



UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

AtalaIA: Uso de Deep Learning para reconhecimento automático de placas de licença automotiva em dispositivos com recursos limitados

Trabalho de Conclusão de Curso

José Vitor Santos Silva



São Cristóvão – Sergipe

2022

UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

José Vitor Santos Silva

AtalaIA: Uso de Deep Learning para reconhecimento automático de placas de licença automotiva em dispositivos com recursos limitados

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Engenharia da Computação.

Orientador(a): Prof. Dr. Leonardo Nogueira Matos
Coorientador(a): Prof. Dr. Bruno Otávio Piedade Prado
e M.Sc Flávio Santos

São Cristóvão – Sergipe

2022

*Dedico este trabalho a minha família e amigos,
cujo incentivo me permitiu chegar até aqui.
Muito obrigado.*

Agradecimentos

Agradeço primeiramente a Deus, por ter me permitido realizar este trabalho. Agradeço também aos meus pais, José Jadson Sousa Silva e Cinelândia Alves dos Santos, por terem me dado todo o apoio necessário desde o meu nascimento. Também sou grato a minha namorada Adriann e a minha irmã Vitória, que sempre me incentivaram a continuar e valorizaram meu esforço.

Não poderia deixar de agradecer aos meus colegas de curso, com os quais compartilhei muitos momentos de alegria. Sempre me lembrarei com carinho das brincadeiras, trabalhos em grupo e apoio mútuo durante as semanas de prova.

Agradeço também ao meu orientador, Prof^o Dr. Leonardo Nogueira Matos, por todo apoio durante minha jornada acadêmica, desde a iniciação científica até o desenvolvimento deste trabalho. Por fim, agradeço aos meus co-orientadores M.Sc Flávio Santos e Prof^o Dr. Bruno Otávio Piedade Prado, pelo empenho durante as reuniões de orientação e sugestões sempre enriquecedoras.

*There are places I'll remember
All my life though some have changed
Some forever, not for better
Some have gone and some remain
All these places have their moments
With lovers and friends I still can recall
Some are dead and some are living
In my life I've loved them all
(The Beatles)*

Resumo

O uso de soluções baseadas em *deep learning* está cada vez mais presente no cotidiano das pessoas. Essas soluções são o componente principal em aplicações de diferentes domínios. Na visão computacional, o emprego das redes neurais convolucionais vem assumindo um papel de destaque, alcançando significativo sucesso em tarefas como classificação de imagens, segmentação semântica e detecção de objetos. Contudo, soluções baseadas em modelos profundos muitas vezes necessitam de uma quantidade significativa de recursos computacionais, por exemplo GPUs, para operar em níveis aceitáveis de latência. Isso encarece e restringe sua implantação em alguns cenários do mundo real. Além disso, o advento do 5G e a crescente demanda por dispositivos inteligentes e sistemas embarcados cujo poder de processamento é limitado, incentivam a criação de redes neurais adaptadas a dispositivos com limitação de memória, processador ou bateria. A combinação de modelos de aprendizagem profunda e sistemas com recursos limitados abre caminho para o surgimento de produtos comerciais mais baratos, democratizando o acesso à tecnologia e expandindo o impacto dela na sociedade. O presente trabalho apresenta o desenvolvimento de um sistema de reconhecimento automático de placas de licença automotiva (ALPR - *Automatic License Plate Recognition*) adequado a cenários do mundo real. O ALPR desempenha um importante papel em diversas aplicações práticas. Ele pode ser usado por agências governamentais para encontrar carros envolvidos em crimes, aplicação das leis de trânsito, controle de fronteira e monitoramento do tráfego rodoviário, também podendo ser usado por empresas privadas para fins comerciais, como cobrança automática de pedágio e controle de acesso a zonas privadas. O sistema ALPR desenvolvido neste trabalho utilizou como base modelos de detecção de objetos específicos para cada uma das etapas de reconhecimento: detecção do veículo, detecção da placa e detecção dos caracteres. Para o desenvolvimento do sistema, construímos uma base de dados original de 5029 imagens de 326 veículos distintos, a qual intitulamos UFS-ALPR. Utilizando a base de dados construída, realizamos experimentos com cinco diferentes arquiteturas: YOLOv4 Tiny, YOLOv5 Nano, YOLOv5 Small, SSD e CenterNet. Posteriormente, analisamos o desempenho dos modelos treinados e embarcamos o sistema proposto em dois dispositivos de processamento limitado: Raspberry Pi 4 e NVIDIA Jetson Nano. A versão convencional do sistema ALPR proposto foi capaz de reconhecer corretamente 87,58% das placas da base de dados UFS-ALPR. Já a versão embarcada no Raspberry Pi 4 e no Jetson Nano reconheceu corretamente 77,05% e 77,51%, há uma taxa de 0,53 FPS e 5,66 FPS, respectivamente.

Palavras-chave: ALPR, *deep learning*, *edge computing*, *object detection*.

Abstract

The use of deep learning based solutions is increasingly present in people's daily lives. These solutions are the key component of applications from different domains. In computer vision, the use of convolutional neural networks has assumed a prominent role, achieving significant success in tasks such as image classification, semantic segmentation, and object detection. However, solutions based on deep models often require a significant amount of computational resources, for example GPUs, to operate at acceptable levels of latency. This makes them expensive and restricts their deployment in some real-world scenarios. In addition, the advent of 5G and the growing demand for intelligent devices and embedded systems, whose processing power is limited, encourage the creation of neural networks adapted to devices with limited memory, processor, or battery. The combination of deep learning models and resource-constrained systems paves the way for cheaper commercial products, democratizing the access to technology and expanding its impact on society. In this context, this work presents the development of an automatic license plate recognition system (ALPR) suitable for real-world scenarios. The ALPR plays an important role in many practical applications. It can be used by government agencies to find cars involved in crime, traffic law enforcement, border control, and road traffic monitoring, and it can also be used by private companies for commercial purposes such as automatic toll collection and access control to private zones. The ALPR system developed in this work was based on specific object detection models for each of the recognition steps: vehicle detection, license plate detection, and character detection. For the development of the system, we built an original database of 5029 images of 326 different vehicles, which we named UFS-ALPR. Using the built database, we performed experiments with five different architectures: YOLOv4 Tiny, YOLOv5 Nano, YOLOv5 Small, SSD and CenterNet. Subsequently, we analyzed the performance of the trained models and embedded the proposed system in two limited processing devices: Raspberry Pi 4 and NVIDIA Jetson Nano. The conventional version of the proposed ALPR system was able to correctly recognize 87.58% of the license plates in the UFS-ALPR database. The embedded version in the Raspberry Pi 4 and Jetson Nano correctly recognized 77.05% and 77.51%, with 0.53 FPS and 5.66 FPS, respectively.

Keywords: ALPR, *deep learning, edge computing, object detection.*

Lista de ilustrações

Figura 1 – Diagrama esquemático da operação de convolução utilizando um <i>kernel</i> 3x3	21
Figura 2 – Resultados anuais na competição ImageNet, evidenciando o abandono do uso dos métodos tradicionais a partir do sucesso da AlexNet em 2012	21
Figura 3 – Arquitetura geral de uma CNN para classificação de imagens	23
Figura 4 – Arquitetura da AlexNet	23
Figura 5 – Exemplo de segmentação. Acima imagem de entrada e abaixo o resultado da segmentação	24
Figura 6 – Arquitetura geral de uma rede <i>autoencoder</i>	25
Figura 7 – Arquitetura U-Net	25
Figura 8 – Exemplo de detecção de objetos	26
Figura 9 – Representação gráfica do cálculo da interseção sobre união	27
Figura 10 – <i>Pipeline</i> da R-CNN	30
Figura 11 – Arquitetura geral da Fast R-CNN	31
Figura 12 – Arquitetura geral da Faster R-CNN	31
Figura 13 – Esquema de funcionamento da YOLOv1	33
Figura 14 – YOLO rodando em imagens de arte e em imagens da natureza obtidas da Internet	33
Figura 15 – Variações da YOLOv5	34
Figura 16 – Comparação entre a arquitetura da YOLOv1 e da SSD	36
Figura 17 – Arquitetura da CenterNet	37
Figura 18 – Exemplo de uma arquitetura geral com o uso de transfer learning. As camadas do fim são treinadas em um novo domínio e o conhecimento do modelo no domínio em que foi originalmente treinado é aproveitado através dos pesos das camadas iniciais	37
Figura 19 – Exemplos de placas de licença capturadas em condições desafiadoras	38
Figura 20 – Modelos de Raspberries	40
Figura 21 – NVIDIA Jetson Nano	41
Figura 22 – Diagrama de blocos do K210	42
Figura 23 – Módulo MAIX-I	43
Figura 24 – MAIX Go Development Board	43
Figura 25 – Rekor Edge	45
Figura 26 – <i>Pipeline</i> proposta por (LAROCA et al., 2018)	48
Figura 27 – Arquitetura do modelo proposto por Gonçalves et al. (2018) para detecção de placas	50
Figura 28 – Arquitetura do modelo proposto por Gonçalves et al. (2018) para reconhecimento dos caracteres	51

Figura 29 – <i>Pipeline</i> proposta por Laroca et al. (2019)	53
Figura 30 – Placas de diferentes <i>layouts</i> usadas por Laroca et al. (2019)	54
Figura 31 – Exemplos de amostras da base de dados UFPR-ALPR	57
Figura 32 – Amostras da base de dados SSIG-ALPR	58
Figura 33 – Interface de rotulação do Dataturks ¹	59
Figura 34 – Amostras da base de dados UFS-ALPR	60
Figura 35 – Distribuição de veículos por tipo	61
Figura 36 – Distribuição de veículos por fabricante	61
Figura 37 – Distribuição de veículos por modelo	62
Figura 38 – Ocorrências dos caracteres	63
Figura 39 – <i>Heat map</i> das placas	64
Figura 40 – <i>Heat map</i> dos veículos	64
Figura 41 – Exemplo de entrada e saída do estágio de detecção do veículo	66
Figura 42 – Exemplo de entrada e saída do estágio de detecção da placa	67
Figura 43 – Tipos de placas brasileiras	67
Figura 44 – Exemplo de entrada e saída do estágio de detecção e reconhecimento dos caracteres	68
Figura 45 – Exemplos de amostras usadas para treinar os modelos de cada uma das etapas do sistema ALPR	71
Figura 46 – Exemplo do processamento utilizado para aumentar a partição de treinamento. Na esquerda, a imagem original e na direita a imagem negativa gerada	71
Figura 47 – Saídas dos modelos que detectam veículos para uma amostra do <i>dataset</i> UFS-ALPR	76
Figura 48 – Saídas dos modelos que detectam placas para uma amostra do <i>dataset</i> UFS-ALPR	77
Figura 49 – Saídas dos modelos que detectam caracteres para uma amostra do <i>dataset</i> UFS-ALPR	78
Figura 50 – Amostras de imagens capturadas na entrada de veículos da UFS	82
Figura 51 – Raspberry Pi usado nos experimentos	84
Figura 52 – Jetson Nano usado nos experimentos	85

Lista de tabelas

Tabela 1 – Comparação de desempenho da Faster R-CNN, YOLOv1 e SSD no <i>dataset</i> Pascal VOC2007	35
Tabela 2 – Comparação dos sistemas comerciais Plate Recognizer, Sighthound e OpenALPR	46
Tabela 3 – Resultados obtidos por Laroca et al. (2018) e pelos sistemas comerciais Sighthound e OpenALPR no <i>dataset</i> UFPR-ALPR	48
Tabela 4 – Resultados da abordagem proposta por Gonçalves et al. (2018) no <i>dataset</i> SSIG-SegPlate	51
Tabela 5 – Resultados do sistema proposto por Gonçalves et al. (2018) no <i>dataset</i> UFPR-ALPR (desconsiderando imagens de motos)	52
Tabela 6 – Resultados obtidos por Laroca et al. (2019) nos oito <i>datasets</i> usados, juntamente com o desempenho de dois sistemas comerciais.	54
Tabela 7 – Dados sobre as partições de treinamento e teste criadas para a base de dados UFS-ALPR	70
Tabela 8 – Métricas dos modelos YOLOv4 Tiny treinados no <i>dataset</i> UFS-ALPR	72
Tabela 9 – Métricas dos modelos YOLOv5 Nano treinados no <i>dataset</i> UFS-ALPR	73
Tabela 10 – Métricas dos modelos YOLOv5 Small treinados no <i>dataset</i> UFS-ALPR	73
Tabela 11 – Métricas dos modelos SSD treinados no <i>dataset</i> UFS-ALPR	74
Tabela 12 – Métricas dos modelos CenterNet treinados no <i>dataset</i> UFS-ALPR	75
Tabela 13 – Métricas dos modelos treinados para detecção de veículos	75
Tabela 14 – Métricas dos modelos treinados para detecção de placas	77
Tabela 15 – Métricas dos modelos treinados para detecção e reconhecimento de caracteres	78
Tabela 16 – Métricas do estágio de detecção do veículo para cada uma das arquiteturas experimentadas no conjunto de teste	79
Tabela 17 – Métricas do estágio de detecção da placa para cada uma das arquiteturas experimentadas no conjunto de teste	79
Tabela 18 – Métricas do estágio de detecção dos caracteres para cada uma das arquiteturas experimentadas no conjunto de teste	80
Tabela 19 – Métricas das dez melhores combinações de modelos, os dados são do conjunto de teste	80
Tabela 20 – Métricas das dez melhores combinações de modelos que atingiram mais de 10 FPS, os dados são do conjunto de teste	81
Tabela 21 – Comparação das métricas do sistema ALPR em um computador convencional e no Raspberry Pi 4	85
Tabela 22 – Comparação das métricas do sistema ALPR em um computador convencional e no Jetson Nano	86

Lista de abreviaturas e siglas

ALPR	<i>Automatic License Plate Recognition</i>
AP	<i>Average Precision</i>
API	<i>Application Programming Interface</i>
APU	<i>Audio Processing Unit</i>
AUC	<i>Area Under Curve</i>
CUDA	<i>Compute Unified Device Architecture</i>
FCN	<i>Fully Convolutional Networks</i>
FPS	<i>Frames por segundo</i>
FPU	<i>Floating Point Unit</i>
GPU	<i>Graphic Processing Unit</i>
IOU	<i>Intersection Over Union</i>
ILSVRC	<i>ImageNet Large-Scale Visual Recognition Challenge</i>
IoT	<i>Internet of Things</i>
KPU	<i>Knowledge Processing Unit</i>
MCU	<i>Microcontroller Unit</i>
LSTM	<i>Long Short Term Memory</i>
MCU	<i>Microcontroller Unit</i>
mAP	<i>Mean Average Precision</i>
MVP	<i>Minimum Viable Product</i>
OCR	<i>Optical Character Recognition</i>
ONNX	<i>Open Neural Network Exchange</i>
PNG	<i>Portable Network Graphics</i>
RAM	<i>Random Access Memory</i>
R-CNN	<i>Region-based Convolutional Neural Network</i>

RPN	<i>Region Proposal Network</i>
RNN	<i>Recurrent Neural Network</i>
SoC	<i>System On Chip</i>
SSD	<i>Single Shot Detector</i>
SVM	<i>Support Vector Machine</i>
YOLO	<i>You Only Look Once</i>

Sumário

1	Introdução	16
1.1	Motivação	17
1.2	Objetivos	18
1.2.1	Objetivos Específicos	18
1.3	Metodologia	18
1.4	Estrutura do Documento	19
2	Conceitos Básicos	20
2.1	Redes Neurais Convolucionais (CNNs)	20
2.1.1	Aplicações das CNNs na Visão Computacional	22
2.1.1.1	Classificação de imagens	22
2.1.1.1.1	AlexNet	23
2.1.1.2	Segmentação semântica	24
2.1.1.2.1	U-Net	25
2.1.1.3	Detecção de objetos	26
2.1.1.3.1	Métricas para avaliação de modelos de detecção de objetos	27
2.1.1.3.2	Modelos de detecção de objetos baseados em proposta de regiões (R-CNN, Fast R-CNN e Faster-RCNN)	29
2.1.1.3.3	YOLO - You Only Look Once	32
2.1.1.3.4	SSD - Single Shot Detector	34
2.1.1.3.5	CenterNet	35
2.1.2	Transferência de aprendizado	37
2.2	<i>Automatic Licence Plate Recognition (ALPR)</i>	38
2.3	Dispositivos para computação na borda	39
2.3.1	Raspberry Pi	39
2.3.2	NVIDIA Jetson Nano	40
2.3.3	Kendryte K210	41
3	Trabalhos Relacionados	44
3.1	Produtos Comerciais	44
3.1.1	Plate Recognizer	44
3.1.2	Sighthound	45
3.1.3	OpenALPR	45
3.1.4	Análise Comparativa	46
3.2	Trabalhos Acadêmicos	46

3.2.1	<i>A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector</i>	46
3.2.2	<i>Real-time Automatic License Plate Recognition Through Deep Multi-Task Networks</i>	49
3.2.3	<i>An Efficient and Layout-Independent Automatic License Plate Recognition System Based on the YOLO detector</i>	52
3.2.4	Outros trabalhos	54
4	Construção da Base de Dados	56
4.1	Outras bases brasileiras para ALPR	56
4.1.1	UFPR-ALPR	56
4.1.2	SSIG-ALPR	57
4.2	Levantamento, preparação e rotulação dos dados	58
4.3	Estatísticas	60
5	Desenvolvimento e Resultados	65
5.1	Sistema ALPR proposto	65
5.1.1	Cenário de uso do sistema	65
5.1.2	Pipeline de reconhecimento	66
5.1.3	Escolha dos modelos	68
5.2	Treinamento e avaliação dos modelos	69
5.2.1	Preparação da base de dados	70
5.2.2	YOLOv4 Tiny	70
5.2.3	YOLOv5 Nano e YOLOv5 Small	73
5.2.4	SSD e CenterNet	74
5.2.5	Comparação dos modelos	75
5.3	Sistema ALPR	78
5.3.1	Definição dos modelos para compor o sistema ALPR embarcado	81
5.4	Implantação do sistema ALPR em dispositivos com recursos limitados	83
5.4.1	Raspberry Pi 4	83
5.4.1.1	Especificação do hardware	83
5.4.1.2	Configuração do ambiente e otimização dos modelos	83
5.4.1.3	Métricas do sistema	84
5.4.2	NVIDIA Jetson Nano	85
5.4.2.1	Especificação do hardware	85
5.4.2.2	Configuração do ambiente e otimização dos modelos	85
5.4.2.3	Métricas do sistema	86
6	Considerações Finais	87

Referências	89
------------------------------	-----------

Apêndices	96
------------------	-----------

APÊNDICE A Métricas das combinações de modelos testadas no sistema ALPR .	97
--	-----------

1

Introdução

Um sistema de reconhecimento automático de placas de licença automotiva, do inglês *automatic license plate recognition* (ALPR), utiliza uma imagem ou uma sequência delas para extrair informações da placa de um veículo. Isso é atingido por meio da combinação de diversas técnicas de processamento de imagens, detecção de objetos e reconhecimento de padrões. Essa tarefa também pode ser chamada de identificação automática de veículos, reconhecimento de placas de automóveis ou reconhecimento óptico de caracteres (OCR - *Optical Character Recognition*) para carros (DU et al., 2012).

A identificação automática placas de licença automotiva desempenha um importante papel em diversas aplicações práticas do mundo real. Ela pode ser usada por agências governamentais para encontrar carros envolvidos em crimes, aplicação das leis de trânsito, controle de fronteira e monitoramento do tráfego rodoviário. Diversos países como Estados Unidos, Japão, Alemanha, Itália, Reino Unido e França já aplicam com sucesso o uso de sistemas ALPR no gerenciamento do trânsito (JIN et al., 2012). O ALPR também pode ser usado por empresas para fins comerciais, como cobrança automática de pedágio e controle de acesso a zonas privadas. Por conta disso, uma vasta gama de sistemas desse tipo foi produzida ao longo dos anos (DU et al., 2012).

Nos últimos anos, os avanços no ALPR estão intimamente ligados ao uso de soluções baseadas em aprendizagem de máquina (*machine learning*) (LAROCA et al., 2019; GONÇALVES et al., 2018). Esse tipo de abordagem vem alcançando resultados promissores e assumindo um papel de destaque que, é importante salientar, não se limita apenas aos sistemas de reconhecimento de placas automotivas.

A crescente disponibilidade de bases de dados e a evolução dos recursos de hardware são alguns dos motivos que propiciaram o avanço do aprendizado de máquina. Esses fatores fizeram com que surgissem máquinas de aprendizado que realizam suas tarefas tão bem quanto os humanos (FUCHS et al., 2021; HE et al., 2015; MANINIS et al., 2016). As redes neurais profundas obtiveram significativo sucesso em diferentes áreas, destacando-se as redes neurais

convolucionais, *convolutional neural networks* (CNNs), que são uma das soluções mais bem sucedidas em problemas de visão computacional (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; SZEGEDY et al., 2015) e as redes neurais recorrentes, *recurrent neural networks* (RNNs), que são amplamente usadas em problemas com dados sequenciais, tais como: reconhecimento de fala (GRAVES; MOHAMED; HINTON, 2013), tradução simultânea (SUTSKEVER; VINYALS; LE, 2014) e processamento de linguagem natural (HERMANN et al., 2015).

No contexto do ALPR, por se tratar de um problema de visão computacional, trabalhos recentes têm explorado o uso de CNNs, em especial aquelas para detecção de objetos. Um exemplo é a família de modelos YOLO - You Only Look Once (REDMON et al., 2016; REDMON; FARHADI, 2017; REDMON; FARHADI, 2018; BOCHKOVSKIY; WANG; LIAO, 2020), que estão entre os mais rápidos modelos profundos para detecção de objetos, sendo capazes de atingir taxas de inferência em tempo real em algumas situações. Em geral, nesse tipo de abordagem, diferentes modelos são treinados para cada estágio da *pipeline* de reconhecimento, como feito por Laroca et al. (2019).

1.1 Motivação

Apesar dos avanços alcançados com o uso do aprendizado profundo na visão computacional, inclusive nos sistemas ALPR, muitas das soluções existentes ainda não são suficientemente robustas para cenários reais. Isso se dá principalmente devido à maioria delas necessitarem de uma quantidade significativa de recursos computacionais, como por exemplo GPUs (*Graphic Processing Units*) para operar em níveis aceitáveis de latência, o que encarece e restringe a implantação de aplicações baseadas em modelos profundos. Por se tratar de um problema desafiador, com diversas variações que diminuem o desempenho do reconhecimento, o ALPR exige o uso de modelos profundos complexos. Além disso, muitas das soluções existentes usam mais de um modelo, um para cada etapa do sistema, o que aumenta ainda mais a necessidade de processamento. O desenvolvimento de soluções com boa relação entre acurácia, custo computacional e tempo de resposta é de suma importância para a viabilização de aplicações desse tipo.

No aprendizado de máquina, os dados desempenham papel fundamental, pois é a partir deles que os modelos aprendem a reconhecer padrões. Por outro lado, eles são também uma barreira. Segundo Cognilytica (2019), em média 80% do tempo gasto em projetos de *machine learning* é destinado a tarefas envolvendo dados, sendo a maior parte disso gasto na rotulação. No desenvolvimento de sistemas ALPR baseados em máquinas de aprendizado, isso também é um desafio. São poucas as bases de dados apropriadamente rotuladas (com rótulos para o veículo, a placa e para os caracteres) disponíveis à comunidade acadêmica, especialmente se considerarmos somente placas brasileiras. Portanto, a construção de bases de dados fomenta a pesquisa não só de sistemas ALPR, mas do aprendizado de máquina como um todo (SUN et al., 2017).

A crescente demanda por dispositivos inteligentes e sistemas embarcados que operam, na maioria das vezes, em tempo real e cujo poder de processamento é limitado, exerce crescente influência para criação de redes neurais adaptadas a dispositivos com limitação de memória, processador ou bateria (SILVA et al., 2021). A combinação de modelos de aprendizagem profunda e sistemas com recursos limitados abre caminho para o surgimento de produtos comerciais mais baratos, democratizando o acesso à tecnologia e expandindo o impacto dela na sociedade. Isso aceleraria a adoção de tecnologias desse tipo em diversos campos, como na computação de borda (*edge computing*).

1.2 Objetivos

O objetivo geral deste trabalho é propor e desenvolver um sistema ALPR adequado a aplicações do mundo real. Durante o desenvolvimento serão exploradas técnicas de aprendizado profundo, especialmente modelos para detecção de objetos, que serão a base da *pipeline* de reconhecimento. Por fim, o sistema proposto será avaliado em relação a requisitos de velocidade de processamento e acurácia.

1.2.1 Objetivos Específicos

- Construir uma base de dados para ALPR;
- Explorar estratégias e treinar modelos atuais que utilizem técnicas de aprendizagem profunda na base construída;
- Implementar a *pipeline* de reconhecimento de placas automotivas utilizando os modelos treinados, bem como avaliar o desempenho do sistema;
- Embarcar o sistema em dispositivos de processamento limitado.

1.3 Metodologia

A metodologia empregada neste trabalho contempla três fases, que são:

1. Construção de uma base de dados para ALPR: Essa etapa envolve a coleta, preparação e rotulação de imagens de veículos. Serão coletadas imagens do trânsito em diferentes locais no ambiente urbano das cidades sergipanas Aracaju e São Cristóvão. Em seguida, serão selecionadas as imagens em que a placa dos veículos está legível e cada veículo será rotulado em relação a tipo (carro, moto, ônibus ou caminhão), marca, modelo, padrão de placa (antigo ou MERCOSUL) e posição do veículo (frontal ou traseira). Por fim, as *bounding boxes* do veículo, da placa e de cada caractere dela serão anotadas utilizando software de rotulação;

2. Desenvolvimento do sistema ALPR: Nessa etapa, algoritmos de *deep learning* serão explorados na base construída. Para tal, serão treinados modelos do estado da arte para detecção de objetos. Esses modelos serão a base da *pipeline* de reconhecimento, que será implementada e avaliada em relação a requisitos de velocidade de processamento e acurácia;
3. Implantação do sistema embarcado: O sistema construído será embarcado em dispositivos com processamento limitado. O objetivo dessa etapa é reduzir os custos de um possível MVP - *Minimum Viable Product* baseado no uso do sistema proposto. Os ambientes de dispositivos de borda serão configurados e os modelos serão convertidos para formatos mais otimizados para esses dispositivos.

1.4 Estrutura do Documento

Este documento está estruturado em capítulos, que são:

- **Capítulo 2 - Conceitos Básicos:** A fundamentação teórica do projeto é apresentada, visando prover as informações necessárias para o entendimento do trabalho;
- **Capítulo 3 - Trabalhos Relacionados:** As tecnologias comercializadas atualmente e o desenvolvimento acadêmico relacionado ao tema são apresentados, tendo como objetivo destacar a relevância do projeto;
- **Capítulo 4 - Construção da Base:** Este capítulo apresenta o detalhamento do processo de levantamento, preparação e rotulação da base de dados, bem como algumas estatísticas acerca dela;
- **Capítulo 5 - Desenvolvimento e Resultados:** Apresenta o desenvolvimento do projeto proposto, descrevendo ferramentas, técnicas, métodos e algoritmos. Também é feita a exposição dos resultados alcançados;
- **Capítulo 5 - Considerações Finais:** Visa apresentar as considerações finais acerca do trabalho, destacando as contribuições, limitações e sugestões para trabalhos futuros.

2

Conceitos Básicos

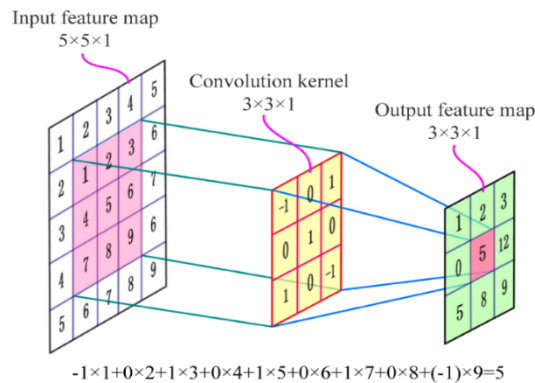
Este capítulo apresenta a fundamentação teórica necessária para entendimento do trabalho. A Seção 2.1 descreve as CNNs, destacando suas diferentes aplicações, bem como a introdução de alguns modelos e métricas. A Seção 2.2 apresenta o ALPR, seus desafios e principais aspectos desse problema. Por fim, na Seção 2.3 são introduzidos dispositivos com recursos limitados usados para processamento de aplicações em computação de borda.

2.1 Redes Neurais Convolucionais (CNNs)

Uma rede neural convolucional é uma rede neural artificial que aplica uma operação de convolução aos dados. Essa operação envolve uma matriz de pesos, chamada de filtro ou *kernel*, e gera como saída outra matriz, chamada de mapa de características (*feature map*). A convolução percorre a matriz de entrada, aplicando operações de multiplicação e soma. Como mostrado na Figura 1, as multiplicações são elemento por elemento (*elementwise*) e são entre o *kernel* e a matriz de entrada, o *kernel* desliza por toda a imagem, gerando o *feature map* de saída. A operação é similar à convolução empregada em processamento de sinais e, por isso, o modelo se chama rede neural convolucional (GOODFELLOW; BENGIO; COURVILLE, 2016).

Os filtros são responsáveis por fazer a extração de características. Eles identificam curvas, linhas, formas e outros padrões. Normalmente os filtros são inicializados de forma aleatória e são aprendidos durante o treinamento pela rede neural. O tamanho do *kernel* e do passo entre as multiplicações (*stride*) são alguns dos parâmetros desse tipo de rede. Conforme a rede fica mais profunda, características mais complexas são extraídas, camadas iniciais detectam formas, curvas, linhas e padrões simples, enquanto as mais internas detectam características como olhos, rostos e outros tipos de padrões.

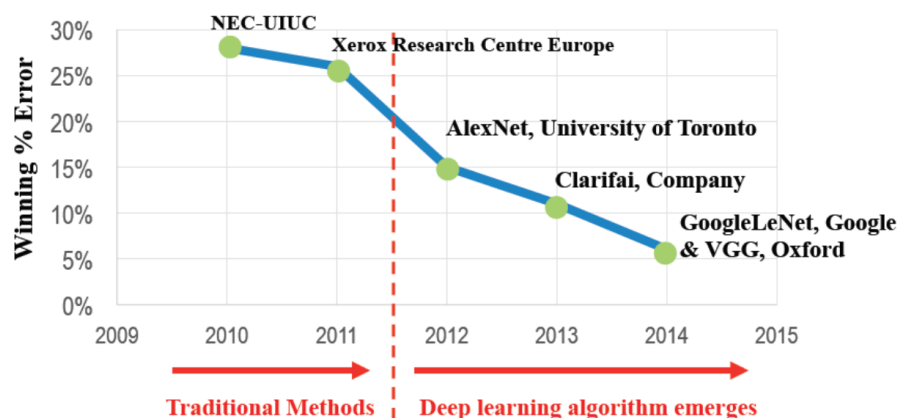
As CNNs foram inicialmente propostas por LeCun et al. (1998), porém, apenas mais recentemente, em 2012, que esse tipo de modelo se popularizou. Isso se deu devido ao sucesso

Figura 1 – Diagrama esquemático da operação de convolução utilizando um *kernel* 3x3

Fonte: [Gong et al. \(2019\)](#)

de uma CNN no desafio anual *ImageNet Large-Scale Visual Recognition Challenge* (ILSVRC) ([RUSSAKOVSKY et al., 2015](#)). A arquitetura em questão, intitulada AlexNet ([KRIZHEVSKY; SUTSKEVER; HINTON, 2012](#)), representou um grande avanço em relação aos anos anteriores e mudou completamente o rumo da competição nos anos seguintes, estimulando a utilização de CNNs em detrimento dos métodos tradicionais. A Figura 2 evidencia a revolução causada pela AlexNet. Antes de 2012 os métodos vencedores usavam *features* específicas e *support vector machines* (SVMs) ([LIN et al., 2010](#); [PERRONNIN; SANCHEZ, 2011](#)), a partir de 2012 arquiteturas de CNNs dominaram a competição ([SIMONYAN; ZISSERMAN, 2014](#); [SZEGEDY et al., 2015](#); [HE et al., 2016](#)).

Figura 2 – Resultados anuais na competição ImageNet, evidenciando o abandono do uso dos métodos tradicionais a partir do sucesso da AlexNet em 2012



Fonte: [Shawahna, Sait e El-Maleh \(2018\)](#)

Numa rede neural convolucional, as primeiras camadas recebem a imagem (ou matriz) de entrada e executam nela convoluções usando múltiplos *kernels*, que geram conjuntos de mapas

de características. Cada um desses mapas carrega informações de algum tipo de padrão, como intensidade e localização de uma característica específica. Além disso, as CNNs comumente possuem camadas que aplicam operações de subamostragem, chamadas de *pooling* (*max pooling* ou *average pooling*, por exemplo). Essas operações envolvem um *kernel* que representa uma janela da matriz de entrada. O *max pooling* retorna o valor máximo da janela sendo analisada, enquanto o *average pooling* retorna o valor médio. Essas operações reduzem a dimensionalidade dos mapas de características, com o intuito de diminuir o custo computacional e suprimir ruídos (BIEDER; SANDKÜHLER; CATTIN, 2021). Numa CNN, existem ainda camadas de ativação. Essas ativações introduzem não-linearidades ao modelo e servem como uma maneira de ativar ou desativar determinados neurônios da rede. São exemplos de funções de ativação a ReLu, Sigmoid e Tanh.

2.1.1 Aplicações das CNNs na Visão Computacional

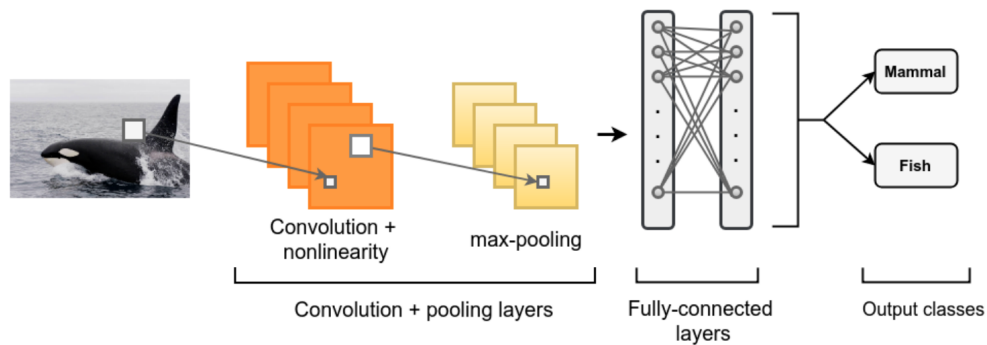
Com o uso do treinamento supervisionado e uma grande quantidade de amostras, as CNNs podem aprender a mapear padrões complexos em mapas de características úteis (JØRGENSEN, 2017). Por conta disso, nos últimos anos as CNNs se destacaram como uma das soluções mais bem sucedidas em tarefas de visão computacional, firmando-se como o estado da arte em problemas como classificação de imagens (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; SZEGEDY et al., 2016; HUANG et al., 2017; HE et al., 2016), detecção de objetos (BOCHKOVSKIY; WANG; LIAO, 2020), segmentação semântica, reconhecimento óptico de caracteres (WOJNA et al., 2017) e ALPR (LAROCA et al., 2019; JØRGENSEN, 2017).

2.1.1.1 Classificação de imagens

O objetivo da classificação de imagens é atribuir à imagem uma classe específica, dentre um conjunto de classes previamente estabelecido. Normalmente, a classificação de imagens se refere a imagens em que apenas uma entidade aparece e é analisada (WANG; SU, 2019).

Em geral, as redes neurais convolucionais para esse tipo de tarefa são estruturadas em duas partes principais, vide Figura 3. A primeira parte consiste em um conjunto de camadas convolucionais, ativações e camadas de *pooling*. Essa parte atua como um extrator de características e é responsável por mapear a imagem de entrada em *feature maps* com dimensões reduzidas, de modo que sejam conservadas as informações mais relevantes da imagem de entrada. A segunda parte recebe os mapas da última camada convolucional e atua como um classificador. Essa etapa consiste em camadas que aplicam uma operação linear. Elas podem ser denominadas camadas lineares, densas ou totalmente conectadas (*fully connected layers*).

Figura 3 – Arquitetura geral de uma CNN para classificação de imagens

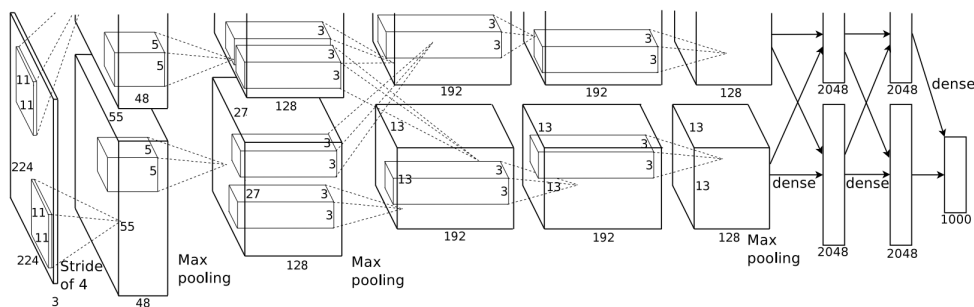


Fonte: Santos et al. (2019)

2.1.1.1.1 AlexNet

A AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) foi a arquitetura vencedora do *ImageNet Large-Scale Visual Recognition Challenge* (ILSVRC) em 2012. Sua arquitetura, vide Figura 4, contém oito camadas ao todo: as cinco primeiras são camadas convolucionais e as três restantes são camadas lineares totalmente conectadas. A saída da última camada linear é passada para função softmax, que produz uma distribuição de probabilidades para os 1000 rótulos das classes da base de dados do desafio ImageNet.

Figura 4 – Arquitetura da AlexNet



Fonte: Krizhevsky, Sutskever e Hinton (2012)

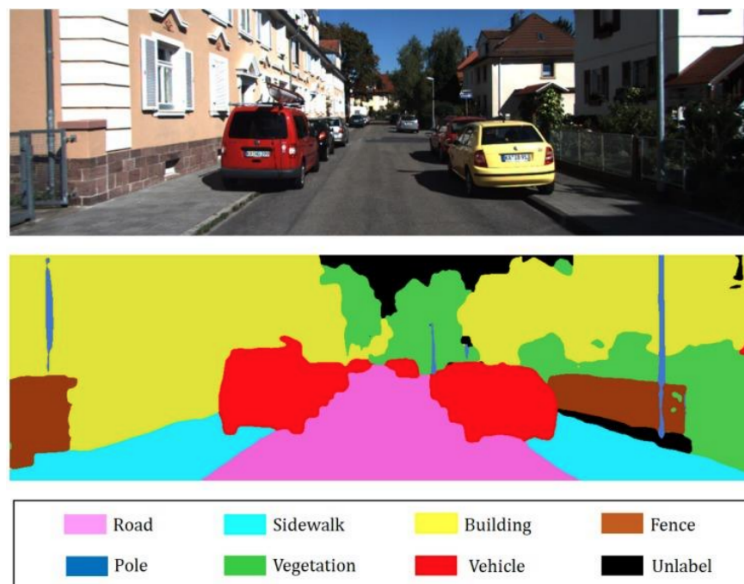
A AlexNet é um modelo bastante complexo, possuindo 60 milhões de parâmetros treináveis. Em modelos profundos tão complexos é comum a rede ficar especializada nos dados de treinamento, caracterizando o sobreajuste (*overfitting*). O *overfitting* é um problema comum nas redes neurais e muitas técnicas foram propostas com o objetivo de mitigar seus efeitos. Esse problema ocorre quando a rede fica especializada nos dados de treinamento, mas com um desempenho ruim nos dados de validação e de teste, indicando que a rede não consegue operar em dados nunca antes vistos e tem uma baixa capacidade de generalização. Durante o treinamento dos modelos para o desafio ImageNet, os criadores da AlexNet utilizaram *dropout*, *weight decay*

e *data augmentation* com o objetivo de mitigar os efeitos do *overfitting*. A AlexNet obteve no desafio ILSVRC de 2012 taxas de erro top-1 e erro top-5 de 37,5% e 17% respectivamente (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

2.1.1.2 Segmentação semântica

A segmentação semântica consiste na tarefa de realizar a rotulação de todos os pixels de uma imagem de acordo com classes específicas. Ao invés de mapear todo o conjunto de pixels que forma uma imagem em uma única classe (classificação tradicional), a segmentação semântica mapeia cada pixel isoladamente. A Figura 5 exemplifica esse processo.

Figura 5 – Exemplo de segmentação. Acima imagem de entrada e abaixo o resultado da segmentação



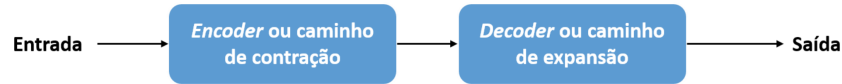
Fonte: Jeong, Yoon e Park (2018)

Uma abordagem recente e bem-sucedida para o problema de segmentação semântica é o uso de *fully convolutional networks* (FCNs) (LONG; SHELHAMER; DARRELL, 2015). As FCNs, como o nome sugere, possuem apenas camadas convolucionais, o que permite que uma imagem de tamanho arbitrário produza um mapa de segmentação do mesmo tamanho. Long, Shelhamer e Darrell (2015) modificaram arquiteturas populares usadas para classificação: VGG (SIMONYAN; ZISSERMAN, 2014), GoogLeNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), substituindo as camadas lineares por camadas convolucionais. Desse modo, a saída dos modelos modificados passou a ser mapas de segmentação, ao invés dos *scores* das classificações.

Estendendo a ideia das FCNs, uma família popular de modelos profundos para segmentação de imagens são as redes baseadas na arquitetura *encoder-decoder*. A arquitetura geral desse

tipo de rede (que também são chamadas de *autoencoders*) está ilustrada na Figura 6 e possui duas partes principais:

Figura 6 – Arquitetura geral de uma rede *autoencoder*



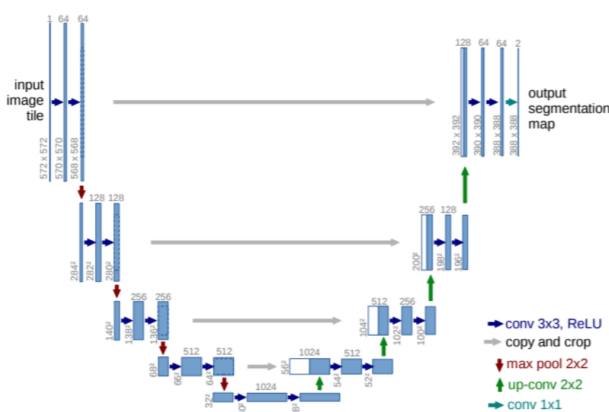
Fonte: Elaborada pelo autor

- *Encoder*: Consiste em um bloco de camadas de convolução e *downsampling*. Os mapas de características gerados por esse ramo possuem resolução espacial bem menores que a imagem original.
- *Decoder*: Consiste em um bloco de camadas de convolução transposta e *upsampling*. Nesse ramo da rede, os *feature maps* gerados pelo encoder são expandidos para a resolução inicial da imagem.

2.1.1.2.1 U-Net

O modelo U-Net (RONNEBERGER; FISCHER; BROX, 2015) é uma boa ilustração da arquitetura *encoder-decoder*. Uma importante característica da U-Net é que o *decoder* é quase simétrico em relação ao *encoder* dando à arquitetura o formato de um “U”, como pode ser visto na Figura 7. A U-Net é particularmente popular para segmentação de imagens médicas e é capaz de combinar as informações contextuais obtidas do caminho de contração com as informações de localização adquiridas do caminho de expansão de maneira eficiente (CAVALCANTE, 2019).

Figura 7 – Arquitetura U-Net

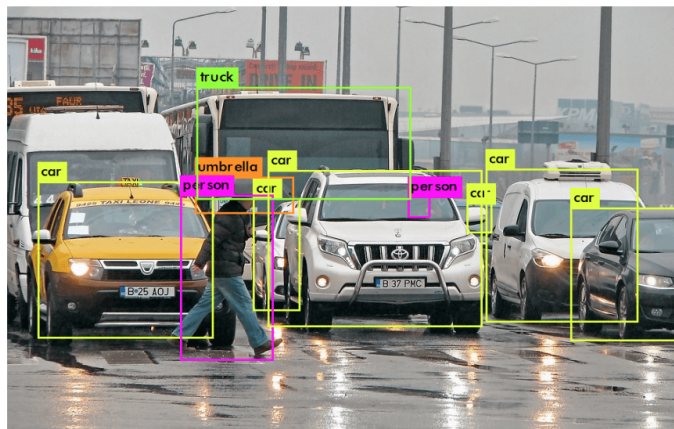


Fonte: Ronneberger, Fischer e Brox (2015)

2.1.1.3 Detecção de objetos

Estendendo a ideia da classificação de imagens, um problema mais complexo é a classificação com localização. Nele, além de saber o que o objeto é (predição da classe), também queremos localizar o objeto na imagem. Para isso, é feita a predição da *bounding box*, isto é, as coordenadas do retângulo que envolve o objeto. Contudo, essa abordagem se limita a um objeto por imagem. A detecção de objetos é a tarefa de classificar e localizar múltiplos objetos dentro de uma imagem, como ilustrado na Figura 8.

Figura 8 – Exemplo de detecção de objetos



Fonte: Potdar, Pai e Akolkar (2018)

A detecção de objetos pode ser encarada como um problema de classificação de zonas específicas de uma imagem. Seguindo essa ideia, algumas abordagens pioneiras para detecção de objetos aplicavam modelos de classificação em janelas que deslizavam pela imagem. Contudo, essa estratégia é extremamente custosa, uma vez que cada imagem tem que passar várias vezes pelo modelo de classificação. Além disso, as janelas precisam ter tamanhos fixos, o que gera detecções não muito precisas. Por conta disso, os algoritmos de janela deslizante foram abandonados em favor de outras abordagens.

Um modelo moderno de detecção de objetos é normalmente composto por duas partes: o *backbone*, que é uma CNN que processa a imagem de entrada, e a cabeça, que é responsável por fazer a predição das classes e das *bounding boxes* dos objetos. A cabeça pode ser categorizada em detectores de objetos de um ou de dois estágios. Os detectores de objeto de um estágio fazem a predição da classe e da *bounding box* no mesmo momento. Os detectores de objetos de dois estágios separam a tarefa de classificação e localização para cada *bounding box* (BOCHKOVSKIY; WANG; LIAO, 2020). São exemplos de detectores de dois estágios os algoritmos baseados na proposta de regiões (R-CNN (GIRSHICK et al., 2014), Fast R-CNN (GIRSHICK, 2015), Faster R-CNN (REN et al., 2015)). Já representando os detectores de objetos de um estágio, temos a YOLO (REDMON et al., 2016; REDMON; FARHADI, 2017; REDMON; FARHADI, 2018;

BOCHKOVSKIY; WANG; LIAO, 2020), SSD (LIU et al., 2016), CenterNet (DUAN et al., 2019) e RetinaNet (LIN et al., 2017) como exemplos.

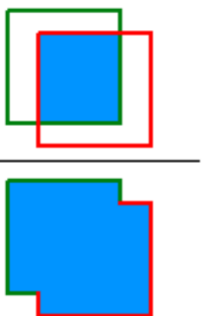
2.1.1.3.1 Métricas para avaliação de modelos de detecção de objetos

IOU - Interseção sobre união

A IOU - *Intersection Over Union*, ou interseção sobre união, é uma métrica usada para medir a acurácia de um detector de objetos em um conjunto de dados específico. É uma métrica bastante popular usada em desafios de detecção de objetos, como o Pascal VOC (EVERINGHAM et al., 2007).

Para ser computada, a IOU precisa da *bounding box* correta (*ground truth*) e da *bounding box* gerada pelo modelo durante a inferência. Uma vez que temos essas duas *bounding boxes*, como o nome da métrica sugere, basta dividirmos a área de interseção pela área de união entre ambas. A Figura 9 apresenta graficamente o cálculo da IOU.

Figura 9 – Representação gráfica do cálculo da interceção sobre união

$$IOU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{área de interseção}}{\text{área de união}}$$


Fonte: Padilla, Netto e Silva (2020)

Quando as duas *bounding boxes* são exatamente iguais, as áreas de interseção e de união são iguais, logo, a IOU vale um. Para avaliarmos o desempenho de um modelo em um *dataset*, basta calcularmos a IOU média. Quanto mais próxima de um, melhores são as predições do modelo. É comum estabelecer limiares baseados na IOU para considerar se uma predição é correta ou incorreta.

Precisão e recall

A avaliação dos métodos de detecção de objetos é principalmente baseada nos conceitos de precisão (P) e *recall* (R), respectivamente definidos como:

$$P = \frac{TP}{TP + FP} = \frac{TP}{\text{todas as detecções}} \quad (2.1)$$

$$R = \frac{TP}{TP + FN} = \frac{TP}{\text{todas as } ground\ truths} \quad (2.2)$$

Em que:

- TP são verdadeiros positivos, *true positive*: ocorre quando uma predição corresponde a uma *bounding box* real;
- FP são falsos positivos, *false positive*: é uma predição incorreta de um objeto que não existe ou uma detecção errada de um objeto existente;
- FN são falsos negativos, *false negative*: ocorre quando existe uma *bounding box ground truth* para um objeto mas que não foi detectada.

É importante notar que, no contexto de detecção de objetos, não se aplicam os verdadeiros negativos, do inglês *true negative* (TN), pois há um número infinito de *bounding boxes* que não devem ser detectadas em qualquer imagem dada. Essas definições (TP, FP, FN) requerem o estabelecimento de uma maneira de avaliar uma detecção como correta ou incorreta. Uma maneira comum de fazer isso é usando a interseção sobre a união (IOU). Ao comparar a IOU com um determinado limiar t , podemos classificar uma detecção como correta ou incorreta. Se $IOU \geq t$, a detecção é considerada correta. Se $IOU < t$, a detecção é considerada incorreta (PADILLA; NETTO; SILVA, 2020).

A precisão é a porcentagem de predições corretas dentre todas as predições feitas pelo modelo, é a capacidade do modelo de identificar apenas objetos que realmente existem. Já o *recall* é a porcentagem de predições corretas dentre todas as ocorrências reais de objetos, um modelo com um bom *recall* dificilmente deixa de detectar um objeto se ele está presente na imagem.

Para um modelo de detecção de objetos ser considerado bom ele deve ter a precisão e o *recall* altos. É possível melhorar a precisão aumentando o critério para considerar uma detecção correta, o que elimina falsos positivos, porém, isso tende a reduzir o *recall*. Por outro lado, se nosso modelo for menos criterioso na detecção, a tendência é aumentar o *recall*, o que aumenta os falsos positivos. Em geral, é importante definir limiares que favoreçam o equilíbrio entre a precisão e o *recall*, porém isso também depende do tipo de aplicação que estamos desenvolvendo. Em aplicações médicas por exemplo, é comum ser preferível maximizar o *recall*.

Uma outra métrica, que deriva diretamente da precisão e do *recall*, é o F1-score. Ele é a média harmônica dessas duas grandezas.

AP - Average Precision

Dentre as métricas usadas para avaliação de modelos de detecção de objetos, a AP - *average precision*, é a mais popular. Ela é obtida calculando-se a área abaixo da curva (*area under*

curve - AUC) da curva precisão-*recall*. Uma das maneiras de fazer isso é usando a interpolação de 11 pontos, que faz uma aproximação do formato do gráfico e facilita o cálculo da AUC. Na interpolação de 11 pontos, a AP é calculada a partir da média dos valores máximos de precisão, dado um conjunto de 11 valores de *recall* igualmente espaçados (PADILLA; NETTO; SILVA, 2020), como mostra a equação 2.3:

$$AP_{11} = \frac{1}{11} \sum_{R \in \{0, 0.1, 0.2, \dots, 0.9, 1\}} P_{\text{interp}}(R) \quad (2.3)$$

onde,

$$P_{\text{interp}}(R) = \max_{\tilde{R}: \tilde{R} \geq R} P(\tilde{R}) \quad (2.4)$$

Segundo a definição da AP_{11} , em vez de usar a precisão $P(R)$ observada em cada valor de *recall* R , o valor da AP é obtido considerando a precisão máxima $P_{\text{interp}}(R)$, cujo *recall* é maior que R .

mAP - Mean Average Precision

A mAP - *mean average precision* é a média da AP sobre todas as classes do *dataset*. Ela é definida pela equação 2.5:

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i \quad (2.5)$$

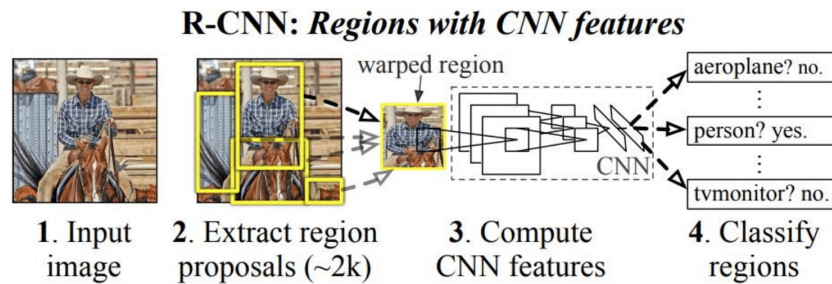
Onde AP_i é o valor da AP para a i -ésima classe e N o total de classes do *dataset*.

2.1.1.3.2 Modelos de detecção de objetos baseados em proposta de regiões (R-CNN, Fast R-CNN e Faster-RCNN)

Os algoritmos baseados na proposta de regiões dividem a tarefa de detecção de objetos em duas etapas: a primeira etapa é responsável por fazer a proposta de regiões, isto é, selecionar as regiões da imagem onde pode haver um objeto. Já a segunda etapa envolve usar uma CNN para extrair as *features* da imagem e fazer a classificação com localização de cada região. Além das probabilidades das classes, também é feita a regressão de valores que representam *offsets* na posição da região para que a *bounding box* envolva melhor o objeto. A Figura 10 exemplifica a *pipeline* do primeiro modelo baseado na proposta de regiões, a R-CNN - Region-based Convolutional Neural Network (GIRSHICK et al., 2014).

A R-CNN foi um grande avanço para detecção de objetos, atingindo uma mAP de 53.3% no *dataset* Pascal VOC 2012 (uma melhora de 30% em relação aos métodos de detecção da época). A R-CNN usa o algoritmo de busca seletiva (UIJLINGS et al., 2013) para fazer a proposta de regiões. Em seguida, passa as regiões por uma rede AlexNet (KRIZHEVSKY; SUTSKEVER;

Figura 10 – Pipeline da R-CNN



Fonte: [Girshick et al. \(2014\)](#)

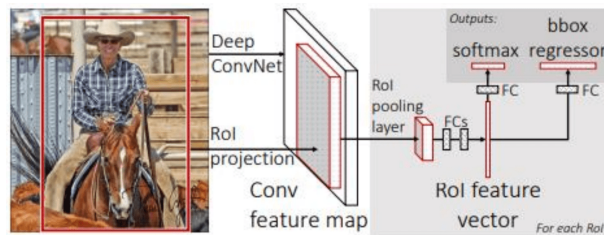
[HINTON, 2012](#)) para obter as *features*, que servem de entrada para uma *support vector machine* (SVM) fazer a classificação. Em seguida, outra etapa faz a regressão do *offset* para refinar as *bounding boxes*. Apesar de fazer previsões bastante precisas, a R-CNN possui algumas limitações:

- Por precisar passar cada região pela CNN a R-CNN acaba sendo bastante lenta (13 segundos por imagem numa GPU e 53 segundos numa CPU), estando bem longe da detecção de objetos em tempo real;
- Cada etapa deve ser ajustada separadamente. Primeiro, é preciso treinar a CNN; em seguida, ajustar a SVM de acordo com as *features* geradas pela CNN. Por fim, também é preciso ajustar a etapa de regressão do *offset* das *bounding boxes*. Tudo isso torna o treinamento demorado e custoso computacionalmente ([GIRSHICK, 2015](#)).

Desde a R-CNN, vários esforços foram realizados para tentar corrigir essas limitações. A Fast R-CNN ([GIRSHICK, 2015](#)) é uma versão que buscou facilitar a *pipeline* de treinamento do modelo e torná-lo mais rápido. A Fast R-CNN compartilha o processamento das camadas convolucionais entre todas as regiões (o que a torna consideravelmente mais rápida que a R-CNN). Além disso, ao invés de usar uma SVM para classificar e fazer a regressão do *offset* para refinar as *bounding boxes* em etapas separadas, estas duas etapas são feitas por camadas lineares incorporadas ao fim da CNN. A CNN faz a classificação com localização para cada região, o que resulta em *bounding boxes* precisas. A Figura 11 apresenta a arquitetura geral da Fast R-CNN. Essas modificações tornaram a Fast R-CNN consideravelmente mais rápida (2 segundos por imagem contra 53 segundos da R-CNN ([GIRSHICK, 2015](#))). Ela atingiu 66,9% de mAP no *dataset* Pascal VOC 2007, contra 66% da R-CNN. Além disso, o treinamento ficou 9 vezes mais rápido que a R-CNN. Isso ocorre pois a Fast R-CNN pode ser treinada de uma só vez e não cada etapa separadamente.

Excluindo o tempo da proposta de regiões, a Fast R-CNN leva 0,3 segundos por imagem ([GIRSHICK, 2015](#)). Com a proposta de regiões, esse tempo sobe para 2 segundos, ficando claro que a etapa de propor regiões é o gargalo desse tipo de método. A Faster R-CNN ([REN et al.](#),

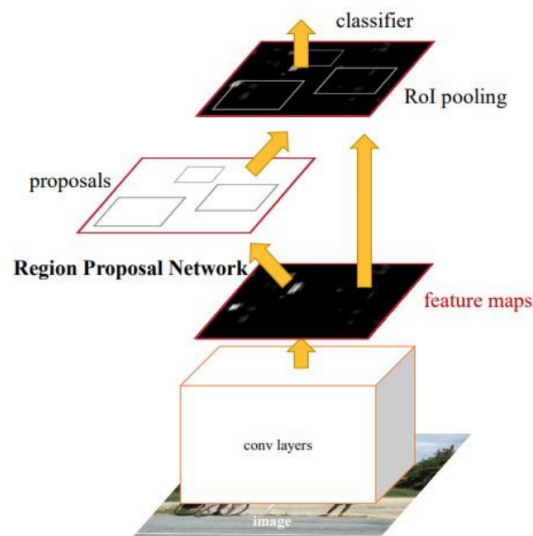
Figura 11 – Arquitetura geral da Fast R-CNN



Fonte: [Girshick \(2015\)](#)

2015) surgiu após a Fast R-CNN e busca reduzir o tempo gasto para propor regiões. [Ren et al. \(2015\)](#) apresentaram mudanças na computação das regiões através da introdução de *region proposals networks* (RPNs), uma rede neural convolucional que aproveita *features* de camadas compartilhadas com a CNN principal e faz a proposta das regiões com um custo praticamente zero. Ao compartilhar convoluções a RPN consegue propor regiões levando apenas 10ms por imagem ([REN et al., 2015](#)).

Figura 12 – Arquitetura geral da Faster R-CNN



Fonte: [Ren et al. \(2015\)](#)

A imagem de entrada é passada pela CNN, que faz a extração de características de cada região. Em seguida, as *features* geradas são passadas pra RPN, que faz a proposta de regiões. Por fim, as regiões geradas pela RPN e as *features* da CNN são passadas para as camadas lineares, que fazem a classificação com localização de cada região. A arquitetura geral da Faster R-CNN é apresentada na Figura 12. Durante o treinamento, o ajuste dos pesos é intercalado entre a RPN e a CNN principal.

Usando a RPN, a Faster R-CNN conseguiu reduzir o tempo de inferência para 0,2 segundos por imagem (contra 2 segundos da Fast R-CNN), uma melhora bastante significativa e ainda manteve a mesma mAP da Fast R-CNN no *dataset* Pascal VOC 2007 (66,9%) (REN et al., 2015).

2.1.1.3.3 YOLO - You Only Look Once

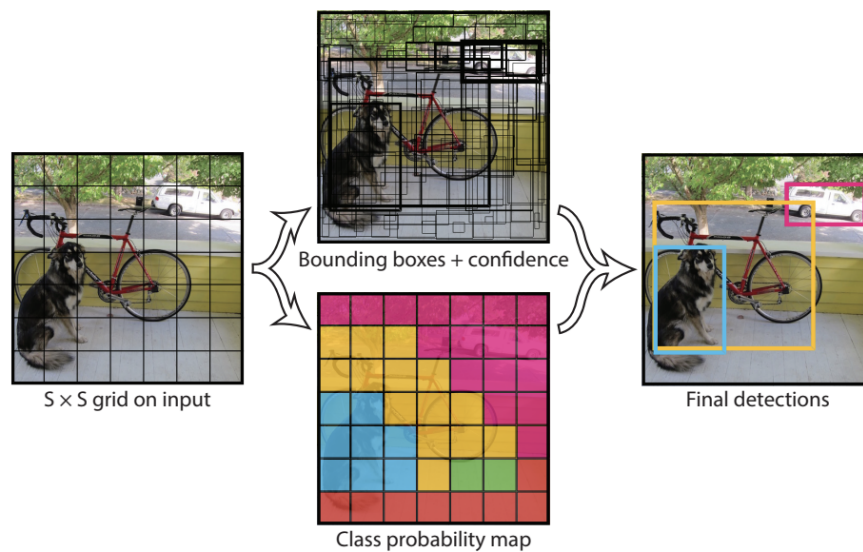
A YOLO (REDMON et al., 2016) é um modelo profundo bastante popular usado para detecção de objetos em tempo real. Diferente de sistemas anteriores, que encaravam o problema de detecção de objetos como uma tarefa de classificação (algoritmo de janela deslizante (FELZENSZWALB et al., 2009) e algoritmos baseados na proposta de regiões (GIRSHICK, 2015)), a YOLO aborda a detecção de objetos como um problema de regressão.

Para realizar detecção de objetos, a YOLO divide a imagem de entrada em $S \times S$ células. Se o centro de um objeto cai dentro de uma célula, aquela célula é responsável por detectar aquele objeto. Para cada célula, são feitas B predições de *bounding boxes* e *scores* de confiança (a probabilidade de existir um objeto naquela *bounding box* e o quão bem ela envolve o objeto). Cada *bounding box* consiste em cinco valores (x, y, w, h, c) , sendo (x, y) as coordenadas do ponto central do objeto (em relação à célula em que ele caiu), (w, h) o comprimento e largura da *bounding box* (em relação ao tamanho da imagem de entrada). Note que a *bounding box* pode ser maior que a célula em que ela cai. E c o *score* de confiança. Cada célula também faz C predições de probabilidades para cada classe da base de dados (REDMON et al., 2016). Em resumo, a principal ideia por trás da YOLO é usar uma CNN para fazer a predição de um vetor $(S \times S \times (B.5 + C))$. A Figura 13 apresenta o esquema de funcionamento da YOLO. Antes de fazer as predições finais, a YOLO usa o algoritmo *non-maximum suppression* para remover *bounding boxes* duplicadas ou com *score* de confiança muito baixo.

A primeira versão da YOLO (REDMON et al., 2016) representou um avanço significativo para detecção de objetos em tempo real, conseguindo realizar 45 inferências por segundo. Uma versão reduzida da YOLO conseguiu impressionantes 155 inferências por segundo. Além de ser extremamente rápida, a YOLO também superou todos os sistemas anteriores de detecção de objetos em tempo real, atingindo o dobro da mAP no *dataset* Pascal VOC 2007 (EVERINGHAM et al., 2007). O principal motivo da YOLO ser tão rápida é que ela é capaz de realizar inferências passando a imagem apenas uma vez pela CNN. Essa é a razão pela qual a arquitetura se chama YOLO – You Only Look Once.

Além de ser veloz, a YOLO preserva informações do contexto global da imagem. Como a YOLO recebe a imagem inteira durante o treinamento e teste, ela implicitamente codifica informações contextuais das classes. Métodos que dividem a imagem de entrada como algoritmos de janela deslizante e estratégias baseadas na proposta de regiões não aproveitam essa informação, o que faz com que a YOLO cometa menos erros de *background* — erros em que a IOU é menor que 0,1 para um dado objeto. Além disso, a YOLO generaliza melhor a representação dos objetos.

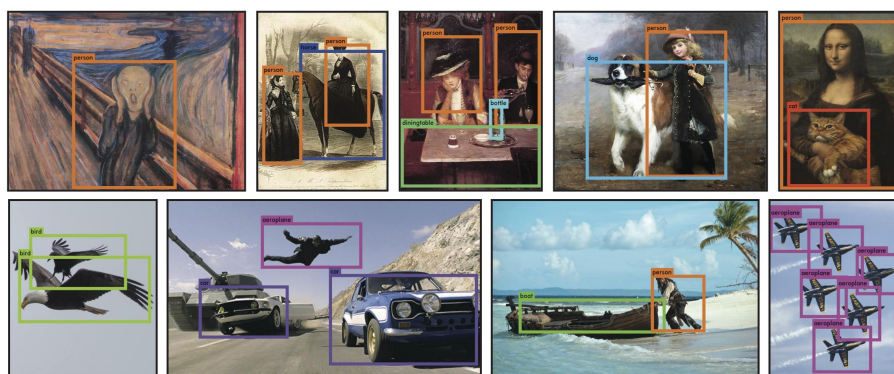
Figura 13 – Esquema de funcionamento da YOLOv1



Fonte: Redmon et al. (2016)

Os autores treinaram a YOLO em imagens naturais e testaram em imagens de arte, vide Figura 14, e dentre os modelos testados, a YOLO foi a que generalizou melhor (REDMON et al., 2016). Apesar disso, a YOLO ainda tem acurácia menor em relação a métodos que não são em tempo real. Outra limitação da YOLO é a dificuldade para localizar objetos pequenos.

Figura 14 – YOLO rodando em imagens de arte e em imagens da natureza obtidas da Internet



Fonte: Redmon et al. (2016)

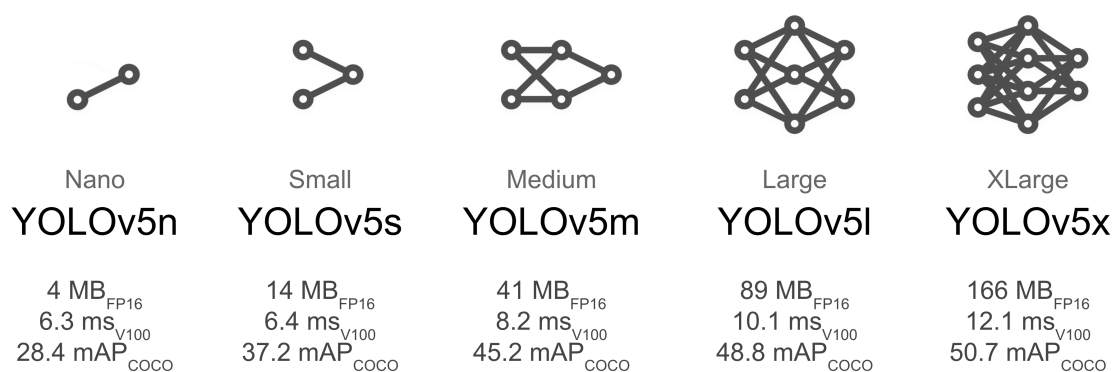
Desde a primeira versão da YOLO (REDMON et al., 2016), novas versões foram publicadas, sempre buscando o aprimoramento do modelo. A YOLOv2 ou YOLO9000 (REDMON; FARHADI, 2017) é a segunda versão e corrigiu algumas das limitações da YOLOv1. A principal mudança foi a introdução das *anchor boxes*, que solucionaram o problema de uma célula fazer a predição de mais de um objeto (a YOLOv1 se limitava a um objeto por célula). Além disso, a

arquitetura da YOLOv2 foi aprimorada e passou a se basear na Darknet-19, ao invés GoogLeNet (SZEGEDY et al., 2015). Outro aprimoramento foi a adição de camadas de *batch normalization*. Todas essas alterações melhoraram o desempenho do modelo, que atingiu 78,6% de mAP no *dataset* Pascal VOC 2007 e 40 inferências por segundo, superando métodos que eram o estado da arte como Faster R-CNN (REN et al., 2015) e SSD - Single Shot Detector (LIU et al., 2016) e sendo significativamente mais rápida.

A YOLOv3 (REDMON; FARHADI, 2018) foi a última versão publicada pelos autores da YOLOv1 e introduziu alguns aprimoramentos (bem menos do que a YOLOv2). Depois da YOLOv3, a YOLOv4 (BOCHKOVSKIY; WANG; LIAO, 2020) foi publicada. Essa versão representou uma evolução bastante significativa em relação à YOLOv3. No *dataset* COCO (LIN et al., 2015), a YOLOv4 melhorou a mAP e o *frame rate* em 10% e 12% respectivamente. Em resumo, Bochkovskiy, Wang e Liao (2020) experimentaram diversas técnicas conhecidas que são aplicadas para melhorar o desempenho de CNNs e analisaram como essas técnicas poderiam ser combinadas para melhorar o desempenho de um modelo de detecção de objetos. A YOLOv4 atingiu 65 FPS (*Frames* por segundo) usando uma NVIDIA TESLA V100 e sua versão reduzida, YOLOv4 *tiny*, 371 inferências por segundo usando uma NVIDIA GEFORCE GTX 1080 TI.

A versão mais recente da YOLO até o momento é a YOLOv5 (JOCHER et al., 2022). Essa versão foi desenvolvida usando o *framework* PyTorch (PASZKE et al., 2019) e conta com diversas modificações na arquitetura. Essas modificações envolvem principalmente diferentes tamanhos de modelos, desde o menor de todos — a YOLOv5 Nano — até um modelo bem mais complexo — a YOLOv5 XL. A Figura 15 mostra as diferentes variações da YOLOv5:

Figura 15 – Variações da YOLOv5



Fonte: Jocher et al. (2022)

2.1.1.3.4 SSD - Single Shot Detector

A SSD - Single Shot Detector é uma arquitetura de CNN voltada para detecção de objetos. A abordagem da SSD é baseada em uma rede convolucional *feed-forward* que produz uma

coleção de tamanho fixo de *bounding boxes* e *scores* para a presença de instâncias de objetos nessas *bounding boxes*. Foi proposta por [Liu et al. \(2016\)](#) e, assim como a YOLO, é um detector de objetos de um estágio: faz a predição da classe e da *bounding box* no mesmo momento. A SSD conseguiu superar a YOLOv1 e a Faster R-CNN tanto em velocidade de processamento quanto em mAP no *dataset* Pascal VOC2007 ([EVERINGHAM et al., 2007](#)). A Tabela 1 apresenta a comparação entre esses modelos.

Tabela 1 – Comparação de desempenho da Faster R-CNN, YOLOv1 e SSD no *dataset* Pascal VOC2007

Modelo	Frame rate (FPS)	mAP (%)
Faster R-CNN	7	73,2
YOLOv1	45	63,4
SSD	59	74,3

Fonte: Elaborado pelo autor

Dada uma imagem, a SSD discretiza as possíveis *bounding boxes* de saída em um conjunto de *bounding boxes* padrão. Essas, por sua vez, possuem diferentes proporções e escalas e estão distribuídas na imagem. No momento da inferência, a rede gera *scores* para a presença de cada classe de objeto em cada *bounding box* e produz ajustes para melhor envolver o objeto. Em seguida, uma etapa de *non-maximum suppression* gera as detecções finais. Além disso, a rede combina previsões de vários *feature maps* com diferentes resoluções para lidar naturalmente com objetos de vários tamanhos ([LIU et al., 2016](#)).

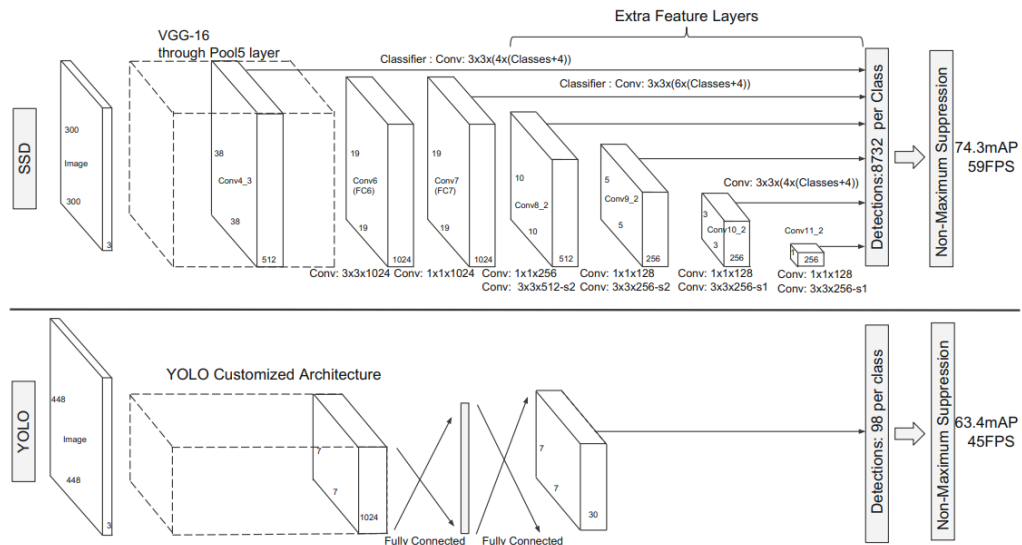
As primeiras camadas de rede são baseadas em uma arquitetura usada para classificação de imagens (truncada antes das camadas de classificação). Essa parte é chamada de rede base. Em seguida, uma estrutura auxiliar produz as detecções ([LIU et al., 2016](#)). A Figura 16 apresenta a comparação entre a arquitetura da YOLOv1 e da SSD (ambos detectores de objetos de um estágio). A SSD é simples em relação aos métodos que requerem propostas de regiões porque elimina completamente o custo da geração de propostas e encapsula todos os cálculos em uma única rede. Isso torna a SSD fácil de treinar e de integrar em sistemas que requerem um componente de detecção ([LIU et al., 2016](#)).

Apesar de possuir um tamanho de entrada de 300×300 a SSD foi capaz de superar a YOLOv1, que possui tamanho de entrada de 448×448, em acurácia e em velocidade.

2.1.1.3.5 CenterNet

A CenterNet foi proposta por [Duan et al. \(2019\)](#) e é um modelo de detecção de objetos de um estágio. Sua arquitetura foi inspirada na CornerNet ([LAW; DENG, 2018](#)), um modelo que baseia a detecção dos objetos no uso de *keypoints*. Grande parte dos modelos modernos usados para detecção de objetos, como a YOLO ([REDMON; FARHADI, 2018](#)) e SSD ([LIU](#)

Figura 16 – Comparação entre a arquitetura da YOLOv1 e da SSD



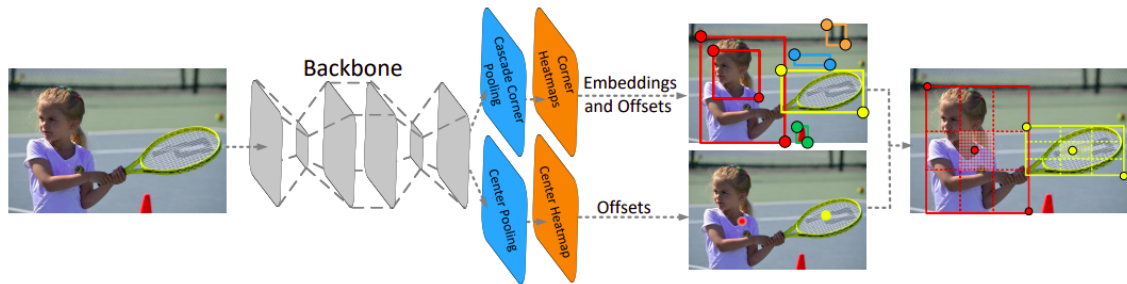
Fonte: Liu et al. (2016)

et al., 2016), se baseiam no uso de *anchor boxes*: um conjunto de retângulos com tamanhos e proporções previamente estabelecidos usados pelos modelos para gerar as *bounding boxes*. Essa abordagem frequentemente sofre com algumas limitações, como a necessidade de um grande número de *anchor boxes* para gerar *bounding boxes* com uma boa IOU e, em algumas abordagens, a necessidade de definir as *anchor boxes* manualmente (DUAN et al., 2019).

O uso de *keypoints* para detecção de objetos visa reduzir as limitações causadas pelas *anchor boxes*. Essa estratégia foi proposta inicialmente por Law e Deng (2018), com a publicação da CornerNet: uma arquitetura que baseia a detecção de objetos no uso de um par de *keypoints*. A CornerNet não tem a necessidade de utilizar *anchor boxes* e atingiu o desempenho de detectores de um estágio do estado da arte. Apesar do sucesso, ela gera um número considerável de detecções incorretas. Isso acontece porque apesar de ser boa para detectar os cantos dos objetos, a CornerNet frequentemente combina dois *keypoints* de maneira errada (DUAN et al., 2019). Para mitigar esse problema, a CenterNet se baseia no uso de três *keypoints* — os dois cantos e o centro do objeto.

A ideia por trás da CenterNet é usar uma rede convolucional como backbone, que aplica operações de *corner pooling* e *center pooling* para gerar um conjunto de cantos e um conjunto de centros. Assim como a CornerNet faz, a CenterNet usa um par de *keypoints* (cantos) para detectar uma *bounding box* em potencial. Em seguida, os *keypoints* de centro detectados são usados para determinar as caixas delimitadoras finais. Essa *pipeline* está apresentada na Figura 17.

Figura 17 – Arquitetura da CenterNet

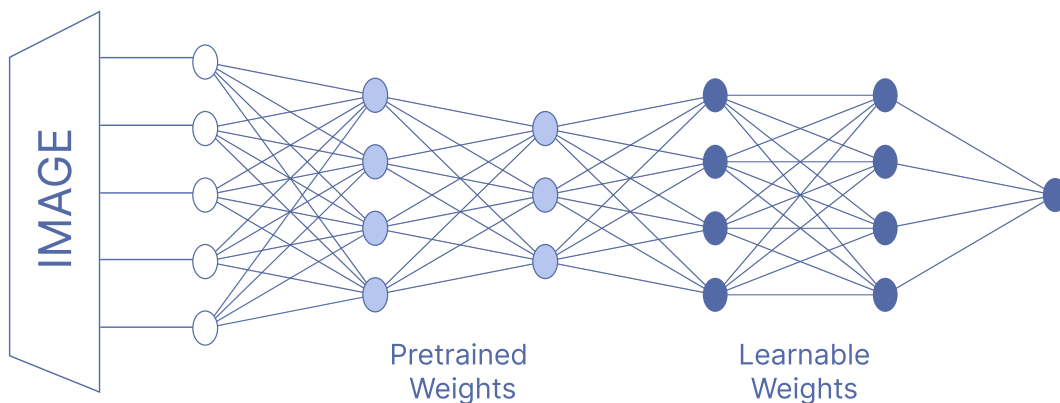


Fonte: [Duan et al. \(2019\)](#)

2.1.2 Transferência de aprendizado

A transferência de aprendizado, *transfer learning*, é uma técnica de treinamento de redes neurais. Ela consiste em aproveitar os pesos de uma rede neural e ajustá-la para um domínio diferente do qual ela foi originalmente treinada. Nessa técnica, os pesos das camadas iniciais são “congelados”, isto é, não são atualizados durante o treinamento, sendo apenas feito o ajuste na última (ou nas últimas) camadas do modelo. A Figura 18 exemplifica o uso de *transfer learning* em uma arquitetura geral.

Figura 18 – Exemplo de uma arquitetura geral com o uso de transfer learning. As camadas do fim são treinadas em um novo domínio e o conhecimento do modelo no domínio em que foi originalmente treinado é aproveitado através dos pesos das camadas iniciais



Fonte: Adaptado de ([CHOUHAN et al., 2020](#))

Essa técnica requer ainda que a última camada do modelo seja alterada para atender ao novo domínio. Por exemplo, se estamos aproveitando os pesos de uma AlexNet treinada no *dataset* ImageNet e o novo domínio possui duas classes, a última camada do modelo deve ser alterada, de modo que a saída passe a ser duas classes e não as 1000 classes do *dataset* ImageNet.

2.2 Automatic Licence Plate Recognition (ALPR)

Um sistema de reconhecimento automático de placas de licença - *automatic license plate recognition* (ALPR), utiliza uma imagem ou uma sequência delas para extrair informações da placa de um veículo. Isso é atingido por meio da combinação de diversas técnicas de processamento de imagens, detecção de objetos e reconhecimento de padrões. Essa tarefa também pode ser chamada de identificação automática de veículos, reconhecimento de placas de automóveis ou reconhecimento óptico de caracteres (OCR - *optical character recognition*) para carros (DU et al., 2012).

O ALPR é uma tarefa complexa devido as muitas variações que podem interferir no desempenho do reconhecimento. As imagens podem ser tiradas por uma câmera colorida, preto e branco ou infravermelha. Além disso, existem variações no ambiente e no tipo das placas, vide Figura 21, que influenciam diretamente o desempenho do sistema.

Figura 19 – Exemplos de placas de licença capturadas em condições desafiadoras



Fonte: Jørgensen (2017)

Os principais desafios no reconhecimento das placas são:

1. Variações dos tipos de placa:

- Localização: Dependendo do tipo de veículo, a placa pode estar localizada em diferentes posições;
- Tamanho: No Brasil, por exemplo, placas de motos tem o tamanho diferente das placas de outros veículos;
- Fonte: Placas de diferentes países podem ser escritas em diferentes fontes, idiomas e tipos de caracteres;

- Cor: Veículos de diferentes categorias podem possuir placas de diferentes cores, como acontece no Brasil;
- Padrão: Nações têm padrões diferentes para placas de veículos, permitindo cores de fundo e fontes diferentes, além da quantidade de letras, números, espaços e suas posições variarem. Essa variação de padrões pode ser um problema em regiões de fronteira ou mesmo dentro de um país com mais de um padrão, como ocorre nos Estados Unidos, onde diferentes estados possuem diferentes padrões de placas. Um país também pode adotar um novo padrão, como ocorreu recentemente no Brasil, que passou a usar o padrão de placas do MERCOSUL. Os sistemas ALPR devem ser capazes de reconhecer vários padrões de placas, uma vez que podem haver vários no mesmo país ou região. No entanto, segundo [Kessentini et al. \(2019\)](#), a maioria das soluções existentes funcionam apenas para um padrão de placa específico.

2. Variações no ambiente:

- Oclusão: A placa do veículo pode estar parcialmente ou totalmente obstruída por sujeira (lama e neve por exemplo) ou por outros elementos da cena;
- Inclinação: A placa pode estar inclinada, o que distorce os caracteres;
- Iluminação: As imagens de entrada podem ter diferentes tipos de iluminação, principalmente devido à iluminação do ambiente e aos faróis dos veículos;
- *Background*: O plano de fundo da cena pode conter padrões semelhantes a placas, como números estampados, para-choques com padrões verticais e adesivos.

Um sistema ALPR tipicamente possui três etapas: Detecção da placa, detecção dos caracteres e reconhecimento dos caracteres da placa. Para o sistema ALPR ser eficiente, todas as etapas devem ser robustas, principalmente a detecção da placa, já que um erro nessa etapa compromete todo o restante da *pipeline*. Muitas abordagens dividem a detecção da placa em dois estágios: primeiro é feita a detecção do veículo na imagem de entrada, depois a placa é detectada dentro da detecção do veículo. Isso torna o sistema mais robusto e elimina falsos positivos ([LAROCA et al., 2018](#)).

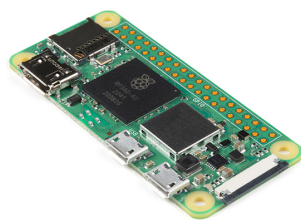
2.3 Dispositivos para computação na borda

2.3.1 Raspberry Pi

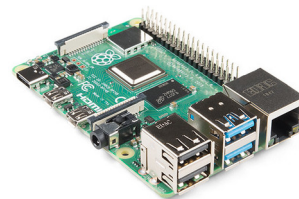
Raspberry Pi é uma série de computadores de placa única (*single board computer*). Foram desenvolvidos inicialmente no Reino Unido, em 2006, com o propósito de serem dispositivos baratos que ajudassem no ensino de computação nas escolas. Seu lançamento comercial foi em fevereiro de 2012. O tamanho compacto, preço acessível e baixo consumo de energia tornaram essa uma das placas mais populares em aplicações para Internet das coisas.

O Raspberry Pi 4 Modelo B é a versão mais recente da popular série Raspberry Pi. Apesar de ter consumo de energia semelhante, em comparação com a geração anterior (Raspberry Pi 3 Modelo B+), ele oferece aumentos significativos de velocidade de processamento, desempenho de multimídia, memória e conectividade (SÜZEN; DUMAN; ŞEN, 2020). O Pi 4 modelo B possui de 2GB a 8GB de memória RAM, processador quad-core Cortex-A72 (ARM v8 64-bit 1.5GHz) e uma GPU Broadcom VideoCore VI (500 MHz). Ele não contém um disco rígido embutido, mas usa um cartão SD para armazenamento que deve conter o sistema operacional Linux. A Fundação Raspberry Pi disponibiliza um sistema operacional específico para os Raspberries, o Raspberry Pi OS ou Raspbian. Trata-se de uma distribuição leve baseada em Debian.

Figura 20 – Modelos de Raspberries



(a) Raspberry Pi Zero



(b) Raspberry Pi 4

Fonte: SparkFunElectronics¹

O Raspberry Pi é amplamente usado no desenvolvimento de sistemas embarcados. O baixo custo e a melhoria contínua de seus componentes ao longo das versões incentiva seu uso na produção de soluções para diversos problemas em diferentes cenários, inclusive no contexto das redes neurais (POPAT; SHETH; JAIN, 2019; GONZALEZ-HUITRON et al., 2021; MANJARI; VERMA; SINGAL, 2019; ZHAO et al., 2018; PENA et al., 2017).

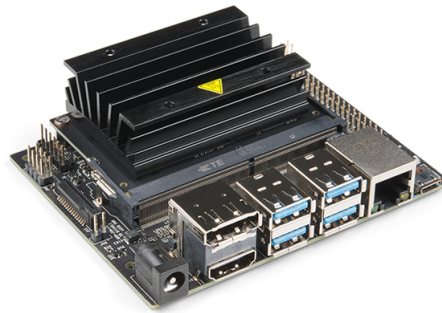
2.3.2 NVIDIA Jetson Nano

A NVIDIA Jetson Nano é a placa de entrada do ecossistema NVIDIA Jetson. Trata-se de um compacto e poderoso computador de placa única que permite a operação paralela de redes neurais para aplicações como classificação de imagem, detecção de objetos, segmentação semântica e processamento de fala. Ela tem um ambiente de desenvolvimento abrangente (JetPack SDK) e bibliotecas desenvolvidas para aplicativos embarcados, aprendizado profundo, IoT - Internet of things, visão computacional, gráficos e multimídia (SAEEDI, 2019).

A Jetson Nano conta com uma unidade de processamento gráfico (128 CUDA cores NVIDIA Maxwell 921 MHz), que dá suporte ao uso de computação paralela via CUDA - Compute Unified Device Architecture. O modelo Jetson Nano possui versões de 2GB ou 4GB de memória RAM e processador quad-core ARM A57 (1.43 GHz). Assim como ocorre com o Raspberry Pi o armazenamento e o sistema operacional precisam de um cartão SD. A NVIDIA disponibiliza uma imagem oficial baseada no Ubuntu, que vem com diversos programas instalados, entre eles

¹ Disponível em: <<https://www.flickr.com/photos/sparkfun/>>. Acesso em novembro de 2021.

Figura 21 – NVIDIA Jetson Nano



Fonte: SparkFunElectronics¹

o JetPack SDK e o TensorRT (*framework* da NVIDIA específico para execução de redes neurais em GPUs da marca). Por conta desses aspectos a Jetson Nano é amplamente utilizada em projetos relacionados à robótica e à automação industrial utilizando *deep learning* (ZHOU et al., 2020; AGUIAR et al., 2020).

2.3.3 Kendryte K210

O Kendryte K210 é um SoC, *system on chip*, que integra mecanismos para visão computacional e processamento de áudio (SIPEED, 2020). Foi lançado em setembro de 2018 e tem como alvo os mercados de inteligência artificial e IoT, mas também pode ser usado como um *microcontroller unit* (MCU) de alto desempenho (TORRES-SÁNCHEZ; ALASTRUEY-BENEDÉ; TORRES-MORENO, 2020).

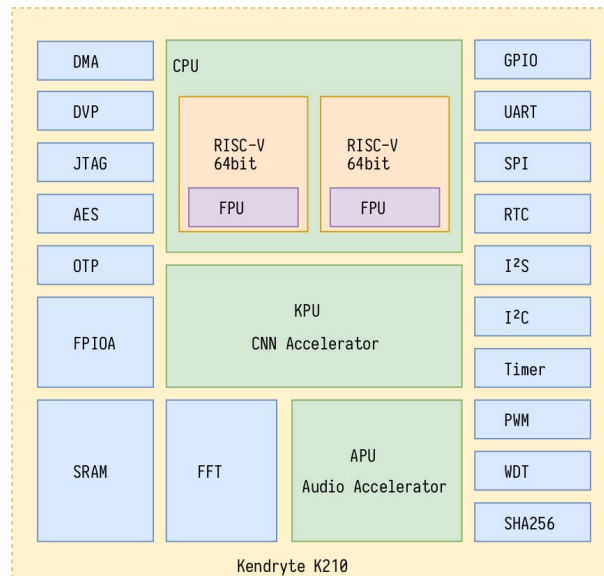
A Figura 22 mostra o diagrama de blocos do Kendryte K210. Seus principais componentes são um processador dual core RISC-V de 64 bits, um acelerador para redes neurais convolucionais (KPU - *knowledge processing unit*), um acelerador para processamento de áudio (*audio processing unit*) e uma memória SRAM de 8 MB. A SRAM é dividida em duas partes: 6 MB são dedicados à computação de uso geral e 2 MB para a KPU, que pode ser usada como memória principal, se a KPU não a estiver usando.

Cada núcleo da CPU contém uma FPU (*floating point unit*) que suporta operações de multiplicação, divisão e raiz quadrada de precisão única e dupla. Cada núcleo também possui um cache de instrução de 32 KB e um cache de dados de 32 KB. A frequência pode ser ajustada de 400 MHz até 800 MHz (SIPEED, 2020).

A KPU é um processador para redes neurais que conta com operações de convolução (*kernels* 1×1 e 3×3), *batch normalization*, funções de ativação (por exemplo, ReLU, sigmoid) e operações *pooling* (por exemplo, *max-pooling* e *average pooling*). Não há limite explícito para o número de camadas de rede. Cada camada da CNN pode ser configurada separadamente,

¹ Disponível em: <<https://www.flickr.com/photos/sparkfun/>>. Acesso em novembro de 2021.

Figura 22 – Diagrama de blocos do K210



Fonte: [Sipeed \(2020\)](#)

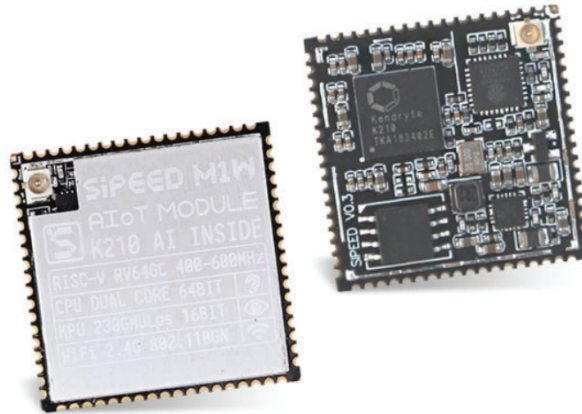
incluindo o número de canais de entrada e os tamanhos de entrada e saída. A inferência em tempo real (mais de 30 FPS) pode ser alcançada se o tamanho da rede neural for menor que 5,9 MB. A capacidade de memória flash disponível é o limite em aplicativos que não são em tempo real ([TORRES-SÁNCHEZ; ALASTRUEY-BENEDÉ; TORRES-MORENO, 2020](#)).

A empresa fornecedora do K210 comercializa também o módulo MAIX-I, vide Figura 23, que integra o K210, um circuito de alimentação, memória flash de 8MB/16MB/128MB e o módulo Wi-Fi ESP8285. Na Figura 23 o K210 está no canto superior esquerdo do chip, à direita dele está o ESP8285 e o conector da antena. O chip de memória flash fica no canto inferior esquerdo.

O módulo MAIX-I é o coração de várias placas de desenvolvimento (MAIX Dock, MAIX Duino, MAIX Go). Essas placas foram desenvolvidas para serem soluções de baixo custo e baixo consumo para o desenvolvimento de aplicações de IA em sistemas embarcados. A Figura 24 apresenta a MAIX Go Development Board, que conta com recursos como USB tipo C, que pode ser usada para alimentação ou UART, conector JTAG, que pode ser usado para upload de código, *debug* ou comunicação, acelerômetro de 3 eixos, microfone I2S, auto-falante, LED RGB, um slot para cartão TF (TransFlash), bateria de lithium de 500mA, display LCD de 2,8 polegadas, câmera e antena Wi-Fi. A MAIX Go Development Board custa US\$ 40.

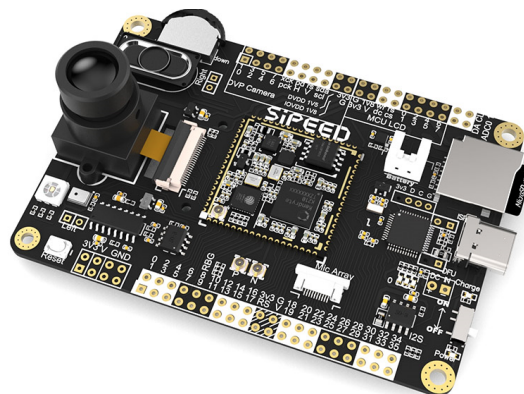
Apesar da ótima relação de custo benefício e do grande potencial das placas baseadas no chip MAIX-I, o desenvolvimento de aplicações nessa plataforma ainda é uma tarefa desafiadora. A documentação para desenvolvedores é pobre, os ambientes de desenvolvimento com baixos níveis de maturidade e o processo de depuração é difícil e às vezes impossível ([TORRES-SÁNCHEZ;](#)

Figura 23 – Módulo MAIX-I



Fonte: [Sipeed \(2020\)](#)

Figura 24 – MAIX Go Development Board



Fonte: [Sipeed \(2020\)](#)

[ALASTRUEY-BENEDÉ; TORRES-MORENO, 2020](#)). Esses desafios tendem a diminuir a partir da popularização dessas plataformas.

3

Trabalhos Relacionados

Este capítulo apresenta a revisão da literatura acerca do desenvolvimento de sistemas ALPR. A revisão foi realizada em duas etapas: em um primeiro momento, buscamos soluções de ALPR comercializadas atualmente, em seguida, nossa busca focou em trabalhos acadêmicos. A busca foi feita dessa forma pois a comparação entre sistemas comerciais e trabalhos acadêmicos é difícil, uma vez que os sistemas comerciais omitem informações técnicas e usam bases de dados privadas. As tecnologias comercializadas atualmente são apresentadas na Seção 3.1 e o desenvolvimento acadêmico relacionado ao tema na Seção 3.2.

3.1 Produtos Comerciais

3.1.1 Plate Recognizer

O Plate Recognizer¹ é uma solução comercial para ALPR, que suporta placas de mais de 200 países e fornece produtos *on-premise* e na nuvem. A solução *on-premise* pode ser instalada em diferentes plataformas, inclusive em dispositivos de borda, o sistema atinge 1 FPS no Raspberry Pi 3/4 e 4 FPS no Jetson Nano da NVIDIA.

O Plate Recognizer possui planos que cobram por câmera ou por requisição. Na cobrança por câmera o valor é de US\$ 35 por câmera por mês. Além de pacotes extras de US\$ 10 para ter informações como marca, modelo, ano e cor e de US\$ 15 para ter acesso ao *dashboard* de visualização dos dados. Já para cobrança por requisição (que são limitadas a 8 por segundo) esse valor é de US\$ 50 para 50.000 requisições mensais, US\$ 150 para 250.000 e US\$ 250 para 500.000. Caso o cliente queira informações como marca, modelo, ano e cor esse preço aumenta 50% e para ter acesso ao *dashboard* o preço também aumenta 50%.

¹ Disponível em: <<https://platerecognizer.com/>>. Acesso em novembro de 2021.

² Disponível em: <<https://www.sighthound.com/>>. Acesso em novembro de 2021.

3.1.2 Sighthound

O Sighthound² suporta placas de mais de 100 países. O serviço pode ser implantado na nuvem ou usado em equipamentos próprios (*on-premise*). O site informa que diversas plataformas podem ser usadas, inclusive em dispositivos de borda, mas não apresenta detalhes acerca do tempo de processamento nem quais são os dispositivos. A única informação é que o sistema roda a 160 FPS em uma GPU (não informam o modelo) e em tempo real usando uma CPU (não informam o modelo).

A cobrança é feita por câmera, sendo a partir de US\$ 30 por câmera por mês. Além disso, o dashboard para visualização dos dados deve ser contratado separadamente. Eles também fornecem outras informações como fabricante, modelo, cor e ano. Em relação à acurácia, a única informação disponível no site é que o sistema possui 99% de acurácia no *dataset Caltech cars* (WEBER, 1999), um *dataset* de apenas 126 imagens.

3.1.3 OpenALPR

O OpenALPR¹ é um sistema comercial que possui uma solução específica para processamento na borda, a Rekor Edge, vide Figura 25. Contudo, o hardware usado e o tempo de processamento não são apresentados no site. Eles também fornecem uma solução baseada em requisições, que custa US\$ 29 para 2.500 requisições mensais, US\$ 99 para 25.000 e US\$ 395 para 125.000. Além disso, o OpenALPR suporta placas de quase 70 países e fornece informações como fabricante, modelo, cor e ano.

Figura 25 – Rekor Edge



Fonte: [OpenALPR \(2021\)](#)

3.1.4 Análise Comparativa

Os produtos comerciais tendem a revelar pouca informação acerca das soluções. Além disso, a comparação entre sistemas comerciais e trabalhos acadêmicos requer muito esforço, uma vez que é preciso estabelecer um ambiente que possibilite uma comparação justa. Por exemplo, é preciso escolher um *dataset* e executar os diferentes sistemas no mesmo hardware. O ideal é que o *dataset* escolhido não tenha sido usado no treinamento de nenhum dos sistemas envolvidos na análise.

A Tabela 2 apresenta a comparação entre os sistemas comerciais Plate Recognizer, Sighthound e OpenALPR, sendo as informações de preço referentes aos valores mensais para utilização dos sistemas (por câmera ou por requisição). Essa informação e a quantidade de países suportados foram obtidas no site de cada sistema. O desempenho dos sistemas no *dataset* UFPR-ALPR foi obtido na comparação feita por Laroca et al. (2019), que propuseram um sistema para o *dataset* UFPR-ALPR (LAROCA et al., 2018) e apresentaram uma comparação de desempenho entre o sistema proposto e os sistemas comerciais Sighthound e OpenALPR.

Tabela 2 – Comparação dos sistemas comerciais Plate Recognizer, Sighthound e OpenALPR

Critério	Plate Recognizer	Sighthound	OpenALPR
Países suportados	200+	100+	70-
Preço por câmera (US\$)	35	30	—
Preço por requisição plano intermediário (US\$)	0,0006	—	0,00396
Taxa de reconhecimento <i>dataset</i> UFPR-ALPR (%)	—	62,3	82,2

Fonte: Elaborado pelo autor

3.2 Trabalhos Acadêmicos

Nesta seção são apresentados trabalhos acadêmicos que usaram *deep learning* no contexto dos sistemas ALPR. Cada subseção apresenta um artigo publicado, detalhando o funcionamento da abordagem proposta, suas contribuições, limitações e resultados. A Seção 3.2.4 apresenta trabalhos relacionados, que de alguma forma inspiraram os apresentados nas seções anteriores.

3.2.1 A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector

Laroca et al. (2018) apresentaram uma abordagem baseada no uso do detector de objetos YOLO para o problema de reconhecimento automático de placas de licença automotiva. Suas principais contribuições são listadas a seguir.

¹ Disponível em: <<https://br.openalpr.com/>>. Acesso em novembro de 2021.

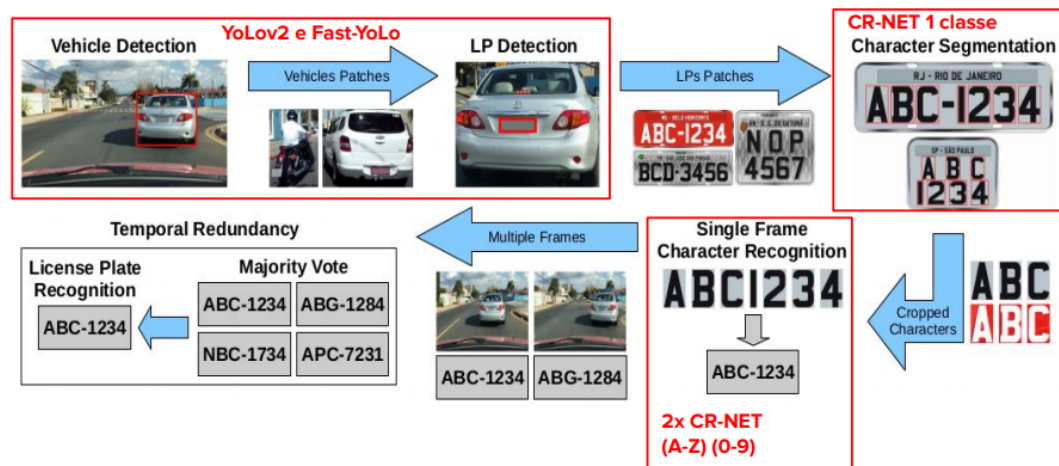
- Um novo sistema ALPR *end-to-end* capaz de realizar inferências em tempo real em uma GPU. O sistema tem como base o uso do detector de objetos YOLO;
- Uma abordagem robusta de dois estágios para segmentação e reconhecimento dos caracteres, principalmente devido ao uso de técnicas de *data augmentation* durante o treinamento como imagens invertidas para as placas de licença e caracteres espelhados;
- Um novo *dataset* público para ALPR (UFPR-ALPR) com 4500 imagens anotadas com mais de 30000 caracteres e placas de 150 veículos distintos;
- Uma avaliação comparativa entre os resultados alcançados pela abordagem proposta e dois sistemas comerciais.

A Figura 26 apresenta a *pipeline* proposta por Laroca et al. (2018). Cada etapa usa uma diferente CNN, assim, cada tarefa pode ser otimizada separadamente. Primeiro, a detecção da placa é feita em dois estágios: o primeiro estágio usa uma YOLOv2 (REDMON; FARHADI, 2017) para detectar o veículo dentro da imagem de entrada e o segundo usa uma Fast YOLO (versão reduzida da YOLOv2) para detectar a placa dentro da detecção do veículo.

Em seguida, uma vez que a placa já foi detectada, é preciso fazer a segmentação e reconhecimento de cada caractere. Laroca et al. (2018) preferiram dividir essas etapas em dois estágios: no primeiro estágio uma CR-NET (MONTAZZOLLI; JUNG, 2017) (trata-se de uma CNN inspirada na arquitetura da Fast YOLO) faz a segmentação dos caracteres, essa etapa possui apenas uma classe (caractere) e sua função é obter as *bounding boxes* de cada caractere da placa.

Na etapa de reconhecimentos dos caracteres são usadas duas redes CR-NET: uma para os dígitos (0-9) e outra para as letras (A-Z), a diferenciação se o caractere é letra ou dígito é feita com base na posição do caractere na placa. Por fim, é feita a combinação do resultado de diferentes *frames* na etapa de redundância temporal: as saídas de diferentes instantes de tempos são combinadas e os caracteres que ocorreram com maior frequência em cada posição são considerados corretos.

Figura 26 – Pipeline proposta por (LAROCA et al., 2018)



Fonte: Adaptado de Laroca et al. (2018)

Os resultados obtidos por Laroca et al. (2018) no *dataset* UFPR-ALPR e a comparação com os sistemas comerciais Sighthound e OpenALPR estão apresentados na Tabela 3:

Tabela 3 – Resultados obtidos por Laroca et al. (2018) e pelos sistemas comerciais Sighthound e OpenALPR no *dataset* UFPR-ALPR

Sistema	Sucesso no reconhecimento (%) Sem redundância	Sucesso no reconhecimento (%) Com redundância
Sighthound	47,39	56,67
OpenALPR	50,94	70,0
(LAROCA et al., 2018)	64,89	78,33

Fonte: (LAROCA et al., 2018)

O sistema proposto conseguiu superar os dois sistemas comerciais usados na comparação. Além disso, Laroca et al. (2018) fizeram experimentos usando o SSIG-ALPR (GONÇALVES et al., 2016), um outro *dataset* de placas brasileiras menos desafiador que o UFPR-ALPR. O SSIG-ALPR é menos desafiador pois possui menos diversidade de *backgrounds*. Nessa base o sistema proposto também superou os sistemas comerciais. É importante enfatizar que embora os sistemas comerciais não tenham sido ajustados para a base UFPR-ALPR, eles usam conjuntos de dados privados muito maiores, o que é uma grande vantagem especialmente em abordagens de aprendizado profundo (LAROCA et al., 2018)

Apesar dos bons resultados obtidos por (LAROCA et al., 2018), a abordagem proposta possui algumas limitações: o sistema é capaz de reconhecer apenas um padrão de placa, embora os autores afirmem que ele seja facilmente adaptado a outros. Além disso, a etapa de detecção e

reconhecimento dos caracteres usa três modelos, isso torna o sistema mais robusto, mas aumenta o custo computacional e tempo de treinamento.

3.2.2 *Real-time Automatic License Plate Recognition Through Deep Multi-Task Networks*

A base do sistema proposto por [Gonçalves et al. \(2018\)](#) é o uso de apenas dois modelos profundos para detecção da placa e reconhecimento dos caracteres. Isso é feito com o objetivo de reduzir a propagação de erro inerente em abordagens que dividem o ALPR em múltiplas sub tarefas. Essas sub tarefas normalmente são (I) detecção do veículo; (II) detecção da placa; (III) segmentação dos caracteres; e (IV) reconhecimento óptico dos caracteres (OCR). Esse tipo de abordagem tem um importante efeito negativo pois os erros de cada sub tarefa são propagadas para o próximo estágio da *pipeline* ALPR. Por conta disso, a junção de todas as sub tarefas tendem a ter uma alta taxa de erro, mesmo quando cada etapa é quase perfeita se avaliada isoladamente. Por exemplo, considerando um sistema que utiliza todos os estágios citados anteriormente (I, II, III, IV) e que possui 0,98 de acurácia em todos eles. Se usarmos esse sistema em placas de sete caracteres, a acurácia final será $0,98^2 * 0,98^7 * 0,98^7$, que pode ser expressado como $0,98^{16}$ ou 0,724. Isso representa uma taxa de erro de 0,276, que inviabiliza a utilização do sistema em algumas aplicações reais.

[Gonçalves et al. \(2018\)](#) propuseram uma nova abordagem *end-to-end* para ALPR que reduz o impacto da propagação de erro mencionada acima e é capaz de ser executada em tempo real. Para isso, apenas duas redes profundas em cascata abrangem todas as etapas do ALPR. Enquanto a primeira rede é responsável por detectar as placas de licença automotiva diretamente na imagem de entrada, ignorando a necessidade de detectar o veículo, a segunda rede recebe as imagens das placas fornecidas pela primeira rede e gera a identificação da placa (em formato de texto).

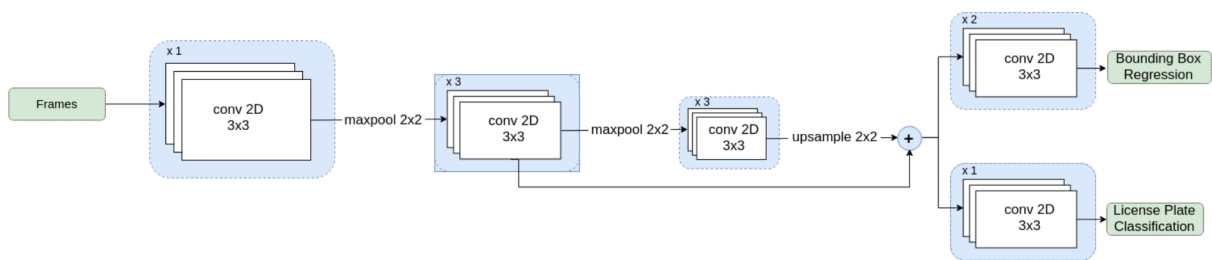
Existem quatro contribuições principais no trabalho de [Gonçalves et al. \(2018\)](#):

- Uma nova rede convolucional profunda projetada especificamente para detectar placas de veículos, que usa uma nova função de perda;
- Um modelo multitarefa capaz de localizar, segmentar e reconhecer caracteres de placas de veículos;
- Uma nova base de dados, que contém 8683 imagens de placas de veículos;
- Duas técnicas de *data augmentation*, que aumentam significativamente os dados de treinamento.

Abordagens anteriores trataram a detecção de placas de veículos de maneira semelhante à detecção geral de objetos. No entanto, [Gonçalves et al. \(2018\)](#) afirmam que uma diferença

fundamental na detecção de placas de veículos é que suas *bounding boxes* só podem ser consideradas corretas se incluírem todos os caracteres. Alguns métodos propõem o aumento arbitrário na borda das *bounding boxes* para garantir que todos os caracteres sejam visíveis (LAROCA et al., 2018), mas Gonçalves et al. (2018) afirmam que essa não é a melhor maneira de lidar com o problema. Em vez disso, uma nova função de perda é utilizada para penalizar as placas mal segmentadas (que não englobam todos os caracteres da placa). O modelo proposto, cuja arquitetura está apresentada na Figura 27, herda muitas ideias de soluções anteriores para detecção de objetos, mas possui aprimoramentos específicos para placas de veículos, como o fato de que a proporção e escala das placas não variarem tanto. Gonçalves et al. (2018) fizeram experimentos com e sem a função de perda proposta e o modelo teve uma acurácia 2,6% maior com o uso dela.

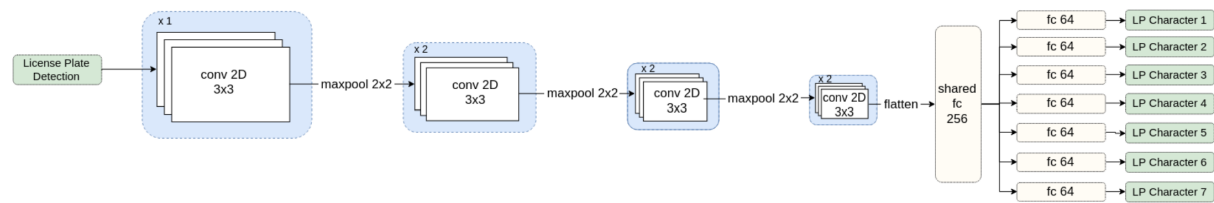
Figura 27 – Arquitetura do modelo proposto por Gonçalves et al. (2018) para detecção de placas



Fonte: Gonçalves et al. (2018)

O modelo responsável por fazer o reconhecimento dos caracteres consiste em uma rede convolucional profunda multi-tarefa. O modelo recebe uma imagem da placa de licença como entrada e produz os sete caracteres sem qualquer etapa de segmentação explícita. Redes multitarefas utilizam a hipótese de que é possível aprender uma representação conjunta que é útil para descrever mais de uma tarefa na mesma imagem (ZHANG; YANG, 2017). No modelo proposto, cada tarefa é a classificação de um dos caracteres da placa. Essas tarefas são muito correlacionadas, pois cada transformação, como translação ou rotação, de um caractere também é aplicada nos caracteres seguintes. Como o objetivo final é classificar os caracteres da placa do veículo, o uso de camadas convolucionais compartilhadas é empregado porque uma representação única deve fornecer boas descrições dos caracteres para cada imagem. Além disso, o modelo foi treinado para reconhecer todos os caracteres das placas do veículo simultaneamente, em vez de empregar duas técnicas separadas (ou seja, uma rede para segmentação e uma rede para OCR), o que aumentaria a propagação do erro.

Figura 28 – Arquitetura do modelo proposto por [Gonçalves et al. \(2018\)](#) para reconhecimento dos caracteres



Fonte: [Gonçalves et al. \(2018\)](#)

Um grande desafio para treinar a arquitetura de rede proposta é que cada tarefa deve aprender a representação de cada letra ou número, por exemplo, a primeira saída da rede deve ser treinada com exemplos de A a Z. No entanto, é extremamente difícil coletar um conjunto de dados brasileiro em que cada letra aparece pelo menos uma vez em cada uma das três primeiras posições devido à política de alocação de placas de veículos no Brasil. Portanto, para superar esse problema, [Gonçalves et al. \(2018\)](#) aumentaram o conjunto de dados de treinamento fazendo diferentes permutações dos caracteres da placa e criando placas de licença sintéticas. Essas técnicas de *data augmentation* aumentaram a acurácia em 2,4% em relação ao mesmo modelo treinado sem o uso delas.

O sistema proposto por [Gonçalves et al. \(2018\)](#) foi avaliado em dois *datasets* para ALPR, o UFPR-ALPR ([LAROCA et al., 2018](#)) e o SSIG-SegPlate ([GONÇALVES et al., 2016](#)). A Tabela 4 apresenta os resultados obtidos no SSIG-SegPlate, nesse *dataset* o sistema foi capaz de superar os trabalhos anteriores e dois sistemas comerciais (Sighthound e OpenALPR).

Tabela 4 – Resultados da abordagem proposta por [Gonçalves et al. \(2018\)](#) no *dataset* SSIG-SegPlate

Abordagem	Taxa de reconhecimento (%)
Gonçalves et al. (2016)	81,8
Montazzolli e Jung (2017)	63,1
Laroca et al. (2018)	85,4
Sighthound	73,1
OpenALPR	87,4
Gonçalves et al. (2018)	88,8

Fonte: [Gonçalves et al. \(2018\)](#)

Os resultados da abordagem proposta por [Gonçalves et al. \(2018\)](#) no *dataset* UFPR-ALPR estão apresentados na Tabela 5, os resultados do trabalho de [Laroca et al. \(2018\)](#) e do sistema comercial OpenALPR também são apresentados. As imagens de motos foram desconsideradas na comparação, uma vez que a abordagem proposta por [Gonçalves et al. \(2018\)](#) não suporta placas de motos. O conjunto de dados UFPR-ALPR possui imagens com *backgrounds* variados,

o que o torna um conjunto de dados desafiador para a rede de detecção proposta por [Gonçalves et al. \(2018\)](#), visto que ela funciona diretamente nas imagens de entrada e há muitos padrões diferentes nas cenas que podem ser confundidos com uma placa de veículo. Para verificar essa hipótese, [Gonçalves et al. \(2018\)](#) avaliou o sistema apenas na etapa de reconhecimento (passando para ela as placas *ground truth*). Com isso, o desempenho do sistema subiu para 76,5%, o que significa que 20,9% das placas erradas foram devido ao fraco desempenho da rede de detecção.

Tabela 5 – Resultados do sistema proposto por [Gonçalves et al. \(2018\)](#) no *dataset* UFPR-ALPR (desconsiderando imagens de motos)

Abordagem	Taxa de reconhecimento (%)
Laroca et al. (2018)	72,2
OpenALPR	57,9
Gonçalves et al. (2018)	55,6

Fonte: [Gonçalves et al. \(2018\)](#)

Uma limitação da abordagem proposta por [Gonçalves et al. \(2018\)](#) é a dificuldade para aplicar o sistema a mais de um padrão de placa, por exemplo, padrões com menos ou mais de sete caracteres. Além disso, o sistema proposto não se aplica a imagens de motos.

3.2.3 *An Efficient and Layout-Independent Automatic License Plate Recognition System Based on the YOLO detector*

O trabalho de [Laroca et al. \(2019\)](#) propôs um sistema ALPR capaz de identificar placas de diferentes *layouts*. O sistema usa o detector de objeto YOLO ([REDMON et al., 2016](#)) em todos os estágios. Em resumo, as maiores contribuições do trabalho de ([LAROCA et al., 2019](#)) são:

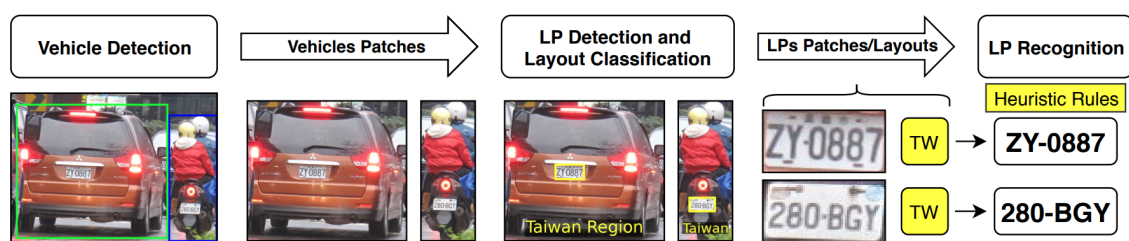
- Um sistema ALPR *end-to-end* capaz de detectar placas de diferentes *layouts*, o que facilita o ajuste do sistema para novos padrões de placas;
- Esse foi o primeiro trabalho a usar um estágio de classificação de *layout* para melhorar o desempenho de reconhecimento do sistema. Isso é alcançado com o uso de regras heurísticas específicas para cada *layout*;
- O sistema proposto foi capaz de realizar inferências a mais de 70 FPS usando uma GPU *high end* (NVIDIA Titan XP), o que indica que diversas aplicações de cenários reais podem usar GPUs *mid end* em tempo real com esse sistema;
- Uso de oito *datasets* públicos (com *layouts* de cinco regiões distintas) no treinamento do sistema. Além disso, [Laroca et al. \(2019\)](#) criaram anotações para 38351 *bounding boxes* de 6239 imagens para os *datasets* com anotações faltando;

- Uma avaliação comparativa detalhada entre o sistema proposto e dois sistemas comerciais nos oito *datasets* usado no treinamento do sistema.

A natureza das imagens de tráfego pode ser muito problemática para abordagens que fazem a detecção da placa diretamente na imagem de entrada (ou seja, sem detecção do veículo primeiro). Isso acontece pois existem muitos blocos textuais que podem ser confundidos com placas, como sinais de trânsito e números de telefone nas vitrines. Além disso, as placas podem ocupar porções muito pequenas da imagem original e os detectores de objetos geralmente têm dificuldade para detectar objetos pequenos (REDMON et al., 2016; GONÇALVES et al., 2019). Por conta disso, a abordagem proposta por (LAROCA et al., 2019) propõe primeiro localizar os veículos na imagem de entrada e, em seguida, detectar suas respectivas placas nas detecções dos veículos.

A Figura 29 apresenta a pipeline proposta por (Laroca et al. (2019)). Cada etapa usa um modelo de detecção de objetos distinto, assim, cada uma pode ser otimizada separadamente. Primeiro, a detecção da placa é feita em dois estágios: o primeiro estágio usa uma YOLOv2 (REDMON; FARHADI, 2017) para detectar o veículo dentro da imagem de entrada e o segundo usa uma *fast* YOLO (versão reduzida da YOLOv2) para detectar a placa dentro da detecção do veículo, ao mesmo tempo que faz a classificação do *layout*. Essa etapa foi treinada para detectar cinco classes de placas: americanas, brasileiras, chinesas, europeias e da região de Taiwan. Esses *layouts* são de oito *datasets* públicos distintos. É importante destacar que placas de diferentes *layouts* (que pertencem à mesma classe/região) podem variar em muitos fatores, por exemplo, na proporção, cores, símbolos, posição dos caracteres, número dos caracteres, entre outros. Laroca et al. (2019) considerou placas de diferentes jurisdições dos Estados Unidos como sendo uma única classe; o mesmo foi feito para placas de países europeus. Placas do mesmo país ou região podem ter algumas diferenças, mas ainda assim compartilham muitas características em comum. A Figura 30 apresenta placas de diferentes *layouts* usadas por Laroca et al. (2019).

Figura 29 – Pipeline proposta por Laroca et al. (2019)



Fonte: Laroca et al. (2019)

Figura 30 – Placas de diferentes *layouts* usadas por Laroca et al. (2019)

Fonte: Laroca et al. (2019)

Em seguida, uma vez que a placa já foi detectada, é feita a detecção e classificação dos caracteres, isso é feito simultaneamente usando uma única CNN. A CR-NET (MONTAZZOLLI; JUNG, 2017), que é uma rede inspirada na arquitetura da Fast YOLO, treinada para essa etapa possui 35 classes (0-9, A-Z, em que o número "0" é classificado juntamente com a letra "O"). Por fim, são usadas regras heurísticas específicas para cada *layout* para obtenção do resultado final.

A Tabela 6 apresenta os resultados obtidos por Laroca et al. (2019) nos oito *datasets* usados, juntamente com o desempenho de dois sistemas comerciais.

Tabela 6 – Resultados obtidos por Laroca et al. (2019) nos oito *datasets* usados, juntamente com o desempenho de dois sistemas comerciais.

<i>Dataset</i>	Sighthound	OpenALPR	Laroca et al. (2019)
Caltech Cars	95,7	99,1	98,7
EnglishLP	92,5	78,6	95,7
UCSD-Stills	98,3	98,3	98,0
ChineseLP	90,4	92,6	97,5
AOLP	87,1	—	99,2
OpenALPR-EU	93,5	91,7	97,8
SSIG-SegPlate	82,8	92,0	98,2
UFPR-ALPR	62,3	82,2	90,0
Média	87,8	90,7	96,9

Fonte: Laroca et al. (2019)

3.2.4 Outros trabalhos

Muitos trabalhos da literatura utilizam CNNs específicas para detecção de objetos na etapa de detecção da placa. Montazzolli e Jung (2017) realizaram experimentos utilizando apenas uma Fast YOLO (REDMON et al., 2016) para detectar a placa diretamente na imagem de entrada,

sem um estágio inicial de detecção do veículo. Eles perceberam que essa abordagem é incapaz de detectar muitas placas, atingindo um *recall* baixo. Isso pode ter acontecido por conta do tamanho da placa em relação a imagem de entrada, uma vez que uma das limitações conhecidas da YOLO é a dificuldade para detectar objetos pequenos (REDMON et al., 2016). Montazzolli e Jung (2017) decidiram então usar dois modelos Fast YOLO em cascata, de modo que o primeiro detecta o veículo na imagem de entrada e o segundo detecta a placa usando a detecção do estágio anterior. Essa abordagem atingiu boas taxas de precisão e *recall* no *dataset* SSIG-ALPR (GONÇALVES et al., 2016).

Montazzolli e Jung (2017) usaram um modelo baseado na YOLO para detectar e reconhecer simultaneamente todos os caracteres dentro de uma imagem cortada de placa. Embora tenham sido obtidas taxas de FPS impressionantes (448 FPS em uma GPU *high end* - NVIDIA TITAN X) nos experimentos realizados no conjunto de dados SSIG-SegPlate (GONÇALVES et al., 2016), menos de 65% das placas foram corretamente reconhecidas. De acordo com os autores, o gargalo da abordagem era o reconhecimento de letras, visto que o conjunto de caracteres de treinamento era altamente desbalanceado.

Kurpiel, Minetto e Nassu (2017) particionaram a imagem de entrada em formando um grid de regiões sobrepostas. Em seguida, aplicaram uma CNN para produzir um *score* para cada região e as placas foram detectadas combinando a saída de regiões vizinhas. Essa abordagem atingiu 83% de *recall* em um *dataset* público introduzido por eles e 230 ms de tempo de inferência em uma GPU GT-740M.

Li et al. (2018) treinaram CNNs usando imagens de textos em geral para detectar a placa do veículo com base em agrupamentos de caracteres. A rede treinada é aplicada usando uma abordagem de janela deslizante na imagem de entrada, a partir disso é gerado um mapa de saliências de regiões candidatas a placas de licença. Essas regiões são aquelas nas quais a CNN notou a presença de grupos de caracteres. Em seguida, uma segunda CNN que é treinada para classificar uma região como placa ou não placa é empregada nas regiões para reduzir a quantidade de falsos positivos. Apesar dessa abordagem ter um bom *recall*, ela não se aplica a abordagens focadas em tempo real, uma vez que cada imagem leva 2 s para ser analisada.

Silva e Jung (2020) retreinaram um modelo com um conjunto de treinamento ampliado composto de imagens reais e geradas artificialmente usando fontes semelhantes aos de placas de certas regiões. Dessa forma, a rede treinada se tornou muito mais robusta para a detecção e classificação de caracteres reais, superando trabalhos anteriores e sistemas comerciais em três conjuntos de dados públicos.

4

Construção da Base de Dados

No aprendizado de máquina, os dados desempenham papel fundamental, pois é a partir deles que os modelos aprendem a reconhecer padrões. Por outro lado, eles são também uma barreira. Segundo [Cognilytica \(2019\)](#), em média 80% do tempo gasto em projetos de *machine learning* é destinado a tarefas envolvendo dados, sendo a maior parte disso gasto na rotulação. Nesse contexto, a criação de novas bases de dados, com os rótulos apropriados, exerce grande impacto e fomento ao desenvolvimento do aprendizado de máquina ([SUN et al., 2017](#)). Devido à grande importância e necessidade de bases de dados, especialmente de imagens de placas brasileiras, decidimos contribuir com a criação de uma base de dados original que pudesse ser usada no desenvolvimento de sistemas ALPR.

Este capítulo apresenta o detalhamento do processo de construção de uma base de dados específica para ALPR, a qual intitulamos UFS-ALPR. A Seção 4.1 apresenta outras bases de dados semelhantes, com imagens de placas de brasileiras. A Seção 4.2 descreve como foi feito o levantamento, preparação e rotulação dos dados, destacando as condições e ferramentas utilizadas. Por fim, a Seção 4.3 apresenta estatísticas acerca da base construída.

4.1 Outras bases brasileiras para ALPR

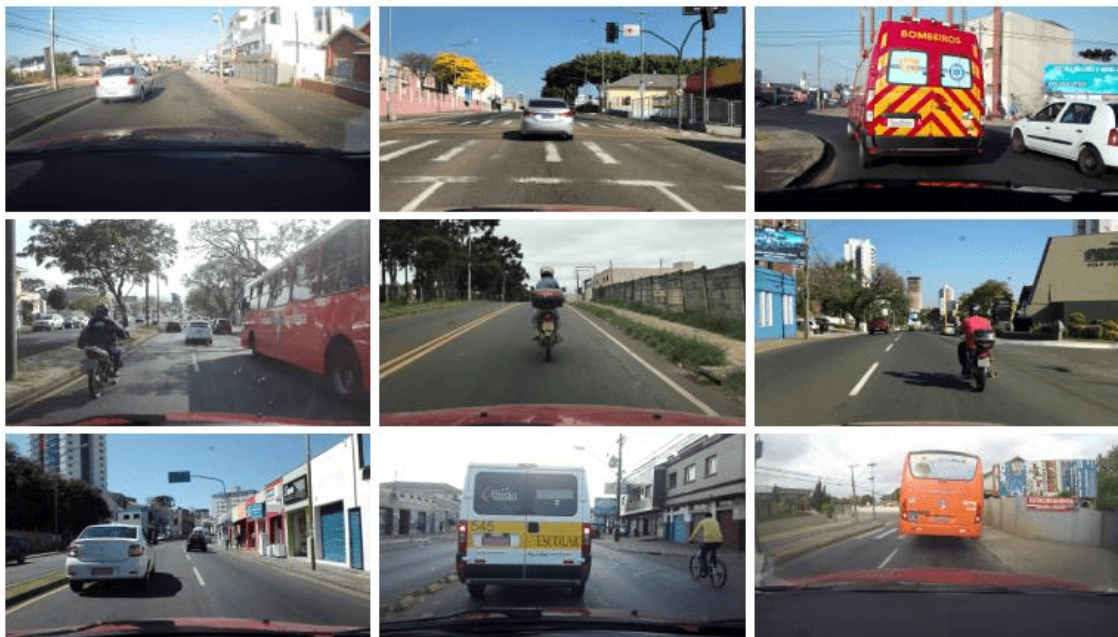
4.1.1 UFPR-ALPR

O *dataset* UFPR-ALPR ([LAROCA et al., 2018](#)) está disponível publicamente para propósitos acadêmicos e possui apenas imagens de placas de licença automotiva no padrão brasileiro (três letras seguidas de um espaço e quatro dígitos). Ele é composto de 4.500 imagens de 150 veículos diferentes (são 30 *frames* por veículo), que contém imagens de carros, motos, caminhões e ônibus, além disso, possui placas cinzas e vermelhas. As *labels* dessa base de dados estão em formato de texto e descrevem as *bounding boxes* do veículo, da placa e de cada caractere dela. Além das *bounding boxes*, as *labels* indicam também o ano, modelo, marca e tipo

do veículo (o tipo pode ser carro, moto, ônibus ou caminhão).

O *dataset* possui três partições: treinamento (1.800 imagens de 60 veículos), validação (900 imagens de 30 veículos) e teste (1.800 imagens de 60 veículos). Uma limitação dessa, e de outras bases desse tipo é que as rotulações são apenas do veículo mais destacado na imagem, apesar de em algumas situações outros veículos aparecerem no *background*. As imagens foram adquiridas usando três câmeras diferentes e estão disponíveis no formato *portable network graphics* (PNG) com tamanho de 1920×1080 pixels. As imagens foram capturadas de dentro de um veículo em movimento.

Figura 31 – Exemplos de amostras da base de dados UFPR-ALPR



Fonte: [Laroca et al. \(2018\)](#)

4.1.2 SSIG-ALPR

O SSIG-ALPR foi proposto por [Gonçalves et al. \(2018\)](#), ele contém 6660 imagens divididas em 3595 imagens de treinamento, 2360 amostras de teste e 705 amostras de validação. São 8683 matrículas (das quais 3368 não possuem rotulação para o texto da placa) de 815 veículos diferentes. As imagens do *dataset* estão em Full-HD e estão disponíveis no formato PNG com tamanho de 1920×1080 pixels cada. O *dataset* inteiro contém um total de 35 gigabytes.

As imagens são apenas de placas do padrão brasileiro (três letras seguidas de um espaço e quatro dígitos). Além disso, os rótulos são apenas para as *bounding boxes* das placas de licença e seu texto, não existindo rótulos para as *bounding boxes* do veículo e de cada um dos caracteres da placa. A coleta das imagens foi feita usando duas câmeras: uma estática e a outra dentro de um veículo em movimento.

Figura 32 – Amostras da base de dados SSIG-ALPR



Fonte: Adaptado de [Gonçalves et al. \(2018\)](#)

4.2 Levantamento, preparação e rotulação dos dados

As imagens foram obtidas em diferentes condições de iluminação nos períodos da manhã, tarde e noite na zona urbana das cidades sergipanas Aracaju e São Cristóvão. Para aumentarmos a diversidade, elas foram capturadas de maneiras variadas. Em alguns momentos os veículos foram gravados de dentro de um carro em movimento e em outros, a câmera foi fixada e gravou veículos se aproximando, se distanciando ou estacionados. As imagens capturadas são da parte frontal e traseira dos veículos, uma variação que não ocorre em outros *datasets*, como o UFPR-ALPR.

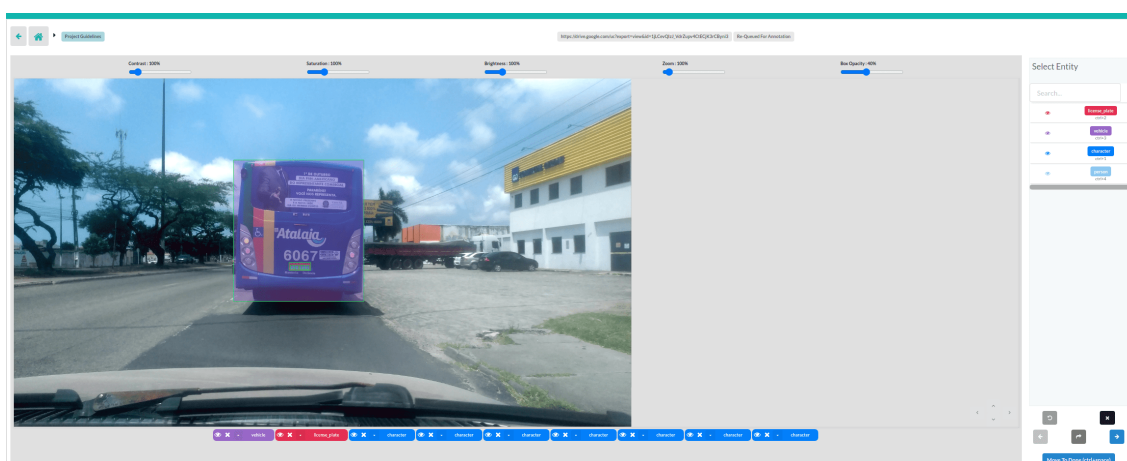
Foram utilizadas as câmeras de dois smartphones (Redmi Note 7 Pro e Redmi Note 8) que gravam em Full-HD (1920x1080) a uma taxa de 30 quadros por segundo. A base construída possui uma boa variedade de *backgrounds*, visto que as imagens foram capturadas em diferentes locais. Além disso, temos situações em que a câmera e o veículo sendo filmado estavam em movimento, em que apenas o veículo se movia e em que câmera e veículo estavam parados.

As filmagens foram feitas de modo que um veículo fosse enquadrado por vez. Fizemos dessa forma para que apenas um veículo precisasse ser rotulado por imagem, porém, em algumas imagens outros veículos estão no *background*, esses estão geralmente obstruídos, muito pequenos ou sem placa visível e por isso não foram rotulados. As filmagens brutas foram analisadas e os

frames em que a placa do veículo estava focada e legível foram selecionados. A quantidade de *frames* selecionados por veículo variou entre 2 e 58 (média de 15 *frames* por veículo).

A base de dados possui ao todo 5029 imagens de 326 veículos. Além disso, a base de dados possui imagens de placas do novo padrão MERCOSUL e do padrão brasileiro. Os 326 veículos foram rotulados separadamente em tipo (carro, moto, ônibus ou caminhão), fabricante, modelo, padrão de placa (MERCOSUL ou antigo) e se a imagem era da parte frontal ou traseira do veículo. Além disso, as *bounding boxes* do veículo, da placa e de cada caractere dela, foram anotadas para todas as 5029 imagens (um veículo por imagem). Nessa etapa utilizamos o software *open source* Dataturks¹, a Figura 33 apresenta sua interface de rotulação. A base de dados possui ao todo 15 gigabytes.

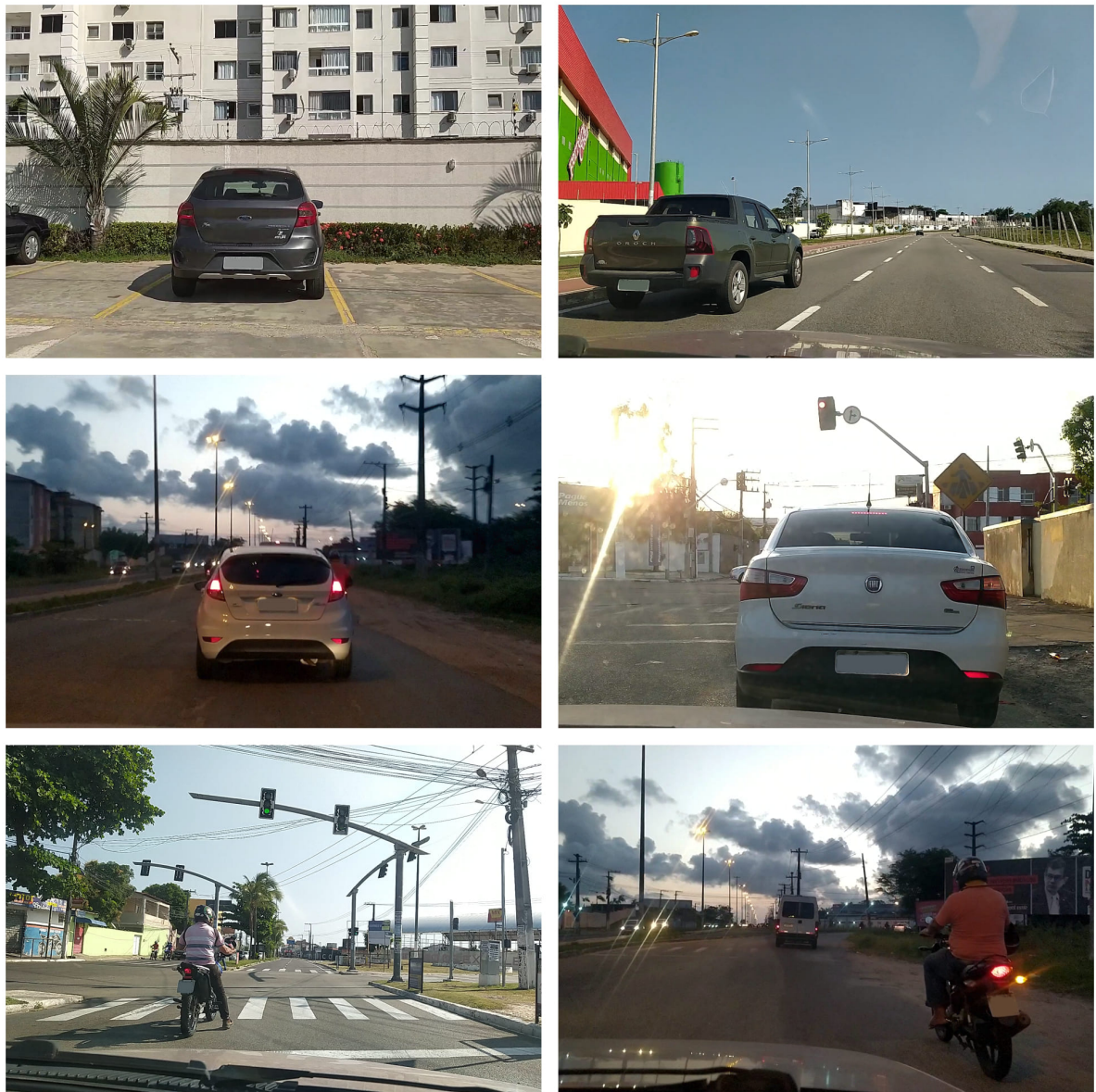
Figura 33 – Interface de rotulação do Dataturks¹



Fonte: Elaborado pelo autor

¹ Disponível em: <https://github.com/DataTurks/DataTurks>. Acesso em novembro de 2020

Figura 34 – Amostras da base de dados UFS-ALPR

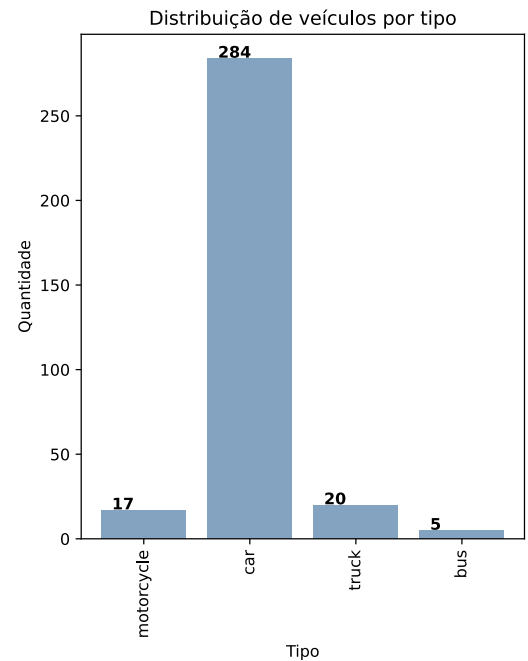


Fonte: Elaborado pelo autor

4.3 Estatísticas

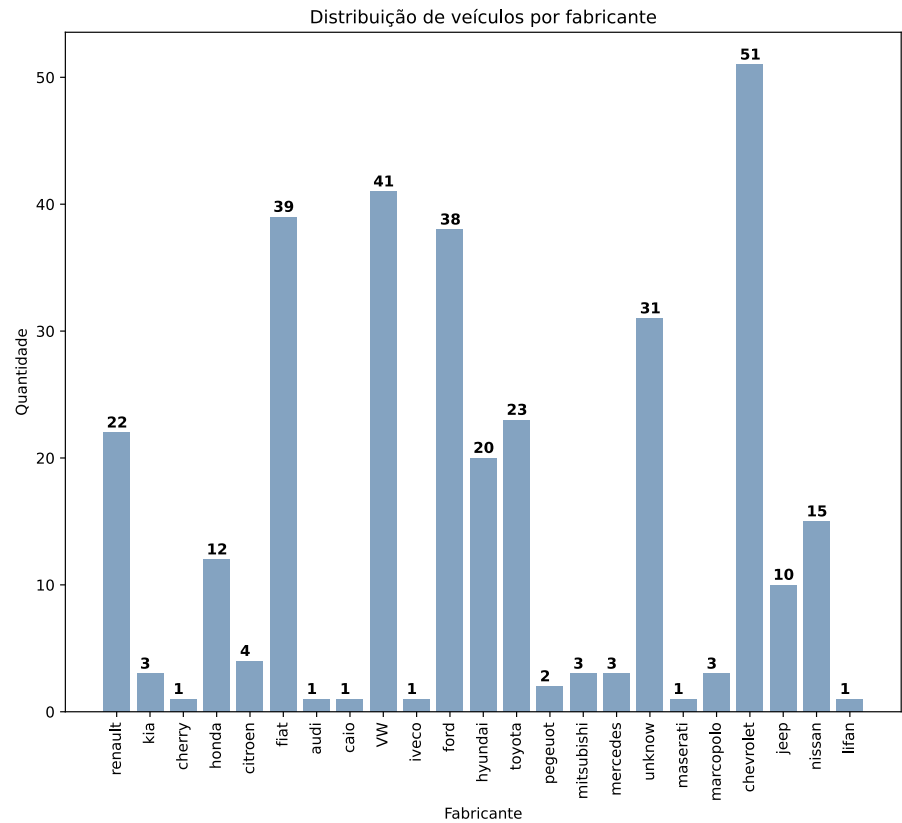
Dos 326 veículos, 116 (1770 imagens) possuem placa do novo padrão MERCOSUL e 210 (3259 imagens) do padrão brasileiro. 149 (1912 imagens) foram filmados frontalmente e 177 (3117 imagens) por trás.

Figura 35 – Distribuição de veículos por tipo



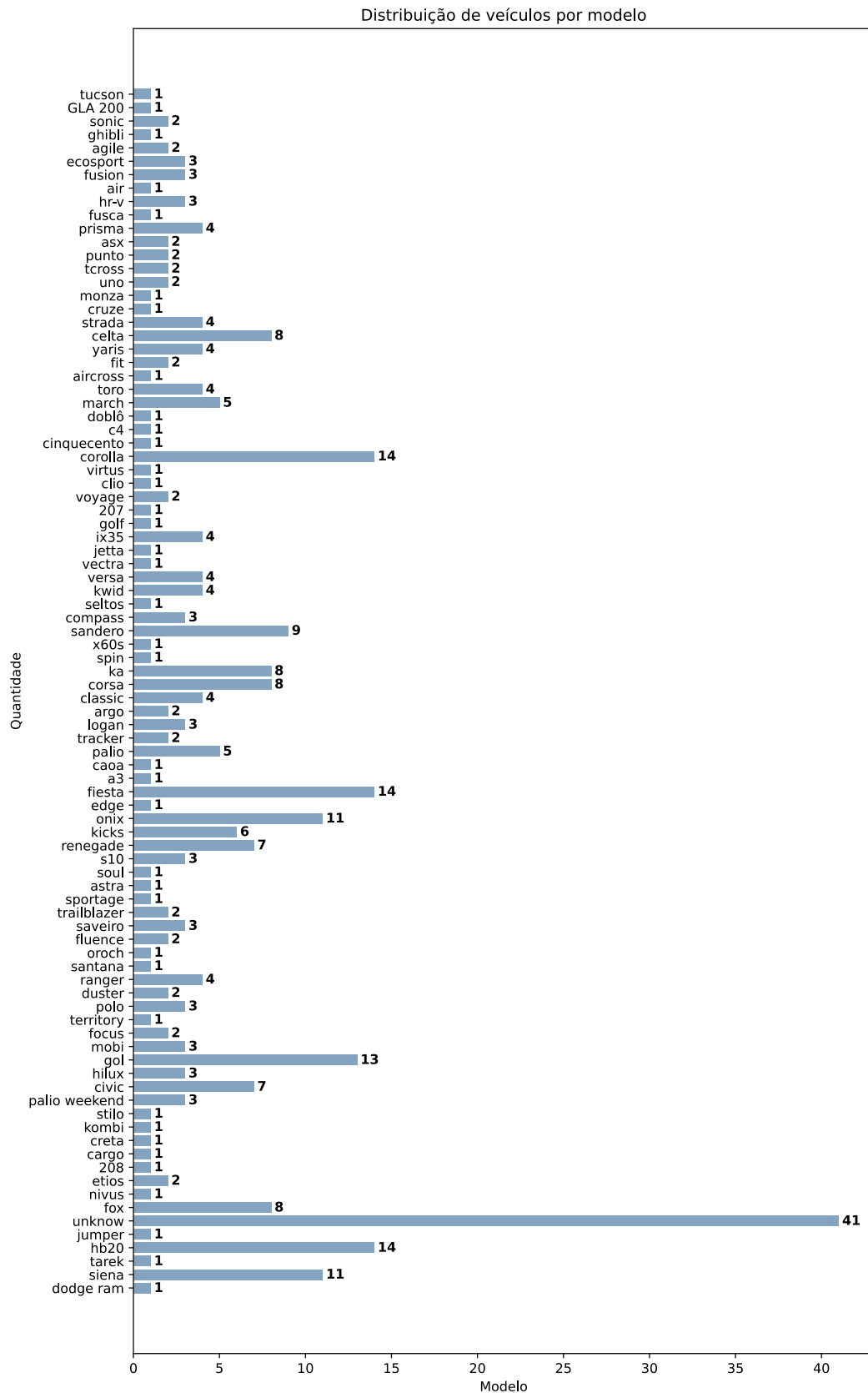
Fonte: Elaborado pelo autor

Figura 36 – Distribuição de veículos por fabricante



Fonte: Elaborado pelo autor

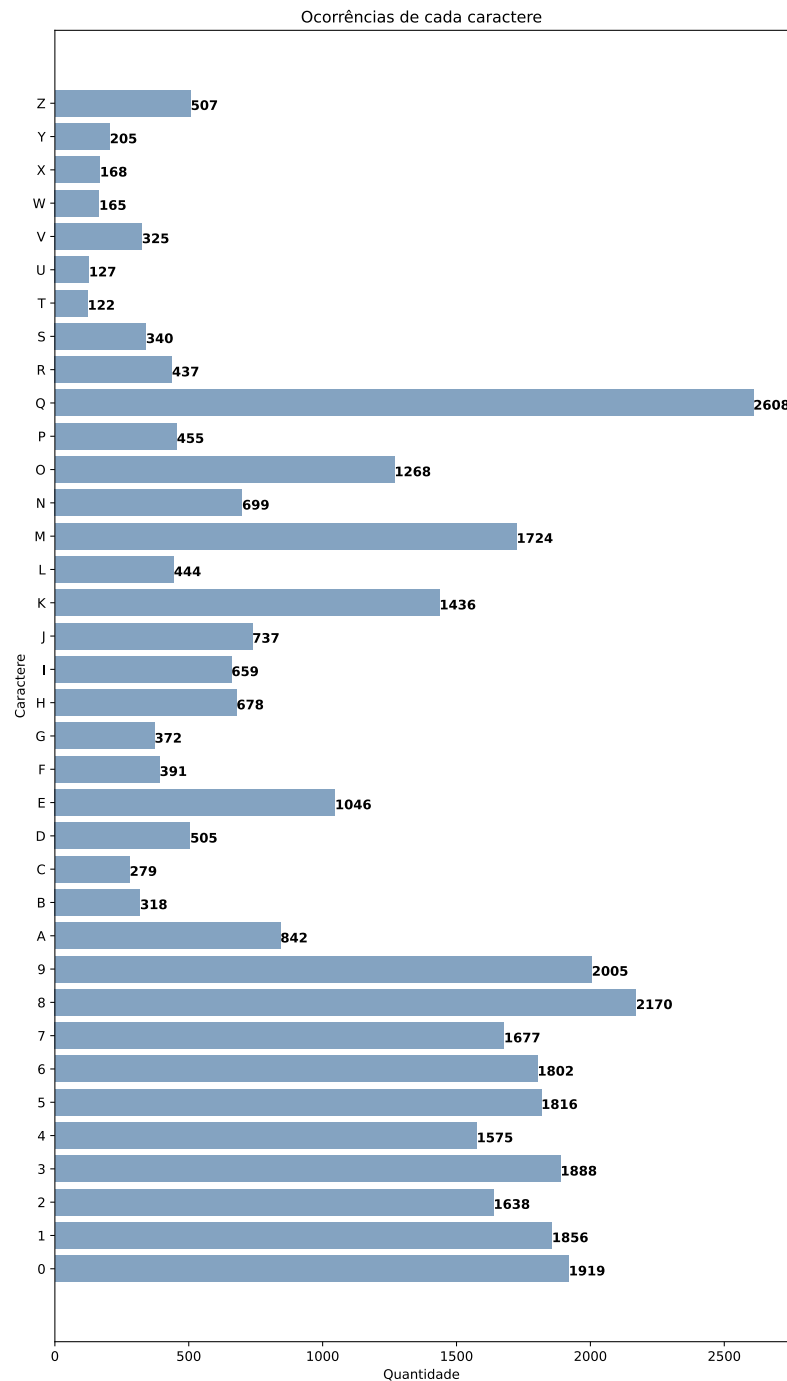
Figura 37 – Distribuição de veículos por modelo



Fonte: Elaborado pelo autor

No Brasil, cada estado usa letras iniciais específicas para suas placas, o que resulta em uma faixa. Em Sergipe (onde o *dataset* foi coletado), as placas variam nas seguintes faixas: HZB 0001 a IAP 9999, NVG 0001 a NVN 9999, OEJ 0001 a OES 9999, OZB 0001 a OZB 9999, QKN 0001 a QKZ 9999 e QMA 0001 a QMP 9999. Portanto, as letras dessas faixas têm mais exemplos do que as outras, conforme mostrado na Figura 38.

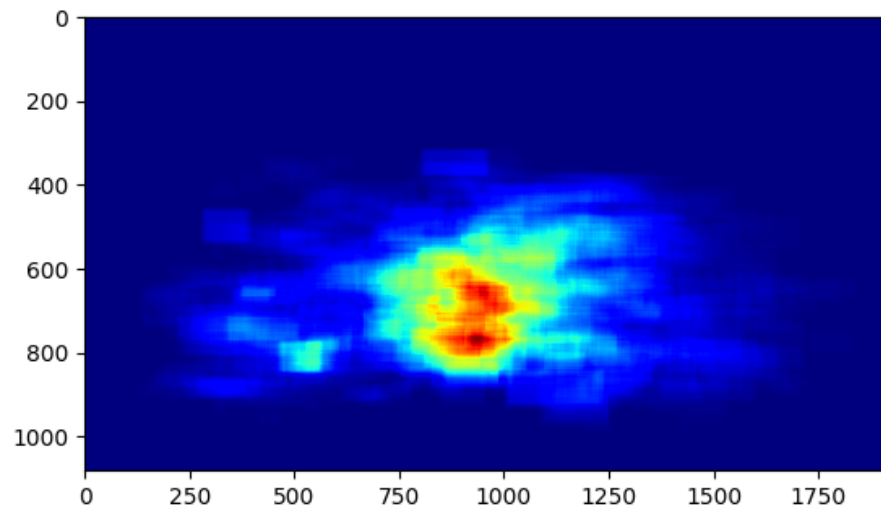
Figura 38 – Ocorrências dos caracteres



Fonte: Elaborado pelo autor

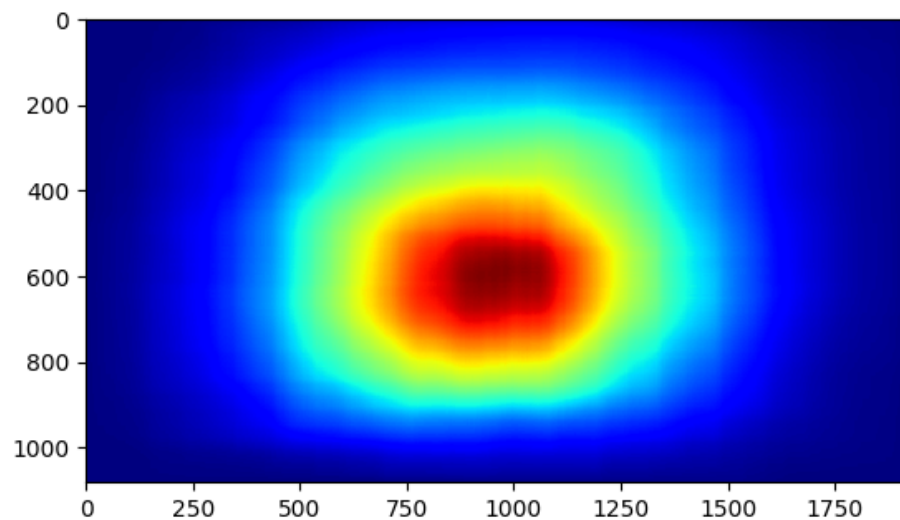
Os mapas de calor das Figuras 39 e 40 respectivamente, representam as localizações onde as placas de licença e os veículos ocorrem com mais frequência nas imagens da base de dados. Podemos notar que os veículos estão em sua maioria localizados no centro das imagens e as placas no centro dos veículos.

Figura 39 – *Heat map* das placas



Fonte: Elaborado pelo autor

Figura 40 – *Heat map* dos veículos



Fonte: Elaborado pelo autor

5

Desenvolvimento e Resultados

Este capítulo apresenta o desenvolvimento do sistema ALPR proposto. A Seção 5.1 descreve o cenário de uso pensado para o sistema, os aspectos gerais da *pipeline* de reconhecimento e justificativas para decisões que foram tomadas. Essa seção apresenta também os modelos utilizados nos nossos experimentos, cujo foco foram cinco diferentes arquiteturas: YOLOv4 Tiny (BOCHKOVSKIY; WANG; LIAO, 2020), YOLOv5 Nano (JOCHER et al., 2022), YOLOv5 Small (JOCHER et al., 2022), SSD (LIU et al., 2016) e CenterNet (LAW; DENG, 2018). O desempenho dessas arquiteturas foi avaliado em cada uma das etapas de reconhecimento e utilizado como base para escolha dos modelos que iriam compor o sistema.

Na Seção 5.2 é feito o detalhamento da forma como os modelos foram treinados e, a seguir, são apresentadas métricas de cada um deles. A implementação do sistema ALPR é descrita na Seção 5.3, que também apresenta métricas de avaliação do mesmo em relação a cada uma das arquiteturas experimentadas. Por fim, a Seção 5.4 discorre sobre a implantação do sistema em dois dispositivos com recursos limitados: Raspberry Pi 4 e NVIDIA Jetson Nano. Essa última etapa envolveu a configuração dos dispositivos, conversão dos modelos para formatos mais otimizados e coleta de métricas de execução do sistema.

5.1 Sistema ALPR proposto

5.1.1 Cenário de uso do sistema

O ALPR é um problema com diversos cenários de uso. Uma vez que o sistema seja capaz de extrair as informações da placa de um veículo, ele pode ser adaptado a diferentes situações. De maneira ampla, o ALPR pode auxiliar agências governamentais na busca por carros envolvidos em crimes, aplicação das leis de trânsito, controle de fronteira e monitoramento do tráfego rodoviário. Também podendo ser utilizado por empresas privadas para fins comerciais, como cobrança automática de pedágio e controle de acesso a zonas privadas.

Nesse trabalho, o caso de uso pensado para utilização do sistema proposto foi o controle de acesso a zonas privadas, mais especificamente portarias de condomínios, guaritas ou estacionamentos. Esse tipo de cenário geralmente possui um ambiente mais controlado, uma vez que os veículos passam pela entrada em baixa velocidade e geralmente apenas um veículo passa por vez.

5.1.2 Pipeline de reconhecimento

Um sistema ALPR tipicamente possui três etapas: Detecção da placa, detecção dos caracteres e reconhecimento dos caracteres da placa. Muitas abordagens dividem a detecção da placa em dois estágios: primeiro é feita a detecção do veículo na imagem de entrada, depois a placa é detectada dentro da detecção do veículo. Nosso sistema segue essa linha de pensamento e se baseia no uso de modelos para detecção de objetos específicos para cada uma das etapas. A *pipeline* proposta se assemelha muito com a de [Laroca et al. \(2019\)](#), tendo três etapas principais:

1. Detecção do veículo: O primeiro estágio é responsável por fazer a detecção do veículo dentro da imagem de entrada. Durante nossos experimentos, adaptamos diferentes modelos para fazer a detecção de uma classe de objeto (veículo). Além disso, fizemos experimentos utilizando duas classes: uma para detecção de motocicletas e outra para os outros tipos de veículos (carros, ônibus e caminhões). Contudo, devido à baixa quantidade de imagens de motocicletas na nossa base de dados, essa abordagem teve resultados piores do que utilizando apenas uma classe. Por conta disso, decidimos focar os experimentos em apenas uma classe. Nessa etapa inicial, a *bounding box* com maior *score* de confiança é selecionada e as outras detecções são ignoradas. Isso foi feito pois cada imagem da base de dados tem apenas um veículo rotulado e o cenário de uso pensado para utilização do sistema (controle de acesso em uma portaria) limita-se, em geral, a um veículo por vez. A *bounding box* do veículo detectado é usada para recortar a imagem de entrada. Em seguida, esse recorte é passado para o próximo estágio.

Figura 41 – Exemplo de entrada e saída do estágio de detecção do veículo



Fonte: Elaborado pelo autor

2. Detecção e classificação de *layout* da placa: Para essa etapa, além de detectar a placa, realizamos a classificação do seu *layout*. Esse tipo de estratégia foi inicialmente proposta

por [Laroca et al. \(2019\)](#) e consiste no uso de modelos capazes de detectar e diferenciar vários tipos de placas. Nesse trabalho, como a base de dados que construímos possui dois tipos de placas brasileiras — padrão brasileiro e padrão MERCOSUL —, adaptamos os modelos para realizar a detecção dessas duas classes. A Figura 43 apresenta exemplos de cada tipo de placa. A etapa de classificação de *layout* torna o sistema mais robusto, uma vez que com essa informação é possível utilizar regras específicas para cada *layout* de placa e reduzir os erros no reconhecimento. Por exemplo, sabemos que as placas do padrão brasileiro são do formato LLL-NNNN (três letras seguidas de quatro números), assim, caso a etapa de reconhecimento aponte um número no lugar de uma letra (ou o contrário), podemos aplicar regras de substituição que melhoram o desempenho do sistema. Após a detecção e classificação do *layout*, uma vez que cada veículo tem apenas uma placa, selecionamos apenas a detecção com o maior *score* de confiança. Essa detecção é utilizada para recortar a placa na imagem do veículo e, em seguida, é passada como entrada para o próximo estágio. No Brasil, as placas de motos possuem uma proporção próxima a de um quadrado, enquanto que as de outros tipos de veículos possuem proporção mais retangular. No nosso sistema, utilizamos a proporção altura/largura para diferenciar entre motos e demais tipos de veículos, essa informação nos permite utilizar regras específicas para motos e melhorar o reconhecimento.

Figura 42 – Exemplo de entrada e saída do estágio de detecção da placa



Fonte: Elaborado pelo autor

Figura 43 – Tipos de placas brasileiras



(a) Padrão brasileiro



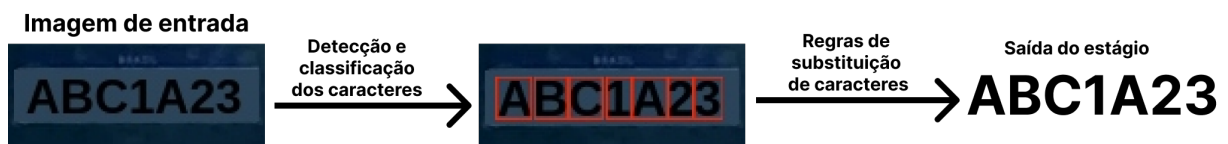
(b) Padrão MERCOSUL

Fonte: Elaborado pelo autor

3. Detecção e reconhecimento dos caracteres da placa: Nesse último estágio, os modelos foram adaptados para detectar caracteres em 35 classes (0-9, A-Z, em que o número "0" é classificado juntamente com a letra "O"). Essa etapa recebe como entrada a placa

do veículo, que foi recortada na etapa anterior. Como as placas brasileiras possuem exatamente sete caracteres, as sete *bounding boxes* com os maiores *scores* de confiança são selecionadas. Em seguida, elas são ordenadas de acordo com as coordenadas da *bounding box*. Combinando a informação da posição do caractere com o *layout* da placa (obtido na etapa anterior) é possível saber se o caractere deve ser um número ou uma letra. Por fim, são utilizadas regras de substituição para reduzir erros no reconhecimento. Foram utilizadas as seguintes regras de substituição: letras em posição de números ($A \rightarrow 4$, $B \rightarrow 8$, $D \rightarrow 0$, $G \rightarrow 6$, $I \rightarrow 1$, $J \rightarrow 1$, $Q \rightarrow 0$, $S \rightarrow 5$, $Z \rightarrow 7$), números em posição de letras ($0 \rightarrow O$, $1 \rightarrow I$, $2 \rightarrow Z$, $6 \rightarrow G$, $7 \rightarrow Z$, $8 \rightarrow B$). A escolha dessas regras se baseou na semelhança dos caracteres e em outros trabalhos da literatura (LAROCA et al., 2019).

Figura 44 – Exemplo de entrada e saída do estágio de detecção e reconhecimento dos caracteres



Fonte: Elaborado pelo autor

5.1.3 Escolha dos modelos

Como nosso objetivo era criar um sistema capaz de ser implantado em dispositivos com recursos limitados, uma das nossas preocupações foi definir uma *pipeline* que exigisse poucos recursos computacionais. Por conta disso, decidimos utilizar modelos de detecção de objetos que fossem leves, isto é, que atingissem *frame rates* razoáveis sem a necessidade de uma GPU em um computador convencional. Num primeiro momento, fizemos a pesquisa de diferentes arquiteturas e avaliamos esses modelos em relação a requisitos de acurácia e tempo de processamento. Nessa pesquisa, focamos em modelos de detecção de objetos de um estágio, uma vez que esse tipo de modelo costuma ter um tempo de processamento menor. Também levamos em consideração arquiteturas que já haviam sido usadas por trabalhos anteriores no desenvolvimento de sistemas ALPR, buscando utilizar versões reduzidas delas sempre que possível.

A família de modelos YOLO já foi amplamente utilizada em trabalhos anteriores no desenvolvimento de sistemas ALPR (LAROCA et al., 2018; LAROCA et al., 2019; JAMTSHO; RIYAMONGKOL; WARANUSAST, 2020). Contudo, a maioria dos trabalhos explora apenas as versões convencionais da YOLO sem realizar testes com as versões reduzidas. Nos nossos experimentos, decidimos focar na utilização das versões mais recentes da YOLO (YOLOv4 (BOCHKOVSKIY; WANG; LIAO, 2020) e YOLOv5 (JOCHER et al., 2022)) e avaliar as variações reduzidas dessas arquiteturas — YOLOv4 Tiny, YOLOv5 Nano e YOLOv5 Small.

Uma das nossas fontes de pesquisa foi a API de detecção de objetos do TensorFlow¹, uma biblioteca mantida pelo TensorFlow (ABADI et al., 2015) que disponibiliza rotinas para treinamento e inferência de diversas arquiteturas de detecção de objetos. Atualmente, a API possui 42 modelos disponíveis para o TensorFlow 2.x e informa para cada um deles a mAP do modelo no *dataset* COCO (LIN et al., 2014) e o tempo de inferência, contudo, não especificam o hardware utilizado para calcular esse tempo de inferência. Avaliamos a lista de modelos disponíveis e escolhemos os dois modelos mais rápidos, que foram SSD MobileNetV2 320x320 e CenterNet MobileNetV2 FPN 512x512, com tempo de inferência de 19ms e 6ms e mAP de 22,2% e 23,4%, respectivamente.

A partir da nossa pesquisa selecionamos os seguintes modelos: YOLOv4 Tiny (BOCHKOVSKIY; WANG; LIAO, 2020), YOLOv5 Nano (JOCHER et al., 2022), YOLOv5 Small (JOCHER et al., 2022), SSD (LIU et al., 2016) e CenterNet (LAW; DENG, 2018). Num primeiro momento, fizemos o treinamento de cada uma dessas arquiteturas em todas as etapas do sistema ALPR. Em seguida, avaliamos o desempenho delas em cada uma das etapas de reconhecimento e selecionamos as que obtiveram a melhor relação entre tempo de processamento e acurácia para compor o sistema otimizado para dispositivos com recursos limitados.

5.2 Treinamento e avaliação dos modelos

Os resultados apresentados nesta seção cobrem um ciclo completo de experimentos com modelos treinados em computadores do Google Colaboratory², com processadores de alto desempenho, memória suficientemente grande e, especialmente, placas gráficas de propósito geral. O cálculo de métricas e teste dos modelos foram realizados em um computador com as seguintes especificações: Processador Core i5 10400, 16GB de RAM DDR4 2667 MHz, SSD 512GB.

A Seção 5.2.1 apresenta a preparação necessária para utilização da base UFS-ALPR no treinamento dos modelos. Cada uma das seções seguintes detalha o treinamento das arquiteturas e apresenta métricas de desempenho das mesmas. Como nosso sistema possui três etapas, cada uma delas é realizada por um modelo e foram experimentadas cinco arquiteturas, ao todo realizamos 15 treinamentos. Para cada uma das arquiteturas treinamos um modelo para detectar veículos, um modelo para placas e um modelo para detectar caracteres. Na Seção 5.2.5 os resultados de todos os treinamentos são agrupados e apresentados para comparação.

¹ Disponível em: https://github.com/tensorflow/models/tree/master/research/object_detection. Acesso em junho de 2022

² Disponível em: <https://colab.research.google.com/>. Acesso em junho de 2022

5.2.1 Preparação da base de dados

Todos os modelos foram treinados utilizando apenas imagens da base de dados construída por nós, a qual intitulamos UFS-ALPR. Ao todo, essa base possui 5029 imagens. Antes de iniciar o treinamento dos modelos, foi necessário fazer a separação da base de dados nos conjuntos de treinamento (70%) e teste (30%). A divisão foi feita de modo que ambos os conjuntos possuíssem uma proporção parecida de imagens de carros, motos, caminhões/ônibus, tipo de placa (antigo ou MERCOSUL) e posição do veículo na imagem (frontal ou traseira). Isso foi feito para reduzir o risco de treinar os modelos com a base desbalanceada, o que poderia prejudicar o desempenho. A Tabela 7 apresenta dados sobre as partições de treinamento e teste da base de dados UFS-ALPR:

Tabela 7 – Dados sobre as partições de treinamento e teste criadas para a base de dados UFS-ALPR

Conjunto	Número de imagens	Placas brasileiras	Placas MERCOSUL	Motos	Carros	Caminhões e ônibus
Treinamento	3499	2258	1241	265	2978	256
Teste	1530	1001	529	76	1308	146

Fonte: Elaborado pelo autor

As rotulações da base de dados retornadas pelo software de rotulação utilizado (DataTurks) são em formato json. Por conta disso, antes de utilizarmos a base para treinar os modelos, foi preciso fazer algumas conversões no formato das anotações. Um outro ponto importante é que mais de um *framework* foi utilizado para treinamento dos modelos, sendo que cada *framework* espera que a base esteja em um formato específico, o que gerou a necessidade de mais conversões no formato da base de dados. Além disso, os modelos de diferentes estágios usam diferentes imagens como entrada. Assim, foi preciso criar três bases de dados a partir da base original, uma para cada estágio da *pipeline* de reconhecimento. A Figura 45 apresenta exemplos de amostras das bases usadas para treinar os modelos de cada uma das etapas do sistema ALPR.

Para realizar o treinamento do modelo que detecta e reconhece caracteres, geramos a imagem negativa para todas as amostras. Assim, duplicamos a quantidade de dados disponíveis para treinamento desse modelo. A Figura 46 apresenta uma amostra original e a imagem negativa gerada para ela.

5.2.2 YOLOv4 Tiny

A YOLOv4 Tiny (BOCHKOVSKIY; WANG; LIAO, 2020) foi treinada utilizando o Darknet (REDMON, 2013–2016). Trata-se de um *framework* para redes neurais escrito em C desenvolvido por um dos criadores da YOLO. O Darknet requer que as rotulações estejam em formato de texto, devendo cada imagem ter um arquivo .txt correspondente. Cada linha desse arquivo é a *bounding box* de um objeto na imagem anotada e possui cinco valores separados por

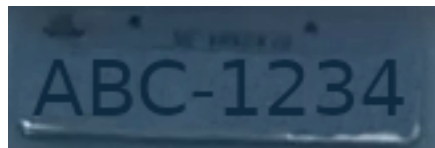
Figura 45 – Exemplos de amostras usadas para treinar os modelos de cada uma das etapas do sistema ALPR



(a) Exemplo de amostra utilizada para treinar o modelo que detecta veículos



(b) Exemplo de amostra utilizada para treinar o modelo que detecta placas



(c) Exemplo de amostra utilizada para treinar o modelo que detecta caracteres

Fonte: Elaborado pelo autor

Figura 46 – Exemplo do processamento utilizado para aumentar a partição de treinamento. Na esquerda, a imagem original e na direita a imagem negativa gerada



(a) Amostra original



(b) Imagem negativa gerada

Fonte: Elaborado pelo autor

espaços em branco: o primeiro valor é o índice da classe do objeto, os dois valores seguintes são as coordenadas do pixel central da *bounding box* relativos ao tamanho da imagem de entrada, por fim, os dois últimos valores são a largura e altura da *bounding box*, também relativos ao tamanho da imagem. Antes de iniciarmos o treinamento, fizemos a conversão da base de dados para esse formato.

O treinamento dos três modelos (veículos, placas e caracteres) foi feito utilizando os hiperparâmetros recomendados do Darknet. O tamanho de entrada foi 416×416, foi utilizado o otimizador SGD com *momentum* (0,9) *weight decay* (0,0005) e *learning rate* de 0,00261. O tamanho do *batch* foi 64, o número de *batches* (iterações de treinamento) utilizado foi 10000 para os modelos usados na detecção do veículo e da placa e 50000 para o modelo que faz detecção e reconhecimento dos caracteres. Em todos os treinamentos foram utilizadas técnicas de *data augmentation default* do Darknet. As transformações utilizadas adicionaram pequenas variações na exposição, saturação e matiz das imagens. O treinamento dos modelos utilizou *transfer learning*, isto é, os modelos foram treinados a partir dos pesos de uma YOLOv4 Tiny pré-treinada no *dataset* COCO (LIN et al., 2014). Os resultados obtidos pelos modelos em cada partição estão apresentados na Tabela 8.

Tabela 8 – Métricas dos modelos YOLOv4 Tiny treinados no *dataset* UFS-ALPR

Modelo	Conjunto	Recall	Precisão	Acurácia de classificação	IOU média	mAP @0.50	Frame rate (FPS)
Veículo	Teste	0,9875	0,9533	1,0	0,8404	0,9943	38,42
	Treinamento	0,9825	0,9933	1,0	0,8806	0,9900	37,85
Placa	Teste	0,9620	0,9658	0,9986	0,8049	0,9892	37,66
	Treinamento	0,9694	0,9985	1,0	0,8447	0,9972	38,90
Caractere	Teste	0,9724	0,9832	0,9397	0,7262	0,9816	35,82
	Treinamento	0,9761	0,9814	0,9669	0,7259	0,9958	37,17

Fonte: Elaborado pelo autor

A diferenciação entre moto e demais tipos de veículos é feita utilizando a proporção placa detectada. Com a YOLOv4 Tiny, em 99,38% das placas a diferenciação foi feita corretamente. A acurácia de classificação mostrada na Tabela 8 corresponde à porcentagem das detecções que foram corretamente classificadas. O modelo que detecta veículos classificou corretamente todos os veículos detectados, uma vez que esse modelo possui apenas uma classe. O modelo que detecta as placas conseguiu classificar corretamente o *layout*, entre padrão brasileiro ou MERCOSUL, de 99,86% das placas detectadas do conjunto de teste. Já o modelo que detecta caracteres, classificou corretamente 93,97% dos caracteres das placas detectadas do conjunto de teste. Para carregar e realizar a inferência com a YOLOv4 Tiny, utilizamos a biblioteca OpenCV, mais especificamente o módulo *dnn*, que dá suporte a operações envolvendo redes neurais.

5.2.3 YOLOv5 Nano e YOLOv5 Small

A implementação da YOLOv5 (JOCHER et al., 2022) e suas variações está disponível utilizando o *framework* PyTorch (PASZKE et al., 2019)¹. Para treinar os modelos, o *dataset* deve estar no mesmo formato requerido pelo Darknet, portanto, não foi preciso realizar nenhuma conversão da base de dados para treinar a YOLOv5 Nano e YOLOv5 Small.

O treinamento dos modelos foi feito utilizando os hiperparâmetros recomendados pelos mantenedores da YOLOv5. Foi utilizado o otimizador SGD com *momentum* de 0,937, *weight decay* de 0,0005 e *learning rate* de 0,01. O tamanho do *batch* utilizado foi 64 e os modelos foram treinados por 30 épocas. O tamanho de entrada dos modelos foi 640x640. Foram utilizadas algumas técnicas de *data augmentation* para aumentar a diversidade das imagens de treino. As transformações utilizadas foram flip horizontal, rotação, variação na saturação, matiz, *gamma* e mudança na escala das imagens (zoom). O treinamento dos modelos utilizou *transfer learning*, ou seja, os modelos foram treinados a partir dos pesos de uma YOLOv5 pré-treinada no *dataset* COCO (LIN et al., 2014). As Tabelas 9 e 10 apresentam as métricas da YOLOv5 Nano e YOLOv5 Small, respectivamente.

Tabela 9 – Métricas dos modelos YOLOv5 Nano treinados no *dataset* UFS-ALPR

Modelo	Conjunto	Recall	Precisão	Acurácia de classificação	IOU média	mAP @0.50	Frame rate (FPS)
Veículo	Teste	0,9941	0,9080	1,0	0,8395	0,9950	33,57
	Treinamento	1	0,9641	1,0	0,9126	0,9973	33,22
Placa	Teste	0,9921	0,9755	1,0	0,8536	0,9950	33,59
	Treinamento	0,9991	0,9980	1,0	0,8894	0,9964	33,56
Caractere	Teste	0,9980	0,9982	0,9385	0,8115	0,9790	30,39
	Treinamento	0,9990	0,9990	0,9673	0,8381	0,9812	30,94

Fonte: Elaborado pelo autor

Tabela 10 – Métricas dos modelos YOLOv5 Small treinados no *dataset* UFS-ALPR

Modelo	Conjunto	Recall	Precisão	Acurácia de classificação	IOU média	mAP @0.50	Frame rate (FPS)
Veículo	Teste	0,9993	0,9311	1,0	0,8673	0,9950	16,57
	Treinamento	1,0	0,9828	1,0	0,9266	0,9938	16,56
Placa	Teste	0,9967	0,9947	0,9980	0,9757	0,9950	16,42
	Treinamento	0,9994	0,9985	0,9945	0,8902	0,9949	15,59
Caractere	Teste	0,9971	0,9973	0,9339	0,8106	0,9790	15,71
	Treinamento	0,9981	0,9985	0,9673	0,8349	0,9708	15,80

Fonte: Elaborado pelo autor

¹ Disponível em: <<https://github.com/ultralytics/yolov5>>. Acesso em junho de 2022

Com a YOLOv5 Small e YOLOv5 Nano a diferenciação entre moto e demais tipos de veículos foi correta em 100% das placas detectadas. Para facilitar o carregamento e inferência dos modelos treinados, eles foram convertidos do PyTorch para o formato do TensorFlow.

5.2.4 SSD e CenterNet

Os treinamentos da SSD e da CenterNet foram feitos utilizando a API de detecção de objetos do TensorFlow, que disponibiliza modelos pré-treinados que podem ser ajustados para *datasets* customizados. Para realizar o treinamento, o *dataset* deve estar no formato TF records. Para tal, primeiramente foi preciso converter as anotações da base de dados para o formato Pascal VOC (EVERINGHAM et al., 2007), no qual cada imagem possui um xml correspondente com as anotações dos objetos. Em seguida, foi utilizado um *script* disponibilizado pela API de detecção de objetos do TensorFlow para gerar os arquivos no formato TF records a partir das anotações no formato Pascal VOC.

A SSD e a CenterNet usadas nos testes possuem a MobileNet V2 (SANDLER et al., 2018) como *backbone*. Em ambos os treinamentos foram utilizados os hiperparâmetros recomendados pela API de detecção de objetos do TensorFlow. A SSD possui tamanho de entrada 320x320 e foi treinada utilizando o otimizador SGD com *momentum* de 0,89 e *learning rate* de 0,079. Já a CenterNet foi treinada utilizando o otimizador Adam (KINGMA; BA, 2014) com *learning rate* de 0,005 e possui tamanho de entrada de 512x512. Ambos os modelos foram treinados por 25000 iterações a partir de pesos de modelos pré-treinados no *dataset* COCO (LIN et al., 2014). As Tabelas 11 e 12 apresentam as métricas da SSD e da CenterNet, respectivamente.

Utilizando a SSD e a CenterNet a diferenciação entre moto e demais tipos de veículos foi correta em 98,85% e 99,86% das placas detectadas no conjunto de teste, respectivamente.

Tabela 11 – Métricas dos modelos SSD treinados no *dataset* UFS-ALPR

Modelo	Conjunto	Recall	Precisão	Acurácia Classificação	IOU média	mAP @0.50	Frame rate (FPS)
Veículo	Teste	0,9542	0,9605	1,0	0,8471	0,9972	40,07
	Treinamento	1,0	0,9957	1,0	0,9373	0,9986	39,43
Placa	Teste	0,9738	0,9353	0,7261	0,7824	0,9856	52,01
	Treinamento	0,9942	0,9739	0,7990	0,8537	0,9894	53,98
Caractere	Teste	0,9354	0,9650	0,9141	0,6580	0,8939	45,43
	Treinamento	0,9675	0,9779	0,9646	0,7012	0,9134	46,91

Fonte: Elaborado pelo autor

Tabela 12 – Métricas dos modelos CenterNet treinados no *dataset* UFS-ALPR

Modelo	Conjunto	Recall	Precisão	Acurácia Classificação	IOU média	mAP @0.50	Frame rate (FPS)
Veículo	Teste	0,9379	0,9986	1	0,8680	0,9898	23,04
	Treinamento	1	0,9997	1	0,9440	0,9922	23,17
Placa	Teste	0,9986	0,9896	0,8481	0,9986	0,9999	24,79
	Treinamento	0,9997	0,9994	0,9422	0,8739	0,9926	25,05
Caractere	Teste	0,9436	0,9687	0,9312	0,6530	0,9520	23,25
	Treinamento	0,9540	0,9716	0,9611	0,6622	0,9642	23,67

Fonte: Elaborado pelo autor

5.2.5 Comparação dos modelos

Esta seção apresenta a comparação dos modelos treinados para detecção de veículos, placas e caracteres. Cada uma das tabelas a seguir agrupa as arquiteturas treinadas para cada tarefa, destacando para cada uma das métricas avaliadas o modelo que teve o melhor desempenho. Todos os dados são do conjunto de teste.

A Tabela 13 apresenta os resultados dos modelos que detectam veículos. Essa etapa é fundamental para a robustez do sistema ALPR, uma vez que um erro nela compromete todo o restante da *pipeline*. Por conta disso, é muito importante que o modelo possua um *recall* próximo de 100%. Dentre os modelos treinados, a YOLOv5 Small foi o que teve melhor desempenho. Em segundo lugar ficou a YOLOv5 Nano, com uma diferença de apenas 0,0052, contudo com o dobro do *frame rate* (33,57 contra 16,57 da YOLOv5 Small). A SSD foi a arquitetura com melhor *frame rate* (40,07).

Tabela 13 – Métricas dos modelos treinados para detecção de veículos

Modelo	Recall	Precisão	IOU média	mAP @0.50	Frame rate (FPS)
YOLOv4 Tiny	0,9875	0,9533	0,8404	0,9943	38,42
YOLOv5 Nano	0,9941	0,9080	0,8395	0,9950	33,57
YOLOv5 Small	0,9993	0,9311	0,8673	0,9950	16,57
SSD	0,9542	0,9605	0,8471	0,9972	40,07
CenterNet	0,9379	0,9986	0,8680	0,9898	23,04

Fonte: Elaborado pelo autor

As métricas dos modelos treinados para detecção de placas estão sumarizadas na tabela 14. Mais uma vez a SSD foi a arquitetura com melhor *frame rate* (52,0181), porém obteve o segundo menor *recall* e o pior desempenho de classificação. A YOLOv5 Nano e YOLOv5 Small obtiveram resultados parecidos para o *recall* e idênticos para acurácia de classificação e mAP.

Figura 47 – Saídas dos modelos que detectam veículos para uma amostra do *dataset* UFS-ALPR

Fonte: Elaborado pelo autor

Considerando esses resultados, a YOLOv5 Nano parece ser uma melhor opção, uma vez que possui o dobro do *frame rate* da YOLOv5 Small e uma precisão 4% superior, o que compensa a pequena diferença no *recall*. Como nosso sistema usa a classificação do tipo de placa para aplicar regras que melhoram o reconhecimento dos caracteres, é muito importante que a acurácia de classificação desses modelos seja alta. Nesse quesito, destacaram-se as variações da YOLOv5, que classificaram corretamente 100% das placas detectadas. Enquanto isso, a SSD e a CenterNet foram as que pior classificaram a placa entre padrão brasileiro e MERCOSUL, com acurácia de 0,7261 e 0,8481, respectivamente.

Figura 48 – Saídas dos modelos que detectam placas para uma amostra do *dataset* UFS-ALPR

Fonte: Elaborado pelo autor

Tabela 14 – Métricas dos modelos treinados para detecção de placas

Modelo	Recall	Precisão	Acurácia de classificação	IOU média	mAP @0.50	Frame rate (FPS)
YOLOv4 Tiny	0,9620	0,9658	0,9986	0,8049	0,9892	37,66
YOLOv5 Nano	0,9921	0,9755	1,0	0,8536	0,9950	33,59
YOLOv5 Small	0,9993	0,9311	1,0	0,8673	0,9950	16,57
SSD	0,9738	0,9353	0,7261	0,7824	0,9856	52,01
CenterNet	0,9986	0,9896	0,8481	0,9986	0,9999	24,79

Fonte: Elaborado pelo autor

A Tabela 15 apresenta as métricas dos modelos treinados para detectar caracteres. A YOLOv5 Nano se destacou positivamente, obtendo o melhor *recall*, precisão, IOU média e sendo a segunda melhor arquitetura na classificação das detecções. A SSD foi mais uma vez a arquitetura mais veloz, atingindo um *frame rate* de 45,43.

Figura 49 – Saídas dos modelos que detectam caracteres para uma amostra do *dataset* UFS-ALPR

Fonte: Elaborado pelo autor

Tabela 15 – Métricas dos modelos treinados para detecção e reconhecimento de caracteres

Modelo	Recall	Precisão	Acurácia de classificação	IOU média	mAP @0.50	Frame rate (FPS)
YOLOv4 Tiny	0,9724	0,9832	0,9397	0,7262	0,9816	35,82
YOLOv5 Nano	0,9980	0,9982	0,9385	0,8115	0,9790	30,39
YOLOv5 Small	0,9971	0,9973	0,9339	0,8106	0,9790	15,71
SSD	0,9354	0,9650	0,9141	0,6580	0,8939	45,43
CenterNet	0,9436	0,9687	0,9312	0,6530	0,9520	23,25

Fonte: Elaborado pelo autor

5.3 Sistema ALPR

O sistema ALPR proposto foi implementado utilizando a linguagem Python. Todos os modelos foram carregados utilizando o TensorFlow, exceto a YOLOv4 Tiny, que utilizou a biblioteca OpenCV para ser carregada. A avaliação do sistema ALPR implementado se deu em duas fases: primeiramente, cada etapa foi implementada e avaliada de maneira isolada, em seguida, elas foram combinadas e o desempenho geral do sistema avaliado.

É importante ressaltar que as métricas calculadas nessa seção levam em consideração regras específicas de cada etapa do sistema proposto. Por exemplo, as etapas de detecção do veículo e da placa consideram apenas a detecção com maior *score* de confiança. Além disso, a métrica da etapa de reconhecimento dos caracteres já considera a aplicação das regras de substituição baseadas no *layout* da placa. As Tabelas 16, 17 e 18 apresentam as métricas de cada

etapa avaliada de maneira isolada, isto é, cada estágio recebeu todas imagens, sem considerar os erros do estágio anterior.

A Tabela 16 mostra o desempenho das arquiteturas testadas no estágio de detecção do veículo. Nessa etapa, a SSD foi o modelo com o melhor resultado, detectando corretamente 94,70% dos veículos do conjunto de teste e atingindo 39,91 FPS.

Tabela 16 – Métricas do estágio de detecção do veículo para cada uma das arquiteturas experimentadas no conjunto de teste

Modelo	Sucesso na detecção (%)	Frame rate (FPS)
YOLOv4 Tiny	93,92	37,73
YOLOv5 Nano	89,80	32,92
YOLOv5 Small	93,00	16,39
SSD	94,70	39,91
CenterNet	93,72	23,38

Fonte: Elaborado pelo autor

No estágio de detecção da placa, vide Tabela 17, a YOLOv5 Small foi o modelo com os melhores resultados de detecção e classificação: 99,28% e 100%, respectivamente. Porém, obteve o pior *frame rate* (16,37). O segundo melhor modelo na detecção foi a CenterNet (98,95%), contudo, obteve apenas 84,67% de sucesso na classificação e o segundo pior *frame rate* (25,11). A YOLOv5 Nano detectou corretamente 96,99% das placas e classificou corretamente o *layout* de 100% delas, atingindo ainda 33,56 FPS.

Tabela 17 – Métricas do estágio de detecção da placa para cada uma das arquiteturas experimentadas no conjunto de teste

Modelo	Sucesso na detecção (%)	Sucesso na classificação (%)	Frame rate (FPS)
YOLOv4 Tiny	94,83	99,86	37,30
YOLOv5 Nano	96,99	100,00	33,56
YOLOv5 Small	99,28	100,00	16,37
SSD	92,74	72,72	53,92
CenterNet	98,95	84,67	25,11

Fonte: Elaborado pelo autor

A Tabela 18 apresenta as métricas do estágio de detecção dos caracteres. A YOLOv5 Small foi o modelo que reconheceu corretamente o maior número de placas, acertando os sete caracteres de 89,41% das imagens. Logo atrás vem a YOLOv5 Nano, que reconheceu corretamente 88,88% das placas. Além disso, foi o modelo que detectou e classificou corretamente o maior número de caracteres, atingindo 99,80% e 93,66%, respectivamente.

Tabela 18 – Métricas do estágio de detecção dos caracteres para cada uma das arquiteturas experimentadas no conjunto de teste

Modelo	Sucesso na detecção (%)	Sucesso na classificação (%)	Placas corretamente reconhecidas (%)	Frame rate (FPS)
YOLOv4 Tiny	96,56	90,73	76,33	36,12
YOLOv5 Nano	99,80	93,66	88,88	29,99
YOLOv5 Small	99,67	93,09	89,41	15,54
SSD	93,19	85,17	64,44	45,45
CenterNet	94,18	87,71	69,73	23,35

Fonte: Elaborado pelo autor

Como nosso sistema possui três etapas e para cada etapa foram realizados testes com cinco diferentes arquiteturas, ao todo existem $5^3 = 125$ possíveis combinações de modelos. Após avaliarmos cada etapa isoladamente, unimos as etapas e avaliamos o desempenho geral do sistema ALPR para cada umas das 125 combinações. Isso foi feito para auxiliar na tomada de decisão de quais arquiteturas iriam compor o sistema ALPR que seria otimizado para dispositivos com recursos limitados. A Tabela 19 apresenta as métricas das 10 melhores combinações de modelos, as métricas de todas as 125 combinações estão apresentadas no Apêndice A.

Tabela 19 – Métricas das dez melhores combinações de modelos, os dados são do conjunto de teste

Modelo veículos	Modelo placas	Modelo caracteres	Placas corretamente reconhecidas (%)	Frame rate (FPS)
SSD	YOLOv5 Small	YOLOv5 Small	87,58	6,71
SSD	YOLOv5 Nano	YOLOv5 Small	86,40	8,47
SSD	YOLOv5 Small	YOLOv5 Nano	85,81	8,42
SSD	YOLOv5 Nano	YOLOv5 Nano	84,90	11,57
SSD	YOLOv4 Tiny	YOLOv5 Small	84,31	8,94
CenterNet	YOLOv5 Small	YOLOv5 Small	84,18	6,22
YOLOv4 Tiny	YOLOv5 Small	YOLOv5 Small	84,05	6,75
YOLOv5 Small	YOLOv5 Small	YOLOv5 Small	83,52	5,56
YOLOv4 Tiny	YOLOv5 Small	YOLOv5 Small	83,20	8,51
CenterNet	YOLOv5 Nano	YOLOv5 Small	82,81	7,56

Fonte: Elaborado pelo autor

A combinação que obteve o melhor resultado foi a SSD para detecção de veículos e YOLOv5 Small para detecção das placas e caracteres. Esse conjunto de modelos foi capaz de reconhecer corretamente 87,58% das placas e executar a 6,71 FPS.

5.3.1 Definição dos modelos para compor o sistema ALPR embarcado

A seleção dos modelos para compor o sistema que seria embarcado no Raspberry Pi 4 e no Jetson Nano se deu em duas etapas:

1. Avaliação das métricas calculadas: Primeiramente, avaliamos as métricas dos modelos em cada um dos estágios, bem como o resultado das combinações. Como nosso objetivo era criar um sistema ALPR leve, capaz de ser implantado em dispositivos com recursos limitados, decidimos focar na utilização das combinações de modelos que atingiram pelo menos um *frame rate* de 10 FPS nos testes realizados. Vale frisar que nos testes utilizamos um computador com processador Core i5 10400, 16GB de RAM DDR4 2667 MHz, 512GB SSD e sem placa gráfica. A Tabela 20 apresenta as 10 melhores combinações que atingiram mais de 10 FPS:

Tabela 20 – Métricas das dez melhores combinações de modelos que atingiram mais de 10 FPS, os dados são do conjunto de teste

Modelo veículos	Modelo placas	Modelo caracteres	Placas corretamente reconhecidas (%)	Frame rate (FPS)
SSD	YOLOv5 Nano	YOLOv5 Nano	84,90	11,57
SSD	YOLOv4 Tiny	YOLOv5 Nano	82,28	12,16
YOLOv4 Tiny	YOLOv5 Nano	YOLOv5 Nano	80,65	11,60
YOLOv4 Tiny	YOLOv4 Tiny	YOLOv5 Nano	79,93	12,15
CenterNet	YOLOv4 Tiny	YOLOv5 Nano	79,21	10,30
SSD	CenterNet	YOLOv5 Nano	78,16	10,51
SSD	YOLOv5 Nano	YOLOv4 Tiny	77,58	12,36
YOLOv5 Nano	YOLOv5 Nano	YOLOv5 Nano	77,12	11,11
YOLOv5 Nano	YOLOv4 Tiny	YOLOv5 Nano	75,88	11,94
YOLOv4 Tiny	CenterNet	YOLOv5 Nano	74,96	10,31

Fonte: Elaborado pelo autor

2. Avaliação das combinações de modelos com imagens de fora da base de dados: Entre os dias 03/05/2022 e 11/05/2022 foram capturadas imagens na entrada de veículos da UFS. Durante as filmagens, o objetivo foi obter imagens que se aproximassem ao máximo do cenário de uso imaginado para o sistema: controle de acesso em uma portaria. A Figura 50 apresenta algumas amostras das imagens capturadas. Posteriormente, essas imagens foram processadas utilizando o sistema ALPR com diferentes combinações de modelos e analisadas manualmente.

Após a avaliação das métricas e testes utilizando imagens de fora da base de dados, decidimos utilizar a combinação de modelos que possui a YOLOv5 Nano em todas as etapas do sistema ALPR que seria embarcado. Apesar de não ter obtido os melhores resultados, essa combinação reconheceu corretamente 77,12% das placas do conjunto de teste e atingiu *frame*

Figura 50 – Amostras de imagens capturadas na entrada de veículos da UFS



Fonte: Elaborado pelo autor

rate de 11,11 FPS, vide Tabela 20. Um ponto positivo dela é que ela possui apenas uma arquitetura, assim, o processo de implantação e otimização para o Raspberry Pi 4 e Jetson Nano foi mais simples. Os seguintes pontos foram considerados para a seleção dos modelos que seriam embarcados:

- A YOLOv5 Small, apesar de ter obtido ótimos resultados de detecção e classificação, foi o modelo com o pior *frame rate* em todas as etapas. Além disso, sua versão reduzida, a YOLOv5 Nano, obteve resultados de classificação e detecção muito próximos ou até mesmo melhores, com o dobro do *frame rate*. Por conta disso, consideramos que a YOLOv5 Nano teve um custo benefício melhor e decidimos não utilizar a YOLOv5 Small;
- A SSD e a CenterNet não obtiveram bons resultados nas etapas de classificação, tanto do *layout* da placa quanto dos caracteres. Devido a isso, decidimos não utilizá-las nesses estágios;
- Apesar dos bons resultados da SSD na etapa de detecção de veículos, nos testes com imagens de fora da base de dados percebemos que a SSD deixava de detectar muitos veículos. O que faz sentido, uma vez que desconsiderando a regra de selecionar apenas a detecção com o maior *score* de confiança, a SSD obteve o segundo pior *recall* dentre os modelos testados (vide Tabela 13) Por conta disso, decidimos não utilizá-la para esse estágio;
- No estágio de detecção de veículos, decidimos utilizar o modelo com o melhor *recall*, uma vez que um erro nessa etapa compromete toda a *pipeline* de reconhecimento. Assim,

decidimos utilizar a YOLOv5 Nano para essa etapa, que foi o modelo com melhor *recall* depois da YOLOv5 Small (vide Tabela 13);

- Na etapa de detecção e classificação do *layout* das placas, a YOLOv5 Nano obteve os melhores resultados (depois da YOLOv5 Small). Por isso, decidimos utilizá-la nesse estágio;
- Na etapa de reconhecimento dos caracteres a YOLOv5 Nano foi a melhor tanto na detecção quanto na classificação, por isso decidimos utilizá-la.

5.4 Implantação do sistema ALPR em dispositivos com recursos limitados

A implantação, ou *deploy*, de modelos de aprendizado em dispositivos como o Raspberry Pi 4 e o Jetson Nano não é uma tarefa simples. Extrair o melhor desse tipo de equipamento requer, na maioria das vezes, muito trabalho de configuração e otimização. As arquiteturas não convencionais possuem algumas particularidades, o que dificulta a preparação do ambiente e execução de testes. Por exemplo, muitas vezes é preciso compilar as bibliotecas especificamente para essas arquiteturas, o que costuma levar bastante tempo.

Essa seção descreve o processo de implantação do sistema ALPR em dois dispositivos com recursos limitados: Raspberry Pi 4 e NVIDIA Jetson Nano. As subseções a seguir detalham as especificações dos dispositivos, o processo de configuração do ambiente, otimização dos modelos e, por fim, as métricas de execução do sistema nesses dispositivos.

5.4.1 Raspberry Pi 4

5.4.1.1 Especificação do hardware

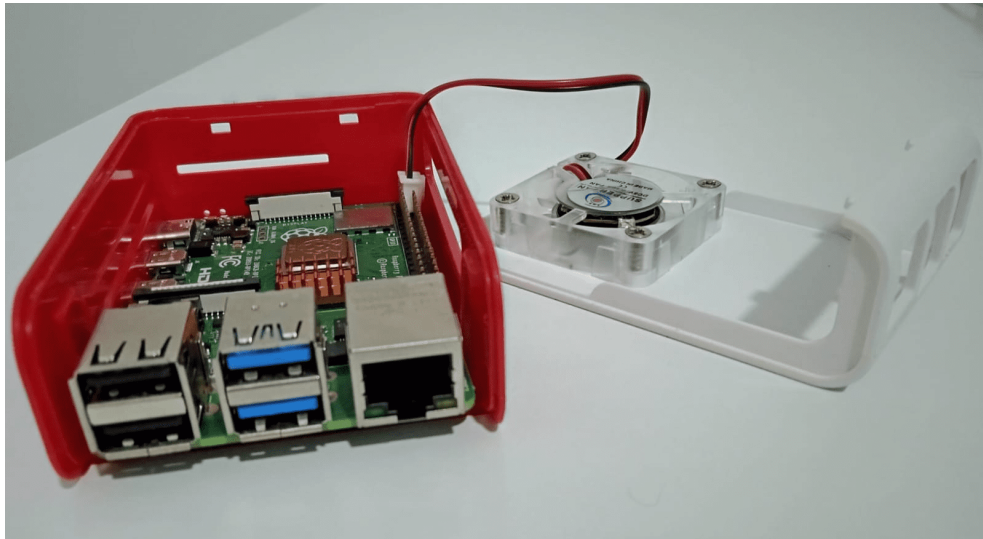
A versão do Raspberry utilizado nos experimentos foi o Raspberry Pi 4 modelo B, com as seguintes especificações:

- 4GB de memória RAM;
- Processador quad core Cortex-A72 (ARM v8 64-bit 1.5GHz);
- GPU Broadcom VideoCore VI (500 MHz).

5.4.1.2 Configuração do ambiente e otimização dos modelos

O sistema operacional utilizado durante os experimentos foi o Raspberry Pi OS. Trata-se de uma distribuição de Linux baseada no Debian, leve e mantida pela Fundação Raspberry Pi¹.

Figura 51 – Raspberry Pi usado nos experimentos



Fonte: Elaborado pelo autor

Essa distribuição era anteriormente chamada de Raspbian. Seguimos as instruções oficiais de instalação e utilizamos a ferramenta Raspberry Pi Imager para gravar o sistema operacional no cartão SD.

Após a configuração inicial do sistema operacional, fizemos a instalação do Python e dos pacotes necessários para execução do sistema ALPR. Instalamos as bibliotecas compiladas especificamente para a arquitetura ARM de 64 bits. Para a execução dos modelos, optamos por utilizar o TensorFlow Lite: a biblioteca oficial do TensorFlow para implantar modelos de aprendizado em ambientes mobile, microcontroladores e dispositivos de *edge computing*.

Para converter a YOLOv5 Nano para o formato do TensorFlow Lite utilizamos os *scripts* disponibilizados no repositório da YOLOv5 Nano (JOCHER et al., 2022).

5.4.1.3 Métricas do sistema

A seguir são apresentadas as métricas do sistema ALPR executado no Raspberry Pi 4, as métricas são da partição de teste da base UFS-ALPR:

¹ Disponível em: <<https://www.raspberrypi.com/software/>>. Acesso em junho de 2022

Tabela 21 – Comparação das métricas do sistema ALPR em um computador convencional e no Raspberry Pi 4

Ambiente	Placas corretamente reconhecidas (%)	Frame rate (FPS)
Computador convencional	77,12	11,11
Raspberry Pi 4	77,05	0,5328

Fonte: Elaborado pelo autor

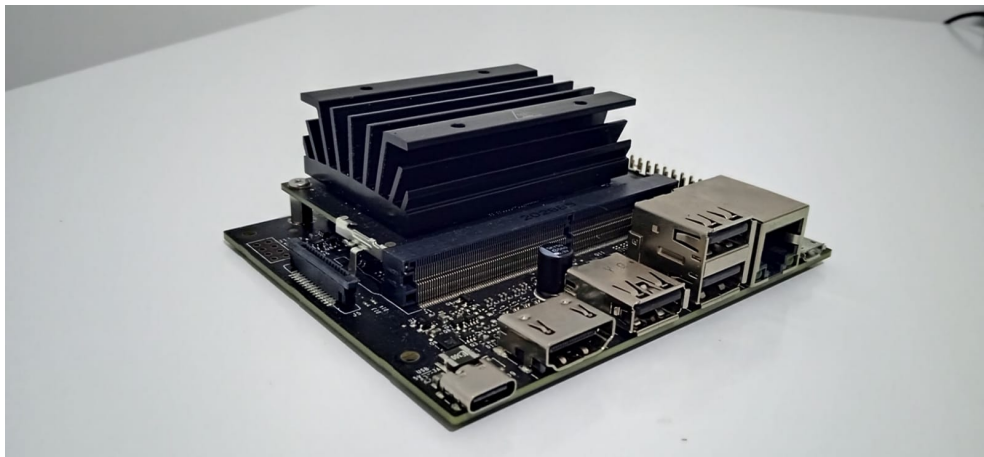
5.4.2 NVIDIA Jetson Nano

5.4.2.1 Especificação do hardware

O Jetson Nano utilizado nos experimentos possui as seguintes especificações:

- Unidade de processamento gráfico (128 CUDA *cores* NVIDIA Maxwell 921 MHz);
- 2GB de memória RAM;
- Processador quad core ARM A57 (1.43 GHz).

Figura 52 – Jetson Nano usado nos experimentos



Fonte: Elaborado pelo autor

5.4.2.2 Configuração do ambiente e otimização dos modelos

Para realizar os testes do sistema ALPR no Jetson Nano, instalamos uma imagem de sistema operacional disponibilizada no site da NVIDIA¹. Essa imagem se baseia no Ubuntu e já vem com o Jetpack² instalado, que é o SDK da NVIDIA para dispositivos de borda, ele contém

o CUDA, cuDNN, TensorRT e diversos outros softwares. Visando garantir que a GPU fosse utilizada, optamos por fazer o carregamento e inferência dos modelos utilizando o TensorRT: *framework* da NVIDIA específico para execução de redes neurais em dispositivos de borda.

A conversão dos modelos para o TensorRT foi realizada em duas etapas: primeiramente, utilizamos um *script* do repositório da YOLOv5 para converter a YOLOv5 Nano para o formato ONNX - *Open Neural Network Exchange*³, que é um formato aberto para representação de redes neurais. Em seguida, utilizamos como base a biblioteca *cvu (computer vision utils)*¹ visando implementar rotinas para converter o modelo do formato ONNX para TensorRT e para carregar e realizar inferência com o modelo convertido.

Após a conversão dos modelos, realizamos testes para verificar se a GPU do Jetson Nano estava sendo utilizada. Em seguida, calculamos as métricas de execução do sistema.

5.4.2.3 Métricas do sistema

A seguir são apresentadas as métricas do sistema ALPR executado no Jetson Nano, as métricas são da partição de teste da base UFS-ALPR:

Tabela 22 – Comparação das métricas do sistema ALPR em um computador convencional e no Jetson Nano

Ambiente	Placas corretamente reconhecidas (%)	Frame rate (FPS)
Computador convencional	77,12	11,11
Jetson Nano	77,51	5,66

Fonte: Elaborado pelo autor

¹ Disponível em: <<https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#write>>. Acesso em junho de 2022

² Disponível em: <<https://developer.nvidia.com/embedded/jetpack>>. Acesso em junho de 2022

³ Disponível em: <<https://onnx.ai/>>. Acesso em junho de 2022

¹ Disponível em: <<https://github.com/BlueMirrors/cvu>>. Acesso em junho de 2022

6

Considerações Finais

O presente trabalho abrangeu todo o ciclo de desenvolvimento de uma aplicação baseada em modelos de *deep learning*. Isso envolveu desde a coleta e rotulação dos dados, treinamento e avaliação de modelos, implementação de uma aplicação que usa as predições dos modelos e até mesmo o *deploy* da aplicação em plataformas não convencionais com recursos limitados. A execução desse trabalho permitiu vivenciar na prática as dificuldades de cada uma dessas etapas.

Uma contribuição relevante do presente trabalho foi a construção de uma base de dados original para ALPR. A base de dados construída, a qual intitulamos UFS-ALPR, conta com 5029 imagens de 326 veículos distintos. Ela possui rótulos que descrevem o tipo do veículo (carro, moto, ônibus, caminhão), marca, modelo, padrão de placa (antigo ou MERCOSUL), posição do veículo (frontal ou traseira) e as *bounding boxes* do veículo, da placa e de cada caractere dela. No momento da escrita deste trabalho, a base ainda não foi disponibilizada para comunidade acadêmica. Porém, sabemos da importância dos dados e do esforço necessário para construção de uma base de dados, portanto, pretendemos publicá-la em breve e estamos tratando com o CINTTEC/UFS dos aspectos legais relacionados com a divulgação e hospedagem no domínio da UFS.

Durante a construção do sistema ALPR proposto, realizamos o treinamento de diversos modelos. Ao todo, foram experimentadas cinco diferentes arquiteturas, cada uma treinada em três tarefas de detecção distintas. O treinamento de modelos profundos é um processo demorado, com particularidades e desafios, apesar disso, os modelos treinados obtiveram resultados bastante promissores. Além disso, a exploração de diferentes arquiteturas permitiu testar combinações e avaliar a relação de custo benefício do sistema em cada uma delas.

Os testes utilizando o Raspberry Pi 4 e o Jetson Nano evidenciaram que muito ainda precisa ser feito para que uma aplicação que precisa executar três modelos profundos por *frame* possa ser usada em um cenário real com um tempo de resposta aceitável nesse tipo de dispositivo. Trabalhos futuros podem explorar o uso de técnicas de compressão de modelos, como poda

(HAN; MAO; DALLY, 2015) e quantização (JACOB et al., 2018), para tentar melhorar a latência/*frame rate* do sistema. Acreditamos que para melhorar o desempenho computacional, o uso de algoritmos de *tracking* pode ser uma alternativa para reduzir o processamento das etapas de detecção do veículo e da placa. Ao usar *tracking* passa-se a fazer detecções espaçadas e não em todos os *frames*. Nos *frames* entre as detecções seriam utilizados algoritmos de *tracking*, que, em geral, são mais leves que modelos profundos. Essa abordagem é chamada de *tracking-by-detection* (HOU; WANG; CHAU, 2019).

Trabalhos futuros podem também explorar mais a base UFS-ALPR, por exemplo a informação do modelo, marca e posição do veículo. Muitos sistemas ALPR comerciais, além do reconhecimento da placa, também disponibilizam esse tipo de informação.

Referências

- ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <<https://www.tensorflow.org/>>. Citado na página 69.
- AGUIAR, A. S. P. de et al. Vineyard trunk detection using deep learning—an experimental device benchmark. *Computers and Electronics in Agriculture*, Elsevier, v. 175, p. 105535, 2020. Citado na página 41.
- BIEDER, F.; SANDKÜHLER, R.; CATTIN, P. C. *Comparison of Methods Generalizing Max- and Average-Pooling*. 2021. Citado na página 22.
- BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. Citado 9 vezes nas páginas 17, 22, 26, 27, 34, 65, 68, 69 e 70.
- CAVALCANTE, M. C. Segmentação de imagens de lesões de pele usando a rede neural convolucional u-net. *Trabalho de conclusão de curso. Universidade Federal do Espírito Santo.*, 2019. Citado na página 25.
- CHOUHAN, V. et al. A novel transfer learning based approach for pneumonia detection in chest x-ray images. *Applied Sciences*, Multidisciplinary Digital Publishing Institute, v. 10, n. 2, p. 559, 2020. Citado na página 37.
- COGNILYTICA. *Data Engineering, Preparation, and Labeling for AI*. 2019. Disponível em: <<https://www.cognilytica.com/2019/03/06/report-data-engineering-preparation-and-labeling-for-ai-2019/>>. Citado 2 vezes nas páginas 17 e 56.
- DU, S. et al. Automatic license plate recognition (alpr): A state-of-the-art review. *IEEE Transactions on circuits and systems for video technology*, IEEE, v. 23, n. 2, p. 311–325, 2012. Citado 2 vezes nas páginas 16 e 38.
- DUAN, K. et al. Centernet: Keypoint triplets for object detection. *CoRR*, abs/1904.08189, 2019. Disponível em: <<http://arxiv.org/abs/1904.08189>>. Citado 4 vezes nas páginas 27, 35, 36 e 37.
- EVERINGHAM, M. et al. *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results*. 2007. [Http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html](http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html). Citado 4 vezes nas páginas 27, 32, 35 e 74.
- FELZENSZWALB, P. F. et al. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 32, n. 9, p. 1627–1645, 2009. Citado na página 32.
- FUCHS, F. et al. Super-human performance in gran turismo sport using deep reinforcement learning. *IEEE Robotics Autom. Lett.*, v. 6, n. 3, p. 4257–4264, 2021. Disponível em: <<https://doi.org/10.1109/LRA.2021.3064284>>. Citado na página 16.
- GIRSHICK, R. Fast r-cnn. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2015. p. 1440–1448. Citado 4 vezes nas páginas 26, 30, 31 e 32.

- GIRSHICK, R. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2014. p. 580–587. Citado 3 vezes nas páginas 26, 29 e 30.
- GONÇALVES, G. R. et al. Real-time automatic license plate recognition through deep multi-task networks. In: IEEE. *2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. [S.l.], 2018. p. 110–117. Citado 6 vezes nas páginas 7, 9, 49, 50, 51 e 52.
- GONÇALVES, G. R. et al. Real-time automatic license plate recognition through deep multi-task networks. In: *Conference on Graphic, Patterns and Images (SIBGRAPI)*. [s.n.], 2018. p. 1–8. Disponível em: <<http://smartsenselab.dcc.ufmg.br/wp-content/uploads/2019/02/paper.pdf>>. Citado 3 vezes nas páginas 16, 57 e 58.
- GONÇALVES, G. R. et al. Multi-task learning for low-resolution license plate recognition. In: SPRINGER. *Iberoamerican Congress on Pattern Recognition*. [S.l.], 2019. p. 251–261. Citado na página 53.
- GONÇALVES, G. R. et al. Benchmark for license plate character segmentation. *Journal of Electronic Imaging*, International Society for Optics and Photonics, v. 25, n. 5, p. 053034, 2016. Citado 3 vezes nas páginas 48, 51 e 55.
- GONG, W. et al. A novel deep learning method for intelligent fault diagnosis of rotating machinery based on improved cnn-svm and multichannel data fusion. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 19, n. 7, p. 1693, 2019. Citado na página 21.
- GONZALEZ-HUITRON, V. et al. Disease detection in tomato leaves via cnn with lightweight architectures implemented in raspberry pi 4. *Computers and Electronics in Agriculture*, Elsevier, v. 181, p. 105951, 2021. Citado na página 40.
- GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. [S.l.]: MIT press, 2016. Citado na página 20.
- GRAVES, A.; MOHAMED, A.-r.; HINTON, G. Speech recognition with deep recurrent neural networks. In: IEEE. *2013 IEEE international conference on acoustics, speech and signal processing*. [S.l.], 2013. p. 6645–6649. Citado na página 17.
- HAN, S.; MAO, H.; DALLY, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. Citado na página 88.
- HE, K. et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2015. p. 1026–1034. Citado na página 16.
- HE, K. et al. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 770–778. Citado 2 vezes nas páginas 21 e 22.
- HERMANN, K. M. et al. Teaching machines to read and comprehend. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2015. p. 1693–1701. Citado na página 17.
- HOU, X.; WANG, Y.; CHAU, L.-P. Vehicle tracking using deep sort with low confidence track filtering. In: IEEE. *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. [S.l.], 2019. p. 1–6. Citado na página 88.

- HUANG, G. et al. Densely connected convolutional networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 4700–4708. Citado na página 22.
- JACOB, B. et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2018. p. 2704–2713. Citado na página 88.
- JAMTSO, Y.; RIYAMONGKOL, P.; WARANUSAST, R. Real-time bhutanese license plate localization using yolo. *ICT Express*, v. 6, n. 2, p. 121–124, 2020. ISSN 2405-9595. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S2405959519303170>>. Citado na página 68.
- JEONG, J.; YOON, T. S.; PARK, J. B. Towards a meaningful 3d map using a 3d lidar and a camera. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 18, n. 8, p. 2571, 2018. Citado na página 24.
- JIN, L. et al. License plate recognition algorithm for passenger cars in chinese residential areas. *Sensors*, Molecular Diversity Preservation International, v. 12, n. 6, p. 8355–8370, 2012. Citado na página 16.
- JOCHER, G. et al. *ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference*. Zenodo, 2022. Disponível em: <<https://doi.org/10.5281/zenodo.6222936>>. Citado 6 vezes nas páginas 34, 65, 68, 69, 73 e 84.
- JØRGENSEN, H. *Automatic license plate recognition using deep learning techniques*. Dissertação (Mestrado) — NTNU, 2017. Citado 2 vezes nas páginas 22 e 38.
- KESSENTINI, Y. et al. A two-stage deep neural network for multi-norm license plate detection and recognition. *Expert systems with applications*, Elsevier, v. 136, p. 159–170, 2019. Citado na página 39.
- KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. Citado na página 74.
- KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2012. p. 1097–1105. Citado 6 vezes nas páginas 17, 21, 22, 23, 24 e 30.
- KURPIEL, F. D.; MINETTO, R.; NASSU, B. T. Convolutional neural networks for license plate detection in images. In: IEEE. *2017 IEEE International Conference on Image Processing (ICIP)*. [S.l.], 2017. p. 3395–3399. Citado na página 55.
- LAROCA, R. et al. A robust real-time automatic license plate recognition based on the yolo detector. In: IEEE. *2018 international joint conference on neural networks (ijcnn)*. [S.l.], 2018. p. 1–10. Citado 12 vezes nas páginas 7, 9, 39, 46, 47, 48, 50, 51, 52, 56, 57 e 68.
- LAROCA, R. et al. An efficient and layout-independent automatic license plate recognition system based on the yolo detector. *arXiv preprint arXiv:1909.01754*, 2019. Citado 12 vezes nas páginas 8, 9, 16, 17, 22, 46, 52, 53, 54, 66, 67 e 68.
- LAW, H.; DENG, J. Cornernet: Detecting objects as paired keypoints. In: *Proceedings of the European conference on computer vision (ECCV)*. [S.l.: s.n.], 2018. p. 734–750. Citado 4 vezes nas páginas 35, 36, 65 e 69.

- LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Ieee, v. 86, n. 11, p. 2278–2324, 1998. Citado na página 20.
- LI, H. et al. Reading car license plates using deep neural networks. *Image and Vision Computing*, Elsevier, v. 72, p. 14–23, 2018. Citado na página 55.
- LIN, T.-Y. et al. Focal loss for dense object detection. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2017. p. 2980–2988. Citado na página 27.
- LIN, T.-Y. et al. *Microsoft COCO: Common Objects in Context*. arXiv, 2014. Disponível em: <<https://arxiv.org/abs/1405.0312>>. Citado 4 vezes nas páginas 69, 72, 73 e 74.
- LIN, T.-Y. et al. *Microsoft COCO: Common Objects in Context*. 2015. Citado na página 34.
- LIN, Y. et al. *NEC-UIUC*. 2010. Disponível em: <https://image-net.org/static_files/files/ILSVRC2010_NEC-UIUC.pdf>. Citado na página 21.
- LIU, W. et al. Ssd: Single shot multibox detector. In: SPRINGER. *European conference on computer vision*. [S.l.], 2016. p. 21–37. Citado 6 vezes nas páginas 27, 34, 35, 36, 65 e 69.
- LONG, J.; SHELHAMER, E.; DARRELL, T. Fully convolutional networks for semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2015. p. 3431–3440. Citado na página 24.
- MANINIS, K. et al. *Deep retinal image understanding International conference on medical image computing and computer-assisted intervention*. [S.l.]: Springer, 2016. Citado na página 16.
- MANJARI, K.; VERMA, M.; SINGAL, G. Creation: Computational constrained travel aid for object detection in outdoor environment. In: IEEE. *2019 15th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*. [S.l.], 2019. p. 247–254. Citado na página 40.
- MONTAZZOLLI, S.; JUNG, C. Real-time brazilian license plate detection and recognition using deep convolutional neural networks. In: IEEE. *2017 30th SIBGRAPI conference on graphics, patterns and images (SIBGRAPI)*. [S.l.], 2017. p. 55–62. Citado 4 vezes nas páginas 47, 51, 54 e 55.
- OPENALPR. *OpenALPR*. 2021. Disponível em: <<https://br.openalpr.com/>>. Citado na página 45.
- PADILLA, R.; NETTO, S. L.; SILVA, E. A. da. A survey on performance metrics for object-detection algorithms. In: IEEE. *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. [S.l.], 2020. p. 237–242. Citado 3 vezes nas páginas 27, 28 e 29.
- PASZKE, A. et al. Pytorch: An imperative style, high-performance deep learning library. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019. p. 8024–8035. Disponível em: <<http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>>. Citado 2 vezes nas páginas 34 e 73.
- PENA, D. et al. Benchmarking of cnns for low-cost, low-power robotics applications. In: *RSS 2017 Workshop: New Frontier for Deep Learning in Robotics*. [S.l.: s.n.], 2017. p. 1–5. Citado na página 40.

- PERRONNIN, F.; SANCHEZ, J. *XRCE*. 2011. Disponível em: <<https://image-net.org/challenges/LSVRC/2011/>>. Citado na página 21.
- POPAT, P.; SHETH, P.; JAIN, S. Animal/object identification using deep learning on raspberry pi. In: *Information and Communication Technology for Intelligent Systems*. [S.l.]: Springer, 2019. p. 319–327. Citado na página 40.
- POTDAR, K.; PAI, C. D.; AKOLKAR, S. A convolutional neural network based live object recognition system as blind aid. *arXiv preprint arXiv:1811.10399*, 2018. Citado na página 26.
- REDMON, J. *Darknet: Open Source Neural Networks in C*. 2013–2016. <<http://pjreddie.com/darknet/>>. Citado na página 70.
- REDMON, J. et al. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 779–788. Citado 9 vezes nas páginas 17, 26, 27, 32, 33, 52, 53, 54 e 55.
- REDMON, J.; FARHADI, A. Yolo9000: better, faster, stronger. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 7263–7271. Citado 6 vezes nas páginas 17, 26, 27, 33, 47 e 53.
- REDMON, J.; FARHADI, A. Yolo3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. Citado 5 vezes nas páginas 17, 26, 27, 34 e 35.
- REN, S. et al. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, v. 28, p. 91–99, 2015. Citado 4 vezes nas páginas 26, 31, 32 e 34.
- RONNEBERGER, O.; FISCHER, P.; BROX, T. U-net: Convolutional networks for biomedical image segmentation. In: SPRINGER. *International Conference on Medical image computing and computer-assisted intervention*. [S.l.], 2015. p. 234–241. Citado na página 25.
- RUSSAKOVSKY, O. et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, Springer, v. 115, n. 3, p. 211–252, 2015. Citado na página 21.
- SAEEDI, B. The jetson artificial intelligence tool chain (jai-tc). 2019. Citado na página 40.
- SANDLER, M. et al. Mobilenetv2: Inverted residuals and linear bottlenecks. *arXiv*, 2018. Disponível em: <<https://arxiv.org/abs/1801.04381>>. Citado na página 74.
- SANTOS, G. L. et al. Accelerometer-based human fall detection using convolutional neural networks. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 19, n. 7, p. 1644, 2019. Citado na página 23.
- SHAWAHNA, A.; SAIT, S. M.; EL-MALEH, A. Fpga-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access*, IEEE, v. 7, p. 7823–7859, 2018. Citado na página 21.
- SILVA, J. V. S. et al. Combining deep learning model compression techniques. *IEEE Latin America Transactions*, v. 100, n. XXX, Oct. 2021. Disponível em: <<https://latamt.ieee9.org/index.php/transactions/article/view/5824>>. Citado na página 18.
- SILVA, S. M.; JUNG, C. R. Real-time license plate detection and recognition using deep convolutional neural networks. *Journal of Visual Communication and Image Representation*, Elsevier, v. 71, p. 102773, 2020. Citado na página 55.

- SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. Citado 2 vezes nas páginas 21 e 24.
- SIPEED. *Sipeed website*. 2020. Disponível em: <<https://www.sipeed.com/>>. Citado 3 vezes nas páginas 41, 42 e 43.
- SUN, C. et al. Revisiting unreasonable effectiveness of data in deep learning era. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2017. Citado 2 vezes nas páginas 17 e 56.
- SUTSKEVER, I.; VINYALS, O.; LE, Q. V. Sequence to sequence learning with neural networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2014. p. 3104–3112. Citado na página 17.
- SÜZEN, A. A.; DUMAN, B.; ŞEN, B. Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn. In: IEEE. *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. [S.l.], 2020. p. 1–5. Citado na página 40.
- SZEGEDY, C. et al. Going deeper with convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2015. p. 1–9. Citado 3 vezes nas páginas 17, 21 e 34.
- SZEGEDY, C. et al. Rethinking the inception architecture for computer vision. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 2818–2826. Citado na página 22.
- TORRES-SÁNCHEZ, E.; ALASTRUEY-BENEDÉ, J.; TORRES-MORENO, E. Developing an ai iot application with open software on a risc-v soc. In: IEEE. *2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS)*. [S.l.], 2020. p. 1–6. Citado 3 vezes nas páginas 41, 42 e 43.
- UIJLINGS, J. R. et al. Selective search for object recognition. *International journal of computer vision*, Springer, v. 104, n. 2, p. 154–171, 2013. Citado na página 29.
- WANG, S.; SU, Z. Metamorphic testing for object detection systems. *arXiv preprint arXiv:1912.12162*, 2019. Citado na página 22.
- WEBER, M. *Caltech cars dataset*. 1999. Disponível em: <http://www.vision.caltech.edu/Image_Datasets/cars_markus/cars_markus.tar>. Citado na página 45.
- WOJNA, Z. et al. Attention-based extraction of structured information from street view imagery. In: IEEE. *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. [S.l.], 2017. v. 1, p. 844–850. Citado na página 22.
- ZHANG, Y.; YANG, Q. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114*, 2017. Citado na página 50.
- ZHAO, Z. et al. Ecart: an edge computing system for real-time image-based object tracking. In: *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. [S.l.: s.n.], 2018. p. 394–395. Citado na página 40.

ZHOU, Z. et al. A portable hong kong sign language translation platform with deep learning and jetson nano. In: *The 22nd International ACM SIGACCESS Conference on Computers and Accessibility*. [S.l.: s.n.], 2020. p. 1–4. Citado na página [41](#).

Apêndices

APÊNDICE A – Métricas das combinações de modelos testadas no sistema ALPR

Tabela 23 – Métricas das 125 combinações de modelos, os dados são do conjunto de teste

Modelo veículos	Modelo placas	Modelo caracteres	Placas corretamente reconhecidas (%)	Frame rate (FPS)
SSD	YOLOv5 Small	YOLOv5 Small	87,58	6,71
SSD	YOLOv5 Nano	YOLOv5 Small	86,41	8,47
SSD	YOLOv5 Small	YOLOv5 Nano	85,82	8,42
SSD	YOLOv5 Nano	YOLOv5 Nano	84,90	11,57
SSD	YOLOv4 Tiny	YOLOv5 Small	84,31	8,95
CenterNet	YOLOv5 Small	YOLOv5 Small	84,18	6,23
YOLOv4 Tiny	YOLOv5 Small	YOLOv5 Small	84,05	6,76
YOLOv5 Small	YOLOv5 Small	YOLOv5 Small	83,53	5,57
YOLOv4 Tiny	YOLOv5 Small	YOLOv5 Nano	83,20	8,52
CenterNet	YOLOv5 Nano	YOLOv5 Small	82,81	7,57
YOLOv5 Small	YOLOv5 Small	YOLOv5 Nano	82,68	6,64
CenterNet	YOLOv5 Small	YOLOv5 Nano	82,35	7,50
SSD	YOLOv4 Tiny	YOLOv5 Nano	82,29	12,17
YOLOv4 Tiny	YOLOv5 Nano	YOLOv5 Small	81,90	8,74
CenterNet	YOLOv5 Nano	YOLOv5 Nano	81,44	9,87
YOLOv5 Small	YOLOv5 Nano	YOLOv5 Small	81,37	6,56
YOLOv4 Tiny	YOLOv4 Tiny	YOLOv5 Small	81,24	9,08
YOLOv5 Small	YOLOv4 Tiny	YOLOv5 Small	80,65	6,86
YOLOv4 Tiny	YOLOv5 Nano	YOLOv5 Nano	80,65	11,60
SSD	CenterNet	YOLOv5 Small	80,59	7,85
YOLOv5 Nano	YOLOv5 Small	YOLOv5 Small	80,33	6,67
YOLOv5 Small	YOLOv5 Nano	YOLOv5 Nano	79,93	8,17
YOLOv4 Tiny	YOLOv4 Tiny	YOLOv5 Nano	79,93	12,15
YOLOv5 Small	YOLOv4 Tiny	YOLOv5 Nano	79,54	8,54
CenterNet	YOLOv4 Tiny	YOLOv5 Nano	79,22	10,31
YOLOv5 Nano	YOLOv5 Small	YOLOv5 Nano	78,82	8,17
YOLOv5 Nano	YOLOv5 Nano	YOLOv5 Small	78,30	8,51
YOLOv5 Nano	YOLOv4 Tiny	YOLOv5 Small	78,17	8,91
SSD	CenterNet	YOLOv5 Nano	78,17	10,52

Modelo veículos	Modelo placas	Modelo caracteres	Placas corretamente reconhecidas (%)	Frame rate (FPS)
SSD	YOLOv5 Nano	YOLOv4 Tiny	77,58	12,36
YOLOv5 Nano	YOLOv5 Nano	YOLOv5 Nano	77,12	11,12
CenterNet	CenterNet	YOLOv5 Small	76,99	7,11
YOLOv4 Tiny	CenterNet	YOLOv5 Small	76,86	7,93
SSD	YOLOv5 Small	YOLOv4 Tiny	76,47	8,99
YOLOv5 Nano	YOLOv4 Tiny	YOLOv5 Nano	75,88	11,95
YOLOv5 Small	CenterNet	YOLOv5 Small	75,23	6,27
CenterNet	CenterNet	YOLOv5 Nano	75,03	9,04
YOLOv4 Tiny	CenterNet	YOLOv5 Nano	74,97	10,32
CenterNet	YOLOv5 Small	YOLOv4 Tiny	73,86	7,97
YOLOv5 Small	CenterNet	YOLOv5 Nano	73,46	7,59
YOLOv5 Small	YOLOv5 Small	YOLOv4 Tiny	73,07	6,81
YOLOv4 Tiny	YOLOv5 Small	YOLOv4 Tiny	72,94	8,99
SSD	YOLOv4 Tiny	YOLOv4 Tiny	72,94	12,88
YOLOv5 Nano	CenterNet	YOLOv5 Small	72,48	7,88
YOLOv4 Tiny	YOLOv5 Nano	YOLOv4 Tiny	72,09	12,54
SSD	CenterNet	YOLOv4 Tiny	72,09	11,13
CenterNet	YOLOv5 Nano	YOLOv4 Tiny	72,03	10,34
YOLOv5 Small	YOLOv5 Nano	YOLOv4 Tiny	71,31	8,73
YOLOv5 Nano	CenterNet	YOLOv5 Nano	71,18	10,26
YOLOv5 Nano	YOLOv5 Small	YOLOv4 Tiny	69,15	8,76
YOLOv5 Nano	YOLOv5 Nano	YOLOv4 Tiny	69,02	12,17
CenterNet	YOLOv4 Tiny	YOLOv4 Tiny	68,63	10,77
YOLOv5 Small	YOLOv4 Tiny	YOLOv4 Tiny	68,43	8,89
YOLOv4 Tiny	CenterNet	YOLOv4 Tiny	68,37	10,93
YOLOv4 Tiny	YOLOv4 Tiny	YOLOv4 Tiny	67,84	12,96
YOLOv5 Small	CenterNet	YOLOv4 Tiny	67,52	8,07
CenterNet	CenterNet	YOLOv4 Tiny	66,93	9,48
SSD	SSD	YOLOv5 Small	66,86	9,51
SSD	SSD	YOLOv5 Nano	66,01	13,43
CenterNet	SSD	YOLOv5 Small	65,75	8,31
YOLOv5 Nano	YOLOv4 Tiny	YOLOv4 Tiny	65,29	12,62
CenterNet	SSD	YOLOv5 Nano	64,84	11,05
YOLOv5 Nano	CenterNet	YOLOv4 Tiny	64,12	10,91
YOLOv4 Tiny	SSD	YOLOv5 Small	64,12	9,48
YOLOv4 Tiny	SSD	YOLOv5 Nano	62,42	13,19
YOLOv5 Small	SSD	YOLOv5 Small	62,29	7,19

Modelo veículos	Modelo placas	Modelo caracteres	Placas corretamente reconhecidas (%)	Frame rate (FPS)
YOLOv5 Small	SSD	YOLOv5 Nano	61,90	9,05
SSD	SSD	YOLOv4 Tiny	60,52	14,50
CenterNet	SSD	YOLOv4 Tiny	59,41	11,85
YOLOv5 Nano	SSD	YOLOv5 Small	59,22	9,42
YOLOv5 Nano	SSD	YOLOv5 Nano	58,04	12,87
YOLOv4 Tiny	SSD	YOLOv4 Tiny	57,19	14,01
YOLOv5 Small	SSD	YOLOv4 Tiny	56,14	9,68
SSD	YOLOv5 Nano	CenterNet	55,82	10,67
SSD	YOLOv5 Small	CenterNet	54,58	7,99
CenterNet	YOLOv5 Small	CenterNet	53,40	7,16
CenterNet	YOLOv5 Nano	CenterNet	52,88	9,18
YOLOv5 Nano	SSD	YOLOv4 Tiny	52,61	13,90
SSD	CenterNet	CenterNet	52,29	9,68
YOLOv5 Small	YOLOv5 Nano	CenterNet	52,16	7,76
SSD	YOLOv4 Tiny	CenterNet	52,09	11,05
CenterNet	YOLOv5 Nano	SSD	51,96	10,92
YOLOv5 Small	YOLOv5 Small	CenterNet	51,70	6,25
YOLOv4 Tiny	YOLOv5 Small	CenterNet	51,57	8,04
SSD	YOLOv5 Small	SSD	51,37	9,49
YOLOv4 Tiny	YOLOv5 Nano	CenterNet	51,11	10,58
SSD	YOLOv5 Nano	SSD	50,85	13,34
CenterNet	YOLOv5 Small	SSD	50,39	8,30
CenterNet	CenterNet	CenterNet	50,26	8,43
YOLOv5 Small	YOLOv5 Small	SSD	49,87	7,11
YOLOv5 Small	YOLOv5 Nano	SSD	49,48	9,13
YOLOv4 Tiny	YOLOv5 Nano	SSD	49,48	13,21
YOLOv4 Tiny	YOLOv5 Small	SSD	49,41	9,41
CenterNet	YOLOv4 Tiny	CenterNet	49,41	9,46
YOLOv4 Tiny	YOLOv4 Tiny	CenterNet	49,08	10,97
YOLOv5 Nano	YOLOv5 Nano	CenterNet	49,02	10,42
YOLOv5 Nano	YOLOv5 Nano	SSD	48,56	12,55
YOLOv5 Nano	YOLOv5 Small	CenterNet	48,56	7,87
YOLOv5 Nano	YOLOv5 Small	SSD	48,24	9,11
YOLOv5 Small	YOLOv4 Tiny	CenterNet	48,04	8,05
YOLOv4 Tiny	CenterNet	CenterNet	47,19	9,65
YOLOv5 Small	CenterNet	CenterNet	47,06	7,31
SSD	CenterNet	SSD	46,80	11,95

Modelo veículos	Modelo placas	Modelo caracteres	Placas corretamente reconhecidas (%)	Frame rate (FPS)
SSD	YOLOv4 Tiny	SSD	45,69	13,96
SSD	SSD	CenterNet	45,36	12,23
YOLOv5 Nano	YOLOv4 Tiny	CenterNet	45,23	10,71
YOLOv5 Nano	CenterNet	CenterNet	45,23	9,59
CenterNet	SSD	CenterNet	44,44	10,25
CenterNet	CenterNet	SSD	44,25	10,02
YOLOv4 Tiny	YOLOv4 Tiny	SSD	44,12	13,65
YOLOv5 Small	CenterNet	SSD	43,40	8,45
YOLOv4 Tiny	CenterNet	SSD	42,94	11,76
YOLOv5 Small	YOLOv4 Tiny	SSD	42,88	9,42
SSD	SSD	SSD	42,09	15,89
CenterNet	YOLOv4 Tiny	SSD	42,09	11,50
YOLOv4 Tiny	SSD	CenterNet	41,90	12,05
CenterNet	SSD	SSD	41,24	12,61
YOLOv5 Nano	CenterNet	SSD	41,05	11,48
YOLOv5 Small	SSD	CenterNet	41,05	8,61
YOLOv5 Nano	YOLOv4 Tiny	SSD	40,98	13,41
YOLOv4 Tiny	SSD	SSD	40,13	15,53
YOLOv5 Small	SSD	SSD	38,76	10,24
YOLOv5 Nano	SSD	CenterNet	38,43	11,81
YOLOv5 Nano	SSD	SSD	36,93	15,03

Fonte: Elaborado pelo autor