Deteção e Reconhecimento de espécies de fauna silvestre e doméstica por monitorização de câmaras de disparo automático

Inês de Sousa Pimenta
Mestrado em Estatística Computacional e Análise de Dados
Departamento de Matemática
Faculdade de Ciências da Universidade do Porto
2025

Deteção e Reconhecimento de espécies de fauna silvestre e doméstica por monitorização de câmaras de disparo automático

Inês de Sousa Pimenta

Relatório de Estágio realizado no âmbito do mestrado em Estatística Computacional e Análise de Dados

Departamento de Matemática

2025

Orientador
Prof. Dr. André Marçal, Professor Auxiliar, FCUP

Supervisor externo na Entidade de Acolhimento
Pedro Moreira, Técnico Sénior de Ambiente, Ascendi

ascendi

Agradecimentos

À minha família, pelo apoio constante e por me terem proporcionado as condições necessárias ao longo destes anos de estudo.

Um agradecimento especial ao professor André Marçal, por toda a disponibilidade e apoio prestado ao longo deste estágio e pelos valiosos conselhos que me motivaram a persistir e a ultrapassar os obstáculos emocionais associados às incertezas do futuro.

Às minhas colegas de estágio, Sandy e Amanda, por tornarem esta experiência mais leve.

Ao Pedro Moreira e a toda a equipa da Ascendi, pela oportunidade concedida e pelo contínuo reconhecimento do meu trabalho.

Resumo

A necessidade de tornar mais eficiente o processo de monitorização da fauna nas infraestruturas da Ascendi motivou o desenvolvimento do presente trabalho, que consistiu na implementação de uma solução para a deteção automática de fauna nas imagens recolhidas pelas câmaras de disparo automático espalhadas ao longo das concessões da Ascendi.

Inicialmente, foi realizado um trabalho manual de seleção e anotação das imagens obtidas ao longo dos últimos dois anos, provenientes destas câmaras. Com esta base de dados anotada, a abordagem adotada foi implementada com recurso ao modelo de deteção YOLOv8, da empresa *Ultralytics*. Adicionalmente, foi desenvolvida uma interface gráfica com o objetivo de facilitar a utilização da ferramenta pelos técnicos da Ascendi.

Os resultados obtidos evidenciaram a utilidade da solução proposta, contribuindo consideravelmente para a redução do tempo despendido na verificação manual das imagens obtidas pelas câmaras, promovendo a automação do processo de monitorização da fauna.

Palavras-chave: Inteligência Artificial, Deep Learning, Rede Neuronal, Rede Neuronal Convolucional, Deteção de Animais, YOLO

Abstract

The need to improve the efficiency of wildlife monitoring in Ascendi's infrastructures motivated the development of this work, which consisted in implementing a solution for the automatic detection of fauna in images captured by motion-activated cameras installed across Ascendi's concessions.

Initially, a manual process was carried out to select and annotate images collected over the past two years from these cameras. With this annotated dataset, the adopted approach was implemented using the YOLOv8 detection model, created by *Ultralytics*. Additionally, a graphical user interface was developed to facilitate the use of the tool by Ascendi's technical employees.

The results demonstrated the usefulness of the proposed solution, contributing to the reduction of time spent on manual verification of the images captured by the cameras by promoting the automation of the wildlife monitoring process.

Keywords: Artificial Inteligence, Deep Learning, Neural Network, Convolutional Neural Network, Animal Detection, YOLO

Índice

Lista de Tabelas

Lista de Figuras

Lista de Abreviaturas

ANN *Artificial Neural Network*.

AP *Average Precision*.

CIoU *Complete Intersection over Union*.

CNN *Convolutional Neural Network*.

COCO *Common Objects in Context*.

DFL *Distribution Focal Loss*.

FCOS *Fully Convolutional One-Stage*.

FN Falsos Negativos.

FP Falsos Positivos.

IA Inteligência Artificial.

IoU *Intersection over Union*.

mAP *mean Average Precision*.

R-CNN *Region-based Convolutional Neural Network*.

ReLU *Rectified Linear Unit*.

RoI *Region of Interest*.

RPN *Region Proposal Network*.

SGD *Stochastic Gradient Descent*.

SIG Sistema de Informação Geográfico.

SiLU *Sigmoid Linear Unit*.

SPPF *Spatial Pyramid Pooling Fast*.

VN Verdadeiros Negativos.

VP Verdadeiros Positivos.

YOLO *You Only Look Once*.

1. Introdução

A fragmentação do habitat, causada pela construção de autoestradas e outras infraestruturas lineares, é amplamente reconhecida como uma das principais causas de perda de biodiversidade, com graves impactos nos recursos naturais e nos serviços dos ecossistemas, como a disponibilidade de água potável e a proteção dos solos [1].

A perda de conectividade decorrente desta fragmentação não limita apenas a área de habitat disponível, como também impede o fluxo genético e a dispersão de muitas espécies. A barreira imposta pelas infraestruturas, aliada às alterações ecológicas nas margens dos habitats (efeito de orla), pode dificultar a mobilidade das espécies, tornando-as vulneráveis ao isolamento.

Espécies com ciclos de vida segmentados, que necessitam de diferentes habitats para completar o seu desenvolvimento, tornam-se particularmente vulneráveis. Durante as suas migrações sazonais, estas espécies frequentemente atravessam infraestruturas rodoviárias, tornando-se suscetíveis à mortalidade por atropelamento. A longo prazo, este isolamento poderá, de facto, aumentar o risco de extinções locais.

A Ascendi é uma empresa portuguesa especializada na gestão de infraestruturas rodoviárias, incluindo a operação, manutenção e cobrança de portagens. Com participações em sete concessões em Portugal (descritas na Tabela 1.1 e representadas geograficamente na Figura 1.1), cobrindo atualmente uma extensão de 707 km, a Ascendi também está presente em mercados internacionais, nomeadamente em França e Espanha.

Concessão	**Distritos Abrangidos**	**Extensão (km)**
Norte	Braga, Vila Real, Porto	179
Grande Porto	Porto	80
Douro Litoral	Porto, Aveiro	55
Costa de Prata	Porto, Aveiro	105
Beiras Litoral e Alta	Aveiro, Viseu, Guarda	172
Pinhal Interior	Coimbra, Leiria, Santarém	93
Grande Lisboa	Lisboa	23

Tabela 1.1: Concessões da Ascendi

Figura 1.1: Localização das concessões da rede Ascendi

1.1. Monitorização da Biodiversidade da Ascendi

Ciente dos impactos ecológicos resultantes da fragmentação do habitat, a Ascendi desenvolveu um plano de ação com o objetivo de mitigar esses efeitos. Para garantir a implementação de medidas eficazes, a Ascendi estabeleceu uma rede de monitorização abrangente ao longo das infraestruturas sobre a sua gestão.

Como parte desta estratégia, a equipa de Biodiversidade da Ascendi tem analisado o atravessamento da fauna através das diversas passagens existentes (passagens agrícolas, viadutos, pontes, passagens hidráulicas, etc.), identificando os principais corredores ecológicos (áreas que conectam habitats fragmentados, permitindo que espécies de fauna se desloquem entre eles). Para a concretização deste estudo, foram instaladas câmaras de disparo automático, como é exemplo a da Figura 1.2. Uma metodologia amplamente utilizada em estudos ecológicos para monitorizar a fauna de forma não invasiva.

As câmaras adquiridas possuem uma velocidade de disparo de aproximadamente 0.2 segundos, um tempo de recuperação de 0.8 segundos e um alcance de deteção de aproximadamente 34 metros. Após instaladas, estas câmaras são deixadas ativas no local durante um período de 15 dias por estação do ano. De forma a atrair os animais para

Figura 1.2: Exemplo de uma câmara de monitorização da Ascendi instalada na Concessão Norte e exemplos de duas imagens capturadas por esta câmara

o campo de visão da câmara, é colocado no local, aquando da sua instalação, um composto atrativo, como a tintura de Valeriana.

Esta tecnologia possibilita então a obtenção de índices de abundância com base no número de passagens registadas, bem como a recolha de informações adicionais, como a idade, o sexo e o comportamento dos indivíduos ao aproximarem-se e utilizarem cada passagem específica.

Outra vantagem desta metodologia é a possibilidade de manutenção mínima no campo, sendo apenas necessária a deslocação para a instalação e recolha das máquinas para obtenção das imagens capturadas, não estando a sua utilização dependente de condições meteorológicas.

No entanto, apesar das suas inúmeras vantagens, a utilização destas câmaras apresenta alguns desafios. Um dos principais constrangimentos deste método é o risco de roubo ou vandalismo dos equipamentos. Além da perda de dados, o furto ou destruição das câmaras representa um custo financeiro significativo, pelo que a necessidade de minimizar esse risco pode limitar a disposição dos dispositivos em campo.

As câmaras são ativadas assim que é detetado movimento. No entanto, em áreas com densa vegetação, grande parte do movimento detetado é causado pelo próprio deslocamento das plantas, seja devido ao vento ou ao seu crescimento. Consequentemente, muitas imagens são capturadas sem a presença de um animal. Nesse sentido, o processamento posterior destas imagens implica um esforço e tempo consideráveis para selecionar todos os registos relevantes, pelo que, a automatização desse processo revela-se

uma solução ideal. E é precisamente este o objetivo do meu estágio na Ascendi: automatizar a deteção, incluindo a identificação dos animais presentes nas imagens recolhidas por esta tecnologia.

Neste estágio, fui integrada no departamento de Gestão da Conservação da Ascendi, especificamente na equipa de Ambiente e Biodiversidade. É de nota ainda referir que durante estes meses na Ascendi tive a oportunidade de realizar uma visita de campo e auxiliar na montagem destas máquinas, observando de perto os desafios referidos associados à sua instalação, especialmente no que toca a garantir uma posição discreta e eficiente.

1.2. Breve resumo e Estrutura do documento

O objetivo deste estágio consistiu, portanto, na criação de uma ferramenta capaz de detetar automaticamente a presença de fauna nas imagens recolhidas pelas câmaras de disparo automático instaladas nas diversas passagens das concessões da Ascendi, localizando o animal e classificando-o com o nome da espécie correspondente.

Os procedimentos para o concretizar passaram pela seleção e organização das imagens capturadas pelas câmaras mencionadas, fazendo a triagem entre as que continham pelo menos um animal das restantes. Em seguida, procedeu-se à respetiva anotação: para cada imagem, com recurso a uma ferramenta apropriada, delimitou-se manualmente o objeto ou objetos de interesse; um ficheiro contendo as informações específicas da imagem e as coordenadas delimitadoras do objeto, assim como a classe atribuída, foi guardado. Preparando-se assim o conjunto de dados para o processo de treino do modelo de deteção.

Dividiu-se o conjunto de dados num conjunto de treino, validação e teste. Treinou-se o modelo de deteção no conjunto de treino, tendo o conjunto de teste sido utilizado para testar a capacidade de generalização do modelo. Para avaliar se a capacidade preditiva do modelo poderia ser melhorada foram aplicadas técnicas de aumento de dados, tendo sido criadas novas imagens a partir das imagens das classes minoritárias existentes no conjunto de treino.

Com o intuito de facilitar a utilização futura desta ferramenta pela Ascendi foi desenvolvida uma interface gráfica de utilizador. Para testar a capacidade de previsão do modelo num novo conjunto de teste, independente do considerado inicialmente, recolheram-se novas imagens de câmaras instaladas na concessão BLA.

A linguagem de programação utilizada para estas tarefas foi o *Python* (versão 3.12.3). Recorreu-se a bibliotecas como o *NumPy* no contexto de operações matriciais. Para a

implementação do modelo de deteção escolhido recorreu-se ao *PyTorch* e à biblioteca *Ultralytics*. A biblioteca *Pillow* foi utilizada para a importação, manipulação e processamento de imagens, enquanto o *Matplotlib* permitiu a visualização dos dados, nomeadamente a exibição de imagens e gráficos relevantes.

Este relatório de estágio está organizado do seguinte modo:

No **Capítulo 2** são descritos os métodos utilizados no desenvolvimento do projeto. São abordados conceitos fundamentais como Redes Neuronais Artificiais e Redes Neuronais Convolucionais. Além disso, é explorado o modelo YOLO, incluindo a sua arquitetura e implementação. O capítulo termina com uma explicação sobre as métricas utilizadas para avaliar o desempenho do modelo.

No **Capítulo 3** é apresentado o conjunto de dados utilizado no estudo, incluindo a sua descrição e o pré-processamento realizado. São discutidos aspetos como a anotação das imagens e as técnicas de *Data Augmentation* aplicadas.

No **Capítulo 4** são analisados os resultados obtidos, nomeadamente o desempenho do modelo treinado. Adicionalmente, é descrita a interface gráfica desenvolvida para a utilização do modelo e os resultados obtidos num conjunto novo de imagens recolhidas da concessão BLA.

Por fim, no **Capítulo 5**, são apresentadas algumas considerações sobre o trabalho realizado, sendo apontadas possíveis melhorias ou trabalhos futuros.

2. Métodos

2.1. Inteligência Artificial e Deteção de Objetos

Em 1939, o matemático inglês Alan Turing desenvolveu uma máquina de decifração de códigos chamada *The Bombe* para o governo britânico, com o objetivo de decifrar o código Enigma utilizado pelo exército alemão durante a Segunda Guerra Mundial. Ter sido capaz de decifrar o código Enigma, uma tarefa anteriormente impossível até para os melhores matemáticos, fez com que Turing se questionasse sobre a inteligência de tais máquinas.

Em 1950, publicou o artigo ”Computing Machinery and Intelligence”, onde conjeturava sobre a criação de máquinas inteligentes e sobre como testar a sua inteligência. Este Teste de Turing é ainda hoje considerado um referencial para identificar a inteligência de um sistema artificial: se um ser humano interagir com outro ser humano e com uma máquina e se este não conseguir distinguir a resposta dada pela máquina da dada pelo ser humano, então considerar-se-à a máquina inteligente [2] [3].

Turing concebia a criação de uma máquina inteligente como uma simulação da mente de uma criança que, ao ser submetida a um curso adequado de ’educação’, maturar-se-ia até alcançar um ’cérebro adulto’. Como o cérebro infantil possui muito pouco mecanismo e muitas ’páginas em branco’, Turing supunha que algo semelhante poderia ser facilmente programado [2].

A expressão Inteligência Artificial (IA) foi oficialmente empregue cerca de seis anos depois, em 1956, no Dartmouth Summer Research Project on Artificial Intelligence (DSRPAI), na Dartmouth College, em New Hampshire, que teve por objetivo reunir investigadores de várias áreas para a criação de uma nova área de pesquisa voltada para a construção de máquinas capazes de simular a inteligência humana. Os participantes incluíam o cientista de computação Nathaniel Rochester, que mais tarde desenhou o IBM 701, o primeiro computador científico comercial [4].

Nos primeiros tempos da IA, o progresso foi rápido na resolução de problemas que, embora intelectualmente exigentes para os humanos, eram relativamente simples para os computadores – desafios que podiam ser expressos através de regras matemáticas bem definidas, como foi exemplo o programa de xadrez Deep Blue da IBM, capaz de processar 200 milhões de jogadas por segundo e que em 1997 conseguiu derrotar o campeão mundial de xadrez, Garry Kasparov. No entanto, a grande dificuldade da IA

revelou-se na resolução de tarefas que são fáceis para os seres humanos, mas difíceis de descrever formalmente – problemas que resolvemos de forma intuitiva e automática, como o reconhecimento de objetos em imagens [3].

A solução passaria por permitir que os computadores aprendessem com a experiência e compreendessem o mundo através de uma hierarquia. Ao adquirir conhecimento a partir da experiência, esta abordagem evita que tenhamos de especificar formalmente todas as informações de que a máquina necessita. Esta hierarquia de representação dos dados permite que o computador construa noções mais complexas a partir de elementos mais básicos. Se representarmos graficamente a forma como esta hierarquia se constrói, obtemos um modelo 'profundo', isto é, com várias camadas. Por este motivo, esta abordagem à inteligência artificial é conhecida por aprendizagem profunda (*deep learning*).

Deep Learning é tido como um subconjunto de *Machine Learning* que, como ilustrado na Figura 2.1, automatiza grande parte do processo de extração de características dos dados, reduzindo substancialmente a necessidade de intervenção humana. Esta área tem sofrido um crescimento exponencial desde a última década, sobretudo devido ao crescente poder computacional conseguido e ao acesso a grandes quantidades de dados.

2.1.1 Redes Neuronais Artificiais

Os algoritmos de *deep learning* têm por base o que se designa por Redes Neuronais Artificiais (*Artificial Neural Network* (ANN)). Este conceito precede o de *deep learning* e teve a sua origem em 1943 no artigo "A logical calculus of the ideas immanent in nervous activity", por Warren McCulloch e Walter Pitts ([5]). Nele, os autores estabeleceram o primeiro modelo matemático de um neurónio, concebendo-o como um dispositivo lógico binário [6].

O princípio do 'tudo ou nada' inerente à transmissão do impulso nervoso, ou seja, o facto de um impulso só ser gerado se o estímulo associado atingir uma determinada intensidade, motivou McCulloch e Pitts a recorrer à lógica simbólica para descrever a atividade neuronal, considerando este princípio suficiente para demonstrar que o comportamento neuronal era essencialmente proposicional [7].

Como ilustrado na imagem da Figura 2.2, o neurónio de McCulloch-Pitts recebe um conjunto de m inputs binários, com determinados pesos binários associados, $w_i, i \in \{1, ...m\}$, realiza a soma ponderada deles e gera um valor de saída com base na seguinte função

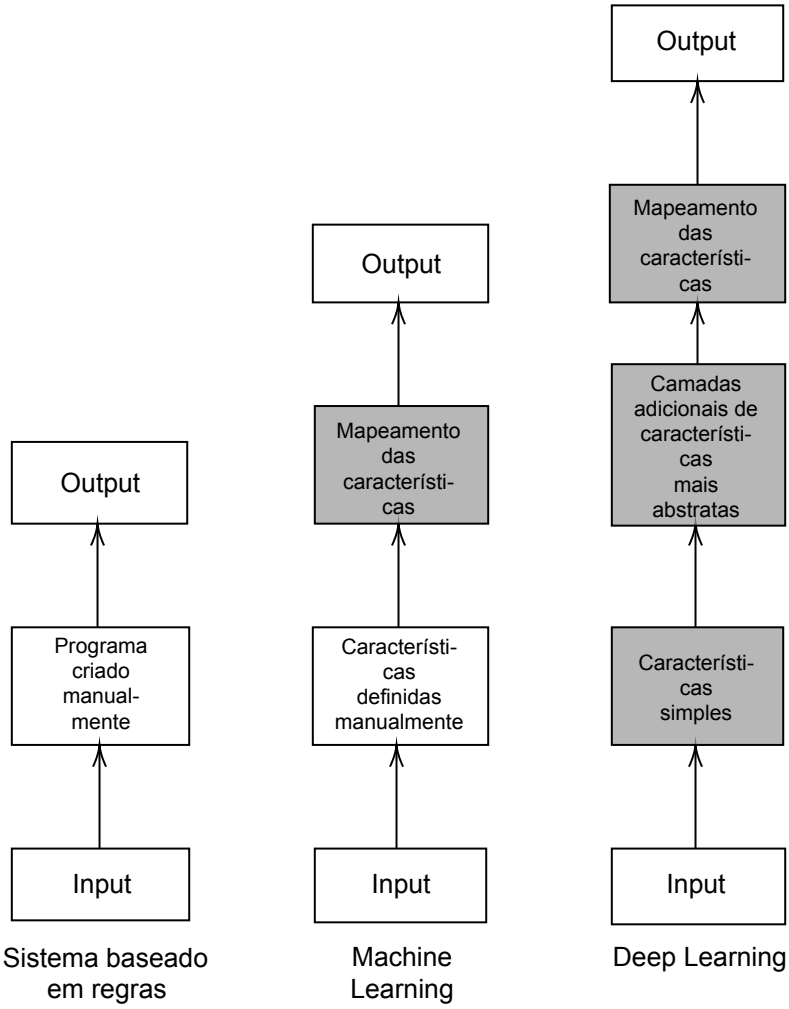


Figura 2.1: Diagramas representativos de como as diferentes partes de um sistema de IA se relacionam entre si. As caixas a cinzento indicam as componentes que são capazes de aprendizagem. Inspirado por [3].

de ativação: se a soma dos inputs ponderados exceder o valor de um determinado limiar (*threshold*), t, o neurónio é 'ativado' e o sinal de saída é 1. Caso contrário, o neurónio não é ativado e o sinal de saída é 0.

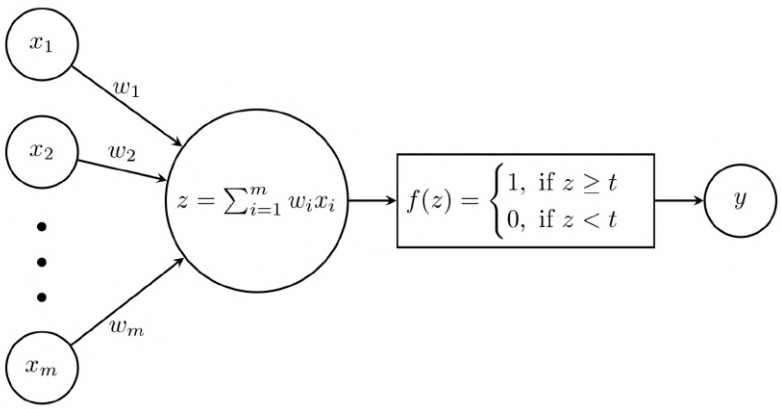


Figura 2.2: Representação do neurónio de McCulloch e Pitts [8]

Atualmente, no campo da neurobiologia, esta noção do sistema nervoso enquanto dispositivo exclusivamente proposicional já não é considerada uma representação fidedigna [9]. No entanto, foi a partir deste propósito que as redes neuronais artificiais se fundaram e passaram a desempenhar um papel fundamental na teoria da computação.

Um único neurónio artificial pode realizar apenas uma decisão muito simples; muitos neurónios conectados entre si podem desempenhar decisões mais complexas. Assim, uma rede neuronal artificial é composta por múltiplas camadas de nós/neurónios, organizadas em três tipos principais: uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída, como ilustrado na Figura 2.3. É calculada a soma ponderada dos *inputs*, sendo o valor obtido sujeito a uma função de ativação. No modelo clássico do neurónio de McCulloch-Pitts, esta função corresponde a uma função degrau unitário. No entanto, existem várias outras funções de ativação amplamente utilizadas em redes neuronais modernas, tais como a função sigmoide ($f(z) = \frac{1}{1+e^{-z}}$, $z \in \mathbb{R}$), a função tangente hiperbólica (tanh, $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, $z \in \mathbb{R}$) e a função *Rectified Linear Unit* (ReLU) ($f(z) = max(0, z)$, $z \in \mathbb{R}$), cada uma com características específicas que influenciam o desempenho da rede. Estas funções de ativação introduzem não-linearidade aos dados, o que possibilita a aprendizagem de padrões mais complexos. O valor resultante da função de ativação é então transmitido como *input* à camada seguinte.

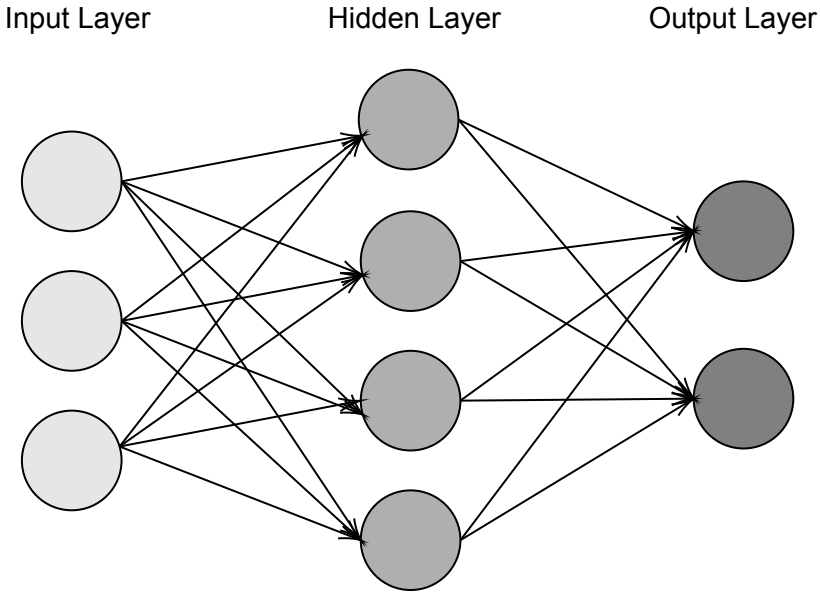


Figura 2.3: Representação gráfica de um Rede Neuronal

Um algoritmo de *deep learning* é na sua essência uma rede neuronal com mais do que três camadas, isto é, com múltiplas camadas ocultas.

Para que a rede seja capaz de tomar decisões autonomamente, é necessário submete-la a um processo de aprendizagem primeiro. Este processo é realizado com recurso a um conjunto de dados de treino, consistindo em observações devidamente anotadas com a categoria pretendida. Durante a fase de aprendizagem da rede, a máquina recebe cada uma destas observações como entrada e gera para cada uma delas uma previsão. O objetivo é que estas previsões se aproximem o melhor possível da solução real. Para tal, determina-se uma função objetivo (ou função perda) que representa o quão distantes estão as previsões do modelo em relação aos valores reais. Com base nesse erro, a máquina ajusta os seus parâmetros internos no sentido de o minimizar. Num modelo típico de *deep learning*, podem existir centenas de milhões de parâmetros/pesos ajustáveis.

O gradiente descendente é um algoritmo de otimização iterativo usado para minimizar a função de perda. A regra de atualização do gradiente descendente tradicional é baseada no cálculo do gradiente da função de perda em relação aos parâmetros do modelo (pesos, viés, etc.), ajustando-os na direção oposta ao gradiente, com um determinado passo (taxa de aprendizagem). A expressão que traduz isto é dada por 2.1.

$$\theta_{i+1} = \theta_i - \alpha \nabla L(\theta), \tag{2.1}$$

onde α representa a taxa de aprendizagem, θ o conjunto de parâmetros e $\nabla L(\theta)$ o respetivo gradiente.

No gradiente descendente tradicional, os gradientes são calculados com base em todo o conjunto de dados, o que pode ser muito exigente do ponto de vista computacional, especialmente em conjuntos de dados muito grandes.

É aqui que entra o Gradiente Descendente Estocástico (*Stochastic Gradient Descent* (SGD)). Em vez de usar todos os dados para calcular o gradiente em cada iteração, o SGD utiliza apenas um pequeno conjunto por iteração, designado por *mini-batch*. Isto torna o processo muito mais rápido e eficiente.

Os pesos/parâmetros são então atualizados, com o objetivo de reduzir o erro progressivamente ao longo do treino.

Após o treino, o desempenho do modelo é avaliado num conjunto de dados distintos, o conjunto de teste. Esta etapa permite verificar a capacidade de generalização do modelo, isto é, a sua aptidão para gerar respostas coerentes perante novas entradas que nunca encontrou durante o treino [10].

2.1.2 Redes Neuronais Convolucionais

As Redes Neuronais Convolucionais (*Convolutional Neural Network* (CNN)) são um tipo de redes neuronais concebidas para processar dados com uma estrutura em forma de grelha, como é o caso das imagens, computacionalmente representáveis por uma matriz bidimensional de píxeis.

O termo 'Convolucionais' deriva do facto de empregarem uma operação matemática denominada por convolução, que vem substituir a multiplicação dos inputs pelos pesos subjacente às redes neuronais tradicionais.

Como exemplificado na Figura 2.4, estas redes são compostas por três tipos de camadas principais: a camada convolucional, a camada de *Pooling* e a camada totalmente conectada (*Fully Connected*).

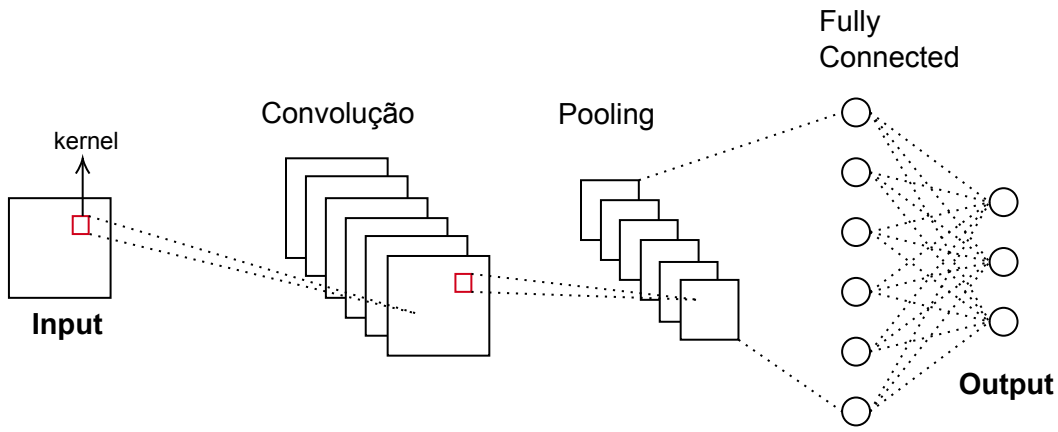


Figura 2.4: Representação simplificada de uma rede neuronal convolucional

Um *kernel* define um filtro que percorre a imagem de entrada, deslocando-se da esquerda para a direita e de cima para baixo (geralmente de pequenas dimensões, muito inferiores às da imagem, como 3×3, 5×5 ou 7×7 píxeis). Em cada posição, é realizado um produto de convolução entre os valores do *kernel* e a região correspondente da imagem. O resultado desta operação é uma nova imagem, uma imagem filtrada, que realça certas características, como arestas, formas ou texturas, dependendo do filtro utilizado. O resultado da aplicação do *kernel* sobre a imagem é geralmente denominado por mapa de características (*feature map*).

Na Figura 2.5, encontra-se representado um exemplo desta operação. Neste caso, um *kernel* de 2×2 percorre uma matriz de entrada 4×4, deslocando-se um elemento de cada vez, da esquerda para a direita e de cima para baixo (*stride* 1×1). Em cada posição, calcula-se a soma dos produtos entre os valores do *kernel* e os da região correspondente da entrada.

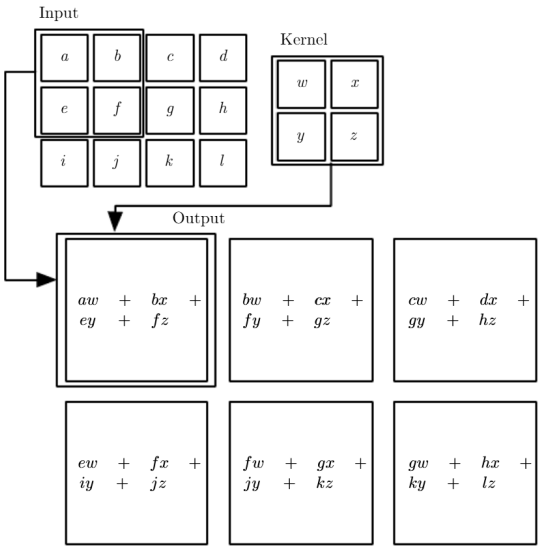


Figura 2.5: Exemplo de uma convolução 2D [3]

A operação de *pooling*, por sua vez, tem por função reduzir as dimensões do mapa de características, substituindo cada região por uma estatística resumida, como o valor máximo ou a média das saídas vizinhas, pretendendo tornar a representação aproximadamente invariante a pequenas translações da entrada e permitindo a diminuição da complexidade computacional do modelo. Na Figura 2.6, encontram-se ilustradas as operações mencionadas, *Max Pooling* e *Average Pooling*. Na primeira é devolvido o valor máximo dentro de uma vizinhança 2×2 e na segunda a média respetiva.

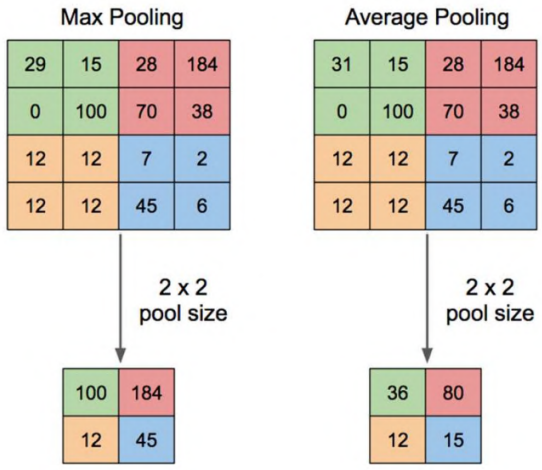


Figura 2.6: Exemplificação de aplicação de *Max Pooling* e *Average Pooling* [11]

Após a extração e redução de características através das camadas convolucionais e de *pooling*, o mapa de características resultante é achatado (*flattened*), sendo transformado num vetor coluna que será transmitido à camada totalmente conectada (*fully connected*).

Esta camada, como o próprio nome indica, conecta cada elemento do vetor unidimensional a todos os neurónios da camada seguinte, permitindo que todas as características extraídas contribuam para a decisão final. Essa conectividade total é então fundamental pois permite à rede combinar a informação local obtida nas etapas anteriores e aprender relações mais complexas e contextos globais, tendo assim um papel crucial na classificação final e geração de previsões.

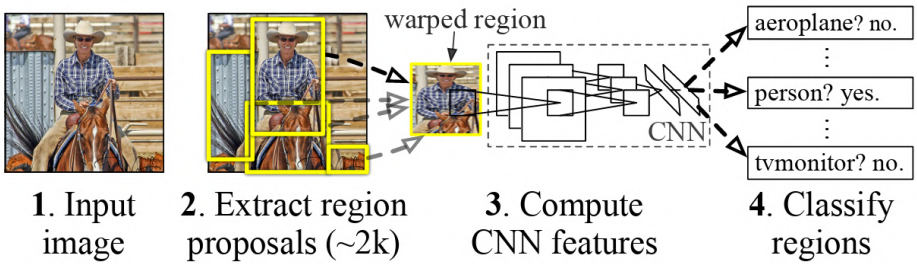
As Redes Neuronais Convolucionais oferecem uma abordagem eficiente para a classificação de imagens e reconhecimento de objetos. No entanto, devido à sua elevada complexidade computacional, o treino destes modelos exige frequentemente unidades de processamento gráfico (GPUs) de grande desempenho.

2.1.3 Modelos de deteção de objetos

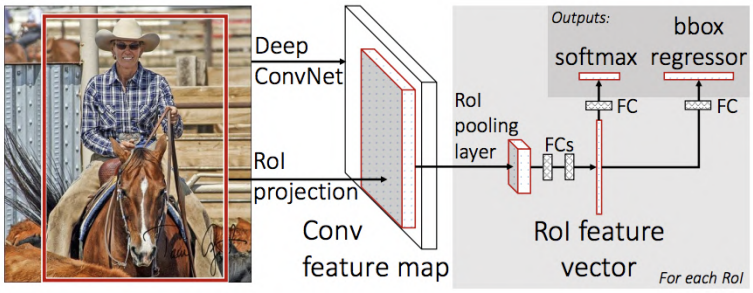
Os sistemas de deteção de objetos tradicionais adaptam classificadores para realizarem a tarefa de deteção. Para detetarem um objeto, esses métodos utilizam um classificador específico e aplicam-no em diferentes localizações e escalas da imagem de teste. O problema desta abordagem é que, como os objetos de interesse podem apresentar diferentes localizações espaciais na imagem e proporções variadas, é necessário selecionar um número muito elevado de regiões, o que pode tornar o processo computacionalmente inviável.

Abordagens mais recentes, como o *Region-based Convolutional Neural Network* (R-CNN), recorrem a métodos de propostas de regiões para gerar, inicialmente, possíveis caixas delimitadoras (*bounding boxes*) na imagem. Em vez de tentar classificar um número muito elevado de regiões, o modelo R-CNN utiliza o algoritmo de procura seletiva (*Selective Search*) para extrair cerca de 2000 regiões por imagem onde irá aplicar um classificador. Este algoritmo de procura seletiva consiste no seguinte: primeiro é gerada uma sub-segmentação inicial da imagem, criando-se um grande número de regiões candidatas; depois aplica-se um algoritmo heurístico de combinação sequencial para unir, de forma recursiva, regiões semelhantes e formar agrupamentos maiores, utilizando-se as regiões agrupadas para produzir as propostas finais de regiões candidatas.

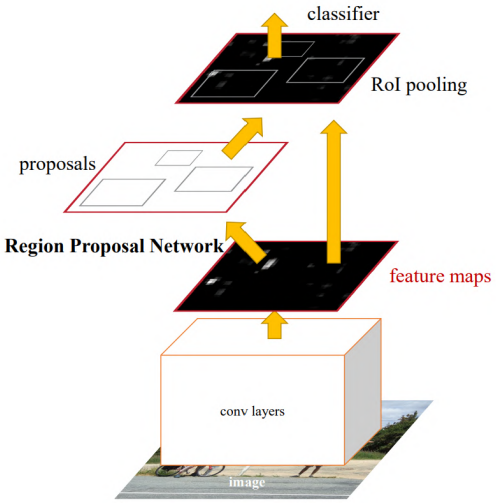
Estas regiões propostas são, então, usadas como base para aplicar um classificador que identifica os objetos nelas contidos. Após essa etapa, é necessário um processamento adicional para refinar as *bounding boxes* geradas, eliminar deteções duplicadas e ajustar os resultados com base na presença de outros objetos em cena. Na Figura 2.7a encontra-se uma representação esquemática deste modelo.



(a) R-CNN [12]



(b) Fast R-CNN [13]



(c) Faster R-CNN [14]

Figura 2.7: Comparação das diferentes arquiteturas R-CNN

Este método continua a ser computacionalmente exigente, já que o classificador tem de ser aplicado a cada uma das 2000 regiões por imagem. Além disso, o *Selective Search* é um algoritmo fixo, sem capacidade de aprendizagem, o que pode resultar na geração de propostas de regiões pouco relevantes, comprometendo a eficácia global do modelo.

Uma versão melhorada que resolve algumas das limitações do modelo inicial foi proposta, resultando num algoritmo de deteção de objetos mais rápido, denominado Fast R-CNN. A abordagem é semelhante à do R-CNN, mas com uma diferença fundamental: em vez de a rede convolucional ser aplicada individualmente a cada proposta de região,

o Fast R-CNN aplica-a diretamente à imagem completa, gerando um mapa de características convolucionais. A partir deste mapa, as regiões propostas são identificadas e ajustadas para uma forma quadrada, sendo depois redimensionadas para um tamanho fixo através de uma camada *Region of Interest* (RoI) *Pooling*, o que permite o seu processamento por uma camada totalmente conectada (*fully connected layer*). A partir do vetor de características da RoI, a rede utiliza uma camada softmax para prever a classe do objeto detetado e uma regressão para ajustar as coordenadas da caixa delimitadora.

O Fast R-CNN, representado esquematicamente na Figura 2.7b, é muito mais rápido do que o R-CNN, pois evita processar separadamente as 2000 propostas de região em cada imagem. A operação convolucional é feita apenas uma vez por imagem, tornando o processo muito mais eficiente.

Contudo, o Fast R-CNN ainda depende do algoritmo *Selective Search* para gerar propostas de regiões, o que continua a ser um processo lento e computacionalmente dispendioso.

Para ultrapassar esta limitação, surge o Faster R-CNN (Figura 2.7c), que elimina completamente o uso do *Selective Search*. Em vez disso, a rede aprende automaticamente a gerar propostas de regiões através de uma rede especializada denominada *Region Proposal Network* (RPN). À semelhança do Fast R-CNN, a imagem é processada por uma rede convolucional que gera um mapa de características, mas agora as propostas de regiões são previstas diretamente por esta sub-rede. Estas propostas são depois ajustadas com a camada RoI *Pooling* e classificadas, tal como nas versões anteriores, com a previsão simultânea das classes e da localização precisa das caixas delimitadoras.

Esta inovação resultou numa melhoria substancial tanto em termos de precisão como de velocidade, tornando o Faster R-CNN num dos modelos amplamente utilizados em tarefas de deteção de objetos.

No contexto da ecologia, em particular para casos de estudo semelhantes ao presente, alguns destes métodos foram já aplicados. Por exemplo, em [15] e [16] os autores utilizaram o modelo pré-treinado Faster-RCNN combinado com a arquitetura InceptionResNetV2 como *backbone* (componente responsável pela extração de características das imagens) para a localização e classificação de fauna em imagens também elas obtidas por câmaras de disparo automático.

No primeiro caso [15], o modelo pré-treinado foi diretamente aplicado a um conjunto de 110 imagens, correspondentes a dez espécies diferentes de mamíferos selvagens europeus, como o javali, o corço e a raposa-vermelha. Os resultados obtidos evidenciaram

uma taxa de deteção da região correta de interesse (área correspondente ao animal) de 94%, e uma precisão de 71% na identificação correta do nome da espécie.

No segundo caso [16], recorreu-se ao que se designa por *fine-tuning*, uma técnica de *transfer learning* que consiste na adaptação das últimas camadas de um modelo previamente treinado a um novo conjunto de dados, preservando o conhecimento adquirido na fase inicial. Aplicando esta abordagem a um conjunto de 30832 imagens de treino, abrangendo 13 classes distintas de mamíferos, alcançou-se uma precisão de 96.88% na tarefa de deteção e 73.92% na tarefa de classificação [16].

Em [17], os autores realizaram um estudo comparativo entre diferentes arquiteturas de deteção de objetos aplicadas a imagens capturadas por câmaras de disparo automático no Parque Nacional do Tigre e Leopardo do Nordeste da China. Para tal, recolheram um conjunto de dados, composto por mais de 25000 imagens de 17 espécies distintas. As arquiteturas analisadas incluíram o modelo YOLOv5 (versão 5 do modelo YOLO - modelo escolhido para o desenvolvimento deste trabalho e que será explorado adiante), o modelo *Fully Convolutional One-Stage* (FCOS), um detetor que elimina a necessidade de caixas (ou âncoras, de tamanho fixo) pré-definidas para localizar objetos, prevendo diretamente as localizações e dimensões dos objetos numa imagem, utilizando uma rede totalmente convolucional; e o modelo *Cascade* R-CNN que constitui uma evolução face aos modelos da família R-CNN, tendo sido concebido para melhorar o desempenho da deteção de objetos difíceis de delimitar com precisão. Os resultados demonstraram um desempenho superior do modelo YOLOv5 face às restantes arquiteturas [17].

2.1.4 YOLO

Os modelos referidos anteriormente (à exceção do modelo YOLOv5 e FCOS), por serem compostos por etapas independentes, têm um grau de complexidade elevado, sendo lentos e difíceis de otimizar, uma vez que cada componente precisa de ser treinada separadamente.

O modelo *You Only Look Once* (YOLO) introduzido em 2015 [18], reformula o problema da deteção de objetos como uma tarefa única de regressão. Uma única rede convolucional prevê as *bounding boxes* e probabilidades de classe respetivas. O YOLO analisa a imagem apenas uma vez, daí a sua designação, prevendo o que está presente e onde.

Comparativamente aos seus predecessores, o YOLO é mais simples: prevê, em simultâneo, múltiplas caixas delimitadoras e as respetivas probabilidades de classe. O modelo é treinado com imagens completas e otimizado diretamente para a tarefa de deteção, o que traz várias vantagens, incluindo maior velocidade, simplicidade e eficácia [18].

Existem já várias versões deste modelo. Aquela que utilizei ao longo deste trabalho foi a versão 8 (YOLOv8), lançada pela empresa de software *Ultralytics* em 2023. Esta versão encontra-se disponível em acesso livre e distingue-se das versões anteriores da arquitetura pelas melhorias em termos de desempenho, precisão e facilidade de integração [19].

2.1.4.1 Descrição do modelo YOLOv8

Na Figura 2.8 é apresentado um diagrama representativo da arquitetura do modelo YOLOv8, disponibilizado no GitHub por um utilizador em colaboração com a *Ultralytics*.

De uma forma geral, o esqueleto (*Backbone*) é responsável pela extração de características da imagem de entrada, como a textura, os contornos, a cor, etc., enquanto a cabeça (*Head*) é responsável por processar os mapas de características gerados, transformando-os no output final do modelo sob a forma de *bounding boxes* (’Bbox’) e classes dos objetos (’Cls’).

Relativamente à legenda do diagrama da Figura 2.8, os diversos retângulos representam as camadas da rede, com rótulos que descrevem o tipo de camada (*Conv*, *C2f*, *Upsample*, *Concat*, SPPF, etc.) e os respetivos parâmetros relevantes. No caso da camada Conv, parâmetros como o tamanho do *kernel* (k), o *stride* (s, que determina o número de pixéis que o *kernel* se desloca a cada passo, sendo que valores maiores reduzem a resolução do mapa de características resultante), o *padding* (p, corresponde à adição

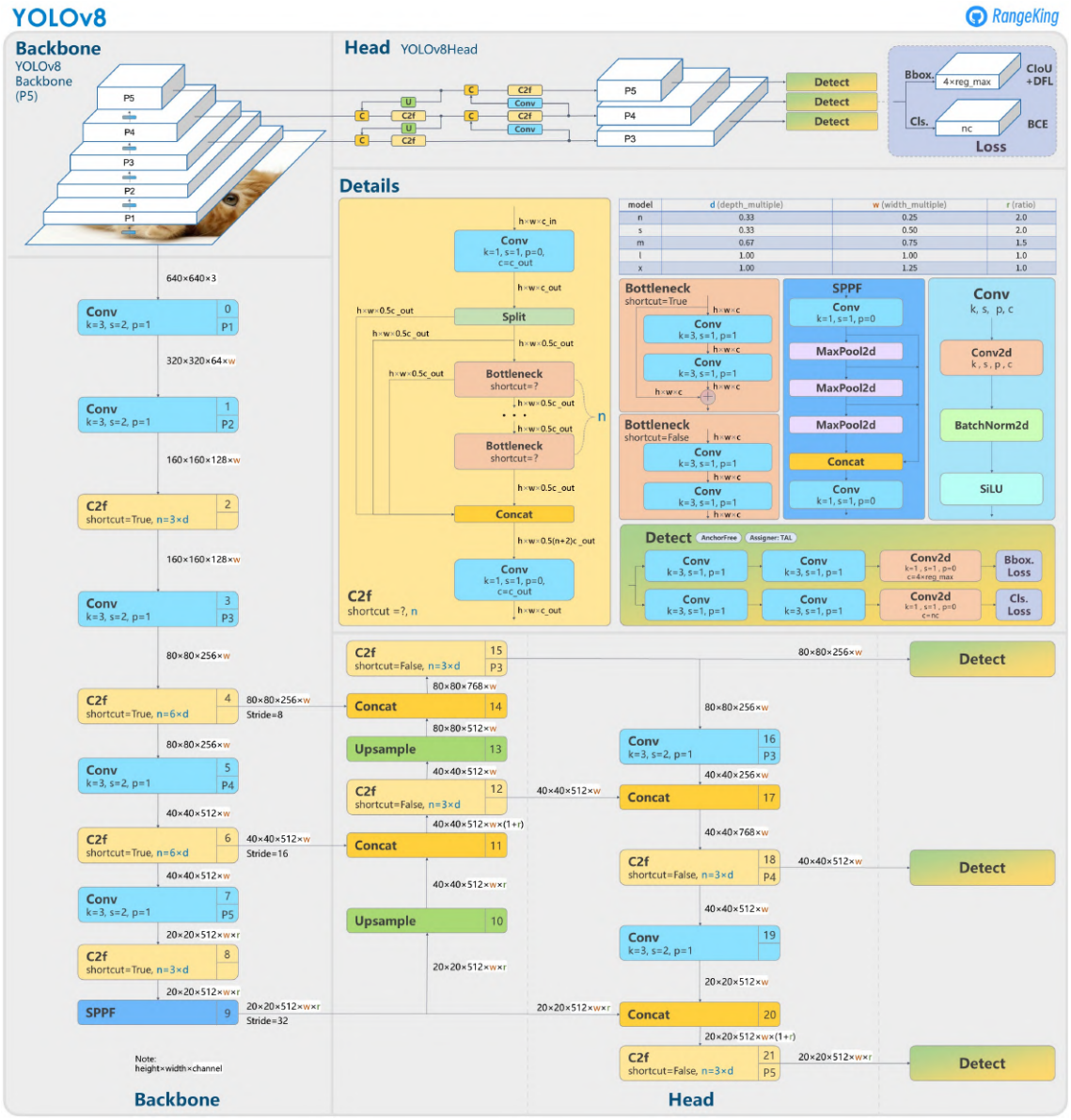


Figura 2.8: Arquitetura do modelo YOLOv8 [20]

de pixéis extra na margem da imagem para evitar redução excessiva do tamanho após a convolução. Se $p = 0$ não é aplicado *padding*, se $p = 1$ é adicionada uma margem de zeros em todos os lados do mapa de características de entrada); entre camadas, é representado o tamanho da imagem/mapa de características resultante e o número de canais de cor respetivo.

No canto superior direito de cada retângulo é indicada a numeração da camada e no canto inferior direito a identificação do bloco onde está a atuar. As setas indicam o fluxo de dados entre as camadas, representando a passagem da informação de uma camada para a seguinte.

A secção *Details*, detalha os módulos *Conv*, *C2f*, *Bottleneck*, SPPF e *Detect* e apresenta uma tabela com as diferentes variantes do modelo (n, s, m, l, x), por ordem crescente de grau de complexidade, sendo a variante 'n' a mais simples e a 'x' a mais complexa. O parâmetro *depth_multiple* determina o número de camadas na rede, controlando a repetição do bloco *Bottleneck* no módulo *C2f*. Valores próximos de 1 deste parâmetro tornam o modelo mais 'profundo', aumentando porventura a sua capacidade de aprendizagem, mas também o custo computacional associado. O parâmetro *width_multiple* ajusta o número de canais em cada camada. Um valor inferior a 1 torna a rede mais estreita, e um valor superior a 1 alarga-a, exigindo mais memória e poder de processamento. O parâmetro *ratio* está relacionado com o fator de escala das imagens, sendo que modelos maiores processam imagens em resoluções mais altas, resultando em maior precisão na deteção de objetos, porém, novamente, tornando o modelo mais pesado computacionalmente.

Os retângulos azuis claro que representam a camada *Conv* consistem numa operação composta por três etapas: uma convolução (Conv2d), seguida de *Batch Normalization* (*BatchNorm2d*) e da função de ativação *Sigmoid Linear Unit* (SiLU).

A camada *Conv2d* diz respeito à operação de convolução propriamente dita. Esta função toma como *input* o tuplo (N, C_{in}, H, W), onde N corresponde ao tamanho do *batch*, C_{in} ao número de canais de entrada e H e W à altura e largura, respetivamente, do *input*. O *output* desta camada pode ser descrito pela expressão 2.2.

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out_j}, k) * \text{input}(N_i, k), \tag{2.2}$$

onde $*$ simboliza a operação de convolução entre o *kernel* e o canal de entrada.

Por exemplo, se quisermos aplicar 4 filtros diferentes do mesmo tamanho a um único canal de entrada, obtemos 4 canais de saída. Cada um destes canais corresponde ao resultado da aplicação de um dos filtros. O mesmo princípio aplica-se quando temos vários canais de entrada. Por exemplo, uma imagem codificada em RGB possui 3 canais: vermelho, verde e azul. Como ilustrado na Figura 2.9, se aplicarmos 4 filtros diferentes, obtemos 4 novos mapas de características (ou canais de saída). Cada canal de saída é então construído somando os resultados da aplicação de um filtro a todos os canais de entrada. Ou seja, para gerar um determinado canal de saída (out(N_i, C_{out_j})), aplicamos um filtro (weight(C_{out_j}, k)) a cada canal de entrada (input(N_i, k)) e somamos os resultados, adicionando geralmente um viés treinável (bias(C_{out_j})).

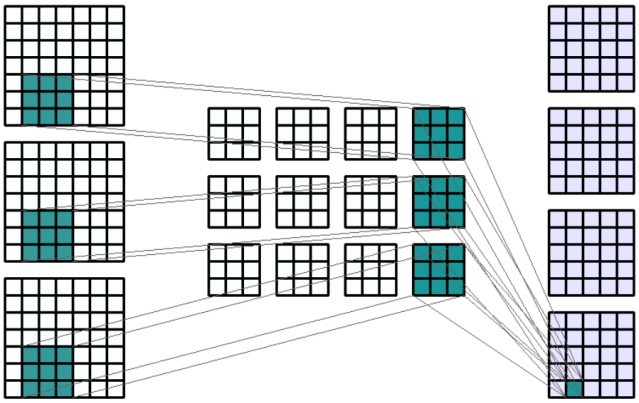


Figura 2.9: Exemplificação gráfica de uma convolução para três canais de entrada [21]

A dimensão do output (H_{out}, W_{out}) dos canais de saída, isto é, o número de colunas e linhas de cada canal de saída, pode ser calculado a partir das expressões 2.3 e 2.4.

$$H_{out} = \left\lfloor \frac{H_{in} + 2P_H - D_H(K_H - 1) - 1}{S_H} \right\rfloor + 1 \tag{2.3}$$

$$W_{out} = \left\lfloor \frac{H_{in} + 2P_W - D_W(K_W - 1) - 1}{S_W} \right\rfloor + 1 \tag{2.4}$$

P_H e P_W correspondem, respetivamente, ao valor do *padding* nas duas dimensões, altura e largura. Da mesma forma, S_H e S_W indicam as dimensões do ***stride***, K_H e K_W as dimensões do *kernel*, e D_H e D_W o fator de dilatação (por defeito, o seu valor é 1 e representa o intervalo entre cada píxel do *kernel* ao percorrer o canal de entrada durante a convolução).

Durante o treino, a distribuição das entradas de cada camada muda constantemente devido à atualização dos pesos nas camadas anteriores, um fenómeno conhecido por desvio covariante interno [22]. Essa variação pode tornar o treino mais lento, exigir uma inicialização cuidadosa dos pesos e dificultar a escolha de taxas de aprendizagem adequadas, já que a rede tem de estar constantemente a readaptar os seus pesos para lidar com novas entradas, tornando a aprendizagem instável. O parâmetro BatchNorm resolve este problema ao normalizar as entradas de cada camada, forçando-as a terem média zero e desvio padrão unitário, estabilizando o fluxo de informação através da rede. Idealmente, esta normalização seria feita considerando todo o conjunto de dados, no entanto, isso é muitas vezes impraticável devido à grande dimensão dos dados. Por isso, opta-se por realizar a normalização em *batches*, isto é, em subconjuntos do conjunto de dados.

É importante notar que se o *batch* for demasiado pequeno, a média e o desvio padrão tornam-se muito sensíveis a *outliers*. Com *batches* suficientemente grandes, as médias e desvios padrão tendem a ser mais estáveis e representativos.

A operação Batch Normalization aqui considerada é dada pela expressão 2.5.

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{V}[x] + \epsilon}} * \gamma + \beta \tag{2.5}$$

A média ($\mathrm{E}[x]$) e o desvio padrão ($\sqrt{\mathrm{V}[x]}$) são calculados para cada *batch* e para cada dimensão ou canal. Os parâmetros γ e β são parâmetros aprendíveis que permitem escalar e deslocar os valores normalizados, possibilitando assim algum controlo dos dados que entram na próxima camada (por exemplo, a percentagem de valores positivos e negativos que entram numa função de ativação como a SiLU). O ϵ é uma constante adicionada à variância do *batch* para estabilidade numérica. Esta normalização permite então a utilização de taxas de aprendizagem mais altas, reduz a sensibilidade à inicialização e atua também como uma forma de regularização [22].

Após a aplicação da operação *Batch Normalization*, é utilizada a função de ativação SiLU. Esta função, representada graficamente na Figura 2.10, é definida como o produto da função sigmoide pelo seu *input*, ou seja, $f(z) = z \times \frac{1}{1+e^{-z}}$, $z \in \mathbb{R}$. Esta função permite ao modelo aprender padrões mais complexos, através da introdução de não-linearidade. Para valores positivos da entrada, o seu comportamento aproxima-se de uma função linear. No entanto, para valores negativos, o *output* tende para 0, sem ser anulado abruptamente como no caso da função de ativação ReLU ($f(z) = max(0,z)$, $z \in \mathbb{R}$).

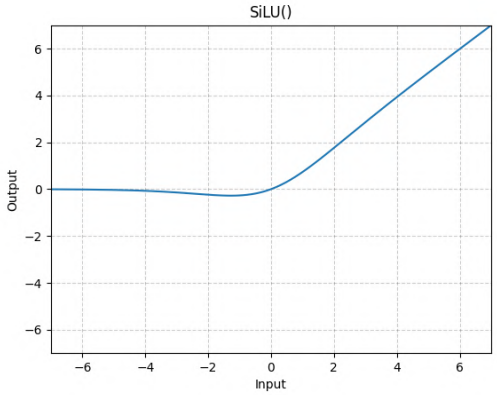


Figura 2.10: Representação gráfica da função de ativação SiLU [23]

Os retângulos a amarelo representam a camada *C2f* (CSP *Bottleneck* com 2 convoluções). Esta camada integra características de alto nível com informação contextual,

contribuindo para uma deteção mais precisa. Consiste num bloco convolucional, cujos mapas de características/canais de saída resultantes são divididos, pela função *Split*. Uma parte é encaminhada para o bloco *Bottleneck*, enquanto a outra segue diretamente para o bloco *Concat*. Como referido anteriormente, o número de blocos *Bottleneck* utilizados é definido pelo parâmetro *depth_multiple* do modelo. No final, o mapa de características proveniente do bloco *Bottleneck* é concatenado com o mapa anteriormente dividido, sendo esse resultado introduzido num bloco convolucional final.

Em cada bloco *Bottleneck*, o parâmetro *shortcut*, também conhecido por ligação residual, é uma ligação que permite contornar uma ou mais camadas da rede, facilitando a propagação do gradiente durante o treino, prevenindo o que se designa por problema de desvanecimento do gradiente. Este é um problema que surge durante o treino de redes neuronais profundas, especialmente em arquiteturas com muitas camadas. Ocorre quando os gradientes da função objetivo, em relação aos parâmetros (pesos) da rede, se tornam extremamente pequenos, dificultando a atualização eficaz dos pesos e, consequentemente, a aprendizagem da rede. Caso *shortcut=False*, a entrada é processada sequencialmente por dois blocos convolucionais.

A camada *Spatial Pyramid Pooling Fast* (SPPF), a azul escuro, foi concebida para acelerar o processamento da rede ao agrupar características de diferentes escalas, obtidas pelas operações MaxPool2d, num mapa de características de tamanho fixo. Como o nome indica, a operação MaxPool2d consiste na aplicação de *Max pooling* sobre um mapa de características bidimensional, processo este descrito anteriormente na secção 2.1.2. O propósito do SPPF é então fornecer uma representação em múltiplas escalas dos mapas de características de entrada. Ao aplicar *pooling* em diferentes escalas, o SPPF permite que o modelo capte características em vários níveis de abstração.

Na *Head*, são utilizadas as operações *Concat* e *Upsample*. A operação *Concat* tem como função combinar mapas de características provenientes de diferentes camadas. Por exemplo, a camada 11 concatena os mapas gerados pela camada *Upsample* 10 com os da camada *C2f* 6. A operação *Upsample*, por sua vez, é utilizada para combinar características de diferentes escalas e aumentar a resolução espacial dos mapas de características

Na camada final, *Detect*, a função de ativação sigmoide é utilizada para calcular a probabilidade de uma caixa delimitadora conter um objeto. Para as probabilidades de classe, é utilizada a função softmax (2.6).

$$softmax(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^{N} \exp(x_j)}, \tag{2.6}$$

onde N representa o número de classes.

O modelo YOLOv8 utiliza três funções de perda distintas para otimizar o seu desempenho: a função *Complete Intersection over Union* (CIoU), a função *Distribution Focal Loss* (DFL) e a função *Cross-Entropy*.

A função perda CIoU é aplicada na regressão das *bounding boxes* e tem por objetivo melhorar a precisão da localização dos objetos. Esta função mede a sobreposição entre as *bounding boxes* previstas e as referência (*ground truth*), tendo em conta a proporção e a distância entre os centros das caixas. Esta função é dada pela expressão 2.7.

$$\mathcal{L}_{CIoU} = 1 - IoU + \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2} + \alpha v, \tag{2.7}$$

onde $\alpha = \frac{v}{(1-IoU)+v}$ e $v = \frac{4}{\pi^2}(\arctan\frac{w^{gt}}{h^{gt}} - \arctan\frac{w}{h})^2$. A *Intersection over Union* (IoU) corresponde à razão entre a área de interseção das *bounding boxes* referência e prevista sobre a união delas (Figura 2.14); gt refere-se à *bounding box* referência (*ground truth*), w à largura e h à altura da *bounding box* prevista e w^{gt} à largura e h^{gt} à altura da *bounding box* referência; b e b^{gt} representam os pontos centrais das *bounding boxes* prevista e referência, respetivamente; $\rho(\cdot)$ corresponde à distância Euclidiana; e c ao comprimento da diagonal da menor *bounding box* que cobre ambas as caixas [24].

A função perda DFL tem por objetivo melhorar a previsão da localização das *bounding boxes* em condições em que o limite de um objeto detetado pode ser ambíguo ou difícil de distinguir, atribuindo mais peso às instâncias mais desafiantes. Em vez de prever uma caixa delimitadora fixa, a DFL estima uma distribuição de probabilidade para a localização das *bounding boxes*, representando as incertezas na previsão do limite do objeto.

A função perda Cross Entropy é aplicada na tarefa de classificação, quantificando a diferença entre as classes reais e as probabilidades previstas geradas pelo modelo.

O otimizador considerado pelo modelo é o AdamW. Este otimizador é um método de descida do gradiente estocástico baseado na estimativa adaptativa dos momentos de primeira e segunda ordem, com um mecanismo adicional de decaimento dos pesos (*weight decay*). O respetivo algoritmo encontra-se representado na Figura 2.11.

Como input são dados ao algoritmo os seguintes parâmetros: a taxa de aprendizagem (γ) (por omissão: 0.001); β_1 e β_2, coeficientes usados para calcular as médias móveis do

input : γ(lr), β_1, β_2(betas), θ_0(params), $f(\theta)$(objective), ϵ (epsilon)
λ(weight decay), *amsgrad*, *maximize*
initialize : $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$ (second moment), $\widehat{v_0}^{max} \leftarrow 0$

for $t = 1$ **to** ... **do**
  if *maximize* :
    $g_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$
  else
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  $\theta_t \leftarrow \theta_{t-1} - \gamma\lambda\theta_{t-1}$
  $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
  $\widehat{m_t} \leftarrow m_t / \big(1-\beta_1^t\big)$
  $\widehat{v_t} \leftarrow v_t / \big(1-\beta_2^t\big)$
  if *amsgrad*
    $\widehat{v_t}^{max} \leftarrow \max(\widehat{v_{t-1}}^{max}, \widehat{v_t})$
    $\theta_t \leftarrow \theta_t - \gamma\widehat{m_t} / \big(\sqrt{\widehat{v_t}^{max}} + \epsilon\big)$
  else
    $\theta_t \leftarrow \theta_t - \gamma\widehat{m_t} / \big(\sqrt{\widehat{v_t}} + \epsilon\big)$

return θ_t

Figura 2.11: Algoritmo do otimizador AdamW [25]

gradiente e do seu quadrado (por omissão: (β_1, β_2) = (0.9, 0.999)); o vetor de parâmetros iniciais ($\boldsymbol{\theta}_0$); a função perda; um épsilon (ϵ) para estabilidade numérica (por omissão: 10^{-8}); o termo de regularização *weight decay* (λ) (por omissão: 0.01); os termos *amsgrad* e *maximize* indicam, respetivamente, se deve ser utilizada a variante AMSGrad deste algoritmo (cuja principal diferença é a de manter o máximo de todos os valores v_t calculados até ao instante atual, t, e utilizar esse valor máximo para normalizar a média móvel dos gradientes, ao contrário do AdamW, que utiliza diretamente o valor no instante atual) e se a função perda deve ser maximizada em relação aos parâmetros, em vez de minimizada (em ambos os casos é assumido por omissão: *False*).

Os momentos de primeira e segunda ordem (m_0 e v_0) e a estimativa máxima do segundo momento ($\hat{v_0}^{max}$) são inicializados. Para cada iteração $t > 0$, calcula-se o gradiente da função objetivo no instante imediatamente anterior. No caso presente, o objetivo é minimizar a função objetivo, como tal calcula-se o gradiente normal. Em seguida, aplica-se o termo de regularização diretamente aos parâmetros ($\theta_t \leftarrow \theta_{t-1} - \gamma\lambda\theta_{t-1}$), penalizando-se assim mais os pesos que apresentaram amplitudes elevadas no passado nos parâmetros associados. Os momentos m_t e v_t são atualizados com base nas médias móveis dos gradientes e dos quadrados dos gradientes, respetivamente. Depois, corrigem-se esses momentos para o viés ($\hat{m}_t$ e $\hat{v}_t$), permitindo uma convergência mais precisa. Por fim, os parâmetros são atualizados com base na média móvel corrigida, normalizada pela soma da raiz do segundo momento corrigido e ϵ. O algoritmo termina

com o retorno dos parâmetros ótimos, θ_t.

2.1.4.2 Implementação

Relativamente à aplicação do modelo propriamente dita, inicialmente é importado o modelo YOLO da Ultralytics, pré-treinado no conjunto de dados *Common Objects in Context* (COCO) que contém cerca de 330 mil imagens de 80 categorias de objetos comuns distintos, como veículos, acessórios, animais, etc.

Para treinar o modelo no novo conjunto de imagens de treino pretendido, criou-se um ficheiro `.yaml`, especificando os caminhos para os conjuntos de dados de treino, validação e teste, bem como o número de classes e os respetivos nomes. Definiu-se ainda o número de épocas, ou seja, o número de vezes que o modelo vê todo o conjunto de treino. Ao longo do treino, foi possível observar que, por volta das 40 a 50 épocas, o desempenho no conjunto de validação estabilizava, o que indica que um número superior de épocas não traria melhorias significativas. Foi também definido o *batch size* e o tamanho da imagem de entrada pretendido (todas as imagens são redimensionadas para esta dimensão antes do processo de treino). Os valores escolhidos para estes dois parâmetros foram definidos de forma a equilibrar a capacidade computacional disponível com a qualidade do treino.

Na Tabela 2.1, encontram-se algumas das especificações do modelo consideradas.

Variante do modelo	YOLOv8s
Tamanho da imagem	640×640 pixéis
Número de épocas	50
Batch size	16
Learning rate	0.000526
Weight decay	0.0005

Tabela 2.1: Especificações do modelo YOLOv8

O *learning rate* considerado pelo modelo é obtido pela expressão `0.002 * 5 / (4 + nc)`, onde `nc` corresponde ao número de classes.

A variante do modelo escolhida foi a variante 's', que permite um equilíbrio razoável entre complexidade e custo computacional. De notar ainda que o GPU da máquina em que o projeto foi desenvolvido foi o NVIDIA GeForce RTX 4060.

Durante o treino do modelo, os pesos do modelo pré-treinado são então ajustados para melhorar a deteção dos novos objetos-alvo. Durante o treino, em cada época, o conjunto

de validação vai avaliando o desempenho do modelo, sem o usar diretamente para aprender, verificando apenas se o modelo está a aprender de forma eficaz. Após o treino, os pesos obtidos são guardados e posteriormente utilizados para realizar previsões sobre o conjunto de teste.

2.2. Métricas de avaliação

A avaliação dos modelos de deteção é essencial para garantir o seu desempenho e fiabilidade. Sem uma avaliação adequada, é impossível verificar a capacidade do modelo de identificar e localizar objetos corretamente, o que pode conduzir a conclusões potencialmente erradas.

Algumas das métricas mais comuns são a *Precision*, a *Recall*, a *Average Precision* (AP) e a *mean Average Precision* (mAP).

Para compreender o cálculo destas métricas as definições que se seguem são fundamentais. Na Figura 2.12, a interseção entre as referências (anotações) e as deteções do modelo, representam os Verdadeiros Positivos (VP), isto é, as deteções corretas feitas pelo modelo. Os elementos detetados pelo modelo que não correspondem a nenhuma referência são os Falsos Positivos (FP), ou seja, deteções incorretas. As referências que não foram detetadas pelo modelo, correspondem aos Falsos Negativos (FN). Os casos em que o modelo corretamente não realizou nenhuma deteção correspondem aos Verdadeiros Negativos (VN).

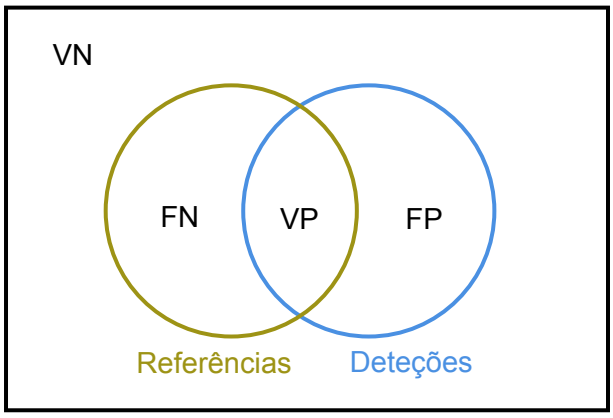


Figura 2.12: Diagrama ilustrativo das relações entre as referências (anotações) e as deteções feitas por um modelo

A *Precision*, ou Precisão, corresponde à razão entre o número de deteções corretas (VP) sobre o número total de deteções feitas pelo modelo (2.8), isto é, as deteções corretas e aquelas que o modelo detetou incorretamente (FP). É uma medida do quão bem o modelo é capaz de fazer deteções que de facto estão corretas. Um valor de precisão

elevado, próximo de 1, indica que o modelo é eficaz a evitar falsos positivos e fornece previsões positivas fiáveis.

$$Precision = \frac{VP}{VP + FP} \tag{2.8}$$

A *Recall*, ou Sensibilidade, é o quociente entre o número de deteções corretas e a soma entre o número de deteções corretas e o número de objetos que o modelo falhou em detetar (FN) (2.9). É uma medida da capacidade do modelo de capturar todos os objetos relevantes numa imagem. Um valor de *recall* elevada indica que o modelo identifica de forma eficaz a maioria dos objetos relevantes nos dados.

$$Recall = \frac{VP}{VP + FN} \tag{2.9}$$

A *F1-Score* é uma média harmónica da precisão e da *recall* (2.10). Esta métrica fornece uma medida equilibrada do desempenho do modelo, considerando tanto os falsos positivos como os falsos negativos.

$$\text{F1-Score} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{2.10}$$

A *Average Precision* corresponde à área sob a curva do gráfico da *Precision-Recall* (2.11), como exemplificado na Figura 2.13.

$$\text{AP} = \int_{r=0}^{1} p(r)\, dr \tag{2.11}$$

A *mean Average Precision* (mAP) corresponde à soma da *Average Precision* de cada classe dividida pelo número total de classes (2.12). Esta métrica é calculada mediante um limiar (*threshold*) determinado pela *Intersection over Union*, que consiste, como anteriormente referido, no quociente entre a área de interseção da caixa delimitadora referência e a prevista sobre a área da união das duas, como ilustrado na Figura 2.14.

$$\text{mAP} = \frac{1}{classes} \sum_{c \in classes} \text{AP}_c \tag{2.12}$$

A mAP50 corresponde à mAP com um *threshold* maior ou igual a 0.50. É uma medida da precisão do modelo considerando apenas as deteções mais ”fáceis”, isto é, só precisa de uma sobreposição razoável entre a caixa delimitadora prevista e a real. A mAP50-95 é a mAP calculada em diferentes *thresholds* de IoU, variando de 0.50 a 0.95. Dá uma

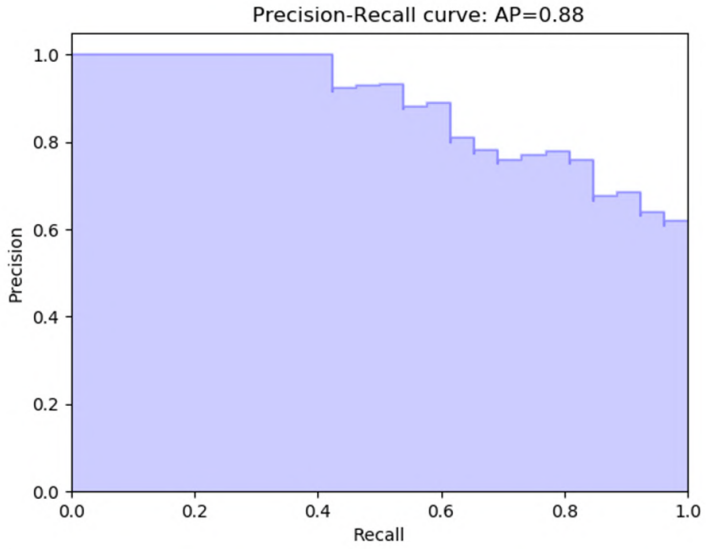


Figura 2.13: Exemplo de uma curva Precision-Recall [26]

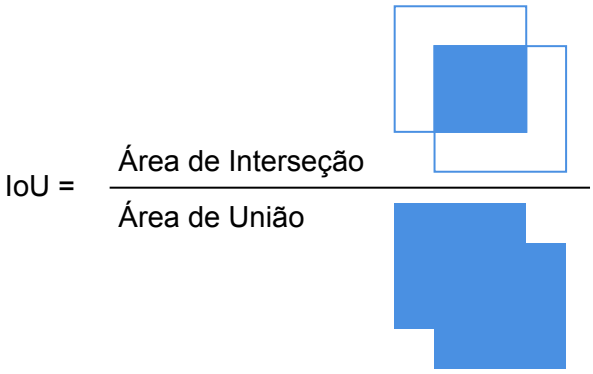


Figura 2.14: Representação gráfica da interseção sobre a união

ideia mais abrangente do desempenho do modelo em diferentes níveis de dificuldade de deteção.

Será então com base nestas métricas que se avaliará a performance do modelo YOLOv8s no conjunto de teste pretendido.

3. Conjunto de Dados

3.1. Descrição

Os dados utilizados neste projeto consistiram em imagens recolhidas por mais de 100 câmaras de disparo automático distribuídas pelas diversas passagens das concessões da Ascendi. O período de recolha estende-se de janeiro de 2022 a janeiro de 2025, totalizando a análise em mais de 500.000 imagens.

Inicialmente, foi realizada uma triagem manual das cerca de 500.000 imagens recolhidas, sendo selecionadas apenas aquelas que exibiam pelo menos um animal, procurando evitar a seleção de muitas imagens sequenciais. Após esta triagem, obteve-se um conjunto de 4933 imagens, com resoluções de imagem variando entre os 2688×1512 píxeis e os 3840×2160 píxeis. As imagens utilizadas encontram-se no formato JPG, em cores RGB com profundidade de 24 bits, ou seja, 8 bits por canal, com valores inteiros variando entre 0 e 255 para cada componente de cor.

Este conjunto inclui imagens capturadas tanto em condições diurnas quanto noturnas, a partir de diferentes perspetivas e distâncias, de 15 classes de animais distintos: *'Ave', 'Cão', 'Lebre/Coelho', 'Gato', 'Micromamífero', 'Raposa', 'Texugo', 'Fuinha', 'Saca-Rabos', 'Gineta', 'Lobo', 'Javali', 'Cervídeo', 'Esquilo', e 'Lontra'*. Na Figura 3.1 é apresentada uma imagem exemplo de cada uma destas classes. As classes 'Ave', 'Lebre/Coelho' 'Micromamífero' e 'Cervídeo', ao contrário das restantes, abrangem várias espécies dentro do mesmo grupo taxonómico uma vez que as semelhanças morfológicas entre estas pode limitar a capacidade discriminativa do modelo. No caso da classe 'Micromamífero', incluem-se ratos e ratazanas, enquanto na classe 'Cervídeo' incluem-se veados e corços. No Anexo A, encontram-se disponíveis mais exemplos destas imagens. O gráfico de barras da Figura 3.2 representa a frequência de cada uma das classes consideradas.

Para proceder à aplicação do modelo, o conjunto de dados foi dividido aleatoriamente nos seguintes três subconjuntos:

- **Treino**: 3453 imagens (70% do conjunto total de imagens)
- **Validação**: 986 imagens (20% do conjunto total de imagens)
- **Teste**: 494 imagens (10% do conjunto total de imagens)

Figura 3.1: Exemplos de imagens recolhidas de cada uma das classes do conjunto de dados pela ordem de leitura: *'Ave', 'Cão', 'Lebre/Coelho', 'Gato', 'Micromamífero', 'Raposa', 'Texugo', 'Fuinha', 'Saca-Rabos', 'Gineta', 'Lobo', 'Javali', 'Cervídeo', 'Esquilo'* e *'Lontra'*

O gráfico de barras apresentado na Figura 3.3 representa a frequência de cada uma das classes de animais nestes três conjuntos.

Como observado no gráfico da Figura 3.2, existe uma disparidade grande entre o número de ocorrências das classes *'Lontra'*, *'Esquilo'*, *'Lobo'* e as restantes. Esta discrepância pode comprometer o desempenho do modelo na deteção destas classes minoritárias. Serão então introduzidas técnicas de aumento de dados (*data augmentation*) a estas classes no conjunto de treino, testando-se à frente se de facto houve ou não uma melhoria na capacidade de previsão do modelo.

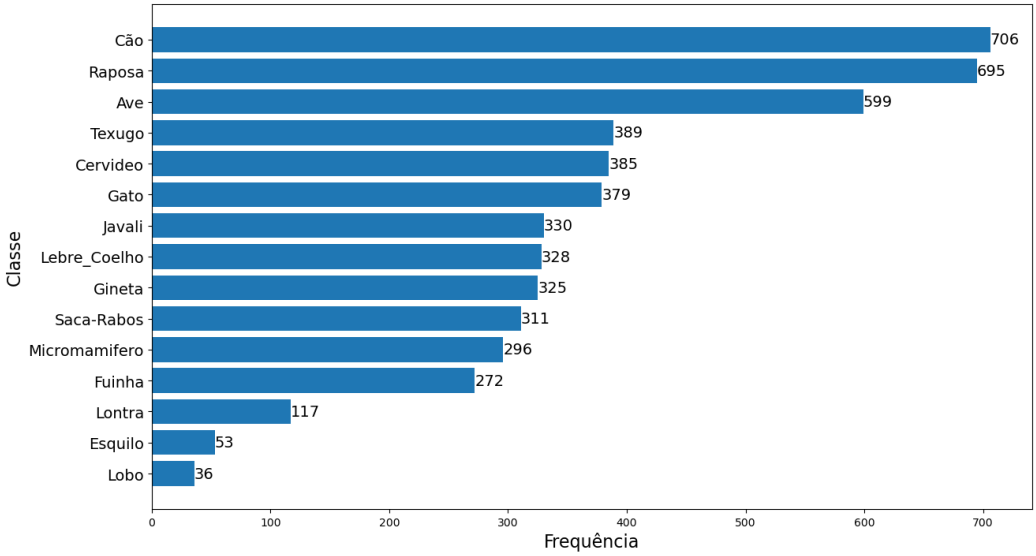


Figura 3.2: Número de ocorrências de cada animal no conjunto de dados

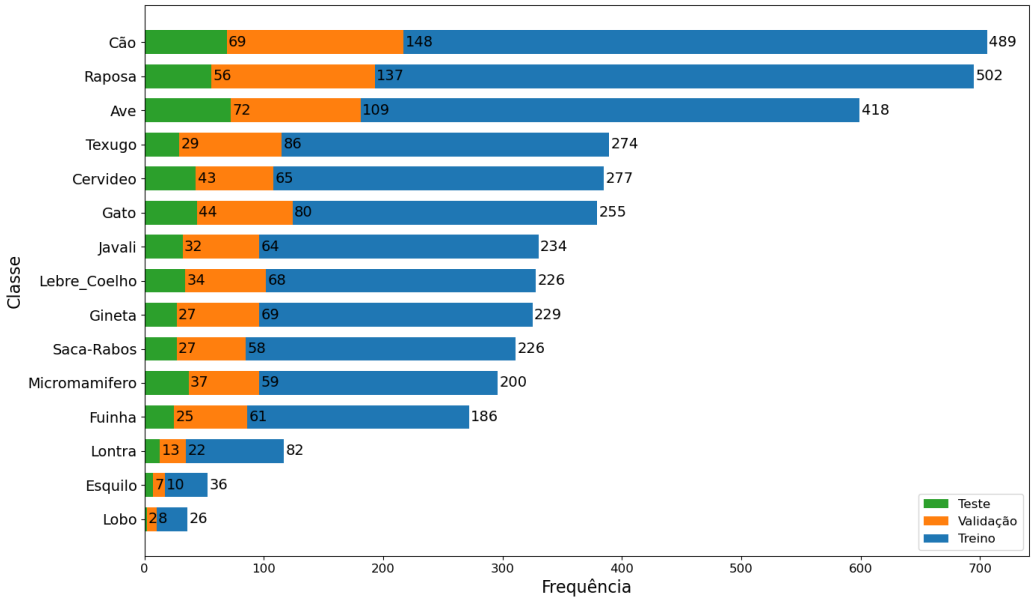


Figura 3.3: Divisão do número de ocorrências de cada animal no conjunto de treino, validação e teste

3.2. Pré-Processamento dos Dados

3.2.1 Anotações

Para o treino, validação e teste dos modelos de deteção é imprescindível a anotação das imagens, isto é, a identificação e delimitação dos objetos de interesse presentes em cada imagem, criando *bounding boxes* que indicam as respetivas localizações e categorias.

Para tal, recorreu-se à ferramenta `labelImg`, desenvolvida em *Python*, que permite precisamente a criação de *bounding boxes* ao redor dos objetos pretendidos, como ilustrado na Figura 3.4a, e o armazenamento dessas anotações em ficheiros XML no formato PASCAL VOC (Visual Object Classes). A Figura 3.4b apresenta um exemplo de um ficheiro deste tipo. As seguintes informações são incluídas: o nome da imagem, a sua localização no computador, as suas dimensões e número de canais de cor, as coordenadas da *bounding box* e a respetiva categoria atribuída (neste caso, um ”Gato”).

(a) Delimitação do objeto usando a ferramenta `labelImg`

```
<annotation>
  <folder>Gato</folder>
  <filename>01010370.JPG</filename>
  <path>C:\Users\ipimenta\Desktop\Imagens organizadas por classes\Gato\01010370.JPG</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>2688</width>
    <height>1512</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>Gato</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>1717</xmin>
      <ymin>729</ymin>
      <xmax>2163</xmax>
      <ymax>911</ymax>
    </bndbox>
  </object>
</annotation>
```

(b) Ficheiro XML, formato PASCAL VOC

Figura 3.4: Exemplo de identificação da região de interesse (*bounding box*) e armazenamento da anotação

Adicionalmente, para a implementação do modelo YOLO, foi necessário converter estes ficheiros num formato compatível com o modelo. Neste novo formato, cada classe é representada por um valor de 0 a 14 (0: ’Ave’, 1: ’Cão’, 2: ’Lebre_Coelho’, 3: ’Gato’, 4: ’Micromamifero’, 5: ’Raposa’, 6: ’Texugo’, 7: ’Fuinha’, 8: ’Saca-Rabos’, 9: ’Gineta’, 10: ’Lobo’, 11: ’Javali’, 12: ’Cervideo’, 13: ’Esquilo’, 14: ’Lontra’), como exemplificado na Figura 3.5. As quatro coordenadas limitadoras das *bounding boxes* são ajustadas para o centro normalizado (3.1) e a dimensão da imagem normalizada (3.2).

$$\begin{aligned} \text{x\_centro} &= \frac{\text{xmin} + \text{xmax}}{2 \cdot \text{image\_width}} \\ \text{y\_centro} &= \frac{\text{ymin} + \text{ymax}}{2 \cdot \text{image\_height}} \end{aligned} \tag{3.1}$$

$$\begin{aligned} \text{largura} &= \frac{\text{xmax} - \text{xmin}}{\text{image\_width}} \\ \text{altura} &= \frac{\text{ymax} - \text{ymin}}{\text{image\_height}} \end{aligned} \tag{3.2}$$

```
13 0.3609182098765432 0.5835905349794238 0.1886574074074074 0.14866255144032922
```

Figura 3.5: Exemplo de uma anotação no formato YOLO. O primeiro valor numérico representa a identificação da classe, seguido pelas coordenadas normalizadas do centro da *bounding box* e respetiva largura e altura

3.2.2 *Data Augmentation*

Para contornar o número escasso de ocorrências das classes *'Esquilo'*, *'Lobo'* e *'Lontra'* que, como já referido, podem comprometer a performance do modelo, aplicaram-se as seguintes técnicas de *data augmentation* a cada uma das imagens destas classes no conjunto de treino:

- **Reflexão**: A nova imagem é refletida em relação ao eixo vertical da imagem original. As anotações associadas às *bounding boxes* são ajustadas, sendo invertidas as coordenadas xmin e xmax da imagem original (xmin_new = image_width - xmax, xmax_new = image_width - xmin). Figura 3.6b.

- **Adição de ruído gaussiano**: A nova imagem é gerada por adição de ruído aleatório, com distribuição normal, diretamente aos valores dos píxeis da imagem original. Para os parâmetros do ruído considerou-se média nula e desvio padrão igual a 100. Figura 3.6c.

- **Alteração da luminosidade:** As novas imagens são geradas com o objetivo de simular diferentes condições de luminosidade, adicionado-se um valor constante positivo para simular um aumento da luminosidade (Figura 3.6d) e um negativo para simular uma diminuição da luminosidade (Figura 3.6e),

- **Desfocagem**: A nova imagem é gerada pela desfocagem da imagem original, tendo-se recorrido para isso a um filtro gaussiano através da função `GaussianBlur` do pacote OpenCV do Python. Este tipo de filtro recebe como parâmetros a dimensão da janela pretendida e um valor para o desvio padrão. Neste caso, escolheu-se um filtro de dimensão 25×25 e desvio padrão 0, suavizando desta forma os detalhes da imagem. Figura 3.6f.

O gráfico de barras da Figura 3.7, representa o número de ocorrências de cada animal no conjunto de treino após aplicação destas técnicas de *data augmentation* às classes minoritárias. À classe *'Esquilo'* foram acrescentadas mais 180 imagens, à classe *'Lontra'* 410 e à classe *'Lobo'* 130. Às restantes classes não foram aplicadas estas operações, mantendo-se o número de ocorrências original, resultando assim num total de 4158 imagens de treino.

(a) Imagem original (b) Imagem refletida (c) Ruído gaussiano

(d) Aumento de luminosidade (e) Redução de luminosidade (f) Imagem desfocada

Figura 3.6: Exemplo de aplicação das técnicas de *data augmentation*

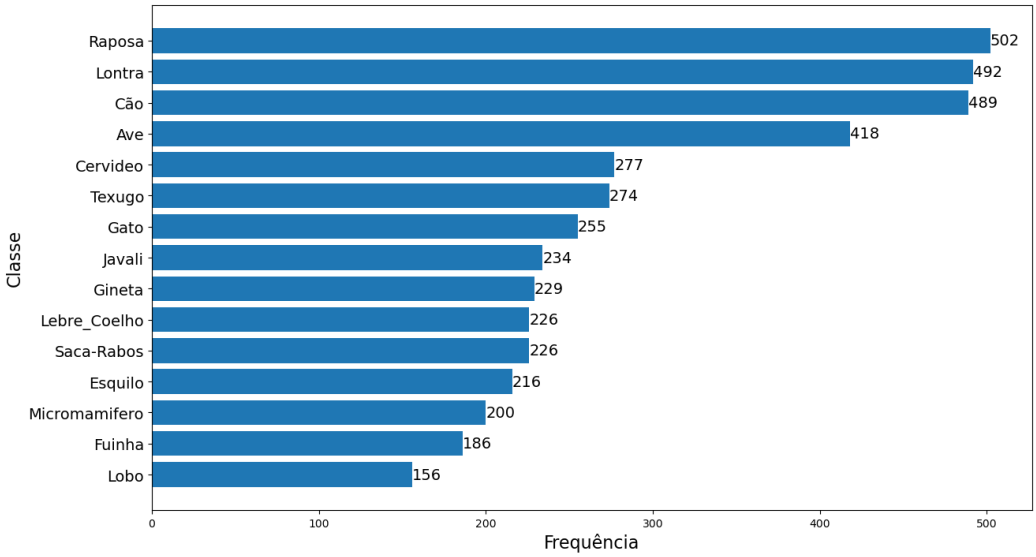


Figura 3.7: Número de ocorrências de cada animal no conjunto de treino, após aplicação das técnicas de *data augmentation*

3.2.3 Resumo

Neste capítulo, descreveu-se o processo de recolha e preparação dos dados utilizados neste projeto.

Com o conjunto de dados preparado e anotado, o próximo passo será treinar e avaliar o modelo de deteção de animais utilizando duas abordagens distintas: uma com o conjunto de treino original e outra com o conjunto de treino aumentado através das técnicas de *data augmentation* referidas.

Esta comparação permitirá analisar o impacto do aumento de dados na performance do modelo. Serão avaliadas métricas como a *Precision*, a *Recall* e a mAP para compreender se as técnicas aplicadas melhoram efetivamente a capacidade preditiva do modelo.

Será também ainda testado o desempenho do modelo num conjunto de dados independente do anteriormente considerado, contendo imagens consideradas irrelevantes, isto é, sem a presença dos objetos de interesse e recolhidas num contexto ambiental distinto, com o objeto de entender de que forma podem os resultados ser afetados.

4. Resultados

Neste capítulo, apresentam-se os resultados da avaliação do desempenho do modelo, considerando primeiro o conjunto de dados de teste definido inicialmente, sob duas condições, com e sem aplicação de *data augmentation* no conjunto de treino, e em segundo um novo conjunto de dados, recolhido por novas câmaras instaladas numa concessão da Ascendi.

4.1. Interface Gráfica do Utilizador

No entanto, antes de se passar aos resultados, é apresentada a interface gráfica que foi criada para tornar a interação com o modelo mais intuitiva e acessível para utilização futura na Ascendi. Esta inteface foi implementada a partir da biblioteca `PySide6` do Python. A Figura 4.1 mostra a janela inicial que é apresentada ao utilizador.

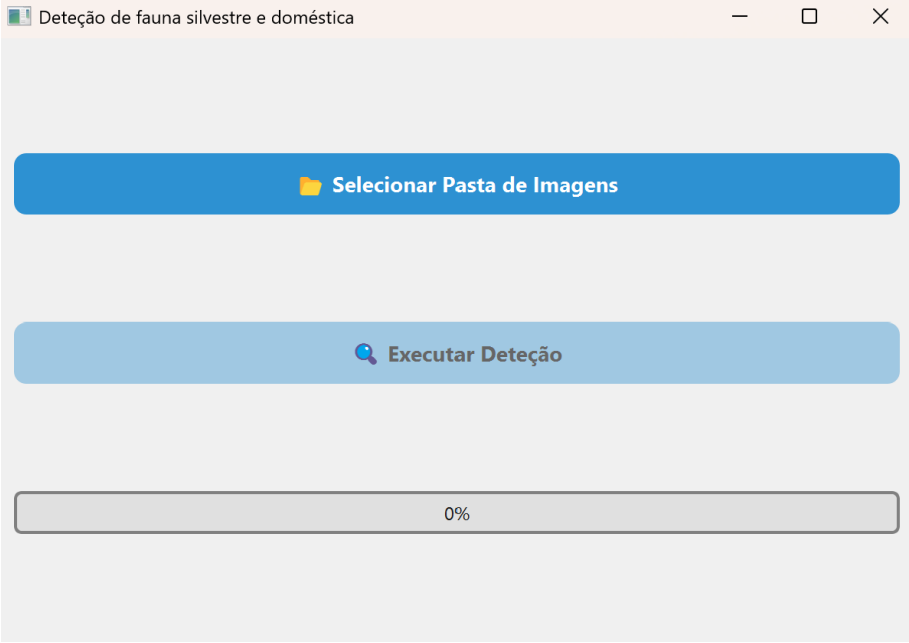


Figura 4.1: Interface gráfica desenvolvida com recurso ao pacote `PySide6` do Python

Sempre que os dados de uma câmara são recolhidos e transferidos para o computador, esta interface permite ao utilizador selecionar a pasta onde as imagens se encontram. Em seguida, ao clicar no botão 'Executar Deteção', o sistema inicia automaticamente o processo de inferência, recorrendo aos pesos ótimos obtidos pelo modelo previamente treinado. O progresso do processo de inferência é apresentado através de uma barra de

progresso e, no final, é exibida uma mensagem indicando 'Deteção Concluída'. As imagens onde o modelo não deteta objetos de interesse são automaticamente eliminadas e aquelas em que deteta são guardadas com a respetiva *bounding box* e classe associada numa nova pasta.

Adicionalmente, é gerado um ficheiro `.csv` contendo o nome, a data e a hora em que cada imagem foi capturada, assim como a respetiva classificação do animal/animais detetado(s), como ilustrado na Figura 4.2a. É também criado um ficheiro `.txt` com a contagem total por classe dos animais detetados (Figura 4.2b). O objetivo é que estes ficheiros sejam posteriormente incorporados no Sistema de Informação Geográfico (SIG) da Ascendi, permitindo a visualização georreferenciada da distribuição das espécies.

	A	B	C	D	E
1	Imagem,Data/Hora,Animais Detetados				
2	01010371.JPG,2022:01:01 01:30:09,"CÃ£o, Gato"				
3	01010588.JPG,2025:01:01 11:06:59,Saca-Rabos				
4	01020062.JPG,2022:01:02 18:05:04,Lebre_Coelho				
5	01030028.JPG,2022:01:03 07:06:39,Raposa				
6	01040149.JPG,2025:01:04 21:10:56,Micromamifero				
7	01050052.JPG,2022:01:05 11:52:01,Micromamifero				
8	01050158.JPG,2025:01:05 07:48:28,Lontra				
9	01060069.JPG,2022:01:06 09:51:39,Texugo				
10	01060145.JPG,2022:01:06 11:15:02,Lebre_Coelho				
11	01060152.JPG,2022:01:06 13:47:36,Texugo				
12	01060159.JPG,2022:01:06 16:04:08,Lebre_Coelho				
13	01060183.JPG,2020:01:06 12:19:27,Raposa				
14	01080028.JPG,2024:01:08 14:57:42,"CÃ£o, Gato"				
15	01080043.JPG,2022:01:08 13:49:08,Raposa				
16	01080085.JPG,2022:01:08 08:40:43,Micromamifero				
17	01090004.JPG,2024:01:09 07:56:30,Ave				
18	01090005.JPG,2024:01:09 07:56:30,Ave				
19	01090006.JPG,2024:01:09 07:56:31,Ave				
20	01090071.JPG,2022:01:09 12:33:30,Raposa				
21	01090121.JPG,2022:01:09 12:27:09,Fuinha				
22	01100056.JPG,2024:01:10 11:54:25,CÃ£o				

(a) Exemplo do ficheiro .csv criado

```
Lebre_Coelho: 41
Gato: 53
Raposa: 58
Esquilo: 5
Saca-Rabos: 29
Cão: 76
Micromamifero: 41
Lontra: 15
Texugo: 33
Ave: 66
Fuinha: 27
Gineta: 26
Lobo: 2
Cervideo: 40
Javali: 31
```

(b) Exemplo do ficheiro .txt criado

Figura 4.2: Representação das deteções feitas pelo modelo para um conjunto de dados, em formatos de texto .csv e .txt

4.2. Desempenho do modelo

Efetuou-se uma análise da capacidade de generalização do modelo ao conjunto de teste. Na Tabela 4.1 são apresentados os resultados das métricas descritas no capítulo 2.2 para o conjunto de teste, obtidas a partir dos dois treinos considerados, sem e com a introdução de *data augmentation*. É de notar que o treino do modelo sem *data augmentation* demorou aproximadamente 1h05min a correr enquanto que com *data augmentation* o tempo foi aproximadamente 1h20min.

	Precision	***Recall***	***F1-Score***	**mAP50**	**mAP50-95**
Sem *data augmentation*	$\sim 90.79\%$	$\sim 88.25\%$	$\sim 89.50\%$	$\sim 94.29\%$	$\sim 77.78\%$
Com *data augmentation*	$\sim 92.05\%$	$\sim 85.18\%$	$\sim 88.50\%$	$\sim 93.43\%$	$\sim 78.34\%$

Tabela 4.1: Resultados obtidos para as métricas de performance do modelo sem e com a introdução de *data augmentation* no conjunto de treino

Os resultados obtidos indicam um bom desempenho do modelo no conjunto de teste em ambas as abordagens de treino, com e sem a aplicação de *data augmentation*, apresentando valores bastante próximos entre si. Não se verificaram melhorias notáveis nas métricas com a introdução de *data augmentation*. Observou-se uma ligeira melhoria na *precision* e no mAP50-95, com aumentos de aproximadamente 1.26% e 0.56%, respetivamente. No entanto, nas restantes métricas, *recall*, *F1-Score* e mAP50, os resultados foram ligeiramente inferiores. Estes dados sugerem que o impacto das técnicas utilizadas, com base nestas métricas, foi marginal e não contribuiu de forma expressiva para o aumento da performance global do modelo. Como tal, para os resultados que se seguem considerar-se-à apenas o modelo com o conjunto de treino original, sem introdução de *data augmentation*.

Na Figura 4.11 encontra-se representada a matriz de confusão respetiva. As linhas correspondem às classes previstas pelo modelo, enquanto as colunas representam as classes verdadeiras. A diagonal principal indica os casos corretamente classificados (Verdadeiros Positivos).

A análise da matriz revela que a maioria das classes foi corretamente identificada pelo modelo. No entanto, observa-se uma frequência de falsos positivos associados à classe *background*, o que indica que o modelo, por vezes, identifica incorretamente regiões sem animais como contendo objetos relevantes. Verifica-se igualmente que algumas classes como a 'Ave', 'Micromamífero' e 'Saca-Rabos' foram pontualmente classificadas como *background*, evidenciando a ocorrência de falsos negativos.

Na Figura 4.4 estão representadas as curvas da *Precision-Recall* para cada uma das classes. A partir destas, como referido anteriormente, é possível obter a AP para cada classe e consequentemente a mAP, estando estes valores indicados na legenda à direita do gráfico.

Estes resultados demonstram que o modelo apresenta um desempenho global elevado com curvas que mantêm valores de precisão elevados ao longo de quase todo o intervalo de *recall*, o que indica uma boa capacidade do modelo para equilibrar a identificação

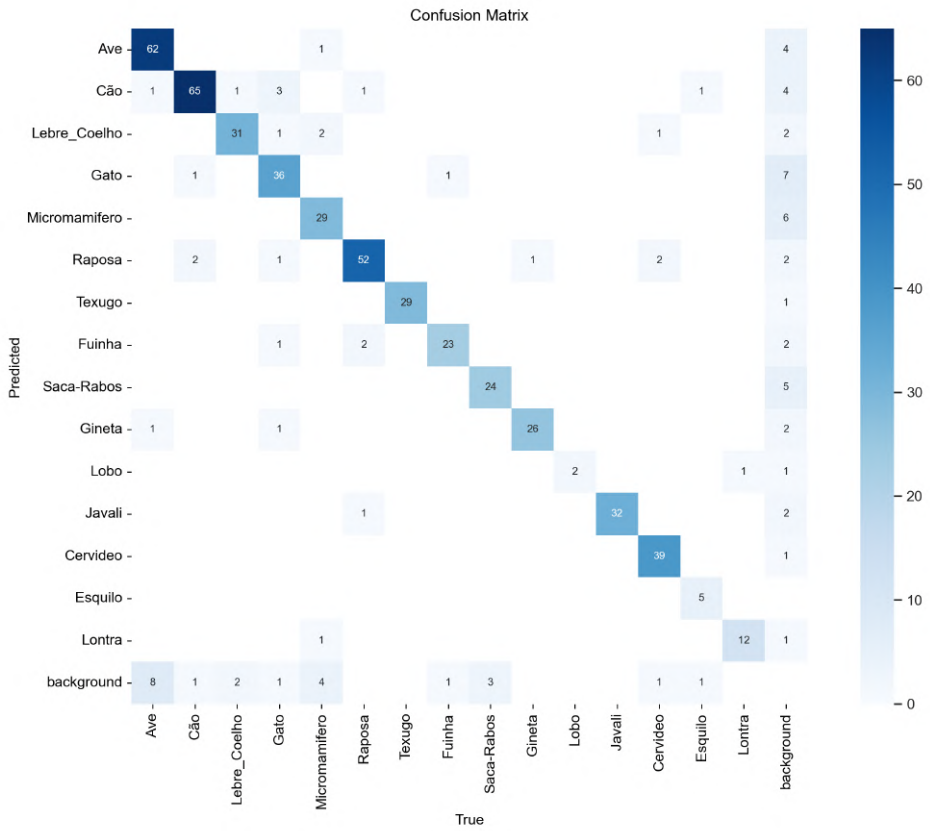


Figura 4.3: Matriz de Confusão obtida para o conjunto de teste sem introdução de *Data Augmentation* no treino

correta dos objetos, minimizando tanto os falsos negativos como os falsos positivos.

As AP das classes 'Esquilo' e 'Micromamífero' registaram os valores mais baixos (0.818 e 0.877, respetivamente), o que no caso do 'Esquilo' poderá estar associado à sua menor representatividade no conjunto de treino, e no caso do 'Micromamifero' a características visuais mais ambíguas.

A Figura 4.5 apresenta a evolução das funções de perda, tanto para o conjunto de treino como para o de validação e a Figura 4.6 a evolução das principais métricas de desempenho, ao longo das 50 épocas de treino do modelo. É possível observar-se uma evolução consistente das funções de perda e das métricas de desempenho ao longo deste processo. As três funções de perda consideradas pelo modelo demonstram uma tendência decrescente relativamente estável tanto no conjunto de treino como no de validação, o que indica uma aprendizagem progressiva e sem sinais evidentes de *overfitting*. As métricas de desempenho, apresentam uma evolução crescente, especialmente nas primeiras 20 épocas. Estes resultados sugerem que o modelo foi capaz de aprender de forma eficaz a identificar e classificar os objetos presentes nas imagens, alcançando um bom equilíbrio entre capacidade de generalização e performance.

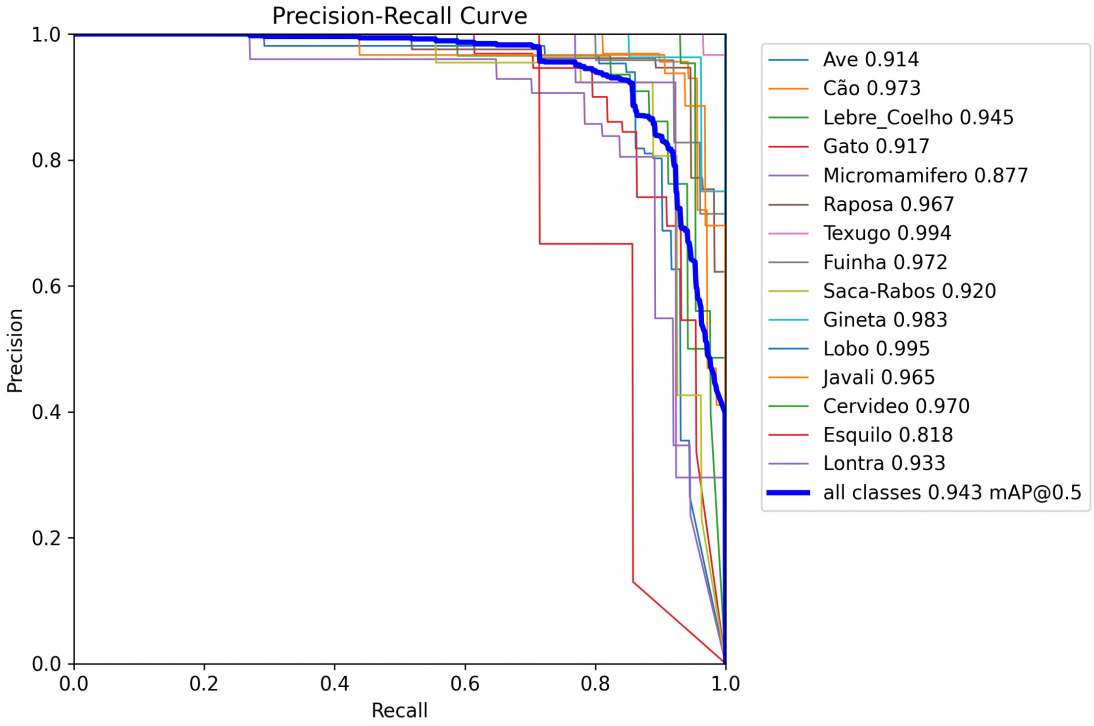


Figura 4.4: Curva *Precision-Recall* para as 15 classes consideradas

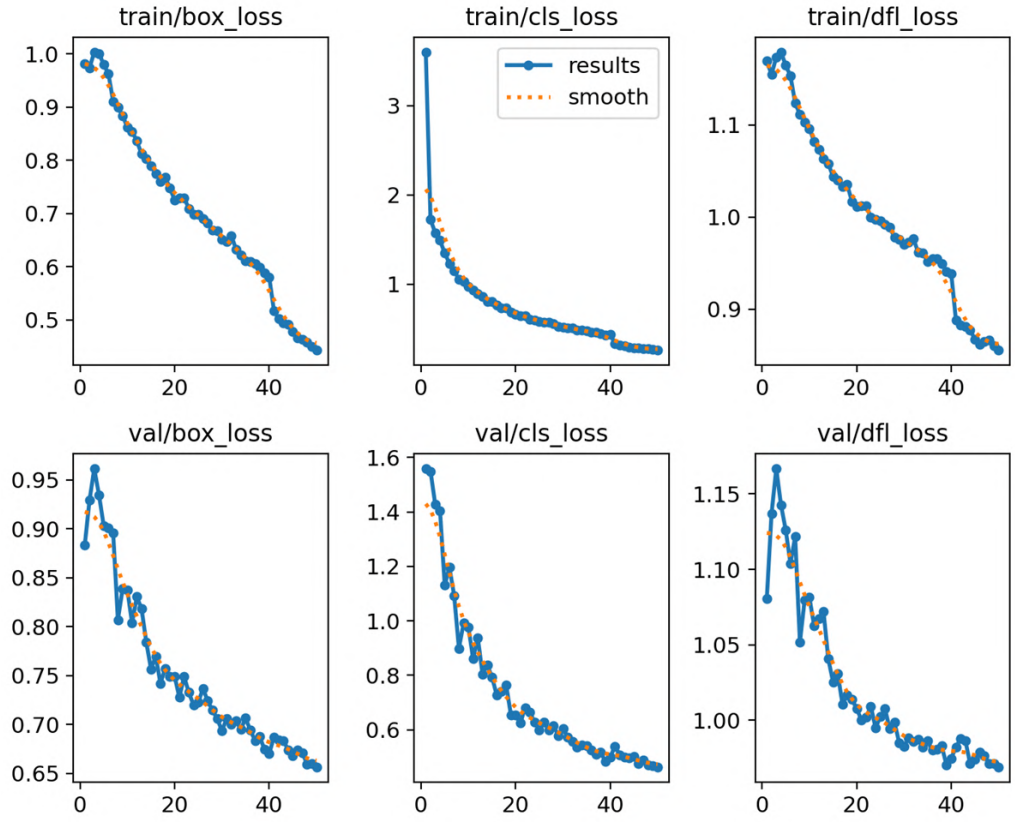


Figura 4.5: Gráficos representativos da evolução das três funções de perda (*box loss*(CIoU), *classification loss* (*Cross-Entropy*) e DFL) ao longo das 50 épocas no conjunto de treino e validação

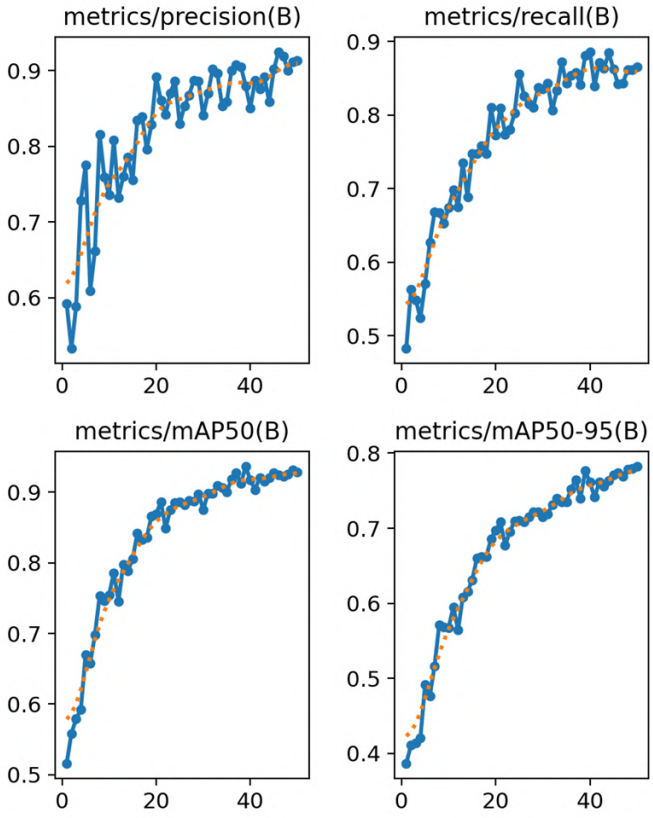


Figura 4.6: Gráficos representativos da evolução das métricas de desempenho (*precision*, *recall*, mAP50 e mAP50-95) ao longo das 50 épocas

4.3. Aplicação a novas imagens recolhidas por câmaras na concessão BLA

As imagens do conjunto de teste anterior resultaram de uma divisão aleatória do conjunto de dados original, representando 10% deste. Ora, muitas imagens do conjunto inicial são sequenciais, isto é, foram capturadas em momentos próximos e em condições semelhantes, o que poderá influenciar favoravelmente os resultados. Além disso, todas as imagens deste conjunto continham sempre pelo menos um animal, não estando o modelo a ser testado para os casos em que de facto não existe qualquer animal na imagem. Como tal, o modelo foi testado em novos conjuntos de dados de teste, totalmente independentes do anterior. Estes dados foram recolhidos por nove câmaras diferentes, não utilizadas anteriormente, distribuídas pela concessão Beiras Litoral e Alta. Este conjunto de imagens não foi sujeito a qualquer tipo de pré-processamento, procurando assim reproduzir-se a forma como o modelo se destina a ser utilizado em contexto operacional.

Um dos primeiros problemas identificados foi a deteção de uma nova espécie de animais não contemplada no treino do modelo: a vaca. Como tal, todas as classificações obtidas pelo modelo para este animal falharam. No entanto, a localização destes animais

nas imagens foi quase sempre certeira. As classificações que o modelo mais frequentemente atribuiu a esta espécie foram as classes 'Cão', 'Cervídeo', 'Javali' ou 'Raposa', como exemplificado na Figura 4.7. Estes animais poderão de facto assemelhar-se fisionomicamente à vaca em determinadas posições ou ângulos.

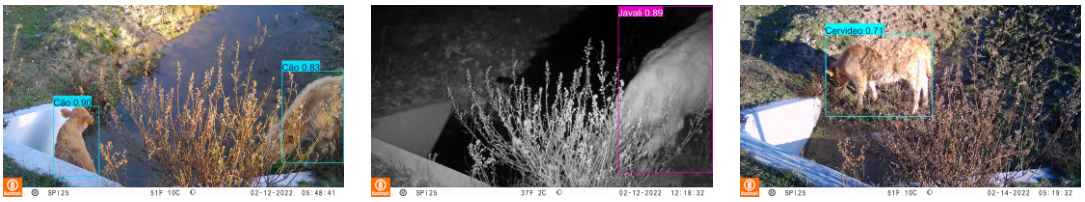


Figura 4.7: Exemplos de deteção de uma espécie desconhecida (vaca) pelo modelo

Outros objetos desconhecidos, como veículos ou pessoas, são também detetados e alvo de tentativas de classificação pelo modelo. São apresentados alguns exemplo de imagens com pessoas e veículos na Figura 4.8.

Figura 4.8: Exemplos de deteção de outros objetos desconhecidos pelo modelo, como pessoas e veículos

Uma vez que o modelo foi pré-treinado no conjunto de dados COCO, que contém uma grande variedade de objetos, como pessoas e veículos [27], é plausível que algumas destas deteções observadas se devam ao conhecimento previamente adquirido durante essa fase de treino. Tal poderá explicar a identificação ocasional de objetos que, embora não estejam presentes no conjunto de dados utilizado neste trabalho, fazem parte das classes originalmente aprendidas pelo modelo.

Além destes casos, um falso positivo frequente foi a classificação de folhagem ou determinadas formas de sombra como 'Ave'. Este erro poderá ser justificado pela semelhança entre a fisionomia das asas das aves e a forma das folhas. Na Figura 4.9 encontram-se alguns exemplos destas classificações erradas.

Figura 4.9: Exemplos de deteção de folhagem classificadas erradamente como 'Ave'

Outros falsos positivos de natureza aparentemente mais aleatória, como os representados na Figura 4.10, foram também registados. No caso da deteção de uma garrafa de água apresentada no canto inferior direito da Figura 4.10, poderá aplicar-se o argumento do conhecimento prévio adquirido pelo modelo durante o pré-treino no conjunto de dados COCO, que inclui a classe genérica *bottle*. Nos restantes casos, os falsos positivos podem justificar-se por semelhanças visuais, como a presença de ramos com forma e orientação semelhantes às pernas dos cervídeos, ou a tonalidade avermelhada de certas rochas que se assemelha à coloração típica da pelagem da raposa.

Figura 4.10: Exemplos de falsos positivos de diversos tipos

Apesar do número elevado de falsos positivos, verificou-se uma redução substancial do número de imagens sem objetos de interesse a detetar, e como tal, de tempo despendido na verificação manual. No entanto, na tentativa de reduzir alguns destes falsos positivos detetados, foram criadas duas novas classes: 'Pessoa' e 'Veiculo', sendo o modelo treinado novamente. Este procedimento revelou-se eficaz. Como tal, uma nova condição foi integrada na interface desenvolvido: sempre que é detetada uma pessoa ou um veículo, as imagens correspondentes são automaticamente eliminadas, evitando assim a sua posterior análise.

Nas Figuras 4.12 a 4.14, apresentam-se as previsões feitas pelo modelo para uma determinada câmara da concessão BLA. Esta câmara capturou 772 imagens no espaço de 12 dias. Deste conjunto de imagens, o modelo descartou 700, não identificando nenhum animal nelas presente.

Como se pode verificar pela observação da matriz de confusão da Figura 4.11, 23 ocorrências da classe 'Ave' e duas da classe 'Javali' foram rejeitadas pelo modelo como objetos de interesse. No caso das aves, como referido anteriormente, isto poderá ser justificado pela sua semelhança com o fundo, que neste caso é essencialmente composto por folhagem. No caso dos javalis, especialmente em condições noturnas, eles poderão surgir com contornos pouco nítidos, confundindo-se também com o ambiente de fundo.

Das 72 previsões que o modelo detetou como relevantes, classificou corretamente 64 e erradamente 8. Destas 8 classificações erradas, 3 foram falsos positivos associados à classe 'Ave', uma 'Raposa' foi classificada como 'Fuinha', duas 'Fuinhas' como 'Texugo', uma 'Gineta' como 'Fuinha' e uma 'Ave' como 'Esquilo'.

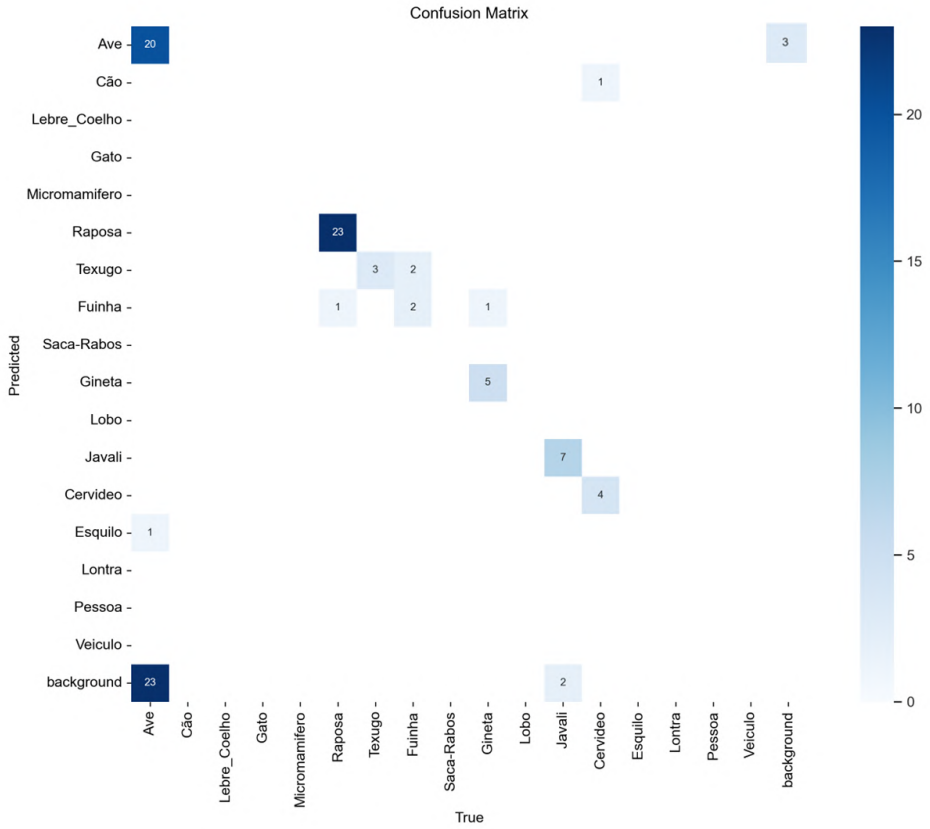


Figura 4.11: Matriz de Confusão para o conjunto novo de imagens recolhidas

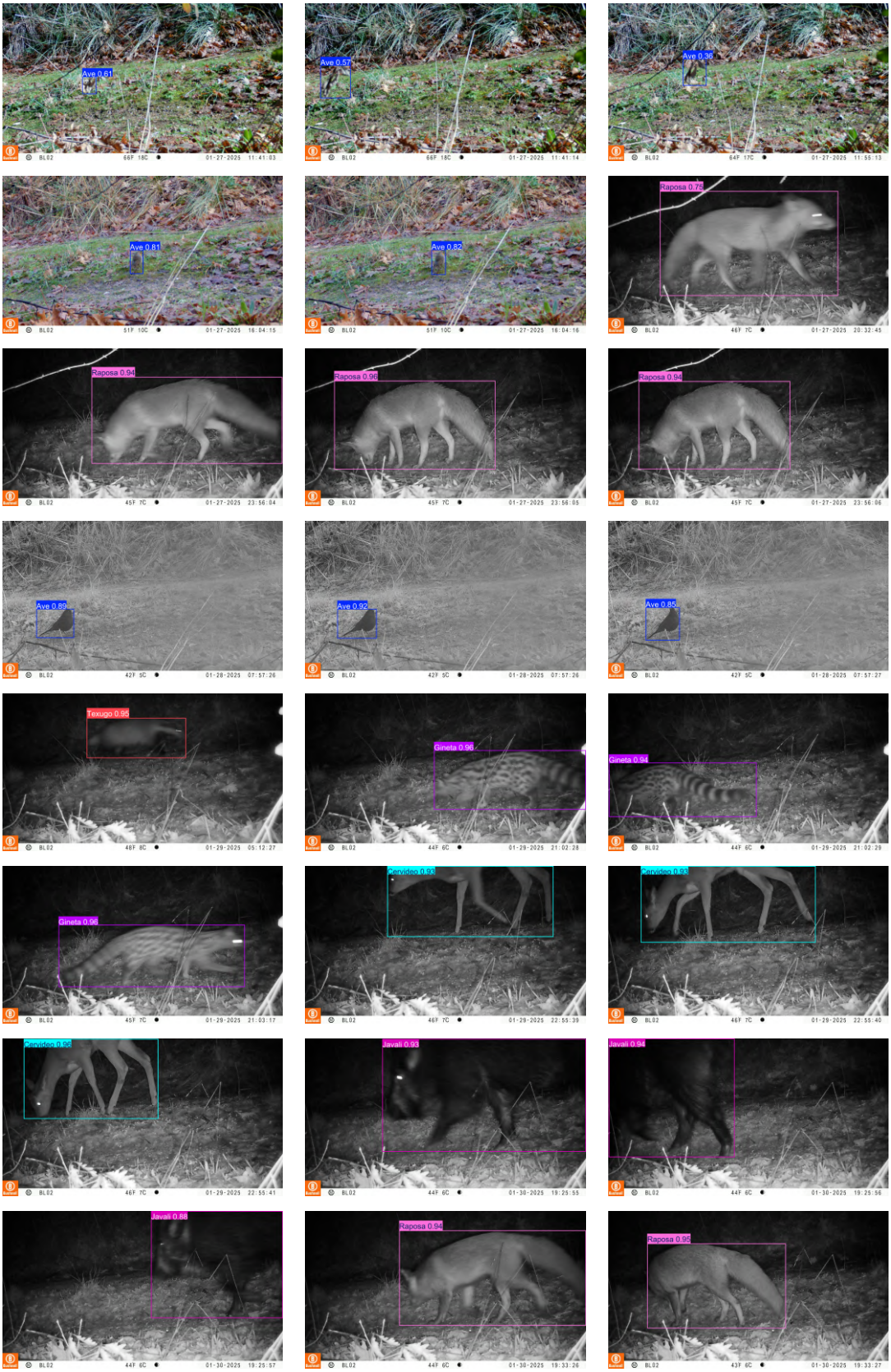


Figura 4.12: Previsões do modelo para as imagens recolhidas por uma câmara instalada na concessão BLA

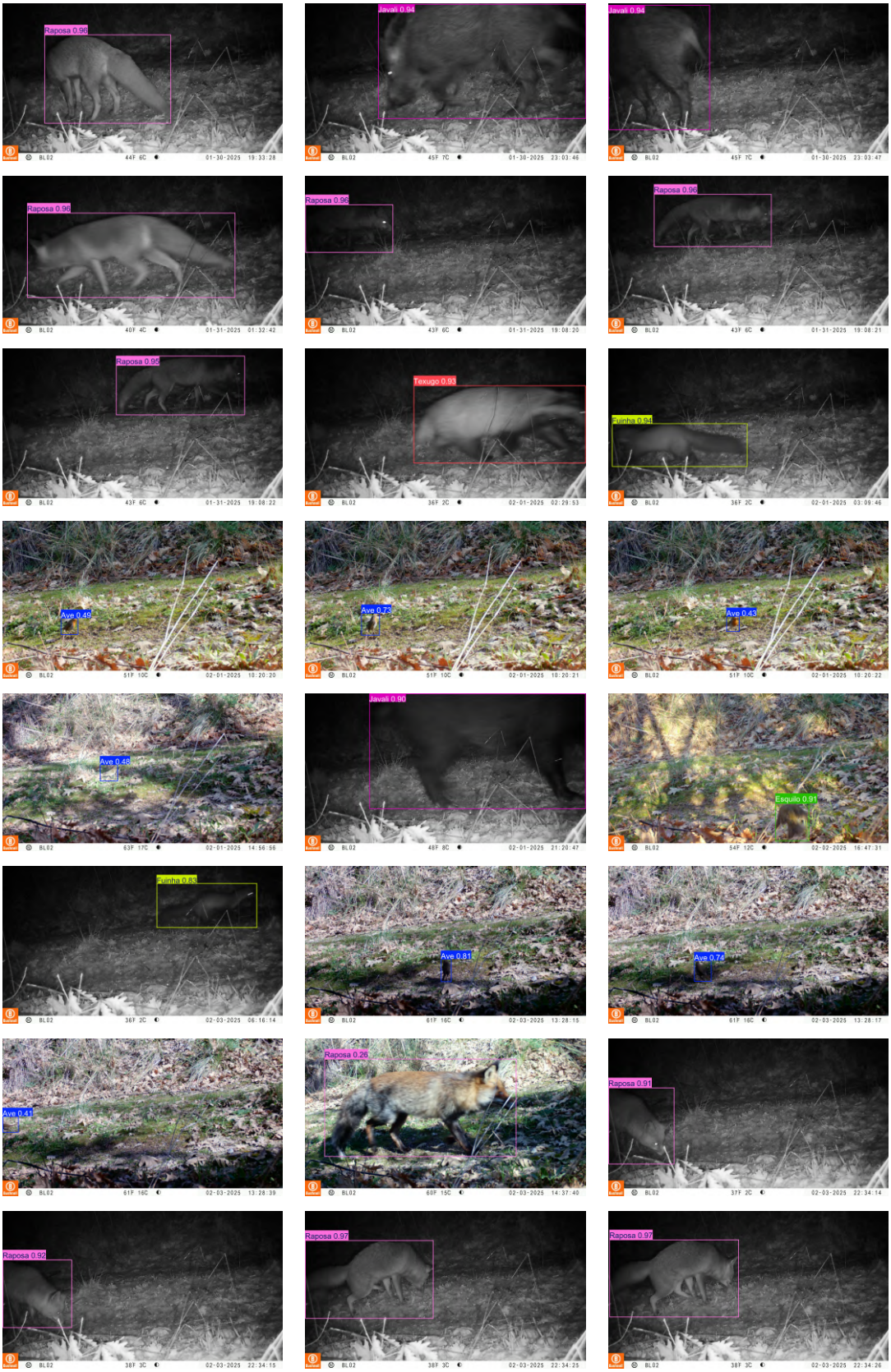


Figura 4.13: Previsões do modelo (cont.)

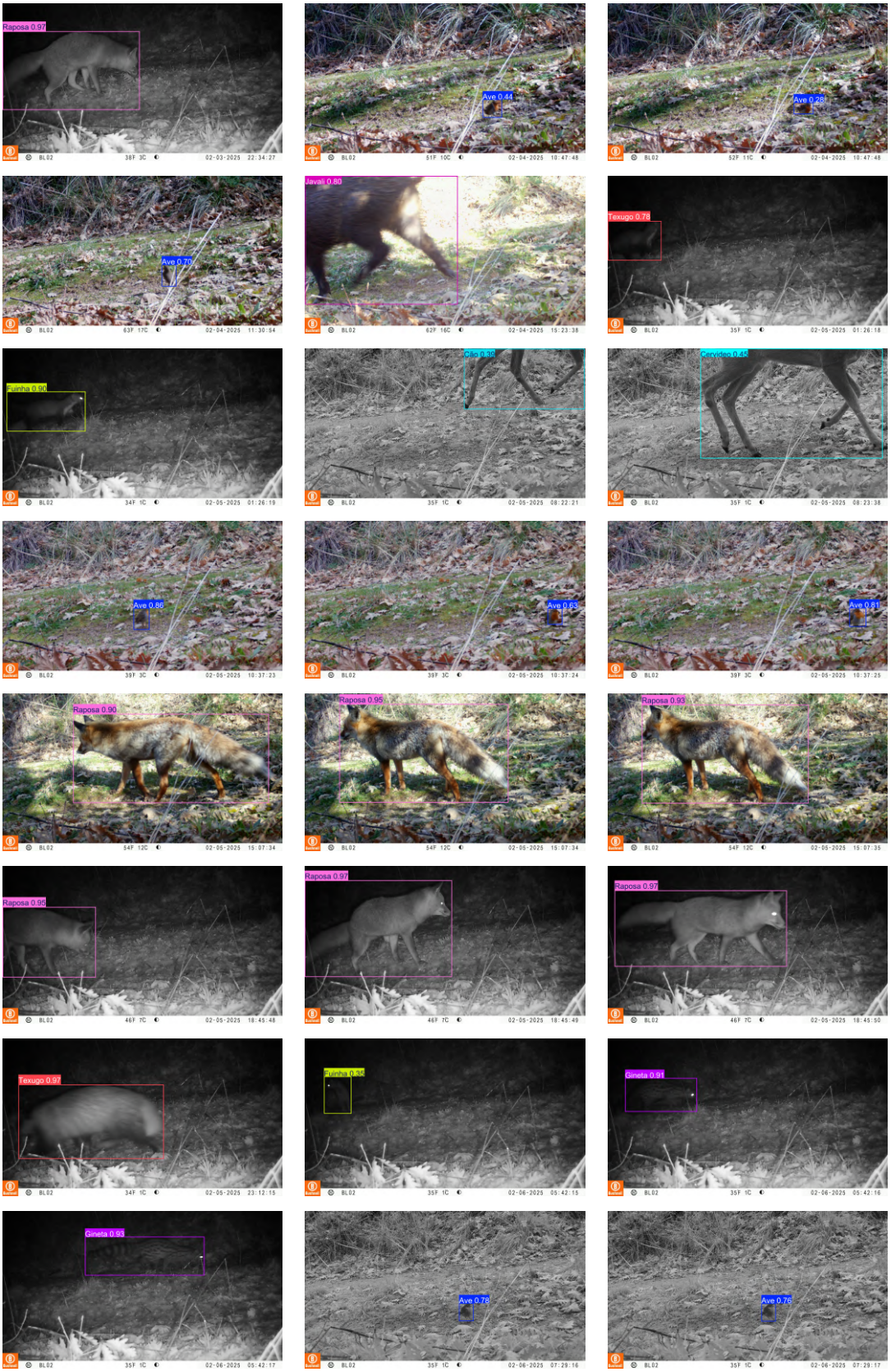


Figura 4.14: Previsões do modelo (cont.)

No Anexo B encontram-se previsões do modelo feitas para uma outra câmara desta concessão, posicionada num ponto de vista mais elevado. Esta câmara capturou 1000 imagens durante um período de um mês. Deste conjunto, o modelo identificou apenas 76 imagens contendo objetos de interesse. A classe mais frequentemente detetada nesta região foi o ’Cão’, seguida pela ’Raposa’ e algumas ’Fuinhas’. Neste conjunto, as confusões mais recorrentes por parte do modelo verificaram-se entre as classes ’Cão’ e ’Raposa’, dada as semelhanças visuais entre ambas as espécies à distância a que os animais se encontram da câmara. Houve também algumas classificações de cães como ’Gato’.

5. Conclusão

Este trabalho demonstrou o potencial da aplicação de modelos de visão computacional, nomeadamente o YOLOv8, na deteção e classificação automática de fauna silvestre e doméstica a partir de imagens recolhidas por câmaras de disparo automático. A abordagem seguida revelou resultados encorajadores, mesmo perante desafios consideráveis, como a variabilidade das condições de iluminação, a presença de vegetação densa e a semelhança morfológica entre algumas espécies.

Não obstante o desempenho alcançado, foram identificadas algumas limitações, que apontam para diversas possibilidades de melhoria futura. A ampliação do conjunto de treino, com a incorporação de um maior número de imagens por classe, poderá reforçar a capacidade discriminativa do modelo, particularmente nas classes menos representadas. A colocação das câmaras em posições mais favoráveis, minimizando ao máximo a obstrução causada pela vegetação envolvente, poderá igualmente contribuir para a melhoria da qualidade das deteções. A adição de novas classes de animais, mesmo que pouco relevantes para a monitorização da fauna pretendida, como a vaca, a ovelha e a cabra, permitirá uma representação mais abrangente da fauna observada no terreno, reduzindo a ocorrência de falsos positivos associados à ausência dessas categorias no treino original. Eventualmente a alteração de alguns dos hiperparâmetros e configurações do modelo poderá contribuir para uma melhoria do seu desempenho.

Em suma, os resultados obtidos confirmam a viabilidade da utilização de modelos de inteligência artificial como ferramenta de apoio à monitorização da biodiversidade, oferecendo um contributo relevante para a Ascendi na gestão ecológica das suas infraestruturas.

Bibliografia

[1] H. Shi, T. Shi, Z. Yang, Z. Wang, F. Han, and C. Wang, “Effect of roads on ecological corridors used for wildlife movement in a natural heritage site,” *Sustainability*, vol. 10, p. 2725, 08 2018.

[2] A. M. Turing, “Computing machinery and intelligence,” *Mind*, vol. LIX, no. 236, pp. 433–460, 10 1950.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[4] M. Haenlein and A. Kaplan, “A brief history of artificial intelligence: On the past, present, and future of artificial intelligence,” *California Management Review*, vol. 61, 07 2019.

[5] W. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity.” *Bulletin of Mathematical Biophysics*, vol. 5, p. 115–133, 1943.

[6] “A logical neuron,” https://marlin.life.utsa.edu/mcculloch-and-pitts.html, accessed: 2025-02-21.

[7] G. Piccinini, “The first computational theory of mind and brain: A close look at mcculloch and pitts’s “logical calculus of ideas immanent in nervous activity”,” *Synthese*, vol. 141, 08 2004.

[8] J. Sinai, “A foray into artificial intelligence, with the help of math, history and python,” https://github.com/JontySinai/PythonAI/blob/master/Assets/mcp-neuron-graph.png, 2017.

[9] B. Drukarch, H. A. Holland, M. Velichkov, J. J. Geurts, P. Voorn, G. Glas, and H. W. de Regt, “Thinking about the nerve impulse: A critical analysis of the electricity-centered conception of nerve excitability,” *Progress in Neurobiology*, vol. 169, pp. 172–185, 2018.

[10] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–44, 05 2015.

[11] B. I. Muhamad Yani and C. Setiningsih, “Application of transfer learning using convolutional neural network method for early detection of terry’s nail,” *Journal of Physics: Conference Series*, vol. 1201, no. 1, p. 012052, may 2019.

[12] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” 2014.

[13] R. Girshick, “Fast r-cnn,” 2015.

[14] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” 2016.

[15] C. Carl, F. Schönfeld, I. Profft, A. Klamm, and D. Landgraf, “Automated detection of european wild mammal species in camera trap images with an existing and pre-trained computer vision model,” *European Journal of Wildlife Research*, vol. 66, 2020.

[16] F. Simões, C. Bouveyron, and F. Precioso, “Deepwild: Wildlife identification, localisation and estimation on camera trap videos using deep learning,” *Ecological Informatics*, vol. 75, p. 102095, 2023.

[17] M. Tan, W. Chao, J.-K. Cheng, M. Zhou, Y. Ma, X. Jiang, J. Ge, L. Yu, and L. Feng, “Animal detection and classification from camera trap images using different mainstream object detection architectures,” *Animals*, 08 2022.

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” 2016.

[19] G. Jocher, A. Chaurasia, and J. Qiu, “Ultralytics yolov8,” 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[20] R. King, “Brief summary of yolov8 model structure,” https://github.com/ultralytics/ultralytics/issues/189, 2023.

[21] “Conv2d: Finally understand what happens in the forward pass,” https://medium.com/data-science/conv2d-to-finally-understand-what-happens-in-the-forward-pass-1bbaafb0b148, accessed: 2025-04-14.

[22] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” 2015.

[23] P. Contributors, “Sigmoid linear unit function,” https://pytorch.org/docs/stable/generated/torch.nn.SiLU.html.

[24] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, “Distance-iou loss: Faster and better learning for bounding box regression,” *CoRR*, vol. abs/1911.08287, 2019.

[25] P. Contributors, “Adamw,” https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html.

[26] Scikit-learn, “Precision-recall,” https://scikit-learn.org/0.21/auto_examples/model_selection/plot_precision_recall.html.

[27] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, “Microsoft coco: Common objects in context,” 2015.

Apêndice

A. Amostra de imagens recolhidas, por classe

Figura 1: Imagens recolhidas da classe ’Ave’

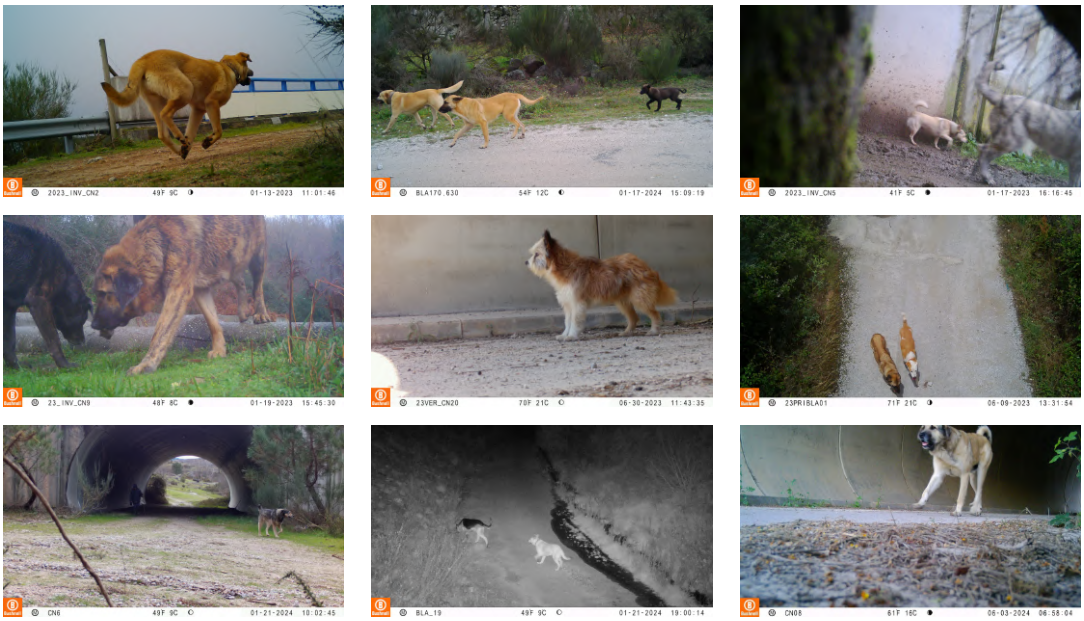


Figura 2: Imagens recolhidas da classe ’Cão’

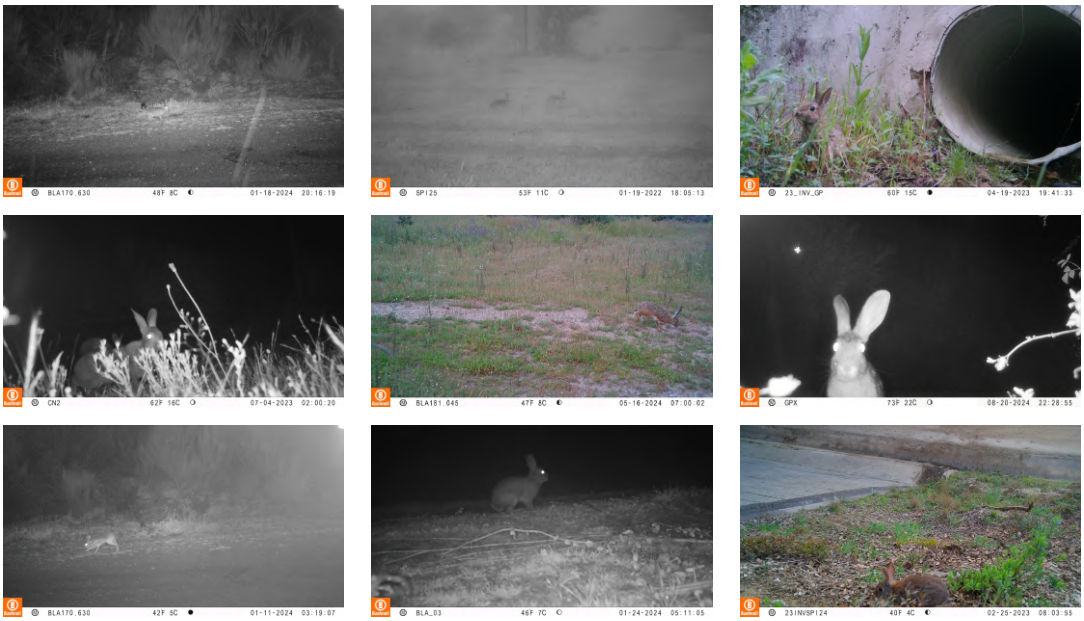


Figura 3: Imagens recolhidas da classe ’Lebre_Coelho’

Figura 4: Imagens recolhidas da classe ’Gato’

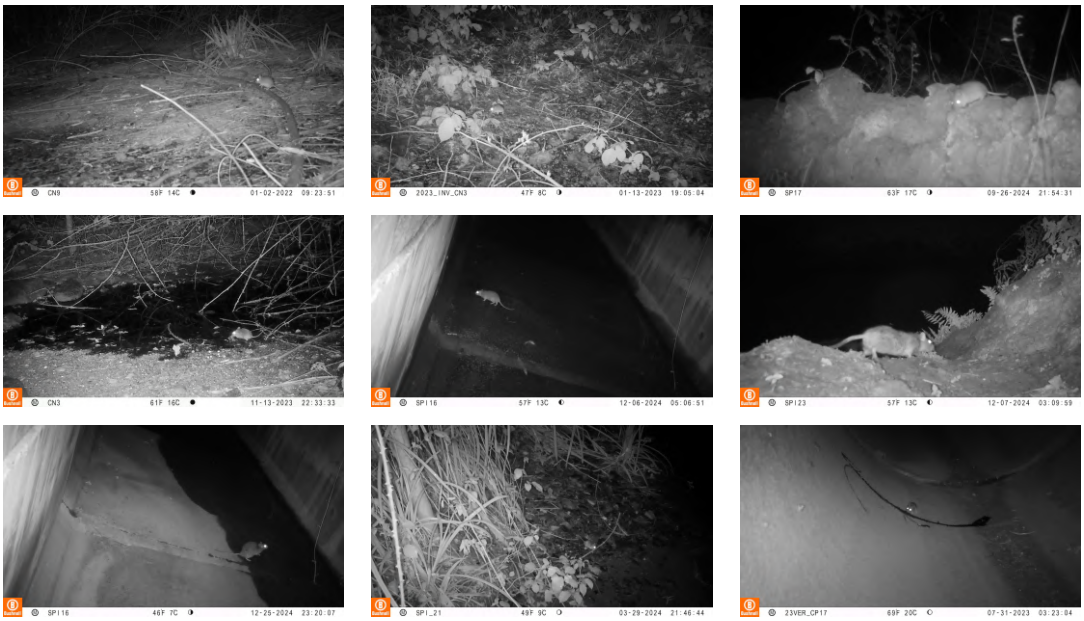


Figura 5: Imagens recolhidas da classe ’Micromamifero’

Figura 6: Imagens recolhidas da classe ’Raposa’

Figura 7: Imagens recolhidas da classe ’Texugo’

Figura 8: Imagens recolhidas da classe ’Fuinha’

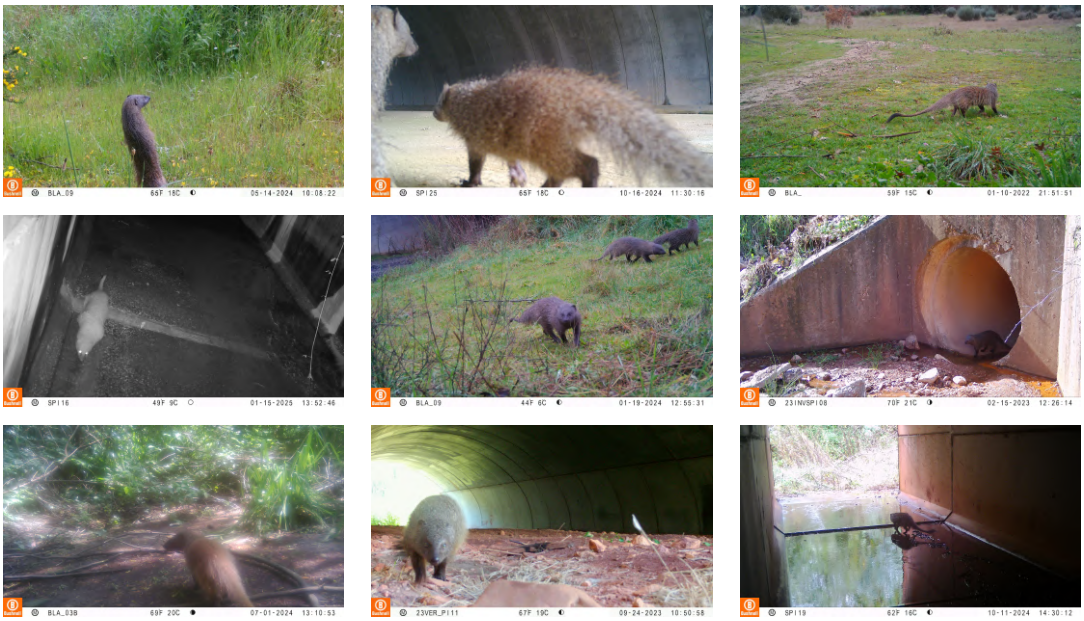


Figura 9: Imagens recolhidas da classe ’Saca-Rabos’

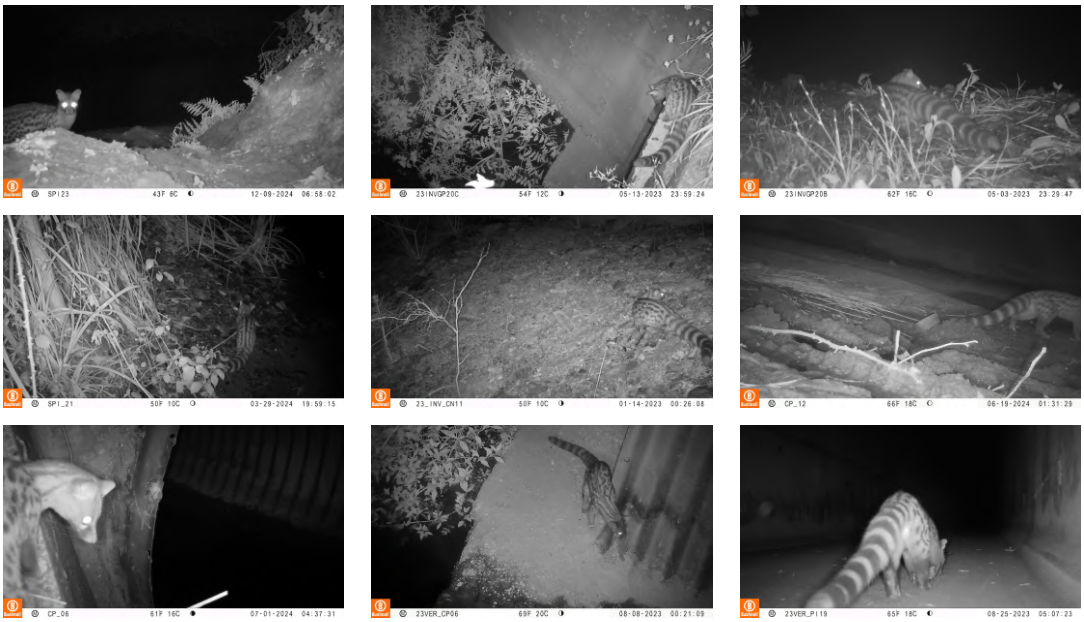


Figura 10: Imagens recolhidas da classe ’Gineta’

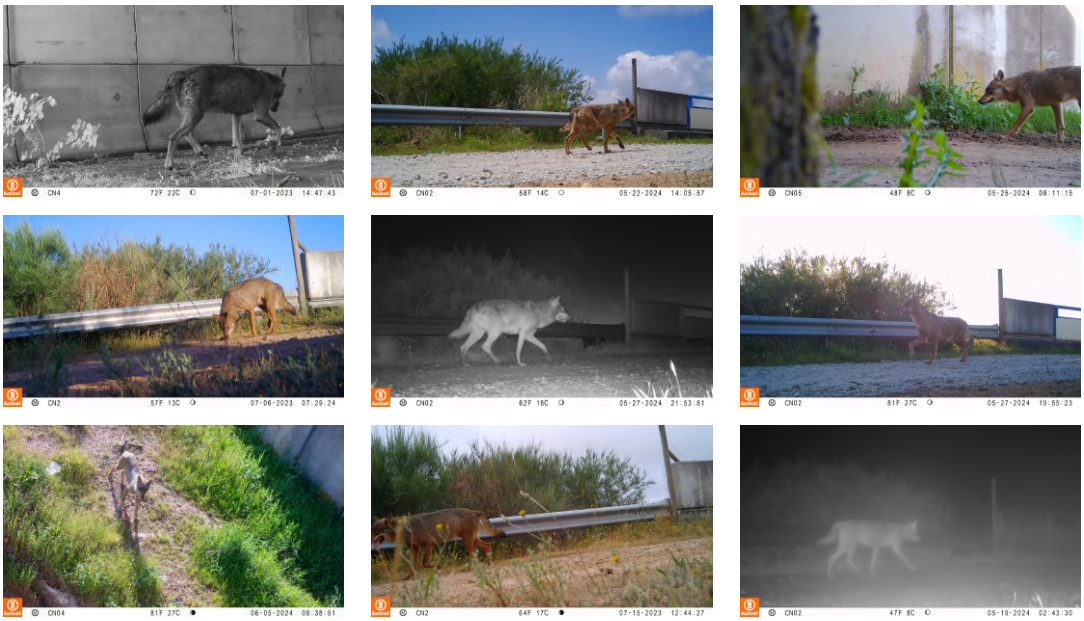


Figura 11: Imagens recolhidas da classe ’Lobo’

Figura 12: Imagens recolhidas da classe ’Javali’

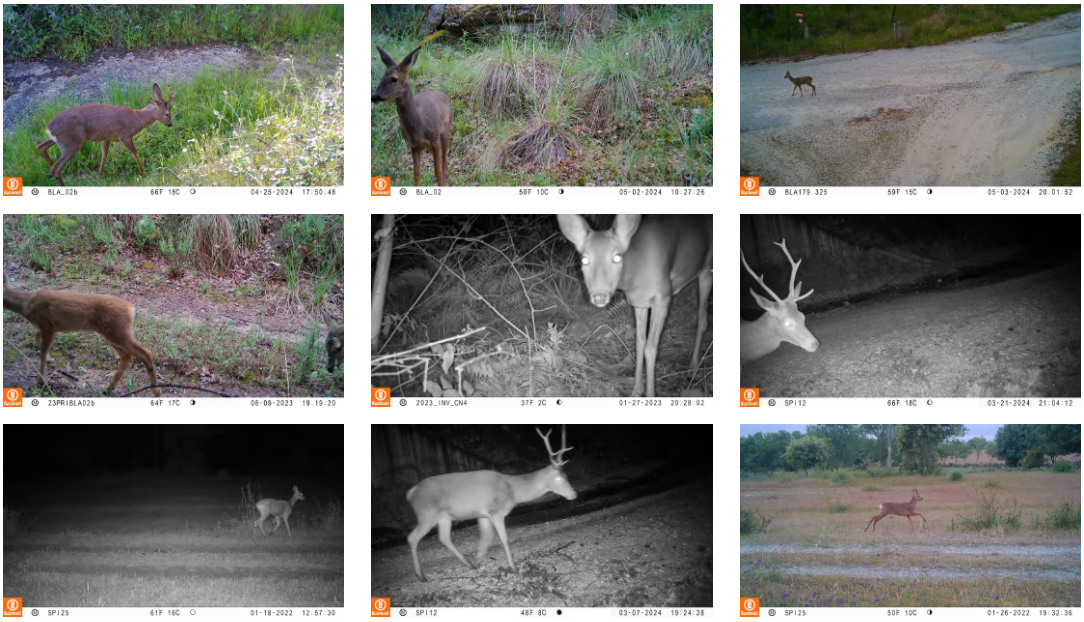


Figura 13: Imagens recolhidas da classe ’Cervídeo’

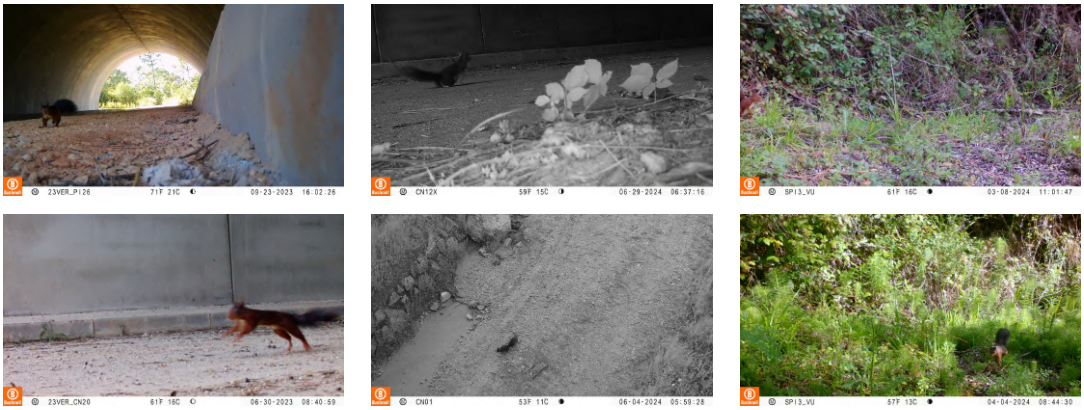
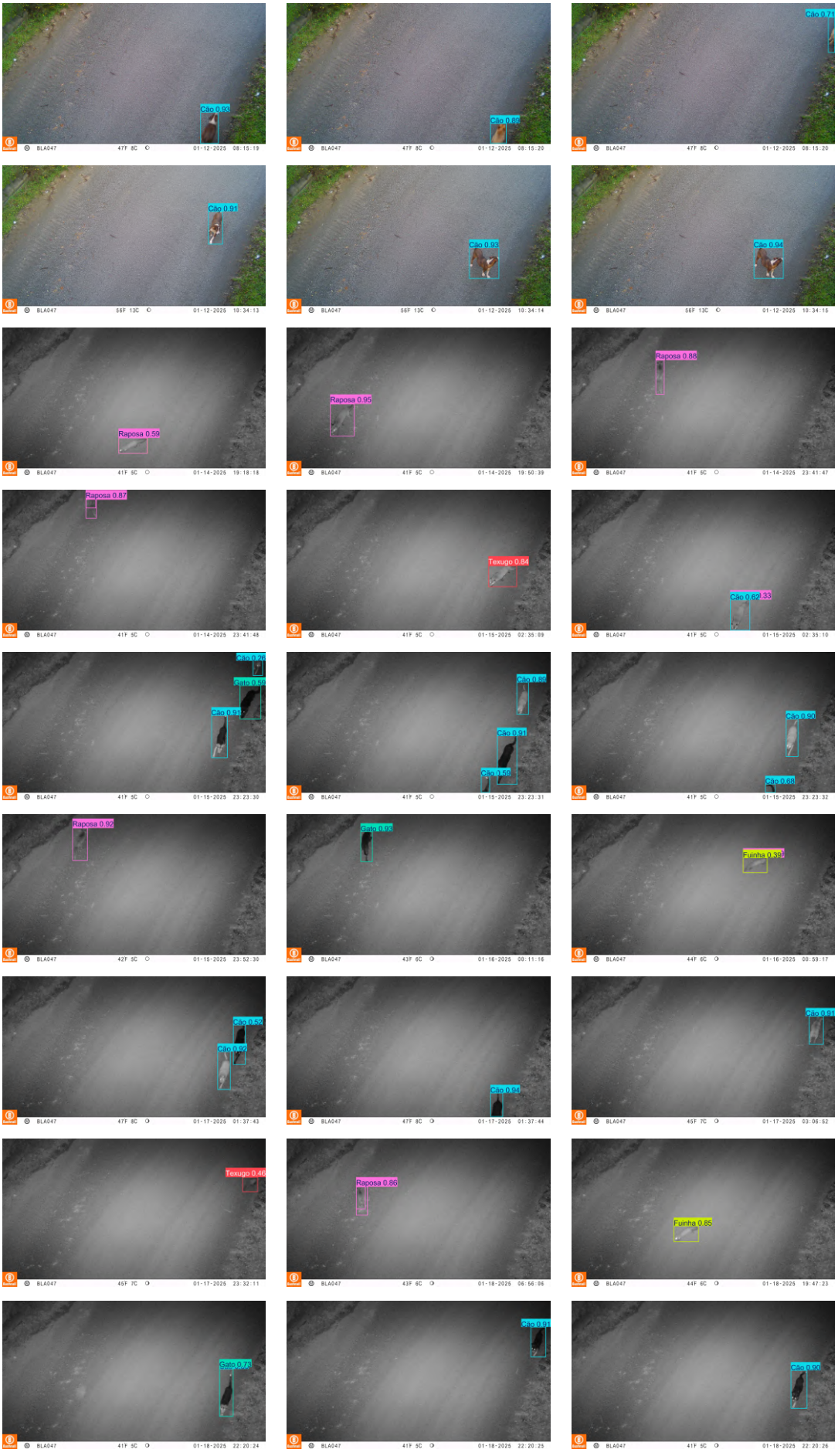


Figura 14: Imagens recolhidas da classe ’Esquilo’

Figura 15: Imagens recolhidas da classe ’Lontra’

B. Previsões do modelo para um conjunto de imagens recolhidas numa determinada passagem da concessão BLA


Cão 0.93
Cão 0.89
Cão 0.77
Cão 0.91
Cão 0.93
Cão 0.94
Raposa 0.59
Raposa 0.95
Raposa 0.88
Raposa 0.87
Texugo 0.84
Cão 0.62
Cão 0.90
Cão 0.68
Raposa 0.92
Gato 0.93
Fuinha 0.39
Cão 0.92
Cão 0.94
Texugo 0.40
Raposa 0.86
Fuinha 0.85
Gato 0.77
Cão 0.92


Cervídeo 0.45
Raposa 0.78
Cão 0.90
Cão 0.82
Cão 0.92
Cão 0.91
Cão 0.94
Cão 0.45
Cão 0.91
Cão 0.91
Cão 0.91
Cão 0.94
Cão 0.91
Raposa 0.94
Raposa 0.76
Cão 0.89
Cão 0.91
Cão 0.58
Cão 0.88
Cão 0.86
Cão 0.90
Cão 0.89
Texugo 0.46
Texugo 0.89
Gato 0.77
Gato 0.41
Cão 0.91
Ave 0.72
Gato 0.74
Raposa 0.71
Raposa 0.42
Raposa 0.36
Raposa 0.47
Raposa 0.82

Figura 19: Previsões do modelo para as imagens recolhidas por uma câmara instalada na concessão BLA