



Universidade Federal do ABC
Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas
Engenharia de Instrumentação, Automação e Robótica

Sistema de Vigilância Autônomo em Tempo Real com Visão Computacional Baseada em *Deep Learning* para Detecção de Drones

**Barbara Souza
Wellington Souza Marques**

Santo André - SP, 06 Abril de 2020

Barbara Souza
Wellington Souza Marques

**Sistema de Vigilância Autônomo em Tempo Real com
Visão Computacional Baseada em *Deep Learning* para
Detecção de Drones**

Monografia apresentada a disciplina Trabalho de Graduação III do curso de Engenharia de Instrumentação, Automação e Robótica da Universidade Federal do ABC.

Universidade Federal do ABC - UFABC
Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas
Engenharia de Instrumentação, Automação e Robótica

Orientador: Luiz Antonio Celiberto Junior

Santo André - SP
06 Abril de 2020

Agradecimentos

Agradecemos primeiramente a Deus por nos dar forças e saúde para concluir este ciclo importante de nossos caminhos.

Agradecemos aos nossos pais pelo apoio e incentivo aos estudos durante toda a nossa trajetória acadêmica. Por nos motivarem em momentos difíceis e nos ensinar a perseverar para que pudéssemos concluir essa etapa de nossas vidas.

À Universidade pela oportunidade de nos desenvolver em um ambiente de excelência, além de prover todos os recursos necessários para elaborar este trabalho e por nos apresentar a relevância científica nos mais variados âmbitos.

Ao orientador pela grande solicitude em nos acompanhar e instruir durante todo o processo de execução deste projeto. E pela oportunidade de compartilhar e aprofundar conhecimentos essenciais para sua realização.

Aos amigos e parentes pela paciência diante de nossa ausência em períodos de dedicação exclusiva aos estudos e pela companhia nos momentos bons e ruins.

*“Não importa quanto a vida possa ser ruim,
sempre existe algo que você pode fazer, e triunfar.
Enquanto há vida, há esperança.”*
(Stephen Hawking)

Resumo

Este trabalho visa explicitar o desenvolvimento de um sistema de vigilância Anti-drone que opera em tempo real de forma autônoma, baseado no sensoriamento via mecanismos de recepção de áudio e imagens de câmera em vídeo, utilizando conceitos de visão computacional e aprendizado de máquina no reconhecimento de padrões que permitam a identificação do objeto de interesse. Sucintamente, o sistema posiciona a câmera em direção ao drone com o uso de um servo-motor através do processamento da informação recebida pelos microfones receptores, ações essas que são coordenadas por um microcontrolador. Com o objeto devidamente posicionado no foco da câmera, a identificação e o alarme de detecção acontecem com a execução de um algoritmo que reúne conceitos de visão computacional baseado em redes neurais. Uma interface também foi implementada para que o usuário seja capaz de operar o sistema de forma manual caso a situação demande, além de controlar a luz em ambientes de pouca luminosidade e poder acessar o histórico de detecção de drone gerado durante a execução do algoritmo de visão computacional.

Palavras-chaves: sistema autônomo; sistema em tempo real; aprendizado profundo; redes neurais artificiais; visão computacional; problema de classificação; segurança; drone; sistemas de vigilância; rastreamento por som.

Abstract

This paper aims to specify the anti-drone surveillance system development for real time and autonomous operation, based on sense devices for audio receiving and video capturing by camera use, applying computational vision concepts e machine learning on the pattern recognition for object identification. Briefly, the system is able to place the camera on the drone direction with the servo motor operation according to the information received by the installed microphones, these actions being coordinated by a microcontroller. With the object well placed on the camera focus, the identification and warning sound happen as the algorithm, that gathers computational vision based on neural networks, runs. A interface was implemented so the user is capable of operating the system manually if the situation requires, in addition to controlling the light bulb for dark environments and accessing the records of the drone detection during the algorithm run.

Keywords: autonomous system; real time systems; deep learning; artificial neural networks; computational vision; classification problem; security; drone; surveillance system; sound tracking

Lista de ilustrações

Figura 1 – Estrutura de um neurônio artificial (LEITE, 2018)	4
Figura 2 – Seleção da região de interesse (EE, 2020)	8
Figura 3 – Exemplo da região de sobreposição	11
Figura 4 – Exemplo da classificação (IGUINA, 2018)	11
Figura 5 – Matriz de Confusão	13
Figura 6 – Captação sonora com isolamento acústico parcial entre microfones	15
Figura 7 – 10000 passos: Valor de perda x Número de passos do conjunto teste	25
Figura 8 – 20000 passos: Valor de perda x Número de passos do conjunto teste	26
Figura 9 – 30000 passos: Valor de perda x Número de passos do conjunto teste	26
Figura 10 – 10000 passos: Valor de perda x Número de passos do conjunto treino	27
Figura 11 – 20000 passos: Valor de perda x Número de passos do conjunto treino	27
Figura 12 – 30000 passos: Valor de perda x Número de passos do conjunto treino	27
Figura 13 – Exemplo de Falso Negativo	28
Figura 14 – Exemplo de Verdadeiro Positivo	28
Figura 15 – Exemplo de Falso Positivo para região com similaridades visuais de um Drone	30
Figura 16 – Parte similar a região com detecção de Falso Positivo descrita na Figura 15	30
Figura 17 – Exemplo de detecção para Verdadeiro Positivo	32
Figura 18 – Exemplo de detecção para Verdadeiro Negativo	32
Figura 19 – Exemplo de detecção para Falso Positivo	32
Figura 20 – Montagem de fotos das várias visões da estrutura	34
Figura 21 – Exemplo de uma das medidas de distância máxima em ambiente escuro	35
Figura 22 – Exemplo de uma das medidas de distância mínima em ambiente claro	35
Figura 23 – Interface implementada	36
Figura 24 – Circuito	48
Figura 25 – Projeção Gráfica	53
Figura 26 – Vista explodida	53
Figura 27 – Circuito elétrico implementado	54
Figura 28 – Diagrama funcional	55

Lista de tabelas

Tabela 1 – Comparação de modelos	6
Tabela 2 – Matriz de confusão MobileNet 10000 passos	29
Tabela 3 – Matriz de confusão MobileNet 20000 passos	29
Tabela 4 – Matriz de confusão MobileNet 30000 passos	29
Tabela 5 – Métricas	30
Tabela 6 – Frames por segundo	31
Tabela 7 – Atributos	33
Tabela 8 – Distâncias mensuradas em ambientes com ou sem luminosidade	35
Tabela 9 – Valor de Perda para 10000 passos	45
Tabela 10 – Valor de Perda para 20000 passos	45
Tabela 11 – Valor de Perda para 30000 passos	46
Tabela 12 – Matriz de Confusão SSD Mobilenet 10000 passos	47
Tabela 13 – Matriz de Confusão SSD Mobilenet 20000 passos	48
Tabela 14 – Matriz de Confusão SSD Mobilenet 30000 passos	48
Tabela 15 – Resultados do Faster RCNN 20000 passos para cálculo de frames por segundo	50
Tabela 16 – Resultados do Mobilenet 10000 passos para cálculo de frames por segundo	51
Tabela 17 – Resultados do Mobilenet 20000 passos para cálculo de frames por segundo	51
Tabela 18 – Resultados do Mobilenet 30000 passos para cálculo de frames por segundo	52

Lista de abreviaturas e siglas

COCO	Common Objects in Context
CPU	Central Process Unit
CSV	Comma Separated Values
FN	Falso Negativo
FP	Falso Positivo
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IoU	Intersection Over Union
ILD	Interaural Level Difference
ILP	Instruction Level parallelism
JPEG	Joint Photographic Experts Group
LDR	Light Dependent Resistor
mAP	mean Average Precision
MLP	Multilayer Perceptron
PU	Poliuretano
R-CNN	Region-Based Convolutional Neural Networks
SSD	Single Shot MultiBox Detector
VP	Verdadeiro Positivo
VN	Verdadeiro Negativo
XML	Extensible Markup Language

Sumário

Introdução	1
Objetivos	2
1 PRESSUPOSTOS TEÓRICOS	3
1.1 Sistemas de tempo real	3
1.1.1 Classificação de sistemas de tempo real	3
1.2 Redes Neurais artificiais	3
1.2.1 Redes Mobilenet	5
1.2.2 Rede neurais recorrentes: Faster RCNN	6
1.2.3 Comparação entre modelos de redes neurais	6
1.3 Framework TensorFlow	7
1.4 Classificação de imagens com TensorFlow	7
1.4.1 Pré-processamento	7
1.4.2 Transferência de aprendizado no treinamento	9
1.4.3 Treinamento do modelo de classificação de imagens	10
1.4.4 Uso de GPUs na etapa de treinamento	12
1.4.5 Aplicação do modelo em tempo real	12
1.4.6 Métricas de desempenho	12
1.5 Sistemas complementares para detecção	14
2 METODOLOGIA	17
2.1 Pré-processamento dos dados	17
2.2 Treinamento	18
2.2.1 Desenvolvimento do treinamento	19
2.2.2 Definição do melhor modelo	20
2.2.3 Métricas do treinamento	21
2.2.4 Métricas físicas	21
2.2.5 Métricas do teste com o dataset apartado	21
2.3 Sistema de rastreamento sonoro	22
2.4 Sistema de iluminação automática	22
2.5 Interface gráfica	23
2.6 Estrutura	23
2.7 Integração dos sistemas	24
2.8 Execução em tempo real	24
3 RESULTADOS E DISCUSSÃO	25

3.1	Resultado e discussão das métricas para algoritmo de detecção	25
3.1.1	Métricas no treinamento e validação	25
3.1.2	Métricas com conjunto de imagens para teste	27
3.1.3	Desempenho do sistema de identificação em termos de velocidade de vídeo	31
3.1.4	Dados obtidos com o modelo executado	31
3.2	Resultados e discussão do sistema em função do rastreamento sonoro	33
3.2.1	Algoritmo de movimentação baseado na intensidade sonora	33
3.2.2	Estrutura física	33
3.3	Resultados e discussão do sistema em função da luz ambiente	34
table	34
3.4	Resultados e discussão da interface gráfica	36
	Conclusão e perspectivas	37
	REFERÊNCIAS	39
	APÊNDICES	43
	APÊNDICE A – TABELAS E CÁLCULOS	45
A.1	Tabelas para construção do gráfico de treino	45
A.2	Cálculos das métricas com os dados de teste	46
A.2.1	Modelo SSD Mobilenet 10000 steps	46
A.2.2	Modelo SSD Mobilenet 20000 steps	46
A.2.3	Modelo SSD Mobilenet 30000 steps	47
A.3	Tabelas para construção da matriz de confusão	47
A.4	Cálculo do resistor R1 na base do transistor TIP120	48
A.5	Tabelas para cálculos de taxa de frames	49
	APÊNDICE B – DIAGRAMAS	53
B.1	Modelo mecânico do rastreador no SolidWorks	53
B.2	Círculo elétrico do sistema	54
B.3	Diagrama de fluxo funcional de todo o sistema	55
	APÊNDICE C – CÓDIGOS	57
C.1	Código da Interface implementado em Python	57
C.2	Código do Rastreador de som implementado em Arduino	59
	ANEXOS	65
	ANEXO A – CÓDIGOS	67

A.1	Código do Treinamento	67
A.2	Código da Detecção	74

Introdução

Apesar do avanço tecnológico observado nos últimos anos ser algo positivo, dadas as facilidades advindas da popularização da Internet e do desenvolvimento e aprimoramento de outras ferramentas, questões de cunho ético têm sido levantadas. Em termos da informação, por um lado se tem o aumento de sua quantidade e qualidade, enquanto no outro problemas de segurança tem emergido, visto que com todo o progresso também surgiram novas formas de como não apenas reforçá-la, mas também de como burlá-la. Nesse sentido, novas tecnologias apesar de facilitarem diversas atividades atualmente, quando utilizadas de forma errada podem vir a gerar situações problemáticas e desconfortáveis sendo vazamento de informações pessoais e quebra de privacidade apenas alguns dos exemplos.

Drones são pequenos “ícones” tecnológicos. Futuristas, os dispositivos trazem facilidades para seus usuários. Tido como *hobby* para alguns, para outros já estão sendo utilizados para entregas, para acesso a áreas remotas ou até servem como protótipo para aeronaves não tripuladas. ([PICCOLOTTO, 2020](#)) Apesar de sua criação gerar facilidades não antes imaginadas, problemas com a segurança também são apontados: possibilidade de espionagem ou até mesmo contrabando e acesso a áreas de segurança, tais como bases militares e aeroportos, são agora pontos de atenção e de estudo dada a nova e desconhecida demanda. ([GRANEMANN, 2019](#))

O desenvolvimento de sistemas capazes de controlar usos indevidos dessa tecnologia se faz necessário e, apesar de alguns avanços, produtos e tecnologias para detecção de drones precisam ser melhorados. Radares militares podem ser eficientes, mas não são convenientes ao uso urbano, em menor escala, por exemplo. Além disso, o objeto em questão a ser detectado possui algumas peculiaridades, tais como aparência facilmente confundida com a de outro objeto voador, baixa altitude de vôo e movimento imprevisível, o que acaba por dificultar seu rastreamento. ([FARINACCIO, 2017](#))

Em paralelo tem-se uma crescente onda de estudo e aperfeiçoamento de técnicas voltadas aos métodos de *deep learning*, ou aprendizado profundo, assim como o aumento do poder computacional, opções de tratamento volume de dados e grandes comunidades, permitindo maior troca de informações e técnicas. ([DINO, 2019](#)) Nesse âmbito, tem-se o campo de visão computacional, cujo primeiro objetivo é entender cenas visuais, reconhecendo objetos, atributos e reconhecimento de interações entre os mesmos.

A proposta deste trabalho advém da união da ferramenta de *deep learning* e a necessidade explicitada de reconhecer drones de modo a garantir a segurança em áreas ou situações nas quais é tido como um risco. Com a câmera posicionada na direção do mesmo, seja isso de forma automática ou manual operado por um usuário, utiliza-se conceitos de

visão computacional para detectá-lo, através de suas principais características físicas e não do seu padrão de movimento, o que poderia fazer com que fosse confundido com outros objetos voadores ou animais.

O objetivo principal consiste na geração de um sistema único operante, envolvendo uma interface amigável ao usuário, reconhecimento de drone por visão computacional, detecção por som para identificar a posição do drone no raio de 180°, iluminação para ambientes escuros, alarme sonoro, geração de relatório e estrutura para permitir o devido acoplamento aos subsistemas e movimentação.

A seção 2 apresenta os objetivos deste Trabalho de Graduação, sendo explicitados em principal e secundários. A seção 3 discorre a respeito da fundamentação teórica acerca, principalmente, de conceitos de *deep learning* e redes neurais, além de informar especificidades teóricas acerca de temas envolvendo, por exemplo, a execução dos scripts. A seção 4 informa a metodologia seguida para obter cada subsistema, enquanto a seção 5 indica os resultados obtidos e os pontos de melhoria encontrados. Por fim, tem-se na seção 6 a conclusão deste trabalho.

Objetivos

O presente Trabalho de Graduação tem como principal objetivo explicitar o desenvolvimento do sistema de vigilância anti-drone, capaz de detectar o objeto voador em questão através do uso de visão computacional fundamentada em ferramentas de *deep learning*, tendo como produto final uma ferramenta de segurança portátil e capaz de vigiar pequenas áreas especificadas.

Como objetivos secundários, passos esses que foram necessários para o projeto como um todo funcionar como um sistema único, tem-se: a criação de uma interface que permite a interação do usuário, possibilitando o poder de escolha e adaptação aos mais variados tipos de situações que coloquem a segurança em risco; localização do drone por quadrante através do uso de microfones para percepção sonora e movimentação automática do sistema; ativação do sistema de luminosidade para ambientes noturnos ou de pouca luminosidade; criação de uma estrutura capaz de suportar os sistemas embarcados e que permita a movimentação conforme desejado.

1 Pressupostos teóricos

1.1 Sistemas de tempo real

Este Trabalho de Graduação tem como principal objeto de estudo aprendizado de máquina ou *machine learning*, dentro da área de aprendizado profundo, utilizado para detecção em tempo real de drones, que são objetos voadores não tripulados. Tal tipo de sistema pode ser descrito como aquele que "controla um ambiente recebendo dados, processando-os e retornando os resultados com rapidez suficiente para afetar o ambiente naquele momento". ([MARTIN, 1965](#)) Genericamente, é um sistema cujo tempo de resposta é um fator determinante para seu devido funcionamento ([KRISHNA, 2001](#)): quanto menor for o tempo de resposta de um sistema, maior será a agilidade em tomadas de decisão e maior efetividade no controle da situação problemática que estiver sendo monitorada. O sistema de vigilância implementado neste projeto exige restrições de tempo visto que possui segurança como finalidade. Dado sua característica didática e sua escala (sendo apenas um protótipo), o desdobramento desse projeto não foi rígido nesses termos.

1.1.1 Classificação de sistemas de tempo real

Sistemas de tempo real podem ser classificados em relação ao quanto importante a atividade executada é. Dado o propósito do sistema aqui descrito (sendo apenas um protótipo), a classificação no tocante do tempo é tida como *soft real time*, o que significa que este pode ter uma resposta lenta, mesmo que isso seja indesejado, pois não ameaça algum fator importante.

Um sistema classificado com *hard real time*, por sua vez, acontece quando tal lentidão na resposta colocaria vidas em risco, por exemplo, ou causaria um dano muito grande que também seria inaceitável. Pode-se citar como um exemplo o próprio sistema de vigilância aqui construído, se caso fosse um sistema real e não para fins acadêmicos, em proporções maiores, como no caso de um sistema militar. ([SEKHAR, 2013](#))

1.2 Redes Neurais artificiais

O processamento das informações é feito por uma rede neural artificial. Redes neurais artificiais nada mais são do que um modelo computacional de um neurônio vivo, presente no sistema nervoso dos animais. É o campo de estudo em que os computadores passam a ter a habilidade de aprender sem serem explicitamente programados. ([GRÜBLER, 2018](#)), [Busson et al. \(2018, 3.3\)](#)

Redes neurais são o método mais moderno de aprendizado de máquina. Até 2006, entretanto, não era possível treinar redes neurais para superar as técnicas que até então eram tradicionais, tais como a árvore de decisão, sendo somente utilizadas com especificidade. Tal quadro foi modificado com o advento do aprendizado profundo, ou *deep learning* como será referenciado neste texto, permitindo modelagem em redes neurais profundas. Ao se tratar de classificação de imagens a fins de visão computacional, redes neurais profundas é a principal ferramenta a se considerar. [Busson et al. \(2018, 3.4\)](#)

Para entender o funcionamento de uma rede neural, é necessário iniciar pela estrutura básica da mesma: o *Perceptron*. Este é o tipo mais básico de neurônio, onde os sinais de entrada são definidos por um vetor x que, por sua vez, são correlacionados a um vetor de pesos na estrutura neural, de modo a gerar o que é chamado de valor de ativação, determinado pelo produto escalar desses vetores. Esse valor passa por uma função matemática de ativação, podendo essa ser ou não linear, produzindo uma saída. [Busson et al. \(2018, 3.4.1\)](#)

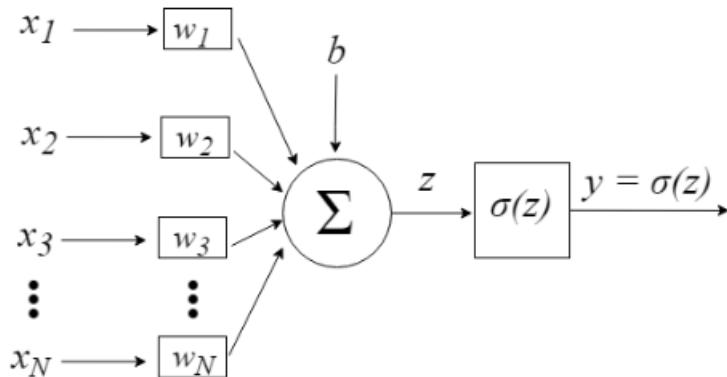


Figura 1 – Estrutura de um neurônio artificial ([LEITE, 2018](#))

Assim como nos próprios seres vivos, apenas um neurônio não é suficiente para que o sistema funcione adequadamente. Existem variadas soluções de modelagem para uma atividade como a classificação, sendo a *Perceptron Multi Classe* uma delas. Nela, múltiplos perceptrons são necessários, sendo organizados de forma paralela e a cada um deles é designado o ativamento de uma classe específica que ocorre quando há maior saída de ativação em um dos neurônios.

Já a modelagem denominada *Perceptron Multicamadas* (MLP, ou *Multi Layer Perceptron*) organiza os neurônios em múltiplas camadas, onde a saída de um neurônio é conectada na entrada de outro nas camadas intermediárias da estrutura. A saída será gerada na última camada, chamada de camada de saída e quanto mais arranjada a rede, mais profundas e complexas as conexões. A informação é aplicada à uma hierarquia estruturada de transformações, sejam elas lineares ou não. Uma rede é considerada profunda quando

possui mais de duas camadas. [Leite \(2018, 3\)](#)

O aprendizado acontece pelo algoritmo chamado retropropagação ou *backpropagation*, no qual os pesos e uma informação interna ao neurônio (chamada *bias*) e são ajustados para aproximar a saída da rede para toda entrada que foi computada no treinamento. A diferença é quantificada através de funções de custo, enquanto o treinamento é definido ao se rotular o material de ensino de modo a classificá-lo. [Busson et al. \(2018, 3.4.2\)](#), [Leite \(2018, 3\)](#)

Neste projeto, por exemplo, através de um *dataset* (ou base) com 270 imagens de drone, rotulou-se uma a uma qual região da imagem correspondia ao objeto através do uso do LabelImg, um *software* gratuito e exclusivo para tal tarefa. Através do processo de convolução de imagens em conjunto com o algoritmo de retropropagação, a rede passa a ser capaz de entender o que é objeto de interesse (chamado classe nesse escopo).

Dada essa característica de fornecer dados à rede, o projeto está alinhado ao que se é definido por aprendizado supervisionado, uma das três categorias que o aprendizado de máquina: supervisionado, não-supervisionado ou por reforço. Sendo que no primeiro, amostras pré-classificadas são mostradas à máquina, mapeando uma entrada para um tipo de saída. [Busson et al. \(2018, 3.3\)](#)

Quanto ao citado *script* de retropropagação, pode-se definir cinco passos: [Busson et al. \(2018, 3.4.2\)](#)

1. Entrada: definição do vetor x
2. Propagação: cálculo da ativação dos neurônios com base na ativação dos neurônios da camada anterior
3. Cálculo de erro
4. Retropropagação do Erro
5. Atualização dos pesos e *bias*

1.2.1 Redes Mobilenet

Mobilenets são modelos *open-source* de alta eficiência para problemas que envolvam visão computacional. Projetados para maximizar a acurácia enquanto levam em consideração a restrição dos equipamentos e as aplicações embarcadas, *Mobilenets* fazem parte da primeira família de modelos de visão computacional para *TensorFlow*. Pequenos, de baixa latência e baixo consumo de energia, são modelos ideais para recursos limitados, tais como os desse projeto. ([HOWARD et al., 2017](#)), ([DEORA, 2018](#))

1.2.2 Rede neurais recorrentes: Faster RCNN

RCNN foi o início de alguns modelos para a detecção de objetos via técnicas de *deep learning*. Definida por *Region-based Convolutional Neural Network* ou Rede Neural Convolucional baseada em Regiões, RCNN se baseia em três passos: escaneamento de imagens de entrada, execução da rede neural convolucional em cada região e, por fim, considera a saída para alimentar o classificador e um regressor linear para estreitar (isso em termos de precisão) a caixa que identifica o objeto.

Em resumo, o RCNN transforma o problema de detecção em um problema de classificação. Quanto ao *Fast* RCNN, este possui várias semelhanças quando comparado ao seu antecessor, mas ganha em termos de velocidade, fazendo extração das imagens antes de propor regiões, além de estender a rede neural ao invés de gerar um novo modelo. Por fim, o *Faster R-CNN* substitui o algoritmo padrão de busca por uma rede neural do tipo rápida. Em linhas gerais, pode-se dizer este combina os dois métodos anteriores. ([XU, 2017](#))

1.2.3 Comparação entre modelos de redes neurais

As informações disponibilizadas pelos desenvolvedores referentes aos valores de desempenho, precisão média e tipo de saída nos modelos considerados podem ser encontrados na tabela 1, considerando que estes foram avaliados em termos de aplicação no mesmo dataset. ([SANDLER, 2019](#))

Tabela 1 – Comparação de modelos

Nome do modelo	Velocidade (ms)	COCO mAP	Tipo de saída
ssd_mobilenet_v2_coco	30	21	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes

Onde mAP é uma medida de desempenho típica usada para classificar modelos, corresponde ao valor da precisão média. ([HUI, 2018](#)) A velocidade é associada ao tempo em milissegundos que o modelo leva para ser executado na detecção efetuada em uma imagem e o tipo de saída é feita através do desenho de caixas com informação do *score* ao redor do objeto de classificação na imagem.

Estas características apresentadas no gráfico servem de apoio para a escolha do modelo pré-treinado mais adequado para as particularidades do projeto. O modelo *SSD Mobilenet* apresenta características importantes para este escopo, com velocidade superior, definida como menor tempo em milissegundos para sua execução.

Porém, como será abordado mais adiante, métricas são estabelecidas para que outros aspectos possam ser considerados e tornar a atividade de escolher entre os dois modelos mais certeira à finalidade proposta pelo sistema de vigilância implementado.

1.3 Framework TensorFlow

As redes neurais foram grandes responsáveis pelo avanço que a área de *machine learning* teve nos últimos anos, constituindo uma posição majoritária nessas questões. O *TensorFlow* é um *framework open source* que torna a atividade de criação de redes neurais mais fácil e é o responsável por encapsular o algoritmo de retropropagação, não sendo necessário realizar a tarefa de implementação do mesmo. ([PATNAIK, 2018](#))

O *TensorFlow* opera criando o que é conhecido como grafo computacional, onde são especificados operações a serem executadas. Seu nome deriva das operações que as redes neurais desempenham em vetores de dados multidimensionais, conhecidos como tensores, e para utilizá-lo é necessário entender alguns conceitos básicos: [Busson et al. \(2018, 3.2.2\)](#)

- Tensor: consiste em um vetor de n dimensões, sendo a estrutura básica de dados constituinte do *TensorFlow*.
- Grafo de computação: são redes compostas por nós conectados por arestas. As arestas possuem pesos, valores, que são passados de um nó para outro. O nó só será capaz de realizar tal operação ao receber todos os inputs necessários e, então, passará sua saída para outro nó que estiver em sua vizinhança.
- Sessões: os nós podem ser separados por sessões.

1.4 Classificação de imagens com TensorFlow

1.4.1 Pré-processamento

As técnicas de pré-processamento de dados são aplicadas com frequência com o intuito de melhorar a qualidade do dados para tornar mais adequada sua utilização no algoritmo de aprendizado de máquina. Esta melhora pode ser obtida através da eliminação ou diminuição de problemas como: ruídos, valores incorretos, dados inconsistentes, divergência nas características, dimensões e formatos. Através deste processo é possível construir modelos mais fiéis a distribuição de dados, além de tornar mais fácil a interpretação de padrões extraídos pelo modelo durante o treinamento. ([FACELI et al., 2011](#))

A seleção das imagens é parte importante desta etapa. Em concordância com o objetivo deste projeto, os principais atributos utilizados para a criação do modelo são as informações de dimensão, contornos e formato do drone contidas nas imagens selecionadas.

Esta etapa é definida por uma série de procedimentos. De início, imagens do objeto a ser identificado devem ser coletadas para a criação de um *dataset*, sendo considerado

uma boa quantidade mínima o valor de 50 fotos para cada classe que será considerada e salvando-as no formato JPEG com tamanho de 600x800 através do uso da biblioteca OpenCV do Python. ([SINJAB, 2019](#))

Para o *framework* utilizado, as imagens do dataset devem atender alguns requisitos quanto à sua qualidade. Um bom conjunto de imagens para a aplicação é constituído pela presença de imagens “não perfeitas”, ou seja, com o fundo contendo outros objetos, tipos variados de iluminação, algumas com o objeto a ser detectado aparecendo parcialmente ou sendo sobreposto por outros objetos. ([BALSYS, 2018](#))

Com tais imagens em mãos, deve-se anotar as informações da região de interesse para o aprendizado em cada uma delas. Isso significa que o processo de identificar o drone é feito manualmente em todas as imagens, caracterizando o que é chamado como aprendizado de máquina supervisionado. Aqui, o objetivo é aprender uma função que mapeia a entrada para um tipo de saída, conforme explicitado na referência [Busson et al. \(2018, 3.3\)](#).

A figura [2](#) mostra como é selecionado a região de interesse com o drone através do aplicativo LabelImg:

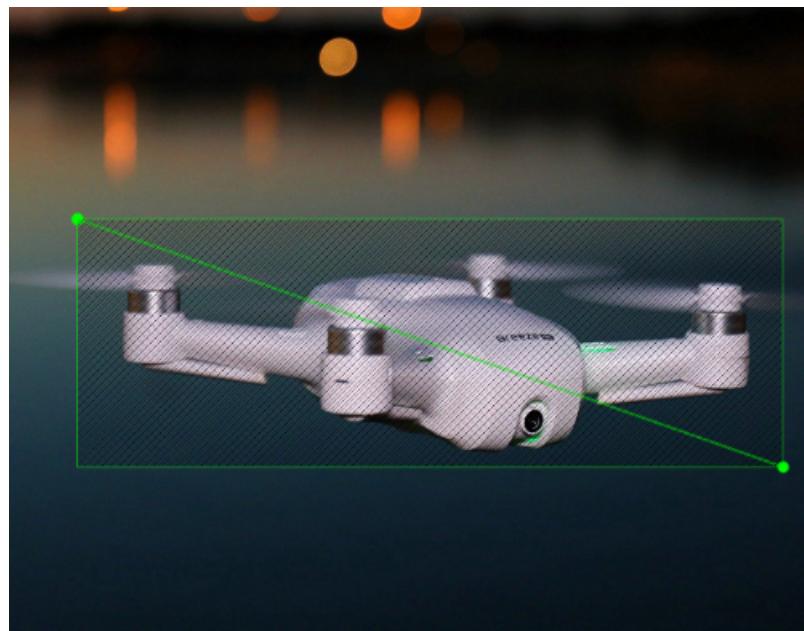


Figura 2 – Seleção da região de interesse ([EE, 2020](#))

Após a execução de tal procedimento, arquivos de formato XML são gerados, possuindo o mesmo nome da imagem que lhe é correspondente, sendo separados em teste e treinamento em determinada proporção para que um *script* seja executado e gere dois arquivos do tipo CSV. Estes, por sua vez, contém informações relevantes, tais como o próprio tamanho das imagens, classe e a posição da classificação realizada.

Em seguida, um outro *script* é considerado, no qual deve-se especificar os rótulos

de cada classe (que no caso é apenas drone), e que por fim irá gerar um arquivo do tipo TFRecords passível de leitura do *framework TensorFlow*, utilizado no treinamento, junto com um arquivo PBTXT que serve para indicar os rótulos (classes).

1.4.2 Transferência de aprendizado no treinamento

Um modelo pré-treinado define-se por um treinamento prévio em um grande conjunto de dados de referência e que pode ser utilizado para resolver algum problema semelhante em aplicações com outros conjuntos de dados. A similaridade pode ser no tipo de *dataset* utilizado ou em sua aplicação, sendo alguns exemplos práticos a detecção de objetos, o reconhecimento de voz ou o reconhecimento de linguagem natural.

O modelo pré-treinado pode ser utilizado das seguintes formas ([ROSE, 2019](#)):

- Uso do modelo para a extração de atributos (*feature extraction*)
- Uso da arquitetura da rede: Para isso deve-se inicializar todos os pesos de forma aleatória para então treinar o modelo de acordo com o dataset
- Uso de forma parcial: Realiza-se o congelamento dos pesos anteriores e a aplicação do treinamento com o *dataset* escolhido em camadas superiores

Para se fazer a alteração nos modelos pré-treinados, utiliza-se recursos de sintonia fina (*finetuning*), ajustando seus parâmetros para se adequar a um novo modelo.

A extração de atributos envolve a redução do número de recursos necessários para descrever uma grande quantidade de dados, além disso também cria novos atributos a partir dos originais. Este processo é de grande importância para o processamento multimídia, pois estes são capazes de resumir as principais informações contidas no conjunto original. ([IPPOLITO, 2019](#))

No campo de imagens, os atributos de grande importância para a extração, são eles: a cor, textura e formato. Esta última pode ser dividida entre duas partes: uma é extração de atributos somente da borda e outra é a extração de atributos pela região de interesse. ([TIAN et al., 2013](#)) Redes Neurais Convolucionais são especializadas em processar e aplicar modelos à um conjunto de imagens, que podem ser organizados em grades. A convolução consiste em gerar uma função a partir de outra através da aplicação de um operador linear, o que acaba por definir uma soma de produtos das funções ao longo da região subentendida pela superposição delas. Esse cálculo é capaz de explicitar as características de alto e baixo nível e propiciar melhor classificação.

1.4.3 Treinamento do modelo de classificação de imagens

O treinamento de um modelo de classificação em uma rede neural, consiste em aplicar um algoritmo de ajuste de pesos baseado na relação entre o valor predito e o valor real esperado. A função deste treinamento é encontrar os parâmetros da rede neural na condição ótima para realizar a classificação adequada, esta condição ótima pode ser encontrada quando a função que rege o treinamento, chamada de Função do Gradiente Descendente converge globalmente.

No caso de problemas que envolvem o uso de aprendizado profundo, não é possível calcular diretamente estes pesos para a convergência global ou então obter uma garantia para encontrar o conjunto de pesos corretos, pois trata-se de um problema de otimização de uma função não convexa que possui mínimos e máximos locais. ([BROWNLEE, 2019b](#))

O algoritmo de treinamento chamado *backpropagation* é amplamente utilizado em problemas que envolvem redes neurais, estes são capazes de ajustar os pesos das redes neurais através de execução, em forma de loop, de um conjunto completo de iterações chamadas de épocas. Para cada época do treinamento, o algoritmo de treinamento faz a predição, mede o erro, e em seguida vai para cada layer anterior e mede a contribuição de cada conexão e finalmente ajusta os pesos para reduzir o erro da função de custo. ([GÉRON, 2019](#))

Para o caso de problemas de classificação binária, o valor de erro (*loss value*) da função de custo pode ser obtido através de relação logarítmica entre o valor predito e o valor real correspondente a uma classe. Este valor quantifica a diferença entre duas distribuições de probabilidade, seu funcionamento dentro do treinamento se baseia na penalização de classificações falsas. A equação que rege esta relação é mostrada a seguir:

$$\text{Cross Entropy} = \text{Perda logaritmica} = - \sum_i^M y_{real} \log(y_{pred}) \quad (1.1)$$

Esta função de custo é aplicada em caso de classificação binária, ocorrendo apenas para uma classe. A entropia cruzada informa a diferença entre duas distribuições de probabilidade, sendo y_{real} a real esperada e y_{pred} a predita, num mesmo conjunto de eventos. ([BROWNLEE, 2019a](#))

Este é o tipo de classificação aplicado neste projeto, onde a única classe encontrada é o drone. Embora valores de detecção do drone sejam dados em uma probabilidade traduzida na informação de porcentagem em “pontuação” (*score*), ainda assim é possível considerá-lo binário ao se ter pontos de gatilhos (*thresholds*) para determinar uma classificação com falsa ou verdadeira para uma classe.

O valor obtido pela função *cross entropy* [1.1](#) pode servir de referência para medir o quanto a probabilidade de uma classe estimada corresponder a classe de destino (*target*).

Dentro do campo de classificação de imagens, adota-se como predição correta para detecções de verdadeiros positivos quando a área de interseção entre a área predita e a área que deveria ser predita (*IoU - Intersection Over Union*) é maior ou igual a 0,5.

Como forma de exemplo, dado duas regiões, onde **A** é a região a ser predita e **B** corresponde à região predita, a relação de IoU com A e B é descrita pela equação 1.2:

$$IoU(A, B) = \frac{A \cap B}{A \cup B} \quad (1.2)$$

A imagem xx exemplifica visualmente como é definida esta métrica:

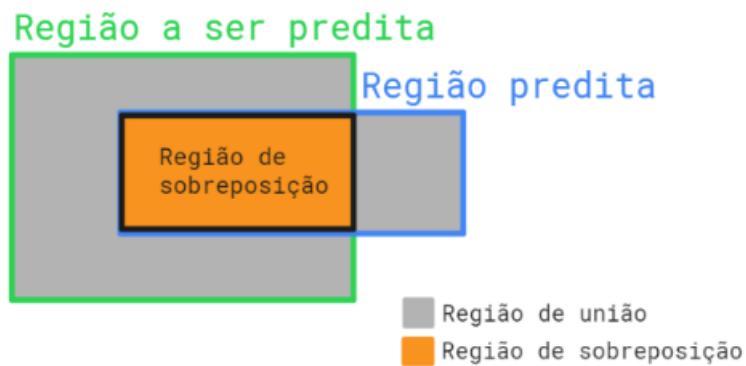


Figura 3 – Exemplo da região de sobreposição

Como exemplo para a classificação do drone, é mostrado na figura 4 a região de seleção a direita correspondente a região a ser predita e a esquerda a região predita pelo modelo durante a etapa de treinamento.

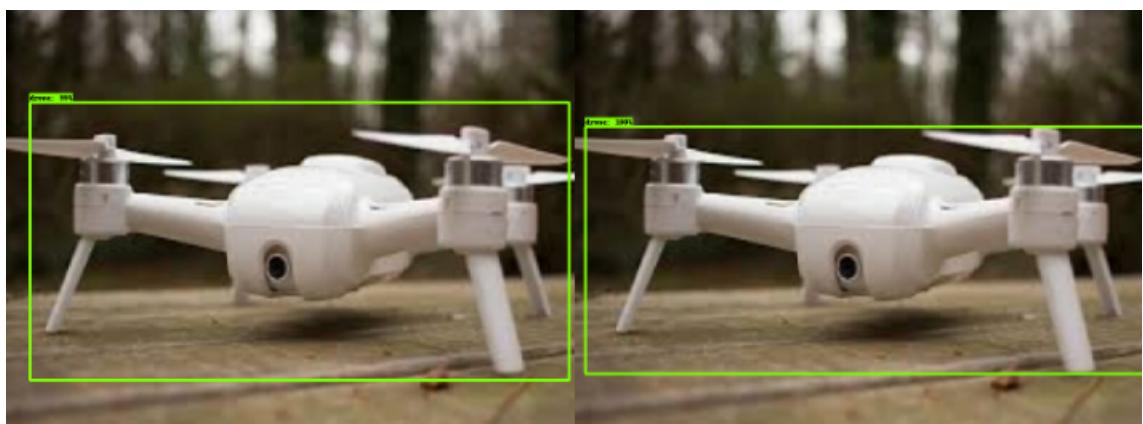


Figura 4 – Exemplo da classificação (IGUINA, 2018)

Ao término do treinamento, utilizando o *framework TensorFlow*, é gerado um arquivo chamado *frozen inference graph* de formato PB e *label map* que possui todos as informações do treinamento em um formato que pode ser executado em qualquer máquina que possua o referente framework.

1.4.4 Uso de GPUs na etapa de treinamento

GPUs (*Graphics Processing Units* ou Unidades Gráficas de Processamento) são processadores especializados originalmente criados para realizar tarefas de computação gráfica compostos por vários processadores simples, chamados de *core*, que facilitam a execução de algoritmos dado sua alta capacidade de gerar paralelismos. (KANSAL, 2018)

GPUs são necessárias para o treinamento de modelos *deep learning*, pois sem elas, utilizando apenas um notebook ou computador pessoal, os treinamentos podem levar dias, visto que a quantidade de tempo para executar uma tarefa é inversamente proporcional ao tamanho do *hardware* disponível. Dado seu alto grau de paralelismo, a tarefa de execução se torna algo simples.

1.4.5 Aplicação do modelo em tempo real

O arquivo final que é produto do treinamento, chamado de grafo de inferência, deve ser aplicado à uma execução em tempo real com a biblioteca *OpenCV* para Python que oferece a capacidade de capturar imagem de câmera de vídeo em *loop* para que o modelo processado seja aplicado aos *frames*.

Além de permitir a validação do modelo visualmente através da saída de vídeo, também é possível captar a velocidade dos *frames* através de uma variável contadora e outra variável de tempo e calcular a velocidade do modelo.

1.4.6 Métricas de desempenho

Em aprendizado de máquinas, a medição de qualidade do modelo é uma parte importante do seu processo de criação. A escolha da métrica mais adequada para avaliá-lo depende do tipo de dados utilizados e dos objetivos a serem alcançados. Neste contexto pode ser levado em consideração se a classificação é binária ou não.

Uma métrica amplamente utilizada para avaliar diversos modelo é a matriz de confusão. (MISHRA, 2018) Esta matriz descreve de forma completa o desempenho de um modelo, pois ela distingue os resultados em quatro classes:

- **Verdadeiro Positivo (VP):** Caso onde a classe foi identificada como Verdadeiro e realmente é esperado verdadeira.
- **Verdadeiro Negativo (VN):** Caso onde a classe é identificada como Negativo e realmente é esperado Negativo.
- **Falso Positivo (FP):** Caso onde a classe é identificada como Positivo, porém era esperado é Falso.

- **Falso Negativo (FN):** Caso onde a classe é identificada como Negativo, porém o esperado é Positivo.

Esta informações são compiladas em uma tabela no formato descrito na figura 5:

Matriz de Confusão		Valor verdadeiro (confirmado por análise)	
		Positivos	Negativos
Valor Previsto (Preditto pelo teste)	Positivos	VP Verdadeiro Positivo	FP Falso Positivo
	Negativos	FN Falso Negativo	VN Verdadeiro Negativo

Figura 5 – Matriz de Confusão

A matriz de confusão pode fornecer informações importantes acerca do modelo. Ela também possibilita encontrar valores de outras métricas importantes e concisas para a avaliação do modelo (GÉRON, 2019), são elas:

Acurácia: Por definição, significa exatidão e rigor. Com os dados obtidos pela fornecidos pela matriz de confusão, ela traduz esta informação de exatidão e rigor do modelo através a razão entre o número de previsões corretas e o número total de amostras entradas. Esta informação é obtida através da equação 1.3:

$$\text{Acurácia} = \frac{VP + VN}{\sum_{n=1}^{20} VP_n + VN_n} \quad (1.3)$$

Precisão: É o número de resultados positivos previstos de forma correta divididos pelo número de resultados positivos previstos pelo classificador. Ela responde a pergunta: Daqueles que foram classificados com corretos, quantos efetivamente eram? Esta informação é traduzida através da equação 1.4:

$$\text{Precisão} = \frac{VP}{VP + FP} \quad (1.4)$$

Revocação (Recall): Métrica denota a sensibilidade do modelo, também denotada como taxa de verdadeiros positivos. Esta métrica pode responder a seguinte pergunta: Que proporção de positivos reais foi identificada corretamente? Matematicamente é definida conforme a equação 1.5:

$$\text{Revocação} = \frac{VP}{VP + VN} \quad (1.5)$$

F1 Score: É a média harmônica entre Precisão e Revocação, possui grande utilidade para se medir a qualidade geral do seu modelo. De forma geral, o modelo é melhor quanto maior for o F1-Score. Esta combinação entre a precisão e o revocação é descrita matematicamente através a da equação 1.6 que se segue:

$$F1\ Score = \frac{2 \cdot \text{Precisão} \cdot \text{Revocação}}{\text{Precisão} + \text{Revocação}} \quad (1.6)$$

Além das métricas aqui descritas, também pode se considerar o valor de perda como uma métrica importante para avaliar a qualidade do modelo. Durante o treinamento, com o uso do *TensorFlow*, esta métrica é informada a cada conjunto épocas executados, desta forma torna-se útil para a avaliação do modelo durante o seu treinamento. As informações referentes a esta métricas são descritas na seção 1.4.1 sobre treinamento de modelo.

1.5 Sistemas complementares para detecção

O sistema de visão computacional pode estar integrado a outros subsistemas que interagem com o mundo físico e com o usuário no intuito de funcionar de forma mais eficaz. No contexto apresentado aqui, este é incorporado à uma plataforma capaz de sustentar os dispositivos eletrônicos e mecânicos concernentes a estes subsistemas. Em relação à estrutura, além de atender os requisitos de tamanho e peso, esta também deve ser capaz de promover a movimentação da câmera, aumentando o campo de visão da mesma.

Seu posicionamento é ajustado o de acordo com a diferença de intensidade sonora recebida por dois microfones isolados acusticamente. O som é emitido pelo objeto em determinada posição e o movimento ocorre dentro de um raio de 180°, sendo este posicionamento decorrente da operação conjunta de um microcontrolador, dois microfones e um servo-motor. A figura 6 ilustra o mecanismo de captação sonora neste conjunto.

Este aparato se baseia no funcionamento natural biológico para localização do som baseado na diferença de nível do som (ILD - *Interaural Level Difference*). Nossos ouvidos podem detectar diferenças de volume entre os ouvidos esquerdo e direito, o que contribui para nossa percepção da direção auditiva. (HARTMANN; CONSTAN, 2002)

Os dispositivos englobados nesta plataforma consistem de um microcontrolador para controle e interfaceamento com a GUI (Interface Gráfica do Utilizador), uma lâmpada para iluminação direta em ambiente escuro, sensor de luz baseado em LDR (Resistor Dependente da Luz), servo-motor para girar a parte móvel da estrutura no rastreamento, microfones para captura de som e câmera para captura de vídeo.

A interface gráfica permite ao usuário interagir com os dispositivos, digitais e físicos envolvidos no sistema, por meio de elementos gráficos. Através dela é possível selecionar o

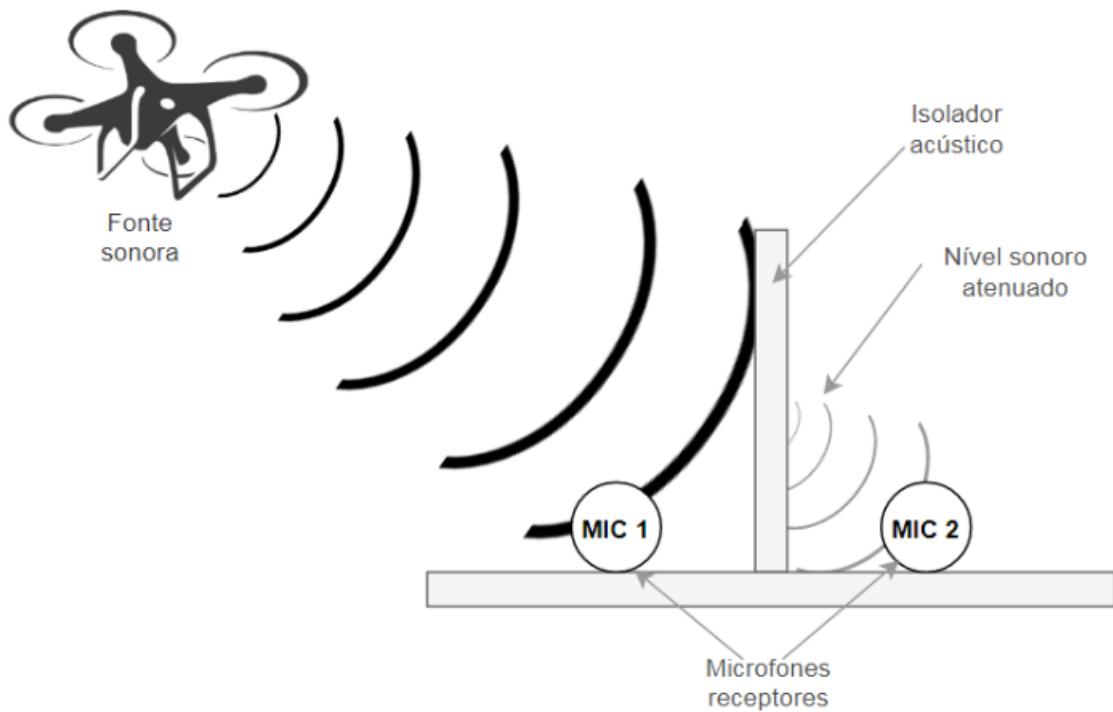


Figura 6 – Captação sonora com isolamento acústico parcial entre microfones

modo de operação, o controle da movimentação, o acionamento da câmera e a geração de relatórios.

A iluminação é um subsistema necessário para melhorar a qualidade da imagem em situações noturnas ou de pouca luminosidade, dado o local de instalação do sistema. Ao condicionar o ambiente a luz artificial de uma lâmpada, posicionada na estrutura móvel instalada, é possível expor o objeto que será identificado, delimitando bordas e evidenciando atributos específicos do drone.

2 Metodologia

Atualmente são encontradas diversas soluções que permitiriam a realização do processo de detecção de drones através do uso de imagens. Entretanto, como já explicitado, a solução de *deep learning* foi a escolhida para esse Trabalho de Graduação, dado o avanço significativo que a área vem tendo, bem como a quantidade de documentação disponível, crescente comunidade e facilidade para encontrar recursos referentes ao assunto.

O desenvolvimento desse projeto se fundamentou no *framework open source Tensorflow* para Python, dado que permite o trabalho com redes neurais para uma gama de *hardwares*, possibilitando execução dos algoritmos com ou sem o uso de GPUs. Contudo, para propiciar facilidade no treinamentos da base, a utilização de GPU no ambiente de desenvolvimento do Google Collab foi considerada, visto sua perceptível agilidade no processamento, diferentemente do que acontece quando se utiliza apenas a capacidade da máquina que se tem à disposição. Dessa forma, foi facilitada a atuação eficiente frente aos erros encontrados durante os treinamentos, permitindo melhor comparação entre os resultados obtidos.

Este *framework* possibilita criar modelos de redes neurais de forma prática e para esta aplicação considerou-se de início tanto o tipo *SSD Mobilenet*, quanto o *Faster RCNN*, dado suas características específicas em termos de velocidade e processamento. Porém, como será abordado com mais profundidade na seção de resultados [3](#), o primeiro foi escolhido para o sistema após os testes, pois atende de forma mais eficaz a proposta: ao se tratar de um sistema de vigilância, a agilidade na resposta e imagens em tempo real são imprescindíveis, e o *Faster RCNN* não foi capaz de atingir tais requisitos.

2.1 Pré-processamento dos dados

Em uma etapa inicial do trabalho, figuras obtidas via Google Imagens foram utilizadas para a criação de uma base de dados com fotos genéricas de drones, onde a característica comum definida era apenas a devida presença de quatro hélices. Dessa forma, objetos voadores não tripulados dos mais variados tipos foram considerados, em cores, cenários e situações diversificadas, além de ângulos e posições sortidas. O único tratamento prévio feito foi a padronização das imagens em um tamanho único de 800x600 com o formato de arquivo JPEG.

Porém, o modelo gerado a partir do treinamento com esta base de dados inicial não se mostrou muito satisfatório para a aplicação. Tal implicação negativa foi decorrente da baixa capacidade de detecção para um modelo específico de drone.

Dessa forma, com o modelo BREEZE 4K em mãos foi gerada uma nova base de dados dedicada à detectar apenas este exemplar de dispositivo. Imagens próprias puderam ser obtidas, considerando todo pequeno ângulo, detalhes e fundos diferenciados, bem como pessoas também puderam aparecer, sendo incorporadas ao cenário, o que era mais difícil de fazê-lo considerando apenas o que era disponibilizado na internet. Com isso, as especificações do *Tensorflow* para classificação de imagens puderam ser devidamente atendidas, conforme visto na seção 1.4.1.

Foi utilizada a ferramenta gratuita de anotação de imagem gráfica *LabelImg* para selecionar manualmente, imagem por imagem, o objeto a ser categorizado no modelo, criando uma classe para o mesmo (drone), pois o sistema final não faz discernimento quanto às características específicas, tais como a quantidade de hélices, ou distinção de marca, uma vez que o propósito final é apenas detectar a presença de um tipo específico de drone (BREEZE 4K).

A primeira etapa do pré-processamento envolveu a seleção da região de interesse de cada figura, classificando-a como um drone, em todas as 270 imagens. Após tal identificação, o software *LabelImg* gera arquivos automaticamente no formato XML com informações da localização do objeto dentro da imagem, possuindo mesmo nome que a foto com a qual está atrelado, sendo usado para as etapas seguintes.

Logo após, as imagens do dataset atreladas aos seus respectivos arquivos XML foram divididas em duas pastas: uma contendo o conjunto de dados para treino e a outra o de teste, adotando uma proporção de 80% e 20% respectivamente. Tendo separado devidamente os dados e gerado tais arquivos, foi criado um arquivo com formato CSV contendo todas as informações dos arquivos XML. As informações contidas neste novo documento indicam o nome de cada uma das imagens, o tamanho referente a altura e largura delas, a classe e a posição que foi rotulada.

O passo seguinte consistiu na geração de um arquivo do tipo TFRecords a partir daquele com formato CSV, sendo uma ação realizada por meio da execução de um *script* de conversão. Este arquivo gerado guarda uma sequência de arquivos binários para serem lidos na rede neural durante o treinamento. Também foi configurado um documento do tipo PBTXT que contém informações da classe a ser detectada, sendo também utilizado na próxima etapa.

2.2 Treinamento

Uma vez que a quantidade de fotos fornecida na etapa anterior é uma quantidade muito pequena, apenas os conjuntos de treino e teste foram utilizados nesse momento, desconsiderando a necessidade de um conjunto de validação, o que pode acontecer para *datasets* maiores.

2.2.1 Desenvolvimento do treinamento

De início, os primeiros treinamentos realizados foram executados diretamente na máquina que estivesse sendo utilizada no momento, se valendo apenas do processador disponível pela mesma. Isso tomava muito tempo, sendo às vezes executado por horas, tornando a avaliação de erros e atuação em melhorias algo muito custoso e demorado. A utilização da GPU no ambiente de desenvolvimento do *Google Colab* para a etapa de treinamento permitiu mais agilidade nessa atividade, o que trouxe a possibilidade de realizar vários treinamentos, comparando-os de forma mais fácil e permitindo a escolha do melhor modelo para a implementação final do sistema.

Implementar um modelo a partir do zero é uma tarefa de alta complexidade e que foge do objetivo principal deste trabalho, sendo esse simplesmente aqui definido como desenvolvimento de um sistema de vigilância. Por isso, foi adotada a transferência de aprendizado de outro modelo pré-treinado, mas que possui mesmo propósito final de detecção de objetos. A análise de medidas de desempenho, tais como acurácia e velocidade, e também a constatação de convergência da função de custo, mostrou que o modelo pré-treinado *Microsoft COCO (Common Objects Context)* ([LIN et al., 2014](#)) atende o escopo do sistema, permitindo atividades que envolvam reconhecimento e classificação de objetos em tempo real.

Da gama de modelos pré-treinados COCO existentes foram utilizados o *Faster RCNN* e o *SSD Mobilenet*, dado suas características já abordadas na seção de pressupostos teóricos, a fim de realizar testes comparativos que pudessem explicitar qual dos dois apresentaria melhor desempenho em tempo real e que também fosse respaldado pelo cálculo de frames por segundo, mais aprofundado na seção [2.2.2](#).

O treinamento consistiu na execução do *script* de treino, previamente configurado no *pipeline* de treino, na máquina virtual do Google Colab. Este utiliza os arquivos TFRecord e PBTXT da etapa de pré-processamento como entrada de dados. Para tal configuração, foi considerado o modelo pré-treinado escolhido, o tamanho do lote (*batch size*), a quantidade de passos por época durante o treino e o teste. O produto final da etapa de treinamento foi o último grafo de inferência salvo no processo de treinamento, este possui formato de arquivo PB.

Durante o processo de treinamento, a cada passo ocorre uma alteração no valor de perda (*loss value*), que está relacionado a convergência do modelo escolhido e o ajuste de pesos na rede neural a cada iteração, ([VINICIUS, 2017](#)) permitindo constatar tal grandeza durante o processo.

Além das informações de perda da função durante o processo de treinamento, também foram obtidas as principais métricas de desempenho na aplicação do modelo no conjunto de dados de teste. Os gráficos que relacionam a quantidade de iterações ao valor

de perda se encontram na seção de resultados e serviram de parâmetro para a escolha do modelo, tendo preferência para aquele que teve maior convergência.

O critério de parada para a finalização do treinamento foi estabelecido conforme a quantidade total de ciclos previamente determinada. Desta forma, foram realizados três treinamentos com o mesmo conjunto dados e modelo pré-treinado, somente alterando a quantidade total de passos, inicialmente-se com 10000, em seguida 20000 e por fim 30000. Para o caso do modelo pré-treinado *Faster RCNN* foi realizado somente um processo de treinamento com 10000 passos, pois foi eliminado no teste de desempenho de velocidade em relação ao modelo com *SSD Mobilenet*.

2.2.2 Definição do melhor modelo

De início, dois treinamentos foram realizados com a mesma quantidade de steps (10000) tanto para o *SSD Mobilenet* quanto para o *Faster RCNN*, utilizando o mesmo dataset para excluir um deles como menos adequado ao objetivo final do sistema de segurança.

Dado que drones são veículos não tripulados controlados remotamente, o tempo para que este possa adentrar certa região pode variar conforme a necessidade percebida por quem o manipula. Sendo assim, é necessário que o sistema seja hábil o suficiente para captá-lo em vídeo.

Foi verificado a velocidade dos modelos através de uma contagem de frames por segundo (FPS). Para realizar esta contagem, foi criado uma variável contadora para cada passagem de frames dentro de um loop e também foi utilizado uma função interna do Python para computar o tempo atual. Ao final foi obtida a taxa de execução dos frames através da seguinte relação:

$$\text{Frames por segundo} = \frac{\text{total de frames}}{\Delta t} \quad (2.1)$$

Foi utilizado como critério para cada modelo o tempo decorrido Δt para 20 frames. Os resultados obtidos se encontram na seção [3.1.3](#) de resultados, já os cálculos envolvidos se encontram na seção [A.5](#) de apêndice.

Dessa forma, foi possível constatar qual modelo é visualmente mais lento, tanto em termos visuais quanto em termos quantitativos mensurados por frames por segundo. Verificar tal característica é importante visto que é indesejada ao produto final do projeto.

Após a definição do melhor modelo, a quantidade de *steps* a ser implementada deve ser verificada para melhor atingir o objetivo final do sistema implementado. Dessa forma, o modelo previamente escolhido foi treinado considerando números de *steps* incrementais, para avaliar qual melhor atende a necessidade de uma imagem fluida e em tempo real.

2.2.3 Métricas do treinamento

Durante a execução do treinamento, informações são fornecidas no terminal de saída de modo que seja possível acompanhá-lo. Nessa etapa, métricas como *loss* (perdas), o número de *steps* em cada época e o tempo de cada execução são definidos em processamento interno da máquina (GPU).

A perda é uma importante característica visto que explicita a convergência do modelo e o ajuste da rede neural. O tempo, por sua vez, varia de acordo com o número de *steps* e apenas define o quanto rápido ou devagar o modelo está sendo treinado. Mais uma vez ressaltando, o treinamento do *Faster RCNN* demorou mais quando comparado ao *SSD Mobilenet*, sendo outro ponto importante para ser usado no sistema de vigilância final, pois o procedimento pode ser repetido mais vezes e comparados de forma mais ágil.

Ao final da etapa do treinamento, o subgrupo de teste é designado a verificar a atuação do modelo e o mesmo também foi realizado com um dataset apartado referente ao próprio drone BREEZE 4K, o que acabou fornecendo dados para análise de desempenho, possibilitando gerar matrizes de confusão, que é uma métrica importante para avaliar treinamentos e que correlaciona o quanto eficaz o modelo foi ao classificar, explicitando onde obteve ou não sucesso.

2.2.4 Métricas físicas

Valores físicos foram mensurados de modo a comparar o grau de abrangência dos modelos *SSD Mobilenet* treinados em termos da área monitorada, obtendo a distância de atuação do treinamento em relação ao foco da câmera. O mesmo valor de altura do posicionamento do dispositivo de imagem, bem como a mesma angulação da lente foi considerada para todas as medidas, tornando a comparação justa. O objeto de medição considerado foi uma trena comum e as medidas máximas e mínimas foram extraídas, sendo esta primeira a maior distância capaz de identificar um drone com um *score* (medida de reconhecimento que aparece no vídeo) não muito baixo, e a última considerando o quanto próximo da câmera o objeto se encontra do foco e ainda pode ser classificado adequadamente.

2.2.5 Métricas do teste com o dataset apartado

O modelo escolhido foi treinado três vezes (com 10000, 20000 e 30000 *steps*) utilizando o mesmo *dataset* e foram devidamente testados com as mesmas imagens apartadas, também referentes ao drone BREEZE 4K, identificando visualmente uma a uma se caso a classificação ocorreu de forma satisfatória ou não, gerando uma tabela que se encontra na seção A.3 do apêndice, onde o que era esperado foi comparado com o que de fato aconteceu.

Tal tabela explicitou se cada uma das imagens resultou em um Falso Positivo, Falso Negativo, Verdadeiro Positivo e Verdadeiro Negativo. Essas variáveis embasam o cálculo das métricas do sistema, sendo elas: matriz de confusão, acurácia, revocação, precisão e F1 score.

2.3 Sistema de rastreamento sonoro

Embora o drone possua uma assinatura visual muito específica, sua intensidade sonora também foi um atributo explorado para que a câmera não se comportasse como dispositivo estático, o que acabaria por limitar o campo de atuação do sistema ao campo de visão da mesma.

Para contornar tal obstáculo e compreender toda uma área em dois quadrantes, dois microfones de eletreto modelo KY-038 foram instalados na estrutura em direções opostas. A utilização de tais sensores se justifica dada o quão característico é o som dos objetos voadores não tripulados, em termos de frequência e intensidade sonora. Dessa forma, a estrutura desenvolvida se movimenta em direção a um quadrante específico dada a presença de um drone na região.

Os microfones de início não possuíam a mesma sensibilidade, sem um padrão de fábrica. Para que ambos apresentassem a mesma calibração, o potenciômetro de cada uma das placas teve que ser ajustado de modo que a saída de ambas no terminal de entrada analógica do microcontrolador do Arduino Leonardo fossem as mesmas.

A detecção do drone considera as duas entradas de som, de maneira que possam ser comparadas: se caso não houver diferença entre as duas, a estrutura se mantém na mesma posição, mas se existir alguma discrepância entre os lados, a câmera é posicionada para a direção que o áudio recebido é mais intenso.

2.4 Sistema de iluminação automática

Por questões de segurança, o sistema deve se manter operante em tempo ininterrupto após sua instalação, podendo esta acontecer em qualquer ambiente. A necessidade de uma preparação para estas condições, tais como o próprio passar do dia, é um aspecto que deve ser considerado. Para permitir o funcionamento adequado da visão computacional em locais de pouca visibilidade, um sistema de iluminação automática foi implementado.

Um potenciômetro, ou *trimpot*, define um *setpoint* para o acionamento da lâmpada e pode ser configurado manualmente considerando as características específicas de cada local de atuação do sistema, limitando a tensão do LDR conforme a necessidade observada.

Este sistema também foi projetado para operar em função do nível da intensidade

sonora no ambiente a fim de melhor a condição luminosa incidente no drone assim que este se aproximasse dos microfones. Desta forma, foi programado para acionar a iluminação quando fosse atingido um limite de intensidade sonora nos na recepção dos microfones.

Uma lógica de programação foi implementada para comandar uma saída digital que chaveia um transistor TIP120, responsável por ligar e desligar o dispositivo emissor de luz. Para limitar a corrente de saída do Arduino, foi calculado o valor do resistor que limita a corrente da base do transistor com base nas leis de Ohm e nas informações fornecidas pelo fabricante do próprio transistor, disponíveis em datasheet. Tais cálculos e o circuito implementado se encontram na seção [B.2](#) do apêndice.

2.5 Interface gráfica

O sistema oferece a possibilidade de controlá-lo manualmente caso a necessidade ocorra, dependendo do tipo de ataque sofrido. Tal demanda foi resolvida através da implementação de uma interface gráfica, disponível no Apêndice ([C.1](#)), permitindo a interação do usuário na movimentação da câmera. O sistema continua classificando o drone através da visão computacional, mas no modo manual a localização do mesmo deve ser percebida e indicada pelo operador.

A interface foi implementada também utilizando a linguagem Python, através do uso da biblioteca tkinter. A interface permite a escolha entre os modos automático e manual: no primeiro, o sistema opera sozinho utilizando o som para o posicionamento da câmera, enquanto no modo manual dois botões para ambos os lados (direita e esquerda) são disponíveis para a movimentação da estrutura que comporta a câmera. Além disso, a lâmpada também pode ser acionada manualmente caso a necessidade seja percebida pelo usuário ao visualizar o vídeo.

2.6 Estrutura

Para acomodar a câmera, os microfones, o *protoboard*, a lâmpada e o Arduino, uma estrutura física foi desenvolvida. Com a utilização de um torno, seus componentes foram manufaturados e seguem a decorrente forma de montagem: uma estrutura externa de alumínio acomoda o motor, tendo como encaixe uma base de *nylon*. O eixo, também de alumínio, possui uma junção de *nylon* para conectar ao motor, permitindo a transferência de movimento para um prato de PU (material similar ao *nylon*, porém mais poroso). O eixo é suportado por uma guia de ferro fundido presente na tampa do esqueleto externo de alumínio, enquanto um arruela de *nylon* evita a sobrecarga do motor. Boa parte dos componentes são acomodados por um suporte de polipropileno 100%, acomodado acima do prato e fixado por parafusos.

2.7 Integração dos sistemas

O sistema final é definido pela integração de todos os subsistemas. A câmera, acoplada na estrutura, pode ser movimentada de forma automática, através da atuação dos microfones receptores posicionados de forma radialmente oposta, ou de maneira manual, com a utilização da interface desenvolvida, permitindo que o usuário decida o posicionamento da câmera. Enquanto isso, a lâmpada presente no protótipo acende caso seja percebido pelo sistema que o ambiente de instalação possui baixa luminosidade ou alto nível sonoro. O algoritmo que compõe a visão computacional do sistema é executado em conjunto com essas ações descritas, sendo todas unidas pela presença de um Arduino e interface gráfica, cujo códigos estão disponíveis no apêndice na seção [C.2](#).

2.8 Execução em tempo real

Após a devida integração dos subsistemas, a execução do modelo em tempo real permite avaliar a completa funcionalidade do sistema e perceber a execução em cadeia das ações realizadas por cada um deles.

Quando um drone adentra a região de monitoramento, os microfones receptores captam o som emitido e o rastreiam, fazendo com o servo-motor responda a esse sensor e movimento a estrutura, colocando o objeto no campo de visão da câmera operante. Um alarme pode ser acionado para indicar que o score de identificação está acima de 70%, mostrando grandes chances da classificação estar adequada.

Além disso, através do uso da interface, um histórico pode se acessado, mostrando as identificações registradas pelo sistema, junto com outras informações relevantes para o usuário, tais como a data. Se caso for observada a necessidade, o sistema pode ser operado manualmente, assim como a luz também pode ser ligada quando for avaliada a conveniência.

3 Resultados e Discussão

O sistema final é composto por uma estrutura física capaz de suportar e movimentar a câmera através do uso de um servo-motor, monitorando a área especificada em dois quadrantes. Com o auxílio de dois microfones receptores, a movimentação acontece de acordo com o controle realizado por um microcontrolador. Este utiliza as entradas de áudio de ambos para calcular o lado ao qual a câmera deve ser posicionada, visto a diferença de intensidade sonora entre os dois sensores de som.

Quando devidamente posicionada, o algoritmo utiliza a imagem da câmera para identificar o que é um drone, utilizando conceitos de visão computacional. O usuário do sistema projetado é capaz de controlá-lo manualmente ou deixá-lo operando em modo automático, caso seja necessário, visto a disponibilidade de uma interface, que também foi desenvolvida neste Trabalho em Graduação.

3.1 Resultado e discussão das métricas para algoritmo de detecção

3.1.1 Métricas no treinamento e validação

Para avaliar os modelos enquanto ainda estão sendo treinados, foram utilizados os valores de perda (*loss value*) apresentados de forma gráfica através da biblioteca *Tensorboard*, sendo gerados automaticamente durante a etapa de avaliação dos modelos com o conjunto separado como teste do dataset. Foi gerado um gráfico para cada treinamento com a quantidade pré-definida de steps, sendo estes apresentados nas figuras [7](#), [8](#) e [9](#) a seguir. Deve-se ressaltar aqui, que legendas não são apresentadas, bem como eixos, e dessa forma as imagens são exibidas como de fato apareceram pelo sistema:

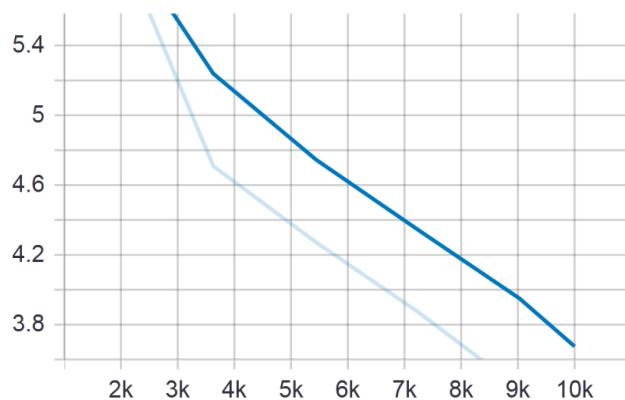


Figura 7 – 10000 passos: Valor de perda x Número de passos do conjunto teste

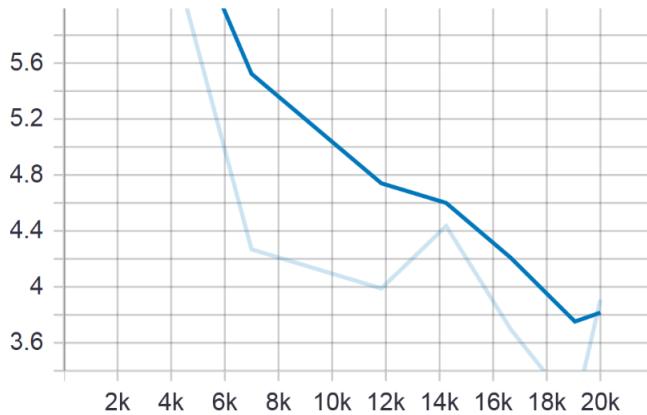


Figura 8 – 20000 passos: Valor de perda x Número de passos do conjunto teste

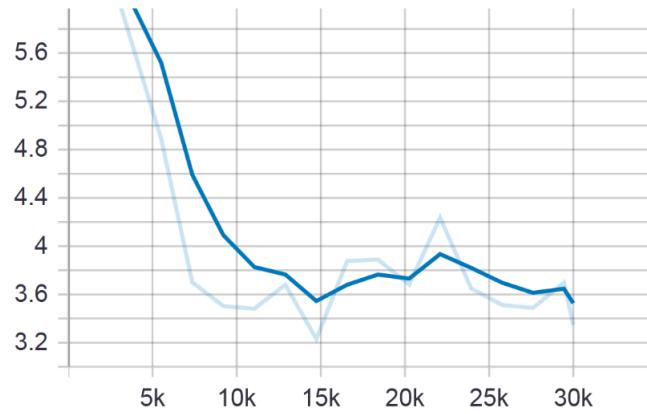


Figura 9 – 30000 passos: Valor de perda x Número de passos do conjunto teste

A convergência da função de custo, expressa pelos valores de perda, foi observada de forma mais evidente no gráfico do modelo *SSD MobileNet* 30000 passos apresentado na figura 9. Esta mostra que a escolha da taxa de aprendizado (*learning rate*) foi adequada para o treino do modelo, de tal forma que ele tenha convergido para uma estreita faixa de valores, aproximadamente entre 3,6 e 4. Estes especificam a diferença entre os valores preditos e os reais esperados, além de seu comportamento dentro da função de custo, sendo característico de um *dataset* balanceado em relação ao tipo de imagem escolhida.

Os valores de perda para o conjunto de treino também convergiram durante o treino. Esta proporção direta de convergência entre o conjunto de testes e o conjunto de treino denota que a rede não parou de aprender em nenhum momento e também não se tornou super ajustada (FACELI et al., 2011).

Os gráficos mostrados nas figuras 10, 11 e 12 mostram o comportamento durante o processo de treinamento com o conjunto dados para treino. As informações sobre as tabelas obtidas se encontram no apêndice na seção A.1.

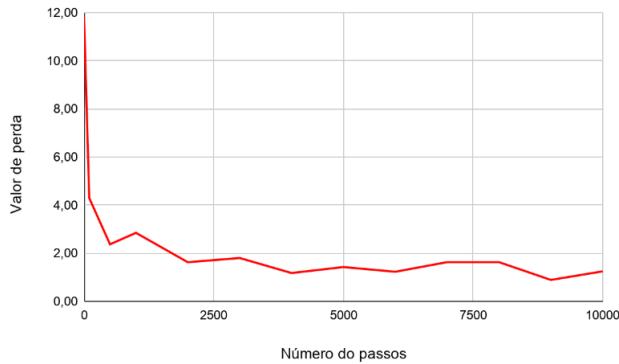


Figura 10 – 10000 passos: Valor de perda x Número de passos do conjunto treino

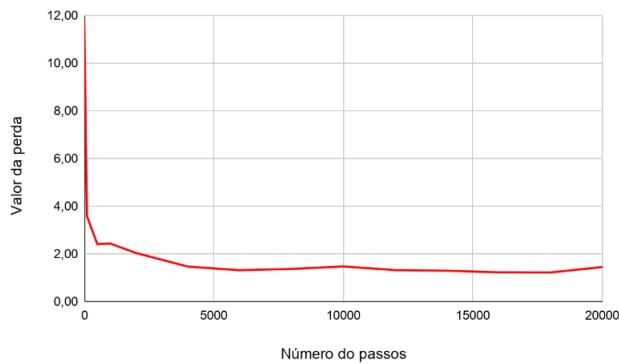


Figura 11 – 20000 passos: Valor de perda x Número de passos do conjunto treino

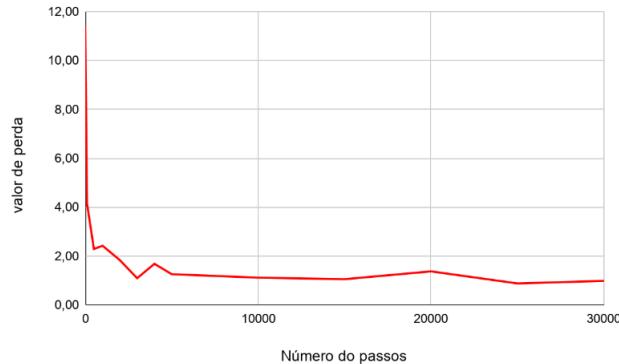


Figura 12 – 30000 passos: Valor de perda x Número de passos do conjunto treino

3.1.2 Métricas com conjunto de imagens para teste

Além da etapa de teste que é realizada pelo próprio treinamento com os 20% de imagens designadas para tal tarefa, foi efetuado um teste apartado com fotos diferentes daquelas que estão presentes no dataset. Dessa forma, o modelo pode ser testado não apenas em mecanismos internos, mas também de forma visível para que dados pudessem ser auferidos do mesmo, de modo a gerar métricas para justificar a escolha do modelo utilizado no sistema final.

O teste resultou em Falsos Positivos, Falsos Negativos, Verdadeiros Positivos e Verdadeiros Negativos, essenciais para os cálculos das métricas que avaliam o modelo. Na figura 13 encontra-se um exemplo de detecção com Falso Negativo (FN) por não detectar o drone presente e Falso Positivo (FP) por acusar um drone com porcentagem alta onde não o há, ambos feitos com base em imagens de um vídeo obtido na internet ([REVIEWS, 2019](#))



Figura 13 – Exemplo de Falso Negativo

Já na figura 14 encontra-se um exemplo de detecção com Verdadeiro Positivo (VP) por detectar precisamente um drone com um score elevado:



Figura 14 – Exemplo de Verdadeiro Positivo

Em todas as imagens foi considerado o verdadeiro negativo quando não ocorreu a classificação de nenhuma outra região da imagem como drone, além daquela que o contém, sendo considerada como um negativo. Isto não ocorreu em nenhum caso na validação com as imagens de teste com os modelos treinados com *SSD Mobilenet*.

Para melhor medida de desempenho para classificação de cada modelo através dos valores preditos e dos valores esperados, foi feita a matriz de confusão para cada um com um conjunto de dados de 20 imagens. Estas métricas se encontram em forma de tabela a seguir.

Tabela 2 – Matriz de confusão MobileNet 10000 passos

Matriz de confusão		Esperado	
		Detecção de drone	Não detecção de drone
Previsto	Detecção de drone	11 VP	5 FP
	Não detecção de drone	2 FN	20 VN

Tabela 3 – Matriz de confusão MobileNet 20000 passos

Matriz de confusão		Esperado	
		Detecção de drone	Não detecção de drone
Previsto	Detecção de drone	16 VP	0 FP
	Não detecção de drone	4 FN	20 VN

Tabela 4 – Matriz de confusão MobileNet 30000 passos

Matriz de confusão		Esperado	
		Detecção de drone	Não detecção de drone
Previsto	Detecção de drone	17 VP	3 FP
	Não detecção de drone	2 FN	20 VN

A partir das informações obtidas pelos gráficos foram calculados os valores de precisão, acurácia, revocação e F1-score, as equações para a determinação são descritas na Metodologia na subseção [1.4.6](#) e seus respectivos desenvolvimentos estão na seção de anexos. Com base nos cálculos realizados, foram obtidos os valores mostrados na tabela [5](#).

Sobre as informações, pode-se inferir que o modelo treinado com 20000 *steps* possui maior qualidade, no sentido geral, devido ao valor de F1-score. De fato foi observado um bom desempenho na execução em tempo real para o mesmo, porém foi percebido um valor maior para Falsos Positivos em relação ao de 30000, contrastando aos dados informados na matriz de confusão. Tal ocorrência pode ser decorrente da diferença entre o conjunto limitado de informação para 20 imagens para testes de métricas e o funcionamento em tempo real em um ambiente diversificado.

Tabela 5 – Métricas

	10000 passos	20000 passos	30000 passos
Precisão	0,687	1,000	0,850
Acurácia	0,775	0,900	0,925
Revocação	0,350	0,440	0,450
F1-score	0,462	0,610	0,580



Figura 15 – Exemplo de Falso Positivo para região com similaridades visuais de um Drone

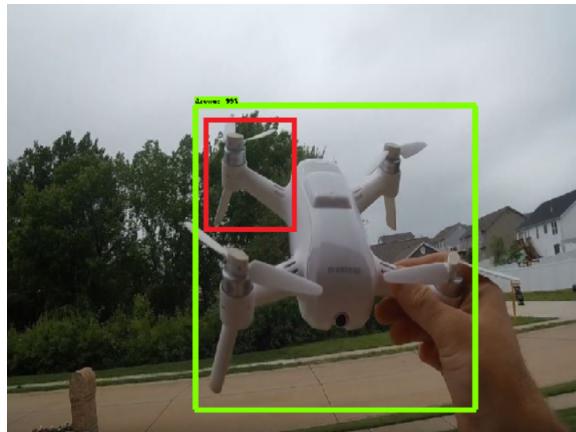


Figura 16 – Parte similar a região com detecção de Falso Positivo descrita na Figura 15

A região encontrada na figura 15 correspondente a entrada de uma garagem com solo branco e tem similaridades de traços com o braço do drone e também com o motor, conforme indicado em vermelho na figura 16. O algoritmo detecta o objeto com base na quantidade de padrões correspondentes ao esperado em determinada região e, dessa forma, houve detecção com falso positivo para a região em vermelho da figura 15 (entrada da garagem). Embora houvesse este tipo de reconhecimento, ainda assim o modelo mostrou bom desempenho para aplicação em tempo real para diversos ambientes.

O modelo com treinamento de 30000 *steps* apresentou melhores resultados neste sentido, com menor ocorrência de Falsos positivos quando executado em tempo real, e por este motivo foi elencado para ser implementado ao sistema.

3.1.3 Desempenho do sistema de identificação em termos de velocidade de vídeo

Após o devido treinamento, o código referente à conexão da captura de vídeo e devida execução do modelo de visão computacional pôde ser desempenhado. Foi possível comparar ambos os recursos disponíveis para treinamento (Mobilenet e Faster RCNN), sendo bastante perceptível a diferença entre ambos: para o escopo deste desenvolvimento, dada sua relevância e aplicabilidade em termos de segurança, uma resposta mais rápida para uma devida classificação do objeto voador na área monitorada, se torna imprescindível.

O Faster RCNN se mostrou mais lento, necessitando de uma imagem estática ou com pouco movimento, o que não permite um funcionamento adequado do sistema, dado que um drone é um objeto voador dinâmico e imprevisível. O Mobilenet, por sua vez, permitiu que o vídeo funcionasse de forma fluída e em tempo real, adequando-se à proposta desse trabalho e, permitindo se necessário, que o usuário controlasse o sistema manualmente conforme a necessidade.

Com o contador de frames e impressão do tempo conforme o funcionamento da captura de imagem e atuação do modelo aconteciam, o cálculo de frames por segundo pôde ser efetuado e a tabela 6 apresenta tais resultados, justificando a escolha do Mobilenet de 30000 *steps* para o sistema final sem considerar apenas o que é observado em vídeo:

Tabela 6 – Frames por segundo

Modelo executado	Frames por segundo
SSD Mobilenet 10000	3,87
SSD Mobilenet 20000	3,45
SSD Mobilenet 30000	4,56
Faster RCNN	0,42

Com base nos valores encontrados na tabela, observa-se que o modelo Faster RCNN possui uma taxa de frames por segundo consideravelmente inferior aos demais modelos treinados com SSD MobileNet. Por conta disso, a não utilização do modelo Faster RCNN é justificada por não atender aos requisitos de tempo real para a detecção de um drone, que pode ter uma velocidade horizontal de até 5 m/s. Com esta característica, ele pode atravessar o campo de visão da câmera na região detecção em aproximadamente 1 segundo e neste contexto o sistema não conseguiria captá-lo devido a baixa taxa de frames do modelo descartado.

3.1.4 Dados obtidos com o modelo executado

Da mesma forma que as métricas foram obtidas através da execução do modelo de detecção com um dataset separado contendo imagens que obtidas a partir de vídeo disponibilizado na internet, o mesmo foi realizado operando-o em tempo real, sendo possível

encontrar exemplos de Verdadeiro Positivo, Verdadeiro Negativo, Falso Positivo e Falso Negativo. As imagens podem ser encontradas a seguir, para exemplificar.



Figura 17 – Exemplo de detecção para Verdadeiro Positivo



Figura 18 – Exemplo de detecção para Verdadeiro Negativo



Figura 19 – Exemplo de detecção para Falso Positivo

O exemplo de detecção para Falso Positivo apresentado na figura 19 mostra que o sistema está sujeito a erros. Porém, como pode ser observado, a detecção possui um score relativamente baixo para este. Além disso, a imagem da região detectada possui

um formato com bordas semelhantes às encontradas na figura de um drone, o que pode justificar tal tipo de detecção.

3.2 Resultados e discussão do sistema em função do rastreamento sonoro

A movimentação do sistema se deu de forma adequada, respondendo bem ao comando do microcontrolador ante o sinal processado das entradas dos dois microfones receptores e mostrando que a estrutura mecânica final comporta bem os componentes e é capaz de se movimentar de maneira estável através da transmissão de movimento do motor.

3.2.1 Algoritmo de movimentação baseado na intensidade sonora

Além de suas características visuais, o drone também possui uma identidade sonora bastante específica. O raio de atuação do sistema de sensoriamento sonoro foi definido como o raio máximo de captação desse som capaz de ser processado pelo algoritmo do microcontrolador de forma eficaz para o comando de movimentação dos servo motores. Os valores relacionados à atuação do sistema de rastreamento se encontram na tabela 7:

Tabela 7 – Atributos

Atributos	Valores
Raio de atuação	0 a 110 cm
Faixa angular de operação	0 a 180 graus
Intensidade sonora no raio máximo de atuação	45 db

O sistema de movimentação baseado na intensidade sonora foi capaz de auxiliar o sistema de visão computacional ao direcionar a câmera horizontalmente em direção ao drone. Com este posicionamento foi possível ampliar o campo de visão da câmera, além de também permitir que o drone ficasse no foco do dispositivo de captação de imagem, em diferentes localizações, dentro de um campo de 180 graus. A identificação pelo sistema de visão computacional pôde atuar com melhor desempenho nessas condições.

3.2.2 Estrutura física

A estrutura montada foi capaz de movimentar a câmera e a lanterna em resposta às entradas dos microfones de maneira estável, com poucas oscilações de imagem, sendo isso de forma tanto manual quanto automática, transmitindo bem a movimentação do motor para o prato. Além disso, ela também foi bastante eficiente ao suportar todos os outros componentes embarcados.

Apesar de ter sido apropriada para o objetivo do sistema, uma estrutura e um sistema capazes de não apenas ter abrangência lateral, mas também inferior e superior, pode ser apontado como ponto de melhoria. Um sistema capaz de movimentar a câmera em uma região correspondente a um domo seria mais efetivo ao identificar drones, visto que estes são objetos voadores de movimento imprevisível.

A figura 20 mostra a estrutura física final com o sistema eletrônico embarcado.

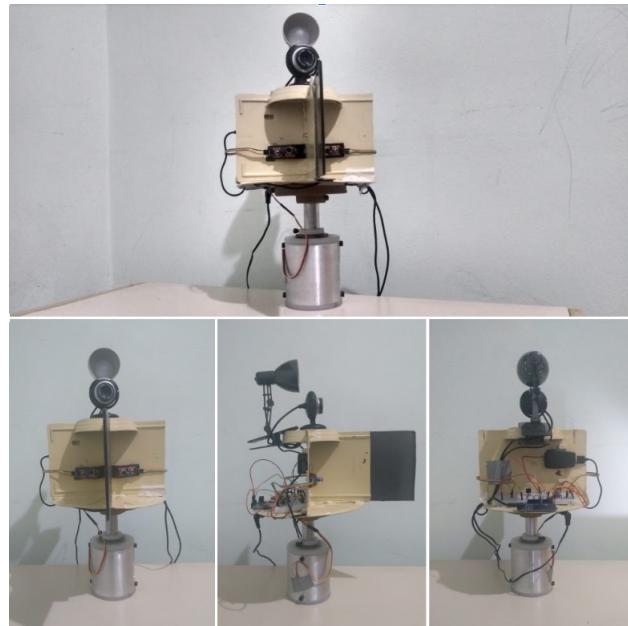


Figura 20 – Montagem de fotos das várias visões da estrutura

Com pode ser observado, os dois microfones são separados fisicamente por uma barreira. Tal separação visa isolar o som proveniente das laterais cada um por vez, de forma que a diferença de intensidade sonora entre eles seja mais evidenciada para o posterior processamento. Os testes práticos mostraram que o mecanismo de controle teve uma resposta melhorada proporcionalmente ao aumento do isolamento. Neste sentido, foi aumentado a extensão da divisão do centro com a inserção de uma divisória extra, que corresponde a chapa de cor preta que se encontra na região central da parte móvel e evidenciada na vista lateral da estrutura na imagem.

3.3 Resultados e discussão do sistema em função da luz ambiente

Conforme visto na seção de métricas físicas, os diferentes treinamentos (10000, 20000 e 30000 steps, todos do tipo Mobilenet) foram individualmente testados. Dessa forma, os dados relacionados a distância de captação de imagem e funcionamento da classificação do objeto puderam ser auferidos, estando em função da presença ou ausência de luminosidade no ambiente.

Com a lâmpada posicionada na mesma altura e mesmo ângulo de lente da câmera, os dados foram captados com uma trena e podem ser verificados na tabela 8 e são exemplificados pelas imagens 21 e 22 que mostram, respectivamente, como foram mensurados os valores de distância máxima e mínima em circunstâncias diferentes de luminosidade: com o drone no foco da câmera, tomou-se distância até o momento em que o sistema apresentava um *score* muito baixo ou deixava de apresentar reconhecimento, correspondendo ao valor máximo de atuação. O mesmo procedimento também foi realizado para curtas distâncias, onde o drone foi aproximado da câmera até que esta não pudesse mais discernir o objeto à sua frente, dada sua proximidade.

Tabela 8 – Distâncias mensuradas em ambientes com ou sem luminosidade

Luminosidade	Modelo (steps)	Distância máxima (cm)	Distância mínima (cm)
Luz ambiente natural	10000	186	27
	20000	230	19
	30000	160	30
Luz proveniente da câmera (ambiente noturno)	10000	181	48
	20000	170	22
	30000	177	26



Figura 21 – Exemplo de uma das medidas de distância máxima em ambiente escuro

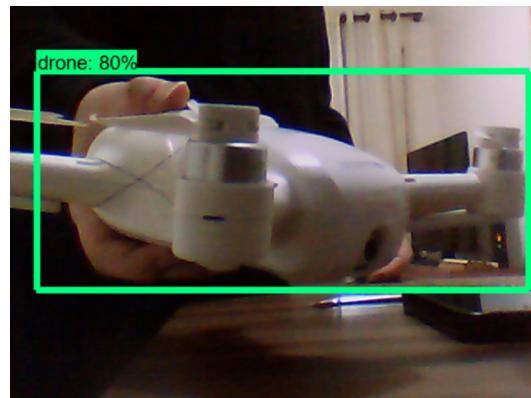


Figura 22 – Exemplo de uma das medidas de distância mínima em ambiente claro

Embora os testes tenham todos apresentado boa capacidade de identificação, os de 10000 *steps* e 20000 *steps* apresentaram frequentemente falsos positivos. Tal ocorrência

serviu de motivo para descartar suas implementações para o conjunto final, de forma que foi definido o modelo de Mobilenet treinado com 30000 *steps* como principal modelo.

O aparelho de iluminação resultante foi eficiente ao operar em modo autônomo ou manual, conforme esperado. Seu funcionamento foi decorrente da associação do sensoriamento via resistor LDR, processamento do sinal de luminosidade ambiente e acionamento da lâmpada. Com base nos valores da faixa de atuação, foi possível constatar a capacidade deste sistema manter as condições de luminosidade adequadas para o funcionamento pleno do sistema de visão computacional.

Muito embora os testes tenham se mostrado satisfatórios, deve-se considerar que as condições de cada ambiente são individuais e específicas, fazendo com que o modelo possa se comportar de maneira diferente. É importante ressaltar que o contraste entre objeto e cenário de fundo é uma condição que impacta diretamente em sua identificação: quanto mais escuro ambiente, mais nítida é a assinatura visual do drone e, portanto, mais bem identificado ele será, apresentando altos *scores*.

3.4 Resultados e discussão da interface gráfica

A interface gráfica implementada se mostrou bastante eficiente e amigável ao usuário final do sistema. Ao teclar o botão Ligar câmera o sistema passa a operar de forma autônoma por definição. O botão Manual aciona a possibilidade de mover a estrutura mecânica de modo a posicionar a câmera e lâmpada para melhor identificação do drone invasor, podendo ou não acionar o dispositivo de luz para aumentar a visibilidade do objeto em ambientes escuros. O sistema volta a operar de forma automática novamente com o acionamento do botão Automático. O histórico de identificações pode ser acessado pelo operador, gerando uma espécie de relatório do sistema em estado operante. A imagem a seguir ilustra a interface gráfica:



Figura 23 – Interface implementada

Conclusões e Perspectivas

Em suma, o sistema teve bom comportamento em termos do modelo implementado e decorrente visão computacional baseada em redes neurais, sendo bastante eficaz na classificação de drones do modelo BREEZE 4K, mostrando uma imagem fluida, ágil em questão de frames por segundo e bastante precisa ao classificar, apresentando altos *scores*.

O sistema ficou mais bem adaptado para enfrentar situações reais com a aplicação do modelo desenvolvido em *deep learning* em conjunto com os subsistemas de rastreamento por intensidade sonora, emissão de alarmes, adaptação da iluminação às condições do ambiente, movimentação do sistema embarcado e interface gráfica do usuário, permitindo o direcionamento da câmera para o objeto a ser monitorado em quaisquer condições de claridade. Neste contexto, a interface tornou esse conjunto mais prático ao permitir que o utilizador escolha operar no modo manual ou automático e seja capaz de obter o histórico de eventos de detecções anteriores.

Embora o projeto apresentado funcione como o esperado pelo escopo definido, ainda está no âmbito de prototipagem simplificada, pois entende-se que para a aplicação com eficiência em campo ainda há diversas limitações e melhorias a serem implementadas. Dentre os aperfeiçoamentos, pode-se citar: Rastreamento com uso de radar, identificação a partir da frequência de rádio operacional do drone, aplicação de aprendizado de máquina para reconhecimento de padrões de som a partir da assinatura sonora drone, aplicação de localização de fonte sonora por defasagem do sinal de som recebido por dois receptores e monitoramento remoto via Web.

Referências

- BALSYS, R. *TensorFlow Object Detection Step by Step custom object detection tutorial*. 2018. Python Lessons. Disponível em: <<https://pylessons.com/Tensorflow-object-detection-step-by-step-custom-object-detection/>>. Acesso em: 7 ago. 2019. Citado na página 8.
- BLOG, D. *tensorflow-object-detection-training-colab*. 2019. Google Colab. Disponível em: <https://colab.research.google.com/github/Tony607/object_detection_demo/blob/master/tensorflow_object_detection_training_colab.ipynb>. Acesso em: 06 abr. 2020. Citado na página 67.
- BROWNLEE, J. *A Gentle Introduction to Cross-Entropy for Machine Learning*. 2019. Machine Learning Mastery. Disponível em: <<https://machinelearningmastery.com/cross-entropy-for-machine-learning/>>. Acesso em: 29 nov. 2019. Citado na página 10.
- BROWNLEE, J. *A Gentle Introduction to the Challenge of Training Deep Learning Neural Network Models*. 2019. Machine Learning Mastery. Disponível em: <<https://machinelearningmastery.com/a-gentle-introduction-to-the-challenge-of-training-deep-learning-neural-network-models/>>. Acesso em: 30 jun. 2019. Citado na página 10.
- BUSSON, A. J. G. et al. Desenvolvendo modelos de deep learning para aplicações multimídia no tensorflow. *Anais do XXIV Simpósio Brasileiro de Sistemas Multimídia e Web: Minicursos*. Editora SBC, p. 67–116, 2018. Citado 5 vezes nas páginas 3, 4, 5, 7 e 8.
- DEORA, R. *MobileNet Research Paper Walkthrough*. 2018. YouTube. Disponível em: <<https://www.youtube.com/watch?v=HD9FnjVwU8g>>. Acesso em: 05 mai. 2019. Citado na página 5.
- DINO. *Como a Machine Learning está revolucionando o cenário dos negócios*. 2019. Terra. Disponível em: <<https://www.terra.com.br/noticias/dino/como-a-machine-learning-esta-revolucionando-o-cenario-dos-negocios,3397de1473cb228e7d5579bc46190c95kqowb3my.html>>. Acesso em: 16 out. 2019. Citado na página 1.
- EE, A. *Drone de bolso Yuneec Breeze 4K*. 2020. 1A. EE. Disponível em: <<https://www.1a.ee/p/yuneec-breeze-4k-pocket-drone/5un1>>. Acesso em: 6 abr. 2020. Citado 2 vezes nas páginas 11 e 8.
- FACELI, K. et al. Inteligência artificial: Uma abordagem de aprendizado de máquina. 2011. Citado 2 vezes nas páginas 7 e 26.
- FARINACCIO, R. *Sistema de detecção de drones dá maior segurança para quem quer privacidade*. 2017. TecMundo. Disponível em: <<https://www.tecmundo.com.br/produto/122757-sistema-detectao-drones-maior-seguranca-quer-privacidade.htm>>. Acesso em: 02 fev. 2020. Citado na página 1.

GÉRON, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. [S.l.]: O'Reilly Media, 2019. Citado 2 vezes nas páginas 10 e 13.

GRANEMANN, E. *Drones: tendências para evolução da regulamentação*. 2019. CanalTech. Disponível em: <<https://canaltech.com.br/drones/drones-tendencias-para-evolucao-da-regulamentacao-152155/>>. Acesso em: 23 nov. 2019. Citado na página 1.

GRÜBLER, M. *Entendendo o funcionamento de uma Rede Neural Artificial*. 2018. Medium. Disponível em: <<https://medium.com/brasil-ai/entendendo-o-funcionamento-de-uma-rede-neural-artificial-4463fcf44dd0>>. Acesso em: 11 jun. 2019. Citado na página 3.

HARTMANN, W. M.; CONSTAN, Z. A. Interaural level differences and the level-meter model. *The Journal of the Acoustical Society of America*, Acoustical Society of America, v. 112, n. 3, p. 1037–1045, 2002. Citado na página 14.

HOWARD, A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. Citado na página 5.

HUI, J. *mAP (mean Average Precision) for Object Detection*. 2018. Medium. Disponível em: <https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173>. Acesso em: 05 mar. 2019. Citado na página 6.

IGUINA, E. *Yuneec Breeze 4K: Cheapest Good Quality Drone You Can Buy*. 2018. Bro.Geek. Disponível em: <<https://brogeek.com/2018/03/01/yuneec-breeze-4k-cheapest-quality-drone-can-buy/>>. Acesso em: 6 abr. 2020. Citado na página 11.

IPPOLITO, P. P. *Feature Extraction Techniques*. 2019. Towards Data Science. Disponível em: <<https://towardsdatascience.com/feature-extraction-techniques-d619b56e31be>>. Acesso em: 15 dez. 2019. Citado na página 9.

KANSAL, P. *How to start with your first Deep Learning Project... A Step by step Guide!* 2018. Medium. Disponível em: <<https://medium.com/@priyagoel0211/deep-learning-environment-set-up-from-a-viewpoint-of-a-beginner-a-step-by-step-guide-dbab6e15a057>>. Acesso em: 17 set. 2019. Citado na página 12.

KRISHNA, C. M. Real-time systems. *Wiley Encyclopedia of Electrical and Electronics Engineering*, Wiley Online Library, 2001. Citado na página 3.

LEITE, T. M. *Redes Neurais, Perceptron Multicamadas e o Algoritmo Back-propagation*. 2018. Medium. Disponível em: <<https://medium.com/ensina-ai/redes-neurais-perceptron-multicamadas-e-o-algoritmo-backpropagation-eaf89778f5b8>>. Acesso em: 10 nov. 2019. Citado 3 vezes nas páginas 11, 4 e 5.

LIN, T.-Y. et al. Microsoft coco: Common objects in context. In: SPRINGER. *European conference on computer vision*. [S.l.], 2014. p. 740–755. Citado na página 19.

MARTIN, J. *Programming real-time computer systems*. [S.l.], 1965. Citado na página 3.

- MISHRA, A. *Metrics to Evaluate your Machine Learning Algorithm*. 2018. Towards Data Science. Disponível em: <<https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234>>. Acesso em: 24 fev. 2019. Citado na página 12.
- PATNAIK, A. *Beginning TensorFlow*. 2018. Medium. Disponível em: <<https://medium.com/datadriveninvestor/beginning-tensorflow-dc041fc23392>>. Acesso em: 26 nov. 2019. Citado na página 7.
- PICCOLOTTO, L. *Como drones têm transformado nossas vidas - quais são os seus riscos*. 2020. GovTech. Disponível em: <<https://govtech.blogosfera.uol.com.br/2020/01/11/como-os-drones-tem-transformado-nossas-vidas-e-quais-sao-os-seus-riscos/>>. Acesso em: 02 fev. 2020. Citado na página 1.
- REVIEWS, R. D. *Yuneec Breeze 4K GPS Drone Flight Review*. 2019. YouTube. Disponível em: <<https://www.youtube.com/watch?v=FtY-SwfBPGM>>. Acesso em: 06 abr. 2020. Citado na página 28.
- ROSE, A. *Fine-tuning with Keras and Deep Learning*. 2019. Pyimagesearch. Disponível em: <<https://www.pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/>>. Acesso em: 3 set. 2019. Citado na página 9.
- SANDLER, M. *ensorflow detection model zoo*. 2019. GitHub. Disponível em: <https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md>. Acesso em: 05 mar. 2020. Citado na página 6.
- SEKHAR, G. V. S. *Real Time Systems*. [S.l.]: Excel Books Private Limited, 2013. Citado na página 3.
- SINJAB, A. *Step by Step: Build Your Custom Real-Time Object Detector*. 2019. Towards Data Science. Disponível em: <<https://towardsdatascience.com/detailed-tutorial-build-your-custom-real-time-object-detector-5ade1017fd2d>>. Acesso em: 26 nov. 2020. Citado na página 8.
- TIAN, D. ping et al. A review on image feature extraction and representation techniques. *International Journal of Multimedia and Ubiquitous Engineering*, Citeseer, v. 8, n. 4, p. 385–396, 2013. Citado na página 9.
- VINICIUS. *Perceptron - Redes Neurais*. 2017. Monolito Nimbus. Disponível em: <<https://www.monolitonimbus.com.br/perceptron-redes-neurais/>>. Acesso em: 05 mai. 2019. Citado na página 19.
- XU, J. *Deep Learning for Object Detection: A Comprehensive Review*. 2017. Towards Data Science. Disponível em: <<https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9>>. Acesso em: 05 mai. 2019. Citado na página 6.

Apêndices

APÊNDICE A – Tabelas e Cálculos

A.1 Tabelas para construção do gráfico de treino

Tabela 9 – Valor de Perda para 10000 passos

SSD Mobilenet 10000 passos	
Número de passos	Valor de perda
0	11,82
100	4,29
500	2,37
1000	2,85
2000	1,63
3000	1,80
4000	1,18
5000	1,43
6000	1,23
7000	1,63
8000	1,63
9000	0,89
10000	1,25

Tabela 10 – Valor de Perda para 20000 passos

SSD Mobilenet 20000 passos	
Número de passos	Valor de perda
0	11,80
100	3,59
500	2,41
1000	2,43
2000	2,04
4000	1,47
6000	1,31
8000	1,37
10000	1,47
12000	1,32
14000	1,29
16000	1,23
18000	1,22
20000	1,45

Tabela 11 – Valor de Perda para 30000 passos

SSD Mobilenet 30000 passos	
Número de passos	Valor de perda
0	11,41
100	4,15
500	2,29
1000	2,42
2000	1,83
3000	1,09
4000	1,68
5000	1,25
10000	1,11
15000	1,05
20000	1,37
25000	0,88
30000	0,98

A.2 Cálculos das métricas com os dados de teste

A.2.1 Modelo SSD Mobilenet 10000 steps

$$Precisão = \frac{VP}{VP + FP} = \frac{11}{11 + 5} = 0,687 \quad (\text{A.1})$$

$$Acurácia = \frac{VP + VN}{\sum_{n=1}^{20} VP_n + VN_n} = \frac{11 + 20}{40} = 0,775 \quad (\text{A.2})$$

$$Revocação = \frac{VP}{VP + VN} = \frac{11}{11 + 20} = 0,350 \quad (\text{A.3})$$

$$F1 Score = \frac{2 \cdot Precisão \cdot Revocação}{Precisão + Revocação} = \frac{2 \cdot 0,68 \cdot 0,35}{0,68 + 0,35} = 0,462 \quad (\text{A.4})$$

A.2.2 Modelo SSD Mobilenet 20000 steps

$$Precisão = \frac{VP}{VP + FP} = \frac{16}{16 + 0} = 1 \quad (\text{A.5})$$

$$Acurácia = \frac{VP + VN}{\sum_{n=1}^{20} VP_n + VN_n} = \frac{16 + 20}{40} = 0,900 \quad (\text{A.6})$$

$$Revocação = \frac{VP}{VP + VN} = \frac{16}{16 + 20} = 0,444 \quad (\text{A.7})$$

$$F1 Score = \frac{2 \cdot Precisão \cdot Revocação}{Precisão + Revocação} = \frac{2 \cdot 1 \cdot 0,44}{1 + 0,44} = 0,611 \quad (\text{A.8})$$

A.2.3 Modelo SSD Mobilenet 30000 steps

$$Precisão = \frac{VP}{VP + FP} = \frac{17}{17 + 3} = 1 \quad (\text{A.9})$$

$$Acurácia = \frac{VP + VN}{\sum_{n=1}^{20} VP_n + VN_n} = \frac{17 + 20}{40} = 0,925 \quad (\text{A.10})$$

$$Revocação = \frac{VP}{VP + VN} = \frac{17}{17 + 20} = 0,459 \quad (\text{A.11})$$

$$F1 Score = \frac{2 \cdot Precisão \cdot Revocação}{Precisão + Revocação} = \frac{2 \cdot 0,85 \cdot 0,45}{0,85 + 0,45} = 0,588 \quad (\text{A.12})$$

A.3 Tabelas para construção da matriz de confusão

Tabela 12 – Matriz de Confusão SSD Mobilenet 10000 passos

SSD Mobilenet 10000 passos									
Nome da imagem	Quantidade	Verdadeiro Positivo (%)	Quantidade	Verdadeiro Negativo (%)	Quantidade	Falso Positivo (%)	Quantidade	Falso Negativo (%)	
img teste_0	1	99	1	100	1	95	0	-	
img teste_1	1	90	1	100	0	-	0	-	
img teste_2	0	58	1	100	1	83	0	-	
img teste_3	1	100	1	100	0	-	0	-	
img teste_4	1	100	1	100	0	-	0	-	
img teste_5	0	-	1	100	1	99	0	-	
img teste_6	1	100	1	100	0	-	0	-	
img teste_7	0	99	1	100	0	-	0	-	
img teste_8	0	98	1	100	0	-	0	-	
img teste_9	0	-	1	100	0	-	1	Drone não detectado	
img teste_10	0	58	1	85	0	-	0	-	
img teste_11	0	98	1	100	0	-	0	-	
img teste_12	0	99	1	100	0	-	0	-	
img teste_13	0	-	1	100	1	99	1	Drone não detectado	
img teste_14	1	99	1	100	0	-	0	-	
img teste_15	1	99	1	100	0	-	0	-	
img teste_16	1	85	1	100	0	-	0	-	
img teste_17	1	100	1	100	0	-	0	-	
img teste_18	1	99	1	100	0	-	0	-	
img teste_19	1	83	1	100	1	80	0	-	
TOTAL		11		20		5		2	

Tabela 13 – Matriz de Confusão SSD Mobilenet 20000 passos

SSD Mobilenet 20000 passos									
Nome da imagem	Quantidade	Verdadeiro Positivo (%)	Quantidade	Verdadeiro Negativo (%)	Quantidade	Falso Positivo (%)	Quantidade	Falso Negativo (%)	
img_teste_0	1	99	1	100	0	-	0	-	
img_teste_1	0	-	1	100	0	-	1	Drone não detectado	
img_teste_2	1	99	1	100	0	-	0	-	
img_teste_3	1	99	1	100	0	-	0	-	
img_teste_4	0	-	1	100	0	-	1	Drone não detectado	
img_teste_5	0	-	1	100	0	-	1	Drone não detectado	
img_teste_6	1	98	1	100	0	-	0	-	
img_teste_7	1	100	1	100	0	-	0	-	
img_teste_8	1	99	1	100	0	-	0	-	
img_teste_9	0	-	1	100	0	-	1	Drone não detectado	
img_teste_10	1	100	1	100	0	-	0	-	
img_teste_11	1	99	1	100	0	-	0	-	
img_teste_12	1	99	1	100	0	-	0	-	
img_teste_13	1	98	1	100	0	-	0	-	
img_teste_14	1	99	1	100	0	-	0	-	
img_teste_15	1	96	1	100	0	-	0	-	
img_teste_16	1	99	1	100	0	-	0	-	
img_teste_17	1	99	1	100	0	-	0	-	
img_teste_18	1	100	1	100	0	-	0	-	
img_teste_19	1	99	1	100	0	-	0	-	
TOTAL		16		20		0		4	

Tabela 14 – Matriz de Confusão SSD Mobilenet 30000 passos

SSD Mobilenet 30000 passos									
Nome da imagem	Quantidade	Verdadeiro Positivo (%)	Quantidade	Verdadeiro Negativo (%)	Quantidade	Falso Positivo (%)	Quantidade	Falso Negativo (%)	
img_teste_0	1	99	1	100	0	-	0	-	
img_teste_1	0	-	1	100	0	-	1	Drone não detectado	
img_teste_2	1	98	1	100	0	-	0	-	
img_teste_3	1	99	1	100	0	-	0	-	
img_teste_4	0	-	1	100	1	98	1	Drone não detectado	
img_teste_5	1	86	1	100	1	98	0	-	
img_teste_6	1	99	1	100	0	-	0	-	
img_teste_7	1	100	1	100	0	-	0	-	
img_teste_8	1	99	1	100	0	-	0	-	
img_teste_9	0	-	1	100	1	78	0	-	
img_teste_10	1	100	1	100	0	-	0	-	
img_teste_11	1	99	1	100	0	-	0	-	
img_teste_12	1	98	1	100	0	-	0	-	
img_teste_13	1	84	1	100	0	-	0	-	
img_teste_14	1	99	1	100	0	-	0	-	
img_teste_15	1	99	1	100	0	-	0	-	
img_teste_16	1	99	1	100	0	-	0	-	
img_teste_17	1	100	1	100	0	-	0	-	
img_teste_18	1	100	1	100	0	-	0	-	
img_teste_19	1	99	1	100	0	-	0	-	
TOTAL		17		20		3		2	

A.4 Cálculo do resistor R1 na base do transistor TIP120

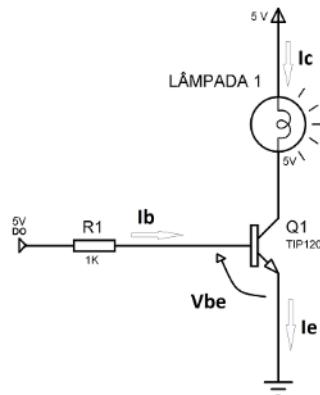


Figura 24 – Circuito

$I_c = 3A$, quando o resistor está ON na configuração de chave. Nas condições ambientes de 25° para esta corrente o transistor possui um ganho (hfe) de 1000. Portanto I_b :

$$I_b = \frac{I_c}{1000} = \frac{3}{1000} = 3mA \quad (\text{A.13})$$

O valor de $I_b = 3mA$ é adequando para a saída digital do microcontrolador do Arduino, que tem limitação de corrente de saída com o máximo de 40 mA. O resistor R_1 necessário para limitar este valor é encontrado através do valor de tensão sobre ele quando a saída do microcontrolador é colocada em nível alto (Volts) e o valor de corrente ao qual será limitado a saída:

$$R_1 = \frac{V_{R1}}{I_b} = \frac{V_{out} - V_{be}}{I_b} = \frac{5 - 2,5}{3mA} = 833\Omega \quad (\text{A.14})$$

Onde $R_1 = 1k\Omega$ é o valor comercial do resistor. Da equação, V_{be} é a tensão entre a base e o emissor do transistor quando ele está ligado em ON. Este valor é fornecido no *datasheet* do fabricante.

A.5 Tabelas para cálculos de taxa de frames

O cálculo a seguir é feito de acordo com os dados da tabela 15:

$$FPS = \frac{\text{Frames}}{\Delta t} = \frac{20}{50,246762 - 2,300931} = 0,4171 \text{ frames por segundo} \quad (\text{A.15})$$

Enquanto o cálculo a seguir corresponde aos dados da tabela 16:

$$FPS = \frac{\text{Frames}}{\Delta t} = \frac{20}{47,100296 - 41,934664} = 3,8717 \text{ frames por segundo} \quad (\text{A.16})$$

O próximo apresenta o cálculo da tabela 17:

$$FPS = \frac{\text{Frames}}{\Delta t} = \frac{20}{47,100296 - 41,934664} = 5,8012 \text{ frames por segundo} \quad (\text{A.17})$$

E por último, o cálculo referente a tabela 18:

$$FPS = \frac{\text{Frames}}{\Delta t} = \frac{20}{53,832744 - 49,445836} = 4,5590 \text{ frames por segundo} \quad (\text{A.18})$$

Tabela 15 – Resultados do Faster RCNN 20000 passos para cálculo de frames por segundo

Faster RCNN 20000 passos	
Frame	Tempo (Data e Hora)
1	2020-03-08 22:55:02.300931
2	2020-03-08 22:55:05.613728
3	2020-03-08 22:55:08.144472
4	2020-03-08 22:55:10.555469
5	2020-03-08 22:55:12.974162
6	2020-03-08 22:55:15.563130
7	2020-03-08 22:55:17.938301
8	2020-03-08 22:55:20.297363
9	2020-03-08 22:55:22.737308
10	2020-03-08 22:55:25.246521
11	2020-03-08 22:55:27.867263
12	2020-03-08 22:55:30.208297
13	2020-03-08 22:55:32.591522
14	2020-03-08 22:55:34.986728
15	2020-03-08 22:55:37.362134
16	2020-03-08 22:55:39.832085
17	2020-03-08 22:55:42.601162
18	2020-03-08 22:55:45.328406
19	2020-03-08 22:55:47.824853
20	2020-03-08 22:55:50.246762

Tabela 16 – Resultados do Mobilenet 10000 passos para cálculo de frames por segundo

SSD Mobilenet 10000 passos	
Frame	Tempo (Data e Hora)
1	2020-03-09 17:12:41.934664
2	2020-03-09 17:12:43.317314
3	2020-03-09 17:12:43.566357
4	2020-03-09 17:12:43.814947
5	2020-03-09 17:12:44.053025
6	2020-03-09 17:12:44.312383
7	2020-03-09 17:12:44.563141
8	2020-03-09 17:12:44.797522
9	2020-03-09 17:12:45.001394
10	2020-03-09 17:12:45.232591
11	2020-03-09 17:12:45.376900
12	2020-03-09 17:12:45.572173
13	2020-03-09 17:12:45.743496
14	2020-03-09 17:12:45.926117
15	2020-03-09 17:12:46.118948
16	2020-03-09 17:12:46.360331
17	2020-03-09 17:12:46.546495
18	2020-03-09 17:12:46.760405
19	2020-03-09 17:12:46.923328
20	2020-03-09 17:12:47.100296

Tabela 17 – Resultados do Mobilenet 20000 passos para cálculo de frames por segundo

SSD Mobilenet 20000 passos	
Frame	Tempo (Data e Hora)
1	2020-03-08 19:07:22.853919
2	2020-03-08 19:07:26.353762
3	2020-03-08 19:07:26.461481
4	2020-03-08 19:07:26.563207
5	2020-03-08 19:07:26.688869
6	2020-03-08 19:07:26.790596
7	2020-03-08 19:07:26.929807
8	2020-03-08 19:07:27.031315
9	2020-03-08 19:07:27.166834
10	2020-03-08 19:07:27.299515
11	2020-03-08 19:07:27.503001
12	2020-03-08 19:07:27.640603
13	2020-03-08 19:07:27.760277
14	2020-03-08 19:07:27.885409
15	2020-03-08 19:07:28.022677
16	2020-03-08 19:07:28.135740
17	2020-03-08 19:07:28.260332
18	2020-03-08 19:07:28.401439
19	2020-03-08 19:07:28.524745
20	2020-03-08 19:07:28.655162

Tabela 18 – Resultados do Mobilenet 30000 passos para cálculo de frames por segundo

SSD Mobilenet 30000 passos	
Frame	Tempo (Data e Hora)
1	2020-02-28 22:23:49.445836
2	2020-02-28 22:23:51.319372
3	2020-02-28 22:23:51.473026
4	2020-02-28 22:23:51.629605
5	2020-02-28 22:23:51.754489
6	2020-02-28 22:23:51.876016
7	2020-02-28 22:23:52.008056
8	2020-02-28 22:23:52.130970
9	2020-02-28 22:23:52.239713
10	2020-02-28 22:23:52.347877
11	2020-02-28 22:23:52.471207
12	2020-02-28 22:23:52.585937
13	2020-02-28 22:23:52.724956
14	2020-02-28 22:23:52.878509
15	2020-02-28 22:23:53.010011
16	2020-02-28 22:23:53.133656
17	2020-02-28 22:23:53.241395
18	2020-02-28 22:23:53.349701
19	2020-02-28 22:23:53.612958
20	2020-02-28 22:23:53.832744

APÊNDICE B – Diagramas

B.1 Modelo mecânico do rastreador no SolidWorks

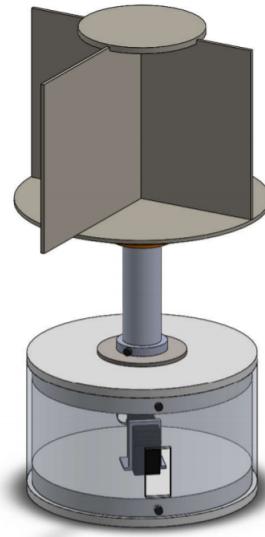


Figura 25 – Projeção Gráfica



Figura 26 – Vista explodida

B.2 Circuito elétrico do sistema

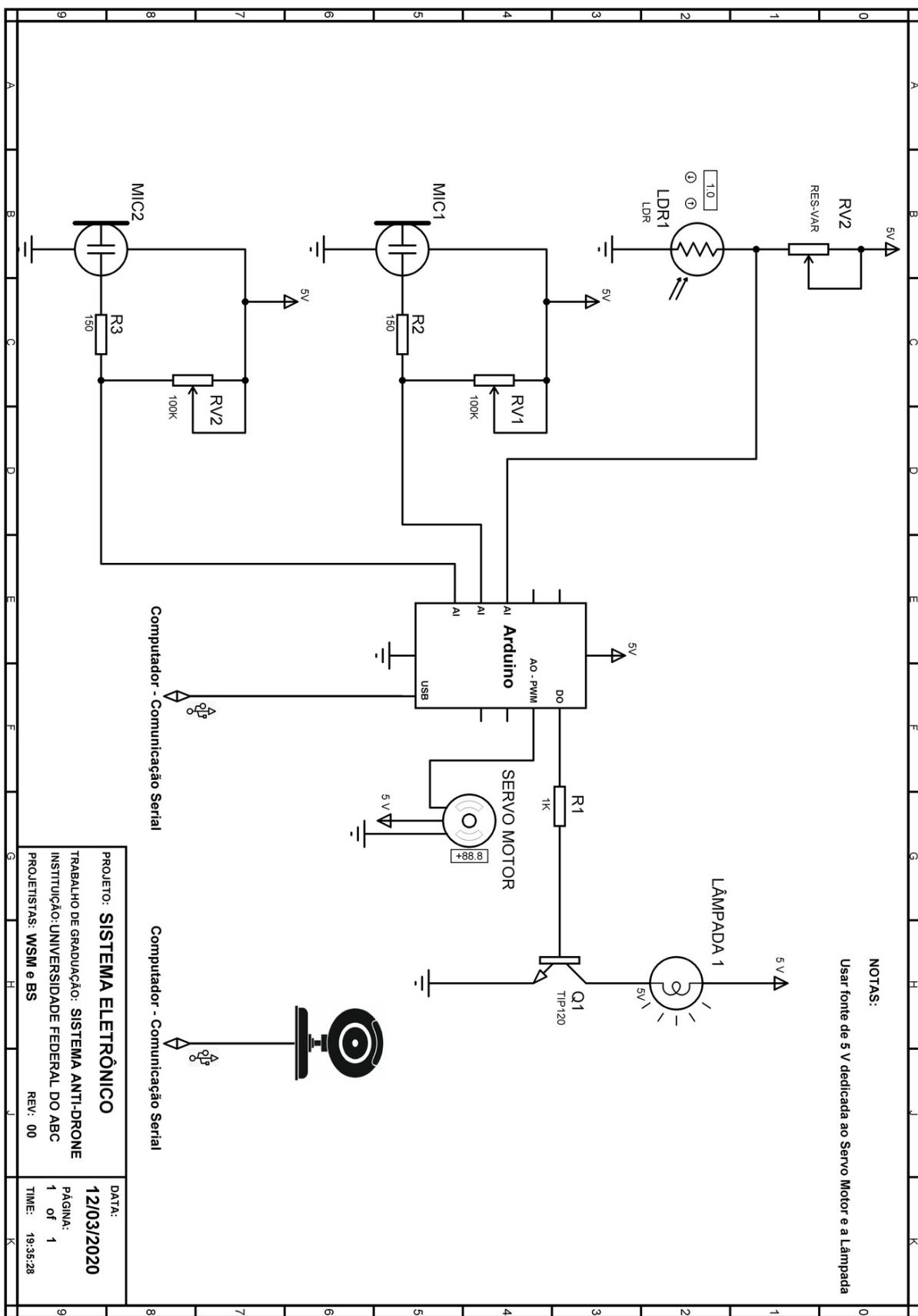


Figura 27 – Circuito elétrico implementado

B.3 Diagrama de fluxo funcional de todo o sistema

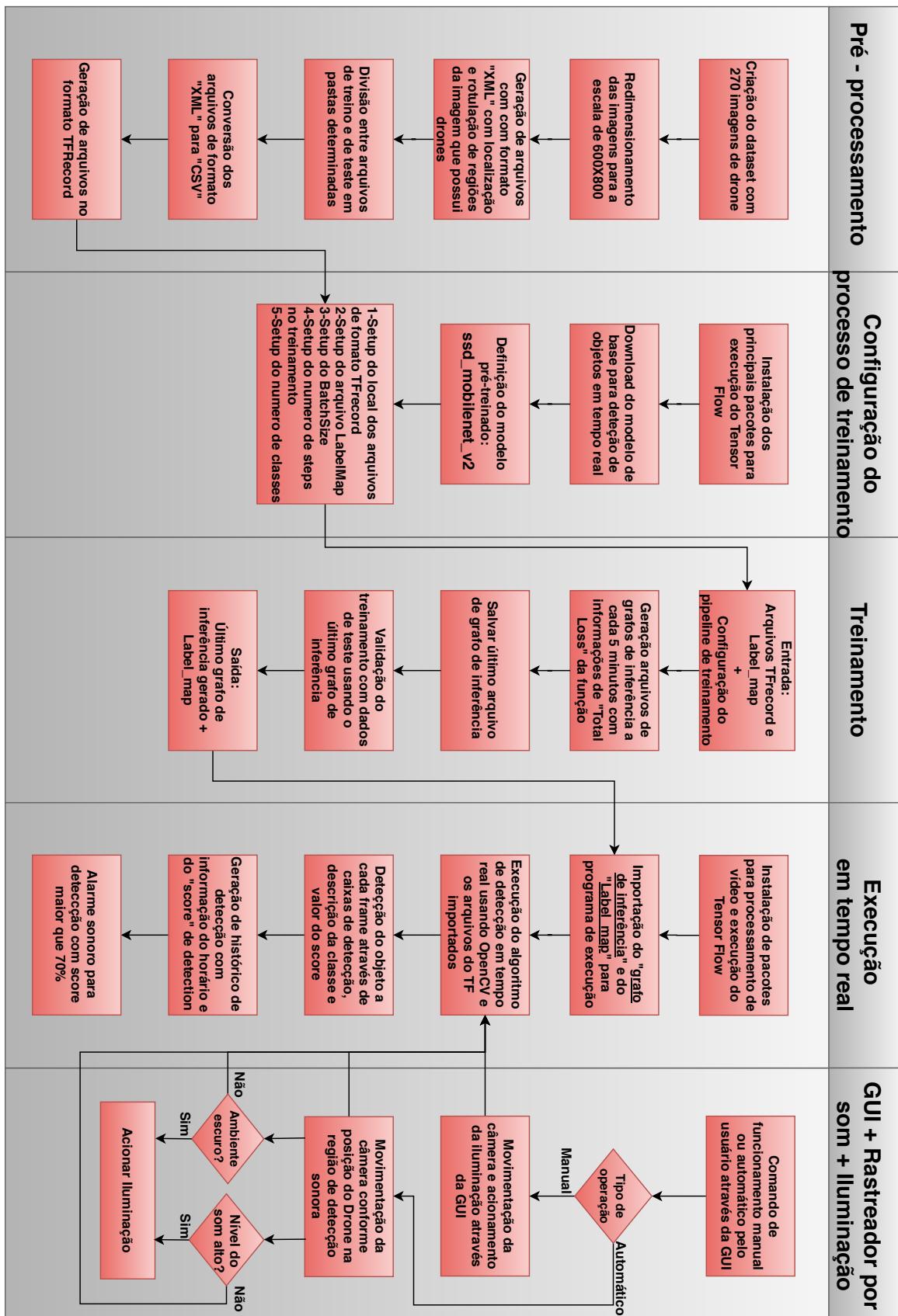


Figura 28 – Diagrama funcional

APÊNDICE C – Códigos

C.1 Código da Interface implementado em Python

```

1 """*****Programa para a execução da interface gráfica de operação do
   ↵ sistema*****"""
2
3 #Importação das principais biblioteca e módulos necessário para o código
4 from tkinter import *      #Biblioteca para criação de interface gráfica
5 import tkinter.messagebox
6 import os                  #Módulo padrão para usar as funcionalidades
   ↵ dependentes do sistema operacional
7 import serial              #Biblioteca para realizar a comunicação
   ↵ serial
8 import subprocess           #Módulo para executar o código como subprocesso
   ↵ para não interferir no principal
9 import sys                 #Módulo fornece recursos usadoss para
   ↵ manipular diferentes partes do ambiente de tempo de execução do
   ↵ Python.
10
11
12 #Comunicação serial - COM9 pode ser modificado de acordo com a porta
   ↵ serial usada no computador
13 ser = serial.Serial('COM9', 115200, timeout=.1)
14
15 #Execução da biblioteca padrão "Tkinter" do python
16 principal = Tk()
17 principal.title("Anti-Drone")
18
19 #Definição das funções dos botões
20 def botao_liga():
21     subprocess.Popen([sys.executable, "DeteccaoTG_SSD_22_02_Detec4_30000.
   ↵ py"]) # Call subprocess
22
23 def botao_gera():
24     os.system('historico.txt')
25
26 def botao_opcao(op):
27     if op == "a":
28         ser.write(b'a')      #comando para Arduino via comunicação serial
29
30     else:
31         ser.write(b'm')      #comando para Arduino via comunicação serial
32

```

```
33 def botao_horario():
34     ser.write(b'z')          #comando para Arduino via comunicação serial
35
36 def botao_antihorario():
37     ser.write(b'x')          #comando para Arduino via comunicação serial
38
39 def botao_liga_ilu():
40     ser.write(b'l')          #comando para Arduino via comunicação serial
41
42 def botao_desliga_ilu():
43     ser.write(b'd')          #comando para Arduino via comunicação serial
44
45
46 #Definição dos atributos textuais e visuais dos títulos interface
47     ↪ gráfica
47 titulo_1 = Label(principal, text = "Sistema de vigilância Anti-Drone",
48     ↪ font=('Calibri','12','bold'))
48 titulo_1.grid(row = 0, column = 0, columnspan = 2)
49
50 titulo_2 = Label(principal, text = "Selecione o tipo de funcionamento:",
51     ↪ font=('Calibri','10','bold'))
51 titulo_2.grid(row = 2, column = 0, columnspan = 2)
52
53 titulo_3 = Label(principal, text = "Selecione o movimento gradual (modo
54     ↪ manual):", font=('Calibri','10','bold'))
54 titulo_3.grid(row = 4, column = 0, columnspan = 2)
55
56 titulo_4 = Label(principal, text = "Ligar ou desligar lâmpada (modo
57     ↪ manual):", font=('Calibri','10','bold'))
57 titulo_4.grid(row = 6, column = 0, columnspan = 2)
58
59 #Definição dos comandos e atributos textuais e visuais dos botões da
60     ↪ interface gráfica
60 botao_liga = Button(principal, text = "Ligar camera", padx = 8, fg =
61     ↪ white", bg = "green", command = botao_liga)
61 botao_liga.grid(row = 1, column = 0)
62
63 botao_gera= Button(principal, text = "Histórico", fg = "white", bg =
64     ↪ green", command = botao_gera)
64 botao_gera.grid(row = 1, column = 1)
65
66 botao_aut = Button(principal, text = "Automático", padx = 28, pady = 10,
67     ↪ fg = "white", bg = "black", command = lambda: botao_opcao("a"))
67 botao_aut.grid(row = 3, column = 0)
68
69 botao_man = Button(principal, text = "Manual", padx = 28, pady = 10, fg
70     ↪ = "white", bg = "gray", command = lambda: botao_opcao("m"))
```

```

70 botao_man.grid(row = 3, column = 1)
71
72 botao_antihorario = Button(principal, text = "<<<", padx = 28, pady =
    ↪ 10, fg = "white", bg = "gray", command = botao_antihorario)
73 botao_antihorario.grid(row = 5, column = 0)
74
75 botao_horario = Button(principal, text = ">>>", padx = 28, pady = 10, fg
    ↪ = "white", bg = "gray", command = botao_horario)
76 botao_horario.grid(row = 5, column = 1)
77
78 botao_liga_ilu = Button(principal, text = "Iluminacao_ON", padx = 28,
    ↪ pady = 10, fg = "white", bg = "orange", command = botao_liga_ilu)
79 botao_liga_ilu.grid(row = 7, column = 0)
80
81 botao_desliga_ilu = Button(principal, text = "Iluminacao_OFF", padx =
    ↪ 28, pady = 10, fg = "white", bg = "blue", command =
    ↪ botao_desliga_ilu)
82 botao_desliga_ilu.grid(row = 7, column = 1)
83
84 principal.mainloop()

```

C.2 Código do Rastreador de som implementado em Arduino

```

1 //Bibliotecas
2 #include <Servo.h>                                //Inclui a biblioteca do
    ↪ Servo
3 Servo servoBase;                                    //Inclui um objeto Servo
    ↪ para controlar o ServoMotor
4
5 /*Janela de amostragem com comprimento para amostra em mS - Necessaria
    ↪ para obter
6 valores de pico nas ondas sonoras emitidas pelo Drone. */
7 const int janelaDeAmostra = 40;
8 const int LDR = A2;                                 //Pino para leitura analogica
    ↪ correspondente a iluminacao
9 const int lampada = 12;                            //Pino para acionamento da lmpada -
    ↪ Chaveada por transistor TIP120
10 unsigned int amostra;                            //Variavel para armazenar a somatoria dos
    ↪ valores de pico do sinal provenientes dos MICS
11 float diferenca;                               //Variavel para armazenar valor da
    ↪ diferencias de valor analogico entre A0 e A1
12 int tolerancia = 2;                            //Variavel de ajuste do valor minimo de
    ↪ diferenca para executar a movimentcao do servo
13 //int pos = 90;                                //Variavel para armazenar posiccao do
    ↪ servo-motor
14 int servoBasePos = 90;                          //Variavel para armazenar do Servo
15 int luz;                                     //Variavel para armazenar informacoes da

```

```
    ↳ intensidade luminosa
16 float nivelA0;           //Variavel para armazenar o nivel do
    ↳ sinal analogico em A0
17 float nivelA1;           //Variavel para armazenar o nivel do
    ↳ sinal analogico em A1
18 float soma_media_0 = 0;   //Variavel para armazenar valor da soma
    ↳ dos sinais em A0 para calculo das mdias
19 float soma_media_1 = 0;   //Variavel para armazenar soma dos sinais
    ↳ em A0 para calculo das mdias
20 float media_0 = 0;        //Variavel para armazenar mdia dos sinais
    ↳ para minimizar ruidos
21 float media_1 = 0;        //Variavel para armazenar a mdia dos
    ↳ sinais para minimizar ruidos
22 char serial;             //Variavel para armazenar valores
    ↳ proveniente da entrada serial
23 //OBS: Os pinos das entradas dos MICs estao descritos na funcao
    ↳ intensidadeSinal(int pinAI)
24 //*****Void Setup
    ↳ *****
25 void setup()
{
26     Serial.begin(9600);      //Comunicacao serial para a verificacao
27     servoBase.attach(3);     //Pino correspondente ao servo
28     pinMode(12, OUTPUT);
29     delay(50);
30 }
31 //*****Void Loop - Loop principal
    ↳ *****
32 void loop() {
33     switch (serial) {
34         case 'm':
35             manual();
36             break;
37
38         case 'a':
39             automatico();
40             break;
41
42         default:
43             automatico();
44             break;
45             break;
46     }
47 }
48 //-----Funcoes principais
    ↳ -----
49 //-----Automatico - rastreador
    ↳ -----
```

```
50 autom_ilum();                                //Verificacao da luz ambiente
51
52 soma_media_0 = 0;                            //reset das mdias
53 soma_media_1 = 0;                            //reset das mdias
54
55 for (int i = 0; i < 2; i++) {
56     nivelA1 = intensidadeSinal(1);
57     soma_media_1 += nivelA1;
58     nivelA0 = intensidadeSinal(0);
59     soma_media_0 += nivelA0;
60 }
61
62 media_0 = ((soma_media_0) / 2);
63 media_1 = ((soma_media_1) / 2);
64
65
66 diferenca = media_1 - media_0 * 1.3;
67
68 //Caso em que microfone em A0 recebe maior intensidade sonora
69 if (diferenca > tolerancia and servoBasePos > 6) {
70     servoBasePos -= 5;
71     servoBase.write(servoBasePos);           //Movimentacao do
    ↵ servo no sentido anti horario
72     delay(10);
73 }
74 //Caso em que microfone em A1 recebe maior intensidade sonora
75 else if ((diferenca < 0) and (diferenca * (-1) > tolerancia) and
    ↵ servoBasePos < 174) {
76     servoBasePos += 5;
77     servoBase.write(servoBasePos);           //Movimentacao do
    ↵ servo no sentido horario
78     delay(10);
79 }
80 //Verificacao da porta serial para sair da rotina
81 if (Serial.available() > 0) {
82     serial = Serial.read();
83     break;
84 }
85 }
86 }
87 //-----Modo manual
    ↵ -----
88 void manual() {
89     while (1) {
90         if (Serial.available() > 0) {          //L o ltimo byte
    ↵ do buffer de entrada serial
91             serial = Serial.read();
```

```
92     if (serial == 'a' or serial == 'm') {
93         break;
94     }
95     else if (serial == 'x' and servoBasePos < 178) { //Movimentacao
96         ↵ horaria
97         servoBasePos = servoBasePos + 5;
98         servoBase.write(servoBasePos);
99         delay(10);
100    }
101    else if (serial == 'z' and servoBasePos > 2) { //Movimentacao
102        ↵ anti horaria
103        servoBasePos = servoBasePos - 5;
104        servoBase.write(servoBasePos);
105        delay(10);
106    }
107    else if (serial == 'l') { //Liga lampada
108        digitalWrite(lampada, HIGH);
109    }
110    else if (serial == 'd') { //Desliga lampada
111        digitalWrite(lampada, LOW);
112    }
113 }
114 //-----Medicao de intensidade
115 float intensidadeSinal(int pinAI) { //AI corresponde a
116     ↵ entrada analogica que esta inserido o microfone
117     unsigned long tempo = millis(); //Valor inicial do
118     ↵ intervalo de amostra - A ser comparado
119     float picoAPico = 0; //Nivel de pico a
120     ↵ pico
121
122     unsigned int sinalMax = 0; //Valor maximo do
123     ↵ sinal
124     unsigned int sinalMin = 1024; //Valor minimo do
125     ↵ sinal
126
127     while (millis() - tempo < janelaDeAmostra) // coleta dados de
128         ↵ amostra a cada 50 mS
129     {
130         amostra = analogRead(pinAI); //leitura do sinal
131         ↵ do microfone na entrada analogica 1
132         if (amostra < 1024) //Retirada de
133             ↵ leituras desnecessarias
134         {
```

```
128     if (amostra > sinalMax)
129     {
130         sinalMax = amostra;                                // Guardar somente
131         ← valores de nivel maximo na deteccao
132     }
133     else if (amostra < sinalMin)
134     {
135         sinalMin = amostra;                                // Guardar somente
136         ← valores de nivel minimo na deteccao
137     }
138     picoAPico = sinalMax - sinalMin;                      // max - min =
139         ← amplitude de pico a pico
140
141     return picoAPico ;
142 }
143 //-----Accionamento automatico da iluminacao
144 //-----
144 void autom_ilum() {
145     luz = analogRead(LDR);
146     //Condicao baseada na leitura intensidade luminosa e sonora do
147     //ambiente
148     if (luz > 1000 or (intensidadeSinal(0) + intensidadeSinal(1)) > 5) {
149         digitalWrite(lampada, HIGH);
150     }
151     else {
152         digitalWrite(lampada, LOW);
153     }
154     delay(5);
```


Anexos

ANEXO A – Códigos

A.1 Código do Treinamento

Disponível em ([BLOG, 2019](#))

```

1 # -*- coding: utf-8 -*-
2 """DetecacaoDeDrone_Treinamento_TG_final_05_02_20.ipynb
3
4 Automatically generated by Colaboratory.
5
6 Original file is located at
7     https://colab.research.google.com/drive/1
8     ↪ RL0gqbj_Y0q7NXUnV2zuZyM84WbSUStP
9
10 ## Configuração dos principais parâmetros
11
12 A maioria do código que segue é disponibilizado pelos Google e serve de
13     ↪ padrão estável para o treinamento de modelos para detecção de
14     ↪ objetos. O código padrão e funcional foi mantido, porém foi
15     ↪ adicionado ou alterado partes do código para que pudessem se
16     ↪ adequar a aplicação do projeto em questão
17 """
18
19 # Local do repositório no GitHub com o dataset de imagens de drone pré-
20     ↪ processadas
21 repo_url = 'https://github.com/wellingtonsm91/deteccao3_drone_dedicado'
22
23 # Número de passos durante o treinamento - por Época
24 num_steps = 30000
25
26 # Número de passos na etapa de validação
27 num_eval_steps = 200
28
29 #Modelo pré-treinados para realizar o treinamento -- Disponibilizado
30     ↪ pelo dataset COCO
31 #A dno nome do modelo está no site dos desenvolvedores
32 MODELS_CONFIG = {
33     'ssd_mobilenet_v2': {
34         #Acurácia média baixa, velocidade alta
35         'model_name': 'ssd_mobilenet_v2_coco_2018_03_29',
36         'pipeline_file': 'ssd_mobilenet_v2_coco.config',
37         'batch_size': 12
38     },
39 }
```

```
31     'faster_rcnn_inception_v2': {
32         ←         #Acurácia média, velocidade média
33         'model_name': 'faster_rcnn_inception_v2_coco_2018_01_28',
34         'pipeline_file': 'faster_rcnn_inception_v2_pets.config',
35         'batch_size': 12
36     },
37     'rfcn_resnet101': {
38         ←         #Maior acurácia, velocidade lenta
39         'model_name': 'rfcn_resnet101_coco_2018_01_28',
40         'pipeline_file': 'rfcn_resnet101_pets.config',
41         'batch_size': 8
42     }
43
44 # Tipo do modelo a ser selecionado - Selecionado um dos modelos acima
45 selected_model = 'ssd_mobilenet_v2'
46
47 MODEL = MODELS_CONFIG[selected_model]['model_name']
48
49 #Nome do arquivo de pipeline na API de detecção de objeto do tensorflow
50     ← ..
51 pipeline_file = MODELS_CONFIG[selected_model]['pipeline_file']
52
53 # Training batch size fits in Colabe's Tesla K80 GPU memory for selected
54     ← model.
55 batch_size = MODELS_CONFIG[selected_model]['batch_size'] # Tamanho do
56     ← lote influencia na convergência
57
58 """## Criação de uma cópia local do repositório com o Dataset no Github
59     ← - Cópia através do mecanismo "git" de clonagem"""
60
61 # Commented out IPython magic to ensure Python compatibility.
62 import os
63
64 # %cd /content
65 #repo disponibilizado pelo Google - Faz-se um clone
66 repo_dir_path = os.path.abspath(os.path.join('..', os.path.basename(
67     ← repo_url)))
68
69 !git clone {repo_url}
70 # %cd {repo_dir_path}
71 !git pull
72
73 """## Instalação dos pacotes necessários para execução do TensorFlow"""
74
75 # Commented out IPython magic to ensure Python compatibility.
```

```
71 # %cd /content
72 !git clone --quiet https://github.com/tensorflow/models.git #--quiet ->
    ↪ suprime a saída
73
74 !apt-get install -qq protobuf-compiler python-pil python-lxml python-tk
75
76 !pip install -q Cython contextlib2 pillow lxml matplotlib
77
78 !pip install -q pycocotools
79
80 # %cd /content/models/research
81 !protoc object_detection/protos/*.proto --python_out=.
82
83 import os
84 os.environ['PYTHONPATH'] += ':/content/models/research:/content/models/
    ↪ research/slim/'
85
86 !python object_detection/builders/model_builder_test.py
87
88 #Importação e verificação da versão do TensorFlow a ser utilizado
89 import tensorflow as tf
90 print(tf.__version__)
91
92 """## Preparação dos arquivos de formato 'tfrecords' - Utilizados para
    ↪ executar o treinamento do modelo"""
93
94 # Commented out IPython magic to ensure Python compatibility.
95 # %cd {repo_dir_path}
96
97 # Conversão dos arquivos de anotação do tipo XML da pasta de treino para
    ↪ arquivo unico do tipo CVS
98 # Geração do arquivo 'label_map.pbtxt' para o diretório 'data/'
99 !python xml_to_csv.py -i data/images/train -o data/annotations/
    ↪ train_labels.csv -l data/annotations
100
101 # Conversão dos arquivos de anotação do tipo XML da pasta de teste para
    ↪ um arquivo único do tipo CVS
102 !python xml_to_csv.py -i data/images/test -o data/annotations/
    ↪ test_labels.csv
103
104 # Geração do arquivo 'train.record'
105 !python generate_tfrecord.py --csv_input=data/annotations/train_labels.
    ↪ csv --output_path=data/annotations/train.record --img_path=data/
    ↪ images/train --label_map data/annotations/label_map.pbtxt
106
107 # Geração do arquivo 'test.record'
```

```
108 !python generate_tfrecord.py --csv_input=data/annotations/test_labels.  
    ↪ csv --output_path=data/annotations/test.record --img_path=data/  
    ↪ images/test --label_map data/annotations/label_map.pbtxt  
109  
110 #atribuição dos arquivos convertidos para suas respectivas variáveis  
111 test_record_fname = '/content/deteccao3_drone_dedicado/data/annotations/  
    ↪ test.record'  
112 train_record_fname = '/content/deteccao3_drone_dedicado/data/annotations/  
    ↪ /train.record'  
113 label_map_pbtxt_fname = '/content/deteccao3_drone_dedicado/data/  
    ↪ annotations/label_map.pbtxt'  
114  
115 """## Download no modelo pre-treinado - Selecionado anteriormente"""  
116  
117 # Commented out IPython magic to ensure Python compatibility.  
118 #De acordo com o modelo pré- treinado selecionado anteriormente  
119 # %cd /content/models/research  
120 import os  
121 import shutil  
122 import glob  
123 import urllib.request  
124 import tarfile  
125 MODEL_FILE = MODEL + '.tar.gz'  
126 DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/  
    ↪ '  
127 DEST_DIR = '/content/models/research/pretrained_model'  
    ↪ #Diretório de destino  
128  
129 if not (os.path.exists(MODEL_FILE)):  
130     urllib.request.urlretrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)  
    ↪ #Endereço do arquivo a ser baixado  
131  
132 tar = tarfile.open(MODEL_FILE)  
133 tar.extractall()  
    ↪ #Extrair o conteúdo do arquivo  
134 tar.close()  
135  
136 os.remove(MODEL_FILE)  
137 if (os.path.exists(DEST_DIR)):  
138     shutil.rmtree(DEST_DIR)  
139 os.rename(MODEL, DEST_DIR)  
140  
141 #Mostrar o conteúdo da pasta onde se encontra o modelo  
142 !echo {DEST_DIR}  
143 !ls -alh {DEST_DIR}  
144
```

```
145 #Caminho para arquivo de sintonia fina do modelo a ser gerado após o
   ↪ treinamento
146 fine_tune_checkpoint = os.path.join(DEST_DIR, "model.ckpt")
147 fine_tune_checkpoint
148
149 """## Configuração do PIPELINE de treino"""
150
151 import os
152 pipeline_fname = os.path.join('/content/models/research/object_detection
   ↪ /samples/configs/', pipeline_file)
153
154 assert os.path.isfile(pipeline_fname), '{} not exist'.format(
   ↪ pipeline_fname) #Garantir uma condição para continuar a execução
   ↪ do código
155
156 #Função para definir o numero de classes com base no arquivo com
   ↪ extensão :pbtxt, utilizado como argumento
157 def get_num_classes(pbtxt_fname):
158     from object_detection.utils import label_map_util
159     label_map = label_map_util.load_labelmap(pbtxt_fname)
160     categories = label_map_util.convert_label_map_to_categories(
161         label_map, max_num_classes=90, use_display_name=True)
162     category_index = label_map_util.create_category_index(categories)
163     return len(category_index.keys())
164
165 import re
   ↪           #Importação de expressões regulares
166
167 num_classes = get_num_classes(label_map_pbtxt_fname)
   ↪           #Número de Classes
168 with open(pipeline_fname) as f:
169     s = f.read()
170 with open(pipeline_fname, 'w') as f:
171
172     # fine_tune_checkpoint
173     s = re.sub('fine_tune_checkpoint: ".*?"',
174                'fine_tune_checkpoint: "{}".format(fine_tune_checkpoint)
   ↪ , s)      #Substituição do ponto de resgistro da sintonia fina
175
176     #Arquivos "tfrecord" - treino e teste
177     s = re.sub(
178         '(input_path: ".*?)(train.record)(.*?)"', 'input_path: "{}".
   ↪ format(train_record_fname), s)
179     s = re.sub(
180         '(input_path: ".*?)(val.record)(.*?)"', 'input_path: "{}".
   ↪ format(test_record_fname), s)
181
```

```
182     # Caminho para arquivo "label_map"
183     s = re.sub(
184         'label_map_path: ".*?"', 'label_map_path: "{}"'.format(
185             label_map_pbtxt_fname), s)
186
187     # Setup do tamanho do lote (batch size) para o treinamento.
188     s = re.sub('batch_size: [0-9]+',
189                 'batch_size: {}'.format(batch_size), s)
190
191     # Setup do número de passos
192     s = re.sub('num_steps: [0-9]+',
193                 'num_steps: {}'.format(num_steps), s)
194
195     # Setup do número de classes
196     s = re.sub('num_classes: [0-9]+',
197                 'num_classes: {}'.format(num_classes), s)
198
199 f.write(s)
200
201 !cat {pipeline_fname}
202
203 #Loca do treino
204 model_dir = 'training/'
205 #Remoção do conteúdo na saída do modelo
206 !rm -rf {model_dir}
207 os.makedirs(model_dir, exist_ok=True)
208
209 """## Execução do Tensorboard"""
210
211 #Download dos arquivos necessário para a execução do visualizado
212     # TensorBoard
213 !wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
214 !unzip -o ngrok-stable-linux-amd64.zip
215
216 #Definição do diretório a ser monitorado pelo TensorBoard
217 LOG_DIR = model_dir
218 get_ipython().system_raw(
219     'tensorboard --logdir {} --host 0.0.0.0 --port 6006 &',
220     .format(LOG_DIR))
221
222 get_ipython().system_raw('./ngrok http 6006 &')
223
224 """## Link para executar o TensorBoard"""
225 ! curl -s http://localhost:4040/api/tunnels | python3 -c \
226     "import sys, json; print(json.load(sys.stdin)[‘tunnels’][0][‘
227         public_url’])"
```

```
226
227 """## Treinamento do modelo"""
228
229 !python /content/models/research/object_detection/model_main.py \
230     --pipeline_config_path={pipeline_fname} \
231     --model_dir={model_dir} \
232     --alsologtostderr \
233     --num_train_steps={num_steps} \
234     --num_eval_steps={num_eval_steps}
235
236 #Verificação do conteúdo gerado após o treinamento - arquivos do tipo:
#      ↪ Grafos de Inferência
237 !ls {model_dir}
238
239 """## Exportação do Grafo de Inferência Treinado
240 Após o treinamento ter sido completo, deve-se extrair o último grafo de
#      ↪ inferência treinado a fim de usá-lo posteriormente na detecção de
#      ↪ objetos
241 """
242
243 import re
244 import numpy as np
245
246 output_directory = './fine_tuned_model'
247
248 lst = os.listdir(model_dir)
249 lst = [l for l in lst if 'model.ckpt-' in l and '.meta' in l]
250 steps=np.array([int(re.findall('\d+', l)[0]) for l in lst])
251 last_model = lst[steps.argmax()].replace('.meta', '')
252
253 last_model_path = os.path.join(model_dir, last_model)
#      ↪ Variável com o atalho para o último modelo treinado
254 print(last_model_path)
255 !python /content/models/research/object_detection/export_inference_graph \
#      ↪ .py \
256     --input_type=image_tensor \
257     --pipeline_config_path={pipeline_fname} \
258     --output_directory={output_directory} \
259     --trained_checkpoint_prefix={last_model_path}
260
261 #Conteúdo do diretório de saída do treino - Deve conter o último Grafo
#      ↪ de Inferência Treinado
262 !ls {output_directory}
263
264 """# Download do último modelo - com extensão '.pb'"""
265
266 import os
```

```

267
268 pb_fname = os.path.join(os.path.abspath(output_directory), "  

   ↪ frozen_inference_graph.pb")
269 assert os.path.isfile(pb_fname), '{} not exist'.format(pb_fname)
270
271 #Visualização do conteúdo na pasta
272 !ls -alh {pb_fname}
273
274 """### Option2 : Download the '.pb' file directly to your local file  

   ↪ system
275 This method may not be stable when downloading large files like the  

   ↪ model '.pb' file. Try **option 1** instead if not working.
276 """
277
278 from google.colab import files
279 files.download(pb_fname)
280
281 """### Download the 'label_map.pbtxt' file"""
282
283 # download para o PC
284 from google.colab import files
285 files.download(label_map_pbtxt_fname)

```

A.2 Código da Detecção

Disponível em (??)

```

1 """Código para execução em tempo real para modelo treinado, deve ser  

   ↪ executado dentro da
2 pasta object_detection na pasta do Framework TensorFlow disponibilizada  

   ↪ em https://github.com/tensorflow/tensorflow"""
3 #Importação das principais bibliotecas para a execução do modelo  

   ↪ treinado para detecção
4 import numpy as np
5 import os
6 import six.moves.urllib as urllib
7 import sys
8 import tarfile
9 import tensorflow as tf
10 import zipfile
11
12 #Importação dos módulos necessário para a execução do código
13 from distutils.version import StrictVersion
14 from collections import defaultdict
15 from io import StringIO
16 from matplotlib import pyplot as plt
17 from PIL import Image

```

```
18
19 #Importação
20 sys.path.append("..")
21 from object_detection.utils import ops as utils_ops
22
23 #Aviso para realizar o upgrade do Tensorflow caso este esteja na versão
24     ↪ anterior a 1.12.0
25 if StrictVersion(tf.__version__) < StrictVersion('1.12.0'):
26     raise ImportError('Please upgrade your TensorFlow installation to v1
27     ↪ .12.*.')
28
29 #%%matplotlib inline
30 #Verificação da versão do Tensorflow instalada no PC
31 print(tf.__version__)
32
33 #Importação dos modulos para visualização dos resultados do grafo de
34     ↪ inferência
35 from utils import label_map_util
36 from utils import visualization_utils as vis_util
37
38 #Definição do nome e dos atalhos do modelo e do arquivo label_map
39     ↪ dentro da pasta "object_detection"
40 MODEL_NAME = 'inference_graph'
41 PATH_TO_FROZEN_GRAPH = MODEL_NAME + '/'
42     ↪ frozen_inference_graph_SSD_22_02_Detec4_30000.pb'
43 PATH_TO_LABELS = 'training/label_map_SSD_22_02_Detec4_30000.pbtxt'
44
45 #Atribuição do grafo de inferência a variável "detection_graph"
46 detection_graph = tf.Graph()
47 with detection_graph.as_default():
48     od_graph_def = tf.GraphDef()
49     with tf.gfile.GFile(PATH_TO_FROZEN_GRAPH, 'rb') as fid:
50         serialized_graph = fid.read()
51         od_graph_def.ParseFromString(serialized_graph)
52         tf.import_graph_def(od_graph_def, name='')
53 #criação de um indice de categoria a partir do arquivo "
54     ↪ label_map_SSD_22_02_Detec4_30000.pbtxt"
55 category_index = label_map_util.create_category_index_from_labelmap(
56     ↪ PATH_TO_LABELS, use_display_name=True)
57
58 #Definição da função para a execução do modelo em uma única imagem
59 #correspondente a cada frame da captura de video em tempo real
60 def run_inference_for_single_image(image, graph):
61     if 'detection_masks' in tensor_dict:
62
63         detection_boxes = tf.squeeze(tensor_dict['detection_boxes'], [
64             ↪ [0]])
```

```
56     detection_masks = tf.squeeze(tensor_dict['detection_masks'],  
57     [0])  
  
58     real_num_detection = tf.cast(tensor_dict['num_detections'][0],  
59     tf.int32)  
60     detection_boxes = tf.slice(detection_boxes, [0, 0], [  
61     real_num_detection, -1])  
62     detection_masks = tf.slice(detection_masks, [0, 0, 0], [  
63     real_num_detection, -1, -1])  
64     detection_masks_reframed = utils_ops.  
65     reframe_box_masks_to_image_masks(                                detection_masks,  
66     detection_boxes, image.shape[0], image.shape[1])  
67     detection_masks_reframed = tf.cast(                                tf.greater(  
68     detection_masks_reframed, 0.5), tf.uint8)  
  
69     tensor_dict['detection_masks'] = tf.expand_dims(  
70     detection_masks_reframed, 0)  
71     image_tensor = tf.get_default_graph().get_tensor_by_name(  
72     'image_tensor:0')  
  
73     output_dict = sess.run(tensor_dict, feed_dict={image_tensor: np.  
74     expand_dims(image, 0)})  
  
75     output_dict['num_detections'] = int(output_dict['num_detections',  
76     [0])  
77     output_dict['detection_classes'] = output_dict[  
78     'detection_classes'][0].astype(np.uint8)  
79     output_dict['detection_boxes'] = output_dict['detection_boxes'][0]  
80     output_dict['detection_scores'] = output_dict['detection_scores'][0]  
81     if 'detection_masks' in output_dict:  
82         output_dict['detection_masks'] = output_dict['detection_masks',  
83     [0]  
84     return output_dict  
  
#Importação da biblioteca "opencv" desenvolvida para aplicativos na área  
    → de Visão computacional  
import cv2  
cap = cv2.VideoCapture(0)  
"""*****Adaptação: Parte desenvolvida para atender os requisitos  
    → deste projeto*****"""  
#Importação de recursos necessário para geração do alarme e do histórico  
    → de detecção  
import datetime      #Informação da data e hora vigente  
from playsound import playsound #módulo para execução de arquivos de  
    → áudio  
import subprocess      #Recurso para executar o código como subprocesso  
    → para não interferir no principal
```

```
85 #Abre o arquivo .txt para armazenar informações da detecção. O arquivo
     → está na mesma pasta deste código de execução
86 f = open("historico.txt", "w+")
87 """*****
88 #Execução do grafo de inferência em tempo real - Loop infinito
89 try:
90     with detection_graph.as_default():
91         with tf.Session() as sess:
92
93             ops = tf.get_default_graph().get_operations()
94             all_tensor_names = {output.name for op in ops for output
95             → in op.outputs}
96             tensor_dict = {}
97             for key in [
98                 'num_detections', 'detection_boxes', 'detection_scores
99             → ,
100                 'detection_classes', 'detection_masks'
101             ]:
102                 tensor_name = key + ':0'
103                 if tensor_name in all_tensor_names:
104                     tensor_dict[key] = tf.get_default_graph().
105                     → get_tensor_by_name(tensor_name)
106
107                     while True:
108                         ret, image_np = cap.read()          #Captura de imagem
109                         """*****Adaptação: Parte desenvolvida para atender os requisitos
110                         → deste projeto*****"""
111                         #Escrever a informação da aplicação, data e hora no
112                         → display
113                         tempo = datetime.datetime.now().strftime("%Y-%m-%d %
114                         → H:%M:%S")
115                         texto = "Sistema anti-drone - " + str(tempo)
116                         org = (10, 20)
117                         thickness = 1
118                         fontScale = 1
119                         color = (0,255,0)
120                         font = cv2.FONT_HERSHEY_COMPLEX_SMALL
121                         image_np = cv2.putText(image_np, texto, org, font,
122                         → fontScale, color, thickness, cv2.LINE_AA)
123                         #print(tempo)
124                         #print(frame)
125
126                         """*****
127                         image_np_expanded = np.expand_dims(image_np, axis=0)
128
129                         output_dict = run_inference_for_single_image(
130                         → image_np, detection_graph)
```

```
123         vis_util.visualize_boxes_and_labels_on_image_array(
124             image_np,
125             output_dict['detection_boxes'],
126             output_dict['detection_classes'],
127             output_dict['detection_scores'],
128             category_index,
129             instance_masks=output_dict.get('detection_masks',
130             ↵ ),
131             use_normalized_coordinates=True,
132             line_thickness=8)
133             cv2.imshow('object_detection', cv2.resize(image_np,
134             ↵ (800, 600)))
135
136 """*****Adaptação: Parte desenvolvida para atender os requisitos deste
137   ↵ projeto*****"""
138
139             #Soar alarme e escrever no histórico para detecções
140             ↵ com score maior que 70%
141             if output_dict['detection_scores'][0] > 0.70 :
142                 probabilidade = output_dict['detection_scores'][0]
143                 ↵ * 100
144                 historico = str(tempo) + " - " + "Drone detectado !
145                 ↵ " + " : " + str(probabilidade) + "%" + '\n'
146                 f.write(historico)
147                 subprocess.Popen([sys.executable, "alarme_script.py
148             ↵ "]) # Abre como subprocesso
149 """
150
151             if cv2.waitKey(25) & 0xFF == ord('q'):      #Condição
152             ↵ para encerrar o laço de execução
153                 f.close()
154                 cap.release()
155                 cv2.destroyAllWindows()
156                 break
157
158
159 except Exception as e:
160     print(e)
161     cap.release()
```