Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Detecção de objetos com a arquitetura YOLO

João Vitor Esteves Gomes

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
Eduardo da Silva Ribeiro

Outubro, 2022
João Monlevade–MG

João Vitor Esteves Gomes

Detecção de objetos com a arquitetura YOLO

Orientador: Eduardo da Silva Ribeiro

Monografia apresentada ao curso de Sistemas de Informação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina "Trabalho de Conclusão de Curso II".

Universidade Federal de Ouro Preto
João Monlevade
Outubro de 2022

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTACAO E SISTEMAS

FOLHA DE APROVAÇÃO

João Vitor Esteves Gomes

Detecção de Objetos com a Arquitetura YOLO

Monografia apresentada ao Curso de Sistemas de Informação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de bacharel em Sistemas de Informação.

Aprovada em 27 de Outubro de 2022

Membros da banca

Doutor Eduardo da Silva Ribeiro - Orientador - Universidade Federal de Ouro Preto
Doutor Talles Henrique de Medeiros - Universidade Federal de Ouro Preto
Doutor Luiz Carlos Bambirra Torres - Universidade Federal de Ouro Preto

Eduardo da Silva Ribeiro, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 7/11/2022

Documento assinado eletronicamente por **Eduardo da Silva Ribeiro**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 07/11/2022, às 15:50, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0423230** e o código CRC **27DE1A69**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3808-0819 - www.ufop.br

Resumo

Um problema cada vez mais relevante e pesquisado na área de Visão Computacional, a detecção de objetos, está muito presente no mundo atual, sendo aplicada em diversas situações diferentes. As técnicas de Aprendizado em Profundidade (*Deep Learning*) são utilizadas atualmente para resolver este problema através do uso de redes neurais convolucionais. Para realizar a implementação dessas técnicas existem vários modelos responsáveis pela detecção de objetos e o mais utilizado, principalmente para as aplicações em tempo real, é o YOLO (*You Only Look Once*). Neste trabalho foi realizada uma pesquisa do estado-da-arte das principais técnicas que utilizam redes neurais convolucionais para detecção de objetos de forma mais rápida e eficiente possível, e por fim é apresentado uma comparação entre algumas das versões existentes do modelo YOLO em um *Dataset* customizado para o desenvolvimento dos testes, a fim de verificar a evolução e a melhora do modelo com as atualizações feitas nas suas novas versões. Nos testes realizados foi possível observar a última atualização do modelo bem superior as demais, principalmente em relação as versões de anos anteriores ao ano de 2020. Na principal métrica utilizada para comparação dos modelos, a YOLOv5 foi o único modelo que obteve uma porcentagem acima dos 50%, chegando a 67,9%.

Palavras-chaves: deteção de objetos. visão computacional. YOLO. redes neurais

Abstract

An increasingly relevant and researched problem in the area of Computer Vision, the object detection, is very present in the current world, being applied in several different situations. Deep Learning techniques are currently used to solve this problem through the use of convolutional neural networks. tional. To carry out the implementation of these techniques there are several models responsible for for the detection of objects and the most used, mainly for applications in time real, is YOLO (You Only Look Once). In this work, a survey of the state-of-the-art of the main techniques that use convolutional neural networks to object detection as quickly and efficiently as possible, and finally a comparison between some of the existing versions of the YOLO model in a Dataset customized for the development of tests, in order to verify the evolution and improvement of the model with the updates made in its new versions. In the tests carried out it was possible to observe the last update of the model much superior to the others, mainly in relation to versions from years prior to the year 2020. In the main metric used for comparison of models, YOLOv5 was the only model that obtained a percentage above 50%, reaching 67.9%.

Key-words: object detection. computer vision. YOLO. neural networks

Lista de ilustrações

Sumário

1 Introdução

A visão é um dos sentidos mais essenciais do ser humano e é importante para que ocorra a interação do mesmo com o meio em que está inserido. A função que o olho humano apresenta, quando analisado pela perspectiva da história óptica é de alta complexidade (TOSSATO, 2005). A visão possui três elementos principais: o objeto que é visto, um olho que vê esse objeto e um meio que está entre eles. Somado a isso o olho capta a energia luminosa refletida pelos objetos e a transforma em impulsos nervosos que são transmitidos ao cérebro de forma automática e involuntária convertendo na imagem do objeto observado (PASSOS; ANDRADE-NETO; LEMAIRE, 2008). Com o grande avanço tecnológico, existem diversas áreas que são responsáveis por criar máquinas e sistemas que simulem os diversos sentidos e funções das pessoas. A visão está inserida nisso, tendo a sua própria tecnologia chamada de Visão Computacional que está inserida na área de Aprendizado de Máquina.

O desenvolvimento de técnicas computacionais como a construção de sistemas que são capazes de adquirir conhecimento de forma automática é um dos objetivos da grande área de Inteligência Artificial, e é neste contexto que a Aprendizagem de máquina está inserida. Um sistema de aprendizado pode ser definido como um programa de computador que toma decisões baseado em experiências acumuladas através da solução bem-sucedida de problemas anteriores. Por ter como principal objetivo simular o comportamento de um ser humano dentro dos ambientes convencionais, os algoritmos de Aprendizado de Máquina são considerados como cavalos de trabalho na era tecnológica em que vivemos (ALPAYDIN, 2020). Existem várias aplicações que são muito importantes atualmente e fazem uso do Aprendizado de Máquina, como por exemplo: diagnósticos médicos (KONONENKO, 2001), detecção de fraudes na internet (AWOYEMI; ADETUNMBI; OLUWADARE, 2017), reconhecimento de fala (PADMANABHAN; PREMKUMAR, 2015), classificação de imagens (LI et al., 2019), dentre outros. Além dessas aplicações citadas, existe também a Detecção de Objetos que faz o uso do Aprendizado de Máquina e é o foco principal deste trabalho.

A partir da década de 2010, surgiram diversas aplicações que utilizam sistemas automatizados capazes de retirar informações a partir de imagens ou vídeos digitais. Estas tecnologias da Visão Computacional, se tornaram essenciais em diversas aplicações, como por exemplo carros autônomos e detecção facial (FRANÇA et al., 2009). Assim como a visão humana, a Visão Computacional também possui como entrada uma imagem (MARENGONI; STRINGHINI, 2009), porém a saída é uma interpretação das características da imagem como um todo ou alguma parte dela. A Visão Computacional se diferencia um pouco do processamento de imagens, ela pode ser entendida como arte de fazer o

computador aprender a ver as coisas e não somente enxergar, pois quando usamos essa expressão, não se fala apenas de se tirar uma foto, mas entender o que está presente nesta foto. Portanto, o processamento de imagens é capaz de preparar a imagem para facilitar a aplicação dos métodos de Visão Computacional. Enquanto a detecção de imagens vai fazer uma análise e interpretação do conteúdo da imagem procurando obter informações relevantes para que consiga detectar a classe e marcar o objeto na imagem.

Figura 1 – Diferença entre processamento e detecção de imagem. (MARENGONI; STRINGHINI, 2009)

1.1 Objetivos

O presente trabalho tem o objetivo de entender as principais abordagens existentes para detecção de objetos em imagens e assim realizar experimentos a fim de detectar os resultados alcançados em alguns modelos já existentes. Fazem parte do escopo a elaboração e revisão da documentação referente aos principais temas e modelos envolvidos no trabalho, desenvolvimento do treinamento e avaliação dos testes realizados. Os objetivos específicos são:

- estudar os *Datasets* para detecção de objetos e avaliar o melhor a ser usado;
- avaliar e implementar modelos de detecção de objetos;
- analisar os resultados alcançados.

2 Detecção de objetos

A detecção de objetos é um problema clássico na área de visão computacional, tendo como principal característica a localização de objetos que existem em uma imagem estática ou em movimento, além de dizer qual objeto é este (LI et al., 2021). Na detecção pode-se determinar quantos objetos de uma classe estabelecida possam existir na imagem e determinar onde este objeto está localizado. Diferentemente da classificação de imagens onde é feita somente a previsão da existência de um objeto em uma determinada imagem, na detecção é mostrado onde está esse objeto através de uma caixa retangular delimitadora. Há mais de 50 anos já existem técnicas propostas para serem capazes de resolver problemas relacionados a Visão Computacional, como o método de detecção por seguimento de bordas proposto por (YAKIMOVSKY, 1976).

O avanço tecnológico das redes neurais artificiais, responsáveis pelas técnicas atuais de detecção de objetos, possibilitou a ampla utilização dessas técnicas a partir dos anos 2000, pois foram desenvolvidos métodos mais eficientes e em conjunto a isto o avanço da tecnologia possibilitou que os hardwares possuíssem mais velocidade e capacidade para utilização desses métodos.

2.1 Redes Neurais

As redes neurais, também chamadas de redes neurais artificiais (ANNs), são um subconjunto de Aprendizado de Máquina. O seu nome e sua arquitetura são influenciados pela semelhança e a simulação do cérebro humano (BISHOP; NASRABADI, 2006), imitando a forma como os nossos neurônios enviam sinais uns aos outros. A simulação das emoções das pessoas ainda está muito longe de se tornar uma realidade, indo em contramão as ANNs que conseguem se inspirar na funcionalidade do cérebro humano, onde centenas de neurônios interconectados processam informações em paralelo, simulando em certos níveis da inteligência humana (WANG, 2003). Com isso, as redes neurais acabam sendo capazes, por exemplo, de reconhecer padrões em determinados meios, classificar imagens e tomar decisões (TACCHINO et al., 2019). Uma rede neural simples possui três camadas, a camada de entrada, outra camada de saída, que é o escopo e, entre estas duas existe uma camada oculta (HASTIE et al., 2009), como mostrado na Figura 2. As camadas são conectadas através de nós e as conexões formadas entre elas formam uma rede de nós interconectados. Esses nós relacionam os valores de saída e entrada e são modelados pela equação:

$$y^{(ij)} = f(\sum_{i'}^{n}(y^{i'(j-1)}w_{i'}^{ij} + b^{ij}) \tag{2.1}$$

em que $y^{(ij)}$ = valor de saída do neurônio i da camada j; n = número de neurônios da camada anterior; $y^{i'(j-1)}$ = valor de saída do neurônio i' da camada anterior; $w_{i'}^{ij}$ = valor do peso sináptico do neurônio i da camada j, ativado pelo neurônio i' da camada anterior; b^ij = valor de compensação do neurônio i da camada j; f = função de ativação do neurônio i.

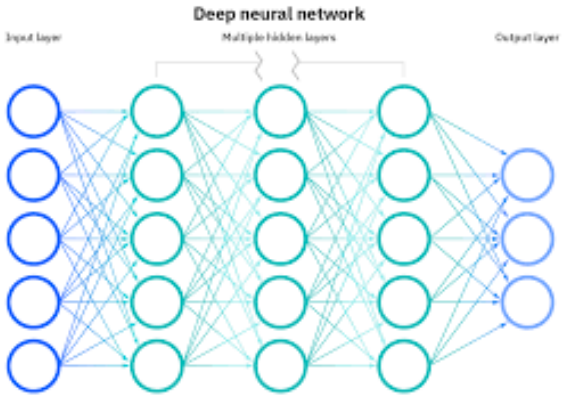

Figura 2 – Rede Neural Artifical. (IBM Cloud Education, 2022)

2.1.1 Neurônio Artificial

O neurônio artificial (nó), mostrado na Figura 3, é um modelo criado para simular o funcionamento de um neurônio real do organismo do ser humano de forma um pouco mais simplificada (HASTIE et al., 2009). Assim como os neurônios biológicos, os neurônios artificiais também podem aprender a resolver um problema. As características principais do neurônio artificial é a representação de conhecimentos que foi baseado em conexões e a sua adaptação a características diferentes (HARMON, 1959). Os nós são ativados através da função de ativação quando há estímulos ou entradas suficientes. Essa ativação se espalha através da rede, criando um resultado ao estímulo. As conexões entre esses neurônios artificiais agem como sinapses simples, fazendo os sinais serem transmitidos de um para o outro. Há sinalização entre as camadas conforme os nós viajam da primeira até a última e são processados ao longo do caminho.

2.1.2 Função de ativação

As funções de ativação permitem que pequenas mudanças nos pesos realizem apenas uma pequena alteração no resultado alcançado na camada de saída da rede neural (ALPAYDIN, 2020). A função de ativação é um fator essencial que poderá permitir com que qualquer rede de neural aprenda.

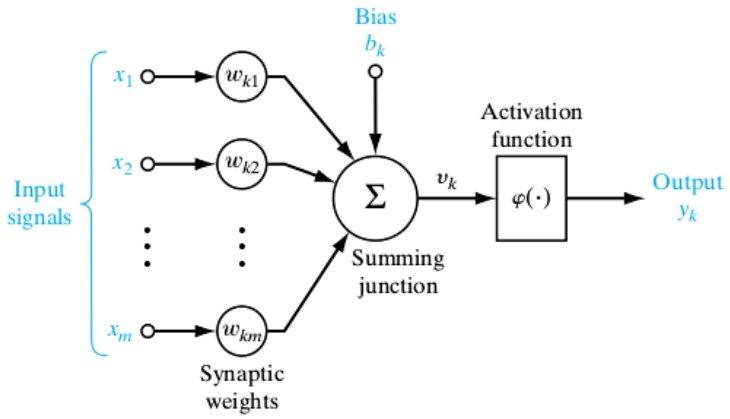

Figura 3 – Neurônio artificial. (Mateus Coelho, 2022)

São elas quem decide se um neurônio vai ser ativado ou não dentro das camadas, por isso é um elemento extremamente importante das redes neurais artificiais. Ou seja, elas verificam se a informação que o neurônio está recebendo é relevante para ser fornecida ou se deve ser ignorada (JAMES et al., 2013).

2.2 Redes Neurais Convolucionais

A Rede Neural Convolucional (CNN) é um modelo de aprendizado profundo (*Deep Learning*) que consegue reconhecer uma imagem que será lida e busca atribuir importância para ela, através da diferenciação de alguns aspectos (brilho, cor, tamanho, entre outros) e objetos presentes na imagem e com isso realizar o reconhecimento do objeto identificado em questão (LI et al., 2021). Com relação a outros algoritmos de classificação, a CNN exige muito menos no processo de pré-processamento, pois ele possui a capacidade de aprender sozinho os filtros e as características de diversos objetos que possam aparecer nas imagens, enquanto no modelo de classificação esse aprendizado é preciso ser feito manualmente. Sendo assim a CNN é uma alternativa viável aos demais métodos. A convolução usada nas redes neurais é a aplicação de um filtro na entrada da rede. O filtro é constituído por uma matriz de tamanho pequeno chamado *kernel*, que é aplicado na imagem representada por uma grande matriz com diferentes valores associados com a coloração de cada pixel, chamado de *tensor*. Esse filtro geralmente de tamanho reduzido (por exemplo 3x3) é utilizado para se calcular os produtos da multiplicação entre *kernel* e *tensor*, após isso é feita uma soma para se obter o valor da saída equivalente a posição do *kernel* que é chamado de *feature map* (mapa de recursos), conforme exemplificado na Figura 4. Esse procedimento é feito até o filtro ter passado por toda matriz de entrada. Após o processo ser repetido diversas vezes, o resultado é o mapa de recursos final que indica os locais e as características existentes em uma imagem (RODRIGUES, 2018).

A CNN é um tipo de rede neural artificial que requer uma camada convolucional,

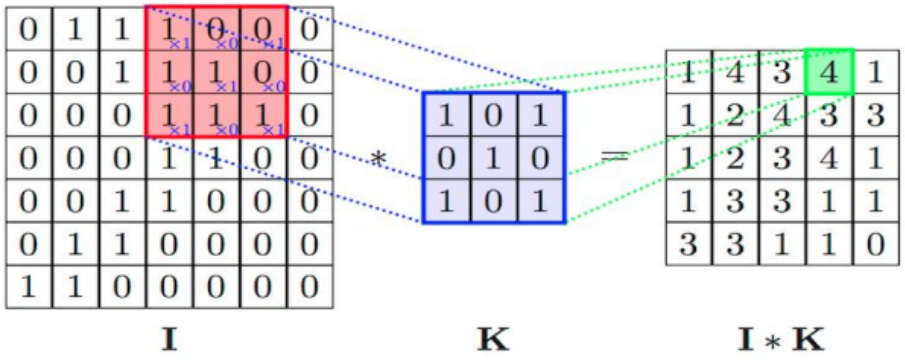

Figura 4 – Exemplo de uma convolução.

mas pode ter outros tipos de camadas, como não linear, pooling, e camadas totalmente conectadas, para criar uma profunda rede neural (ALBAWI; MOHAMMED; AL-ZAWI, 2017). A convolução do filtro existente na camada convolucional é representada pela equação:

$$x_{ij}^{l} = \sum_{a=0}^{\text{m-1}} \sum_{b=0}^{\text{m-1}} \omega_{ab} \gamma_{(i+a)(j+b)}^{l-1} \tag{2.2}$$

nela é mostrada a convolução de uma imagem de tamanho NxN por um filtro γ de tamanho mxm. A saída terá tamanho $(N-m+1)x(N-m+1)$. O l corresponde a localização do mapa e do filtro na rede.

Portanto, para a extração das características, a CNN possui três elementos básicos:

- Camada de convolução: realiza a extração dos primeiros recursos capturados na imagem. Onde, conforme já citado, na extração o filtro de convolução possui tamanho reduzido e esses filtros passam por todos os dados de entrada e realizam a operação de convolução.
- Camada de *Pooling*: tem o papel de diminuir o tamanho dos dados capturados pela camada de convolução, fazendo com que o próximo passo convolucional receba uma forma diferente de representação de dados do que a anterior. Sua finalidade pode ser expressa por diminuir o poder computacional necessário para se processar os dados, e principalmente, extrair as principais características das imagens a fim de que as classes possam ser reconhecidas não importando seu tamanho e como estão distribuídas na imagem
- Rede totalmente conectada: a camada localizada no final da rede neural, é onde os recursos que foram extraídos pelas camadas anteriores são aplicados para a classificação. Após os filtros usados anteriormente tem-se os dados achatados e transformados em um vetor de números de uma dimensão que são conectados por uma ou mais camadas totalmente conectadas.

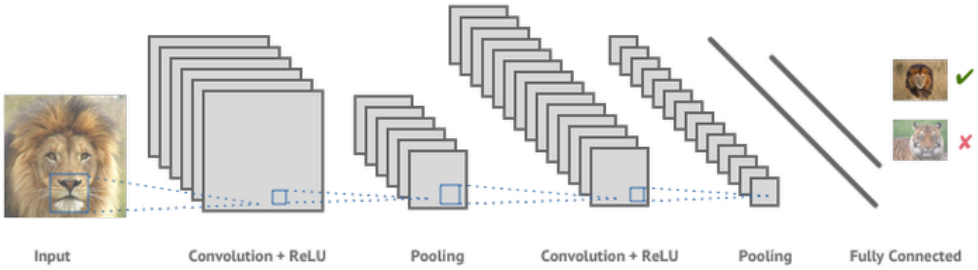

Figura 5 – Exemplo da arquitetura, utilizando as camadas de convolução, pooling e totalmente conectadas. (Pedro Ney Stroski, 2022)

2.3 Treinamento da Rede

Por ser um modelo de *Deep Learning* confiável para realizar previsões automatizadas, as CNNs conseguem extrair as características essenciais de forma automática a partir de uma entrada fornecida (ZHAO et al., 2019). Isso é possível pois as redes são treinadas anteriormente através de um conjunto de dados pré-selecionados e assim é possível estabelecer padrões através do treinamento e a partir desse ponto definir um nível de confiança que será usado para realizar as detecções no processo.

Para treinar uma rede neural e verificar se os pesos estão adequados é aconselhado dividir as amostras existentes entre treinamento e validação. Sugere-se que a divisão feita seja de 60% a 90% das amostras do *Dataset* para treinamento e 40% a 10% para validação (Sérgio Eduardo Palmiere, 2016). Denomina-se época de treinamento cada vez que é apresentado ao modelo uma amostra para ajuste dos pesos durante o processo de treinamento.

O treinamento pode ser feito quantas vezes for necessário e isso é definido nos parâmetros de treinamento existentes no algoritmo. A apresentação de todos os padrões de treinamento disponíveis corresponde a uma época, ao ser finalizado treinamento é interrompido e a rede é testada com os dados de validação (ZHAO et al., 2019). Os dados usados para construir o modelo final geralmente vêm de vários conjuntos de dados. Três principais são frequentemente usados em diferentes estágios na criação do modelo, são eles: conjunto de treinamento, conjunto de validação e conjunto de testes.

- Dados de treinamento: é um conjunto de dados de exemplos usados durante o processo de aprendizagem e é usado para ajustar os parâmetros e treinar o modelo.
- Dados de validação: é um conjunto de dados de exemplos usados para ajustar os hiper parâmetros através da comparação de diferentes modelos.
- Dados de teste: é um conjunto de dados usado para comprovar que aquele modelo realmente funciona. São dados que foram ignorados no treinamento e no processo de escolha de hiper parâmetros.

Durante o processo de treinamento o modelo da CNN vai aprender e cometer erros. Para cada erro que o modelo comete, há uma penalidade e isso é representado no valor da perda (*loss*) para cada época e no final da última época do treinamento a CNN deve apresentar o mínimo de perda e a maior precisão que for possível (ALPAYDIN, 2020). Os algoritmos para treinamento das redes além de possuir os conjuntos de dados que foram citados acima, também necessitam de outras funcionalidades essenciais para o aperfeiçoamento de uma CNN. Entre essas funcionalidades estão: a função de custo e o procedimento de otimização.

2.3.1 Função de custo (*Loss function*)

A função de custo, também chamada de *Loss function*, é uma função que quantifica a proximidade entre um valor inferido pela rede e um valor de referência de acordo com alguma métrica estabelecida (JAMES et al., 2013). Ou seja, a função de custo é utilizada para calcular a diferença entre a saída que se deseja e a saída que a rede neural está apresentando no treinamento da CNN.

Podemos entender que função de custo trabalha com seguinte pensamento: "O resultado final do treinamento da rede é parecido com o que era esperado?". Por exemplo, a rede trouxe a saída 0.2, mas era esperado 1.0, isso significa que a rede "errou" por 0.8 pontos. Porém é preciso fazer uma média de todas as épocas de treinamento para calcular esses "erros".

A fórmula para calcular a *Loss Function* pode ser vista abaixo (3.3). Basicamente o que ela faz é somar os quadrados de todas as diferenças. Em geral, os pesos são definidos inicialmente com valores aleatórios, que com frequência produzem um valor alto de perda quando começamos a treinar uma rede neural. Porém as funções de custo acabam causando um alto custo computacional, pois é necessário ajustar todos os nós das redes após a aplicação da função e isso é tratado nos algoritmos de otimização.

$$f(m,b) = \frac{1}{N}\sum_{i=1}^{\mathrm{n}}(y_i - (mx_i + b))^2 \tag{2.3}$$

2.3.2 Algoritmo de otimização

As não linearidades pertencentes às redes neurais fazem com que a maioria das funções custo se tornem não convexas. Por esse motivo, no treinamento de redes neurais não se usam otimizadores com garantia de convergência global, mas sim procedimentos de otimização iterativos e baseados no gradiente, que buscam reduzir o custo a um valor baixo (BISHOP; NASRABADI, 2006). O procedimento de otimização visa minimizar a função custo.

O algoritmo *Gradient Descent* que é o mais utilizado para minimizar a função de custo em algoritmos de aprendizado de máquina (RUDER, 2016), é dividido em três tipos:

- *Batch Gradient Descent*: neste, o algoritmo processa todos os exemplos de treinamento para cada iteração. Mas se o número de exemplos de treinamento for grande, o *Batch Gradient Descent* terá um custo computacional muito grande. Portanto, se o número de exemplos de treinamento for grande, a *Batch Gradient Descent* não é preferível. Para este tipo de problema é preferível o *Stochastic Gradient Descent* ou o *Minibatch Gradient Descent*.

$$\theta = \theta - \eta * \nabla_\theta J(\theta) \tag{2.4}$$

- *Stochastic Gradient Descent*: neste, o algoritmo processa um exemplo de treinamento por iteração. Portanto, os parâmetros vão sendo atualizados mesmo após uma iteração onde foi processado apenas um exemplo. Por isso, é muito mais rápido do que o *Batch Gradient Descent*. Mas ainda assim, quando o número de exemplos de treinamento é grande ele processa apenas um exemplo, o que pode ser uma sobrecarga adicional para o sistema, pois o número de iterações será muito grande.

$$\theta = \theta - \eta * \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.5}$$

- *Minibatch Gradient Descent*: neste, o algoritmo funciona mais rápido do que o *Batch Gradient Descent* e *Stochastic Gradient Descent*. Aqui n exemplos em que $n < m$ são processados por iteração. Portanto, mesmo que o número de exemplos de treinamento seja grande, ele é processado em lotes de n exemplos de treinamento de uma vez. Assim, funciona para exemplos de treinamento maiores e com menor número de iterações.

$$\theta = \theta - \eta * \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{2.6}$$

2.3.3 Regularização

Independente da *Loss function* utilizada, comumente é adicionado à mesma um termo de penalização a valores muito grandes nos parâmetros do modelo em uma técnica conhecida como regularização. O propósito da regularização é ajustar o equilíbrio entre *underfitting* e *overfitting* (JAMES et al., 2013). Pode determinar se um modelo preditivo está fazendo o sub ajuste ou o sob reajuste dos dados de treinamento consultando o erro de previsão nos dados de treinamento e nos dados de avaliação. Ao penalizar parâmetros com valor alto, o processo de otimização impede que poucos parâmetros que se encaixam

bem a um determinado exemplo se sobressaiam aos demais, pois esses mesmos parâmetros podem não produzir bons resultados nos demais exemplos (KOEHRSEN, 2018).

O *overfitting* ocorre quando o modelo se adaptou muito bem aos dados com os quais está sendo treinado; porém, não generaliza bem para novos dados. Ou seja, o modelo "decorou" o conjunto de dados de treino, mas não aprendeu de fato o que diferencia aqueles dados para quando precisar enfrentar novos testes. O *underfitting* ocorre quando o modelo não se adapta bem sequer aos dados com os quais foi treinado (JAMES et al., 2013). Para evitar a ocorrência do *overfitting*, existe um algoritmo para o treinamento das redes neurais que tem como fundamento eliminar alguns neurônios de forma aleatória durante o processo de aprendizagem, esse algoritmo é chamado de *droupout*.

Em um conjunto de dados muito grande, onde existem diversas variáveis, além do problema do *overfitting*, também é gerado outro problema ao criar um modelo muito complexo que tem tantos cálculos, que é o alto custo computacional, sendo demorado e altamente intensivo para o computador treinar aquele modelo. Em ambos os casos, a regularização pode ser uma boa ferramenta para resolver o problema, já que ela remove as variáveis menos importantes do modelo.

3 Abordagens para Detecção de Objetos

Na detecção de objetos existe uma contradição, no momento da classificação não é preciso ter a sensibilidade da posição, já a detecção precisa ter essa sensibilidade. Para realizar essa função existem alguns desafios que acabam tornando esta tarefa mais difícil de ser realizada com uma grande precisão, entre esses desafios estão:

- a variedade do tamanho que os objetos podem ser encontrados;
- o grande número de objetos distintos que podem ser encontrados;
- detecção da posição do objeto marcada pela caixa delimitadora e a acurácia da classificação dos objetos encontrados.

Existem diversos modelos que são dedicados à tarefa de detecção de objetos onde cada um deles procura cumprir de forma eficiente e prática esta função (ZHAO et al., 2019). Estes modelos são categorizados em dois tipos: modelos de 2 estágios e modelos de 1 estágio.

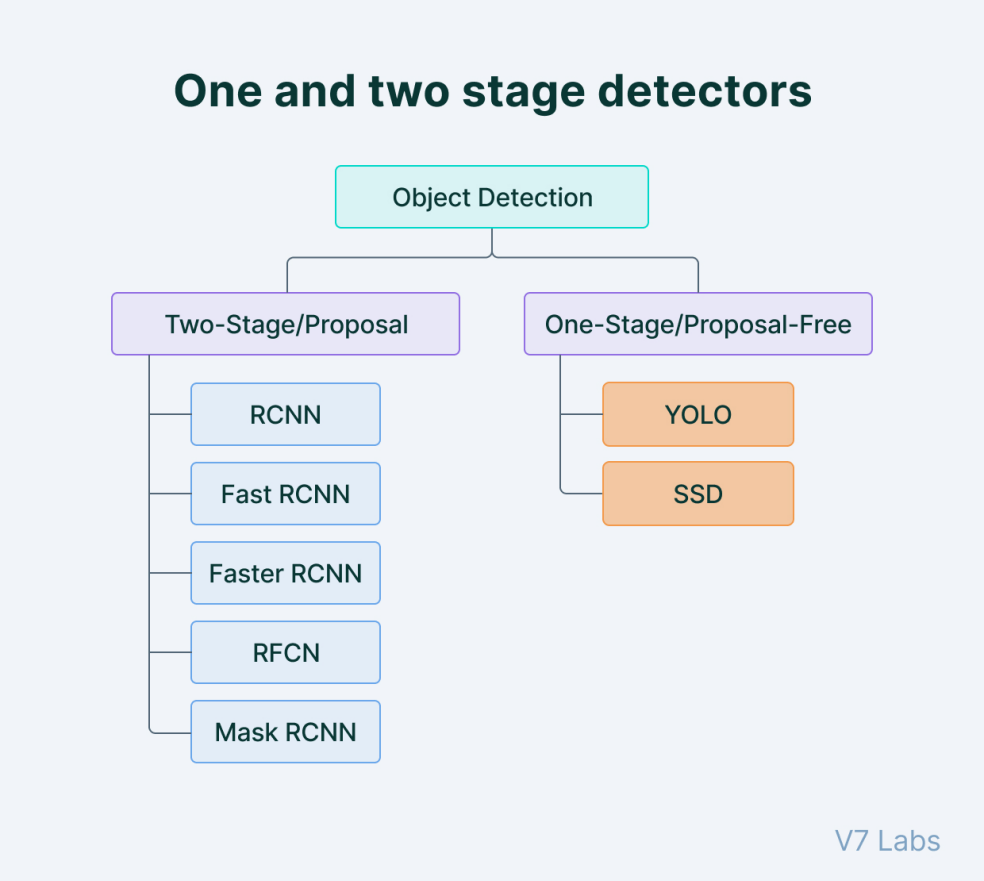

Figura 6 – Modelos que fazem uso de 2 e 1 passo na CNN. (Alberto Rizzoli, 2022)

Neste trabalho será feito o uso e a comparação das diversas versões da arquitetura YOLO que é o principal modelo utilizado para detecção de objetos atualmente e faz parte do grupo dos modelos de 1 estágio (REDMON et al., 2016).

Figura 7 – Comparação entre arquiteturas. (ANKIT SACHAN, 2022)

3.1 Abordagens com 2 estágios

Os detectores de dois estágios dividem a tarefa de detecção de objetos em dois pontos: a extração das regiões de interesse da imagem e a classificação dessa região. Entre os modelos que fazem o uso dessa abordagem estão o R-CNN, R-FCN e o *Faster* CNN (GIRSHICK et al., 2014).

3.1.1 R-CNN

Uma técnica bastante conhecida na área de detecção de objetos, a R-CNN foi a pioneira na resolução de um problema antigo no uso de CNN para detecção, que era o alto custo computacional causado pela prática da técnica de "janela deslizante". A R-CNN utiliza de um algoritmo que faz uso da pesquisa seletiva, que reduz o número de caixas delimitadoras que são alimentadas na classificação das imagens para cerca de 2.000 propostas por região (GIRSHICK et al., 2014). Essa técnica usa dicas como textura, intensidade e cor para gerar todas as localizações possíveis do objeto. Após isso as caixas delimitadoras são inseridas no classificador de imagens da CNN do modelo. Quando a rede recebe as caixas delimitadoras é feito o treinamento das regiões possíveis de forma separada, recebendo assim a informação se o objeto realmente existe no local. Na Figura 8 é possível exemplificar como o funciona o modelo R-CNN, após receber a imagem de entrada é feita a extração das regiões propostas a terem caixas delimitadoras, depois aquela região é "retirada"e irá passar pela CNN onde será feita a extração das características e por fim na saída da última camada é feita a classificação do objeto que foi identificado pela extração, nesse caso uma pessoa.

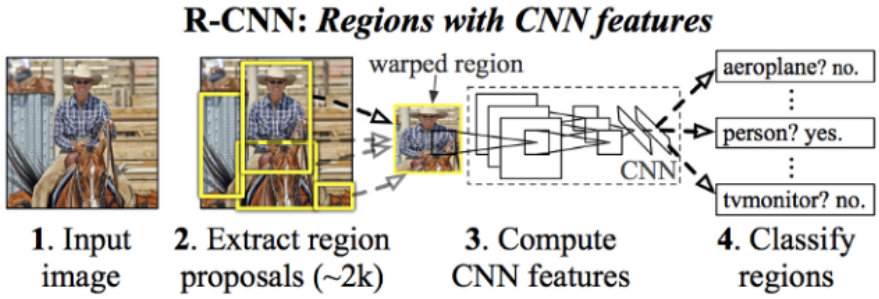

Figura 8 – R-CNN. (GIRSHICK et al., 2014)

3.1.2 *Faster* R-CNN

Esta abordagem é uma evolução da arquitetura R-CNN. Nesta, apesar do tempo de treinamento ser baixo sem impactar na precisão, a rede ainda não é rápida o suficiente para ser usada em tempo real. O que ocasiona este empecilho é o algoritmo de pesquisa seletiva que é utilizado no processo, conforme falado anteriormente. Portanto a *Faster* R-CNN foi proposta para não utilizar a pesquisa seletiva e ao invés disso é usado um algoritmo para a geração de proposta de região chamado Rede de Proposta de Região.

A *Faster* R-CNN contém duas redes: Rede de Proposta de Região (RPN) e a Rede de detecção de objetos. A imagem inteira é enviada como entrada para a rede convolucional gerando o mapa de características, porém ao invés da busca seletiva é utilizada a RPN, Figura 9. A RPN apenas produz as coordenadas das caixas delimitadoras, mas não tenta classificar nenhum objeto, tendo como propósito apenas propor as regiões que contém objetos (REN et al., 2015).

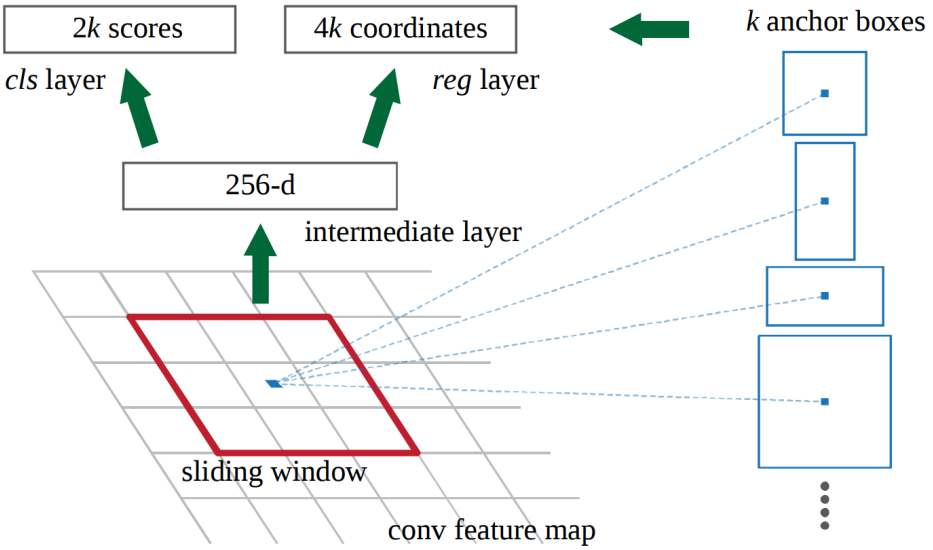

Figura 9 – RPN. (REN et al., 2015)

Esta rede trabalha da seguinte forma:

- uma janela de tamanho $3x3$ se move pelo mapa de características na última camada de uma rede neural inicial com objetivo de gerar dimensões menores para a imagem.
- são geradas diversas regiões possíveis para cada local da janela baseada em k caixas delimitadoras.
- cada uma dessas regiões geradas indica uma probabilidade de ter um objeto nessa região e as coordenadas que apresentam as caixas delimitadoras.
- cada lugar do mapa de características é examinado e são consideradas k caixas diferentes centralizadas nele de diferentes formatos. Para cada caixa é calculada a probabilidade de existir um objeto ali e as coordenadas.

Após a RPN, entra em uso a Rede de Detecção de objetos. É adicionada uma camada de *pooling*, camadas totalmente conectadas e por fim uma camada de otimização para a classificação (Geeks Forge, 2022), como ilustrado na Figura 10. Essa arquitetura obteve velocidade e precisão no processo muito melhores do que a sua arquitetura anterior, sendo um dos métodos com melhor desempenho atualmente para detecção de objetos.

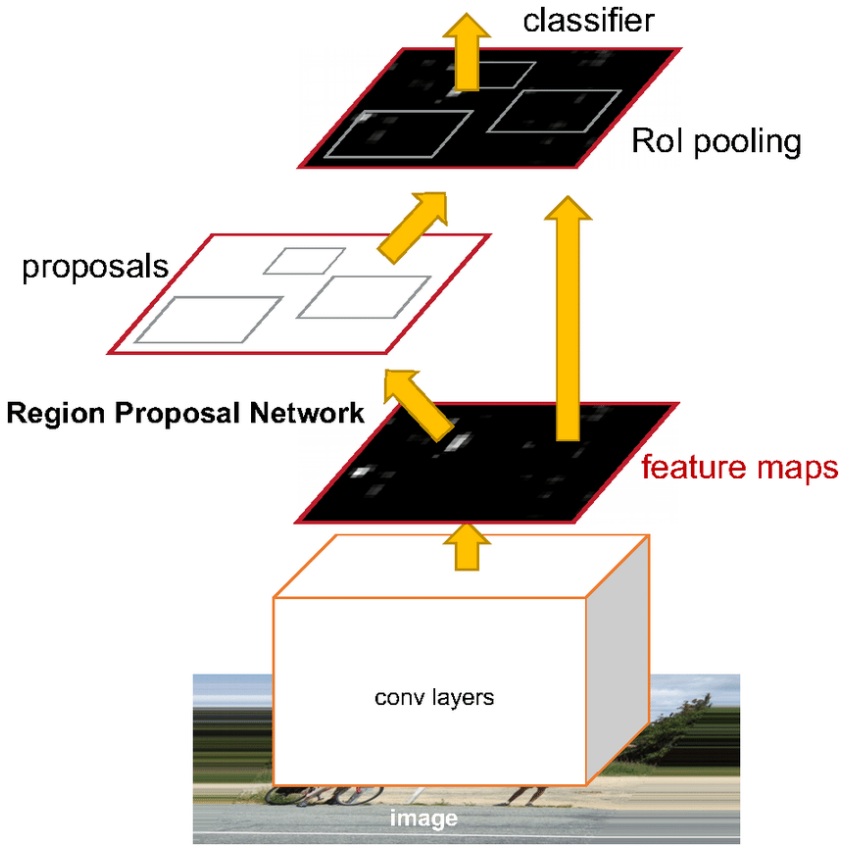

Figura 10 – *Faster* R-CNN. (Geeks Forge, 2022)

3.2 R-FCN

A abordagem R-FCN, assim como a *Faster* R-CNN é uma arquitetura de detecção de dois estágios focada na proposta de identificar a região em que o objeto está. Porém, para diminuir o número de operações realizadas na detecção, a camada de agrupamento é retirada do processo e colocada uma camada de convolução de 100 camadas anterior a sub-rede (DAI et al., 2016). A imagem é trabalhada para transformar a dimensão de saída da camada em 1024 pixels. Uma outra camada é então adicionada para conectar a última camada de convolução compartilhada antes de entregar o seu resultado de saída, conforme ilustrado na Figura 11.

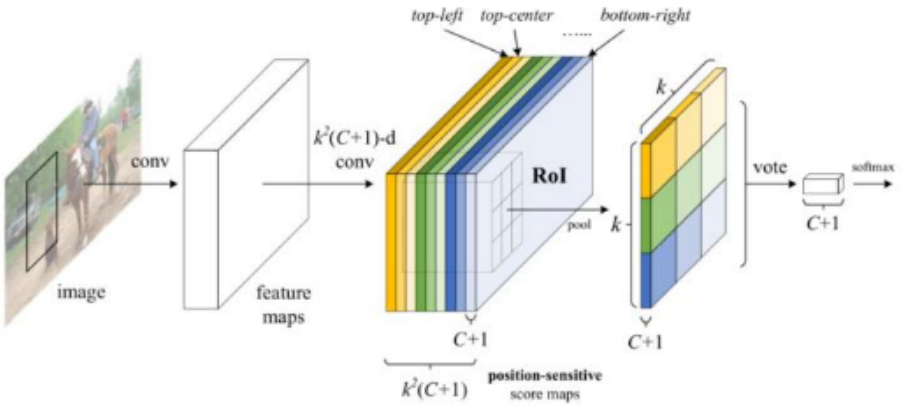

Figura 11 – R-FCN. (DAI et al., 2016)

3.3 Abordagens com um estágio

As abordagens de detecção de objetos que fazem uso de apenas um passo, retiram do seu treinamento o processo de extração das áreas que possuem características de um objeto. Sendo assim a rede realiza a extração de características dentro das caixas delimitadoras que podem conter objetos em um só estágio (HUANG; PEDOEEM; CHEN, 2018).

3.3.1 YOLO *(You Only Look Once)*

A arquitetura YOLO é um método bastante utilizado para detecção de objetos de aplicações em tempo real (FANG; WANG; REN, 2019). Uma das suas principais aplicações é o seu uso nas câmeras de trânsito, pois possui um baixo tempo de execução e treinamento, como mostrado por Wu, Wang e Liu (2021) nos experimentos realizados no seu trabalho. Neste método ele reconhece uma imagem e tenta aprender quais são as possíveis classes de objetos existentes nela.

A YOLO utiliza uma rede única, onde ela recebe uma imagem de entrada e ela é treinada totalmente de forma que as caixas delimitadoras são previstas em conjunto com as possíveis classes para cada uma delas. Portanto, por fazer o uso de apenas uma rede

para realizar o processo, a YOLO consegue obter uma maior velocidade no seu processo e assim sendo preferido para detecção de objetos em tempo real (HUANG; PEDOEEM; CHEN, 2018). Outro fator que aumenta a velocidade da rede YOLO é a capacidade de rodar em uma de taxa de atualização de até 45 frames por segundo (FPS).

A YOLO divide a imagem em uma rede de grade SxS, onde em cada uma dessas redes são retiradas caixas delimitadoras (REDMON et al., 2016). Para cada uma dessas caixas é feita uma probabilidade de existir um objeto nela ou não, se essa probabilidade está acima de um patamar pré-definido ela é candidata a ser utilizada para a localização do objeto na imagem de entrada, exemplificado na Figura 12.

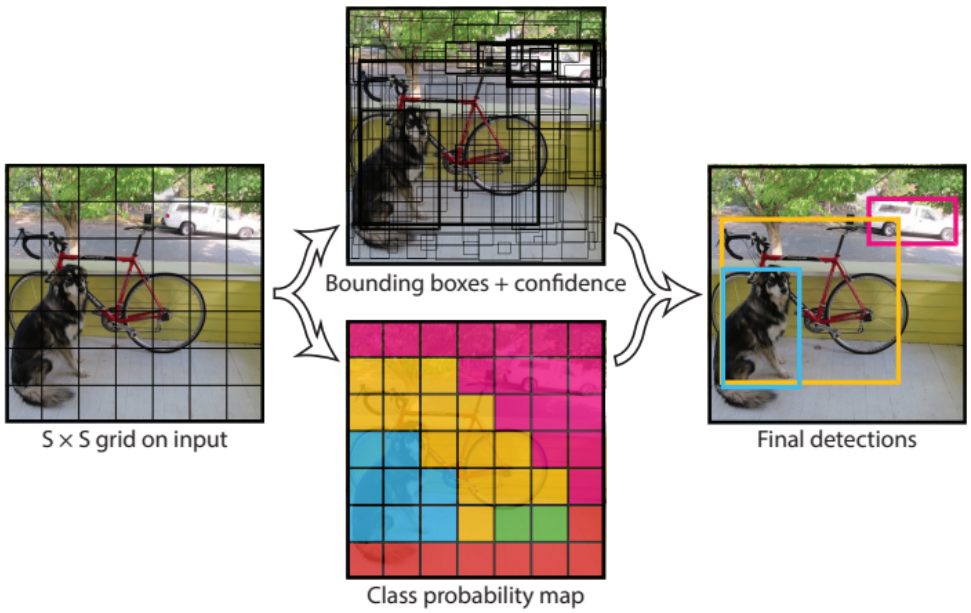

Figura 12 – YOLO. (REDMON et al., 2016)

Na Figura 13 é possível observar a estrutura inicial do modelo YOLO. A rede neural possui 24 camadas convolucionais, seguidas de 2 camadas totalmente conectadas, alternando entre 1 x 1 camadas para que reduza os espaços das camadas anteriores. As camadas convolucionais da CNN do modelo YOLO foram pré-treinadas na arquitetura de classificação de imagens ImageNet, onde nesse pré-treinamento as imagens de entrada foram treinadas com a metade da sua resolução (REDMON et al., 2016). As camadas convolucionais iniciais da rede extraem características da imagem enquanto as camadas totalmente conectadas preveem as probabilidades e as coordenadas da imagem de saída.

As funções de ativações utilizadas para os nós da CNN do modelo YOLO foram duas (REDMON et al., 2016): uma função de ativação linear para a camada final e as demais camadas fizeram uso da função de ativação linear com vazamento:

$$\phi(x) = \begin{matrix} x, if x > 0 \\ 0.1x, otherwise \end{matrix} \tag{3.1}$$

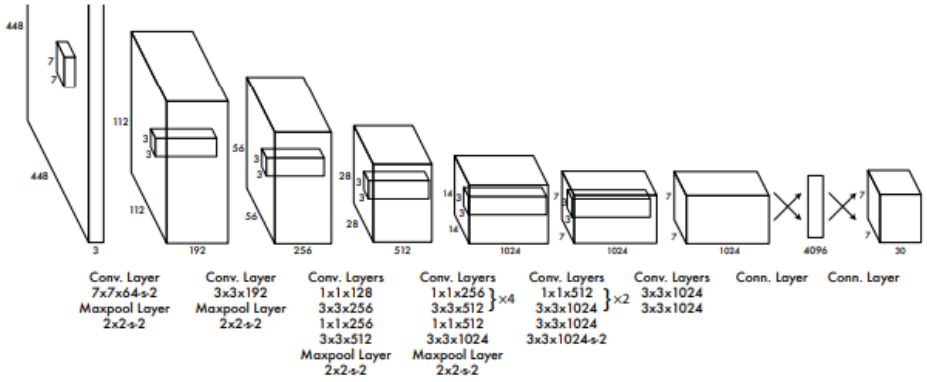

Figura 13 – Arquitetura YOLO. (REDMON et al., 2016)

No momento do treinamento da rede YOLO, cada imagem de entrada acaba localizando muitas células de grades que não contém objetos na predição (ZHAO et al., 2019). Isso faz com que o nível de confiança dessas células sejam quase zero, superando os níveis das células que realmente contém objetos na sua caixa delimitadora e isso causa uma instabilidade para o resultado do treinamento da rede. O tratamento desta adversidade é feito com o incremento da quantidade de perda das caixas delimitadoras que fizeram a predição de objetos e diminui a quantidade de perda para as caixas que não contém objetos (REDMON et al., 2016). Foram usados dois parâmetros para corrigir isso, $\lambda_{coord} = 5$ e $\lambda_{noobj} = 5$.

As *Loss Functions* utilizadas para otimização do modelo foram divididas da seguinte forma:

$$
\begin{aligned}
\lambda_{coord} \quad & \sum_{i=0}^{S^2} \sum_{j=0}^{B} \Gamma_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
+ \quad & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \Gamma_{ij}^{obj} [(\sqrt{\omega_i} - \sqrt{\hat{\omega}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
+ \quad & \sum_{i=0}^{S^2} \sum_{j=0}^{B} \Gamma_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
+ \lambda_{noobj} \quad & \sum_{i=0}^{S^2} \sum_{j=0}^{B} \Gamma_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
+ \quad & \sum_{i=0}^{S^2} \Gamma_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2)
\end{aligned}
\tag{3.2}
$$

onde Γ_i^{obj} mostra se o objeto está presente na célula i e Γ_{ij}^{obj} mostra se o preditor da jth caixa delimitadora na célula i é o preditor que realiza a previsão do objeto no local (REDMON et al., 2016). O S representa as grades que dividem a imagem, B representa as caixas de predição, C representa as classes preditas e ω e h são as dimensões das caixas.

A função de perda da YOLO é dividida em três partes: uma primeira que é responsável por encontrar as coordenadas da caixa delimitadora, a segunda responsável pela previsão da marcação de uma caixa e a última responsável pela previsão da marcação de alguma classe, Figura 14. Todas elas são funções de perdas quadráticas e são moduladas por uma pontuação entre a previsão e o pixel verdadeiro da imagem (O'Reilly Media, 2022).

Figura 14 – Função de perda. (O'Reilly Media, 2022)

- coordenadas das caixas delimitadoras:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \Gamma_{ij}^{obj}[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \Gamma_{ij}^{obj}[(\sqrt{\omega_i} - \sqrt{\hat{\omega}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

- previsão da marcação da caixa:

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} \Gamma_{ij}^{obj}(C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \Gamma_{ij}^{obj}(C_i - \hat{C}_i)^2$$

- classificação da caixa:

$$\sum_{i=0}^{S^2} \Gamma_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2) \tag{3.3}$$

A *Loss Function* é aplicada apenas se for identificado um objeto na célula da grade, penalizando também o locais em que as caixas delimitadoras são colocadas, por conta de supostos objetos identificados.

A vantagem da YOLO frente aos outros métodos é que esse faz as predições da classe com uma única passada na rede. Antes dele, esses outros principais sistemas de detecção de objetos faziam a detecção através da divisão da imagem em várias partes e depois em cada pedaço da imagem se executava um classificador em cada uma dessas regiões (ZHAO et al., 2019). O classificador pode ser transformado em um detector de objetos se for usado uma janela que se desloca pela imagem. Em cada etapa é rodado um classificador para obter uma predição a respeito de que tipo de objeto está presente dentro da janela. Essa métrica pode retornar centenas ou milhares de predições para aquela imagem, mas é mantido apenas aquelas que o classificador possui uma certeza maior. Isso significa que é necessário rodar o mesmo classificador dezenas ou milhares de vezes sobre a mesma imagem. Comparando com esses outros métodos, a YOLO possui uma abordagem completamente diferente. Ele não é um classificador tradicional que foi reajustado para ser um detector de objetos, mas sim foi feito para ser capaz de rodar por toda a imagem apenas de uma vez, fazendo isso de uma forma inteligente.

Lançada no ano de 2015, o modelo foi logo reconhecido por apresentar um método novo e inovador que impactou na sua rapidez durante processo de detecção juntamente com a boa eficácia apresentada no modelo (REDMON et al., 2016). A YOLO é uma das funcionalidades de Visão Computacional que mais ganhou destaque nos últimos anos. Em uma breve comparação, enquanto outros modelos que possuem uma alta taxa de acurácia levam por volta de 0.5 segundos para processamento de uma imagem o YOLO consegue detectar com o mesmo nível de precisão em menos de 0.05 segundos.

Na YOLO cada célula da grade faz a predição de no máximo duas caixas delimitadoras e somente uma classe dentro dessas caixas, essa limitação faz com que o modelo seja um pouco restrito quando queremos detectar muitos objetos que estão próximos uns aos outros (REDMON et al., 2016). Portanto, pequenos objetos que podem aparecer em bando, como pequenos animais, é um desafio para o modelo inicial da rede YOLO.

Por ser um modelo de código totalmente aberto, o que é mais um dos fatores do seu grande sucesso, a YOLO recebe atualizações diversas no seu algoritmo. Portanto após 7 anos do seu lançamento a YOLO possui várias versões do seu modelo, dentre elas as suas principais versões são:

- YOLOv1 (Jun, 2015): primeira versão;
- YOLOv2 (Dec, 2016): a segunda versão da YOLO teve como objetivo melhorar as deficiências da primeira versão, como por exemplo erros de localização de objetos, principalmente de pequeno porte. Essas melhorias foram feitas de maneira que não afetasse a velocidade do modelo, que é a característica mais impactante com relação aos outros modelos (REDMON; FARHADI, 2017). Entre as melhorias estão:

  - Normalização em lote: foi adicionada uma nova forma de normalização a arquitetura YOLO, aumentando a velocidade do treinamento e eliminando a necessidade de outros tipos de normalização, como o overfitting.
  - Classificador de alta resolução: a nova versão da YOLO passou a treinar as imagens em resolução mais alta (448 x 448) por 10 épocas, fazendo com que a rede consiga ajustar os filtros para uma resolução mais alta, aumentando a acurácia da rede.
  - Caixas ancoradas para as caixas delimitadoras: foi retirado o uso das camadas totalmente conectadas para prever as caixas delimitadoras em vez de prever coordenadas diretamente da rede convolucional. Foram adicionadas caixas de ancoragem para prever as caixas delimitadoras.
  - Treinamento de multi-escala: foi treinada em diferentes tamanhos de resolução de entrada 320 x 320 a 608 x 608. A escolha da dimensão é feita aleatoriamente a cada 10 épocas, fornecendo mais precisão a 90 de FPS.

- YOLOv3 (Apr, 2018): a YOLOv3 é uma versão aprimorada da YOLO e da YOLOv2. As grandes diferenças existentes entre o YOLOv3 e as versões mais antigas tem relação com a velocidade, precisão e especificidade das classes. Por ser uma versão que foi desenvolvida dois anos após a anterior houve uma grande evolução entre as versões. A YOLOv2 utiliza Darknet-19 como seu extrator de recursos, enquanto o YOLOv3 agora usa o Darknet-53. O Darknet-53 possui 53 camadas convolucionais em vez das 19 anteriores, tornando-o mais poderoso do que o Darknet-19 e mais eficiente do que os demais, por isso o YOLOv3 é rápido e preciso em termos de acurácia. Houve também uma melhora com relação a detecção de objetos pequenos que é um problema nas versões anteriores do modelo, pois existe imprecisões na detecção desses objetos (REDMON; FARHADI, 2018). A YOLOv3 aumentou a precisão média para objetos pequenos para 13.3%, um grande avanço com relação a YOLOv2. A versão três do modelo usa um novo classificador para previsão de classes durante o treinamento, essa abordagem *multilabel* permite que as classes sejam mais específicas e múltiplas para caixas delimitadoras individuais, diferente das versões anteriores que permitiam que uma caixa possuísse apenas uma classe.

- YOLOv4 (Apr, 2020): melhoria na velocidade de inferência e acurácia, além de ser mais eficiente para rodar em GPUs. A versão quatro do modelo foi disponibilizada por autores diferentes das três primeiras, foi publicada por Alexey Bochkovskiy, Chien-Yao Wang e Hong-Yuan Mark Liao. A maioria dos modelos precisos modernos requerem muitas GPUs para treinamento com um grande lote e fazer isso com uma GPU torna o treinamento muito lento e impraticável. A YOLOv4 resolve esse problema criando um detector de objetos que pode ser treinado em uma única GPU com um tamanho de lote menor. Isso possibilita treinar um detector de objetos

super-rápido e preciso com uma única GPU (BOCHKOVSKIY; WANG; LIAO, 2020). A YOLO v4 alcança resultados de última geração em velocidade de tempo real no conjunto de dados COCO com 43,5% de acurácia média rodando a 65 FPS em uma GPU Tesla V100, por exemplo, conforme mostrado na Figura 15.

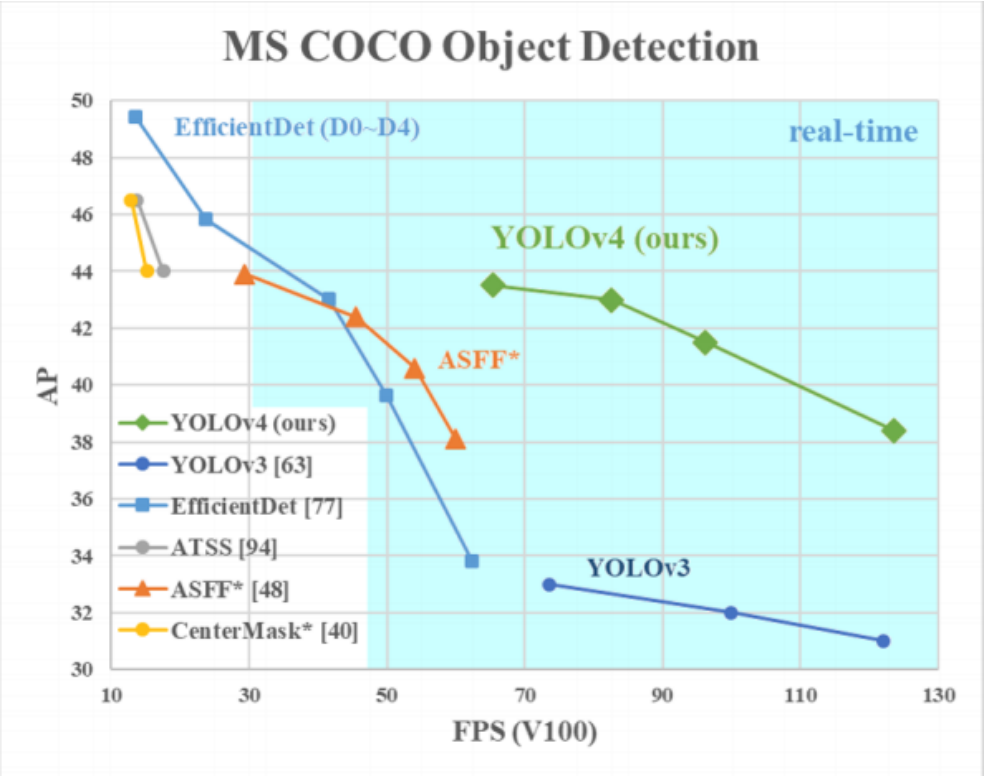

Figura 15 – YOLOv4 - *C*OCO Dataset. (BOCHKOVSKIY; WANG; LIAO, 2020)

- YOLOv5 (May, 2020): apesar de não existir nada documentado sobre a quinta versão do modelo, identificou-se melhorias na inferência, na precisão e reduziu o tamanho do modelo diminuindo o número de parâmetros, sendo a mudança mais importante, pois acaba impactando diretamente na velocidade com que o modelo consegue atuar. Juntamente com essas mudanças houve uma nova evolução para detecção de objetos pequenos.

  A arquitetura da YOLOv5, possui três pontos principais. O primeiro é a construção das camadas *Backbone*, formada por uma rede neural convolucional, é responsável por extrair características das imagens de entrada, Figura 16. Na YOLOv5, as camadas Backbone implementam o *Cross Stage Partial Networks* (CSPDarknet53), utilizando uma versão modificada para a arquitetura da rede (WANG et al., 2020). O intuito do CSP é mitigar os efeitos do problema de duplicação do gradiente proveniente de outras redes convolucionais resultando em um menor número de parâmetros, diminuindo assim, as computações feitas pela rede e aumentando a acurácia dela. O segundo ponto são as camadas Neck, cujo objetivo é combinar as características extraídas das camadas Backbone para passá-las para a fase de predição. A arquitetura

da YOLOv5 utiliza a *Path Aggregation Network* (PANet), que melhora o processo de segmentação da imagem, conseguindo preservar as informações espaciais. Por último, temos um conjunto de camadas *Head*, que são camadas que recebem as características das camadas *Neck* para realizar as predições das caixas delimitadoras e das classes pertencentes de cada objeto (LIU et al., 2018).

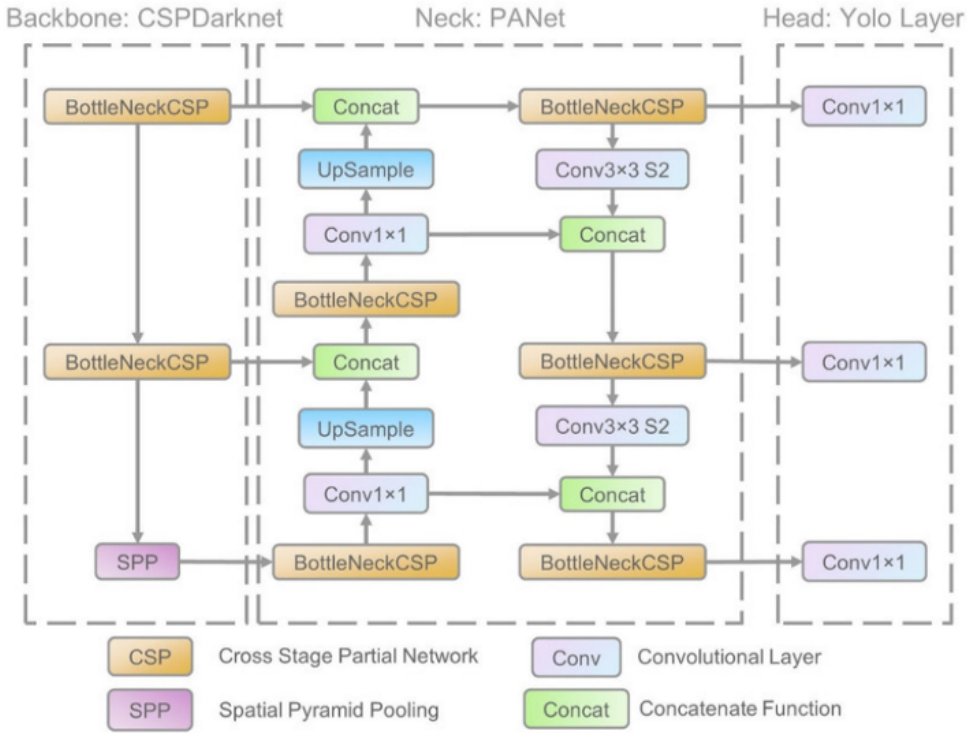

Figura 16 – Arquitetura YOLOv5. (Abhay Mishra, 2022)

Não existe uma confirmação oficial de uma sexta versão da YOLO, porém são encontrados modelos da arquitetura nomeados de YOLOv6, assim como alguns gráficos e tabelas confirmando o avanço da YOLO com relação a precisão e tempo na detecção de objetos comparados com a YOLOv5, como mostrado na Figura 17 retirada de um repositório que contém uma sexta versão do modelo.

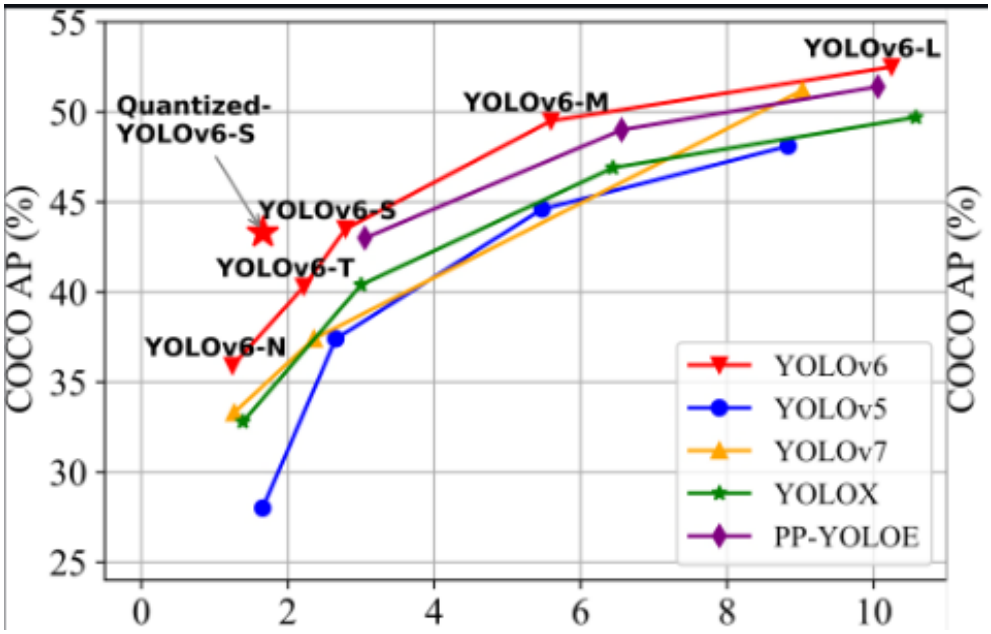

Figura 17 – Resultados - YOLOv6.

4 Desenvolvimento

Nesta seção será apresentado as ferramentas utilizadas para desenvolvimento dos testes realizados e como foi feito os testes para comparação dos modelos.

4.1 Ferramentas

4.1.1 COCO *Dataset*

A realização dos testes englobou 7500 imagens retiradas do *Dataset* COCO (Common Objects in Context) que é um conjunto de dados preparado para detecção, segmentação e rotulagem de objetos em grande escala, (Tsung-Yi Lin, 2022). Foram selecionadas 100 imagens de 75 diferentes classes de objetos disponibilizadas pelo Dataset. Na Figura 18 é mostrada a interface do COCO onde é possível selecionar quais as classes dos objetos e as imagens encontradas através desse filtro.

Foram selecionadas 100 imagens de 75 diferentes classes de objetos disponibilizadas pelo *Dataset*, totalizando um total de 7500 imagens. Essas classes são: *person, handbag, tie, umbrella, suitcase, bicycle, backpack, motorcycle, bus, train, boat, truck, car, traffic light, stop sign, bench, dog, fire hydrant, bird, cat, horse, sheep, cow, elephant, bear, giraffe, zebra, surfboard, baseball glove, tennis racket, skateboard, kite, baseball bat, snowboard, sports ball, bowl, knife, cup, bottle, wine glass, fork, spoon, hot dog, donut, cake, pizza, broccoli, carrot, apple, orange, sandwich, banana, chair, potted plant, dining table, toilet, bed, couch, tv, mouse, keyboard, cell phone, remote, laptop, microwave, toaster, refrigerator, oven, book, vase, clock, scissors, hair drier, teddy bear* e *toothbrush.*

4.1.2 *LabelImg*

Para anotação das classes dentro das imagens foi utilizado o *software LabelImg*, muito popular na área de detecção de objetos em imagem, é uma ferramenta de rotulagem de dados de código aberto que permite selecionar vários detalhes em uma imagem, (heartexlabs, 2022). Neste trabalho a ferramenta foi utilizada para marcar o pixel da imagem onde está presente a classe do objeto requisitado, como excemplificado na 20. Após a marcação, o *software* salva essa anotação em um arquivo .txt que é formado pelas coordenadas da localização do objeto na imagem, Figura 19, com os detalhes que serão utilizados no treinamento da rede.

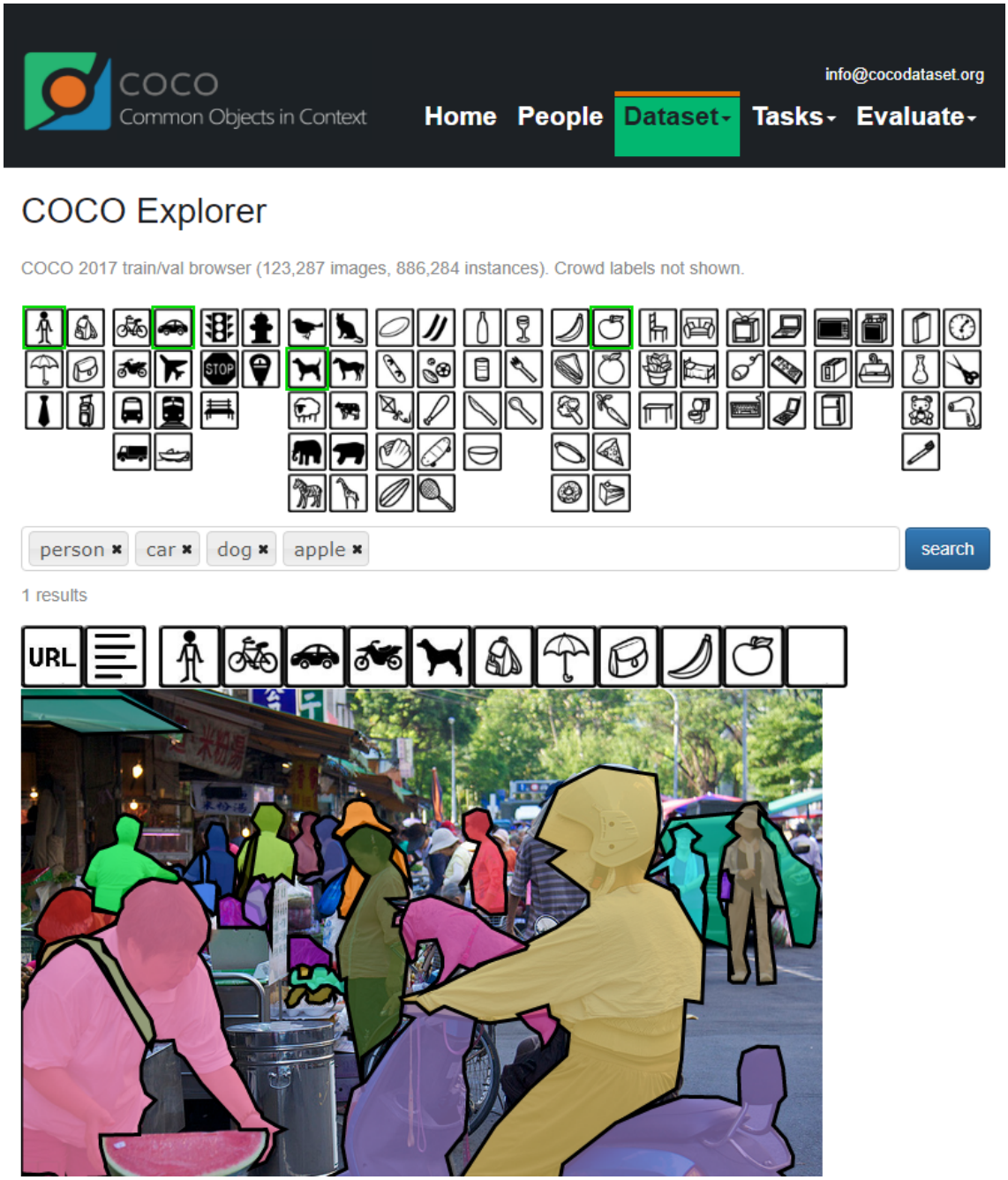

Figura 18 – COCO *Dataset*.

4.1.3 *Google Colab*

O ambiente principal para realizar os testes de treinamento do *Dataset* foi o *Google Colab (Google Colaboratory)*, ferramenta gratuita que permite desenvolver aplicações de *Machine Learning* e *Deep Learning* pela nuvem. O *Colab* disponibiliza o uso remoto de uma GPU, para serem processados os treinamentos. A escolha da placa de vídeo é definida de forma automática pelo *Colab* e a disponibilizada para realização deste trabalho foi Tesla T4 de 16gb de VRAM, com média de limite de uso diário de 12 horas ou então se ultrapassar o limite de memória estabelecido pelo ambiente, Figura 21.

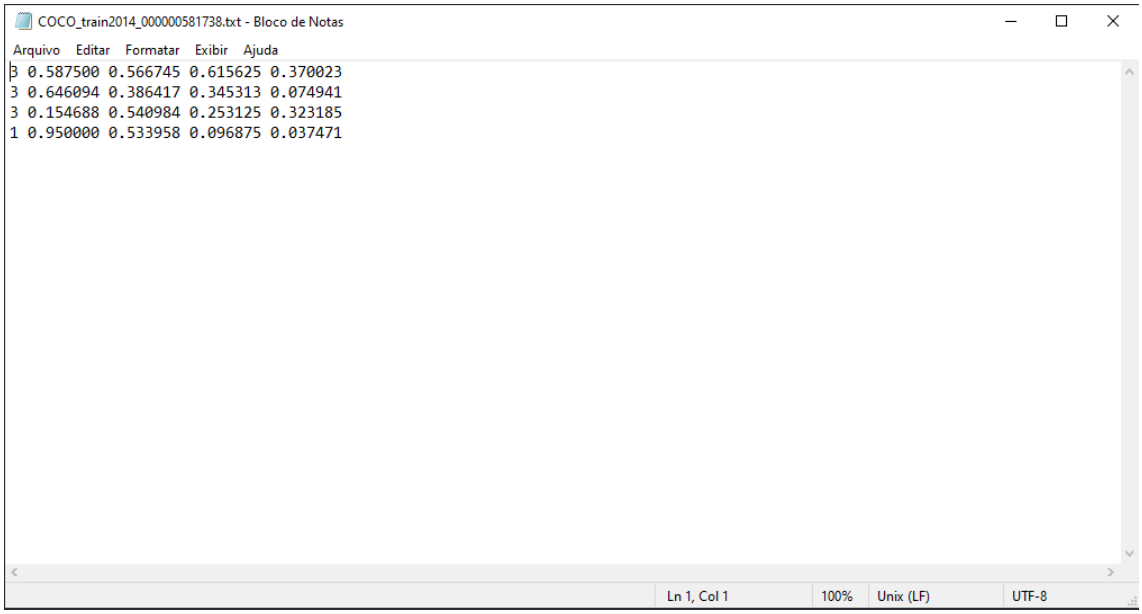

Figura 19 – Arquivo de texto - *LabelImg*.

4.2 Métricas de avaliação

Para avaliação da qualidade e acurácia da detecção dos objetos no treinamento, foram usadas algumas métricas. Abaixo é possível ver como funciona qualquer modelo de treinamento através dos acertos e falhas (Von Wangenheim, 2022), mostrando a diferença entre expectativa e realidade da classificação.

- O Verdadeiro Positivo (VP) representa uma classificação correta da classe Positiva.
- O Falso Positivo (FP) representa uma classificação errada para a classe Positiva, quando o resultado real era para ser da classe Negativa.
- O Verdadeiro Negativo (VN) representa uma classificação correta da classe Negativa.
- O Falso Negativo (FN) representa uma classificação errada para a classe Negativa, quando o resultado real era para ser da classe Positiva.

Com essas informações é possível calcular algumas métricas de classificação, que são acurácia, precisão, recall e o *F1-Score*.

A acurácia tem como função fornecer a visão geral do desempenho do modelo e indicar quão bem foi feita a classificação, é expressa pela fórmula abaixo:

$$acurácia = \frac{VP + VN}{VP + VN + FP + FN} \tag{4.1}$$

A precisão tem como função medir as classificações da classe positiva, fornecendo uma visão geral de quantas estão corretas, é expressa pela fórmula abaixo:

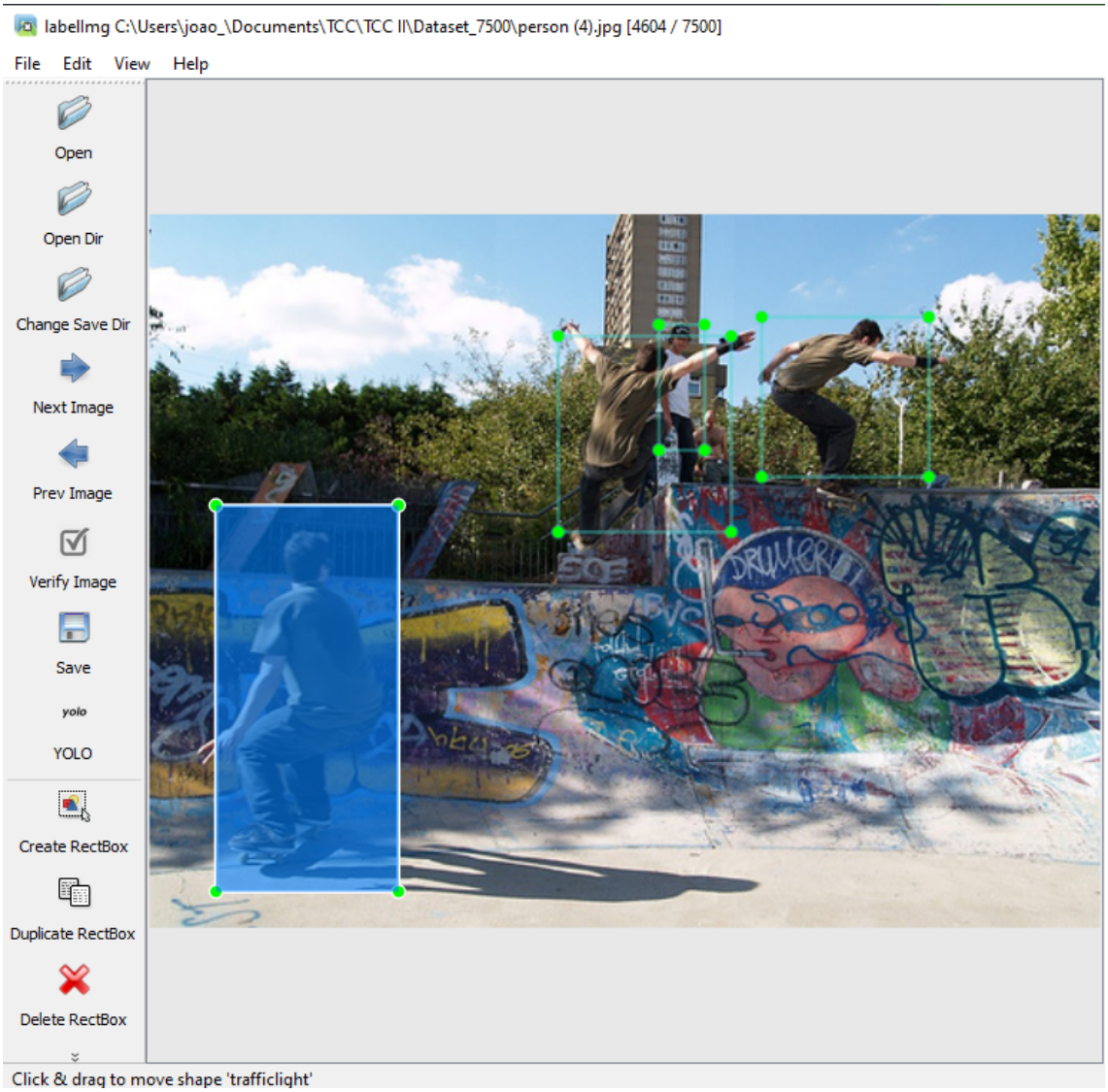

Figura 20 – *LabelImg*.

$$precisão = \frac{VP}{VP + FP} \tag{4.2}$$

O recall tem como função, tomar todas as classes positivas capturadas e qualificar quantas estão corretas, é expressa pela fórmula abaixo:

$$recall = \frac{VP}{VP + FN} \tag{4.3}$$

A *F1-Score* tem como função realizar a média harmônica entre a precisão e o recall, é expressa pela fórmula abaixo:

$$F1\text{-}Score = 2 * \frac{precisão * recall}{precisão + recall} \tag{4.4}$$

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   51C    P8    10W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Figura 21 – GPU *Google Colab*.

Alguns avaliadores tentam determinar a qualidade da detecção, procurando as semelhanças no que foi marcado na imagem e o que está previsto nela. Com base nessas métricas, existe o mean average precision (mAP). mAp é uma técnica de avaliação muito difundida atualmente no campo de Deep Learning. Sua principal característica é poder comparar diferentes modelos, contrapondo a precisão com o recall.

Como já mencionado, as classes classificadas correta e incorretamente são levadas em consideração no cálculo da precisão. Ou seja, é a associação entre as classificações corretas com todas as classificações. No recall o cálculo é focado em todos os objetos que estão na imagem, inclusive as classes que não foram classificadas. A unção é descrita pela relação entre a classificação correta e o número total de objetos na imagem. Portanto, com base na curva de precisão e recall, tem-se a AP (Jonathan Hui, 2022).

Na Figura 22 tudo que está para baixo da curva representa o AP. Mediante isto, a AP corresponde a precisão média de todos os valores de recall entre 0 e 1. O mAP é expresso pela interpolação de precisão com 11 pontos com recall. Por fim, considera-se mAP como a média da AP entre as classes do modelo, (João Gustavo A. Amorim, 2022).

4.3 Experimentos e resultados

Neste capítulo é destinado a mostrar os experimentos que foram feitos com os modelos YOLOv2, YOLOv3, YOLOv4 e YOLOv5 com o *Dataset* customizado que foi retirado do COCO e a explanação dos resultados e um comparativo deles.

É preciso realizar a instalação do *Darknet* que já foi abordado anteriormente. Existe uma maneira simples de realizar isso, clonando o repositório da *Darknet*. Após

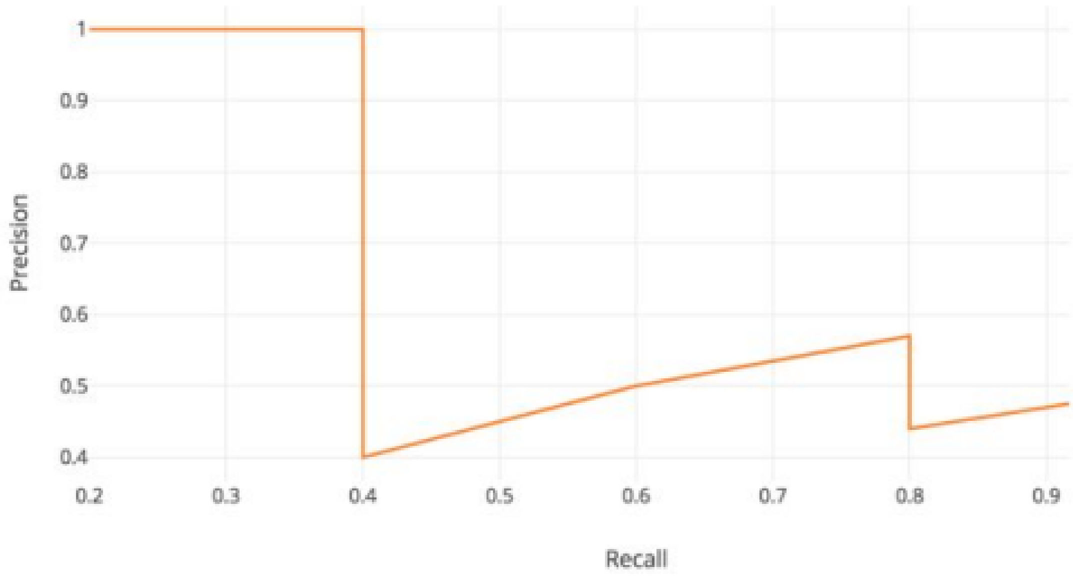

Figura 22 – Curva AP. (Jonathan Hui, 2022)

o clone, basta entrar na pasta e executar o *Makefile* do projeto, que é onde algumas configurações importantes ficam armazenadas, mostrado na Figura 23. Esse script teve algumas alterações que são necessária para o desenvolvimento dos testes.

Figura 23 – Instalação *Darknet*

Para realizar o processo de treinamento foi necessário baixar alguns pesos já treinados no *Dataset COCO* completo, que são utilizados como parâmetro para o resultado final do meu treinamento. A linha de código para realizar o treinamento é exemplificado na Figura 24. É passado o arquivo de data, onde está especificado a quantidade de classes a ser treinado e o caminho dos arquivos de treino e validação. O arquivo de configuração (.cfg) também é passado, é nele onde estão todos os parâmetros e as especificações das camadas convolucionais que vão treinar a rede. E por último é passado o peso pré-treinado.

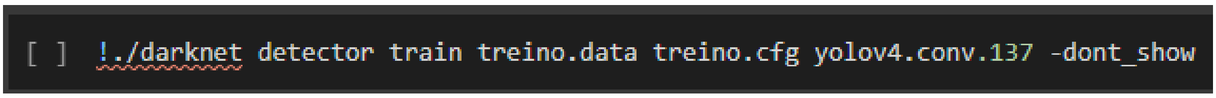

Figura 24 – Treinamento

Nas tabelas 1, 2, 3 e 4 é possível observar os parâmetros utilizados para o treinamento da rede dos modelos YOLOv2, YOLOv3, YOLOv4 e YOLOv5, respectivamente.

batch	16
subdivisions	16
width	416
height	416
channels	3
learning rates	0.001
max batches	500
steps	400 e 450
camadas convolucionais	23

Tabela 1 – Parâmetros da YOLOv2.

batch	16
subdivisions	16
width	416
height	416
channels	3
learning rates	0.001
max batches	500
steps	400 e 450
camadas convolucionais	75

Tabela 2 – Parâmetros da YOLOv3.

batch	16
subdivisions	16
width	416
height	416
channels	3
learning rates	0.001
max batches	500
steps	400 e 450
camadas convolucionais	110

Tabela 3 – Parâmetros da YOLOv4.

Batch é um parâmetro que define o número de amostras a serem trabalhadas antes de atualizar os parâmetros do modelo interno. *Subdivisions* são as pequenas divisões feitas na *batch*, vai variar esse parâmetro de acordo com a capacidade da GPU para que não ocorra extouro de memória. *Width* e *height* é o tamanho que o modelo vai usar pra redimensionar o tamanho da imagem para o treinamento. Os *channels* são a quantidade dos canais de cores da imagem. *Max batches* é a quantidade de épocas do treinamento, ou seja, o número de vezes que o modelo treinará todo o *Dataset*. *Learning rates* é a taxa de aprendizado que determina o ritmo em que os pesos da CNN vão ser atualizados. Os *steps* é um parâmetro que realiza uma atualização no gradiente durante as épocas de treinamento. E por último é o número de camadas convolucionais existentes no modelo.

Após a execução dos treinamentos com os modelos e os parâmetros ajustados

batch	16
subdivisions	16
width	640
height	640
channels	3
max batches	500
steps	400 e 450
camadas convolucionais	215

Tabela 4 – Parâmetros da YOLOv5.

anteriormente, os resultados obtidos foram os seguintes:

4.3.1 YOLOv2

A segunda versão da YOLO, foi o treinamento onde foi obtido o pior resultado com relação a precisão na detecção dos objetos, conforme mostrado na Figura 25. O valor de precisão não foi determinado, o mesmo aconteceu com o valor do *F1-Score*, o valor de *recall* foi 0.00. Isso acabou resultando no mAP final de 2.06% apenas, sendo um valor muito abaixo com relação aos demais modelos. Esse resultado extremamente ruim pode ser explicado pelo baixo número de camadas convolucionais usadas para o treinamento da rede, ocasionando assim um baixo número de interações no momento de classificar as classes dos objetos. O treinamento da rede obteve um *average loss* de 2.17, podendo ser observado na Figura 26.

```
for conf_thresh = 0.25, precision = -nan, recall = 0.00, F1-score = -nan

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.020598, or 2.06 %
```

Figura 25 – Resultado da YOLOv2.

O desenvolvimento do modelo YOLOv2 pode ser encontrado nos links do *notebook* e do *Google Drive* a seguir:

- *Notebook do Colab*: https://colab.research.google.com/drive/1AOhsDD8Ml1jfgTkI4DfoFlF8G-JdHwRP?usp=sharing
- Pasta no *Google Drive*: https://drive.google.com/drive/folders/1ho0r3J-kNcct5TQqAkexGd4z6dKTtcKa?usp=sharing

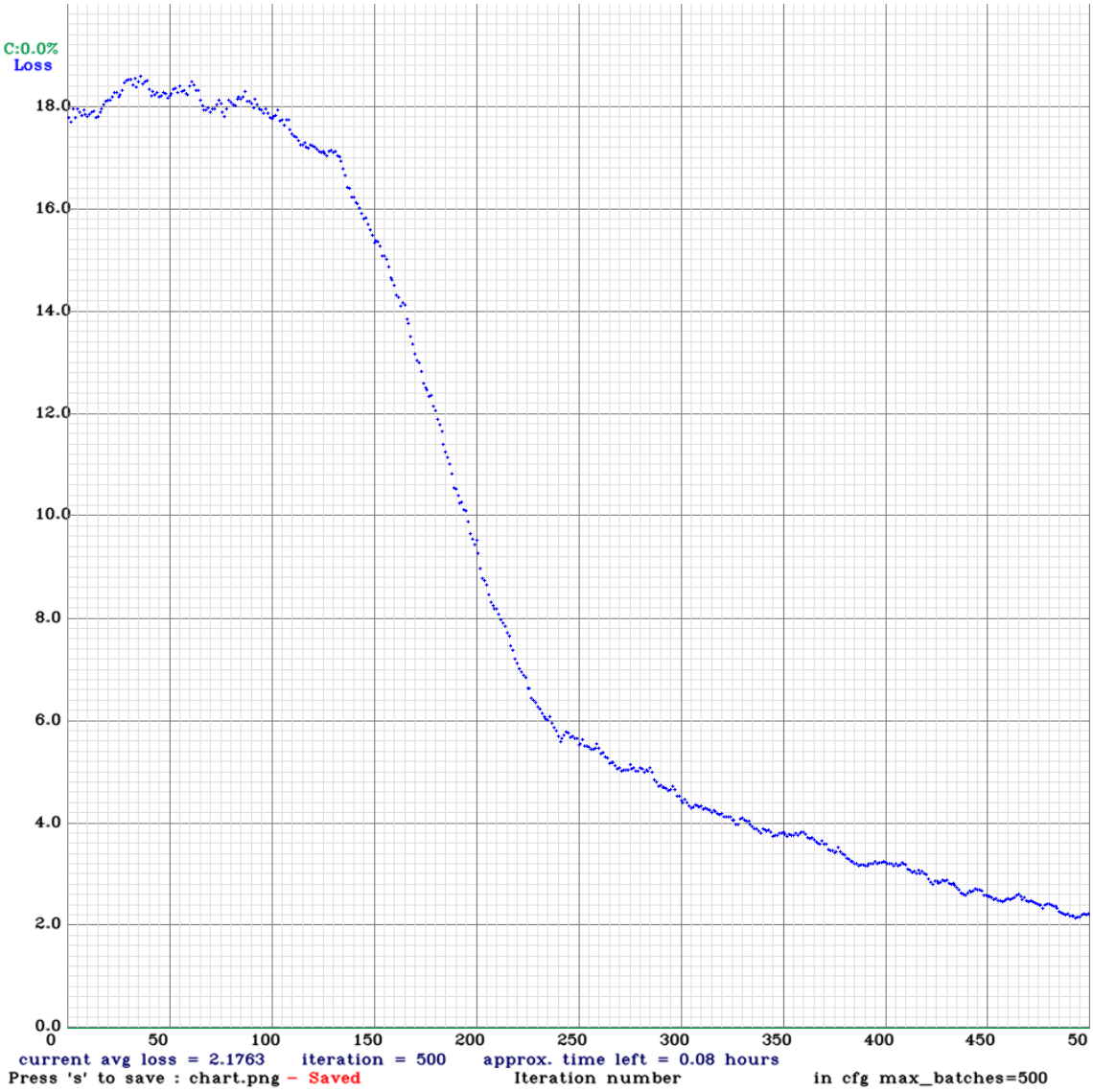

Figura 26 – Curva de *loss* YOLOv2.

4.3.2 YOLOv3

No segundo teste realizado na YOLOv3 o resultado obtido foi bem melhor do que no primeiro treinamento, subindo dos 2.06% para 27% de mAP, com precisão de 0.46, o valor de *recall* foi 0.50 e o *F1-Score* foi 0.48, Figura 27. Essa melhora muito significativa com relação a YOLOv2 pode ser explicada pelo número de camadas convolucionais utilizadas para o treinamento da YOLOv3 ser mais do que triplo do modelo anterior. O *average loss* desse treinamento foi de 3.08 ao final do treinamento da rede, Figura 28.

```
for conf_thresh = 0.25, precision = 0.46, recall = 0.50, F1-score = 0.48

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.270022, or 27.00 %
```

Figura 27 – Resultado da YOLOv3.

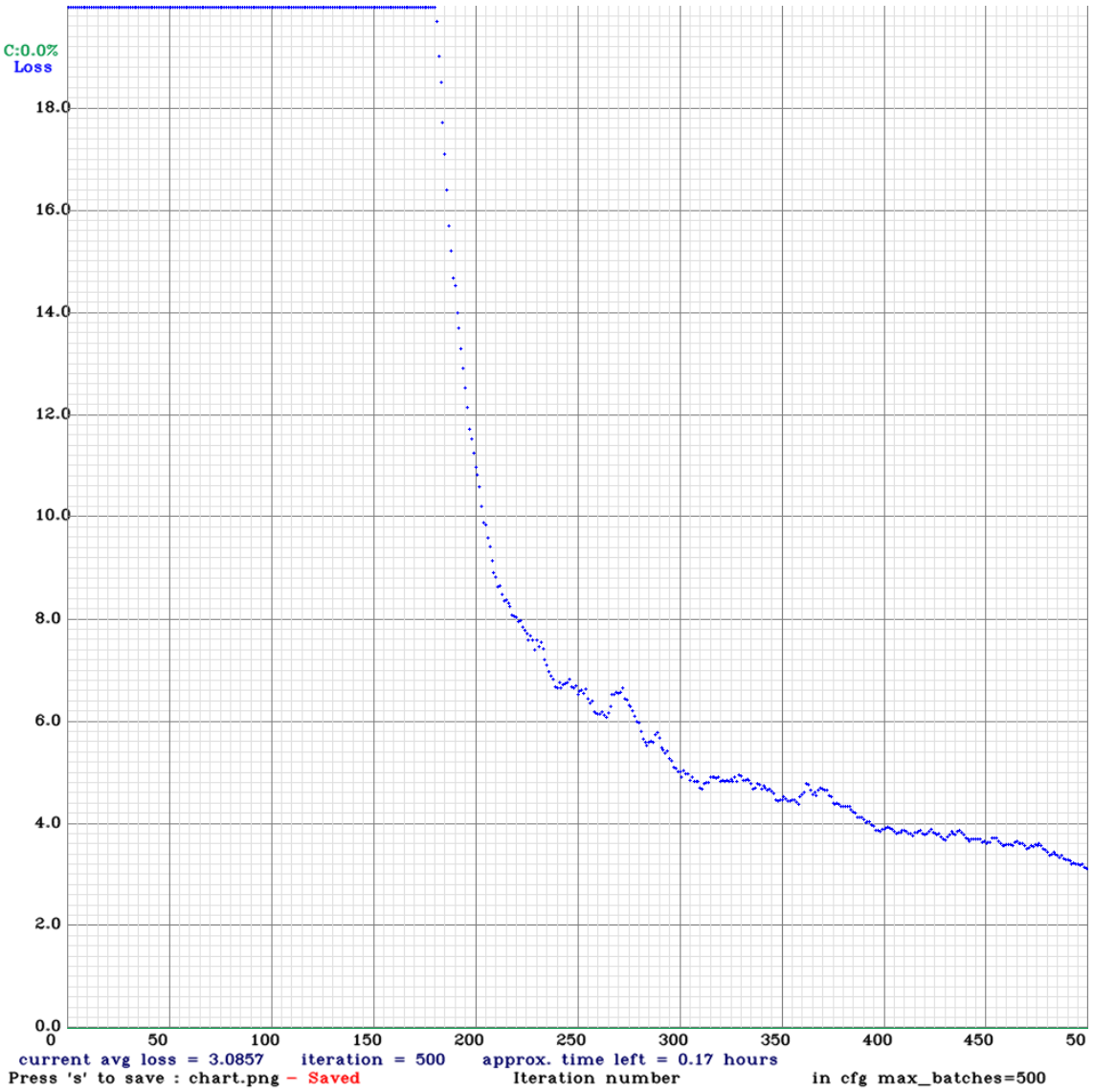

Figura 28 – Curva de *loss* YOLOv3.

O desenvolvimento do modelo YOLOv3 pode ser encontrado nos links do *notebook* e do *Google Drive* a seguir:

- *Notebook do Colab*: https://colab.research.google.com/drive/1QqHzEOnmI6ysFxkXJ8kEbDmUoxyZ8hGD?usp=sharing
- Pasta no *Google Drive*: https://drive.google.com/drive/folders/14RkAt3q7V9et3hugopCgpLi3hUNYZJXW?usp=sharing

4.3.3 YOLOv4

Na quarta versão do modelo, o treinamento melhorou a precisão apresentando um resultado bem superior as versões anteriores. Na Figura 29 mostra que o valor de precisão encontrado foi de 0.27, o valor de *recall* foi 0.43 e o *F1-Score* foi de 0.33. O mAP final

foi de quase 40%, o que é considerado um resultado não tão ruim se observar o número de camadas convolucionais que foram usadas para o treinamento. O modelo da YOLOv4 obteve um *average loss* de 4.03 sendo maior que os anteriores, Figura 30.

```
for conf_thresh = 0.25, precision = 0.27, recall = 0.43, F1-score = 0.33

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.388896, or 38.89 %
```

Figura 29 – Resultado da YOLOv4.

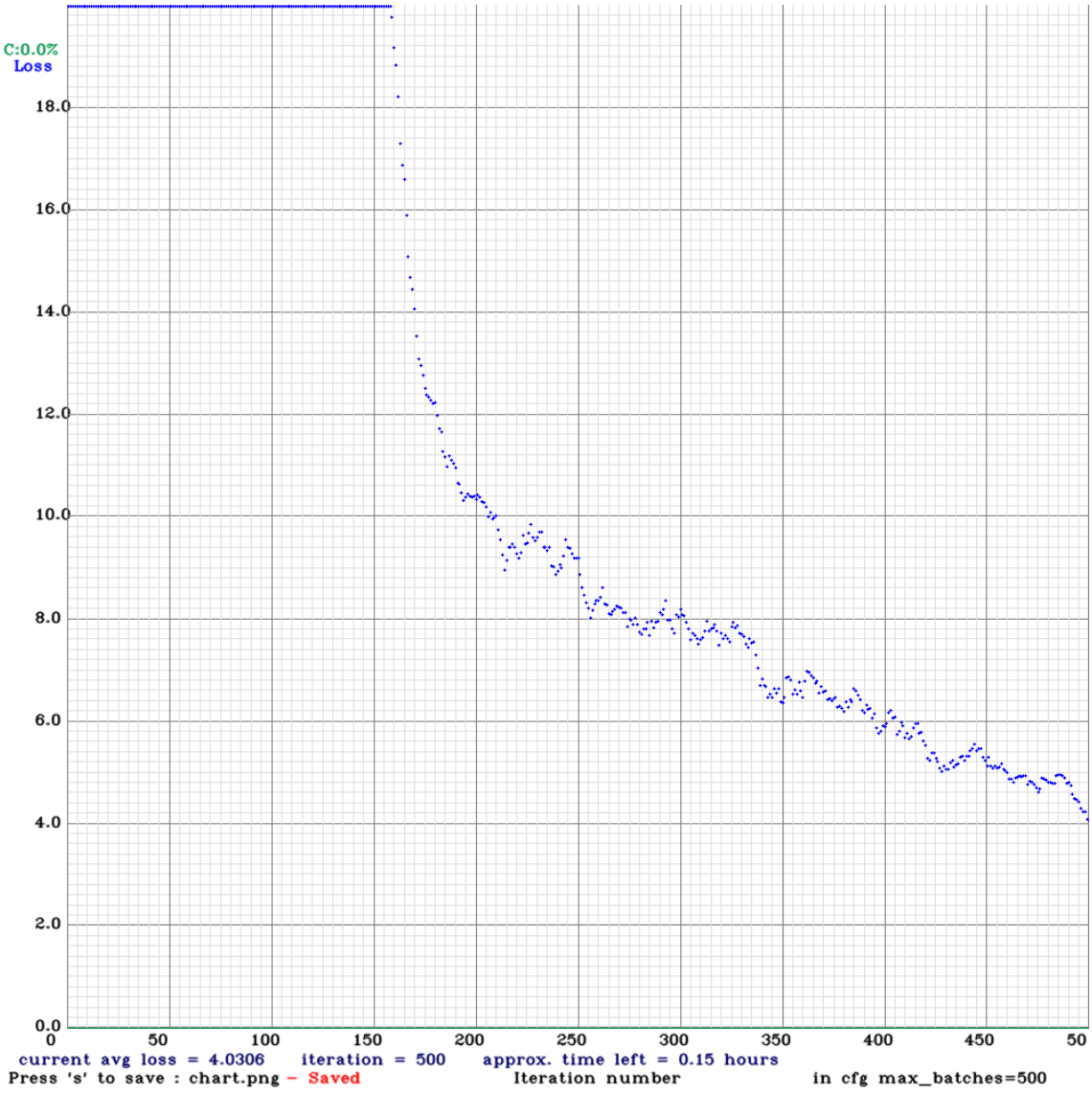

Figura 30 – Curva de *loss* YOLOv4.

O desenvolvimento do modelo YOLOv4 pode ser encontrado nos links do *notebook* e do *Google Drive* a seguir:

- *Notebook do Colab*: https://colab.research.google.com/drive/17pFy9Y2WB1Oym6v6x

to3qynNX_AKOXaa?usp=sharing

- Pasta no *Google Drive*: https://drive.google.com/drive/folders/10cOvf3GqyAhoogKGAkrR8X3TX1Q0EGpR?usp=sharing

4.3.4 YOLOv5

O melhor resultado foi obtido na última versão da YOLO, foi atingido um mAP de quase 70%, mais precisamente 67,9%, conforme mostrado na 31, com 0.819 de precisão, 0.90 de *recall* e 0.67 ded *F1-Score*. Esse resultado bem superior tem relação com o número de camadas convolucionais bem maiores do que os modelos anteriores, sendo quase o dobro da YOLOv4. O interessante é que o treinamento dessa rede foi feito com as dimensões das imagens maiores do que os três modelos anteriores sem que afetasse o desempenho do treinamento. O *average loss* foi quase nulo ao final do treinamento, Figura 32.

```
Validating runs/train/exp/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 7020913 parameters, 0 gradients, 15.8 GFLOPs
               Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  4.94it/s]
                 all        750         51      0.675      0.665      0.679      0.414
```

Figura 31 – Resultado da YOLOv5.

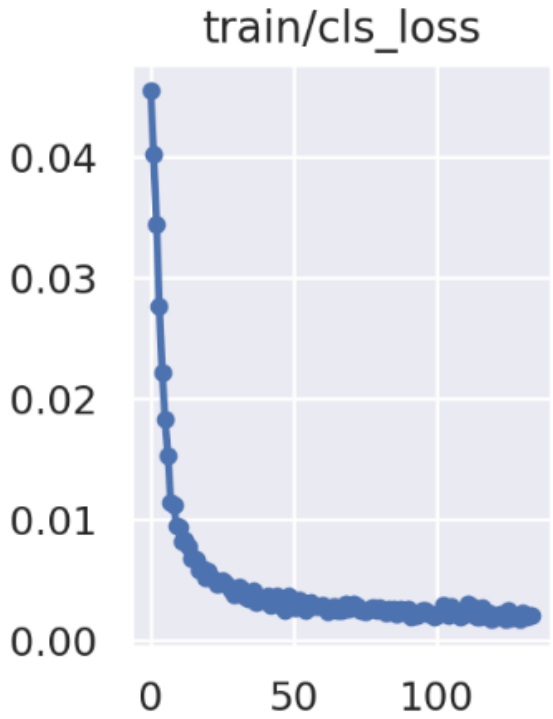

Figura 32 – Curva de *loss* YOLOv5.

O desenvolvimento do modelo YOLOv5 pode ser encontrado nos links do *notebook* e do *Google Drive* a seguir:

Modelo	**mAP**
YOLOv2	2.06%
YOLOv3	27.00%
YOLOv4	38.89%
YOLOv5	67.9%

Tabela 5 – Comparação de mAP

Modelo	**Tempo(min)**
YOLOv2	42
YOLOv3	48
YOLOv4	52
YOLOv5	25

Tabela 6 – Tempo de treinamento

- *Notebook do Colab*: https://colab.research.google.com/drive/13r59pxZViZjyZ0SW9HVx-r6T6q4QR5G5?usp=sharing
- Pasta no *Google Drive*: https://drive.google.com/drive/folders/1pZgzPHbxSlIgFgz-W7zwE5k0B3JR5NL-?usp=sharing

4.3.5 Comparação entre os modelos

Realizando uma comparação geral entre os modelos, podemos observar na tabela 5 os mAP obtidos no treinamento de todos os modelos, onde conforme falado anteriormente o pior resultado foi a YOLOv2 e o melhor foi a YOLOv5.

Com relação aos tempos de treinamento de cada rede, na tabela 6 é possível observar o desempenho aproximado de cada rede. É importante lembrar que o número de camadas interfere no tempo de treinamento.

Os tempos de treinamento dos três primeiros modelos treinados são bem parecidos, pois apesar do número de camadas serem diferentes o número reduzido do *Dataset* utilizado acaba influenciando para não divergir tanto esse tempo. Com relação ao tempo da YOLOv5, ele é justificado pelo treinamento ter sido interrompido na época 234 pois o modelo identificou que não estava havendo avanço nos resultados das últimas 100 épocas treinadas.

Figura 33 – Detecção pessoa - YOLOv2

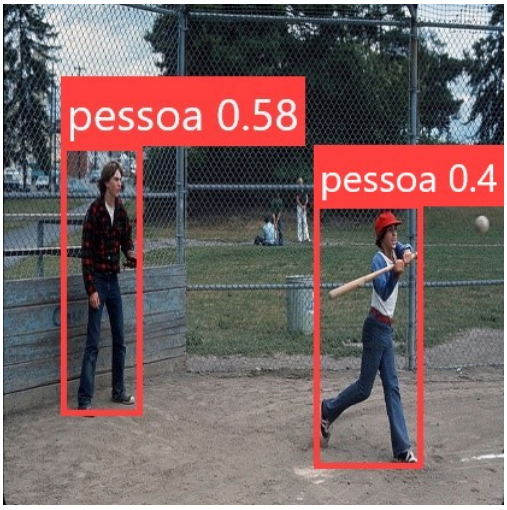

Figura 34 – Detecção pessoa - YOLOv3

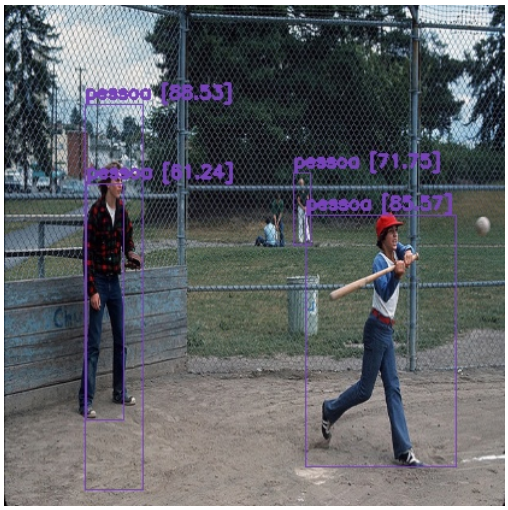

Figura 35 – Detecção pessoa - YOLOv4

Figura 36 – Detecção pessoa - YOLOv5

Agora falaremos sobre os testes na prática da detecção de objetos em uma imagem para exemplificar os resultados alcançados nos treinamentos. Foram selecionadas duas imagens do *Dataset* de treinamento, essas imagens foram escolhidas por conter objetos das classes que obtiveram os melhores resultados no treinamento dos modelos, essas classes são: pessoa, ônibus e carro. O motivo dessas classes obterem um resultado superior as demais é a quantidade de vezes em que elas aparecem nas imagens do *Dataset* e o seu tamanho. Na YOLOv5 que obteve o maior mAP, por exemplo, a classe de pessoa obteve um resultado de 0.721 no seu mAP individual, a de ônibus 0.833 e a classe de carro 0.446.

Na imagem onde existem algumas pessoas é possível observar que os modelos YOLOv4 e YOLOv5 teve pequenas variações com relação a precisão do objeto detectado, com bons resultados em ambos, todas as detecções tiveram precisão acima de 60%. A YOLOv4, porém, teve uma pequena imprecisão onde ele detectou duas pessoas em um local onde existe apenas uma. O modelo YOLOv3 detectou duas pessoas, onde uma teve a precisão acima de 50% e a outra abaixo. Uma pessoa no fundo da imagem que foi detectada

nas versões superiores não foi detectada na terceira versão. E por último a YOLOv2 não teve êxito em detectar nenhuma das pessoas existentes na imagem.

Na imagem onde existem carros e ônibus é possível observar um resultado um pouco melhor do que na detecção das pessoas. Isso ocorre por conta de o ônibus ser um objeto bem maior do que uma pessoa e assim facilita a detecção dele na imagem. Neste exemplo a YOLOv2 conseguiu identificar um objeto pelo menos diferentemente da outra imagem, foi detectado um ônibus com 23% de precisão. Na YOLOv3 foram detectados dois objetos, um carro e um ônibus, com 0.3% e 0.59% respectivamente. A quarta versão da YOLO, diferentemente das outras três versões foi a única que não obteve êxito na detecção do ônibus, porém somente ela conseguiu detectar uma pessoa dentro do ônibus com 58.2% de precisão. Além disso a YOLOv3 cometeu a mesma falha que teve na primeira imagem e detectou dois carros onde só existe um e na detecção verdadeira do veículo ele obteve uma precisão de 61.3%. Por fim, a YOLOv5 foi a versão que obteve os melhores resultados, conseguindo uma precisão de 0.8% no ônibus, 0.9% no mesmo carro que as demais versões detectaram e ainda detectou um segundo carro com 0.3%.

Figura 37 – Detecção veículo - YOLOv2

Figura 38 – Detecção veículo - YOLOv3

Figura 39 – Detecção veículo - YOLOv4

Figura 40 – Detecção veículo - YOLOv5

5 Conclusão

No trabalho apresentado acima, foi feito uma análise do uso das CNNs utilizando as quatro últimas versões oficiais da abordagem YOLO em uma *Dataset* personalizado, buscando confirmar as melhorias realizadas no modelo desde o lançamento da sua primeira versão em 2015. A presente pesquisa apresentou fortes contribuições do ponto de vista teórico sobre o tema, entregando conceitos, termos e os principais métodos existentes na área de detecção de objetos em imagens. Apesar das dificuldades técnicas encontradas no desenvolvimento dos testes, que foram causadas pela limitação do tempo e processamento da GPU disponibilizada pelo *Google Colab*, considera-se que o objetivo proposto foi alcançado, que foi a comparação entre os resultados do treinamento das versões da YOLO.

Desta forma, foi possível verificar através dos estudos e dos testes práticos que a abordagem YOLO realmente apresenta uma proposta interessante, principalmente com as evoluções alcançadas pela rede na sua quinta versão, a YOLOv5. Esta apresentou bons resultados com relação a tempo e precisão nos experimentos realizados, considerando as limitações já citadas e o tamanho reduzido do *Dataset* treinado. Porém, apesar disso é válido ressaltar que a rede ainda possui algumas limitações, principalmente com relação a detecção de pequenos objetos que estão espalhados na imagem junto a outros, como exemplificado nas figuras abaixo.

Para trabalhos futuros, entende-se que pode ser feito um treinamento da YOLOv5 aplicando configurações mais aperfeiçoadas para alcançar resultados melhores, como aumentar o valor da subdivisão, aumentar o número de imagens do *Dataset* e a utilização de um ambiente de treinamento com uma GPU com melhor capacidade de processamento. Com essa melhora no treinamento é possível realizar a integração da CNN treinada junto a uma API de detecção de objetos.

Figura 41 – Detecção de pessoas

Figura 42 – Detecção de pessoas

Referências

Abhay Mishra. *How to Use Yolo v5 Object Detection Algorithm for Custom Object Detection*. 2022. Disponível em: <https://www.analyticsvidhya.com/blog/2021/12/how-to-use-yolo-v5-object-detection-algorithm-for-custom-object-detection-an-example-use-case/>, Acesso em: 20 de agosto 2022. Citado 2 vezes nas páginas 5 e 30.

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. Citado na página 14.

Alberto Rizzoli. *The Ultimate Guide to Object Detection*. 2022. Disponível em: <https://www.v7labs.com/blog/object-detection-guide>. Acesso em: 30 de agosto 2022. Citado 2 vezes nas páginas 5 e 19.

ALPAYDIN, E. *Introduction to machine learning*. [S.l.]: MIT press, 2020. Citado 3 vezes nas páginas 9, 12 e 16.

ANKIT SACHAN. *Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN,YOLO,SSD*. 2022. Disponível em: <https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/>. Acesso em: 19 de agosto 2022. Citado 2 vezes nas páginas 5 e 20.

AWOYEMI, J. O.; ADETUNMBI, A. O.; OLUWADARE, S. A. Credit card fraud detection using machine learning techniques: A comparative analysis. In: IEEE. *2017 international conference on computing networking and informatics (ICCNI)*. [S.l.], 2017. p. 1–9. Citado na página 9.

BISHOP, C. M.; NASRABADI, N. M. *Pattern recognition and machine learning*. [S.l.]: Springer, 2006. v. 4. Citado 2 vezes nas páginas 11 e 16.

BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. Citado 2 vezes nas páginas 5 e 29.

DAI, J. et al. R-fcn: Object detection via region-based fully convolutional networks. *Advances in neural information processing systems*, v. 29, 2016. Citado 2 vezes nas páginas 5 e 23.

FANG, W.; WANG, L.; REN, P. Tinier-yolo: A real-time object detection method for constrained environments. *IEEE Access*, IEEE, v. 8, p. 1935–1944, 2019. Citado na página 23.

FRANÇA, J. A. de et al. Uma implementação do algoritmo levenberg-marquardt dividido para aplicações em visão computacional. *Semina: Ciências Exatas e Tecnológicas*, v. 30, n. 1, p. 51, 2009. Citado na página 9.

Geeks Forge. *Faster R-CNN | ML*. 2022. Disponível em: <https://www.geeksforgeeks.org/faster-r-cnn-ml/>. Acesso em: 18 de outubro 2022. Citado 2 vezes nas páginas 5 e 22.

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. Citado 3 vezes nas páginas 5, 20 e 21.

HARMON, L. D. Artificial neuron. *Science*, American Association for the Advancement of Science, v. 129, n. 3354, p. 962–963, 1959. Citado na página 12.

HASTIE, T. et al. *The elements of statistical learning: data mining, inference, and prediction*. [S.l.]: Springer, 2009. v. 2. Citado 2 vezes nas páginas 11 e 12.

heartexlabs. *labelImg*. 2022. Disponível em: <https://github.com/heartexlabs/labelImg>. Acesso em: 14 de abril 2022. Citado na página 32.

HUANG, R.; PEDOEEM, J.; CHEN, C. Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers. In: IEEE. *2018 IEEE International Conference on Big Data (Big Data)*. [S.l.], 2018. p. 2503–2510. Citado 2 vezes nas páginas 23 e 24.

IBM Cloud Education. *O que são redes neurais?* 2022. Disponível em: <https://www.ibm.com/br-pt/cloud/learn/neural-networks>. Acesso em: 06 de abril 2022. Citado 2 vezes nas páginas 5 e 12.

JAMES, G. et al. *An introduction to statistical learning*. [S.l.]: Springer, 2013. v. 112. Citado 4 vezes nas páginas 13, 16, 17 e 18.

Jonathan Hui. *mAP (mean Average Precision) for Object Detection*. 2022. Disponível em: <https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173>, Acesso em: 22 de agosto 2022. Citado 3 vezes nas páginas 5, 36 e 37.

João Gustavo A. Amorim. *Visão Computacional::Métricas::Mean Average Precision*. 2022. Disponível em: <https://lapix.ufsc.br/ensino/visao/visao-computacionaldeep-learning/visao-computacionalmetricasmean-average-precision/>. Acesso em: 14 de outubro 2022. Citado na página 36.

KOEHRSEN, W. Overfitting vs. underfitting: A complete example. *Towards Data Science*, 2018. Citado na página 18.

KONONENKO, I. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in medicine*, Elsevier, v. 23, n. 1, p. 89–109, 2001. Citado na página 9.

LI, S. et al. Deep learning for hyperspectral image classification: An overview. *IEEE Transactions on Geoscience and Remote Sensing*, IEEE, v. 57, n. 9, p. 6690–6709, 2019. Citado na página 9.

LI, Z. et al. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, IEEE, 2021. Citado 2 vezes nas páginas 11 e 13.

LIU, S. et al. Path aggregation network for instance segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2018. p. 8759–8768. Citado na página 30.

MARENGONI, M.; STRINGHINI, S. Tutorial: Introdução à visão computacional usando opencv. *Revista de Informática Teórica e Aplicada*, v. 16, n. 1, p. 125–160, 2009. Citado 3 vezes nas páginas 5, 9 e 10.

Mateus Coelho. *Fundamentos De Redes Neurais*. 2022. Disponível em: <http://www2.decom.ufop.br/imobilis/fundamentos-de-redes-neurais/>. Acesso em: 11 de agosto 2022. Citado 2 vezes nas páginas 5 e 13.

O'Reilly Media. *Detector Loss function (YOLO loss)*. 2022. Disponível em: <https://www.oreilly.com/api/v2/epubs/9781789130331/files/assets/7b0edf3c-b27d-4d1a-952d-ad10176656fd.png>, Acesso em: 20 de agosto 2022. Citado 2 vezes nas páginas 5 e 26.

PADMANABHAN, J.; PREMKUMAR, M. J. J. Machine learning in automatic speech recognition: A survey. *IETE Technical Review*, Taylor & Francis, v. 32, n. 4, p. 240–251, 2015. Citado na página 9.

PASSOS, E. C.; ANDRADE-NETO, A.; LEMAIRE, T. Comportamento ótico do olho humano e suas ametropias. *Caderno de Física da UEFS*, v. 6, n. 1-2, p. 7–18, 2008. Citado na página 9.

Pedro Ney Stroski. *O que são redes neurais convolucionais?* 2022. Disponível em: <https://www.electricalelibrary.com/2018/11/20/o-que-sao-redes-neurais-convolucionais/>. Acesso em: 11 de agosto 2022. Citado 2 vezes nas páginas 5 e 15.

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. Citado 5 vezes nas páginas 5, 20, 24, 25 e 27.

REDMON, J.; FARHADI, A. Yolo9000: better, faster, stronger. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 7263–7271. Citado na página 27.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. Citado na página 28.

REN, S. et al. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, v. 28, 2015. Citado 2 vezes nas páginas 5 e 21.

RODRIGUES, D. A. Deep learning e redes neurais convolucionais: reconhecimento automático de caracteres em placas de licenciamento automotivo. Universidade Federal da Paraíba, 2018. Citado na página 13.

RUDER, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016. Citado na página 17.

Sérgio Eduardo Palmiere. *Arquiteturas e Topologias de Redes Neurais Artificiais*. 2016. Disponível em: <https://embarcados.com.br/redes-neurais-artificiais/>, Acesso em: 19 de abril 2022. Citado na página 15.

TACCHINO, F. et al. An artificial neuron implemented on an actual quantum processor. *npj Quantum Information*, Nature Publishing Group, v. 5, n. 1, p. 1–8, 2019. Citado na página 11.

TOSSATO, C. R. A função do olho humano na óptica do final do século xvi. *Scientiae Studia*, SciELO Brasil, v. 3, p. 415–441, 2005. Citado na página 9.

Tsung-Yi Lin. *COCO.Common Objects in Context*. 2022. Disponível em: <https://cocodataset.org>. Acesso em: 5 de abril 2022. Citado na página 32.

Von Wangenheim. *Avaliando, Validando e Testando o seu Modelo: Metodologias de Avaliação de Performance*. 2022. Disponível em: <https://lapix.ufsc.br/ensino/reconhecimento-de-padroes/avaliando-validando-e-testando-o-seu-modelo-metodologias-de-avaliacao-de-performance/>. Acesso em: 14 de outubro 2022. Citado na página 34.

WANG, C.-Y. et al. Cspnet: A new backbone that can enhance learning capability of cnn. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. [S.l.: s.n.], 2020. p. 390–391. Citado na página 29.

WANG, S.-C. Artificial neural network. In: *Interdisciplinary computing in java programming*. [S.l.]: Springer, 2003. p. 81–100. Citado na página 11.

WU, T.-H.; WANG, T.-W.; LIU, Y.-Q. Real-time vehicle and distance detection based on improved yolo v5 network. In: IEEE. *2021 3rd World Symposium on Artificial Intelligence (WSAI)*. [S.l.], 2021. p. 24–28. Citado na página 23.

YAKIMOVSKY, Y. Boundary and object detection in real world images. *Journal of the ACM (JACM)*, ACM New York, NY, USA, v. 23, n. 4, p. 599–618, 1976. Citado na página 11.

ZHAO, Z.-Q. et al. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, IEEE, v. 30, n. 11, p. 3212–3232, 2019. Citado 4 vezes nas páginas 15, 19, 25 e 27.