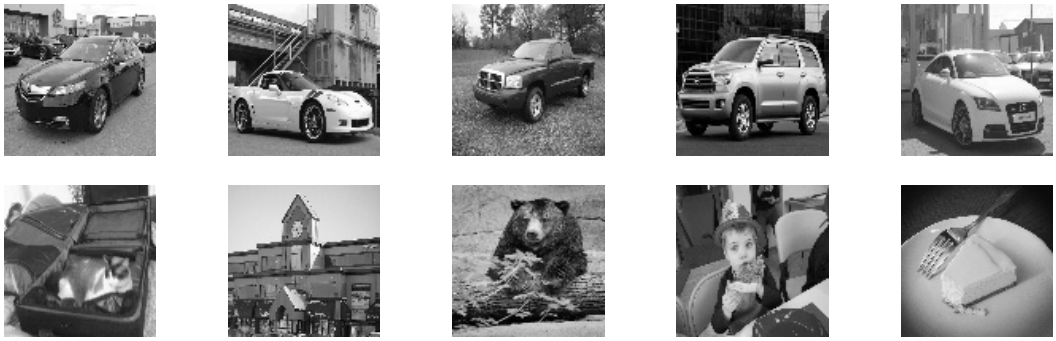
positivas (carros) e 5000 imagens negativas (que não são carros) e essas imagens variam no seu tamanho, por isso é necessário a padronização das imagens para um determinado tamanho, foi utilizado o tamanho de 52x52 pixels. Esse redimensionamento é demonstrado na figura 30.

Figura 30: Aplicações iniciais do algoritmo

```
Data_Path = 'imagens/cars_train/'  # caminho da pasta onde estão as imagens para treinamento (positivas)
Data_Path2 = 'imagens/Not_A_Car/'   # caminho da pasta onde estão as imagens para treinamento (negativas)
Tamanho_Padrao = (52, 52)      # tamanho utilizado para todas imagens

Entrada_positiva = []  # lista onde será armazenado as imagens positivas
Entrada_negativa = []  # lista onde será armazenado as imagens negativas

for i in range(0,5000):   # for utilizado para percorrer as 5000 imagens
    Entrada_positiva.append(cv2.imread(Data_Path + str(i+1) '.jpg'))   # adiciona a imagem na lista de entrada
    Entrada_positiva[i] = cv2.cvtColor(Entrada_positiva[i], cv2.COLOR_BGR2GRAY) # converte a imagem RGB para escala de cinza
    Entrada_positiva[i] = cv2.resize(Entrada_Positiva[i], Tamanho_Padrao) # redimensiona as imagens para a dimensão 52x52
    cv2.write(str(i+1) + '.jpg', Entrada_positiva[i]) # cria a nova imagem redimensionada
```

A seguir são mostrado alguns exemplos das imagens positivas e negativas utilizadas.

Figura 31: Exemplos de imagens utilizada no treinamento.

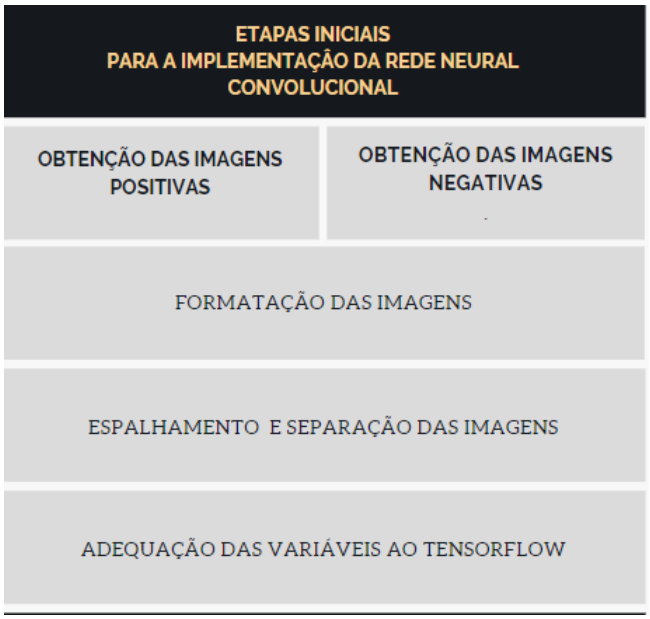


Após a padronização das imagens, foi criado uma lista em *python* composta de duas listas, onde a primeira representa a matriz de entrada e a segunda lista representa os valores de suas respectivas saídas , ou seja, se é um carro (Saída = 1) ou se não é (Saída = 0), tudo isso para efetuar o espalhamento dos dados desse vetor, sendo um passo necessário para o treinamento da rede afim de reduzir os erros, depois desse espalhamento, as duas listas são separas em uma lista de entrada e uma lista de saída. Por fim, ainda se faz necessário a realização da padronização da lista de entrada para a entrada da rede neural em termos de variáveis na forma de tensores, para isso, é utilizado a função *tf.placeholder* para criar o vetor em forma de tensor. Também utilizou-se a função *tf.reshape*, onde a mesma tem como objetivo redimensionar os vetores de entradas em forma de lista para um vetor de tensor. A figura 32 representa um pouco dessas configurações, já na figura 33.

Figura 32: Código utilizado para espalhamento dos vetores de entrada

```
Dados_Treinamento()
random.shuffle(dados_treinamento)  # espalha os dados contidos na lista

X_treino = []  # Lista para os valores de entrada espalhados
Y_treino = []  # Lista para os valores de saida espalhados
for Entrada, Saida in dados_treinamento:
    X_treino.append(Entrada) # Adiciona os valores de entrada
    Y_treino.append(Saida)   # Adiciona os valores de saida (One Hot)
```

Figura 33: Fluxograma das etapas iniciais de uma CNN

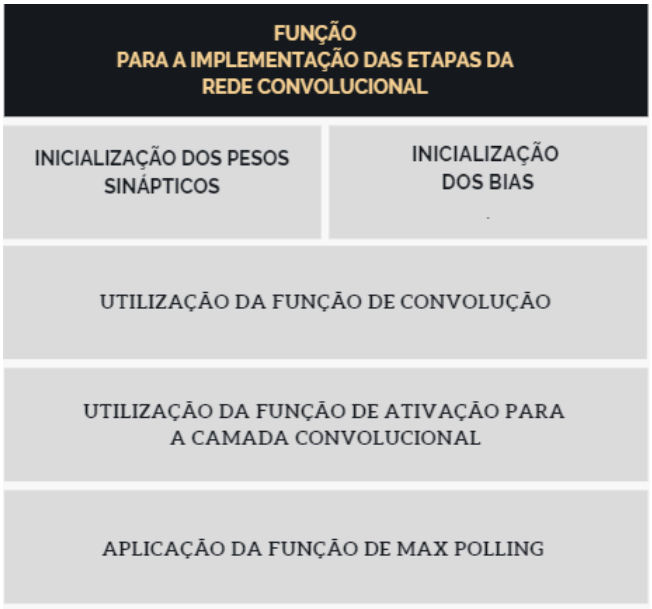


Uma rede neural convolucional é composta de três etapas: a camada de convolução, a camada de *pooling* e a camada de saída totalmente conectada. No algoritmo foi feito uma função para simular essas camadas, a função compõe a etapa dos filtros de convolução e dos filtros de *pooling*. Para a camada de convolução os filtros são ajustados automaticamente com valores dos pesos aleatórios para que consigam distinguir as características relevantes através da operação de convolução, já para a saída dessa camada é utilizado uma função de ativação retificar linear de forma a reconhecer algumas caractéristicas do vetor de entrada, essa etapa é comumente chamada de mapa de características. Na camada de pooling, foi utilizado um passo de dimensões 2x2 (*stride*) e o tipo de preenchimento (*same padding*) que preenche com zeros a imagem, a função que aplica a camada de *max polling* é *tf.nn.max_pool*, onde os parâmetros dela são o tamanho da janela, o *stride* e o *padding*. Vale salientar que as dimensões do vetor de entrada é reduzido pela metade a cada vez que passa pela função. A figura 34 representa essa função e a figura 35 representa o fluxograma da função descrita.

Figura 34: Função para a realização da CNN

```python
def Criar_Rede_Convolucional(Dado_Entrada, Num_Canais_Entrada, Num_filtros, filtro_shape, pooling_shape, nome):
    filtro_conv_shape = [filtro_shape[0], filtro_shape[1], Num_Canais_Entrada, Num_filtros]  # Dimensão do filtro de convolução

    # Inicializa os pesos sinápticos e bias para o filtro
    pesos = tf.Variable(tf.truncated_normal(filtro_conv_shape, stddev = 0.03), nome = nome +'_P')
    bias = tf.Variable(tf.truncated_normal([Num_filtros]), nome = nome + '_b')

    # Configura a operação da camada convolucional
    camada_saida = tf.nn.conv2d(Dado_Entrada, pesos, [1, 1, 1, 1], padding='SAME')

    # Adiciona o bias
    camada_saida += bias

    # Aplica a função de ativação Relu no resultado
    camada_saida = tf.nn.relu(camada_saida)

    # Aplica o max Polling
    ksize = [1, pooling_shape[0], polling_shape[1], 1]   # Dimensões da camada de Pooling
    strides = [2, 2, 2, 2]  # O passo utilizado para o max polling (2x2)
    camada_saida = tf.nn.max_pool(out_layer, ksize=ksize, strides=strides, padding='SAME') # Utilização da função Max Pooling
```

Figura 35: Fluxograma da função da realização da CNN



Foram utilizados duas camadas convolucionais e duas camadas de pooling, com a intenção de reduzir o tamanho da imagem e consequentemente, reduzir o custo computacional, na primeira camada foi utilizado 16 filtros e na segunda camada foram utilizados 32 filtros. Para a terceira etapa, a camada de saída totalmente conectada, o vetor de saída tem uma única dimensão diferentemente das camadas anteriores, ou seja, é necessário fazer uma remodelação desse vetor, onde inicialmente a matriz tem dimensões 52x52 e ao passar por essa função duas vezes, reduz seu tamanho em quatro vezes, resultando em uma matriz 13x13, além disso é necessário multiplicar pela quantidade de filtros utilizados (mapa de caracteríticas) na última camada. Após a realização dessa etapa, esse vetor de saída agora entra em uma rede neural de perceptron de multicamada como mostrado na figura 36, para o projeto foi utilizado apenas uma camada escondida com 80 neurônios, sendo possível conseguir um resultado para fins acadêmicos satisfatório.

Figura 36: Modelamento da Rede Neural

```
camada1 = Criar_Rede_Convolucional(x_shaped, 1, 16, [5, 5], [2, 2], nome='camada1')
camada2 = Criar_Rede_Convolucional(camada1, 16, 32, [5, 5], [2, 2], nome='camada2')

Camada_Tot_Conec = tf.reshape(layer2, [-1, 13 * 13 * 32])  # redimensiona a matriz para um vetor 36864 posições


# Utilização da camada profunda da Rede Neural - foi utilizado 300 Neurônios
pd1 = tf.Variable(tf.truncated_normal([13 * 13 * 32, 80], stddev=0.03), nome='Pd1') # Configura os valores dos pesos
bd1 = tf.Variable(tf.truncated_normal([80], stddev=0.01), nome='bd1') # configura os valores do bias
camada_profunda1 = tf.matmul(Camada_Tot_Conec, Pd1) + bd1  # realiza a operações da rede neural
camada_profunda1 = tf.nn.relu(camada_profunda1)  # Aplica a função de ativação Relu

# Utilização da camada de saída da Rede Neural
pd2 = tf.Variable(tf.truncated_normal([100, 2], stddev=0.03), nome='Pd2') # Configura os valores dos pesos
bd2 = tf.Variable(tf.truncated_normal([2], stddev=0.01), nome='bd2') # configura os valores dos bias
camada_profunda2 = tf.matmul(camada_profunda2, Pd2) + bd2 # realiza as operações da rede neural
y_ = tf.nn.softmax(camada_profunda2) # utiliza a função de ativação softmax
```

Nessa rede neural são efetuados as operações básicas como: multiplicação dos pesos sinápticos pelos dados de entrada, adição do valor do *bias* e o cálculo do nível de ativação dos neurônios pela função de ativação que são dados pelas equações (3.1), (3.2) e (3.3). Essa etapa se repete duas vezes, uma entre a camada de entrada e a camada oculta, e uma segunda vez entre a camada oculta e a camada de saída. Na camada de saída é utilizado a função de ativação *softmax* que é bastante utilizada como classificador nessa camada, tendo como objetivo representar a probabilidade de cada classe para cada valor de entrada.

Para o realização do treinamento da rede, foi utilizado o algoritmo de retropropagação, que é baseado na aprendizagem por correção de erro, sendo necessário a modelação de uma função de custo. Para isso foi utilizado a função *cross entropy* que é disponível na biblioteca do *tensorflow* sendo uma das mais utilizadas, essa função mede a distância entre os valores de codificação da saída desejada *one hot* (Vetor de saida tem apenas "um valor quente", exemplo: [0 1 0]) com a saída da rede. A figura 37 representa essas configurações.

Figura 37: Algoritmo referente a função de custo e ao otimizador

```
# Variaveis de otimização
taxa_aprendizado = 0.0001
epocas = 3
tamanho_lote = 100

# Adiciona a função de custo
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=camada_profunda2, labels=y)

# Adiciona um Otimizador (Adam Optimizer)
otimizador = tf.train.AdamOptimizer(learning_rate=taxa_aprendizado).minimize(cross_entropy)

# Cria operações corretas de previsão e avaliação de precisão
predicao_correta = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
precisao = tf.reduce_mean(tf.cast(predicao_correta, tf.float32))
```

A função *softmax_cross_entropy_with_logits* recebe dois argumentos, o primeiro é o vetor da camada de saída (valor real) e o segundo é o vetor alvo de treinamento (valor

desejado). A função compara os dois resultados utilizando a função *cross entropy* e o resultado é o cálculo dessa entropia por amostra de treinamento, sendo necessário reduzir esse tensor a um valor único (escalar) para ser utilizado posteriormente, a função que realiza essa tarefa é *tf.reduce_mean*. Além disso, é utilizado um otimizador *AdamOptmizer*, que é um algoritmo de otimização baseado no gradiende estocástico bastante utilizado em aprendizado profundo sendo responsável por atualizar os pesos sinápticos da rede de forma iterativa com base nos dados de treinamento e na taxa de aprendizado.

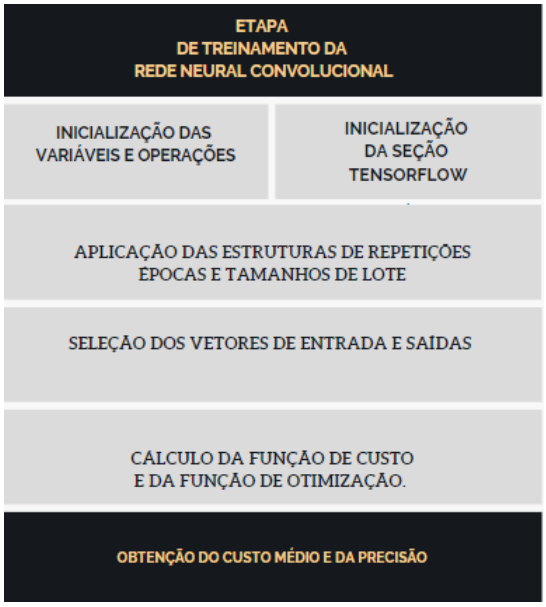
A variável predição correta utiliza a função *tf.equal* que retorna verdadeiro ou falso dependendo se os argumentos fornecidos forem iguais ou não. A função *tf.argmax* retorna o índice do valor maximo em um vetor ou *tensor*. Portanto, essa variável retorna um tensor do tamanho *[m x 1]*, onde m é o quantidade de entradas, de valores verdadeiros ou falsos, indicando se a rede neural previu de forma correta, ou não. Para se calcular a exatidão média desse vetor, é necessário transformar o vetor de valores *booleanos* para *float* para então, executar a função *tf.reduce_mean*, como foi feito anteriormente.

Para reduzir o processamento computacional, o treinamento não será realizado de forma sequencial , será feito atráves de lotes, nessa forma de aprendizagem o ajuste dos pesos são realizados após a apresentação de todos os exemplos de treinamento que constituem uma época. Foi estipulado 100 lotes para o treinamento, como também, a utilização de 3 épocas. O treinamento completo é demonstrado na figura 38, bem como o fluxograma dessa etapa representado pela figura 39.

Figura 38: Código referente ao treinamento da rede neural

```
# configura o operador de inicialização do tensor flow
init_op = tf.global_variables_initializer()



with tf.Session() as sess: # Abre a seção do tensor flow

    sess.run(init_op) # inicializa as variáveis
    tamanho_total_lote = 10000 / tamanho_lote  # quantidade de lotes
    for epoca in range(epocas):
        custo_medio = 0
        j = 0
        for i in range(tamanho_total_lote):
            x, y = (X_treino[j:(j+100)], Y_treino[j:(j+100)])  # seleciona 100 entradas e saídas
            _, c = sess.run([otimizador, cross_entropy],   # realiza a operação de otimização e cross entropy
                            feed_dict={x: X_treino, y: Y_treino})
            j += 100
            custo_medio += c / tamanho_total_lote  # calcula o custo medio por epoca
        teste_precisao = sess.run(precisao,   # realiza a operacao de precisão
                        feed_dict={x: X_teste, y: Y_teste})

        # printa o custo medio e precisão por epoca
        print(f'Epoca: {epoca + 1}, custo =  {custo_medio}, teste de precisão = {teste_precisao}')
```

Figura 39: Fluxograma referente ao treinamento da rede neural



As operações do *tensorflow* são realizados dentro de uma seção, por isso, é necessário inicializar suas operações, como também suas variáveis. Depois disso é utilizado um laço de repetição *for* para percorrer cada época de treinamento e é inicializado uma variável de custo médio para acompanhar o custo (*cross entropy*) em cada época.

Como a forma de treinamento está dividida entre 100 lotes, serão utilizados para cada lote 100 vetores de entradas e saídas de treino, para efetuar os cálculos da função de custo e a otimização dos pesos sinápticos.

A função *sess.run* executa as funções de otimização e *cross entropy*, são atribuidos duas variáveis para essas funções, quantativamente, não importa o valor da otimização, o que importa é o valor da operação de *cross entropy* que é atribuido à variável c, onde a mesma é utilizada para calcular o custo médio por época.

Por fim é calculado a precisão tanto por época como a total. O resultado pode ser visto na figura 40.

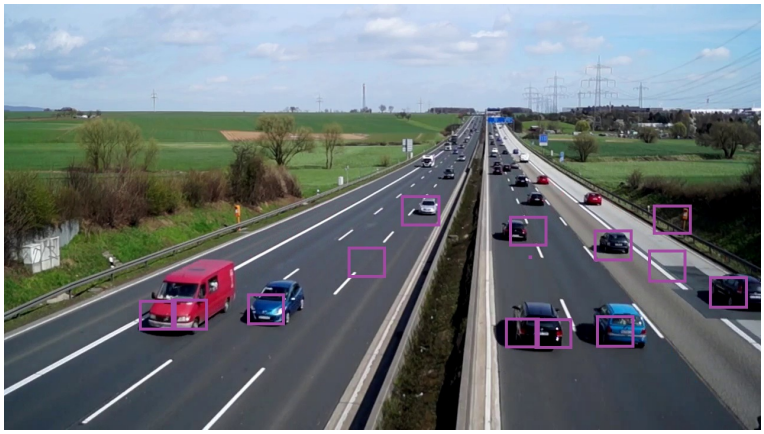
Figura 40: Resultado do treinamento da rede neural convolucional

```
Epoca: 1, custo = 0.683,  teste de precisão = 0.712
Epoca: 2, custo = 0.229,  teste de precisão = 0.783
Epoca: 3, custo = 0.137,  teste de precisão = 0.791
Treinamento completo
0.7899
```

Portanto, a partir da imagem 40 pode se verificar que a cada época é diminuido o custo, e consequentemente, aumentado a taxa de certeza da rede. Vale salientar que essa precisão é realizada utilizando imagens de validação contendo apenas um veículo por imagem, portanto, para essa aplicação, onde é necessário identificar atráves de uma caixa

delimitadora vários carros em um mesmo frame, foi realizado uma divisão da imagem original em várias imagens menores, utilizando janelas de 52x52 pixels, e fornecendo essas novas entradas para a rede neural. Com isso, foi possível detectar os veículos na imagem, já que para cada janela que passa pela rede neural, ela consta se existe ou não a existência do objeto referido. Para a aplicação da caixa delimitadora, foi utilizado as posições desses pixels que tiveram como resposta da rede neural o valor positivo. A figura 41 relata essa aplicação.

Figura 41: Imagem com a aplicação da CNN.



6 Resultados

Neste capítulo, serão apresentados os resultados obtidos pelas técnicas utilizadas para a detecção de veículos, vale salientar, que o objetivo desse trabalho não é encontrar a melhor opção, mas sim, estudar algumas das técnicas existentes de forma a ter uma ampliação do conhecimento desses assuntos a fim da utilização dos mesmo em qualquer outra aplicação.

Na figura 28, foi utilizado, a fim de demonstração, apenas um frame de um determinado vídeo. Pode se observar que na figura 28.a teve um melhor desempenho em comparação a figura 28.b. Pode-se verificar que na imagem contendo o algoritimo de *Haarcascade* (28.b), tem uma maior seleção de caixas delimitadoras, porém, com uma maior taxa de erro, ou seja, com várias marcações "falsas"(que não contém o objeto requerido). Vale salientar, que o uso do classificador *Haarcascade* foi utilizado sem nenhuma forma de pré processamento da imagem, apenas a utilização da escala de cinza, a fim de reduzir o trabalho computacional. Verificando a figura 28.a, nota-se que ela selecionou de forma correta todos os carros em que foi posto a caixa delimitadora, porém com o advento da distância, ou seja, da diminuição do objeto, não foi feito nenhuma marcação nos demais carros, justamente pelo fato que foi delimitado uma área mínima afim de reduzir os erros, como pode ser encontrado na figura 28.b.

Portanto, sempre é recomendável o tratamento na imagem, independente da aplicação, e independente da utilização de algoritmos já treinados, a fim de reduzir as possibilidades de erro.

Para o treinamento da rede neural convolucional foram utilizados 10000 imagens para treinamento e 10000 imagens para a validação(teste), onde a metade das imagens são imagens ditas positivas (que contém o objeto carro) e a outra metade são imagens ditas negativas, que não tem o objeto desejado. Após o treinamento, essa rede teve a seguinte taxa de resultado utilizando 100 neurônios na camada oculta, 2 camadas de convolução, com 16 e 32 filtros de características, e 2 camadas de *polling*.

Tabela 1: Tabela referente ao treinamento da CNN

Epoca	Custo	Precisão
1	0.683	0.712
2	0.229	0.783
3	0.137	0.791

Onde a precisão, determina a taxa de acerto da rede em relação as imagens de validação e época representa a iteração no treinamento da rede neural afim de reduzir o erro através da mudança dos pesos sinápticos. O resultado final é representado para a terceira época, onde resulta em uma taxa de acerto de aproximadamente 80% para detecção

se existe ou não o objeto de interesse na imagem. De fato, em termos de aplicações em redes neurais, essa porcentagem não é ótima, mas devido as limitações de processamento computacional e o intuito desse trabalho, foi considerado satisfatório. Para a aplicação se fez necessário uma forma alternativa para identificar os veículos, da mesma forma utilizadas pelas técnicas anteriores. Então foi realizado uma divisão dos frames utilizados em imagens menores, ou seja, o frame original foi recortado em várias imagens de dimensão 52x52 com o intuito da identificação separada dos veículos como a rede normalmente faria, e através do conhecimento dos pixels dessas regiões e do resultado da rede neural, foi possível, utilizando a imagem total e a utilização das caixas delimitadoras fazer uma predição dos veículos na imagem. Essa forma de verificação, demanda um processamento computacional bastante elevado, já que seria necessário passar pela verificação imagens menores para compor a imagem total, considerando apenas um frame, já que um vídeo em codificação padrão contém 30 frames por segundo, se torna bastante complexo para o computador fazer essa operação em tempo real, em termos de processamento. Mas, como o intuito desse trabalho é o estudo das técnicas, essa complicação é deixado de lado.

Em geral, uma rede neural para ter uma melhor taxa de acerto, uma das opções mais comuns utilizadas é a utilização de uma maior quantidade de dados de entrada para efetuar o treinamento. Além disso, a utilização de uma maior quantidade de neurônios na camada oculta, ou até adicionar mais camadas tornam a rede com melhor rendimento, porém o custo computacional para realizar essas etapas adicionais se tornam bastante altas.

Outra forma de caracterizar uma rede neural, é fazendo a comparação dos resultados em termos de resultados falsos ou positivos. Existem 4 classes de resutaldos, o verdadeiro positivo, que representa o valor que a rede neural caracterizou como verdadeiro, e de fato é verdadeiro. O verdadeiro negativo, que a rede neural caracterizou como falso, e de fato é falso. O falso-positivo, que a rede neural caracterizou como verdadeiro, porém a resposta correta é que o valor é falso. E por último o falso-negativo, que a rede neural caracteriza como negativo, porém é positivo. A tabela abaixo demonsta a matriz de confusão, que representa essa comparação entre as 4 categorias.

Tabela 2: Comparação da matriz de confusão

Tipo de classificação	Percentagem
Verdadeiro Positivo	76%
Verdadeiro Negativo	69%
Falso Positivo	24%
Falso Negativo	31%

Para fazer a comparação das técnicas utilizadas, foram utilizados apenas 180 frames, totalizando em 6 segundos de vídeo, de forma a capturar a quantidade de caixas delimitadoras que estão corretas e a quantidade que estavam erradas para formar uma

comparação entre as três técnicas.

Tabela 3: Comparação das técnicas

Técnica	Posivitas	Negativas	Total
Técnicas de PDI	21	0	21
HaarCascade	27	13	40
CNN	23	10	33

Foram utilizados tanto para o algoritmo de *Haarcascade* como para a utilização das técnicas de PDI e visão computacional, o *minSize*, ou seja, o tamanho mínimo para o preenchimento da caixa delimitadora com uma área de 1500, aproximadamente. O resultado é mostrado na tabela 2, pode ser verificado que a primeira técnica teve um desempenho bem melhor, de 100%, porém alguns objetos não são reconhecidos, como é o caso da segunda opção, que tem um maior número de veículos reconhecidos, contudo, com uma maior taxa de erro, já que o aproveitamento foi de 67%. Já para a CNN, foi verificado um valor intermedário entre os 3 (71%), contendo um bom resultado, comparado a taxa de certeza da rede, vale salientar, que a taxa da rede não se tornou maior ou melhor que as demais técnicas, devido a limitação computacional para a realização do treinamento da mesma.

7 Conclusão

Nos dias de hoje, aplicações de visão computacional estão nas mais diversas áreas de conhecimento, resolvendo problemas através do uso de câmeras que antes eram necessária a intervenção humana. Por isso, se torna necessário ter o conhecimento dessas técnicas, como também, as etapas de pre-processamento da imagem. De fato, o PDI é um requisito necessário nas aplicações de visão computacional já que é trabalhado a imagem em um mais baixo nível.

A popularização das redes neurais tornou mais prático a resolução de muitos problemas complexos, e com a popularização, vem o constante estudo, trazendo, ultimamente, várias arquiteturas inovadoras de redes neurais, como é o caso da rede neural convolucional, que se tornou bastante utilizada em problemas utilizando imagens, já que tem a característica da invariância a rotações e distorções locais.

Esse trabalho consiste no estudo de diferentes técnicas para chegar ao mesmo resultado, a utilização de técnicas de PDI e visão computacional tornou possível a implementação de forma mais simples, porém com necessidade de vários conhecimentos relativos a área. Já para as redes neurais, foram introduzidos o conhecimento básico das RNA, bem como sua estruturas, para melhor entendimento no momento da criação e treinamento da mesma. Além disso foi utilizado ainda classificadores *HaarCascade* no intuito de tornar a comparação entre as técnicas mais apurada.

O trabalho em si, não necessariamente, tem como intuito procurar a melhor opção para as detecções de objetos, mas sim, ter um amplo conhecimento das possíveis técnicas utilizadas. Foram comparadas apenas para um fim de caracterização. Além de que essas técnicas tem como serem melhoradas, já que para o resultado final, foram testadas outras técnicas que não se encaixaram bem ao propósito, mas ao chegar no aprofundamento desse assunto, é possível chegar numa solução ótima desta aplicação.

REFERÊNCIAS

[1] (BALLARD, BROWN, 1982) Ballard, Dana H., Brown, Christopher M, 1982. *Computer Vision.* Prentice Hall, Nova York , Estados Unidos.

[2] (Bogdanchikov, Zhaparov, Suliyev, 2013) Bogdanchikov, A. and Zhaparov, M. and Suliyev, R, 2013. *Python to learn programming.* IOPScience SciTech 2013.

[3] (BORGES, 2014) Borges, L. E, 2014. *Python para desenvolvedores.* Editora Novatec, Rio de Janeiro.

[4] (GLENDA, 2007) Glenda, Michele (2007). *Processamento de Imagens no Auxílio à Detecção de Câncer de Pele utilizando Redes Neurais Artificiais.* Dissertação de graduação, Ciência da Computação, Universidade Federal de Goiás, Catalão.

[5] (GONZALES and WOODS, 2000) Gonzalez, R. and Woods, R. (2000). *Processamento de Imagens Digitais.* Editora Edgard Blucher, São Paulo.

[6] (HAYKIN, 2001) Haykin, S. (2001). *Redes Neurais: Príncipios e Práticas*. Editora Bookman, Porto Alegre.

[7] (LUGER, 2004) Luger, G. (2004) *Inteligência Artificial: Estruturas e Estratégias para a Solução de Problemas Complexos*. Editora Bookman, Porto Alegre,.

[8] (MARTINELLI, 1999) Martinelli, E. (1999) *Extração de Conhecimento de Redes Neurais Artificiais.* Dissertação de mestrado, Instituto de Ciências Matemáticas e de Computação, Universidade Federal de São Paulo, São Carlos.

[9] (MARQUES FILHO e VIEIRA NETO, 1999) Marques Filho, O. and Viera Neto, Hugo. *Processamento Digital de Imagens.* Editora Brasporte, Rio de Janeiro.

[10] (McCULLOCH and PITTS, 1943) McCulloch, W. S. and Pitts, W. (1943). *A Logical Calculus of the Ideas Imminent in Nervous Activity.*

[11] (MIYAZAKI, 2017) Miyazaki, Caio, 2017. *Redes neurais convolucionais para aprendizagem e reconhecimento de objetos 3D.* Dissertação de graduação, Escola de engenharia de São Carlos, Universidade de São Paulo, São Carlos.

[12] (ÓSORIO and VIEIRA, 1999) Osorio, F. and Vieira, R. (1999). *Sistemas Híbridos Inteligentes*. XIX Congresso da Sociedade Brasileira de Computação, Rio de Janeiro. Encontro Nacional de Inteligência Artificial.

[13] (PHIL, 2018) Phil, Simom, 2018. *Too big to ignore: the business case for big Data* Editora John Wiley and Sons, Nova York, Estados Unidos.

[14] (PRATT, 2001) Pratt, W. (2001). *Digital Image Processing.* Editora John Wiley and Sons, Nova York, Estados Unidos.

[15] (RAVI and KHAN, 2013) Ravi, S and Am Khan, 2013. *Morphological Operations for Image Processing : Understanding and its Applications*. Conference Proceedings, NCVSComs 2013.

[16] (SINGH, 2009) Singh, K. P, 2009. *Artificial neural network modeling of the river water quality: A case study.* Ecological Modelling, 2009.

[17] *Car Dataset - FGCOMP 2013 (Stanford).* Disponível em:<https://ai.stanford.edu/ jkrause/cars/car_dataset.html>.

[18] Krause,J., Stark, M., Deng, J., Fei-Fei, Li, 2013. (*3D Object Representations for Fine-Grained Categorization*). Sydney, Australia, 2013.

[19] *COCO - Common Objects in Context Dataset.* Disponível em:<http://cocodataset.org/home> Acesso em: 04 mar. 2019.

[20] PAVLOVSKY, VOJTECH. *Introduction To Convolutional Neural Networks.* Disponível em: <http://www.vaetas.cz/posts/intro-convolutional-neural-networks/> Acesso em: 04 mar. 2019.

[21] *Convolutional Neural Networks Tutorial in TensorFlow.* Disponível em: <http://adventuresinmachinelearning.com/convolutional-neural-networks-tutorialtensorflow/> Acesso em: 04 mar. 2019.