UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

AMANDA SCHMIDT DE LIMA

WAGNER RODRIGUES ULIAN AGOSTINHO

SISTEMA DE PROCESSAMENTO DE BIG DATA APLICADO À MONITORAÇÃO DE AUTOMÓVEIS EM TEMPO REAL

CURITIBA

2022

AMANDA SCHMIDT DE LIMA
WAGNER RODRIGUES ULIAN AGOSTINHO

SISTEMA DE PROCESSAMENTO DE BIG DATA APLICADO À MONITORAÇÃO DE AUTOMÓVEIS EM TEMPO REAL

Big Data Processing System Applied to Real-Time Vehicle Monitoring

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação do Curso de Bacharelado em Engenharia da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Heitor Silvério Lopes

Coorientador: Prof. M.Sc. Clayton Kossoski

CURITIBA

2022

4.0 Internacional

Esta licença permite compartilhamento, remixe, adaptação e criação a partir do trabalho, mesmo para fins comerciais, desde que sejam atribuídos créditos ao(s) autor(es). Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.

AMANDA SCHMIDT DE LIMA
WAGNER RODRIGUES ULIAN AGOSTINHO

SISTEMA DE PROCESSAMENTO DE BIG DATA APLICADO À MONITORAÇÃO DE AUTOMÓVEIS EM TEMPO REAL

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação do Curso de Bacharelado em Engenharia da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 03/Novembro/2022

Heitor Silvério Lopes
Doutorado
Universidade Tecnológica Federal do Paraná

Leyza Elmeri Baldo Dorini
Doutorado
Universidade Tecnológica Federal do Paraná

Ricardo Lüders
Doutorado
Universidade Tecnológica Federal do Paraná

André Eugênio Lazzarett
Doutorado
Universidade Tecnológica Federal do Paraná

CURITIBA

2022

Dedicamos esse trabalho às nossas famílias,
amigos, professores que fizeram parte dessa
jornada e à Deus que tornou tudo isso possível
mesmo em um momento tão complicado.

AGRADECIMENTOS

Agradecemos aos professores orientadores, Prof. Dr. Heitor Silvério Lopes e Clayton Kossoski pelos ensinamentos, orientações, pela compreensão em momentos complicados, por toda a ajuda e por terem aceito e se disponibilizado a orientarem nosso trabalho da melhor forma possível.

Agradecemos também à UTFPR e seus docentes, que nos transmitiram muitos conhecimentos durante a nossa jornada acadêmica, possibilitando que nos tornássemos exímios profissionais.

Agradecemos aos nossos colegas que estiveram presentes nessa jornada e fizeram a graduação ser mais leve e proveitosa.

Por fim, agradecemos principalmente à Deus, por nos guiar e nos iluminar a todo momento e às nossas famílias por todo o suporte prestado nesta fase.

À todos os envolvidos nessa trajetória, nosso muito obrigado.

Algo que você recebe por ter sorte e algo que
você ganha por ser reconhecido são coisas
totalmente diferentes.
(All Might - Boku no Hero Academia)

RESUMO

O alto número de roubos de veículos juntamente com a baixa quantidade de automóveis que contam com alguma solução de rastreamento tornam fundamental a criação de soluções de identificação e reconhecimento de automóveis. Sistemas de visão computacional podem ser aliados no monitoramento de áreas públicas como ruas e rodovias. No entanto, a análise de automóveis é uma tarefa desafiadora devido a muitos fatores, incluindo diversidade de veículos, características de marca e modelo, cor, direção do movimento, posição da imagem e mudanças de iluminação. Além disso, o custo computacional dos sistemas de detecção é alto. Nesse trabalho de conclusão de curso descreve-se a estratégia adotada para a concepção e desenvolvimento de um sistema para identificar automóveis e indexá-los através de placa, marca e cor. Esse sistema será integrado a um sistema de processamento Big Data, permitindo disponibilizar as informações extraídas para consultas baseadas nessas características.

Palavras-chave: processamento digital de imagens; big data; reconhecimento de veículos; deep learning; machine learning.

ABSTRACT

The high number of vehicle thefts together with the low number of cars that have some tracking solution make it essential to create vehicle identification and recognition solutions. Computer vision systems can be allies in monitoring public areas like streets and highways. However, vehicle analytics is a challenging task due to many factors, including diversity of vehicles, brand and model features, color, direction of movement, image position and lighting changes. In addition, the computational cost of detection systems is high. This course conclusion work describes the strategy adopted for the design and development of a system to identify vehicles and index them through license plate, brand and color. This system will be integrated with a Big Data processing system, allowing the information extracted to be made available for queries based on these characteristics.

Keywords: digital image processing; big data; vehicle recognition; deep learning; machine learning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Siglas

ABNT	Associação Brasileira de Normas Técnicas
CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico
EPS	*Encapsulated PostScript*
PDF	Formato de Documento Portátil, do inglês *Portable Document Format*
PS	*PostScript*
UTFPR	Universidade Tecnológica Federal do Paraná
LABIC	Laboratório de Inteligência Computacional
LDAP	do inglês *Lightweight Directory Access Protocol*
CCE	do inglês *Categorical Cross-Entropy*
CNN	Redes neurais convolucionais, do inglês Convolutional Neural Networks
ReLU	do inglês *Rectified Linear Unit*
YOLO	“Você só olha uma vez”, do inglês *You Only Look Once.*

SUMÁRIO

1 INTRODUÇÃO

De acordo com dados publicados pelo SINESP (Sistema Nacional de Informações de Segurança Pública) no ano de 2019, o número de roubos de veículos no Brasil passou a marca de 1 milhão entre 2015 e 2019 (SACHETO, 2019). Em 2021, os dados do SINESP (2021) mostram que esse número chegou a 1.377.327 ocorrências registradas até fevereiro, sendo 7870 no estado do Paraná. Apesar do alto número de incidentes e do tamanho da frota de veículos existentes, apenas 5% da frota conta com alguma solução de rastreamento (LOGWEB, 2019).

Em virtude do aumento constante da circulação de veículos, principalmente em ambientes urbanos, cada vez mais surgem problemas complexos e que carecem de soluções práticas e automatizadas. Existe uma necessidade do setor de engenharia de tráfego e do departamento de trânsito de obter informações precisas e em tempo-real sobre automóveis nas ruas, não somente para controle e monitoramento do tráfego, mas também para segurança e redução de custos (GUINGO; THOME; RODRIGUES, 2002).

Com o avanço da tecnologia, observa-se cada vez mais o uso de computadores como auxiliares do homem na execução de tarefas. Utilizando câmeras de boa qualidade e técnicas de processamento digital de imagens, pode-se extrair informações essenciais de um veículo como modelo, marca, placa e cor e utilizá-las para diversas aplicações como localizar carros roubados, identificar veículos clonados, coletar estatísticas de fluxo de tráfego em determinada região e muitas outras (GUINGO; THOME; RODRIGUES, 2002).

A literatura atual sobre visão computacional para vigilância de tráfego apresenta várias limitações, como modelos de detecção deficientes, conjuntos de dados desbalanceados usados para treinamento e falta de testes em diferentes conjuntos de dados e vídeos. Eles diminuem ou impedem a aplicabilidade no contexto do mundo real.

Redes Neurais Convolucionais (RNC) (ou *Convolutional Neural Network* – CNN) é o estado da arte para tarefas de reconhecimento de imagem e visão computacional por seus resultados surpreendentes apresentados na *Competição ImageNet Large Scale Visual Recognition (ILSVRC)* em 2012 (YAMASHITA *et al.*, 2018). No entanto, CNN é computacionalmente mais cara. Requer uma GPU de alto desempenho para treinar modelos e o desempenho de aprendizado em *deep learning* depende do número de amostras (ou experiências anteriores) utilizadas. Para obter os melhores resultados de inferência, é necessário treinar modelos com milhares (ou centenas de milhares) de imagens usando um dataset balanceado em número de amostras por classe e com relativa boa qualidade delas.

Com base em tudo o que foi exposto, este projeto propõe integrar um sistema de processamento de *Big Data* aplicado à monitoração de automóveis em tempo real utilizando redes neurais convolucionais, como a YOLO (do inglês, You Only Look Once) (REDMON *et al.*, 2015). Além disso, para realizar o processamento distribuído e garantir escalabilidade, foram configuradas as tecnologias Docker, para conteinerização do sistema, e Kubernetes, para gerenciamento

desses contêineres. Para comunicação dos módulos foi utilizado o Apache Kafka e para persistência e consulta dos dados o MongoDB.

1.1 Objetivos

Nesta seção são apresentados os objetivos gerais do projeto.

1.1.1 Objetivo geral

Desenvolver um sistema de processamento de *Big Data* que suporte um sistema de vigilância de automóveis com diversos fluxos de vídeos.

1.1.2 Objetivos específicos

- Planejar e modelar um sistema capaz de processar fluxos de vídeo de maneira escalável;
- Implementar o sistema usando recursos tecnológicos de software distribuído mais recentes e a infraestrutura de hardware do LABIC;
- Identificar e rastrear um ou mais veículos em streams(fluxos) de vídeo, em tempo real;
- Criar um novo conjunto de dados de automóveis circulantes no Brasil que será disponibilizado publicamente para apoiar pesquisadores e profissionais da área;
- Realizar extrações de características de imagens de automóveis como categoria, marca, modelo, cor e placa;
- Armazenar as características dos automóveis, localização, imagens e *timestamps* em um banco de dados, onde poderão ser consultas posteriormente.
- Permitir a consulta e análise dos dados, assim como a criação de alertas através de uma interface amigável (*frontend e backend*).

1.2 Estrutura do trabalho

Este trabalho é constituído por 6 capítulos. No segundo capítulo, são apresentados os principais conceitos teóricos envolvidos nesse trabalho e as ferramentas utilizadas na solução. No terceiro capítulo, por sua vez, são apresentados os trabalhos relacionados e importantes contribuições desse tema. Já o quarto e quinto capítulos apresentam, respectivamente, a meto-

dologia e os resultados encontrados. Por fim, o sexto e último capítulo apresenta as conclusões sobre o trabalho realizado.

2 REVISÃO DE LITERATURA

Neste capítulo são apresentados os principais conceitos teóricos e tecnologias envolvidas nesse trabalho.

2.1 Big Data

De acordo com o Gartner[1], Big Data (ou Grandes Dados) são ativos de informações de alto volume, alta velocidade e/ou alta variedade que exigem formas econômicas e inovadoras de processamento de informações que permitem insights aprimorados, tomada de decisões e automação de processos. Este termo é muito usado na tecnologia da informação para referir a dados que são produzidos em uma grande quantidade e velocidade, originados potencialmente de diversas fontes, e com possíveis graus de incompletude ou redundância. Usualmente, não é possível gravar todos os dados para serem analisados mais tarde porque uma aplicação real de Big Data pode requerer muito espaço, além da capacidade (e velocidade) de qualquer mídia de armazenamento. Estruturas de dados e algoritmos tradicionais de ordenação, por exemplo, são inviáveis com imensas quantidades de dados. A computação monolítica, sem recursos de distribuição do poder computacional, é absolutamente ineficiente nesta área. Além disso, também refere-se a big data o conjunto de abordagens, técnicas, e ferramentas para analisar grandes quantidades de dados que precisam ser processados em um tempo razoável para serem usados, tratados e armazenados (OUSSOUS *et al.*, 2018). Assim, ferramentas tradicionais de análises de dados são ineficientes e até inviáveis para trabalhar com big data.

Neste contexto, muita pesquisa está sendo realizada acerca de big data e diversas arquiteturas, sistemas e *frameworks* foram desenvolvidos. O termo é calcado em 3 principais características conhecidas como 3 V's:

- Volume: Uma grande quantidade de dados é gerada diariamente das mais diversas fontes. O Datasphere publicado em 2021 pela IDC (2021) apontou que em 2020, 64.2ZB de dados foram criados ou replicados e a perspectiva dada por Dave Reinsel, vice presidente senior da IDC's Global DataSphere, é de que quantidade de dados digitais criados nos próximos cinco anos será maior que o dobro da quantidade de dados criados desde o advento do armazenamento digital, sendo que o segmento de dados que mais cresce é proveniente de internet das coisas e mídias sociais.

- Velocidade: A velocidade do processamento desses dados deve acompanhar a velocidade dos negócios, que é imediatista. Por exemplo, Wallmart gera mais de 2,5PB de dados a cada hora das transações de seus clientes (OUSSOUS *et al.*, 2018). Portanto, é preciso extrair as informações em tempo hábil para a tomada de decisão.

[1] https://www.gartner.com/en/information-technology/glossary/big-data

- Variedade: Big Data não tem o objetivo de tratar apenas dados não estruturados, na verdade ele é considerado um *Any Data* e sempre haverá uma necessidade de implementar algum tipo de estrutura para evoluir os metadados e facilitar o acesso e análise aos dados dados estruturados e não estruturados, texto, sensores, áudio, vídeo, arquivos de logs, etc.

E algumas fontes, como Anuradha *et al.* (2015), ainda citam mais dois V's adicionais:

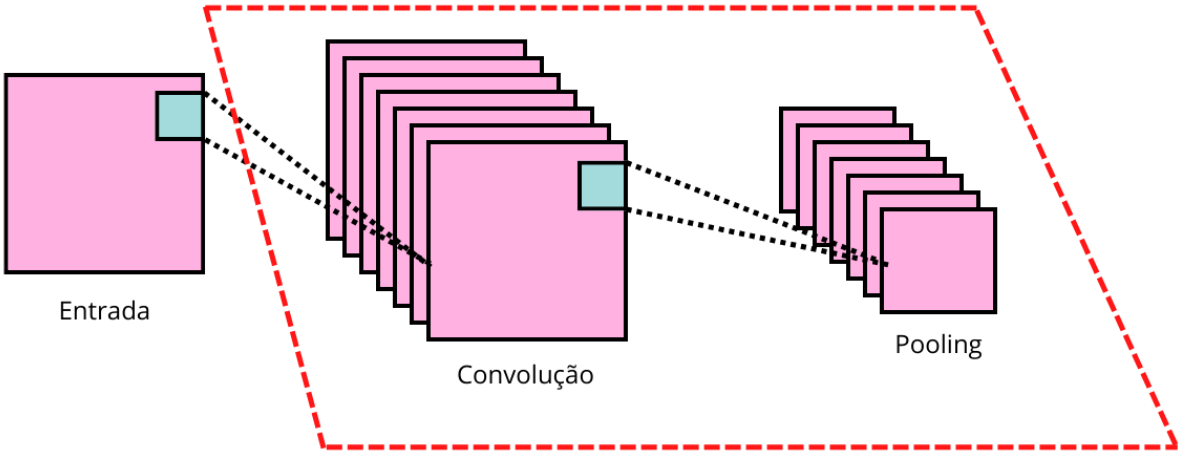
- Veracidade: Com a falta de confiabilidade de algumas fontes de dados é importante garantir que os dados são autênticos e fazem sentido. Por exemplo, dados oriundos de redes sociais, como comentários em publicações, precisam de análise humana para determinar se são verdadeiros ou *fake news*, pois uma mentira reproduzida milhares de vezes pode se tornar "verdade". Para isso, é importante utilizar meios para garantir a qualidade da informação e evitar a tomada de decisões com base em dados incertos e/ou imprecisos.
- Valor: De acordo com a Oracle, empresa que introduziu o termo, Big Data é frequentemente caracterizado por uma "baixa densidade de valor", isso porque pode se armazenar um grande volume de dados sem objetivo nenhum. É importante ter definido o que se é buscado com clareza para aumentar a precisão das análises, reduzir riscos e melhorar a tomada de decisão.

2.2 Redes Neurais Convolucionais

Redes Neurais Convolucionais (CNNs) são consideradas o estado da arte pelo desempenho em aplicações práticas no reconhecimento de dados como imagens e visão computacional. Apesar de ser uma teoria antiga, apresentada pela primeira vez em 1989 por Le Cun et al., ganhou notoriedade em 2012 na competição *ImageNet Large Scale Visual Recognition (ILSVRC)* quando Alex Krizhevsky reduziu erro de 26% para 15% na tarefa de classificação de imagens (YAMASHITA *et al.*, 2018).

Quando comparadas aos processos tradicionais de visão computacional, que utilizavam de métodos manuais para identificar imagens, as CNNs se destacam por ter uma abordagem escalável para as tarefas de reconhecimento de objetos, falas ou imagens e consistem de 3 tipos principais de camadas com funções diferentes.

Figura 1 – Exemplo de funcionamento das redes neurais convolucionais e suas diferentes camadas. Fonte: Adaptada de (PHUNG; RHEE, 2019)



2.2.1 Camada de Convolução

A primeira é a camada de convolução. Nessa camada, é realizada uma operação de convolução entre a entrada e o detector de recursos (também conhecido como *kernel* ou máscara), que é uma matriz que armazena os pesos atribuídos que representam o filtro de uma convolução de imagens no domínio espacial (VARGAS; PAES; VASCONCELOS, 2016).

Porém, é importante ressaltar que nessa operação o filtro não é invertido como acontece em convoluções típicas e, por isso, apesar de ser nomeada como convolução, as CNNs na verdade utilizam Correlação Cruzada (GOODFELLOW; BENGIO; COURVILLE, 2016). Sendo assim, essa operação pode ser representada como:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n K(m,n) I(i+m, j+n), \tag{1}$$

onde **S** é a saída gerada, **I** representa a imagem de entrada, **K** é o *kernel* e **i** e **j** são as coordenadas do ponto da imagem na qual a operação está sendo realizada.

2.2.2 Função de Ativação

É comum que após a convolução seja aplicada uma função de ativação para que seja possível aprender e executar tarefas mais complexas. Essa função é uma transformação não-linear aplicada à saída de cada uma das camadas de rede. Sem ela o sinal de saída se torna uma função linear, o que limitaria a capacidade de tomada de decisão. Existem algumas opções de funções de ativação na literatura, algumas das mais populares serão listadas a seguir:

2.2.2.1 ReLU

Unidade linear retificada (ReLU) é uma das funções de ativação mais utilizadas por ser fácil de otimizar por sua similaridade com unidades lineares. A diferença entre as duas é que a saída da ReLU para valores negativos é 0, que é sua parte retificada.

Esse comportamento não-linear faz com que as derivadas através da unidade linear retificada apresentem um gradiente alto e consistente em todos os lugares em que a função está ativa, ou seja, se a entrada for negativa, a saída será 0 e a função não será ativada. Por isso, a vantagem dessa função é que ela é mais fácil de treinar e é computacionalmente barata. No entanto, o lado negativo dela é que pelo gradiente ser 0 quando a entrada é negativa, o modelo não é capaz de aprender.

A ReLU pode ser interpretada pela função de ativação:

$$g(z) = max\left\{0,z\right\}, \tag{2}$$

E seu comportamento é a curva apresentada na figura 2.

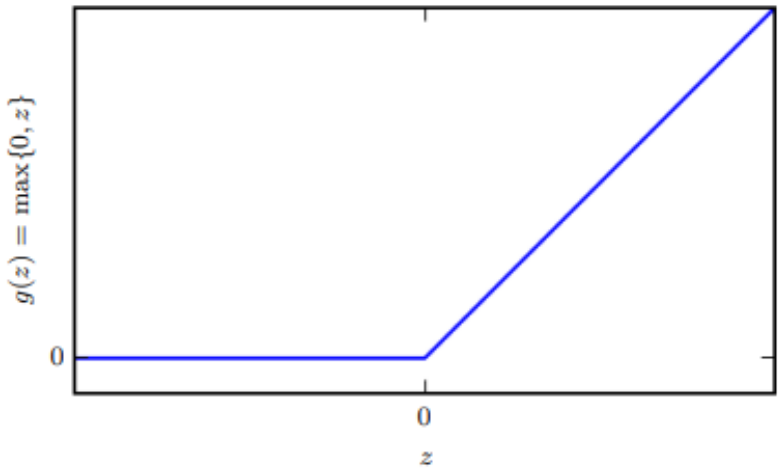


Figura 2 – A função de ativação linear retificada
Fonte: (GOODFELLOW; BENGIO; COURVILLE, 2016)

2.2.2.2 Leaky ReLU

Criada por Maas *et al.* (2013), a Leaky ReLU é uma generalização da ReLU que propõe resolver o problema do gradiente quando a função não está ativa, ou seja, quando $z_i < 0$, em vez de ter uma inclinação plana como a ReLU, ela possui uma pequena inclinação. Portanto, apresenta um gradiente diferente de 0 em todo seu domínio. As três generalizações da ReLU apresentam a seguinte forma:

$$g(z,\alpha)_i = max(0,z_i) + \alpha_i min(0,z_i), \tag{3}$$

onde α_i geralmente assume valores pequenos como 0.01, que apresentou a convergência mais rápida nos experimentos.

2.2.2.3 Sigmoid

As funções Sigmoide ficaram famosas em *deep learning* porque podem ser utilizadas como funções de ativação. Sua curva característica, como podemos ver na figura 3, apresenta forma de S e a principal razão de usar essa função é porque ela existe dentro do intervalo (0,1), o que a torna eficiente para modelos com distribuições de probabilidade, como a distribuição de Bernoulli (GOODFELLOW; BENGIO; COURVILLE, 2016).

Essa função apresenta a seguinte forma:

$$\sigma(z) = \frac{1}{1 + exp(-z)}, \tag{4}$$

Onde σ é a função sigmóide logística.

A função de ativação Sigmóide satura em 0 para valores baixos de entrada z e satura em 1 para valores altos de entrada z. No entanto, essa característica de saturação traz o problema de dissipação do gradiente. À medida que o método de retropropagação progride da saída para a entrada, o gradiente se torna cada vez menor e perto de zero e, como resultado, o gradiente descendente nunca converge para a melhor solução, o que pode prejudicar o treinamento.

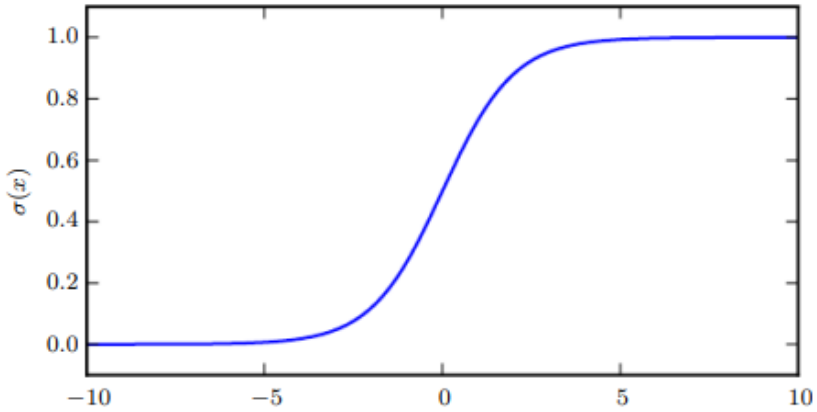


Figura 3 – Função Sigmoide
Fonte: (GOODFELLOW; BENGIO; COURVILLE, 2016)

2.2.2.4 Softmax

Já a Softmax, pode ser considerada uma generalização da função sigmoide. Porém, diferentemente ,essa função deve ser utilizada sempre que quisermos representar uma distribuição de probabilidade com n valores possíveis e é frequentemente utilizada na saída do classificador (GOODFELLOW; BENGIO; COURVILLE, 2016).

Essa função é definida como:

$$softmax(z)_i = \frac{exp(z_i)}{\sum_j exp(z_j)}, \tag{5}$$

Onde o resultado da exponenciação é muito perto de 0 (nunca 0) para $y_i < 0$ e um valor alto para y_i alto.

A Softmax não é afetada por valores negativos, visto que a exponenciação mantém os valores positivos. Já o termo de normalização $(\sum_j exp(z_j))$ garante que os valores da saída $(softmax(z)_i)$ somem 1, sendo que a saída de cada classe está no intervalo 0 e 1, o que compõe uma distribuição de probabilidade entre as n classes.

2.2.3 Camada de Pooling

Outra camada muito importante é a camada de *pooling* que atua reduzindo a taxa de amostragem da saída da camada de convolução, é preciso ressaltar que ela reduz a altura e a largura de um mapa de recursos, mas não sua profundidade. Isso permite diminuir o número de parâmetros necessários para aprender e o custo computacional na rede, além de tornar o modelo mais robusto à variações espaciais (VARGAS; PAES; VASCONCELOS, 2016).

2.2.3.1 Max Pooling

É a forma mais comum de realizar a operação de *pooling*. Essa estratégia escolhe o valor de máximo de uma região, não importando a localização exata do recurso, mas sim se ele pode ser encontrado em uma região da imagem.

Um exemplo do funcionamento da função do Max Pooling utilizando uma janela 2×2 pode ser visto na figura 4.

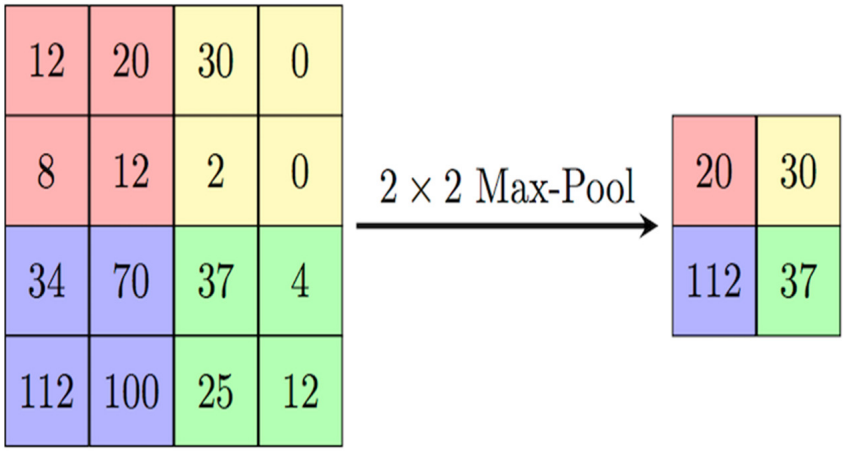


Figura 4 – Exemplo de funcionamento do Max Pooling
Fonte: (S; A; MURUGAN, 2022)

2.2.3.2 Global Average Pooling

Essa estratégia calcula a saída média de cada mapa de recursos da camada anterior e o vetor resultante é alimentado diretamente na camada Softmax. Portanto, utilizando essa operação teremos um mapa de recursos final com N canais e a média dos valores em cada um deles, o que reduz significativamente os dados e prepara o modelo para a camada de classificação final.

Uma vantagem do *Global Average Pooling* é que por não ter parâmetros para otimizar, assim como o *Max Pooling*, o overfitting nessa camada é evitado. Outra vantagem é que ele é mais nativo à estrutura de convolução do que as camadas totalmente conectadas e, por isso, impõe correspondências entre mapa de recursos e categorias, levando os mapas a serem interpretados como mapas de confiança de categorias (ZHANG *et al.*, 2018).

Um exemplo do funcionamento do *Global Average Pooling* pode ser visto na figura 5.

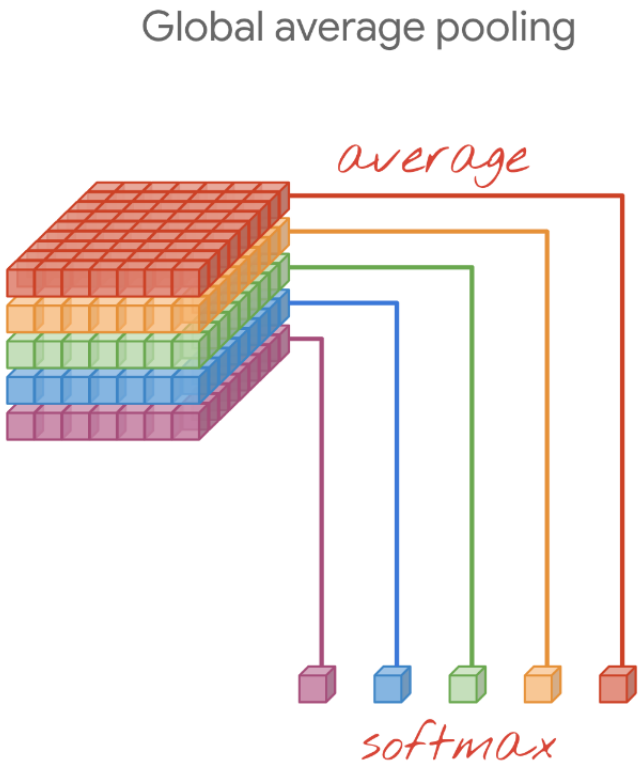


Figura 5 – Exemplo de funcionamento do Global Average Pooling
Fonte: (LAKSHMANAN; GöRNER; GILLARD, 2021)

2.3 Funções de Custo

Funções de custo são necessárias para computar a qualidade da predição realizada antes do treinamento de modelos em sistemas supervisionados. Essa função tem o papel de indicar quão longe se está da predição ideal e a capacidade do modelo de resolver o problema a que ele se propõe.

Isso permite que seja possível quantificar o custo de aceitar as predições geradas pelos parâmetros do modelo. Relacionando isso com a saída esperada, pode-se ter uma noção sobre a melhora, ou não, do sistema que está sendo treinado. O custo total calculado somente sobre a distribuição de treinamento, também conhecido como risco empírico, pode ser definido como (GOODFELLOW; BENGIO; COURVILLE, 2016):

$$J(\Theta) = \frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)};\theta),y^{(i)}), \tag{6}$$

onde m é a quantidade de amostras de treinamento, L é a função que indica o custo para cada exemplo, e $y^{(i)}$ é a saída esperada e $f(x^{(i)};\theta)$ é a predição para a entrada $x^{(i)}$.

Nas subseções que seguem será descrita a função de custo: *Cross-entropy*.

2.3.1 Cross-Entropy

Conhecida como função de custo de entropia cruzada e comumente utilizada em aprendizado de máquina, é uma medida baseada na entropia calculando a diferença entre duas distribuições de probabilidade. É definida na forma:

$$CE = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_{kn} \cdot \log \hat{y}_{kn}, \tag{7}$$

sendo K a quantidade de classes do problema, N a quantidade de exemplos, $\hat{y}_{kn}$ a predição para a classe k do exemplo n e y_{kn} a predição esperada para a classe k do exemplo n, para $y_{kn} \in \{0,1\}$.

Nessa forma, ela também é conhecida como *Categorical Cross-Entropy* e é usada para tarefa de classificação multiclasse. Já a *Binary Cross-Entropy* é utilizada para tarefas de classificação binária, modificação criada para contornar o problema gerado quando a saída esperada em y é 0.

2.4 Otimizadores

Otimizadores são métodos utilizados para alterar os atributos da rede neural, como gradiente e taxa de aprendizado, de forma a reduzir custos. Uma das formas mais famosas é a estratégia de "Descida do Gradiente", um algoritmo de otimização de primeira ordem que depende da derivada de primeira ordem de uma função de custo, com objetivo de descobrir como essa função pode ser utilizada para atingir um mínimo. Surgiram ao longo dos anos, diversos algoritmos ou métodos com modificações para otimizar o desempenho final. Na subseção seguinte será apresentado o Adam, um dos otimizadores mais famosos para implementação dessa estratégia de descida do grandiente.

2.4.1 Adam

Introduzido por Kingma e Ba (2014), o Adam, nome derivado da expressão *"adaptive moment estimation"*, é um método para otimização estocástica eficiente e adota uma estratégia adaptativa, com apenas um gradiente de primeira ordem e pouca necessidade de memória.

O Adam é uma combinação de outros dois algoritmos: o AdaGrad e o RMSProp (KINGMA; BA, 2014) com *momentum*. O primeiro adapta a taxa de aprendizado de todos os parâmentros do modelo e funciona bem com gradientes esparsos. Enquanto o segundo é uma adaptação do AdaGrad, para ter uma melhor perfomance em configurações em linha e não estacionárias e utiliza *momentum* no gradiente reescalonado para gerar atualizações de parâmetros (GOODFELLOW; BENGIO; COURVILLE, 2016).

Momentum é um algoritmo proposto para solucionar um problema de mau condicionamento das funções de otimização, que podem ser mais sensíveis a movimentações em um sentido do que no outro. Essa solução proposta faz a movimentação se basear no último movimento realizado e não apenas no gradiente atual. O Adam utiliza uma estratégia com dois níveis de *momentum* e as atualizações são estimadas diretamente utilizando a média do dois momentos do gradiente (KINGMA; BA, 2014).

Primeiro, é feita uma estimativa do momento de primeira ordem do gradiente (GOODFELLOW; BENGIO; COURVILLE, 2016). O primeiro *momentum* leva em consideração os últimos gradientes calculados e pode ser definido como:

$$\boldsymbol{s} = \rho_1 \cdot \boldsymbol{s} + (1 - \rho_1) \cdot \boldsymbol{g}, \tag{8}$$

sendo $\boldsymbol{s}$ o primeiro momento, ρ_1 a taxa de decaimento para o momento no intervalo $[0,1]$ e $\boldsymbol{g}$ o gradiente da função.

A estimativa do segundo *momentum* pode ser definida como:

$$\mathbf{r} = \rho_2 \cdot \boldsymbol{r} + (1 - \rho_2) \cdot (\mathbf{g} \odot \mathbf{g}), \tag{9}$$

sendo $\mathbf{r}$ o segundo momento, ρ_2 a taxa de decaimento para o momento no intervalo $[0,1]$ e $(\mathbf{g} \odot \mathbf{g})$ quadrado de cada uma das derivadas parciais do gradiente.

Em seguida, são feitas as correções estimadas para o primeiro e segundo momento, sendo elas:

$$\hat{\boldsymbol{s}} = \frac{\boldsymbol{s}}{(1 - \rho_1^t)}, \text{e} \tag{10}$$

$$\hat{\boldsymbol{r}} = \frac{\boldsymbol{s}}{(1 - \rho_2^t)}, \text{respectivamente.} \tag{11}$$

sendo t a iteração.

Por último, é feita uma atualização do cálculo realizado:

$$\boldsymbol{\Delta\theta} = -\epsilon\frac{\hat{s}}{\sqrt{\hat{r}} + \delta}, \tag{12}$$

onde $\Delta\theta$ é a movimentação a ser feita para minimizar a função de custo, ϵ é a taxa de aprendizado global, δ é uma pequena constante de establização numérica (GOODFELLOW; BENGIO; COURVILLE, 2016).

Assim, o Adam é adequado para problemas que são grandes em termos de dados e/ou parâmetros, configurações não estacionárias e problemas com gradiente com muito ruído ou esparso (KINGMA; BA, 2014).

2.5 Regularização

Ao treinar grandes redes neurais, mesmo com o treinamento se baseando na otimização das funções de custo calculadas sobre os dados, existe um desafio para definir quanto treinar o modelo. Com pouco treino o modelo não aprenderá com os conjuntos de testes e com muito treino o modelo vai parar de generalizar e se adaptará, aprendendo o ruído estático do conjunto de treinamento, fenômeno conhecido como *overfitting* (GOODFELLOW; BENGIO; COURVILLE, 2016). Consequentemente, esse fenômeno diminuirá a perfomance no conjunto de testes, aumentando o erro de generalização.

Para reduzir essa adaptação do modelo e evitar o *overfitting*, existem as técnicas de regularização. Regularização é a modificação feita num algoritmo de aprendizado com o objetivo de reduzir o erro de generalização, mesmo que isso implique aumentar o erro no conjunto de treinamento (GOODFELLOW; BENGIO; COURVILLE, 2016). Existem muitas estratégias para isso, algumas colocam restrições no modelo de aprendizagem, outras adicionam termos extras na função ou reduzem os pesos do modelo, entre outras técnicas existentes que, se bem escolhidas, podem melhorar o desempenho do modelo. Na subseção seguinte, será apresentada a técnica *Early-Stopping* que foi a estratégia utilizada nesse trabalho.

2.5.1 Early-Stopping

A estratégia *Early-Stopping* é uma das mais usadas formas de regularização em *deep learning* e sua popularidade vem da sua simplicidade e efetividade. Consiste em, durante o treinamento, monitorar o conjunto de testes para avaliar se o desempenho do modelo começou a degradar, ou seja, verificar se o modelo está se adapdanto excessivamente ao conjunto de treinamento, e então interromper o treinamento antecipadamente, como o nome indica.

Uma abordagem utilizada é treinar o modelo para um grande número de épocas. Isso é feito dividindo o grupo de treinamento em duas partes: treinamento e validação. Durante o treinamento, o modelo será monitorado a cada época no conjuto de validação. Assim, quando

o desempenho do modelo de dados nesse conjunto começar a degradar, o treinamento será interrompido.

Um exemplo do funcionamento do *Early-Stopping* pode ser visto na figura 6, retirada do livro do Goodfellow, Bengio e Courville (2016).

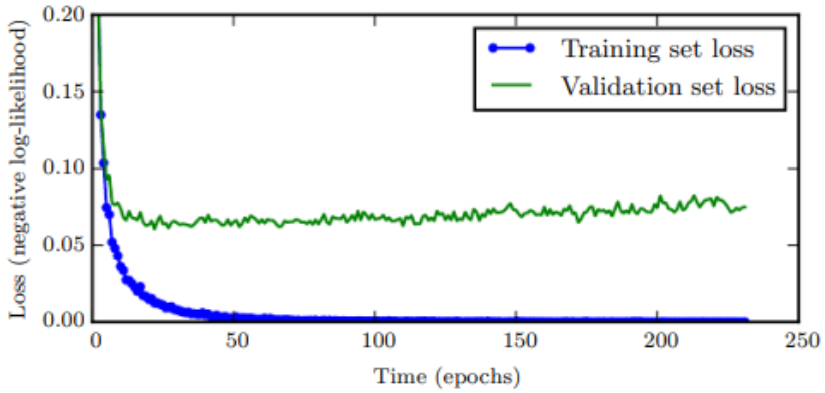


Figura 6 – Exemplo de funcionamento do Early-Stopping
Fonte: (GOODFELLOW; BENGIO; COURVILLE, 2016)

Na imagem, podemos notar que o custo do conjunto de treinamento, em azul, vai diminuindo, enquanto após algumas epócas, o custo do conjunto de validação, em verde, começa a aumentar. Isso indica a diminuição na capacidade de generalização do modelo e, portanto, a queda no desempenho, indicando que o treinamento pode ser interrompido.

2.6 Validação

Como apresentado na subseção 2.5.1, dividir o modelo nos conjuntos de teste e treinamento pode ser problemático durante o treinamento do modelo. Isso porque avaliar o desempenho do modelo no conjunto de testes, apresentaria um problema enviesado e não representaria o cenário real. Se essa avaliação fosse feita no conjunto de treino, poderia causar *overfitting*. Portanto, a solução apresentada seria repetir o treinamento e os testes em uma divisão do conjunto de dados original, que resultaria em um novo conjunto chamado "Validação".

No entanto, essa divisão pode levar a um novo problema. O novo conjunto de dados poderia ser um conjunto pequeno, o que implica incerteza estatística acerca dos dados, pois poderia ter muito ruído. Para isso, são apresentadas as soluções de validação cruzada, procedimento usado para estimar o desempenho do modelo em novos dados, sendo o mais comum o procedimento *K-fold* (GOODFELLOW; BENGIO; COURVILLE, 2016), que será apresentado na próxima subseção.

2.6.1 K-fold

Na estratégia do K-fold, os dados são divididos em K subconjuntos iguais sem sobreposição de dados. Um subconjunto é utilizado para validação e os demais $K - 1$ subconjuntos sendo utilizados para treinamento. Para isso, existem algumas táticas comuns para escolha de valor de K:

- Representativo: o valor de K é escolhido de forma que cada amostra de dados seja grande o suficiente para ser representativo.
- $K = 10$: Valor escolhido através de experimentos, por geralmente apresentar bons resultados.
- $K = n$: sendo n o tamanho do conjunto de dados.

A figura 7 apresenta um exemplo da divisão dos dados para $K = 5$, sendo os blocos destacados em vermelho como utilizados para testes do conjunto de validação e os demais, em verde, para treinamento.

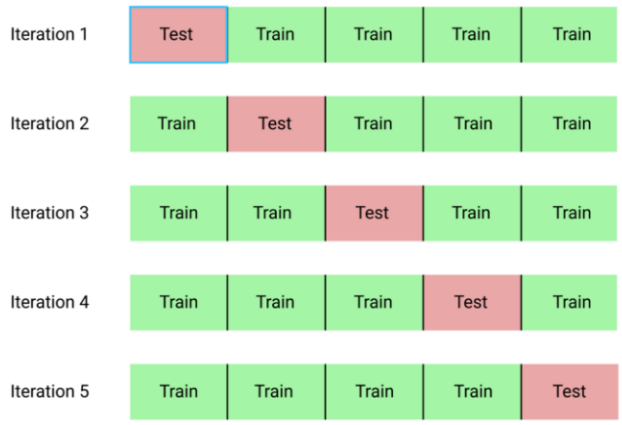


Figura 7 – Exemplo da aplicação do K-Fold para K = 5.
Fonte: Imagem retirada da internet.

2.7 Transfer Learning

Transfer Learning, do português Aprendizado de Transferência, é uma técnica de *machine learning* onde o conhecimento de um modelo obtido no treinamento de uma tarefa é reaproveitado para uma segunda tarefa.

O objetivo dessa técnica é que ao treinar uma segunda tarefa, o desempenho seja melhorado e o tempo de aprendizado reduzido. Para isso, os recursos do primeiro modelo precisam ser gerais e a segunda tarefa precisa ser suficientemente relacionada a tarefa de origem. Assim, o primeiro treinamento gerará uma grande quantidade de informações sobre a solução da

primeira tarefa e esse conhecimento será o ponto de partida para o aprendizado da segunda tarefa, como podemos ver na figura 8. Esse recurso é muito utilizado em problemas de aprendizado *multi task*, como por exemplo uma rede neural convolucional treinada para classificar os datasets de imagens ImageNet ou Open Images.

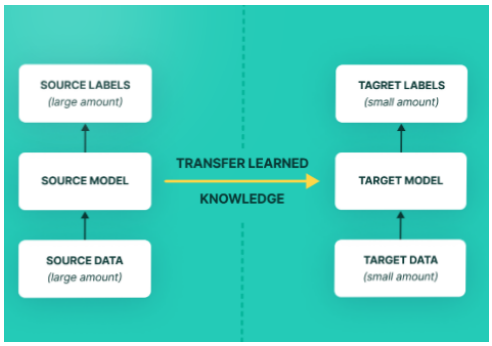


Figura 8 – Diagrama de funcionamento do aprendizado de transferência.
Fonte: Imagem retirada da internet.

2.8 Trabalhos Relacionados

Neste capítulo são apresentados trabalhos com *datasets* publicados na literatura e trabalhos com uso de tecnologias semelhantes.

2.8.1 Datasets

Sobre a visão prática, os *datasets* têm um problema especial: a localidade do veículo. Em geral, montadoras produzem automóveis diferentes para países diferentes. Geralmente, as categorias de veículo são padronizadas, mas os nomes e os modelos dos automóveis podem ser diferentes em dois países por decisão da montadora. Assim, os conjuntos de dados de automóveis existentes são tendenciosos ou focados em mercados específicos dependendo da região.

A tabela 1 apresenta uma comparação de conjuntos de dados com número de imagens, número de classes (modelos), balanceamento de classes, aumento de dados e observações.

Tabela 1 – Comparação de datasets para detecção de carros
Fonte: Autoria Própria

Dataset	# imagens	# classes	Classes Balanceamento	Data augmentation	Observações
Cars dataset (KRAUSE *et al.*, 2013a; KRAUSE *et al.*, 2013b)	16,185	196 classes de carros	Não	Não	Ambos os conjuntos de dados contêm diferentes tamanhos de imagem, características do veículo (cor, forma), planos de fundo complexos e qualidades de imagem.
CompCars (YANG *et al.*, 2015b; YANG *et al.*, 2015a)	164,344	163 marcas de carro com 1,716 modelos	Não	Não	As amostras de automóveis são principalmente automóveis do mercado chinês.
DVM-Cars (HUANG *et al.*, 2021a; HUANG *et al.*, 2021b)	1,451,784	899 Modelos de carros de mercado do Reino Unido	Não	Não	Estrutura desequilibrada (e.g. algumas intraclasses têm poucas amostras).
VMMRdb (TAFAZZOLI; FRIGUI; NISHIYAMA, 2017)	291,752	9,170	Não	Não	Falta de comparações com muitos datasets listados na literatura.
BRCars (KUHN; MOREIRA, 2021)	300,325	427 modelos	Não	Sim	Caracterizado pela falta de padronização quanto às perspectivas dos automóveis, ângulos de imagem e quantidade de amostras por classe.

Krause *et al.* (2013a) desenvolveu o *dataset* Stanford Cars, compostos por dois conjuntos de dados de carros diferentes. O primeiro compreende o BMW-10, conjunto de 10 sedãs BMW de granulação ultrafina (512 imagens). A segunda, contém um conjunto de 197 modelos de automóveis (16.185 imagens). A aquisição dos dados é baseada em um site de carros populares para uma lista de carros produzidos desde 1990, então aplica um procedimento de

desduplicação agressiva, baseado em hash perceptual, a um subconjunto de imagens de exemplo para essas classes. Os dados são armazenados de acordo com a marca, o modelo e o ano. As anotações contém a quantidade de modelos de classe.

O *dataset* Comprehensive Cars (CompCars) de Yang *et al.* (2015b) é dividido em um conjunto de imagens retiradas da web e outro de imagens câmeras de segurança. O conjunto de dados retirado da web contém 161 marcas e 1.687 modelos. O conjuto de dados de segurança contém 50.000 imagens da frente dos carros extraídas de câmeras de CFTV e rotuladas com caixas delimitadoras, modelo e cor do carro. No total, são 136.726 imagens de carros inteiros e 27.618 imagens de partes do carro. O que difere esse *dataset* dos outros é que ele oferece 4 recursos exclusivos de classificação sendo: hierarquia, atributos, ângulos de visão que se têm do veículo *("front (F)", "rear (R)", "side (S)", "front-side (FS)", "rearside (RS)", and "All-View")* e parte do carro. Nesse estudo, para validar a usabilidade da base de dados eles propuseram diferentes aplicações, sendo elas classificação refinada de carros e verificação de modelo carro, baseadas em rede neural convolucional (CNN) que foi escolhida por obter grande sucesso empírico em muitos problemas de visão computacional, especificamente o modelo Overfeat que é pré-treinado no ImageNet e ajustado com imagens de carros para classificar e prever atributos, que seria a outra aplicação, sugerida pelos autores, do dataset.

A base de dados DVM-CAR (HUANG *et al.*, 2021a; HUANG *et al.*, 2021b) contém 1.4 milhões de imagens de 899 modelos de carros, bem como sua especificação de modelo correspondente e informações de vendas ao longo de mais de dez anos no mercado do Reino Unido. Esse conjunto de dados é focado em pesquisas e aplicativos de marketing visual. As imagens foram selecionadas a partir de oito pontos de observação. Para cada imagem, o fundo do carro é removido. Os dados são armazenados seguindo o formato marca-modelo-ano-cor. Existem tabelas com detalhes da montadora, informações da imagem (por exemplo, ID, nome, ponto de vista previsto), preço, dados de vendas e acabamento (por exemplo, emissão de gás, tipo de combustível, tamanho do motor).

Em Tafazzoli, Frigui e Nishiyama (2017) foi produzido um conjunto de dados rotulado com marca, modelo e ano de produção do veículo. Comparado com outros dataset, conjunto de dados VMMR é maior em escala e diversidade, contendo 9.170 classes compostas por 291.752 imagens, abrangendo modelos fabricados entre 1950 e 2016. As imagens variam de diferentes dispositivos e vários ângulos de visão, garantindo uma ampla gama de variações para dar conta de vários cenários que podem ser encontrados durante os testes, em um cenário da vida real. A arquitetura de rede escolhida foi a Resnet, por superar outras arquiteturas e foi utilizado o modelo pré-treinado no ImageNet, ajustado nos conjuntos de dados em experimento com o mesmo tamanho, número de épocas e taxa de aprendizagem. Eles compararam a performance do dataset deles com o apresentado por Yang *et al.* (2015b), foram escolhidas apenas classes que estavam presentes em ambos os conjuntos para que a comparação fizesse sentido. O resultado foi que a performance do modelo treinado em CompCars degrada pouco em relação ao modelo treinado em VMMR, mas o resultado da performance no conjunto de dados mesclado

(CompCars + VMMR), superou os dois casos anteriores, o que prova que empregar dados de treinamento adicionais pode aumentar os resultados da classificação.

O *dataset* de automóveis brasileiros do Kuhn e Moreira (2021) contém 300.325 imagens de modelos em várias perspectivas, incluindo vista interna e painel. Eles criaram o conjunto de dados de um site de publicidade de automóveis e as imagens foram tiradas por usuários com diferentes níveis de habilidades fotográficas.

Zhang *et al.* (2022) utilizaram o YOLOv5 aprimorado como base para detecção de automóveis em tempo real, objetivo era utilizar um método que reduzisse a taxa de falsa detecção desses automóveis causada por uma oclusão. O dataset utilizado para o treinamento foi criado com imagens extraídas de frames de vídeos e imagens selecionadas do BIT Vehicle Dataset (DONG *et al.*, 2015). No total, ele teve 2844 imagens, sendo 2553 utilizadas para validação do treinamento e 291 imagens testadas e avaliadas por algoritmos. O conjunto de imagens para o treinamento possuía imagens de diferentes ângulos, horários do dia e condições diferentes. As imagens foram separados em 8 categorias diferentes, sendo elas Bus, Minibus, Family Sedan (Sedan), Taxi, Heavy Truck (Caminhão pesado), Truck (Caminhão), SUV, e Special Vehicle (Veículo Especial). Eles utilizaram o algoritmo *Flip-Mosaic* para melhorar a percepção da rede, os passos descritos para o aprimoramento foram retirados e traduzidos de Zhang *et al.* (2022) e são os seguintes :

- *Para cada imagem no conjunto de dados, 3 imagens foram selecionadas aleatoriamente novamente no conjunto de dados.*

- *Os pontos de achatamento (x, y) em uma imagem em branco foram selecionados aleatoriamente. As fotos em branco foram divididas em quatro partes. As quatro fotos foram divididas em quatro partes divididas pelo ponto central, e as partes em excesso foram descartadas.*

Figura 9 – Algoritmo de aumento de dados do Mosaico tradicional
Fonte: (ZHANG *et al.*, 2022)

- *As imagens compostas no item anterior, foram processadas por três operações aleatórias para distribuição homogênea. Os dados foram aprimorados virando a imagem para a esquerda, para a direita, para cima e para baixo. Ao mesmo tempo, foi introduzido um ruído aleatório para melhorar a generalização do modelo.*

Figura 10 – Algoritmo de aumento de dados do Flip-Mosaic.
Fonte: (ZHANG *et al.*, 2022)

O resultado obtido pela rede aprimorada apresentou melhor desempenho na identificação de SUVs e Sedans e teve pouca melhora no desempenho geral, de aproximadamente 0.5%.

2.8.2 Framework escalável para processamento de fluxos de vídeo

O trabalho de Kossoski *et al.* (2022) apresenta um pipeline de processamento distribuído de big data que permite processar vários fluxos de vídeos de maneira escalável. Os autores criaram um sistema nomeado de "Scalable Face Mask Detection Pipeline" (SFMDP) proposto para o processamento de fluxos de vídeo em escala que é constituído de duas partes principais:

1. O sistema de ingestão de dados processa streams de vídeo em tempo real ou quase em tempo real, usando Apache Kafka[2], uma plataforma de streaming distribuída de código aberto baseada no paradigma produtor e consumidor. Os fluxos de vídeos são recebidos de produtores que publicam frames em um ou mais tópicos Kafka, que, por sua vez, atende a um grupo de consumidores, que são processos que cooperam para consumir dados de algum tópico e realizar processamentos e armazenamentos diversos.

2. O SFMDP contém três módulos desacoplados que consomem frames de um tópico Kafka e realizam operações para (a) detecção de corpo humano, (b) detecção de cabeça e (c) classificação de máscara facial. Os principais componentes desses blocos são arquiteturas DNN como Yolo-v4, ResNet e um modelo próprio de inferência (FaceMask-v1), baseado na transferência de aprendizado MobileNetv2, treinado no conjunto de dados UTFPR-FMD1 (imagens diversas e classes balanceadas: 50% faces em máscara fácil, 50% faces com máscara facial). O banco de dados MongoDB armazena os frames processados relevantes que podem ser usados para consultas estruturadas.

Embora o SFMDP (Figura 11) seja voltado a detecção de máscaras faciais, no contexto de COVID-19, em ambiente de vigilância de pessoas, os autores destacam que este framework

[2] https://kafka.apache.org/

pode ser facilmente adaptado para quaisquer outras atividades que envolvam processamento de fluxos de vídeos como automóveis, pessoas, e eventos diversos.

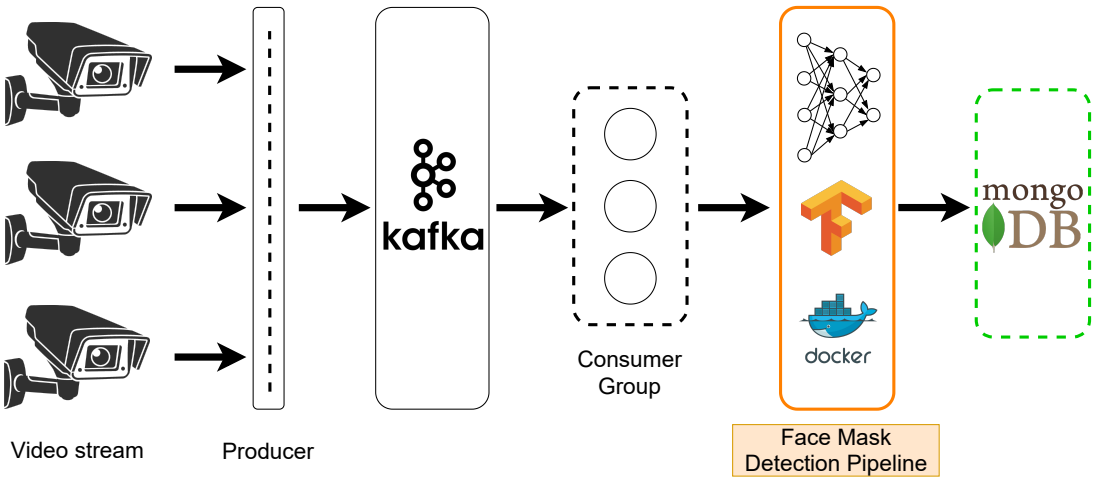


Figura 11 – Visão geral do Scalable Face Mask Detection Pipeline
Fonte: (KOSSOSKI *et al.*, 2022)

3 MATERIAIS E MÉTODOS

Neste capítulo estão detalhados os materiais e métodos empregados para o desenvolvimento deste trabalho. Na primeira seção estão descritos os materiais e ferramentas utilizados para realizar as atividades e na segunda seção os procedimentos, testes e implantação da solução da proposta.

3.1 Materiais

3.1.1 Kubernetes

O processamento em tempo real de grandes quantidades de dados faz com que seja necessário o uso de uma arquitetura capaz de trazer escalabilidade, facilidade de *deploy* de aplicações e facilidade de gerenciamento de todas aplicações. Para atender essas necessidades o uso do Kubernetes, um sistema de código aberto usado para automatizar a implantação, escalonamento e gerenciamento de aplicações em contêineres, tem se mostrado uma ótima opção (MEDEL *et al.*, 2016), visto que tem sido usado em pesquisas bem semelhantes, no sentido de processamento de *Big Data* (HUANG; ZHOU; ZHANG, 2021).

3.1.2 Kafka

Apache Kafka é a arquitetura mais popular utilizada para processar um streaming de dados (HIRAMAN; M.; C., 2018). Kafka é um sistema de mensagens *publish-subscribe*, distribuído e escalável, originalmente desenvolvido pelo Linkedin e posteriormente transformado em *open source*, projetado com as seguintes características:

- Escalabilidade: Permite expandir e contrair o armazenamento e processamento de forma elástica, pensando em clusterização e escalabilidade horizontal.
- Alta taxa de transferência: Projetado para suportar um grande volume de dados, usando um cluster de máquinas com latências tão baixas quanto 2ms.
- Armazenamento Permanente: Armazena *streams* de dados de maneira confiável num cluster particionado, replicado, distribuído e tolerante a falhas.
- Suporte a vários clientes: Suporta fácil integração de clientes de diferentes fontes, como Postgres, JMS, Elasticsearch, AWS S3.
- Bibliotecas: Como Java, .NET, PHP, Ruby, and Python.

3.1.3 MongoDB

O MongoDB é um software de banco de dados, *open source* e orientado a documentos. Escrito na linguagem C++, é caracterizado como um banco de dados NoSQL projetado para trabalhar com grandes volumes de dados distribuídos, permitindo gerenciar informações orientadas a documentos, armazená-las ou recuperá-las.

O MongoDB Atlas permite que você coloque facilmente os bancos de dados Mongo na nuvem usando servidores da Amazon, Google e Microsoft. É uma plataforma amplamente utilizada para novos projetos e com perspectivas de crescimento por proporcionar fácil implementação e configuração em tempo real, recursos de segurança pré-configurados (como autenticação e criptografia) e alta disponibilidade, usando conjuntos replicados e oferecendo opções de backup totalmente gerenciadas, incluindo recuperação incremental de dados. Nesse projeto ele será disponibilizado em uma página web para formar a interface do usuário e tornar as consultas às informações necessárias, através de filtros (como marca, modelo, cor, etc.), mais fáceis e efetivas.

3.1.4 Python

Para implementação e integração desse trabalho, foi utilizada a linguagem Python. Python é uma linguagem de programaçãode alto nível, *open source*, multiparadigma e que ganhou popularidade por ter uma grande quantidade de bibliotecas e frameworks disponíveis gratuitamente. Portanto, possui uma grande comunidade de pessoas desenvolvedoras por ampliar as possibilidades de desenvolvimento de sistemas e ter uma sintaxe mais simples e intuitiva.

3.1.5 TensorFlow

TensorFlow é uma biblioteca de *deep learning* de código aberto criada para definir, treinar e implementar modelos de aprendizado de máquina e aplicada para diversas tarefas, como computação numérica, entre outras (GOLDSBOROUGH, 2016).

A linguagem dele é muito similar a do Python e nela os algoritmos de aprendizado de máquina são apresentados como grafos computacionais (GOLDSBOROUGH, 2016). Os benefícios desse tipo de estrutura é que não só as funções estruturais de redes neurais já estão prontas, como também é muito mais fácil de visualizar as dependências do modelo. 3

No TensorFlow, os nós do grafo representam operações e as arestas, chamadas de *tensor*, representam os dados *fluindo* de um nó a outro.

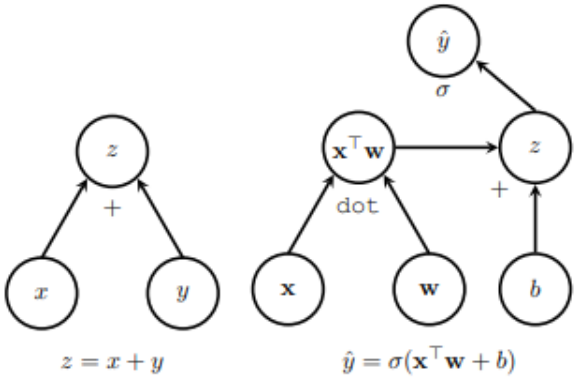


Figura 12 – Exemplo de grafo computacional.
Fonte: (GOLDSBOROUGH, 2016)

3.1.5.1 Keras

Keras é uma API de alto nível, escrita em Python, para *deep learning* que pode ser executada com o TensorFlow, sendo ele o *back-end* do Keras. O objetivo dessa biblioteca é tornar a implementação de redes neurais fácil, reduzindo as ações necessárias para implementar código comum e permitindo que os desenvolvedores foquem tanto nos principais conceitos de *deep learning*, pois fornece módulos integrados para todos os cálculos de rede neural, quanto na computação envolvendo os tensores, grafos computacionais, entre outros.

3.1.6 NodeJS

Uma das tecnologias mais utilizadas no desenvolvimento de aplicações *server-side* atualmente é o NodeJS. Basicamente é um ambiente de execução de Javascript fora do navegador web, muito utilizado para aplicações multidirecionais como chats e transmissão e visualização de dados.

Sua execução é *single-thread*, ou seja, apenas uma thread, chamada Event Loop, é responsável por tratar as requisições recebidas. Cada requisição é tratada como um novo evento e, por isso, com o NodeJS é possível tratar requisições concorrentes, através de chamadas de entrada e saída não-bloqueantes.

Então, para o desenvolvimento do lado do servidor do nosso aplicativo, usamos o nodeJS. Nosso servidor é composto por várias rotas que são utilizadas para armazenar ou solicitar dados, essas rotas são acessadas através de solicitações HTTP. Assim se, por exemplo, enviarmos uma requisição HTTP do tipo POST com os dados de um novo alerta para o link em nosso servidor com o final /alertas esses dados serão salvos no banco de dados e o servidor os retornará para indicar sucesso.

3.1.7 ReactJS

ReactJS é a mais popular biblioteca *front-end* de JavaScript, implementada para o desenvolvimento de interfaces de usuários para sites e aplicações. Ela permite o desenvolvimento de aplicativos com melhores experiências do usuário, rapidez e robustez e pode ser integrada a outras bibliotecas e frameworks, como AngularJS.

O principal conceito do React é uma árvore baseada em componentes JavaScript que imita uma árvore DOM, (*Document Object Model*), conhecido como VDOM ou DOM virtual. DOM virtual permite que a manipulação de elementos seja feita no objeto armazenado na memória e não no navegador, melhorando a performance da aplicação (AGGARWAL, 2018).

3.1.8 SendGrid

SendGrid é um provedor SMTP (*Simple Mail Transfer Protocol*) de envio de e-mails, têm sua plataforma baseada em nuvem e possui uma infraestrutura escalável. Já sendo utilizado por grandes empresas como Spotify e Uber, se consolidou no mercado pela confiabilidade na entrega, permitindo testes e gerenciamento de listas aos clientes e por possibilitar aos desenvolvedores criar e testar a API de e-mail ou usar a integração SMTP. Nesse trabalho, o SendGrid foi utilizado para o envio de alertas programados pelo usuário relativos a um veículo específico e vai ser melhor detalhado na seção 3.2.

3.1.9 Yolov5

É um modelo de detecção de objetos de um estágio que fornece maior precisão e resultados mais rápidos na implementação de detecção de veículos em tempo real. (MAHTO *et al.*, 2020). O YOLOv5 foi publicado em maio de 2020 por Jocher (2020) e é uma das versões mais utilizadas do YOLO *(You Only Look Once)*, proposto em 2015 por Redmon *et al.* (2015), por ter alcançado a perfomance de estado da arte no *dataset* COCO.

O YOLO é baseado em uma única *Convolutional neural network* (CNN). A CNN divide uma imagem em regiões e, em seguida, prevê as caixas de limite e probabilidades para cada região. O YOLO vê a imagem inteira durante o tempo de treinamento e de teste para codificar implicitamente informações contextuais sobre as classes e aparências.

A arquitetura de um detector de objetos de um estágio moderno usualmente é composta por um *backbone*, pré-treinado no ImageNet e uma *head*, responsável por prever classes e caixas delimitadoras de objetos (BOCHKOVSKIY; WANG; LIAO, 2020). O YOLOv5 foi feito utilizando o framework PyTorch e, por sua vez, consiste de:

- Backbone: Bottleneck CSP (Cross Stage Partial Networks) e SPP (Spatial Pyramid Pooling)

- Neck: PANet (Path Aggregation Network)
- Head: YOLO, a saída gera caixas de detecção, indicando a categoria, coordenadas e confiança.

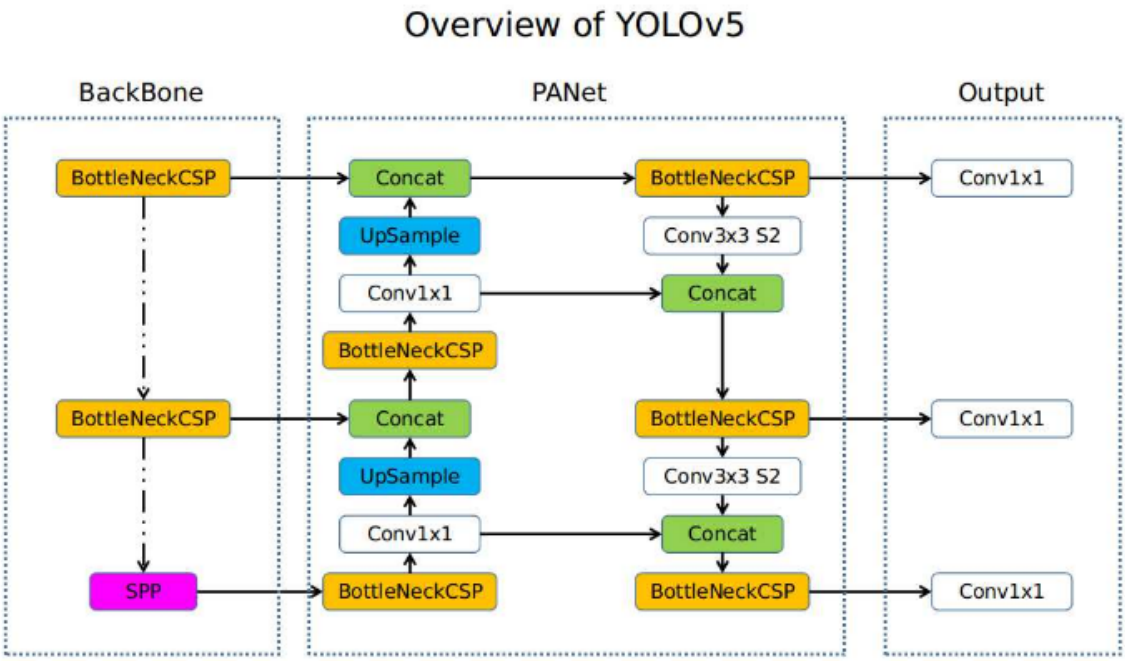


Figura 13 – Arquitetura do YOLOv5
Fonte: Disponível em (JOCHER, 2020) e criado pelo usuário "seekFire"

O Bottleneck CSP é utilizado para reduzir a quantidade de cálculo e aumentar a velocidade, enquanto a estrutura SPP realiza a extração de diferentes escalas para o mesmo mapa de recursos e pode gerar três escalas de mapas, o que ajuda a melhorar a precisão da detecção. O bloco de agregação de caminho PANet melhora a detecção de objetos com diferentes escalas, transmitindo recursos de localização de mapas de recursos inferiores e mapas de de recursos superiores bidirecionalmente. A saída do módulo *head* é usada principalmente para prever alvos de tamanhos diferentes em mapas de recursos (ZHAO *et al.*, 2021).

3.1.10 StrongSORT

MOT, do inglês Multiple Object Tracking é uma técnica experimental usada para estudar como nosso sistema visual rastreia vários objetos em movimento. DeepSORT, um dos primeiros métodos que aplicam *deep learning* a tarefa de rastreamento de multi objetos, alegadamente apresenta um desempenho inferior quando comparado com outros métodos mais atuais. Para contornar esses problemas e comprovar que o DeepSORT ainda pode ser utilizado, Du *et al.* (2022) apresentaram o StrongSORT, que nada mais é que um algoritmo de DeepSORT melhorado.

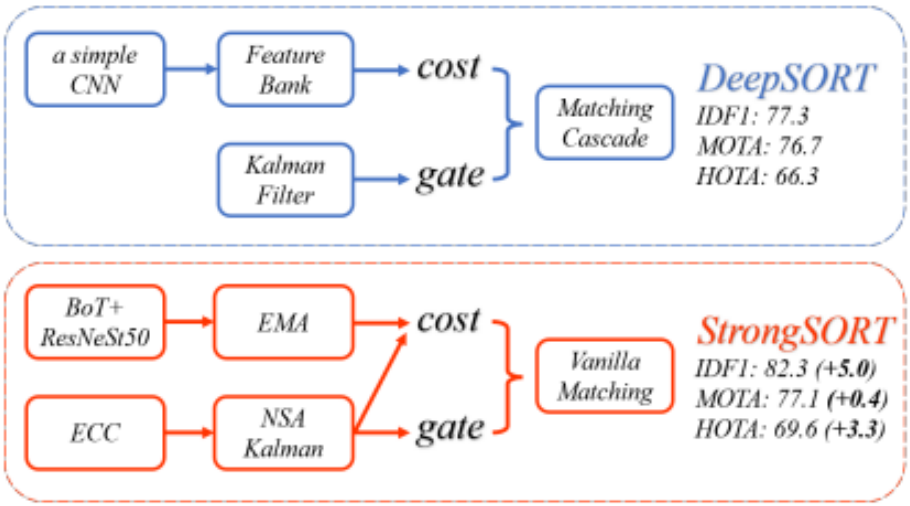


Figura 14 – Comparação do framework StrongSORT × DeepSORT
Fonte: (DU *et al.*, 2022)

Na imagem 14 podemos observar as principais alterações realizadas, principalmente nos braços de aparência e movimento. No primeiro, o backbone e o pré-treinamento que consistiam em uma CNN passaram a utilizar um BoT++ ResNeSt50, um extrator de recursos com objetivo de extrair mais características, e o *Feature Bank*, ou banco de recursos, foi substituído por uma estratégia de atualização de recursos EMA (*média exponencial móvel*), que reduz o consumo de tempo e melhora a qualidade de correspondência. Já no braço de movimento, foi adicionado um ECC (coeficiente de correlação aprimorado) para compensação do movimento de câmera e o filtro Kalman foi substituído pelo NSA Kalman para melhorar a detecção de ruído (DU *et al.*, 2022).

3.1.11 EfficientNet

EfficientNet é um modelo de CNN e usa uma técnica chamada de Coeficiente Composto para dimensionar uniformemente largura, profundidade e resolução, diferente da prática convencional que dimensiona arbitrariamente esses fatores. Usando o método de escalonamento e framework AutoML MNAS, a rede é ajustada para obter máxima precisão, mas pode se tornar um gargalo caso a rede seja computacionalmente pesada (TAN; LE, 2019). Os autores desenvolveram sete modelos que superaram a precisão de última geração da maioria das redes neurais convolucionais e com muito mais eficiência como podemos ver na figura 15.

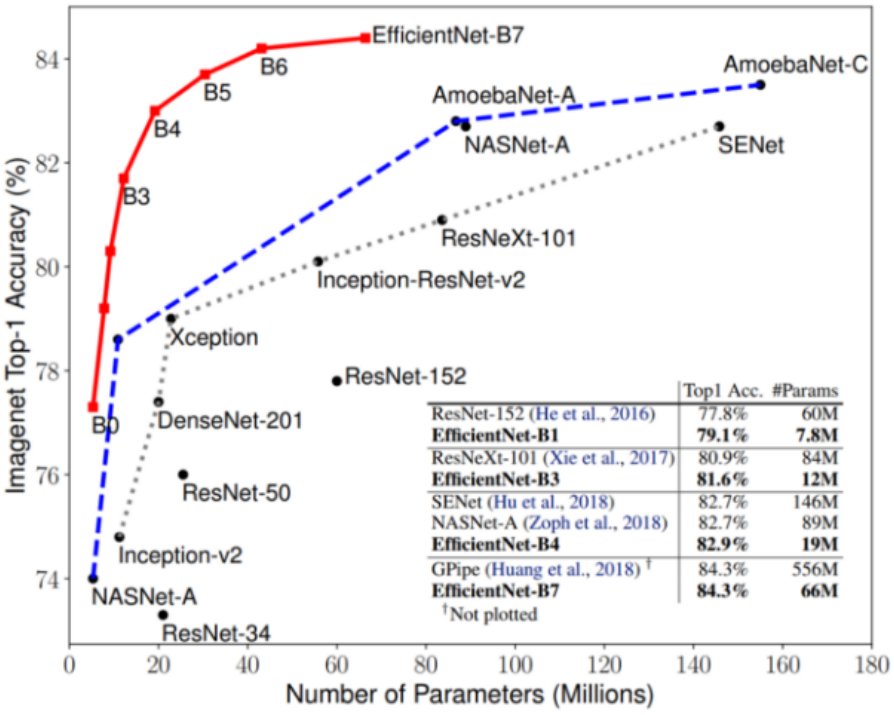


Figura 15 – Comparação de desempenho da Efficientnet com outras CNNs
Fonte: (TAN; LE, 2019)

3.2 Métodos

Esta seção apresenta o caminho percorrido para propor um sistema que resolva o problema levantado no capítulo 1. Para facilitar a compreensão do problema, e apresentar sua resolução, dividiu-se o trabalho em duas partes principais. A primeira parte envolveu uma revisão bibliográfica e definição e a modelagem da arquitetura proposta. A segunda parte detalha a implementação da infraestrutura, os módulos de identificação e a aplicação web, que fornece uma interface amigável com os usuários do sistema. A Figura 16 mostra uma visão geral da metodologia aplicada.

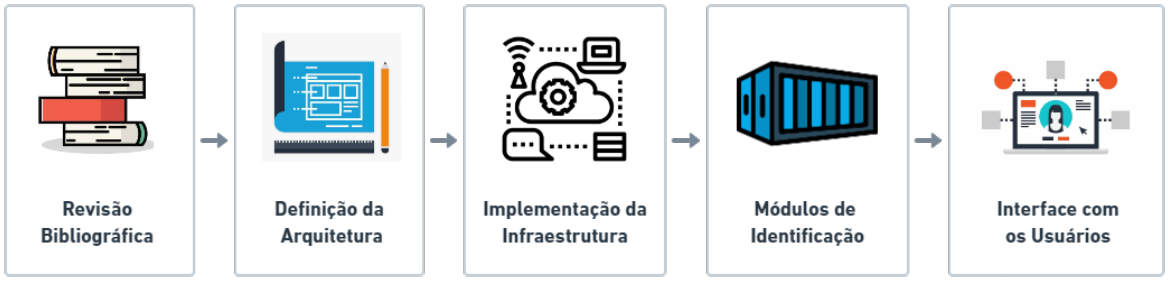


Figura 16 – Visão geral da metodologia aplicada.
Fonte: Autoria Própria

3.2.1 Definição da arquitetura

Diversas tecnologias que potencialmente atenderiam ao escopo deste projeto foram encontradas com a revisão bibliográfica realizada. Além disso, foram definidos os passos que deveriam ser seguidos pelo sistema para se obter a saída desejada:

- Detecção de veículos em frames
- Rastreamento (*tracking*) de veículos em fluxos de vídeo (vários frames em sequência)
- Detecção e leitura da placa dos automóveis
- Classificação da cor, categoria, marca e modelo dos automóveis
- Persistência das informações adquiridas para consultas posteriores
- Disponibilização dessas informações de forma facilitada para os usuários

A partir deste passo-a-passo, foi necessário definir os computadores a serem utilizados para realizar este processamento. Dentre os requisitos necessários para implantação deste projeto, precisavam-se de computadores com sistema operacional Linux e uma arquitetura capaz de permitir o processamento das entradas de vídeo em quase tempo real, além de uma forma fácil de escalar com base no número de entradas de vídeos simultâneas que seriam processadas. Optou-se por utilizar três computadores (ou três servidores, um *cluster*) do Laboratório de Inteligência Computacional (LABIC)[1] devido a disponibilidade destas máquinas para realização do trabalho e ao atendimento a requisitos técnicos como capacidade e poder computacional (CPU e GPU) e de memória. De acordo com a homepage, o LABIC é um laboratório de pesquisa vinculado ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI) da Universidade Tecnológica Federal do Paraná – UTFPR (antigo CEFET-PR) no campus Curitiba. As principais áreas de pesquisa desenvolvidas são: aprendizado profundo e aprendizado de máquina, visão computacional, computação evolutiva, mineração de dados e big data, Bioinformática. O contexto deste trabalho está inserido nas áreas de visão computacional, mineração de dados, e big data.

A Tabela 2 apresenta a configuração básica dos computadores utilizados.

Tabela 2 – Servidores disponíveis para implantação da solução
Fonte: Autoria Própria

Máquina	CPU	RAM	GPU	Rede
CPU10	Intel(R) Xeon(R) CPU E5-2450 v2 @ 2.50GHz	32GB	-	1Gbit
GPU10	Intel(R) Core(TM) i7-5820K CPU @ 3.30GHz	32GB	NVIDIA GeForce RTX 2060	1Gbit
GPU11	Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz	16GB	NVIDIA GeForce RTX 2060	1Gbit

Para realizar o processamento distribuído e orquestração de contêineres decidiu-se, respectivamente, pelas tecnologias Docker e Kubernetes. Dessa forma, é possível usar os três

[1] https://labic.utfpr.edu.br/

computadores de maneira mais transparente e escalável, tanto horizontalmente (adicionando mais contêineres no sistema), quanto verticalmente (adicionando mais poder de processamento com outras máquinas).

O gerenciamento de fluxos de vídeos é realizado pelo Apache Kafka porque ele facilita a capacidade de escala e distribuição do sistema com seu sistema de tópicos e tem uma alta velocidade de transmissão de dados sendo utilizado por grandes empresas (SHREE *et al.*, 2017). Para persistência dos dados e posterior consulta, optou-se por utilizar o MongoDB para guardar os dados já processados. O trabalho relacionado a este publicado por Kossoski *et al.* (2022) utilizou Docker, Kafka, e MongoDB para realizar o processamento e armazenamento de metadados de faces com e sem máscara facial em um contexto de vigilância de pedestres acerca da pandemia do COVID-19. Os autores também utilizaram computadores do LABIC e tiveram sucesso no uso destas tecnologias para processamento com baixa latência e tempo de processamento em quase tempo real.

Uma sistema web *frontend* e *backend* foi desenvolvido para facilitar a consulta no banco de dados e recuperação de dados pertinentes. A Figura 17, apresenta uma visão geral de toda arquitetura deste sistema, onde os blocos azuis são os módulos desenvolvidos e os blocos brancos são os tópicos do Kafka.

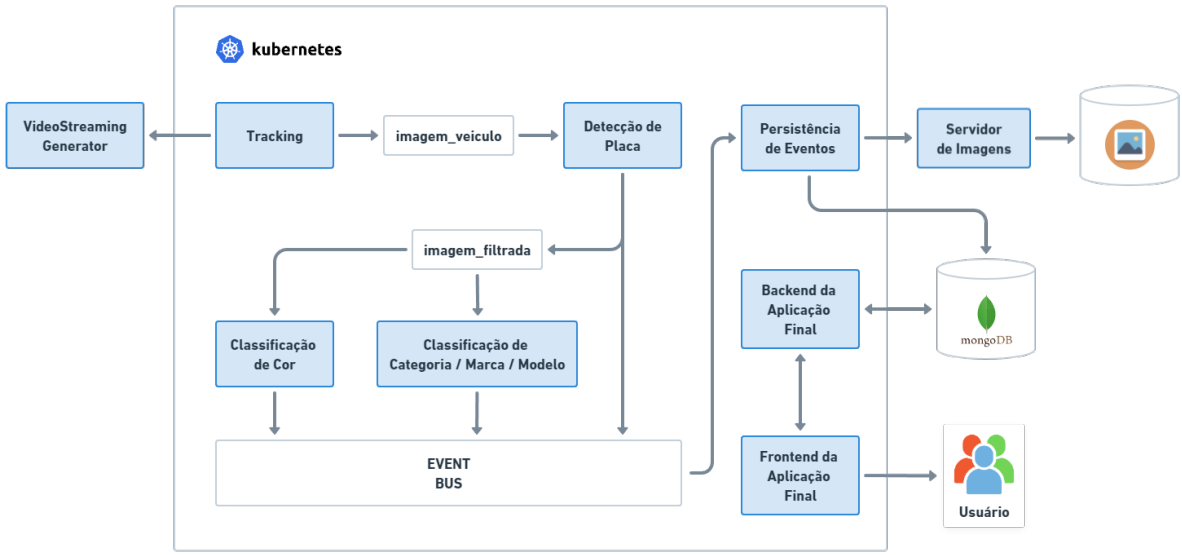


Figura 17 – Diagrama da arquitetura final do sistema
Fonte: Autoria Própria

3.2.2 Implementação da Infraestrutura

A implementação da infraestrutura necessária para rodar todos os módulos do sistema foi realizada depois de ter a arquitetura definida, a seguir são apresentados detalhes de cada tecnologia e configuração.

3.2.2.1 Kubernetes

A partir da definição dos computadores a serem utilizados, foi possível planejar e implantar um cluster Kubernetes com a ferramenta *kubeadm* (Kubernetes Contributors, 2022), que facilita a criação de um *cluster* Kubernetes seguindo boas práticas, com ela foi criado um *cluster* onde cada máquina tem as responsabilidades descritas na Tabela 3, basicamente temos a CPU10 que é a responsável por orquestrar e gerenciar as outras máquinas e os comandos enviados ao Kubernetes, enquanto as máquinas GPU10 e GPU11 são os trabalhadores responsáveis por executar os contêineres com base nos recursos disponíveis em cada uma. Para tornar possível a utilização das GPU's presentes nas máquinas foi necessário utilizar uma extensão da NVIDIA (NVIDIA Contributors, 2022) criada para uso dentro do sistema de orquestração do Kubernetes, visto que esse recurso não está disponível por padrão. Dessa forma torna-se possível a "solicitação" de GPU's no processo de implantação de contêineres que necessitam desse tipo de processamento dentro do *cluster* Kubernetes.

Tabela 3 – Responsabilidade das máquinas no *cluster* Kubernetes
Fonte: Autoria Própria

Nome da Máquina	Responsabilidade
CPU10	Master - Orquestração
GPU10	Worker
GPU11	Worker

3.2.2.2 Kafka

A configuração e implantação do serviço Kafka foi realizada na sequência, quando definiu-se a plataforma de comunicação entre os módulos existentes no nosso sistema. A instalação do Kafka foi realizada dentro do *cluster* Kubernetes, dessa forma o próprio Kafka poderia escalar de forma facilitada caso necessário, com aumento do número de *brookers* e/ou *zookepers*, visto que os mesmos são *pods* dentro do sistema Kubernetes. Além disso foi criada uma interface com a rede externa do *cluster*, para que o Kafka pudesse ser utilizado em programas dentro da rede do LABIC, facilitando o teste de *scripts* sem a necessidade de conteinerização. Feito isso, o Kafka estava pronto para ser usado por qualquer módulo dentro ou fora da orquestração do Kubernetes.

3.2.2.3 MongoDB e Servidor de Imagens

A terceira etapa foi a instalação das formas de persistência do sistema. A primeira delas é o banco de dados NoSQL MongoDB responsável por guardar todas as informações já processadas pelos nossos módulos de identificação, detecção, leitura e classificação. A segunda é um servidor de imagens, responsável por receber as imagens dos automóveis que fossem

processados e guardar essas imagens de forma que pudessem ser acessadas posteriormente por outros serviços. Ambos foram instalados na CPU10 que foi definida como máquina principal do nosso sistema e portanto ficaram disponíveis dentro da rede de computadores do LABIC. Além disso foi cirada uma interface dentro do *cluster* Kubernetes para que os contêineres que estivessem sendo orquestrados pelo Kubernetes também pudessem acessar o banco de dados e o servidor de imagens.

3.2.2.4 Gerador de fluxo de vídeos

A última etapa foi a criação de um gerador de fluxo de vídeos. Esse gerador foi utilizado para realizar testes no sistema através de vídeos gravados, ele foi desenvolvido para ler qualquer documento de vídeo e gerar um fluxo transmitido por HTTP que poder ser lido por qualquer outro consumidor, dessa forma é possível emular câmeras remotas utilizando arquivos em formatos de vídeo padrão como .mp4, .wmv, .mkv e etc. Utilizando esse gerador foi possível testar o sistema sem depender de fluxos em tempo real.

3.2.3 *Datasets*

Após o estudo das referências bibliográficas, foi decidido usar o método de classificação através de redes convolucionais, para isso foi necessário o uso de dois *Datasets*, um para o treinamento da classificação de cores e outro para o treinamento da classificação de categorias, marcas e modelos.

O conjunto de dados para treinamento da classificação de cores foi o *VCoR (Vehicle Color Recognition) Dataset* (PANETTA *et al.*, 2021), criado com imagens retiradas do Google que foram posteriormente limpas, pós-processadas, anotadas e consolidadas em um conjunto de dados. Ele contém mais de 10 mil amostras de imagens e 15 classes de cores, quase duas vezes mais diversificadas que o maior conjunto de dados existente (PANETTA *et al.*, 2021). As 15 categorias de cores representam os modelos de cores de automóveis mais populares de acordo com a CarMax, incluindo: branco, preto, cinza, prata, vermelho, azul, marrom, verde, bege, laranja, dourado, amarelo, roxo, rosa e bronze (PANETTA *et al.*, 2021). Um exemplo das cores encontradas nesse conjunto de dados pode ser visto na Figura 18.

Durante os testes de treinamento e o estudo do conjunto de dados foram realizadas algumas modificações, visto que algumas classes eram muito parecidas, e outras tinham uma probabilidade muito baixa de serem encontradas no cotidiano, então no final o conjunto de dados foi modificado para possuir as cores: amarelo (junção entre amarelo e dourado), azul, branco, laranja, marrom, prata (junção de prata e cinza), preto, verde e vermelho. Além disso foi realizado um rebalanceamento, deixando em média 900 imagens de cada classe para treino/validação e 150 imagens para teste. O balanceamento de imagens por classe é muito importante e tem influência direta na qualidade final do modelo (KOSSOSKI *et al.*, 2022).

Figura 18 – Exemplos de imagens do Dataset de cores
Fonte: (PANETTA *et al.*, 2021)

Já o conjunto de dados para treinamento das classificações de categoria, marca e modelo foi construído com base no Dataset UTFPR-CMMD (ALBINI; GUTOSKI; LOPES, 2019) construído por alunos do LABIC em 2018. Foram adicionadas nesse dataset novas marcas e modelos de carros, além de novas imagens seguindo o seguinte fluxo.

Definiram-se novos modelos a serem adicionados ou removidos com base no site *Webmotors* que demonstra a popularidade de certos modelos ou marcas de carros, e assim poder determinar se essa classe deveria ser retirada ou mantida do dataset. A próxima etapa do processo foi a de confirmar a presença de automóveis e recortar as imagens, centrando os carros e desconsiderando o fundo, para essa função, foi escolhida a abordagem de se utilizar o algoritmo de detecção de objetos YOLOv5 para detectar os automóveis, criar o *bounding box* e recortá-lo de forma automática, para esse projeto foi utilizado um modelo já pré-treinado do YOLOv5 disponível no PyTorch Hub.

Após essa etapa foi realizada uma revisão manual para detectar imagens erradas que pudessem passar despercebidas pela YOLO, como carrinhos de brinquedo ou de jogos online. Na sequência, um novo script em python foi escrito, aonde imagens que possuíam um *aspect ratio* irregular para uma imagem de carro ou possuíam uma resolução muito baixa eram consideradas “lixo” e assim removidas. As demais imagens foram redimensionadas, ajustando a largura para manter a mesma proporção que a original, usando a altura de 480px como referência. Com isso a penúltima etapa foi a de melhorar a distribuição de imagens por modelos, fazendo com que cada classe tivesse em média 137 imagens. Por fim, os modelos foram separados em categorias definidas como: HATCH, MINIVAN, PICAPE, SEDAN, SPORT e SUV. Podemos ver a representação dessas categorias na Figura 19

A Figura 20 apresenta o diagrama do fluxo apresentado nesta subseção. Após seguir todas essas etapas, temos uma nova versão do *Dataset* contendo 6 categorias de carros, 26 marcas e 304 modelos divididos entre 41679 imagens.

(a) HATCH (b) MINIVAN (c) PICAPE

(d) SEDAN (e) SPORT (f) SUV

Figura 19 – Exemplos de imagens do Dataset
Fonte: Autoria Própria

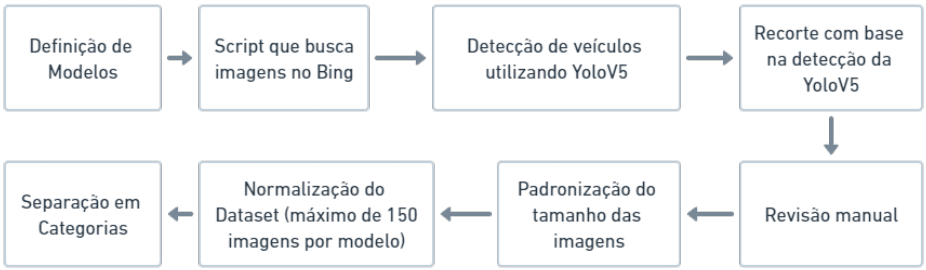


Figura 20 – Fluxo para melhora do conjunto de dados
Fonte: Autoria Própria

3.2.4 Módulos do sistema

Com a infraestrutura pronta, foi possível iniciar a implementação dos módulos do sistema que são responsáveis pelo processamento dos fluxos de vídeo produzidos pelo gerador. A Figura 21 apresenta uma visão geral dos módulos.

3.2.4.1 Tracking

O módulo de *tracking* tem duas responsabilidades. A primeira é detectar os carros que aparecem nos *frames* dos vídeos de entrada. Para isso, escolheu-se utilizar o modelo de detecção de objetos YOLOv5, que fornece uma ótima precisão e velocidade quando utilizado em GPU, conforme apresentado na seção anterior de materiais. A segunda responsabilidade é rastrear os veículos detectados durante todo percurso deles no vídeo. Para isso, foi utilizada a técnica StrongSORT, que melhora do método de *deep learning* DeepSORT, capaz de rastrear múltiplos objetos ao mesmo tempo. Na sequência, este módulo envia todas as imagens de automóveis encontradas pela detecção de objetos que tenham um tamanho mínimo e uma confiança mínima (para evitar o envio de imagens muito pequenas que não teriam muita utili-

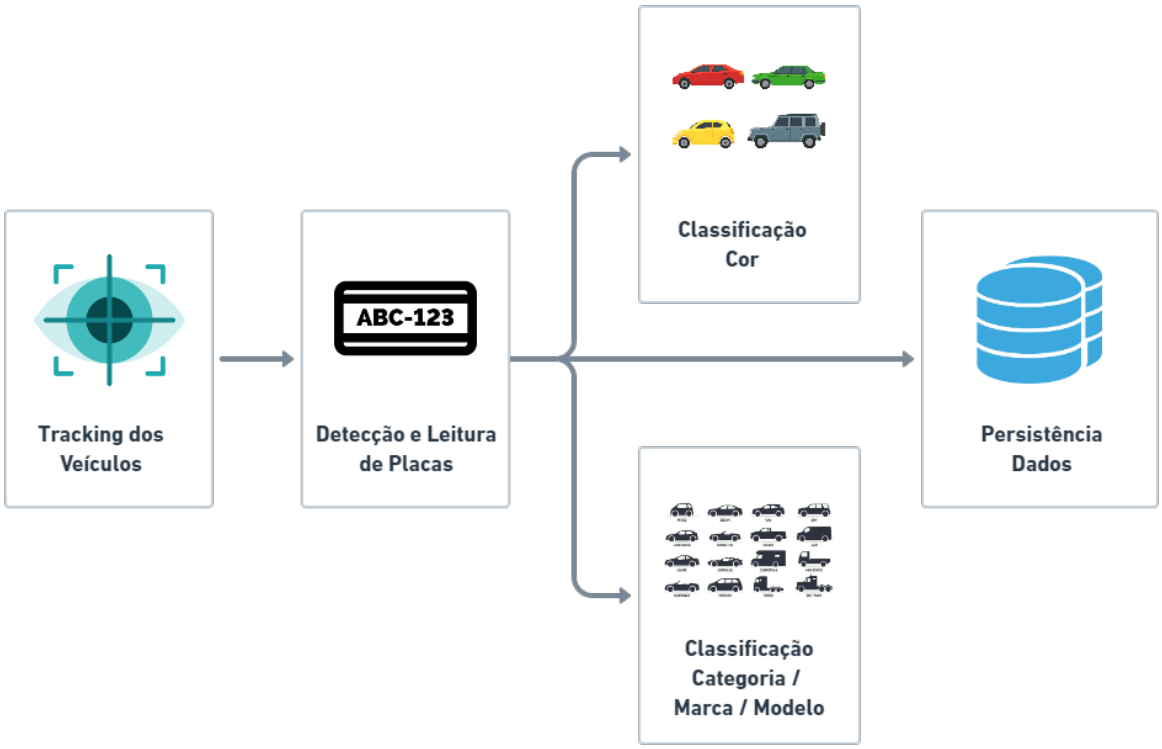


Figura 21 – Diagrama dos módulos do sistema
Fonte: Autoria Própria

dade) juntamente com o ID dado a elas pelo algoritmo de rastreamento para a próxima etapa de processamento utilizando um tópico do Kafka responsável por receber esses dados.

É possível ver um exemplo do processamento do módulo de *tracking* na Figura 22. Os objetos dentro das caixas coloridas foram reconhecidos pelo YOLOV5 e tem um rótulo que corresponde ao identificador dado pelo DeepSORT, seguido do nome do objeto e a confiança com que o objeto foi identificado.

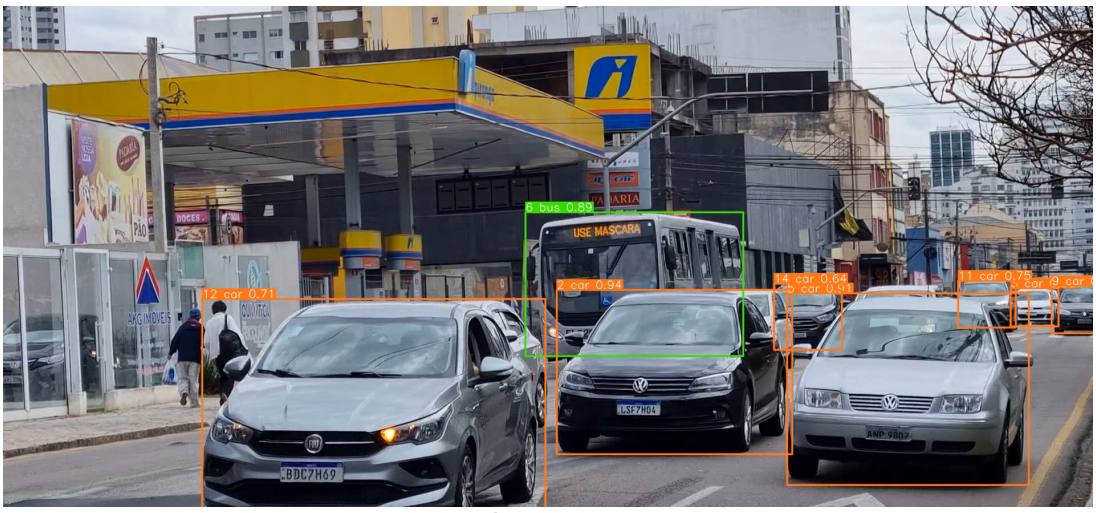


Figura 22 – Exemplo do módulo de tracking em funcionamento
Fonte: Autoria Própria

3.2.4.2 Detecção e leitura de placas

O módulo de detecção e leitura de placas foi baseado no projeto do aluno Moises Dias para detecção e reconhecimento de placas (Moises Dias, 2021), ele foi implementado em basicamente três passos.

O primeiro passo é a detecção da placa em uma imagem e o recorte da mesma, conforme apresenta a Figura 23.

Figura 23 – Primeiro passo - Detecção de placa
Fonte: Moises Dias

O segundo passo é segmentar os caracteres da placa. Para isso, primeiramente são executadas as ações descritas na Figura 24. Basicamente, a placa é colocada em tons de cinza, então ela é borrada, binarizada e, por fim, dilatada para preencher melhor os caracteres. Após o tratamento da imagem é utilizado um algoritmo que encontra os contornos da imagem e separa os caracteres, conforme pode ser observado na Figura 25.

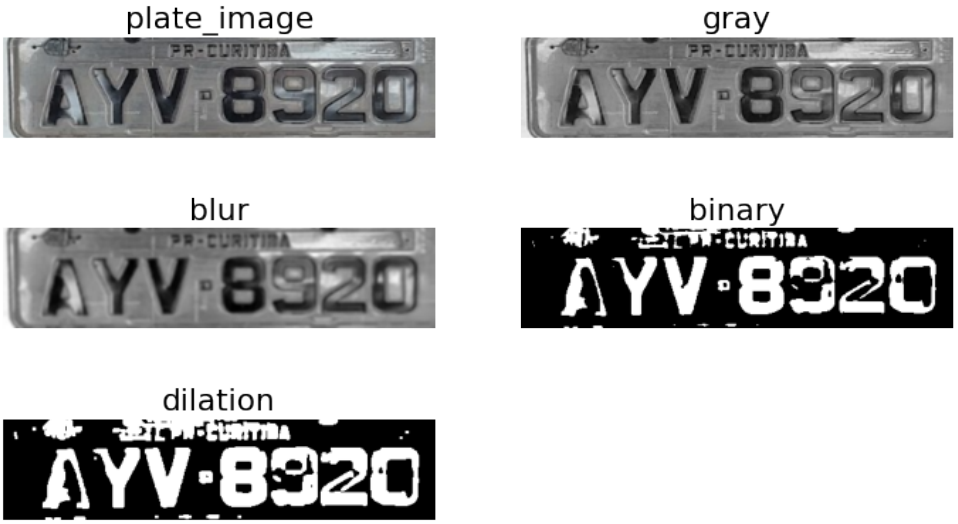


Figura 24 – Segundo passo - Tratamento da imagem
Fonte: Moises Dias

Figura 25 – Segundo passo - Separação dos caracteres
Fonte: Moises Dias

O último passo é classificar os caracteres que foram encontrados anteriormente. Para isso, são utilizados pesos já treinados da rede MobileNet para predizer cada carácter separado no passo dois, obtendo o resultado da Figura 26.

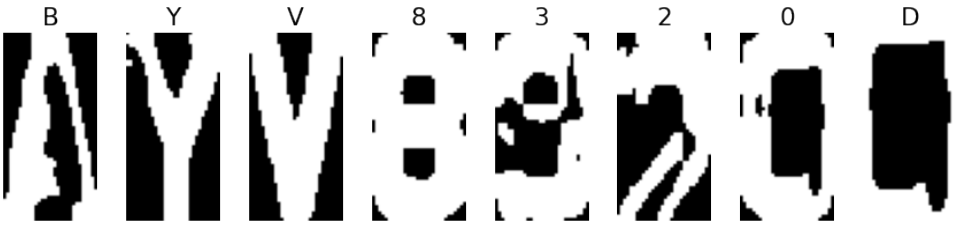


Figura 26 – Terceiro passo - Predição de Caracteres
Fonte: Moises Dias

Adicionalmente, foram implementadas algumas mudanças no código-fonte que deixariam esse módulo pronto para uso na arquitetura desenhada anteriormente. Primeiramente, foi adicionada a leitura de mensagens do tópico de imagens de automóveis que vem do módulo de *Tracking*, então a imagem é processada pela função de detecção e leitura de placas e publicada no tópico de imagens filtradas. Cabe ressaltar que apenas as imagens de automóveis com uma placa detectada são processadas, para evitar casos como o da Figura 27 que são passados pra frente pelo módulo anterior.

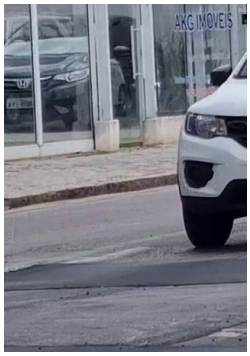


Figura 27 – Veículo cortado sem placa
Fonte: Autoria Própria

3.2.4.3 Classificações de Cor, Categoria, Marca e Modelo

Esta seção apresenta a descrição dos módulos que fazem as classificações de cor, categoria, marca e modelo. Todos eles foram desenvolvidos com uma metodologia parecida e por isso serão tradados nessa mesma seção. Para o seu desenvolvimento os seguintes passos foram seguidos:

1. Treinamento das redes para classificação de cor, categoria, marca e modelo.
2. Criação dos módulos de predição utilizando os modelos treinados e a leitura e envio de dados pelos tópicos do Kafka.

O primeiro passo é o treinamento das redes para classificação de cor, categoria, marca e modelo, para isso utilizamos os *Datasets* que foram mencionados anteriormente. A estratégia para utilização desses conjuntos de dados foi a seguinte: 80% dos dados foi usado para treinamento enquanto 20% dos dados foi separado para realização dos testes, isso garante que o conjunto de testes não será utilizado em nenhum momento do treinamento da rede reproduzido o que aconteceria durante o uso do modelo no cenário real, a divisão das imagens que ficaria em cada parte foi realizada de forma aleatória visto que os conjuntos de dados já eram bem distribuídos.

Realizada essa divisão podemos partir para o treinamento. Durante o treinamento o conjunto de dados é separado mais uma vez, mas agora em tempo de execução, ele é dividido mais uma vez em 80% para o treinamento em si e 20% para validação durante o treinamento. Esses conjuntos sofrem o processo de *data augmentation*, adicionando imagens com *flip* horizontal e uma pequena rotação. Além disso foi configurado um *early stopping* como forma de regularização, procurando evitar que o modelo se adapte de forma exagerada aos dados de treinamento, fenômeno também conhecido como *overfitting*. Vale ressaltar que todos esses procedimentos foram realizados tanto para o treinamento da classificação de cores quanto para classificação de categoria, marca e modelo.

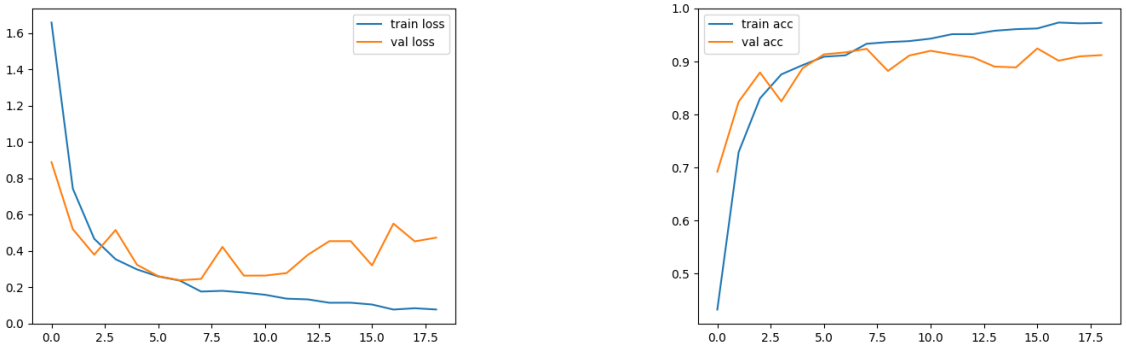
Focando primeiramente na classificação de cores de automóveis, foram treinadas duas redes conhecidas na literatura totalmente do zero, sendo elas a VGG16 e a Xcpetion, ambas com um número menor de camadas. Como a quantidade de imagens e classes era consideravelmente pequena, e as redes tinham pouca profundidade não foi necessário a utilização da estratégia de *transfer learning*. Na Tabela 4 podemos observar os parâmetros utilizados para o treinamento de ambos modelos.

Os resultados do treino podem ser observador com a rede VGG16 na Figura 28 que mostra o gráfico de *loss* e acurácia do treinamento, o mesmo pode ser observado na Figura 29 para a rede Xception.

Após ambos os treinamentos, o conjunto de dados de teste foi utilizado para verificar a acurácia do modelo em um conjunto que não tivesse sido utilizado no treinamento, os resultados

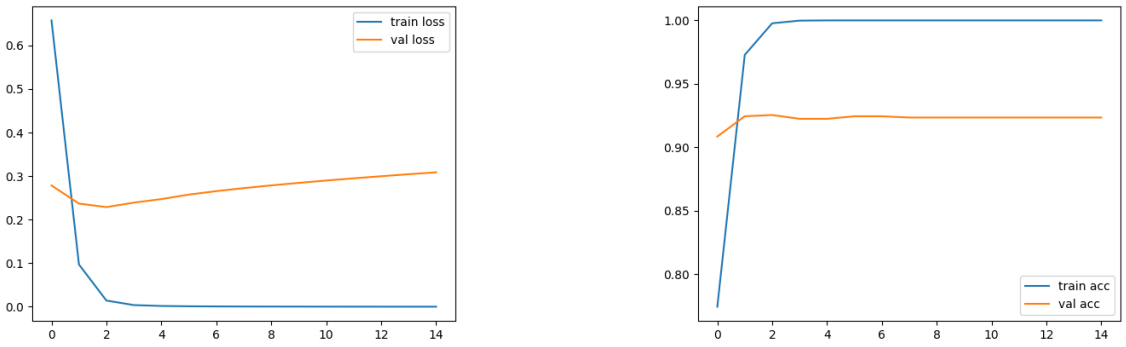
Tabela 4 – Parâmetros de treinamento do modelo de cores
Fonte: Autoria Própria

Parâmetros	Valor
Máximo de épocas	20
Early Stopping	12
Loss	Categorial Crossentropy
Otimizador	Adam
Learning Rate	0.0001
Beta 1	0.9
Beta 2	0.999
Batch Size	32



(a) Gráfico de *loss* **(b) Gráfico de *accuracy***

Figura 28 – Gráficos gerados pelo treino com a VGG16
Fonte: Autoria Própria



(a) Gráfico de *loss* **(b) Gráfico de *accuracy***

Figura 29 – Gráficos gerados pelo treino com a Xception
Fonte: Autoria Própria

estão apresentados na Tabela 5, que mostrou o modelo da Xception como sendo o superior e logo o escolhido para classificação de cores do sistema.

Tabela 5 – Resultados do treinamento das redes
Fonte: Autoria Própria

Modelo	Accuracy	Top-3 Accuracy
VGG16	90%	99.40%
Xception	93%	99,65%

Tratando da classificação de categoria, marca e modelo, foram treinadas algumas redes conhecidas na literatura utilizando o método de *transfer learning*, visto que a tentativa de treinar as redes do zero ocasionava o *overfitting*. As redes testadas foram Xception, InceptionV3, EfficientNetB1 e a EfficientNetV2S, a maioria delas com um número de camadas bem maior, devido a quantidade de classes (principalmente no treinamento de modelos) e imagens. A Tabela 6 apresenta os parâmetros utilizados para o treinamento de todos esses modelos.

Tabela 6 – Parâmetros de treinamento dos modelos de categoria, marca e modelo
Fonte: Autoria Própria

Parâmetros	**Valor**
Pesos base	*Imagenet*
Máximo de épocas	30
Early Stopping	10
Loss	Categorial Crossentropy
Otimizador	Adam
Learning Rate	0.0001
Beta 1	0.9
Beta 2	0.999
Batch Size	32

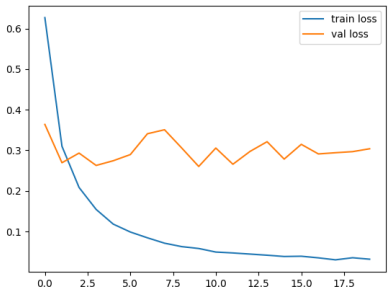
O conjunto de dados de teste foi utilizado depois de todos os treinamentos para verificar a acurácia dos modelos em um conjunto que não tivesse sido utilizado no treinamento. A Tabela 7 mostra uma comparação entre todos os modelos treinados para classificação de Categoria, Marca e Modelo. Com essa comparação, foi possível escolher os melhores pesos (em negrito na tabela) para cada classificação e assim prosseguir com a implementação dos módulos.

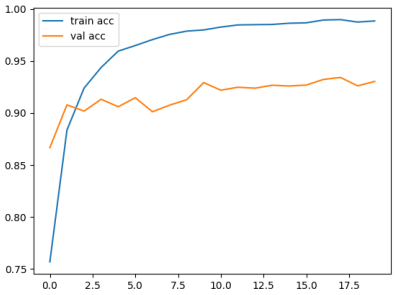
Tabela 7 – Resultados do treinamento das redes
Fonte: Autoria Própria

	Categoria		**Marca**		**Modelo**	
Modelo	**Accuracy**	**Top-3 Accuracy**	**Accuracy**	**Top-3 Accuracy**	**Accuracy**	**Top-3 Accuracy**
Xception	**93.82%**	**99.89%**	73.53%	89.89%	59.11%	78.63%
InceptionV3	92.42%	99.80%	78.05%	91.79%	54.98%	75.65%
EfficientNetB1	91.99%	99.82%	93.57%	98.12%	83.59%	94.67%
EfficientNetV2S	93.42%	99.84%	**94.62%**	**98.07%**	**87.86%**	**96.19%**

As figuras abaixo apresentam os gráficos de *loss* e *accuracy* das redes com melhor desempenho para categoria, marca e modelo.

O módulo de predição de cores é responsável por utilizar o peso da rede escolhida para predizer a cor do veículo recebido pelo tópico de imagens filtradas e enviar para o tópico de eventos. Já a classificação de categoria, marca e modelo ficam dentro do mesmo módulo, pois os resultados da classificação de categoria e marca são utilizados para definir o modelo do veículo.

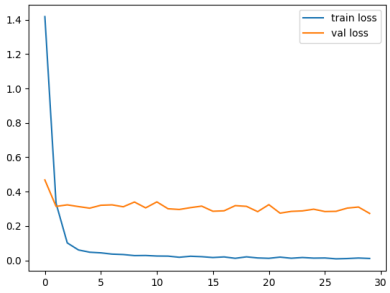


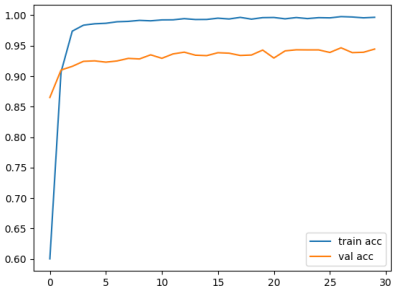


(a) Gráfico de *loss* **(b) Gráfico de *accuracy***

Figura 30 – Gráficos gerados pelo treino com a Xception para categorias

Fonte: Autoria Própria

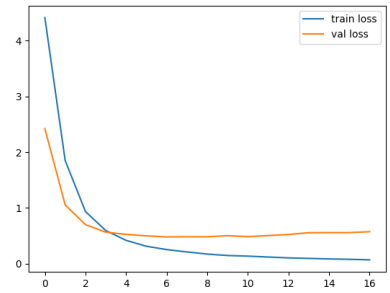


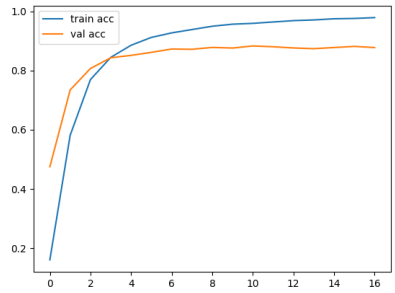


(a) Gráfico de *loss* **(b) Gráfico de *accuracy***

Figura 31 – Gráficos gerados pelo treino com a EfficientNetV2S para marcas

Fonte: Autoria Própria





(a) Gráfico de *loss* **(b) Gráfico de *accuracy***

Figura 32 – Gráficos gerados pelo treino com a EfficientNetV2S para modelos

Fonte: Autoria Própria

3.2.4.4 Persistência de eventos

O módulo de persistência de eventos é o último módulo deste sistema, responsável por armazenar a informação de todos os outros módulos, dessa forma eles não precisam efetuar conexão com o banco de dados, nem gastar tempo de processamento executando leituras e escritas no banco, além disso ele também é responsável por enviar e-mails de alertas para os usuários quando um automóvel que tenha as características configuradas pelo usuário é

encontrado. Para isso, ele consome os dados do tópico chamado *"Event Bus"* como é mostrado na Figura 17 de arquitetura final, ele diferencia os eventos recebidos por uma variável nas mensagens que chegam no tópico que pode ter os seguintes valores:

- ***PlateDetection***: evento enviado pelo módulo de detecção de placa;
- ***ColorClassification***: evento enviado pelo módulo de classificação de cor;
- ***CarClassification***: evento enviado pelo módulo de classificação de categoria, marca e modelo;

A cada evento que é lido por esse módulo ele segue a lógica mostrada no fluxograma da Figura 33.

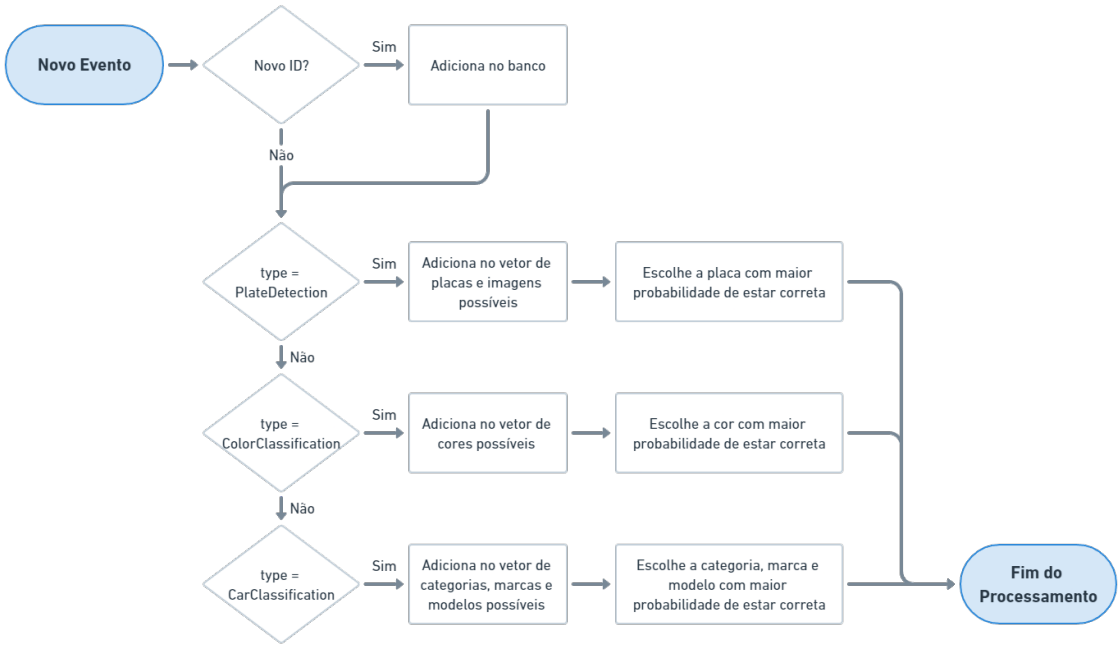


Figura 33 – Fluxograma da lógica do módulo de persistência de eventos
Fonte: Autoria Própria

Basicamente a cada evento recebido, se um novo ID é recebido um novo objeto criado no banco contendo somente a data de criação, data de modificação e o ID. Esse novo ID pode chegar com qualquer tipo de evento. Após verificar se o automóvel já existe no banco temos um processo para cada tipo de evento.

Para um evento do tipo detecção de placa, o módulo salva a imagem utilizando o servidor de arquivos e salvo o caminho pra ela na variável de possíveis imagens do banco, além disso ele adiciona a placa no vetor de todas as placas que foram encontradas para aquele automóvel e escolhe a que tem a maior probabilidade de estar correta, nesse caso é a que possui a formatação mais próxima da correta para placas brasileiras e que possui mais ocorrência.

Quando se trata de um evento do tipo classificação de cor, a cor e a sua confiança são adicionados no vetor de cores possíveis e dentro desse vetor é escolhida a cor que tem mais

chance de estar correta, pra isso é feita uma média de todas as confianças das mesmas cores e então escolhida a cor que tem a maior confiança média.

Para um evento de classificação de carro a categoria, a marca e o modelo são adicionados em seus respectivos vetores assim como na classificação de cor e depois também são escolhidas a melhor categoria, marca e modelo com base na média de confiança. Todas essas informações são salvas no banco de dados, utilizando o formato mostrado na Tabela 8.

Tabela 8 – Automóveis armazenados no banco de dados
Fonte: Autoria Própria

Campos	Tipo	Descrição
ID	Int	ID do veículo dado pelo módulo de *tracking*
Data Criação	Date	Data que esse objeto foi criado
Última Modificação	Date	Data da última modificação desse objeto
Posição	String	Posição onde o veículo foi avistado
Placa	String	Placa do veículo
Cor	String	Cor do veículo
Categoria	String	Categoria do veículo
Marca	String	Marca do veículo
Modelo	String	Modelo do veículo
Imagens Possíveis	String[]	Todas as imagens que foram processadas desse veículo
Placas Possíveis	String[]	Todas as placas retiradas das imagens processadas
Cores Possíveis	ObjectArray [{ cor: String, confiança: Float }]	Todas as cores e suas confianças de classificação retiradas das imagens processadas
Categorias Possíveis	ObjectArray [{ categoria: String, confiança: Float }]	Todas as categorias e suas confianças de classificação retiradas das imagens processadas
Marcas Possíveis	ObjectArray [{ marca: String, confiança: Float }]	Todas as marcas e suas confiança de classificação retiradas das imagens processadas
Modelos Possóveis	ObjectArray [{ modelo: String, confiança: Float }]	Todos os modelos e suas confiança de classificação retiradas das imagens processadas

Por fim, sempre que um automóvel para de receber atualizações de eventos por um tempo o módulo verifica se existe um alerta configurado para um automóvel com aquelas características, em caso afirmativo ele envia um e-mail para o usuário com todas as informações daquele veículo usando o serviço de e-mails Sendgrid.

3.2.5 Interface com usuários

Uma vez implementada a infraestrutura e os módulos responsáveis pelo processamento de dados do sistema, foi necessário criar uma forma de disponibilizar esses dados para o usuário final. Para isso, foi criada uma aplicação *web* composta por um *Frontend* e um *Backend* com base nos requisitos levantados, sendo eles:

- Possuir um sistema de login conectado com o acesso de usuários do *cluster* de computadores do LABIC.

- Possuir um mecanismo de busca versátil com base nas categorias extraídas do processamento de dados, sendo elas: placa, categoria, marcar, modelo, cor, data e localização.
- Possuir um *Dashboard* com análises básicas, como quantidade de carros por cor e por marca e com uma lista de automóveis encontrados, onde é possível visualizar informações específicas sobre determinado veículo quando necessário.
- Ter a capacidade de criar alertas com base em filtros para que seja enviado um e-mail quando um veículo com determinadas características for encontrado.

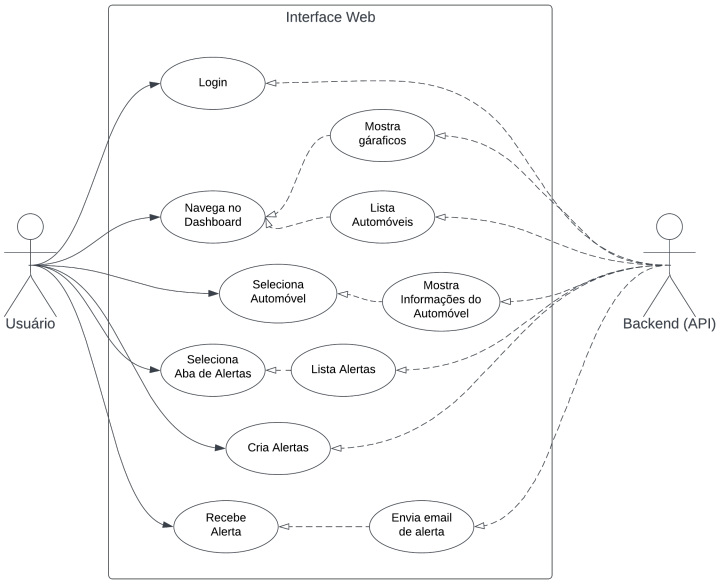


Figura 34 – Diagrama de Casos de Uso
Fonte: Autoria Própria

3.2.5.1 Frontend

O primeiro passo para o desenvolvimento da interface com o usuário foi a criação do diagrama de casos de uso que pode ser visto na Figura 34, com esse diagrama e os requisitos levantados foi possível iniciar a prototipagem de uma aplicação *web*, essa prototipação foi feita utilizando a ferramenta de *design* de interfaces Figma, e pode ser observada nas figuras 35, 36 e 37. Realizada a prototipação, a aplicação foi desenvolvida utilizando ReactJS, integrada com o Backend que será descrito a seguir, testada e implantada no *cluster* Kubernetes.

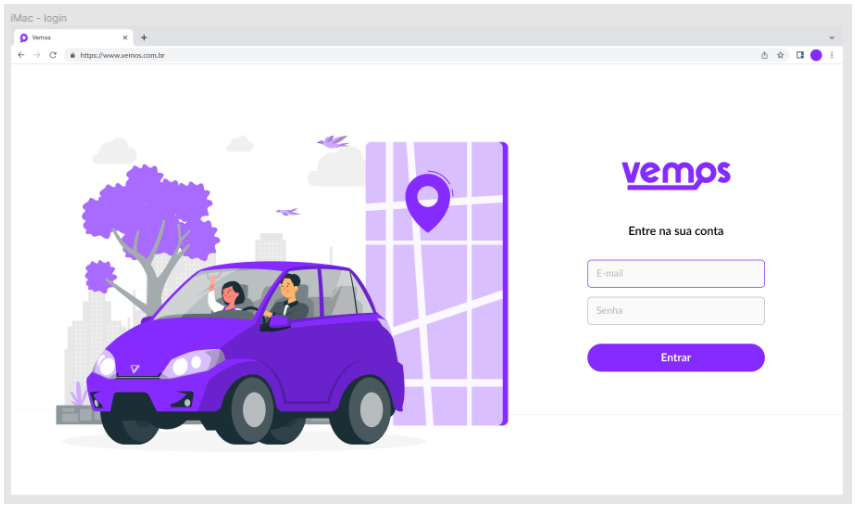


Figura 35 – Protótipo da página de Login
Fonte: Autoria Própria

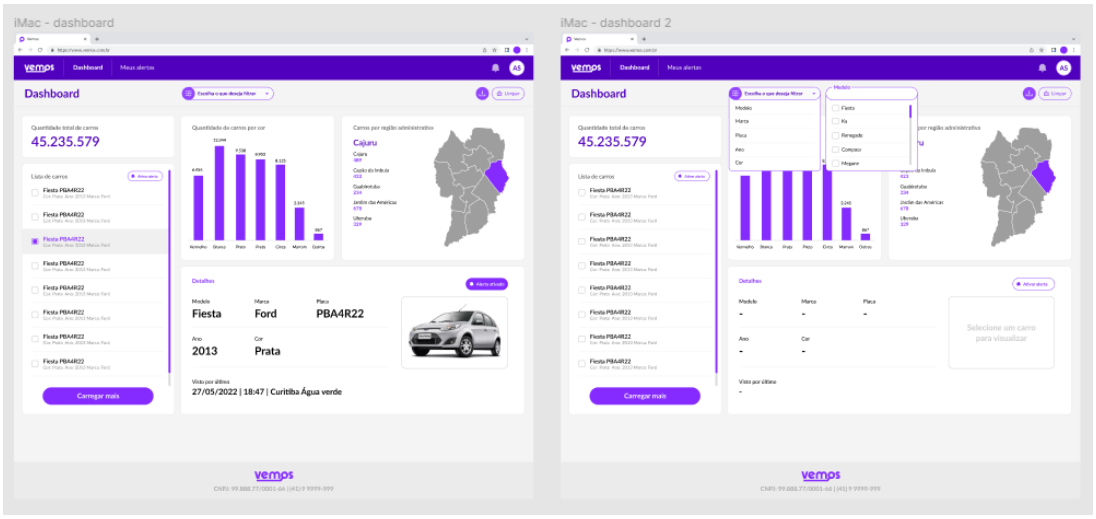


Figura 36 – Protótipo da página de Dashboard
Fonte: Autoria Própria

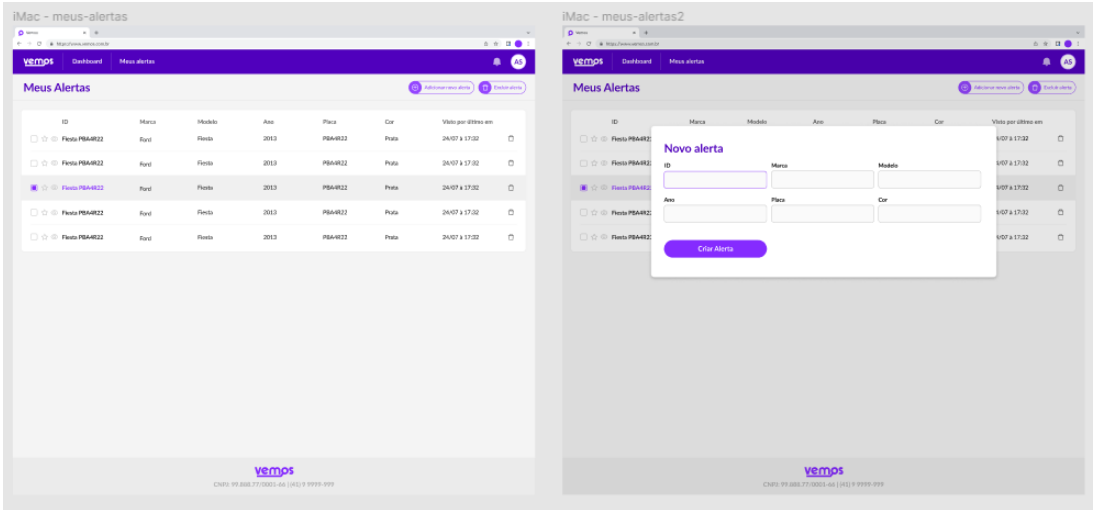


Figura 37 – Protótipo da página de Alertas
Fonte: Autoria Própria

3.2.5.2 Backend

Depois dos requisitos serem definidos, foi planejada a estrutura do *Backend* da aplicação. Na tabela 9 é possível ver as rotas que necessitavam ser desenvolvidas no *Backend*, cada rota possui uma responsabilidade na apresentação de informações e na interação do usuário com o *Frontend*. A coluna 'Rota' representa o final da URL do *Backend*, já a coluna 'Tipo' diz se a rota é do tipo GET, que não recebe nenhuma variável de entrada, ou do tipo POST, que recebe variáveis no corpo da requisição, a coluna 'Entrada' informa quais variáveis são esperadas pela rota enquanto a coluna 'Retorno' apresenta os dados que a rota devolve após chamada.

Tabela 9 – Rotas existentes no *Backend*
Fonte: Autoria Própria

Rota	Tipo	Entrada	Retorno
/**authenticate**	POST	Usuário e Senha	Token de autenticação do usuário ou Falha
/**vehicles/fields**	GET	-	Características possíveis
/**vehicles/values**	POST	Característica	Valores possíveis para essa característica
/**vehicles**	POST	Filtro de características	Lista de automóveis com base no filtro
/**vehicles/agg**	POST	Filtro de características	Agregação de dados com base no filtro
/**vehicles/count**	POST	Filtro de características	Contagem de dados com base no filtro
/**alerts**	GET	-	Lista de todos os alertas criados
/**alerts**	POST	Objeto alerta	Sucesso ou Falha
/**alerts/delete**	POST	ID do alerta	Sucesso ou Falha

O *Backend* foi desenvolvido utilizando NodeJS, integrado com o *Frontend*, testado e implantado no *cluster* Kubernetes. Vale ressaltar que por mais que o objeto de aplicação do trabalho sejam automóveis, e nesse caso as características estão ligadas a esses objetos, o funcionamento do *Backend* foi pensado para facilitar o uso para qualquer outro objeto de aplicação, sendo necessário fazer poucas alterações na implementação.

4 RESULTADOS

Nesse capitulo são apresentados os resultados que foram obtidos depois de aplicada a metodologia detalhada no Capítulo 4. Além disso, é realizada a comparação com resultados da literatura, assim como uma análise dos resultados. Vale ressaltar que todo código fonte do projeto, assim como a explicação de como usa-lo pode ser encontrado no repositório do Git Hub do projeto [1].

4.1 Infraestrutura

Após o desenvolvimento de toda infraestrutura do projeto, foram obtidos alguns resultados que valem ser comentados, principalmente pelo fato de facilitarem o desenvolvimento de projetos futuros que tem como objetivo o processamento de grandes quantidades de dados e de fluxos de vídeos. Os resultados foram os seguintes:

1. Serviço de criação de *streaming* de video: com ele é possível transformar qualquer arquivo de video em um fluxo HTTP, que pode ser processado por outros serviços dentro da rede do cluster de computadores do LABIC. O *streaming* de video gerado por esse serviço funciona como em *livestreams* encontradas na internet, podendo transferir vídeos em 1080p (FullHD) em tempo real.

2. Serviço capaz de guardar imagens dentro do cluster de computadores e disponibilizar essas imagens para acesso externo, de forma que possam ser acessados por usuários através de quaisquer tipos de interfaces, como no caso das imagens de automóveis que podem ser acessadas pela interface desenvolvida nesse projeto de qualquer local que possua acesso a internet.

3. O banco de dados MongoDB pode ser utilizado para leitura e escrita de dados de qualquer lugar da rede, além de possuir uma interface que permite acesso de qualquer local usando as credenciais corretas, essa interface pode ser observada na Figura 39.

4. O serviço Kafka pode ser utilizado para comunicação entre serviços de forma distribuída podendo alcançar grandes velocidades, se utilizado de forma correta. Assim como no caso do MongoDB, foi disponibilizada uma interface para tornar o uso e a configuração mais fáceis, a interface pode ser visualizada na Figura 40.

5. O *cluster* Kubernetes permite criação automática de conteineres, balanceamento de carga, e pode ser integrado com o serviço Kafka. A capacidade de especificar e solicitar o uso de recursos necessários para execução de um determinada aplicação, como GPU, CPU e memóra torna a gestão do cluster mais fácil e transparente.

[1] https://github.com/wagnerdriva/TCC2

Com os resultados de infraestrutura é possível facilmente distribuir o sistema para novas entradas de fluxos de vídeo como mostra a Figura 38, nesse caso se tivéssemos três entradas de fluxos de vídeo, seria possível escalarmos cada módulo entre todos os computadores do cluster Kubernetes com uma distribuição dos dados processados pelo kafka, com pequenas alterações nas configurações de cada módulo, possibilitando assim o processamento de todas as fontes de dados de forma simultânea.

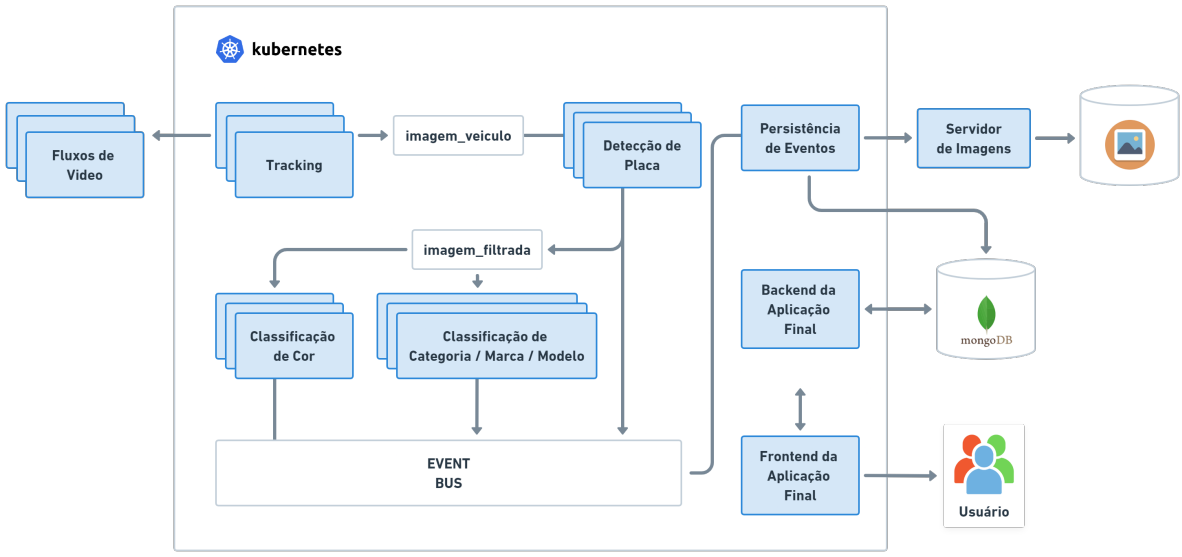


Figura 38 – Interface Kafka
Fonte: Autoria Própria

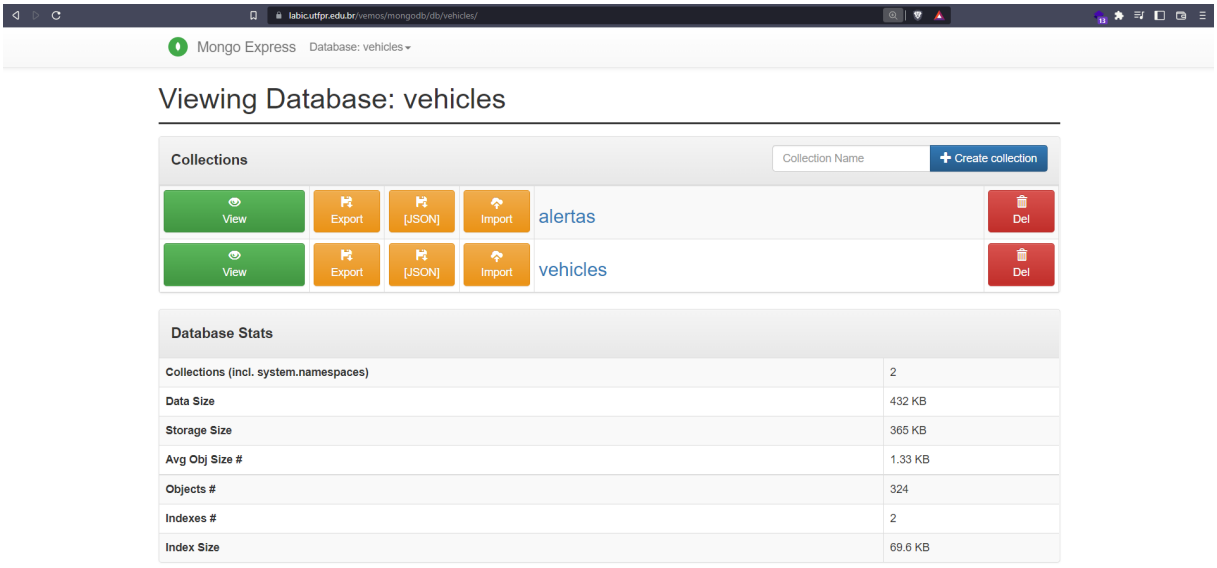


Figura 39 – Interface MongoDB
Fonte: Autoria Própria

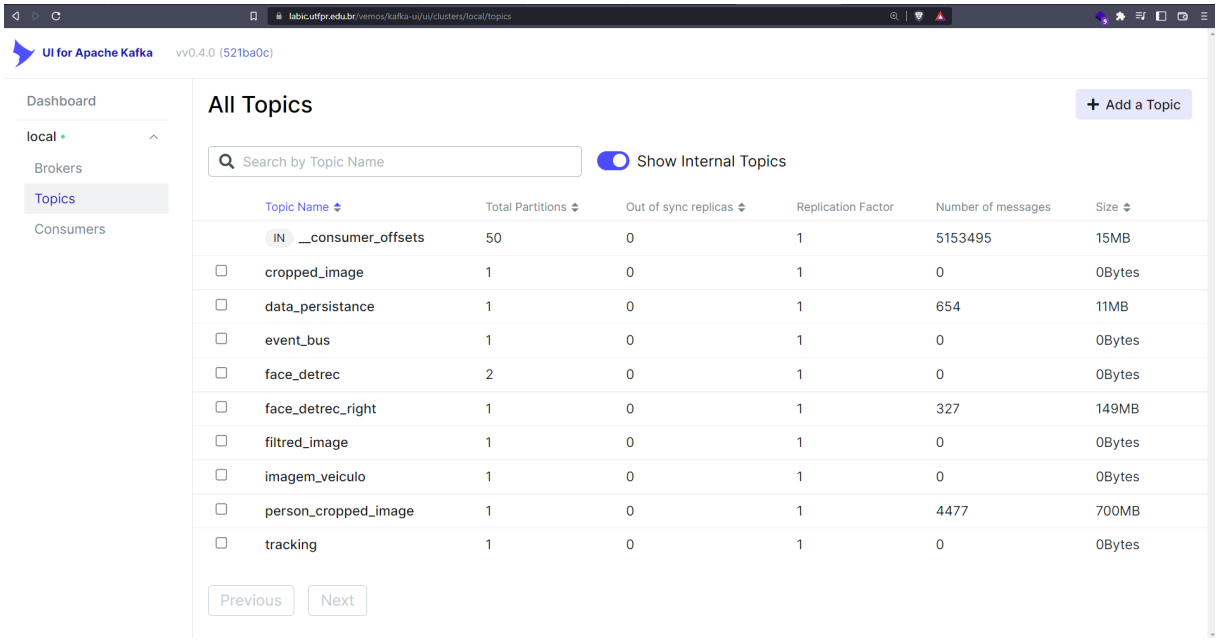


Figura 40 – Interface Kafka
Fonte: Autoria Própria

4.2 Módulos

Nessa seção, serão discutidos e analisados os resultados obtidos depois da implementação dos módulos de processamento de dados.

4.2.1 Módulo de *tracking*

O módulo de tracking é responsável por todo o filtro inicial do fluxo de vídeo, visto que ele deve identificar e rastrear todos os veículos que aparecem nesse fluxo. Isso torna este módulo um gargalo do sistema, visto que o algoritmo de rastreamento não pode ser distribuído, então para cada fluxo de vídeo podemos ter somente um módulo de tracking. Dado esse contexto, demos prioridade para que esse módulo rodasse utilizando uma das GPU's NVIDIA GeForce RTX 2060 disponiveis, a Tabela 10 nos mostra a velocidade de processamento desse módulo pela quantidade de veículos no frame sendo analisado, sendo que a velocidade de processamento é dividida em duas partes: a detecção de veículos no frame feita pela YOLOv5 e o rastreamento dos veículos feito pelo algoritmo StrongSORT.

Tabela 10 – Capacidade de processamento do módulo de tracking
Fonte: Autoria Própria

Quantidade de Veículos no Frame	Média Processamento YoloV5 (s)	Média Processamento StrongSORT (s)	Total (s)
0 veículos	0.010	-	0.010
3 veículos	0.025	0.070	0.095
8 veículos	0.050	0.200	0.250

Fazendo os testes e a média geral do processamento de 3 vídeos de 20 minutos, onde existiam momentos sem nenhum veículo assim como momentos contento 8 veículos simultaneamente, em FullHD (1080p), HD (720p) com 60fps, 24fps e 10 fps, foi possível concluir:

- A resolução do video praticamente não afeta a velocidade, visto que a imagem é reduzida para uma largura de 256px antes das inferências, e o tempo de execução dessa conversão é quase nulo comparado aos outros processamentos.
- O módulo de tracking tem a capacidade de ler o fluxo de vídeo com uma velocidade média de 10 frames por segundo, velocidades maiores que essa ele não é capaz de acompanhar, mas essa velocidade é suficiente para rastrear todos os veículos que aparecem nos vídeos.
- Essa velocidade é afetada pela capacidade da GPU. Em um teste realizado com uma GPU melhor, presente em outro computador do laboratório, percebemos uma melhora significativa na velocidade. Em casos de ruas muito mais movimentadas, uma melhora na GPU provavelmente seria necessária.

4.2.2 Módulo de detecção e leitura de placas

O módulo de detecção e leitura de placas é responsável por identificar se a imagem enviada pelo módulo de tracking contém uma placa e converter as imagens das placas detectadas em *strings*. A Tabela 11 mostra o tempo médio em segundos do processamento de cada imagem usando tanto a GPU quanto a CPU. Pode-se observar que há uma grande diferença no tempo de processamento, principalmente pelo fato de que para cada placa existente são necessários fazer em média 7 predições de caracteres utilizando a rede convolucional, deixando o módulo bem lento quando utilizando CPU. Vista essa grande diferença, e o fato de que ainda se tinha uma GPU disponível no sistema, optou-se por deixar esse módulo rodando na GPU restante. Como somente as imagens com um tamanho maior que 300 x 300px e uma confiança de 0.70 de que o objeto identificado é um carro são enviadas para esse módulo, não foi necessário o uso de mais de múltiplas instâncias desse script rodando, uma única foi suficiente.

Tabela 11 – Capacidade de processamento do módulo de placas
Fonte: Autoria Própria

Hardware	Tempo médio de detecção e leitura por imagem (s)
CPU	3.830
GPU	0.450

4.2.3 Módulo de classificação de cor

A única responsabilidade do módulo de classificação de cores é realizar a predição com base no modelo que foi escolhido durante o treinamento das redes convolucionais, a Tabela 12 nos mostra a comparação do resultado que adquirimos com o treinamento da rede Xception, com duas redes encontradas na literatura:

Tabela 12 – Comparação entre acurácias das redes
Fonte: Autoria Própria

Modelo	Acurácia
Xception	93.00%
(RACHMADI; PURNAMA, 2015)	94.47%
(CHEN; BAI; LIU, 2014)	92.82%

Por mais que a acurácia do modelo proposto não tenha alcançado a do modelo de (RACHMADI; PURNAMA, 2015), a velocidade de predição utilizando CPU do modelo proposto é em média 1.2 segundos enquanto a deles é de 3 segundos, tornando o modelo proposto bem mais eficiente em termos de velocidade de processamento. Como não se tinham mais GPU's disponíveis, esse módulo teve que ser rodado em CPU's, mas em contrapartida foi possível utilizar duas instâncias desse módulo rodando ao mesmo tempo, devido a facilidade de distribuir esse módulo através do Kubernetes e do Kafka.

4.2.4 Módulo de classificação de categoria, marca e modelo

O objetivo desse módulo era classificar a categoria, marca e modelo dos automóveis utilizando a predição das redes escolhidas durante o treinamento com base no conjunto de dados para as ruas brasileiras que foi criado, a única referência da literatura encontrada para o Brasil foi feita pelos próprios integrantes do LABIC (ALBINI; GUTOSKI; LOPES, 2019), tirando isso foi feita uma comparação com conjuntos de dados semelhantes aos nossos encontrado em Ni e Huttunen (2020), essa comparação pode ser vista na Tabela 13.

Tabela 13 – Capacidade de processamento do módulo categoria, marca e modelos
Fonte: Autoria Própria

	Acurácia		
Modelo	**Categoria**	**Marca**	**Modelo**
Modelos Escolhidos	93.82%	94.62%	87.86%
(ALBINI; GUTOSKI; LOPES, 2019)	-	82,36%	82,36%
(NI; HUTTUNEN, 2020)	90.1%	99.3%	94.30%

Pode-se ver que houve um grande avanço baseado na literatura brasileira, mas se tratando da comparação com a literatura global não houve grandes melhoras quando falamos de marcas e modelos, mas houve na parte de categorias. Vale ressaltar que a identificação em conjunto de categoria, marca e modelo leva em média 1.5 segundos sendo executada em CPU.

Assim como no caso da classificação de cores, não existiam mais GPU's disponíveis para o uso desse módulo, mas da mesma forma foi possível rodar duas instâncias de processamento ao mesmo tempo para tornar possível o processamento em tempo real.

Um ponto a ser ressaltado é que devido a termos imagens mais frontais dos automóveis, na prática a predição de categorias não funcionou tão bem, devido a falta de profundidade nas imagens, tornando a distinção entre automóveis *SEDAN* e *HATCH* bem complicada, um exemplo disso pode ser observador na Figura 41, onde um veículo *HATCH* foi classificado como *SEDAN*. Por outro lado, a detecção de marcas funcionou muito bem devido a, provavelmente, identificação das logos das marcas na parte frontal dos carros.

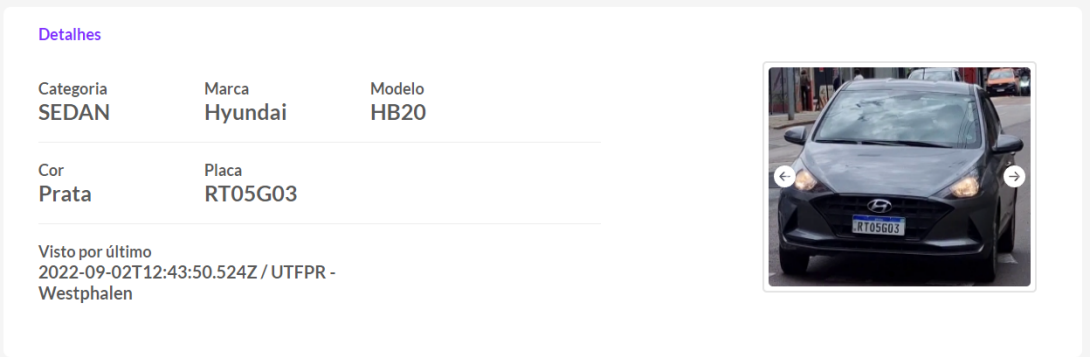


Figura 41 – Exemplo de veículo classificado com categoria errada
Fonte: Autoria Própria

4.3 Interface

A última parte de resultados foi a interface com o usuário, seguindo o desenvolvimento descrito na seção de metodologias foi possível desenvolver uma interface de fácil utilização e com um código fácil para ser reutilizado com outras bases no futuro. Primeiramente, têm-se uma tela de login que pode ser visualizada na Figura 42, é possível reparar que ela foi desenvolvida de forma idêntica ao protótipo mostrado anteriormente, outro ponto relevante é que os dados utilizados para login são os mesmos utilizados para acesso ao cluster de computadores do LABIC, evitando acessos não desejados, esses acessos são autorizados pelo servidor LDAP que já existia previamente no cluster.

Após o acesso, o usuário é levado a página que contém o dashboard geral, que pode ser visualizado na Figura 43, nele são mostrados o total de automóveis já encontrados, um gráfico de quantidade de automóveis por cores e um gráfico da quantidade de automóveis por marca, além disso têm-se uma lista com os primeiros 1000 automóveis encontrados, que podem ser selecionados para aparecerem no card do canto inferior direito que mostra todos os detalhes do veículo selecionado.

Para tornar a busca mais fácil, têm-se um sistema de filtros que podem ser aplicados como mostra a Figura 44, primeiramente temos um filtro de primeiro nível onde são escolhidas

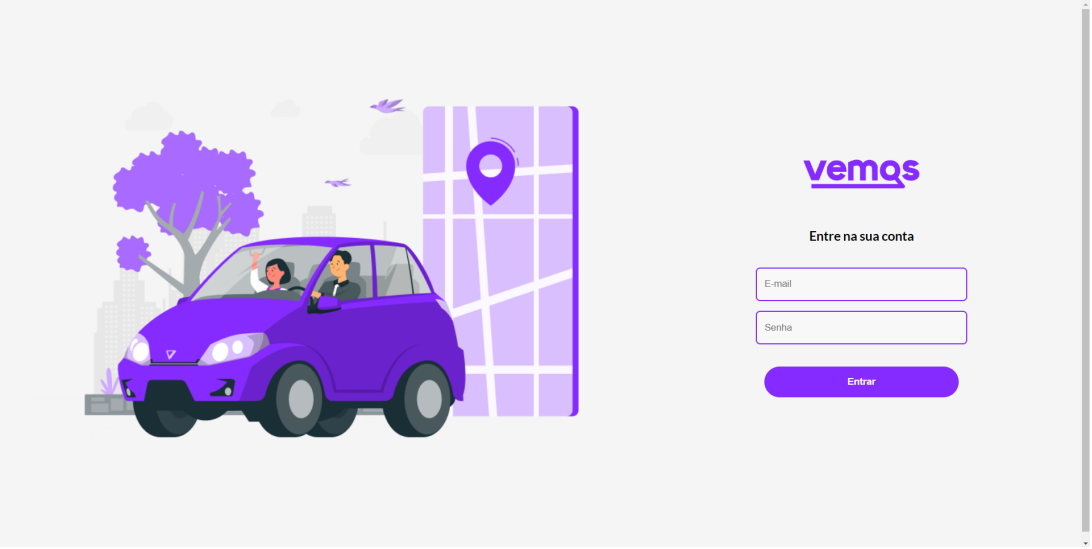


Figura 42 – Tela de login finalizadas
Fonte: Autoria Própria

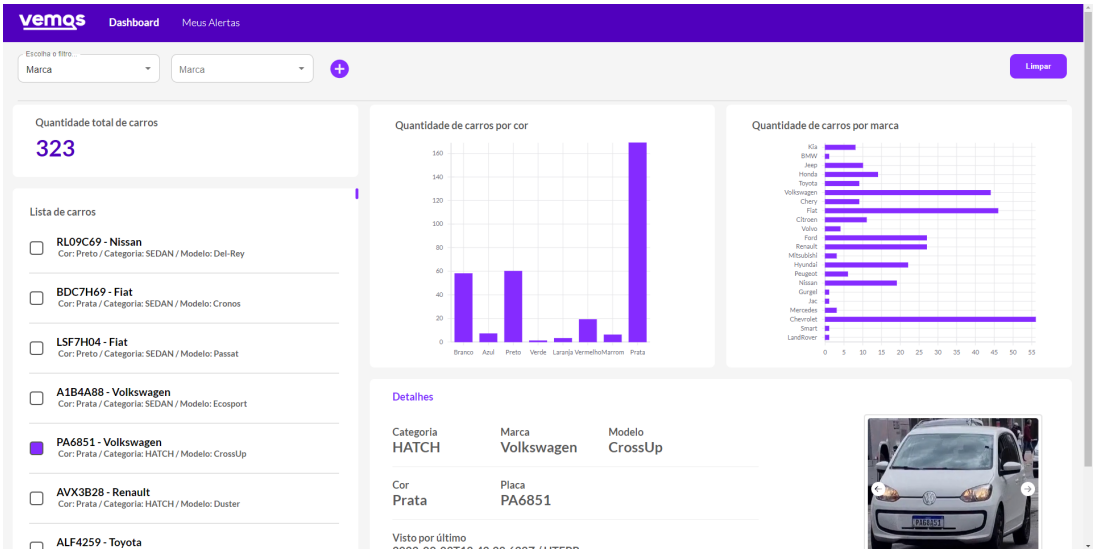
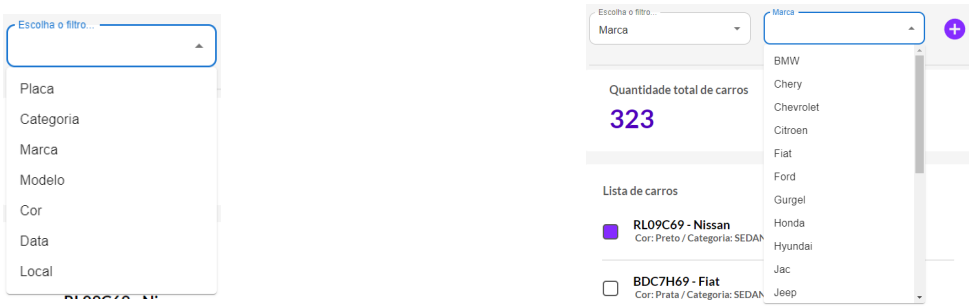


Figura 43 – Dashboard geral
Fonte: Autoria Própria

as variáveis que querem ser utilizadas no filtro, uma vez escolhida a variável são carregados de forma dinâmica os possíveis valores dessa variável que podem ser adicionados de forma a filtrar os resultados mostrados no dashboard como pode-se ver na Figura 45, onde é aplicado o filtro da marca *Citroen*. Um ponto interessante a ser ressaltado, é que como os filtros são criados de forma dinâmica, fica fácil a modificação do *backend* da aplicação para gerar filtros de qualquer objeto salvo no banco de dados.

Por último, têm-se a criação de alertas com base nas características dos automóveis, a tela de criação pode ser vista na Figura 46. Basicamente, é necessário colocar um e-mail, para o qual o alerta será enviado, e uma ou mais características que irão acionar o alerta. Dessa forma, sempre que um veículo com essas características for encontrado, um e-mail será enviado.



(a) Filtro primeiro nível **(b) Filtro segundo nível**

Figura 44 – Exemplo de seleção de filtros
Fonte: Autoria Própria

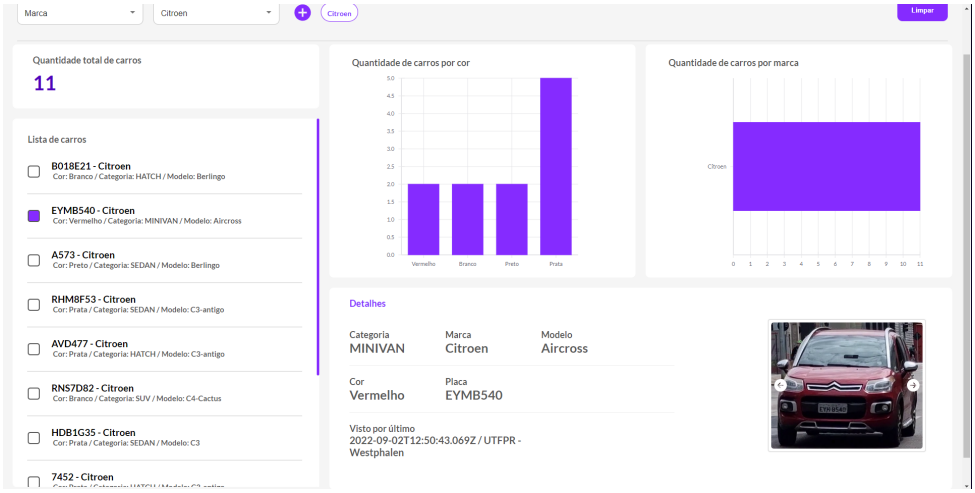


Figura 45 – Dashboard com filtros aplicados
Fonte: Autoria Própria

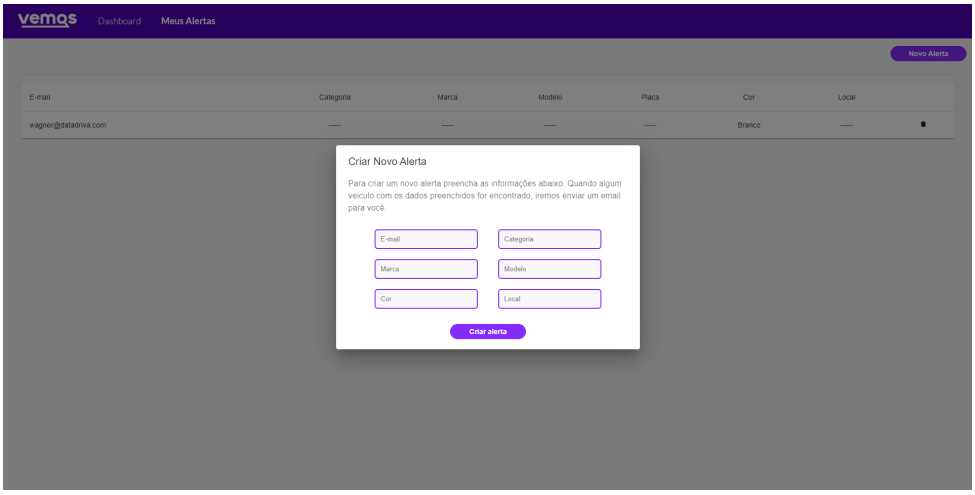


Figura 46 – Tela de criação de alertas
Fonte: Autoria Própria

5 CONCLUSÃO

O aumento constante da circulação de automóveis em ambientes urbanos faz com que necessidades voltadas ao controle, monitoramento e segurança de tráfego surjam. Por mais que com o avanço da tecnologia sistemas de monitoramento de veículos com câmeras de boa resolução tenham surgido, a capacidade de processar a grande quantidade de dados gerados por esse sistema e gerar conhecimento para ajudar na tomada de decisões ainda foi pouco explorada. Levando isso em consideração, o objetivo desse trabalho foi desenvolver um sistema de processamento de grandes conjuntos de dados aplicado a um sistema de detecção de categoria, marca, modelo, cor e placa de automóveis em fluxos de vídeo dentro do contexto de vigilância do mundo real.

Utilizando ferramentas como Kubernetes, Kafka e MongoDB, foi possível o desenvolvimento de uma infraestrutura capaz de suportar e facilitar o uso de processamento distribuído de fluxos de dados que além de utilizado nesse projeto, poderá ser utilizado em projetos futuros. Essa infraestrutura, no entanto, tem alguns pontos a serem considerados, como a necessidade de um conhecimento prévio sobre o uso do Kubernetes e do Kafka, que não são tão intuitivos a primeira vista e a capacidade limitada de alocação de GPU's pelo Kubernetes, visto que uma vez que uma GPU é solicitada por uma aplicação ela não pode ser utilizada por outra, mesmo que essa mesma GPU tenha memória e capacidade de processamento sobrando.

Além da infraestrutura, obteve-se como resultado um *pipeline* de dados capaz de identificar, rastrear e extrair características (placa, cor, categoria, marca e modelo) de automóveis em fluxos de vídeos gerados em tempo real. Esse *pipeline* utiliza toda capacidade da infraestrutura, rodando de forma distribuída e tornando possível escalar para o processamento de quantos fluxos de vídeos forem necessários. Vale ressaltar que durante seu desenvolvimento, foi criada uma nova versão do *Dataset* de carros circulantes brasileiros, o qual foi utilizado no treinamento das redes neurais convolucionais capazes de identificar, categorias, marcas e modelos de automóveis. Por mais que os resultados tenham sido satisfatórios, o posicionamento das câmeras de teste poderia ser melhorado, de forma a melhorar a classificação de categorias. O *Dataset* também tem seus pontos de evolução, visto que ele influencia diretamente na acurácia das redes treinadas, se tornando uma possível limitação da capacidade de acertar as características finais dos automóveis encontrados.

Para apresentar todos os resultados de forma a facilitar tomadas de decisões e gerar alertas com base nas características extraídas dos automóveis foi desenvolvida uma aplicação web utilizando as tecnologias NodeJS e ReactJS. A aplicação atendeu todos os critérios desejados, mas ainda tem muito espaço para o desenvolvimento de novas análises e até mesmo a apresentação de novas bases de dados.

Assim, foram alcançados todos os objetivos apresentados na Seção 1.1. Foram estudadas as técnicas e conceitos apresentados no Capítulo 2, que embasaram principalmente toda a parte de treinamento e escolha das ferramentas utilizadas, e permitiram a elaboração do passo-

a-passo do desenvolvimento e definição da arquitetura final. Foram estudadas também técnicas de deep learning, sendo algumas delas apresentadas no Capítulo 3. Por fim, o objetivo proposto por esse trabalho foi alcançado e a integração do sistema proposto foi realizada com sucesso, mas ainda pode-se ver vários pontos de melhoria para serem trabalhados futuramente.

5.1 Trabalhos Futuros

Como forma de continuação desse trabalho, as seguintes melhorias são propostas:

- Buscar formas de melhorar o desempenho do rastreamento de automóveis, estudando a possibilidade de distribuir o algoritmo.
- Buscar formas de melhorar o desempenho do algoritmo de detecção e leitura de placas, estudando a possibilidade de realizar as predições das letras/números da placa de forma distribuída.
- Usar a predição dos mais prováveis na escolha final dos atributos classificados por redes convolucionais, não somente o mais provável de cada classe.
- Integrar algum sistema que permita escrever *queries* em alto nível, como SQL-like ou buscas textuais.

REFERÊNCIAS

AGGARWAL, S. Modern web-development using reactjs. **International Journal of Recent Research Aspects**, v. 5, n. 1, p. 133–137, 2018.

ALBINI, L.; GUTOSKI, M.; LOPES, H. S. Car make and model classification with deep learning methods. *In*: FERNANDES, B. J. T.; JúNIOR, A. P. (Ed.). **Anais do 14 Congresso Brasileiro de Inteligência Computacional**. Curitiba, PR: ABRICOM, 2019. p. 1–7. ISBN 978-856997201-3.

ANURADHA, J. *et al.* A brief introduction on big data 5vs characteristics and hadoop technology. **Procedia Computer Science**, Elsevier, v. 48, p. 319–324, 2015.

BOCHKOVSKIY, A.; WANG, C.; LIAO, H. M. Yolov4: Optimal speed and accuracy of object detection. **Computing Research Repository**, abs/2004.10934, 2020. Disponível em: https://arxiv.org/abs/2004.10934.

CHEN, P.; BAI, X.; LIU, W. Vehicle color recognition on urban road by feature context. **IEEE Transactions on Intelligent Transportation Systems**, v. 15, n. 5, p. 2340–2346, 2014.

DONG, Z. *et al.* Vehicle type classification using a semisupervised convolutional neural network. **IEEE Transactions on Intelligent Transportation Systems**, v. 16, n. 4, p. 2247–2256, 2015.

DU, Y. *et al.* **StrongSORT: Make DeepSORT Great Again**. arXiv, 2022. Disponível em: https://arxiv.org/abs/2202.13514.

GOLDSBOROUGH, P. **A Tour of TensorFlow**. arXiv, 2016. Disponível em: https://arxiv.org/abs/1610.01178.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. Convolutional networks. *In*: **Deep learning**. [*S.l.*]: MIT Press Cambridge, MA, USA, 2016. v. 2016, p. 330–372.

GUINGO, B. C.; THOME, A. C. G.; RODRIGUES, R. J. Reconhecimento automático de placas de veículos automotores através de redes neurais artificiais. 2002. Disponível em: http://www.nce.ufrj.br/labic/downloads/2cbcomp_2002.pdf.

HIRAMAN, B. R.; M., C. V.; C., K. A. A study of apache kafka in big data stream processing. *In*: **2018 International Conference on Information , Communication, Engineering and Technology (ICICET)**. [*S.l.*: *s.n.*], 2018. p. 1–3.

HUANG, J. *et al.* Dvm-car: A large-scale automotive dataset for visual marketing research and applications. **arXiv preprint arXiv:2109.00881**, 2021.

HUANG, J. *et al.* **DVM-CAR Dataset**. [*S.l.*]: GitHub, 2021. https://deepvisualmarketing.github.io/.

HUANG, W.; ZHOU, J.; ZHANG, D. On-the-fly fusion of remotely-sensed big data using an elastic computing paradigm with a containerized spark engine on kubernetes. **Sensors**, v. 21, n. 9, 2021. ISSN 1424-8220. Disponível em: https://www.mdpi.com/1424-8220/21/9/2971.

IDC. **https://www.idc.com/getdoc.jsp?containerId=prUS47560321**. 2021. Disponível em: https://www.idc.com/getdoc.jsp?containerId=prUS47560321
urldate = 2022-04-04,.

JOCHER, G. Yolov5. 2020. Disponível em: https://github.com/ultralytics/yolov5.

KINGMA, D. P.; BA, J. **Adam: A Method for Stochastic Optimization**. arXiv, 2014. Disponível em: https://arxiv.org/abs/1412.6980.

KOSSOSKI, C. *et al.* A scalable analytics pipeline for COVID-19 face mask surveillance. **Learning & Nonlinear Models**, SBIC, v. 20, n. 1, p. 62–73, 2022.

KRAUSE, J. *et al.* 3d object representations for fine-grained categorization. *In*: **4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)**. Sydney, Australia: [*s.n.*], 2013.

KRAUSE, J. *et al.* **Cars Dataset**. [*S.l.*]: GitHub, 2013. https://ai.stanford.edu/~jkrause/cars/car_dataset.html.

Kubernetes Contributors. **Creating a cluster with kubeadm**. 2022. Disponível em: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/.

KUHN, D. M.; MOREIRA, V. P. Brcars: a dataset for fine-grained classification of car images. *In*: IEEE. **2021 34nd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)**. [*S.l.*], 2021.

LAKSHMANAN, V.; GöRNER, M.; GILLARD, R. **Practical Machine Learning for Computer Vision**. [*S.l.*]: O'Reilly Media, Inc., 2021. ISBN 9781098102364.

LOGWEB. **Abese Preve Crescimento do Número de Veículos Rastreados no Brasil**/. 2019. Disponível em: https://www.logweb.com.br/abese-preve-crescimento-do-numero-de-veiculos-rastreados-no-brasil/, urldate = 2021-07-18.

MAAS, A. L. *et al.* Rectifier nonlinearities improve neural network acoustic models. *In*: CITESEER. **Proc. icml**. [*S.l.*], 2013. v. 30, n. 1, p. 3.

MAHTO, P. *et al.* Refining yolov4 for vehicle detection. **International Journal of Advanced Research in Engineering and Technology (IJARET)**, SSRN, p. 409–419, 2020. Disponível em: https://ssrn.com/abstract=3628439.

MEDEL, V. *et al.* Modelling performance and resource management in kubernetes. p. 257–262, 12 2016.

Moises Dias. **Plate Detect and Recognize**. 2021. Disponível em: https://github.com/moises-dias/easyPark/tree/master/deteccao_placas/Plate_detect_and_recognize.

NI, X.; HUTTUNEN, H. Vehicle attribute recognition by appearance: Computer vision methods for vehicle type, make and model classification. **CoRR**, abs/2006.16400, 2020. Disponível em: https://arxiv.org/abs/2006.16400.

NVIDIA Contributors. **NVIDIA device plugin for Kubernetes**. 2022. Accessed: 2022-07-10. Disponível em: https://github.com/NVIDIA/k8s-device-plugin.

OUSSOUS, A. *et al.* Big data technologies: A survey. **Journal of King Saud University - Computer and Information Sciences**, v. 30, n. 4, p. 431–448, 2018. ISSN 1319-1578. Disponível em: https://www.sciencedirect.com/science/article/pii/S1319157817300034.

PANETTA, K. *et al.* Artificial intelligence for text-based vehicle search, recognition, and continuous localization in traffic videos. **Artificial Intelligence**, v. 2, n. 4, p. 684–704, 2021. ISSN 2673-2688. Disponível em: https://www.mdpi.com/2673-2688/2/4/41.

PHUNG; RHEE. A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets. **Applied Sciences**, v. 9, p. 4500, 10 2019.

RACHMADI, R. F.; PURNAMA, I. K. E. Vehicle color recognition using convolutional neural network. **Computing Research Repository**, abs/1510.07391, 2015. Disponível em: http://arxiv.org/abs/1510.07391.

REDMON, J. *et al.* You only look once: Unified, real-time object detection. **Computing Research Repository**, abs/1506.02640, 2015. Disponível em: http://arxiv.org/abs/1506.02640.

S, B. K.; A, P.; MURUGAN, D. **A review of Deep learning Techniques for COVID-19 identification on Chest CT images**. arXiv, 2022. Disponível em: https://arxiv.org/abs/2208.00032.

SACHETO, C. **Roubo de veículos ultrapassa marca de 1 milhão no Brasil em 4 anos**. 2019. Disponível em: https://noticias.r7.com/sao-paulo/roubo-de-veiculos-ultrapassa-marca-de-1-milhao-no-brasil-em-4-anos-11102019 Acesso em: 18 de julho de 2021.

SHREE, R. *et al.* Kafka: The modern platform for data management and analysis in big data domain. *In*: **2017 2nd International Conference on Telecommunication and Networks (TEL-NET)**. [*S.l.*: *s.n.*], 2017. p. 1–5.

SINESP. **Dados Nacionais de Segurança Pública - UF**. 2021. Disponível em: https://dados.gov.br/dataset/sistema-nacional-de-estatisticas-de-seguranca-publica/resource/feeae05e-faba-406c-8a4a-512aec91a9d1 Acesso em: 18 de julho de 2021.

TAFAZZOLI, F.; FRIGUI, H.; NISHIYAMA, K. A large and diverse dataset for improved vehicle make and model recognition. *In*: **Proceedings of the IEEE conference on computer vision and pattern recognition workshops**. [*S.l.*: *s.n.*], 2017. p. 1–8.

TAN, M.; LE, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. **CoRR**, abs/1905.11946, 2019. Disponível em: http://arxiv.org/abs/1905.11946.

VARGAS, A. C. G.; PAES, A.; VASCONCELOS, C. N. Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres. *In*: SN. **Proceedings of the XXIX Conference on Graphics, Patterns and Images**. [*S.l.*], 2016. v. 1, n. 4.

YAMASHITA, R. *et al.* Convolutional neural networks: an overview and application in radiology. **Insights into Imaging**, v. 9, 2018. Disponível em: https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9.

YANG, L. *et al.* **CompCars Dataset**. [*S.l.*]: GitHub, 2015. http://mmlab.ie.cuhk.edu.hk/datasets/comp_cars/index.html.

YANG, L. *et al.* A large-scale car dataset for fine-grained categorization and verification. *In*: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [*S.l.*: *s.n.*], 2015.

ZHANG, B. *et al.* Alphamex: A smarter global pooling method for convolutional neural networks. **Neurocomputing**, v. 321, p. 36–48, 2018. ISSN 0925-2312. Disponível em: https://www.sciencedirect.com/science/article/pii/S0925231218310610.

ZHANG, Y. *et al.* Real-time vehicle detection based on improved yolov5. **Sustainability**, v. 14, n. 19, 2022. ISSN 2071-1050. Disponível em: https://www.mdpi.com/2071-1050/14/19/12274.

ZHAO, J. *et al.* A wheat spike detection method in uav images based on improved yolov5. **Remote Sensing**, v. 13, n. 16, 2021. ISSN 2072-4292. Disponível em: https://www.mdpi.com/2072-4292/13/16/3095.