Trabalho de Conclusão de Curso

Introduzindo MLOps: reconhecimento de placas de licença de caminhões

de **Warley Vital Barbosa**

orientado por
Prof. Dr. Tiago Figueiredo Vieira

Universidade Federal de Alagoas
Instituto de Computação
Maceió, Alagoas
19 de Dezembro de 2022

UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação

INTRODUZINDO MLOPS: RECONHECIMENTO DE PLACAS DE LICENÇA DE CAMINHÕES

Trabalho de Conclusão de Curso submetido ao Instituto de Computação da Universidade Federal de Alagoas como requisito parcial para a obtenção do grau de Cientista de Computação.

Warley Vital Barbosa

Orientador: Prof. Dr. Tiago Figueiredo Vieira

Banca Avaliadora:

Ícaro Bezerra Queiroz de Araújo Prof. Dr., UFAL
Bruno Georgevich Ferreira Me., UFAL

Maceió, Alagoas
19 de Dezembro de 2022

Catalogação na Fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico

Bibliotecário: Marcelino de Carvalho Freitas Neto – CRB-4 - 1767

B238i Barbosa, Warley Vital.
Introduzindo MLOps : reconhecimento de placas de licença de caminhões/ Warley Vital Barbosa. – 2022.
33 f. : il.

Orientador: Tiago Figueiredo Vieira.
Monografia (Trabalho de conclusão de curso em Ciência da Computação) – Universidade Federal de Alagoas, Instituto de Computação. Maceió, 2022.

Bibliografia: f. 32-33.

1. Aprendizado do computador. 2. *Software de* Aprendizado do computador. 3. Reconhecimento automático. 4. Sinais e placas de sinalização. 5. Reconhecimento de caracteres ópticos. I. Título.

CDU: 004.81:159.953.5

Agradecimentos

Aos meus pais Edmilson e Weliane, pelos ensinamentos que me ajudaram a trilhar esse caminho e por sempre me apoiarem nas minhas decisões.

Aos meus avós, Enoque e Gidelma, pelo carinho inesgotável.

Ao meu irmão Walysson, que me inspirou a iniciar essa caminhada pela computação há 12 anos. Ao meu irmão Fausto, que não para de me ensinar o que é ter disciplina e vontade de alcançar horizontes cada vez mais altos.

Às minhas primas Daniela, Vivianne e Andressa, que conseguiram suportar minha imaturidade por dois anos no início dessa graduação.

Aos meus amigos Elvys e Douglas, por todas as conversas, ideias, incentivos e broncas.

Ao meu orientador Prof. Tiago Vieira pelos ensinamentos sobre liderança, pela confiança e, especialmente, paciência.

Ao Centro de Inovação EDGE, pelos constantes desafios e pelas ótimas pessoas com as quais me permite trabalhar.

À minha namorada Talita, que sempre me apoiou e cuidou de mim, mesmo quando eu não mereci.

22 de Dezembro de 2022, Maceió - AL

Resumo

Grande parte dos projetos de *Machine Learning* (ML) não chegam a produção. Há vários desafios enfrentados no que tange o gerenciamento de dados e modelos, logo não há soluções universais. O novo paradigma de *Machine Learning Operations* (MLOps) foi criado justamente para abordar esses desafios e tornar o ciclo de vida do desenvolvimento de modelos de aprendizado de máquina mais eficiente e eficaz. Portanto, neste trabalho, foram introduzidos os principais conceitos de MLOps através da apresentação de uma solução de *Optical Character Recognition* (OCR) com modelos de detecção de objetos para a tarefa de *Automatic License Plate Recognition* (ALPR) já validada na indústria. Algumas ferramentas de MLOps foram também introduzidas de forma a exemplificar os principais conceitos de MLOps na prática. Os resultados mostraram a facilidade na criação e manutenção *pipelines* de dados, bem como deixam claro como tais processos são flexíveis o suficiente para que o fluxo de dados seja replicável, com o mínimo de mudanças, em contextos similares, como OCR de códigos vagões ou containers. Por fim, também foram detalhadas atividades que podem ser adicionadas no *workflow* para que as operações de desenvolvimento e implantação de modelos de ML sejam mais completas.

***Palavras-chave*: *Aprendizado de Máquina*, *Operações de Aprendizado de Máquina*, *Sistemas de Software de Aprendizado de Máquina*, *Reconhecimento Automático de Placas de Licença*, *Reconhecimento Óptico de Caracteres*.**

Abstract

Most Machine Learning (ML) projects do not reach production. Several challenges are faced when managing data and models, so there are no universal solutions. The new paradigm of Machine Learning Operations (MLOps) was created precisely to address these challenges and make the lifecycle of the development of machine learning models more efficient and effective. Therefore, in this work, the main concepts of MLOps were introduced through the presentation of an Optical Character Recognition (OCR) solution with object detection models for the Automatic License Plate Recognition (ALPR) task already validated in the industry. Some MLOps tools were also introduced to exemplify the main concepts of MLOps in practice. The results showed the ease of creating and maintaining data pipelines and making it clear how flexible such processes are so that the data flow is replicable, with minimal changes, in similar contexts, such as OCR of wagon or container codes. Finally, it was detailed which activities can be added to the workflow so that the operations of developing and deploying ML models are more complete.

***Keywords*: *Machine Learning*, *Machine Learning Operations*, *Machine Learning Software Systems*, *Automatic License Plate Recognition*, *Optical Character Recognition*.**

Lista de Figuras

Lista de Abreviaturas

ALPR *Automatic License Plate Recognition.* vi, 2–6, 9, 21, 27, 29, 31

CD *Code Detection.* 19, 26

CI/CD *Continuous Integration and Continuous Delivery.* 4, 10, 31

CNN *Convolutional Neural Network.* 6

CR *Code Recognition.* 19, 21, 26

DevOps *Development and Operations.* 1

DL *Deep Learning.* viii, 6

MAP *Mean Average Precision.* 9

ML *Machine Learning.* vi, viii, 1, 4, 7

MLOps *Machine Learning Operations.* vi, 1, 4, 31

MLSS *Machine Learning Software Systems.* 1–3, 31

OCR *Optical Character Recognition.* vi, 2, 4, 5

SCM *Software Configuration Management.* 22, 23

SMB *Server Message Block.* 16

VPN *Virtual Private Network.* 16

YOLO *You Only Look Once.* 6

Sumário

Capítulo 1

Introdução

1.1 Motivação

Aproximadamente 90% dos modelos de Aprendizado de Máquina (ML, do inglês *Machine Learning*) nunca chegam a produção (Weiner, 2020). Seja por sua natureza experimental, que invariavelmente tende a criar modelos ruins que precisam ser descartados, seja pela inerente dificuldade de implantar e manter modelos em produção de forma confiável e eficiente (Shankar *et al.*, 2022).

São vários desafios que precisam ser estudados e resolvidos se o objetivo é operacionalizar (i.e., desenvolver e implantar) modelos de aprendizado de máquina. Questões acerca do gerenciamento dos dados, ou implantação e versionamento dos modelos, bem como a escolha das ferramentas adotadas precisam ser respondidas. Frequentemente, não há soluções universais para esses problemas, mas sim soluções que precisam ser adaptadas ao contexto específico do projeto.

O uso de ML pode trazer muitos benefícios, mas também apresenta desafios específicos. Quando um projeto de ML é planejado cuidadosamente, é possível minimizar os riscos de acumular débito técnico, ou seja, a necessidade de investir muito tempo e esforço para resolver problemas ou adotar tecnologias que não foram devidamente consideradas no planejamento inicial (Sculley *et al.*, 2014).

Nesse sentido, um novo paradigma emergiu para que os *softwares* dos dias atuais pudessem utilizar atributos da área de aprendizado de máquina. Esse paradigma é chamado de Operações de Aprendizado de Máquina (MLOps, do inglês *Machine Learning Operations*). Similar à cultura de Desenvolvimento e Operações de *software* (DevOps, do inglês *Development and Operations*), o novo paradigma objetiva producionalizar, isso é, implementar e manter de forma consistente e eficiente, os Sistemas de Aprendizado de Máquina (MLSS, do inglês *Machine Learning Software Systems*) ao preencher a lacuna entre o desenvolvimento e as operações (Kreuzberger, Kühl e Hirschl, 2022).

Ainda no início de seu desenvolvimento, MLOps é um assunto pouco conhecido por

muitos profissionais da área, como cientistas de dados e engenheiros de aprendizado de máquina. Isso pode ser comparado à situação de DevOps no passado. Mäkinen *et al.* (2021), a partir de uma pesquisa com 331 cientistas de dados, divide as organizações representadas na pesquisa em três categorias: aquelas que precisam descobrir a melhor forma de usar os dados, outras que focam em desenvolver e implantar o primeiro modelo e, por fim, as que gerenciam vários modelos e *datasets*, bem como retreinos e reimplantações. Os autores também concluem que MLOps é um passo natural nas organizações que transitam o uso de ML além da prova de conceito, isso é, utilizando modelos de ML para auxiliar nas suas atividades rotineiras, principalmente as que necessitem de frequentes treinamentos e implantações.

No trabalho de Antunes *et al.* (2022), por exemplo, foi proposta uma infraestrutura para o gerenciamento de projetos de ciência de dados, que permita a colaboração, reprodutibilidade, implantação e monitoramento de modelos. B. M. A. Matsui e D. H. Goya (2021) buscaram compreender quais os cenários e os desafios para implementação de DevOps em processos de ML. Em outro trabalho recente (B. Matsui e D. Goya, 2022), os mesmos autores propuseram um conjunto de artefatos (diagramas e/ou figuras ilustrativas) para representar boas práticas em MLOps, visando uma melhor compreensão e adoção dos processos de MLOps por parte dos profissionais e pesquisadores da área.

Souza (2021) apresentou, através de um estudo de caso, um mapeamento de benefícios e desafios ao se adotar MLOps em projetos de ML. Similarmente, Silva *et al.* (2021) desenvolveram um guia voltado a profissionais de ML que apresenta todo o ciclo de desenvolvimento e implantação de modelos usando práticas de MLOps. Com isso, os autores buscaram introduzir os principais conceitos do paradigma MLOps para capacitar empresas parceiras atuais e futuras.

Aguiar (2020) relatou sua experiência ao coordenar a adoção dos processos de MLOps em uma parceria entre academia e governo. O autor descreveu os principais desafios e lições aprendidas na adoção de MLOps, como a necessidade de se manter um registro de modelos. Nogare, Mello e Lopes (2022) relatam as dificuldades encontradas por times de ciência de dados na gestão do ciclo de vida de modelos de aprendizado de máquina e propuseram uma plataforma de MLOps que as solucionam.

Experiências ruins na fase de implantação de modelos ML podem prejudicar sua adoção na indústria (Paleyes, Urma e Lawrence, 2020). Assim, tendo em vista os desafios no desenvolvimento de MLSS, este trabalho busca apresentar uma solução de Reconhecimento Óptico de Caracteres (OCR, do inglês *Optical Character Recognition*) para a tarefa de Reconhecimento Automático de Placas de Licença (ALPR, do inglês *Automatic License Plate Recognition*) validada e implantada em uma aplicação industrial, utilizando estratégias e métodos de MLOps. Nesse contexto industrial, portanto, a utilização de MLOps é justificada pelo fato de que seus processos podem automatizar diversos processos relacionados ao desenvolvimento e operação de sistemas de aprendizado de máquina,

como a preparação de dados, o treinamento, a validação e o monitoramento de modelos, o que pode economizar tempo e reduzir a margem de erro humano.

1.2 Objetivos

1.2.1 Objetivos Gerais

Introduzir os principais conceitos do campo de Operações de Aprendizado de Máquina (e.g., gerenciamento de dados, desenvolvimento de modelos, verificação de modelos, e implantação de modelos) atráves do desenvolvimento de uma solução implantada em um cenário industrial. Espera-se, ao fim, deixar claro como um conjunto de ações padronizadas facilitam o desenvolvimento e implantação de MLSS.

1.2.2 Objetivos Específicos

Pretende-se alcançar o objetivo geral através do uso dos seguintes objetivos específicos;

1. Versionar dados – visa garantir integridade e consistência dos dados, bem como permitir a reprodutibilidade dos modelos desenvolvidos;
2. Treinar modelos de detecção de objetos – visa solucionar a tarefa ALPR em amostras de caminhão;
3. Converter modelo para um formato aberto – visa checar possibilidade de implantação na aplicação;
4. Realizar *benchmark* de performance – visa checar performance da aplicação;
5. Implantar modelos na aplicação – visa tornar os modelos disponíveis para uso em produção;
6. Implementar as etapas de MLOps – visa criar *pipelines* de dados para automatizar os processos de desenvolvimento e implantação dos modelos;

1.3 Limitações

O trabalho aqui apresentado vem sendo o suporte para parte dos *softwares* desenvolvidos pelo laboratório Centro de Inovação EDGE [1], onde o autor colocou em prática seus conhecimentos nas áreas de inteligência artificial e visão computacional. Uma importante característica dos projetos no EDGE é a variedade de tarefas a solucionar, que vão desde

[1] `http://edgebr.org/`

o reconhecimento de objetos tradicional, até a combinação de diversos modelos e estratégias de processamento de imagens que compõe soluções industriais. No que diz respeito à implantação, os modelos desenvolvidos podem ser tanto integrados em um *software* complexo, como implantados em um *container* em um dispositivo de borda.

Uma das principais limitações desse trabalho é que não existe uma solução única que possa resolver todos os problemas enfrentados. É preciso considerar o contexto e as necessidades específicas de cada *software* e adaptar a abordagem de acordo. Além disso, é importante salientar que os processos que compõe MLOps ainda estão sendo aprimorados, buscando flexibilidade suficiente para abranger as diferentes características dos desafios apresentados. Instruir a empresa parceira para a implantação de mecanismos em seus processos de trabalho também tende a ser difícil por causa dos processos já estabelecidos. Portanto, algumas etapas, como *Continuous Integration and Continuous Delivery* (CI/CD) e monitoramento, ou subetapas, como verificação da integridade dos dados, não serão abordadas por ainda não terem sido sistematizadas no fluxo de desenvolvimento e implantação dos modelos, apesar de algumas serem realizadas oportunamente.

1.4 Organização do Trabalho

O trabalho foi organizado de forma a apresentar ao longo dos capítulos tanto a metodologia quanto o desenvolvimento do caso de uso. No Capítulo 2 é apresentado um embasamento teórico sobre *Optical Character Recognition* (OCR), *Automatic License Plate Recognition* (ALPR) e *Machine Learning Operations* (MLOps). No Capítulo 3 são descritas as etapas desenvolvidas, como as ferramentas utilizadas, a construção da base de dados, o modelo escolhido, bem como o fluxo de desenvolvimento e implantação de modelos de *Machine Learning* (ML). O Capítulo 4 é dedicado a exposição e discussão das consequências de se aderir às boas práticas de MLOps, mesmo que por enquanto incipientes. Finalmente, são apresentadas as conclusões do trabalho.

Capítulo 2

Fundamentação Teórica

2.1 OCR de Placas de Licença

Optical Character Recognition (OCR) é uma tecnologia que permite a leitura e reconhecimento automático de caracteres presentes em imagens ou documentos digitalizados. O OCR é uma ferramenta útil para digitalizar documentos e transformá-los em arquivos eletrônicos, para facilitar o acesso, armazenamento e compartilhamento. Além da digitalização de documentos, o OCR também pode ser utilizado em aplicações de leitura de tela, que permitem aos usuários com deficiência visual acessar conteúdo em dispositivos eletrônicos. Para realizar o OCR, é necessário utilizar algoritmos de processamento de imagem e técnicas de reconhecimento de padrões para analisar a imagem e identificar os caracteres presentes nela. Depois de identificados, os caracteres são convertidos em texto eletrônico, que pode ser armazenado, editado ou utilizado em outras aplicações.

Um sistema de OCR tradicional consiste de múltiplos estágios de processamento, sendo eles: aquisição da imagem, pré-processamento, segmentação dos caracteres, extração de atributos, classificação e, possivelmente, pós-processamento. Parte desses estágios podem ser substituídos por redes neurais profundas (ver Figura 2.1). Um exemplo é o trabalho proposto por LeCun *et al.* (1989), que apresenta uma aplicação de redes neurais para o reconhecimento de dígitos manuscritos após um pré-processamento mínimo dos dados e uma arquitetura da rede altamente restrita e especificamente projetada para a tarefa. Hoje, como o maior volume de dados, capacidade computacional e melhores arquiteturas de redes neurais, sistemas de OCR tendem a serem compostos por uma ou mais redes neurais, possibilitando soluções completas para tarefas cada vez mais complexas, desde a entrada dos dados até a saída das previsões, sem a necessidade de intervenção humana ou pré-processamento de dados (Li *et al.*, 2022).

Outra aplicação importante do OCR é o reconhecimento automático de placas de licença veículo, comumente chamada de *Automatic License Plate Recognition* (ALPR), que pode ser utilizado em sistemas de vigilância ou em sistemas de gerenciamento de

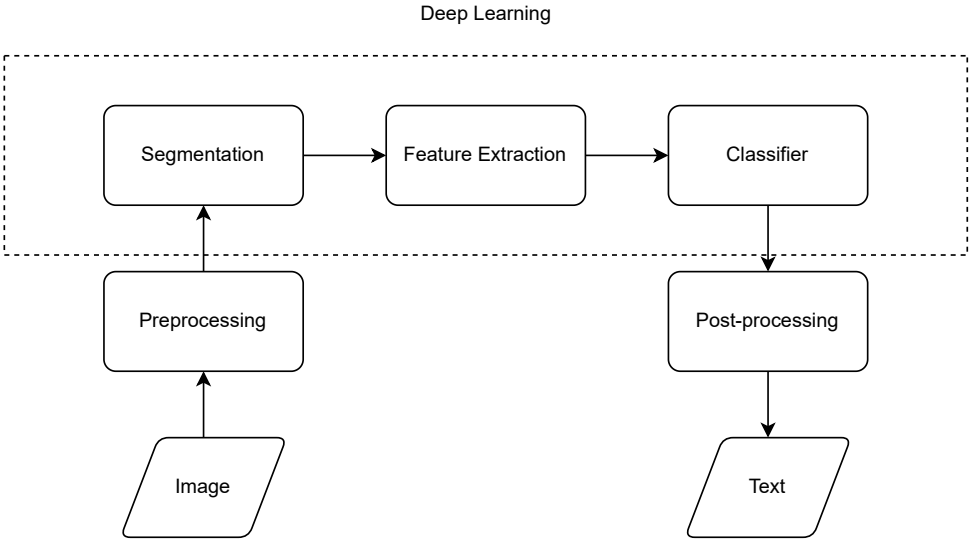

Figura 2.1: Estágios de um sistema OCR clássico. Aprendizado Profundo (DL, do inglês *Deep Learning*) é capaz de substituir algumas etapas. Diagrama feito pelo autor.

tráfego. Dentre as tarefas de ALPR, duas são: detecção da placa e reconhecimento dos caracteres contidos nela. Weihong e Jiaoyang (2020) elencaram os algoritmos mais utilizados na resolução da tarefa de reconhecimento de placas de licença, bem como os desafios técnicos enfrentados. Pode-se notar que os modelos provenientes da família *You Only Look Once* (YOLO) tendem a serem uma das principais escolhas, especialmente em contextos onde tempo de inferência deve ser curto.

A arquitetura de rede neural YOLO é uma técnica de detecção de objetos em imagens e vídeos, desenvolvida por Redmon *et al.* (2016). Ela é baseada em uma rede neural convolucional profunda (CNN, do inglês *Convolutional Neural Network*) e é capaz de detectar e classificar vários objetos em uma imagem ou vídeo de uma única vez, ao contrário de técnicas anteriores que precisavam examinar cada região da imagem separadamente. É também considerada uma das mais rápidas e precisas técnicas de detecção de objetos, e tem sido amplamente utilizada em aplicações como vigilância por câmeras, reconhecimento de placas de licença e detecção de objetos em imagens de satélite. Montazzolli e Jung (2017), por exemplo, propuseram um sistema baseado na arquitetura para construir um sistema ALPR. Ademais, a arquitetura YOLO vem sendo constantemente melhorada ao longo dos anos (P. Jiang *et al.*, 2022), com a adição de informações de multiescala, novas funções de ativação e perda, entre outras mudanças, o que tende a mantê-la como parte de diversas soluções de detecção de objetos.

2.2 MLOps

MLOps é um paradigma recente e sem uma definição formal consensual. No entanto, Kreuzberger, Kühl e Hirschl (2022) conduziram uma pesquisa para identificar os

princípios, componentes e papéis necessários de MLOps e, ao fim, gerar uma arquitetura geral para o design de sistemas de aprendizado de máquina. Os autores definem as operações de aprendizado de máquina como "um paradigma, incluindo aspectos como boas práticas, conjuntos de conceitos, bem como uma cultura de desenvolvimento quando se trata da conceitualização, implementação, monitoramento, implantação e escalabilidade fim-a-fim de produtos de aprendizado de máquina" [1]. Em outras palavras, as operações de aprendizado de máquina englobam todos os aspectos de desenvolvimento e gerenciamento de sistemas de aprendizado de máquina, desde a criação de modelos até sua implantação e manutenção em produção.

Segundo Ashmore, Calinescu e Paterson (2021), o processo de desenvolvimento de uma solução de ML na indústria consiste de quatro estágios: gerenciamento de dados, aprendizado do modelo, verificação do modelo e implantação do modelo. Cada um desses estágios pode ser dividido em passos menores, que podem ou não ser sequenciais (ver Figura 2.2).

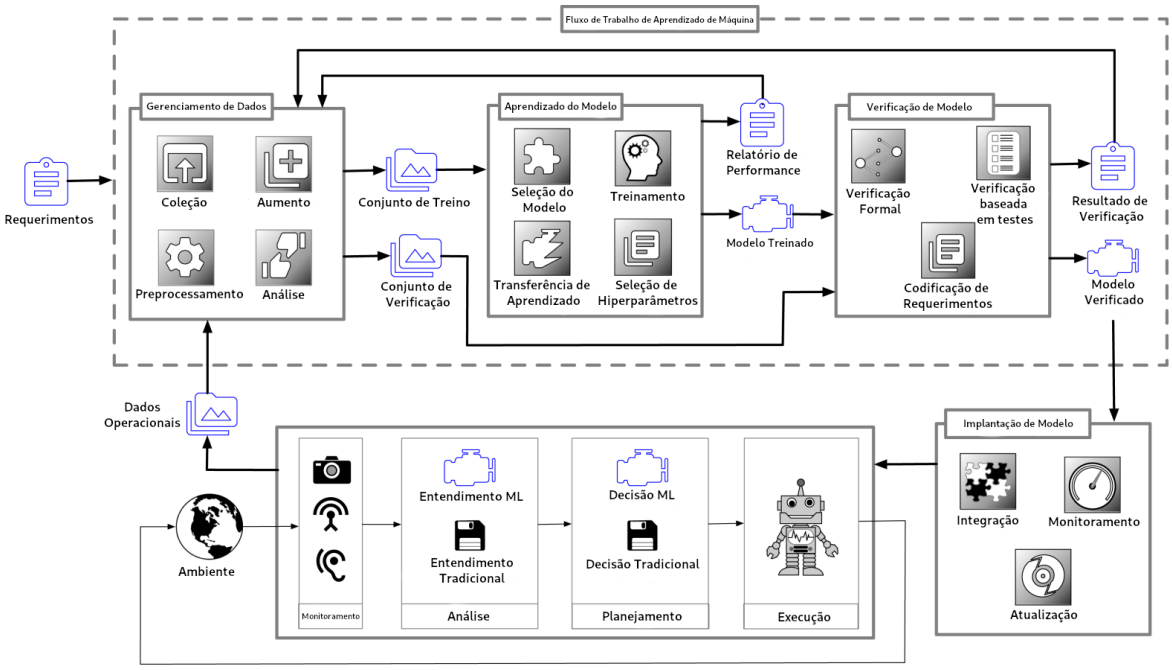

Figura 2.2: Estágios do ciclo de vida de projetos de *Machine Learning* (ML). Traduzido de Ashmore, Calinescu e Paterson (2021)

2.2.1 Gerenciamento de Dados

É o estágio que prepara os dados que serão usados para o treinamento do modelo. O gerenciamento de dados é uma das partes mais importantes do fluxo de MLOps, senão a mais importante. Segundo Sculley *et al.* (2014), as dependências relativas aos dados

[1] Trecho traduzido pelo autor, p. 8.

custam mais que aquelas condicionadas nos códigos. É preciso, portanto, ter cautela quanto aos prazos e a necessidade de melhoria nas métricas, tal que a qualidade dos dados coletados e processados não seja prejudicada. Nesse contexto, é importante reconhecer a existência de conflito no que tange à validação prévia dos dados (e.g., checar a precisão das anotações) em frente velocidade de experimentação (Shankar *et al.*, 2022).

A primeira etapa de um projeto de aprendizado de máquina já definido apropriadamente, isso é, uma solução para o problema foi proposta e acordada entre as partes interessadas (i.e., clientes, engenheiros etc.), é a aquisição de dados. O passo de aquisição de dados apresenta diversos desafios. Por exemplo, em certas ocasiões, os dados podem serem disponibilizados via algum servidor do cliente, enquanto em outros casos, pode ser necessário um *web scrapping*, isso é, um processo de coleta automática dados úteis da web para a implementação da solução.

Similarmente, algum tipo de pré-processamento (e.g., remover artefatos, como ruídos ou reflexos, que sejam prejudiciais ao aprendizado do modelo), anotação (e.g., rotular objetos de interesse e obter suas coordenadas em uma imagem) e aumento (e.g., criar novas amostras a partir de um conjunto pré-definido de transformações, como transformações no espaço de cores ou geométricas) de dados tendem a ser realizados.

Por fim, é comum que problemas possam aparecer em fases subsequentes, como um desempenho inadequado (e.g., baixa acurácia em sistemas de classificação) do modelo na etapa aprendizado e/ou verificação. Ferramentas que auxiliem a investigação e apontamento de onde exatamente está a causa da performance inadequada se tornam necessários. Nesse caso, até simples *scripts* de plotagem para avaliar a qualidade das anotações são úteis.

2.2.2 Aprendizado do Modelo

A segunda etapa de MLOps lida com o aprendizado do modelo. Pode-se dividi-lo nos passos de seleção de modelo, treinamento e otimização de hiper-parâmetros.

A seleção de modelos depende do contexto da tarefa a ser resolvida. Por exemplo, um modelo da linhagem de detectores de objetos YOLO pode ser escolhido para uma tarefa de detecção de objetos, pois tende a ter um *trade-off* nos termos de tempo de inferência e qualidade da detecção. Geralmente, um modelo simples (i.e., um modelo com um menor número de parâmetros) tende a ser escolhido para facilitar a experimentação e, consequentemente, diminuir o tempo de desenvolvimento. Outras considerações que podem ser feitas dizem respeito ao entendimento das predições, algo comum em aplicações cujo resultado pode ter implicações sérias (e.g., classificação de tumores em imagens médicas), o que tende a favorecer algoritmos com maior grau de interpretabilidade. Não é raro, portanto, que diferentes modelos sejam treinados antes que uma decisão final seja tomada quanto a escolha do modelo mais adequado como parte da solução da tarefa.

O treinamento do modelo selecionado é o passo seguinte. O *dataset*, que é a saída da etapa de gerenciamento de dados, alimenta o treinamento de um modelo previamente selecionado. É nesse passo que o modelo aprende os padrões representacionais nos dados para possivelmente solucionar o problema – fenômeno caracterizado pela convergência do modelo. Deve-se garantir que há recursos computacionais suficientes para o treinamento ser realizado, bem como ter cautela com tarefas cuja privacidade dos dados é importante.

A fase de otimização de hiper-parâmetros lida com o ajuste de um conjunto de parâmetros que definem o aprendizado do modelo. Por exemplo, o número de épocas, ou ciclos de treinamento, é um parâmetro que controla, em parte, o tempo de treinamento. A depender do algoritmo, tal hiper-parâmetro pode ser um diferencial na convergência do modelo.

Por fim, vale destacar que a experimentação e a iteratividade são importantes no processo de desenvolvimento de modelos de aprendizado de máquina por várias razões. Em primeiro lugar, permitem que os cientistas de dados e engenheiros de aprendizado de máquina testem e avaliem diferentes hipóteses e abordagens para resolver um problema específico. Isso pode ajudar a identificar as soluções mais eficazes e a evitar o risco de fazer suposições incorretas ou usar abordagens ineficazes. Além disso, a experimentação e a iteratividade permitem que os desenvolvedores ajustem e melhorem continuamente os modelos com base nos resultados obtidos, o que pode levar a modelos mais precisos e robustos.

2.2.3 Verificação do Modelo

A terceira etapa consiste em garantir que o modelo não só satisfaz os requerimentos funcionais do software, mas que também generalize para dados não vistos durante seu aprendizado. Especificamente, uma aplicação para OCR deve ser capaz de não só reconhecer os caracteres de interesse, mas também ir além das características intrínsecas daqueles caracteres que foram usados para o aprendizado. Nesse contexto, há três passos a tomar: codificação de requerimentos, verificação formal, e verificação baseada em testes.

No primeiro passo definem-se as métricas de performance do modelo, que idealmente devem ser traduzidas diretamente em valor ao negócio. No contexto de detecção de objetos, uma métrica importante é *Mean Average Precision* (MAP) para avaliar o *trade-off* entre a precisão e a cobertura do detector, todavia pode não ser a mais importante para uma tarefa de ALPR resolvida com modelos de detecção. A acurácia seria uma melhor escolha, pois a interpretação dada aos resultados do sistema poderiam ser melhor comunicadas às partes interessadas. Em outras palavras, considerando que a detecção dos códigos corretos é um acerto, então para uma acurácia hipotética de 90% pode-se dizer que a cada 10 veículos, 9 têm seus códigos corretamente identificados. A acurácia, portanto, provê informações diretamente relacionadas ao objetivo do sistema, algo que

precisão e cobertura não fazem por serem específicas aos objetos detectados.

A verificação formal, por sua vez, checa se o modelo realmente cumpre os requisitos estabelecidos em seu planejamento. Um modelo de predição de risco, por exemplo, pode ser analisado frente aos critérios definidos pelos entidades regulamentadoras, como interpretabilidade ou explicabilidade. Outras formas podem ser usadas, como a utilização de provas matemáticas de corretude, apesar de raro (Ashmore, Calinescu e Paterson, 2021).

Por fim, o modelo passa a ser testado em conjuntos de dados diferentes daquele usado para treinamento. Quanto mais próximo tais conjuntos são do mundo real, e diferente dos dados de treinamento, mais robusto é o teste. Nesse passo pode-se definir um limiar mínimo de aceitação para a métrica alvo.

2.2.4 Implantação do Modelo

A quarta e última etapa consiste de integrar o modelo na infraestrutura onde ele será usado para consumo por parte de usuários finais ou outros sistemas. É importante que o modelo seja constantemente monitorado para que qualquer indício de degradação de performance em novos dados seja identificado e o modelo seja atualizado. Além disso, pode acontecer que as etapas de verificação e implantação do modelo sejam acopladas e automatizadas em um sistema de integração e entrega contínua (CI/CD para sistemas de aprendizado de máquina), embora ainda existam desafios quanto ao gerenciamento simultâneo de código, dados e modelos.

Capítulo 3

Metodologia

O foco deste trabalho é apresentar os principais conceitos de MLOps. As próximas seções apresentam como as ferramentas foram utilizadas. Destaca-se que o presente trabalho não pretende justificar a escolha das ferramentas, visto que uma pesquisa comparativa a respeito de ferramentas de MLOps não é o foco neste trabalho. Para um trabalho contendo uma pesquisa comparativa sugere-se o trabalho apresentado em Souza (2021). Além disso, a instalação das ferramentas também não será abordada. Similarmente, não se pretende justificar o *pipeline* de ALPR escolhido. Após análise das amostras de dados e da literatura no ano de 2021, em especial o trabalho Montazzolli e Jung (2017), foi proposto uma solução de OCR de placas de caminhão em dois estágios: detecção de código e detecção de caracteres. Ambos os modelos foram desenvolvidos com o método representado na Figura 3.1.

Em resumo, para cada modelo, inicia-se com o gerenciamento dos dados necessários à solução da tarefa, que fica sob responsabilidade dos *scripts* internos (ver Seção 3.1.1) e da ferramenta DVC [1] (ver Seção 3.1.3), cujo propósito é permitir o versionamento de dados, automatizar o fluxo de trabalho de ML e gerenciar os experimentos. Posteriormente, os mesmos *scripts* auxiliam na preparação do conjunto de dados em uma estrutura adequada ao treinamento e validação do modelo, que nesse contexto é a rede neural YOLOv5. Artefatos de treino, como os pesos da rede ou métricas, podem ser versionados com o DVC. Por fim, as etapas de verificação e implantação de modelo são realizadas. A verificação usa ainda dos *scripts* internos para realização de *benchmarks* que produz métricas de performance sobre os dados de validação. Os resultados são versionados e visualizados no Studio [2] (ver Seção 3.1.4), uma aplicação web que simplifica o rastreio e visualização detalhada de experimentos. Caso a performance do modelo passe de um limiar pré-definido por uma entidade regulatória (90% de acurácia), pode-se promovê-lo para estágio de implantação, que consiste em integrá-lo na aplicação desenvolvida para a empresa parceira.

[1] `https://dvc.org/`
[2] `https://dvc.org/doc/studio`

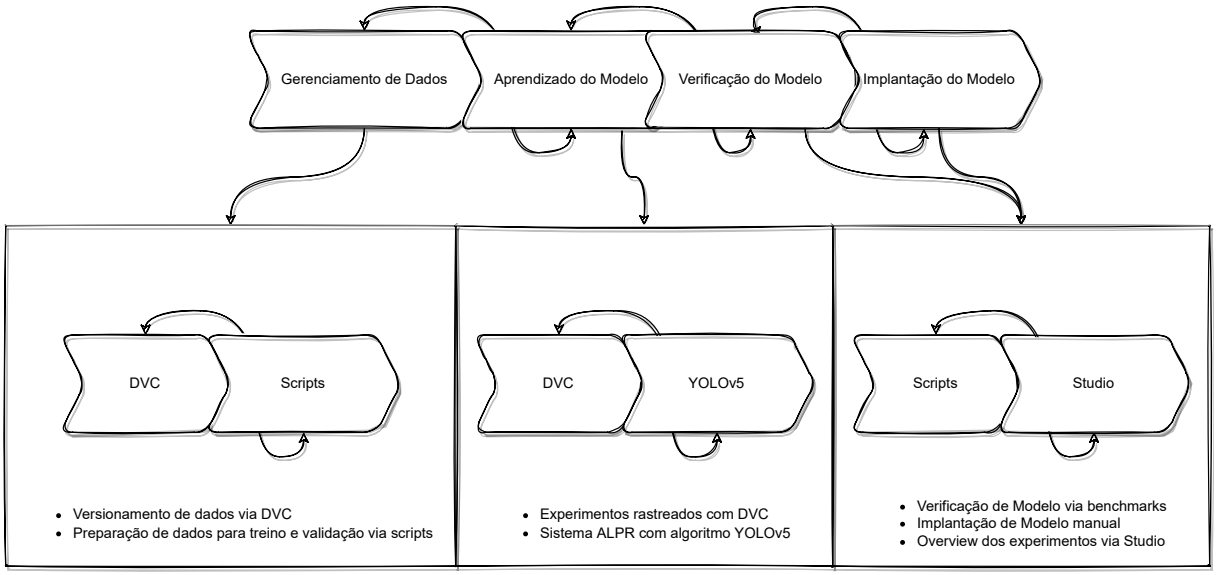

Figura 3.1: Estágios de MLOps e respectivas ferramentas utilizadas. Diagrama feito pelo autor.

3.1 Ferramentas de MLOps

As ferramentas utilizadas e aqui apresentadas são específicas para MLOps, todavia muitas outras secundárias também foram importantes no desenvolvimento da solução. Um exemplo é o pacote `pandas` [3], uma biblioteca comum para análise de dados tabulares, especialmente dados `.csv`. Neste trabalho, utilizou-se da biblioteca para avaliar resultados dos modelos no *benchmark,* para fins de análise de erros.

3.1.1 *Scripts*

Um *script* é um conjunto de comandos ou instruções que são executados por um interpretador de código ou por um compilador. Em geral, os *scripts* são utilizados para automatizar tarefas, realizar cálculos, acessar banco de dados ou redes, entre outras funções. Alguns exemplos de uso comuns de *scripts* são a automação de tarefas de administração de sistemas, a geração de relatórios, a integração de sistemas e a criação de aplicações web.

No contexto deste trabalho, os *scripts* são códigos que auxiliam o gerenciamento de dados e modelos, como a preparação do *dataset* em uma estrutura adequada para treinamento ou análises exploratórias, como a visualização das amostras, que pode ser usada para investigar a qualidade das amostras (e.g., se os caracteres da placa estão legíveis). A Tabela 3.1 mostra os *scripts* (como comandos) utilizados no projeto. Além disso, alguns *scripts* implementados internamente nas ferramentas escolhidas foram utilizados, como a exportação do modelo de detecção de objetos YOLOv5 para outras representações (e.g.,

[3] `https://pandas.pydata.org/`

extract	Realiza a extração dos *batches*.
gen-dataset	Prepara os *batches* extraídos em um *dataset*.
prepare-model	Converte um modelo PyTorch para o formato ONNX.
benchmark-csharp	Avaliar qualidade dos modelos em termos de acurácia.

Tabela 3.1: *Scripts* de preparação de dados, preparação de modelos e *benchmark*.

ONNX [4]).

3.1.2 PyTorch

PyTorch é uma biblioteca de tensores otimizada para aprendizado profundo usando CPUs e GPUs. PyTorch foi utilizado na implementação do modelo escolhido (YOLOv5).

3.1.3 *Data Version Control*

Uma ferramenta de código-aberto que colabora na solução de vários casos de uso, desde o versionamento de dados e modelos até o gerenciamento de experimentos via *pipelines* de dados – uma sequência de ações (ou fluxo de trabalho de dados) que se deseja reproduzir de forma segura. DVC permite capturar e gerenciar (armazenanado local ou remotamente, seja no Amazon S3, Google Cloud Storage ou outro) as diferentes versões dos dados ou modelos via metadados que são associados aos `commits` de um repositório.

Um `commit` é um registro de mudanças realizadas em um repositório de código fonte. Em sistemas de controle de versão, como o Git [5], os `commits` são utilizados para documentar as alterações realizadas no código fonte ao longo do tempo, permitindo a revisão e o rastreamento dessas mudanças. Cada `commit` possui um identificador único, uma mensagem de `commit` que descreve as alterações realizadas e informações sobre quem realizou as mudanças e quando elas foram feitas. Os `commits` são geralmente agrupados em *branches* (ramificações) ou em *tags* (marcações) para facilitar o gerenciamento do código fonte e a colaboração entre os desenvolvedores.

As vantagens da ferramenta incluem consistência e conformidade nos dados, isso é, os dados são os mesmos independente do local de uso, bem como ser grátis e de código-aberto. A consistência dos dados é importante quando diferentes estações de trabalho precisam ser usadas para treinamento ou análise exploratória, visto que tais atividades devem idealmente serem realizadas sobre os dados mais recentes. A Listagem 3.1 mostra um exemplo de versionamento de um dado qualquer (`data.xml`), que será enviado para um armazenamento externo à plataforma de hospedagem de código (e.g., Google Drive). O arquivo `data.xml.dvc`, por sua vez, representa `data.xml` e contém, internamente, os metadados (e.g., *hash* do tipo MD5) necessários ao controle de versão do dado original

[4] `https://onnx.ai/`
[5] `https://git-scm.com/`

(`data.xml`).

```
$ dvc add data.xml
$ tree
.
 data.xml
 data.xml.dvc

$ cat data.xml.dvc
outs:
  - md5: 6137cde4893c59f76f005a8123d8e8e6
    path: data.xml

$ file .dvc/cache/d8/acabbfd4ee51c95da5d7628c7ef74b
.dvc/cache/61/37cde4893c59f76f005a8123d8e8e6: ASCII text
```

Listagem 3.1: Exemplo de adição de dado. Adaptado da documentação DVC.

Neste trabalho, a ferramenta foi escolhida para gerenciar as mudanças nos dados, evitando problemas (e.g., perda ou inconsistência dos dados, caracterizada pela divergência dos dados em diferentes estações de trabalho), bem como encadear diferentes estágios em *pipelines* que representam o fluxo de desenvolvimento e implantação de um modelo. Um estágio DVC representa um comando individual, um *script*, ou código fonte que alcance determinado objetivo no fluxo de trabalho do projeto. A Listagem 3.2 mostra um exemplo de estágio de treinamento que envolve a execução de um comando (`python train.py`) que depende de um arquivo (`features.csv`) e produz as seguintes saídas: o modelo treinado (`model.pt`), um arquivo com a métrica de acurácia (`accuracy.json`) e um arquivo com dados para gerar um gráfico (`auc.json`).

```
stages:
  build:
    cmd: python train.py
    deps:
      - features.csv
    outs:
      - model.pt
    metrics:
      - accuracy.json:
          cache: false
    plots:
      - auc.json:
          cache: false
```

Listagem 3.2: Exemplo de estágio DVC. Retirado da documentação DVC.

Ao longo de um projeto de aprendizado de máquina, pode-se dividir o processo em etapas ou estágios que estabelecem os recursos de entrada (ou dependências), como um conjunto de dados, e os produtos de saída, como um modelo treinado. Um conjunto

de etapas configura um *pipeline* de dados. Os artefatos gerados (e.g., *dataset*, modelo treinado, entre outros) em cada estágio podem ser automaticamente versionados, o que permite uma melhor organização e reprodutibilidade de todo o fluxo. A maior parte das configurações estão em dois arquivos, `dvc.yaml` e `params.yaml`, que tratam dos estágios e seus parâmetros, respectivamente.

Alguns pontos negativos da ferramenta incluem complexidade, custo de aprendizado e dependência técnica. É necessário aprender a maior parte das funcionalidades da ferramenta para que o investimento tenha um retorno que compense o esforço humano de aprendizado. Nesse caso, é importante existir uma cultura de colaboração na equipe e organização para que o conhecimento seja compartilhado, evitando um débito técnico sobre o uso da ferramenta. No que tange o versionamento de dados, especificamente, o DVC não avalia a integridade dos dados, isso é, se foram corrompidos ou não. Dessa forma, é também necessário que a equipe inclua tais mecanismos em seus processos de MLOps. Apesar disso, dado que o DVC permite resgatar versões anteriores, esse problema é minimizado.

3.1.4 Iterative Studio

Studio [6] é uma aplicação web de código-fechado acessível de forma *online* ou *on premise* desenvolvida pela mesma empresa do DVC, a Iterative [7]. Juntamente com DVC, a aplicação simplifica a visualização, gerenciamento e compartilhamento dos experimentos, bem como permite a construção de um registro de modelos. É uma ferramenta grátis, mas tem funcionalidades pagas, como mais que dois colaboradores por time.

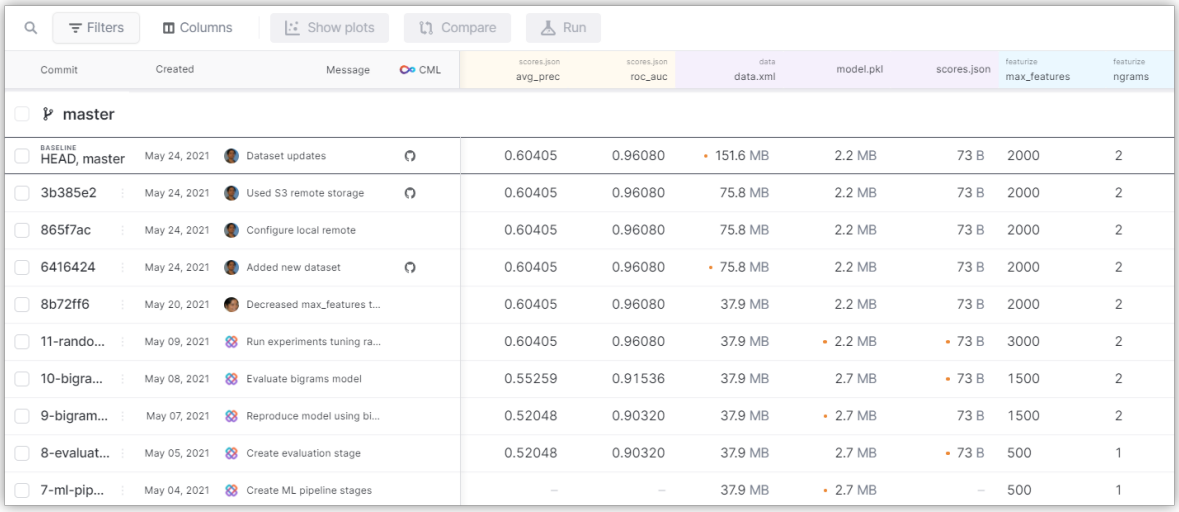

Figura 3.2: Interface principal do Studio para um projeto.

[6] `https://studio.iterative.ai/`
[7] `https://iterative.ai/`

Considerando que a configuração [8] do projeto na plataforma foi realizada, os *commits* enviados ao repositório do projeto (e.g., GitHub [9]) são obtidos pela aplicação, como pode ser visto na Figura 3.2. A interface principal mostra cada `commit` e as respectivas métricas, modelo, dados e hiper-parâmetros.

3.2 Banco de Dados

3.2.1 Aquisição de Dados

O processo de aquisição foi realizado através de um servidor *Server Message Block* (SMB) fornecido pela empresa parceira para que a coleta de imagens de caminhão fosse possível. Assim, através de uma *Virtual Private Network* (VPN) fornecida pela empresa parceira e implantada no local de trabalho do presente autor, foi possível realizar a obtenção das amostras. Cerca de 10 mil amostras formavam um lote, ou *batch*, e foram obtidas em um lote por vez até que todos os dados fossem coletados.

A base de dados original foi construída de abril até outubro de 2021 e compreende cerca de 125 mil imagens anotadas em um processo iterativo de anotação. O processo consistiu de um esquema manual, onde o colaborador rotulou objetos sem ajuda externa, e outro semi-automático, onde um modelo pré-treinado na tarefa (detecção de código ou de caracteres) auxilia o colaborador na rotulação – o colaborador começa a anotação com as predições do modelo para cada amostra. No presente trabalho apenas uma porção das imagens (aproximadamente 10.000) foi utilizada para os fins representativos do ferramental de MLOps (ver Seção 3.1).

3.2.2 Pré-processamento

Notou-se que parte das amostras apresentava uma faixa de metadados que não agregava ao aprendizado do modelo, logo foram removidas das imagens – a imagem original teve tal região cortada. Um exemplo pode ser visto na Figura 3.3.

3.2.3 Anotação de Dados

Considerando a solução proposta, os objetos a anotar foram placas de licença (considerando toda a imagem), e seus respectivos caracteres (considerando apenas a região da placa). Modelos pré-treinados (detector de placa e detector de caracteres) provindos de Montazzolli e Jung (2017) foram utilizados para auxiliar nas anotações da seguinte forma (ver Figura 3.4):

[8] `https://dvc.org/doc/studio/get-started`
[9] `https://github.com/`

Figura 3.3: Imagem de um caminhão cuja placa deve ser detectada e seus caracteres reconhecidos. Metadados na parte inferior, que é removido na fase de pré-processamento.

1. As amostras foram passadas à rede de detecção de placa (FV/LPD-NET) e suas coordenadas foram salvas;

2. As coordenadas foram usadas para recortar a placa da imagem original, que formaram, então, a entrada da rede de segmentação e reconhecimento de caracteres (LPR/CR-NET), cujos resultados foram as coordenadas de cada caractere, bem como o código textual da placa;

3. Por fim, as coordenadas da placa e dos caracteres, bem como o código textual, foram validados manualmente com o auxílio da ferramenta de anotação LabelMe[10].

Ao término do processo acima, um lote de anotações (cerca de 10.000 imagens) foi produzido e versionado com DVC. Pouco tempo depois do versionamento, um novo *dataset* foi gerado com auxílio dos *scripts* de preparação de dados em um estágio DVC, cujas dependências do estágio eram os lotes versionados e a sua saída, o *dataset* em si. Qualquer mudança posterior nos lotes poderia ser identificada pelo DVC, o que implica na geração de uma nova versão do *dataset*.

[10] `https://github.com/wkentaro/labelme`

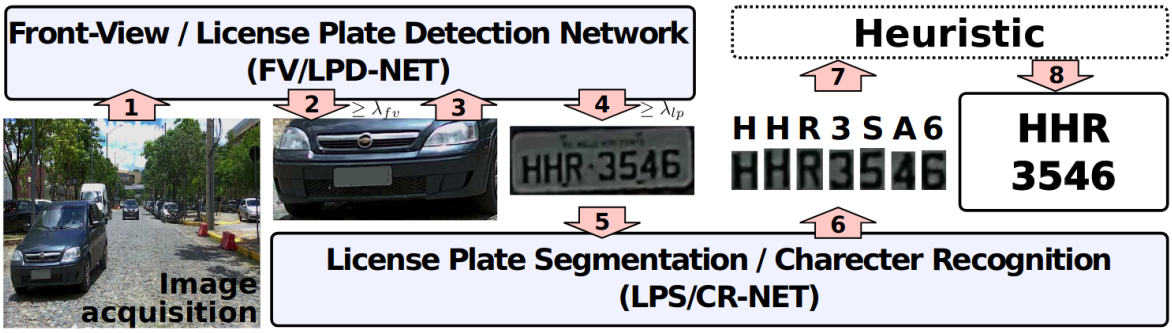

Figura 3.4: *Pipeline* proposto por Montazzolli e Jung (2017).

3.2.4 Aumento de Dados

As transformações realizadas foram responsabilidade dos *scripts* de treinamento associados à YOLOv5 [11]. A Listagem 3.3 apresenta os hiperparâmetros, seguido de seus valores padrão e uma breve explicação de cada um.

```
lr0: 0.01  # taxa de aprendizado inicial (SGD=1E-2, Adam=1E-3)
lrf: 0.01  # taxa de aprendizado final para a política OneCycleLR (lr0 *
  lrf)
momentum: 0.937  # SGD momentum/Adam beta1
weight_decay: 0.0005  # decaimento de peso do otimizador 5e-4
warmup_epochs: 3.0  # épocas de aquecimento (frações são ok)
warmup_momentum: 0.8  # momentum inicial na fase de aquecimento
warmup_bias_lr: 0.1  # taxa de aprendizado inicial para os viéses na fase
  de aquecimento
box: 0.05  # ganho na função de perda relativa às caixas
cls: 0.5  # ganho na função de perda relativa às classes
cls_pw: 1.0  # penalização relativa para falsos positivos nas classes (
  param. da BCELoss)
obj: 1.0  # ganho na função de perda relativa aos objetos (escala com os
  pixels)
obj_pw: 1.0  # penalização relativa para falsos positivos nos objetos (
  param. da BCELoss)
iou_t: 0.20  # limiar de IoU no treinamento
anchor_t: 4.0  # limite de múltiplos de âncoras
# anchors: 3  # âncoras por camada de saída (0 para ignora)
fl_gamma: 0.0  # parâmetro gamma da perda focal (efficientDet padrão gamma
  =1.5)
hsv_h: 0.015  # aumento de matiz na imagem (fração)
hsv_s: 0.7  # aumento de saturação na imagem (fração)
hsv_v: 0.4  # aumento de valor na imagem (fração)
degrees: 0.0  # rotação da imagem (+/- graus)
translate: 0.1  # translação da imagem (+/- fração)
scale: 0.5  # escala da image (+/- ganho)
```

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

```
shear: 0.0  # cisalhamentoo da imagem (+/- graus)
perspective: 0.0  # perspectiva da imagem (+/- fração), intervalo 0-0.001
flipud: 0.0  # probabilidade de virar a imagem de cima-baixo
fliplr: 0.5  # probabilidade de virar a imagem de direita-esquerda
mosaic: 1.0  # probabilidade de aplicar aumento "mosaic"
mixup: 0.0  # probabilidade de aplicar aumento "mixup"
copy_paste: 0.0  # probabilidade de aplicar aumento de segmentação "copy-
  paste"
```

Listagem 3.3: Hiperparâmetros e valores padrão.

3.3 Modelo de Aprendizado de Máquina

Inicialmente foi escolhido utilizar as redes disponibilizadas por Montazzolli e Jung (2017). No entanto, por questões relativas à licença do repositório e aos pesos da rede (o conjunto de dados usado para treino não permite uso comercial), foi decidido procurar outras arquiteturas similares. No fim, o modelo escolhido para compor o sistema de ALPR a ser desenvolvido foi o YOLOv5 (Jocher, 2020).

Os *scripts* de treinamento e exportação de modelos da YOLOv5 tornavam o o trabalho mais produtivo ao permitir uma boa velocidade de experimentação com diferentes variantes (`yolov5s`, `yolov5m`, `yolov5l` e `yolov5x`, entre outras) e hiperparâmetros. Além disso, os *benchmarks* de performance MAP no *dataset* de validação COCO 2017 [12] eram os melhores dentre os detectores de objetos do estado da arte da época.

3.4 Treinamento dos Modelos

Para este trabalho, a variante mais leve da YOLOv5 foi selecionada tanto para o modelo CD quanto CR (atualmente, o modelo `yolov5n`), e os hiperparâmetros de treinamento e aumento de dados foram os padrões já estabelecidos, exceto o tamamho das imagens, número de épocas e otimizador. O tamanho das imagens foi 512 para a tarefa de detecção de placas, e 256 para a tarefa de reconhecimento de caracteres. O número de épocas foi 10, visto que o objetivo deste trabalho foi mostrar a execução do *pipeline* de desenvolvimento do modelo, logo foi necessário uma rápida iteração. Por fim, o otimizador foi o AdamW pela rapidez em convergência do modelo. A divisão do conjunto de dados foi feita em 90/10 – um subconjunto de aproximadamente 9 mil amostras de treino e 1 mil amostras de validação. O restante dos hiperparâmetros estão na Listagem 3.4. Em `hyp`, por exemplo, estão os hiperpâmetros que podem ser otimizados via algoritmos genéticos.

```
weights: yolov5n.pt
cfg: ''
```

[12] `https://cocodataset.org/`

```
data: datasets/truck/cr/yolo/data.yaml
hyp:
  lr0: 0.01
  lrf: 0.1
  momentum: 0.937
  weight_decay: 0.0005
  warmup_epochs: 3.0
  warmup_momentum: 0.8
  warmup_bias_lr: 0.1
  box: 0.05
  cls: 0.5
  cls_pw: 1.0
  obj: 1.0
  obj_pw: 1.0
  iou_t: 0.2
  anchor_t: 4.0
  fl_gamma: 0.0
  hsv_h: 0.015
  hsv_s: 0.7
  hsv_v: 0.4
  degrees: 0.0
  translate: 0.1
  scale: 0.5
  shear: 0.0
  perspective: 0.0
  flipud: 0.0
  fliplr: 0.0
  mosaic: 1.0
  mixup: 0.0
  copy_paste: 0.0
epochs: 10
batch_size: 192
imgsz: 256
rect: false
resume: false
nosave: false
noval: false
noautoanchor: false
noplots: false
evolve: null
bucket: ''
cache: ram
image_weights: false
device: '0'
multi_scale: false
single_cls: false
optimizer: AdamW
```

```
sync_bn: false
workers: 8
project: outputs/truck/cr/
name: exp
exist_ok: true
quad: false
cos_lr: false
label_smoothing: 0.0
patience: 100
freeze:
- 0
save_period: -1
seed: 0
local_rank: -1
entity: null
upload_dataset: false
bbox_interval: -1
artifact_alias: latest
save_dir: outputs/truck/cr/exp
```

Listagem 3.4: Hiper-parâmetros de treinamento do modelo para *Code Recognition* (CR).

Similarmente ao estágio de preparação de dados, a etapa de treinamento também tem seu próprio estágio (`train`). Qualquer mudança no *dataset* (dependência do estágio de treinamento) seria identificada pelo DVC, que provocaria a necessidade de um novo treinamento do modelo.

3.5 Verificação e Implantação

Ao fim dos treinamentos, ambos os modelos foram convertidos para o formato ONNX (estágio de preparação do modelo, ou `prepare-model`) e utilizados em um *benchmark* (estágio de *benchmark*, ou `benchmark-csharp`) da aplicação desenvolvida junto à empresa parceira para avaliar se a acurácia de reconhecimento passava do limiar pré-definido definido pela Receita Federal do Brasil (RFB) – 90% de acurácia. A aplicação incorpora ambos os modelos para formar o sistema ALPR. Caso a acurácia passe do limiar, o modelo poderia ser implantado na (copiado para) aplicação e versionado tradicionalmente com git, senão o fluxo de aprendizado de máquina deveria ser reiniciado: avaliar erros de predição (e.g., placas ou caracteres difíceis de localizar e/ou classificar corretamente) e definir se a necessidade do momento será de mais amostras (tal que os dados de treinamento representem melhor as placas e caracteres encontradas em produção) ou melhores amostras (códigos mais legíveis do que aqueles atualmente existentes no *dataset*), ou retreino com novos hiperparâmetros.

Capítulo 4

Resultados e Discussões

Após a inicialização do DVC dentro do projeto – que é versionado por algum *Software Configuration Management* (SCM), como `git`, utilizada neste trabalho – alguns arquivos de configuração serão criados e devem ser versionados, como pode ser visto na Listagem 4.1.

```
$ dvc init
Initialized DVC repository.

You can now commit the changes to git.

...

$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
  new file:   .dvc/.gitignore
  new file:   .dvc/config
  new file:   .dvcignore
```

Listagem 4.1: Inicialização do repositório com DVC

Posteriormente, com as amostras já obtidas, o que deve ser feito é versioná-las. No contexto desse trabalho, as imagens e rótulos foram comprimidas em arquivos `zip`s para facilitar a transmissão inicial. O versionamento pode ser realizado com o comando `dvc add <target>`.

```
$ tree datasets
datasets
 cd
     batches
         andre_image_part_0.zip
```

```
6           andre_label_part_0.zip
7  cr
8        batches
9            andre_image_part_0.zip
10           andre_label_part_0.zip
11
12 4 directories, 4 files
13
14 $ dvc add datasets/cd/batches
15 100% Adding...
16 $ dvc add datasets/cr/batches
17 100% Adding...
```

Listagem 4.2: Adicionando *batches* de amostras com DVC

Similarmente à inicialização, alguns novos arquivos foram criados: são arquivos com extensão `.dvc` que representam os dados que deseja-se versionar fora do SCM, algo comum a depender de seu tamanho. Os arquivos `.dvc`, no entanto, precisam ser versionados normalmente no SCM, pois é com eles que o DVC consegue sincronizar os dados versionados externamente, isso é, fora do SCM. Após registrar as mudanças via `git commit` deve-se realizar sua submissão para o repositório com `git push`.

```
1  $ git status
2  On branch main
3
4  Your branch is up to date with 'origin/main'.
5
6  Changes to be committed:
7    (use "git restore --staged <file>..." to unstage)
8    new file:   datasets/cd/.gitignore
9    new file:   datasets/cd/batches.dvc
10   new file:   datasets/cr/.gitignore
11   new file:   datasets/cr/batches.dvc
```

Listagem 4.3: Visualizando arquivos `.dvc` para versionar com git

Como citado previamente, os dados reais (arquivos `zip`) também devem ser versionados – para que o versionamento possa ser feito com êxito, é preciso configurar com DVC um armazenamento local e/ou remoto. Note-se que utilização de todas as funcionalidades do Studio depende da configuração de um armazenamento remoto. Aqui foi escolhido o Google Drive, mas outros protocolos são possíveis, como ssh ou Amazon S3 [1]. Ver Listagem 4.4 para os comandos necessários. A mínima mudança nos dados reais, como uma mudança em qualquer linha de qualquer arquivo texto com as anotações ou alguma corrupção inesperada, é o suficiente para o DVC "marcar" tais dados como modificados. O estado dos dados versionados pode ser checado com o comando `dvc status`. Assim,

[1] `https://dvc.org/doc/command-reference/remote`

diferentes colaboradores do time de desenvolvimento sempre terão acesso a mesma versão dos dados.

```
$ dvc remote add -d gdrive gdrive://1JcUitcqVE9hjnzbdAULxtvKoPjv6DILb/dvcstore
Setting 'gdrive' as a default remote.
$ dvc push
4 files pushed
```

Listagem 4.4: Adicionado armazenamento remoto e versionando dados com `dvc push`

Com os dados reais já versionados, segue-se para o próximo passo do fluxo de desenvolvimento e implantação de modelos de aprendizado de máquina: preparação dos conjuntos de treinamento e validação. Aqui usam-se *scripts* auxiliares para extrair os arquivos `.zip` e, em seguida, criar a estrutura necessária ao treinamento [2]. A especificação desse estágio está na Listagem 4.5 e poder ser interpretada assim: para cada tarefa (e.g., dados CD, dados CR, etc.), use as dependências descritas em `deps` e execute o comando em `cmd` para produzir a saída especificada em `outs`.

```
stages:
  prepare-data:
    foreach: ${dataset.yolo.tasks}
    do:
      cmd: >-
        python company/cvt/tools/main.py
        extract
        --project company-${dataset.yolo.scope}-${item.task}
        --root-dir ${base.dataset-dir}/${dataset.yolo.scope}/${item.task}/batches
        --dst-dir ${base.dataset-dir}/${dataset.yolo.scope}/${item.task}/yolo
        --n-jobs ${base.n-jobs}
        gen-dataset
        --dataset yolo
        --validation-set-from ${base.dataset-dir}/${dataset.yolo.scope}/${item.task}/validation.csv
        --test-ratio ${dataset.yolo.test-set-ratio}
        --samples-from ${dataset.yolo.samples-from}
      deps:
      - ${base.dataset-dir}/${dataset.yolo.scope}/${item.task}/batches
      - ${base.dataset-dir}/${dataset.yolo.scope}/${item.task}/validation.csv
      outs:
        - ${base.dataset-dir}/${dataset.yolo.scope}/${item.task}/yolo:
            cache: false
```

Listagem 4.5: Definição do estágio de preparação de dados. Parte do arquivo de

[2] https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data#11-create-datasetyaml

configuração `dvc.yaml`.

Esse e outros estágios (`train`, `prepare-model`, `benchmark-csharp`) usam das capacidades do DVC para configuração baseada em *templating*, que permite inserir valores de diferentes fontes de configuração, e *Hydra composition*, que permite estruturar a configuração do projeto em diferentes fontes. Na Listagem 4.6 é apresentada a estrutura de fontes de configuração, enquanto que na Listagem 4.7 podem-se ver os parâmetros reais para o estágio. Todas as configurações na pasta `conf` são agregadas em um `params.yaml` no momento que um experimento é feito com `dvc exp run`. Pode-se perceber, portanto, a flexibilidade que a ferramenta permite ter caso seja necessário resolver a mesma tarefa (OCR) em imagens similares (e.g., vagões, containers etc.): basta criar novos arquivos de configuração. Caso o *pipeline* de dados seja abstrato o suficiente, todas as etapas serão realizadas sem necessidade de mudanças. Essa automatização de processos permite que os cientistas e engenheiros foquem em atividades mais críticas e manuais, como exploração de dados e experimentação com modelos.

```
$ tree conf -A
conf
 benchmark
      truck.yaml
 config.yaml
 dataset
      raw
          truck.yaml
      yolo
           truck.yaml
 train
       truck.yaml
```

Listagem 4.6: Estrutura de pastas com arquivos de configuração

```
scope: truck
samples-from:
test-set-ratio: 0

tasks:
  cd:
    task: cd
  cr:
    task: cr
```

Listagem 4.7: Parâmetros de `conf/dataset/yolo/truck.yaml` preparação do dataset YOLO

Posteriormente, pode-se modificar diretamente alguma das fontes ou modificar parâmetros rapidamente com `dvc exp run -S` – um novo `params.yaml` será criado para

garantir a reprodutibilidade do *pipeline* com `dvc repro`. Ao fim de um experimento é criado um arquivo chamado `dvc.lock`, que contém as *hashes* das dependências de cada estágio. Assim, o DVC consegue discernir quais estágios devem ser executados novamentes após checar suas dependências. A Listagem 4.8 mostra a saída desse comando quando uma execução já foi feita anteriormente.

```
$ dvc repro
'datasets/truck/cd/batches.dvc' didn't change, skipping
'datasets/truck/cd/validation.csv.dvc' didn't change, skipping
Stage 'prepare-data-raw@cd-val' didn't change, skipping
'datasets/truck/cr/batches.dvc' didn't change, skipping
'datasets/truck/cr/validation.csv.dvc' didn't change, skipping
Stage 'prepare-data-raw@cr-val' didn't change, skipping
Stage 'prepare-data@cd' didn't change, skipping
Stage 'train@cd' didn't change, skipping
Stage 'prepare-model@cd' didn't change, skipping
Stage 'prepare-data@cr' didn't change, skipping
Stage 'train@cr' didn't change, skipping
Stage 'prepare-model@cr' didn't change, skipping
Stage 'benchmark-csharp@validation' didn't change, skipping
Stage 'detect-ocr@validation' didn't change, skipping
Stage 'benchmark-ocr@validation' didn't change, skipping
Data and pipelines are up to date.
```

Listagem 4.8: Log de saída após `dvc repro` quando não há modificações nas dependências

Todos os estágios no DVC são encadeados e podem ser representados em um grafo acíclico dirigido. Dessa forma, e com `dvc.lock` que salva o *hash* das dependências, o DVC consegue analisar possíveis mudanças nas dependências de cada estágio e, se for o caso, omiti-los. A Figura 4.1 mostra uma visualização do *pipeline* de dados como grafos de dependência. É possível notar, por exemplo, que o estágio `benchmark-csharp@validation` (o `@` vem do atributo `foreach` do DVC, que permite um mesmo estágio ser realizado com diferentes parâmetros) depende de outros quatro estágios:

- `prepare-data-raw@cd-val`, que cria uma estrutura de pastas para os dados de validação com as imagens originais dos caminhões.
- `prepare-model@cd`, que copia o modelo *Code Detection* (CD) em formato PyTorch e o converte para formato `.onnx`.
- `prepare-data-raw@cr-val`, que cria uma estrutura de pastas para os dados de validação com as respectivas imagens das placas de cada caminhão.
- `prepare-model@cr`, que copia o modelo *Code Recognition* (CR) em formato PyTorch e o converte para formato `.onnx`.

Pode-se ir além e analisar as dependências de forma recursiva para cada um dos estágios citados. Ademais, é possível notar que o DVC, mais uma vez, tende a facilitar o desenvolvimento de soluções de ML, como a solução aqui apresentada para a tarefa de ALPR, ao garantir, por exemplo, que o estágio citado (`benchmark-csharp`) só será executado quando suas dependências forem satisfeitas: dois modelos diferentes, um detector de placa de veículos e um detector de caracteres de placa de licença. É preciso enfatizar: o engenheiro nada precisa fazer além da configuração inicial do *pipeline*. Esse *workflow* poderia ser melhorado com ferramentas de CD/CI (e.g., CML[3]), onde uma máquina de trabalho local ou na nuvem está à espera de um trabalho de treinamento, que é enviado automaticamente após modificações em alguma das dependências do estágio de treinamento (`train`) serem detectadas.

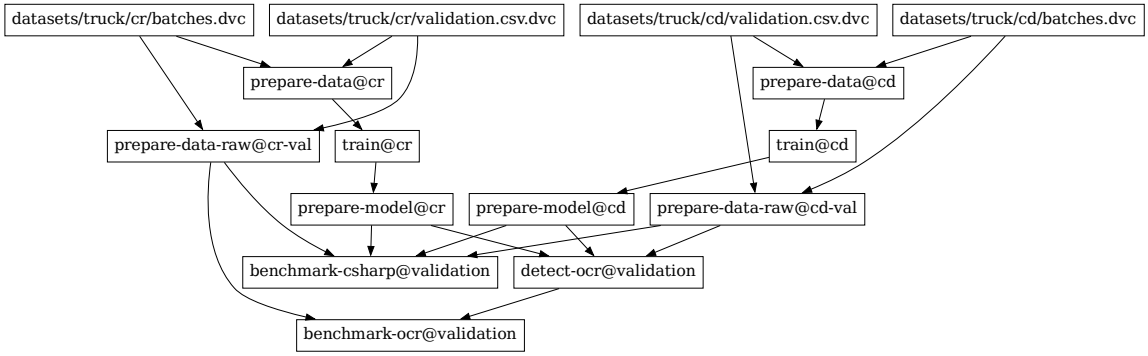

Figura 4.1: Grafos de dependência do *pipeline* do projeto. Diagrama exportado pela ferramenta DVC em conjunto com *scripts* internos.

Caso deseje-se realizar um novo experimento com algum parâmetro diferente, como uma mudança nos pesos da rede, pode-se executar a seguinte linha de comando na Listagem 4.9 – parte da saída foi omitida por ser demasiadamente extensa. Novamente o DVC checa por mudanças nas dependências e executa apenas os estágios necessários. Enquanto que no presente trabalho é mostrado apenas a variação de um único hiperparâmetro, pode-se imaginar ainda um diagrama mais complexo que o representado na Figura 4.1, com mais dependências e saídas. Dificilmente um ser humano seria capaz de lidar com tantas dependências e fluxos. Pior ainda em caso de mudanças em dependências, que muitas vezes podem passar despercebidas sem uma checagem robusta.

```
$ dvc exp run -S 'train.tasks.cd.weights=yolov5s.pt' -S 'train.tasks.cr.
    weights=yolov5s.pt'.
'datasets/truck/cr/batches.dvc' didn't change, skipping
'datasets/truck/cr/validation.csv.dvc' didn't change, skipping
Stage 'prepare-data@cr' didn't change, skipping
Running stage 'train@cr':
```

[3] `https://cml.dev/`

```
> python company/ocr/models/yolo/train.py  --batch-size 192  --cache ram
    --data datasets/truck/cr/yolo/data.yaml --device 0 --epochs 10  --hyp
    company/ocr/data/hyp/hyp_nofliplr.yaml --imgsz 256  --optimizer AdamW --
    patience 100 --project outputs/truck/cr/ --weights yolov5s.pt --exist-ok
     && rm datasets/truck/cr/yolo/labels/*.cache && sed -i '/^$/d;s/[[:blank
    :]]//g' outputs/truck/cr/exp/results.csv
# ... log de treinamento
Updating lock file 'dvc.lock'
Running stage 'prepare-model@cr':
> mkdir -p models/truck
> cp outputs/truck/cr/exp/weights/best.pt models/truck/cr.pt
> python company/ocr/models/yolo/export.py  --weights models/truck/cr.pt
    --include onnx  --opset 15 --simplify  --optimize  --imgsz 256  --
    dynamic
# ... log de preparação do modelo
Updating lock file 'dvc.lock'
# ... mais logs
Running stage 'benchmark-csharp@validation':
> dotnet run --dataset-cd-path datasets/truck/cd/raw/validation --dataset-
    cr-path datasets/truck/cr/raw/validation --heuristic-path dll/OCR.
    Heuristics/substitutions/truckSubs.json  --cr-full-or-cd-path models/
    truck/cd.onnx --cr-path models/truck/cr.onnx --project dll/OCR.
    Validation/OCR.Validation.csproj  --framework net5.0  --split-type 1 --
    version-db 1  --imgsz-cd 512 --imgsz-cr 256 --out-results-csv outputs/
    truck/benchmark-csharp/validation --out-results-metrics-json outputs/
    truck/benchmark-csharp/validation
acc@0.2: 0.96 | acc@0.3: 0.96 | acc@0.4: 0.96 | acc@0.5: 0.96 | acc@0.6:
    0.96 | acc@0.7: 0.94 | acc@0.8: 0.9 | acc@0.9: 0.04

Updating lock file 'dvc.lock'
```

Listagem 4.9: Linha de comando e log após execução de um novo experimento

A evolução dos artefatos, como dados, modelos e métricas, pode ser visualizada com a aplicação **Studio** na Figura 4.2 – outras formas são possíveis diretamente no terminal, como via `dvc metrics show`, `dvc plots` ou `dvc exp show`, que também tendem a ser úteis no *workflow* de experimentação. Nota-se um ganho de 31.39% de acurácia no experimento atual em relação ao anterior. A aplicação permite ver todos os parâmetros e identificar, como esperado, que a mudança entre os experimentos foi nos pesos (i.e., arquitetura `yolov5s` em vez da `yolov5n`) usados no treinamento. Apesar de visualizar-se apenas dois experimentos, pode-se imaginar a facilidade de análise caso fosse um número maior. Pode-se, ainda, criar outros *branches* e compará-los com o *branch* principal para isolar os experimentos ou o *baseline*, que normalmente é a *branch* principal.

É possível ainda realizar o registro de modelos de várias formas: usando do comando

`dvc add` (mais simples) para versionar os modelos ou com a ferramenta GTO [4]. Em outros momentos, caso houver necessidade de retornar um modelo já implantado para uma versão prévia, pode-se realizar o `dvc checkout` e passar o `commit` associado à versão do modelo. O uso da GTO facilita essas atividades, mas não está no escopo do trabalho. Além disso, pode-se realizar a geração de relatórios automáticos, CI/CD, entre outras ações. A verificação e implantação do modelo, portanto, são aqui apresentadas de forma semi-automática. A verificação em si aconteceu já na conversão do modelo para ONNX (estágio `prepare-model`) e na checagem de que a acurácia mínima (i.e., `acc@0.2`) no *benchmark* passou de 90%.

Os modelos de aprendizado de máquina foram integrados em uma aplicação CSharp. A integração foi realizada manualmente, copiando os modelos para a aplicação e criando uma nova versão da mesma. Apesar de o componente aqui apresentado ser relativo ao OCR de placas de caminhões, a arquitetura de software foi projetada para ser flexível e modular, para que no longo prazo novos componentes (e.g., vagão e container) fossem adicionados.

Por fim, considerando que caracteres semelhantes pudessem ser confundidos pelo OCR, como "0" e "O", foram desenvolvidas heurísticas (mapeamentos de letras para números e vice-versa) para mitigar os possíveis erros. Após a inferência, portanto, na fase de pós-processamento, verifica-se se uma letra ou dígito é esperado para uma dada posição da placa – considerando as regras de formação de uma placa de licença. Caso a detecção do sistema ALPR seja uma letra onde deveria ser um número, deve-se substituí-la pelo número apropriado. Similarmente para detecção de número onde deveria ser uma letra. Por exemplo, para uma placa cujo código correto é "ABC1234", mas que o sistema OCR reconheça "A8C1234", o caractere predito como "8" será substituído por um "B" porque na posição 2 do código não deve existir dígito.

Vale destacar que as ferramentas aqui utilizadas foram suporte para o desenvolvimento de um trabalho de inspeção de componentes eletrônicos em placas de circuitos impressos. O presente autor participou ativamente junto à equipe de pesquisa e tem publicação aceita, como primeiro autor, na 18ª VISAPP 2023[5], que acontecerá de 19 a 23 de Fevereiro de 2023. Vê-se, ainda, que a adoção de MLOps otimiza também a pesquisa dentro da indústria, pois remove as barreiras (e.g., que versão dos dados utilizados em um treinamento, que hiperparâmetros foram usados em outro etc.) que prejudicam o ciclo de experimentação de diferentes estratégias de aprendizado, um estágio fundamental na solução de problemas.

[4] `https://mlem.ai/doc/gto`
[5] `https://visapp.scitevents.org/Home.aspx`

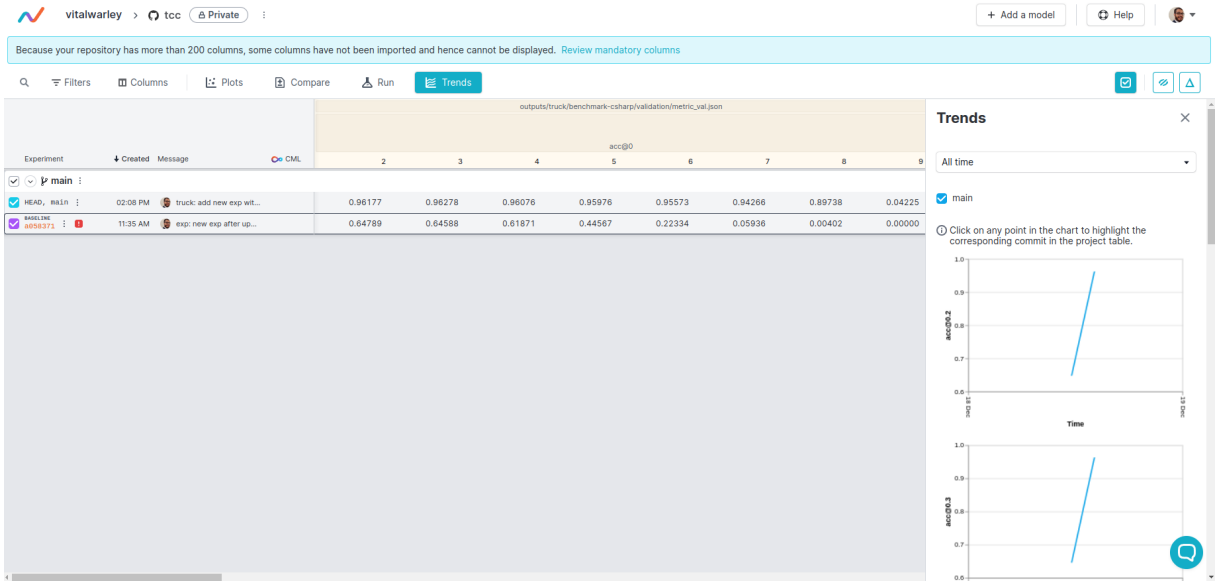

Figura 4.2: Interface da aplicação Studio mostrando métricas de diferentes experimentos. Pode-se notar uma tendência de um experimento para o outro, onde há uma melhora de 31.39% na métrica acurácia..

Capítulo 5

Conclusão

O presente trabalho introduz *Machine Learning Operations* (MLOps) através da solução da tarefa de *Automatic License Plate Recognition* (ALPR) com o treinamento e implantação de modelos de detecção de objetos. Os resultados apresentados exemplificam as principais etapas para implantação bem sucedida de soluções de aprendizado de máquina: gerenciamento de dados, aprendizado do modelo, verificação do modelo e implantação do modelo. Foram discutidas as vantagens em se utilizar das ferramentas que permitem a execução automática de todos estágios que compõem um projeto de aprendizado de máquina. Além disso, o trabalho mostra o que ainda pode ser feito para melhorar o *workflow* de MLOps no contexto profissional do autor.

O método proposto para solucionar um problema específico na indústria, através da utilização dos conceitos de MLOps, mostrou ser eficaz. Isso sugere que o seguimento de boas práticas nos processos de desenvolvimento e implantação de modelos de aprendizado de máquina pode trazer resultados positivos, especialmente em projetos de larga escala e com muitos colaboradores. O caminho a seguir é continuar a busca por mais automação dos processos manuais e, frequentemente, passíveis de erros. Dessa forma, as atividades mais criativas são priorizadas e, como resultado, tendem a ser realizadas de maneira mais eficiente pelos profissionais encarregados do fluxo de trabalho, incluindo cientistas, desenvolvedores e engenheiros.

Portanto, como sugestão de trabalhos futuros, ficou explícita a necessidade de sistematizar os outros passos do qual consiste o desenvolvimento de *Machine Learning Software Systems* (MLSS). Especificamente, pretende-se adicionar atividades de checagem de dados (e.g., analisar se anotações estão corretas), bem como automatizar o treinamento, verificação e implantação do modelo via *Continuous Integration and Continuous Delivery* (CI/CD). Dessa forma, vê-se também continuidade na exploração, estudo e uso de mais ferramentas, como MLEM [1], GTO e CML [2], para alcançar tais objetivos.

[1] `https://mlem.ai/`
[2] `https://cml.dev/`

Bibliografia

Aguiar, Carla Silva Rocha (2020). "Desafios na adoção de MLOps por time DevOps-projeto de co-desenvolvimento entre Governo e Academia para a introdução de e-gov 3.0". Em: *Anais Estendidos do XI Congresso Brasileiro de Software: Teoria e Prática*. SBC, pp. 144–147.

Antunes, Victor Wilvert *et al.* (2022). "Infraestrutura de monitoramento e operação para aprendizado de máquina". Em.

Ashmore, Rob, Radu Calinescu e Colin Paterson (2021). "Assuring the machine learning lifecycle: Desiderata, methods, and challenges". Em: *ACM Computing Surveys (CSUR)* 54.5, pp. 1–39.

Jiang, Peiyuan *et al.* (2022). "A Review of Yolo algorithm developments". Em: *Procedia Computer Science* 199, pp. 1066–1073.

Jocher, Glenn (mai. de 2020). *YOLOv5 by Ultralytics*. Versão 7.0. DOI: 10.5281/zenodo.3908559. URL: https://github.com/ultralytics/yolov5.

Kreuzberger, Dominik, Niklas Kühl e Sebastian Hirschl (2022). "Machine Learning Operations (MLOps): Overview, Definition, and Architecture". Em: *arXiv preprint arXiv:2205.02302*.

LeCun, Yann *et al.* (1989). "Handwritten Digit Recognition with a Back-Propagation Network". Em: *Advances in Neural Information Processing Systems*. Ed. por D. Touretzky. Vol. 2. Morgan-Kaufmann. URL: https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

Li, Chenxia *et al.* (2022). "PP-OCRv3: More Attempts for the Improvement of Ultra Lightweight OCR System". Em: *arXiv preprint arXiv:2206.03001*.

Mäkinen, Sasu *et al.* (2021). "Who needs MLOps: What data scientists seek to accomplish and how can MLOps help?" Em: *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE, pp. 109–112.

Matsui, Beatriz e Denise Goya (2022). "Conjunto de Artefatos para Representação de Boas Práticas em MLOps". Em: *Anais Estendidos do XIII Congresso Brasileiro de Software: Teoria e Prática*. Uberlândia/MG: SBC, pp. 32–35. DOI: 10.5753/cbsoft_estendido.2022.226912. URL: https://sol.sbc.org.br/index.php/cbsoft_estendido/article/view/22301.

Matsui, Beatriz Mayumi Andrade e Denise Hideko Goya (2021). "Applying DevOps to Machine Learning Processes: A Systematic Mapping". Em: *Anais do XVIII Encontro Nacional de Inteligência Artificial e Computacional*. SBC, pp. 559–570.

Montazzolli, Sérgio e Claudio Jung (2017). "Real-time brazilian license plate detection and recognition using deep convolutional neural networks". Em: *2017 30th SIBGRAPI conference on graphics, patterns and images (SIBGRAPI)*. IEEE, pp. 55–62.

Nogare, Diego, Rodrigo Fernandes Mello e Marco Antonio Lopes (2022). "Automação no processo de publicação de modelos de Ciência de Dados". Em: *Anais Estendidos do XIII Congresso Brasileiro de Software: Teoria e Prática*. SBC, pp. 40–43.

Paleyes, Andrei, Raoul-Gabriel Urma e Neil D Lawrence (2020). "Challenges in deploying machine learning: a survey of case studies". Em: *ACM Computing Surveys (CSUR)*.

Redmon, Joseph *et al.* (2016). "You only look once: Unified, real-time object detection". Em: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

Sculley, David *et al.* (2014). "Machine learning: The high interest credit card of technical debt". Em.

Shankar, Shreya *et al.* (2022). "Operationalizing Machine Learning: An Interview Study". Em: *arXiv preprint arXiv:2209.09125*.

Silva, Raul Ikeda Gomes da *et al.* (2021). "MLOps-Transformando Teoria em Prática". Em.

Souza, João Vitor Ramos de (2021). "Adoção de MLOps: desafios de gerenciar código, modelo e dados automaticamente". Em.

Weihong, Wang e Tu Jiaoyang (2020). "Research on License Plate Recognition Algorithms Based on Deep Learning in Complex Environment". Em: *IEEE Access* 8. Conference Name: IEEE Access, pp. 91661–91675. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2994287.

Weiner, Joyce (2020). "Why AI/data science projects fail: how to avoid project pitfalls". Em: *Synthesis Lectures on Computation and Analytics* 1.1, pp. i–77.