
UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

# AtalaIA: Aperfeiçoamento de sistema para reconhecimento automático de placas de licença automotiva e construção de um produto para segurança de áreas restritas

Trabalho de Conclusão de Curso

Cristiano Lima Oliveira


São Cristóvão – Sergipe

2023

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Cristiano Lima Oliveira

# AtalaIA: Aperfeiçoamento de sistema para reconhecimento automático de placas de licença automotiva e construção de um produto para segurança de áreas restritas

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 de Computação.

Orientador(a): Prof. Dr. Leonardo Nogueira Matos
Coorientador(a): M.Sc Flávio Santos e M.Sc Rafael Andrade da Silva

São Cristóvão – Sergipe

2023

*Dedico este trabalho de conclusão de curso com imenso carinho e saudade a minha tia Rosana, que infelizmente não está mais entre nós. Sua presença em minha vida foi um exemplo de força, determinação e amor incondicional.*

*Tia sempre foi uma das minhas principais fontes de inspiração, incentivando-me a buscar o conhecimento e perseguir meus sonhos. Seu apoio e encorajamento foram fundamentais para que eu pudesse trilhar este caminho acadêmico.*

*Embora não esteja fisicamente presente, sinto sua presença espiritual ao meu lado a cada passo na vida. Sei que a senhora está olhando por mim, guiando-me e orgulhando-se de minhas conquistas.*

*Este trabalho é dedicado a senhora, tia. Agradeço por tudo que a senhora fez por mim, pelos valores que transmitiu e pelos momentos preciosos que compartilhamos juntos.*

*Pelo jeito da senhora quando eu alcançava alguma conquista, tenho certeza que o céu está em festa ao ver este trabalho concluído. Saiba que sua influência e amor continuam a me inspirar a cada dia. Sentirei sua falta, mas sempre lembrarei com gratidão e amor tudo que a senhora representou em minha vida.*

*Este trabalho é dedicado a senhora, tia, com profunda gratidão e amor eterno.*

# Agradecimentos

Primeiramente quero agradecer ao meu orientador, Prof. Dr. Leonardo Nogueira Matos, e também aos envolvidos no grupo de pesquisa Ludiico, por confiarem a mim este tema desafiador. Sua experiência, conhecimento e dedicação juntamente com os coorientadores M.Sc Flávio Santos e M.Sc Rafael Andrade da Silva, foram essenciais para moldar meu trabalho, refinar minha metodologia e oferecer ideias preciosas ao longo do processo.

Gostaria também de expressar meu agradecimento ao Prof. Dr. Rafael Oliveira Vasconcelos por avaliar este trabalho e fornecer valiosos comentários e sugestões durante a defesa. Suas contribuições foram fundamentais para o aprimoramento deste estudo.

Agradeço aos professores do curso, que transmitiram seus conhecimentos de maneira inspiradora e incentivadora. Suas aulas e orientações contribuíram significativamente para a minha formação acadêmica.

Gostaria de expressar minha gratidão aos amigos e familiares que estiveram ao meu lado durante todo o processo. O apoio, encorajamento e compreensão foram essenciais para minha motivação e bem-estar emocional ao longo deste caminho desafiador.

Em especial, quero agradecer à minha mãe, Cristiane, que enfrentou dificuldades como mãe solteira, mas sempre lutou para que eu recebesse a melhor educação. Ela sempre acreditou no meu potencial e foi um exemplo de fé e perseverança, mostrando-me que posso alcançar grandes conquistas na vida.

Também agradeço à minha irmã, Cláudia, por ser a primeira referência acadêmica na família e por fortalecer em mim a ideia de que posso alcançar meus objetivos, desde que seja consistente e paciente, transformando as dificuldades em força para não desistir, permitindo-me trilhar meu caminho com mais experiência.

No que diz respeito à minha área de atuação profissional, sou grato a Givaldo, que atuando com *hardware* e sistemas operacionais foi o primeiro exemplo que tive na área de computação. Apesar de trabalhar profissionalmente como mecânico em uma oficina de motocicletas, ele sempre demonstrou habilidades em computação. Observando-o resolver os problemas que surgiam no computador da minha casa quando eu era criança percebi que para desenvolver-me na área bastava ter foco e força de vontade.

A todos vocês, meus sinceros agradecimentos. Cada um de vocês desempenhou um papel fundamental neste trabalho de conclusão de curso e na minha vida. Sou grato pelas oportunidades, pelo apoio e pelas lições aprendidas ao longo desta jornada. Este trabalho é o resultado de uma contribuição coletiva e estou honrado por tê-lo compartilhado com pessoas tão especiais.

*Somos o que repetidamente fazemos.*
*A excelência, portanto, não é*
*um ato, mas um hábito.*
*(Aristóteles)*

# Resumo

Considerando os recentes avanços tecnológicos em visão computacional, faz-se necessário que os produtos e serviços de segurança utilizem esse conhecimento. Com o uso do aprendizado profundo, é possível desenvolver ferramentas inteligentes que podem operar em cenários como a aplicação de leis de trânsito por meio de sistemas de reconhecimento de placas de veículos, reconhecimento facial para autorização de acesso a áreas restritas e identificação de comportamento criminoso por meio da detecção de atitudes suspeitas. Os problemas de segurança em portarias podem decorrer de erros humanos no processo de tomada de decisão em relação à concessão de acesso a indivíduos ou veículos, portanto, a assistência tecnológica é útil para a prevenção de erros. Com isso em mente, este trabalho visa aprimorar um sistema de Reconhecimento Automático de Placas de Veículos (ALPR) desenvolvido com modelos de Redes Neurais Convolucionais (CNNs) para aumentar a segurança em portarias de estabelecimentos e condomínios. Para alcançar esse aprimoramento, foi desenvolvido no sistema um módulo de rastreio de objetos que aproveita dados do processamento de imagens anteriores no reconhecimento de uma nova imagem de um mesmo veículo, possibilitando redução na latência de processamento e permitindo desenvolver outras funcionalidades, como o tratamento de redundância temporal, que eleva a acurácia dos resultados do sistema de ALPR, bem como a funcionalidade de contagem de veículos. Além disso, um protótipo de câmera IP foi construído usando uma NodeMCU Esp32Cam, visando oferecer uma opção de câmera IP de baixo custo. Foram desenvolvidas interfaces no sistema para a comunicação via MQTT e RTSP entre câmeras e unidades de processamento, permitindo que o sistema de ALPR seja usado com várias câmeras disponíveis no mercado compatíveis com o protocolo RTSP, assim como com o protótipo de câmera IP desenvolvido através do protocolo MQTT. Além disso, utilizaram-se métodos de compressão em um modelo YOLOv5 Nano, como poda e quantização. Como resultado, foi possível aprimorar o sistema de ALPR aumentando a acurácia dos resultados de 27,18% para 56,05% por meio do tratamento de redundância temporal, e houve uma redução de 33% na latência do processamento com a implementação do módulo de rastreio.

**Palavras-chave**: YOLOv5. rastreio. câmera. compressão. configuração. ALPR.

# Abstract

Given the recent advancements in computer vision technology, it is necessary for security products and services to utilize this knowledge. By using deep learning, it is possible to develop intelligent tools that can operate in scenarios such as the enforcement of traffic laws through vehicle license plate recognition systems, facial recognition for access authorization to restricted areas, and identification of criminal behavior through the detection of suspicious actions. Security issues at entrances can arise from human errors in the decision-making process regarding granting access to individuals or vehicles, thus technological assistance is useful for error prevention. With this in mind, this work aims to enhance a system of Automatic License Plate Recognition (ALPR) developed using Convolutional Neural Network (CNN) models to increase security at establishment and condominium entrances. To achieve this enhancement, an object tracking module was developed in the system, leveraging data from previous image processing to recognize a new image of the same vehicle. This allows for a reduction in processing latency and enables the development of additional functionalities, such as temporal redundancy treatment, which enhances the accuracy of the ALPR system's results, as well as vehicle counting functionality. Furthermore, a prototype IP camera was constructed using a NodeMCU Esp32Cam, aiming to provide a low-cost IP camera option. Interfaces were developed in the system for MQTT and RTSP communication between cameras and processing units, allowing the ALPR system to be used with various cameras available on the market compatible with the RTSP protocol, as well as the IP camera prototype developed using the MQTT protocol. Additionally, compression methods were applied to a YOLOv5 Nano model, such as pruning and quantization. As a result, it was possible to enhance the ALPR system by increasing the accuracy of the results from 27.18% to 56.05% through temporal redundancy treatment, and there was a 33% reduction in processing latency with the implementation of the tracking module.

**Keywords**: YOLOv5. tracking. compression. setup. ALPR.

# Lista de ilustrações

# Lista de quadros

# Lista de tabelas

# Lista de códigos

# Lista de algoritmos

# Lista de abreviaturas e siglas

| | |
|---|---|
| CNN | Redes Neurais Convolucionais ou *Convolutional Neural Network* |
| ALPR | Reconhecimento Automático de Placas de Veículos ou *Automatic License Plate Recognition* |
| ANPR | Reconhecimento Automático de Placas Numéricas ou *Automatic Number Plate Recognition* |
| CFTV | Circuito Fechado de Televisão |
| DVR | Gravador de Vídeo Digital ou *Digital Video Recorder* |
| LED | Diodo Emissor de Luz ou *Light-Emitting Diode* |
| PIR | Sensor de Infravermelho Passivo ou *Passive Infrared Sensor* |
| NAS | Armazenamento em Rede ou *Network-Attached Storage* |
| FTP | Protocolo de Transferência de Ficheiros ou *File Transfer Protocol* |
| RTSP | Protocolo de Transmissão em Tempo Real ou *Real-Time Streaming Protocol* |
| RTP | Protocolo de Transporte em tempo Real ou *Real-time Transport Protocol* |
| IP | Protocolo Internet ou *Internet Protocol* |
| API | Interface de Programação de Aplicação ou *Application Programming Interface* |
| MQTT | Transporte de Filas de Mensagem de Telemetria ou *Message Queuing Telemetry Transport* |
| M2M | Máquina para Máquina ou *Machine to Machine* |
| OSI | Interconexão de Sistemas Abertos ou *Open System Interconnection* |
| HTTP | Protocolo de Transferência de Hipertexto ou *Hypertext Transfer Protocol* |
| URI | Identificador Uniforme de Recurso ou *Uniform Resource Identifier* |
| IoT | Internet das Coisas ou *Internet of Things* |
| TCP | Protocolo de Controle de Transmissão ou *Transmission Control Protocol* |

| | |
|---|---|
| OASIS | Organização para o Avanço das Normas de Informação Estruturada ou *Organization for the Advancement of Structured Information Standards* |
| UML | Linguagem de Modelagem Unificada ou *Unified Modeling Longuoge* |
| SSID | Identificador do Conjunto de Serviços ou *Service Set Identifier* |
| SQL | Linguagem de Consulta Estruturada ou *Structured Query Language* |
| SGBD | Sistema Gerenciador de Banco de Dados |
| JSX | JavaScript XML |
| IDE | Ambiente de Desenvolvimento Integrado ou *Integrated Development Environment* |
| GCC | Coleção de Compiladores GNU ou *GNU Compiler Collection* |
| MCU | Unidade de Microcontrolador ou *Microcontroller Unit* |
| YOLO | Você Só Olha Uma Vez ou *You Only Look Once* |
| ROI | Região de Interesse ou *Region of Interest* |
| SoC | Sistema em Chip ou *System On Chip* |
| IOU | Interseção sobre União ou *Intersection Over Union* |

# Sumário

# 1 Introdução

Apesar de toda a evolução tecnológica acerca da segurança privada, como a utilização de portaria remota e câmeras com sensores, ainda há ocorrências de invasões simplistas em locais restritos como condomínios. Casos assim não são raros de acontecer, podendo ser encontrado em buscas pela internet notícias sobre infratores se passarem por moradores em condomínios de luxo para realizar furtos em residências (G1, 2022).

Este caso exemplifica situações por vezes originadas por falha humana, as quais podem ser causadas devido ao mau treinamento da equipe de segurança, a falta de atenção no momento da tentativa de invasão, ou até mesmo a falta de recursos tecnológicos no ambiente de trabalho. Em qualquer caso, o resultado se dá por alguma pessoa mal intencionada conseguindo o acesso almejado ao ambiente restrito. Surge então a necessidade de criação e utilização de ferramentas que possam minimizar ocorrências como as apresentadas.

E para isso, as tecnologias modernas possibilitam desenvolver soluções que atuam de maneira automática e contínua em cenários de segurança como recursos complementares. Um exemplo disso é o circuito fechado de televisão (CFTV), voltado para a segurança tanto privada como pública, sendo considerado por estudiosos como um “bem banal” (PIZA et al., 2019).

Este, no geral, é formado pela instalação de câmeras de vigilância dentre outros dispositivos conectados a uma central na qual pode consistir de monitores que apresentam as imagens capturadas por estas câmeras, como também informações de dispositivos complementares como sensores. No ambiente de monitoramento também pode dispor de dispositivos para armazenamento das imagens e a presença de profissionais para acompanhar as informações recebidas em tempo real.

Contudo, apesar desse recurso ajudar a fortalecer a segurança e comprovadamente diminuir a realização de ações de infratores em diversos cenários, como estacionamentos, centros urbanos e áreas residenciais (PIZA et al., 2019), ainda há espaço para a ocorrência de falhas de segurança, como a permissão de acesso em um condomínio dada a veículo roubado (INFONET,

2018). Um provável motivo para essa brecha de segurança pode ser devido à permissão de pessoas ou veículos acessarem um ambiente ser em alguns casos responsabilidade total dos porteiros, por vezes sem recursos tecnológicos para orientá-los na melhor decisão.

## 1.1 Objetivos

O objetivo geral deste trabalho é aperfeiçoar um sistema de ALPR desenvolvido por Silva (2022), principalmente através da criação de um módulo de rastreio que permite aumentar a acurácia dos resultados com o tratamento de redundância temporal e reduzir a latência ao desabilitar parcialmente parte do *pipeline* principal de detecção devido o aproveitamento de informações processadas entre imagens. Por fim, o sistema com a inclusão do módulo de rastreio é avaliado, assim como as versões comprimidas do modelo YOLOv5 Nano destinado à detecção das placas veiculares que também visa a redução de latência.

Os objetivos específicos são apresentados a seguir:

- Desenvolver um protótipo de câmera IP (*Internet Protocol*) de baixo custo com o NodeMCU Esp32Cam;
- Construir interfaces no sistema de ALPR (*Automatic License Plate Recognition*) para comunicação através dos protocolos MQTT (*Message Queuing Telemetry Transport*) e RTSP (*Real-Time Streaming Protocol*);
- Aperfeiçoar a acurácia dos resultados do sistema de ALPR através do tratamento de redundância temporal;
- Desenvolver a funcionalidade de contagem de veículos para análise de fluxo em portarias;
- Realizar testes das diferentes configurações do produto em contexto de uso real e análise dos resultados obtidos.

## 1.2 Metodologia

A metodologia que rege este trabalho é particionada em três etapas, sendo elas:

1. Construção e avaliação de configurações: Essa etapa envolve a construção física e lógica de uma câmera IP de baixo custo através do uso da placa de desenvolvimento Esp32Cam com bibliotecas disponíveis que possibilitam a adição de recursos como uma interface web para estabelecer uma conexão entre a placa e uma rede *wireless*, permitir a comunicação da placa com o sistema de ALPR através do uso do protocolo MQTT, uma página web para permitir configuração de comunicação e imagem, e um sistema de arquivos para permitir que as configurações sejam salvas permanentemente na memória *flash* da placa. Já para

possibilitar a integração de câmeras IPs com o sistema também há o desenvolvimento de interfaces no sistema de ALPR com a linguagem de programação Python para recebimento de imagens através da comunicação com protocolos MQTT ou RTSP. Com isso, foram realizados testes práticos com o sistema de ALPR executando em um servidor e também em um Jetson Nano para avaliação da taxa de quadros (*framerate*) e da acurácia dos resultados coletados;

2. Construção e avaliação do módulo de rastreio: Nessa etapa, com a parte física e comunicativa já encaminhadas, foi desenvolvido um módulo de rastreio visando unir os modelos de detecção de objetos com uma lógica de expansão da caixa delimitadora retornada pelo detector de placas para permitir o rastreio de placas veiculares no sistema de ALPR, reduzindo a latência de inferência por desabilitar parcialmente parte do *pipeline* principal do sistema e aumentando a acurácia dos resultados através do tratamento de redundância temporal. Sendo avaliado o desempenho através do processamento de cinco vídeos gravados na portaria de veículos da Universidade Federal de Sergipe (UFS) identificando o ganho de acurácia, redução de latência e desempenho do contador de veículos;

3. Compressão do modelo YOLOv5 Nano e avaliação de desempenho: Por fim, a última etapa envolve a compressão do modelo YOLOv5 Nano treinado para detectar a placa na imagem recortada pós-processamento do detector de veículos. Para isso foram utilizadas ferramentas desenvolvidas pela Neural Magic[1] buscando a redução da latência tanto na execução completa do *pipeline* quanto na execução parcial que ocorre quando o módulo de rastreio está habilitado. Para medir o desempenho foram realizadas inferências em CPU através do ambiente do Kaggle[2] em 1530 imagens rotuladas da base de dados UFS-ALPR através dos modelos densos YOLOv5 Nano e YOLOv8 Nano, além de quatro modelos comprimidos variando entre 30% e 40% de poda com ou sem quantização dos pesos. A partir dos resultados das inferências foi possível comparar os modelos considerando as métricas precisão, *recall*, mIOU, acurácia da classificação do tipo e do padrão da placa, e a taxa de quadros;

## 1.3 Estrutura do Documento

O documento está estruturado em seis capítulos, os quais são descritos a seguir com exceção do capítulo introdutório:

- **Capítulo 2 - Fundamentação Teórica**: Visa apresentar toda a fundamentação teórica para entendimento e elaboração do trabalho;

[1] Disponível em: <https://neuralmagic.com/>. Acesso em abril de 2023
[2] Disponível em: <https://www.kaggle.com/>. Acesso em abril de 2023

- **Capítulo 3 - Trabalhos Relacionados**: Apresenta os produtos similares ao desenvolvido neste trabalho com base em pesquisas de mercado;
- **Capítulo 4 - Desenvolvimento**: Tem por objetivo apresentar o desenvolvimento das configurações de *hardware* que utilizam a câmera construída com o NodeMCU Esp32Cam, como também o desenvolvimento do módulo de rastreio integrado ao sistema de ALPR e a compressão de um modelo YOLOv5 Nano. Além disso, apresenta ferramentas e tecnologias vinculadas ao trabalho;
- **Capítulo 5 - Resultados Alcançados**: São apresentados os resultados com base nos testes realizados com as configurações propostas durante o trabalho. Além disso, apresenta também os resultados gerados pela implantação do módulo de rastreio e a compressão do modelo YOLOv5 Nano;
- **Capítulo 6 - Considerações Finais**: É feita uma análise geral sobre o trabalho realizado, e os trabalhos futuros para aprimoramento do projeto.

# 2

# Fundamentação Teórica

Neste Capítulo, será apresentada a fundamentação teórica para o entendimento acerca dos conceitos e ferramentas utilizadas para o desenvolvimento deste trabalho.

Na Seção 2.1, é tratado o tema redes neurais convolucionais a partir de um contexto histórico, as métricas principais para avaliação de desempenho dos modelos treinados, algumas aplicações das redes neurais convolucionais na área de visão computacional, a arquitetura YOLO, alguns conceitos sobre algoritmos de rastreio e sobre a transferência de aprendizado, e estratégias para compressão de modelos. Na Seção 2.2, são apresentados o conceito e os desafios sobre a base principal deste trabalho, sendo o uso do aprendizado profundo (*deep learning*) para o reconhecimento automático de placas veiculares. Já nas Seções 2.3 e 2.4 são descritos o MQTT e o RTSP, os quais são os principais protocolos de comunicação utilizados no sistema.

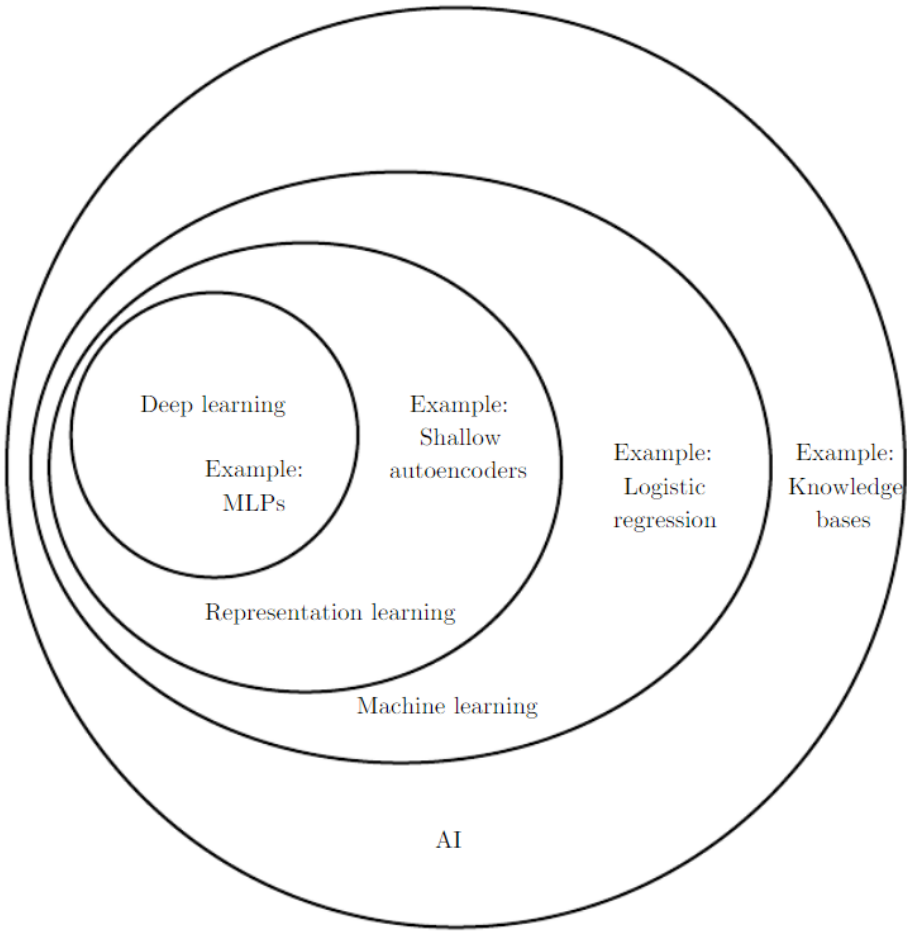
## 2.1 Redes Neurais Convolucionais (CNNs)

Para dar contexto às CNNs tudo começa partindo do termo "inteligência artificial", o qual é generalizado pela população até mesmo para referenciar subáreas como a de aprendizado de máquina (*machine learning*) e aprendizado profundo, sendo ilustrada a divisão destas subáreas através da Figura 1. Justamente por serem subáreas têm-se que nem toda inteligência artificial se refere a uma solução de aprendizado de máquina (ORACLE, 2023). E quanto à definição, a inteligência artificial é *esforço para automatizar tarefas intelectuais executadas por seres humanos* (CHOLLET, 2017).

Originalmente os princípios básicos da inteligência artificial (IA) surgiram na década de 50 através dos trabalhos realizados por Alan Turing, que já desempenhava papel fundamental na computação desde 1936 quando fez a descrição de uma máquina de computação abstrata que mais tarde seria intitulada como máquina de Turing. Esta consiste em três partes, a primeira trata-se de uma **fita** usada como dispositivo de entrada, saída e memória de trabalho. Também

dispõe de uma **unidade de controle** que indica o estado corrente da máquina e possui uma unidade de leitura e escrita que acessa uma célula da fita de cada vez, movimentando-se para esquerda ou direita. Já a última parte se trata da **função de transição** que comanda as leituras, escritas e movimentações ao longo da fita, além de definir o estado da máquina (MENEZES, 2009, pp. 133–134). Sendo esta visão de uma máquina abstrata computacional a base para toda a computação moderna.

Figura 1 – Inteligência artificial e suas subáreas



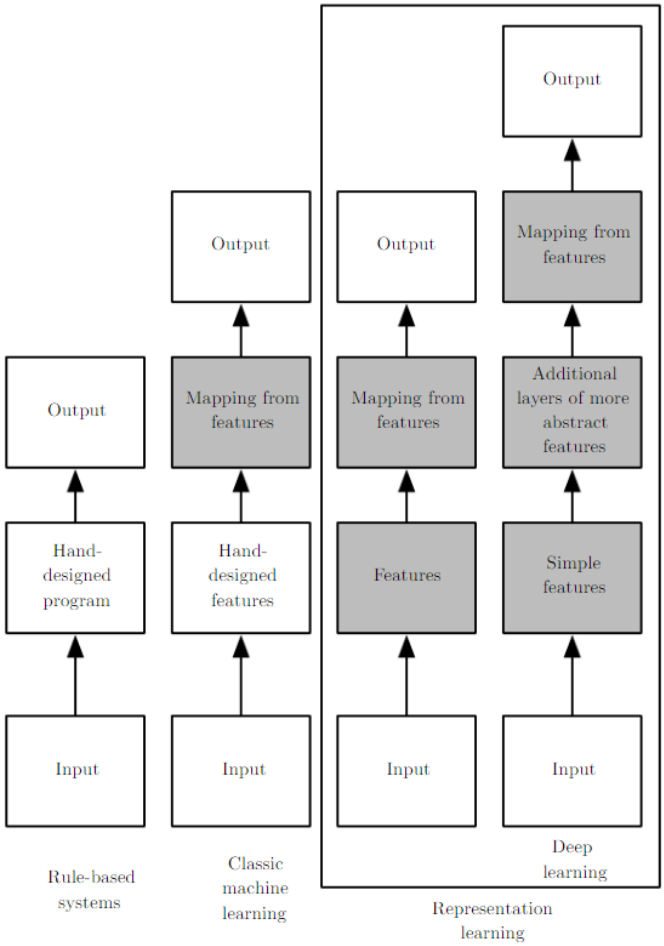

Fonte: Goodfellow, Bengio e Courville (2016)

As primeiras ideias clássicas sobre IA surgiram na década de 50 quando Alan Turing propôs o **teste de Turing**, o qual aborda o conceito de inteligência em máquinas. Neste teste existem duas entidades, um interrogador e o computador, sendo o desafio principal o interrogador decidir se está conversando com um humano ou um computador após propor algumas perguntas por escrito. Então, para o computador conseguir passar por este teste tem que dispor da capacidade de realizar **processamento de linguagem natural** para permitir uma comunicação por meio de um idioma natural, **representação de conhecimento** para armazenar as informações que sabe ou "ouve" e **raciocínio automatizado** para usar as informações que sabe para responder às perguntas

do interrogador e chegar a novas conclusões. Além disso, deve-se utilizar o **aprendizado de máquina** para ser possível detectar e extrapolar padrões se adaptando a novas circunstâncias. Vale ressaltar que existe o **teste de Turing total** que também trabalha com as disciplinas de **visão computacional** e **robótica**, possibilitando o interrogador checar a capacidade de percepção visual do indivíduo do qual deseja descobrir se é um computador, e também dispor da possibilidade de passar objetos físicos por meio de uma janela (RUSSELL et al., 2010, pp. 4–7)

Dado o conceito sobre inteligência artificial e o quão amplo este termo é, uma das características desejadas para esta é que consiga aprender sem a necessidade de estabelecer de antemão um conjunto de regras. Então surge o termo aprendizado de máquina, sendo uma família de técnicas de desenvolvimento de modelos de inteligência computacional capazes de fazer estes adquirirem o conhecimento das regras a partir de informações recebidas como entrada, sejam por amostras individuais ou mediante uma base de dados.

Figura 2 – Diferença entre algoritmos clássicos de IA, algoritmos de aprendizado de máquina e algoritmos de aprendizado profundo




Fonte: Goodfellow, Bengio e Courville (2016, p. 10)

Na Figura 2, é ilustrada a diferença de como é adquirido o conhecimento entre os

algoritmos clássicos de IA em subáreas diferentes. Entre estes estão algoritmos baseados na definição manual de regras, os clássicos de aprendizado de máquina dos quais as características dos dados devem ser manualmente definidas por um supervisor antes de realizar o treinamento do modelo, e por último o aprendizado profundo que tem as características determinadas através do próprio modelo.

Vale ressaltar que não são todos os algoritmos de aprendizado de máquina que necessitam que as amostras utilizadas para treinamento sejam rotuladas, sendo algo comum quando é desejado treinar um modelo, por exemplo, por meio da arquitetura YOLO. Para deixar isso mais claro, esse conjunto de técnicas de aprendizado de máquina é dividido principalmente em três categorias (RUSSELL et al., 2010, pp. 605–607), sendo elas:

- **Aprendizado Supervisionado**: A abordagem utilizada dispõe de um conjunto de dados e um supervisor que retrata a ação de um humano, o qual define rótulos (ou classes) para os dados com base nas características destes. Exemplo disso é uma aplicação de um banco considerando que pode ser desejado que a partir da idade, histórico de dívidas, renda e sexo de um cliente seja determinado o risco de inadimplência caso seja liberado crédito, sendo os riscos determinados pelos rótulos *baixo*, *médio* e *alto*. Então, uma base de dados contendo amostras constituídas pelas características dos clientes do banco tem os riscos rotulados por meio de um supervisor, e então através do treinamento com esta base de dados rotulada o modelo encontra padrões que serão as regras utilizadas para conseguir classificar novos clientes com as devidas características processadas pelo modelo, prevendo o risco de inadimplência.

- **Aprendizado Não-Supervisionado**: Neste caso não há um supervisor atuando para classificar manualmente as amostras da base de dados. Em vez disso, o modelo recebe as características como entrada sem rótulos associados e realiza o agrupamento destes dados baseando-se nas correlações identificadas entre eles.

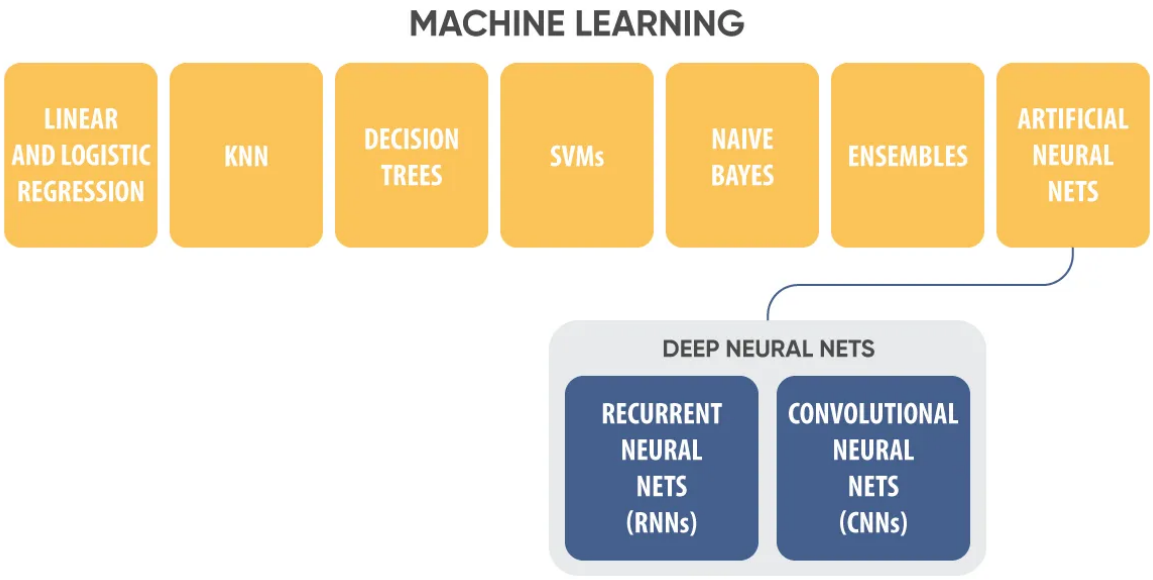
- **Aprendizado por Reforço**: Nesta o aprendizado ocorre por meio de recompensas, pois não há uma base de dados para o modelo identificar padrões a partir de amostras conhecidas. Então, o modelo adquire conhecimento interagindo com o ambiente, recebendo recompensas caso se aproxime do objetivo ou sendo punido caso se afaste.

Estas são as três categorias principais da área de aprendizado de máquina, e estão contidos em cada uma delas algoritmos utilizados para obtenção de modelos inteligentes. Dentre estes algoritmos o de principal interesse neste trabalho é o de redes neurais artificiais, e mais especificamente os de redes neurais convolucionais (*Convolutional Neural Networks*, CNNs).

A forma de treinamento das CNNs acontece por meio do aprendizado supervisionado, e como pode ser observado na Figura 3 estão contidas nas redes neurais artificiais, diferindo-se dos algoritmos clássicos de redes neurais artificiais principalmente pelo uso da operação de

convolução em pelo menos uma de suas camadas em vez da multiplicação geral de matrizes (GOODFELLOW; BENGIO; COURVILLE, 2016, p. 326). Então, antes de entender as CNNs é preciso conhecer as redes neurais artificiais compostas por neurônios artificiais que imitam o sistema nervoso humano no processo de aprendizagem, sendo na história uma das primeiras definições sobre neurônio artificial dada em 1943 por Warren McCulloch and Walter Pitts, sendo o modelo proposto nomeado como *MP Neuron* (*McCulloch-Pitts Neuron*) (MCCULLOCH; PITTS, 1943). Este modelo trabalha com valores lógicos tanto na entrada como na saída do neurônio, utilizado principalmente para problemas de classificação. Em 1957 este modelo foi ajustado e nomeado como *Perceptron* de uma camada (ROSENBLATT, 1958).

Figura 3 – Subáreas do aprendizado de máquina e das redes neurais artificiais



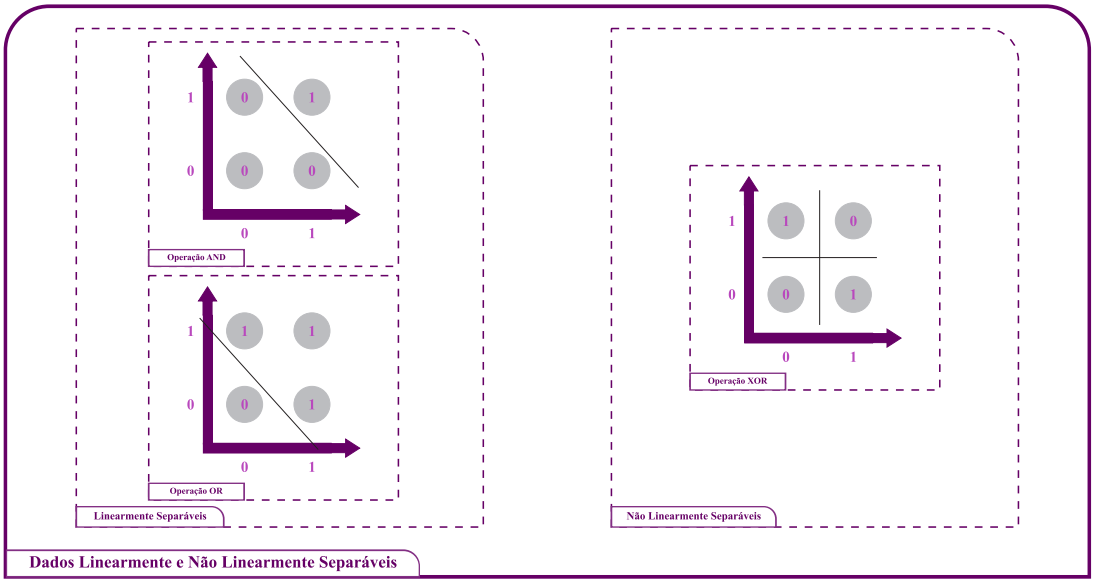

Fonte: Bobriakov (2019)

Os modelos são estruturalmente semelhantes, surgindo as diferenças devido o *Perceptron* associar valores de pesos aos dados de entrada no neurônio, além de ser possível trabalhar com qualquer dado de valor real, ao contrário do *MP Neuron* que trabalha apenas com valores lógicos, o que o torna menos flexível. Mas considerando que o *MP Neuron* trabalhava em uma implementação de um circuito lógico, convém os valores "tudo ou nada". Porém, nos dois modelos os dados de entrada devem ser linearmente separáveis como ilustrado pela Figura 4, além de permitirem manipulação de um limitante (*threshold*), sendo este o parâmetro do neurônio que indica o limiar para ativação.

Na Figura 5, é apresentada uma ilustração do *Perceptron* de uma camada. Cada entrada tem obrigatoriamente um peso associado que indica a intensidade que aquele sinal deve chegar no neurônio, sendo estes valores de pesos considerados pela **função soma**. No caso das redes

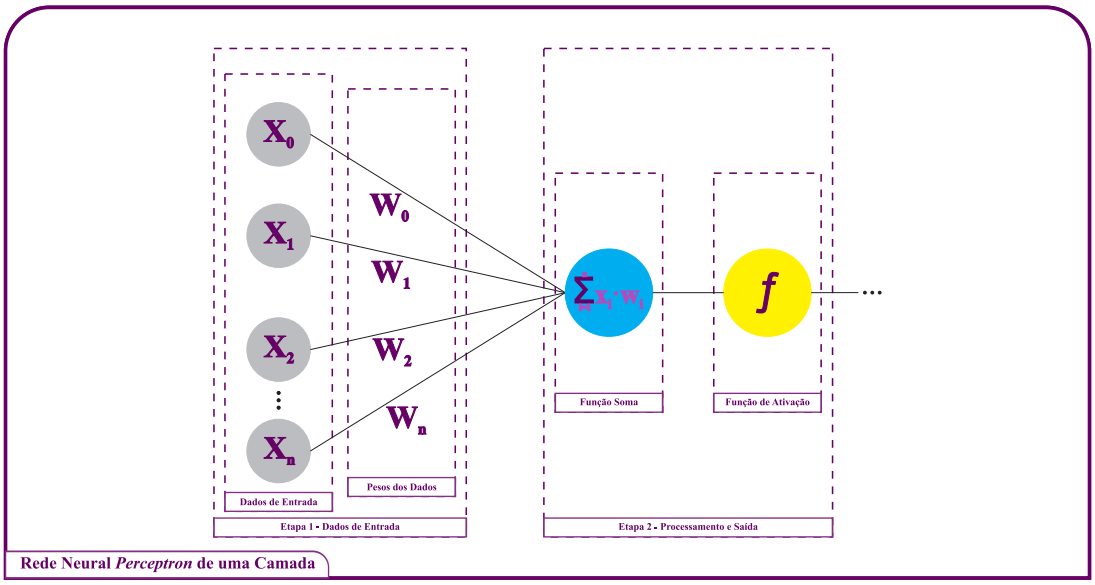
neurais, os pesos das conexões são os parâmetros que vão dar o conhecimento ao modelo, pois os padrões serão identificados a partir do caminho percorrido mediante o(s) neurônio(s) ao receber dados de entrada. Sendo assim, durante o treinamento são ajustados os valores dos pesos para tornar capaz a identificação dos padrões desejados.

Figura 4 – Tipos de separação dos dados




Fonte: Autor.

Figura 5 – Estrutura de um neurônio artificial *Perceptron*



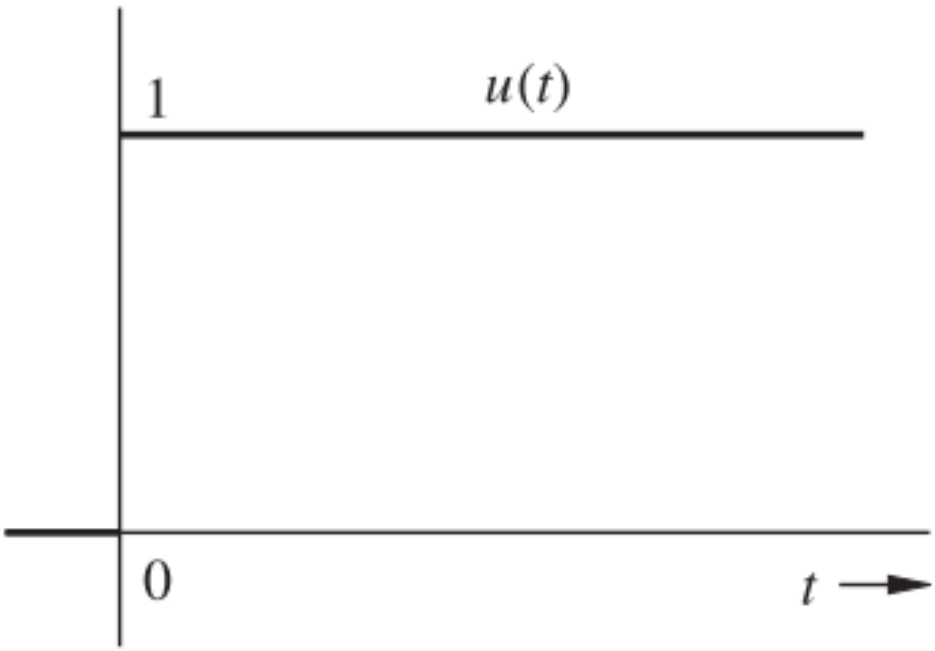

Fonte: Autor.

Na **função soma** é feita a operação de soma ponderada entre os valores de entrada e os pesos, obtendo-se um valor utilizado para decidir a intensidade do sinal de saída daquele neurônio pela **função de ativação**. No caso do *MP Neuron* a saída da **função de ativação** apresenta um valor lógico que é o resultado da soma ponderada aplicado em uma função degrau (Figura 6) responsável por determinar a ativação do neurônio. Já quanto ao *Perceptron* podem ser utilizadas outras funções como a sigmóide que retorna um valor real entre -1 e 1. No entanto, originalmente a função de ativação utilizada no modelo *Perceptron* também é a função degrau, sendo o limite de ativação definido pela translação da função no eixo dos valores de entrada da função.

Como dito anteriormente, o *Perceptron* de uma camada não pode trabalhar com dados não linearmente separáveis, surge então o *Perceptron* multicamadas (*MultiLayer Perceptron*, MLP), que em vez de ter duas etapas como apresentado na Figura 5, tem três, por conter uma etapa com pelo menos uma camada oculta (*hidden layer*), como apresentado na Figura 7.

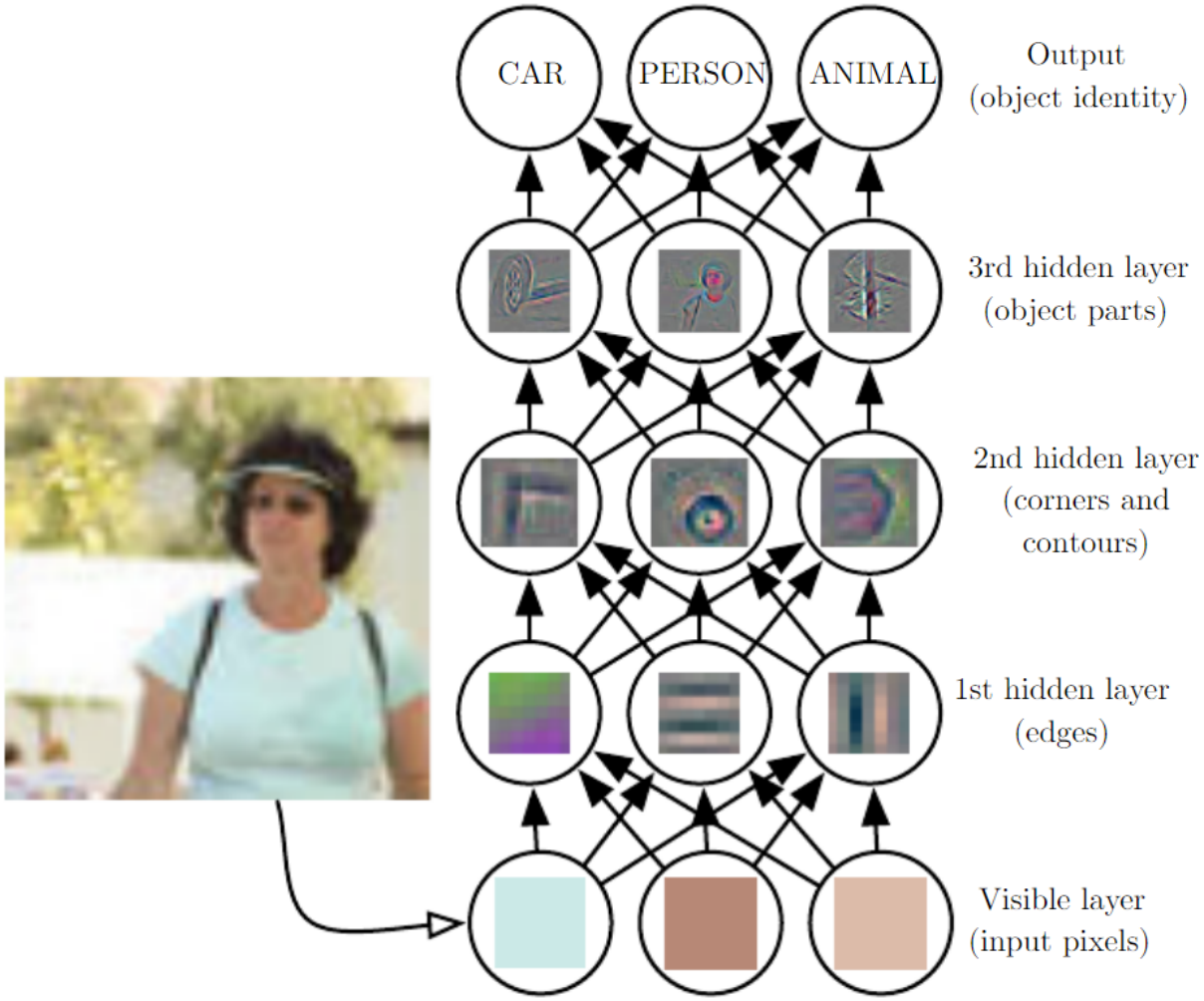
Com a inclusão de camadas ocultas no modelo chega-se na subárea de aprendizado profundo, onde as redes neurais contêm camadas ocultas formadas por vários neurônios conectados seguindo uma ideia similar a estruturada em um *Perceptron* de uma camada. Essas camadas são importantes para identificar padrões mais abstratos como pode ser observado na Figura 7, sendo responsável pela maior parte do processamento interno da rede neural (SILVA et al., 2017).

Figura 6 – Função degrau




Fonte: Lathi (2007, p. 91)

Figura 7 – Ilustração de aplicação com camadas ocultas

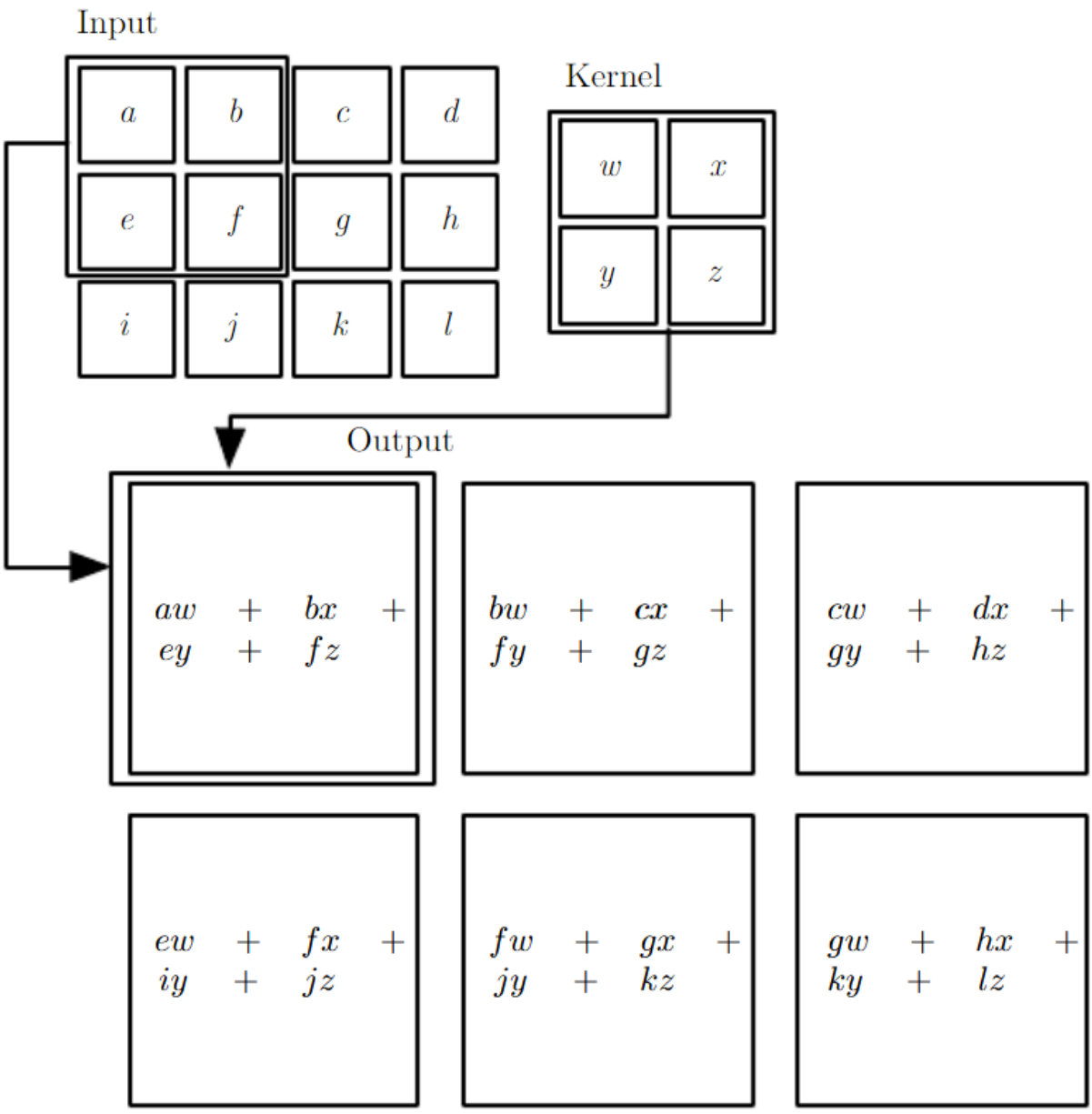



Fonte: Goodfellow, Bengio e Courville (2016, p. 6)

A necessidade de utilizar redes neurais convolucionais surge a partir do momento em que é necessário trabalhar com dados de entrada de tamanhos variados e o modelo deve ocupar menos memória. Isso acontece porque sem utilizar a operação linear de convolução cada pixel da imagem é individualmente processado pelos neurônios, então considerando uma imagem com 1920 pixels de largura por 1080 pixels de altura seriam considerados 2.073.600 pixels no processamento.

Vale ressaltar que em imagens coloridas existem três canais de cores, sendo assim a imagem é formada por uma matriz 1920x1080 para o canal vermelho, uma para o canal verde e uma para o canal azul, onde considerando um mesmo pixel é atribuída uma intensidade de cor em cada matriz formando através da combinação a cor desejada na visualização da imagem. Então, o montante de dados a serem processados passa a ser de 6.220.800 pixels, o que levaria a várias operações de ponto flutuante durante o processamento da imagem pela rede neural.

Figura 8 – Operação de convolução aplicada a uma matriz 2D




Fonte: Goodfellow, Bengio e Courville (2016, p. 330)

Assumindo que a matriz de entrada (*Input*) (ver Figura 8) representa a matriz de um dos canais de cores de uma imagem, têm-se ilustrado na Figura 8 o processo de extração de características feito pelas redes neurais convolucionais, sendo possível observar como é realizada a operação de convolução por toda a matriz de entrada com outra matriz chamada de núcleo (*Kernel*).

Similar ao que foi dito ao apresentar as redes neurais artificiais que para cada característica do dado de entrada há um peso associado, neste caso das redes neurais convolucionais os pesos estão localizados nos núcleos e são geralmente inicializados com valores aleatórios. Apesar

destes núcleos serem aplicados em várias regiões diferentes da matriz de entrada, os pesos têm valores fixos enquanto o núcleo está se deslocando, mas passam por ajustes durante o treinamento do modelo.

Em vez de ser realizada a soma ponderada de todos os pesos multiplicados com os respectivos valores das características como acontece em uma rede neural não convolucional, no caso do uso da convolução é feita a operação por partes, gerando então uma matriz de características com dimensão menor do que a matriz de entrada. Nesta matriz reduzida cujo nome é mapa de características (*feature map*) em cada célula estará o resultado da operação de convolução entre o núcleo e alguma das regiões da matriz de entrada. Com essa estratégia há um aumento de desempenho do modelo tanto em velocidade por lidar com menor quantidade de parâmetros, quanto em memória.

Ainda sobre a Figura 8 o mapa de características contém dimensão 2x3 devido o núcleo ter dimensão 2x2 enquanto a imagem contém dimensão 3x4. Considerando um passo (*stride*) para o deslocamento do núcleo de uma posição da esquerda para a direita e de cima para baixo, então são realizadas três operações convolucionais enquanto o núcleo está ocupando a primeira e segunda linha, e mais três operações quando o núcleo passa a ocupar a segunda e terceira linha da matriz de entrada. Com essa estratégia, em vez da rede neural realizar um processamento considerando doze características (ou parâmetros) como entrada, são consideradas apenas seis. Vale ressaltar que ao ser desejado o treinamento de um modelo de redes neurais convolucionais são definidos o tamanho do núcleo e o passo como hiperparâmetros, os quais são parâmetros constantes e definidos antes do processo de treinamento do modelo.

### 2.1.1 Métricas de Avaliação

A necessidade de estabelecer métricas de avaliação surge quando há diversos algoritmos disponíveis para a resolução de um mesmo problema, levando ao treinamento de diferentes modelos em busca do qual apresenta melhor desempenho com base nos requisitos que determinam uma boa solução para o problema.

Com base nisso, as métricas de avaliação são necessárias em competições e desafios voltados para a área de aprendizado de máquina para avaliação de modelos desenvolvidos pelos participantes, como no *ImageNet Large Scale Visual Recognition Challenge* (RUSSAKOVSKY et al., 2015) e o *Pascal Visual Object Classes Challenge* (EVERINGHAM et al., 2015).

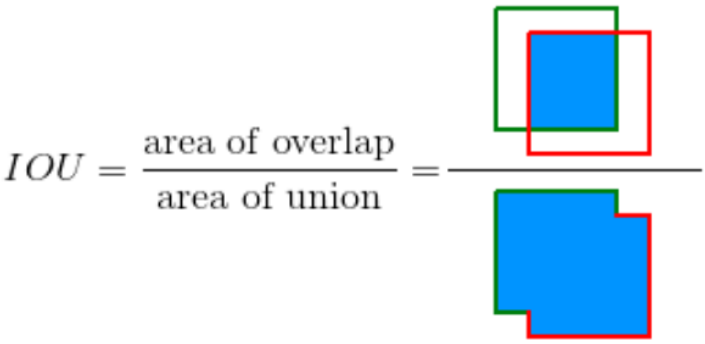
Esta Subseção tem por objetivo apresentar as métricas de avaliação IOU (*Intersection Over Union*) (ver Seção 2.1.1.1), matriz de confusão, acurácia (*accuracy*), precisão (*precision*), *recall* e $F_{1_{score}}$ (ver Seção 2.1.1.2) necessárias para determinar as métricas AP (precisão média, ou *Average Precision*) e mAP (média das médias das precisões, ou *median Average Precision*) (ver Seção 2.1.1.3) mais comumente utilizadas entre a comunidade científica e em competições (PADILLA et al., 2020).

#### 2.1.1.1 IOU

Esta métrica é fundamental para determinar se uma detecção de um objeto realizada por um modelo foi feita com sucesso ao considerar que no resultado da inferência realizada são retornadas caixas delimitadoras usadas para localizar um objeto em uma imagem, e também são retornadas as respectivas classes, cada uma associada a um dos objetos detectados.

O IOU mede a sobreposição entre a região que o objeto está cercado pela verdadeira caixa delimitadora (*ground-truth bounding box*) e a região inferida pelo modelo. Para calcular esta sobreposição é feita a operação de divisão da interseção das áreas sobre a união, como apresentado na Figura 9.

Figura 9 – Métrica Interseção Sobre União (IOU)




Fonte: Padilla et al. (2020)

Conhecido o valor IOU é possível determinar se a detecção foi realizada corretamente ao considerar um limite para aceitação. Se o valor resultante para o IOU for maior ou igual que o limite determinado, a detecção é considerada correta, consequentemente se for abaixo do limite é considerada incorreta.

#### 2.1.1.2 Matriz de Confusão, Acurácia, Precisão, *Recall* e $F_{1_{score}}$

A matriz de confusão é uma métrica utilizada em várias áreas como em processamento de linguagem natural (KAVITHA et al., 2016), acústica (MILLER; NICELY, 1955) e visão computacional (LI; LI; DENG, 2019). Esta diz respeito ao desempenho de um classificador ou detector por haver quatro situações possíveis de resultados relacionados a ela, as quais são descritas a seguir:

- **Verdadeiro Positivo (*True Positive* - TP)**: Considerando o contexto de detecção de objetos, trata-se da classe e da caixa delimitadora inferida para um objeto condizer com a classe e a verdadeira caixa delimitadora deste objeto localizado em uma imagem.

- **Verdadeiro Negativo (*True Negative* - TN)**: Ocorre quando não é detectado um objeto em uma região da imagem em que ele realmente não está. Vale ressaltar que este parâmetro não é considerado na detecção de objetos, pois como é considerada uma região da imagem para indicar a localização do objeto existem infinitas possibilidades de enquadramento em que não estará contido o objeto, sendo todas verdadeiros negativos (PADILLA et al., 2020).

- **Falso Positivo (*False Positive* - FP)**: Acontece quando é identificado um objeto que é inexistente, ou quando é identificado um objeto que não condiz com o esperado, ocorrendo principalmente por similaridades. Por exemplo, uma névoa pode ser confundida e detectada incorretamente como uma nuvem, sendo esse um grande desafio na área de aprendizado de máquina (KRSTINIć et al., 2020).

- **Falso Negativo (*False Negative* - FN)**: Ao contrário do anterior não ocorre a classificação de um objeto que deveria ter sido classificado. Sendo um exemplo disso uma imagem contendo um veículo, porém ao ser processada em um modelo treinado para detecção de veículos o resultado da inferência não apresenta a detecção deste.

Na Tabela 1, é apresentada a matriz de confusão com múltiplas classes, em cada linha está a classe verdadeira para os objetos e nas colunas estão as classes inferidas pelo modelo. Ao considerar a diagonal desta matriz é obtida para cada classe a quantidade de **verdadeiros positivos**, pois os valores destas células estão associados a contagem dos casos em que os resultados verdadeiros e os resultados da predição correspondem-se.

Tabela 1 – Matriz de confusão com múltiplas classes

| | $\lambda_1$ | $\lambda_2$ | $\lambda_3$ | $\lambda_4$ |
|---|---|---|---|---|
| $\lambda_1$ | 8 | 0 | 0 | 0 |
| $\lambda_2$ | 4 | 9 | 1 | 1 |
| $\lambda_3$ | 3 | 0 | 7 | 0 |
| $\lambda_4$ | 1 | 0 | 2 | 9 |

Fonte: Krstinić et al. (2020)

Por outro lado, os valores que estão fora da diagonal indicam quando um objeto detectado recebe uma classe que diverge da qual deveria receber. Exemplo disso é uma névoa incorretamente classificada como uma nuvem existindo para ambas classes distintas no modelo de detecção utilizado.

Com as informações coletadas através da matriz de confusão é possível definir as métricas acurácia, precisão e *recall*, possibilitando a criação de matrizes para a precisão e o *recall* representadas respectivamente na Tabela 2 e na Tabela 3.

Tabela 2 – Matriz de precisão com múltiplas classes

| | $\lambda_1$ | $\lambda_2$ | $\lambda_3$ | $\lambda_4$ |
|---|---|---|---|---|
| $\lambda_1$ | 0,5 | 0 | 0 | 0 |
| $\lambda_2$ | 0,25 | 1 | 0,1 | 0,1 |
| $\lambda_3$ | 0,19 | 0 | 0,7 | 0 |
| $\lambda_4$ | 0,06 | 0 | 0,2 | 0,9 |

Fonte: Krstinić et al. (2020)

Tabela 3 – Matriz de *recall* com múltiplas classes

| | $\lambda_1$ | $\lambda_2$ | $\lambda_3$ | $\lambda_4$ |
|---|---|---|---|---|
| $\lambda_1$ | 1 | 0 | 0 | 0 |
| $\lambda_2$ | 0,27 | 0,6 | 0,07 | 0,07 |
| $\lambda_3$ | 0,3 | 0 | 0,7 | 0 |
| $\lambda_4$ | 0,08 | 0 | 0,17 | 0,75 |

Fonte: Krstinić et al. (2020)

Quando um modelo classifica com apenas um rótulo é criada uma matriz de confusão a partir apenas de valores binários, *sim* e *não*, tanto para as linhas quanto para as colunas. Nesse cenário a acurácia do modelo é determinada pela Equação 2.1, a qual mede a taxa de assertividade das inferências do modelo através da operação de divisão do total de verdadeiros positivos e negativos sobre a quantidade total de amostras processadas $N$.

$$Accuracy = \frac{TP + TN}{N} \tag{2.1}$$

Já a precisão mede a habilidade do modelo em identificar objetos que realmente existem, e por isso na Equação 2.2 para determiná-la só é considerado o montante de verdadeiro e falso positivo.

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{All\ detections} \tag{2.2}$$

O *recall* mede a habilidade do modelo identificar corretamente o objeto dentre todas as ocorrências reais deste, sendo calculado como apresentado pela Equação 2.3.

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{All\ ground\ truths} \tag{2.3}$$

Já considerando múltiplas classes, a precisão é determinada realizando a divisão do valor de cada célula da matriz de confusão de múltiplas classes com a soma dos valores da coluna correspondente à célula em questão. Por outro lado, o *recall* é determinado realizando a divisão do valor da célula pela soma de todos os valores da linha na qual a célula está localizada.

Com a precisão e o *recall* calculados é possível definir a métrica $F_{1_{score}}$, que se trata da média harmônica (Equação 2.4) entre a precisão e o *recall* não sendo consideradas prioridades

entre uma ou outra no cálculo, sendo o resultado dado por um valor entre 0 e 1 que identifica a qualidade geral do modelo (PáDUA, 2020).

$$F_{1_{score}} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = 2 \cdot \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{2.4}$$

#### 2.1.1.3 AP e mAP

A métrica AP é definida a partir do cálculo da área abaixo da curva (*Area Under the Curve*, AUC) do gráfico formado pela precisão e o *recall*. O valor resultante do cálculo dessa área é diretamente proporcional ao valor da precisão e do *recall* simultaneamente.

Porém, deve ser considerada uma reorganização dos pontos do gráfico por geralmente apresentar em casos práticos uma curva na forma de zigue-zague, sendo um problema ao realizar o cálculo da área (PADILLA et al., 2020).

Para resolver este problema podem ser consideradas duas abordagens para reorganização dos pontos, sendo a primeira delas através da interpolação de 11 pontos calculada através da Equação 2.5.

$$\begin{aligned} AP_{11} &= \frac{1}{11} \cdot \sum_{R \epsilon \{0, 0.1, \ldots, 0.9, 1\}} P_{interp}\left(R\right) \\ onde,\ P_{interp}\left(R\right) &= \max_{\tilde{R}:\tilde{R} \geq R} P(\tilde{R}). \end{aligned} \tag{2.5}$$

Nesta abordagem a forma da curva de precisão por *recall* é resumida pelos valores de máxima precisão considerando um conjunto de valores de *recall* particionados em 11 níveis igualmente espaçados. Como pode ser observada na Equação anterior (Equação 2.5) o AP é definido a partir da precisão máxima em $P_{interp}\left(R\right)$ onde o valor do *recall* é maior que R, em vez de ser a precisão observada para cada nível de *recall* R.

Na segunda abordagem é obtido o AP interpolando a precisão em cada nível considerando a máxima precisão da qual o valor de *recall* é maior ou igual ao $R_{n+1}$, como apresentado na Equação 2.6.

$$\begin{aligned} AP_{all} &= \sum_{n} \left(R_{n+1} - R_n\right) P_{interp}\left(R_{n+1}\right) \\ onde,\ P_{interp}\left(R_{n+1}\right) &= \max_{\tilde{R}:\tilde{R} \geq R_{n+1}} P(\tilde{R}). \end{aligned} \tag{2.6}$$

Considerando o cálculo do AP para cada classe é possível determinar a métrica mAP, que consiste do resultado da operação de divisão de todos os APs calculados pela quantidade de classes $N$, como apresentado na Equação 2.7.

$$mAP = \frac{1}{N} \cdot \sum_{i=1}^{N} AP_i \tag{2.7}$$

### 2.1.2 Classificação de Imagens

A necessidade da classificação de imagens surge principalmente ao considerar que há atualmente diversos dispositivos que permitem realizar fotografias, seja através de câmeras profissionais ou com um simples celular. Com essa facilidade de realizar novos registros nos mais diversos cenários, e a possibilidade de disponibilizá-los via mídias sociais, blogs, entre outros meios, o processo de categorização dado o volume de dados disponíveis é dificultado.

Um exemplo disso pode ser um *e-commerce*, o qual sem uma aplicação para realizar a categorização das imagens pode resultar em difícil manutenibilidade da base de dados, pois ao ser desejado em algum momento realizar a filtragem das imagens por tênis esportivos não será possível exceto os produtos tenham sido manualmente classificados no processo de cadastro. Porém, ao considerar o uso de algoritmos de classificação é possível encontrar padrões entre as imagens associando-as em categorias, como ter várias imagens de chinelos e estas serem através da classificação automaticamente categorizadas como pertencentes à classe “chinelo”. Isso possibilita que usuários do *e-commerce* facilmente realizem a filtragem por “chinelo” e visualize apenas produtos relevantes, mesmo que os produtos não tenham sido classificados manualmente mediante uma estratégia supervisionada.

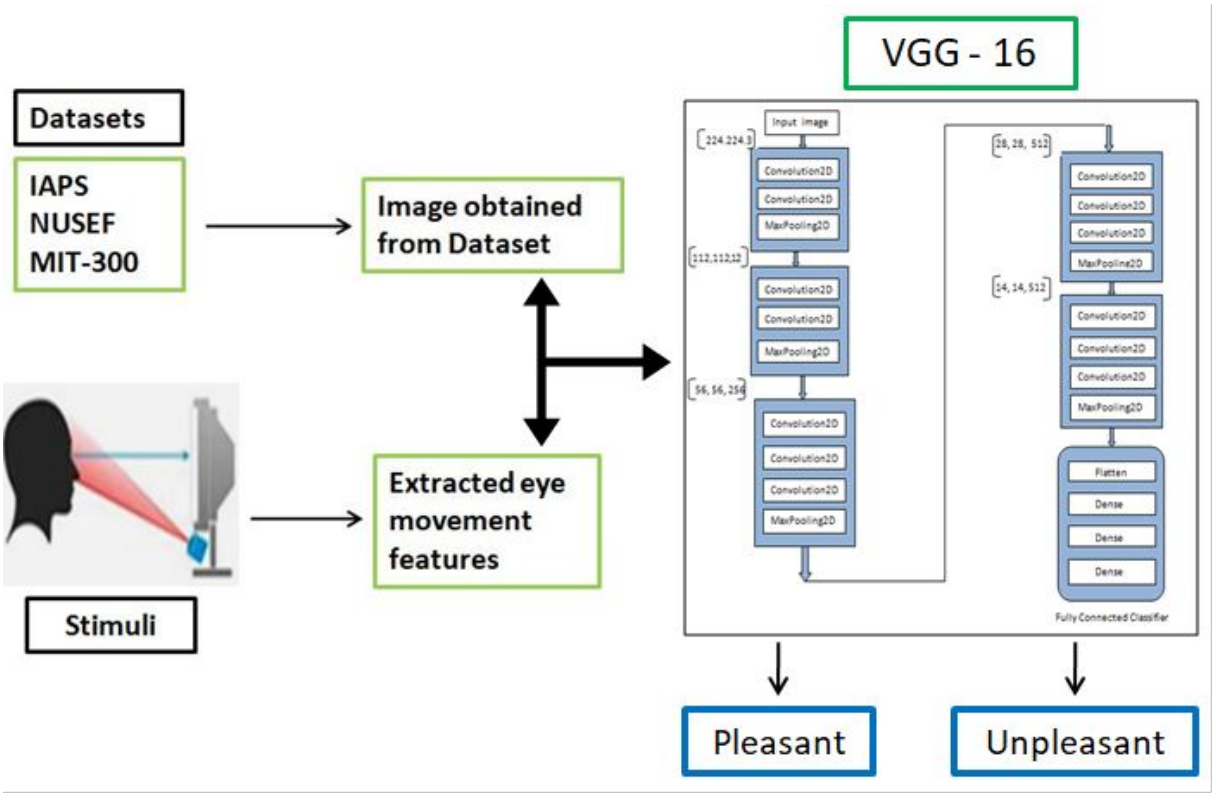
Através do uso das CNNs nesse contexto da classificação de objetos pode ser criada uma funcionalidade que não é diretamente útil para um *e-commerce*, mas é viável para uma loja física que deseja medir o grau de satisfação dos clientes. Esta funcionalidade realiza a classificação de imagens por meio de estímulos visuais, como a categorização de imagens de pessoas tristes, felizes, ou neutras (TAMULY; JYOTSNA; AMUDHA, 2019).

No geral, assim como pode ser observada na arquitetura VGG-16 utilizada em um sistema para classificação de estímulos apresentado na Figura 10, as arquiteturas com redes neurais convolucionais voltadas para problemas de classificação tem principalmente duas etapas. A primeira é formada por camadas convolucionais, ativação e agrupamento máximo (*max-pooling*) para realizar a extração de características, mapeando apenas as informações mais importantes da imagem em mapas de características.

Vale destacar sobre o funcionamento do agrupamento máximo, pois este é importante para a redução do tamanho da matriz entre camadas de neurônios na extração de características de forma parecida a como ocorre nas camadas de convolução, também por meio de uma janela deslizante. Mas este não tem parâmetros como o núcleo que contém os pesos, então em vez de realizar uma operação matemática convolucional este apenas pega o maior valor entre as células adjacentes da matriz que terá a dimensão reduzida e coloca na respectiva célula da matriz de saída com base na localização atual do janelamento feito, sendo possível observar este procedimento

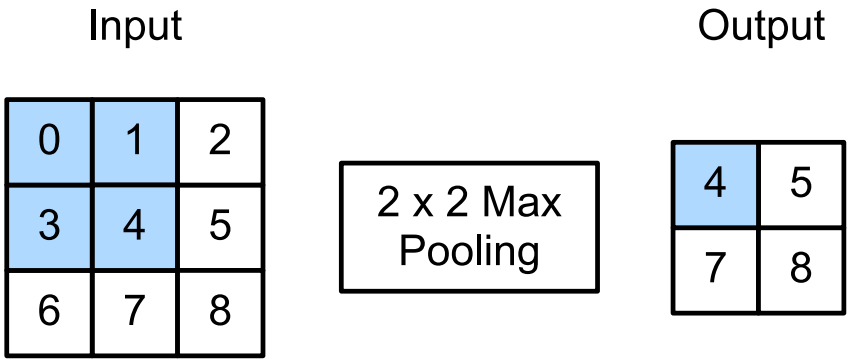
na Figura 11. Também há o agrupamento médio (*average-pooling*) que considera a média dos valores das células adjacentes no janelamento, mas o agrupamento máximo é preferível por aumentar o grau de invariância no tempo da saída (ZHANG et al., 2021, Chapter 7). Quanto ao passo definido para o procedimento, é geralmente fixo e igual à dimensão da janela de aplicação do agrupamento.

Figura 10 – Sistema de classificação de imagens através da arquitetura VGG-16 utilizando CNN




Fonte: Tamuly, Jyotsna e Amudha (2019)

Figura 11 – Aplicação do agrupamento máximo em uma matriz

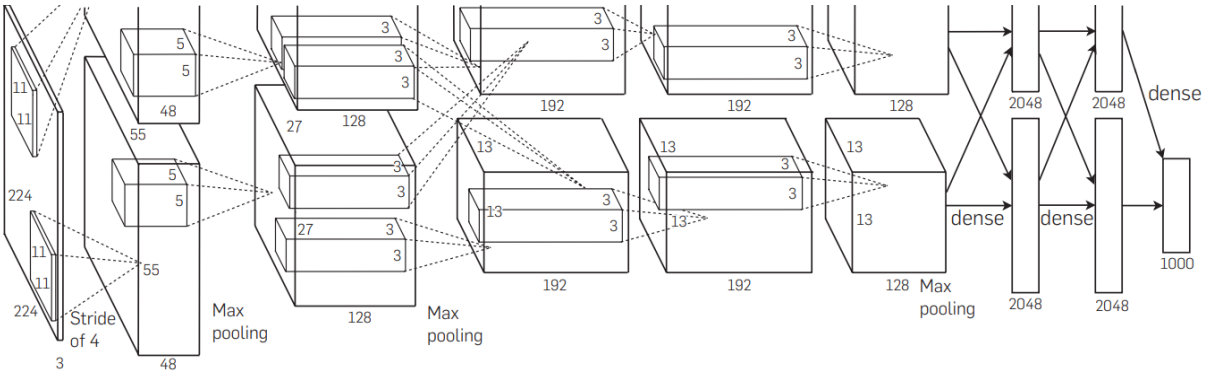



Fonte: Zhang et al. (2021, Chapter 7)

Já a segunda etapa atua a partir dos mapas de características para realizar a classificação mediante operações lineares. As camadas desta segunda etapa são chamadas de camadas densas ou completamente conectadas (*fully connected layers*).

Uma das arquiteturas mais famosas nessa tarefa de classificação através do uso da CNN é a AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2017), a qual alcançou no desafio *ImageNet Large Scale Visual Recognition Challenge* de 2012 taxas de erro top-1 de 37,5%, a qual verifica dentre a distribuição de probabilidade das classes se a de maior probabilidade condiz com a classe verdadeira, e top-5 de 17%, a qual verifica se entre as cinco classes com maiores probabilidades está incluída a classe verdadeira. Sua principal característica é a alta profundidade da rede que apesar de ser computacionalmente cara de processar, é uma solução viável por ser acelerada via GPU, alcançando alto desempenho e levando esta a ser a arquitetura vencedora da competição (RUSSAKOVSKY et al., 2013).

Figura 12 – Arquitetura AlexNet

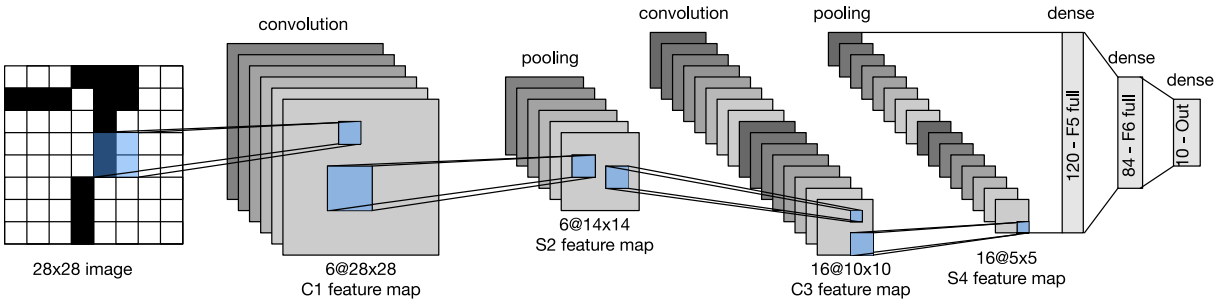



Fonte: Krizhevsky, Sutskever e Hinton (2017)

Na Figura 12, é apresentada a arquitetura AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2017), sendo possível observar que esta contém cinco camadas destinadas para extração de características, sendo as três últimas camadas densas para classificação. Na camada de saída em especial é realizada a distribuição de probabilidade entre 1000 classes através da função *softmax*.

Vale ressaltar que ela é baseada em uma das primeiras arquiteturas publicadas que utilizou CNN na sua abordagem também voltada para a área de visão computacional, que se trata da LeNet (LECUN et al., 1989). Esta consiste em duas camadas convolucionais para extração de características (convolução e agrupamento) e três camadas densas, como pode ser observado na Figura 13.

Figura 13 – Arquitetura LeNet




Fonte: Zhang et al. (2021, Chapter 7)

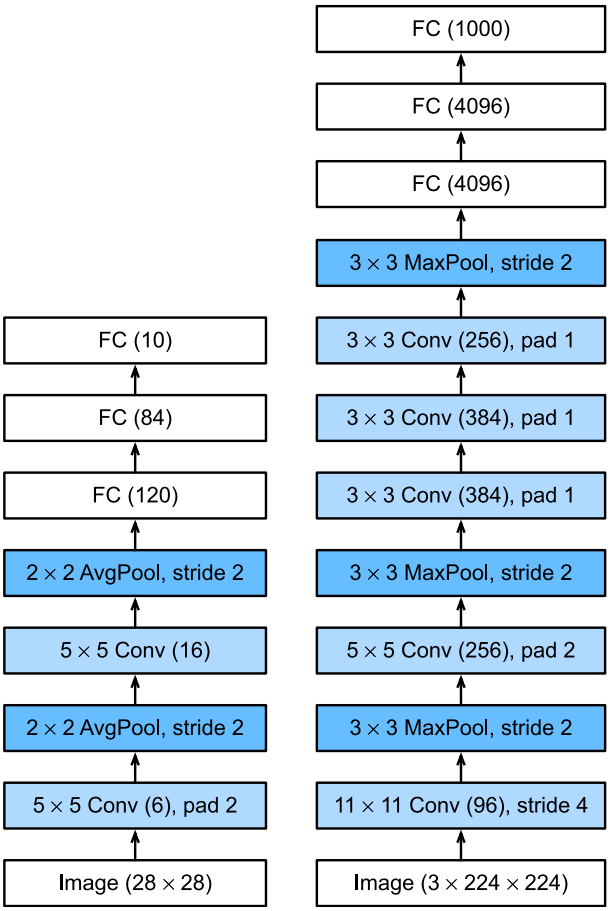
A arquitetura LeNet foi construída por Yann LeCun visando reconhecer dígitos manuscritos em imagens. Ao ser combinado com a SVM (*Support Vector Machine*) que era uma solução dominante chegou a atingir taxas de erros inferiores a 1% por dígito (ZHANG et al., 2021, Chapter 7).

Na Figura 14, é possível notar as semelhanças entre as duas arquiteturas. As diferenças consistem da profundidade da rede neural e da função de ativação, a qual na AlexNet trata-se do uso da função *ReLu*, enquanto na LeNet é a função sigmóide.

Ainda sobre a arquitetura AlexNet, esta contém 60 milhões de parâmetros treináveis, o que pode gerar um problema de sobreajuste (*overfitting*) ao ser necessário um enorme volume de dados para compensar a quantidade de parâmetros livres. Para entender o sobreajuste pode ser pensado um problema solucionado com um modelo de árvore de decisão com uma estrutura na qual a quantidade de folhas é suficiente para armazenar todos os dados utilizados para treinamento, classificando-os com perfeição. Mas apesar do modelo conseguir classificar com perfeição os dados de treinamento quando ocorre o sobreajuste, há uma baixa capacidade de generalização. Ou seja, apresenta péssimo desempenho para classificação de novas amostras que não fazem parte da base de dados de treinamento.

Sendo assim, são utilizadas estratégias para fugir do sobreajuste como o aumento da quantidade de amostras na base de dados (*data augmentation*), que considerando os dados como sendo imagens é gerado um maior volume de amostras ao construir novas imagens a partir de modificações das que já existem na base de dados, podendo ser realizadas mudanças no posicionamento do objeto diminuindo a dependência do modelo quanto à posição destes, e também ajustes de brilho e tonalidades, reduzindo a sensibilidade do modelo para cores, dentre outras estratégias (ZHANG et al., 2021, Chapter 14).

Figura 14 – Arquitetura LeNet (esq.) e AlexNet (dir.)




Fonte: Zhang et al. (2021, Chapter 8)

Além desta, também é utilizada a estratégia de abandono (*dropout*) que atua injetando ruídos durante o processamento de cada camada interna da rede neural enquanto ocorre a propagação de informações pela rede no sentido da camada de entrada para a camada de saída (*forward propagation*). Os ruídos ocorrem através do "abandono" de alguns neurônios, zerando-os antes de ser realizado o processamento da camada subsequente (ZHANG et al., 2021, Chapter 5).

Por último, foi utilizada a estratégia de redução de peso (*weight decay*). Esta é uma das técnicas de regularização, a qual atua minimizando a função de custo ou os pesos, a depender dos parâmetros do cálculo realizado para aplicação da técnica. A estratégia de redução de peso é útil quando há limitação do tamanho da rede neural, ou então não é possível aumentar a quantidade de amostras na base de dados de treinamento (ZHANG et al., 2021, Chapter 3).

### 2.1.3 Segmentação Semântica

A segmentação semântica é uma abordagem para rotulação de imagens no nível de pixels com resultado semelhante ao da técnica de classificação apresentada anteriormente, contanto que

haja o retorno das coordenadas do objeto classificado.

Considerando então a classificação de imagem com o enquadramento do objeto, o resultado retornado pode conter outros objetos diferentes na região da caixa delimitadora. Já na segmentação semântica ocorre o oposto por ser trabalhado o processo de classificação em cada pixel em vez de várias regiões por toda a imagem, conseguindo contornar a silhueta do objeto classificado, como pode ser observado na Figura 15. Há também a segmentação por instância que rotula instâncias diferentes de objetos de uma mesma classe como apresentado na Figura 16, se assemelhando mais com uma abordagem de detecção de objetos sem a presença de outros objetos não classificados na região da caixa delimitadora, como apresentado na Figura 17 (SHANMUGAMANI; MOORE, 2018, pp. 129-131).

A abordagem mais simples de realizar a segmentação semântica é por meio de janela deslizante, sendo a imagem dividida em várias partes e aplicando a rotulação nelas individualmente. O problema desta abordagem é a ineficiência já que esta não aproveita as características compartilhadas pelas regiões processadas conforme o deslizamento da janela na imagem.

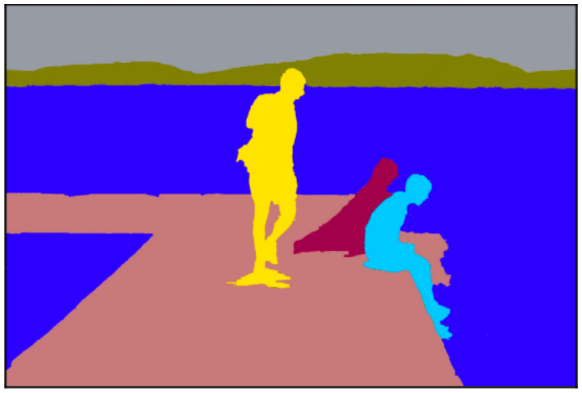
Este ponto negativo ao utilizar janela deslizante é tratado ao trabalhar com redes completamente convolucionais (*Fully Convolucional Network*, FCN). O uso da FCN funciona realizando a extração de características e redução das dimensões espaciais da imagem através das operações convolucionais (*downsample*). Após a redução, por meio de uma camada convolucional 1x1 é transformado o número de canais em número de classes, sendo nas camadas seguintes recuperada a altura e a largura original da imagem através da convolução transposta (*upsample*) (ZHANG et al., 2021, Chapter 13). Vale ressaltar que podem ser realizados ajustes em qualquer arquitetura baseada em redes neurais convolucionais visando a obtenção de um modelo para segmentação semântica (SHELHAMER; LONG; DARRELL, 2017).

Redes baseadas na arquitetura codificador-decodificador (*encoder-decoder*) consiste do processo explicado acima de redução e expansão das dimensões espaciais para recuperar as dimensões da imagem. Um exemplo de arquitetura codificadora-decodificadora é a SegNet (BADRINARAYANAN; KENDALL; CIPOLLA, 2017), apresentada na Figura 18. Mesmo a SegNet conseguindo um resultado menos grosseiro se comparado a uma abordagem original com uso de FCN, ainda são produzidos resultados grosseiros pela perda de informações geradas através do processo de agrupamento (*pooling*).

Originalmente as FCNs consiste apenas das camadas convolucionais sem a aplicação de agrupamento ou qualquer tipo de alteração das dimensões espaciais, resultando em alto custo computacional (Figura 19). Devido ao problema com custo computacional foi proposta a abordagem que pode ser aplicada a qualquer arquitetura de CNN através da substituição das camadas densas por uma camada convolucional na qual a sua profundidade é igual à quantidade de classes (SHANMUGAMANI; MOORE, 2018, pp. 134).

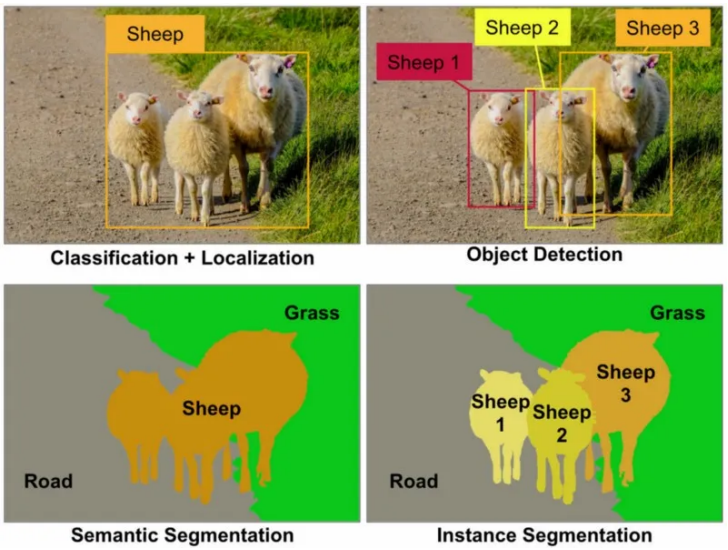
Figura 15 – Resultado da aplicação da segmentação semântica


Figura 16 – Resultado da aplicação da segmentação por instância



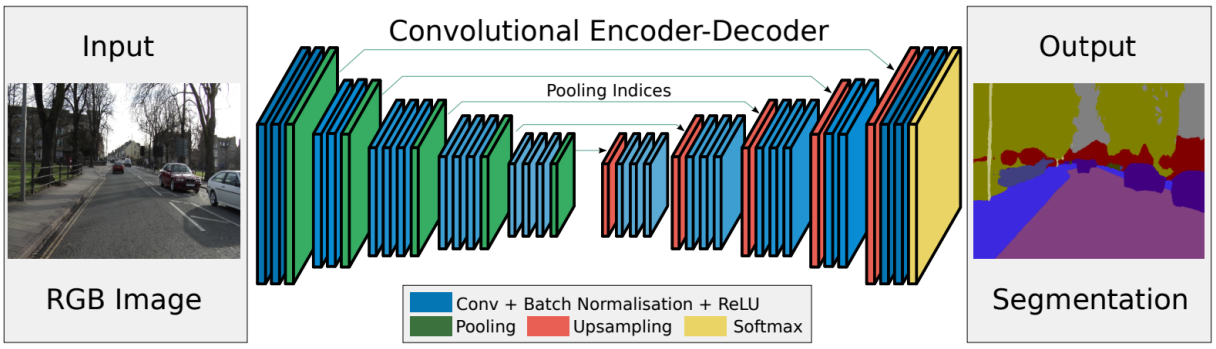
Fonte: Shanmugamani e Moore (2018, pp. 130-131)

Figura 17 – Semelhanças entre segmentação, classificação e detecção



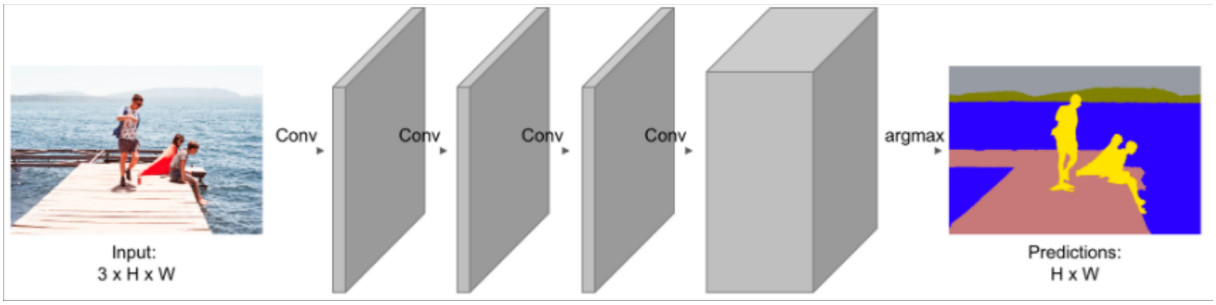

Fonte: Parmar (2018)

Figura 18 – Arquitetura SegNet




Fonte: Badrinarayanan, Kendall e Cipolla (2017)

Figura 19 – Abordagem da FCN sem codificador-decodificador



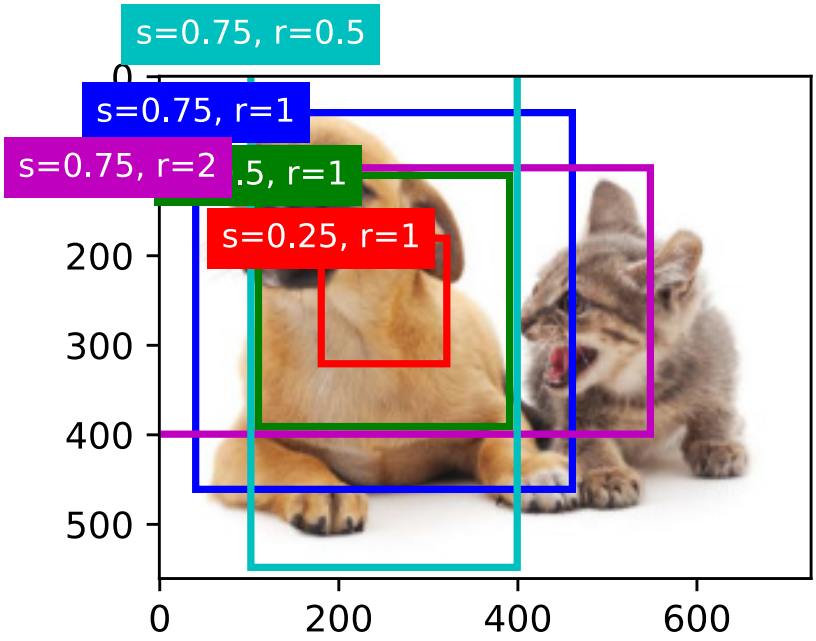

Fonte: Shanmugamani e Moore (2018, pp. 134)

### 2.1.4 Detecção de Objetos

Diferente da classificação de objetos que originalmente reconhece apenas um objeto por imagem, a detecção de objetos surge como uma ferramenta capaz de classificar todos os objetos em uma imagem, além de localizá-los por meio de caixas delimitadoras. Para determinar as caixas delimitadoras que contêm objetos são realizadas várias amostragens de regiões na imagem, sendo efetuadas associações de classes para os objetos localizados.

Uma forma de lidar com a amostragem das regiões é por meio de caixas de ancoragem (*anchor boxes*), as quais passaram a ser implementada na arquitetura YOLO a partir da versão 2 (ou YOLO9000) (REDMON; FARHADI, 2017, p. 2). Ao utilizá-las são geradas múltiplas caixas delimitadoras através da variação dos parâmetros de escala $s$ que pode ter valores reais entre 0 e 1, e variação da proporção $r$ (*aspect ratio*) (ZHANG et al., 2021, Chapter 14). Com base nestes parâmetros são definidas a largura $w_{anchor\_box}$ e a altura $h_{anchor\_box}$ de uma caixa de ancoragem através da Equação 2.8 e da Equação 2.9. Na Figura 20, são apresentadas cinco caixas de ancoragem centradas em um mesmo pixel.

Figura 20 – Caixas de ancoragem centradas em um mesmo pixel




Fonte: Zhang et al. (2021, Chapter 14)

$$w_{anchor\_box} = h \cdot s \cdot \sqrt{r} \tag{2.8}$$

$$h_{anchor\_box} = \frac{h \cdot s}{\sqrt{r}} \tag{2.9}$$

A quantidade de caixas de ancoragem é definida pela Equação 2.10, a qual estabelece que o montante de caixas de ancoragem geradas depende da quantidade de pixels que compõe a imagem, o tamanho da lista de escalas $N$ e o tamanho da lista de proporções $M$.

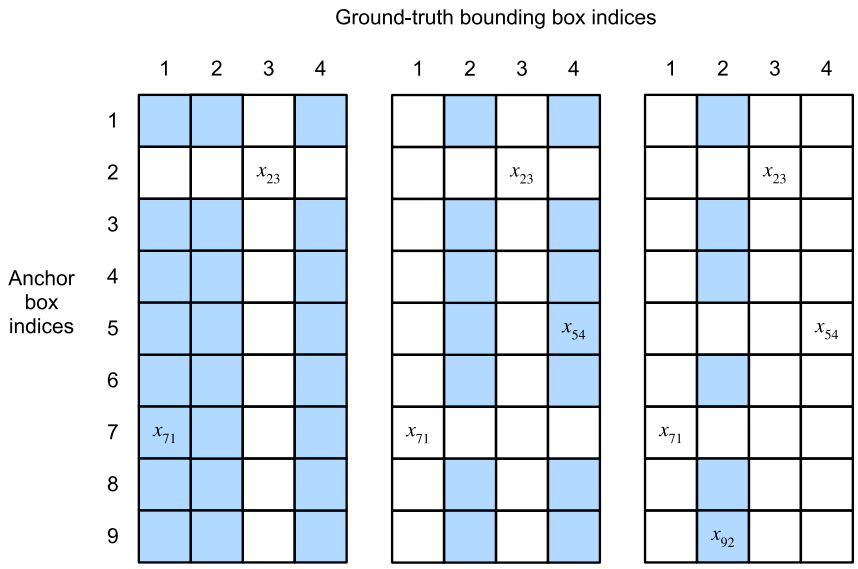
$$anchor_{montante} = w_{imagem} \cdot h_{imagem} \cdot N \cdot M \tag{2.10}$$

Uma abordagem usando força bruta considerando inúmeras escalas e proporções leva a ótimos resultados, já que são verificadas todas as possíveis caixas delimitadoras na busca por objetos. Mas o custo computacional para processar todas essas combinações possíveis é impraticável, sendo na realidade utilizadas listas de escalas e de proporções reduzidas.

A partir destas caixas de ancoragem é feito um procedimento de verificação durante o treinamento ou validação do modelo visando associá-las às verdadeiras caixas delimitadoras dos objetos. Para isso a métrica IOU apresentada na Subseção 2.1.1.1 é utilizada ao ser construída uma matriz da qual cada linha corresponde a uma caixa de ancoragem e cada coluna corresponde a uma

verdadeira caixa delimitadora, sendo o conteúdo de cada célula um valor de IOU considerando a caixa de ancoragem da linha $i$ com a verdadeira caixa delimitadora da coluna $j$. Ao ser preenchida a matriz é utilizada como entrada para um algoritmo que busca o maior valor de IOU dentre as células, representado na Figura 21.

Figura 21 – Matriz de caixas de ancoragem x verdadeiras caixas delimitadoras




Fonte: Zhang et al. (2021, Chapter 14)

O algoritmo busca o maior valor de IOU considerando todas as células da matriz e verifica a qual caixa de ancoragem e verdadeira caixa delimitadora este valor está associado. Após relacionar a linha e a coluna associadas ao maior valor de IOU, estas são deletadas da matriz. O procedimento se repete até chegar ao momento em que todas as verdadeiras caixas delimitadoras foram associadas com as devidas caixas de ancoragem.

No entanto, simplesmente associar todas as verdadeiras caixas delimitadoras às caixas de ancoragem não garante bons resultados, podendo acontecer de existir uma sobreposição ruim entre elas, o que significa que a área da caixa de ancoragem pode não conter, ou contém uma pequena parte do objeto. Para evitar que estes casos sejam considerados pelo algoritmo anteriormente descrito é utilizado um valor limitante, sendo realizada a associação apenas se o calculado valor de IOU for maior ou igual ao limitante. Caso não seja, então na etapa de classificação dos objetos a caixa de ancoragem que seria associada a uma determinada verdadeira caixa delimitadora tem classe definida como plano de fundo (*background*), já que a sua região demarcada não correspondeu minimamente com a região que o objeto se encontra.

Com as associações entre caixas de ancoragem e verdadeiras caixas delimitadoras realizadas inicia-se o processo de classificação dos objetos. Vale ressaltar que na criação da base de dados para treinamento ou validação de um modelo de detecção de objetos são realizadas

anotações em arquivos para cada imagem pertencente, sendo o conteúdo das anotações as coordenadas das verdadeiras caixas delimitadoras dos objetos e a respectiva classe.

Dito isto, a etapa de classificação é executada nas caixas de ancoragem selecionadas e o resultado é comparado com a classe presente na anotação da imagem que rotula o objeto da verdadeira caixa delimitadora associada. Então pode ocorrer de todos os objetos de uma imagem serem corretamente localizados pelo modelo, mas para alguns objetos a classe inferida não corresponde com a classe que o objeto deveria receber.

Já quando não há verdadeiras caixas delimitadoras para ser utilizada a abordagem descrita anteriormente, como quando imagens não rotuladas vão ser processadas pelo modelo, pode ocorrer de várias caixas de ancoragem similares cercarem um mesmo objeto. Neste caso, pode ser realizado o procedimento de supressão não máxima (*non-maximum suppression*), sendo buscado o maior valor de confiança entre os valores associados as caixas delimitadoras, selecionando a caixa delimitadora com o maior valor de confiança e removendo todas as outras candidatas, como apresentado nas Figuras 22 e 23. O valor de confiança é calculado no processo de classificação do objeto dada uma caixa de ancoragem, pois a partir desta é feita a distribuição de probabilidades entre todas as classes para verificar qual destas o objeto provavelmente corresponde (ZHANG et al., 2021, Chapter 14).

Figura 22 – Caixas delimitadoras antes de aplicar a supressão não máxima

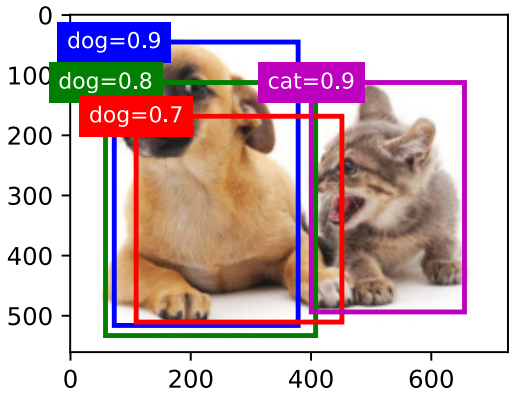
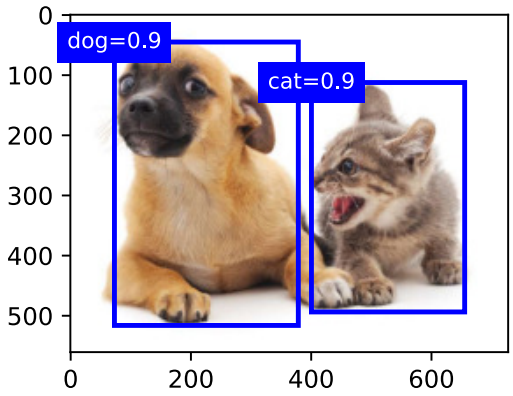



Figura 23 – Caixas delimitadoras após aplicar a supressão não máxima




Fonte: Zhang et al. (2021, Chapter 14)

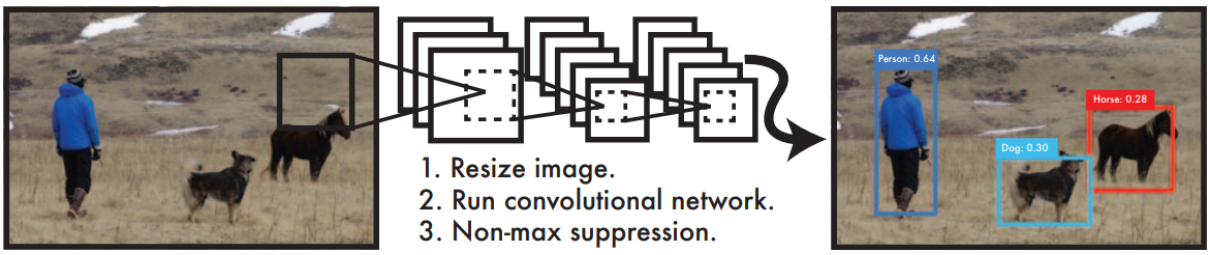
#### 2.1.4.1 *You Only Look Once* (YOLO)

A arquitetura YOLO (REDMON et al., 2016) cujo sistema de detecção é apresentado na Figura 24 surgiu com o proposito de transformar o problema de detecção de objetos em um simples problema de regressão linear. A YOLO trata-se de uma simples rede neural convolucional que implementa a técnica de detecção com apenas um estágio, unificando o processo de predição de múltiplas caixas delimitadoras com a distribuição de probabilidades das classes para os objetos localizados, como mostrado na Figura 25. No cenário de detecção de objetos anterior a YOLO

existiam soluções com abordagens que consideravam o uso de classificadores ou localizadores para realizar detecções em uma imagem, sendo um exemplo o DPM (*Deformable Parts Model*), o qual usa uma abordagem de janela deslizante aplicando um filtro em todas as posições e em diferentes escalas em uma imagem. Nesta abordagem pode ser pensado no detector como um classificador que tem como entrada uma imagem, uma posição dentro dessa imagem e uma escala, sendo determinado no processo de classificação se há ou não uma instância do objeto da classe alvo na região delimitada (FELZENSZWALB et al., 2010, p. 2).

Também há a R-CNN (GIRSHICK et al., 2013) baseada em regiões, diferindo-se do DPM por gerar várias caixas delimitadoras candidatas na imagem e aplicar o classificador em cada uma delas. Nesta arquitetura há um pós-processamento composto pela remoção de detecções duplicadas, realização de refinamento das caixas delimitadoras, entre outros tratamentos que no geral torna o processamento lento e complexo de otimizar por consistir de vários componentes que demandam cuidados individualizados.

Figura 24 – Sistema de detecção da YOLO




Fonte: Redmon et al. (2016, p. 779)

Entre os benefícios gerados por esta unificação das etapas de predição e classificação está a velocidade, pois com essa abordagem na primeira versão desta arquitetura foram alcançadas as taxas de quadros de 45 FPS para o processamento com o modelo padrão e 150 FPS no modelo reduzido chamado de *Fast* YOLO, sendo uma solução indicada para aplicações que lidam com imagens em tempo real. Já quanto ao seu funcionamento é realizada a divisão da imagem em *SxS* grades (*grids*), sendo cada grade responsável pela detecção do objeto do qual o centro está na respectiva célula de grade (*grid cell*) (REDMON et al., 2016, p. 780).

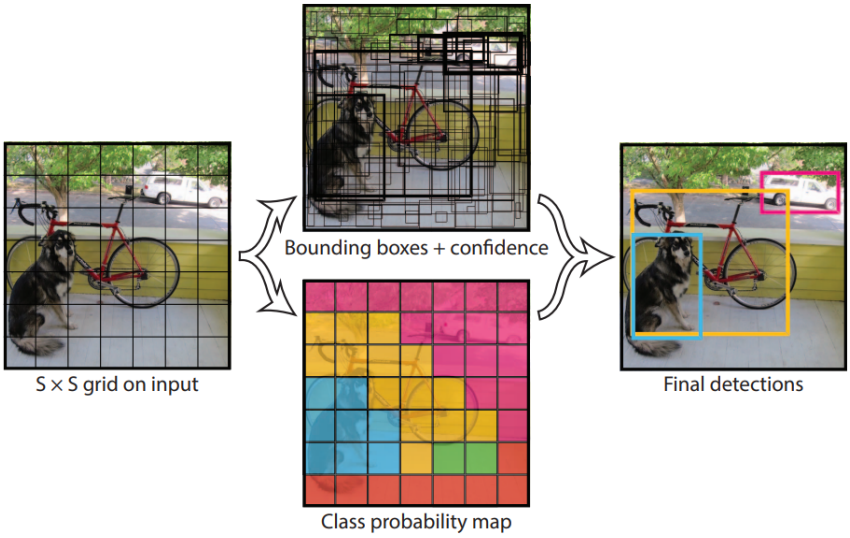
Em cada célula de grade são realizadas $B$ predições de caixas delimitadoras compostas por cinco parâmetros ($x_{centro}$, $y_{centro}$, $w$, $h$, $c$), sendo o parâmetro $c$ o nível de confiança que o modelo tem sobre existir um objeto em uma caixa delimitadora. Já os outros quatro parâmetros são respectivamente o centro no eixo horizontal, o centro no eixo vertical, a largura e a altura da caixa delimitadora. Caso nenhum objeto exista em uma célula de grade é atribuído o valor 0 para a confiança, senão é definido como confiança o mesmo valor que o da métrica IOU ao ser calculada considerando a verdadeira caixa delimitadora. Quanto à predição da classe é avaliada

uma distribuição de probabilidades por célula de grade independente da quantidade de caixas delimitadoras que a compõe.

A primeira versão da YOLO foi inspirada pelo modelo classificador de imagens GoogLeNet. Sendo assim, a rede neural da YOLO é composta por 24 camadas convolucionais seguidas por 2 camadas densas para a versão padrão, e 9 camadas convolucionais com poucos filtros seguidas por também 2 camadas densas na versão *Fast* YOLO.

Como o nome sugere, a YOLO necessita visualizar a imagem apenas uma vez (*You Only Look Once*) por ser considerado o contexto representado por todas as características da imagem para realização da predição das caixas delimitadoras. Isso consequentemente leva à redução de menos da metade dos erros de confusão de objetos com o plano de fundo se comparado com abordagens que não consideram o contexto, como a R-CNN. E apesar da primeira versão ter perdido em acurácia na comparação com os sistemas de detecção que estavam como estado da arte, foi perceptível o ganho obtido quanto ao poder de generalização quando foi realizado o treinamento do modelo com imagens naturais e feito testes com imagens artísticas, superando com larga margem métodos como DPM e R-CNN (REDMON et al., 2016, p. 780).

Figura 25 – Procedimento com grades realizado pela YOLO




Fonte: Redmon et al. (2016, p. 780)

As limitações da primeira versão são por restrições espaciais na predição das caixas delimitadoras, pois cada célula de grade apenas prevê duas caixas delimitadoras e só pode ter uma classe, limitando a quantidade de objetos que podem ser previstos. Além disso, o modelo tem dificuldades para detectar objetos pequenos como revoada de pássaros, ocasionando erros de localização de objetos que geram impacto no *recall* (REDMON et al., 2016, p. 782).

Na versão 2 (YOLOv2) o foco foi melhorar o *recall* e a processo de localizar os objetos sem perda de acurácia do processo de classificação (REDMON; FARHADI, 2017, p. 6518),

além de ser realizado um trabalho para aproveitamento de bases de dados para classificadores visando utilizá-las no treinamento dos modelos de detecção. Já o projeto da arquitetura foi modificado com a adição da normalização por lotes (*batch normalization*) em todas as camadas convolucionais, acelerando a convergência das redes neurais profundas (ZHANG et al., 2021, Chapter 8) e tornando desnecessário o uso da técnica de abandono para evitar o sobreajuste. Ao implementar a normalização por lotes foi obtido um aumento de 2% na métrica mAP.

Outra característica importante é que nesta versão passou a ser possível detectar mais de um objeto em uma célula de grade por serem utilizadas caixas de ancoragem, melhorando o desempenho na tarefa de reconhecimento de múltiplos objetos (MATHEW et al., 2022, p.89) pelo desacoplamento do mecanismo de previsão de classe da localização espacial. Porém, com esta mudança houve influência significativa nas métricas, obtendo-se redução do mAP de 69,5% para 69,2% e aumento do *recall* de 81% para 88%. Já quanto a rede convolucional utilizada deixou de ser a GoogLeNet da primeira versão e passou a ser a Darknet-19, composta por 19 camadas convolucionais e 5 camadas de agrupamento máximo (REDMON; FARHADI, 2017, p. 6521).

Na versão YOLOv3 não houve grandes mudanças, mas ainda assim houve uma melhoria do desempenho considerando que passou a ser utilizado um *backbone* mais eficiente, este dizendo respeito a rede neural convolucional que processa a imagem de entrada. Esta mudança originou-se por adaptações realizadas na Darknet-19 que levou ao desenvolvimento da Darknet-53, composta por 53 camadas (REDMON; FARHADI, 2018, pp. 2–3) e apresentada em uma comparação com outros *backbones* na Figura 26.

Figura 26 – Comparação de *backbones* com o Darknet-53

| Backbone | Top-1 | Top-5 | Bn Ops | BFLOP/s | FPS |
|---|---|---|---|---|---|
| Darknet-19 [15] | 74.1 | 91.8 | 7.29 | 1246 | **171** |
| ResNet-101[5] | 77.1 | 93.7 | 19.7 | 1039 | 53 |
| ResNet-152 [5] | **77.6** | **93.8** | 29.4 | 1090 | 37 |
| Darknet-53 | 77.2 | **93.8** | 18.7 | **1457** | 78 |

Fonte: Redmon e Farhadi (2018, p. 3)

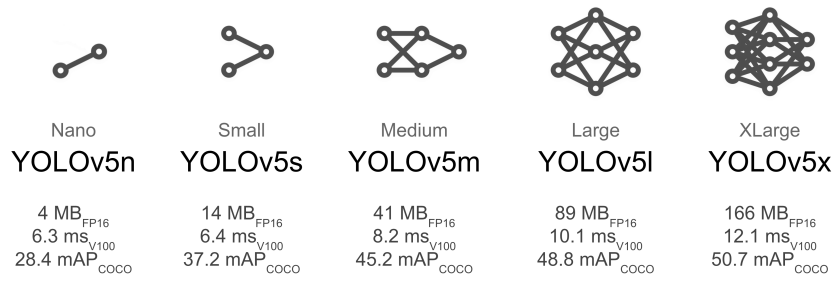
A YOLOv4 que foi lançada em 2020 abordou novos métodos de aumento da quantidade de amostras na base de dados como o *Mosaic* e o *Self-Adversarial Training*. Além disso, passou a ser utilizado algoritmos genéticos para otimizar a seleção de hiperparâmetros, sendo também modificados alguns métodos já existentes para tornar mais eficiente o processo de treinamento e detecção (BOCHKOVSKIY et al., 2020, p. 6).

Já a YOLOv5, que também foi lançada em 2020, se diferencia das versões anteriores por ser baseada na *framework* Pytorch em vez da *framework* Darknet. Além disso, foi lançada pela

Ultralytics[1] e houve mudanças na arquitetura por categorização do tamanho dos modelos, indo do menor modelo (YOLOv5 Nano) até o maior (YOLOv5 X), apresentados na Figura 27.

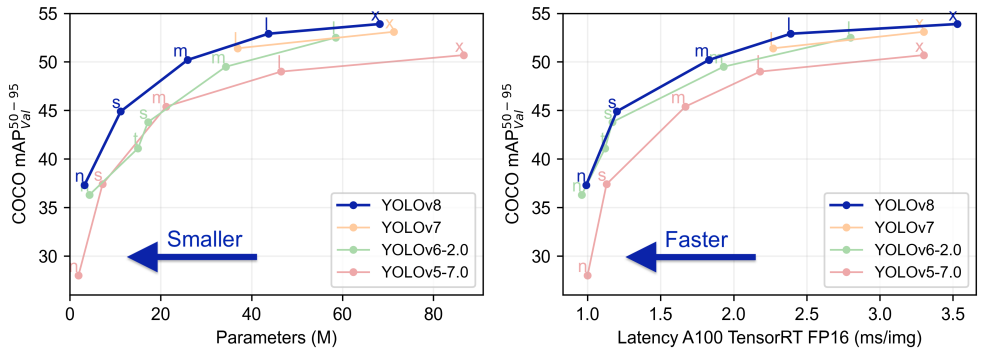
Após a YOLOv4 passaram a existir diversas versões da YOLO, dividindo opiniões sobre quais podem ser consideradas versões oficiais e quais não. Além disso, devido aos vários versionamentos, há dificuldade para rastrear diretamente o que foi melhorado. Já quanto a diferença de desempenho entre as versões pode ocorrer dos resultados de *benchmark* de modelos de arquiteturas em versões mais antigas apresentarem desempenhos melhores do que versões recentes, o que pode acontecer devido ao ajuste de hiperparâmetros. Em todo caso, após a YOLOv5 vieram a YOLOv6, YOLOv7 e YOLOv8, sendo esta última a mais atual e também lançada pela Ultralytics, diferente da YOLOv6 e YOLOv7. A Figura 28 apresenta a diferença de desempenho entre estas quatro versões.

Figura 27 – Categorias dos modelos na arquitetura YOLOv5




Fonte: Jocher et al. (2022)

Figura 28 – Comparação de desempenho entre a YOLOv5, YOLOv6, YOLOv7 e YOLOv8




Fonte: Jocher et al. (2022)

[1] Disponível em: <https://ultralytics.com/>. Acesso em abril de 2023

### 2.1.5 Rastreio de Objetos

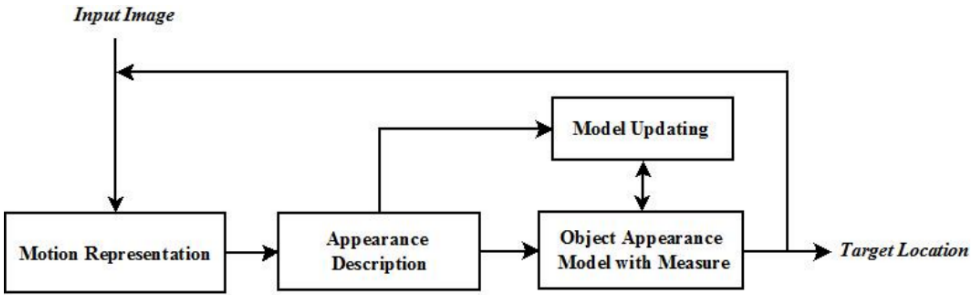
O rastreio de objetos é um dos principais campos de pesquisa na área de visão computacional devido às diversas aplicações em que é útil como o reconhecimento de movimentos, detecção de anomalias, análises esportivas, navegação robótica e condução automática. O rastreio é definido pela estimativa do estado do objeto alvo em uma sequência de imagens, então é esperado que o algoritmo seja robusto para identificar a localização do objeto mesmo com variação de iluminação, baixa resolução de imagem, variação do plano de fundo, oclusão parcial da referência, variação de escala, movimentação rápida, deformação, rotação, dentre outros fatores que tornam o desenvolvimento de algoritmos para rastreio de objetos algo desafiador (ABBASI; REZAEIAN, 2021, pp. 1–2).

Um dos benefícios do uso de algoritmos de rastreio é ser feita apenas uma inicialização com o objeto alvo aproveitando durante o rastreio informações já processadas anteriormente, reduzindo a latência de inferência de novas imagens. Já em modelos para detecção toda a estrutura é executada a cada nova imagem recebida, sem aproveitamento de informações já processadas em imagens anteriores e geralmente com uso de operações convolucionais que custam caro computacionalmente.

Para Qin (2014) um típico sistema de rastreio de objetos pode ser decomposto nos componentes modelo ou representação de movimento (*motion representation*), descrição da aparência (*appearance description*), modelo de aparência do objeto com medidas (*object appearance model with measure*) e atualização do modelo (*model updating*), como pode ser observado na Figura 29.

O componente de **modelo do movimento** atua restringindo o espaço para a busca do objeto alvo quando é recebida uma nova imagem no sistema, sendo geralmente dividido em quatro categorias. A primeira é a predição implícita de movimentos que não aplica restrições ao modelo de movimento, utilizando métodos de otimização em vez disso, mas para obtenção de bons resultados é necessário que o alvo tenha movimentos relativamente pequenos.

Figura 29 – Diagrama de um típico sistema de rastreio baseado em aparência




Fonte: Qin (2014, p. 3)

Na segunda categoria os modelos utilizam janela deslizante em que são consideradas todas as possíveis localizações na imagem, não fazendo nenhuma suposição sobre a movimentação do objeto. O problema surge pela alta carga computacional necessária para execução, não sendo viável considerando que os rastreadores devem performar em tempo real.

A terceira categoria possível é também por meio de janela deslizante, mas em uma região localizada considerando a posição anterior do objeto, se tornando problemático quando o objeto alvo se desloca muito rapidamente. E por último, e diferente das duas últimas categorias, os modelos não utilizam janela deslizante, usando em vez disso um modelo de movimento gaussiano probabilístico centrado na posição anterior do objeto alvo (QIN, 2014, pp. 7–9).

Outro componente do sistema de rastreio é a **descrição de aparência** que desempenha papel fundamental, pois a qualidade da descrição remete diretamente à qualidade da atuação do rastreio. Este componente torna o objeto alvo distinguível dos objetos que não são alvos no espaço das características. Comumente para a descrição de uma região de pixels são utilizados descritores estatísticos, como histogramas e matriz de covariância, consistindo a diferença por no histograma acomodar as características uma de cada vez levando a uma carga exponencial com o número de características quando estas estão em conjunto. Por outro lado, com a matriz de covariância são fundidas múltiplas características que podem estar correlacionadas possibilitando a integração de informações de espaço, cores e gradiente em uma única matriz (QIN, 2014, pp. 9–12).

Já o componente **modelo de aparência do objeto** pode ser visto sob duas abordagens visando a constância durante o rastreio do objeto. A primeira abordagem é via modelos generativos que agem modelando o objeto alvo e ignorando o plano de fundo, sendo obtido um modelo de aparência para o objeto no qual o rastreador encontrará a região que melhor corresponde com a localização alvo (ABBASI; REZAEIAN, 2021, p. 2). Vale ressaltar que como o plano de fundo é ignorado, então para conseguir sustentar o rastreio sob mudanças de aparência do objeto alvo são realizadas atualizações *online*, que será explicada logo mais. Já a outra abordagem considera o plano de fundo no processamento, tratando-se de modelos discriminativos que buscam um limite de decisão que melhor separa o que é objeto do que é plano de fundo (QIN, 2014, pp. 12–16).

Por último, o componente **atualização de modelo** assume um papel delicado ao ser responsável por tornar o rastreio tolerante a mudanças no objeto e no plano de fundo. Estas mudanças podem ser divididas em variações intrínsecas, voltadas para as mudanças de pose ou deformação da forma do objeto, ou variações extrínsecas em que estão relacionadas mudanças na iluminação, movimento da câmera e oclusão.

Essa tolerância surge a partir do uso de métodos adaptativos que possibilitam realizar a atualização do modelo de aparência do objeto, pois este precisa ser ajustado para as novas circunstâncias e para isso é necessária a utilização de algoritmos *online*, possibilitando a aprendizagem contínua durante a execução do rastreio.

O componente de **atualização de modelo** é delicado por marcar uma região ao redor do objeto por meio de uma caixa delimitadora, sendo possível surgir pequenos erros no processo de divisão do que é relevante ser considerado em primeiro plano e o que deve ser considerado plano de fundo, levando então à degradação do processo de rastreio. Sendo assim, é necessário um balanço entre adaptatividade e estabilidade ao utilizar métodos de aprendizagem *online* (QIN, 2014, pp. 16–18).

Outra descrição sobre algoritmos de rastreio é dada em Fiaz et al. (2018, p. 3). Este trabalho sugere que algoritmos de rastreio podem ser categorizados como rastreador de único ou múltiplos objetos, generativo ou discriminativo como descrito anteriormente, contextual ou não, e *online* ou *offline*. As duas primeiras tratam de algoritmos que podem realizar o rastreio de apenas um, ou de múltiplos objetos na sequência de imagens. Já os algoritmos categorizados como contextual consideram o contexto da imagem avaliado através dos pixels para realizar o processamento, enquanto os não contextuais não consideram.

Quanto as categorizações entre generativa ou discriminativa é dito que a primeira está relacionada com algoritmos que realizam o processamento por meio de uma busca pela janela de melhor enquadramento com a referência, já na discriminativa encontra-se o objeto alvo na cena atualizada através da discriminação do plano de fundo.

As categorizações *online* ou *offline* identificam como que a aprendizagem ocorre, pois na *online* são realizadas adaptações no algoritmo de rastreio a partir de novas informações do objeto e do plano de fundo ainda durante o rastreio. Por outro lado, na *offline* é necessário fazer um treinamento com algumas imagens antes do procedimento de rastreio, pois não serão feitas adaptações durante a execução, como acontece ao utilizar filtros ASEF (*Average of Synthetic Exact Filters*) e UMACE (*Unconstrained Minimum Average Correlation Energy*).

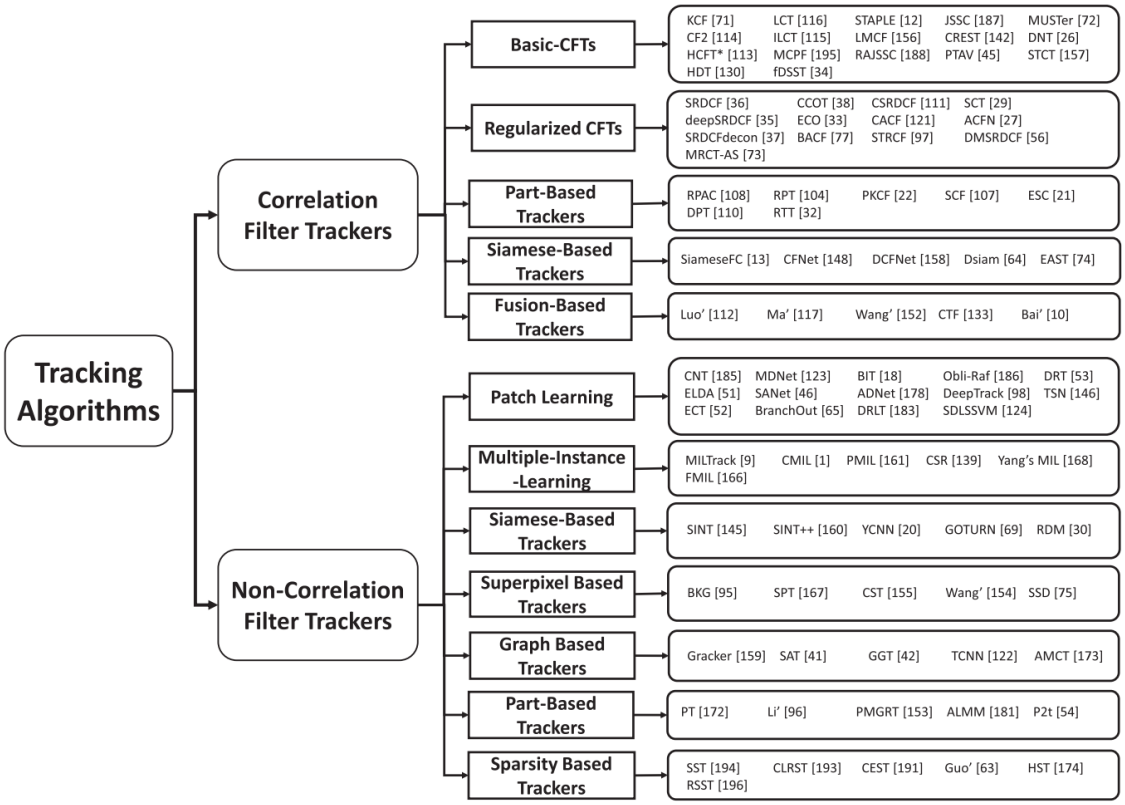
Além do particionamento dos algoritmos de rastreio através das categorizações, há abordagens diferentes entre eles para lidar com as características dos dados, sendo uma delas por meio de ferramentas como os Histogramas de Gradientes Orientados (*Histogram of Oriented Gradients*, HOG), Transformação de Características Invariantes em Escala (*Scale-Invariant Feature Transform*, SIFT), Padrão Binário Local (*Local Binary Pattern*, LBP) e *Colornames*. Enquanto em outra abordagem está o uso do aprendizado profundo no desenvolvimento da ferramenta de rastreio, pois devido ao potencial de codificar informações em vários níveis o algoritmo torna-se mais robusto para lidar com variações de aparência dos alvos com o ônus de necessitar de vários dados de treinamento.

Os rastreadores também podem ser classificados com base no uso de filtros correlacionais (*Correlation-Filter based Trackers*, CFTs), sendo apresentada a taxonomia dos algoritmos de rastreio a partir desta classificação na Figura 30. Nos algoritmos CFTs é inicializado o filtro de correlação através da região de interesse em que está o objeto na primeira imagem da sequência, então durante o rastreio é inferida a posição do objeto em uma nova imagem recebida a partir da posição estimada pelo processamento realizado na última imagem.

Nesse processo a representação da aparência do objeto é realizada através da extração de características para a construção de um mapa de características com limites suavizados através da aplicação de um filtro cosseno. O mapa resultante da operação de correlação surge a partir da multiplicação elementar entre o filtro de aprendizagem adaptativa e as características extraídas por meio da transformada de Fourier, sendo em seguida aplicada a transformada inversa de Fourier para se obter um mapa de confiança atribuindo pontuação máxima de confiança na nova posição do objeto na imagem atual, sendo realizada a atualização do local recentemente previsto para o alvo através da extração de características e atualização dos filtros de correlação (FIAZ et al., 2018, p. 5).

Abbasi e Rezaeian (2021, p. 2) diz que os maiores esforços na construção de rastreadores estão em basear estes a filtros discriminativos correlacionais (*Discriminative Correlation Filter*, DCF) devido à robustez e eficiência em comparação com abordagens tradicionais.

Figura 30 – Taxonomia de algoritmos de rastreio




Fonte: Fiaz et al. (2018, p. 5)

#### 2.1.5.1 KCF

O algoritmo de rastreio KCF foi a primeira extensão teórica bem sucedida do DCF padrão através da formulação *kernelized* (LUKEZIC et al., 2017, p. 3) visando realizar a aprendizagem e detecção sobre *patches* de imagens eficientemente, sendo *patches* equivalente às regiões de

interesse com o objeto alvo para rastreio. A motivação para o desenvolvimento partiu do sucesso do uso de filtros correlacionais no cenário de rastreio de objetos (HENRIQUES et al., 2015, p. 2) devido à convolução entre duas regiões de interesse em diferentes translações relativas equivaler a uma simples operação de multiplicação no domínio de Fourier, permitindo ser realizada a especificação de um classificador linear para várias translações ou mudanças na imagem de uma só vez.

Mas assim como ocorreu como parte da motivação do desenvolvimento do algoritmo CSRT que será apresentado na próxima Subseção (2.1.5.2), no KCF é buscado o desenvolvimento de uma solução que permita fugir das limitações impostas pela utilização da transformada de Fourier (HENRIQUES et al., 2015, p. 3).

A problemática surge porque os algoritmos anteriores ao KCF focaram mais no ambiente do que no objeto (amostras negativas) por meio de regiões de interesse em diferentes localizações e escalas, refletindo um conhecimento prévio de que o classificador será avaliado sob estas condições. Porém, há infinitas possibilidades de serem obtidas amostras negativas dada uma imagem, sendo um desafio selecionar quais destas considerar mantendo a demanda computacional baixa devido à restrição crítica quanto a latência para inferência, comumente ocorrendo a escolha de apenas algumas amostras aleatoriamente como em Zhang, Zhang e Yang (2012).

Sendo assim, foram desenvolvidos no KCF mecanismos para incorporar analiticamente milhares de amostras em diferentes translações relativas sem a necessidade de explicitamente interagir sobre elas, os quais recebem o nome de matriz circulante. Isto foi possível graças ao trabalho realizado no domínio de Fourier, o que tornou viável o uso de modelos específicos para as translações.

Isso permitiu ser proposto um rastreador baseado em *Kernel Ridge Regression*, obtendo-se uma versão *kernelized* de um filtro correlacional linear. Então, foi possível aproveitar as vantagens do truque do núcleo (*kernel trick*), que permite atuar no espaço de características original sem computar as coordenadas dos dados em um espaço dimensional superior (ZHANG, 2018) juntamente com a mesma complexidade computacional dos filtros de correlação linear. O que possibilitou alcançar uma estrutura que incorpora múltiplos canais de características que por utilizar um núcleo linear foi conseguida uma rápida extensão dos filtros de correlação linear para o caso de múltiplos canais, permitindo o uso de recursos estado da arte para dar um importante impulso no desempenho (HENRIQUES et al., 2015, p. 7).

Em outras palavras, o KCF usa inúmeras amostras negativas com complexidade reduzida devido o uso do duplo espaço para aprender dados de alta dimensão com o truque do núcleo, e o aproveitamento da propriedade de diagonalizar a matriz circulante no domínio de Fourier, resultando em um algoritmo com baixa complexidade computacional e consequentemente com maior velocidade (YADAV; PAYANDEH, 2021, p. 4).

A estrutura da matriz circulante é formada através do uso de uma amostra da imagem do

alvo em duas dimensões, gerando amostras adicionais a partir de todas as translações possíveis da imagem. O uso de todas essas amostras negativas na fase de aprendizagem resulta em um classificador discriminatório mais eficiente.

Assumindo que as características da imagem são representadas por $a_{ij}$ na matriz $A$ na Equação 2.11, a matriz circulante é gerada a partir do conteúdo da matriz A através da função $circ$ apresentada na Equação 2.12, onde $a_m$ representa cada linha da matriz $A$.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \tag{2.11}$$

$$X_m = circ(a_m) \tag{2.12}$$

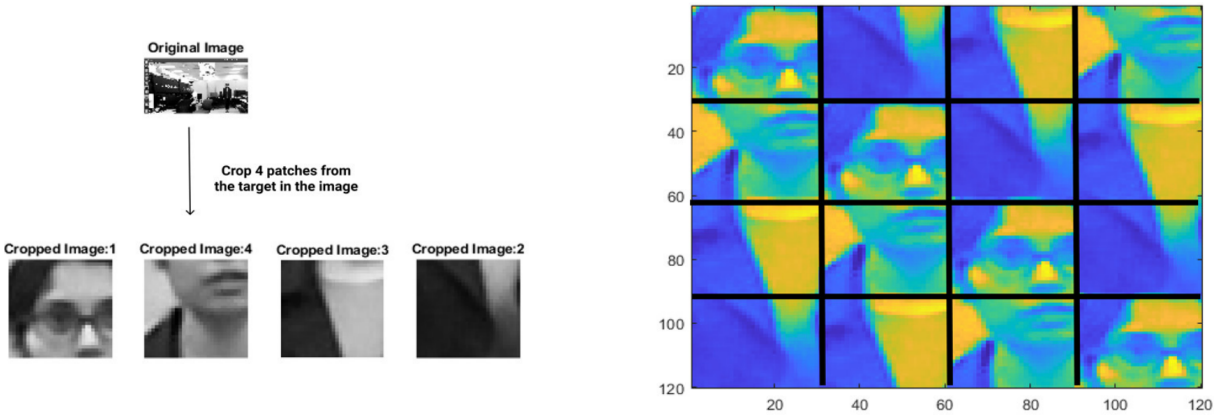
Considerando as quatro linhas na matriz de características, os resultados da função $circ$ podem ser representados como apresentado na Equação 2.13. Já a matriz circulante é representada na Equação 2.14.

$$\begin{aligned} X_0 &= circ(a_0) = circ(a_{11}a_{12}a_{13}a_{14}) \\ X_1 &= circ(a_1) = circ(a_{21}a_{22}a_{23}a_{24}) \\ X_2 &= circ(a_2) = circ(a_{31}a_{32}a_{33}a_{34}) \\ X_3 &= circ(a_3) = circ(a_{41}a_{42}a_{43}a_{44}) \end{aligned} \tag{2.13}$$

$$X' = \begin{bmatrix} X_0 & X_1 & X_2 & X_3 \\ X_3 & X_0 & X_1 & X_2 \\ X_2 & X_3 & X_0 & X_1 \\ X_1 & X_2 & X_3 & X_0 \end{bmatrix} \tag{2.14}$$

Como cada $X_m$ contém uma linha da matriz $A$ com quatro elementos, a matriz circulante $X'$ tem dimensão $16x16$. Na Figura 31, é ilustrado um exemplo do processo e da matriz circulante obtida.

Figura 31 – Matriz circulante construída a partir de quatro regiões de interesse do objeto alvo




Fonte: Yadav e Payandeh (2021, p. 5)

#### 2.1.5.2 CSRT

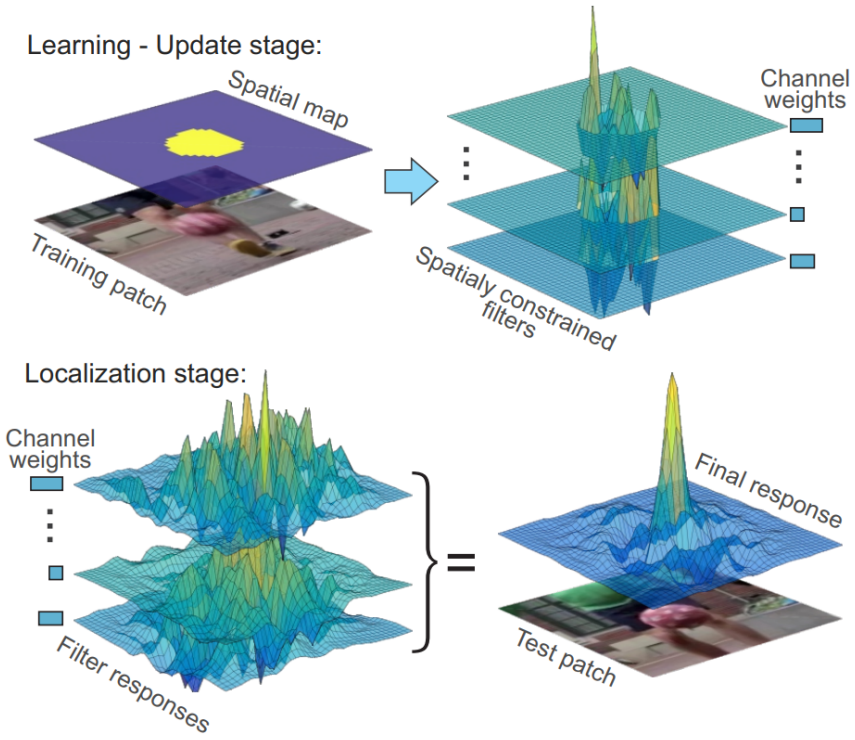
O algoritmo de rastreio CSRT (*Channel and Spatial Reliability Tracker*) ou CSR-DCF (*Discriminative Correlation Filter with Channel and Spatial Reliability*) (LUKEZIC et al., 2017) atua na problemática de rastreio de objetos com curto prazo (*short-term*) e livre de modelos (*model-free*). Ou seja, o objeto alvo deve ser continuamente localizado em uma sequência de imagens a partir de um único exemplo da sua aparência. Na taxonomia apresentada na Figura 30 este algoritmo se enquadra na categoria de algoritmos rastreadores baseados em filtros correlacionais regularizados (*Regularized CFTs*).

A proposta do algoritmo é realizar alguns ajustes tanto na atualização como no processo de rastreio na abordagem padrão do DCF, sendo na Figura 32 apresentada uma visão geral da abordagem do CSRT. Um dos problemas encontrados ao considerar a abordagem padrão do rastreio baseado em DCF é devido o deslocamento circular, o qual é utilizado para aprendizagem do filtro a partir de uma resposta anteriormente definida na imagem de treinamento.

A correlação circular é utilizada para possibilitar de forma mais eficiente a aprendizagem através da transformação rápida de Fourier (*Fast Fourier Transform*, FFT), mas devido o uso da FFT o alcance da área de detecção é limitado pela restrição de igualdade no tamanho da região de busca e do filtro. Então, a estratégia utilizada para resolver o problema é realizar a aprendizagem do filtro com uma região maior, contudo isso resulta também no aumento da área que consiste o plano de fundo levando a perdas significativas de desempenho como apresentado na Figura 33.

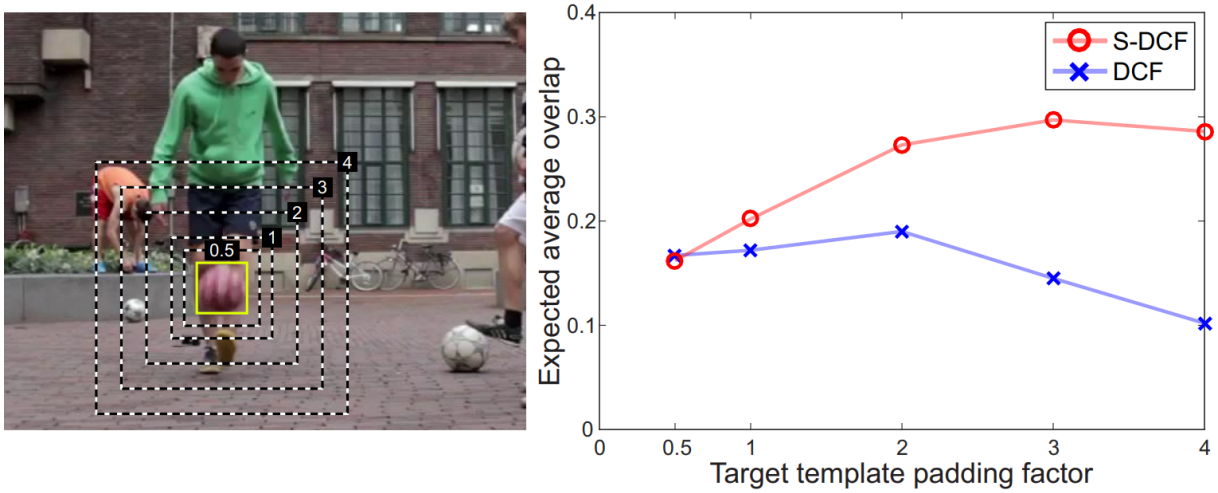
Outro problema encontrado é com relação à suposição de que a forma do objeto alvo pode ser bem aproximada por um retângulo alinhado aos eixos, caso contrário o filtro pode tender a aprender o plano de fundo deteriorando e levando a falhas no procedimento de rastreio do objeto.

Figura 32 – Visão geral da abordagem CSR-DCF




Fonte: Lukezic et al. (2017, p. 2)

Figura 33 – Resultados de desempenho do DCF padrão e do S-DCF em gráfico de Sobreposição Média Esperada (EAO) como função do tamanho da região de busca




Fonte: Lukezic et al. (2017, p. 3)

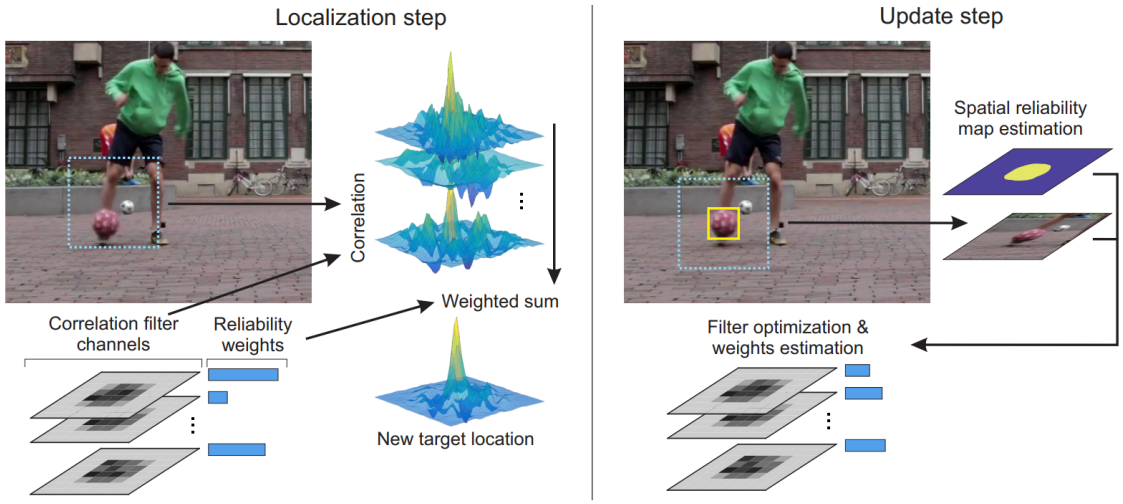
Então, o CSR-DCF atua tratando estes dois problemas por meio de pontuações de confiabilidade espacial e de canais. Um mapa de confiabilidade espacial atua ajustando o suporte

do filtro para a parte adequada para rastreio aumentando a região de busca e melhorando o rastreio de objetos não retangulares. Já os valores de confiabilidade dos canais refletem a qualidade dos filtros treinados, sendo usados como coeficientes de ponderação de características na localização. Na Figura 34, é possível observar uma visão geral destas modificações trabalhando na etapa de localização e de atualização.

Ainda sobre a Figura 34, na etapa de localização as características são extraídas da região de busca centrada na posição estimada do objeto e correlacionadas com o filtro treinado. Com isso, o objeto é localizado pela soma dos resultados de correlação ponderados com as pontuações estimadas de confiabilidade do canal.

Já na etapa de atualização a região de treinamento é centralizada no local estimado pela etapa de localização. Então, tanto o histograma do primeiro plano que atua na identificação do objeto alvo como o histograma do plano de fundo são extraídos, sendo atualizados pela média exponencial de movimentação considerando a imagem atual e a anterior. Vale ressaltar que o mapa de confiabilidade espacial é extraído para cada imagem, independente de haver grandes mudanças no objeto alvo, seja mediante deformação ou rotação.

Figura 34 – Aplicação das modificações através do mapa de confiabilidade espacial




Fonte: Lukezic et al. (2017, p. 8)

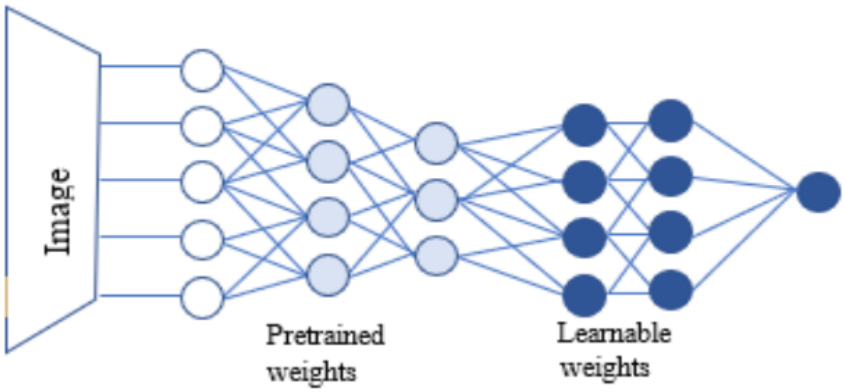
### 2.1.6 Transferência de Aprendizado

A transferência de aprendizado é utilizada para transferir o conhecimento adquirido por um modelo treinado em uma base de dados para um novo modelo com uma base de dados diferente. Um caso de uso seria realizar a transferência de conhecimento de um modelo treinado na base de dados ImageNet, que não contém a classe “sobrancelhas”, para obtenção de um modelo cujo contexto é classificar sobrancelhas.

A ideia da transferência de aprendizado é aproveitar a capacidade de generalização da etapa de extração de características do modelo pré-treinado, sendo esse o nome dado para o modelo previamente treinado em uma base de dados que é utilizado para transferência de conhecimento. Então, a partir dessa transferência o novo modelo pode identificar mais facilmente bordas, texturas, formas e composição dos objetos se este modelo for destinado para realizar detecção de objetos em imagens (ZHANG et al., 2021, Chapter 14).

Além disso, quando o conjunto de dados utilizado para o treinamento do novo modelo é pequeno, recorrer à transferência de aprendizado permite tratar o problema de sobreajuste no processo de treinamento do modelo. O procedimento básico para a transferência é treinar um modelo base e copiar a estrutura das $N$ camadas iniciais destinadas para extração de características para o modelo alvo, existindo duas abordagens principais para lidar com essas camadas de extração de características. A primeira é através do congelamento dos pesos associados aos neurônios dessas $N$ camadas iniciais do modelo pré-treinado (FARHOOD et al., 2022), sendo feita uma inicialização aleatória nas camadas densas que realizam a classificação, como apresentado na Figura 35.

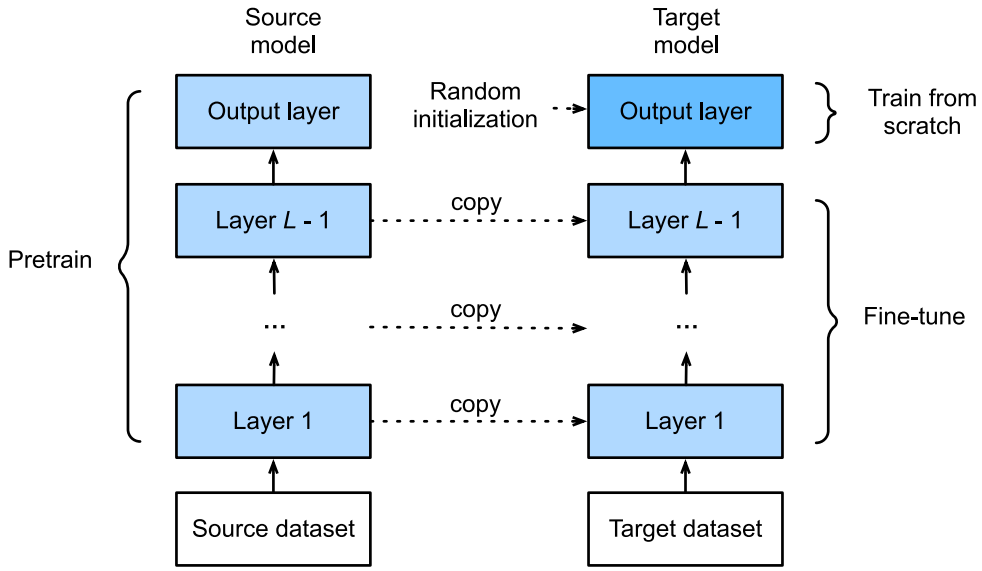
Figura 35 – Transferência de aprendizado com congelamento dos pesos do modelo pré-treinado




Fonte: Chouhan et al. (2020, p. 6)

A outra abordagem é mediante ajuste fino (*fine-tuning*), mas sua aplicação depende do tamanho da base de dados do modelo alvo e da quantidade de parâmetros nas $N$ camadas iniciais do modelo pré-treinado. Um caso em que pode surgir um sobreajuste é por meio do uso de uma base de dados do modelo alvo pequena, mas com grande número de parâmetros nas camadas transferidas, sendo preferível nesse caso usar a abordagem de congelamento. No entanto, se a base de dados for grande ou o número de parâmetros for pequeno, o ajuste fino passa a ser uma escolha (YOSINSKI et al., 2014, p. 2). Na Figura 36, é ilustrado o procedimento de ajuste fino dos pesos.

Figura 36 – Transferência de aprendizado com ajuste fino dos pesos do modelo pré-treinado




Fonte: Zhang et al. (2021, Chapter 14)

O procedimento do ajuste fino é realizar o ajuste dos pesos copiados do modelo pré-treinado sem alterar a estrutura que compõe a etapa de extração de características durante o treinamento do novo modelo.

### 2.1.7 Compressão de Modelos

A compressão de modelos é um tópico importante ao considerar a possibilidade de trabalhar com modelos de aprendizado de máquina em ambientes sob restrição de processamento e/ou de memória. Exemplo disso é o caso de satélites espaciais que as limitações exigem otimizações dos dispositivos ou aplicações que vão ser implantadas. Um desafio de operar modelos de aprendizado profundo neste contexto é devido geralmente ser necessário equipamentos robustos para conseguir processar em tempo hábil as informações, como imagens terrestres tiradas do espaço para realizar detecções que podem chegar a resolução de 8000x8000 pixels.

Neste contexto, não é viável ocupar tanta largura de banda para enviar estas informações para o planeta terra visando realizar o processamento em computação em nuvem, e também não é viável aumentar tanto o peso e gastos energéticos com dispositivos físicos robustos o suficiente para realizar um processamento local. A utilização de técnicas de compressão podem permitir o uso do aprendizado profundo no contexto espacial, contornando os problemas descritos (LOFQVIST; CANO, 2020).

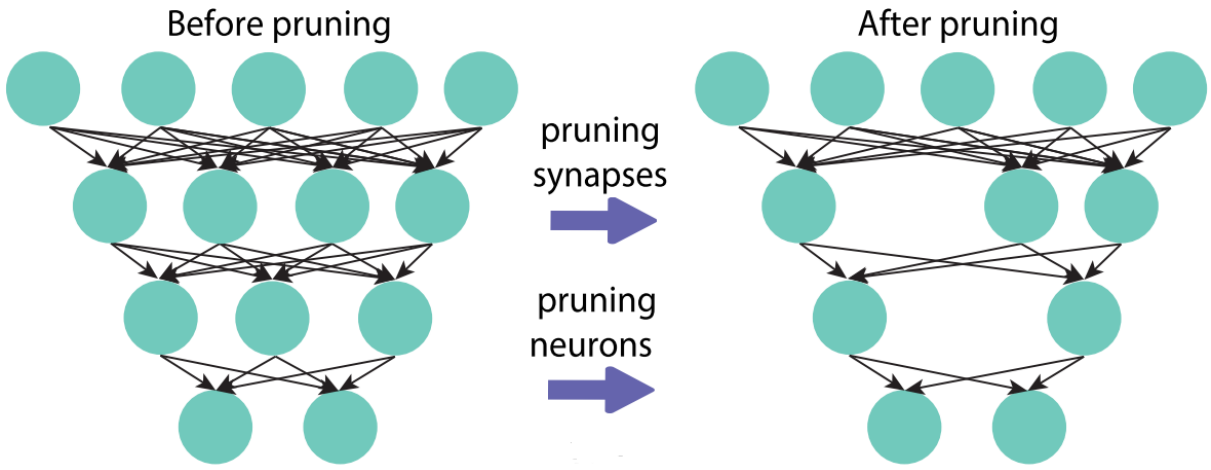
Quando é realizada a compressão de um modelo denso são esperadas mudanças em dois atributos principais, o tamanho do modelo e a latência de inferência. Além destes dois, a acurácia do modelo pode sofrer alteração devido modificações estruturais na rede neural, principalmente

quando parâmetros relevantes acabam sendo considerados no processo de compressão. No entanto, é esperado que sejam considerados parâmetros que geram pouca ou nenhuma influência no desempenho do modelo, permitindo redução da latência e do tamanho do modelo enquanto se mantém ou perde pouca acurácia dos resultados.

Uma abordagem de compressão é por meio da **poda (*pruning*)**, a qual funciona com base na eliminação de redundância dos pesos através da remoção de conexões entre neurônios, ou então a remoção total de neurônios. No primeiro caso são zerados os valores dos pesos nas matrizes, e como cada peso está associado a uma conexão zerá-los desfaz a comunicação entre os neurônios envolvidos, sendo esta abordagem chamada de **poda não estruturada (*Unstructured Pruning*)**. Já na segunda abordagem a poda acontece seguindo "estruturas", consequentemente recebe o nome de **poda estruturada (*Structured Pruning*)**.

Na **poda não estruturada** é obtido um modelo escasso (*sparsity*), que a depender do quanto o modelo foi podado pode ser alcançada redução até no consumo de memória. No entanto, esta abordagem pode levar a perdas de compatibilidade dos dados com o paralelismo em GPU, ou CPU com múltiplos núcleos (MA et al., 2020, p. 1). Na imagem 37 é apresentada esta abordagem.

Figura 37 – Procedimento de poda não-estruturada



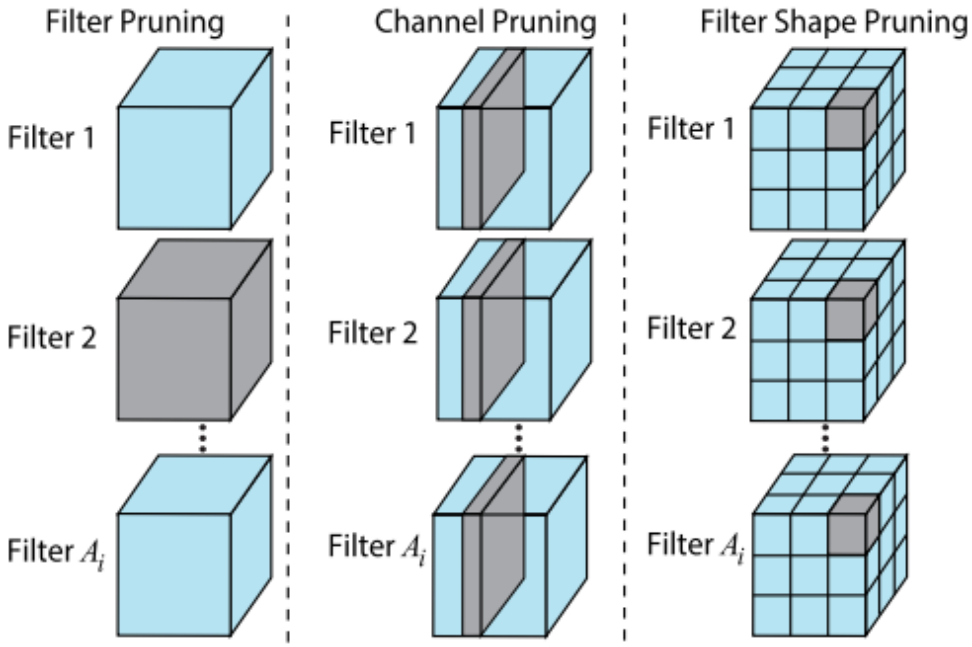

Fonte: Ma et al. (2020, p. 2)

Já na **poda estruturada** são consideradas "estruturas" ao serem zerados os valores dos pesos, sendo realizadas podas de canais inteiros, filtros ou partes dos filtros, como ilustrado na Figura 38. Nesta abordagem a matriz mantém-se estruturalmente completa, mas com dimensões reduzidas, resolvendo o problema de compatibilidade com o paralelismo gerado na poda não estruturada (MA et al., 2020, p. 2).

Diferente da poda que realiza a redução na redundância dos pesos, a **quantização** que é outra forma de compressão atua reduzindo o tamanho destes pesos através da redução do número de bits necessários para representá-los (HAN; MAO; DALLY, 2016). Os benefícios disso são

mais voltados para o dispositivo físico que atuará como unidade de processamento já que há um impacto no tamanho do modelo, sendo possível realizar a execução do modelo em dispositivos de sistemas embarcados dos quais têm limitação quanto a capacidade de armazenamento.

Figura 38 – Procedimentos possíveis para poda estruturada




Fonte: Ma et al. (2020, p. 2)

Um exemplo disto é o trabalho feito em Yan, Liu e Fu (2021), no qual foi trabalhada a poda dinâmica atrelada à quantização dos pesos para o uso de um FPGA (*Field-Programmable Gate Array*) como unidade de processamento. Como o FPGA não dispõe de unidades para realizar operações de ponto flutuante a quantização é aplicada para reduzir o tamanho dos pesos de 32 bits para 8 bits. Mas vale ressaltar que há outros métodos de quantização como a quantização binária, ternária ou deslocamento de bit (*bit-shift*).

## 2.2 *Automatic Licence Plate Recognition* (ALPR)

O reconhecimento automático de placas veiculares é objeto de pesquisa na área de visão computacional devido às suas diversas aplicações, sendo exemplos a prática das leis de trânsito, controle de acesso a áreas restritas e monitoramento de tráfego.

Porém, este tema envolve uma considerável complexidade ao ter que lidar com as técnicas de aprendizagem de máquina aplicadas ao treinamento de modelos capazes de lidar com imagens em diversos cenários diferentes, dos quais devem oferecer robustez suficiente para extrair cadeias de caracteres de placas de veículos com alta acurácia.

Ou seja, o modelo lidará com imagens com variações de iluminação, opacidade e posicionamento, resultando facilmente em placas que não estão alinhadas à câmera, o que significa que a sua forma geométrica que originalmente tem três dimensões é distorcida ao ser capturada por uma câmera convencional. Então, só é possível utilizar as medidas reais da placa como forma de validação para evitar falsos positivos sob restrições, como por meio de uma referência de um objeto na imagem com medidas reais conhecidas para aplicação de uma transformação de perspectiva na placa, com o ônus de tornar o processamento mais custoso.

Além disso, o modelo treinado tem que lidar também com planos de fundo por vezes completamente diferentes, principalmente quando o reconhecimento deve ser feito com imagens capturadas por câmeras em deslocamento, como as câmeras de bordo instaladas em veículos.

A seguir, são apresentados outros pontos que tornam desafiador o processo de desenvolvimento de modelos para ALPR visando o mercado:

- Ofuscação: Neste caso a placa pode estar suja, principalmente por lama considerando como referência o Brasil, onde principalmente em regiões interioranas facilmente há zonas rurais. Assim como podem ocorrer ofuscações devido ao posicionamento de objetos entre a câmera e a placa;
- Proporções: Considerando os padrões para placas brasileiras, as proporções destas para motocicletas distinguem-se das de carros. Sendo assim, torna-se complexo considerar as proporções da placa como forma de validação dos resultados da etapa de detecção de placas no *pipeline*;
- Padrão: No Brasil as placas podem conter algumas personalizações, sejam elas por um valor extra pago no momento do emplacamento do veículo, ou por se tratar de casos específicos como o de veículos comerciais. Exemplo disso são as placas dos táxis, com o plano de fundo ou os caracteres em vermelho. Além disso, as diferenças entre ser o padrão antigo ou o Mercosul, pois a fonte da letra dos caracteres mudam, o modelo antigo contém três letras e quatro números enquanto o Mercosul contém quatro letras e três números, e o próprio desenho da placa também muda, como pode ser observado na Figura 39. Vale ressaltar que só considerando o padrão Mercosul já há seis personalizações diferentes das placas para carros, e seis para motocicletas, como pode ser observado na Figura 40.

Considerando as complicações existentes para a utilização do ALPR em cenários reais, os trabalhos desenvolvidos por pesquisadores geralmente tendem a envolver restrições importantes como a especificação do posicionamento para alinhamento da placa com a câmera, definição de áreas específicas da imagem que o veículo deve estar, condições de iluminação, dentre outras restrições utilizadas para alcançar bons resultados. Além disso, são geralmente trabalhadas partes do *pipeline* de processamento, sendo em alguns casos apenas utilizados modelos para detecção direta da placa do veículo e realizar em sequência o reconhecimento dos caracteres (SUNG;

YU; KOREA, 2020). Neste caso, a restrição surge da necessidade de trabalhar com imagens em altas resoluções, como 3k ou 4k, que pela alta resolução possibilita aplicar um sistema de ALPR em um fluxo de imagens de veículos em alta velocidade sem a necessidade de estabelecer uma região de interesse (ROI) para evitar que objetos semelhantes a uma placa sejam erroneamente detectados.

Figura 39 – Conversão entre modelo de placa antigo e mercosul


Fonte: Saigg (2020)

Figura 40 – Personalizações disponíveis nas placas mercosul

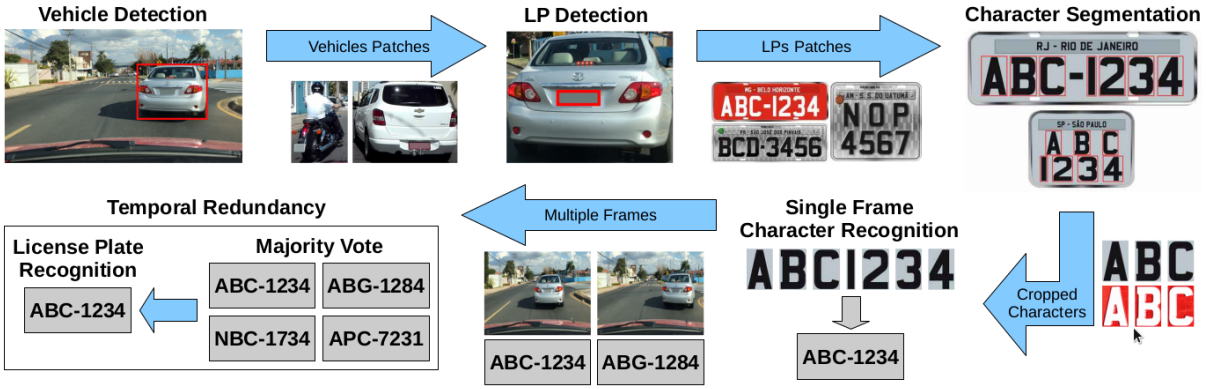

Fonte: Fonseca (2023)

No trabalho desenvolvido por Laroca et al. (2018), e que foi a principal referência no trabalho realizado por Silva (2022), trata da construção completa de um *pipeline* de detecção,

incluindo neste a criação de uma base de dados chamada UFPR-ALPR. Esta base de dados permite realizar o treinamento do modelo para a utilização do ALPR em tempo real sob condições reais de uso com o mínimo de restrições.

Para alcançar bons resultados, sobretudo quanto às detecções incorretas de placas, faz parte do *pipeline* a detecção do veículo, como pode ser observado na Figura 41. A utilização de um detector de veículos é para reduzir casos dos quais objetos similares sejam confundidos por placas, diminuindo consideravelmente os casos de falsos positivos.

Figura 41 – *Pipeline* completo para sistema de ALPR




Fonte: Laroca et al. (2018)

Além disso, foi desenvolvido o tratamento de redundância temporal que permite por meio de um sistema de votação serem determinados os caracteres que mais se repetem considerando os resultados retornados para uma mesma placa em várias imagens. Isso possibilita que seja alcançada uma melhor acurácia nos resultados para cada veículo.

## 2.3 Protocolo de Comunicação MQTT

O MQTT trata-se de um protocolo de comunicação originalmente projetado para comunicação entre máquinas (*Machine to Machine*, M2M), mas é atualmente aplicado nos mais diversos cenários de internet das coisas (*Internet of Things*, IoT). Para ser utilizado o protocolo requer uma rede subjacente como a TCP/IP, que pode fornecer via protocolo TCP uma conexão ordenada sem perdas. Porém, já existe uma versão do MQTT adaptada que possibilita usar o protocolo UDP na camada de transporte para trabalhar com dispositivos como sensores de baixa potência que enviam dados ocasionalmente (COPE, 2018b), versão essa que será descrita ao longo desta Seção.

O MQTT é hoje um dos principais protocolos para aplicações de internet das coisas devido a sua leveza por transmitir dados como vetor de bytes, além da simplicidade de utilização

se comparado, por exemplo, ao protocolo HTTP mesmo com a versão HTTP/2 permitindo uma comunicação bidirecional similar a um *WebSocket*. A vantagem do MQTT é por ser pensado especificamente para a comunicação entre máquinas, além da sua maturidade já estabelecida (SINGH, 2017), mostrando-se em testes realizados com redes 3G até 93 vezes mais rápido do que o HTTP (SEROZHENKO, 2017).

Além de ser veloz, o MQTT oferece recursos como o de qualidade de serviço (*Quality of Service*, QoS), que permite determinar a importância da entrega da mensagem transmitida, permitindo que haja controle de prioridade sobre a garantia de entrega das mensagens publicadas pelos dispositivos. Vale ressaltar que quanto maior for o nível QoS associado às mensagens mais recursos serão alocados para garantir a entrega desta, consequentemente aumenta a latência da transmissão e a largura de banda necessária.

Para fins de comparação com outros protocolos que como o MQTT utilizam a arquitetura publicador/assinante, as Tabelas 4, 5 e 6 apresentam características dos protocolos MQTT, AMQP e DDS extraídas de World (2023), Craggs (2022), Group (2023) e Platform (2018). Vale ressaltar sobre as Tabelas que o protocolo DDS funciona de forma descentralizada mediante comunicação entre pares (Peer to Peer, P2P), dispensando o uso de um servidor. Além disso, a taxa de mensagens por dispositivo pode variar a depender do dispositivo utilizado como publicador, o servidor que está intermediando a comunicação, entre outros fatores que influenciam nos resultados de testes.

Tabela 4 – Características do protocolo MQTT

| | **MQTT** |
|---|---|
| **Nome do protocolo** | *Message Queuing Telemetry Transport* |
| **Arquitetura** | Publicador/assinante |
| **Estrutura** | Tópicos |
| **Protocolo de transporte** | TCP |
| **Segurança** | TLS e/ou autenticação por usuário e senha (possível suporte SASL) |
| **Observabilidade do cliente** | Estado da conexão é conhecido |
| **Modo de mensagens** | Assíncrono |
| **Fila de mensagens** | Utilizada quando um assinante perde conexão |
| **Tamanho de mensagem** | 256 MB |
| **Tipo de conteúdo** | Qualquer tipo |
| **Taxa de mensagens por dispositivo** | 45 mensagens por segundo |

Fonte: Autor.

Tabela 5 – Características do protocolo AMQP

| | **AMQP** |
|---|---|
| **Nome do protocolo** | *Advanced Message Queuing Protocol* |
| **Arquitetura** | Publicador/assinante |
| **Estrutura** | Exchanges e filas |
| **Protocolo de transporte** | TCP |
| **Segurança** | TLS e/ou autenticação por usuário e senha (possível suporte SASL) |
| **Observabilidade do cliente** | Estado da conexão é desconhecido |
| **Modo de mensagens** | Síncrono e assíncrono |
| **Fila de mensagens** | Utilizada como recurso principal |
| **Tamanho de mensagem** | 128 MB recomendado e 2 GB teórico |
| **Tipo de conteúdo** | Qualquer tipo |
| **Taxa de mensagens por dispositivo** | - |

Fonte: Autor.

Tabela 6 – Características do protocolo DDS

| | **DDS** |
|---|---|
| **Nome do protocolo** | *Data Distribution Service* |
| **Arquitetura** | Publicador/assinante |
| **Estrutura** | Tópicos |
| **Protocolo de transporte** | UDP |
| **Segurança** | Autenticação, controle de acesso, confidencialidade e integridade |
| **Observabilidade do cliente** | Descobre dinamicamente os dispositivos |
| **Modo de mensagens** | Assíncrono |
| **Fila de mensagens** | - |
| **Tamanho de mensagem** | Não há limite definido pois mensagens grandes podem ser fragmentadas |
| **Tipo de conteúdo** | Qualquer tipo |
| **Taxa de mensagens por dispositivo** | Até 10.000 mensagens/segundo |

Fonte: Autor.

Além disso, diferente da arquitetura de comunicação requisição-resposta (*request-response*) utilizada pelo HTTP, o MQTT implementa a arquitetura publicador/assinante (*publisher/subscriber*), e com esta arquitetura é opcionalmente associado um *broker*, que atua como um servidor que centraliza a comunicação entre publicadores e assinantes por meio de tópicos. O servidor se encarrega de encaminhar mensagens recebidas para nenhum ou vários dispositivos assinantes de um tópico, dispensando do dispositivo publicador o papel de conhecer e estabelecer conexão diretamente com cada um destes dispositivos assinantes que estão consumindo as mensagens.

Ou seja, considerando um cenário no qual um sensor deve enviar informações de temperatura para outros três dispositivos, primeiro é estabelecida a conexão entre o sensor e cada um destes dispositivos para então as mensagens serem enviadas.

Já com a utilização do servidor junto a arquitetura publicador/assinante o sensor estabelece a conexão com este servidor apenas uma vez e continuamente envia mensagens contendo a temperatura atual registrada. Então o servidor é quem se encarrega de direcionar esta mensagem para os outros três dispositivos que dependem da informação.

No decorrer desta seção o protocolo será apresentado em mais detalhes, onde serão melhor definidos os termos utilizados nesta breve descrição, e tantos outros que o compõem.

### 2.3.1 Histórico

O MQTT foi criado em 1999 por Andy Stanford Clark da IBM e Arlen Nipper da Arcom Systems com o propósito de conectar sistemas de telemetria de oleodutos via satélite (COPE, 2018a). Isso explica a presença de alguns recursos como o de QoS citado anteriormente, tanto entre publicador e servidor, como entre servidor e assinante, já que o protocolo foi pensado para atuar sob condições extremas com uma comunicação satisfatória.

Mesmo que tenha surgido como um protocolo particular da IBM, em 2010 foi liberado para uso gratuito. Mas a padronização só foi ocorrer em 2014, através da OASIS.

Devido às vantagens do protocolo este rapidamente passou a ser padronizado para as aplicações IoT, sendo a seguir apresentadas as principais versões do protocolo na atualidade.

- **MQTT v3.1.0**: Versão na qual a IBM em 2013 submeteu o protocolo[2] ao organismo de especificação da OASIS[3];
- **MQTT v3.1.1**: Atualmente está em pleno uso e trata-se da versão 4 do protocolo por existir uma não correspondência entre a numeração da versão do protocolo declarada no mercado e a apresentada no pacote para as aplicações. Esta não correspondência das versões declaradas surgiu devido apenas um byte ser destinado para a identificação do protocolo, sendo a versão 3.1.0 indicada no pacote como versão 3, e no caso da versão 3.1.1 é indicada como versão 4 (CRAGGS, 2016);
- **MQTT v5**: Atualmente está em uso limitado e foi a partir desta que a versão declarada do protocolo passou a ser igual a visível pelas aplicações.

Além destas versões que trabalham com o protocolo TCP na camada de transporte também há uma versão que trabalha com o protocolo UDP, sendo apresentada a seguir na Subseção 2.3.1.1.

[2] Disponível em: <https://lists.oasis-open.org/archives/mqtt/201302/msg00000.html>. Acesso em agosto de 2022
[3] Disponível em: <https://www.oasis-open.org/>. Acesso em maio de 2023

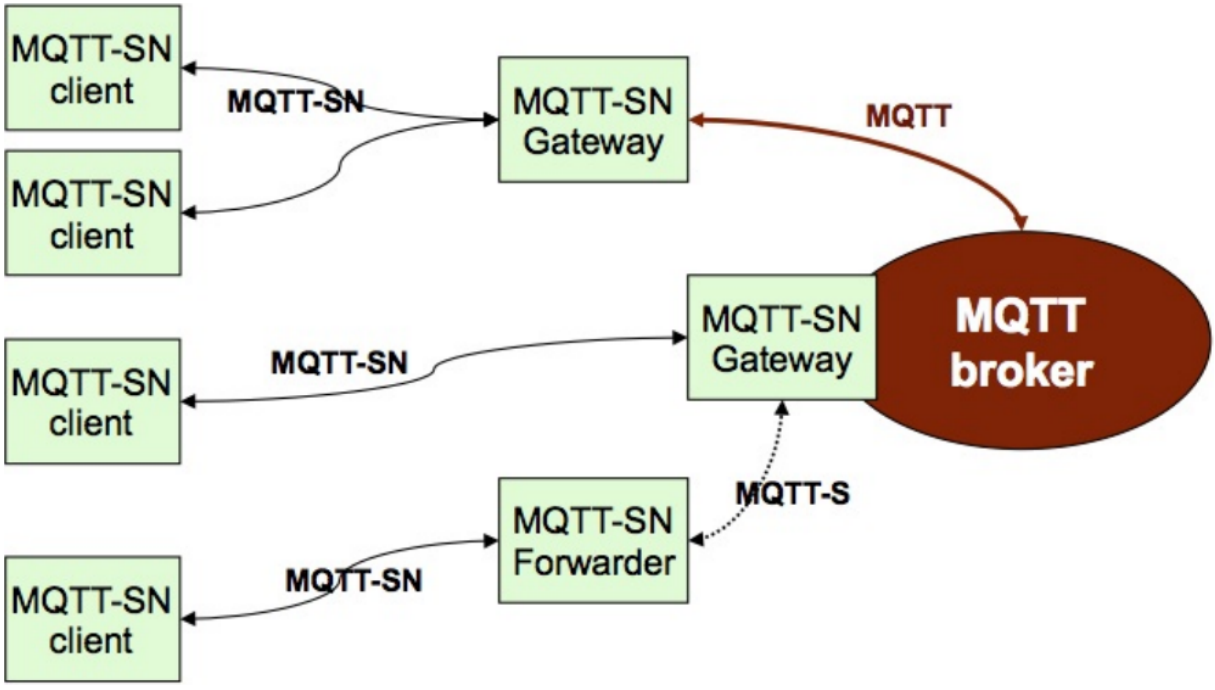
#### 2.3.1.1 MQTT-SN

Surgiu da necessidade da utilização de uma comunicação centrada em dados para redes de sensores sem fio, dos quais devido à baixa capacidade de armazenamento, processamento e energética não suporta a utilização do MQTT com o protocolo de transporte sendo o TCP, responsável por entregar pacotes em uma sequência ordenada e sem perdas, o que o torna pesado e dispensável para estes cenários.

Devido o MQTT ser centrado em dados foram realizadas algumas adaptações para as peculiaridades de um ambiente de comunicação sem fio, sendo possível utilizar sobre camadas APS do ZigBee (STANFORD-CLARK; TRUONG, 2013).

As diferenças arquiteturais surgem devido o MQTT-SN ter nós de *gateway* e *forwarder*, como pode ser observado na Figura 42. O *gateway* é um nó que atua entre os nós dos clientes e o servidor, enquanto o *forwarder* é utilizado quando é necessário fazer um encapsulamento do pacote enviado pelo sensor cliente para o nó *gateway*, que enviará o pacote para o servidor (SOUZA, 2018). A seguir, é apresentado o Quadro 1 com as principais diferenças entre o MQTT e o MQTT-SN.

Figura 42 – Comunicação via protocolo MQTT-SN




Fonte: Documento de especificação do MQTT-SN

Quadro 1 – Diferenças entre MQTT e MQTT SN

| **Especificação** | **MQTT** | **MQTT Sn** |
|---|---|---|
| Protocolo de transporte | TCP | UDP |
| Rede | TCP-IP | Zigbee |
| Tamanho mínimo de mensagem | 2 bytes – PING | 1 byte |
| Tamanho máximo de mensagem | <= 256 MB | < 128 bytes |
| Alimentação por bateria | - | SIM |
| Clientes em modo Sleep | - | SIM |
| *Connectionless mode* (QoS -1) | - | SIM |
| Tópico ID | Não. O tópico pode ser nomeado | Sim, utilizando apenas 2 bytes, economizando processamento |

Fonte: Souza (2018)

### 2.3.2 Conceitos

Será apresentado nesta Subseção como o protocolo MQTT é composto por alguns conceitos básicos discutidos anteriormente, e alguns outros, estejam bem definidos. A seguir os termos são descritos:

- Publicador (*Publisher*): Se trata do cliente que publica pacote de dados para um ou vários tópicos, bastando que o dispositivo esteja em uma sessão ativa com o servidor para desempenhar esta função. É possível também atuar em paralelo como assinante de um ou vários tópicos;
- Assinante (*Subscriber*): É o cliente que recebe pacotes de dados do tópico do qual é assinante;
- Mensagem (*Message*): Trata-se do pacote de dados trafegado entre os clientes e o servidor. Há um limite no protocolo quanto ao tamanho da mensagem de 268.435.455 bytes;
- Tópico (*Topic*): Usado para endereçar as mensagens, o tópico pode ser considerado um repositório de mensagens, sendo acessível aos clientes devidamente conectados ao servidor e não armazena histórico. Ou seja, não há uma linha do tempo das mensagens enviadas e recebidas, sendo descartadas uma vez que todos os clientes que são assinantes deste tópico as recebem;
- Sessão (*Session*): É o termo dado para a conexão permanentemente ativa entre um cliente e o servidor, permitindo que haja troca de mensagens entre eles sem a necessidade de algum deles realizar uma requisição para isto. Vale ressaltar que caso seja desejado é

possível estabelecer uma conexão de sessão limpa, então qualquer mensagem que estivesse armazenada em *buffer* como pendente é descartada. E quanto ao *buffer*, há um limite de 5000 mensagens armazenadas para cada cliente associado ao tópico;

- Manter Vivo (*Keep Alive*): No momento da conexão o cliente pode determinar o tempo máximo, em segundos, que poderá ficar sem se comunicar com o servidor. Se o cliente passou desse tempo sem enviar mensagem para algum tópico deve ser enviada uma mensagem de *ping* para indicar que continua conectado, e também verificar se o servidor está disponível (HIVEMQ, 2015a). Quando é percebido que o cliente não enviou mensagens durante o intervalo de tempo definido, e também não enviou mensagem de *ping* após este intervalo de tempo, o servidor determina que aquele cliente perdeu a conexão. Caso tenha perdido a conexão, o servidor encerra a sessão com o cliente e envia a mensagem de última vontade deste para o tópico ao qual esta mensagem está endereçada.

Além desses termos são apresentados nas Subseções a seguir alguns outros também importantes.

#### 2.3.2.1 Servidor (*Broker*)

No MQTT é usado para atuar como intermediário da comunicação entre o(s) publicador(es) e o(s) assinante(s). Ou seja, um dos principais papéis do servidor é armazenar parcialmente as mensagens e distribuí-las.

Para um cliente se conectar ao servidor não é obrigatório utilizarem alguma forma de identificação, seja para publicar ou assinar algum tópico, podendo realizar uma conexão anônima. No entanto, pode ser configurado no servidor um nome de usuário e senha para que só dispositivos que tenham essas credenciais tenham a conexão estabelecida. Ainda assim, os dispositivos podem realizar uma conexão anônima, mas necessitam de autenticação para iniciar uma sessão com o servidor.

Vale ressaltar que para a utilização de alguns recursos como os de QoS e a persistência de dados, que serve para o servidor guardar informações como os tópicos que o cliente é assinante, é necessário que este cliente tenha alguma identificação única no servidor, sendo necessário no momento da conexão indicar um nome ao invés de realizar a conexão anônima.

#### 2.3.2.2 Qualidade de Serviço (*Quality of Service*)

Define a prioridade desejada da garantia de entrega da mensagem publicada. Existem três níveis de QoS comuns tanto no protocolo MQTT nas versões que utilizam o protocolo TCP como na versão MQTT-SN. Porém, esta última tem um nível extra de QoS.

Os níveis em comum entre as versões são:

a) QoS 0 indica que a mensagem pode ser enviada e não há necessidade de garantia da sua entrega. É o nível indicado no caso de dados que ao serem perdidos não causam prejuízos para as aplicações que dependem destas informações;

b) QoS 1 indica que deverá haver uma garantia que a mensagem chegou aos assinantes do tópico pelo menos uma vez, sendo possível então a mensagem ser publicada várias vezes no tópico por possível atraso da mensagem de confirmação de entrega do pacote ao publicador;

c) QoS 2 garante que a mensagem publicada será entregue exatamente uma vez aos assinantes.

Já no MQTT-SN há o nível -1 ou 3, como apresentado pela especificação modo sem conexão (*connectionless mode*) no Quadro 1, indicando que a mensagem deve ser publicada sem o estabelecimento prévio de uma conexão com o servidor, e neste caso é requerido nome curto para o tópico, ou um simples identificador.

#### 2.3.2.3 Retenção (*Retain*)

A retenção é indicada por uma *flag* associada a mensagem no momento da publicação em um tópico. Ela permite indicar se esta mensagem deve permanecer disponível para recebimento após o servidor encaminhá-la para todos os clientes atualmente assinantes do tópico.

Ou seja, quando a *flag* está inativa na mensagem, uma vez que o servidor encaminhou esta para todos os clientes a mensagem é descartada. Então, se algum outro dispositivo assinar o tópico após este momento ele só receberá alguma mensagem quando ocorrer uma nova publicação no tópico.

Porém, se esta *flag* estiver ativa, este último dispositivo receberá a mensagem mesmo estabelecendo a assinatura no tópico após todos os clientes já assinados terem recebido esta. Sendo assim, a mensagem fica disponível para novas assinaturas permanentemente, sendo substituída ou desabilitada apenas após ser realizada uma nova publicação no tópico também com a *flag* ativa.

Vale ressaltar que caso uma nova publicação seja feita com uma mensagem sem conteúdo e com a *flag* habilitada, a mensagem retida será removida e novos clientes não receberão nenhuma mensagem automaticamente. Por outro lado, se uma nova mensagem recebida com a *flag* ativa contiver algum conteúdo, a anterior é descartada e esta última fica disponível para os clientes receberem ao assinarem o tópico.

#### 2.3.2.4 Testamento (*Last Will And Testament*)

O testamento é um recurso oferecido pelo servidor que permite a todos os clientes definirem uma mensagem de última vontade no momento da conexão. Esta mensagem é definida como qualquer outra que pode ser publicada, permitindo ser definida a *flag* de retenção, definir QoS, tópico que deverá ser enviada e o conteúdo (HIVEMQ, 2015b). A mensagem é encaminhada

para todos os assinantes do tópico predefinido caso o dispositivo cliente perca abruptamente a conexão com o servidor.

Vale ressaltar que quando a desconexão ocorre por meio de troca de mensagens conforme a vontade do dispositivo de desconectar-se do servidor, o testamento é descartado. Ou seja, a mensagem de testamento só é enviada em casos do cliente perder a conexão com a internet, ou ocorra algum problema técnico que resulte no travamento ou desligamento do dispositivo, levando então ao rompimento da conexão entre o cliente e o servidor.

### 2.3.3 Segurança

Além da possibilidade de configurar credenciais para autenticar a conexão, o servidor disponibiliza outros recursos de segurança que podem ser utilizados para restringir acesso às informações, ou garantir uma maior segurança acerca do tráfego do conteúdo como o uso do TLS (*Transport Layer Security*) ou SSL (*Secure Sockets Layer*).

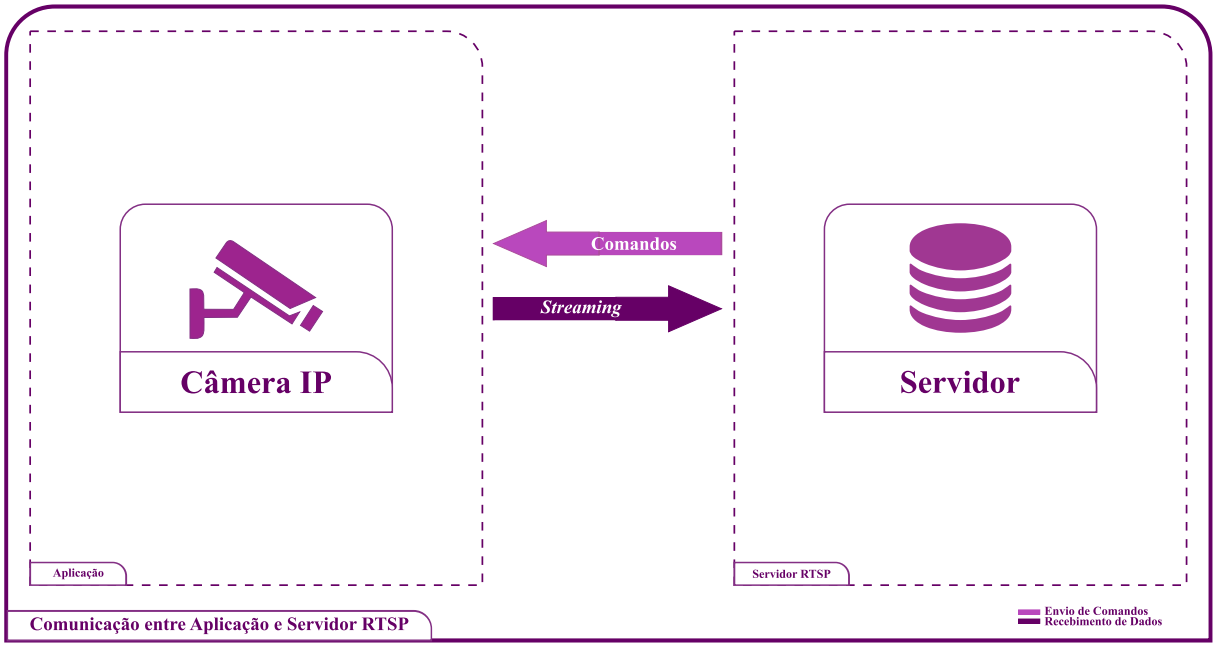
Também é possível trabalhar com a criptografia em conjunto com estes recursos, podendo ser aplicada na mensagem sendo responsabilidade das aplicações dos clientes que usam o protocolo MQTT empregar técnicas de criptografar e descriptografar a mensagem;

## 2.4 Protocolo de Comunicação RTSP

O protocolo de *streaming* em tempo real (*Real-Time Streaming Protocol*, RTSP) trata-se de um protocolo de rede da camada de aplicação que atua em tempo real entre o *streaming* de áudio e vídeo, e o acesso ao conteúdo por parte dos usuários, como ilustrado por meio de um exemplo com câmera IP na Figura 43. A vantagem de utilizar este protocolo é a possibilidade de acesso ao conteúdo de mídias pela internet com interações como reproduzir, pausar, avançar e retroceder o conteúdo, partindo disso a expressão usada “controlado como um Videocassete”, sem necessidade de ter que baixar e armazenar a mídia no dispositivo físico para isso (DEMIR, 2021). Vale ressaltar que apesar de ser similar, o RTSP é incompatível com o HTTP, além de não ser suportado diretamente nos navegadores de internet, sendo necessário incorporar aplicações extras para conseguir fazer com que o conteúdo seja renderizado em uma página web, como por meio da API de código aberto WebRTC[4]. E quanto ao servidor destinado ao protocolo RTSP, este tem que ser totalmente dedicado, levando a dificuldades com escalabilidade (AMARAL, 2022).

[4] Disponível em: <https://webrtc.org/?hl=pt-br>. Acesso em março de 2023

Figura 43 – Comunicação entre aplicação e servidor RTSP




Fonte: Autor.

O protocolo surgiu a partir da experiência de *streaming* da RealNetworks através do RealAudio, depois renomeado para RealPlayer, que foi a primeira plataforma de áudio para internet lançada em 1995 (REDAçãO, 2021). E também mediante experiência da LiveMedia da Netscape, que quando estava para ser desenvolvida com a tecnologia e equipe da InSoft (adquirida pela Netscape) tinha o objetivo de possibilitar usuários terem acesso facilitado a áudio e vídeo *on-demand*, videoconferência em tempo real e telefonia por internet (CORPORATION, 1996). Vale ressaltar que nesse projeto já era considerado ser utilizado o protocolo de transporte em tempo real (*Real-time Transport Protocol*, RTP), sendo tradicionalmente utilizado pelos servidores de RTSP (EMEZI, 2022). Então, foi por meio de uma parceria entre as duas empresas e a Universidade Columbia que o RTSP surgiu entre 1996 e 1997, sendo padronizado em 1998 (RFC 2326).

A forte conexão estabelecida pelo protocolo entre cliente e servidor, e seus recursos de gerenciamento de *streaming*, fizeram este rapidamente ser utilizado por diversas aplicações. Das mundialmente conhecidas atualmente o protocolo já foi adotado em plataformas como YouTube, Spotify, Skype e reprodutores de mídia como o VLC. Contudo, a usabilidade deste na área de CFTV recebe destaque, sendo adotado por várias empresas como a Intelbras[5] e a Hikvision[6]

Quanto ao seu funcionamento, este é adjacente aos protocolos tanto TCP como UDP. Através do protocolo TCP é transmitido comandos para gerenciar o conteúdo visualizado,

[5] Disponível em: <https://www.intelbras.com/pt-br/>. Acesso em maio de 2023
[6] Disponível em: <https://www.hikvision.com/pt-br/>. Acesso em maio de 2023

enquanto o protocolo UDP atua na transmissão do conteúdo em si.

# 3
# Trabalhos Relacionados

Para realização das pesquisas dos produtos foi necessário primeiramente estabelecer definições claras de critérios para servir como filtros na busca, possibilitando restringir quais produtos tendem a apresentar soluções similares às requeridas pelo produto desenvolvido neste trabalho.

Considerando isso, a seguir são apresentados estes critérios, e logo mais os produtos encontrados no mercado.

## 3.1 Critérios de Pesquisa

O principal critério que define a similaridade é o produto dispor de recursos inteligentes com base na proposta do reconhecimento dos caracteres das placas dos veículos. Porém, este produto deve estar em um contexto de interação com dispositivos em borda ou servidor em nuvem, não sendo admitidos produtos que se tratam apenas de uma API. Também são admitidos produtos com o processamento do ALPR feito diretamente no dispositivo de borda, considerado então casos em que o próprio *firmware* do dispositivo que compõe a câmera consegue realizar o processamento das imagens por meio de modelo(s) treinado(s) de redes neurais.

Além disso, é esperado que os dados resultantes do processamento possam ser acessados pelo próprio usuário do produto, seja por meio de relatórios gerados e salvos no próprio dispositivo, ou interfaces externas, como por meio de *dashboards*.

## 3.2 Produtos

Definidos os critérios de pesquisa, foram buscados no mercado (inter)nacional produtos similares ao desenvolvido neste trabalho. A seguir são apresentados alguns destes produtos, e os seus respectivos recursos.

### 3.2.1 Intelli-Vision Smart City/Security

O Smart City/Security[1] é um dos produtos oferecidos pela empresa Intelli-Vision localizada em San Jose na Califórnia, a qual tem por objetivo oferecer funcionalidades de segurança e vigilância tanto para o governo como para empresas através da análise inteligente de vídeo com o uso de modelos treinados através do aprendizado profundo, sendo dispensável a necessidade de supervisão humana em alguns cenários de vigilância. Além disso, é considerada uma das suas principais características a redução de alarmes falsos já que dispensam sensores como o PIR, os quais são geralmente utilizados para detecção de movimentos, substituídos então pelos modelos treinados.

A seguir, são apresentadas as funcionalidades do produto:

a) detecção de movimento através da análise de vídeo;

b) detecção de violação da câmera;

c) detecção de intrusão;

d) passagem incorreta em linhas ou vias;

e) detecção de objetos deixados ou removidos;

f) detecção de multidões;

g) detecção de desordens;

h) contagem de veículos ou pessoas;

i) reconhecimento facial;

j) reconhecimento de placas de veículos ALPR (ou ANPR);

k) busca e resumo de vídeos;

l) processamento pode acontecer diretamente no dispositivo, em um servidor local ou em nuvem;

m) detecção de quedas e coações.

### 3.2.2 Intelbras VIP 7250 LPR IA FT

A VIP 7250 LPR IA FT[2] trata-se de uma câmera IP com resolução de 2 MP da Intelbras cujo objetivo é realizar a leitura automática de placas de veículos em entradas e saídas de áreas restritas e ruas urbanas. Porém, tem a especificidade de dispor de uma assertividade superior à 90% apenas se o veículo estiver em uma velocidade inferior a 50 km/h.

[1] Disponível em: <https://www.intelli-vision.com/ai-video-analytics-smart-city-security/>. Acesso em julho de 2022

[2] Disponível em: <https://www.intelbras.com/pt-br/camera-ip-com-leitura-automatica-de-placas-vip-7250-lpr-ia-ft>. Acesso em julho de 2022

Figura 44 – Intelbras VIP 7250 LPR IA FT


Fonte: Intelbras

O processamento inteligente ocorre diretamente no dispositivo da câmera, permitindo o armazenamento dos dados resultantes no próprio dispositivo através do uso de um cartão micro-SD. Mas também dispõe de uma interface para salvar os dados localmente em um computador.

Com base nas especificações do dispositivo são apresentadas a seguir as descrições de algumas funcionalidades:

a) leitura automática de placas de até um veículo por imagem;

b) identificação de cor e marca de veículos;

c) geração de relatórios;

d) entrada e saída de alarme;

e) suporte para cartão micro-SD de até 128 GB;

f) possibilita cadastro de uma lista de placas para liberação ou bloqueio de acesso de veículos;

g) contém a possibilidade de gerar alerta ao identificar veículos suspeitos;

h) assertividade de captura de placa superior à 95%;

i) assertividade de leitura correta superior à 90%;

j) oferece uma interface web para gerenciamento;

k) possui certificação de proteção IP67, que garante um dispositivo à prova de poeira e protegido contra a imersão em água de até 1 metro de profundidade por 30 minutos.

### 3.2.3 Genetec

A Genetec trata-se de uma empresa localizada em Montreal, na província de Quebec no Canadá. Esta empresa desenvolve soluções inteligentes voltadas para a área de visão computacional, sendo uma de suas principais propostas a integração dos produtos, oferecendo o serviço unificado aos consumidores mediante um sistema nomeado de Security Center. Nas Subseções 3.2.3.1 e 3.2.3.2 são apresentados dois produtos voltados para o reconhecimento de placas de veículos.

#### 3.2.3.1 AutoVu SharpV

O dispositivo AutoVu SharpV[3] trata-se de um produto voltado para ALPR considerando um posicionamento fixo da câmera, a qual já realiza o reconhecimento. Sendo assim, todo o processamento e análise das imagens via redes neurais é realizado na própria unidade, não sendo necessário o uso de servidores ou serviços em nuvem para tal.

Figura 45 – Genetec AutoVu SharpV

Fonte: Genetec

Algumas outras características encontradas nas especificações técnicas são apresentadas a seguir:

a) o sensor da câmera de ALPR tem resolução de 1280x960 pixels a uma taxa de 30 quadros por segundo;

b) alcance padrão de captura de 3 à 18,25 metros;

c) oferece comprimentos de onda infravermelho de 940 nm, 850 nm, 740 nm e 590 nm;

d) possui certificação de proteção IP66 e IP67, sendo a diferença a IP66 garantir um dispositivo à prova de poeira e proteção contra jatos d'água, enquanto a IP67 garante

[3] Disponível em: <https://www.genetec.com/binaries/content/assets/genetec-pt/recursos/especificacao-de-produtos/productspecs_pt_autovu-sharpv_web.pdf>. Acesso em julho de 2022

um dispositivo à prova de poeira e protegido contra a imersão em água de até 1 metro de profundidade 30 minutos;

e) transmissão de vídeo com as especificações H.264 obtendo taxa de até 30 quadros por segundo, e MJPEG obtendo taxa de até 15 quadros por segundo.

Vale ressaltar que estas informações são baseadas na versão em português das especificações técnicas do produto. Na versão em inglês algumas especificações são melhores, como a resolução da imagem do sensor da câmera indicada como 1920x1200 pixels com uma taxa de 30 quadros por segundo, e a possibilidade de estabelecer conexão mediante uma rede de dados 4G/LTE.

#### 3.2.3.2 AutoVu SharpZ3

O AutoVu SharpZ3[4] trata-se de uma versão mais robusta da apresentada anteriormente. Esta dispõe de recursos para trabalhar bem em contextos onde há movimentação da câmera, como ao ser utilizada em um veículo.

Figura 46 – Genetec AutoVu SharpZ3

Fonte: Genetec

Dentre os recursos melhorados estão a quantidade e qualidade das portas de entrada e saída, um aumento na quantidade de lentes instaladas no dispositivo, dentre outros apresentados a seguir:

a) o sensor da câmera ALPR tem resolução de 1456x1088 pixels a uma taxa de 30 quadros por segundo;

b) alcance de até 19 metros com placas retrorrefletoras;

c) análises de reconhecimento de tipo de veículo, cor e origem da placa;

d) contém três sensores ópticos;

[4] Disponível em: <https://resources.genetec.com/en-datasheets/autovu-sharpz3>. Acesso em julho de 2022

e) posicionamento GPS avançado com identificação que o dispositivo está parado;

f) até 4 câmeras ALPR de alta definição na mesma unidade do produto;

g) lentes de 8mm, 12mm, 16mm e 25mm;

h) possibilidade de unificação com o sistema Security Center da Genetec;

i) conta com o processador Intel Atom Processor E3950 e um co-processador adicional dedicado ao aprendizado de máquina;

j) oferece comprimentos de onda de iluminação infravermelho de 940 nm, 850 nm, 740 nm e 590 nm;

k) possui certificação de proteção IP66 e IP67.

### 3.2.4 Hikvision

A Hikvision[5] é uma empresa chinesa voltada para o desenvolvimento de equipamentos para vigilância por vídeo, tanto para uso civil como militares. Segundo dados de pesquisa da IHS Markit[6], em 2015 a empresa alcançou 19,5% de participação de mercado na indústria global de vigilância por vídeo. Além disso, foi classificada como líder mundial por cinco anos consecutivos de participação de mercado em equipamentos de vigilância por vídeo.

#### 3.2.4.1 DS-TCG205-B e DS-TCG227

A câmera DS-TCG205-B[7] é um dos produtos oferecidos para ANPR (*Automatic Number Plate Recognition*), a qual pode reconhecer o tipo de veículo, como SUV, MPV e Pickup, caminhão, ônibus e van, além de reconhecer a cor do veículo.

[5] Disponível em: <https://www.hikvision.com/pt-br/>. Acesso em maio de 2023

[6] Disponível em: <https://us.hikvision.com/en/about/hikvision-global>. Acesso em abril de 2023

[7] Disponível em: <https://www.hikvision.com/pt-br/products/ITS-Products/Entrance---Exit-Management/Entrance---Exit-Management/ds-tcg205-b-/>. Acesso em abril de 2023

Figura 47 – Câmera ANPR da Hikvision


Fonte: Hikvision

A seguir são apresentadas algumas características do produto:

a) a câmera de 2 MP tem resolução máxima de 1920×1200 pixels a uma taxa de 25 ou 30 quadros por segundo;

b) sensor da câmera com abertura de 1/2,8";

c) suporta lentes de 2,8 mm à 12 mm;

d) oferece suporte à iluminação infravermelho;

e) compatível com os protocolos TCP, HTTP, DHCP, DNS, RTP, RTSP, NTP e FTP;

f) suporta identificação de veículos sem placa;

g) taxa de captura de placas maior do que 99,5% quando a velocidade do veículo é no máximo 60 km/h;

h) suporta até 30000 veículos entre a lista branca, que permite acesso aos veículos, e a lista de bloqueio;

i) possui certificação de proteção IP67 e IK10;

j) possibilita o uso de cartões de memória SD/SDHC.

Já a câmera DS-TCG227[8] compartilha de vários recursos similares à câmera DS-TCG205-B, sendo apresentadas a seguir as principais diferenças:

a) suporta lentes de 3,1mm à 9mm;

b) a câmera dispõe apenas de uma taxa de 25 quadros por segundo;

c) suporta apenas cartão de memória TF de até 128GB.

#### 3.2.4.2 DS-TCG227-A e DS-TCG405-E

A câmera DS-TCG227-A[9] se trata de uma opção mais robusta, capaz de lidar com placas sujas, passagem rápida de vários veículos, além de disponibilizar controle *offline*. Ela é similar à câmera DS-TCG227, compartilhando das mesmas diferenças da câmera DS-TCG205-B.

A seguir são apresentados outros recursos disponibilizados:

a) suporta retransmissão para controle da barreira do veículo;

b) suporta múltiplos quadros de LPR;

c) alcança acurácia maior que 96% para reconhecimento da direção da movimentação do veículo;

d) a acurácia de reconhecimento de placa é maior do que 98%;

Por outro lado, a câmera DS-TCG405-E[10] difere da DS-TCG227-A ao ponto no qual as imagens são de 4MP, o sensor tem abertura de 1/3", a resolução máxima é de 2688×1520 pixels com vídeo em tempo real, e as lentes vão de 3,1 mm à 6 mm.

## 3.3 Comparação dos Produtos

Para facilitar a visualização das semelhanças e diferenças entre os produtos encontrados no mercado e o desenvolvido neste trabalho (AtalaIA), são apresentadas através das Tabelas 7, 8 e 9 alguns recursos comparativos. Vale ressaltar quanto aos custos relacionados ao produto AtalaIA, pois não foi considerado o preço do sistema de ALPR para determinar o valor apresentado. Então, os valores indicados são de equipamentos físicos e de servidor em nuvem para dar suporte ao funcionamento do produto.

Ainda sobre os custos apresentados pelo produto AtalaIA, na sequência dos valores apresentados na Tabela são consideradas a configuração com a câmera IP Intelbras VIP 3230 B SL G2 trabalhando junto ao servidor Ampere A1 Compute com as especificações apresentadas

[8] Disponível em: <https://www.hikvision.com/pt-br/products/ITS-Products/Entrance---Exit-Management/Entrance---Exit-Management/DS-TCG227/>. Acesso em abril de 2023

[9] Disponível em: <https://www.hikvision.com/pt-br/products/ITS-Products/Entrance---Exit-Management/Entrance---Exit-Management/ds-tcg227-a/>. Acesso em abril de 2023

[10] Disponível em: <https://www.hikvision.com/pt-br/products/ITS-Products/Entrance---Exit-Management/Entrance---Exit-Management/ds-tcg405-e/>. Acesso em abril de 2023

na Subseção 4.3.5 (24 GB de RAM e 4 ocpus), uma configuração com a câmera IP construída neste trabalho com o Esp32Cam junto ao servidor Ampere A1 Compute, e por último uma configuração com o Jetson Nano registrando as imagens e as processando localmente.

Tabela 7 – Comparação entre os produtos AtalaIA, Smart City/Security e VIP 7250 LPR IA FT

| | **AtalaIA** | **Smart City/Security** | **VIP 7250 LPR IA FT** |
|---|---|---|---|
| **Contador de veículos** | Sim | Sim | Não especificado |
| **Reconhecedor de cor veicular** | Não | Não especificado | Sim |
| **Distingue o tipo de veículo** | Sim | Não especificado | Não especificado |
| **Reconhece a marca do veículo** | Não | Não especificado | Sim |
| **Escalável** | Sim | Sim | Não |
| **Processamento em nuvem** | Sim | Sim | Não |
| **Processamento na borda** | Sim | Sim | Sim |
| **Possibilita integração com mecanismo de automação** | Sim | Sim | Não especificado |
| **Identifica violação à câmera** | Não | Sim | Não |
| **Compatibilidade de uso com câmeras IP avulsas do mercado** | Sim | Não especificado | Não |
| **Possibilidade de trabalhar com câmeras analógicas** | Sim | Não especificado | Não |
| **Preço no mercado (ou custo)** | R$ 888,00<br>R$ 328,00<br>R$ 3.000,00 | Não disponível publicamente | R$ 9.100,00 |

Tabela 8 – Comparação entre os produtos AutoVu SharpV, AutoVu SharpZ3 e DS-TCG-205-B

| | **AutoVu SharpV** | **AutoVu SharpZ3** | **DS-TCG-205-B** |
|---|---|---|---|
| **Contador de veículos** | Não especificado | Não especificado | Não especificado |
| **Reconhecedor de cor veicular** | Não especificado | Sim | Sim |
| **Distingue o tipo de veículo** | Não especificado | Sim | Sim |
| **Reconhece a marca do veículo** | Não especificado | Não especificado | Não especificado |
| **Escalável** | Não | Não | Não |
| **Processamento em nuvem** | Não | Não | Não |
| **Processamento na borda** | Sim | Sim | Sim |
| **Possibilita integração com mecanismo de automação** | Não especificado | Não especificado | Não especificado |
| **Identifica violação à câmera** | Não | Não | Não especificado |
| **Compatibilidade de uso com câmeras IP avulsas do mercado** | Não | Não | Não |
| **Possibilidade de trabalhar com câmeras analógicas** | Não | Não | Não |
| **Preço no mercado (ou custo)** | Não disponível publicamente | Não disponível publicamente | Não encontrado |

Tabela 9 – Comparação entre os produtos DS-TCG227, DS-TCG227-A e DS-TCG405-E

| | **DS-TCG227** | **DS-TCG227-A** | **DS-TCG405-E** |
|---|---|---|---|
| **Contador de veículos** | Não especificado | Não especificado | Não especificado |
| **Reconhecedor de cor veicular** | Sim | Sim | Sim |
| **Distingue o tipo de veículo** | Sim | Sim | Sim |
| **Reconhece a marca do veículo** | Não especificado | Não especificado | Não especificado |
| **Escalável** | Não | Não | Não |
| **Processamento em nuvem** | Não | Não | Não |
| **Processamento na borda** | Sim | Sim | Sim |
| **Possibilita integração com mecanismo de automação** | Não especificado | Sim | Sim |
| **Identifica violação à câmera** | Não especificado | Não especificado | Não especificado |
| **Compatibilidade de uso com câmeras IP avulsas do mercado** | Não | Não | Não |
| **Possibilidade de trabalhar com câmeras analógicas** | Não | Não | Não |
| **Preço no mercado (ou custo)** | Não encontrado | Não encontrado | R$ 8.733,76 |

# 4

# Desenvolvimento

Considerando o exemplo introdutório de um veículo não autorizado realizando uma tentativa de acesso a um condomínio ou shopping center, o sistema inteligente proposto neste trabalho possui potencial de realizar automaticamente uma consulta em uma base de dados e indicar ao porteiro que este veículo em questão está rotulado como roubado, ou que não está associado a nenhum morador do condomínio por meio da cadeia de caracteres da placa retornada pelo ALPR. Além disso, o sistema de reconhecimento de placas veiculares pode incorporar novas funcionalidades, como a inferência da velocidade, útil em ambientes internos de condomínios, como também pode ser realizada a contagem de veículos, o controle de automação em portarias, entre outras.

Foram desenvolvidas variadas configurações de utilização do sistema considerando diferentes protocolos e câmeras, além do aprimoramento do sistema de ALPR visando o amadurecimento deste para a construção de um produto para a área de segurança de portarias, o qual consiste do uso de dispositivos de sistemas embarcados ou servidores em nuvem contendo os modelos treinados de redes neurais convolucionais. Estas unidades de processamento são conectadas a câmeras IP ou gravadores digitais de vídeo (*Digital Video Recorder*, DVR) que dispõem de canais para comunicação via internet mediante protocolo RTSP, sendo recebidas as imagens em tempo real no sistema e executado o processo de reconhecimento dos caracteres da placa.

Quanto ao próprio aperfeiçoamento do sistema, foi desenvolvido um módulo de rastreio de placas e realizado um trabalho de compressão com o modelo YOLOv5 Nano destinado à detecção das placas visando reduzir a latência do *pipeline* de processamento do sistema, e também melhorar a acurácia dos resultados.

Este Capítulo tem por objetivo apresentar o desenvolvimento do produto de hardware 4.1 que inclui a construção de uma câmera IP de baixo custo, as otimizações desenvolvidas no sistema de ALPR 4.2, apresentado as duas estratégias adotadas e como estas foram implantadas no

sistema, sendo também descritas nesta Seção as funcionalidades possíveis de serem desenvolvidas a partir do uso do módulo de rastreio. Por último, são apresentadas as descrições das ferramentas, dispositivos e serviços utilizados no trabalho 4.3.

## 4.1 Configurações do Produto de *Hardware*

Foram montadas inicialmente duas configurações para o produto com a câmera IP de baixo custo desenvolvida neste trabalho, sendo cada uma delas submetida a testes na portaria principal de veículo da Universidade Federal de Sergipe para avaliação do desempenho do sistema de ALPR sem nenhuma otimização realizada, sendo executado apenas o *pipeline* principal.

Também foi desenvolvida uma terceira configuração, a qual serviu para testar o desempenho do sistema de ALPR com otimizações mediante o uso de câmeras analógicas instaladas na portaria da prefeitura de Aracaju por meio de uma parceria com a empresa Pulsatrix[1], sendo as imagens enviadas em tempo real para o sistema de ALPR por meio do protocolo RTSP mediante uso de um DVR.

Estas três configurações foram definidas por análise feita em quatro etapas:

a) escolha do dispositivo que atuará na captura das imagens na portaria de algum estabelecimento ou condomínio;

b) caso seja escolhida uma câmera que se comunica a partir do protocolo MQTT, então é necessária a instalação de alguma ferramenta de comunicação baseada no protocolo em um servidor em nuvem, ou local, podendo até mesmo ser utilizado caso seja compatível diretamente no dispositivo de sistema embarcado. Um dos principais objetivos da utilização de uma ferramenta para comunicação via MQTT é o tráfego das imagens quando não há um sistema embarcado em uma carcaça de câmera com poder de processamento suficiente para execução do *pipeline* para ALPR localmente. Além de ser útil também para o envio de comandos e monitoramento de informações de funcionamento dos dispositivos caso necessário;

c) sendo escolhido um dispositivo compatível com o protocolo RTSP, é necessário estabelecer um identificador uniforme de recursos (*Uniform Resource Identifier*, URI) para o sistema de ALPR conseguir se conectar e receber as imagens em tempo real;

d) definir do dispositivo ou serviço em nuvem que atuará como unidade de processamento, executando o sistema de ALPR.

Para a criação da câmera, comum para ambas configurações 4.1.1 e 4.1.2, foi utilizado o NodeMCU Esp32cam 4.3.3. Este dispositivo é o responsável por registrar as imagens e enviá-las via protocolo MQTT em tempo real por meio de um tópico especificado ao servidor intermediador da comunicação.

[1] Disponível em: <http://pulsatrix.com.br/>. Acesso em maio de 2023

Este servidor realizará o gerenciamento de acesso às imagens através do tópico para que o serviço em nuvem, ou o dispositivo de sistema embarcado contendo o algoritmo e modelos para ALPR, realize a execução a fim de serem obtidos os caracteres da placa veicular. Para isso, a ferramenta baseada no MQTT escolhida neste trabalho para atuar como servidor foi o Mosquitto MQTT 4.3.2.

Com base nisso, a Subseção 4.1.1 apresenta a configuração voltada para o processamento de ALPR em nuvem por meio de uma instância Ampere A1 Compute do serviço *Oracle Infrastructure Cloud*[2], a qual é detalhada na Subseção 4.3.5. Já na Subseção 4.1.2, é apresentada a configuração para o processamento de ALPR a partir de um sistema embarcado, através da execução do algoritmo e modelos de redes neurais em um Jetson Nano, o qual apresentou bons resultados para esta finalidade segundo o trabalho feito por Silva (2022).

Vale ressaltar que estas duas primeiras configurações têm por objetivo a realização dos primeiros testes práticos do sistema de ALPR funcionando juntamente com a câmera IP enviando imagens em tempo real através do protocolo MQTT. Não sendo então prioridade, nestes primeiros testes, a extração de bons resultados.

No entanto, para as configurações desenvolvidas neste trabalho que utilizam câmeras IPs disponíveis no mercado, ou que recebem imagens mediante uso do protocolo RTSP junto a um DVR compatível, são buscados bons resultados visando inserir a aplicação em soluções para o mercado. Com base nisso, a Subseção 4.1.3 é dedicada a descrever estas configurações, uma associada ao uso do Ampere A1 Compute, e outra associada ao uso do Jetson Nano.

### 4.1.1 ALPR com Execução em Serviço em Nuvem mediante Protocolo MQTT

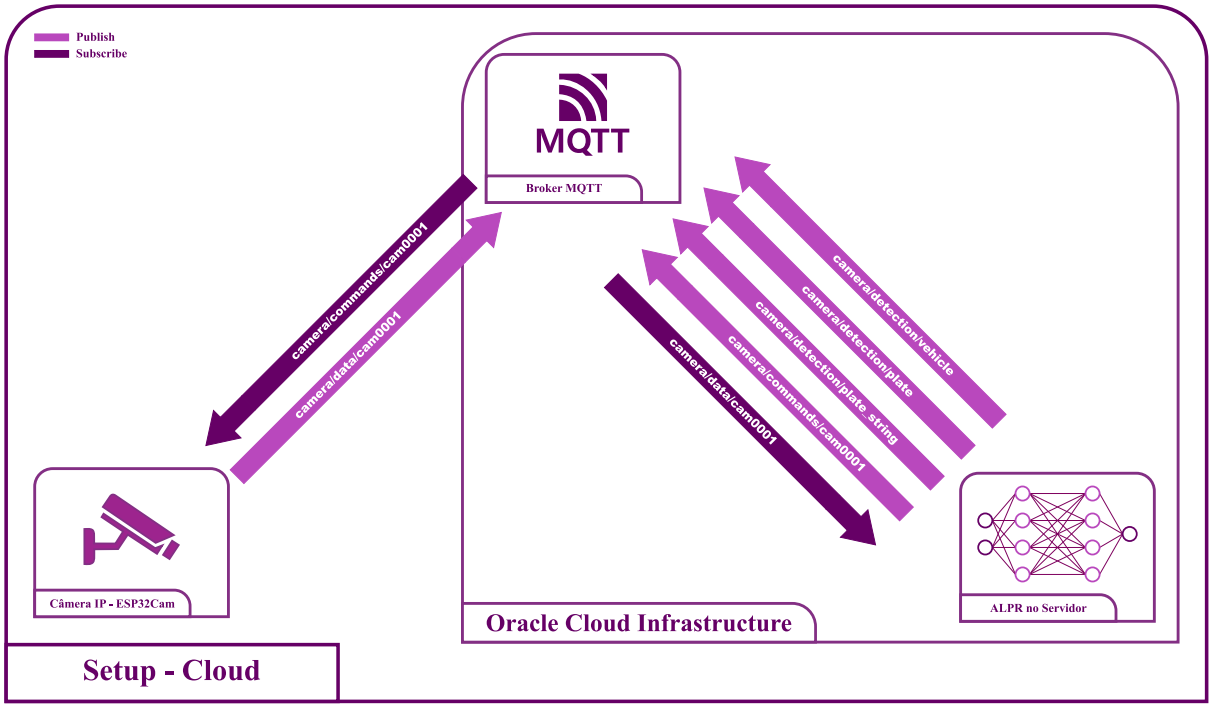
Para esta configuração, as imagens capturadas pela câmera IP de baixo custo são enviadas para um serviço em nuvem localizado em São Paulo, o qual como descrito anteriormente, trata-se de uma instância Ampere A1 Compute oferecida pela Oracle.

Na Figura 48, é apresentada uma ilustração da arquitetura de comunicação, desde a captura das imagens realizadas pelo Esp32Cam, até o recebimento e processamento destas por meio do *pipeline* principal do sistema de ALPR que está em execução na instância do Ampere A1 Compute.

Percebe-se nesta ilustração algumas informações descritas nas setas que dão o sentido do fluxo de dados entre os dispositivos ou serviços envolvidos. Estas informações tratam-se dos tópicos dos quais os clientes, que neste caso seriam a câmera formada pelo Esp32Cam e o sistema de ALPR, estão publicando ou recebendo dados.

[2] Disponível em: <https://www.oracle.com/br/cloud/>. Acesso em novembro de 2022

Figura 48 – Arquitetura de comunicação com MQTT para a configuração com o servidor Ampere A1 Compute

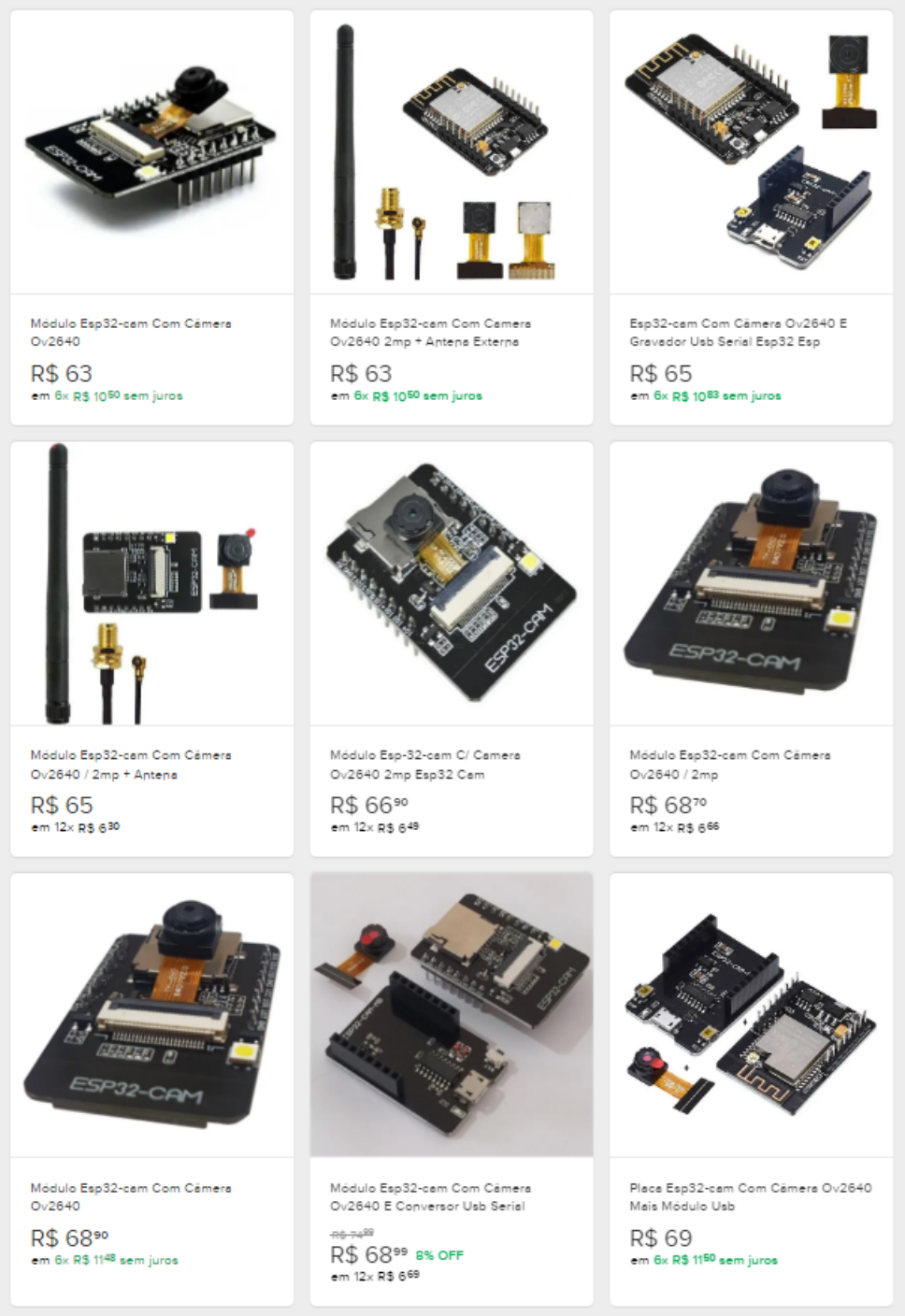



Fonte: Autor.

Estes dados trocados no contexto dessa arquitetura de comunicação são fluxos de vetor de bytes que correspondem às imagens. Porém, um destes tópicos que se trata do *camera/commands/cam0001* está vinculado aos comandos para controle do Esp32Cam, descritos na Subseção A.2.2.

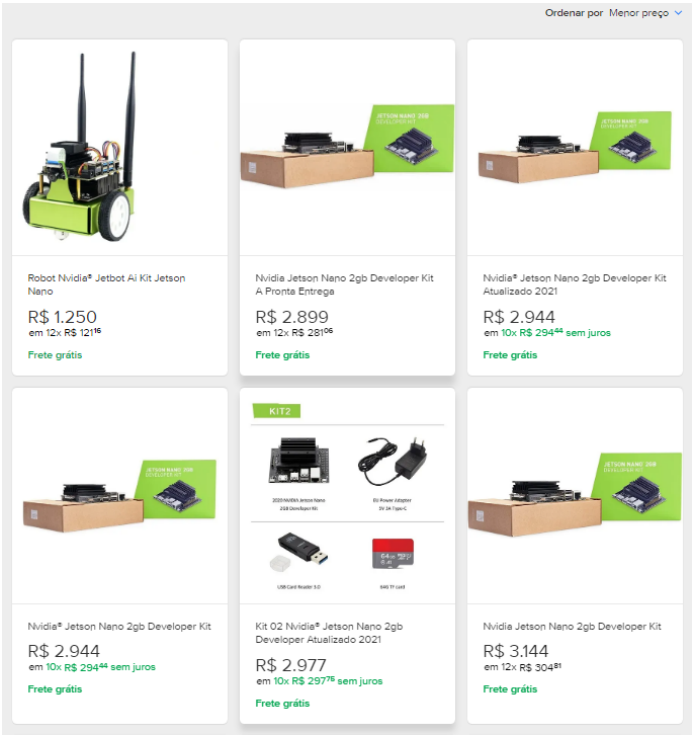
O objetivo de realizar testes com uma configuração envolvendo o servidor Ampere A1 Compute se dá pela possibilidade de reduzir custos do produto final, podendo-o deixar com preços mais competitivos no mercado. Com essa abordagem a câmera IP com o Esp32Cam seria o único componente do projeto que não está em nuvem, o qual no momento em que este trabalho está sendo desenvolvido custa em torno de 65 reais um kit completo no mercado nacional, como pode ser observado na Figura 49. Já o Jetson Nano, utilizado na configuração que será apresentada na próxima Subseção, custa valores acima de 2900 reais também no mercado nacional, como pode ser observado na Figura 50.

Figura 49 – Preços encontrados para o Esp32Cam no Mercado Livre




Fonte: Mercado Livre

Figura 50 – Preços encontrados para o Jetson Nano no Mercado Livre




Fonte: Mercado Livre

### 4.1.2 ALPR com Execução em Dispositivo com Sistema Embarcado mediante Protocolo MQTT

Esta configuração dispõe da mesma base de funcionamento que a apresentada anteriormente, mas com algumas mudanças, principalmente nos dispositivos envolvidos na arquitetura da comunicação.

A princípio, há o envolvimento de três dispositivos físicos sendo eles o Jetson Nano, o Raspberry Pi Zero W e o Esp32Cam. Vale ressaltar que a inclusão do Raspberry Pi Zero W nesta comunicação partiu da ideia de escalar mais o processamento através da adição de mais dispositivos servindo de unidade de processamento, desenvolvendo então um sistema distribuído.

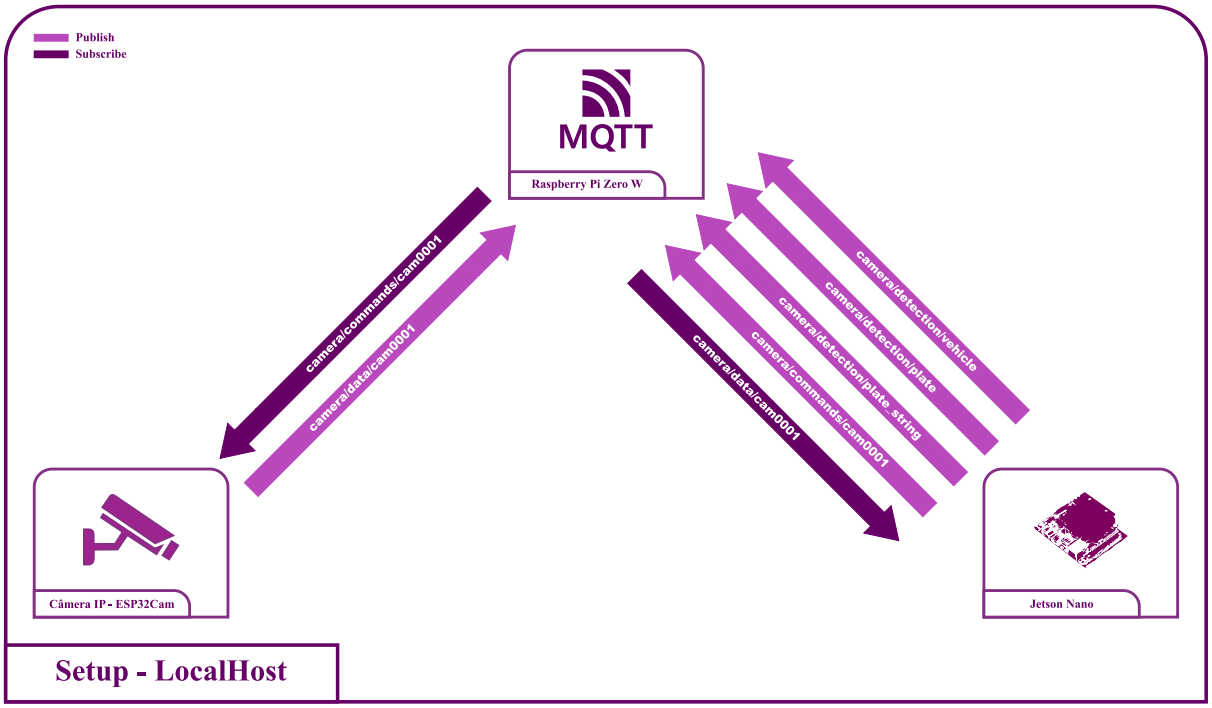
Porém, foi observado que visando a utilização principalmente do Jetson Nano em um esquema de escalabilidade, isso seria inviável pelo alto custo do produto final. Sendo assim, será considerado em cenários dos quais a câmera estará separada de um dispositivo de sistema embarcado com o sistema de ALPR, o servidor MQTT instalado na própria unidade de processamento do sistema de ALPR, reduzindo até mesmo a latência da comunicação.

Como pode ser observado na Figura 51, a comunicação através dessa configuração parte da captura das imagens através da câmera IP. Diferindo da configuração com o uso do servidor Ampere A1 Compute em que as imagens vão para a mesma unidade da qual será realizado

o processamento ALPR, porque neste caso as imagens vão para o Raspberry Pi Zero W, que atua como servidor MQTT. A partir do Raspberry Pi Zero W é feito o encaminhamento destas imagens para o Jetson Nano, o qual é de fato a unidade de processamento.

Já quanto aos tópicos e o funcionamento quanto às inscrições e publicações através do protocolo MQTT, segue o mesmo contexto relatado na descrição da configuração utilizando o servidor Ampere A1 Compute.

Figura 51 – Arquitetura de comunicação com MQTT para a configuração utilizando o Jetson Nano

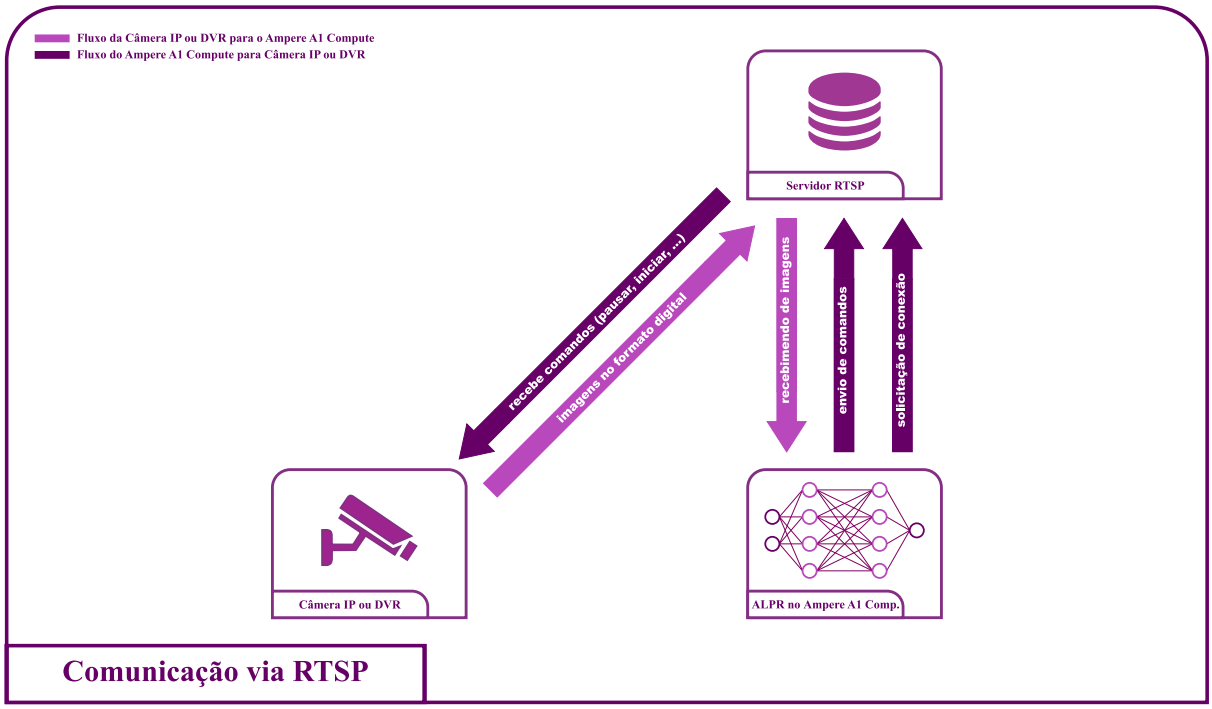



Fonte: Autor.

### 4.1.3 Sistema de ALPR com Interface para Protocolo RTSP

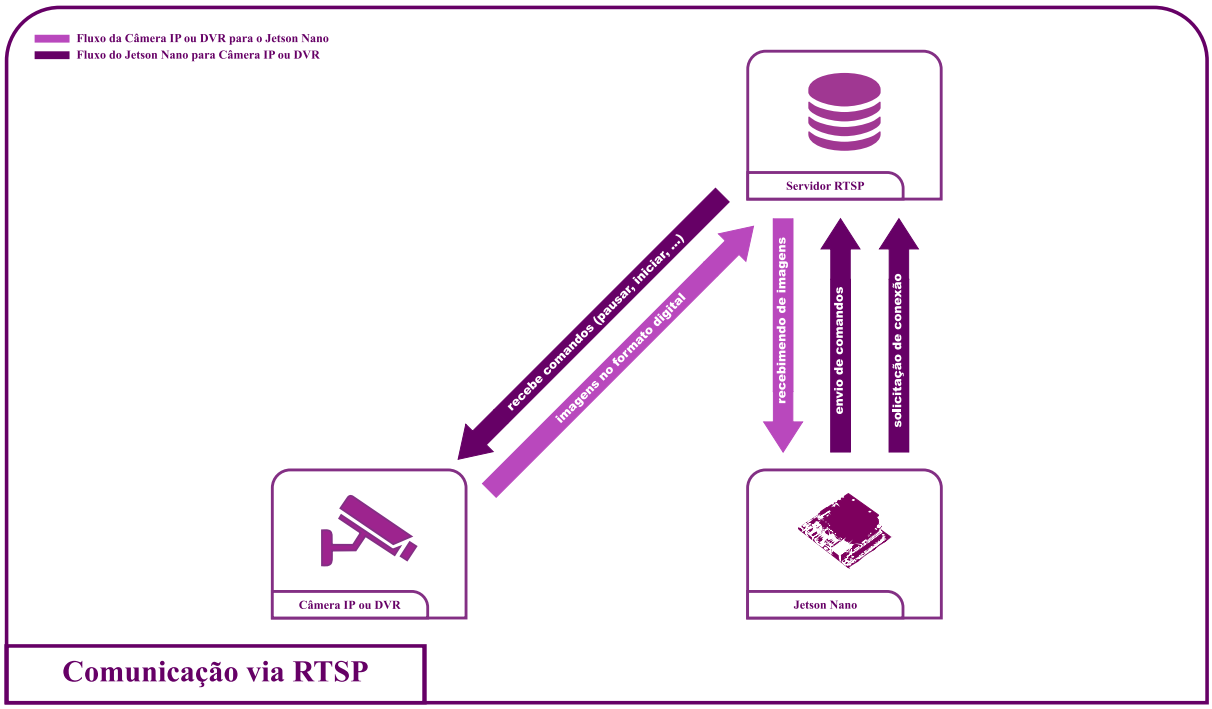
Esta configuração foi pensada principalmente visando possibilitar o uso de câmeras IPs e DVRs disponíveis no mercado junto ao sistema de ALPR. Sendo possível por câmeras e alguns DVRs de marcas como a Intelbras suportar o protocolo RTSP como um dos possíveis protocolos para acesso externo de imagens em tempo real. Nas Figuras 52 e 53, são ilustradas arquiteturas de comunicação via RTSP tanto com o servidor Ampere A1 Compute quanto com o Jetson Nano, recebendo as imagens de uma câmera IP ou de um DVR através de um URI configurado nestes.

Figura 52 – Arquitetura de comunicação com protocolo RTSP junto ao servidor Ampere A1 Compute




Fonte: Autor.

Figura 53 – Arquitetura de comunicação com protocolo RTSP junto ao Jetson Nano




Fonte: Autor.

Vale ressaltar que esta interface RTSP permite comunicação entre o sistema de ALPR e um DVR, então câmeras não IPs como a VHD 3230 B G5[3] da Intelbras podem ser utilizadas para trabalhar na coleta de imagens para o sistema de ALPR, já que suas imagens são tratadas pelo DVR ao qual estão conectadas, chegando ao sistema de ALPR já no formato digital.

## 4.2 Otimizações no Sistema de ALPR

Para tornar o sistema de ALPR desenvolvido apto para atuar no mercado foi necessário realizar algumas otimizações visando deixá-lo mais próximo do funcionamento em tempo real. Para isso foi buscado elevar a taxa de quadros e conseguir melhorar a acurácia dos resultados retornados para evitar cadeias de caracteres de placas que não existem, ou que não correspondem ao veículo em questão.

Para esta etapa do trabalho foram selecionadas duas abordagens principais, a primeira delas foi a criação de um módulo que permite o rastreio de objetos, possibilitando identificar se os resultados retornados são de um mesmo veículo, alcançando redução da latência do processamento e aumento da acurácia com algumas modificações e funcionalidades apresentadas na Subseção 4.2.1.

Vale ressaltar que o módulo de rastreio de objetos tem dois perfis, sendo um deles o uso aplicado a situações das quais o resultado do ALPR estará vinculado com algum sistema de automação na entrada de veículos em uma área restrita, e o outro é voltado para situações das quais o ALPR está sendo utilizado mais para controle dos veículos, não sendo relacionado a sistemas de automação.

Contudo, enquanto a etapa de otimização descrita acima atua no algoritmo do sistema de ALPR, a segunda etapa atua no modelo de detecção da placa do veículo através da compressão do modelo descrita na Subseção 4.2.2.

### 4.2.1 Módulo de Rastreio de Objetos

A importância do desenvolvimento desse módulo parte da necessidade de distinguir os veículos aos quais os resultados retornados devem ser associados. Ou seja, sem este módulo o sistema retorna as cadeias de caracteres das placas para cada nova imagem recebida que contém pelo menos um veículo sem se importar em saber se os vários resultados retornados ao processar várias imagens são referentes a um mesmo veículo, ou se também foram retornadas cadeias de caracteres referentes a outros.

Considerando uma aplicação básica do ALPR em um contexto real de uso a falta do módulo tornaria inviável a utilização do sistema, pois poderia causar uma sobrecarga em uma

[3] Disponível em: <https://www.intelbras.com/pt-br/camera-infravermelho-multi-hd-vhd-3230-b-g5>. Acesso em março de 2023

API (*Application Programming Interface*) ao realizar várias requisições para consultar se o veículo pode ter acesso ao ambiente em questão. Essas várias requisições poderiam ser trocadas por uma única requisição, retornando o resultado de um único veículo, o qual foi detectado em várias imagens processadas ao longo do tempo em que ficou no campo de visão da câmera.

Então, considerando o desenvolvimento deste módulo, a Subseção 4.2.1.1 descreve como o módulo de rastreio foi adicionado ao *pipeline* do sistema de ALPR, apresentando e descrevendo alguns fluxogramas que ilustram a interação entre o módulo de rastreio e o *pipeline* principal.

Já na Subseção 4.2.1.2 é descrito como foi desenvolvido o tratamento de redundância temporal, o qual realiza um trabalho de composição de dígitos da placa com base em todas as cadeias de caracteres retornadas para a placa de um mesmo veículo durante o rastreio deste, além de tratar do trabalho realizado pelos dois diferentes perfis de funcionamento do módulo de rastreio citados no início desta Seção. Por último, na Subseção 4.2.1.3, é apresentado como foi utilizado o módulo de rastreio para realizar a contagem de veículos.

#### 4.2.1.1 Inclusão ao *Pipeline* do ALPR

O desafio para a utilização de algum algoritmo de rastreio junto ao sistema de ALPR é fazer ambos trabalharem bem juntos, sem redução de desempenho por possível mau funcionamento do algoritmo de rastreio comprometendo o funcionamento de todo o sistema de ALPR. A princípio foi buscado utilizar os algoritmos KCF ou o CSRT integrados ao sistema, os quais são disponíveis na API *Tracker*[4] da biblioteca OpenCV (*Open Source Computer Vision*) na linguagem Python.

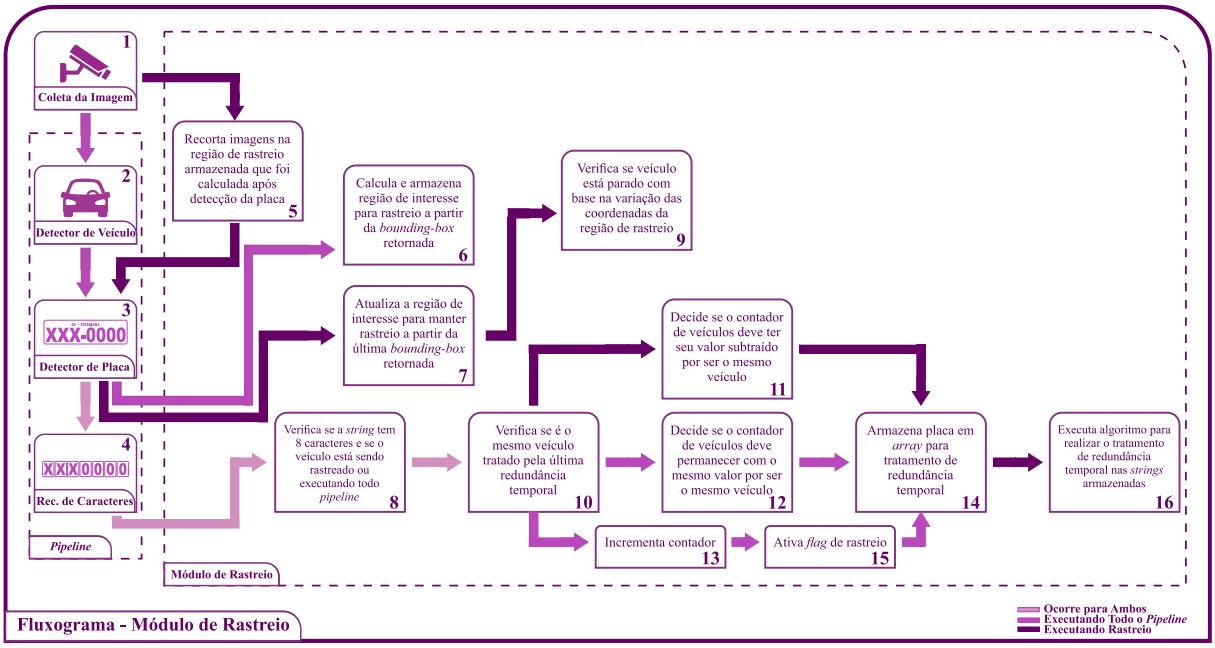
A abordagem para trabalhar com cada um desses é parecida com a apresentada no fluxograma da Figura 54 para o módulo de rastreio desenvolvido, no qual pode ser observado que quando não está sendo realizado rastreio apenas o detector de veículos é executado a cada nova imagem recebida. Porém, uma vez detectado um veículo, todo o *pipeline* principal do sistema de ALPR é executado.

Ao ser identificada a caixa delimitadora da placa na imagem processada passa a ser executado o algoritmo de rastreio, evitando que no intervalo de tempo no qual o veículo percorrerá o trajeto pelo ambiente registrado pela câmera haja novas execuções do detector de veículos, reduzindo assim a latência do processamento.

Vale ressaltar que a principal função do detector de veículo no *pipeline* é estabelecer uma região de interesse (*Region of Interest*, ROI), que evita a detecção de outros objetos em cena que são similares a uma placa veicular. Então, apenas é enviada para o detector de placas essa região da imagem que corresponde a área ocupada pelo veículo.

[4] Disponível em: <https://docs.opencv.org/3.4/d0/d0a/classcv_1_1Tracker.html>. Acesso em março de 2023

Figura 54 – Fluxograma do módulo de rastreio implantado ao sistema de ALPR




Fonte: Autor.

Para algoritmos de rastreio a ROI tem ainda mais relevância, porque indica a posição inicial do objeto que deve ser rastreado nas imagens seguintes, sem necessidade de ser treinado um modelo para realizar o trabalho. Sendo assim, o algoritmo de rastreio torna-se mais leve e rápido do que utilizar em todas as imagens os detectores.

A partir de testes realizados com o algoritmo de KCF junto ao *pipeline* principal do sistema, e em outro momento com o algoritmo de CSRT, foi percebido que estes algoritmos perdem facilmente a referência do objeto indicado na imagem, o qual se trata da placa. Ou então indicavam falsamente objetos no plano de fundo.

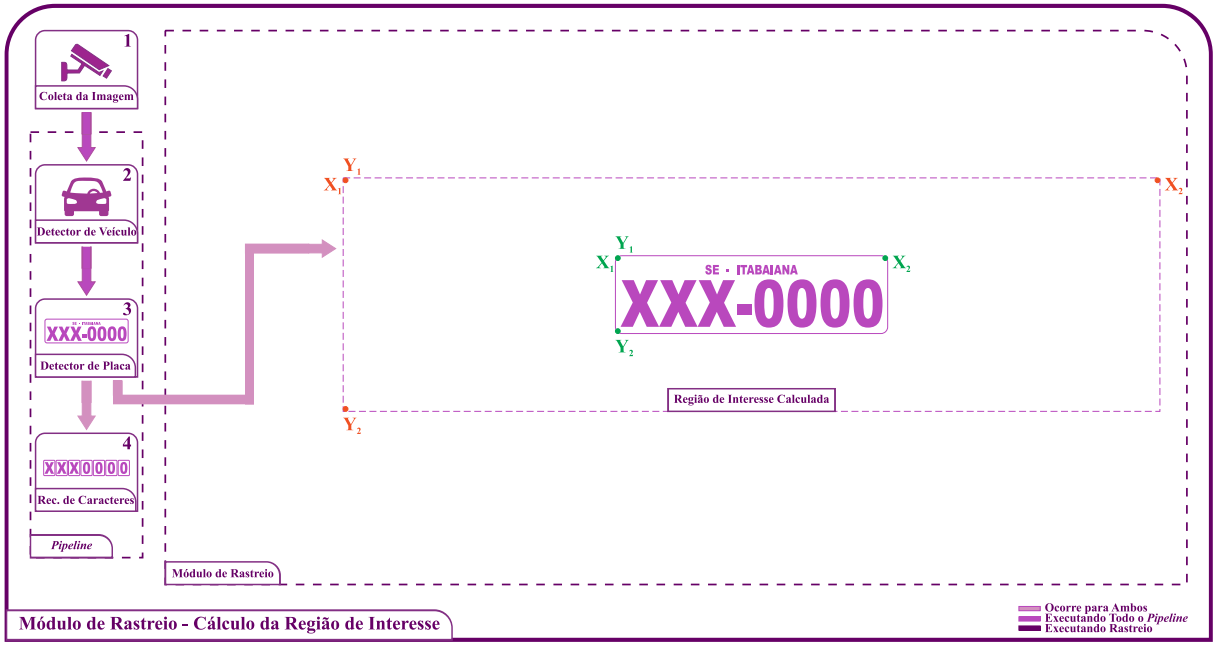
Dentre os testes realizados a única configuração que alcançou um desempenho razoável foi um trabalho feito com o algoritmo CSRT fazendo o rastreio do veículo em vez do rastreio da placa, e ainda assim perdia com certa facilidade a referência. Vale ressaltar que o objetivo da utilização destes algoritmos de rastreio simples era de inabilitar tanto o detector de veículo como o de placas, mantendo em execução dos detectores do *pipeline* principal apenas o reconhecedor de caracteres durante o rastreio do objeto.

Como estas abordagens não funcionaram foi decidido desenvolver um módulo de rastreio que trabalhasse com as caixas delimitadoras retornadas pelo detector de placa. Sendo assim, foi primeiro testado utilizar um algoritmo preditor que ao serem recebidas as duas primeiras caixas delimitadoras da placa de um veículo fizesse o calculado de onde esta placa estaria na próxima imagem recebida, começando então a ser realizado o rastreio. Mas como a aplicação é em tempo real, travamentos e atrasos no recebimento das imagens tornaram esta abordagem

inviável, alcançando um desempenho pior do que utilizando apenas o *pipeline* principal.

Então, foi considerada uma abordagem que, em vez de prever onde a placa estaria na próxima imagem, fosse calculada uma ROI com base na caixa delimitadora atual da placa detectada que substituiria na imagem seguinte o detector de veículos, indicando uma região que provavelmente a placa estará. Com isso foi mantida a execução do detector de placa e desabilitado o detector de veículos durante o processo de rastreio, sendo esta abordagem ilustrada pela Figura 55.

Figura 55 – Região de interesse baseada na caixa delimitadora retornada da placa



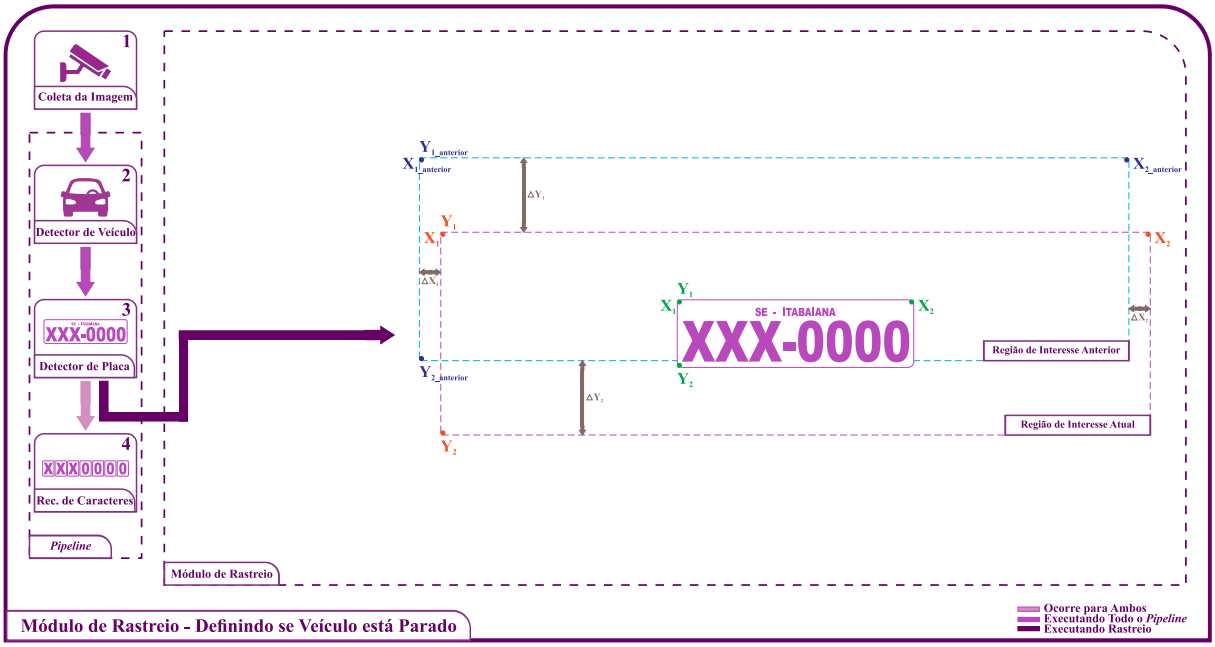

Fonte: Autor.

Com essa abordagem foi alcançada a consistência do rastreio, principalmente pela possibilidade de determinar o fator de escala para o cálculo dessa região de interesse, que pode ser utilizado como um ajuste fino considerando a velocidade que os carros passam pela câmera, o próprio posicionamento da câmera, entre outros. No caso da Figura 55, é ilustrada a ROI com o cálculo realizado com fator de escala 1, determinando que a região de interesse deve ter o $X_1$ igual ao $x_1$ da caixa delimitadora da placa subtraído por uma vez a largura da placa, acontecendo o mesmo para as outras três coordenadas. Porém, no caso do $X_2$ e $Y_2$ o valor da largura e altura da placa são respectivamente somados.

Também é aproveitada a região de interesse para determinar se o veículo está parado, sendo útil devido à naturalidade do veículo aguardar uma portaria ser aberta para entrar. Então, com o veículo parado só é necessário o processamento ocorrer uma vez para extrair a cadeia de caracteres da placa, evitando que seja utilizado desnecessariamente o reconhecedor de caracteres nas imagens seguintes das quais o veículo continua parado no mesmo local. Para isso é verificada

a variação das coordenadas entre a região de interesse calculada na última caixa delimitadora, e a região de interesse calculada com base na caixa delimitadora atual, como apresentado pela Figura 56.

Figura 56 – Definindo se veículo está parado




Fonte: Autor.

Supondo que o veículo esteja se movendo de cima para baixo há uma pequena variação no eixo horizontal (X), como também ocorre uma variação até mais considerável no eixo vertical (Y) entre as regiões de interesse. A partir disso é verificada a variação destas coordenadas e, caso para todas elas a variação for menor do que 1,5% da largura da imagem para o eixo horizontal, e 1,5% da altura da imagem no eixo vertical, então é considerado que o veículo está parado e não é executado o reconhecedor de caracteres. Por outro lado, se pelo menos uma das coordenadas variar mais do que 1,5% volta a ser realizado o reconhecimento dos caracteres. Vale ressaltar que este fator de tolerância é um parâmetro ajustável no sistema de ALPR.

#### 4.2.1.2 Tratamento de Redundância Temporal

Sugerido por Laroca et al. (2018), o tratamento de redundância temporal é importante para melhorar a acurácia dos resultados retornados pelo sistema, permitindo que ao ser encerrada a execução do módulo de rastreio em um veículo seja vinculada apenas uma cadeia de caracteres da placa a este com um aumentando da acurácia dos dígitos reconhecidos.

O procedimento é realizado por meio da execução de um sistema de votação que analisa os caracteres que mais se repetiram em cada dígito nas cadeias de caracteres armazenadas da placa durante o rastreio, construindo a partir dessa votação uma cadeia de caracteres que tende a

melhor corresponder aos dígitos reais da placa do veículo. Porém, como dito anteriormente, o processamento do tratamento de redundância temporal ocorre apenas quando o rastreio para um determinado veículo é encerrado, então é importante definir claramente quando que o algoritmo para tratamento de redundância temporal deve atuar considerando o contexto ao qual o sistema de ALPR estará implantado. É devido a isso que são necessários dois perfis de funcionamento do módulo de rastreio.

Para o primeiro perfil é considerada a situação do sistema de ALPR estar implantado em uma portaria com a qual o veículo irá parar para aguardar alguma ação automática que depende do resultado do reconhecimento dos caracteres da placa. Considerando isso, o ponto crítico desse perfil é o momento em que deve ser feito o tratamento de redundância, pois o módulo de rastreio é por padrão configurado para continuar em execução aguardando o veículo sair do campo de visão da câmera para executar o tratamento de redundância e retornar a placa. Porém, caso isso aconteça o sistema dependente do resultado do ALPR nunca receberá a informação para a tomada de decisão, sendo necessário que ocorra o tratamento de redundância ainda durante o rastreio, no momento em que é identificado que o veículo está parado.

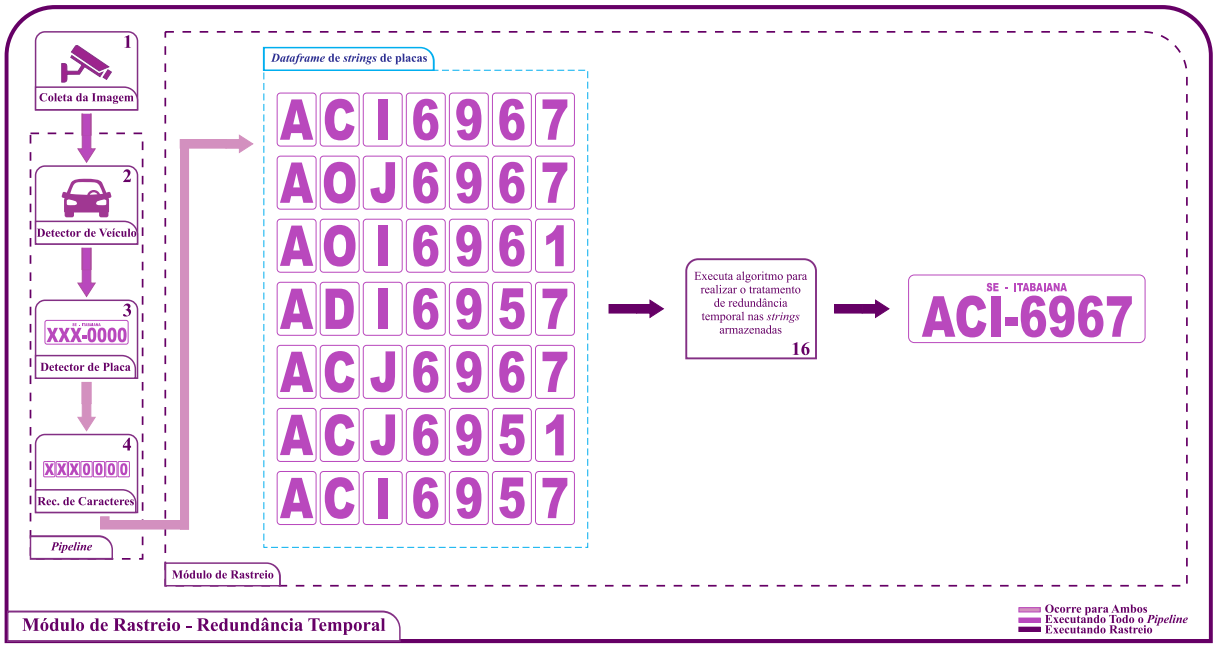
Vale destacar que quando o veículo é identificado como parado só continua em funcionamento o detector de placa, mas uma vez que o veículo volta a se deslocar o reconhecedor de caracteres volta a ser executado. No entanto, neste primeiro perfil o reconhecedor de caracteres não deve voltar mais a funcionar enquanto perpetuar o rastreio da placa do veículo, pois já deverá ter sido feita a comunicação entre o sistema de ALPR e o sistema de automação, sendo desnecessário manter o reconhecedor de caracteres em execução, permitindo reduzir até a latência enquanto o rastreio continua sendo executado até o veículo sair de cena.

Foi também considerada a possibilidade de finalizar por completo o rastreio do objeto, mas não seria viável porque ao veículo voltar a se deslocar este seria considerado um novo veículo, sendo necessário um esforço maior para corrigir a inconsistência gerada no contador. Sendo assim, foi determinado que o módulo de rastreio continuará em execução mesmo que o reconhecedor de caracteres não esteja mais em execução devido o tratamento de redundância temporal já ter sido realizado.

Já quanto ao segundo perfil, o cenário de não conter sistemas dependentes do ALPR na portaria faz com que não haja motivos para realizar um tratamento de redundância temporal durante o processo de rastreio. Sendo assim, o funcionamento base do módulo de rastreio é suficiente, só acontecendo o tratamento após o veículo sair do campo de visão da câmera, aproveitando então o armazenamento de mais cadeias de caracteres, possibilitando alcançar maior acurácia no resultado.

Na Figura 57, é ilustrado o procedimento do tratamento de redundância temporal no módulo de rastreio.

Figura 57 – Processo de votação para tratamento de redundância temporal




Fonte: Autor.

#### 4.2.1.3 Contador de Veículos

Como o deslocamento do veículo é rastreado a partir do momento em que é realizada a primeira detecção através da execução de todo o *pipeline* principal do sistema de ALPR, e por padrão finaliza quando o veículo sai do campo de visão da câmera, torna-se possível realizar a contagem destes. Isso se torna útil quando há necessidade de monitorar o fluxo de carros em uma portaria, ou até mesmo controlar a quantidade de vagas em um estacionamento.

Durante o desenvolvimento do algoritmo para realizar a contagem dos veículos foi percebido em testes inconsistências do valor do contador. Estas aconteceram principalmente porque nem sempre o módulo de rastreio conseguia acompanhar a placa do veículo rastreado por todo o seu deslocamento no ambiente capturado pela câmera. Isso ocorre por um dos desafios ao trabalhar com rastreio de objetos que é a oclusão, seja por a câmera perder sinal temporariamente, ou pessoas passarem à frente da placa do veículo durante o rastreio, ocorrendo de um mesmo veículo ser contado mais de uma vez.

Então, a abordagem implementada para tornar o módulo de rastreio robusto aos problemas de oclusão foi a de armazenar temporariamente a última lista de cadeias de caracteres de placas utilizadas pelo último tratamento de redundância, permitindo verificar para cada nova cadeia de caracteres retornada pelo reconhecedor de caracteres para o veículo atual se esta fez parte do processamento do veículo anterior.

Se durante todo o rastreio do veículo atual nenhuma das cadeias de caracteres retornadas

fez parte do processamento anterior, é assumido que o veículo rastreado difere do anterior. Por outro lado, caso seja identificado que alguma das cadeias de caracteres fez parte, então é decidido entre decrementar ou manter o valor do contador a depender se esta cadeia de caracteres foi retornada ao ser executado todo o *pipeline* na primeira detecção, ou se foi no momento em que já estava sendo realizado o rastreio. Além da correção da inconsistência é mesclada a lista de cadeias de caracteres do processamento atual com a do processamento anterior visando aumentar a acurácia do resultado do tratamento de redundância temporal.

### 4.2.2 Compressão do Modelo YOLOv5 Nano

A ideia principal do funcionamento do módulo de rastreio era de ser trabalhado algum dos algoritmos baseados em filtros correlacionais discriminativos em conjunto com o *pipeline*. Então, a inicialização seria a partir da execução de todo o *pipeline* na primeira imagem para se obter a região de interesse com o objeto alvo, que se dá pela placa do veículo, sendo mantida atualizações do algoritmo de rastreio nas imagens subsequentes, e com a nova atualização da posição do objeto a imagem recortada seria encaminhada para o reconhecedor de caracteres.

Contudo, como dito antes, não foram alcançados bons resultados nos testes realizados com os algoritmos de rastreio CSRT e KCF. Sendo assim, com a nova abordagem construída para o módulo de rastreio descrita anteriormente, surge a necessidade de reduzir ao máximo a latência do detector de placas para buscar um desempenho aproximado do qual era esperado com a ideia principal.

Então, com base nas pesquisas realizadas nos métodos de compressão de modelos YOLO, em um primeiro momento foi encontrado um módulo que possibilita a poda de pesos através do próprio repositório oferecido pela Ultralytics para a YOLOv5[5]. No entanto, após alguns testes foi percebido que o tamanho do modelo não foi reduzido, assim como a latência das inferências se manteve, mas houve uma perda considerável de acurácia dos resultados.

Ao ser pesquisada a causa foi encontrado ainda no repositório da YOLOv5 que este módulo de fato zera alguns pesos da rede neural realizando a poda, mas mantém intacta a estrutura do modelo. Então, por não possibilitar serem alcançados os benefícios esperados esta abordagem foi desconsiderada, já que há a necessidade de modificações estruturais da rede para os resultados esperados aparecerem.

Então, mediante novas buscas foram encontradas três ferramentas robustas oferecidas pela empresa Neural Magic a partir de trabalhos desenvolvidos na área de compressão de modelos profundos (KURTZ et al., 2020). Estas ferramentas que se complementam tem como proposta disponibilizar recursos para remover dos modelos parâmetros redundantes que influenciam no tempo de inferência e tamanho do modelo, mas que não geram considerável influência na acurácia destes, tornando quando removidos o modelo mais leve e rápido. As ferramentas estão

[5] Disponível em: <https://github.com/ultralytics/yolov5/issues/304>. Acesso em abril de 2023

voltadas para várias arquiteturas de aprendizado profundo, inclusive YOLOv5, visando alcançar o desempenho da execução do modelo denso acelerado por GPU (*Graphics Processing Unit*) apenas utilizando a CPU (*Central Processing Unit*) com o modelo comprimido.

Para realizar a compressão é utilizada a ferramenta SparseML[6]. Esta trata-se de uma biblioteca que permite treinar modelos podados e/ou quantizados a partir de dados customizados por poucas linhas de código. Isso é possível a partir de duas abordagens oferecidas, sendo uma opção a realização do ajuste fino a partir da transferência de conhecimento esparso, ou então realizar o procedimento de escassez do modelo do zero. Estas abordagens são descritas a seguir:

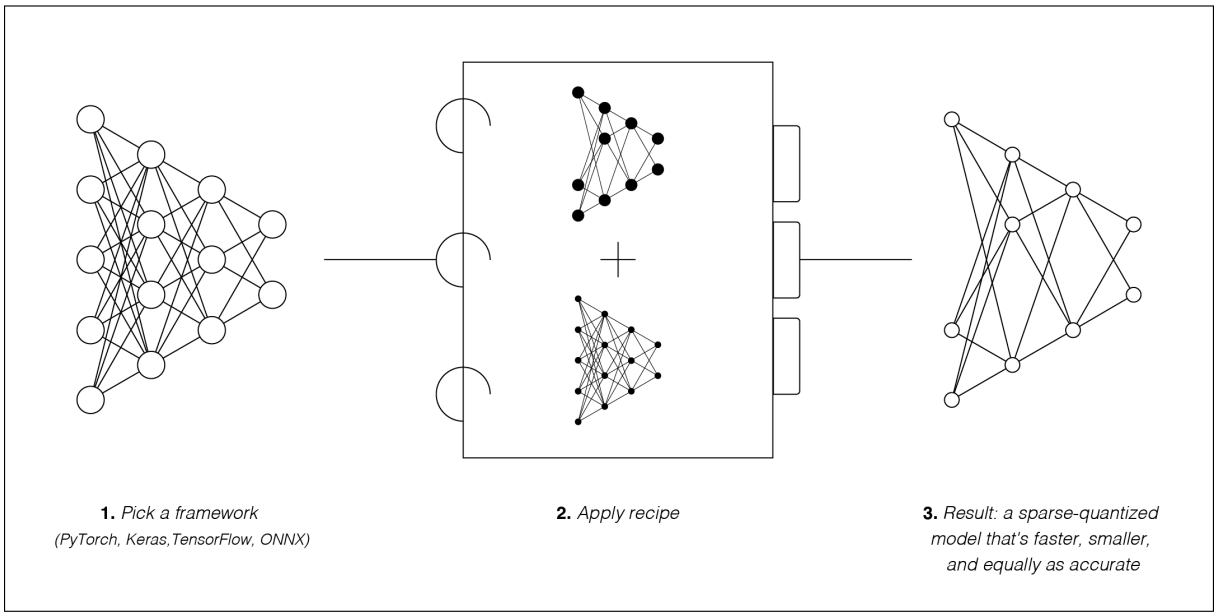
- **Transferência de Conhecimento Esparso ou *Sparse Transfer Learning***: Trata-se da forma mais recomendada pela própria empresa para se obter o modelo esparso, partindo da possibilidade de realizar um ajuste fino a partir de modelos *pre-sparsified*, dos quais são oferecidos por outra ferramenta da Neural Magic, chamada SparseZoo[7]. A qual trata-se de um repositório de modelos esparsos e/ou quantizados, oferecendo as respectivas receitas (*recipes*) para a aplicação do ajuste fino. E quanto ao ajuste fino do modelo, trata-se do ajuste de toda a rede neural do modelo *pre-sparsified* para trabalhar com os novos dados de treinamento, sendo estes os próprios dados customizados do usuário para o contexto pretendido de detecção. A SparseML oferece suporte para esta abordagem quando se trata de modelos para classificação de imagem, detecção de imagens ou processamento de linguagem natural;

- **Procedimento de Escassez do Zero ou *Sparsifying from Scratch***: Possibilita a aplicação da poda e/ou quantização de qualquer modelo através da ferramenta SparseML. Porém, o tempo de treinamento é maior por envolver mais épocas, que indicam quantas vezes o modelo deve rever todos os dados de treinamento para realizar ajustes dos pesos. E, além disso, os hiperparâmetros devem ser definidos manualmente, requerendo vários testes para buscar resultados ótimos, diferente do método anterior que a SparseZoo já fornece os hiperparâmetros que devem ser utilizados no procedimento.

Para definir o procedimento de poda e/ou de quantização em qualquer uma das duas abordagens acima é necessário definir por meio de um arquivo uma receita. Nesta são descritos modificadores, sendo as instruções necessárias para ser indicado como deve acontecer o processo de poda e/ou quantização. Na Figura 58, é apresentada uma visão geral do procedimento realizado pela SparseML.

[6] Disponível em: <https://docs.neuralmagic.com/products/sparseml>. Acesso em abril de 2023
[7] Disponível em: <https://docs.neuralmagic.com/products/sparsezoo>. Acesso em abril de 2023

Figura 58 – Visão geral do procedimento de poda e/ou quantização pela SparseML




Fonte: SparseML

A SparseML foi utilizada neste trabalho para obtenção de versões comprimidas do modelo YOLOv5 Nano voltado para detecção de placas de veículos. Para isso, foi considerada a abordagem de transferência de conhecimento esparso de quatro modelos pré-treinados na base de dados COCO e que passaram pelo procedimento de poda e/ou quantização. Os modelos são YOLOv5 Nano com 30% de poda e sem quantização, YOLOv5 Nano com 30% de poda e com quantização, YOLOv5 Nano com 40% de poda e sem quantização, e YOLOv5 Nano com 40% de poda e com quantização. As receitas oferecidas pela SparseZoo para realização do ajuste fino destes modelos são respectivamente apresentadas em Algoritmo 1, Algoritmo 2, Algoritmo 3 e Algoritmo 4.

Algoritmo 1 – Receita para ajuste fino com 30% de poda e sem quantização

```
version: 1.1.0
# General variables
num_epochs: 50

training_modifiers:
  - !EpochRangeModifier
    start_epoch: 0
    end_epoch: eval(num_epochs)
pruning_modifiers:
  - !ConstantPruningModifier
    start_epoch: 0.0
    params: __ALL_PRUNABLE__
```

Algoritmo 2 – Receita para ajuste fino com 30% de poda e com quantização

```
version: 1.1.0
# General variables
num_epochs: 55
quantization_epochs: 5
quantization_lr: 1.e-5
final_lr: 1.e-9

training_modifiers:
  - !EpochRangeModifier
    start_epoch: 0
    end_epoch: eval(num_epochs)
  - !LearningRateFunctionModifier
    start_epoch: eval(num_epochs - quantization_epochs)
    end_epoch: eval(num_epochs)
    lr_func: cosine
    init_lr: eval(quantization_lr)
    final_lr: eval(final_lr)
pruning_modifiers:
  - !ConstantPruningModifier
    start_epoch: 0.0
    params: __ALL_PRUNABLE__
quantization_modifiers:
  - !QuantizationModifier
    start_epoch: eval(num_epochs - quantization_epochs)
    submodules:
      - model
    custom_quantizable_module_types: ['SiLU']
    exclude_module_types: ['SiLU']
    quantize_conv_activations: False
    disable_quantization_observer_epoch: eval(num_epochs - 1)
    freeze_bn_stats_epoch: eval(bn_freeze_epoch)
```

Algoritmo 3 – Receita para ajuste fino com 40% de poda e sem quantização

```
version: 1.1.0
# General variables
num_epochs: 50

training_modifiers:
  - !EpochRangeModifier
    start_epoch: 0
    end_epoch: eval(num_epochs)
pruning_modifiers:
  - !ConstantPruningModifier
    start_epoch: 0.0
    params: __ALL_PRUNABLE__
```

Algoritmo 4 – Receita para ajuste fino com 40% de poda e com quantização

```
version: 1.1.0
# General variables
num_epochs: 55
quantization_epochs: 5
quantization_lr: 1.e-5
final_lr: 1.e-9

training_modifiers:
  - !EpochRangeModifier
    start_epoch: 0
    end_epoch: eval(num_epochs)
  - !LearningRateFunctionModifier
    start_epoch: eval(num_epochs - quantization_epochs)
    end_epoch: eval(num_epochs)
    lr_func: cosine
    init_lr: eval(quantization_lr)
    final_lr: eval(final_lr)
pruning_modifiers:
  - !ConstantPruningModifier
    start_epoch: 0.0
    params: __ALL_PRUNABLE__
quantization_modifiers:
  - !QuantizationModifier
    start_epoch: eval(num_epochs - quantization_epochs)
    submodules:
      - model
    custom_quantizable_module_types: ['SiLU']
    exclude_module_types: ['SiLU']
    quantize_conv_activations: False
    disable_quantization_observer_epoch: eval(num_epochs -
       quantization_epochs + 2)
    freeze_bn_stats_epoch: eval(num_epochs - quantization_epochs + 1)
```

No ajuste fino dos modelos escassos pré-treinados foi considerada a mesma base de dados utilizada no treinamento do modelo denso, que se trata da base de dados UFS-ALPR desenvolvida por Silva (2022). Sendo assim, foram definidas 3499 imagens para treinamento e 1530 imagens para testes.

Para realizar o treinamento e consequentemente o ajuste fino é definido em um arquivo com extensão YAML o caminho para as pastas com as imagens e os arquivos de texto contendo as verdadeiras caixas delimitadoras dos objetos e as respectivas classes, mantendo a formatação padrão da YOLOv5. O arquivo foi nomeado como “data.yaml” e seu conteúdo é apresentado no Algoritmo 5.

Algoritmo 5 – Conteúdo do arquivo YAML com definições dos dados da base de dados UFS-ALPR

```
path: /kaggle/working/datasets/ufs-alpr-license_plate/
train: images/train
val: images/test

names:
  0: old
  1: mercosul
```

Como pode ser observado no arquivo "data.yaml" há duas classes para as placas, com rótulo 0 está a classe *old* identificando as placas veiculares no padrão antigo. Já com rótulo 1 está a classe *mercosul*, que identifica o mais novo padrão de placas adotado no Brasil.

Para realizar o treinamento através da SparseML, a SparseZoo disponibiliza o comando CLI (*Command-Line Interface*) já com alguns parâmetros preenchidos, indicando o modelo pré-treinado podado e/ou quantizado que será utilizado no ajuste fino, os hiperparâmetros, a receita com os modificadores que serão aplicados e o arquivo de configurações do modelo. Na mesma sequência de apresentação das receitas, são apresentados a seguir comandos CLI para o treinamento de cada modelo no Algoritmo 6, Algoritmo 7, Algoritmo 8 e Algoritmo 9.

Algoritmo 6 – Comando CLI da SparseML para ajuste fino do modelo com 30% de poda sem quantização

```
sparseml.yolov5.train \
  --cfg yolov5n.yaml \
  --weights
     zoo:cv/detection/yolov5-n/pytorch/ultralytics/coco/pruned30-none
     -vnni \
  --recipe
     zoo:cv/detection/yolov5-n/pytorch/ultralytics/coco/pruned30-none
     -vnni?recipe_type=transfer_learn \
  --data /kaggle/working/data.yaml \
  --patience 0 \
  --hyp hyps/hyp.finetune.yaml
```

Algoritmo 7 – Comando CLI da SparseML para ajuste fino do modelo com 30% de poda com quantização

```
sparseml.yolov5.train \
  --cfg yolov5n.yaml \
  --weights
     zoo:cv/detection/yolov5-n/pytorch/ultralytics/coco/pruned30_
     quant-none-vnni?recipe_type=transfer_learn \
  --recipe
     zoo:cv/detection/yolov5-n/pytorch/ultralytics/coco/pruned30_
     quant-none-vnni?recipe_type=transfer_learn \
  --data /kaggle/working/data.yaml \
  --patience 0 \
  --hyp hyps/hyp.finetune.yaml
```

Algoritmo 8 – Comando CLI da SparseML para ajuste fino do modelo com 40% de poda sem quantização

```
sparseml.yolov5.train \
  --cfg yolov5n.yaml \
  --weights
     zoo:cv/detection/yolov5-n/pytorch/ultralytics/coco/pruned40-none
     \
  --recipe
     zoo:cv/detection/yolov5-n/pytorch/ultralytics/coco/pruned40-none
     ?recipe_type=transfer_learn \
  --data /kaggle/working/data.yaml \
  --patience 0 \
  --hyp hyps/hyp.VOC.yaml
```

Algoritmo 9 – Comando CLI da SparseML para ajuste fino do modelo com 40% de poda com quantização

```
sparseml.yolov5.train \
  --cfg yolov5n.yaml \
  --weights
     zoo:cv/detection/yolov5-n/pytorch/ultralytics/coco/pruned40_
     quant-none ?recipe_type=transfer_learn \
  --recipe
     zoo:cv/detection/yolov5-n/pytorch/ultralytics/coco/pruned40_
     quant-none ?recipe_type=transfer_learn \
  --data /kaggle/working/data.yaml \
  --patience 0 \
  --hyp hyps/hyp.VOC.yaml
```

Para realizar o treinamento foi utilizado o Kaggle, que se trata de uma subsidiária da Google composta por uma comunidade *online* voltada para o aprendizado de máquina, permitindo

os usuários ter acesso a bases de dados para diversas problemáticas, recursos para construção de modelos inteligentes na própria plataforma, publicar base de dados de forma pública ou privada, dentre outros recursos (MOLTZAU, 2019). Então, após ser realizado o ajuste fino é obtido para cada modelo um arquivo no formato PyTorch (.pt) contendo os melhores valores para os pesos da rede neural para a detecção das placas.

Com o arquivo de pesos gerado a própria SparseML permite realizar a exportação do formato .pt para .onnx (*Open Neural Network Exchange*, ONNX[8]), que se trata de um formato de código aberto (*open-source*) para representação de modelos de aprendizado profundo. Essa exportação foi realizada através do comando CLI apresentado no Algoritmo 10.

Algoritmo 10 – Comando CLI da SparseML para exportação do modelo do formato .pt para .onnx

```
sparseml.yolov5.export_onnx \
  --weights yolov5_runs/train/exp/weights/best_pruned.pt \
  --dynamic
```

Após ser feita a exportação é possível já realizar inferências em novas imagens utilizando o modelo ajustado, e para isso há uma terceira ferramenta disponibilizada pela Neural Magic, que se trata da DeepSparse[9]. Esta ferramenta visa permitir ser alcançado o desempenho de inferência do ambiente com aceleração via GPU, em um ambiente apenas com CPU, por meio do aproveitamento do trabalho realizado ao tornar o modelo esparso.

Para realizar a inferência em uma imagem utilizando a DeepSparse foi utilizado o trecho apresentado em Código 1.

Código 1 – Código para realizar inferência com a DeepSparse

```
path_images = "/kaggle/input/dataset-ufs-alpr/license_plate/"
path_export =
   "/kaggle/working/yolov5_runs/train/exp/DeepSparse_Deployment/"
model_stub = f"{path_export}best_pruned.onnx"
images = [f"{path_images}vehicle_0_frame_1.png"]

yolo_pipeline = Pipeline.create(
    task="yolo",
    model_path=model_stub,
)

ini = time.time()
pipeline_outputs = yolo_pipeline(images=images, iou_thres=0.5,
   conf_thres=0.001) # INFERENCE WITH DEEP SPARSE
print(f'latency was {(time.time() - ini) * 1000}ms')
```

[8] Disponível em: <https://onnx.ai/>. Acesso em abril de 2023
[9] Disponível em: <https://docs.neuralmagic.com/products/deepsparse>. Acesso em abril de 2023

## 4.3 Tecnologias e Dispositivos Utilizados no Produto de Hardware

Quanto às tecnologias voltadas para o desenvolvimento concentram-se na construção do *firmware* para a câmera de baixo custo 4.3.1, implementação da comunicação de dados entre os dispositivos 4.3.2, e os dispositivos (4.3.3 e 4.3.4) e serviço em nuvem utilizado 4.3.5.

### 4.3.1 Arduino IDE

O Arduino IDE[10] é um ambiente de desenvolvimento voltado para a construção de *firmware* e carregamento deste diretamente para o microcontrolador das mais diversas placas de prototipagem disponíveis no mercado, indo além das próprias versões das placas Arduino. Além disso, dispõe de um compilador robusto, que se trata do GCC, para compilar o algoritmo desenvolvido pelo usuário e extrair um arquivo binário em formato Intel HEX, sendo este carregado para o microcontrolador.

Essa IDE permite, por exemplo, realizar a programação do Esp32Cam utilizando a mesma sintaxe utilizada para a construção de códigos para Arduino, dada por uma linguagem imperativa similar ao C++, com algumas funções e constantes predefinidas que tornam a programação amigável principalmente para iniciantes da área de sistemas embarcados.

Quando é criado um novo ambiente para a construção de um algoritmo este é chamado de *sketch*, que dispõe de duas funções mínimas que são *setup()* e *loop()*. A função *setup()* é executada apenas uma vez, logo ao ser iniciado o dispositivo, já a função *loop()* fica com sua rotina sendo executada a todo momento, em um ciclo infinito (PRATOMO; PERDANA, 2017).

Apesar da Espressif ter desenvolvido a Esp-IDF, que é a *framework* oficial para desenvolvimento de aplicações para as placas da família Esp, esta disponibiliza na própria documentação o link que leva a um arquivo JSON de configurações para adicionar suporte às placas da família Esp no Arduino IDE (KOYANAGI, 2018), além de disponibilizar diversas bibliotecas para serem adicionadas também ao Arduino IDE para tornar o desenvolvimento de soluções mais prático.

### 4.3.2 Mosquitto MQTT

O Mosquitto MQTT[11] é uma ferramenta voltada para possibilitar a criação de servidores para o protocolo MQTT nas versões 3.1, 3.1.1 e 5, além de oferecer suporte à TLS e SSL.

Por ser construído na linguagem C um dos diferenciais é a leveza, que torna a sua instalação possível nos mais diversos tipos de dispositivos, desde um Raspberry Pi Zero W até computadores robustos, dispondo de bons resultados nos cenários de *hardware* limitado se comparado a outras soluções (TORRES; ROCHA; SOUZA, 2016).

[10] Disponível em: <https://www.arduino.cc/en/software>. Acesso em novembro de 2022
[11] Disponível em: <https://mosquitto.org/>. Acesso em novembro de 2022

Porém, o ônus para esse ambiente de comunicação ser mais leve é a perda de alguns recursos extras oferecidos por outros servidores, como interfaces web para gerenciamento de tópicos e clientes. Além disso, também tem a questão deste não aproveitar o paralelismo em processadores com muitos núcleos, não sendo também possível construir *cluster* com ele, que é um dos recursos que outros servidores como o HiveMQ[12] oferecem.

### 4.3.3 Esp32Cam

O Esp32Cam trata-se de uma placa para prototipagem da família Esp da Espressif. Essa placa tem como chip o ESP32-S, dispõe de suporte para algumas lentes como a OV2640, cartão microSD e contém vários GPIOs para estabelecer conexão com periféricos (SANTOS, 2019). Além destes recursos, outros são apresentados a seguir:

a) contém um módulo 802.11b/g/n Wi-Fi BT SoC;

b) contém 520 KB SRAM interna e 4 MB de PSRAM externa, onde esta última possibilita trabalhar com imagens em resoluções grandes que demandam de muito espaço alocado na memória;

c) oferece suporte à UART/SPI/I2C/PWM/ADC/DAC;

d) suporta as câmeras OV2640 e OV7670;

e) suporta modos *sleep*, permitindo economia energética em momentos ociosos;

f) tem incorporado Lwip and FreeRTOS;

g) contém modos de operação STA/AP/STA+AP;

Figura 59 – Esp32Cam com módulo para carregamento de *firmware* e câmera

Fonte: Usina Info

[12] Disponível em: <https://www.hivemq.com/>. Acesso em novembro de 2022

Porém, diferente de outras placas da família Esp, ele não vem com o conector microUSB embutido para realizar a conexão diretamente com o computador para carregar algum *firmware* desenvolvido. Mas é possível utilizar módulos separados para isto, ou até mesmo outras placas da família Esp, como a Esp32 Dev Module que mediante conexão entre as pinagens das duas placas pode ser usada carregando o *firmware* para o Esp32Cam.

### 4.3.4 Jetson Nano

O Jetson Nano[13] trata-se de uma placa robusta voltada para aplicações de computação de aprendizado profundo em dispositivos de sistema embarcado. Há a possibilidade de usá-la em diversas áreas, mas geralmente o seu foco está na área de robótica, principalmente voltado para visão computacional (PETTIGREW, 2020).

O que o torna querido é o fato de ser possível trabalhar com redes neurais profundas paralelamente (ALECRIM, 2020), oferecendo assim um melhor desempenho para lidar, por exemplo, com a classificação de imagem, detecção de objetos e processamento de linguagem natural devido ao aceleramento via GPU integrada a placa, a qual é usada para realizar os cálculos matemáticos necessários pelo modelo. Além disso, dispõe de *JetPacks*, sendo os ambientes de desenvolvimento da plataforma baseados na distribuição Ubuntu.

Figura 60 – NVidia Jetson Nano

Fonte: NVidia

Quanto aos recursos oferecidos pela placa, estes são listados a seguir:

[13] Disponível em: <https://www.nvidia.com/pt-br/autonomous-machines/embedded-systems/jetson-nano/>. Acesso em novembro de 2022

a) dispõe do processador quad-core (Cortex-A57) de 1,43 GHz;

b) tem como GPU a NVidia Maxwell de 128 núcleos;

c) oferece 2 GB de memória RAM LPDDR4;

d) o armazenamento por meio de um cartão microSD, sendo aconselhável que este tenha no mínimo 32GB de capacidade para suportar o sistema operacional e às aplicações padrões do JetPack;

e) oferece para conectividade HDMI, USB 3.0 tipo A, duas portas USB 2.0 tipo A, micro-USB 2.0, uma entrada Gigabit Ethernet com suporte 802.11ac;

f) quanto a conectividade para periféricos tem 40 pinos para GPIO, I2C, I2S, SPI e UART.

### 4.3.5 Ampere A1 Compute

O Ampere A1 Compute[14] é um serviço em nuvem oferecido pela *Oracle Cloud Infrastructure*, o qual pode ser trabalhado com sistemas operacionais tanto da Microsoft, como distribuições Linux.

Ele dispõe de processadores Altra da Ampere com até 80 núcleos por CPU, que são processadores baseados na arquitetura ARM capazes de funcionar na frequência máxima de 3,0 Ghz, sendo que cada núcleo tem seu próprio I-cache L1 de 64 KB, cache D L1 de 64 KB e cache D L2 de 1 MB. Há possibilidade de dimensionar o servidor, podendo ser usado entre 1 e 80 OCPUs, e entre 1 e 64 GB de memória por núcleo.

É possível utilizar parte das configurações possíveis com o Ampere A1 Compute gratuitamente para sempre, por ser um dos serviços inclusos no plano sempre gratuito (*Always Free*) da OCI (*Oracle Cloud Infrastructure*), sendo possível dispor de um servidor com até 4 OCPUs e 24GB de memória RAM. Tendo direito também a até 200 GB de armazenamento em bloco via SSD, 4 Gbps de largura de banda e 3000 horas gratuitas.

[14] Disponível em: <https://www.oracle.com/br/cloud/compute/arm/>. Acesso em novembro de 2022

# 5

# Resultados Alcançados

Neste Capítulo, são primeiramente apresentados os resultados alcançados com as configurações descritas nas Subseções 4.1.1 e 4.1.2, sendo na Seção 5.1 demonstrados os resultados obtidos para a configuração que utiliza a câmera IP de baixo custo junto ao servidor Ampere A1 Compute. Já na Seção 5.2, são demonstrados e discutidos os resultados para a configuração envolvendo também a câmera IP de baixo custo, mas com o Jetson Nano como unidade de processamento.

A parte de otimizações é iniciada na Seção 5.3, a partir dos resultados obtidos com as otimizações providas pela implementação do módulo de rastreio ao sistema de ALPR, tanto para a configuração envolvendo o servidor Ampere A1 Compute, como a que envolve o Jetson Nano. Por último, na Seção 5.4, são apresentadas os resultados para avaliação do desempenho das versões comprimidas do modelo YOLOv5 Nano, além de uma comparação entre elas e dois modelos densos utilizados também utilizados para detecção das placas veiculares.

## 5.1 Resultados com câmera IP de baixo custo e sistema de ALPR em nuvem

### 5.1.1 Taxas de quadros do fluxo de imagens entre o Esp32Cam e o servidor Ampere A1 Compute

Para realizar a medição da taxa de quadros associada à cada uma das resoluções de imagens possíveis a serem trabalhadas com o Esp32Cam foi desenvolvido um algoritmo para ser executado no servidor, o qual atua a partir dos dados recebidos pelos tópicos *camera/data/cam0001* e *camera/commands/cam0001*.

Este algoritmo identifica na mensagem enviada para o Esp32Cam via protocolo MQTT, através do tópico *camera/commands/cam0001*, qual resolução de imagem este irá trabalhar. Então

é contabilizada a quantidade de imagens que o Esp32Cam envia para o servidor através do tópico *camera/data/cam0001* enquanto é cronometrado o tempo até que seja enviado um novo comando para o Esp32Cam trocar de resolução, que acontece em intervalos de 5 minutos.

Ao ser enviado um novo comando é realizado o cálculo da taxa de quadros para a resolução que estava sendo avaliada, sendo feita a divisão da quantidade de imagens recebidas do Esp32Cam pelo tempo cronometrado em segundos.

Fazendo isso com o Esp32Cam conectado a uma rede móvel 3G, e também mediante WiFi 802.11n em uma rede banda larga, foram identificados os resultados apresentados na Tabela 10. Vale ressaltar que a largura de banda utilizada para a conexão com WiFi 802.11n foi em torno de 84 Mbps para *download* e 83 Mbps para *upload* com uma incerteza de até 10 Mbps.

Tabela 10 – Taxa de quadros do fluxo de imagens entre o Esp32Cam e o servidor Ampere A1 Compute

| **Resolução (px)** | **Taxa de quadros em conexão móvel 3G (FPS)** | **Taxa de quadros em conexão banda larga (FPS)** |
|---|---|---|
| **160x120** | 4,39 | 17,57 |
| **240 x 160** | 3,54 | 12,08 |
| **320 x 240** | 2,08 | 9,55 |
| **352 x 288** | 1,47 | 8,27 |
| **640 x 480** | 0,87 | 3,75 |
| **800 x 600** | 0,62 | 3,43 |
| **1024 x 768** | 0,95 | 2,71 |
| **1280 x 1024** | 0,53 | 2,80 |
| **1600 x 1200** | 0,37 | 1,73 |

Fonte: Autor.

Percebe-se através dos resultados da Tabela 10 uma discrepância de desempenho considerável entre o Esp32Cam conectado a uma rede banda larga se comparado a este conectado a uma rede móvel.

Ao considerar que o desempenho de uma rede banda larga tende a ser superior à rede móvel, esses resultados não aparentam possíveis problemas no Esp32Cam, mas ao analisar os resultados apresentados pelo desempenho da configuração utilizando o Jetson Nano (ver Seção 5.2.1) fica mais aparente uma possível instabilidade por parte da placa.

Além disso, vale ressaltar que de acordo foram sendo determinadas resoluções de imagens maiores do que 800x600 para as configurações do Esp32Cam, foram realizados ajustes no nível de compressão para que fosse evitado problemas de mau funcionamento da placa por falta de espaço.

Percebe-se também que o desempenho apresentado nestas medições limita o desempenho do sistema de ALPR, pois este consegue processar a uma taxa de quadros de 3,78 FPS. Ou seja, ao considerar o uso da rede móvel para conectar o Esp32Cam com a internet, a qual foi utilizada para os testes na UFS, é impraticável a detecção em tempo hábil.

### 5.1.2 Desempenho da configuração com o sistema de ALPR em nuvem

Os testes para esta configuração aconteceram na portaria principal de veículos da UFS no turno da manhã do dia 24/08/2022 e no turno da tarde do dia 25/08/2022. Em cada dia foram realizados os testes com duração de aproximadamente 2 horas e 30 minutos. Na Figura 61, é possível observar a câmera instalada na portaria da UFS.

Figura 61 – Configuração durante testes na portaria da UFS

Fonte: Autor.

Para coletar os resultados o sistema de ALPR foi ajustado para realizar o salvamento automático de imagens apenas dos veículos que, considerando o resultado do sistema de ALPR, a cadeia de caracteres tivesse comprimento válido para uma placa de veículo. Então, com estas informações coletadas foram identificadas manualmente em cada imagem com auxílio do aplicativo SINESP Cidadão[1] a cadeia de caracteres real de cada placa para inferir a acurácia do sistema de ALPR sob esse contexto. Na Tabela 11, são apresentados os resultados encontrados.

[1] Disponível em: <https://www.gov.br/pt-br/apps/sinesp-cidadao>. Acesso em maio de 2023

Tabela 11 – Resultados alcançados ao trabalhar com o servidor Ampere A1 Compute

| **Resolução (px)** | **núm. de detecções corretas** | **núm. de detecções incorretas** | **núm. de placas ilegíveis** | **núm. de veículos** |
|---|---|---|---|---|
| **640 x 480** | 5 | 27 | 0 | 30 |
| **800 x 600** | 36 | 117 | 5 | 144 |

Fonte: Autor.

É percebido pela Tabela 11 um alto número de reconhecimentos incorretos que pode ser devido à baixa qualidade de imagem nos registros realizados pelo Esp32Cam, principalmente quando trabalhada em resoluções menores do que 800x600. Posteriormente será observado mediante resultados utilizando o Jetson Nano que ao aumentar a resolução há uma melhora significativa na quantidade de acertos.

## 5.2 Resultados com câmera IP de baixo custo e sistema de ALPR em um Jetson Nano

### 5.2.1 Taxas de quadros do fluxo de imagens entre o Esp32Cam e o Jetson Nano

Para realizar as medições das taxas de quadros considerando o uso de um dispositivo de sistema embarcado, a única diferença em relação ao procedimento realizado na configuração com o servidor Ampere A1 Compute é que o algoritmo, no caso atual, realiza medições através da execução do sistema de ALPR no Jetson Nano com o Raspberry Pi Zero W atuando com o Mosquitto MQTT como servidor MQTT. Considerando isso, a Tabela 12 apresenta os resultados das medições para o uso de uma rede móvel 3G e uma rede sem fio banda larga.

Nestes resultados apresentados é mais visível a inconsistência do desempenho ao variar as resoluções, pois é esperado que o Esp32Cam conectado a uma conexão banda larga estável, sem interferência por uso de dispositivos *bluetooth* próximos, fosse melhor que os resultados encontrados para o uso de uma conexão com rede móvel 3G.

Dentre as medições para as taxas de quadros com a conexão banda larga é percebido que o uso da resolução 240x160 obteve um resultado inferior aos das resoluções 320x240 e 352x288. Isso supostamente indica uma perda de desempenho da própria rede, que em outros momentos foram realizados novos testes dos quais a inconsistência permaneceu, levando a hipótese de ser algo relacionado com a própria placa do Esp32Cam.

Tabela 12 – Taxas de quadros do fluxo de imagens entre o Esp32Cam e o Jetson Nano

| **Resolução (px)** | ***Framerate* em conexão móvel 3G (FPS)** | ***Framerate* em conexão banda larga (FPS)** |
|---|---|---|
| **160x120** | 24,85 | 20,44 |
| **240 x 160** | 24,82 | 11,01 |
| **320 x 240** | 24,60 | 14,84 |
| **352 x 288** | 23,00 | 12,97 |
| **640 x 480** | 12,15 | 5,17 |
| **800 x 600** | 9,45 | 4,53 |
| **1024 x 768** | 6,21 | 3,21 |
| **1280 x 1024** | 6,20 | 2,45 |
| **1600 x 1200** | 5,75 | 3,00 |

Fonte: Autor.

Devido essa inconsistência foram realizadas pesquisas em busca de soluções para o problema descrito, sendo encontrado que para uma melhor estabilidade de conexão *WiFi* com o Esp32Cam é necessário recorrer ao uso de uma antena externa. Esta é soldada diretamente na placa que já tem a trilha disponível para realizar a ligação com a antena, desabilitando o uso da antena integrada à placa.

Vale ressaltar que mesmo com este problema encontrado nos resultados as medições principalmente para a conexão com a rede móvel 3G são boas. Pois, ao considerar que a taxa para o sistema de ALPR executar completamente sem otimizações é de 5,66 FPS, e foi alcançado na maior resolução de imagem uma taxa de quadros de aproximadamente 5,75 FPS na captura e envio das imagens para o Jetson Nano, o desempenho do Esp32Cam não limita o desempenho do sistema de ALPR.

### 5.2.2 Desempenho da configuração com o sistema de ALPR em um Jetson Nano

A realização dos testes para esta configuração aconteceu também na portaria principal de veículos da UFS, mas no dia 26/08/2022 no turno da tarde, também com duração de aproximadamente 2 horas e 30 minutos.

Os resultados obtidos são apresentados na Tabela 13, na qual pode ser observado que para resoluções maiores como 1024x768 os resultados melhoram se comparado às outras resoluções trabalhadas.

Tabela 13 – Resultados alcançados ao trabalhar com o Jetson Nano

| **Resolução (px)** | **núm. de detecções corretas** | **núm. de detecções incorretas** | **núm. de placas ilegíveis** | **núm. de veículos** |
|---|---|---|---|---|
| **640 x 480** | 6 | 33 | 1 | 15 |
| **800 x 600** | 74 | 510 | 13 | 248 |
| **1024 x 768** | 57 | 151 | 3 | 75 |

Fonte: Autor.

## 5.3 Resultados do Processamento ALPR com Módulo de Rastreio

Para avaliação dos ganhos obtidos com o uso do módulo de rastreio junto ao *pipeline* principal do sistema de ALPR foi considerado um cenário de teste com 5 vídeos gravados na portaria principal de veículos da UFS. Estes vídeos registraram os veículos a partir de diferentes posicionamentos, oferecendo bons contextos de testes. Foi escolhido o uso de vídeos em vez de câmeras em tempo real porque é esperado que tanto o sistema de ALPR no servidor Ampere A1 Compute como no Jetson Nano analisassem os mesmos dados de testes.

As informações coletadas foram inseridas em uma planilha após o processamento de cada imagem. Quanto às informações coletadas foram consideradas a identificação se o módulo de rastreio estava sendo executado na imagem, a numeração do veículo para verificar a atuação do contador de veículos, a cadeia de caracteres inferida da placa, as latências de cada etapa do *pipeline* do sistema, a data e hora que ocorreu o processamento da imagem e a placa resultante do processamento de redundância temporal. Este último foi sendo definido no fim do rastreio de cada veículo, e associado por uma coluna aos registros coletados relacionados àquele veículo rastreado.

Também foram salvas as primeiras imagens detectadas para cada veículo, quando todo o *pipeline* principal do sistema é executado. Isso foi necessário para permitir identificar o ganho de acurácia provido pelo uso do tratamento de redundância temporal em relação à execução completa do *pipeline* principal.

No Jetson Nano foram realizadas no total 22640 detecções considerando todos os quadros dos 5 vídeos, destas apenas 478 detecções foram por execução completa do *pipeline* principal do sistema de ALPR, e as 22162 detecções restantes foram através do módulo de rastreio. Considerando as inferências realizadas com o módulo de rastreio ativo foi obtida uma latência média de 123,69 ms, enquanto para a execução completa do *pipeline* sem o módulo de rastreio a latência média foi de 184,97 ms.

Já com o servidor Ampere A1 Compute foram realizadas 22541 detecções, sendo 473 com o *pipeline* principal com latência média de 275,46 ms, e 22068 com o módulo de rastreio ativo com latência média de 184,18 ms.

Na Tabela 14, é apresentada a latência média em cada etapa de detecção do *pipeline* principal do sistema tanto ao ser executado no Jetson Nano, como no servidor Ampere A1 Compute.

Tabela 14 – Latências das etapas do *pipeline* principal do sistema de ALPR

| **Unidade de Processamento** | **det. de veículo (ms)** | **det. de placa (ms)** | **rec. de caracteres (ms)** |
|---|---|---|---|
| **Jetson Nano** | 57,41 | 55,88 | 71,68 |
| **Ampere A1 Compute** | 92,48 | 89,78 | 93,20 |

Fonte: Autor.

Quanto às funcionalidades, no contador de veículos foi observado um bom funcionamento ao considerar o tratamento de inconsistências geradas por oclusão da placa, perda temporária de sinal com a câmera, dentre outras circunstâncias que levam ao encerramento do módulo de rastreio precocemente. No entanto, ainda persistem inconsistências devido o reconhecedor de caracteres por vezes retornar sequências de cadeias de caracteres de placas completamente diferentes das reais a depender da posição relativa da câmera em relação ao veículo. Assim como ocorre quando estão sendo processadas imagens de motocicletas.

Ao ser analisado o trabalho desenvolvido por Silva (2022), é percebido que na construção da base de dados UFS-ALPR foram conseguidas poucas amostras de motocicletas se comparado ao montante de amostras de carros, o que explica o mau desempenho do sistema de ALPR quando o veículo é uma motocicleta.

Quanto a funcionalidade de redundância temporal, para avaliar a acurácia entre o uso e o não uso desta foi considerado todos os registros da planilha de métricas coletados quando todo o *pipeline* foi executado no Jetson Nano, ou seja, foram consideradas 478 detecções. Porém, dentre estas, algumas detecções foram removidas por serem considerados elementos presentes em adesivos na lateral dos veículos, e também em casos que alguns caracteres da placa estavam sob oclusão total. Com isso foram considerados então 471 registros para avaliação.

Então, foi analisada imagem por imagem de forma supervisionada para identificar e anotar na planilha em uma nova coluna a placa real para cada um dos 471 registros de veículos. Foi verificado após isso a correspondência entre as cadeias de caracteres de placas inferidas pelo modelo com as cadeias de caracteres reais, e logo depois as cadeias de caracteres resultantes do tratamento de redundância temporal com as cadeias de caracteres reais. Foram então extraídos os resultados apresentados na Tabela 15.

Tabela 15 – Comparação de resultados entre a acurácia com o tratamento de redundância temporal e a acurácia com o *pipeline* principal do sistema de ALPR

| **Tratamento de Redundância Temporal** | ***strings* incorretas** | ***strings* corretas** | **acurácia (%)** |
|---|---|---|---|
| **Desativada** | 343 | 128 | 27,18 |
| **Ativada** | 207 | 264 | 56,05 |

Fonte: Autor.

Aproveitando estes registros utilizados para análise de desempenho do tratamento de

redundância temporal, foi verificado também quantos destes 471 registros se tratam de veículos duplicados que levam à inconsistência do contador de veículos. O resultado foi que 79 dos 471 registros são de veículos que já tinham sido registrados pelo menos uma vez, mas que por serem motos, ou então carros que passam pela câmera em um posicionamento ruim para a realização do processamento, dificultaram o tratamento desenvolvido para verificação e correção de inconsistências.

## 5.4 Desempenho com a Compressão do Modelo YOLOv5 Nano

Para ser possível avaliar o desempenho entre os modelos comprimidos e os densos foi adaptado um *pipeline* para definir os valores das métricas a partir da inferência de 1530 imagens da base de dados UFS-ALPR.

Esse *pipeline* identifica para cada uma das 1530 imagens qual é a verdadeira caixa delimitadora do veículo, necessária para realizar o recorte sempre dos mesmos veículos para avaliação em diferentes modelos, simulando assim o procedimento natural do *pipeline* principal de ALPR, no qual o detector de veículo atua identificando a região de interesse para detecção da placa.

Então, a imagem recortada do veículo é encaminhada para a realização da inferência pelo detector de placas avaliado, coletando a caixa delimitadora predita da placa do veículo para realização do cálculo do IOU, e logo em seguida a definição dos verdadeiros positivos (TP), falsos positivos (FP), falsos negativos (FN), precisão, *recall*, IOU média (mIOU), acurácia da classificação da placa entre moto ou carro (acc m/c), acurácia da classificação do tipo de placa entre *old* e mercosul (acc o/m), e a taxa de quadros alcançada na unidade FPS.

Para avaliação e comparação de desempenho foram considerados os modelos densos YOLOv5 Nano e YOLOv8 Nano, sendo este último treinado para se obter uma visão de desempenho através da versão mais recente nas comparações. Já quanto aos modelos comprimidos, foram considerados o YOLOv5 com 30% de poda, YOLOv5 com 30% de poda e com quantização, YOLOv5 com 40% de poda, e YOLOv5 com 40% de poda com quantização, sendo os resultados coletados apresentados na Tabela 16. Vale ressaltar sobre a inferência dos modelos comprimidos, pois foi utilizado o DeepSparse como é recomendado pela Neural Magic para extração de bons resultados. Além disso, foi considerado apenas o cenário de detecção utilizando CPU, sem aceleração por GPU, no qual é o cenário que mais é destacado pela Neural Magic como o que mais apresenta diferenças positivas para os modelos comprimidos através da SparseML se comparado com os modelos densos.

Na Tabela 17, são apresentados os resultados obtidos no final do treinamento destes modelos, onde é retornado pelo próprio procedimento de treinamento as métricas precisão, *recall* e mAP50.

Tabela 16 – Resultados alcançados pelos modelos densos e pelos modelos comprimidos

| **Modelo** | **TP** | **FP** | **FN** | **precisão (%)** | ***recall* (%)** | **mIOU (%)** | **acc c/m (%)** | **acc o/m (%)** | ***framerate* (FPS)** |
|---|---|---|---|---|---|---|---|---|---|
| **YOLOv5 Nano** | 1525 | 91 | 5 | 94,4 | 99,7 | 85,8 | 99,7 | 96,6 | 17,8 |
| **YOLOv8 Nano** | 1493 | 0 | 37 | 1 | 97,6 | 87,4 | 99,8 | 99,7 | 7,6 |
| **YOLOv5 Nano 30% s/ quant.** | 1480 | 50 | 50 | 96,7 | 96,7 | 82,3 | 99,9 | 93,7 | 8,7 |
| **YOLOv5 Nano 30% c/ quant.** | 1480 | 50 | 50 | 96,7 | 96,7 | 82,3 | 99,9 | 93,7 | 8,5 |
| **YOLOv5 Nano 40% s/ quant.** | 1523 | 7 | 7 | 99,5 | 99,5 | 85,7 | 99,7 | 89,4 | 8,3 |
| **YOLOv5 Nano 40% c/ quant.** | 1522 | 8 | 8 | 99,5 | 99,5 | 85,7 | 99,7 | 89,2 | 8,7 |

Fonte: Autor.

Tabela 17 – Resultados de treinamento alcançados pelos modelos densos e pelos modelos comprimidos

| **Modelo** | **precisão (%)** | ***recall* (%)** | **mAP50 (%)** |
|---|---|---|---|
| **YOLOv5 Nano** | 99,6 | 98,9 | 99,5 |
| **YOLOv8 Nano** | 99,8 | 98,1 | 99,2 |
| **YOLOv5 Nano 30% s/ quant.** | 98,7 | 97,4 | 99,1 |
| **YOLOv5 Nano 30% c/ quant.** | 98,7 | 97,4 | 99,1 |
| **YOLOv5 Nano 40% s/ quant.** | 99,1 | 97,5 | 99,4 |
| **YOLOv5 Nano 40% c/ quant.** | 99,2 | 97,5 | 99,5 |

Fonte: Autor.

Também foi avaliado o tamanho final dos modelos considerando o arquivo no formato PyTorch (.pt), gerado pelo próprio processo de treinamento e contém os melhores valores de pesos. E também foi considerado o formato ONNX (.onnx) que foi utilizado nos modelos para a realização dos testes através do Kaggle. Com isso, foram obtidos os resultados apresentados na Tabela 18.

Tabela 18 – Tamanhos dos modelos densos e dos modelos comprimidos

| **Modelo** | **.pt (MB)** | **.onnx (MB)** |
|---|---|---|
| **YOLOv5 Nano** | 3,81 | 7,12 |
| **YOLOv8 Nano** | 6,22 | 12,24 |
| **YOLOv5 Nano 30% s/ quant.** | 3,68 | 7,17 |
| **YOLOv5 Nano 30% c/ quant.** | 3,68 | 7,17 |
| **YOLOv5 Nano 40% s/ quant.** | 3,68 | 7,17 |
| **YOLOv5 Nano 40% c/ quant.** | 14,47 | 7,17 |

Fonte: Autor.

Como pode ser percebido tanto pelos resultados de desempenho nas inferências como no tamanho final dos modelos, não foram obtidos bons resultados ao utilizar as ferramentas

disponibilizadas pela Neural Magic no cenário deste trabalho. Vale destacar que as recomendações da Neural Magic para como deve ser feito o treinamento para se obter o ajuste fino do modelo, e a utilização do DeepSparse para realização das inferências na CPU com os modelos comprimidos, foram todas seguidas.

Outro ponto importante de destacar é um problema ocorrido durante o ajuste fino de um modelo pré-treinado para se obter o modelo YOLOv5 Nano com 40% de poda e quantização. Nas épocas finais do treinamento destinadas para aplicação da quantização no modelo ocorreu um erro interno da SparseML, e é possível perceber o impacto deste erro ao observar o tamanho do arquivo contendo os pesos no formato .pt na Tabela 18. Foram realizadas novas tentativas em outros momentos, mas o mesmo problema persistiu sem qualquer caso similar com solução encontrado por meio de pesquisas na internet.

# 6

# Considerações Finais

Neste trabalho, o objetivo principal foi aprimorar um sistema de reconhecimento automático de placas veiculares para torná-lo comercialmente viável. Os objetivos iniciais foram totalmente trabalhados ao longo do projeto.

Inicialmente, foi desenvolvida e testada a construção de uma câmera de baixo custo usando o Esp32Cam. No entanto, verificou-se que essa câmera não apresentava bom desempenho para cenários gerais de uso, sendo mais recomendado o uso de câmeras disponíveis no mercado para obter uma solução mais robusta. No entanto, em cenários mais restritos, onde é possível ajustar melhor a posição da câmera e garantir uma boa iluminação ambiente para capturar imagens nítidas da placa, essa câmera pode ser uma solução viável. Além disso, conectá-la a uma antena externa pode melhorar seus resultados.

Foram desenvolvidas interfaces de comunicação utilizando os protocolos MQTT e RTSP para permitir a integração do sistema com câmeras compatíveis disponíveis no mercado. Isso possibilitou a realização de testes em tempo real, tanto para verificar o desempenho do módulo de rastreio usando câmeras instaladas na portaria da prefeitura da cidade de Aracaju, Sergipe, como para testar as funcionalidades relacionadas a esse módulo. Essa integração também permitiu que o sistema de reconhecimento automático de placas trabalhasse em conjunto com a câmera desenvolvida neste trabalho.

No que diz respeito ao módulo de rastreio, também foi realizado aprimoramento do sistema, resultando em uma redução média de latência de 61,28 ms para a configuração com o Jetson Nano e 91,28 ms para a configuração com o servidor Ampere A1 Compute da Oracle, em relação ao tempo total necessário para executar todo o *pipeline* principal do sistema. Além disso, o uso do tratamento de redundância temporal melhorou a acurácia dos resultados, resultando em um aumento de 28,87% na configuração com o Jetson Nano ao processar cinco vídeos gravados na portaria da Universidade Federal de Sergipe. No entanto, o desempenho do reconhecedor de caracteres afeta o funcionamento do contador de veículos, sendo necessário adicionar mais

amostras à base de dados UFS-ALPR e realizar um novo treinamento dos modelos para torná-los mais robustos na detecção de motocicletas e para obter um bom desempenho em ambientes noturnos, propiciando um aumento ainda maior da acurácia dos resultados.

Quanto à compressão do modelo YOLOv5 Nano, embora as ferramentas da Neural Magic não tenham sido bem-sucedidas, esse foi o primeiro passo na busca pela redução da latência das inferências. Recomenda-se continuar essa etapa estudando métodos ou ferramentas de compressão para futuros trabalhos de desenvolvimento do sistema de ALPR. É importante mencionar que, além dos modelos podados pela técnica de ajuste fino da Neural Magic, também foram realizados testes criando um modelo escasso do zero, mas os resultados obtidos foram semelhantes aos modelos com uma taxa de aproximadamente 11 FPS, o que não é relevante para este trabalho, pois houve uma piora no desempenho em comparação ao modelo denso já utilizado no sistema.

# Referências

ABBASI, S.; REZAEIAN, M. Visual object tracking using similarity transformation and adaptive optical flow. *Multimedia Tools and Applications*, 2021. Citado 3 vezes nas páginas 52, 53 e 55.

ALECRIM, E. *Jetson Nano 2 GB é uma placa da Nvidia para robótica e IA*. 2020. Acessado em 11 de Novembro de 2022. Disponível em: <https://tecnoblog.net/noticias/2020/10/06/nvidia-jetson-nano-2-gb-computador-robotica-inteligencia-artificial/>. Citado na página 113.

AMARAL, C. *O que é o protocolo RTSP e para que ele serve?* 2022. Acessado em 24 de Março de 2023. Disponível em: <https://k2ponto.com.br/blog/o-que-e-o-protocolo-rtsp-e-para-que-ele-serve/>. Citado na página 75.

BADRINARAYANAN, V.; KENDALL, A.; CIPOLLA, R. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. Citado 2 vezes nas páginas 42 e 44.

BOBRIAKOV, I. *Artificial Intelligence vs. Machine Learning vs. Deep Learning. What is the difference?* 2019. Acessado em 26 de Março de 2023. Disponível em: <https://medium.com/activewizards-machine-learning-company/artificial-intelligence-vs-machine-learning-vs-deep-learning-what-is-the-difference-a5e2bc8b835f>. Citado na página 27.

BOCHKOVSKIY, A. et al. Yolov4: Optimal speed and accuracy of object detection. *arXiv: Computer Vision and Pattern Recognition*, 2020. Citado na página 50.

CHOLLET, F. *Deep Learning with Python*. [S.l.]: Manning, 2017. ISBN 9781617294433. Citado na página 23.

CHOUHAN, V. et al. A novel transfer learning based approach for pneumonia detection in chest x-ray images. *Applied Sciences*, v. 10, n. 2, 2020. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/10/2/559>. Citado na página 61.

COPE, S. *Beginners Guide To The MQTT Protocol*. 2018. Acessado em 2 de Setembro de 2022. Disponível em: <http://www.steves-internet-guide.com/mqtt/>. Citado na página 70.

COPE, S. *Introduction to MQTT-SN (MQTT for Sensor Networks)*. 2018. Acessado em 2 de Setembro de 2022. Disponível em: <http://www.steves-internet-guide.com/mqtt-sn/>. Citado na página 67.

CORPORATION, N. C. *Netscape Announces New Real-Time Audio and Video Framework for Internet Applications*. 1996. Acessado em 24 de Março de 2023. Disponível em: <https://www.cs.columbia.edu/~hgs/rtp/newsrelease81.html>. Citado na página 76.

CRAGGS, I. *Towards The Next Version of MQTT*. 2016. Acessado em 2 de Setembro de 2022. Disponível em: <https://www.eclipse.org/community/eclipse_newsletter/2016/september/article3.php>. Citado na página 70.

CRAGGS, I. *MQTT vs AMQP for IoT*. 2022. Acessado em 10 de Maio de 2023. Disponível em: <https://www.hivemq.com/blog/mqtt-vs-amqp-for-iot/>. Citado na página 68.

DEMIR, H. *The History of RTSP Streaming*. 2021. Acessado em 24 de Março de 2023. Disponível em: <https://dev.to/hamitdemir/the-history-of-rtsp-streaming-3060>. Citado na página 75.

EMEZI, S. *RTSP Streaming- Using Real Time Streaming Protocol for your events in 2022*. 2022. Acessado em 24 de Março de 2023. Disponível em: <https://inevent.com/blog/tech-and-trends/rtsp-streaming.html#History_of_RTSP_streaming>. Citado na página 76.

EVERINGHAM, M. et al. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 2015. Citado na página 32.

FARHOOD, H. et al. Recent advances of image processing techniques in agriculture. In: ____. [S.l.: s.n.], 2022. p. 129–153. ISBN 9780323905084. Citado na página 61.

FELZENSZWALB, P. F. et al. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 32, n. 9, p. 1627–1645, 2010. Citado na página 48.

FIAZ, M. et al. Handcrafted and deep trackers: Recent visual object tracking approaches and trends. *arXiv: Computer Vision and Pattern Recognition*, 2018. Citado 2 vezes nas páginas 54 e 55.

FONSECA, G. *Como Identificar A Cidade Na Placa Mercosul*. 2023. Acessado em 13 de Março de 2023. Disponível em: <https://doutormultas.com.br/como-identificar-a-cidade-na-placa-mercosul/#O_que_e_a_placa_Mercosul>. Citado na página 66.

G1. *Grupo que invadia condomínios de luxo em vários estados: veja o que se sabe até agora*. 2022. Acessado em 4 de Setembro de 2022. Disponível em: <https://g1.globo.com/es/espirito-santo/noticia/2022/07/25/grupo-que-invadia-condominios-de-luxo-em-varios-estados-veja-o-que-se-sabe-ate-agora.ghtml>. Citado na página 19.

GIRSHICK, R. B. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. Disponível em: <http://arxiv.org/abs/1311.2524>. Citado na página 48.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado 5 vezes nas páginas 24, 25, 27, 30 e 31.

GROUP, O. M. *What is DDS?* 2023. Acessado em 10 de Maio de 2023. Disponível em: <https://www.hivemq.com/blog/mqtt-vs-amqp-for-iot/>. Citado na página 68.

HAN, S.; MAO, H.; DALLY, W. J. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. 2016. Citado na página 63.

HENRIQUES, J. F. et al. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015. Citado na página 56.

HIVEMQ. *Keep Alive and Client Take-Over - MQTT Essentials Part 10*. 2015. Acessado em 2 de Setembro de 2022. Disponível em: <https://www.hivemq.com/blog/mqtt-essentials-part-10-alive-client-take-over/>. Citado na página 73.

HIVEMQ. *Last Will and Testament - MQTT Essentials: Part 9*. 2015. Acessado em 2 de Setembro de 2022. Disponível em: <https://www.hivemq.com/blog/mqtt-essentials-part-9-last-will-and-testament/#:~:text=In/%20MQTT/%2C/%20you/%20use/%20the,flag/%2C/%20QoS/%2C/%20and/%20payload.> Citado na página 74.

INFONET. *Carro roubado é encontrado dentro de condomínio*. 2018. Acessado em 21 de Setembro de 2022. Disponível em: <https://infonet.com.br/noticias/cidade/carro-roubado-e-encontrado-dentro-de-condominio/>. Citado na página 20.

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 na página 51.

KAVITHA, A. S. et al. Text segmentation in degraded historical document images. *Egyptian Informatics Journal*, 2016. Citado na página 33.

KOYANAGI, F. *Instalando ESP32 no Arduino IDE: Método fácil*. 2018. Acessado em 4 de Novembro de 2022. Disponível em: <https://www.fernandok.com/2018/09/instalando-esp32-no-arduino-ide-metodo.html?m=1&gclid=CjwKCAjw8JKbBhBYEiwAs3sxN7u1VF9RBS_KlDbEDlEetbOek3SE7DIGx8j7baD9h8J1rXtMbrESehoC9wsQAvD_BwE>. Citado na página 111.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Communications of The ACM*, 2017. Citado na página 39.

KRSTINIć, D. et al. Multi-label classifier performance evaluation with confusion matrix. *Computer Science and Information Technology*, 2020. Citado 2 vezes nas páginas 34 e 35.

KURTZ, M. et al. Inducing and exploiting activation sparsity for fast inference on deep neural networks. In: III, H. D.; SINGH, A. (Ed.). *Proceedings of the 37th International Conference on Machine Learning*. Virtual: PMLR, 2020. (Proceedings of Machine Learning Research, v. 119), p. 5533–5543. Disponível em: <http://proceedings.mlr.press/v119/kurtz20a.html>. Citado na página 103.

LAROCA, R. et al. A robust real-time automatic license plate recognition based on the yolo detector. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2018. p. 1–10. Citado 3 vezes nas páginas 66, 67 e 100.

LATHI, B. *Sinais e Sistemas Lineares - 2.ed*. Bookman, 2007. ISBN 9788560031139. Disponível em: <https://books.google.com.br/books?id=ySxoo2TVeeYC>. Citado na página 29.

LECUN, Y. et al. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1989. Citado na página 39.

LI, S.; LI, S.; DENG, W. Blended emotion in-the-wild: Multi-label facial expression recognition using crowdsourced annotations and deep locality feature learning. *International Journal of Computer Vision*, 2019. Citado na página 33.

LOFQVIST, M.; CANO, J. *Accelerating Deep Learning Applications in Space*. 2020. Citado na página 62.

LUKEZIC, A. et al. Discriminative correlation filter with channel and spatial reliability. In: . [S.l.: s.n.], 2017. Citado 4 vezes nas páginas 55, 58, 59 e 60.

MA, X. et al. *Non-Structured DNN Weight Pruning – Is It Beneficial in Any Platform?* 2020. Citado 2 vezes nas páginas 63 e 64.

MATHEW, N. C. et al. Comparison of yolo versions for object detection from aerial images. *International journal of engineering technology and management sciences*, 2022. Citado na página 50.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, p. 115–133, 1943. Citado na página 27.

MENEZES, P. *Linguagens Formais e Autômatos: Volume 3 da Série Livros Didáticos Informática UFRGS*. Bookman Editora, 2009. ISBN 9788577807994. Disponível em: <https://books.google.com.br/books?id=OEBJ5ga7nvUC>. Citado na página 24.

MILLER, G. A.; NICELY, P. E. An analysis of perceptual confusions among some english consonants. *Journal of the Acoustical Society of America*, 1955. Citado na página 33.

MOLTZAU, A. *What is Kaggle?* 2019. Acessado em 13 de Abril de 2023. Disponível em: <https://medium.com/dataseries/what-is-kaggle-4751e384e916>. Citado na página 110.

ORACLE. *O que é IA? Saiba mais sobre inteligência artificial*. 2023. Acessado em 26 de Março de 2023. Disponível em: <https://www.oracle.com/br/artificial-intelligence/what-is-ai/>. Citado na página 23.

PADILLA, R. et al. A survey on performance metrics for object-detection algorithms. *International Conference on Systems, Signals, and Image Processing*, 2020. Citado 4 vezes nas páginas 32, 33, 34 e 36.

PARMAR, R. *Detection and Segmentation through ConvNets*. 2018. Acessado em 03 de Abril de 2023. Disponível em: <https://towardsdatascience.com/detection-and-segmentation-through-convnets-47aa42de27ea>. Citado na página 43.

PETTIGREW, C. *O Melhor Processador de AI: NVIDIA Jetson Nano Ganha Prêmio Vision Product of the Year de 2020*. 2020. Acessado em 11 de Novembro de 2022. Disponível em: <https://blog.nvidia.com.br/2020/07/24/jetson-nano-vision-product-award/>. Citado na página 113.

PIZA, E. L. et al. Cctv surveillance for crime prevention. *Criminology & Public Policy*, v. 18, n. 1, p. 135–159, 2019. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1111/1745-9133.12419>. Citado na página 19.

PLATFORM, F. *HTTP vs MQTT performance tests*. 2018. Acessado em 10 de Maio de 2023. Disponível em: <https://medium.com/@flespi/http-vs-mqtt-performance-tests-f9adde693b5f>. Citado na página 68.

PRATOMO, A. B.; PERDANA, R. S. Arduviz, a visual programming ide for arduino. In: *2017 International Conference on Data and Software Engineering (ICoDSE)*. [S.l.: s.n.], 2017. p. 1–6. Citado na página 111.

PáDUA, M. *Machine Learning -Métricas de avaliação: Acurácia, Precisão e Recall, F1-score*. 2020. Acessado em 30 de Março de 2023. Disponível em: <https://medium.com/@mateuspdua/machine-learning-m%C3%A9tricas-de-avalia%C3%A7%C3%A3o-acur%C3%A1cia-precis%C3%A3o-e-recall-d44c72307959>. Citado na página 36.

QIN, L. *Online machine learning methods for visual tracking*. Tese (Theses) — Université de Technologie de Troyes, maio 2014. Disponível em: <https://theses.hal.science/tel-03357061>. Citado 3 vezes nas páginas 52, 53 e 54.

REDAçãO, A. E. V. *Saiba qual foi a primeira plataforma de streaming e como ela surgiu*. 2021. Acessado em 24 de Março de 2023. Disponível em: <https://atitudeevisao.com.br/saiba-qual-foi-a-primeira-plataforma-de-streaming-e-como-ela-surgiu/>. Citado na página 76.

REDMON, J. et al. You only look once: Unified, real-time object detection. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2016. p. 779–788. Citado 3 vezes nas páginas 47, 48 e 49.

REDMON, J.; FARHADI, A. Yolo9000: Better, faster, stronger. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2017. p. 6517–6525. Citado 3 vezes nas páginas 44, 49 e 50.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *arXiv: Computer Vision and Pattern Recognition*, 2018. Citado na página 50.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958. Citado na página 27.

RUSSAKOVSKY, O. et al. Detecting avocados to zucchinis: What have we done, and where are we going? *IEEE International Conference on Computer Vision*, 2013. Citado na página 39.

RUSSAKOVSKY, O. et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015. Citado na página 32.

RUSSELL, S. et al. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010. (Prentice Hall series in artificial intelligence). ISBN 9780136042594. Disponível em: <https://books.google.com.br/books?id=8jZBksh-bUMC>. Citado 2 vezes nas páginas 25 e 26.

SAIGG, L. *Veja como ficaram as novas placas do Mercosul*. 2020. Acessado em 13 de Março de 2023. Disponível em: <https://www.brasal.com.br/veiculos/blog/2020/02/14/como-ficaram-as-novas-placas-mercosul/>. Citado na página 66.

SANTOS, R. *Introducing the ESP32-CAM*. 2019. Acessado em 10 de Novembro de 2022. Disponível em: <https://randomnerdtutorials.com/esp32-cam-video-streaming-face-recognition-arduino-ide/>. Citado na página 112.

SEROZHENKO, M. *MQTT vs. HTTP: which one is the best for IoT?* 2017. Acessado em 2 de Setembro de 2022. Disponível em: <https://medium.com/mqtt-buddy/mqtt-vs-http-which-one-is-the-best-for-iot-c868169b3105>. Citado na página 68.

SHANMUGAMANI, R.; MOORE, S. *Deep Learning for Computer Vision: Expert Techniques to Train Advanced Neural Networks Using TensorFlow and Keras*. Packt Publishing, 2018. ISBN 9781788295628. Disponível em: <https://books.google.com.br/books?id=dgdOswEACAAJ>. Citado 3 vezes nas páginas 42, 43 e 44.

SHELHAMER, E.; LONG, J.; DARRELL, T. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. Citado na página 42.

SILVA, I. N. da et al. Artificial neural network architectures and training processes. In: ____. *Artificial Neural Networks : A Practical Course*. Cham: Springer International Publishing, 2017. p. 21–28. ISBN 978-3-319-43162-8. Disponível em: <https://doi.org/10.1007/978-3-319-43162-8_2>. Citado na página 29.

SILVA, J. V. S. *AtalaIA: Uso do Deep Learning para reconhecimento automático de placas de licença automotiva em dispositivos de processamento limitado*. Monografia (Trabalho de Conclusão de Curso) — Universidade Federal de Sergipe, São Cristovão, 2022. Citado 5 vezes nas páginas 20, 66, 90, 107 e 121.

SINGH, R. *Internet of Things: Battle of The Protocols (HTTP vs. Websockets vs. MQTT)*. 2017. Acessado em 2 de Setembro de 2022. Disponível em: <https://www.linkedin.com/pulse/internet-things-http-vs-websockets-mqtt-ronak-singh-cspo/>. Citado na página 68.

SOUZA, F. *MQTT-SN: MQTT para rede de sensores*. 2018. Acessado em 13 de Março de 2023. Disponível em: <https://embarcados.com.br/mqtt-sn-mqtt-para-rede-de-sensores/>. Citado 2 vezes nas páginas 71 e 72.

STANFORD-CLARK, A.; TRUONG, H. L. *MQTT For Sensor Networks (MQTT-SN)*. [S.l.], 2013. Disponível em: <https://www.oasis-open.org/committees/download.php/66091/MQTT-SN_spec_v1.2.pdf>. Acesso em: 2 set. 2022. Citado na página 71.

SUNG, J.-Y.; YU, S.-B.; KOREA, S.-h. P. Real-time automatic license plate recognition system using yolov4. In: *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*. [S.l.: s.n.], 2020. p. 1–3. Citado na página 66.

TAMULY, S.; JYOTSNA, C.; AMUDHA, J. Deep learning model for image classification. *null*, 2019. Citado 2 vezes nas páginas 37 e 38.

TORRES, A.; ROCHA, A.; SOUZA, J. Análise de desempenho de brokers mqtt em sistema de baixo custo. In: . [S.l.: s.n.], 2016. Citado na página 111.

WORLD, R. W. *MQTT vs DDS in IoT | Difference between MQTT and DDS*. 2023. Acessado em 10 de Maio de 2023. Disponível em: <https://www.rfwireless-world.com/Terminology/MQTT-vs-DDS.html>. Citado na página 68.

YADAV, S.; PAYANDEH, S. Critical overview of visual tracking with kernel correlation filter. *Technologies*, v. 9, n. 4, 2021. ISSN 2227-7080. Disponível em: <https://www.mdpi.com/2227-7080/9/4/93>. Citado 2 vezes nas páginas 56 e 58.

YAN, W.; LIU, T.; FU, Y. Yolo-tight: An efficient dynamic compression method for yolo object detection networks. In: *2021 13th International Conference on Machine Learning and Computing*. New York, NY, USA: Association for Computing Machinery, 2021. (ICMLC 2021), p. 378–384. ISBN 9781450389310. Disponível em: <https://doi.org/10.1145/3457682.3457740>. Citado na página 64.

YOSINSKI, J. et al. *How transferable are features in deep neural networks?* 2014. Citado na página 61.

ZHANG, A. et al. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021. Citado 11 vezes nas páginas 38, 40, 41, 42, 44, 45, 46, 47, 50, 61 e 62.

ZHANG, G. *What is the kernel trick? Why is it important?* 2018. Acessado em 16 de Abril de 2023. Disponível em: <https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d>. Citado na página 56.

ZHANG, K.; ZHANG, L.; YANG, M.-H. Real-time compressive tracking. *European Conference on Computer Vision*, 2012. Citado na página 56.

# Apêndices

# APÊNDICE A – Desenvolvimento da Câmera IP com o NodeMCU Esp32Cam

Para desenvolver a câmera IP de baixo custo foi escolhido o Esp32Cam por este já conter recursos como *bluetooth* e *WiFi* nativamente, sem a necessidade de custos extras com módulos como acontece quando o projeto utiliza um Arduino. Além disso, o Esp32Cam dispõe de diversas bibliotecas para aproveitar suas funcionalidades de maneira prática e rápida quando usa o ambiente de desenvolvimento do Arduino IDE para a construção do *firmware*.

O principal recurso considerado para a escolha dessa plataforma é a possibilidade de conexão *wireless* com a internet, possibilitando uma maior flexibilidade do posicionamento da câmera para extrair da melhor forma uma boa imagem para o processamento ALPR.

Este apêndice visa a apresentação da construção da câmera IP, indicando tanto a construção física A.1 como a construção do *firmware* A.2 apropriado que possibilitasse a realização de diversas configurações do Esp32Cam sem a necessidade de alteração do código-fonte, que é importante tendo em vista que para carregar um novo *firmware* para a plataforma é necessário desmontar a câmera.

## A.1 Construção Física da Câmera

Para a construção da câmera foi utilizada uma carcaça simples, geralmente utilizada como câmera falsa em estabelecimentos 62. Essa carcaça foi utilizada para encapsular o Esp32Cam e protegê-lo de poeira e respingos de chuva. Além disso, foi utilizado um tripé adaptado com um pedaço de madeira compensada para instalar a carcaça da câmera, mantendo-a estável 63.

Quanto a alimentação, o Esp32Cam demanda uma fonte que suporte pelo menos uma corrente de 1A e 5V de tensão, ou pelo menos 1A e 3,3V de tensão. Vale ressaltar a importância de associar essa fonte a fios que suportem essa corrente, evitando limitação de desempenho por energia insuficiente, que pode gerar problemas como de reinicialização inesperada, mau funcionamento da captura do sinal *WiFi* pela antena e imagens corrompidas.

Figura 62 – Carcaça de câmera falsa utilizada na montagem do dispositivo de captura das imagens

Fonte: Autor.

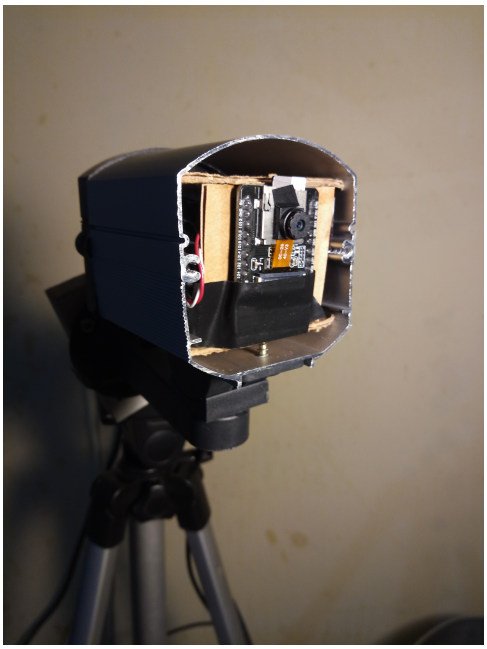
Figura 63 – Tripé com carcaça de câmera parafusada

Fonte: Autor.

Quanto à fonte de alimentação para energizar o NodeMCU foi utilizado um carregador portátil Xiaomi 2C de 20000mAh em vez de uma fonte reguladora de tomada. Esta escolha parte da ideia de possibilitar uma maior flexibilidade na utilização da câmera, possibilitando posicionar esta em lugares estratégicos para melhor enquadramento dos veículos, que a depender da situação não é possível na utilização de tomadas até mesmo com o uso de extensões.

Já quanto ao posicionamento do Esp32Cam dentro da carcaça da câmera foi necessário adicionar um suporte para deixá-lo alinhado na vertical e estável 64. Para esse suporte foram realizados recortes de papelão considerando as medidas da carcaça, além de algumas perfurações para encaixar a pinagem do Esp32Cam. Essas perfurações também possibilitaram a ligação dos fios de alimentação e terra por dentro da carcaça da câmera 65.

Figura 64 – Imagem frontal do suporte com o Esp32Cam na carcaça



Fonte: Autor.

Figura 65 – Imagem traseira do suporte com o Esp32Cam

Fonte: Autor.

Vale ressaltar que para manter os fios presos na pinagem foi realizado um enrolamento da parte desencapada do fio na pinagem da placa, dificultando possível perda total ou parcial de contato entre o fio e o pino.

A escolha por não realizar uma soldagem foi devido à dificuldade que surgiria quando fosse necessário carregar alguma nova versão do *firmware* durante os testes.

## A.2 Desenvolvimento do *Firmware*

Para o *firmware* teve como objetivo a possibilidade de realizar configurações do Esp32Cam dinamicamente, para que assim pudessem ser realizadas diferentes conexões ou configurações da câmera sem a necessidade de carregar para a placa uma nova versão de *firmware*.

Sendo assim, foram escolhidas algumas bibliotecas para tornar o desenvolvimento mais prático e rápido, além de tornar mais simples o processo de correção de falhas por lidar com uma codificação mais enxuta do *firmware*.

Durante esta Seção serão apresentadas as bibliotecas principais utilizadas no desenvolvimento e o papel desempenhado por cada uma para conseguir o resultado esperado do Esp32Cam atuando como câmera IP.

### A.2.1 *WiFiManager.h*

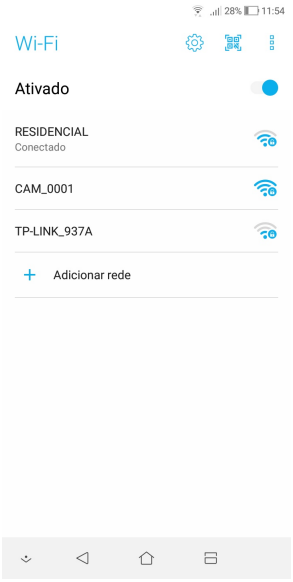
A biblioteca *WiFiManager.h*[1] possibilita usufruir de uma interface de configuração para estabelecer uma conexão *WiFi* no Esp32Cam através de um ponto de acesso que pode ser visto por vários tipos de dispositivos sem fio, dispensando configurar as credenciais de uma rede sem fio diretamente no *firmware*.

Quanto aos dispositivos que podem ser usados para estabelecer conexão com esse ponto de acesso gerado pela biblioteca estão celulares, notebooks, ou qualquer dispositivo que possibilite realizar uma conexão sem fio por meio de uma interface de rede e disponha de algum navegador para acessar a página web através do endereço padrão 192.168.0.1. Então, a interface web acessada através desse IP permite realizar a conexão do Esp32Cam a partir do SSID e senha de alguma rede sem fio disponível no ambiente.

Com as funcionalidades desta biblioteca foi associada uma contagem regressiva, cujo objetivo é habilitar novamente o ponto de acesso e disponibilizar por requisição o acesso à página web de configuração a partir do endereço 192.168.0.1 quando é identificado não haver conexão do Esp32Cam com a internet por pelo menos 5 minutos.

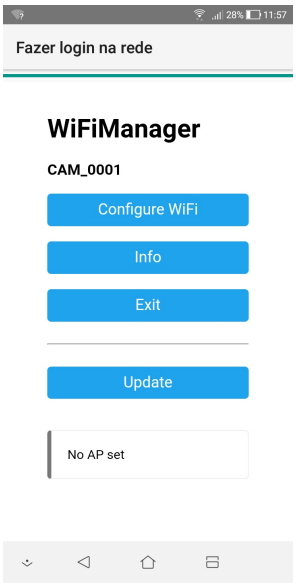
[1] Disponível em: <https://github.com/tzapu/WiFiManager>. Acesso em agosto de 2022

Figura 66 – Ponto de acesso disponível para conexão implementado por meio da biblioteca *WiFiManager.h*



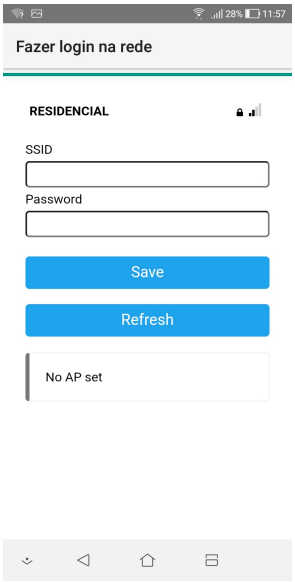

Fonte: Autor.

Figura 67 – Tela inicial da página web para configurar uma nova rede sem fio no Esp32Cam através do *WiFiManager.h*




Fonte: Autor.

Figura 68 – Tela de configuração de uma nova rede sem fio no Esp32Cam através do *WiFiManager.h*



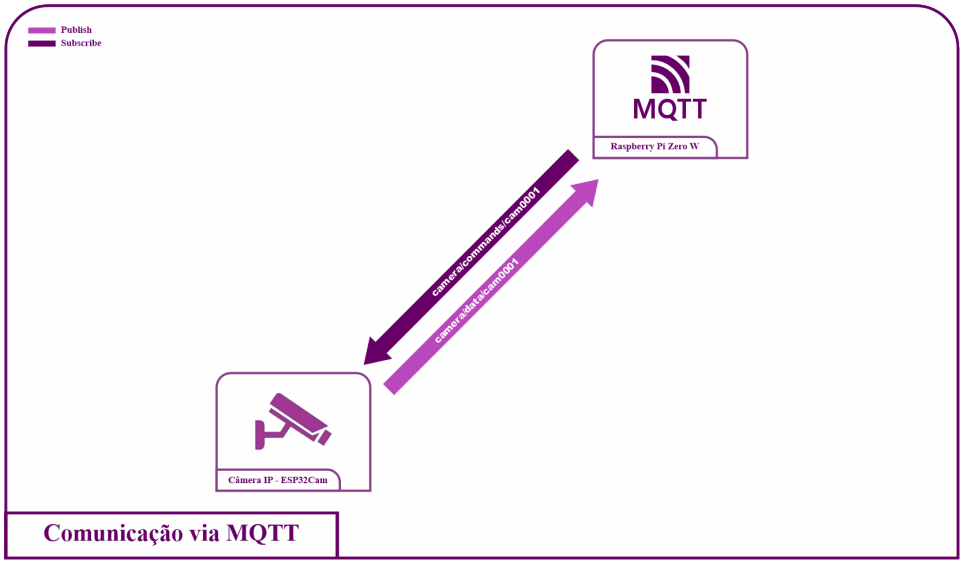

Fonte: Autor.

### A.2.2 *PubSubClient.h*

A biblioteca *PubSubClient.h*[2] está vinculada ao uso do protocolo de comunicação MQTT no Esp32Cam. De maneira geral esta fornece recursos para ser realizada a conexão da placa com algum servidor MQTT, permitindo a assinatura ou publicação em tópicos na forma de fluxo de vetor de bytes enviados por lotes, como também vetor de bytes enviados completamente no caso de dados simples como uma mensagem "olá, mundo!".

No contexto do trabalho desenvolvido a publicação como fluxo de vetor de bytes por lotes é a utilizada por ser trabalhado grandes volumes de dados, evitando que as informações sejam corrompidas.

Figura 69 – Comunicação entre o Esp32Cam e o servidor MQTT




Fonte: Autor.

Para a publicação dos vetores de bytes que compõem as imagens foi utilizado o tópico *camera/data/cam0001*. Já o tópico *camera/commands/cam0001* está vinculado a comandos para controle do Esp32Cam, sendo possível com o envio de uma cadeia de caracteres através desse tópico serem definidas configurações como:

a) resolução das imagens, determinada por valores numéricos conforme a biblioteca *camera.h* associada ao NodeMCU Esp32Cam funciona, sendo então:
   - a resolução de 1600 x 1200 px associada ao valor 13;
   - a resolução de 1280 x 1024 px associada ao valor 12;
   - a resolução de 1024 x 768 px associada ao valor 10;
   - a resolução de 800 x 600 px associada ao valor 9;

[2] Disponível em: <https://github.com/knolleary/pubsubclient>. Acesso em agosto de 2022

- a resolução de 640 x 480 px associada ao valor 8;
- a resolução de 352 x 288 px associada ao valor 6;
- a resolução de 320 x 240 px associada ao valor 5;
- a resolução de 240 x 160 px associada ao valor 3;
- a resolução de 160x120 associada ao valor 1.

b) qualidade da imagem, determinar por valores inteiros entre 0 e 63, que se trata do quão comprimida esta imagem será para reduzir o volume de dados, melhorando assim a taxa de quadros de acordo o valor atribuído a este parâmetro for maior. Porém, o mais habitual é que os valores estejam entre 10 e 63;

c) brilho das imagens, com a intensidade determinada por valores numéricos inteiros entre -2 e 2;

d) contraste, determinado por valores inteiros entre -2 e 2;

e) saturação, determinado por valores inteiros também entre -2 e 2.

### A.2.3 *ESPAsyncWebServer.h*

A biblioteca *ESPAsyncWebServer.h*[3] permite fazer do Esp32Cam um servidor web assíncrono. Ou seja, a plataforma consegue por meio de requisições HTTP disponibilizar páginas web construídas em HTML, CSS e Javascript para lidar com dados mediante uma interface amigável.

Esta biblioteca permitiu que fosse desenvolvida uma interface web 70 para realização de configurações no Esp32Cam de forma dinâmica. Pois, uma vez tendo acesso a essa interface por meio de uma requisição, o usuário pode realizar configurações como:

a) nova conexão com rede sem fio mesmo com o Esp32Cam já conectado a uma previamente configurada;

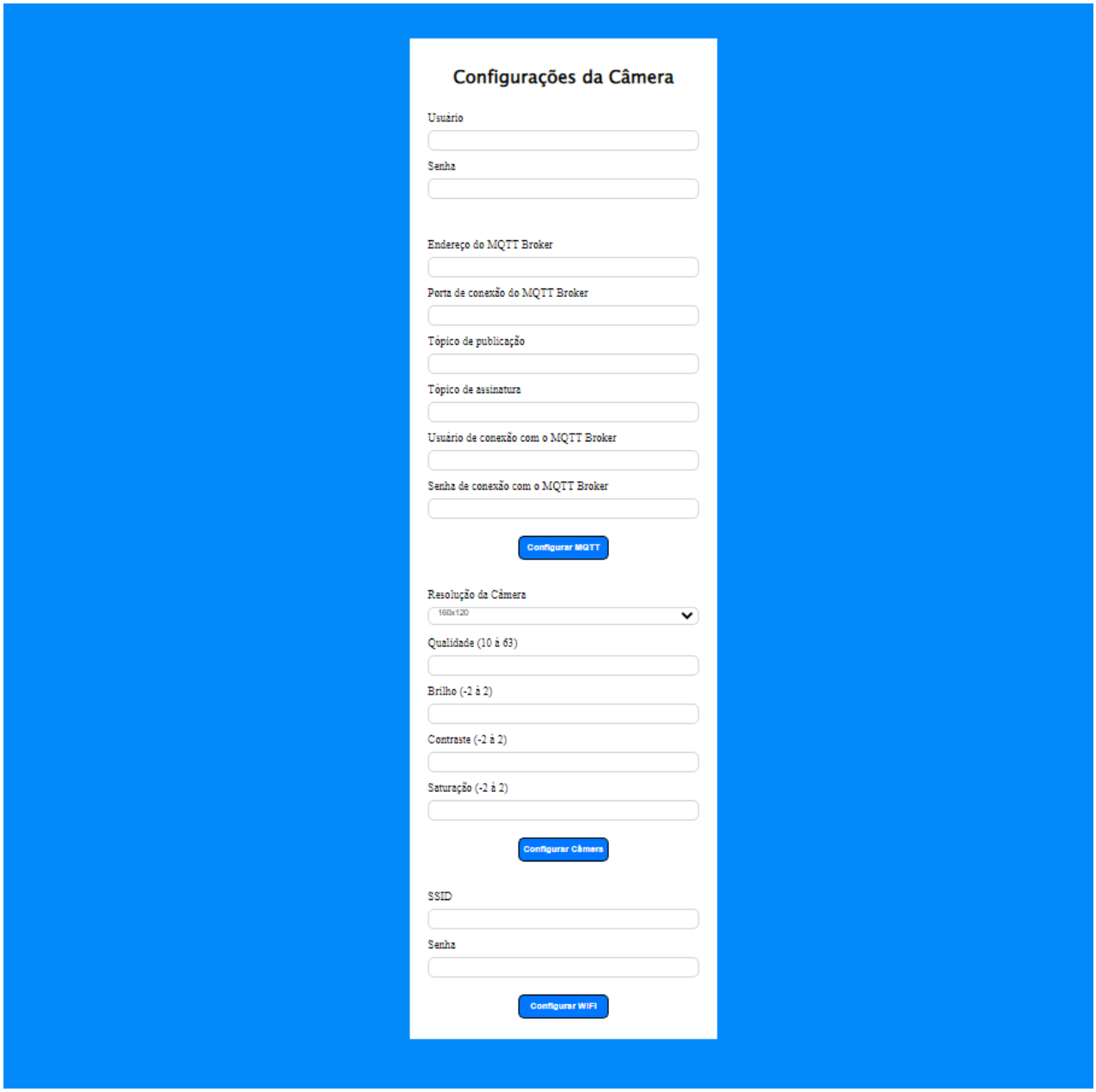
b) nova conexão com algum servidor MQTT, também podendo ser realizada se o Esp32Cam já estiver trocando informações com outro servidor previamente configurado, substituindo-o;

c) configurar a câmera de forma similar a como ocorre quando esta configuração é feita por meio de uma mensagem via protocolo MQTT pelo tópico *camera/commands/-cam0001* A.2.2.

Vale ressaltar que se não for determinado um IP fixo para o Esp32Cam este fica dependente do endereço atribuído via DNS (*Domain Name System*) pelo roteador da rede utilizada, sendo necessário visualizar entre a listas de usuários da rede qual foi o IP destinado à placa.

[3] Disponível em: <https://github.com/me-no-dev/ESPAsyncWebServer>. Acesso em agosto de 2022

Figura 70 – Interface web para configurações do Esp32Cam desenvolvida com a biblioteca *ESPAsyncWebServer.h*




Fonte: Autor.

Uma vez sabido este endereço IP basta digitá-lo no navegador web e inserir a porta 8080, a qual foi determinada no *firmware* como a porta associada às requisições do servidor assíncrono. A partir disso são realizadas requisições para envio de informações digitadas na interface para definir as configurações do Esp32Cam através de rotas configuradas no próprio *firmware* usando recursos da biblioteca, as quais são apresentadas a seguir:

a) / trata-se da rota raíz, a qual realiza uma solicitação da página web de configuração

ao Esp32Cam;

b) **/camconfig** está voltada para o envio das configurações determinadas por meio da interface com o destino sendo o Esp32Cam;

c) **/mqttconfig** realiza a requisição para o Esp32Cam configurar uma nova conexão com algum servidor MQTT, sendo indicado os tópicos de assinaturas e publicação;

d) **/wificonfig** permite fazer uma nova conexão com uma rede *wireless* ao enviar o SSID e a senha desta.

### A.2.4 *LITTLEFS.h*

O LITTLEFS.h[4] trata-se de uma biblioteca que atribui um sistema de arquivos ao Esp32Cam, permitindo a criação de pastas e arquivos para o armazenamento permanente de informações na memória *flash* do Esp32Cam. Isso é importante, pois sem a utilização deste recurso seria necessário a cada nova inicialização do Esp32Cam configurar manualmente uma conexão *WiFi*, uma conexão com algum servidor MQTT e configurar novamente a câmera com a resolução e atributos de imagens desejados.

Então no contexto de uso do Esp32Cam como câmera IP, essa biblioteca ajuda no armazenamento das configurações escolhidas de maneira permanente na memória externa SPI NOR Flash de 4MB.

[4] Disponível em: <https://github.com/lorol/LITTLEFS>. Acesso em agosto de 2022