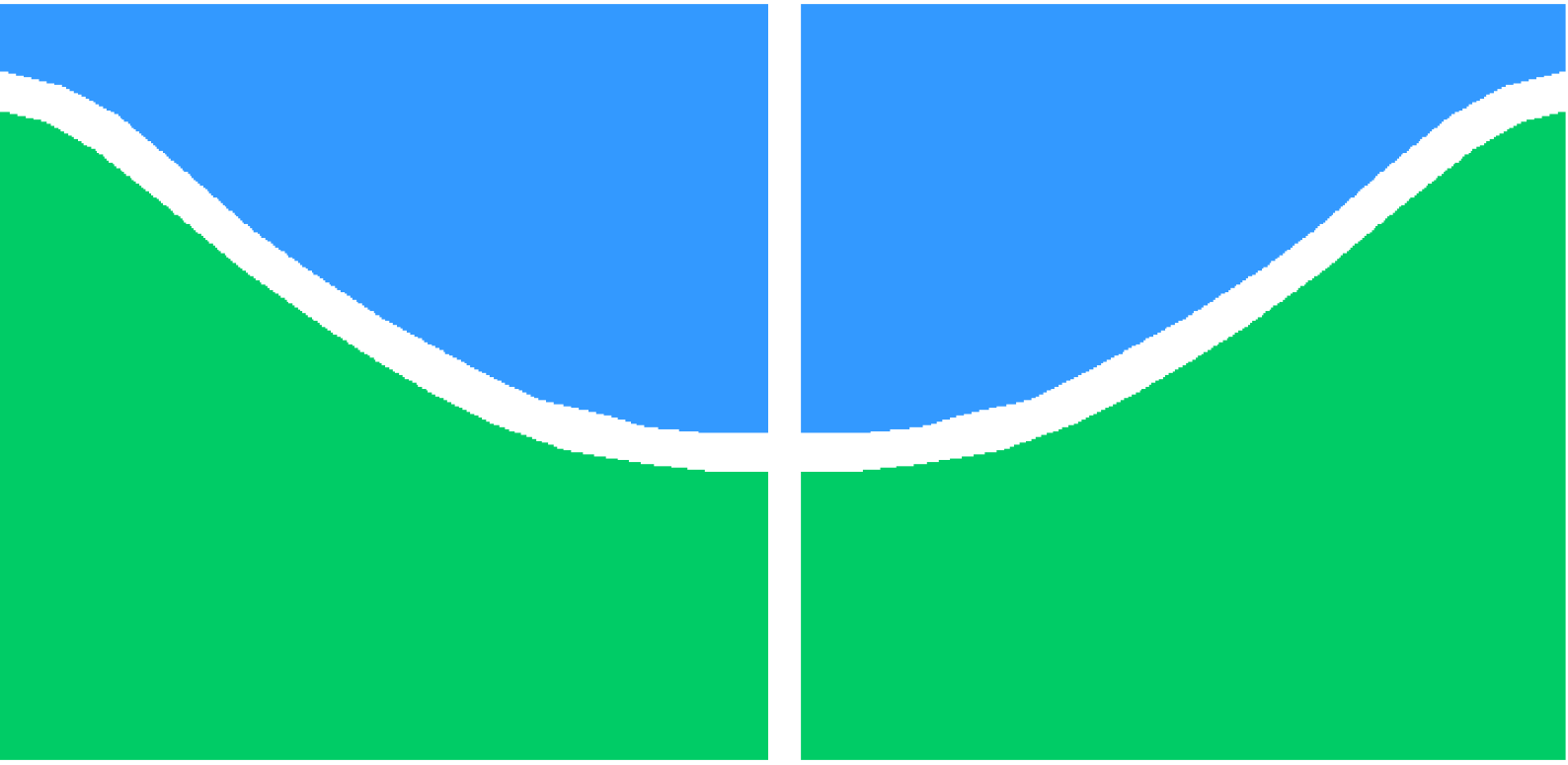


Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Uma análise comparativa de *datasets* para detecção de pedestres

Autor: Elias Bernardo Marques Magalhães

Orientador: Prof. Giovanni Almeida Santos

Brasília, DF

2022

Elias Bernardo Marques Magalhães

Uma análise comparativa de *datasets* para detecção de pedestres

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Giovanni Almeida Santos

Brasília, DF

2022

Elias Bernardo Marques Magalhães
Uma análise comparativa de *datasets* para detecção de pedestres/ Elias Bernardo Marques Magalhães. – Brasília, DF, 2022-
85 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Giovanni Almeida Santos

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2022.

1. Detecção de pedestres. 2. Aprendizado de máquina. I. Prof. Giovanni Almeida Santos. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Uma análise comparativa de *datasets* para detecção de pedestres

CDU 02:141:005.6

Elias Bernardo Marques Magalhães

Uma análise comparativa de *datasets* para detecção de pedestres

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 05 de maio de 2022:

Prof. Giovanni Almeida Santos
Orientador

Prof. Dr. Maurício Serrano
Examinador

Profa. Dra. Milene Serrano
Examinadora

Brasília, DF
2022

Agradecimentos

Agradeço principalmente à minha família, em especial minha mãe e meu irmão, que me deram todo apoio durante toda a minha jornada e luta na UnB. Agradeço também ao meu orientador, o Professor Giovanni, por ter me dado todo o suporte e conhecimento necessário para a realização deste trabalho. Agradeço também a todos os meus amigos, que sempre me ajudaram em momentos difíceis.

Resumo

O interesse em veículos autônomos tem aumentado consideravelmente nos últimos anos. A utilização de veículos autônomos pode alterar consideravelmente o panorama de transporte atual, reduzindo consideravelmente o número de acidentes e aumentando a segurança no trânsito como um todo, visto que mais de 90% dos acidentes são causados por falhas humanas. Nesse sentido, um veículo autônomo precisa conhecer e entender o que acontece ao seu redor para que possa mitigar e prevenir acidentes. Pensando nisso, esse trabalho se propõe a contribuir no tema de veículos autônomos, com ênfase na detecção de pedestres. O objetivo consiste em analisar e avaliar os *datasets* PSU e INRIA utilizando o algoritmo YOLOv3. Também foi proposto um novo *dataset*, capturado a partir de uma câmera veicular, buscando simular o cenário observado por um veículo autônomo, com imagens diurnas e noturnas. O processo realizado nos *datasets* PSU e INRIA Person utilizou o cenário de uso como meio de investigação, objetivando avaliar o desenvolvimento do objetivo, documentando-se o processo de desenvolvimento. Conclui-se então que o algoritmo é capaz de detectar pessoas em imagens, e que o *dataset* proposto pode contribuir para o desenvolvimento de novos modelos, porém com aberturas a melhorias.

Palavras-chaves: Detecção de pedestres. YOLOv3. Dataset.

Abstract

Interest in autonomous vehicles has increased considerably in recent years. The use of autonomous vehicles can considerably change the current transport scenario, considerably reducing the number of accidents and increasing traffic safety as a whole, since more than 90% of accidents are caused by human error. In this sense, an autonomous vehicle needs to know and understand what happens around it so that it can mitigate and prevent accidents. With that in mind, this work proposes to contribute to the theme of autonomous vehicles, with emphasis on pedestrian detection. The objective is to analyze and evaluate the PSU and INRIA datasets using the YOLOv3 algorithm. A new dataset captured from a vehicle camera was also proposed, seeking to simulate the scenario observed by an autonomous vehicle, with day and night images. The process carried out in the PSU and INRIA Person datasets used the usage scenario as a means of investigation, aiming to evaluate the development of the objective, documenting the entire development process. It is concluded that the algorithm is capable of detecting people in images, and that the proposed dataset can contribute to the development of new models, but with openings for improvements.

Key-words: Pedestrian detection. Yolov3. Dataset.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

YOLO	*You only look once*
XGBoost	*eXtreme Gradient Boosting*
SVM	*Support Vector Machine*
PSU	*Pennsylvania State University*
IOU	*Intersection over Union*
HOG	*Histogram of Oriented Gradients*
TP	*True Positive*
TN	*True Negative*
FP	*False Positive*
FN	*False Negative*
SSD	*Solid-State Drive*
GPU	*Graphics Processing Unit*
VAs	Veículos autônomos

Lista de símbolos

$\sum$	Somatório

Sumário

Parte I

Introdução

1 Introdução

Apesar de veículos autônomos (VAs) não ser um tema de estudo recente, ele tem atraído bastante a atenção. Nos anos 80, já apareceram os primeiros veículos autônomos, com destaque para o *NavLab* (DOWLING et al., 1987). O advento computacional e, consequentemente, de novas tecnologias e algoritmos de detecção de objetos, tem acelerado a evolução da área. Segundo Osorio, Heinen e Fortes (2001), os VAs têm chamado a atenção de pesquisadores da área de Inteligência Artificial (IA), principalmente pelos desafios que esse novo domínio propõe, de dar a esses sistemas a capacidade de raciocínio inteligente e de interação com o meio em que estiverem inseridos.

Para Santos (2017), os VAs "*devem ser capazes de operar nas vias existentes atualmente em diferentes condições ambientais e de tráfego e em conjunto com veículos não autônomos*", o que poderia alterar consideravelmente o panorama do transporte atual. De acordo com Cacilo et al. (2015), 93,5% dos acidentes de trânsito são causados por falhas humanas, número que pode ser reduzido substancialmente com a utilização de VAs.

De acordo com a norma J3016 (SAE, 2021), a automação de direção é classificada em níveis que vão de 0 (quando não há automação de condução) à 5 (quando a automação de condução é completa). Ainda segundo ela, os níveis 3 a 5 de direção autônoma exigem que os VAs conheçam o ambiente ao seu redor, de forma que possam mitigar e prevenir acidentes. Nesse sentido, a utilização de algoritmos de detecção de objetos em imagens se mostra importante fator para VAs. Danapal et al. (2020) cita a rede neural *YOLO* como um potencial candidato para a detecção de objetos em tempo real.

Dessa forma, dada a infinitude de temas possíveis de serem trabalhados em relação a VAs, o presente trabalho tem como foco o processo de detecção e identificação de pedestres, de forma que seja possível para um veículo identificar pessoas ao seu redor. Será utilizado o algoritmo YOLOv3 e proposto um *dataset* focado na detecção de pedestres a partir de imagens capturadas por uma câmera veicular, comparando as métricas de precisão média entre o *dataset* proposto e o INRIA e PSU, analisando cada um deles.

1.1 Questão da pesquisa

Segundo Intel (2017), VAs possuem o potencial de salvar mais de 585 mil vidas entre 2035 e 2045, além de reduzir os custos de segurança pública relacionados a acidentes de trânsito em mais de $234 bilhões, o que demonstra a importância de VAs no contexto atual. Nesse sentido, a detecção de pedestres é importante fator para que VAs possam evitar acidentes, garantindo a segurança não somente dos passageiros dos veículos, mas

também dos pedestres ao redor.

Assim, a questão de pesquisa deste trabalho é: **Como detectar pedestres utilizando o algoritmo YOLOv3, e ainda como capturar e preparar um conjunto de dados, documentando todo o processo de preparação dos dados e treinamento, além de comparar as características e resultados de cada um dos *datasets***?

1.2 Objetivos do trabalho

O objetivo geral deste trabalho é comparar *dataset* focados na detecção de pedestres no contexto de veículos autônomos. Além disso, também possui como objetivos específicos:

1. Identificar e avaliar *datasets* já existentes com imagens de pedestres para treinamento e teste do algoritmo;
2. Elaborar um novo *dataset* com imagens capturadas a partir de uma câmera veicular;
3. Documentar o processo de treinamento e avaliação utilizando o algoritmo YOLOv3.

1.3 Organização do trabalho

Este trabalho é dividido em cinco capítulos, sendo eles:

- **Capítulo 1 - Introdução**: Faz uma breve contextualização do tema, apresentando a questão de pesquisa e os objetivos;
- **Capítulo 2 - Fundamentos teóricos**: Apresenta os fundamentos teóricos necessários para a compreensão dos algoritmos e *datasets* utilizados neste trabalho;
- **Capítulo 3 - Metodologia**: Esta seção descreve a metodologia utilizada no processo de desenvolvimento do projeto;
- **Capítulo 4 - Resultados**: Apresenta a documentação do desenvolvimento da pesquisa, além de dos resultados obtidos ao final, e
- **Capítulo 5 - Conclusão**: Apresenta a conclusão referente aos resultados obtidos, além de exibir as considerações para trabalhos futuros.

Parte II

Fundamentos teóricos

2 Fundamentos teóricos

Neste capítulo, são apresentados, de forma breve, os conceitos teóricos que fundamentam este trabalho. Em particular, será descrita a teoria de funcionamento do algoritmo utilizado (YOLOv3), dos dois conjuntos de dados utilizados, INRIA e PSU e das métricas utilizadas.

2.1 YOLOv3

You only look once (Yolo) é um sistema de detecção de objetos em tempo real (REDMON; FARHADI, 2018). Sua terceira versão (YOLOv3) é uma evolução incremental do Yolov2, e utiliza uma variante da *Darknet* (REDMON, 2013–2016), um *framework* de redes neurais de código aberto escrito em C[1].

O algoritmo funciona a partir da aplicação de uma única rede neural numa imagem completa, onde a rede divide a imagem em regiões, prevendo as *caixas delimitadoras*[2] e probabilidades para cada região. As caixas delimitadoras são então ponderadas pelas probabilidades previstas. Segundo Redmon e Farhadi (2018), tais características o tornam muito mais rápido que outros algoritmos como *R-CNN* e *Faster R-CNN*. O funcionamento da rede, de forma geral, é descrito a seguir:

1. No primeiro passo, o sistema divide a imagem de entrada numa grade $S \times S$. Caso o centro de determinado objeto esteja numa célula da grade, tal célula será responsável por detectar o objeto. Esse passo pode ser visto na Figura 2.1

Figura 2.1 – Grade $S \times S$ na imagem de entrada.

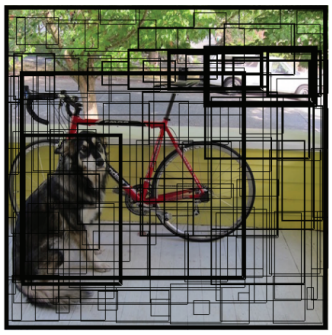
Fonte: Redmon et al. (2016)

[1] Linguagem de programação compilada de propósito geral, estruturada, imperativa, procedural, padronizada pela Organização Internacional para Padronização

[2] Descreve a localização espacial de um objeto numa área retangular

2. Depois, cada célula da grade prevê as caixas delimitadores B, e a confidência para cada uma dessas caixas. Cada caixa delimitadora consiste de cinco predições: x, y, w, h e a confidência. Os valores x, y representam o centro da caixa em relação aos limites da célula da grade, w, h representam a largura (*widht*) e altura (*height*) em relação à toda a imagem, e a confidência representa o *Intersection-Over-Union*[3] (IOU) entre o valor previsto e o valor verdadeiro. Esse passo pode ser visto na Figura 2.2

Figura 2.2 – Caixas delimitadoras e confidência



Fonte: Redmon et al. (2016)

3. As células da grade também preveem as probabilidades de uma classe condicional C, sendo condicionadas à célula da grade que contém um objeto. É previsto apenas um conjunto de probabilidades de classe por célula de grade, independentemente do número de caixas delimitadoras B. O número de classes varia de acordo com o modelo treinado. Esse passo pode ser visto na Figura 2.3

Figura 2.3 – Mapa de probabilidades de classes.

Fonte: Redmon et al. (2016)

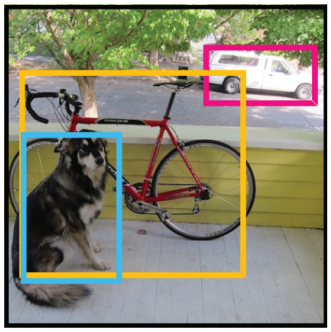
4. Ao se realizar o teste, as probabilidades da classe condicional C e as previsões de confiança de cada caixa delimitadora individual são multiplicadas de acordo com a equação 2.1

[3] Estatística usada para medir a similaridade e diversidade de conjuntos de amostras

$$Pr(Classe_i|Objeto) * Pr(Objeto) * IOU = Pr(Classe_i) * IOU, \quad (2.1)$$

onde Pr representa a predição realizada. Ao final, são dados os valores de confiança de cada classe para cada caixa delimitadora, com as detecções em si, conforme visto na Figura 2.4

Figura 2.4 – Detecções realizada na etapa de testes no YOLO.



Fonte: Redmon et al. (2016)

Vale salientar que essa seção apresentou apenas os conceitos básicos de forma a contextualizar o funcionamento do YOLO. Contudo, ainda existem diversas informações e detalhes a respeito da construção e do funcionamento do algoritmo.

2.2 Datasets INRIA & PSU

O INRIA Person é um *dataset* que foi coletado como parte dos trabalhos realizados por Dalal e Triggs (2005). O *dataset* consiste em amostras positivas e negativas de pessoas, focado na detecção de pedestres, tendo como fonte outros *dataset*s, coleções pessoais e até mesmo imagens do Google. O *dataset* possui anotações, porém dois pontos são fortemente destacados pelo autor:

1. Somente pessoas em pé e com altura (em *pixels*[4]) maior que 100 foram anotadas.
2. As anotações podem não estar corretas.

O segundo ponto é particularmente importante, visto que de fato foi verificado que algumas imagens não estão corretamente anotadas. Isso é um dos pontos a ser mais bem abordado na segunda parte deste trabalho.

Já o *dataset* PSU também é um conjunto de dados focado em pedestres contendo amostras positivas e negativas. No entanto, ao contrário do INRIA Person, ele

[4] Menor parte de uma imagem

apresenta cenários em que os pedestres estão aglomerados entre diferentes veículos e com variações de aparência no ambiente do mundo real, resultando em imagens com pedestres desobstruídos, parcialmente ocluídos, e totalmente ocluídos (THU; SUVONVORN; KARNJANADECHA, 2018). A Figura 2.5 mostra alguns exemplos de amostras positivas do *dataset*, onde é possível perceber multidões e pessoas parcialmente obstruídas.

Figura 2.5 – Amostras positivas do PSU.

Fonte: Thu, Suvonvorn e Karnjanadecha (2018)

Segundo Thu, Suvonvorn e Karnjanadecha (2018), foram capturados dados de pedestres em posições de múltiplas visualizações de acordo com várias posturas, dentre elas: ereto, caminhando, de bicicleta, andando de moto, na posição esquerda, direita, de costas e também com algumas partes obstruídas. Foram utilizados os dispositivos *OPPO A57* e *Samsung Note 5*, e as fotos foram tiradas em regiões de mercados e no campus PSU.

2.3 Métricas

Antes de explicar as métricas utilizadas, faz-se necessário explicar os conceitos de *Verdadeiro positivo, verdadeiro negativo, falso positivo e falso negativo*, visto que as escolhidas para serem utilizadas neste trabalho, descritas a seguir, utilizam tais conceitos, direta ou indiretamente.

- **Verdadeiro positivo (*True Positive* - TP)**: Quando o valor previsto é de fato o valor real;
- **Verdadeiro negativo (*True Negative* - TN)**: Quando o valor previsto corresponde ao valor real, e valor real era uma amostra negativa;

- **Falso positivo (*False Positive* - FP)**: Quando o valor previsto é de uma amostra positiva, porém o valor real não corresponde à uma;
- **Falso negative (*False negative* - FN)**: Quando não é previsto um valor, porém o valor real é de uma amostra positiva.

Uma vez que tais conceitos foram explicados, é possível utilizá-los para realizar a avaliação de modelos de classificação. Dentre as diversas métricas utilizadas para a validação e avaliação de algoritmos de detecção, neste trabalho, serão utilizadas as métricas *Precisão, Recall* e *average Precision* (aP), para o YOLOv3.

- **Precisão**: Avalia a quantidade de verdadeiros positivos (TP) em relação à soma de todos os valores positivos. É calculado de acordo com a equação 2.2:

$$Precisão = \frac{TP}{TP + FP} \tag{2.2}$$

- **Recall**: Avalia a capacidade do algoritmo de detectar com sucesso resultados classificados como positivos. É calculado de acordo com a equação 2.3:

$$Recall = \frac{TP}{TP + FN} \tag{2.3}$$

- ***average Precision***: É a soma ponderada das precisões em cada limite, onde o peso é o aumento no *recall* (GAD, 2020), e é calculada de acordo com a equação 2.4. Foi escolhida para ser utilizada no YOLOv3 pelo fato de o modelo padrão da rede ter sido treinado com mais de 80 classes, e não apenas uma (pessoa) como neste trabalho.

$$aP = \sum_{k=0}^{k=n-1} [Recall(k) - Recall(k-1)] * Precisão(k), \tag{2.4}$$

onde n é o número de limites

Parte III

Metodologia

3 Metodologia

De acordo com Prodanov e Freitas (2013), o método científico é *"um conjunto de procedimentos adotados com o propósito de atingir o conhecimento"*, e também *"o conjunto de processos ou operações mentais que devemos empregar na investigação"*, sendo parte integrante do que constitui o conhecimento científico, que, de acordo com Gil (2008), possui como característica fundamental sua verificabilidade. Ainda segundo o autor, para que um conhecimento possa ser considerado científico, deve ser possível determinar o método que possibilitou chegar nesse conhecimento. Dessa forma, as próximas seções deste capítulo irão detalhar a metodologia utilizada neste processo. Em especial, a seção 3.1 iŕa classificar em detalhes a pesquisa, e a seção 3.2 irá explicar o processo metodológico realizado.

3.1 Classificação da pesquisa

A pesquisa científica é, segundo Prodanov e Freitas (2013), a realização de um estudo planejado, cuja finalidade é descobrir respostas a partir da aplicação do método científico. Nesse sentido, a classificação da pesquisa mostra-se um importante conceito da metodologia científica, visto que existem diversos tipos de pesquisa.

Segundo o autor, existem várias formas de classificar uma pesquisa, sendo as clássicas: quanto à natureza; quanto aos objetivos; quanto aos procedimentos e quanto à abordagem. Nesse sentido, esse capítulo busca explicar a metodologia a partir da pesquisa científica realizada, explicando seus principais pontos de acordo com os pontos de vista anteriormente citados.

3.1.1 Abordagem da pesquisa

Uma pesquisa pode, quanto à sua abordagem, ser classificada em **quantitativa** ou **qualitativa**. A pesquisa quantitativa considera aquilo que pode ser mensurado, traduzindo em números, opiniões, e informações para que se possa classificá-las e analisá-las (GIL, 2008). No desenvolvimento da pesquisa quantitativa, faz-se necessário formular hipóteses e classificar as relações entre as diversas variáveis da pesquisa, permitindo assim uma precisão nos resultados.

Já a pesquisa qualitativa não requer uso de métodos e técnicas estatísticas, como ocorre na quantitativa. Nela, o processo e seu significado são os principais focos da abordagem, tendo o ambiente como fonte direta dos dados, não havendo foco em comprovar hipóteses estabelecidas.

Dados os conceitos sobre as abordagens de uma pesquisa, neste trabalho optou-se por realizar uma abordagem híbrida, *quantitativo-qualitativa*. A abordagem quantitativa, neste trabalho, tem como objetivo coletar e mensurar as métricas discutidas na Seção 2.3, bem como a capacidade do algoritmo em detectar pedestres. A abordagem qualitativa se dará em cima da análise que será feita com o embasamento dos dados e resultados coletados.

3.1.2 Natureza da pesquisa

Do ponto de vista da natureza da pesquisa, ela pode ser classificada em **básica**, quando se pretende gerar conhecimentos novos sem uma aplicação prática prevista, e **aplicada**, quando se deseja gerar conhecimentos direcionados à solução de problemas específicos. Nesse sentido, a pesquisa realizada neste trabalho é uma pesquisa aplicada, visto que existem objetivos específicos que serão abordados.

3.1.3 Objetivo da pesquisa

Prodanov e Freitas (2013) classificam os objetivos da pesquisa em três, sendo eles: **pesquisa exploratória**, **pesquisa descritiva** e **pesquisa explicativa**. Segundo o autor, a pesquisa exploratória têm como objetivo proporcionar mais informações sobre o assunto a ser investigado, permitindo a sua definição e delineamento; a pesquisa descritiva busca descrever as características de determinada população ou fenômeno ou relações entre variáveis, e a pesquisa explicativa busca identificar fatores que determinam ou contribuem para a ocorrência dos fenômenos.

Para este trabalho, foi escolhida uma pesquisa do tipo exploratória, por se objetivar um maior aprofundamento no objeto de estudo, e também pelo fato de, assim como dito pelo autor, a pesquisa exploratória possuir um planejamento flexível, permitindo o estudo do tema sob diversos ângulos e aspectos.

3.1.4 Procedimentos

Segundo Gil (2008), os procedimentos técnicos são a maneira pela qual se obtêm os dados necessários para a elaboração de uma pesquisa, sendo o elemento mais importante para a identificação de um delineamento[1]. Para o autor, podem ser definidos dois grupos de delineamentos: os que se valem das chamadas *fontes de papel*, e aqueles cujos dados são fornecidos por pessoas. Nesse sentido, esta pesquisa utiliza dois procedimentos de pesquisa, sendo eles *pesquisa bibliográfica* e *cenário de uso*, respectivamente.

A pesquisa bibliográfica, segundo o autor, têm como objetivo colocar o pesquisador em contato direto com todo o material já escrito sobre o assunto de pesquisa. Dessa forma,

[1] Planejamento da pesquisa em sua dimensão mais ampla

permitiu que se pudesse alcançar um maior nível de conhecimento sobre os objetos de estudo deste trabalho.

Já para Rezende (2003), cenários são instâncias de caso de uso, ricas em detalhes contextuais, sendo uma *"narrativa, textual ou pictórica, de uma situação envolvendo usuários, processos e dados reais ou potenciais"*. Logo, a utilização do cenário de uso neste trabalho se dá para descrever o contexto em que a investigação proposta pela pesquisa é realizada, sendo de fundamental importância para a construção do trabalho.

Uma vez em que a classificação da pesquisa foi finalizada, será mais fácil realizar o desenvolvimento da mesma, pois os principais pilares de construção do processo de pesquisa estão definidos. A Tabela 3.1 apresenta, de forma resumida, todas as classificações de pesquisas definidas para a realização deste trabalho.

Tabela 3.1 – Metodologia da Pesquisa.

Abordagem da pesquisa	**Natureza da pesquisa**	**Objetivos da pesquisa**	**Procedimentos da pesquisa**
Qualitativa-quantitativa	Aplicada	Exploratória	Pesquisa bibliográfica / Cenário de uso

Fonte: Autor

3.2 Processo metodológico

Durante o desenvolvimento as atividades deste pesquisa, optou-se por utilizar o *Kanban*. O *Kanban* é um método visual para gerenciar e organizar o trabalho e, segundo Kniberg (2010), permite que o trabalho seja dividido em pequenos pedaços, facilitando a visualização do fluxo de trabalho como um todo, e permitindo o controle e limite de trabalho em progresso. Ainda segundo ele, o *Kanban* é comumente utilizado em projetos que envolvem metodologias ágeis. Nesse sentido, a utilização do *Kanban* permitiu a organização das tarefas a serem feitas neste projeto, possibilitando também a montagem do cronograma do projeto.

Assim, observa-se, na Tabela 3.2, o cronograma de atividades que foram realizadas no Trabalho de Conclusão de Curso 1 (TCC1). O cronograma pode ser dividido em seis macro atividades: *Definir o tema, Definir escopo da pesquisa, Pesquisar referencial teórico, Realizar a pesquisa, Documentar o processo de desenvolvimento da pesquisa e Apresentar TCC*.

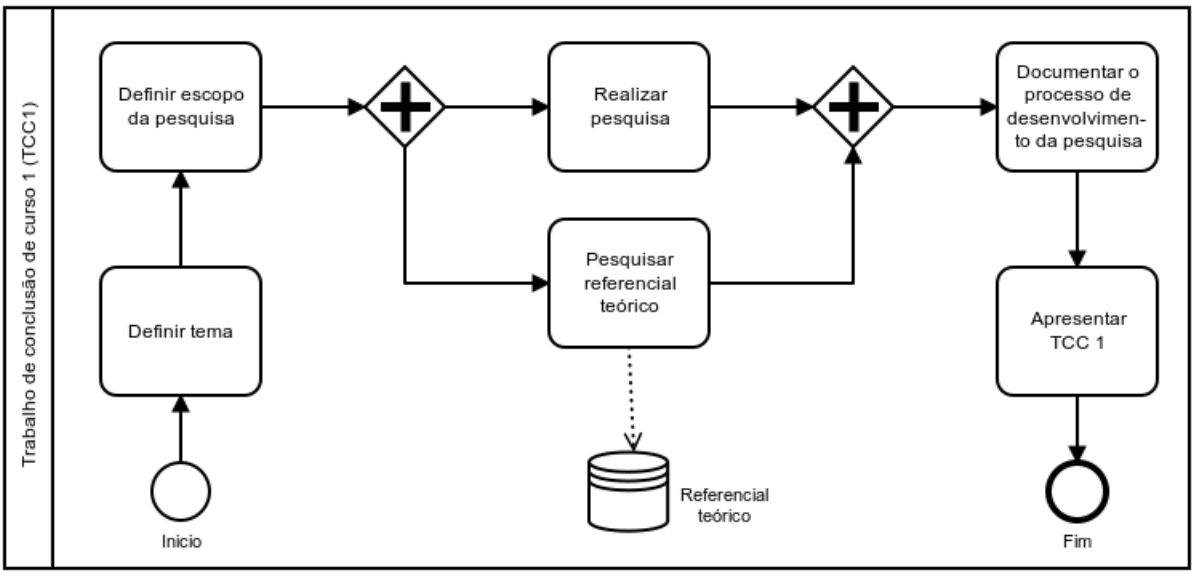
Tabela 3.2 – Cronograma de atividades do TCC 1

Cronograma Junho	**Julho**	**Agosto**	**Setembro**	**Outubro**	**Novembro**
Definir o tema	X				
Definir escopo da pesquisa	X				
Pesquisar referencial teórico	X	X			
Realizar a pesquisa		X	X	X	
Documentar o processo de desenvolvimento da pesquisa			X	X	X
Apresentar TCC 1					X

Fonte: Autor

Para facilitar a compreensão do processo metodológico, foi elaborado um diagrama que pode ser observado na Figura 3.1. Nessa imagem, são descritas as macro atividades contidas no cronograma de atividades, no formato de um diagrama BPMN[2].

Figura 3.1 – Processo Metodológico para o TCC1.



Fonte: Autor

- **Definir Tema**: Atividade que propõe a escolha do tema e contexto a ser abordado.

[2] Diagrama de notação de modelagem para processos de negócios, mas também utilizado em situações diversas

Nela, o orientador sugeriu alguns temas para que ao final fosse escolhido o de maior afinidade do pesquisador;

- **Definir escopo da pesquisa**: Uma vez que o tema é definido, é necessário definir o escopo da pesquisa, limitando seus objetivos, de forma a deixar clara a contextualização do tema;
- **Pesquisar referencial teórico**: Com o tema e escopos definidos, é necessário buscar referências na área escolhida, de forma a aumentar o conhecimento do pesquisador, além de contextualizá-lo de forma clara e objetiva. Nessa atividade, é realizado o procedimento de *pesquisa bibliográfica*, cujo processo pode ser encontrado na Seção 3.2.1;
- **Realizar a pesquisa**: A atividade de realizar a pesquisa engloba, de forma geral, o *cenário de uso*, onde são realizadas etapas que visam coletar, analisar e interpretar os dados que serão gerados por meio da pesquisa. Os detalhes dessa atividade podem ser vistos na Seção 3.2.2;
- **Documentar o processo de desenvolvimento da pesquisa**: Atividade em que será realizada a escrita dos dados e informações obtidas durante o processo de desenvolvimento da pesquisa, e
- **Apresentar TCC 1**: Apresentar o processo executado e os resultados obtidos para o professor orientador e a banca examinadora.

Tabela 3.3 – Cronograma de atividades para o TCC 2

Cronograma	**Dezembro**	**Janeiro**	**Fevereiro**	**Março**	**Abril**
Analisar o *dataset* INRIA, re-rotulando-o	X	X			
Capturar e rotular um novo *dataset* focado em pedestres	X	X			
Treinar e avaliar o novo *dataset* proposto			X	X	
Documentar o processo de pesquisa		X	X	X	X
Apresentar TCC 2					X

Fonte: Autor

Já a Tabela 3.3 mostra o cronograma de atividades que foram realizadas no Trabalho de Conclusão de Curso 2 (TCC2). Assim como o cronograma anterior, ele pode ser dividido em macro atividades: *Analisar o dataset INRIA, Capturar e rotular um novo dataset focado em pedestres, Treinar e avaliar o novo dataset proposto, Realizar a pesquisa, Documentar o processo da pesquisa e Apresentar TCC 2.*

- **Analisar o dataset INRIA** A partir dos problemas encontrados com o *dataset* INRIA nas atividades desenvolvidas no TCC1, será feita uma re-rotulagem do mesmo, comparando as anotações novas com as já existentes, e re-avaliando o *dataset* com a nova rotulagem;

- **Capturar e rotular um novo *dataset* focado em pedestres** Após treinar e avaliar os modelos utilizando os *datasets* INRIA e PSU, verificou-se que ambos não representavam o cenário que um veículo autônomo iria enfrentar. Logo, decidiu-se por elaborar um novo *dataset* feito a partir de imagens capturadas de uma câmera veicular dentro de um percurso estabelecido. Após a captura das imagens, também foi feita a rotulagem das mesmas;

- **Treinar e avaliar o novo *dataset* proposto** Após a rotulagem das imagens do novo *dataset*, foi feito o treinamento de um novo modelo, avaliando-o no novo *dataset*, e comparando-o com o modelo padrão do YOLOv3;

- **Documentar o processo de pesquisa**: Atividade em que foi realizada a escrita dos dados e informações obtidas durante o processo de desenvolvimento da pesquisa e geração do novo *dataset*, e

- **Apresentar TCC 2**: Apresentar o processo executado e resultados obtidos para o professor orientador e a banca examinadora.

3.2.1 Levantamento bibliográfico

Segundo Fonseca (2002), a pesquisa bibliográfica *"é feita a partir do levantamento de referenciais teóricos já analisados e publicados por meios escritos e eletrônicos como livros, artigos científicos e páginas de web sites"*. Tal modalidade de pesquisa foi utilizada na atividade *Realizar a pesquisa*, abordada anteriormente.

Nesse sentido, o procedimento deu-se da seguinte forma: para o YOLOv3, foram captados e lidos artigos encontrados no site oficial do YOLO[3]. A Tabela 3.4 mostra os artigos selecionados, cuja escolha se deu por focarem na rede neural, permitindo um maior conhecimento sobre ela.

[3] <https://pjreddie.com/publications/>

Tabela 3.4 – Artigos selecionados a partir do site oficial do YOLO

Título	Autor(es)	Data
YOLOv3: An Incremental Improvement	Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi	2018
You Only Look Once: Unified, Real-Time Object Detection	Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi	2016
YOLO9000: Better, Faster, Stronger	Joseph Redmon, Ali Farhadi	2016

Fonte: Autor

Também foi realizada uma pesquisa utilizando-se o Periódico Capes[4] e a base de dados IEEE Xplore[5]. Realizou-se uma busca utilizando-se as seguintes palavras chaves: "Yolo and pedestrian detection", cujos artigos selecionados podem ser vistos conforme a Tabela 3.5. Vale salientar que foram escolhidos os artigos, cujo foco se dava não apenas no algoritmo de detecção, mas também na detecção de pedestres em si.

Ao final do processo de pesquisa bibliográfica, o pesquisador pôde compreender melhor conceitos sobre o tema abordado. Logo, seu objetivo primordial foi alcançado, visto que houve um maior conhecimento e entendimento sobre o assunto da pesquisa.

3.2.2 Realizar pesquisa

A partir do cenário de us,o a atividade *Realizar pesquisa* é construída, sendo possível, nela, coletar, analisar e interpretar os dados. A Figura 3.2 mostra o diagrama BPMN para a atividade, cujos processos serão detalhados a seguir.

- **Prova de conceito YOLOv3**: Atividade feita para testar se o algoritmo YOLOv3 funcionará bem para a proposta realizada, além de instruir o pesquisador quanto sua utilização;

- **Elaboração do *dataset***: Descreve o processo de elaboração, captura e organização do *dataset* proposto, a partir da utilização de uma câmera veicular dentro de um percurso pré-estabelecido;

[4] <https://www-periodicos-capes-gov-br.ez54.periodicos.capes.gov.br/>
[5] <https://ieeexplore.ieee.org/Xplore/home.jsp>

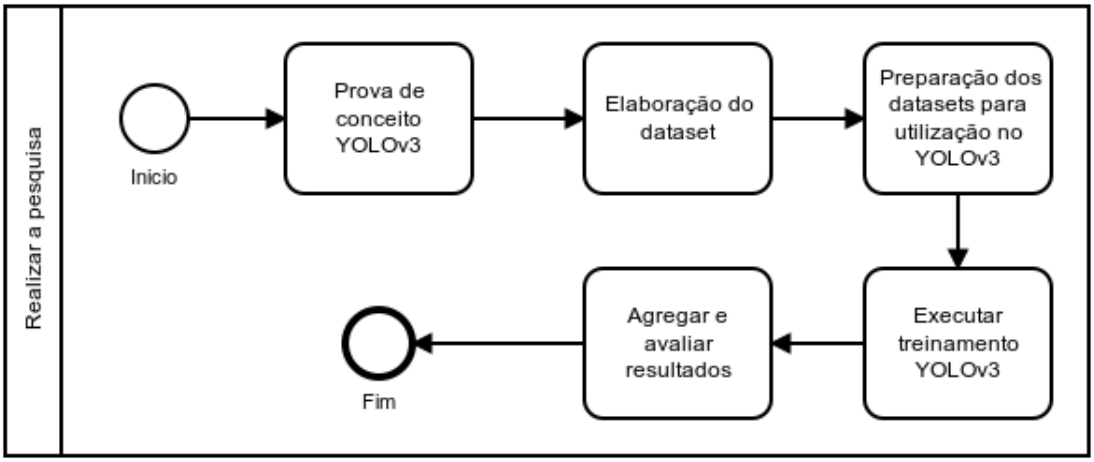
Tabela 3.5 – Artigos selecionados após a busca na base IEEE Xplore

Título	**Autor(es)**	**Data**
Real-Time Pedestrian Detection Based on Improved YOLO Model	Congcong Zhao e Bin Chen	2019
Pedestrian Detection Based on YOLO Network Model	Wenbo Lan e Jianwu Dang e Yangping Wang e Song Wang	2018
Improving the Accuracy of Pedestrian Detection in Partially Occluded or Obstructed Scenarios	Redge Melroy Castelino e Gabriel Passos Moreira Pinheiro e Bruno Justino Garcia Praciano e Giovanni Almeida Santos e Lothar Weichenberger e Rafael Timóteo De Sousa Júnior	2020
Real-time pedestrian detection using SVM and AdaBoost	Rupesh A. Kharjul e Vinit K. Tungar e Yogesh P. Kulkarni e Samarth K. Upadhyay e Rakesh Shirsath	2015
Pedestrian Detection Based on Motion Compensation and HOG/SVM Classifier	Fen Xu; Feng Xu	2013

Fonte: Autor

- **Preparação dos *datasets* para utilização no YOLOv3**: Uma vez que o pesquisador teve um primeiro contato com o YOLOv3, o próximo passo consiste em preparar os *datasets* para que possam ser treinados e avaliados nele. Nesse passo, é feita a rotulagem do *dataset* PSU, além de preparar o *dataset* INRIA Person para trabalhar com os formatos de anotação esperados pelo YOLOv3;
- **Executar treinamento YOLOv3**: Com os *datasets* preparados e configurados, a atividade atual consiste em de fato executar o treinamento utilizando o YOLOv3. Nessa atividade, são documentados todos os passos necessários para realizar o treinamento, de forma que qualquer pessoa possa reproduzi-lo, e
- **Agregar e avaliar resultados**: Ao final, todos os dados e resultados obtidos após

Figura 3.2 – Diagrama representando a atividade Realizar Pesquisa.



Fonte: Autor

a execução dos treinamentos serão agregados e avaliados de acordo com as métricas definidas na seção 2.3, e também serão discutidos os resultados individuais em cada *dataset*, procurando fazer uma análise focando nas caraterísticas dos mesmos.

O próximo capítulo irá detalhar o desenvolvimento do projeto, seguindo o procedimento de cenário de uso. Ao final, será possível observar os resultados obtidos na pesquisa.

Parte IV

Resultados

4 Resultados Obtidos

4.1 Prova de conceito YOLO

Segundo Nogueira e Guimarães (2008), prova de conceito (POC) define-se como "uma técnica que permite demonstrar que uma determinada ideia é tecnicamente possível, ou seja, pode ajudar a verificar se uma arquitetura é construível". Nesse sentido, construir uma prova de conceito utilizando o YOLOv3 tem por objetivo permitir que o pesquisador possa entender o funcionamento do processo de treinamento e avaliação, permitindo que ele possa avaliar se o algoritmo é adequado para ser utilizado no contexto de detecção de pedestres, visto que o YOLO é um detector de objetos.

Para a realização da prova de conceito, foi utilizado um notebook da marca *Dell* modelo *Gaming* 7567, com as seguintes especificações: processador Intel core I7-7700HQ, 32GB de memória RAM, placa de vídeo dedicada NVIDIA GeForce GTX 1050Ti *Mobile* e 1TB de SSD para armazenamento. O sistema operacional Linux foi utilizado, mais especificamente a distro[1] Manjaro KDE. A POC foi realizada a partir dos seguintes passos:

1. Primeiramente, é preciso baixar o código fonte e compilar o *darknet*[2]. Para isso, os seguintes comandos devem ser seguidos num terminal (console):

```
$ git clone https://github.com/pjreddie/darknet
$ cd darknet
$ make
```

2. Depois, é necessário baixar o modelo pré-treinado do YOLOv3.

```
$ wget https://pjreddie.com/media/files/yolov3.weights
```

3. E então, deve-se realizar a detecção:

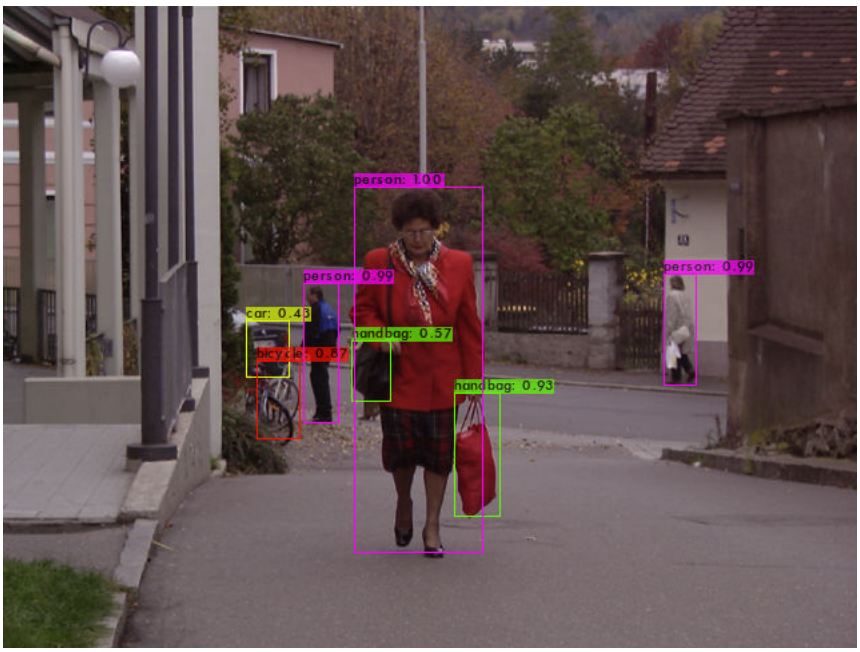
```
$ ./darknet detect cfg/yolov3.cfg yolov3.weights <caminho>
```

   Onde o caminho deve indicar o caminho até a imagem e nome da mesma.

Ao final, a *darknet* irá exibir no terminal os objetos detectados, a confidência, e quanto tempo foi necessários para encontrá-los. Ele também irá salvar os objetos encontrados numa nova imagem, chamada `predictions.png`.

[1] Sistema operacional criado a partir de uma coleção de softwares
[2] Framework de redes neurais escritos em C. É utilizado pelo YOLOv3

Figura 4.1 – Imagem do *dataset* PSU com as predições feitas pelo YOLOv3.



Fonte: Adaptado de Thu, Suvonvorn e Karnjanadecha (2018)

Conforme visto na Figura 4.1, o algoritmo YOLOv3 consegue detectar diversas classes de objetos na imagem. Logo, é possível e viável continuar com o experimento, visto que a prova de conceito mostrou de fato ser possível detectar objetos com detector.

4.2 Elaboração do dataset

A captura de um *dataset* próprio teve como objetivo elaborar um cenário mais próximo do enfrentado por um veículo autônomo, dado que os *datasets* INRIA e PSU, apesar de possuírem foco em pessoas, não refletem o mesmo cenário que seria capturado por um veículo autônomo, e apresentarem os problemas descritos na seção 4.5. Nesse sentido, esse capítulo irá mostrar o trajeto realizado, a câmera utilizada e o processo de organização e separação das imagens.

4.2.1 Trajeto realizado

Foram escolhidos dois cenários para a captura das imagens: um diurno, e outro noturno. O cenário noturno teve como objetivo verificar o desempenho do algoritmo em imagens com pouca iluminação, além de disponibilizar um cenário não presente nos *datasets* PSU e INRIA. O trajeto escolhido incluiu as seguintes regiões administrativas do Distrito Federal: Taguatinga, Samambaia e Ceilândia, focando trechos de grande movimento de pessoas. A Figura 4.2 mostra o trajeto capturado durante o cenário diurno, que

se inicia em Taguatinga norte, indo para Samambaia, Ceilândia, e finalizando novamente em Taguatinga, totalizando 21.8km, feito em aproximadamente 58 minutos.

Figura 4.2 – Trajeto diurno da captura do *dataset* proposto

Fonte: Autor

Já a Figura 4.3 mostra o trajeto capturado durante o cenário noturno. Apesar de passar pelas mesmas regiões administrativas do cenário noturno, o percurso não é o mesmo; pelo fato de haverem menos pessoas pelo horário, o trajeto foi desviado para passar pelo Shopping JK - Ceilândia, no intuito de visualizar o máximo de pessoas possível. O trajeto totalizou 30.6km, numa viajem de aproximadamente 1 hora e 21 minutos.

Figura 4.3 – Trajeto noturno da captura do *dataset* proposto

Fonte: Autor

4.2.2 Escolha da câmera

Para a escolha da câmera, foram escolhidos três critérios principais: Resolução da gravação, armazenamento e preço. Dados tais critérios de escolha, foi feita uma busca no sítio Mercado livre, e 5 câmeras foram selecionadas, sendo descritas de acordo com a Tabela 4.1.

Dados os critérios acima, com foco no preço, a câmera escolhida foi a **Black Box Gp2**. A câmera, cujo modelo pode ser visto na Fig. 4.4, grava aproximadamente 3 horas e meia de vídeo, gravando trechos de 3 minutos com o nome *MOVIXXXX.avi*, onde as letras *X* representam um número que varia de acordo com a gravação[3]. Caso as gravações excedam o tamanho do armazenamento, o arquivo mais antigo é excluído para liberar espaço, tornando a gravação cíclica.

[3] Por exemplo, o primeiro vídeo gravado possui o nome MOVI0001.avi

Tabela 4.1 – Número de imagens por *dataset*

Modelo	Resolução	Armazenamento	Preço
Black Box Gp2	1280 x 720p (30fps)	Até 32gb	R$ 295,65
Black Box Gp Dual	Dianteira: 1280 x 720p (30fps), Traseira: 858 x 480p (30fps)	Até 32gb	R$ 535,50
Black Box Gp5	1920 x 1080p (30fps)	Até 128gb	R$ 496,86
Xiaomi 70mai	1920 x 1080p (30fps)	Até 128gb	R$ 448,85
Intelbras Dc 3101	1920 x 1080p (30fps)	Até 64gb	R$ 499,90

Fonte: Autor

Figura 4.4 – Câmera modelo Black Box Gp2.

Fonte: BLACKBOX.CAMERAS.VEICULARES (2022)

Após a captura dos vídeos, foi necessário extrair as imagens dos mesmos. Como a câmera grava à 30 *frames* por segundo, optou-se por utilizar apenas 2 *frames* por segundo. Tal escolha permitiu manter os acontecimentos do vídeo sem muitas perdas, ao mesmo tempo que não gerou uma quantidade alta de imagens. A Tabela 4.2 mostra a quantidade de imagens extraídas, nos cenários diurno e noturno além do tempo total de gravação.

Tabela 4.2 – Número de imagens extraídas por cenário

Cenário	Imagens	Duração da gravação
Diurno	6.324	57 min 12 seg
Noturno	7.784	81 min 28 seg

Fonte: Autor

O trecho de código a seguir mostra a função utilizada para a extração dos *frames* dos vídeos. Nele, é utilizado a biblioteca *python* OpenCV[4], uma biblioteca muito comun

[4] https://opencv.org/

para o desenvolvimento de aplicativos na área de Visão computacional. A função recebe o caminho de um arquivo, carrega-o em memória, e itera por cada frame. Como 15 *frames* são ignorados por vez, apenas 2 são salvos por segundo, seguindo o seguinte padrão de nomenclatura: `MOVINNNN_frame_MMMM.jpg`, onde `NNNN` representa um número que varia de acordo com a gravação, e `MMM` é um número indicando o o *frame* atual.

```
# (...)
FRAME_SKIP = 15
# (...)

def get_frames(file: str):
    cap = cv2.VideoCapture(file)
    i = 0
    _, mode, filename = file.split("/")
    frame_count = 0
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        if i > FRAME_SKIP - 1:
            frame_count += 1
            cv2.imwrite(
                './data/'+ mode + "/" + \
                filename.replace(".avi", "_") + \
                'frame_'+str(frame_count*FRAME_SKIP)+'.jpg',
                frame
            )
            i = 0
            continue
        i += 1

    cap.release()
    cv2.destroyAllWindows()

# (...)
```

4.2.3 Categorização das imagens

Após a geração das imagens a partir dos arquivos de vídeo, o próximo passo consiste em separá-las em amostras positivas (com pedestres) e negativas (sem pedestres).

Conforme visto na Tabela 4.2, é necessário classificar cerca de 14.108 imagens, o que torna a classificação manual uma tarefa complexa e demorada. Nesse sentido, foi criado um *script* que pré-classificou as imagens em positivas e negativas, de forma a facilitar a separação manual. O *script* utiliza a biblioteca *ImageAI*, que é um projeto de código aberto focado em facilitar desenvolvedores a construírem aplicações que utilizam Visão Computacional e *Deep Learning* com poucas linhas de código(MOSES; OLAFENWA, 2018–).

O trecho a seguir mostra o *script*: nele, é utilizado um modelo do YOLOv3 para realizar a classificação, limitando a detecção apenas para pessoas. Todos os arquivos foram carregados de um diretório e, caso não tivessem sido classificados, eram separados em uma amostra positiva ou negativa, caso uma caixa delimitadora com a classe *person* (pessoa) fosse encontrada. Após a classificação, o arquivo original é copiado para outro diretório, de acordo com a sua classificação.

```
from math import ceil
import multiprocessing
from imageai.Detection import ObjectDetection
import os
import shutil
import glob

MODE = "noite"
execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel(detection_speed='fast')

files_list = glob.glob(f"./data/{MODE}/**.jpg")

def convert_to_path(files):
    for index, file in enumerate(files):
        print(f"Arquivo atual: {file} ({index}/{len(files)})")
        if os.path.isfile(
            f"./parsed_data/{MODE}/pos/{file.split('/')[-1]}"
            ) or \
            os.path.isfile(
                f"./parsed_data/{MODE}/neg/{file.split('/')[-1]}"
```

```
            ):
            pass
        else:
            detections = detector.detectObjectsFromImage(
                input_image=os.path.join(execution_path , file),
                output_type='array',
                custom_objects=detector.CustomObjects(person=True),
            )
            if any(obj['name'] == 'person' for obj in detections[1]):
                shutil.copyfile(
                    file,
                    f"./parsed_data/{MODE}/pos/{file.split('/')[-1]}"
                )
            else:
                shutil.copyfile(
                    file,
                    f"./parsed_data/{MODE}/neg/{file.split('/')[-1]}"
                )

if __name__ == "__main__":
    convert_to_path(files_list)
```

O tempo de execução médio por imagem foi de 10 segundos, resultando em aproximadamente 39 horas necessárias para que todas as imagens fossem pré-classificadas pelo YOLOv3. A Tabela 4.3 mostra a divisão por cenário: durante o dia, cerca de 23% das amostras foram positivas, contra aproximadamente 7% no cenário noturno. Ao ser feita uma primeira verificação manual, constatou-se a presença de amostras contendo falso-negativos, onde, apesar de haverem pedestres, o algoritmo não os detectou. Além disso, outro fato que pode explicar a diferença de valores é o fato de o algoritmo YOLOv3 ter uma maior dificuldade em detectar pessoas em cenários com pouca ou baixa iluminação, e o fato de o atributo *detection_speed='fast'* ter sido utilizado[5].

Nesse sentido, o próximo passo consistiu em classificar manualmente as amostras separadas pelo *script*. Cada imagem foi aberta, uma a uma, sendo movida para o diretório correto caso a classificação tenha sido incorreta. A Tabela 4.4 indica os novos valores de amostras, por cenário. Nela, verifica-se um aumento na quantidade de amostras positivas na ordem de 11.27% no cenário diurno, e 2.18% no cenário diurno, o que indica que de fato haviam amostras com falsos-negativos. Vale salientar que, apesar de a diferença no

[5] Indica a velocidade de detecção do algoritmo. O valor *fast* pode deixar a detecção mais rápida, ao custo de uma menor acurácia

Tabela 4.3 – Imagens classificadas pelo algoritmo YoloV3

Cenário	Amostras Positivas	Amostras Negativas
Diurno	1.461 (23.10%)	4.863 (76.90%)
Noturno	541 (6.95%)	7.243 (95.05%)

Fonte: Autor

cenário noturno ser menor que a do cenário noturno, seu impacto é, de fato, maior: durante a captura noturna haviam menos pedestres nas ruas, fazendo com que a quantidade de amostras positivas fossem significativamente menor que no cenário diurno.

Tabela 4.4 – Imagens reclassificadas após a triagem do Yolov3

Cenário	Amostras Positivas	Amostras Negativas	Diferença
Diurno	2.167 (34.27%)	4.157 (65,73%)	+11.27% (positivas)
Noturno	711 (9.13%)	7.073 (90.87%)	+2.18% (positivas)

Fonte: Autor

Após a classificação, o *dataset* está pronto para ser anotado e utilizado para gerar um novo modelo. A próxima seção, 4.3, irá detalhar o processo de preparação do *dataset*, em conjunto com os demais utilizados nesse projeto, e a seção 4.4 demonstrará o processo de treinamento com o YOLOv3.

4.3 Preparação dos datasets para utilização no YOLOv3

Antes de iniciar os treinamentos utilizando-se o YOLOv3, é necessário garantir que os *datasets* possuam não apenas as amostras, mas também as anotações correspondentes. Segundo Bochkovskiy, Wang e Liao (2020), o YOLOv3 espera um arquivo .txt para cada arquivo de imagem, e o arquivo deve conter, em cada linha, os dados da(s) caixa(s) delimitadora(s)[6], com o seguinte formato:

```
<classe do objeto> <x> <y> <largura> <altura>
```

Onde a **classe do objeto** é um número inteiro que representa o número da classe do objeto; **x** e **y** representam as coordenadas da caixa delimitadora nos eixos x e y, respectivamente, e **largura** e **altura** representam a altura e largura da caixa delimitadora, respectivamente. Cada arquivo `.txt` pode ter nenhuma (amostra negativa), uma ou várias linhas (uma ou mais pessoas marcadas). No caso do projeto realizado neste trabalho, todas

[6] <https://github.com/AlexeyAB/Yolo_mark/issues/60>

as classes do objeto sempre serão 0, visto que se deseja detectar apenas a classe *person*. A Fig. 4.5 mostra um arquivo de exemplo com três anotações, todas no formato descrito anteriormente.

Figura 4.5 – Exemplo de anotações no formato esperado pelo YOLOv3.

```
0 0.67578125 0.4270833333333333 0.0609375 0.26666666666666666
0 0.871875 0.45 0.05625 0.2791666666666667
0 0.778125 0.46458333333333335 0.05 0.24166666666666667
```

Fonte: Autor

4.3.1 INRIA Person

O *dataset* INRIA Person possui anotações de caixas delimitadoras, porém elas estão no formato *Pascal Challenge* (DALAL, 2005). Tal formato não é compatível com o YOLOv3, sendo necessário convertê-los. Para isso, é utilizado um *script* escrito em *python*, disponibilizado por Xu (2020), conforme visto a seguir:

```
# (...)
def solve(mode):
  # /pos
  annotations_path = './INRIAPerson/{}/annotations/'.format(mode)
  save_path = "./data/{}/".format(mode)
  if not os.path.exists(save_path):
     os.makedirs(save_path)
  annotation_files = os.listdir(annotations_path)
  for file in annotation_files:
     with open(annotations_path+file, 'rb') as f:
           ann = f.read()
     img_size = re.findall('\d+ x \d+ x \d+', str(ann))
     img_w, img_h = re.findall('\d+', img_size[0])[0:2]
     img_w = int(img_w)
     img_h = int(img_h)
     Ground_Truth_list = re.findall(
        '\(\d+, \d+\)[\s\-]+\(\d+, \d+\)',
        str(ann)
     )
     coordinates = [re.findall('\d+', box) for box in Ground_Truth_list]
     coordinates = np.array(coordinates, dtype='int')
     with open(save_path + file, 'w') as w:
           for pos in coordinates:
```

```
                x = (pos[0] + pos[2]) / 2.0
                y = (pos[1] + pos[3]) / 2.0
                width = pos[2] - pos[0]
                heigth = pos[3] - pos[1]
                newline = '0 '+ str(1.0*x/img_w)+' '+ \
                str(1.0*y/img_h) + ' '+str(1.0*width/img_w) +\
                ' '+str(1.0*heigth/img_h)
                w.write(newline + '\n')
    # /neg
    neg_img_path = './INRIAPerson/{}/neg/'.format(mode)
    for file in os.listdir(neg_img_path):
        fp = open(save_path+file.split('.')[0] + '.txt', 'a')
        fp.close()

if __name__ == '__main__':
    solve('Train')
    solve('Test')
```

A função `solve` recebe como parâmetro o modo, que pode ser `Train` ou `Test`. Depois, para cada arquivo de anotação, ele encontra as coordenadas no formato *Pascal Challenge*; converte-as para o formato YOLOv3, e salva-as no diretório `./data/`. Após esse processo, o *dataset* está pronto para ser utilizado.

4.3.2 PSU

Diferentemente do INRIA, o *dataset* PSU não possui nenhum tipo de anotação de caixas delimitadoras ou mesmo separação em conjunto de treino e teste. Logo, o primeiro passo é realizar, de forma manual, tais anotações. Existem diversos produtos de software que permitem a anotação manual de imagens, sendo escolhido para este trabalho o **labelImg**[7]. Ele é um software escrito na linguagem *python*, que utiliza o Qt[8] como interface gráfica (TZUTALIN, 2015).

Para gerar as anotações, é necessário antes realizar a instalação do **labelImg**. Existem duas formas de instalá-lo: a primeira é clonando o repositório e construindo o software a partir do código fonte, e a segunda é instalar o código pré-compilado utilizando o *Python Package Index*[9], um repositório de softwares escritos em *python*. Neste trabalho,

[7] https://github.com/tzutalin/labelImg
[8] Software multiplataforma para criação de interfaces. Disponível em: https://www.qt.io/
[9] https://pypi.org/

optou-se pela segunda opção, pelo fato de o único requerimento ser a versão do *python* igual ou superior à 3.0[10].

A instalação do *software* foi realizada conforme os passos abaixo:

1. Como o sistema utilizado é o Manjaro Linux, é necessário instalar o pip. Para isso, é utilizado o pacman, um gerenciador de pacotes do Arch Linux[11]

   ```
   $ sudo pacman -S python-pip
   ```

2. O Segundo passo consiste em instalar o labelImg utilizando o pip:

   ```
   $ pip install labelImg
   ```

Com isso o labelImg estará instalado e pronto para ser utilizado. O uso do *software* é relativamente simples, sendo necessário apenas executar o seguinte comando num terminal:

```
$ labelImg [CAMINHO_DO_DIRETORIO_DE_IMAGENS] [ARQUIVO_DE_CLASSES]
```

Onde `CAMINHO_DA_PASTA_DE_IMAGENS` indica o caminho do diretório com as imagens a serem anotadas, e `ARQUIVO_DE_CLASSES` indica o diretório para o arquivo contendo a definição de classes a serem anotadas.

Após o comando ser executado, será aberta a interface gráfica do *software*. Conforme observado na Figura 4.6, o labelImg possui algumas áreas onde são agrupadas informações e/ou ferramentas:

- **A**: Essa seção contém ferramentas relacionadas ao carregamento de imagens e diretórios;
- **B**: Essa seção possui as ferramentas necessárias para avançar ou voltar à imagem anterior, verificar a imagem atual ou salvá-la;
- **C**: Botão que define o formato de saída da rotulagem, de acordo com o algoritmo selecionado. Vale salientar que, como é utilizado o algoritmo YOLOv3, é necessário alterar para a opção correspondente, assim como na Figura 4.6;
- **D**: Contém as ferramentas relacionadas à adição, remoção e duplicação de caixas delimitadoras;
- **E**: Contém as ferramentas utilizadas para dar zoom e ajustar o aspecto da imagem;

[10] É possível verificar a versão com o comando "python –version"num terminal
[11] O Manjaro Linux é uma distro que possui o Arch linux como base.

- **F**: Areá onde a imagem atual é exibida, e são realizadas as marcações de caixas delimitadoras utilizando-se o mouse. Na Figura 4.6, é possível observar um pedestre com sua correspondente anotação destacada;
- **G**: Seção referente às configurações de rótulos (*labels*). É possível definir a utilização de um rótulo padrão, selecionar o rótulo dentre os disponíveis no arquivo `ARQUIVO_DE_CLASSES`, e visualizar as rotulagens já realizadas, e
- **H**: Seção que exibe a lista de imagens com base no caminho definido em `CAMINHO_DA_PASTA_DE_IMAGENS`.

Figura 4.6 – Interface do programa labelImg.

Fonte: Autor

Uma vez que as ferramentas e áreas da interface foram brevemente explicadas, a utilização do *software* é bastante simples: basta marcar as caixas delimitadoras correspondentes com o mouse; salvar as rotulagens e seguir para a próxima imagem. O labelImg irá salvar tais rotulagens em um arquivo com a extensão `.txt` no diretório de imagens definido em `CAMINHO_DA_PASTA_DE_IMAGENS`, com o mesmo nome da imagem sendo anotada. Não é necessário realizar qualquer tipo de anotação nas amostras negativas[12].

Para o PSU, as anotações foram realizadas de forma rigorosa, onde qualquer pessoa ou silhueta que pudesse ser reconhecida em amostras positivas foi rotulada, inclusive de pessoas obstruídas.

[12] Porém o YOLOv3 ainda espera um arquivo .txt para amostras negativas.

4.3.3 Gerar arquivo obj.data

O arquivo `obj.data` é utilizado pelo YOLOv3 para indicar o número de classes, o arquivo com o caminho das amostras de treino teste, o arquivo de classes e a pasta onde devem ser salvos os pesos durante o *backup*, respectivamente, conforme pode-se ser visto na Figura 4.7. Esse arquivo é utilizado pelo YOLOv3 para realizar o treinamento e validação dos resultados, sendo necessário gerá-lo para ambos os *datasets*.

Figura 4.7 – Conteúdo do arquivo obj.data.

```
classes= 1
train  = data/Train.txt
valid  = data/Test.txt
names  = data/obj.names
backup = backup
```

Fonte: Autor

Contudo, para que o `obj.data` possa ser gerado, também é necessário que existam outros dois arquivos: `Train.txt` e `Test.txt`. Assim como dito anteriormente, tais arquivos indicam os caminhos das amostras de treino e teste, respectivamente, contendo uma imagem por linha.

A geração dos arquivos `Train.txt` e `Test.txt` para o INRIA Person foi mais simples, visto que o *dataset* já possui a divisão das amostras em treino e teste, sendo necessário apenas gerar os arquivos. Para isso, o seguinte *script python*, disponibilizado por Xu (2020), foi utilizado:

```
def solve(mode):
    img_path = os.getcwd()+"/data/{}/".format(mode)
    save_path = "./data/"
    save_file = '{}.txt'.format(mode)
    with open(save_path + save_file, 'w') as w:
        for file in os.listdir(img_path):
            if file == 'labels':
                continue
            w.write(img_path + file + '\n')


if __name__ == '__main__':
    solve('Train')
    solve('Test')
```

O *script* trabalha a partir da utilização da função `solve`, que espera como argu-

mento duas *strings*: o modo, se treino (*'Train'*) ou teste (*'Test'*), e o tipo da amostra, se positiva ('pos') ou negativa ('neg'). Ao ser executada, a função gera duas variáveis: `img_path` com o caminho relativo das imagens, e `save_path`, com o destino em que serão salvos os arquivos de saída. Na sequência, para cada amostra no `img_path`, uma linha com o caminho é inserida no arquivo Train/Test.txt, populando ambos os arquivos ao final da execução.

O processo é um pouco diferente para o *dataset* PSU, visto que além de não possuir anotações, ele também não possui a divisão nos conjuntos de teste e treino. Logo, um *script*, também em *python*, foi criado para dividir o *dataset* nos conjuntos de treino e teste, sendo feita a divisão com 80% e 20% das amostras, respectivamente, e gerar os arquivos `Train.txt` e `Test.txt`. O *script* segue os seguintes passos:

1. O primeiro passo consiste em definir as variáveis com o caminho das amostras positivas e negativas, além da divisão em treino e teste. Na sequência, são criadas duas listas contendo o caminho das amostras positivas e negativas. Para as amostras negativas, caso o arquivo `.txt` com o mesmo nome da imagem não exista, ele é criado.

```
POSITIVE_PATH, NEGATIVE_PATH = "./positive", "./negative"
TRAIN_SPLIT, TEST_SPLIT  = 0.8,  0.2

negative_images = [f for f in listdir(NEGATIVE_PATH) \
    if isfile(join(NEGATIVE_PATH, f)) and f.endswith('.jpg')]
positive_images = [f for f in listdir(POSITIVE_PATH) \
    if isfile(join(POSITIVE_PATH, f)) and f.endswith('.jpg')]

for index, image in enumerate(negative_images):
    if not path.isfile(f"./negative/{image.split('.')[0]}.txt"):
        filename = f"./negative/{image.split('.')[0]}.txt"
        with open(filename, "w") as f:
            f.write("")
```

2. O Segundo passo consiste em aleatorizar a lista com o caminho das imagens, para permitir que a divisão em treino e teste, a ser feita posteriormente, tenha o mínimo de viés possível:

```
shuffled_negative_images = [*negative_images]
shuffled_positive_images = [*positive_images]

shuffle(shuffled_negative_images)
shuffle(shuffled_positive_images)
```

3. O terceiro passo consiste em definir os conjuntos de treino e teste, com base na divisão feita nas variáveis `TRAIN_SPLIT` e `TEST_SPLIT`:

```
train = shuffled_negative_images[0: \
    int(len(shuffled_negative_images) * TRAIN_SPLIT)] + \
    shuffled_positive_images[0: \
        int(len(shuffled_positive_images) * TRAIN_SPLIT)]

test = \
    shuffled_negative_images[int(len(shuffled_negative_images) \
        * TRAIN_SPLIT):len(shuffled_negative_images)] + \
    shuffled_positive_images[int(len(shuffled_positive_images) \
        * TRAIN_SPLIT): len(shuffled_positive_images)]
```

4. O quarto passo consiste em copiar as imagens e *labels* para um novo diretório, chamado `./yolo_read`, onde `[CONJUNTO]` indica as variáveis `train` ou `test`; `[TIPO]` indica o tipo de conjunto, podendo ser "Train"ou "Test", e `[AMOSTRA]` indica o tipo da amostra, podendo ser "negative"ou "positive":

```
for file in [CONJUNTO]:
    filename = file.split(".")[0]
    copyfile(
        f"./[AMOSTRA]/{filename}.txt",
        f"./yolo_ready/[TIPO]/{filename}.txt")
    copyfile(
        f"./[AMOSTRA]/{filename}.jpg",
        f"./yolo_ready/[TIPO]/{filename}.jpg")
```

5. Por fim, as variáveis `train` e `test` são reatribuídas, sendo adicionados os trechos `data/Train/` e `data/Test/`, respectivamente, para cada imagem de cada conjunto. A adição do texto `data/` se dá pelo fato de que as imagens e *labels* ficarem dentro da pasta de mesmo nome durante o treinamento. O *script* termina com a criação dos arquivos `Train.txt` e `Test.txt`:

```
train = [ f"data/Train/{img}" for img in train]
test = [ f"data/Test/{img}" for img in test]

with open("./yolo_ready/Train.txt", "w") as f:
    f.write("\n".join(train))

with open("./yolo_ready/Test.txt", "w") as f:
    f.write("\n".join(test))
```

O último passo necessário para que o arquivo possa ser corretamente utilizado consiste em gerar o arquivo `obj.names`. Como neste projeto o interesse é apenas o reconhecimento de pedestres, o arquivo é criado contendo apenas uma linha com a seguinte palavra: `person`, indicando a classe "pessoa".

4.3.4 Dataset proposto

Após a geração e a classificação das imagens do *dataset* proposto, conforme descrito na seção 4.2, é necessário anotar as amostras. Para isso, também é utilizado o *software* **labelImg**, assim como no PSU (seção 4.3.2).

Como o *dataset* possui uma quantidade de imagens negativas muito superior às imagens positivas, foi escolhido um percentual de 15% de amostras para serem negativas, sendo criada uma função que escolhe aleatoriamente as imagens negativas e salva-as num diretório preparado para o *dataset*, conforme o código a seguir:

```
def prepare_neg_images(mode):
    if mode not in ["Train", "Test"]:
        raise ValueError(f"Valor do modo inválido." + \
        f"Deve ser 'Train' ou 'Test', {mode} recebido.")

    MAX_DAY_IMAGES = int(
        len(glob.glob("./parsed_data/dia/pos/**.jpg")) * 0.15
    )
    MAX_NIGHT_IMAGES = int(
        len(glob.glob("./parsed_data/noite/pos/**.jpg")) * 0.15
        )



    day_neg_images = sample(
        glob.glob("./parsed_data/dia/neg/**.jpg"), MAX_DAY_IMAGES
    )
    night_neg_images = sample(
        glob.glob("./parsed_data/noite/neg/**.jpg"), MAX_NIGHT_IMAGES
    )

    for image in [*day_neg_images[0:int(MAX_DAY_IMAGES * 0.8)], \
        *night_neg_images[0:int(MAX_NIGHT_IMAGES * 0.8)]]:
        file_name = image.split("/")[-1]
        shutil.copyfile(image, f"./final_data/{mode}/{file_name}")
```

Após a escolha aleatória das amostras negativas, as amostras negativas também são copiadas para o diretório preparado para o *dataset*, com suas respectivas anotações. Na sequência, são criados os arquivos *Train.txt* e *Test.txt*, e os arquivos de anotações das amostras negativas[13], finalizando a geração dos arquivos necessários para a execução do treinamento no YOLOv3. O *dataset* está disponibilizado no Apêndice A, e as anotações dos *datasets* INRIA e PSU estão no apêndice B.

A Tabela 4.5 exibe a quantidade de imagens por *dataset*, considerando a divisão entre treino e teste realizada no PSU. Também é mostrado um terceiro *dataset* chamado "INRIA & PSU", que consiste da junção de ambos os *datasets*.

Tabela 4.5 – Número de imagens por *dataset*

Dataset	**Treino**	**Teste**	**Total**
INRIA	1832	741	2572
PSU	1215	304	1519
INRIA & PSU	3047	1045	4092
Dataset proposto	2551	640	3191

Fonte: Autor

Com todos os *datasets* anotados e configurados para o treinamento, a etapa de preparação dos *datasets* é finalizada. Essa etapa é fundamental para que o treinamento dos modelos possa ser realizado da melhor forma.

4.4 Executar treinamento YOLOv3

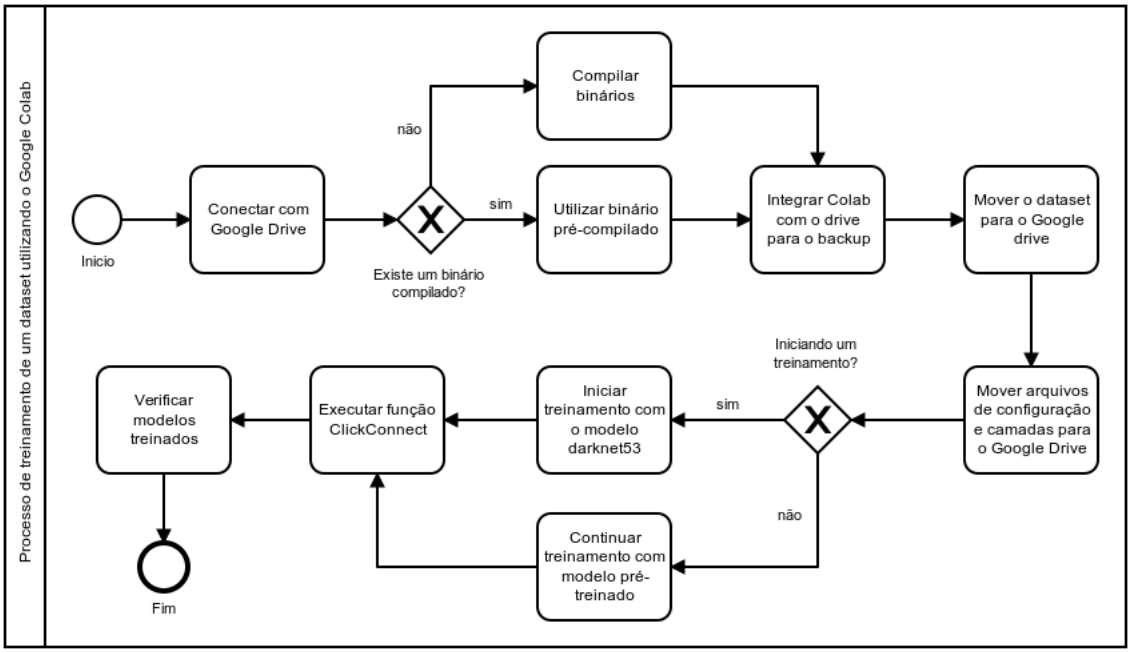
Esta seção tem como objetivo demonstrar como foi realizado o treinamento dos modelos com o YOLOv3. Vale salientar que, por questões de desempenho e *hardware*[14], não foi possível realizar o treinamento utilizando a máquina descrita na seção 4.1 em tempo hábil. Apesar dela ser capaz de realizar o treinamento, a sua placa de vídeo, uma NVIDIA GeForce GTX 1050Ti *Mobile*, não oferece um desempenho aceitável para o treinamento.

Logo, optou-se por utilizar o Google Colab[15], que nada mais é do que um ambiente virtual que permite a execução de códigos python diretamente do navegador, sendo, na prática, um serviço que permite a execução de notebooks Jupyter[16]. Ele é especialmente adequado para projetos que envolvem ciência de dados e aprendizado de máquina (GOOGLE, 2021), e disponibiliza GPU's Nvidia K80s, T4s, P4s e P100s[17]

[13] Como as amostras negativas não possuem anotações, são apenas arquivos vazios
[14] Parte física do computador
[15] <https://colab.research.google.com/>
[16] Interface gráfica que permite a edição e execução de códigos de diversas linguagens em um navegador web
[17] Especificadas até Outubro de 2021, porém, assim como dito pelo Google no FAQ, estão sujeitas à alterações.

O fluxo de execução do treinamento utilizando o Google Colab é apresentado de forma resumida na Figura 4.8, servindo como um guia para os passos que serão apresentados.

Figura 4.8 – Fluxo de execução de treinamento de um Dataset utilizando o Google Colab.



Fonte: Autor

1. **Conectar com Google Drive**:

   A primeira etapa consiste em conectar o Google Colab com o Google Drive, visto que todos os dados necessários para a realização do treinamento estarão armazenados no *drive*, inclusive o *backup* dos modelos treinados. Essa etapa é crucial para a execução do treinamento, sendo necessário executar a seguinte instrução:

   ```
   from google.colab import drive
   drive.mount('/content/drive')
   ```

2. **Compilar binários**:

   Caso seja a primeira vez que o treinamento esteja sendo executado, não haverá um binário salvo no *drive*. Logo, é necessário clonar o código fonte do YOLOv3 e compilá-lo, gerando o binário necessário para o treinamento, conforme as instruções a seguir. Vale salientar que o Google colab permite a execução de instruções *bash* desde que o comando seja prefixado com o símbolo "!".

   ```
   $ !git clone https://github.com/kriyeng/darknet/
   $ %cd darknet
   $ !git checkout feature/google-colab
   ```

```
$ !make
$ !cp ./darknet /content/drive/MyDrive/darknet/darknet
```

nas instruções anteriores, o código fonte do YOLOv3 é baixado para o *colab*, sendo utilizado um *fork*[18] disponibilizado por Ibáñez (2019). O *fork* foi utilizado por possuir algumas otimizações para o uso no Google Colab, como salvar os pesos do modelo a cada mil iterações[19] e o cálculo de métricas como *maP* diretamente a partir do binário gerado.

Após o repositório ser clonado, a *branch* é alterada para `feature/google-colab`, e o código é então compilado. Ao final, o binário gerado é salvo no Google drive, dentro de uma pasta chamada **darknet/**.

Vale salientar que esse passo deve ocorrer apenas caso o binário **darknet** não tenha sido gerado anteriormente. Após a primeira execução e a certeza de que o binário está salvo no google drive, deve-se comentar todas as linhas acima, prefixando um "#". Não há problemas em executar essa instrução diversas vezes, porém apenas fará com que o tempo de uso e recursos do Google colab sejam desperdiçados.

3. **Utilizar binário pré-compilado**:

   Caso o binário já esteja salvo no Google drive, é necessário apenas copiá-lo para o ambiente do *colab* e alterar as suas permissões, o que pode ser feito de acordo com as instruções a seguir:

   ```
   $ !mkdir darknet
   $ %cd darknet
   $ !cp /content/drive/MyDrive/darknet/bin/darknet ./darknet
   $ !chmod +x ./darknet
   ```

4. **Integrar Colab com o drive para o backup**:

   Este passo consiste em realizar um *link* simbólico entre o ambiente do *colab* e o Google Drive, de forma que os pesos salvos durante a execução do treinamento sejam salvos diretamente na pasta especificada no Drive. Isso acontece com a seguinte instrução:

   ```
   $ !ln -s /content/drive/MyDrive/darknet/backup \
   $ /content/darknet/backup
   ```

5. **Mover os *datasets* para o Google drive**:

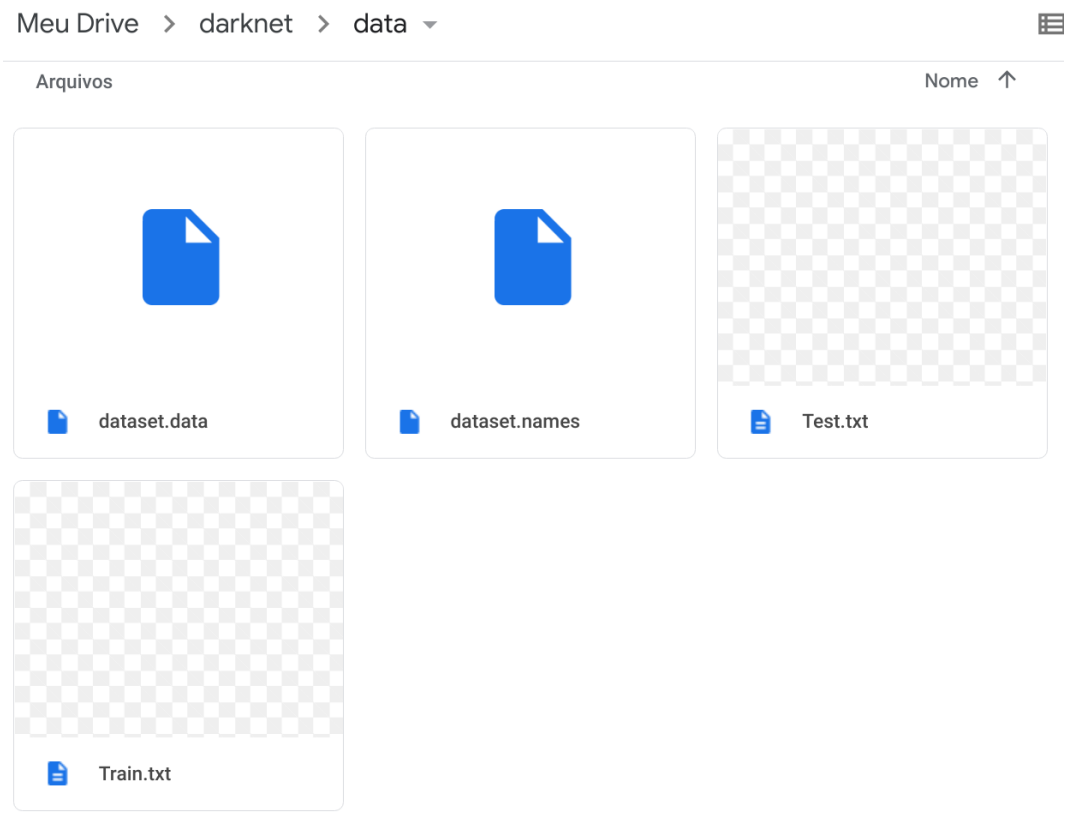
   Antes de iniciar o treinamento dentro do ambiente do Colab, é imprescindível que os arquivos dos *datasets* estejam no Google Drive. Neste trabalho, optou-se por

[18] Repositório criado a partir de outro repositório inicial
[19] O valor padrão do YOLOv3 é a cada 10.000 iterações

compactar os arquivos dentro de uma pasta chamada **data/**, contendo as amostras de treino e teste, gerando um arquivo com extensão `.zip`. Esse arquivo foi adicionado ao Google drive dentro da pasta **darknet/**.

Na sequência, é criada uma pasta chamada **data/**, porém dentro da pasta **darknet**, no Google drive. Nela são adicionados quatro arquivos, sendo eles: `Test.txt`, `Train.txt`, `obj.names` e `obj.data`, conforme visto na Figura 4.9. Vale salientar que as instruções para gerar os arquivos acima foram discutidas na Seção 4.3.3.

Figura 4.9 – Arquivos dentro da pasta darknet/data/, no Google Drive.



Fonte: Autor

6. **Mover arquivos de configuração e camadas para o Google Drive**:

   Uma vez que os arquivos do(s) *dataset(s)* estejam dentro do *drive*, o próximo passo consiste em configurar e enviar o arquivo de configuração utilizado pelo YOLOv3, `yolov3.cfg`. Apesar de o arquivo de configuração estar contido ao se clonar o repositório no passo **Compilar binários**, é necessário realizar algumas alterações para se trabalhar apenas com a classe *person*. Logo, o primeiro passo é baixar o arquivo `yolov3.cfg` na máquina local (e não dentro do colab). Para isso, deve-se executar o seguinte comando:

```
$ wget https://github.com/kriyeng/darknet/blob/master/cfg/yolov3.cfg
```

Na sequência, é necessário editar o arquivo. De acordo com Bochkovskiy, Wang e Liao (2020), as seguintes alterações devem ser feitas:

- A linha **batch** deve ser alterada para `batch=64`;
- A linha **subdivisions** deve ser alterada para `subdivisions=16`;
- A linha **max_batches** deve ser alterada para `max_batches=6000`;
- A linha **steps** deve ser alterada para 80% e 90% do valor de **max_batches**, respectivamente. Nesse caso, para `steps=4800,5400`;
- As linhas **widht** e **height** devem ser alteradas para 416 ou qualquer múltiplo de 32;
- Em cada uma das três camadas iniciadas com a expressão [**yolo**], deve-se alterar a linha **classes** para `classes=1`, visto que deseja-se detrectar apenas uma classe, e
- Em cada uma das três camadas convolucionais ([**convolutional**]) antes das camadas [**yolo**], deve-se alterar a linha **filters** para `filters=18`.

Tais alterações são necessárias para ajustar o algoritmo para o treinamento com uma classe. Elas visam realizar tal adequação, otimizando o treinamento.

É necessário também baixar os pesos convolucionais pré-treinados no *Imagenet*, ou os pesos do modelo *darknet53*. Eles serão usados como os pesos iniciais para o treinamento do modelo. Para isso, é necessário executar a instrução abaixo, também na máquina local:

```
$ wget https://pjreddie.com/media/files/darknet53.conv.74
```

Por fim, deve-se enviar ambos os arquivos para a pasta **darknet/** dentro do Google Drive. Com isso, o treinamento estará pronto para ser inicializado.

7. **Iniciar treinamento com o modelo darknet53 ou continuar a partir do modelo pré-treinado**

   Com todos os arquivos necessários no Google drive, o próximo passo consiste em descompactar o arquivo com as amostras do *dataset*. Isso é realizado a partir da instrução abaixo:

   ```
   $ !unzip "/content/drive/MyDrive/darknet/data.zip" \
   $ -d "/content/darknet"
   ```

   Onde `data.zip` é o arquivo compactado contendo o *dataset*, assim como descrito no passo **Mover o *dataset* para o Google drive**. A descompactação é feita

diretamente para o ambiente do colab, logo deve-se haver atenção com os limites de uso do disco da ferramenta.

Em seguida, deve-se alterar o Ambiente de execução para utilizar uma GPU. Para isso, é preciso, no *colab*, clicar no menu **Ambiente de execução** e depois na opção **Alterar tipo de ambiente de execução**. Será aberta uma janela com a opção **Acelerador de hardware**, o qual deve ser alterado para **GPU**, salvando essa opção ao final. Com o ambiente alterado para ser executado utilizando uma GPU, o treinamento pode então ser iniciado. Para isso, as instruções utilizadas são:

```
$ %cd /content/darknet
$ !./darknet detector train \
$ "/content/drive/MyDrive/darknet/data/obj.data" \
$ "/content/drive/MyDrive/darknet/yolov3.cfg" \
$ "/content/drive/MyDrive/darknet/darknet53.conv.74" \
$ -dont_show
```

Nota-se que os arquivos especificados são indicados diretamente pelo seu caminho no Google drive, por isso a importância de se configurar e subir corretamente o *dataset* e o arquivo de configuração no Google drive. Caso a intenção seja continuar o treinamento a partir da ultima versão de pesos salvos, deve-se alterar a linha

```
$ "/content/drive/MyDrive/darknet/darknet53.conv.74"
```

Para:

```
$ "/content/drive/MyDrive/darknet/backup/yolov3_last.weights"
```

Com essa alteração, o binário irá utilizar o último modelo de pesos salvos na pasta **backup/** dentro da pasta **darknet/**, no Google Drive. O arquivo `yolov3_last.weights` é gerado automaticamente a cada 100 iterações feitas no treinamento, garantindo que, caso algum problema ocorra durante o treinamento, seja possível continuar de um ponto mais próximo, sem que todo o treinamento seja perdido. A Figura 4.10 mostra um exemplo do *log* de saída ao se executar o treinamento no Google colab.

8. **Executar função ClickConnect**:

   Um fato importante a ser salientado é que o *colab* possui um tempo de inatividade, desconectando o usuário caso ele não perceba nenhum tipo de iteração após certo tempo. Logo, é utilizado um *script javascript* que emula cliques na plataforma, "enganado"o *colab* a pensar que é um usuário interagindo:

```
function ClickConnect(){
    console.log("Working");
```

Figura 4.10 – Exemplo de saída do treinamento numa célula do Google Colab.

```
 19989: 0.545906, 0.542252 avg loss, 0.000010 rate, 3.619701 seconds, 1279296 images
Loaded: 0.000039 seconds

 19990: 0.831716, 0.571198 avg loss, 0.000010 rate, 3.628995 seconds, 1279360 images
Resizing
352 x 352
 try to allocate additional workspace_size = 69.98 MB
 CUDA allocate done!
Loaded: 0.000040 seconds

 19991: 0.537650, 0.567843 avg loss, 0.000010 rate, 4.424898 seconds, 1279424 images
Loaded: 0.000043 seconds

 19992: 0.618795, 0.572938 avg loss, 0.000010 rate, 4.484376 seconds, 1279488 images
Loaded: 0.000044 seconds
```

Fonte: Autor

```
    document
        .querySelector("#top-toolbar > colab-connect-button")
        .shadowRoot.querySelector("#connect").click();
}
setInterval(ClickConnect,120000)
```

O *script* funciona a partir da execução da função `ClickConnect`, sendo executada a cada 20 minutos. Porém, vale salientar que após 12 horas o ambiente irá automaticamente se desconectar, de forma obrigatória.

9. **Verificar modelos treinados**:

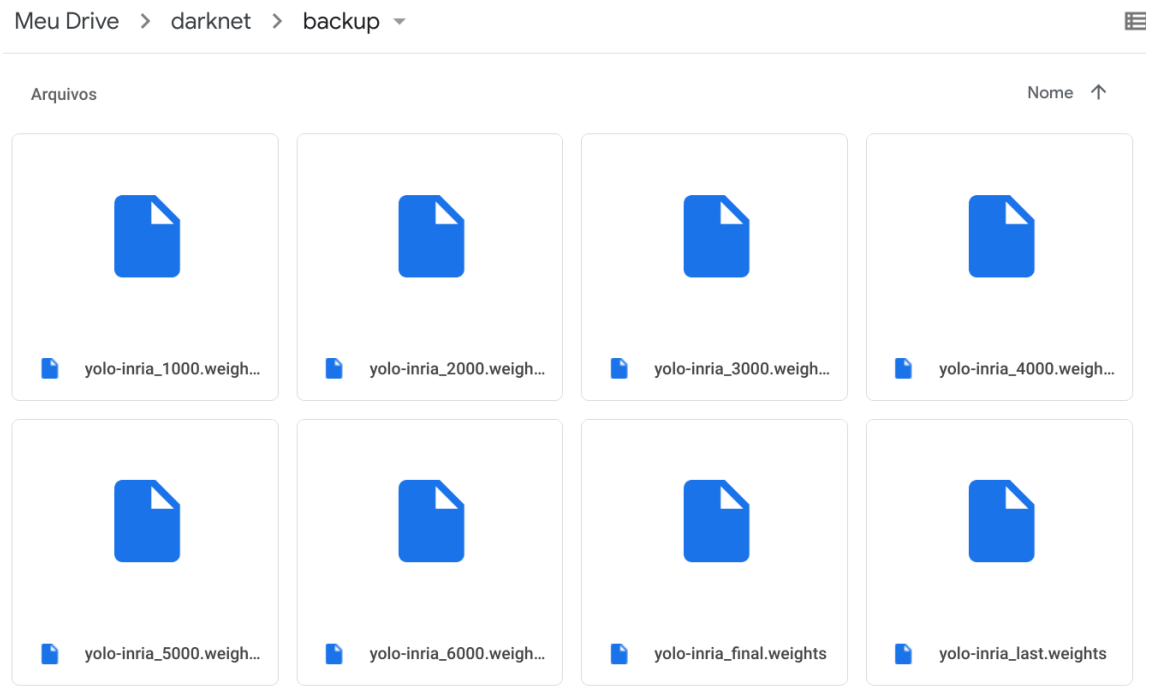
   Uma vez que o treino tenha sido finalizado, os pesos finais estarão salvos na pasta de *backup* definida pelo usuário. A Figura 4.11 mostra os arquivos salvos no Google drive, sendo possível observar também os pesos salvos a cada 1000 execuções[20]. O peso final é chamado `yolov3_final.weights`, sendo o modelo gerado ao término do treinamento. Os demais arquivos, além de servirem como *backup*, também podem ser utilizados para verificar a performance dos modelos com o incremento de *batches*, verificando se houve ou não sobreajuste[21] na hora de validá-los.

Note que os arquivos são enviados automaticamente para o Google drive, porém é necessário que haja espaço disponível para tal. Caso o espaço disponível no Google Drive não seja suficiente, o *backup* não será realizado, ocorrendo um erro durante o treinamento.

Ao final desse processo, é possível realizar as predições de pedestres nos *datasets* usados neste trabalho. Note que o processo de treinamento é consideravelmente genérico, podendo ser utilizado em outros contextos que não a detecção de pedestres.

[20] Os arquivos são salvos com o nome yolov3_N.weights, onde N é o número total de execuções
[21] Quando o modelo se ajusta muito bem durante o treino, mas não performa bem nos treinos

Figura 4.11 – Arquivos de pesos salvos na pasta de backup no Google drive.



Fonte: Autor

4.5 Agregar e avaliar resultados

Os resultados obtidos ao testar o algoritmo YOLOv3 podem ser vistos na Tabela 4.6. Nela, são descritos os resultados de aP nos três conjuntos de dados utilizados neste trabalho[22], e também os valores obtidos com o modelo padrão da rede YOLOv3. Para o *dataset* INRIA, os valores de aP obtidos são superiores à 91%. Tais valores diminuem consideravelmente ao se avaliar o PSU, obtendo-se os valores 76,02% e 67,62% para a rede padrão e a rede treinada do YOLOv3, respectivamente. Por fim, a rede padrão do YOLOv3 performa ligeiramente melhor no conjunto INRIA & PSU do que no conjunto PSU, porém ainda bem inferior em comparação com o INRIA.

Em ambos os casos, a rede padrão se mostrou mais performática que as redes treinadas no *colab*; uma possível explicação para isso é o fato de a rede padrão ser treinada utilizando o *dataset* COCO, que possui consideravelmente mais imagens que o PSU e o INRIA juntos[23]

Para o *dataset* proposto, observa-se os valores de 53,82% para o modelo gerado a partir do próprio conjunto de imagens, e 58,80% para o modelo padrão do YOLOv3. Aqui, observa-se uma acurácia inferior aos demais *datasets*, mesmo com o modelo padrão da rede YOLOv3; tal fato pode ocorrer pela própria anotação do *dataset* em si, que considerou pessoas distantes para a marcação (conforme descrito na seção 4.3.4), pela qualidade de

[22] PSU, INRIA e INRIA & PSU e o Dataset Proposto.
[23] Mais de 200000 mil imagens anotadas contra pouco mais de 4000 para o PSU e INRIA combinados.

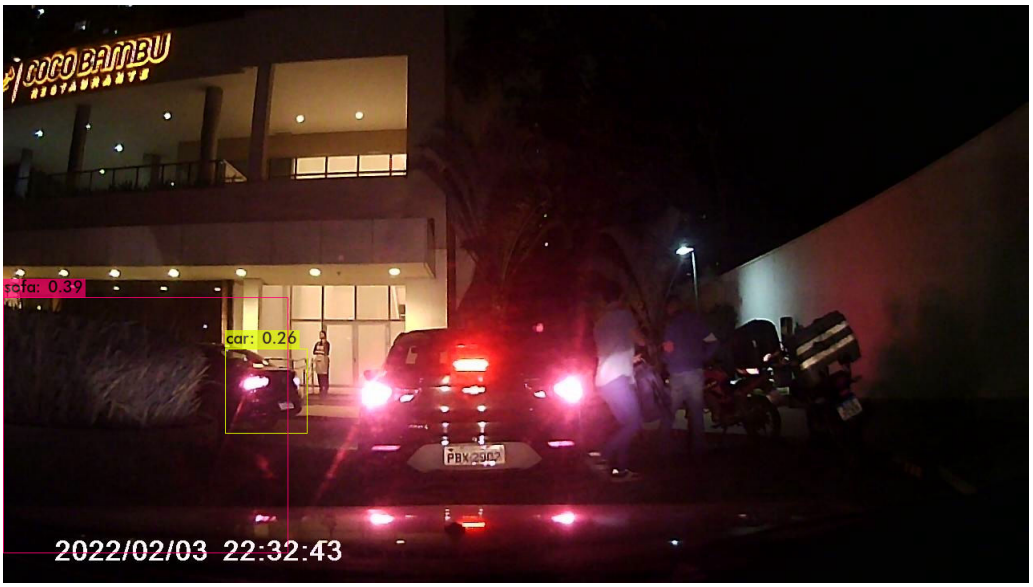
Tabela 4.6 – aP no algoritmo YOLOv3 para a classe Pessoa

Dataset	**Modelo**	**aP** (%)
INRIA	Yolo padrão (COCO *dataset*)	96,18
INRIA	Yolo custom (INRIA)	91,41
PSU	Yolo padrão (COCO *dataset*)	76,02
PSU	Yolo custom (PSU)	67,62
INRIA & PSU	Yolo padrão (COCO *dataset*)	79,29
INRIA & PSU	Yolo custom (INRIA & PSU)	66,73
Dataset Próprio & PSU	Yolo padrão (COOCO *dataset*)	58,80
Dataset Próprio & PSU	Yolo custom (Dataset próprio)	53.82

Fonte: Autor

imagem da câmera e também pelo fato de o algoritmo não funcionar tão bem em cenários noturnos. A Figura 4.12 exemplifica esse cenário, exibindo uma amostra de treino com as predições marcadas pelo YOLOv3. Nela, verifica-se que a rede detectou apenas um veículo (*car*), apesar de haverem dois, e um falso positivo de um sofá, onde na verdade há apenas vegetação. Nenhuma das três pessoas presentes na imagem foram detectadas.

Figura 4.12 – Amostra de treinamento do Dataset proposto com a predição feita pelo YOLOv3

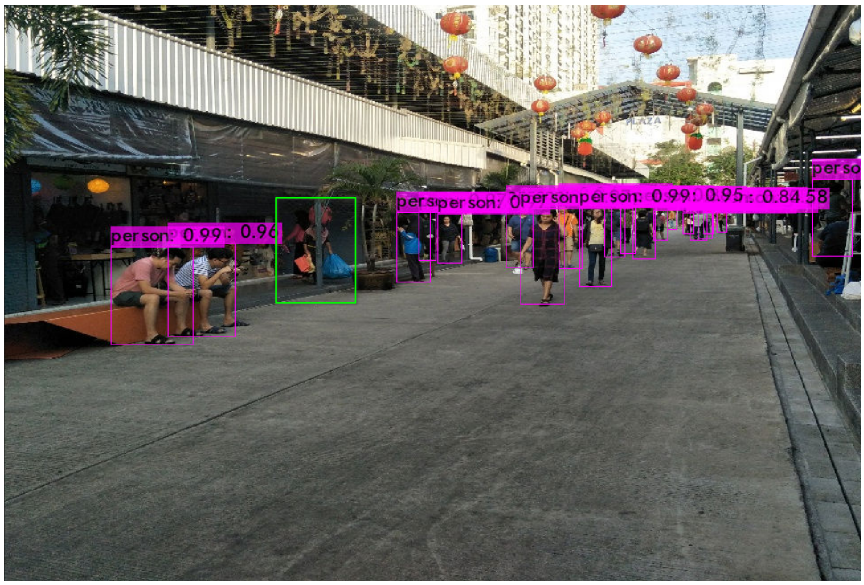


Fonte: Autor

Para o PSU, observam-se valores de aP mais baixos, mesmo com a rede padrão do YOLOv3. Isso pode acontecer pelo fato de o PSU possuir imagens de multidões e pessoas parcialmente obstruídas, além de ter sido anotado de forma bastante rigorosa, conforme dito na Seção 4.3.2. A Figura 4.13 exemplifica esse fato: nela, é realizada uma detecção utilizando-se o modelo padrão do YOLOv3 onde, apesar de muitas pessoas serem de fato

identificadas corretamente, algumas pessoas parcialmente obstruídas não são detectadas (marcadas manualmente em verde).

Figura 4.13 – Imagem de exemplo mostrando as predições realizadas pela rede padrão do YOLOv3 e a marcação de um falso negativo (em verde)



Fonte: Adaptado de Thu, Suvonvorn e Karnjanadecha (2018)

Outro ponto a ser notado é que os valores de aP obtidos com a validação no *dataset* INRIA não são confiáveis. Isso acontece por, conforme dito por Dalal (2005), nem todas as anotações estarem corretas. Tal fato fica explícito na Imagem 4.14, onde três pessoas são corretamente anotadas, porém uma quarta pessoa, parcialmente oclusa, não é. Esse fato se repete em diversas imagens do *dataset*, tornando os valores de aP não confiáveis.

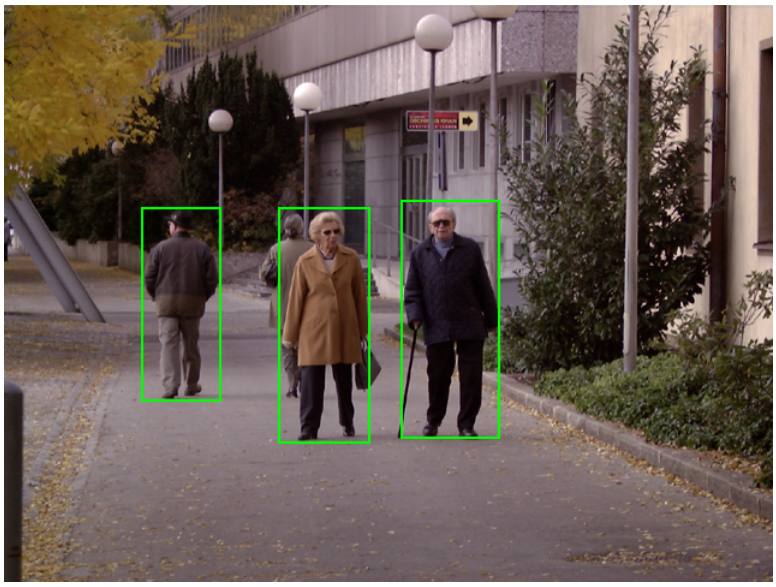
Logo, optou-se por re-anotar todas as imagens do *dataset*, sem utilizar nenhuma das anotações já existentes. O objetivo foi de verificar se a métrica obtida foi influenciada por uma possível marcação incorreta, buscando revalidar os resultados. O processo de remarcação seguiu os mesmos passos descritos na seção 4.3.2, e a Tabela 4.7 mostra a quantidade de caixas delimitadoras do *dataset* padrão e as obtidas após a re-anotação. Nela, verifica-se a quantidade de 1826 e 4776, para o *dataset* padrão e o re-anotado, respectivamente, ou um aumento de aproximadamente 161%.

Tabela 4.7 – Quantidade de marcações no *dataset* INRIA

Dataset	**Caixas delimitadoras**
INRIA Padrão	1826
INRIA Re-anotado	4776

Fonte: Autor

Figura 4.14 – Uma das amostras de treino do INRIA com as anotações padrão desenhadas em verde.



Fonte: Adaptado de Dalal (2005)

Após a remarcação, o *dataset* re-anotado foi treinado novamente, de forma que pudesse ser re-avaliado. O treinamento seguiu os passos descritos na seção 4.4, e os resultados obtidos podem ser vistos na Tabela 4.8, onde verificam-se os valores de 89,47% e 69,05% para os modelos Padrão e customizado do YOLOv3, respectivamente. Tais valores representam uma redução de **6,41%** e **22,36%** para os modelos, sendo bastante considerável para o modelo customizado[24]. No caso do modelo treinado no *dataset* COCO, apesar de a redução ser pequena, ela indica que de fato as marcações incorretas dos dados geraram resultados não confiáveis, mostrando ser necessário olhá-los com cautela sem antes analisar as imagens e marcações.

Tabela 4.8 – aP no algoritmo YOLOv3 para o INRIA Re-anotado

Dataset	**Modelo**	**aP** (%)
INRIA Re-anotado	Yolo padrão (COCO *dataset*)	89,47
INRIA Re-anotado	Yolo custom (INRIA Re-anotado)	69,05

Fonte: Autor

[24] Treinado a partir das próprias imagens e anotações do INRIA re-anotado

Parte V

Conclusão

5 Conclusão

Nota-se, no decorrer do Capítulo 4, que os principais objetivos foram alcançados com o desenvolvimento do projeto. Em especial, pode-se citar:

- **Identificar e avaliar *datasets* já existentes com imagens de pedestres para treinamento e teste do algoritmo**:

  Verifica-se esse a realização desse objetivo nas Seções 4.3 - Preparação dos *datasets* para utilização e 4.5 - Agregar e avaliar resultados, onde os *datasets* escolhidos - PSU e INRIA Person - são preparados para a utilização no YOLOv3. Em especial, o *dataset* PSU é completamente rotulado, e todo o processo é documentado de forma que qualquer pesquisador possa reproduzir os passos em outros *datasets*.

  Durante a avaliação dos algoritmos, verificou-se que o *dataset* INRIA Person não possuía uma rotulagem precisa, onde determinadas amostras não estavam com pessoas obstruídas anotadas. Tal fato não ocorreu no *PSU*, onde a rotulagem foi realizada de forma minuciosa.

- **Elaborar um novo *dataset* com imagens capturadas a partir de uma câmera veicular**:

  Verifica-se esse a realização desse objetivo nas Seções 4.2 - Elaboração do *dataset* e 4.3.4 - *Dataset* proposto. Nelas, são detalhados os processos de captura e organização das imagens. Foram capturados mais de 2 horas de vídeo, sendo percorridos mais de 50 quilômetros e geradas mais de 13 mil imagens.

  Ao final, mais de 2500 imagens foram rotuladas, e o modelo gerado obteve uma aP de aproximadamente 58% na rede padrão do YOLOv3 (COCO *dataset*), valor que indica a possibilidade de melhorias no processo de captura e geração do *dataset*.

- **Documentar o processo de treinamento e avaliação utilizando o algoritmo YOLOv3**:

  Verifica-se esse a realização desse objetivo nas Seções 4.4 - Executar treinamento YOLOv3 e 4.5 - Agregar e avaliar resultados. Nelas, os detalhes de treinamento utilizando o YOLOv3 são documentados, sendo feita a sua avaliação ao final.

  Observa-se que a rede padrão utilizada pelo YOLOv3 performou melhor que todos os modelos treinados. Observa-se também o fato de não apenas os modelos treinados mas também a rede padrão do YOLOv3 performarem de forma não tão satisfatória no *dataset* PSU. Aqui cabe a possibilidade de uma análise mais profunda, de forma a verificar em quais amostras o algoritmo performou de maneira fraca, tentando

agrupar caraterísticas em comum nessas amostras, além de verificar o impacto das não-detecções no contexto de veículos autônomos. Para o *dataset* proposto, os valores de aP obtidos não foram tão altos, indicando a possibilidade de melhorias em trabalhos futuros.

Com a realização de tais objetivos, verifica-se que os objetivos propostos para a conclusão no TCC foram feitos: foi mostrado como detectar pedestres utilizando o algoritmo YOLOv3 e o processo de preparação dos dados e do treinamento foi documentado, além da captura do novo *dataset*. Ainda existem melhorias e novas análises passíveis de serem feitas, que serão descritas na próxima seção.

5.1 Trabalhos futuros

Conforme visto na seção 4.5, os resultados obtidos não foram tão altos. Possíveis fatores que explicam tal resultado são o fato de o YOLOv3 não detectar objetos tão precisamente em imagens com pouca iluminação, e a qualidade das imagens também não ser tão alta. Logo, trabalhos futuros podem explorar esse resultado, buscando melhorar a precisão do modelo gerado. Dentre as possibilidades, capturar um novo *dataset* utilizando uma câmera de maior qualidade e aumentar a quantidade de amostras, principalmente noturnas, podem ajudar a aumentar a precisão dos modelos.

Pode-se também focar em analisar os resultados com ênfase nas falhas, buscando descobrir quais amostras mais tiveram falsos positivos e falsos negativos, agrupando-as por características em comum. Isso permitiria melhorar os agrupamentos de falhas, facilitando a documentação e a geração de novos *datasets*. A verificação de marcações para *datasets* já existentes também pode ser um importante fator de análise em trabalhos futuros, tal qual ocorreu ao se rotular novamente o *dataset* INRIA, conforme a seção 4.5, onde de fato percebeu-se que uma marcação mais criteriosa fez o YOLOv3 performar de forma inferior.

Por fim, trabalhos futuros também podem buscar utilizar versões mais recentes do algoritmo YOLO, como o YOLOv4 ou YOLOv5, comparando as métricas descritas nesse trabalho de forma a verificar o impacto das atualizações no algoritmo.

Referências

BLACKBOX.CAMERAS.VEICULARES. *Câmera Veicular Automotiva Carro - Black Box Gp2 - Hd Real*. 2022. Urlhttps://produto.mercadolivre.com.br/MLB-1238928639-cmera-veicular-automotiva-carro-black-box-gp2-hd-real-$_JM?quantity = 1.*Citadonapágina*51.

BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. Citado 2 vezes nas páginas 55 e 68.

CACILO, A. et al. *Hochautomatisiertes Fahren auf Autobahnen - Industriepolitische Schlussfolgerungen*. 2015. Citado na página 23.

DALAL, N. *INRIA Person Dataset*. 2005. Disponível em: <http://pascal.inrialpes.fr/data/human/>. Citado 3 vezes nas páginas 56, 73 e 74.

DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. [S.l.: s.n.], 2005. v. 1, p. 886–893 vol. 1. Citado na página 29.

DANAPAL, G. et al. Sensor fusion of camera and lidar raw data for vehicle detection. In: . [S.l.: s.n.], 2020. p. 1–6. Citado na página 23.

DOWLING, K. et al. *NAVLAB: An Autonomous Navigation Testbed*. Pittsburgh, PA, 1987. Citado na página 23.

FONSECA, J. J. S. da. *Metodologia da Pesquisa científica*. Universidade Estadual do Ceará, 2002. 127 p. Disponível em: <http://www.ia.ufrrj.br/ppgea/conteudo/conteudo-2012-1/1SF/Sandra/apostilaMetodologia.pdf>. Citado na página 40.

GAD, A. F. *Evaluating Object Detection Models Using Mean Average Precision (mAP)*. 2020. Disponível em: <https://blog.paperspace.com/mean-average-precision/>. Citado na página 31.

GIL, A. C. *Metodos e Tecnicas de Pesquisa Social*. 6. ed. [S.l.]: ATLAS - GRUPO GEN, 2008. ISBN 978-8522451425. Citado 2 vezes nas páginas 35 e 36.

GOOGLE. *Colaboratory: Frequently Asked Questions*. 2021. Disponível em: <https://research.google.com/colaboratory/faq.html>. Citado na página 64.

IBÁÑEZ, D. *Yolo-v3 and Yolo-v2 for Windows and Linux*. GitHub, 2019. Disponível em: <https://github.com/kriyeng/darknet/>. Citado na página 66.

INTEL, N. *Intel Predicts Autonomous Driving Will Spur New 'Passenger Economy' Worth $7 Trillion*. 2017. Disponível em: <https://newsroom.intel.com/news-releases/intel-predicts-autonomous-driving-will-spur-new-passenger-economy-worth-7-trillion/>. Citado na página 23.

KNIBERG, H. *Kanban and Scrum - making the most of both (Enterprise Software Development)*. Paperback. [S.l.]: lulu.com, 2010. 120 p. ISBN 978-0557138326. Citado na página 37.

MOSES; OLAFENWA, J. *ImageAI, an open source python library built to empower developers to build applications and systems with self-contained Computer Vision capabilities*. 2018–. Disponível em: <https://github.com/OlafenwaMoses/ImageAI>. Citado na página 53.

NOGUEIRA, J. H. dos; GUIMARÃES, O. *Método para manutenção de sistema de software utilizando técnicas arquiteturais*. 101 p. Dissertação (Mestrado em Sistemas Digitais) — Universidade de São Paulo, São Paulo, 2008. Citado na página 47.

OSORIO, F.; HEINEN, F.; FORTES, L. Controle inteligente de veículos autônomos: Automatização do processo de estacionamento de carros. 01 2001. Citado na página 23.

PRODANOV, C. C.; FREITAS, E. C. de. *Metodologia do Trabalho Científico: Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico*. 2. ed. Editora Feevale, 2013. ISBN 9788577171583. Disponível em: <https://books.google.com.br/books?id=zUDsAQAAQBAJ>. Citado 2 vezes nas páginas 35 e 36.

REDMON, J. *Darknet: Open Source Neural Networks in C*. 2013–2016. <http://pjreddie.com/darknet/>. Citado na página 27.

REDMON, J. et al. You only look once: Unified, real-time object detection. In: . [S.l.: s.n.], 2016. p. 779–788. Citado 3 vezes nas páginas 27, 28 e 29.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *arXiv*, 2018. Citado na página 27.

REZENDE, J. L. de. Aplicando técnicas de conversacão para facilitacão de debates no ambiente aulanet. In: . Rio de Janeiro, 2003. Disponível em: <http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0115649_03_cap_08.pdf>. Citado na página 37.

SAE. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. 2021. Disponível em: <https://www.sae.org/standards/content/j3016_202104/>. Citado na página 23.

SANTOS, L. C. B. d. *Implantação de veículos autônomos no contexto brasileiro : avaliação dos fatores que influenciam no interesse de uso com equações estruturais*. 171 f. Monografia (Especialização) — Faculdade de Tecnologia, Universidade de Brasília, Brasìlia / DF, 2017. Citado na página 23.

THU, M.; SUVONVORN, N.; KARNJANADECHA, M. A new dataset benchmark for pedestrian detection. In: . [S.l.: s.n.], 2018. p. 17–22. Citado 3 vezes nas páginas 30, 48 e 73.

TZUTALIN. *LabelImg*. GitHub, 2015. Disponível em: <https://github.com/tzutalin/labelImg>. Citado na página 57.

XU, Z. *Pedestrian Detection YOLOv3 in INRIA*. GitHub, 2020. Disponível em: <https://github.com/Zyjacya-In-love/Pedestrian_Detection_YOLOv3_in_INRIA>. Citado 2 vezes nas páginas 56 e 60.

Apêndices

APÊNDICE A – Utilizando o dataset proposto

O *dataset* proposto está disponibilizado no *Github*, uma plataforma de hospedagem de código-fonte e arquivos com controle de versão usando o *Git*[1]. Para utilizá-lo, é necessário *clonar* o repositório do *dataset*, seguindo as instruções abaixo:

```
$ git clone https://github.com/ebmm01/person-detection-dataset.git
```

Na sequência é necessário entrar no diretório com os arquivos:

```
$ cd person-detection-dataset
```

O diretório possuirá duas pastas, *Train* e *Test* contendo as imagens e suas respectivas rotulagens, além de quatro arquivos:

- **Train.txt**: Arquivo indicando as amostras utilizadas para treino;
- **Test.txt**: Arquivo indicando as amostras utilizadas para teste;
- **dataset.names**: Arquivo que indica as classes de treino e teste utilizadas. Como o foco é a detecção de pessoas, o arquivo possui apenas uma linha escrita *person*, e
- **dataset.data**: Arquivo que contém informações utilizadas pelo YOLOv3 para o treinamento. Ele indica os arquivos de treino e teste, além do arquivo de classes e a pasta onde será salvo o *backup* dos modelos gerados ao longo do treinamento.

As instruções para o treinamento utilizando o YOLOv3 estão descritas na seção 4.4.

[1] Um sistema de controle de versão de arquivos, usado principalmente no desenvolvimento de software

APÊNDICE B – Anotações dos datasets

As anotações dos *datasets* INRIA and PSU estão disponibilizadas no *Github*, uma plataforma de hospedagem de código-fonte e arquivos com controle de versão usando o *Git*. Para utilizá-lás, é necessário *clonar* o repositório do *dataset*, seguindo as instruções abaixo:

```
$ git clone https://github.com/ebmm01/psu-and-inria-yolo-annotations.git
```

Na sequência é necessário entrar no diretório com os arquivos:

```
$ cd psu-and-inria-yolo-annotations
```

O diretório possuirá duas pastas, inria/ e psu/, contendo as rotulagens dos respectivos *datasets*.