UNIVERSIDADE REGIONAL INTEGRADA DO ALTO URUGUAI E DAS MISSÕES
CAMPUS DE ERECHIM
DEPARTAMENTO DE ENGENHARIAS E CIÊNCIA DA COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GIOVANI ANDRÉ MENEGUEL

VISION DETECT: SISTEMA PARA DETECÇÃO E CONTAGEM DE OBJETOS

ERECHIM - RS
2021

GIOVANI ANDRÉ MENEGUEL

VISION DETECT: SISTEMA PARA DETECÇÃO E CONTAGEM DE OBJETOS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do grau de Bacharel, Departamento de Engenharias e Ciência da Computação da Universidade Regional Integrada do Alto Uruguai e das Missões Campus de Erechim.

Orientador: Prof. MSc. Daniel Menin Tortelli

ERECHIM - RS
2021

AGRADECIMENTOS

Primeiramente agradeço a minha família por todo o suporte dado durante a graduação e minha formação profissional, e principalmente por estarem ao meu lado em todos os momentos da vida. Agradeço a minha mãe Vilma, meu pai Geraldo, meus irmãos Diego, Roni e Renan e em especial a minha irmã Leonice que sempre me motivou e me ajudou no decorrer de toda minha caminhada. Também agradeço ao meu sobrinho Anthony por trazer leveza e felicidade nesta difícil trajetória. Muito obrigado por tudo, amo vocês.

É de suma importância fazer um agradecimento muito especial a todos aqueles que pavimentaram a estrada para que eu pudesse seguir aprendendo e me desafiando, os professores. Obrigado a todos os educadores que de alguma forma se eternizaram em mim através da educação. Gratidão aos professores do curso de Ciência da Computação por todos os ensinamentos, em especial ao Professor Daniel Menin por me orientar neste trabalho e também à Professora Cleusa Balestrin que me acompanha desde o ensino médio e me ajudou muito na revisão ortográfica.

Agradeço aos meus colegas pelos anos de estudo, ensinamentos e companheirismo. Ao meu colega Tiago Bellaver por todas as conversas e aprendizados, e de forma muito especial a Letícia May que esteve junto comigo em toda a graduação, parceira de (quase) todos os trabalhos, muito obrigado por me lembrar dos compromissos da faculdade e por sempre se fazer presente nos momentos difíceis.

Gostaria de agradecer também a todos os laços que perduram a muito tempo e aos que fiz durante a faculdade e no trabalho. É ao lado dos amigos que vale a pena celebrar a vida, por isso muito obrigado de coração a todos os meus amigos e amigas.

Para finalizar quero agradecer a todo mundo que acreditou em mim e de alguma forma se fez presente durante toda graduação e em especial na construção e desenvolvimento deste trabalho. Encerro esta etapa com o coração cheio de amor e felicidade por compartilhar esta jornada com pessoas tão especiais. Muito Obrigado!

Se a educação sozinha não transforma a sociedade, sem ela tampouco a sociedade muda.

(Paulo Freire)

RESUMO

O uso de tecnologias inovadoras está ganhando cada vez mais destaque em sistemas e utilitários do dia a dia. Os algoritmos de visão computacional se encaixam nesse cenário possibilitando sua usabilidade na automatização e simplificação de diversas atividades nas mais variadas áreas. Vinculada à estrutura de um sistema embarcado, essa tecnologia pode ser inovadora e útil trazendo mais acuracidade e rapidez para tarefas repetitivas, como por exemplo, a contagem de objetos de pequena escala. O objetivo deste trabalho foi construir um sistema embarcado que utiliza algoritmos de visão computacional com o intuito de reconhecer e realizar a contagem de objetos. O desenvolvimento das funcionalidades do sistema fez uso da linguagem Python. Foram utilizados os *frameworks* Django e TensorFlow para o desenvolvimento *web* e treinamento do modelo de detecção, respectivamente. O *dataset*, bem como as imagens utilizadas para a detecção, foram capturadas por meio do sistema embarcado. A estrutura de *hardware* do sistema embarcado foi desenvolvida utilizando os produtos advindos do Raspberry Pi.

Palavras-chave: Visão Computacional. Sistemas Embarcados. TensorFlow. Detecção de Objetos.

ABSTRACT

The use of innovative technologies is gaining more and more prominence in everyday systems and utilities. Computer vision algorithms fit into this scenario, enabling its usability in the automation and simplification of diverse activities in the most varied areas. Linked to the structure of an embedded system, this technology can be innovative and useful, bringing more accuracy and speed to repetitive tasks, such as counting small-scale objects. The objective of this work was to build an embedded system that uses computer vision algorithms in order to recognize and perform object counting. The development of system features made use of the Python language. Django and TensorFlow frameworks were used for web development and detection model training, respectively. The data set, as well as the images used for detection, were captured through the embedded system. The hardware structure of the embedded system were developed using the products accrued from the Raspberry Pi.

Keywords: Computer Vision. Embedded System. TensorFlow. Object Detection.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CSI	*Camera Serial Interface*
CSS	*Cascading Style Sheets*
CCD	*Charge Coupled Device*
CPU	*Central Process Unit*
CV	*Computer Vision*
GPIO	*General Purpose Input/Output*
GPU	*Graphics Processing Unit*
HDMI	*High-Definition Multimedia Interface*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
IP	*Internet Protocol*
JPEG	*Joint Pictures Expert Group*
JSON	*JavaScript Object Notation*
LED	*Light Emitting Diode*
MP	*Mega Pixels*
OpenCV	*Open Source Computer Vision Library*
Pascal VOC	*Pascal Visual Object Class*
PC	*Personal Computer*
PNG	*Portable Network Graphics*
RGB	*Red Green Blue*
SD	*Secure Digital*
UML	*Unified Modeling Language*
USB	*Universal Serial Bus*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Algoritmos de Visão Computacional são mencionados desde a década de 50, porém foi apenas na década de 80 que os primeiros trabalhos com essa tecnologia foram iniciados. O objetivo desse tipo de algoritmo é replicar a visão humana através de *hardware* e *software*, podendo assim fazer o reconhecimento do ambiente e extrair informações dele (BARELLI, 2018).

Outra área similar aos algoritmos citados acima, é o Processamento Digital de Imagens. Embora não exista um acordo geral entre os pesquisadores sobre o limiar de pesquisa entre as duas áreas, é possível afirmar que o processamento de imagens é parte fundamental do funcionamento de algoritmos de Visão Computacional. Uma das características que diferenciam as duas áreas é a utilização da inteligência artificial, comumente utilizada para reproduzir a visão humana e seus aspectos (GONZALEZ; WOODS, 2009).

Esses algoritmos podem ser utilizados para facilitar diversas ações cotidianas, e podem também ser empregados em diferentes áreas, tal como na área da saúde e na área industrial. Contudo, para o desenvolvimento dos mesmos, é necessário uma base de conhecimento matemático e de programação, bem como entender como é possível permitir um computador enxergar e entender o que está próximo a ele. Novas bibliotecas e linguagens estão surgindo para simplificar e abstrair boa parte do desenvolvimento, deixando os algoritmos mais fáceis de serem utilizados (BARELLI, 2018).

Baseado nessas informações o presente trabalho propõe uma solução que pode ser utilizada na organização de estoques ou na venda de materiais, trabalhando especificamente com objetos de pequena escala. Muitas vezes a contagem de pequenos itens pode ser exaustiva e imprecisa, principalmente se for feita de forma manual e repetitiva. Por esses motivos, a solução desenvolvida facilitará essa ação, utilizando algoritmos de Visão Computacional para reconhecer os itens dispostos em uma superfície. Ao reconhecer os objetos, o algoritmo pode gerar uma tabela com as informações dos itens e sua quantidade.

Este projeto uniu conhecimento de diferentes áreas da computação para criar um utilitário completo e funcional. O componente do *hardware* foi desenvolvido através da construção de um sistema embarcado com base em elementos do produto Raspberry Pi, utilizando além do microcomputador, o sistema operacional e o módulo de câmera. Foram feitas configurações para permitir o acesso externo ao aparelho e, também, disponibilizando um servidor que possibilite o *stream* das imagens da câmera, para que sejam utilizadas na parte lógica dos algoritmos. A estrutura do sistema embarcado foi montada e instalada de forma a facilitar a obtenção das imagens.

A solução desenvolvida possui um sistema *web* que facilita a interação do usuário com suas funcionalidades. Para a programação foi utilizada a linguagem de programação Python e as bibliotecas disponíveis. O sistema *web* foi desenvolvido com base no *framework* Django,

utilizando um banco de dados PostgreSQL que foi responsável por armazenar as informações e o *dataset* para o treinamento do modelo. O modelo de detecção foi treinado com base no *framework* TensorFlow e com o *dataset* criado através das imagens do sistema embarcado.

O presente trabalho é descrito em 7 capítulos, incluindo a Introdução. O Capítulo 2 descreve as principais tecnologias utilizadas no trabalho, através da fundamentação teórica. Na pesquisa bibliográfica são apresentados conceitos importantes para a construção do trabalho. O Capítulo 3 conceitua as ferramentas utilizadas no desenvolvimento do sistema. O Capítulo 4 aborda a implementação do sistema através dos diagramas, ferramentas utilizadas e descrição do desenvolvimento. O Capítulo 5 apresenta as telas do sistema *web*, bem como suas características e explicações. Para finalizar, o Capítulo 6 mostra os resultados obtidos através do treinamento do modelo e da detecção de objetos, enquanto o Capítulo 7 conclui o trabalho com as considerações finais do autor e os trabalhos futuros para continuidade desse trabalho.

2 TECNOLOGIAS

Este capítulo é dedicado à escrita da fundamentação teórica, a principal base de conhecimento para a construção do trabalho. Serão detalhados aspectos importantes em torno das tecnologias e conteúdos abordados no presente trabalho. Inicialmente será descrito como uma imagem é capturada e armazenada em um arquivo digital, em seguida um detalhamento de conceitos importantes relacionados ao campo de visão computacional. Para finalizar serão destacados os algoritmos de *deep learning*, utilizados para o treinamento do modelo de detecção, bem como informações sobre sistemas embarcados que serão utilizados como parte do *hardware* do programa.

2.1 Câmeras Digitais

As câmeras digitais são uma evolução das câmeras analógicas, pois ambas seguem o conceito utilizando uma lente para a captura dos feixes de luz, porém o armazenamento é feito de forma diferente, as câmeras digitais geram arquivos de mídia em formato digital em extensões comumente conhecidas como JPEG ou PNG. Já a captura da imagem é feita através de sensores eletrônicos fotossensíveis que transformam a luz que incide na lente em dados digitais (BARELLI, 2018).

O sensor muito utilizado em câmeras digitais, como *smartphones*, *webcams* e outros é o *Charge Coupled Device* (CCD). Esse tipo de sensor utiliza uma matriz de capacitores fotossensíveis para capturar a imagem e transformá-la, com a ajuda de um conversor, em informações binárias de acordo com a tensão elétrica produzida pelo mesmo. Cada placa recebe uma resposta diferente que é proporcional a energia luminosa projetada na sua superfície e representa um *pixel* na imagem (GONZALEZ; WOODS, 2009). A Figura 1 mostra como é feita a captura de uma imagem digital situando aspectos importantes da cena como o objeto fotografado, a iluminação, a refletância e o resultado da saída após o processo de quantização.

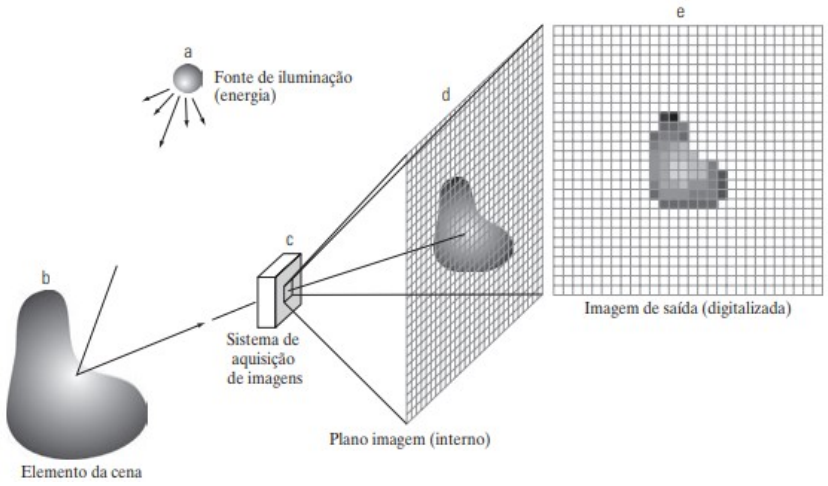
2.2 Formação de uma imagem digital

Uma imagem monocromática pode ser descrita como um arranjo numérico bidimensional descrito por *f(x,y)*. O valor de f é a intensidade luminosa de um ponto com coordenadas x e y. Esses pontos são chamados de *pixels* e juntos através de linhas e colunas formam uma imagem.

Uma imagem colorida seguindo o sistema de cores RGB, por exemplo, pode ser compreendida como o resultado da soma de três vetores que representam a intensidade de vermelho, verde e azul em cada ponto da matriz (QUEIROZ; GOMES, 2006).

A maior parte das imagens consiste na combinação de uma fonte de luz e a reflexão da

Figura 1 – Processo de aquisição de uma imagem digital.



Fonte: Gonzalez e Woods (2009)

mesma nos elementos da cena. Por esse motivo a função descrita anteriormente pode ser obtida através do produto de dois componentes denominados de iluminação e refletância. A iluminação é a quantidade de luz que incide na cena que está sendo vista e a refletância é a quantidade de luz refletida pelos objetos (GONZALEZ; WOODS, 2009).

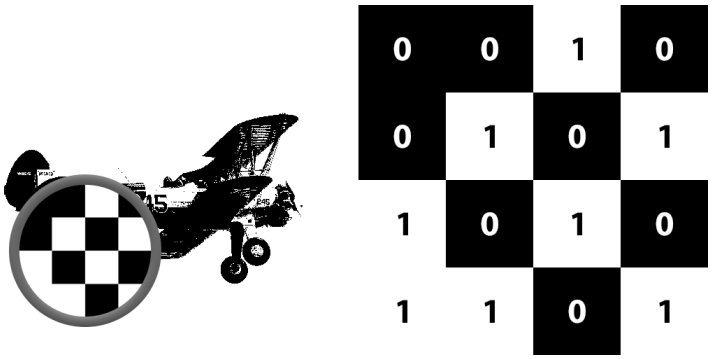
2.2.1 Espaços de Cores

Espaços de cores, ou sistemas de cores, são modelos matemáticos usados para descrever as colorações através de uma fórmula, facilitando a especificação das mesmas por meio de uma estrutura padronizada e aceita. Existem diversas formas de representação das cores e por meio delas é possível compor imagens de diferentes maneiras (GONZALEZ; WOODS, 2009).

O sistema binário é o mais simples para representar uma imagem. Neste sistema os *pixels* da imagem são representados pelo valor 0 que representa a cor preta ou pelo valor 1 que representa a cor branca. Para o registro da imagem é necessário definir um limiar para a intensidade da luz a fim de ser feita a separação dos *pixels* pretos e brancos. Esse sistema de cores é muito utilizado para a segmentação de objetos e extração de características da imagem. A Figura 2 apresenta uma imagem no sistema de cor binário e ao lado a formação da matriz de *pixels* da mesma.

A escala de cinza é feita apenas por um canal que representa a intensidade da luz sendo cada *pixel* representado por um valor inteiro entre 0 e 255. O valor 0 representa a cor preta e vai se tornando mais claro conforme é aumentado, sendo o valor máximo 255 que representa a cor branca. Esse sistema de cor é bastante usado em algoritmos de visão computacional, que não necessitem da cor do objeto, a fim de reconhecer e contar elementos. São consideradas boas alternativas por possuírem apenas um canal, com menos informações o processamento se torna mais rápido. A Figura 3 apresenta uma imagem no sistema de cor cinza e ao lado a formação

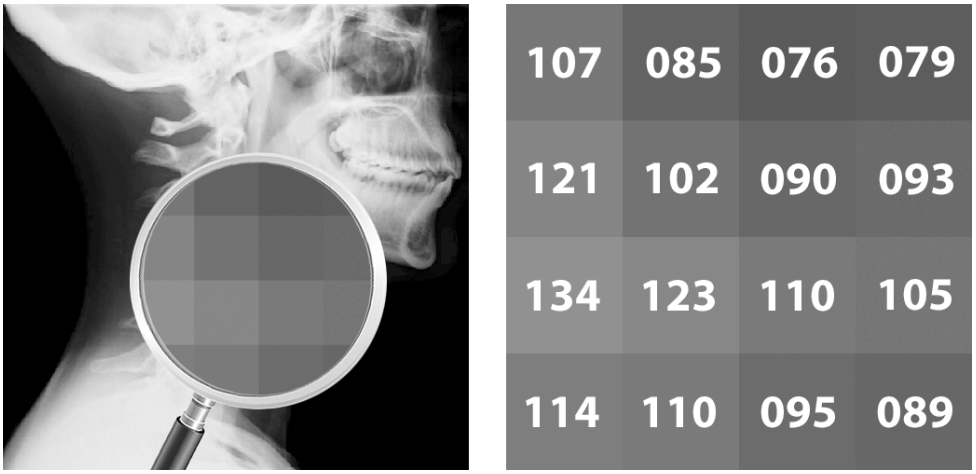
Figura 2 – Formação de uma imagem no sistema de cor binário.



Fonte: Barelli (2018)

da matriz de *pixels* da mesma.

Figura 3 – Formação de uma imagem no sistema de cor cinza.



Fonte: Barelli (2018)

O sistema RGB é o espaço de cor das imagens coloridas e é comumente usado na exibição das imagens de monitores e televisores. Esse sistema possui 3 canais de cores que representam a intensidade de verde, vermelho e azul para formar diferentes tonalidades. Cada *pixel* pode ser compreendido por um vetor que possui os três valores de intensidades citados anteriormente. .

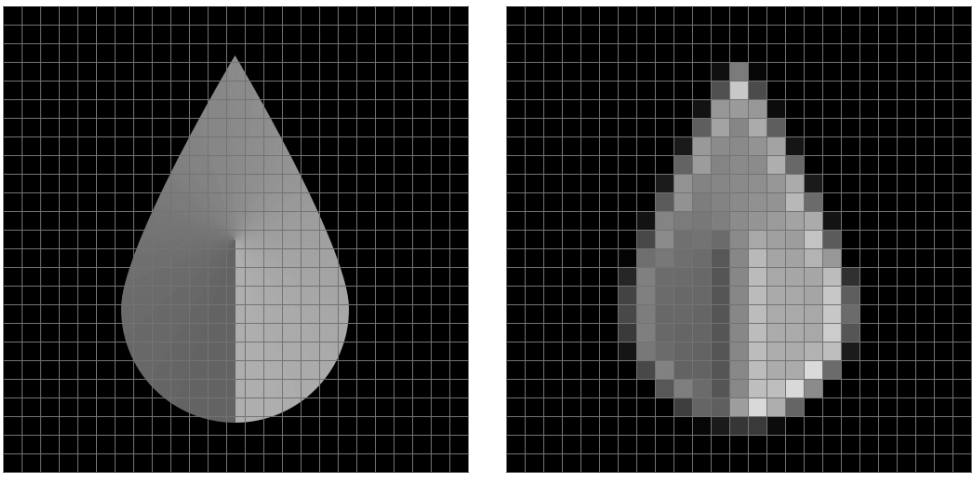
2.2.2 Resolução e Quantização

A resolução de uma imagem é a quantidade de informações que ela guarda, ou seja, quanto maior a resolução, maior o nível de detalhes e informações contidas na mesma. A resolução é medida com base nos números de *pixels* obtidos através do produto de colunas e linhas da imagem. Esse conceito implica também no armazenamento da imagem, pois dependendo da resolução o arquivo pode ocupar mais espaço (BARELLI, 2018).

A métrica de resolução também é usada nas câmeras digitais, sendo que uma câmera que possui 8MP, por exemplo, pode gerar uma imagem digital com cerca de 8 milhões de *pixels*. Essa medida é associada ao tamanho e detalhamento da fotografia, visto que quanto mais *pixels* maior e mais detalhada é a imagem.

O processo de quantização é a digitalização dos valores de amplitude. Seu objetivo é definir uma única cor para cada *pixel*. Em imagens obtidas através das câmeras digitais, esse processo é feito pelas placas dos sensores que obtém apenas uma informação, porém ao alterar as colunas e linhas de um arquivo a definição dos novos *pixels* é feita através do processo de quantização. Existem diversos métodos para obter esse resultado, como o cálculo da média por exemplo (QUEIROZ; GOMES, 2006). A Figura 4 demonstra o processo de quantização de uma imagem, comparando o antes e depois do processo.

Figura 4 – Processo de quantização de uma imagem digital.



Fonte: Barelli (2018)

2.3 Visão Computacional

Visão Computacional é a tecnologia que permite às máquinas enxergarem, ou seja, capacita os computadores a interpretarem as informações através de imagens fornecidas para eles. Essa ciência utiliza de técnicas de *hardware* e *software* para que os computadores possam ter conhecimento do ambiente ao seu redor, extraindo dados através de imagens e vídeos. A partir das informações processadas permite o computador reconhecer, manipular e identificar os objetos que compõem a cena (MILANO; HONORATO, 2014).

> A visão computacional é a construção de descrições explícitas e significativas de objetos físicos a partir de imagens. A compreensão da imagem é muito diferente do processamento da imagem, que estuda transformações de imagem a imagem e não a descrição explícita da construção. As descrições são um pré-requisito para o reconhecimento, manipulação e o pensamento sobre os objetos. (BALLARD; BROWN, 1982)

Os conceitos de visão computacional e reconhecimento de imagens são frequentemente associados, porém existe uma diferença entre eles. Reconhecimento de imagens é uma

parte integrante dos algoritmos de CV e funciona através de análise de *pixels* das imagens, a fim de identificar padrões conhecidos que podem ser, por exemplo, objetos já apresentados ao algoritmo. A visão computacional, entretanto possui um cenário abrangente com diversas outras funções de análise e processamento de imagens (ACADEMY, 2018).

Os algoritmos de visão computacional possuem algumas limitações, como a dificuldade em representar a visão humana de forma completa, pois a mesma se utiliza das funções de diferentes órgãos que não podem ser simulados por um computador. Além disso, cada sistema precisa ser analisado e escrito especificamente para uma função, elevando o custo necessário para o desenvolvimento dessas aplicações (KOT, 2021).

As vantagens de sistemas de CV também são inúmeras, como a possibilidade de automação de atividades, melhora na qualidade de produtos e outras aplicabilidades que serão destacadas nos próximos capítulos.

2.3.1 Histórico e Evolução

O conceito de visão computacional e as primeiras experiências envolvendo essa tecnologia foram feitas na década de 50, entretanto somente a cerca de 30 anos depois foram iniciados os primeiros trabalhos. Nessa época o objetivo da área era imitar a visão humana de modo a recriar esse sentido de forma completa, mas com o passar do tempo foi notada uma grande dificuldade em desenvolver esse modelo, principalmente pela falta de informações sobre como o cérebro humano interpreta as imagens (MILANO; HONORATO, 2014).

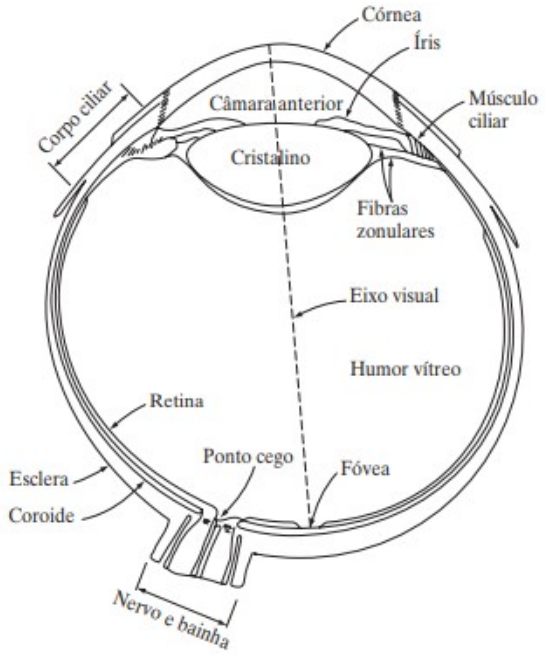
No ano de 1982, os autores Ballard e Brown escreveram a obra *Computer Vision*, citada neste trabalho anteriormente para conceituar visão computacional. Essa obra descreve diversos conhecimentos importantes para a área de visão computacional e foi fundamental para o desenvolvimento e evolução desses algoritmos (BALLARD; BROWN, 1982).

Desde os primeiros trabalhos houve uma grande evolução na área de processamento de imagem e visão computacional. Diversos algoritmos e bibliotecas podem ser usados para facilitar o desenvolvimento desses sistemas, de modo que não seja necessário um aprendizado robusto sobre os conceitos matemáticos empregados na execução desses processos (BARELLI, 2018).

2.3.2 Visão Humana

A visão humana é feita através de diversas partes que complementam a anatomia do olho humano. Entre as partes mencionadas, destacam-se as principais que são: a córnea, a esclera e a retina. A córnea é um tecido resistente que faz a cobertura da superfície anterior do olho, como o prolongamento desse tecido existe uma membrana opaca que reveste o restante do globo ocular que é chamada de esclera. A retina é a membrana mais interna do globo ocular, ela se estende por todo o interior do olho. A Figura 5 demonstra a anatomia do olho e onde estão as membranas citadas acima.

Figura 5 – Anatomia simplificada do olho humano



Fonte: Gonzalez e Woods (2009)

Basicamente pode-se descrever a visão humana através da luz de um objeto externo que forma uma imagem na retina. A visão então é obtida por receptores sensíveis à luz, chamados cones e bastonetes, que se situam ao longo da superfície da retina. Cada olho possui cerca de 6 a 7 milhões de cones que se localizam na fóvea, porção central da retina, e são sensíveis a aspectos das cores. O número de bastonetes é maior chegando a cerca de 75 a 150 milhões que são distribuídos pela superfície da retina, servem para dar uma imagem geral do campo de visão e não estão envolvidos na visualização das cores, visto que são sensíveis a baixos níveis de iluminação (GONZALEZ; WOODS, 2009).

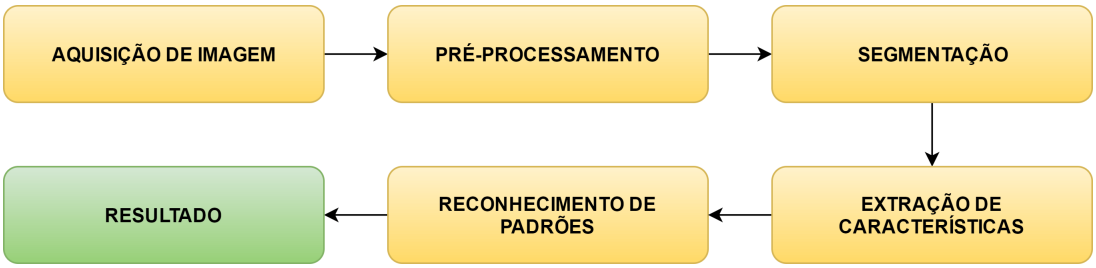
A visualização das imagens pelo olho humano é parecida com a aquisição das imagens digitais, mas no caso da visão humana o reconhecimento e a compreensão das imagens é feita através de neurônios aptos a realizar essa função de forma automática. Na área da visão computacional, o problema de reconhecimento de imagens é resolvido através de algoritmos de redes neurais treinadas que são capazes de extrair características através da matriz de *pixels* (KOT, 2021).

2.3.3 Etapas de Funcionamento

O funcionamento de algoritmos de visão computacional não possui um padrão de implementação, o mesmo deve ser analisado conforme a solução que irá ser implementada (MILANO; HONORATO, 2014). Todavia, a maioria dos sistemas possuem um fluxo de procedimento em comum que pode ser separado em etapas, como mostra a Figura 6.

A descrição das etapas do fluxo de funcionamento serão descritas a seguir, apresen-

Figura 6 – Etapas de funcionamento de um algoritmo de visão computacional



Fonte: Barelli (2018)

tando ao final da execução o resultado do sistema desenvolvido com base na implementação proposta e da imagem utilizada.

1. **Aquisição de imagem:** o processo de aquisição de imagem foi detalhado no capítulo de formação de uma imagem;
2. **Pré-processamento:** na etapa de pré-processamento são usados os métodos para a melhoria da imagem a fim de corrigir iluminação, nitidez, contrastes e outras características da imagem;
3. **Segmentação:** o processamento de uma imagem pode ser custoso, seja por custo de memória, processamento ou armazenamento. Por esse motivo, a etapa de segmentação tem o objetivo de destacar as regiões importantes da foto, onde se encontram os objetos de interesse, gerando a diminuição da área da imagem e consequentemente dos custos (RUDEK; COELHO; JR, 2001);
4. **Extração das características:** nesta etapa são extraídas características matemáticas importantes da imagem, como bordas, formatos e movimentos (MILANO; HONORATO, 2014);
5. **Reconhecimento de padrões:** ao final do funcionamento com todos os dados sobre a imagem coletados é feita a identificação de padrões que possuem semelhança entre si ou com uma base de dados. Existem diversas ferramentas que realizam processos matemáticos e computacionais, com o objetivo de agrupar objetos semelhantes por meio do reconhecimento de padrões (RUDEK; COELHO; JR, 2001).

2.3.4 Aplicabilidade

Muitas áreas podem ser beneficiadas com sistemas que envolvem visão computacional. Essa tecnologia pode ser usada para automatizar e facilitar diversas funções. Como exemplificam Danilo de Milano e Luciano Barrozo Honorato (MILANO; HONORATO, 2014), é possível citar diversas aplicabilidades, entre elas, pode-se destacar:

- **Reconhecimento facial:** essa tecnologia pode ser empregada para encontrar uma pessoa em câmeras de segurança ou para o desbloqueio de um dispositivo celular, além da possibilidade de extrair informações sobre o estado emocional do indivíduo através da análise

da imagem;

- **Meio ambiente:** pode ser usada na área ambiental para detectar mudanças climáticas e também crimes ambientais como queimadas e desmatamento em tempo real através de imagens de satélites;
- **Medicina:** existem sistemas conhecidos como Sistemas Computacionais de Apoio ao Diagnóstico (CAD) que servem para dar um apoio ao médico na hora de prescrever um diagnóstico, esses sistemas se utilizam de imagens de exames para detectar fraturas, lesões pulmonares e podem ser úteis também em exames de mamografia;
- **Indústria:** é possível automatizar o controle de qualidade de uma produção e em áreas mais específicas pode ser útil na caracterização e classificação de minérios.

2.4 *Deep Learning*

O funcionamento do cérebro humano sempre foi considerado algo complexo, pela capacidade de armazenar e processar grande quantidade de informação. Movidos por esse conceito, diversos pesquisadores buscaram simular esse procedimento para enfim obter o processo de aprendizagem por experiência e de reconhecimento de padrões, através de neurônios programados artificialmente. Com esses estudos foram criadas as Redes Neurais que simulam o comportamento de uma rede de neurônios, capazes de interagir entre si e trocar informações (DOMINGOS, 2017).

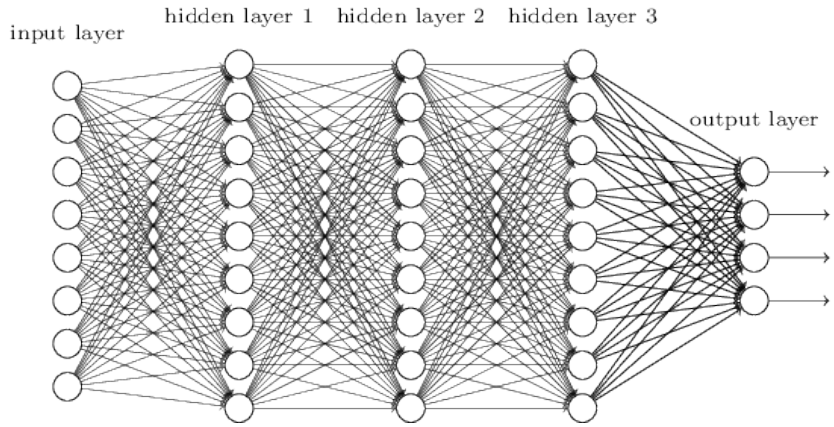
Deep Learning, ou Aprendizado profundo, é um ramo do aprendizado de máquina que consiste em treinar um modelo que busca aprender padrões através dos dados de entrada. São utilizadas grandes massas de dados que passam por um processamento não linear de camadas de redes neurais. Portanto, permite às máquinas aprenderem conforme entram em contato com novos dados, não sendo necessária a intervenção humana. A Figura 7 demonstra o funcionamento dos algoritmos de deep learning, sendo *input layer* a entrada dos dados, *hidden layers* o processamento dos dados através da interação das redes neurais e por fim, a saída representada por *output layer* (HENRIQUE, 2018).

Entre as várias usabilidades dos algoritmos de *deep learning* é possível citar: os carros autônomos, previsões no mercado de ações, reconhecimento facial, reconhecimento de objetos em uma imagem, sistema de recomendação como de roupas ou em catálogo de filmes e séries, entre vários outros que podem usar essa tecnologia para reconhecer padrões ou fazer a aprendizagem através de experiência (DOMINGOS, 2017).

2.5 Sistemas Embarcados

Sistemas embarcados são considerados os sistemas completos e independentes dedicados a realizar uma determinada tarefa. Podem ser vistos como uma caixa preta onde as entradas são fornecidas ao sistema, processadas e por fim geram uma saída que depende do propósito do sistema. Portanto o usuário final não terá acesso ao programa que foi embutido no dispositivo,

Figura 7 – Funcionamento de uma rede neural



Fonte: Henrique (2018)

mas pode interagir com ele através de uma interface que é definida através de botões, sensores, teclados, entre outros.

> Um sistema embarcado é um sistema baseado em microprocessador que é construído para controlar uma função ou uma gama de funções e não é projetado para ser programado pelo usuário final da mesma forma que um PC é. (HEATH, 2002)

Entre as principais características dos sistemas embarcados é possível destacar: o baixo custo, menor consumo de energia, tamanho reduzido, confiabilidade e a segurança. Todos estes fatores propiciam uma vasta utilização desses dispositivos em projetos de diversas áreas e para diferentes utilidades. Outros aspectos são importantes para o bom funcionamento desses dispositivos como: confiabilidade, manutenibilidade e disponibilidade (CUNHA, 2007).

Os sistemas embarcados são compostos por uma unidade de processamento, um circuito integrado, capaz de fazer o processamento das informações de entrada através do *software* desenvolvido para aquele dispositivo. Além do processador, o *hardware* pode conter diversos tipos de periféricos como sensores, câmeras, acelerômetros, entre outros (BARROS; CAVALCANTE, 2010).

Existem diversas classificações para os sistemas embarcados que definem o funcionamento ou até a fonte de energia do aparelho. Entre as principais podemos citar:

- **Propósito geral:** aplicações mais parecidas com computadores de mesa, onde existe grande interação entre o usuário e o sistema, seja por meio de/ou monitores, teclados e controles;
- **Reativo:** o funcionamento é por meio de respostas a eventos externos, sejam eles periódicos ou assíncronos. Existe a necessidade de uma entrada para realizar o funcionamento, após essa entrada a saída deve ser gerada logo em seguida;
- **Sistema em tempo real:** sistemas cujo tempo de resposta é extremamente necessário. São divididos em *hard real-time*, onde o tempo de resposta é rígido e pode haver consequências caso o mesmo não seja respeitado, e *soft real-time* que são executados em um

período de tempo e não possuem consequências graves caso o limite de tempo não for cumprido;

- **Operados a bateria:** sistemas que possuem uma bateria e utilizam a energia da mesma para o funcionamento;
- **Consumo fixo:** sistemas que só funcionam se estiverem conectados a uma fonte de energia, como uma tomada, por exemplo (MARWEDEL, 2021).

Esses dispositivos estão muito presentes no cotidiano podendo ser encontrados em diferentes áreas e utilitários. É possível fazer uso dos mesmos no controle industrial, na área automotiva, médica e automação residencial. O sistema de freios antitravamento (ABS) e sensores de estacionamento são exemplos de uso no setor automobilístico, além de diversos eletrônicos como geladeiras, máquinas de lavar e *videogames* que utilizam sistemas embarcados (BARROS; CAVALCANTE, 2010).

3 FERRAMENTAS DE DESENVOLVIMENTO

Para além do embasamento teórico é necessário conhecer os instrumentos utilizados no desenvolvimento do trabalho, por esse motivo o presente capítulo busca detalhar o ferramental previsto na construção do trabalho. Serão detalhados os elementos utilizados na construção do sistema embarcado, sendo o Raspberry Pi e o módulo de câmera na parte física, o Raspberry Pi OS como sistema operacional e o *software* mpjg-streamer empregado de modo que seja o servidor *streamer* das imagens da câmera. Para o desenvolvimento do sistema será detalhada a linguagem Python e suas bibliotecas, bem como o *framework* TensorFlow para o treinamento do modelo de detecção de objetos.

3.1 Raspberry Pi

Raspberry Pi Foundation é uma instituição que tem o objetivo de democratizar o acesso ao poder computacional para pessoas de todo o mundo. Possuem projetos para o aprendizado de programação e técnicas de criação digital para a juventude, por meio de capacitação a escolas para que possam ofertar esse conhecimento aos alunos, bem como parceiras em eventos e clubes com organizações juvenis. Tornam o poder computacional acessível a todos através do fornecimento do seu produto, uma placa de computador única de baixo custo e alto desempenho denominado Raspberry Pi (RASPBERRYPI, 2021a).

O Raspberry Pi é um projeto de *hardware* aberto proposto inicialmente para ser uma solução de um computador popular. Sua usabilidade pode ser feita através de dispositivos de entrada e saída utilizando sistema operacional com interface gráfica. Considerado um minicomputador o Raspberry Pi é do tamanho de um cartão de crédito, porém mesmo sendo pequeno possui diversos componentes que permitem sua utilização em variados projetos. Com o passar do tempo e o sucesso desse projeto, foi possibilitada sua utilização em outros propósitos como no desenvolvimento de sistemas embarcados (OLIVEIRA, 2017).

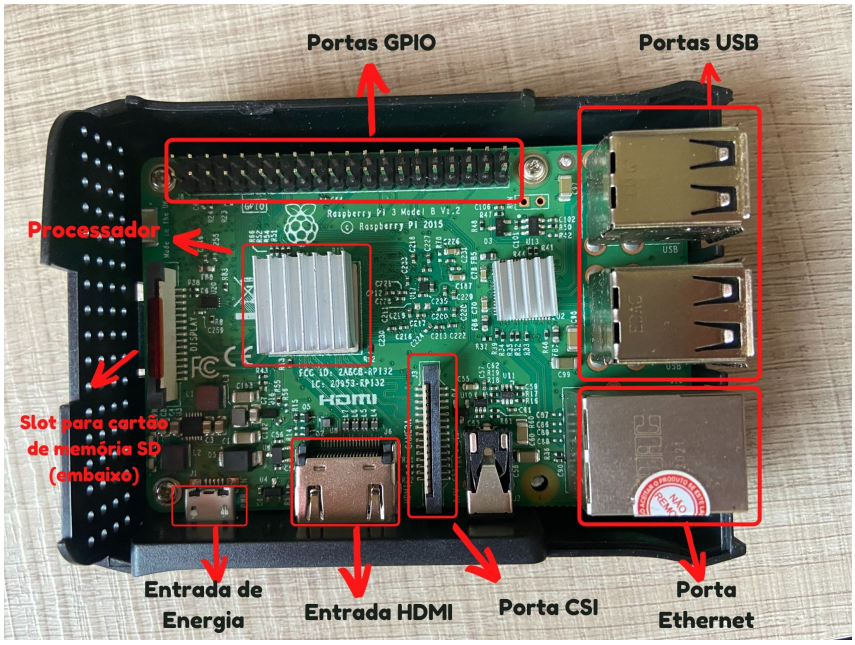
Existem diversos sistemas embarcados que utilizam outra ferramenta bastante conhecida na sua arquitetura, o Arduino. Esse dispositivo é comumente usado em sistemas que buscam um maior controle na parte do *hardware*. Entretanto este trabalho optou por utilizar o Raspberry Pi, pois entre as principais vantagens destacam-se:

- Sistema operacional que facilita o trabalho e configuração do projeto através de uma interface gráfica;
- Conectividade *bluetooth* e WI-FI integrada, sem precisar da conexão de componentes extras;
- Processador mais potente, favorece seu uso em funções mais complexas que envolvem processamento de *softwares* (FRUTUOSO et al.,).

3.1.1 Versão usada com arquitetura

O *hardware* utilizado para o trabalho é o Raspberry Pi 3 Model B que possui diversas funções e utilitários disponíveis para facilitar a execução e desenvolvimento. A Figura 8 mostra a arquitetura desse sistema situando os principais componentes do mesmo.

Figura 8 – Arquitetura do *hardware* Raspberry Pi 3 Model B



Fonte: Autor

- **Processador:** é o coração do aparelho e responsável por executar o sistema operacional e os *softwares* aplicativos;
- **Porta USB:** necessária para conectar os periféricos no aparelho;
- **Porta *Ethernet*:** mesmo possuindo conectividade *wireless*, esse elemento é importante caso for necessário conectar um cabo de rede para obter uma conexão mais estável;
- **Conector HDMI:** fornece saída de áudio e vídeo;
- **Entrada de energia:** utilizado como uma entrada de carregamento com intuito de fornecer energia para o aparelho;
- **Porta CSI:** utilizada para fazer a conexão da câmera;
- **Portas GPIO:** conjunto de pinos responsáveis por fazer a comunicação de entrada e saída digital;
- ***Slot* para cartão de memória SD:** necessário para comportar um cartão de memória SD que tem a função de armazenar o sistema operacional e os dados necessários (RICHARDSON; WALLACE, 2013).

3.2 Raspberry Pi OS

Raspberry Pi OS é um sistema operacional gratuito que foi desenvolvido com base no Debian e otimizado para o Raspberry Pi, por isso é considerado como o sistema operacional padrão desse tipo de *hardware*. É também considerado completo por possuir milhares de pacotes que facilitam a usabilidade e o desenvolvimento nesse sistema.

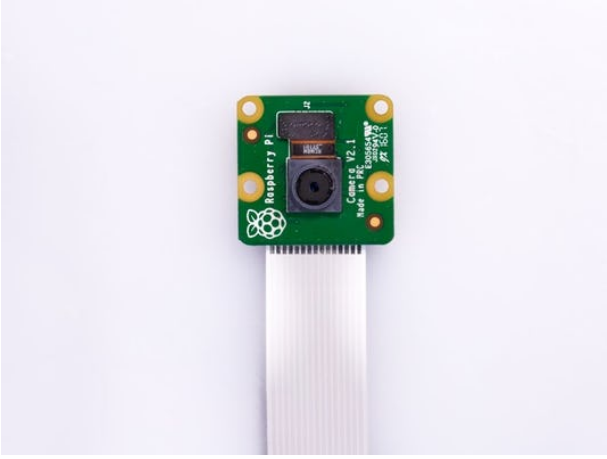
Existem diversos sistemas operacionais que podem ser usados no Raspberry Pi, por exemplo RISC OS, Arch Linux e Pidora. Porém o Raspberry Pi OS é o mais recomendado por ter sido desenvolvido pensando na otimização para o *hardware*. Neste trabalho, a versão usada é a *kernel version* 5.10, todavia este *software* está em desenvolvimento ativo o que propicia diversas atualizações (RASPBERRYPI, 2021c).

3.3 Módulo de Câmera Raspberry Pi v2

A Raspberry Pi Foundation possui um módulo de câmera original para utilizar em aplicações que necessitam de imagens ou vídeos em alta definição. O mesmo pode ser conectado em todos os modelos de Raspberry Pi para complementar diferentes tipos de projetos, como aplicações de segurança doméstica. Esse dispositivo é totalmente programável e por isso existem diversas bibliotecas que podem ser usadas para ajudar no desempenho da câmera, seja aplicando filtros na imagem ou disponibilizando os registros da câmera para alguma aplicação.

O Módulo de Câmera v2 possui um sensor Sony IMX219 de 8MP e pode ser conectado no Raspberry Pi através da porta CSI por meio de um cabo de fita. Suporta diversos modos de vídeos, além de fazer captura de fotos. É uma evolução do módulo de câmera original que possuía 5MP. Esse módulo pode ser acessado de diversas formas, seja por aplicações *web* ou para utilização das imagens em algoritmos (RASPBERRYPI, 2021b). A Figura 9 apresenta o Módulo de Câmera Raspberry Pi v2.

Figura 9 – Módulo de câmera Raspberry Pi v2



Fonte: RaspberryPi (2021b)

É possível utilizar outros tipos de câmeras que são conectadas através da porta USB do aparelho, porém essa abordagem captura imagens com menos quadros por segundo e gera um alto custo para a CPU. Por isso, este trabalho optou por utilizar o módulo de câmera original pela facilidade na utilização, acesso e conexão do equipamento. O modo de câmera também tem a possibilidade de conectar o dispositivo diretamente na GPU, gerando arquivos de vídeo com mais qualidade e mais quadros por segundo. Com o bônus de utilizar menos a CPU, que ficará disponível para rodar as outras funções do sistema (RASPBERRYPI, 2021b).

As câmeras IP também são uma possibilidade de utilização na obtenção das imagens, afinal esse tipo de câmera disponibiliza as imagens em tempo real através de uma rede LAN. Este utilitário poderia substituir o sistema embarcado, visto que exerceria a mesma função neste trabalho. Para a troca dos equipamentos seria necessário configurar o novo dispositivo na aplicação *web*, de modo que ele disponibilizasse as imagens corretamente para os processos que necessitam destas informações.

3.4 Linguagem de programação Python

A linguagem de programação Python é uma linguagem de alto nível interpretada, que foi criada no ano de 1990, mesmo sendo antiga é considerada uma das linguagens mais utilizadas. Sua semântica dinâmica e de fácil usabilidade propicia fácil manutenção nos códigos, além de possuir compatibilidade com a maior parte dos sistemas operacionais. Por esse motivo, essa linguagem está sendo cada vez mais utilizada, sendo considerada uma ótima opção para quem está começando a programar.

Sua utilização neste trabalho foi de grande importância visto que essa linguagem pode ser utilizada em diversos tipos de algoritmos, como em programação *web, back-end* e também no desenvolvimento de soluções em diversas áreas como ciência de dados, inteligência artificial e na construção de algoritmos de visão computacional. Esses aspectos são possíveis, pois é uma linguagem que possui bibliotecas diversificadas, propiciando a facilidade de sua aplicação em diversos âmbitos (LUBANOVIC, 2014).

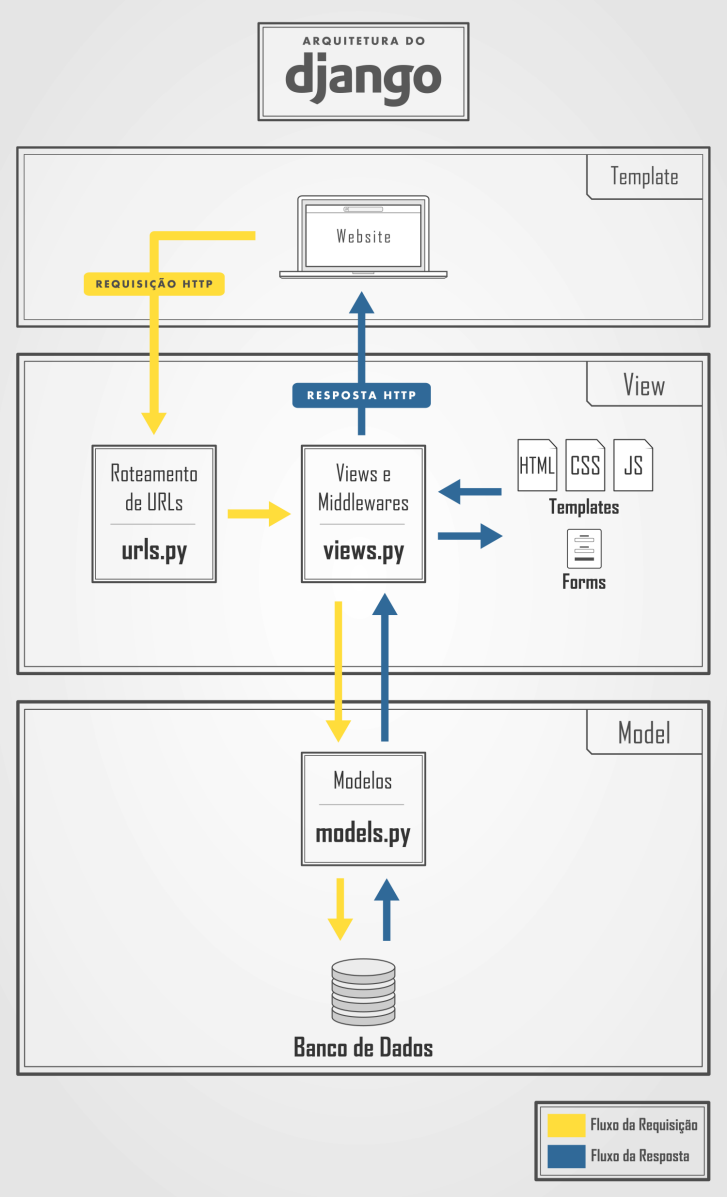
3.4.1 Django

Django é um *framework* para a linguagem Python que tem uma estrutura de alto nível com o objetivo de criar aplicações com desenvolvimento rápido, um design limpo e pragmático. Foi construído por desenvolvedores experientes e é responsável por cuidar da maior parte da construção de um sistema *web*. Seu uso é gratuito e de código aberto e seu desenvolvimento e instalação são feitos de forma rápida e fácil.

Dentre as principais características positivas é possível citar: a rapidez no desenvolvimento de projetos, as diversas funcionalidades prontas que o *framework* possui, o alto nível de segurança proveniente da autenticação dos sistemas e do controle a erros comuns em programas *web* e também sua escalabilidade. O Django utiliza uma arquitetura denominada de *Model View*

Template, onde cada camada é responsável por uma parte do funcionamento (FOUNDATION, 2021). A figura 10 demonstra a arquitetura de funcionamento do *framework* Django.

Figura 10 – Arquitetura do *framework* Django



Fonte: Academy (2021)

- ***Model*:** é a camada que se relaciona com o banco de dados. Possui os objetos que definem a estrutura dos dados e é capaz de gerenciar de forma prática as informações da aplicação;
- ***View*:** é a camada responsável por receber as solicitações HTTP e faz o retorno dessas chamadas com conteúdos que podem ser dados ou uma página HTML;
- ***Template*:** é a camada que renderiza os conteúdos estáticos que podem ser conteúdos de frontend: HTML e CSS, por exemplo (DOCS, 2021).

3.4.2 OpenCV

A biblioteca OpenCV é um recurso de código aberto que foi construído para facilitar a estruturação e desenvolvimento de programas baseados em visão computacional e aprendizado de máquina. Possui mais de 2500 algoritmos otimizados que possibilitam o processamento e melhoramento de imagens além do reconhecimento de faces, identificação de objetos e outras técnicas.

Seu código fonte é escrito na linguagem de programação C++ e está disponível em diversas outras linguagens como por exemplo Python e Java. Além de ser suportado por diversos sistemas operacionais como Windows e Linux (OPENCV, 2021).

3.5 TensorFlow

TensorFlow é um *framework* desenvolvido para facilitar a computação numérica de dados. Foi projetado com o propósito de implementar algoritmos de aprendizado de máquina como por exemplo, *deep learning*. Desenvolvido pela Google Brain Team, possui uma diversa gama de utilitários que se distribuem entre algoritmos de treinamento de modelos para classificação ou detecção de objetos, entre outros processos. Além de possuir portabilidade com inúmeros dispositivos e sistemas operacionais, é possível desenvolver um grande projeto unindo um *cluster* de computadores e até algo mais simples para ser utilizado em dispositivos móveis (HOPE; RESHEFF; LIEDER, 2017).

O principal intuito do TensorFlow é facilitar o desenvolvimento de sistemas que utilizam criação ou implantação de modelos de *Machine Learning*, por esse motivo existem fatores que facilitam e corroboram com a sua ampla usabilidade:

- **Criação fácil de modelos:** oferece diferentes níveis de abstração possibilitando a escolha da ferramenta que mais se encaixa na sua necessidade;
- **Produção robusta de *Machine Learning* em qualquer lugar:** a portabilidade é um destaque desse *framework*, possibilita a implantação independentemente de linguagem ou plataforma utilizada;
- **Experimentos poderosos para pesquisa:** possibilita a criação e treinamento de modelos agéis e flexíveis, sem sacrificar desempenho ou velocidade da aplicação.

3.6 mpjg-streamer

O utilitário mjpg-streamer, de código aberto e disponível no GitHub, é um aplicativo utilizado por meio de linha de comando que possibilita a cópia de imagens JPG através de um ou mais periféricos de entrada. É capaz de transmitir esses arquivos em redes, cujo protocolo é TCP/IP e esteja conectado em uma *webcam* ou uma câmera. Foi desenvolvido para a utilização em sistemas embarcados que possuem pouco poder de processamento e por esse motivo possui

suporte para diversos dispositivos de entrada, entre eles o Raspberry Pi e o seu módulo de câmera (LIAM, 2021).

4 IMPLEMENTAÇÃO DO SISTEMA

O objetivo do desenvolvimento é utilizar imagens providas pela câmera do sistema embarcado para enriquecer o *dataset* de treinamento com fotos dos objetos, e também utilizar essas informações para fazer a detecção e contagem dos itens. Para facilitar a utilização, foi projetada uma interface *web* que possibilita o fácil manuseio, tanto para o cadastro das informações do sistema, quanto a execução do modelo de detecção. O treinamento do modelo exige tempo e poder computacional, além de ser uma ação que será executada poucas vezes. Por esse motivo, ele foi deixado fora da aplicação e deve ser executado manualmente. Na organização dos dados de objetos e imagens foi utilizado um banco de dados e diversas pastas que contém as fotos de cada objeto separadamente. Além do banco de dados PostgreSQL, outros recursos foram utilizados e serão destacados neste capítulo.

A seguir são demonstrados o diagrama de arquitetura do sistema, bem como os diagramas UML utilizados no desenvolvimento do trabalho. Serão detalhados também o desenvolvimento dos componentes de *hardware* e *software*.

4.1 Recursos utilizados para o desenvolvimento

Diversos aplicativos e ferramentas, além daquelas já destacadas, foram utilizados durante o desenvolvimento do trbalho. Entre os principais se destacam:

- **PyCharm:** ambiente desenvolvimento utilizado na programação da linguagem Python;
- **PostgreSQL:** é um sistema de gerenciamento de banco de dados relacional;
- **Git:** é um sistema de controle de versões utilizado para gerenciar o código durante seu desenvolvimento e alteração;
- **GitHub:** é uma plataforma de hospedagem de código utilizado para gerenciar versões e armazenar o repositório de códigos;
- **Astah Community:** ferramenta utilizada para o desenvolvimento dos diagramas UML;
- **Overleaf:** é um editor da linguagem LaTeX utilizado para escrever e editar trabalhos e arquivos.

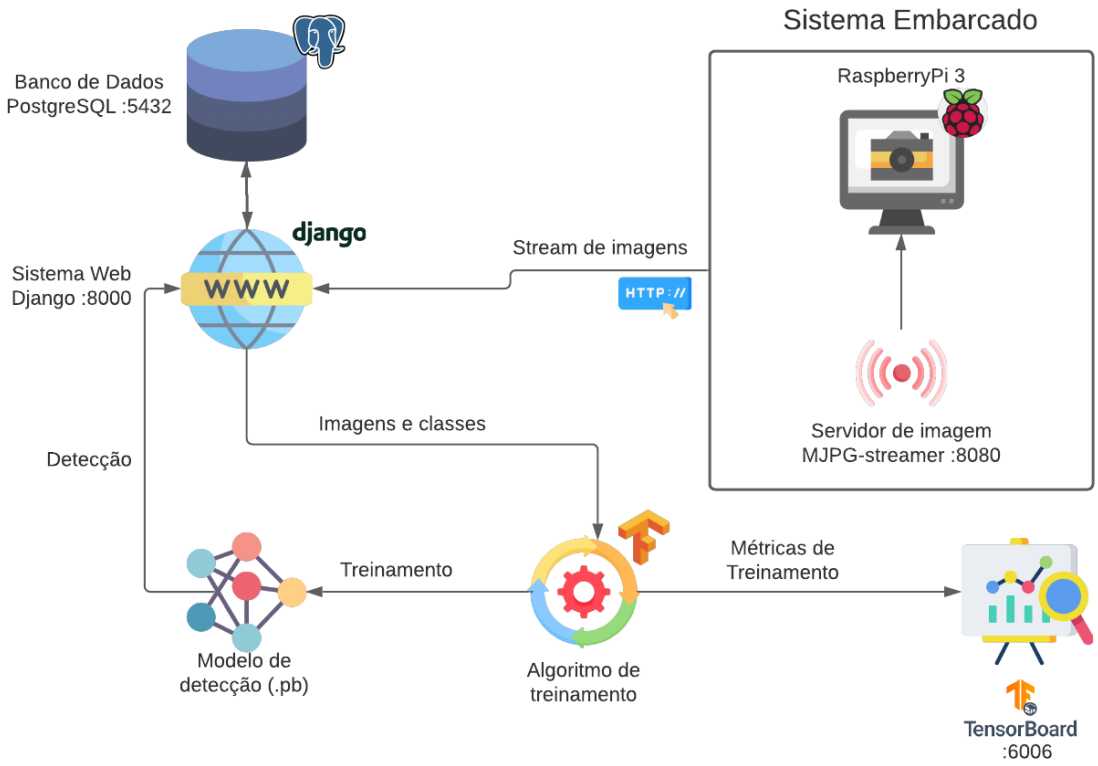
4.2 Diagramas

No desenvolvimento de sistemas que envolvem diversos componentes e lógicas, é imprescindível a utilização de diagramas que possam detalhar o seu funcionamento e construção. Por esse motivo, o presente capítulo exemplificará os principais diagramas utilizados na execução do trabalho. Num primeiro momento é apresentada a arquitetura do sistema e após são demonstrados os diagramas UML, muito utilizados na elaboração da estrutura de projetos de *software*.

4.2.1 Arquitetura do Sistema

A arquitetura de um sistema provê a lógica de interação entre os componentes, além de estabelecer como será feita a comunicação e que componentes estão envolvidos em cada processo. A Figura 11 mostra o diagrama da arquitetura do sistema com a comunicação entre os componentes, a porta que cada servidor está utilizando e também as tecnologias de cada item.

Figura 11 – Arquitetura do Sistema



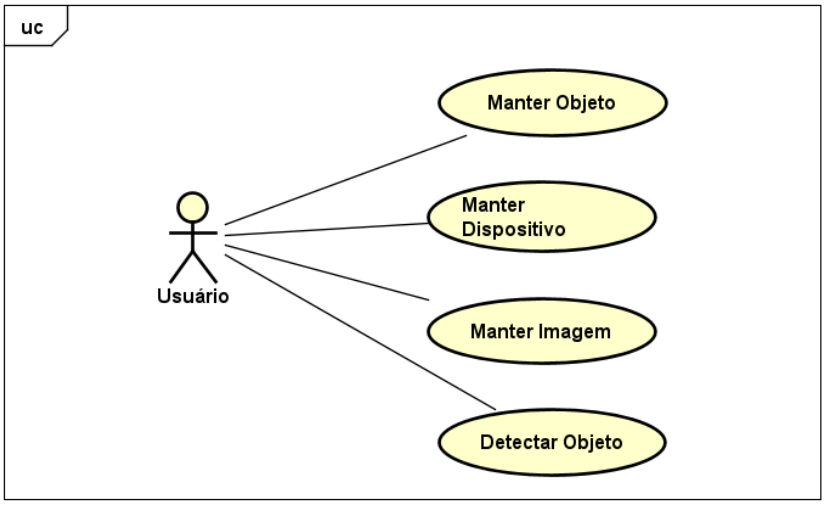
Fonte: Autor

4.2.2 Diagrama de Casos de Uso

Os diagramas de casos de uso facilitam a compreensão do comportamento do sistema através de uma linguagem simples e coesa. Seu objetivo é apresentar uma visão geral das principais funcionalidades que o programa oferta para os usuários, além de possibilitar a fácil identificação dos requisitos do sistema. Este diagrama possui um ator, que pode ser um usuário, e as funções que esse ator pode realizar (GUEDES, 2018).

Na Figura 12 é demonstrado o diagrama de casos de uso. O usuário interage com os casos de uso que consistem em: manter objeto (descrito no Quadro 1), manter imagem (descrito no Quadro 2), manter dispositivo (descrito no Quadro 3) e detectar objeto (descrito no Quadro 4).

Figura 12 – Diagrama de Casos de Uso



Fonte: Autor

Quadro 1 – Descrição de Caso de Uso - Manter Objeto

Caso de Uso:	Manter Objeto
Ator(es):	Usuário
Pré-condições:	-
Pós-condições:	Objeto pode ser cadastrado, alterado ou excluído do sistema

	Ator		**Sistema**	
1	Ativa a interface de exibição dos objetos			
		2	Exibe a tela com os objetos cadastrados	
3	Clica no botão "Novo Objeto"			A1
		4	Exibe o formulário de cadastro de novos objetos	
5	Digita os campos solicitados e clica no botão "Adicionar"			A1
		6	Grava os dados do Objeto	
		7	Mostra a tela com a informação de todos os objetos	
		8	Encerra caso de uso	
9	Escolha o objeto a ser alterado			
		10	Exibe as informações do objeto	
11	Clica no botão "Atualizar Objeto"			A1
		12	Exibe o formulário de atualização	
13	Modifica as informações desejadas e clica no botão "Salvar Alterações"			A1
		14	Grava os dados alterados	
		15	Encerra caso de uso	
16	Escolha o objeto a ser excluído			
		17	Exibe as informações do objeto	
18	Clica no botão "Excluir objeto"			A1
		19	Grava a exclusão do objeto	
		20	Encerra caso de uso	

A1	Cancela caso de uso

Fonte: Autor

Quadro 2 – Descrição de Caso de Uso - Manter Imagem

Caso de Uso:	Manter Imagem
Ator(es):	Usuário
Pré-condições:	-
Pós-condições:	Imagem pode ser cadastrada ou excluída do sistema

	Ator		**Sistema**	
1	Ativa a interface de objetos			
		2	Mostra a página com todos os objetos	
3	Clica no objeto que deseja adicionar a imagem			A1
		4	Mostra as informações do objeto	
5	Clica no botão "Adicionar Imagem"			A1
		6	Mostra as informações da adição de imagem	
7	Escolhe entre o campo de inserir imagem ou adicionar imagem do sistema embarcado			A1
8	Clica no botão "Adicionar"			
		9	Grava as informações da imagem	
		10	Mostra o formulário de adição de imagem	
		11	Encerra o caso de uso	
12	Clica no objeto			
		13	Mostra as informações do objeto	
14	Clica no ícone de lixeira acima da imagem desejada			
15	Confirma a exclusão			A1
		16	Grava a exclusão da imagem	
		17	Encerra o caso de uso	

A1	Cancela caso de uso

Fonte: Autor

Quadro 3 – Descrição de Caso de Uso - Manter Dispositivo

Caso de Uso:	Manter Dispositivo
Ator(es):	Usuário
Pré-condições:	-
Pós-condições:	Dispositivo pode ser cadastrado ou excluído do sistema

	Ator		**Sistema**	
1	Ativa a interface da página inicial			
		2	Mostra a página inicial	
3	Clica no ícone de adição no box de dispositivos			A1
		4	Mostra o formulário de cadastro de dispositivo	
5	Digita os campos solicitados			A1
6	Clica no botão “Adicionar”			
		7	Verifica os dados digitados	
		8	Grava os dados do Dispositivo	
		9	Mostra a página inicial	
		10	Encerra caso de uso	
11	Clica no ícone de lixeira ao lado do dispositivo desejado			
12	Confirma a exclusão			A1
		13	Grava a exclusão do dispositivo	
		14	Encerra caso de uso	

A1	Cancela caso de uso

Fonte: Autor

Quadro 4 – Descrição de Caso de Uso - Detectar Objeto

Caso de Uso:	Detectar Objeto
Ator(es):	Usuário
Pré-condições:	Treinamento do algoritmo não deve estar rodando
Pós-condições:	Imagem com os objetos destacados e informações mostrada na tela

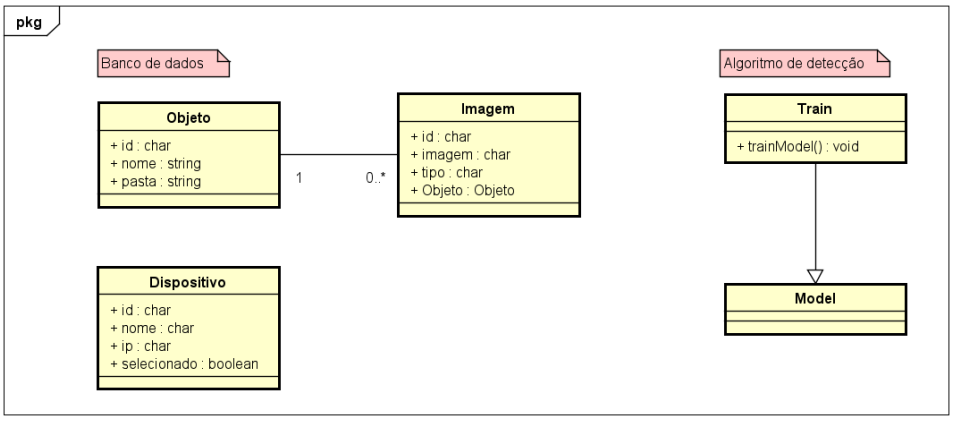
	Ator		**Sistema**	
1	Ativa a interface de detecção dos objetos			
		2	Mostra a tela com as informações de detecção de objetos	
3	Clica no botão “Detectar Objetos”			
		4	Exibe a imagem com os objetos destacados	
		5	Exibe uma tabela com informações dos objetos	
		6	Encerra caso de uso	

Fonte: Autor

4.2.3 Diagrama de Classes

O diagrama de classes é um dos mais utilizados no desenvolvimento do trabalho. Possibilita a compreensão das classes, métodos e atributos que compõem o sistema, bem como as relações que serão estabelecidas entre os mesmos (GUEDES, 2018). Na Figura 13 é demonstrado o diagrama de classes.

Figura 13 – Diagrama de Classe

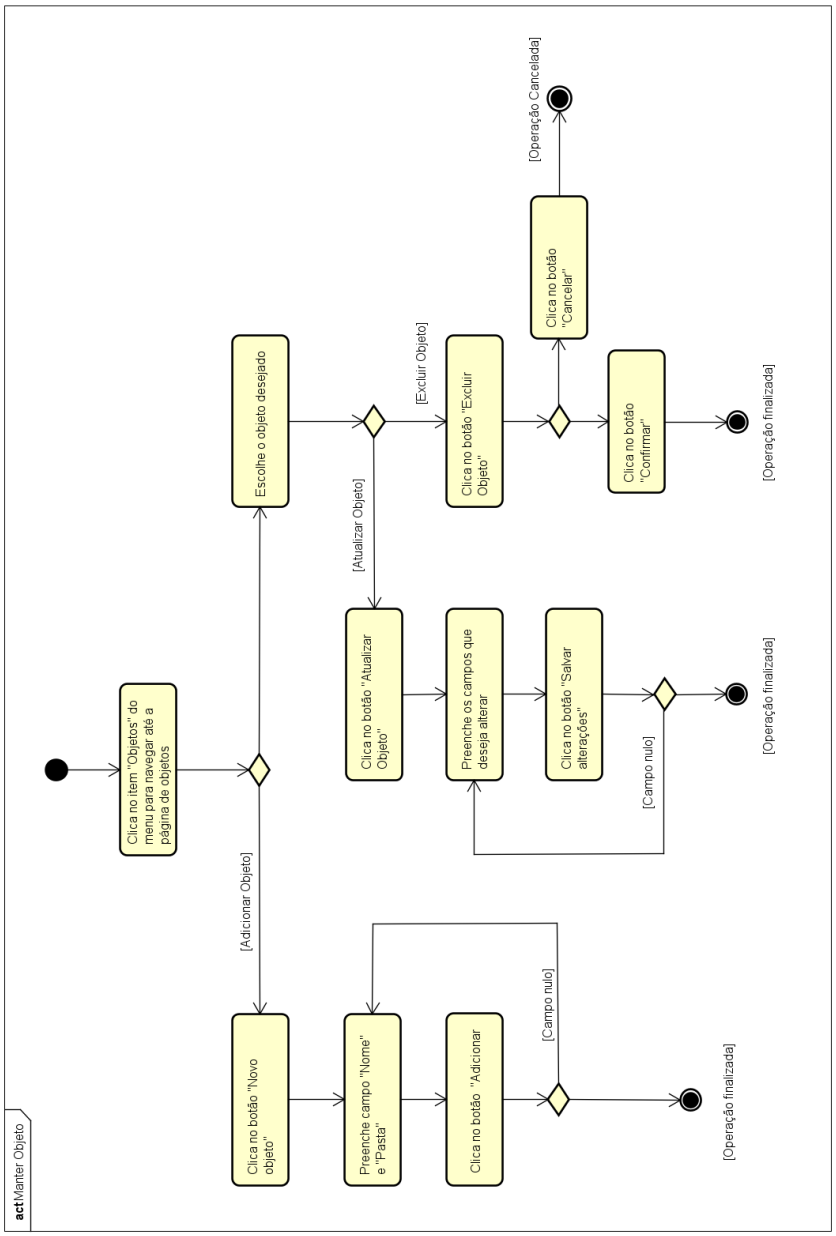


Fonte: Autor

4.2.4 Diagramas de Atividade

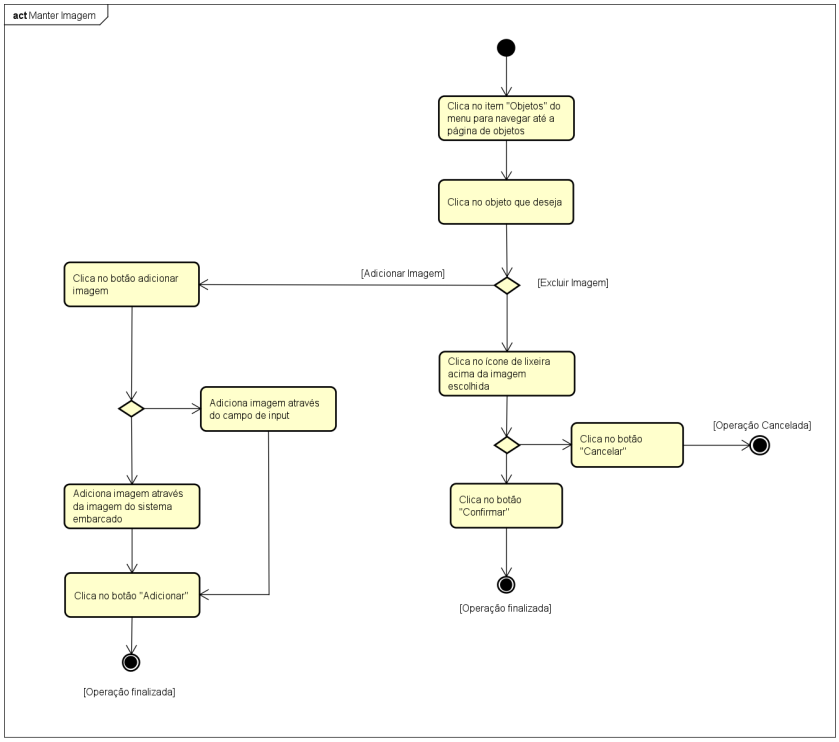
Os diagramas de atividades são utilizados para definir uma sequência de ações e condições na execução de um processo. É um dos diagramas mais detalhistas, visto que se aprofunda na definição do passo-a-passo de uma atividade. As Figuras a seguir definem os diagramas de atividade dos seguintes processos: manter objeto (descrito na Figura 14), manter imagem (descrito na Figura 15), manter dispositivo (descrito na Figura 16) e detectar objeto (descrito na Figura 17).

Figura 14 – Diagrama de Atividade - Manter Objeto



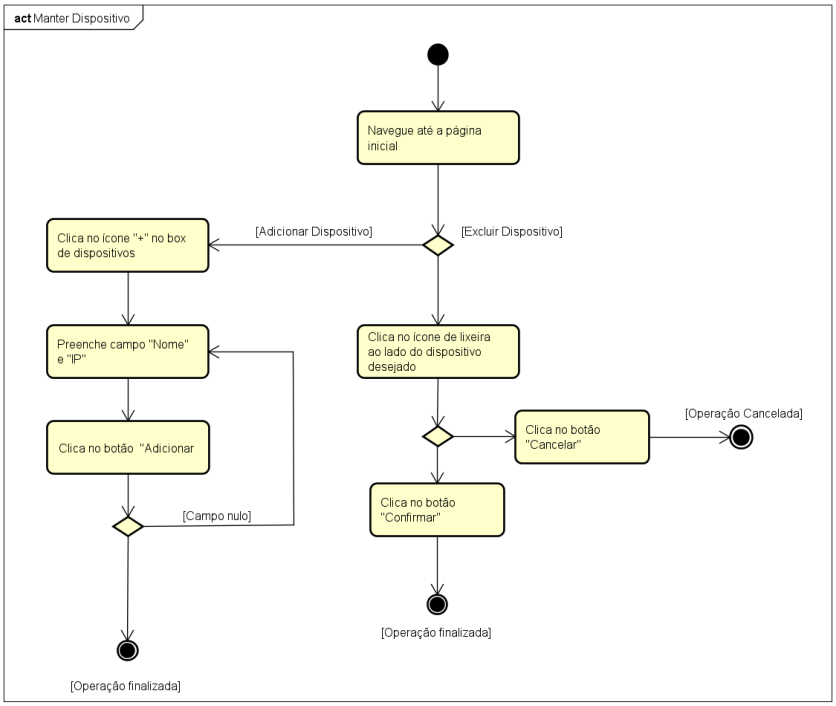
Fonte: Autor

Figura 15 – Diagrama de Atividade - Manter Imagem



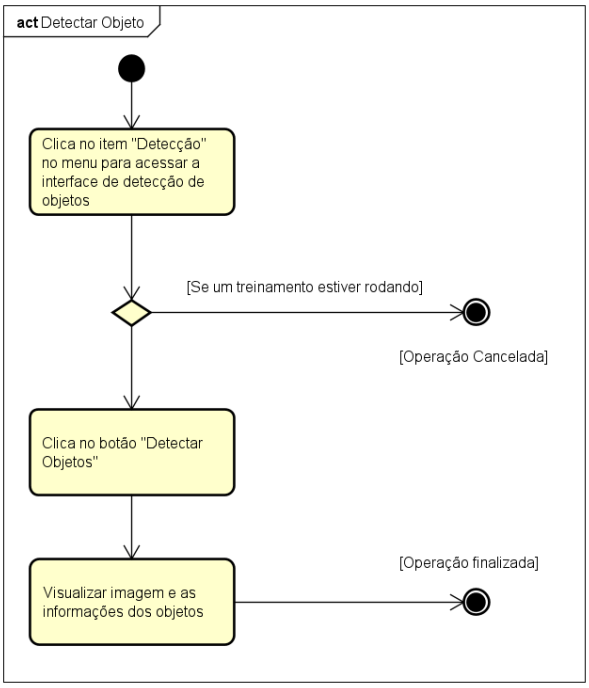
Fonte: Autor

Figura 16 – Diagrama de Atividade - Manter Dispositivo



Fonte: Autor

Figura 17 – Diagrama de Atividade - Detectar Objeto

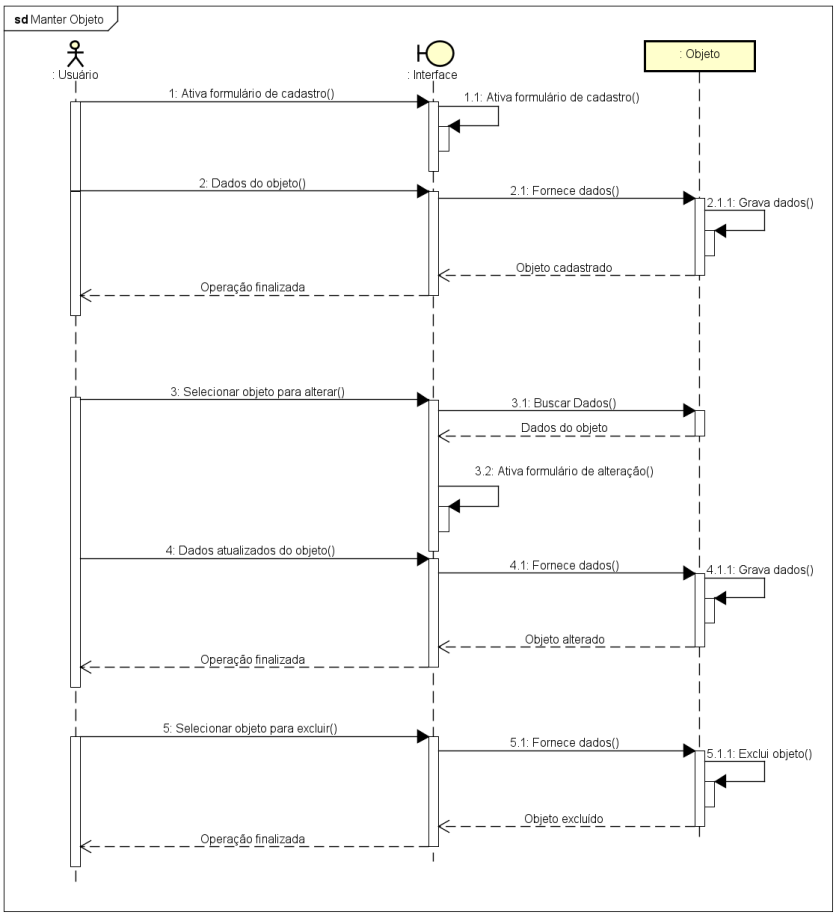


Fonte: Autor

4.2.5 Diagramas de Sequência

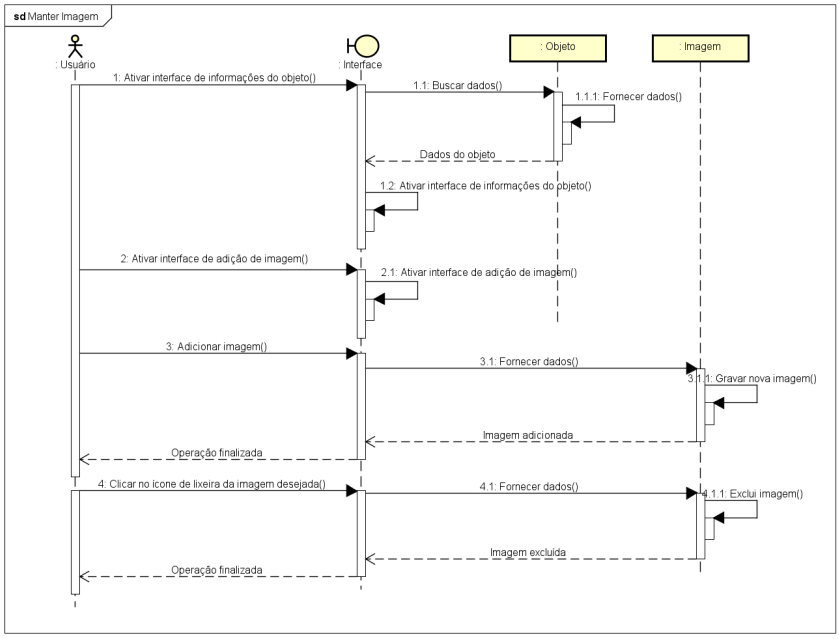
O diagrama de sequência é um diagrama comportamental que busca definir a sequência de ações que vão acontecer em um determinado processo. Documenta quais mensagens e em que ordem elas devem ser disparadas, bem como quais os elementos envolvidos nessas ações. Baseia-se no diagrama de caso de uso, geralmente cada caso de uso possui um diagrama de sequência a fim documentar com mais especificidade as suas informações. Os diagramas a seguir representam os seguintes processos: manter objeto (descrito na Figura 18), manter imagem (descrito na Figura 19), manter dispositivo (descrito na Figura 20) e detectar objeto (descrito na Figura 21).

Figura 18 – Diagrama de Sequência - Manter Objeto



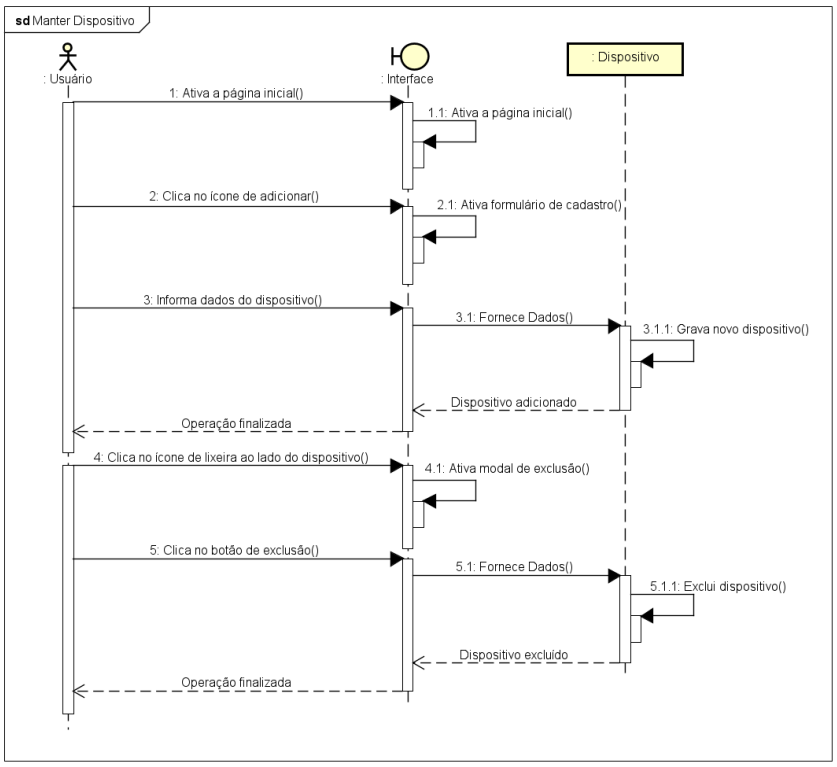
Fonte: Autor

Figura 19 – Diagrama de Sequência - Manter Imagem



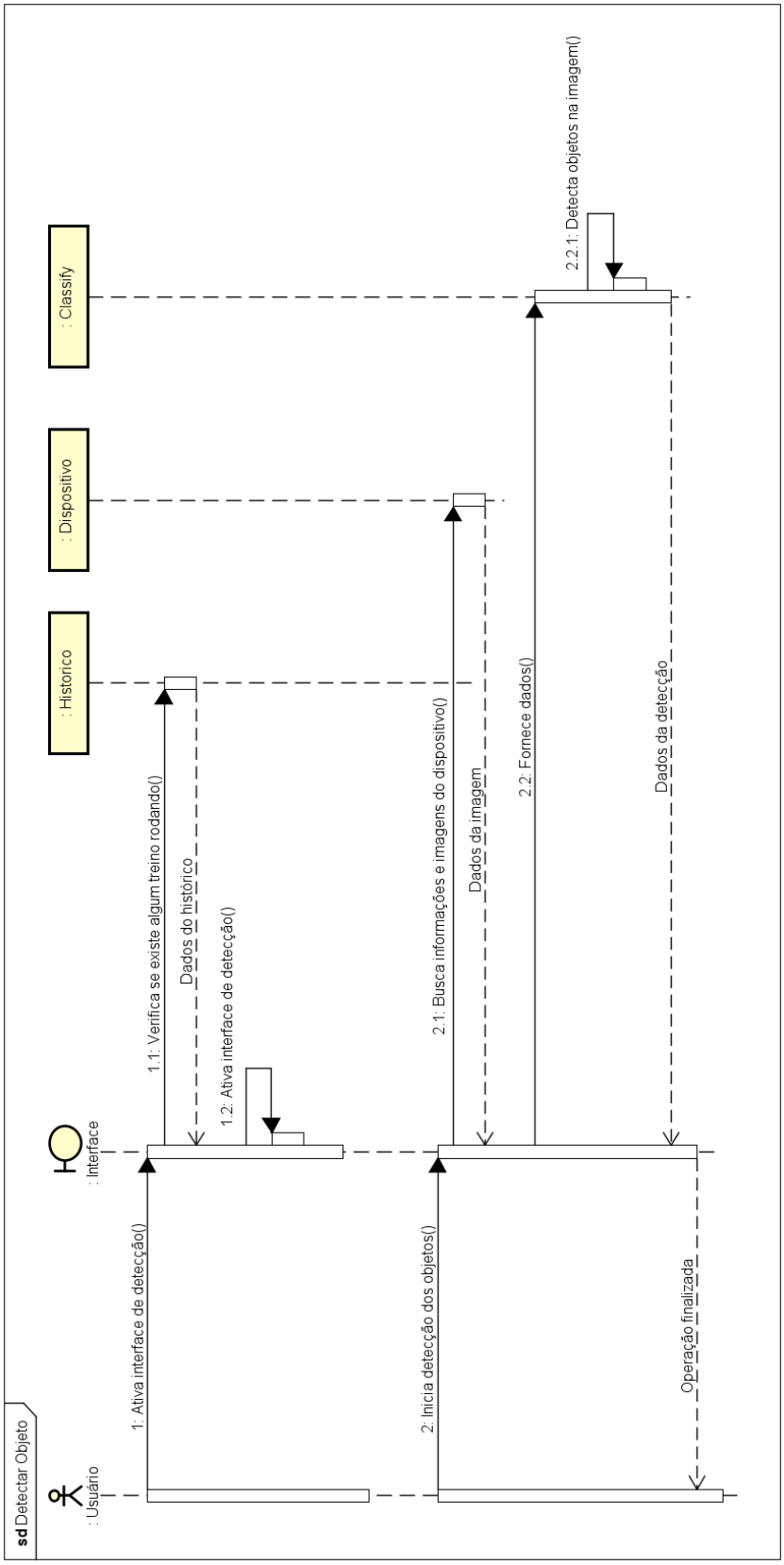
Fonte: Autor

Figura 20 – Diagrama de Sequência - Manter Dispositivo



Fonte: Autor

Figura 21 – Diagrama de Sequência - Detectar Objeto



Fonte: Autor

4.3 Construção do Sistema Embarcado

A parte de *hardware* foi feita através de um sistema embarcado que utilizou um dispositivo de Raspberry Pi 3, um módulo de Câmera v2 e um cartão SD 64GB utilizado para armazenar o sistema operacional e os dados do dispositivo. Para a melhoria do trabalho, também fez-se uso de um suporte para a câmera, objetivando proteger e melhorar o ângulo de obtenção das imagens, e um *case* com um *cooler* para proteção e redução do calor gerado pelos componentes. A Figura 22 mostra os componentes antes da montagem do sistema.

Figura 22 – Componentes do sistema embarcado

Fonte: Autor

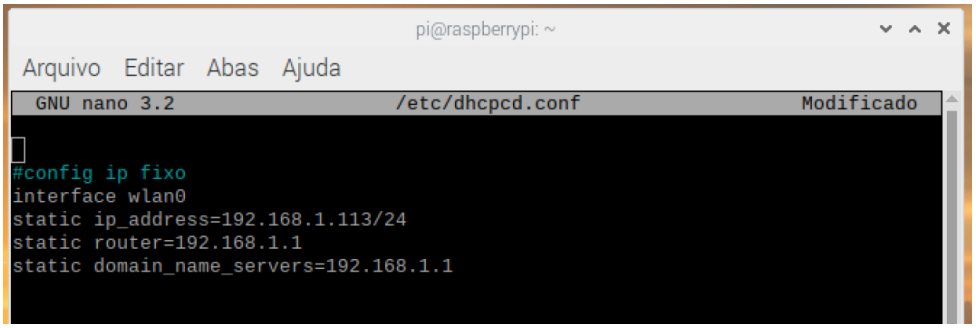
A construção do componente foi feita como demonstrado na Figura 23. Após esse processo, foram configurados os principais aspectos do funcionamento do sistema embarcado. Através da memória externa (cartão SD), foi instalada a imagem do sistema operacional e feitas as primeiras configurações na inicialização do mesmo. Entre os ajustes feitos estão: a configuração no armazenamento da memória (a fim de disponibilizar menor espaço para o sistema operacional e maior espaço para os outros dados do sistema), e a habilitação da câmera conectada na porta CSI.

Figura 23 – Sistema embarcado montado

Fonte: Autor

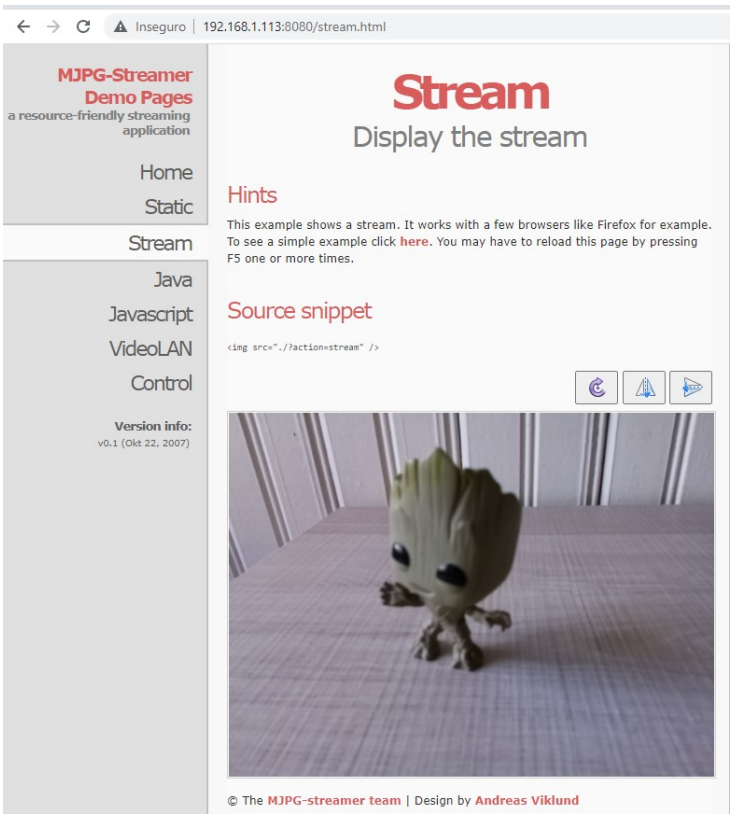
Após o término da configuração inicial do *hardware* foi feita a instalação do mjpg-streamer, que é um servidor capaz de gerar o *stream* de imagens da câmera para serem usadas na construção do *dataset* e na detecção do objetos. Como mostra a Figura 24, foi feita a configuração no arquivo */etc/dhcpcd.conf* com o intuito de estabelecer um IP fixo para o dispositivo. As imagens foram compartilhadas através de um IP local e um público, ambos na porta 8080, com a finalidade de acessar o sistema embarcado, tanto na rede local quanto fora dela. A Figura 25 mostra uma interface de demonstração do servidor com o *stream* das imagens.

Figura 24 – Configuração do IP fixo no Raspberry Pi



Fonte: Autor

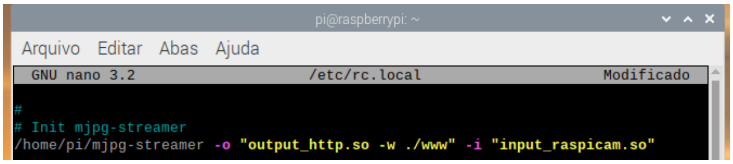
Figura 25 – Interface de demonstração do servidor *streamer*



Fonte: Autor

Para finalizar a construção do *hardware* foi configurado no arquivo */etc/rc.local* a inicialização do aplicativo de *streamer* para que o mesmo seja iniciado junto ao *boot* do sistema operacional. A configuração é mostrada na Figura 26. Após isso, o aparelho foi instalado em uma parede com a câmera em um ângulo propício para a obtenção de imagens úteis e com boa resolução e luminosidade. Para futuras atualizações, o sistema pode ser acessado remotamente ou as mesmas podem ser feitas através do cartão SD, que possibilita a fácil remoção do sistema. A Figura 27 demonstra a instalação final do sistema embarcado.

Figura 26 – Inicialização do servidor *streamer* no *boot* do SO



Fonte: Autor

Figura 27 – Instalação do sistema embarcado

Fonte: Autor

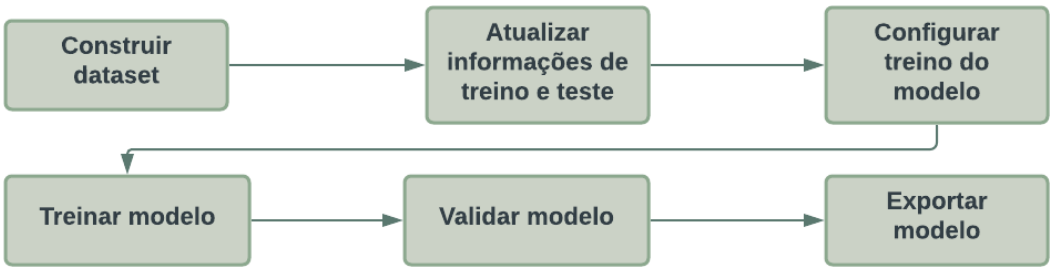
4.4 Treinando modelo de detecção

Na área de aprendizado de máquina, os modelos são construídos através do processamento de dados de entrada e por meio destes é possível de fazer previsões ou reconhecer algum padrão que já lhe foi apresentado. São utilizados dois tipos de dados para o aprendizado de um modelo, os de treino e os de teste. Primeiro, o algoritmo é treinado com os primeiros dados e, em seguida, sua acurácia é testada com base nos dados de teste. Se a acurácia for aceitável, o mesmo pode ser utilizado na aplicação (BUSSON et al., 2018).

Este capítulo detalha a construção do modelo utilizado para fazer a detecção dos objetos. O *framework* TensorFlow possui diversos modelos prontos, treinados com diversos dados e para diferentes propósitos. Entretanto, este trabalho faz uso de um modelo próprio de treino com base no *dataset* de imagens tiradas pelo sistema embarcado. A Figura 28 apresenta o fluxo-

grama com os passos do treinamento do modelo, baseado em um tutorial disponibilizado pelo TensorFlow[1] para a criação de modelos customizados.

Figura 28 – Fluxograma de treinamento do modelo



Fonte: Autor

Antes de iniciar os processos descritos no fluxograma, é essencial fazer o *download* de algumas ferramentas que serão utilizados nos próximos passos. Além do *framework* Tensorflow foi preciso fazer *download* dos seguintes utilitários:

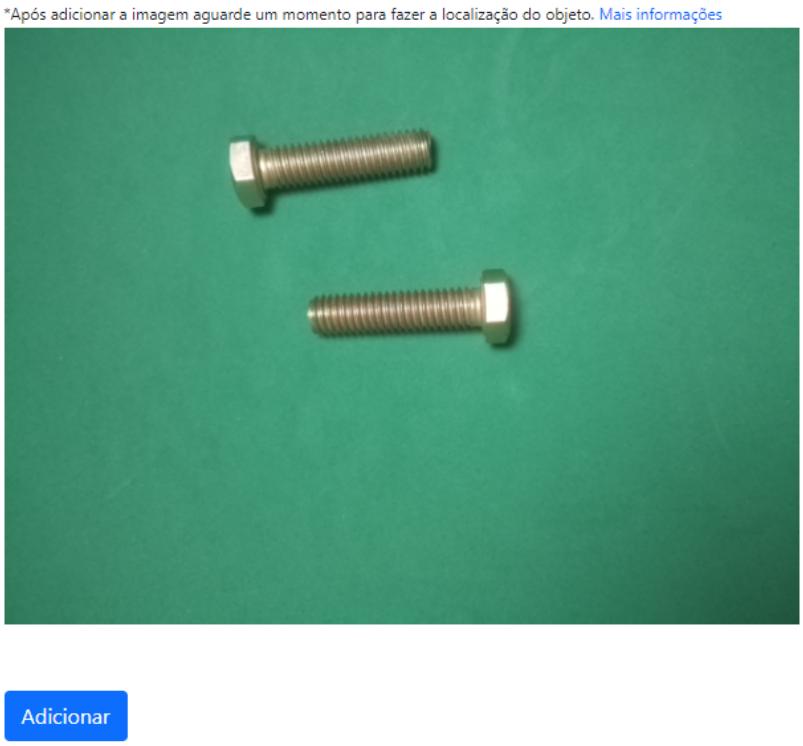
- **TensorFlow Model Garden** [2]**:** é necessário clonar o repositório *Model Garden* do Tensorflow que contém diversas implementações e utilitários para auxiliar no treinamento do modelo;
- ***labelImage*:** é uma ferramenta gráfica utilizada para destacar os objetos em uma imagem criando um arquivo xml que contém as informações do objeto e da imagem através da estrutura Pascal VOC;
- ***Protobuf*:** usado para configurar modelos e definir parâmetros do TensorFlow, funciona através da serialização de estruturas de dados.

4.4.1 Construir *dataset*

Para o treinamento de um modelo, os dados são uma peça chave. Por esse motivo, é necessário utilizar uma base de dados que contenha uma quantidade considerável de informações relevantes. Como mostrado na Figura 29, a criação do *dataset* será por meio da aplicação *web*, que captura as imagens fornecidas pelo sistema embarcado e armazena em um banco de dados para facilitar o acesso e manutenção.

[1] https://tensorflow-object-detection-api-tutorial.readthedocs.io/
[2] https://github.com/tensorflow/models

Figura 29 – Adição de imagem ao dataset



Fonte: Autor

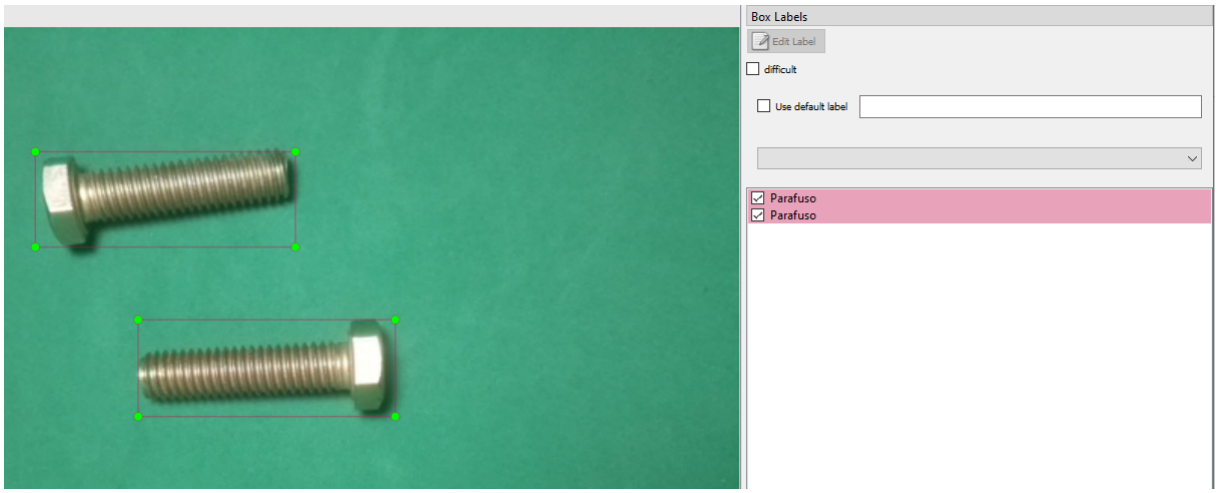
Após a captura e adição da imagem, é aberto a visualização do utilitário *labelImage*, conforme demonstrado na Figura 30. Neste passo, se faz necessário destacar a imagem com uma caixa e utilizando o *label* como o nome do objeto destacado. Salvando essas informações, um arquivo xml é gerado de acordo com a convenção Pascal VOC, demonstrado na Figura 31. Este arquivo contém as informações da imagem, bem como o nome e as coordenadas do item na figura.

Neste trabalho foi utilizado um *dataset* com cerca de 700 imagens, sendo 7 objetos com uma média de 100 imagens por objeto. Conforme mostrado na Figura 32, as imagens dos objetos foram separadas em pastas para que os dados sejam controlados facilmente e, também, para garantir que terá imagens de todos os objetos, tanto no treino, quanto no teste do modelo.

4.4.2 Atualizar informações de treino e teste

Após a construção do *dataset*, foi possível gerar as informações para realizar o treinamento do modelo. As informações necessárias estão armazenadas no banco de dados. Por esse motivo, é possível preparar essas informações através da página de treinamento disponível no sistema *web*. O primeiro passo da execução foi a divisão dos dados do *dataset* entre treino e teste. Para isso, foram calculados o total de imagens em cada pasta dos objetos, e destes, 80% foram copiados para uma nova pasta de treino e o restante foi copiado para uma nova pasta de teste. A Figura 33 demonstra o código de separação entre treino e teste.

Figura 30 – Interface do utilitário *labelImage*



Fonte: Autor

Outro processo necessário para o treinamento foi a atualização do mapa de *labels* do objeto através do arquivo *label_map.pbtxt*. As informações necessárias são o id e o nome do objeto, obtidos através da busca no banco de dados. O mapeamento é feito e salvo seguindo a estrutura demonstrada na Figura 34.

Para utilizar os dados do modelo no treino e no teste foi preciso convertê-los para um formato reconhecível pelo TensorFlow, chamado de TFRecord. Um arquivo TFRecord é um formato simples que possibilita armazenar os dados das imagens e do mapa de *labels* em uma sequência de *strings* binárias. Esses arquivos são gerados por meio de funções disponibilizadas pelo próprio TensorFlow. A Figura 35 mostra o código de criação dos arquivos, sendo que foram gerados dois arquivos um para o treino e um para o teste.

4.4.3 Configurar modelo pré-treinado

Embora seja possível criar um modelo a partir do zero, este trabalho utilizou um modelo pré-treinado disponibilizado pelo TensorFlow. O modelo utilizado foi o SSD ResNet50 V1 FPN 640x640, que possui um bom balanceamento entre velocidade e desempenho. Este modelo está disponível em um repositório, chamado TensorFlow 2 Detection Model Zoo [3], assim como diversos outros modelos pré-treinados.

O modelo possui algumas informações que devem ser preenchidas e foi necessário fazer configurações no *pipeline* do treinamento. As informações que devem ser usadas são com base nos arquivos gerados na etapa de atualização das informações de treino e teste. Foi preciso informar onde estão disponibilizados os arquivos gerados, como os TFRecords e o mapa de *labels*. Além dessas configurações, o *pipeline* possui diversas outras que podem ser alteradas conforme o poder computacional do dispositivo utilizado, métricas e outros quesitos.

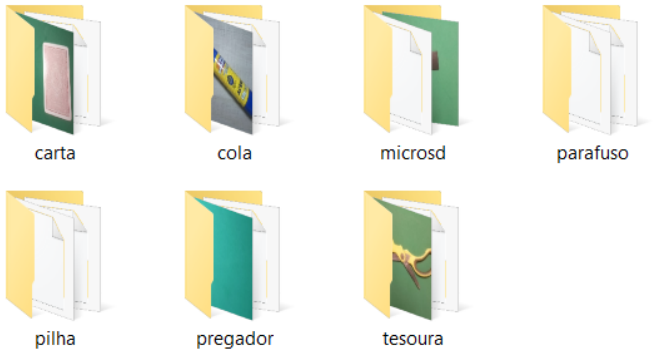
[3] https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

Figura 31 – Exemplo da convenção Pascal VOC

```xml
<annotation>
    <folder>parafuso</folder>
    <filename>309.jpg</filename>
    <path>C:\projetos\vision-detect\media\parafuso\309.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>640</width>
        <height>480</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>Parafuso</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>242</xmin>
            <ymin>193</ymin>
            <xmax>412</xmax>
            <ymax>257</ymax>
        </bndbox>
    </object>
    <object>
        <name>Parafuso</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>174</xmin>
            <ymin>82</ymin>
            <xmax>346</xmax>
            <ymax>145</ymax>
        </bndbox>
    </object>
</annotation>
```

Fonte: Autor

Figura 32 – *Dataset* separado por pastas dos objetos



Fonte: Autor

Figura 33 – Código de separação do *dataset*

```python
def partition_dataset(objeto, percent_to_test=0.2):
    # preparação das pastas
    train_dir = os.path.join(TRAIN)
    test_dir = os.path.join(TEST)
    source_dir = os.path.join(IMAGES + objeto.pasta)
    if not os.path.exists(train_dir):
        os.makedirs(train_dir)
    if not os.path.exists(test_dir):
        os.makedirs(test_dir)

    # lista as imagens da pasta
    images = [f for f in os.listdir(source_dir) if '.jpg' in f]

    # calcula a separação do total de imagens para treino e teste
    num_images = len(images)
    num_test_images = math.ceil(percent_to_test * num_images)

    # adiciona parte dos arquivos a pasta test
    for i in range(num_test_images):
        id_image = rand.randint(0, len(images) - 1)
        filename = images[id_image]
        xml_filename = os.path.splitext(filename)[0] + '.xml'
        copyfile(os.path.join(source_dir, filename), os.path.join(test_dir, filename))
        copyfile(os.path.join(source_dir, xml_filename), os.path.join(test_dir, xml_filename))
        images.remove(images[id_image])

    # adiciona o restante dos arquivos a pasta train
    for filename in images:
        copyfile(os.path.join(source_dir, filename), os.path.join(train_dir, filename))
        xml_filename = os.path.splitext(filename)[0] + '.xml'
        copyfile(os.path.join(source_dir, xml_filename), os.path.join(train_dir, xml_filename))
```

Fonte: Autor

Figura 34 – Exemplo de estrutura do *labelmap*

```
item{
    id: 2
    name: 'Cola'
}
item{
    id: 3
    name: 'Parafuso'
}
```

Fonte: Autor

4.4.4 Treinar modelo

Após a configuração e geração dos arquivos foi possível iniciar o treinamento do modelo de detecção de objetos. O modelo é gerado através de um *script* disponibilizado pelo TensorFlow e também com base nas configurações de camadas das redes neurais feitas no *pipeline*. O processo de treinamento é demorado e custa muito poder computacional. Devido a isso, foi

Figura 35 – Código de geração dos TFRecords

```
def generate_records(path_to_save, record):
    # criando arquivo .record
    writer = tf.io.TFRecordWriter(record)
    path_to_save = os.path.join(path_to_save)
    # criando dataframe com os arquivos .xml
    examples = xml_to_csv(path_to_save)
    # criando estrutura com o label_map
    grouped = split(examples, 'filename')
    for group in grouped:
        # criando o record com base nas imagens e labels
        tf_example = create_tf_example(group, path_to_save)
        writer.write(tf_example.SerializeToString())
    writer.close()
    print('Successfully created the TFRecord file: {}'.format(record))
```

Fonte: Autor

preciso dedicar uma janela de horário específica para rodar o mesmo, visto que o dispositivo pode gerar lentidão se forem utilizados outros recursos ao mesmo tempo. Por conta da demora na execução, esse processo foi separado do sistema *web* e foi executado manualmente através de linhas de comando.

O processo de treinamento finaliza quando chegar às etapas configuradas ou quando atingir um percentual de acerto confiável. O *pipeline* foi configurado com 25 mil etapas e a cada 100 passos era gerado um relatório com informações que descreviam a precisão, taxa de erro e outras informações do modelo, conforme mostrado na Figura 36. A métrica mais utilizada é a *total_loss* que deve ser inferior a 1 para ser aceitável. É preciso monitorar essas informações visando parar o treinamento quando necessário.

Figura 36 – Informações de saída no treinamento do modelo

```
I1113 16:17:19.888238  9584 model_lib_v2.py:698] Step 1700 per-step time 19.315s
INFO:tensorflow:{'Loss/classification_loss': 0.12612918,
 'Loss/localization_loss': 0.07739253,
 'Loss/regularization_loss': 0.25079226,
 'Loss/total_loss': 0.454314,
 'learning_rate': 0.03599995}
I1113 16:17:19.934525  9584 model_lib_v2.py:701] {'Loss/classification_loss': 0.12612918,
 'Loss/localization_loss': 0.07739253,
 'Loss/regularization_loss': 0.25079226,
 'Loss/total_loss': 0.454314,
 'learning_rate': 0.03599995}
```

Fonte: Autor

4.4.5 Validar modelo

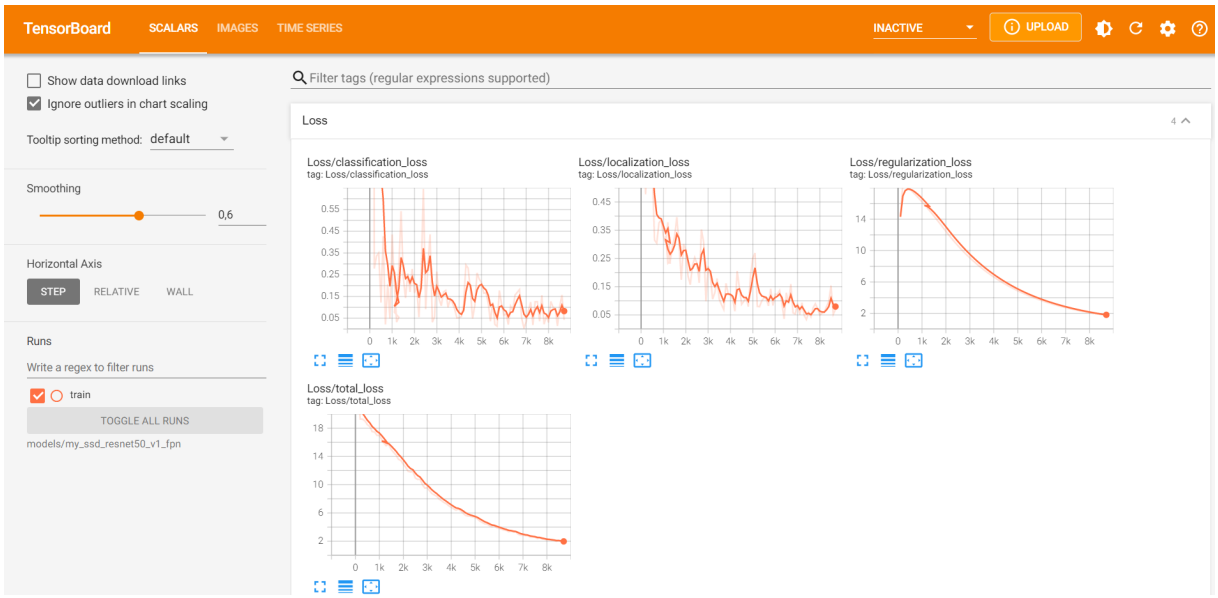
A validação do modelo é um processo importante para comprovar a efetividade do mesmo. Como descrito anteriormente, a cada 100 etapas o treinamento retorna uma saída com

as informações que são utilizadas para validar o modelo. Além dessas informações, existem outros métodos que podem ser utilizados para avaliar a acurácia, precisão e outras informações sobre o desempenho do modelo.

O TensorFlow possui um algoritmo que testa o modelo com base nas imagens da pasta teste. Esse arquivo Python é executado através de linhas de comando, e avalia a acuracidade do modelo através da predição das imagens testes. Além disso, utiliza os dados fornecidos pelo treino para gerar métricas consistentes. Tanto o treinamento quanto o teste geram arquivos com as informações que podem ser interpretadas e exibidas pela ferramenta do TensorFlow, chamada de TensorBoard.

O TensorBoard possibilita visualizar diversas métricas de desempenho do modelo. Este utilitário permite monitorar e visualizar continuamente as informações do treinamento, como a acurácia, resultado dos testes, avaliação de desempenho, entre outras. Após sua instalação e configuração, é possível visualizar um painel que possui gráficos com as métricas utilizadas na validação do modelo. A Figura 37 demonstra a interface do TensorBoard com as métricas de um modelo.

Figura 37 – Métricas exibidas no TensorBoard



Fonte: Autor

4.4.6 Exportar modelo

Para finalizar o processo de treinamento do modelo é preciso exportá-lo. Depois de extrair o gráfico de inferência recém-treinado, é possível utilizá-lo no sistema com o intuito de realizar a detecção dos objetos. O modelo é exportado através de um comando disponibilizado pelo TensorFlow. Basta rodar o arquivo Python através das linhas de comando e o modelo será exportado para uma nova pasta disponível para a utilização.

4.5 Algoritmo de detecção

Após o treinamento, foi necessário desenvolver o algoritmo com o intuito de utilizar o modelo para detectar e fazer a contagem dos objetos. Essa função foi desenvolvida e disponibilizada através de uma API com a rota configurada pelo *framework* Django. O acesso a API foi feito por meio da linguagem JavaScript que recebia as informações e renderizava na tela para a visualização do usuário.

O primeiro passo para o desenvolvimento do algoritmo foi o carregamento das informações necessárias. O carregamento do modelo é demorado e por esse motivo, o modelo e o mapa dos *labels* foram acessados de forma global, para não ser necessário repetir esse processo toda a vez que a função for executada, conforme mostrado na Figura 38.

Figura 38 – Carregando informações para a detecção

```
# Definições do caminho dos arquivos
path_to_labels = 'C:/projetos/vision-detect/TensorFlow/workspace/training_demo/annotations/label_map.pbtxt'
path_to_saved_model = "C:/projetos/vision-detect/TensorFlow/workspace/training_demo/exported-models/my_model/" \
                      "saved_model"

# Carregando modelo e mapa de labels
detect_fn = tf.saved_model.load(path_to_saved_model)
category_index = label_map_util.create_category_index_from_labelmap(path_to_labels)
```

Fonte: Autor

Para iniciar a função de detecção e contagem de objetos, é preciso preparar algumas informações. São geradas duas imagens, uma antes e uma depois da detecção. A primeira é capturada através do sistema embarcado e a segunda é criada após a detecção, com base na primeira imagem. Outra informação importante é a porcentagem mínima para o modelo considerar como um objeto. Essa variável, chamada de *min_score*, foi definida com 70%. A Figura 39 demonstra o trecho de código responsável por capturar as imagens.

Figura 39 – Preparação das imagens

```
def detect_objects(request):
    # ajustando nome da imagem e informações iniciais
    path_image = 'C:/projetos/vision-detect/visiondetect/static/detection/'
    time_now = strftime('%Y%m%d%H%M%S', gmtime())
    image_origin = path_image + time_now + '_origin.jpg'
    name_image_final = time_now + '_final.jpg'
    image_final = path_image + name_image_final
    min_score = 0.7  # porcentagem minima para mostrar box do objeto

    # capturando imagem do sistema embarcado
    dispositivo = Dispositivo.objects.filter(selecionado=True)[0]
    url = 'http://{0}:8080/?action=stream'.format(dispositivo.ip)
    cap = cv2.VideoCapture(url)
    ret, frame = cap.read()
    cv2.imwrite(image_origin, frame)
    image_path = image_origin
```

Fonte: Autor

Após o carregamento do modelo e a captura das imagens, a inferência foi executada com base na imagem original. Com a execução desse processo foram gerados dados que possibilitam a contagem dos objetos e marcação dos mesmos na imagem para uma melhor visualização do que foi detectado. A Figura 40 demonstra a execução da inferência e extração dos dados da mesma.

Figura 40 – Processo de inferência

```
# rodando inferencia do modelo com base na imagem
image_np = load_image_into_numpy_array(image_path)
input_tensor = tf.convert_to_tensor(image_np)
input_tensor = input_tensor[tf.newaxis, ...]
detections = detect_fn(input_tensor)
num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy() for key, value in detections.items()}
detections['num_detections'] = num_detections
detections['detection_classes'] = detections['detection_classes'].astype(np.int64)
image_np_with_detections = image_np.copy()
```

Fonte: Autor

Para finalizar o processo, foram utilizadas as informações geradas para a exibição. Com as informações coletadas, foram criadas as caixas para destaque dos objetos na imagem. A imagem final foi salva e as informações foram utilizadas para gerar um *array* com as informações da contagem dos objetos. Após esses processos, os dados foram retornados no formato JSON para serem utilizados na exibição para o usuário na aplicação *web*. A Figura 41 exibe os processos de aplicação e retorno dos dados coletados.

Figura 41 – Processo de aplicação e retorno dos dados

```
# adicionando box nos objetos detectados
viz_utils.visualize_boxes_and_labels_on_image_array(
    image_np_with_detections,
    detections['detection_boxes'],
    detections['detection_classes'],
    detections['detection_scores'],
    category_index,
    use_normalized_coordinates=True,
    max_boxes_to_draw=200,
    min_score_thresh=min_score,
    agnostic_mode=False)
# salvando a imagem
plt.savefig(image_final)
# realizando contagem dos objetos
contagem = return_objets_count(detections['detection_classes'].astype(np.int64),
                               detections['detection_scores'], min_score)
# retornando informações da detecção
data = {'imagem': '/static/detection/' + name_image_final, 'contagem': json.dumps(contagem)}
return JsonResponse(data)
```

Fonte: Autor

A Figura 42 demonstra a função de contagem dos objetos. A função cria uma estrutura de dados, com o nome do objeto e a quantidade do mesmo. A contagem é feita com base na porcentagem gerada pelo algoritmo, e o nome do objeto foi acessado através do banco de dados de acordo com o id retornado pelo *array*.

Figura 42 – Função de Contagem de objetos

```
# função de contagem dos objetos
def return_objets_count(id_objects, scores, min_score):
    contagem = {}
    for i in range(0, len(id_objects)):
        sc_i = scores[i]
        if sc_i > min_score:
            r_objeto = get_object_or_404(Objeto, pk=id_objects[i])
            if r_objeto.nome in contagem:
                contagem[r_objeto.nome] = contagem[r_objeto.nome] + 1
            else:
                contagem[r_objeto.nome] = 1
    return contagem
```

Fonte: Autor

O modelo é inicializado junto com o servidor, visto que demanda um tempo para o mesmo ser carregado. Após a disponibilização do modelo, o *frame* capturado demora cerca de 3 segundos para passar pela detecção e ser disponibilizado para a visualização do usuário. Para os trabalhos futuros poderia ser exibida essa detecção mais dinamicamente, mostrando um novo *frame* após um tempo determinado ou após a imagem anterior ser devidamente processada.

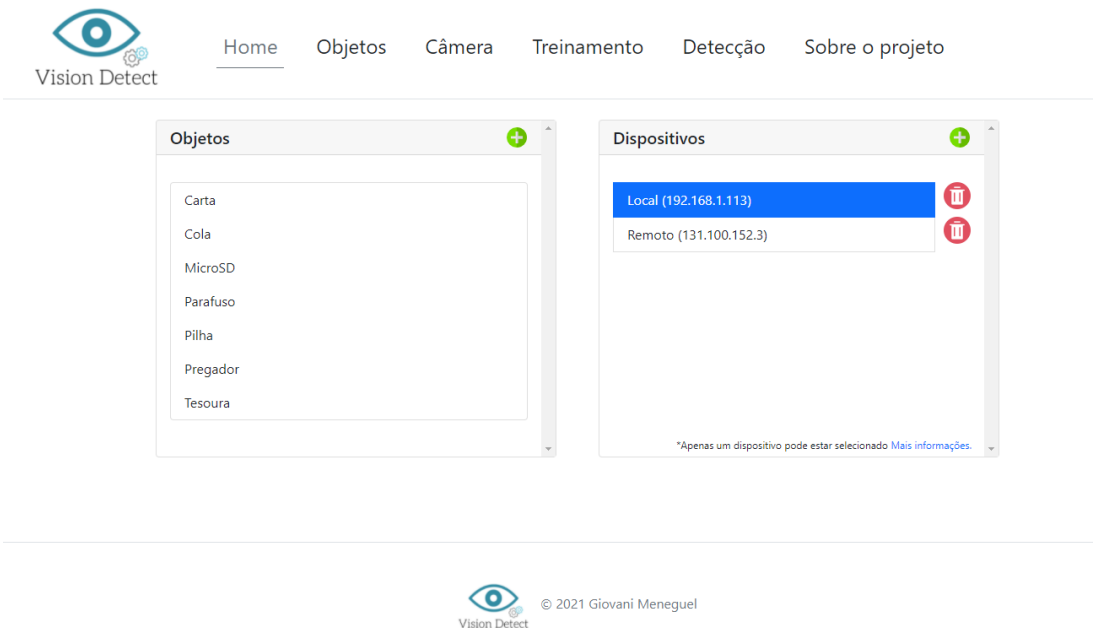
5 DESCRIÇÃO DO SISTEMA

Este capítulo apresenta a descrição das telas do aplicativo *web*. Para facilitar a interação do usuário com o trabalho, foi desenvolvido através do *framework* Django, um sistema *web* que contém as informações do utilitário, como as imagens dos objetos, uma interface de treinamento, interface de detecção e outras telas que são descritas neste capítulo. Além do Django, foram aproveitados outros utilitários como HTML e CSS para estruturar e adicionar estilo ao site. Também foi usado JavaScript com intuito de estabelecer alguns comportamentos para a implementação.

O nome do trabalho, Vision Detect, teve inspiração na junção de *Vision Computer*, Visão Computacional em português, com o termo *Detection*, que significa detecção, fazendo uma combinação da área de estudo com a funcionalidade principal do trabalho.

A Figura 43 mostra a página inicial da aplicação. Através dela é possível visualizar os objetos cadastrados, além de possibilitar o cadastro de mais objetos. É possível também selecionar o dispositivo que será utilizado para captura das imagens do sistema embarcado. É possível selecionar apenas um dispositivo e as informações do mesmo serão utilizadas em outros processos do sistema.

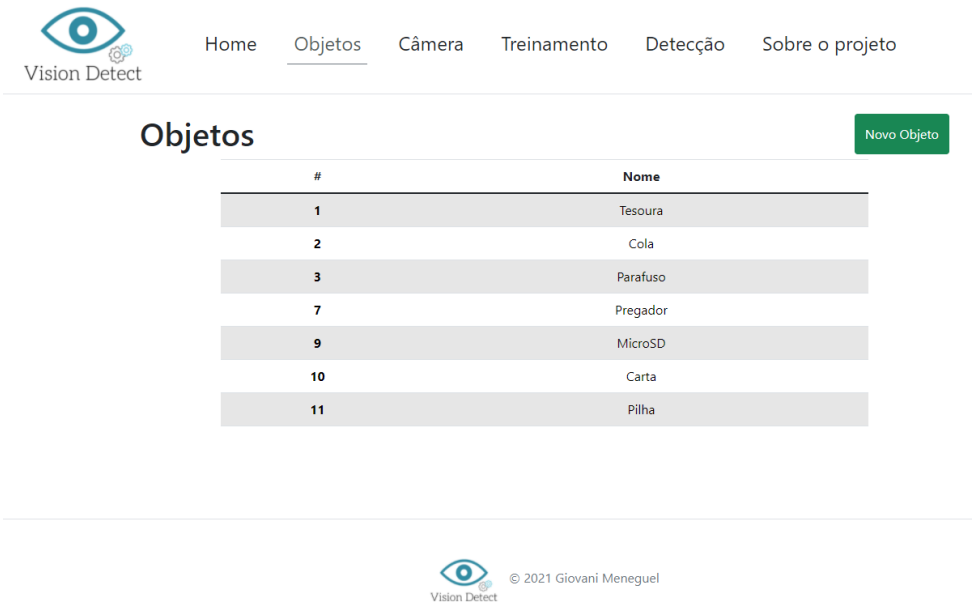
Figura 43 – Página Inicial



Fonte: Autor

A página de objetos contém uma lista com todos os itens cadastrados. Nela é possível cadastrar novos objetos, bem como acessar e visualizar as informações existentes relativas aos mesmos. A Figura 44 demonstra a página de objetos do sistema.

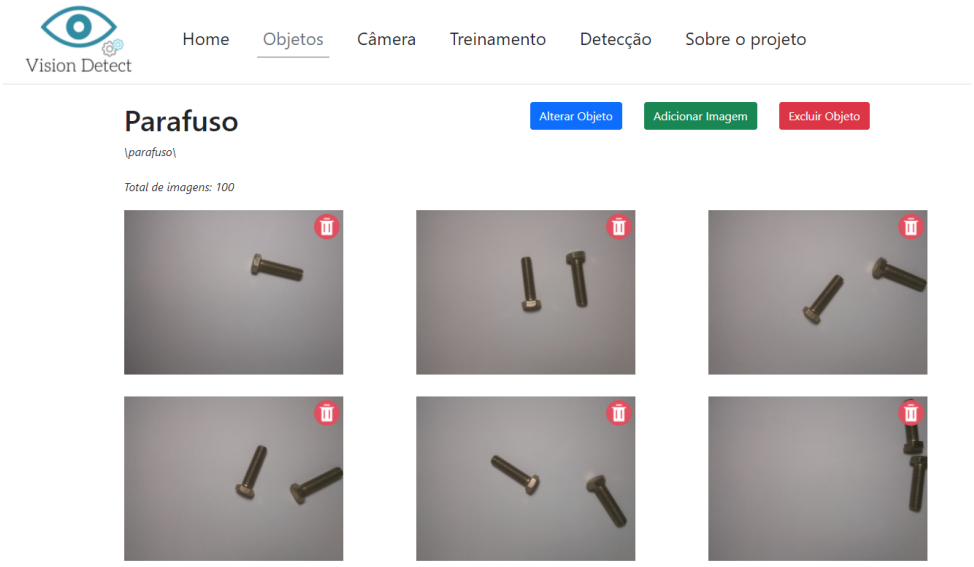
Figura 44 – Página de Objetos



Fonte: Autor

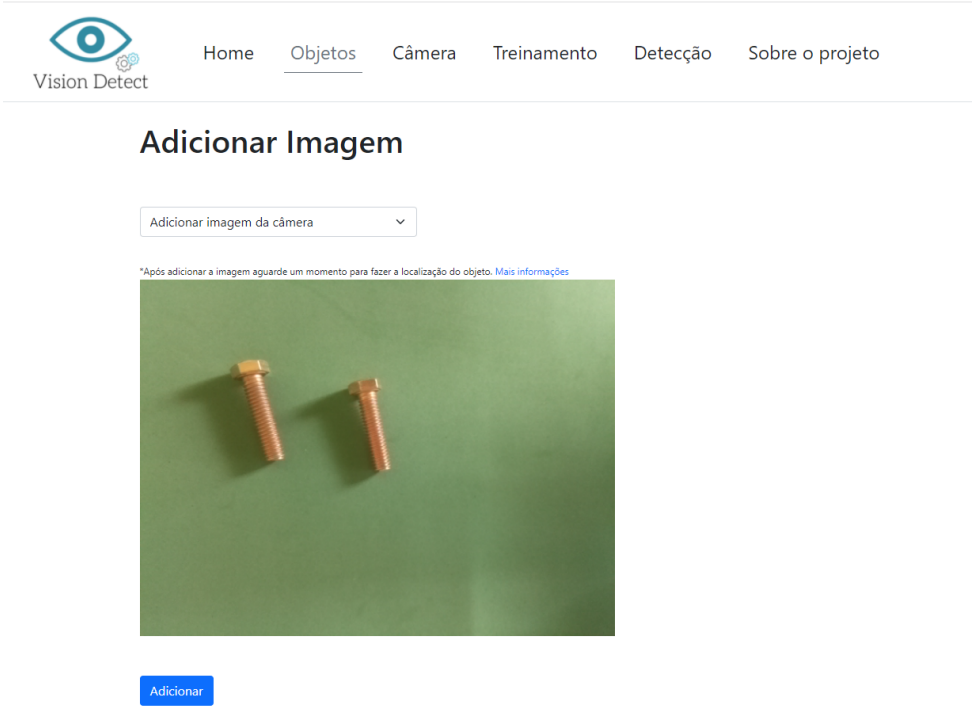
Ao acessar um objeto, a interface demonstrada na Figura 45 é exibida. Nesta página é possível visualizar as informações do objeto e todas as imagens cadastradas. Os botões exibidos no topo da página possibilitam a alteração e exclusão do item, e o botão de adicionar uma imagem redireciona para uma nova página de cadastro de imagem demonstrado na Figura 46.

Figura 45 – Página de informações do Objeto



Fonte: Autor

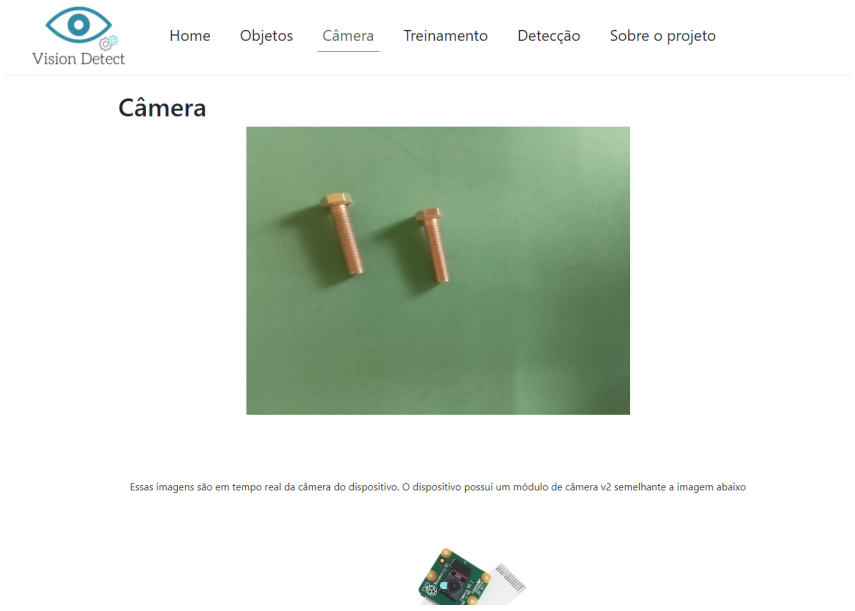
Figura 46 – Página de adição de Imagem



Fonte: Autor

A Figura 47 mostra a página de exibição da câmera. A partir dela é possível visualizar o *stream* da câmera. Essa página serve para testar o funcionamento das imagens provindas do sistema embarcado.

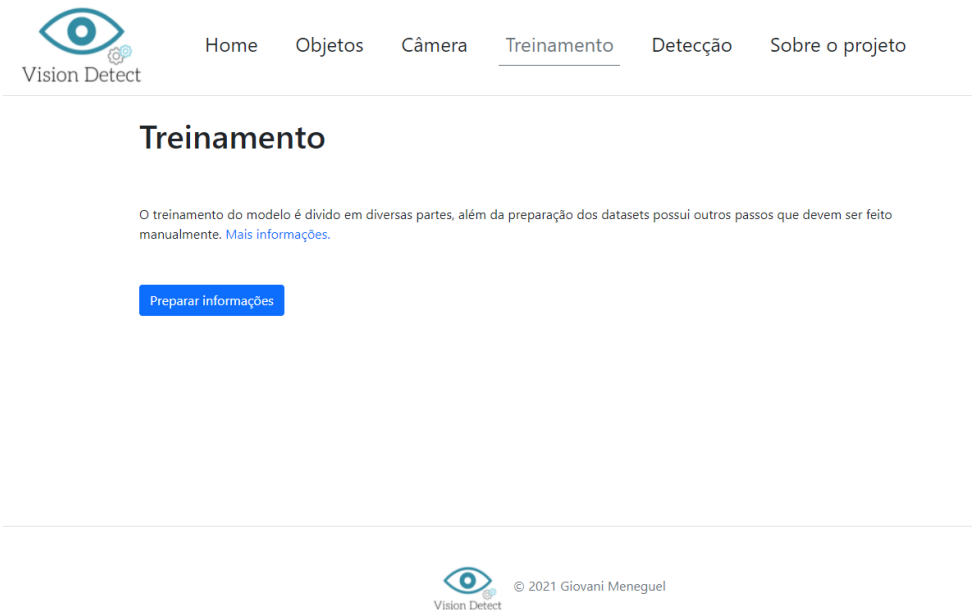
Figura 47 – Página de visualização da câmera



Fonte: Autor

A página de treinamento é mostrada na Figura 48. A página de treinamento, como descrito no capítulo anterior, permite ao usuário preparar as informações para o treinamento do modelo. Essa página apenas gera as informações iniciais. Os passos restantes devem ser feitos através de linhas de comando.

Figura 48 – Página de Treinamento



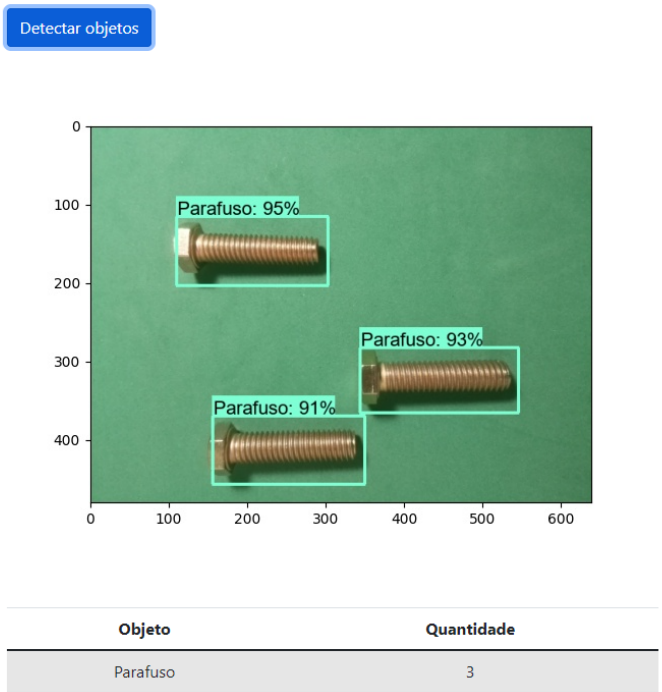
Fonte: Autor

A Figura 49 demonstra a página de detecção. Nesta página é feita a contagem e detecção dos objetos. Ao clicar no botão para indentifica-los, uma imagem é capturada através da câmera do sistema embarcado e então, o modelo é executado para a detecção das informações da imagem. Ao final do processo, é exibida a figura com uma tabela de informações sobre os itens detectados na imagem, como mostra a Figura 50. Se um objeto não estiver cadastrado no sistema de forma correta, o mesmo não será considerado nesse processo.

Figura 49 – Página de Detecção

Fonte: Autor

Figura 50 – Página com as informações da Detecção



Fonte: Autor

A página de informações do sistema é demonstrada na Figura 51. Essa página tem a função de exibir diversas informações úteis sobre o funcionamento e usabilidade do sistema.

Figura 51 – Página de informações sobre o sistema

Fonte: Autor

6 RESULTADOS OBTIDOS

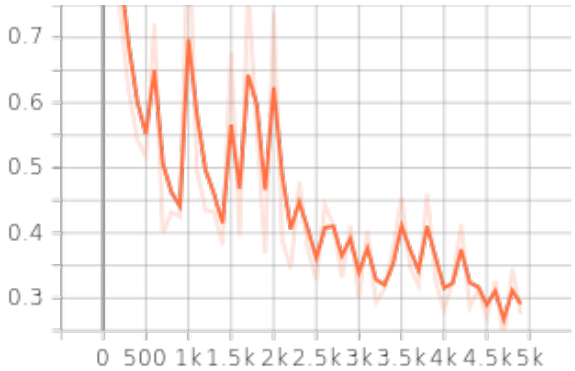
Neste capítulo são detalhados os processos de testagem e a descrição dos resultados obtidos, durante e após o desenvolvimento do sistema. Ao finalizar a implementação de todas as funcionalidades é de suma importância fazer a testagem das mesmas, a fim de obter métricas para concluir a eficácia do utilitário. Por esse motivo, foram feitas avaliações dos elementos que constituem a arquitetura do sistema e as considerações são descritas a seguir.

O utilitário *web* foi um elemento essencial para o trabalho, pois facilitou a usabilidade do sistema, com a possibilidade de visualizar e modificar os dados cadastrados de forma rápida, além da possibilidade de acessar o *stream* das imagens e a detecção dos objetos de forma prática. O sistema embarcado facilitou a obtenção dos dados, visto que após sua configuração, o mesmo não necessitou ser modificado e todas as imagens foram capturadas facilmente. O *hardware* utilizado foi satisfatório, desempenhando bem suas funcionalidades, porém a câmera escolhida, apesar de ter uma boa qualidade, possui o foco manual e de difícil utilização, o que dificultou a configuração do mesmo.

Durante o processo de desenvolvimento, diferentes modelos foram treinados, utilizando *datasets* menores e com menos quantidade de objetos. Com a evolução do mesmo obteve-se a base de dados utilizada no treinamento do modelo final do trabalho. O *dataset* utilizado é baseado no cadastro de objetos diversos com uma média de 100 imagens por objeto. Essa quantidade de dados é satisfatória para realizar o reconhecimento, contudo ela pode ser incrementada a fim de melhorar a acurácia e a assertividade do modelo.

O treinamento do modelo final durou cerca de 20 horas, devido a quantidade de dados e objetos disponibilizados para o treino e o teste. O treinamento foi finalizado depois de 5 mil etapas, onde atingiu 0.2907 no parâmetro *total_loss*, conforme demonstrado no gráfico da Figura 52. A acurácia do modelo obtido foi de 88%, um bom percentual o que caracteriza o modelo satisfatório para a utilização.

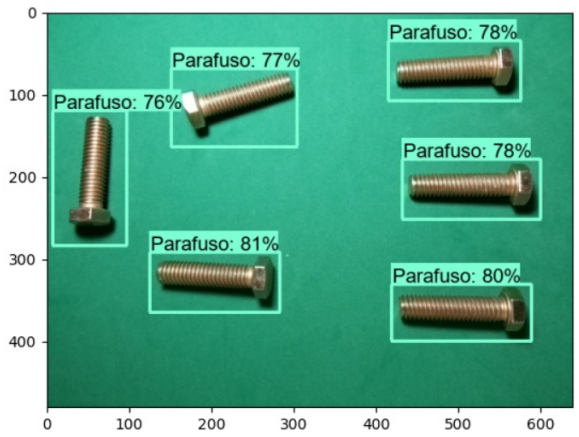
Figura 52 – Métrica do modelo



Fonte: Autor

Os resultados práticos da testagem do modelo de detecção e contagem de objetos estão demonstrados nas Figuras 53, 54, 55 e 56.

Figura 53 – Teste prático



Objeto	Quantidade
Parafuso	6

Fonte: Autor

Figura 54 – Teste prático

Objeto	Quantidade
Parafuso	3
Cola	1

Fonte: Autor

Figura 55 – Teste prático

Objeto	Quantidade
Cola	1
Tesoura	1

Fonte: Autor

Figura 56 – Teste prático

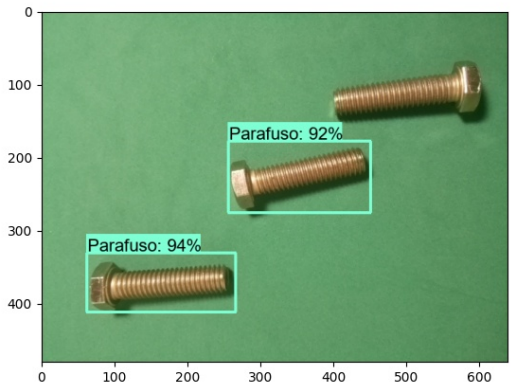
Objeto	Quantidade
Cola	1
Tesoura	1
Parafuso	1

Fonte: Autor

Apesar do bom funcionamento do modelo, foram observados alguns comportamentos

que prejudicam a acurácia do modelo de detecção. O problema encontrado diz respeito a objetos que não são detectados, dependendo da posição que estão dispostos na captura de imagem. Esse comportamento pode ser resolvido com o acréscimo de dados ao dataset. A Figura 57 demonstra o comportamento descrito.

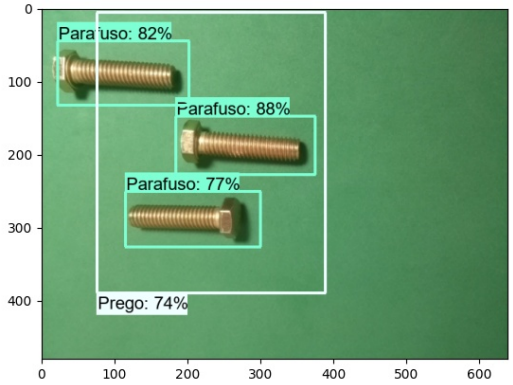
Figura 57 – Teste prático - objeto não reconhecido



Fonte: Autor

Outro problema encontrado possui relação com objetos pequenos. O modelo tende a reconhecer o fundo da imagem como o objeto, visto que ao selecionar o item na ferramenta *labelImage*, uma boa parte do objeto passa a ser o fundo, pois seu tamanho é muito pequeno, tornando se difícil de fazer a seleção. Esse comportamento foi observado em pregos, porcas e alfinetes. Portanto, esses objetos foram removidos do sistema e, consequentemente, não foram utilizados no treinamento do modelo final. A Figura 58 demonstra o erro descrito no processo de detecção.

Figura 58 – Teste prático - objeto reconhecido indevidamente



Fonte: Autor

Em resumo, é possível destacar os seguintes pontos como resultados para o trabalho:

- O *dataset* utilizado é satisfatório, porém para uma maior assertividade, é necessário cadastrar mais imagens para compor a base de dados;
- É recomendado evitar objetos pequenos que podem causar comportamentos inesperados, ou tomar cuidado no ato de captura e identificação do objeto para que a caixa de identificação não possua maior parte como sendo a superfície em que o item está disposto;
- O modelo atingiu métricas positivas com o *total_loss* em 0.2907 e a acurácia sendo 88%;
- A utilização do sistema *web* e do sistema embarcado acrescenta praticidade e desempenho para o trabalho, entretanto convém ressaltar que é recomendado utilizar uma câmera com foco automático;
- É importante ressaltar que objetos não cadastrados corretamente no sistema não serão detectados pelo modelo.

7 CONCLUSÃO

O presente trabalho teve como principal objetivo a implementação de um sistema que fosse capaz de realizar a contagem e detecção de objetos. Por conta disso, as primeiras etapas foram de estudo, envolvendo as tecnologias e o referencial teórico, tendo como base a área de visão computacional. Em vista desses estudos, foi possível entender o conceito para o desenvolvimento do sistema, bem como quais ferramentas e utilitários poderiam ser usados para a construção do mesmo.

Com a elaboração deste trabalho, foi possível entender conceitos mais específicos sobre o campo de visão computacional, desde o processo de obtenção de uma imagem até o treinamento de um modelo para a detecção de objetos. Ademais, estes temas foram abordados tanto de forma teórica quanto de forma prática, possibilitando um melhor aprendizado sobre todos os aspectos, auxiliando também, no conhecimento e utilização das diversas ferramentas desta área e de outras como: o TensorFlow, *framework* Django e os utilitários da marca Raspberry Pi.

O estudo ainda auxiliou na compreensão de forma prática dos diversos aspectos de diferentes áreas que o curso de Ciência da Computação engloba. Além do aprofundamento da pesquisa na área de inteligência artificial, visão computacional e processamento de imagens, outros conceitos foram melhor explorados, como os da área de Redes e Arquitetura de Computadores, muito utilizados na construção do sistema embarcado. Para a implementação do sistema *web* foram necessários conhecimentos de Engenharia de *Software*, Lógica e outros temas estudados ao longo do curso. Em vista disso, é importante ressaltar que este é um trabalho multidisciplinar, onde houve a possibilidade de agregar diversas competências para a elaboração do mesmo.

Através dos resultados apresentados, com base na acurácia, testagem do sistema e outros quesitos, pode-se provar que o sistema é capaz de auxiliar no processo de contagem de objetos. Como foi demonstrado, é possível cadastrar no sistema, quaisquer itens que precisam ser detectados ou contados, contanto que sejam em uma escala pequena, conforme apresentados nos exemplos do trabalho. Desta forma, é importante destacar que o trabalho pode auxiliar em atividades como vendas ou no controle de estoque. Embora tenham ocorridos alguns erros na detecção, a funcionalidade de adição de imagem pode resolver o problema, pois quanto maior a base de dados, maior será a acurácia e a assertividade do sistema.

Para trabalhos futuros é fundamental levar em conta alguns pontos, como: a criação do *dataset*, o *layout* da implementação web e o treinamento do modelo. A criação do dataset é uma execução muito trabalhosa, pois é necessário capturar e destacar o objeto na imagem. Esta função simples se torna algo demorado e cansativo, visto que o tamanho do *dataset* deve ser satisfatório, e todo o trabalho acaba sendo manual. Por meio destas informações, é possível automatizar esse processo, ou então buscar um meio de não se fazer necessário o destacamento do objeto na figura. Como apresentado nos Resultados, é pertinente acrescentar dados ao *dataset* para aperfeiçoar o modelo, considerando este ponto como um aprimoramento para o sistema.

Outro ponto que demanda alto custo computacional, e se torna um processo demorado, é o treinamento do modelo. Como descrito anteriormente, o modelo final demorou cerca de 20 horas para finalizar a execução. Por esse motivo, podem ser desenvolvidas outras técnicas de treinamento que busquem diminuir essa janela de tempo e facilitar a execução do mesmo. Tornando esse processo mais simples, é possível automatizá-lo nas etapas de treinamento disponíveis na implementação *web*.

No que diz respeito ao sistema *web*, é possível fazer algumas modificações com vista a melhorar a experiência do usuário. Na página de detecção é possível mostrar o *stream* da câmera exibindo as informações de detecção, melhorando a forma de apresentação que é feita utilizando apenas um *frame*. Este processo pode ser criado em um servidor à parte que utilize as imagens do sistema embarcado e disponibilize o vídeo com a detecção já executada. Além disso, o sistema pode ser alterado para melhorar o *layout* e a usabilidade, ou até ser modificado, a fim de atender outras demandas que envolvam a área de detecção de objetos ou de visão computacional.

A experiência vivenciada na construção deste trabalho auxiliou para uma maior aprendizagem sobre a área de visão computacional, reforçando a ideia de como ela é rica e pode abranger diversas usabilidades, visando melhorar os mais variados setores da sociedade. Portanto, a pesquisa e a busca de conhecimento nesta área, que possui projetos que podem ser agregados na utilização e evolução da tecnologia, tornam-se imprescindíveis.

REFERÊNCIAS

ACADEMY, E. D. S. **O que é Visão Computacional?** 2018. Disponível em: <https://blog.dsacademy.com.br/o-que-e-visao-computacional/>. Acesso em: 19 de junho de 2021. Citado na página 7.

ACADEMY, P. **Desenvolvimento web com Python e Django**. 2021. Disponível em: <https://pythonacademy.com.br/blog/desenvolvimento-web-com-python-e-django-introducao>. Acesso em: 24 de setembro de 2021. Citado na página 17.

BALLARD, D. H.; BROWN, C. M. **Computer Vision**. [S.l.]: Prentice Hall, 1982. v. 1. Citado 2 vezes nas páginas 6 e 7.

BARELLI, F. **Introdução à Visão Computacional: Uma abordagem prática com Python e OpenCV**. [S.l.]: Casa do Código, 2018. v. 1. Citado 6 vezes nas páginas 1, 3, 5, 6, 7 e 9.

BARROS, E.; CAVALCANTE, S. Introdução aos sistemas embarcados. **Artigo apresentado na Universidade Federal de Pernambuco-UFPE**, p. 36, 2010. Citado 2 vezes nas páginas 11 e 12.

BUSSON, A. J. G. et al. Desenvolvendo modelos de deep learning para aplicações multimídia no tensorflow. **Sociedade Brasileira de Computação**, 2018. Citado na página 35.

CUNHA, A. F. O que são sistemas embarcados. **Saber Eletrônica**, v. 43, n. 414, p. 1–6, 2007. Citado na página 11.

DOCS, M. W. **Introdução ao Django**. 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Django/Introduction>. Acesso em: 24 de setembro de 2021. Citado na página 17.

DOMINGOS, P. **O algoritmo mestre: como a busca pelo algoritmo de machine learning definitivo recriará nosso mundo**. [S.l.]: Novatec Editora, 2017. Citado na página 10.

FOUNDATION, D. S. **Documentação Oficial**. 2021. Disponível em: <https://www.djangoproject.com/start/overview/>. Acesso em: 20 de setembro de 2021. Citado na página 17.

FRUTUOSO, E. B. et al. Arduino® e raspberry pi®: Uma comparação de especificações e aplicações de minicomputadores. Citado na página 13.

GONZALEZ, R. C.; WOODS, R. C. **Processamento digital de imagens .** [S.l.]: Pearson Educación, 2009. Citado 4 vezes nas páginas 1, 3, 4 e 8.

GUEDES, G. T. **UML 2-Uma abordagem prática**. [S.l.]: Novatec Editora, 2018. Citado 2 vezes nas páginas 21 e 25.

HEATH, S. **Embedded systems design**. [S.l.]: Elsevier, 2002. Citado na página 11.

HENRIQUE, J. **Uma breve introdução a Redes neurais e "Deep Learning"**. 2018. Disponível em: <https://medium.com/@joohenrique_7804/uma-breve-introdu%C3%A7%C3%A3o-a-redes-neurais-e-deep-learning-243bf283f731>. Acesso em: 30 de outubro de 2021. Citado 2 vezes nas páginas 10 e 11.

HOPE, T.; RESHEFF, Y. S.; LIEDER, I. **Learning tensorflow: A guide to building deep learning systems**. [S.l.]: "O'Reilly Media, Inc.", 2017. Citado na página 18.

KOT, E. **Visão Computacional: O que é?** 2021. Disponível em: <https://kotengenharia.com.br/visao-computacional-o-que-e/>. Acesso em: 19 de junho de 2021. Citado 2 vezes nas páginas 7 e 8.

LIAM, J. **Documentação oficial**. 2021. Disponível em: <https://github.com/jacksonliam/mjpg-streamer>. Acesso em: 14 de outubro de 2021. Citado na página 19.

LUBANOVIC, B. **Introducing Python: Modern Computing in Simple Packages**. [S.l.]: "O'Reilly Media, Inc.", 2014. Citado na página 16.

MARWEDEL, P. **Embedded system design: embedded systems foundations of cyber-physical systems, and the internet of things**. [S.l.]: Springer Nature, 2021. Citado na página 12.

MILANO, D. de; HONORATO, L. B. **Visão computacional**. 2014. Citado 4 vezes nas páginas 6, 7, 8 e 9.

OLIVEIRA, S. de. **Internet das coisas com ESP8266, Arduino e Raspberry PI**. [S.l.]: Novatec Editora, 2017. Citado na página 13.

OPENCV. **Documentação Oficial**. 2021. Disponível em: <https://opencv.org/about/>. Acesso em: 04 de junho de 2021. Citado na página 18.

QUEIROZ, J. E. R. de; GOMES, H. M. Introdução ao processamento digital de imagens. **Rita**, v. 13, n. 2, p. 11–42, 2006. Citado 2 vezes nas páginas 3 e 6.

RASPBERRYPI. **Documentação Oficial**. 2021. Disponível em: <https://www.raspberrypi.org/about/>. Acesso em: 13 de julho de 2021. Citado na página 13.

RASPBERRYPI. **Documentação Oficial**. 2021. Disponível em: <https://www.raspberrypi.org/documentation/accessories/camera.html>. Acesso em: 09 de setembro de 2021. Citado 2 vezes nas páginas 15 e 16.

RASPBERRYPI. **Raspberry Pi Documentation+**. 2021. Disponível em: <https://www.raspberrypi.org/documentation/computers/os.html>. Acesso em: 13 de setembro de 2021. Citado na página 15.

RICHARDSON, M.; WALLACE, S. Primeiros passos com o raspberry pi. **Primeira Edição. Novatec Editora Ltda**, v. 20, 2013. Citado na página 14.

RUDEK, M.; COELHO, L. d. S.; JR, O. C. Visão computacional aplicada a sistemas produtivos: Fundamentos e estudo de caso. **XXI Encontro Nacional de Engenharia de Produção-2001, Salvador**, 2001. Citado na página 9.