INSTITUTO FEDERAL
Paraíba
Campus João Pessoa

CURSO SUPERIOR DE BACHARELADO EM ENGENHARIA ELÉTRICA

CARLOS VICTOR DOS SANTOS FARIAS

DESENVOLVIMENTO DE SISTEMA PARA DETECÇÃO DE DEFORMAÇÕES EM ESTRADAS UTILIZANDO REDES NEURAIS ARTIFICIAIS

João Pessoa
2023

CARLOS VICTOR DOS SANTOS FARIAS

DESENVOLVIMENTO DE SISTEMA PARA DETECÇÃO DE DEFORMAÇÕES EM ESTRADAS UTILIZANDO REDES NEURAIS ARTIFICIAIS

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Superior de Bacharelado em Engenharia Elétrica do Instituto Federal da Paraíba como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Elétrica.

Orientador:
Patric Lacouth da Silva, Doutor

João Pessoa
2023

Dados Internacionais de Catalogação na Publicação (CIP)
Biblioteca Nilo Peçanha do IFPB, *campus* João Pessoa

F224d Farias, Carlos Victor dos Santos.

Desenvolvimento de sistema para detecção de deformações em estradas utilizando redes neurais artificiais / Carlos Victor dos Santos Farias. - 2023.

55 f. : il.

TCC (Graduação - Curso Superior de Bacharelado em Engenharia Elétrica) - Instituto Federal de Educação da Paraíba / Unidade Acadêmica de Processos Industriais, 2023.

Orientação : Profº D.r Patric Lacouth da Silva.

1. Redes neurais artificiais 2. Perceptron multicamada. 3. Visão computacional. 4. Sistemas embarcados. 5. Tráfego de automóveis. I. Título.

CDU 004.032.26(043)

Lucrecia Camilo de Lima
Bibliotecária - CRB 15/132

CARLOS VICTOR DOS SANTOS FARIAS

DESENVOLVIMENTO DE SISTEMA PARA DETECÇÃO DE DEFORMAÇÕES EM ESTRADAS UTILIZANDO REDES NEURAIS ARTIFICIAIS

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Superior de Bacharelado em Engenharia Elétrica do Instituto Federal da Paraíba como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Elétrica.

Trabalho aprovado em 06 de fevereiro de 2023 pela banca examinadora:

Documento assinado digitalmente
gov.br FRANKLIN MARTINS PEREIRA PAMPLONA
Data: 16/03/2023 16:53:36-0300
Verifique em https://validar.iti.gov.br

_____

Franklin Martins Pereira Pamplona, Doutor
Examinador, IFPB

Documento assinado digitalmente
gov.br LINCOLN MACHADO DE ARAUJO
Data: 15/03/2023 20:14:14-0300
Verifique em https://validar.iti.gov.br

_____

Lincoln Machado de Araújo, Doutor
Examinador(a), IFPB

Documento assinado digitalmente
gov.br PATRIC LACOUTH DA SILVA
Data: 14/03/2023 14:27:15-0300
Verifique em https://validar.iti.gov.br

_____

Patric Lacouth da Silva, Doutor
Orientador, IFPB

João Pessoa
2023

AGRADECIMENTOS

Não teria como agradecer primeiramente a outra pessoa senão minha mãe, Maria Rosineide dos Santos, a Rosa. Sou muito grato a todo o amor, todo cuidado e toda a dedicação que esta mulher me deu e é graças a ela que tenho a capacidade de desenvolver este trabalho.

Ao meu orientador, Prof. Patric Lacouth, que aceitou o desafio de trabalhar com um aluno com pouquíssimo tempo para desenvolver um projeto de pesquisa. Fico muito feliz por ter tido a oportunidade de desenvolver um trabalho junto de um professor que eu admiro tanto quanto o senhor.

A todos os meus professores e amigos que tive o prazer de conhecer durante minha jornada acadêmica e que me ajudaram durante essa caminhada, em especial Ana Karolina, Anna Paula, Daniel Henrique, Pâmella Araújo e Richard Sidney. Deixo uma menção honrosa aos meus outros amigos de vida, que me deram suporte emocional sempre que precisei, Karina Lelis e Rafael Vitor.

RESUMO

O tráfego de automóveis, sejam eles urbanos ou de grande porte, em estradas ou rodovias que possuem trincas, buracos ou qualquer outro tipo de deformação na sua estrutura, além de poder causar danos às peças desses automóveis, fazem com que motoristas e pedestres corram o risco de sofrer acidentes graves. É pensando nesta problemática que este trabalho sugere a criação de um sistema capaz de detectar deformações em vias de tráfego de automóveis. Para isto, foram utilizadas as redes neurais artificiais, desenvolvidas na linguagem Python, para o treinamento de um modelo capaz de identificar tais deformações, dado um banco de dados que contém imagens de estradas encontradas na África do Sul. Após uma série de otimizações e testes, descritos neste trabalho, utilizados para melhorar e comprovar a assertividade da rede desenvolvida, um modelo final foi definido e a sua precisão, para o banco de dados utilizado neste projeto, foi de 93,30%. Ao final do trabalho, algumas melhorias futuras são propostas para o aprimoramento das funcionalidades e ferramentas criadas neste projeto.

Palavras-chave: *Redes Neurais Artificiais, Perceptron Multicamada, Visão Computacional, Python, Sistemas Embarcados.*

ABSTRACT

The car traffic, whether urban or large, on roads or highways that have cracks, holes or any other type of deformation in their structure, in addition to being able to damage the parts of these vehicles, make drivers and pedestrians take the risk of serious car accidents. It is with this problem in mind that this work suggests the creation of a system capable of detecting deformations in car traffic lanes. For this, artificial neural networks, developed in the Python language, were used to train a model capable of identifying such deformations, given a database containing images of roads found in South Africa. After a series of optimizations and tests, described in this work, used to improve and prove the assertiveness of the developed network, a final model was defined and its accuracy, for the database used in this project, was 93.30%. At the end of the work, some future improvements are proposed to upgrade the functionalities and tools created in this project.

Keywords: *Artificial Neural Networks, Multilayer Perceptron, Computer Vision, Python, Embedded Systems.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CC	Corrente Contínua
CPU	*Central Processing Unit*
GPU	Graphics *Processing Unit*
HDMI	*High-Definition Multimedia Interface*
LED	*Light-Emitting Diode*
RGB	*Red, Green and Blue*
USB	*Universal Serial Bus*

SUMÁRIO

1 INTRODUÇÃO

Quando estamos falando de deformações em pistas que são utilizadas para o tráfego de automóveis, podemos facilmente englobar desde as vias das pequenas ruas pavimentadas do bairro simples onde moramos, até as grandes estradas responsáveis pela movimentação de milhares de caminhões por dia. Essas deformações podem ser identificadas como simples trincas, onde a pista apresenta rachaduras em sua extensão, ou mesmo buracos e depressões de variados tamanhos e formatos.

O desconforto estético não é o único problema nessas situações, visto que buracos em pistas e estradas podem causar danos financeiros aos donos de automóveis e veículos de grande porte. Carros, motos ou caminhões que passam por cima de grandes buracos ou deformações severas em uma pista, geralmente com velocidades moderadas ou elevadas, podem danificar seriamente seus veículos, sendo algumas vezes necessária a troca de peças ou de partes do automóvel. Um exemplo deste fato está na BR 365, mais especificamente no trecho que liga Patos de Minas e Uberlândia, onde nos últimos cinco anos foram registrados incontáveis casos de caminhões e carros que tiveram seus pneus estourados devido às deformações da pista (Patos Hoje, 2023).

Ainda mais sério do que o prejuízo financeiro está o risco de integridade física de motoristas e pedestres, visto que qualquer tipo de deformação em uma pista pode causar acidentes. Para se ter uma ideia da dimensão desse cenário nas estradas brasileiras, entre janeiro e setembro de 2021, 47,7 mil acidentes foram registrados nas rodovias federais, tendo a má infraestrutura das estradas apontada como um dos principais causadores (Folha de São Paulo, 2021). Esses acidentes levaram o governo federal a ter um gasto de aproximadamente R$ 8,9 bilhões de reais durante o mesmo período de tempo, gasto esse que ultrapassa o dobro do valor investido em infraestrutura das rodovias federais, R$ 4,16 bilhões.

Foi pensando nesta problemática que este trabalho busca desenvolver métodos de identificação em tempo real de deformidades em estradas. O objetivo principal do projeto não é resolver de vez a problemática do tipo de acidente descrito acima, mas contribuir para os estudos e o desenvolvimento de ferramentas capazes de sinalizar e alertar motoristas para o risco de eventuais perigos nas estradas. Para isso, este trabalho se utiliza de ferramentas como a rede neural artificial, utilizada para a criação do modelo de um sistema capaz de avaliar fotografias digitais e dizer, com certo grau de

precisão, se existe ou não algum tipo de deformidade em nas estradas mostradas nestas imagens. Além disso, princípios de visão computacional e sistemas embarcados somam o escopo do trabalho com a finalidade de dar mobilidade e aplicação real ao sistema desenvolvido a partir da rede neural artificial.

Este trabalho é dividido principalmente por Capítulos, que também são divididos em Seções, cada uma abordando e especificando algum tema ou aspecto individual dos mesmos. O atual Capítulo é representado pelo número 1 e é dedicado à Introdução do trabalho. O Capítulo 2 apresenta toda a Fundamentação Teórica utilizada como embasamento para o trabalho. O Capítulo 3 é reservado para a exposição da Metodologia. O Capítulo 4 apresenta os Resultados obtidos ao fim da pesquisa deste projeto. Por fim, o Capítulo 5 explana as Considerações Finais.

2 FUNDAMENTAÇÃO TEÓRICA

Para a construção de um sistema de aprendizagem de máquina a partir de redes neurais artificiais muitos elementos, que podem já ter sido desenvolvidos por terceiros ou que devem ser criados do zero, devem ser integrados entre si. A utilização de linguagens de programação apropriadas, bem como de bibliotecas e bancos de dados são uma parte essencial do trabalho e devem ser cuidadosamente pensadas.

Este Capítulo trata de explicar a fundamentação teórica, os processos, a linguagem de programação e as ferramentas utilizadas para o desenvolvimento do projeto.

2.1 PROCESSAMENTO DIGITAL DE IMAGENS

Devido às inúmeras aplicações nas mais diversas áreas do conhecimento, os estudos e os desenvolvimentos em Processamento Digital de Imagens (PDI) vem sofrendo uma grande expansão desde o início do Século XX. De forma geral, o uso do PDI pode ser dividido em dois grandes grupos: a captação, o tratamento e a exibição de informações que serão analisadas por humanos ou pelos próprios computadores (Filho & Neto, 1999).

Documentos históricos datam as primeiras aplicações da área durante os anos 20, quando cabos submarinos eram utilizados para transportar imagens digitalizadas de jornais por equipamentos conhecidos como impressoras telegráficas entre Londres e Nova Iorque. Mas foi só com o surgimento e o aprimoramento dos computadores que o PDI passou a ser mais explorado e utilizado. Uma das primeiras e mais notórias aplicações do PDI ocorreu no ano de 1964 no *Jet Propulsion Laboratory*, em Pasadena na Califórnia, onde imagens da Lua transmitidas pelo Ranger 7 foram tratadas para que ruídos fossem retirados (Gonzalez & Woods, 2000).

Desde então inúmeras outras aplicações foram desenvolvidas e registradas no PDI. A aplicação de reconhecimento de elementos é amplamente utilizada por profissionais da área da Biologia para que contagens automáticas sejam realizadas. Na medicina o PDI auxilia no reconhecimento de patologias ou mesmo na exibição de partes internas do corpo humano para o auxílio de diagnósticos. Áreas como Geografia e Engenharia Civil se utilizam do PDI para obter informações como nivelamento,

tamanho e composição de terrenos. As Engenharias Elétrica, Mecatrônica e de Automação Industrial fazem uso de visão computacional para aprimorar robôs ou máquinas autônomas que serão utilizadas em indústrias, subestações e afins. Estes são apenas alguns dos exemplos das mais diversas aplicações que mostram a multidisciplinaridade do PDI e o quão essa ferramenta pode ser aproveitada na execução de tarefas rotineiras (Filho & Neto, 1999).

2.1.1 IMAGEM

De acordo com o dicionário *Oxford Languages* a palavra "Imagem" tem sua origem no latim, *imāgo*, que faz referência a "semelhança", "representação" ou ainda "retrato", além de pode ter alguns dos seguintes significados:

"1. Representação, reprodução ou imitação da forma de uma pessoa ou de um objeto. [...]. 2. Aspecto particular pelo qual um ser ou um objeto é percebido; cena, quadro. 3. Reprodução invertida de um ser ou de um objeto, transmitida por uma superfície refletora. 4. Reprodução estática ou dinâmica de seres, objetos, cenas etc. obtida por meios técnicos." (Google, 2022).

Se na antiguidade a única maneira de se capturar uma imagem em forma de representação física se resumia a pinturas ou esculturas, a partir do século XIX esse cenário começa a ser mudado com a criação das primeiras câmeras fotográficas. A primeira delas que se tem registro remete ao ano de 1839 com o francês Louis Jacques Mandé Daguerre que, a partir da Academia Francesa de Ciência, anuncia ao mundo sua mais nova criação (Channel, 2022). As técnicas de aquisição das imagens com certeza são consideradas ultrapassadas se comparadas às do século XXI, mas entre esses dois séculos uma evolução enorme tomou conta dessa área.

Apesar de aquilo que é considerado a primeira câmera digital criada ter sido registrada no ano de 1975 por Steve Sasson, no laboratório de *Eastman Kodak Company*, a verdade é que até o início dos anos 2000 as câmeras analógicas dominavam o mercado e eram as mais consumidas pela população geral (MeioBit, 2008). Foi só com o barateamento das câmeras digitais que as imagens digitais começaram a se popularizar, sendo elas um dos elementos principais deste projeto e o foco de discussão da próxima Seção.

2.1.2 IMAGENS DIGITAIS

Uma imagem em escala de cinza, ou monocromática, pode ser definida como uma função *f(x, y)*, onde *x* e *y* representam as coordenadas da imagem, e o resultado da função representa a intensidade luminosa naquela coordenada específica (Gonzalez & Woods, 2000). Além disso, *f(x, y)* pode ser definida como uma função resultante do produto entre *r(x, y)*, função representante da refletância ou transmitância próprias do objeto, com *i(x, y)*, função representante da iluminância incidente sobre o objeto. Ou seja, a função *f(x, y)* pode ser representada de acordo com a Equação 1 mostrada abaixo.

$$f(x, y) = i(x, y) \cdot r(x, y) \quad (1)$$

Nas imagens monocromáticas cada uma das coordenadas (*x, y)* é chamada de *pixel* e, como seu tamanho, geralmente, é de um *byte*, este pode armazenar valores que variam de 0, cor preta, a 255, cor branca.

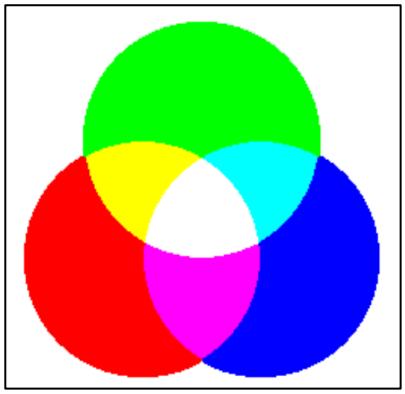
Já as imagens multicanais, ou seja, para aquelas que contém mais de um canal de cor, a função *f(x, y)* é aplicada para cada uma dessas cores. Os valores dos *pixels* também variam entre 0 e 255 para cada um dos canais, porém não representam mais as escalas de cinza, e sim as escalas de cores ou configurações definidas por cada sistema.

2.1.3 SISTEMA RGB

Um dos sistemas multicanal de imagens é o RGB, do inglês *Red, Green and Blue*, que tem esse nome por construir imagens a partir de três canais: um vermelho, um verde e um azul. No caso do sistema RGB, cada um dos três canais varia os valores de seus *pixels* entre 0 e 255, sendo o primeiro a representação da cor branca, o último a representação da cor preta, e os valores entre ambos representam a escala de cor de cada um dos três tons que dão nome ao sistema (Filho & Neto, 1999).

A adição dos valores *pixel* a *pixel* desses três canais de cores formam as várias outras cores existentes em uma imagem. Um exemplo da adição de cores primárias em imagens digitais para a criação de cores secundárias e a cor branca pode ser vista na Figura 1.

Figura 1 – Adição de Cores Primárias de Acordo com o Modelo RGB.



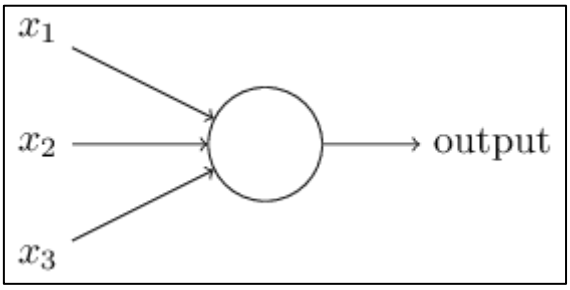
FONTE: (Filho & Neto, 1999).

2.2 REDES NEURAIS ARTIFICIAIS

O campo de estudos a respeito das redes neurais artificiais se baseia nos próprios neurônios biológicos para desenvolver sistemas capazes de tomar decisões a partir de uma árvore de escolhas e um treinamento de resultados esperados. Sendo assim, a partir de um sistema bem definido e estruturado de neurônios artificiais é possível implementar uma ferramenta de aprendizagem de máquina, bem como de reconhecimento de padrões.

Os primeiros registros a respeito das redes neurais artificiais foram os de Warren McCulloch e Walter Pitts que, inspirados pelos recentes avanços no campo da computação e da lógica booleana propostos por Alan Turing e John Von Neumann, se uniram e publicaram um artigo chamado *A Logical Calculus of the Ideas Immanent in Nervous Activity* (Kovács, 2006). Apesar de ser extremamente simples, o modelo de neurônio artificial proposto por Warren McCulloch, visto na Figura 2, bem como o artigo publicado por ele e seu companheiro, é até hoje referência para trabalhos na área ou mesmo para a teoria de redes neurais artificiais no geral.

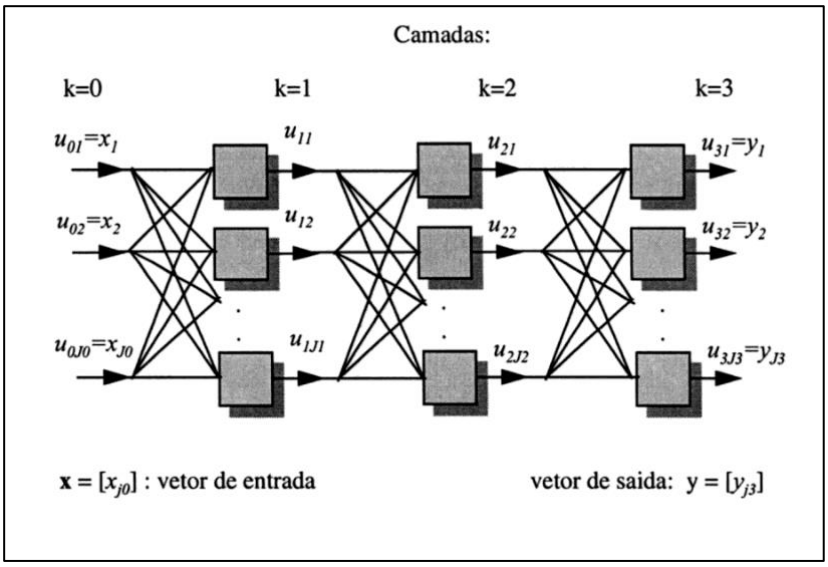
Figura 2 – Modelo de Neurônio Artificial com Três Entradas e Uma Saída.



FONTE: (DeepLearningBook, 2022).

Em um curto espaço de tempo, na década de 1950, Rosenblatt continuou desenvolvendo as ideias de Warren McCulloch na Universidade de Cornell, criando uma rede neural artificial mais robusta e volumosa do que a originalmente proposta por seu antecessor. Em sua proposta havia múltiplos neurônios do tipo discriminadores lineares que ele chamou de *Perceptrons* dispostos em várias camadas, sendo a primeira chamada de "Camada de Entrada", a última chamada de "Camada de Saída" e as intermediárias chamadas de "Camadas Ocultas" (Kovács, 2006). A Figura 3 mostra uma rede neural artificial multicamada de K camadas.

Figura 3 – Modelo de Rede Neural Proposto por Rosenblatt.



FONTE: (Kovács, 2006).

2.3 FERRAMENTAS COMPUTACIONAIS

Esta Seção trata de especificar todos os *softwares* e plataformas utilizados durante a execução do projeto.

2.3.1 PYTHON

O Python é uma linguagem de programação de *software* livre, o que significa dizer que, apesar das especificações da linguagem estarem em posse da *Python Software Foundation*, a mesma pode ser utilizada por qualquer pessoa, inclusive em projetos e produtos corporativos. A linguagem de alto nível é conhecida por possuir uma sintaxe simples e de fácil compreensão humana, o que facilita no momento da aprendizagem, principalmente de pessoas que nunca tiveram contato com programação (Borges, 2014).

A criação do Python se deu no ano de 1990, por Guido Van Rossum no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação da Holanda, e seu principal objetivo na época era auxiliar físicos e engenheiros com as demandas de cálculo e processamento de dados em suas pesquisas. Para conceber o Python, Guido se baseou em outra linguagem de programação que já existia na época chamada ABC (Borges, 2014).

Hoje, no ano de 2022, o Python já conta com três versões lançadas, inúmeros módulos criados pela comunidade de usuários para as mais diversas aplicações e uma alta aceitabilidade na indústria da tecnologia, sendo utilizado por empresas como o *Google* e a *Microsoft* (Borges, 2014). Com tanto uso e relevância no cenário atual da Tecnologia da Informação, e sendo o Python uma plataforma *open source*, a linguagem já possui mais de 198 mil pacotes e mais de 137 mil bibliotecas criadas e disponíveis para *download* (Terminal Root, 2019).

Uma dessas bibliotecas é o *keras*, uma Interface de Programação de Aplicação, ou API, que do inglês significa *Application Programming Interface*, utilizada na aprendizagem de máquina e no reconhecimento de padrões para projetos que utilizam esta linguagem. Em sua documentação oficial a API assegura seguir as melhores práticas para redução de carga cognitiva: "*[...] oferece APIs consistentes e simples, minimiza o número de ações do usuário necessárias para casos de uso comuns e fornece mensagens de erro claras e acionáveis. Ele também possui extensa documentação e guias do desenvolvedor."* (Keras, 2022).

Utilizando como base os padrões do *keras*, outra importante biblioteca utilizada na aprendizagem de máquina e no reconhecimento de padrões com Python é o *tensorflow*. O *tensorflow* é uma plataforma de código aberto especializada na criação de redes neurais artificiais treinadas para detectar padrões e correlações em uma determinada base de dados. De acordo com os próprios mantenedores da plataforma, o *tensorflow* pode ser utilizado por desenvolvedores que variam de iniciantes na área, que podem se utilizar de APIs de alto nível como a *Sequential*, até especialista, que podem fazer uso de APIs mais complexas e com níveis de abstração mais baixos (TensorFlow, 2022).

A linguagem Python, assim como as duas bibliotecas apresentadas acima, foram utilizadas integralmente neste projeto, vista a maior facilidade de criação e reutilização de elementos pré criados nas áreas de aprendizagem de máquina e redes neurais.

2.3.2 GOOGLE COLAB

A plataforma *Colaboratory*, ou simplesmente *Colab*, é um serviço de armazenamento em nuvem e de execução de códigos escritos na linguagem Python do *Google*. O *Colab* oferece muitas vantagens na sua utilização, como a falta da necessidade de configurações locais da máquina, acesso à Unidades de Processamento Gráfico ou GPUs, do inglês *Graphics Processing Units*, do *Google* com ou sem custo financeiro adicional e a facilidade do compartilhamento dos projetos entre seus usuários. Além disso, a plataforma não é voltada apenas para os profissionais da área de tecnologia, como também para estudantes, cientistas e pesquisadores que podem acessar e utilizar todas as funções básicas gratuitamente (Google, 2022).

Para este projeto todos os códigos-fonte escritos foram armazenados e executados na plataforma *Colab*, sendo utilizados os recursos básicos e gratuitos oferecidos pelo *Google*.

2.3.3 GOOGLE DRIVE

O *Drive* nada mais é do que um serviço de armazenamento, sincronização e compartilhamento de arquivos na nuvem criado e mantido pelo *Google*. A plataforma possui um amplo leque de possibilidades que vão desde o simples armazenamento de arquivos pessoais, a sincronização de pastas e arquivos em máquinas locais com a nuvem, ou até mesmo o compartilhamento e a colaboração em arquivos para equipes acadêmicas ou comerciais. Outra vantagem é que o *Drive* pode ser acessado pelos dispositivos móveis, por *tablets*, *notebooks* ou *desktops* (Drive, 2022).

Outra grande vantagem da plataforma, e também o motivo pelo qual a mesma foi escolhida para fazer parte deste trabalho, é a capacidade de integração com o *Google Colab*. A partir de uma API criada pela própria empresa *Google*, os arquivos e pastas armazenadas em uma conta do *Drive* podem ser acessados e processados via código. A Seção Código Base mostra como foi feita a integração entre os dois serviços.

2.4 ROBÔS AUTÔNOMOS

A definição acerca do que são robôs têm se mostrado algo totalmente dinâmico, ou seja, vem sendo alterada e evoluída na medida em que a evolução tecnológica permite novos avanços na automatização e automação de máquinas. O da palavra

"robô" foi popularizada por Karel Capek, um dramaturgo de origem tcheca que escreveu e exibiu uma peça chamada "Robôs Universais de Rossum" no ano de 1921. A palavra utilizada por Karel seria o resultado da junção de outras duas palavras tchecas, sendo elas *rabota*, que é uma palavra tcheca que pode ser traduzida literalmente para o português como "trabalho obrigatório", e *robotnik* que significa "servo" (Mataric, 2007).

Alguns anos depois, em 1950, quando os transistores tinham acabado de ser inventados e a noção de computação e das máquinas computadores estavam passando por um processo de evolução e modernização (Gadelha, 2022), a ideia acerca dos robôs estava bem mais evoluída e os humanos começavam a pensar nas infinitas possibilidades de utilização dos mesmos. Um exemplo disso é que neste mesmo ano o escritor estadunidense Isaac Asimov publicou o livro *I, Robot*, do inglês "Eu, Robô", que contava com uma compilação de nove contos individuais, cada um deles cogitando como seria a relação entre humanos e robôs num futuro próximo.

Asimov ainda criou o que ele mesmo chamou como "As três leis da robótica", que são resumidas em: um robô não pode permitir que um humano sofra qualquer tipo de dano; um robô deve obedecer todas as ordens dos humanos, exceto nos casos em que a lei anterior seja quebrada; e um robô não pode causar dano a si mesmo, exceto para cumprir as duas leis anteriores (Frazão, 2022). Essas três leis são lembradas até hoje e foram incorporadas no estudo e no desenvolvimento acadêmico e industrial da área da robótica.

Avançando para as décadas de 60, 70 e 80, começaram a ser desenvolvidos os primeiros robôs como conhecemos hoje em dia. Estes possuíam um equipamento de *hardware* muito maior do que os encontrados hoje em dia no mercado, além de processadores e slots de memória tão simples que limitavam e muito o processamento e a movimentação destes. Alguns desses robôs são: *Shakey*, construído no final da década de 60 pela *Stanford Research Institute*, que já inovou em sua época por possuir sensores de contato e uma câmera; *Cart*, desenvolvido na Universidade de Stanford em 1977 por Hans Moravec, que é considerado hoje em dia como um dos criadores da robótica moderna; e o *Rover*, desenvolvido em 1983 na *Carnegie Mellon University*, que se utilizada de uma câmera e sensores de ultrassom para se movimentar (Mataric, 2007).

Já na década de 1990 os robôs permitiam estudos mais aprofundados e aplicações diversas, como é o caso do futebol com robôs autônomos. Este cenário exige a criação e o correto funcionamento de um sistema de reconhecimento de espaço e

aprendizagem de máquina robustos, dado que uma partida de futebol, mesmo que de robôs, seja totalmente imprevisível. Costa e Pegoraro (2000, p.141) descrevem alguns dos passos necessários para o funcionamento de um conjunto desses robôs no final dos anos 90. Em seu trabalho, foi utilizada a visão computacional em tempo real, um sistema de aprendizagem de máquina previamente treinado com imagens de teste, além do *hardware* específico para que os robôs consigam segurar a bola e se movimentar corretamente (Costa & Pegoraro, 2000).

O estudo e o desenvolvimento da robótica moderna vem sendo atualizada e evoluída desde então, permitindo cada vez mais a construção de robores menores e com um maior poder de processamento, capazes de realizar cada vez mais tarefas diferentes. Um exemplo disso são os robores móveis e autônomos, que nada mais são do que máquinas que possuem a capacidade de se movimentar em um determinado espaço, além de realizar determinadas tarefas repetitivas sem a intervenção humana durante esse processo (Aguiar & Silva, 2021).

2.5 MICROPROCESSADORES

As placas microcontroladoras, ou simplesmente microcontroladores, são circuitos de placa única e integrada, que possuem memórias voláteis e/ou não voláteis, além de serem capazes de se conectar com diversos periféricos de entrada e saída através de portas e pinos lógicos (UFCG, 2020). Em resumo, os microprocessadores são pequenos computadores de baixo custo e complexidade de manipulação que são largamente utilizados para a modelagem e prototipagem de projetos de automação e robótica, inclusive a educacional. Esta Seção fala sobre dois tipos distintos de placas microcontroladoras: o *Raspberry Pi* e o *Arduino*.

2.5.1 RASPBERRY PI

O *Raspberry Pi* foi desenvolvido pela *Raspberry Pi Foundation*, uma instituição de caridade com sede no Reino Unido e que, segundo os mesmos, tem como principal missão o fomento da utilização de ferramentas computacionais para jovens e adultos, com o objetivo de que os mesmos alcancem todo o seu potencial (Raspberry Pi, 2022). A fundação citada ainda investe seus recursos em três principais pilares do conhecimento: a educação formal, trabalhada nas escolas e faculdades pelo mundo; a educação informal, ou seja, aquela em que depende unicamente do indivíduo e dos

meios que o mesmo possui para aprender sozinho; e a pesquisa, onde a própria fundação pode entender como melhorar o desempenho de tudo que é desenvolvido pelos mesmos.

Seguindo essa visão, a Raspberry Foundation criou no ano de 2012 o *Raspberry Pi*, que nada mais é do que um computador de placa única. A primeira versão do *Raspberry Pi* contou com uma Unidade de Processamento Central ou CPU, do inglês *Central Processing Units*, *Single Core* de 700MHz e singelos 256MB de memória RAM (Open Source, 2022). Para se ter uma ideia, a versão mais atual do *Raspberry Pi*, até o momento da escrita deste trabalho, é o *Raspberry Pi 4 Model B*, que possui um processador *Quad-Core* de 1,5GHz e até 8GB de memória RAM (Raspberry Pi 4 Tech Specs, 2022). Este último modelo pode ser adquirido por um valor médio de U$ 35,00, o que é um valor muito baixo comparado a computadores pessoais de pequeno porte vendidos no mercado.

O sistema operacional do *Raspberry* é o *Linux*, ou variações dele, e o pequeno computador opera no ecossistema de código aberto, o que facilita muito que seus usuários aprendam e evoluam códigos previamente escritos. A placa ainda conta com pinos que podem ser utilizados para entrada e saída de dados, bem como entradas de Barramento Serial Universal ou USB, do inglês *Universal Serial Bus*, entrada de Interface Multimídia de Alta Definição ou HDMI, do inglês *High-Definition Multimedia Interface*, e suporte à cartões de memória (Open Source, 2022). Tudo isso faz com que esta pequena placa microcontroladora seja ideal para projetos de pequeno porte e protótipos de novas soluções.

No ramo da robótica, o *Raspberry Pi* pode ser empregado em diversas aplicações, como, por exemplo, robôs que se utilizam de sensores ultrassônicos para se mover em determinados ambientes, pequenos robôs móveis controlados por aplicativos móveis, robôs de *Lego* com as mais variadas funcionalidades ou até mesmo réplicas de robôs fictícios com capacidade de reconhecimento de voz e imagem (TechTudo, 2016).

3 METODOLOGIA

Tendo como base todos conceitos e ferramentas demonstrados no Capítulo anterior, este descreve todas as etapas realizadas durante a execução do projeto para que os resultados, mostrados no próximo Capítulo, fossem alcançados.

3.1 CAPTAÇÃO DE IMAGENS PARA O BANCO DE DADOS

Na intenção de se utilizar um banco de dados limpo e com o menor índice de ruídos ou vieses possível, as imagens utilizadas para o treinamento das redes neurais artificiais abordadas abaixo foram retiradas do site *Kaggle*. O *Kaggle* se define como uma comunidade aberta aos pesquisadores, praticantes da aprendizagem de máquina e cientistas de dados, sendo esta uma subsidiária do *Google*.

A seleção de imagens utilizada especificamente neste projeto foi retirada de um desafio do site chamado "*MIIA Pothole Image Classification Challenge*", que foi construído a partir de imagens de estradas localizadas na África do Sul (Kaggle, 2023). Algumas das imagens em questão se tratavam de imagens de estradas convencionais e uniformes, enquanto outra parte das imagens retratava as mesmas estradas, porém com buracos em sua extensão.

O banco de imagens do desafio continha os seguintes arquivos:

- "trainidslabels.csv": Esse arquivo do tipo de valores separados por vírgula continha os nomes das imagens que deveriam ser utilizadas durante o treinamento do modelo em sua primeira coluna e o classificador da existência de um buraco na referida imagem ou não em sua segunda coluna, sendo este último um valor binário com "1" para a existência de buraco ou "0" caso contrário;
- "testidsonly.csv": Esse arquivo, também do tipo de valores separados por vírgula, continha as imagens que deveriam ser utilizadas em testes externos aos realizados durante o treinamento do modelo e continha apenas os nomes das imagens selecionadas;
- "all_data": Por fim, o último arquivo se tratava de uma pasta que continha todas as imagens, tanto de treinamento como de teste, ou seja, todas as imagens contidas nesta pasta tinham seu nome referenciado em um dos

arquivos descritos anteriormente. No total, foram utilizadas 5676 imagens, sendo 4026 destinadas ao treinamento e teste dos modelos e 1650 destinadas para validação após os mesmos estarem prontos. As Figuras 4 e 5 mostram exemplos de imagens encontradas no banco de dados para estradas com e sem deformações na pista.

Figura 4 – Exemplo de Imagem Encontrada no Banco de Dados de Estrada sem Deformações na Pista.

FONTE: (Kaggle, 2023).

Figura 5 – Exemplo de Imagem Encontrada no Banco de Dados de Estrada com Deformações na Pista.

FONTE: (Kaggle, 2023).

O link para o desafio descrito acima está referenciado no Apêndice I.

3.2 ARMAZENAMENTO DE ARQUIVOS NO GOOGLE DRIVE

Tendo baixado todos os arquivos descritos na Seção anterior, o próximo passo se deu no armazenamento destes em um repositório do *Google Drive*. Para o caso deste trabalho, o repositório escolhido foi o do próprio aluno, sendo este criado a partir do e-mail acadêmico do mesmo e possuindo apenas as configurações padrão da plataforma, ou seja, nenhum *slot* de processamento ou armazenamento adicional foi adquirido.

Após os arquivos estarem carregados na nuvem, o código descrito na Figura 6 foi utilizado para que o repositório do *Google Drive* fosse integrado à plataforma *Google Colab* utilizada pelo aluno.

Figura 6 – Inicialização de Repositório do Google Drive no Colab.

```
from google.colab import drive;

drive.mount('/content/drive');
```

FONTE: Autoria Própria.

Vale destacar que o repositório contou com a mesma disposição encontrada na pasta original adquirida no desafio do *Kaggle*, ou seja, uma pasta de arquivos que continha a pasta de imagens utilizadas para o banco de dados, bem como os dois arquivos com valores separados por vírgula utilizados na separação das imagens de treinamento e de teste.

3.3 CÓDIGO BASE

Com o intuito de se obter uma maior organização durante a execução deste trabalho, bem como levando em conta o fato de que mais de uma rede neural artificial foi construída no processo, cinco arquivos de código-fonte escritos em Python foram criados. Os arquivos foram criados de maneira que um código base, comum em todos os códigos-fonte, fosse escrito em cada um deles, mas que a partir disso cada um seguiria com diferentes códigos-fonte escritos de acordo com as necessidades de cada um. Todos os arquivos criados foram salvos em um repositório do *GitHub* e podem ser encontrados no Apêndice II.

É importante destacar que a escolha do modelo utilizado no código base foi pensado para ser possível de ser embarcado. Por esse motivo descartou-se a possibilidade da utilização de modelos como o VGG16 ou o VGG19, que se utilizam de, respectivamente, 16 e 19 camadas de neurônios (Simonyan & Zisserman, 2014), visto que os arquivos gerados a partir destas abordagens seriam relativamente grandes para o contexto de sistemas embarcados que, geralmente, possuem até 2GB de memória RAM. Já o método utilizado neste trabalho foi o *MultiLayer Perceptron*, que contou com uma camada de entrada, uma camada oculta e uma camada de saída.

Para o código base, a primeira parte do mesmo pode ser vista na Figura 7. Com o repositório das imagens da base de dados alocado no *Google Drive* já inicializado na plataforma *Google Colab*, o caminho até os arquivos de imagens, bem como o caminho onde o arquivo em Python deveria ser salvo foram declarados em forma de variáveis globais.

Figura 7 – Importação de imagens para base de dados.

```
import pandas as pd;
import numpy as np;
import cv2 as cv;

imagesPath = '/content/drive/MyDrive/ColabNotebooks/Images/all_data/';
docPath = '/content/drive/MyDrive/ColabNotebooks/Documents/';

trainIds = pd.read_csv((docPath + 'train_ids_labels.csv'));
trainImages = [];

for index in range(0, len(trainIds)):
    label = trainIds['Image_ID'][index] + '.JPG';
    image = cv.imread((imagesPath + label), 1);
    image = cv.cvtColor(image, cv.COLOR_BGR2GRAY);
    image = cv.resize(image, (160, 120));
    trainImages.append({'Image': image, 'Pothole': trainIds['Label'][index]});

testIds = pd.read_csv((docPath + 'test_ids_only.csv'));
testImages = [];

for id in testIds['Image_ID']:
    label = id + '.JPG';
    image = cv.imread((imagesPath + label), 1);
    image = cv.cvtColor(image, cv.COLOR_BGR2GRAY);
    image = cv.resize(image, (160, 120));
    testImages.append((image, label));

trainImagesDF = pd.DataFrame(trainImages, columns=['Image', 'Pothole']);
testImagesDF = pd.DataFrame(testImages, columns=['Image', 'Label']);
```

FONTE: Autoria Própria.

Para o tratamento inicial das imagens foram utilizados dois laços de repetição *for*, um responsável por criar uma variável do tipo *DataFrame* para as imagens de treinamento e outro responsável por criar uma variável do mesmo tipo para as imagens de teste. Durante a execução do laço de repetição as imagens foram convertidas de RGB para tons de cinza, bem como redimensionadas para dimensões menores que as originais, isso se deu para que os dados de entrada estejam de acordo com os dados esperados nas funções responsáveis pelo treinamento da rede. Cada uma das imagens foi adicionada a uma linha dos respectivos *DataFrames* na forma de um *array* convencional, juntamente de um segundo campo informando a existência ou não de um buraco na imagem, para o caso do banco de treinamento, e com outro campo informando o rótulo da imagem, para o caso do banco de teste.

A partir disso o pré-processamento das imagens foi iniciado. A Figura 8 mostra a conversão dos *DataFrames* inicializados a partir da etapa anterior em listas, uma para treinamento e validação e outra para testes. Isso foi feito pois o método de treinamento do modelo, mostrado adiante, deve receber como parâmetro imagens agrupadas em uma estrutura de dados do tipo lista.

Figura 8 – Conversão de DataFrames para Listas.

```
trainImagesList = trainImagesDF.values.tolist();
testImagesList = testImagesDF.values.tolist();
```

FONTE: Autoria Própria.

A Figura 9 mostra então a normalização das imagens de treinamento e validação, onde cada elemento de cada *array*, que representa os *pixels* de cada imagem da base de dados, foi convertido do intervalo de valores entre 0 e 255 para o intervalo de valores entre 0.0 e 1.0 afim de se obter uma normalização desses elementos. Além disso, uma nova lista contendo apenas os rótulos das imagens de treinamento da base de dados foi criada.

Figura 9 – Normalização e Separação de Rótulos para Treinamento.

```
trainInputImages = [(image[0]/255.0) for image in trainImagesList];
trainLabels = np.array([label[1] for label in trainImagesList]);
```

FONTE: Autoria Própria.

A partir das listas criadas na etapa anterior, foi realizado um *split*, ou seja, uma separação na base de imagens de treinamento entre quatro listas que são mostradas na Figura 10. Estas listas serão utilizadas posteriormente para o treinamento e a validação do modelo da rede neural.

Figura 10 – Separação da Base de Imagens de Treinamento.

```
from sklearn.model_selection import train_test_split;

xTrain, xTest, yTrain, yTest = train_test_split(trainInputImages, trainLabels,
                                                test_size=0.2, random_state=42)
```

FONTE: Autoria Própria.

As duas listas que continham de fato os *arrays* das imagens de treinamento e validação foram então convertidas para o tipo de *array numpy*, além de terem uma dimensão adicionada ao seu final, como mostrado na Figura 11. Foi preciso fazer isso para que o método de treinamento do modelo, visto adiante, aceitasse essas listas como parâmetros.

Figura 11 – Conversão das Listas de Treinamento para Numpy Array e Adição de uma Dimensão.

```
xTrain = np.array(xTrain);
xTest = np.array(xTest);

xTrain = xTrain[..., tf.newaxis]
xTest = xTest[..., tf.newaxis]
```

FONTE: Autoria Própria.

Já as duas listas que continham os rótulos das imagens foram transformadas a partir de um *One Hot Encoding,* para que a coluna com as opções 1, para “há buraco na imagem”, e 0, para “não há buraco na imagem”, fosse substituída por duas colunas de valores binários. Nessas novas colunas, uma delas armazenou um valor 1 para o caso de a imagem relativa àquela linha mostrar um buraco e 0 caso contrário, já a segunda implementou a mesma lógica, porém com valores 1 para o caso de a imagem não conter um buraco. Essa transformação pode ser vista na Figura 12.

Figura 12 – Aplicação de One Hot Encoding.

```
yTestHot = tf.one_hot(yTest,2);
yTrainHot = tf.one_hot(yTrain,2);
```

FONTE: Autoria Própria.

A partir desse código base, quatro tipos diferentes de abordagens foram adotadas para comparação posterior. As próximas Seções descrevem os processos de cada uma das abordagens, além dos códigos-fonte escritos.

3.4 DEFINIÇÃO DA ARQUITETURA BÁSICA

A primeira abordagem utilizada para o treinamento de uma das redes neurais artificiais desenvolvidas neste trabalho foi a de apenas uma rede com parâmetros fixos e base de dados estática. Segue abaixo os trechos de códigos escritos, bem como suas explicações. A Figura 13 mostra o processo de criação do modelo utilizado. Aqui vale destacar que a camada oculta da rede recebeu um parâmetro *units* no valor 768 e um parâmetro *activation* do tipo *sigmoid*. Já a camada de saída recebeu um units no valor 2 e um *activation* do tipo *softmax*. Por fim, a rede foi definida com um learning rate de $5e^{-5}$. Todos esses valores foram definidos com base no estudo de trabalhos similares, como o de Júnior (2022) e o de Correia (2022), e os parâmetros *units* e *learning rate* serão testados nas Seções posteriores deste trabalho.

Figura 13 – Criação do Modelo de Rede Neural Artificial para Arquitetura Básica.

```
import tensorflow as tf;
from tensorflow.keras import regularizers;
from tensorflow.keras.optimizers import Adam

REG = 0.001;
DROP = 0.1;

model = tf.keras.Sequential();
model.add(tf.keras.layers.Flatten());

model.add(tf.keras.layers.Dense(768, activation='sigmoid', kernel_regularizer=regularizers.l2(REG)));
model.add(tf.keras.layers.Dropout(DROP));

model.add(tf.keras.layers.Dense(2, activation='softmax'));

model.compile(optimizer=Adam(learning_rate=5e-5), loss='categorical_crossentropy', metrics=['accuracy']);
```

FONTE: Autoria Própria.

Com o modelo da rede neural artificial pronto, chegou a vez do último passo para esta arquitetura: a extração de resultados, dada a base de imagens definida anteriormente. A Figura 14 mostra essa extração sendo atribuída à variável *history*. Para a extração de dados, utilizou-se um valor de 1000 épocas, além de um valor de *validation_split* de 20% das imagens. Além disso, um *EarlyStopping* foi definido com o parâmetro *patience* que recebeu o valor de 100 e o "*min_delta*" que recebeu 0.

Figura 14 – Criação do Arquivo History.

```
history = model.fit(xTrain,yTrainHot,epochs=1000,validation_split=0.2,
                    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', min_delta=0, patience=100)]);
```

FONTE: Autoria Própria.

Os resultados obtidos na última etapa e que foram atribuídos à variável *history* podem ser vistos na Seção Resultados para Arquitetura Básica.

3.5 OTIMIZAÇÃO DE ARQUITETURA ATRAVÉS DO LEARNING RATE

A segunda abordagem utilizada se deu com a otimização da arquitetura desenvolvida na Seção anterior, onde mais de uma rede neural artificial foi treinada e testada para diferentes valores de *Learning Rate*, dada a mesma base de dados e as mesmas configurações do modelo. A Figura 15 mostra a criação deste modelo da rede neural artificial onde é possível observar que sua definição é muito parecida com a rede neural artificial de arquitetura básica, com a diferença de que esta é encapsulada em uma função chamada *build_model*. É possível ver também que quatro valores de *Learning Rate* foram utilizados para a construção dos modelos, sendo esses valores: $5e^{-5}$, $5e^{-6}$, e^{-5} e e^{-6}.

A Figura 16 mostra a criação da variável *tuner* que será utilizada na criação do arquivo *history* e, consequentemente, no treinamento das redes. É nesta variável, através do parâmetro *max_trials*, que é declarada a quantidade máxima de tentativas que serão feitas, ou seja, o número máximo de redes que serão treinadas e testadas. Para o caso deste trabalho o número máximo de três tentativas foi definido.

Figura 15 - Criação do Modelo de Rede Neural para Otimização de Arquitetura por Learning Rate.

```
import tensorflow as tf;
from tensorflow.keras import regularizers;
from tensorflow.keras.optimizers import Adam

def build_model(hp):
  REG = 0.001;
  DROP = 0.1;

  model = tf.keras.Sequential();
  model.add(tf.keras.layers.Flatten());

  model.add(tf.keras.layers.Dense(768, activation='sigmoid', kernel_regularizer=regularizers.l2(REG)));
  model.add(tf.keras.layers.Dropout(DROP));

  model.add(tf.keras.layers.Dense(2, activation='softmax'));

  model.compile(optimizer=Adam(hp.Choice('learning_rate',values=[5e-5,5e-6,1e-5,1e-6])),
                loss='categorical_crossentropy', metrics=['accuracy']);

  return model
```

FONTE: Autoria Própria.

Figura 16 – Definição da Variável Tuner para Otimização de Arquitetura por Learning Rate.

```
import keras_tuner as kt;
from kerastuner.tuners import RandomSearch
from tensorflow import keras;

tuner = kt.RandomSearch(
    build_model,
    objective="val_accuracy",
    max_trials=3,
    overwrite=True,
    directory="/content/drive/MyDrive/ColabNotebooks/Documents2",
    project_name="saved_model",
)
```

FONTE: Autoria Própria.

Com as configurações já preparadas pelos trechos de código anteriores, a Figura 17 mostra como se deu a criação do arquivo *history* para esta abordagem. É possível observar que o número de épocas utilizadas para cada rede foi de 500 épocas, já que um número maior dificultaria o processamento de cada um dos modelos e tornaria inviável o treinamento dos mesmos. Ao final da execução, aquele modelo que obteve um maior valor de precisão teve seu arquivo *history* salvo no repositório do *Google Drive*.

Figura 17 – Criação do Arquivo History para Otimização de Arquitetura por Learning Rate.

```
tuner.search(xTrain, yTrainHot, epochs=500, validation_split=0.2,
             callbacks = [tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',
                                                           min_delta=0, patience=100)]);

bestModels = tuner.get_best_models(num_models=1)
highestScoreModel= bestModels[0]

highestScoreModel.fit(x=xTrain, y=yTrainHot, batch_size=64, epochs=500,
                      verbose=1, validation_split=0.2)

highestScoreModel.save("/content/drive/MyDrive/ColabNotebooks/Documents2")
```

FONTE: Autoria Própria.

Os resultados obtidos para esta abordagem e que foram atribuídos à variável *history* podem ser vistos na Seção Resultados para Otimização de Arquitetura Através do Parâmetro Learning Rate.

3.6 OTIMIZAÇÃO DE ARQUITETURA ATRAVÉS DOS PERCEPTRONS DA CAMADA OCULTA

A segunda estratégia de melhoria realizada se deu com a otimização da arquitetura desenvolvida na Seção Código Base, onde mais de uma rede neural artificial foi treinada e testada, assim como na Seção anterior, mas dessa vez para diferentes número de *Perceptrons* utilizados na camada oculta dos modelos, dada a mesma base de dados e as mesmas configurações do modelo base. A Figura 18 mostra a criação deste modelo da rede neural artificial onde é possível observar que sua definição é muito parecida com a rede neural artificial de arquitetura básica, com a diferença de que esta é encapsulada em uma função chamada *build_model*. É possível ver também que para o parâmetro *units*, que diz respeito à quantidade de *Perceptrons* da camada oculta, os valores testados estiveram entre o intervalo de 128 até 1024 unidades, com intervalos de 128 unidades entre cada valor utilizado.

Figura 18 – Criação do Modelo de Rede Neural para Otimização de Arquitetura por Número de Perceptrons da Camada Oculta.

```
[ ]  import tensorflow as tf;
     from tensorflow.keras import regularizers;
     from tensorflow.keras.optimizers import Adam

     def build_model(hp):
       REG = 0.001;
       DROP = 0.1;

       model = tf.keras.Sequential();
       model.add(tf.keras.layers.Flatten());

       model.add(tf.keras.layers.Dense(hp.Int("units",min_value=128,max_value=1024,step=128),
                                       activation='sigmoid', kernel_regularizer=regularizers.l2(REG)));
       model.add(tf.keras.layers.Dropout(DROP));

       model.add(tf.keras.layers.Dense(2, activation='softmax'));

       model.compile(optimizer=Adam(learning_rate=5e-5), loss='categorical_crossentropy', metrics=['accuracy']);

       return model
```

FONTE: Autoria Própria.

A Figura 19 mostra a criação da variável *tuner* que será utilizada na criação do arquivo *history* e, consequentemente, no treinamentos das redes. É nesta variável, através do parâmetro *max_trials*, que é declarada a quantidade máxima de tentativas que serão feitas, ou seja, o número máxima de redes que serão treinadas e testadas. Para o caso deste trabalho o número máximo de dez tentativas foi definido.

Com as configurações já preparadas pelos trechos de código anteriores, a Figura 20 mostra como se deu a criação do arquivo *history* para esta abordagem. É possível observar que o número de épocas utilizadas para cada rede foi de 500 épocas, já que um número maior dificultaria o processamento de cada um dos modelos e tornaria inviável o treinamento dos mesmos. Ao final da execução, aquele modelo que obteve um maior valor de precisão teve seu arquivo *history* salvo no repositório do *Google Drive*.

Os resultados obtidos para esta abordagem e que foram atribuídos à variável *history* podem ser vistos na Seção Resultados para Otimização de Arquitetura Através dos Perceptrons da Camada Oculta.

Figura 19 – Definição da Variável Tuner para Otimização de Arquitetura por Número de Perceptrons da Camada Oculta.

```
[ ]  import keras_tuner as kt;
     from kerastuner.tuners import RandomSearch
     from tensorflow import keras;

     tuner = kt.RandomSearch(
         build_model,
         objective="val_accuracy",
         max_trials=10,
         overwrite=True,
         directory="/content/drive/MyDrive/ColabNotebooks/Documents5",
         project_name="saved_model",
     )
```

FONTE: Autoria Própria.

Figura 20 – Criação do Arquivo History para Otimização de Arquitetura por Número de Perceptrons da Camada Oculta.

```
[ ]  tuner.search(xTrain, yTrainHot, epochs=500, validation_split=0.2,
                  callbacks = [tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',
                                                                min_delta=0, patience=100)]);

     bestModels = tuner.get_best_models(num_models=1)
     highestScoreModel= bestModels[0]

[ ]  history = highestScoreModel.fit(x=xTrain, y=yTrainHot, batch_size=64, epochs=500,
                           verbose=1, validation_split=0.2)

     highestScoreModel.save("/content/drive/MyDrive/ColabNotebooks/Documents5")
```

FONTE: Autoria Própria.

3.7 TESTE PARA ARQUITETURA BÁSICA COM DIFERENTES TAMANHOS DE IMAGENS

Com a intenção de comprovar os resultados obtidos pelo treinamento da arquitetura básica apresentada na Seção Código Base, bem como verificar a possibilidade de se haver uma maior eficiência da rede caso as imagens do banco de dados tivessem dimensões diferentes de 160x120 pixels, que foi a dimensão utilizada para todos os outros cenários descritos neste trabalho, os códigos mostrados abaixo foram desenvolvidos.

A Figura 21 mostra o código escrito para a construção do modelo da rede, sendo este igual ao utilizado na construção da arquitetura básica apresentada na Seção Código Base.

Figura 21 – Criação do Modelo de Rede para Arquitetura Básica com Imagens maiores.

```
import tensorflow as tf;
from tensorflow.keras import regularizers;
from tensorflow.keras.optimizers import Adam

REG = 0.001;
DROP = 0.1;

model = tf.keras.Sequential();
model.add(tf.keras.layers.Flatten());

model.add(tf.keras.layers.Dense(768, activation='sigmoid', kernel_regularizer=regularizers.l2(REG)));
model.add(tf.keras.layers.Dropout(DROP));

model.add(tf.keras.layers.Dense(2, activation='softmax'));

model.compile(optimizer=Adam(learning_rate=5e-5), loss='categorical_crossentropy', metrics=['accuracy']);
```

FONTE: Autoria Própria.

A partir do modelo definido, o último passou foi o treinamento do mesmo, bem como a geração do arquivo *history*, como pode ser visto na Figura 22.

Figura 22 - Criação do Arquivo History para Arquitetura Básica com Imagens Maiores.

```
history = model.fit(xTrain,yTrainHot,epochs=1000,validation_split=0.2,
                    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',
                                                                min_delta=0, patience=100)]);
```

FONTE: Autoria Própria.

Vale destacar que três dimensões de imagens foram testadas para este modelo e que os resultados obtidos para cada uma delas serão exibidos na Seção Resultados para Arquitetura Básica com Diferentes Tamanhos de Imagens.

3.8 TESTE PARA ARQUITETURA BÁSICA COM TAMANHO DE CONJUNTOS DE TREINAMENTO DIFERENTES

A última abordagem explorada foi a de testes a partir da arquitetura básica onde o tamanho dos conjuntos de treinamento e de validação foram alterados. Esta abordagem é muito parecida com as de otimização da arquitetura, porém com a diferença de que aqui o que muda entre um modelo treinado e outro são as imagens utilizadas para treinamento e teste, bem como o tamanho do banco de dados para cada um deles.

A Figura 23 mostra como se deu a criação do modelo da rede neural artificial, bem como o armazenamento dos arquivos *history* gerados ao final de cada laço de

repetição. A lógica utilizada para esta abordagem foi a de se alterar um dos parâmetros do método *train_test_split* que é responsável pela divisão de imagens de treinamento e de validação em suas respectivas variáveis. Quatro valores diferentes de *splits* foram utilizados, sendo eles 20%, 30%, 40% e 50%. No mais, 500 épocas também foram utilizadas para cada um dos modelos, pelos mesmos motivos apresentados nas Seções anteriores.

Os resultados obtidos na última etapa e que foram atribuídos à variável *history* podem ser vistos na Seção Resultados para Arquitetura Básica com Tamanho de Conjuntos de Treinamento Diferentes.

Figura 23 – Criação do Modelo e do Histoty de Rede Neural para Arquitetura Básica com Tamanho de Conjunto de Imagens Diferentes.

```
from sklearn.model_selection import train_test_split;
from tensorflow.keras import regularizers;
from tensorflow.keras.optimizers import Adam
import tensorflow as tf;
import numpy as np

REG = 0.001;
DROP = 0.1;

for x in range(2, 6):

  xTrain, xTest, yTrain, yTest = train_test_split(trainInputImages, trainLabels,
                                                  test_size=0.1*x, random_state=42);
  xTrain = np.array(xTrain);
  xTest = np.array(xTest);
  xTrain = xTrain[..., tf.newaxis]
  xTest = xTest[..., tf.newaxis]

  yTestHot = tf.one_hot(yTest,2);
  yTrainHot = tf.one_hot(yTrain,2);

  model = tf.keras.Sequential();
  model.add(tf.keras.layers.Flatten());

  model.add(tf.keras.layers.Dense(768, activation='sigmoid', kernel_regularizer=regularizers.l2(REG)));
  model.add(tf.keras.layers.Dropout(DROP));

  model.add(tf.keras.layers.Dense(2, activation='softmax'));

  model.compile(optimizer=Adam(learning_rate=5e-5), loss='categorical_crossentropy', metrics=['accuracy']);

  history = model.fit(xTrain,yTrainHot,epochs=500,validation_split=0.1*x,
                      callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',
                                                                  min_delta=0, patience=100)]);
```

FONTE: Autoria Própria.

4 RESULTADOS

Este Capítulo apresenta os resultados obtidos para cada um dos experimentos realizados e descritos nas Seções do Capítulo Metodologia.

4.1 RESULTADOS PARA ARQUITETURA BÁSICA

A rede neural artificial desenvolvida nesta abordagem obteve resultados bastante interessantes, tendo uma precisão máxima aproximada de 98,57% durante a fase de treinamento do modelo e uma precisão máxima aproximada de 93,30% para a fase de validação. Esses resultados podem ser vistos nas Figuras .

Figura 24 - Resultados Obtidos para Imagens de Treinamento de Arquitetura Básica.

```
model.evaluate(xTrain,yTrainHot,verbose=2)

101/101 - 3s - loss: 0.1227 - accuracy: 0.9857 - 3s/epoch - 30ms/step
[0.122696153819561, 0.9857142567634583]
```

FONTE: Autoria Própria.

Figura 25 - Resultados Obtidos para Imagens de Validação de Arquitetura Básica.

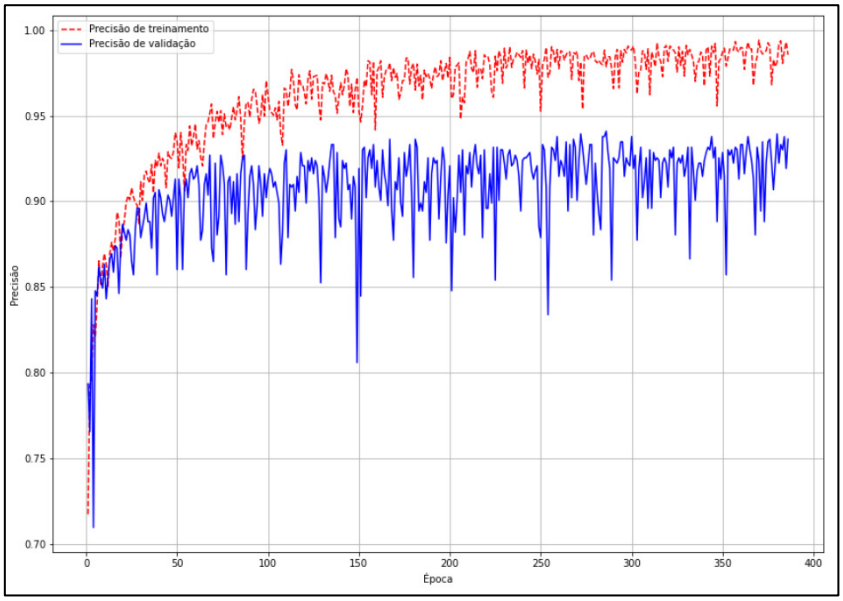
```
model.evaluate(xTest,yTestHot,verbose=2)

26/26 - 1s - loss: 0.2449 - accuracy: 0.9330 - 799ms/epoch - 31ms/step
[0.24493443965911865, 0.9330024719238281]
```

FONTE: Autoria Própria

Já na Figura 26 é possível observar a evolução da precisão do modelo de acordo com o passar das épocas. É importante salientar que entre as épocas de número 350 e 400 houve uma interrupção do treinamento do modelo. Isso ocorreu pois o modelo não identificou grandes alterações de precisão nas últimas 100 épocas executadas.

Figura 26 – Evolução da Precisão de Acordo com a Época para Arquitetura Básica.



FONTE: Autoria Própria

4.2 RESULTADOS PARA OTIMIZAÇÃO DE ARQUITETURA ATRAVÉS DO PARÂMETRO LEARNING RATE

Para a simulação da otimização de arquitetura, onde mais de uma rede neural artificial foi construída, os resultados podem ser vistos na Tabela 1. Três redes diferentes foram treinadas a partir deste método e obtiveram pontuações variando entre 91,45% e 93,63%. Essa pontuação não se trata da precisão máxima de cada modelo, e sim um sistema interno do método que classifica os modelos para que o melhor seja escolhido. Como podemos perceber, o modelo que obteve um melhor desempenho neste cenário foi o que se utilizou de um *Learning Rate* de $5e^{-5}$. Este teste comprova a correta utilização desde valor de parâmetro na arquitetura básica, dado o cenário, a metodologia e a base de dados descritos durante o desenvolvimento deste trabalho.

Tabela 1 – Pontuações Obtidas Para Cada Learning Rate Durante Otimização da Arquitetura.

Tentativa	Learning Rate	Pontuação
1	$5e^{-5}$	93,63%
2	$1e^{-5}$	92,55%
3	$5e^{-6}$	91,46%

FONTE: Autoria Própria

A partir da variável *tuner*, que guarda todos os modelos treinados, o melhor dos modelos é então atribuído para a variável *model* para que seus resultados sejam exibidos. Esta atribuição pode ser vista na Figura 27.

Figura 27 – Atribuição do Melhor Modelo para Otimização de Arquitetura através do Learning Rate.

```
model = tuner.get_best_models(num_models=1)
```

FONTE: Autoria Própria

A rede neural artificial reconhecida como a mais eficiente entre as treinadas nesta abordagem obteve resultados bem próximos aos encontrados a partir da arquitetura básica, tendo uma precisão máxima aproximada de 97,17% durante a fase de treinamento do modelo e uma precisão máxima aproximada de 90,57% para a fase de validação. Esses resultados podem ser vistos nas Figuras 28 e 29.

Figura 28 - Resultados Obtidos para Imagens de Treinamento de Otimização de Arquitetura Através do Learning Rate.

```
model[0].evaluate(xTrain,yTrainHot,verbose=2)

101/101 - 5s - loss: 0.1937 - accuracy: 0.9717 - 5s/epoch - 46ms/step
[0.1937445104122162, 0.9717391133308411]
```

FONTE: Autoria Própria

Figura 29 – Resultados Obtidos para Imagens de Validação de Otimização de Arquitetura Através do Learning Rate.

```
model[0].evaluate(xTest,yTestHot,verbose=2)

26/26 - 2s - loss: 0.3113 - accuracy: 0.9057 - 2s/epoch - 62ms/step
[0.31128281354904175, 0.9057071805000305]
```

FONTE: Autoria Própria

4.3 RESULTADOS PARA OTIMIZAÇÃO DE ARQUITETURA ATRAVÉS DOS PERCEPTRONS DA CAMADA OCULTA

Para a simulação da otimização de arquitetura através da alteração de valores atribuídos ao parâmetro que define o número de *Perceptrons* da camada oculta, onde mais de uma rede neural artificial foi construída, os resultados podem ser vistos na

Tabela 2. No total, oito redes diferentes foram treinadas a partir deste método e obtiveram pontuações que variaram entre 93,48% e 94,10%. Assim como no cenário apresentado na Seção anterior, essa pontuação não se trata da precisão máxima de cada modelo, e sim um sistema interno do método que classifica os modelos para que o melhor seja escolhido. Essas pontuações mostram que os melhores valores para o parâmetro de número de *Perceptrons* são 256 e 768, visto que ambos obtiveram um resultado de 94,10%. Este resultado comprova a correta utilização do valor 768 para o parâmetro referido durante a construção da arquitetura básica, dado o cenário, a metodologia e a base de dados descritos neste trabalho.

Tabela 2 – Pontuações Obtidas Para Cada Número de Perceptrons Durante Otimização da Arquitetura.

Tentativa	Número de *Perceptrons*	Pontuação
1	128	93,48%
2	256	94,10%
3	384	93,94%
4	512	93,48%
5	640	93,48%
6	768	94,10%
7	896	93,79%
8	1024	93,79%

FONTE: Autoria Própria

A partir da variável *tuner*, que guarda todos os modelos treinados, o melhor dos modelos é então atribuído para a variável *model* para que seus resultados sejam exibidos. Esta atribuição pode ser vista na Figura 30.

Figura 30 – Atribuição do Melhor Modelo para Otimização de Arquitetura.

```
model = tuner.get_best_models(num_models=1)
```

FONTE: Autoria Própria

A rede neural artificial reconhecida como a mais eficiente entre as treinadas nesta abordagem obteve resultados bem próximos aos encontrados a partir da arquitetura básica, tendo uma precisão máxima aproximada de 98,57% durante a fase de treinamento do modelo e uma precisão máxima aproximada de 92,18% para a fase de validação. Esses resultados podem ser vistos nas Figuras 31 e 32.

Figura 31 - Resultados Obtidos para Imagens de Treinamento de Otimização de Arquitetura Através dos Perceptrons da Camada Oculta.

```
model[0].evaluate(xTrain,yTrainHot,verbose=2)

101/101 - 5s - loss: 0.1324 - accuracy: 0.9857 - 5s/epoch - 49ms/step
[0.1324131339788437, 0.9857142567634583]
```

FONTE: Autoria Própria

Figura 32 – Resultados Obtidos para Imagens de Validação de Otimização de Arquitetura Através dos Perceptrons da Camada Oculta.

```
model[0].evaluate(xTest,yTestHot,verbose=2)

26/26 - 3s - loss: 0.2560 - accuracy: 0.9218 - 3s/epoch - 109ms/step
[0.25598782300949097, 0.921836256980896]
```

FONTE: Autoria Própria

4.4 RESULTADOS PARA ARQUITETURA BÁSICA COM DIFERENTES TAMANHOS DE IMAGENS

No caso do teste realizado para arquitetura básica onde imagens com dimensões maiores foram utilizadas no treinamento, os resultados podem ser vistos na Tabela 3.

Os resultados apresentados abaixo levam em consideração o banco de imagens apresentado na Seção Captação de Imagens para o Banco de Dados do Capítulo Metodologia, onde mais de 5.000 imagens fazem parte de sua composição. O alto número de imagens do banco, juntamente com o fato de que um servidor de moderada capacidade foi utilizado, são fatores que explicam o estouro de memória RAM e, consequentemente, a falha nas tentativas de treinamento do modelo. Estes resultados mostram que o tamanho das imagens utilizadas durante a construção da arquitetura básica é o apropriado para o cenário deste trabalho.

Tabela 3 – Resultados Obtidos para Imagens de Treinamento de Arquitetura Básica com Imagens Maiores.

Tentativa	Dimensões das Imagens	Resultado
1	800 x 600	Não foi possível completar o treinamento do modelo devido ao estouro de memória durante a fase de carregamento das imagens em formato de *DataFrame* no *Google Colab*.
2	400 x 300	Não foi possível completar o treinamento do modelo devido ao estouro de memória durante a fase de conversão das imagens do formato de *DataFrame* para Lista.
3	200 x 150	Não foi possível completar o treinamento do modelo devido ao estouro de memória durante a fase de normalização das imagens.

FONTE: Autoria Própria

4.5 RESULTADOS PARA ARQUITETURA BÁSICA COM TAMANHO DE CONJUNTOS DE TREINAMENTO DIFERENTES

Nesta abordagem quatro diferentes modelos foram treinados, cada um possuindo um número diferente de imagens para treinamento, validação e teste. Esta divisão foi realizada durante a criação do arquivo *history* e os valores utilizados, bem como os resultados obtidos em cada tentativa, podem ser vistos na Tabela 4.

Como pode ser visto abaixo, as precisões máximas obtidas durante a fase de treinamento dos modelos se manteve com valores altos em todos os casos, enquanto as precisões máximas obtidas durante a fase de testes tendem a reduzir entre os modelos na medida em que a quantidade de imagens utilizadas nos treinamentos aumentou. Os resultados obtidos neste teste demonstram que as quantidades de imagens divididas para treinamento e validação, bem como para testes, durante a construção da arquitetura básica foram os que apresentaram maiores números de precisão final para o modelo construído a partir do cenário, da metodologia e do banco de dados apresentados neste trabalho.

Tabela 4 – Resultados Obtidos Durante Testes de Arquitetura Básica com Tamanhos de Conjuntos de Treinamento Diferentes.

Tentativa	Split de Imagens Utilizado	Precisão Máxima Alcançada
1	80% para Treinamento 20% para Validação	99,45% durante Treinamento 93,78% durante Validação
2	70% para Treinamento 30% para Validação	99,84% durante Treinamento 92,79% durante Validação
3	60% para Treinamento 40% para Validação	99,24% durante Treinamento 90,78% durante Validação
4	50% para Treinamento 50% para Validação	99,60% durante Treinamento 88,58% durante Validação

FONTE: Autoria Própria

4.6 RESUMO DOS RESULTADOS

A Tabela 5 abaixo mostra um resumo dos resultados apresentados neste Capítulo. Nela é possível ver os melhores valores obtidos em casa cenário de treinamento, otimização e teste.

É possível observar que todos os resultados encontrados possuem valores muito próximos, o que é um forte indicativo de que a metodologia utilizada no código base, bem como na arquitetura básica, é robusta e assertiva. No fim, todos os casos de otimização e testes realizados após a construção da arquitetura básica demonstraram que o modelo criado se utilizou de métodos e valores de parâmetros precisos e que a chance deste estar sob o efeito de um possível "vício" é mínima.

Tabela 5 – Resumo dos Resultados Obtidos.

Cenário	Precisão Máxima Obtida Durante Treinamento	Precisão Máxima Obtida Durante Validação
Arquitetura Básica	98,57%	93,30%
Otimização de *Learning Rate*	97,17%	90,57%
Otimização de *Perceptrons*	98,57%	92,18%
Diferentes Tamanhos de Imagens	Não Encontrada	Não Encontrada
Diferentes Tamanhos de *splits*	99,45%	93,78%

FONTE: Autoria Própria

Por fim, para efeito de validação do modelo desenvolvido, a Tabela 6 mostra um teste realizado com imagens aleatórias contidas no banco de dados separado apenas para testes, sendo estas imagens não utilizadas em nenhum processo de treinamento ou de validação do modelo. A Tabela mostra a imagem em questão na primeira coluna, a informação de haver ou não uma deformação na pista na segunda coluna, e a saída da Rede na terceira. Para esta última, podemos ler o valor 0 como a falta de deformação na pista e o valor 1 como a presença de uma deformação. Neste caso, oito das dez validações das imagens foram assertivas. O código utilizado para os testes das imagens pode ser visto no Apêndice IV.

Tabela 6 – Teste do Modelo de Arquitetura Básica.

Imagem Utilizada em Teste	Há Deformação na Pista?	Saída da Rede
	Sim	[1, 0]
	Não	[1, 0]
	Não	[1, 0]

	Sim	[0, 1]
	Sim	[0, 1]
	Não	[1, 0]
	Não	[1, 0]

	Não	[1, 0]
	Não	[0, 1]
	Não	[1, 0]

FONTE: Autoria Própria

5 CONSIDERAÇÕES FINAIS

Neste trabalho, uma rede neural artificial do tipo *MultiLayer Perceptron* foi construída a partir da linguagem de programação Python, onde seu objetivo seria o reconhecimento de deformações em estradas e pistas utilizadas no trânsito de veículos urbanos. O modelo obtivo através da rede se mostrou bastante assertivo e sua precisão foi validada através das otimizações e testes realizados posteriormente por meio da confirmação dos valores dos parâmetros e das configurações do banco de dados utilizados na construção da arquitetura básica do modelo.

A proposta inicial do projeto seria, além do treinamento do modelo de rede neural artificial descrito acima, a implementação do mesmo em um sistema embarcado tal qual o *Arduino* ou o *Raspberry Pi*. Como podemos ver, apesar do código escrito para a arquitetura básica conter um trecho que transforma o modelo em um arquivo do tipo *TFLite*, estando este disponível no Apêndice III, não foi possível embarcar o sistema a tempo da entrega e da apresentação deste trabalho. Isso se deu, pois foram encontradas diversas restrições no hardware do protótipo do robô autônomo que foi utilizado como o sistema embarcado do projeto, como, por exemplo, peças que não funcionavam e que não poderiam ser trocadas em um curto espaço de tempo.

Podemos descrever algumas possíveis ações que devem ser tomadas na continuação deste trabalho, como, por exemplo, a criação e/ou a utilização de um banco de imagens contendo fotografias referentes a estradas e pistas localizadas em diferentes estados e cidades ou, caso este seja o caso, referentes a um escopo de estudo específico. Além disso, o modelo criado neste trabalho pode ser embarcado em algum tipo de microcontrolador e a sua utilização no dia a dia pode ser estudada para que os resultados obtidos neste e nos trabalhos posteriores sejam comprovados.

REFERÊNCIAS

Aguiar, N. A., & Silva, I. J. (2021). *Desenvolvimento de um Robô Móvel para o setor de Armazenamento.* Fonte: XII Simpósio de Iniciação Científica, Didática e de Ações Sociais da FEI: https://fei.edu.br/sites/sicfei/artigos2022/SICFEI_2022_paper_68.pdf

Arduino. (2022). *About Arduino*. Acesso em 21 de Novembro de 2022, disponível em Arduino: https://www.arduino.cc/en/about

Arduino. (2022). *Mega 2560 Rev3*. Acesso em 21 de Novembro de 2022, disponível em Arduino: https://docs.arduino.cc/hardware/mega-2560

Borges, L. E. (2014). *Python para Desenvolvedores.* São Paulo: Novatec.

Channel, i. (2022). *Qual foi a primeira câmera fotográfica do mundo?* Acesso em 15 de 12 de 2022, disponível em iPhoto Channel: https://iphotochannel.com.br/primeira-camera-fotografica-do-mundo/

Costa, A. H., & Pegoraro, R. (01 de Set/Out/Nov de 2000). CONSTRUINDO ROBÔS AUTÔNOMOS PARA PARTIDAS DE FUTEBOL: O TIME GUARANÁ. *SBA Controle & Automação, 11*, pp. 141-149. Acesso em Novembro de 2022, disponível em https://www.sba.org.br/revista/vol11/v11a259.pdf

DeepLearningBook. (2022). *Capítulo 6 – O Perceptron – Parte 1*. Acesso em 11 de Janeiro de 2022, disponível em DeepLearningBook: https://www.deeplearningbook.com.br/o-perceptron-parte-1/#:~:text=Perceptron%20%C3%A9%20uma%20rede%20neural,os%20dados%20de%20entrada%20fornecidos.

Drive. (2022). *Visão Geral*. Acesso em 16 de 12 de 2022, disponível em Google Drive: https://www.google.com/intl/pt-BR/drive/

Filho, O. M., & Neto, H. V. (1999). *Processamento Digital de Imagens.* Rio de Janeiro: Brasport. Fonte: http://projetoaprendizagemgrupo4.pbworks.com/w/file/fetch/96395952/Processamento%20Digital%20de%20Imagens.pdf

FilipeFlop. (2022). *O que são shields para Arduino?* Acesso em 21 de Novembro de 2022, disponível em FilipeFlop: https://www.filipeflop.com/blog/o-que-sao-shields-para-arduino/

Folha de São Paulo. (20 de Dezembro de 2021). *Folha de São Paulo*. Acesso em 11 de Janeiro de 2023, disponível em Folha de São Paulo: https://piaui.folha.uol.com.br/muito-buraco-para-pouco-asfalto/#:~:text=Em%202021%2C%20a%20CNT%20identificou,os%20pneus%20de%20um%20carro.

Frazão, D. (2022). *Isaac Asimov*. Acesso em 2022, disponível em eBiografia: https://www.ebiografia.com/isaac_asimov/

Gadelha, J. (2022). *A Evolução dos Computadores*. Acesso em 2022, disponível em UFF: http://www.ic.uff.br/~aconci/evolucao.html

Gonzalez, R. C., & Woods, R. E. (2000). *Processamento de Imagens Digitais.* Blucher. Fonte: https://books.google.com.br/books?hl=pt-BR&lr=&id=9CbTDwAAQBAJ&oi=fnd&pg=PA1&dq=processamento+digital+de+imagens&ots=zhrFRH9Vui&sig=k9UR1HwE5Q5DxsX5onnqgW0gPqI#v=onepage&q&f=false

Google. (2022). *Google Colab*. Acesso em 16 de 12 de 2022, disponível em Colab: https://colab.research.google.com/

Google. (2022). *Imagem*. Acesso em 15 de 12 de 2022, disponível em Google: https://www.google.com/search?q=imagem+dicion%C3%A1rio&oq=imagem+dicion%C3%A1rio

Kaggle. (2023). *MIIA Pothole Image Classification Challenge*. Acesso em 19 de Janeiro de 2023, disponível em Kaggle: https://www.kaggle.com/datasets/salimhammadi07/miia-pothole-image-classification-challenge

Keras. (2022). *Keras*. Acesso em 16 de 12 de 2022, disponível em Keras.

Kovács, Z. L. (2006). *Redes Neurais Artificiais.* São Paulo: Livraria da Física.

Mataric, J. M. (2007). *Introdução à Robótica.* São Paulo: Unesp.

McRoberts, M. (2015). *Arduino Básico.* São Paulo: Novatec Editora Ltda.

MeioBit. (2008). *A primeira câmera digital do mundo*. Acesso em 15 de 12 de 2022, disponível em MeioBit: https://meiobit.com/15452/a-primeira-camera-digital-do-mundo/

Open Source. (2022). *O que é Raspberry Pi?* Acesso em 21 de Novembro de 2022, disponível em Open Source: https://opensource.com/resources/raspberry-pi#:~:text=Raspberry%20Pi%20is%20the%20name,and%20variations%20released%20since%20then.

Patos Hoje. (11 de Janeiro de 2023). *Buraco em ponte da BR 365 estoura vários pneus, causa acidentes e motoristas cobram providências*. Acesso em 11 de Janeiro de 2023, disponível em Patos Hoje: https://www.patoshoje.com.br/noticias/buraco-em-ponte-da-br-365-estoura-varios-pneus-causa-acidentes-e-motoristas-cobram-providencias-77205.html

Raspberry Pi. (2022). *About Us*. Acesso em 21 de Novembro de 2022, disponível em Raspberry Pi: https://www.raspberrypi.org/about/

Raspberry Pi 4 Tech Specs. (2022). Acesso em 21 de Novembro de 2022, disponível em Raspberry Pi: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/

Simonyan, K., & Zisserman, A. (04 de Setembro de 2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *Cornell University*.

TechTudo. (2016). *Conheça os robôs mais curiosos criados com o Raspberry Pi*. Acesso em 21 de Novembro de 2022, disponível em Tech Tudo: https://www.techtudo.com.br/noticias/2016/10/conheca-os-robos-mais-curiosos-criados-com-o-raspberry-pi.ghtml

TensorFlow. (2022). *Visão Geral*. Acesso em 16 de 12 de 2022, disponível em TensorFlow: https://www.tensorflow.org/overview

Terminal Root. (19 de Dezembro de 2019). *As 30 melhores bibliotecas e pacotes Python para Iniciantes*. Acesso em 18 de Janeiro de 2023, disponível em Terminal Root: https://terminalroot.com.br/2019/12/as-30-melhores-bibliotecas-e-pacotes-python-para-iniciantes.html

UFCG, I. (2020). *O Que É Um Microcontrolador?* Acesso em 21 de Novembro de 2022, disponível em IEEE RAS UFCG: https://edu.ieee.org/br-ufcgras/o-que-e-um-microcontrolador/

APÊNDICE I

Segue abaixo o link para o desafio da plataforma *Kaggle*, onde o banco de imagens utilizado neste trabalho foi retirado:

https://www.kaggle.com/datasets/salimhammadi07/miia-pothole-image-classification-challenge

APÊNDICE II

Segue abaixo o link para o repositório do aluno, onde todos os códigos desenvolvidos neste trabalho foram salvos:

https://github.com/euvictorfarias/pothole-identifier-python

APÊNDICE III

Segue abaixo, na Figura 33, o código escrito para que o modelo criado neste trabalho pudesse ser convertido em um arquivo do tipo *TFLite* e que, consequentemente, fosse possível de ser embarcado:

Figura 33 – Conversão do Modelo Criado para TFLite.

```
import tensorflow as tf;

converter = tf.lite.TFLiteConverter.from_saved_model('/content/drive/MyDrive/ColabNotebooks/Documents/');
tfliteModel = converter.convert();
```

FONTE: Autoria Própria

APÊNDICE IV

Segue abaixo, na Figura 34, o código escrito para que o modelo criado neste trabalho fosse testado:

Figura 34 – Código Utilizado para Teste do Modelo de Arquitetura Básica.

```
import tensorflow as tf;
interpreter = tf.lite.Interpreter('/content/drive/MyDrive/ColabNotebooks/Documents/modelTfLite.tflite');
interpreter.allocate_tensors();

import cv2 as cv;
import numpy as np;
import matplotlib.pyplot as plt

index = 30;
imagesPath = '/content/drive/MyDrive/ColabNotebooks/Images/all_data/';
image = testImagesDF['Image'][index];
label = '/content/drive/MyDrive/ColabNotebooks/Images/all_data/' + testImagesDF['Label'][index];

testImageForModel = cv.imread(label, 1);
testImageForModel2 = cv.cvtColor(testImageForModel, cv.COLOR_RGB2GRAY);
testImageForModel2 = cv.resize(testImageForModel2, (160, 120));
testImageForModel3 = np.array(testImageForModel2, dtype=np.float32) / 255.0;
testImageForModel3 = testImageForModel3[np.newaxis, ..., np.newaxis];

interpreter.set_tensor(0, testImageForModel3)
interpreter.invoke()
output = interpreter.get_tensor(10)
```

FONTE: Autoria Própria

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
Campus João Pessoa
Av. Primeiro de Maio, 720, Jaguaribe, CEP 58015-435, João Pessoa (PB)
CNPJ: 10.783.898/0002-56 - Telefone: (83) 3612.1200

Documento Digitalizado Ostensivo (Público)

Relatório de TCC Atualizado

Assunto: Relatório de TCC Atualizado
Assinado por: Victor Farias
Tipo do Documento: Anexo
Situação: Finalizado
Nível de Acesso: Ostensivo (Público)
Tipo do Conferência: Cópia Simples

Documento assinado eletronicamente por:

- **Carlos Victor dos Santos Farias**, **ALUNO (20162610001) DE BACHARELADO EM ENGENHARIA ELÉTRICA - JOÃO PESSOA**, em 16/03/2023 18:11:20.

Este documento foi armazenado no SUAP em 16/03/2023. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 781230
Código de Autenticação: 0622fe769f