NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE INFORMÁTICA

ANDRÉ MANUEL RASTEIRO DE ARAÚJO

Licenciado em Engenharia Informática

CONTAGEM AUTOMÁTICA DE INSETOS EM ARMADILHAS

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa

⟨Outubro⟩, ⟨2023⟩

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE INFORMÁTICA

CONTAGEM AUTOMÁTICA DE INSETOS EM ARMADILHAS

ANDRÉ MANUEL RASTEIRO DE ARAÚJO

Licenciado em Engenharia Informática

Orientador: Carlos Damásio
Professor Associado, Universidade NOVA de Lisboa

Coorientadores: Fernando Birra
Professor Auxiliar, Universidade NOVA de Lisboa

Ludwig Krippahl
Professor Auxiliar, Universidade NOVA de Lisboa

Júri

Presidente: Doutor Vasco Miguel Moreira do Amaral
Prof. Associado, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Arguente: Doutora Teresa Cristina de Freitas Gonçalves
Prof. Associada, Escola de Ciências e Tecnologia da Universidade de Évora.

Orientador: Doutor Carlos Augusto Isaac Piló Viegas Damásio
Prof. Associado, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
⟨Outubro⟩, ⟨2023⟩

Contagem automática de insetos em armadilhas

Este documento foi gerado com o processador (pdf/Xe/Lua)LaTeXe o modelo NOVAthesis (v6.9.1) [32].

Para a minha família, mesmo aqueles que não estão comigo fisicamente, mas ajudaram a moldar a pessoa que hoje sou. Obrigado por tudo.

Agradecimentos

Gostaria de começar por agradecer ao meu orientador Carlos Damásio por todo o apoio, paciência e confiança depositada em mim durante o processo de desenvolvimento da dissertação. Quero agradecer também aos meus co-orientadores Fernando Birra e Ludwig Krippahl cujo conhecimento em cada uma das suas áreas acabaram por facilitar a aprendizagem e desenvolvimento do projeto em mãos. Tenho de agradecer também a minha inclusão no projeto Smart Farm 4.0, o qual me motivou a terminar a minha dissertação.

De seguida, quero agradecer aos meus amigos, tanto aqueles que fiz ao longo deste percurso académico, como aqueles que nunca me deixaram de apoiar no dia-a-dia, foram eles que me motivaram e inspiraram a concluir esta fase da minha vida.

Por fim, e não menos importante, agradecer à minha família, que me deram condições para frequentar o ensino superior e que sempre foram pacientes e compreensivos ao longo do meu percurso académico.

Resumo

Com o aumento de novas pragas em culturas agrícolas, torna-se necessário efetuar uma monitorização constante dos insetos em terrenos de cultivo de forma a agir conforme a intensidade da praga. Esta monitorização pode ser efetuada através da contagem de insetos em armadilhas, sendo que atualmente esta contagem é efetuada, em muitos casos, de forma manual, acabando por ser uma tarefa complexa e morosa, necessitando a intervenção de um especialista de forma a identificar a espécie que se pretende monitorizar.

Atualmente, com tecnologias de visão computacional torna-se possível automatizar esta tarefa recorrendo a aprendizagem profunda, obtendo valores de erro de contagem baixos e levando menos tempo a efetuar a mesma tarefa. No entanto, estas tecnologias acabam por ser economicamente não alcançáveis para a maior parte dos agricultores e quando a identificação ocorre *in loco*, os valores de exatidão de classificação decrescem substancialmente.

Por estes motivos, o objetivo da nossa solução passa por criar um sistema que, através de imagens tiradas com um simples telemóvel com câmara, consiga contabilizar o número de ocorrências de uma certa espécie em armadilhas para insetos, recorrendo a técnicas de aprendizagem profunda.

Assim, pretendemos criar uma solução que esteja disponível a nível económico a qualquer tipo de agricultor, e que mantenha um nível de exatidão de classificação relativamente elevado.

Palavras-chave: Monitorização, Armadilhas, Aprendizagem Profunda, Contagem, Classificação, Exatidão

Abstract

With the emergence of new pests that damage field crops, there is a need to continuously monitor these pests populations to prevent further damage. This monitoring can be achieved through insect counting from pheromone traps. In most cases, these counts are made manually and require an insect specialist, making these an arduous and time-consuming task.

Recently, computer vision techniques allowed this task to become automated with the help of deep learning, leading to small counting errors, and taking less time to complete the task. However, these types of technologies are expensive which means not every farmer has the means to utilize these techniques. Moreover, when the classification happens in the field, the accuracy of classification drops.

For these reasons, our approach is to develop a system that, by using pheromone trap images taken with a cellphone that has a camera, is able to count how many insects of a specific species are present in those images, making use of deep learning techniques.

Thus, we want to create a solution that is affordable on an economic basis to any type of farmer and which rate of classification accuracy is still high.

Keywords: Monitoring, Pheromone traps, Deep learning, Counting, Classification, Accuracy

Índice

Índice de Figuras

Índice de Tabelas

1

Introdução

Tendo em conta o aumento de novas pragas existentes devido às alterações climáticas e ao comércio internacional, que proporciona a deslocação destes organismos, torna-se necessário monitorizar os terrenos agrícolas para se proceder a contagens de insetos em armadilhas para se produzir as curvas de voo (contagens ao longo do tempo) dos mesmos.

Recentemente desenvolveram-se sistemas inteligentes capazes de fazer esta monitorização e posteriormente contar o número de ocorrências associadas a classes específicas de insetos, retirando o fardo da identificação e contagem de insetos em inúmeras armadilhas.

No entanto, estas soluções requerem um certo investimento económico não acessível a todos e o processo de recolha dos dados é efetuado em condições controladas, recorrendo a câmaras de alta capacidade e sistemas que auxiliam o processo fotográfico [20].

O nosso sistema passa por, através de fotografias simples, tiradas com a câmara de um telemóvel, conseguir obter uma contagem do número de insetos pertencentes a uma dada espécie que se pretenda monitorizar.

Esta dissertação enquadra-se no projeto Smart Farm 4.0, cujo objetivo é contribuir de forma decisiva para a transição e democratização de uma agricultura inteligente, mais sustentável, preditiva e autónoma.

É então, neste âmbito, que esta dissertação foi elaborada, seguindo-se uma descrição mais detalhada deste capítulo. Primeiramente é explicado o contexto em que o trabalho foi realizado (1.1). De seguida é apresentado o problema que se pretende resolver (1.2), e finalmente são apresentados os objetivos e contribuições (1.3).

1.1 Contexto

O número de novas pragas em terrenos agrícolas e doenças de plantas têm vindo a aumentar em Portugal e na Europa. As alterações climáticas, para além de prejudicarem de forma direta a produtividade agrícola devido a temperaturas extremas e redução de precipitação, também contribuem para a propagação de insetos, fungos e bactérias que acabam por ser nefastos para certas culturas [1]. Existem dados que comprovam que este aumento de pragas e doenças de plantas que chegam à Europa está de facto a ocorrer [39].

Uma forma de combater este aumento é através da contínua monitorização e deteção precoce de certas espécies de insetos/pragas, de modo a que posteriormente se possa agir de forma adequada, quer por mecanismos de controlo ou por mecanismos de erradicação. Por outro lado, pode também ser interessante ter dados relacionados com a abundância de outras espécies que não sejam necessariamente pragas, de modo a obter dados sobre o declínio da população de certos insetos.

Tradicionalmente esta monitorização baseava-se numa identificação e contagem manual dos insetos que tinham sido capturados por armadilhas. No entanto, este método requer pessoal especializado, é um trabalho árduo e que requer tempo. Por isso, recentemente surgiram aplicações baseadas em tecnologias da Internet das Coisas (IoT - Internet of Things) de modo a automatizar e monitorizar em tempo real e de modo remoto atividades relacionadas com a agricultura [37]. Técnicas de monitorização à base de imagens são utilizadas em várias áreas, com especial atenção no nosso caso para a monitorização de doenças de plantas [52].

Porém, estas técnicas de monitorização acabam por ser utilizadas em sistemas que não são acessíveis à maioria dos agricultores. Estes sistemas de monitorização requerem investimento numa câmara, componentes auxiliares ao sistema de maneira a manter o sistema operacional diariamente e ainda necessitam de manutenção de material o que acaba por fazer com que este tipo de tecnologia não seja economicamente barata, acabando por ser um entrave à utilização por parte de qualquer agricultor que necessite de efetuar a monitorização de pragas nas suas culturas.

Esta dissertação detalha então o sistema desenvolvido que, através de fotografias tiradas *in loco* a armadilhas de captura de insetos, tem capacidade para reconhecer e contabilizar a quantidade de insetos de uma certa espécie presentes nessas imagens, para que posteriormente o agricultor analise os dados e possa decidir qual a maneira apropriada de lidar com tal espécie.

1.2 Descrição do problema

O problema que pretendemos resolver neste trabalho é a necessidade de os agricultores monitorizarem e contabilizarem a quantidade de insetos de uma certa espécie prejudicial para as culturas, para se necessário agirem posteriormente. Esta tarefa por si só é complexa e morosa devido à quantidade de placas de feromonas a contabilizar, mais a necessidade da contabilização ser efetuada por um especialista, que como veremos mais à frente neste estudo, pode demorar muito mais tempo comparativamente a uma contagem automática por parte de um sistema baseado em aprendizagem profunda.

Já existem soluções que permitem fazer esta contabilização de forma automática e em tempo real, no entanto usam tecnologia e hardware que acabam por ser dispendiosos, uma vez que necessitam de componentes que permitam ao sistema trabalhar diariamente, e.g. um sistema de arrefecimento geral, necessitam de armazenamento de dados, ligação

à Internet, e de outros dispositivos que não são economicamente acessíveis a pequenos agricultores [4].

Levanta-se ainda outro problema, agora relacionado com classificação de insetos *in loco*. A presença de sombras, ramos, folhas, entre outros, é o suficiente para diminuir a precisão de classificação [23]. Por esse motivo, estudos de monitorização à base de imagens montaram um sistema específico para fotografar as armadilhas [20].

Posto isto, criou-se um sistema de deteção e classificação e posteriormente de contagem automática de insetos, que só necessitando de imagens tiradas *in loco*, tem a capacidade de detetar e classificar insetos com um valor considerado razoável para a precisão de classificação (este limiar de razoabilidade foi determinado tendo em conta estudos efetuados e as condições disponíveis).

1.3 Objetivos e Contribuições

Esta dissertação tem por base dois objetivos. O primeiro objetivo abrange o problema que outros sistemas já existentes não estão a resolver com eficácia, que é a perda de precisão de classificação *in loco*[23]. O segundo e objetivo final desta dissertação consiste em fornecer ferramentas tecnológicas com base em aprendizagem profunda a uma maior porção de agricultores, com o objetivo de auxiliar o controlo de pragas emergentes.

Estes objetivos vão ao encontro dos objetivos delineados pelo projeto Smart Farm 4.0, cujos objetivos passavam por:

- Conceber e desenvolver soluções inovadoras e acessíveis tendo por base as tecnologias base da indústria 4.0 (IoT, IA, big data, sistemas ciberfísicos, robots autónomos), para a aplicação ao setor agrícola, principalmente horti-, fruti- e viticultura;

- Dinamizar práticas de agricultura sustentável, preditiva, autónoma e geradoras de elevado valor acrescentado, para valorizar a indústria agroalimentar da Região Oeste, nacional e internacional.

Assim estudou-se a possibilidade de ter um sistema que:

- Se aproxime dos níveis de exatidão na classificação de insetos em armadilhas de outros sistemas já existentes, mas que fazem a deteção e classificação de insetos em armadilhas em situações controladas.

- Seja um sistema de fácil aquisição, estando disponível para qualquer agricultor, independentemente do estatuto económico do mesmo.

- Seja um sistema que, através de fotografias tiradas *in loco* das armadilhas de insetos, com recurso a um telemóvel com câmara, contabilize o número de insetos da espécie que se pretende monitorizar presentes naquela armadilha.

1.4 Estrutura da Dissertação

Tendo apresentado o contexto, o problema e os objetivos para a elaboração da nossa dissertação, apresentamos agora a estrutura deste documento. Os capítulos que compõem este documento são os seguintes:

- Capítulo 2: Este capítulo pretende demonstrar a evolução dos sistemas de deteção e classificação automática de insetos e de que forma chegámos às arquiteturas baseadas em aprendizagem profunda. Este capítulo apresenta ainda estudos que foram efetuados recentemente na área de deteção e classificação de insetos em armadilhas.

- Capítulo 3: O objetivo deste capítulo é explicar a metodologia utilizada durante o desenvolvimento do projeto, desde a obtenção do conjunto de dados até ao treino e avaliação dos modelos. É explicado em pormenor a maneira como se obtiveram os dados que constituem o conjunto de dados utilizado no treino e avaliação dos modelos, assim como, é explicada a metodologia utilizada durante o treino dos modelos, incluindo a maneira como se escolhe o melhor modelo para cada tarefa de classificação e ainda a forma utilizada na avaliação da performance desses modelos.

- Capítulo 4: O capítulo da implementação comporta todo o trabalho efetuado por forma a cumprir os objetivos delineados no capitulo 1. Este trabalho inclui os diferentes métodos utilizados na segmentação dos insetos presentes na placa, os métodos utilizados de forma a gerar a máscara binária e de seguida recorte da placa que serve de armadilha, os programas criados com o objetivo de auxiliar a obtenção de dados para o treino dos modelos e por fim o modo como os modelos foram implementados.

- Capítulo 5: Neste capítulo são apresentados os resultados obtidos durante o treino dos modelos, de forma a justificar a escolha dos modelos utilizados na tarefa de classificação de insetos e são apresentados também os resultados obtidos na fase de avaliação dos modelos previamente escolhidos, de forma a ilustrar a performance dos mesmos no conjunto de teste.

- Capítulo **??**: Este capítulo ilustra e explica a forma de como a nossa aplicação funciona, desde o momento em que a imagem é fornecida e processada, até à contagem final de insetos da espécie pretendida, passando pela forma de como os modelos treinados são utilizados no momento de classificação. São ainda apresentadas neste capítulo, as estatísticas de desempenho da nossa aplicação.

- Capítulo 6: Por fim, o último capítulo tem por objetivo finalizar a dissertação efetuada apresentando as conclusões obtidas após o desenvolvimento do projeto e apresenta ainda algumas ideias que poderão ter relevo na melhoria da aplicação desenvolvida.

2

Estado da Arte

Neste capítulo são apresentados alguns trabalhos relacionados com a monitorização de insetos em culturas agrícolas de modo a expor a evolução e atual estado dos sistemas utilizados para resolver o problema exposto no capítulo anterior. Assim este capítulo estará dividido em quatro partes. Na primeira será apresentada a evolução da tecnologia aplicada de maneira a automatizar o ato de deteção de insetos tanto em plantas como em armadilhas para insetos (2.1). A segunda parte irá introduzir as redes neuronais convolucionais (CNN) e apresentar algumas das arquiteturas mais relevantes (2.2). A terceira parte irá focar-se concretamente nas tecnologias e sistemas mais recentes nesta temática de deteção automática de insetos (2.3). Finalmente a última parte fará um balanço daquilo que foi discutido nas seções anteriores, de forma a apresentar conclusões e darmos início ao que será o desenvolvimento do nosso projeto (2.4).

2.1 Evolução de sistemas de deteção automática

Originalmente, uma forma de monitorização de insetos ocorria através da identificação e contagem manual de famílias de insetos em armadilhas à base de feromonas. Este tipo de trabalho era árduo, demorado e requeria que fosse efetuado por um especialista capaz de distinguir diferentes espécies de insetos e pragas. Para além destes fatores, existe a ocorrência de erros por parte do ser humano, como por exemplo a incapacidade de distinção de certas espécies, contagem do mesmo objeto mais do que uma vez e até mesmo deixar escapar alguns objetos [6, 20].

Por estas razões, chegou-se à conclusão de que seria necessário automatizar este processo de identificação e contagem [34]. Com o crescimento das tecnologias da Internet das Coisas (*IoT - Internet of Things*), estudos que aplicam este tipo de tecnologias de modo a automatizar tarefas associadas à agricultura e monitorização utilizando sistemas de visão têm vindo a aumentar [37]. Entre estes sistema de visão estão os sistemas que utilizam técnicas de monitorização à base de imagens, que permitem monitorizar, entre outras coisas, pragas de culturas agrícolas [52].

Tradicionalmente, estudos de deteção e classificação de objetos em imagens utilizavam

métodos de processamento de imagem como o modelo de contorno ativo (*active contour*) [24], o algoritmo *SIFT* (*scale-invariant feature transform*) [33], o método de deteção do Histograma de Gradientes Orientados (*Histogram of Oriented Gradients - HOG*) [10], atributos Haar [54], máquina de vetores de suporte (*Support Vector Machine - SVM*) [50] e redes neuronais artificiais (*Artificial Neural Network - ANN*) [35].

Entrando no domínio específico de monitorização de pragas à base de imagens, estudos neste domínio utilizavam maioritariamente técnicas tradicionais de processamento de imagem e de aprendizagem automática.

- Xia et al.[59] separaram os insetos do plano de fundo utilizando segmentação através de *watershed* e detetaram as pragas utilizando classificação baseando-se na distância de Mahalanobis, que é uma alternativa à distância euclidiana, que em vez de se basear na distância entre dois pontos, baseia-se na distância entre um ponto e uma distribuição, e é mais eficaz a medir similaridade entre um conjunto de valores e um conjunto de valores obtido a partir dos dados [25].

- Li et al. [28] detetaram pragas de tamanho reduzido em superfícies de folhas através da análise multifractal, obtendo valores mais altos de precisão e sensibilidade comparativamente ao método de *watershed*.

- Wen et al.[57] utilizaram um descritor *SIFT* (*scale-invariant feature transform*), que é basicamente um histograma 3-D de localizações e orientações de gradientes utilizado para descrever a vizinhança de cada ponto de interesse, e posteriormente compararam a performance de seis classificadores diferentes (*minimum least square linear classifier-MLSLC*, *K nearest neighbour classifier-KNNC*, *Parzen density based linear classifier-PDLC*, *principal component analysis expansion linear classifier-PCALC*, *nearest mean classifier-NMC e support vector machine-SVM*) para classificar insetos em pomares.

- Wang et al. [56] classificaram imagens de insetos utilizando máquina de vetores de suporte (*Support Vector Machine - SVM*) e redes neuronais artificiais (*Artificial Neural Network - ANN*) após a extração dos atributos.

- Bakkay et al. [2] detetaram traças utilizando agrupamento de dados através de *k-means* e posteriormente resolveram os casos de insetos a tocarem-se. O processo de agrupamento em *clusters* dividiu o grupo de dados em três grupos diferentes: um primeiro que representa insetos individuais, outro que representa "ruído" (*noise*) e um terceiro que agrupa tudo o que é considerado como insetos a tocarem-se. Para este último grupo é então executada uma dilatação dos contornos, é aplicado o algoritmo proposto cujo objetivo será a separação dos contornos de insetos a tocarem-se e finalmente é aplicado o algoritmo de *watershed*.

- Solis-Sánchez et al. [49] aplicaram o algoritmo *LOSS* para uma identificação inicial da praga seguido da técnica *SIFT*, de modo a detetar insetos em armadilhas.

- Por fim Bechar et al. [3] utilizaram métodos de reconhecimento de padrões utilizando máximos locais de modo a detetar moscas brancas.

Os métodos previamente descritos obtiveram bons resultados no aspeto de identificar e segmentar os insetos presentes nas imagens, no entanto a classificação através dos métodos tradicionais implicava que a pessoa especificasse os atributos manualmente e aplicasse algoritmos de aprendizagem automática após a extração dos atributos. Isto pode gerar problemas quando se torna necessário combinar atributos, uma vez que os métodos tradicionais não conseguem criar e aplicar este tipo de atributos complexos. Foi então que recentemente com métodos de aprendizagem profunda com base em redes neuronais convolucionais (as chamadas CNN) que se conseguiu resolver este problema. A grande diferença entre estas redes e os métodos tradicionais é que estas redes convolucionais têm a capacidade de descobrir as características e atributos de um objeto num certo problema, aprendendo os dados por si só, sem a necessidade de ação humana. Assim, atributos complexos e de difícil compreensão para o ser humano, conseguem ser aprendidos e posteriormente aplicados pelas redes CNN. Não obstante, o treino de redes CNN necessita de um número muito maior de dados, assim como é mais moroso e necessita de uma maior capacidade computacional.

2.2 CNN e arquiteturas relevantes

Antes de se passar à apresentação dos estudos mais recentes associados ao tema desta dissertação, torna-se necessário explicar o que é uma rede CNN.

2.2.1 Redes Neuronais Convolucionais (CNN)

Uma rede neuronal convolucional, mais conhecida como CNN (figura 2.1), é um modelo de rede neuronal especializado para processamento de imagens em duas dimensões, tendo, no entanto, a possibilidade de trabalhar com dados em uma e três dimensões. A rede tem este nome uma vez que a peça fundamental desta rede são as camadas convolucionais que lhe pertencem, e que fazem operações de convolução. Uma convolução é definida como a integral do produto de uma das funções por uma cópia deslocada e invertida da outra.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau$$

No contexto da rede, a convolução é a simples aplicação de um filtro a um *input* que resulta numa ativação. A repetição da aplicação do mesmo filtro a um *input* resulta num mapa de ativações que se denomina de mapa de atributos, que indica a localização e relevo de um atributo detetado num *input*, e.g. numa imagem.

Aprofundando mais a explicação, a convolução é uma operação linear que envolve a multiplicação de um conjunto de pesos ao *input*, sendo que este conjunto de pesos, uma

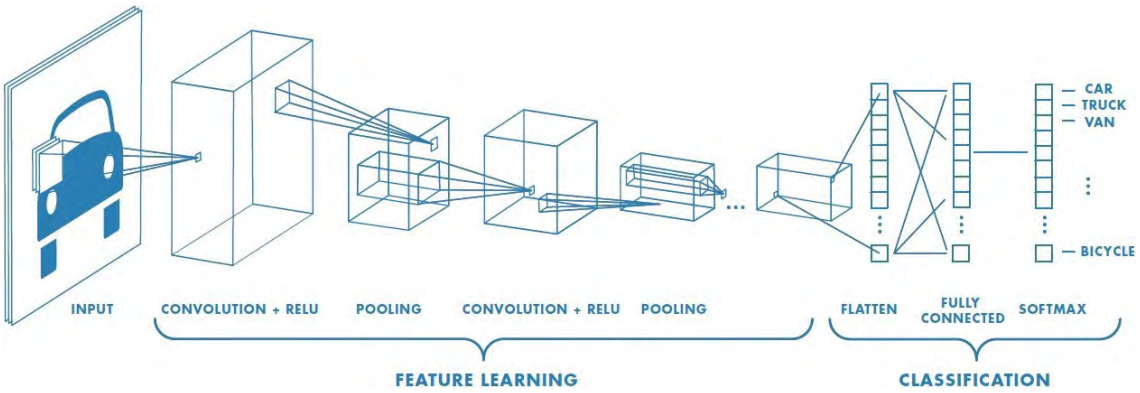
vez que se trata de uma imagem a duas dimensões, acaba por ser uma matriz, à qual se dá o nome de filtro ou *kernel*. De notar que a multiplicação que ocorre é na verdade o produto escalar entre ambas as matrizes.

Este filtro é mais pequeno que os dados de *input*, o que acaba por ser intencional uma vez que permite que o mesmo filtro seja multiplicado pela matriz associada ao *input* várias vezes em diferentes pontos do *input* original. Caso o filtro seja concebido para detetar um tipo específico de atributo, a aplicação sistemática do mesmo filtro em todos os pontos da imagem, permite ao mesmo descobrir esse mesmo atributo em qualquer zona da imagem.

O resultado da multiplicação do filtro pela matriz de *input* é um valor. À medida que o filtro é aplicado múltiplas vezes, o resultado torna-se numa matriz que representa uma filtragem do *input*. Assim, esta matriz resultante é designada mapa de atributos (*feature map*).

2.2.2 Arquitetura de uma Rede Neuronal Convolucional

Qualquer que seja a rede neuronal convolucional, esta é composta por múltiplas camadas, sendo que os três principais tipos de camadas presentes numa rede neuronal convolucional são as camadas convolucionais, camadas de *pooling* e camadas totalmente conectadas (*fully-connected*). No entanto, como se pode ver na figura 2.1, existem outros tipos de camadas que também têm os seus benefícios. De seguida serão apresentados os tipos de camadas mais comuns em redes neuronais convolucionais.



Fonte: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Figura 2.1: Exemplo de arquitetura de uma CNN

2.2.2.1 Camada de *input* (*Input Layer*)

A camada de *input* deverá conter os dados da imagem. Estes dados são representados através de três matrizes (uma matriz para cada cor do RGB).

2.2.2.2 Camadas Convolucionais (*Convolution Layer*)

A camada de convolução como discutido anteriormente pode ser vista como a camada que vai extrair os atributos. Esta camada obtém-se através de diferentes aplicações de filtros, mais conhecidos como *kernels*, à imagem de *input*. Estes filtros vão percorrer a imagem pixel a pixel e produzir uma imagem filtrada que terá aproximadamente o mesmo tamanho da imagem de *input*. Consoante o número de diferentes *kernels* a serem aplicados, esse será o número de imagens filtradas resultantes. Cada filtro extrai um atributo diferente da imagem, e.g. arestas, padrões de cor, e é o conjunto de imagens filtradas que compõe a camada convolucional.

2.2.2.3 Função de ativação

As funções de ativação são funções que permitem à rede aprender padrões complexos presentes nos dados. Existem diferentes tipos de funções de ativação, sendo as três mais comuns as seguintes:

- **sigmoide:**
$$f(x) = \frac{1}{1 + e^{-z}}$$
- **tanh:**
$$f(x) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$
- **ReLU:**
$$f(x) = max(0, x)$$

A função de ativação sigmoide não é mais utilizada na prática, sendo apresentada apenas para reconhecer que tal função existe, até porque a função tanh é uma melhoria da função sigmoide no aspeto de permitir valores positivos e negativos.

Destas três a ReLU será aquela que mais vezes será utilizada. Esta função transforma todos os valores de píxeis negativos em zeros. Esta função em comparação com as outras é menos suscetível ao problema de dissipação de gradientes, para além de que acaba por ser mais eficiente em termos computacionais uma vez que outra particularidade desta função é o facto de não ativar todos os neurónios da rede ao mesmo tempo [27]. Para resolver o problema de saturação na região negativa, existe a possibilidade de usar uma outra função de ativação *Leaky* ReLU.

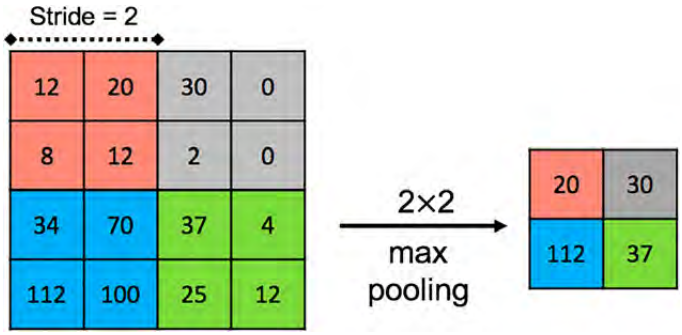
2.2.2.4 Camadas de *Pooling* (*Pooling Layer*)

A camada de *pooling* é responsável por reduzir o tamanho espacial da rede, sendo que basicamente reduz o número de parâmetros e computações da rede, conseguindo assim controlar o problema de sobreajuste. É ainda importante para extrair atributos que não variam com rotações e translações.

Existem vários tipos de *pooling*, no entanto os dois tipos de *pooling* mais utilizados são o *max pooling* e o *average pooling*. O *max pooling* foca-se em devolver o valor máximo da porção da imagem que está a ser inspecionada pelo *kernel*, como demonstra a figura 2.2. Por outro lado, o *average pooling* foca-se em devolver o valor da média de todos os valores cobertos pelo *kernel*.

O *max pooling* também funciona como supressor de ruído, uma vez que descarta as ativações que estão a introduzir ruído. Isto acontece uma vez que o *max pooling* foca-se em reter os atributos de maior importância, aqueles que se apresentam de forma saliente nos dados, levando a que grande parte dos dados sejam rejeitados. *Já o average pooling* retém grande parte dos dados, tendo em conta todos os atributos.

Assim a escolha do método de *pooling* não é óbvio, sendo necessário ter em conta o problema e o objetivo, uma vez que *max pooling* foca-se nos atributos mais importantes, deixando de lado o detalhe enquanto que o *average pooling* encoraja a rede a ter em conta a extensão completa do objeto, mas pode acabar por não extrair os atributos realmente importantes.



Fonte: Imagem retirada de [17]

Figura 2.2: *Max pooling* com *stride* de 2

As camadas anteriormente descritas fazem parte da fase de reconhecimento e aprendizagem de atributos. Posteriormente a esta fase, temos a fase de classificação que compreende outras camadas.

2.2.2.5 Camada de *Flattening* (*Flatten Layer*)

O processo de *flattening* dos dados consiste na transformação dos dados num vetor de uma dimensão de modo a servir de *input* para a próxima camada na nossa CNN. Assim a partir do *output* das camadas convolucionais e de *pooling*, construímos um longo vetor de atributos que posteriormente é ligado à camada totalmente conectada (*fully-connected layer*).

2.2.2.6 Camadas Totalmente Conectadas (*Fully-connected Layer*)

Na parte final da nossa rede neuronal convolucional temos uma ou mais camadas designadas por camadas totalmente conectadas. Têm esta designação devido àquilo descrito

na camada anterior, ou seja, qualquer que seja o *output* resultante das camadas anteriores, este serve agora de *input* para os nós destas camadas. O *output* da última camada deste tipo irá servir de *input* para a última camada da nossa CNN, o que permitirá obtermos a classificação final.

2.2.2.7 Camada *Softmax*/Logistica (*Softmax/Logistic Layer*)

Esta é a última camada da nossa CNN. É esta camada que nos dá o resultado em termos de classificação da imagem que recebemos como *input*. Caso o problema que pretendemos classificar seja do tipo binário, e.g. ser o que procuramos, ou não, esta camada será uma camada logística. Caso o problema se trate de uma classificação mais abrangente, com múltiplas classes, a camada a utilizar será *softmax*.

2.2.3 Arquiteturas CNN

Como explicado na secção anterior, as redes neuronais convolucionais têm uma arquitetura geral baseando-se em três tipos de camadas. Isto permite que diferentes arquiteturas CNN sejam criadas, estudadas e aprimoradas, de modo a obter a melhor performance de deteção e classificação de objetos em imagens. Esta secção pretende apresentar a evolução destas mesmas arquiteturas e quais as arquiteturas mais utilizadas nos estudos mais recentes.

2.2.3.1 R-CNN, *Fast R-CNN* e *Faster R-CNN*

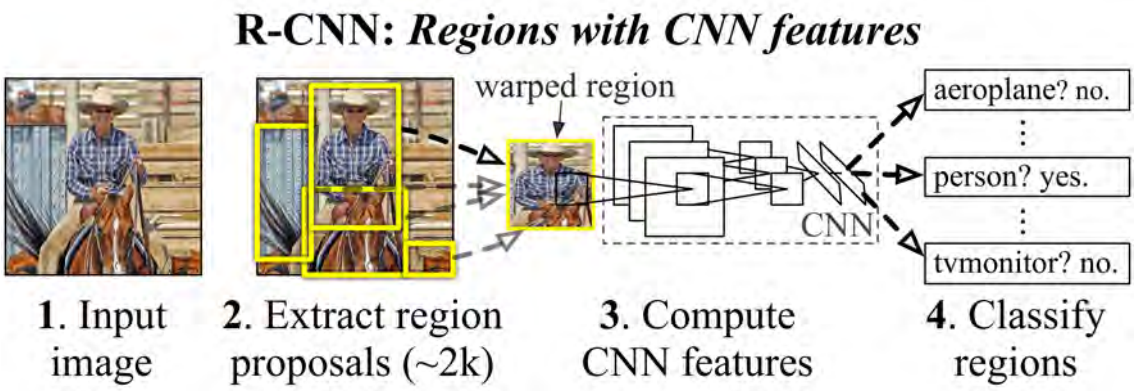
A primeira arquitetura a ser utilizada no âmbito de deteção de objetos foi a R-CNN, e acabou por obter melhor performance quando comparada com os modelos de deteção de objetos já existentes que se baseavam na extração de atributos e aprendizagem automática.

Ross Girshick et al. [16] propuseram um método em que através de uma pesquisa seletiva, 2000 regiões de uma imagem são extraídas, sendo estas regiões designadas de regiões propostas. Assim, em vez de ser necessário classificar todas as regiões de uma imagem, apenas se classifica este número de regiões. Estas regiões servem então de *input* para uma rede neuronal convolucional que produz como output um vetor de atributos com 4096 dimensões. A ideia por detrás desta arquitetura está representada na figura 2.3.

Neste caso a CNN funciona como um extrator de atributos, os quais são de seguida passados para uma máquina de vetores de suporte (SVM) de modo a classificar a presença do objeto na região proposta. Para além de prever esta presença do objeto, o algoritmo também prevê 4 valores de compensação de modo a aumentar a precisão da caixa envolvente.

Este método foi então designado por *Regions with CNN features* (R-CNN), que traduzido à letra significa regiões com atributos de CNN.

Uma vez que a R-CNN tinha alguns aspetos menos positivos, Ross Girshick [15] decidiu aprimorar a arquitetura desenvolvida por si de modo a ter um algoritmo de deteção de objetos mais rápido, e deu-lhe o nome de *Fast R-CNN*, ou R-CNN rápida em português.

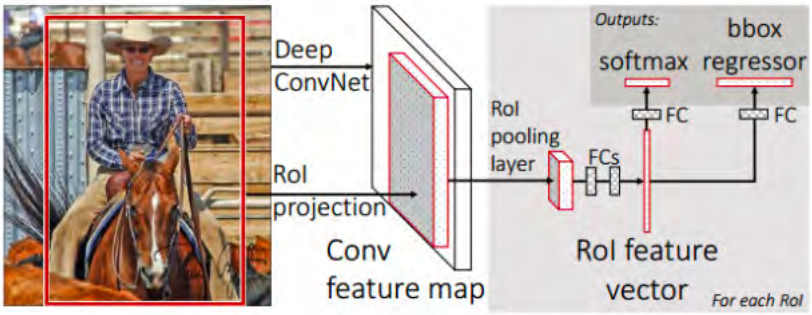


Fonte: Imagem retirada de [16]

Figura 2.3: R-CNN

A ideia é a mesma que a ideia que originou a R-CNN, só que agora em vez de passarmos regiões à CNN, a CNN recebe como *input* a imagem, e é a CNN que vai gerar um mapa convolucional de atributos. É então neste mapa que identificamos as regiões propostas e transformamos, recorrendo a *Region of Interest pooling*, de modo a ter um tamanho fixo para passarmos para a camada totalmente conectada.

No fim é utilizado uma camada *softmax* de modo a prever a classe da região proposta e os valores de compensação da caixa envolvente. A figura 2.4 demonstra esta nova abordagem.



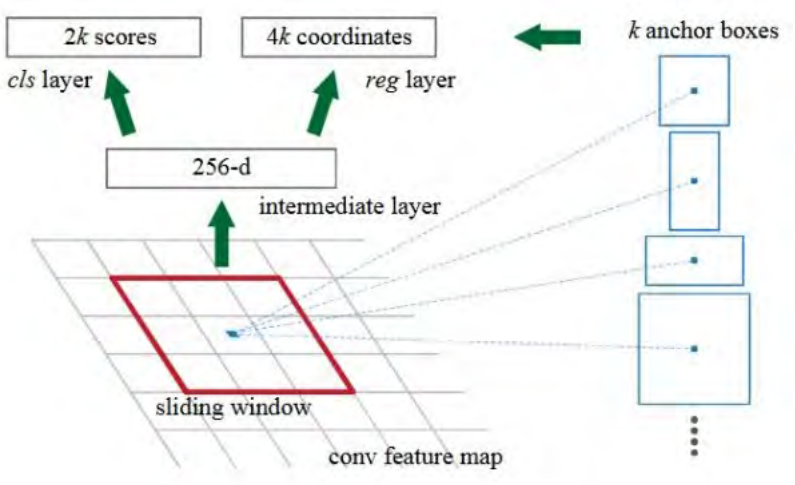
Fonte: Imagem retirada de [15]

Figura 2.4: *Fast R-CNN*

Esta abordagem é mais rápida que a abordagem anterior, uma vez que não estamos a passar como *input* 2000 regiões, mas sim apenas uma imagem que irá gerar um mapa de atributos.

Ambos os algoritmos descritos anteriormente utilizam pesquisa seletiva de modo a descobrir as regiões a propor, no entanto pesquisa seletiva é lenta e compromete a performance da rede e do algoritmo no seu todo. Por essa razão Shaoqing Ren et al. [44] desenvolveram um algoritmo de deteção de objetos capaz de substituir pesquisa seletiva e que permitia à rede aprender as regiões a propor.

Similarmente ao que ocorre na *Fast R-CNN*, a imagem é passada como *input* à rede neuronal convolucional, a qual vai produzir um mapa de atributos. De seguida, em vez de se utilizar pesquisa seletiva, utiliza-se uma *RPN (region proposal network)*, que se encontra representada na figura 2.5, que irá prever as regiões a propor. As regiões propostas previstas são então transformadas recorrendo a *Region of Interest pooling*, que posteriormente é utilizada para classificar a região proposta e os valores de compensação da caixa envolvente.



Fonte: Imagem retirada de [44]

Figura 2.5: *Region Proposal Network*

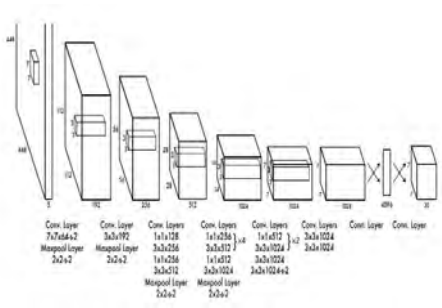
Isto permitiu que este método fosse 250 vezes mais rápido que a R-CNN original e 10 vezes mais rápido que a *Fast R-CNN* e permitiu finalmente deteção de objetos em tempo real.

2.2.3.2 *YOLO (You Only Look Once)*

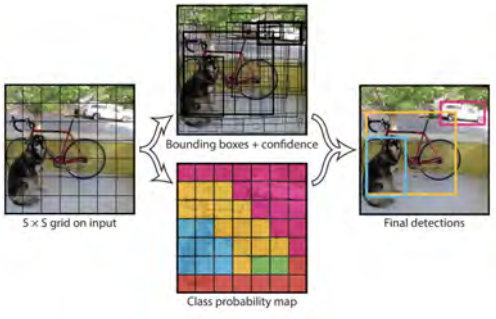
Depois do desenvolvimento destas arquiteturas à base de duas fases, uma fase de separação da imagem em diferentes regiões e uma fase para classificação e ajuste das caixas envolventes, começaram a ser desenvolvidas arquiteturas de deteção de objetos de apenas uma fase. Uma dessas arquiteturas foi a rede *You Only Look Once (YOLO)*[42], traduzido à letra, "só olhas uma vez".

Este método de deteção de objetos utiliza apenas a rede neuronal convolucional, e é a rede que prevê as caixas envolventes e as probabilidades de classes para cada uma dessas caixas. Este sistema modela a deteção como sendo um problema de regressão. Uma imagem é dividida numa grelha de S x S, e em cada célula desta grelha é previsto um número B de caixas envolventes e um valor de confiança para cada uma destas caixas. Depois, para cada uma destas caixas envolventes a rede prevê probabilidades de pertencer a uma classe C e valores de compensação para a caixa envolvente. Caixas que tenham uma probabilidade de classe acima do valor de confiança são selecionadas e utilizadas

para localizar o objeto na imagem. Na figura 2.6 temos uma demonstração da arquitetura 2.6(a) e uma demonstração do modelo 2.6(b)



(a) Arquitetura. Esta rede tem 24 camadas convolucionais seguidas de 2 camadas totalmente conectadas. Camadas convolucionais 1x1 alternadas reduzem o espaço de atributos de camadas anteriores.



(b) Modelo. A imagem é divida numa grelha de S x S, que é posteriormente dividida em caixas envolventes e probabilidades de classes, de modo a obter a deteção final de objetos.

Fonte: Imagens retiradas de [42]

Figura 2.6: Arquitetura e Modelo YOLO

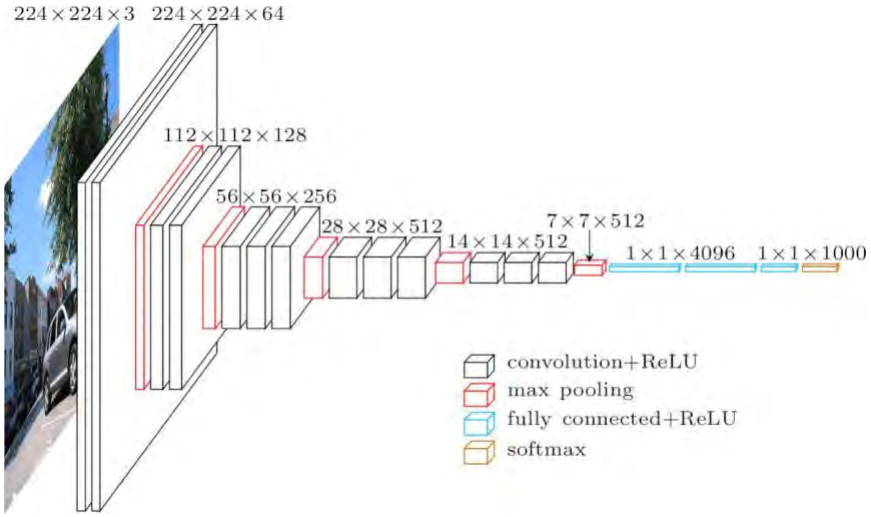
Comparativamente às arquiteturas de duas fases, a rede YOLO é mais rápida e é menos provável de gerar falsos positivos no plano de fundo, no entanto comete mais erros de localização e tem dificuldades com objetos de tamanho reduzido numa imagem.

2.2.3.3 VGG-16

Rede neuronal convolucional proposta por K. Simonyan e A. Zisserman em [48]. Esta rede atingiu 92.7% no top-5 de exatidão no desafio *ImageNet* de 2014. Tendo como base a AlexNet, melhorou a mesma ao substituir o tamanho dos filtros (11 e 5 na primeira e segunda camada convolucional, respetivamente) por múltiplos filtros de tamanho 3 x 3.

Quanto à arquitetura, o *input* da rede tem tamanho fixo, sendo imagens RGB de 224 x 224. A imagem é passada por uma fila de camadas convolucionais, onde os filtros são de 3 x 3. O *stride* na convolução é de 1 pixel e o *padding* (processo no qual se adiciona camadas de zeros à imagem de *input*) é tal de modo que a resolução espacial seja preservada após o processo de convolução. Esta arquitetura tem ainda 4 camadas de *max pooling* que sucedem algumas camadas convolucionais. *Max pooling* é efetuado em janelas de 2 x 2 com *stride* de 2 (O *stride* define o passo entre píxeis a serem verificados).

Após a fila de camadas convolucionais, temos três camadas totalmente conectadas. As primeiras duas camadas têm 4096 canais cada e a terceira camada tem 1000 canais, visto estar adaptada para a *ImageNet* que contém 1000 classes para classificação. No fim temos a camada *softmax* que irá efetuar a classificação. Todas as camadas escondidas utilizam a função de ativação ReLU. A figura 2.7 ilustra a arquitetura desta rede.

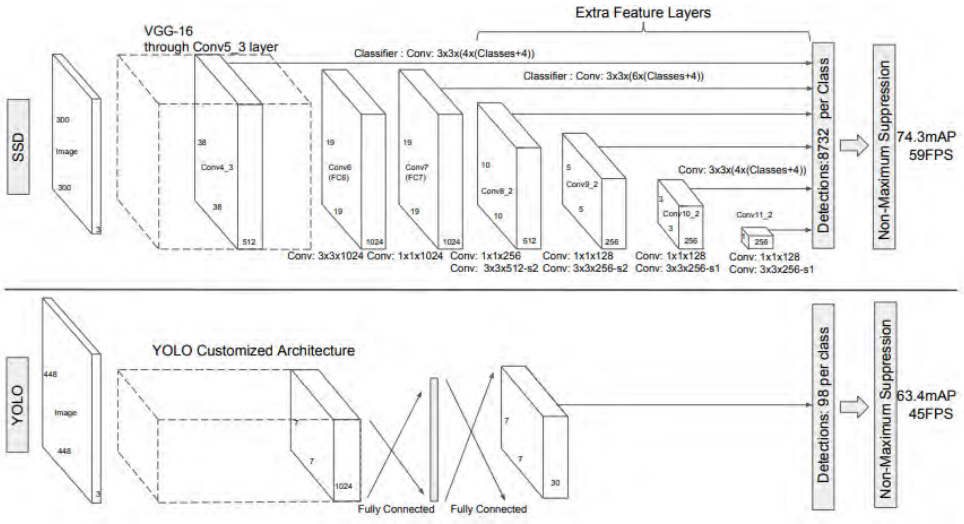


Fonte: https://neurohive.io/en/popular-networks/vgg16/

Figura 2.7: Arquitetura VGG-16

2.2.3.4 *SSD (Single Shot Multibox Detector)*

A rede *SSD (Single Shot Multibox Detector)*[30], é composta por dois componentes: um modelo que é a espinha dorsal e uma cabeça SSD. A espinha dorsal normalmente é uma rede de classificação de imagens previamente treinada. No caso do estudo foi utilizada uma arquitetura VGG-16, sendo que as camadas totalmente conectadas foram descartadas. No lugar destas aparece a cabeça SSD, que é basicamente um conjunto auxiliar de camadas convolucionais de modo a permitir a extração de atributos em múltiplas escalas e a reduzir progressivamente o tamanho do *input* da próxima camada. A figura 2.8 mostra esta arquitetura e compara-a com a arquitetura da rede YOLO.



Fonte: Imagem retirada de [30]

Figura 2.8: Arquiteturas SSD e YOLO

Tal como já acontecia na arquitetura YOLO, a imagem de *input* é dividida numa grelha e cada célula desta grelha é responsável por detetar objetos nessa região da imagem. No entanto temos problemas se existir mais do que um objeto dentro duma célula e/ou tiverem diferentes formas. Para resolver este problema criou-se o conceito de caixa âncora. Estas caixas âncora são pré-definidas e cada uma delas é responsável por uma forma e tamanho dentro de uma célula da grelha. Durante o treino, a SSD utiliza uma fase de correspondência de modo a fazer a correspondência entre a caixa âncora apropriada com as caixas envolventes de cada objeto da *groundtruth*. Assim, a caixa âncora com maior valor de sobreposição com um objeto fica responsável por prever a sua classe e a sua localização na imagem. No entanto, continua a existir o problema de pior classificação em objetos de menores dimensões.

2.2.3.5 *Retinanet*

Retinanet [29] é mais uma arquitetura baseada na deteção de objetos em uma fase, e acabou por se tornar numa das melhores. *Retinanet* tem base em duas melhorias em relação a outros modelos de deteção de objetos de uma fase. Estas melhorias são *Feature Pyramid Networks (FPN)* e *Focal Loss*.

O conceito por detrás destas redes de pirâmides de atributos é a construção de pirâmides de atributos a partir de pirâmides de imagens, ou seja, tendo uma imagem, faz-se subamostras desta reduzindo a resolução e o tamanho sucessivamente, formando-se uma pirâmide. Os atributos são depois extraídos em cada camada desta pirâmide de forma a detetar os objetos.

Várias arquiteturas utilizam esta estrutura em pirâmide, e no caso da FPN obtemos uma boa semântica em todos os níveis uma vez que combina atributos ricos em semântica a uma baixa resolução com atributos com semântica fraca, mas alta resolução. Isto é possível graças à criação de um caminho de cima para baixo com conexões laterais a camadas convolucionais construídas de baixo para cima.

Na arquitetura da *Retinanet*, a qual se encontra representada na figura 2.9, existem 4 componentes principais:

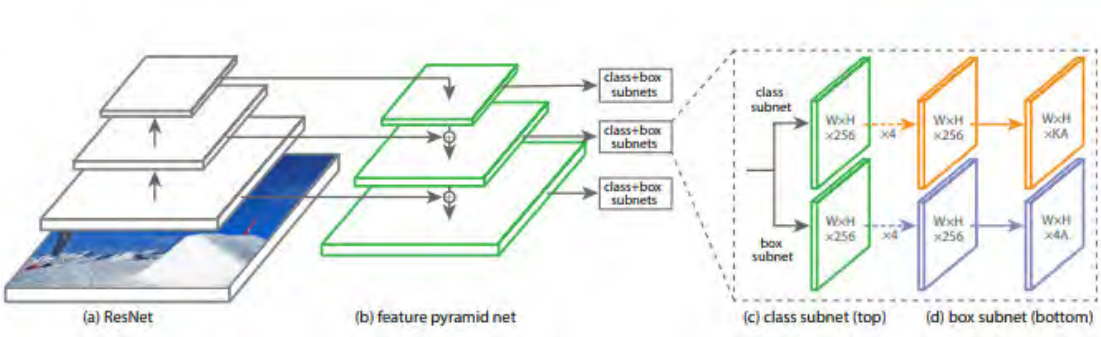
- **Caminho de baixo para cima (*Bottom-up pathway*)** - Isto é a espinha dorsal da rede que calcula os mapas de atributos a diferentes escalas.

- **Caminho de cima para baixo (*Top-down pathway*) e conexões laterais** - O caminho de cima para baixo faz *upsampling* dos mapas de atributos espacialmente mais grosseiros que estão nos níveis superiores da pirâmide e as conexões laterais fundem as camadas *top-down* com as camadas *bottom-up* que tenham o mesmo tamanho espacial.

- **Sub-rede de classificação** - Prevê a probabilidade do objeto se encontrar naquela localização para cada uma das caixas âncora e classes de objetos.

- **Sub-rede de regressão** - Faz a regressão da compensação para cada caixa envolvente das caixas âncora para cada objeto pertencente à *groundtruth*.

Esta arquitetura introduziu ainda *Focal Loss*. *Focal Loss* é uma melhoria de *Cross-Entropy Loss* e foi introduzida de modo a lidar com problemas em falta de equilíbrio entre classes. Isto acontece maioritariamente em modelos de uma fase devido à falta de equilíbrio entre classes pertencentes ao primeiro plano e as classes pertencentes ao plano de fundo, devido às caixas âncora. Apenas algumas caixas âncora terão um objeto pertencente à *groundtruth* enquanto que a maior parte irá ter uma classe de plano de fundo. *Focal Loss* consegue então reduzir a contribuição da perda de exemplos fáceis (deteções com alta probabilidade) e aumenta a importância de correção de exemplos mal classificados.



Fonte: Imagem retirada de [29]

Figura 2.9: Arquitetura *Retinanet*. Esta arquitetura utiliza uma *Feature Pyramid Network (FPN)* por cima de uma arquitetura *feedforward ResNet* (a) de modo a gerar uma pirâmide de atributos convolucionais multi-escala (b). A esta espinha dorsal da arquitetura juntam-se duas sub-redes, uma para classificação de caixas âncora (c) e uma para fazer a regressão de caixas âncora para caixas de objetos da *groundtruth* (d). Esta arquitetura é simples intencionalmente, o que permite o foco numa nova função de perda, *focal loss*, que elimina a diferença em termos de exatidão dos detetores de duas fases para os detetores de uma fase, permitindo correr o processo de forma mais rápida.

Assim, obteve-se um nível de exatidão similar ao dos detetores de duas fases, ainda mesmo em deteção de objetos de dimensões reduzidas, que era o grande problema de outras arquiteturas de uma fase como o caso da YOLO e da SSD.

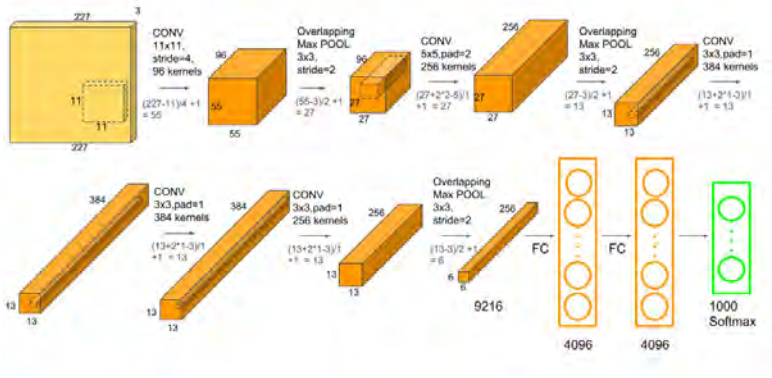
2.2.3.6 AlexNet

Arquitetura que ganhou o nome devido à pessoa que a criou, Alex Krizhevsky et al. [27]. Esta rede ajudou a dar o primeiro passo em termos de reconhecimento da aprendizagem profunda, ao atingir um top-5 erro de 15.3% no desafio *ImageNet* de 2012.

Esta rede tem a capacidade de classificar imagens dentro de 1000 classes, tendo por isso um vetor de output com 1000 valores representado a probabilidade de pertencer à classe C, sendo que a soma de todos os elementos do vetor é 1. O *input* desta rede tem de

ser uma imagem RGB de dimensões 256 x 256, obrigando a que imagens de treino e teste tenham estas dimensões. Cortes aleatórios de 227 x 227 são gerados a partir da imagem de *input* para serem passados para a primeira camada da rede.

A arquitetura desta rede encontra-se ilustrada na figura 2.10 e baseia-se em 5 camadas convolucionais e 3 camadas totalmente conectadas, tendo ainda 3 camadas de *Overlapping Max Pooling*. As duas primeiras camadas convolucionais são seguidas por camadas de *overlapping max pooling*. As três próximas camadas convolucionais estão conectadas umas às outras, sendo que a quinta é seguida de uma camada de *overlapping max pooling* final. Por fim, o *output* desta série de camadas é passado para uma série de camadas totalmente conectadas que no fim tem um classificador *softmax* com 1000 classes.



Fonte: https://learnopencv.com/understanding-alexnet/

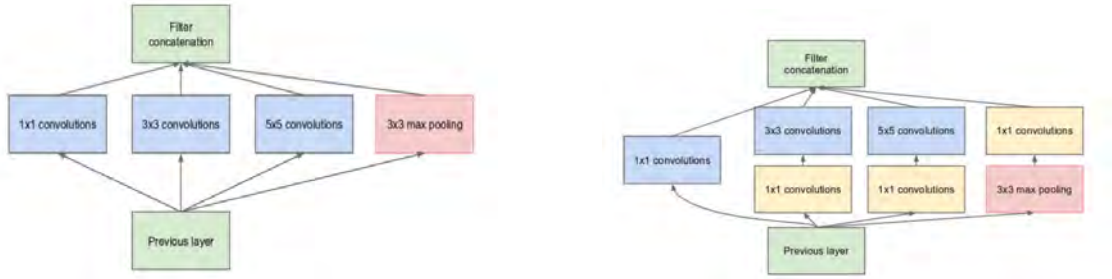
Figura 2.10: Arquitetura AlexNet

Utiliza-se ainda a função de ativação ReLU após as camadas convolucionais e camadas totalmente conectadas. *Overlapping max pooling* é similar ao conhecido *max pooling*, sendo a diferença relacionada com a sobreposição de janelas adjacentes. Os autores utilizaram janelas de 3 x 3 com *stride* de 2. De modo a reduzir o sobreajuste aplicaram a técnica de *dropout*. Esta técnica baseia-se em existir uma probabilidade de um neurónio não ser considerado no processo de propagação.

2.2.3.7 *Inception (GoogLeNet)*

Szegedy et al. [51] criaram esta rede com o intuito de resolver alguns problemas existentes com redes profundas. O objeto que pretendemos identificar pode ocupar uma área com tamanho diferente dependendo da imagem. Isto torna difícil a escolha do tamanho correto para o *kernel*, uma vez que um *kernel* maior é preferível para informação distribuída globalmente enquanto o *kernel* menor é preferível para informação distribuída de maneira mais local. Existe ainda o problema de redes muito profundas serem propícias a gerar sobreajuste. Amontoar operações convolucionais de maneira ingénua é computacionalmente caro.

Por esta razão os autores decidiram ter filtros com tamanhos variados no mesmo nível, fazendo com que a rede em vez de ficar mais profunda, agora fica mais larga. Os autores consideraram então dois modelos de módulos, um ingénuo, ilustrado em 2.11(a), que aplica convolução com três tamanhos diferentes de filtros (1x1, 3x3, 5x5), efetuando também *max pooling* e no final efetua uma concatenação dos outputs destas camadas e envia para o próximo módulo, e um outro modelo,representado em 2.11(b), com redução de dimensões em que os autores limitam o número de canais de *input* através da adição de uma convolução de 1x1 antes das convoluções de 3x3 e 5x5. É ainda adicionada uma convolução de 1x1 após *max pooling*.



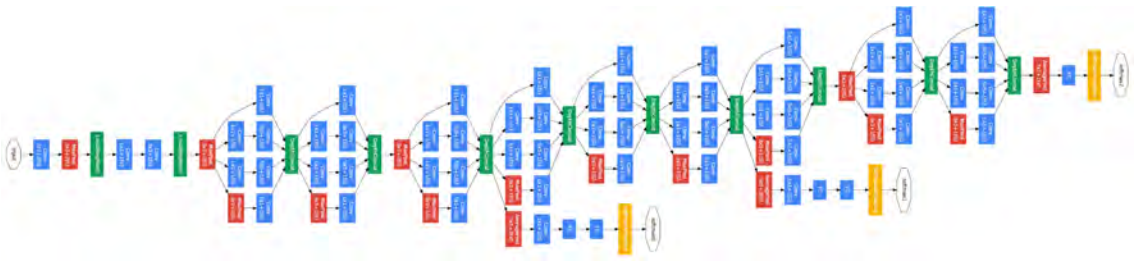
(a) Módulo ingénuo de *Inception*.

(b) Novo módulo de *Inception*.

Fonte: Imagens retiradas de [51]

Figura 2.11: Diferentes módulos *Inception*

Foi com este último modelo, que se criou a *GoogLeNet* (fig 2.12). Esta rede tem 9 módulos em fila, em profundidade tem 22 camadas, 27 considerando-se as camadas de *pooling*. Utiliza *global average pooling* em vez de *max pooling* no fim do último módulo. Tendo em conta que é uma rede profunda, está sujeita ao problema de dissipação de gradientes. De modo a evitar que a rede "morresse" a meio, os autores adicionaram dois classificadores auxiliares que basicamente aplicam *softmax* aos outputs de dois dos módulos e calculam a perda. A perda total é uma soma ponderada desta perda calculada pelos auxiliares e da perda real.



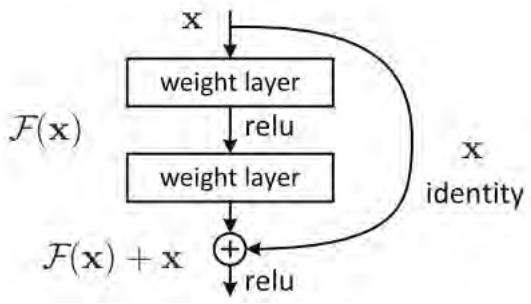
Fonte: Imagem retirada de [51]

Figura 2.12: Arquitetura *GoogLeNet*

2.2.3.8 *ResNet*

Proposta por He et al. [18], esta rede pretendia solucionar ou aliviar o problema do treino de redes muito profundas.

Primeiramente os autores provaram empiricamente que existia um limite em quão profunda a rede poderia ser antes de começar a ficar sobreajustado, e o erro de teste deixar de reduzir e começar a aumentar. Foi então que os autores introduziram uma nova camada neuronal à rede, o bloco residual representado na figura 2.13.



Fonte: Imagem retirada de [18]

Figura 2.13: Bloco residual

Este bloco tem uma alteração importante chamada de *Skip Connection*.

Estas conexões de salto entre camadas adicionam os outputs de camadas anteriores ao output das camadas empilhadas. Isto permite o treino de redes muito mais profundas do que anteriormente era possível. Este tipo de redes residuais atingiu uma percentagem de erro de 3.57% no desafio *ImageNet* de 2015.

A arquitetura onde se testou estas conexões de salto seguiu 2 heurísticas inspiradas na rede VGG:

- **1.** Se os mapas de atributos de output tivessem a mesma resolução, então a profundidade do mapa de filtro permanecia inalterada.
- **2.** Se o mapa de atributos no output fosse reduzido a metade, e.g. 32x32 -> 16x16, então a profundidade do mapa de filtro era duplicada.

2.2.3.9 *MobileNet*

Modelo desenvolvido por Howard et al. [22] para ser utilizado em aplicações móveis. Esta rede utiliza convoluções separáveis em profundidade, reduzindo o número de parâmetros comparativamente a redes com convoluções normais. Este tipo de convolução é baseada em duas operações, uma convolução em profundidade e uma convolução ponto a ponto.

Os autores descrevem ainda que é aplicada normalização em lote e a função de ativação ReLU após todas as camadas exceto a última camada totalmente conectada que é ligada a uma camada *softmax* para classificação. Utiliza-se também uma camada de

average pooling de maneira a reduzir a resolução espacial a 1 antes de se ligar à camada totalmente conectada.

A diferença entre esta arquitetura e outras prende-se no novo tipo de convolução separável em profundidade. A ideia surgiu da suposição de que a profundidade e dimensão espacial dum filtro eram separáveis, daí o nome. Convolução separável em profundidade é um convolução em profundidade seguida de uma convolução ponto a ponto.

A convolução em profundidade é um convolução espacial DK x DK baseada em canais. Supondo que temos 5 canais, iremos ter 5 DK x DK convoluções espaciais. Convolução separável em profundidade é um mapa de uma única convolução em cada *input* de um canal separadamente. Por isso o número de canais de output é o mesmo de canais de *input*.

A convolução ponto a ponto é uma convolução 1 x 1 de modo a mudar a dimensão. É uma convolução com tamanho de *kernel* de 1 x 1 que simplesmente combina os atributos criados pela convolução em profundidade.

2.3 Estudos recentes de deteção e classificação de insetos

2.3.1 Contagem automática de insetos em armadilhas à base de feromonas

Hong et al. [20] ao criarem um sistema de contagem automática de pragas para monitorização de *Matsucoccus thunbergianae*, chegaram à conclusão que o processo de contagem e identificação manual de insetos, tendo em conta o número de insetos a contar na armadilha, era significativamente mais lento comparativamente a arquiteturas CNN de classificação e deteção de insetos.

Em média, o tempo de contagem e identificação manual de insetos foi de 199, 198 e 501 segundos em armadilhas com menos de 300, entre 300 e 500 e mais de 500 insetos respetivamente. Ora, estes tempos de contagem são aproximadamente 8 a 27 vezes mais lentos do que o tempo de contagem com uma arquitetura *Faster R-CNN Resnet* 101 com um tamanho de *input* de 1024 e são 85 a 276 vezes mais lentos do que o processo de contagem com uma arquitetura *SSD MobileNet*v.2 com 640 de tamanho de *input*.

No entanto, rapidez não interessa se os insetos estiverem a ser mal identificados. Por essa razão os autores arranjaram uma forma de medir a performance do detetor de objetos. Chamaram a essa métrica erro de contagem e a equação era a seguinte:

$$counting\ error(\%) = \sum_{i=1}^{N} |\frac{C_i - \hat{C}_i}{C_i}| \times 100$$

onde C_i é o número de insetos contados manualmente por um especialista na armadilha i e $\hat{C}_i$ é o número de insetos da espécie pretendida detetados pelo detetor de objetos na armadilha i.

Os resultados encontram-se na figura 2.14, quando a condição de recorte foi de 12x8 e na figura 2.15 quando a condição de recorte foi de 6x4.

Model	Input Size	Counting Time (s)	Counting Error (%)
Faster R-CNN Resnet 101	1024	14.14	2.11
Faster R-CNN Resnet 101	512	9.17	3.69
EfficientDet	1024	14.44	3.37
EfficientDet	512	5.29	3.42
Retinanet50	1024	6.58	3.30
Retinanet50	640	4.78	2.95
SSD Mobilenet v.2	640	3.81	2.32
SSD Mobilenet v.2	320	3.63	3.32

Fonte: Tabela retirada de [20]

Figura 2.14: Resultados erro de contagem de diferentes arquiteturas com condição de recorte 12x8

Model	Input Size	Counting Time (s)	Counting Error (%)
Faster R-CNN Resnet 101	1024	3.90	3.95
Faster R-CNN Resnet 101	512	2.50	4.02
EfficientDet	1024	3.92	4.70
EfficientDet	512	1.68	4.21
Retinanet50	1024	1.88	3.83
Retinanet50	640	1.45	3.74
SSD Mobilenet v.2	640	1.40	3.65
SSD Mobilenet v.2	320	1.19	6.69

Fonte: Tabela retirada de [20]

Figura 2.15: Resultados do erro de contagem de diferentes arquiteturas com condição de recorte 6x4

Estas condições de recorte surgem visto que a redução da imagem de *input* aumenta a performance de deteção tendo como contrapartida o processo geral de deteção ser mais lento visto que o processo tem de ser repetido mais vezes visto que em vez de existir uma imagem, agora temos uma imagem repartida em várias. Neste estudo os autores chegaram à conclusão que as melhores dimensões da imagem pós-corte seriam 12x8 e 6x4, sendo que o estudo foi então efetuado tendo em conta estas duas condições de pós-corte.

Neste estudo foram utilizadas 16 condições diferentes para 4 tipos de modelos e tamanhos de *input* diferentes. Assim utilizou-se *Faster R-CNN Resnet* 101 com tamanhos de *input* de 512 e 1024, *EfficientDet* D0 e D4 com tamanhos de *input* 512 e 1024 respetivamente, *Retinanet* 50 com tamanhos de 1024 e 640 e por fim *SSD MobileNet* v.2 com tamanho de *input* 640 e 320.

De modo a adicionar robustez ao modelo, efetuou-se *transfer learning* utilizando pesos pré treinados com o *COCO dataset* e técnicas de *data augmentation* como rotações verticais e horizontais, *crop and pad* aleatórios, contraste e ajustes de brilho.

A performance dos modelos treinados foi posteriormente avaliada com recurso à métrica de precisão média (*average precision*), que não é a média dos valores de precisão,

mas sim equivalente à área abaixo das curvas de precisão (*precision*) e sensibilidade (*recall*). Para calcular os valores de precisão e sensibilidade utiliza-se as equações abaixo:

$$Precisão(\%) = \frac{TP}{TP+FP} \times 100 \tag{2.1}$$

$$Sensibilidade(\%) = \frac{TP}{TP+FN} \times 100 \tag{2.2}$$

onde TP, FP e FN representam verdadeiro positivo(*true positive*), falso positivo (*false positive*) e falso negativo (*false negative*) respetivamente. Para ser considerado um verdadeiro positivo (TP) é necessário que o inseto que aparece na imagem seja classificado corretamente. Caso contrário é um falso negativo (FN). O inseto que não se encontra na imagem é considerado um verdadeiro negativo se a classificação é incorreta, caso contrário é considerado um falso positivo(FP).

É preciso ter ainda em conta o quão a área prevista se sobrepõe à *groundtruth* e a isto se dá o nome de *intersection over union (IoU)*. Associada a esta interseção existe um limiar ou *threshold*, por norma 0.5, mas que no caso do estudo destes autores, também exploraram os valores com limiar de 0.3.

2.3.2 Técnicas modernas de aprendizagem automática e profunda na classificação e deteção de insetos

Um outro estudo interessante para a temática desta dissertação foi elaborado por Thenmozhi Kasinathan, Dakshayani Singaraju e Srinivasulu Reddy Uyyala [23]. Este artigo é relevante uma vez que compara a performance de algoritmos de aprendizagem automática com uma rede neuronal convolucional na exatidão de classificação.

Neste estudo, foi realizada a deteção e classificação de insetos para os *datasets* de Wang [56] e de Xie [60] para diferentes tipos de campos de cultivo. O *dataset* de Wang tem nove classes de insetos e o *dataset* de Xie tem 24 classes. No *dataset* de Wang o conjunto de treino consiste em 162 imagens de insetos e o conjunto de teste em 63 imagens. O *dataset* de Xie contém 785 imagens de insetos no conjunto de treino e 612 imagens de insetos no conjunto de teste.

Normalmente recorre-se a técnicas de melhoramento de imagem de maneira a reduzir o ruído e melhorar a precisão. Neste caso,os *datasets* utilizados já se encontravam pré-processados e segmentados. Todavia, uma vez que ambos os *datasets* utilizados tinham poucas imagens de insetos, recorreu-se a *image augmentation*. As imagens de insetos foram redimensionadas para 227 x 227 píxeis e foram aplicadas diferentes técnicas de *data augmentation* como rotação, recortes e *flipping*. Após esta fase, o conjunto de treino de Wang continha agora 1296 imagens de insetos e o conjunto de treino de Xie continha 6280 imagens.

Após o processo de extração de atributos, os autores aplicaram os algoritmos de ANN, SVM, KNN e NB de modo a classificar os insetos em classes. Utilizou-se a CNN também de

maneira a comparar performance. A experiência foi conduzida com *9-fold cross-validation* para ambos os *datasets* de forma a avaliar a performance de cada um dos algoritmos.

Os classificadores utilizados foram então:

- **Classificador ANN:** Uma rede neuronal simples com uma camada de *input*, uma camada "escondida" e uma camada de output. Inicialmente, pesos aleatórios são atribuídos. Os pesos finais são determinados e o rácio de ativação da camada de output é calculado. Uma rede neuronal artificial multicamada *feed-forward* é designada para identificar e classificar moscas-brancas e tripes em fase adulta. A capacidade de reconhecimento de insetos do modelo foi melhorada com um modelo *back-propagation* ANN para distinção de *Spodoptera exigua*.

- **Classificador SVM:** É um algoritmo de aprendizagem automática supervisionada. O espaço dimensional é baseado no número de atributos que existem. A classificação de insetos ocorre através da identificação do hiper-plano que diferencia muito bem as classes. Estes tipos de classificadores foram eficazes na minimização de erros na identificação de mosquitos e moscas da fruta.

- **Classificador KNN:** É um algoritmo de "*lazy learning*". Neste algoritmo, os pesos são atribuídos consoante a contribuição dos vizinhos, de modo que vizinhos mais próximos tenham mais peso do que vizinhos mais distantes. Em problemas de classificação com muitas dimensões e poucas amostras, este algoritmo obtém melhores resultados na identificação de espécies de borboletas.

- **Classificador de *Naive Bayes*:** Este classificador é baseado no Teorema de Bayes e pressupõe que a presença de um atributo não tem influência na presença de outros atributos numa classe de insetos. Este tipo de classificação foi aplicado para prever com que probabilidade é que um tuplo do *dataset* de insetos em campos de soja pertence a uma classe em particular.

- **Classificação de insetos com modelo CNN:** Este modelo contém cinco camadas convolucionais, três camadas *max-pooling*, uma camada *flatten* , uma camada totalmente conectada e uma camada de output *softmax* para a classificação das imagens de insetos. A imagem foi redimensionada para 64 x 64. O modelo tem a capacidade de correr cada imagem rapidamente e reduz as operações computacionais por camada e as necessidades de memória. Cada camada convolucional e cada camada *max-pooling* usa filtros 3 x 3 e 2 x 2 respetivamente. A camada totalmente conectada é definida de forma a aprender atributos de alto nível para a classificação final de insetos.

Thenmozhi Kasinathan, Dakshayani Singaraju e Srinivasulu Reddy Uyyala criaram ainda um algoritmo para a deteção de pragas. Esse algoritmo é o seguinte:

- A imagem é carregada e redimensionada para 227 x 227. O *OpenCV* lê a cor da imagem na ordem BGR (azul, verde, vermelho). Uma imagem de máscara "*mask*" é criada que contém uma matriz de zeros com o mesmo tamanho que a imagem de *input*.

- As matrizes que representam o primeiro plano e o plano de fundo são criadas e preenchidas com 0's. Estas duas matrizes são utilizadas internamente quando o inseto que se apresenta em primeiro plano é segmentado do plano de fundo.

- Define-se as coordenadas da caixa envolvente. Esta compreende a região onde se encontra o inseto que queremos segmentar na imagem de *input*.

- Aplica-se uma adaptação do algoritmo de *GrabCut* [45], através da aplicação da máscara previamente criada à imagem de *input* de modo a separar o inseto em primeiro plano do fundo colorido da imagem, selecionando-se o modo de inicialização da caixa envolvente e deixando o algoritmo correr durante cinco iterações.

- Gera-se a imagem de máscara output "*mask_output*" de tal modo que os píxeis que provavelmente pertencem ao fundo da imagem, e aqueles que realmente pertencem ao fundo, são atualizados para zero e os píxeis que provavelmente pertencem ao primeiro plano da imagem, e aqueles que realmente pertencem ao primeiro plano são atualizados para um na máscara "*mask*". Para obter a imagem segmentada "*image_seg*" os canais BGR da imagem de *input* foram multiplicados pela máscara de imagem de output "*mask_output*".

- Cria-se outra imagem de máscara com o mesmo tamanho da imagem de *input* com uma matriz que consiste em zeros. Esta máscara tem o nome de "*background_img*". Posteriormente todos os píxeis desta imagem são convertidos para píxeis brancos. Esta máscara é adicionada à imagem segmentada "*image_seg*", gerando uma imagem "*image_segmented*".

- Esta última imagem é processada convertendo o modelo de cor RGB para um modelo de cor HSV [*Hue, Saturation and Value*] (Tonalidade, Saturação e Valor), e é aplicado o Desfoque Gaussiano de modo a reduzir o ruído Gaussiano. Aumenta-se o contraste da imagem utilizando Equalização de histograma. Foi ainda utilizado uma adaptação do *inverted binary tresholding*.

- A função *findCountours()* do *OpenCV* é aplicada para identificar os contornos na imagem binária. Após descobrir estes contornos o maior é selecionado e os contornos que sobram são eliminados.

- Através da função *minAreaRect()* a área mínima do retângulo que envolve o maior contorno é calculada. As coordenadas deste retângulo são utilizadas para desenhar o retângulo rodado na imagem de *input* que contém o inseto.

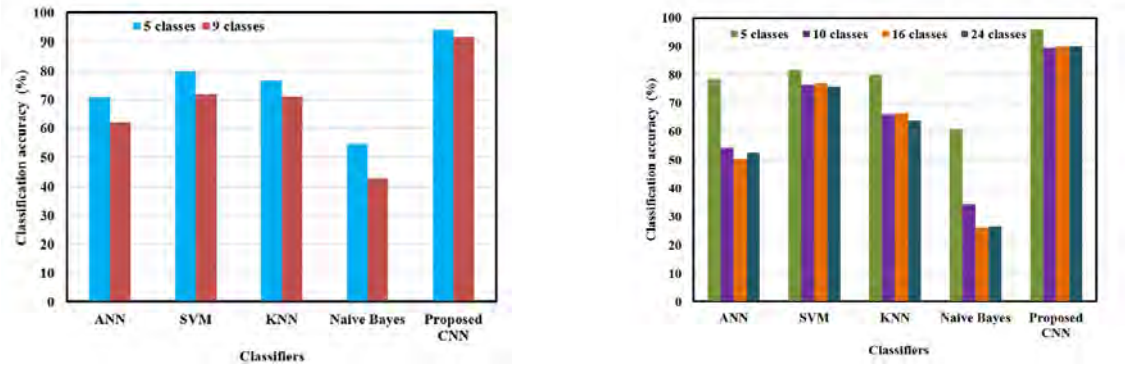
Este algoritmo foi aplicado para os *datasets* de Xie [60], Wang [56], Deng [11] e IP102 [58].

De maneira a avaliar a performance dos modelos treinados os autores utilizaram a exatidão de classificação, tendo utilizado a seguinte equação para a calcular:

$$classification\ accuracy = \frac{TP + TN}{TP + TN + FN + FP}$$

onde, TP, FP, FN e TN representam verdadeiro positivo, falso positivo, falso negativo e verdadeiro negativo, respetivamente.

Comparando os resultados da precisão de classificação, provou-se que o modelo CNN consegue melhores resultados com valores de 91.5% e 90% para nove e vinte e quatro classes de insetos nos *datasets* de Wang e Xie respetivamente. A figura 2.16 demonstra os resultados obtidos para o *dataset* de Wang 2.16(a) e o *dataset* de Xie 2.16(b).



(a) (b)

Fonte: Gráficos retirados de [23]

Figura 2.16: Resultados para o *dataset* de Wang (a) e *dataset* de Xie (b)

Os autores concluíram também que obter um maior valor de precisão em terreno e tempo real é mais complicado devido à presença de sombras, folhas, terra e ramos em terrenos de agricultura extensos.

2.3.3 Deteção e classificação em tempo real de insetos

Relacionado com monitorização em tempo real, Bjerge et al. [4], desenvolveram um sistema e um algoritmo para deteção e classificação em tempo real de visitas de insetos a flores.

Este sistema em termos de *hardware* é caro uma vez que os componentes principais do sistema são um computador NVIDIA Jetson Nano Developer Kit que tem CPU e GPU e uma webcam, Brio Ultra HD Pro Webcam da Logitech. O sistema tem ainda um sistema de refrigeração, com vista a ajudar o computador a arrefecer em dias quentes de verão enquanto a GPU funciona continuamente. As imagens foram armazenadas num SSD com 500GB de espaço. O computador encontrava-se ligado à Internet graças a um router WiFi.

Dados relacionados com posição e identidade dos insetos são transferidos diariamente através da ligação à internet, enquanto todos os dados são guardados em SSD.

Em termos de software, o computador executa dois programas em paralelo. A câmara grava continuamente imagens enquanto em paralelo o sistema corre um modelo treinado de aprendizagem profunda (YOLOv3), que está em tempo real a detetar e classificar os insetos que estão a ser capturados pela câmara. Apenas informação relacionada com posição e identificação é transferida para o posto remoto, enquanto as imagens são guardadas no SSD que se encontra no local, de modo a reduzir a quantidade de dados a enviar na rede. No entanto, apenas imagens que contém deteções de insetos com um limiar de confiança acima de 10% é que são guardadas.

O algoritmo de visão computacional desenvolvido foi designado de Classificação de Insetos e Rastreamento (*Insect Classification and Tracking – ICT*), capaz de rastrear e contar o número de insetos pertencentes a espécies conhecidas. Para cada movimento dum inseto detetado, é guardado o tempo inicial, duração, tamanho média da caixa envolvente definida pelo inseto e identificação do inseto. Este algoritmo consiste em 5 passos, sendo que os dois primeiros ocorrem *in loco* e os restantes ocorrem remotamente. Na primeira parte ocorre a captura de imagens e deteção individual dos insetos nessas mesmas imagens, recorrendo à CNN treinada. O local do inseto na imagem é estimado no segundo passo. A parte remota envolve uma fase de filtragem, em que objetos que não apresentaram movimento são removidos uma vez que provavelmente não são insetos. O passo seguinte é o de rastreamento, e consiste em rastrear o inseto na sequência de imagens. Este passo certifica-se que cada inseto é contado apenas uma vez em cada visita. Cada imagem é associada a uma das classes de insetos definidas e no caso de discórdia entre imagens, ganha a classe com maioria dos votos. O último passo consiste em gerar estatísticas relacionadas com o tempo, duração e distância percorrida de cada inseto.

De maneira a classificar os insetos *in loco*, o modelo utilizado foi a CNN *darknet53* (YOLOv3) com um total de 53 camadas que tem uma precisão média (*mean Average Precision -mAP*) de 57.9% no *COCO dataset*. Esta precisão é comparável com outras redes "meta", mas tem um tempo de processamento inferior. Esta *darknet53* é melhor a detetar objetos pequenos que a rede YOLOv2. O mapa de atributos (*feature map*) da *darknet53* tem coordenadas para a caixa envolvente e um valor de confiança para cada objeto detetado. A resolução escolhida para a imagem foi de 608 x 608 para ser utilizada na rede YOLOv3. Foram exploradas diferentes combinações de tamanhos de imagem, número de iterações de treino e tamanho do filtro do *kernel* de modo a descobrir a rede *darknet53* ótima para classificar oito classes de insetos. O modelo final acabou por usar as definições padrão do YOLOv3, mas o treino foi efetuado com imagens de resolução 832 x 832 e a validação com imagens de resolução 608 x 608 píxeis.

Quanto aos *datasets* utilizados, os *datasets* de treino, validação e teste para o modelo de aprendizagem profunda basearam-se em imagens gravadas durante o verão de 2019 com uma mistura de plantas. Destes dados, imagens de sete espécies que aparecem frequentemente com datas diferentes foram selecionadas, anotadas e identificadas de modo a

treinar o modelo CNN. No total, 2121 imagens de fundo sem insetos e 5757 imagens com insetos foram utilizadas na fase de treino. Para além destas 7 classes associadas a espécies de insetos, existe ainda uma classe à parte denominada "Outros artrópodes" que contém uma mistura de outros insetos como por exemplo moscas, formigas, aranhas, entre outros que ocorrem com menos frequência (247 moscas, <60 de outras taxonomias). Para esta classe e a classe *Aglais urticae*, não foram encontradas imagens suficientes para se criar um *dataset* de teste. Aplicaram-se técnicas de aumento de dados a todas as imagens utilizando as definições padrão do YOLOv3.

Para o treino do modelo, apenas o *dataset* de treino foi utilizado e o número de iterações de treino foi escolhido com base no momento em que a média de erro de treino deixou de descer. O modelo escolhido utilizou então 30000 iterações, o que levou a valores mais altos de precisão e sensibilidade comparativamente a usar menos de 28000 iterações de treino. Um valor mais alto de iterações acabaria por levar a sobreajuste.

Foi utilizado um valor de sobreposição de Interseção sobre União (*Intersection over Union – IoU*) de 20%. Por norma o valor utilizado é de 50%, no entanto neste caso como os insetos são muito pequenos comparativamente à imagem, as caixas envolventes anotadas e previstas podem não se sobrepor mesmo quando o inseto é detetado com sucesso. Para se certificarem que o número previsto de insetos de cada classe se aproximava do número real de insetos (precisão igual a sensibilidade) em cada classe nas imagens, foi ajustado o limiar de confiança de cada classe individualmente. Desta forma, a quantidade de insetos que o modelo não encontrou (falsos negativos) era igual à quantidade de previsões que não continham insetos (falsos positivos). Outra estratégia implicava maximizar a sensibilidade através da redução do limiar de confiança.

Quanto aos resultados, a classe de "Outros artrópodes" obteve o menor valor de precisão e sensibilidade, possivelmente porque esta classe contém anotações com aparência muito diferente. Os valores de precisão, sensibilidade e *mean Average Precision (mAP)* na fase de validação foram calculados tendo em conta o mesmo limiar de confiança da fase de treino. A *mean Average Precision* obtém-se fazendo a média de todas as áreas abaixo de todas as curvas de precisão-sensibilidade (*precision-recall*). O modelo YOLOv3 na fase de validação teve um valor médio de sensibilidade de 73%, indicando que 27% dos indivíduos no *dataset* de validação não foram detetados. Obteve-se um valor de precisão de 72%, indicando que 28% dos insetos foram mal classificados, incluindo falsas deteções em elementos do plano de fundo. Especial atenção para as pequenas moscas *Episyrphus balteaus* e *Eupeodes corolla* que foram difíceis de separar. O baixo valor de 67% para a precisão de classificação de *Eupeodes corolla* deve-se ao facto de 18 indivíduos terem sido mal classificados pelo algoritmo, que deveria ter classificado como *Episyrphus balteaus*. O mesmo padrão pode ser visto na classificação de *Bombus lapidarius* e *Bombus terrestris*.

Uma média de 87% para mAP é considerada alta comparativamente aos 58% do *COCO dataset*. No entanto este último *dataset* contém muito mais classes (80) que as 8 classes do estudo em causa e limiares de Interseção sobre União mais conservadores. O sistema detetou no total 2994 movimentos de insetos ao longo de 98 dias onde o algoritmo ICT

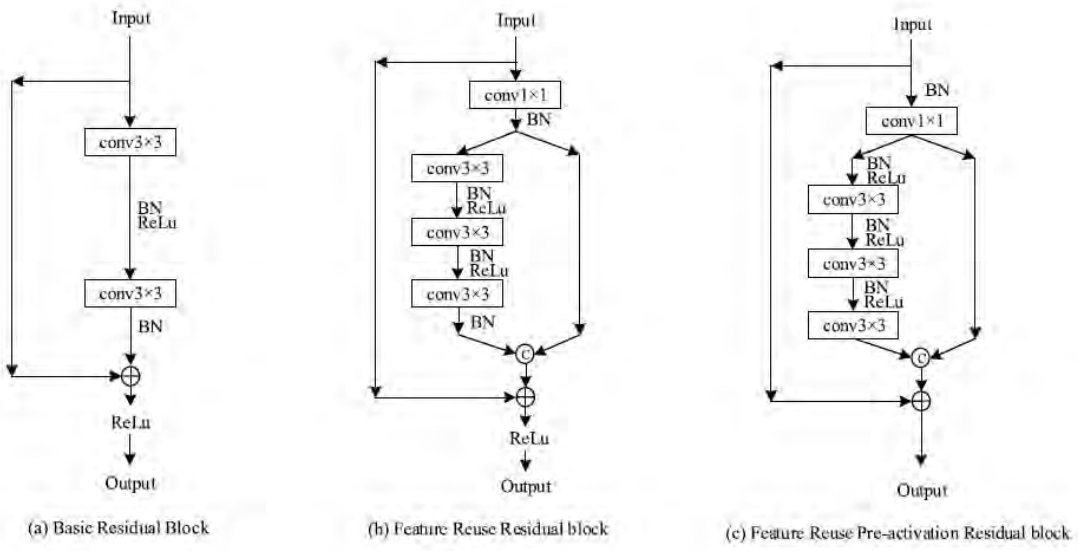
obteve uma precisão média de 89% na classificação correta de espécies. A precisão de classificação foi pior para as moscas das flores devido a muitos falsos positivos em que elementos do plano de fundo eram avaliados como insetos. A espécie mais comum nas imagens foi a abelha *Apis mellifera*, que obteve um valor de precisão de 98%.

2.3.4 Reutilização de atributos para reconhecimento de insetos

Por fim, um estudo de Ren et al. [43] baseou-se numa rede de reutilização de atributos.

Neste estudo os modelos foram treinados com o *dataset* IP102 que é um *dataset* de larga escala para reconhecimento de pragas de insetos. Uma vez que a tarefa de classificação neste *dataset* tem muita em conta a atenção ao detalhe, gera-se a hipótese de reutilização de atributos de forma a melhorar a performance de precisão.

Neste estudo testou-se empilhar blocos de *Feature Reuse Residual* de modo a criar uma rede (*Feature Reuse Residual Network*) denominada *FR-ResNet*. Comparando com o bloco residual original utilizado em *ResNets*, é adicionada uma conexão extra entre o *input* do bloco residual básico e a concatenação de outputs dos dois ramos. Posteriormente o resultado deste processo é passado para o próximo bloco residual de reutilização de atributos, após a operação de soma. Houve ainda a necessidade de reduzir em metade o número de filtros através da utilização duma camada convolucional de 1 x 1, de modo a manter a dimensão original dos atributos e reduzir a complexidade, sendo ainda necessário utilizar três camadas convolucionais em vez de apenas duas. A figura 2.17 demonstra os blocos considerados no estudo.



Fonte: Imagem retirada de [43]

Figura 2.17: Diferentes blocos residuais. Bloco residual básico (a), bloco Residual de Reutilização de Atributos (b), bloco Residual de Reutilização de Atributos com ativação prévia (c)

Antes de se apresentar a proposta para novo bloco residual, é necessário explicar o bloco residual original. Este bloco residual original pode ser expresso pelas seguintes equações:

$$y_l = h(x_l) + F(x_l, w_l) \tag{2.3}$$

$$x_{l+1} = f(y_l) \tag{2.4}$$

onde x_{l+1} e x_l representam o output e *input* do l-ésimo bloco residual na rede, F denota uma função residual e w_l são parâmetros do l-ésimo bloco. A função h(xl) representa um mapeamento de identidade: $h(x_l) = x_l$, e f denota a função ReLU. Blocos residuais sequencialmente empilhados constroem uma rede residual.

O estudo pretendia então explorar os efeitos de reutilização de atributos para o bloco residual original através da adição de uma conexão extra ao sinal de *input* do bloco residual. De modo a igualar tamanho e dimensão os mapas de atributos de *input* passam através de uma camada convolucional 1 x 1 sem uma função de ReLU. De modo a maximizar a performance da rede, experimentaram-se diferentes tamanhos para o bloco residual.

Assim o bloco Residual de Reutilização de Atributos (*Feature Reuse Residual block*) pode ser denotado pelas seguintes fórmulas:

$$y_l = h(x_l) + F(g(x_l), w_l) o g(x_l) \tag{2.5}$$

$$x_{l+1} = f(y_l) \tag{2.6}$$

onde g é uma função que transforma o tamanho e as dimensões do mapa de atributos, o qual é realizado por uma camada convolucional 1 x 1 sem ReLU. Todavia, a redução de tamanho dos atributos tem impacto no erro de teste e este estudo prova que utilizar uma camada de *average pooling* antes da camada convolucional de 1 x 1 no bloco de redução de amostra pode levar a uma melhor performance que outras soluções.

De maneira a otimizar a rede, os autores realçam a necessidade de determinar princípios importantes como o tamanho do bloco residual e o local de redução de tamanho do mapa de atributos. No caso de *ResNets* originais, a unidade residual básica consiste numa pilha de 2 camadas convolucionais de 3 x 3. De maneira a fazer comparação de performance de diferentes tamanhos para blocos residuais, os autores exploraram diferentes tipos de convoluções em todos os blocos residuais com um número total de parâmetros similar. Usam a denotação B(M) para a estrutura do bloco residual e onde M representa a lista de tamanhos de *kernel* das camadas convolucionais num bloco residual. Por exemplo B (1,3,3,3) define um bloco residual com uma camada convolucional 1 x 1 e três camadas convolucionais 3 x 3. Fez-se a experiência no *dataset CIFAR-10* e os resultados mostram que B (1,3,3,3) obtém a melhor performance quando o número de épocas é 164 ou 500.

A experiência demonstrou ainda que a performance varia consoante o local de redução de tamanho do mapa de atributos no bloco de redução de amostra. Foram consideradas 4 variantes: (A) adicionar uma camada de *average pooling* antes da camada convolucional de 1 x 1; (B) adicionar uma camada 3 x 3 de *max pooling* com um *stride* de 2 antes da camada convolucional de 1 x 1; (C) uma camada convolucional de 1 x 1 com *stride* de 2; (D) a primeira camada convolucional 3 x 3 com *stride* de 2. Chegou-se à conclusão que com a variante A obtém-se um menor valor de erro de teste.

Procedeu-se então à classificação do *dataset IP102*. Diferentes aspetos afetam a performance da classificação neste *dataset*, como por exemplo as cores do objeto e do plano de fundo serem similares, o *dataset* contém imagens do ciclo de vida das espécies o que torna difícil a identificação especialmente na altura de larva e por fim as pragas são frequentemente similares entre si. Isto torna este *dataset* desafiante no que toca à tarefa de classificar.

É necessário limitar o número de parâmetros, uma vez que mais planos irão aumentar o número de parâmetros que por sua vez levará a sobreajuste. Assim, limitou-se o número de parâmetros e apenas se utilizou uma rede residual de reutilização de atributos com base no bloco residual básico. Foi utilizado *SGD (Stochastic gradient descent – Gradiente Descendente)* com *mini-batch* de tamanho 64. O ritmo de aprendizagem foi inicializado a 0.01 e dividido por 10 a cada 40 épocas. A decadência do peso é de 0.0005 e o momento é de 0.9. Utilizaram-se ainda técnicas de aumento de dados para a fase de treino. Primeiramente a imagem foi redimensionada para 256 x 256. De seguida uma região retangular é aleatoriamente recortada com uma proporção de tela aleatória no intervalo de [3/4, 4/3] e uma área aleatória no intervalo de [0.08, 1]. Posteriormente a região recortada é redimensionada para uma imagem de 224 x 224. Por fim, é aplicada normalização.

Os modelos foram treinados com o conjunto de treino e avaliada a performance com o conjunto de teste. A *FR-ResNet* foi construída com diferentes tamanhos em termos de profundidade e avaliada a performance neste *dataset* e posteriormente comparada com modelos base de *ResNet*. *A FR-ResNet* foi ainda comparada com vários modelos "meta" como é o caso da *AlexNet*, *ResNet-50*, *ResNet-101*, *Googlenet*, *VGG-16* e *DenseNet121*. Para fazer esta comparação foi calculado o valor de exatidão através da seguinte equação:

$$exatidão = \frac{TP + TN}{TP + TN + FN + FP}$$

tendo ainda sido utilizada a métrica F1, que se obtém com a seguinte equação:

$$F1\ score = \frac{2 \times Precisão \times Sensibilidade}{Precisão + Sensibilidade}$$

Observou-se que a rede *FR-ResNet* teve uma exatidão de 54.73% e um valor F1 de 53.58 no set de teste o que superou os resultados de todos os outros modelos. Uma rede *FR-ResNet* com 50 camadas consegue uma melhor performance que uma de 34 camadas. Uma *FR-ResNet* com 101 camadas consegue um erro de treino menor que a *FR-ResNet*

com 50 camadas, no entanto a exatidão no conjunto de teste é pior que esta última, uma vez que o aumento de parâmetros leva a sobreajuste.

Posteriormente combinou-se o método desenvolvido com diferentes redes residuais: *ResNet*, *Pre-ResNet* e WRN e avaliou-se a performance nos *datasets* de CIFAR-10, CIFAR-100 e SVN, de modo a comprovar a eficácia da maneira como os autores abordaram o problema.

2.4 Conclusão

Pode concluir-se que o presente e o futuro da monitorização de pragas em culturas agrícolas é a monitorização baseada em aprendizagem profunda. Modelos de redes neuronais convolucionais (CNN) conseguem obter melhores resultados no que toca à exatidão de classificação de insetos comparativamente a outros métodos explorados, mas em contrapartida é necessária uma grande quantidade de dados para treinar uma rede deste tipo com o objetivo de obter resultados aceitáveis. Existem técnicas que a partir de um conjunto de dados, conseguem aumentar este conjunto e tornar estes dados menos enviesados utilizando recortes, rotações, translações, entre outras.

No domínio de deteção e classificação de espécies, já existem trabalhos com excelentes resultados, havendo uma redução de exatidão quando esta é efetuada em tempo real, e/ou com pouco pós-tratamento dos dados. Isto deve-se à dificuldade de classificação na presença de sombras, ramos, terra e outros fatores externos. Por norma estas soluções também requerem um investimento económico considerável.

A escolha dos modelos e das métricas de avaliação a utilizar também dependem do problema de classificação que se pretende resolver e da quantidade de dados disponíveis. Assim, de entre os modelos de arquitetura de redes neuronais convolucionais previamente apresentados, o tipo de modelo que mais se aproxima da tarefa que se pretende efetuar, neste caso a contagem de indivíduos de uma espécie presentes naquela imagem de armadilha, serão modelos com foco na classificação de imagens.

3

Metodologia

Este capítulo pretende descrever a metodologia utilizada na dissertação, desde a recolha e processamento dos dados, à utilização dos mesmos no treino dos modelos de classificação utilizados pelo sistema de contagem. Será abordada também a metodologia por detrás do treino e avaliação dos modelos de classificação testados de forma a obter o melhor classificador possível para a nossa tarefa.

Existem várias formas de abordar o problema em questão, no entanto este projeto teve inicialmente o seu foco em classificação de forma binária, ou seja, em cada armadilha apenas procuramos detetar um tipo de espécie, tendo por isso duas classes, uma que afirma que o inseto detetado pertence àquela espécie e outra que afirma que o inseto não pertence àquela espécie, pelo que existirá um modelo para cada tipo de espécie pretendido. Optou-se por esta abordagem, uma vez que existia uma grande diferença no número de indivíduos de cada espécie. De modo a filtrar regiões da placa que possam ter algum tipo de semelhança com um tipo específico de inseto, desenvolveu-se um modelo adicional de classificação de insetos que fazia a distinção entre zonas de placas com insetos de zonas aleatórias de placa sem insetos, na expectativa de obter melhores resultados.

3.1 Obtenção do conjunto de dados

Os dados recolhidos e disponibilizados, continham informação relativa ao ponto de observação biológico (POB), à data de recolha, local da armadilha, a espécie de inseto que se observou na armadilha e a quantidade, comentários adicionais e por fim a imagem em si. Para este projeto foram disponibilizadas 448 imagens de armadilhas de insetos, mais a informação associada a cada uma delas.

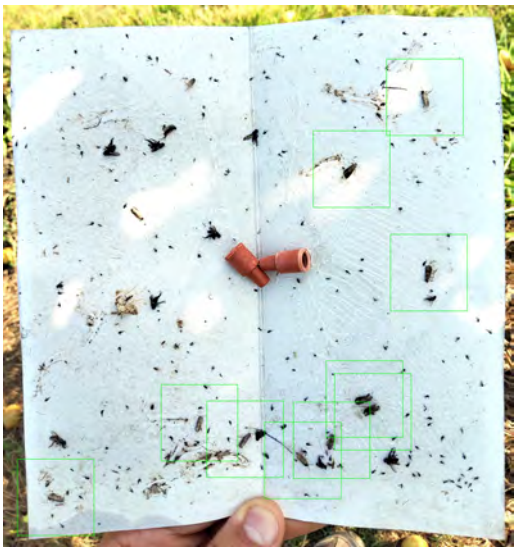
Estes dados necessitavam de pré-processamento, antes de serem utilizados no treino dos modelos de classificação, assim como na fase de classificação por parte dos modelos treinados. Foram efetuados recortes, redimensionamentos e normalização da escala das armadilhas nas imagens, de maneira a facilitar a tarefa de deteção, removendo elementos não importantes para a deteção como por exemplo a área da imagem fora dos limites da armadilha e/ou regiões da placa sem insetos.

Outro aspeto a ter em consideração para o processo de contagem, é o facto de as imagens das armadilhas serem fotografadas semanalmente, pelo que foi necessário ter cautela para não haver contagens duplas.

Na figura 3.1 temos as fotografias retiradas a armadilhas de *Cydia funebrana*, separadas com um intervalo de duas semanas (a primeira fotografia foi tirada a 04/08/2021 e a segunda a 25/08/2021), onde temos a informação que na primeira temos 3 insetos da espécie *Cydia funebrana* 3.1(a) e na segunda já tínhamos 10 indivíduos da espécie *Cydia funebrana* 3.1(b).

Na figura 3.2 temos uma imagem da espécie que se pretende contabilizar.

(a) Fotografia tirada dia 04/08/21. A verde encontram-se marcados os insetos da espécie Funebrana.



(b) Fotografia tirada dia 25/08/21. A verde encontram-se marcados os insetos da espécie Funebrana.

Figura 3.1: Fotografias tiradas à mesma armadilha de *Cydia funebrana* em dias diferentes

Figura 3.2: *Cydia funebrana*

Como se pode notar, alguns problemas que poderiam condicionar a qualidade da

deteção e classificação são a presença de outros insetos para além daquele que de facto se pretende contabilizar, assim como a presença de terra e outros objetos, e ainda o facto da fotografia da armadilha não ser tirada nas mesmas condições para todas as fotografias, levando a variações nas imagens que alimentarão os algoritmos de classificação.

Assim, uma vez que o nosso conjunto de dados inicial se baseava em imagens de armadilhas que continham mais do que um tipo de insetos e mais do que um inseto por tipo, o nosso objetivo inicial passou por extrair cada inseto presente na imagem de modo a obter um conjunto de dados robusto com o objetivo de treinar modelos de classificação para cada tipo de espécie com um tipo de armadilha designado. Para tal, criaram-se programas auxiliares de modo a tornar a extração uma tarefa mais simples, e que a partir de uma imagem fotografada de uma armadilha, permitiu a extração de cada um dos insetos presentes na imagem. Esta extração consistiu em gerar caixas envolventes, através das coordenadas do clique do rato, nas quais os insetos se encontravam próximos do centro das mesmas, para que posteriormente obtivéssemos imagens com as dimensões corretas e centradas nos insetos pretendidos. Posteriormente removeram-se casos de contagens duplas de insetos.

3.2 Metodologia de treino e avaliação dos modelos

O objetivo desta secção é explicar todo o processo por detrás da fase de treino e posteriormente de escolha dos modelos de classificação que serão testados de modo a serem utilizados depois na aplicação de contagem automática de insetos. Para tal esta secção será divida em duas fases, sendo apresentado primeiro a metodologia utilizada na fase de treino das arquiteturas que foram testadas, onde recorremos a um conjunto de dados para treinar e validar cada um dos modelos e posteriormente será apresentada a metodologia utilizada na fase de avaliação de cada um dos modelos de forma a compreender a performance dos modelos treinados na tarefa de classificação em questão, sendo que para tal recorremos a métricas de avaliação bem conhecidas no ramo da aprendizagem profunda.

3.2.1 Fase de treino do modelo

Para começar a fase de treino dos modelos, foi necessário fazer a divisão dos conjuntos de dados em conjuntos de treino, validação e teste. Uma vez que o nosso projeto tem 4 modelos distintos, um modelo de identificação de insetos, um modelo de identificação de sésias, um modelo de identificação de funebranas e um modelo de identificação de bichados, a metodologia utilizada para cada um destes modelos é idêntica, mudando apenas a quantidade de dados utilizada para treino, validação e teste para cada um dos modelos. A divisão efetuada no conjunto de dados, foi a seguinte: 80% dos dados do conjunto de dados total, foram utilizados para treino e validação dos modelos, sendo os restantes 20% utilizados como conjunto de teste na fase de avaliação dos modelos. A

escolha dos dados que farão parte do conjunto de treino ou do conjunto de teste foi uma escolha aleatória efetuada com o auxílio da função *train_test_split* da biblioteca *sklearn*[41]. Quanto ao conjunto de validação, este foi obtido a partir do conjunto de treino utilizando *StratifiedKFold* da biblioteca *sklearn*. Uma vez que o número de partições do nosso projeto é 5, isso significa que 20% do conjunto treino é utilizado para validação sendo que os restantes 80% são utilizados para realmente treinar o modelo. Estas partições permitem também que no momento de treino dos modelos, o modelo seja validado e treinado com conjuntos de dados diferentes em diferentes iterações, com o objetivo de obter um modelo mais robusto e capaz de generalizar.

O treino dos modelos foi efetuado numa máquina local com uma placa gráfica NVIDIA GeForce GTX 1660 Ti e com RAM de 16GB e com o auxílio da biblioteca *Keras*[7]. A função *fit()* efetua o treino de um modelo previamente construído. Para tal, enviamos para esta função os conjuntos de treino e validação que no nosso caso não foram necessariamente os conjuntos originais, mas sim geradores de imagens, conceito que será explicado posteriormente. Necessitamos ainda de especificar durante quantas épocas o modelo irá treinar sendo que especificámos, também com recurso a parâmetros desta função, que pretendemos parar o treino mais cedo caso uma condição se verifique e que pretendemos guardar o estado do modelo sempre que este apresentar melhores resultados que anteriormente.

Este processo de parar o treino do modelo antes de se chegar ao número de épocas pré-definido é conhecido como *early stopping* e é uma forma de evitar que o nosso modelo fique sobreajustado ao conjunto de dados de treino. A biblioteca *Keras* oferece a possibilidade de definir uma função com este intuito, sendo que temos de definir a métrica que pretendemos avaliar e durante quantas épocas permitimos que esta métrica não apresente melhorias, ou seja, se a métrica que está a ser monitorizada não apresentar melhorias após x épocas, então paramos o treino do modelo. Este valor de x é conhecido como paciência e para o nosso projeto decidimos dar um valor de 10, sendo que monitorizámos o valor de perda do nosso conjunto de validação. Definimos ainda uma função que guardava automaticamente o estado do modelo a partir do momento em que este apresentasse um valor de perda no conjunto de validação menor do que aquele alcançado pelo modelo até ali guardado.

Estando definido a maneira como o nosso modelo foi guardado e a maneira de como o nosso treino foi parado mais cedo se necessário, treinámos o nosso modelo durante 100 épocas, o que acabou por ser um valor por acrescento no sentido em que os nossos modelos não foram treinados durante as épocas todas, acabando por parar via *early stopping*.

Ainda no tópico de sobreajustamento dos dados, falaremos agora no processo que ocorre entre o momento de divisão do conjunto de dados em conjunto de treino, validação e teste e o momento de início de treino do modelo. Com o auxílio da classe *ImageDataGenerator* da biblioteca *Keras*, temos a capacidade de efetuar *data augmentation* em tempo real, ou seja, em cada época de treino do nosso modelo, teremos imagens que sofreram

processos de *data augmentation*, sendo esses processos alterações na luminosidade da imagem, rotações na imagem, reflexões, alterações de perspetiva e recortes. Para a função de treino do modelo é então passado como argumento, não as imagens originais de treino e validação, mas sim um gerador de imagens, que acabará por enviar um conjunto de transformações efetuadas às imagens originais. O número de imagens geradas em cada época é o número de imagens pertencentes ao nosso conjunto de treino ou conjunto de validação naquela iteração a dividir pelo número correspondente ao tamanho dos *batches*, que no nosso caso esse número foi 32 ou 16. Isto sendo uma forma de *data augmentation* acaba por reduzir a possibilidade de sobre ajustamento de dados e torna o modelo mais robusto.

É importante referir que os modelos foram treinados 5 vezes, número equivalente ao número de partições, sendo que posteriormente se fez a média dos resultados referentes à *accuracy* no conjunto de validação de maneira a escolher qual a melhor arquitetura a utilizar.

Por fim, falta mencionar como foi efetuada a escolha do modelo a utilizar na tarefa de classificação. A escolha do modelo teve em conta 4 fatores diferentes: A arquitetura do modelo, o optimizador utilizado, qual a taxa de aprendizagem utilizada juntamente com o optimizador e por fim qual o número de blocos "descongelados" de forma a permitir o treino destas camadas com o objetivo de alcançar os melhores resultados de classificação. De forma a ser possível tirar conclusões é necessário fixar outros parâmetros que possam fazer variar os nossos resultados pelo que, começámos por utilizar *batches* de 32 no treino dos modelos, com o optimizador *Adam* com uma taxa de aprendizagem de 0.00001 e não permitimos o treino de blocos pertencentes às camadas de extração de atributos. Posteriormente quando obtivermos os resultados deste treino podemos variar um dos parâmetros que fixámos nesta fase, fixando agora a arquitetura que obteve melhores resultados. Quanto à escolha do modelo, esta teve como prioridade o maior valor de *accuracy* obtido no conjunto de validação, sendo que no caso de valores iguais optou-se em termos de arquitetura por aquela que é mais leve e necessita de menos tempo para efetuar inferências. Quanto a "empates" no caso dos optimizadores, não há um critério rigoroso de escolha, assim como no caso das taxas de aprendizagem, enquanto no caso dos blocos, será mais vantajoso o treino de menos blocos.

O diagrama presente na figura 3.3 pretende resumir a metodologia utilizada no treino dos modelos.

3.2.2 Avaliação do modelo

Com o término do treino dos modelos, é agora necessário avaliar a performance do modelo escolhido em dados que este ainda não conhece, ou seja, o conjunto que separámos ao início a que se dá o nome de conjunto de teste. Este conjunto não sofreu alterações uma vez que o seu objetivo é aproximar-se o máximo possível de um caso real. O modelo irá fazer uma previsão recebendo uma imagem deste conjunto sendo posteriormente

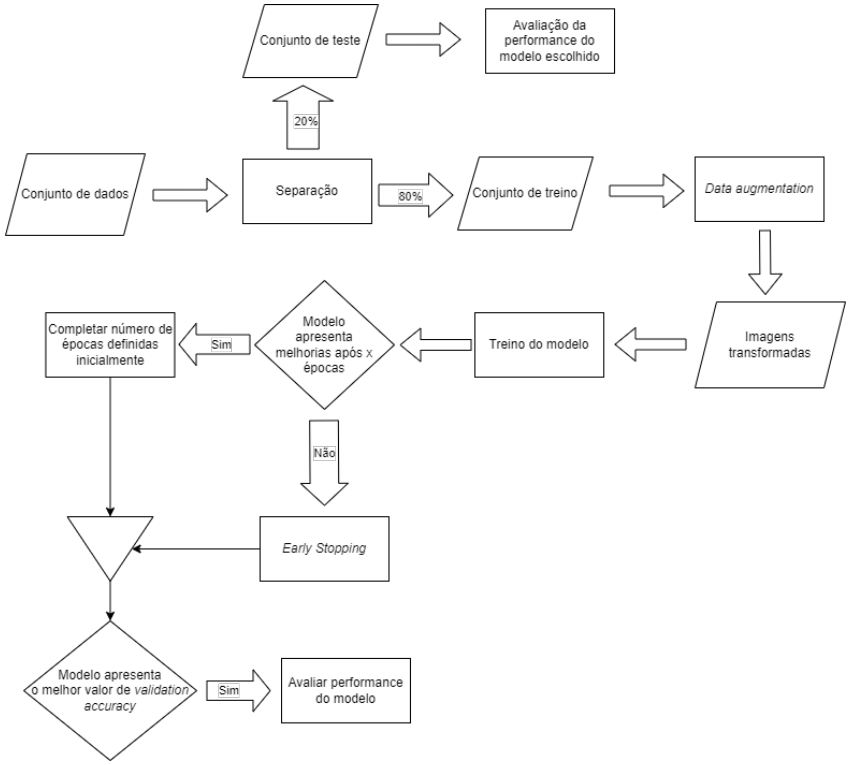


Figura 3.3: Diagrama associado à metodologia utilizada

comparada a classificação efetuada pelo modelo com o valor real da imagem. Dependendo do resultado poderemos estar na presença de um verdadeiro positivo (VP), caso a previsão do modelo é de que estamos na presença da classe positiva e essa previsão coincide com a *ground truth*, um falso positivo (FP), caso o modelo diga que estamos na presença da classe positiva mas na realidade estamos na presença da classe negativa, um falso negativo (FN), caso a previsão do modelo seja de que estamos na presença da classe negativa mas na realidade a classe é positiva e ou um verdadeiro negativo (VN), caso o modelo tenha previsto corretamente que estamos na presença da classe negativa. Na tabela 3.1 temos uma ilustração de como se interpreta um resultado da previsão do nosso modelo, aplicado à classe Sésia.

Tabela 3.1: Matriz de confusão em tabela aplicada ao nosso problema.

Caso	**Realidade**	**Previsão do Modelo**	**Isto é um...**
1	Sésia	Sésia	Verdadeiro Positivo
2	Sésia	Não é Sésia	Falso Negativo
3	Não é Sésia	Sésia	Falso Positivo
4	Não é Sésia	Não é Sésia	Verdadeiro Negativo

Tendo esta informação associada a cada uma das previsões feitas pelo nosso modelo para o conjunto de teste podemos recorrer a métricas de avaliação de performance. A forma mais direta de avaliação de performance é a matriz de confusão, na qual é possível

mostrar em forma de tabela o número total de verdadeiros positivos, falsos positivos, falsos negativos e verdadeiros negativos. Uma vez que o objetivo do modelo é ir de encontro à realidade, pretende-se ter verdadeiros positivos e verdadeiros negativos, evitando ter falsos positivos e falsos negativos, podendo em alguns casos considerar-se mais importante um número mais baixo de falsos negativos comparativamente a falsos positivos ou vice-versa.

Foram utilizadas métricas de avaliação, as quais não são tão visualmente diretas como a matriz de confusão, mas que utilizando a fórmula que as define nos darão um valor entre 0 e 1, sendo que quanto mais próximo de 1 for o valor, melhor a performance do modelo. Assim, temos uma ideia se o modelo treinado se adequa ao problema de classificação ou não.

Dentro dessas métricas temos a precisão e a sensibilidade como base, que nos permitem perceber se o nosso modelo se adequa ao problema que pretendemos resolver. No caso da precisão, esta métrica é mais relevante quando se pretende que o nosso modelo não produza um elevado número de falsos positivos. Já no caso da sensibilidade, esta métrica pode ser considerada mais relevante quando se pretende um número reduzido de falsos negativos. Por fim, quando se pretende um equilíbrio em termos de falsos positivos e falsos negativos, a melhor métrica a utilizar é a métrica F1, que utiliza a média harmónica das duas métricas anteriormente referidas.

Podemos agora apresentar as fórmulas utilizadas para calcular as métricas de avaliação de performance.

$$Precisão = \frac{VP}{VP + FP} \tag{3.1}$$

$$Sensibilidade = \frac{VP}{VP + FN} \tag{3.2}$$

$$F1\ score = \frac{2 \times Precisão \times Sensibilidade}{Precisão + Sensibilidade} \tag{3.3}$$

A equação 3.1 é utilizada de modo a calcular a precisão do nosso modelo. Esta métrica mede, de entre os casos positivos previstos pelo modelo, que porção destes são realmente positivos. Aplicada ao nosso problema, esta métrica seria a mais relevante caso o objetivo mais importante para os agricultores fosse ter o mínimo de falsos positivos possível, ou seja, um número elevado de previsões de insetos de espécie poderia levar os agricultores a tomar medidas que acabariam por ser desnecessárias.

A equação 3.2 utiliza-se para calcular a sensibilidade do nosso modelo. Esta métrica mede, de entre todos os casos que são realmente positivos, que porção destes são previstos como positivo pelo nosso modelo. Esta métrica é também conhecida como taxa de verdadeiros positivos. No nosso problema esta métrica é certamente relevante, uma vez que uma taxa elevada de falsos negativos significaria que o problema existe (número de insetos acima dum limite considerado desprezável) mas que o mesmo não estaria a ser detetado, pelo que um valor alto de sensibilidade seria certamente desejado.

Por último, a equação 3.3 é a média harmónica das métricas anteriormente descritas, e ajuda a avaliar o nosso modelo de uma forma geral. Esta métrica tem em conta, tanto os falsos positivos como os falsos negativos. No nosso problema foi a métrica que considerámos mais relevante, uma vez que tem em conta tanto falsos negativos como falsos positivos, esperando ter um valor baixo para ambos os tipos de erro.

4

Implementação

O objetivo deste capítulo é explicar e ilustrar o trabalho efetuado de modo a atingir os objetivos previamente definidos.

O objetivo do programa desenvolvido é, recebendo uma imagem como *input*, devolver no *output*, a contagem de insetos de uma espécie que se pretende monitorizar presentes nessa mesma imagem. Tendo isto em conta, torna-se necessário efetuar alguns processos que facilitem o processamento da imagem, de forma a isolar os insetos presentes na imagem de forma a que posteriormente seja possível classificar os mesmos de forma a diferenciar os mesmos com o objetivo final de efetuar a contagem daqueles que realmente nos interessam.

É com esta ideia em mente que se efetuou a seguinte sequência de passos: Fornecimento da imagem como *input*, segmentação da imagem recebida, recorte da imagem por regiões de interesse, classificação das regiões de interesse geradas e por fim contagem das regiões de interesse classificadas como sendo o inseto da espécie que se pretende monitorizar.

Para a parte de classificação foi necessário treinar um modelo de classificação, através de aprendizagem profunda. Assim, utilizou-se a metodologia de obtenção de dados de forma a gerar dados para os conjuntos de treino e validação do nosso modelo.

Nas próximas secções serão abordados em detalhe cada um dos passos já enumerados.

4.1 Segmentação e construção do conjunto de dados

Inicialmente a imagem não sofreu alterações, como é o caso de rotações, redimensionamentos ou cortes, pelo que os passos posteriores de processamento de imagem foram efetuados sobre a imagem original.

O primeiro processo efetuado foi o de segmentação da imagem. A imagem foi transformada numa escala de cinzentos e posteriormente testaram-se diferentes métodos de segmentação de imagem. Os métodos testados foram segmentação com valores de *threshold* introduzidos manualmente e com "passo" de 0.1 e métodos de segmentação automática, dentro da qual se testou *Otsu's Thresholding*[40], *Niblack Thresholding*[38], *Sauvola*

Thresholding[47] e por fim o algoritmo de segmentação de Chan-Vese[14].

Todos os métodos de segmentação foram testados com a imagem presente na figura 4.1. Optou-se por esta imagem, uma vez que continha uma boa quantidade de insetos a identificar e a placa da armadilha não estava totalmente suja nem totalmente limpa, permitindo testar com eficácia métodos de segmentação.

Figura 4.1: Imagem original testada. A verde estão marcados os indivíduos da espécie Sésia.

4.1.1 Segmentação Manual

Esta forma de segmentação de imagem é a mais simples. Neste método de segmentação é definido um limite (*threshold*), sendo que para todos os pixéis da imagem é avaliado o seu valor. Se este valor estiver abaixo do limite escolhido, então o valor do pixel em questão é alterado para 0. Caso contrário o valor é atualizado para 1.

O resultado desta segmentação aplicada à nossa imagem é apresentada na figura 4.2.

4.1.2 *Otsu's Thresholding, Niblack Thresholding* e *Sauvola Thresholding*

Algoritmos de segmentação através de *thresholding* automático seguem por norma os mesmos passos: processamento da imagem de *input*, criação de histograma de distribuição de pixéis, computação do limite a utilizar e alteração dos pixéis consoante o limite. É na parte de computação do limite em que os diferentes métodos começam a divergir.

No método de Otsu, o objetivo é descobrir o limite no qual a variância entre o primeiro plano e o plano de fundo é mínimo. Assim o método passa por iterar todos os valores

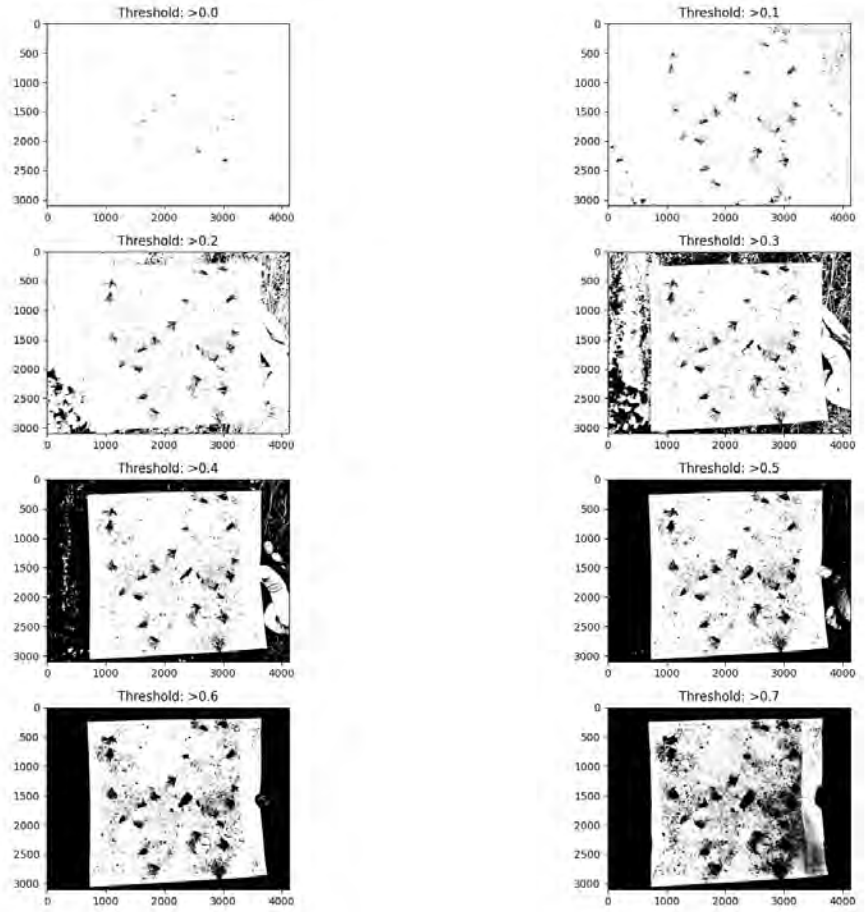


Figura 4.2: Segmentação manual

possíveis para o limite e escolher o limite para o qual a variância entre os pixéis de primeiro plano e plano de fundo é menor. Escolhido o limite, o processo é idêntico ao da segmentação manual.

Os métodos de Niblack e Sauvola são métodos de limitação local, úteis para imagens com planos de fundo não uniformes. Nestes métodos, em vez de se calcular um limite para ser utilizado em toda a imagem, calcula-se para cada pixel vários limites utilizando a média e o desvio padrão da vizinhança do pixel(esta vizinhança é definida por uma janela em torno do pixel). O método de Sauvola, pegou no método de Niblack e tentou resolver o problema do ruído no caso de janelas vazias.

Na figura 4.3 é apresentado o resultado dos métodos previamente descritos aplicados à nossa imagem.

4.1.3 Segmentação de Chan-Vese

Por fim, o último método de segmentação testado foi o algoritmo de Chan-Vese. Este método é utilizado para segmentar objetos sem fronteiras bem definidas. O algoritmo

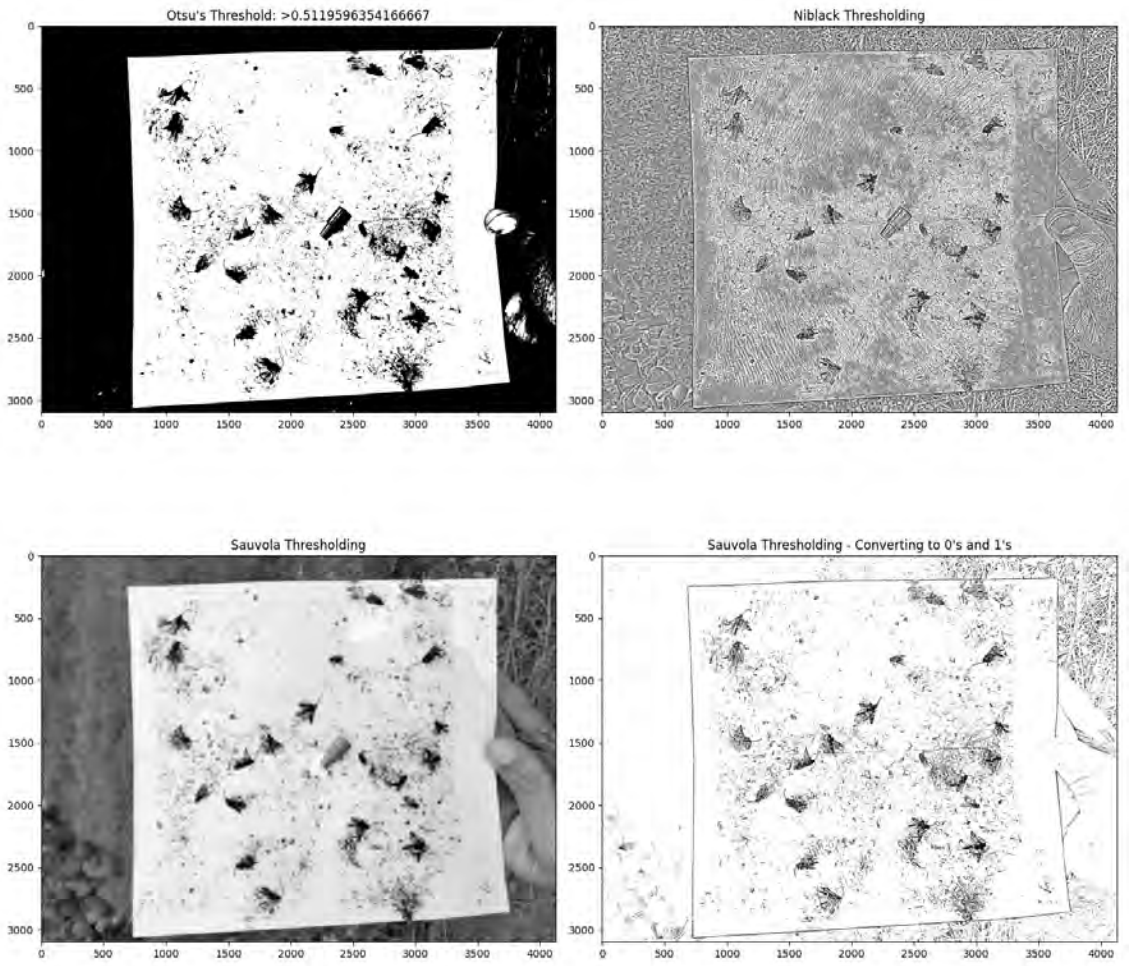


Figura 4.3: Métodos de segmentação automática: *Otsu's Thresholding*, *Niblack Thresholding* e *Sauvola Thresholding*

baseia-se em conjuntos de níveis que evoluem iterativamente de modo a minimizar uma função energética. Este método acaba por ser computacionalmente mais caro que outros, no entanto acaba por obter melhores resultados comparativamente a outros métodos disponíveis.

A figura 4.4 ilustra o método de Chan-Vese aplicado à nossa imagem de teste.

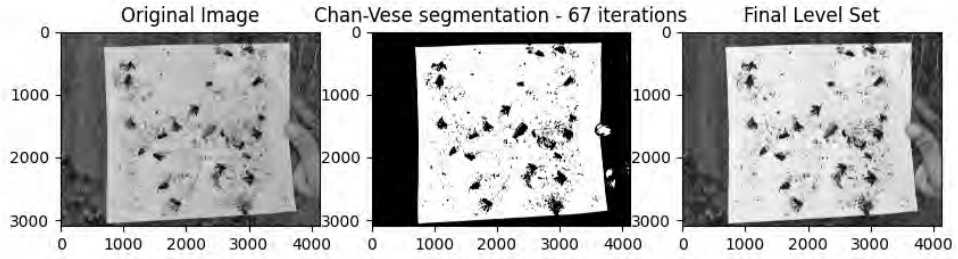


Figura 4.4: Segmentação de Chan-Vese

4.2 Alterações morfológicas de filtragem e função *findContours()*

É importante mencionar que os testes dos métodos de segmentação foram efetuados para 6 imagens diferentes, as quais podem ser observadas na figura 4.5.

Perante os resultados dos diferentes métodos de segmentação testados, para os próximos passos acabou-se por escolher apenas três dos métodos previamente descritos. Estes métodos são a segmentação manual com o limite marcado a 0.1, o método de *Thresholding* de Otsu e o algoritmo de segmentação de Chan-Vese, uma vez que para as imagens testadas estes foram aqueles em que os insetos se apresentavam com maior relevo perante o fundo e o meio envolvente, o que acabava por reduzir também o ruído presente na imagem.

Selecionados os três métodos iniciais de segmentação, procederam-se a alterações morfológicas de filtragem. Este tipo de alterações estão relacionadas com a forma e morfologia de atributos numa imagem. Para qualquer que seja a operação morfológica aplicada é necessária uma forma que representará a vizinhança em torno dos pixéis da imagem, geralmente denominada de "*footprint*". No nosso caso a forma utilizada é um disco, sendo que foram testados diferentes valores para o raio do mesmo.

Quanto às alterações morfológicas utilizadas foram as seguintes:

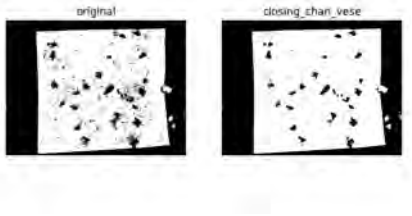
Figura 4.5: Imagens utilizadas para testar métodos de segmentação

- **Erosão:** Consiste em atualizar o valor do pixel para o mínimo de todos os pixéis presentes na vizinhança escolhida. Esta alteração faz com que buracos e espaços entre diferentes regiões fiquem com maior relevo e pequenos detalhes sejam eliminados.
- **Dilatação:** Consiste em atualizar o valor do pixel para o máximo de todos os pixéis presentes na vizinhança escolhida. Esta alteração faz com que buracos entre regiões diferentes fiquem mais pequenos e pequenas intrusões na fronteira de uma região são preenchidas.
- **Abertura:** Consiste na operação morfológica de erosão, seguida da operação morfológica de dilatação. Nesta alteração as regiões que sobrevivem à operação de erosão são repostas ao seu tamanho original através da operação de dilatação.
- **Fecho:** Consiste na operação morfológica de dilatação, seguida da operação morfológica de erosão. Esta alteração consegue "tapar buracos"nas regiões mantendo o tamanho original das regiões.

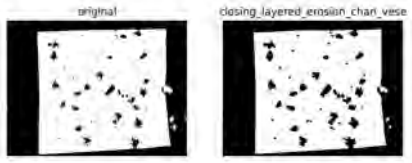
Assim submeteu-se a nossa imagem de teste a uma simples operação de erosão e dilatação, assim como de abertura e ainda a uma simples operação de fecho. Testou-se ainda a possibilidade de acumular operações morfológicas de filtragem.

Na figura 4.6 apresentamos o resultado deste tipo de operações à nossa imagem de teste após ser segmentada com o algoritmo de Chan-Vese. Em 4.6(a) a operação executada foi a de fecho, sendo que o disco da vizinhança tinha raio 15, e a operação foi executada sobre a imagem segmentada com o algoritmo de Chan-Vese. Na imagem 4.6(b) temos um exemplo de conjunção de operações, sendo que neste caso foi executada primeiro uma operação de fecho, seguida de uma operação de erosão e a vizinhança trata-se de um disco de raio 12.

Com estas operações de alteração de morfologia de filtragem, pretende-se reduzir a quantidade de ruído presente na imagem, pelo que após a aplicação destas operações,



(a) Operação de fecho com "*footprint*"de raio 15



(b) Operação de fecho seguida de uma operação de erosão com "*footprint*"de raio 12

Figura 4.6: Alterações de morfologia de filtragem

escolheram-se aquelas operações que reduziram uma boa quantidade de ruído, não afetando o contorno dos insetos presentes na imagem, reduzindo então as transformações de morfologia de filtragem a um conjunto de apenas oito. Após esta experiência, chegou-se à conclusão que a segmentação através do algoritmo de Chan-Vese apresentava de uma forma geral melhores resultados, sendo que os testes posteriormente se efetuaram com apenas este tipo de segmentação inicial.

Após a aplicação de operações de morfologia de filtragem, utilizou-se o algoritmo de procura de contornos da biblioteca *OpenCV*[5] (*findContours()*), de modo a obtermos os contornos presentes nas imagens que tinham agora menos ruído. Basicamente, este algoritmo procura por alterações no valor de intensidade dos pixéis e marca essa alteração como sendo um contorno. Para o caso do nosso programa os parâmetros utilizados na função *findContours()* são: a imagem previamente segmentada e que sofreu as alterações de morfologia de filtragem, *RETR_CCOMP* que recolhe todos os contornos descobertos e os organiza de forma hierárquica em dois níveis e *CHAIN_APPROX_SIMPLE* que simplesmente significa que em vez de se guardar os pontos todos do contorno, os pontos considerados redundantes são desprezados.

Na figura 4.7 temos o resultado do algoritmo de procura de contornos aplicado à imagem original.

4.3 Recorte

Após termos uma lista com todos os contornos encontrados pelo algoritmo, o objetivo era agora extrair os insetos da imagem através do seu contorno. Isto implicava a criação de várias regiões da imagem, de maneira a obter o número de regiões com um inseto da espécie pretendida presente nessa regiões. No final, o número de regiões com o inseto pretendido será igual ao número total de insetos pretendidos presentes na imagem. Para tal, utilizou-se uma função da biblioteca *OpenCV* de nome *boundingRect()*, que recebendo um contorno devolve a coordenada x e y do canto superior esquerdo e a largura e comprimento do retângulo que cobre o conjunto de pontos pertencente a esse contorno. Por fim procura-se o retângulo previamente descrito na imagem original de modo a efetuar

Figura 4.7: Contornos aplicados à imagem original

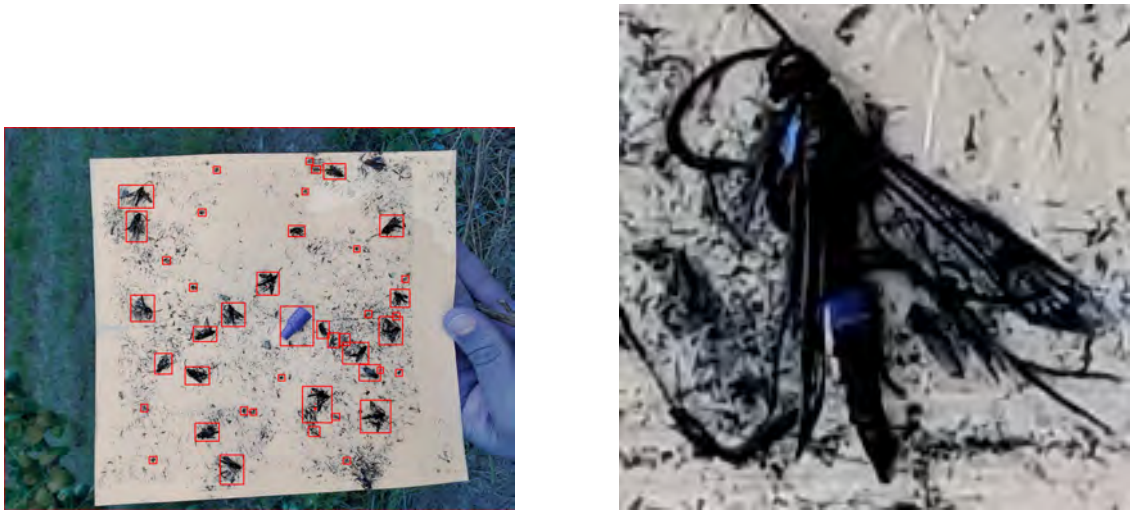
o recorte da região em questão. Repete-se o processo para todos os contornos gerados de modo a ter a imagem original transformada em regiões e guarda-se o resultado localmente.

Resumindo, tendo encontrado os contornos presentes na imagem, percorre-se a lista de todos os contornos e gera-se um retângulo para cada um deste contornos, o qual irá servir para posteriormente se recortar esta região da imagem.

Na figura 4.8 mostramos o resultado do algoritmo previamente descrito. Em 4.8(a) temos uma imagem que demonstra os retângulos gerados pelo algoritmo enquanto que em 4.8(b) é apresentada uma região exemplo que resultou do recorte de um dos retângulos previamente gerados.

Decidiu-se também remover todos os retângulos cuja área era menor que $1600px^2$, visto que o tamanho dos insetos que nos interessavam acabava por ser superior à área escolhida.

Tendo o algoritmo de recorte construido, corremos o programa agora com diferentes imagens de teste de modo a uma vez mais reduzir o modo de segmentação apropriado. O grupo de oito possibilidades foi agora reduzido a duas após se avaliar as regiões recortadas pelo algoritmo. Por essa razão, optou-se por uma alteração de morfologia de fecho com valor para a *"footprint"* de 15 e de uma alteração de fecho seguida de uma alteração de



(a) Retângulos gerados para a nossa imagem

(b) Exemplo de um recorte de uma região

Figura 4.8: Resultado do processo de geração de retângulos e posterior recorte de uma região.

erosão com o valor para a "*footprint*" de 12.

4.4 Remoção do plano de fundo

A abordagem anterior levava a que insetos que se encontrassem na fronteira entre o plano de fundo e a placa da armadilha acabassem por ser ignorados, sendo o contorno do inseto desenhado por fora do mesmo. Assim, numa tentativa de reduzir a área ocupada pelo plano de fundo, decidiu-se recortar a imagem pelo contorno da placa que compõe a armadilha de insetos. A imagem resultante acabará por ter menos pixéis para processar e a área ocupada pelo plano de fundo acaba por ser menor.

Com esta ideia em mente, testaram-se várias técnicas que acabariam por detetar a posição da placa na imagem e efetuar um recorte na imagem através do contorno da mesma. Esta abordagem acaba por poder ser dividida em duas fases distintas com diferentes técnicas para cada uma das fases.

4.4.1 Fase de geração de máscara

A fase inicial da nossa abordagem passa por, a partir da imagem original, gerar uma máscara que filtre o plano de fundo da nossa placa de modo a remover o mesmo da nossa imagem. Esta máscara terá os pixéis pertencentes ao plano de fundo a preto e os pixéis do objeto pretendido a branco.

4.4.1.1 Máscara via *Thresholding*

Nesta abordagem, o algoritmo começou por ler a imagem num canal a três cores utilizando o modelo BGR (azul, verde e vermelho). De notar que o canal mais comum é

o RGB, no entanto com a biblioteca utilizada *OpenCV*, a imagem é lida no formato BGR. A esta imagem foi aplicado um desfoque gaussiano, de forma a reduzir a quantidade de ruído presente na imagem o que ajudará nas próximas fases do algoritmo. De seguida, a imagem agora desfocada foi convertida para o modelo de cor HSV (tonalidade, saturação e valor) utilizando uma função disponibilizada pela biblioteca *OpenCV*. Recorreu-se a este modelo de cor uma vez que a placa que serve de armadilha é sempre de cor branca[1], sendo que o plano de fundo tende a ser folhagem pelo que o conjunto de saturação e valor seria suficiente para filtrar a nossa placa do plano de fundo.

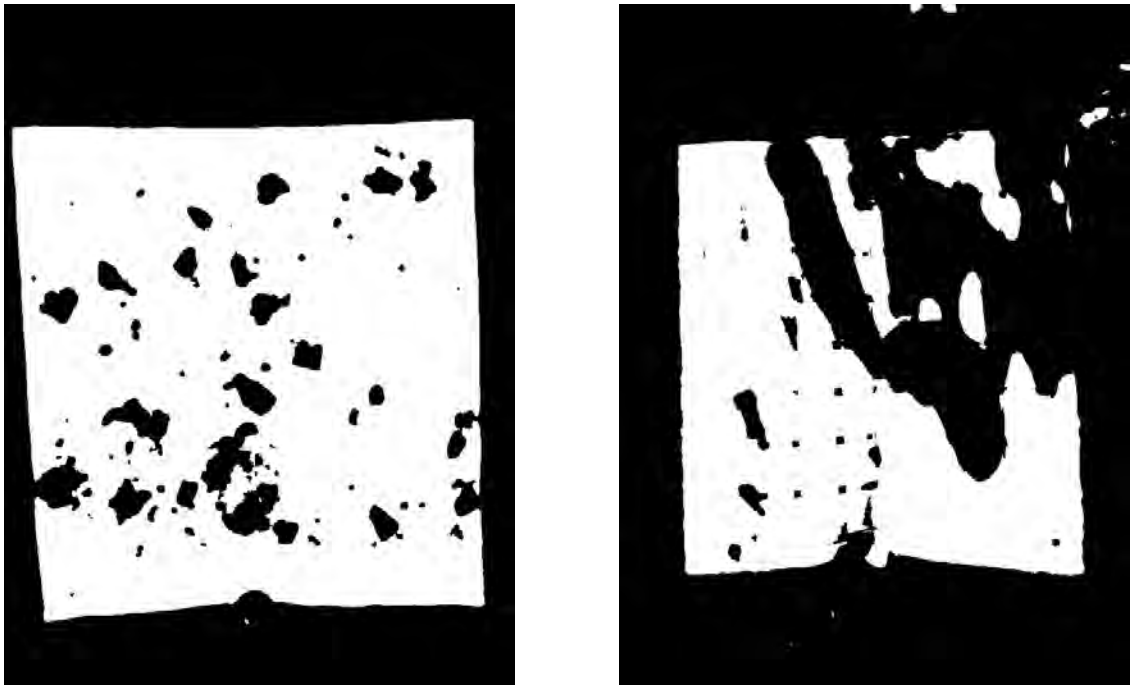
Procedeu-se então à recolha dos canais respetivos à saturação e valor da imagem de modo a gerar um histograma de valores. Neste histograma é possível visualizar os valores para pixéis que ocorrem com maior frequência na imagem. Deste modo, através da análise do histograma pretende-se filtrar os valores de saturação e valor mais frequentes uma vez que grande parte da área ocupada na imagem será ocupada pela placa que pretendemos recortar. Por essa razão procedeu-se à recolha dos máximos e mínimos locais do histograma. Para obter a máscara binária referente à saturação, utilizou-se o primeiro mínimo local após o máximo global do histograma referente à saturação. Utilizou-se este valor posteriormente na função *threshold()* da biblioteca *OpenCV*, passando como argumentos um vetor com os valores de saturação para os pixéis da imagem, o mínimo local, 255 e por fim qual o tipo de *threshold* a efetuar, que neste caso foi *THRESH_BINARY_INV*. Efetuou-se o mesmo processo para obter a máscara binária referente ao valor, sendo que neste caso escolheu-se o máximo local que se encontrasse entre os valores 130 e 160, caso contrário o valor escolhido seria 150. Utilizou-se uma vez mais a função *threshold()* com um vetor com os valores de valor para os pixéis da imagem, o máximo local ou 150, 255 e por fim o *threshold THRESH_BINARY*. Finalmente efetuou-se o "E" lógico das duas máscaras para obter a máscara final.

Pelos resultados obtidos, deu para perceber que o algoritmo funcionava extremamente bem nalguns casos, como demonstrado na imagem 4.9(a), no entanto para fotos em que havia grande desequilíbrio entre luminosidade, por exemplo placa à sombra e plano de fundo extremamente iluminado, o algoritmo apresentava falhas, levando a que existissem zonas pretas na zona da placa, resultantes de sombras presentes na imagem original, zonas as quais impossibilitavam recolher o contorno preciso da placa numa fase posterior. Um exemplo de um mau resultado está ilustrado na imagem 4.9(b). Decidiu-se testar então outras formas de obter esta máscara.

4.4.1.2 Máscara via *Kmeans*

Uma vez que a técnica anterior levava a que a máscara surgisse com imperfeições nos casos previamente descritos, testou-se outro tipo de abordagem. Foi então experimentado o método *kmeans* da biblioteca *OpenCV*.

[1] Existem placas de armadilhas de outras cores, no entanto não foram consideradas devido à falta de exemplos



(a) Exemplo de uma boa máscara binária (b) Exemplo de uma má máscara binária

Figura 4.9: Resultados do processo de obtenção de máscara binária via *thresholding*

O algoritmo *K-Means*[31] consiste em separar os dados em diferentes grupos, sendo no nosso caso uma divisão de pixéis da imagem. Esta divisão é efetuada pixel a pixel sendo atribuído inicialmente um grupo a cada pixel. De seguida, define-se um centróide para cada grupo de pixéis e calcula-se para cada pixel qual o grupo cujo centróide se encontra mais próximo desse pixel. Repete-se o processo até que não haja mudanças de grupo para os pixéis. No caso do método utilizado, pode-se designar um critério de paragem do algoritmo associado ao número de iterações a efetuar e/ou um valor de precisão mínimo. No nosso caso utilizámos um valor máximo de iterações de 100 e um valor de precisão de 0.2. O valor de K selecionado foi 10, sendo que este valor representa o número de grupos em que os pixéis da imagem serão divididos. Após o término do algoritmo, visto que o número de grupos era superior a 2, restava selecionar quais os grupos que deveriam ser representados a branco e quais os que deveriam ser representados a preto. No entanto esta forma de abordar o problema, fazia com que o mesmo não fosse automático e não havia real forma de o automatizar, pelo que este método acabou por ser descartado e decidimos experimentar outros métodos que sigam a mesma ideia, mas de uma forma automática.

4.4.1.3 Máscara via KNN

A ideia de classificar um grupo de pixéis através da sua vizinhança parecia ser uma abordagem inteligente, numa tentativa de "tapar os buracos" na máscara que surgiam

quando havia sombras e mudanças bruscas de luminosidade na imagem. Por esta razão decidiu-se aplicar o algoritmo *k nearest neighbors*[9]. Este algoritmo é um método de aprendizagem supervisionada que faz uma previsão para um dado ponto baseando-se no rótulo aplicado aos pontos da sua vizinhança. Este algoritmo é simples e fácil de implementar não tendo real necessidade de aperfeiçoar diferentes parâmetros, no entanto é um algoritmo lento quando utilizado em problemas com grande quantidade de exemplos, algo que se acabou por comprovar na nossa experiência. É um exemplo de um algoritmo em que o tempo de treino é inferior ao tempo de efetuar a previsão.

Uma vez que se trata de um método de aprendizagem supervisionada começámos por adquirir e etiquetar os dados de forma a treinar o classificador. Assim, procedeu-se ao etiquetamento de 90 imagens de exemplos de armadilhas de insetos, marcando a área da imagem que pertencia à placa propriamente dita, e a área que sobrava que seria etiquetada como plano de fundo. Para agilizar o processo de etiquetar as imagens utilizou-se as ferramentas disponibilizadas pelo *APEER*[8], de forma a obter as máscaras correspondentes à placa.

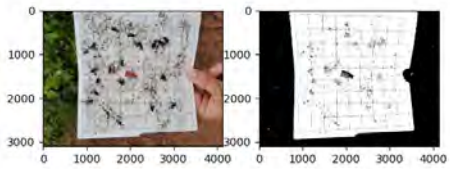
Tendo as imagens originais e as máscaras correspondentes ao nosso conjunto de dados, dividiu-se este conjunto em metade de forma aleatória por forma a diminuir a carga de dados a passar ao nosso classificador. Uma vez que cada imagem tinha dimensões de 4128x3096, recolheram-se em cada imagem 1000 pontos de forma aleatória, tendo sido efetuado o mesmo para cada máscara correspondente (com o mesmo fator de aleatoriedade, de forma que cada ponto da imagem corresponda ao mesmo ponto na máscara). Dos 45000 pontos retirados, 80% foram utilizados para treino e os 20% restantes para teste. De seguida, utilizou-se *GridSearch* de forma a descobrir qual os melhores parâmetros a utilizar na altura de treino do modelo, nomeadamente o número de "vizinhos" a utilizar no classificador. Treinou-se então um classificador *k nearest neighbors* com 3 como número de vizinhos, tendo sido testado posteriormente o classificador.

Os resultados revelaram ser razoáveis, no entanto não se revelaram melhores que o método previamente testado com a agravante que a fase de obter a máscara demorava aproximadamente 4 minutos, para cada imagem. Na figura 4.10, apresentamos alguns resultados deste método de obtenção de máscara binária.

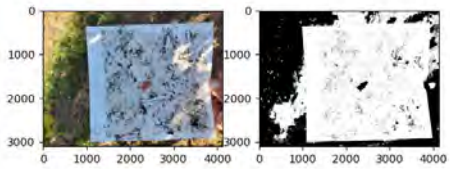
4.4.1.4 Máscara via *Random Forest*

Visto que já tínhamos imagens anotadas, e forma de arranjar pontos para treino dum modelo, testámos um outro classificador que já tínhamos utilizado numa abordagem diferente.

Previamente, tinha-se testado para cada imagem, recolher quatro áreas quadradas de pontos pertencentes ao centro da imagem, servindo como *ground truth* e anotada como pertencente à placa, e quatro áreas correspondentes aos cantos da imagem sendo anotadas como não pertencentes à placa, utilizando-se posteriormente o classificador de floresta aleatória[19]. Uma vez mais os resultados não foram satisfatórios, devido possivelmente



(a) Exemplo de uma boa máscara binária

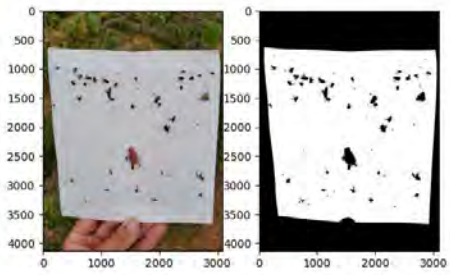


(b) Exemplo de uma má máscara binária

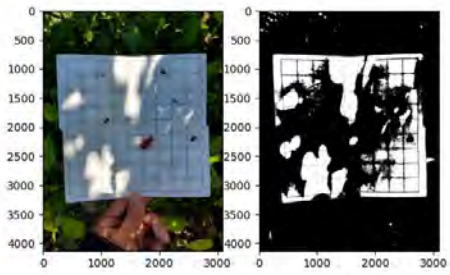
Figura 4.10: Resultados do processo de obtenção de máscara binária via algoritmo *K-nearest neighbors*

à existência de imagens em que a placa que serve de armadilha se encontrar encostada a um dos cantos da imagem.

Testou-se então o método utilizado previamente com o classificador KNN, mas desta vez utilizando o classificador de floresta aleatória no momento de classificação. Os resultados, que podem ser vistos na figura 4.11, acabaram por não ser melhores que o método utilizado com o KNN pelo que se decidiu não utilizar este método e passar à próxima fase de trabalho.



(a) Exemplo de uma boa máscara binária



(b) Exemplo de uma má máscara binária

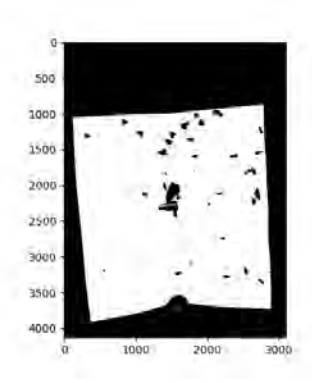
Figura 4.11: Resultados do processo de obtenção de máscara binária via algoritmo de floresta aleatória

4.4.1.5 Abordagem final

A abordagem atualmente utilizada começa com a leitura da imagem RGB original. Posteriormente aplica-se um algoritmo de balanceamento de brancos com o algoritmo de mancha branca com um percentil de 85. Este algoritmo procura a mancha mais próxima de branco para usar como referência. A imagem resultante é passada para o modelo "ycbcr "de modo a posteriormente se filtrar os valores do canal “cb” superiores a 120. Este foi o

valor escolhido após se ter feito a análise dos histogramas associados a este canal. Quanto à imagem resultante no modelo RGB, os valores são passados para *float* e normalizados, de forma que depois se filtre os valores do canal verde superiores a 0.7 e o mesmo para os valores do canal azul. Por fim, a máscara final é obtida através do "e" lógico entre a máscara filtrada no modelo RGB e a máscara filtrada no modelo "ycbcr". A figura 4.12 mostra os resultados do processo que é atualmente utilizado para obter a máscara binária de uma imagem. Em 4.12(a) temos a imagem original de exemplo, e em 4.12(b) temos aquele que será o resultado após o processo previamente descrito.

(a) Imagem original



(b) Máscara binária resultante

Figura 4.12: Processo de obtenção de máscara binária

4.4.2 Fase de recorte

Após obtermos a máscara binária da imagem original, necessitávamos agora de fazer a análise da máscara obtida de modo a filtrar a região de pixéis a preto em volta da maior região de pixéis a branco, a qual representava a região ocupada pela nossa placa que serve de armadilha para insetos.

Para esta tarefa foram testados três métodos diferentes.

4.4.2.1 Recorte via Linhas de Hough

Esta abordagem teve início com a utilização do algoritmo de procura de contornos da biblioteca *OpenCV*, aplicado à máscara binária gerada na fase anterior. Utilizou-se o método de extração de contornos exteriores (*RETR_EXTERNAL*) e posteriormente filtraram-se os contornos retornados pelo algoritmo de modo a obter o contorno de maior área que em teoria será o contorno correspondente ao contorno exterior da placa que serve de armadilha. A este contorno aplicou-se o algoritmo de envoltura convexa da biblioteca *OpenCV* de modo a obter o polígono que mais se adequa ao contorno encontrado, de modo a aproximar o contorno à forma da placa. Por fim aplicou-se o algoritmo de deteção de arestas de modo a obter as arestas do polígono anteriormente gerado.

Tendo as arestas do polígono, que no caso ótimo representariam as arestas da nossa placa original, utilizámos o algoritmo de linhas de Hough[13] da biblioteca *scikit-learn*[41], com vista a obter as coordenadas dos pontos dos quatro cantos da folha. Este algoritmo devolve os valores dos declives e os valores da ordenada na origem das equações das quatro retas com maior relevo no espaço Hough. Posteriormente separaram-se as retas em verticais e horizontais de modo a intersetar as retas verticais com as horizontais para que no fim se obtenham os pontos de interseção, os quais serão os cantos da folha. A figura 4.13 apresenta o resultado de correr o algoritmo de procura de arestas à esquerda, ao centro é apresentado os pontos de maior relevo no espaço de Hough e por fim à direita temos as retas que foram obtidas com o algoritmo e desenhadas na imagem original. Para este exemplo não há falhas a apresentar, sendo que o resultado acaba por ser perfeito. No entanto, como se encontra representado na figura 4.14, existem ocorrências em que apesar de termos uma máscara binária decente, que gera uma geometria boa para a placa, o algoritmo de linhas de Hough consegue apenas descobrir 3 das 4 retas, impossibilitando assim o recorte da placa.

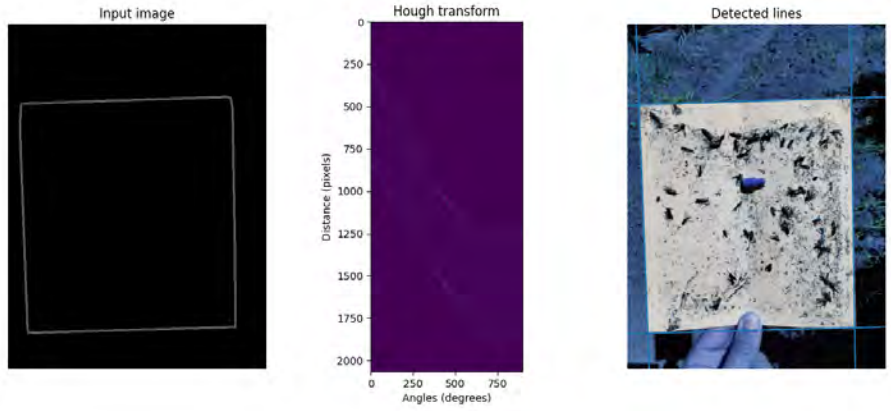


Figura 4.13: Exemplo de uma identificação correta de linhas

Tendo as coordenadas dos cantos da folha, foi necessário ordenar os pontos começando pelo canto superior esquerdo. Já com os pontos ordenados, tornou-se possível "endireitar" a placa, uma vez que esta poderia estar inclinada pela maneira como a mesma foi fotografada e assim, obtivemos finalmente apenas a zona pertencente à placa. Para tal utilizámos a função *getPerspectiveTransform()* da biblioteca *OpenCV* de modo a obtermos uma matriz de transformação que correlaciona a posição dos pontos na imagem original, com a posição onde se quer os pontos na imagem resultante. Com essa matriz restava chamar a função *warpPerspective()* da mesma biblioteca para efetuar a transformação propriamente dita, a qual resultou no recorte pretendido.

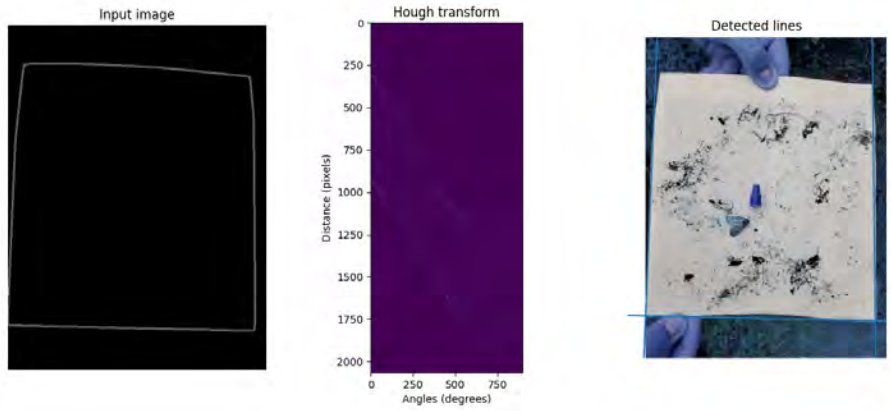


Figura 4.14: Exemplo de uma identificação incorreta de linhas

4.4.2.2 Recorte via Análise de sinal

Uma vez que a abordagem anterior não teve resultados infalíveis nas imagens que estavam a ser testadas, testou-se uma outra abordagem em relação ao modo como calculávamos as coordenadas dos pontos pertencentes aos cantos da placa. A abordagem anterior pecava no aspeto em que nem sempre as retas de maior relevo correspondiam às quatro arestas da placa, sendo que as retas encontradas variavam tendo em conta o número de pontos que gerávamos no algoritmo de linhas de Hough. Assim, e uma vez que os pixéis da nossa placa são brancos e os pixéis do plano fundo seriam pretos, tínhamos uma fronteira bem definida daquilo que eram pixéis pertencentes à placa e pixéis que estariam fora da placa. Ao descobrir esta zona de fronteira, descobriríamos os quatro cantos da placa. Decidiu-se então analisar esta mudança de pixéis na máscara binária previamente gerada, utilizando-se como auxiliar a função sigmoide.

Tal como no método anteriormente descrito, após se ter gerado a máscara binária da imagem original, encontrou-se o maior contorno presente na mesma e aproximou-se a forma deste à forma de um polígono regular. Aplicou-se o algoritmo de deteção de arestas a este polígono e procedeu-se então ao cálculo dos cantos da placa. Com o auxílio de uma estrutura de dados tabular, tivemos acesso aos valores dos pixéis de toda a máscara binária. É importante mencionar que dividimos a máscara binária em quatro partes, representando os quadrantes da máscara associado ao canto superior esquerdo, canto superior direito, canto inferior direito e canto inferior esquerdo. Percorremos as linhas e as colunas da nossa estrutura de dados tabular e obtivemos os valores dos pixéis a branco em cada uma das linhas e colunas que representavam a máscara binária. Colocando os valores do número de pixéis brancos em função da posição na máscara binária, percebemos que podíamos aproximar a função à função de uma sigmoide e posteriormente tínhamos a capacidade de calcular os pontos em que a função crescia e/ou decrescia exponencialmente e estabilizava, dependendo do quadrante em que nos encontrávamos, uma vez que eram estes os pontos de fronteira, devido à alteração de zona de pixéis pretos para zona

de pixéis brancos. Utilizando a equação:

$$P = x0 - \frac{1}{k} \times log(19)$$

onde P representa a abcissa ou ordenada que se pretende obter, conseguimos obter os pontos em que o crescimento e/ou decrescimento de pixéis brancos ocorria. O valor de x0 e de k é obtido utilizando a função *curve_fit* da biblioteca *SciPy*[55], função que aproxima os nossos dados a uma função à escolha, que no nosso caso se trata da função sigmoide.

Ao termos os parâmetros da sigmoide para cada um dos quadrantes, tínhamos então os pontos associados aos cantos da placa na imagem original, os quais facilmente se ordenavam uma vez que sabíamos a que coordenadas cada canto correspondia. Restava só aplicar as mesmas técnicas aplicadas no método anterior, isto é, a mudança de perspetiva e o recorte da imagem. Também para este método obtivemos resultados variáveis, sendo que apresentamos na figura 4.15, um exemplo em que através do método descrito conseguimos obter as coordenadas dos 4 cantos da folha (representados por um ponto a vermelho) 4.15(a), e um exemplo em que o mesmo método aplicado acabou por gerar pontos incorretos, muito possivelmente devido à alteração de luminosidade na imagem 4.15(b).

(a) Exemplo de uma identificação correta de pontos

(b) Exemplo de uma identificação incorreta de pontos

Figura 4.15: Resultados da obtenção de pontos através do método de análise de sigmoide

O diagrama presente na figura 4.16 pretende ilustrar os processos que ocorrem de modo a fazer o recorte da imagem original, consoante a abordagem utilizada.

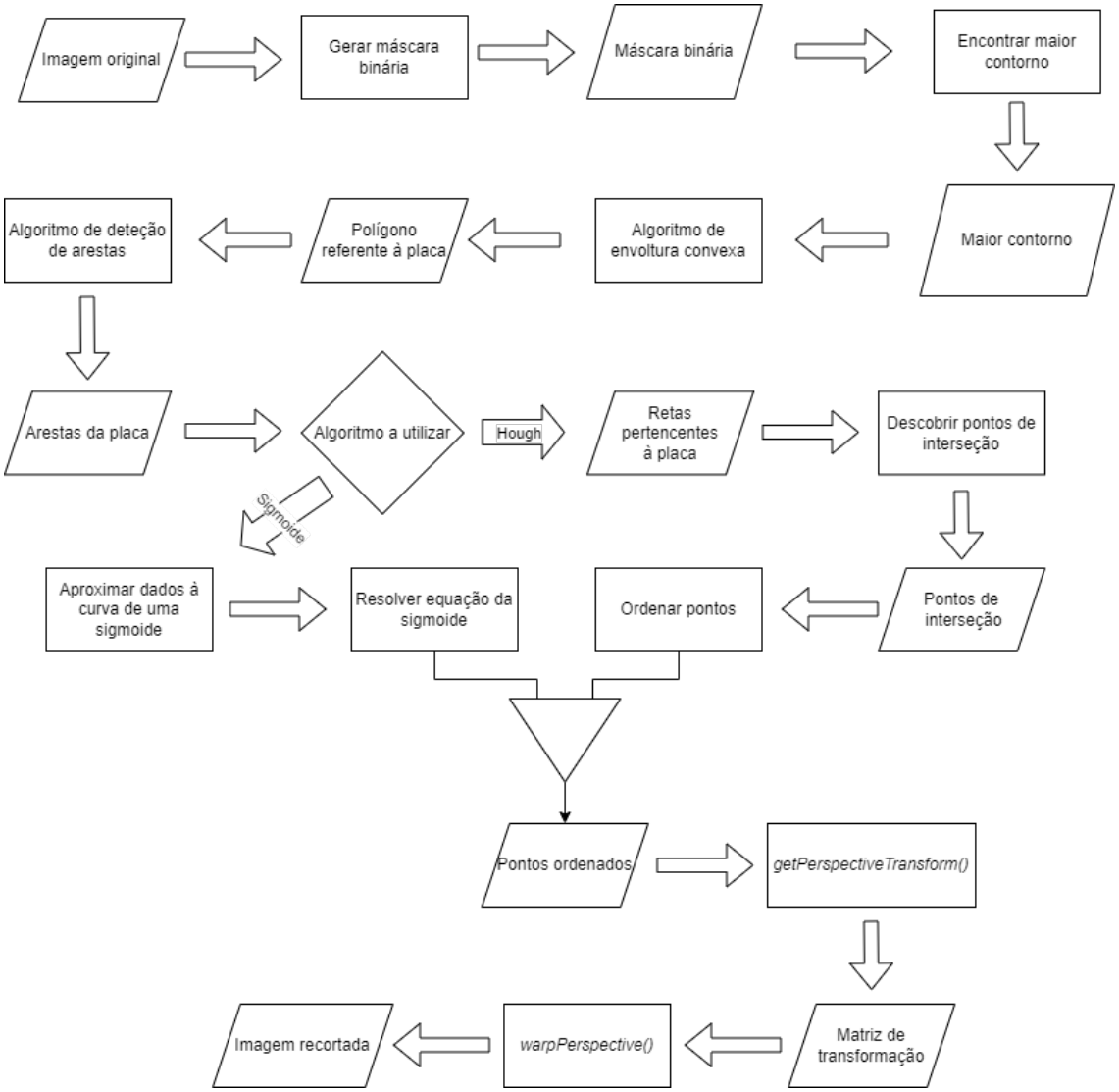


Figura 4.16: Diagrama que ilustra ambas as abordagens anteriores

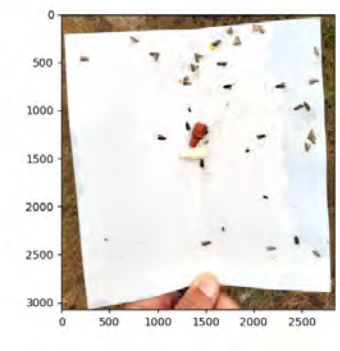
4.4.2.3 Abordagem final

Ainda insatisfeitos com a solução testada, acabámos por optar por uma solução mais simples, uma vez que as opções anteriores estariam bastante dependentes da qualidade da máscara binária, a qual, como explicado anteriormente não teria a melhor qualidade devido à presença de sombras e/ou exposição direta ao sol. Por esta razão optámos por uma abordagem que faria um recorte como inicialmente se pretendia, mas com uma maior margem caso erros aconteçam resultantes da fase de obtenção da máscara binária.

Esta solução começa por colocar os dados numa estrutura de dados tabular e com o auxílio de uma janela deslizante, disponibilizada pela biblioteca *Pandas*[36], inicializámos a suavização dos dados de modo a reduzir o ruído presente na máscara binária. Com os dados suavizados resta descobrir a primeira ocorrência de um pixel branco nestes dados suavizados. O valor obtido será correspondente ao mínimo global, que neste caso

corresponde ao canto superior esquerdo. Para descobrir o máximo, procuramos por pixéis pretos para lá da segunda metade da máscara binária. Se estes existirem, o máximo será correspondente à primeira ocorrência de um pixel preto nos dados. Se estes não existirem, o máximo corresponde à largura ou altura da imagem dependendo se estamos a verificar as colunas ou as linhas da estrutura de dados. Os máximos correspondem ao canto inferior direito da máscara binária. Tendo estes valores, ajustamos os mesmos, dando alguma margem, de forma a não recortar partes que possam pertencer à placa. Finalmente procedemos ao recorte da imagem. A figura 4.17 pretende pôr lado a lado a imagem original e a imagem que se obteve após o algoritmo de recorte implementado de modo a demonstrar a transformação efetuada. Como podemos observar, a imagem presente em 4.17(a) não está ajustada ao tamanho da placa pelo que podemos remover uma boa parte do plano de fundo, sem comprometer a placa presente na imagem original. Através do algoritmo obtivemos a imagem presente em 4.17(b) onde conseguimos ver que a imagem está mais ajustada ao espaço ocupado pela placa tendo-se perdido boa parte de vegetação em plano de fundo, inútil para o objetivo em mãos.

(a) Imagem original



(b) Imagem resultante do processo de recorte

Figura 4.17: Processo de recorte da imagem original

4.5 Construção do conjunto de dados

Tendo reduzido o plano de fundo da imagem, de modo a facilitar o processamento da mesma, podemos agora iniciar o processo de segmentação de insetos.

Apesar das melhorias nesta fase, os resultados não foram satisfatórios devido à presença de lixo na placa o que dificulta a tarefa de separar o que é inseto daquilo que não é que por sua vez dificulta a operação de obter os contornos pertencentes aos insetos. Devido a este percalço houve a necessidade de mudar a maneira de obtenção dos nossos dados para treino e validação dos nossos modelos, e até mesmo a maneira de obtenção de

regiões para os nossos modelos procederem à tarefa de classificação como será ilustrado mais tarde.

Uma vez que a tarefa de obtenção de imagens de insetos de forma automática se revelou inconsistente, criou-se um programa, que apesar de obrigar o utilizador a efetuar a recolha de forma manual, agilizou a tarefa de obter imagens de insetos a partir de uma imagem maior. Este programa permite que a partir de cliques na imagem original, se gerem dados que serão interpretados por outro programa de modo a extrair uma região de 224x224 (tamanho aceite pelos modelos que viriam a ser treinados), região a qual terá no centro o inseto da espécie que se pretendia estudar. Os dados gerados incluem o nome da espécie a ser catalogada, as coordenadas do canto superior esquerdo da caixa que envolve o inseto, a largura e altura desta mesma caixa (que no nosso caso será sempre 224x224), a largura e altura da imagem após o processo de remoção de fundo, o nome do ficheiro associado à imagem original e o nome do ficheiro associado à imagem gerada após a marcação dos insetos.

Aprofundando a descrição anterior, o primeiro programa, apresenta ao utilizador a imagem recortada e este ao clicar numa região da imagem irá gerar dados associados a um quadrado de dimensões 224x224 pixéis, com centro nas coordenadas do clique. Estes dados são guardados localmente, para que um segundo programa os interprete e faça um recorte na região gerada pelo programa anterior. Para além dos dados gerados, este programa desenha também retângulos nas zonas que foram marcadas de modo a posteriormente se guardar uma imagem contendo todas as marcações que foram efetuadas.

O segundo programa irá buscar os dados guardados localmente, interpretando-os de forma a obter as coordenadas do canto superior esquerdo e do canto inferior direito, para posteriormente se recortar a região referente às coordenadas obtidas na imagem original. O recorte resulta de um *slicing* do *array*, uma vez que a imagem original é representada como um *array*.

Este segundo programa poderá gerar também um número de regiões aleatórias, que não intersetam as regiões que foram marcadas como sendo do inseto em estudo, de forma a obter dados negativos não enviesados. Para tal é necessário percorrer todas as regiões que foram marcadas como sendo de um inseto pelo programa anterior e verificar que a área de interseção destas regiões com a zona gerada aleatoriamente é 0 (na prática estas regiões são quadrados de dimensões 224x224). Se tal for verdade para todas as regiões, podemos guardar a zona gerada como um exemplo negativo para o nosso conjunto de dados.

Tendo as imagens dos insetos nas dimensões pretendidas, procedeu-se à remoção manual dos insetos que se encontravam na mesma posição da placa em semanas consecutivas, de modo a remover imagens duplicadas no nosso conjunto de dados.

4.6 Conjunto de dados

Das 448 imagens originais disponibilizadas, foram utilizadas 92 imagens relativas à espécie Sésia, 33 imagens relativas à espécie Bichado e 146 imagens relativa à espécie Funebrana. As imagens restantes acabaram por não ser utilizadas, uma vez que, ou a espécie observada não tinha indivíduos suficientes para construir um conjunto de dados, ou a placa fotografada se encontrava em mau estado sendo impossível fazer a identificação dos sujeitos da espécie, ou então pertencem à espécie Molesta que acabou por ser desprezada.

Para a tarefa de reconhecimento dos indivíduos pertencentes a cada espécie, recorreu-se ao ficheiro disponibilizado que apresentava a contagem de sujeitos na imagem em questão e aos ficheiros de protocolo de cada inseto de forma a identificar corretamente os insetos presentes na placa na altura em que a fotografia foi tirada. O processo de contagem de insetos pertencentes a cada tipo de espécie na imagem foi efetuado por um especialista, no entanto o processo de identificação de espécimens na imagem foi efetuado por nós investigadores, o que levou a desprezar a tentativa de identificação de indivíduos da espécie Molesta.

Após a fase de construção do conjunto de dados, o nosso conjunto de dados continha agora 1084 imagens de insetos da espécie Sésia, 1657 imagens de insetos da espécie Funebrana e 100 imagens da espécie Bichado, uma espécie que comparativamente às outras duas tinha um número muito inferior de exemplos positivos, tendo sido escolhido para fazer uma comparação de performance com as outras classes que tinham bem mais exemplos. Uma vez que se optou por ter um modelo inicial que tenta filtrar o lixo presente na imagem, sendo um modelo de classificação de insetos, necessitámos também de ter um conjunto de dados associado a insetos, tendo este conjunto 4005 exemplos de insetos. Neste conjunto de insetos estão também presentes as imagens dos insetos pertencentes à classe Sésia, Funebrana e Bichado. As imagens de insetos não pertencentes às espécies previamente enunciadas foram obtidas nas mesmas imagens que serviram de base para obter exemplos das espécies.

De modo a manter equilíbrio entre classes no conjunto de dados referente ao modelo de classificação de insetos, geraram-se imagens aleatórias até perfazer um número de exemplos equivalente ao número de insetos extraídos. Para tal foi necessário recorrer ao programa de interpretação de caixas envolventes, de forma a obter as caixas que englobavam insetos na imagem, de forma a posteriormente gerar caixas envolventes que não intersetassem estas últimas, permitindo assim gerar exemplos negativos para equilíbrio das classes. Já para o caso das classes específicas, os exemplos negativos são representados pelos insetos que foram extraídos do conjunto de imagens referentes à espécie em questão, tendo-se posteriormente removido manualmente o excesso de exemplos negativos de modo a manter equilíbrio entre classes. Esta remoção teve em conta a similaridade entre exemplos.

Posteriormente estes conjuntos de dados foram utilizados no treino de diferentes tipos de modelos de classificação, sendo a separação ilustrada na tabela 4.1. Temos um modelo

de classificação de insetos com 4005 insetos e 4005 casos de zona de placa sem insetos, um modelo de classificação da espécie Sésia com 1084 insetos da espécie Sésia e 1084 insetos que se encontravam em imagens comuns aos insetos dessa espécie, um modelo de classificação da espécie Funebrana com 1657 insetos dessa espécie e 1415 insetos nas condições previamente descritas e finalmente um modelo de classificação da espécie Bichado com 100 insetos da espécie Bichado e 100 insetos não pertencentes a essa espécie mas presentes nas placas destinadas ao controlo do Bichado. Este último conjunto de dados tem a particularidade de ser claramente inferior em comparação aos dois conjuntos de dados associados às classes com maior representatividade.

Tabela 4.1: Número de indivíduos em cada uma das classes para cada um dos modelos.

Modelo/Classe	Exemplos positivos	Exemplos negativos	Total
Insetos	4005	4005	8010
Sésia	1084	1084	2168
Funebrana	1657	1415	3072
Bichado	100	100	200

4.7 Treino dos modelos

Quanto à implementação dos modelos, recorreu-se uma vez mais à biblioteca *Keras*[7]. Esta biblioteca disponibiliza vários modelos de aprendizagem profunda juntamente com os pesos resultantes do treino num conjunto de dados pré-definido, possibilitando a utilização de modelos robustos treinados com conjuntos de dados de grandes proporções, o que acaba por mitigar o problema de insuficiência de dados para modelos com um grande número de parâmetros a treinar. Assim esta biblioteca permitiu-nos recorrer a uma técnica denominada *transfer learning*, a qual se baseia em utilizar um modelo pré-treinado para a fase de extração de atributos permitindo "construir" e treinar camadas por cima deste modelo pré treinado para a fase de classificação, de forma a adaptar-se ao nosso problema. No nosso caso, utilizámos uma seleção de modelos da biblioteca *Keras*, tendo treinado estes modelos com o conjunto de dados do *ImageNet* e posteriormente adicionámos para além de uma camada de *dropout* com fator 0.2, uma camada densa no fim do modelo com 2 neurónios de *output*, de forma a adequar-se ao número de classes do nosso problema. Visto que abordámos este problema como sendo um problema de cariz binário, a nossa última camada é uma camada de ativação *softmax* com 2 neurónios de *output*.

Do aspeto geral dos modelos falta mencionar a camada de *input* que aceitará imagens de dimensões 224x224x3, sendo 224 o valor de altura e largura em pixéis da imagem e 3 o número de canais, que neste caso se refere ao RGB. Esta é a razão pela qual, as nossas imagens no momento de obtenção do conjunto de dados foram formadas como sendo áreas de 224 por 224.

Recorreu-se à técnica de *transfer learning* uma vez que ao procedermos à construção do nosso modelo de raiz, os resultados estavam aquém do que esperávamos, podendo estar relacionado com a necessidade de um grande número de dados, que neste caso são imagens, para permitir ao modelo treinar com eficiência. Esta técnica serve para mitigar esse problema, acabando por permitir a utilização conjunta com outra técnica conhecida no treino de modelos de aprendizagem profunda, chamada *fine-tuning*. Os modelos importados da biblioteca *Keras* estariam prontos a efetuar uma tarefa de classificação, no entanto como estariam treinados para fazer uma classificação entre 1000 classes, é necessário desprezar as últimas camadas destes modelos. Por essa razão, as últimas camadas acabam por ser substituídas e treinadas por nós, sendo que as camadas pertencentes ao modelo treinado no conjunto de dados *ImageNet* ficam "congeladas" no que diz respeito à aprendizagem. Poderíamos ficar por aqui, no entanto, existe a possibilidade de melhorar o nosso modelo se efetuarmos *fine-tuning*. Esta técnica consiste em "descongelar" as últimas camadas do modelo que importámos de maneira a que os pesos sejam atualizados e possivelmente exista uma melhoria na classificação. Na nossa experiência permitimos o treino de blocos de camadas convolucionais, sendo que variámos o número de blocos a treinar entre iterações.

Falta mencionar dois fatores com impacto no treino dos modelos. O primeiro trata-se da função de perda utilizada, que por se tratar de um problema binário, utilizámos a função de perda entropia cruzada binária. O outro fator é o optimizador utilizado, tendo sido testados em diferentes iterações os optimizadores *Adam*[26], *Adagrad*[12], *SGD*[46] e *RMSprop*[53]. O objetivo destes optimizadores será atualizar os atributos da rede neuronal, nomeadamente os pesos dos neurónios e a taxa de aprendizagem do modelo de modo a minimizar a função de perda.

Tendo falado em todos os fatores associados ao treino do nosso modelo falta agora mencionar a escolha das arquiteturas a testar para cada um dos nossos modelos. Uma vez que para a tarefa em questão tem maior relevância a precisão e exatidão na contagem dos insetos comparativamente ao tempo de processamento, optou-se por um modelo de duas fases, em que a primeira fase servirá para propor regiões onde existe a possibilidade de existirem insetos e a segunda fase será de classificação destas regiões. Desprezou-se então a utilização de um modelo *YOLO*[42] e testaram-se modelos proeminentes na tarefa de classificação de imagens. Por essa razão treinaram-se os modelos *VGG-16*[48], *VGG-19*, *ResNet50*[18], *ResNet101*, *MobileNet*[22] e *Inception*[51].

4.8 Aplicação desenvolvida

A nossa aplicação tem início quando é passada uma imagem como argumento. Esta imagem é recortada através da metodologia implementada de modo a remover o máximo de área pertencente ao plano de fundo. A imagem resultante, que terá menores dimensões será então avaliada quanto à possível presença de regiões propícias a conter insetos dentro da mesma. Estas regiões irão ter dimensões de 224x224, e a obtenção destas regiões passa

por enquadrar as regiões numa limitação de um número total de pixéis brancos ou pretos de modo a não enviar para o classificador regiões totalmente a branco, isto é, que de certeza que não existirá um inseto nelas ou regiões totalmente a preto, isto é, regiões que não farão parte da placa. A figura 4.18 demonstra o resultado deste processo, onde estão presentes a azul as regiões que serão posteriormente classificadas pelo nosso primeiro classificador.

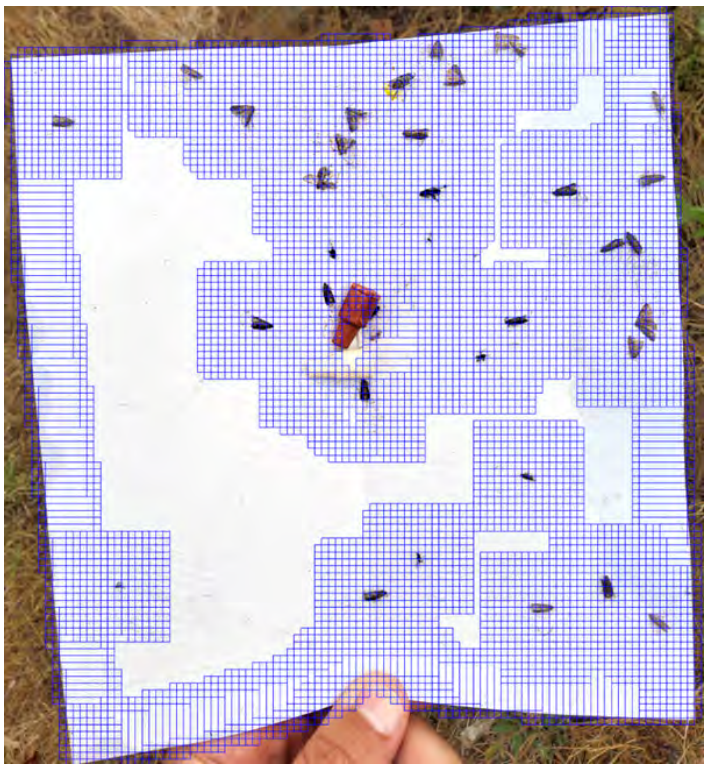


Figura 4.18: Regiões a azul que serão enviadas para o classificador de insetos

Ainda que as áreas na imagem pareçam menores que 224x224, isto deve-se à interseção de regiões de 224x224 devido ao salto de 28 entre regiões.

Estas regiões serão então enviadas para o nosso classificador de insetos de forma que sejam classificadas e para que posteriormente, fiquemos apenas com regiões que têm de facto insetos. Após a classificação, temos apenas interesse nas regiões que foram classificadas como tendo um inseto dentro das mesmas. É sobre estas regiões que se vai aplicar o algoritmo de supressão de não-máximos[21]. Este algoritmo consiste em remover regiões que se intersetem com outras e que apresentem um valor de confiança inferior ao das regiões que interseta. Este algoritmo permite assim evitar contagens duplas do mesmo inseto e ao mesmo tempo permite que insetos que se encontrem relativamente próximos possam ser ambos contabilizados. Será importante encontrar o equilíbrio entre reduzir o número de contagens duplas, o que poderá levar a que dois insetos próximos um do outro sejam contabilizados como sendo apenas um, ou reduzir o número de insetos que se perdem devido a proximidade, o que poderá aumentar o número de contagens duplas em algumas ocasiões.

Após a aplicação do algoritmo temos então as regiões da placa que contêm de facto insetos. São estas regiões que pretendemos classificar de modo a saber quantos insetos da espécie a monitorizar existem na imagem original. Assim, as regiões irão ser enviadas para o modelo mais específico, isto é, modelo de classificação de Sésia, Funebrana ou Bichado, de forma a separarmos insetos aleatórios daqueles que realmente interessam. Nesta fase já não é necessário a aplicação do algoritmo de supressão de não-máximos, sendo que no final temos o número de regiões que foram classificadas como sendo o inseto a monitorizar. É este valor que será apresentado ao utilizador no final do programa. A imagem apresentada na figura 4.19 ilustra a janela que será apresentada ao utilizador antes da contagem final.

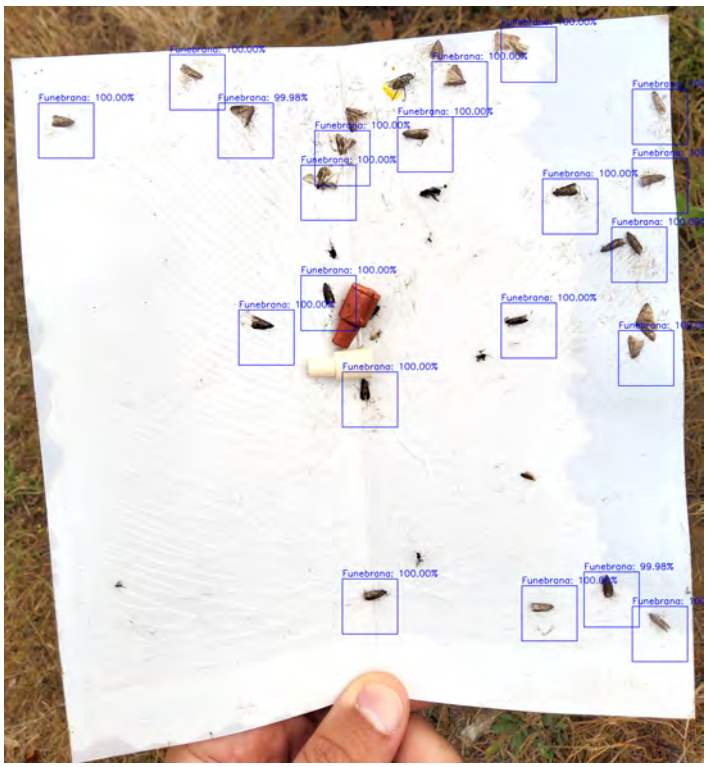


Figura 4.19: Resultado dos modelos aplicados à imagem de exemplo

É importante mencionar que para além desta última janela, o programa apresenta também ao utilizador uma janela com os resultados após o algoritmo de supressão de não-máximos, de modo a permitir que o utilizador faça retificações às classificações efetuadas pelos modelos, atualizando sempre a contagem de insetos, resultando assim numa contagem "semi-automática". Se o utilizador não desejar efetuar nenhum tipo de retificação basta fechar a janela que o programa prossegue de forma automática. Na figura 4.20 temos o exemplo de um resultado de interação por parte de um utilizador, onde as caixas a verde representam classificações por parte do modelo que o utilizador considerou estarem corretas, a vermelho classificações que foram consideradas corretas pelo modelo mas que o utilizador considerou errado, ou seja, falsos positivos, e as caixas a azul são caixas que acabaram por ser desprezadas pelo modelo, mas que o utilizador considera ser

pertencentes à classe positiva, ou seja falsos negativos.

Figura 4.20: Exemplo (meramente ilustrativo) de interação com um utilizador. A vermelho temos os falsos positivos, a azul os falsos negativos e a verde os verdadeiros positivos. Temos ainda uma dupla contagem no canto superior direito.

O diagrama que representa o fluxo de dados é apresentado na figura 4.21.

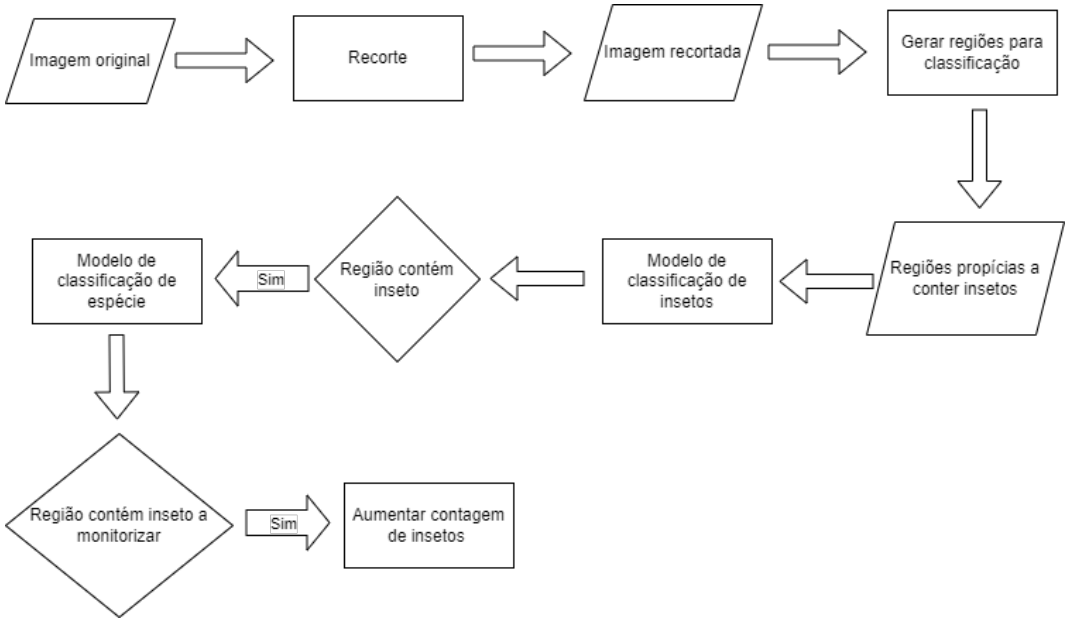


Figura 4.21: Diagrama associado ao programa desenvolvido

5

Interpretação dos resultados

Este capítulo pretende expor os resultados obtidos durante o treino dos modelos e posteriormente a performance dos modelos no conjunto de teste e num ambiente mais próximo da realidade. Os resultados aqui demonstrados irão justificar algumas das escolhas efetuadas e apresentadas em capítulos anteriores.

5.1 Resultados do treino do modelo de classificação de insetos

Como mencionado, a escolha dos modelos apresentados previamente no capítulo 4.7, foi efetuada consoante a metodologia apresentada no capítulo 3.2. De seguida, neste capítulo serão apresentados os resultados obtidos para o treino dos diferentes tipos de modelo ao se aplicar a metodologia previamente descrita. Estes resultados servirão assim de justificação perante as diferentes possibilidades no que toca à escolha da arquitetura e parâmetros a utilizar aquando do treino de um modelo de classificação binária de imagens.

Inicialmente ponderou-se quais as arquiteturas a testar para a tarefa proposta. Tendo em contas os modelos apresentados no capítulo de estado de arte, o desempenho de cada um desses modelos na tarefa de classificação de imagens e a disponibilidade dos mesmos na biblioteca *Keras*, acabámos por selecionar para treino as seguintes arquiteturas: *MobileNet*, *VGG-16*, *VGG-19*, *ResNet50*, *ResNet101* e por fim *Inception*. Cada uma destas arquiteturas tem suporte na biblioteca *Keras* facilitando assim a implementação das mesmas no nosso projeto.

Tendo selecionado as arquiteturas a testar, fixámos os valores dos outros parâmetros de modo a manter o treino consistente e permitir tirar conclusões em relação à melhor arquitetura a utilizar na tarefa em questão. Seguindo a metodologia previamente apresentada, utilizámos um tamanho para os *batches* de 32, utilizámos o optimizador *Adam* com uma taxa de aprendizagem de 0.00001 e não recorremos a *fine tuning* nesta fase do treino, deixando as camadas pertences à extração de atributos "congeladas"e permitindo apenas o treino das camadas densas pertencentes às camadas de classificação do modelo. É importante mencionar este pormenor uma vez que se recorreu a *transfer learning* na

construção dos modelos, sendo que todos os modelos foram treinados com o conjunto de dados do *ImageNet* pelo que as camadas pertencentes à fase de extração de atributos irão ter os pesos associados ao treino com o *ImageNet*, pelo que se permitirmos que alguns destes blocos sejam treináveis, isto é, que atualizem os seus pesos no momento do treino, poderemos obter resultados mais favoráveis, algo que será demonstrado posteriormente.

A tabela 5.1 apresenta então os resultados obtidos no treino do modelo utilizando cada uma das arquiteturas selecionadas.

Tabela 5.1: Resultados do treino do modelo para classificação de insetos consoante a arquitetura utilizada.

Arquitetura/Métrica	***Validation accuracy***	***Test accuracy***	***F1-Score***
MobileNet	0.9766	0.98444	0.9845
VGG16	0.9878	0.9875	0.9877
VGG19	0.9877	0.9881	0.9884
ResNet50	0.9914	0.9888	0.9890
ResNet101	**0.9920**	**0.9919**	**0.9921**
Inception	0.9543	0.9576	0.9590

Os valores de *validation accuracy* apresentados na tabela são referentes à média resultantes dos treinos com diferentes conjuntos de validação, sendo o modelo treinado cinco vezes. Os valores de *accuracy* de teste e *F1-score* surgem da previsão do modelo que apresentou melhor valor de exatidão de classificação nas iterações anteriores.

Como se pode ver pelos resultados presentes na tabela, a arquitetura que apresentou melhores valores no conjunto de validação foi a arquitetura *ResNet101* com *accuracy* no conjunto de validação de 0.9920, ou seja, sensivelmente 99.20%. É importante referir também que as arquiteturas *VGG* obtiveram um bom valor de *accuracy*, no entanto esta arquitetura acaba por ser uma arquitetura mais pesada comparativamente a arquiteturas *ResNet*. Seria perfeitamente correto escolher a arquitetura *ResNet50*, que obteve um valor de 99.14% no conjunto de validação, o que representa uma diferença de performance de apenas 0.16%, sendo que esta arquitetura tem por base a mesma estrutura da arquitetura *ResNet101*, havendo diferença no número de camadas no quarto bloco convolucional o que a acaba por tornar numa arquitetura mais leve no aspeto de espaço ocupado e é também uma arquitetura que demora menos tempo a efetuar inferências. Ainda assim, uma vez que não foram impostos limites no aspeto espacial e de tempo, e sendo o objetivo uma contagem precisa de insetos, acabámos por selecionar e utilizar como arquitetura base a *ResNet101*. De notar também que foi esta arquitetura que acabou por ter melhores resultados no que toca às métricas de teste, com *accuracy* no conjunto de teste de 99.19% e *F1-Score* de 0.9921.

Agora que temos a arquitetura base selecionada, podemos alterar um dos parâmetros previamente fixos, de modo a testar qual o melhor de entre as possibilidades. Decidimos testar então qual seria o melhor optimizador para a tarefa de classificação de insetos. Os

restantes parâmetros continuaram inalterados para esta fase de testes.

Para a fase de teste de optimizadores, foram testados o optimizador *Adam*, o optimizador *Adagrad*, o optimizador *RMSprop* e por fim o optimizador *SGD*. Neste caso como se pode ver pela tabela 5.2, os resultados entre optimizadores pouco variam, havendo pouca ou quase nenhuma diferença entre os dois melhores optimizadores, neste caso o *Adam* e o *RMSprop*. Estes obtiveram um valor de *validation accuracy* de 99.16% e 99.13%, respetivamente. Os optimizadores restantes também não tiveram uma má performance sendo que o optimizador *Adagrad* obteve 98.39% e o optimizador *SGD* obteve 97.80% de *accuracy* no conjunto de validação. Em relação às métricas de teste, uma vez mais os valores do optimizador *Adam* e *RMSprop* apresentam pouca diferença, sendo que neste caso é o optimizador *RMSprop* que acaba por apresentar melhores valores, tanto na *accuracy* de teste como no valor de *F1-Score*, com um valor de 99.38% para a *accuracy* de teste e um valor de 0.9939 para o *F1-Score*. Apesar do optimizador *Adam* ter obtido valores de 99.31% para a *accuracy* e 0.9933 para o *F1-Score*, valores que são inferiores comparativamente com o *RMSprop*, uma vez que apresentou melhor valor na métrica associada ao conjunto de validação, concluiu-se que para esta tarefa, o melhor optimizador seria o *Adam*.

Tabela 5.2: Resultados do treino do modelo para classificação de insetos consoante o optimizador utilizado.

Optimizador/Métrica	***Validation accuracy***	***Test accuracy***	***F1-Score***
Adam	**0.9916**	0.9931	0.9933
Adagrad	0.9839	0.9856	0.9860
RMSprop	0.9913	**0.9938**	**0.9939**
SGD	0.9780	0.9782	0.9787

Estando selecionada a arquitetura e o optimizador a utilizar, queríamos agora testar o efeito de efetuar *fine tuning* no nosso modelo. Para tal, utilizando a arquitetura *ResNet101* e o optimizador *Adam*, efetuámos alterações na base do nosso modelo completo de modo a gradualmente permitir o treino de camadas que previamente não teriam a capacidade de ser treinadas. Para tal é necessário compreender que a fase de extração de atributos está dividida em blocos pelo que os testes a serem efetuados consistem em gradualmente permitir que o bloco que precede o último bloco treinável tenha a possibilidade de ser treinado também.

Como explicado anteriormente no início da construção do modelo, e nos testes anteriores, o modelo utilizado permitia o treino apenas das últimas camadas, camadas associadas à fase de classificação, enquanto as camadas que as precediam estavam "congeladas", não permitindo que os pesos das mesmas fossem atualizados. Agora e como está demonstrado na tabela 5.3, testámos o resultado de permitir a atualização destes pesos.

Nesta tabela, cada linha representa o valor de *accuracy* obtido no conjunto de validação, no conjunto de teste e o valor de F1-Score permitindo o treino daquele número de blocos, onde 0 significa que apenas os blocos da fase de classificação foram treinados e 5

Tabela 5.3: Resultados do treino do modelo para classificação de insetos consoante o número de blocos "descongelados"para treino.

Blocos/Métrica	***Validation accuracy***	***Test accuracy***	***F1-Score***
0 Blocos	0.9915	0.9919	0.9921
1 Bloco	0.9920	0.9919	0.9920
2 Blocos	**0.9925**	0.9913	0.9915
3 Blocos	0.9924	**0.9925**	**0.9927**
4 Blocos	0.9920	0.9913	0.9913
5 Blocos	0.9916	0.9906	0.9909

significa que todos os blocos de camadas convolucionais foram treinados. Olhando para a tabela percebemos que obtemos a melhor performance no conjunto de validação quando decidimos treinar dois blocos, obtendo 99.25%. No entanto, olhando para as métricas de teste, existem três abordagens diferentes que obtiveram melhores resultados que a abordagem de treino de dois blocos, que é o caso de não treinar blocos, que obteve 99.19% de *accuracy* de teste e 0.9921 de *F1-Score*, o caso de treinar apenas um bloco, que obteve o mesmo resultado de *accuracy* que o caso anterior mas um *F1-Score* de 0.9920 e o caso de treinar três blocos, que obteve um resultado de *accuracy* de teste de 99.25% e um *F1-Score* de 0.9927, obtendo então o melhor resultado geral para a métrica de *accuracy* de teste e de *F1-Score*.

Desprezando para já as métricas de teste, concluímos então que atualizando os pesos dos dois últimos blocos é a melhor abordagem, sendo que se perde capacidade de classificação quando mais blocos são treinados. De notar que no caso de treino de blocos, no caso de resultados semelhantes entre abordagens, não há uma forma de decidir qual a melhor abordagem, a não ser talvez pelo tempo gasto no treino dos modelos e a memória ocupada durante o treino dos mesmos, que será maior no caso de ser necessário treinar um maior número de blocos. É importante realçar também que para o teste de qual o número de blocos a treinar, reduziu-se o número de tamanho dos *batches* de 32 para 16.

Por fim, falta descobrir qual a melhor taxa de aprendizagem a utilizar em conjunto com o nosso optimizador para a tarefa proposta. Para este conjunto de testes testaram-se cinco valores diferentes.

Analisando os resultados ilustrados na tabela 5.4, podemos chegar à conclusão que o modelo com melhor performance é o modelo que utiliza uma taxa de aprendizagem de 0.00001. Esta abordagem conseguiu o melhor valor para o conjunto de validação com um valor de *accuracy* de 99.47%, tendo obtido também os melhores valores relativos ao conjunto de teste, com uma *accuracy* de teste de 99.44% e um valor de *F1-Score* de 0.9945.

Está assim descoberto, aquele que será o modelo que em princípio será o melhor na tarefa de separação de insetos de regiões da imagem sem interesse. Este modelo segue uma arquitetura *ResNet101* com treino de dois blocos pertencentes às camadas de extração de atributos e utiliza o optimizador *Adam* com taxa de aprendizagem de 0.00001.

Tabela 5.4: Resultados do treino do modelo para classificação de insetos consoante a taxa de aprendizagem utilizada.

Aprendizagem/Métrica	***Validation accuracy***	***Test accuracy***	***F1-Score***
0.001	0.9881	0.9888	0.9889
0.0001	0.9939	0.9906	0.9908
0.0005	0.9905	0.9888	0.9891
0.00001	**0.9947**	**0.9944**	**0.9945**
0.00005	0.9944	0.9919	0.9921

5.2 Resultados do treino dos modelos de classificação de espécies

Como referido anteriormente, para efetuar a contagem de insetos de uma espécie optámos por treinar modelos para cada uma das espécies pretendidas, tornando um problema que poderia ser de multiclasse, num modelo de classificação binária. Esta abordagem não é problemática até porque as imagens das armadilhas estão separadas por espécie, permitindo a que o utilizador utilize o modelo correto dependendo da espécie que se pretende contar. Nas secções seguintes serão apresentados os resultados do treino de cada um dos modelos criados para classificação de insetos da espécie Sésia, Funebrana e Bichado. A abordagem utilizada, por se tratar de um problema de classificação binária, acaba por ser idêntica à utilizada no treino do modelo de classificação de insetos, alterando apenas os dados utilizados.

5.2.1 Modelo de classificação de Sésia

Tal como no modelo de classificação de insetos, iniciou-se o processo de treino do modelo fixando alguns parâmetros e fazendo variar as arquiteturas do modelo, por entre as arquiteturas previamente selecionadas. Os resultados obtidos podem ser visualizados na tabela 5.5.

Tabela 5.5: Resultados do treino do modelo para classificação de Sésia consoante a arquitetura utilizada.

Arquitetura/Métrica	***Validation accuracy***	***Test accuracy***	***F1-Score***
MobileNet	0.9053	0.92	0.92
VGG16	0.9543	0.97	0.96
VGG19	0.9605	0.95	0.95
ResNet50	0.9684	**0.98**	**0.98**
ResNet101	**0.9735**	**0.98**	0.97
Inception	0.8591	0.85	0.85

Analisando os resultados ilustrados nas tabelas, podemos chegar à conclusão de que a arquitetura com melhor performance é a arquitetura *ResNet101* com um valor de *accuracy*

de 97.35% no conjunto de validação, sendo que uma vez mais a arquitetura *ResNet50* obteve o segundo melhor resultado nesta métrica, com uma diferença inferior a 1%. De seguida foram as arquiteturas *VGG* a obter os melhores resultados com uma diferença pouco acentuada, e por fim a *MobileNet* e a *Inception* obtiveram resultados com uma diferença considerável. Olhando agora paras a métricas de teste, uma vez mais as arquiteturas *ResNet* obtiveram melhores resultados sendo que para a *accuracy* de teste, tanto a *ResNet50* como a *ResNet101* conseguiram um resultado de 98% enquanto na métrica *F1-Score* a melhor foi a *ResNet50* com 0.98.

Decidimos então prosseguir com o treino da arquitetura *ResNet101*, passando agora para o treino do modelo fazendo variar os optimizadores. Os resultados deste treino estão presentes na tabela 5.6. Neste treino, foi o optimizador *RMSprop* que obteve melhores resultados com um valor de 97.69% na *accuracy* de validação. Nas métricas de teste, obteve 98% na *accuracy* de teste e um valor de *F1-Score* de 0.98. Idêntico a estes resultados temos o optimizador Adam com uma *accuracy* de validação de 97.18%, uma *accuracy* de teste de 98% e um *F1-Score* de 0.98. Prosseguimos então o treino utilizando o optimizador *RMSprop*.

Tabela 5.6: Resultados do treino do modelo para classificação de Sésia consoante o optimizador utilizado.

Optimizador/Métrica	***Validation accuracy***	***Test accuracy***	***F1-Score***
Adam	0.9718	**0.98**	**0.98**
Adagrad	0.9329	0.93	0.93
RMSprop	**0.9769**	**0.98**	**0.98**
SGD	0.9335	0.93	0.93

Era necessário agora verificar se *fine tuning* poderia ajudar o nosso modelo a obter melhores valores na classificação desta espécie. Ainda que a diferença tivesse sido inferior a 2%, permitir o treino de 2 blocos ajudou o modelo a obter melhores resultados. Como se encontra ilustrado na tabela 5.7, a abordagem de treinar 2 blocos permitiu ao modelo obter uma *accuracy* de validação de 98.65%, sendo que esta abordagem também foi a que obteve melhores resultados nas métricas de teste de uma forma geral, obtendo 0.99 em ambas as métricas de teste.

Tabela 5.7: Resultados do treino do modelo para classificação de Sésia consoante o número de blocos "descongelados"para treino.

Blocos/Métrica	***Validation accuracy***	***Test accuracy***	***F1-Score***
0 Blocos	0.9724	0.98	0.97
1 Bloco	0.9803	0.98	0.98
2 Blocos	**0.9865**	**0.99**	**0.99**
3 Blocos	0.9853	0.98	0.98
4 Blocos	0.9797	0.97	0.97
5 Blocos	0.9814	0.96	0.96

Finalmente, faltava verificar qual a taxa de aprendizagem que melhor se adequava ao nosso modelo. A tabela 5.8 apresenta os resultados ao efetuar o treino com as mesmas taxas de aprendizagem utilizadas no treino do nosso modelo de classificação de insetos.

Tabela 5.8: Resultados do treino do modelo para classificação de Sésia consoante a taxa de aprendizagem utilizada.

Aprendizagem/Métrica	***Validation accuracy***	***Test accuracy***	***F1-Score***
0.001	0.8952	0.9685	0.9677
0.0001	0.9679	0.9572	0.9540
0.0005	0.7836	0.9730	0.9722
0.00001	**0.9910**	**0.9797**	**0.9787**
0.00005	0.9769	0.9685	0.9673

Olhando uma vez mais para a tabela de resultados, podemos concluir que o melhor valor a utilizar como taxa de aprendizagem é 0.00001. Este valor obteve uma *accuracy* no conjunto de validação de 99.10%, valor que acaba por ser mais do que 1% superior ao segundo melhor valor, que foi obtido pela taxa de 0.00005, que conseguiu um valor de *accuracy* de validação de 97.69%. Também nas métricas de teste, a abordagem que utilizou 0.00001 como taxa de aprendizagem obteve os melhores valores com 97.97% e 0.9787 para a *accuracy* de teste e *F1-Score* respetivamente.

Concluímos então que o modelo a utilizar na tarefa de identificação de insetos da espécie Sésia será um modelo com arquitetura *ResNet101* com treino de dois blocos pertencentes às camadas de extração de atributos e que utiliza o optimizador *RMSprop* com taxa de aprendizagem de 0.00001.

5.2.2 Modelo de classificação de Funebrana

O modelo de classificação de Funebrana seguiu a mesma metodologia do modelo de classificação de Sésia, mas para uma espécie de inseto diferente. A arquitetura escolhida para este modelo foi a arquitetura *ResNet50*, que obteve uma *accuracy* de validação de 96.22%. Mencionar também que foi a arquitetura *ResNet101* que obteve os melhores valores para ambas as métricas de teste. Estes resultados estão presentes na tabela 5.9.

Tabela 5.9: Resultados do treino do modelo para classificação de Funebrana consoante a arquitetura utilizada.

Arquitetura/Métrica	***Validation accuracy***	***Test accuracy***	***F1-Score***
MobileNet	0.8938	0.9301	0.9340
VGG16	0.9565	0.9545	0.9587
VGG19	0.9548	0.9610	0.9645
ResNet50	**0.9622**	0.9642	0.9668
ResNet101	0.9605	**0.9675**	**0.9704**
Inception	0.7066	0.7122	0.7579

Em relação ao optimizador utilizado, aquele que apresentou melhores resultados foi o optimizador *Adam* com um valor de *accuracy* de validação de 96.05%, sendo que este mesmo optimizador foi o que obteve melhores resultados nas métricas de teste, com 97.89% de *accuracy* de teste e um *F1-Score* de 0.9805. A tabela 5.10 ilustra os resultados obtidos durante o treino com variação de optimizadores.

Tabela 5.10: Resultados do treino do modelo para classificação de Funebrana consoante o optimizador utilizado.

Optimizador/Métrica	***Validation accuracy***	***Test accuracy***	***F1-Score***
Adam	**0.9605**	**0.9789**	**0.9805**
Adagrad	0.9373	0.9561	0.9595
RMSprop	0.9593	0.9707	0.9730
SGD	0.9361	0.9447	0.9485

Quanto a *fine tuning*, a tabela 5.11 apresenta os resultados obtidos ao permitir o treino de blocos na fase de extração de atributos. Interpretando os resultados na tabela, podemos concluir uma vez mais que "descongelando" dois blocos para treino se obtém os melhores resultados. Esta abordagem conseguiu uma *accuracy* de validação de 97.31%, tendo sido este o melhor resultado. Nas métricas de teste, a abordagem de treino de apenas um bloco teve os melhores resultados com 98.05% para *accuracy* de teste e um *F1-Score* de 0.9821.

Tabela 5.11: Resultados do treino do modelo para classificação de Funebrana consoante o número de blocos "descongelados"para treino.

Blocos/Métrica	***Validation accuracy***	***Test accuracy***	***F1-Score***
0 Blocos	0.9642	0.9577	0.9604
1 Bloco	0.9670	**0.9805**	**0.9821**
2 Blocos	**0.9731**	0.9691	0.9717
3 Blocos	0.9727	0.9594	0.9634
4 Blocos	0.9703	0.9740	0.9762
5 Blocos	0.9699	0.9659	0.9685

Por fim, a taxa de aprendizagem escolhida para utilizar juntamente com o optimizador foi de 0.00005 uma vez que foi este que obteve um maior valor de *accuracy* de validação com um valor de 97.88%. Quanto às métricas de teste, foi a abordagem com o valor de 0.0001 para taxa de aprendizagem que obteve os melhores valores para a *accuracy* de teste e *F1-Score*, com 97.40% para o valor de *accuracy* e 0.9763 para o valor de *F1-Score*. Os resultados podem ser visualizados na tabela 5.12.

O nosso melhor modelo seguirá então uma arquitetura *ResNet50*, sendo que o optimizador a utilizar é o optimizador *Adam* com uma taxa de aprendizagem de 0.00005 e que permita o treino de 2 blocos pertencentes à fase de extração de atributos.

Tabela 5.12: Resultados do treino do modelo para classificação de Funebrana consoante a taxa de aprendizagem utilizada.

Aprendizagem/Métrica	***Validation accuracy***	***Test accuracy***	***F1-Score***
0.001	0.9552	0.9626	0.9659
0.0001	0.9662	**0.9740**	**0.9763**
0.0005	0.9630	0.9707	0.9733
0.00001	0.9670	0.9626	0.9651
0.00005	**0.9788**	0.9659	0.9689

5.2.3 Modelo de classificação de Bichado

O modelo de classificação de Bichado seguiu a mesma metodologia dos dois modelos de classificação anteriores, mas cujo objetivo é identificar insetos da espécie Bichado. Uma particularidade deste modelo foi o número reduzido de dados. Isto fará com que tivéssemos apenas 20 exemplos positivos e 20 exemplos negativos para teste do modelo, sendo que aumentar o número de exemplos para teste implicaria reduzir o número de exemplos para treino o que não seria necessariamente melhor. O baixo número de exemplos de teste poderá dar um falso valor de confiança em relação ao modelo treinado, podendo ter uma variação nos valores das métricas consoante os exemplos utilizados para teste.

Tendo em conta as circunstâncias previamente descritas, procedeu-se então ao treino dos diferentes modelos. As tabelas 5.13, 5.14, 5.15 e 5.16 ilustram os resultados em função da variação da arquitetura do modelo, optimizador utilizado, número de blocos a treinar e por fim taxa de aprendizagem. Começando pela arquitetura escolhida, tendo em conta os resultados, optou-se por utilizar a arquitetura *ResNet50* que apresentou 95% de *accuracy* de validação. As arquiteturas *ResNet101* e *VGG-16* conseguiram obter o mesmo valor nesta métrica, no entanto são arquiteturas mais pesadas em termos de espaço e mais lentas no momento de inferência. Também no aspeto das métricas de teste, foi esta arquitetura que apresentou valores superiores, sendo os resultados para ambas as arquiteturas *ResNet* idênticos com 90% de *accuracy* de teste e um *F1-Score* de 0.9130. Estes valores acabaram por ser superiores aos valores obtidos utilizando a arquitetura *VGG-16*, que apresentou um valor de 87.5% para *accuracy* de teste e 0.8781 para o *F1-Score*.

Relativamente ao optimizador, aquele que mostrou melhores resultados foi o *RMSprop* com uma diferença superior a 2% relativamente ao segundo melhor optimizador que neste caso foi o optimizador *Adam*. Em relação às métricas de teste, este optimizador conseguiu também os melhores valores, no entanto, neste aspeto o optimizador *Adam* conseguiu obter os mesmos valores que o optimizador *RMSprop*. Foi então selecionado o *RMSprop* como o optimizador a utilizar.

No que diz respeito ao número de blocos a "descongelar", e em contraste com os outros modelos já treinados, para este conjunto de dados, obtivemos os melhores resultados na métrica de *accuracy* de validação quando decidimos permitir o treino de apenas o último bloco da fase de extração de atributos. Com essa abordagem obtivemos uma *accuracy*

de 98.75%, superando as abordagens de não treinar nenhum bloco ou a abordagem de permitir o treino de dois blocos. É importante realçar, no entanto que esta abordagem não foi aquela que apresentou melhores resultados nas métricas de teste, obtendo menos 5% na *accuracy* de teste comparativamente aos melhores resultados e menos 0.0416 no *F1-Score*. Ainda assim e seguindo a mesma metodologia, iremos selecionar a opção que apresenta melhor valor na métrica de validação, que neste caso foi a de permitir o treino de apenas 1 bloco.

Fica então a faltar qual a taxa de aprendizagem a utilizar com o nosso optimizador *RMSprop*. Das abordagens testadas aquela que se revelou superior às outras foi a abordagem que utiliza uma taxa de aprendizagem de 0.0005. Esta obteve um resultado de *accuracy* de validação de 97.50%, a mesma percentagem apresentada pela abordagem que utilizou uma taxa de 0.0001, no entanto, esta última abordagem acabou por obter um valor mais baixo nas métricas de teste pelo que se optou pela primeira opção.

Tabela 5.13: Resultados do treino do modelo para classificação de Bichado consoante a arquitetura utilizada.

Arquitetura/Métrica	***Validation accuracy***	***Test accuracy***	***F1-Score***
MobileNet	0.8313	**0.90**	0.9048
VGG16	**0.95**	0.875	0.8781
VGG19	0.9063	0.85	0.8696
ResNet50	**0.95**	**0.90**	**0.9130**
ResNet101	**0.95**	**0.90**	**0.9130**
Inception	0.6688	0.75	0.7368

Tabela 5.14: Resultados do treino do modelo para classificação de Bichado consoante o optimizador utilizado.

Optimizador/Métrica	***Validation accuracy***	***Test accuracy***	***F1-Score***
Adam	0.9563	**0.925**	**0.9333**
Adagrad	0.85	0.875	0.8889
RMSprop	**0.9813**	**0.925**	**0.9333**
SGD	0.7188	0.70	0.7391

5.3 Modelos utilizados

De forma a resumir aquilo que foi a apresentação dos diferentes resultados do treino dos modelos, apresentamos agora os modelos cuja performance foi avaliada com recurso aos 20% que sobrou para conjunto de teste, e que acabaram por ser utilizados na nossa aplicação.

Tabela 5.15: Resultados do treino do modelo para classificação de Bichado consoante o número de blocos "descongelados"para treino.

Blocos/Métrica	*Validation accuracy*	*Test accuracy*	*F1-Score*
0 Blocos	0.9725	**0.95**	**0.9546**
1 Bloco	**0.9875**	0.90	0.9130
2 Blocos	0.975	**0.95**	**0.9546**
3 Blocos	0.9688	**0.95**	**0.9546**
4 Blocos	0.9563	0.90	0.9130
5 Blocos	0.9625	**0.95**	**0.9546**

Tabela 5.16: Resultados do treino do modelo para classificação de Bichado consoante a taxa de aprendizagem utilizada.

Aprendizagem/Métrica	*Validation accuracy*	*Test accuracy*	*F1-Score*
0.001	0.9625	**0.95**	**0.9546**
0.0001	**0.975**	0.90	0.9130
0.0005	**0.975**	**0.95**	**0.9546**
0.00001	0.9563	**0.95**	0.9524
0.00005	0.9688	0.90	0.9130

5.3.1 Modelo de classificação de insetos

O modelo de classificação de insetos tem o objetivo de separar o lixo presente na placa, dos insetos que tenham sido capturados pela armadilha. Esta classificação ocorre após a fase de obtenção de regiões candidatas. As regiões são consideradas candidatas se cumprirem requisitos relacionados com o número de pixéis brancos e pretos numa certa área de 224x224, isto porque se o número de pixéis a branco for extremamente alto então o mais certo é esta área não conter um inseto, que se encontra marcado com pixéis pretos na máscara binária. Com uma ideia idêntica se pode desprezar áreas com um número elevado de pixéis pretos, uma vez que para ser um inseto presente na placa, a área necessariamente tem pixéis a branco associados à placa.

Tendo em conta os resultados demonstrados, a arquitetura que acabou por ser escolhida para este modelo foi a *ResNet101*, cuja diferença para a arquitetura *ResNet50* apresentada no capítulo de Estado de Arte é o número de camadas, nomeadamente no quarto bloco de camadas. Este modelo foi treinado com todos os tipos de insetos presentes nas imagens das armadilhas que nos foram disponibilizadas, mais concretamente com as imagens das armadilhas Funebrana, Sésia e Bichado. O optimizador utilizado foi o *Adam* (*Adaptive Moment Estimation*) e a função de custo/perda foi a entropia cruzada binária. Este optimizador tende a ser um bom optimizador de uma forma geral, no entanto poderá ser computacionalmente caro. Utilizou-se uma taxa de aprendizagem de 0.00001 juntamente com o optimizador e por fim recorreu-se a *fine-tuning*, sendo que se decidiu "descongelar" para treino 2 blocos pertencentes às camadas de extração de atributos.

5.3.2 Modelos de classificação de espécie (Sésia, Funebrana e Bichado)

Os modelos de classificação de espécie, especializam-se na classificação de insetos de modo a identificar os insetos pertencentes a cada uma das espécies em estudo. Esta classificação ocorre após a classificação efetuada pelo modelo dos insetos, pelo que esta classificação irá separar os insetos de outras espécies irrelevantes para o estudo, que se encontravam presentes na mesma armadilha, dos insetos da espécie que se pretende observar. O modelo recebe então as regiões que foram consideradas como sendo insetos pelo modelo de classificação de insetos sendo que posteriormente irá efetuar uma classificação do tipo binário separando as regiões em duas classes, Espécie e não Espécie (*e.g.* Sésia e não Sésia, Funebrana e não Funebrana). Após esta classificação é possível apresentar o número de insetos para a espécie em estudo presentes na imagem original. Treinámos então, um modelo deste tipo para cada uma das espécies que se pretendiam observar.

5.3.2.1 Modelo de classificação da espécie Sésia

A arquitetura escolhida para este modelo foi a arquitetura *ResNet101*, a mesma utilizada pelo classificador de insetos. Optou-se por esta arquitetura uma vez que não foram impostas restrições em relação ao tempo necessário para treino nem ao tamanho ocupado pelo modelo, fatores que poderiam ter influência na escolha do modelo a utilizar. Este modelo foi treinado utilizando duas classes, uma com imagens de insetos da espécie Sésia presentes nas placas e outra classe para insetos que estivessem presentes nas placas destinadas à espécie Sésia. O optimizador utilizado neste caso foi o *RMSprop* (*Root Mean Squares Propagation*) e a função de custo/perda foi a entropia cruzada binária. Uma vantagem deste optimizador é ele convergir rapidamente. A taxa de aprendizagem escolhida para este modelo foi 0.00001 e utilizou-se *fine-tuning*, sendo que se decidiu "descongelar" para treino 2 blocos pertencentes às camadas de extração de atributos.

5.3.2.2 Modelo de classificação da espécie Funebrana

A arquitetura escolhida para este modelo foi a arquitetura *ResNet50*. Este modelo foi treinado utilizando duas classes, uma com imagens de insetos da espécie Funebrana presentes nas placas e outra classe para insetos que estivessem presentes nas placas destinadas à espécie Funebrana. O optimizador utilizado neste caso foi o *Adam*, a taxa de aprendizagem utilizada foi 0.00005 e a função de custo/perda foi uma vez mais a entropia cruzada binária. Utilizou-se *fine-tuning*, sendo que se decidiu "descongelar" para treino 2 blocos pertencentes às camadas de extração de atributos.

5.3.2.3 Modelo de classificação da espécie Bichado

Para este modelo a arquitetura escolhida foi a *ResNet50*. Uma particularidade deste modelo treinado é o número claramente inferior de dados para treino, tendo sido selecionado treinar um modelo para esta espécie sub-representada de forma a comparar com

modelos cujas espécies tinham maior representação. Uma vez mais separou-se as classes para treino em imagens da espécie Bichado e imagens de insetos não-relevantes para o estudo em causa. O optimizador utilizado neste caso foi o *RMSprop*, com uma taxa de aprendizagem de 0.0005 e a função de custo/perda foi uma vez mais a entropia cruzada binária. Utilizou-se *fine-tuning*, sendo que se decidiu "descongelar" para treino apenas um bloco pertencente às camadas de extração de atributos.

5.4 Performance do modelo de classificação de insetos

Estando concluído o treino do modelo que irá fazer a classificação de insetos, iremos agora apresentar nesta secção a performance desse modelo no conjunto de teste.

A matriz de confusão presente na figura 5.1 mostra os resultados das previsões do nosso melhor modelo no conjunto de teste, conjunto que representava 20% do nosso conjunto de dados total.

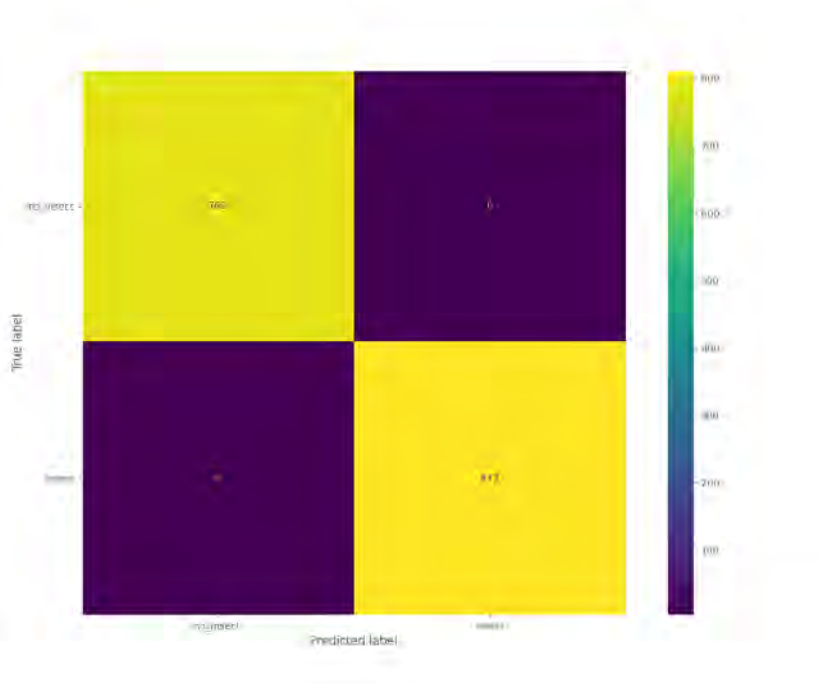


Figura 5.1: Matriz de confusão resultante das previsões do melhor modelo de classificação de insetos sobre o conjunto de teste

Como podemos observar, existem 6 casos em que o nosso modelo classificou de forma errada uma imagem como sendo a imagem de um inseto. Existem diferentes possibilidades para a razão pela qual se obtiveram estes 6 falsos positivos. Uma das razões, é o motivo pelo qual se criou este modelo inicialmente. Na presença de objetos não comuns no plano de fundo ou que apareçam na placa numa das imagens, se esse objeto não se encontrar presente no conjunto de treino, o modelo poderá classificar esse objeto como sendo um inseto de forma errada. Uma outra possibilidade é o modelo reconhecer certas

partes dos corpos dos insetos, como por exemplo, as asas ou o corpo em si, como sendo o inseto por completo. Quanto à presença de 4 falsos negativos, este problema pode estar relacionado com a falta de dados relativos àquela espécie de inseto presente na placa que serve de armadilha. Ainda assim, o facto de obtermos 812 verdadeiros positivos acaba por revelar que de um aspeto geral o modelo tem um comportamento próximo de excelente.

De forma a condensar o que foi dito anteriormente, podemos recorrer à métrica F1 de forma a ter um valor mais expressivo. Utilizando então a fórmula 3.3 presente no capítulo 3, este modelo obteve um *F1-Score* de 0.99 neste conjunto de teste, um valor quase perfeito para esta métrica.

5.5 Performance dos modelos de classificação de espécies

Da mesma maneira que testámos a performance do melhor modelo treinado como classificador de insetos, também testámos a performance de todos os modelos de espécies treinados previamente sendo os resultados apresentados de seguida. Dependendo da espécie, poderá haver justificações diferentes ou similares em relação ao porquê de o modelo errar em certas instâncias.

Comecemos por apresentar os resultados obtidos pelo nosso modelo de classificação de Funebrana no conjunto que foi designado como sendo de teste. Este conjunto representa 20% do conjunto total de dados referentes às armadilhas de Funebrana.

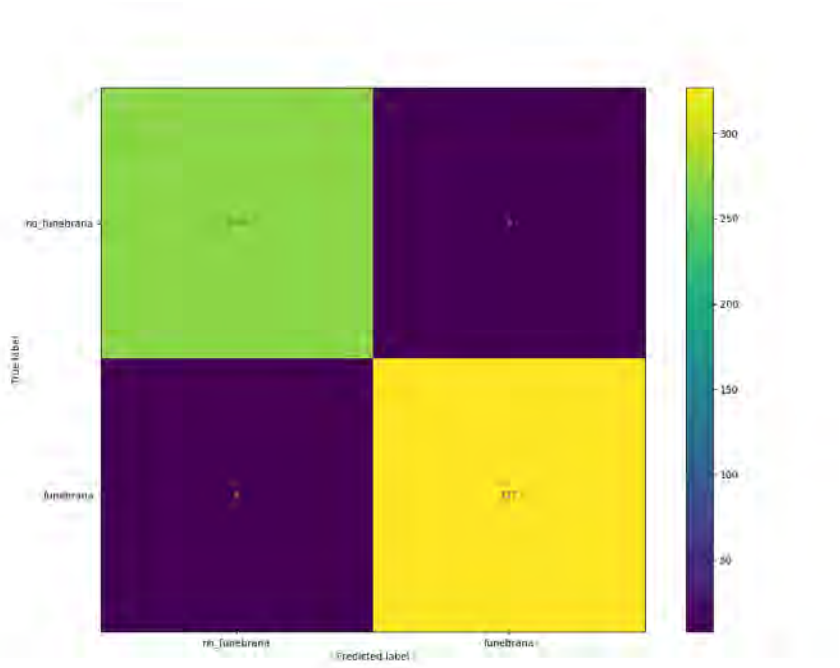


Figura 5.2: Matriz de confusão resultante das previsões do melhor modelo de classificação de Funebrana sobre o conjunto de teste

Como podemos calcular a partir da matriz de confusão presente na figura 5.2, este

modelo tem um valor de *F1-Score* de 0.97, sendo um modelo bastante bom na identificação de indivíduos desta espécie.

Analisando mais pormenorizadamente, o nosso modelo apresentou o mesmo número de falsos positivos e negativos, valor o qual foi substancialmente inferior ao número de verdadeiros positivos que o modelo obteve, tendo o modelo apresentado 327 verdadeiros positivos e apenas 8 falsos positivos e/ou 8 falsos negativos, perfazendo um total de 16 classificações erradas num universo de 615 exemplos. A figura 5.3 pretende ilustrar os falsos positivos obtidos, enquanto que a figura 5.4 ilustra os falsos negativos.

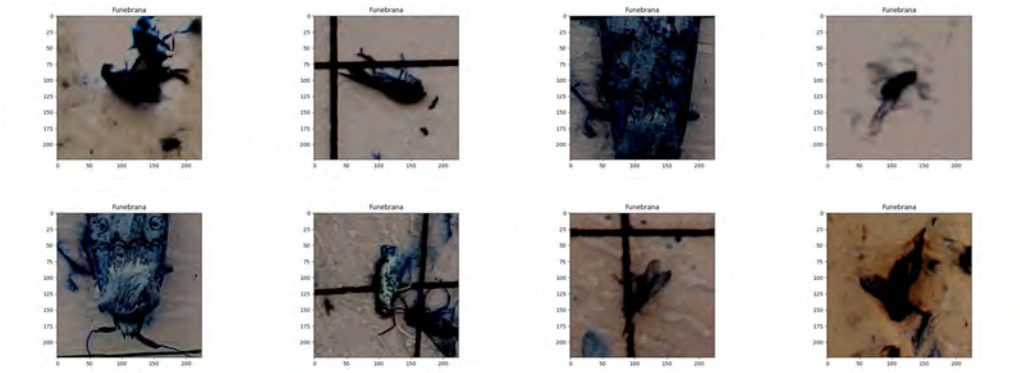


Figura 5.3: Falsos positivos gerados pelo modelo de classificação de Funebrana

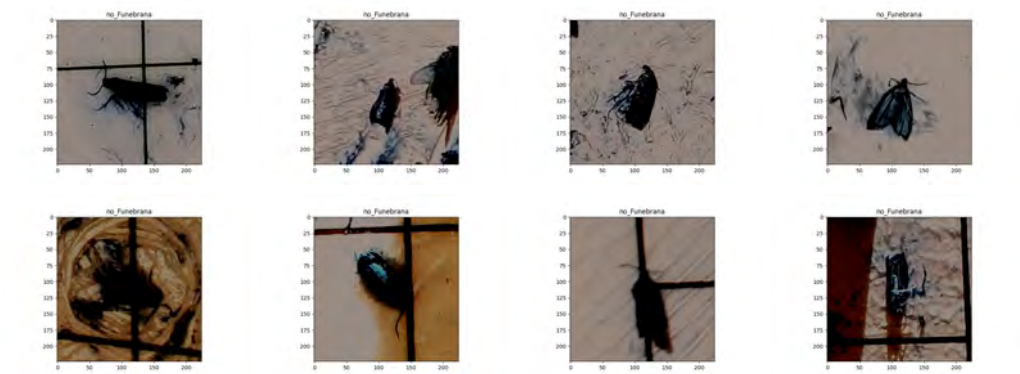


Figura 5.4: Falsos negativos gerados pelo modelo de classificação de Funebrana

Seguimos o mesmo procedimento para avaliar os nossos modelos de classificação de Bichado e Sésia.

O nosso modelo de classificação de Bichado apresentou um valor de 0.91 na métrica *F1-Score*, resultado obtido após o cálculo efetuado com os dados presentes na figura 5.5.

Este modelo tem um comportamento interessante, podendo existir variações quando utilizado um conjunto de teste diferente, uma vez que o número de espécies testadas é baixo e após uma análise mais cautelosa, chegámos à conclusão que 2 dos 3 falsos positivos acabaram por pertencer a um inseto de uma outra espécie que provavelmente acabou por não ter representação no conjunto de treino. A figura 5.6 mostra os 3 falsos positivos gerados por este modelo, para o conjunto de teste utilizado.

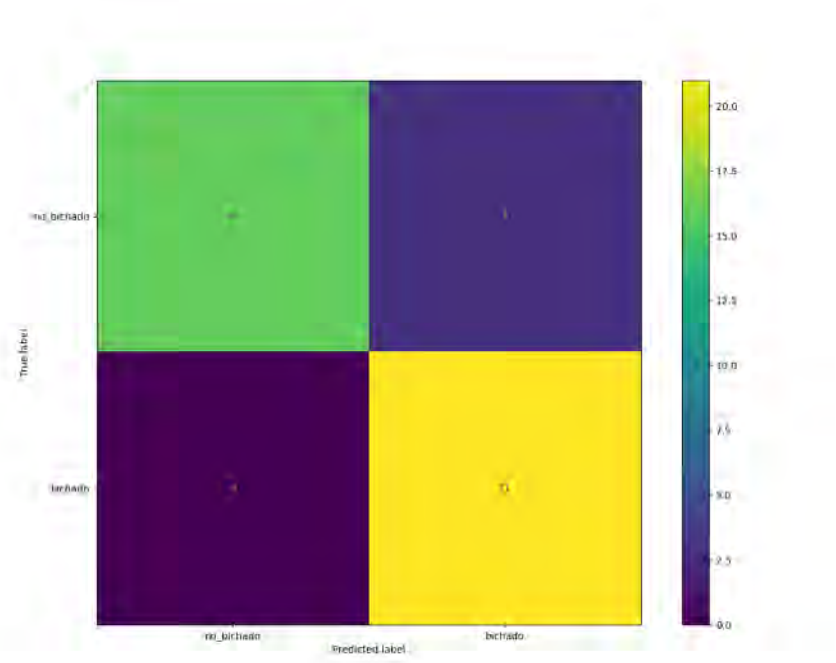


Figura 5.5: Matriz de confusão resultante das previsões do melhor modelo de classificação de Bichado sobre o conjunto de teste

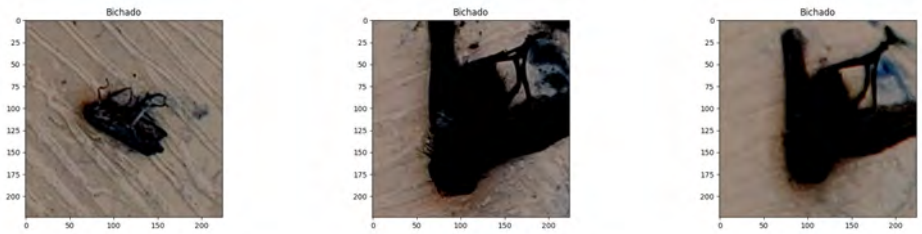


Figura 5.6: Falsos negativos gerados pelo modelo de classificação de Bichado

Este problema acabaria por se repetir com um outro modelo treinado, neste caso o de Sésia em que uma outra espécie, que por acaso apresentava os mesmos tons de cor de laranja que a espécie que se pretendia monitorizar, acabaria por ter pouca representação na classe negativa do conjunto de treino o que levaria a que o nosso modelo acabasse por a identificar como sendo pertencente à espécie a monitorizar levando a que o nosso modelo apresentasse falsos positivos. O inseto em causa encontra-se representado na figura 5.7. Apesar deste problema, que acabaria por levar o modelo a obter 3 falsos positivos, o modelo apresentou ainda 5 falsos negativos, os quais se encontram representados na figura 5.8. Uma possível explicação para a ocorrência de falsos negativos prende-se com a qualidade das imagens que foram apresentadas ao modelo. Este fator, assim como variações na apresentação em geral do inseto, por exemplo, se este tem as asas abertas ou encobertas, podem levar a que o modelo acabe por não reconhecer o mesmo. Apesar destes casos em que o modelo fez uma previsão incorreta, o número de verdadeiros positivos supera e em muito estas classificações incorretas, uma vez que temos 216 verdadeiros

positivos. A matriz de confusão referente a este modelo encontra-se na figura 5.9.

Figura 5.7: Inseto de uma outra espécie que acaba por ser incorretamente classificado como sendo Sésia

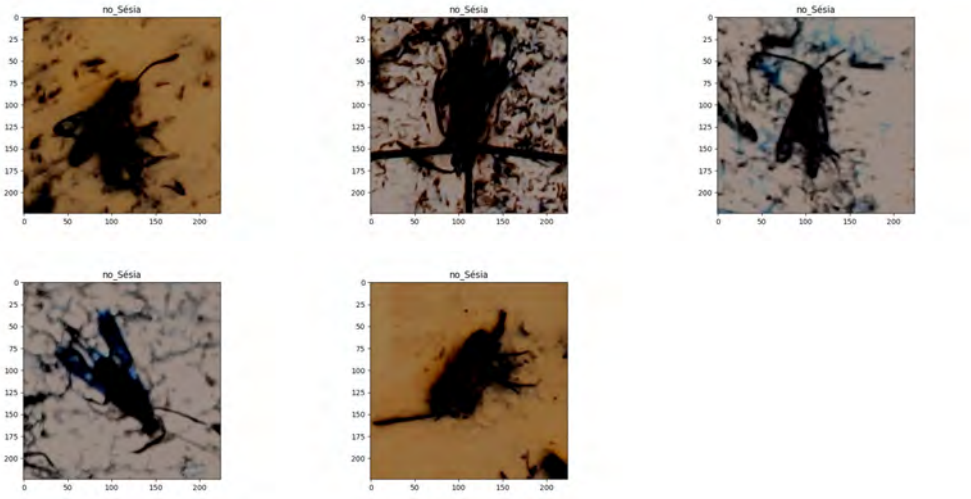


Figura 5.8: Falsos negativos gerados pelo modelo de classificação de Sésia

Podemos agora calcular o valor de *F1-Score* para o nosso modelo de classificação de Sésia, o qual acabou por apresentar um valor de 0.98 para esta métrica no conjunto de teste.

5.6 Performance dos modelos no detetor de objetos

Após se testar cada um dos modelos, queríamos agora verificar a performance dos mesmos ao se utilizar a nossa aplicação de contagem automática de insetos. Assim, escolheu-se um dos modelos de espécies, que neste caso foi o da espécie Sésia, sendo uma espécie relativamente mais fácil de detetar e distinguir de outros insetos, e percorrendo todas as imagens relativas a essa espécie colocámos o nosso modelo de insetos e de Sésia à prova numa experiência aproximada daquela em que o modelo seria utilizado no dia a dia.

Após correr o nosso programa detetor e classificador de objetos, conseguimos construir uma matriz de confusão associada à nossa capacidade de identificar Sésias. Os resultados

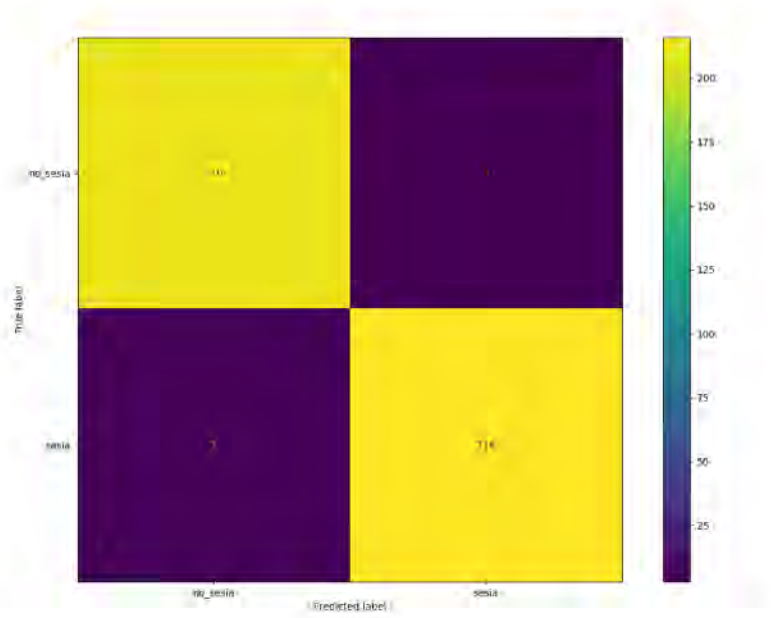


Figura 5.9: Matriz de confusão resultante das previsões do melhor modelo de classificação de Sésia sobre o conjunto de teste

estão presentes na figura 5.10.

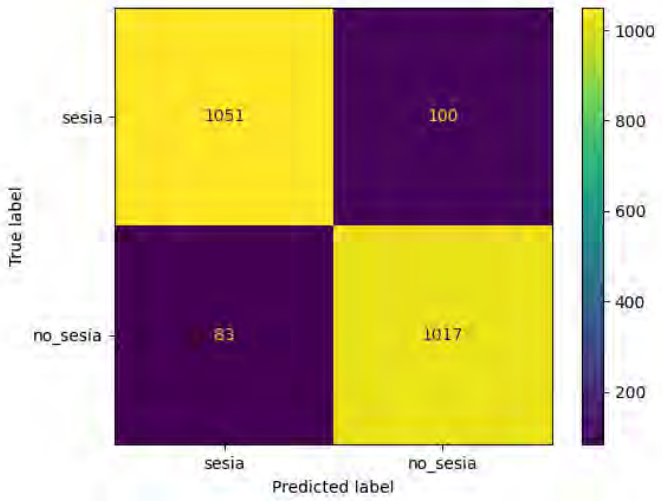


Figura 5.10: Matriz de confusão resultante das previsões do melhor modelo de classificação de Sésia para cada uma das imagens originais

Utilizando uma vez mais a fórmula 3.3 presente no capítulo 3, podemos obter o valor da métrica *F1-Score* para o modelo que treinámos. Obtemos então um valor de 0.92, valor o qual revela que o nosso modelo de identificação de Sésia é muito bom, mas tem espaço para melhorias.

O número de falsos positivos neste caso, para além do problema do inseto "impostor", prende-se também com a ineficácia do modelo de insetos de conseguir filtrar partes do corpo de insetos que já não se encontram na placa, levando a que por exemplo asas de uma Sésia que se encontrava presente na placa na semana anterior, acabem por ser classificadas como insetos e posteriormente o nosso modelo de Sésia irá identificar aquelas asas como pertencendo a um inseto da espécie Sésia, levando a um falso positivo. A figura 5.11 ilustra este problema. A imagem 5.11(a) precede a imagem 5.11(b) em 2 semanas, e como podemos ver, a zona marcada a vermelho na imagem 5.11(a) continha insetos da espécie Sésia naquela semana. No processo de arrancarem os insetos da placa, partes do corpo desses insetos acabaram por permanecer, o que levou a que o modelo na imagem 5.11(b) acabe por reconhecer a asa como sendo um inseto da espécie.

(a) Placa com insetos. A vermelho zona onde ficarão partes dos corpos destes insetos

(b) Modelo acaba por reconhecer asa como sendo Sésia

Figura 5.11: Partes de insetos acabam por permanecer na placa após semanas consecutivas

Tentou-se evitar este problema no teste efetuado, no entanto, pelo menos 6 dos 83 falsos positivos podem ser inseridos neste tipo de problema. Os restantes falsos positivos, estão maioritariamente associados a insetos que podem ser confundidos como sendo da espécie Sésia, havendo alguma confusão na identificação de alguns insetos, que é preciso realçar uma vez mais, foi efetuada por nós. De modo a melhorar a performance do modelo neste tipo de insetos, será necessário reforçar a representação dos mesmos na classe negativa do conjunto de treino do modelo de classificação de espécie. Esta sub-representação da espécie no nosso trabalho, está relacionada com a incerteza e dificuldade em identificação dos insetos de outras espécies parecidas em formato ao da nossa espécie a monitorizar. A figura 5.12 demonstra insetos de outras espécies que acabam por gerar

falsos positivos, problema que acaba por se repetir em outras imagens de placas.

Figura 5.12: Exemplo de output do programa com retificação, de forma a demonstrar falsos positivos, representados com caixa envolvente vermelha.

Quanto a falsos negativos, para além do problema de incapacidade de o modelo extrair atributos em casos em que os insetos têm o corpo numa posição estranha, no caso do detetor de objetos, existe a possibilidade de certos insetos não serem reconhecidos pelo modelo de classificação de insetos, pelo que estes não chegarão a ser classificados pelo modelo de classificação de Sésias, ou existe também a possibilidade de alguns insetos serem removidos, pois devido à proximidade entre insetos, o algoritmo de supressão de não-máximos acabará por ignorar estes. Existe ainda um terceiro fator, que neste caso remete à fase de proposição de regiões, onde caso a máscara binária esteja mal gerada,

isto é, zona de insetos acaba por estar à sombra ou cheia de sujidade, esta região acabará por ser desprezada, não sendo considerada sequer no nosso modelo de classificação de insetos.

Certificámo-nos de que o classificador de Sésias recebia todas as regiões a classificar, inclusive as que se perdiam devido às razões previamente descritas, sendo que ainda assim obtivemos 100 falsos negativos. Este valor é proveniente da incapacidade de o modelo de classificação de Sésia reconhecer certos atributos que extraiu na fase de treino. Esta incapacidade estará relacionada com o estado em que o inseto se encontra, existindo casos em que os insetos se encontram espalmados, estado da placa, ou seja, a quantidade de sujidade e se a placa se encontra em boas condições. Existe ainda a possibilidade de a região que é recebida por este modelo não a ser a mais propícia, apanhando apenas parte de o inseto, em vez de o apanhar por completo e existe ainda a influência da qualidade da imagem em si algo que acaba por afetar a performance do modelo.

É necessário mencionar ainda que, ocorreram 21 duplas contagens, ou seja, um inseto que gerou duas caixas envolventes, mesmo após o algoritmo de supressão de não-máximos, sendo que ocorreu ainda uma tripla contagem. Em contrabalanço, tivemos pelo menos 4 situações em que ocorreu uma perda de insetos, isto é, uma caixa acabou por encobrir mais do que um inseto. A figura 5.13 ilustra a ocorrência da tripla contagem, previamente mencionada.

5.7 Estatísticas de desempenho da aplicação

Após apresentarmos os resultados associados à performance dos modelos presentes na nossa aplicação, nesta secção pretendemos demonstrar o tempo que a nossa aplicação demora a processar uma imagem e a obter a contagem final de insetos da espécie que se pretende monitorizar. É difícil obter um resultado fixo relacionado com o tempo que demora todo o processo associado à nossa aplicação, uma vez que o mesmo depende da quantidade de regiões que são originalmente enviadas para o primeiro modelo de classificação e aquelas que são enviadas posteriormente para o segundo modelo. Ou seja, por norma uma imagem de uma placa que contenha mais sujidade irá sempre demorar consideravelmente mais tempo do que uma outra placa que esteja praticamente limpa, uma vez que o número de regiões que serão enviadas para o primeiro modelo será muito superior. Um outro fator que poderá influenciar a demora do modelo prende-se com o número de insetos presentes na placa. Quanto mais regiões da placa forem consideradas regiões de insetos, maior o número de inferências a efetuar pelo nosso segundo modelo.

Dito isto, para a imagem de exemplo presente na figura 4.18, o nosso programa demorou 15.40 segundos na fase de recorte e obtenção de regiões propícias a conterem insetos, 24 segundos na inferência associada ao primeiro modelo de classificação, 0.014 segundos para o algoritmo de supressão de não-máximos e por fim 2 segundos para a classificação efetuada pelo modelo de classificação de espécie. Estes resultados foram obtidos utilizando a GPU nos momentos de classificação por parte dos modelos. Com o recurso da

Figura 5.13: Output gerado que contém uma tripla contagem (um inseto presente em 3 caixas envolventes) na zona inferior direita da imagem.

GPU, o primeiro modelo demora 89ms por cada momento de inferência, perfazendo um total de 24 segundos. Sem recurso da GPU, isto é, utilizando o CPU, esta operação demora agora 800ms por cada momento de inferência, perfazendo um total de 3 minutos para o processo de classificação de todas as regiões.

Testando agora a performance do programa para uma imagem com maior nível de sujidade, como é exemplo a imagem da figura 5.14, o nosso modelo demora 24.40 segundos no recorte e obtenção de regiões, 45 segundos na inferência do primeiro modelo, sendo que o tempo por cada inferência é o mesmo que no exemplo anterior, como seria de esperar, 0.03 segundos para o algoritmo de supressão de não-máximos e por fim 2 segundos na classificação por parte do último modelo. Testou-se uma vez mais correr o programa sem recorrer à GPU sendo que agora a inferência associada ao primeiro modelo passou a demorar 803 ms por cada momento de inferência com um total de 5.97 minutos

para o processo completo.

Figura 5.14: Exemplo de imagem com placa com maior quantidade de sujidade

Assim com recurso a GPU, o nosso programa é capaz de terminar o processo de contagem de insetos em aproximadamente 42 segundos no caso de uma placa relativamente limpa e aproximadamente 1 minuto e 11 segundos no caso de uma placa com uma grande quantidade de sujidade.

5.8 Conclusões

Concluindo tudo o que foi discutido neste capítulo, os modelos treinados apresentaram resultados muito bons no conjunto destinado ao teste dos mesmos, existindo justificação para os poucos casos em que os modelos acabam por errar, sendo que no caso do modelo de classificação de insetos, o modelo acaba por gerar falsos positivos, uma vez que confunde partes de insetos, por exemplo as asas, como sendo o inseto completo. Os falsos negativos podem estar relacionados com pouca representação daquele inseto no conjunto de treino, o que poderá levar este modelo a classificar esse inseto como sendo um objeto estranho.

Quanto aos modelos de classificação de espécies, no caso da Sésia, os falsos positivos acabavam por estar relacionados com insetos que ou tinham um formato idêntico ao da Sésia, ou tinham o mesmo tom de laranja no seu corpo. Isto significa que poderá ser necessário aumentar a representação destes insetos "estranhos" no conjunto de treino. O mesmo problema ocorreu no modelo de classificação de Bichado, onde um inseto que

tinha pouca representação no conjunto de treino, acabou por ser identificado como sendo Bichado, incorretamente. Uma particularidade desde modelo foi o de não apresentar falsos negativos. Quanto ao modelo de classificação de Funebrana, este apresentou o mesmo número de falsos positivos e falsos negativos.

Posteriormente, ao correrm o nosso programa de modo a testar os nossos modelos, percebemos que o nosso modelo de insetos não estaria a ter a capacidade de filtrar partes de corpo de insetos que já não se encontravam na placa, o que levou a um número elevado de falsos positivos associados ao classificador de Sésia quando aplicado no nosso programa. Os nossos modelos, têm capacidade então de reconhecer partes do corpo de um inseto, e classificar corretamente a espécie, ainda que ela já não esteja presente na placa. Um exemplo disto é demonstrado na figura 5.11. Filtrando manualmente este problema, acabámos por obter um valor de 0.92 de *F1-Score* com 100 falsos negativos, os quais se prendem com problemas associados à qualidade de imagem, estado de preservação do inseto e da placa assim como o modo como o inseto que foi capturado se apresenta (nomeadamente a posição das asas) e 83 falsos positivos, relacionados com espécies semelhantes àquela que se está a monitorizar, assim como incapacidade de filtragem de partes pertencentes a insetos que já não se encontram presentes na placa. De forma a melhorar estes resultados, um aumento de casos negativos, nomeadamente o das espécies com formatos semelhantes, no conjunto de treino do modelo de classificação de Sésia poderá diminuir o caso de falsos positivos, enquanto para o caso dos falsos negativos poderá ser necessário aumentar o número de casos positivos em diferentes estados de preservação e/ou posição das asas de forma a obter melhores resultados neste parâmetro.

6

Conclusão

Este capítulo fará um balanço final daquilo que foi alcançado no projeto desenvolvido tendo em conta os objetivos iniciais e dará uma perspetiva de possível trabalho a efetuar no futuro de forma a melhorar a experiência do utilizador e melhorar ainda mais a performance do programa final.

6.1 Conclusão

Concluindo, o objetivo desta dissertação tinha em vista o desenvolvimento de um sistema capaz de efetuar o trabalho de outros sistemas já existentes, mas que por diferentes razões, quer sejam económicas quer sejam razões associadas às condições do meio envolvente, não estariam disponíveis para qualquer agricultor. Assim, o sistema desenvolvido tem capacidade para efetuar uma contagem automática de insetos presentes em imagens tiradas com a câmara de um telemóvel, com um bom valor de exatidão e precisão. O programa foi desenvolvido de forma a permitir a classificação e contagem de insetos em condições abaixo de ótimas, não necessitando de uma câmara fotográfica excelente de forma a obter uma imagem sem qualquer tipo de problemas e não necessitando de um protocolo rigoroso na forma como as placas devem ser fotografadas ou as condições das mesmas. Assim, os modelos treinados e integrados no programa têm condições para efetuar classificações e posteriormente contagem em placas com presença de lixo, terra, ervas e até pedaços de insetos que acabam por ser também reconhecidos pelo modelo.

Para além do sistema principal, foram ainda desenvolvidos programas auxiliares de forma a tornar o futuro processo de obtenção de dados para treino um processo mais simples e célere. Através destas ferramentas auxiliares, foi possível gerar um conjunto de dados anotados com mais de 4000 insetos. Foi ainda desenvolvido um algoritmo de ajustamento de imagem à placa de armadilha e um programa de treino de modelos de forma a permitir o estudo de qual os melhores modelos a utilizar na deteção e classificação de novas espécies.

Assim, de uma forma geral os resultados satisfazem as necessidades que foram delineadas no princípio do trabalho e que foram apresentadas no capítulo inicial desta

dissertação.

6.2 Trabalho futuro

Quanto a trabalho futuro, acredito que existe uma boa margem para melhorar a performance do programa tendo em vista os mesmos objetivos. Primeiramente, após o novo protocolo em relação à forma como as placas são fotografadas, pode-se voltar ao início do trabalho e verificar se é possível melhorar ainda mais o recorte da zona de placa, algo que se revelou difícil devido às zonas de sombra e luz simultâneas na mesma imagem. Pode-se então voltar a aplicar algoritmos como o algoritmo que usufruía do espaço de Hough para descobrir as retas pertencentes à placa, ou então o processo de fazer a análise do sinal da sigmoide.

Associado a este problema está o processo de segmentação dos insetos, que acabou por ser uma ideia da qual se desistiu, mas que se pode revelar bastante útil, uma vez que ajudaria o processo de obtenção de dados que se tornaria assim automático e acabaria por servir também na fase de obtenção de regiões propícias a classificação por parte do modelo, sendo que é esta fase que acaba por ser o "*bottleneck*" da nossa aproximação.

Outro aspeto que poderia acabar por ser interessante explorar, seria a remoção automática de insetos que se encontram na mesma posição em duas semanas consecutivas, evitando-se assim contagens duplas.

Por fim, no aspeto de modelos de classificação, poderá ser interessante testar a performance de um modelo multiclasse, visto que foi uma abordagem explorada superficialmente e poderá ser interessante também, verificar as atualizações nas arquiteturas de aprendizagem profunda especializadas na classificação de imagens, uma vez que é uma área em constante evolução.

Bibliografia

[1] *A agricultura e as alterações climáticas*. URL: https://www.eea.europa.eu/pt/sinais-da-aea/sinais-2015/artigos/a-agricultura-e-as-alteracoes-climaticas (acedido em 29/01/2022) (ver p. 1).

[2] M. C. Bakkay et al. "Automatic Detection of Individual and Touching Moths from Trap Images by Combining Contour-based and Region-based Segmentation". Em: *IET Computer Vision* 12 (out. de 2017). DOI: 10.1049/iet-cvi.2017.0086 (ver p. 6).

[3] I. Bechar et al. "On-Line Video Recognition and Counting of Harmful Insects". Em: *2010 20th International Conference on Pattern Recognition*. 2010, pp. 4068–4071. DOI: 10.1109/ICPR.2010.989 (ver p. 7).

[4] K. Bjerge, H. M. Mann e T. T. Høye. "Real-time insect tracking and monitoring with computer vision and deep learning". Em: *Remote Sensing in Ecology and Conservation* (2021). ISSN: 20563485. DOI: 10.1002/rse2.245 (ver pp. 3, 26).

[5] G. Bradski. "The OpenCV Library". Em: *Dr. Dobb's Journal of Software Tools* (2000) (ver p. 47).

[6] J. Cho et al. "Automatic identification of whiteflies, aphids and thrips in greenhouse based on image analysis". Em: *International Journal of Mathematics and Computers in Simulation* 1 (jan. de 2007), pp. 46–53 (ver p. 5).

[7] F. Chollet et al. *Keras*. https://keras.io. 2015 (ver pp. 36, 62).

[8] *Cloud-automated image analysis*. URL: https://www.apeer.com/home/ (acedido em 19/09/2023) (ver p. 52).

[9] T. M. Cover e P. E. Hart. *Approximate formulas for the information transmitted bv a discrete communication channel*. 1952 (ver p. 52).

[10] N. Dalal e B. Triggs. "Histograms of Oriented Gradients for Human Detection". Em: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2005)* 2 (jun. de 2005) (ver p. 6).

[11] L. Deng et al. "Research on insect pest image detection and recognition based on bio-inspired methods". Em: *Biosystems Engineering* 169 (2018), pp. 139–148. ISSN: 1537-5110. DOI: https://doi.org/10.1016/j.biosystemseng.2018.02.008. URL: https://www.sciencedirect.com/science/article/pii/S1537511017302969 (ver p. 26).

[12] J. Duchi, E. Hazan e Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." Em: vol. 12. Jan. de 2010, pp. 257–269 (ver p. 63).

[13] R. Duda e P. Hart. "Use of the Hough Transformation To Detect Lines and Curves in Pictures". Em: *Commun. ACM* 15 (jan. de 1972), pp. 11–15. DOI: 10.1145/361237.361242 (ver p. 55).

[14] P. Getreuer. "Chan-Vese Segmentation". Em: *Image Processing On Line* 2 (ago. de 2012), pp. 214–224. DOI: 10.5201/ipol.2012.g-cv (ver p. 42).

[15] R. Girshick. "Fast r-cnn". Em: (abr. de 2015). DOI: 10.1109/ICCV.2015.169 (ver pp. 11, 12).

[16] R. Girshick et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". Em: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (nov. de 2013). DOI: 10.1109/CVPR.2014.81 (ver pp. 11, 12).

[17] A. Gupta et al. "Deep Learning in Image Cytometry: A Review". Em: *Cytometry Part A* 95 (dez. de 2018). DOI: 10.1002/cyto.a.23701 (ver p. 10).

[18] K. He et al. "Deep Residual Learning for Image Recognition". Em: (dez. de 2015). URL: http://arxiv.org/abs/1512.03385 (ver pp. 20, 63).

[19] T. K. Ho. "Random decision forests". Em: *Proceedings of 3rd international conference on document analysis and recognition*. Vol. 1. IEEE. 1995, pp. 278–282 (ver p. 52).

[20] S. J. Hong et al. "Automatic pest counting from pheromone trap images using deep learning object detectors for matsucoccus thunbergianae monitoring". Em: *Insects* 12 (4 abr. de 2021). ISSN: 20754450. DOI: 10.3390/insects12040342 (ver pp. 1, 3, 5, 21, 22).

[21] J. Hosang, R. Benenson e B. Schiele. "Learning non-maximum suppression". Em: (mai. de 2017) (ver p. 64).

[22] A. G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". Em: (abr. de 2017). URL: http://arxiv.org/abs/1704.04861 (ver pp. 20, 63).

[23] T. Kasinathan, D. Singaraju e S. R. Uyyala. "Insect classification and detection in field crops using modern machine learning techniques". Em: *Information Processing in Agriculture* 8 (3 set. de 2021), pp. 446–457. ISSN: 22143173. DOI: 10.1016/j.inpa.2020.09.006 (ver pp. 3, 23, 26).

[24] M. Kass e A. Witkin. *Snakes: Active Contour Models*. 1988, pp. 321–331 (ver p. 6).

[25] N. Kato et al. "A handwritten character recognition system using directional element feature and asymmetric Mahalanobis distance". Em: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 21(3) (abr. de 1999), pp. 258–262. DOI: 10.1109/34.754617 (ver p. 6).

[26] D. Kingma e J. Ba. "Adam: A Method for Stochastic Optimization". Em: *International Conference on Learning Representations* (dez. de 2014) (ver p. 63).

[27] A. Krizhevsky, I. Sutskever e G. E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. URL: http://code.google.com/p/cuda-convnet/ (ver pp. 9, 17).

[28] Y. Li, C. Xia e J. Lee. "Detection of Small-sized Insect Pest in Greenhouses Based on Multifractal Analysis". Em: *Optik - International Journal for Light and Electron Optics* 126 (jun. de 2015). DOI: 10.1016/j.ijleo.2015.05.096 (ver p. 6).

[29] T.-Y. Lin et al. "Focal Loss for Dense Object Detection". Em: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PP (jul. de 2018), pp. 1–1. DOI: 10.1109/TPAMI.2018.2858826 (ver pp. 16, 17).

[30] W. Liu et al. "SSD: Single Shot MultiBox Detector". Em: (dez. de 2015). DOI: 10.1007/978-3-319-46448-0_2. URL: http://arxiv.org/abs/1512.02325%20http://dx.doi.org/10.1007/978-3-319-46448-0_2 (ver p. 15).

[31] S. Lloyd. "Least square quantization in PCM". Em: *IEEE Transactions on Information Theory - TIT* 28 (jan. de 1982) (ver p. 51).

[32] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (ver p. ii).

[33] D. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". Em: *International Journal of Computer Vision* 60 (nov. de 2004), pp. 91–110. DOI: 10.1023/B%3AVISI.0000029664.99615.94 (ver p. 6).

[34] N. MacLeod, M. Benfield e P. Culverhouse. "Time to automate identification". Em: *Nature* 467 (set. de 2010), pp. 154–5. DOI: 10.1038/467154a (ver p. 5).

[35] W. McCulloch e W. Pitts. "A Logical Calculus of the Idea Immanent in Nervous Activity". Em: *Bulletin of Mathematical Biology* 5 (dez. de 1943), pp. 115–133. DOI: 10.1007/BF02478259 (ver p. 6).

[36] W. McKinney et al. "Data structures for statistical computing in python". Em: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX. 2010, pp. 51–56 (ver p. 58).

[37] J. Muangprathub et al. "IoT and agriculture data analysis for smart farm". Em: *Computers and Electronics in Agriculture* 156 (jan. de 2019), pp. 467–474. DOI: 10.1016/j.compag.2018.12.011 (ver pp. 2, 5).

[38] W. Niblack. *An Introduction to Digital Image Processing*. Delaware Symposia on Language Studies5. Prentice-Hall International, 1986. ISBN: 9780134806747. URL: https://books.google.pt/books?id=XOxRAAAAMAAJ (ver p. 41).

[39] *Novas pragas e doenças das plantas em Portugal*. URL: https://florestas.pt/conhecer/novas-pragas-e-doencas-das-plantas-em-portugal/ (acedido em 29/01/2022) (ver p. 1).

[40] N. Otsu. "A Threshold Selection Method from Gray-Level Histograms". Em: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (1979), pp. 62–66. DOI: 10.1109/TSMC.1979.4310076 (ver p. 41).

[41] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". Em: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (ver pp. 36, 55).

[42] J. Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". Em: (jun. de 2015) (ver pp. 13, 14, 63).

[43] F. Ren, W. Liu e G. Wu. "Feature reuse residual networks for insect pest recognition". Em: *IEEE Access* 7 (2019), pp. 122758–122768. ISSN: 21693536. DOI: 10.1109/ACCESS.2019.2938194 (ver p. 29).

[44] S. Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". Em: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (jun. de 2015). DOI: 10.1109/TPAMI.2016.2577031 (ver pp. 12, 13).

[45] C. Rother, V. Kolmogorov e A. Blake. *"GrabCut-Interactive Foreground Extraction using Iterated Graph Cuts* (ver p. 25).

[46] S. Ruder. "An overview of gradient descent optimization algorithms". Em: *arXiv preprint arXiv:1609.04747* (2016) (ver p. 63).

[47] J. Sauvola e M. Pietikäinen. "Adaptive document image binarization". Em: *Pattern Recognition* 33 (2 2000), pp. 225–236. ISSN: 0031-3203. DOI: https://doi.org/10.1016/S0031-3203(99)00055-2. URL: https://www.sciencedirect.com/science/article/pii/S0031320399000552 (ver p. 42).

[48] K. Simonyan e A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". Em: (set. de 2014). URL: http://arxiv.org/abs/1409.1556 (ver pp. 14, 63).

[49] L. Solis-Sánchez et al. "Scale invariant feature approach for insect monitoring". Em: *Computers and Electronics in Agriculture* 75 (jan. de 2011), pp. 92–99. DOI: 10.1016/j.compag.2010.10.001 (ver p. 6).

[50] J. Suykens e J. Vandewalle. "Least Squares Support Vector Machine Classifiers". Em: *Neural Processing Letters* 9 (jun. de 1999), pp. 293–300. DOI: 10.1023/A:1018628609742 (ver p. 6).

[51] C. Szegedy et al. "Going Deeper with Convolutions". Em: (set. de 2014). URL: http://arxiv.org/abs/1409.4842 (ver pp. 18, 19, 63).

[52] A. Thorat, S. Kumari e N. D. Valakunde. "An IoT based smart solution for leaf disease detection". Em: *2017 International Conference on Big Data, IoT and Data Science (BID)*. 2017, pp. 193–198. DOI: 10.1109/BID.2017.8336597 (ver pp. 2, 5).

[53] T. Tieleman, G. Hinton et al. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". Em: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31 (ver p. 63).

[54] P. Viola e M. Jones. "Rapid object detection using a boosted cascade of simple features". Em: *Comput. Vis. Pattern Recog* 1 (jan. de 2001) (ver p. 6).

[55] P. Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". Em: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2 (ver p. 57).

[56] J. Wang et al. "A new automatic identification system of insect images at the order level". Em: *Knowledge-Based Systems* 33 (set. de 2012), pp. 102–110. DOI: 10.1016/j.knosys.2012.03.014 (ver pp. 6, 23, 26).

[57] C. Wen, D. Guyer e W. Li. "Local feature-based identification and classification for orchard insects". Em: *Biosystems Engineering* 104 (nov. de 2009), pp. 299–307. DOI: 10.1016/j.biosystemseng.2009.07.002 (ver p. 6).

[58] X. Wu et al. *IP102: A Large-Scale Benchmark Dataset for Insect Pest Recognition*. URL: https://github.com/ (ver p. 26).

[59] C. Xia et al. "Automatic identification and counting of small size pests in greenhouse conditions with low computational cost". Em: *Ecological Informatics* 29 (2015). Special Issue on "Ecosystem Assessment and Management: Selected papers presented at the International Conference on Ecological Informatics 2013, 27-28 June 2013, Seoul, Korea", pp. 139–146. ISSN: 1574-9541. DOI: https://doi.org/10.1016/j.ecoinf.2014.09.006. URL: https://www.sciencedirect.com/science/article/pii/S1574954114001228 (ver p. 6).

[60] C. Xie et al. "Automatic classification for field crop insects via multiple-task sparse representation and multiple-kernel learning". Em: *Computers and Electronics in Agriculture* 119 (nov. de 2015), pp. 123–132. DOI: 10.1016/j.compag.2015.10.015 (ver pp. 23, 26).