Contagem e Classificação de Veículos por Visão Computacional

André Vitor Maus
Faculdade de tecnologia SENAI
Joinville
Brasil
andrevmaus@gmail.com

Roney Kuntz
Faculdade de tecnologia SENAI
Joinville
Brasil
roneykz@gmail.com

Valdir Linhares Junior
Faculdade de tecnologia SENAI
Joinville
Brasil
valdirlinharesjunior@gmail.com

Tatiele Martins Razera
Faculdade de tecnologia SENAI
Joinville
Brasil
tatiele.razera@edu.sc.senai.br

Ademir Camillo Junior
Faculdade de tecnologia SENAI
Joinville
Brasil
ademir.camillo@edu.sc.senai.br

ABSTRACT

The increase in the number of vehicles, mainly on urban roads, generates a series of new challenges for traffic managers, being extremely important to know the vehicles that travel on the roads, as well as which routes are most used in intersections. Thus, in this project an application based on computer vision is being presented, capable of detecting vehicles separated by types and which routes are being traveled, thus generating reports that contribute to the organization of traffic.

PALAVRAS-CHAVE

Análise de trânsito; Contagem de veículos; Reconhecimento de Objetos; *OpenDataCam*.

1 INTRODUÇÃO

De acordo com o Denatran, o Brasil apresenta um carro para cada 4,4 habitantes [1], o que gera uma série de dificuldades para os gestores de trânsito.

Conhecer a quantidade de veículos, suas rotas e gerar estatísticas são métodos que colaboram para o bom gerenciamento do trânsito, principalmente em grandes cidades. Como alternativa para gerar estas informações, está o uso da Visão Computacional e técnicas de Inteligência Artificial.

A Visão Computacional vêm sendo aplicada em diferentes áreas, como na análise de vídeos de trânsito. Sendo assim a Prefeitura Municipal de Joinville-SC solicitou à Faculdade de Tecnologia Senai Joinville-SC o desenvolvimento de uma ferramenta capaz de facilitar o trabalho de contagem e classificação de veículos na cidade. Hoje o setor de análise e planejamento urbano utiliza dados de laços indutivos e análise manual de vídeos gravados com *drones*.

Dessa forma foi proposta uma aplicação *web*, a qual utilizou o sistema *OpenDataCam* [2] com *YOLO* [3], para realizar a análise de imagens de trânsito coletadas por um *drone* do ponto de vista aéreo ortogonal.

O objetivo deste trabalho é desenvolver uma aplicação *web*, utilizando bibliotecas e soluções de visão computacional, para o reconhecimento de veículos, onde será possível visualizar as informações do trânsito através de relatórios e *dashboards*, com os quais os gestores municipais poderão analisar a quantidade de veículos que trafegam nas rotas mapeadas e assim gerar soluções para melhor o fluxo de veículos da cidade..

Este artigo será dividido em 7 seções. Introdução; Ferramentas Utilizadas; Trabalhos Relacionados; Métodos; Resultados e Discussões; Considerações Finais e Referências.

2 FERRAMENTAS UTILIZADAS PARA DETECÇÃO DE VEÍCULOS

Nesta seção serão apresentadas as ferramentas utilizadas para detecção, contagem e rastreio de objetos que compõem a estrutura da aplicação *web* desenvolvida.

Para detecção de veículos em vídeo, utilizou-se o *framework* de redes neurais *Darknet*, trabalhando em conjunto com o modelo de detecção de objetos *YOLOv3* [4], a biblioteca de algoritmos de visão computacional *OpenCV* e a plataforma *CUDA* (*Compute Unified Device Architecture*) [5]. Essa última, sendo responsável por trazer o poder de processamento das *GPUs* (*Graphics Processing Units*) para as tarefas computacionais realizadas durante o processo de detecção de objetos, acelerando em torno de 500 vezes o tempo de detecção de objetos por *frame*, em comparação com tempo de detecção alcançado em CPUs (*Central Processing Units*), o que possibilita a detecção de vídeos em tempo real com o *hardware* adequado.

Capturando as informações geradas por essa estrutura de detecção de objetos está a ferramenta *OpenDataCam*. Responsável por realizar o rastreio e contagem dos veículos em vídeo e converter os dados gerados pelo modelo de detecção em arquivos Json e CSV, dos quais extraiu-se informações relevantes para criação de

relatórios e gráficos disponibilizados na aplicação após cada processamento de vídeo.

2.1 YOLO E DARKNET

O *YOLOv1* [3] (*You only look once*) é um modelo de detecção de objetos criado em 2015 por Joseph Redmon. Diferentemente dos sistemas de detecção por classificadores deslizantes e baseados em R-CNNs (Redes Neurais aplicadas por região), onde cada componente era treinado separadamente e ambas as soluções analisavam uma mesma imagem várias vezes e de forma particionada, o *YOLO* analisa toda a imagem a tratando como um único problema de regressão, posicionando as caixas delimitadoras nos objetos a serem detectados diretamente pelos *pixels* da imagem inteira, daí vem a origem do nome *YOLO*, "você só olha uma vez".

Na primeira versão do *YOLO* a arquitetura de rede convolucional utilizada era baseada na *GoogLeNet*, contando com 24 camadas convolucionais e mais 2 camadas completamente conectadas.

De acordo com [6], uma CNN (*Convolutional Neural Network*) pode ser definida como uma rede neural na qual uma ou mais camadas aplicam a operação matemática denominada "convolução" ao invés da multiplicação de matrizes padrão utilizada em redes neurais convencionais.

Na versão *YOLOv2* (ou *YOLO9000*) [7], apresentada em 2017, foi desenvolvido um algoritmo de treinamento capaz de detectar e classificar objetos. Foram treinados mais de 9000 objetos com o auxílio do *Dataset* de imagens ImageNet [8] e do *Dataset* de detecção de objetos COCO [9] (*Common Objects In Context*).

As mudanças mais significativas foram a adoção de caixas âncora, aumento na precisão, diminuição de 33% dos requisitos computacionais e a adoção de uma nova rede neural, a *Darknet-19*, com 19 camadas convolucionais e 5 camadas de redimensionamento de imagens.

No contexto da análise de imagens, pode-se dizer que cada camada de uma CNN representa um "mapa de características". Na maioria dos casos, os dados inseridos na rede neural são imagens representadas por uma matriz tridimensional, onde cada dimensão se refere a um canal de cor no formato *RGB* (*Red, Green, Blue*). Cada *pixel* da imagem é representado por uma posição na camada inicial, equivalente a uma característica específica[10].

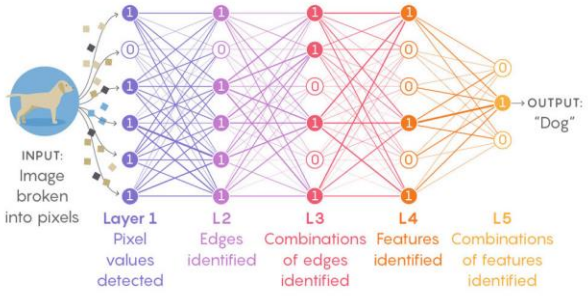

Figura 1: Fluxo de dados em uma rede neural.

Na Fig. 1 pode-se observar o fluxo percorrido pelos dados em uma rede neural artificial, onde cada camada possui uma função específica. Ao final, após identificadas e combinadas as características delimitadas na etapa de treinamento, obtém-se a saída esperada como resultado.

No YOLOv3 [4], ainda desenvolvido por Redmon, foi utilizada uma rede neural híbrida com camadas convolucionais e conexões com atalhos, conceito vindo das redes neurais residuais, contando no total com 53 camadas convolucionais, portanto passou a se chamar Darknet-53. Seus resultados de precisão se comparam a rede neural ResNet-152, porém duas vezes mais rápida na comparação de análise de *frames* por segundo (FPS).

Em uma rede neural artificial, pesos (*weights*) são os coeficientes que amplificam ou minimizam o sinal de entrada para um determinado neurônio na rede. Geralmente representados graficamente por linhas ou setas indo de um neurônio para outro. No mesmo contexto, vieses (*biases*) são valores escalares adicionados à entrada de dados para garantir que pelo menos alguns nós serão ativados por camada na rede, independente da intensidade do sinal. Vieses permitem que uma rede neural possa tentar novas adaptações ou comportamentos durante o processo de aprendizado em um modelo de detecção de objetos. Tanto os pesos quanto os vieses são modificados durante o processo de treinamento [11].

No contexto da aplicação *web* desenvolvida, a biblioteca *OpenCV* [12] auxilia fornecendo algoritmos responsáveis pela tarefa de detecção de objetos para a rede neural Darknet em conjunto com o modelo de detecção de objetos *YOLOv3*. A biblioteca *OpenCV* possui mais de 2500 algoritmos otimizados que incluem um conjunto abrangente tanto de algoritmos clássicos quanto de última geração em visão computacional e *machine learning*.

2.2 CUDA

De acordo com [5] e [13] a *CUDA* é uma plataforma de computação paralela desenvolvida pela NVIDIA, programada em ANSI C. Sua finalidade é utilizar a *GPU* paralelamente a CPU para realizar operações de forma mais rápida. As tarefas são executadas simultaneamente nos núcleos da *GPU*, chamados *CUDA Cores*, logo, a quantidade de núcleos *CUDA* é diretamente proporcional ao processamento computacional.

De acordo com [13] com o advento das *GPUs* NVIDIA *CUDA* houve um aumento de precisão nos cálculos computacionais, por exemplo, cálculos utilizando operadores "multiplicação-adição fundidos" que resultaram em *floating points* com arredondamento conforme a IEEE 754, o que não era possível com as CPUs da época.

A partir de março de 2007 *GPUs* como a NVIDIA Quadro FX 4600 e NVIDIA Tesla C870, que possuíam o processador gráfico G80 e arquitetura Tesla, passaram a ter versionamentos de bibliotecas de rede neural profunda chamadas de cuDNN (*NVIDIA CUDA Deep Neural Network*). Estas bibliotecas fornecem implementações para rotinas padrão, como convoluções, *pooling* e normalizações.

2.3 OpenDataCam

O sistema *OpenDataCam* é um sistema criado pela *Moovel Group GmbH* [2], uma empresa alemã focada no desenvolvimento de sistemas para otimização do tráfego de veículos.

OpenDataCam é uma ferramenta de código aberto, ele quantifica e rastreia objetos em movimento com análise de vídeo. Foi projetado para ser uma solução acessível e econômica, buscando melhorar a compreensão das interações em ambientes urbanos.

O *OpenDataCam* pode ser usado a partir de uma câmera conectada a um minicomputador ou a partir de imagens de trânsito pré-gravadas.

O algoritmo de detecção de objetos conta e rastreia os objetos em movimento. O sistema faz uso dos recursos da *GPU* e precisa ser executado em ambiente Linux com *hardware* habilitado para *GPU CUDA*, isso permite que sejam executadas várias *threads* simultaneamente. No lado do *software*, o *OpenDataCam* faz uso da biblioteca de detecção de objetos *YOLO*. A biblioteca é treinada para a detecção dos veículos nas imagens, portanto o vídeo fornecido alimenta a biblioteca *YOLO* que detecta os veículos em cada quadro.

3 TRABALHOS RELACIONADOS

O trabalho apresentado em [14] introduz um novo detector de veículos de multi-categoria, que atinge detecções precisas e em tempo real para imagens capturadas por *Drones* (*UAVs*).

No artigo, os autores propõem um módulo de conexão adjacente de multi escala (ACM - *adjacent connection module*), que combina imagens com alta-resolução, porém semanticamente fracas com imagens de semântica forte de uma camada adjacente mais profunda.

O módulo ACM oferece mais informações de contexto efetivo para enriquecer a representação e reduzir a interferência na detecção de veículos de pequeno porte.

Em um segundo momento, é proposta a estratégia de treinamento de perda dupla (*Alternate Double-loss Training*), que pode aprender representações de recursos para todos os exemplos, visando solucionar o problema de desbalanceamento entre a quantidade de exemplos fáceis e difíceis na detecção de veículos, onde geralmente existem poucos exemplos de difícil detecção em comparação com a grande quantidade de exemplos de fácil detecção.

Os resultados experimentais no *Stanford Drone Dataset* e no *Dataset* criado pelos autores demonstram que o método proposto pode detectar veículos de pequeno porte com alta precisão e conseguir detecções de veículos em tempo real.

Já no trabalho [15], o foco do estudo foi a criação do *dataset* de *benchmark* em larga escala *VisDrone2018*, com a intenção de fomentar a pesquisa em detecção e rastreio de objetos em vídeos gravados por *drones*. Uma vasta coleção de instâncias de objetos foi reunida, anotada e organizada para impulsionar avanços em algoritmos de detecção e rastreio de objetos, enfatizando a captura de vídeos em ambientes do cotidiano. Notavelmente, o *dataset* foi coletado em mais de 14 diferentes cidades na China com diversos modelos de *drones*, contendo diversos cenários do mundo real.

4 METODOLOGIA

O sistema de Análise de Trânsito implementado neste trabalho, foi criado com o objetivo de integrar uma interface *web* com *software* de código aberto *OpenDataCam* v3.0.1, disponibilizando um ambiente que facilita o gerenciamento e a análise de dados gerados a partir do processamento dos vídeos de trânsito.

O sistema conta com um cadastro de usuários cujo banco de dados foi preparado para uma futura integração via API (*Application Programming Interface*), com o sistema de gerenciamento de usuários da Prefeitura de Joinville-SC. O sistema de *login* da prefeitura, gerencia os dados de todos os colaboradores, no entanto quando algum usuário solicitar o acesso, o administrador do sistema deverá fazer a liberação de acesso às ferramentas de rastreio e dados de análises.

O carregamento dos vídeos será realizado diretamente no servidor, guardando assim o histórico de processamento, bem como uma verificação e validação dos metadados antes da realização do *upload* para o processamento, onde o mesmo precisará apresentar resolução mínima de 720p, tempo máximo de uma hora ou tamanho igual ou menor a 500 Mb.

Como requisitos funcionais de entrada, pode-se citar o *upload* de vídeos que obrigatoriamente devem possuir vista aérea. Também devem ser inseridos dados reais da captura do vídeo, como a data e hora de início.

A marcação das vias para análise deverá ser realizada diretamente sobre o vídeo, utilizando a interface oferecida pelo *OpenDataCam*.

Ao final do processamento são gerados relatórios em formato *CSV* e *Json*, nos quais os usuários podem verificar a contagem dos veículos, separados por tipo e rotas.

Conforme pode ser visto na Fig. 2, o *Front-End* do sistema foi desenvolvido em linguagem de marcação HTML5, um mecanismo para adicionar estilos em páginas HTML, chamado CSS3, linguagem de programação *Javascript* e fazendo uso do *framework web Bootstrap* v4.5.3.

A interface principal conta com cinco telas que podem ser acessadas a partir de um menu superior, sendo elas, *dashboard*, relatório de contagem, relatório de vídeos, tela do *OpenDataCam* e relatório de usuários.

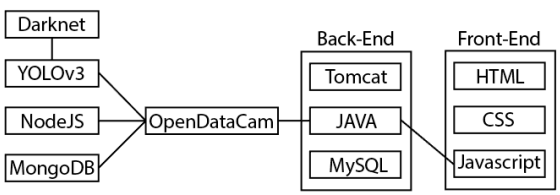

Figura 2: Diagrama de estrutura do sistema

O *Dashboard* apresenta o resumo dos dados gerados pelas análises através de gráficos criados com a biblioteca *Highcharts*

v8.2.2, que facilita a identificação do volume do tráfego no local avaliado.

A tela de relatórios carrega uma consulta com os dados do banco de dados MySQL v8.0.22 e foi criada com a biblioteca *DataTable* v1.10.22 permitindo ao usuário extrair todos os dados gerados durante as análises. Os dados podem ser exportados para os formatos CSV e XLS, além de permitir a impressão a partir de um arquivo com extensão PDF (*Portable Document Format*).

A tela de vídeos também faz uso da biblioteca *DataTable* v1.10.22 e apresenta um relatório com todos os arquivos enviados para o servidor do sistema. A partir dela pode-se escolher o vídeo que será analisado, bem como definir os parâmetros de resolução e limite de confiança, que serão usados no *OpenDataCam* durante a análise.

A tela que exibe a interface do *OpenDataCam* permite ao usuário iniciar o sistema de contagem e parar ao final da análise. Ao iniciar o *OpenDataCam*, o sistema faz uso de uma classe Java desenvolvida no *Back-End* que simula o terminal do *Linux*, inserindo o comando para que o *OpenDataCam* seja iniciado.

O sistema *OpenDataCam* é carregado em uma *div iframe*, que recebe a URL do servidor *NodeJS* e exibe a tela com a interface do *OpenDataCam*. A partir desse momento se tem a opção de traçar e nomear linhas sobre o vídeo, as quais serão os pontos de contagem de veículos.

Após a identificação das linhas de contagem, a coleta dos dados pode ser iniciada. Ao final do processamento o usuário finaliza o *OpenDataCam* e os dados que foram gravados no banco de dados MongoDB v4.4.0 durante o processamento, são lidos por uma classe Java e gravados em um banco de dados MySql v8.0.22 para que sejam apresentados no *dashboard*.

Em se tratando de *hardware*, foi utilizado um *desktop* com processador Intel(R) Pentium(R) CPU G4560 @ 3.50GHz, 16GB de memória RAM DDR4 2600Hz e placa de vídeo GeForce GTX 1050 Ti 4GB, com sistema operacional Linux Ubuntu 20.04 LTS.

Para o *Back-End* foi utilizada a linguagem de programação Python v.3.6, a biblioteca *OpenCV* v.4, a rede neural *Darknet-53*, o sistema de detecção de objetos *YOLOv3*, juntamente com o peso para o reconhecimento de veículos na vista aérea ortogonal, biblioteca cuDNN v7.6.5 para *CUDA* v10.1, *NodeJS* v.12.18.3, banco de dados não relacional MongoDB v4.4.0 e banco de dados relacional MySQL v8.0.22 e o pacote *javascript OpenDataCam* v3.0.1 (*ODC*).

Após a instalação de todas a ferramentas o arquivo *makefile* da *Darknet* foi configurado para possibilitar a utilização da *GPU*, da plataforma de computação paralela *CUDA* e da biblioteca *OpenCV*, na sequência foram copiados para dentro da pasta *Darknet* o sistema de detecção *YOLOv3* e o respectivo peso da aplicação. Após este processo foi realizada a configuração básica do ODC, que consistiu em alterar o arquivo config.json, especificando a *URL* do banco de dados *MongoDB* e o caminho de instalação da rede neural *Darknet*.

Posteriormente foi compilado o pacote *ODC* com o gerenciador de pacotes do *NodeJS* (*NPM*), para que desta forma pudesse ser utilizada a linguagem de programação *Javascript* no lado servidor.

O *Front-End* e o *Back-End* são integrados no momento em que uma classe Java executa o pacote do *NodeJS*, conforme abordado anteriormente. Esta mesma classe é executada quando é interrompida a análise.

Os vídeos utilizados para testes, foram gravados com durações menores do que dez minutos com auxílio de um *drone DJI Spark* posicionado a uma altura média de 95 metros acima de interseções de ruas na cidade de Joinville-SC. Os mesmos foram carregados para o servidor por meio da interface de *upload* do sistema.

Os pesos foram obtidos no repositório *Darknet* no *GitHub* do usuário Jekhor (Yauhen Kharuzhy) [16]. Considerando que o foco do trabalho consiste no reconhecimento de imagens e não no treinamento de pesos, a utilização dos pesos já treinados com imagens ortogonais aéreas, foi suficiente para detectar os veículos das classes car, truck e bus.

A metodologia para obter o percentual de acertos consistiu em contar os veículos dos vídeos de forma manual e posteriormente comparar com o número de detecções obtido na análise do *OpenDataCam*. Feito isso foram analisados os totais de cada método de contagem e comparados os resultados, que podem ser vistos na tabela, onde foi identificada a incidência de falsos positivos e falsos negativos.

O limite de confiança padrão (*threshold*) do *OpenDataCam* é determinado em 20%. Somente após realizado o cálculo para obter o percentual de acertos identificou-se que este parâmetro poderia ser customizado para resultar em detecções mais precisas. A título de testes decidiu-se aplicar um *threshold* de 85%, e constatou-se que as detecções aparentemente se tornaram mais precisas, eliminando detecções que anteriormente eram falsos positivos ou negativos.

5 RESULTADOS E DISCUSSÕES

O sistema possui autenticação, cadastro de usuários, *dashboards*, exportação de relatórios e *upload* de vídeos para o servidor.

Ao acessar o sistema, o usuário será imediatamente direcionado para o *dashboard*, onde será possível verificar quatro gráficos com informações das análises realizadas. A Fig. 3 apresenta uma visão geral do *Dashboard* com todos os gráficos disponíveis.

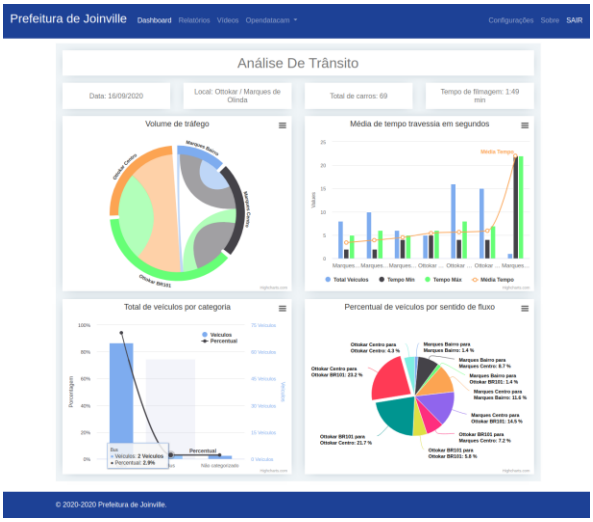

Figura 3: Página inicial do sistema

A Fig. 4, apresenta um diagrama de cordas onde é possível ver o volume de tráfego da análise realizada com as imagens aéreas do cruzamento entre as ruas Ottokar Doerffel e Marquês de Olinda, sendo assim, é possível perceber que o maior fluxo de veículos ocorreu na rua Ottokar Doerffel no sentido centro para a BR 101 onde um total de 16 veículos foram identificados.

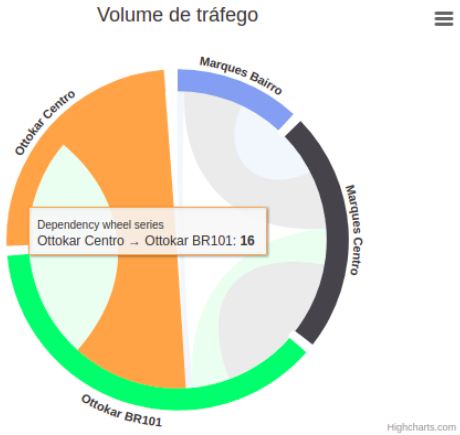

Figura 4: Diagrama de cordas relativo ao trânsito do cruzamento entre a Av. Marquês de Olinda e Rua Ottokar Doerffel

A Fig. 5 apresenta o gráfico do tempo médio de travessia dos veículos entre 2 pontos de contagem demarcados no vídeo analisado, bem como o tempo mínimo e máximo e o total de veículos contados em cada sentido do cruzamento.

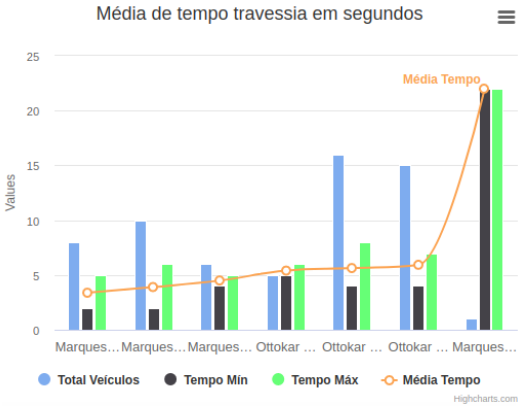

Figura 5: Gráfico de barras verticais com as médias de tempo de travessia de veículos entre as linhas marcadas no cruzamento

Na Fig. 6 foi exposto um gráfico de veículos por categoria. No destaque pode-se ver a categoria ônibus com apenas 2,9% representando 2 veículos do total de 69 veículos reconhecidos no respectivo vídeo.

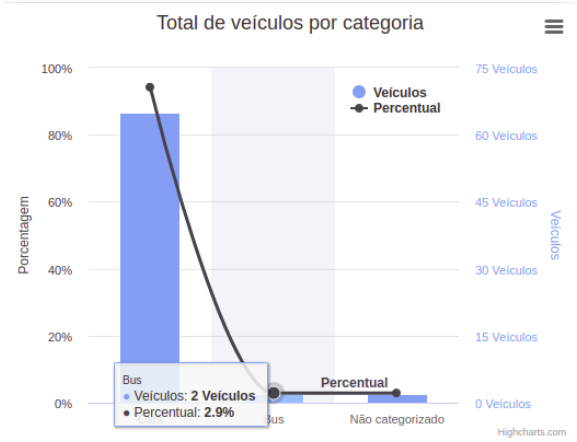

Figura 6: Categorias de veículos identificadas

Na Fig. 7, é apresentado o percentual de veículos por fluxo. Em destaque o sentido “Ottokar Centro” para “Ottokar BR 101" com 23,2% do tráfego analisado no respectivo vídeo.

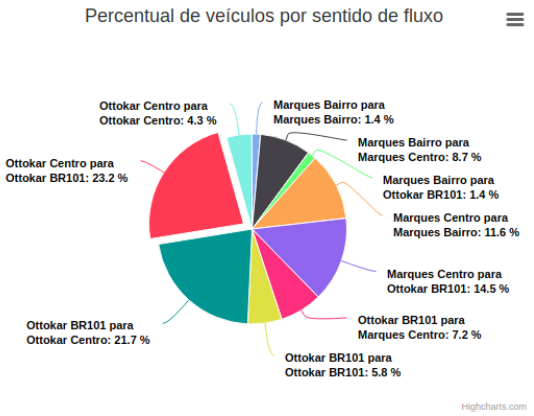

Figura 7: Percentual de veículos por sentido de fluxo

O sistema conta com telas para *upload* e pesquisa de vídeos. Na modal que dá início a análise, pode ser escolhida a resolução na qual o vídeo será processado e o limite de confiança do modelo de detecção, representado por um *slider* que vai de 1 a 99%. Esse último, sendo responsável por excluir da análise qualquer detecção de veículos com percentual de confiança abaixo do limite selecionado.

Na Fig. 8 podem ser observadas as linhas demarcadas na extremidade de cada rua, as quais são necessárias para iniciar a análise. Toda vez que um veículo passar por uma destas linhas ele será contabilizado e a quantidade será apresentada sobre cada linha.

Figura 8: Detalhe da delimitação de linhas para a contagem de veículos

Durante o processo no *OpenDataCam*, para fins de rastreamento, a rede neural analisa e compara o *frame* anterior e o subsequente para então manter o veículo com o mesmo *id*. Como pode ser visto na Fig. 9, a interligação dos pontos em cada *frame* resulta na criação das linhas de trajetória de cada veículo, representadas por cores distintas.

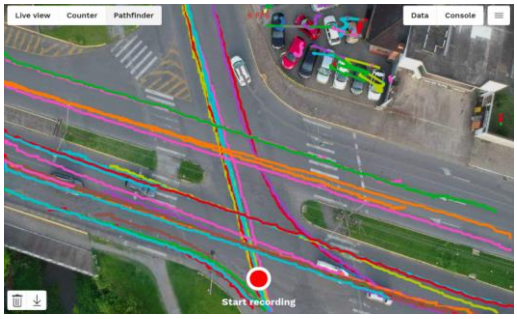

Figura 9: Detalhe do rastreamento de veículos

Após a análise o usuário tem a opção de avaliar os resultados dos vídeos analisados anteriormente e extrair os relatórios de forma rápida.

Na Tabela 1 são apresentados os resultados das análises feitas com o sistema *OpenDataCam*. Foram usadas 3 amostras de filmagens aéreas, sendo as amostras 01 e 03 capturadas por *drone* fornecido pela Faculdade Senai Joinville-SC e a amostra 02, baseada em um vídeo capturado por um *drone* do departamento de mobilidade urbana da Prefeitura de Joinville-SC.

Conforme descrito nos métodos, o percentual de acertos considerando-se as 3 amostras, foi de 102,03%. Esse resultado atípico deve-se à identificação de falsos positivos pelo *YOLOv3* durante o processamento dos vídeos. Quando analisados os percentuais de acertos por categoria, o padrão identificado geral se repete, inclusive com a ocorrência de casos onde o sistema registrou a contagem de veículos mas não conseguiu categorizá-los nas classes às quais ele está treinado para reconhecer.

Tabela 1: Percentual de reconhecimento de veículos

Categorias	Contagem Humana	OpenDataCam	Acuracidade
Amostra 1	**35**	**36**	**102,9%**
car	35	36	102,9%
truck	0	0	0,0%
bus	0	0	0,0%
não categorizado	0	0	0,0%
Amostra 2	**70**	**70**	**100,0%**
car	67	63	94,0%
truck	1	0	0,0%
bus	2	2	100,0%
não categorizado	0	5	0,0%
Amostra 3	**94**	**97**	**103,2%**
car	92	96	104,3%
truck	2	0	0,0%
bus	0	0	0,0%
não categorizado	0	1	0,0%

6 CONSIDERAÇÕES FINAIS

O aspecto urbano e as classes de veículos podem mudar significativamente de acordo com a região analisada, por exemplo, no sudeste asiático poderiam ser necessários treinamentos de redes neurais com pesos considerando a classe de veículo auto-riquixá, conhecido localmente como bajaj ou tuk-tuk, veículo incomum na maioria das regiões do planeta.

Os pesos utilizados no modelo de detecção aplicado neste trabalho não reconhecem motocicletas, pois as imagens manipuladas no treinamento dos respectivos pesos não eram de uma região com grande incidência desse tipo de veículo.

Para melhorar a taxa de precisão na detecção de veículos e a detecção de um maior número de classes, como por exemplo, a classe motocicleta, sugere-se treinar uma rede neural alimentada com imagens capturadas na região de Joinville-SC, onde futuramente serão originados os vídeos para as análises com a ferramenta.

O tempo de voo de um *drone* atualmente ainda é bastante limitado, em média de 10 a 25 minutos por carga de bateria. Do tempo total de voo, ainda deve-se levar em conta o tempo necessário para posicionar o *drone* no ponto desejado. Quanto maior for o tempo de voo, maior será a amostragem de dados capturados e melhores serão os resultados das análises como, por exemplo, a possibilidade de capturar uma maior quantidade de veículos por vídeo durante os principais horários de tráfego intenso.

Como proposta futura pode ser criado um pequeno dispositivo com o *hardware* adequado, acoplado a um poste, permitindo assim a visualização do tráfego de veículos em um ângulo de 45º. Sendo assim poderiam ser extraídas imagens de determinados períodos do dia, não sendo necessário uma pessoa para operar o *drone* em várias ocasiões no mesmo local.

O recurso de análise de imagens no âmbito municipal poderia ser melhor aproveitado aplicando-o em semáforos, tornando-os inteligentes. Utilizar estas análises somente para estatísticas pontuais seria uma forma menos eficaz de aproveitar os recursos oferecidos pelas tecnologias atuais de visão computacional.

Outra sugestão de implementação futura é tornar possível a busca em um mapa com *clustered markers*, o que traz duas vantagens: a visual e a organização de análises em um mesmo local em datas ou horários diferentes.

Pode-se utilizar um modelo de mistura Gaussiana para segmentação da imagem, extraindo somente as vias, excluindo, por exemplo, calçada, carros estacionados e tudo que ficasse fora dos limites desejados.

Por fim, poderiam ser criados *containers* para mais de um usuário poder acessar o sistema ao mesmo tempo.

7 REFERÊNCIAS

[1] Associação Nacional dos Detrans. Brasil já tem 1 carro a cada 4 habitantes. 2013. http://www.and.org.br/brasil-ja-tem-1-carro-a-cada-4-habitantes-diz-denatran/Acessado em: 16/11/2020.

[2] Move-Lab, 2020. OpenDataCam. Acessado em 23/06/2020 de https://www.move-lab.com/projects/opendatacam.

[3] REDMON, Joseph et al. You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. p. 779-788.

[4] REDMON, Joseph; FARHADI, Ali. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018.

[5] RYOO, Shane et al. Optimization principles and application performance evaluation of a multithreaded *GPU* using *CUDA*. In: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. 2008. p. 73-82

[6] GOODFELLOW, Ian et al. Deep learning. Cambridge: MIT press, 2016.

[7] REDMON, Joseph; FARHADI, Ali. YOLO9000: better, faster, stronger. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2017. p. 7263-7271.

[8] KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. Communications of the ACM, v. 60, n. 6, p. 84-90, 2017.

[9] LIN, Tsung-Yi et al. Microsoft coco: Common objects in context. In: European conference on computer vision. Springer, Cham, 2014. p. 740-755.

[10] ZHAO, Zhong-Qiu et al. Object detection with deep learning: A review. IEEE transactions on neural networks and learning systems, v. 30, n. 11, p. 3212-3232, 2019.

[11] PATTERSON, Josh; GIBSON, Adam. Deep learning: A practitioner's approach. " O'Reilly Media, Inc.", 2017.

[12] BRADSKI, Gary; KAEHLER, Adrian. Learning *OpenCV*: Computer vision with the OpenCV library. " O'Reilly Media, Inc.", 2008.

[13] WHITEHEAD, Nathan; FIT-FLOREA, Alex. Precision & performance: Floating point and IEEE 754 compliance for NVIDIA GPUs. rn (A+ B), v. 21, n. 1, p. 18749-19424, 2011.

[14] YANG, Jianxiu; XIE, Xuemei; YANG, Wenzhe. Effective Contexts for UAV Vehicle Detection. IEEE Access, v. 7, p. 85042-85054, 2019.

[15] ZHU, Pengfei et al. Vision meets drones: A challenge. arXiv preprint arXiv:1804.07437, 2018.

[16] Yauhen Kharuzhy (Jekhor), 2018. repositório Darknet. Acessado em 04/05/2020 de https://github.com/jekhor/darknet.