



UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
ITI UFSCAR - PROGRAMA DE ENSINO, PESQUISA E EXTENSÃO DA UFSCAR
CURSO DE PÓS GRADUAÇÃO LATO SENSU
MBA EM MACHINE LEARNING IN PRODUCTION

Gustavo Abbomerato Zantut

Detecção e leitura de placas em tempo real

São Carlos/SP
2023

Gustavo Abbomerato Zantut

Detecção e leitura de placas em tempo real

Trabalho de conclusão de curso apresentado em cumprimento à exigência curricular do curso Pós-Graduação Lato Sensu MBA em Machine Learning in Production pela Universidade Federal de São Carlos.

Orientadora: Profª. Dra. Marilde Terezinha Prado Santos

São Carlos/SP
2023

Ficha de identificação da obra

A ficha de identificação é elaborada pelo próprio autor
Orientações em:

<https://www.sibi.ufscar.br/servicos/gerador-de-ficha-catalografica>

Gustavo Abbomerato Zantut

Detecção e leitura de placas em tempo real

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de “Especialista em MBA em Machine Learning in Production” e aprovado em sua forma final pelo Curso Pós-Graduação Lato Sensu MBA em Machine Learning in Production.

São Carlos/SP, 18 de Novembro de 2023.

Banca Examinadora:

Prof^a. Dra. Marilde Terezinha Prado Santos
Instituição UFSCAR

Prof. Segundo, Dr.
Instituição UFSCAR

Prof. Terceiro, Dr.
Instituição UFSCAR

RESUMO

Esse trabalho visa realizar a detecção de placa de veículos em tempo real por meio de tecnologias como yolo, openalpr, kafka, mage ai, docker, micro serviços, gstreamer e outras ferramentas que apoiam esse objetivo. Através de um input de imagem(vídeo /foto, stream em tempo real) o sistema é capaz de detectar a placa e identificar os dígitos presentes, bem como anotar esses dígitos na imagem e enviar as informações até uma plataforma em nuvem através de serviços de mensageria, rodando em plataformas Windows e/ou arm64(Raspberry Pi 4).

Palavras-chave: alpr; anpr; kafka kraft; kafka; arduino kafka; raspberry; docker; image detection; gstreamer; docker desktop; mageai ai; yolo; yolov5; openalpr; leitor de placa; detecção de placa.

ABSTRACT

This work aims to perform vehicle license plate detection in real time using technologies such as yolo, openalpr, kafka, mage ai, docker, microservices, gstreamer and other tools that support this objective. Through an image input (video /photo, real-time stream) the system is capable of detecting the license plate and identifying the characters present, as well as annotating the image and sending the information to a cloud platform through messaging services and running on Windows and/or arm64(Raspberry Pi 4) platforms.

Keywords: alpr; anpr; kafka kraft; kafka; arduino kafka; raspberry; docker; image detection; gstreamer; docker desktop; magea ai; yolo; yolov5; openalpr; plate recognition; plate reader.

LISTA DE FIGURAS

Figura 1 – Comparação de tamanho e latência vs. desempenho do Modelo no conjunto de dados COCO mAP 50-95.	20
Figura 2 – Comparação de latência vs. desempenho do Modelo no conjunto de dados COCO mAP 50-95.	21
Figura 3 – Matriz de confusão.	22
Figura 4 – Curva F1.	22
Figura 5 – Curva R.	23
Figura 6 – Curva PR.	23
Figura 7 – Resultados finais de treinamento e teste.	24
Figura 8 – Distribuição das categorias.	25
Figura 9 – Diagrama da Arquitetura.	27
Figura 10 – Print da tela inicial do json server.	30
Figura 11 – Print da tela de desenvolvimento do Pipeline na ferramenta MageAI. .	31
Figura 12 – Padrão de nomenclatura dos <i>frames</i> após renomeados.	31
Figura 13 – Exemplo de resultado em log.	33
Figura 14 – Exemplo de resultados exibidos via json server.	34
Figura 15 – Exemplo de resultados exibidos via recebimento de mensagens Kafka em tela lcd.	34
Figura 16 – Exemplo de resultados sendo exibidos no frame.	35
Figura 17 – Resultados disponíveis via BigQuery.	35

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

alpr	Automatic License Plate Recognition - Localiza placas na imagem (Yolov5 (JOCHER, 2020)).
anpr	Automatic Number Plate Recognition - Identifica os digitos da placa (openalpr (HILL, 2020)).
frame renamer	Renomeia os frames para serem exibidos como video via gstreamer.
I/O	<i>Input/Output</i> - Entrada e saída de dados
json server	Servidor JSON para armazenamento local das placas identificadas baseado na biblioteca json-server (NPM, 2015).
png streamer	Reproduz png's como video via gstreamer (WALTHINSEN, 2001).
pred displayer	Mostra o frame das detecções em tempo real via gstreamer (WALTHINSEN, 2001).
pred etl	Pipeline de dados do com Mage AI (DANG, 2021) que pega as placas do Kafka e envia para o BigQuery.
pred poster	Prediction Poster - Posta as placas identificadas no json server e/ou Kafka e as carimba nos respectivos frames.

LISTA DE SÍMBOLOS

SUMÁRIO

1	INTRODUÇÃO	13
1.1	RECOMENDAÇÕES DE USO	14
1.1.1	Configuração do Ambiente de Implantação	14
1.1.2	Configurações do Sistema	14
1.1.2.1	<i>Docker-Compose</i>	14
1.1.2.2	<i>Serviços Específicos</i>	15
1.1.2.2.1	alpr (Automatic License Plate Recognition - Localiza placas na imagem (Yolov5 (JOCHER, 2020)).)	15
1.1.2.2.2	anpr (Automatic Number Plate Recognition - Identifica os digitos da placa (openalpr (HILL, 2020)).)	15
1.1.2.2.3	pred poster (Prediction Poster - Posta as placas identificadas no json server e/ou Kafka e as carimba nos respectivos frames.)	16
1.1.2.2.4	json server (Servidor JSON para armazenamento local das placas identificadas baseado na biblioteca json-server (NPM, 2015).)	16
1.1.2.2.5	pred display (Mostra o frame das detecções em tempo real via gstreamer (WALTHINSEN, 2001).)	16
1.1.2.2.6	pred etl (Pipeline de dados do com Mage AI (DANG, 2021) que pega as placas do Kafka e envia para o BigQuery.)	16
1.1.2.2.7	frame renamer (Renomeia os frames para serem exibidos como video via gstreamer.)	17
1.1.2.2.8	png streamer (Reproduz png's como video via gstreamer (WALTHINSEN, 2001).)	17
1.1.2.3	Atualizações e Manutenção	17
1.1.2.4	Backup e Recuperação de Dados	17
1.1.2.5	Integração com Outros Sistemas	18
1.2	OBJETIVOS	18
1.2.1	Objetivo Geral	18
1.2.2	Objetivos Específicos	18
2	DESENVOLVIMENTO	20
2.1	SELEÇÃO DE TECNOLOGIAS	20
2.1.1	Detecção de Placas	20
2.1.2	Leitura de Placas	21
2.2	CONFECÇÃO DO CONJUNTO DE DADOS	24
2.3	DESENHO DA ARQUITETURA	25
2.3.1	Ambiente	25
2.3.2	Serviços	26
2.3.3	Comunicação entre serviços	27

2.3.4	Diagrama da Arquitetura	27
2.4	DESENVOLVIMENTO E PERSONALIZAÇÃO DOS SERVIÇOS	28
2.4.1	alpr	28
2.4.2	anpr	28
2.4.3	pred poster	29
2.4.4	json server	30
2.4.5	pred display	30
2.4.6	pred etl	30
2.4.7	frame renamer	30
2.4.8	png streamer	32
2.5	DISPONIBILIZAÇÃO DOS RESULTADOS	32
3	UTILIZAÇÃO	36
3.1	ESTRUTURA DOS DIRETÓRIOS	36
3.1.1	Diretórios de I/O	36
3.1.2	Diretórios de Configuração	37
3.1.3	Diretórios de Conjunto de dados	37
3.2	DESEMPENHO	38
3.2.1	Configurações de Hardware	38
3.2.1.1	Computador de Mesa	38
3.2.1.2	Raspberry PI 4	38
3.2.2	alpr	38
3.2.2.1	Computador de Mesa	38
3.2.2.2	Raspberry PI 4	38
3.2.3	anpr	39
3.2.3.1	Computador de Mesa	39
3.2.3.2	Raspberry PI 4	39
3.3	PRÓXIMOS PASSOS	39
3.3.1	Otimização de Imagens	39
3.3.2	Desacoplamento e Dependências	39
3.3.3	Aferição de Resultados	39
3.3.4	Treinamento	40
3.4	CÓDIGO FONTE	40
4	CONCLUSÃO	41
	REFERÊNCIAS	42

1 INTRODUÇÃO

A detecção de placas veiculares desempenha um papel cada vez mais crucial nos sistemas modernos de segurança, monitoramento de tráfego e gestão de áreas urbanas. Este projeto propõe-se a desenvolver um sistema flexível, eficiente e eficaz para a detecção de placas em tempo real, que vai além das limitações das soluções tradicionais, integrando-se de maneira harmoniosa em ambientes diversos, uma flexibilidade que o Docker com uma arquitetura baseada em micro serviços consegue fornecer com maestria.

Em um cenário onde a maioria das soluções de leitura de placas são comercializadas e carecem de código aberto, a abordagem adotada para este projeto difere significativamente. Optamos por não iniciar a construção do sistema a partir do zero; (SAGAN, 2011, Se voce quer fazer uma torta de maçã do zero, primeiro você deve inventar o universo). Nesse contexto, a escolha mais pragmática e com maior chance de sucesso foi aprimorar e modificar bibliotecas já existentes.

Essa abordagem não apenas otimiza recursos, mas também permite uma rápida implementação, aproveitando o trabalho preexistente de comunidades de desenvolvedores. Ao construir sobre bases sólidas de bibliotecas estabelecidas, podemos concentrar nossos esforços na personalização e melhoria de funcionalidades específicas, atendendo assim às demandas específicas de um sistema de detecção de placas robusto e adaptável.

Este documento descreverá detalhadamente a metodologia adotada, as tecnologias integradas e os resultados alcançados, enfatizando a inovação que ocorre ao aprimorar soluções existentes. O projeto não apenas visa fornecer uma solução prática para a detecção de placas veiculares, mas também a contribuir para a comunidade de desenvolvedores ao oferecer uma alternativa viável e acessível em um cenário muitas vezes dominado por soluções proprietárias.

Os serviços que compõe o sistema são:

- a) alpr (Automatic License Plate Recognition - Localiza placas na imagem (Yolov5 (JOCHER, 2020)).);
- b) anpr (Automatic Number Plate Recognition - Identifica os dígitos da placa (openalpr (HILL, 2020)).);
- c) pred poster (Prediction Poster - Posta as placas identificadas no json server e/ou Kafka e as carimba nos respectivos frames.);
- d) json server (Servidor JSON para armazenamento local das placas identificadas baseado na biblioteca json-server (NPM, 2015).);
- e) pred display (Mostra o frame das detecções em tempo real via gstreamer (WALTHINSEN, 2001).);
- f) pred etl (Pipeline de dados do com Mage AI (DANG, 2021) que pega as placas do Kafka e envia para o BigQuery.);

- g) frame renamer (Renomeia os frames para serem exibidos como video via gstreamer.);
- h) png streamer (Reproduz png's como video via gstreamer (WALTHINSEN, 2001).).

1.1 RECOMENDAÇÕES DE USO

Abaixo listaremos as melhores práticas e recomendações para uma implementação de sucesso do sistema de detecção de placas.

1.1.1 Configuração do Ambiente de Implantação

Para um melhor desempenho e acurácia do sistema, é fortemente recomendado que a captura das imagens seja realizada em um ambiente bem iluminado. Além disso, é aconselhável que as imagens sejam capturadas de forma que a placa esteja o menos rotacionada possível. Para atingir os melhores resultados, é essencial que o equipamento de captura seja capaz de fornecer imagens nítidas dos veículos em movimento.

A qualidade deste equipamento está diretamente relacionada à distância e/ou velocidade com que os veículos estão se movimentando. Recomenda-se, portanto, que o equipamento seja escolhido levando em consideração esses fatores, garantindo assim a obtenção de imagens de alta qualidade que contribuirão significativamente para o desempenho geral do sistema.

Também, para o pred poster funcionar corretamente, é necessário um cluster Kafka disponível no ambiente, caso contrário, o código deve ser alterado para não buscar o Kafka.

Vale ressaltar que os serviços de identificação de leitura de placas são custosos computacionalmente, logo, caso esteja rodando o sistema em plataformas com recursos escassos, recomenda-se deixar o mínimo de serviços possíveis rodando em conjunto e para a melhor comunicação entre os serviços em hosts diferentes, a capacidade da transmissão de dados entre eles impacta diretamente a latência de comunicação dos serviços.

1.1.2 Configurações do Sistema

O sistema foi desenvolvido com flexibilidade em mente, facilitando a configuração em diferentes ambientes. Para uma implementação eficaz, destacamos as principais configurações relacionadas aos serviços principais do projeto.

1.1.2.1 Docker-Compose

O projeto inclui arquivos de docker-compose prontos para uso em ambientes Windows e ARM64. Esses arquivos vêm com configurações iniciais funcionais, exigindo apenas que os volumes sejam fornecidos. Certifique-se de incluir o modelo do Yolov5 treinado no

diretório apropriado do `alpr` (Automatic License Plate Recognition - Localiza placas na imagem (Yolov5 (JOCHER, 2020)).) para garantir o funcionamento correto.

1.1.2.2 Serviços Específicos

Cada serviço no sistema, por ser construído com base em bibliotecas existentes, oferece uma variedade de configurações que podem ser ajustadas para atender às necessidades específicas do projeto. Abaixo estão listadas as principais configurações para cada serviço:

1.1.2.2.1 `alpr` (Automatic License Plate Recognition - Localiza placas na imagem (Yolov5 (JOCHER, 2020)).)

- a) **source**: Especifica a origem do vídeo ou imagens a serem processados.
- b) **weights**: Define o caminho para os pesos pré-treinados do modelo Yolov5.
- c) **save_crop**: Habilita ou desabilita a opção de salvar as regiões detectadas, como caixas delimitadoras, em arquivos de imagem separados.
- d) **conf_thres**: Configura o limiar de confiança para a detecção de objetos. Um valor mais baixo pode resultar em mais detecções, mas com menor confiança.
- e) **max_det**: Especifica o número máximo de detecções permitidas por imagem.
- f) **save_frame (criada para otimizar I/O (Input/Output - Entrada e saída de dados) e permitir que os frames do vídeo sejam salvos em png para stream)**: Habilita ou desabilita a opção de salvar *frames* do vídeo.
- g) **save_detected_frame (criada para otimizar I/O (Input/Output - Entrada e saída de dados))**: Habilita ou desabilita a opção de salvar *frames* apenas quando uma placa é detectada.
- h) **save_each_n_frames (criada para otimizar I/O (Input/Output - Entrada e saída de dados))**: Configura a frequência de salvamento dos *frames* durante o processamento.

1.1.2.2.2 `anpr` (Automatic Number Plate Recognition - Identifica os dígitos da placa (openalpr (HILL, 2020)).)

- a) **topn**: Especifica o número de máximo de placas diferentes fornecidas.
- b) **country**: Determina o conjunto de dados de treinamento usado para o reconhecimento de placas.
- c) **pattern**: Define os padrões aceitos como placa, restringindo o reconhecimento a padrões específicos de placas veiculares.

- d) **contrast_detection_threshold**: Percentual das placas com baixo contraste. Este parâmetro controla a sensibilidade do sistema para placas de veículos com baixo contraste em relação ao fundo.
 - e) **max_plate_angle_degrees**: Máxima rotação permitida da placa na imagem. Isso ajuda a lidar com variações na orientação das placas em relação à câmera.
 - f) **postprocess_min_confidence**: Confiança mínima de leitura da placa.
 - g) **postprocess_confidence_skip_level**: Dígitos com confiança inferior a este valor são rebalanceados para evitar leituras de dígitos alto nível de incerteza.
- 1.1.2.2.3 pred poster (Prediction Poster - Posta as placas identificadas no json server e/ou Kafka e as carimba nos respectivos frames.)
- a) **BOOTSTRAP_SERVERS**: Especifica a string de servidores do Kafka.
 - b) **TOPIC_NAME**: Define o nome do tópico utilizado para envio das mensagens no Kafka.
 - c) **server_url**: Indica a URL para o json server (Servidor JSON para armazenamento local das placas identificadas baseado na biblioteca json-server (NPM, 2015).) para operações de post.
- 1.1.2.2.4 json server (Servidor JSON para armazenamento local das placas identificadas baseado na biblioteca json-server (NPM, 2015).)

A única configuração necessária é um arquivo chamado "db.json" no diretório "data" com o formato dos posts enviados pelo pred poster (Prediction Poster - Posta as placas identificadas no json server e/ou Kafka e as carimba nos respectivos frames.).

- 1.1.2.2.5 pred display (Mostra o frame das detecções em tempo real via gstreamer (WALTHINSEN, 2001).)
- a) **DISPLAY**: Indica o *display* onde o GStreamer apresentará o *frame* com a placa identificada.
- 1.1.2.2.6 pred etl (Pipeline de dados do com Mage AI (DANG, 2021) que pega as placas do Kafka e envia para o BigQuery.)
- a) **table_id**: Especifica o caminho da tabela de destino no BigQuery.
 - b) **bootstrap_server**: Indica a string de servidores Kafka.
 - c) **topic**: Define o tópico utilizado para a comunicação no Kafka.

- d) **consumer_group:** Grupo de consumidores do tópico ao qual o pipeline está associado.

Para a integração efetiva com o BigQuery, é imprescindível configurar uma chave do BigQuery associada a uma conta de serviço. Certifique-se de obter e configurar corretamente essa chave para garantir o acesso e a funcionalidade adequada do pipeline. Ademais, a ferramenta Mage AI fornece diversos tipos de integração com pouco esforço de desenvolvimento, sendo extremamente personalizável.

1.1.2.2.7 frame renamer (Renomeia os frames para serem exibidos como video via gstreamer.)

Não se aplica, ele simplesmente renomeia as imagens da ultima detecção para o gstreamer entender a ordem dos *frames* e reproduzir como vídeo.

1.1.2.2.8 png streamer (Reproduz png's como video via gstreamer (WALTHINSEN, 2001).)

- a) **RUN_NAME:** Indica o nome da pasta abaixo do diretório "runs/detect" com os *frames* a serem mostrados em sequencia após renomeador por frame renamer (Renomeia os frames para serem exibidos como video via gstreamer.).

1.1.2.3 Atualizações e Manutenção

Recomenda-se regulares validações e implementações das atualização de software, principalmente o Docker, e também a verificação e implementação de novas versões de bibliotecas, frameworks e imagens utilizadas. A manutenção preventiva do ambiente é muito importante para garantir a qualidade das detecções bem como a latência da informação.

1.1.2.4 Backup e Recuperação de Dados

O sistema foi projetado com uma robusta estratégia de backup e recuperação de dados, a fim de garantir alta rastreabilidade e persistência das informações processadas. Algumas das principais características são:

- a) **Registro Detalhado:** Cada *frame* capturado pelo sistema é registrado com um carimbo de data e hora indicando o momento da inserção no sistema. Isso proporciona uma trilha de auditoria precisa para cada instância de processamento.
- b) **Numeração de Placas:** Cada placa identificada é numerada de acordo com o *frame* ao qual está associada. Essa abordagem simplifica a correlação entre as placas e os *frames* correspondentes, facilitando a recuperação de informações específicas.

- c) **Registro de Processamentos:** Todos os processamentos realizados pelo sistema são registrados de forma detalhada. Isso inclui informações sobre detecções, reconhecimentos, e quaisquer outros eventos críticos. Os registros são essenciais para garantir que a informação não se perca ou se multiplique indevidamente.
- d) **Verificação de Logs:** A integridade e consistência dos logs são verificadas regularmente para prevenir possíveis falhas ou inconsistências. Esse procedimento é crucial para garantir a precisão e confiabilidade dos dados armazenados.

Essas práticas asseguram não apenas a persistência do estado anterior do sistema, mas também possibilitam uma recuperação eficiente de dados, minimizando riscos de perda ou corrupção de informações ao longo do tempo.

1.1.2.5 Integração com Outros Sistemas

O core desse sistema é basicamente analisar uma imagem e devolver um json com as informações da placa, logo a integração com outras plataformas tende a ser pouco complexa, além de estar tudo em *containers*, através de arquivos Dockerfile e Docker compose, o código se torna uma documentação mais técnica dos requisitos, arquitetura, ambiente e configurações.

1.2 OBJETIVOS

A seguir apresentaremos os principais objetivos do projeto, detalhando cada objetivo específico que guiou o desenvolvimento e aprimoramento do projeto.

1.2.1 Objetivo Geral

O principal propósito deste projeto é desenvolver e implementar um sistema de reconhecimento de placas veiculares em tempo real, com foco na entrega de resultados precisos e na versatilidade de saídas, incluindo um pipeline completo até um ambiente em nuvem.

1.2.2 Objetivos Específicos

- a) **Reconhecimento em Tempo Real:** Cada *frame* capturado pelo sistema é registrado com um carimbo de data e hora indicando o momento da inserção no sistema. Isso proporciona uma trilha de auditoria precisa para cada instância de processamento.
- b) **Diversidade de Saídas:** Oferecer uma variedade de opções de saída para maximizar a utilidade do sistema. Isso abrange a gravação e exibição das placas identificadas nos *frames* via gstreamer, a geração de arquivos JSON estruturados, a exposição de uma API JSON para consulta externa, a integração com o Kafka

para comunicação assíncrona, e a implementação de um pipeline completo para transferência de dados até o ambiente em nuvem.

- c) **Adaptabilidade:** Projetar o sistema de forma flexível, permitindo a configuração ajustável para se adaptar a diferentes ambientes e requisitos específicos. Isso inclui a capacidade de ajustar as saídas desejadas, parâmetros de detecção e a integração harmoniosa com diferentes ambientes de captura de vídeo.
- d) **Rastreabilidade:** Implementar mecanismos para rastrear cada frame, numerar cada placa associada a estes *frame* e garantir a persistência do estado anterior. Todos os processamentos são registrados detalhadamente, ficando claro em qual etapa do processo está cada imagem.

2 DESENVOLVIMENTO

O desenvolvimento do projeto foi estruturado em cinco grandes etapas, sendo elas a seleção das tecnologias, confecção do conjunto de dados de treinamento, desenho da arquitetura, desenvolvimento e personalização dos serviços e a disponibilização dos resultados. A seguir, detalharemos cada etapa, destacando os desafios enfrentados e as decisões chave para as tomadas durante processo.

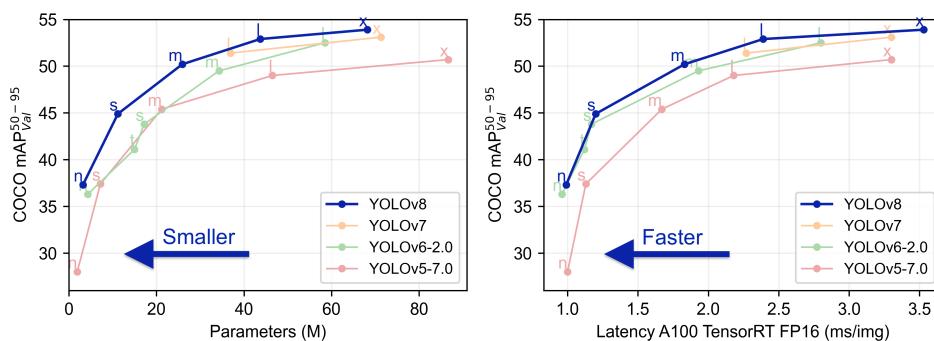
2.1 SELEÇÃO DE TECNOLOGIAS

2.1.1 Detecção de Placas

A tecnologia para a tarefa de detecção de placas foi o Yolov5 e os principais motivos foram a flexibilidade para treinamento personalizado, diversidade de inputs aceitos, disponibilidade em *containers*, alta gama de parâmetros para personalizar as detecções e também o tamanho(quantidade de parâmetros) é relativamente menor que sua versões sucessoras que embora performem melhor, conforme imagem abaixo, a detecção das placas não necessita de tanta complexidade.

Pode-se observar a partir da Figura 1 e Figura 2 que a latência dos sucessores é bem parecida, porém o tamanho dos modelos do Yolov5 são bem menores ate a versão média("m"), sendo mais performático principalmente em plataformas com recursos reduzidos.

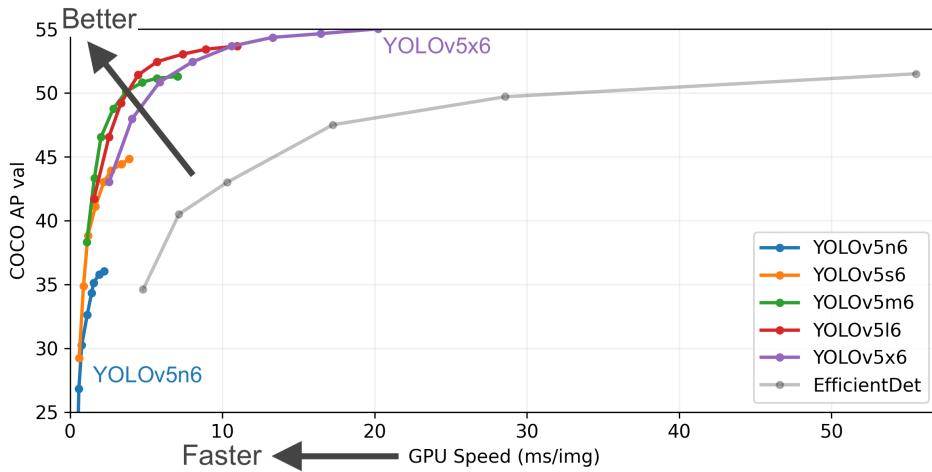
Figura 1 – Comparação de tamanho e latência vs. desempenho do Modelo no conjunto de dados COCO mAP 50-95.



Fonte: <https://github.com/ultralytics/ultralytics>

A arquitetura do Yolov5-7.0 utilizada foi a nano versão 6, (Yolov5n6) e as categorias a serem identificadas são placa carro, placa carro Mercosul, placa moto e placa moto Mercosul. Essa divisão foi necessária por conta na diferença dos padrões entre os modelos de placa. O modelo foi treinado com imagens de diversas resoluções, a que se mostrou mais eficiente e eficaz foi a de 256x256 pixels(tamanho relativamente pequeno de imagem, redu-

Figura 2 – Comparação de latência vs. desempenho do Modelo no conjunto de dados COCO mAP 50-95.



Fonte: <https://github.com/ultralytics/ultralytics>

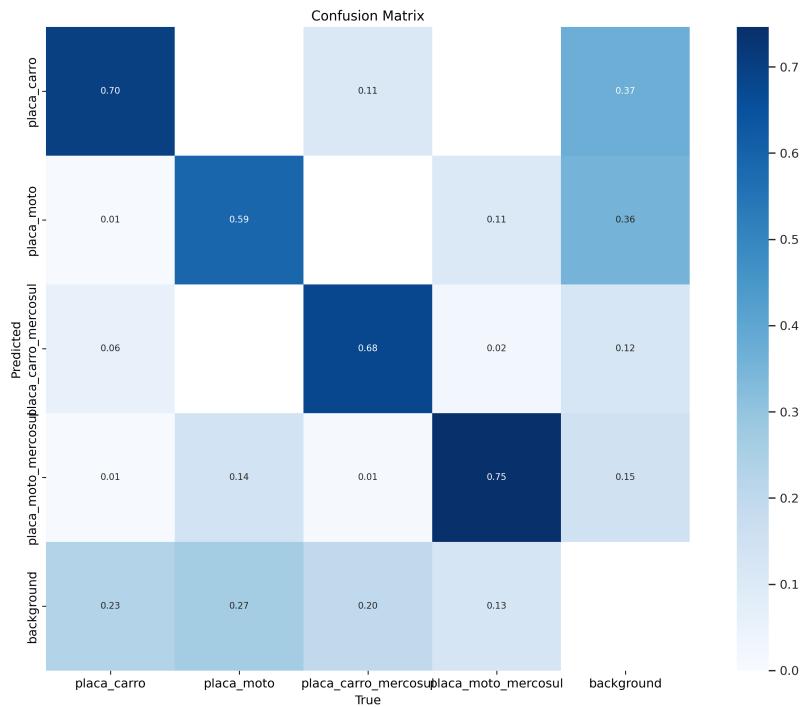
zindo tempo de I/O (*Input/Output* - Entrada e saída de dados) significativamente e sem muita perda de desempenho na detecção e leitura de placas), com 345 imagens por lote e por 150 épocas; os resultados podem ser observados na Figura 3, Figura 4, Figura 5, Figura 6 e Figura 7:

Como é de se esperar as placas de moto são as que menos foram corretamente identificadas, parte desse desempenho abaixo das demais categorias se deve à quantidade de amostras fornecidas para treino, que estão em menor quantidade que as de carros. Entretanto, apesar das placas de moto padrão Mercosul possuirem ainda menos amostras que as placas de moto no padrão antigo, performaram bem por conta de seu distinto formato e combinação de cores. O Yolo_v5 é uma biblioteca de visão computacional que realiza detecção de objetos em imagens através de redes neurais profundas e técnicas de processamento de imagem como data augmentation. No geral o modelo performou bem com os dados de treino e seu aprendizado convergiu para os dados de teste e validação, foram testados diversas arquiteturas de modelo e diversos tamanhos de imagens e o que melhor atendeu a demanda foi o menor modelo do Yolov5(nano) com imagens em baixa resolução atendeu bem a demanda, otimizando a utilização de recursos, diminuindo o tempo de inferência e ainda mantendo a qualidade de detecção e leitura das placas.

2.1.2 Leitura de Placas

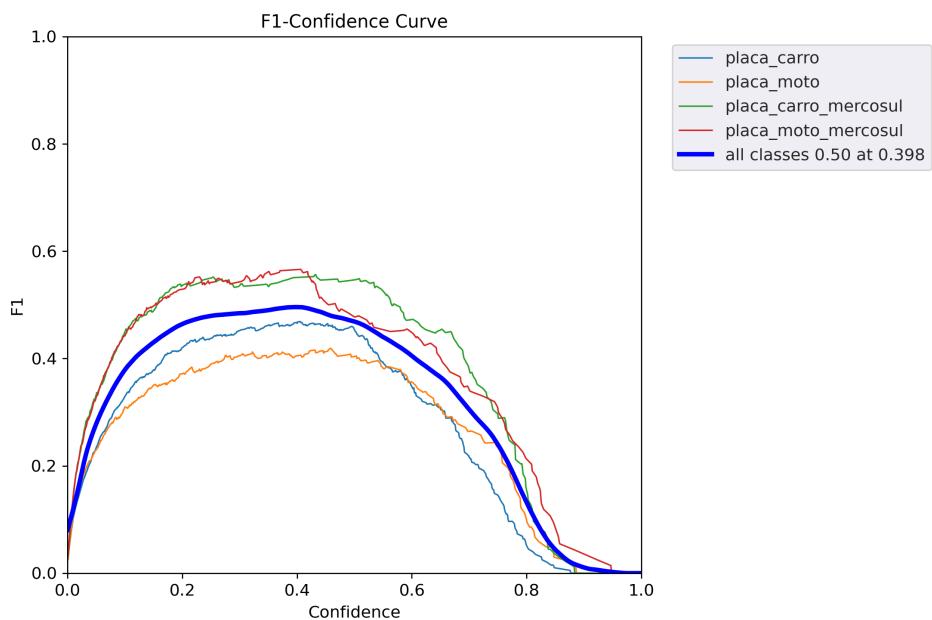
A abordagem inicial para a leitura de placas partiu da tentativa de construir uma ferramenta "do zero", utilizando bibliotecas como OpenCV, Tesseract e EasyOCR. No entanto, esse processo se mostrou muito desafiador, complexo e moroso, principalmente devido às variações nas características das imagens, como brilho, contraste, placas riscadas

Figura 3 – Matriz de confusão.



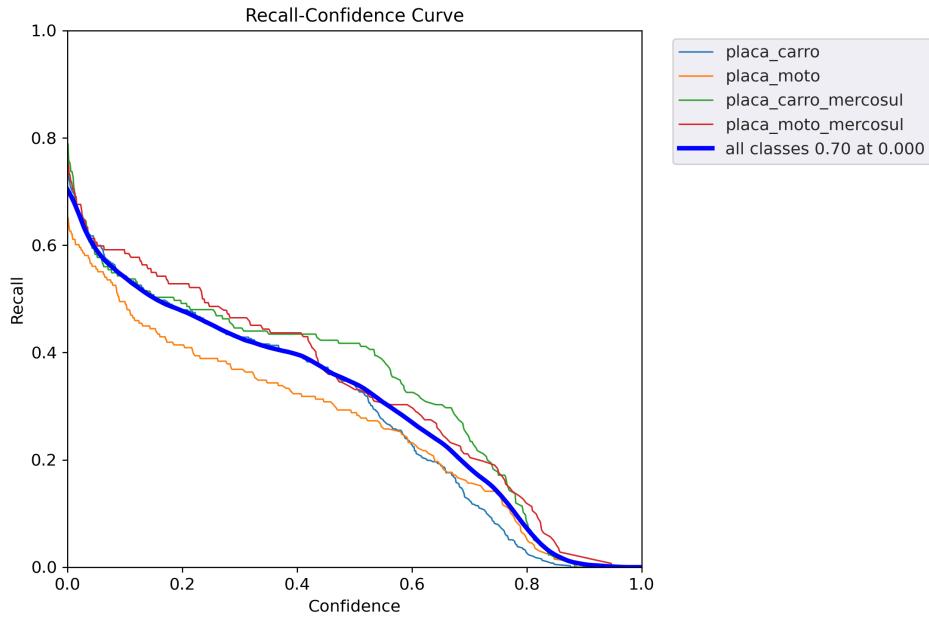
Fonte: Output de treinamento do Modelo Yolov5.

Figura 4 – Curva F1.



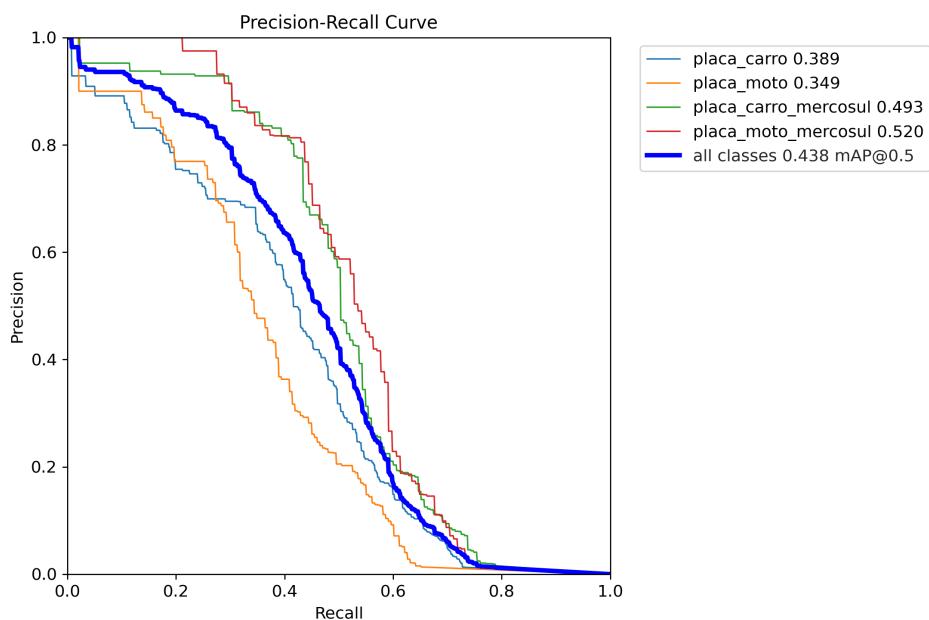
Fonte: Output de treinamento do Modelo Yolov5.

Figura 5 – Curva R.



Fonte: Output de treinamento do Modelo Yolov5.

Figura 6 – Curva PR.

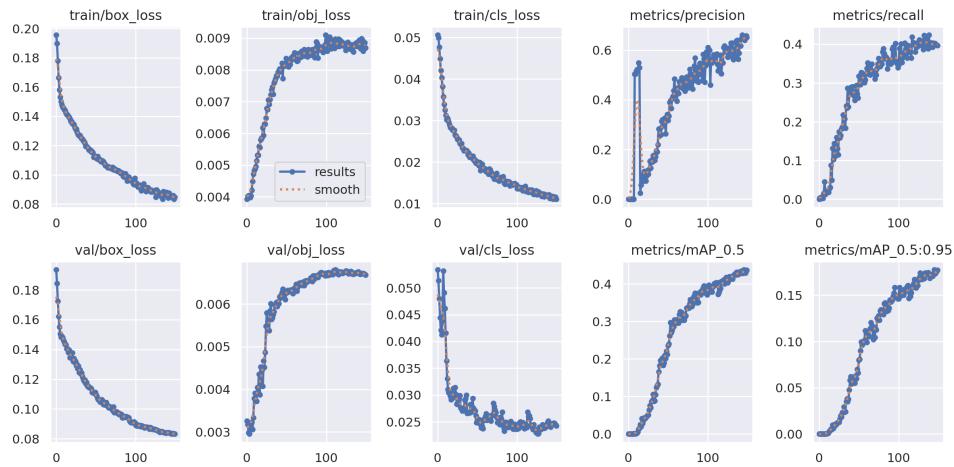


Fonte: Output de treinamento do Modelo Yolov5.

e rotação do texto. Esses fatores tornaram o desenvolvimento dessa frente pouco promissor.

Diante dessas dificuldades, a pesquisa por soluções existentes e não dependentes de API's pagas nos levou à biblioteca OpenALPR. Uma biblioteca open-source que se

Figura 7 – Resultados finais de treinamento e teste.



Fonte: Output de treinamento do Modelo Yolov5.

mostrou promissora durante os testes, sendo capaz de ler placas de diversos países e também retornar o resultado de forma estruturada. A detecção das placas nas imagens ocorre através da biblioteca Tesseract e também aplicando técnicas de processamento de imagens com OpenCV. Vale ressaltar que a biblioteca é utilizada nesse projeto apenas como leitor de placas, porém ela possui outras funcionalidades como detector de placa(que não se mostrou tão eficaz para nosso caso de uso), POST dos resultados e I/O (*Input/Output* - Entrada e saída de dados) próprio (que não funcionaram como deveriam seguindo a documentação) e por ser uma biblioteca desenvolvida em C++, linguagem que possuímos um conhecimento não tão profundo, não estão sendo utilizadas.

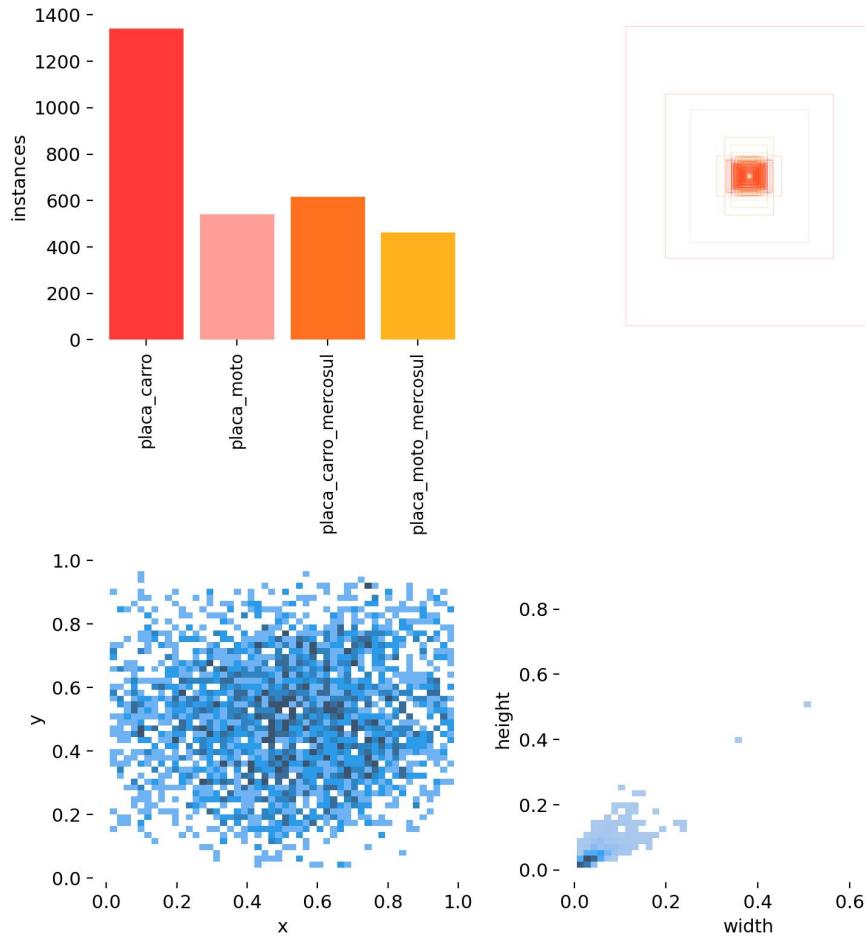
2.2 CONFECÇÃO DO CONJUNTO DE DADOS

O conjunto de dados de detecção de placas foi formado por capturas de *frames* de vídeos do YouTube, nos quais veículos transitavam, podendo ou não conter placas. A identificação das placas em cada *frame* foi realizada por meio da ferramenta Yolo Label. O processo de criação do conjunto de dados envolveu a categorização de 5.468 imagens, sendo distribuídas da seguinte maneira: 3.200 imagens foram destinadas ao conjunto de treinamento, 1.722 para o conjunto de teste e 546 para o conjunto de validação. A distribuição das entre as categorias pode ser observada na Figura 8.

Os dados de treinamento para a detecção de placas são compostos por:

- Arquivo ".txt" contendo as categorias existentes no conjunto de dados.;
- Imagens contendo ou não as categorias a serem detectadas.;
- Arquivo ".txt" com o nome de cada *frame* informando a localização da placa conforme documentação da biblioteca.;

Figura 8 – Distribuição das categorias.



Fonte: Output de treinamento do Modelo Yolov5.

Já para a leitura de placas, não foi criado nenhum conjunto de dados personalizado, apenas foram criados os arquivos de template com as devidas dimensões das placas brasileiras conforme legislação vigente e os padrões alfanuméricos a serem considerados.

2.3 DESENHO DA ARQUITETURA

2.3.1 Ambiente

Para o desenvolvimento e implantação do sistema, foi escolhido o ambiente Docker devido à sua flexibilidade, capacidade multiplataforma e eficiência na gestão de contêineres. A escolha do Docker oferece diversos benefícios, destacando-se a flexibilidade, reprodutibilidade, documentação como código, manutenção, escalabilidade e orquestração.

2.3.2 Serviços

- a) alpr (Automatic License Plate Recognition - Localiza placas na imagem (Yolov5 (JOCHER, 2020)).) - Baseado na biblioteca Yolov5(Python), detecta se há/ qual a localização da(s) placa(s) na imagem, salva-las e eventualmente salvar os frames;
- b) anpr (Automatic Number Plate Recognition - Identifica os digitos da placa (openalpr (HILL, 2020)).) - Baseado na biblioteca OpenALPR(C++), analisa as placas salvas pelo serviço alpr (Automatic License Plate Recognition - Localiza placas na imagem (Yolov5 (JOCHER, 2020)).), lê os dígitos através de OCR e salva o resultado no formato json;
- c) pred poster (Prediction Poster - Posta as placas identificadas no json server e/ou Kafka e as carimba nos respectivos frames.) - Desenvolvido em python, posta as placas identificadas em um servidor json local na porta 80 e também os envia ao serviço de mensageria(Kafka);
- d) json server (Servidor JSON para armazenamento local das placas identificadas baseado na biblioteca json-server (NPM, 2015).) - Baseado no pacote json-server, é um servidor básico que permite realizar requisições em arquivos de formato json;
- e) pred display (Mostra o frame das detecções em tempo real via gstreamer (WALTHINSEN, 2001).) - Desenvolvido em python, utilizando principalmente a biblioteca GStreamer, esse serviço grava a placa detectada no seu respectivo *frame* e após, inicia uma stream com o *frame* para ele aparecer na tela;
- f) pred etl (Pipeline de dados com Mage AI (DANG, 2021) que pega as placas do Kafka e envia para o BigQuery.) - ETL feito com MageAI que recebe as mensagens enviadas ao serviço de mensageria, transforma o json em informação tabular e insere a informação em tabela do BigQuery;
- g) frame renamer (Renomeia os frames para serem exibidos como vídeo via gstreamer.) - Desenvolvido em python, o serviço renomeia e move os *frames* gravados para um nome/pasta que o GStreamer interprete a sequência correta dos frames. Deve ser utilizado apenas quando a predição tiver sido encerrada;
- h) png streamer (Reproduz png's como vídeo via gstreamer (WALTHINSEN, 2001).) - Baseado na biblioteca GStreamer, o serviço cria uma stream através dos arquivos ".png" renomeados pelo serviço frame renamer.

Obs.: Os serviços "frame renamer" e "png streamer" são opcionais e devem ser executados após o encerramento das previsões caso queira assistir o vídeo completo dos *frames* e placas detectadas.

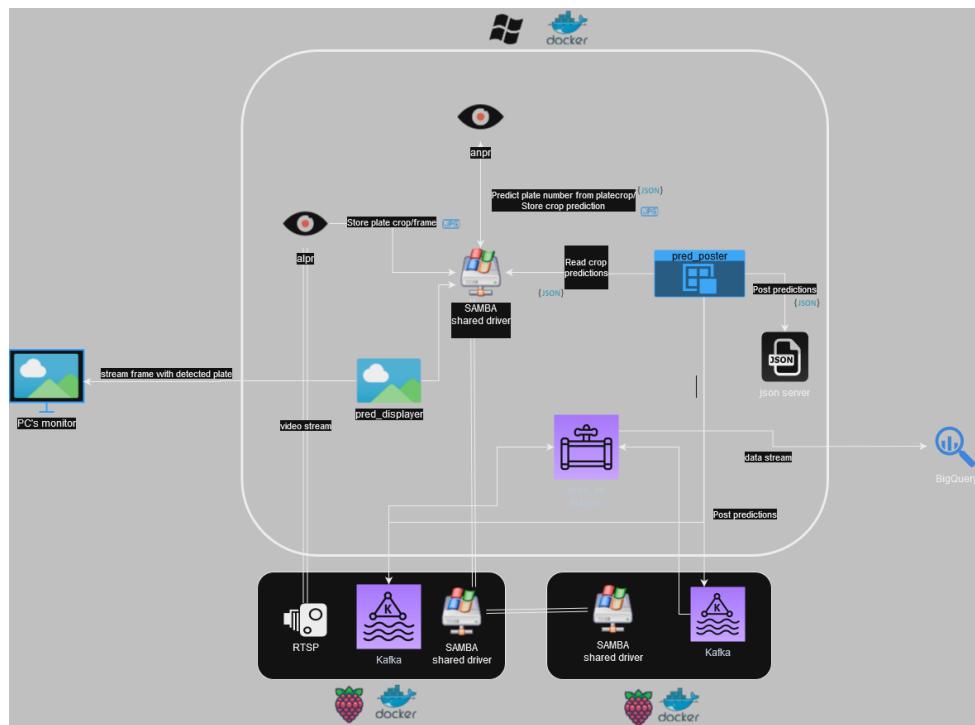
2.3.3 Comunicação entre serviços

A comunicação entre os serviços, desde a detecção da placa até o pred poster (Prediction Poster - Posta as placas identificadas no json server e/ou Kafka e as carimba nos respectivos frames.), é realizada por meio de entrada e saída de dados (IO) integrada entre os hosts pelo Samba Shared Driver. Este método permite que os serviços armazenem os resultados em um local compartilhado, facilitando o acesso das etapas subsequentes. A partir daí, as mensagens são enviadas via requests(json server) e Kafka(que será ingerido pelo pred etl e disponibilizadas via tabela no big BigQuery).

2.3.4 Diagrama da Arquitetura

Na Figura 9 podemos observar o diagrama da arquitetura desenhado conforme as premissas colocadas, ele não contém os serviços "frame renamer" e "png streamer" por serem execuções opcionais e posteriores às etapas de predição.

Figura 9 – Diagrama da Arquitetura.



Fonte: Autoria própria

2.4 DESENVOLVIMENTO E PERSONALIZAÇÃO DOS SERVIÇOS

2.4.1 alpr

A biblioteca Yolov5 é muito completa, bem documentada e manutenida, com suas várias opções de personalização. Porém, toda essa facilidade carrega uma complexidade técnica razoável na construção do sistema, se tornando o entendimento do código um dos principais desafios. Tal complexidade foi fundamental para o aprimoramento do nível técnico da equipe, que apesar da ferramenta em seu estado padrão atender boa parte dos requisitos, ainda faltavam alguns detalhes e melhorias, principalmente nos gargalos de gravação e leitura em disco. Para esse serviço, todas as melhorias aplicadas são acessíveis por argumentos na chamada do script, estando descritas abaixo:

- a) save_frame - Argumento de tipo booleando onde verdadeiro significa que é para salvar todos os *frames* e falso indica que não é para salvar nenhum frame;
- b) save_detected_frame - Argumento de tipo booleando onde verdadeiro significa que *frames* onde placas foram detectadas devem ser salvos (independente do argumento acima) já o falso apenas segue o comportamento padrão;
- c) save_each_n_frames - Argumento do tipo numérico onde o número significa a frequencia de salvamento dos frames, 1 significa que salvar-se-ão todos os frames, 2 indica um sim um não, 3 indica salvar a cada 3 *frames* e assim sucessivamente.

Outro desafio enfrentado por conta da escolha do ambiente Docker, a função do Yolo de mostrar os frames/stream de vídeo não funcionam por padrão e não foi encontrada nenhuma solução estável multiplataforma até o momento da confecção deste documento, logo esse requisito descumprido por conta do ambiente virou um novo serviço, o pred displayer (Mostra o frame das detecções em tempo real via gstreamer (WALTHINSEN, 2001).). Mesmo sendo por uma questão preciosista(ver as imagens em tempo real), esse requisito é essencial para podermos tornar as previsões do modelo mais palpáveis e em tempo de execução.

2.4.2 anpr

A biblioteca OpenALPR apesar de muito completa, como foi desenvolvida em C++, optamos por criar um *script* em python que lida com a complexidade de persistir o processamento das imagens das placas na ordem, caminhos e argumentos corretos que por fim executa via linha de comando a chamada do OpenALPR e gravando seu resultado em formato json dentro de um arquivo de logs. O *script* também é responsável pelo salvamento dos dados da ultima rodada de previsões, limpeza e criação dos novos diretórios. Para esse projeto foram criados os padrões de placa para atender as categorias sendo analisadas, totalizando 4 arquivos de configuração das características das placas, sendo esses:

- a) brg.conf - Características da placa de carro padrão antigo;

- b) brms.conf - Características da placa de carro padrão Mercosul;
- c) brmt.conf - Características da placa de moto padrão antigo;
- d) brmtms.conf - Características da placa de carro padrão Mercosul;

Adicionalmente foram criados 4 arquivos de configuração de padrão de dígitos de cada tipo de placa, sendo esses:

- a) brg.patterns - Sequência válida de dígitos para a placa de carro padrão antigo;
- b) brms.patterns - Sequência válida de dígitos para a placa de carro padrão Mercosul;
- c) brmt.patterns - Sequência válida de dígitos para a placa de moto padrão antigo;
- d) brmtms.patterns - Sequência válida de dígitos para a placa de moto padrão Mercosul;

Cada placa enviada para processamento do leitor de placas leva em conta de qual caminho ela veio para ser analisada com os parâmetros de configuração e padrão de dígitos corretos. Vale ressaltar que a biblioteca OpenALPR possui diversos parâmetros de configuração, listados na seção "Configuração do Sistema", que foram otimizados para nosso projeto. Mesmo com todo esse suporte da ferramente, a leitura de placas de motos no geral é um desafio por estar presente apenas na parte de trás do veículo e normalmente mais rotacionada.

2.4.3 pred poster

Desenvolvido "do início", consiste de um *script* em python que:

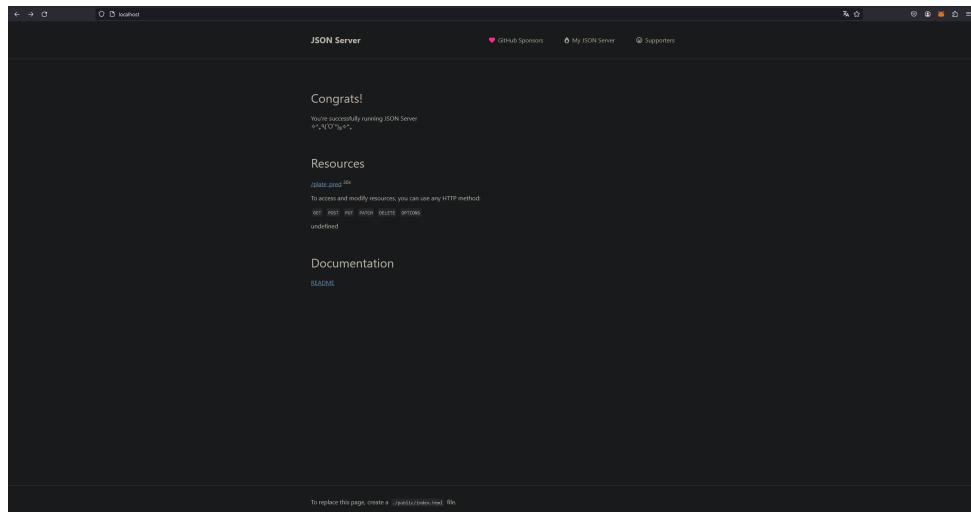
- a) Preparação do ambiente - Salva e limpa os dados da ultima execução do json server, cria o topico "plate_detector"no Kafka;
- b) Leitura de pastas - Procura pelos arquivos de leitura de placas com os resultados obtidos;
- c) Carimbo de imagens - Registra a(s) placa(s) detectada(s) no seu respectivo frame;
- d) Envio dos resultados - Posta os resultados das leituras de placas no servidor json e Kafka;
- e) Loop - Executa itens anteriores exceto item "a";

Lembrando que esse *script* contém variáveis que necessitam de alterações conforme seu ambiente, como, *string* de conexão do Kafka, url do servidor json e caso queira alterar o nome do tópico Kafka.

2.4.4 json server

Baseado no pacote json-server, é apenas uma versão em *container*, sem melhorias ou personalizações. veja tela inicial da ferramenta em Figura 10.

Figura 10 – Print da tela inicial do json server.



Fonte: Autoria própria

2.4.5 pred displayer

Desenvolvido do início, *script* na linguagem python que gerencia a ordem dos *frames* com placas lidas e os exibe via GStreamer um a um com delay personalizável. A biblioteca GStreamer também se mostrou um desafio por ser algo novo e também pela maneira como a biblioteca recebe e trata parâmetros.

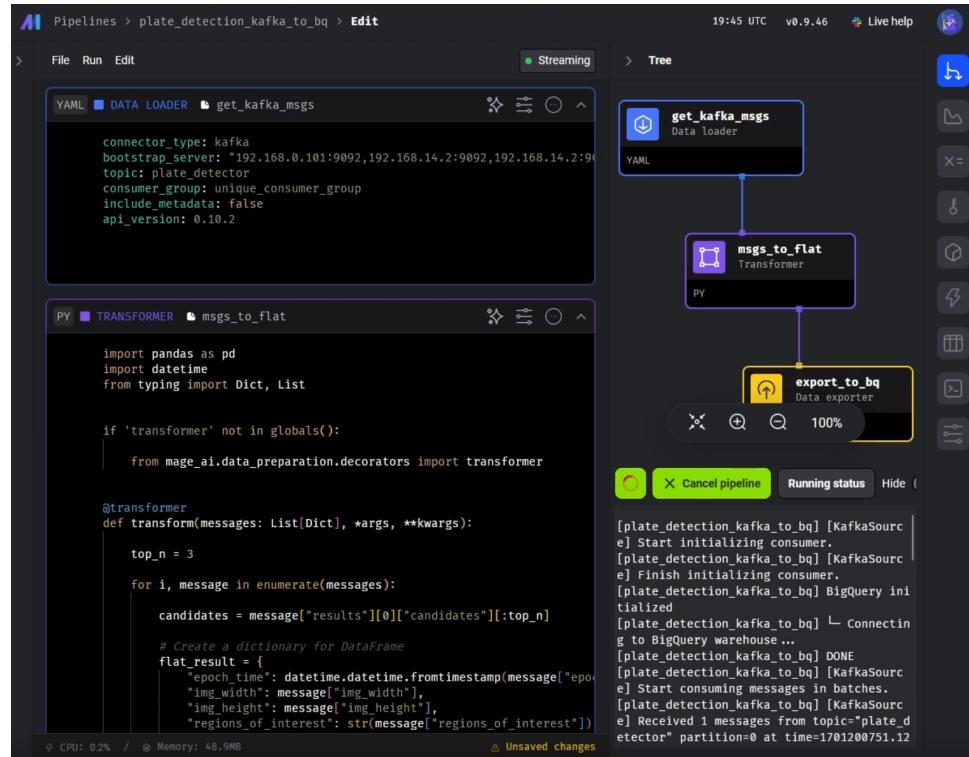
2.4.6 pred etl

O desenvolvimento se deu através da ferramenta low-code MageAI(vide Figura 11), tornando a construção de um streaming de dados em tempo real do Kafka ate o BigQuery simples e direta. Foi necessário apenas informar os caminhos de origem e destino, a chave de conta de serviço do big query e desenvolver um *script* python de como deve ocorrer a transformação dos dados em JSON para tabular.

2.4.7 frame renamer

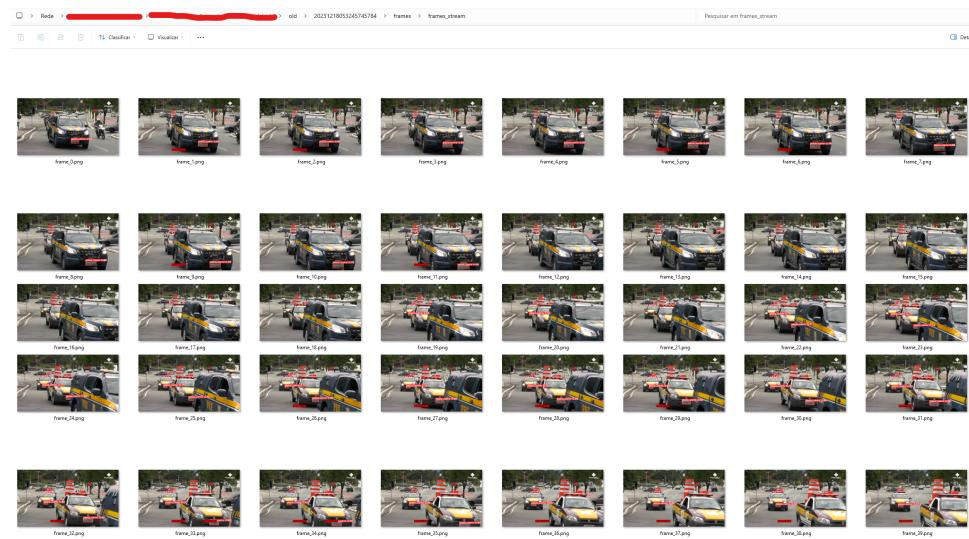
Script desenvolvido em python que copia e renomeia os *frames* para um nome que a biblioteca GStreamer entenda a ordem de reprodução, conforme Figura 12. O desenvolvimento desse *script* foi bem tranquilo porque as principais funções já existiam de alguma forma no projeto.

Figura 11 – Print da tela de desenvolvimento do Pipeline na ferramenta MageAI.



Fonte: Autoria própria

Figura 12 – Padrão de nomenclatura dos frames após renomeados.



Fonte: Autoria própria

2.4.8 png streamer

Desenvolvimento bem similar ao pred display, porém a saída é uma stream sequencial de todos os frames, após renomeados, formando um vídeo. Logo os desafios foram bem similares e relacionados ao próprio uso da biblioteca.

2.5 DISPONIBILIZAÇÃO DOS RESULTADOS

A disponibilização dos resultados se da através das seguintes formas:

- a) Logs - Logs armazenados no driver compartilhado Samba, veja Figura 13;
- b) Servidor JSON - Acessando "http://localhost:80" podemos conferir todas as placas postadas e realizar consultas personalizadas, veja Figura 14;
- c) Kafka - Se inscrevendo ao tópico "plate_detector" do seu cluster Kafka, veja Figura 15;
- d) Frames armazenados - Acessando os *frames* das imagens com o carimbo da predição, veja Figura 16;
- e) BigQuery - No caminho 'plate-detector-data.LANDING.detections' do BigQuery, veja Figura 17;
- f) Stream de frames - Na stream em tempo de execução *frame* a *frame* ou após renomear os frames, criando a stream completa.

Figura 13 – Exemplo de resultado em log.

```
{  
    "version": 2,  
    "data_type": "alpr_results",  
    "epoch_time": 1703849921533,  
    "img_width": 62,  
    "img_height": 35,  
    "processing_time_ms": 134.227539,  
    "regions_of_interest": [  
        {  
            "x": 0,  
            "y": 0,  
            "width": 62,  
            "height": 35  
        }  
    ],  
    "results": [  
        {  
            "plate": "COU9321",  
            "confidence": 77.339798,  
            "matches_template": 1,  
            "plate_index": 0,  
            "region": "",  
            "region_confidence": 0,  
            "processing_time_ms": 132.093765,  
            "requested_topn": 10,  
            "coordinates": [  
                {  
                    "x": 0,  
                    "y": 0  
                },  
                {  
                    "x": 60,  
                    "y": 0  
                },  
                {  
                    "x": 60,  
                    "y": 25  
                },  
                {  
                    "x": 0,  
                    "y": 25  
                }  
            ],  
            "candidates": [  
                {  
                    "plate": "COU9321",  
                    "confidence": 77.339798,  
                    "matches_template": 1  
                }  
            ]  
        }  
    ],  
    "file": "20231229113840036999_0.log",  
    "category": "placa_carro",  
    "id": "ubM2qC8"  
}
```

Fonte: Autoria própria.

Figura 14 – Exemplo de resultados exibidos via json server.

```

▶ e:          {}
▼ 1:
  version:      2
  data_type:    "alpr_results"
  epoch_time:   1704765370141
  img_width:    109
  img_height:   55
  processing_time_ms: 18.523729
▶ regions_of_interest: []
▼ results:
  ▶ 0:
    plate:        "OVO5202"
    confidence:   89.403023
    matches_template: 1
    plate_index:   0
    region:        ""
    region_confidence: 0
    processing_time_ms: 9.778885
    requested_topn: 10
    ▶ coordinates: []
    ▼ candidates:
      ▶ 0:
        plate:        "OVO5202"
        confidence:   89.403023
        matches_template: 1
      ▶ 1:
        plate:        "OVO5202"
        confidence:   86.999809
        matches_template: 1
      ▶ 2:
        plate:        "OVO5202"
        confidence:   86.734146
        matches_template: 1
      ▶ 3:
        plate:        "OVO5202"
        confidence:   84.453468
        matches_template: 1
      ▶ 4:
        plate:        "OVO5202"
        confidence:   84.330933
        matches_template: 1
      ▶ 5:
        plate:        "OVO5202"
        confidence:   84.093025
        matches_template: 1
  file:          "20240109015609485163_0.log"
  category:     "placa_carro"
  id:           "PVEVvst"

```

Fonte: Autoria própria.

Figura 15 – Exemplo de resultados exibidos via recebimento de mensagens Kafka em tela lcd.



Fonte: Autoria própria.

Figura 16 – Exemplo de resultados sendo exibidos no frame.



Fonte: Autoria própria.

Figura 17 – Resultados disponíveis via BigQuery.

Resultados da consulta

Linha	epoch_time	RELAÇÕES	GRÁFICO	PREVISÃO	JSON	DETALHES DA EXECUÇÃO	GRÁFICO DE EXECUÇÃO	plate	confidence	coordinates	plate_0	confidence_plate_0	plate_1	confidence_plate_1	coordinates_plate_1	epoch_time	confidence
1	2023-10-20T21:07:13.946000	99	img_width: 41	[[{"x": 0, "y": 0, "width": 89, "height": 100}]]	849572	2023102021044604751.jpg	CD4450	78.031022	By > Y < 0, Y > 0, X <	CD4450	78.031022	CD4450	82.234206	CD4450	82.234206	CD4450	
2	2023-10-20T21:07:14.623579000	133	63	[[{"x": 0, "y": 0, "width": 133, "height": 149}]]	14.96638	2023102021044604751.jpg	F019303	88.0319405	By > Y < 0, Y > 0, X < 133, Y < 149	F019303	88.0319405	F019303	88.0319405	F019303	88.0319405	F019303	
3	2023-10-20T21:07:21.7830000	58	32	[[{"x": 0, "y": 0, "width: 58, height": 32}]]	73.64079	2023102021044604751.jpg	DD04591	78.048520	By > Y < 0, C < 32, Y > 0, X <	DD04591	78.048520	DD04591	78.048520	DD04591	78.048520	DD04591	
4	2023-10-20T21:07:44.5137000	76	38	[[{"x": 0, "y": 0, "width: 76, height": 38}]]	12.10270	2023102021044604751.jpg	0004899	78.052020	By > Y < 0, C < 38, Y > 0, X <	0004899	78.052020	0004899	77.641884	0004899	77.641884	0004899	
5	2023-10-20T21:08:25.373000	112	58	[[{"x": 0, "y": 0, "width: 112, height": 58}]]	3.206496	2023102021044604751.jpg	D.844603	87.359870	By > Y < 0, C < 58, Y > 0, X < 112, Y < 58	D.844603	87.359870	D.844603	87.359870	D.844603	87.359870	D.844603	
6	2023-10-20T20:32:32.946000	74	36	[[{"x": 0, "y": 0, "width: 74, height": 36}]]	87.76002	2023102021044604751.jpg	0004503	79.26204	By > Y < 0, C < 36, Y > 0, X <	0004503	79.26204	0004503	77.707037	0004503	77.707037	0004503	
7	2023-10-20T20:32:33.546000	77	36	[[{"x": 0, "y": 0, "width: 77, height": 36}]]	9.88391	2023102021044604751.jpg	0004503	83.077915	By > Y < 0, C < 36, Y > 0, X <	0004503	83.077915	0004503	83.077915	0004503	83.077915	0004503	

Fonte: Autoria própria.

3 UTILIZAÇÃO

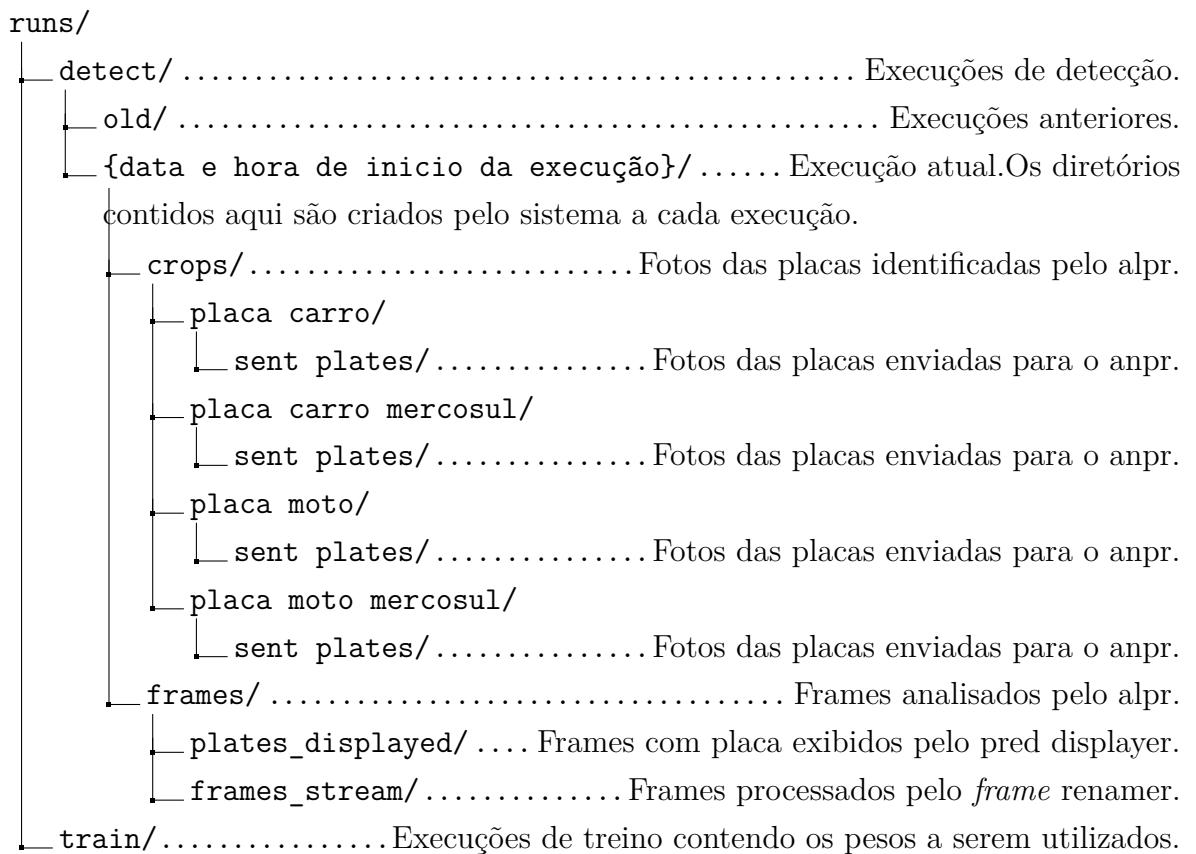
Nesta seção passaremos pelos pontos importantes a serem considerados e informações a serem fornecidas para o bom funcionamento do sistema.

3.1 ESTRUTURA DOS DIRETÓRIOS

A estrutura dos diretórios é uma parte crucial para o funcionamento do sistema, através dos diretórios que os serviços comunicam-se até certa etapa e é essencial entender onde está disposta cada informação.

3.1.1 Diretórios de I/O

A seguir, apresentamos o diretório "runs" que contém os *frames* e recortes das execuções do alpr e também os modelos e pesos de cada execução de treino.



Em seguida temos o diretório "logs", onde ocorre o armazenamento e transferência de todo tipo de informação de logs sobre o processamento das imagens, desde qual *frame* contém cada recorte até em qual parte da etapa a análise de uma imgem está.

`logs/{Nome da execução atual}/ .. Execução atual.` Os diretórios contidos aqui são criados pelo sistema a cada execução.

```

└── sent_plates/..... Arquivo de log contendo o nome das placas enviadas para processamento para cada categoria.
    ├── placa carro/
    ├── placa carro mercosul/
    ├── placa moto/
    └── placa moto mercosul/
    
└── processed_plates/Pasta intermediária(fila) contendo arquivos de logs das placas identificadas e ainda não postadas para cada categoria.
    ├── placa carro/
    ├── placa carro mercosul/
    ├── placa moto/
    └── placa moto mercosul/
    
└── posted_plates/. Armazena os arquivos de logs com as placas postadas para cada categoria.
    ├── placa carro/
    ├── placa carro mercosul/
    ├── placa moto/
    └── placa moto mercosul/

```

3.1.2 Diretórios de Configuração

A seguir apresentamos os diretórios que possuem arquivos de configuração para funcionamento do sistema.

`json_source/` Arquivo onde o json-server armazena as informações e também contém o modelo do formato esperado dos arquivos json a serem postados.

`/bq_key.json..` JSON contendo chave de configuração da conta de serviço do Google BigQuery.

3.1.3 Diretórios de Conjunto de dados

```

brplates/..... Contém os dados de treino, teste e validação.
└── labels/Contém um arquivo informando as categorias existentes para treinamento.
└── test/..... Conjunto de dados de teste.
└── train/..... Conjunto de dados de treino.
└── val/..... Conjunto de dados de validação.

```

3.2 DESEMPENHO

Abaixo, detalharemos sobre o desempenho dos 2 principais e mais computacionalmente custos serviços, alpr e anpr, nas 2 plataformas testadas, Windows e Raspberry e também as configurações de cada um.

3.2.1 Configurações de Hardware

A seguir descrevemos as configurações de cada hardware utilizado.

3.2.1.1 Computador de Mesa

Processador: Intel Core i5 12600K @ 4.9 GHz, Memória:64GB DDR5 @ 5768 MHz, Armazenamento sistema: NVME, Armazenamento volume: driver compartilhado de rede do Samba(porta Gigabit e cabos de rede padrão CAT6),Sistema Operacional:Windows 11, Placa de Vídeo: RTX3060 Ti.

3.2.1.2 Raspberry PI 4

Processador:Quad-core ARM Cortex-A72 @ 1.5 GHz, Memória:8GB LPDDR4 @ 3200 MHz, Armazenamento sistema: cartão de memória, Armazenamento volume: SSD via UBS 3.0, Sistema Operacional: Raspberry Pi OS 64-bit. Obs.: Vale lembrar que os serviços legados rodando no dispositivo(DHCP, SAMBA, Kafka,NFS e TFTP) também consomem parte dos recursos que já não são tão abundantes.

3.2.2 alpr

Abaixo podemos observar o desempenho para o serviço alpr.

3.2.2.1 Computador de Mesa

O serviço foi bem performático dado a alta disponibilidade de recursos, levando em média 15 milissegundos por frame(salvando a cada 3 *frames* e somente onde placas foram detectadas).A capacidade de processamento de vídeo contribuiu bastante para esse tempo baixo de processamento, porém, o I/O (*Input/Output* - Entrada e saída de dados) é um ponto de atenção, garantir que estamos salvando somente o necessário.

3.2.2.2 Raspberry PI 4

A média de tempo para o serviço analisar um *frame* foi de 400 milissegundos, apesar de ser um tempo bem maior, ainda é satisfatório pensando que um equipamento atenderia a 1 faixa de acesso

3.2.3 anpr

Aqui apresentamos o desempenho para o serviço anpr.

3.2.3.1 Computador de Mesa

O serviço também foi bem performático dado a alta disponibilidade de recursos, levando em média 12 milissegundos por recorte de placa.

3.2.3.2 Raspberry PI 4

A média de tempo para o serviço analisar um *recorte de placa* foi de 50 milissegundos, muito dessa performance vem da eficiência da linguagem de programação de baixo nível em que o serviço foi desenvolvido, apesar de ser um tempo quase 5x maior, ainda é muito satisfatório para o seu propósito.

3.3 PRÓXIMOS PASSOS

Ainda sim, o projeto possui pontos de melhoria, aprimoramento e atenção que descreveremos nas seções a seguir.

3.3.1 Otimização de Imagens

Otimizar as imagens Docker, fornecendo apenas o mínimo necessário para a aplicação funcionar, principalmente em bibliotecas terceiras como a do anpr e alpr (Yolov5 e OpenALPR)

3.3.2 Desacoplamento e Dependências

Apesar da aplicação já estar bem dividida, ainda existem funções que seria importante desenvolver o desacoplamento e/ou encadeamento, como a criação dos diretórios base que hoje é realizada no start do anpr e a exibição do *frame* apenas após a imagem ser carimbada com a placa(no Windows não foi necessário, porém, no Raspberry se mostrou uma melhoria pertinente). Existem também algumas funções/configurações que podem ser transformadas em argumentos para uma personalização mais fácil durante o uso.

3.3.3 Aferição de Resultados

Devido à natureza desestruturada da informação sendo tratada(imagens e caracteres dentro de imagens) a aferição definitiva dos resultados pode-se dar apenas manualmente, imagem a imagem identificando o acerto ou não de cada placa, conjunto e unidade de dígitos. Para um ótimo retreino dos modelos é necessário estruturar um pipeline considerando essas etapas. Ademais, observamos uma assertividade satisfatória tanto em identificar a

localização quanto em ler a placas, existem sempre os caracteres dúbios como "i", "j", "0", "O" e etc., mas esses podem ser tratados tanto através de aumentando o limite de certeza da indicação quando observando a frequência no intervalo de tempo, ex.: se em 10 placas, 1 leu "DJI0984" e as outras 9 "DJJ0084", com certeza a placa mais provável do intervalo foi a "DJJ0084", mas novamente, é necessária uma forma estruturada de mensurar esses resultados.

3.3.4 Treinamento

O treinamento da detecção de placas se mostrou bem flexível e versátil, porém ainda sim, podem existir situações onde seja preciso treinar para diferentes condições de ambiente, luz e contraste, nesse caso teríamos que incrementar o conjunto de dados de treino do alpr. Já para o anpr, o treino para a leitura de placas mais acurada depende da criação de conjunto de dados de letras, letras rotacionadas, com e sem contraste e todas rotuladas, talvez faça sentido avançar nesse ponto caso a acurácia seja algo inegociável em cada frame.

3.4 CÓDIGO FONTE

Todo o código fonte do projeto se encontra em <https://github.com/gustavozantut/Toll>.

4 CONCLUSÃO

Os principais objetivos do projeto foram alcançados, chegamos a um sistema de detecção e leitura de placas eficaz e extremamente flexível quanto a integração para comunicação das informações, bem como, fornecendo resultados coesos e bem estruturados sobre a leitura de placas. Esse desenvolvimento mostra que é possível através de micro serviços distribuídos em multi plataformas são plenamente capazes de fornecer dados em tempo perto do real, por diversos tipos de saída, sobre a leitura de placas em imagens, sendo também possível adaptar o sistema para realizar processamentos em lotes, integrar com sistemas de validação de placas/saldo em aplicativos de pedágio, sistemas de segurança de condomínio, controle de frotas e uma infinidade de outras aplicações. De qualquer forma, há de se observar requisitos como tempo de resposta esperado, qualidade da imagem, tempo médio de processamento de um frame, latencia das imagens, disponibilidade de recursos computacionais, posição da câmera, posição do sol e uma série de outros parâmetros e variáveis para desenhar uma arquitetura robusta que suporte a capacidade total do sistema garantindo seu ótimo aproveitamento.

REFERÊNCIAS

DANG, Tommy. **Mage AI**. [S.l.: s.n.], 2021. Disponível em:
<https://github.com/mage-ai/mage-ai>.

HILL, Matthew. **openalpr**. [S.l.: s.n.], 2020. Disponível em:
<https://github.com/openalpr/openalpr>.

JOCHER, Glenn. **Ultralytics YOLOv5**. [S.l.: s.n.], 2020. Disponível em:
<https://github.com/ultralytics/yolov5>.

NPM. **json-server**. [S.l.: s.n.], 2015. Disponível em:
<https://www.npmjs.com/package/json-server>.

SAGAN, Carl. **Cosmos**. [S.l.]: Ballantine Books, 2011.

WALTHINSEN, Erik. **Gstreamer**. [S.l.: s.n.], 2001. Disponível em:
<https://gitlab.freedesktop.org/gstreamer/gstreamer>.