Projeto Final de Graduação

OCR de Placas Veiculares em Dispositivos Android

André dos Santos

Orientado por
Prof. Dr. Tiago Figueiredo Vieira

Universidade Federal de Alagoas
Instituto de Computação
Maceió, Alagoas
12 de Dezembro, 2023

UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação

OCR DE PLACAS VEICULARES EM DISPOSITIVOS ANDROID

Projeto Final de Graduação submetido ao Instituto de Computação da Universidade Federal de Alagoas como requisito parcial para obtenção do Bacharelado em Engenharia de Computação.

André dos Santos

Orientador: Prof. Dr. Tiago Figueiredo Vieira

Banca Examinadora:

Tiago Figueiredo Vieria	Prof. Dr., IC-UFAL
Erick de Andrade Barbosa	Prof. Dr., IC-UFAL
Bruno Georgevich Ferreira	Me., DEI-UP

Maceió, Alagoas
12 de Dezembro, 2023

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

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

S237o Santos, André dos.
OCR de placas veiculares em dispositivos Android / André dos Santos. – 2023.
35 f. : il.

Orientador: Tiago Figueiredo Vieira.
Monografia (Trabalho de conclusão de curso em Engenharia de Computação) - Universidade Federal de Alagoas, Instituto de Computação. Maceió, 2023.

Bibliografia: f. 34-35.

1. Yolo (Algoritmo). 2. Android (Recurso eletrônico). 3. Dispositivos móveis. 4. ALPR. 5. Detecção de objetos. 6. Reconhecimento de caracteres. 7. YOLOv5s. I. Título.

CDU: 004.352.242

Agradecimentos

Gostaria de agradecer aos meus pais, Vandete dos Santos e Grigorio Vicente dos Santos, por todos os ensinamentos e esforços para que eu pudesse me dedicar aos estudos, que mesmo sem possuírem tal oportunidade, sempre incentivaram a utilização do estudo como meio de alcaçar meus objetivos.

Gostaria também de agradecer ao Warley Vital Barbosa por confiar nas minhas capacidades e pelo compartilhamento de experiências, proporcionando um desenvolvimento pessoal e profissional durante o período que trabalhamos juntos.

Finalmente, agradeço ao meu orientador, Prof. Tiago Viera, pela paciência, confiança, ensinamentos pessoais, profissionais e por aceitar de imediato me acompanhar na elaboração deste trabalho.

12 de Dezembro de 2023, Maceió - AL

Resumo

No presente trabalho efetuou-se o desenvolvimento de um sistema ALPR (do inglês, *Automatic License Plate Recognition*) para realizar o reconhecimento automático de placas veiculares em *Smartphones* Android utilizando algoritmos de detecção de objetos Yolov5s. O sistema consiste em duas etapas nomeadas *Code Detection* (CD) e *Code Recognition* (CR), respectivamente. Inicialmente, a localização da placa é efetuada na etapa CD, produzindo a caixa delimitadora, a qual é utilizada para extrair a região de interesse que será utilizada na etapa CR, responsável pelo reconhecimento dos caracteres presentes na placa. O sistema apresenta acurácia de reconhecimento de 89.6% no conjunto de teste, consumo de memória inferior a 10% durante a utilização da aplicação e eficiência energética na *GPU Mobile* consumindo aproximadamente 54% menos de bateria em comparação com a execução em *CPU Mobile*.

***Palavras-chave*: YOLO, Android, Dispositivos Móveis, ALPR, Detecção de Objetos, Reconhecimento de Caracteres, YOLOv5s.**

Abstract

We have developed an ALPR application to extract vehicle license plates from images using YOLOV5s object detection algorithms on Android mobile devices. The system consists of a two-step approach, designated Code Detection (CD) and Code Recognition (CR), respectively. Initially, the plate location is done on CD phase, yielding the plate bounding box, which is used to extract the region of interest that will be used on CR phase, responsible for executing character recognition on the plate region. The system reaches a recognition accuracy of 89.6% on the test set, memory consumption lower than 10% during application execution, and power efficiency on Mobile GPU consuming about 54% battery lower than Mobile CPU execution.

***Keywords*: *YOLO*, *Mobile Devices*, *ALPR*, *Object Detection*, *Character Recognition*, *ALPR*, *YOLOv5s*.**

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

ALPR *Automatic License Plate Recognition*

CD *Code Detection*

CR *Code Recognition*

YOLO *You Only Look Once*

SSD *Single-Shot Multibox Detector*

CNN *Convolutional Neural Network*

RCNN *Region-based Convolutional Neural Networks*

FP32 *Single-Precision Floating-Point*

FP16 *Half-Precision Floating-Point*

AP *Average Precision*

mAP *Mean Average Precision*

SOTA *State Of The Art*

IoU *Intersection over Union*

NPU *Neural Processing Unit*

DSP *Digital Signal Processor*

Conteúdo

Capítulo 1

Introdução

1.1 Motivação

O estudo do reconhecimento automático de placas veiculares possui aplicação prática diversos setores do transporte e segurança, como aplicação das leis, controle de acesso, controle de pesagem rodoviária e inspeção veicular. O desenvolvimento de tais sistemas tem crescido com o avanço da tecnologia, como câmeras dedicadas com *software* embarcado, proporcionando portabilidade e confiança na obtenção das informações pertinentes acerca dos veículos. Todavia, esta tecnologia necessita de alto investimento, demanda treinamento para utilização e um aparato técnico para o correto funcionamento, restringindo-se a comercialização empresarial. Em contrapartida, dispositivos móveis como *Smartphones* provêm portabilidade, acessibilidade e capacidades computacionais propiciais para execução destes sistemas, fornecendo resultados compatíveis com as restrições de recursos presentes no ambiente móvel.

Como apresentado em Ignatov et al. (2019), dispositivos móveis como *Smartphones* atuais apresentam capacidades computacionais equivalentes a GPUs (*Graphics Processing Unit*) de *desktops* lançadas anos atrás, permitindo ainda a integração de tecnologias como NPUs (*Neural Processing Unit*) e DSPs (*Digital Signal Processor*) para execução de modelos de Aprendizado Profundo em dispositivos de todos os seguimentos de consumo, abrangendo desde modelos básicos até avançados. A adoção de técnicas de Aprendizado Profundo no desenvolvimento de sistemas ALPR (*Automatic License Plate Recognition*) tem substituído implementações tradicionais, Shashirangana et al. (2021) aponta que esta tendência está relacionada ao alto desempenho de métodos atuais de detecção de objetos baseados em Aprendizado Profundo quando aplicados em situações restritas, apresentando robustez a modificações do ambiente, viabilizando adaptação em diversos cenários de aplicação.

A disponibilidade de dispositivos móveis capacitados para execução de modelos de Aprendizado Profundo e o alto desempenho de tais modelos em sistemas ALPR favorece o estudo e desenvolvimento de soluções portáteis, solucionando os obstáculos presentes em aplicações tradicionais, como custo excessivo de equipamentos e treinamento especializado.

1.2 Objetivos

1.2.1 Objetivos Gerais

Desenvolver um sistema ALPR para realizar o reconhecimento automático de placas veiculares utilizando modelos de detecção de objetos em um dispositivo móvel Android.

1.2.2 Objetivos Específicos

1. Apresentar o processo de coleta e rotulação das imagens para a construção da base de dados.

2. Utilizar o conjunto de ferramentas do *framework* Tensorflow Lite para permitir a execução do sistema no ambiente móvel.

3. Realizar *benchmarks* para comparar o desempenho do sistema no *Desktop* e no *Smartphone Android*.

1.3 Organização do Trabalho

Este trabalho foi organizado em capítulos que sintetizam etapas do desenvolvimento da solução, abordando detalhes teóricos a tecnologias aplicadas. O capítulo 2 apresenta o embasamento teórico acerca de sistemas de ALPR, Aprendizado Profundo aplicado a este contexto, sistemas de ALPR mobile e métricas utilizadas para avaliar o desempenho do sistema proposto. O capítulo 3 descreve a metodologia empregada no desenvolvimento do trabalho, detalhando a aquisição e rotulação dos dados, os conjuntos de teste e validação para execução dos *benchmarks* em *Desktop* e *Smartphone*, arquitetura do sistema e detalhes acerca da aplicação Android desenvolvida. O capítulo 4 é dedicado a apresentação e discussão dos resultados. Finalmente, conclusões sobre o trabalho são apresentadas no capítulo 5.

Capítulo 2

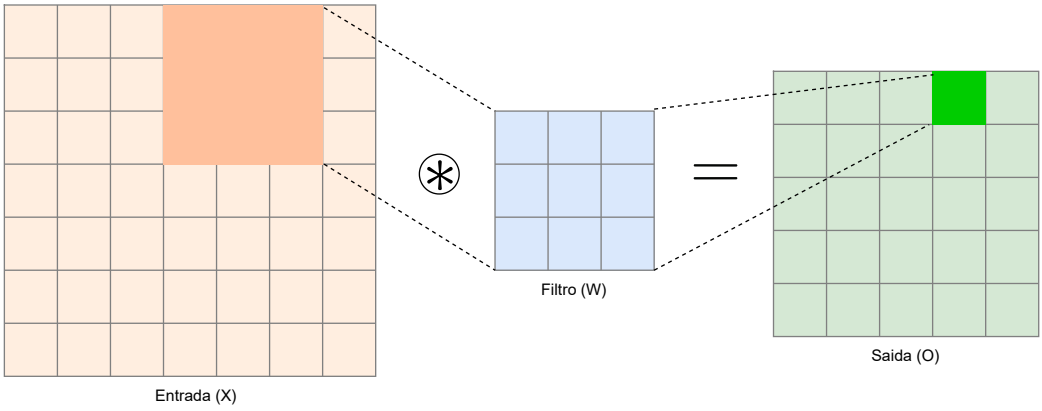
Fundamentação Teórica

2.1 Conceitos Fundamentais

2.1.1 Redes Neurais Convolucionais

Redes Neurais Convolucionais ou CNNs (do inglês, *Convolutional Neural Networks*) são especificações de redes neurais artificiais voltadas para o tratamento de dados representados em vetores multidimensionais, como imagens, vídeos ou espectrogramas. O estudo e desenvolvimento de tais arquiteturas baseiam-se no fato de que o reconhecimento de pequenos padrões em estágios iniciais permitem o entendimento global do conteúdo em análise. Como apresentado em LeCun et al. (2015), CNNs são caracterizadas pela presença de camadas convolucionais e *pooling*. Camadas convolucionais são utilizadas nos estágios iniciais para aquisição de características acerca do conteúdo presente no dado de entrada, efetuando convoluções com filtros aprendidos durante o treinamento dos modelos.

Figura 2.1: A operação de convolução entre o dado de entrada X (a esquerda) e o filtro W (ao centro) produzindo o mapa de características O (a direita).



Fonte: Autor

A convolução pode ser interpretada como uma operação de sobreposição do filtro com uma sub-região da entrada, contendo as mesmas dimensões. O filtro utilizado no pro-

cesso de convolução visa identificar características e padrões contidos na entrada X, que englobam elementos mais simples nas camadas iniciais, como arestas e contornos, até características mais complexas nas camadas mais profundas, como padrões hierárquicos. Os pesos do filtro W são encontrados durante o treinamento em um processo de minimização do erro e o funcionamento da operação de convolução é apresentado na Figura 2.1, cuja convolução é representada por $\circledast$.

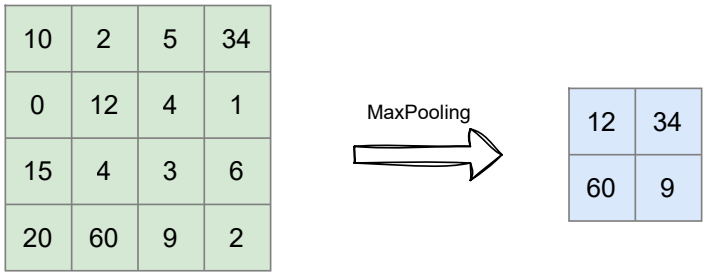
$$O\left[i,j\right] = \left(X \circledast W\right)\left[i,j\right] = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} X\left[i+m, j+n\right] \cdot W\left[m,n\right] \tag{2.1}$$

A definição matemática da convolução discreta é apresentada na equação 2.1, onde:

- X representa a entrada
- W representa o filtro
- O representa a saída
- M e N representam as dimensões do filtro e da entrada, respectivamente.
- m e n são utilizados para iterar através das linhas e colunas do filtro W e da entrada X durante a operação de convolução.
- i e j são utilizados para preencher o mapa de características resultante O durante cada convolução entre uma sub-região da entrada X e o filtro W, atribuindo o valor a posição $O[i,j]$ da saída.

As camadas de *pooling* são utilizadas para permitir a redução de dimensionalidade e condensar as informações mais importantes obtidas após a etapa de convolução. Existem diferentes tipos de *Pooling*, como MaxPooling e AveragePooling, diferindo somente na forma como a computação da saída é realizado. MaxPooling busca enfatizar os maiores valores, enquanto AveragePooling busca obter uma média dentro da sub-região da entrada delimitada pela camada de *pooling*.

Figura 2.2: Operação de MaxPooling para um kernel 2x2 sobre um mapa de característica A produzindo um segundo mapa com a informação condensada O.



Fonte: Autor

O funcionamento da operação de MaxPooling é apresentado na Figura 2.2, produzindo um novo mapa de característica de saída O reduzido contendo os maiores valores de cada sub-região. Camadas de convolução e *Pooling* são elementos característicos de CNNs assim como funções de ativação, responsáveis pela adição de não-linearidade, possibilitando ainda a utilização conjunta com estratégias de normalização e regularização.

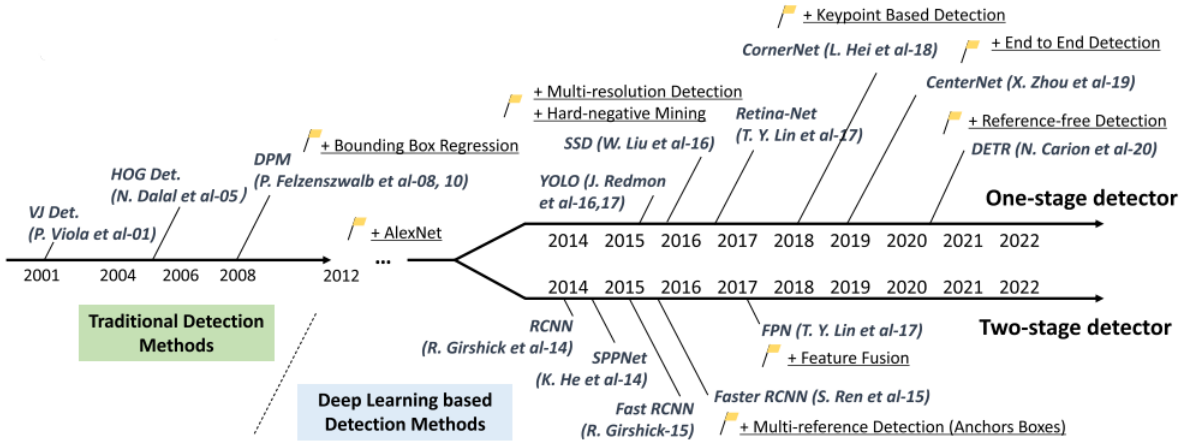
2.1.2 Detecção de Objetos

Detecção de Objetos é uma área da visão computacional destinada ao estudo e desenvolvimento de técnicas para efetuar a localização e classificação de instâncias de objetos em imagens digitais. Zou et al. (2023) apresentam uma síntese da evolução desta área de estudo nos últimos 20 anos. A ascensão do Aprendizado Profundo foi um marco para o desenvolvimento de técnicas contemporâneas de detecção de objetos, promovendo ampla adoção de detectores baseados em CNNs.

A Figura 2.3 apresenta a cronologia do estudo e desenvolvimento de modelos para resolução de problemas de detecção de objetos. O estudo contemporâneo de detecção de objetos teve seu ponto de partida significativo em 2012, com a introdução do modelo AlexNet, fundamentado em Redes Neurais Convolucionais. Este modelo alcançou taxas de erro substancialmente reduzidas no desafio ImageNet, definido como uma competição de avaliação de modelos em tarefas de classificação e localização de objetos. Esse marco evidenciou de maneira convincente o potencial de Redes Neurais Convolucionais de oferecer contribuições significativas para a área de detecção de objetos.

Detectores tradicionais como Viola Jones e HOG eram referências no estudo de detecção de objetos, utilizando técnicas como AdaBoost e distribuição de gradientes para realizar a detecção e localização nas imagens de entrada, mas apresentavam desvantagens quando submetidos a ambientes restritos. A partir de 2012, os detectores desenvolvidos passaram a adotar majoritariamente modelos de CNNs como base da implementação,

Figura 2.3: Cronologia de Detecção de Objetos



Fonte: Zou et al. (2023)

dividindo-se em duas abordagens:

- **Detector de estágio único** ou "*One-Stage Detector*": Classificação dos objetos e regressão das caixas delimitadoras são feitas de simultânea.
- **Detector de dois estágios** ou "*Two-Stage Detector*": Regiões de possíveis objetos são propostas em um primeiro estágio e a classificação dos objetos contidos nas regiões prováveis é realizada em um segundo estágio, para cada região proposta.

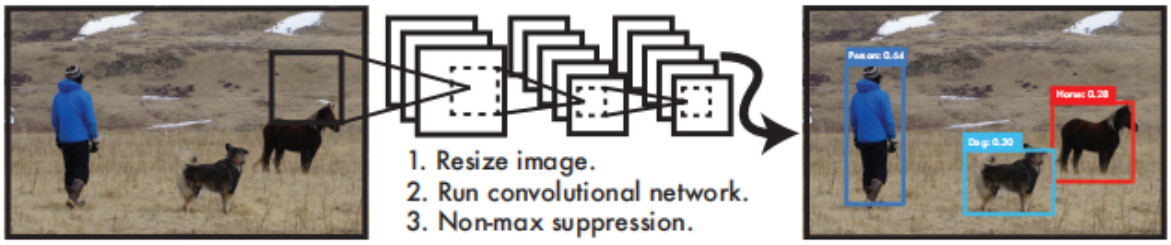
Algoritmos como *RCNN*, *Fast RCNN* e *Faster RCNN* são detectores de dois estágios, diferindo somente pela adição de melhorias incrementais, mas mantendo-se a ideia inicial de utilizar regiões propostas em um estágio inicial para então executar a classificação. Tais detectores apresentam problemas relacionados a complexidade e velocidade de execução devido à abundância na quantidade de regiões provindas do estágio inicial, dificultando a aplicabilidade de tais algoritmos em aplicações reais.

Diante dos problemas presentes na estratégia de dois estágios, detectores de um estágio como YOLO, SSD e RetinaNet foram desenvolvidos visando executar a localização e classificação dos objetos em uma única etapa, permitindo a obtenção das caixas delimitadoras e classes de tais objetos simultaneamente.

2.1.3 YOLO: You Only Look once

Conforme apresentado em 2.1.2, YOLO é um algoritmo de detecção de objetos de estágio único (*One-Stage*) desenvolvido por Redmon et al. (2016). Este algoritmo busca tratar o problema de detecção de objetos como um simples problema de regressão, utilizando a imagem de entrada para inferir as caixas delimitadoras e as probabilidades das classes presentes em cada caixa simultaneamente.

Figura 2.4: Sistema de detecção YOLO. (1) Redimensionamento da imagem para 448x448, (2) execução de um única CNN na entrada, (3) e pós-processamento com *NMS* para obter as caixas delimitadoras e as probabilidades finais.



Fonte: Redmon et al. (2016)

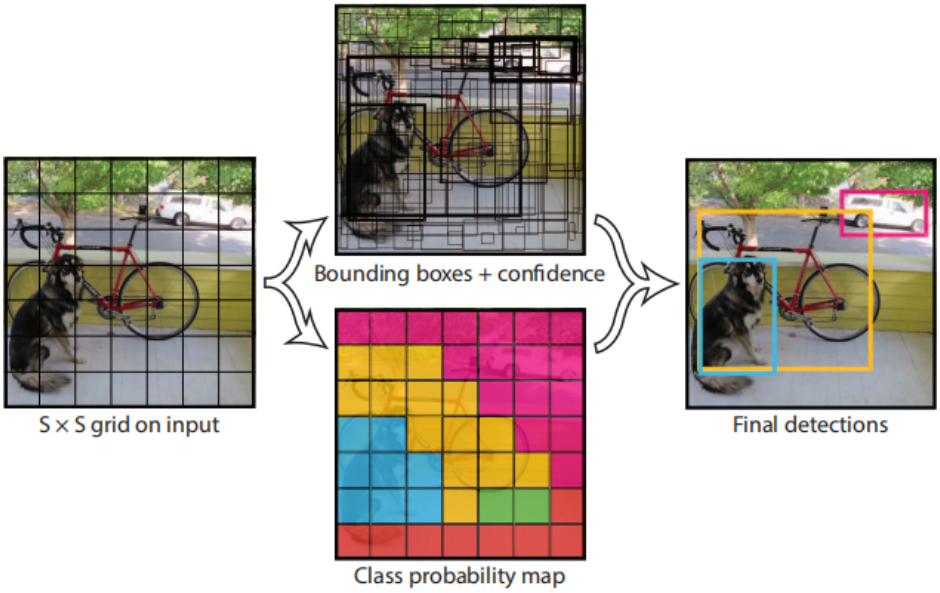
O sistema de detecção do modelo YOLO é mostrado na Figura 2.4, constituído por uma única CNN responsável por inferir as caixas delimitadoras e as probabilidades dos objetos contidos na imagem de entrada em uma única avaliação, comumente chamada de processo de *forward*. Estas características possibilitam que o sistema seja rápido no processo de inferência e analise a imagem em sua totalidade, diferente de técnicas baseadas em *slide windows* e propostas de regiões, codificando implicitamente informações contextuais sobre as classes, bem como sua aparência.

A arquitetura da rede YOLO busca unificar os componentes de detecção de objetos em uma única rede, habilitando o treinamento *end-to-end* e a utilização em aplicações de *streaming* de vídeo com até 25ms de latência. O algoritmo executa as seguintes fases:

1. Divide a imagem em um *grid* $S \times S$ células.
2. Cada célula estima B caixas delimitadoras e suas confianças de detecção.
3. Cada célula estima C probabilidades condicionais das classes.

A Fase 2 atribui a responsabilidade da detecção dos objetos as células que contenham o centro desses objetos, inferindo as caixas delimitadoras juntamente com a confiança de detecção, que representa a confiança do modelo sobre a presença de um objeto em determinada caixa. A Fase 3 produz as C probabilidades para cada objeto, representando a confiança do modelo na identificação do objeto como uma classe c.

Figura 2.5: A rede YOLO modela a detecção como um problema de regressão dividindo a imagem em uma *grid* $S \times S$, inferindo B caixas delimitadoras e a confiança associada a essas caixas, além das probabilidades das C classes. Essas previsões são codificadas como um tensor $S \times S \times (B * 5 + C)$.



Fonte: Redmon et al. (2016)

A representação gráfica do funcionamento do modelo YOLO é apresentada na Figura 2.5, seguindo as fases enumeradas em 2.1.3. As detecções finais são obtidas mediante a aplicação do método *Non-Max Suppression* (NMS), que visa eliminar detecções redundantes ao considerar todas as caixas detectadas. Essa arquitetura constitui a primeira versão do algoritmo YOLO, servindo como base para o desenvolvimento de uma família de modelos subsequentes, caracterizados por melhorias incrementais.

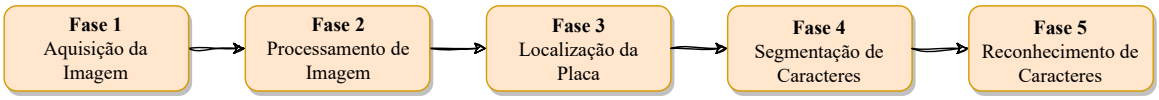
2.2 Revisão da Literatura

2.2.1 Sistemas ALPR

A placa é um identificador veicular único que possibilita acesso aos dados do veículo e do proprietário, sendo uma das informações de maior relevância utilizada como entrada para sistemas de consulta veicular. O conceito de ALPR foi proposto inicialmente em 1976 para auxiliar forças policiais na aplicação das leis de trânsito, contudo passou a ser adotado de forma ampla somente na década de 1990. Esta idealização é considerada uma ramificação do conceito de OCR, especializando-se no reconhecimento de caracteres presentes na região delimitada pela placa, onde geralmente o formato do texto, a distâncias entre caracteres e os tipos de fontes são pré-determinados.

Um sistema convencional de ALPR é constituído de diversas fases como apresentado

Figura 2.6: Fases de um sistema tradicional de ALPR



Fonte: Autor

em F. et al. (2021), Huang and Wang (2022) e Godage and Wimalaratne (2019), representadas de forma ampla na Figura 2.6. Após a aquisição dos dados, técnicas tradicionais de processamento de imagens e visão computacional como detecção de bordas e segmentação são utilizadas para extrair a região da placa e efetuar a separação dos caracteres, respectivamente. Com os caracteres devidamente separados, estratégias como Correspondência de Padrões (*Template Matching*) são utilizadas para reconhecer os caracteres, retornando ao final do processo o texto contido na placa. O avanço constante da tecnologia proporcionou a utilização de métodos mais robustos em algumas destas fases, como utilização de Redes Neurais Profundas, no contexto de detecção de objetos, para seleção da região da placa e classificadores para reconhecimento de caracteres.

2.2.2 ALPR e Aprendizado Profundo

Aplicações de ALPR têm como foco extrair o texto presente na região delimitada pela placa, permitindo a digitalização, armazenamento e utilização em consultas em sistemas veiculares. Lecun et al. (1998) realizou o primeiro trabalho baseado em CNNs para efetuar o reconhecimento de dígitos escritos à mão utilizando o conjunto de dados MNIST, inspirando a aplicação de Aprendizado Profundo no contexto de reconhecimento de caracteres e consequentemente aplicações OCR baseadas em Redes Neurais.

A utilização de modelos da família YOLO em aplicações de ALPR passou a ser bastante empregada devido ao ótimo custo-benefício em tempo de inferência e acurácia. Montazzolli and Jung (2017) desenvolveram um sistema de ALPR fim-a-fim para veículos brasileiros com placas no formato antigo, onde as etapas de localização da placa e reconhecimento de caracteres utilizavam dois modelos FAST-YOLO em sequência, de forma que a saída do primeiro alimentava a entrada do segundo. Riaz et al. (2020) e Cabral et al. (2021) utilizaram YOLOv3 e YOLOv4, respectivamente, para detecção do veículo sucedida da detecção da placa, buscando restringir a fase de localização da placa à região da imagem original obtida através da caixa delimitadora do veículo detectado. Estes trabalhos salientam a dificuldade de detectar diretamente a região da placa, fomentando a necessidade da detecção do veículo em um primeiro estágio devido à pequena proporção da placa com relação à imagem completa, caracterizando o problema de detecção de pequenos objetos.

No cenário de veículos brasileiros, a adoção do padrão de placas Mercosul em 2020 adicionou a problemática da existência simultânea de placas no formato Antigo (LLLNNNN) e Mercosul (LLLNLNN), onde L representa a presença de letras e N a presença de números em determinada posição. A dificuldade está no fato do quinto caractere assumir valores numéricos e alfabéticos, quando apenas números eram possíveis para esta posição no formato Antigo (LLLNNNN). Tal problema pode amplificar as confusões de caracteres parecidos, como O e 0, B e 8, 1 e I. Santos et al. (2023) apresentou um sistema de ALPR para vigilância em universidades brasileiras, onde utilizou YOLOv5 nas etapas de localização da placa e detecção do modelo, Antigo ou Mercosul, buscando amenizar o impacto da presença de placas em ambos os formatos e os desafios no reconhecimento do quinto caractere.

2.2.3 ALPR e Dispositivos Mobile

A portabilidade pode ser uma característica importante em sistemas de ALPR, diante da necessidade de locomoção, variação da perspectiva e ângulo para melhorar a captura dos dados referida na primeira fase do pipeline da Figura 2.6. O estudo e desenvolvimento de sistemas de ALPR em plataformas móveis tornou-se uma alternativa mais versátil e acessível devido ao surgimento de dispositivos embarcados especializados em execução de Aprendizado Profundo, como NVIDIA Jetson, e o desenvolvimento de chips mais eficientes, permitindo que dispositivos como *smartphones* e Raspberry Pi sejam utilizados na construção de tais sistemas.

A adoção de Aprendizado Profundo em dispositivos *Mobile* esta em ascensão devido disponibilidade de dispositivos aptos computacionalmente, *hardware* dedicado e *frameworks* que permitem uma implementação fácil e eficiente destes modelos. Ignatov et al. (2019) apresenta uma visão detalhada da diversidade e potencial dos *smartphones* atuais para a execução de modelos de Aprendizado Profundo. O desenvolvimento de SoCs (*System on a chip*) modernos e acessíveis, com hardware dedicado a execução de Redes Neurais como NPUs e aceleração Gráfica como GPUs permitem que desenvolvedores dediquem-se ao estudo e implementação de sistemas *mobile* de Aprendizado Profundo, fornecendo resultados semelhantes a GPUs de *desktops* apresentadas há alguns anos.

O desenvolvimento de sistemas ALPR em dispositivos móveis possui grandes desafios relativos a restrições dos recursos. Godage and Wimalaratne (2019) apresentaram uma solução de ALPR para análise de placas do Sri Lanca, baseada em técnicas clássicas como detecção de bordas, operações morfológicas e manipulação de histogramas de imagens em conjunto com a utilização de bibliotecas conceituadas como OpenCV e Tesseract. Huang and Wang (2022) propôs uma solução híbrida para análise de licenças veiculares chinesas na qual utiliza correção de distorção, detecção de bordas e segmentação por cor para obter possíveis candidatos a regiões da placa, utilizando ao final desta fase uma modificação da

LeNet-5 para a seleção da região mais provável. Izidio et al. (2018) propôs um sistema embarcado em Raspberry Pi 3 para reconhecimento de placas no padrão brasileiro antigo, utilizando Tiny YOLOv3 para as fases de localização da placa e uma CNN para o reconhecimento de caracteres. O sistema era alimentado por bateria, o que proporcionava a mobilidade, todavia problemas como baixa iluminação e inclinação da placa poderiam prejudicar o desempenho do sistema. No contexto de aplicações ALPR *mobile*, ainda existe uma preferência na utilização de bibliotecas conceituadas como OpenCv e Tesseract, mas o avanço na área de detecção de objetos tem contribuído para mudança rápida nas estratégias de implementação de tais sistemas, como apresentado em Shashirangana et al. (2021).

2.3 Métricas de Avaliação

2.3.1 Acurácia de ALPR

Acurácia refere-se a proximidade do valor predito ser igual ao valor esperado. Tal métrica é utilizada para obter o desempenho do pipeline completo quando todos os 7 caracteres são considerados. O cálculo da acurácia é obtido utilizando a expressão:

$$Acuracia = \frac{\#\text{Amostras Corretas}}{\#\text{Amostras Totais}} \tag{2.2}$$

2.3.2 Mean Average Precision (mAP)

A Precisão Média (mAP) é uma métrica utilizada em tarefas de detecção de objetos para representar o desempenho nas tarefas de localização e classificação de objetos em uma imagem. Elementos fundamentais para o cálculo de mAP são as métricas de classificação precisão (do inglês, *Precision*), sensibilidade (do ingês, *Recall*) e o conceito de IoU (do inglês, *Intersection over Union*) relativo à sobreposição das caixas delimitadoras. Tais elementos são definidos pelas equações 2.3, 2.4 e 2.5.

$$Precision = \frac{TP}{TP + FP} \tag{2.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

$$IoU = \frac{\text{Área de sobreposição}}{\text{Área de União}} \tag{2.5}$$

Quando utilizadas no contexto de detecção de objetos, os elementos TP, FP e FN encontrados nas equações 2.3 e 2.4 podem ser interpretados como:

- **TP** é contabilizado quando o objeto esta presente no *groundtruth* e o modelo foi capaz de encontrá-lo. Em detecção de objetos, ocorre quando o IoU das caixas do objeto predito e do *groundtruth* é superior a um limiar X predefinido.

- **FP** É contabilizado quando o modelo não identifica um objeto que se encontra no *groundtruth*. Em detecção de objetos, ocorre quando IoU das caixas do objeto predito e do objeto verdadeiro é inferior a um limiar X, ou quando não há objetos na imagem e o modelo efetuou uma detecção inexistente no *groundtruth*.

- **FN** É contabilizado quando o objeto existe no *groundtruth*, mas o modelo não detecta, ou seja, quando o modelo não gera um nenhuma predição para o objeto real.

A precisão média (AP) é entendida como a área abaixo da curva *precision* x *recall*, comprimindo a informação de tal curva como um valor escalar.

$$AP = \sum_{k=1}^{n-1} [r(k) - r(k-1)] \times \max\left(p(k), pk(k-1)\right) \tag{2.6}$$

A equação 2.6 é utilizada para obter o valor da precisão média (AP) aproximando a curva *precision-recall* por retângulos, onde a largura de cada retângulo é fornecida por $r(k) - r(k-1)$ e a altura $max\left(p(k), p(k-1)\right)$.

$$mAP = \frac{1}{\mathrm{M}} \sum_{i}^{\mathrm{M}} AP_i \tag{2.7}$$

A equação 2.7 é utilizada para computar o valor de mAP, que é obtido tomando-se a média das AP entre as M classes presentes no conjunto de dados.

2.3.3 Distância de Levenshtein

A Distância de Levenshtein, também conhecida como Distância de Edição, é uma métrica de distância entre *strings* que computa o número mínimo de inserções, exclusões ou substituições necessárias para transformar uma cadeia de caracteres de entrada X em uma sequência de caracteres desejada Y. A definição recursiva da Distância Levenshtein é apresentada na equação 2.8.

$$lev(X,Y) = \begin{cases} |X| & \text{if } |Y| = 0 \\ |Y| & \text{if } |X| = 0 \\ lev\left(X[1:], Y[1:]\right) & \text{if } X[0] = Y[0] \\ 1 + min \begin{cases} lev\left(X[1:], Y\right) \\ lev\left(X, Y[1:]\right) \\ lev\left(X[1:], Y[1:]\right) \end{cases} & \text{caso contrário} \end{cases} \tag{2.8}$$

Abaixo, um exemplo didático do funcionamento desta métrica, onde:
X = "casa" e Y = "caixa".

1. Inserir a letra "i" antes do "s" em casa, tornando-a "caisa".
2. Substituir "s" por "x" em "caisa", tornando-a "caixa".

Como foram utilizadas duas operações, $lev(\text{casa}, \text{caixa}) = 2$. Neste trabalho, tal métrica é utilizada para obter uma análise das modificações necessárias nas predições incorretas do sistema proposto, extraindo-se uma média dos valores da Distância de Levenshtein para as amostras com a *string* da placa predita não nula.

Capítulo 3

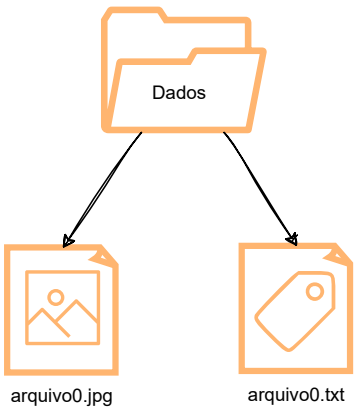
Metodologia

3.1 Conjunto de Dados

3.1.1 Aquisição e Rotulação

As imagens utilizadas para o desenvolvimento dos modelos foram coletadas de câmeras fixas em postos de pesagem rodoviária, onde sensores alocados em torno da balança acionam a captura do *frame* atual dos veículos posicionados na via de pesagem. As amostras são capturadas com diferentes níveis de brilho, horários e perspectivas, como imagens dianteiras e frontais dos veículos.

Figura 3.1: Estrutura das anotações do formato YOLO

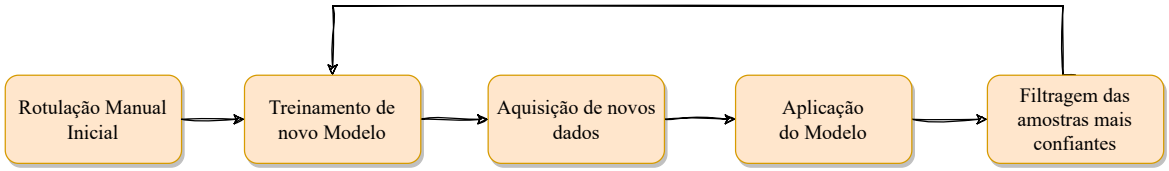


Fonte: Autor

A Figura 3.1 representa a estrutura de armazenamento dos dados após o processo de rotulação. O arquivo de rótulo segue o formato YOLO para detecção de objetos, onde cada objeto presente na imagem corresponde uma linha do arquivo de texto, a qual inclui informações sobre a classe e a caixa delimitadora do objeto. A estratégia de utilizar um sistema de ALPR completamente baseado em detecção de objetos, permite

que anotações para as etapas de Detecção de código (CD) e Reconhecimento de Caracteres (CR) compartilhem o mesmo processo, excetuando-se o modelo utilizado. A abordagem adotada assemelha-se à aplicada em Cabral et al. (2021), onde o emprego de rotulação semi-supervisionada permite utilizar os modelos adquiridos em uma primeira etapa, de forma cíclica e iterativa até a completa rotulação dos dados. O processo de rotulação utilizou ferramentas dedicadas como Wada (2022) em um estágio inicial e opencv (2023) posteriormente, devido às facilidades de exportação das anotações.

Figura 3.2: Processo iterativo de rotulação dos dados utilizando uma estratégia semi-supervisionada.



Fonte: Autor

A Figura 3.2 representa o processo contínuo de rotulação. A etapa de Rotulação Manual indicada no primeiro bloco foi realizada apenas uma vez, neste processo foram rotuladas manualmente cerca de 10000 imagens para as tarefas CD e CR. Esta etapa do desenvolvimento do sistema possui grande impacto na qualidade dos modelos, demandando empenho e responsabilidade de todos envolvidos no processo. A rotulação das imagens para a tarefa CD consiste em marcar a caixa delimitadora da placa na imagem original do veículo, a qual possui resolução base de 640x480. Para a tarefa de CR, obtém-se a região da placa indicada na rotulação de CD, delimitando-se as caixas de todos os caracteres contidos em tal região. O processo se repete até que todos os dados sejam rotulados ou descartados, quando não satisfazem a filtragem de confiança indicada no último bloco.

3.1.2 Conjunto de Dados

Os conjuntos de teste e validação são fundamentais para conferir o desempenho dos modelos em amostras distintas utilizadas no treinamento. Tais conjuntos são utilizados como dados de entrada dos *benchmarks* desenvolvidos para execução em *Desktop* e Android, que utilizam *hardware* distinto visando a obtenção de métricas relativas à capacidade do sistema proposto em reconhecer corretamente o texto contido na região da placa. O conjunto de validação possui 1260 imagens adquiridas de postos de pesagens em rodovias brasileiras, sendo utilizado para comparar o desempenho do modelo TFLite relativo ao modelo em Pytorch no ambiente de execução Desktop, a fim de verificar se a conversão entre os *frameworks* prejudicou o desempenho do modelo que rodará na aplicação final.

O conjunto de teste possui 58 amostras coletas manualmente em situações cotidianas, utilizando o dispositivo Zebra TC210K, tomado como referência para os testes de desempenho. Acerca da qualidade das imagens, a resolução da câmera foi configurada para 8.9MB (2976x2976) e a proporção da imagem foi mantida em 1:1 para facilitar o redimensionamento e ajuste para a entrada do modelo, simulando a funcionalidade da região de interesse (*Region of Interest*) implementada somente na versão completa da aplicação, onde originalmente escolhe-se uma sub-região quadrada com proporção 1:1 da imagem original, restringindo espaço de busca do detector de placa.

Figura 3.3

Fonte: Autor

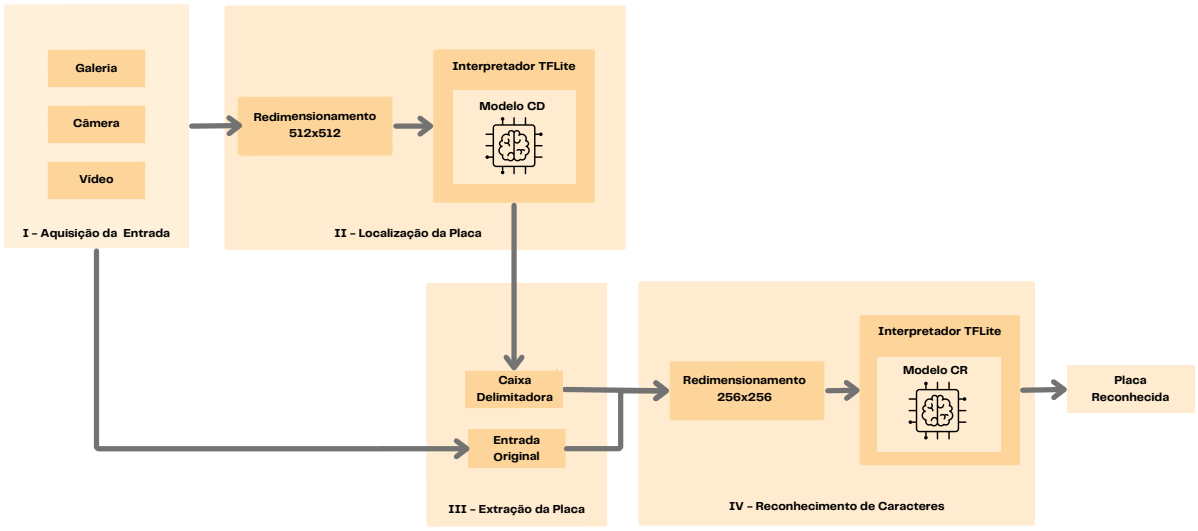
Algumas amostras presentes no conjunto de teste são apresentadas na Figura 3.3, apresentando variações em diferentes pontos de vista, como traseira e frontal, iluminação e inclinação da câmera do *smartphone* no momento da captura. A preservação das informações dos veículos e proprietários é feita omitindo-se as placas de cada veículo com máscara de privacidade definida por um retângulo preto.

3.2 Arquitetura do Sistema

O sistema proposto utiliza dois modelos de detecção de objetos YOLOv5s para efetuar o reconhecimento da placa, executando de forma sequencial a localização da placa e o reconhecimento de caracteres, apresentadas nas fases 3 e 5 da Figura 2.6, respectivamente. O sistema foi desenvolvido utilizando a versão *s* (do inglês, *small*) do modelo YOLOv5, representando a segunda versão mais simples deste algoritmo, que possui versões menores como YOLOv5n (*nano*) e versões mais complexas como YOLOv5m (*medium*), YOLOv5l (*large*) e YOLOv5x (*extra-large*). A disponibilidade dessas variantes amplia a aplicabilidade da YOLOv5 em uma variedade de contextos, abrangendo desde aplicações que demandam processamento ágil, como as versões *n* e *s*, até aquelas que requerem elevada precisão de detecção e pontuações de *mAP*, mesmo que isso implique em tempos de processamento mais extensos, como observado nas versões *m*, *l* e *x*.

O modelo YOLOv5s possui a capacidade de localizar e classificar o objeto simultaneamente, sem a necessidade da segmentação de objetos apresentada na fase 4 da Figura 2.6. Esta característica possibilita também sua aplicação na tarefa de reconhecimento dos caracteres que ocorre após a fase de localização da placa. Os modelos foram nomeados seguindo a tarefa a qual desempenham, CD (do inglês, *Code Detector*) para detecção e localização da placa e CR (do inglês, *Code Recognizer*) para o reconhecimento de caracteres. Para a fase de localização da placa, o modelo CD possui apenas a classe *placa* como alvo, enquanto o modelo CR possui 36 classes possíveis, de *A...Z* e *0...9* para a fase de reconhecimentos dos caracteres aplicada à região da imagem original delimitada pela caixa resultante da aplicação do CD.

Figura 3.4: Visão Geral do Sistema ALPR proposto.



Fonte: Autor

A Figura 3.4 apresenta o processo de execução do sistema proposto, o qual baseia-se em uma estratégia sequencial, onde a saída do modelo CD é utilizada para extrair a região da placa da imagem original e alimentar o modelo CR. Etapas de redimensionamento da entrada dos modelos são as únicas operações de processamento de imagens efetuadas, e foram omitidas do diagrama. O funcionamento do sistema é descrito abaixo:

1. **Redimensionamento da Imagem**: Após aquisição da imagem, o redimensionamento para 512x512 é feito para garantir compatibilidade com a entrada do modelo CD.

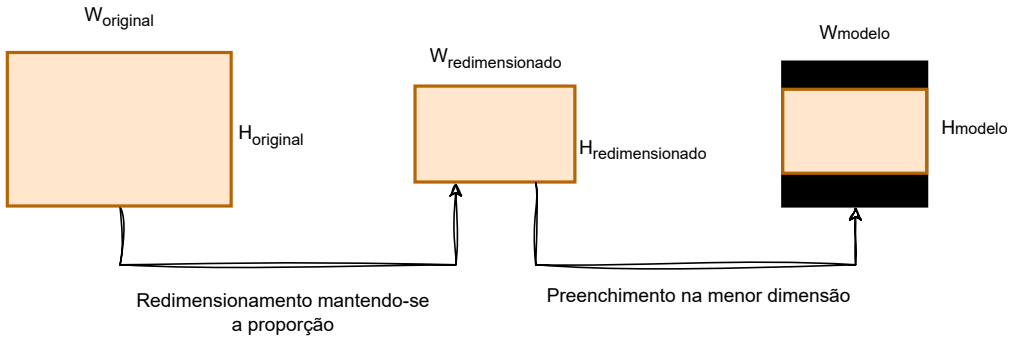
2. **Localização da Placa**: O modelo CD utiliza-se da imagem redimensionada para efetuar a localização da região da placa, produzindo como saída a caixa delimitadora de tal região.

3. **Redimensionamento da Placa**: Após aquisição da região da placa, o redimensionamento para 256x256 é feito para garantir compatibilidade com a entrada do modelo CR.

4. **Reconhecimento dos Caracteres**: O modelo CR utiliza a região da placa redimensionada para efetuar o reconhecimento de caracteres, produzindo as caixas delimitadoras, classes e confianças de detecção.

O redimensionamento da entrada dos modelos é fundamental para o funcionamento correto do sistema, tal processamento é necessário devido à possibilidade da imagem original assumir dimensões variadas e proporções diferentes de 1:1, o que deve ocorrer com frequência diante da diversidade de resoluções de câmeras presentes nos dispositivos suportados pela aplicação.

Figura 3.5: Estrategia de redimensionamento



Fonte: Autor

O processo de redimensionamento é apresentado na imagem 3.5, sendo realizado para a entrada de cada modelo. O objetivo é manter a proporção entre largura e altura de tal forma que a maior dimensão seja mapeada para o tamanho ideal escolhido para a entrada do modelo, adicionando-se preenchimentos simétricos ao longo da dimensão menor após o redimensionamento para obter uma nova imagem com dimensão quadrada, com as dimensões relativas de cada modelo.

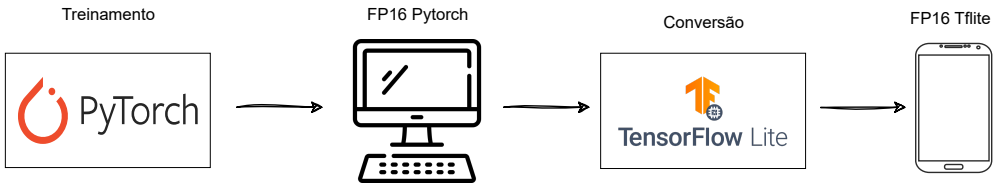
3.2.1 Exportação dos modelos para TFLite

A implantação de modelos de Inteligência Artificial em plataformas com recursos limitados é um desafio atual que depende das particularidades do dispositivo alvo. Em um contexto de utilização de Aprendizado Profundo, é necessário utilizar estratégias de quantização ou redução da precisão da representação dos dados para adaptar os modelos as limitações de memória e consumo energético. A utilização de *frameworks* para efetuar tais otimizações contribuem para a obtenção de implementações eficientes e facilidade na realização de testes, permitindo que desenvolvedor realize testes com diversas modificações visando encontrar uma configuração que satisfaça as necessidades da tarefa. Tensorflow Lite é um conjunto de ferramentas utilizado para facilitar a execução de Aprendizado de Máquina (*Machine Learning*) em dispositivos móveis, embarcados e IoT (*Internet of Things*), no presente trabalho esta ferramenta foi utilizada especificadamente para permitir a execução dos modelos no *smartphone*, diante da circunstância em que a execução nativa de modelos no formato PyTorch (.pt) não se mostra direta.

O treinamento dos modelos foi realizado por meio dos *scripts* providos pelo repositório original da YOLOv5 Jocher (2020), que possui código aberto. O *framework* Pytorch foi utilizado no processo de treinamento individual dos modelos CD e CR, salvos ao final do processo no formato Pytorch (.pt) utilizando representação em FP16 (*Float Point 16 bits*). Como definido em Tensorflow (2023), TensforFlow Lite foi utilizado no processo conversão, produzindo modelos em .tflite também em FP16. Após a conversão, os modelos tornaram-se levemente menores devido às otimizações do *TFLite Converter*. Para o modelo de detecção de placa (CD) a redução foi de 0.2MB e para o modelo de reconhecimento de caracteres (CR) a redução foi de 0.1MB. A conversão foi feita utilizando um utilitário fornecido no repositório da YOLOv5, que permite a transformação dos modelos em PyTorch para diversos formatos como tflite, onnx, openvino e coreml. Após a finalização do processo de conversão, os modelos CD e CR apresentam tamanhos de 13.5 MB e 13.6 MB, respectivamente.

A Figura 3.6 representa o processo de conversão dos modelos CD e CR originalmente representados em FP16 no PyTorch para FP16 no TFLite visando possibilitar a execução em dispositivos móveis.

Figura 3.6: Processo de conversão dos modelos de Pytorch para TFLite



Fonte: Autor

3.3 Aplicação Android

3.3.1 Tecnologias Utilizadas

O desenvolvimento da aplicação Android utilizou como referência a implementação nativa da inferência do modelo YOLOv5s para detecção de objetos de *zldrobit* Fang (2021), devido à necessidade da implementação de métodos utilizados no pós-processamento da YOLOv5 que não podem ser portados para o TFLite, como o NMS (do inglês, *Non Maximum Supression*). Android Studio e Java foram utilizados para o desenvolvimento da aplicação Android, utilizada para a interação com o usuário. As dependências necessárias para o funcionamento da aplicação são:

- **TensorFlow Lite 2.4.0** de Tensorflow (2021) foi utilizado para carregamento dos modelos exportados em .tflite, bem como acesso à execução em CPU e GPU.
- **Android-Image-Cropper 4.5.0** de Canato (2023) foi utilizado para a escolha da região de interesse da imagem/video em análise.
- **API Android Legacy Camera** foi utilizada para aquisição dos *frames* provenientes da inferência em video utilizando a câmera do *smartphone* em tempo real.

3.3.2 Funcionalidades

As funcionalidades da aplicação visam oferecer diversas maneiras de operar o sistema proposto, possibilitando sua utilização sem a exigência de conhecimento prévio sobre a execução interna do sistema ALPR integrado a aplicação Android. A seguir, são apresentadas as principais funcionalidades implementadas:

1. Permitir o reconhecimento da placa em fotos da galeria do smartphone.
2. Viabilizar o reconhecimento da placa em fotos adquiridas direto da câmera nativa do smartphone, sem a necessidade de sair da aplicação.

3. Permitir o reconhecimento da placa em tempo real utilizando a aquisição continua de *frames*, executando o sistema em cada novo *frame* provido pela câmera do Smartphone.

4. Selecionar o dispositivo de execução de cada modelo, CPU ou GPU.

5. Escolher o número de *threads* utilizados para inferência (apenas em CPU).

6. Limitar a quantidade de *frames* na inferência em video.

7. Possibilitar a escolha da região do veículo, selecionando uma região quadrada na imagem ou video, permitindo translação e ampliação ou redução da região.

3.3.3 Telas da aplicação

A aplicação desenvolvida possui 4 telas principais apresentadas nas Figuras 3.7, 3.8, 3.9 e 3.10 relativas às configurações de execução do sistema ALPR e escolha do dado de entrada.

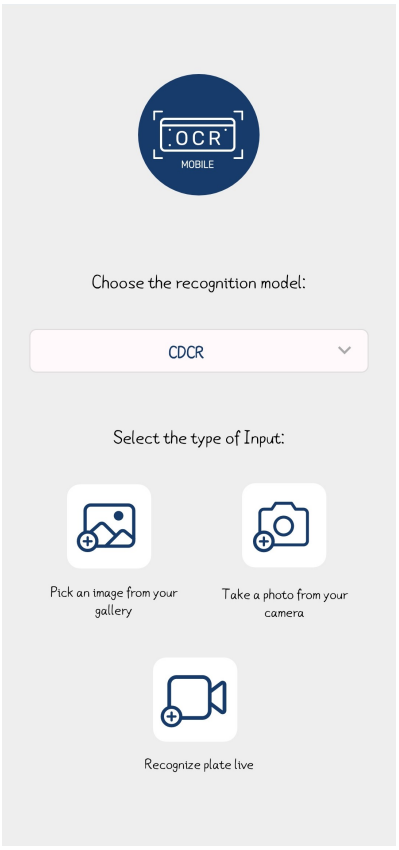


Figura 3.7: Tela de Escolha

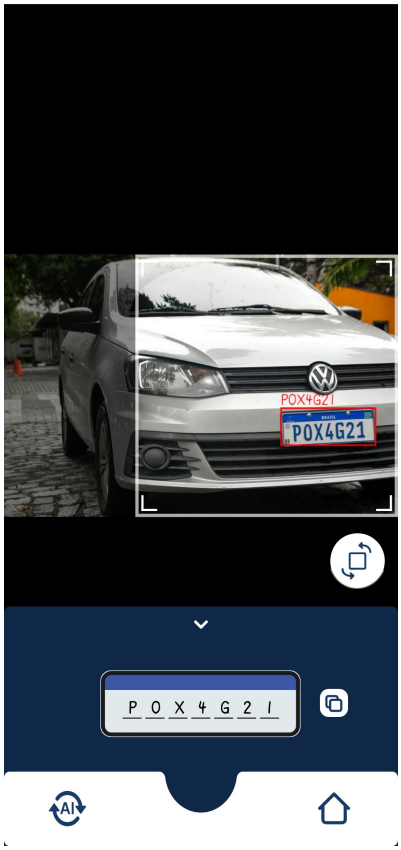


Figura 3.8: Tela de execução estática

A tela da Figura 3.7 é destinada à escolha da estratégia de execução do sistema de ALPR, com as opções CDCR e CRFULL. A estratégia de execução CRFULL é experimental e utiliza apenas um modelo YOLOv5s para efetuar o reconhecimento da placa

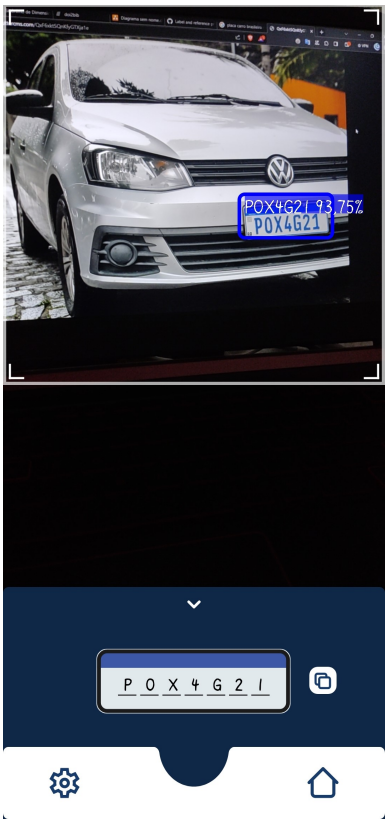


Figura 3.9: Tela de execução dinâmica

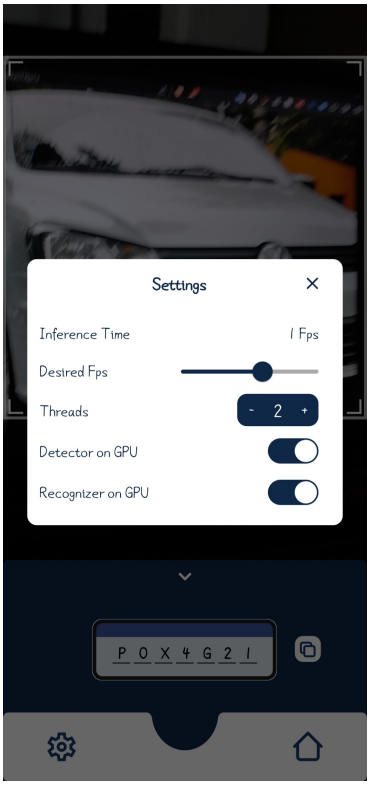


Figura 3.10: Tela de Configuração na execução dinâmica

diretamente na imagem original, sem a necessidade da fase de localização da placa. Ademais, é utilizada para seleção do formato de entrada dos dados, com as opções de carregamento de uma imagem da galeria, captura de uma imagem utilizando a câmera ou execução do sistema em tempo real através da aquisição dos *frames* continuamente. A Figura 3.8 é destinada à execução do sistema no modo estático, as opções de entrada utilizando carregamento da galeria ou foto utilizando a câmera direcionam o usuário para esta tela, a qual possibilita a escolha da região de interesse, rotação da imagem atual, cópia da placa reconhecida realização de uma nova inferência ou retorno a tela de escolha.

A seleção da região de interesse é uma funcionalidade utilizada para reduzir a região de busca da placa na execução do modelo CD, fornecendo as possibilidades de ampliação, redução e translação. Tal funcionalidade exige a escolha de uma região quadrada, o que facilita o redimensionamento da região antes da execução do modelo CD, minimizando os problemas de distorção da imagem e direcionando a região de buscado do modelo.

O botão localizado na região inferior direita da tela permite a rotação da imagem em 90° no sentido horário. Algumas marcas de dispositivos possuem convenções diferentes para a rotação da imagem na câmera nativa, tal cenário é contemplado por funcionalidade.

A Figura 3.9 é utilizada quando a opção de execução em video na tela da Figura 3.7 é selecionada, permitindo a escolha da região de interesse assim como na execução do sistema em dados estáticos. O sistema é executado para cada *frame* provido pela câmera,

atualizando a placa reconhecida de forma contínua, assim como o texto presente no *card* da placa, que pode ser copiado. A Figura 3.10 é destinada à configuração do sistema na execução dinâmica, possibilitando ao usuário a seleção do FPS (do inglês, *Frames Per Second*), número de *threads* na execução dos modelos em CPU e alocação dos modelos em GPU.

A escolha do *Desired FPS* para coleta continua de *frames* é importante devido à possibilidade do dispositivo utilizado sofrer com o processamento em alta taxa de quadros, isso pode aumentar significativamente a temperatura do aparelho. Esta opção limita a taxa de coleta dos *frames*, possibilitando redução do esforço da CPU ou GPU no processo de inferência dos modelos CD e CR. O FPS desejado pode não ser atingido devido às limitações de processamento do aparelho, assim a informação do FPS atingido no elemento *Inference Time* torna-se útil para uma escolha eficiente de *Desired FPS*.

3.3.4 Avaliação de desempenho do Sistema Proposto

Foram desenvolvidos dois *benchmarks* para avaliar o desempenho do sistema ALPR proposto em um ambiente tradicional de *desktop* e no ambiente restrito do *smartphone* Android, nomeados *Desktop Benchmark* e *Mobile Benchmark*, respectivamente.

O *Desktop Benchmark* foi desenvolvido em python com as bibliotecas yolov5-pip e opencv, facilitando o processo de carregamento, inferência dos modelos e o redimensionamento de imagens. Tal *benchmark* utiliza os conjuntos de validação e teste mencionados na seção 3.1, produzindo um gráfico contendo a acurácia de ALPR atingida pelo sistema utilizando os dois modelos, CD e CR, em TFLite e Pytorch para cada um dos conjuntos. O objetivo deste *benchmark* é verificar se a conversão entre *frameworks* impacta o desempenho do sistema em um ambiente de *desktop* tradicional.

O *Mobile Benchmark* foi implementado como uma aplicação Android, utilizando o mesmo mecanismo de execução dos modelos da aplicação original a medida que permite a escolha de uma pasta contendo todas as imagens do conjunto de teste executando o sistema ALPR proposto para cada imagem, salvando ao final do processo um arquivo *json* utilizado para computação das métricas em python, devido à facilidade. O *Mobile Benchmark* é utilizado para obter métricas como *mAP*, acurácia de ALPR, Distância de Levenshtein, consumo de memória e consumo de bateria no ambiente restrito do *smartphone*.

3.3.5 Hardware Utilizado

O dispositivo base para a realização dos *benchmarks* é o Zebra TC210K, e possui as seguintes configurações de hardware e software:

Tabela 3.1: Especificações do Smartphone

Item	**Especificação**
Capacidade da bateria	3300 mAh
Memória RAM	3GB
Memória Interna	32GB
Sistema Operacional	Android 10 Pie
Câmera Principal	13Mpx
SoC	Qualcomm Snapdragon 660 1.84GHz
Resolução da Tela	720x1280

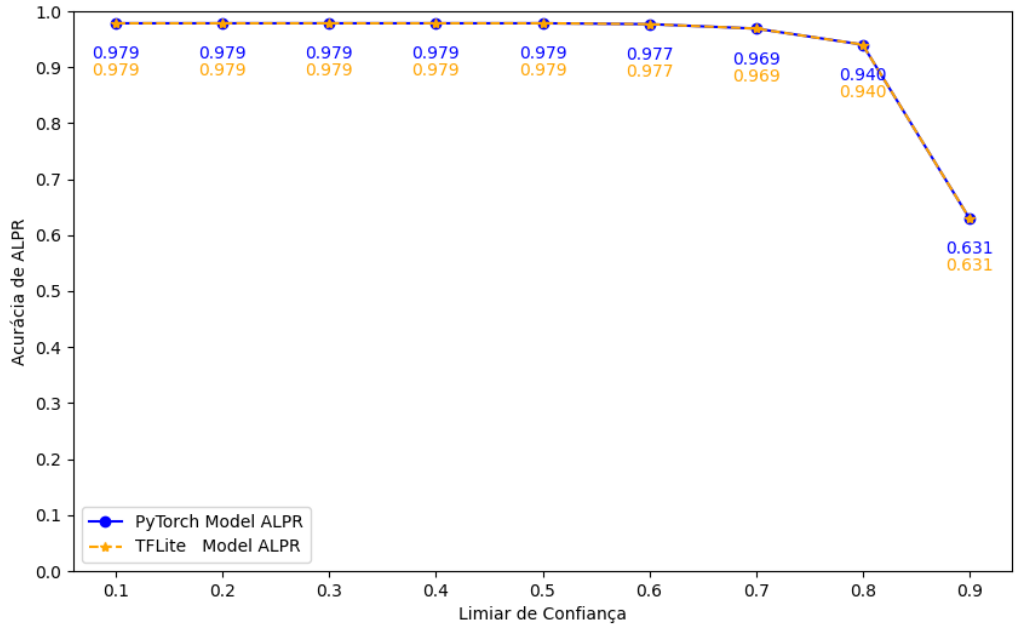
Este Smartphone é classificado como um dispositivo do segmento intermediário base, devido ao *SoC* Snapdragon 660, validando o funcionamento do sistema proposto em aparelhos com configurações relativamente comuns. Como apresentado em 3.1.2, este dispositivo foi utilizado para efetuar a captura das amostras presentes no conjunto de Teste.

Capítulo 4

Resultados e Discussões

4.1 Desktop Benchmark

Figura 4.1: Acurácia de ALPR no conjunto de Validação

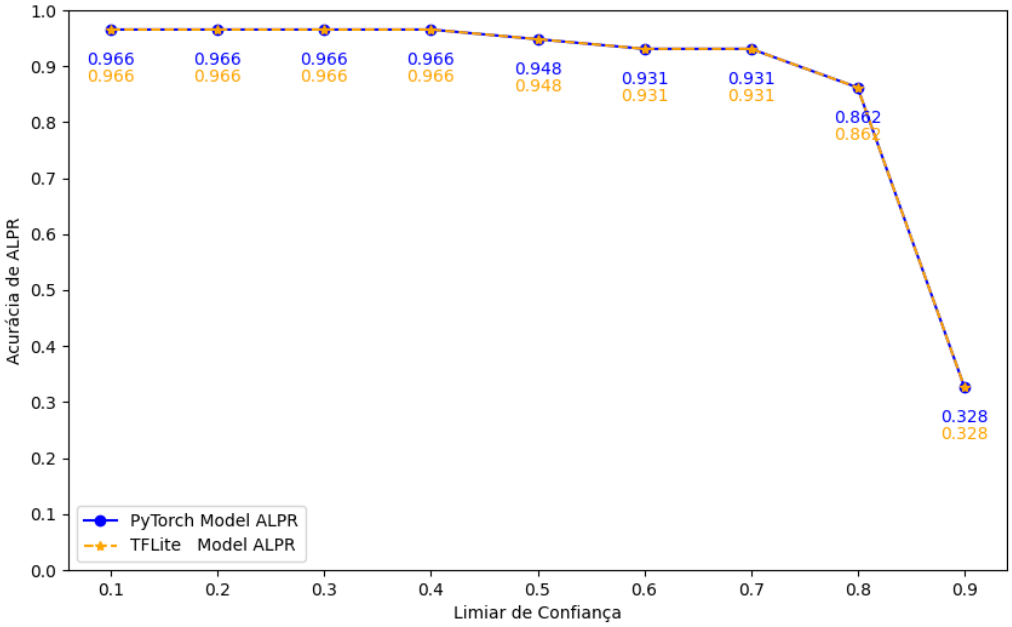


Fonte: Autor

Ao executar o *Desktop Benchmark* no conjunto de validação com 1260 imagens, obtivemos o desempenho apresentado na Figura 4.1, onde é possível verificar que mesmo após a conversão entre os *frameworks*, o sistema fundamentado nos modelos TFLite é capaz de reproduzir os resultados de acurácia de ALPR alcançados pelo sistema que utiliza os modelos PyTorch. Tais resultados são relativos à exata comparação da licença de código predita com a esperada, quando o modelo acerta todos o 7 caracteres.

A Figura 4.2 apresenta o desempenho dos modelos desenvolvidos no conjunto de Teste utilizando-se o Desktop Benchmark. Neste contexto, onde as imagens foram obtidas

Figura 4.2: Acurácia de ALPR no conjunto de Teste



Fonte: Autor

de situações cotidianas, o sistema ALPR utilizando os modelos TFLite obteve a exata acurácia de ALPR do sistema fundamentado em modelos originais, no formato Pytorch.

Ambos os gráficos representam a acurácia de reconhecimento dos 7 caracteres quando todas as caixas são detectadas com uma confiança acima do limiar indicado no eixo *X*, comparando o desempenho dos modelos original (Pytorch) e otimizado (TFLite) para execução em sistemas carentes de recursos computacionais, quando comparados com *desktops* tradicionais.

4.2 Mobile Benchmark

4.2.1 Análise de predições

O desempenho do modelo utilizando o Mobile Benchmark é restrito ao conjunto de Teste, devido às limitações de tempo de inferência para muitas amostras e armazenamento do *Smartphone* Android utilizado, que possui apenas 32GB de memória interna.

Tabela 4.1: Métricas de reconhecimento de caracteres do sistema ALPR

Acurácia no Desktop	**Acurácia no Mobile**	**Distância Levenshtein**
96.6%	89.66%	1

A Tabela 4.1 indica um decréscimo de 7% na acurácia do sistema proposto rodando no *Smartphone* em comparação com a execução dos modelos em um *Desktop*. O Gráfico 4.2 sugere que tal diminuição na acurácia seja causada pela execução no ambiente mobile, uma vez que os modelos em TFLite performam semelhantemente aos modelos PyTorch no *Desktop Benchmark*. Uma possível explicação para tal decréscimo seria a otimização das operações suportadas pelo Interpretador do TFLite no smartphone, que podem diferir das implementações presentes nos dispositivos convencionais como *desktops*. Tal suposição pode ser sustentada pelo fato de que os modelos TFLite integrados na aplicação também foram utilizados no *Desktop Benchmark*, onde apresentaram a mesma acurácia de ALPR dos modelos originais, em Pytorch.

Tabela 4.2: Erros de reconhecimento no conjunto de Teste

Placa Verdadeira	**Placa Predita**
XXHXXXX	XXUXXXX
XXIXXXX	XXTXXXX
QXXXXXX	OXXXXXX
OXXXXXX	DXXXXXX

A Tabela 4.2 e a última coluna da Tabela 4.1 fornecem uma visão dos erros cometidos pelo sistema proposto no reconhecimento da licença de código final. Os caracteres reconhecidos corretamente foram substituídos por X para preservação de informações do veículo e proprietário.

Figura 4.3: Imagens com placas reconhecidas incorretamente

Fonte: Autor

As imagens relativas às predições incorretas são apresentadas na Figura 4.3, adicionando-se uma máscara retangular para preservação de informações de modo que apenas o caractere reconhecido incorretamente seja apresentado. A possível causa de tais inconsistências nas predições pode ser explicada pelo ângulo de captura das imagens, onde a região da placa ficou limitada a parte inferior, além da inclinação existente que distorce os caracteres, facilitando erros no reconhecimento de caracteres parecidos, como Q e O, O e D presentes nas duas últimas imagens.

O valor de 1 para a Distância de Levenshtein, indica que o sistema proposto falhou em reconhecer corretamente em média um caractere presente na região da placa, o que pode ser tolerável em certas aplicações, pois conhecer 5 ou mais caracteres de uma licença de código reduz o intervalo de possibilidades para aquisição da placa correta com 7 caracteres.

4.2.2 Localização das Caixas

Tabela 4.3: mAP do Sistema ALPR proposto

Model	**mAP@0.5:0.95**	**mAP@0.5**
CD	0.62	0.85
CR	0.24	0.77

A Tabela 4.3 apresenta o desempenho do sistema com relação à localização das caixas das placas e caracteres. Para a etapa de detecção da placa, 0.85 indica uma boa precisão de localização das caixas de cada placa quando o limiar de IoU é 0.5. Na etapa de reconhecimento de caracteres obtemos um valor de 0.62, indicando que mesmo diante da dependência da precisão do CD na detecção da região da placa, as caixas dos caracteres foram localizadas com precisão acima da média. Ao focar a análise em um intervalo de limiares IoU, temos uma diminuição dos valores, o que é esperado acontecer, pois pequenas variações na localização das caixas preditas afetam bastante a localização para limiares próximos de 1. O impacto é maior na etapa de Reconhecimento de caracteres, pois as caixas são pequenas e qualquer variação afeta consideravelmente a localização da predição.

4.2.3 Tempos de Inferência

O tempo necessário para executar o pipeline completo do sistema pode ser um aspecto crítico a depender da utilização da aplicação desenvolvida, tornando necessária a possibilidade de executar o sistema em dispositivos dedicados, como *Mobile GPU*.

Tabela 4.4: Tempos de Inferência do sistema proposto

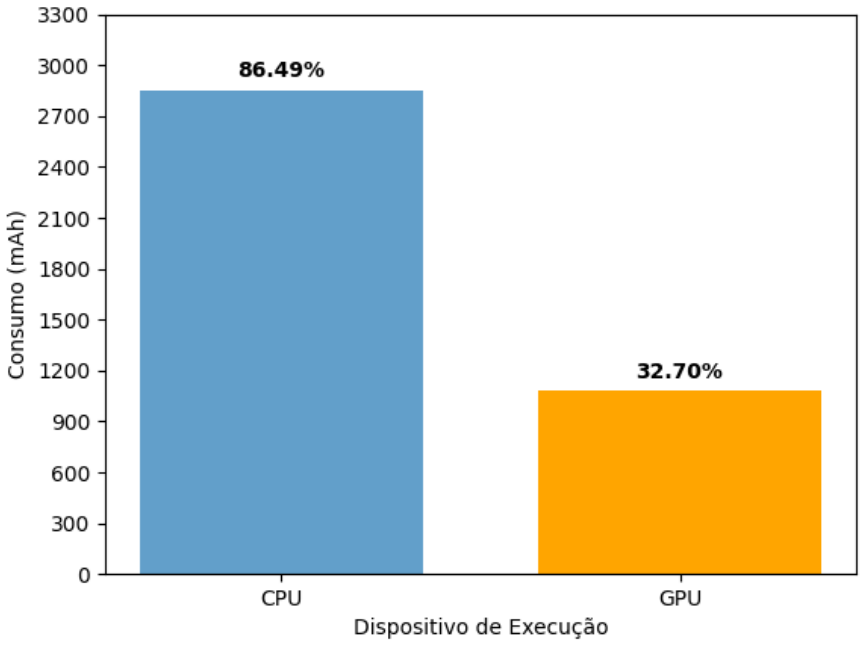
Dipositivo	**CD (ms)**	**CR (ms)**	**Sistema Completo (ms)**
CPU	1325.88 ± 250.74	353.79 ± 67.04	1679.67 ± 317.83
GPU	239.12 ± 45.90	80.09 ± 15.19	320.02 ± 61.09

A Tabela 4.4 demonstra que a execução em *Mobile GPU* fornece um ganho de aproximadamente 5.2x em comparação com a execução em CPU. Contudo, é aconselhável analisar os requisitos do contexto em que a aplicação será empregada, uma vez que execução em GPU pode ocasionar problemas de aquecimento quando persistida por um longo tempo, em contrapartida, a utilização da CPU para inferência diminui a frequência de tais problemas, mas aumenta o tempo de resposta do pipeline de inferência.

4.2.4 Consumo de Recursos

A execução de modelos de Aprendizado profundo em dispositivos Android utiliza recursos de forma intensiva para produzir a saída de forma rápida, o consumo de bateria e memória transmitem informações pertinentes para a análise da execução de modelos em dispositivos com recursos limitados.

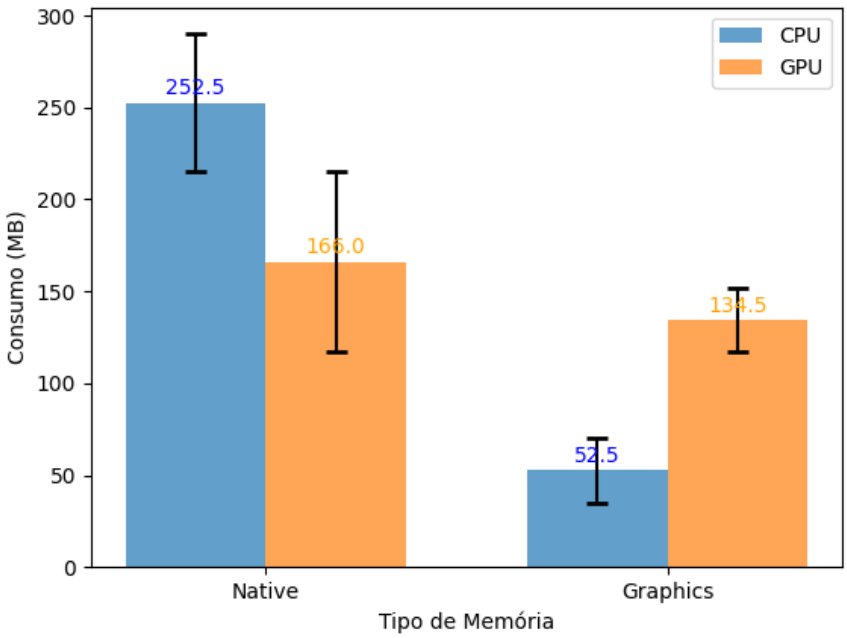
Figura 4.4: Consumo de Bateria quando ambos os modelos são executados na CPU ou GPU.



Fonte: Autor

O consumo de bateria da aplicação desenvolvida em execução no *smartphone* Zebra TC210K com 100% de bateria inicial é apresentado na Figura 4.4 . Tal dispositivo possui a capacidade de 3300mAh de bateria, o gráfico mostra o consumo deste recurso mediante a 10000 execuções do sistema utilizando a estratégia CDCR. Para este experimento foram escolhidas aleatoriamente 50 imagens do conjunto de teste do Benchmark Mobile, executando-se os modelos CD e CR por 200 iterações, totalizando 10000 inferências. As porcentagens de consumo foram obtidas tomando-se a diferença da capacidade da bateria em mAh nos instantes inicial e final da realização do teste. A execução em GPU mostra-se mais eficiente energeticamente, demostrando uma diferença de consumo de aproximadamente 54% com relação à CPU, o que pode ser explicado pelas otimizações das operações matriciais implementadas na GPU, as quais são necessárias para execução dos modelos, possibilitando a inferência rápida e otimizada.

Figura 4.5: Consumo de memórias Native e Graphics quando ambos os modelos são executados na CPU ou GPU.



Fonte: Autor

o consumo de memórias *Native* e *Graphics* da aplicação desenvolvida durante a execução do sistema ALPR é apresentado na Figura 4.5. Os dados foram obtidos utilizando a ferramenta *Memory Profiler* do Android Studio, utilizado no desenvolvimento da aplicação. Os dados foram coletados durante 3 execuções do *Mobile Benchmark* utilizando 50 amostras escolhidas aleatoriamente do conjunto de teste. Esta ferramenta fornece a informação do consumo de memória da aplicação de forma gráfica e em tempo

real, possibilitando a análise do consumo de diferentes tipos de memória, como memoria de código, *Native*, *Graphics*, Java e *Stack*. Os tipos *Native* e *Graphics* são as mais importantes no contexto da aplicação, com as seguintes definições:

- **Memória Native** representa o consumo da execução de funções nativas (em C/C++), sendo utilizada nas operações do Tensorflow Lite.
- **Memória Graphics** representa o consumo de atualizações da interface e utilização da GPU

O dispositivo de teste possui 3GB de memória, mas apenas 2.7GB estão disponíveis para utilização. O gráfico de barras apresenta os valores mínimo, médio e máximo do consumo de memória *Native* e *Graphics* quando ambos os modelos utilizam o mesmo dispositivo de execução. A alocação de ambos os modelos na CPU, produz um maior consumo de memória Nativa em comparação com a execução em GPU, atingindo máximos de 290MB e mínimos de 215MB. Neste cenário, a utilização da GPU está relacionada a atualização dos elementos da interface gráfica, produzindo um consumo máximo de 70MB de memória Gráfica. A alocação dos modelos na GPU apresenta um consumo equilibrado de ambos os tipos de memória, atingindo valores máximos de 215MB e 152MB e mínimos de 117MB, para as memórias *Native* e *Graphics*, respectivamente.

Capítulo 5

Conclusão

Este trabalho apresentou um estudo acerca da utilização de Redes Neurais e Aprendizado Profundo para o desenvolvimento de um software ALPR dedicado ao reconhecimento de placas veiculares brasileiras seguindo os formatos Antigo e Mercosul. Tais sistemas possuem grande importância na automatização da aquisição dos dados veiculares, contribuindo para a digitalização, armazenamento e consultas veiculares de forma prática e eficiente. Os resultados obtidos evidenciam o potencial do sistema, o qual alcançou 89.6% de reconhecimento das placas no conjunto de teste sem aplicação de heurísticas, baixo consumo de memória, velocidade no reconhecimento e eficiência energética quando a execução dos modelos em GPU Mobile é comparada a execução em CPU Mobile. Os resultados mostram que a conversão dos modelos para TFLite não afeta o desempenho do sistema quando executado no *desktop*, atingindo taxas de reconhecimento iguais ao sistema original baseado em modelos utilizando o *framework* Pytorch quando executado sobre as mesmas condições em um ambiente tradicional. Finalmente, a aplicação desenvolvida permite fácil usabilidade, sem conhecimento prévio do usuário, configurações para execução do sistema e variabilidade nas formas de aquisição dos dados, visando contemplar os cenários mais comuns da aplicação de sistemas ALPR.

5.1 Trabalhos Futuros

Diante de algumas limitações na atual implementação da aplicação, possíveis aprimoramentos futuros são apresentados abaixo:

- Aplicação de heurísticas para evitar reconhecimentos incorretos decorrentes da semelhança de caracteres como O e 0, I e 1, B e 8, presentes no formato Mercosul.
- Desenvolver modelos CD com input retangular visando aproveitar-se do formato retangular caraterístico das placas.

- Buscar desenvolver o mecanismo de inferência em lotes, visando utilizar ainda mais as otimizações de paralelismo da GPU Mobile.
- Adicionar novas formato de amostras para o treinamento dos modelos, como motocicletas.
- Quantização dos modelos em int8, visando reduzir consumo de memória e aceleração na execução dos modelos.

Bibliografia

Cabral, J. P., Santos, V. G. D., Souza, C., Silva, A., Dantas, A., Cacho, N., Lopes, F., and Araujo, D. (2021). An efficient CNN-based approach for automatic brazilian license plate recognition. In *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE.

Canato (2023). Android image cropper. `https://github.com/CanHub/Android-Image-Cropper`.

F., A. G. S., , F, M. M., A, H. R., N, M., Z, M., F, Z. M., and S., M. A. S. M. A. (2021). A live-video automatic number plate recognition (ANPR) system using convolutional neural network (CNN) with data labelling on an android smartphone. *International Journal of Emerging Technology and Advanced Engineering*, 11(10):88–95.

Fang, J. (2021). yolov5. `https://github.com/zldrobit/yolov5`.

Godage, L. H. and Wimalaratne, G. (2019). Real-time mobile vehicle license plates recognition in sri lankan conditions. In *2019 14th Conference on Industrial and Information Systems (ICIIS)*. IEEE.

Huang, P. and Wang, W. (2022). Research and design of automatic license plate recognition system based on android platform. In *2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC)*. IEEE.

Ignatov, A., Timofte, R., Kulik, A., Yang, S., Wang, K., Baum, F., Wu, M., Xu, L., and Van Gool, L. (2019). Ai benchmark: All about deep learning on smartphones in 2019. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE.

Izidio, D. M. F., Ferreira, A. P. A., and Barros, E. N. S. (2018). An embedded automatic license plate recognition system using deep learning. In *2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC)*. IEEE.

Jocher, G. (2020). Yolov5 by ultralytics.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Montazzolli, S. and Jung, C. (2017). Real-time brazilian license plate detection and recognition using deep convolutional neural networks. In *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. IEEE.

opencv (2023). Cvat. `https://github.com/opencv/cvat`.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

Riaz, W., Azeem, A., Chenqiang, G., Yuxi, Z., Saifullah, and Khalid, W. (2020). YOLO based recognition method for automatic license plate recognition. In *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications(AEECA)*. IEEE.

Santos, D., Claro, D. B., and Gondim, J. (2023). Monitoring vehicle plate detection in brazilian universities. In *Proceedings of the XIX Brazilian Symposium on Information Systems*. ACM.

Shashirangana, J., Padmasiri, H., Meedeniya, D., and Perera, C. (2021). Automated license plate recognition: A survey on methods and techniques. *IEEE Access*, 9:11203–11225.

Tensorflow (2021). Tensorflow lite maven repository. `https://mvnrepository.com/artifact/org.tensorflow/tensorflow-lite-gpu/2.4.0`.

Tensorflow (2023). Tensorflow lite: For mobile and edge. `https://www.tensorflow.org/lite/guide`.

Wada, K. (2022). Labelme. `https://github.com/wkentaro/labelme`.

Zou, Z., Chen, K., Shi, Z., Guo, Y., and Ye, J. (2023). Object detection in 20 years: A survey. *Proceedings of the IEEE*, 111(3):257–276.