TF - Detecção de placas de trânsito

Leonardo Becker de Oliveira
GRR20211779

I. Introdução

Os estudos naturalísticos de direção brasileiro, são pesquisas relacionadas a área de transportes, mobilidade urbana e segurança viária, cujo um dos objetivos é a extração de características comportamentais do condutor, enquanto conduz o veículo.

O principal objetivo do seguinte trabalho foi a detecção através da visão computacional das placas de pare e de sinalização da velocidade da via junto de suas respectivas coordenadas geográficas.

Link para repositório contendo o trabalho no github:
https://github.com/LeonardooBecker/TF-VisaoComputacional

II. Motivação

A escolha desses dois tipos de placas foi solicitada pelo professor Jorge Tiago Bastos - departamento de transportes. O intuito principal é transformar um trabalho feito ”manualmente”, onde pesquisadores eram responsáveis por visualizar os vídeos coletados pela própria pesquisa, e anotar os pontos em que havia esses tipos de placas, passando a ser algo feito pela máquina através de conceitos relacionados a visão computacional.

III. Proposta inicial

Primeiramente, foi tentando encontrar datasets para o treinamento da rede de visão computacional que estivessem disponibilizados pela internet. Foi possível achar um dataset[1] que serviu de base para o início do treinamento. O dataset conta com imagens diversas de placas de trânsito brasileiro, mas infelizmente as placas de pare não estavam presentes nele enquanto as placas de velocidade da via estavam todas em uma mesma classificação.

Observando a maneira em que as imagens presentes no dataset foram geradas, foi seguido o mesmo modelo para criar alguns exemplos de placas de pare. As imagens foram compostas utilizando a ferramenta Canvas devido a sua vasta quantidade de background disponíveis, junto de imagens com fundo transparente da placa de ”pare”. Figura [1]

Tendo as imagens geradas, com auxílio da ferramenta online *Make Sense*, foi feito a rotulagem individual de cada uma delas (de acordo com a arquitetura YOLO), obtendo no final um novo dataset composto só por imagens de pare, do qual foi concatenado ao encontrado.

Figura 1. Placa de pare gerada com Canvas

IV. Treinamentos

Para a obtenção dos pesos utilizados na visão computacional, foi empregue a arquitetura YOLO devido a sua eficiência e capacidade de detectar múltiplos objetos em uma mesma imagem, além da facilidade para trabalhar e gerar novas redes.

Foram utilizados então dois principais arquivos dentre disponibilizados no github da ultralytics [2], são eles:

“train.py” - Programa desenvolvido em python responsável pelo fornecimento de pesos utilizado na visão computacional.

“detect.py” - Programa desenvolvido em python que utiliza dos pesos fornecidos anteriormente, para a detecção de objetos em imagens.

A seguir será apresentado a ordem de execução das ideias, para conseguir chegar em um resultado satisfatório. Vale ressaltar que a quantidade de épocas e baterias utilizadas foram escolhidas de forma que o resultado final fosse no mínimo 90% de acertos. Além disso, foram realizados mais de uma chamada do programa dentro de cada treinamento, mas todos seguiram o mesmo padrão especificado em cada parte.

A. Treinamentos 1

O primeiro treinamento utilizado foi feito com base na concatenação do Dataset [1] e o gerado pelo Canvas. Os pesos obtidos foram submetidos alguns vídeos gerados pela plataforma de coleta para a detecção, do qual apresentou resultados satisfatórios para a identificação de placas em geral, porém, com baixa assertividade e pouca generalização.

Sabendo disso, foi desenvolvido um programa em Python [3], cujo objetivo é utilizar dos pesos iniciais para a detecção e registro das imagens realizando também o preenchimento dos arquivos de label conforme arquitetura YOLO.

A partir disso, foi tomado outro trabalho manual – a correção da classificação dos arquivos de label. Então, para

cada nova imagem gerada pelo programa em Python, foi necessário a verificação visual se estava realmente de acordo.

B. Treinamentos 2

Sabendo da boa taxa de detecção de placas, o próximo objetivo foi a separação das placas de velocidade, ou seja, foram descartadas as classes iniciais e geradas 12 novas classes, sendo elas: Limite de velocidade 20 km/h até Limite de velocidade 100 km/h (totalizando 9, do qual somente as 6 iniciais foram realmente utilizadas), placa de preferencial, placa de pare e placa de estacionamento. Com isso, foi necessário realizar novamente o trabalho manual, mas dessa vez separando as imagens para esse novo conjunto de classes.

Além da classificação de imagens dos vídeos, foi gerado um novo Dataset que contém as placas de velocidade, visando abranger certos limites que não estão presentes nos vídeos. Figura [2]

Figura 2. Placa limite de velocidade gerada com Canvas

Realizado o treinamento, os pesos obtidos tiveram a característica de detectar placas com facilidade, mas ainda não estava sendo assertiva sobre o limite de velocidade correto.

C. Treinamentos 3

A fim de tentar otimizar o processo de geração do dataset e visando abranger todas as variações de placas, foi realizado a alteração do programa "detect.py" disponibilizado pela YOLO V5, do qual ao abrir a webcam, todas as imagens (com detecção de placa) são salvas em uma pasta, junto do seu respectivo rótulo completo. As simulações de placa foram geradas utilizando exemplares aberto no celular, explorando diferentes angulações e distâncias.

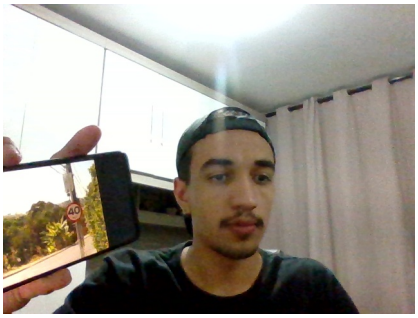

Figura 3. Simulação de placas

Com isso, inúmeras imagens dos diferentes tipos de placas foram geradas, do qual foi atribuído novamente a arquitetura YOLO, a fim de obter uma nova valoração de pesos.

Os novos pesos apresentaram resultados excelentes na detecção de placa, mas deixou a desejar quando imposta aos vídeos gerados pela plataforma de coleta. Provavelmente isso se deu pelo cenário e condições de como o dataset foi obtido, dado que o fundo era sempre o mesmo, e a imagem exemplar da placa não possuía alterações além do ângulo do celular.

D. Treinamentos 4

Por fim, por mais que os pesos obtidos anteriormente estivesse falho ao detectar placas quando em contato com os vídeos da plataforma, ainda havia a identificação em alguns frames, do qual as imagens geradas foram utilizadas a fim de fornecer variação de ambiente para o dataset.

Também se fez necessário para facilitar o processo, o desenvolvimento de um novo programa em Python [4], cujo objetivo é corrigir as classes presentes em cada arquivo de label. Sua única função é alterar o campo referente a classe nos arquivos de rótulo, para ficar de acordo com a imagem real.

Esse último processo foi repetido algumas vezes a fim de popular o Dataset. Resultado final em figura 4.

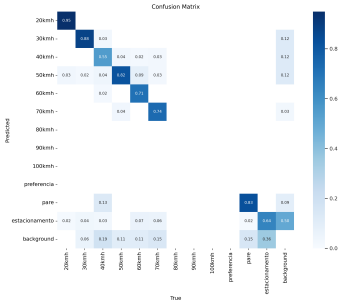

Figura 4. Resultado final

O último Dataset treinado foi composto por 1833 imagens de treinamento e 393 imagens de validação. Resultados em figura [2]

V. Identificação de coordenadas geográficas

Para obter as coordenadas dos frames onde houve a detecção de algum tipo de placa, foi utilizado o *paddleOCR*, disponível no Github theos-ai [5]. O *paddleOCR* é baseado na arquitetura YOLO pode ser obtido facilmente através da biblioteca Python *"easy_paddle_ocr"*, do qual dado uma imagem, ele retorna o texto contido nela junto de uma taxa de confiança. A identificação do texto não foi explorada tão a fundo, portanto, foi trabalhado somente em cima dos resultados fornecidos pela biblioteca para registrar as coordenadas.

VI. Conclusão

Com a elaboração e estruturação do trabalho descrito, foi possível gerar um Dataset de grande utilidade para a área de transportes urbanos, além disso, com a modificação e criação

dos programas descritos ao longo do texto, foi e é possível alimentar o Dataset cada vez mais, do qual o tornaria cada vez mais preciso.

Além disso, caberia a fim de testes explorar outras variedades de placa, de forma que diminuísse as confusões existentes, como por exemplo a inserção das placas de estacionamento, em que já estão inseridas com o objetivo de reduzir o erro quando escolha das placas realmente desejadas.

Por fim, para concluir o objetivo incialmente proposto de realizar o processo completo de detecção e registro das coordenadas, seria necessário elaborar alguns outros programas responsáveis por tratar e apresentar as informações obtidas pela detecção dos textos nos frames capturados, porém, com a alta demanda de tempo para realizar os treinamentos de pesos e conferência/correção das imagens obtidas, acabou se tornando inviável a última parte para domínio deste trabalho.

VII. Referências

[1]https://universe.roboflow.com/yolo-uiruj/brazilian-traffic-signs

[2]https://github.com/ultralytics/yolov5

[3]https://github.com/LeonardooBecker/TF-VisaoComputacional/blob/main/geraDataset.py

[4]https://github.com/LeonardooBecker/TF-VisaoComputacional/blob/main/arrumaClasse.py

[5]https://github.com/theos-ai/easy-paddle-ocr