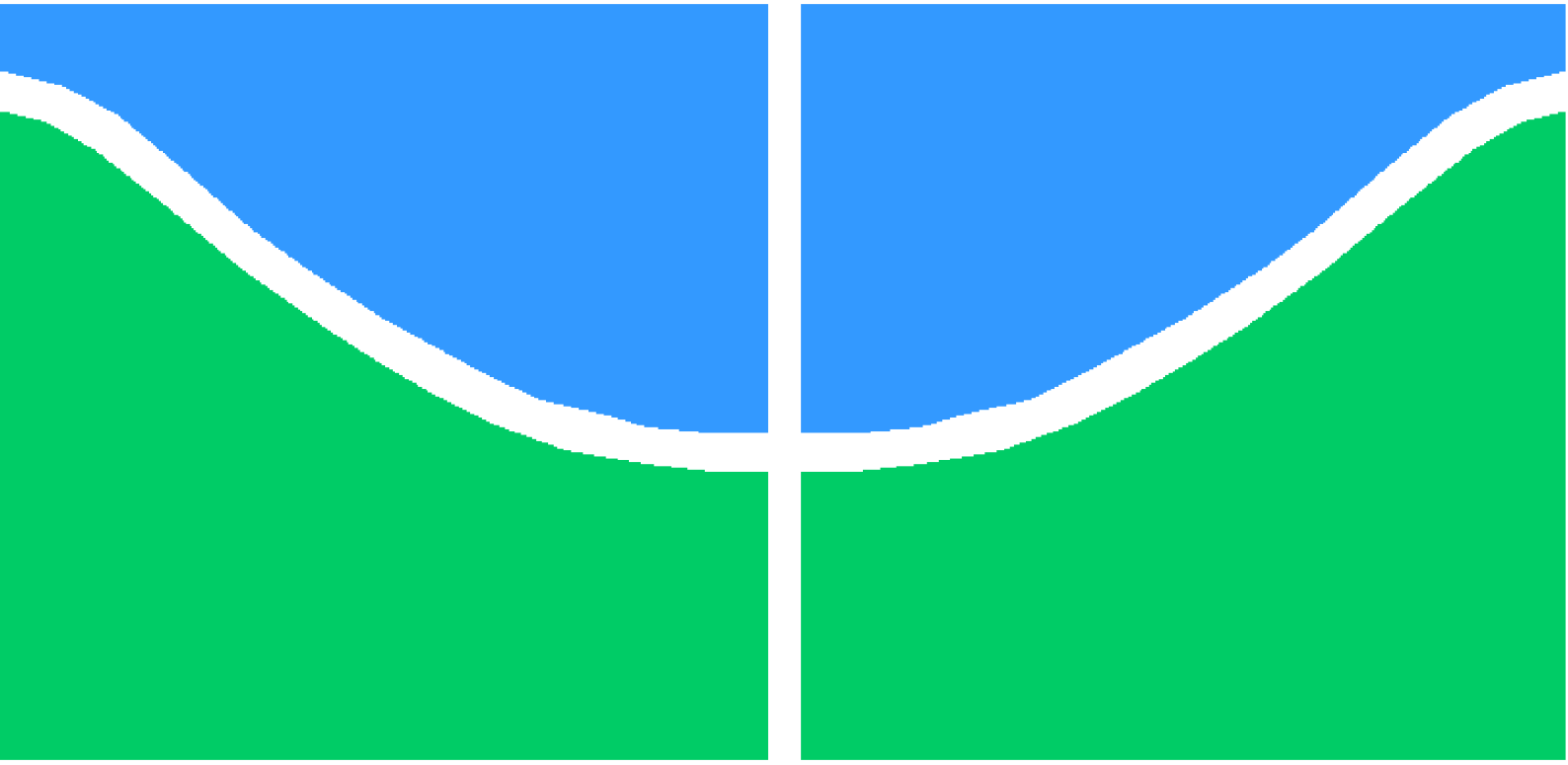

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia Aeroespacial

Detecção de heliportos utilizando redes neurais convolucionais

Autor: João Victor Rodrigues dos Santos
Orientador: Prof. Dr. Giancarlo Santilli

Brasília, DF
2022

João Victor Rodrigues dos Santos

Detecção de heliportos utilizando redes neurais convolucionais

Monografia submetida ao curso de graduação em Engenharia Aeroespacialda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia Aeroespacial.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Giancarlo Santilli

Brasília, DF

2022

João Victor Rodrigues dos Santos
Detecção de heliportos utilizando redes neurais convolucionais/ João Victor Rodrigues dos Santos. – Brasília, DF, 2022-
130 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Giancarlo Santilli

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2022.

1. Redes neurais convolucionais. 2. Reconhecimento de imagens. I. Prof. Dr. Giancarlo Santilli . II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Detecção de heliportos utilizando redes neurais convolucionais

CDU 02:141:005.6

João Victor Rodrigues dos Santos

Detecção de heliportos utilizando redes neurais convolucionais

Monografia submetida ao curso de graduação em Engenharia Aeroespacialda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia Aeroespacial.

Trabalho aprovado. Brasília, DF, 10 de maio de 2022:

Prof. Dr. Giancarlo Santilli
Orientador

Prof. Dr. Paolo Gessini
Convidado 1

Prof. Dr. Domenico Simone
Convidado 2

Brasília, DF
2022

Agradecimentos

Agradeço, primeiramente, a minha família especialmente a minha mãe por todo o apoio e incentivo que me foi dado durante toda a minha vida.

Aos meus amigos, por todo o apoio e parceria durante os anos de faculdade.

Aos meus pets, por serem os melhores do mundo.

Ao meu orientador, Giancarlo Santilli e aos professores Domenico Simone e Paolo Gessini pelo apoio na construção do trabalho.

A todas as pessoas que passaram pela minha vida e foram importantes na minha caminhada.

Resumo

Redes neurais convolucionais são o estado da arte em trabalhos de reconhecimento de imagens e visão computacional. A grande evolução dos algoritmos de inteligência artificial aliados com a evolução do poder computacional e da grande quantidade de dados permitiram que pesquisadores de diversas faculdades do mundo, ano após ano, construíssem modelos de aprendizado profundo mais precisos. As redes convolucionais são utilizadas em problemas de reconhecimento de imagens em diversos segmentos, desde saúde até aplicações na indústria aeroespacial. Na indústria aeroespacial os segmentos envolvendo drones e mobilidade aérea urbana estão entre os mais promissores para a próxima década. Um dos grandes desafios para a indústria aeroespacial e automotiva para os próximos anos são os veículos autônomos, que podem revolucionar a mobilidade da sociedade no século XXI. O objetivo desse trabalho consiste em aplicar redes neurais convolucionais especializadas no reconhecimento de objetos, buscando construir um algoritmo capaz de reconhecer heliportos, os quais podem ser aproximados como pistas de pouso que podem ser utilizados por veículos autônomos. O referencial do trabalho é composto por artigos históricos que são a base das redes neurais utilizadas em larga escala atualmente. O funcionamento das redes neurais também é exposto no trabalho, desde os modelos mais simples da década de 1950 até os modelos que são considerados estado da arte atualmente. A metodologia do trabalho também é exposta e consiste do fluxo de trabalho, que é utilizado em larga na escala na academia e no mercado em projetos de Deep Learning. A parte final do trabalho apresenta os detalhes da modelagem, além da apresentação e avaliação dos resultados obtidos durante o trabalho.

Palavras-chaves: Redes neurais convolucionais. Inteligência artificial. Detecção de objetos. Aeroespacial. *Deep Learning*.

Abstract

Convolutional Neural Networks are the state of the art in image recognition and Computer Vision work. The great evolution of Artificial Intelligence algorithms combined with the evolution of computational power and the large amount of data allowed researchers from di□erent universities around the world to build more accurate models of Deep Learning year after year. Convolutional Networks are used in problems of image recognition in several segments, from health to applications in the aerospace industry. In the aerospace industry, the segments involving drones and urban air mobility are among the most promising for the next decade. One of the great challenges for the aerospace and automotive industry for the coming years is the autonomous vehicles, that can revolutionize the mobility of society in the 21st century. The objective of this work is to apply Convolutional Neural Networks specialized in object recognition, seeking to build an algorithm capable of recognizing heliports, which can be approached as landing strips that can be used by autonomous vehicles. The theoretical framework is composed of historical articles that are the basis of neural networks used on a large scale today. The learning process of Neural Networks is also exposed in the work, from the simplest models of the 1950s to models that are considered state of the art today. The methodology of the work is also exposed and consists of the workflow that is widely used in the academy and at the job market in Deep Learning projects. The final part of the work presents the details of the modeling in addition to the presentation and evaluation of the results obtained during the project.

Key-words: Convolutional Neural Networks. Artificial Intelligence. Object Detection . Aerospace. Deep Learning.

Lista de ilustrações

Lista de abreviaturas e siglas

MNIST	Modified National Institute of Standards and Technology database
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
ReLU	Rectified Linear Unit
CNN	Convolutional Neural Network
RGB	Red Green Blue
SVM	Support vector machine
YOLO	You Only Look Once
MSE	Mean Squared Error
RMSE	Root Mean Square Error
MAE	Mean Absolute Error
GPU	Graphics Processing Unit
CPU	Central Process Unit

Lista de símbolos

x	Vetor de entrada
w	Pesos tensoriais
h	Resultado do produto entre inputs e pesos
λ	Função de ativação
b	Bias
$\hat{y}$	Output da rede neural
n	Número de camadas ocultas
N	Regiões de interesse
σ	Sigmoide
σ	Derivada sigmoide
$tanh$	Tangente hiperbólica
$tanh$	Derivada tangente hiperbólica
S	Softmax
S	Derivada softmax
R	ReLU
R	Derivada ReLU
Lr	Leaky ReLu
Lr	Derivada Leaky ReLu
Pr	PReLu
Pr	Derivada Leaky ReLu
y	label
	Cross Entropy
E	Função de custo

E	Gradiente da função de custo
w	Peso atualizado
b	Bias atualizado
β	Learning rate

Sumário

1 Introdução

1.1 Contextualização

Com a evolução do poder computacional e dos algoritmos de *Machine Learning* e *Deep Learning*, diversas empresas e pesquisadores começaram a utilizar a versatilidade dos algoritmos para resolver problemas nos mais diversos segmentos, (KHANDANI; KIM; LO, 2010) e (KRUPPA et al., 2013) utilizaram técnicas de *Machine Learning* para modelagem de risco de crédito.

No âmbito empresarial, (VAFEIADIS et al., 2015) e (JAIN; KHUNTETA; SRIVASTAVA, 2020) utilizaram *Machine Learning* para prever a ocorrência de *churn* entre clientes que assinam um determinado serviço. Modelagem de aprendizado de máquina aliada com técnicas de mecânica de dano contínuo foram utilizadas por (ZHAN; LI, 2021) buscando prever o tempo de vida restante de aços utilizados na indústria aeroespacial.

O processamento e reconhecimento de imagens ganharam bastante destaque nos últimos anos devido aos diferentes trabalhos utilizando técnicas de aprendizado profundo que, ano após ano, tornam-se mais indispensáveis devido ao grande nível de precisão que tais técnicas conseguem alcançar, (TIAN; FU, 2020) e (LE et al., 2020) realizaram um estudo de revisão sobre as diferentes arquiteturas de redes neurais artificiais utilizadas em problemas médicos.

Os algoritmos de aprendizado profundo, assim como os algoritmos tradicionais de aprendizado de máquina, podem ser utilizados para a solução de diversos problemas em diversos segmentos diferentes. (ZHANG et al., 2021) realizou um trabalho de detecção de diferentes microrganismos (bactérias, fungos, e parasitas) em imagens microscópicas.

Modelos de *Deep Learning* também são utilizados em processamento de imagens conforme atestado por (KARANAM; SRINIVAS; KRISHNA, 2020) e, são utilizados no reconhecimento de diferentes desordens no corpo humano, como nos trabalhos de (YIN; LI; WU, 2020) e (DANESHJOU et al., 2021), também podem utilizados na indústria automotiva, conforme o atestado por (BöROLD et al., 2020).

Com a evolução do *Deep Learning*, um novo campo de estudo chamado visão computacional ascendeu. A visão computacional consiste no campo de estudo no qual computadores possam modelar e replicar a visão dos seres humanos utilizando softwares e hardware. Em seu trabalho, (FENG et al., 2019) realizou um estudo sobre a implementação dos diferentes algoritmos de visão computacional.

Aplicações de visão computacional na academia foram realizadas por (KUMAR

et al., 2018), que abordou o design, construção e controle de um veicule aquático autônomo, (GUPTA et al., 2021) realizou um estudo sobre as implementações atuais de visão computacional em carros autônomos.

Na indústria aeroespacial, (AL-KAFF et al., 2018) realizou uma revisão sobre os diferentes algoritmos e implementações de modelos de visão computacional utilizados em veículos aéreos não tripulados na última década. (LI et al., 2020) realizou um grande trabalho construindo e implementando um algoritmo de visão computacional, utilizado para o controle e navegação de drones empregados em corridas que requerem um grande nível de precisão em suas manobras.

1.2 Justificativa

Existem dois segmentos dentro do mercado aeroespacial que estão em amplo crescimento: os mercados de delivery via drones e de mobilidade aérea urbana. (KELLERMANN; BIEHLE; FISCHER, 2020) realizou uma revisão de literatura sobre a utilização de drones focados em serviços de mobilidade aérea urbana e delivery.

Segundo o jornal GLOBE NEWSWIRE, previsões indicam que o mercado de entrega via drones pode atingir o valor de 7 bilhões de dólares em 2027 (GLOBE..., a) e o mercado de o mercado de mobilidade aérea urbana pode atingir 8 bilhões de dólares até 2030 (GLOBE..., b).

Mesmo com as projeções de alto crescimento para esses mercados, existe um grande desafio de engenharia que é encontrado pelos engenheiros que trabalham nesse mercado: o consumo energético dos drones. Diversos pesquisadores realizam estudos com o objetivo de solucionar esse problema. (DUKKANCI; KARA; BEKTAŞ, 2021), (FOTOUHI; DING; HASSAN, 2021), (COKYASAR et al., 2021) e (CHENG; ADULYASAK; ROUSSEAU, 2020) realizaram trabalhos buscando otimizar o consumo de energia em drones utilizando métodos alternativos.

É importante notar que os quatro trabalhos citados no parágrafo anterior preocuparam em otimizar consumo energético em drones, porém utilizando conceitos de otimização de rotas, otimização de velocidade e estruturas especializadas na troca de células energéticas. Entretanto, os estudos não eram focados no desenvolvimento de células energéticas de maior capacidade.

O desafio proposto por esse trabalho é construir um modelo de Deep Learning que seja capaz de reconhecer objetos que serão aproximados como pistas de pouso, nas quais os veículos autônomos possam realizar um pouso em segurança, onde a empresa responsável possa realizar um trabalho de carga, troca de bateria ou pouso do veículo autônomo. (COKYASAR, 2021) realizou um trabalho abordando o tema de troca de

bateria utilizando programação não linear mista para encontrar o melhor ponto onde os veículos possam passar por um procedimento de troca da célula de energia.

1.3 Objetivo Geral

O objetivo geral desse trabalho é construir um modelo utilizando redes neurais convolucionais para a detecção de heliportos em imagens, que estão em diferentes localizações e ambientes. A intenção desse trabalho também consiste em fomentar trabalhos subsequentes em busca da aplicação prática em tempo real desse algoritmo.

1.3.1 Objetivos Específicos

- Encontrar um dataset composto por imagens de heliportos;
- Treinar um algoritmo baseado em redes neurais convolucionais;
- Medir o desempenho do algoritmo com diferentes métricas relacionadas à precisão e qualidade dos modelos;
- Verificar se as redes neurais convolucionais conseguem obter bons resultados na classificação de heliportos.

1.4 Organização Documento

O documento é dividido em cinco capítulos. No primeiro capítulo estão descritos a contextualização, os objetivos e justificativa para execução do trabalho.

O segundo capítulo abrange todo o referencial teórico, iniciando na contextualização histórica sobre Deep Learning. O referencial teórico abrange desde as primeiras técnicas utilizadas em redes neurais até os últimos avanços encontrados na academia sobre o tema. Nele também estão inclusos os conceitos referentes ao treinamento de redes neurais, como backpropagation, gradiente descendente e funções de perda.

O terceiro capítulo é referente à metodologia utilizada para a solução do problema com fluxogramas indicando o caminho seguido pelo autor.

O quarto capítulo é referente aos resultados obtidos durante o treinamento de um modelo baseado em redes neurais convolucionais.

O quinto capítulo consiste em uma conclusão sobre o trabalho desenvolvido.

2 Referencial Teórico

2.1 Contextualização histórica

Conforme citado na seção 1.1, as redes neurais artificiais estão permitindo grandes avanços em diversos campos de trabalho atualmente. O grande poder de processamento e generalização é o responsável pela grande aplicabilidade das redes neurais e foi desenvolvido ao longo de muitos anos de pesquisas na área acadêmica.

O conceito de rede neural é derivado da analogia de uma estrutura artificial que funcione como o cérebro humano, representando matematicamente a estrutura do neurônio biológico que é composto pelos dendritos, axônios e pelo corpo celular. O primeiro modelo matemático de um neurônio foi proposto por Warren MuCulloch e Walter Pitts em 1943, sendo denominado de *Threshold Logic Unit* (MCCULLOCH; PITTS, 1943).

No artigo de 1958, Frank Rosenblatt propôs a modelagem do neurônio matemático proposto por (MCCULLOCH; PITTS, 1943) em uma rede neural artificial camada única chamada de *perceptron* (ROSENBLATT, 1958). O *perceptron* de Rosenblatt consiste no modelo mais simples de uma rede neural artificial *feedforward* com um classificador. O funcionamento do *perceptron* será abordado com mais profundidade na seção 2.2.

No livro de 1962, Rosenblatt foi o pioneiro em apresentar o conceito de um *perceptron* composto por multicamadas (ROSENBLATT, 1962). O *perceptron* multicamadas que é conhecido como *Multilayer Perceptron* é a base das redes neurais profundas que estão revolucionando as aplicações de *Deep Learning* na última década.

Na década de 1960, Alexey Ivakhnenko produziu vários artigos tratando do *Multilayer Perceptron*. Entre seus trabalhos, um dos mais importantes foi o artigo de 1968 (IVAKHNENKO, 1968), no qual Ivakhnenko formulou um algoritmo de aprendizado para o Multilayer Perceptron, sendo um dos pioneiros em criar uma rede neural funcional.

Em 1970, Seppo Linnainmaa foi o pioneiro em introduzir o conceito de *backpropagation* em sua tese de mestrado (SEPPO, 1970). Linnainmaa em 1976 publicou um artigo sobre o tema, aprofundando os estudos de seu mestrado (LINNAINMAA, 1976). O *backpropagation* é o responsável pela otimização dos pesos da rede neural buscando otimizar a função de perda.

Seppo Linnainmaa foi o pioneiro em apresentar o conceito de backpropagation, porém o grande responsável por introduzir esse conceito nas técnicas de redes neurais foi David Rumelhart no seu artigo de 1986 (RUMELHART; HINTON; WILLIAMS, 1986).

Um dos pioneiros nos estudos das redes neurais convolucionais que são o foco

desse trabalho é o cientista Kunihiko Fukushima, e seu trabalho de 1980 foi um dos percursores para as redes neurais convolucionais que são amplamente utilizadas atualmente (FUKUSHIMA, 1980).

Yann LeCun, um dos alunos de Fukushima, também foi um grande pesquisador no tema de redes neurais convolucionais (LECUN et al., 1990) e (CUN et al., 1990) são exemplos desses trabalhos com o objetivo de reconhecimento de padrões. Outro trabalho extremamente importante de de Yann LeCun foi realizado em 1998, utilizando o conjunto de dados MNIST. O MNIST consiste em um grande banco de imagens compostas por números em escala de cinza escritos a mão que atualmente são utilizados por iniciantes no tema de redes neurais (LECUN et al., 1998a).

Alguns trabalhos na última década focados em redes neurais convolucionais serão apresentados em sequência. Em 2012, Alex Krizhevsky colaborando com outros pesquisadores desenvolveram uma rede neural conhecida como AlexNet. O grande feito da rede neural foi participar de uma competição conhecida como ILSVRC (ImageNet Large Scale Visual Recognition Challenge) sediada pelo ImageNet, que dispunha de um banco com milhões de imagens. A rede AlexNet conseguiu resultados espetaculares na competição (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

Com o sucesso da AlexNet, outras redes neurais foram desenvolvidas com o objetivo de vencer a competição, e em 2013 a rede conhecida como ZFNet venceu a competição ILSVRC (ZEILER; FERGUS, 2014). Em 2014 a rede conhecida como GoogLeNet foi a vencedora da competição (SZEGEDY et al., 2015), outra rede de grande destaque em 2014 foi a VGG NET que também obteve resultados espetaculares na competição (SIMONYAN; ZISSERMAN, 2015).

A Microsoft ResNet é uma rede neural que pode ser considerada o estado da arte, foi desenvolvida pela Microsoft e possui versões com diferentes quantidades de camadas convolucionais (HE et al., 2016a).

2.2 *Perceptron*

Conforme citado na seção 2.1, o *perceptron* consiste no modelo simplificado de uma rede neural para classificação de padrões linearmente separáveis. O *perceptron* consiste de um único neurônio com pesos tensoriais que podem ser ajustados e um *bias*, sendo uma estrutura análoga ao cérebro humano (HAYKIN, 2003). A Figura 1 representa a estrutura de um *perceptron*, enquanto a Figura 2, além de representar a estrutura, também apresenta uma analogia com os componentes do cérebro humano. No *perceptron* ocorrem quatro operações principais:

- Os sinais de entrada no *perceptron* são representados pelo vetor $x = [x_1, x_2, \ldots, x_n]$,

esses sinais podem representar diferentes tipos de informações como características de uma pessoa ou os pixels que compõem uma imagem (HAYKIN, 2003);

- Um conjunto de pesos tensoriais pelo qual o vetor de entrada é multiplicado, os pesos são representados pelo vetor $w = [w_1, w_2, \ldots, w_n]$. Os pesos possuem analogia com as sinapses no cérebro humano, sendo responsáveis por transmitir o vetor de entrada (HAYKIN, 2003);

- Um neurônio composto por um somatório responsável pela operação $h = \sum_{i=1}^{n} x_i w_i + b$. O termo b corresponde ao bias (viés), que é um valor constante que fornece um grau de liberdade a mais para a equação (HAYKIN, 2003);

- Na última etapa, o valor resultante do somatório passa por uma função de ativação que gera a saída $\hat{y} = \lambda(h)$. Existem várias funções de ativação que serão descritas na seção 2.7, alguns exemplos são: sigmoide, softmax, tangente hiperbólica, rectified linear unit(ReLU).

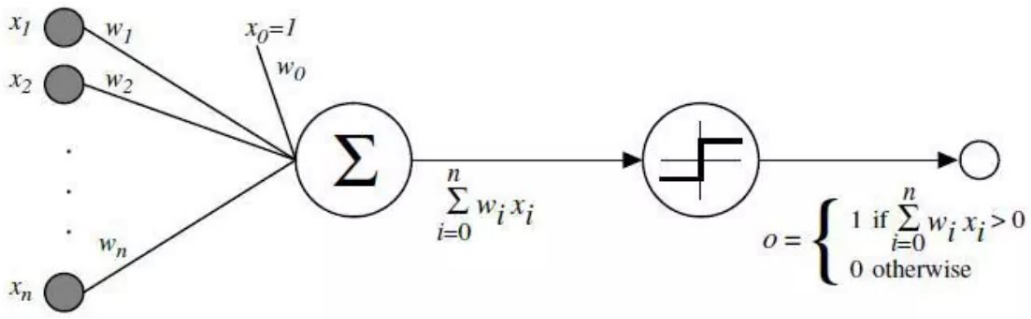

Figura 1 – Estrutura de um *Perceptron*.

Fonte: (KURENKOV, 2020)

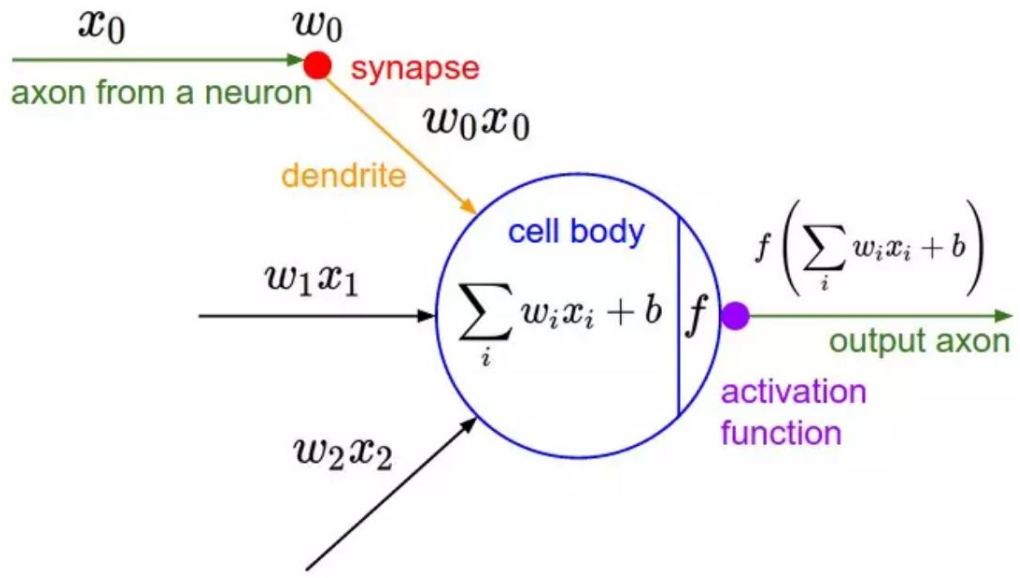

Figura 2 – Estrutura de um *Perceptron*.

Fonte: (KURENKOV, 2020)

2.3 *Multilayer Perceptron*

A rede de *perceptron* multicamadas consiste de um conjunto de neurônios de entrada, uma quantidade *n* de camadas ocultas compostas por um número *n* de neurônios computacionais e uma camada de saída. O *perceptron* multicamadas consiste de uma generalização do *perceptron*, apresentado na seção 2.2 e seu aprendizado/otimização ocorre pelo algoritmo de *backpropagation* (HAYKIN, 2003). O conceito de *backpropagation* será exposto na seção 2.8.3.

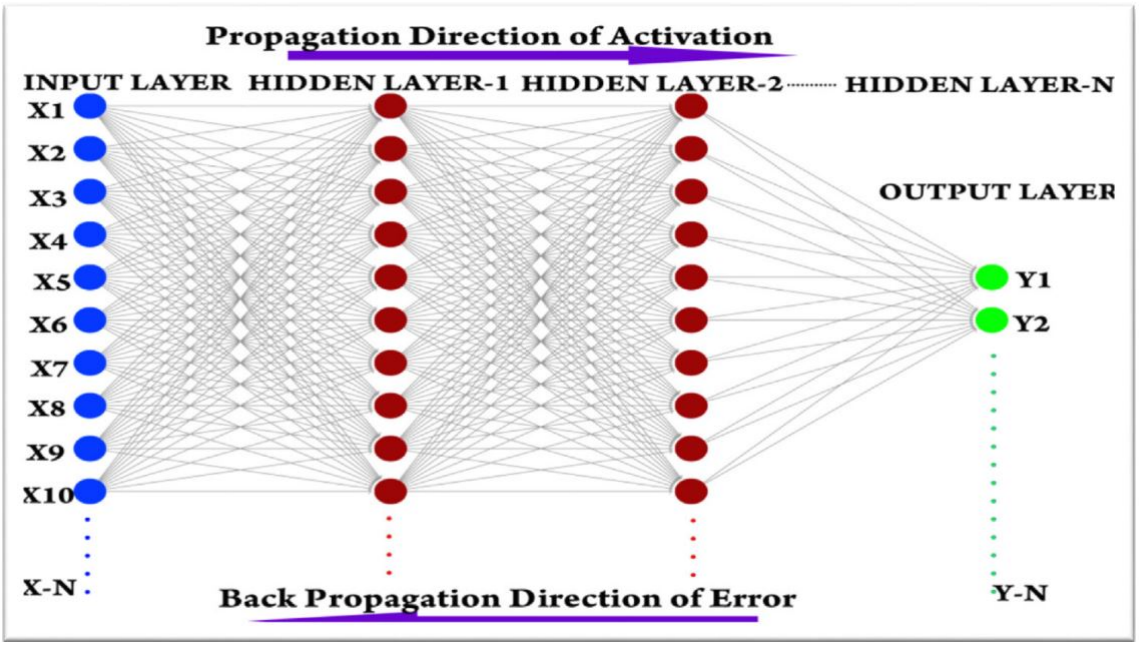

Figura 3 – Estrutura de um *Multilayer Perceptron*.

Fonte: (AVUÇLU; BAŞÇIFTÇI, 2018)

A Figura 3 representa a estrutura de um perceptron multicamadas, é semelhante à estrutura apresentada na seção 2.2 sendo composta por:

- **Camada de entrada**, na figura 3 é representada pela primeira camada, onde o vetor de entrada $x = [x_1, x_2, \ldots, x_n]$ define os neurônios (HAYKIN, 2003);

- **Camadas ocultas**, o sinal de entrada é propagado através das n camadas ocultas, com os pesos fixos. Através de uma função de ativação cada neurônio gera uma resposta que é utilizada para alimentar o próximo neurônio. Normalmente redes neurais artificiais com um número grande de camadas ocultas, são denominadas redes neurais profundas (HAYKIN, 2003);

- **Camada de saída**, na última camada um valor de saída e um erro são gerados. Através do gradiente descendente, os pesos são atualizados iterativamente, partindo da camada de saída até a camada de entrada (*backpropagation*), em busca da otimização dos pesos w_{ij} que minimizem a função de erro (*Loss Function*) (HAYKIN, 2003).

2.4 Redes Neurais Convolucionais

As redes neurais convolucionais (CNN) são um tipo de algoritmo utilizado em *Deep Learning*, que possuem grande aplicabilidade no campo de reconhecimento de imagens e visão computacional. A primeira aplicação bem sucedida de uma rede neural convolucional foi desenvolvida em 1998 por Yann Lecun (LECUN et al., 1998b).

A Figura 4 representa uma estrutura de rede neural convolucional. Normalmente as redes neurais convolucionais são compostas por:

- ***Input***: A primeira etapa em uma CNN são os dados de entrada. Em problemas que abordam imagens, a entrada é composta de um tensor que representa a largura, altura e profundidade. Normalmente, em problemas de reconhecimento de imagens, o tensor de entrada é tridimensional onde a profundidade é determinada pelos canais de cores presentes (RGB) (GOODFELLOW; BENGIO; COURVILLE, 2016);

- ***Convolutional layer***: A função de uma camada convolucional consiste em extrair os diferentes recursos de uma imagem e montar um *feature map*. Essa operação ocorre através de filtros que varrem o tensor de entrada com uma operação *element wise*, gerando o *output* que é conhecido como *feature map*. Os pesos que são introduzidos na seção 2.2 são representados pelos valores multiplicativos que compõem os filtros (KANG; SONG; SUN, 2019);

- ***Activation Function***: As funções de ativação também estão presentes nas redes neurais convolucionais. Normalmente a função de ativação mais utilizada em redes neurais convolucionais é a ReLU. A ReLU é exposta na seção 2.6.4 e possui como vantagem permitir a esparsidade da rede, gerando um aumento de velocidade (YAMASHITA et al., 2018);

- ***Pooling Layer*** A camada de *pooling* possui como objetivo simplificar e reduzir a dimensionalidade do *output* gerado pela camada anterior. O objetivo dessa redução de dimensionalidade é reduzir o custo computacional do modelo, além de encontrar padrões mais evidentes evidenciados pela camada anterior (YAMASHITA et al., 2018);

- ***Fully Connected Layer*** A fully connected layer funciona como as redes neurais descritas na seção 2.3. O *output* das camadas anteriores deve passar por um processo conhecido como *flattening*, que possui como objetivo transformar o tensor resultante em um vetor unidimensional. A camada densa é responsável pela resposta final do modelo (YAMASHITA et al., 2018).

A Figura 5 representa a estrutura de uma rede neural convolucional detalhada. Nota-se que os processos de *forward propagation* da seção 2.8.2 e *backpropagation* da seção 2.8.3 estão presentes.

Redes neurais convolucionais podem apresentar desde algumas unidades de camadas até centenas de camadas convolucionais. No artigo de 2016 (HE et al., 2016b), a rede ResNets é avaliada utilizando desde dezenas de camadas até o limite de 152 em um trabalho de reconhecimento de imagens.

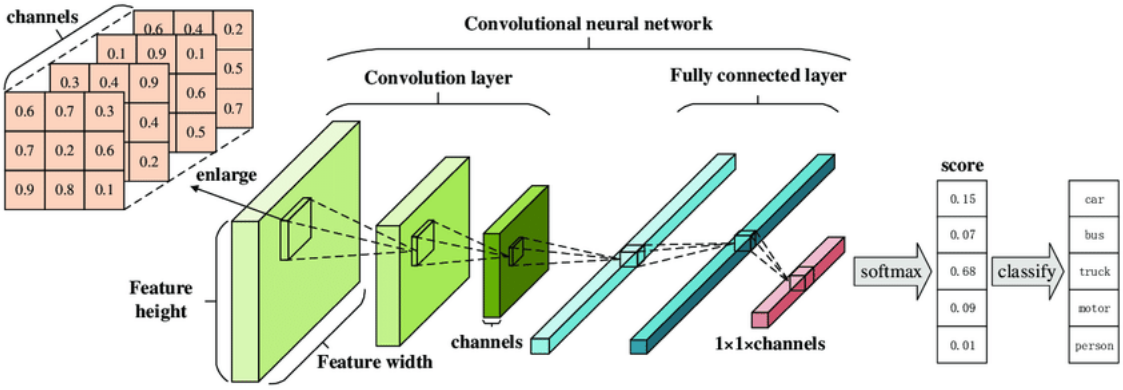

Figura 4 – Estrutura de uma rede neural convolucional.

Fonte: (KANG; SONG; SUN, 2019)

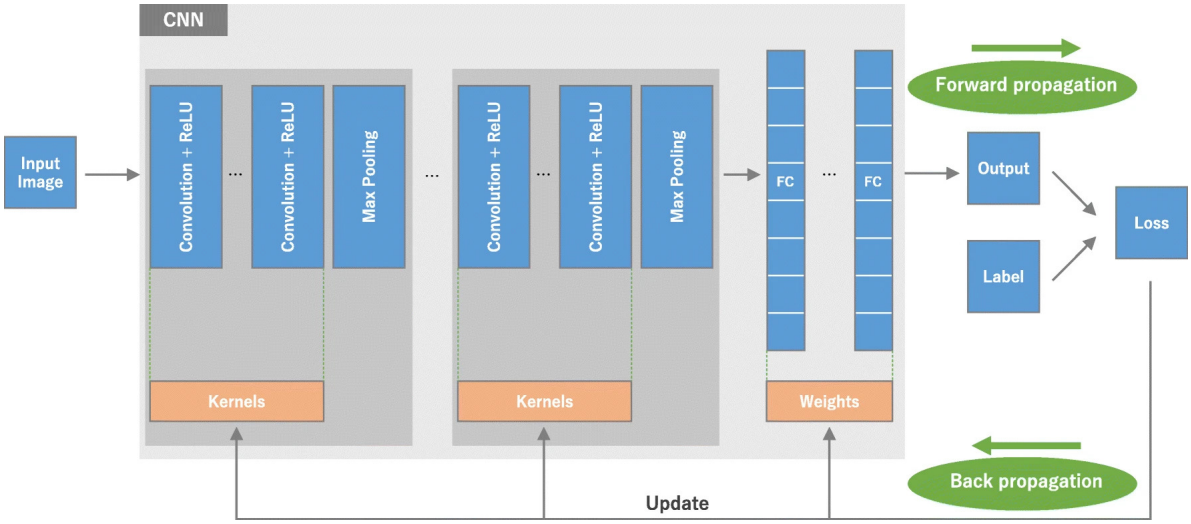

Figura 5 – Estrutura de uma rede neural convolucional detalhada.

Fonte: (YAMASHITA et al., 2018)

A Figura 6 representa o processo de convolução, e nota-se que existem vários pixels com magnitude zero nas bordas da imagem. Esse processo é conhecido como *padding* e possui como função adicionar pixels ao redor da imagem antes da operação de convolução,

com o objetivo de preservar a dimensionalidade da imagem após o processo de convolução, além de manter as informações presentes nas bordas da imagem (NGUYEN et al., 2019).

A Figura 7 representa o processo de *pooling*. O processo de *pooling* apresentado nessa figura em específico é denominado de *max pooling*, que consiste em capturar o maior valor em uma janela 2x2. O objetivo do processo de *pooling* é reduzir a dimensionalidade do *feature map*. O custo computacional é reduzido com a aplicação do *pooling* (SUN et al., 2017).

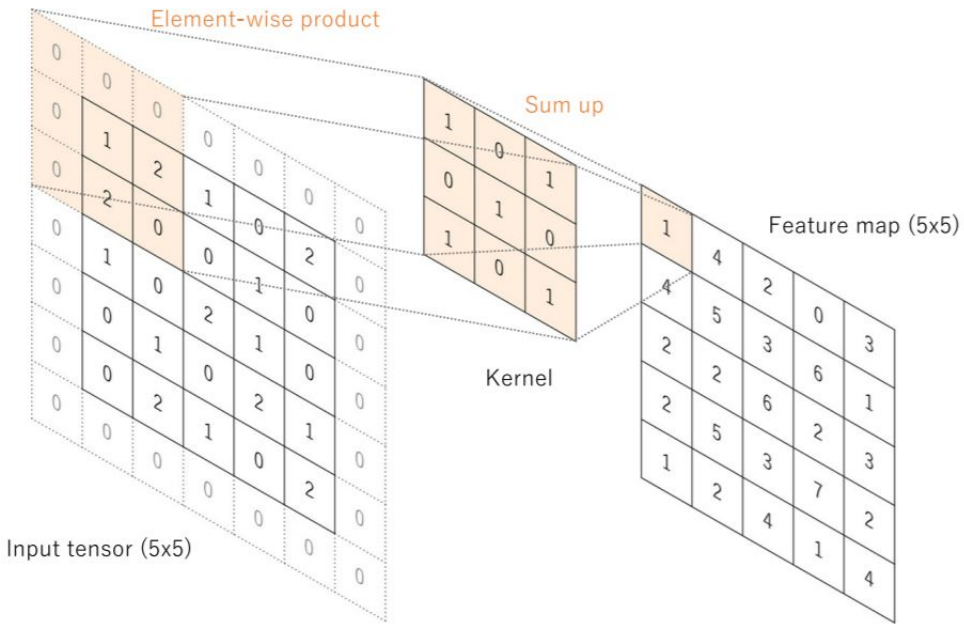

Figura 6 – Convolução.

Fonte: (YAMASHITA et al., 2018)

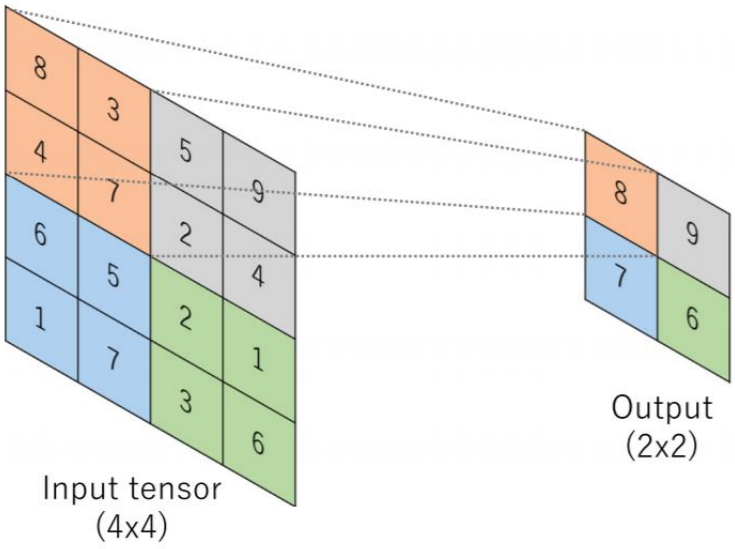

Figura 7 – *Pooling*.

Fonte: (YAMASHITA et al., 2018)

2.4.1 Reconhecimento e detecção de objetos

No campo de Deep Learning relacionado ao trabalho com imagens, existem duas abordagens principais. O reconhecimento de imagem aborda os trabalhos que possuem como objetivo identificar se um objeto existe ou não na imagem.

Considere por exemplo a Figura 8, considerando um problema tradicional de classificação de imagens, uma rede neural convolucional deveria informar se o animal presente na foto é um cachorro.

Na aplicação de detecção de objetos, a rede neural pode trabalhar com uma única classe ou com múltiplas classes, mas o grande diferencial é que o algoritmo deve ser capaz de identificar e localizar cada objeto na imagem.

Considerando a figura 9, definida por uma situação em que existem múltiplos objetos na imagem, a utilização de um algoritmo de detecção se torna favorável na identificação e localização dos objetos.

Figura 8 – Cachorro.

Fonte: Autoral

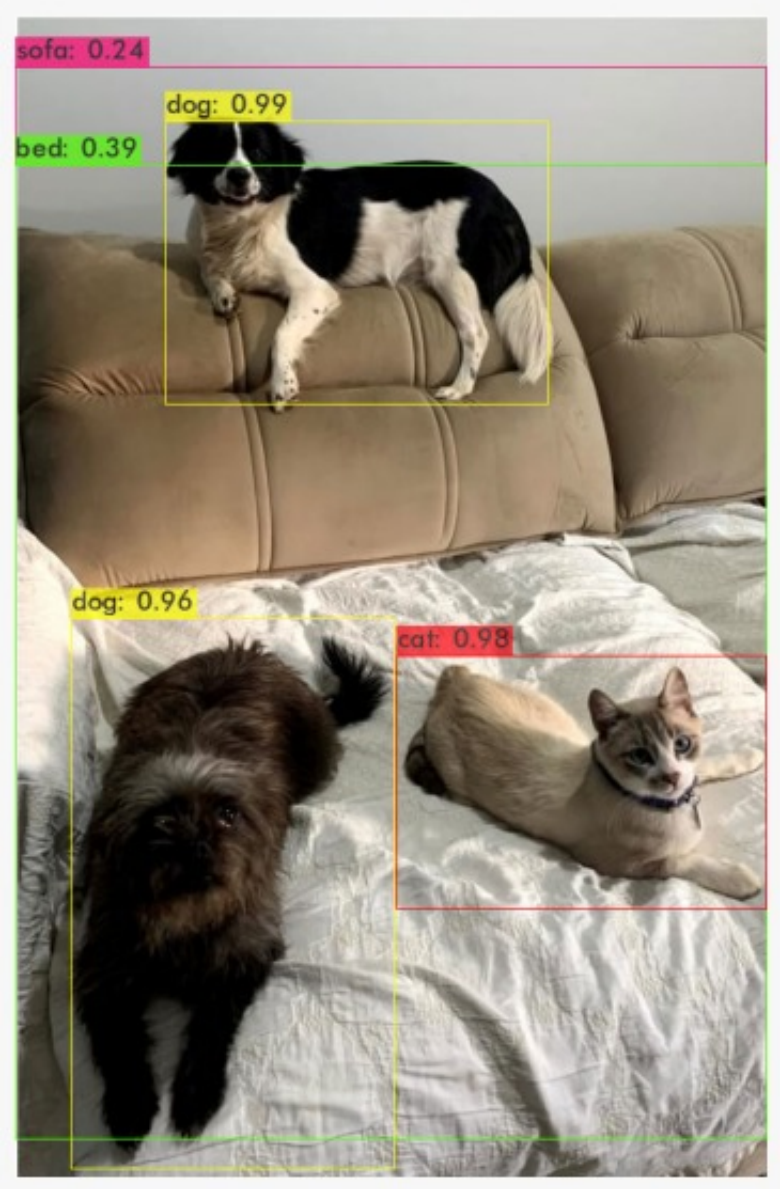

Figura 9 – Múltiplos animais.

Fonte: Autoral

2.5 *You Only Look Once*

O *You Only Look Once* (Yolo) é um algoritmo bastante utilizado para a detecção de objetos em imagens. O funcionamento do algoritmo é baseado nas redes neurais convolucionais e sua abordagem consiste em:

- dividir a imagem em uma grade de nXn células;
- as células presentes na grade geram as caixas delimitadoras, e determinam o índice de confiança de existir um objeto naquela caixa delimitadora;
- Em cada caixa delimitadora é feito o cálculo da probabilidade de um determinado objeto pertencer a uma determinada classe (REDMON et al., 2016).

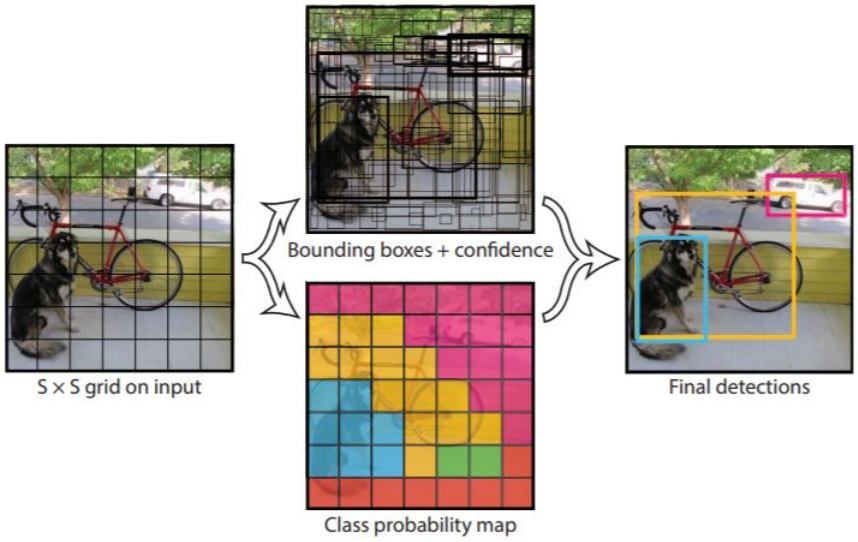

Figura 10 – Yolo.

Fonte: (REDMON et al., 2016)

A Figura 10 representa as etapas de um algoritmo Yolo.

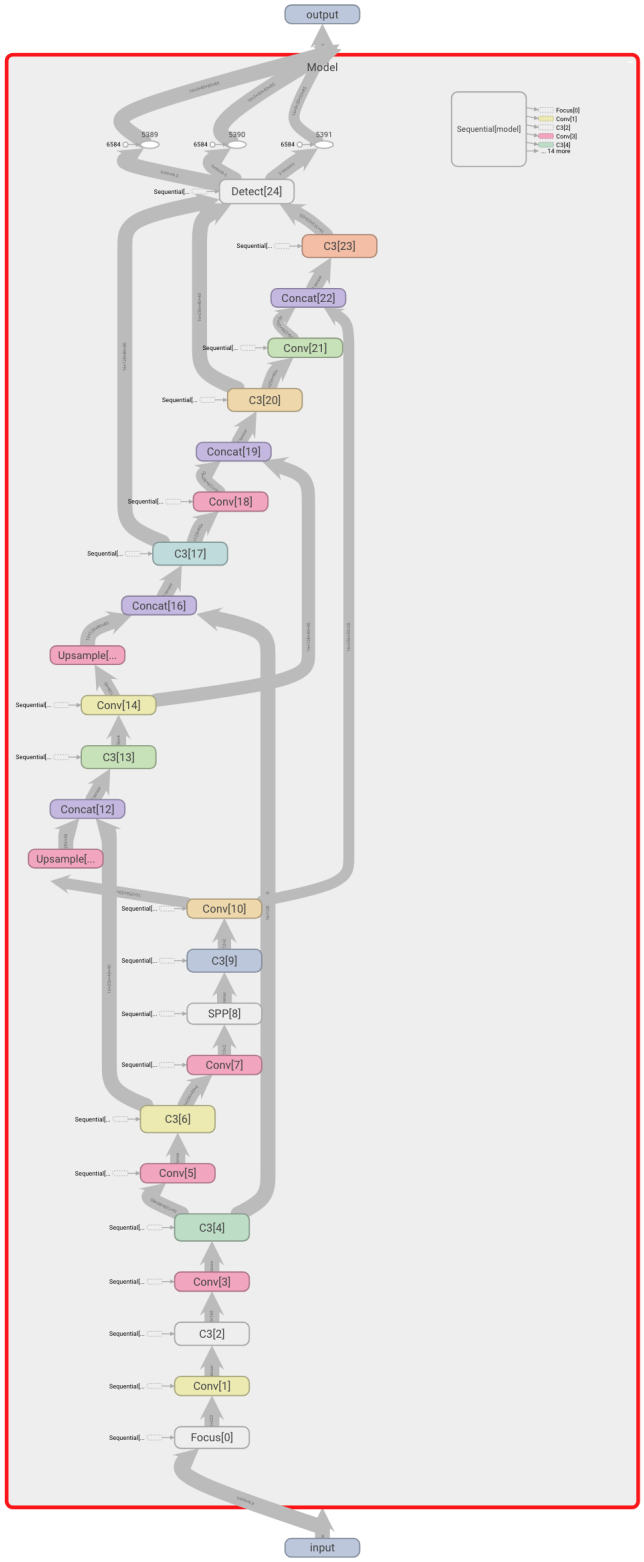

Figura 11 – Arquitetura Yolo

Fonte: (YOLOV5,)

A Figura 11 representa a arquitetura simplificada de um modelo Yolo. Na imagem é possível notar as camadas convolucionais e de *pooling*, além dos processos intermediários

que ocorrem na rede neural.

2.6 Funções de ativação

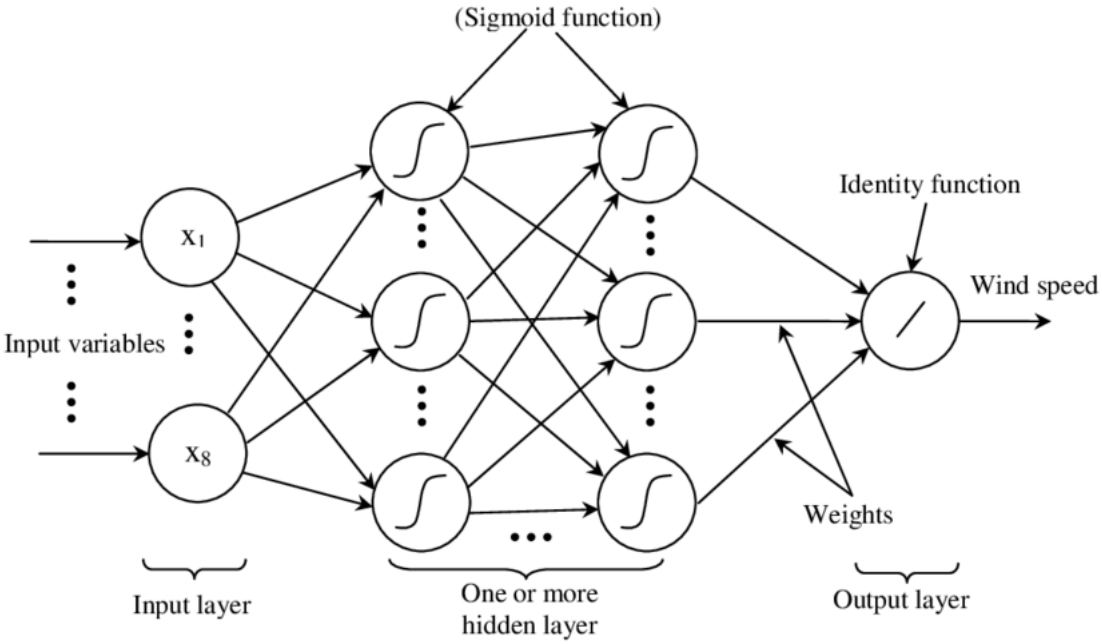

Figura 12 – Estrutura de um *Multilayer Perceptron*.

Fonte: (BOUZGOU, 2012)

A função de ativação ou função de transferência é um dos elementos mais importantes na estrutura de uma rede neural. As principais atribuições de uma função de ativação incluem introduzir um componente não linear na estrutura da rede, o qual permite uma maior eficiência no aprendizado de padrões complexos. Outra função extremante importante das funções de transferência é decidir se um neurônio deve ser ativado ou não, isto é, definir se a saída de um neurônio é relevante para ser propagada para as próximas camadas (GOODFELLOW; BENGIO; COURVILLE, 2016).

A Figura 12 representa uma rede neural que apresenta diversas funções sigmoides como ativação nas camadas ocultas. Normalmente, a saída de cada neurônio é definida pela equação 2.1:

$$\hat{y} = \lambda((w_{ij} \cdot x_i) + b) \tag{2.1}$$

O b na equação 2.1 é denominado bias, que é um valor constante utilizado para ajustar a saída do neurônio (GOODFELLOW; BENGIO; COURVILLE, 2016). Alguns

exemplos de função de ativação comumente utilizadas nos neurônios de redes neurais artificiais são:

2.6.1 Sigmoide

A função sigmoide é apresentada na figura 13 e representada pela equação 2.2, sua derivada é apresentada na figura 14 e representada pela equação 2.3.

$$\sigma(x) = \frac{1}{1 + e^{x}} \tag{2.2}$$

$$\sigma\,(x) = \sigma(x)(1 - \sigma(x)) \tag{2.3}$$

Inicialmente a função sigmoide foi uma das mais utilizadas em arquiteturas de redes neurais devido ao seu comportamento binário variando entre 0 e 1. A função sigmoide é especialmente útil em problemas de classificação binários. A desvantagem da utilização da função sigmoide é que a derivada tende a zero, gerando um efeito chamado de dissipação do gradiente (o produto obtido pela propagação do gradiente dissipa), que possui como consequência a dificuldade da rede continuar otimizando a sua função de perda (FENG; LU, 2019).

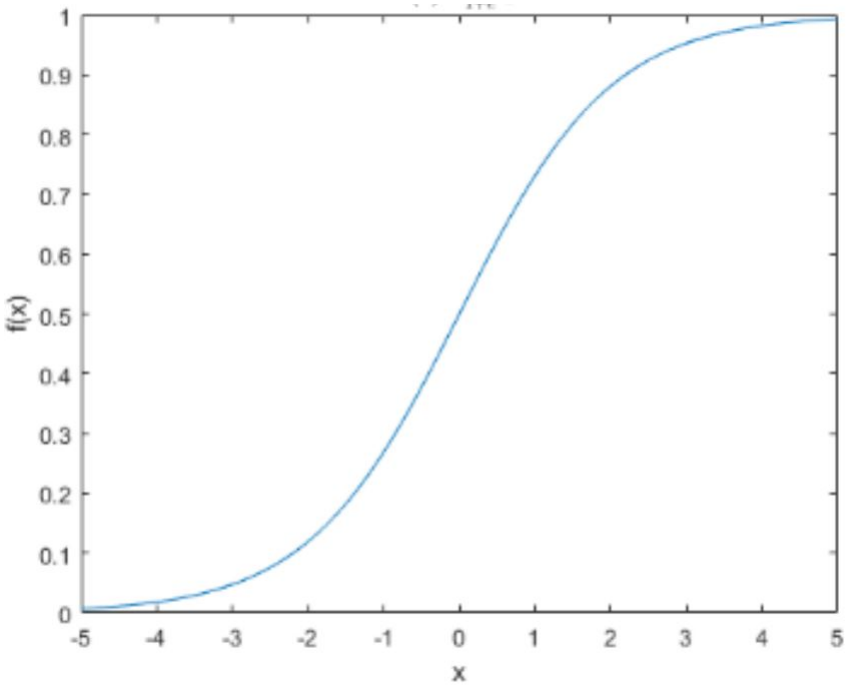

Figura 13 – Função Sigmoide.

Fonte: (FENG; LU, 2019)

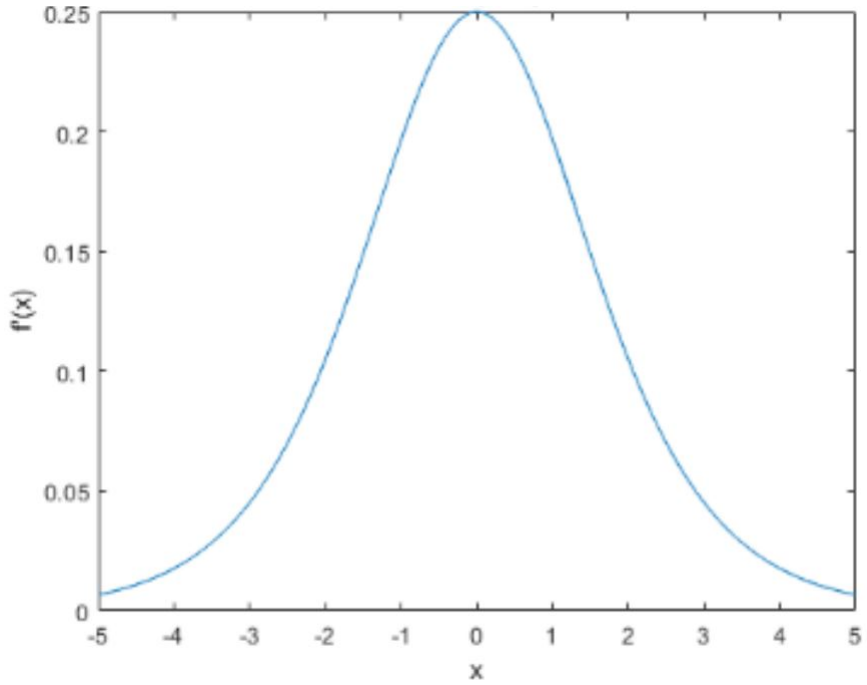

Figura 14 – Derivada da função Sigmoide.

Fonte: (FENG; LU, 2019)

2.6.2 Tangente hiperbólica

A função tanh é semelhante à função sigmoide devido ao seu formato e não linearidade. As principais diferenças são que a tangente hiperbólica é centrada no zero com variação entre -1 e 1 (FENG; LU, 2019).

A função tanh é apresentada na figura 15 e representada pela equação 2.4, sua derivada é apresentada na figura 16 e representada pela equação 2.5.

$$tanh(x_i) = \frac{sinhx_i}{coshx_i} = \frac{e^{x_i} - e^{-x_i}}{e^{x_i} + e^{-x_i}} = \frac{2}{1 + e^{-2x_i}} - 1 = 2\sigma(2x_i) - 1 \tag{2.4}$$

$$tanh\,(x_i) = \frac{4e^{-2x_i}}{(1 + e^{-2x_i})^2} \tag{2.5}$$

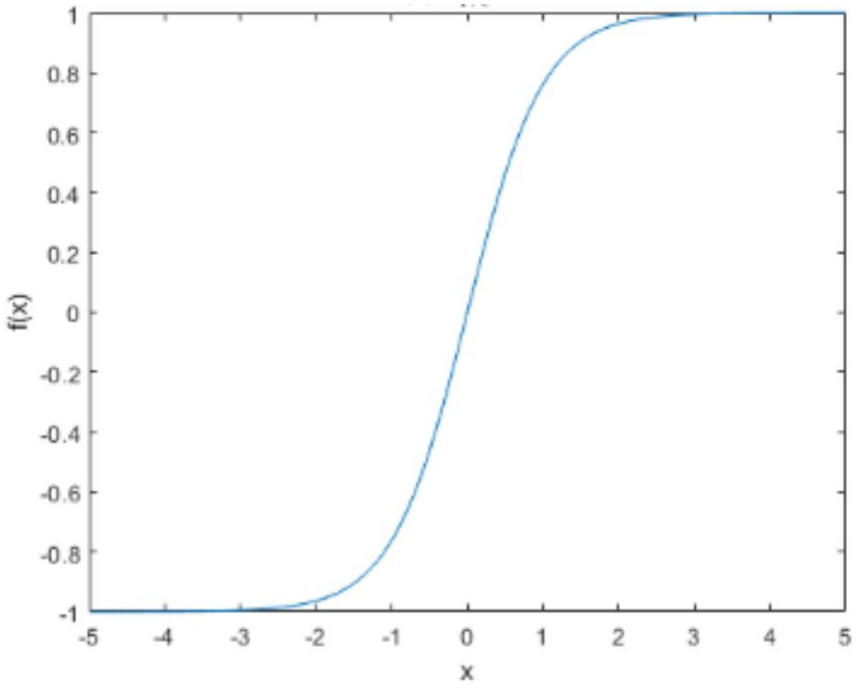

Figura 15 – Função tangente hiperbólica.

Fonte: (FENG; LU, 2019)

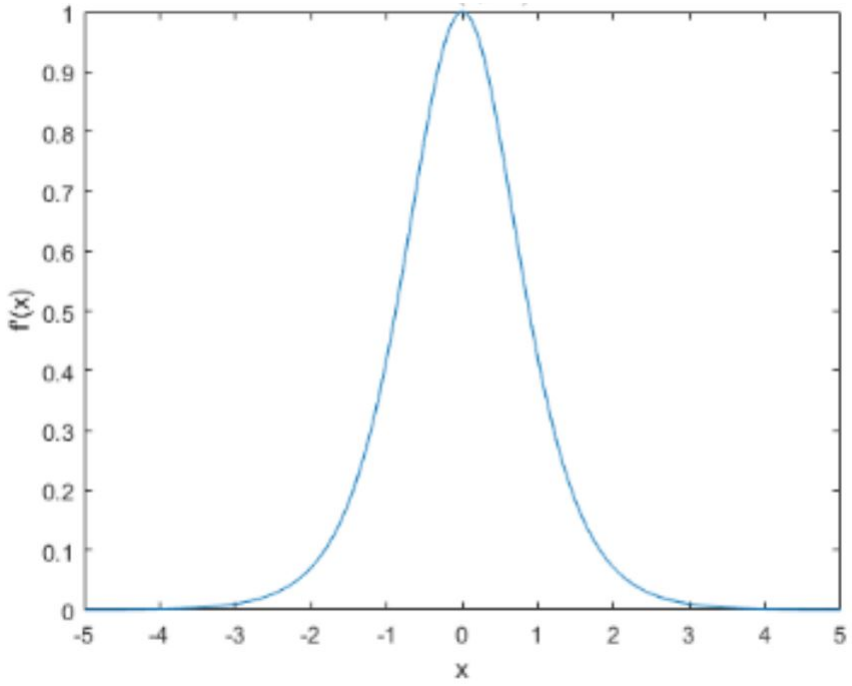

Figura 16 – Derivada tangente hiperbólica.

Fonte: (FENG; LU, 2019)

2.6.3 Softmax

A função softmax é uma das funções de ativação mais utilizadas em problemas de classificação multiclasse. O output de uma função softmax é uma distribuição de probabilidade para cada classe envolvida no problema (GAO; PAVEL, 2017).

A função softmax é representada pela equação 2.6, sua derivada é representada pela equação 2.7.

$$S_i(x_i) := \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} \tag{2.6}$$

$$S_i(x_i) = S_i(\delta_{ji} - S_j) \tag{2.7}$$

Na equação 2.6, x_i representa a probabilidade da classe naquele índice e x_j representa a probabilidade de todas as demais classes.

2.6.4 Rectified Linear Unit

A Rectified Linear Unit (ReLU) é uma das funções de ativação mais utilizadas nas aplicações recentes de redes neurais convolucionais. A ReLU é uma função não linear que possui como vantagem o fato do valor da função $f(x) = 0$ para $x_i < 0$ e x quando $x_i > 0$. Esse comportamento permite que nem todos os neurônios sejam ativados, permitindo que a rede seja esparsa, redes esparsas normalmente oferecem menos ruídos e maior poder preditivo (GOODFELLOW; BENGIO; COURVILLE, 2016).

Para entender o conceito de uma rede neural esparsa, é interessante considerar novamente a analogia de um neurônio biológico com um neurônio artificial. No corpo humano, os neurônios não são ativados todos ao mesmo tempo, cada um possui uma resposta específica aos diferentes sinais que são enviados ao cérebro. Uma rede neural artificial esparsa permite que os neurônios artificiais possam processar as partes importantes de um problema de maneira mais eficiente que uma rede neural densa (GOODFELLOW; BENGIO; COURVILLE, 2016).

A função ReLU apresentada na figura 17 e representada pela equação 2.8, sua derivada apresentada é na figura 18 e representada pela equação 2.9.

$$R(x_i) = (0, x_i) = \begin{cases} x_i, x_i > 0 \\ 0, x_i < 0 \end{cases} \tag{2.8}$$

$$R'(x_i) = \begin{cases} 1, x_i > 0 \\ 0, x_i < 0 \end{cases} \tag{2.9}$$

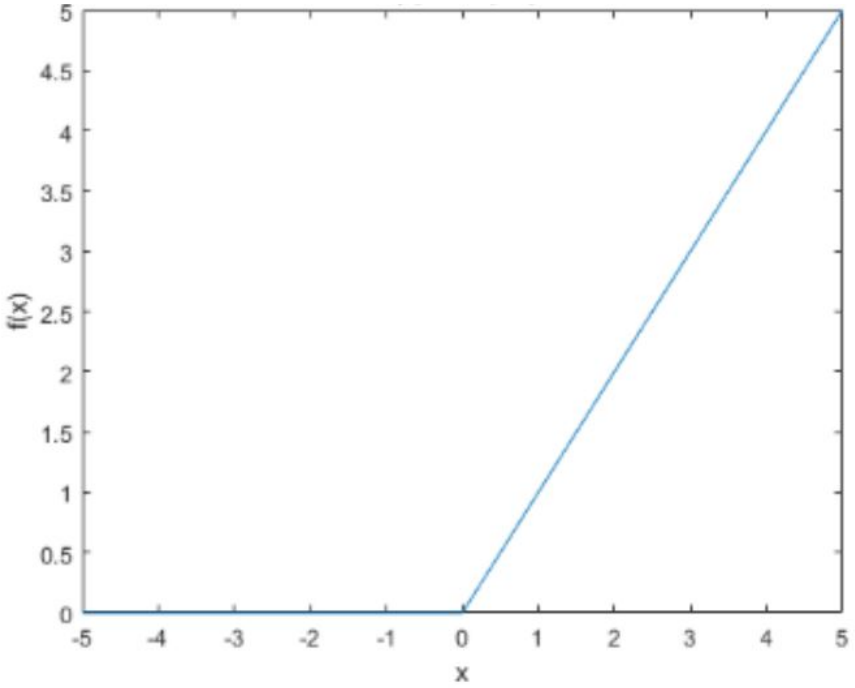

Figura 17 – *Rectified Linear Unit*.

Fonte: (FENG; LU, 2019)

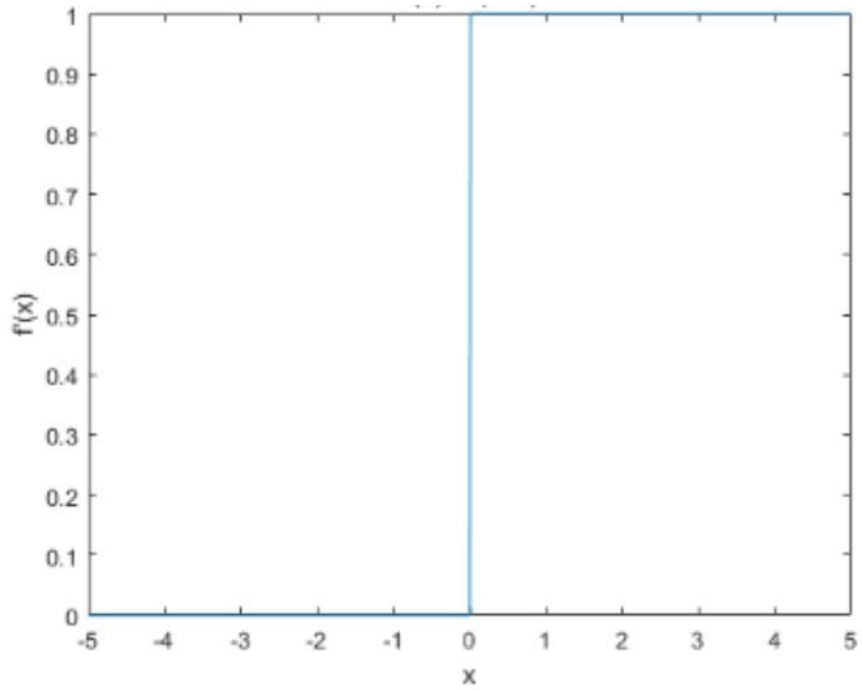

Figura 18 – Derivada *Rectified Linear Unit*.

Fonte: (FENG; LU, 2019)

A ReLU é uma das funções de ativação mais poderosas devido a estabilidade

de sua derivada, sendo 1 para $x_i > 0$ e 0 para $x_i < 0$. A estabilidade dessa função de ativação permite uma aplicabilidade muito mais eficiente que as funções sigmoide e tangente hiperbólica devido ao fato de não ocorrer a dissipação do gradiente conforme os valores de x_i aumentam (FENG; LU, 2019).

A desvantagem da ReLU é chamada de “Dying ReLu", que ocorre quando o valor resultante da soma ponderada na entrada do neurônio da aplicação da função é negativo.

Após a aplicação da função o valor de saída será zero assim como o valor da derivada. Quando essa situação ocorre, o neurônio continuamente vai gerar apenas zeros. A consequência desse acontecimento é que os pesos daquele neurônio não serão atualizados durante a etapa de backpropagation, fazendo com que uma parte da rede fique sem utilidade (FENG; LU, 2019). Existem variantes da ReLU chamadas de *Leaky ReLu* e PReLu, que tem como proposta solucionar o problema de Dying ReLu.

2.6.5 Leaky ReLu

A Leaky ReLu surgiu como uma proposta de solucionar o problema de *Dying ReLu*, presente na função Rectified Linear Unit. A ideia é fornecer uma pequena inclinação constante α_i para valores menores que zero, evitando o problema de *Dying ReLu* (GOODFELLOW; BENGIO; COURVILLE, 2016).

A função *Leaky ReLu* é apresentada na figura 19 e representada pela equação 2.10, sua derivada é apresentada na figura 20 e representada pela equação 2.11.

$$Lr(x_i) = \begin{cases} x_i, x_i > 0 \\ \alpha_i x_i, x_i < 0 \end{cases} \tag{2.10}$$

$$Lr(x_i) = \begin{cases} 1, x_i > 0 \\ \alpha_i, x_i < 0 \end{cases} \tag{2.11}$$

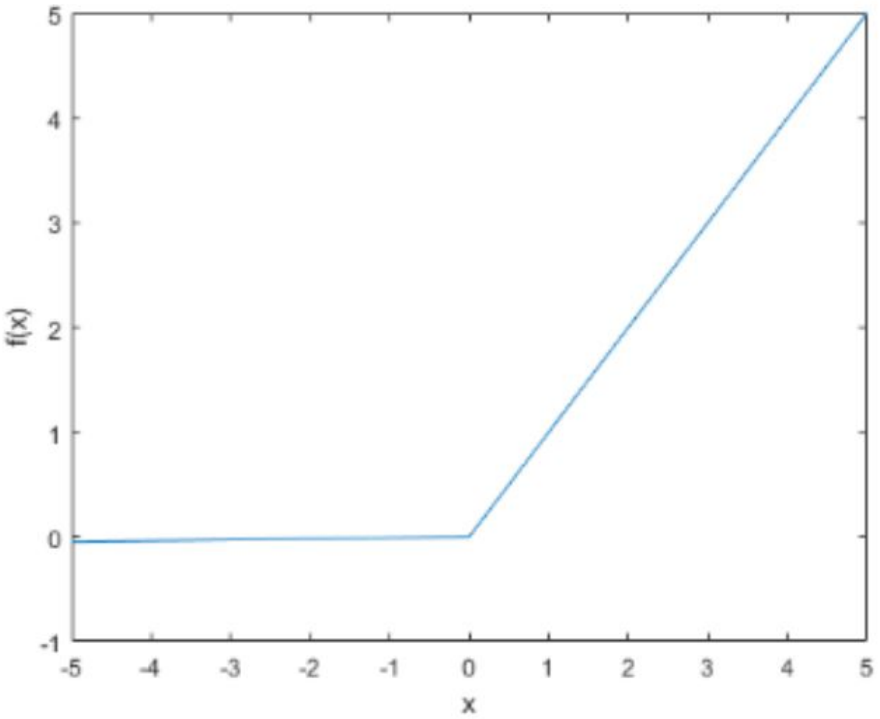

Figura 19 – Leaky ReLu.

Fonte: (FENG; LU, 2019)

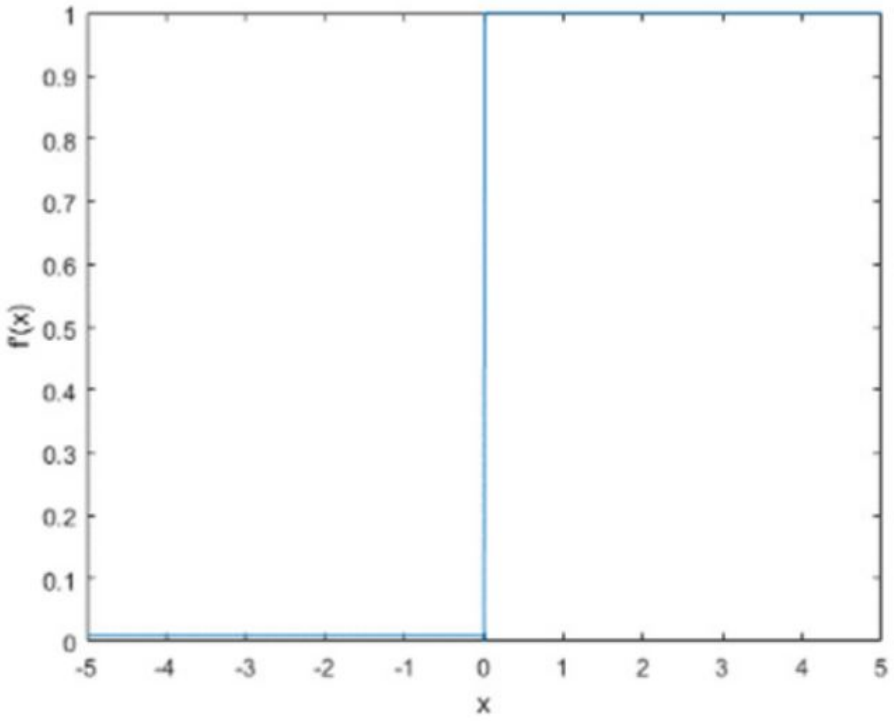

Figura 20 – Derivada Leaky ReLu.

Fonte: (FENG; LU, 2019)

2.6.6 PReLu

A função PReLu é semelhante à Leaky ReLu com a diferença que o parâmetro α_i é variável e pode ser otimizado através de *backpropagation*. Com $\alpha_i = 0$, PReLu se torna simplesmente a função ReLU e para valores pequenos e contantes de α_i, se torna a função Leaky ReLu (GOODFELLOW; BENGIO; COURVILLE, 2016).

A função PReLu e sua derivada são dadas por:

$$Pr(x_i) = \begin{cases} x_i, x_i > 0 \\ \alpha_i x_i, x_i < 0 \end{cases} \tag{2.12}$$

$$Pr'(x_i) = \begin{cases} 1, x_i > 0 \\ \alpha_i, x_i < 0 \end{cases} \tag{2.13}$$

2.7 Loss function

Diversas funções de erro (*loss functions*) podem ser utilizadas em problemas de *Deep Learning*, existem funções de erros para aplicações em problemas de regressão, onde o *output* é um valor numérico ou problemas de classificação binários/multiclasse, onde o *output* é uma determinada classe (AGGARWAL, 2018).

Funções de erro normalmente utilizadas por problemas de regressão são descritas nas equações 2.14, 2.15 e ??.

A equação 2.14 é denominada *Mean Square Error* e é medida como a média das diferenças do quadrados entre o valor real y e o valor de *output* fornecido pela rede neural $\bar{y}$ (AGGARWAL, 2018).

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y}_i)^2 \tag{2.14}$$

O RMSE (Root Mean Square Error) é bastante popular entre problemas de regressão e consiste na raiz quadrada do MSE. O RMSE é descrito pela equação 2.15:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.15}$$

O *Mean Absolute Error* é medido como a média da soma das diferenças absolutas entre o valor real y e o valor de *output* fornecido pela rede neural $\bar{y}$ (AGGARWAL, 2018).

$$\text{MAE} = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n} \tag{2.16}$$

A equação 2.17 representa a Cross Entropy Loss também conhecida como Log Loss. Essa função de custo mede a diferença da distribuição de probabilidades, onde essas

distribuições estão relacionadas com a ocorrência real e a prevista calculada pelo modelo. Na equação 2.17, y_k representa o valor real e $\hat{y}_k$ representa o valor estimado pelo modelo (NASR; BADR; JOUN, 2002).

$$_n = \frac{1}{n} \sum_{k=1}^{n} [y_k \ln \hat{y}_k + (1 - y_k) \ln (1 - \hat{y}_k)] \tag{2.17}$$

2.8 Otimização

Grande parte dos algoritmos de *Deep Learning* envolvem otimização, isto é, minimizar ou maximizar alguma função $f(z)$ com a variação de z. Normalmente a maioria dos problemas que utilizam Deep Learning buscam minimizar $f(z)$, enquanto a otimização ocorre através da minimização de $-f(z)$. Nos problemas de *Deep Learning*, o principal objetivo é minimizar a função objetivo, que também pode ser denominada de função de perda (*Loss Function*) (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.8.1 *Gradient descent*

O gradiente descendente é um dos algoritmos mais utilizados para os problemas de otimização em *Deep Learning*. O objetivo do gradiente descendente é encontrar o mínimo de uma função arbitrária iterativamente.

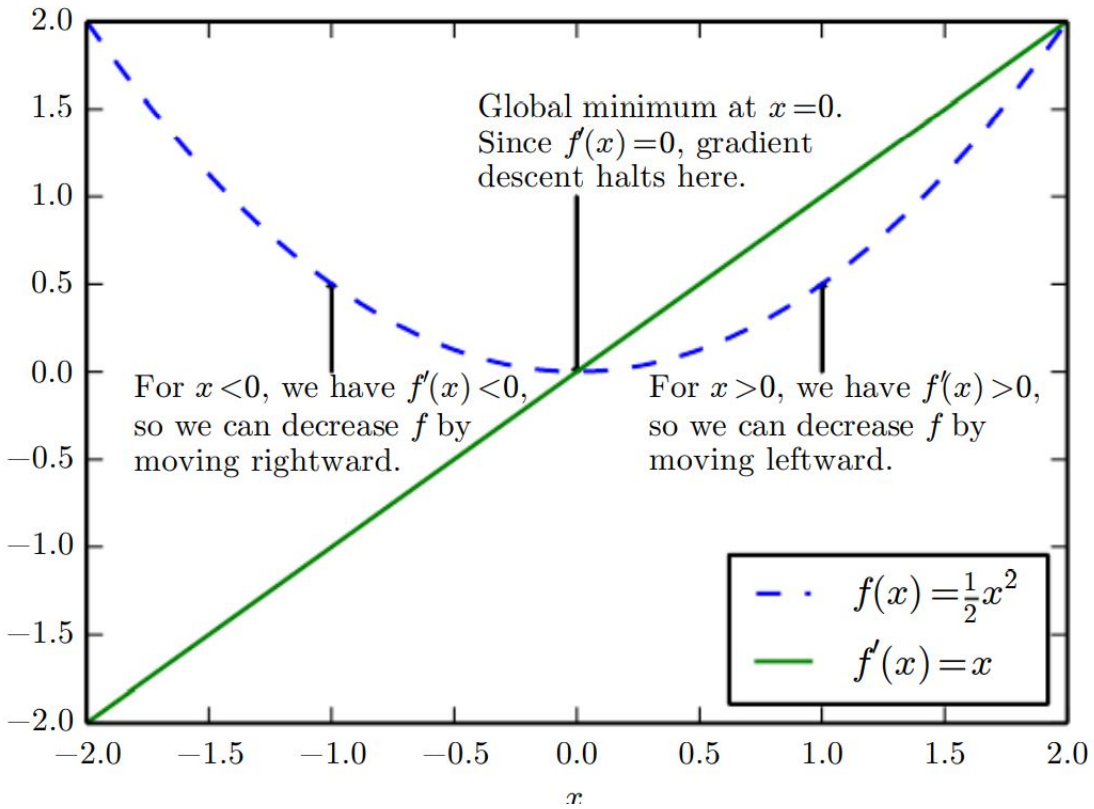

Figura 21 – *Gradient Descent*.

Fonte: (GOODFELLOW; BENGIO; COURVILLE, 2016)

A Figura 21 representa como as derivadas de uma função genérica permitem a descida do gradiente até um ponto de mínimo global.

Considerando a função sigmoide descrita na seção 2.6.1:

$$\sigma(x) = \frac{1}{1 + e^{x}} \tag{2.18}$$

A primeira etapa para calcular a descida do gradiente é tomar a derivada da função de ativação, considerando uma rede neural artificial simples com função de ativação no neurônio de saída sendo a função sigmoide:

$$\begin{aligned} \sigma\,(x) &= \frac{\partial}{\partial x}\frac{1}{1 + e^{-x}} \\ &= \frac{e^{-x}}{(1 + e^{-x})^{2}} \\ &= \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}} \\ &= \sigma(x)(1 - \sigma(x)) \end{aligned} \tag{2.19}$$

Considerando diversos inputs $(x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)} \ldots, x^{(n)})$, pode- se utilizar a formula de erro apresentada na equação 2.20:

$$E = -\frac{1}{n} \sum_{i=1}^{n} (y_i \ln(\hat{y}_i) + (1 - y_i) \ln(1 - \hat{y}_i)) \tag{2.20}$$

Onde y_i é o label (valor exato) fornecido para a rede neural e o valor predito pela saída da rede neural é calculado por $\hat{y}_i = \lambda(wx^{(i)} + b)$.

Em seguida deve-se calcular o gradiente da função de custo E, nos pontos $(x_1, x_2, \ldots, x_n)$ através das derivadas parciais.

$$E = \left[\frac{\partial}{\partial w_1} E, \cdots, \frac{\partial}{\partial w_n} E, \frac{\partial}{\partial b} E \right] \tag{2.21}$$

Pode-se considerar o erro produzido por cada ponto para simplificação de cálculos, em seguida deve-se calcular a derivada desse ponto. O erro total será a média dos erros de todos os pontos.

O erro produzido por cada ponto é descrito pela equação 2.22:

$$E = -y \ln(\hat{y}) - (1 - y) \ln(1 - \hat{y}) \tag{2.22}$$

Para calcular a derivada desse erro em relação aos pesos, deve-se inicialmente calcular a derivada do output em relação aos pesos, logo $\frac{\partial}{\partial w_i} \hat{y}$ onde $\hat{y} = \lambda(Wx + b)$.

Portanto :

$$\begin{aligned}
\frac{\partial}{\partial w_i} \hat{y} &= \frac{\partial}{\partial w_i} \lambda(wx + b) \\
&= \lambda(wx + b)(1 - \lambda(wx + b)) \cdot \frac{\partial}{\partial w_i} (wx + b) \\
&= \hat{y}(1 - \hat{y}) \cdot \frac{\partial}{\partial w_i} (wx + b) \\
&= \hat{y}(1 - \hat{y}) \cdot \frac{\partial}{\partial w_i} (w_1 x_1 + \cdots + w_i x_i + \cdots + w_n x_n + b) \\
&= \hat{y}(1 - \hat{y}) \cdot x_i
\end{aligned} \tag{2.23}$$

A próxima etapa é calcular a derivada da função de custo E em um ponto x, em relação aos pesos tensoriais.

$$
\begin{aligned}
\frac{\partial}{\partial w_i} E &= \frac{\partial}{\partial w_i}[-y \log(\hat{y}) - (1-y)\log(1-\hat{y})] \\
&= -y \frac{\partial}{\partial w_i} \log(\hat{y}) - (1-y)\frac{\partial}{\partial w_i} \log(1-\hat{y}) \\
&= -y \cdot \frac{1}{\hat{y}} \cdot \frac{\partial}{\partial w_i}\hat{y} - (1-y) \cdot \frac{1}{1-\hat{y}} \cdot \frac{\partial}{\partial w_i}(1-\hat{y}) \\
&= -y \cdot \frac{1}{\hat{y}} \cdot \hat{y}(1-\hat{y})x_i - (1-y) \cdot \frac{1}{1-\hat{y}} \cdot (-1)\hat{y}(1-\hat{y})x_i \\
&= -y(1-\hat{y}) \cdot x_i + (1-y)\hat{y} \cdot x_i \\
&= -(y-\hat{y})x_i
\end{aligned}
\tag{2.24}
$$

Repetindo o mesmo processo para o *bias*, temos que tomando a derivada da função de custo em relação ao bias tem-se:

$$
\frac{\partial}{\partial b} E = -(y-\hat{y}) \tag{2.25}
$$

Pode-se perceber que através da equação 2.24 que a derivada do gradiente é dada por $E = -(y-\hat{y})\,(x_1, x_2, x_3, \ldots, x_n, 1)$. Com a equação pode-se inferir que quanto mais próximo o valor predito do valor real menor é o gradiente (AGGARWAL, 2018).

O gradiente descendente consiste em subtrair um múltiplo do gradiente da função de custo em cada ponto, atualizando os pesos com a equação 2.26 e o *bias* com a equação 2.27. O parâmetro β das equações 2.26 e 2.27 é chamado de *Learning Rate*, que determina o tamanho do passo em cada iteração (AGGARWAL, 2018).

$$
w_i \leftarrow w_i + \beta(y-\hat{y})x_i \tag{2.26}
$$

$$
b \leftarrow b + \beta(y-\hat{y})x_i \tag{2.27}
$$

2.8.2 *Feedforward*

O *Feedforward* consiste na etapa de propagação do sinal através dos neurônios de entrada, é o mesmo processo descrito na seção 2.3. A informação de entrada é propagada até as camadas intermediárias, onde ocorre a definição de uma função f, que é utilizada para encontrar uma saída y (ZHANG et al., 2007). A Figura 22 apresenta esse processo em um *Multilayer Perceptron* de duas camadas.

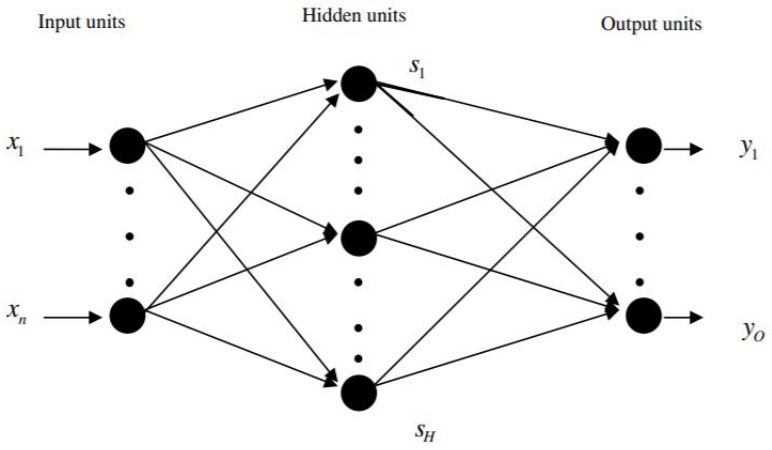

Figura 22 – *Feedfoward Neural Network*.

Fonte : (ZHANG et al., 2007)

2.8.3 *Backpropagation*

Após a base de *feedforward*, uma saída $\hat{y}$ é gerada além de um valor de erro $E(\theta)$. A fase de *backpropagation* consiste em propagar o erro no sentido contrário, geralmente utilizando o gradiente descendente descrito na seção 2.8.1.

Para o processo de *backpropagation* é necessário que, tanto a função de erro quanto a função de transferência sejam diferenciáveis. Em redes neurais profundas são utilizadas soluções computacionais para acelerar e simplificar o processo,o qual é composto por diversas operações de álgebra linear e cálculo multivariável (AGGARWAL, 2018).

O processo de *backpropagation* é iterativo e sua principal função consiste em utilizar o gradiente descendente da função de perda com relação aos pesos para atualizar os pesos e *bias* entre todas as camadas da rede neural (*input layer - hidden layer - output layer*), conforme as equações 2.26 e 2.27 durante um número *n* de iterações ou até um valor de erro suficiente na camada de saída (AGGARWAL, 2018).

A Figura 23 apresenta uma rede neural com centenas de neurônios indicando o sentido de *feedforward* e *backpropagation*.

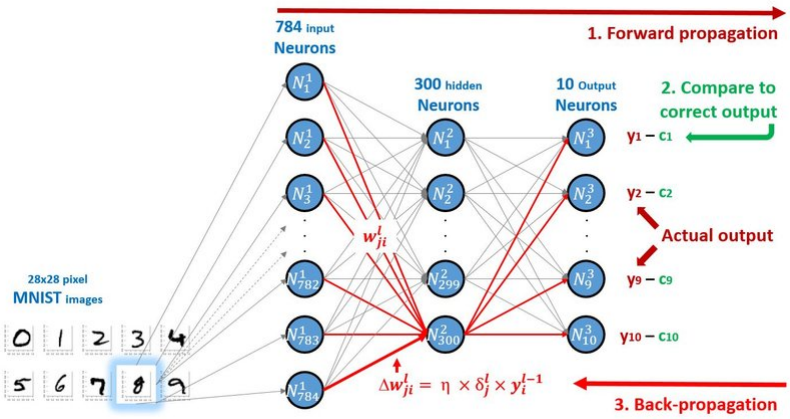

Figura 23 – *Backpropagation*.

Fonte: (KHACEF; NASSIM; MIRAMOND, 2018)

3 Metodologia

A Figura 51 representa o planejamento desse trabalho, que é dividido em três partes principais: problematização, revisão de literatura e a modelagem do problema utilizando *Deep Learning*. As etapas serão detalhadas nas seções 3.1, 3.2, 3.3.1 e 3.3.2.

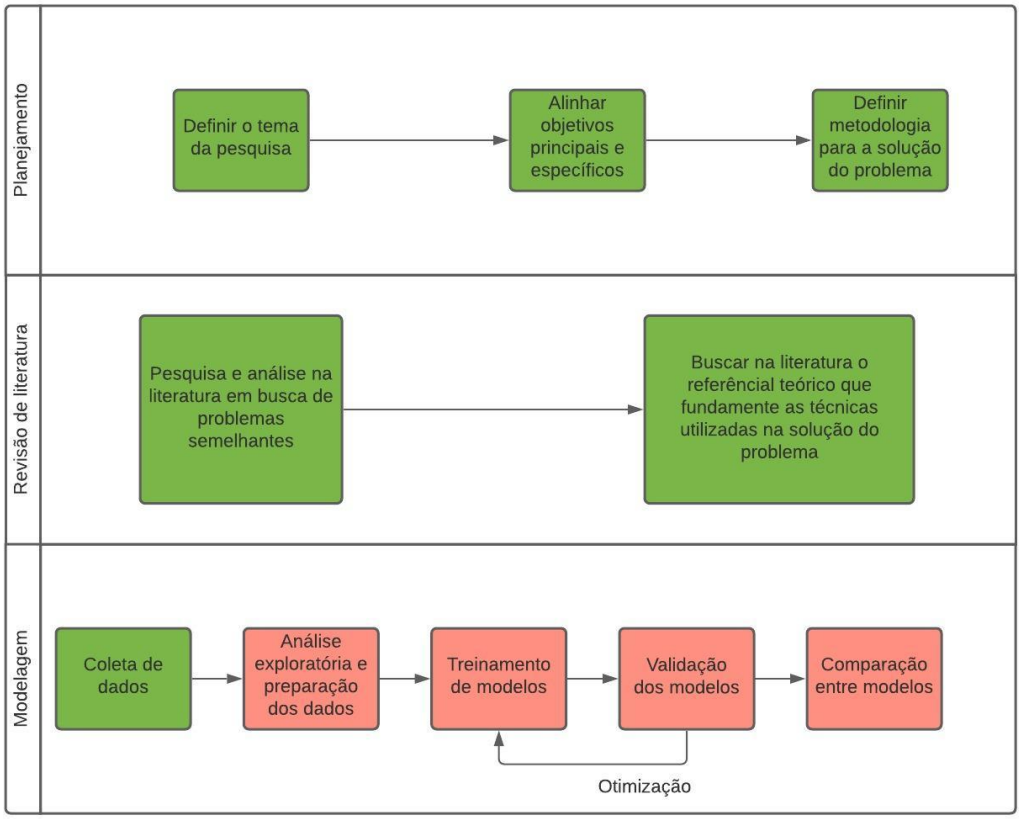

Figura 24 – Fluxograma do método de trabalho

Fonte: Autoral

3.1 Planejamento

A primeira etapa consistiu em buscar um possível problema que possa ser solucionado usando redes neurais na industria aeroespacial.

O problema escolhido para ser o tema desse trabalho é uma área de grande importância nas próximas décadas para a indústria aeroespacial, portanto, iniciar um trabalho visando a utilização de inteligência artificial em veículos aéreos na graduação abre um leque de oportunidades em pesquisas futuras.

3.2 Revisão Literatura

A revisão de literatura no trabalho pode ser dividida em duas grandes frentes. A primeira é relacionada com a busca na literatura por trabalhos que utilizaram *Deep Learning*, especialmente no campo de imagens. A segunda frente foi buscar na literatura à base matemática que pudesse consolidar as técnicas que serão utilizadas na modelagem computacional do problema.

A fonte principal na busca de trabalhos semelhantes foram periódicos científicos, sendo o *Science Direct* fonte primária. A parte matemática do trabalho foi baseada principalmente em livros consolidados sobre *Deep Learning*.

3.3 Desenvolvimento

A modelagem do problema utilizando redes convolucionais é dividida em várias etapas que serão explicitadas na seções 3.3.1 e 3.3.2.

3.3.1 Coleta de dados

Em um projeto de *Deep Learning*, os dados são os principais insumos para solucionar o problema. A coleta de dados foi iniciada antes mesmo do referencial teórico, pois não faria sentido construir um referencial teórico se não fosse possível encontrar os dados que vão suportar o problema.

A ideia inicial da coleta de dados consistia em procurar por coordenadas geográficas de localizações onde possivelmente existiriam heliportos. Com as coordenadas, é possível realizar um trabalho de extração de imagens daquela localidade utilizando as ferramentas do Google.

Essa ideia inicial acabou não sendo necessária pois, em 2019, o pesquisador Emre Baseski construiu um banco de dados composto por diferentes imagens de heliportos, em diferentes localidades do planeta com diferentes características (BAşESKI, 2019). No seu trabalho o autor explicita como ocorreu a coleta dessas imagens. Apesar do banco de dados estar ponto, é possível utilizar a mesma técnica utilizada no trabalho de Emre Baseski para conseguir mais imagens para diversificar o banco de dados caso seja necessário.

Figura 25 – Heliporto.

Fonte: (BAŞESKI, 2019)

Figura 26 – Heliporto.

Fonte: (BAŞESKI, 2019)

Figura 27 – Heliporto.

Fonte: (BAşESKI, 2019)

As Figuras 26 e 27 representam exemplos de heliportos que estão no banco de dados.

3.3.2 Modelagem

A modelagem do trabalho será construída conforme o fluxo da imagem 28. O workflow é autoral, porém segue o modelo difundido de workflow especializado em projetos de inteligência artificial, conforme atestado pelo artigo "Guiding the Creation of Deep Learning-based Object Detectors"(CASADO-GARCíA; HERAS, 2018).

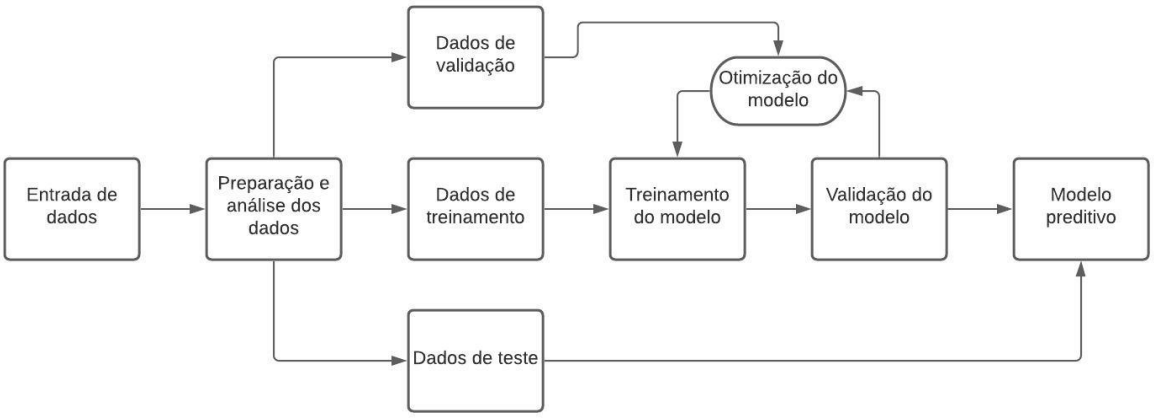

Figura 28 – *Workflow* do projeto.

Fonte: Autoral

A ideia do trabalho é construir um algoritmo especializado no reconhecimento de objetos em diferentes imagens de heliportos. As imagens serão os insumos utilizados pelo algoritmo *Yolo* (*You Only Look Once*). A função dos algoritmos consiste em buscar por padrões dentro da imagem. Os algoritmos, além de identificar, também são capazes de fornecer as coordenadas da posição em que o objeto está localizado.

A primeira tarefa que será realizada durante o processo de modelagem consiste em avaliar a qualidade do banco de dados e enriquecê-lo com novas imagens, caso seja necessário. Os dados de entrada serão divididos em treinamento, validação e dados de teste conforme o *workflow* da Figura 28. Este fluxo é amplamente utilizado na academia e no mercado nos projetos de inteligência artificial.

A modelagem consistirá na utilização de algoritmos convolucionais especializados na detecção de objetos em imagens, como o Yolo descrito na seção 2.5. Após a construção dos modelos, o algoritmo será otimizado na busca do menor erro possível.

Após a validação do algoritmo, o resultado final consistirá em uma análise minuciosa dos resultados utilizando diversas métricas de avaliação. O tempo necessário para o treinamento do modelo e detecção dos objetos também será avaliado. Um *threshold* interessante para o trabalho de conclusão de curso é acima de 80%.

As principais ferramentas utilizadas para desenvolvimento do trabalho serão o Python e Jupyter Notebook. O Python possui diversas bibliotecas especializadas em *Deep Learning* que suportam álgebra tensorial e cálculo multivariável como TensorFlow, Pytorch, Keras e OpenCV.

O computador no qual será feito o treinamento é composto por um processador AMD Ryzen 2600, 16 Gb de memória Ram e GPU Rx 580. As GPU(Graphics Processing

Unit) são utilizadas no treinamento de modelos de *Deep Learning* pois aceleram o processo de treinamento através do paralelismo, que permite realizar mais cálculos simultaneamente do que uma CPU tradicional (MADIAJAGAN; RAJ, 2019).

4 Resultados

O capítulo apresenta os resultados obtidos com a modelagem utilizando o modelo de Deep Learning descrito na seção 2.5.

4.1 Configurações computacionais

Para o treinamento do modelo Yolo foi utilizado o processo conhecido como Transfer Learning, que consiste em utilizar uma rede neural pré-treinada. A rede neural pré-treinada utiliza um modelo que foi treinado em um dataset diferente do utilizado na tarefa final.

A vantagem de realizar o processo de Transfer Learning é utilizar as técnicas de otimização descritas na seção 2.8.1 para generalizar os pesos da rede neural que serão aplicados no modelo de detecção de heliportos. O processo de treinamento também ocorre em velocidade superior, pois a estrutura da rede neural e os pesos iniciais constituem o modelo pré-treinado (MA et al., 2021).

O modelo foi treinado no ambiente virtual disponibilizado pelo Google, denominado Google Colab. A vantagem de utilizar o ambiente virtual é que o Google disponibiliza um modelo de placa de vídeo (GPU Tesla T4), o qual é capaz de acelerar o processo de treinamento. A arquitetura utilizada no modelo foi o YoloV5.

matplotlib - 3.2.2
numpy - 1.18.5
opencv-python - 4.1.2
Pillow - 7.1.2
PyYAML - 5.3.1
requests - 2.23.0
scipy - 1.4.1
torch - 1.7.0
torchvision - 0.8.1
tqdm - 4.41.0
tensorboard - 2.4.1
pandas - 1.1.4
seaborn - 0.11.0

Tabela 1 – Bibliotecas Utilizadas

A Tabela 1 apresenta as principais bibliotecas utilizadas para suportar o modelo.

4.2 Pré-processamento

Para o treinamento do modelo foram utilizadas 107 imagens do dataset de treinamento e 30 imagens do dataset validação. O dataset de validação é importante para o modelo, pois suas métricas são utilizadas durante o processo de otimização descrito na seção 2.8.

A etapa de pré-processamento mais importante para um modelo de detecção de objetos utilizando o modelo Yolo está relacionada às annotations da imagem. As annotations são utilizadas com o objetivo de que o modelo reconheça o objeto que deve ser identificado durante o processo de treinamento.

A Figura 29 representa como as annotations da figura localizam o centro e as coordenadas de um determinado objeto dentro de uma imagem.

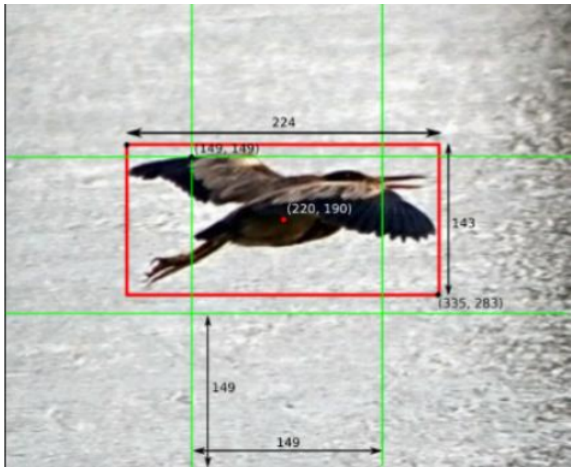

Figura 29 – *Annotation*.

Fonte: (Upulie Handalage; Lakshini Kuganandamurthy, 2021)

Inicialmente, as imagens de heliportos do dataset estavam com as annotations no formato PascalVoc, as quais não podem ser utilizados pelo modelo Yolo.

A Figura 30 representa as annotations no formato PascalVoc, indicando as coordenadas de localização do objeto alvo na imagem.

O formato PascalVoc não pode ser utilizado pelo Yolo, porém é possível converter para um arquivo txt, que pode ser utilizado pelo modelo. A Figura 31 representa esse novo formato de annotations.

```
<?xml version='1.0' encoding='utf-8'?>
<annotation>
    <folder>JPEGImages</folder>
    <filename>h50001.jpg</filename>
    <path>/home/joao/proje/heliport2/JPEGImages/h50001.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>640</width>
        <height>640</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>heliport</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>257</xmin>
            <ymin>160</ymin>
            <xmax>382</xmax>
            <ymax>310</ymax>
        </bndbox>
    </object>
</annotation>
```

Figura 30 – Annotations em PascalVoc.

Fonte: Autoral

```
0 0.49759615384615385 0.36538461538461536 0.19471153846153846 0.234375
```

Figura 31 – Annotations em formato *Yolo*.

Fonte: Autoral

A Figura 32 representa a imagem do dataset que possui as coordenadas representadas pelas Figuras 30 e 31.

Figura 32 – Imagem do dataset.

Fonte: Autoral

4.3 Treinamento e validação

Os principais parâmetros definidos para o treinamento do modelo são apresentados na tabela 2.

Nome	Valor
Learning Rate	0.01
Batch Size	16
Weight decay	0.0005
Layers	213
Parameters	7012822
Otimizador	Stochastic Gradient Descent
Epochs	150

Tabela 2 – Principais hiperparâmetros do modelo

As Figuras 33 e 34 representam um *batch* de imagens de treinamento e validação, o *Batch Size* consiste na quantidades de amostras que são propagadas pela rede neural.

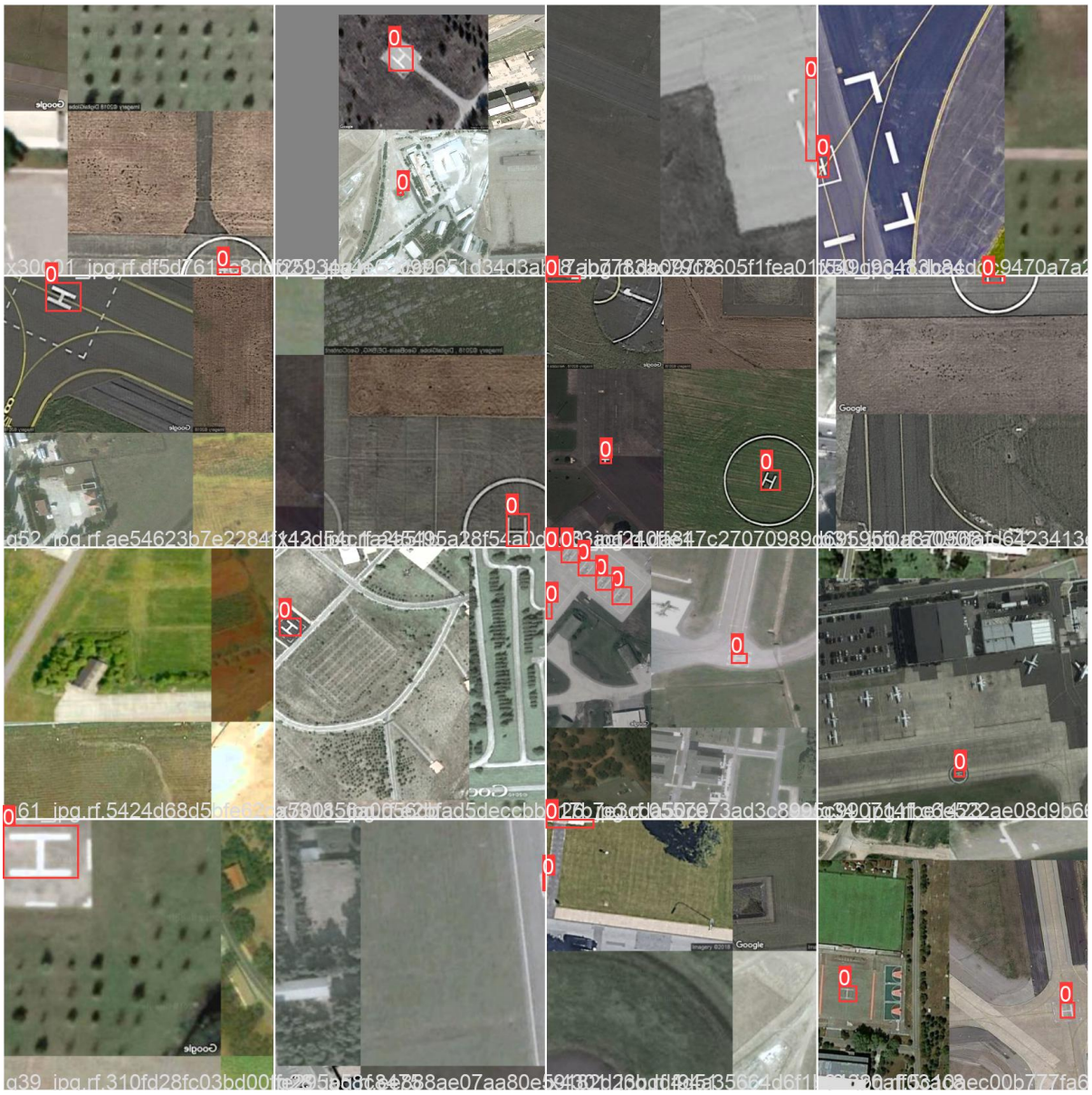

Figura 33 – Batch de imagens de treinamento.

Fonte: Autoral

Figura 34 – Batch de imagens de validação.

Fonte: Autoral

A Figura 35 representa a Box Loss, que é uma Loss Function utilizada pelo Yolo. O objetivo de qualquer modelo de Deep Learning é minimizar essa função através de processos de otimização.

O modelo apresentado nesse trabalho utiliza o Stochastic Gradient Descent como otimizador. A Box Loss representa o quão bem o algoritmo consegue localizar o centro de um objeto, além da cobertura do objeto pelas Bounding Boxes. A equação 4.1 representa a formulação matemática da Loss Function.

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} 1_i^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2
\end{aligned}
\tag{4.1}
$$

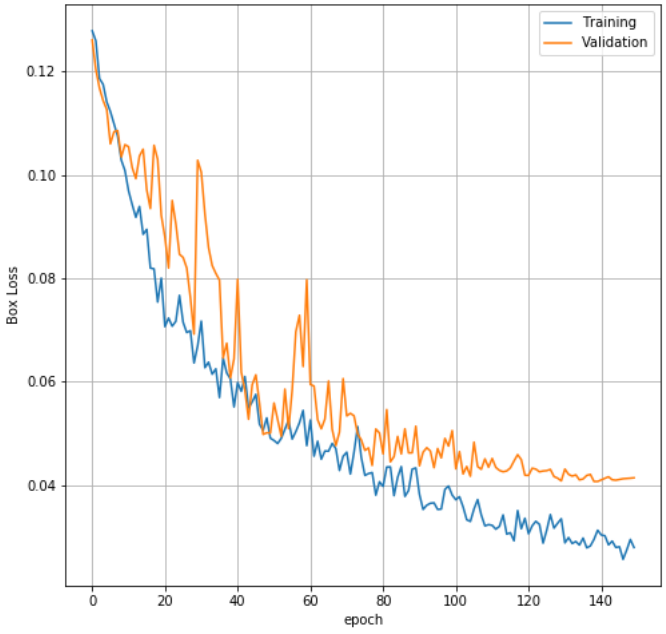

Figura 35 – Box Loss com 150 *Epochs*.

Fonte: Autoral

Considerando a Figura 35 pode-se notar que é interessante treinar o modelo por aproximadamente 150 Epochs. Utilizar um grande número de Epochs no treinamento do modelo resulta na divergência entre a Box Loss no Dataset de treinamento e validação, indicando Overfitting. A Figura 36 indica esse cenário com um modelo treinado utilizando 6000 Epochs.

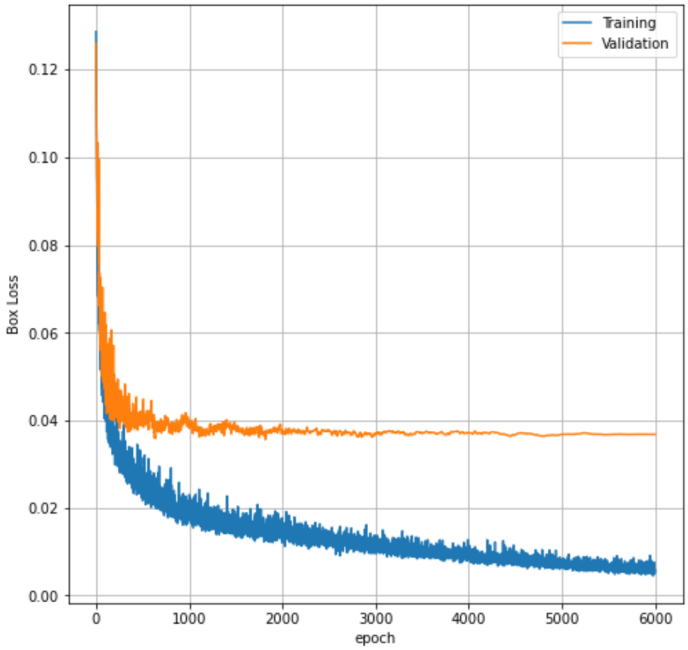

Figura 36 – Box Loss com 6000 *Epochs*.

Fonte: Autoral

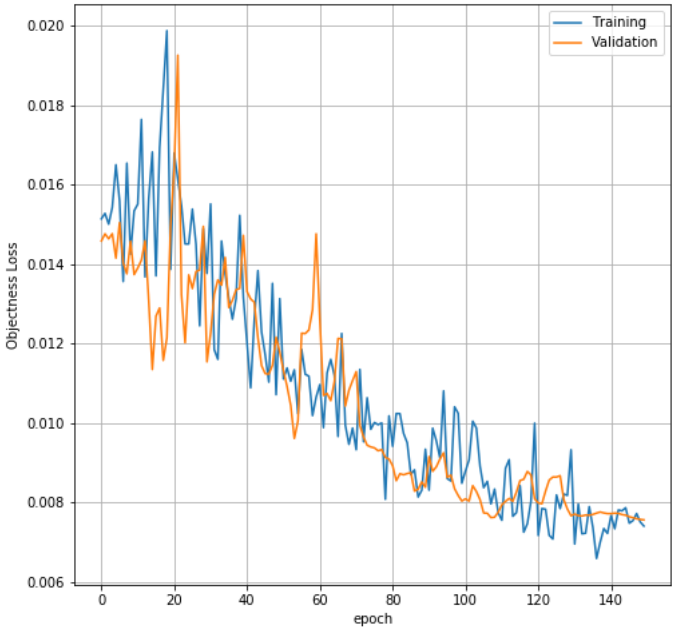

Figura 37 – *Objectness Loss*.

Fonte: Autoral

A Figura 37 representa a Objectness Loss, que consiste na probabilidade de um

determinado objeto existir na região de interesse. A Objectness Loss é representada pelo parâmetro C_i na equação 4.1.

4.4 Detecção

Após o processo de treinamento e validação, o modelo treinado que apresentou as melhores métricas foi utilizado para o processo de detecção. O tempo de treinamento é relativamente longo, variando entre 2 e 12 horas dependendo do número de epochs.

Figura 38 – Heliporto.

Fonte: Autoral

Figura 39 – Heliporto.

Fonte: Autoral

Figura 40 – Heliporto.

Fonte: Autoral

Figura 41 – Heliporto.

Fonte: Autoral

Figura 42 – Heliporto.

Fonte: Autoral

Figura 43 – Heliporto.

Fonte: Autoral

Figura 44 – Heliporto.

Fonte: Autoral

Figura 45 – Heliporto.

Fonte: Autoral

Figura 46 – Heliporto.

Fonte: Autoral

Figura 47 – Heliporto.

Fonte: Autoral

A Figura 48 representa as curvas de precision e recall, que são métricas comumente utilizadas para avaliar o modelo Yolo. As equações 4.2 e 4.3 representam os parâmetros de cada métrica.

Na equação 4.2, TP são os verdadeiros positivos e FP são os falsos positivos. Na equação 4.3, TP são os verdadeiros positivos e FN são os falsos negativos.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.2}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.3}$$

Figura 48 Precision e Recall.

Fonte: Autoral

A Figura 49 indica a Mean Average Precision, a qual é calculada como a média ponderada da precisão em diferentes threshold. Na equação 4.4, AP_i é a precisão média para a classe i e N é o número total de classes.

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{4.4}$$

Figura 49 Mean Average Precision Curve.

Fonte: Autoral

A Figura 50 representa a curva F1. O score F1 é uma métrica que representa o balanceamento entre as métricas precision e recall, que foram apresentadas anteriormente.

$$F1 = 2 \frac{Precision \quad Recall}{Precision + Recall} \tag{4.5}$$

Figura 50 F1 Curve.

Fonte: Autoral

4.5 Discussão

O modelo Yolo apresenta as métricas de precision e recall na faixa de 82%, além de um F1 score de 80%. Os resultados se mostraram consistentes nas imagens de teste.

Considerando as imagens de detecção, é possível perceber que o algoritmo consegue reconhecer claramente os heliportos independente da altura e posição do heliporto na imagem, mas o algoritmo encontra di culdades para detectar heliportos que foram degradados pela ação do tempo.

Analisando a Figura 35, na qual é considerado o decréscimo(da ss) ao longo das epochs, pode-se perceber que não é interessante treinar o modelo por um número grande de epochs, pois o modelo tende ao over tting.

Considerando o treinamento por 150 epochs, o tempo necessário para treinar o modelo é próximo de 10 horas na GPU. Aproximadamente 107 imagens foram utilizadas durante o treinamento, pode-se concluir que o treinamento possui bastante custo computacional, pois o banco de dados não é extenso. O processo de detecção é rápido, sendo necessários aproximadamente 6.7ms por imagem.

Na primeira parte do trabalho de conclusão de curso foi estipulada uma precisão na ordem de 80%, o resultado obtido no trabalho de 82% é superior a esse threshold.

No processo de modelagem diversos parâmetros foram otimizados com o objetivo

de obter um resultado maior de precisão, porém não representaram grandes ganhos. Aumentar a quantidade de amostras do banco de dados possui potencial de melhorar os resultados, porém com o trade-o de aumentar o tempo de treinamento.

5 Conclusão

A evolução dos algoritmos de inteligência arti cial estão possibilitando grandes avanços nos campos de reconhecimento de imagens e visão computacional em diversos segmentos. Avanços nos campos de saúde, indústria automotiva e aeroespacial estão moldando a utilização da inteligência arti cial. Na indústria aeroespacial, os veículos autônomos estão em grande destaque na academia e no mercado, com grande projeção de crescimento para a próxima década. O objetivo do trabalho consistiu em utilizar redes neurais convolucionais para construir um modelo de Deep Learning capaz de reconhecer heliportos em diferentes ambientes que possam ser utilizados por veículos autônomos.

O trabalho consistiu em forjar o referencial teórico, o qual é a base do modelo que foi implementado na segunda parte do trabalho de conclusão do curso. Além do referencial, a metodologia também foi de nida seguindo critérios utilizados em trabalhos anteriores na área de Deep Learning É importante citar que, além do processo de construção do referencial e modelagem, a busca por um banco de dados de qualidade para a implementação do modelo também foi um ponto chave do trabalho.

Os resultados obtidos pelo modelo construído durante o trabalho de conclusão de curso foram satisfatórios. Trabalhos futuros derivados desse projeto também foram mapeados e consistem em otimizar o modelo construído durante o trabalho e realizar uma implementação prática em escala do algoritmo em tempo real.

Referências

AGGARWAL, C. C. Neural Networks and Deep Learning. Springer International Publishing, 2018. Disponível em: <https://doi.org/10.1007/978-3-319-94463-0>. Citado 3 vezes nas páginas 44, 48 e 49.

AL-KAFF, A. et al. Survey of computer vision algorithms and applications for unmanned aerial vehicles. Expert Systems with Applications, Elsevier BV, v. 92, p. 447 463, fev. 2018. Disponível em: <https://doi.org/10.1016/j.eswa.2017.09.033>. Citado na página 20.

AVUÇLU, E.; BA“ÇIFTÇI, F. New approaches to determine age and gender in image processing techniques using multilayer perceptron neural network. Applied Soft Computing, Elsevier BV, v. 70, p. 157 168, set. 2018. Disponível em: <https://doi.org/10.1016/j.asoc.2018.05.033>. Citado na página 27.

BA³ESKI, E. A new remote sensing dataset for heliport detection. In: 2019 9th International Conference on Recent Advances in Space Technologies (RAST). [S.l.: s.n.], 2019. p. 419 422. Citado 3 vezes nas páginas 52, 53 e 54.

BöROLD, A. et al. Recognition of car parts in automotive supply chains by combining synthetically generated training data with classical and deep learning based image processing. Procedia CIRP, Elsevier BV, v. 93, p. 377 382, 2020. Disponível em: <https://doi.org/10.1016/j.procir.2020.03.142>. Citado na página 19.

BOUZGOU, H. Advanced Methods for the Processing and Analysis of Multidimensional Signals: Application to Wind Speed. Tese (Doutorado), 01 2012. Citado na página 36.

CASADO-GARCíA, ; HERAS, J. Guiding the creation of deep learning-based object detectors. 09 2018. Citado na página 54.

CHENG, C.; ADULYASAK, Y.; ROUSSEAU, L.-M. Drone routing with energy function: Formulation and exact algorithm. Transportation Research Part B: Methodological, Elsevier BV, v. 139, p. 364 387, set. 2020. Disponível em: <https://doi.org/10.1016/j.trb.2020.06.011>. Citado na página 20.

COKYASAR, T. Optimization of battery swapping infrastructure for e-commerce drone delivery. Computer Communications, Elsevier BV, v. 168, p. 146 154, fev. 2021. Disponível em: <https://doi.org/10.1016/j.comcom.2020.12.015>. Citado na página 20.

COKYASAR, T. et al. Designing a drone delivery network with automated battery swapping machines. Computers & Operations Research, Elsevier BV, v. 129, p. 105177, maio 2021. Disponível em: <https://doi.org/10.1016/j.cor.2020.105177>. Citado na página 20.

CUN, Y. L. et al. Handwritten digit recognition: Applications of neural net chips and automatic learning. In: Neurocomputing. Springer Berlin Heidelberg, 1990. p. 303 318. Disponível em: <https://doi.org/10.1007/978-3-642-76153-9_35>. Citado na página 24.

DANESHJOU, R. et al. How to evaluate deep learning for cancer diagnostics factors and recommendations. Biochimica et Biophysica Acta (BBA) - Reviews on Cancer, Elsevier BV, v. 1875, n. 2, p. 188515, abr. 2021. Disponível em: <https://doi.org/10.1016/j.bbcan.2021.188515>. Citado na página 19.

DUKKANCI, O.; KARA, B. Y.; BEKTA“, T. Minimizing energy and cost in range-limited drone deliveries with speed optimization. Transportation Research Part C: Emerging Technologies, Elsevier BV, v. 125, p. 102985, abr. 2021. Disponível em: <https://doi.org/10.1016/j.trc.2021.102985>. Citado na página 20.

FENG, J.; LU, S. Performance analysis of various activation functions in arti cial neural networks. Journal of Physics: Conference Series, IOP Publishing, v. 1237, p. 022030, jun. 2019. Disponível em: <https://doi.org/10.1088/1742-6596/1237/2/022030>. Citado 6 vezes nas páginas 37, 38, 39, 41, 42 e 43.

FENG, X. et al. Computer vision algorithms and hardware implementations: A survey. Integration, Elsevier BV, v. 69, p. 309 320, nov. 2019. Disponível em: <https://doi.org/10.1016/j.vlsi.2019.07.005>. Citado na página 19.

FOTOUHI, A.; DING, M.; HASSAN, M. DroneCells: Improving spectral e ciency using drone-mounted ying base stations. Journal of Network and Computer Applications, Elsevier BV, v. 174, p. 102895, jan. 2021. Disponível em: <https://doi.org/10.1016/j.jnca.2020.102895>. Citado na página 20.

FUKUSHIMA, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition una ected by shift in position. Biological Cybernetics, Springer Science and Business Media LLC, v. 36, n. 4, p. 193 202, abr. 1980. Disponível em: <https://doi.org/10.1007/bf00344251>. Citado na página 24.

GAO, B.; PAVEL, L. On the properties of the softmax function with application in game theory and reinforcement learning. 04 2017. Citado na página 40.

GLOBE Drone Package Delivery Market to Hit USD 7,388.2 Million by 2027. <https://www.globenewswire.com/news-release/2020/11/30/2136699/0/en/Drone-Package-Delivery-Market-to-Hit-USD-7-388-2-Million-by-2027-Diverse-Entities-Such-as-Amaz html>. Acessado em 18/02/2021. Citado na página 20.

GLOBE Global Urban Air Mobility Market Study 2020. <https://www.globenewswire.com/news-release/2021/01/05/2153622/0/en/Global-Urban-Air-Mobility-Market-Study-2020-Market-Size-is-Expected-to-Reach-7-9-Billion-by-2030 html>. Acessado em 18/02/2021. Citado na página 20.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. Deep Learning [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado 7 vezes nas páginas 28, 36, 40, 42, 44, 45 e 46.

GUPTA, A. et al. Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues. Array, Elsevier BV, v. 10, p. 100057, jul. 2021. Disponível em: <https://doi.org/10.1016/j.array.2021.100057>. Citado na página 20.

HAYKIN, S. Redes Neurais: Princípios e Prática. [S.l.]: Bookman, 2003. ASIN B0006AXUII. Citado 4 vezes nas páginas 24, 25, 26 e 27.

HE, K. et al. Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2016. Disponível em: <https://doi.org/10.1109/cvpr.2016.90>. Citado na página 24.

HE, K. et al. Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2016. Disponível em: <https://doi.org/10.1109/cvpr.2016.90>. Citado na página 29.

IVAKHNENKO, A. G. The group method of data handling a rival of the method of stochastic approximation. Soviet Automatic Control, v. 1, n. 3, p. 43 55, 1968. Citado na página 23.

JAIN, H.; KHUNTETA, A.; SRIVASTAVA, S. Churn prediction in telecommunication using logistic regression and logit boost. Procedia Computer Science, Elsevier BV, v. 167, p. 101 112, 2020. Disponível em: <https://doi.org/10.1016/j.procs.2020.03.187>. Citado na página 19.

KANG, X.; SONG, B.; SUN, F. A deep similarity metric method based on incomplete data for tra c anomaly detection in IoT. Applied Sciences, MDPI AG, v. 9, n. 1, p. 135, jan. 2019. Disponível em: <https://doi.org/10.3390/app9010135>. Citado 2 vezes nas páginas 28 e 29.

KARANAM, S. R.; SRINIVAS, Y.; KRISHNA, M. V. Study on image processing using deep learning techniques. Materials Today: Proceedings, Elsevier BV, out. 2020. Disponível em: <https://doi.org/10.1016/j.matpr.2020.09.536>. Citado na página 19.

KELLERMANN, R.; BIEHLE, T.; FISCHER, L. Drones for parcel and passenger transportation: A literature review. Transportation Research Interdisciplinary Perspectives, Elsevier BV, v. 4, p. 100088, mar. 2020. Disponível em: <https://doi.org/10.1016/j.trip.2019.100088>. Citado na página 20.

KHACEF, L.; NASSIM, A.; MIRAMOND, B. Confronting machine-learning with neuroscience for neuromorphic architectures design. In: . [S.l.: s.n.], 2018. Citado na página 50.

KHANDANI, A. E.; KIM, A. J.; LO, A. W. Consumer credit-risk models via machine-learning algorithms. Journal of Banking & Finance, Elsevier BV, v. 34, n. 11, p. 2767 2787, nov. 2010. Disponível em: <https://doi.org/10.1016/j.jbank n.2010.06.001>. Citado na página 19.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. Imagenet classi cation with deep convolutional neural networks. In: . [S.l.: s.n.], 2012. p. 1097 1105. Citado na página 24.

KRUPPA, J. et al. Consumer credit risk: Individual probability estimates using machine learning. Expert Systems with Applications, Elsevier BV, v. 40, n. 13, p. 5125 5131, out. 2013. Disponível em: <https://doi.org/10.1016/j.eswa.2013.03.019>. Citado na página 19.

KUMAR, G. S. et al. Autonomous underwater vehicle for vision based tracking. Procedia Computer Science, Elsevier BV, v. 133, p. 169 180, 2018. Disponível em: <https://doi.org/10.1016/j.procs.2018.07.021>. Citado na página 20.

KURENKOV, A. A brief history of neural nets and deep learning. Skynet Today 2020. Citado 2 vezes nas páginas 25 e 26.

LE, W. T. et al. Overview of machine learning: Part 2. Neuroimaging Clinics of North America, Elsevier BV, v. 30, n. 4, p. 417 431, nov. 2020. Disponível em: <https://doi.org/10.1016/j.nic.2020.06.003>. Citado na página 19.

LECUN, Y. et al. Handwritten digit recognition with a back-propagation network. In: TOURETZKY, D. (Ed.). Advances in Neural Information Processing Systems Morgan-Kaufmann, 1990. v. 2. Disponível em: <https://proceedings.neurips.cc/paper/1989/ le/53c3bce66e43be4f209556518c2fcb54-Paper.pdf>. Citado na página 24.

LECUN, Y. et al. Gradient-based learning applied to document recognition. Proceedings of the IEEE, v. 86, n. 11, p. 2278 2324, 1998. Citado na página 24.

LECUN, Y. et al. Gradient-based learning applied to document recognition. Proceedings of the IEEE, Institute of Electrical and Electronics Engineers (IEEE), v. 86, n. 11, p. 2278 2324, 1998. Disponível em: <https://doi.org/10.1109/5.726791>. Citado na página 28.

LI, S. et al. Autonomous drone race: A computationally e cient vision-based navigation and control strategy. Robotics and Autonomous Systems, Elsevier BV, v. 133, p. 103621, nov. 2020. Disponível em: <https://doi.org/10.1016/j.robot.2020.103621>. Citado na página 20.

LINNAINMAA, S. Taylor expansion of the accumulated rounding error. BIT , Springer Science and Business Media LLC, v. 16, n. 2, p. 146 160, jun. 1976. Disponível em: <https://doi.org/10.1007/bf01931367>. Citado na página 23.

MA, X. et al. Deep learning method for makeup style transfer: A survey. Cognitive Robotics, Elsevier BV, v. 1, p. 182 187, 2021. Disponível em: <https://doi.org/10.1016/j.cogr.2021.09.001>. Citado na página 57.

MADIAJAGAN, M.; RAJ, S. S. Parallel computing, graphics processing unit (GPU) and new hardware for deep learning in computational intelligence research. In: Deep Learning and Parallel Computing Environment for Bioengineering Systems. Elsevier, 2019. p. 1 15. Disponível em: <https://doi.org/10.1016/b978-0-12-816718-2.00008-7>. Citado na página 56.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics, Springer Science and Business Media LLC, v. 5, n. 4, p. 115 133, dez. 1943. Disponível em: <https://doi.org/10.1007/bf02478259>. Citado na página 23.

NASR, G.; BADR, E.; JOUN, C. Cross entropy error function in neural networks: Forecasting gasoline demand. In: . [S.l.: s.n.], 2002. p. 381 384. Citado na página 45.

NGUYEN, A.-D. et al. Distribution padding in convolutional neural networks. In: 2019 IEEE International Conference on Image Processing (ICIP). IEEE, 2019. Disponível em: <https://doi.org/10.1109/icip.2019.8803537>. Citado na página 30.

REDMON, J. et al. You only look once: Uni ed, real-time object detection. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2016. Disponível em: <https://doi.org/10.1109/cvpr.2016.91>. Citado na página 34.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, American Psychological Association (APA), v. 65, n. 6, p. 386 408, 1958. Disponível em: <https://doi.org/10.1037/h0042519>. Citado na página 23.

ROSENBLATT, F. Principles of neurodynamics: perceptrons and the theory of brain mechanisms. [S.l.]: Spartan Books, 1962. ASIN B0006AXUII. Citado na página 23.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. Nature, Springer Science and Business Media LLC, v. 323, n. 6088, p. 533 536, out. 1986. Disponível em: <https://doi.org/10.1038/323533a0>. Citado na página 23.

SEPPO, L. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Dissertação (Mestrado) University of Helsinki, 1970. Citado na página 23.

SIMONYAN, K.; ZISSERMAN, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. 2015. Citado na página 24.

SUN, M. et al. Learning pooling for convolutional neural network. Neurocomputing, Elsevier BV, v. 224, p. 96 104, fev. 2017. Disponível em: <https://doi.org/10.1016/j.neucom.2016.10.049>. Citado na página 30.

SZEGEDY, C. et al. Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) [S.l.: s.n.], 2015. p. 1 9. Citado na página 24.

TIAN, Y.; FU, S. A descriptive framework for the field of deep learning applications in medical images. Knowledge-Based Systems, Elsevier BV, v. 210, p. 106445, dez. 2020. Disponível em: <https://doi.org/10.1016/j.knosys.2020.106445>. Citado na página 19.

Upulie Handalage; Lakshini Kuganandamurthy. Real-time object detection using yolo: A review. Unpublished, 2021. Disponível em: <https://rgdoi.net/10.13140/RG.2.2.24367.66723>. Citado na página 58.

VAFEIADIS, T. et al. A comparison of machine learning techniques for customer churn prediction. Simulation Modelling Practice and Theory, Elsevier BV, v. 55, p. 1 9, jun. 2015. Disponível em: <https://doi.org/10.1016/j.simpat.2015.03.003>. Citado na página 19.

YAMASHITA, R. et al. Convolutional neural networks: an overview and application in radiology. Insights into Imaging, Springer Science and Business Media LLC, v. 9, n. 4, p. 611 629, jun. 2018. Disponível em: <https://doi.org/10.1007/s13244-018-0639-9>. Citado 4 vezes nas páginas 28, 29, 30 e 31.

YIN, W.; LI, L.; WU, F.-X. Deep learning for brain disorder diagnosis based on fMRI images. Neurocomputing, Elsevier BV, out. 2020. Disponível em: <https://doi.org/10.1016/j.neucom.2020.05.113>. Citado na página 19.

YOLOV5. <https://github.com/ultralytics/yolov5>. Acessado em 20/04/2022. Citado na página 35.

ZEILER, M. D.; FERGUS, R. Visualizing and understanding convolutional networks. In: Computer Vision ECCV 2014. Springer International Publishing, 2014. p. 818 833. Disponível em: <https://doi.org/10.1007/978-3-319-10590-1_53>. Citado na página 24.

ZHAN, Z.; LI, H. A novel approach based on the elastoplastic fatigue damage and machine learning models for life prediction of aerospace alloy parts fabricated by additive manufacturing. International Journal of Fatigue, Elsevier BV, v. 145, p. 106089, abr. 2021. Disponível em: <https://doi.org/10.1016/j.ijfatigue.2020.106089>. Citado na página 19.

ZHANG, J.-R. et al. A hybrid particle swarm optimization back-propagation algorithm for feedforward neural network training. Applied Mathematics and Computation, Elsevier BV, v. 185, n. 2, p. 1026 1037, fev. 2007. Disponível em: <https://doi.org/10.1016/j.amc.2006.07.025>. Citado 2 vezes nas páginas 48 e 49.

ZHANG, Y. et al. Deep learning for imaging and detection of microorganisms. Trends in Microbiology, Elsevier BV, jan. 2021. Disponível em: <https://doi.org/10.1016/j.tim.2021.01.006>. Citado na página 19.

Apêndices

APÊNDICE A CODIGOS

```
import argparse
import math
import os
import random
import sys
import time
from copy import deepcopy
from datetime import datetime
from pathlib import Path

import numpy as np
import torch
import torch.distributed as dist
import torch.nn as nn
import yaml
from torch.cuda import amp
from torch.nn.parallel import DistributedDataParallel as DDP
from torch.optim import SGD, Adam, AdamW, lr_scheduler
from tqdm import tqdm

FILE = Path(__file__).resolve()
ROOT = FILE.parents[0] # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT)) # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd())) # relative

import val # for end-of-epoch mAP
from models.experimental import attempt_load
from models.yolo import Model
from utils.autoanchor import check_anchors
from utils.autobatch import check_train_batch_size
from utils.callbacks import Callbacks
from utils.datasets import create_dataloader
from utils.downloads import attempt_download
from utils.general import (LOGGER, check_dataset, check_file,
    check_git_status, check_img_size, check_requirements,
```

```
                                check_suffix, check_yaml, colorstr, get_latest_run,
                                    increment_path, init_seeds,
                                intersect_dicts, labels_to_class_weights,
                                    labels_to_image_weights, methods, one_cycle,
                                print_args, print_mutation, strip_optimizer)
from utils.loggers import Loggers
from utils.loggers.wandb.wandb_utils import check_wandb_resume
from utils.loss import ComputeLoss
from utils.metrics import fitness
from utils.plots import plot_evolve, plot_labels
from utils.torch_utils import EarlyStopping, ModelEMA, de_parallel,
    select_device, torch_distributed_zero_first

LOCAL_RANK = int(os.getenv('LOCAL_RANK', -1)) #
    https://pytorch.org/docs/stable/elastic/run.html
RANK = int(os.getenv('RANK', -1))
WORLD_SIZE = int(os.getenv('WORLD_SIZE', 1))


def train(hyp, opt, device, callbacks): # hyp is path/to/hyp.yaml or hyp
    dictionary
    save_dir, epochs, batch_size, weights, single_cls, evolve, data, cfg,
        resume, noval, nosave, workers, freeze = \
        Path(opt.save_dir), opt.epochs, opt.batch_size, opt.weights,
            opt.single_cls, opt.evolve, opt.data, opt.cfg, \
        opt.resume, opt.noval, opt.nosave, opt.workers, opt.freeze

    # Directories
    w = save_dir / 'weights' # weights dir
    (w.parent if evolve else w).mkdir(parents=True, exist_ok=True) # make dir
    last, best = w / 'last.pt', w / 'best.pt'

    # Hyperparameters
    if isinstance(hyp, str):
        with open(hyp, errors='ignore') as f:
            hyp = yaml.safe_load(f) # load hyps dict
    LOGGER.info(colorstr('hyperparameters: ') + ', '.join(f'{k}={v}' for k, v
        in hyp.items()))

    # Save run settings
    if not evolve:
        with open(save_dir / 'hyp.yaml', 'w') as f:
```

```
            yaml.safe_dump(hyp, f, sort_keys=False)
        with open(save_dir / 'opt.yaml', 'w') as f:
            yaml.safe_dump(vars(opt), f, sort_keys=False)

    # Loggers
    data_dict = None
    if RANK in [-1, 0]:
        loggers = Loggers(save_dir, weights, opt, hyp, LOGGER) # loggers
            instance
        if loggers.wandb:
            data_dict = loggers.wandb.data_dict
            if resume:
                weights, epochs, hyp, batch_size = opt.weights, opt.epochs,
                    opt.hyp, opt.batch_size

        # Register actions
        for k in methods(loggers):
            callbacks.register_action(k, callback=getattr(loggers, k))

    # Config
    plots = not evolve # create plots
    cuda = device.type != 'cpu'
    init_seeds(1 + RANK)
    with torch_distributed_zero_first(LOCAL_RANK):
        data_dict = data_dict or check_dataset(data) # check if None
    train_path, val_path = data_dict['train'], data_dict['val']
    nc = 1 if single_cls else int(data_dict['nc']) # number of classes
    names = ['item'] if single_cls and len(data_dict['names']) != 1 else
        data_dict['names'] # class names
    assert len(names) == nc, f'{len(names)} names found for nc={nc} dataset in
        {data}' # check
    is_coco = isinstance(val_path, str) and
        val_path.endswith('coco/val2017.txt') # COCO dataset

    # Model
    check_suffix(weights, '.pt') # check weights
    pretrained = weights.endswith('.pt')
    if pretrained:
        with torch_distributed_zero_first(LOCAL_RANK):
            weights = attempt_download(weights) # download if not found locally
        ckpt = torch.load(weights, map_location='cpu') # load checkpoint to
            CPU to avoid CUDA memory leak
```

```
        model = Model(cfg or ckpt['model'].yaml, ch=3, nc=nc,
            anchors=hyp.get('anchors')).to(device) # create
        exclude = ['anchor'] if (cfg or hyp.get('anchors')) and not resume
            else [] # exclude keys
        csd = ckpt['model'].float().state_dict() # checkpoint state_dict as
            FP32
        csd = intersect_dicts(csd, model.state_dict(), exclude=exclude) #
            intersect
        model.load_state_dict(csd, strict=False) # load
        LOGGER.info(f'Transferred {len(csd)}/{len(model.state_dict())} items
            from {weights}') # report
    else:
        model = Model(cfg, ch=3, nc=nc, anchors=hyp.get('anchors')).to(device)
            # create

    # Freeze
    freeze = [f'model.{x}.' for x in (freeze if len(freeze) > 1 else
        range(freeze[0]))] # layers to freeze
    for k, v in model.named_parameters():
        v.requires_grad = True # train all layers
        if any(x in k for x in freeze):
            LOGGER.info(f'freezing {k}')
            v.requires_grad = False

    # Image size
    gs = max(int(model.stride.max()), 32) # grid size (max stride)
    imgsz = check_img_size(opt.imgsz, gs, floor=gs * 2) # verify imgsz is
        gs-multiple

    # Batch size
    if RANK == -1 and batch_size == -1: # single-GPU only, estimate best batch
        size
        batch_size = check_train_batch_size(model, imgsz)
        loggers.on_params_update({"batch_size": batch_size})

    # Optimizer
    nbs = 64 # nominal batch size
    accumulate = max(round(nbs / batch_size), 1) # accumulate loss before
        optimizing
    hyp['weight_decay'] *= batch_size * accumulate / nbs # scale weight_decay
    LOGGER.info(f"Scaled weight_decay = {hyp['weight_decay']}")
```

```
g0, g1, g2 = [], [], [] # optimizer parameter groups
for v in model.modules():
    if hasattr(v, 'bias') and isinstance(v.bias, nn.Parameter): # bias
        g2.append(v.bias)
    if isinstance(v, nn.BatchNorm2d): # weight (no decay)
        g0.append(v.weight)
    elif hasattr(v, 'weight') and isinstance(v.weight, nn.Parameter): #
        weight (with decay)
        g1.append(v.weight)

if opt.optimizer == 'Adam':
    optimizer = Adam(g0, lr=hyp['lr0'], betas=(hyp['momentum'], 0.999)) #
        adjust beta1 to momentum
elif opt.optimizer == 'AdamW':
    optimizer = AdamW(g0, lr=hyp['lr0'], betas=(hyp['momentum'], 0.999)) #
        adjust beta1 to momentum
else:
    optimizer = SGD(g0, lr=hyp['lr0'], momentum=hyp['momentum'],
        nesterov=True)

optimizer.add_param_group({'params': g1, 'weight_decay':
    hyp['weight_decay']}) # add g1 with weight_decay
optimizer.add_param_group({'params': g2}) # add g2 (biases)
LOGGER.info(f"{colorstr('optimizer:')} {type(optimizer).__name__} with
    parameter groups "
            f"{len(g0)} weight (no decay), {len(g1)} weight, {len(g2)}
                bias")
del g0, g1, g2

# Scheduler
if opt.cos_lr:
    lf = one_cycle(1, hyp['lrf'], epochs) # cosine 1->hyp['lrf']
else:
    lf = lambda x: (1 - x / epochs) * (1.0 - hyp['lrf']) + hyp['lrf'] #
        linear
scheduler = lr_scheduler.LambdaLR(optimizer, lr_lambda=lf) #
    plot_lr_scheduler(optimizer, scheduler, epochs)

# EMA
ema = ModelEMA(model) if RANK in [-1, 0] else None

# Resume
```

```
start_epoch, best_fitness = 0, 0.0
if pretrained:
    # Optimizer
    if ckpt['optimizer'] is not None:
        optimizer.load_state_dict(ckpt['optimizer'])
        best_fitness = ckpt['best_fitness']

    # EMA
    if ema and ckpt.get('ema'):
        ema.ema.load_state_dict(ckpt['ema'].float().state_dict())
        ema.updates = ckpt['updates']

    # Epochs
    start_epoch = ckpt['epoch'] + 1
    if resume:
        assert start_epoch > 0, f'{weights} training to {epochs} epochs is
            finished, nothing to resume.'
    if epochs < start_epoch:
        LOGGER.info(f"{weights} has been trained for {ckpt['epoch']}
            epochs. Fine-tuning for {epochs} more epochs.")
        epochs += ckpt['epoch'] # finetune additional epochs

    del ckpt, csd

# DP mode
if cuda and RANK == -1 and torch.cuda.device_count() > 1:
    LOGGER.warning('WARNING: DP not recommended, use torch.distributed.run
        for best DDP Multi-GPU results.\n'
                   'See Multi-GPU Tutorial at
                       https://github.com/ultralytics/yolov5/issues/475 to
                       get started.')
    model = torch.nn.DataParallel(model)

# SyncBatchNorm
if opt.sync_bn and cuda and RANK != -1:
    model = torch.nn.SyncBatchNorm.convert_sync_batchnorm(model).to(device)
    LOGGER.info('Using SyncBatchNorm()')

# Trainloader
train_loader, dataset = create_dataloader(train_path,
                                          imgsz,
                                          batch_size // WORLD_SIZE,
```

```
                                        gs,
                                        single_cls,
                                        hyp=hyp,
                                        augment=True,
                                        cache=None if opt.cache == 'val'
                                            else opt.cache,
                                        rect=opt.rect,
                                        rank=LOCAL_RANK,
                                        workers=workers,
                                        image_weights=opt.image_weights,
                                        quad=opt.quad,
                                        prefix=colorstr('train: '),
                                        shuffle=True)
    mlc = int(np.concatenate(dataset.labels, 0)[:, 0].max()) # max label class
    nb = len(train_loader) # number of batches
    assert mlc < nc, f'Label class {mlc} exceeds nc={nc} in {data}. Possible
        class labels are 0-{nc - 1}'

    # Process 0
    if RANK in [-1, 0]:
        val_loader = create_dataloader(val_path,
                                       imgsz,
                                       batch_size // WORLD_SIZE * 2,
                                       gs,
                                       single_cls,
                                       hyp=hyp,
                                       cache=None if noval else opt.cache,
                                       rect=True,
                                       rank=-1,
                                       workers=workers * 2,
                                       pad=0.5,
                                       prefix=colorstr('val: '))[0]

        if not resume:
            labels = np.concatenate(dataset.labels, 0)
            # c = torch.tensor(labels[:, 0]) # classes
            # cf = torch.bincount(c.long(), minlength=nc) + 1. # frequency
            # model._initialize_biases(cf.to(device))
            if plots:
                plot_labels(labels, names, save_dir)

            # Anchors
```

```
            if not opt.noautoanchor:
                check_anchors(dataset, model=model, thr=hyp['anchor_t'],
                    imgsz=imgsz)
            model.half().float() # pre-reduce anchor precision

        callbacks.run('on_pretrain_routine_end')

    # DDP mode
    if cuda and RANK != -1:
        model = DDP(model, device_ids=[LOCAL_RANK], output_device=LOCAL_RANK)

    # Model attributes
    nl = de_parallel(model).model[-1].nl # number of detection layers (to
        scale hyps)
    hyp['box'] *= 3 / nl # scale to layers
    hyp['cls'] *= nc / 80 * 3 / nl # scale to classes and layers
    hyp['obj'] *= (imgsz / 640) ** 2 * 3 / nl # scale to image size and layers
    hyp['label_smoothing'] = opt.label_smoothing
    model.nc = nc # attach number of classes to model
    model.hyp = hyp # attach hyperparameters to model
    model.class_weights = labels_to_class_weights(dataset.labels,
        nc).to(device) * nc # attach class weights
    model.names = names

    # Start training
    t0 = time.time()
    nw = max(round(hyp['warmup_epochs'] * nb), 100) # number of warmup
        iterations, max(3 epochs, 100 iterations)
    # nw = min(nw, (epochs - start_epoch) / 2 * nb) # limit warmup to < 1/2 of
        training
    last_opt_step = -1
    maps = np.zeros(nc) # mAP per class
    results = (0, 0, 0, 0, 0, 0, 0) # P, R, mAP@.5, mAP@.5-.95, val_loss(box,
        obj, cls)
    scheduler.last_epoch = start_epoch - 1 # do not move
    scaler = amp.GradScaler(enabled=cuda)
    stopper = EarlyStopping(patience=opt.patience)
    compute_loss = ComputeLoss(model) # init loss class
    LOGGER.info(f'Image sizes {imgsz} train, {imgsz} val\n'
                f'Using {train_loader.num_workers * WORLD_SIZE} dataloader
                    workers\n'
                f"Logging results to {colorstr('bold', save_dir)}\n"
```

```
            f'Starting training for {epochs} epochs...')
    for epoch in range(start_epoch, epochs): # epoch
        ------------------------------------------------------------------
        model.train()

        # Update image weights (optional, single-GPU only)
        if opt.image_weights:
            cw = model.class_weights.cpu().numpy() * (1 - maps) ** 2 / nc #
                class weights
            iw = labels_to_image_weights(dataset.labels, nc=nc,
                class_weights=cw) # image weights
            dataset.indices = random.choices(range(dataset.n), weights=iw,
                k=dataset.n) # rand weighted idx

        # Update mosaic border (optional)
        # b = int(random.uniform(0.25 * imgsz, 0.75 * imgsz + gs) // gs * gs)
        # dataset.mosaic_border = [b - imgsz, -b] # height, width borders

        mloss = torch.zeros(3, device=device) # mean losses
        if RANK != -1:
            train_loader.sampler.set_epoch(epoch)
        pbar = enumerate(train_loader)
        LOGGER.info(('\n' + '%10s' * 7) % ('Epoch', 'gpu_mem', 'box', 'obj',
            'cls', 'labels', 'img_size'))
        if RANK in [-1, 0]:
            pbar = tqdm(pbar, total=nb,
                bar_format='{l_bar}{bar:10}{r_bar}{bar:-10b}') # progress bar
        optimizer.zero_grad()
        for i, (imgs, targets, paths, _) in pbar: # batch
            ------------------------------------------------------------
            ni = i + nb * epoch # number integrated batches (since train start)
            imgs = imgs.to(device, non_blocking=True).float() / 255 # uint8 to
                float32, 0-255 to 0.0-1.0

            # Warmup
            if ni <= nw:
                xi = [0, nw] # x interp
                # compute_loss.gr = np.interp(ni, xi, [0.0, 1.0]) # iou loss
                    ratio (obj_loss = 1.0 or iou)
                accumulate = max(1, np.interp(ni, xi, [1, nbs /
                    batch_size]).round())
                for j, x in enumerate(optimizer.param_groups):
```

```
            # bias lr falls from 0.1 to lr0, all other lrs rise from 0.0
                to lr0
            x['lr'] = np.interp(ni, xi, [hyp['warmup_bias_lr'] if j == 2
                else 0.0, x['initial_lr'] * lf(epoch)])
            if 'momentum' in x:
                x['momentum'] = np.interp(ni, xi,
                    [hyp['warmup_momentum'], hyp['momentum']])

    # Multi-scale
    if opt.multi_scale:
        sz = random.randrange(imgsz * 0.5, imgsz * 1.5 + gs) // gs * gs
            # size
        sf = sz / max(imgs.shape[2:]) # scale factor
        if sf != 1:
            ns = [math.ceil(x * sf / gs) * gs for x in imgs.shape[2:]] #
                new shape (stretched to gs-multiple)
            imgs = nn.functional.interpolate(imgs, size=ns,
                mode='bilinear', align_corners=False)

    # Forward
    with amp.autocast(enabled=cuda):
        pred = model(imgs) # forward
        loss, loss_items = compute_loss(pred, targets.to(device)) #
            loss scaled by batch_size
        if RANK != -1:
            loss *= WORLD_SIZE # gradient averaged between devices in
                DDP mode
        if opt.quad:
            loss *= 4.

    # Backward
    scaler.scale(loss).backward()

    # Optimize
    if ni - last_opt_step >= accumulate:
        scaler.step(optimizer) # optimizer.step
        scaler.update()
        optimizer.zero_grad()
        if ema:
            ema.update(model)
        last_opt_step = ni
```

```
            # Log
            if RANK in [-1, 0]:
                mloss = (mloss * i + loss_items) / (i + 1) # update mean losses
                mem = f'{torch.cuda.memory_reserved() / 1E9 if
                    torch.cuda.is_available() else 0:.3g}G' # (GB)
                pbar.set_description(('%10s' * 2 + '%10.4g' * 5) %
                                     (f'{epoch}/{epochs - 1}', mem, *mloss,
                                         targets.shape[0], imgs.shape[-1]))
                callbacks.run('on_train_batch_end', ni, model, imgs, targets,
                    paths, plots, opt.sync_bn)
                if callbacks.stop_training:
                    return
            # end batch
                ----------------------------------------------------------------------------------------------

        # Scheduler
        lr = [x['lr'] for x in optimizer.param_groups] # for loggers
        scheduler.step()

        if RANK in [-1, 0]:
            # mAP
            callbacks.run('on_train_epoch_end', epoch=epoch)
            ema.update_attr(model, include=['yaml', 'nc', 'hyp', 'names',
                'stride', 'class_weights'])
            final_epoch = (epoch + 1 == epochs) or stopper.possible_stop
            if not noval or final_epoch: # Calculate mAP
                results, maps, _ = val.run(data_dict,
                                           batch_size=batch_size // WORLD_SIZE * 2,
                                           imgsz=imgsz,
                                           model=ema.ema,
                                           single_cls=single_cls,
                                           dataloader=val_loader,
                                           save_dir=save_dir,
                                           plots=False,
                                           callbacks=callbacks,
                                           compute_loss=compute_loss)

            # Update best mAP
            fi = fitness(np.array(results).reshape(1, -1)) # weighted
                combination of [P, R, mAP@.5, mAP@.5-.95]
            if fi > best_fitness:
                best_fitness = fi
```

```
            log_vals = list(mloss) + list(results) + lr
            callbacks.run('on_fit_epoch_end', log_vals, epoch, best_fitness, fi)

            # Save model
            if (not nosave) or (final_epoch and not evolve): # if save
                ckpt = {
                    'epoch': epoch,
                    'best_fitness': best_fitness,
                    'model': deepcopy(de_parallel(model)).half(),
                    'ema': deepcopy(ema.ema).half(),
                    'updates': ema.updates,
                    'optimizer': optimizer.state_dict(),
                    'wandb_id': loggers.wandb.wandb_run.id if loggers.wandb else
                        None,
                    'date': datetime.now().isoformat()}

                # Save last, best and delete
                torch.save(ckpt, last)
                if best_fitness == fi:
                    torch.save(ckpt, best)
                if (epoch > 0) and (opt.save_period > 0) and (epoch %
                    opt.save_period == 0):
                    torch.save(ckpt, w / f'epoch{epoch}.pt')
                del ckpt
                callbacks.run('on_model_save', last, epoch, final_epoch,
                    best_fitness, fi)

            # Stop Single-GPU
            if RANK == -1 and stopper(epoch=epoch, fitness=fi):
                break

            # Stop DDP TODO: known issues
                shttps://github.com/ultralytics/yolov5/pull/4576
            # stop = stopper(epoch=epoch, fitness=fi)
            # if RANK == 0:
            #     dist.broadcast_object_list([stop], 0) # broadcast 'stop' to
                all ranks

        # Stop DPP
        # with torch_distributed_zero_first(RANK):
        # if stop:
        #     break # must break all DDP ranks
```

```
        # end epoch
            ----------------------------------------------------------------------------------------------------
    # end training
        ----------------------------------------------------------------------------------------------------
    if RANK in [-1, 0]:
        LOGGER.info(f'\n{epoch - start_epoch + 1} epochs completed in
            {(time.time() - t0) / 3600:.3f} hours.')
        for f in last, best:
            if f.exists():
                strip_optimizer(f) # strip optimizers
                if f is best:
                    LOGGER.info(f'\nValidating {f}...')
                    results, _, _ = val.run(
                        data_dict,
                        batch_size=batch_size // WORLD_SIZE * 2,
                        imgsz=imgsz,
                        model=attempt_load(f, device).half(),
                        iou_thres=0.65 if is_coco else 0.60, # best pycocotools
                            results at 0.65
                        single_cls=single_cls,
                        dataloader=val_loader,
                        save_dir=save_dir,
                        save_json=is_coco,
                        verbose=True,
                        plots=True,
                        callbacks=callbacks,
                        compute_loss=compute_loss) # val best model with plots
                    if is_coco:
                        callbacks.run('on_fit_epoch_end', list(mloss) +
                            list(results) + lr, epoch, best_fitness, fi)

        callbacks.run('on_train_end', last, best, plots, epoch, results)
        LOGGER.info(f"Results saved to {colorstr('bold', save_dir)}")

    torch.cuda.empty_cache()
    return results


def parse_opt(known=False):
    parser = argparse.ArgumentParser()
```

```
parser.add_argument('--weights', type=str, default=ROOT / 'yolov5s.pt',
    help='initial weights path')
parser.add_argument('--cfg', type=str, default='', help='model.yaml path')
parser.add_argument('--data', type=str, default=ROOT /
    'data/coco128.yaml', help='dataset.yaml path')
parser.add_argument('--hyp', type=str, default=ROOT /
    'data/hyps/hyp.scratch-low.yaml', help='hyperparameters path')
parser.add_argument('--epochs', type=int, default=300)
parser.add_argument('--batch-size', type=int, default=16, help='total
    batch size for all GPUs, -1 for autobatch')
parser.add_argument('--imgsz', '--img', '--img-size', type=int,
    default=640, help='train, val image size (pixels)')
parser.add_argument('--rect', action='store_true', help='rectangular
    training')
parser.add_argument('--resume', nargs='?', const=True, default=False,
    help='resume most recent training')
parser.add_argument('--nosave', action='store_true', help='only save final
    checkpoint')
parser.add_argument('--noval', action='store_true', help='only validate
    final epoch')
parser.add_argument('--noautoanchor', action='store_true', help='disable
    AutoAnchor')
parser.add_argument('--evolve', type=int, nargs='?', const=300,
    help='evolve hyperparameters for x generations')
parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
parser.add_argument('--cache', type=str, nargs='?', const='ram',
    help='--cache images in "ram" (default) or "disk"')
parser.add_argument('--image-weights', action='store_true', help='use
    weighted image selection for training')
parser.add_argument('--device', default='', help='cuda device, i.e. 0 or
    0,1,2,3 or cpu')
parser.add_argument('--multi-scale', action='store_true', help='vary
    img-size +/- 50%%')
parser.add_argument('--single-cls', action='store_true', help='train
    multi-class data as single-class')
parser.add_argument('--optimizer', type=str, choices=['SGD', 'Adam',
    'AdamW'], default='SGD', help='optimizer')
parser.add_argument('--sync-bn', action='store_true', help='use
    SyncBatchNorm, only available in DDP mode')
parser.add_argument('--workers', type=int, default=8, help='max dataloader
    workers (per RANK in DDP mode)')
```

```
    parser.add_argument('--project', default=ROOT / 'runs/train', help='save
        to project/name')
    parser.add_argument('--name', default='exp', help='save to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing
        project/name ok, do not increment')
    parser.add_argument('--quad', action='store_true', help='quad dataloader')
    parser.add_argument('--cos-lr', action='store_true', help='cosine LR
        scheduler')
    parser.add_argument('--label-smoothing', type=float, default=0.0,
        help='Label smoothing epsilon')
    parser.add_argument('--patience', type=int, default=100,
        help='EarlyStopping patience (epochs without improvement)')
    parser.add_argument('--freeze', nargs='+', type=int, default=[0],
        help='Freeze layers: backbone=10, first3=0 1 2')
    parser.add_argument('--save-period', type=int, default=-1, help='Save
        checkpoint every x epochs (disabled if < 1)')
    parser.add_argument('--local_rank', type=int, default=-1, help='DDP
        parameter, do not modify')

    # Weights & Biases arguments
    parser.add_argument('--entity', default=None, help='W&B: Entity')
    parser.add_argument('--upload_dataset', nargs='?', const=True,
        default=False, help='W&B: Upload data, "val" option')
    parser.add_argument('--bbox_interval', type=int, default=-1, help='W&B:
        Set bounding-box image logging interval')
    parser.add_argument('--artifact_alias', type=str, default='latest',
        help='W&B: Version of dataset artifact to use')

    opt = parser.parse_known_args()[0] if known else parser.parse_args()
    return opt


def main(opt, callbacks=Callbacks()):
    # Checks
    if RANK in [-1, 0]:
        print_args(vars(opt))
        check_git_status()
        check_requirements(exclude=['thop'])

    # Resume
    if opt.resume and not check_wandb_resume(opt) and not opt.evolve: # resume
        an interrupted run
```

```
        ckpt = opt.resume if isinstance(opt.resume, str) else get_latest_run()
            # specified or most recent path
        assert os.path.isfile(ckpt), 'ERROR: --resume checkpoint does not
            exist'
        with open(Path(ckpt).parent.parent / 'opt.yaml', errors='ignore') as f:
            opt = argparse.Namespace(**yaml.safe_load(f)) # replace
        opt.cfg, opt.weights, opt.resume = '', ckpt, True # reinstate
        LOGGER.info(f'Resuming training from {ckpt}')
    else:
        opt.data, opt.cfg, opt.hyp, opt.weights, opt.project = \
            check_file(opt.data), check_yaml(opt.cfg), check_yaml(opt.hyp),
                str(opt.weights), str(opt.project) # checks
        assert len(opt.cfg) or len(opt.weights), 'either --cfg or --weights
            must be specified'
        if opt.evolve:
            if opt.project == str(ROOT / 'runs/train'): # if default project
                name, rename to runs/evolve
                opt.project = str(ROOT / 'runs/evolve')
            opt.exist_ok, opt.resume = opt.resume, False # pass resume to
                exist_ok and disable resume
        if opt.name == 'cfg':
            opt.name = Path(opt.cfg).stem # use model.yaml as name
        opt.save_dir = str(increment_path(Path(opt.project) / opt.name,
            exist_ok=opt.exist_ok))

    # DDP mode
    device = select_device(opt.device, batch_size=opt.batch_size)
    if LOCAL_RANK != -1:
        msg = 'is not compatible with YOLOv5 Multi-GPU DDP training'
        assert not opt.image_weights, f'--image-weights {msg}'
        assert not opt.evolve, f'--evolve {msg}'
        assert opt.batch_size != -1, f'AutoBatch with --batch-size -1 {msg},
            please pass a valid --batch-size'
        assert opt.batch_size % WORLD_SIZE == 0, f'--batch-size
            {opt.batch_size} must be multiple of WORLD_SIZE'
        assert torch.cuda.device_count() > LOCAL_RANK, 'insufficient CUDA
            devices for DDP command'
        torch.cuda.set_device(LOCAL_RANK)
        device = torch.device('cuda', LOCAL_RANK)
        dist.init_process_group(backend="nccl" if dist.is_nccl_available()
            else "gloo")
```

```
# Train
if not opt.evolve:
    train(opt.hyp, opt, device, callbacks)
    if WORLD_SIZE > 1 and RANK == 0:
        LOGGER.info('Destroying process group... ')
        dist.destroy_process_group()

# Evolve hyperparameters (optional)
else:
    # Hyperparameter evolution metadata (mutation scale 0-1, lower_limit,
        upper_limit)
    meta = {
        'lr0': (1, 1e-5, 1e-1), # initial learning rate (SGD=1E-2,
            Adam=1E-3)
        'lrf': (1, 0.01, 1.0), # final OneCycleLR learning rate (lr0 * lrf)
        'momentum': (0.3, 0.6, 0.98), # SGD momentum/Adam beta1
        'weight_decay': (1, 0.0, 0.001), # optimizer weight decay
        'warmup_epochs': (1, 0.0, 5.0), # warmup epochs (fractions ok)
        'warmup_momentum': (1, 0.0, 0.95), # warmup initial momentum
        'warmup_bias_lr': (1, 0.0, 0.2), # warmup initial bias lr
        'box': (1, 0.02, 0.2), # box loss gain
        'cls': (1, 0.2, 4.0), # cls loss gain
        'cls_pw': (1, 0.5, 2.0), # cls BCELoss positive_weight
        'obj': (1, 0.2, 4.0), # obj loss gain (scale with pixels)
        'obj_pw': (1, 0.5, 2.0), # obj BCELoss positive_weight
        'iou_t': (0, 0.1, 0.7), # IoU training threshold
        'anchor_t': (1, 2.0, 8.0), # anchor-multiple threshold
        'anchors': (2, 2.0, 10.0), # anchors per output grid (0 to ignore)
        'fl_gamma': (0, 0.0, 2.0), # focal loss gamma (efficientDet default
            gamma=1.5)
        'hsv_h': (1, 0.0, 0.1), # image HSV-Hue augmentation (fraction)
        'hsv_s': (1, 0.0, 0.9), # image HSV-Saturation augmentation
            (fraction)
        'hsv_v': (1, 0.0, 0.9), # image HSV-Value augmentation (fraction)
        'degrees': (1, 0.0, 45.0), # image rotation (+/- deg)
        'translate': (1, 0.0, 0.9), # image translation (+/- fraction)
        'scale': (1, 0.0, 0.9), # image scale (+/- gain)
        'shear': (1, 0.0, 10.0), # image shear (+/- deg)
        'perspective': (0, 0.0, 0.001), # image perspective (+/- fraction),
            range 0-0.001
        'flipud': (1, 0.0, 1.0), # image flip up-down (probability)
        'fliplr': (0, 0.0, 1.0), # image flip left-right (probability)
```

```
        'mosaic': (1, 0.0, 1.0), # image mixup (probability)
        'mixup': (1, 0.0, 1.0), # image mixup (probability)
        'copy_paste': (1, 0.0, 1.0)} # segment copy-paste (probability)

    with open(opt.hyp, errors='ignore') as f:
        hyp = yaml.safe_load(f) # load hyps dict
        if 'anchors' not in hyp: # anchors commented in hyp.yaml
            hyp['anchors'] = 3
    opt.noval, opt.nosave, save_dir = True, True, Path(opt.save_dir) #
        only val/save final epoch
    # ei = [isinstance(x, (int, float)) for x in hyp.values()] # evolvable
        indices
    evolve_yaml, evolve_csv = save_dir / 'hyp_evolve.yaml', save_dir /
        'evolve.csv'
    if opt.bucket:
        os.system(f'gsutil cp gs://{opt.bucket}/evolve.csv {evolve_csv}') #
            download evolve.csv if exists

    for _ in range(opt.evolve): # generations to evolve
        if evolve_csv.exists(): # if evolve.csv exists: select best hyps
            and mutate
            # Select parent(s)
            parent = 'single' # parent selection method: 'single' or
                'weighted'
            x = np.loadtxt(evolve_csv, ndmin=2, delimiter=',', skiprows=1)
            n = min(5, len(x)) # number of previous results to consider
            x = x[np.argsort(-fitness(x))][:n] # top n mutations
            w = fitness(x) - fitness(x).min() + 1E-6 # weights (sum > 0)
            if parent == 'single' or len(x) == 1:
                # x = x[random.randint(0, n - 1)] # random selection
                x = x[random.choices(range(n), weights=w)[0]] # weighted
                    selection
            elif parent == 'weighted':
                x = (x * w.reshape(n, 1)).sum(0) / w.sum() # weighted
                    combination

            # Mutate
            mp, s = 0.8, 0.2 # mutation probability, sigma
            npr = np.random
            npr.seed(int(time.time()))
            g = np.array([meta[k][0] for k in hyp.keys()]) # gains 0-1
            ng = len(meta)
```

```
                v = np.ones(ng)
                while all(v == 1): # mutate until a change occurs (prevent
                    duplicates)
                    v = (g * (npr.random(ng) < mp) * npr.randn(ng) *
                        npr.random() * s + 1).clip(0.3, 3.0)
                for i, k in enumerate(hyp.keys()): # plt.hist(v.ravel(), 300)
                    hyp[k] = float(x[i + 7] * v[i]) # mutate

            # Constrain to limits
            for k, v in meta.items():
                hyp[k] = max(hyp[k], v[1]) # lower limit
                hyp[k] = min(hyp[k], v[2]) # upper limit
                hyp[k] = round(hyp[k], 5) # significant digits

            # Train mutation
            results = train(hyp.copy(), opt, device, callbacks)
            callbacks = Callbacks()
            # Write mutation results
            print_mutation(results, hyp.copy(), save_dir, opt.bucket)

        # Plot results
        plot_evolve(evolve_csv)
        LOGGER.info(f'Hyperparameter evolution finished {opt.evolve}
            generations\n'
                    f"Results saved to {colorstr('bold', save_dir)}\n"
                    f'Usage example: $ python train.py --hyp {evolve_yaml}')


def run(**kwargs):
    # Usage: import train; train.run(data='coco128.yaml', imgsz=320,
        weights='yolov5m.pt')
    opt = parse_opt(True)
    for k, v in kwargs.items():
        setattr(opt, k, v)
    main(opt)
    return opt


if __name__ == "__main__":
    opt = parse_opt()
    main(opt)
```

```
import argparse
import json
import os
import sys
from pathlib import Path
from threading import Thread

import numpy as np
import torch
from tqdm import tqdm

FILE = Path(__file__).resolve()
ROOT = FILE.parents[0] # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT)) # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd())) # relative

from models.common import DetectMultiBackend
from utils.callbacks import Callbacks
from utils.datasets import create_dataloader
from utils.general import (LOGGER, box_iou, check_dataset, check_img_size,
    check_requirements, check_yaml,
                           coco80_to_coco91_class, colorstr, increment_path,
                               non_max_suppression, print_args,
                           scale_coords, xywh2xyxy, xyxy2xywh)
from utils.metrics import ConfusionMatrix, ap_per_class
from utils.plots import output_to_target, plot_images, plot_val_study
from utils.torch_utils import select_device, time_sync


def save_one_txt(predn, save_conf, shape, file):
    # Save one txt result
    gn = torch.tensor(shape)[[1, 0, 1, 0]] # normalization gain whwh
    for *xyxy, conf, cls in predn.tolist():
        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) /
            gn).view(-1).tolist() # normalized xywh
        line = (cls, *xywh, conf) if save_conf else (cls, *xywh) # label format
        with open(file, 'a') as f:
            f.write(('%g ' * len(line)).rstrip() % line + '\n')
```

```
def save_one_json(predn, jdict, path, class_map):
    # Save one JSON result {"image_id": 42, "category_id": 18, "bbox":
        [258.15, 41.29, 348.26, 243.78], "score": 0.236}
    image_id = int(path.stem) if path.stem.isnumeric() else path.stem
    box = xyxy2xywh(predn[:, :4]) # xywh
    box[:, :2] -= box[:, 2:] / 2 # xy center to top-left corner
    for p, b in zip(predn.tolist(), box.tolist()):
        jdict.append({
            'image_id': image_id,
            'category_id': class_map[int(p[5])],
            'bbox': [round(x, 3) for x in b],
            'score': round(p[4], 5)})


def process_batch(detections, labels, iouv):
    """
    Return correct predictions matrix. Both sets of boxes are in (x1, y1, x2,
        y2) format.
    Arguments:
        detections (Array[N, 6]), x1, y1, x2, y2, conf, class
        labels (Array[M, 5]), class, x1, y1, x2, y2
    Returns:
        correct (Array[N, 10]), for 10 IoU levels
    """
    correct = torch.zeros(detections.shape[0], iouv.shape[0],
        dtype=torch.bool, device=iouv.device)
    iou = box_iou(labels[:, 1:], detections[:, :4])
    x = torch.where((iou >= iouv[0]) & (labels[:, 0:1] == detections[:, 5])) #
        IoU above threshold and classes match
    if x[0].shape[0]:
        matches = torch.cat((torch.stack(x, 1), iou[x[0], x[1]][:, None]),
            1).cpu().numpy() # [label, detection, iou]
        if x[0].shape[0] > 1:
            matches = matches[matches[:, 2].argsort()[::-1]]
            matches = matches[np.unique(matches[:, 1], return_index=True)[1]]
            # matches = matches[matches[:, 2].argsort()[::-1]]
            matches = matches[np.unique(matches[:, 0], return_index=True)[1]]
        matches = torch.from_numpy(matches).to(iouv.device)
        correct[matches[:, 1].long()] = matches[:, 2:3] >= iouv
    return correct
```

```
@torch.no_grad()
def run(
        data,
        weights=None, # model.pt path(s)
        batch_size=32, # batch size
        imgsz=640, # inference size (pixels)
        conf_thres=0.001, # confidence threshold
        iou_thres=0.6, # NMS IoU threshold
        task='val', # train, val, test, speed or study
        device='', # cuda device, i.e. 0 or 0,1,2,3 or cpu
        workers=8, # max dataloader workers (per RANK in DDP mode)
        single_cls=False, # treat as single-class dataset
        augment=False, # augmented inference
        verbose=False, # verbose output
        save_txt=False, # save results to *.txt
        save_hybrid=False, # save label+prediction hybrid results to *.txt
        save_conf=False, # save confidences in --save-txt labels
        save_json=False, # save a COCO-JSON results file
        project=ROOT / 'runs/val', # save to project/name
        name='exp', # save to project/name
        exist_ok=False, # existing project/name ok, do not increment
        half=True, # use FP16 half-precision inference
        dnn=False, # use OpenCV DNN for ONNX inference
        model=None,
        dataloader=None,
        save_dir=Path(''),
        plots=True,
        callbacks=Callbacks(),
        compute_loss=None,
):
    # Initialize/load model and set device
    training = model is not None
    if training: # called by train.py
        device, pt, jit, engine = next(model.parameters()).device, True,
            False, False # get model device, PyTorch model
        half &= device.type != 'cpu' # half precision only supported on CUDA
        model.half() if half else model.float()
    else: # called directly
        device = select_device(device, batch_size=batch_size)

        # Directories
```

```
        save_dir = increment_path(Path(project) / name, exist_ok=exist_ok) #
            increment run
        (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True,
            exist_ok=True) # make dir

        # Load model
        model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data,
            fp16=half)
        stride, pt, jit, engine = model.stride, model.pt, model.jit,
            model.engine
        imgsz = check_img_size(imgsz, s=stride) # check image size
        half = model.fp16 # FP16 supported on limited backends with CUDA
        if engine:
            batch_size = model.batch_size
        else:
            device = model.device
            if not (pt or jit):
                batch_size = 1 # export.py models default to batch-size 1
                LOGGER.info(f'Forcing --batch-size 1 square inference
                    (1,3,{imgsz},{imgsz}) for non-PyTorch models')

        # Data
        data = check_dataset(data) # check

    # Configure
    model.eval()
    cuda = device.type != 'cpu'
    is_coco = isinstance(data.get('val'), str) and
        data['val'].endswith('coco/val2017.txt') # COCO dataset
    nc = 1 if single_cls else int(data['nc']) # number of classes
    iouv = torch.linspace(0.5, 0.95, 10, device=device) # iou vector for
        mAP@0.5:0.95
    niou = iouv.numel()

    # Dataloader
    if not training:
        if pt and not single_cls: # check --weights are trained on --data
            ncm = model.model.yaml['nc']
            assert ncm == nc, f'{weights[0]} ({ncm} classes) trained on
                different --data than what you passed ({nc} ' \
                                f'classes). Pass correct combination of --weights
                                    and --data that are trained together.'
```

```
        model.warmup(imgsz=(1 if pt else batch_size, 3, imgsz, imgsz)) # warmup
        pad = 0.0 if task in ('speed', 'benchmark') else 0.5
        rect = False if task == 'benchmark' else pt # square inference for
            benchmarks
        task = task if task in ('train', 'val', 'test') else 'val' # path to
            train/val/test images
        dataloader = create_dataloader(data[task],
                                       imgsz,
                                       batch_size,
                                       stride,
                                       single_cls,
                                       pad=pad,
                                       rect=rect,
                                       workers=workers,
                                       prefix=colorstr(f'{task}: '))[0]

    seen = 0
    confusion_matrix = ConfusionMatrix(nc=nc)
    names = {k: v for k, v in enumerate(model.names if hasattr(model, 'names')
        else model.module.names)}
    class_map = coco80_to_coco91_class() if is_coco else list(range(1000))
    s = ('%20s' + '%11s' * 6) % ('Class', 'Images', 'Labels', 'P', 'R',
        'mAP@.5', 'mAP@.5:.95')
    dt, p, r, f1, mp, mr, map50, map = [0.0, 0.0, 0.0], 0.0, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0
    loss = torch.zeros(3, device=device)
    jdict, stats, ap, ap_class = [], [], [], []
    pbar = tqdm(dataloader, desc=s,
        bar_format='{l_bar}{bar:10}{r_bar}{bar:-10b}') # progress bar
    for batch_i, (im, targets, paths, shapes) in enumerate(pbar):
        t1 = time_sync()
        if cuda:
            im = im.to(device, non_blocking=True)
            targets = targets.to(device)
        im = im.half() if half else im.float() # uint8 to fp16/32
        im /= 255 # 0 - 255 to 0.0 - 1.0
        nb, _, height, width = im.shape # batch size, channels, height, width
        t2 = time_sync()
        dt[0] += t2 - t1

        # Inference
```

```
out, train_out = model(im) if training else model(im, augment=augment,
    val=True) # inference, loss outputs
dt[1] += time_sync() - t2

# Loss
if compute_loss:
    loss += compute_loss([x.float() for x in train_out], targets)[1] #
        box, obj, cls

# NMS
targets[:, 2:] *= torch.tensor((width, height, width, height),
    device=device) # to pixels
lb = [targets[targets[:, 0] == i, 1:] for i in range(nb)] if
    save_hybrid else [] # for autolabelling
t3 = time_sync()
out = non_max_suppression(out, conf_thres, iou_thres, labels=lb,
    multi_label=True, agnostic=single_cls)
dt[2] += time_sync() - t3

# Metrics
for si, pred in enumerate(out):
    labels = targets[targets[:, 0] == si, 1:]
    nl = len(labels)
    tcls = labels[:, 0].tolist() if nl else [] # target class
    path, shape = Path(paths[si]), shapes[si][0]
    seen += 1

    if len(pred) == 0:
        if nl:
            stats.append((torch.zeros(0, niou, dtype=torch.bool),
                torch.Tensor(), torch.Tensor(), tcls))
        continue

    # Predictions
    if single_cls:
        pred[:, 5] = 0
    predn = pred.clone()
    scale_coords(im[si].shape[1:], predn[:, :4], shape, shapes[si][1])
        # native-space pred

    # Evaluate
    if nl:
```

```
                tbox = xywh2xyxy(labels[:, 1:5]) # target boxes
                scale_coords(im[si].shape[1:], tbox, shape, shapes[si][1]) #
                    native-space labels
                labelsn = torch.cat((labels[:, 0:1], tbox), 1) # native-space
                    labels
                correct = process_batch(predn, labelsn, iouv)
                if plots:
                    confusion_matrix.process_batch(predn, labelsn)
            else:
                correct = torch.zeros(pred.shape[0], niou, dtype=torch.bool)
            stats.append((correct.cpu(), pred[:, 4].cpu(), pred[:, 5].cpu(),
                tcls)) # (correct, conf, pcls, tcls)

            # Save/log
            if save_txt:
                save_one_txt(predn, save_conf, shape, file=save_dir / 'labels'
                    / (path.stem + '.txt'))
            if save_json:
                save_one_json(predn, jdict, path, class_map) # append to
                    COCO-JSON dictionary
            callbacks.run('on_val_image_end', pred, predn, path, names, im[si])

        # Plot images
        if plots and batch_i < 3:
            f = save_dir / f'val_batch{batch_i}_labels.jpg' # labels
            Thread(target=plot_images, args=(im, targets, paths, f, names),
                daemon=True).start()
            f = save_dir / f'val_batch{batch_i}_pred.jpg' # predictions
            Thread(target=plot_images, args=(im, output_to_target(out), paths,
                f, names), daemon=True).start()

    # Compute metrics
    stats = [np.concatenate(x, 0) for x in zip(*stats)] # to numpy
    if len(stats) and stats[0].any():
        tp, fp, p, r, f1, ap, ap_class = ap_per_class(*stats, plot=plots,
            save_dir=save_dir, names=names)
        ap50, ap = ap[:, 0], ap.mean(1) # AP@0.5, AP@0.5:0.95
        mp, mr, map50, map = p.mean(), r.mean(), ap50.mean(), ap.mean()
        nt = np.bincount(stats[3].astype(np.int64), minlength=nc) # number of
            targets per class
    else:
        nt = torch.zeros(1)
```

```
# Print results
pf = '%20s' + '%11i' * 2 + '%11.3g' * 4 # print format
LOGGER.info(pf % ('all', seen, nt.sum(), mp, mr, map50, map))

# Print results per class
if (verbose or (nc < 50 and not training)) and nc > 1 and len(stats):
    for i, c in enumerate(ap_class):
        LOGGER.info(pf % (names[c], seen, nt[c], p[i], r[i], ap50[i],
            ap[i]))

# Print speeds
t = tuple(x / seen * 1E3 for x in dt) # speeds per image
if not training:
    shape = (batch_size, 3, imgsz, imgsz)
    LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS
        per image at shape {shape}' % t)

# Plots
if plots:
    confusion_matrix.plot(save_dir=save_dir, names=list(names.values()))
    callbacks.run('on_val_end')

# Save JSON
if save_json and len(jdict):
    w = Path(weights[0] if isinstance(weights, list) else weights).stem if
        weights is not None else '' # weights
    anno_json = str(Path(data.get('path', '../coco')) /
        'annotations/instances_val2017.json') # annotations json
    pred_json = str(save_dir / f"{w}_predictions.json") # predictions json
    LOGGER.info(f'\nEvaluating pycocotools mAP... saving {pred_json}...')
    with open(pred_json, 'w') as f:
        json.dump(jdict, f)

    try: #
        https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocoEvalDemo.ipynb
        check_requirements(['pycocotools'])
        from pycocotools.coco import COCO
        from pycocotools.cocoeval import COCOeval

        anno = COCO(anno_json) # init annotations api
        pred = anno.loadRes(pred_json) # init predictions api
```

```
            eval = COCOeval(anno, pred, 'bbox')
            if is_coco:
                eval.params.imgIds = [int(Path(x).stem) for x in
                    dataloader.dataset.im_files] # image IDs to evaluate
            eval.evaluate()
            eval.accumulate()
            eval.summarize()
            map, map50 = eval.stats[:2] # update results (mAP@0.5:0.95, mAP@0.5)
        except Exception as e:
            LOGGER.info(f'pycocotools unable to run: {e}')

    # Return results
    model.float() # for training
    if not training:
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to
            {save_dir / 'labels'}" if save_txt else ''
        LOGGER.info(f"Results saved to {colorstr('bold', save_dir)}{s}")
    maps = np.zeros(nc) + map
    for i, c in enumerate(ap_class):
        maps[c] = ap[i]
    return (mp, mr, map50, map, *(loss.cpu() / len(dataloader)).tolist()),
        maps, t


def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--data', type=str, default=ROOT /
        'data/coco128.yaml', help='dataset.yaml path')
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT /
        'yolov5s.pt', help='model.pt path(s)')
    parser.add_argument('--batch-size', type=int, default=32, help='batch
        size')
    parser.add_argument('--imgsz', '--img', '--img-size', type=int,
        default=640, help='inference size (pixels)')
    parser.add_argument('--conf-thres', type=float, default=0.001,
        help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.6, help='NMS IoU
        threshold')
    parser.add_argument('--task', default='val', help='train, val, test, speed
        or study')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or
        0,1,2,3 or cpu')
```

```
    parser.add_argument('--workers', type=int, default=8, help='max dataloader
        workers (per RANK in DDP mode)')
    parser.add_argument('--single-cls', action='store_true', help='treat as
        single-class dataset')
    parser.add_argument('--augment', action='store_true', help='augmented
        inference')
    parser.add_argument('--verbose', action='store_true', help='report mAP by
        class')
    parser.add_argument('--save-txt', action='store_true', help='save results
        to *.txt')
    parser.add_argument('--save-hybrid', action='store_true', help='save
        label+prediction hybrid results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save
        confidences in --save-txt labels')
    parser.add_argument('--save-json', action='store_true', help='save a
        COCO-JSON results file')
    parser.add_argument('--project', default=ROOT / 'runs/val', help='save to
        project/name')
    parser.add_argument('--name', default='exp', help='save to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing
        project/name ok, do not increment')
    parser.add_argument('--half', action='store_true', help='use FP16
        half-precision inference')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for
        ONNX inference')
    opt = parser.parse_args()
    opt.data = check_yaml(opt.data) # check YAML
    opt.save_json |= opt.data.endswith('coco.yaml')
    opt.save_txt |= opt.save_hybrid
    print_args(vars(opt))
    return opt


def main(opt):
    check_requirements(requirements=ROOT / 'requirements.txt',
        exclude=('tensorboard', 'thop'))

    if opt.task in ('train', 'val', 'test'): # run normally
        if opt.conf_thres > 0.001: #
            https://github.com/ultralytics/yolov5/issues/1466
            LOGGER.info(f'WARNING: confidence threshold {opt.conf_thres} >>
                0.001 will produce invalid mAP values.')
```

```
            run(**vars(opt))

        else:
            weights = opt.weights if isinstance(opt.weights, list) else
                [opt.weights]
            opt.half = True # FP16 for fastest results
            if opt.task == 'speed': # speed benchmarks
                # python val.py --task speed --data coco.yaml --batch 1 --weights
                    yolov5n.pt yolov5s.pt...
                opt.conf_thres, opt.iou_thres, opt.save_json = 0.25, 0.45, False
                for opt.weights in weights:
                    run(**vars(opt), plots=False)

            elif opt.task == 'study': # speed vs mAP benchmarks
                # python val.py --task study --data coco.yaml --iou 0.7 --weights
                    yolov5n.pt yolov5s.pt...
                for opt.weights in weights:
                    f = f'study_{Path(opt.data).stem}_{Path(opt.weights).stem}.txt'
                        # filename to save to
                    x, y = list(range(256, 1536 + 128, 128)), [] # x axis (image
                        sizes), y axis
                    for opt.imgsz in x: # img-size
                        LOGGER.info(f'\nRunning {f} --imgsz {opt.imgsz}...')
                        r, _, t = run(**vars(opt), plots=False)
                        y.append(r + t) # results and times
                    np.savetxt(f, y, fmt='%10.4g') # save
                os.system('zip -r study.zip study_*.txt')
                plot_val_study(x=x) # plot


if __name__ == "__main__":
    opt = parse_opt()
    main(opt)

import argparse
import os
import sys
from pathlib import Path

import torch
import torch.backends.cudnn as cudnn
```

```
FILE = Path(__file__).resolve()
ROOT = FILE.parents[0] # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT)) # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd())) # relative

from models.common import DetectMultiBackend
from utils.datasets import IMG_FORMATS, VID_FORMATS, LoadImages, LoadStreams
from utils.general import (LOGGER, check_file, check_img_size, check_imshow,
    check_requirements, colorstr, cv2,
                           increment_path, non_max_suppression, print_args,
                               scale_coords, strip_optimizer, xyxy2xywh)
from utils.plots import Annotator, colors, save_one_box
from utils.torch_utils import select_device, time_sync


@torch.no_grad()
def run(
        weights=ROOT / 'yolov5s.pt', # model.pt path(s)
        source=ROOT / 'data/images', # file/dir/URL/glob, 0 for webcam
        data=ROOT / 'data/coco128.yaml', # dataset.yaml path
        imgsz=(640, 640), # inference size (height, width)
        conf_thres=0.25, # confidence threshold
        iou_thres=0.45, # NMS IOU threshold
        max_det=1000, # maximum detections per image
        device='', # cuda device, i.e. 0 or 0,1,2,3 or cpu
        view_img=False, # show results
        save_txt=False, # save results to *.txt
        save_conf=False, # save confidences in --save-txt labels
        save_crop=False, # save cropped prediction boxes
        nosave=False, # do not save images/videos
        classes=None, # filter by class: --class 0, or --class 0 2 3
        agnostic_nms=False, # class-agnostic NMS
        augment=False, # augmented inference
        visualize=False, # visualize features
        update=False, # update all models
        project=ROOT / 'runs/detect', # save results to project/name
        name='exp', # save results to project/name
        exist_ok=False, # existing project/name ok, do not increment
        line_thickness=3, # bounding box thickness (pixels)
        hide_labels=True, # hide labels
        hide_conf=True, # hide confidences
```

```
        half=False, # use FP16 half-precision inference
        dnn=False, # use OpenCV DNN for ONNX inference
):
    source = str(source)
    save_img = not nosave and not source.endswith('.txt') # save inference
        images
    is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
    is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://',
        'https://'))
    webcam = source.isnumeric() or source.endswith('.txt') or (is_url and not
        is_file)
    if is_url and is_file:
        source = check_file(source) # download

    # Directories
    save_dir = increment_path(Path(project) / name, exist_ok=exist_ok) #
        increment run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True,
        exist_ok=True) # make dir

    # Load model
    device = select_device(device)
    model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data,
        fp16=half)
    stride, names, pt = model.stride, model.names, model.pt
    imgsz = check_img_size(imgsz, s=stride) # check image size

    # Dataloader
    if webcam:
        view_img = check_imshow()
        cudnn.benchmark = True # set True to speed up constant image size
            inference
        dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt)
        bs = len(dataset) # batch_size
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt)
        bs = 1 # batch_size
    vid_path, vid_writer = [None] * bs, [None] * bs

    # Run inference
    model.warmup(imgsz=(1 if pt else bs, 3, *imgsz)) # warmup
    dt, seen = [0.0, 0.0, 0.0], 0
```

```
for path, im, im0s, vid_cap, s in dataset:
    t1 = time_sync()
    im = torch.from_numpy(im).to(device)
    im = im.half() if model.fp16 else im.float() # uint8 to fp16/32
    im /= 255 # 0 - 255 to 0.0 - 1.0
    if len(im.shape) == 3:
        im = im[None] # expand for batch dim
    t2 = time_sync()
    dt[0] += t2 - t1

    # Inference
    visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if
        visualize else False
    pred = model(im, augment=augment, visualize=visualize)
    t3 = time_sync()
    dt[1] += t3 - t2

    # NMS
    pred = non_max_suppression(pred, conf_thres, iou_thres, classes,
        agnostic_nms, max_det=max_det)
    dt[2] += time_sync() - t3

    # Second-stage classifier (optional)
    # pred = utils.general.apply_classifier(pred, classifier_model, im,
        im0s)

    # Process predictions
    for i, det in enumerate(pred): # per image
        seen += 1
        if webcam: # batch_size >= 1
            p, im0, frame = path[i], im0s[i].copy(), dataset.count
            s += f'{i}: '
        else:
            p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)

        p = Path(p) # to Path
        save_path = str(save_dir / p.name) # im.jpg
        txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode
            == 'image' else f'_{frame}') # im.txt
        s += '%gx%g ' % im.shape[2:] # print string
        gn = torch.tensor(im0.shape)[[1, 0, 1, 0]] # normalization gain whwh
        imc = im0.copy() if save_crop else im0 # for save_crop
```

```
annotator = Annotator(im0, line_width=line_thickness,
    example=str(names))
if len(det):
    # Rescale boxes from img_size to im0 size
    det[:, :4] = scale_coords(im.shape[2:], det[:, :4],
        im0.shape).round()

    # Print results
    for c in det[:, -1].unique():
        n = (det[:, -1] == c).sum() # detections per class
        s += f"{n} {names[int(c)]}{'s' * (n > 1)}, " # add to string

    # Write results
    for *xyxy, conf, cls in reversed(det):
        if save_txt: # Write to file
            xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) /
                gn).view(-1).tolist() # normalized xywh
            line = (cls, *xywh, conf) if save_conf else (cls, *xywh)
                # label format
            with open(txt_path + '.txt', 'a') as f:
                f.write(('%g ' * len(line)).rstrip() % line + '\n')

        if save_img or save_crop or view_img: # Add bbox to image
            c = int(cls) # integer class
            label = None if hide_labels else (names[c] if hide_conf
                else f'{names[c]} {conf:.2f}')
            annotator.box_label(xyxy, label, color=colors(c, True))
            if save_crop:
                save_one_box(xyxy, imc, file=save_dir / 'crops' /
                    names[c] / f'{p.stem}.jpg', BGR=True)

# Stream results
im0 = annotator.result()
if view_img:
    cv2.imshow(str(p), im0)
    cv2.waitKey(1) # 1 millisecond

# Save results (image with detections)
if save_img:
    if dataset.mode == 'image':
        cv2.imwrite(save_path, im0)
    else: # 'video' or 'stream'
```

```
                    if vid_path[i] != save_path: # new video
                        vid_path[i] = save_path
                        if isinstance(vid_writer[i], cv2.VideoWriter):
                            vid_writer[i].release() # release previous video
                                writer
                        if vid_cap: # video
                            fps = vid_cap.get(cv2.CAP_PROP_FPS)
                            w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                            h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                        else: # stream
                            fps, w, h = 30, im0.shape[1], im0.shape[0]
                        save_path = str(Path(save_path).with_suffix('.mp4')) #
                            force *.mp4 suffix on results videos
                        vid_writer[i] = cv2.VideoWriter(save_path,
                            cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
                    vid_writer[i].write(im0)

            # Print time (inference-only)
            LOGGER.info(f'{s}Done. ({t3 - t2:.3f}s)')

        # Print results
        t = tuple(x / seen * 1E3 for x in dt) # speeds per image
        LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS per
            image at shape {(1, 3, *imgsz)}' % t)
        if save_txt or save_img:
            s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to
                {save_dir / 'labels'}" if save_txt else ''
            LOGGER.info(f"Results saved to {colorstr('bold', save_dir)}{s}")
        if update:
            strip_optimizer(weights) # update model (to fix SourceChangeWarning)


def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT /
        'yolov5s.pt', help='model path(s)')
    parser.add_argument('--source', type=str, default=ROOT / 'data/images',
        help='file/dir/URL/glob, 0 for webcam')
    parser.add_argument('--data', type=str, default=ROOT /
        'data/coco128.yaml', help='(optional) dataset.yaml path')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int,
        default=[640], help='inference size h,w')
```

```
    parser.add_argument('--conf-thres', type=float, default=0.25,
        help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU
        threshold')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum
        detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or
        0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results
        to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save
        confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped
        prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save
        images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by
        class: --classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true',
        help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented
        inference')
    parser.add_argument('--visualize', action='store_true', help='visualize
        features')
    parser.add_argument('--update', action='store_true', help='update all
        models')
    parser.add_argument('--project', default=ROOT / 'runs/detect', help='save
        results to project/name')
    parser.add_argument('--name', default='exp', help='save results to
        project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing
        project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int,
        help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true',
        help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true',
        help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16
        half-precision inference')
```

```
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for
        ONNX inference')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1 # expand
    print_args(vars(opt))
    return opt


def main(opt):
    check_requirements(exclude=('tensorboard', 'thop'))
    run(**vars(opt))


if __name__ == "__main__":
    opt = parse_opt()
    main(opt)


%pip install -qr requirements.txt
%pip install -q roboflow

import torch
import os
from IPython.display import Image, clear_output

from roboflow import Roboflow
rf = Roboflow(api_key="OGPhH6OvEzdv0Z02XTYX")
project = rf.workspace("joao-victor").project("helipad-ekash")
dataset = project.version(4).download("yolov5")

gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Sem Gpu')
else:
  print(gpu_info)

!python train.py --img 416 --batch 16 --epochs 150 --data
    {dataset.location}/data.yaml --weights yolov5s.pt

!python detect.py --weights runs/train/exp/weights/best.pt --img 416 --conf
    0.5 --source {dataset.location}/test/images
```

```
import glob
from IPython.display import Image, display

for imageName in glob.glob('./runs/detect/exp20/*.jpg'):
    display(Image(filename=imageName))
    print("\n")
```

APÊNDICE B Figuras

Figura 51 Fluxograma do método de trabalho

Figura 52 Work ow do projeto.

Figura 53 Heliporto.

Fonte: Autoral

Figura 54 Heliporto.

Fonte: Autoral

Figura 55 Heliporto.

Fonte: Autoral

Figura 56 Heliporto.

Fonte: Autoral

Figura 57 Heliporto.

Fonte: Autoral

Figura 58 Heliporto.

Fonte: Autoral

Figura 59 Heliporto.

Fonte: Autoral

Figura 60 Heliporto.

Fonte: Autoral

Figura 61 – Heliporto.

Fonte: Autoral

Figura 62 – Heliporto.

Fonte: Autoral