Politécnico
Castelo Branco
Escola Superior
de Tecnologia

Disaster Person Detector - A importância da Inteligência Artificial na Deteção de Pessoas em Desastres

Projeto I

Beatriz Sofia da Fonseca Marques nº20220447

Filipe Alexandre Mira Serôdio nº20220396

Orientadores

Professora Doutora Ana Paula Neves Ferreira da Silva

Professor Doutor Arlindo Ferreira da Silva

Trabalho de Projeto apresentado à Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco para cumprimento dos requisitos necessários à obtenção do grau de Licenciado em Engenharia Informática, realizado sob a orientação científica da Professora Doutora Ana Paula Neves Ferreira da Silva e Professor Doutor Arlindo Ferreira da Silva, do Instituto Politécnico de Castelo Branco.

Setembro 2025

Composição do júri

Presidente do júri

Doutor, Alexandre José Pereira Duro da Fonte

Professor Adjunto, Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco

Vogais

Doutora, Ana Paula Neves Ferreira da Silva

Professora Adjunta, Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco

Doutor, José Carlos Meireles Monteiro Metrolho

Professor Coordenador, Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco

Resumo

Durante situações de desastres como inundações, terramotos, a urgência em resgatar as pessoas que estão em risco de perigo representa um desafio difícil. As operações tradicionais de busca e resgate conduzidas por humanos têm dificuldades devido às condições adversas que enfrentam e ao limite de recursos.

Para superar estes desafios, têm sido usados *Unmanned Aerial Vehicles* (UAVs), utilizam o processamento de imagens para localizar humanos no meio do desastre. Contudo, as aplicações existentes, que dependem de abordagens baseadas no *Deep Learning* (DL) para analisar as imagens, são lentas a detetar as pessoas. Este atraso é um problema crítico, considerando que a rapidez é essencial em situações de emergência. Neste trabalho propomos a utilização de técnicas de DL para a criação de modelos inteligentes com o objetivo de conseguirem ajudar os operacionais a encontrar pessoas em cenários de desastre.

Para se perceber melhor o trabalho de investigação dedicado à resolução deste problema foi realizado um estado da arte de modelos de DL aplicadas à deteção de pessoas em cenários de desastre. Este estudo permitiu constatar um interesse crescente neste tema nos últimos anos.

O trabalho desenvolvido implicou também a procura de *datasets* para deteção de pessoas em cenários de desastre adequados para serem utilizados no treino dos modelos. Esta procura evidenciou que os *datasets* já existentes não representavam bem os cenários de desastre e, portanto, os autores dos artigos analisados decidiram criar os próprios. Após selecionar e analisar o *dataset* escolhido procedeu-se à fase de treino de um modelo, avançando assim no desenvolvimento da solução proposta.

Palavras chave

Deteção de pessoas em cenários de desastre; *Deep Learning* (DL); Modelos de *Deep Learning*; *Dataset* para deteção de pessoas em cenários de desastre

Abstract

During disasters like floods, earthquakes, the urgency in rescuing people in imminent danger its a hard challenge. Traditional operations of search and rescue conducted by humans face difficulties due to the adverse conditions they encounter and the limited resources.

To overcome these challenges, Unmanned Aerial Vehicles (UAVs) have been used, which use image processing to locate humans in the middle of the disaster. However, existing applications, which rely on Deep Learning (DL) based approaches to analyze the images, are slow to detect people. This delay in response is a critical problem, considering that speed is essential in emergency situations. In this study we propose the utilization of DL techniques to create intelligent models capable of helping operatives find people in disaster scenarios.

To better understand the research work available in the literature dedicated to solving this problem a state-of-the-art study was carried out on the use of DL models used to detect people in disaster scenarios. This study revealed a growing interest in this topic in recent years.

The work carried out also involved looking for suitable datasets for detection of people in disaster scenarios to train the models. This search revealed that the existing datasets didn't represent accurately disaster scenarios, therefore, the authors of the articles analyzed opted for creating their own. After selecting and analyzing the dataset chosen the training phase of a model was carried out, thus advancing the development of the proposed solution.

Keywords

Detection of people in disaster scenario; Deep Learning (DL); Deep Learning Models; Dataset for detection of people in disaster scenarios

Índice geral

Índice de figuras

Índice de equações

Lista de tabelas

Lista de abreviaturas, siglas e acrónimos

ADAM (Adaptive Moment Estimation)

AI (Artificial Intelligence)

AIDER (Aerial Image Dataset for Emergency Response Applications)

ANN (Redes Neuronais Artificiais)

C2A (Combination to Application)

CNN (Convolutional Neural Networks)

COCO (Common Object in Context)

DAB-DETR (Dynamic Anchor Box DETR)

DETR (Detection Transformer)

DDETR (Deformable DETR)

DINO (DETR with Improved Denoising Achor Box)

DN-DETR (Denoising DETR)

DL (Deep Learning)

FC (Fully-Connected)

FCOS (Fully Convolutional One-Stage Object Detection)

GRU (Gated Recurrent Units)

IA (Inteligência Artificial)

im-YOLOv5 (Improved YOLOv5)

k-NN (K-Nearest Neighbors)

LIP (Look into Person)

LSP/MPII-MPHB (Leeds Sports Pose/ Multiple Poses Human Body)

LSTM (Long Short-Term Memory)

mAP (mean Average Precision)

MDP (Markov Decision Process)

ML (Machine Learning)

OCR (Optical Character Recognition)

P (Precisão)

PAFNet (Paddle Anchor Free Network)

PDD (Pos-Disaster Dataset)

PGI (Informação de Gradiente Programável)

PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses)

R (Recall)

R-CNN (Regional-based Convolutional Neural Network)

RNN (Recurrent Neural Networks)

RPN (Rede de Propostas Regionais)

SAR (Search and Rescue)

SARD (Software Assurance Reference Dataset)

SOTA (State-of-the-art)

SPPF (Spatial Pyramid Pooling – Fast)

SVM (Support-Vector Machine)

TTFNet (Training-Time-Friendly Network for Real-Time Objetcion Detection)

U2Net (U-Squared Net)

UAV (Unmanned Aerial Vehicles)

V-REP (Virtual Robot Experimentation Plataform)

VOC (Visual Object Classes)

YOLO (You Only Look Once)

YOLOv5l (YOLOv5 large)

YOLOv9-c (YOLOv9 compact)

YOLOv9-e (YOLOv9 extended)

YOLO-NASm (You Only Look Once Neural Architecture Search medium)

1. Introdução

Os desastres naturais constituem uma das maiores ameaças para qualquer sociedade no mundo, devido à sua imprevisibilidade e ao elevado potencial de destruição. Ao longo dos anos, tem-se verificado um aumento no número de ocorrências desses fenómenos [1]. Estes fenómenos não causam apenas danos materiais às sociedades que afetam, mas também levam à perda de vidas. Para mitigar esses danos, existem equipas de resgate que se deslocam às áreas afetadas, com o objetivo de salvar as pessoas em situações de risco. No entanto, estas equipas enfrentam condições adversas devido aos terrenos adversos e de difícil acesso, o que torna as operações de resgate ainda mais desafiadoras [2].

Posto isto, um dos grandes desafios enfrentados pelas equipas de resgate consiste em localizar as vítimas de forma rápida, eficiente e segura, de modo a evitar acidentes desnecessários. Neste sentido, com os avanços da Inteligência Artificial (IA) e do *Machine Learning* (ML), surgiram novas ferramentas para apoiar as equipas de resgate. Sistemas automatizados baseados em IA, particularmente redes neurais convolucionais, têm sido desenvolvidos para analisar imagens capturadas por UAV durante missões de busca e resgate. Estes sistemas são capazes de processar rapidamente grandes quantidades de imagens, identificando e localizando vítimas, especialmente em ambientes desafiadores onde o acesso humano é limitado ou perigoso [3].

Nas secções seguintes deste capítulo, serão apresentados o enquadramento deste trabalho, os objetivos que se pretendem alcançar com o seu desenvolvimento, o planeamento do projeto e a organização do relatório.

1.1. Enquadramento

Como mencionado anteriormente, um dos grandes desafios para as equipas de resgate em cenários de desastres naturais, é a localização de vítimas no menor tempo possível, da forma mais eficaz e segura possível. No entanto, ao recorrer exclusivamente aos métodos tradicionais, baseados fundamentalmente na mão de obra humana, as equipas de resgate tornam se muito limitadas pelo tempo, pela capacidade física e pela dificuldade em cobrir grandes áreas. Por esse motivo, tecnologias como os UAVs já se consolidaram como ferramentas indispensáveis nessas operações, especialmente quando combinadas com sistemas automatizados baseados em Inteligência Artificial [4].

Estes sistemas automatizados têm um grande potencial para ajudar nestas operações, ao permitir a análise eficiente de imagens capturadas pelos *Unmanned Aerial Vehicles* (UAVs) e a deteção precisa de pessoas em cenários adversos. No entanto, o desenvolvimento de modelos de IA para este propósito apresenta alguns desafios. Fatores como a diversidade de posturas corporais das vítimas, oclusões causadas por escombros e a complexidade dos ambientes dificultam a tarefa [3].

Com base nestes desafios, este projeto procura explorar a aplicação de algoritmos de DL para treinar um modelo a partir de um *dataset* representativo, capaz de identificar pessoas em diferentes cenários. A solução proposta não só visa aumentar a eficácia das operações de resgate, mas também reduzir os custos associados ao uso intensivo de recursos humanos.

Adicionalmente, espera-se que o projeto contribua para a investigação no domínio da IA aplicada a missões de resgate, abordando limitações existentes e propondo melhorias que possibilitem uma transição mais eficaz para cenários do mundo real.

1.2. Objetivos

No âmbito da Unidade Curricular de Projeto I, pretende-se identificar os *datasets* disponíveis e fazer uma revisão do trabalho já feito de forma a identificar os melhores modelos para serem testados. Desta forma, foram delineados determinados objetivos com o intuito de confirmar se é possível, ou não, treinar uma rede neuronal convolucional para detetar eficazmente sinistrados. Assim, os objetivos delineados são:

- Fazer uma revisão do estado da arte, para verificar que modelos de DL já foram utilizados;

- Fazer um levantamento dos *datasets* existentes;
- Identificar bibliotecas em Python que permitem o treino de redes convolucionais;
- Fazer um primeiro trabalho experimental, usando um dos *datasets* disponíveis que permita ajustar os modelos de DL mais promissores na deteção de sinistrados.

1.3. Planeamento do Projeto

O trabalho foi dividido em cinco tarefas que foram realizadas ao longo do 1º semestre do ano letivo 24/25. Essas tarefas foram:

- **Tarefa A**: Elaboração do relatório: esta tarefa estendeu-se ao longo dos dois semestres, sendo continuamente atualizado o progresso do trabalho.
- **Tarefa B:** Revisão Sistemática: realização de uma investigação sobre a aplicação dos modelos de aprendizagem profunda (DL) na procura de pessoas em cenários de desastre.
- **Tarefa C:** Procura de *datasets*: pesquisa e identificação de conjuntos de dados publicados por outros investigadores com aplicação na procura de pessoas em cenários de desastre.
- **Tarefa D:** Identificar bibliotecas: identificação das bibliotecas em Python que permitem o treino dos modelos.
- **Tarefa E:** Realização do primeiro trabalho experimental: utilizando um dos *datasets* disponíveis, procurou-se tentar treinar os modelos mais promissores na procura de pessoas em cenários de desastre disponíveis.

Na Tabela 1 podemos ver o cronograma das tarefas realizadas ao longo do Projeto I.

Tabela 1- Cronograma do Projeto I

	2024				2025						
Tarefas	Setembro	Outubro	Novembro	Dezembro	Janeiro	Fevereiro	Março	Abril	Maio	Junho	Julho
A											
B											
C											
D											
E											

1.4. Estrutura do Relatório

Este relatório é constituído por seis capítulos. Neste primeiro capítulo é descrito o problema e a solução procurada. Para além da contextualização, são apresentados os objetivos delineados e as tarefas planeadas, numa fase inicial, a serem seguidas ao longo do seu desenvolvimento.

No segundo capítulo, explora-se a área de IA, com ênfase na sua subárea de DL mais especificamente redes neuronais e visão computacional, uma vez que estas são o foco deste trabalho. Adicionalmente, são explorados os diferentes tipos de aprendizagem computacional existentes e como se processa a sua avaliação.

No terceiro capítulo, apresenta-se o estudo do estado da arte que foi realizado e onde se relata um conjunto de trabalhos publicados que fazem uso de DL para a deteção de pessoas em cenários de desastre. Neste capítulo, são delineados os objetivos da pesquisa efetuada, identificados os diversos artigos que foram encontrados e posteriormente uma análise dos mesmo. Este capítulo encerra com as principais conclusões retiradas da análise.

No quarto capítulo, são apresentadas todas as tecnologias e ferramentas aplicadas neste trabalho. Sendo apresentado o propósito da sua utilização neste trabalho.

No quinto capítulo, são identificados os *datasets* que vão ser utilizados no treino bem como a sua estrutura, são identificados também os modelos de DL que vão ser treinados e é feito um primeiro trabalho experimental. Este capítulo encerra com as conclusões que obtivemos durante o mesmo.

Por fim, o sexto capítulo apresenta as principais conclusões retiradas do desenvolvimento deste trabalho e complementarmente são descritos os desafios enfrentados e trabalho futuro.

2. Inteligência Artificial (IA)

Em 1950, Alan Turing publicou "Computing Machinery and Intelligence". Neste artigo Turing pergunta "Será que máquinas conseguem pensar?" ("Can machine think?"). Para isto criou um teste "Teste de Turing" ou "Jogo de Imitação" ("Turing Test" ou "Imitation Game"), onde um interrogador humano tenta distinguir um computador de um humano a partir de respostas em texto [5]. O teste consiste em um humano interrogador, um humano e a entidade artificial todos em salas diferentes. O interrogador apenas pode comunicar com o outro humano e com entidade artificial através de um terminal e é lhe pedido para determinar qual é o humano e qual é a entidade artificial a partir das respostas dadas por ambos a um conjunto de questões. Se o interrogador não for capaz de distinguir qual é a entidade artificial considera-se que a entidade passa ao teste e é considerada inteligente [6].

Em 1956, John McCarthy cunhou o termo *Artificial Intelligence (AI)*, Inteligência Artificial (IA) em português, na primeira conferência sobre IA na universidade em Dartmouth. Uns anos mais tarde apresentou um artigo importante, propôs um paradigma para resolver o problema de senso comum com o uso de modelos formais de raciocínio baseados em lógica. Criou a linguagem LISP, que ainda é usado atualmente para construir sistemas de IA [5].

Searle, em 1980, apresentou uma objeção ao teste de Turing com a sua experiência "Sala Chinesa" ("Chinese Room"), supondo que criamos um programa que aparenta compreender chinês, se metermos um *input* com caracteres chineses ele deve processá-lo e produzir um *output* em caracteres chineses, se for capaz de convencer um interrogador então passa ao teste de Turing. Searle coloca a questão "O programa consegue mesmo compreender Chinês ou sabe apenas simular a habilidade de compreender Chinês?" [6, 7]. Com os resultados desta experiência, Searle dividiu visões de IA em dois grupos. *Weak AI* trata os computadores como um dispositivo útil para testar hipóteses relacionadas com processos mentais e para simular o desempenho cerebral. *Strong AI* é considerado um computador programado apropriadamente de forma a ser equivalente a um cérebro humano e as suas atividades mentais [7].

Atualmente uma IA é uma tecnologia que permite a computadores e máquinas simular comportamentos humanos, como a aprendizagem, compreensão, resolução de problemas. A IA é composta por várias subáreas cada uma focada em aspetos específicos de replicação da inteligência humana ou resolver problemas de um certo tipo. Apesar de algumas das subáreas se sobreporem as principais são Machine Learning, Deep Leaning, Redes Neuronais, Robótica, Processamento de Linguagem Natural, Visão Computacional [8]. Normalmente a divisão é feita em apenas 3 áreas, IA, ML e DL (Ver Figura 1, Fonte: [9]). Nesta aplicação o nosso foco vai ser no DL e nas Redes Neuronais mais especificamente as Redes Neuronais Convolucionais.

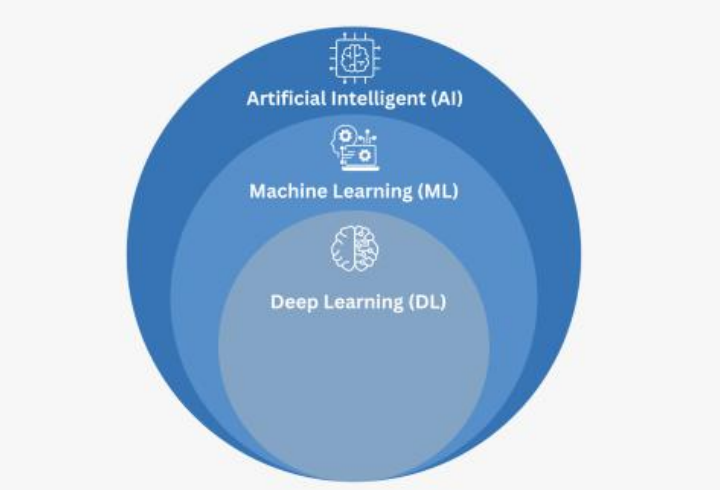

Figura 1- IA, ML e DL

Perceber os 4 tipos de IA ajuda a entender melhor ML.

- "Máquinas Reativas" (Reactive machine) este tipo de sistemas IA funciona seguindo regras pré-definidas, não têm a capacidade de aprender com nova informação.
- "Memória Limitada" (Limited memory) estes sistemas já têm a capacidade de aprender de experiências passadas. Desta forma estes sistemas podem treinar e fazer decisões informadas.
- "Teoria de Mente" (Theory of mind) descreve a ideia de que um sistema de IA possa entender emoções humanas e usar essa informação para prever futuras ações e decidir sozinho. Este tipo de sistema pode revolucionar muitas áreas permitindo um comportamento mais intuitivo e empático das máquinas.
- "IA Autoconsciente" (Self-aware AI) refere o cenário hipotético de que um sistema IA possa ter autoconsciência, possui uma consciência humana e compreende a sua própria existência no mundo [8].

2.1. Machine Learning (ML)

ML é um tipo de IA que permite às máquinas aprenderem a partir dos dados fornecidos sem ser explicitamente programadas para isso. O processo de aprendizagem consiste em ajustar e otimizar os parâmetros dos modelos, de modo que o comportamento do modelo generalize padrões presentes nos dados de treino e possa realizar previsões ou classificações sobre novos dados [10]. ML consiste em criar um algoritmo que faça previsões de forma eficiente e precisa, o sucesso destes algoritmos depende do tipo de dados que lhes é fornecido visto que está inerentemente relacionado com análise de dados e estatísticas.

Machine Learning admite várias aplicações práticas como visão computacional que inclui reconhecimento de objetos e deteção facial, deteção de fraude para cartões de crédito, aprender a jogar jogos como xadrez, sistema de recomendações muito utilizado em redes sociais e sites de compras online [11].

Podemos dividir o processo de aprendizagem dos algoritmos de Machine Learning em 3 partes a primeira é o Processo de Decisão, os algoritmos são usados para fazer previsões e classificações, baseado nos dados fornecidos o algoritmo vai produzir uma estimativa sobre um padrão nos dados. O segundo é Função de Erro, esta função vai avaliar as previsões do modelo, se houver exemplos a função pode comparar a precisão do modelo com o que já tinha sido apresentado anteriormente. O terceiro é Processo de Otimização do Modelo, se o modelo demonstrar potencial para um ajuste mais preciso aos dados do conjunto de treino, ajustamos os pesos para reduzir uma discrepância que possa haver com os outros exemplos. Este processo é repetido e os pesos vão sendo atualizados autonomamente pelo algoritmo até o nível de erro aceitável seja atingido [12].

Machine Learning abrange técnicas que permitem às máquinas identificar padrões e melhorar o seu desempenho aprendendo com dados, no entanto para treinar têm de usar um modelo. Podemos dividir esses modelos em 3 categorias "Aprendizagem Supervisionada", "Aprendizagem Não Supervisionada" e "Aprendizagem por Reforço". (Ver Figura 2. Fonte: [13])

Figura 2- Tipos de aprendizagem

2.1.1. Aprendizagem Supervisionada

Aprendizagem supervisionada é usada para prever os resultados ou classificar dados a partir do treino do algoritmo com *datasets* onde os dados estão identificados (*label*). Os dados são introduzidos no modelo, o modelo ajusta os pesos até ficar ajustado apropriadamente [12]. Este tipo de aprendizagem ajuda organizações a resolver problemas de grande escala, alguns métodos utilizados são:

- Naive Bayes: Como podemos entender pelo nome, este algoritmo utiliza o teorema de Bayes, é um algoritmo que assume independência entre variáveis e usa probabilidades para classificar objetos a partir de recursos. Isso significa que a presença de uma característica não afeta a presença de outra na probabilidade de um determinado resultado [14].
- Regressão linear: A regressão linear é utilizada para identificar a relação entre uma variável dependente e uma ou mais variáveis independentes, normalmente é aproveitada para fazer previsões. Para cada tipo de regressão linear, procura- se traçar uma linha de melhor ajuste, que é calculada através do método dos mínimos quadrados [14].
- Regressão logística: A regressão logística é utilizada quando a variável dependente é categórica, é utilizado para determinar se uma entrada pertence a um determinado grupo, através das saídas binárias [15].
- Máquina de vetores de suporte (SVM): Máquina de vetores de suporte cria coordenadas para cada objeto no espaço n-dimensional e usa hiperplanos para agrupar objetos por recursos comuns. Esse hiperplano é conhecido como limite de decisão, separando as classes de pontos de dados em ambos os lados do plano [14].
- Árvores de decisão: Árvore de decisão é um modelo semelhante à árvore de probabilidade que separa continuamente os dados para fazer previsões utilizando um classificador que determina em que categoria uma entrada se enquadra, verificando as folhas e os nós [15].
- k-Nearest Neighbors (k-NN): k-Nearest Neighbors classifica pontos de dados com base na sua proximidade e associação com outros dados disponíveis. É uma técnica que agrupa os objetos mais próximos em um conjunto de dados e encontra o modo ou característica média entre os objetos. A sua facilidade de uso e baixo tempo de cálculo torna-o o algoritmo preferido dos cientistas de dados. Mais usado para mecanismo de recomendação e reconhecimento de imagens [14].

- Random Forest: Random forest é uma coleção de muitas árvores de decisão de um subconjunto aleatório de dados, o que pode resultar em melhor precisão de previsão do que uma única árvore de decisão [15].
- Algoritmos de boosting: algoritmos de boosting, como Gradient Boosting Machine, XGBoost e LightGBM, usam a aprendizagem em conjunto. Uma amostra de dados aleatória é selecionada, ajustada ao modelo e treinada sequencialmente, cada modelo vai tentar compensar os pontos fracos do seu precedente [15].

Este tipo de aprendizagem tem imensas aplicações, como por exemplo classificação de e-mails de spam, deteção de fraude, reconhecimento e classificação de imagens [16].

2.1.2. Aprendizagem Não Supervisionada

Aprendizagem Não Supervisionada usa algoritmos para analisar e fazer *clusters* de *datasets* não identificados (*unlabeled*). Estes algoritmos descobrem padrões escondidos sem necessidade de intervenção humana. A habilidade do método de descobrir esses padrões fá-lo ideal para análise de dados e reconhecimento de padrões, alguns dos métodos utilizados são *k-means*, mistura de modelo gaussiano para fazer *clustering*. Este tipo de aprendizagem tem imensas aplicações, como por exemplo recomendação de produtos, deteção de anomalias, agrupamento de clientes [16].

2.1.3. Aprendizagem por Reforço

Aprendizagem por Reforço é um tipo de aprendizagem automática onde o agente aprende a tomar decisões interagindo com um ambiente e sendo recompensado ou penalizado de acordo com as suas ações. Para traduzirmos a lógica para uma forma que a máquina possa executá-la utilizamos o modelo matemático *Markov Decision Process* (MDP) onde são definido um conjunto de possíveis estados, um conjunto de ações possíveis em cada estado, uma função de recompensa e um modelo de transição que define a probabilidade de acabar num certo estado tendo em conta a ação realizada [17].

2.2. Deep Learning

Deep Learning é uma área do Machine Learning, que se foca no uso de redes neuronais, formadas por múltiplas camadas e parâmetros, para extrair informações úteis a partir dos dados fornecidos [18].

Estas redes organizam as camadas em formato de cascata, ou seja, as camadas mais próximas dos *inputs* aprendem características simples sobre as informações recebidas, enquanto as camadas próximas do *output* vão aprender características mais complexas em virtude das camadas anteriores [18].

O Deep Learning inclui diversos algoritmos altamente complexos e diferentes tipos de redes neuronais, que foram desenvolvidas para responder a diferentes problemas ou conjuntos de dados [19]. Embora cada uma destas redes tenha suas vantagens, compartilham uma limitação comum, são frequentemente consideradas "black boxes" dificultando a compreensão do seu comportamento interno [19].

Entre os tipos de redes mais discutidos no contexto do Deep Learning estão as Convolutional Neural Networks (CNN) e as Recurrent Neural Networks (RNN), que serão explicadas nas próximas secções [18, 19, 20, 21].

2.2.1. Convolutional Neural Networks (CNN)

A primeira rede neuronal convolucional (CNN), conhecida como LeNet, foi apresentada por Yann LeCun e sua equipa no artigo “Gradient-Based Learning Applied to Document Recognition”, publicado em 1998. Esta arquitetura foi utilizada principalmente para tarefas como OCR (Optical Character Recognition) e reconhecimento de caracteres em documentos [18].

Este tipo de redes é uma classe especial de redes neuronais artificiais (ANN) projetadas para lidar de maneira eficiente com tarefas de reconhecimento de padrões em imagens e vídeos [19, 21]. Neste tipo de redes, o objetivo principal é preservar a relação espacial entre os pixéis da imagem, uma coisa

que não é possível fazer de forma eficaz com o uso das redes neuronais artificiais tradicionais, devido ao tratamento de forma isolado dos pixéis. As redes neuronais convolucionais superam essa limitação ao processarem subáreas da imagem de cada vez, possibilitando a preservação do contexto espacial [21].

As redes neuronais convolucionais são formadas por 3 tipos de camadas principais, camada convolucional, camada de pooling e a camada fully-connected (FC) [19, 22]. Em usos com uma complexidade elevada este tipo de redes pode conter até milhares de camadas, que trabalham em conjunto para processar e refinar o *input* inicial. Através do processo de convolução, as camadas iniciais identificam padrões básicos, como cores e bordas, e à medida que o *input* avança pelas camadas seguintes da rede, as características extraídas tornam-se progressivamente mais complexas, permitindo à rede identificar formas maiores e, por fim, reconhecer o objeto pretendido [19].

Para compreender o funcionamento destas redes, é essencial analisar como cada tipo de camada contribui para o processo de extração e processamento de características. (Ver Figura 3. Fonte: [23])

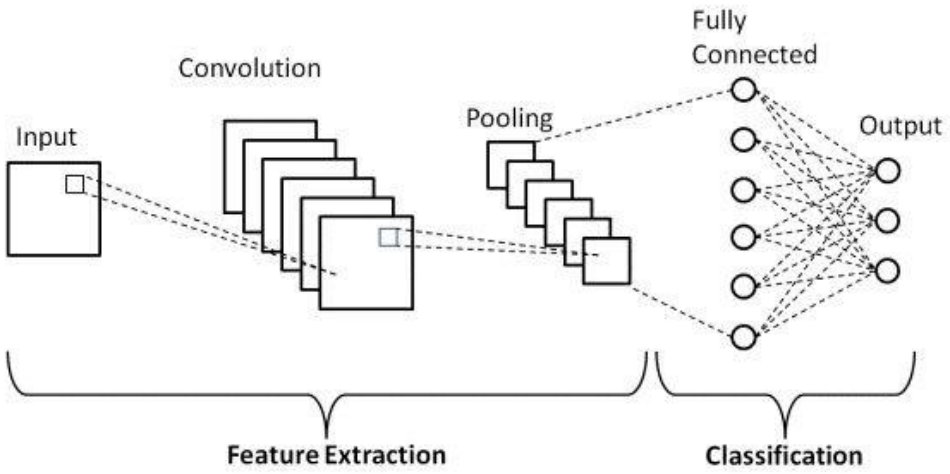

Figura 3- Camadas principais das redes neuronais convolucionais

A camada convolucional é o tipo principal de camadas destas redes, sendo nela onde ocorre a maior parte dos cálculos. Para fazer esses cálculos, ela exige alguns componentes como os dados de entrada, que no caso de uma imagem consistem numa matriz tridimensional representando a altura, largura e profundidade dela; kernel, que percorre a entrada em busca de padrões específicos, como bordas ou texturas; e um mapa de características, que é o resultado das operações realizadas entre a entrada e o filtro. Este processo, conhecido como convolução, permite à rede identificar padrões fundamentais na entrada, que servirão de base para camadas mais profundas [22].

Além disso, várias camadas deste tipo podem ser empilhadas, criando uma estrutura hierárquica onde cada camada subsequente interpreta padrões mais complexos com base nas características extraídas pelas camadas anteriores. Um exemplo deste funcionamento é a deteção de uma bicicleta numa imagem, podemos pensar numa bicicleta como a soma de várias partes, como os pedais, as rodas, entre outras. Cada parte individual da bicicleta constitui um padrão de baixo nível na rede, e a combinação dessas partes representa um padrão de alto nível, criando uma hierarquia de recursos dentro da CNN. Em suma, a camada convolucional converte a imagem em valores numéricos, permitindo à rede neural interpretar e extrair padrões relevantes [22].

Após a camada convolucional, entra em ação a camada de pooling, também conhecida como downsampling. Esta camada realiza uma redução dimensional, reduzindo o número de parâmetros na entrada. Assim como na camada convolucional, a operação de pooling usa um filtro que percorre toda a entrada, mas contrariamente ao comportamento da camada anterior este filtro não tem pesos. Em vez disso, o filtro usa uma função de agregação para simplificar a informação [22].

Existem dois tipos principais de pooling, sendo eles o max pooling e o average pooling. No max pooling, o filtro seleciona o pixel com o valor máximo dentro de uma pequena área da imagem e envia-o para o array de saída. Esse método tende a ser mais utilizado em comparação com o average pooling, onde o filtro calcula o valor médio dentro dessa mesma área para gerar o valor a ser enviado para a saída. Embora muita informação seja perdida durante a operação de pooling, ela traz benefícios significativos para a rede neural convolucional, como a redução da complexidade, a melhoria da eficiência e a mitigação do risco de *overfitting* [22]. (Ver Figura 4. Fonte:[24])

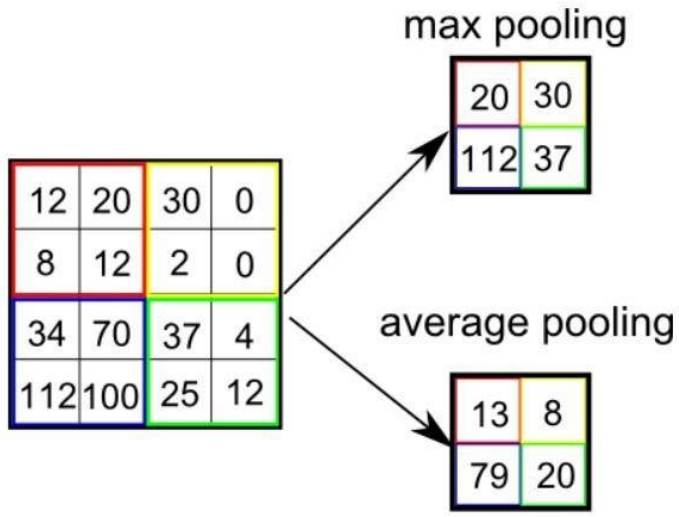

Figura 4- Tipos de pooling das redes neuronais convolucionais

Após a camada de pooling, a próxima etapa do processo é a camada FC. Ao contrário das camadas convolucionais e de pooling, onde os neurónios são conectados apenas a uma parte da entrada, na camada FC, cada nó da camada de saída está diretamente ligado a um nó da camada anterior. Esta camada é responsável pela tarefa de classificação, utilizando as características extraídas pelas camadas anteriores e seus respetivos filtros. Enquanto as camadas convolucionais e de pooling geralmente utilizam funções de ativação ReLU, as camadas totalmente ligadas frequentemente fazem uso da função de ativação softmax, que atribui uma probabilidade ao *input*, variando de 0 a 1, para determinar a classe final da imagem [22].

2.2.2. Recurrent Neural Networks (RNN)

As Redes Neuronais Recorrentes (RNN) são especialmente úteis para recuperar padrões armazenados a partir de versões corrompidas, tendo desempenhado um papel crucial como precursoras das máquinas de Boltzmann e dos *autoencoders*. Em 1986, Michael Jordan introduziu as redes de Jordan, uma das primeiras arquiteturas projetadas para aprendizagem supervisionada em sequências, marcando um marco significativo no desenvolvimento das RNN [18].

RNN são uma classe de redes neuronais profundas projetadas para trabalhar com dados sequenciais ou temporais. Diferente das redes tradicionais, como as redes neuronais *feedforward*, as RNN possuem uma "memória" que lhes permite utilizar informações de entradas anteriores para influenciar as saídas atuais [25]. Podemos ver a diferença entre as duas na Figura 5 (fonte: [26])

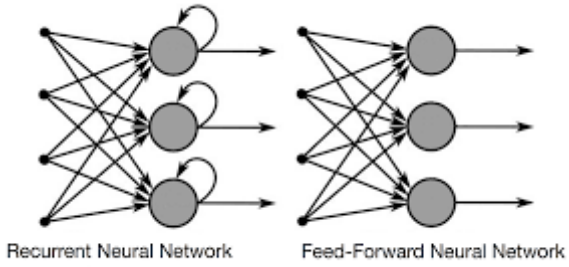

Figura 5- Comparação de RNN e redes neuronais feedforward

Essa capacidade de "lembrar" e integrar informações ao longo do tempo torna as RNN ideais para uma ampla gama de tarefas. Em cada etapa do processamento, a RNN combina a entrada atual com o estado anterior, criando uma saída que reflete tanto a informação nova quanto o contexto armazenado. Essa estrutura de feedback facilita o processamento de dados temporais, sendo extremamente útil em tarefas como tradução de idiomas, análise de sentimentos, reconhecimento de fala e previsão de séries temporais, como níveis de inundação com base em dados meteorológicos e de maré [20, 25].

Outra característica que distingue as RNN dos outros tipos de redes neuronais é o compartilhamento dos parâmetros de peso entre as camadas da rede. Ou seja, os mesmos pesos são aplicados em diferentes etapas do processamento da sequência, o que permite que a rede aprenda com base em dependências temporais. Esses pesos, por sua vez, são ajustados por meio dos processos de retro propagação e descida do gradiente, os quais ajudam na otimização do modelo [25].

Apesar de as RNN serem redes dinâmicas e eficazes, elas enfrentam um grande desafio durante o treino: os gradientes retro propagados podem se amplificar ou diminuir a cada passo. Isso leva a problemas como o desaparecimento ou explosão do gradiente. Com o tempo, esses gradientes podem desaparecer ou crescer de maneira descontrolada, dificultando a captura de dependências de longo prazo em sequências e comprometendo a aprendizagem da rede [20].

As Long Short-Term Memory (LSTM) e as Gated Recurrent Units (GRU) são variantes avançadas das RNN, desenvolvidas para resolver o problema do desaparecimento do gradiente e melhorar a capacidade de lidar com dependências de longo prazo. As LSTMs, criadas por Sepp Hochreiter e Juergen Schmidhuber, possuem “células” nas camadas ocultas com três portas (entrada, saída e esquecimento) que controlam o fluxo de informações essenciais para a previsão, permitindo que a rede retenha ou descarte informações conforme necessário [25].

Por outro lado, as GRUs têm uma arquitetura mais simples, com apenas duas portas: uma de reinicialização e outra de atualização. Embora as GRUs sejam mais eficientes computacionalmente, com menos parâmetros e um tempo de treino mais rápido, ambas as arquiteturas resolvem as limitações das RNN tradicionais, tornando-as mais eficazes em tarefas que exigem memória de longo prazo e comumente usadas em contextos de tempo real ou recursos limitados [25].

2.2.3. Visão Computacional

A Visão computacional, ramo do deep learning que procura habilitar máquinas a interpretar e compreender dados visuais, apresenta uma trajetória marcada por avanços significativos ao longo das últimas seis décadas. Desde as primeiras experiências neurofisiológicas em 1959, que revelaram como cérebros processam formas simples, até o desenvolvimento de tecnologias como a digitalização de imagens e o reconhecimento ótico de caracteres (OCR) na década de 1970, esses avanços foram marcos importantes para o desenvolvimento da área [27].

Os avanços nas décadas seguintes, como o surgimento do Neocognitron em 1982 e o desenvolvimento de algoritmos para reconhecimento de padrões, culminaram em inovações transformadoras nos anos 2000. A criação do *dataset* ImageNet, em 2010, e o sucesso do modelo

AlexNet em competições de reconhecimento de imagens em 2012 consolidaram a aplicação de redes neuronais convolucionais (CNN), reduzindo drasticamente taxas de erro e abrindo caminho para as soluções de Visão computacional modernas [27].

Com esse histórico de inovações, a Visão computacional consolidou-se como um dos campos mais promissores da inteligência artificial. Atualmente, é um campo da inteligência artificial (IA) que utiliza machine learning e redes neuronais para habilitar computadores e sistemas a extraírem informações relevantes de imagens digitais, vídeos e outros dados visuais, permitindo-lhes tomar decisões ou ações com base em padrões identificados [27]. Inspirada no funcionamento da visão humana, mas utilizando câmeras, Visão computacional permite que máquinas processem e analisem milhares de imagens por minuto, superando rapidamente as capacidades humanas na identificação de defeitos e padrões impercetíveis [27].

Esse processo baseia-se em etapas fundamentais, como aquisição e pré-processamento de imagens, extração de características e reconhecimento visual. Imagens capturadas por câmeras são convertidas em sinais digitais e submetidas a técnicas de redução de ruído, ajuste de contraste e redimensionamento, preparando-as para análises mais detalhadas. Métodos como deteção de bordas, análise de textura e reconhecimento de padrões transformam informações complexas em vetores de características simplificados, permitindo que sistemas identifiquem objetos ou cenários com elevada precisão [28]. Atualmente, a Visão computacional é amplamente aplicada em setores como manufatura, automotivo, energia e utilidades, demonstrando potencial crescente [27].

A visão computacional tem avançado significativamente, principalmente devido à integração de modelos estocásticos e heurísticos que simulam o reconhecimento visual humano. Esses modelos combinam processamento sequencial e paralelo, imitando parcialmente a arquitetura do cérebro humano, o que possibilita a extração de padrões básicos e a compreensão de cenas tridimensionais complexas. Esses avanços abrem novas perspetivas para aplicações práticas, como a análise de cenas dinâmicas e a interpretação visual em tempo real [29].

Com base nesses avanços, a visão computacional pode ser subdividida em várias tarefas específicas, cada uma com seu próprio foco e aplicação. Estas tarefas variam desde a simples classificação de imagens até o rastreamento de objetos em movimento, e cada uma delas desempenha um papel crucial em diferentes contextos, como segurança, indústria e saúde. A seguir, exploraremos alguns dos principais tipos de tarefas em visão computacional, destacando as suas aplicações e a importância no desenvolvimento de soluções inteligentes [27].

A classificação de imagens é uma tarefa fundamental em visão computacional, onde um sistema é capaz de identificar e categorizar o conteúdo de uma imagem. Este processo é crucial para aplicações como a moderação de conteúdo em plataformas de redes sociais, onde pode ser utilizado para identificar e separar automaticamente imagens que contenham conteúdo indesejável [27].

A deteção de objetos vai além da classificação, permitindo que o sistema não apenas reconheça um objeto em uma imagem, mas também localize e categorize diferentes instâncias desse objeto dentro da cena. Essa tarefa é amplamente utilizada em áreas como a indústria, onde pode ser empregue para identificar danos em produtos em uma linha de montagem ou detetar máquinas que necessitam de manutenção. A deteção de objetos, portanto, ajuda a automatizar processos e a melhorar a eficiência na produção [27].

O rastreamento de objetos é uma extensão da deteção, em que o sistema acompanha um objeto ao longo do tempo, normalmente em sequência de imagens ou vídeos em tempo real. Esse tipo de tarefa é crucial para sistemas como os de veículos autônomos, que precisam não só identificar objetos como pedestres, outros veículos e estruturas viárias, mas também monitorizá-los em movimento para evitar colisões e seguir as leis de trânsito. A capacidade de rastrear objetos em movimento é vital para a segurança e a navegação eficiente desses sistemas [27].

A recuperação de imagens baseada em conteúdo usa a visão computacional para pesquisar e recuperar imagens de grandes bases de dados baseando-se no seu conteúdo visual, ao invés de depender apenas de tags ou metadados. Esse processo pode incluir a anotação automática das imagens, substituindo a necessidade de marcação manual. A recuperação de imagens é particularmente útil em sistemas de gestão de ativos digitais, pois permite que as imagens sejam localizadas com maior precisão, melhorando a organização e a acessibilidade dos dados visuais [27].

Essas tarefas são apenas algumas das várias aplicações que a visão computacional pode oferecer, e sua implementação tem o potencial de transformar uma ampla gama de setores, desde a indústria até a mobilidade urbana.

2.2.4. You Only Look Once (YOLO)

O algoritmo YOLO (You Only Look Once) representa uma das maiores inovações no campo da visão computacional, particularmente na tarefa de deteção de objetos em imagens. A sua proposta baseia-se numa ideia simples, mas poderosa, que em vez de utilizar abordagens complexas compostas por múltiplas etapas, onde primeiro são escolhidas várias zonas da imagem que podem conter objetos (as chamadas propostas de regiões) e depois são analisadas cada uma separadamente com um classificador para identificar o objeto dentro delas, o YOLO trata a deteção como um problema de regressão direta sobre a imagem inteira, ou seja, em vez de procurar por regiões específicas e classificá-las uma a uma, o modelo aprende a prever, de forma contínua e num único passo, as coordenadas espaciais das caixas delimitadoras e as respetivas classes dos objetos diretamente a partir dos dados da imagem. Um único modelo neural é responsável por identificar simultaneamente o que está presente na imagem e onde está localizado [30].

A primeira versão do YOLO foi apresentada por Joseph Redmon e colegas em 2016, com o objetivo de superar limitações de métodos anteriores como R-CNN (Regional-based Convolutional Neural Network) e DPM (Deformable Parts Model), que eram mais lentos e exigiam pipelines complicados. O YOLO simplificou todo esse processo ao propor uma arquitetura unificada e treinável de ponta a ponta, ou seja, um modelo em que todas as etapas da deteção, desde a leitura da imagem até à previsão final das localizações e categorias dos objetos, são integradas num único sistema que pode ser treinado de forma conjunta. Esta abordagem permite uma otimização mais eficiente do desempenho, sem necessidade de separar ou treinar componentes individualmente. O YOLO é assim capaz de realizar deteção em tempo real a uma velocidade de até 45 FPS (frames por segundo), com uma versão ainda mais rápida a atingir 155 FPS [30].

Uma das grandes vantagens deste modelo é a sua capacidade de fazer inferência global sobre a imagem, ou seja, analisar a imagem como um todo em vez de olhar apenas para pequenas partes isoladas. Esta abordagem permite ao modelo compreender o contexto completo e a relação entre os objetos e o fundo, reduzindo erros como falsas deteções em áreas que não têm objetos importantes. Por isso, o modelo generaliza melhor para diferentes situações e ambientes. Isto faz do YOLO uma escolha popular em aplicações práticas como vigilância, veículos autónomos e robótica [30].

Ao longo do tempo, foram desenvolvidas diversas versões sucessoras do YOLO, cada uma com o objetivo de aumentar a precisão, reduzir a complexidade computacional ou melhorar a robustez em cenários variados. Essas versões introduziram melhorias técnicas em áreas como arquitetura da rede, normalização, uso de diferentes tipos de conexões e estratégias de treino. Apesar de diferenças específicas entre as versões, todas mantêm a estrutura base original: dividir a imagem numa grelha e prever múltiplas caixas delimitadoras e classes por célula [26, 27].

À medida que novas versões foram surgindo, como o YOLOv2, YOLOv3 até YOLOv7, o algoritmo foi evoluindo para incluir mecanismos mais sofisticados e adaptáveis, respondendo às necessidades de diferentes contextos de aplicação, desde dispositivos móveis até grandes servidores de dados. Algumas versões foram desenvolvidas por comunidades independentes, mantendo a filosofia do YOLO, mas com variações em frameworks e métodos [26, 27].

Em resumo, o YOLO é hoje reconhecido como um marco na evolução da deteção de objetos por conseguir combinar eficiência, velocidade e precisão. A sua abordagem simples, mas eficaz permitiu romper com paradigmas anteriores e estabelecer novos caminhos para a inteligência artificial aplicada à perceção visual [25, 26, 27].

2.3. Métrica de Avaliação de Modelos

A avaliação de modelos de DL, como os baseados na arquitetura YOLO, é essencial para garantir a eficácia em aplicações reais. Após o treino, o modelo deve ser validado com dados não vistos durante o treino, esta divisão visa evitar *overfitting* e *underfitting.*

Overfitting ocorre quando um modelo apresenta elevada precisão nos dados de treino, mas desempenho significativamente inferior em dados de teste, indicando que o modelo memorizou os dados de treino em vez de aprender padrões generalizáveis. Em conjuntos de dados pequenos, onde os erros ou inconsistências do conjunto de dados de treino são vistos como padrões válidos, levam o modelo a aprender incorretamente que esses são padrões importantes. Em conjuntos de dados muito grandes, a dispersão dos dados ou alta variabilidade pode dificultar a identificação de padrões [33].

Underfitting ocorre quando o modelo apresenta um desempenho insatisfatório nos dados de treino e teste porque não conseguiu capturar padrões relevantes, normalmente ocorre em modelos muito simples ou regularização excessiva que restringe a flexibilidade do modelo, pré-processamento inadequado ou tempo insuficiente de treino também pode causar *underfitting* [33].

Na Figura 6 podemos ver melhor a diferença entre *overfitting, underfitting* e um ajuste correto [34].

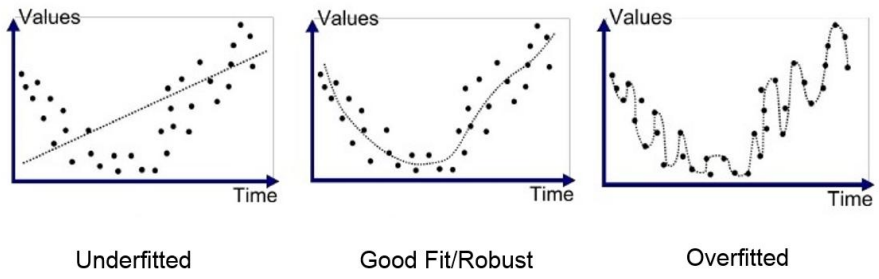

Figura 6- Overfitting, Underfitting e Good Fit

Dado que existem várias métricas e que estas estão associadas ao tipo de problema, apenas serão apresentadas aqui as métricas de avaliação para deteção de objetos, uma vez que é o problema que se pretende resolver no âmbito deste projeto. Portanto, as métricas utilizadas para este problema são a precisão, *recall* e mAP (*mean Average Precision*).

Para uma melhor compreensão de como são calculadas estas métricas primeiro é fundamental analisar a matriz de confusão da qual estas métricas derivam. A matriz de confusão permite visualizar de forma mais compreensiva a performance de um modelo na fase de teste, comparando os valores previstos pelos valores reais do *dataset* [35]. Esta matriz categoriza as predições em quatro tipos possíveis de resultados que são [36] (para melhor compreensão ver o diagrama na Figura 7):

- Verdadeiros Positivos (VP): o modelo prevê que uma instância pertence a uma classe e ela realmente pertence a essa classe.
- Verdadeiros Negativos (VN): o modelo prevê que a instância não pertence a uma classe e ela realmente não pertence a essa classe.
- Falsos Positivos (FP): o modelo prevê que a instância pertence a uma classe, mas ela não pertence a essa classe.
- Falsos Negativos (FN): o modelo prevê que a instância não pertence a uma classe, mas na realidade ela pertence.

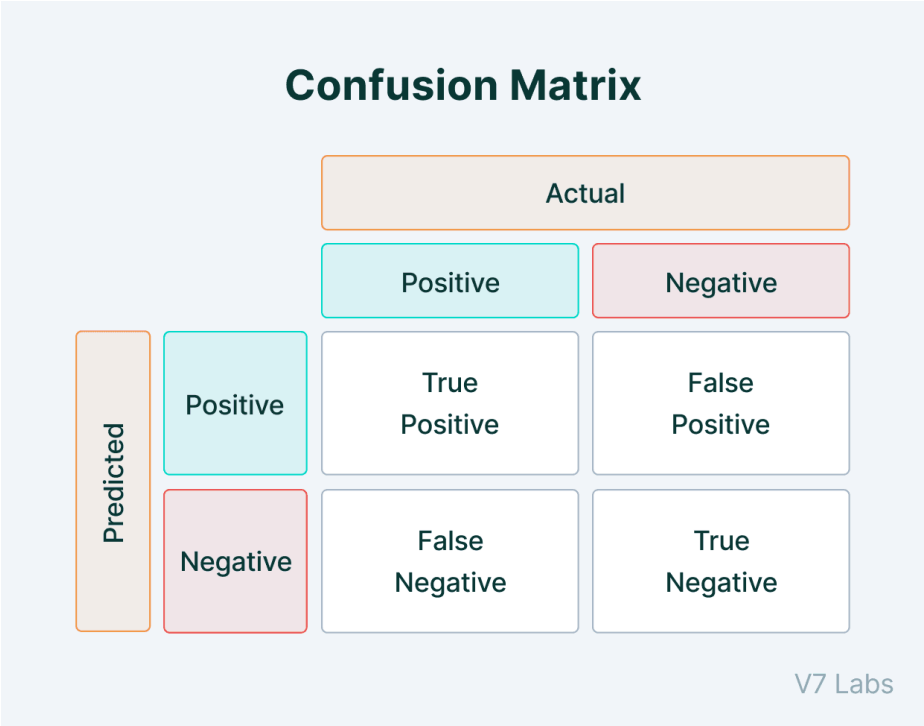

Figura 7- Matriz de confusão

A partir destes componentes podemos calcular a Precisão, *Recall* e mAP, que permitem avaliar a robustez e eficácia do modelo.

A Precisão é uma métrica que se concentra na precisão das previsões positivas. Vai calcular a proporção de instâncias positivas previstas corretamente em relação ao total de instâncias previstas como positivas, como podemos ver na Equação 1 [37].

$$Precisão = \frac{VP}{VP + FP}$$

Equação 1- Cálculo da precisão

O *Recall* mede a capacidade do modelo em encontrar todos os exemplos positivos. Calcula a proporção de instâncias positivas previstas corretamente em relação ao total de instâncias positivas reais, como podemos ver na Equação 2 [37].

$$Recall = \frac{VP}{VP + FN}$$

Equação 2- Cálculo do recall

A Figura 8 permite entender melhor a diferença entre Precisão e *Recall* [38]. Como podemos ver a precisão tem em consideração todas as previsões positivas para uma determinada classe, enquanto o *recall* considera todas as previsões corretas para a classe em questão com o intuito de saber apenas as que são verdadeiras positivas.

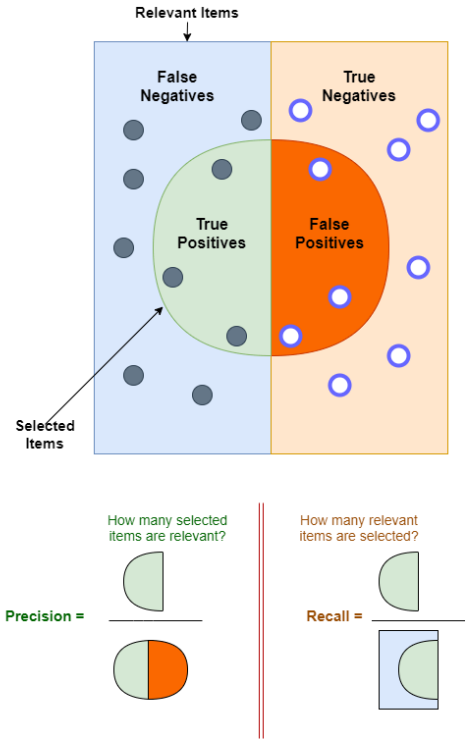

Figura 8- Exemplo de Precisão e Recall

O mAP incorpora o equilíbrio entre a precisão e o *recall* e considera tanto falsos positivos quanto falsos negativos. Essa propriedade torna o mAP uma métrica adequada para a maioria das aplicações de deteção. É calculado encontrando a *Average Precision* (AP) para cada classe e fazer a média pelo número de classes, como podemos ver na Equação 3, N representa o número de classes e APi representa o AP da classe i. [36].

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

Equação 3- Cálculo do mAP

3. Revisão Sistemática

Para fazer a pesquisa do estudo do estado da arte sobre os projetos que já foram feitos na área de deteção de pessoas em cenários de desastre usando visão computacional utilizámos a metodologia PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses). O foco nesta pesquisa foi analisar o que já tinha sido desenvolvido de forma a podermos analisar os modelos e *datasets* já utilizados para ver qual se adequa melhor ao que pretendemos desenvolver e de que forma podemos arranjar novas soluções ou melhorar as já existentes. De acordo com esta metodologia foram seguidos os seguintes elementos:

- Fontes usadas
- Estratégia de pesquisa
- Critérios de inclusão
- Critérios de exclusão
- Extração e análise de dados
- Conclusão

3.1. Fontes Usadas

A pesquisa de trabalhos e artigos nesta área foi feita maioritariamente na Scopus. A Scopus é uma plataforma que combina resumos científicos e citações numa base de dados permitindo assim o acesso a um número vasto de artigos científicos em diversas áreas. É possível fazer uma pesquisa avançada permitindo a configuração de strings de pesquisa usando palavras-chave, também permite a utilização de operadores lógicos, como o AND, OR e NOT, na construção da string de pesquisa.

3.2. Estratégia de Pesquisa

Primeiro fizemos algumas pesquisas na Scopus usando diferentes strings de pesquisa para tentar encontrar as que nos mostravam mais artigos sobre o tema em que estamos a trabalhar, concluímos que a melhor string de pesquisa seria (”human detection” AND disasters AND “machine learning” OR “deep learning”). A pesquisa foi feita dia 11 de novembro de 2024 e obtivemos 27 resultados.

3.3. Critérios de Inclusão

Para fazer a inclusão dos artigos para análise aplicámos alguns filtros (1) apenas artigos publicados entre 2018 e 2024, (2) apenas artigos em inglês ou português. Depois de usarmos estes filtros permaneceram para análise 24 artigos.

3.4. Critérios de Exclusão

Os critérios de inclusão levaram à identificação de 24 artigos. Destes 24 foram excluídos 6 após a leitura do título e outros 6 após a leitura do resumo, permanecendo assim 12 para analisarmos. (Ver Tabela 2)

Tabela 2- Artigos encontrados na pesquisa

Ano	Título	Análise do Título (S/N)	Análise do resumo (S/N)
2023	UAV based Human Detection for Search and Rescue Operations in Flood	Sim	Sim
2023	Low-Power Embedded System for Deep Learning-based Object Detection and Tracking in River Systems	Sim	Sim

2020	A fusion of visible and infrared images for victim detection	Sim	Não
2024	An efficient No-Line-of-Sight learning approach for Prediction of Static Victim Detection Using Genetic Algorithm and Kth Nearest Neighbor Data Classification	Sim	Não
2019	Drone based near real-time human detection with geographic localization	Sim	Não
2019	Gi4DM 2019 - GeoInformation for Disaster Management	Sim	Não
2023	Real-time human search and monitoring system using unmanned aerial vehicle	Sim	Sim
2020	Drone-surveillance for search and rescue in natural disaster	Sim	Sim
2023	Autonomous Navigation System for Multi-Quadrotor Coordination and Human Detection in Search and Rescue	Sim	Sim
2023	A Comparative Performance Study of Support Vector Machine, KNN, and Ensemble Classifiers on through-wall human detection Dataset	Sim	Não
2022	Training a Disaster Victim Detection Network for UAV Search and Rescue Using Harmonious Composite Images	Sim	Sim
2024	PDD: Post-Disaster Dataset for Human Detection and Performance Evaluation	Sim	Sim
2023	IR-UWB radar based through-wall human detection with limited data via multi-hop residual CNN	Não	Não
2024	Human Detection In Search And Rescue Operations Using Embedded Artificial Intelligence	Sim	Sim
2024	Smart and Safe Conveyor Belt System for Bottling Plants Using IoT	Não	Não
2018	Indoor trapped-victim detection system	Não	Não
2024	Human Detection From Unmanned Aerial Vehicles' Images for Search and Rescue Missions: A State-of-the-Art Review	Sim	Sim
2024	Enhancing unmanned ground vehicle performance in SAR operations: integrated gesture-control and deep learning framework for optimized victim detection	Não	Não
2024	YOLO-IHD: Improved Real-Time Human Detection System for Indoor Drones	Sim	Sim

2024	Trapped human victims Machine Learning detection Model Using Recursive Feature Elimination with SVM Classification Algorithm	Sim	Não
2023	Raspberry Pi Alive Human Detection Robot Using PIR Sensor	Não	Não
2022	RescueNet: YOLO-based object detection model for detection and counting of flood survivors	Sim	Sim
2024	Internet of Things Assisted Human Detection and Rescue Robot: A Game Changer in the Disaster Environment	Sim	Sim
2018	Through wall human detection under small samples based on deep learning algorithm	Não	Não

Desses 12 não conseguimos encontrar o texto integral de 2 deles, pelo que apenas 10 passaram à fase seguinte. Depois de lermos esses 10 artigos excluímos 5 artigos, usámos como critério de exclusão: (1) o foco do artigo era em construção de UAVs (*Unmanned Aerial Vehicle*), (2) foco no funcionamento em ambientes de simulação 3D ou interior (3) *dataset* utilizado não coincide com o que estamos a procurar. Com a aplicação destes critérios apenas 5 artigos passaram à fase seguinte.

3.5. Extração e Análise de Dados

Para além dos 5 artigos que encontrámos no Scopus analisámos mais 3 que foram encontrados no kaggle ou foram referenciados em reviews, ficando assim com um total de 8 artigos para analisar. Destes artigos retirámos informação que considerámos pertinente como o (1) Ano, (2) *Dataset* usado e se criou o *dataset* onde foi buscar as imagens, (3) Quantos dados tem o *dataset*, (4) Modelo usado, (5) Resultados, (6) Opensource. A lista de artigos pode ser vista na Tabela 3.

Tabela 3- Artigos analisados

Ref.	Título
[39]	UAV based Human Detection for Search and Rescue Operations in Flood
[40]	Training a Disaster Victim Detection Network for UAV Search and Rescue Using Harmonious Composite Images
[41]	PDD: Post-Disaster Dataset for Human Detection and Performance Evaluation
[42]	Human Detetcion In Search And Rescue Operations Using Embedded Artificial Intelligence
[43]	RescueNet: YOLO-based object detection model for detection and counting of flood survivors
[44]	Implementation of Victims Detection Framework on Post Disaster Scenario
[45]	Saving Lives From Above: Person Detection In Disaster Response Using Deep Neural Networks
[46]	UAV-Enhanced Combination to Application: Comprehensive Analysis and Benchmarking of a Human Detection Dataset for Disaster Scenarios

O artigo [39] de Samta Gaur e J Sathish Kumar foi publicado em 2023, este artigo propõe a utilização de drones equipados com câmeras de alta resolução para realizar buscas em áreas afetadas por desastres naturais, como inundações. O *dataset* que utilizaram foi o COCO (Common Objects in Context) e o modelo que utilizaram foi o YOLOv4 (You Only Look Once), para o treinar usaram o Darknet com o backbone CSPDarkNet53, rede convolucional opensource desenvolvida em C++ com 162 camadas e as deteções são feitas nas camadas 139, 150 e 161. O backbone captura características hierárquicas das imagens, o software que usaram para analisar os testes foi o Google Colab. Os testes foram feitos com um conjunto de dados personalizados, composto por 20 imagens com humanos e 50 imagens com objetos não-humanos, os resultados foram de 79,46% de precisão com deteção de 89 dos 112 humanos. Os autores não mencionam explicitamente se disponibilizam o código e o *dataset* utilizados no estudo.

O artigo [40] de Ning Zhang , Francesco Nex , George Vosselman e Norman Kerle foi publicado em 2022, este artigo propõe a criação de imagens para treino onde são visíveis apenas partes do corpo e uma rede de harmonização que vai ajudar a tornar essas imagens mais realistas. Foi criado um *dataset* próprio e as imagens que usaram para fazer a composição foram retiradas dos *datasets* COCO e LIP (Look into Person). Para criar estas imagens foi usado o Ubuntu 18.04 com o framework PyTorch, para treinar a rede de harmonização das imagens usam 85 fundos e 1500 imagens com pessoas para gerar um total de 3000 imagens harmonizadas com o tamanho de 512X512, as imagens foram anotadas usando 19 classes pré-definidas como left/right arm, left/right leg, e torso. Mais tarde as classes que envolvem partes do corpo vão ser fundidas em 5 partes do corpo, upper limbs, upper limbs + torso, lower limbs, lower limbs+ torso, e corpo inteiro. As classes para a roupa são fundidas com as 5 classes das partes do corpo. Para fazer o treino do *dataset* primeiro escolheram modelos que já tinham sido

treinados com o COCO, FCOS (Fully Convolutional One-Stage Object Detection), TTFNet (Training-Time-Friendly Network for Real-Time Objetcion Detection), PAFNet (Paddle Anchor Free Network) e YOLOv5, para os primeiros 3 foram usadas as implementações do PaddleDetection e para o YOLO foram usadas as próprias implementações. Com os testes podemos verificar que o modelo com melhores resultados foi o YOLOv5 mais especificamente o YOLOv5l. Depois disto ainda são feitos mais treinos alterando alguns parâmetros com o intuito de melhorar o modelo acabando por ter uma precisão média de 44,9%. O *dataset* e o código podem ser encontrados no seu website [46, 47].

O artigo [41] de Haoqian Song , Weiwei Song , Long Cheng, Yue Wei e Jinqiang Cui foi publicado em 2024, este artigo propõe a avaliação de modelos já existentes e a criação de um novo *dataset* , PDD (Pos-Disaster Dataset), que ajude a resolver o problema dos *datasets* existentes não serem realistas o suficiente. As imagens usadas no *dataset* são de diferentes fontes, 2316 de ruínas reais e 1144 de partes do corpo. O *dataset* tem imagens de diferentes distâncias e resoluções, as anotações em cada imagem foram feitas manualmente, no final para treino ficaram 832 imagens com corpos enquanto a validação e o teste terão 100 imagens cada, cada imagem tem o tamanho de 640X640. Os modelos estudados foram o Faster R-CNN (Regional-based Convolutional Neural Network), YOLOv7, YOLOv8, YOLO-NASm (You Only Look Once Neural Architecture Search), im-YOLOv5 (Improved YOLOv5), DETR (Detection Transformer), DDETR (Deformable DETR), DAB-DETR (Dynamic Anchor Box DETR), DN-DETR (Denoising DETR), DINO (DETR with Improved Denoising Achor Box). Os modelos YOLO e Faster R-CNN são treinados por 300 épocas, o DETR e DDETR são treinados por 500 épocas e os DAB-DETR, DN-DETR e DINO são treinados por 100 épocas. De forma geral o YOLO-NASm é o melhor para detetar sobreviventes e tem uma taxa de falha melhor, puderam verificar também que os modelos YOLO têm melhor precisão em detetar humanos enquanto os modelos DETR têm mais precisão na caixa delimitadora. O *dataset* e código estão disponíveis no seu github [49].

O artigo [42] de Ahmed Abdullah Hussein Al-azzani, Mohd Ridzuan Ahmad foi publicado em 2024. Este artigo propõe a utilização de drones para deteção de pessoas em cenários de desastre. Para o *dataset* usaram imagens do SARD (Software Assurance Reference Dataset) e *datasets* públicos do SeaDroneSee, utilizaram o Roboflow para fazer as anotações das imagens, as anotações para as pessoas são *seated, laying down, walking, swimmer* e *boats*, as imagens têm o tamanho de 640X640. Os modelos utilizados foram MobileNetv2 e EfficientDet clonados do TensorFlow, para detetar as pessoas sem usar conexões externas foi usado o TinyML. Para treinar os modelos foi usado o Google Colaboratory. Com os testes puderam verificar que o MobileNet é a melhor opção com uma precisão de 90%. Os autores não mencionam explicitamente se disponibilizam o código e o *dataset* utilizados no estudo.

O artigo [43] de B. V. Balaji Prabhu foi publicado em 2022. Este artigo propõe a utilização de drones equipados com câmeras para realizar buscas em áreas afetadas por inundações. O *dataset* utilizado foi criado pelos autores, contendo 970 imagens nos formatos .jpg e .png, este *dataset* deteta pessoas e animais. As imagens foram obtidas de diversas fontes, incluindo o UAV Laboratory do Departamento de Engenharia Aeroespacial do Indian Institute of Science, Bangalore. O modelo utilizado foi uma versão modificada do YOLO, denominada RescueNet, que eles próprios criaram, treinada especificamente para detectar e contar pessoas e animais em situações de inundação. Para o treino, 873 imagens (90% do *dataset*) foram anotadas usando a ferramenta "LabelImg", enquanto as 97 imagens restantes foram reservadas para testes, inicialmente as imagens tinham o tamanho de 416X416, mas foram reduzidas para 208X208 para facilitar a deteção precisa das pessoas e animais. Os resultados obtidos com a implementação do modelo proposto demonstraram a sua eficácia na deteção e contagem de pessoas e animais afetados. O modelo alcançou um mAP (mean Average Precision) de 98%. Os autores não mencionam explicitamente se disponibilizam o código e o *dataset* utilizados no estudo.

O artigo [44] de Indra Adji Sulistijono foi publicado em 2018. Este artigo propõe a utilização de drones equipados com câmeras e GPS para realizar buscas em áreas afetadas por desastres. O *dataset* utilizado foi uma versão modificada do PASCAL VOC (Visual Object Classes) 2012, contendo apenas a classe "pessoa" e criando cópias modificadas das imagens para representar posições diversificadas de

vítimas. Os autores utilizaram uma versão modificada do modelo ResNet50 como base para o bloco convolucional da sua arquitetura, adaptando-o especificamente para a deteção e contagem de pessoas em cenários pós-desastre. Para o treino, foram utilizadas 70 épocas com 1000 iterações cada, totalizando 70.000 passos. A arquitetura combinava o ResNet50 com uma Rede de Propostas Regionais (RPN) e foi treinada em uma versão modificada do *dataset* PASCAL VOC. Os testes foram realizados em um ambiente simulado usando o software V-REP (Virtual Robot Experimentation Platform), bem como em imagens com fundos complexos e vítimas reais. Os resultados mostraram que o modelo é capaz de detetar vítimas em cenários complexos, com alguns falsos positivos e negativos, utilizando um limiar de confiança de 80%. O sistema foi testado em altitudes variando de 5,4 a 26,9 metros, demonstrando eficácia, embora com limitações em altitudes mais elevadas. Os autores não mencionam explicitamente se disponibilizam o código e *dataset* utilizados no estudo.

O artigo [45] de Reza Bahmanyar , Nina Merkle foi publicado em 2023. O *dataset* usado foi um criado pelos próprios com 311 imagens que foram separadas entre 259 imagens com 6934 anotações para treino, 25 imagens com 2706 anotações para validação e 27 imagens com 410 anotações para teste, as imagens utilizadas foram adquiridas por drones 4K e MACS montados em helicópteros e drones, contém imagens de diferentes cenários e diferentes regiões como Alemanha, Holanda, Suíça, Espanha, França e Nepal. O tamanho das imagens varia entre 4864X3232 px, 5184X3456 px, 5616X3744 px e 8000X6000 px. O modelo utilizado foi o YOLOv3 pré-treinado no *dataset* MS COCO, Darknet53 como backbone, ResNet101 e ResNet152 para deteção de objetos. Também usaram o ADAM (Adaptive Moment Estimation) optimizer que é um algoritmo utilizado para reduzir as perdas durante o treino de redes neuronais. O modelo alcançou uma precisão de 60%. Os autores não mencionam explicitamente se disponibilizam o código e o *dataset* utilizados no estudo.

O artigo [46] de Ragib Amin Nihal, Benjamin Yen, Katsutoshi Itoyama e Kazuhiro Nakadai foi publicado em 2024. Este artigo propõe a criação de um novo *dataset* para ajudar nas operações SAR (Search and Rescue) utilizando UAV. O *dataset* criado combina partes de AIDER(Aerial Image Dataset for Emergency Response Applications) e LSP/MPII-MPHB (Leeds Sports Pose/ Multiple Poses Human Body) *datasets*, foram retiradas imagens de cenários e de poses humanas respetivamente, usaram o U2Net (U-Squared Net), rede neuronal conhecida pela sua performance em deteção de objetos, para fazer a composição das imagens e ficaram no total com 10215 imagens e 360000 objetos humanos, chamaram-lhe C2A (Combination to Application), as anotações são *bent, kneeling, lying, sitting, upright* para os objetos e *traffic incident, fire, flood, e collapsed building* para os cenários de desastre. O tamanho das imagens varia de 123X152 e 5184X3456, sendo o tamanho mais comum 322 e 600 px, o tamanho dos objetos varia entre >10 e 300px, o mais comum é de 10-50px e o menos comum é de 50-300px, que apenas corresponde a 1%. Os modelos utilizados foram Faster R-CNN, RetinaNet, Cascade R-CNN, Dino, Rtmdet, YOLOv5, YOLOv9-c (YOLOv9 compact), YOLOv9-e (YOLOv9 extended). A avaliação dos modelos foi feita usando NVIDIA A100 GPUs com uma resolução de 640X640 por 50 épocas, foi usado o ADAM optimizer e os frameworks mmDetection, Detectron2, Ultralytics. O modelo com melhor desempenho foi o YOLOv9-e com um mAP de 68,83% na habilidade de identificar humanos. Os autores disponibilizam o código e *datasets* no seu github e no kaggle [50].

3.6. Análise dos *Datasets*

Na maioria dos artigos foram criados *datasets* [39, 40, 41, 42, 43, 44, 45], os *datasets* já existentes apenas num caso foram usados para testar um modelo [39]. As imagens para criar os *datasets* foram retiradas de vários domínios públicos e até mesmo dos *datasets* já existentes (COCO, AIDER, LSP/MPII-MPHB, PASCAL-VOC, SARD, SeaDroneSee, LIP).

Os *datasets* que foram criados pelos autores foram feitos fazendo a harmonização de imagens de cenários de desastres e imagens de pessoas em diferentes situações, em diferentes posições ou apenas partes do corpo. Depois da harmonização foram retiradas as imagens com pouca qualidade ou o tamanho dos objetos, as pessoas e em um caso animais [43], era demasiado pequeno. Em alguns casos ainda foi tido em conta o tipo de cenário, por exemplo edifícios colapsados, onde os objetos podem

estar cobertos por pó o que pode tornar mais difícil a sua deteção. De todos os *datasets* mencionados apenas conseguimos acesso ao *dataset* e código de alguns artigos [39, 40, 45].

O *dataset* [47] do artigo [40], que podemos aceder a partir do github [48], foi criado usando um pipeline que consiste em 3 passos. Recolher imagens de cenários de terramotos e edifícios colapsados, depois recortou-se partes do corpo de imagens de pessoas e foram coladas nas imagens dos cenários obtendo assim uma imagem composta. Depois as imagens compostas foram harmonizadas e mais tarde foram utlizadas para aperfeiçoar um detetor pré-treinado. Neste *dataset* o foco foi a identificação de pessoas parcialmente cobertas por escombros, para obter as imagens das pessoas foi utilizado o LIP *dataset.* O LIP contém 50000 imagens de pessoas selecionadas do *dataset* COCO. Cada imagem estava anotada usando 19 classes pré-definidas como *left/right arm*, *left/right leg,* e *torso,* foram eliminadas as imagens com pouca resolução, desfocadas, monocromáticas e imagens sem partes do corpo expostas. As classes com partes do corpo foram fundidas em 5 classes *upper limbs, upper limbs + torso, lower limbs, lower limbs + torso,* e *full body.* As classes relacionadas com a roupa foram fundidas com estas 5 classes. As imagens compostas foram geradas usando a Equação 4:

$$(I^c = I^b \times (1 - M^f) + I^f \times M^f)$$

Equação 4- Criação de uma imagem composta

Dado um fundo da imagem I^b, uma imagem em primeiro plano I^f, e a máscara binária M^f correspondente à parte do corpo na imagem em primeiro plano, geramos a imagem composta I^c.

De seguida é usado uma rede de harmonização com um mecanismo de *self-attention* para que não haja tanta diferença de luminosidade, cor e textura das características entre os cenários e as partes do corpo nas imagens. No final o set de treino é composto por 3000 imagens compostas, 85 cenários e 1500 partes do corpo, com o tamanho de 512X512. O set de validação e teste é composto por 250 imagens de vítimas reais, estas imagens são divididas em 3 subsets, um deles é usado para a validação após o treino e os outros dois são utilizados para teste.

O *dataset* do artigo [41] encontra-se no github [49] e o código dos modelos encontra-se no github [51]. Este *dataset* foi criado a partir de imagens recolhidas da internet e métodos de coleção offline, de 2316 imagens de ruínas pós-desastre foram selecionadas 1144 que continham corpos humanos, este *dataset* contém 155 imagens recolhidas de *datasets* de outros artigos, incluindo o referido acima. Para garantir a qualidade das imagens foram usadas regras para filtrar as imagens como, é necessário verificar que as imagens são realmente de cenários pós-desastre, excluir marcas d'água e selecionar as imagens com corpos humanos. A deteção de anomalias, avaliação da qualidade das imagens e anotação de todas as imagens foi feito pela mesma pessoa para garantir consistência. Os cenários de desastres nestas imagens incluem terramotos, inundações, furacões e incêndios, em áreas rurais, áreas urbanas muito povoadas, edifícios em ruínas, entre outros. Para fazer a anotação foi utilizada a ferramenta "LabelImg". A variação de pessoas nas imagens varia entre 1 e 24, sendo 1 pessoa o mais comum e o seu tamanho também varia bastante. Ainda foram removidas mais 112 imagens devido a terem demasiadas pessoas aglomeradas. A distribuição do set de treino, validação e teste foi de 8:1:1, o treino consistiu de 832 imagens e a validação e o teste continham 100 imagens cada, cada imagem com o tamanho de 640X640.

O *dataset* do artigo [46] encontra-se no github [50] e no kaggle [52]. Este *dataset* foi criado usando um pipeline sistemático, foram utilizadas imagens d AIDER e do LSP/MPII-MPHB. O AIDER contém imagens de grandes desastres como fogo/fumo, inundações, edifícios colapsados/escombros e acidentes rodoviários. O LSP/MPII-MPHB contém 26675 imagens com 29732 instâncias de corpos humanos em várias posições como *bent, kneeling, sitting, upright*, e *lying*, tem anotações detalhadas das poses humanas. O primeiro passo do pipeline foi usar o modelo U2Net para remover o fundo das imagens com as poses isolando os corpos, de seguida são analisadas as poses para criar as caixas delimitadoras e situações onde o corpo é muito pequeno são removidas para evitar ruído. Por fim para cada desastre são sobrepostas poses aleatoriamente, o tamanho das figuras humanas também é

aleatório tendo em conta a dimensão do cenário de desastre, são ajustadas novamente as caixas para o tamanho da figura humana. Este *dataset* foi chamado de C2A com um total de 10215 imagens com mais de 360000 objetos humanos, o tamanho das imagens varia entre 123X152 pixéis e 5184X3456 pixéis sendo o mais comum entre 322 e 600 pixéis, o tamanho dos objetos varia bastante com 47% a >10 pixéis, 52% 10-50 pixéis e 1% 50-300 pixéis. As anotações para as posições são as do LSP/MPII-MPHB e para os cenários são *traffic incident, fire, flood,* e *collapsed building .*

3.7. Principais Conclusões

O estudo de análise realizado permite constatar um crescente interesse e investigação do uso de CNN e UAV para a deteção de pessoas em cenários de desastre. Este interesse está diretamente relacionado à importância de localizar pessoas em desastres. Diversos autores destacam a relevância da criação de novos *datasets* que representem de forma mais fiel situações reais, permitindo que os UAV identifiquem pessoas de maneira mais precisa e eficaz.

Por esse mesmo motivo podemos verificar que a maioria dos autores optaram por criar o seu próprio *dataset* visto que os que existiam não eram diversificados o suficiente, tanto a nível de cenários de desastres representados como a nível do tipo de formas em as pessoas eram representadas. Todos os artigos que criaram o seu próprio *dataset* descreveram detalhadamente como ele foi criado, fazendo imagens compostas através da harmonização. As aplicações para fazer esta harmonização foram variadas.

Quanto aos modelos podemos observar que quase todos os artigos optam por utilizar vários modelos podendo assim fazer a comparação e verificar qual funciona melhor para aquele *dataset*. Foram usadas maioritariamente diversas variações do YOLO e diversas variações do DETR pré-treinadas em outros *datasets*. A maioria dos autores apresentou os resultados e mesmo que não tenham apresentado identificaram o melhor modelo dentro dos utilizados. O que teve maior destaque foi sempre alguma variação do YOLO. O YOLO é um modelo de deteção de objetos em tempo real, o que o torna popular é a sua velocidade é, extremamente rápido pois não lida com pipelines complexos, a sua alta precisão de deteção, melhor generalização e tem código aberto logo a comunidade pode aprimorá-lo constantemente e é livre de licenças de uso. Apenas olhando para a imagem uma vez ele vai identificar a classe e a sua localização. Primeiro divide a imagem em uma grelha e cada célula vai ser responsável por fazer caixas delimitadores caso haja um objeto naquela célula e é retornada a confiança de cada caixa delimitadora. Para cada caixa a célula também faz a previsão da classe, depois são eliminadas as caixas cuja confiança não atinge o *threshold* definido [53].

4. Tecnologias e Ferramentas Utilizadas

4.1. Python

O Python é uma linguagem de programação de alto nível, orientada a objetos, interpretada. O Python também é semanticamente dinâmico, isto quer dizer que verifica o tipo de variáveis em tempo de execução logo não é necessário declarar o tipo das variáveis. É amplamente utilizado devido à sua sintaxe simples, legibilidade e ampla gama de bibliotecas [54]. Python é bastante popular nas áreas de IA pela presença de bibliotecas, como por exemplo, NumPy, Scikit-Learn, Pandas, Tensorflow, PyTorch, que permitem operações complexas em *datasets* volumosos demonstrado alta escalabilidade [55].

Assim sendo, e considerando a notável relevância e evolução do Python nas áreas de IA, optou-se por utilizar esta linguagem no desenvolvimento deste trabalho. A versão de Python que escolhemos para o desenvolvimento da aplicação foi a 3.12.10 e a aplicação vai ser criada com PyQt5 no VSCode.

4.1.1. PyTorch

O PyTorch é um framework de ML de código aberto desenvolvido pela Meta AI, possui extensas bibliotecas de modelos pré-configurados tornando-o compatível com uma ampla variedade de arquiteturas de redes neuronais, implementa tensores que podem ser executados em GPUs o que permite uma computação mais rápida [56]. Foi usado para treinar e executar redes neuronais (definição de camadas, cálculos de gradientes e otimização).

4.1.2. NumPy

O NumPy fornece um grande conjunto de funções e operações de biblioteca que ajudam os programadores a executar facilmente cálculos numéricos, é uma base para muitas bibliotecas científicas em Python. Esses tipos de cálculos numéricos são amplamente utilizados em tarefas como conversão do formato de dados, cálculos auxiliares de métricas, pré-processamento de entradas e implementa arrays multidimensionais de alto desempenho [57].

4.1.3. OpenCV

O OpenCV é uma biblioteca de código aberto amplamente utilizada para visão computacional, processamento de imagens e ML, permite aos desenvolvedores realizar operações complexas de análise de imagens e vídeos em tempo real, como identificar objetos. Foi usado para o carregamento e pré processamento de dados virtuais e desenho de *bounding boxes* sobre imagens [58].

4.1.4. Matplotlib

O Matplotlib é uma biblioteca de visualização de dados avançada que transforma números em gráficos de uma forma simples e eficiente, podem ser criados gráficos 2D e 3D, estáticos, animados ou até interativos [59]. Foi usado para criar os gráficos como do mAP e visualizar resultados de inferência.

4.1.5. Pandas

Pandas é uma biblioteca de código aberto que providencia uma abordagem rápida e flexível, com estruturas robustas para se trabalhar com dados relacionais ou anotados, de maneira simples e intuitiva. Permite a manipulação e análise de dados em tabelas, facilita a exploração, limpeza e processamento de dados e suas tabelas (DataFrame) [60]. Foi usado para organizar e exportar os resultados de deteção em formato de tabela e análise de estatísticas de métricas.

4.1.6. SciPy

O SciPy é uma biblioteca de computação construída sobre NumPy, providencia mais funções utilitárias para otimização, estatísticas e processamento de sinal [61]. Foi usado para o cálculo de métricas, funções matemáticas e algoritmos na fase de testes.

4.1.7. Tqdm

O Tqdm é uma biblioteca que fornece uma barra de progresso rápida e extensível para loops e iteráveis, facilitando a visualização do progresso do seu código [62]. Foi usado durante o treino para ver o progresso de cada época.

4.1.8. PyYAML

O PyYAML é uma biblioteca conhecida por analisar e manipular arquivos YAML, oferece métodos para carregar dados YAML em objetos Python [63]. Foi usada para a leitura de configuração de hiperparâmetros, arquitetura dos modelos e estruturas dos *datasets.*

4.1.9. Seaborn

O Seaborn é uma biblioteca para criar gráficos estatísticos, construída sobre o matplotlib e integra a estrutura de dados com pandas, amplamente usado para análise exploratória de dados em ciência de dados [64]. Foi usado na criação de histogramas de distribuição e visualização de matrizes de confusão.

4.1.10.SuperGradients

O SuperGradients é uma biblioteca de código aberto que permite o treino ou afinamento de modelos SOTA (*State-of-the-art*) pré-treinados para usos comuns aplicados a visão computacional, suporta deteção de objetos e classificação de imagens [65]. Foi usado como framework principal do modelo YOLO-NASm, um dos modelos que vamos treinar.

4.1.11.Ultralytics

O ultralytics cria modelos YOLO de ponta oferecendo desempenho incomparável em termos de precisão e velocidade, ele baseia-se em versões anteriores, introduzindo novos recursos e melhorias para desempenho, flexibilidade e eficiência aprimorados [66]. É utilizada para o treino, validação, inferência e exportação dos modelos YOLOv5 e YOLOv9, modelos que vamos testar.

Concluída a descrição das bibliotecas responsáveis pelo suporte técnico e computacional dos modelos, apresenta-se de seguida a biblioteca PyQt5, a qual será utilizada para o desenvolvimento da aplicação.

4.1.12.PyQt5

O PyQt5 é uma ligação da linguagem Python para o Qt5, um conjunto de biblioteca destinadas ao desenvolvimento de interfaces gráficas de aplicações. O Qt são bibliotecas que implementam APIs de alto-nível para aceder a funcionalidades modernas de sistemas operativos, como gestão de janelas e eventos. Ao disponibilizar essas funcionalidades no ecossistema Python, possibilita a criação de aplicações robustas e visualmente ricas, mantendo a portabilidade entre diferentes sistemas operativos [67].

4.2. Google Colaboratory

O Google Colaboratory, também conhecido como Google Colab, é uma plataforma de *notebook Jupyter* gratuita baseada em nuvem, oferecida pelo Google, que permite escrever e executar código Python diretamente no navegador, sem necessidade de instalação ou configuração prévia. Uma das suas grandes vantagens é o acesso gratuito a GPUs mas por tempo limitado. Os *notebooks* permitem combinar código executável e texto explicativo, organizados em células, o que facilita a estruturação e compreensão do conteúdo. Cada célula pode ser executada individualmente ou em sequência.

Os *notebooks* são armazenados diretamente no Google Drive, o que simplifica a partilha e a colaboração em tempo real com outras pessoas [68]. Na Figura 9 podemos ver o *notebook* do modelo YOLOv9-e, mais especificamente a parte onde ligamos ao Google Drive para termos acesso ao *dataset* e onde clonamos o repositório do modelo e instalamos as dependências.

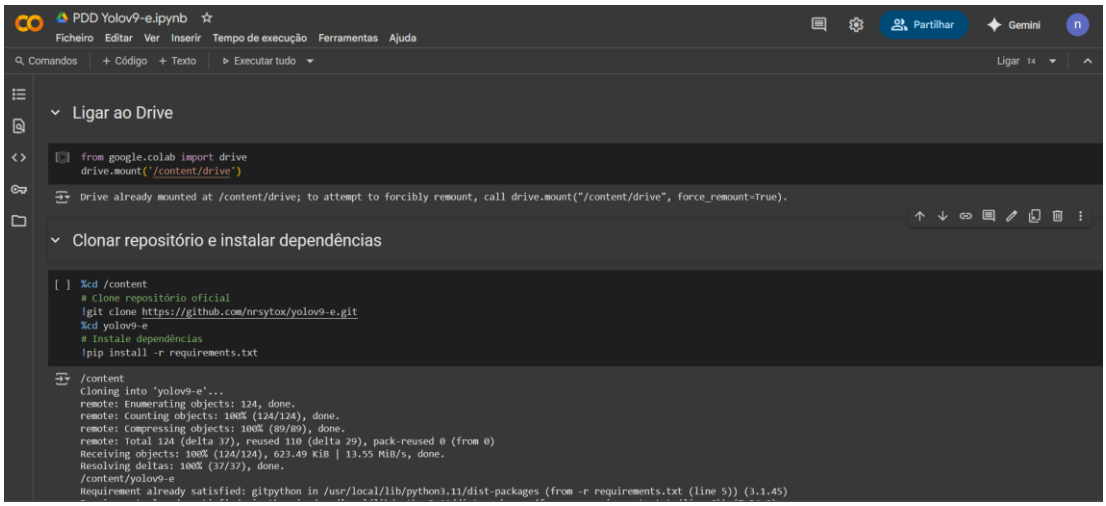

Figura 9- Google Colaboratory Notebook

4.3. Kaggle Notebook

Kaggle Notebook é um ambiente computacional baseado em nuvem que permite análises reprodutivas e colaborativas. Têm vários tipos de notebooks mas o mais comum é o *Jupyter notebooks*. Este consiste na sequência de células, podem ser de texto explicativo ou código executável, que podem ser executadas de forma individual ou sequencial. O *Kaggle* oferece GPUs por um período limitado de 30h semanais gratuitamente. Para introduzir *datasets* no *notebook* é necessário criar esse recurso previamente e adicioná-lo às fontes. Para guardar os ficheiros de *outputs* criados é necessário fazer download visto que são apagados no final de cada sessão [69]. Na

Figura 10 podemos ver o *notebook* do modelo YOLOv9-e, mais especificamente a parte onde clonamos o repositório com o modelo e instalamos as dependências.

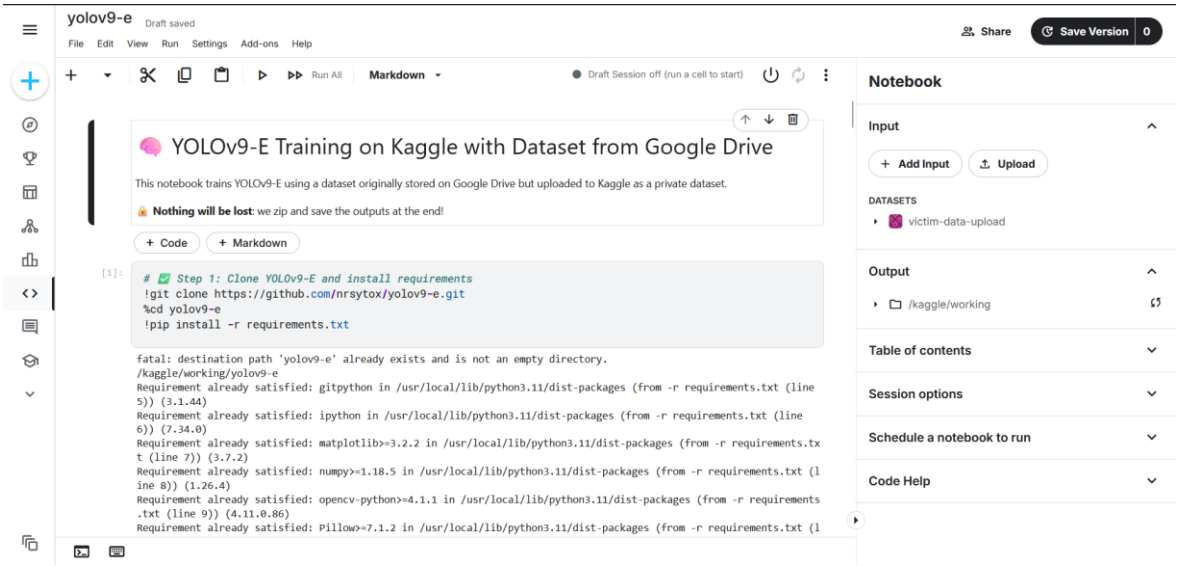

Figura 10- Kaggle Notebook

4.4. Docker Desktop

O *Docker Desktop* permite criar, partilhar e executar containers de forma simples e eficiente. Permite reduzir o tempo gasto em configurações complexas, automatizando tarefas como o mapeamento de portos, gestão do sistema de ficheiros, e outras configurações padrão. Permite, ainda, que os *containers* usem o GPU da máquina para acelerar o treino e permite a personalização do ambiente utilizando *Dockerfiles*. Além disso, é atualizado regularmente com correções de bug e

atualizações de segurança [70]. Usámos esta ferramenta para treinar os modelos. Na Figura 11 podemos ver as imagens dos modelos criados no *Docker*.

Name	Tag	Image ID	Created	Size	Actions
yolonas-c2a	latest	1298c5f71cf1	13 days ago	26.59 GB	
yolov9-c2a	latest	7c7a8dc49596	15 days ago	21.67 GB	
yolov5-pdd	latest	fb08cf15fd37	1 month ago	30.38 GB	

Figura 11- Imagens dos modelos no Docker Desktop

4.5. VS Code

O VS Code (Visual Studio Code) é um editor de código-fonte multiplataforma, gratuito e de código aberto, desenvolvido pela Microsoft. Suporta a maioria das linguagens de programação e integra-se nativamente com sistemas de controlo de versões como o Git, permitindo trabalhar com repositórios remotos ou locais (no Github). Através de extensões, pode ainda incorporar modelos de IA, adaptando-se ao fluxo de trabalho e ao tipo de projeto em desenvolvimento [71]. Usámos esta ferramenta para desenvolver a aplicação. Podemos ver na Figura 12 um excerto do código do layout da aplicação.

```
class PeopleDetectorApp(QWidget):
    def __init__(self):
        """)

        # Layout principal
        main_layout = QVBoxLayout()
        main_layout.setContentsMargins(20, 20, 20, 20)
        main_layout.setSpacing(20)

        # Top layout com label de contagem e progress bar
        top_layout = QVBoxLayout()
        self.label_count = QLabel("Carregue uma imagem")
        self.label_count.setAlignment(Qt.AlignCenter)
        self.label_count.setStyleSheet("font-size: 20px; font-weight: bold;")
        top_layout.addWidget(self.label_count)

        self.progress_bar = QProgressBar()
        self.progress_bar.setValue(0)
        self.progress_bar.setVisible(False)
        top_layout.addWidget(self.progress_bar)

        main_layout.addLayout(top_layout, 10)

        # Layout para caixas de input/output
        img_layout = QHBoxLayout()
        img_layout.setSpacing(20)

        self.label_input = DropLabel(self)
        self.label_input.setText("📁 Arrasta ou seleciona a imagem")
        self.label_input.setAlignment(Qt.AlignCenter)
        self.label_input.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)
        self.label_input.setMinimumSize(300, 300)
        self.label_input.setStyleSheet("""
            QLabel { border: 2px dashed white; border-radius: 10px;
                     background-color: rgba(240, 240, 240, 50); color: #f0f0f0; }
        """)
```

Figura 12- Excerto de código no VS Code

4.6. Github

O Github é uma plataforma de desenvolvimento colaborativo que aloja projetos na nuvem utilizando o sistema de controle de versões chamado Git. A plataforma ajuda os desenvolvedores a armazenar e administrar o código e faz o registo de mudanças. Geralmente é código aberto, o que permite realizar projetos compartilhados e manter o acompanhamento detalhado do seu progresso [72]. Os modelos usados estão alojados no github e foi daí que os importámos. Podemos ver na

Figura 13 o github do YOLOv5.

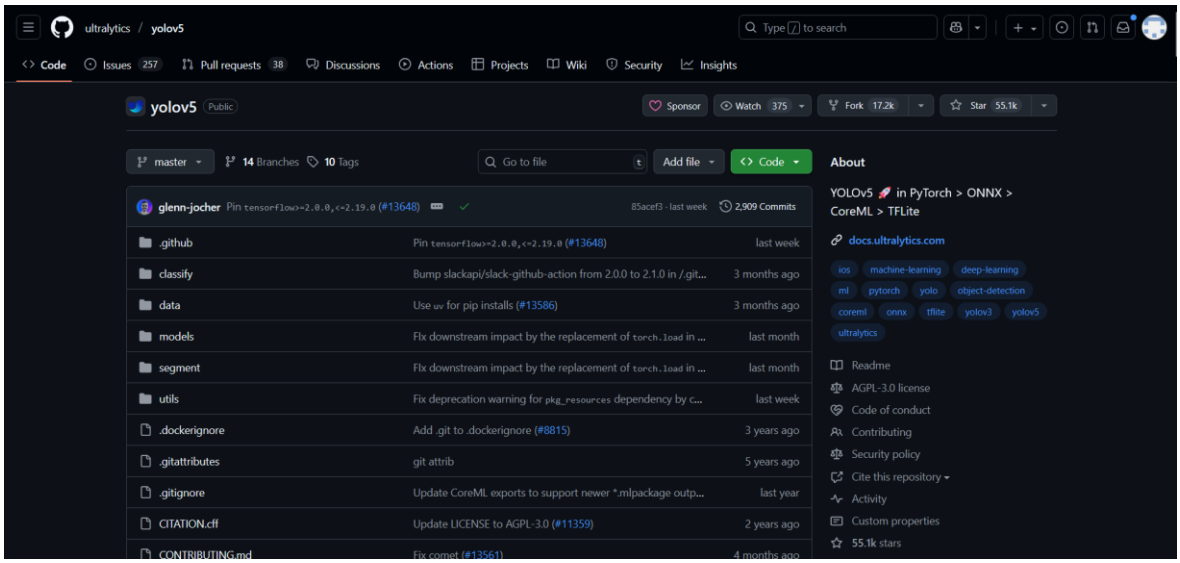

Figura 13- Projeto do YOLOv5 no github

5. Primeiro Trabalho Experimental

Para decidirmos que modelo vamos utilizar para a nossa aplicação decidimos utilizar os três *datasets* que foram disponibilizados pelos autores de alguns dos artigos que analisámos e o modelo que teve melhor desempenho em cada um desses artigos. Os modelos que vamos comparar são o YOLOv5l, YOLOv9-e e o YOLO-NASm, os *datasets* são o PDD, VictimData e o C2A [39,40, 45].

5.1. Datasets

Para utilizarmos os *datasets* no treino dos modelos YOLO foi necessário primeiro prepará-los e organizá-los. Os *datasets* PDD e VictimData, por não estarem estruturados, obrigaram-nos a criar scripts para os dividir em duas partes: treino, com 80% dos dados, e validação, com os 20% restantes. Depois dessa divisão todos os *datasets* foram organizados da mesma forma, ficando com uma pasta para as imagens e outra para os rótulos, cada uma subdividida em *train* e *val*. Mais tarde foi acrescentada uma pasta *annotations* com as anotações das imagens em formato .json em vez de .txt porque o modelo YOLO-NASm só funcionava com as anotações nesse formato. Na Figura 14 podemos ver um diagrama para melhor entender a organização dos *datasets*.

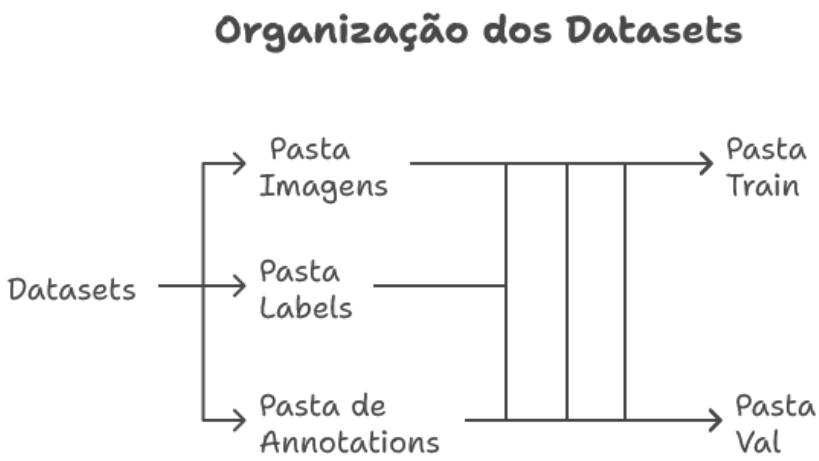

Figura 14- Diagrama da organização dos datasets

O *dataset* C2A, por ser de grande dimensão e já possuir separação entre treino, validação e teste, foi aproveitado diretamente a partir do artigo de referência [52]. A sua organização original colocava imagens e *labels* dentro das pastas *train, val* e *test*, mas nós alterámos essa estrutura para uniformizar com os outros *datasets*, passando a ter uma pasta de imagens e outra *labels*, cada uma subdividida em *train* e *val*. Além disso, optámos por usar o conjunto de teste na fase de avaliação para todos os modelos e comparar os resultados de cada modelo para cada *dataset*. Como neste *dataset* os *labels* estão no formato .json não foi necessário criar uma pasta *annotations*. Na Figura 15 podemos ver um diagrama com a organização deste *dataset.*

Figura 15- Diagrama da organização do dataset C2A

5.2. Modelos Selecionados

Tendo definido os modelos a serem comparados – YOLOv5l, YOLOv9-e e YOLO-NASm – torna-se pertinente apresentar as suas principais características de forma a compreender melhor as diferenças entre eles e interpretar de forma mais clara os resultados obtidos em cada modelo. Esta contextualização é fundamental para enquadrar de forma crítica os resultados experimentais apresentados neste estudo.

5.2.1. YOLOv5l

O YOLOv5l, desenvolvido pela Ultralytics, representa a quinta iteração da família de modelos YOLO, amplamente reconhecida pelo seu impacto na área de deteção de objetos em tempo real. Construído sobre a framework PyTorch, o YOLOv5l destaca-se pela sua elevada velocidade de inferência e precisão, oferecendo um equilíbrio otimizado para aplicações práticas que exigem desempenho em tempo real. A designação "l" (*large*) refere-se à variante de maior capacidade da linha YOLOv5, caracterizada por um número superior de parâmetros e camadas em relação às variantes menores, permitindo maior capacidade de generalização e deteção de objetos em contextos complexos [73]. Este modelo foi referido no artigo [41].

5.2.2. YOLOv9-e

O YOLOv9-e, uma evolução significativa sobre a base robusta do YOLOv5, incorpora inovações arquiteturais que visam otimizar tanto a eficiência como a precisão do processo de deteção. Entre as principais contribuições, destacam-se a Informação de Gradiente Programável (PGI) e a Rede de Agregação de Camadas Eficiente Generalizada, ambas concebidas para mitigar a perda de informação ao longo das camadas da rede. Um dos fundamentos teóricos é o Princípio do Gargalo de Informação, que reconhece que, à medida que os dados atravessam múltiplas camadas, existe uma tendência para a degradação ou perda de informação relevante. O YOLOv9-e contorna este problema através da PGI, que assegura a preservação dos gradientes e das características críticas durante todo o fluxo de processamento, possibilitando atualizações mais precisas dos pesos da rede. Além disso, o modelo adota Funções Reversíveis, que permitem recuperar a informação original a partir das representações internas, reduzindo assim a perda cumulativa em camadas profundas. Estas inovações resultam num modelo mais robusto, capaz de manter elevados níveis de precisão em cenários complexos. A letra "e" indica a variante (*extended)*, com maior profundidade e capacidade de representação [74]. Este modelo foi referido no artigo [46].

5.2.3. YOLO-NASm

O YOLO-NASm (*Neural Architecture Search*), desenvolvido pela Deci AI, é um modelo de deteção de objetos concebido por meio de técnicas avançadas de Pesquisa de Arquitetura Neural. O seu design visa superar limitações identificadas nas versões anteriores da família YOLO, com especial ênfase na quantização eficiente e na otimização da relação precisão–latência. A arquitetura recorre a blocos de reconhecimento de quantização e a quantização seletiva, estratégias que permitem converter o modelo para formatos mais leves, como o INT8, mantendo perdas mínimas de precisão. Estas abordagens possibilitam implementações eficientes em dispositivos com recursos computacionais limitados, sem comprometer significativamente a qualidade da deteção. A versão analisada, identificada pela letra "m" (*medium*), representa um ponto intermédio entre velocidade e capacidade de deteção, sendo adequada para cenários de uso geral que requerem um equilíbrio entre desempenho e custo computacional [75]. Este modelo foi referido no artigo [40].

5.3. Treino Experimental

Para o primeiro treino experimental decidimos utilizar o *dataset* mais pequeno, PDD, para treinar cada modelo como forma de podermos ajustar os modelos e ficarmos com uma ideia de quanto tempo o treino demorará. Decidimos fazer um treino com *batch* 4 e 50 épocas para cada modelo assim podemos ficar com uma ideia do tempo que cada um demora a completar o treino.

O trabalho experimental foi feito no Google Colaboratory utilizando um GPU NVIDIA T4 disponibilizado pelo ambiente. Começámos por organizar o *dataset* como descrito anteriormente, sendo este depois carregado para o Google Drive, por fim criámos o ficheiro .yaml para YOLO-NASm e outro ficheiro .yaml para o YOLOv5l e YOLOv9-e, permitindo assim definir a estrutura dos parâmetros de treino. Podemos ver exemplos dos .yaml para o *dataset* PDD na Figura 16 e Figura 17.

```yaml
Dir: /content/drive/MyDrive/pdd_pdd_yolo

images:
  train: images/train
  val: images/val

labels:
  train: annotations/train.json
  val: annotations/val.json


names:
  0: person
nc: 1
```

Figura 16- Exemplo do .yaml do YOLO-NASm para o dataset PDD

```yaml
path: /content/drive/MyDrive/pdd_pdd_yolo
train: images/train
val: images/val

nc: 1
names: ['person']
```

Figura 17- Exemplo do .yaml do YOLOv5l e YOLOv9-e para o dataset PDD

5.3.1. Implementação do Trabalho Experimental

Para realizar o treino dos diferentes modelos, utilizámos sempre comandos com uma estrutura muito semelhante, como se pode observar na Figura 18, na Figura 19 e na Figura 20. Em todos os casos, o processo base consiste em executar o ficheiro train.py do repositório correspondente, fornecendo ao script um conjunto de parâmetros que são essenciais para o treino. Entre estes parâmetros destacam-se o tamanho do *batch*, o número de épocas, a indicação do ficheiro que descreve a organização do *dataset*, o nome da pasta onde os resultados devem ser guardados e a escolha do dispositivo de execução, podendo este ser CPU ou GPU.

Assim, embora cada código apresentado nas figuras corresponda a um modelo diferente, todos seguem a mesma lógica de funcionamento, mantendo este núcleo de parâmetros em comum. Esta uniformidade permite compreender facilmente o padrão de treino e aplicá-lo a qualquer um dos modelos utilizados, garantindo consistência no processo experimental.

```
!python train.py \
  --img 640 \
  --batch 4 \
  --epochs 50 \
  --data /content/drive/MyDrive/pdd_pdd_yolo/pdd.yaml \
  --cfg /content/yolov5/models/yolov5l.yaml \
  --weights yolov5l.pt \
  --name yolov5l_trabalho_experimental \
  --project /content/drive/MyDrive/pdd_yolo_output/yolov5\
  --device 0
```

Figura 18- Código exemplo para começar treino yolov5

```
%cd /content/yolov9-e/
!python train.py \
--batch 4 --epochs 50 --img 640 --device 0 \
--data /content/drive/MyDrive/pdd_pdd_yolo/pdd.yaml\
--weights /content/weights/yolov9-e.pt \
--cfg models/detect/yolov9-e.yaml \
--hyp hyp.scratch-high.yaml \
--project /content/drive/MyDrive/teste \
--name yolov9_e_treino
```

Figura 19- Código exemplo para começar treino yolov9

```
%cd /content/YOLO-NAS
!python3 train.py \
 --data /content/drive/MyDrive/pdd_pdd_yolo/pdd_yoloNas.yaml \
 --batch 4 \
 --epoch 50 \
 --model yolo_nas_m \
 -n /content/drive/MyDrive/teste
```

Figura 20 - Código exemplo para começar treino yolo-nas

Neste trabalho, devido a dificuldades técnicas impostas pelo Google Colab, como já foi referido anteriormente, foi necessário recorrer a uma funcionalidade disponível nestes modelos que permite retomar o treino a partir de um ponto previamente guardado. Essa abordagem, designada por treino por checkpoints, possibilita a continuação do processo sem que seja necessário reiniciá-lo desde o início, aproveitando os pesos resultantes da última execução. Tal funcionalidade revelou-se essencial para ultrapassar as limitações de tempo de execução e de interrupções do ambiente.

Na Figura 21 apresenta-se um exemplo de como retomar o treino no YOLOv5, através da utilização do parâmetro --resume, que aponta para o ficheiro de pesos gerado na última sessão. De forma equivalente, o mesmo procedimento pode ser aplicado ao YOLOv9, como ilustrado na Figura 22, onde o treino é retomado a partir do último ficheiro de pesos disponível.

```
# Para YOLOv5
!python train.py --resume /content/drive/MyDrive/pdd_yolo_output/yolov5/yolov5l_pdd_detect/weights/last.pt
```

Figura 21 - Código para recomeçar o treino yolov5

```
!python train.py --resume /content/drive/MyDrive/pdd_output/weights/last.pt
```

Figura 22 - Código para recomeçar o treino yolov9

No caso do YOLO-NAS, o processo segue a mesma lógica, exigindo apenas alguns campos adicionais, como ilustrado na Figura 23, o que assegura a consistência do método de continuação do treino em todos os modelos analisados. Convém acrescentar que, para o YOLO-NAS, o retomar do treino só funciona corretamente se os ficheiros ckpt_latest.pth e ckpt_best.pth forem copiados para a pasta runs do repositório clonado. Este procedimento pode ser realizado, por exemplo, através dos seguintes comandos presentes na Figura 24:

```
!python train.py \
  --data /content/drive/MyDrive/new_dataset3/C2A_YOLONAS.yaml\
  --batch 4 \
  --model yolo_nas_m \
  -n teste \
  --weight /content/YOLO-NAS/runs/c2a/ckpt_latest.pth\
  --resume
```

Figura 23 - Código para recomeçar o treino yolo-nas

```
!mkdir -p /content/YOLO-NAS/runs/c2a
!ln -sf /content/drive/MyDrive/c2a/ckpt_best.pth /content/YOLO-NAS/runs/c2a/ckpt_best.pth
!ln -sf /content/drive/MyDrive/c2a/ckpt_latest.pth /content/YOLO-NAS/runs/c2a/ckpt_latest.pth
```

Figura 24 - Código para copiar os ficheiros de checkpoint para a pasta runs do YOLO-NAS

Desta forma, garante-se que o YOLO-NAS consegue identificar corretamente os *checkpoints* existentes e retomar o treino a partir do último estado guardado, mantendo a uniformidade do processo experimental com os outros modelos.

Por fim, no caso do YOLO-NAS, foi necessário proceder a algumas adaptações relacionadas com a versão do Python e determinadas dependências incompatíveis com este modelo. Para assegurar a correta compatibilidade entre o ambiente de execução e os requisitos do YOLO-NAS, recorreu-se à instalação de uma versão específica do Python e ao ajuste das bibliotecas necessárias. Esse processo encontra-se exemplificado na Figura 25, onde é apresentado o código utilizado para configurar o ambiente adequado ao treino deste modelo.

```
!wget -O mini.sh https://repo.anaconda.com/miniconda/Miniconda3-py310_25.5.1-0-Linux-x86_64.sh
!chmod +x mini.sh
!bash ./mini.sh -b -f -p /usr/local
# Aceitar termos Conda
!conda config --set channel_priority strict
!conda config --add channels defaults
!conda tos accept --channel https://repo.anaconda.com/pkgs/main
!conda tos accept --channel https://repo.anaconda.com/pkgs/r

# Instalar pacotes essenciais
!conda install -q -y ipykernel jupyter google-colab -c conda-forge

# Registrar kernel
!python -m ipykernel install --name "py310" --user
```

Figura 25 - Código de configuração do ambiente para treino do YOLO-NAS

5.3.2. Resultados Obtidos

Na Tabela 4 podemos ver o mAP50 do treino de cada modelo e o tempo de treino em horas. Por baixo da tabela podemos ver os gráficos correspondentes ao mAP50 de cada época ao longo do treino.

Tabela 4- Resultados do trabalho experimental

Modelos	mAP50	Tempo de treino (horas)
YOLOv5l	0.8990	1:17
YOLOv9-e	0.0154	1:40
YOLO-NASm	0.7231	2:03

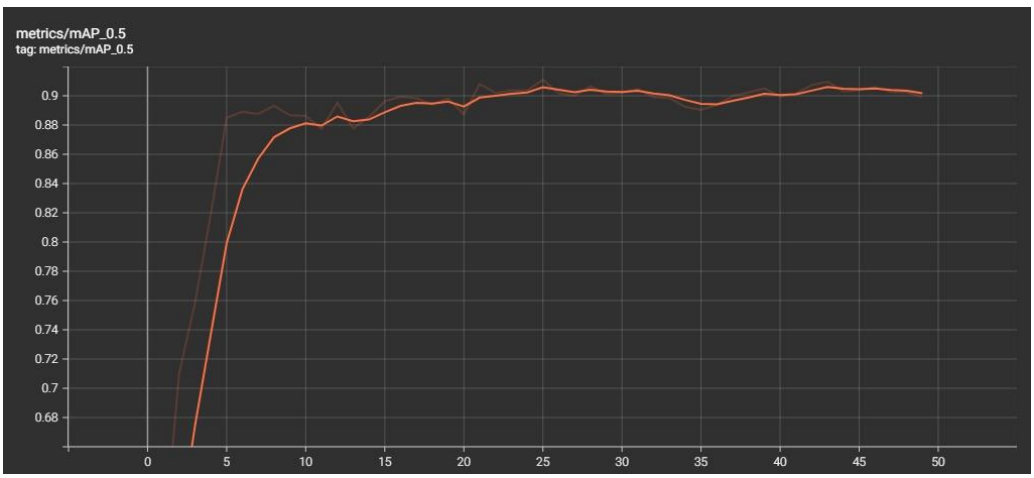

Figura 26 - Gráfico do mAP50 ao longo das épocas (YOLOv5l)

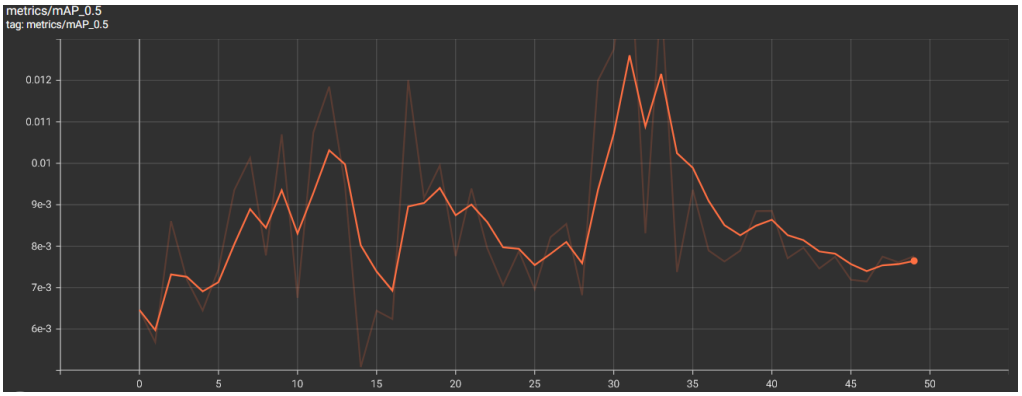

Figura 27- Gráfico do mAP50 ao longo das épocas (YOLOv9-e)

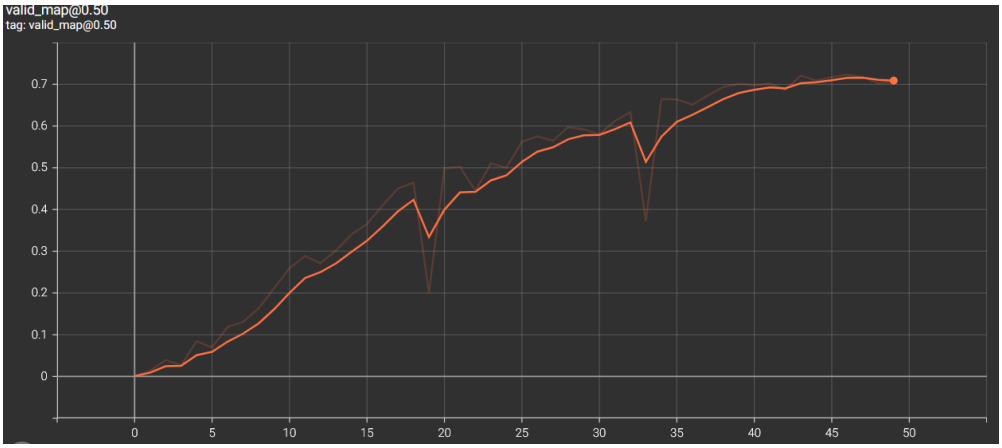

Figura 28- Gráfico do mAP50 ao longo das épocas (YOLO-NAS)

Como podemos ver na tabela o YOLO-NASm é o modelo com maior tempo de treino, mas apresenta bons resultados. O modelo que apresenta melhores resultados é o YOLOv5l. Nos gráficos (Figura 26, Figura 27, Figura 28) podemos ver que em quase todos os casos o modelo apresenta uma melhoria de mAP50 ao longo das épocas. Na Figura 27 podemos ver que o modelo YOLOv9-e não mostra grande melhoria ao longo do treino e o seu desempenho é muito baixo, como apenas este modelo tem desempenho tão reduzido podemos concluir que o problema não estará no *dataset* mas sim no treino do modelo, pode precisar de mais épocas para aprender a generalizar os padrões ou o *batch size* ser pequeno demais.

5.3.3. Principais Conclusões

Neste primeiro trabalho experimental conseguimos completar o treino que tínhamos planeado com cada modelo, que no total demorou à volta de 2 semanas no total, começámos por fazer o YOLOv5l e o YOLO-NASm e depois de termos esses treinos completos passámos para o YOLOv9-e, para contornar o problema do uso de GPU limitado do Google Colaboratory usámos uma conta de gmail para cada modelo, podendo assim treinar todos os modelos ao mesmo tempo. Pudemos perceber que o modelo YOLO-NASm ao contrário dos outros apenas funciona com a versão do Python 3.10 devido a dependências do super-gradients que não existiam na versão mais recente, 3.11.13. Tivemos de fazer algumas alterações no código base para corrigir problemas de estrutura de saída e ajustes de compatibilidade com o PyTorch, para guardar estas alterações foram criados repositórios no github para o YOLO-NASm e YOLOv9-e (https://github.com/nrsytox/YOLO-NAS.git https://github.com/nrsytox/yolov9-e.git). Também optámos por explorar a opção de treino por *checkpoints*, uma vez que o Google Colaboratory impõe um tempo limite de utilização do GPU, esta abordagem foi fundamental para permitir guardar o progresso parcial do treino e retomar o processo posteriormente sem perder o trabalho já realizado.

6. Conclusões

O trabalho aqui apresentado teve como principal objetivo compreender os estudos já realizados na área da deteção de pessoas em desastres naturais. Em situações como inundações ou terramotos, a rapidez no resgate de sinistrados é um fator determinante, representando um desafio acrescido pelas condições adversas do terreno, mas também pela dificuldade em detetar pessoas entre escombros e destroços. Atualmente, já são utilizados UAVs em missões de busca e resgate, mas as soluções existentes apresentam limitações, nomeadamente na velocidade de deteção de pessoas. Neste tipo de situações a rapidez na deteção dos sinistrados é crucial para o seu resgate. Assim, este projeto procurou analisar soluções anteriores que recorrem à IA para apoiar as equipas de resgate.

Portanto, procurámos descobrir que outros trabalhos de DL já tinham sido feitos para deteção de pessoas em cenários de desastre. Após essa pesquisa, procurámos identificar que modelos e *datasets* cada trabalho utilizava e disponibilizava.

Inicialmente, investigámos os tipos de aprendizagens computacionais existentes, de forma a compreender a melhor abordagem ao problema em estudo. Confirmou-se que o problema que pretendemos resolver com este trabalho requer uma aprendizagem supervisionada. Foi também mostrada a necessidade de serem utilizadas métricas corretas para avaliar o desempenho dos modelos de DL.

O estudo do estado da arte permitiu analisar 8 artigos. Este trabalho de investigação permitiu-nos descobrir os modelos utilizados para este tipo de trabalhos e quais apresentavam melhores resultados. Permitiu-nos também analisar os *datasets* disponíveis. Podemos observar que os modelos mais mencionados e com melhor desempenho foram variantes do modelo YOLO e os três *datasets* disponíveis foram criados pelos autores dos artigos onde foram mencionados. É importante destacar que nenhum dos trabalhos analisados apresentou simultaneamente a comparação entre os três modelos considerados neste estudo (YOLOv5l, YOLOv9-e, YOLO-NASm) nem avaliou o desempenho destes três *datasets* em conjunto, o que reforça a importância da nossa abordagem.

Em síntese, esta primeira fase do projeto permitiu-nos compreender as principais abordagens de DL aplicadas à deteção de pessoas em cenários de desastre. Verificámos que os modelos mais utilizados são diferentes versões do YOLO, o que confirmámos também no nosso primeiro trabalho experimental, onde explorámos o processo de treino dos mesmos e os respetivos resultados. Importa salientar que os modelos utilizados correspondem aos mencionados nos artigos em que os autores disponibilizaram os respetivos *datasets* [39,40, 45]. Assim, na segunda fase do projeto iremos treinar as três variantes do modelo YOLO nos três *datasets* identificados, de forma a comparar o seu desempenho e identificar a melhor combinação. Posteriormente, será desenvolvida uma aplicação que integre o modelo mais eficaz, com o objetivo de apoiar as operações de resgate.

Referências

[1] Inemesit Ukpanah, "Is The Number of Natural Disasters Increasing?," greenmatch. Accessed: Jan. 25, 2025. [Online]. Available: https://www.greenmatch.co.uk/blog/are-natural-disasters-increasing

[2] Lance Piatt, "Overcoming Search and Rescue Challenges." Accessed: Jan. 25, 2025. [Online]. Available: https://rigginglabacademy.com/overcoming-search-rescue-challenges/

[3] A. A. B. Abdelnabi and G. Rabadi, "Human Detection From Unmanned Aerial Vehicles' Images for Search and Rescue Missions: A State-of-the-Art Review," *IEEE Access*, vol. 12, pp. 152009–152035, 2024, doi: 10.1109/ACCESS.2024.3479988.

[4] Rui Parreira, "Drones da GNR ajudaram a salvar vidas em mais de 200 operações de emergência e socorro - Computadores - SAPO Tek," *SAPO*, Oct. 2024, Accessed: Jan. 27, 2025. [Online]. Available: https://tek.sapo.pt/noticias/computadores/artigos/drones-da-gnr-ajudaram-a-salvar-vidas-em-mais-de-200-operacoes-de-emergencia-e-socorro

[5] "What Is Artificial Intelligence (AI)? | IBM." Accessed: Nov. 30, 2024. [Online]. Available: https://www.ibm.com/topics/artificial-intelligence

[6] R. E. Neapolitan and X. Jiang, *Artificial intelligence: With an introduction to machine learning*. CRC press, 2018.

[7] M. Flasiński, *Introduction to artificial intelligence*. Springer, 2016.

[8] "What is (AI) Artificial Intelligence? | Online Master of Engineering | University of Illinois Chicago." Accessed: Dec. 01, 2024. [Online]. Available: https://meng.uic.edu/news-stories/ai-artificial-intelligence-what-is-the-definition-of-ai-and-how-does-ai-work/

[9] "🚀 AI, ML & Deep Learning: What's the Difference? 🚀." Accessed: Aug. 22, 2025. [Online]. Available: https://www.linkedin.com/pulse/ai-ml-deep-learning-whats-difference-mark-john-nakgc

[10] "ISO - Machine learning (ML): All there is to know." Accessed: Dec. 01, 2024. [Online]. Available: https://www.iso.org/artificial-intelligence/machine-learning

[11] M. Mohri, "Foundations of machine learning," 2018, *MIT press*.

[12] "What Is Machine Learning (ML)? | IBM." Accessed: Dec. 01, 2024. [Online]. Available: https://www.ibm.com/topics/machine-learning

[13] "Uma introdução a Machine Learning | by Morvana Bonin | Medium." Accessed: Aug. 22, 2025. [Online]. Available: https://medium.com/@morvanabonin/uma-introdu%C3%A7%C3%A3o-a-machine-learning-2c70e915873a

[14] "O que é aprendizado supervisionado? | IBM." Accessed: Dec. 15, 2024. [Online]. Available: https://www.ibm.com/br-pt/topics/supervised-learning

[15] "O que são modelos de machine learning?" Accessed: Dec. 01, 2024. [Online]. Available: https://www.databricks.com/br/glossary/machine-learning-models

[16] "Tipos e Aplicações de Machine Learning | Artigo KAIZENTM." Accessed: Dec. 02, 2024. [Online]. Available: https://kaizen.com/pt/insights-pt/tipos-aplicacoes-machine-learning/

[17] "Processo de decisão de Markov - GeeksforGeeks." Accessed: Jan. 24, 2025. [Online]. Available: https://www.geeksforgeeks.org/markov-decision-process/

[18] P. P. Shinde and S. Shah, "A Review of Machine Learning and Deep Learning Applications," in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 2018, pp. 1–6. doi: 10.1109/ICCUBEA.2018.8697857.

[19] Jim Holdsworth and Mark Scapicchio, "What is deep learning?," IBM. Accessed: Dec. 02, 2024. [Online]. Available: https://www.ibm.com/topics/deep-learning

[20] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[21] R. Y. Choi, A. S. Coyner, J. Kalpathy-Cramer, M. F. Chiang, and J. Peter Campbell, "Introduction to machine learning, neural networks, and deep learning," *Transl Vis Sci Technol*, vol. 9, no. 2, 2020, doi: 10.1167/tvst.9.2.14.

[22] "What are convolutional neural networks?," IBM. Accessed: Dec. 08, 2024. [Online]. Available: https://www.ibm.com/topics/convolutional-neural-networks

[23] "Structure of the model. 2.2.1. CNN. The CNN, a type of artificial… | Download Scientific Diagram." Accessed: Aug. 22, 2025. [Online]. Available: https://www.researchgate.net/figure/Structure-of-the-model-221-CNN-The-CNN-a-type-of-artificial-neural-network-excels_fig2_378435606

[24] "Guia de Redes Neurais — Parte 4. Redes Neurais Convolucionais | by Guilherme Vallim Machado | Medium." Accessed: Aug. 22, 2025. [Online]. Available: https://medium.com/@guilhermevallimmachado/guia-de-redes-neurais-parte-5-6c6a550d71d

[25] Cole Stryker, "What is a recurrent neural network?" Accessed: Dec. 08, 2024. [Online]. Available: https://www.ibm.com/topics/recurrent-neural-networks

[26] "Série: Deep Learning With Python — Parte 6 (2) | by Pretatech | Medium." Accessed: Aug. 13, 2025. [Online]. Available: https://apretatech.medium.com/s%C3%A9rie-deep-learning-with-python-parte-6-2-f0beef60dd35

[27] "What is computer vision?" Accessed: Dec. 15, 2024. [Online]. Available: https://www.ibm.com/think/topics/computer-vision

[28] Y. Chen, S. Wang, L. Lin, Z. Cui, and Y. Zong, "Computer Vision and Deep Learning Transforming Image Recognition and Beyond," *International Journal of Computer Science and Information Technology*, vol. 2, pp. 45–51, Dec. 2024, doi: 10.62051/ijcsit.v2n1.06.

[29] N. Pinto and R. Filho, "VISÃO COMPUTACIONAL: UM NOVO CAMPO DE PESQUISA EM COGNIÇÃO VISUAL*."

[30] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779 – 788. doi: 10.1109/CVPR.2016.91.

[31] A. Nazir and Mohd. A. Wani, "You Only Look Once - Object Detection Models: A Review," in *2023 10th International Conference on Computing for Sustainable Global Development (INDIACom)*, 2023, pp. 1088–1095.

[32] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, "A Review of Yolo Algorithm Developments," *Procedia Comput Sci*, vol. 199, pp. 1066–1073, 2022, doi: https://doi.org/10.1016/j.procs.2022.01.135.

[33] "What Is Overfitting vs. Underfitting? | IBM." Accessed: Aug. 12, 2025. [Online]. Available: https://www.ibm.com/think/topics/overfitting-vs-underfitting

[34] "What is underfitting and overfitting in machine learning and how to deal with it. | by Anup Bhande | GreyAtom | Medium." Accessed: Sep. 12, 2025. [Online]. Available: https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76

[35] "What is a confusion matrix? | IBM." Accessed: Aug. 12, 2025. [Online]. Available: https://www.ibm.com/think/topics/confusion-matrix

[36] "Mean Average Precision (mAP) Explained: Everything You Need to Know." Accessed: Aug. 12, 2025. [Online]. Available: https://www.v7labs.com/blog/mean-average-precision

[37] "Precision e Recall: Um macete para você sair dessa matrix de confusão | by Christiano Peres | Medium." Accessed: Aug. 12, 2025. [Online]. Available: https://medium.com/@christianoDS/precision-e-recall-um-macete-para-voc%C3%AA-nunca-mais-se-confundir-1d28b8ce5f2c

[38] "Accuracy vs. Precision vs. Recall in Machine Learning | Encord." Accessed: Aug. 12, 2025. [Online]. Available: https://encord.com/blog/classification-metrics-accuracy-precision-recall/

[39] S. Gaur and J. S. Kumar, "UAV based Human Detection for Search and Rescue Operations in Flood," in *2023 10th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)*, 2023, pp. 1038–1043. doi: 10.1109/UPCON59197.2023.10434788.

[40] N. Zhang, F. Nex, G. Vosselman, and N. Kerle, "Training a Disaster Victim Detection Network for UAV Search and Rescue Using Harmonious Composite Images," *Remote Sens (Basel)*, vol. 14, no. 13, Nov. 2022, doi: 10.3390/rs14132977.

[41] H. Song, W. Song, L. Cheng, Y. Wei, and J. Cui, "PDD: Post-Disaster Dataset for Human Detection and Performance Evaluation," *IEEE Trans Instrum Meas*, vol. 73, pp. 1–14, 2024, doi: 10.1109/TIM.2023.3346508.

[42] A. A. H. Al-Azzani and M. R. Ahmad, "HUMAN DETECTION IN SEARCH AND RESCUE OPERATIONS USING EMBEDDED ARTIFICIAL INTELLIGENCE," *J Teknol*, vol. 86, no. 3, pp. 187–194, Nov. 2024, doi: 10.11113/jurnalteknologi.v86.19497.

[43] B. V. B. Prabhu, R. Lakshmi, R. Ankitha, M. S. Prateeksha, and N. C. Priya, "RescueNet: YOLO-based object detection model for detection and counting of flood survivors," *Model Earth Syst Environ*, vol. 8, no. 4, pp. 4509–4516, 2022, doi: 10.1007/s40808-022-01414-6.

[44] I. A. Sulistijono *et al.*, "Implementation of Victims Detection Framework on Post Disaster Scenario," in *2018 International Electronics Symposium on Engineering Technology and Applications (IES-ETA)*, 2018, pp. 253–259. doi: 10.1109/ELECSYM.2018.8615503.

[45] R. Bahmanyar and N. Merkle, "SAVING LIVES FROM ABOVE: PERSON DETECTION IN DISASTER RESPONSE USING DEEP NEURAL NETWORKS," in *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Copernicus Publications, Dec. 2023, pp. 343–350. doi: 10.5194/isprs-annals-X-1-W1-2023-343-2023.

[46] R. A. Nihal, B. Yen, K. Itoyama, and K. Nakadai, "UAV-Enhanced Combination to Application: Comprehensive Analysis and Benchmarking of a Human Detection Dataset for Disaster Scenarios," Aug. 2024, [Online]. Available: http://arxiv.org/abs/2408.04922

[47] "SURFdrive." Accessed: Jan. 21, 2025. [Online]. Available: https://surfdrive.surf.nl/files/index.php/s/yPtgoTrB4CohOL0

[48] "GitHub - noahzn/VictimDet: Official PyTorch Implementation of 'Training a Disaster Victim Detection Network for UAV Search and Rescue Using Harmonious Composite Images.'" Accessed: Jan. 21, 2025. [Online]. Available: https://github.com/noahzn/VictimDet?tab=readme-ov-file#start-of-content

[49] "GitHub - HaoqianSong/Post-Disaster-Dataset: A post-disaster survivor dataset is used for UAV post-disaster personnel search and rescue and statistics, it contains the real-world post-disaster UAV images from multiple perspectives, scenes, and distances." Accessed: Jan. 21,

2025. [Online]. Available: https://github.com/HaoqianSong/Post-Disaster-Dataset?tab=readme-ov-file

[50] "GitHub - Ragib-Amin-Nihal/C2A: This repository contains the code and information for the paper 'UAV-Enhanced Combination to Application: Comprehensive Analysis and Benchmarking of a Human Detection Dataset for Disaster Scenarios.'" Accessed: Jan. 22, 2025. [Online]. Available: https://github.com/Ragib-Amin-Nihal/C2A

[51] "GitHub - HaoqianSong/Detection-Models: A post-disaster survivor dataset is used for UAV post-disaster personnel search and rescue and statistics, it contains the real-world post-disaster UAV images from multiple perspectives, scenes, and distances." Accessed: Jun. 24, 2025. [Online]. Available: https://github.com/HaoqianSong/Detection-Models

[52] "C2A Dataset: Human Detection in Disaster Scenarios." Accessed: Jan. 24, 2025. [Online]. Available: https://www.kaggle.com/datasets/rgbnihal/c2a-dataset/data

[53] "Detecção de Objetos com YOLO - Uma abordagem moderna | IA Expert Academy." Accessed: Jan. 25, 2025. [Online]. Available: https://iaexpert.academy/2020/10/13/deteccao-de-objetos-com-yolo-uma-abordagem-moderna/

[54] "What is Python? Executive Summary | Python.org." Accessed: Aug. 07, 2025. [Online]. Available: https://www.python.org/doc/essays/blurb/

[55] "6 Reasons Why Is Python Used for Machine Learning." Accessed: Aug. 07, 2025. [Online]. Available: https://www.newhorizons.com/resources/blog/why-is-python-used-for-machine-learning

[56] "O que é PyTorch? | IBM." Accessed: Aug. 12, 2025. [Online]. Available: https://www.ibm.com/br-pt/think/topics/pytorch

[57] "Entendendo a biblioteca NumPy. O que é o NumPy? | by Luiz Santiago Jr. | Ensina.AI | Medium." Accessed: Aug. 12, 2025. [Online]. Available: https://medium.com/ensina-ai/entendendo-a-biblioteca-numpy-4858fde63355

[58] "O que é OpenCV e para que serve? | Asimov Academy." Accessed: Aug. 12, 2025. [Online]. Available: https://hub.asimov.academy/blog/o-que-e-opencv-e-para-que-serve/

[59] "Matplotlib: guia completo para iniciantes em visualização de dados | Asimov Academy." Accessed: Aug. 13, 2025. [Online]. Available: https://hub.asimov.academy/blog/matplotlib-visualizacao-de-dados/

[60] "Pandas Python: Como usar a ferramenta #1 de análise de dados." Accessed: Aug. 13, 2025. [Online]. Available: https://hub.asimov.academy/blog/pandas-python/

[61] "Introduction to SciPy." Accessed: Aug. 13, 2025. [Online]. Available: https://www.w3schools.com/python/scipy/scipy_intro.php

[62] "Tqdm Python: Um guia com exemplos práticos | DataCamp." Accessed: Aug. 13, 2025. [Online]. Available: https://www.datacamp.com/pt/tutorial/tqdm-python

[63] "O que é YAML? | IBM." Accessed: Aug. 13, 2025. [Online]. Available: https://www.ibm.com/br-pt/think/topics/yaml

[64] "An introduction to seaborn — seaborn 0.13.2 documentation." Accessed: Aug. 13, 2025. [Online]. Available: https://seaborn.pydata.org/tutorial/introduction

[65] "super-gradients · PyPI." Accessed: Aug. 13, 2025. [Online]. Available: https://pypi.org/project/super-gradients/2.0.0/

[66] "Início - Documentos Ultralytics YOLO." Accessed: Aug. 13, 2025. [Online]. Available: https://docs.ultralytics.com/pt/#the-evolution-of-object-detection

[67] "Riverbank Computing | Introduction." Accessed: Aug. 13, 2025. [Online]. Available: https://riverbankcomputing.com/software/pyqt/

[68] "Damos-lhe as boas-vindas ao Colab - Colab." Accessed: Aug. 06, 2025. [Online]. Available: https://colab.research.google.com/?hl=pt_PT#scrollTo=Wf5KrEb6vrkR

[69] "Getting Started on Kaggle | Kaggle." Accessed: Aug. 06, 2025. [Online]. Available: https://www.kaggle.com/docs/notebooks

[70] "Docker Desktop | Docker Docs." Accessed: Aug. 07, 2025. [Online]. Available: https://docs.docker.com/desktop/

[71] "Visual Studio Code - Code Editing. Redefined." Accessed: Aug. 22, 2025. [Online]. Available: https://code.visualstudio.com/

[72] "O que é GitHub: para que serve, como funciona e como utilizar." Accessed: Aug. 07, 2025. [Online]. Available: https://ebaconline.com.br/blog/o-que-e-github

[73] "Guia Abrangente do Ultralytics YOLOv5 - Documentos Ultralytics YOLO." Accessed: Aug. 14, 2025. [Online]. Available: https://docs.ultralytics.com/pt/yolov5/

[74] "YOLOv9: Um Salto à Frente na Tecnologia de Detecção de Objetos." Accessed: Aug. 14, 2025. [Online]. Available: https://docs.ultralytics.com/pt/models/yolov9/#how-can-i-train-a-yolov9-model-using-python-and-cli

[75] "YOLO-NAS - Documentos Ultralytics YOLO." Accessed: Aug. 14, 2025. [Online]. Available: https://docs.ultralytics.com/pt/models/yolo-nas/#citations-and-acknowledgements