Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e de Computação

Marcos Paulo Almeida Dal Maso - 173700

Curso: Eng. Elétrica Integral - 11

Detecção de pedestres e placas de carros com Deep Learning

Orientador(a): José Mário de Martino

Campinas

02/2020

AGRADECIMENTOS

Agradeço ao meu orientador José Mário de Martino por aceitar conduzir o meu trabalho de pesquisa, assim como a Marcelo Pederiva pelo apoio no desenvolvimento do trabalho e apoio nas pesquisas.

A todos os meus professores do curso de Engenharia Elétrica da Universidade Estadual de Campinas e a todos os demais professores que fizeram parte de minha formação acadêmica pela excelência da qualidade técnica de cada um.

E por fim, aos meus pais, familiares e todos os amigos que sempre estiveram ao meu lado me apoiando ao longo de toda a minha trajetória.

RESUMO

Com o crescente desenvolvimento de tecnologias de inteligência artificial e de reconhecimento de imagens, as pessoas cada vez mais têm sua privacidade e suas informações expostas sem ao menos tomarem conhecimento. Desse modo, considerando uma situação de desenvolvimento de um carro autônomo, em que é necessário usar uma câmera para detecção de objetos, placas de veículos e rostos de pessoas, se tornam informações pessoais potencialmente suscetíveis à violação. Este trabalho aborda esse tema de segurança de dados, utilizando de técnicas e algoritmos de aprendizado profundo (deep learning), mais especificamente o algoritmo You Only Look Once (YOLO) para reconhecer esses objetos em tempo real, e borrá-los com o intuito de tornar o conteúdo da placa e o rosto das pessoas no ambiente irreconhecíveis.

Neste trabalho serão descritos os processos: de extração das características utilizadas na etapa de classificação, de coleta dos dados, de criação da base de dados final, do desenvolvimento do algoritmo que utiliza redes neurais para classificar os objetos, além do processamento e filtro desses objetos para a garantia de privacidade. Ao final do trabalho será mostrado um comparativo evidenciando os resultados alcançados.

Palavras-chave: Aprendizado de máquina, Inteligência artificial, YOLO, Detecção de objetos, Redes Neurais

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

Carros autônomos têm sido alvo de grande estudo em diversas universidades, centros de pesquisa e indústrias nos últimos anos. Os estudos buscam aprimorar a direção autônoma de tal forma que não seja necessário a intervenção humana no controle do carro (DICKMANNS, 2003).

Com o avanço da tecnologia autônoma, a Sociedade de Engenheiros Automotivos (The Society of Automotive Engineers, SAE) propôs a categorização de níveis de automação para os veículos (SAE, 2018). O modelo SAE J3016 define 6 níveis de automação para carros, no qual, o nível 0 representa nenhuma automação, os níveis 1 a 3 sugerem que o motorista seja o principal controlador do veículo enquanto algumas aplicações autônomas são parcialmente utilizadas. Por fim, o nível 4 e 5 são definidos quando o veículo é controlado sem a assistência de um motorista.

Dessa forma, níveis mais altos de automação, exigem mais sensores para controlar um veículo de forma segura. Por exemplo, um veículo completamente autônomo, necessita não só reconhecer a própria velocidade através de sensores de odometria e Unidades de Medição Inercial (Inertial Measurement Unit - IMU), como também se localizar mundialmente, com GPS, e localmente com ultrassom, LIDAR e câmeras.

A utilização de câmeras para aplicações de carros autoguiados tem sido de grande relevância, pois além de ser um sensor de baixo custo, fornecem informações relevantes como linhas de trânsito (McCALL; TRIVEDI, 2004), sinais de trânsito, cores do semáforo e pedestres (KONIG et al,. 2017). Além do mais, as câmeras representam a visão humana de um motorista, o principal sensor que é utilizado para dirigir durante muitos anos (KHALIFA et al,. 2008).

Atualmente o uso de Aprendizado de Máquina (Machine Learning) tem sido uma das principais soluções para problemas de veículos autônomos, principalmente na área de percepção do veículo. Técnicas de aprendizado permitem o treinamento de máquinas para classificar e localizar objetos em uma imagem, de forma rápida e precisa. Este processo é conhecido como Detecção de Objetos (Object Detection).

Atualmente, as abordagens mais promissoras para a detecção de objetos são

baseadas em redes neurais (Neural Networks) que simulam o processo de funcionamento dos neurônios no cérebro humano, de maneira matemática, permitindo assim desenvolver um processo de treinamento e aprendizagem por parte da máquina (EVERINGHAM et al., 2010).

1.2 Justificativa

Apesar das câmeras fornecerem ao veículo inúmeras informações que o ajude a interpretar o ambiente, ela também grava informações que infringe a privacidade das pessoas na rua. Por exemplo, a captura de placas de veículos e de rostos de pedestres nas imagens.

Para contornar este problema, este projeto busca detectar placas de carros e pedestres que aparecem nas imagens, para futuramente apagar ou distorcer essas informações em tempo real impedindo o reconhecimento e, assim, protegendo sua privacidade. A solução visa manter o uso de câmeras para auxiliar a direção autônoma e ao mesmo tempo proteger a privacidade das pessoas.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo deste trabalho é configurar e avaliar um sistema baseado na arquitetura YOLO que seja capaz de utilizar técnicas de aprendizado de máquina para detectar placas de veículos e pedestres, a fim de garantir a privacidade dos mesmos.

1.3.2 Objetivos Específicos

Abaixo são citados os objetivos específicos que este trabalho visa alcançar para atingir o objetivo geral.

- Estabelecer o dataset a ser utilizado para o estudo.
- Treinar uma rede neural a partir do YOLO uma rede neural para reconhecer placas de veículos e pedestres.
- Avaliar a precisão obtida e melhorar os hiperparâmetros do treinamento.
- Desenvolver uma forma de garantir a privacidade dos objetos tratados.

2. REVISÃO BIBLIOGRÁFICA

Neste capítulo serão abordados trabalhos correlatos à pesquisa, em seguida diversos conceitos necessários para o entendimento do trabalho desenvolvido.

2.1 Conceitos Básicos

2.2.1 Aprendizagem de máquina

O aprendizado de máquina é a área da inteligência artificial que evoluiu do estudo de reconhecimento de imagens e da teoria do aprendizado computacional, sendo responsável pelos métodos e algoritmos que possuem a capacidade de aprender com informações obtidas através de dados de entrada. O conceito desta forma de aprendizado se baseia no próprio processo cognitivo humano. Portanto, ela parte do princípio de que através da sua tentativa de previsão e consequente erro pode-se aprimorar a resposta dada, ou seja, o aprendizado é adquirido através da experiência. Os algoritmos desta área podem ser divididos em diferentes categorias, dentre elas pode-se citar o aprendizado não supervisionado e o aprendizado supervisionado, o qual foi utilizado neste trabalho.

2.1.2 Aprendizado Supervisionado

Os algoritmos de aprendizado supervisionado são modelos que aprendem a associar alguma entrada a alguma saída, dado um conjunto de treinamento de exemplos de entradas e seus respectivos resultados. Em muitos casos, esses resultados podem ser difíceis de coletar automaticamente e devem ser fornecidos por um “supervisor” humano, mas o termo ainda se aplica mesmo quando o conjunto de treinamento for coletado automaticamente (GOODFELLOW; BENGIO; COURVILLE, 2015).

Para efetuar o aprendizado, o algoritmo sabe previamente qual é a resposta esperada de um sistema, e então dado uma certa entrada ele compara seu resultado de saída com o valor previamente estipulado como correto, para que assim, o mesmo se ajuste em busca de minimizar o erro de sua resposta.

Existem diversos algoritmos que seguem esta estratégia, como por exemplo: Máquina de Vetores de Suporte (Support Vector Machine - SVM), Regressão Linear, Árvores de decisão, K-Vizinhos Próximos, Redes Neurais, entre outros modelos.

Ressalta-se que no presente trabalho foi utilizado o modelo de redes neurais, já que é o mais apropriado e utilizado para o tratamento de dados provenientes do reconhecimento de imagens.

2.1.3 Redes Neurais

As redes neurais artificiais (RNAs) foram primeiramente concebidas em 1943 por Warren McCulloch, um psicólogo norte-americano, e o matemático Walter Pitts, que formularam o primeiro modelo de um neurônio como uma função no qual a sua saída era a soma de valor das diversas entradas.

Mais tarde, em 1958, Frank Rosenblatt, um psicólogo norte-americano, desenvolveu um modelo promissor denominado perceptron, que simula o processo de reconhecimento de padrões através de uma estrutura de unidades sensoriais conectadas a uma única camada de neurônios. Esta estrutura era constituída de 3 camadas: uma camada inicial responsável por receber as entradas de um meio externo, uma camada intermediária de pesos ajustáveis e por fim uma camada de saída, que possui a resposta do modelo, sendo o tipo mais simples de rede neural feedforward: um classificador linear (ROSENBLATT, 1958).

Apesar da grande inovação trazida com o surgimento do perceptron no ramo da inteligência artificial, contudo, o mesmo possuía limitações em distinguir somente padrões que fossem linearmente separáveis. Entretanto, novos avanços de redes neurais artificiais na classificação de padrões só voltaram a surgir após a descrição do algoritmo backpropagation, o qual provou definitivamente que as redes neurais podem resolver os problemas ditos não-linearmente separáveis.

De forma breve, neste modelo tem-se que os dados de entrada são inseridos na camada de entrada e em seguida são multiplicados pelos pesos aleatoriamente gerados. A saída de cada neurônio da camada de entrada pode ser descrita pela Equação 2.1, onde Y é a saída do neurônio k, o qual é dado por uma combinação linear onde X são as n entradas e W os n pesos atribuídos.

$$Y_k = \sum_{i=1}^{n} W_{ki} * X_{ki} \tag{2.1}$$

Antes de avançar, estas saídas passam por uma espécie de filtro denominada função de ativação. As funções de ativação geralmente são funções não lineares,

cujo objetivo é possibilitar com que as saídas da rede neural possam ser valores contínuos diferentes de 0 e 1.

Após passar por esta função de ativação, a resposta de um neurônio da camada anterior passa a ser a entrada do neurônio da camada posterior. Este processo se repete até que se alcance a última camada onde os neurônios são modelados de forma com que se obtenha a resposta do sistema no formato desejado. Nesta última camada normalmente são utilizadas funções específicas para avaliar o resultado, como o caso da função softmax, descrita na equação 2.2, o qual representa a probabilidade da variável x ser de uma das classes definidas, sendo assim especialmente útil em nossa tarefa de classificação de objetos.

$$P(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j)} \tag{2.2}$$

O aprendizado de uma rede neural é obtido ajustando os pesos de cada neurônio, que ao final de cada iteração são ajustados através do algoritmo backpropagation, onde o erro obtido através da camada de saída se propaga por todas as camadas anteriores fazendo com que os pesos de cada neurônio possam ser ponderados baseados neste erro. Com base nesse ajuste um novo ciclo de backpropagation se repete, até que o erro obtido seja o menor possível. Um exemplo da estrutura de uma rede neural pode ser visto na Figura 1.

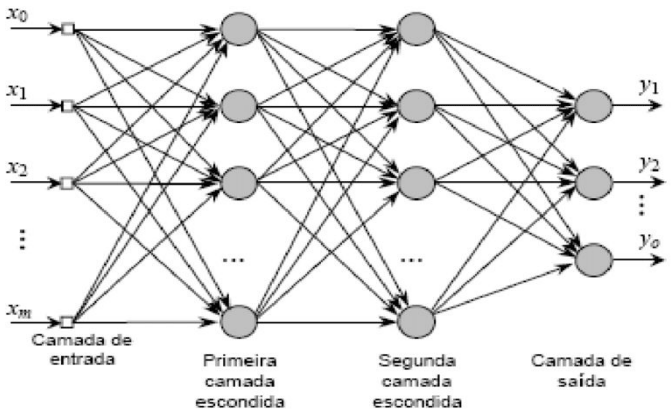

Figura 1 - Modelo de uma rede neural.

Esta breve descrição relata o funcionamento de uma iteração do processo de aprendizado de uma rede neural, porém, para que ela possa representar e se adaptar de maneira eficiente ao modelo proposto são necessárias diversas iterações e camadas. Desse modo uma importante alteração desse modelo são as redes

neurais convolucionais (CNN) que são simplesmente redes neurais que usam convolução no lugar da multiplicação geral da matriz em pelo menos uma de suas camadas (GOODFELLOW; BENGIO; COURVILLE, 2015).

Assim, a partir dessa estrutura diversas redes neurais foram desenvolvidas como o RetinaNet, Faster R-CNN, SSD513 entre outros, entretanto um modelo que se sobressai é o YOLO (You Only Look Once).

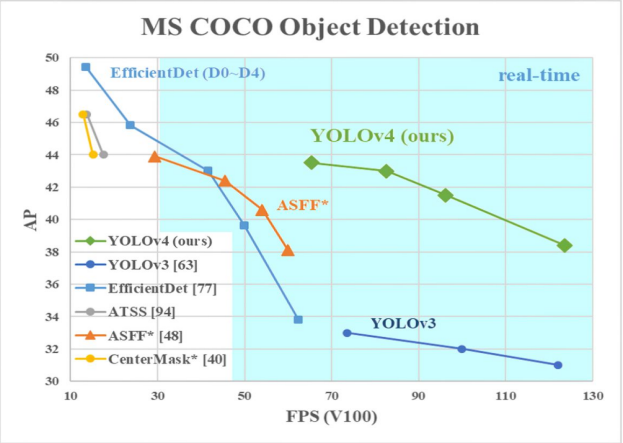

Figura 2 - Comparação de desempenho entre o YOLOv4 e outras arquiteturas.

Nota-se pela Figura 2, que a versão 4 do YOLO tem um desempenho superior de precisão (AP) em relação a sua versão anterior e velocidade de processamento (FPS) superior a outros métodos sem perder significativamente em precisão (BOCHKOVSKY; CHIEN-YAO; HONG-YUAN, 2020).

2.1.4 YOLO

Com modelos de detecção apresentando uma ótima precisão e uma resposta em tempo real, a resolução de problemas que exigem esse desempenho se torna viável. Assim, o YOLO se destaca como uma das estratégias mais relevantes para extração e detecção de informações de uma imagem, ou de vídeo.

O YOLO foi proposto em 2016, em sua primeira versão, por Joseph Redmon (REDMON; FARHADI, 2016). Este modelo foi apresentado em duas arquiteturas, Regular YOLO e Tiny YOLO, uma apresentando a melhor precisão dos modelos da época e a outra, com uma arquitetura menor, apresentando o menor tempo de resposta, respectivamente. No ano seguinte, Redmon apresentou a segunda versão do YOLO. Esta, por sua vez, apresentava uma detecção mais rápida, precisa e com

a capacidade de detectar vários objetos simultaneamente (REDMON; FARHADI, 2017). Em 2018, YOLO teve sua terceira versão, com algumas alterações e pequenos ganhos de desempenho comparado ao ano anterior (REDMON; FARHADI, 2018).

Por fim, em 2020, Alexey Bochkovskiy implementou pequenas alterações em diversos fatores do YOLOv3, desde estratégias de aumento de dados (Data Augmentation) até mudanças nos módulos de detecção e métodos de pós-processamento. Com isso, Bochkovskiy apresentou um modelo, YOLOv4, com uma performance destacada dentre as demais (BOCHKOVSKY; CHIEN-YAO; HONG-YUAN, 2020).

O YOLO é baseado em uma única Rede Neural Convolucional (CNN). é possível notar pela Figura 3,A CNN divide uma imagem de entrada em uma grade de SxS células. Cada uma dessas células é responsável por prever regiões delimitadoras e cada região possui B caixas delimitadoras, cada uma com sua pontuação de confiança e pontuação de C classes para detecção e mais 5 parâmetros. Contudo, sua pontuação de confiança não diz nada sobre que tipo de objeto está na caixa, apenas se o formato da caixa é bom. Sendo que uma caixa delimitadora descreve o retângulo que envolve um objeto (REDMON; FARHADI, 2016).

Além disso, esse um fator de “5” que vem do fato que cada caixa prevista tem 5 componentes [x, y, w, h, pontuação de confiança]. Onde (x, y) denota o centro da caixa em relação à célula da grade correspondente, enquanto (w, h) denota o comprimento e altura em relação à imagem inteira. Em geral, as dimensões da caixa delimitadora são previstas aplicando uma transformação de espaço logarítmico à saída e, em seguida, multiplicando com uma âncora, ou seja, caixas delimitadoras padrão pré-definidas (KATHURIA, 2018).

Assim a dimensão do kernel gerado é o seguinte:

$$[S \times S \times (B \times (5 + C))] \tag{2.3}$$

E o número de caixas delimitadoras:

$$[S \times S \times B] \tag{2.4}$$

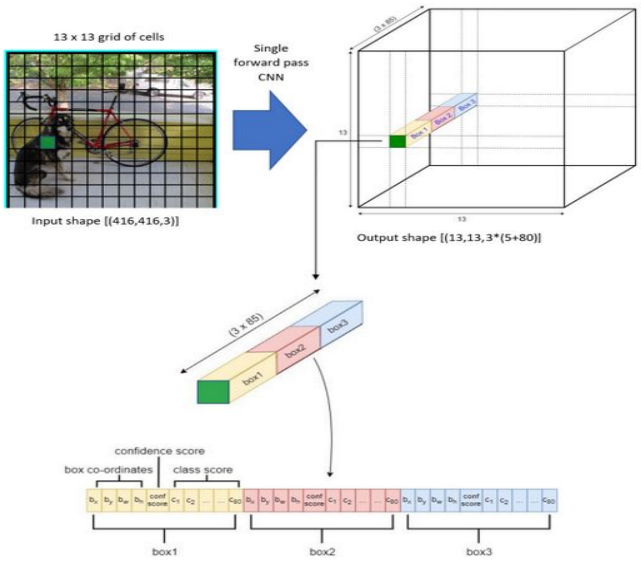

Figura 3 - Parâmetros de detecção em cada célula do YOLO.

O YOLOv1 tem como parâmetros S=7 e B=2, assim ele é capaz de prever 98 caixas delimitadoras simultaneamente. O YOLOv2 tem S=13 e B=5, prevendo 845 caixas delimitadoras simultaneamente. Já o YOLOv3 faz previsões em 3 escalas diferentes, reduzindo o tamanho da imagem original, no caso de 416x416, por fatores de 32, 16, 8 assim tendo como resultado final 3 tipos de imagens com S=13, S=26, S=52 e B=3 para cada um deles, assim ele prevê 10647 caixas delimitadoras simultaneamente. Essas outras escalas permitiram, por exemplo, o aperfeiçoamento do YOLOv3 na detecção de imagens menores. A Figura 4 demonstra melhor como essas caixas delimitadoras são representadas para diferentes dimensões.

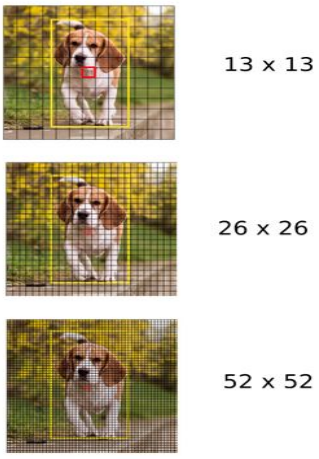

Figura 4 - Predição de caixas delimitadoras em diferentes escalas no YOLOv3

A arquitetura do YOLOv3, ilustrada na Figura 5, consiste em 53 camadas de

convolução que reduzem o tamanho da imagem de entrada através da técnica de pooling para extração de recursos. Uma estrutura típica do YOLO treinado com a base de dados COCO tem para a camada de saída o tamanho de [SxSx255] sendo S, da equação 2.3, a divisão de células do YOLO. Além disso, como já dito, há 3 caixas delimitadoras para cada célula, o que representa o parâmetro B, e 80 classes equivalentes ao parâmetro C. Entretanto, como no presente trabalho o número de classes é diferente esse valor final é alterado de 255 para 21. Deve-se atentar também ao fato que a versão reduzida do YOLO ou Tiny YOLO é uma rede relativamente menor, constituída de somente 9 camadas de convolução, mas que possui os mesmos parâmetros anteriores (REDMON; FARHADI, 2018).

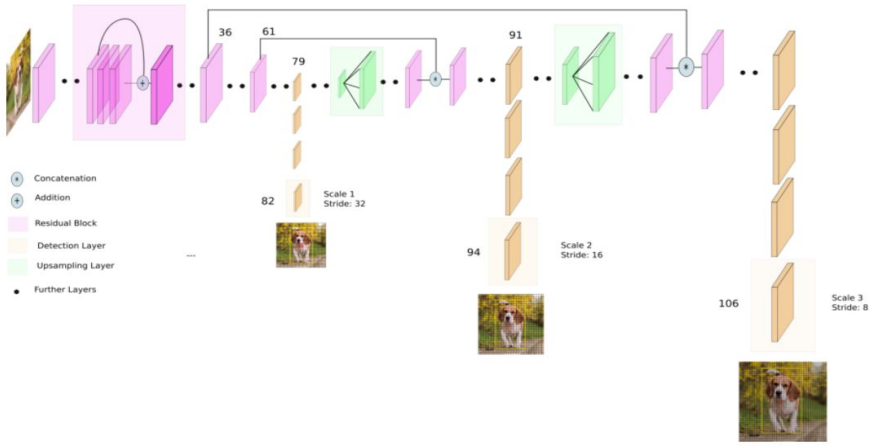

Figura 5 - Representação esquemática da rede neural do YOLOv3.

Para classificação e detecção do objeto em si durante a execução dos testes, a equação 2.5 demonstra que, o modelo designa cada caixa e sua classe específica com uma pontuação de confiança, e então multiplica por sua respectiva probabilidade condicional de classe. Essa pontuação avalia a probabilidade da classe aparecer na caixa e com a multiplicação pela interseção sobre união (IoU) avalia o quão precisa as coordenadas da caixa são.

$$P_r(Classe_i|Objeto)xP_r(Objeto)xIoU_{pred.}^{verdade} = P_r(Classe_i)xIoU_{pred.}^{verdade} \qquad (2.5)$$

Por fim dado que a probabilidade obtida é maior do que os limiares pré-estabelecidos então o objeto é tido como detectado, caso contrário a região permanece sem nenhuma caixa delimitadora, um exemplo dessa sequência pode ser visto na Figura 6.

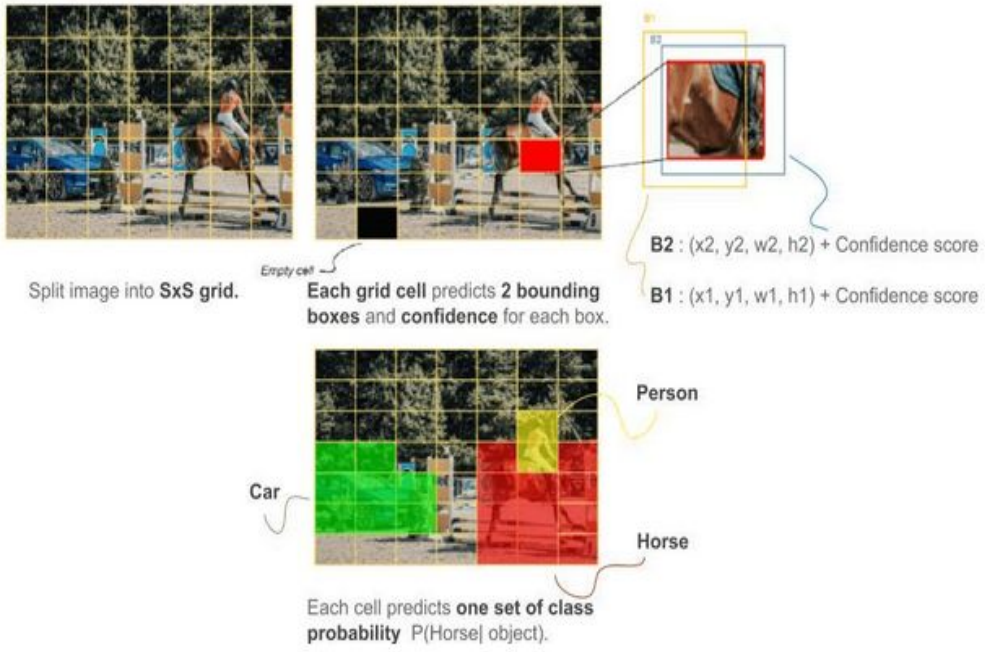

Figura 6 - Representação da predição de classes pelo YOLO.

Assim, o YOLO prevê simultaneamente várias caixas delimitadoras e suas probabilidades para essas classes. Cada caixa delimitadora tem pontuações que refletem a probabilidade da caixa prevista conter um objeto (pontuação de classe), bem como sua precisão de posicionamento (pontuação de confiança).

Dado a estrutura dessa rede neural, resta somente ajustar os pesos e otimizar a detecção ao longo do treinamento. A Figura 7 mostra a equação usada pelo YOLO que usa erro quadrático de soma de três partes (SSE) para tentar diminuir os erros ao longo do treinamento gerando a chamada função de perdas (Loss Function). Desse modo, a função é dada pelo termo de perdas de localização que penaliza as coordenadas da caixa delimitadora $[x_i, y_i, w_i, h_i]$, termo de perdas de classificação que penaliza probabilidades condicionais de classe $[p_i(c)]$, além do termo de perdas de confiança que penaliza a presença ou não de objetos $[C_i]$. Esses termos são ainda expandidos em um somatório para cada caixa delimitadora e para cada célula (CHUNG, 2019).

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$
$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$
$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$
$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Figura 7 - Cálculo da função de perdas do YOLO.

Observa-se que essa é a formulação para a primeira versão. Nela os últimos três termos no YOLO são os erros quadrados, enquanto no YOLO v3, eles foram substituídos por termos de erro de entropia cruzada. Ou seja, a confiança do objeto e as previsões de classe no YOLO v3 agora são previstas por meio de regressão logística.

2.1.5 YOLO versão 4

O YOLOv4 foi desenvolvido por três desenvolvedores Alexey Bochkovskiy, Chien-Yao Wang e Hong-Yuan Mark Liao e lançado em abril de 2020. Essa nova versão é uma melhoria importante do YOLOv3, e com a implementação de uma nova arquitetura no “Backbone” e as modificações no “Neck” o mAP (precisão média) melhorou em 10% e o número de FPS (quadros por segundo) em 12%. Além disso, ficou mais fácil treinar essa rede neural em uma única GPU (RUGERY, 2020).

Um detector moderno geralmente é composto de duas partes, um “backbone” pré-treinado na ImageNet e um “head” que é usado para prever classes e caixas delimitadoras de objetos. O “head” pode ser um detector de objetos de um estágio, a exemplo do YOLO ou de dois estágios, como o R-CNN. Pode haver também um estágio intermediário entre o “backbone” e o “head”, o chamado “neck”, e essas camadas são geralmente usadas para coletar “features maps”, que são informações essenciais da detecção obtidas pelo resultado de um filtro aplicado a uma camada, que no caso do YOLOv4 vem de diferentes estágios, e não somente do estágio anterior. (BOCHKOVSKY; CHIEN-YAO; HONG-YUAN, 2020).

O principal objetivo do backbone é de extrair as informações essenciais, que permitem identificar o objeto, sendo sua seleção um passo fundamental para melhorar o desempenho geral da detecção. A arquitetura do “backbone” do YOLOv4 é baseada no “CSPDarknet53” (WANG, 2020) que contém dois métodos principais para melhorar as detecções.

O primeiro deles são os métodos do tipo “Bag of freebies” são o conjunto de métodos que aumentam o custo do treinamento, enquanto mantém o custo da inferência baixo. Exemplo disso é o uso de aumento de dados (Data Augmentation) para aumentar a variabilidade de uma imagem e assim melhorar a generalização do treinamento do modelo. Distorção fotométrica para criar novas variedades da mesma imagem ajustando o brilho, matiz, contraste, saturação e ruído. Distorção geométrica para girar, inverter, redimensionar ou cortar a imagem. Isso para citar alguns dos principais métodos (RUGERY, 2020).

Além disso, existem os métodos “Bag of specials" que aumentam um pouco os custos de inferência mas podem melhorar significativamente a precisão da detecção de objetos. Um exemplo disso é o uso da função de ativação “Mish” que evita a saturação, que geralmente causa lentidão no treinamento devido a gradientes quase nulos. Antes com o uso da função “ReLU” um grande viés ou “bias” negativo causava essa saturação fazendo com que os pesos não fossem atualizados durante a fase de retropropagação (backpropagation), tornando os “neurônios” inoperantes para previsão (RUGERY, 2020).

Já o “neck” é formado pelo bloco Spatial Pyramid Pooling Layer (SPP), que remove a restrição de tamanho fixo da imagem de entrada da rede, ou seja, o SPP permite que uma rede neural convolucional (CNN) não precise de uma imagem de entrada de tamanho fixo, permitindo assim obter as características essenciais do objeto ou “feature maps” de modo mais preciso. (ZHANG et al., 2015).

E também é adicionado o Path Aggregation Network (PANet) como o método de agregação de parâmetros de diferentes níveis “backbone” para diferentes níveis de detector. Através do “PANet”, é solucionado o problema de perder as informações localizadas no começo das camadas que podem ser necessárias para ajustar a previsão, conforme avançamos através das camadas da rede neural. Isso é feito introduzindo uma arquitetura que permite uma melhor propagação das

informações entre as camadas de “baixo para cima” ou de “cima para baixo” (LIU et al., 2018).

Resumindo, o YOLOv4 usa como “backbone” o “CSPDarknet53”, como “neck” o SPP e PAN e como “head” o YOLOv3, que é um detector de 1 estágio do tipo “dense prediction”. E portanto, pela Figura 8, podemos visualizar o esquema entre estágios da arquitetura do YOLOv4.

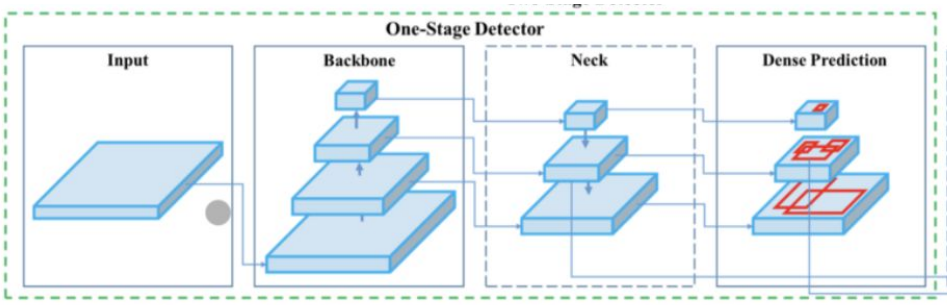

Figura 8 - Representação esquemática dos componentes do YOLOv4.

Destaca-se que o YOLO segue em pleno desenvolvimento e novas atualizações como as versões YOLOv5 e o “PP-YOLO” já estão sendo projetadas para melhorar ainda mais a versão atual.

2.1.6 Open CV

O Open Source Computer Vision Library, popularmente conhecido como OpenCV, é uma biblioteca multiplataforma, para o desenvolvimento de aplicações de visão computacional. Com módulos de processamento de imagem, estrutura de dados, álgebra linear e GUI (Interface Gráfica do Usuário) ele permite criar filtros de imagem, calibração de câmera, reconhecimento de objetos, análise estrutural entre outras aplicações com processamento em tempo real de imagens. O código para executar essa biblioteca é feito na linguagem de programação python.

2.1.7 Blur

Para proteger a privacidade das informações mais uma etapa de ajustes da imagem precisa ser feito. A próxima fase é o processamento dessas imagens em tempo real, e detecção real dos objetos aplicando-se um efeito conhecido como “Blur” que nada mais é do que borrar a imagem, ou seja, reduzir os detalhes dos objetos reconhecidos para que não seja possível identificá-los, garantindo assim sua privacidade.

Em processamento de imagem, um filtro comumente utilizado é a convolução entre um kernel, uma pequena matriz de convolução, e uma imagem. Essa operação é usada para desfocar, aumentar a nitidez, detectar bordas e muito mais. Para a operação de blur, um exemplo de matriz utilizada é visto na Figura 9 (Shapiro, 2001)

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Figura 9 - Exemplo de Gaussian Blur kernel.

Depois de definida a função do filtro, cada pixel da imagem que possui um número associado a ele passa por um processo de convolução com o kernel. Para cada bloco 3x3 de pixels da imagem de entrada, multiplica-se cada pixel desse bloco pelo número na posição correspondente do kernel e, em seguida, é feita a soma. Essa soma se torna um novo pixel na imagem de saída. O processo é repetido até que todos os pixels da imagem de entrada passem pelo filtro. A Figura 10, a seguir, representa visualmente esse processo (Shapiro, 2001).

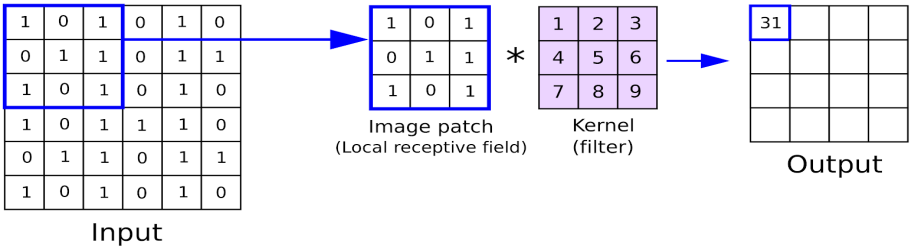

Figura 10 - Implementação de um kernel aplicado a uma imagem.

2.1.8 Google Colab

O ambiente do Colab permite a escrita e execução de código em python por meio do navegador e é especialmente útil para aprendizado de máquina e análise de dados. Mais tecnicamente, o Colab é um serviço hospedado de notebook Jupyter que não requer configuração para uso, ao mesmo tempo que fornece acesso gratuito a recursos de computação, incluindo GPUs. Porém os recursos do Colab não são garantidos e ilimitados, e a capacidade de uso dos recursos variam às vezes , além disso, toda vez que o ambiente é fechado precisa ser reiniciado para poder ser utilizado novamente, o que pode causar perda de arquivos.

3. MATERIAIS E MÉTODOS

3.1 Banco de Dados

O primeiro procedimento para realização do treinamento é formar um banco de dados com uma série de imagens e mapear os objetos a serem detectados em cada uma dessas imagens. Para este trabalho foi utilizado um database já existente com esses dados denominado “UNICAMP_PRPI”.

A partir daí foram selecionadas 3 configurações do YOLO para realização de comparações de desempenho entre outros testes. São eles: Tiny YOLO versão 3, Tiny YOLO versão 4 e Regular YOLO versão 4. É importante frisar que será dado um enfoque maior no Tiny YOLO versão 4, por se tratar da versão teoricamente mais apropriada para testes de rápido processamento, mas também com boa eficácia, ideais para aplicações em tempo real como na detecção de objetos através de câmeras em veículos.

3.2.1 Arquivos

Para configurar o ambiente a ser usado no Google Colab e realizar os treinamentos, é necessário primeiramente obter um modelo pré-treinado, por isso é preciso baixar e configurar o ambiente para o funcionamento do Darknet, depois disso pode-se alterar alguns arquivos para que o modelo personalizado possa ser treinado.

O banco de dados de imagens precisa ser separado em dois arquivos de texto: Um para testes e um para treino. As imagens de treino serão usadas para ajustar os pesos da rede neural, servindo como entrada do sistema. As imagens de teste são usadas para testar se o algoritmo consegue performar adequadamente com novas imagens, evitando o efeito indesejado de sobre-ajuste ou “overfitting”, visto na Figura 11, que ocorre quando o modelo se ajusta muito bem ao conjunto de dados anteriormente observado, mas se mostra ineficaz para prever novos resultados. Assim, através de um script feito em python são gerados esses dois arquivos de texto com a lista de imagens associadas.

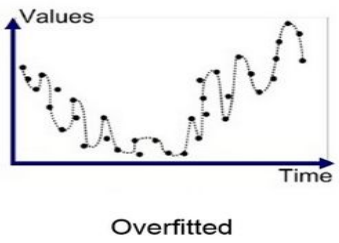

Figura 11 - Exemplo de interpolação com sobre-ajuste.

Como existem somente duas classes a serem detectadas é criado um outro arquivo de texto chamado “obj.names” com o nome delas. Dentro dele, é nomeado as classes “Plate” para as placas de veículos e “Person” para a detecção de pessoas.

Para que o treinamento seja feito é preciso ainda especificar o caminho dos arquivos a serem executados. Assim, em um novo arquivo chamado “obj.data” é criado. Nele escreve-se primeiro o número de classes que no caso são 2, depois o caminho dos arquivos de texto de teste e treinamento, previamente realizados, o caminho do arquivo com o nome das classes, “obj.names”, e também uma pasta de backup para salvar os arquivos “.weights”, com a configuração dos pesos das diversas camadas do YOLO, durante o processo de iteração do algoritmo, para que caso o processo seja interrompido, ele possa voltar da iteração mais recente. Observa-se que esses arquivos com os pesos são modificados para serem obtidos após 100 iterações.

Por fim, o arquivo mais relevante para modelar o treinamento é o arquivo “.cfg” que contém especificações mais específicas de como serão realizados os treinamentos e também a configuração da rede neural do YOLO, sendo assim esse é o único arquivo a ser alterado para cada versão diferente do YOLO testada, ou para realizar quaisquer modificações de aperfeiçoamento do treinamento durante o presente projeto especificado. Esse arquivo “.cfg” é obtido do Github original do projeto darknet para cada versão a ser testada.

3.2 Treinamento

Com os arquivos definidos, só resta baixar os pesos convolucionais pré-treinados. Na versão do Tiny YOLOv3 ele é denominado “darknet53.conv.74”. Na versão Regular YOLOv4 ele é denominado por “yolov4.conv.137” e na versão Tiny YOLO v4 ele é denominado “yolov4.conv.29”.

Inicialmente o primeiro modelo testado foi o Tiny YOLOv3. Para esse treinamento o arquivo “yolov3-tiny.cfg” foi alterado, de modo que os seguintes parâmetros foram alterados da versão original do YOLO para o treino:

- batches=64
- subdivisions=64
- width=608
- height=608
- max_batches=4000
- steps=3200,3600
- filters=21
- classes=2

De modo geral, o parâmetro “batches” define o número de amostras, nesse caso imagens, que serão processadas em cada iteração (batch) de modo a reduzir as perdas. Concluída uma iteração de “batches” os gradientes são atualizados.

O parâmetro “subdivisions” define o número de mini “batches” em um lote, servindo assim para o processamento em paralelo da GPU. Nesse caso os 64 “batches” são divididos 64 vezes, resultando em 1 imagem processada por “mini-batch”, assim o processo deve se repetir 64 vezes para que 1 iteração possa ser completa.

O parâmetro “width” e “height”, configura o YOLO para que cada imagem seja redimensionada para o tamanho da rede durante o treinamento e detecção. O parâmetro “max batches” define o número total de iterações do treinamento (batches) e é modificado para 2000 vezes o número de classes. O parâmetro “steps” define que nas interações de número 3200 e 3600, a taxa de aprendizagem será multiplicada pelo fator de escala e devem corresponder a 80% e 90% do valor de “max batches”. O parâmetro “classes” define o número de classes a serem detectadas. O parâmetro “filters” define quantos “convolutional kernels” tem nas camadas, e tem como origem a equação 2.3.

$$filters = (número\ de\ classes + 5) * 3 = (2 + 5) * 3 = 21 \qquad (3.1)$$

O segundo modelo testado foi o Tiny YOLOv4. Para esse treinamento o arquivo “yolov4-tiny.cfg” foi alterado, de modo que os seguintes parâmetros foram alterados da versão original do YOLO para o treino:

- batches=64
- subdivisions=64
- width=608
- height=608
- max_batches=4000
- steps=3200,3600
- filters=21
- classes=2

O último modelo testado foi o Regular YOLOv4. Para esse treinamento o arquivo “yolov4.cfg” foi alterado, de modo que os seguintes parâmetros foram alterados da versão original do YOLO para o treino:

- batches=64
- subdivisions=32
- width=608
- height=608
- max_batches=4000
- steps=3200,3600
- filters=21
- classes=2

Nesse caso o “subdivisions” foi alterado para 32. Portanto, 2 imagens são processadas por “mini-batch” e assim o processo deve se repetir 32 vezes para que 1 iteração possa ser completa.

Pode-se ainda melhorar vários outros parâmetros da rede neural, bem como sua estrutura, porém para comparar-se as 3 configurações do modelo da maneira mais próxima possível, somente esses 8 parâmetros foram alterados.

3.3 Teste

3.3.1 Imagens e vídeos

Para realizar os testes com imagens e vídeos os parâmetros “batch” e “subdivisions” no arquivo “.cfg” devem ser alterados para 1, além disso, foi usado um threshold, que determina a probabilidade mínima para confirmar a detecção da imagem, de 50%.

Deve-se destacar que principalmente para aplicações com vídeos, esses testes precisam ser realizados com arquivos no ambiente do Google Colab, ou seja, não são feitos em operações que rodam em tempo real, sendo assim imprescindível o uso do OpenCV para que esse processamento possa ser feito.

O OpenCV já possui uma função que implementa o Gaussian Blur, para borrar os objetos, sendo necessário apenas definir o tamanho da matriz do kernel, que deve ser de um tamanho ímpar com altura e largura iguais. Enfatiza-se que é preciso filtrar, não a imagem completa, mas somente uma porção dela, por isso foi empregada a seguinte estratégia.

Primeiro, dentro do ambiente do OpenCV usa-se as informações do YOLO para identificar o objeto em questão. Cria-se então uma caixa com as informações a respeito da altura e largura do objeto e sua posição relativa na imagem. A partir disso cria-se uma borda colorida ao redor desse retângulo com cores diferentes para cada tipo de classe a ser identificada.

A imagem então passa pelo processo do filtro de Gaussian blur, o resultado é visto no lado esquerdo da Figura 12, depois é criado sobre a imagem original uma máscara com a área do objeto identificado em branco e o restante fica em preto. O resultado é visto no lado direito da Figura 12.

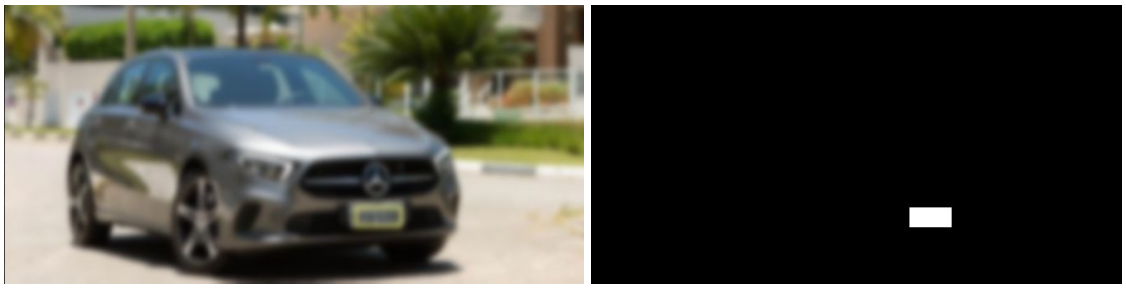

Figura 12 - Imagem com Blur à esquerda e máscara aplicada em uma imagem à direita.

Para finalizar o processo, os pixels em branco da máscara são trocados com os pixels da mesma área da imagem com o filtro de blur. Já a área em preto, permanece do mesmo jeito, assim como na imagem original.

O mesmo procedimento deve se repetir para cada objeto a ser identificado, seja ele a placa do veículo ou uma pessoa. O resultado final com a detecção de uma placa é visto na Figura 13.

Figura 13 - Resultado final da aplicação do filtro.

3.4 Métricas

3.4.1 IoU

A métrica de interseção sobre união (IoU) mede para cada caixa delimitadora a sobreposição entre a caixa prevista e a caixa com a detecção verdadeira, como na equação 3.2.

$$IoU = \frac{\text{Área de Sobreposição}}{\text{Área de União}} \tag{3.2}$$

Isso significa que, para uma previsão, podemos obter uma previsão como sendo Positivo Verdadeiro (TP) ou como Falso Positivo (FP), alterando o limite de IoU. Dessa forma, a escolha desse parâmetro influencia diretamente na realização dos cálculos dos parâmetros a seguir.

Por exemplo, digamos que para a Figura 14, se o limite de IoU dela for 0,5 e o valor de IoU para a predição da caixa com borda laranja for 0,6, então a predição é classificado como Positivo Verdadeiro (TP). Por outro lado, se IoU dela for 0,4, ela é classificada como Falso Positivo (FP) (KHANDELWAL, 2019).

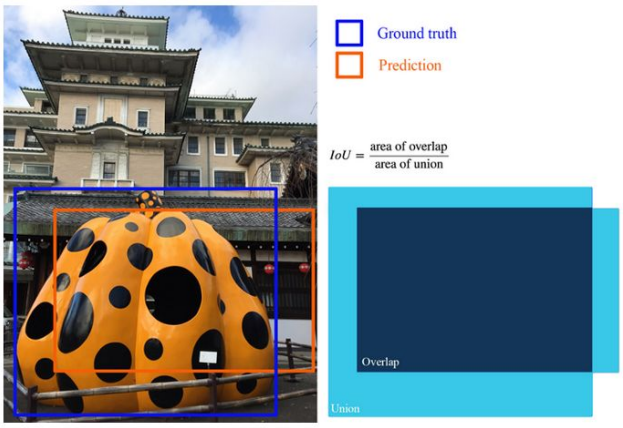

Figura 14 - Representação do IoU.

3.4.2 Precisão e Recall

A métrica de “Recall” mede o quão bem você encontra todos positivos verdadeiros, ou seja, de todos os objetos detectáveis, quantos o algoritmo reconheceu, sua fórmula é dada pela equação 3.3.

$$Recall = \frac{TP}{TP+FN} \tag{3.3}$$

A precisão por sua vez, mede a precisão de suas previsões, ou seja, a porcentagem de suas previsões está correta, sua fórmula é dada pela equação 3.4.

$$Precisão = \frac{TP}{TP+FP} \tag{3.4}$$

Sendo TP os verdadeiros positivos, ou seja, predição positiva de um objeto verdadeiro, e FN os falsos negativos, ou seja, falha na predição de um objeto que estava lá. A partir desses dois parâmetros pode-se traçar a curva precisão-recall, como visto na Figura 15 com 3 classes diferentes e que é usado para o cálculo do da precisão médio do algoritmo (mAP) (KHANDELWAL, 2019).

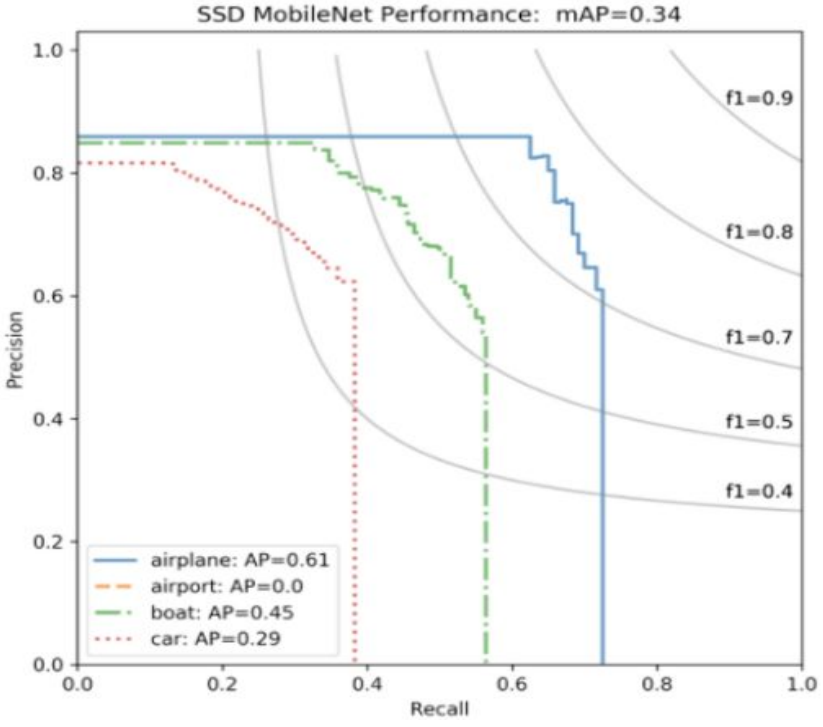

Figura 15 - Curva Precisão-Recall.

3.4.3 Precisão média (mAP)

A definição geral para a métrica de precisão média (AP) é encontrar a área sob a curva do gráfico recall-precisão.

Para isso é preciso primeiro da precisão interpolada que é a precisão média medida comumente em 11 níveis de divisão igualmente espaçados em 0, 0,1, …, 0,9, 1,0 da curva precisão-recall, assim como na Figura 16. Em seguida, soma-se os valores de precisão para cada nível de recall definido anteriormente e então divide-se pelo total de níveis estabelecidos.

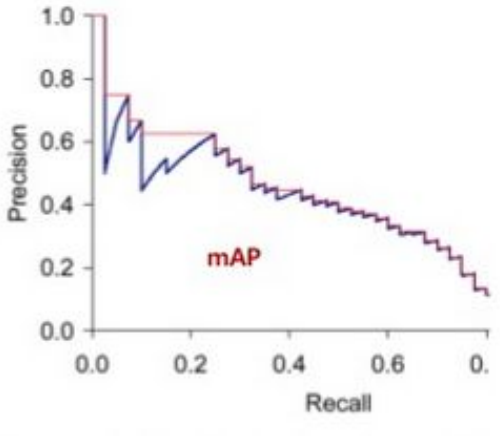

Figura 16 - Curva Precisão-Recall dividido em níveis.

Assim a equação nesse caso se resume a:

$$AP = \frac{1}{11} \sum_{r \in \{0,\ 0.1,\ldots,1\}} p_{interp(r)} \tag{3.5}$$

Já o mAP é tido como a média do AP:

$$mAP = \frac{\sum_{i \in classes} AP_I}{Núm.\ total\ de\ classes} \tag{3.6}$$

Em alguns contextos, o AP é calculado para cada classe e a média é calculada para obter o mAP. Mas em outros, eles significam a mesma coisa. No YOLO, o treinamento realizado utilizou como base para o conjunto de dados o Common Objects In Context (COCO) e nesse contexto não há diferença entre AP e mAP (KHANDELWAL, 2019).

Ressalta-se que há ainda o “F1 score” que é a média harmônica entre os parâmetros de precisão e recall.

Assim, de modo geral, se o modelo é neutro quanto a importância de falsos positivos (FPs) e falsos negativos (FNs), a pontuação “F1 score” é melhor para avaliar o modelo de melhor desempenho. Se os FPs não forem aceitáveis, a melhor métrica é a precisão. Se os FNs não forem aceitáveis, então a melhor métrica é o recall.

4. RESULTADOS E DISCUSSÃO

4.1 Perdas (Loss Function)

Após aproximadamente 4 horas para as versões Tiny e 18 horas para a versão completa e 4000 iterações para todos eles, no ambiente do Google Colab, o processo termina e um arquivo com os últimos pesos configurados são obtidos. Depois do treinamento é possível obter o gráfico de perdas e precisão média ao longo do treinamento do modelo, porém somente a partir da iteração em que o treinamento começa ou continua. Dessa maneira, depois do treinamento, obteve-se o gráfico, na Figura 17, de perdas ao longo do treinamento do modelo com Tiny YOLOv3.

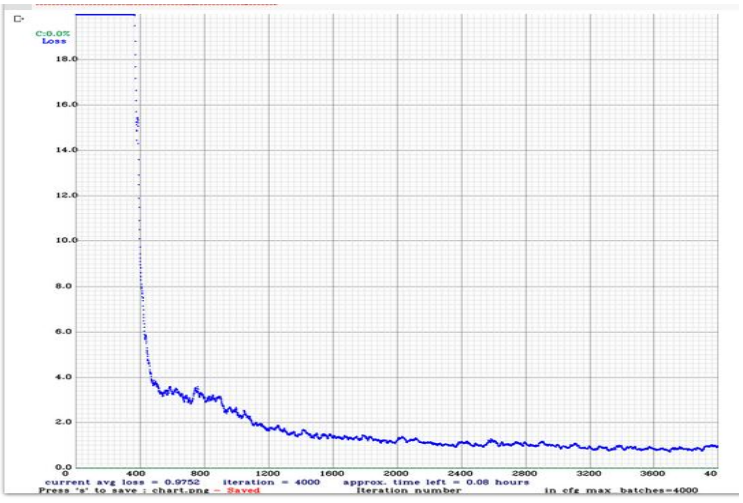

Figura 17 - Gráfico de perdas ao longo do treinamento do modelo com Tiny YOLOv3.

O perfil da curva, com redução constante das perdas, sugere que o treinamento foi bem ajustado, superando o sub-ajuste a partir de mais de 400 iterações e sem sobre-ajustes excessivos sobre o banco de dados de imagens utilizados no treinamento. Caso um efeito de sobre-ajuste ou “overfitting” ocorresse então seria obtido um perfil de gráfico mais parecido com o da Figura 18.

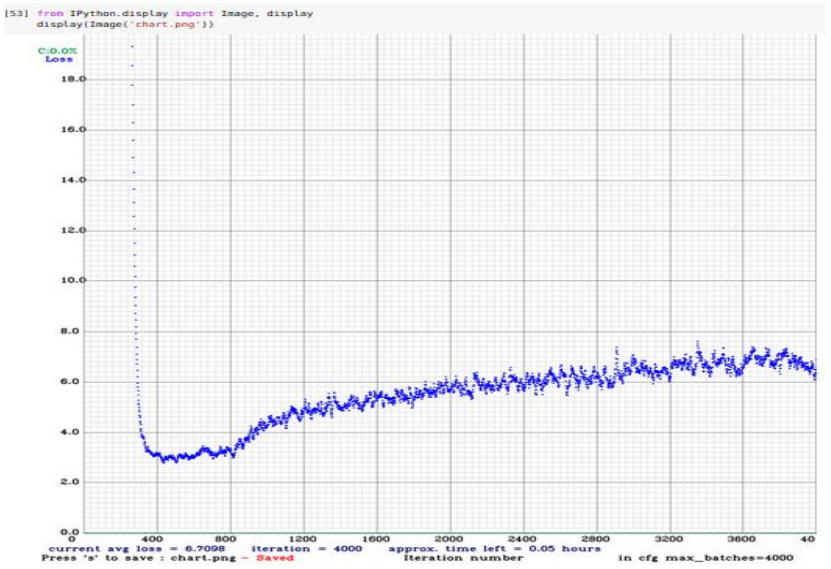

Figura 18 - Exemplo de treinamento com sobre-ajuste.

Consequentemente, caso acontecesse uma situação de erro de generalização, devido ao ajuste em excesso do modelo a partir do conjunto de dados de entrada, tornando-o ineficaz para prever novos resultados, seria observada uma grande variância do erro com o aumento de iterações do treinamento (BROWNLEE, 2019). Na Figura 19 pode-se observar a diferença teórica entre um modelo bem ajustado e um com esse erro de generalização, também chamado de “overfitting”. Observa-se também que caso o treinamento continuasse com mais de 4000 iterações para o modelo utilizado, esse erro de generalização também poderia ocorrer.

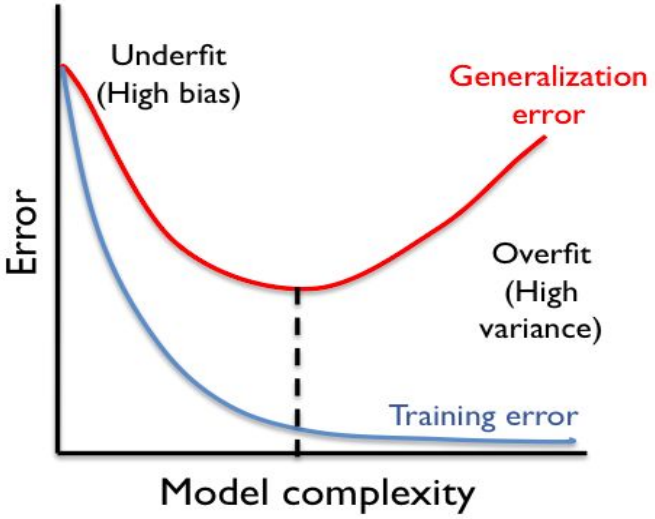

Figura 19 - Curva de complexidade do modelo vs erro para dados de treino e teste.

Já com o Tiny YOLOv4 visualiza-se na Figura 20, as perdas representado pela curva em azul diminuindo e a precisão média “mAP” representado pela curva vermelha aumentando ao longo do treinamento do modelo.

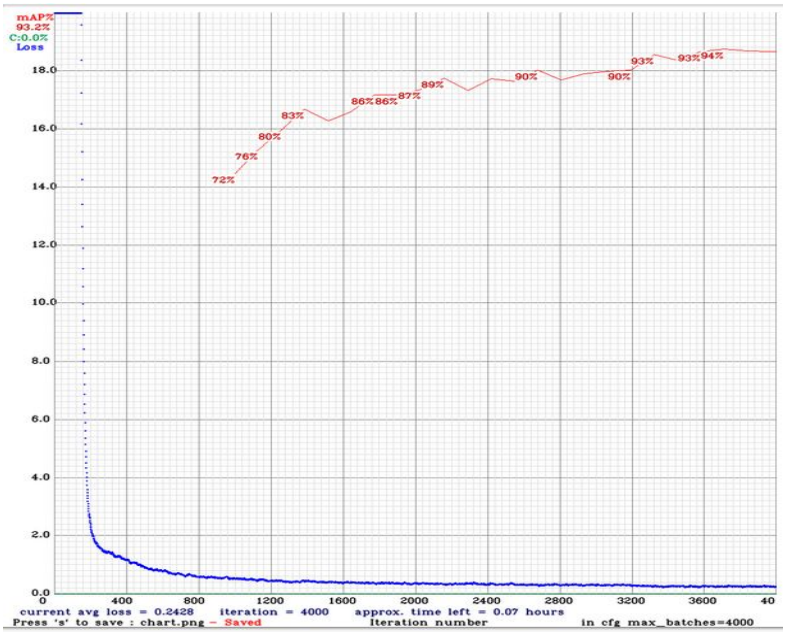

Figura 20 - Gráfico de perdas e precisão média ao longo do treinamento do modelo com Tiny YOLOv4.

Devido a interrupções no treinamento do Regular YOLO versão 4, não foi possível obter o gráfico completo a respeito da evolução do treinamento, pois o treinamento foi interrompido por causa de desconexões com o Google Colab, e dado que o gráfico de perdas começa da primeira iteração em que foi retomado, ele fica incompleto perdendo as informações a respeito das iterações anteriores. Entretanto, pode-se através de outros parâmetros obter informações a respeito da qualidade do treinamento.

4.2 Parâmetros

A fim verificar o desempenho do treinamento do YOLO, pode-se analisar alguns parâmetros como a interseção sobre união (IoU), recall, precisão, “F1 score”, True Positives (TP), Falsos Positivos (FP), Falsos Negativos (FN). A Tabela 1, apresenta os valores desses parâmetros para as versões do YOLO analisadas. A Tabela 2, por sua vez apresenta a precisão média dessas versões do YOLO para as duas classes de objetos detectadas.

Parâmetro	Tiny YOLOv3	Tiny YOLOv4	Regular YOLOv4
Precisão	85%	90%	91%
Recall	62%	93%	97%
F1 score	72%	91%	94%
Average IoU	62,0%	70,5%	74,4%
TP	360	537	558
FP	62	62	57
FN	217	40	19

Tabela 1 - Comparação entre parâmetros para diferentes versões do YOLO.

Classe	Tiny YOLOv3	Tiny YOLOv4	Regular YOLOv4
Placas	80,90%	96,87%	97,32%
Pessoas	45,12%	89,55%	96,42%

Tabela 2 - Comparação entre valores da precisão média (mAP) para diferentes versões do YOLO.

Para determinar-se a velocidade com que o algoritmo funciona é necessário rodar um vídeo e então medir o número de frames por segundo (FPS), isto é, o número de quadros processados por segundo. Esses valores de FPS variam ao longo da execução do algoritmo, e a comparação da Tabela 3, mostra que a qualidade da resolução do vídeo também influenciou os resultados.

Tipo de vídeo	Tiny YOLOv3	Tiny YOLOv4	Regular YOLOv4
Vídeo em 360p	109,6 FPS	111,9 FPS	18,8 FPS
Vídeo em 720p	22,7 FPS	23,4 FPS	17,8 FPS

Tabela 3 - Velocidade de processamento para diferentes versões do YOLO.

De todo modo, observa-se que todas as situações as versões Tiny do YOLO são executadas mais rapidamente do que a versão Regular, com destaque para a versão Tiny YOLOv4 com os melhores resultados. Além disso, a velocidade de processamento depende fundamentalmente do hardware escolhido para executar o

código, já que os testes foram feitos com uma GPU NVIDIA Tesla T4 disponibilizada no ambiente do Google Colab. De modo que em aplicações reais com câmeras acopladas em veículos em que o custo é um fator fundamental para escolha do hardware esses valores de FPS para processamento serão menores, dado ainda que o processamento adicional com a aplicação do efeito de BLUR com o OpenCV não foi considerado nessa etapa.

Uma última diferença entre esses 3 modelos, fundamental para realizar os testes, é o tamanho do arquivo ".weights" com as configurações de pesos do YOLO resultante do treinamento.

Tipo de arquivo	Tiny YOLOv3	Tiny YOLOv4	Regular YOLOv4
".weights"	34,7MB	23,5MB	256MB

Tabela 4 - Comparação entre tamanho do arquivo ".weights" para diferentes versões do YOLO.

Portanto, pela análise da Tabela 4, conclui-se que a versão Tiny YOLOv4, com 24MB, tem o menor tamanho em relação ao Tiny YOLOv3 com 35MB e ao Regular YOLOv4, com 256MB. O tamanho do arquivo é importante dependendo do tipo de hardware em que se utilizará o YOLO, em um celular por exemplo, 256MB pode ser um tamanho considerável para se ter em memória.

4.3 Imagens

Treinado o modelo, pode-se verificar visualmente o comportamento das diferentes versões do YOLO com a aplicação do efeito de BLUR no ambiente do OpenCV para então manter o sigilo de dados sob diferentes circunstâncias e ambientes. O teste com diferentes imagens permite um teste rápido do comportamento do YOLO nas mais variadas condições.

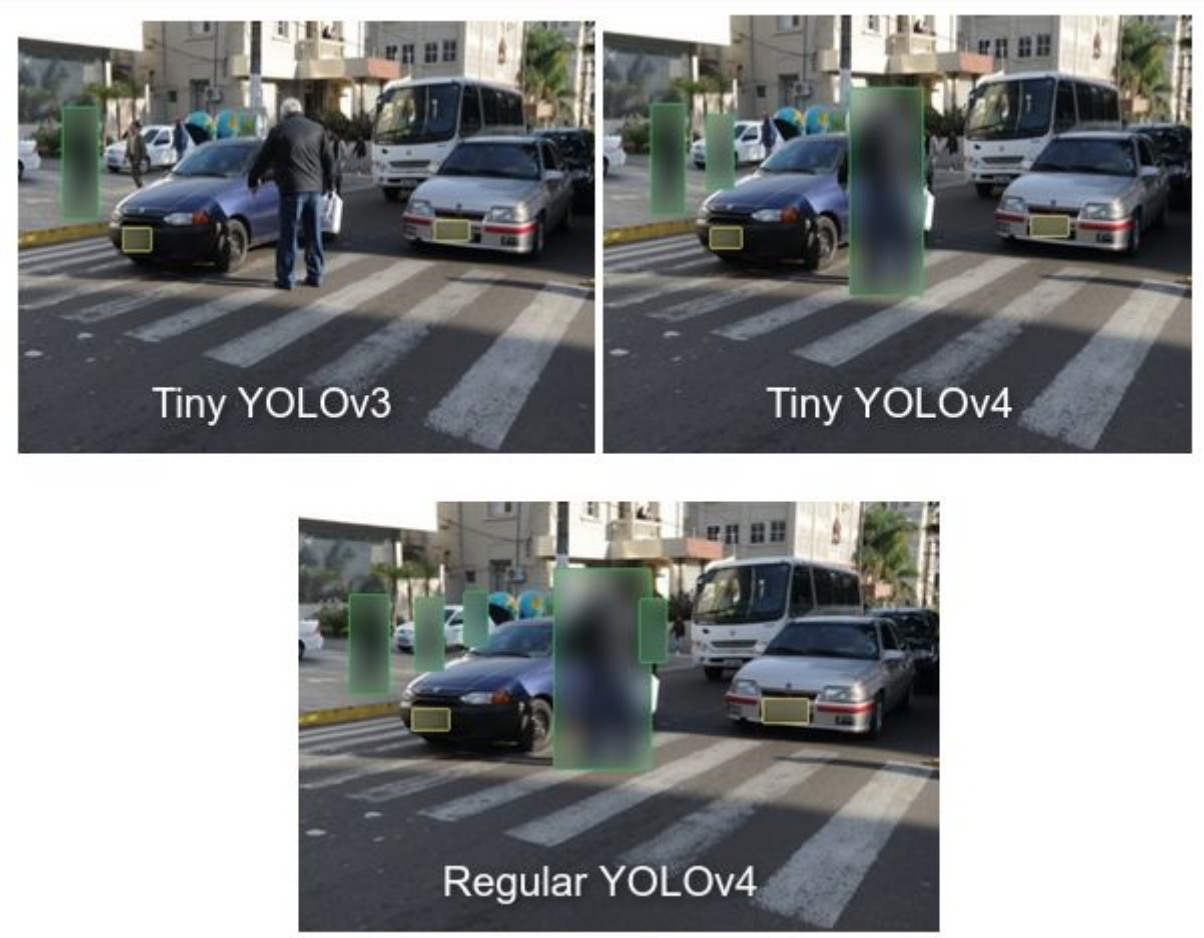

Figura 21 - Teste de aplicação do Tiny YOLOv3, Tiny YOLOv4 e Regular YOLOv4.

Na Figura 21, é possível ver que existem múltiplas pessoas e carros, nele a versão Tiny YOLOv3 detectou 2 placas e 1 pessoa. O Tiny YOLOv4 reconheceu 2 placas e 3 pessoas. O Regular YOLOv4 por sua vez detectou 5 pessoas e 2 placas de carros.

Figura 22 - Teste do Tiny YOLOv3, Tiny YOLOv4 e Regular YOLOv4 com placa da mercosul e pessoas no fundo.

Na Figura 22, observa-se uma situação com pessoas ao fundo de um carro com uma placa da Mercosul, modelo padrão das placas para novos veículos no Brasil, mas que ainda não é a mais comum em circulação nas ruas. A versão Tiny YOLOv3 não detectou nada. O Tiny YOLOv4 não reconheceu a placa, mas detectou 2 caixas delimitadoras com pessoas e o Regular YOLOv4 por sua vez identificou a placa do carro e todas as 5 pessoas presentes.

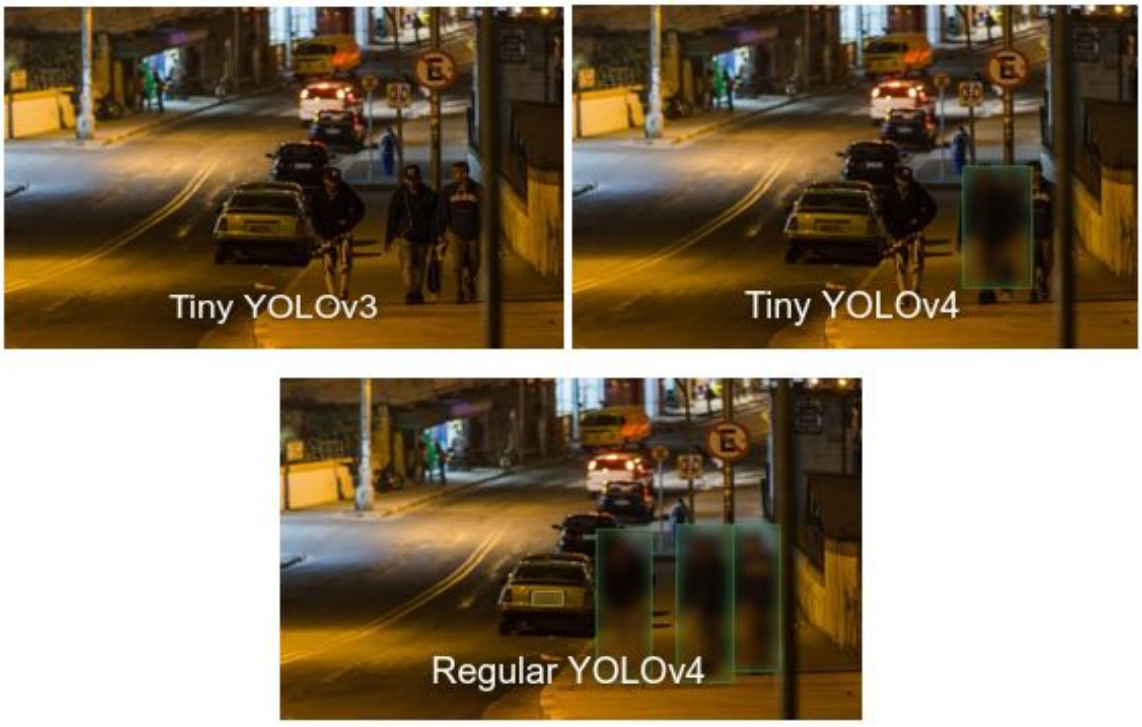

Figura 23 - Teste do Tiny YOLOv3, Tiny YOLOv4 e Regular YOLOv4 à noite.

Na Figura 23, nota-se uma situação noturna com a placa de um vídeo e 3 pedestres. A versão Tiny YOLOv3 não detectou nada. O Tiny YOLOv4 detectou 1 caixa delimitadora interseccionando 2 pessoas e o Regular YOLOv4 por sua vez identificou a placa do carro e as 3 pessoas.

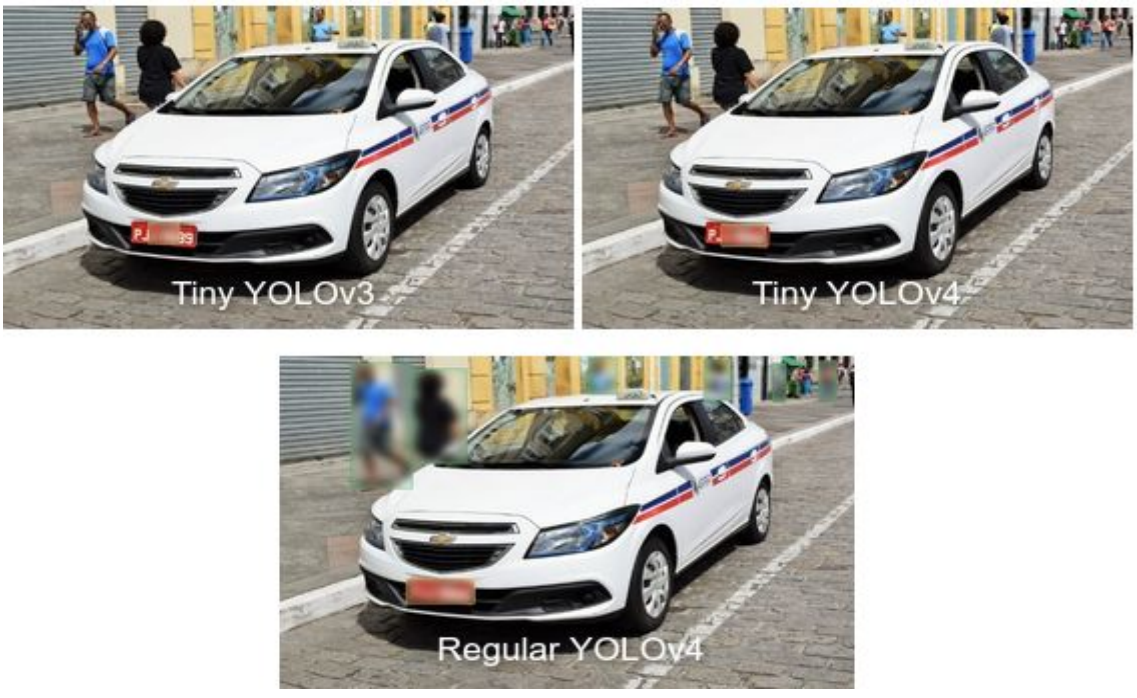

Figura 24 - Teste do Tiny YOLOv3, Tiny YOLOv4 e Regular YOLOv4 em placas vermelhas com pessoas no fundo.

A Figura 24, é constituída de um táxi com placa vermelha e pessoas ao fundo. Nele o Tiny YOLOv3 teve uma interseção com a placa menor do que o desejado e nenhuma pessoa identificada. A versão Tiny YOLOv4 detectou uma porção um pouco maior da placa com exceção da primeira letra, e também não teve sucesso na detecção de pessoas. Já a versão Regular YOLOv4, detectou as placas e as pessoas inclusive as que estavam bem ao fundo da imagem em um bar.

Figura 25 - Teste do Tiny YOLOv3, Tiny YOLOv4 e Regular YOLOv4 no trânsito.

Na Figura 25, nota-se uma situação comum de trânsito com somente 2 placas de veículos, em que todos as 3 versões do YOLO identificaram o objeto.

Figura 26 - Teste do Tiny YOLOv3, Tiny YOLOv4 e Regular YOLOv4 com placas de fundo verde.

Na Figura 26, percebe-se que para uma placa com uma cor de fundo diferente, o Tiny YOLOv4 e o Regular YOLOv4 conseguiram identificar o objeto, ao contrário do Tiny YOLOv3 que não a identificou e além disso detectou um resultado falso positivo .

4.4 Vídeo

Por fim, com uso de um vídeo foi possível averiguar o comportamento do algoritmo e observar situações em que ele se mostra eficaz ou não na solução do problema de maneira mais próxima a uma aplicação real. O threshold utilizado nesse caso foi de 0,4.

Figura 27 - Sequência 1 de frames de vídeo do trânsito de São Paulo com Tiny YOLOv3, Tiny YOLOv4 e Regular YOLOv4.

Figura 28 - Sequência 2 de frames de vídeo do trânsito de São Paulo com Tiny YOLOv3, Tiny YOLOv4 e Regular YOLOv4.

Figura 29 - Sequência 3 de frames de vídeo do trânsito de São Paulo com Tiny YOLOv3, Tiny YOLOv4 e Regular YOLOv4.

A sequência de frames das figuras 27, 28 e 29, comprovam os resultados anteriores que demonstram a menor capacidade do Tiny YOLOv3 para detectar pessoas e placas, com somente uma detecção de cada tipo, enquanto que o Tiny YOLOv4 e o Regular YOLOv4 são mais consistentes. Além disso, o Regular YOLOv4 leva vantagem sobre os demais na questão de detecção de pessoas.

4.5 Análise

Em comparação entre as versões Tiny YOLOv3 e Tiny YOLOv4, nota-se que a velocidade do algoritmo através do parâmetro de FPS foi parecida. Entretanto, a precisão e qualidade das detecções aumentou consideravelmente de 81% com o Tiny YOLOv3 para 96% com a versão Tiny YOLOv4 para a detecção de placas, e de respectivamente 45% para 90% na detecção de pessoas.

Em relação às versões Tiny YOLOv4 e Regular YOLOv4, a qualidade das detecções de placas foi similar, porém há um aumento de precisão de cerca de 8%

na detecção de pessoas. A velocidade, entretanto, cai de aproximadamente 110FPS para 20FPS.

A detecção de pessoas por se tratar de um objeto que tem uma forma imensamente variável, é mais difícil de ser filtrada e encontrada pelo YOLO tendo um índice de precisão sempre menor que o de detecção de placas para todos os modelos testados. Sendo assim, para versões Tiny, sua detecção foi abaixo de 90% indicando resultados nem sempre precisos, revelando que para esse tipo de aplicação escolher entre velocidade e desempenho é um ponto fundamental.

No caso das placas de veículos, apesar de o banco de dados do projeto envolver fundamentalmente as placas atuais do Brasil, a detecção funciona também com outras placas, seja ela com cores diferentes, ou com padrões de escrita diferente.

5. CONCLUSÃO

Dado todos os dados e procedimentos realizados no presente trabalho, observa-se que as versões “Tiny” em relação à versão completa do YOLO são executadas de maneira mais rápida pelos dispositivos, porém tem uma precisão reduzida. Portanto, é importante saber em qual tipo de aplicação tais modelos serão aplicados, para escolher melhor qual versão será utilizada.

Dessa forma, em uma aplicação como a especificada inicialmente que consiste em detectar placas de veículos e pedestres através de uma câmera acoplada a um carro para sua automação, é fundamental que a velocidade de processamento seja suficientemente rápida, para que em todo momento se mantenha a privacidade das informações, com o uso do YOLO, ao mesmo tempo em que são executadas outros procedimentos para a operação autônoma do veículo. Sendo assim é preferível utilizar as versões “Tiny”. Entretanto, como a qualidade e eficiência das detecções também deve ser a melhor possível então com isso, das opções analisadas a versão Tiny YOLOv4 é a ideal.

Apesar de a precisão ser boa o suficiente para atender as necessidades especificadas ressalta-se que ainda ocorrem falsos positivos, quando o algoritmo assinala a presença de um objeto que na verdade não é correto, e falsos negativos, quando um objeto a ser detectado não é reconhecido.

Pelos testes aplicados em diferentes imagens foi ainda possível perceber que a presença de outras variáveis influenciam na qualidade de detecção do algoritmo, como por exemplo, a cor da placa, a posição e distância dos objetos em relação a câmera, chuva, condições adversas de iluminação e ainda escolha do threshold.

Portanto, a fim de melhorar futuramente esses resultados, o banco de dados utilizados pode ser modificado, a fim de obter-se uma variedade maior de imagens e atender melhor situações específicas identificadas em que o reconhecimento não é adequado de modo a aprimorar o treinamento do YOLO. Também com as novas versões do YOLO, pode ser possível que a precisão e velocidade aumentem ainda mais para aperfeiçoamento futuro das detecções.

REFERÊNCIAS

McCALL, J; TRIVEDI, M. **An integrated, robust approach to lane marking detection and lane tracking.** IEEE Intelligent Vehicles Symposium, Proceedings, p. 533-537, 2004. Citado página 5

KONIG, D. *et al.* **Fully Convolutional Region Proposal Networks for Multispectral Person Detection.** IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, p. 243-250, 2017. Citado página 5

KHALIFA, O. *et al.* **Realtime lane detection for autonomous vehicles.** Proceedings of the International Conference on Computer and Communication Engineering 2008, ICCCE08: Global Links for Human Development, p. 82-88, 2008. Citado página 5

DICKMANNS, E. **The development of machine vision for road vehicles in the last decade**. IEEE Intelligent Vehicles Symposium, Proceedings, p.1, p.268-281, 2003. Citado página 5.

Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles; J3016 201806. **SAE,** 2018. Disponível em: <https://www.sae.org/standards/content/j3016_201806/>. Acesso em: 20 dez. 2020. Citado página 5.

EVERINGHAM, M. *et al.* **The pascal visual object classes (VOC) challenge**, International Journal of Computer Vision, vol. 88, no. 2, p. 303-338, 2010. Citado página 6.

ROSENBLATT, F. **The perceptron: A probabilistic model for information storage and organization in the brain**. Psychological Review, v. 65, n. 6, p. 386–408, 1958. Citado página 8.

GOODFELLOW, I; BENGIO, Y; COURVILLE, A. **Deep Learning (Adaptive Computation and Machine Learning series)**, p.140, p.296, 2015. Citado páginas 7 e 9.

REDMON, J; FARHADI, A. **YOLO9000: Better, faster, stronger.** Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, 2017. Citado página 10.

KATHURIA, A. What's new in YOLO v3? **Towards Data Science**, 2018. Disponível em: <https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b/>. Acesso em: 8 nov. 2020. Citado página 10.

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, Las Vegas, NV, USA, 2016. Citado página 10.

REDMON, J; FARHADI, A. **YOLOv3: An Incremental Improvement,** 2018. Citado

página 10 e 12.

CHUNG, W; ROYCE, C. **Traffic Sign Detection using You Only Look Once Framework,** 2019. Citado página 13.

BOCHKOVSKY, A; CHIEN-YAO, W; HONG-YUAN, M. **YOLOv4: Optimal Speed and Accuracy of Object Detection,** 2020**.** Citado páginas 10 e 15.

WANG, C *et al*. **CSPNet: A new backbone that can enhance learning capability of CNN.** Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPR Workshop), 2020. Citado página 15.

RUGERY, P. Explanation of YOLO V4 a one stage detector. **Medium**, 2020. Disponível em: <https://becominghuman.ai/explaining-yolov4-a-one-stage-detector-cdac0826cbd7/>. Acesso em: 20 dez. 2020. Citado página 15.

ZHANG, X *et al.* **Spatial pyramid pooling in deep convolutional networks for visual recognition**. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 37(9):1904–1916, 2015. Citado página 15.

LIU, S *et al*. **Path aggregation network for instance segmentation.** In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), p. 8759–8768, 2018. Citado página 16.

SHAPIRO L *et al.* **Computer Vision**, Prentice Hall, p. 137-150, 2001. Citado página 17.

KHANDELWAL, R. Evaluating performance of an object detection model. **Towards Data Science**, 2019. Disponível em: <https://towardsdatascience.com/evaluating-performance-of-an-object-detection-model-137a349c517b/>. Acesso em: 8 nov. 2020. Citado páginas 23, 24 e 25.

BROWNLEE, J. Loss and Loss Functions for Training Deep Learning Neural Networks. **Machine Learning Mastery**, 2019. Disponível em: <https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/>. Acesso em: 10 dez. de 2020. Citado página 27.

GLOSSÁRIO

Bag of freebies: Métodos da estrutura do YOLOv4

Bag of specials: Métodos da estrutura do YOLOv4

Backbone: Espinha dorsal, estágio inicial do YOLOv4

Batches: Lotes

Blur: Embaçar

COCO: Common Objects In Context

CNN: Rede Neural Convolucional

Dense Prediction: Tarefa de predizer a classificação de cada pixel

Features Maps: Saída de um filtro aplicado à camada anterior

Filters: Filtros

FP: Falso Positivo

FN: Falso Negativo

FPS: Quadros por segundo

Github: Plataforma de hospedagem de código-fonte

GPS: Sistema de Posicionamento Global

GPU: Unidade de processamento gráfico

Head: Cabeça, estágio final do YOLO versão 4

Height: Altura

ImageNet: Banco de dados de imagens

IoU: Interseção sobre união

IMU: Unidade de medição inercial

LIDAR: Detecção de luz e alcance

Loss Function: Função de Perdas

Machine Learning: Aprendizado de máquina

Max Batches: Máximo de lotes

mAP: Precisão média

Mish: Função de ativação não monotônica auto regularizada

Neck: Pescoço, estágio intermediário do YOLO versão 4

Open CV: Open Source Computer Vision Library

Overfitting: Sobre-ajuste

PANet: Método de agregação de parâmetros

Python: Linguagem de programação de alto nível

Pooling: Reduz a dimensão espacial de uma camada convolucional

Regular YOLOv4: Versão regular do YOLO versão 4

ReLU: Função de ativação linear retificada

RNA: Redes Neurais Artificiais

SAE: Sociedade de Engenheiros Automotivos

Softmax: Função exponencial normalizada

Subdivisions: Subdivisões

SSP: Camada de “Pooling” da Pirâmide Espacial

Steps: Passos

SVM: Support Vector Machine

Threshold: Limiar

Tiny YOLOv3: Versão reduzida do YOLO versão 3

Tiny YOLOv4: Versão reduzida do YOLO versão 4

TP: Positivo verdadeiro

Width: Largura

YOLO: You Only Look Once