Instituto Superior
de Engenharia
Politécnico de Coimbra

DEPARTAMENTO DE ENGENHARIA
ELETROTÉCNICA

Desenvolvimento de aplicações utilizando AI

Trabalho de Projeto para a obtenção do grau de Mestre em Engenharia Eletrotécnica

Especialização em Automação e Comunicações em Sistemas de Energia

Autor

Cláudio Rafael Rodrigues Fonseca

Orientadores

Inácio de Sousa Adelino da Fonseca

Fernando José Pimentel Lopes

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

Coimbra, março 2024

RESUMO

Num mundo cada vez mais acelerado, a necessidade de maior praticidade no nosso dia a dia torna-se quase imperativa para trabalhos que vão desde as tarefas domésticas às tarefas profissionais. A busca por maior eficiência e otimização de tempo impulsiona a procura por soluções que simplifiquem as nossas atividades diárias. No cenário pessoal, a praticidade economiza-nos tempo para nos focarmos em aspetos mais significativos da vida. Já no âmbito profissional, a praticidade facilita a execução das tarefas e impulsiona a inovação e a competitividade.

A pensar neste cenário idealizou-se um sistema capaz de detetar tags HTML, tanto em folhas de papel como em *Wireframes* ou em páginas WEB. Este sistema de deteção é proposto no presente trabalho e para a implementação do mesmo foram analisados vários sistemas de deteção de objetos na área da visão computacional. Estes sistemas são modelos de treino baseados em redes neurais convolucionais constituídos por algoritmos capazes de aprender a reconhecer objetos em imagens. Para conseguirem aprender a interpretar as imagens, estes algoritmos necessitam de ser alimentados por um *dataset* suficientemente grande e variado, onde consigam distinguir elementos entre imagens semelhantes. Para isso, foram criados 4 conjuntos de dados com imagens que são compostas por Layouts de Páginas WEB feitos à mão em folhas brancas, *wireframes* e páginas WEB com tags HTML. Com o *dataset* criado, os modelos foram treinados e testados, demonstrando que é possível detetar com sucesso tags HTML com eficácia.

Numa fase posterior a este trabalho, pretende-se implementar as tags detetadas e desenvolver a página WEB, evitando assim a necessidade de implementar código HTML cada vez que é produzido um novo *layout* de página WEB. Este sistema permite que a produção de novas páginas WEB simples seja feita de forma mais rápida e precisa, poupando tempo e recursos. Para além disso, permite que pessoas com pouco conhecimento técnico possam criar *layouts* WEB de maneira eficaz e rápida.

Palavras-chave: YOLO, *Dataset*, Detectron2

ABSTRACT

In an increasingly fast-paced world, the need for greater practicality in our daily lives has become almost imperative for jobs ranging from domestic chores to professional tasks. The search for greater efficiency and time optimization drives the demand for solutions that simplify our daily activities. On the personal front, practicality saves us time to focus on more meaningful aspects of life. In the professional sphere, practicality makes it easier to carry out tasks and drives innovation and competitiveness.

With this scenario in mind, a system capable of detecting HTML tags was devised, both on sheets of paper and in wireframes or web pages. This detection system is proposed in this paper and, in order to implement it, various object detection systems in the field of computer vision were analysed. These systems are training models based on convolutional neural networks made up of algorithms capable of learning to recognize objects in images. In order to learn to interpret images, these algorithms need to be fed a sufficiently large and varied data set, where they can distinguish elements between similar images. To this end, 4 *dataset*s were created with images consisting of sheets of paper, wireframes and web pages with HTML tags. With the *dataset* created, the models were trained and tested, demonstrating that it is possible to successfully detect HTML tags.

At a later stage, the aim is to implement the tags detected and develop the web page, thus avoiding the need to implement HTML code every time a new web page layout is produced. This system allows new simple web pages to be produced more quickly and accurately, saving time and resources. It also allows people with little technical knowledge to create web layouts quickly and efficiently.

Keywords: YOLO, Data set, Detectron2

AGRADECIMENTOS

Em primeiro lugar gostaria de agradecer à minha família e à minha namorada, pois sem eles não seria possível percorrer esta jornada académica. Merecem uma gratidão especial por todo o apoio, amor e compreensão ao longo deste percurso. Foram um grande incentivo a alcançar os meus objetivos com determinação e dedicação.

Aos professores orientadores, Professor Inácio Fonseca e Professor Fernando Lopes, pela ajuda e disponibilidade durante todo o desenvolvimento deste trabalho.

Aos amigos e colegas, pelo companheirismo e palavras de incentivo.

A todos vós, que de uma maneira ou outra, contribuíram para o sucesso deste trabalho, muito obrigado!

CONTEÚDO

ÍNDICE DE TABELA

Índice de figuras

LISTA DE ABREVIATURAS

ANN – *Artificial Neural Network*

AP – *Average Precision*

CNN – *Convolution Neural Network*

CSP – *Cross Stage Partial*

CSS – *Cascading Style Sheets*

FAIR – *Facebook Artificial Intelligence Research*

FPS – *Frame per Second*

FPN – *Feature Pyramid Network*

GPU – *Graphics Processing Unit*

HTML – *Hyper Text Markup Language*

ICR – *Intelligent Character Recognition*

IEEE – *Institute of Electrical and Electronics Engineers*

ILSVR – *ImageNet Large Scale Visual Recognition Challenge*

IOU – *Intersection Over Union*

ISEC – Instituto Superior de Engenharia de Coimbra

K-NN – *K-Nearest Neighbors*

MAP – *Mean Average Precision*

MSCOCO – *Microsoft Common Objects in Context*

NMS – *Non-Max Suppression*

OCR – *Optical Character Recognition*

PR – *Precision and Recall*

PASCALVOC – *PASCAL Visual Object Classes*

RELU – *Rectified Linear Unit*

RGB – *Red, Green, and Blue*

SVM – *Support Vector Machines*

VGG – *Visual Geometry Group*

YOLO – *You Only Look Once*

1 INTRODUÇÃO

Neste capítulo é feita uma breve descrição do presente trabalho, apresentando os seus objetivos, resultados esperados e por fim a estrutura do documento.

1.1 Enquadramento do trabalho e objetivos

Num mundo cada vez mais dinâmico, complexo e onde o tempo é escasso, a necessidade de obter tecnologias ágeis e simples que priorizam a praticidade tanto a nível pessoal como profissional é essencial para aumentar a nossa eficiência e contribuir para uma vida mais equilibrada. Neste trabalho é proposto um sistema capaz de detetar tags HTML em folhas de papel, *Wireframes* ou Páginas WEB e numa fase posterior implementar esse sistema num aplicativo que seja capaz de replicar o *layout* definido e transformar esse *layout* numa página WEB ou transformar esse *layout* numa *interface* para um aplicativo.

Neste trabalho serão estudados alguns sistemas de deteção de objetos usados na área da visão computacional e analisado qual o sistema que melhor se adequa ao trabalho proposto.

Para a elaboração deste projeto, antes do estudo de análise aos sistemas de deteção disponíveis, foi necessário juntar conjuntos de imagens que vão ser utilizadas no sistema de deteção de objetos. Foram utilizadas várias imagens e as mesmas imagens foram divididas em 4 *datasets*. O 1º *dataset* são imagens de folhas de papel com *layouts* feitos à mão (deu origem ao 2º *dataset*), o 2º *dataset* são imagens de páginas WEB desenvolvidas em HTML e CSS (deu origem ao 3º *dataset*), o 3º conjunto são imagens de *Wireframes* onde algumas são feitas à mão e outras retiradas da internet (deu origem ao 4º *dataset*). O 4º *dataset* é a união dos 3 conjuntos de dados, nomeadamente do 1º *dataset*, do 2º *dataset* e do 3º *dataset* (deu origem ao 5º *dataset*).

Com todos os conjuntos de dados definidos procedeu-se à implementação dos sistemas de deteção analisados neste trabalho. O desempenho de cada sistema foi avaliado e foi feita uma comparação entre os sistemas analisados.

1.2 Estrutura do documento

De forma sumária, o presente trabalho encontra-se assim dividido:

1. **Capítulo 1:** O capítulo 1 apresenta o âmbito e objetivos deste trabalho.
2. **Capítulo 2:** No 2º capítulo é feito um enquadramento teórico dos principais temas relativos à área da visão computacional que são importantes para a compreensão deste projeto.
3. **Capítulo 3:** Neste capítulo é abordado o estado da arte do modelo de deteção YOLO, um dos principais modelos de deteção na área da visão computacional.
4. **Capítulo 4:** No 4º capítulo é feita uma referência aos recursos utilizados no desenvolvimento do projeto.
5. **Capítulo 5:** Neste capítulo os testes e o seus resultados são feitos e é discutido e avaliado o desempenho de cada modelo implementado.
6. **Capítulo 6:** No capítulo 6 é resumido todo o trabalho desenvolvido e abordado o trabalho futuro a desenvolver.

2 ENQUADRAMENTO TEÓRICO

2.1 Conceitos

2.1.1 Visão computacional

A visão computacional é o processo de modelação e replicação da visão humana usando *software* e *hardware*. Todo este processo de processamento visual é bastante complexo. Imaginando duas pessoas numa sala, e que uma delas atira uma bola e a outra pessoa agarra a mesma bola com as mãos [1]. Este processo parece bastante simples, e para nós, humanos, na generalidade dos casos é, mas os humanos têm a vantagem de vidas inteiras de contexto prático para treinar como distinguir objetos, a que distância eles estão, se estão a mover-se ou não, se há algo de errado numa imagem, entre outras situações. Tentar replicar estes casos numa máquina é um processo extremamente complexo, mas não impossível.

O método encontrado para replicar nas máquinas a forma como agem e pensam os seres humanos é a inteligência artificial, onde as máquinas podem imitar o comportamento humano e aprender com a sua experiência, realizando posteriormente tarefas de forma autónoma.

2.1.1.1 Como funciona a Visão computacional

A visão computacional sendo um campo da inteligência artificial permite que modelos de aprendizagem obtenham informações relevantes de imagens digitais, vídeos e outros elementos. Recolhendo grandes quantidades de dados, a visão computacional executa uma análise de dados de modo que o modelo de aprendizagem consiga identificar padrões, diferenciar objetos e identificar elementos visuais [2]. Portanto, a melhor forma de treinar uma máquina é "enchê-la" de imagens rotuladas, com milhares de imagens, milhões, se possível, e submetê-las às várias técnicas de *software* existentes, que permitam à máquina reconhecer padrões em todos os elementos existentes na imagem e assim aumentar a precisão. Duas das principais tecnologias usadas para reconhecimento de imagens são o *deep learning* e às redes neurais convolucionais.

O *deep learning* sendo um subconjunto da inteligência artificial, envolve o uso de redes neurais profundas, conseguindo aprender automaticamente sobre a interpretação de dados visuais, caso consiga receber um *dataset* suficientemente grande, onde seja possível distinguir elementos entre imagens semelhantes. Como o sistema auto-aprende, a tarefa de reconhecimento de imagens por parte do ser humano deixa de ser necessária.

As redes neurais convolucionais ajudam na compreensão nos modelos de *deep learning*, dividindo as imagens em pixéis que recebem rótulos. Estes rótulos são utilizados para realizar as convoluções (uma operação matemática em que duas funções reproduzem uma terceira função), aproveitando a função terciária para fazer

recomendações sobre os elementos que está a "observar" (ver Figura 1). A cada ciclo, a rede neural realiza convoluções e avalia a veracidade das suas recomendações.

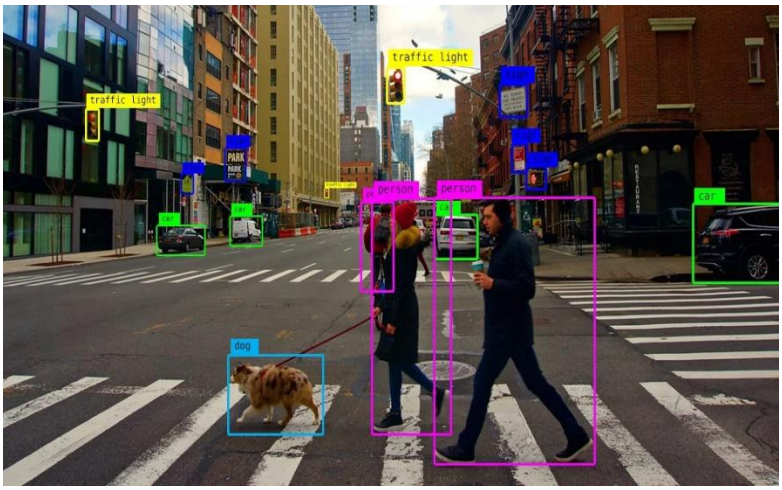


Figura 1 – Sistema de CV que deteta vários elementos numa imagem [3].

2.1.1.2 Técnicas de aprendizagem

As técnicas de aprendizagem são:

- **Classificação de imagem –** Esta técnica permite classificar ou rotular uma imagem baseada no seu conteúdo, utilizando os modelos de aprendizagem já mencionados anteriormente.
- **Localização –** Permite identificar a posição exata do elemento dentro da imagem ou vídeo. Essa localização é representada por uma caixa delimitadora (*Bounding Box*) (ver Figura 2).

Figura 2 – *Classification* + *Localization* [4].

- **Deteção de objetos (*Bounding Box Detection*)** – Esta técnica permite identificar e localizar elementos mais específicos numa imagem. Dando um exemplo de uma foto onde aparecem vários tipos de carros, utilizando a técnica de classificação de imagens só poderíamos rotular a imagem como

carros, com *Bounding Box Detection* é possível classificar todos os objetos individualmente e identificar os vários tipos de carros, como carrinhas, autocarros, camiões, etc (ver Figura 3).

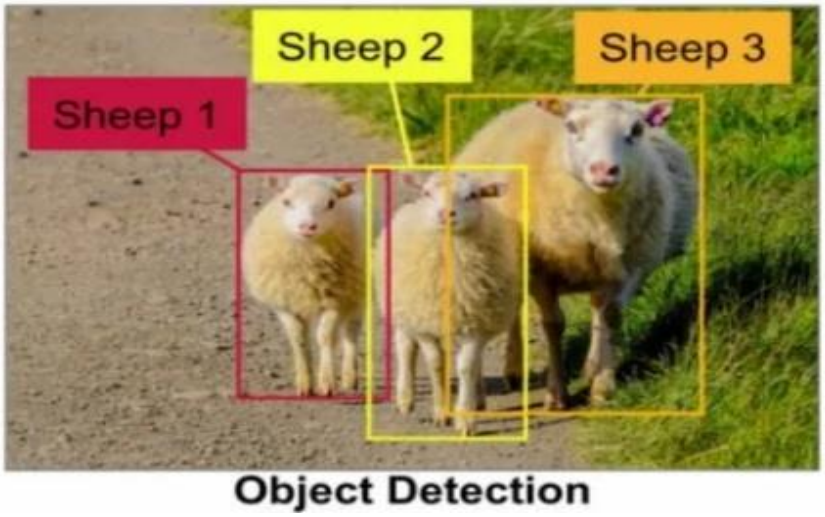


Figura 3 – Deteção de objetos [4].

- **Segmentação Semântica –** É uma técnica diferente das técnicas anteriores onde a *bounding box* é gerada. A segmentação semântica traça um limite ao redor de cada objeto, rotula cada pixel da imagem, criando um mapa de categorias de pixéis para toda a imagem (ver Figura 4).

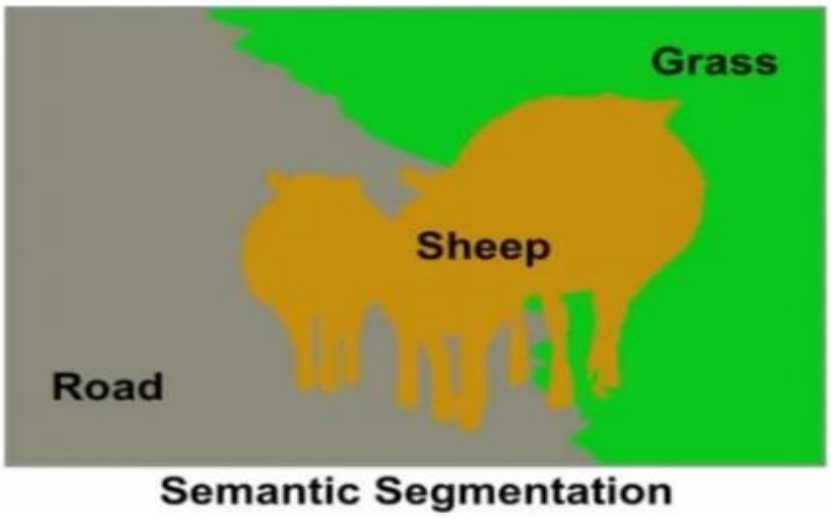


Figura 4 – Segmentação Semântica [4].

- **Segmentação de instâncias –** Está técnica em vez de atribuir os mesmos valores de pixel a todos os objetos da mesma classe, tenta segmentar e mostrar diferentes instâncias da mesma classe, atribuindo um ID único a cada classe, resultando numa imagem onde todos os objetos são separados por limites de pixéis (ver Figura 5).

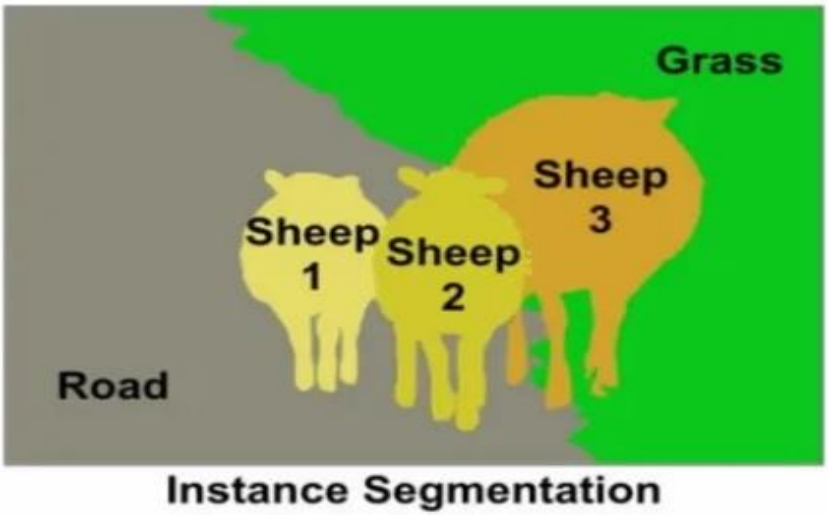


Figura 5 – *Instance Segmentation* [4].

2.1.1.3 Evolução da Visão Computacional

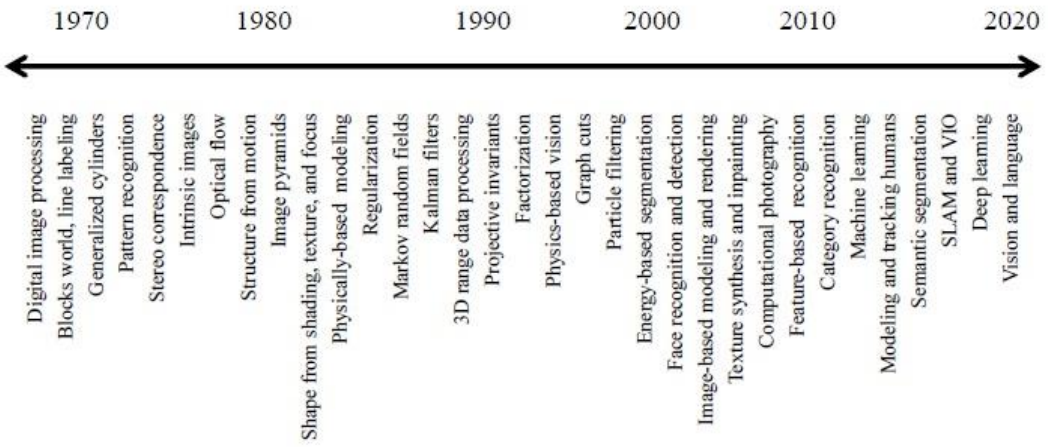


Figura 6 – Linha do tempo de algumas descobertas mais importantes relativas à visão computacional [5].

A Figura 6 ilustra o surgimento das várias técnicas em função do tempo.

Os primeiros sistemas de visão computacional surgiram na década de 60 e 70, com a primeira tecnologia a aparecer em 1959. Esta tecnologia de imagens permitiu que computadores digitalizassem e adquirissem imagens. Volvidos 4 anos, em 1963, já era possível os computadores transformarem imagens bidimensionais em tridimensionais. Em 1974 apareceu a introdução da tecnologia de reconhecimento ótico de caracteres (OCR – *Optical Character Recognition*), esta tecnologia tinha a capacidade de reconhecer qualquer texto impresso independentemente do tipo de letra ou tamanho, dentro de imagens, como fotos ou documentos digitalizados. [6]

Outra tecnologia semelhante, surgida na mesma altura foi o reconhecimento inteligente de caracteres (ICR – *Intelligent Character Recognition*), esta tecnologia também

permite reconhecer textos, mas usando redes neurais [7]. São duas tecnologias ainda hoje muito utilizadas para conversão de documentos impressos em papel para documentos digitais, reconhecimento de matrículas de viaturas, tradução automática de textos, entre outras.

Na década de 80 e 90, o foco dos investigadores concentrou-se no desenvolvimento de algoritmos em *machine learning* para visão computacional. Em 1982 o neurocientista David Marr tratava a visão como um sistema de processamento de informação e dividiu esse sistema em três níveis, nível computacional, algoritmo e físico, e introduziu algoritmos para máquinas detetarem bordas, cantos, curvas e formas básicas semelhantes. Ao mesmo tempo o cientista informático japonês Kunihiko Fukushima desenvolveu uma rede neural que permitia reconhecer padrões, chamada Neocognitron, que incluía camadas convolucionais em uma rede neural [8].

No início deste milénio o estudo dos investigadores da visão computacional estava focado no reconhecimento de objetos, e em 2001 dois investigadores do MIT, Paul Viola e Michael Jones apresentaram o primeiro algoritmo em *machine learning* para a deteção de faces em tempo real. Esta tecnologia ficou conhecida como *Viola-Jones Object Detection Framework*, sendo um dos maiores avanços desta época [9].

Em 2010 surgiu a primeira competição anual de reconhecimento visual, chamada *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) com o objetivo de promover o desenvolvimento de melhores técnicas de visão computacional e avaliar o estado da arte. A *ImageNet* contém milhões de imagens, rotuladas com mais de 21 mil classes e uma base de redes neurais convolucionais [10].

A ILSVRC conseguiu grandes proezas no reconhecimento de imagens reduzindo significativamente a taxa de erros por imagem. Algumas das técnicas mais marcantes (e vencedoras) utilizadas nesta competição foram:

- AlexNet – Vencedor do ano 2012 e desenvolvido por Alex Krizhevsky, Ilya Sutskever e Geoffrey Hinton, conseguiu uma taxa de erro de 15.3% [11].
- GoogLeNet/Inception – O vencedor de 2014 e desenvolvido pela Google.It, conseguiu uma taxa de erro de 6.67% [11].
- VGGNet – 2º classificado do ano 2014 e desenvolvido por Karen Simonyan e Andrew Zisserman, conseguiu uma taxa erro de 7.3% [11].
- ResNet – O vencedor do ano de 2015 e desenvolvido por Kaiming He, conseguiu uma taxa de erro de apenas 3.57%, superando o desempenho de nível humano neste *dataset* [11].

Outras tecnologias noutros contextos também foram desenvolvidas como a YOLO (You Only Look Once), Faster R-CNN, Mask R-CNN, entre outras, permitindo o crescimento das várias tecnologias e o aumento de eficácia no reconhecimento de imagens.

2.1.1.4 Aplicações da Visão Computacional

A visão computacional pode ser aplicada em variadíssimas áreas no mundo atual, tais como:

- Reconhecimento facial (Figura 7);

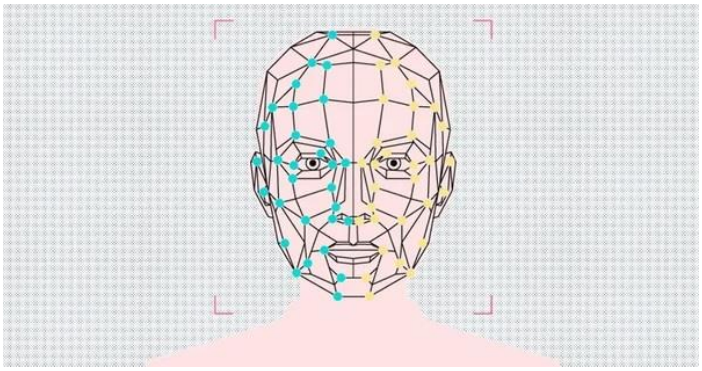


Figura 7 – Reconhecimento Facial [3].

- Realidade aumentada (Figura 8);

Figura 8 – Realidade aumentada [3].

- Veículos autónomos (Figura 9);

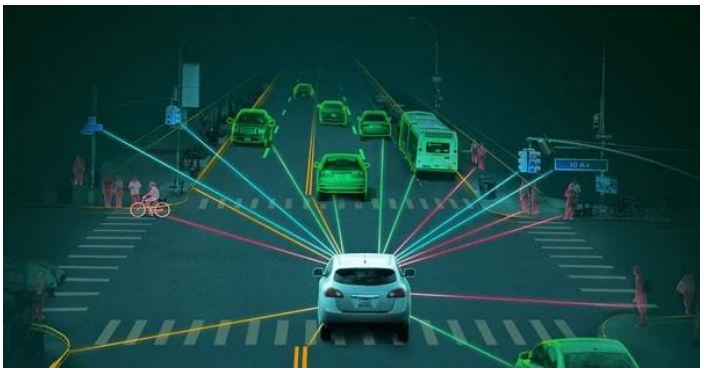


Figura 9 – Veículos autónomos [3].

- Saúde (Figura 10);

Figura 10 – Saúde [3].

- Agricultura (Figura 11);

Figura 11 – Agricultura [3].

- Gestão de conteúdos (Figura 12);

Figura 12 – Gestão de conteúdos [3].

2.1.2 Inteligência Artificial, *Machine Learning* e *Deep Learning*

Conforme mencionado no subcapítulo 2.1.1.1, a visão computacional é um campo da inteligência artificial, mais propriamente da *Deep learning*. Apesar de *Deep learning*, inteligência artificial e *machine learning* serem da mesma área e muitas vezes relacionarem ambas as palavras, a verdade é que os seus significados são diferentes.

A inteligência artificial é um campo da ciência da computação dedicado para a criação de sistemas que possam realizar tarefas, que caso fossem realizadas por um ser humano exigiria um certo grau de inteligência, como a capacidade de resolver problemas, compreensão de linguagem natural, reconhecimento de padrões, tomadas de decisões, etc. De forma abreviada, a inteligência artificial é o conceito de criar sistemas inteligentes.

Machine learning é uma subárea da inteligência artificial concentrada no desenvolvimento de algoritmos e modelos de computador que permitem que máquinas aprendam com dados a melhorarem o seu desempenho em tarefas específicas, não havendo a necessidade de as tarefas serem programadas. As *machine learning* conseguem aprender com exemplos e dados envolvendo algumas técnicas de aprendizagem como os modelos *K-nearest neighbors* e *Support Vector Machines*, que são chamados de modelos de aprendizagem Supervisionada – *Supervised Learning*, e modelos como o *K-Means Clustering* que são chamados de modelos de aprendizagem não supervisionada – *Unsupervised learning* [5].

Deep learning é uma subárea da *machine learning* baseada em redes neurais profundas, inspiradas na estrutura do cérebro humano. As redes neurais são compostas por múltiplas camadas de processamento capazes de aprender representações complexas de dados, como visão computacional, processamento de linguagem natural e reconhecimento de padrões (ver Figura 13).

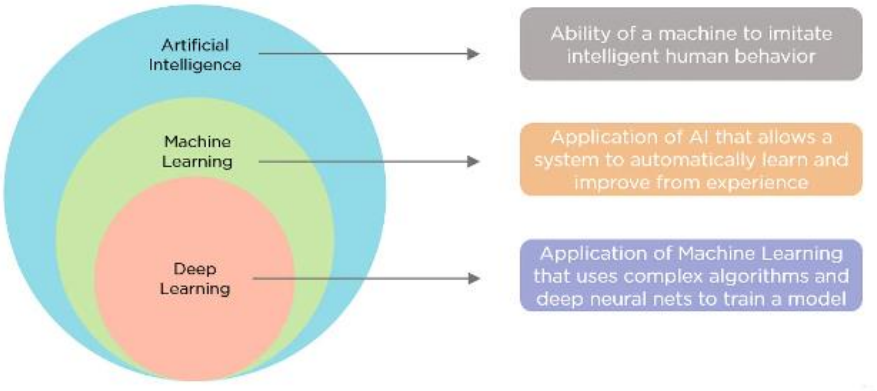


Figura 13 – Diferença entre os termos Inteligência Artificial, *Machine Learning* e *Deep Learning* [12].

2.1.2.1 K–Means Clustering

O *K-Means Clustering* é um algoritmo de *machine learning* não supervisionado, com o objetivo de agrupar dados semelhantes em conjuntos chamados de clusters [13]. Dando um exemplo prático de uma loja comercial onde o gerente pretende entender melhor os seus clientes de forma a poder oferecer os melhores descontos e ofertas. Classificar os clientes de forma manual é um processo difícil e moroso e é aqui que é aplicado o algoritmo *K-Means Clustering*. Baseado nas informações do cliente, o

algoritmo pode receber como variáveis de entrada a idade dos clientes, o seu sexo, quais os produtos mais comprados, frequência de compras etc. Tendo toda esta informação e aplicando o algoritmo *K-Means Clustering* é possível obter várias informações como, qual a preferência de compras da mulher ou do homem, qual a frequência de gastos de uma pessoa jovem e de uma pessoa de idade, entre outras, sendo depois possível diferenciar cada grupo e oferecer ofertas especiais a cada grupo de pessoas.

O funcionamento deste algoritmo inicia-se com a definição do número de agrupamentos ou conjuntos (*k clusters*), definido o número de *k clusters* é necessário inicializar os centroides, o centroide é o centro do *k cluster*. A etapa seguinte calcula a distância entre o ponto de dados y e o centroide x, usando a métrica de distância euclidiana.

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(xi - yi)^2}, \ \ em\ R^n$$

Após calcular todas as distâncias dos pontos a cada centroide, este serão agrupados no conjunto onde a distância ao centroide é mais curta. Em seguida os centroides são reinicializados calculando a média de todos os pontos de dados x de cada cluster.

$$Ci = \frac{1}{|Ni|}\sum xi\,, em\ R^n$$

A etapa seguinte é a repetição das duas etapas anteriores até que os centroides de cada cluster estejam na posição mais "central" de cada cluster [14] (Figura 14).

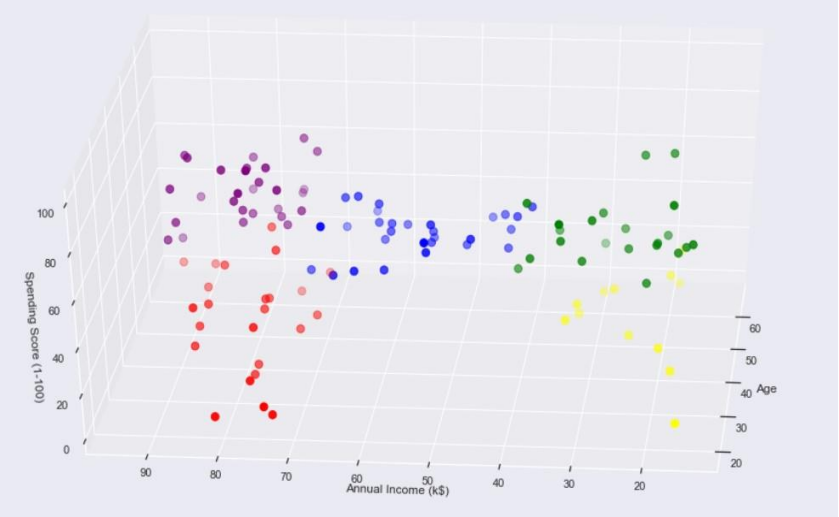


Figura 14 – Exemplo de um *dataset* obtido pelo algoritmo *K-Means Clustering* [15].

2.1.2.2 K–Nearest Neighbors – K-NN

O *K-Nearest Neighbors* é um algoritmo de *machine learning* supervisionado que tenta classificar cada amostra de um *dataset* avaliando a sua distância relativamente às amostras mais próximas (vizinhos mais próximos). Se os vizinhos pertencerem a uma determinada classe, a amostra em questão será classificada nessa categoria [16].

Dando um exemplo prático, a Figura 15 representada abaixo, apresenta dois grupos de classes: azul e vermelho. A amostra a cor verde ainda não está classificada e será necessário definir a qual classe pertence. De forma intuitiva pode-se dizer que pertence à classe azul, mas o algoritmo não possui intuição e necessita de um cálculo matemático para poder definir qual a sua classe. Se definirmos com base nos 4 vizinhos mais próximos (k=4) a amostra irá pertencer à classe azul, mas se definirmos com base nos 9 vizinhos mais próximos (k=9) amostra irá pertencer à classe vermelha. A classificação da amostra é definida com base na votação da maioria, neste caso, com 9 vizinhos mais próximos a classe dominante é a classe vermelha, com 4 vizinhos a classe dominante era a classe azul. Portanto, dependendo do valor de k, poderemos ter diferentes classificações.

Ao utilizar valores de k baixos o algoritmo pode ter alta variância, originando problemas de *overfitting* (*overfitting* ocorre quando o algoritmo apresenta um bom desempenho nos dados de treino, mas um fraco desempenho com dados nunca vistos), se utilizarmos valores de k altos, podemos ter baixa variância, podendo haver problemas de *underfitting* (*underfitting* ocorre quando o algoritmo tem dificuldades com os dados de treino e com novos dados) [17]. A escolha de k dependerá dos dados de entrada, pois dados com valores mais discrepantes provavelmente terão melhor resultado com valores de k altos.

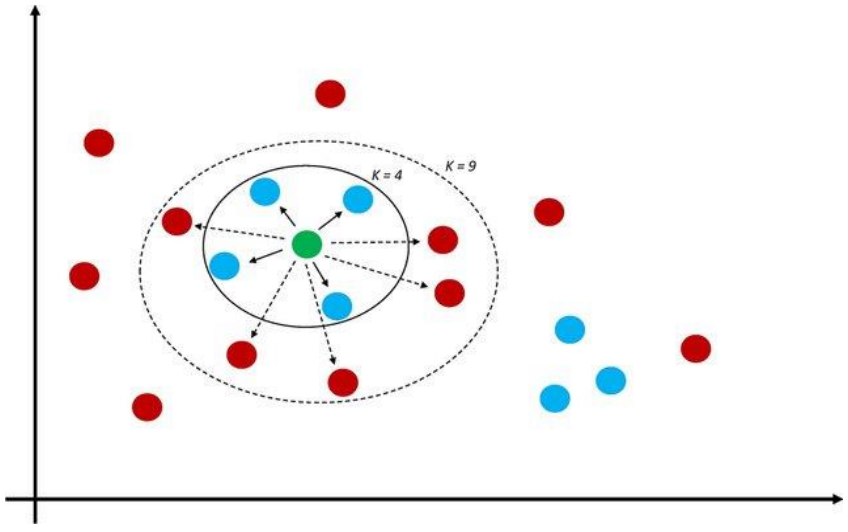


Figura 15 - Exemplo prático do algoritmo *K-Nearest Neighbors* [18].

2.1.2.3 Support Vector Machines – SVM

Support Vector Machines é um algoritmo de *machine learning* supervisionado usado para tarefas de classificação linear ou não linear, regressão ou deteção de *outliers*. O principal objetivo do algoritmo SVM é traçar um hiperplano num espaço N - dimensional que melhor separa as diferentes classes desse espaço dimensional maximizando a margem entre os pontos de dados pertencentes a cada classe.

A SVM é um algoritmo simples, mas com grande poder de precisão e ao mesmo tempo apresenta um nível de computação baixa. Devido à sua simplicidade, adaptabilidade e baixa computação, os SVMs são usados em várias tarefas como classificação de textos, classificação de imagens, deteção de spam, deteção facial, etc. [19]

Dando um exemplo prático do funcionamento do algoritmo SVM, supõe-se que existem duas classes, a classe azul e a classe vermelha, com vários pontos num espaço bidimensional com as coordenadas x e y. O algoritmo com os dados recolhidos gera o hiperplano que melhor separa as duas classes, sendo esta linha o limite de decisão, o melhor hiperplano é gerado maximizando as margens de ambas as classes, onde o ponto de dados mais próximo tem a maior distância ao hiperplano. Qualquer ponto de dados que caia para um dos lados será classificada de azul ou vermelha (Figura 16).

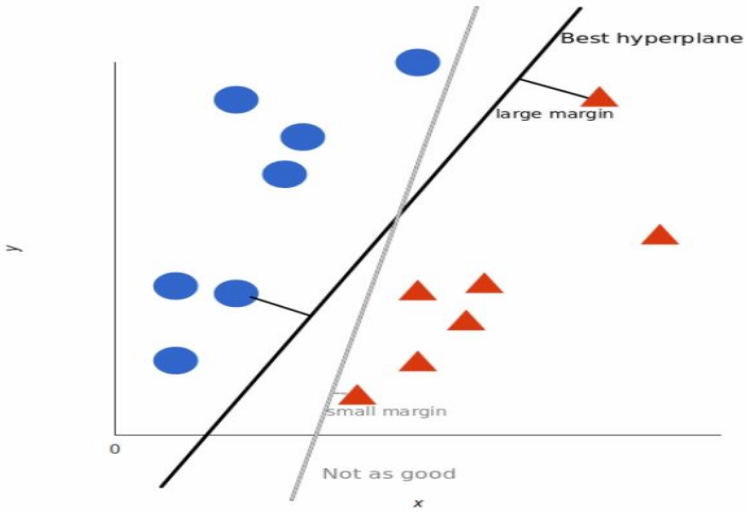


Figura 16 – Exemplo prático bidimensional de SVM [20].

2.1.2.4 *Artificial Neural Network* – ANN

Artificial Neural Network é um modelo inspirado a partir dos neurônios do cérebro humano. As redes neurais artificiais contêm neurônios artificiais que são chamados de unidades. Estas unidades juntas formam uma série de camadas que juntas constituem toda a rede neural artificial. Na generalidade uma ANN possui uma camada de entrada, camadas ocultas e camada de saída.

A camada de entrada recebe dados que a rede neural precisa para analisar e aprender, posteriormente passam por uma ou várias camadas ocultas que analisam as unidades recebidas da camada de entrada a fim de extrair todos os dados relevantes,

por fim, a camada de saída forma uma resposta com os dados obtidos a partir das camadas anteriores.

Existem alguns tipos de redes neurais artificias, sendo as mais conhecidas e mais utilizadas a rede neural *feedforward*, rede neural recorrente e a rede neural convolucional [21]:

- Rede neural *feedforward*: é uma rede onde os dados de entrada viajam apenas numa única direção.
- Rede neural recorrente: esta rede salva os resultados obtidos de camada e devolve esse resultado à entrada para prever melhor o resultado da camada. A 1º camada de rede neural recorrente é semelhante à rede neural *feedforward* e a rede neural recorrente inicia o processo assim que a saída da 1º camada é calculada, nas camadas seguintes, cada unidade lembrará algumas das informações das etapas anteriores para processar e reutilizá-las nas camadas seguintes. Se a previsão da rede estiver incorreta, o sistema aprende automaticamente e continua a trabalhar em direção à previsão correta durante a retro propagação.
- Rede neural convolucional: a rede neural convolucional é semelhante à rede neural *feedforward*. As conexões entre as unidades possuem pesos que determinam a influência de uma unidade em outra unidade. A rede neural convolucional será a tecnologia utilizada no sistema proposto do projeto. Na seção 2.1.3 aborda-se com mais detalhes as redes neurais convolucionais.

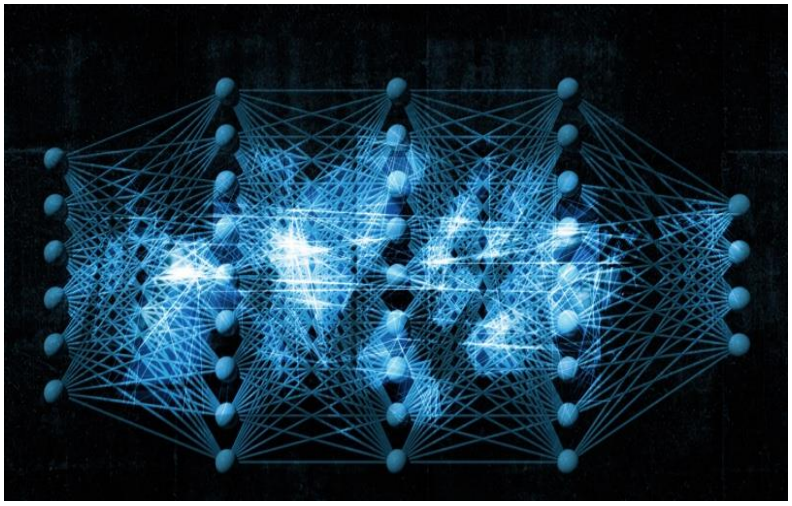


Figura 17 – Rede neural artificial [22].

2.1.3 Redes Neurais Convolucionais – CNN

As redes neurais convolucionais são algoritmos do *deep learning* que ajudam na compreensão de imagens, vídeos entre outros. A arquitetura de uma CNN é semelhante a uma rede de neurónios do cérebro humano. O cérebro humano no segundo que vê uma imagem processa uma grande quantidade de informações, isto porque, cada neurónio trabalha no seu campo recetivo e está conectado a outros neurónios que envolvem todo o campo visual. Na estrutura de uma rede neural convolucional a estrutura é igual. As CNN recebem uma imagem como entrada, e o algoritmo de forma organizada tenta identificar padrões como linhas, curvas, objetos, rostos de forma a conseguir detetar o que está na imagem.

2.1.3.1 Arquitetura de uma CNN

De uma forma básica uma rede neural convolucional começa com uma imagem de entrada, que passará por camadas convolucionais, camadas de *pooling* e camadas totalmente conectadas e acabará com uma função de saída que poderá ser do tipo *Softmax*, *Sigmoid*, *Bounding Box Regression*, entre outras que faz a previsão do que está a detetar na imagem.

Existem várias formas de organizar uma rede neural convolucional. Como o número de filtros usados nas camadas convolucionais, o número de camadas *pooling*, tipo de função de ativação, a forma de disposição e a quantidade dessas camadas é que diferencia as várias arquiteturas de uma rede neural convolucional em termos de desempenho e eficiência (Figura 18).

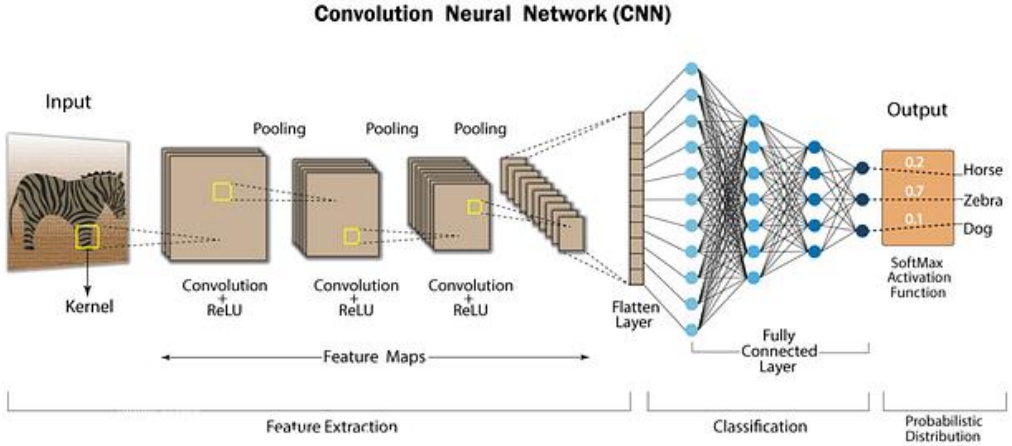


Figura 18 – Exemplo de uma arquitetura de uma rede neural convolucional [23].

Esta rede funciona da seguinte forma:

1. **Imagem de entrada** – A imagem de entrada numa rede neural convolucional é composta por uma matriz de valores de pixéis. Cada valor de pixel indica a sua tonalidade de cor. Numa imagem em tons de cinzento, cada pixel é representado por um único valor de 0 a 255, ou seja, de preto a

branco, enquanto numa imagem colorida, cada pixel é representado por um conjunto de valores, chamado RGB – *Red*, *Green* e *Blue* (Figura 19).

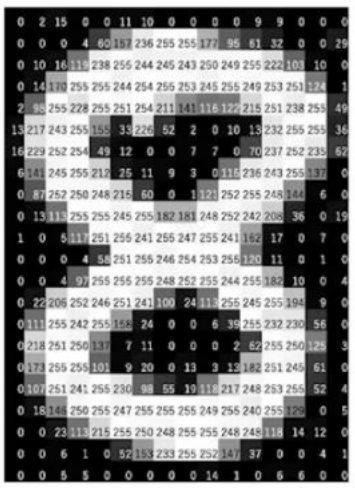


Figura 19 – Exemplo de imagem em tons de cinzento com os valores de cada pixel [24].

2. **Camada convolucional** – A camada convolucional é a camada principal de uma rede neural. As camadas convolucionais consistem num conjunto de filtros/*kernels*. Estes filtros são responsáveis por aprender e detetar características e padrões específicos nas imagens de entrada. Cada filtro produz um mapa de características que destaca a presença de um padrão específico na imagem de entrada.

   Existem vários tipos de filtros, com dimensões de 1x1xC, 3x3xC, 5x5xC, 7x7xC, entre outros C canal de profundidade. Dando um exemplo de uma imagem com dimensões 7x7x3, isto significa que a imagem de entrada tem uma altura de 7 unidades (pixéis), uma largura de 7 unidades (pixéis) e 3 canais de profundidade. Quando um filtro de 3x3x3 é aplicado é gerado um novo valor. Este valor é obtido através da convolução de uma região da imagem com o filtro, este valor é colocado numa nova matriz de valores. Este processo é feito em todos os pixéis da imagem. Com um canal de profundidade com valor 3, significa que serão gerados 3 mapas característicos.

   Para o filtro percorrer todas regiões da imagem é necessário definir o valor de *stride* (passo). O *stride* refere-se ao número de pixéis que o filtro se move ao longo das regiões da imagem. Se o *stride* tiver valor 1, significa que o filtro só se move 1 unidade de pixel por vez, se for 2, o filtro salta dois pixéis a cada passo e assim sucessivamente. O valor de *stride* afeta diretamente o tamanho de saída após a convolução, quanto maior o valor de *stride* menor será o tamanho da imagem e menor será a precisão do resultado (Figura 20).

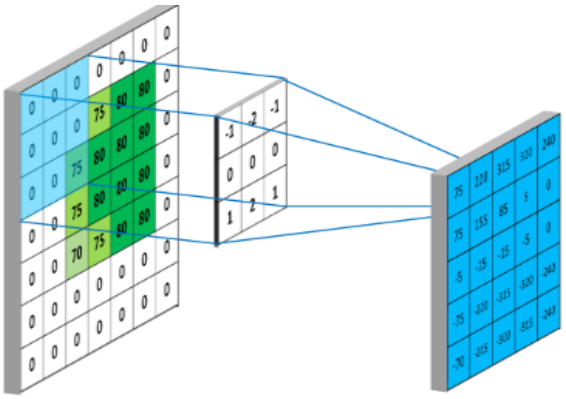


Figura 20 – Exemplo de uma camada convolucional com filtro [25].

3. ***Pooling*** – Camadas de *pooling* são usadas para reduzir as dimensões do mapa de características. Existem duas técnicas de *pooling*, uma delas é a *max pooling*, esta técnica extrai o valor máximo de uma determinada região do mapa característico, outra técnica utilizada é a *average pooling* que retorna a média de valores de uma determinada região da imagem (Figura 21).

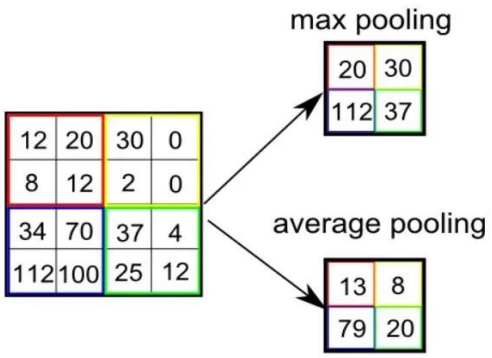


Figura 21 – Camada de *pooling* [25].

4. ***Dropout*** – A camada de *dropout* é usada para prevenir o *overfitting*. O *overfitting* é um fenômeno comum em redes neurais e ocorre quando o modelo se ajusta excessivamente aos dados de treino. Para evitar esse problema, é utilizada a camada *dropout*, onde alguns neurônios (unidades) são eliminados de forma arbitrária da rede neural durante o processo do treino e desta forma essas unidades não contribuem para o cálculo de saída. Isto é feito em cada etapa do treino com uma probabilidade fixa.

5. **Camadas totalmente conectadas** – A camada totalmente conectada conecta todos os neurônios da camada anterior a todos os neurônios da próxima camada. Esta camada é responsável por combinar recursos aprendidos pelas camadas anteriores a fim de realizar a previsão final da imagem.

6. **Função de ativação** – As funções de ativação são responsáveis por introduzir não linearidades nas redes neurais, permitem aprender relações complexas entre dados e são aplicadas nas saídas de cada neurônio. Existem vários tipos de operações não lineares, como a Sigmóide, a Tanh, ReLU, *Softmax* entre outras. A função Sigmóide mapeia os valores de entrada num intervalo entre 0 e 1 e é frequentemente usada em camadas de saída para problemas de classificação binária. A função Tanh é semelhante à função Sigmóide, mas mapeia valores de entrada entre -1 e 1. A função ReLU retorna o valor de entrada se for positivo e zero caso seja negativo. É bastante usada devida à sua eficiência computacional e capacidade de aprendizagem. A função *Softmax* é usada para classificações de multiclasse. Transforma um vetor de valores em uma distribuição de probabilidade, onde a soma de todas as saídas é igual a 1 [26].

7. ***Padding*** – É uma técnica de preenchimento que serve para ajustar as dimensões das imagens de entrada e assim preservar os valores das bordas da imagem durante aplicação das camadas convolucionais (Figura 22).

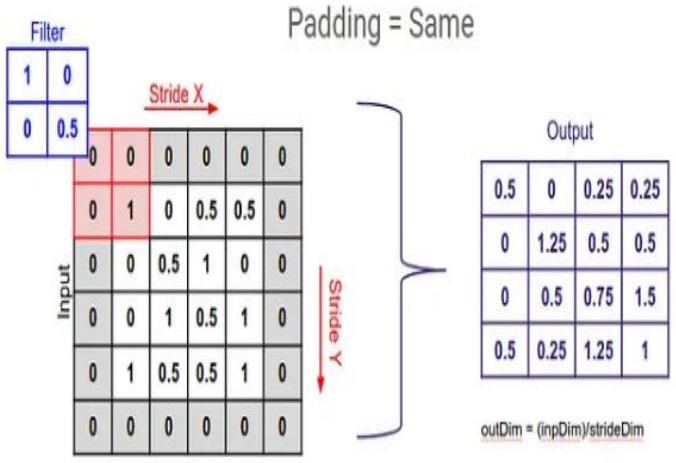


Figura 22 – Exemplo de *Padding* [27].

2.1.4 Arquiteturas de CNNs mais conhecidas

Algumas redes neurais convolucionais já foram mencionadas no subcapítulo 2.1.1.3, mas neste subcapítulo serão apresentadas as suas arquiteturas, a forma como se organizam e quão importantes foram e são na área da visão computacional.

2.1.4.1 LeNet-5

O modelo LeNet-5 foi a primeira arquitetura de uma rede neural convolucional. Foi desenvolvida em 1998 por Yann Lecun com o objetivo de identificar dígitos escritos à mão. Este modelo consiste em 7 camadas. A primeira camada consiste numa imagem de entrada de 32x32x1, em seguida é convolvida numa camada convolucional com 6 filtros de tamanho 5x5 e *stride* de 1, dessa operação resulta uma *feature map* de 28x28x6. A segunda camada convolucional resulta numa operação de *average pooling* com um tamanho de 2x2 e *stride* de 2, resultando uma *feature m*ap de 14x14x6. A terceira convolução envolve 16 filtros com tamanho de 5x5 e *stride* de 1, originando uma imagem de 10x10x16. A quarta camada é uma operação de *average pooling* com 16 filtros de 5x5 e *stride* de 1, resultando numa imagem reduzida de 5x5x16. A quinta e última camada convolucional está totalmente conectada com 120 filtros com tamanho de 5x5. Nesta camada, cada uma das 120 unidades será conectada às 400 (5x5x16) unidades da camada anterior e a sexta camada também é totalmente conectada com 84 unidades. A sétima e última camada do modelo tem uma saída *Softmax* com 'n' possibilidades de classes (Figura 23).

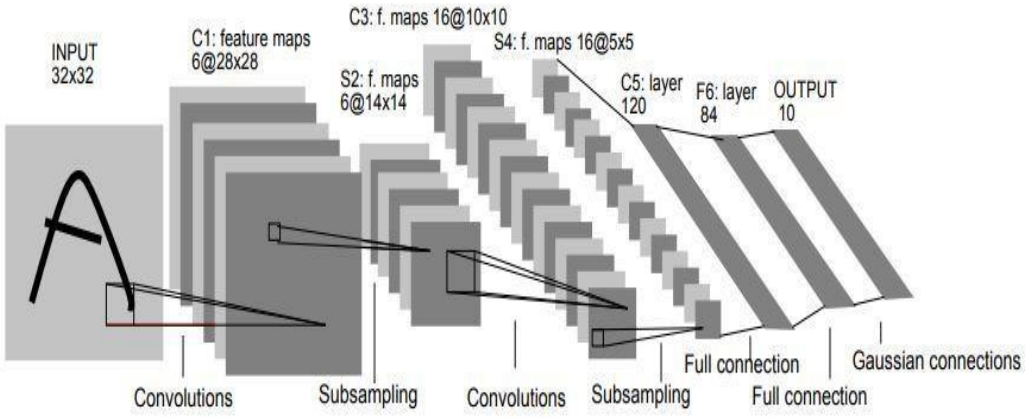


Figura 23 – Arquitetura LeNet-5 [28].

2.1.4.2 AlexNet

O modelo AlexNet foi desenvolvido por Alex Krizhevskt em 2012 para competir na competição ILSVRC. A arquitetura do AlexNet é bastante semelhante ao modelo LeNet-5, mas mais profunda e com mais camadas convolucionais empilhadas umas sobre as outras. AlexNet é composto por 5 camadas convolucionais, com combinações de camadas *max-pooling*, 3 camadas totalmente conectadas e 2 camadas *dropout*. A relu, função de ativação, é utilizada em todas as camadas. A função de ativação utilizada na camada de saída é a *Softmax*. Possui cerca de 60 milhões de parâmetros. A AlexNet implementou novas técnicas como a funçao de ativação ReLU *Nonlinearity*, permitindo um tempo de treino mais rápido relativamente à arquitetura LeNet. Outra nova técnica implementada foi o treino multi-GPU, a multi-

GPU permite dividir os neurónios do modelo em várias GPUs e assim o modelo consegue reduzir o tempo de treino [29].

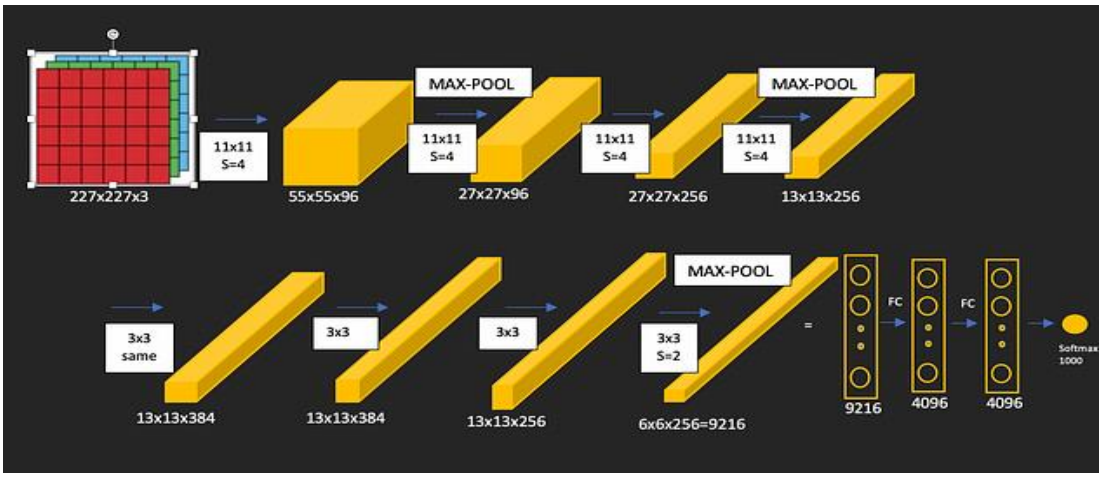


Figura 24 – Arquitetura AlexNet [29].

2.1.4.3 VGGNet

A arquitetura VGGNet foi desenvolvida em 2014, para também, competir na competição ILSVRC ficando em segundo lugar na competição. Este modelo possui várias arquiteturas, sendo a VGGNet-16 uma das arquiteturas disponíveis. A VGGNet-16, possui 16 camadas convolucionais que vão até 95 milhões de parâmetros, utilizando vários filtros como o *max pooling* e *padding*, terminando com uma camada de função de ativação *Softmax*. Esta arquitetura apresenta um bom desempenho comparativamente a outras arquiteturas, sendo também uma das arquiteturas mais utilizadas (Figura 25).

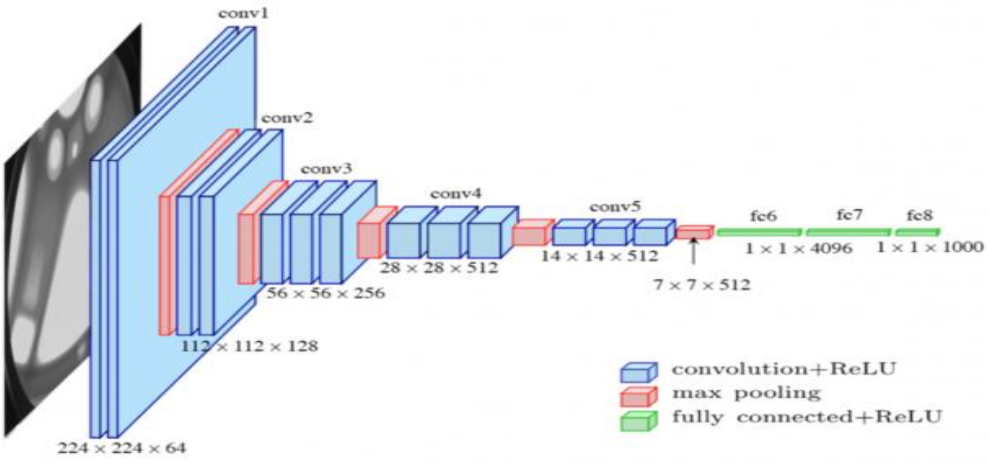


Figura 25 – Arquitetura VGG16 padrão [28].

2.1.4.4 *Inception* (*GoogleNet*)

Como o próprio nome indica, esta arquitetura foi desenvolvida pela *GoogleNet* e alcançou o primeiro lugar na competição ILSVRC em 2014. Este modelo conseguiu

alcançar uma arquitetura mais profunda com uma série de técnicas inovadoras, incluindo uma convolução com um filtro de 1x1, permitindo a combinação de convoluções com diferentes tamanhos em paralelo. Esta técnica permite reduzir as convoluções maiores com a diminuição da dimensão do mapa de características e, posteriormente, passar para convoluções maiores (Figura 26) [28].

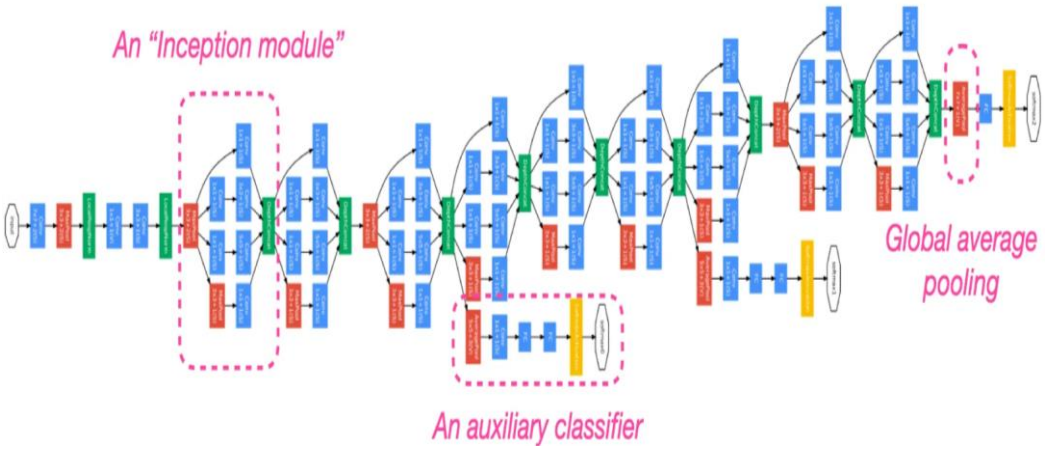


Figura 26 – Arquitetura *GoogleNet* [28].

2.1.4.5 *Inception* - ResNet

O modelo Resnet foi o vencedor da competição ILSVRC em 2015, e foi desenvolvido por Kaiming He e inspirada na arquitetura *Inception*. É uma rede com 152 camadas, com mais de um milhão de parâmetros. Desenvolvido essencialmente para tarefas de classificação de imagens, a ResNet também pode ser usada para resolver problemas de processamento de linguagem natural, como por exemplo a conclusão de frases (Figura 27).

Figura 27 – Arquitetura Resnet [30].

2.1.4.6 *Detectron*

O modelo open-source *Detectron* foi desenvolvido pelo *Facebook AI Research* (FAIR) em 2018, com a sua arquitetura a ser baseada em alguns modelos conhecidos, como a ResNet, ResNexT e VGG16. Pouco tempo depois, a FAIR lançou uma nova versão chamada Detectron2, com um algoritmo mais simples de utilizar e muito mais rápido. Sendo um algoritmo simples e bastante flexível, com o Detectron2 é possível utilizar vários modelos diferentes, como modelos mais adaptados para detetar objetos e outros modelos mais adaptados para segmentação de imagens (Figura 28).

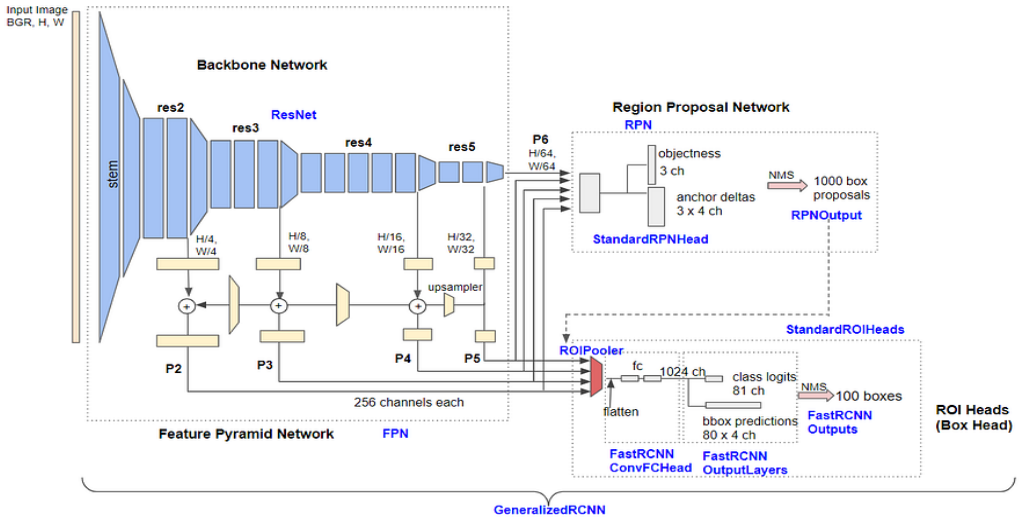


Figura 28 – Arquitetura *Detectron* baseada em RCNN-FPN [31].

2.1.4.7 Yolo – *You Only Look Once*

A primeira versão da YOLO foi lançada em 2015 e criada por Joseph Redmon, Ali Farhadi, Ross Girshick e Santosh Divvala. É uma das arquiteturas CNNs mais conhecidas, tornando-se um estado de arte para aplicações de visão computacional para deteção e classificação de objetos em tempo real. A sigla YOLO significa "Você só olha uma vez", e como o próprio nome indica, o objetivo do YOLO é detetar o objeto na primeira vez que olha para a imagem.

Desde 2015 foram lançadas várias versões, sendo a última conhecida a versão YOLO v8, lançada em 2023. Algumas das razões para a popularidade da YOLO é a sua rapidez, precisão, versatilidade e simplicidade. Sendo um algoritmo open-source, a comunidade está constantemente a melhorar o modelo, fazendo com que desde o seu lançamento tenha havido imensas melhorias. No capítulo 3 será detalhado como funciona a arquitetura YOLO e todas as versões lançadas até 2023 (Figura 29).

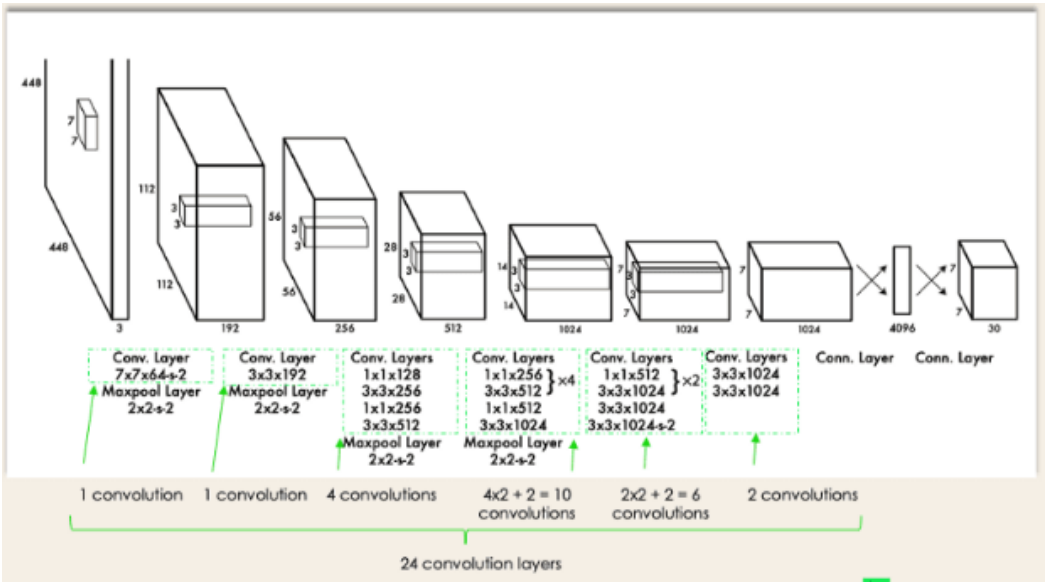


Figura 29 – Exemplo de uma Arquitetura YOLO [32].

2.1.4.8 MobileNetV1

O modelo MobileNet foi desenvolvido para classificar ou detetar imagens em dispositivos móveis. Foi desenvolvido por Andrew G Trillion em 2017. É uma arquitetura CNN bastante pequena o que torna o modelo muito fácil de usar em dispositivos com uma capacidade de processamento mais baixa como telemóveis, tablets ou drones. Conta com 28 camadas e é composto por camadas convolucionais, funções de ativação ReLU, camadas de *max-pooling*, terminando com uma camada totalmente conectada e uma função de ativação *Softmax* (Figura 30).

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figura 30 – Aquitetura MobileNet [33].

2.2 Conclusão

Neste capítulo apresentaram-se vários conceitos literários essenciais que oferecem uma base sólida para a compreensão da área de visão computacional e a sua ligação à inteligência artificial e *deep learning*. Foram apresentadas uma série de técnicas clássicas incluindo as arquiteturas modernas de redes neurais, fundamentais para o desenvolvimento e implementação de sistemas inteligentes capazes de interpretar e extrair informações valiosas a partir de dados visuais.

3 YOLO ESTADO PRESENTE

Como mencionado no subcapítulo 2.1.2.7, o YOLO é um algoritmo capaz de detetar objetos em tempo real de forma rápida e precisa. O YOLO é capaz de prever qual o objeto presente na imagem de forma precisa e identificar também qual a sua localização delimitando uma caixa no objeto encontrado.

A Figura 29 é uma arquitetura YOLO composta por 24 camadas convolucionais, 4 camadas de *max pooling* e 2 camadas totalmente conectadas. Inicialmente a arquitetura redimensiona a imagem de entrada para 448x448, antes de chegar à primeira camada convolucional, na primeira camada a imagem é transformada num tensor de 7x7x30. O tensor é responsável por fornecer as informações da coordenada da caixa delimitadora e a distribuição de probabilidades sobre todas as classes rotuladas para o sistema proposto [32].

3.1 Como funciona o modelo YOLO

O sistema YOLO divide a imagem de entrada numa grelha de SxS, onde cada célula da grelha fica responsável por prever B caixas delimitadores (*bouding box*) e prever a classe do objeto apresentando uma pontuação de confiança (Figura 31). A pontuação de confiança reflete o quão confiante está o modelo relativamente à sua previsão. Senão existir nenhum objeto na célula a pontuação de confiança deverá ser zero, caso contrário, a pontuação de confiança deverá ser igual à intersecção sobre união entre a caixa prevista e a região verdadeira (*Ground truth Region*) [34].

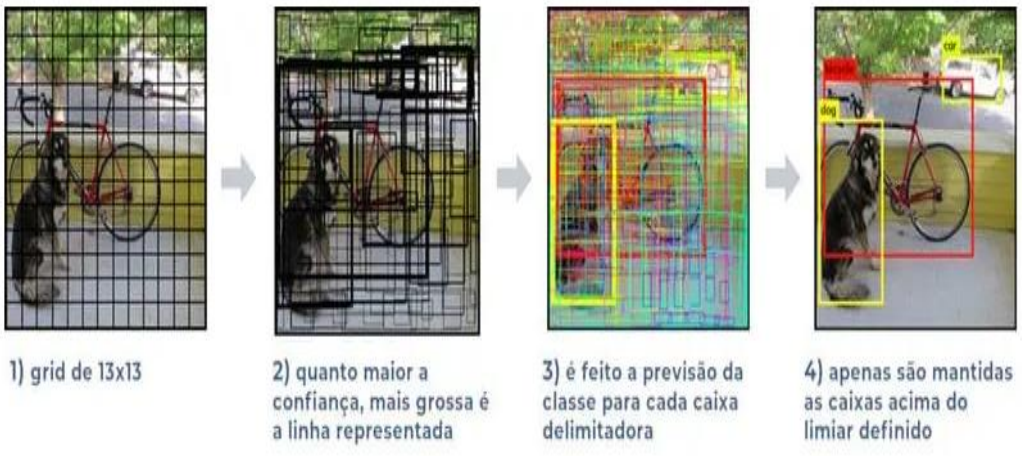


Figura 31 – Previsão de caixas delimitadoras do YOLO [34].

A determinação das caixas delimitadoras é feita com base numa previsão e o YOLO pode prever imensas caixas delimitadoras, mas nem todas estão corretas. Para determinar quais as caixas delimitadoras que melhor distinguem os objetos na

imagem, o YOLO determina os atributos dessas caixas delimitadoras usando um módulo de regressão composto por 5 atributos: pc, bx, by, bh, bw (Figura 32), sendo:

- pc – Representa a pontuação de confiança para a probabilidade de conter um determinado objeto;
- bx e by – Representam as coordenadas do centro da caixa delimitadora em relação à célula da grelha envolvente;
- bh e bw – Representa a altura e a largura da caixa delimitadora relativamente à célula da grelha envolvente.

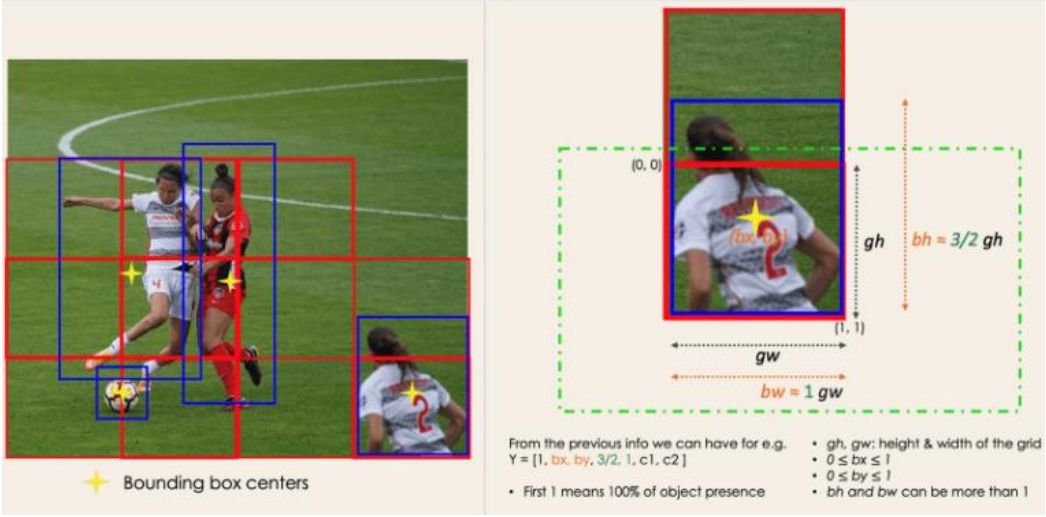


Figura 32 – Imagem ilustrativa dos atributos das caixas delimitadoras presentes numa imagem YOLO [32].

Para obter a pontuação de confiança é feito um cálculo de área entre as caixas delimitadoras previstas e a caixa delimitadora verdadeira (*Ground truth region*). Primeiro delimita-se a área de intersecção entre as duas caixas delimitadores correspondentes e verifica-se qual o valor da área de intersecção, de seguida, calcula-se a área total entre as duas caixas delimitadoras. Tendo estes dois valores conhecidos, a área de intersecção será dividida pela área total das caixas delimitadoras, dividido a duas regiões, o valor fornecido dá a estimativa de quão próxima a caixa delimitadora prevista está da caixa delimitadora verdadeira. Este resultado é conhecido como IOU – *Interception Over Union*. O resultado de IOU deverá estar entre 0 e 1. O objetivo do IOU é poder definir um limite e descartar caixas delimitadoras previstas com valores abaixo desse limite, considerando apenas caixas com valores superiores ao limite definido (Figura 33) [35].

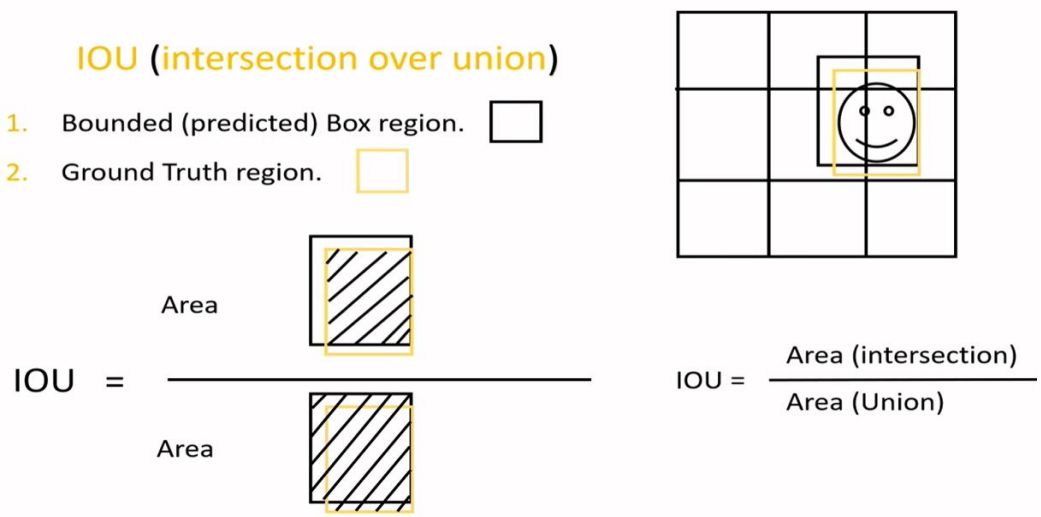


Figura 33 – Métrica de desempenho IOU – *Intersection Over Union* [36].

Definir o limite do IOU pode não ser suficiente para identificar corretamente as caixas delimitadoras verdadeiras, isto porque um objeto pode ter várias caixas IOU acima do limite. Para contornar esse problema usa-se o *Non-Max Supression* – NMS, o NMS mantém apenas as caixas com a maior pontuação de confiança (Figura 34).

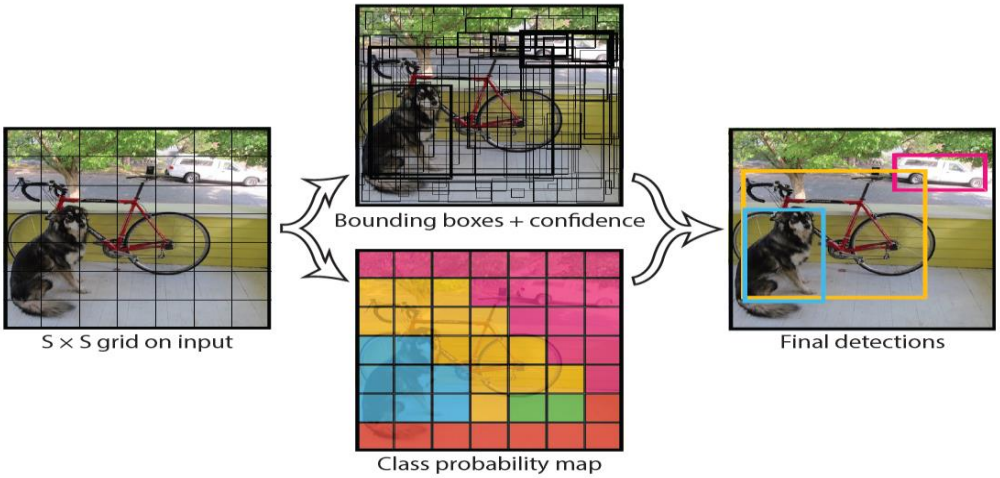


Figura 34 – Modelo de Deteção YOLO [35].

3.2 Arquiteturas YOLO

A Figura 35 apresenta a evolução das várias versões da rede YOLO e os instantes temporais onde surgiram.

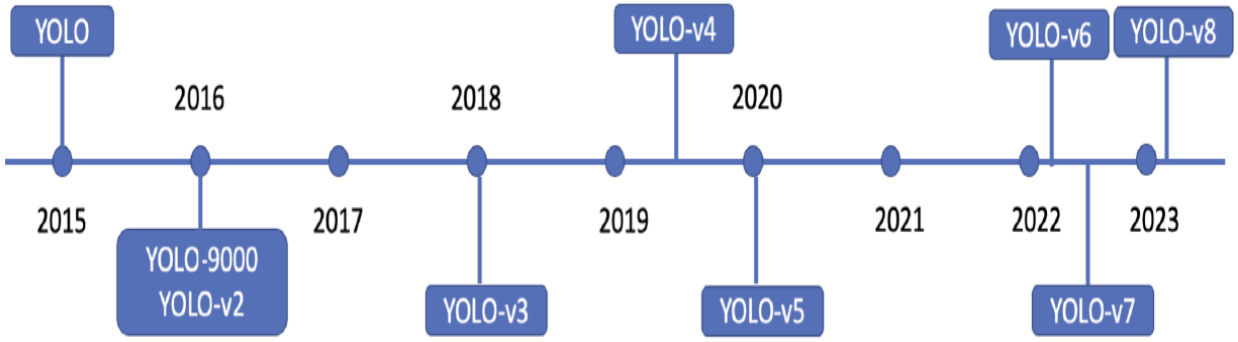


Figura 35 – Linha do tempo do YOLO [37].

3.2.1 YOLO: *Unified*, a 1ª versão

A primeira versão do YOLO foi lançada em 2015 por Joseph Redmon, Santosh Divvala, Ross Girshick e Ali Farhadi. A arquitectura foi inspirada no modelo GoogleNet de classificação de imagens. O YOLOv1 é composta por 24 camadas convolucionais, estas camadas utilizam convoluções de 1x1 seguidas por convoluções de 3x3 a fim de reduzir o tamanho da imagem, 4 camadas de *max-polling* e 2 camadas totalmente conectadas. Ver Figura 29 [35].

A arquitetura utiliza a *framework* Darknet que foi treinada no *dataset* ImageNet-1000. Utilizar a estrutura da Darknet traz vantagens para o modelo, como a eficiência de dedução, dá suporte à GPU (a Darknet utiliza a arquitetura CUDA para aproveitamento do processamento paralelo oferecido pelas GPUs da NVIDIA, aumentando de forma significativa a inferência e o treino em comparação com o processamento na CPU), facilidade de uso e capacidade de treinar modelos personalizados [38].

Os autores do YOLOv1 também criaram uma versão mais rápida, chamada Fast YOLO. O Fast YOLO é uma versão com menos camadas convolucionais, são 9 em vez de 24 e usa filtros mais pequenos, reduzindo assim a carga computacional durante a inferência, tornando este modelo muito mais rápido que a versão original. O YOLOv1 é capaz de detetar objetos em tempo real a uma velocidade de 45 imagens por segundo, enquanto o Fast YOLO é capaz de detetar objetos a uma velocidade de 155 imagens por segundos. Embora seja um modelo mais rápido a detetar imagens, é mais propenso a cometer erros, diminuindo a sua precisão relativamente ao modelo original.

A sua avaliação de desempenho no *dataset* PASCAL VOC alcançou um mAP50 de 63.4%, enquanto o Fast YOLO atingiu os 52.7% [39].

3.2.2 YOLO9000 ou YOLOv2

YOLO9000 ou YOLOv2 foi lançada em 2016 pelos mesmos criadores do YOLOv1, Joseph Redmon e Ali Farhadi.

Como o próprio nome indica, a versão número 2 do YOLO foi projetada para melhorar a precisão, tornar a deteção de objetos mais rápida e ser capaz de detetar uma maior variedade de classes de objetos. Algumas das principais mudanças do YOLO para a YOLOv2 foram:

- Normalização em lote: Ajuda a melhorar a precisão e estabilidade do modelo. Ao adicionar a normalização em lote em todas as camadas convolucionais, obtém-se uma melhoria de 2% no MAP - *Mean Average Precision*. Devido a introdução da normalização em lote foi possível remover o *dropout* do modelo sem causar danos de *overfitting* [40].

- Classificador de resolução mais alta: O YOLOv1 treina a rede com uma imagem de entrada de 224x224 e no momento de deteção obtém uma resolução de imagem de até 448x448. Isto faz com que o modelo se ajuste a uma nova resolução que causa uma diminuição no valor mAP. O YOLOv2 treina com uma resolução de imagem de 448x448 durante 10 épocas em dados ImageNet dando tempo à rede para ajustar os filtros para uma resolução mais alta. Este aumento no tamanho de imagem melhorou significativamente o valor de MAP em 4% [40].

- Caixas de âncora: No YOLOv2 as camadas totalmente conectadas foram removidas e as caixas de âncora foram introduzidas para prever as caixas delimitadoras. As caixas de âncora utilizam a tecnologia *k-means clustering* para determinar um *dataset*. Este novo algoritmo permite que a YOLO tenha uma variedade mais ampla de tamanhos dos objetos. Embora o MAP com a introdução das caixas de âncora tenha diminuído, o *Recall* aumentou, o que significa que o modelo tem espaço para melhorar [40].

- Recursos refinados: Um dos principais problemas do YOLOv1 era a deteção de objetos pequenos na imagem, no YOLOv2 esse problema foi resolvido com a divisão do mapa de características em 13x13, permitindo que o YOLOv2 seja mais capaz de localizar objetos menores na imagem e seja mais eficaz na deteção de objetos maiores [40].

- Treinos em multiescala: Uma fraqueza no YOLOv1 é a deteção de objetos com tamanhos de entradas diferentes, ou seja, o YOLOv1 treinado com

imagens de pequenas dimensões para a deteção de um determinado objeto, teria problemas em detetar o mesmo objeto caso a imagem tivesse um tamanho maior. No YOLOv2 este problema foi resolvido treinando o modelo com diferentes dimensões, variando entre 320x320x a 608x608. Isto permitiu que o modelo aprenda a prever objetos com vários tamanhos diferentes com maior precisão [40].

- Darknet 19: No modelo YOLOv2 foi introduzido a estrutura Darknet 19 com 19 camadas convolucionais, 5 camadas de *max polling* e uma camada de *Softmax*. Com a introdução do Darknet 19 a deteção para além de ser mais rápida, acaba também por ser mais precisa (Figura 36) [40].

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figura 36 – Darknet 19 [40].

Em termos de desempenho, a YOLOv2 conseguiu atingir um mAP50 de 76.8% a 67 FPS e 78.6% a 40 FPS [40].

3.2.3 YOLOv3

A terceira versão do YOLO foi lançada em 2018 com o mesmo objetivo da versão anterior, aumentar a precisão da deteção de objetos e aumentar a velocidade de deteção.

Uma das principais novidades do YOLOv3 foi a introdução do conceito de redes de pirâmide de recursos (FPN - *Feature Pyramid Network*), com recurso a este conceito é possível realizar três previsões em escalas diferentes para cada local na imagem de

entrada, ajudando o modelo a detetar com melhor desempenho a deteção de objetos pequenos na imagem.

Outra das principais novidades é a implementação da arquitetura Darknet 53 (Figura 37). Esta rede possui 53 camadas convolucionais, com filtros 3x3 e 1x1. É uma rede mais profunda que a Darknet 19 utilizada na versão 2 do YOLO e também possui ligações de atalho entre camadas [41].

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figura 37 – Darknet 53 [41].

Outra melhoria é relativa às caixas de âncora com diferentes escalas e proporções. No YOLOv2 todas as caixas de âncora possuem o mesmo tamanho, o que limitava a capacidade do algoritmo em detetar objetos de diferentes tamanhos, no YOLOv3 as caixas de âncora são dimensionadas e as proporções são variadas para corresponder melhor ao tamanho e à forma dos objetos que estão a ser detetados [41].

Outra diferença relativamente ao YOLOv2 foi a inclusão de classificadores logísticos para cada classe em vez de usar o *Softmax* utilizado na versão anterior. Com classificadores é possível obter mais que uma classe de objetos.

No conjunto de *dataset* PASCAL VOC o YOLOv3 obteve um mAP50 de 87.54% a uma velocidade de 30 FPS (Figura 38) [42].

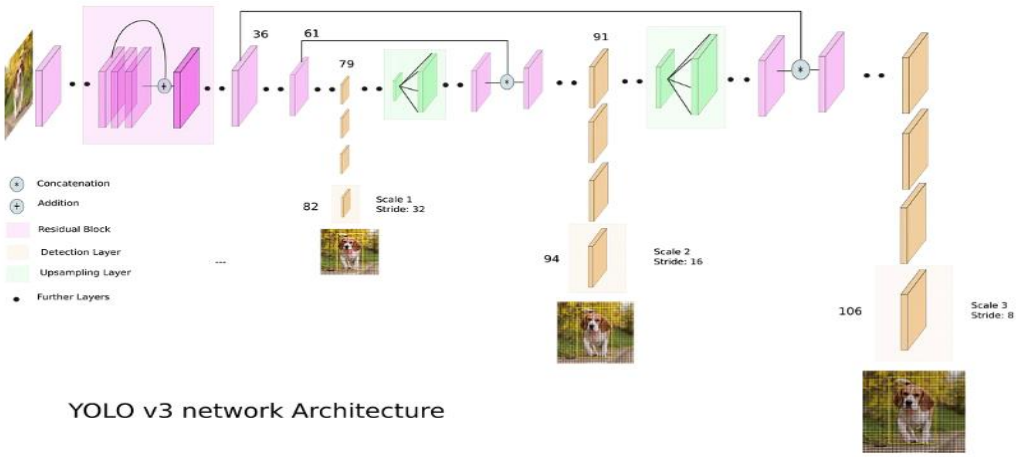


Figura 38 – Arquitetura YOLOv3 [43].

3.2.4 YOLOv4

O YOLOv4 é a quarta versão do modelo YOLO e foi desenvolvido por Alexey Bochkovskiy, Chien-Yao Wang e Hong-Yuan Mark Liao e lançada em 2019. A YOLOv4 apresenta melhorias relativamente ao seu antecessor, como a velocidade de inferência e precisão, é mais eficiente no processamento em GPUs, pois utiliza menos memória, mas a principal diferença para com o YOLOv3 é a nova arquitetura *backbone* implementada. Numa fase experimental os autores experimentaram 3 *backbones*: CSPResNext-50, CSPDarknet-53, e EfficientNet-B3, acabando por optar pela *backbone* CSPDarknet53 – *Cross Stage Partial Darknet53* com cerca de 29 camadas convolucionais e filtros 3x3 [39] [44].

A sua avaliação de performance no conjunto de *dataset* MS COCO atingiu um mAP de 43.5% a uma velocidade de 63 FPS (Figura 39) [45].

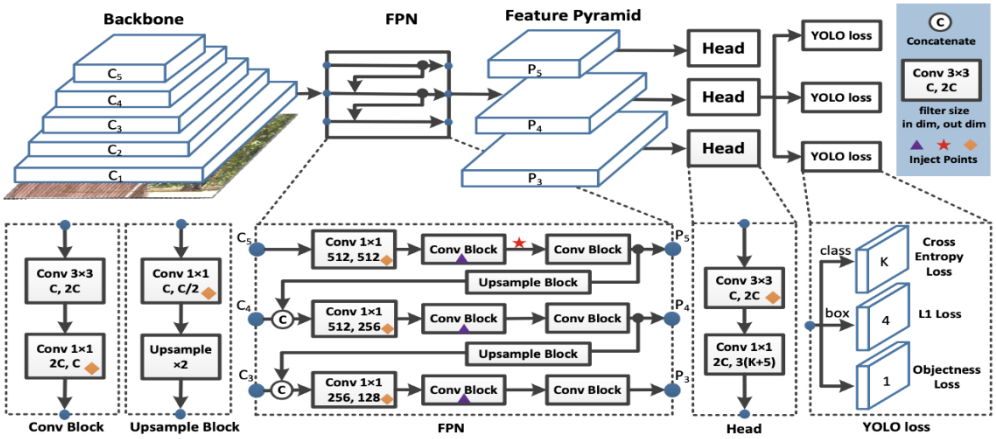


Figura 39 – Arquitetura YOLOv4 [46].

3.2.5 YOLOv5 da Ultralytics Inc.

O YOLOv5 foi lançado em junho de 2022 como um projeto *open source* pela empresa Ultralytics, e ainda hoje é mantido pela Ultralytics com atualizações regulares. A sua arquitetura é semelhante à versão YOLOv4 mas com melhorias significativas na velocidade, precisão e simplicidade.

Uma das principais novidades nesta nova versão é a utilização da biblioteca PyTorch. Ao contrário das versões anteriores que foram escritas na linguagem C, o YOLOv5 foi escrito em *Python*, tornando o modelo mais simples de usar e capaz de ser integrado com outros projetos que utilizam a biblioteca PyTorch. [47]

Nesta nova versão foram introduzidos novos procedimentos de treino como o aumento de dados durante o treino para expor o modelo a uma gama mais ampla de variações semânticas do que o conjunto treinado isoladamente. Este processo é uma grande ajuda quando há uma quantidade limitada de dados para treinar. Outras alterações foram implementadas, como a diminuição da arquitetura tornando o modelo mais leve, suporte para novas arquiteturas de base e mais eficientes como CSPDarknet53, CSPResNeXt50 entre outros [48].

O modelo YOLOv5 também possui 4 variantes chamadas de YOLOv5s, YOLOv5m, YOLOv5l e YOLOv5x (Figura 40, Figura 41 e Figura 42). Estas 4 variantes possuem diferentes configurações que impactam no número de parâmetros, na capacidade de deteção, na velocidade do treino e na precisão. A escolha do modelo a utilizar depende da necessidade de cada projeto, escolhendo a versão YOLOv5s o modelo será mais pequeno, mais rápido na inferência e mais rápido no treino que o modelo YOLOv5x, no entanto a precisão será mais baixa. Escolhendo a versão YOLOv5m haverá um maior equilíbrio entre as configurações.

Model	Average Precision (@50)	Parameters
YOLO-v5s	55.8%	7.5 M
YOLO-v5m	62.4%	21.8 M
YOLO-v5l	65.4%	47.8 M
YOLO-v5x	66.9%	86.7 M

Figura 40 – Avaliação de *performance* entre modelos YOLOv5 [39].

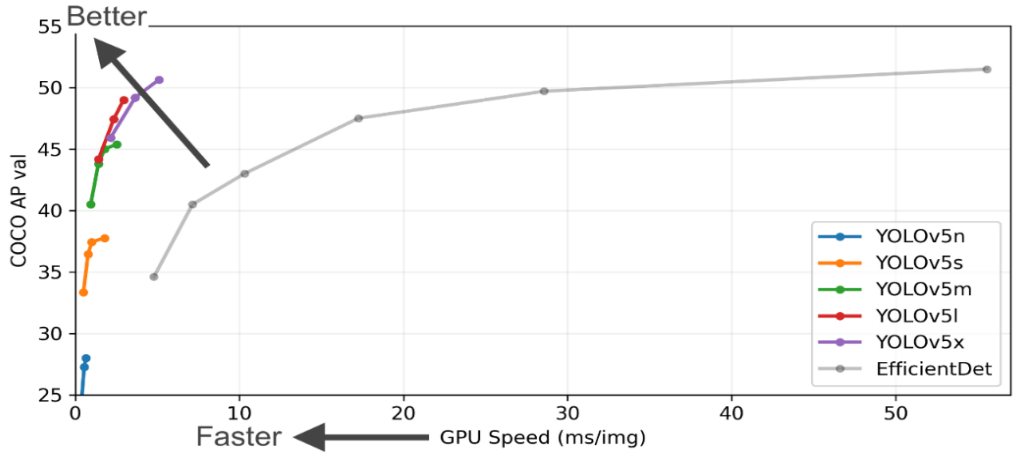


Figura 41 – Métricas YOLOv5 [49].

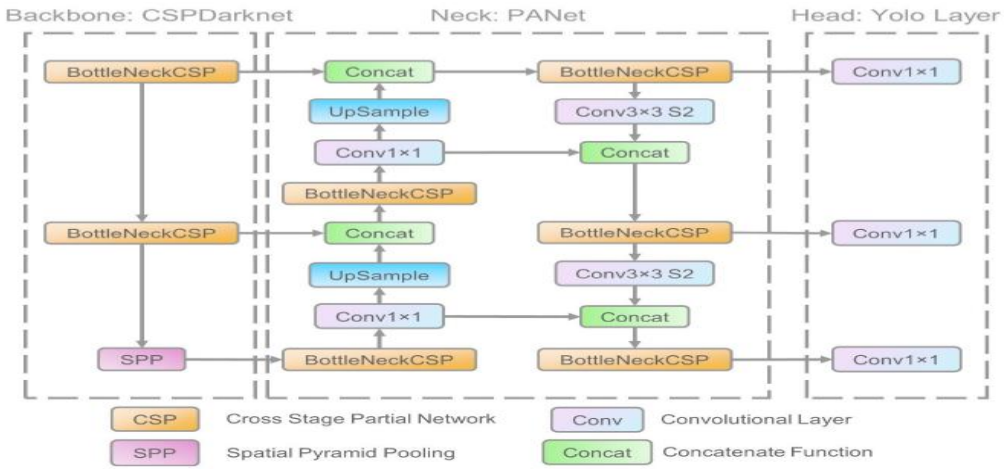


Figura 42 – Arquitetura de rede YOLOv5 [50].

3.2.6 YOLOv6 da Meituan Inc.

O YOLOv6 foi apresentado em 2022 e desenvolvido pela empresa chinesa de comércio eletrônico Meituan Inc. com o objetivo de ser utilizada para aplicações industriais, nomeadamente para condições onde a luminosidade é baixa, oclusões e para objetos de pequenas dimensões [51].

O YOLOv6 apresenta melhorias significativas relativamente à versão YOLOv5, como uma nova arquitetura, deteção sem a tecnologia de caixas de âncora, sem a necessidade de caixas de âncora predefinidas o modelo torna-se mais flexível e robusto às variações do tamanho do objeto. Outra novidade é relativa à rede de pirâmide de recursos, os mapas de recursos de pirâmide são construídos em todos os níveis da rede neural, permitindo o modelo detetar objetos de diferentes tamanhos e resoluções [52].

Este modelo conseguiu apresentar excelentes desempenhos comparativamente aos seus antecessores, nomeadamente na taxa de imagens por segundo. O modelo YOLOv6-N (Figura 43 e Figura 44) conseguiu atingir 35,9% *average precision* no *dataset* COCO a uma taxa de transferência de 1234 fps numa GPU NVIDIA Tesla T4 e o modelo YOLOv6-S 43,3% *average precision* a 869 fps [52]. Os seus antecessores apresentaram resultados mais baixos, o YOLOv1 apresentou taxas de 45 fps e a versão anterior, YOLOv5 apresentou valores à volta de 140fps, valores que já eram bastante bons, mas o YOLOv6 conseguiu elevar a fasquia para valores muito superiores [53].

Variant	mAP 0.5:0.95 (COCO-val)	FPS Tesla T4	Parameters (Million)
YOLO-v6-N	35.9 (300 epochs)	802	4.3
YOLO-v6-T	40.3 (300 epochs)	449	15.0
YOLO-v6-RepOpt	43.3 (300 epochs)	596	17.2
YOLO-v6-S	43.5 (300 epochs)	495	17.2
YOLO-v6-M	49.7	233	34.3
YOLO-v6-L-ReLU	51.7	149	58.5

Figura 43 – Avaliação de performance entre modelos YOLOv6 [39].

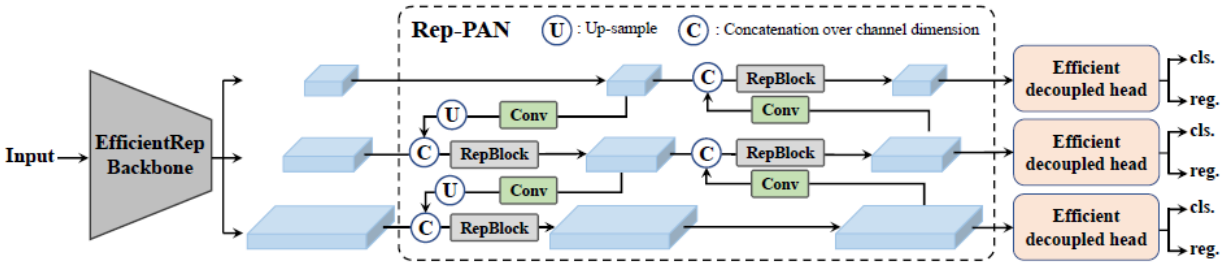


Figura 44 – Arquitetura de rede YOLOv6 [53].

3.2.7 YOLOv7

O YOLOv7 foi desenvolvido pelos mesmos autores da YOLOv4 em 2022 numa Universidade de Taiwan [54].

O YOLOv7 apresentou uma série de melhorias, com uma arquitetura de rede neural mais eficiente, aumentado a velocidade e precisão na deteção de objetos. Implementou novamente o método das caixas de âncora, utiliza nove caixas de âncora permitindo detetar uma maior variedade de formas e tamanhas de objetos. Permite a introdução de imagens com dimensão 1280x1280, com imagens de entrada de maior tamanho o algoritmo consegue ser mais preciso na deteção de objetos.

O YOLOv7 conseguiu atingir ótimas velocidades e precisão, na faixa de 5 FPS a 160 FPS com uma precisão de 56.8% (Figura 45 e Figura 46) [54].

Model	Size (Pixels)	mAP (@50)	Parameters
YOLO-v7-tiny	640	52.8%	6.2 M
YOLO-v7	640	69.7%	36.9 M
YOLO-v7-X	640	71.1%	71.3 M
YOLO-v7-E6	1280	73.5%	97.2 M
YOLO-v7-D6	1280	73.8%	154.7 M

Figura 45 – Avaliação de desempenho entre os modelos YOLOv7 [39].

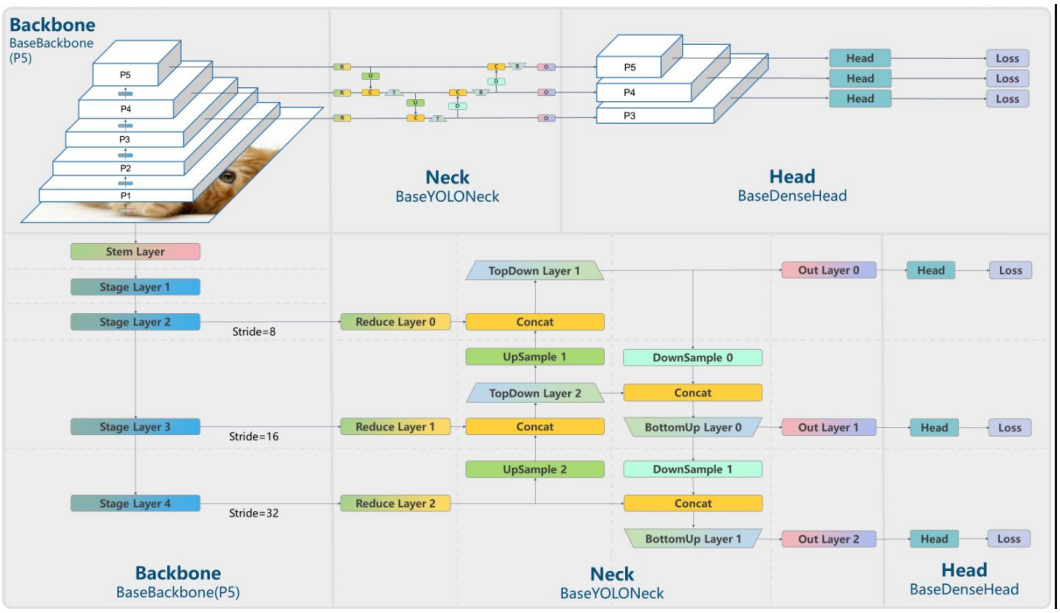


Figura 46 – Arquitetura YOLOv7 [55].

3.2.8 YOLOv8 da Ultralytics Inc.

Tal como o modelo YOLOv5, a YOLOv8 (Figura 47 e Figura 48) foi lançada pela Ultralytics em 2023, sendo o sucessor do YOLOv5 e a última versão da família YOLO. Ao contrário das versões anteriores esta versão pode ser usada para deteção de objetos, classificação de imagens e segmentação de instâncias.

Relativamente ao seu antecessor YOLOv5, a nova versão da Ultralytics sofreu algumas mudanças. As caixas de âncora deixaram de ser utilizadas, reduzindo o número de previsões de caixas delimitadoras e consequentemente acelera a supressão não máxima. Esta opção também torna o modelo menos rígido e mais versátil no ajuste a novos dados.

Outra novidade é relativa à técnica de aumento de dados. Esta nova técnica introduzida chama-se *Mosaic Data Augmentation*. Esta técnica une quatro imagens diferentes e insere esta imagem unida no modelo como entrada, fazendo com que o modelo aprenda a localizar os objetos em diferentes posições [56] [57].

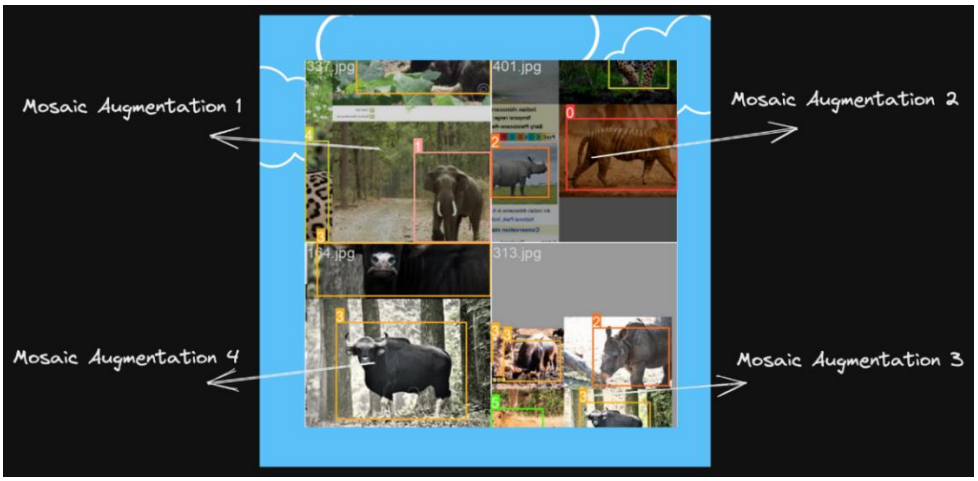


Figura 47 – Técnica *Mosaic Data Augmentation* [57].

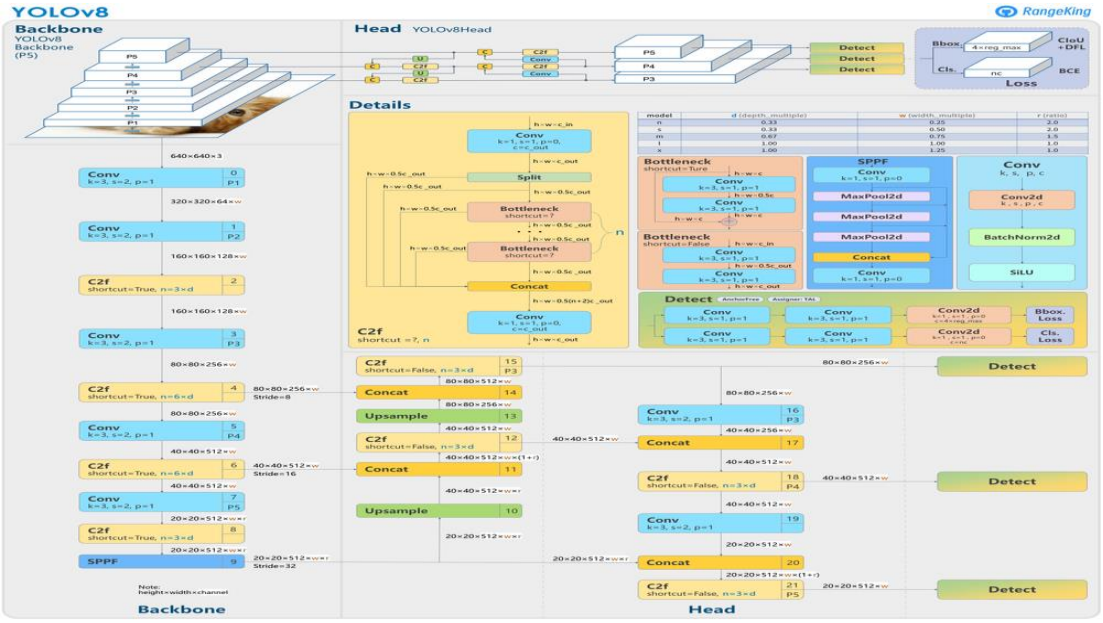


Figura 48 – Arquitetura YOLOv8 [58].

3.3 Medidas de desempenho para deteção de objetos

As medidas de desempenho servem para avaliar a eficiência e precisão dos modelos de deteção de objetos. Estas ferramentas são muito importantes pois ajudam avaliar a capacidade de deteção dos modelos [59].

Eis as mais importantes:

- ***Intersection over Union* (IoU)** – Conforme já dito em capítulos anteriores, IoU é o cálculo da área de sobreposição entre a caixa delimitadora prevista pelo modelo e a caixa delimitadora verdadeira dividida pela área de união dessas duas caixas. Esta medida é fundamental para avaliar a precisão da localização dos objetos [59] [60].
- ***Precision and Recall* (PR)** – *Precision* quantifica a proporção de verdadeiros positivos entre todas as previsões positivas, avaliando a capacidade de evitar falsos positivos, por outras palavras, *Precision* é uma

medida que verifica se o modelo previu corretamente todas as vezes que deveria ter previsto. Dando um exemplo prático com uma imagem que contém 5 carros, o modelo encontra apenas um carro corretamente, a precisão foi perfeita, a única suposição que fez está correta.

O *Recall* calcula a proporção de verdadeiros positivos entre todos os positivos reais, medindo a capacidade do modelo de detetar todas as instâncias de uma determinada classe, entre outras palavras, *Recall* é a capacidade do modelo em encontrar todos os verdadeiros positivos (ver Figura 49) [59] [60].

$$Precision = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falsos positivos}}$$

$$Recall = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falsos negativos}}$$

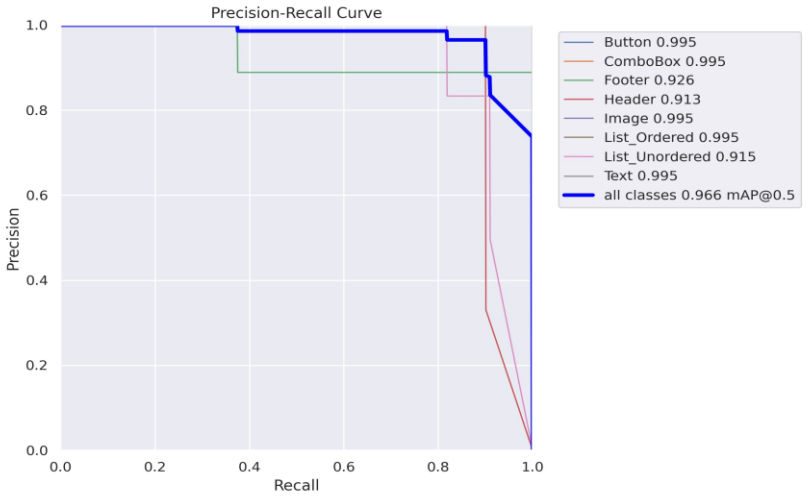


Figura 49 – Gráfico *Precision-Recall Curve*.

- ***Average Precision* (AP)** – AP mede a precisão para a deteção de cada classe do objeto. Para calcular AP é necessário calcular em primeiro a medida *Precision* em vários níveis de *Recall* e em seguida é calculado a média desses valores [59] [60].

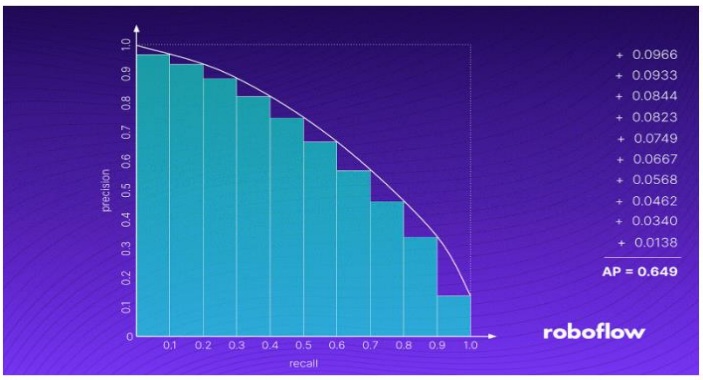


Figura 50 – Exemplo do cálculo da medida *Average Precision*(AP) [60].

- ***Mean Average Precision* (mAP)** – Com todos os valores de AP para cada classe calculados é possível calcular a média de todos esses valores e assim obtém-se o mAP. Em mAP existem duas medidas de forma a ter medidas mais abrangentes. Com mAP50 o limite de IoU é de 0,50, é uma medida utilizada para detetar objetos mais fáceis de deteção, enquanto a medida mAP95 calcula em vários limites entre 0,50 e 0,95, fornecendo uma visão mais abrangente do desempenho em diferentes níveis de dificuldade de deteção (ver Figura 50) [59].

- **F1 Score** – Combina a medida *Precison* e *Recall* numa única medida. Fornece uma avaliação equilibrada do desempenho do modelo enquanto considera falsos positivos e falsos negativos (ver Figura 51).

$$F1\ Score = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

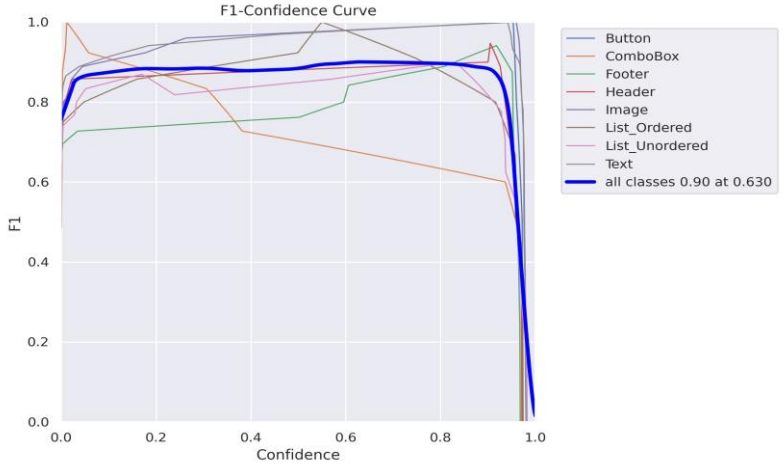


Figura 51 – Gráfico F1 Score.

3.4 *Dataset* públicos para avaliação de desempenho

Para avaliar o desempenho dos vários modelos de deteção de objetos existem plataformas com *datasets* definidos, fundamentais para avaliar e rastrear o progresso dos vários modelos existentes na área da visão computacional [61].

Alguns *datasets* públicos mais conhecidos são:

- ImageNet – é um *dataset* de reconhecimento visual em grande escala com mais de 14 milhões de imagens rotuladas e mais de 20 mil categorias de objetos, sendo maioritariamente usado para avaliar algoritmos de classificação de imagens [62];

- MS COCO – *Microsoft Common Objects in Context* é um *dataset* de deteção de objetos em grande escala com mais de 200 mil imagens rotuladas com caixas delimitadoras e máscaras de segmentação de instância para 90

categorias de objetos. É usado para avaliar algoritmos de deteção de objetos e segmentação de instâncias [63];

- PascalVOC – *Pascal Visual Object Classes* é um *dataset* de deteção de objetos com mais de 16 mil imagens rotuladas com caixas delimitadoras para 20 categorias de objetos usado para a classificação de imagens e deteção de objetos [64];
- Cityscapes – *dataset* em grande escala para segmentação semântica de imagens com mais de 25 mil imagens rotuladas com anotações em nível pixel para 34 categorias de objetos. Estes conjuntos contêm apenas cenas de estradas urbanas. Usado maioritariamente para aprendizagem de veículos autónomos [65];
- CIFAR-10/100 – *dataset* com 60 mil imagens rotuladas. Este *dataset* é dividido em dois *datasets*, um *dataset* com 10 classes de objetos e outro *dataset* com 100 classes de objetos [66].

3.5 Desempenho de alguns projetos com modelos YOLO na deteção de objetos

3.5.1 Deteção de incêndio florestal com YOLOv5, EfficientNet e EfficientDet

Com os fogos florestais a serem cada vez mais frequentes devido às alterações climáticas foi testado um sistema de deteção de fogos florestais.

Detetar fogos florestais pode ser uma tarefa árdua, existem diferentes tipos de floresta, com vários cenários de vegetação e vários tipos de incêndio como fogos no solo, fogo em árvores. Todos estes fatores do meio ambiente com grande diversidade na forma, textura e cor trazem uma grande dificuldade para a deteção de fogos florestais. Para elevar o grau de precisão, foi proposto que três modelos trabalhassem em conjunto na deteção de fogos. Para a deteção de fogos foram escolhidos os detetores YOLOv5 e EfficientDet, estes dois detetores só irão detetar fogos. Estes modelos não vão considerar toda a imagem, para a classificação de toda a imagem, reunindo toda a informação da imagem será utilizado o modelo EfficientNet. Os resultados de deteção final serão decididos seguindo uma estratégia baseada na decisão entre os três modelos.

Para a realização deste teste foram utilizadas várias imagens de *datasets* públicos: BowFire, FD-*dataset*, ForestryImages, VisiFire, etc, representando um total de 10581 imagens, onde 2976 imagens contêm fogos e 7605 imagens que não contêm fogos.

Foram aplicadas diferentes estratégias para os 3 modelos: YOLOv5, EfficientDet e EfficientNet. YOLOv5 e EfficientDet foram treinados com 2381 imagens com fogos e testadas em 476 com imagens com fogos. EfficientNet foi treinada com 2381 imagens com fogos e 5804 imagens sem fogos e testada com 476 imagens com fogos

e 1160 imagens sem fogos. O treino foi feito com um GPU NVIDIA GTX 2080TI durante 300 épocas, tendo conseguido atingir os valores de map 79.7% (utilizando os modelos YOLOv5 e EfficientDet) e 79% (utilizando os 3 modelos, respetivamente YOLOv5, EfficientDet e EfficientNet). De forma isolada o YOLOv5 atingiu um mAP50 de 70,5% e EfficientDet um mAP50 de 75,7 [67].

3.5.2 Deteção de objetos voadores com YOLOv8

Com o aumento exponencial de diversos tipos de drones, como drones caseiros e drones militares e como são equipamentos que são difíceis de detetar, um grupo de investigadores do Instituto Tecnológico da Georgia decidiu testar um sistema de deteção de objetos utilizando o YOLOv8 capaz de detetar veículos aéreos não tripulados.

Para este estudo foi utilizado um *dataset* com 11998 imagens com 4 classes rotuladas: drones, helicópteros, pássaros e aviões.

As imagens foram divididas em 90% imagens de treino e 10% imagens de validação e treinadas durante 190 épocas, alcançado um mAP50-95 de 83.5% [68].

3.5.3 Deteção de gestos em tempo real utilizando o YOLOv4

O desenvolvimento da tecnologia trouxe-nos vários equipamentos modernos que são essenciais para o nosso dia a dia. Um dos vários equipamentos disponíveis são os comandos eletrônicos capazes de controlar vários equipamentos, como um comando de controlo convencional capaz de controlar o aparelho de ar condicionado ou uma televisão, comando de controlo de reconhecimento de fala e controlo de reconhecimento visual.

Neste estudo os investigadores propõem um sistema capaz de detetar gestos com as mãos num sistema de reprodução de música incorporado no carro. Com os vários gestos da mão pretende-se pausar a música, iniciar a música, parar a música, diminuir e aumentar volume e trocar de música, perfazendo um total de 7 gestos. Foi utilizado um *dataset* com 245 imagens e foram processados com um tamanho de resolução de 416x416. O *hardware* usado foi uma GPU NVIDIA GTX1070.

O experimento através do algoritmo YOLOv4 conseguiu atingir um resultado mAP50 de 99.7% e o mAP95 atingiu 98.35%.

Para além da implementação dos gestos de mãos capazes de controlar a reprodução de músicas, este estudo também pretende verificar qual o método mais seguro entre os botões incorporados no automóvel para controlar as faixas de música ou os gestos de mãos durante a condução do automóvel. O experimento foi feito com 8 voluntários, 4 homens e 4 mulheres com idades entre os 20 e os 23 anos. Foi utilizado um ambiente de condução virtual onde os voluntários simulavam uma condução normal na vida real e enquanto conduziam era solicitado aos voluntários que

controlassem a música consoante as instruções indicadas. O experimento foi dividido entre dois modelos, o primeiro modelo os voluntários controlavam a reprodução das músicas com os gestos das mãos e no segundo modelo os voluntários controlavam a reprodução da música utilizando os botões presentes no carro. A comparação entre os dois modelos é quantificada com o número de colisões durante a condução. O experimento resultou que durante a condução onde era possível controlar a reprodução por gestos obteve um menor número de colisões em comparação com o segundo modelo [69].

3.5.4 Deteção de máscaras anti COVID-19 com o YOLOv5

Uma das formas mais eficazes de restringir a propagação das viroses é o uso de máscara facial e para detetar quem usa máscaras foi testado um modelo capaz de detetar as pessoas que usam máscara e as pessoas que não usam máscara com o modelo YOLOv5s e o modelo YOLOv5x através do *Google Colaboratory*. Este estudo foi avaliado com centenas de imagens de pessoas usando máscaras e pessoas que não usam máscaras. O treino em ambos os modelos foi de 100 épocas e obtiveram resultados mAP bastantes similares. Os estudos mostram também que o modelo YOLOv5s comparativamente ao modelo YOLOv5x apresenta em termos de desempenho e velocidade melhores resultados e tendo em conta que os valores de mAP são similares, para este tipo de problemas o uso do modelo YOLOv5s é o mais recomendado [70].

3.5.5 Deteção de objetos marítimos com YOLOv7

Para ajudar nas missões de busca e salvamentos de pessoas em mar aberto, foi estudado um sistema capaz de detetar pessoas, barcos e outros objetos dentro de água.

O modelo utilizado foi o YOLOv7 e o experimento foi feito com um *dataset* de 14227 imagens, onde 8930 imagens foram utilizadas para treinar o sistema, 1547 para validação e 3750 para teste. As imagens contêm 5 categorias: *swimer, boat, jetski, life savings appliances* e *buoy*. O treino foi feito durante 60 épocas utilizando a GPU 8 NVIDIA Tesla V100.

O experimento foi feito em dois modelos, o modelo base do YOLOv7 e outro modelo baseado no YOLOv7, mas com alguns complementos adicionais feitos pela equipa de investigação designado por YOLOv7-sea. Seguindo as métricas padrão o YOLOv7 obteve um mAP50 de 83.12%, mAP75 de 56.72 e um AP de 53.61, enquanto o modelo YOLOv7-sea obteve um mAP50 de 90.72%, mAP75 de 64.15% e um AP de 59%.

O modelo YOLOv7-sea obteve um resultado 7% mais alto que o modelo YOLOv7, demonstrando que para estes tipos de cenários o modelo YOLOv7-sea consegue ter melhores resultados nas missões de busca e salvamentos [71].

3.5.6 Deteção de defeitos na superfície de um painel solar

Outro experimento tem como finalidade a deteção de defeitos numa superfície de um painel solar utilizando um modelo de deteção, pois os autores da experiência afirmam que a inspeção liderada por humanos traz muitas desvantagens, como a inspeção célula por célula que é demorada, cansativa e de eficiência reduzida. O sistema de deteção é baseado no modelo YOLOv5 com algumas modificações. O sistema proposto tem como base o YOLOv5 e dentro do bloco CSP foi integrado uma convolução deformável, este tipo de operação introduz deslocamentos deformáveis permitindo que a rede ajuste os locais de amostra com base nos dados de entrada, o que acaba por ser benéfico para capturar padrões espacialmente variantes. Com este melhoramento da arquitetura conseguiram alcançar um mAP de 89.64%, enquanto a arquitetura original alcançou um mAP de 81.79% [39].

Outro experimento foi feito para detetar defeitos encontrados durante o processo de fabricação de células cristalinas como manchas/regiões escuras e microfissuras. Este tipo de defeitos tem um impacto bastante negativo na eficiência dos painéis. Para detetar estes defeitos foram utilizadas duas versões do modelo YOLOv4. O modelo YOLOv4 aprimorado alcançou um mAP de 98.8% a uma velocidade de 62,9 ms, enquanto o modelo YOLOv4-s alcançou um mAP de 91% a uma velocidade de 28,2 ms [39].

3.6 Comparação de desempenhos entre modelos YOLO

A escolha entre os vários modelos de deteção de objetos, nomeadamente dos modelos YOLO para o desenvolvimento de determinados projetos não é uma escolha fácil. Há alguns fatores a ter em conta na hora de escolher a melhor versão para um determinado projeto, como qual o melhor modelo com o melhor valor FPS para detetar objetos em tempo real? Qual a inferência de velocidade entre o CPU e GPU? Qual o modelo GPU a escolher? Qual o modelo mais preciso? Qual o modelo mais rápido?

Tendo em conta estas questões abordam-se algumas comparações feitas entre alguns modelos YOLO.

3.6.1 Avaliação de desempenho entre YOLOv5, YOLOv7 e YOLOv8 utilizando a GPU NVIDIA Jetson AGX Orin

Poucos avaliadores comparam o YOLOv7 com a YOLOv5 e YOLOv8 da Ultralytics. A Stereolabs, uma empresa especializada em desenvolvimento de

tecnologias de visão computacional, decidiu em 2023 comparar os três modelos com latências reais usando TensorRT 8.4 e JetPack5 obtendo os resultados indicados na Figura 52 e Figura 53.

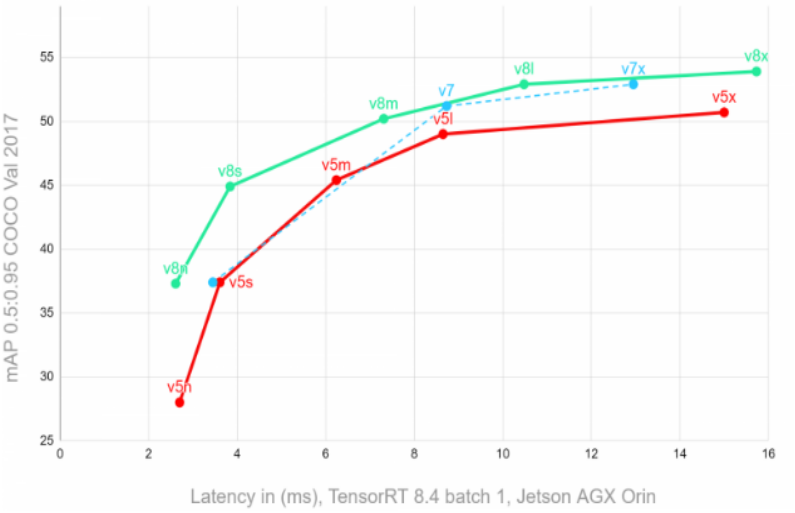


Figura 52 – Avaliação de desempenho entre YOLOv5, YOLOv7 e YOLOv8 [72].

MODELO	PA	AP0,5	AGX ORIN (FPS)	RTX 4070TI (FPS)
v5n	28	45,7	370	934
v8n	37,3	52,5	383	1163
v7-minúsculo	37,4	55,2	290	917
v5s	37,4	56,8	277	877
v8s	44,9	61,8	260	925
v5m	45,4	64,1	160	586
v8m	50,2	67,2	137	540
v5l	49	67,3	116	446
v7	51,2	69,7	115	452
v8l	52,9	69,8	95	391
v5x	50,7	68,9	67	252
v7x	52,9	71,1	77	294
v8x	53,9	71,0	64	236

Figura 53 – Detalhes da Avaliação de desempenho [72].

Retirando algumas notas relativas às avaliações demonstradas, verifica-se que em termos de precisão e velocidade o modelo YOLOv8n apresenta o melhor resultado. Para imagens com resoluções de 1280x1280 o YOLOv7 continua a ser a melhor escolha para inferência em alta resolução, pois o YOLOv8 ainda não treina modelos com essa resolução. Na generalidade dos casos o modelo YOLOv8 demonstra melhores resultados que os seus antecessores [72].

3.6.2 Avaliação de desempenho de FPS entre YOLOv5, YOLOv6 e YOLOv7 utilizando uma CPU

Este experimento mede a capacidade da velocidade de inferência dos modelos YOLOv5, YOLOv6 e YOLOv7 num CPU modelo i7 6850K com 32 GB de ram (ver Figura 54).

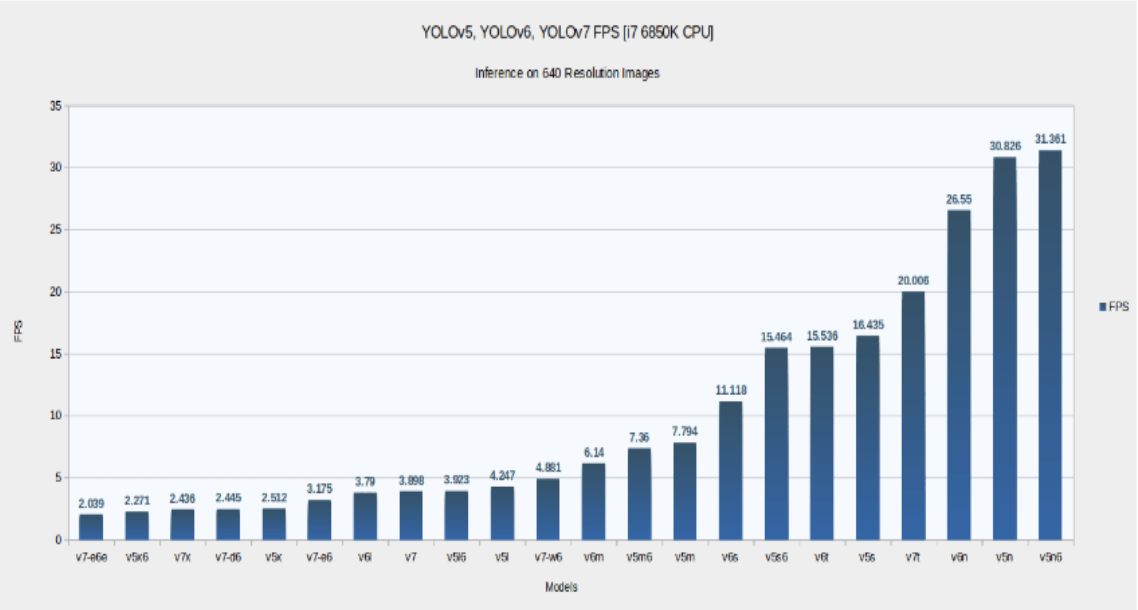


Figura 54 – Avaliação de desempenho FPS entre os modelos YOLOv5, YOLOv6 e YOLOv7 com CPU [73].

Analisando o gráfico é fácil de verificar que o modelo YOLOv5n conseguiu obter os melhores resultados, alcançando uma velocidade de 31.261 FPS [73].

3.6.3 Avaliação de desempenho de FPS entre YOLOv5, YOLOv6 e YOLOv7 utilizando uma GPU modelo NVIDIA RTX 4090

Outro experimento feito para avaliação da velocidade de inferência, mas desta vez foi utilizado uma GPU modelo NVIDIA RTX 4090 (ver Figura 55).

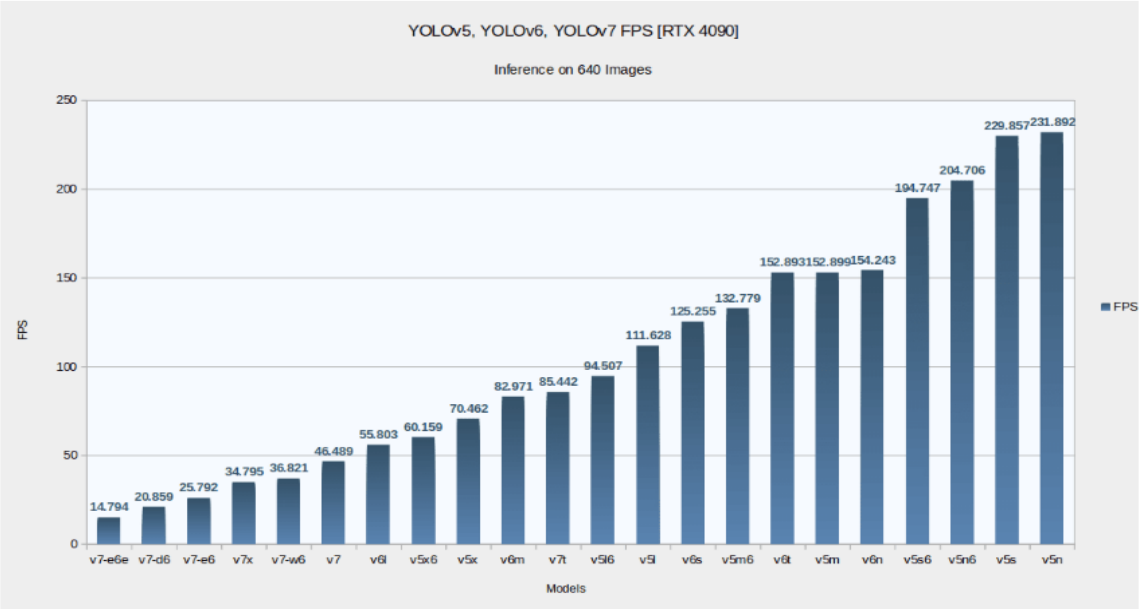


Figura 55 - Avaliação de desempenho FPS entre os modelos YOLOv5, YOLOv6 e YOLOv7 com GPU [73].

Utilizando uma GPU a velocidade de inferência é bastante mais alta. O modelo YOLOv5n passou de uma velocidade de 30 FPS para 231 FPS, quase 8x superior ao teste anterior [73].

3.6.4 Comparação dos modelos YOLO usando GPU NVIDIA Tesla P100, V100, GTX 1080 Ti e RTX 4090

Neste experimento é feita a comparação entre os modelos YOLO utilizando 4 modelos de GPU: NVIDIA Tesla P100, V100, GTX 1080 Ti e RTX 4090, com o objetivo de encontrar o modelo mais rápido (ver Figura 56).

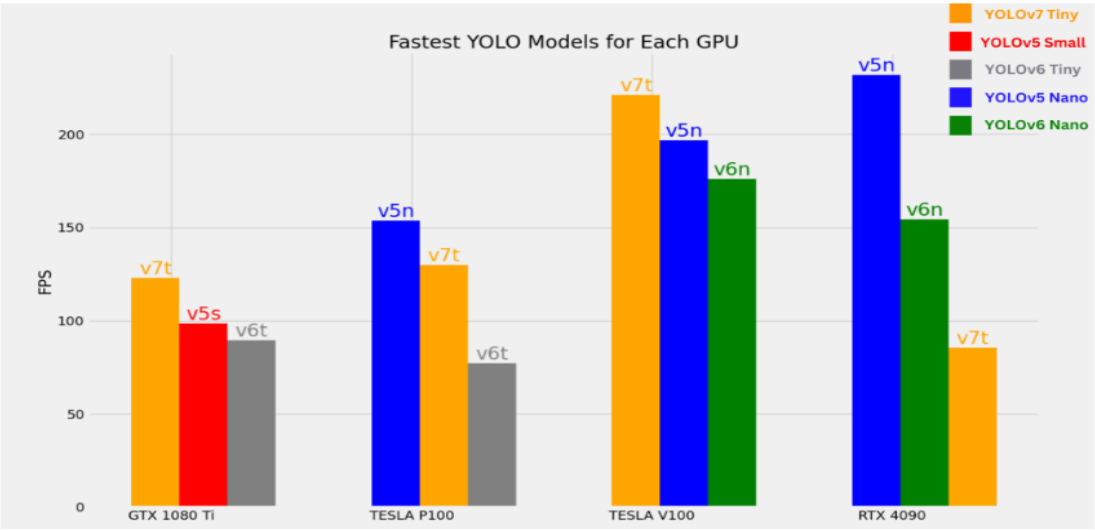


Figura 56 – Modelo YOLO mais rápido utilizando os GPUs: RTX, GTX e Tesla [73].

Analisando o gráfico, o modelo YOLOv5n é o mais rápido quando é utilizado na GPU RTX 4090, o segundo modelo mais rápido é o modelo YOLOv7t quando utilizado na GPU Tesla V100. O modelo YOLOv6t alcançou sempre resultados menores comparativamente aos modelos YOLOv5 e YOLOv7 em todos os GPUs [73].

3.6.5 Comparação de desempenho YOLO mAP e FPS

Nesta avaliação foi feita a comparação entre os diferentes modelos de CPU e GPU de acordo com o mAP (*mean Average Precision*) e FPS.

- 1ª comparação usando um CPU modelo i7 6850K:

  Em modelos CPUs não são utilizados os modelos grandes da família YOLO, por isso, neste experimento foi utilizado o modelo YOLOv5n, YOLOv6n e YOLOv7 Tiny. As imagens são dimensionadas para uma resolução de 640x640 (ver Figura 57).

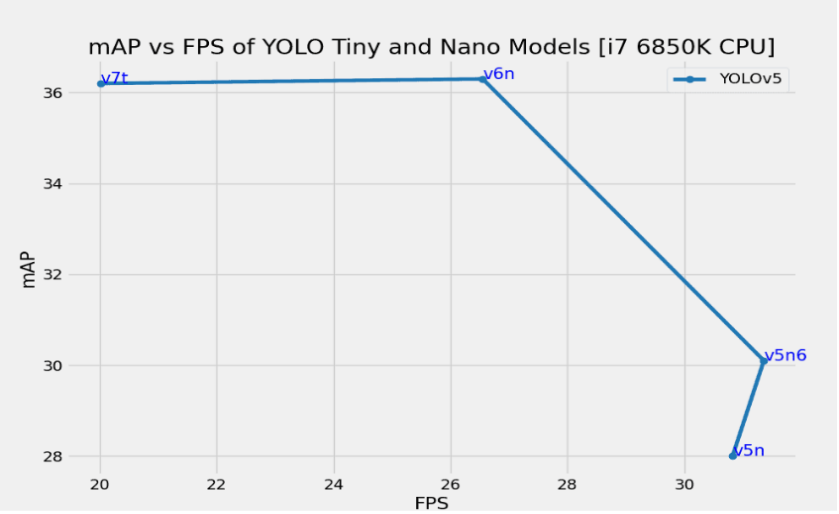


Figura 57 – Comparação de mAP e FPS entre modelos YOLO [73].

Ao analisar o gráfico verifica-se que os modelos YOLOv7-tiny e o modelo YOLOv6n conseguem atingir melhores resultados na métrica de mAP, enquanto o modelo YOLOv5n alcança um mAP mais baixo mas consegue obter uma velocidade de inferência superior [73].

- 2ª comparação entre modelos GPUs:

Neste experimento foram utilizados os modelos YOLOv5, YOLOv6 e YOLOv7, com imagens dimensionadas para uma resolução de 640x640. Os modelos de GPU comparados são: NVIDIA RTX 4090, Tesla P100, Tesla V100 e NVIDIA GTX 1080 Ti (ver Figura 58, Figura 59, Figura 60 e Figura 61).

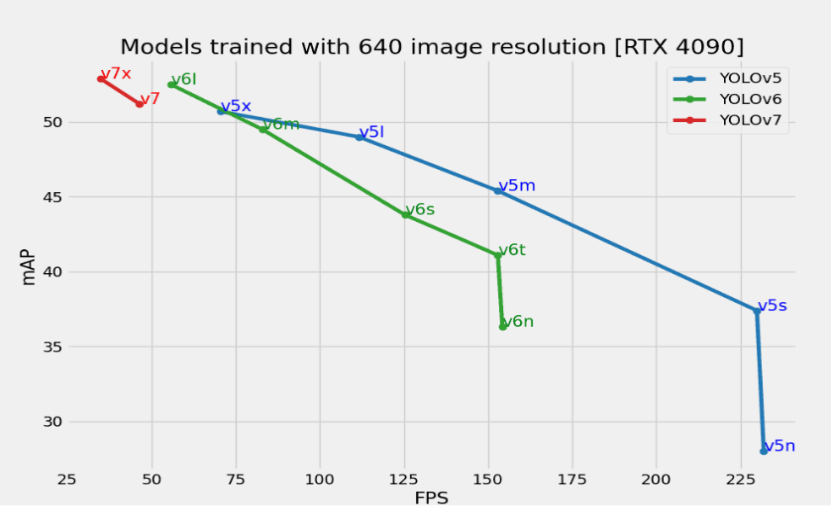


Figura 58 - Comparação de mAP e FPS entre modelos YOLO usando a GPU RTX 4090 [73].

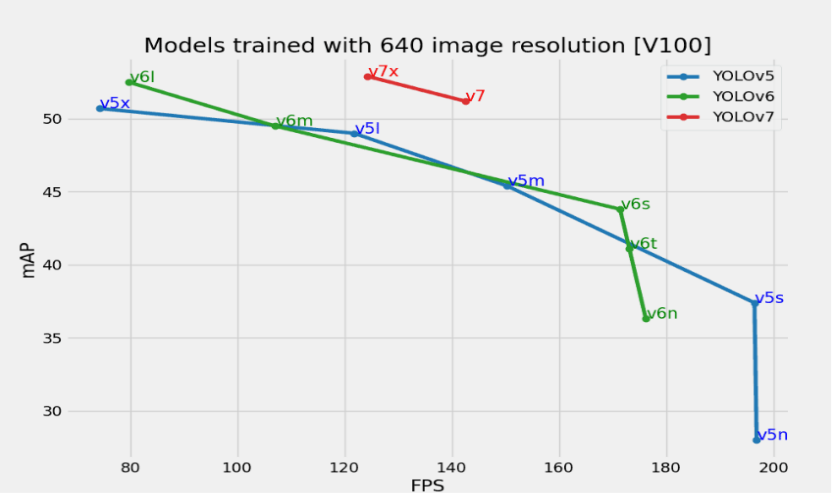


Figura 59 - Comparação de mAP e FPS entre modelos YOLO usando a GPU TESLA V100 [73].

Figura 60 - Comparação de mAP e FPS entre modelos YOLO usando a GPU TESLA P100 [73].

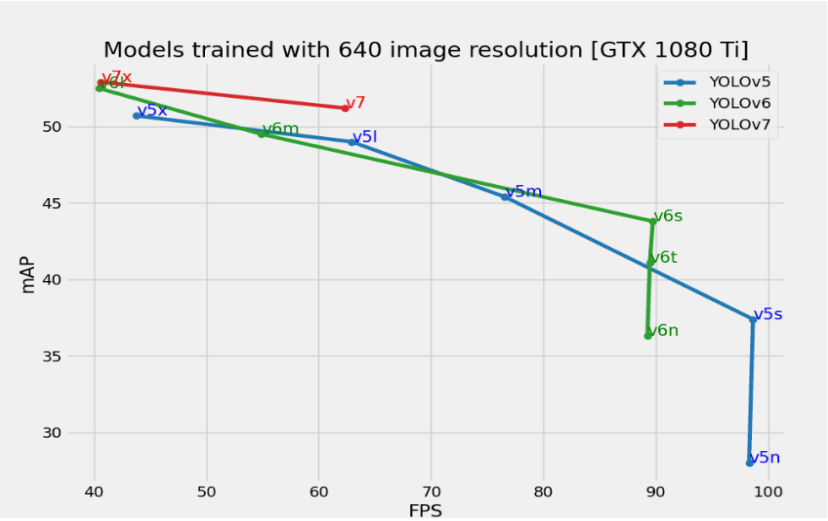


Figura 61 - Comparação de mAP e FPS entre modelos YOLO usando a GPU NVIDIA GTX 1080 Ti [73].

Analisando e comparando os gráficos entre os processadores GPUs, fica claro mais uma vez que a versão YOLOv5n consegue atingir velocidades de inferência superiores comparativamente aos outros modelos e com os modelos GPUs atinge uma velocidade muito maior do que com o modelo de CPU.

Em todos os processadores GPU o modelo YOLOv7 consegue métricas de mAP muito maiores que os restantes modelos [73].

3.6.6 Comparação entre modelos YOLOv5, YOLOv6, YOLOv7 e YOLOv8

Analisando as métricas das últimas 4 versões oficiais da YOLO, verifica-se que em todos os tamanhos a versão YOLOv8 consegue obter os melhores resultados tanto na precisão como na velocidade de inferência conforme representado na Figura 62 [74].

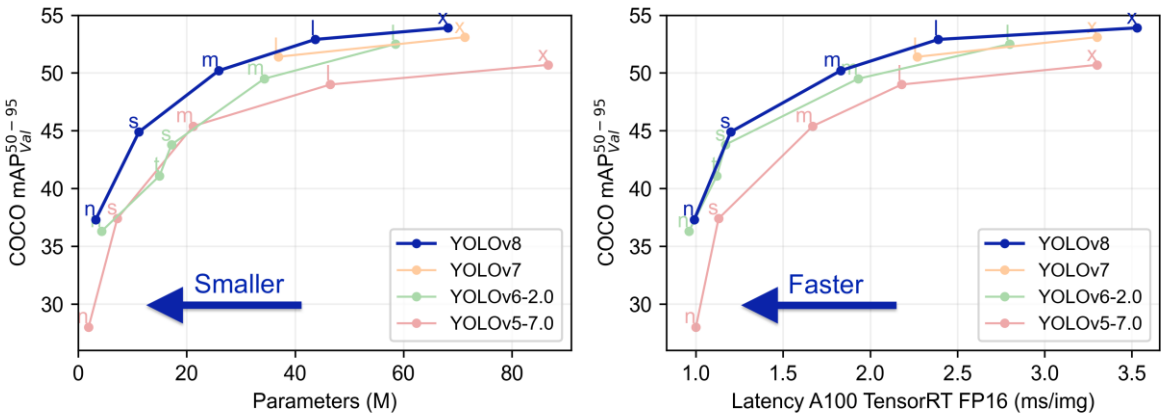


Figura 62 – Comparação entre os modelos YOLOv5, YOLOv6, YOLOv7 e YOLOv8 [75].

3.7 Conclusão

Este capítulo oferece uma análise dos vários modelos YOLO, destacando a trajetória evolutiva e salientando as melhorias contínuas dos diferentes modelos em velocidade e precisão na deteção de objetos. É mostrado que o modelo é bastante versátil pois é capaz de criar modelos de treino capazes de detetar incêndios, gestos, objetos voadores e barcos com velocidade e eficácia em diferentes contextos.

4 RECURSOS UTILIZADOS NO CASO DE ESTUDO

Neste capítulo serão abordados os recursos utilizados na criação do *dataset* e processo de treino.

4.1 Roboflow

O Roboflow [76] é uma plataforma *online* que permite processar e gerir dados de imagens para tarefas de visão computacional. Com a plataforma Roboflow é possível rotular as imagens para a deteção de objetos, segmentar, entre outras tarefas, essenciais para treino de deteção de imagens. Outra característica muito importante possível na plataforma Roboflow é a técnica de aumento de dados, esta técnica serve para aumentar a diversidade dos dados de treino, o que faz com que os modelos se tornem mais robustos e eficazes.

Neste projeto, a plataforma Roboflow foi a escolhida para armazenar e processar os vários conjuntos de imagens criados, devido à sua potencialidade como a rotulação de imagens e aumento de dados.

Definido o modelo YOLO a treinar, este extrai diretamente o *dataset* armazenado na plataforma Roboflow através da seleção do tipo de exportação pretendido e convertido num link (ver Figura 63). A escolha do tipo de exportação era definida consoante o modelo a treinar, caso fosse o modelo YOLOv5 o formato escolhido seria "YOLOv5-PyTorch", YOLOv8 o formato seria "YOLOv8" e caso fosse o Detectron2 para a deteção de objetos seria o formato "coco".

Na Figura 64 temos um exemplo de um dos conjuntos de dados criado. Neste *dataset* é possível verificar que existem 312 imagens, estas imagens estão divididas em treino, validação e teste, respetivamente em 88:8:4, também é possível verificar que as imagens foram redimensionadas para a resolução 640x640 e que foram aplicadas algumas técnicas de aumento de dados.

Na Figura 65 é apresentado o ambiente de trabalho onde é possível rotular as imagens do *dataset* e na Figura 66 são representadas as classes do *dataset*.

Existem outras plataformas semelhantes ao Roboflow como a plataforma LabelBox, COCO Annotator ou VGG Image Annotator, mas não foram utilizadas neste projeto.

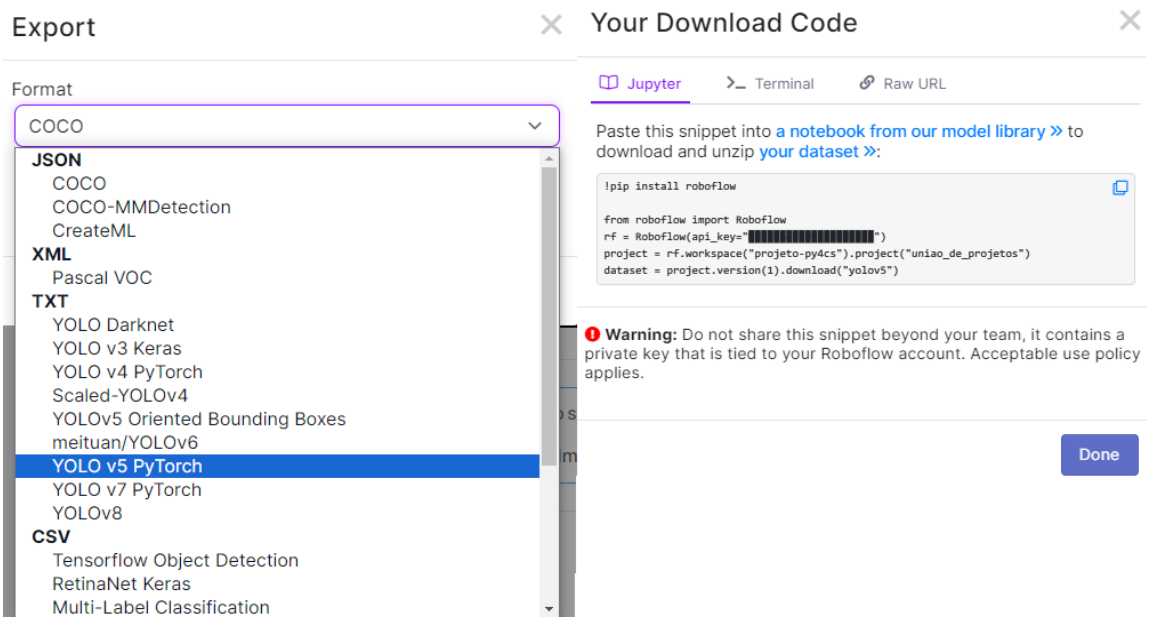


Figura 63 – *Link* do Roboflow para *download* do *dataset*.

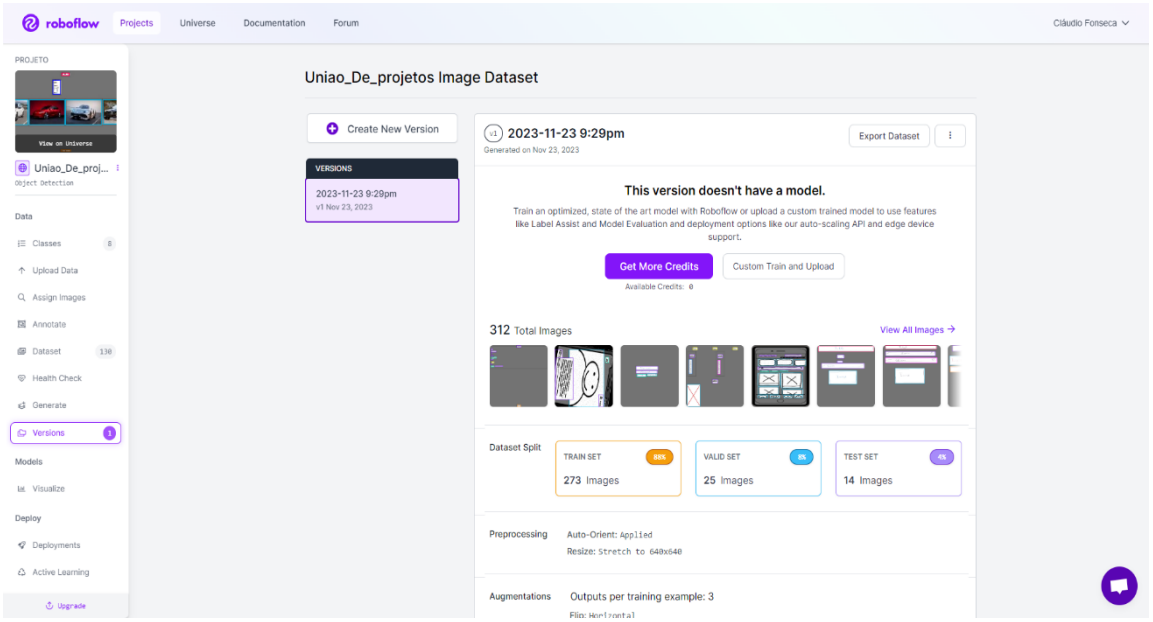


Figura 64 – Ambiente de trabalho da plataforma Roboflow.

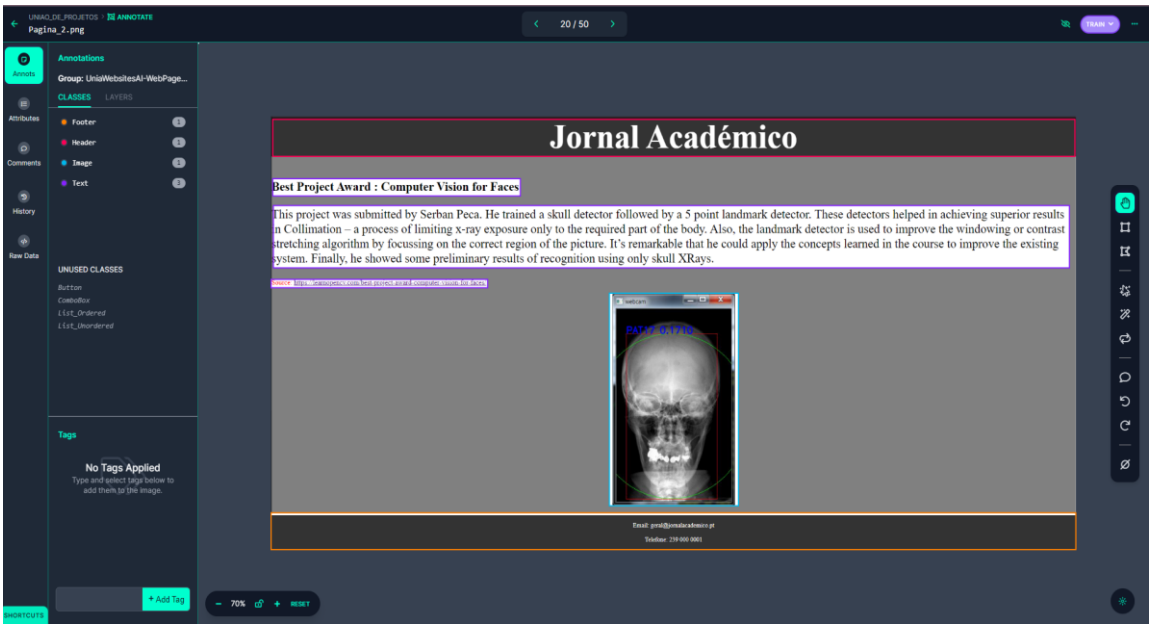


Figura 65 – Ambiente de trabalho Roboflow para rotular imagens.

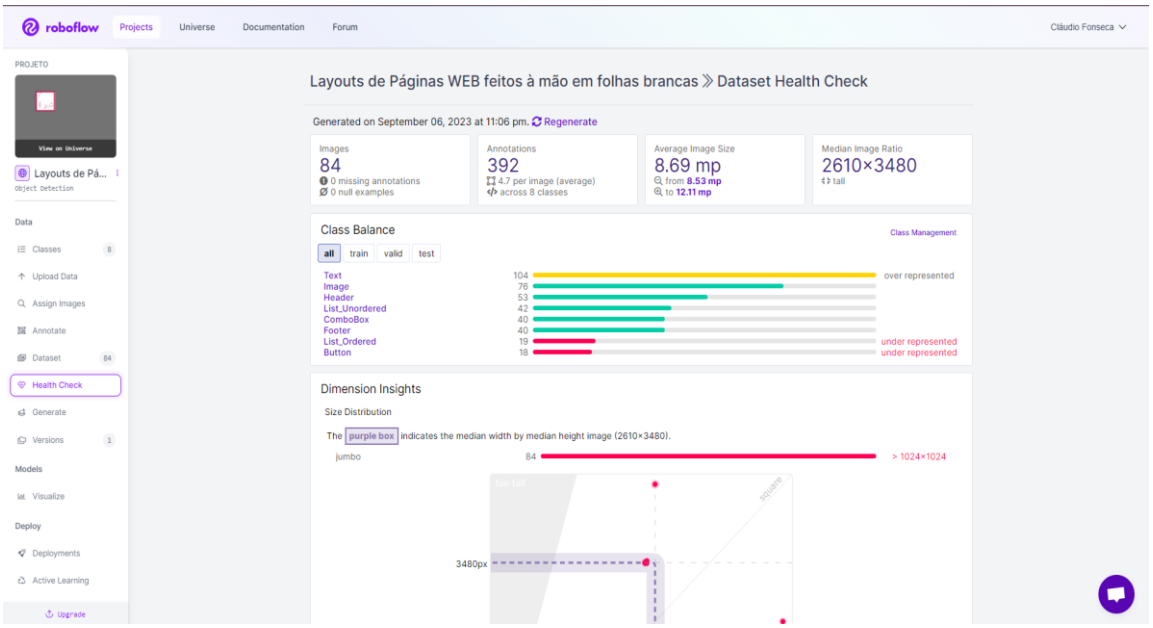


Figura 66 – Ambiente de trabalho Roboflow com as classes do *dataset*.

4.2 GoogleColab

O GoogleColab é uma plataforma online via cloud que permite escrever e executar código em *Python* em ambiente Jupyter Notebook. Esta plataforma foi utilizada devido as suas características fundamentais para a execução do projeto, como: [77]

- Acesso gratuito às unidades de processamento gráfico (GPU) para acelerar o treino dos modelos de *machine learning*, sem acesso gratuito às GPU o autor

teria que utilizar o processamento do próprio computador o que levaria uma eternidade no sentido figurativo;

- Armazenamento de dados no Google drive e integração no google drive, permitindo importar *dataset* e salvar modelos diretamente no Google drive;
- Bibliotecas de *machine learning* como TensorFlow e PyTorch, processamento de dados e visualização já estão pré-instaladas, facilitando a configuração.

4.3 PyTorch

O PyTorch é uma biblioteca de código aberto para *machine learning* desenvolvida pela Facebook's AI Research lab (FAIR). É uma *framework* com grande flexibilidade e facilidade de uso, sendo por isso muito utilizada para aplicações na área de visão computacional e processamento de linguagem natural. Foi criada para fornecer modelos mais fáceis e práticos do que outras bibliotecas como o TensorFlow.

Os modelos utilizados neste projeto como YOLOv5, YOLOv8 e Detectron2 utilizam a biblioteca Pytorch [78].

4.4 YOLOv5 e YOLOv8

Conforme já mencionado no capítulo 3, o YOLOv5 e YOLOv8 são algoritmos de deteção de objetos capazes de detetar objetos em imagens e em tempo real.

4.5 Detectron2

O detectron2 também é um algoritmo capaz de detetar objetos em imagens e em tempo real, tendo também a particularidade de ser capaz de fazer segmentação de imagens conforme mencionado no capítulo 2.1.2.6

4.6 Conclusão

Este capítulo de forma resumida apresenta as ferramentas utilizadas que foram essenciais na execução deste trabalho, desde a plataforma Roboflow e GoogleColab, à biblioteca PyTorch e aos modelos YOLO e Detectron2 capazes de detetar objetos em imagens em tempo real.

5 TESTES E RESULTADOS

Neste capítulo vão ser apresentados os testes e os resultados obtidos no processo de treino para este projeto. Foram utilizados 3 modelos de deteção de objetos, e serão apresentados os resultados de cada modelo e feita a comparação de resultados entre os vários modelos.

5.1 *Dataset*

Para esta experiência foram criados 5 *datasets* (ver Tabela 1). O 1º *dataset* é composto por imagens com *layouts* de páginas WEB feitos à mão numa folha de papel branca (Figura 67), tiradas por um telemóvel modelo Xiaomi Redmi Note 9 Pro, em diferentes ângulos, distâncias e ambiente. O 1º *dataset* conta com 44 imagens e foi usado para começar os primeiros treinos da rede. No entanto, verificou-se que o conjunto de imagens era pequeno e os resultados fracos.

No 2º *dataset* foram adicionadas mais imagens, totalizando um total de 202 imagens. As 44 imagens do 1º *dataset* mantiveram-se e foram adicionadas as mesmas imagens mas desta vez foram digitalizadas através de um scanner, totalizando assim um total de 84 imagens, as restantes imagens adicionadas foram através da técnica de aumento de dados presente na plataforma Roboflow, aproveitando as imagens originais mas adicionando ruído (ver Figura 68), embaciamento entre outras características.

No 3º *dataset* criou-se um *dataset* de páginas WEB desenvolvidas em HTML e CSS, composto por um conjunto de 48 imagens. As páginas WEB foram feitas pelo autor com conteúdo arbitrário (ver Figura 69).

No 4º *dataset* foi criado um conjunto de imagens de *Wireframes*. Os *wireframes* são representações visuais esquemáticas e simplificadas de uma *interface* de utilizador (UI), são utilizadas na fase inicial do processo de design para esboçar uma estrutura e o *layout* de uma página WEB, aplicativo móvel ou outro produto. Este modelo é composto por 62 imagens (ver Figura 70) e as a imagens foram retiradas da internet de forma aleatória.

O 5º *dataset* é a união de todos os conjuntos de dados anteriores (à exceção do 1º *dataset*). Procedeu-se a esta união de dados para tentar criar um modelo de treino capaz de detetar os *layouts* em qualquer uma das situações.

Após submissão de todas as imagens nos conjuntos de dados é necessário rotular os objetos desenhando caixas delimitadoras em torno do objeto (ver Figura 65). Na maioria dos conjuntos de dados criados os objetos foram rotulados entre 8 classes disponíveis, são elas: *Text, Image, Button, ComboBox, Header, Footer, List_Unordered, List_Ordered.*

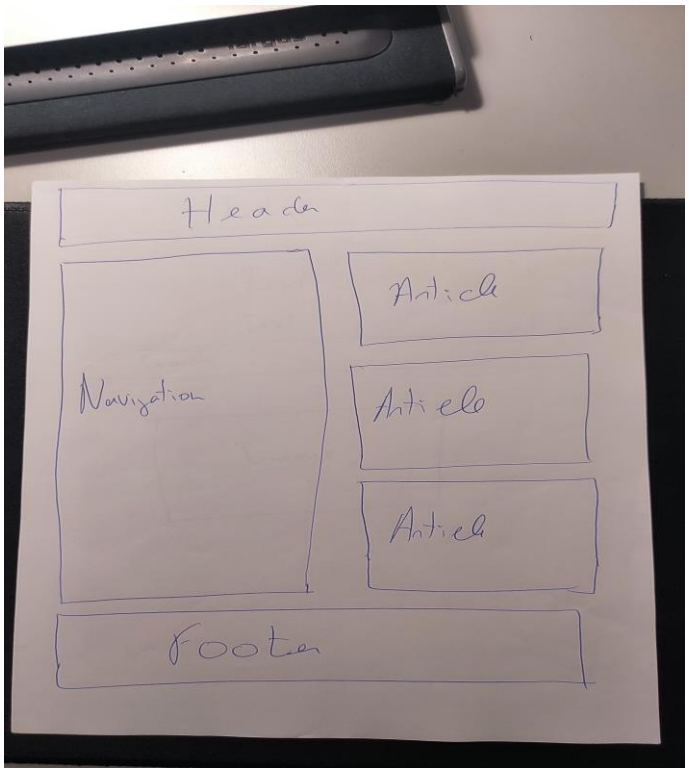


Figura 67 – *Layout* de uma página Web feito à mão.

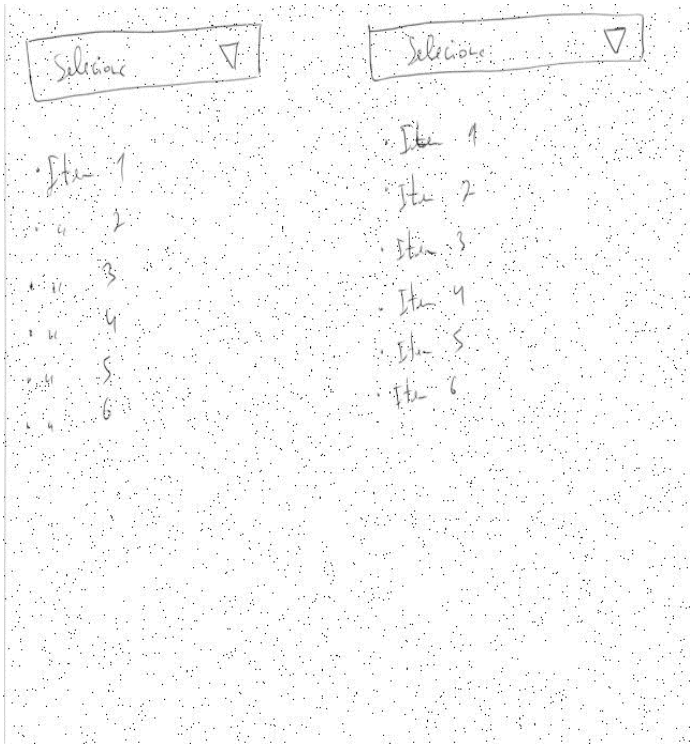


Figura 68 – Imagem original com ruído.

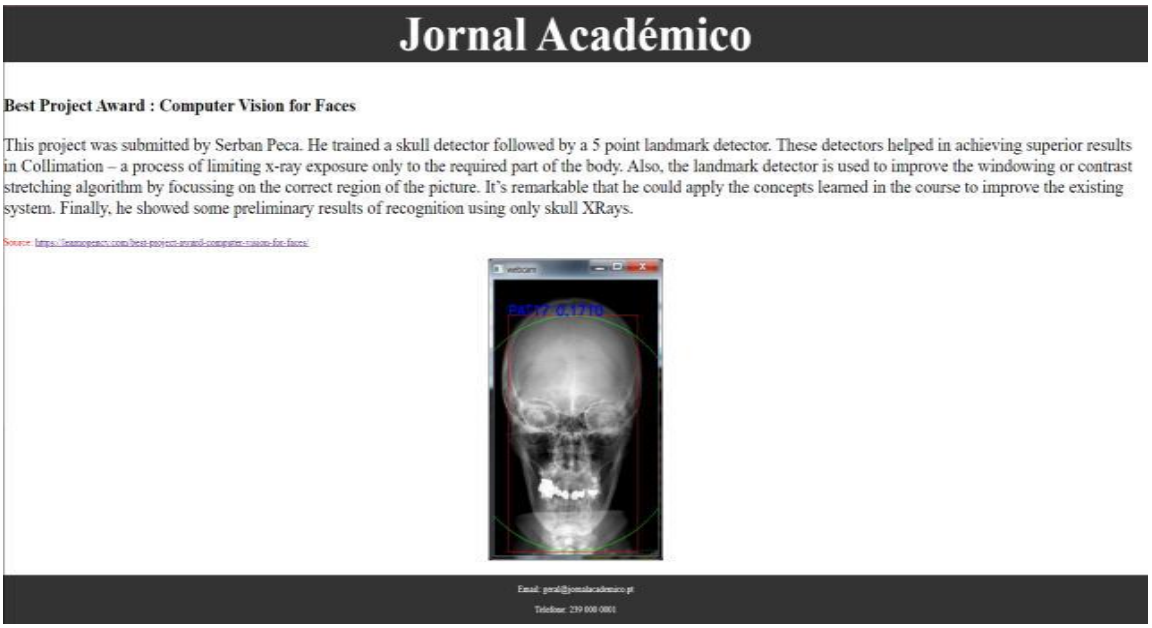


Figura 69 – Imagem do 3º *dataset* de Página WEB.

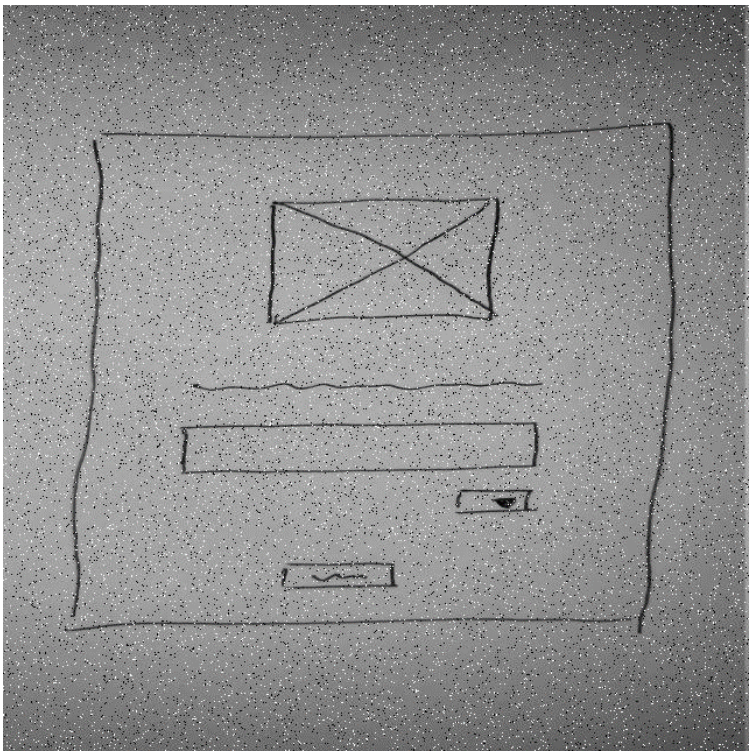


Figura 70 – *Layout* de um *Wireframe*.

Tabela 1 – Conjuntos de dados criados.

Imagens	Total	Treino	Teste	Validação	Total de classes
1º *Dataset*	44	35	5	3	5
2º *Dataset*	202	177	17	8	8
3º *Dataset*	48	42	3	3	8
4º *Dataset*	62	54	5	3	5
5º *Dataset*	312	273	25	14	8

5.2 Modo de execução

Após a criação dos *dataset*, sucedeu-se uma série de experimentos entre os modelos. Foram escolhidos 3 modelos para o sistema de deteção de objetos, o modelo YOLOv5, YOLOv8 e Detectron2. Nos modelos YOLOv5 e YOLOv8 como existem diferentes variantes: *nano*, *small*, *medium*, *large* e *x-large*, as variantes treinadas foram: *nano*, *medium* e *x-large*. No total foram treinados 7 modelos em diferentes épocas.

Cada modelo tem disponível um tutorial na plataforma GoogleColab, sendo possível executar o treino e obter todos os resultados na plataforma, não sendo necessário recorrer a nenhum editor de código nem a outra ferramenta fora da plataforma Colab [79] [80] [81].

5.3 Análise e desempenho dos modelos

Neste subcapítulo é feita uma análise aos resultados dos modelos de deteção de objetos, avaliando as *performances* de desempenho para as várias classes, obtidas em cada treino.

5.3.1 Análise e desempenho modelo YOLOv5

5.3.1.1 *2º Dataset* – *Layouts* de Páginas WEB feitos à mão em folhas brancas

A 1ª análise será feita ao *dataset* de *Layouts* de Páginas WEB feitos à mão em folhas brancas, utilizando o modelo YOLOv5.

Inicialmente foi estudado qual a variante do modelo que conseguia apresentar melhores resultados, treinando os modelos em diferentes épocas. No contexto de modelos de treino de *deep learning*, como o YOLO, uma época (*epoch*) refere-se a uma única passagem completa por todo o conjunto de treino durante o processo de treino do modelo. Em cada época, o modelo recebe todas as amostras do conjunto de treino para atualizar os seus pesos (*weights*). O treino de um modelo geralmente envolve várias épocas e a quantidade de épocas usadas podem ser ajustadas consoante o desempenho do modelo e a sua convergência. O objetivo é que à medida que o modelo vê os dados repetidos, o modelo deve melhorar a sua capacidade de generalização para além dos exemplos específicos do conjunto de treino. Em alguns casos, usar demasiadas épocas pode levar a um *overfitting*. Para controlar este problema, verifica-se a perda no conjunto de treino e no conjunto de validação, e se a perda no conjunto de validação começar a aumentar enquanto a perda no conjunto de treino continua a diminuir, isso pode indicar *overfitting*, e logo, interromper o treino pode ser a melhor opção. Para combater este problema automaticamente é possível implementar um método denominado por "*early stopping*", que especifica o número de épocas (x) consecutivas após as quais o modelo termina se não melhor o desempenho. Nos modelos implementados no projeto se

após 100 épocas não houver melhorias no desempenho, o treino termina e limitando-se a presença de *overfitting* nos pesos [82].

As Tabela 2, Tabela 3 e Tabela 4 apresentam os resultados da métrica *mean Average Precision* para diferentes épocas e treino do modelo YOLOv5. Analisando o mAP na Tabela 5 o modelo que atingiu melhores valores de mAP foi o modelo YOLOv5m treinado para 1000 épocas.

Tabela 2 – Métrica de desempenho mAP modelo YOLOv5n – 2º *Dataset*.

Modelo - Yolov5n														
	Classes	200		400		600		535/800		789/1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	All	0,826	0,649	0,977	0,777	0,956	0,774	0,973	0,787	0,975	0,797	0,977	0,797	789/1000	789/1000
1	Button	0,995	0,697	0,995	0,687	0,995	0,720	0,995	0,830	0,995	0,781	0,995	0,830	600	535/800
2	Combobox	0,548	0,446	0,964	0,697	0,826	0,650	0,978	0,687	0,944	0,677	0,978	0,697	535/800	400
3	Footer	0,869	0,740	0,884	0,715	0,955	0,814	0,898	0,749	0,908	0,775	0,955	0,814	600	600
4	Header	0,995	0,785	0,995	0,802	0,978	0,763	0,986	0,813	0,995	0,814	0,995	0,814	789/1000	789/1000
5	Image	0,995	0,900	0,995	0,903	0,995	0,903	0,995	0,892	0,995	0,906	0,995	0,906	789/1000	789/1000
6	List_Ordered	0,493	0,330	0,995	0,790	0,972	0,766	0,942	0,789	0,995	0,836	0,995	0,836	789/1000	789/1000
7	List_UnOrdered	0,776	0,560	0,995	0,821	0,933	0,784	0,995	0,737	0,972	0,806	0,995	0,821	535/800	400
8	Text	0,942	0,750	0,995	0,800	0,991	0,792	0,995	0,800	0,991	0,779	0,995	0,800	535/800	400

Tabela 3 - Métrica de desempenho mAP modelo YOLOv5m – 2º *Dataset*.

Modelo - Yolov5m														
	Classes	200		400		332/600		338/800		440/1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	All	0,962	0,784	0,957	0,806	0,979	0,801	0,957	0,800	0,981	0,824	0,981	0,824	440/1000	440/1000
1	Button	0,995	0,830	0,995	0,808	0,995	0,863	0,995	0,863	0,995	0,880	0,995	0,880	440/1000	440/1000
2	Combobox	0,995	0,708	0,804	0,610	0,995	0,756	0,964	0,698	0,995	0,711	0,995	0,756	440/1000	600
3	Footer	0,954	0,764	0,926	0,836	0,912	0,743	0,845	0,689	0,898	0,744	0,954	0,836	200	400
4	Header	0,995	0,804	0,995	0,814	0,995	0,814	0,995	0,838	0,995	0,823	0,995	0,838	440/1000	800
5	Image	0,995	0,918	0,995	0,900	0,995	0,885	0,995	0,899	0,995	0,917	0,995	0,918	440/1000	200
6	List_Ordered	0,859	0,662	0,995	0,820	0,972	0,777	0,898	0,790	0,995	0,921	0,995	0,921	440/1000	440/1000
7	List_UnOrdered	0,911	0,767	0,988	0,830	0,971	0,784	0,973	0,788	0,976	0,801	0,988	0,830	400	400
8	Text	0,995	0,816	0,995	0,834	0,995	0,786	0,991	0,836	0,995	0,793	0,995	0,836	1000	800

Tabela 4 - Métrica de desempenho mAP modelo YOLOv5x – 2º *Dataset*.

Modelo - Yolov5x														
	Classes	200		400		600		800		534/1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	All	0,954	0,773	0,943	0,818	0,958	0,812	0,973	0,829	0,953	0,809	0,973	0,829	800	800
1	Button	0,995	0,813	0,995	0,863	0,995	0,798	0,995	0,720	0,995	0,848	0,995	0,863	534/1000	400
2	Combobox	0,995	0,745	0,995	0,810	0,904	0,700	0,995	0,818	0,995	0,760	0,995	0,818	534/1000	800
3	Footer	0,884	0,678	0,967	0,820	0,907	0,751	0,954	0,824	0,954	0,813	0,967	0,824	400	800
4	Header	0,995	0,841	0,978	0,840	0,977	0,830	0,955	0,856	0,995	0,852	0,995	0,856	534/1000	800
5	Image	0,995	0,892	0,995	0,937	0,995	0,891	0,995	0,941	0,995	0,904	0,995	0,941	534/1000	800
6	List_Ordered	0,851	0,693	0,713	0,633	0,924	0,840	0,955	0,836	0,863	0,741	0,955	0,840	800	600
7	List_UnOrdered	0,919	0,706	0,907	0,819	0,967	0,868	0,902	0,850	0,828	0,747	0,967	0,868	600	600
8	Text	0,995	0,813	0,995	0,818	0,995	0,821	0,995	0,828	0,995	0,807	0,995	0,828	800	800

Tabela 5 - Métrica de desempenho mAP entre todos os modelos – 2º *Dataset*.

All															
Variante		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
n	*YOLOv5n*	*0,826*	*0,649*	*0,977*	*0,777*	*0,956*	*0,774*	*0,973*	*0,787*	*0,975*	*0,797*	0,977	0,797	789/1000	789/1000
m	*YOLOv5m*	*0,962*	*0,784*	*0,957*	*0,806*	*0,979*	*0,801*	*0,957*	*0,800*	*0,981*	*0,824*	0,981	0,824	440/1000	440/1000
x	*YOLOv5x*	*0,954*	*0,773*	*0,943*	*0,818*	*0,958*	*0,812*	*0,973*	*0,829*	*0,953*	*0,809*	0,973	0,829	800	800

O modelo YOLOv5m treinado para 1000 épocas atingiu um valor de mAP50 de 98.1% e um mAP95 de 82.4%. De referir que o modelo não chegou a completar as 1000 épocas, e isto deve-se ao método *early stopping*. O treino foi interrompido na época 440, pois já não apresentava melhorias nas 100 épocas anteriores.

Conforme é possível ver na Figura 71 este modelo analisou um total de 17 imagens com um total de 73 rótulos (*Instances*), obtendo uma precisão geral de 94.6% e um *Recall* de 98.1%.

```
Stopping training early as no improvement observed in last 100 epochs. Best results observed at epoch 339, best model saved as best.pt.
To update EarlyStopping(patience=100) pass a new patience value, i.e. `python train.py --patience 300` or use `--patience 0` to disable EarlyStopping.

440 epochs completed in 0.638 hours.
Optimizer stripped from runs/train/exp/weights/last.pt, 42.3MB
Optimizer stripped from runs/train/exp/weights/best.pt, 42.3MB

Validating runs/train/exp/weights/best.pt...
Fusing layers...
Model summary: 212 layers, 20881221 parameters, 0 gradients, 47.9 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  3.39it/s]
                   all         17         73      0.946      0.981      0.981      0.824
                Button         17          3      0.961          1      0.995       0.88
              ComboBox         17          7      0.992          1      0.995      0.711
                Footer         17          8      0.816          1      0.898      0.744
                Header         17         10          1      0.963      0.995      0.823
                 Image         17         12      0.983          1      0.995      0.917
          List_Ordered         17          6      0.833          1      0.995      0.921
        List_Unordered         17         11          1      0.885      0.976      0.801
                  Text         17         16      0.983          1      0.995      0.793
Results saved to runs/train/exp
```

Figura 71 – Métricas do modelo YOLOv5m, Época=1000, 2º *Dataset*.

A função *Loss* refere-se à medida cumulativa de quão bem o modelo está a desempenhar-se durante o treino e é projetada para penalizar os erros de predição das *bounding boxes* e na classificação de objetos. O objetivo é tornar o modelo mais preciso e capaz de detetar objetos com precisão nas imagens.

Na Figura 72 são apresentados 3 tipos de gráficos para o conjunto de treino e o conjunto de validação. Os gráficos em cima representam o valor de perda de treino que é medida para durante cada época, enquanto os gráficos debaixo representam o valor de perda de validação que é medida após cada época. A função *Box Loss* mede o quão preciso o modelo está a prever as coordenadas das caixas delimitadoras dos objetos na imagem. Esta função calcula a diferença entre as coordenadas previstas das caixas e as coordenadas reais das caixas. A função *Objectness Loss* penaliza o modelo por fazer previsões erradas com pontuações de confiança elevadas ou por não detetar objetos com pontuações de confianças baixas. A função *Classification Loss* considera a precisão das previsões de classes de objetos, medindo a diferença entre as probabilidades de classe previstas e os rótulos de classe reais dos objetos. Analisando ambos os conjuntos de perda é possível verificar que no *dataset* de validação o valor de perda não diminui igualmente com o *dataset* de treino,

significando que o modelo não melhora a partir de uma determinada época. Neste caso o modelo deixa de apresentar melhorias de treino sensivelmente a partir da época 200 e é por isso que o *early stopping* acabou por ser ativado para o treino em questão.

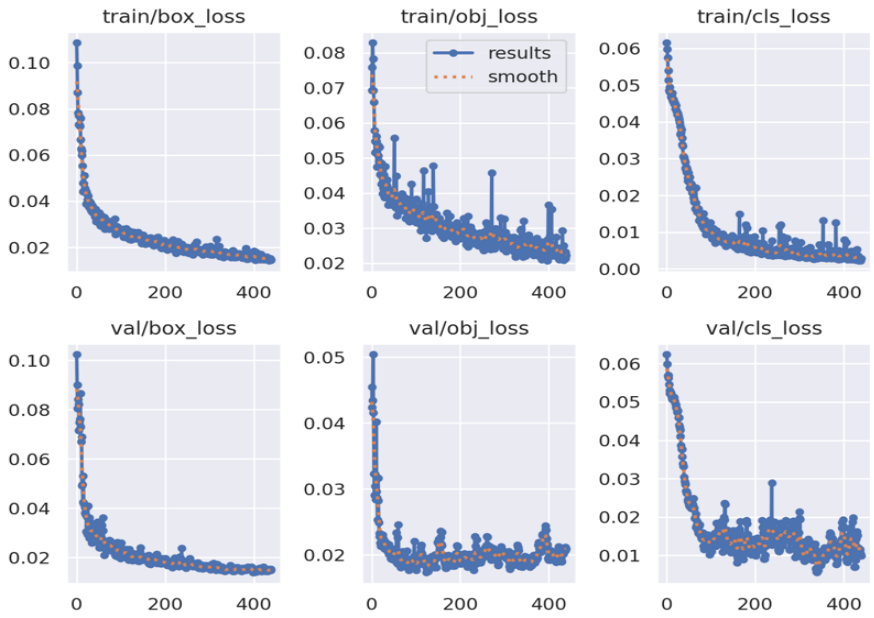
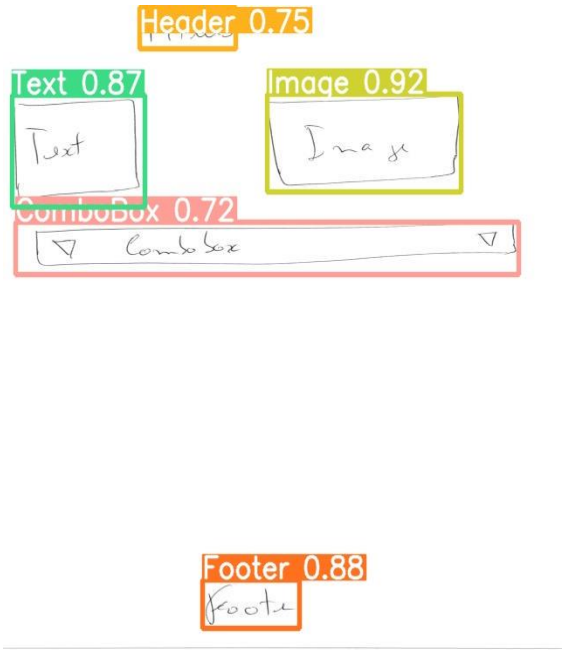
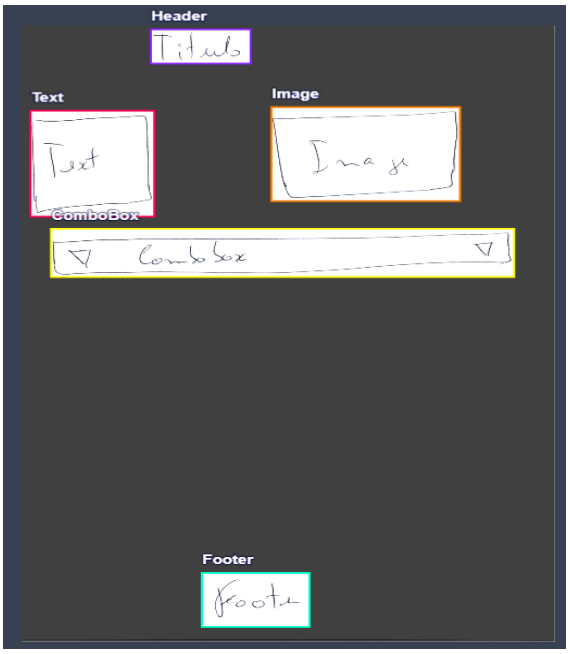


Figura 72 – Funções de perda (*Loss*) do modelo YOLOv5m, Época=1000, 2º *Dataset*.

Nas Figura 73 e Figura 74 são apresentados alguns exemplos de deteção de objetos em imagens de teste. Os resultados obtidos apresentam bons valores de precisão, acertando em todas as categorias e com uma percentagem de precisão sempre acima dos 70%. As imagens à direita são as imagens rotuladas na plataforma Roboflow com as categorias presentes na imagem e que depois são comparadas com as categorias detetadas, nomeadamente a categoria "*Header*", "*Test*", "*Image*", "*ComboBox*", "*Footer*", "*List_Unordered*" e "*List_Ordered*".

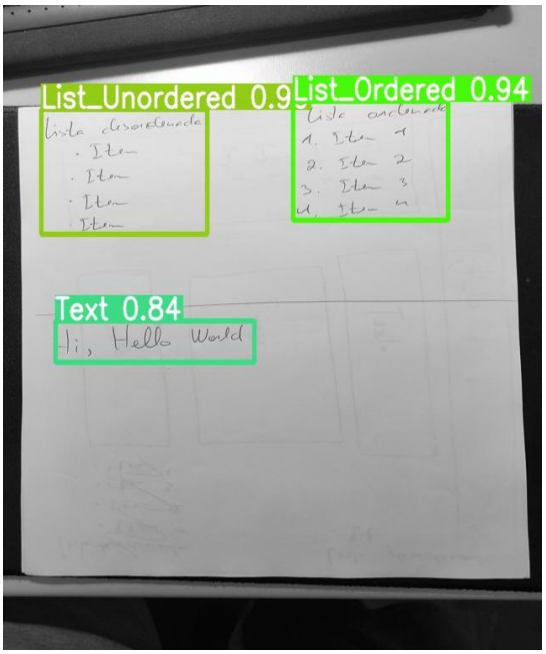


a) Deteção numa imagem de teste.

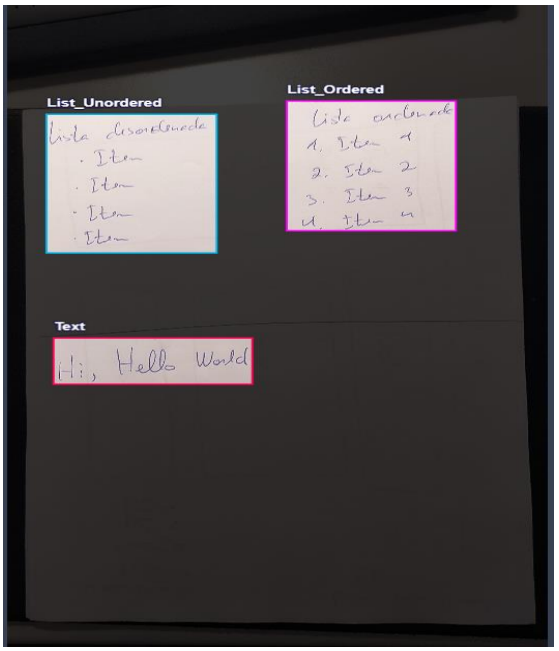


b) Imagem rotulada com as diferentes categorias.

Figura 73 – Exemplo de Imagem de teste – 2º *Dataset*.



a) Deteção numa imagem de teste.



b) Imagem rotulada com as diferentes categorias.

Figura 74 – Exemplo de uma 2ª Imagem de teste – 2º *Dataset*.

5.3.1.2 *3º Dataset* – Páginas WEB desenvolvidas em HTML e CSS

A 2ª análise é feita a um conjunto de páginas WEB desenvolvidas em HTML e CSS. O modo de procedimento será igual à 1ª análise. Verificando as Tabela 6,

Tabela 7, Tabela 8 e Tabela 9 é possível concluir que os melhores resultados foram obtidos no modelo YOLOv5x com 600 épocas de treino, atingindo um mAP50 de 96.5% e um mAP95 de 79.6%. Neste sistema de treino o modelo também não completou as 600 épocas, terminou 10 épocas antes. Mais uma vez o *early stopping* foi ativado visto não haver melhorias de resultados nas épocas anteriores e de facto se analisarmos a Figura 76 há uma estagnação na melhoria de resultados a partir da época 300.

Tabela 6 - Métrica de desempenho mAP modelo YOLOv5n – 3º *Dataset*.

Modelo - Yolov5n														
	Classes	200		400		600		546/800		514/1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	All	0,546	0,381	0,674	0,520	0,874	0,638	0,894	0,657	0,879	0,642	0,894	0,657	546/800	546/800
1	Button	0,388	0,210	0,746	0,565	0,913	0,617	0,995	0,656	0,995	0,664	0,995	0,664	546/800	514/1000
2	Combobox	0,555	0,458	0,888	0,682	0,995	0,847	0,995	0,834	0,945	0,762	0,995	0,847	600	600
3	Footer	0,05	0,025	0,111	0,100	0,497	0,448	0,995	0,895	0,995	0,796	0,995	0,895	546/800	546/800
4	Header	0,551	0,126	0,54	0,273	0,828	0,203	0,995	0,498	0,828	0,531	0,995	0,531	546/800	514/1000
5	Image	0,995	0,825	0,995	0,936	0,995	0,906	0,995	0,918	0,995	0,829	0,995	0,936	200	400
6	List_Ordered	0,995	0,796	0,995	0,895	0,995	0,895	0,497	0,398	0,995	0,697	0,995	0,895	200	400
7	List_UnOrdered	0,497	0,448	0,497	0,448	0,995	0,796	0,995	0,697	0,497	0,497	0,995	0,796	600	600
8	Text	0,336	0,157	0,62	0,264	0,774	0,391	0,684	0,361	0,784	0,359	0,784	0,391	0,514	600

Tabela 7 - Métrica de desempenho mAP modelo YOLOv5m – 3º *Dataset*.

Modelo - Yolov5m														
	Classes	200		366/400		367/600		440/800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	All	0,642	0,817	0,645	0,878	0,634	0,826	0,679	0,880	"-"	"-"	0,679	0,880	440/800	440/800
1	Button	0,656	0,830	0,664	0,995	0,699	0,995	0,731	0,995	"-"	"-"	0,731	0,995	366/400	440/800
2	Combobox	0,747	0,995	0,735	0,995	0,787	0,945	0,761	0,995	"-"	"-"	0,787	0,995	200	367/600
3	Footer	0,133	0,166	0,348	0,497	0,224	0,249	0,398	0,497	"-"	"-"	0,398	0,497	366/400	440/800
4	Header	0,385	0,828	0,548	0,828	0,465	0,828	0,448	0,828	"-"	"-"	0,548	0,828	200	366/400
5	Image	0,933	0,995	0,895	0,995	0,895	0,995	0,882	0,995	"-"	"-"	0,933	0,995	200	200
6	List_Ordered	0,895	0,995	0,697	0,995	0,769	0,995	0,896	0,995	"-"	"-"	0,896	0,995	200	440/800
7	List_UnOrdered	0,995	0,995	0,895	0,995	0,895	0,995	0,995	0,995	"-"	"-"	0,995	0,995	200	200
8	Text	0,392	0,732	0,378	0,723	0,310	0,608	0,421	0,741	"-"	"-"	0,421	0,741	440/800	440/800

Tabela 8 - Métrica de desempenho mAP modelo YOLOv5x – 3º *Dataset*.

Modelo - Yolov5x														
	Classes	200		400		590/600		431/800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	All	0,869	0,666	0,925	0,784	0,965	0,796	0,881	0,707	"-"	"-"	0,965	0,796	590/600	590/600
1	Button	0,995	0,753	0,995	0,798	0,995	0,750	0,995	0,796	"-"	"-"	0,995	0,798	200	400
2	Combobox	0,990	0,766	0,995	0,895	0,995	0,760	0,995	0,809	"-"	"-"	0,995	0,895	400	400
3	Footer	0,497	0,398	0,995	0,895	0,995	0,995	0,995	0,995	"-"	"-"	0,995	0,995	400	590/600
4	Header	0,745	0,498	0,745	0,363	0,995	0,673	0,828	0,549	"-"	"-"	0,995	0,673	590/600	590/600
5	Image	0,995	0,935	0,995	0,982	0,995	0,930	0,995	0,862	"-"	"-"	0,995	0,982	200	400
6	List_Ordered	0,995	0,796	0,995	0,895	0,995	0,796	0,497	0,348	"-"	"-"	0,995	0,895	200	400
7	List_UnOrdered	0,995	0,796	0,995	0,995	0,995	0,995	0,995	0,796	"-"	"-"	0,995	0,995	200	400
8	Text	0,737	0,388	0,684	0,448	0,755	0,492	0,745	0,500	"-"	"-"	0,755	0,500	590/600	431/800

Tabela 9 - Métrica de desempenho mAP entre todos os modelos – 3º *Dataset*.

All															
Variante		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
n	YOLOv5n	0,546	0,381	0,674	0,520	0,874	0,638	0,894	0,657	0,879	0,642	0,894	0,657	546/800	546/800
m	YOLOv5m	0,642	0,817	0,645	0,878	0,634	0,826	0,679	0,880	"-"	"-"	0,679	0,880	440/800	440/800
x	YOLOv5x	0,869	0,666	0,925	0,784	0,965	0,796	0,881	0,707	"-"	"-"	0,965	0,796	590/600	590/600

Conforme é possível ver na Figura 75 este modelo testou um total de 3 imagens de teste com um total de 24 rótulos (*Instances*), obtendo uma precisão geral de 73.4% e um Recall de 94.9%. A Figura 76 apresenta os gráficos de desempenho.

```
Stopping training early as no improvement observed in last 100 epochs. Best results observed at epoch 489, best model saved as best.pt.
To update EarlyStopping(patience=100) pass a new patience value, i.e. `python train.py --patience 300` or use `--patience 0` to disable EarlyStopping.

590 epochs completed in 1.100 hours.
Optimizer stripped from runs/train/exp/weights/last.pt, 173.2MB
Optimizer stripped from runs/train/exp/weights/best.pt, 173.2MB

Validating runs/train/exp/weights/best.pt...
Fusing layers...
Model summary: 322 layers, 86220517 parameters, 0 gradients, 203.9 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  5.55it/s]
                   all          3         24      0.734      0.949      0.965      0.796
                Button          3          3      0.878          1      0.995       0.75
             Combobox           3          4      0.871          1      0.995      0.736
                Footer          3          1      0.733          1      0.995      0.995
                Header          3          2          1      0.791      0.995      0.673
                 Image          3          7      0.935          1      0.995       0.93
          List ordered          3          1      0.495          1      0.995      0.796
        List unordered          3          1       0.48          1      0.995      0.995
                  Text          3          5       0.48        0.8      0.755      0.492
Results saved to runs/train/exp
```

Figura 75 - Métricas do modelo YOLOv5x, Época=600, 3º *Dataset*.

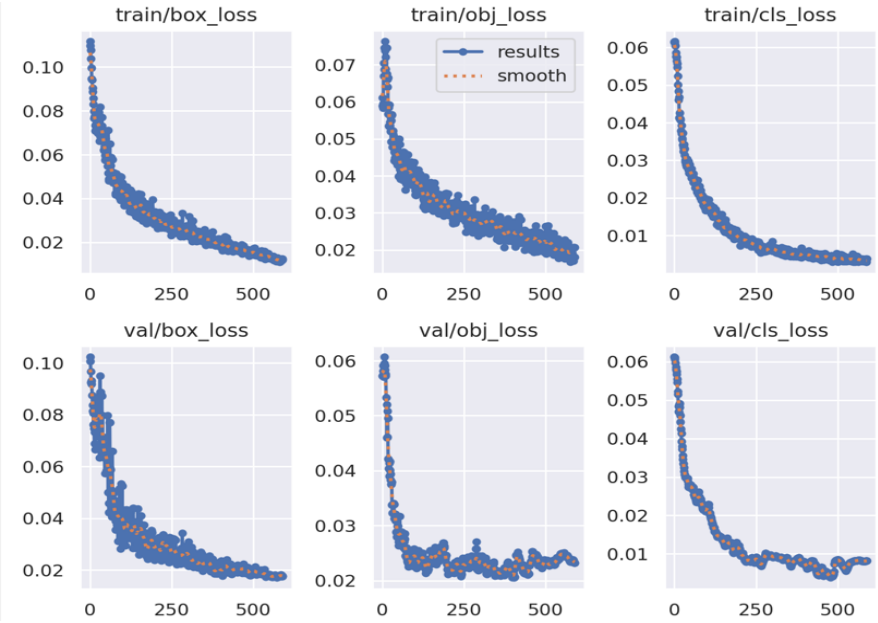
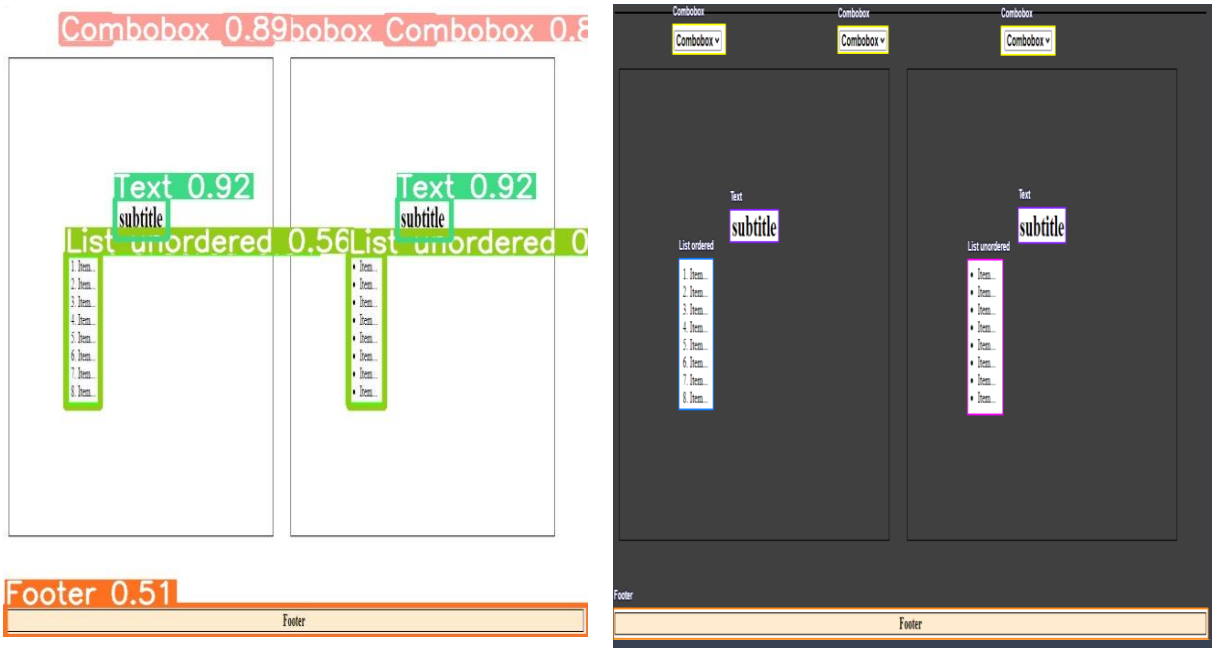


Figura 76 – Funções de perda (*Loss*) do modelo YOLOv5x, Época=600, 3º *Dataset*.

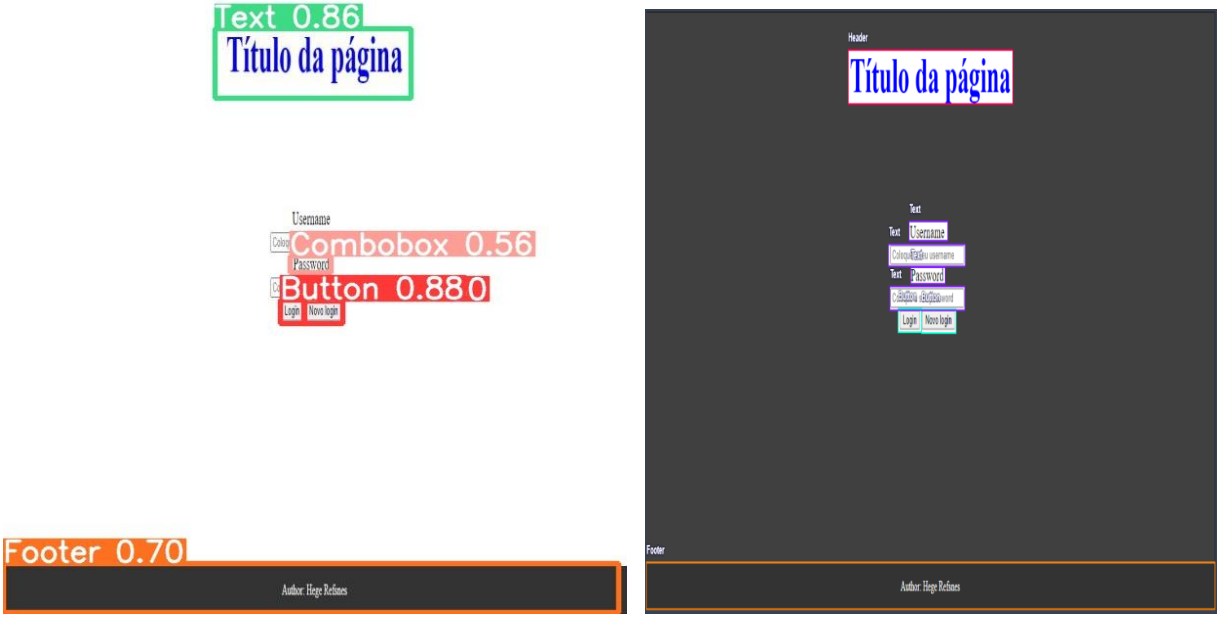
Na Figura 77 todos os resultados foram certeiros, apesar de haver três categorias, nomeadamente a categoria "*List_unordered*", "*List_ordered*" e "*Footer*" que

apresentaram uma precisão pouco acima dos 50%. Na Figura 78 não conseguiu detetar todas as categorias, faltando detetar a categoria "*Text*" e confundiu uma categoria "*Text*" com uma categoria "*ComboBox*".



a) Deteção numa imagem de teste. b) Imagem rotulada com as diferentes categorias.

Figura 77 – Exemplo de Imagem de teste –3º *Dataset*.



a) Deteção numa imagem de teste. b) Imagem rotulada com as diferentes categorias.

Figura 78 – Exemplo de uma 2ª Imagem de teste – 3º *Dataset*.

5.3.1.3 *4º Dataset – Wireframes*

Na 3º análise treinou-se um conjunto de imagens para detetar as *tags* em *Wireframes*. Mais uma vez, analisando as Tabela 10, Tabela 11, Tabela 12 e Tabela 13 verifica-se que os melhores resultados dão-se no modelo Yolov5x treinado para 400 épocas. O treino não chegou a completar as 400 épocas, terminando o programa na época 366 com um mAP50 de 63% e um mAP95 38%.

Nos modelos YOLOv5n e YOLOv5x não foram treinados modelos com 1000 épocas. Como em ambos os modelos desde da época 400 nunca foi possível completar as 1000 épocas, não houve necessidade de treinar estes modelos com treinos de 1000 épocas.

Tabela 10 - Métrica de desempenho mAP modelo YOLOv5n – 4º *Dataset*.

Modelo - Yolov5n														
Classes		200		360/400		360/600		362/800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,487*	*0,227*	0,457	0,257	0,56	0,295	0,513	0,266	"-"	"-"	0,560	0,295	360/600	360/600
1	*Button*	*0,499*	*0,240*	0,443	0,225	0,716	0,380	0,665	0,710	"-"	"-"	0,716	0,710	360/600	362/800
2	*Combobox*	*0,035*	*0,014*	0,029	0,013	0,0293	0,013	0,0347	0,016	"-"	"-"	0,035	0,016	200	362/800
3	*Image*	*0,795*	*0,483*	0,829	0,569	0,904	0,580	0,839	0,518	"-"	"-"	0,904	0,580	360/600	360/600
4	*Text*	*0,618*	*0,172*	0,527	0,223	0,614	0,208	0,513	0,161	"-"	"-"	0,618	0,223	200	360/400

Tabela 11 - Métrica de desempenho mAP modelo YOLOv5m – 4º *Dataset*.

Modelo - Yolov5m														
Classes		200		400		306/600		317/800		240/1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,607*	*0,354*	0,614	0,353	0,628	0,339	0,561	0,335	0,349	0,490	0,628	0,490	306/600	240/1000
1	*Button*	*0,875*	*0,421*	0,8	0,4	0,832	0,393	0,744	0,436	0,449	0,449	0,875	0,449	200	240/1000
2	*Combobox*	*0,007*	*0,00211*	0,0369	0,0117	0,0409	0,007	0,0159	0,007	0,0067	0,007	0,041	0,012	306/600	400
3	*Image*	*0,85*	*0,685*	0,869	0,661	0,95	0,651	0,836	0,627	0,75	0,750	0,950	0,750	306/600	240/1000
4	*Text*	*0,697*	*0,306*	0,751	0,34	0,69	0,305	0,649	0,270	0,191	0,191	0,751	0,340	400	400

Tabela 12 - Métrica de desempenho mAP modelo YOLOv5x – 4º *Dataset*.

Modelo - Yolov5x														
Classes		200		366/400		255/600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,601*	*0,374*	0,63	0,38	0,606	0,356	"-"	"-"	"-"	"-"	0,630	0,380	366/400	366/400
1	*Button*	*0,696*	*0,427*	0,828	0,418	0,869	0,386	"-"	"-"	"-"	"-"	0,869	0,427	255/600	200
2	*Combobox*	*0,018*	*0,014*	0,0951	0,0712	0,0192	0,0134	"-"	"-"	"-"	"-"	0,095	0,071	366/400	366/400
3	*Image*	*0,939*	*0,744*	0,873	0,668	0,947	0,707	"-"	"-"	"-"	"-"	0,947	0,744	255/600	200
4	*Text*	*0,752*	*0,312*	0,724	0,362	0,589	0,317	"-"	"-"	"-"	"-"	0,752	0,362	200	366/400

Tabela 13 - Métrica de desempenho mAP entre todos os modelos – 4º *Dataset*.

All															
Variante		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
n	YOLOv5n	0,487	0,227	0,457	0,257	0,560	0,295	0,513	0,266	"-"	"-"	0,560	0,295	360/600	360/600
m	YOLOv5m	0,607	0,354	0,614	0,353	0,628	0,339	0,561	0,335	0,349	0,490	0,628	0,490	306/600	240/1000
x	YOLOv5x	0,601	0,374	0,630	0,380	0,606	0,356	"-"	"-"	"-"	"-"	0,630	0,380	366/400	366/400

Analisando os resultados da Figura 79, este modelo testou 5 imagens com 61 rótulos com uma precisão de 77,6% e um Recall de 58.7%. De uma forma geral os valores são satisfatórios, com valores acima dos 50%, tirando o valor da classe "*ComboBox*". A classe "*ComboBox*" obteve valores bastante baixos relativamente às outras classes, isto deve-se em grande parte à quantidade de imagens que apresentam a tag "*Combobox*". A Figura 80 apresenta os gráficos de desempenho.

```
Stopping training early as no improvement observed in last 100 epochs. Best results observed at epoch 265, best model saved as best.pt.
To update EarlyStopping(patience=100) pass a new patience value, i.e. `python train.py --patience 300` or use `--patience 0` to disable EarlyStopping.

366 epochs completed in 1.089 hours.
Optimizer stripped from runs/train/exp/weights/last.pt, 173.2MB
Optimizer stripped from runs/train/exp/weights/best.pt, 173.2MB

Validating runs/train/exp/weights/best.pt...
Fusing layers...
Model summary: 322 layers, 86200330 parameters, 0 gradients, 203.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  4.07it/s]
                   all          5         61      0.776      0.587       0.63       0.38
                Button          5         17          1      0.558      0.828      0.418
              ComboBox          5          3          1          0     0.0951     0.0712
                 Image          5         26       0.64      0.923      0.873      0.668
                  Text          5         15      0.466      0.867      0.724      0.362
Results saved to runs/train/exp
```

Figura 79 - Métricas do modelo YOLOv5x, Época=400, 4º *Dataset*.

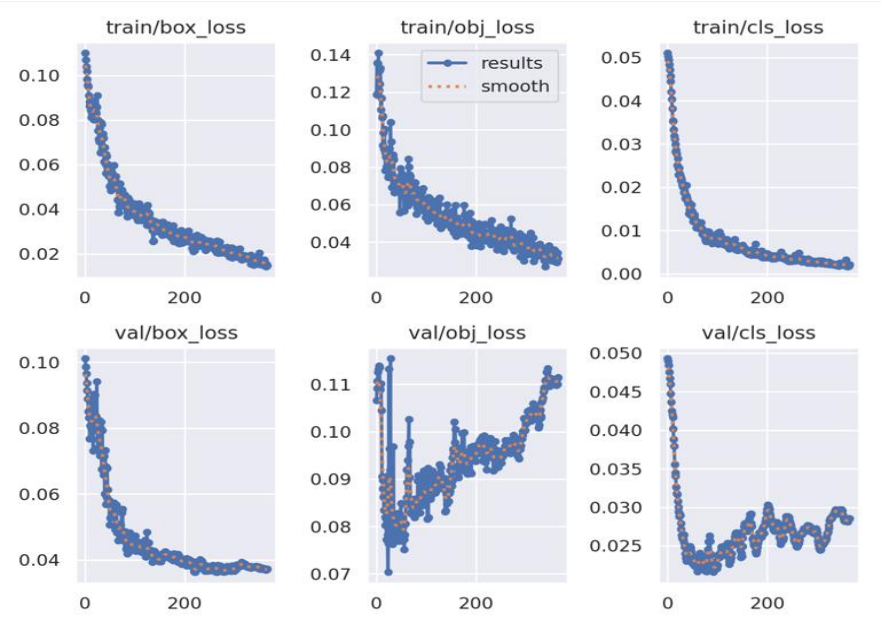
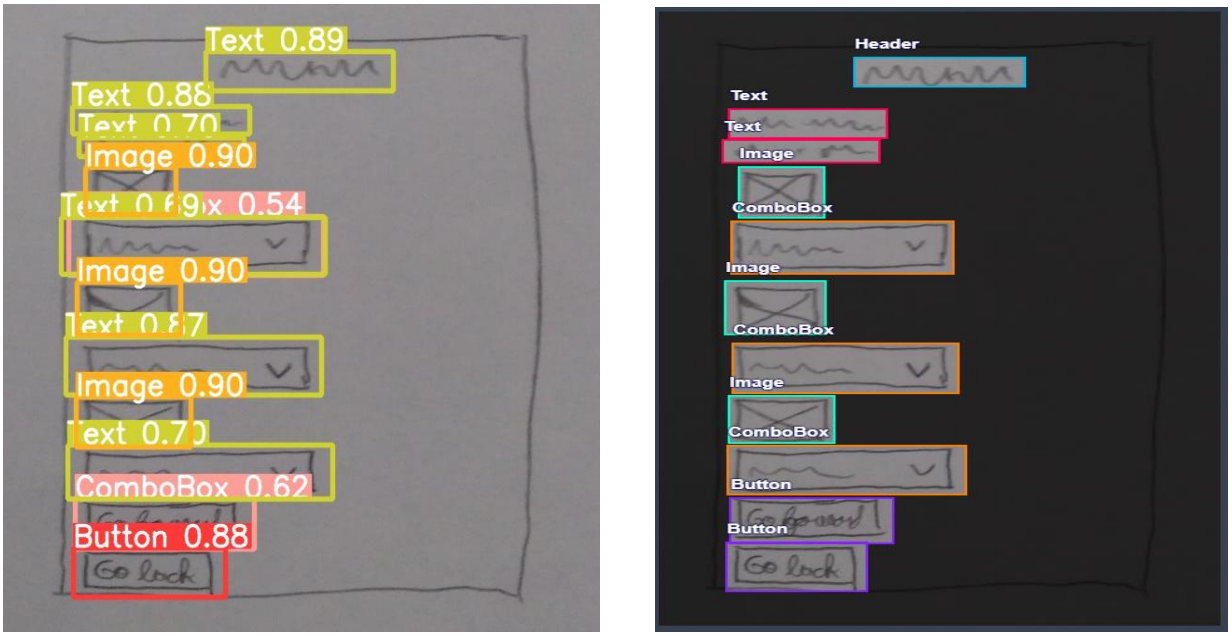


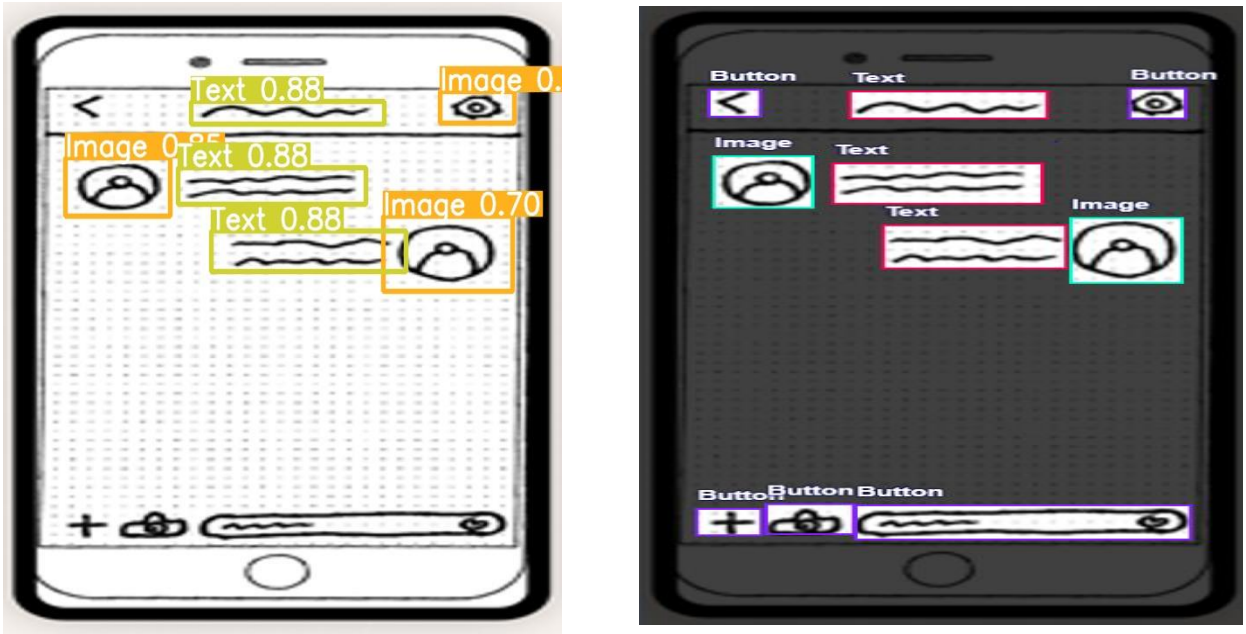
Figura 80 - Funções de perda (*Loss*) do modelo YOLOv5x, Época=600, 4º *Dataset*.

Analisando a Figura 81, o sistema logo na primeira *tag* erra a deteção. Atribui uma categoria "*text*" quando devia ter detetado uma *tag "Header"*. Na 1º tag "*Combobox"* o sistema deteta corretamente, mas deteta também uma *tag* "*text*" que também está errado e erra as restantes *tags* "*Combobox*". Erra também na deteção da 1º *tag* "*Button*" detetando uma *tag* "*Combobox*", as restantes deteções estão corretas. Na Figura 82 acertou em todas os elementos, mas não detetou a maioria das classes "*button*".



a) Deteção numa imagem de teste. b) Imagem rotulada com as diferentes categorias.

Figura 81 – Exemplo de Imagem de teste – 4º *Dataset.*



a) Deteção numa imagem de teste. b) Imagem rotulada com as diferentes categorias.

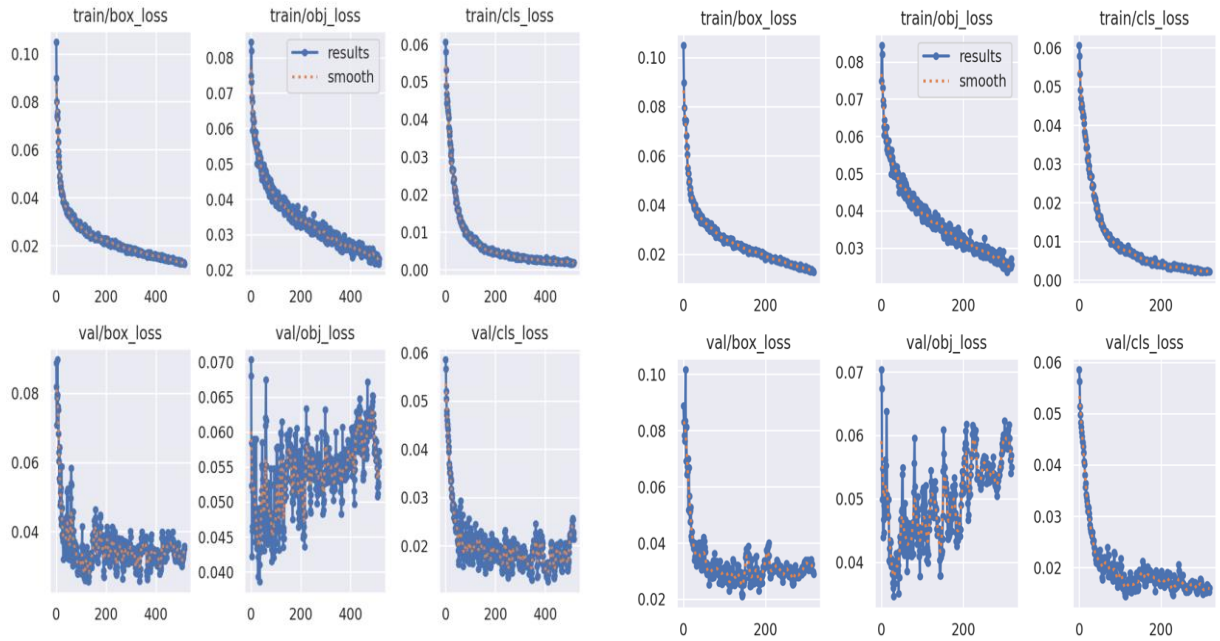
Figura 82 – Exemplo de uma 2ª Imagem de teste – 4º *Dataset.*

5.3.1.4 *5º Dataset* – União entre o 2º, 3º e 4º *dataset*

A quarta análise é a união entre todos os conjuntos de dados criados. O modelo irá ser treinado para detetar as classes em *wireframes*, páginas WEB HTML e *layouts* em páginas em branco. De igual modo às análises anteriores verificaram-se as tabelas do modelo treinado, nomeadamente as Tabela 15, Tabela 16, Tabela 17 e Tabela 18, e o melhor resultado obtido em métrica mAP é o modelo YOLOv5x, treinado para 800 épocas, com um mAP50 de 87,1% e um mAP95 de 69,8%. Mais uma vez o modelo não completou todas as épocas de treino, terminou ao fim de 516 épocas e o modelo indicou que os melhores resultados se deram na 415ª época. Visto que o melhor resultado deu-se a metade da época solicitada, será comparado o modelo de treino entre as épocas 400 e 800. Na Tabela 14, avaliando as métricas de desempenho o treino com 800 épocas consegue resultados ligeiramente superiores, tirando a métrica Recall que consegue estar acima 2 pontos percentuais. Relativamente às funções de perda o modelo treinado para 400 épocas tem um *overfitting* ligeiramente menor e por fim ao comparar a deteção de imagens à época 800 consegue apresentar resultados um pouco melhores (ver Figura 83).

Tabela 14 – Comparação de métricas entre a época 400 e 800 no modelo de treino YOLOv8x.

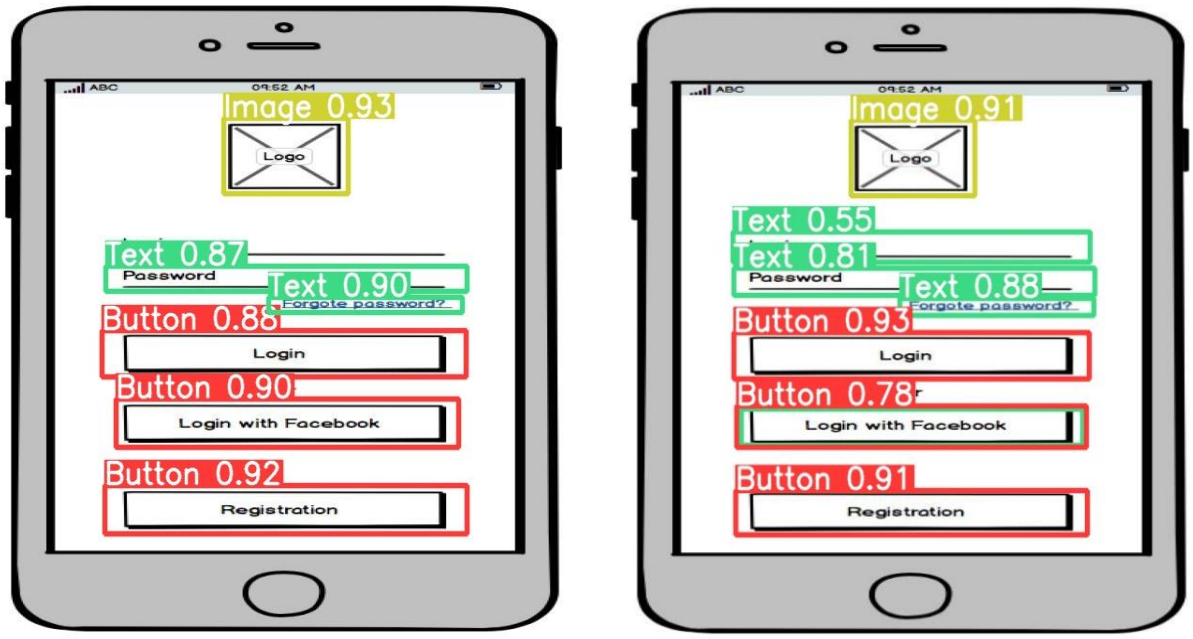
Época	Precisão	Recall	mAP50	mAP95
400	80%	81,1%	86,1%	68,3
800	87,4	78,5%	87,1	69,8



a) Funções de perda na Época 800.

b) Funções de perda na Época 400.

Figura 83 - Funções de perda (*Loss*) do modelo YOLOv5x, 5º *Dataset*.



a) Deteção numa imagem de teste época 800. b) Deteção numa imagem de teste época 400.

Figura 84 – Deteção em *wireframe* às épocas 400 e 800 para a mesma imagem, modelo YOLOv5x, 5º *Dataset*.

a) Deteção numa imagem de teste época 800. b) Deteção numa imagem de teste época 400.

Figura 85 – Exemplo de Imagem de teste, modelo YOLOv5x – 5º *Dataset*.

a) Deteção numa imagem de teste época 800. b) Deteção numa imagem de teste época 400.

Figura 86 – Exemplo de 2ª Imagem de teste, modelo YOLOv5x – 5º *Dataset*.

As Figura 84, Figura 85 e Figura 86 apresentam os resultados de deteção das classes, às épocas 400 e 800, para três conjunto de imagens, sendo a primeira de uma imagem em *wireframe*.

Tabela 15 - Métrica de desempenho mAP modelo YOLOv5n – 5º *Dataset*.

Modelo - Yolov5n														
	Classes	200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,665*	*0,462*	0,768	0,571	0,82	0,602	0,812	0,628	0,817	0,595	0,820	0,628	600	800
1	*Button*	*0,8*	*0,484*	0,713	0,486	0,791	0,498	0,708	0,469	0,713	0,495	0,800	0,498	200	600
2	*Combobox*	*0,602*	*0,476*	0,701	0,560	0,747	0,605	0,795	0,632	0,740	0,577	0,795	0,632	800	800
3	*Footer*	*0,599*	*0,448*	0,941	0,727	0,951	0,738	0,94	0,766	0,965	0,69	0,965	0,766	1000	800
4	*Header*	*0,665*	*0,409*	0,872	0,631	0,891	0,659	0,848	0,640	0,929	0,673	0,929	0,673	1000	1000
5	*Image*	*0,783*	*0,612*	0,753	0,583	0,777	0,600	0,794	0,643	0,762	0,564	0,794	0,643	800	800
6	*List_Ordered*	*0,433*	*0,309*	0,592	0,426	0,746	0,553	0,853	0,704	0,913	0,664	0,913	0,704	1000	800
7	*List_UnOrdered*	*0,729*	*0,494*	0,83	0,639	0,903	0,629	0,823	0,642	0,790	0,614	0,903	0,642	600	800
8	*Text*	*0,705*	*0,463*	0,742	0,515	0,756	0,532	0,739	0,527	0,725	0,486	0,756	0,532	600	600

Tabela 16 - Métrica de desempenho mAP modelo YOLOv5m – 5º *Dataset*.

Modelo - Yolov5m														
	Classes	200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,782*	*0,607*	0,840	0,669	0,841	0,643	0,812	0,628	0,817	0,595	0,841	0,669	600	400
1	*Button*	*0,695*	*0,445*	0,720	0,493	0,702	0,474	0,708	0,469	0,713	0,495	0,720	0,495	400	1000
2	*Combobox*	*0,771*	*0,628*	0,805	0,655	0,816	0,661	0,795	0,632	0,740	0,577	0,816	0,661	600	600
3	*Footer*	*0,899*	*0,742*	0,961	0,784	0,962	0,775	0,940	0,766	0,965	0,690	0,965	0,784	1000	400
4	*Header*	*0,940*	*0,732*	0,926	0,716	0,893	0,643	0,848	0,640	0,929	0,673	0,940	0,732	200	200
5	*Image*	*0,893*	*0,733*	0,893	0,746	0,891	0,715	0,794	0,643	0,762	0,564	0,893	0,746	200	400
6	*List_Ordered*	*0,465*	*0,369*	0,771	0,662	0,786	0,617	0,853	0,704	0,913	0,664	0,913	0,704	1000	800
7	*List_UnOrdered*	*0,791*	*0,628*	0,891	0,740	0,913	0,722	0,823	0,642	0,790	0,614	0,913	0,740	600	400
8	*Text*	*0,799*	*0,580*	0,750	0,556	0,761	0,536	0,739	0,527	0,725	0,486	0,799	0,580	200	200

Tabela 17 - Métrica de desempenho mAP modelo YOLOv5x – 5º *Dataset*.

Modelo - Yolov5x															
Classes		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,828*	*0,671*	*0,861*	0,683	0,867	0,701	0,871	0,698	0,863	0,690	0,871	0,701	800	600
1	*Button*	*0,756*	*0,550*	*0,775*	0,560	0,816	0,581	0,792	0,607	0,862	0,636	0,862	0,636	1000	1000
2	*Combobox*	*0,754*	*0,648*	*0,792*	0,651	0,845	0,684	0,819	0,656	0,844	0,703	0,845	0,703	600	1000
3	*Footer*	*0,957*	*0,817*	*0,976*	0,768	0,976	0,824	0,957	0,809	0,934	0,786	0,976	0,824	400	600
4	*Header*	*0,924*	*0,712*	*0,980*	0,769	0,940	0,749	0,957	0,737	0,959	0,733	0,980	0,769	400	400
5	*Image*	*0,961*	*0,811*	*0,976*	0,798	0,902	0,777	0,956	0,784	0,973	0,822	0,976	0,822	400	1000
6	*List_Ordered*	*0,596*	*0,483*	*0,620*	0,521	0,704	0,557	0,765	0,605	0,613	0,503	0,765	0,605	800	800
7	*List_UnOrdered*	*0,834*	*0,708*	*0,954*	0,793	0,886	0,774	0,899	0,791	0,862	0,717	0,954	0,793	400	400
8	*Text*	*0,837*	*0,639*	*0,816*	0,602	0,886	0,661	0,820	0,597	0,858	0,618	0,886	0,661	600	600

Tabela 18 - Métrica de desempenho mAP entre todos os modelos – 5º *Dataset*.

All															
Variante		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
n	*YOLOv5n*	*0,665*	*0,462*	*0,768*	*0,571*	*0,820*	*0,602*	*0,812*	*0,628*	*0,817*	*0,595*	0,820	0,628	600	800
m	*YOLOv5m*	*0,782*	*0,607*	*0,840*	*0,669*	*0,841*	*0,643*	*0,812*	*0,628*	*0,817*	*0,595*	0,841	0,669	600	400
x	*YOLOv5x*	*0,828*	*0,671*	*0,861*	*0,683*	*0,867*	*0,701*	*0,871*	*0,698*	*0,863*	*0,690*	0,871	0,701	800	600

5.3.2 Análise e desempenho modelo YOLOv8

5.3.2.1 *2º Dataset* – *Layouts* de Páginas WEB feitos à mão em folhas brancas

Os resultados no modelo YOLOv8 referentes ao 2º *dataset*, apresentaram resultados muito bons. Na Tabela 22 com todas as métricas mAP gerais dos modelos treinados é possível observar que os resultados são próximos uns dos outros. No valor de mAP50 todos os modelos de treino em todas as épocas tiveram valores superiores a 95% e para a métrica de mAP95 os valores foram sempre superiores a 80%. Tendo em conta que os valores são muitos próximos entre os modelos a escolha do modelo recairia para o modelo mais pequeno, pois a taxa de velocidade para a deteção de objetos também é um fator a ter em conta. O modelo escolhido é o YOLOv8n, 800 épocas. O valor de precisão é de 92,5%, Recall é de 95,5%. A função de perda apresenta também bons resultados, pois está constantemente a diminuir, indicando que o modelo consegue aprender progressivamente ao longo de cada época. As Tabela 19, Tabela 20 e Tabela 21 apresentam o desempenho para cada classe. Relativamente às Figura 89 e Figura 90, todas as deteções foram efetuadas corretamente com alta precisão. As Figura 87 e Figura 88 apresentam o desempenho do modelo escolhido.

Tabela 19 - Métrica de desempenho mAP modelo YOLOv8n – 2º *Dataset*.

Modelo - Yolov8n														
Classes		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,959*	*0,819*	0,973	0,824	0,975	0,840	0,989	0,851	0,982	0,822	0,989	0,851	800	800
1	*Button*	*0,995*	*0,815*	0,995	0,760	0,995	0,824	0,995	0,811	0,995	0,735	0,995	0,824	200	600
2	*Combobox*	*0,995*	*0,723*	0,978	0,755	0,978	0,793	0,978	0,798	0,995	0,729	0,995	0,798	200	800
3	*Footer*	*0,926*	*0,812*	0,954	0,758	0,982	0,821	0,967	0,790	0,967	0,783	0,982	0,821	800	600
4	*Header*	*0,995*	*0,875*	0,995	0,872	0,995	0,860	0,995	0,844	0,977	0,835	0,995	0,875	200	200
5	*Image*	*0,995*	*0,924*	0,995	0,941	0,995	0,927	0,995	0,940	0,995	0,945	0,995	0,945	200	1000
6	*List_Ordered*	*0,825*	*0,745*	0,948	0,836	0,948	0,829	0,995	0,900	0,995	0,886	0,995	0,900	800	800
7	*List_UnOrdered*	*0,947*	*0,845*	0,924	0,827	0,915	0,804	0,995	0,887	0,951	0,823	0,995	0,887	800	800
8	*Text*	*0,991*	*0,814*	0,995	0,845	0,995	0,859	0,995	0,833	0,985	0,837	0,995	0,859	400	600

Tabela 20 - Métrica de desempenho mAP modelo YOLOv8m – 2º *Dataset*.

Modelo - Yolov8m														
Classes		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,973*	*0,847*	0,962	0,828	0,985	0,858	0,965	0,850	0,966	0,847	0,985	0,858	600	600
1	*Button*	*0,995*	*0,824*	0,995	0,796	0,995	0,838	0,995	0,821	0,995	0,824	0,995	0,838	200	800
2	*Combobox*	*0,995*	*0,813*	0,978	0,771	0,995	0,824	0,964	0,837	0,995	0,834	0,995	0,837	200	800
3	*Footer*	*0,898*	*0,770*	0,912	0,794	0,912	0,745	0,939	0,807	0,926	0,799	0,939	0,807	800	200
4	*Header*	*0,995*	*0,883*	0,995	0,876	0,995	0,845	0,943	0,824	0,913	0,798	0,995	0,883	200	800
5	*Image*	*0,995*	*0,923*	0,995	0,920	0,995	0,933	0,995	0,951	0,995	0,931	0,995	0,951	200	600
6	*List_Ordered*	*0,948*	*0,843*	0,913	0,819	0,995	0,934	0,927	0,850	0,995	0,894	0,995	0,934	600	600
7	*List_UnOrdered*	*0,965*	*0,863*	0,911	0,795	0,995	0,888	0,959	0,853	0,915	0,841	0,995	0,888	600	200
8	*Text*	*0,995*	*0,857*	0,995	0,852	0,995	0,854	0,995	0,856	0,995	0,853	0,995	0,857	200	

Tabela 21 - Métrica de desempenho mAP modelo YOLOv8x – 2º *Dataset*.

Modelo - Yolov8x														
Classes		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,975*	*0,850*	*0,973*	0,851	0,984	0,869	0,980	0,856	0,971	0,857	0,984	0,869	600	600
1	*Button*	*0,995*	*0,796*	*0,995*	0,838	0,995	0,841	0,995	0,817	0,995	0,841	0,995	0,841	200	600
2	*Combobox*	*0,995*	*0,830*	*0,995*	0,805	0,995	0,871	0,995	0,770	0,944	0,770	0,995	0,871	200	600
3	*Footer*	*0,850*	*0,737*	*0,954*	0,787	0,971	0,815	0,967	0,839	0,939	0,852	0,971	0,852	600	1000
4	*Header*	*0,995*	*0,876*	*0,986*	0,845	0,977	0,867	0,952	0,843	0,986	0,881	0,995	0,881	200	1000
5	*Image*	*0,995*	*0,945*	*0,995*	0,927	0,976	0,940	0,995	0,950	0,995	0,966	0,995	0,966	200	1000
6	*List_Ordered*	*0,995*	*0,923*	*0,927*	0,882	0,995	0,896	0,995	0,909	0,948	0,861	0,995	0,923	200	200
7	*List_UnOrdered*	*0,980*	*0,897*	*0,941*	0,860	0,965	0,852	0,944	0,856	0,965	0,820	0,980	0,897	200	200
8	*Text*	*0,995*	*0,862*	*0,995*	0,865	0,995	0,868	0,995	0,861	0,995	0,864	0,995	0,868	200	600

Tabela 22 - Métrica de desempenho mAP entre todos os modelos – 2º *Dataset*.

All														
Variante		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
n	*YOLOv5n*	*0,959*	*0,819*	*0,973*	*0,824*	*0,975*	*0,840*	*0,989*	*0,851*	*0,982*	*0,822*	0,989	0,851	800	800
m	*YOLOv5m*	*0,973*	*0,847*	*0,962*	*0,828*	*0,985*	*0,858*	*0,965*	*0,850*	*0,966*	*0,847*	0,985	0,858	600	600
x	*YOLOv5x*	*0,975*	*0,850*	*0,973*	*0,851*	*0,984*	*0,869*	*0,980*	*0,856*	*0,971*	*0,857*	0,984	0,869	600	600

```
800 epochs completed in 1.868 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 6.2MB
Optimizer stripped from runs/detect/train/weights/best.pt, 6.2MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.0.14 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3007208 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  1.94it/s]
                   all         17         73      0.925      0.955      0.989      0.851
                Button         17          3      0.908          1      0.995      0.811
              ComboBox         17          7          1      0.642      0.978      0.798
                Footer         17          8      0.688          1      0.967       0.79
                Header         17         10      0.973          1      0.995      0.844
                 Image         17         12      0.966          1      0.995       0.94
          List_Ordered         17          6      0.993          1      0.995        0.9
        List_Unordered         17         11       0.95          1      0.995      0.887
                  Text         17         16      0.921          1      0.995      0.833
Speed: 0.2ms pre-process, 3.2ms inference, 0.0ms loss, 1.2ms post-process per image
Saving runs/detect/train/predictions.json...
Results saved to runs/detect/train
```

Figura 87 - Métricas do modelo YOLOv8n, Época=800, 2º *Dataset*.

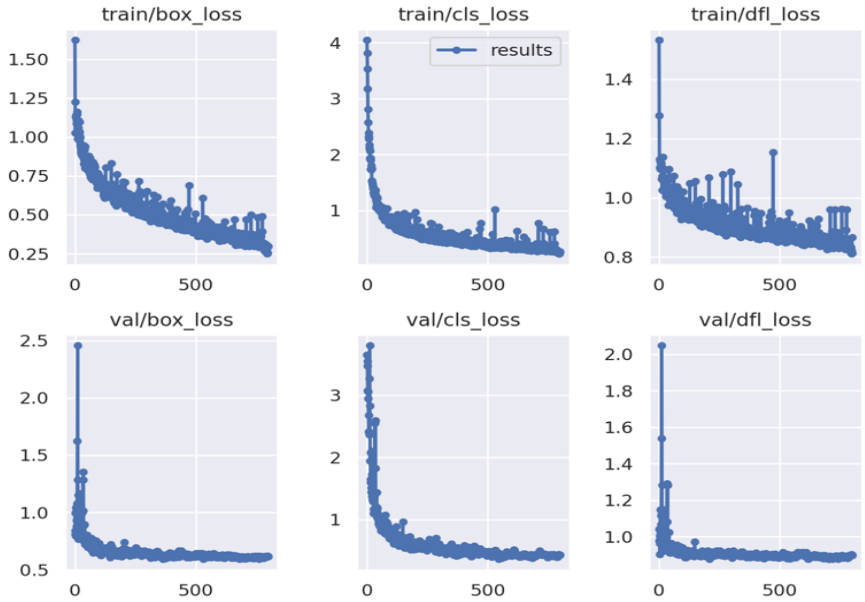
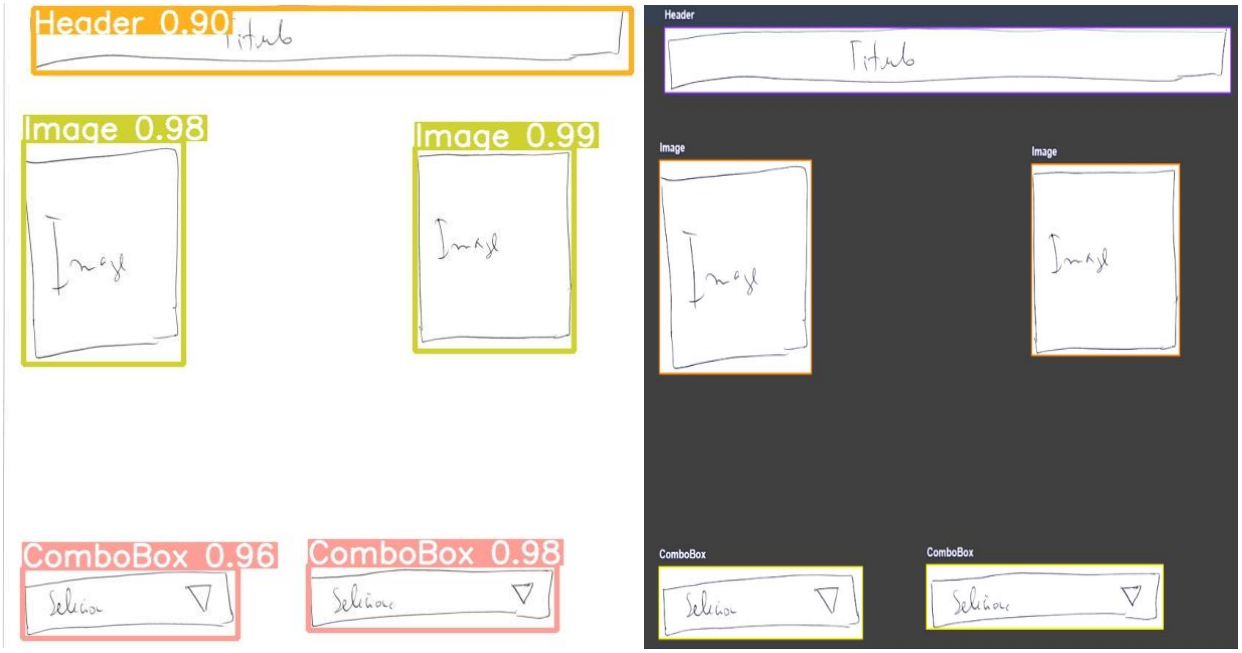
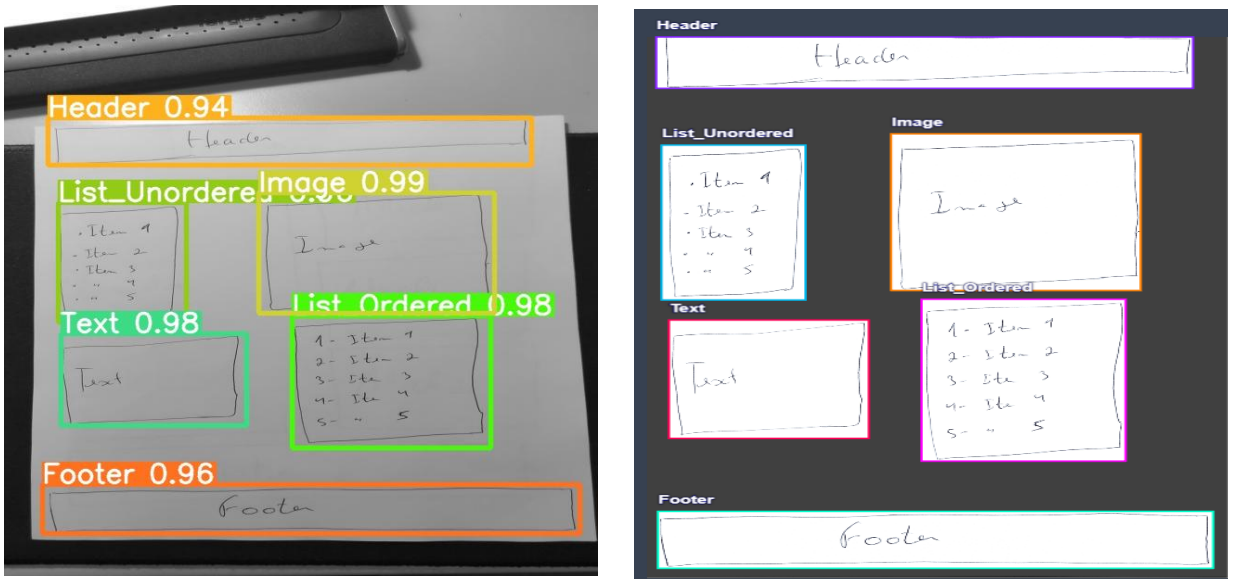


Figura 88 - Funções de perda (*Loss*) do modelo YOLOv8n, Época=800, 2º *Dataset*.



a) Deteção numa imagem de teste – 2º *Dataset* b) Imagem rotulada com as diferentes categorias.

Figura 89 – Exemplo de Imagem de teste, modelo YOLOv8n – 2º *Dataset*.



a) Deteção numa imagem de teste. b) Imagem rotulada com as diferentes categorias.

Figura 90 – Exemplo de 2ª Imagem de teste, modelo YOLOv8n – 2º *Dataset*.

5.3.2.2 *3º Dataset* – Páginas WEB desenvolvidas em HTML e CSS

No 3º *dataset* relativo às páginas WEB, as métricas de desempenho também demonstraram bons resultados, conforme podemos ver nas Tabela 23, Tabela 24, Tabela 25 e Tabela 26. Os melhores resultados ocorreram no modelo YOLOv8x época 600. O modelo obteve uma precisão de 88,7%, recall de 97,5%, mAP50 de 98,4% e um mAP95 de 79,9%. Na Figura 92, as funções de perda demonstram que o modelo não necessita de mais épocas de treino. O modelo a partir da época 500 não demonstra sinais de melhoria. Na Figura 91 apresentam-se as métricas de desempenho para o modelo.

Relativamente à deteção de objetos nas imagens, na Figura 93 a categoria "*Text*" foi aplicada incorretamente no *Título* da página, o "*Footer*" não foi detetado, assim como mais duas categorias "*Text*", as categorias "*Button*" foram detetadas corretamente. Na Figura 94 todas as categorias foram detetadas corretamente e apenas uma categoria "*Footer*" não foi detetada.

Tabela 23 - Métrica de desempenho mAP modelo YOLOv8n – 3º *Dataset*.

Modelo - Yolov8n															
Classes		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,949*	*0,756*	0,969	0,783	0,972	0,772	0,973	0,797	0,966	0,785	0,973	0,797	800	800
1	*Button*	*0,995*	*0,642*	0,995	0,681	0,995	0,660	0,995	0,743	0,995	0,782	0,995	0,782	200	1000
2	*Combobox*	*0,995*	*0,802*	0,995	0,809	0,995	0,847	0,995	0,796	0,995	0,866	0,995	0,866	200	1000
3	*Footer*	*0,995*	*0,895*	0,995	0,895	0,995	0,796	0,995	0,895	0,995	0,895	0,995	0,895	200	200
4	*Header*	*0,995*	*0,498*	0,995	0,604	0,995	0,598	0,995	0,598	0,995	0,524	0,995	0,604	200	400
5	*Image*	*0,995*	*0,995*	0,995	0,975	0,995	0,977	0,995	0,971	0,995	0,948	0,995	0,995	200	200
6	*List_Ordered*	*0,995*	*0,895*	0,995	0,895	0,995	0,895	0,995	0,796	0,995	0,796	0,995	0,895	200	200
7	*List_UnOrdered*	*0,995*	*0,895*	0,995	0,895	0,995	0,895	0,995	0,995	0,995	0,895	0,995	0,995	200	800
8	*Text*	*0,624*	*0,427*	0,786	0,510	0,809	0,507	0,818	0,585	0,76	0,572	0,818	0,585	800	800

Tabela 24 - Métrica de desempenho mAP modelo YOLOv8m – 3º *Dataset*.

Modelo - Yolov8m															
Classes		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,961*	*0,824*	0,968	0,813	0,959	0,804	0,946	0,823	0,957	0,823	0,968	0,824	400	200
1	*Button*	*0,995*	*0,766*	0,995	0,817	0,995	0,775	0,995	0,767	0,995	0,801	0,995	0,817	200	400
2	*Combobox*	*0,995*	*0,823*	0,995	0,837	0,995	0,811	0,995	0,856	0,995	0,885	0,995	0,885	200	1000
3	*Footer*	*0,995*	*0,895*	0,995	0,995	0,995	0,895	0,995	0,995	0,995	0,895	0,995	0,995	200	400
4	*Header*	*0,995*	*0,747*	0,995	0,574	0,995	0,648	0,995	0,700	0,995	0,652	0,995	0,747	200	200
5	*Image*	*0,995*	*0,995*	0,995	0,995	0,995	0,995	0,995	0,982	0,995	0,925	0,995	0,995	200	200
6	*List_Ordered*	*0,995*	*0,895*	0,995	0,895	0,995	0,995	0,995	0,995	0,995	0,995	0,995	0,995	200	600
7	*List_UnOrdered*	*0,995*	*0,995*	0,995	0,895	0,995	0,796	0,995	0,895	0,995	0,995	0,995	0,995	200	200
8	*Text*	*0,719*	*0,478*	0,752	0,491	0,704	0,515	0,603	0,397	0,687	0,434	0,752	0,515	400	600

Tabela 25 - Métrica de desempenho mAP modelo YOLOv8x – 3º *Dataset*.

Modelo - Yolov8x														
	Classes	200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	All	0,930	0,798	0,962	0,782	0,984	0,799	0,946	0,796	0,966	0,815	0,984	0,815	600	1000
1	Button	0,995	0,769	0,995	0,729	0,995	0,756	0,995	0,755	0,995	0,770	0,995	0,770	200	1000
2	Combobox	0,995	0,823	0,995	0,830	0,995	0,895	0,995	0,832	0,995	0,857	0,995	0,895	200	600
3	Footer	0,995	0,895	0,995	0,895	0,995	0,895	0,995	0,895	0,995	0,995	0,995	0,995	200	1000
4	Header	0,995	0,648	0,995	0,598	0,995	0,600	0,995	0,672	0,995	0,609	0,995	0,672	200	800
5	Image	0,995	0,973	0,995	0,982	0,995	0,959	0,995	0,932	0,995	0,971	0,995	0,982	200	400
6	List_Ordered	0,995	0,995	0,995	0,895	0,995	0,796	0,995	0,995	0,995	0,995	0,995	0,995	200	200
7	List_UnOrdered	0,995	0,995	0,995	0,796	0,995	0,895	0,995	0,895	0,995	0,895	0,995	0,995	200	200
8	Text	0,476	0,287	0,730	0,530	0,906	0,598	0,600	0,388	0,763	0,424	0,906	0,598	600	600

Tabela 26 - Métrica de desempenho mAP entre todos os modelos – 3º *Dataset*.

All														
	Variante	200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
n	YOLOv5n	0,949	0,756	0,969	0,783	0,972	0,772	0,973	0,797	0,966	0,785	0,973	0,797	800	800
m	YOLOv5m	0,961	0,824	0,968	0,813	0,959	0,804	0,946	0,823	0,957	0,823	0,968	0,824	400	200
x	YOLOv5x	0,930	0,798	0,962	0,782	0,984	0,799	0,946	0,796	0,966	0,815	0,984	0,815	600	1000

```
600 epochs completed in 1.641 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 136.7MB
Optimizer stripped from runs/detect/train/weights/best.pt, 136.7MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.0.14 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 68131272 parameters, 0 gradients, 257.4 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  2.54it/s]
                   all          3         24      0.887      0.975      0.984      0.799
                Button          3          3      0.949          1      0.995      0.756
              Combobox          3          4      0.911          1      0.995      0.895
                Footer          3          1      0.965          1      0.995      0.895
                Header          3          2      0.946          1      0.995        0.6
                 Image          3          7      0.971          1      0.995      0.959
          List ordered          3          1       0.87          1      0.995      0.796
        List unordered          3          1      0.865          1      0.995      0.895
                  Text          3          5      0.619        0.8      0.906      0.598
Speed: 0.4ms pre-process, 43.5ms inference, 0.0ms loss, 1.8ms post-process per image
Saving runs/detect/train/predictions.json...
Results saved to runs/detect/train
```

Figura 91 - Métricas do modelo YOLOv8x, Época=600, 3º *Dataset*

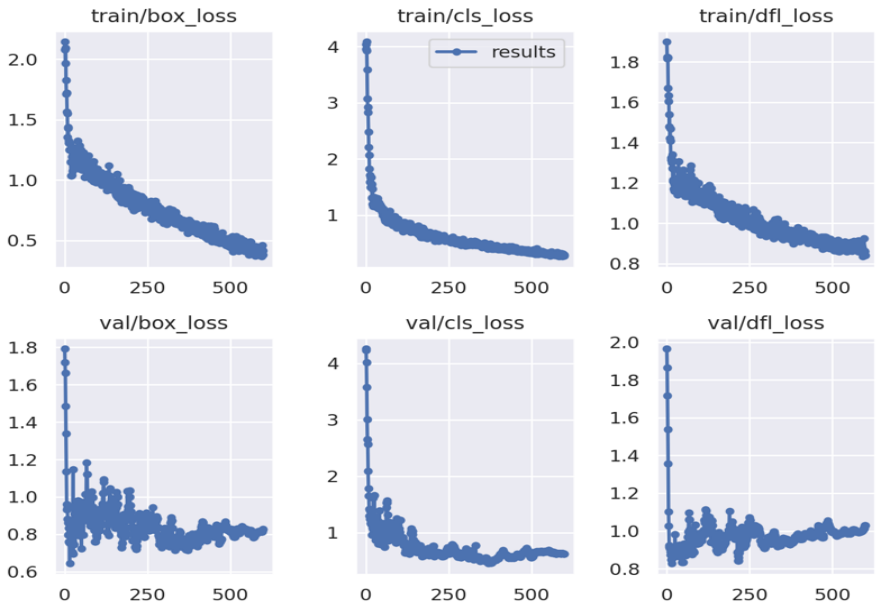
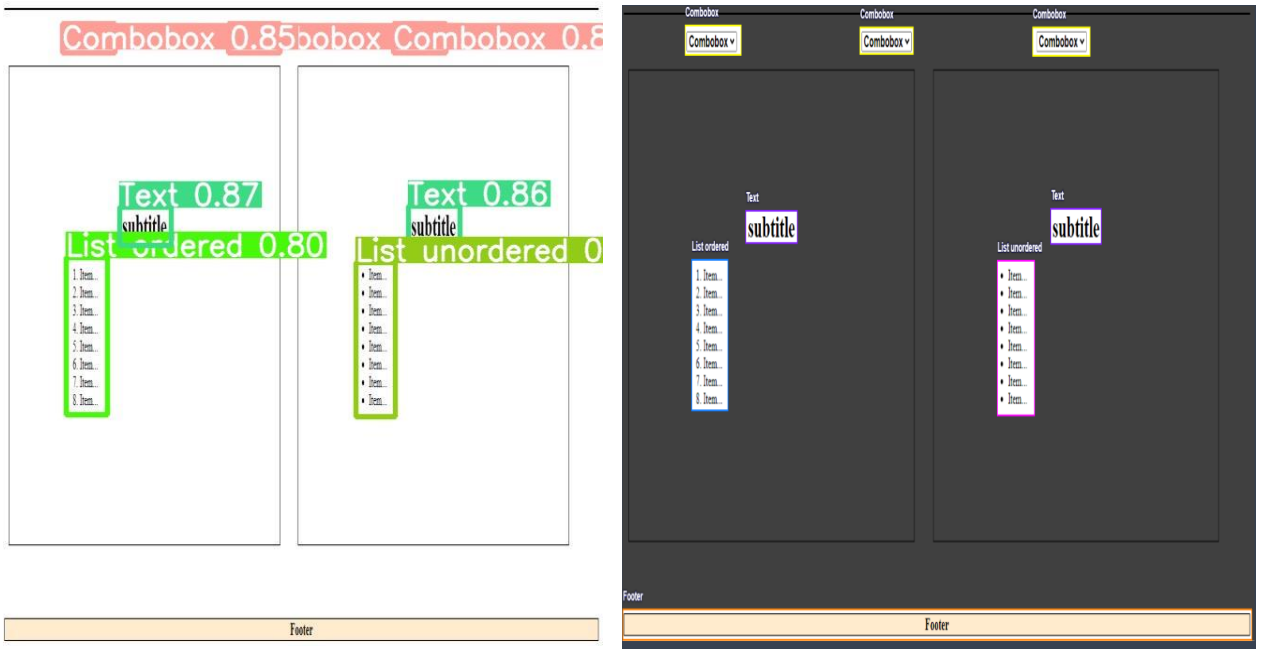


Figura 92 - Funções de perda (*Loss*) do modelo YOLOv8x, Época=800, 3º *Dataset*.

a) Deteção numa imagem de teste. b) Imagem rotulada com as diferentes categorias.

Figura 93 – Exemplo de Imagem de teste, modelo YOLOv8x – 3º *Dataset*.



a) Deteção numa imagem de teste.

b) Imagem rotulada com as diferentes categorias.

Figura 94 – Exemplo de 2ª Imagem de teste, modelo YOLOv8x – 3º *Dataset*.

5.3.2.3 *4º Dataset – Wireframes*

Em relação ao 4º *dataset* de *Wireframes* treinado no modelo YOLOv8, o melhor resultado obtido aconteceu no YOLOv8n treinado para 1000 épocas. O mAP50 obteve 61,6%, o mAP95 37,5%, a precisão foi de 80,2% e o Recall de 39,4%. Não são resultados satisfatórios (ver Tabela 27, Tabela 28, Tabela 29 e Tabela 30 e Figura 95 e Figura 96), deverá ser necessário aumentar a quantidade de imagens com mais exemplos de forma haver uma maior variedade de todas as categorias.

Relativamente às imagens detetadas e analisando as Figura 97 e Figura 98, apesar de as métricas apresentarem baixos resultados, o modelo conseguiu detetar a maioria das categorias.

Tabela 27 - Métrica de desempenho mAP modelo YOLOv8m – 4º *Dataset*.

Modelo - Yolov8n														
Classes		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,472*	*0,316*	0,53	0,360	0,552	0,371	0,518	0,348	0,616	0,375	0,616	0,375	1000	1000
1	*Button*	*0,552*	*0,334*	0,712	0,399	0,677	0,407	0,645	0,436	0,562	0,355	0,712	0,436	400	800
2	*Combobox*	*0,059*	*0,024*	0,02	0,015	0,0059	0,005	0,0474	0,024	0,399	0,128	0,399	0,128	1000	1000
3	*Image*	*0,902*	*0,749*	0,805	0,615	0,785	0,632	0,715	0,540	0,826	0,646	0,902	0,749	200	200
4	*Text*	*0,373*	*0,158*	0,675	0,413	0,74	0,438	0,665	0,391	0,675	0,371	0,740	0,438	600	600

Tabela 28 - Métrica de desempenho mAP modelo YOLOv8m – 4º *Dataset*.

Modelo - Yolov8m														
Classes		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,546*	*0,372*	0,566	0,364	0,554	0,383	0,557	0,375	0,537	0,396	0,566	0,396	400	1000
1	*Button*	*0,534*	*0,325*	0,663	0,434	0,61	0,375	0,658	0,456	0,635	0,444	0,663	0,456	400	800
2	*Combobox*	*0,163*	*0,0619*	0	0	0,0688	0,044	0,232	0,123	0,3	0,240	0,300	0,240	1000	1000
3	*Image*	*0,907*	*0,747*	0,824	0,697	0,899	0,732	0,797	0,649	0,966	0,775	0,966	0,775	400	1000
4	*Text*	*0,579*	*0,353*	0,775	0,325	0,608	0,382	0,542	0,273	0,249	0,125	0,775	0,382	400	200

Tabela 29 - Métrica de desempenho mAP modelo YOLOv8x – 4º *Dataset*.

Modelo - Yolov8x														
Classes		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,596*	*0,466*	0,576	0,388	0,602	0,375	0,525	0,398	0,564	0,409	0,602	0,466	600	200
1	*Button*	*0,455*	*0,327*	0,692	0,461	0,724	0,434	0,769	0,559	0,660	0,477	0,769	0,559	800	800
2	*Combobox*	*0,376*	*0,376*	0,0833	0,0583	0,0949	0,0569	0,000	0,000	0,152	0,106	0,376	0,376	200	200
3	*Image*	*0,858*	*0,725*	0,949	0,746	0,819	0,626	0,848	0,703	0,889	0,723	0,949	0,746	400	400
4	*Text*	*0,693*	*0,434*	0,58	0,286	0,77	0,384	0,482	0,331	0,557	0,332	0,770	0,434	600	200

Tabela 30 - Métrica de desempenho mAP entre todos os modelos – 4º *Dataset*.

All														
Variante		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
n	*YOLOv8n*	*0,472*	*0,316*	*0,530*	*0,360*	*0,552*	*0,371*	*0,518*	*0,348*	*0,616*	*0,375*	0,616	0,375	1000	1000
m	*YOLOv8m*	*0,546*	*0,372*	*0,566*	*0,364*	*0,554*	*0,383*	*0,557*	*0,375*	*0,537*	*0,396*	0,566	0,396	400	1000
x	*YOLOv8x*	*0,596*	*0,466*	*0,576*	*0,388*	*0,602*	*0,375*	*0,525*	*0,398*	*0,564*	*0,409*	0,602	0,466	600	200

```
1000 epochs completed in 1.226 hours.
Optimizer stripped from runs/detect/train2/weights/last.pt, 6.2MB
Optimizer stripped from runs/detect/train2/weights/best.pt, 6.2MB

Validating runs/detect/train2/weights/best.pt...
Ultralytics YOLOv8.0.14 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3006623 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  3.34it/s]
                   all          5         61      0.802      0.394      0.616      0.375
                Button          5         17      0.868      0.388      0.562      0.355
              ComboBox          5          3          1          0      0.399      0.128
                 Image          5         26      0.607      0.654      0.826      0.646
                  Text          5         15      0.732      0.533      0.675      0.371
Speed: 0.4ms pre-process, 4.3ms inference, 0.0ms loss, 2.3ms post-process per image
Saving runs/detect/train2/predictions.json...
Results saved to runs/detect/train2
```

Figura 95 - Métricas do modelo YOLOv8n, Época=1000, 4º *Dataset*.

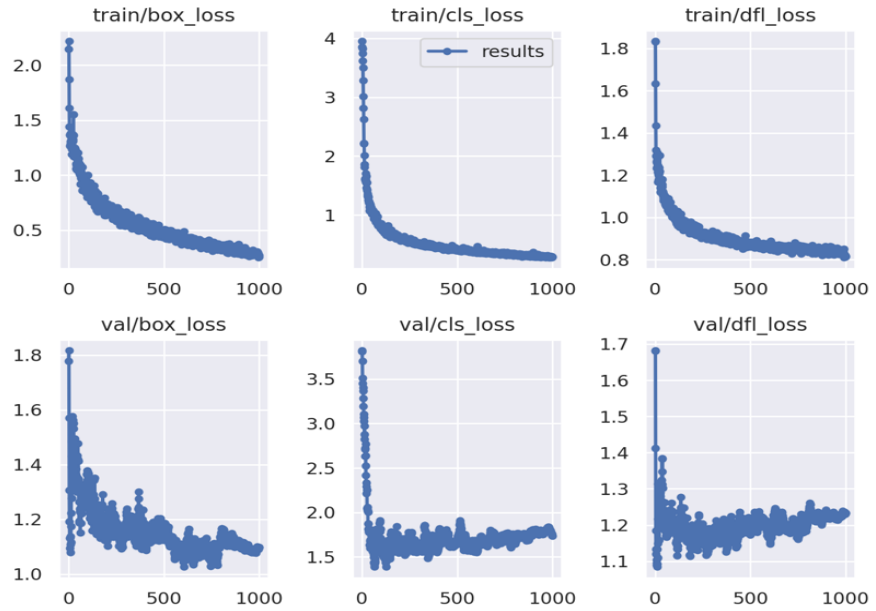
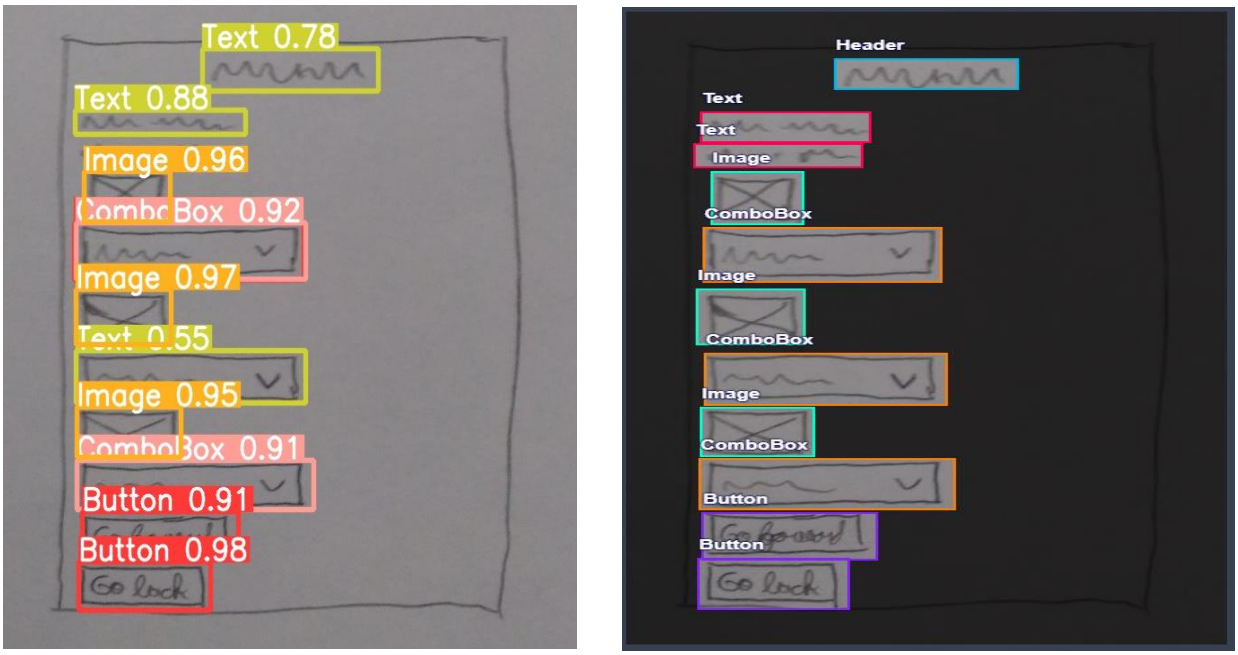


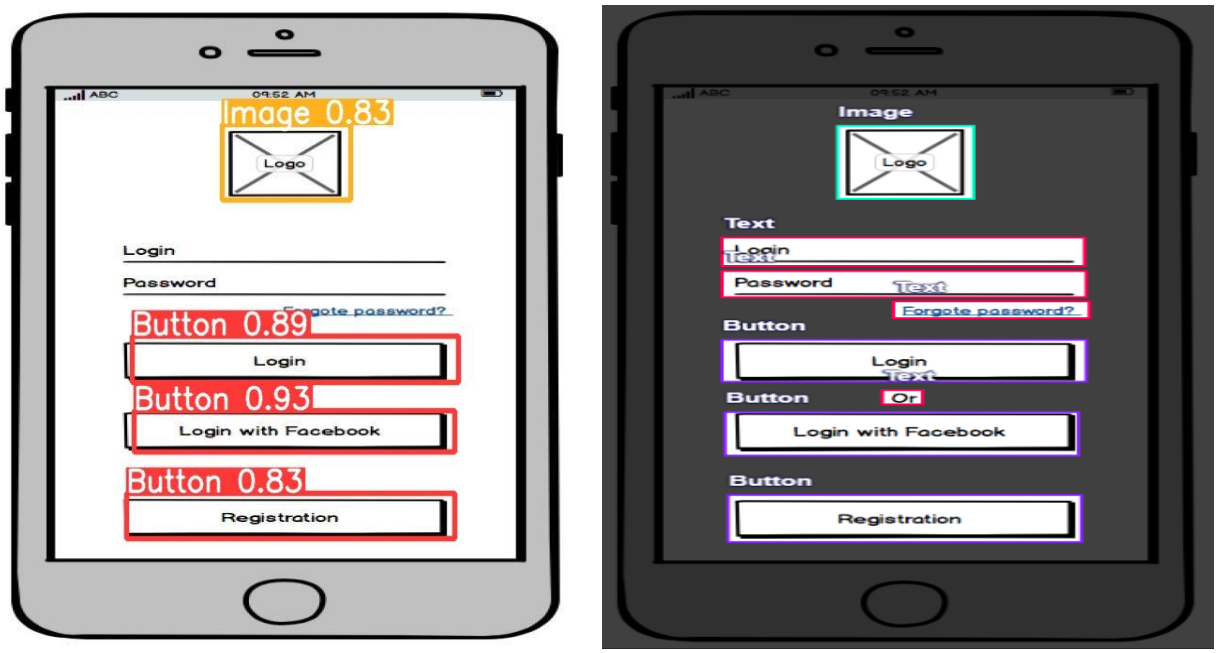
Figura 96 - Funções de perda (*Loss*) do modelo YOLOv8n, Época=1000, 4º *Dataset*.



a) Deteção numa imagem de teste.

b) Imagem rotulada com as diferentes categorias.

Figura 97 – Exemplo de Imagem de teste, modelo YOLOv8n – 4º *Dataset*.



a) Deteção numa imagem de teste. b) Imagem rotulada com as diferentes categorias.

Figura 98 – Exemplo de 2ª Imagem de teste, modelo YOLOv8n – 4º *Dataset*.

5.3.2.4 *5º Dataset* – União entre o 2º, 3º e 4º *dataset*

No 5º *dataset* para o modelo YOLOv8, os resultados foram muito bons e no geral até há quase uma homogeneidade entre modelos, como já se tinha verificado no modelo YOLOv5. Na Tabela 31 estão os 4 melhores modelos de treino, as melhores métricas foram obtidas no modelo YOLOv8x treinado para 1000 épocas, mas analisando a Figura 99 a) com as funções de perda, verifica-se que o modelo sofre de *overfitting*, apesar do gráfico *cls_loss* relativo à precisão das previsões feitas apresentar uma aprendizagem progressiva ao longo das épocas, os outros dois gráficos não apresentam melhorias. O gráfico *box_loss* que mede a precisão da previsão das coordenadas das caixas delimitadoras e o gráfico *dfl_loss* que mede a confiança nas suas previsões, estes dois gráficos não estão a ter uma aprendizagem progressiva, pelo contrário, assim, comparemos as funções de perda entre os outros 3 modelos.

Comparando Figura 99 b) e a Figura 100 a) e b) o modelo YOLOv8n treinado para 200 épocas tem melhores funções de perda, com uma aprendizagem progressiva ao longo das épocas, não sofrendo de *overfitting*.

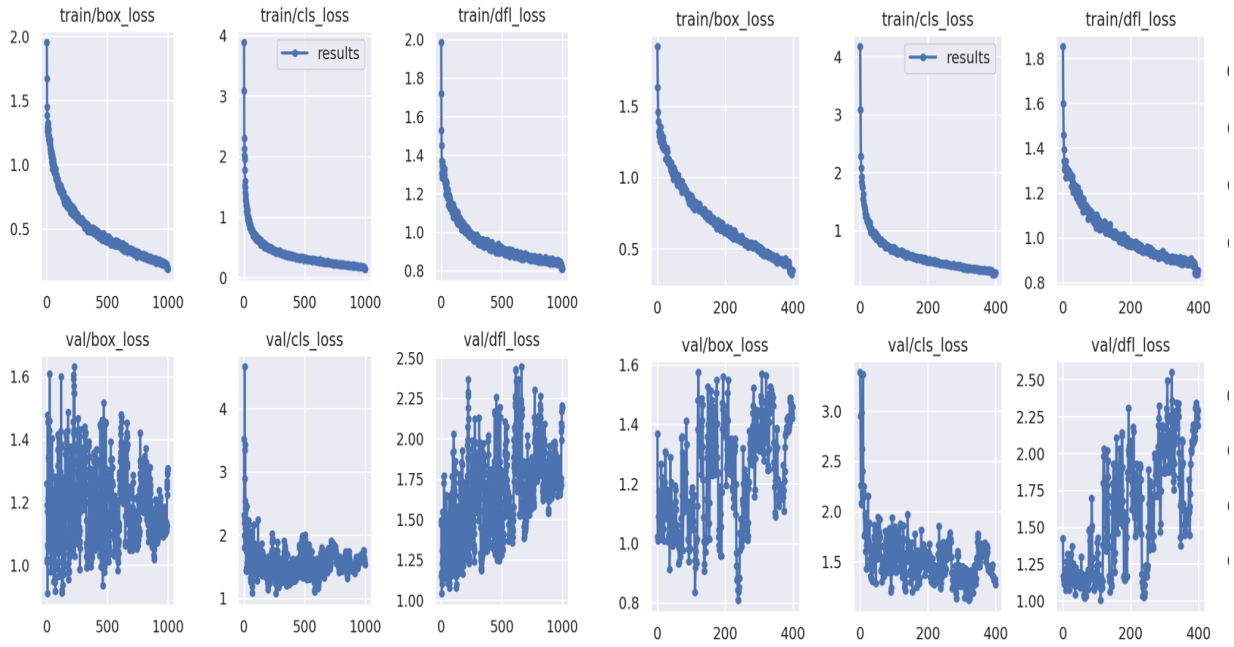
Agora comparemos a Figura 101, Figura 102 e Figura 103 do modelo YOLOv8n 200 épocas e o modelo YOLOv8x 1000 épocas. Na Figura 101, o modelo YOLOv8n apresenta melhores resultados, acerta em todas as categorias ao contrário do modelo YOLOv8x. Na Figura 102, novamente, o modelo YOLOv8n tem melhores resultados, acerta em todas as deteções, faltando apenas detetar uma categoria "*Text*". Na Figura 103 ambos os modelos acertam em todas as categorias com o modelo YOLOv8x a apresentar maiores níveis de confiança nas suas

deteções. As Tabela 32, Tabela 33 e Tabela 34 apresentam o desempenho por categoria para cada modelo, enquanto a Tabela 35 compara o desempenho dos modelos.

Posto isto, a escolha recairá para o modelo YOLOv8n treinado para 200 épocas, pois apresenta maior precisão nos resultados e sendo um modelo mais pequeno a velocidade de previsão nas deteções será maior.

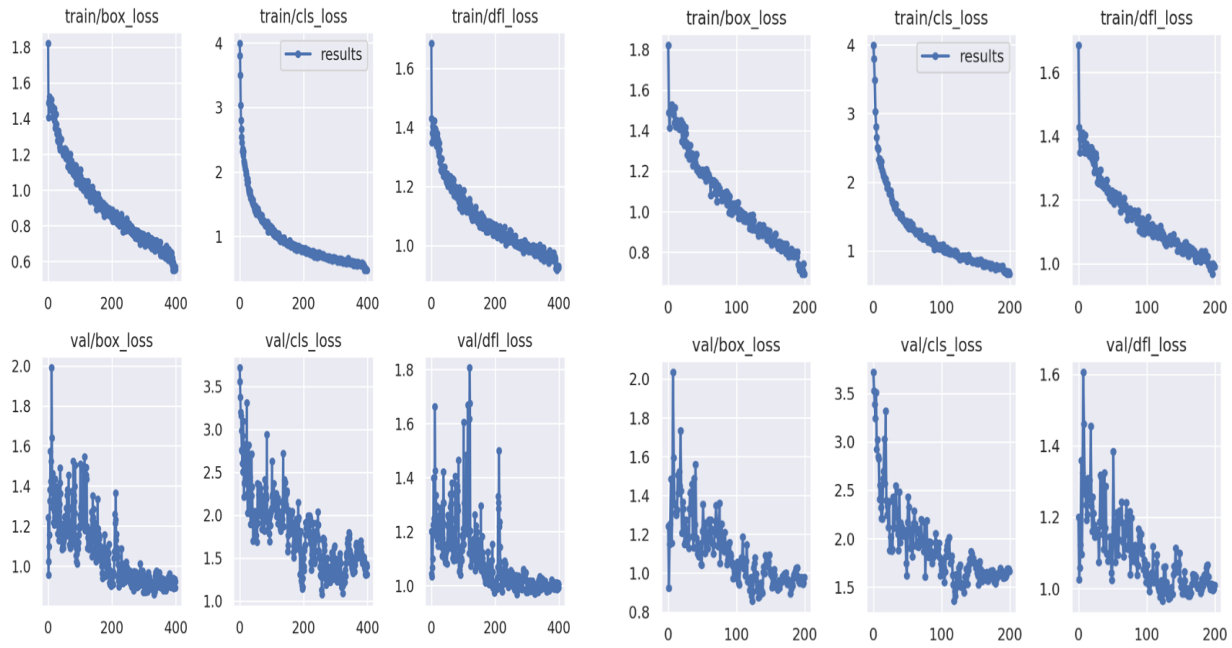
Tabela 31 - Comparação de métricas entre as melhores épocas do modelo de treino YOLOv8.

Modelo/Época	Precisão	Recall	mAP50	mAP95
YOLOv8n/200	87,1%	73,9%	84,9%	68,4%
YOLOv8n/400	84,1%	80,4%	84,2%	70,1%
YOLOv8m/400	85,4%	80,9%	88,4%	73,9%
YOLOv8x/1000	86,5%	84,8%	89,7%	75,6%



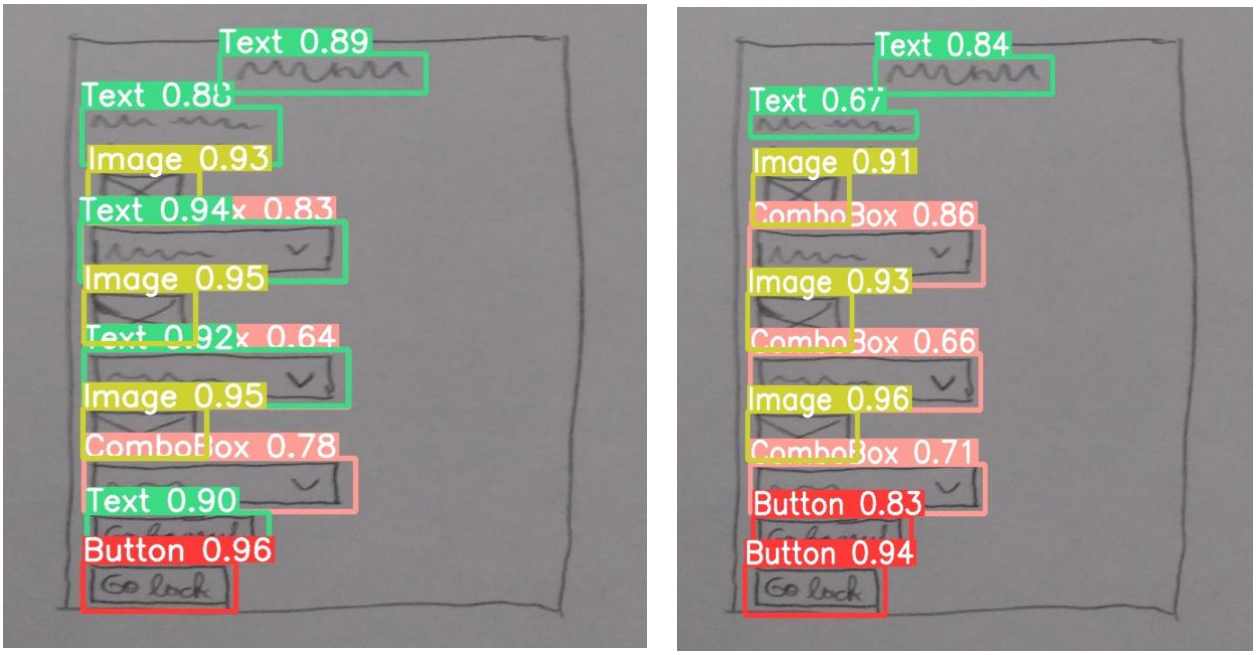
a) Funções de perda (*Loss*), YOLOv8x Época 1000. b) Funções de perda (*Loss*), YOLOv8m Época 400.

Figura 99 – Funções de Perda para os modelos YOLOv8 – 5º *Dataset*.



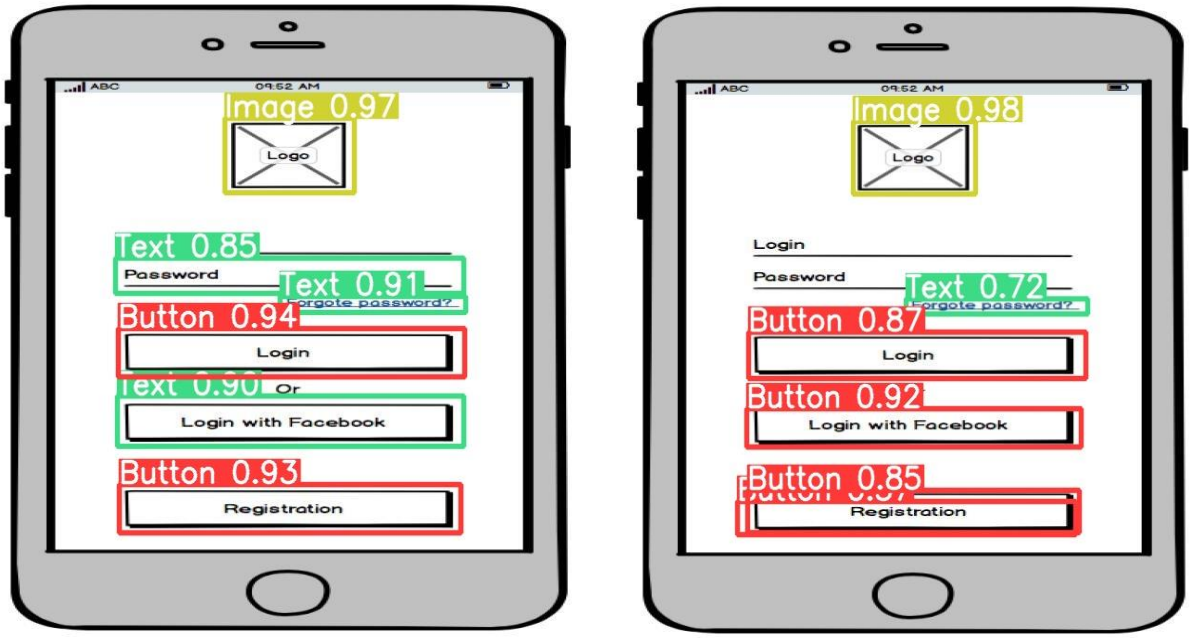
a) Funções de perda (*Loss*), YOLOv8n Época 400. b) Funções de perda (*Loss*), YOLOv8n Época 200.

Figura 100 – Funções de Perda para os modelos YOLOv8n – 5º *Dataset*.



a) Deteção numa imagem de teste, YOLOv8x. b) Deteção numa imagem de teste, YOLOv8n

Figura 101 – Deteção para os modelos YOLOv8x e YOLOv8n, exemplo n. º1 – 5º *Dataset*.



a) Deteção numa imagem de teste, YOLOv8x. b) Deteção numa imagem de teste, YOLOv8n.

Figura 102 – Deteção para os modelos YOLOv8x e YOLOv8n, exemplo n. º2 – 5º *Dataset*.

a) Deteção numa imagem de teste, YOLOv8x. b) Deteção numa imagem de teste, YOLOv8n.

Figura 103 – Deteção para os modelos YOLOv8x e YOLOv8n, exemplo n. º3 – 5º *Dataset*.

Tabela 32 - Métrica de desempenho mAP modelo YOLOv8n – 5º *Dataset*.

Modelo - Yolov8n														
Classes		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,849*	*0,684*	0,842	0,701	0,828	0,661	0,827	0,685	0,845	0,696	0,849	0,701	200	400
1	*Button*	*0,718*	*0,518*	0,681	0,522	0,666	0,447	0,71	0,543	0,745	0,565	0,745	0,565	1000	1000
2	*Combobox*	*0,832*	*0,672*	0,8	0,640	0,851	0,688	0,869	0,712	0,805	0,689	0,869	0,712	800	800
3	*Footer*	*0,958*	*0,819*	0,967	0,850	0,967	0,814	0,962	0,839	0,951	0,808	0,967	0,850	400	400
4	*Header*	*0,977*	*0,756*	0,953	0,759	0,959	0,737	0,877	0,694	0,877	0,712	0,977	0,759	200	400
5	*Image*	*0,836*	*0,715*	0,942	0,812	0,82	0,702	0,864	0,768	0,855	0,729	0,942	0,812	400	400
6	*List_Ordered*	*0,818*	*0,651*	0,738	0,656	0,758	0,615	0,689	0,591	0,943	0,814	0,943	0,814	1000	1000
7	*List_UnOrdered*	*0,898*	*0,775*	0,902	0,789	0,87	0,722	0,908	0,774	0,851	0,707	0,908	0,789	800	400
8	*Text*	*0,756*	*0,568*	0,749	0,576	0,733	0,567	0,737	0,562	0,73	0,547	0,756	0,576	200	400

Tabela 33 - Métrica de desempenho mAP modelo YOLOv8m – 5º *Dataset*.

Modelo - Yolov8m														
Classes		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,852*	*0,706*	0,884	0,739	0,868	0,730	0,871	0,724	0,877	0,726	0,884	0,739	400	400
1	*Button*	*0,749*	*0,549*	0,772	0,565	0,681	0,519	0,816	0,611	0,760	0,557	0,816	0,611	800	800
2	*Combobox*	*0,918*	*0,730*	0,889	0,767	0,912	0,780	0,918	0,782	0,844	0,727	0,918	0,782	200	800
3	*Footer*	*0,952*	*0,833*	0,976	0,868	0,972	0,845	0,943	0,793	0,968	0,828	0,976	0,868	400	400
4	*Header*	*0,914*	*0,731*	0,944	0,766	0,928	0,761	0,951	0,771	0,935	0,756	0,951	0,771	800	800
5	*Image*	*0,872*	*0,761*	0,891	0,779	0,904	0,779	0,859	0,757	0,880	0,767	0,904	0,779	600	400
6	*List_Ordered*	*0,778*	*0,682*	0,868	0,714	0,831	0,734	0,780	0,658	0,863	0,731	0,868	0,734	400	600
7	*List_UnOrdered*	*0,907*	*0,783*	0,923	0,822	0,931	0,799	0,906	0,799	0,977	0,837	0,977	0,837	1000	1000
8	*Text*	*0,725*	*0,580*	0,808	0,629	0,786	0,624	0,796	0,621	0,790	0,607	0,808	0,629	400	400

Tabela 34 - Métrica de desempenho mAP modelo YOLOv8x – 5º *Dataset*.

Modelo - Yolov8x														
Classes		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
0	*All*	*0,897*	*0,745*	*0,888*	0,748	0,891	0,754	0,839	0,703	0,897	0,756	0,897	0,756	200	1000
1	*Button*	*0,867*	*0,673*	*0,797*	0,626	0,796	0,590	0,723	0,540	0,814	0,639	0,867	0,673	200	200
2	*Combobox*	*0,900*	*0,770*	*0,937*	0,800	0,908	0,792	0,855	0,746	0,930	0,795	0,937	0,800	400	400
3	*Footer*	*0,934*	*0,770*	*0,948*	0,822	0,948	0,834	0,929	0,808	0,943	0,834	0,948	0,834	400	600
4	*Header*	*0,952*	*0,753*	*0,961*	0,774	0,949	0,767	0,929	0,772	0,964	0,779	0,964	0,779	1000	1000
5	*Image*	*0,880*	*0,779*	*0,840*	0,734	0,933	0,809	0,923	0,814	0,894	0,788	0,933	0,814	600	800
6	*List_Ordered*	*0,868*	*0,753*	*0,850*	0,758	0,863	0,779	0,708	0,581	0,891	0,773	0,891	0,779	1000	600
7	*List_UnOrdered*	*0,946*	*0,840*	*0,947*	0,837	0,935	0,840	0,884	0,767	0,931	0,806	0,947	0,840	400	200
8	*Text*	*0,826*	*0,619*	*0,820*	0,636	0,792	0,619	0,762	0,598	0,808	0,638	0,826	0,638	200	1000

Tabela 35 - Métrica de desempenho mAP entre todos os modelos – 5º *Dataset*.

All														
Variante		200		400		600		800		1000		Best Results		Epochs	
		MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95	MAP50	MAP 50:95
n	*YOLOv8n*	*0,849*	*0,684*	*0,842*	*0,701*	*0,828*	*0,661*	*0,827*	*0,685*	*0,845*	*0,696*	0,849	0,701	200	400
m	*YOLOv8m*	*0,852*	*0,706*	*0,884*	*0,739*	*0,868*	*0,730*	*0,871*	*0,724*	*0,877*	*0,726*	0,884	0,739	400	400
x	*YOLOv8x*	*0,897*	*0,745*	*0,888*	*0,748*	*0,891*	*0,754*	*0,839*	*0,703*	*0,897*	*0,756*	0,897	0,756	200	1000

5.3.3 Análise e desempenho modelo Detectron2

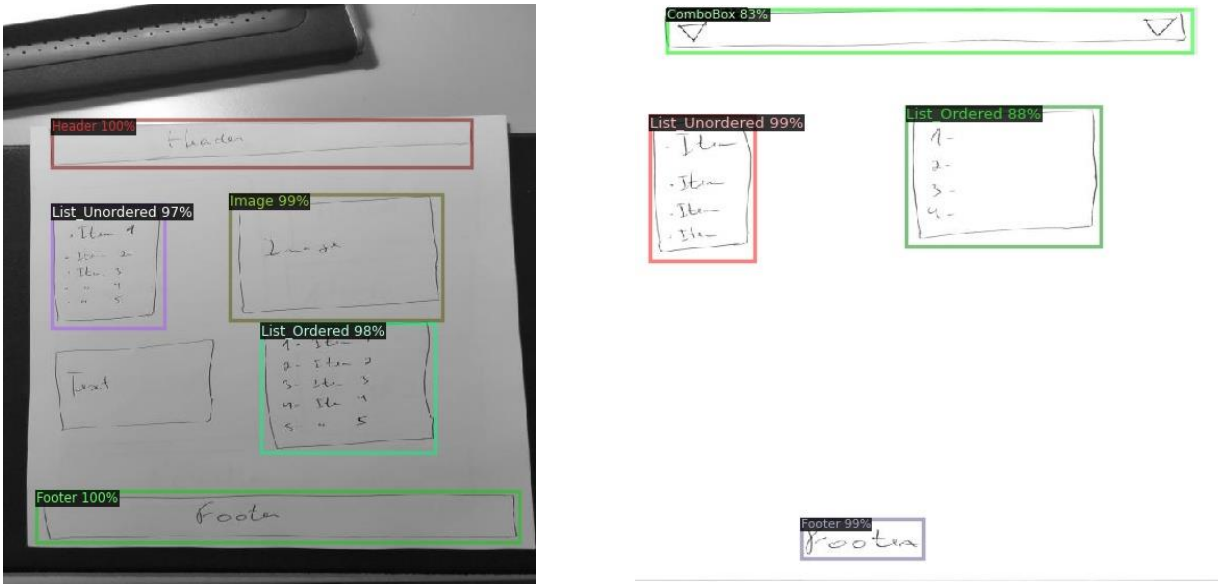
Neste capítulo serão analisadas as experiências com o modelo de deteção de objetos Detectron2.

5.3.3.1 *2º Dataset – Layouts* de Páginas WEB feitos à mão em folhas brancas

Na primeira análise ao 2º *dataset* o modelo Detectron2 mostra resultados positivos. Os melhores resultados foram obtidos no modelo treinado para 2000 épocas com uma precisão de 64.7% e um mAP de 73.6%. Analisando as Figura 104 b) e a) o modelo acertou em todas as suas deteções e apenas não detetou uma categoria. A Tabela 36 apresenta o desempenho para as diferentes categorias.

Tabela 36 - Métrica de desempenho mAP modelo Detectron2 – 2º *Dataset*.

Modelo - Detectron2												
	Classes	400		600		800		1000		2000		Best Results	Epochs
		AP	AP50	AP	AP50	AP	AP50	AP	AP50	AP	AP50	MAP50	MAP50
0	*All*	0,298		0,388		0,387		0,367		0,647		0,736	2000
1	*Button*	0,71		0,633		0,744		0,611		0,8		0,800	2000
2	*Combobox*	0		0,329		0,118		0,215		0,269		0,329	600
3	*Footer*	0,74		0,667		0,558		0,658		0,945		0,945	2000
4	*Header*	0,123	*0,403*	0,589	*0,525*	0,535	*0,508*	0,454	*0,367*	0,827	*0,736*	0,827	2000
5	*Image*	0,668		0,749		0,801		0,83		0,916		0,916	2000
6	*List_Ordered*	0		0,058		0		0		0,532		0,532	2000
7	*List_UnOrdered*	0,143		0,079		0,341		0,166		0,884		0,884	2000
8	*Text*	0		0		0		0		0		0,000	0



a) Deteção em imagem de teste, época 2000. b) Deteção em imagem de teste, época 2000.

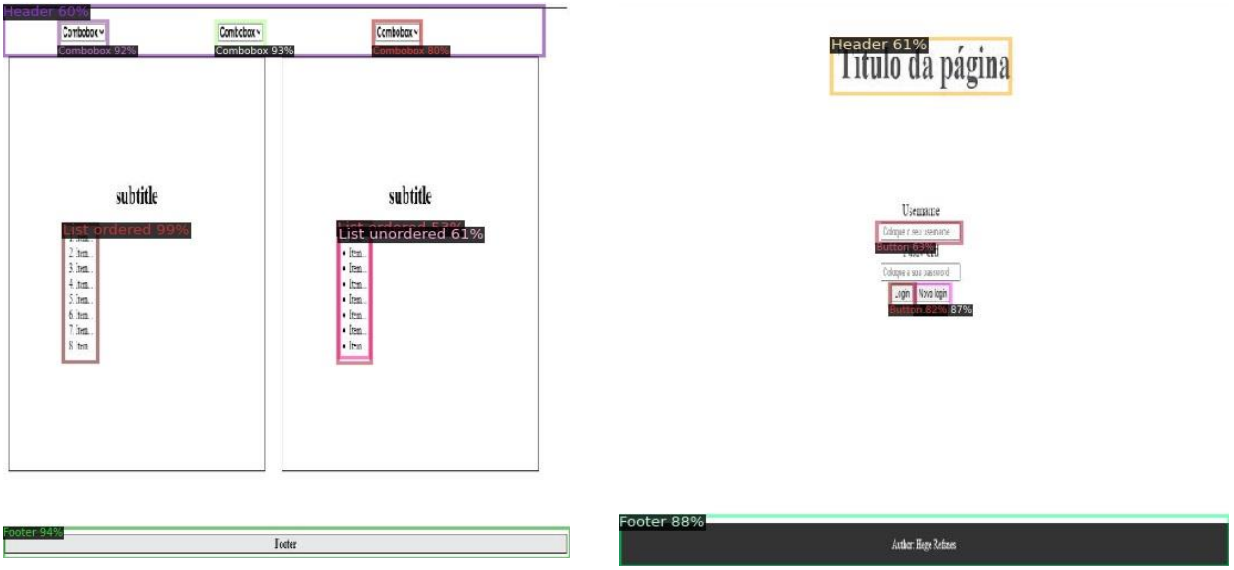
Figura 104 – Deteção para o modelo Detectron2 – 2º *Dataset*.

5.3.3.2 *3º Dataset* – Páginas WEB desenvolvidas em HTML e CSS

No 3º *dataset* é analisado o modelo de treino das páginas WEB desenvolvidas em HTML e CSS. Analisando a Tabela 37 os resultados não são satisfatórios, há categorias com baixa precisão e outras com precisão zero. Apesar das métricas não serem razoáveis, ao analisar as Figura 105 a) e b), o modelo conseguiu detetar com relativa precisão as categorias presentes nas imagens. Na Figura 105 imagem a) o modelo detetou todas as categorias corretamente, mas detetou mais duas categorias que não estão presentes na foto, a categoria "*Header*" e a categoria "*List_Ordered*" que só existe numa imagem e não nas duas como o modelo deteta. Na Figura 105 imagem b) acerta na categoria *"Header"*, na categoria *"Footer"* e em duas categorias *"button"* e não acerta em uma categoria *"button"*. Também não conseguiu detetar a categoria *"Text"* presente na imagem.

Tabela 37 - Métrica de desempenho mAP modelo Detectron2 – 3º *Dataset*.

Modelo - Detectron2												
Classes		200		400		600		800		1000		Best Results	Epochs
		AP	AP50	AP	AP50	AP	AP50	AP	AP50	AP	AP50	MAP50	
0	*All*	0,085		0,267		0,455		0,491		0,533		0,678	1000
1	*Button*	0,067		0		0,398		0,7		0,855		0,855	1000
2	*Combobox*	0,381		0,614		0,468		0,539		0,577		0,614	400
3	*Footer*	0,151		0,454		0,825		0,8		0,8		0,825	600
4	*Header*	0	*0,203*	0	*0,322*	0	*0,599*	0	*0,678*	0	*0,678*	0,000	0
5	*Image*	0		0		0		0		0		0,000	0
6	*List_Ordered*	0		0,8		0,8		0,6		0,8		0,800	1000
7	*List_UnOrdered*	0		0		0,7		0,8		0,7		0,800	800
8	*Text*	0		0		0		0		0		0,000	0



a) Deteção em imagem de teste, época 1000. b) Deteção em imagem de teste, época 1000.

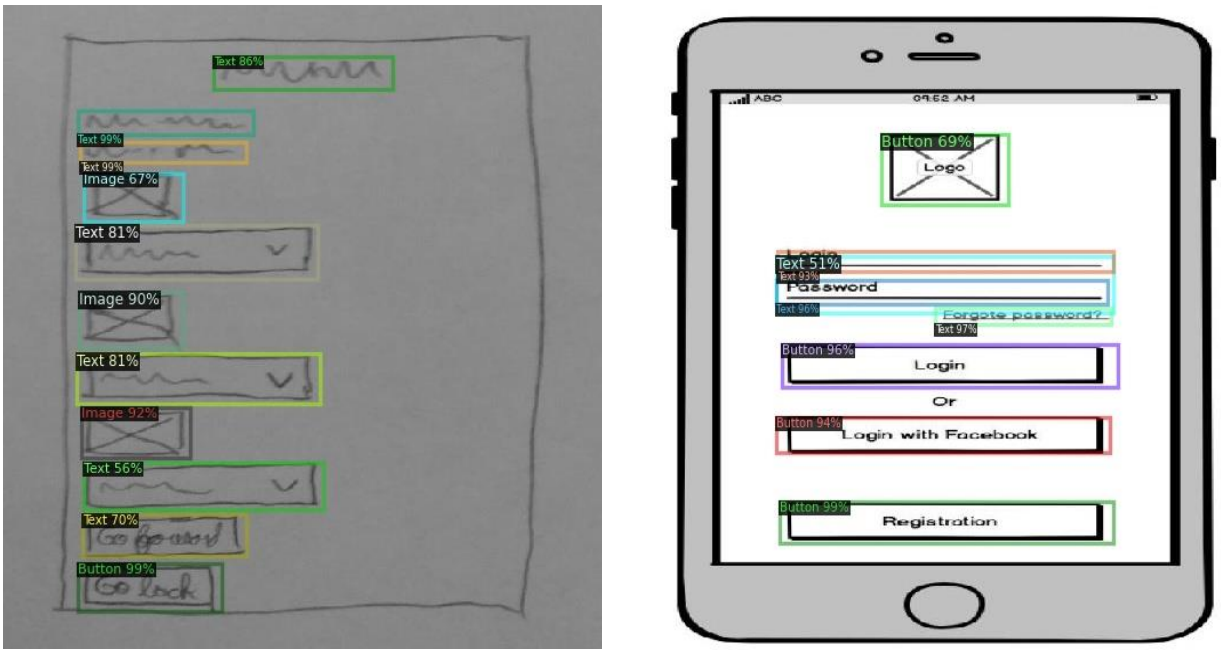
Figura 105 – Deteção para o modelo Detectron2 – 3º *Dataset*.

5.3.3.3 *4º Dataset* – *Wireframes*

Como o 3º *dataset*, as métricas de *performance* do 4º *dataset* apresentam valores de precisão baixos, a melhor pontuação é de 43,7%. Na Figura 106 b) quase todas as deteções foram certeiras, apenas uma deteção está incorreta, está como categoria "button" e deveria ser "Image". Na imagem da Figura 106 a), há muitas categorias detetadas incorretamente, como o caso da categoria "*Text*" que foi detetada onde deveria ser a categoria "*ComboBox*". A Tabela 38 apresenta o desempenho para cada categoria.

Tabela 38 - Métrica de desempenho mAP modelo Detectron2 – 4º *Dataset*.

Modelo - Detectron2												
Classes		200		400		600		800		1000		Best Results	Epochs
		AP	AP50	AP	AP50	AP	AP50	AP	AP50	AP	AP50	MAP50	MAP50
0	*All*	*0,22*		0,272		0,228		0,293		0,304		0,437	800
1	*Button*	*0,359*		0,364		0,378		0,448		0,525		0,525	800
2	*Combobox*	*0*	*0,337*	0	0,358	0	0,321	0	0,437	0	0,304	0,000	0
3	*Image*	*0,404*		0,598		0,413		0,641		0,563		0,641	800
4	*Text*	*0,337*		0,4		0,352		0,372		0,435		0,435	1000



a) Deteção em imagem de teste, época 800.

b) Deteção em imagem de teste, época 800.

Figura 106 – Deteção para o modelo Detectron2 – 4º *Dataset*.

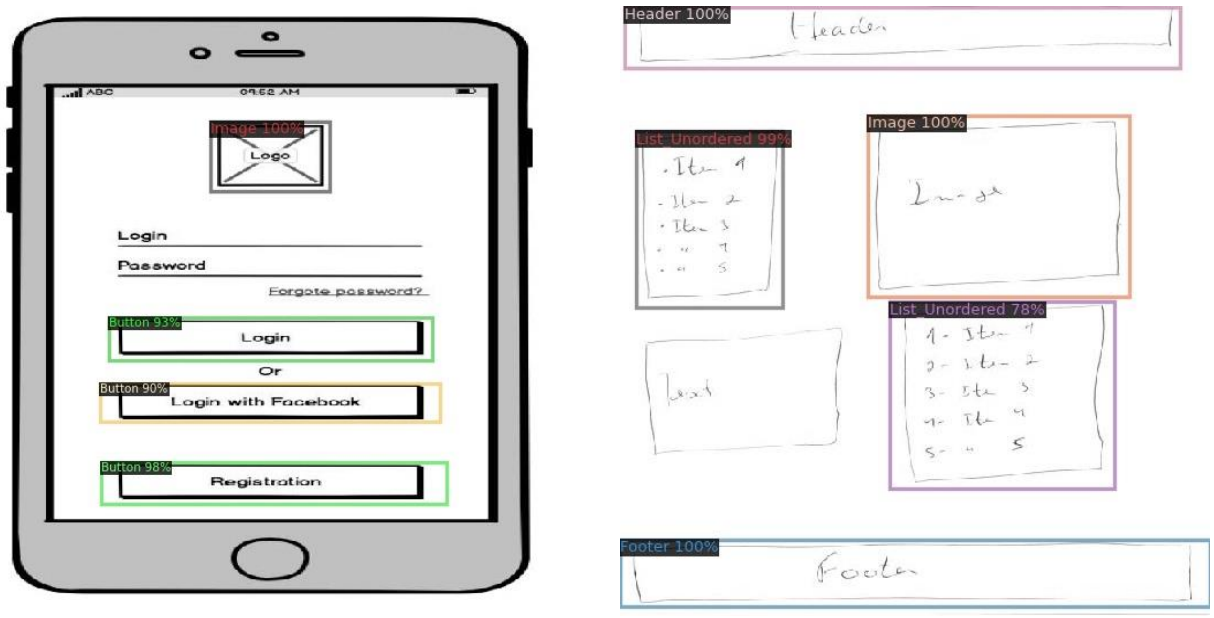
5.3.3.4 *5º Dataset* – União entre o 2º, 3º e 4º *dataset*

No quinto e último *dataset* as métricas de *performance* conseguem obter melhores resultados comparativamente aos *datasets* anteriores. O melhor resultado é obtido no modelo treinado para 4000 épocas uma precisão de 52,4% e um mAP50 de

62,5%. As deteções nas Figura 107 a) e b) foram todas certeiras, apesar de o modelo não ter conseguido detetar três categorias. A Tabela 39 apresenta o desempenho para cada categoria.

Tabela 39 - Métrica de desempenho mAP modelo Detectron2 – 5º *Dataset*.

	Modelo - Detectron2 - Uniao de Projetos															
	Classes	400		600		800		1000		2000		4000		5000		Best Results	Epochs
		AP	AP50	AP	AP50	AP	AP50	AP	AP50	AP	AP50	AP	AP50	AP	AP50	MAP50	MAP50
0	*All*	0,102		0,202		0,275		0,326		0,431		*0,54*		*0,524*		0,666	4000
1	*Button*	0,117		0		0,138		0,164		0,346		*0,45*		*0,420*		0,450	4000
2	*Combobox*	0		0,219		0,225		0,146		0,247		*0,489*		*0,335*		0,489	4000
3	*Footer*	0		0,397		0,464		0,522		0,607		*0,817*		*0,843*		0,843	5000
4	*Header*	0	*0,133*	0,141	*0,294*	0,436	*0,364*	0,539	*0,476*	0,568	*0,545*	*0,671*	*0,666*	*0,642*	*0,625*	0,671	4000
5	*Image*	0,701		0,444		0,69		0,688		0,683		*0,722*		*0,759*		0,759	5000
6	*List_Ordered*	0		0		0		0		0,187		*0,365*		*0,365*		0,365	4000
7	*List_UnOrdered*	0		0,42		0,273		0,548		0,848		*0,815*		*0,831*		0,848	5000
8	*Text*	0		0		0		0		0		*0*		*0,000*		0,000	0



a) Deteção em imagem de teste, época 4000. b) Deteção numa imagem de teste, época 4000.

Figura 107 – Deteção para o modelo Detectron2 – 5º *Dataset*.

5.4 Comparação entre os modelos

Neste subcapítulo será feita a comparação entre os modelos treinados utilizando os melhores modelos de treino de cada detetor de objetos.

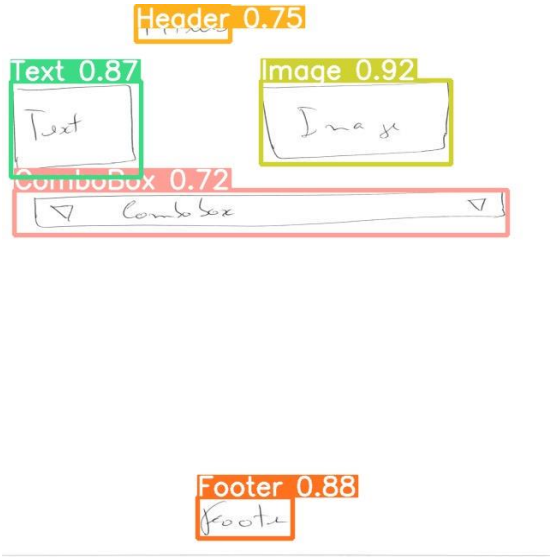
5.4.1 *2º Dataset – Layouts* de Páginas WEB feitos à mão em folhas brancas

A primeira comparação é feita ao *2ºdataset* com os *layouts* de páginas Web. Comparando os modelos através da Tabela 40 com as métricas de *performance*, todos

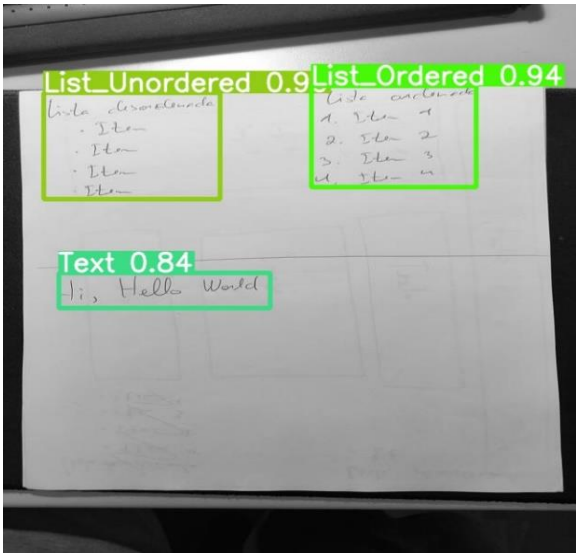
os modelos apresentam resultados muito positivos, principalmente os modelos YOLO com valores superiores a 85%. Observando as Figura 108, Figura 109 e Figura 110, todos os modelos acertaram nas suas previsões e com valores de confiança elevados, apenas o modelo Detectron2 na Figura 110 não detetou a categoria "*Text*". Com resultados próximos entre os modelos a escolha para a deteção deste tipo de *dataset* iria para o modelo YOLOv8n, pois é o modelo que apresenta melhores resultados.

Tabela 40 - Comparação entre os melhores modelos – 2º *Dataset*.

Modelo	Época	Precisão	mAP50
YOLOv5m	1000	94.6%	98.1%
YOLOv8n	800	95,5%	98.9%
Detectron2	2000	64.7%	73.6%

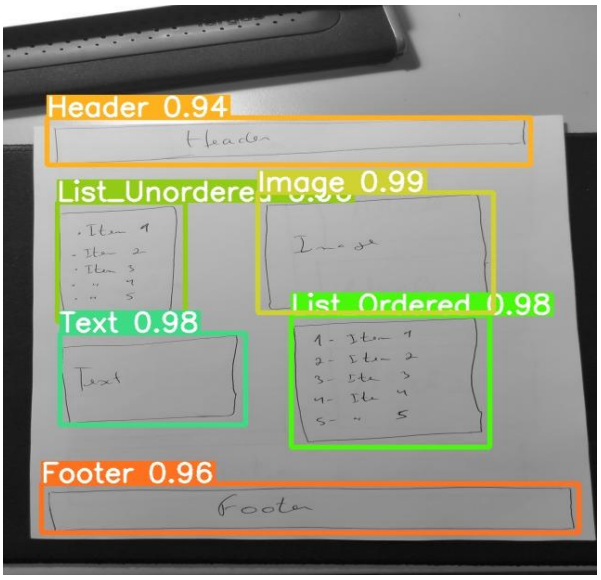


a) Deteção em imagem de teste, YOLOv5m

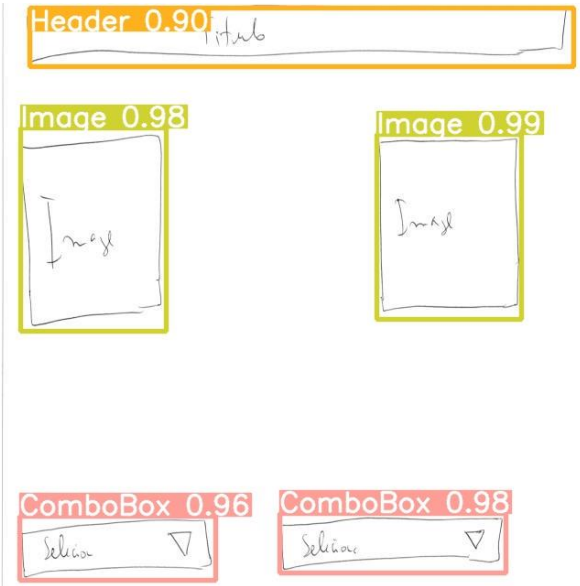


b) Deteção em imagem de teste, YOLOv5m.

Figura 108 – Deteção para o modelo YOLOv5m – 2º *Dataset*.

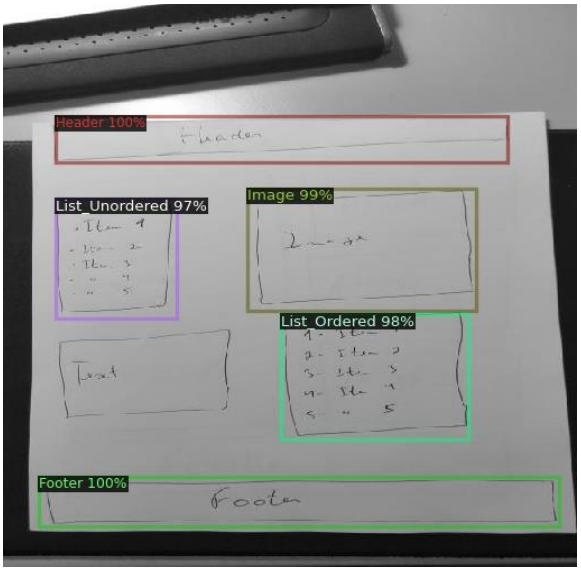


a) Deteção em imagem de teste, YOLOv8n.

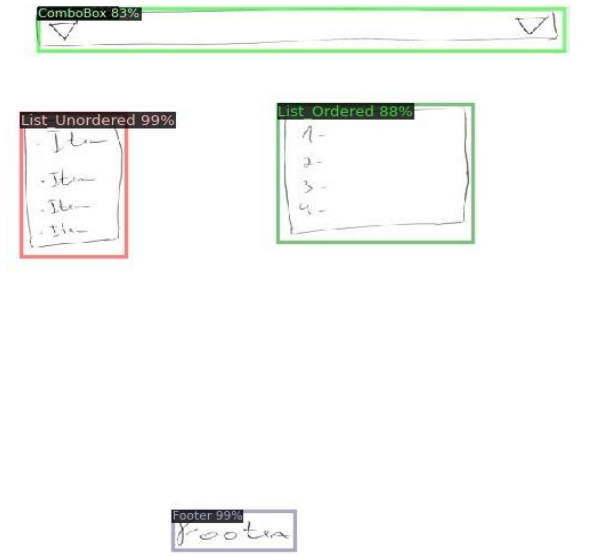


b) Deteção em imagem de teste, YOLOv8n.

Figura 109 – Deteção para o modelo YOLOv5n – 2º *Dataset*.



a) Deteção em imagem de teste, Detectron2.



b) Deteção em imagem de teste, Detectron2.

Figura 110 – Deteção para o modelo Detectron2 – 2º *Dataset*.

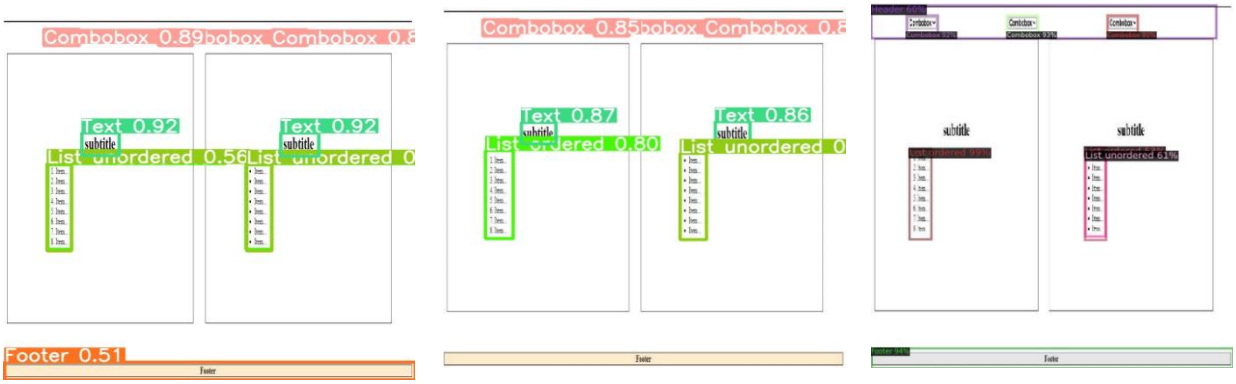
5.4.2 *3º Dataset* – Páginas WEB desenvolvidas em HTML e CSS

Comparando os três modelos aquele que conseguiu atingir resultados mais altos foi o modelo YOLOv8x treinado para 600 épocas com 88,7% de precisão e um mAP50 de 98,4%. Comparando as imagens de facto o modelo mais certeiro é o modelo YOLOv8x, errou apenas uma categoria na Figura 111, detetou uma categoria "*Text*" referente à tag "Título de página" quando devia ter detetado a

categoria "*Header*" e não detetou 3 categorias. O YOLOv5x errou ao detetar 3 categorias e não detetou uma categoria. O Detectrou2 errou ao detetar 4 categorias e não detetou 3 categorias, confirmando que o modelo YOLOv8x é o melhor modelo para este *dataset*. Na Figura 112 houve também categorias mal detetadas. A Tabela 41 apresenta o desempenho para cada categoria.

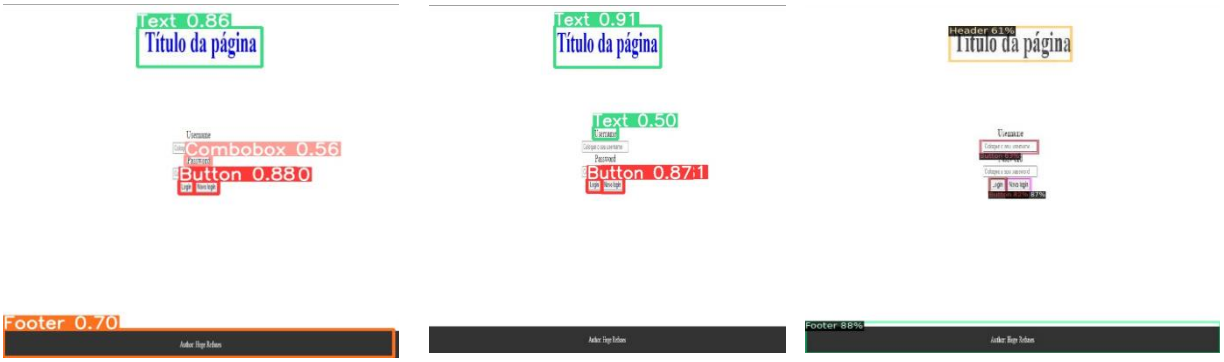
Tabela 41 - Comparação entre os melhores modelos – 3º *Dataset*.

Modelo	Época	Precisão	mAP50
YOLOv5x	600	73.4%	96.5%
YOLOv8x	600	88.7%	98.4%
Detectron2	1000	53.3%	67.8%



a) Imagem de teste, YOLOv5x. b) Imagem de teste, YOLOv8x. c) Imagem de teste, Detectron2.

Figura 111 – Deteção para os modelos: YOLOv5x, YOLOv8x e Detectron2 (imagem n. º1) – 3º *Dataset*.



a) Imagem de teste, YOLOv5x. b) Imagem de teste, YOLOv8x. c) Imagem de teste, Detectron2.

Figura 112 – Deteção para os modelos: YOLOv5x, YOLOv8x e Detectron2 (imagem n. º2) – 3º *Dataset*.

5.4.3 *4º Dataset – Wireframes*

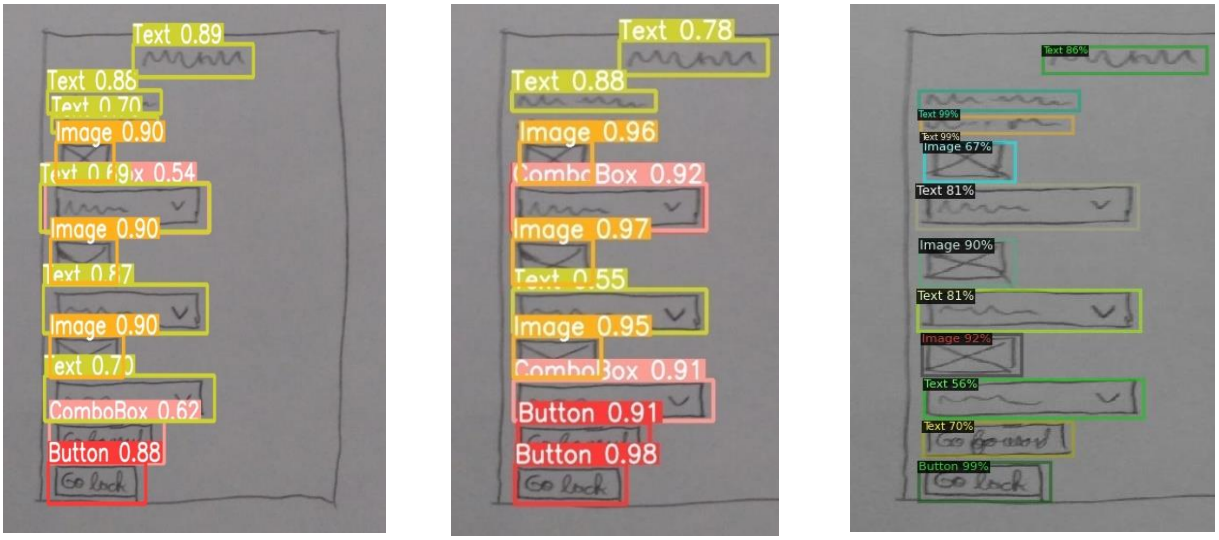
Na comparação entre modelos no 4º *dataset* referente às *Wireframes*, o modelo YOLO da versão 8 apresenta melhores resultados. Atingiu uma precisão de 80,2%

e um mAP50 de 61.6%. Se compararmos as imagens na Figura 113 o modelo YOLOv8n é o mais preciso, errou duas categorias e não detetou duas, enquanto o modelo YOLOv5x errou três categorias e não detetou uma. O detectron2 apresenta os piores resultados errou 6 categorias. A Tabela 42 apresenta a comparação de desempenho entre os três modelos. A Figura 114 apresenta outra imagem de teste onde algumas categorias foram inadequadamente classificadas.

O modelo YOLOv8n é o melhor modelo de treino para este *dataset*.

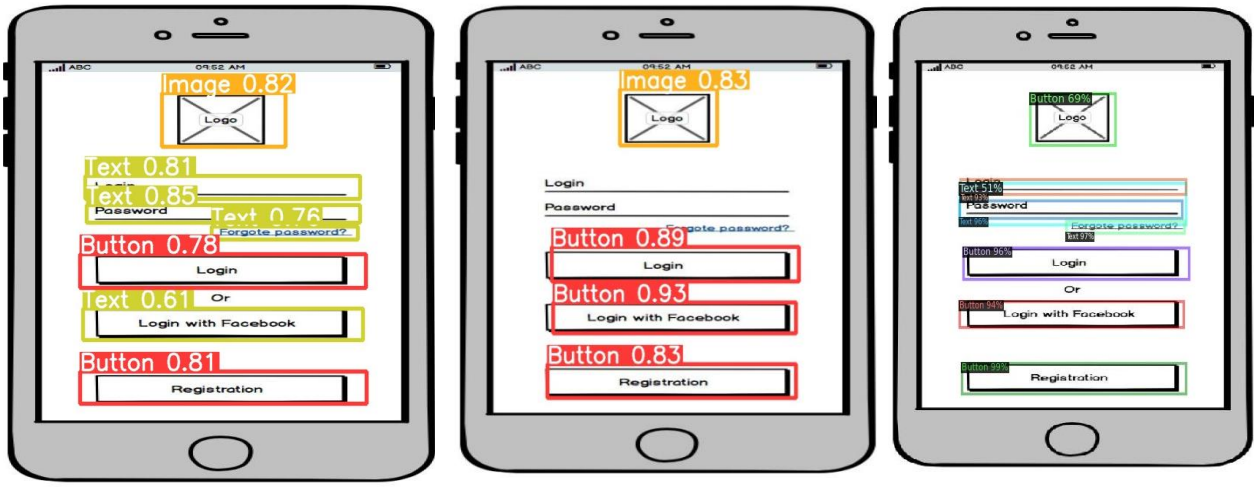
Tabela 42 - Comparação entre os melhores modelos – 4º *Dataset*.

Modelo	**Época**	**Precisão**	**mAP50**
YOLOv5x	400	77.6%	63%
YOLOv8n	1000	80.2%	61.6%
Detectron2	800	29.3%	43.7%



a) Imagem de teste, YOLOv5x. b) Imagem de teste, YOLOv8n. c) Imagem de teste, Detectron2.

Figura 113 – Deteção para os modelos: YOLOv5x, YOLOv8x e Detectron2 (imagem n. º1) – 4º *Dataset*.



a) Imagem de teste, YOLOv5x. b) Imagem de teste, YOLOv8n. c) Imagem de teste, Detectron2.

Figura 114 – Deteção para os modelos: YOLOv5x, YOLOv8x e Detectron2 (imagem n. º2) – 4º *Dataset*.

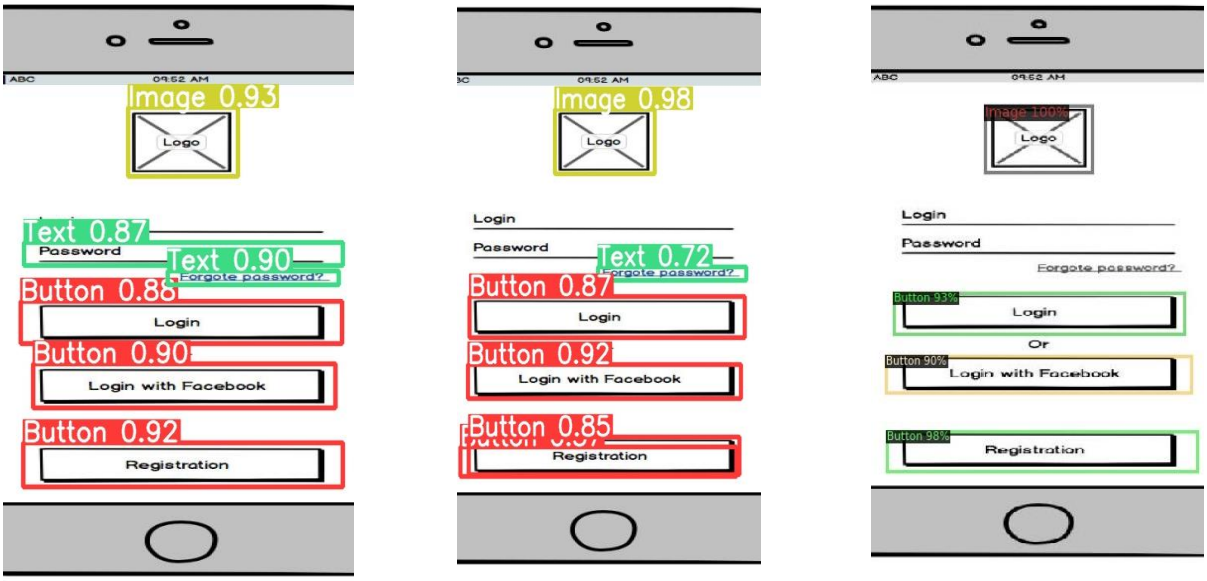
5.4.4 *5º Dataset* – União entre o 2º, 3º e 4º *dataset*

A última comparação é ao 5º *dataset*. Nesta comparação os modelos YOLO mostram métricas próximas entre si, enquanto o modelo Detectron2 tem métricas mais baixas, mas igualmente razoáveis.

Comparando as figuras que vai desde a Figura 115, Figura 116 e Figura 117, é possível verificar que os resultados são semelhantes, com o modelo YOLOv5x a apresentar resultados ligeiramente superiores ao modelo YOLOv8n e a ser melhor que o modelo Detectron2. A Tabela 43 apresenta a comparação de desempenho entre os três modelos.

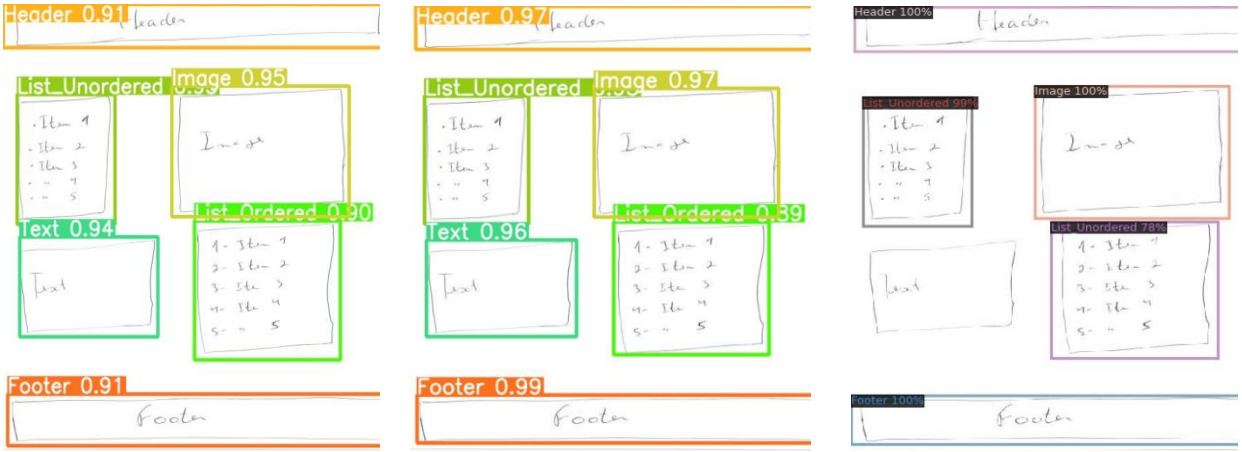
Tabela 43 - Comparação entre os melhores modelos – 5º *Dataset*.

Modelo	Época	Precisão	mAP50
YOLOv5x	800	87.4%	87.1%
YOLOv8n	200	87.1%	84.9%
Detectron2	4000	52.4%	62.5%



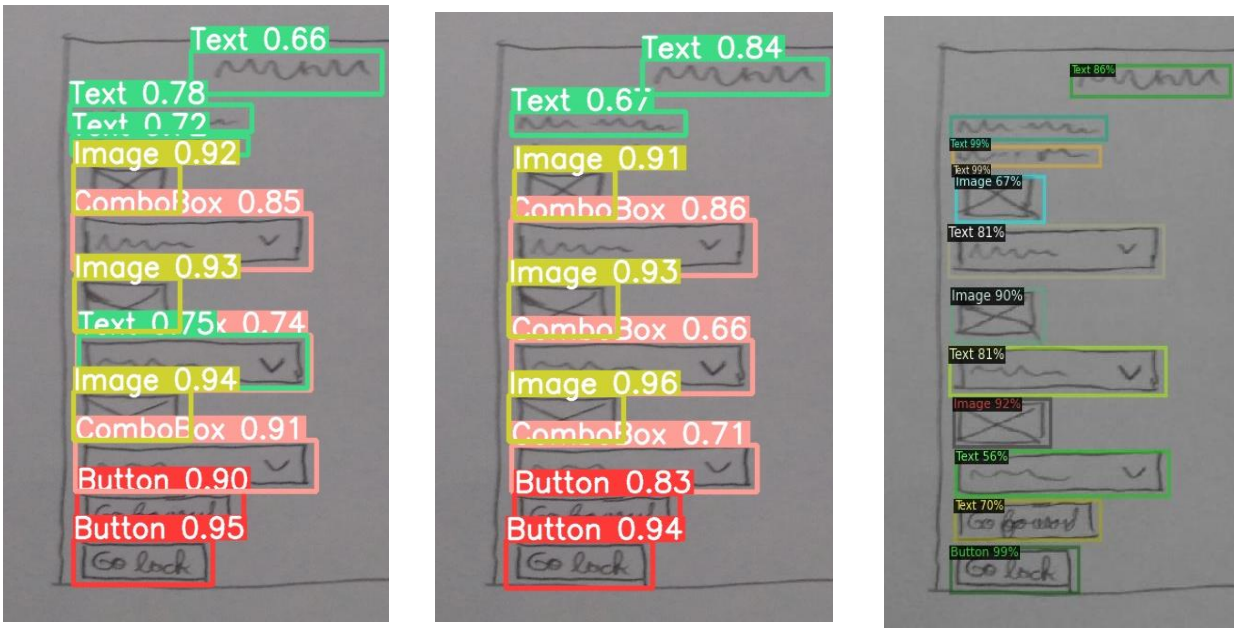
a) Imagem de teste, YOLOv5x. b) Imagem de teste, YOLOv8n. c) Imagem de teste, Detectron2.

Figura 115 – Deteção para os modelos: YOLOv5x, YOLOv8x e Detectron2 (imagem n. º1) – 5º *Dataset*.



a) Imagem de teste, YOLOv5x. b) Imagem de teste, YOLOv8n. c) Imagem de teste, Detectron2.

Figura 116 – Deteção para os modelos: YOLOv5x, YOLOv8x e Detectron2 (imagem n. º2) – 5º *Dataset*.



a) Imagem de teste, YOLOv5x. b) Imagem de teste, YOLOv8n. c) Imagem de teste, Detectron2.

Figura 117 – Deteção para os modelos: YOLOv5x, YOLOv8x e Detectron2 (imagem n. º3) – 5º *Dataset*.

5.5 Conclusão

Com todos os modelos testados e analisados foi possível obter vários resultados para cada *dataset*. Concluindo a análise para cada *dataset*, os modelos de treino para o 2º *dataset* foram bastante positivos, todos os modelos obtiveram resultados positivos, acertando na generalidade as suas previsões.

O 3º *dataset* relativo às páginas WEB feitas em HTML e CSS também obteve bons resultados nos modelos YOLO enquanto que o modelo Detectron2 já teve mais dificuldades na deteção de categorias apresentando níveis de confiança mais baixos.

O 4º *dataset* referente às *Wireframes* todos os modelos apresentaram mais dificuldades neste *dataset*, esta dificuldade deve-se particularmente por serem imagens mais pequenas e com os objetos mais próximos uns dos outros o que é uma dificuldade acrescida para estes tipos de modelos.

No 5º e último *dataset* relativo à união dos *datasets* os resultados são muito positivos em todos os modelos, com níveis de deteção muito elevados.

Na generalidade os modelos YOLO conseguem ter resultados mais satisfatórios que o Detectron2 e a escolha para este tipo de projeto seria optar pelo modelo YOLOv8.

6 CONCLUSÕES

O objetivo central deste trabalho foi a implementação de um sistema de deteção de objetos, fundamentado na capacidade de treinamento de uma rede neural de aprendizagem profunda, com o objetivo principal de integrar este sistema em uma aplicação capaz de replicar o *layout* presente nas imagens, proporcionando, assim, uma abordagem inovadora para o desenvolvimento de páginas WEB.

Durante o processo de treino foi utilizado um *dataset* divididos em quatro partes, permitindo verificar a eficácia do sistema de deteção de objetos para cada conjunto. O *dataset* é composto por uma grande variedade de imagens, com diferentes ângulos, luzes e cenários.

Foram testadas 3 versões, o modelo YOLOv5, modelo YOLOv8 e o modelo Detectron2. Entre estes modelos foram conduzidos testes para avaliar o seu desempenho e eficiência de cada um. Os modelos atingiram bons resultados e são capazes, na maioria dos casos, de detetar de forma correta as tags HTML. O modelo que obteve melhores resultados foi o modelo YOLOv8, sendo o modelo mais adequado para o objetivo do trabalho.

O potencial deste trabalho reside na aplicação prática de um sistema de deteção capaz de detetar automaticamente as tags HTML e replicá-las para o desenvolvimento de páginas WEB. Sem a necessidade de codificação manual das tags HTML este processo torna-se fácil, ágil e acessível para qualquer pessoa sem conhecimentos técnicos possa criar *layouts* de páginas WEB de maneira eficaz.

BIBLIOGRAFIA

[1] "O Que é Visão Computacional?," 24 Junho 2022. [Online]. Available: https://blog.dsacademy.com.br/o-que-e-visao_computacional/.

[2] Simplilearn, "O que é visão computacional: aplicações, benefícios e como aprender," 10 Agosto 2023. [Online]. Available: https://www.simplilearn.com/computer-vision-article.

[3] "What is computer vision and how does it work with artificial intelligence?," 30 December 2022. [Online]. Available: https://www.algotive.ai/blog/what-is-computer-vision-and-how-does-it-work-with-artificial-intelligence.

[4] N. Murali, "Image Classification vs Semantic Segmentation vs Instance Segmentation," 29 Abril 2021. [Online]. Available: https://nirmalamurali.medium.com/image-classification-vs-semantic-segmentation-vs-instance-segmentation-625c33a08d50.

[5] R. Szeliski, "Computer Vision: Algorithms and Applications 2nd Edition", The University of Washington: Springer, 2021, p. 11.

[6] "What Is Optical Character Recognition (OCR)?," 5 Janeiro 2022. [Online]. Available: https://www.ibm.com/blog/optical-character-recognition/.

[7] Klippa, "What is ICR, and Why Do You Need It?," 30 Novembro 2022. [Online]. Available: https://www.klippa.com/en/blog/information/what-is-icr/.

[8] IBM, "What is computer vision?," [Online]. Available: https://www.ibm.com/topics/computer-vision.

[9] "Algorithm, Face Detection using Viola Jones," 13 Junho 2023. [Online]. Available: https://www.mygreatlearning.com/blog/viola-jones-algorithm/.

[10] J. Brownlee, "A Gentle Introduction to the ImageNet Challenge (ILSVRC)," 5 Julho 2019. [Online]. Available: https://machinelearningmastery.com/introduction-to-the-imagenet-large-scale-visual-recognition-challenge-ilsvrc/.

[11] A. Vidhya, "CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more…," 16 Novembro 2017. [Online]. Available: https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5.

[12] Shruti, "AI vs Machine Learning vs Deep Learning: Know the Differences," 23 Fevereiro 2023. [Online]. Available: https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/ai-vs-machine-learning-vs-deep-learning.

[13] Geersforgeeks, "K means Clustering – Introduction," 25 Agosto 2023. [Online]. Available: https://www.geeksforgeeks.org/k-means-clustering-introduction/.

[14] N. Sharma, "Clustering K-Means explicado," 2023 Agosto 8. [Online]. Available: https://neptune.ai/blog/k-means-clustering.

[15] "K-Means clustering with Mall Customer Segmentation Data | Full Detailed Code and Explanation," [Online]. Available: https://www.analyticsvidhya.com/blog/2021/05/k-means-clustering-with-mall-customer-segmentation-data-full-detailed-code-and-explanation/.

[16] IBM, "K-Nearest Neighbors Algorithm," [Online]. Available: https://www.ibm.com/topics/knn.

[17] D. Tech, "O que é e como funciona o algoritmo KNN?," [Online]. Available: https://didatica.tech/o-que-e-e-como-funciona-o-algoritmo-knn/.

[18] "What makes a good prediction? Feature importance and beginning to open the black box of machine learning in genetics," Setembro 2022. [Online]. Available: https://www.researchgate.net/publication/356781515_What_makes_a_good_prediction_Feature_importance_and_beginning_to_open_the_black_box_of_machine_learning_in_genetics/figures?lo=1.

[19] Geeksforgeeks, "Support Vector Machine (SVM) Algorithm," 10 Junho 2023. [Online]. Available: https://www.geeksforgeeks.org/support-vector-machine-algorithm/.

[20] B. Stecanella, "Support Vector Machines (SVM) Algorithm Explained," 22 Junho 2017. [Online]. Available: https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/.

[21] Geeksforgeeks, "Artificial Neural Networks and its Applications," 2 Junho 2023. [Online]. Available: https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/.

[22] P. Saxena, "EXPLaiNED - Convolutional Neural Network," 1 Março 2021. [Online]. Available: https://indiaai.gov.in/article/convolutional-neural-network.

[23] N. Shahriar, "What is Convolutional Neural Network — CNN (Deep Learning)," 1 Fevereiro 2023. [Online]. Available: https://nafizshahriar.medium.com/what-is-convolutional-neural-network-cnn-deep-learning-b3921bdd82d5.

[24] H. Singh, "How Do Computers Store Images?," 29 Maio 29. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/.

[25] "Basics, Convolutional Neural Networks," 7 Abril 2017. [Online]. Available: https://mlnotebook.github.io/post/CNN1/.

[26] M. Gurucharan, “Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network,” [Online]. Available: https://www.upgrad.com/blog/basic-cnn-architecture/.

[27] A. M. Clappis, “Uma introdução as redes neurais convolucionais utilizando o Keras,” [Online]. Available: https://medium.com/data-hackers/uma-introdu%C3%A7%C3%A3o-as-redes-neurais-convolucionais-utilizando-o-keras-41ee8dcc033e.

[28] V. Sharma, “ResNets: Por que eles têm desempenho melhor do que os ConvNets clássicos? (Análise Conceitual),” 2021 Janeiro 2021. [Online]. Available: ResNets: Por que eles têm desempenho melhor do que os ConvNets clássicos? (Análise Conceitual).

[29] V. Sharma, “ResNets: Why Do They Perform Better than Classic ConvNets? (Conceptual Analysis),” 29 Janeiro 2021. [Online]. Available: https://towardsdatascience.com/resnets-why-do-they-perform-better-than-classic-convnets-conceptual-analysis-6a9c82e06e53.

[30] S.-H. Tsang, “Review: ResNet — Winner of ILSVRC 2015 (Image Classification, Localization, Detection),” 15 September 2018. [Online]. Available: https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8.

[31] H. Honda, “Digging into Detectron 2 — part 1,” 5 Janeiro 2020. [Online]. Available: https://medium.com/@hirotoschwert/digging-into-detectron-2-47b2e794fabd.

[32] Z. Keita, “YOLO Object Detection Explained,” 2022 Setembro 2022. [Online]. Available: https://www.datacamp.com/blog/yolo-object-detection-explained.

[33] Srudeep, “An Overview on MobileNet: An Efficient Mobile Vision CNN,” 10 Junho 2020. [Online]. Available: https://medium.com/@godeep48/an-overview-on-mobilenet-an-efficient-mobile-vision-cnn-f301141db94d.

[34] G. Alves, “Detecção de Objetos com YOLO – Uma abordagem moderna,” 13 Outubro 20220. [Online]. Available: https://iaexpert.academy/2020/10/13/deteccao-de-objetos-com-yolo-uma-abordagem-moderna/.

[35] J. Redmon, “You Only Look Once: Unified, Real-Time Object Detection v1”.

[36] D. Learning, “YOLO Basic Introduction. | You only LIVE once. | Object Detection.,” [Online]. Available: https://www.youtube.com/watch?v=TxPimS50PU8.

[37] S. R. Hussain, “Detecting Cow Teat Stall Numbers with CowStallNumbers Dataset,” Abril 2023.

[38] J. Redmon, “Darknet: Open Source Neural Networks in C,” [Online]. Available: https://pjreddie.com/darknet/.

[39] M. Hussain, "YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection," 23 Junho 2023.

[40] A. F. Joseph Redmon, "YOLO9000: Better, Faster, Stronger - v2".

[41] A. F. Joseph Redmon, "YOLOv3: An Incremental Improvement," 2018.

[42] A. Chakure, "All you need to know about YOLO v3 (You Only Look Once)," 1 Março 2021. [Online]. Available: https://dev.to/afrozchakure/all-you-need-to-know-about-yolo-v3-you-only-look-once-e4m.

[43] A. Kathuria, "What's new in YOLO v3?," 23 Abril 2018. [Online]. Available: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b.

[44] A. Bochkovski, "YOLOv4: Optimal Speed and Accuracy of Object Detection," 2019.

[45] V. Praharsha, "YOLOv4 model architecture," [Online]. Available: https://iq.opengenus.org/yolov4-model-architecture/.

[46] J. Solawetz, "Scaled-YOLOv4 is Now the Best Model for Object Detection," 3 Dezembro 2020. [Online].

[47] "Guide to Yolov5 for Real-Time Object Detection," 19 Dezembro 2020. [Online]. Available: https://analyticsindiamag.com/yolov5/.

[48] J. Solawets, "What is YOLOv5? A Guide for Beginners.," 29 Junho 2020. [Online]. Available: https://blog.roboflow.com/yolov5-improvements-and-evaluation/.

[49] Ultralytics, "Yolov5," [Online]. Available: https://github.com/ultralytics/yolov5.

[50] H. L. K. L. L. C. a. Y. L. Renjie Xu, "A Forest Fire Detection System Based on Ensemble Learning," 2021.

[51] A. Mehra, "Evolution of YOLO Object Detection Model From V5 to V8," 31 Março 2023. [Online]. Available: https://www.labellerr.com/blog/evolution-of-yolo-object-detection-model-from-v5-to-v8/.

[52] M. Inc., "YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications," 2022.

[53] J. S. Joseph Nelson, "YOLOv5 is Here: State-of-the-Art Object Detection at 140 FPS," 10 Junho 2020. [Online]. Available: https://blog.roboflow.com/yolov5-is-here/.

[54] A. Bochokosvkiy, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," 2022.

[55] Open-mmlab, "Mmyolo," [Online]. Available: https://github.com/open-mmlab/mmyolo.

[56] F. Jacob Solawetz, 11 Janeiro 2023. [Online]. Available: https://blog.roboflow.com/whats-new-in-yolov8/.

[57] M. Krishnakumar, "A Gentle Introduction to YOLOv8," 8 Julho 2023. [Online]. Available: https://wandb.ai/mukilan/wildlife-yolov8/reports/A-Gentle-Introduction-to-YOLOv8--Vmlldzo0MDU5NDA2.

[58] Ultralytics, "Ultralytics," [Online]. Available: https://github.com/ultralytics/ultralytics/issues/189.

[59] A. V. Glenn-jocher, "Performance Metrics Deep Dive," 12 11 2023. [Online]. Available: https://docs.ultralytics.com/guides/yolo-performance-metrics/#object-detection-metrics.

[60] J. Solawets, "What is Mean Average Precision (mAP) in Object Detection?," 6 Maio 2020. [Online]. Available: https://blog.roboflow.com/mean-average-precision/.

[61] G. Boesch, "Computer Vision Model Performance Evaluation (Guide 2024," [Online]. Available: https://viso.ai/computer-vision/model-performance/#:~:text=Importance%20of%20benchmarking,-Benchmarking%20is%20used&text=By%20comparing%20models%20on%20common,in%20computer%20vision%20model%20development..

[62] "About ImageNet," [Online]. Available: https://www.image-net.org/about.php.

[63] M. C. dataset, "MS COCO dataset," [Online]. Available: https://cocodataset.org/#home.

[64] P. VOC, "The PASCAL Visual Object Classes Homepage," [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/.

[65] C. DATASET, "CITYSCAPES DATASET," [Online]. Available: https://www.cityscapes-dataset.com/.

[66] A. Krizhevsky, "The CIFAR-10 dataset," [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html.

[67] R. Xu, "A Forest Fire Detection System Based on Ensemble Learning".

[68] D. Reis, "Real-Time Flying Object Detection with YOLOv8".

[69] Z. Zheng, "Gesture recognition real-time control system based on YOLOV4," 2022.

[70] V. Sharma, "Face Mask Detection using YOLOv5 for COVID-19".

[71] H. Zhao, "YOLOv7-sea: Object Detection of Maritime UAV Images based on Improved YOLOv7".

[72] StereoLabs*, “Performance Benchmark of YOLO v5, v7 and v8,” [Online]. Available: https://www.stereolabs.com/blog/performance-of-yolo-v5-v7-and-v8/.

[73] V. G. Sovit Rath, “Performance Comparison of YOLO Object Detection Models – An Intensive Study,” [Online]. Available: https://learnopencv.com/performance-comparison-of-yolo-models/#FPS-Performance-Comparison-of-YOLO-Models-on-CPU.

[74] A. Mehra, “Evolution of YOLO Object Detection Model From V5 to V8,” [Online]. Available: https://www.labellerr.com/blog/evolution-of-yolo-object-detection-model-from-v5-to-v8/.

[75] Ultralytics. [Online]. Available: https://github.com/ultralytics/ultralytics.

[76] Roboflow, “Roboflow,” [Online]. Available: https://app.roboflow.com/projeto-py4cs.

[77] G. Inc., “Google Colaboratory,” [Online]. Available: https://colab.google/.

[78] M. AI, “PyTorch,” [Online]. Available: https://pytorch.org/.

[79] Roboflow, “GoogleColab- YOLOv5,” [Online]. Available: https://colab.research.google.com/github/roboflow-ai/yolov5-custom-training-tutorial/blob/main/yolov5-custom-training.ipynb.

[80] Ultralytics, “GoogleColab - YOLOv8,” [Online]. Available: https://colab.research.google.com/github/ultralytics/ultralytics/blob/main/examples/tutorial.ipynb.

[81] T. Gilbert, “GoogleColab - Detectron2,” [Online]. Available: https://colab.research.google.com/github/TannerGilbert/Object-Detection-and-Image-Segmentation-with-Detectron2/blob/master/Detectron2_inference_with_pre_trained_model.ipynb.

[82] Keras, “EarlyStopping,” [Online]. Available: https://keras.io/api/callbacks/early_stopping/.

[83] J. Solawetz, “What is Mean Average Precision (mAP) in Object Detection?,” [Online]. Available: https://blog.roboflow.com/mean-average-precision/.

[84] V. Sharma, “ResNets: Why Do They Perform Better than Classic ConvNets?,” 29 Janeiro 2021. [Online]. Available: https://towardsdatascience.com/resnets-why-do-they-perform-better-than-classic-convnets-conceptual-analysis-6a9c82e06e53.