UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

Talles Nascimento Rodrigues

Aprendizado profundo na identificação de objetos em veículos autônomos

São Carlos

2021

Talles Nascimento Rodrigues

Aprendizado profundo na identificação de objetos em veículos autônomos

Trabalho de conclusão de curso apresentado ao Centro de Ciências Matemáticas Aplicadas à Indústria do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, como parte dos requisitos para conclusão do MBA em Ciências de Dados.

Área de concentração: Ciências de Dados

Orientador: Prof. Dr. Afonso Paiva Neto

São Carlos

2021

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi, ICMC/USP, com os dados fornecidos pelo(a) autor(a)

Rodrigues, Talles Nascimento
S856m Aprendizado profundo na identificação de objetos em veículos autônomos / Talles Nascimento Rodrigues ; orientadora Gleice da Silva Castro Perdoná. – São Carlos, 2021.
54 p. : il. (algumas color.) ; 30 cm.

Monografia (MBA em Ciências de Dados) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2021.

1. Ciência de Dados. 2. Bioinformática. 3. Single-cell RNA-seq. 4. Classificação celular. I. Perdoná, Gleice da Silva Castro, orient. II. Título.

Talles Nascimento Rodrigues

Aprendizado profundo na identificação de objetos em veículos autônomos

Trabalho de conclusão de curso apresentado ao Centro de Ciências Matemáticas Aplicadas à Indústria do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, como parte dos requisitos para conclusão do MBA em Ciências de Dados.

Data de defesa: 22 de janeiro de 2022

Comissão Julgadora:

Prof. Dr. Afonso Paiva Neto
Orientador

Professor
Convidado1

Professor
Convidado2

São Carlos
2021

RESUMO

Rodrigues, T. N.. **Aprendizado profundo na identificação de objetos em veículos autônomos**. 2021. 54p. Monografia (MBA em Ciências de Dados) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2021.

A detecção de objetos em sistemas de veículos autônomos é crucial para o planejamento de rota e ações dos veículos. No entanto, a detecção de objetos é um dos componentes de software com maior recorrência de falhas e podem desencadear condições de risco à segurança de condutores, passageiros e ao trânsito. Aprendizado profundo tem sido empregado cada vez mais em visão computacional e tem se mostrado capaz de localizar e categorizar objetos em tempo real. Este trabalho utilizou um dos algoritmos de detecção de objetos mais recentes, Yolov5, em uma base de dados da pública da Roboflow pela primeira vez e avaliou como a variação da quantidade de épocas, função de custo e otimizador impacta na eficiência de detecção.

Palavras-chave: detecção, segurança, aprendizado profundo, Yolov5.

ABSTRACT

Rodrigues, T. N.. **Object identification in self-driving cars with Deep learning**. 2021. 54p. Monografia (MBA em Ciências de Dados) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2021.

Object detection in autonomous vehicle systems is crucial for route planning and vehicle control. However, object detection is a software component with signifcant recurrence of failures and can lead to risky conditions for the traffic's safety. Deep learning has been being frequently used in computer vision application and it has proven to be able to locate and categorize objects in real time. This research employed one of the most recent object detection algorithms, Yolov5, in a public database provided by Roboflow for the first time. It also evaluated how the variation in the number ofepochs, cost function and optimizer impacts the detection efficiency.

Keywords:detectoin, safety, deep learnig, Yolov5

SUMÁRIO

1 INTRODUÇÃO

Veículos autônomos têm sido amplamente estudados nos últimos anos devido aos benefícios em potencial na redução de acidentes e melhor fluxo de automóveis no trânsito (TAKUMI et al., 2017). Munir et al. (2018) define os principais questionamentos de veículos autônomos como i) Onde estou ii) O que está ao meu redor iii) O que acontecerá em seguida e iv) O que eu devo fazer. A descrição do ambiente iii) é resultado da detecção de objetos tais como estrada, pessoas, carros, animais, etc.

Takumi et al. (2017) também afirma que a implementação de veículos autônomos se dá quando um sistema computadorizado percebe o ambiente e controla o automóvel baseado nessas informações. No entanto, como descrito por GARCIA et al. (2020), os sistemas de software de veículos autônomos são suscetíveis à erros ou falhas e podem desencadear condições críticas de segurança para os passageiros e o ambiente ao redor.

Detecção de objetos é um dos componentes com maior recorrência de falhas em sistemas de veículos autônomos e influenciam significativamente o planejamento de rota e ação do automóvel, como descrito por GARCIA et al. (2020).

Aprendizado Profundo (do inglês, Deep Learning) aplicado em algoritmos de Aprendizado de máquina (do inglês, Machine Learning) tem a capacidade de detectar objetos em tempo real (REN et al., 2016). Dessa forma, estudar métodos de Aprendizagem Profunda tem potencial de alavancar os sistemas de percepção de ambiente em veículos autônomos.

Uma das principais motivações para os avanços em direção autônoma é a não vulnerabilidade da tecnologia de detecção e controle a erros comuns aos humanos tais como distração, fadiga e direção emotiva; fatores frequentes em acidentes de trânsito (BRUMMELEN et al., 2018).

As pesquisas em algoritmos de detecção de objetos em aprendizado profundo baseados inteiramente em imagens têm seguido duas linhas. A primeira em detecção de objetos em dois estágios, focado em precisão e a segunda, em único estágio, focado em tempo de resposta.

1.1 Objetivos gerais e específicos

A detecção de objetos em sistemas de veículos autônomos é essencial para o planejamento de rota, bem como prevenção de acidentes. O principal objetivo desta pesquisa é aplicar algum método de Aprendizado Profundo para identificar e caracterizar objetos capturados por sistemas de percepção de veículos autônomos.

Os objetivos específicos desta monografia são:

1. estudar os modelos de detecção e classificação de objetos mais empregados em visão computacional, com enfoque em veículos autônomos;
2. aplicar uma das metodologias em um conjunto de dados real;
3. avaliar como variações nos parâmetros ou camadas da arquitetura afetam a detecção e classificação de objetos.

1.2 Organização da monografia

Este trabalho está dividido em 5 capítulos. Após a introdução, no Capítulo 2, os componentes básicos que compõe os veículos autônomos são apresentados, bem como tecnologias de sensoriamento e detecção de objetos.

No Capítulo 3, alguns conceitos necessários para melhor entendimento da abordagem deste trabalho e da arquitetura utilizada são apresentados.

A seguir, no Capítulo 4, os resultados obtidos para os experimentos elaborados são exibidos junto com uma breve discussão dos fatores que influenciaram o desempenho do algoritmo.

Por fim, no Capítulo 5, as considerações finais sobre o trabalho são apresentadas e algumas sugestões feitas para trabalhos futuros derivados a partir das metodologias empregadas.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo descreverá as principais características de veículos autônomos (VA), com destaque ao sistema de detecção e classificação de objetos que compreende uma combinação entre hardware e software. Os componentes de hardware são responsáveis em levantar dados do ambiente por meio de sensores como, por exemplo, LiDAR's e câmeras. Por outro lado, o software é responsável por processar as saídas oriundas dos sensores, interpretá-los e convertê-los em informação útil que será utilizada na navegação do automóvel. Aprendizado profundo tem avançado as metodologias de compreensão dos dados, mas ainda possui desafios a serem explorados.

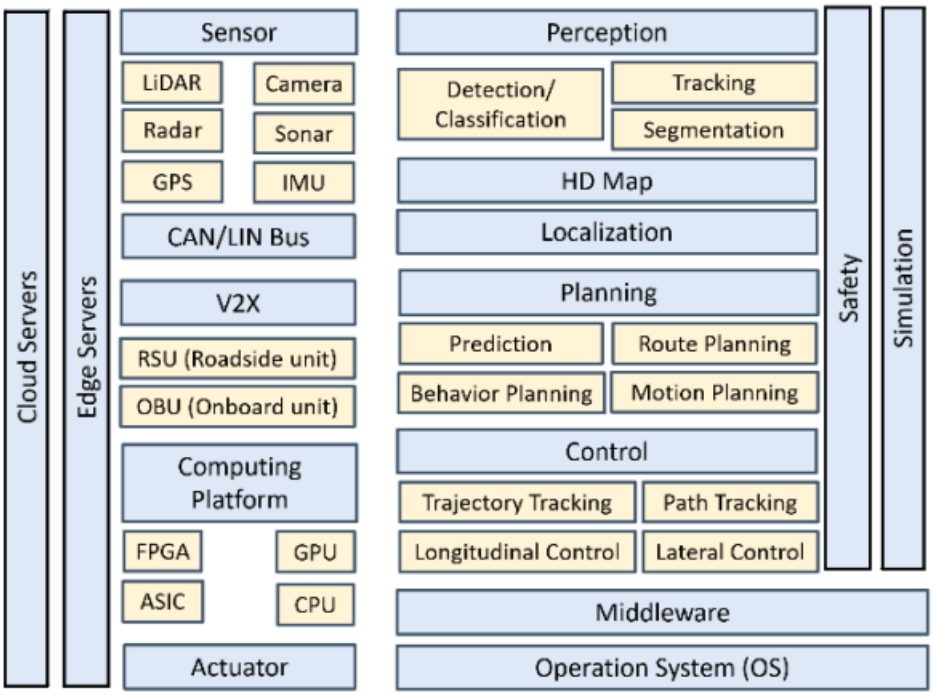
2.1 Veículos autônomos

Veículos autônomos transformarão o mercado de transporte ao melhorar a qualidade de vida, segurança das estradas e eficiência de uso de combustível (WISEMAN, 2021).Eles consistem em software e hardware trabalhando em conjunto para alcançar a automação de condução/direção no mundo físico (GARCIA et al., 2020).Diferentes arquiteturas já foram implementadas em veículos autônomos (HUANG; CHEN, 2020). Os elementos básicos de um veículo autônomo são ilustrados na Figura 1.

De acordo com Huang e Chen (2020), os componentes de hardware são controle de aceleração, freios e direção; auto sensoriamento, localização e sensoriamento de ambiente; plataforma de computação para processamento de dados. Em software: percepção; mapeamento; localização; predição de eventos; planejamento de rota; planejamento de movimento e controle. O sistema de percepção coleta informações de sensores e descobre conhecimento relevante do ambiente. Ele desenvolve uma compreensão contextual do ambiente de condução, como detecção, rastreamento e segmentação de obstáculos, sinais de trânsito/marcação e áreas dirigíveis de espaço livre. Com base nos sensores implementados, a tarefa de percepção do ambiente pode ser realizada usando LIDARs, câmeras, radares ou uma fusão entre esses três tipos de dispositivos (HUANG; CHEN, 2020).

Veículos autônomos podem ter diferentes níveis de automação que vão desde à assistência do condutor até a condução totalmente autônoma (MUHAMMAD et al., 2020). A Figura 2 ilustra os graus de automação veicular. O primeiro depende exclusivamente do controle do motorista, que ainda pode ser auxiliado por alarmes como, por exemplo, sensor de proximidade de ré. O segundo conta com a automação de funções individuais tais como um controlador automático de velocidade. No terceiro nível, o condutor não precisa usar os pés e mãos, mas ainda deve estar atento ao ambiente. Já o quarto grau não exige atenção ou ação, mas requer intervenção em ocasiões de perigo potencial. No penúltimo estágio, o motorista não é responsável pelo comportamento do veículo e não

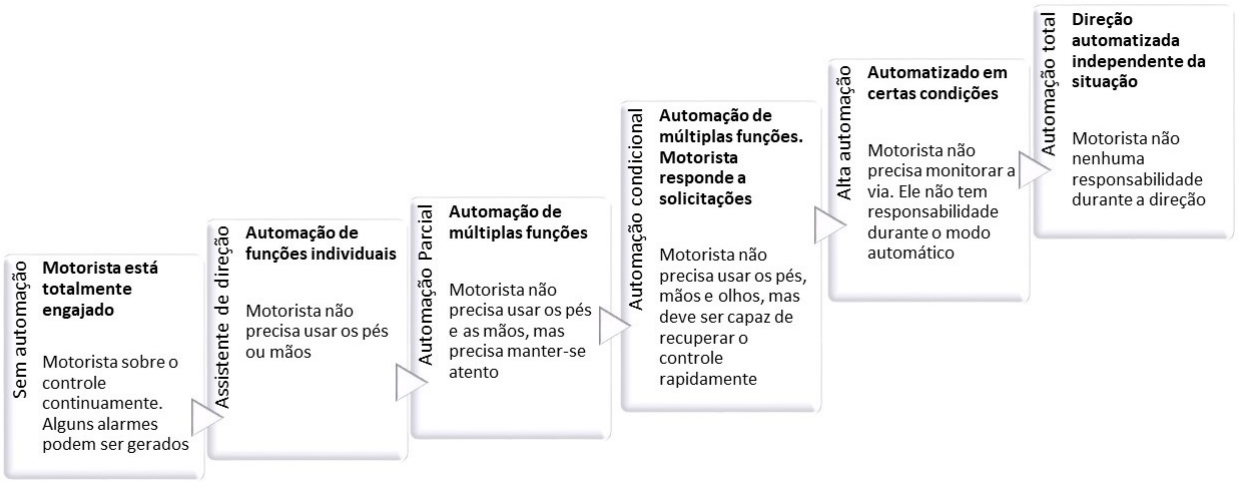
Figura 1 – Arquitetura - Veículo Autônomo



Fonte: Huang e Chen (2020)

necessita intervir, enquanto no último grau, a automação é responsável inteiramente pelo controle do veículo. É importante mencionar que o maior nível de automação ainda não foi alcançado em larga escala.

Figura 2 – Níveis de automação em veículos autônomos



Fonte: Adaptado de Muhammad et al. (2020)

Os sistemas veiculares com nível de automação inicial ou até mesmo avançado já causaram ferimentos e acidentes fatais ao longo dos anos (GARCIA et al., 2020). Dada a criticidade de segurança de tais veículos, é imperativo que os sistemas de detecção de objetos tenham mínimo erro e tempo de resposta. O sistema de detecção é um dos componentes de software mais importantes no sistema de navegação do veículo (BRUMMELEN et al., 2018). A percepção do ambiente e a aplicação de inteligência computacional (algoritmo de detecção) são essenciais ao sistema de detecção.

A seguir, as principais propriedades do sistema de percepção serão descritas, seguidas pelos algoritmos de detecção e objetos.

2.1.1 Sistema de percepção

O sistema percepção de veículos autônomos usa sensores para escanear e monitorar continuamente o ambiente, semelhante ao sensoriamento humano (visão, audição, etc) (MAURER et al., 2016). As principais tecnologias de sensoriamento em veículos autônomos podem incluir radares, sonares, LiDARs e câmeras (FENG et al., 2020).

Feng et al. (2020) e Brummelen et al. (2018) descrevem os tipos de sensoriamento. *Radares*: emitem ondas de rádio que são refletidas por um obstáculo, mede o tempo de execução do sinal e estima a velocidade radial do objeto pelo efeito Doppler. Eles são robustos contra várias condições de iluminação e clima, mas classificar objetos (pedestres, objetos estáticos por meio de radares é muito desafiador devido à sua baixa resolução. *Sonares*: sensores ultrassônicos enviam ondas sonoras de alta frequência para medir a distância até os objetos. Eles são normalmente aplicados para detecção de objetos próximos e em cenários de baixa velocidade, como sistemas de assistência ao estacionamento.No entanto, são deficientes na detecção de objetos a mais de 2 m de distância. *LiDARs*: fornecem informações de profundidade precisas dos arredores na forma de pontos 3D. Eles medem os reflexos de feixes de laser que emitem com uma certa frequência. Os LiDARs são robustos para diferentes condições de iluminação e menos afetados por variações condições climáticas, como neblina e chuva, do que as câmeras visuais. No entanto, LiDARs típicos são inferiores às câmeras para classificação de objetos, uma vez que não podem capturar as texturas finas de objetos e seus pontos tornam-se esparsos com objetos distantes, apresentando má medição a distâncias menores que 2 metros.*Câmeras*: imagens capturadas por câmeras visuais e térmicas podem fornecer informações detalhadas sobre os arredores de um veículo. São boas para detectar e classificar obstáculos, assim como detecção de pista (BRUMMELEN et al., 2018). No entanto, câmeras são sensíveis à luz e condições adversas do ambiente (por exemplo: chuva), além de não medirem distâncias (FENG et al., 2020) e ainda serem computacionalmente mais custosas comparadas a sensores ativos.

Atualmente, devido às limitações e aos altos custos dos sensores disponíveis, a maioria dos veículos comerciais inclui apenas nível de autonomia que requer atenção. Os

recursos autônomos nesses veículos geralmente consistem em frenagem de emergência, detecção de ponto cego e/ou manutenção de faixa (BRUMMELEN et al., 2018). Dessa forma, a ênfase deste trabalho será na detecção e classificação de objetos a partir de imagens geradas pelos sistema de percepção de veículos autônomos.

2.1.2 Algoritmos de detecção de objetos

A detecção de objetos é a tarefa de reconhecer e localizar vários objetos em um cenário. Os objetos são geralmente reconhecidos ao estimar-se uma probabilidade de classificação e localizados em caixas delimitadoras (FENG et al., 2020).

Uma rede neural profunda é uma ferramenta computacional expressiva para aprender representações hierárquicas de características, dada uma grande quantidade de dados (FENG et al., 2020). Neste sentido, métodos que empregam aprendizado profundo têm sido estudados para a compreensão de cena em direção autônoma.

Os métodos de detecção de objetos por aprendizado profundo seguem uma de duas abordagens: os pipelines de detecção de objetos de dois estágios ou um estágio (JANAI et al., 2020). O primeiro apresenta maior precisão na detecção de objetos, mas tempo de resposta insuficiente para aplicações em tempo reais. O segundo, exibe tempo de resposta satisfatória em tempo real, mas com deficiência em precisão (JANAI et al., 2020). Abaixo, os diferentes métodos e trabalhos relacionados dessas duas abordagens são apresentados.

Duplo estágio

Girshick et al. (2014) propõe R-CNNs para resolver o problema de localização de objetos por meio de um paradigma de "reconhecimento usando regiões". Eles geram muitas propostas de região usando busca seletiva, extraem um vetor de características para cada proposta usando uma CNN e classificam cada região com um SVM linear. Hosang et al. (2015) trabalhou na detecção de pedestres aplicando a arquitetura R-CNN com pre-treinamento em base de dados. Os resultados foram promissores, mas notou-se a necessidade do aprofundamento na precisão do modelo

Girshick (2015) melhorou a qualidade de detecção de objetos da R-CNN ao propor a Fast RCNN, um algoritmo de treinamento de estágio único usando uma perda multitarefa que, em conjunto, aprende a classificar propostas de objetos e refinar suas localizações espaciais. Qian et al. (2016) criou um sistema de detecção de sinais de tráfego em estradas com um método de proposta de região híbrida que leva em consideração as informações complementares de cor e borda; e um Fast R-CNN para realizar a classificação e a regressão da caixa delimitadora simultaneamente. No entanto, este método se mostrou sensível às condições de iluminação, cor e variação de escalas de objetos.

Ren et al. (2016) introduziu as Redes de Proposta de Região (RPN), que compartilham características convolucionais de imagem completa com a rede de detecção e,

portanto, não incorrem em custos computacionais adicionais na arquitetura chamada de Faster-RCNN. Fan, Brown e Smith (2016) aplicou o método Faster R-CNN para investigar quantas regiões propostas resultam em melhor precisão na detecção de objetos em veículos autônomos. Prabhakar et al. (2017) também utilizou o método e encontrou resultado satisfatório na classificação e detecção de carros e outros automóveis. O método também se mostrou eficiente na detecção de animais, pessoas e de automóveis em condições de ambientes desfavoráveis (por exemplo: chuvas). Entretanto, o método não classificou corretamente os objetos não presentes na base de treinamento.

Único estágio

Liu et al. (2016) desenvolveu o SSD (Single Shot Detector) que extrai mapas de recursos de diferentes escalas para detecção e adotou caixas de âncora de diferentes escalas e proporções em uma camada convolucional para detecção de objetos diretamente.Garnett et al. (2017) empregou o SSD para detecção de objetos categorizados e estimativa de pose, com o StixelNet para obstáculos em geral. O método apresentou precisão inferior quando comparado a algoritmos de detecção de dois estágios.

Lin et al. (2017b) apresentou o RetinaNet, uma rede única e unificada composta de uma rede de backbone e duas sub-redes de tarefas específicas: a primeira de classificação, a segunda a caixa delimitadora. Ale, Zhang e Li (2018) aplicou a ferramenta no problema de detecção de estradas danificadas. Outro autor, Pei et al. (2020), aplicou o RetinaNet na detecção de pedestres. O resultado foi tempo de resposta inferior comparado aos de dois estágios, porém menor precisão na detecção.

Redmon et al. (2016) sugeriu aprender simultaneamente o conjunto caixas delimitadoras espacialmente separadas e as probabilidades de classe a partir dos mapas de recursos superiores, dando a este método o nome de YoLo - You only look once. Sarda, Dixit e Bhan (2021) utilizou o YoLo na detecção de objetos em veículos autônomos. O modelo foi capaz de detectar vários objetos que apareciam na estrada, criando uma caixa delimitadora e uma legenda ao redor do objeto. No entanto, houveram falsos positivos e falsos negativos na classificação e detecção. Isso pode causar acidentes fatais ou ineficiência no plano de navegação traçado a partir dos objetos identificados. Outro fator negativo foi detecção e classificação deficiente de objetos.

2.1.3 Desafios na detecção de objetos em veículos autônomos

As estradas para carros são ambientes dinâmicos, a iluminação e o fundo nos ambientes variam, as dimensões e posições dos veículos na imagem são diversas e a forma, o tamanho e a cor do veículo são diferentes. (WANG et al., 2019). A detecção em áreas urbanas é difícil devido à grande variedade de aparências e oclusões de objetos causadas por outros objetos ou pelo próprio objeto de interesse. Além disso, a semelhança dos objetos entre si ou com o fundo e efeitos físicos como sombras projetadas ou reflexos podem

dificultar a detecção de objetos (JANAI et al., 2020).

Por outro lado, as técnicas de aprendizado profundo para veículos autônomos são treinadas em dados coletados em determinado ambiente e não são confiáveis em condições meteorológicas adversas, enquanto que sistema de detecção de objetos deveria ser robusto o suficiente para se adaptar ao ambiente circundante (MUHAMMAD et al., 2020).

A detecção de objetos se torna um desafio quando direção urbana complexa (tráfego intenso, cruzamentos, etc.), detecção de obstáculos - especialmente detecção de pedestres e ciclistas e direção autônoma sem informações a priori, condições de tempo e iluminação ruins (BRUMMELEN et al., 2018).

A aplicação de inteligência computacional nos dados gerados pelos sensores é capaz de caracterizar, classificar ou identificar objetos. Em um cenário ideal, a combinação entre sistema de sensoriamento e processamento de dados deveria identificar pessoas, automóveis, ciclistas, placas de sinalização, condições de pista. Dessa forma, o melhor plano de navegação poderia ser traçado e executado. Na prática, o desafio é estabelecer um balanço entre precisão (informação correta sobre o ambiente), tempo de resposta (rápido entendimento do cenário e decisão do que fazer) e robustez (em condições adversas ou não cobertas durante o treinamento) (FENG et al., 2020). Esse trabalho estará centrado na avaliação de um modelo, e possíveis variações, de detecção de objeto de único estágio em torno desses pilares (precisão, velocidade, robustez) aplicado a uma base de dados pública.

3 CONCEITOS BÁSICOS

Este capítulo é dedicado a descrever os componentes básicos da arquitetura do algoritmo Yolo.

3.1 CONCEITOS BÁSICOS

3.1.1 Rede Neural Artificial

Um neurônio artificial é um componente básico das redes neurais artificiais e seu design e funcionalidade são derivados de um neurônio biológico. No caso de um neurônio biológico, a informação chega pelos dendritos, é processada no soma e passado adiante pelo axônios. No neurônio artificial, a informação chega por meio de entradas que são ponderadas (cada entrada pode ser multiplicada individualmente por um peso). O corpo de um neurônio artificial então soma as entradas ponderadas, polariza e "processa" a soma com uma função de transferência. No final, a informação processada é transmitida pela(s) saída(s).

Matematicamente, um neurônio é descrito pela 4.1.

$$y(k) = F(\sum_{i=0}^{m} w_i(k) \cdot x_i(k) + b) y_r \log(y_p) + (1 - y_r) \log(1 - y_p)) \tag{3.1}$$

Onde,

$x_i(k)$ = valor da entrada em um tempo discreto k, tendo i variando de 0 a m

$w_i(k)$ = peso em um tempo discreto k, tendo i variando de 0 a m

b = viés

F = função de transferência

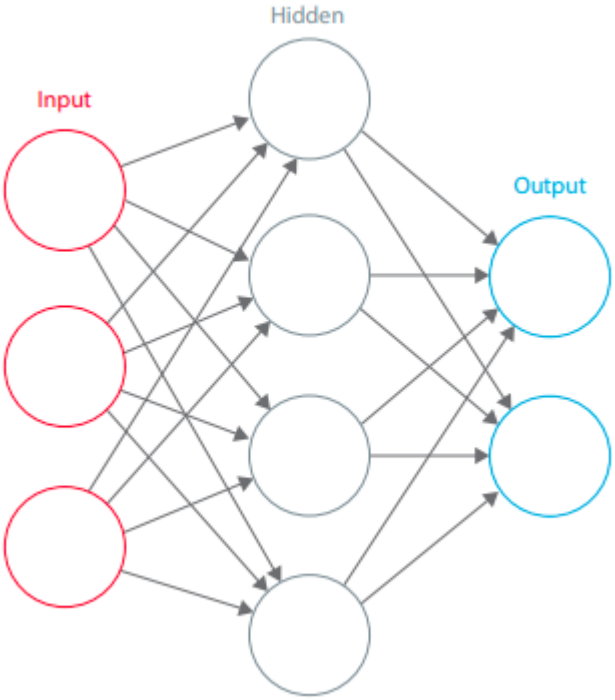
$y_i(k)$ = saída em um valor discreto k

Uma rede neural é um sistema de neurônios interconectados que trocam mensagens entre si (HIJAZI et al., 2015). De acordo com Hijazi et al. (2015), as conexões possuem um peso numérico que é ajustado durante o treinamento a fim de possibilitar o reconhecimento de imagens. Ainda, a rede consiste de múltiplas camadas, cada uma com diversos neurônios de detecção de caraterísticas (HIJAZI et al., 2015).

Um estrutura básica de uma rede neural artificial é mostrada na Figura 6. A entrada geralmente é na forma de um vetor multidimensional que é distribuído para camadas

ocultas. As camadas ocultas, então, decidirão a partir da camada anterior e avaliarão como uma mudança estocástica em si mesma prejudica ou melhora o resultado final (aprendizagem). O empilhamento de diversas camadas ocultas é chamado de aprendizado profundo, do inglês Deep Learning (O'SHEA; NASH, 2015).

Figura 3 – Redes Neurais Artificiais



Fonte: Hijazi et al. (2015)

Cada conexão é associada a um peso numérico e a saída de cada neurônio na camada oculta é o resultado da aplicação de uma função chamada de ativação. Segundo Wang (2003), o objetivo da função de ativação é, além de introduzir a não linearidade na rede neural, limitar o valor do neurônio para

3.1.1.1 Algoritmos de otimização

Os algoritmos de otimização são métodos para ajustar os atributos de uma rede neural, tais como pesos e taxa de aprendizado, a fim de reduzir os erros, ou ainda, o custo do seu modelo.

O Gradiente Descendente Estocástico (SGD) é exemplo de algoritmo de otimização que busca minimizar a função de perda. Os parâmetros do modelo são alterados após o cálculo da perda em cada exemplo (RUDER, 2016). O otimizador que estima o momento adaptativo é chamado de Adam. A ideia é reduzir a velocidade de convergência e ser mais cuidadoso com mínimos locais. Além de armazenar uma média exponencialmente

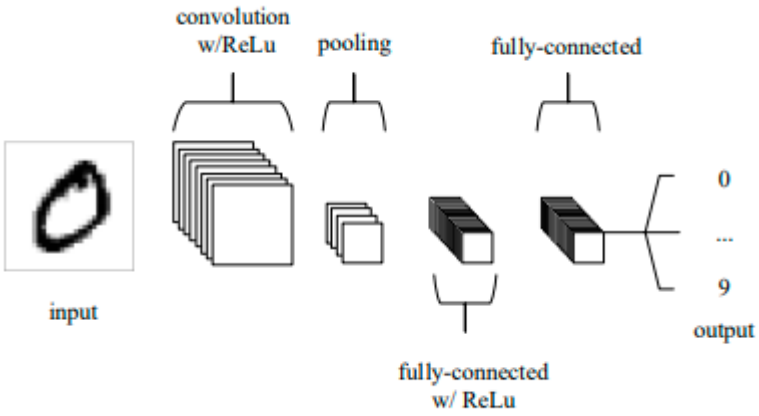
decadente de gradientes passados ao quadrado, o Adam também mantém uma média exponencialmente decadente dos gradientes passados(KINGMA; BA, 2014).

3.1.2 CNN - Convolutional Neural Networks

CNN, do inglês *Convolutional Neural Networks*, redes neurais convolucionais, tem papel fundamental na detecção de padrões, seja no processamento de imagens ou reconhecimento de voz (ALBAWI; MOHAMMED; AL-ZAWI, 2017).

As quatro camadas básicas de uma CNN são i. camada de entrada, ii. convolução, iii. agrupamento/redução e iv. camadas totalmente conectadas. As empilhar esses três elementos , uma arquitetura CNN é formada (O'SHEA; NASH, 2015).

Figura 4 – Arquitetura básica CNN



Fonte: O'Shea e Nash (2015)

O'Shea e Nash (2015) e Hijazi et al. (2015) descrevem as quatro camadas da seguinte forma:

1. *Camada de entrada*: vetor multidimensional. Em caso de tratamento de imagens, os valores dos pixeis da imagem;

2. *Camadas de convolução*: extraem diferentes características da entrada.A camada convolucional determinará a saída dos neurônios conectados à entrada pelo cálculo do produto escalar entre seus pesos e a região conectada ao volume de entrada. A unidade linear retificada (comumente abreviada para ReLu) visa aplicar uma função de ativação "elemento a elemento"para a saída da ativação produzida pela camada anterior. A primeira camada de convolução extrai aspectos de baixo nível, como bordas, linhas e cantos. As camadas de maior nível extraem propriedades de nível superior.

3. *Camada de agrupamento/redução*: executa a redução da resolução ao longo da dimensionalidade espacial da entrada fornecida, reduzindo ainda mais o número de parâmetros.

4. *Redes totalmente conectadas*: essas camadas somam matematicamente uma ponderação da camada anterior de característica, produzindo pontuação de classes para futuras classificações.

4 METODOLOGIA

Este capítulo é dedicado a descrever os métodos empregados para detectar objetos capturados por sistema de percepção de veículos autônomos, disponíveis em uma base de dados pública. O trabalho utilizará um algoritmo de detecção em único estágio (Yolov5) e avaliará como variações em funções internas, parâmetros ou arquitetura influenciam a detecção.

Inicialmente esta seção traz uma visão geral sobre o algoritmo Yolo e o Yolo versão 5. Em seguida, são apresentados algumas ferramentas computacionais que auxiliarão o desenvolvimento do trabalho, bem como algumas técnicas de pré-processamento de dados. Por fim, é apresentado como o algoritmo é treinado para uma base de dados específica.

4.1 CONCEITOS BÁSICOS

4.1.1 Aprendizado de máquina

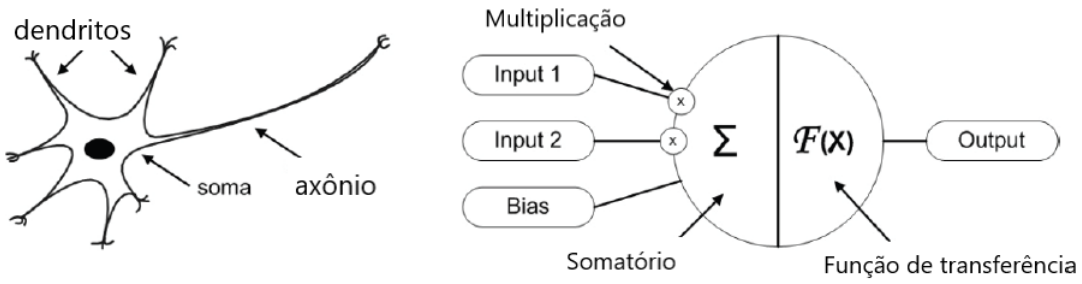
Aprendizado de máquina (do inglês Machine Learning) é a utilização de técnicas e procedimentos computacionais automáticas baseado em operações lógicas ou binárias que aprende uma tarefa a partir de uma série de exemplo (ZHANG, 2010).

Existem duas formas de um algoritmo de aprendizado de máquina adquirir conhecimento. A primeira é supervisionada, ou seja, uma função é inferida a partir de um conjunto de dados rotulados. Neste método, o conjunto de dados é dividido entre treinamento e teste, sendo o primeiro usado para ajustar o modelo e o segundo, para verificar a eficácia. Por outro lado, a processo de adquirir conhecimento a partir de dados não rotulados é conhecido como aprendizado não supervisionado (SHARMA; KUMAR, 2017).

4.1.2 Rede Neural Artificial

Um neurônio artificial é um componente básico das redes neurais artificiais e seu design e funcionalidade são derivados de um neurônio biológico. No caso de um neurônio biológico, a informação chega pelos dendritos, é processada no soma e passado adiante pelo axônios. No neurônio artificial, a informação chega por meio de entradas que são ponderadas (cada entrada pode ser multiplicada individualmente por um peso). O corpo de um neurônio artificial então soma as entradas ponderadas, polariza e processa a soma com uma função de transferência. No final, a informação processada é transmitida pela(s) saída(s) (output) (SUZUKI, 2011). A Figura 5 traz uma representação visual dos dois tipos de neurônios.

Figura 5 – Neurônio biológico (esquerda) e artificial (direita)



Fonte: adaptado de (SUZUKI, 2011)

Matematicamente, um neurônio artificial é descrito pela Equação 4.1.

$$y(k) = F(\sum_{i=0}^{m} w_i(k) \cdot x_i(k) + b) \tag{4.1}$$

Onde,

$x_i(k)$ = valor da entrada em um tempo discreto k, tendo i variando de 0 a m

$w_i(k)$ = peso em um tempo discreto k, tendo i variando de 0 a m

b = viés (bias)

F = função de transferência

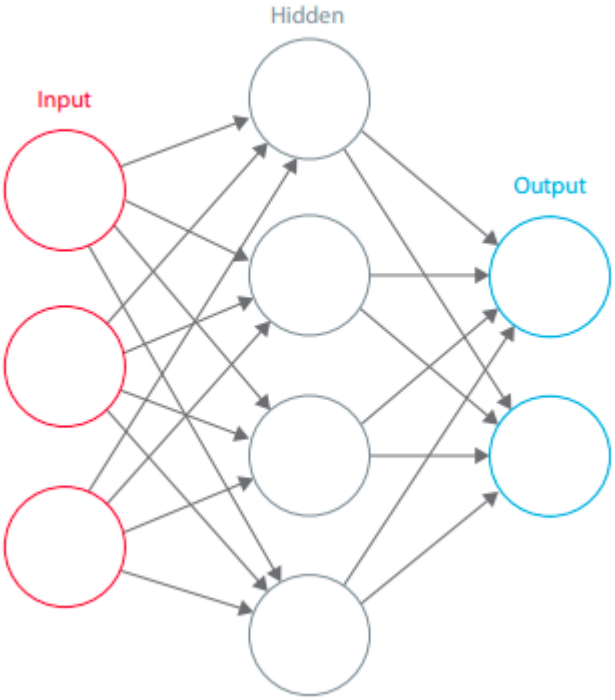
$y_i(k)$ = saída em um valor discreto k

Uma rede neural é um sistema de neurônios interconectados que trocam mensagens entre si (HIJAZI et al., 2015). De acordo com Hijazi et al. (2015), as conexões possuem um peso numérico que é ajustado durante o treinamento a fim de possibilitar o reconhecimento de imagens. Ainda, a rede consiste de múltiplas camadas, cada uma com diversos neurônios de detecção de caraterísticas (HIJAZI et al., 2015).

Um estrutura básica de uma rede neural artificial é mostrada na Figura 6. A entrada geralmente é na forma de um vetor multidimensional que é distribuído para camadas ocultas. As camadas ocultas, então, decidirão a partir da camada anterior e avaliarão como uma mudança estocástica em si mesma prejudica ou melhora o resultado final (aprendizagem). O empilhamento de diversas camadas ocultas é chamado de aprendizado profundo, do inglês Deep Learning (O'SHEA; NASH, 2015).

Cada conexão é associada a um peso numérico e a saída de cada neurônio na camada oculta é o resultado da aplicação de uma função chamada de ativação. Segundo Wang (2003), o objetivo da função de ativação é, além de introduzir a não linearidade na

Figura 6 – Redes Neurais Artificiais



Fonte: Hijazi et al. (2015)

rede neural, limitar o valor do neurônio para que a rede neural não seja paralisada por neurônios divergentes.

4.1.2.1 Função de ativação

Sharma e Sharma (2017) lista as principais funções de ativação como Função de Passo Binário (binary step function), Linear, Sigmoid, Tangente, ReLu, Leaky ReLU, ReLu parametrizado, Unidade exponencial linear, Swish, SoftMax. A seguir, as funções Sigmoid e Leaky ReLu serão descritas, dado sua importância para o algoritmo Yolov5.

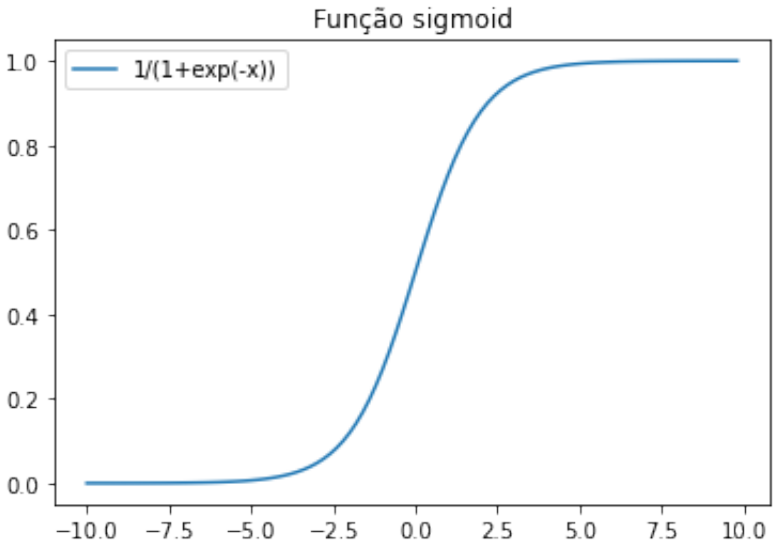
Sigmoid: é a função de ativação mais utilizada por ser não linear. A função Sigmoid transforma os valores de entrada para uma faixa entre 0 a 1 e é definida como:

$$f(x) = 1/(1 + e^{-x}) \tag{4.2}$$

Além disso, a função Sigmoid não é simétrica em relação a zero, o que significa que os sinais de todos os valores de saída dos neurônios serão os mesmos. A Figura 7 ilustra a função.

Leaky ReLu: é uma versão da função ReLU na qual os valores negativos de x são definidos como um componente linear extremamente pequeno de x ao invés de 0.

Figura 7 – Função Sigmoid



Fonte: o autor (2021)

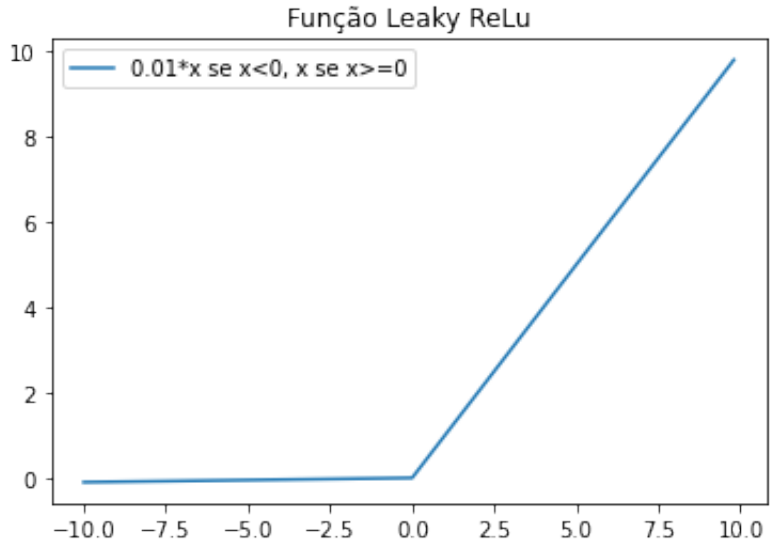
Matematicamente é expresso como:

$$f(x) = 0.01 \cdot x, x < 0$$

$$f(x) = x, x >= 0$$

A função Leaky ReLu é exemplificada na Figura 8.

Figura 8 – Leaky ReLU



Fonte: o autor (2021)

4.1.2.2 Função de perda ou de custo

Uma função de perda (loss function) ou custo é outro conceito fundamental no contexto de redes neurais e representa o preço a pagar por uma predição quando comparada à um valor real (ROSASCO et al., 2004). Em geral, quando menor o custo a se pagar por uma predição, melhor o método de predição (STEINWART, 2007). Um exemplo comum de função de perda é a entropia cruzada (Cross Entropy) definida na Equação 4.3.

$$CrossEntropyLoss = -(y_r \cdot \log(y_p) + (1 - y_r) \cdot \log(1 - y_p)) \tag{4.3}$$

Onde:

r = valor real

p = valor predito

Nesta função, o custo se torna mais alto quando as predições divergem dos valores reais.

A segunda função de custo mencionada neste trabalho á Entropia Cruzada Binária com perda de Logits - Binary Cross-Entropy with Logits Loss (BCEWithLogitsLoss) definida na Equação 4.4. Esta adiciona um peso às classes e tem custo elevado para classificações erradas.

$$l(x, y) = \begin{cases} soma(L), & \text{se } redução = sum \\ média(L), & \text{se } redução = mean \\ L = (l_1, ..., l_N)^T, & \text{sendo } l_n = -w_n[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n))] \end{cases} \tag{4.4}$$

Onde,

N = tamanho do lote

w_n= peso

y_n= saída ou valor predito

x_n = entrada ou valor real

A última função de perda descrita é a Função de Perda Focal (Focal Loss). A Perda Focal é projetada para lidar com o cenário de detecção de objeto no qual há um desequilíbrio extremo entre as classes de primeiro e segundo plano durante o treinamento.

O fator de perda total adiciona o elemento $(1-pt)^{\gamma}$ ao critério de entropia cruzada padrão. Ao fazer $\gamma > 0$, reduz-se a perda relativa de exemplos bem classificados ($pt > 0.5$), colocando mais foco em exemplos difíceis e mal classificados (LIN et al., 2017b).

$$FL(pt) = -\alpha t(1-pt)^{\gamma} \cdot \log(pt) \tag{4.5}$$

4.1.2.3 Algoritmos de otimização

Os algoritmos de otimização são métodos para ajustar os atributos de uma rede neural, tais como pesos e taxa de aprendizado, a fim de reduzir os erros, ou ainda, o custo do seu modelo.

O Gradiente Descendente Estocástico (do inglês Stocastic Gradient Descendent - SGD) é exemplo de algoritmo de otimização que busca minimizar a função de perda. Os parâmetros do modelo são alterados após o cálculo da perda em cada exemplo ou grupo de exemplos (RUDER, 2016). A atualização de um parâmetro é dada pela Equação 4.6

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla_{\theta} J[\theta; x^i, y^i)] \tag{4.6}$$

onde,

θ = parâmetro

α = taxa de aprendizado

$\nabla_{\theta} J$ = derivada da função J em relação a θ

x^i e y^i = par dos dados de treinamento

O segundo otimizador mencionado é o Adam - estimador de momento adaptativo (do inglês Adaptative Moment Estimator) . O objetivo deste otimizador é reduzir a velocidade de convergência e ser mais cuidadoso com mínimos locais. Além de armazenar uma média exponencialmente decadente de gradientes passados ao quadrado, o Adam também mantém uma média exponencialmente decadente dos gradientes passados (KINGMA; BA, 2014). Os parâmetros são atualizados de acordo com a Equação 4.7.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t^* + \epsilon}} m_t^* \tag{4.7}$$

onde,

$m_t^* = \frac{m_t}{m_t}$

$v_t^* = \frac{v_t}{1-\beta_2^t}$

m_t = média dos gradientes

v_t = gradiente passado quadrado

$\beta_1 = 0.9$

$\beta_2 = 0.999$

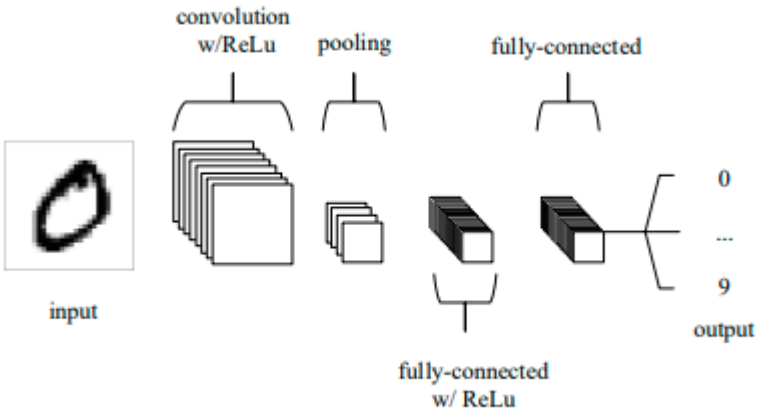
$\epsilon = \exp 10^{-8}$

4.1.3 CNN - Convolutional Neural Networks

CNN, do inglês *Convolutional Neural Networks*, redes neurais convolucionais, tem papel fundamental na detecção de padrões, seja no processamento de imagens ou reconhecimento de voz (ALBAWI; MOHAMMED; AL-ZAWI, 2017).

As quatro camadas básicas de uma CNN são i. camada de entrada, ii. convolução, iii. agrupamento/redução e iv. camadas totalmente conectadas. Ao empilhar esses três elementos, uma arquitetura CNN é formada (O'SHEA; NASH, 2015).

Figura 9 – Arquitetura básica CNN



Fonte: O'Shea e Nash (2015)

O'Shea e Nash (2015) e Hijazi et al. (2015) descrevem as quatro camadas da seguinte forma:

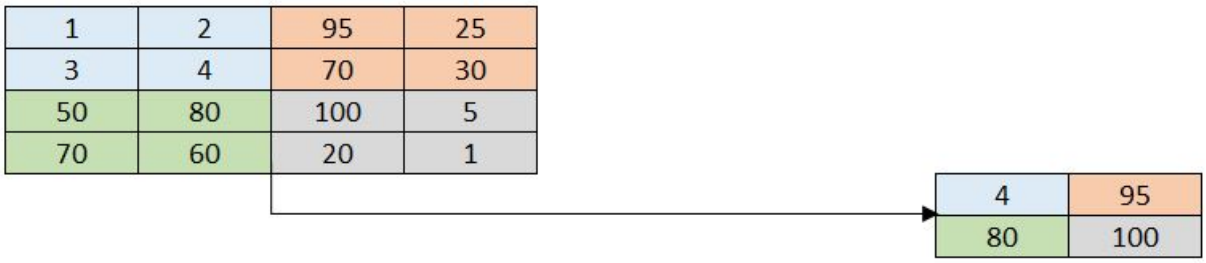
1. *Camada de entrada*: vetor multidimensional. Em caso de tratamento de imagens, os valores dos pixeis da imagem;

2. *Camadas de convolução*: são inspiradas nas operações de filtragem existentes em processamento de imagens. A camada convolucional determinará a saída dos neurônios conectados à entrada pelo cálculo do produto escalar entre seus pesos e a região

conectada ao volume de entrada (operação chamada de convolução). O valor da operação de convolução realizada em cada pixel é processada por uma função de ativação.Elas extraem diferentes características da entrada: a primeira camada os aspectos de baixo nível, como bordas, linhas e cantos, enquanto que as camadas de maior nível propriedades de nível superior.

3. *Camada de agrupamento/redução*: executa a redução da resolução ao longo da dimensionalidade espacial da entrada fornecida, reduzindo ainda mais o número de parâmetros. Essa camada geralmente é colocada depois da camada convolucional para, além de reduzir a dimensão de entrada, selecionar as características mais relevantes da saída da camada convolucional anterior. O tipo mais comum de operação nessa camada é o agrupamento por valor máximo (do inglês Max Pooling) em que é calculado o valor máximo de uma região de um mapa de características e o usa para criar um mapa com resolução reduzida (GHOLAMALINEZHAD; KHOSRAVI, 2020). A Figura 10 ilustra uma região que passou por agrupamento (max pooling).

Figura 10 – Amplificação por interpolação do vizinho mais próximo



Fonte: adaptado de (GHOLAMALINEZHAD; KHOSRAVI, 2020)

4. *Camadas totalmente conectadas*: essas camadas somam matematicamente uma ponderação da camada anterior de característica, produzindo pontuação de classes para futuras classificações.

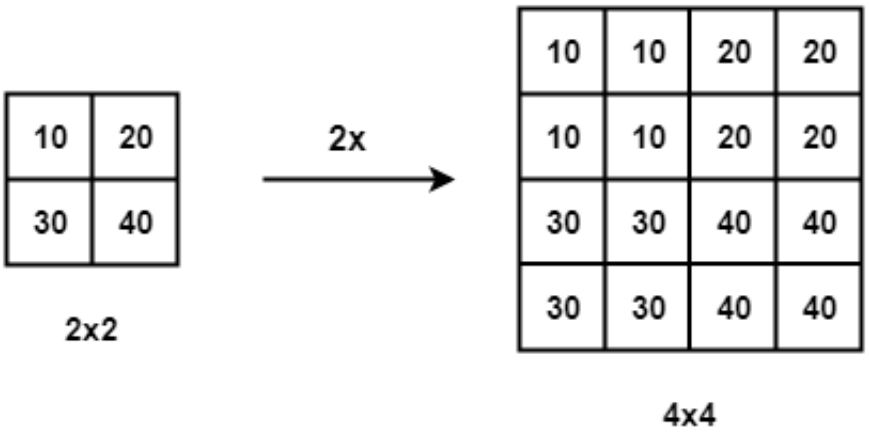
4.1.4 Amplificação

Amplificação (do inglês upsampling) é um método de amplificação de imagens que se dá pelo aumento do número de linhas e / ou colunas (dimensões) usando alguma técnica de interpolação para o preenchimento das lacunas geradas.

Um dos métodos mais utilizados é o de interpolação pelo vizinho mais próximo na qual as lacunas são preenchidas pelo valor do pixel mais próximo da imagem original

(HAN, 2013). Se ampliarmos uma imagem em 2, por exemplo, um pixel será ampliado para uma área de 2×2 com a mesma cor, assim como ilustrado na Figura 11.

Figura 11 – Amplificação por interpolação do vizinho mais próximo



Fonte: https://theailearner.com/2018/12/29/image-processing-nearest-neighbour-interpolation/

Este é o algoritmo mais rápido e o único que não insere novas cores no resultado.

4.1.5 Normalização de lotes

A normalização de lote normaliza as ativações da rede entre as camadas em lotes de modo que os lotes tenham uma média de 0 e uma variação de 1 (IOFFE; SZEGEDY, 2015). A normalização de lote normalmente é escrita como na Equação 4.8.

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} \cdot \gamma + \beta \tag{4.8}$$

A média e desvio padrão são calculados por dimensão para cada mini-lote (do inglês mini batch) e γ e β são parâmetros aprendidos a partir da entrada que podem ser usados para controlar o formato dos dados que irão para a próxima camada. No entanto, por padrão são considerados 1 e 0, respectivamente (IOFFE; SZEGEDY, 2015).

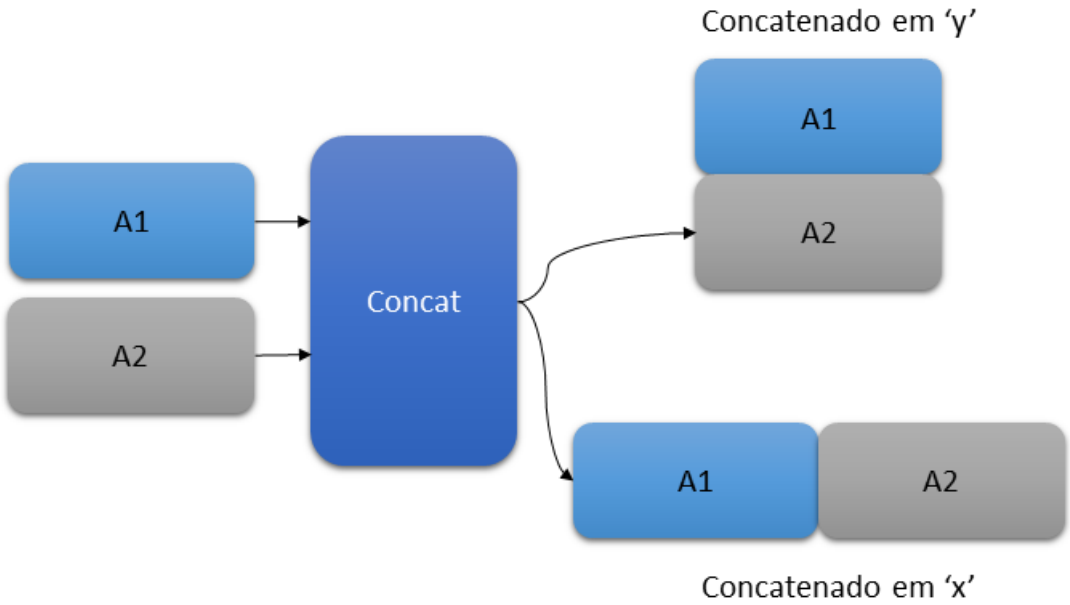
A normalização de lotes torna o treinamento mais rápido de modelos, pois reduz a mudança de covariância interna e corrige a distribuição de ativações de rede e, ainda permite a utilização de taxas de aprendizados maiores.

4.1.6 Concatenação

Operação que concatena dois tensores ao longo de uma dimensão existente. A Figura 12 ilustra dois tipos de concatenação. Em redes neurais o processo de concatenação

de tensores pode ser necessário quando a rede possui diferentes ramificações e essas ramificações precisam ser combinadas em um único tensor para ser processado pelas próximas camadas.

Figura 12 – Concatenação (concat)



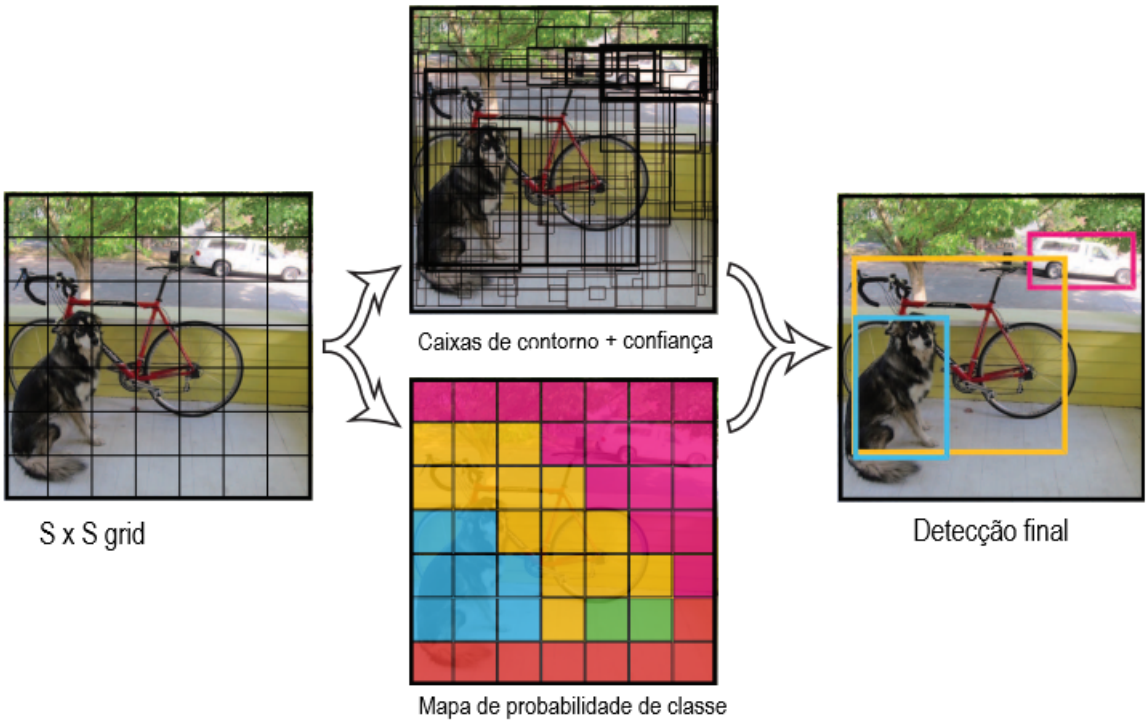
Fonte: o autor (2021)

4.2 Yolo

Yolo - do inglês *You only look once* (você apenas olha uma vez), é um modelo de detecção de objetos de único estágio, sem a existência de camadas intermediárias de detecção de regiões (WANG et al., 2019). A classificação e detecção são obtidos diretamente da imagem por uma rede convolucional (REDMON et al., 2016). O algoritmo YOLO desenvolvido por Redmon et al. (2016) divide a imagem de entrada em um grid $s \times s$. Se o centro de um objeto estiver em uma célula do grid, aquela célula é responsável por detectar o objeto. Cada célula também prediz B caixas de contorno (do inglês *bounding boxes*) e a pontuação (*score*) de confiança das mesmas. A Figura 13 ilustra a ideia básica deste algoritmo.

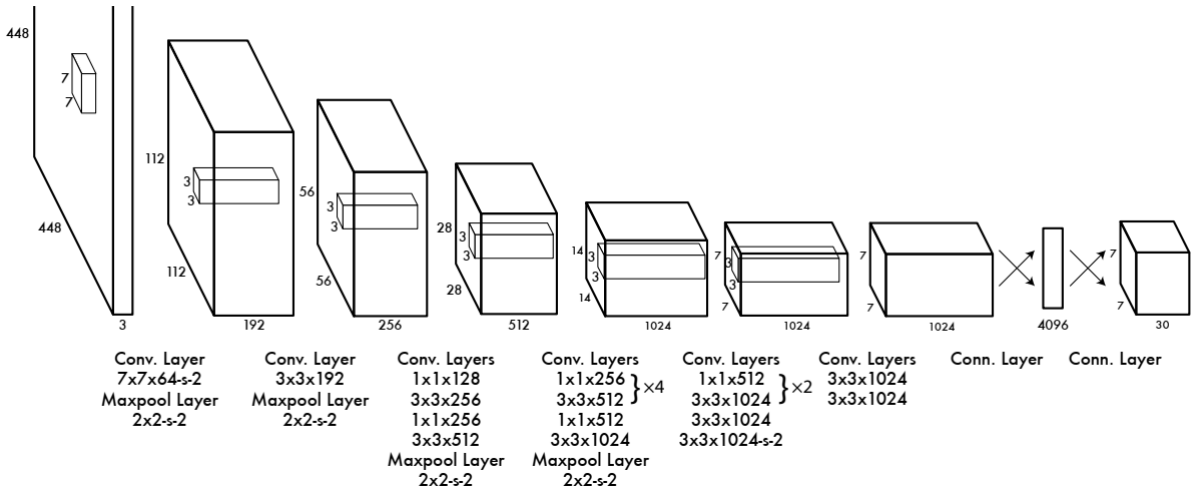
Redmon et al. (2016) utiliza as camadas convolucionais iniciais para extrair uma rede de características da imagem, enquanto as camadas totalmente conectadas predizem as probabilidades e coordenadas. A arquitetura Yolo tem 24 camadas convolucionais seguidas de 2 camadas totalmente conectadas, conforme ilustrado na Figura 14.

Figura 13 – Modelo Yolo (You only look once)



Fonte: adaptado de Redmon et al. (2016)

Figura 14 – Arquitetura YoLo



Fonte: Redmon et al. (2016)

O Yolo passou por diversas melhorias ao longo dos anos incluindo as técnicas mais avançadas da época, o que fez Yolo estar frequentemente entre os melhores algoritmos de

detecção de objetos (THUAN, 2021).

YoloV2: adiciona normalização em lotes para permitir o treinamento mais rápido e estável de redes neurais profundas e estabilizar a distribuição das camadas de entrada durante o treinamento, assim como o uso de classificadores de maior resolução e caixas de âncoras na predição de objetos (LI; YANG, 2018).

Yolov3: maior rede (30 camadas convolucionais) com o uso do ResNet e detecção de múltiplas escalas em busca de maior precisão (REDMON et al., 2016).

Yolov4: utiliza como espinha dorsal (backbone) para extração de características o CSPDarknet53, como pescoço (neck) SPP e PAN para determinação das melhores característica e como cabeça (head) o YoloV3 previsão do vetor contendo as coordenadas da caixa delimitadora (centro, altura, largura), a pontuação de confiança da previsão e as classes de probabilidade (BOCHKOVSKIY; WANG; LIAO, 2020).

4.3 Yolov5

O Yolov5 é um algoritmo de detecção em único estágio e possui três partes principais: espinha dorsal, pescoço, e cabeça (ULTRAANALYTICS, 2021).

Na espinha dorsal, uma rede parcial de estágio cruzado (do inglês cross stage partial networks - CSPNet) é usada para extrair características importantes da imagem de entrada. A CSPNet tem menor uso memória e custo computacional e permite melhor precisão em algoritmos de detecção.

A segunda parte (pescoço) é encarregada de gerar pirâmides de características que ajudam modelos a generalizar o dimensionamento de objetos, ou ainda, a identificar o mesmo objeto em diferentes tamanhos e escalas. Essas pirâmides são úteis para o desempenho de modelos em dados ainda não vistos. A técnica utilizada pelo YOLOv5 é a PANet, embora existam outras como FPN (LIN et al., 2017a), BiFPN (TAN; PANG; LE, 2020) e SPP (HE et al., 2015).

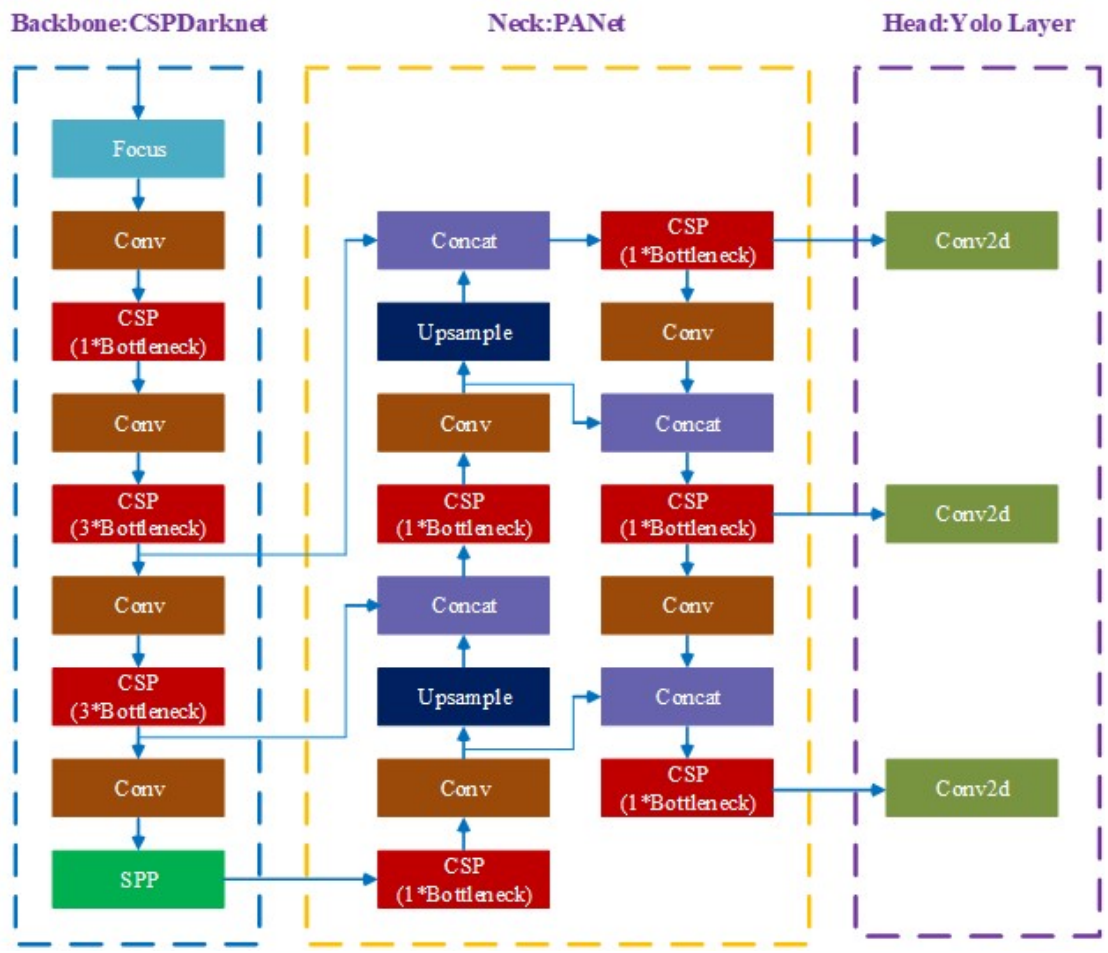
A cabeça (head) do modelo faz a detecção final: as caixas de contorno são aplicadas e as probabilidades de classe de cada objeto calculado. No YOLOv5, o modelo da cabeça é igual ao YOLO v3 e v4. A Figura 15 ilustra a arquitetura do Yolov5.

Os componentes constituintes dessa arquitetura são detalhados na Figura 16.

4.4 Aspectos computacionais para desenvolvimento do trabalho

A implementação dos algoritmos será feita no Google Colab, uma plataforma online que permite a criação de códigos Python pelo navegador. Além disso, o Colab não requer configuração para ser usado, ao mesmo tempo que fornece acesso gratuito a recursos de computação tais como GPUs. As bibliotecas PyTorch e TensorFlow serão

Figura 15 – Arquitetura YOLO v5



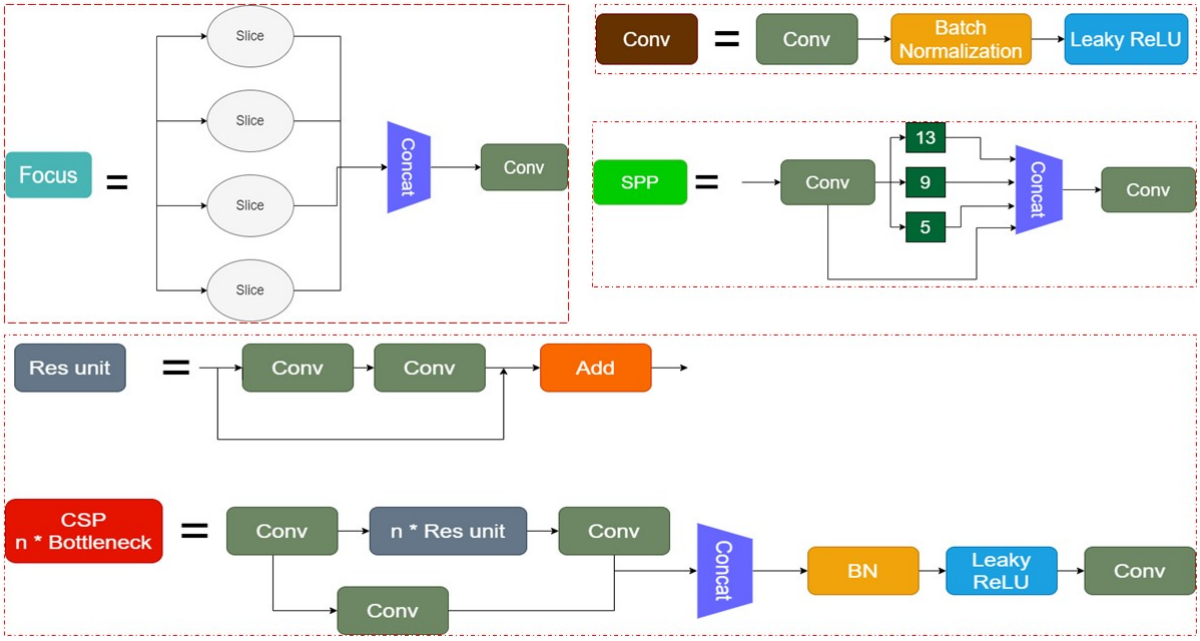
Fonte: Adaptado de https://github.com/ultralytics/yolov5/issues/280

essenciais ao desenvolvimento da aplicação. A primeira é uma biblioteca de código aberto para Python, desenvolvida pela equipe do Facebook AI Research, utilizada amplamente em aplicações de Aprendizado Profundo com GPUs e CPUs. Já o TensorFlow, é uma biblioteca de código aberto para aprendizado de máquina desenvolvido pela equipe do Google Brain. O resultado dos experimentos será armazenado na plataforma Weights & Biases (W&B), uma plataforma utilizada para acompanhar experimentos de Machine Learning. O Google Colab, Pytorch, TensorFlow e W&B podem ser encontrados nos endereços a seguir: *https://research.google.com/colaboratory/*, *https://pytorch.org/*, *https://www.tensorflow.org/*, *https://wandb.ai/site*.

4.5 Pré-processamento dos dados

Primeiramente as imagens sem objetos anotados serão segregadas e deixadas de fora da construção do modelo. Em seguida, será explorado a existência de duplicidade, tendo como premissa que *não é possível detectar o mesmo objeto com as mesmas coordenadas em*

Figura 16 – Componentes da arquitetura YOLO v5



Fonte: Adaptado de (YAO et al., 2021)

Figura 17 – Google Colab, Pytorch, Tensorflow, W&B

Fonte: Google Image

imagens diferentes ou na mesma imagem. Note que em uma base com imagens aumentadas (criadas a partir da aplicação de um filtro em imagens existentes), seria possível detectar um objeto em uma mesma localização múltiplas vezes, mas essas imagens não passaram por aumentação. Nesse caso, imagens duplicadas serão removidas da base de dados. Além disso, os rótulos não pertencentes ao universo de representação do problema serão tratados como anomalias e também removidos. O tamanho das imagens também será padronizado a fim de garantir melhor desempenho do algoritmo.

4.6 Treinamento e validação

Os dados serão divididos em dois subconjuntos, treinamento e teste. O primeiro será utilizado para ajustar o modelo de detecção e o segundo, para validar as métricas de desempenho.

As imagens iniciais serão divididas aleatoriamente, sem reposição, em uma proporção de 67% para treinamento e 33% para teste. Uma outra etapa fundamental é a definição do tamanho de lote, ou batch, (quantidade de imagens utilizadas por iteração no loop de treinamento) e épocas, ou epoch, (quantidade de vezes que o modelo passará por todos as imagens). A cada época, o algoritmo passará por todos os lotes de imagens e atualizará os pesos do modelo de forma a minimizar a função de custo. Além disso, ao final de cada época, o desempenho do modelo é avaliado em uma parcela de imagens destinadas a validação. Se os pesos atuais implicarem em um maior mAP, eles serão salvos como os melhores. Além disso, os pesos da última iteração também são armazenados como ponto de retorno caso o treinamento modelo esteja piorando. A otimização dos parâmetros foi feita usando os métodos SGD e Adam.

Após o treinamento, com os melhores pesos determinados, o modelo será empregado para detectar objetos nas imagens dedicadas a teste e as métricas de desempenho serão reavaliadas. Quanto maior a média de precisão-média (mAP), melhor o modelo.

4.7 Métricas de avaliação

Área de intersecção sobre união (do inglês Intersection over union - IoU) mede a sobreposição entre duas fronteiras. Essa métrica é usada para mensurar o quanto a fronteira prevista se sobrepõe à verdade (a fronteira do objeto real). Se o valor de IoU for 0.5, é necessário somente 50% de sobreposição para realizar a classificação, sendo ela um verdadeiro positivo ou falso positivo. A Figura 18 ilustra a IoU.

Precisão (Precision) mede a precisão de suas previsões, ou ainda, a porcentagem de suas previsões que estão corretas. A Equação 4.9 descreve o cálculo de precisão.

$$Precision = \frac{TP}{TP + FP} \tag{4.9}$$

onde,

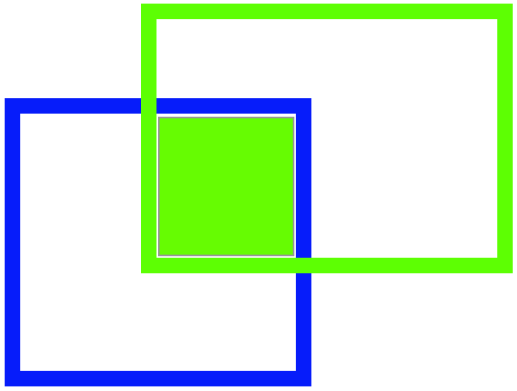
TP = True Positive (objetos classificadas corretamente)

FP = False Positive (objetos classificados erroneamente)

Revocação (Recall) mede quão bom os positivos estão sendo encontrados. A Equação 4.10 descreve matematicamente o Recall.

$$Recall = \frac{TP}{TP + FN} \tag{4.10}$$

Figura 18 – Área de intersecção sobre união - IoU



Fonte: o autor (2021)

onde,

TP = True Positive (objetos classificadas corretamente)

FP = False Positive (objetos classificados erroneamente)

FN = False Negative (objetos não classificados)

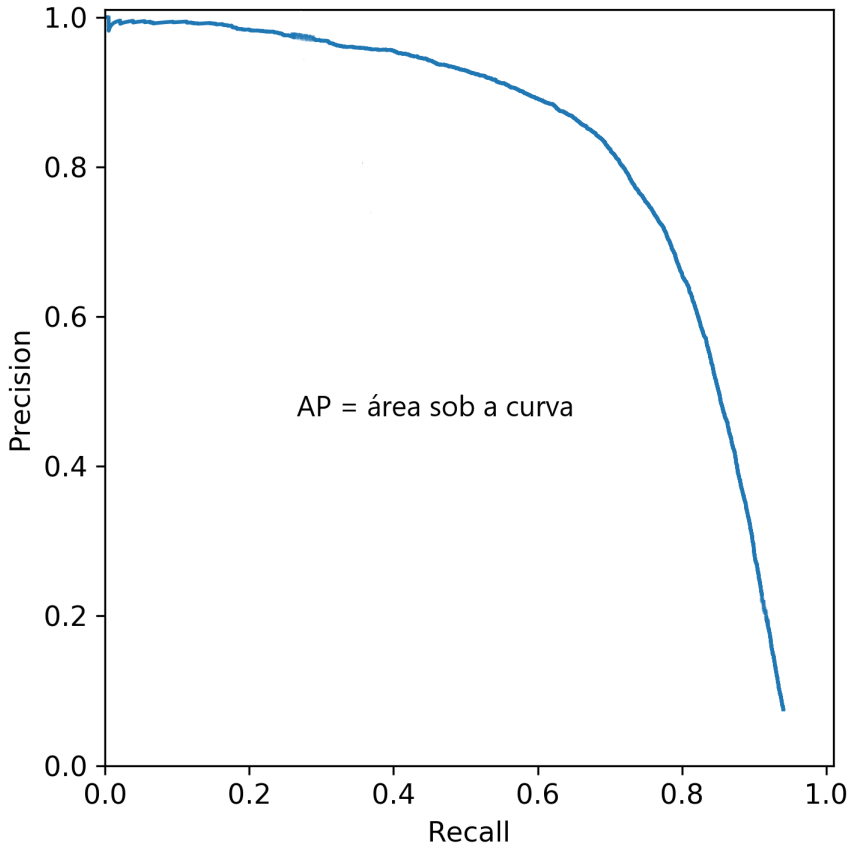
Os valores de precision e recall estão entre 0 e 1, sendo 0 o pior valor e 1 o melhor. Geralmente estas duas métricas são analisadas conjuntamente, pois cada uma avalia um aspecto diferente da classificação.

A **média de precisão-média** (mean average precision - mAP) para um conjunto de detecções é a média sobre as classes, da precisão média interpolada para cada classe (HENDERSON; FERRARI, 2016). A precisão média de uma classe é dada pela área sob a curva precision/recall para as detecções, assim como ilustrado na Figura 19. Em geral, quanto maior o valor de mAP, melhor o modelo de detecção de objetos.

4.8 Base de dados

A base de dados é composta de 15000 imagens. Os objetos são representados por 11 diferentes classes, tais como carro (“car”), pedestres (“pedestrian”), ciclista (“biker”), caminhão (“truck”) e outras relacionadas à semáforos: trafficLight-Red, trafficLight-Green, trafficLight, trafficLight-RedLeft, trafficLight-GreenLeft, trafficLight-Yellow, trafficLight-

Figura 19 – Média de Precisão-média: mAP



Fonte: adaptado de https://github.com/ultralytics/yolov3/issues/898

YellowLeft. Ainda mais, as imagens não foram capturas em ambientes adversos (noite, chuva, neve, neblina).

Os arquivos estão disponíveis na plataforma Roboflow, em diversos formatos, inclusive para *Yolov5* com a biblioteca Pytorch. A versão manipulada nesse trabalho é a *fixed small*, com as imagens na resolução de 512×512 pixels e um total de aproximadamente 1,1 GB de dados, os quais podem ser encontrados em https://public.roboflow.com/object-detection/self-driving-car e recuperados. Para mais informações sobre como recuperar os dados, verifique o Anexo A.2.

4.9 Tratamento dos dados

Ao total, 26300 imagens foram obtidas da base de dados. Um *dataframe* em Pandas foi construído para auxiliar o tratamento dos dados. Em cada linha desta estrutura havia as posições (x, y) inferior e superior da caixa de contorno, a classe do objeto (biker: 0, car: 1, pedestrian: 2, trafficLight: 3, trafficLight_Green: 4, trafficLight_GreenLeft: 5, traffi-

Tabela 1 – Dataframe com características dos dados

Classe	xi	yi	xu	yu	imagem	lables
10	0.43359	0.48828	0.0166015	0.0283203	/14780...jpg	/14780...txt
1	0.458984	0.494140	0.0244140	0.0263671	/14780...jpg	/14780...txt
2	0.9287109	0.5107421	0.0361328	0.21875	/14780...jpg	/14780...txt

cLight_Red: 6, trafficLight_RedLeft: 7, trafficLight_Yellow: 8, trafficLight_YellowLeft: 9, truck: 10), o caminho da imagem (*.jpg*) e do arquivo com os rótulos (*.txt*). A Tabela 1 traz alguns registros desse dataframe.

As linhas do dataframe com os mesmos valores de classe, e coordenadas (x,y) das caixas de contorno foram eliminadas devido à premissa de que "não é possível detectar o mesmo objeto com as mesmas coordenadas em imagens diferentes". Dessa forma, essas imagens e arquivos de labels não foram considerados no treinamento do modelo. No entanto, o caso de objetos identificados múltiplas vezes em uma mesa imagem não foi tratado.

A obtenção do total de rótulos por classe deu-se pela contabilização de todas os rótulos do dataframe pandas. O total de objetos por classes está ilustrado na Figura 20.

4.10 Treinamento e teste do modelo Yolov5

A divisão dos dados entre treinamento e teste foi realizada usando a função *train_test_split* da biblioteca Scikit-Learn do Python. Essa função divide aleatoriamente um conjunto de dados entre treinamento e teste de acordo com proporção desejada. O dataframe construído durante a limpeza foi a entrada desta função e as saídas geradas foram os caminhos dos arquivos das imagens rótulos para treinamento e teste.

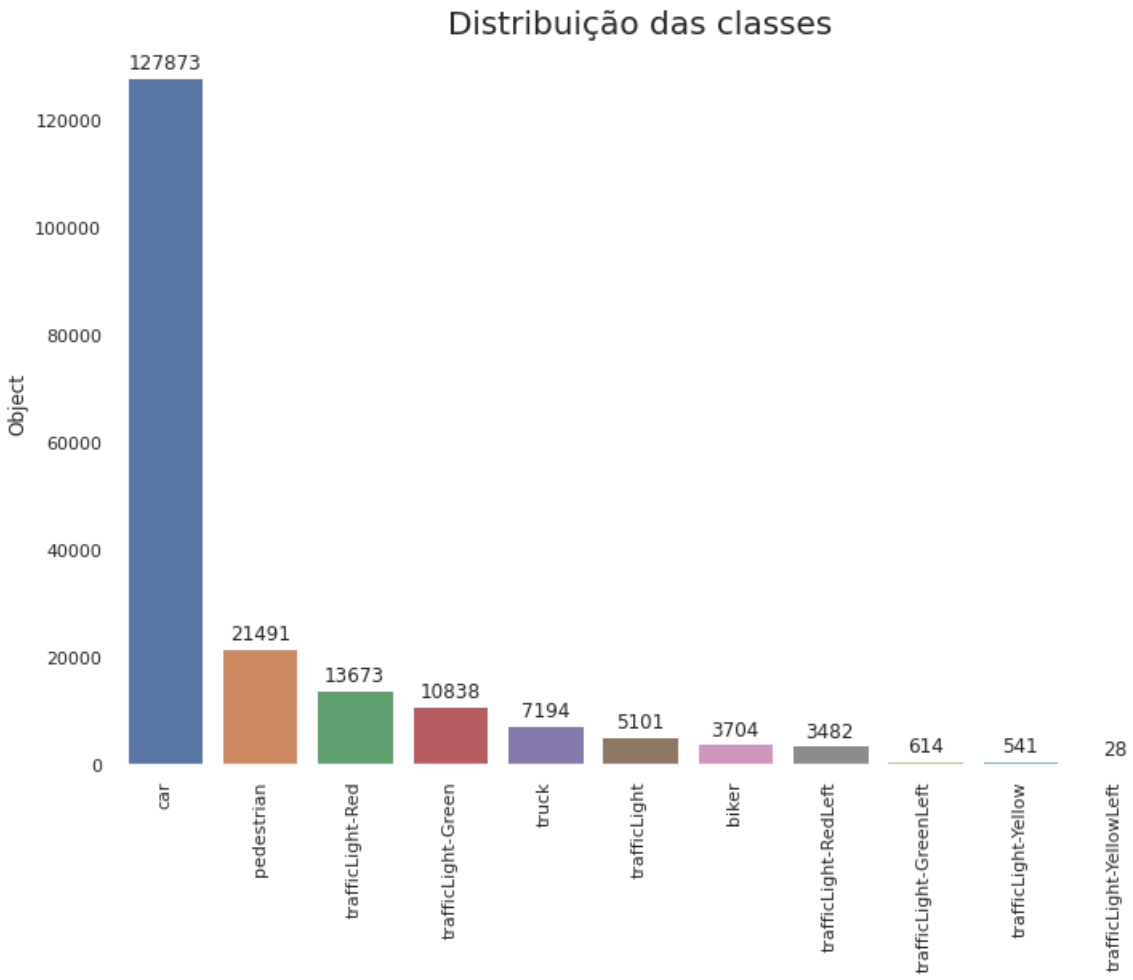
Com os dados limpos, o treinamento do modelo pôde ser realizado com o comando abaixo. Utilizou-se o yolov5s, arquitetura mais simples do Yolo, descrita na seção 4.3.

```
1 !python train.py --img <SIZE_IMG_ENTRADA> --batch <LOTE> --epochs <
    EPOCAS> --data <ARQUIVOS(IMG_LABELS).yaml> --cfg <MODELO_YOLO.yaml
    > --weights <PESOS_MODELO.pt> --name <PATH_RESULTADOS> --adam <
    REMOVER_PARA_SGD> --cache <OPCIONAL>
```

O algoritmo automaticamente redimensiona as imagens de entrada de acordo com o valor de SIZE_IMG_ENTRADA. Neste caso, o tamanho foi ide 416 a fim de reduzir o custo computacional no backbone, neck e head do algoritmo. A validação do modelo treinado se deu pelo comando descrito abaixo.

```
1 !python val.py --data <CAMINHO_ARQUIVOS(IMAGENS_LABELS).yaml> --
    weights <PESOS_MODELO.pt> --img <TAMANHO_IMG_ENTRADA> --name <
    PATH_RESULTADOS>
```

Figura 20 – Distribuição das classes.



Fonte: o autor (2021).

5 RESULTADOS

Nessa seção são descritos os resultados de detecção de objetos em uma base de dados pública extraída da plataforma Roboflow, cuja as imagens foram capturadas por sistemas de percepção de veículos autônomos. Mais especificamente, será mostrado um método de tratamento de dados inicial, o treinamento da rede Yolov5 para diferentes configurações de épocas, otimizador e função de custo, os desempenhos dos modelos nas métricas de avaliação discutidas anteriormente e a análise visual de diversos casos de teste extraídos da própria base.

É importante destacar que até o momento não foi encontrado nenhum trabalho em que a média de precisão-média do algoritmo Yolov5 era avaliado para combinações distintas de épocas, algoritmos de otimização e função de custo nos dados da Roboflow, descritos na seção 4.8.

5.1 Experimentos

Para descobrir os melhores hiperparâmetros do modelo, foram realizados diversos experimentos. Em cada um deles um tipo de hiperparâmetro foi alterado, a fim de se observar o efeito no valor da métrica mAP com área de intersecção da união de 50% (mAP@0.5). Entre os principais hiperparâmetros avaliados, destaca-se: i) o tamanho do batch, ii) o otimizador, iii) a quantidade de épocas de treinamento, e vi) a função de custo.

Para mais informações sobre essas análises, consulte o anexo A.1.

Alguns experimentos também foram realizados usando a técnica de data augmentation embutido na implementação original da rede Yolov5. Quanto a variação do tamanho do batch e da utilização de data augmentation, os resultados preliminares não apresentaram diferenças significativas.

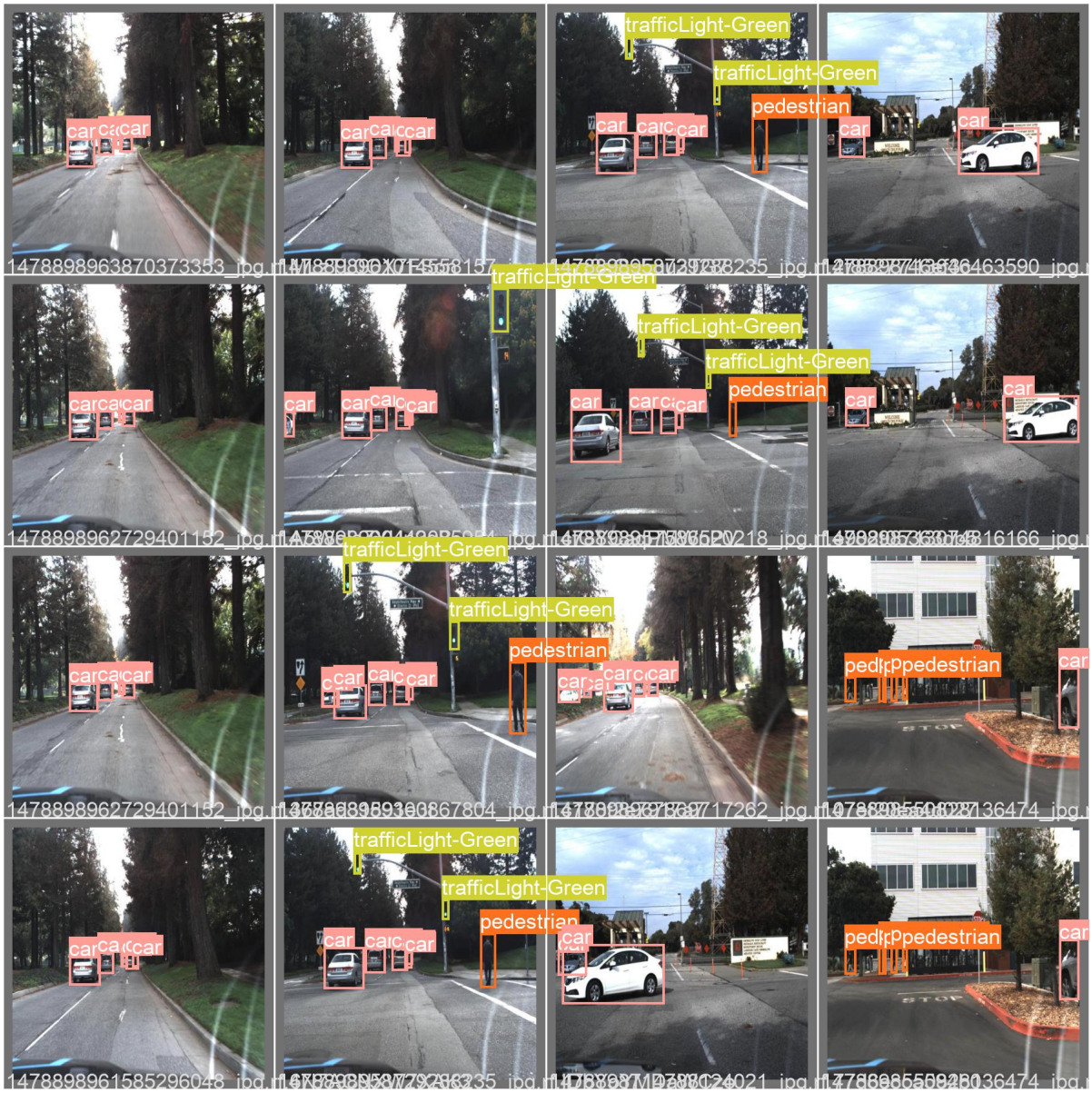
As Figuras 21 e 22 ilustram os objetos da base de dados e os objetos detectados pelo modelo, respectivamente.

Observou-se nos experimentos que a média de precisão-média (mAP) em geral foi maior (melhor) para os modelos criados com o otimizador SGD comparado ao Adam. Quanto a função de custo, os modelos com a BCEWithLogitsLoss apresentaram maior mAP que a Focal Loss. A Figura 23 exibe os valores de mAP obtidos nos experimentos.

Em geral, o melhor modelo Yolov5 foi treinado por 240 épocas, com a função de custo BCEWithLogitsLoss e teve mAP@0.5 para todas as classes igual a 74.1%. A Figura 24 ilustra a curva PR (precision-recall) do modelo para todas as classes detectadas.

O crescimento do valor de mAP ao longo das épocas (epochs) de treinamento é

Figura 21 – Objetos reais.



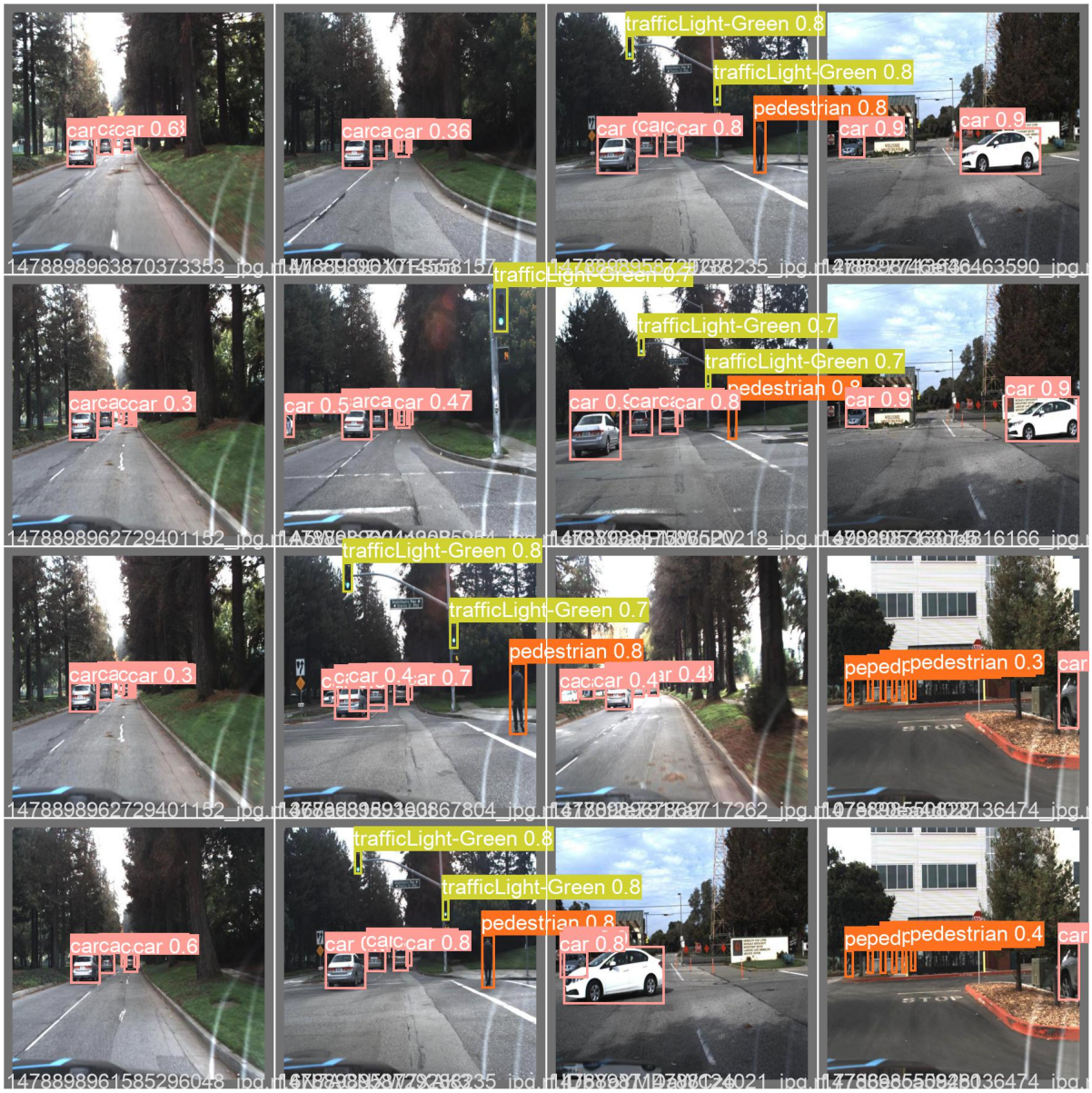
Fonte: o autor (2021).

exibido na Figura 25. Observe que o ganho em mAP se torna pequeno após 200 épocas de treinamento. Comparando o mAP dos modelos treinados por 240 e 200, nota-se que o aumento foi de 0.004 para 2:17h adicionais de processamento. Portanto, com um custo computacional menor, o modelo treinado por 200 épocas seria uma melhor opção para um ambiente de produção.

A matriz-confusão da Figura 26 traz uma visão geral de como o modelo classificou os diversos objetos comparado à rotulação verdadeira da base.

Observa-se que a maioria dos carros (“car”) foi classificada corretamente, no entanto

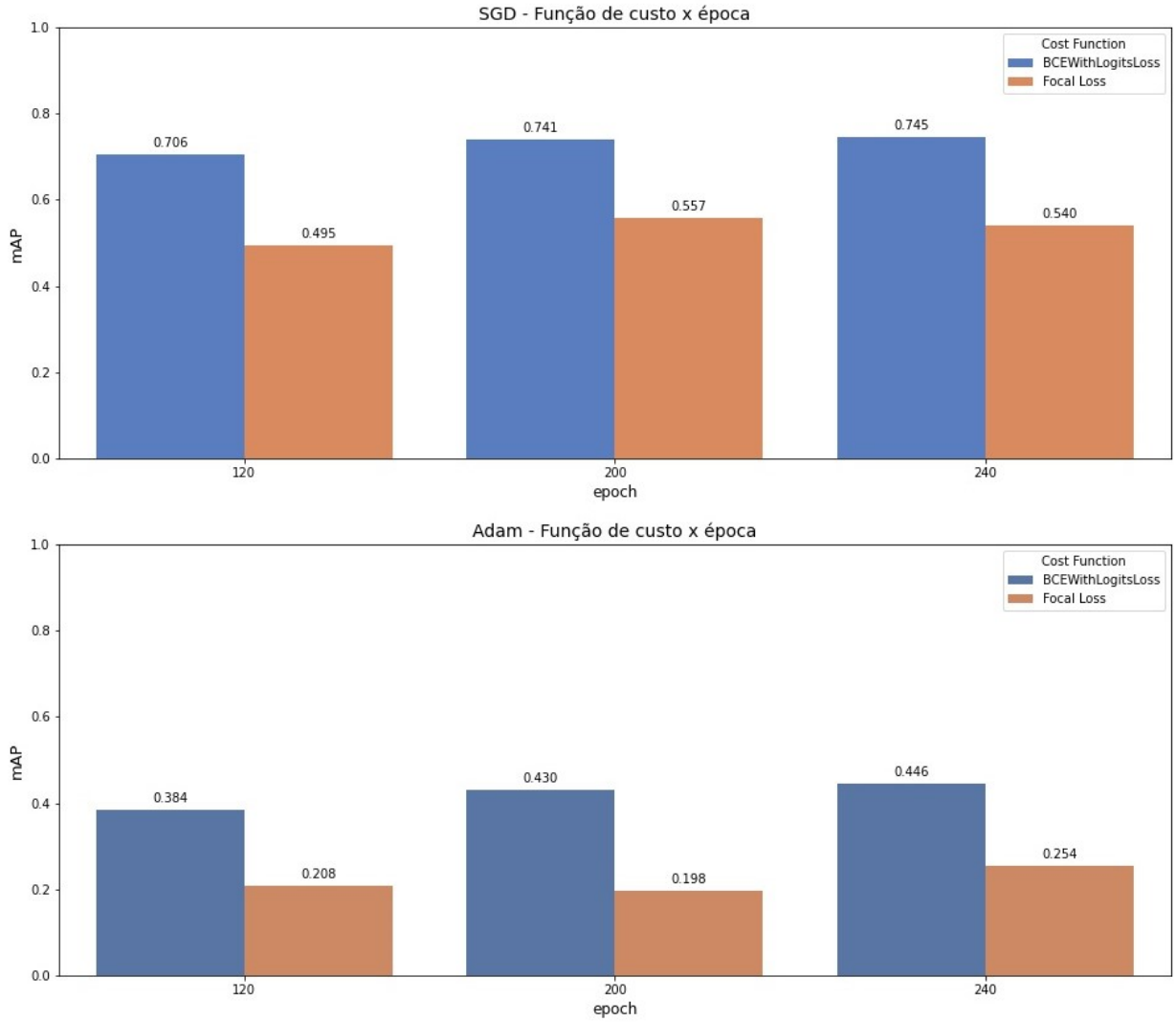
Figura 22 – Objetos detectados pelo modelo.



Fonte: o autor (2021).

alguns caminhões (“truck”) foram categorizados como carro. Além disso, a maioria dos pedestres (“pedestrian” e ciclistas (“biker”) foram identificados corretamente, mas em casos raros, o algoritmo confundiu ciclistas e pedestres. Por outro lado, os caminhões foram rotulados com acerto em geral, mas enquadrados como um sinal de trânsito inusitadamente em pouquíssimas ocorrências. Ademais, os sinais foram bem rotulados (com exceção do “trafficLight_yellow_Left”), contudo as diferentes cores de sinal levou a diversas classificações erradas. O “trafficLight_yellow_Left” foi o objeto com maior dificuldade de detecção, sendo totalmente classificado como background da imagem. Sobre os objetos identificados onde havia somente o plano de fundo, os carros e pedestres tiveram a maior

Figura 23 – mAP x otimizador x época x função de custo.



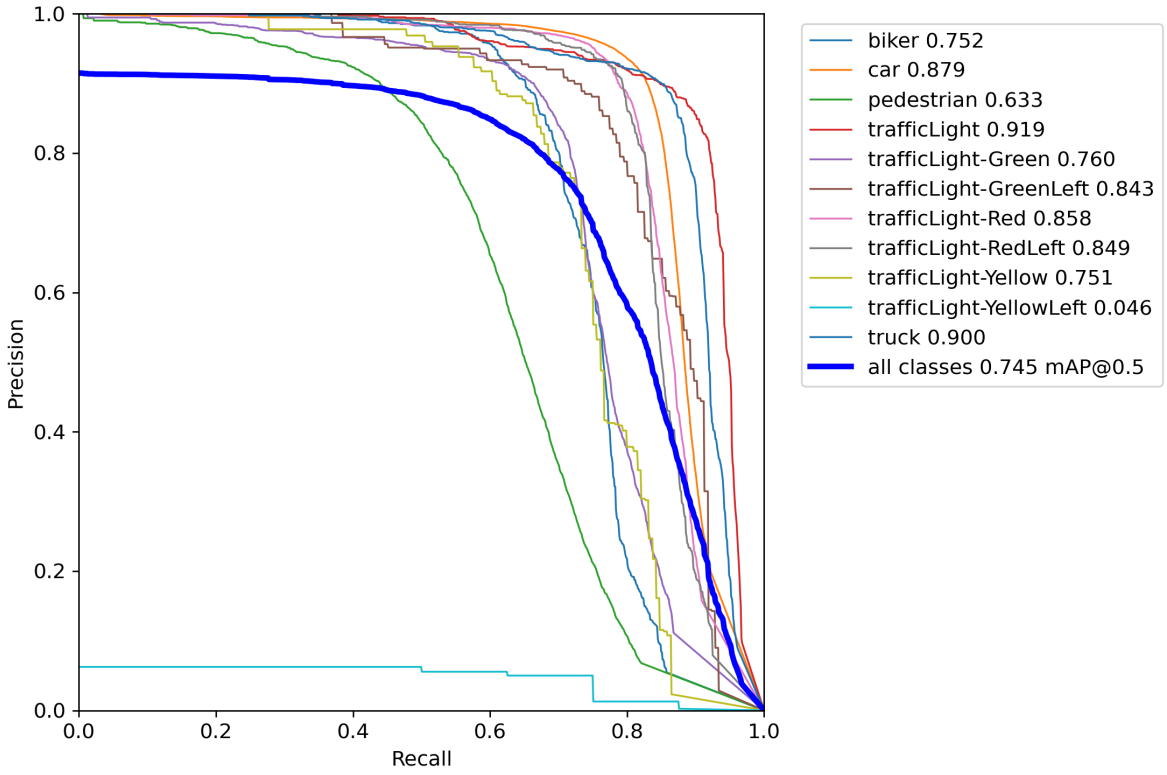
Fonte: o autor (2021).

taxa de falsos positivos.

A Tabela 2 lista os valores de precisão, revocação e mAP geral de todos os experimentos realizados. O conjunto completo de hiperparâmetros do modelo que apresentou melhores resultados na métrica de avaliação mAP pode ser encontrado no A.3.

É imprescindível ressaltar que até o momento nenhum trabalho com essa ótica de avaliação nessa base de dados foi encontrado. Os valores apresentados podem servir de base para futuras pesquisas.

Figura 24 – Curva PR (Precision-Recall).

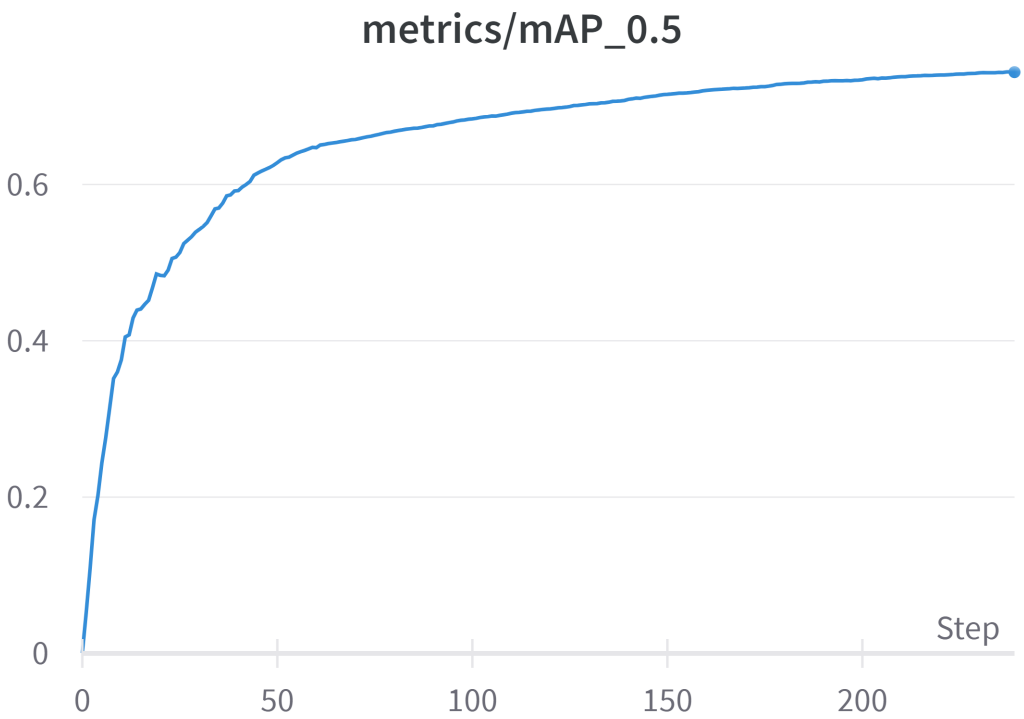


Fonte: o autor (2021).

Tabela 2 – Experimentos

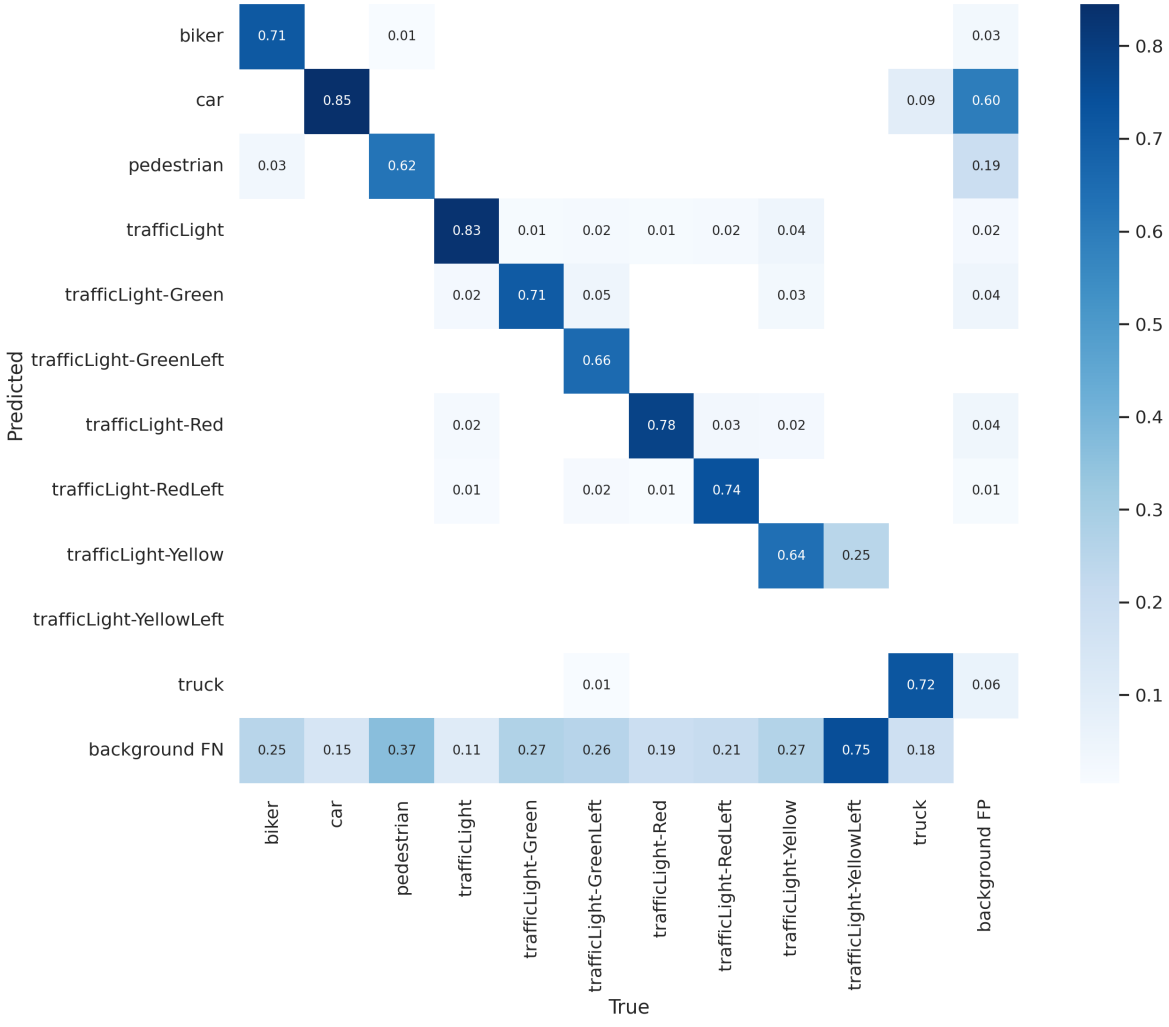
Id	Batch	Epochs	Otimizador	Loss	P	R	mAP @ 0.5	Tempo (h)
1	64	120	SGD	BCEWithLogits	0.82	0.665	0.706	08:20
2	64	120	Adam	BCEWithLogits	0.626	0.364	0.384	07:52
3	64	120	SGD	Focal loss	0.829	0.445	0.495	19:20
4	64	120	Adam	Focal loss	0.504	0.311	0.208	08:40
5	64	200	SGD	BCEWithLogits	0.865	0.688	**0.741**	12:19
6	64	200	Adam	BCEWithLogits	0.652	0.400	0.430	12:21
7	64	200	SGD	Focal loss	0.879	0.505	0.505	05:52
8	64	200	Adam	Focal loss	0.487	0.285	0.198	15:48
9	64	240	SGD	BCEWithLogits	0.882	0.676	**0.745**	14:36
10	64	240	Adam	BCEWithLogits	0.683	0.402	0.446	14:38
11	64	240	SGD	Focal loss	0.866	0.491	0.540	11:22
12	64	240	Adam	Focal loss	0.551	0.276	0.254	22:28

Figura 25 – mAP x epochs.



Fonte: o autor (2021).

Figura 26 – Matriz confusão.



Fonte: o autor (2021).

6 CONCLUSÃO

A detecção de objetos em sistemas de veículos autônomos é fundamental para o planejamento de rota e sequente controle do veículo. Este trabalho utilizou um dos algoritmos de detecção de objetos mais recentes em uma base de dados pública e analisou como a variação da quantidade de épocas, função de custo e otimizador impacta na eficiência do modelo construído. O algoritmo Yolov5 com otimizador SGD e função de custo BCEWithLogitsLoss, treinado por 240 épocas, apresentou melhor resultado, com média de precisão-média (mAP) igual a $74,5\%$. No entanto, o melhor balanço entre custo computacional e mAP foi encontrado em 200 épocas. É importante mencionar a média de precisão-média poderia ser maior se alguns rótulos do treinamento relacionados às luzes de tráfego fossem removidos ou se a quantidade imagens contendo essas classes fosse maior durante o treinamento, ou seja, se houvesse melhor balanceamento de classes.

Como recomendação para futuros trabalhos, outras arquiteturas mais complexas e maiores do Yolov5 poderiam ser exploradas na mesma base de dados. Além disso, uma técnica de aumentação de dados poderia ser aplicada à priori do treinamento. Uma outra linha de pesquisa seria o desenvolvimento de um sistema de alerta de colisão. Há ainda o cenário em que a eficiência do modelo poderia ser avaliada em condições adversas de tráfego, como por exemplo, chuva, neblina e noite.

REFERÊNCIAS

ALBAWI, S.; MOHAMMED, T. A.; AL-ZAWI, S. Understanding of a convolutional neural network. In: IEEE. **2017 International Conference on Engineering and Technology (ICET)**. [S.l.], 2017. p. 1–6.

ALE, L.; ZHANG, N.; LI, L. Road damage detection using retinanet. In: **2018 IEEE International Conference on Big Data (Big Data)**. [S.l.: s.n.], 2018. p. 5197–5200.

BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. **YOLOv4: Optimal Speed and Accuracy of Object Detection**. 2020.

BRUMMELEN, J. V. et al. Autonomous vehicle perception: The technology of today and tomorrow. **Transportation research part C: emerging technologies**, Elsevier, v. 89, p. 384–406, 2018.

FAN, Q.; BROWN, L.; SMITH, J. A closer look at faster r-cnn for vehicle detection. In: IEEE. **2016 IEEE intelligent vehicles symposium (IV)**. [S.l.], 2016. p. 124–129.

FENG, D. et al. Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 22, n. 3, p. 1341–1360, 2020.

GARCIA, J. et al. **A Comprehensive Study of Autonomous Vehicle Bugs**. In: . New York, NY, USA: Association for Computing Machinery, 2020. (ICSE '20), p. 385–396. ISBN 9781450371216. Disponível em: <https://doi-org.ez67.periodicos.capes.gov.br/10.1145/3377811.3380397>.

GARNETT, N. et al. Real-time category-based and general obstacle detection for autonomous driving. In: **Proceedings of the IEEE International Conference on Computer Vision Workshops**. [S.l.: s.n.], 2017. p. 198–205.

GHOLAMALINEZHAD, H.; KHOSRAVI, H. Pooling methods in deep neural networks, a review. **arXiv preprint arXiv:2009.07485**, 2020.

GIRSHICK, R. Fast r-cnn. In: **Proceedings of the IEEE international conference on computer vision**. [S.l.: s.n.], 2015. p. 1440–1448.

GIRSHICK, R. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2014. p. 580–587.

HAN, D. Comparison of commonly used image interpolation methods. In: **Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)**. [S.l.: s.n.], 2013. v. 10.

HE, K. et al. Spatial pyramid pooling in deep convolutional networks for visual recognition. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 37, n. 9, p. 1904–1916, 2015.

HENDERSON, P.; FERRARI, V. End-to-end training of object class detectors for mean average precision. In: SPRINGER. **Asian Conference on Computer Vision**. [S.l.], 2016. p. 198–213.

HIJAZI, S. et al. Using convolutional neural networks for image recognition. **Cadence Design Systems Inc.: San Jose, CA, USA**, p. 1–12, 2015.

HOSANG, J. et al. Taking a deeper look at pedestrians. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2015. p. 4073–4082.

HUANG, Y.; CHEN, Y. **Autonomous Driving with Deep Learning: A Survey of State-of-Art Technologies**. 2020.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: PMLR. **International conference on machine learning**. [S.l.], 2015. p. 448–456.

JANAI, J. et al. Computer vision for autonomous vehicles: Problems, datasets and state of the art. **Foundations and Trends® in Computer Graphics and Vision**, v. 12, n. 1–3, p. 1–308, 2020. ISSN 1572-2740. Disponível em: <http://dx.doi.org/10.1561/0600000079>.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014.

LI, R.; YANG, J. Improved yolov2 object detection model. In: IEEE. **2018 6th International Conference on Multimedia Computing and Systems (ICMCS)**. [S.l.], 2018. p. 1–6.

LIN, T.-Y. et al. Feature pyramid networks for object detection. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2017. p. 2117–2125.

LIN, T. Y. et al. Focal loss for dense object detection. In: **Proceedings of the IEEE international conference on computer vision**. [S.l.: s.n.], 2017. p. 2980–2988.

LIU, W. et al. Ssd: Single shot multibox detector. In: SPRINGER. **European conference on computer vision**. [S.l.], 2016. p. 21–37.

MAURER, M. et al. **Autonomous driving: technical, legal and social aspects**. [S.l.]: Springer Nature, 2016.

MUHAMMAD, K. et al. Deep learning for safe autonomous driving: Current challenges and future directions. **IEEE Transactions on Intelligent Transportation Systems**, p. 1–21, 2020.

MUNIR, F. et al. Autonomous vehicle: The architecture aspect of self driving car. In: **Proceedings of the 2018 International Conference on Sensors, Signal and Image Processing**. New York, NY, USA: Association for Computing Machinery, 2018. (SSIP 2018), p. 1–5. ISBN 9781450366205. Disponível em: <https://doi-org.ez67.periodicos.capes.gov.br/10.1145/3290589.3290599>.

O'SHEA, K.; NASH, R. An introduction to convolutional neural networks. **arXiv preprint arXiv:1511.08458**, 2015.

PEI, D. et al. A fast retinanet fusion framework for multi-spectral pedestrian detection. **Infrared Physics & Technology**, Elsevier, v. 105, p. 103178, 2020.

PRABHAKAR, G. et al. Obstacle detection and classification using deep learning for tracking in high-speed autonomous driving. In: **2017 IEEE Region 10 Symposium (TENSYMP)**. [S.l.: s.n.], 2017. p. 1–6.

QIAN, R. et al. Road surface traffic sign detection with hybrid region proposal and fast r-cnn. In: **2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)**. [S.l.: s.n.], 2016. p. 555–559.

REDMON, J. et al. You only look once: Unified, real-time object detection. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 779–788.

REN, S. et al. **Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks**. 2016.

ROSASCO, L. et al. Are loss functions all the same? **Neural computation**, MIT Press, v. 16, n. 5, p. 1063–1076, 2004.

RUDER, S. An overview of gradient descent optimization algorithms. arxiv 2016. **arXiv preprint arXiv:1609.04747**, 2016.

SARDA, A.; DIXIT, S.; BHAN, A. Object detection for autonomous driving using yolo [you only look once] algorithm. In: IEEE. **2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)**. [S.l.], 2021. p. 1370–1374.

SHARMA, D.; KUMAR, N. A review on machine learning algorithms, tasks and applications. **International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)**, v. 6, n. 10, p. 1548–1552, 2017.

SHARMA, S.; SHARMA, S. Activation functions in neural networks. **Towards Data Science**, v. 6, n. 12, p. 310–316, 2017.

STEINWART, I. How to compare different loss functions and their risks. **Constructive Approximation**, Springer, v. 26, n. 2, p. 225–287, 2007.

SUZUKI, K. **Artificial neural networks: methodological advances and biomedical applications**. [S.l.]: BoD–Books on Demand, 2011.

TAKUMI, K. et al. Multispectral object detection for autonomous vehicles. In: **Proceedings of the on Thematic Workshops of ACM Multimedia 2017**. [S.l.: s.n.], 2017. p. 35–43.

TAN, M.; PANG, R.; LE, Q. V. Efficientdet: Scalable and efficient object detection. In: **Proceedings of the IEEE/CVF conference on computer vision and pattern recognition**. [S.l.: s.n.], 2020. p. 10781–10790.

THUAN, D. Evolution of yolo algorithm and yolov5: the state-of-the-art object detection algorithm. 2021.

ULTRAANALYTICS. Yolov5. In: . [s.n.], 2021. Disponível em: <https://ultralytics.com/yolov5>.

WANG, H. et al. A comparative study of state-of-the-art deep learning algorithms for vehicle detection. **IEEE Intelligent Transportation Systems Magazine**, IEEE, v. 11, n. 2, p. 82–95, 2019.

WANG, S.-C. Artificial neural network. In: _____. **Interdisciplinary Computing in Java Programming**. Boston, MA: Springer US, 2003. p. 81–100. ISBN 978-1-4615-0377-4. Disponível em: <https://doi.org/10.1007/978-1-4615-0377-4_5>.

WISEMAN, Y. Autonomous vehicles. In: **Encyclopedia of Information Science and Technology, Fifth Edition**. [S.l.]: IGI Global, 2021. p. 1–11.

YAO, J. et al. A real-time detection algorithm for kiwifruit defects based on yolov5. **Electronics**, Multidisciplinary Digital Publishing Institute, v. 10, n. 14, p. 1711, 2021.

ZHANG, Y. **New advances in machine learning**. [S.l.]: BoD–Books on Demand, 2010.

Anexos

ANEXO A – ANEXO

A.1 Experimentos - batch e augmentation

Os dados dos experimentos com batch de tamanho 32 e data augmentation próprio do Yolov5 estão sumarizados na Tabela 3.

Tabela 3 – Experimentos - batch e augmentation

Batch	Epochs	Otimizador	Loss	Augmentation	P	R	mAP@0.5	Tempo(h)
32	120	SGD	BCE	No	0.846	0.646	0.703	07:47
32	120	Adam	BCE	No	0.603	0.358	0.375	08:47
64	120	SGD	BCE	Yes	0.855	0.637	0.701	07:30
64	240	SGD	BCE	Yes	0.904	0.674	0.744	15:01
64	240	Adam	BCE	Yes	0.636	0.369	0392	13:52

A.2 Arquivos

Jupyter Notebook

https://github.com/TallesRodrigues/Object-detection-autonomous-vehicle

Imagens e labels

https://www.kaggle.com/tallesrodrigues/autonomous-vehicle-roboflow

A.2.1 Versões de bibliotecas

Torch = 1.10.0+cu111
Utils = 1.0.1
GPU = Tesla P100-PCIE-16GB
matplotlib >= 3.2.2 numpy >= 1.18.5 opencv-python >= 4.1.2 Pillow >= 7.1.2 PyYAML >= 5.3.1 requests >= 2.23.0 scipy >= 1.4.1 torch >= 1.7.0 torchvision >= 0.8.1 tqdm >= 4.41.0 pandas >= 1.1.4 seaborn >= 0.11.0 YOLOv5 = GPL-3.0 license

A.3 Hiperparâmetros do melhor modelo

lr0: 0.01, lrf: 0.1, momentum: 0.937, weight_decay: 0.0005, warmup_epochs: 3.0, warmup_momentum: 0.8, warmup_bias_lr: 0.1, box: 0.05, cls: 0.5, cls_pw: 1.0, obj: 1.0, obj_pw: 1.0, iou_t: 0.2, anchor_t: 4.0, fl_gamma: 0.0, hsv_h: 0.015, hsv_s: 0.7, hsv_v: 0.4, degrees: 0.0, translate: 0.1, scale: 0.5, shear: 0.0, perspective: 0.0, flipud: 0.0, fliplr: 0.5, mosaic: 1.0, mixup: 0.0, copy_paste: 0.0