Aprendizado Profundo Aplicado no Monitoramento de Insetos em Culturas de Citros

Felipe Vieira dos Santos - 39316

Dissertação apresentada à Escola Superior de Tecnologia e de Gestão de Bragança para obtenção do Grau de Mestre em Engenharia Industrial no âmbito do duplo diploma com à Universidade Tecnológica Federal do Paraná.

Trabalho orientado por:

Prof. Paulo Leitão

Prof. Frederic Conrad Janzen

Prof. José Alberto Cardoso Pereira

Bragança

Fevereiro - 2020

Aprendizado Profundo Aplicado no Monitoramento de Insetos em Culturas de Citros

Felipe Vieira dos Santos - 39316

Dissertação apresentada à Escola Superior de Tecnologia e de Gestão de Bragança para obtenção do Grau de Mestre em Engenharia Industrial no âmbito do duplo diploma com à Universidade Tecnológica Federal do Paraná.

Trabalho orientado por:

Prof. Paulo Leitão

Prof. Frederic Conrad Janzen

Prof. José Alberto Cardoso Pereira

Bragança

Fevereiro - 2020

Agradecimentos

Primeiramente, gostaria de demonstrar meu agradecimento ao o meu orientador Prof. Paulo Leitão, pela orientação, dedicação, incentivo ao longo do desenvolvimento deste trabalho.

Ao Prof. José Alberto Cardoso Pereira, o qual exerceu a função de coorientador, agradeço a ele pelo apoio, disponibilidade e a pronta ajudar com o desempenhar das tarefas relacionadas aos insetos e a disponibilização de informações a respeita das pragas. Ao Prof. Luis Fernando Piardi, como colaborador, pela orientação, incentivo, conselhos e apoio dados para o pleno desenvolvimento da redação desta dissertação. Necessito mostrar também minha gratidão ao Prof. Frederic Conrad Janzen por ter aceito convite e se tornado o meu coorientador à distância no âmbito do programa de Dupla Diplomação. Ao Luis Pereira, pela precisa colaboração com aquisição das fotografias e com as orientação sobre os insetos de interesse. Demonstro minha gratidão à Nátali Bahena Benck, pelo incentivo e apoio prestados, assim como na ajuda com a redação desta dissertação.

Um agradecimento especial à Universidade Tecnológica Federal do Paraná (UTFPR) e ao Instituto Politécnico de Bragança (IPB), os quais viabilizaram e possibilitaram a realização deste mestrado no âmbito do programa de Dupla Diplomação.

Por último, mas não menos importante, gostaria de expressar minha gratidão à minha família pelo incentivo constante e sempre presente em minha trajetória, disponibilizando constantemente preocupação e apoio internacional.

Resumo

Recentemente, detectou-se em Portugal, o inseto *Trioza erytrea*, o qual tem a possibilidade de transmitir a doença *Huanglongbing* (*Citrus Greening*) prejudicial à cultura dos citrinos. Esta patologia ainda não possui cura e tem alto potencial lesivo ao cultivo. Embora a doença não se manifestou-se em território português, métodos preventivos, baseados no monitoramento da quantidade de *Triozas*, são indispensáveis para efetivas estratégias de contingência, assim como antecipar uma possível disseminação. O método tradicional de monitoramento é realizado por intermédio de um profissional incumbido de identificar e contabilizar manualmente a *Trioza* e mais quatro outros insetos (indicadores da presença de *Trioza*), a partir de armadilhas autocolantes instaladas nas plantações de citros. Esse modo de execução consome tempo considerável, é suscetível a erros e demanda um profissional que nem sempre está disponível. Nesse sentido, o presente trabalho se propôs a desenvolver uma solução computacional, mediante a utilização de Redes Neurais Convolucional (RNC). Isso se deu por meio de uma aplicação, a qual desempenha a função de identificar, classificar e contar automaticamente os insetos citados. Realizou-se o estudo do desempenho de arquiteturas de RNC, as quais possuem a potencialidade de reconhecer insetos a partir de imagens. Como resultado, mostrou-se que a arquitetura SSD com *Inception*-v2 obteve melhor desempenho na identificação de *Trioza*. Porém, num primeiro momento, não se tem a viabilização da substituição do método tradicional pela aplicação, isso em decorrência das deficiências nos processos de aprendizagem.

Palavras-chave: Aprendizagem Profunda, Redes Neurais Convolucionais, Citros, Processamento de Imagens, Trioza erytre, Faster R-CNN, SSD, Inception v2, ResNet-50.

Abstract

Recently, the insect *Trioza erytrea* was detected in Portugal, which has the possibility of transmitting the disease *Huanglongbing* (Citrus Greening) harmful to the citrus culture. This pathology has no cure yet and has a high potential for injury to crops. Although the disease did not manifest itself in Portuguese territory, preventive methods, based on monitoring the amount of *Triozas*, are indispensable for effective contingency strategies, as well as anticipating a possible spread. The traditional method of monitoring is carried out by means of a professional responsible for manually identifying and accounting for *Trioza* and four other insects (indicators of the presence of *Trioza*), using self-adhesive traps installed in citrus plantations. This mode of execution consumes considerable time, is susceptible to errors and demands a professional who is not always available. In this sense, the present work proposed to develop a computational solution, through the use of Convolutional Neural Networks (CNN). This was done through an application, which performs the function of automatically identifying, classifying and counting the insects mentioned. The study of the performance of CNN architectures was carried out, which have the potential to recognize insects from images. As a result, it was shown that the SSD architecture with Inception-v2 achieved better performance in identifying *Trioza*. However, at first, there is no possibility of substituting the traditional method for application, due to deficiencies in the learning processes.

Keywords: Deep learning, Convolutional Neural Network; Citrus, Image processing, Trioza erytre, Faster R-CNN, SSD, Inception-v2, ResNet-50.

Conteúdo

Lista de Tabelas

Lista de Figuras

Siglas

API *Application Programming Interface.*

BDA Banco de Dados de Aprendizagem.

GDE Gradiente Descendente Estocástica.

GSM *Global System for Mobile Communications.*

GUI *Graphical User Interface.*

IPB Instituto Politécnico de Bragança.

PMC Perceptron de Múltiplas Camadas.

R-CNN *Region - Convolution Neural Network.*

ReLU *Rectified Linear Unit.*

ResNet *Residual Neural Network.*

RNC Rede Neural Convolucional.

RNCP Rede Neural Convolucional Profunda.

RoIP *Region of Interest Pooling.*

RRP Rede de Regiões Propostas.

SSD *Single Shot multiBox Detector*.

UTFPR Universidade Tecnológica Federal do Paraná.

YOLO *You Only Look Once*.

Capítulo 1

Introdução

Neste capítulo será apresentado as bases necessárias para o entendimento do trabalho.

Inicialmente, será realizada a descrição do contexto em que o trabalho se insere e a demonstração dos principais problemas decorrentes do processo tradicional de monitoramento de pragas, destacando sua importância (Seção1.1). Posteriormente, será apresentado os objetivos pretendidos com esta dissertação (Seção 1.2). E, por fim, será destacado as contribuições produzidas pelo desenvolvimento deste trabalho (Seção 1.3) e a estrutura do documento (Seção 1.4).

1.1 Enquadramento

É fato notório que o mundo está em constante processo de transformações industriais, tendo seu início no século XVII. Esse movimento traz vários benefícios, pode-se citar: o aumento de produção, diminuição de desperdícios, diminuição de gastos, de entre tantos outros decorrentes do processo de produção.

Atualmente, os processos industriais encontram-se na denominada Industria 4.0, esta pode ser definida como a integração de várias tecnologias no processo produtivo, por exemplo, inovação disruptiva, processamento de dados, *Big Data Analytics*, *Cyber Physical System*, serviços em nuvem, impressão 3D, segurança cibernética, robôs autônomos,

Internet das Coisas, aprendizagem profunda, simulações, entre outros [1]. Estes avanços possuem como propósito, proporcionar inúmeras vantagem e oportunidades. Como agregar valor, aumentar produtividade e diminuição de desperdícios [1].

Mas não é apenas a indústria, com toda a sua importância, que pode beneficiar com as tecnologias desenvolvidas na Industria 4.0. O monitoramento de pragas em plantações também deve ser beneficiado devido à sua alta relevância e importância para o gerenciamento de uma plantação, realizando o auxílio para utilização de estratégias necessárias para a tomada de decisão quanto ao controle de doenças.

Nesse contexto, a utilização de tecnologias de aprendizagem profunda, *Big Data Analytics* e Internet das Coisas da Indústria 4.0 podem permitir a automatização do processo de monitoramento de pragas com maior velocidade e precisão. Estes são responsáveis também pela aplicação da solução em diferentes sítios e ainda auxiliam produtores rurais com maior dinamismo e eficiência no combate de pragas ao indicar medidas correlativas [2]. Como consequência, acaba por tornar mais barato o processo se o compararmos com o método tradicional, que precisa de um especialista em insetos.

Utiliza-se, para esta finalidade, as Redes Neurais Convolucionais Profunda (RNCPs) capazes de reconhecer objetos a partir da extração de características de imagens. Esse processo de aprendizagem consiste no treinamento de arquiteturas de RNCP (*Faster* R-CNN e SSD com *Inception*-v2 e *ResNet*-50), mediante processamento de uma grande amostra de exemplos de insetos já classificados.

Neste momento, está sendo realizado o monitoramento de cinco espécies de insetos, *Trioza erytreae*, *Syrphidae*, *Coccinellidae*, *Chrysopidae* e *Philaenus* em plantações de citrinos (laranja, limão, tangerina), localizada em Vairão, uma região nas proximidades da cidade de Porto. Este tem por objetivo a investigação de uma possível dispersão da doença *Huanglongbing* (*Citros Greening*). Atuamente esta doença não possui cura e mediante contato com a planta, causa, inicialmente, a produção de frutos amargos, rijos e incomestíveis, sendo que com o agravamento do quadro, resulta a morte da planta [2]. Embora a *Trioza erytreae* seja o único inseto vetor da possível doença, os outros quatro insetos são indicadores da presença da *Trioza erytreae*.

1.2 Objetivos

O objetivo deste trabalho consiste realizar um estudo e assim desenvolver uma solução automática de monitoramento de pragas em plantações. Isso será concebível através de uma ferramenta capaz de identificar, classificar e contabilizar os insetos a partir de fotografias de armadilhas autocolantes dispostas em plantações de citros.

O seu desenvolvimento se dará mediante tecnologias de aquisição de imagem, aprendizagem profunda (*deep learning*) e de análise de dados, os quais serão adquiridos, processados e guardados em um banco de dados de monitoramento.

Os objetivos específicos para o desenvolvimento deste trabalho são:

- Criação de Banco de Dados de Aprendizagems (BDAs) (treino e teste), os quais serão concebidos a partir de fotografias de insetos de interesse que deverão ser adquiridas com a utilização de diferentes estratégias de caracterização.

- Treinar as atuais arquiteturas de RNCP com diferentes configurações e BDAs com o intuito de comparar as que apresenta melhores resultados em classificação e identificação de insetos, juntamente com um estudo de desempenho.

- Desenvolver uma aplicação gráfica que execute as RNCPs treinada para identificar, classificar e contabilizar os insetos de interesse e armazenar os dados em um banco de dados de monitoramento. Esta aplicação gráfica tem o intuito de ser intuitiva para que agricultores possam utilizá-las e assim obterem acesso às informações de imagens retiradas do seu próprio cultivo.

1.3 Contribuições

Hoje em dia existem diversos meios de se identificar e classificar insetos, sendo possível a utilização de várias estratégias para este fim, desde um profissional com a habilidade de identificar a classificá-los, até soluções de visão computacionais. Entretanto, ao se fazer a avaliação dos métodos de visão computacional que utilizam RNCPs, é possível constatar que podem ser melhorados.

Pensando nisso, o trabalho em questão apresenta como proposta a utilização de algumas alternativas passíveis de serem implementadas com o intuito realizar contribuições ao processo de automatização do monitoramento de pragas em plantações, proporcionando melhorias e atualizações.

Um exemplo disso é referente ao processo de aquisição e criação de BDA, que por diversas vezes são criados a partir de imagens retiradas da internet, as quais, em muitos caso, não representam a realidade dos insetos nas armadilhas [3]–[5]. Nesse trabalho, utilizou-se fotografias reais dos insetos nas armadilhas da plantação de citros. Outro ponto que também será aperfeiçoado diz respeito às arquiteturas das RNCPs, as quais, muitas vezes encontram-se defasadas e virtude do surgimento de arquiteturas mais modernas. Isso se dará com uso das *Faster Region - Convolution Neural Network* (R-CNN) [6] e *Single Shot multiBox Detector* (SSD) [7], que possuem recursos mais rápidos e eficientes quando comparados com as *AlexNet* ou VGG16 [8]. A escolha pelo uso da aprendizagem por transferência (*transfer learning*) representam uma ótimo opção para treinamento RNCPs, assim, diminui a carga computacional necessária e ocasiona diminuição dos custos do sistema de treinamento.

Uma aplicação, denominada por *Insect Detection*, traz como propósito ser intuitiva para que agricultores possam utilizá-la, e desta maneira, possuírem acesso a informações de imagens retiradas do seu próprio cultivo.

Outra contribuição deste trabalho refere-se ao estudo de desempenho do treinamento de diferentes arquiteturas de RNCPs, usadas para identificar e classificar os insetos agentes da praga de citros.

1.4 Estrutura do Documento

Após o este primeiro capitulo, a presente dissertação será dividida em mais 5 outros capítulos.

O Capítulo 2 é dedicado à análise do estado da arte da automatização do monitoramento de pragas, assim como a fundamentação teórica das tecnologias que serão usadas no decorrer desse estudo. O Capítulo 3 é destinado à apresentação dos métodos e ferramentas adotadas para o desenvolvimento dessa dissertação. O Capítulo 4 representa a trajetória de desenvolvimento deste projeto, tendo seu maior foco nas etapa de criação dos BDAs e treinamento das RNCPs.

Já na fase final desta estrutura, será apresentado no Capítulo 5 os resultados e as respectivas análises do desenvolvimento e da solução. Por fim, o Capítulo 6 apresenta as considerações finais e propostas destinadas aos trabalhos futuros.

Capítulo 2

Contexto e Tecnologias

Ao analisar uma atividade natural e corriqueira, como é a de olhar para algo e realizar rapidamente a sua identificação, não se percebe o quão complexa é essa atividade, a qual envolve inúmeros processos inconscientes e consciente do cérebro humano.

Neste capítulo, será apresentado o Estado da Arte, assim como realizada a introdução das bases fundamentais e conceitos que envolvem o processo de uma máquina conseguir ver e interpretar o que ela está vendo.

Primeiramente, será apresentado o estado da arte da automatização do monitoramento de pragas, mostrando como atualmente é realizado a identificação de objetos (Seção 2.1). Em seguida, será apresentado a fundamentação teórica. Essa análise terá como ponto de partida as explicações de conceitos mais gerais, como é o caso da visão computacional, da inteligência artificial e das redes neurais artificiais (Seções 2.2, 2.3 e 2.4).

A próxima abordagem será referente às tecnologias importantes para este trabalho, aprendizagem profunda (Sessão 2.5), bem como as Redes Neurais Convolucionais (RNCs) (Seção 2.6), apresentando uma ênfase maior nas diferentes meta-arquiteturas (Seção 2.8) e extratores de características (Seção 2.9).

2.1 Estado da Arte da Automatização do Monitoramento de Pragas

A agricultura desempenha um papel muito importante para o crescimento econômico [9]. E encontrar formas de melhorar a atividade agrícola tem gerado novas tecnologias, as quais passam a ter a capacidade de integrar vários sistemas de monitoramento e processamento de dados em tempo real. Portanto, levam a uma maior eficiência da produção, e diminuem o consumo de água e fertilizantes [10].

Tradicionalmente, o processo de identificação de insetos é realizado por um profissional de forma manual, o qual se revela suscetível a erros e demanda um profissional que nem sempre está disponível, conseguintemente não é uma tarefa fácil e consome tempo considerável de se executar [5]. Este deve possuir a técnica e habilidade de identificação, as quais são provenientes de estudos taxonômico e também adquiridos por experiencia [5]. Além disso, a demanda por profissionais capacitados para realizar este processo é muito maior que a oferta, ou seja, o recurso humano é escasso.

Nesse contexto, processos de monitoramento de pragas e identificação automática de insetos tem despertado o interesse de várias pesquisas [11]. Analisando de forma elementar, é possível afirmar que a identificação de insetos automática de insetos a partir de imagens se trata de identificação automática de objetos em imagens. Há mais de duas décadas que a identificação automática de objetos é um dos problemas mais fundamentais da visão computacional. Nos últimos, anos as RNCP têm desempenhado um papel essencial nessa tarefa [8].

O processo no qual uma RNCP consegue identificar insetos é decorrente de uma atividade de aprendizagem. Essa consiste em um treino com vários exemplos de insetos já identificados, os quais possuem a denominação de BDA.

A seleção manual de imagens e o uso de algoritmos de aumento de dados são técnicas de criação BDA e o uso delas influência consideravelmente a performance do classificador de objetos [5]. As coletas das imagens dos insetos para BDA podem ser proveniente diretamente das armadilhas [9], ou de banco dados disponíveis na internet [3]–[5], por

exemplo, *ImageNet*, *Google*, *Naver* e *FrashEye*. E em muitos trabalhos que desenvolveram classificadores e identificadores automáticos de insetos, foram utilizaram bancos de dados da internet para compor o BDA, as quais, em muitos caso, não representam a realidade dos insetos nas armadilhas[3]–[5]. Assim, podendo influenciar negativamente a performance do classificador.

Diferentes arquiteturas de RNCP foram utilizadas na tarefa de identificar insetos [3], [5], [9], como, por exemplo *AlexNet* [12]: uma rede com oito camadas profundas, e foi o primeiro modelo de RNCP a iniciar uma nova era de identificação de objeto; *VGG16* [13]: aumentou ainda mais a profundidade da rede para 16-19 camadas e uso de filtros convolucionais menores de 3x3 ao invés de 5x5 ou 7x7, as quais eram utilizadas anteriormente na *AlexNet*; *You Only Look Once* (YOLO) [14]: Foi o primeiro detector de um estágio, abandonaram completamente o paradigma de detecção "detecção de regiões proposta + verificação" que eram usando anteriormente [8], [12], [13].

Contudo, as arquiteturas de RNCPs estão sempre em constante evolução e surgindo novas com muito mais recursos e melhores performances e corrigindo os problemas. Que é o caso das arquiteturas *Faster* R-CNN e SSD, que ambas acrescentaram velocidade e precisão na identificação em objetos [6], [7]. Dessa forma, investigar o desempenho delas em tarefas de identificação de insetos é muito pertinente.

O uso das RNCP requer um alto poder computacional, o qual se revela ser um sério problema devido aos elevados custos e a alta demanda energética. Para contornar essa situação é possível utilizar a aprendizagem por transferência (*transfer learning*) [3]. Para isso, utiliza-se uma rede pré-treinada com um BDA anterior, a qual será reconfigurada e retreinada com os insetos. Isso faz com que apenas as camadas inicias e finais da RNCP sejam ajustadas. Gera-se assim, como consequência a diminuição acentuada da carga computacional e torna possível seu uso em computadores mais simples [3].

A integração de sistemas de monitoramento de pragas inclui tecnologias de sensores, comunicação em nuvem, dentre outras [9]. Sua principal função é aquisição, análise e publicação dos dados. Dessa forma, o sistema de análise de informação pode fazer uma previsão dinâmica das populações de pragas de acordo com os dados [9]. Outra

possibilidade é o fornecimento de métodos precisos de prevenção e tratamento [9].

A aquisição da imagem é um dos componentes mais importantes da identificação de objetos, ou neste caso, mais precisamente, de insetos. Ela é realizada mediante o uso de câmaras acopladas em armadilhas autocolantes, conforme é ilustrado na Figura 2.1a [9]. Os insetos ficam colados nessas armadilhas acima descritas, as quais são posicionadas em pontos da plantação, como mostrado na Figura 2.1b [9].

Após este procedimento de identificação e classificação dos insetos automaticamente, a análise dos dados é feita e os resultados de reconhecimento são enviados para a estação mediante o uso de um sistema de rádio [9].

(a) Equipamento para aquisição de imagens.

(b) Armadilhas autocolantes.

Figura 2.1: Sistema de monitoramento de pragas em plantações [9].

2.2 Visão Computacional

A visão computacional representa uma área de estudo interdisciplinar. Busca desenvolver sistemas computacionais de interpretação visual semelhante a visão humana. O principal objetivo pode ser definido sob dois pontos de visão: um biológico, o qual visa apresentar modelos computacionais do sistema visual humano. Da engenharia, o qual visa construir sistemas autónomos que possibilitem a execução de alguma tarefa que o sistema visual humano pode executar, sendo possível, até mesmo ser superior ao realizado por seres

humanos [15].

Notoriamente o campo da visão computacional é complexo, praticamente nenhum dos problema foi completamente resolvido. Isso se dá por diversas razões, sendo uma das principais o fato de o sistema visual humano ser simplesmente esplêndido para muitas tarefas, e assim sendo, a modelagem desses sistemas para um modelo computacional é extremante difícil [15].

O interesse nessa área teve início na década de 60. Roberts foi um dos primeiros a realizar trabalhos nesse sentido, os quais discutiram as possibilidade de extrair informações geométricas 3D a partir de fotografias em perspectiva 2D de poliedros [16]. Os trabalhos de Hubel e Weisel, também tiveram importância na construção desse conhecimento, repercutindo sobre sensibilidade e neurônios seletivos à orientação do córtex visual de um gato [17], [18].

As características e propriedades do sistema visual humano inspiram engenheiros a projetar e desenvolver sistemas de visão computacional em tarefas de recepção visual [15], isto inclui: reconhecimento e detecção de objetos, assim como entendimento de uma cena [19].

É importante observar também a fusão de vários campos relacionadas, como: processamento de imagens (tratamento de imagens "brutas" antes de análises adicionais), fotogramétrica (calibração de câmeras) e computação gráfica (geração de imagens computorizadas) [15].

Nos últimos anos, as RNCP tornaram-se padrão nas estruturas no campo da visão computacional, sendo que o número de camadas e complexidade são desenvolvidas para fazer com que as RNCP ofereçam precisão quase humana em tarefas, tais como classificação, detecção e segmentação [19].

O desenvolvimento do trabalho está intrinsecamente relacionado ao campo da visão computacional, pois busca extrair informações a partir de imagens. Principalmente, ao se relacionar com ponto de vista da engenharia, por intermédio da construção de sistemas autônomos para realizar alguma tarefa, neste caso, a tarefa de reconhecimento de insetos.

2.3 Inteligência Artificial

Por milhares de anos, o homem tenta entender como o ser humano pensa, entende, prevê e manipula o mundo ao seu redor. O campo da inteligência artificial vai ainda mais longe: ela tenta não somente entender, mas também construir entidades inteligentes [20].

Os trabalhos no campo da inteligência começaram logo após a segunda Guerra Mundial, principalmente após o surgimentos do termo inteligência artificial em 1956 pelo John McCarthy [20], o qual é conhecido como o pai desse sistema [21].

A inteligência artificial é ainda uma disciplina jovem em sua estrutura, interesse e métodos, não estão tão bem definidas quando comparadas com as disciplinas mais clássicas. Por esta razão acabam se tornando algo mais amplo, englobando vários outros campos.

Vários pesquisadores e livros didáticos tentam achar uma definição apropriada, podendo a inteligência artificial ser definida como o ramo da ciência da computação que se ocupa da automação do comportamento inteligente. Além disso, num contexto de ser como qualquer outra ciência, isto é, um empreendimento humano, pode ser melhor definida como: a coleção de problemas e metodologias estudadas pelos pesquisadores da inteligência artificial [22].

O objetivo da inteligência artificial é desenvolver paradigmas ou algoritmos em máquinas para realizar tarefas cognitivas, para os quais humanos são melhores atualmente. Esse sistema deve ser capaz de fazer três coisas: armazenar conhecimento, aplicar o conhecimento armazenado para resolver problemas e adquirir novos conhecimentos através da experiencia [18].

Os componentes fundamentais mostrados na Figura 2.2, de um sistema de IA são:

- A **representação** a qual consiste no uso difundido de uma linguagem de estruturas simbólicas para representar tanto o conhecimento genérico sobre um problema de interesse, como o conhecimento específico sobre a solução do problema.

- O **raciocínio** que é a habilidade de resolver problemas, em tais situações, são utilizados processos probabilístico, permitindo o sistema lidar com incertezas [20].

- E por último a **aprendizagem**, subdividida em quatro elementos: o ambiente que fornece alguma informação para um elemento de aprendizagem para aperfeiçoar a **base de conhecimento**, e finalmente o elemento de desempenho utiliza a base de conhecimento para executar a sua tarefa [18], [23].

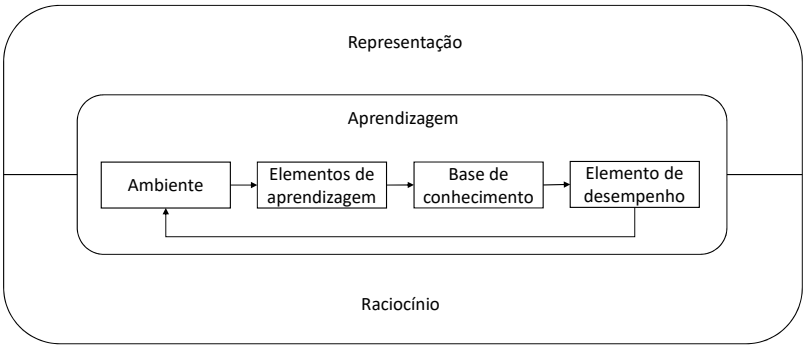


Figura 2.2: Componentes principais de um sistema de inteligência artificial [18].

O recente progresso das tecnologias científicas gera um crescimento acelerado no desenvolvimento de novas técnicas. Essas, por sua vez, levam o mundo a entrar rapidamente na nova era da inteligência artificial [19].

Atualmente, a inteligência artificial abrange uma grande variedade de subcampos, desde o geral, como aprendizagem e percepção, até os específicos, como é o caso de jogar xadrez, dirigir carros, diagnosticar doenças [20].

2.3.1 Aprendizagem de Máquina

As pessoas estão cercadas pelas tecnologias baseadas em aprendizagem de máquina, tais como: mecanismos de pesquisa, software *Anti-Spam* de *e-mail*, sistemas de proteção de detecção de fraudes de cartão de crédito, carros modernos que são equipados com sistemas de prevenção de acidentes.

Nas últimas décadas, a aprendizagem de máquina tornou-se uma ferramenta muito comum em quase todas as tarefas que exigem extração de informações de banco de dados. Constitui também, uma área muito fértil de pesquisa, produzindo uma série de problemas e algoritmos para a sua solução [22].

As ferramentas de aprendizado de máquina preocupam-se em conceder aos programas a capacidade de "aprender" e de se adaptar [24]. Podemos classificar aprendizagem tendo como base os seus algoritmos ou paradigmas, quais sejam: aprendizagem supervisionada, aprendizagem são supervisionadas e aprendizagem por reforço.

Outros tipos de aprendizagem existem, mas a abordagem será realizada, especificamente, nesses três tipos:

- **Aprendizagem supervisionada:** É utilizada quando se pretende classificar objetos em imagens usando várias imagens já classificadas para aprender o que são os objetos. Na aprendizagem supervisionada os agentes observam alguns exemplos de entrada-saída e aprendem a partir de uma função que mapeia o comportamento da entrada com a saída. [20].

- **Aprendizagem não-supervisionada:** O uso mais comum desse tipo de aprendizagem é utilizado em detecção de agrupamentos, os quais são potencialmente úteis de exemplos de entrada. No aprendizado não-supervisionada, o agente aprende padrões na entrada sem necessitar de nenhuma realimentação explicito seja fornecido [20].

- **Aprendizagem por reforço:** Na aprendizagem supervisionada, um "professor" usa exemplos para treinar um aprendiz. Já no aprendizado por reforço o próprio agente de aprendizagem, por tentativa, erro e realimentação, aprende uma técnica ótima para alcançar seus objetivos [22].

A aprendizagem de máquina envolve dois tipos de processamento de informação: o indutivo, no qual padrões gerais e regras são determinados a partir dos dados brutos e da experiencia. E o dedutivo no qual são utilizadas regras gerais para determinar fatos específicos.

A aprendizagem baseada em similaridade utiliza a indução, por exemplo, enquanto uma prova de um teorema é uma dedução baseada em axiomas conhecidos e em outros teoremas existentes [18].

A ferramenta que será desenvolvida neste trabalho deve ter a habilidades de "aprender" a determinar o que é um inseto, para que assim, quando estiver realizando a tarefa de identificação e classificação, seja capaz de adaptar-se ou generalizar os insetos.

Este etapa de "aprender" será desenvolvida a partir de um processo instituído de uma grande amostra de exemplos de insetos já classificados e identificados, a qual tem o intuito de servir como a base de conhecimento de aprendizagem. Dessa forma, durante o processo de aprendizagem a RNCP mapeia o comportamento da entrada com a saída, extraindo informações. Consta-se, com isso, que representa um processo de aprendizagem supervisionada.

2.4 Redes Neurais Artificiais

O cérebro humano é um computador altamente complexo, não-linear e paralelo, o qual é capaz de processar informações de uma forma bem diferente do computador digital convencional. Essas características têm servido de inspiração para trabalhos com redes neurais artificiais ou somente redes neurais [18].

As redes neurais, também conhecidas como conexionismo, são sistemas paralelos compostos de estruturas simples de processamento (nodos) que calculam funções matemáticas (normalmente não-lineares) [25].

Cada unidade de processamento é interligada com grande número de conexões umas com as outras, e dispostas em uma ou mais camadas, geralmente unidirecionais. Na maioria dos modelos estas conexões estão associadas a pesos os quais armazenam o conhecimento e serve para ponderar a entrada recebida por cada nodos [25].

Segundo a definição de George Luger: "As redes neurais não aprendem adicionando representações à sua base de conhecimento; em vez disso, elas aprendem modificando a sua estrutura global, de modo a se adaptar às contingências no mundo que habitam" [22].

2.4.1 Neurônio Artificial

Partindo do pressuposto das aspirações biológicas, uma unidade de processamento das redes neurais artificiais é chamada de neurônio artificial, sendo que o seu funcionamento se assemelha muito com o neurônio biológico. O diagrama em blocos da Figura 2.3 mostra o modelo de um neurônio artificial descrito por McCulloch e Pitts [26].

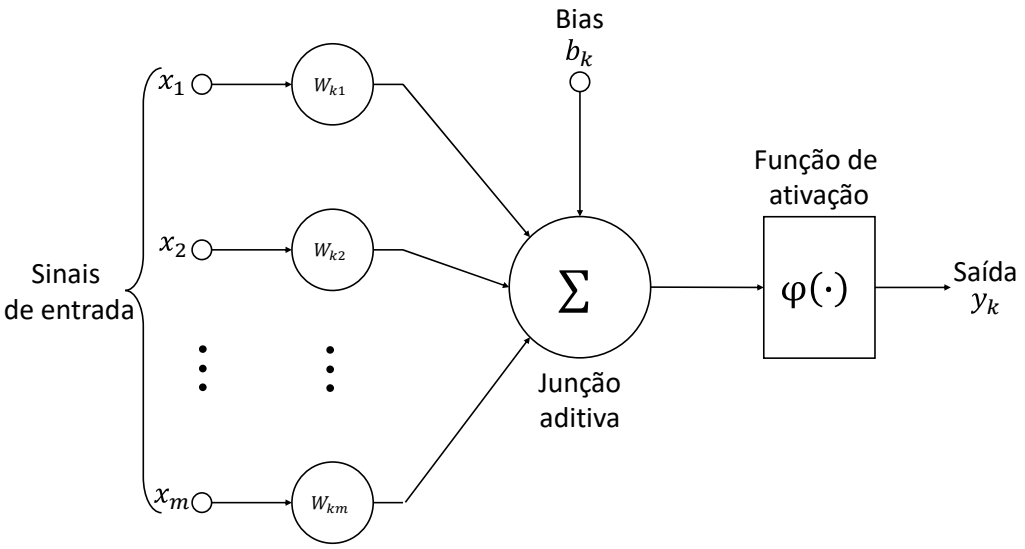


Figura 2.3: Modelo de neurônio artificial [18].

O modelo possui m terminais de entrada, x_1, x_2,..., x_m (representando os detritos) e um terminal de saída y_k (representando o axônio). Para simular o comportamento das sinapses, cada entrada é acompanhando por um peso (determinam "em que grau" o neurônio deve considerar sinais de disparo) acoplado w_{k1}, w_{k2},..., w_{km} cujos valores podem ser positivos ou negativos. sendo assim, o efeito de uma sinapse pós-sináptico é dado por $x_m w_{km}$ [18].

A junção aditiva do neurônio artificial tem a função de somar os valores dos sinais de entrada $x_m w_{km}$, as operações descritas aqui constituem um combinador linear.

Da mesma forma que o neurônio biológico possui um disparo a partir do limiar de excitação (*threshold*), a ativação do modelo é obtida através da aplicação de uma função de ativação $\varphi(\cdot)$, que ativa ou não a saída [18].

Por fim, é adicionado externamente um bias, b_k que tem o efeito de aumentar ou

diminuir a entrada líquida da função de ativação [18], [25].

Matematicamente o neurônio artificial k, pode ser representando pelo par de Equações a 2.1 e 2.2.

$$u_k = \sum_{j=1}^{m} w_{kj} x_1 \tag{2.1}$$

$$y_k = \varphi(u_k + b_k) \tag{2.2}$$

Dependendo do valor do b_k, a relação entre o campo local induzido ou potencial de ativação v_k do neurônio k e a saída do combinador linear u_k é modificada, como mostrado na Equação 2.3.

$$v_k \ = u_k + b_k \tag{2.3}$$

2.4.2 Função de Ativação

A função de ativação $\varphi(v_k)$ define a saída de um neurônio. Não sendo necessariamente zero ou um, podemos identificar quatro tipos básicos de função de ativação: função degrau (*Heaviside*), função sigmoidal, função rampa (*Rectified Linear Unit* (ReLU)) e *Softmax*.

Função degrau (*Heaviside*)

A função degrau é normalmente referida como Função *Heaviside*. Para esse tipo de função de ativação, descrito na Figura 2.4, podemos definir matematicamente pela Equação 2.4.

$$\varphi(v) = \begin{cases} 1 \ se \ v \ \geq \ 0 \\ 0 \ se \ v \ < \ 0 \end{cases} \tag{2.4}$$

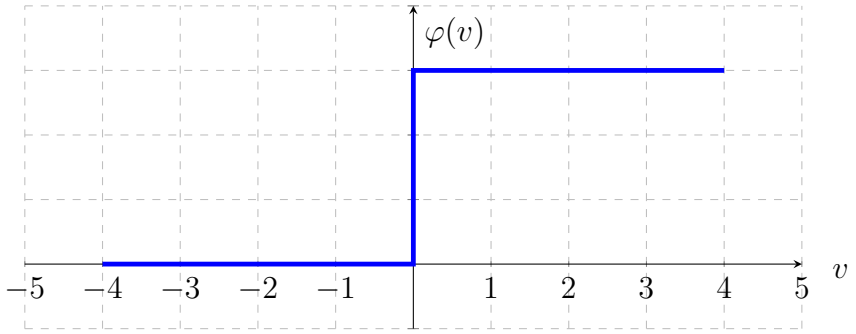


Figura 2.4: Função degrau (step).

Consequentemente, a saída de um neurônio k para esse tipo de função de ativação é expresso na Equação 2.5.

$$y_k = \begin{cases} 1 \; se \; \sum_{j=1}^{m} w_{kj}.x_j + b_k \; \geq \; 0 \\ 0 \; se \; \sum_{j=1}^{m} w_{kj}.x_j \; + b_k < \; 0 \end{cases} \tag{2.5}$$

A saída do neurônio assume o valor 1 se o campo local induzido daquele neurônio é não-negativo, e 0 caso for negativo. Essa definição descreve a propriedade tudo-ou-nada do modelo de McCulloch e Pitts [18], [26].

Função sigmoidal (S-*shape*)

A função sigmoidal, também conhecida como S-*shape*, descrita na Figura 2.5, é uma função de ativação semi-linear, limitada e monotônica.

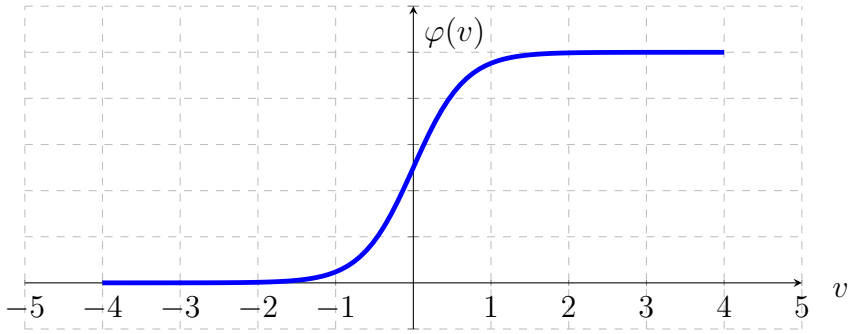


Figura 2.5: Função sigmoidal (S-*shape*).

É possível definir várias funções sigmoidais, umas das mais importantes é a função logística definida na Equação 2.6.

$$\varphi(v) = \frac{1}{1 + \ e^{-av}} \tag{2.6}$$

Onde a é o parâmetro de inclinação da função sigmóida, aumentando o valor de a a inclicação da curva aumenta. Enquanto a função degrau assume o valor de 0 ou 1, uma função sigmoide assume intervalos contínuos de valores entre 0 e 1 [18], [25].

Unidade Linear Retificada (ReLU)

Nas últimas décadas, as redes neurais usavam funções não-lineares suaves, como funções degrau e sigmóidal. Atualmente, a função não-linear mais popular é a unidade linear retificada (ReLU) [27].

A função de ativação ReLU é descrita na Figura 2.6 e é definida na Equação 2.7.

$$\varphi(v) = \begin{cases} 1 \ se \ v \ \geq \ 0 \\ 0 \ se \ v \ < \ 0 \end{cases} \quad \text{ou} \quad ReLu(v) = max\left\{0, v\right\} \tag{2.7}$$

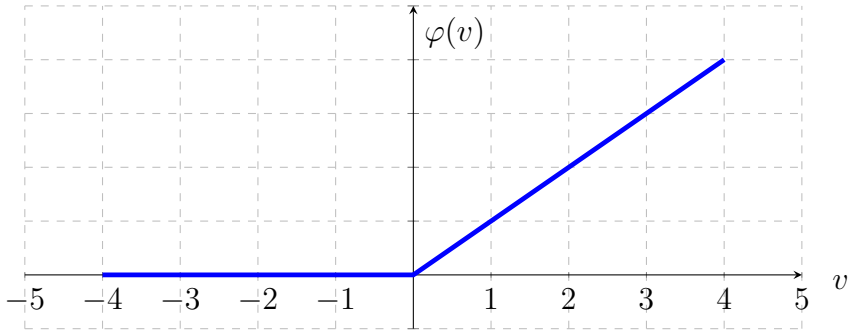


Figura 2.6: Função rampa (ReLU).

Esse tipo de função de ativação tem a capacidade de aprende mais rapidamente em redes com muitas camadas, permitindo o treinamento não supervisionado de uma rede neural profunda [27].

Softmax

Até então foram apresentadas funções de ativação que lidam melhor com classificação de no máximo duas classes. Quando desejamos lidar com problemas de classificação de múltiplas classes, podemos usar a função *softmax* [28].

O nome "*softmax*" pode ser um pouco confuso, consistindo na combinação de dois termos, sendom eles: a função *argmax* que tem como resultado um vetor único, o qual não é continua ou diferenciável. E o termo "*soft*" deriva do fato de que a função *softmax* é continua e diferenciável [28].

A função de ativação *softmax* pode ser definida na Equação 2.8.

$$\varphi(v) = softmax(v_i) = \frac{e^{v_i}}{\sum_{j=1}^{n} e^{v_j}} \; para \; i = 1, \; ..., n \tag{2.8}$$

As saídas do *softmax* sempre somam 1, portanto um aumento do valor de uma unidade corresponde necessariamente a uma diminuição no valor dos outros.

Sempre que desejamos representar uma distribuição de probabilidade entre n classes diferentes sobre uma variável discreta, podemos usar a função *softmax*. A Figura 2.7 mostra um exemplo no qual a função *softmax* transforma *logits* $[2.0, 1.0, 0.1]$ em um vetor de probabilidades $[0.7, 0.2, 0.1]$ [29].

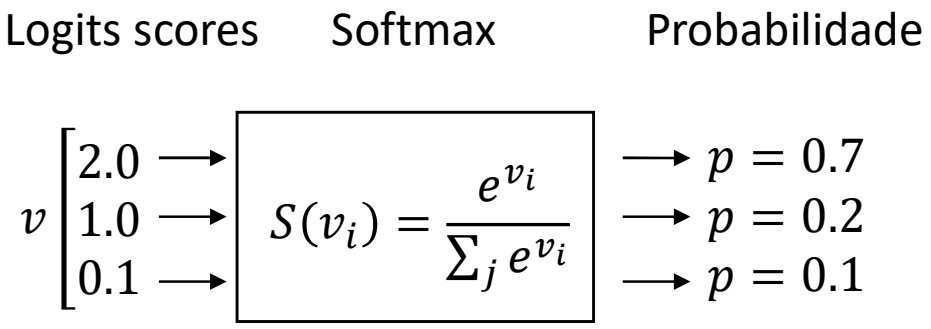


Figura 2.7: Saída da função *softmax* em termos de probabilidade [29].

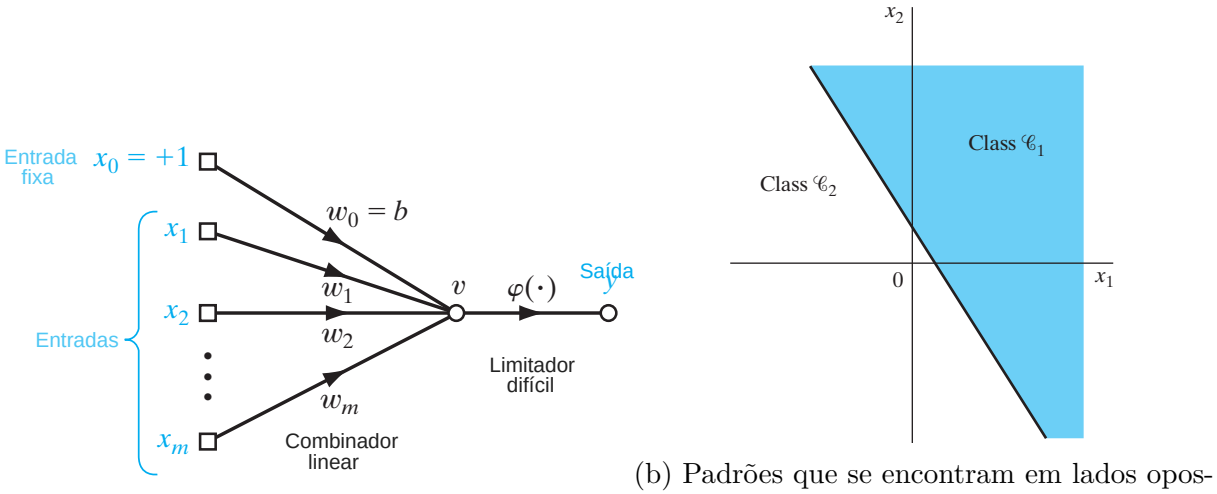
Do ponto de vista da neurociência, é interessante pensar na função *softmax* como uma maneira de criar uma forma de competição, entre as unidades que neles participam. Essa representação é análoga à inibição lateral que se acredita existir entre neurônios próximos

no córtex.

No extremo (quando a diferença de uma classe entre as outras é muto grande em magnitude), torna-se uma forma de "vencedor-leva-tudo"(na qual uma saída tem um valor muito próximo de 1 e as outras tendem a 0) [28].

2.4.3 Perceptron de Múltiplas Camadas

O perceptron é a forma mais simples de uma rede neural, a Figura 2.8a exemplifica o grafo perceptron. Consiste em um único neurônio com pesos sinápticos ajustáveis e bias, sua utilização é limitada para classificar padrões linearmente separáveis [18].



(a) Perceptron.

(b) Padrões que se encontram em lados opostos de um hiperplano.

Figura 2.8: Perceptron e superfície de decisão de hiperplano [18].

Os padrões (vetores) usados para treinar os perceptrons são retirados de duas classes, então o algoritmo do perceptron converge e posiciona a superfície de decisão em um hiperplano entre classes distintas, exemplificado na Figura 2.8b. A prova de convergência do algoritmo é conhecida como teorema de convergência do perceptron [18].

Para solucionar problemas não linearmente separáveis, faz-se o uso de redes com uma ou mais camadas intermediarias ou escondidas, conhecida como Rede Perceptron de Múltiplas Camadas (PMC), podendo também ser chamada de rede **totalmente conectada**

(um neurônio em qualquer camada está conectado a todos os neurônios da camada anterior) [25].

A Figura 2.9, mostra um grafo de arquitetura de uma rede perceptron de múltiplas camada [18].

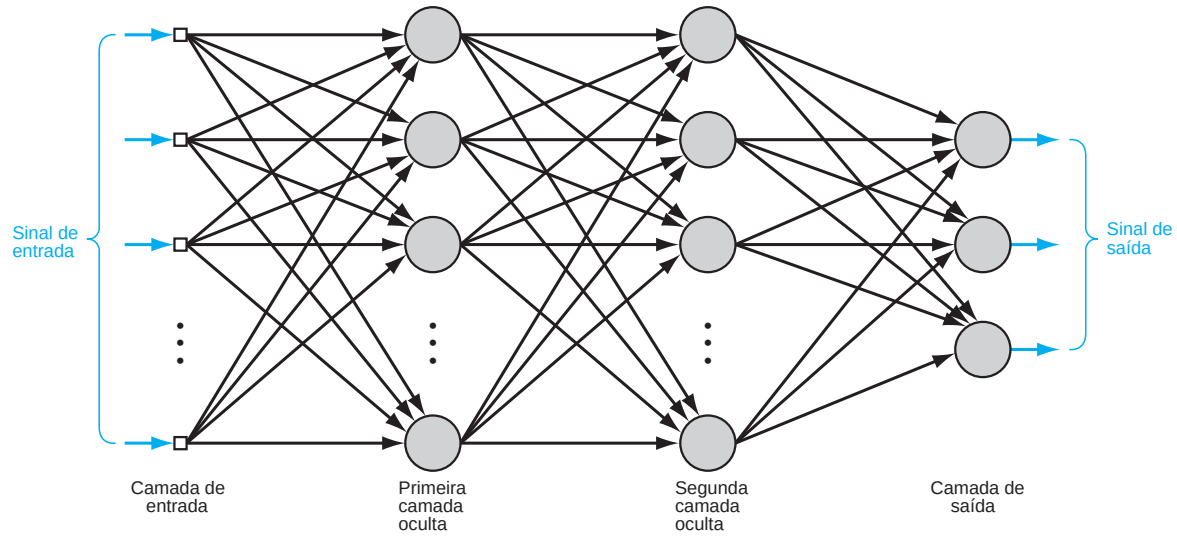


Figura 2.9: Grafo de arquitetura de uma Perceptron de Múltiplas Camadas [18].

Tipicamente, a rede PMC compõem de unidades sensoriais (neurônios) que constituem a **camada de entrada** (sinal de entrada se propaga para frente através da rede, camada por camada), uma ou mais **camadas ocultas** de neurônios computacionais e uma **camada de saída** de neurônios computacionais. Os neurônios ocultos capacitam a rede a aprender tarefas complexas, extraindo progressivamente as características mais significativas dos padrões (vetores) de entrada [18].

Retro-propagação de erro (*Back-propagation*)

As redes PMC tem sido amplamente usada para resolver problemas difíceis, através de um processo de treinamento supervisionado com um algoritmo muito popular, conhecido como **retro-propagação de erro (*back-propagation*)**. O desenvolvimento do algoritmo de retro-propagação de erro representa um marco nas redes neurais, pois fornece um método

computacional eficiente para o treinamento de PMC [18].

O algoritmo de retro-propagação de erro é baseado na regra delta, por isso pode ser chamado de regra delta de generalização. Esse algoritmo propõe uma forma de definir o erro dos neurônios das camadas ocultas, possibilitando o ajuste de seus pesos. Os ajustes dos pesos são realizados utilizando o método do gradiente [25].

A função de custo a ser minimizada é uma função de erro ou energia, representada pela Equação 2.9.

$$E = \frac{1}{2} \sum_{p} \sum_{i=1}^{k} (d_i^p - y_i^p)^2 \tag{2.9}$$

Onde E é medida de erro total, p é o número de padrões, k é o número de unidades de saída, d_i é a i-ésima saída desejada e y_i é a i-ésima saída gerada pela rede.
Essa Equação 2.9 defini o erro total cometido pela rede, ou a quantidade em que, para todos os padrões p de um dado conjunto, as saídas geradas pela rede diferem das saídas desejadas [25].

Considerando, que a minimização do erro para cada padrão individual leva à minimização do erro total, temos que o erro passa a ser definido como:

$$E = \frac{1}{2} \sum_{j=1}^{k} (d_j - y_i)^2 \tag{2.10}$$

A regra delta diz, que a variação dos pesos seja definida de acordo com o gradiente descendente do erro com relação ao peso, ou seja, a variação do peso de um dado padrão seja definida pela Equação 2.11 e que função de ativação (função semilineares) é definida pela Equação 2.12 [25].

$$\Delta w_{ji} \propto -\frac{\partial E}{\partial w_{ji}} \tag{2.11}$$

$$y_j^p = f_j(\text{net}_j^p) \tag{2.12}$$

onde

$$\text{net}_j = \sum_{i=1}^{n} w_{ji}.x_i \tag{2.13}$$

A constante n representa o número e conexões de entrada do neurônio j, e w_{ji}, o peso da conexão entre a entrada x_i e o neurônio j.

A fim de definir, como cada um dos pesos de cada neurônio da rede deve ser ajustado, na forma de diminuir o erro total, utilizando operações matemáticas (regra da cadeia), temos que:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial w_{ji}} \tag{2.14}$$

A segunda derivada é simples de calcular então a Equação 2.14 fica:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial \text{net}_j} x_i \tag{2.15}$$

A primeira derivada, da Equação 2.14, mede o erro do neurônio j, e geralmente é abreviada para δ_j.

$$\frac{\partial E}{\partial \text{net}_j} = \delta j = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial \text{net}_j} \tag{2.16}$$

A segunda derivada da Equação 2.16 é definia como:

$$\frac{\partial y_j}{\partial \text{net}_j} = \frac{\partial f'(\text{net}_j)}{\partial \text{net}_j} = f(\text{net}_j) \tag{2.17}$$

Se o neurônio estiver na última camada, o seu erro pode ser facilmente definido utilizando a Equação 2.10. Assim, a Equação 2.19, que é a mesma formula da regra delta original.

$$\frac{\partial E}{\partial y_j} = \frac{\partial (\frac{1}{2} \sum_{i=1}^{k} (d_i - y_i)^2)}{\partial y_j} = (d_j - y_j) \tag{2.18}$$

Substituindo os dois termos do lado direito da Equação 2.16, obtém-se a Equação 2.19.

$$\delta j = (d_j - y_i) f'(\text{net}_j) \tag{2.19}$$

Caso o neurônio j não seja um nodo de saída, a regra da cadeira é utilizada para escrever:

$$\frac{\partial E}{\partial y_j} = \sum_{l=1}^{M} \frac{\partial E}{\partial \text{net}_l} \frac{\partial \text{net}_l}{\partial y_j} = \sum_{l=1}^{M} \frac{\partial E}{\partial \text{net}_l} \frac{\partial \sum_{i=1}^{n} w_{il}.y_i}{\partial y_j} = \sum_{l=1}^{M} \frac{\partial E}{\partial \text{net}_l} w_{jl} \tag{2.20}$$

onde

$$\sum_{l=1}^{M} \frac{\partial E}{\partial \text{net}_l} w_{jl} = \sum_{l=1}^{M} \delta j w_{jl} \tag{2.21}$$

Substituindo mais uma vez os dois termos do lado direito da Equação 2.16, obtém-se que, para os neurônios das camadas intermediárias, o erro é definido por:

$$\delta j = f(\text{net}_j) \sum_{l} \partial_l w_{jl} \tag{2.22}$$

Pode-se assim generalizar a fórmula de ajuste de pesos proposta na Equação 2.11 para:

$$\Delta w_{ji} = \eta \delta_j x_i \tag{2.23}$$

ou

$$w_{ji} = (t+1) = w_{ji}(t) + \eta \delta_j(t) x_i(t) \tag{2.24}$$

Concluindo, a aprendizagem por retro-propagação de erro consiste em dois principais passos através das camadas da rede: um passo pra frente, a **propagação (*forward*)** e um passo pra trás, a **retro-propagação (*backward*)**. Na propagação, um padrão de atividade (vetor de entrada) é aplicado aos neurônios sensoriais da rede e seu efeito se propaga através da rede, sendo que os pesos sinápticos da rede são todos **fixos**. Durante a retro-propagação, os pesos sinápticos são **ajustados** de acordo com a regra de correção do erro [18].

Especificamente, a resposta real da rede é subtraída de uma resposta desejada para produzir um sinal de erro. Se o neurônio for de saída, o erro δ_j será definido pela Equação 2.19. Caso contrário, δ_j será dado pela Equação 2.22. Este sinal de erro é então propagado para trás através da rede, contra a direção das conexões sinápticas, derivando desta situação o nome "retro-propagação de erro" (*back-propagation*) [18], [25].

2.5 Aprendizagem Profunda

Aprendizagem representativa é um conjunto de métodos que permite automaticamente descobrir representações necessárias para a detecção ou classificação a partir de entrada de dados. Já a aprendizagem profunda (*deep learning*) consiste em métodos de aprendizagem representativa com múltiplas camadas de representação, os quais são obtidos através da composição simples, mas não lineares, que transformam a representação em um nível mais alto e um pouco mais abstrato [27].

Por exemplo, aborda-se nesta dissertação, em tarefas de classificação dos insetos *Trioza erytreae* em imagens, que as camadas superiores amplificam aspectos importantes (a estrutura característica das asas) e suprimem aspectos irrelevantes (a cor amarela da armadilha) para a classificação.

As características aprendidas na primeira camada representam tipicamente a presença ou ausência de bordas em determinadas orientações e localizações na imagem. A segunda camada tipicamente detecta os motivos através da localização de determinadas disposições de bordas, independentemente de pequenas variações nas bordas. A terceira camada pode montar padrões em combinações maiores que correspondem a partes de objetos familiares. Já as camadas subsequentes detectariam objetos como combinações dessas partes [27].

O aspecto chave da aprendizagem profunda é que estas camadas de características não são configuradas manualmente, elas são aprendidas a partir de dados da base de conhecimento, representando assim um procedimento de aprendizagem. A forma mais comum é a aprendizagem supervisionada. Para a construção de um classificador é necessário primeiro coletar um grande conjunto de exemplos, sendo cada um dos objetos etiquetados conforme sua categoria [27].

Durante o processo de treinamento de uma RNCP, é mostrada à máquina uma imagem e esta produz uma saída na forma de um vetor de pontuação, um para cada categoria. O objetivo é que cada categoria desejada tenha a pontuação mais alta de todas as categorias. Uma função mede o erro (ou distância), a máquina modifica seus parâmetros (pesos) internos a fim de reduzir esse erro. Em um sistema de aprendizagem profundo pode haver

milhões de pesos ajustáveis e centenas de milhões de exemplos [27].

2.6 Redes Neurais Convolucionais

Recentemente, as RNCs têm se destacado e alcançaram um grande desempenho em tarefas de reconhecimento de imagens [30]. Projetadas especialmente para reconhecer formas bidimensionais (imagens) com um alto grau de invariância quanto a translação, escala, inclinação dentre outras formas de distorções. Em outras palavras, de fazer generalizações com confiabilidade [18].

Sua arquitetura profunda combina várias operações distribuídas em camadas. Para melhor compreensão a Figura 2.10 mostra a planta arquitetural simplificada de uma RNC, a qual classifica insetos. Podemos separar sua estrutura em duas principais camadas, sendo elas, **Camada Convolutiva** e **Camada Totalmente Conectada**.

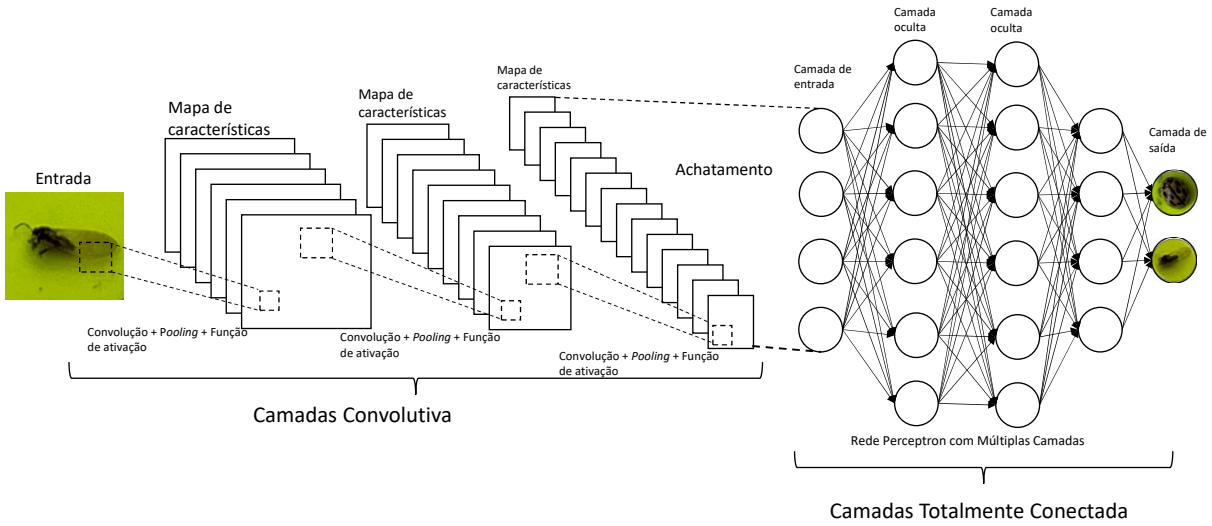


Figura 2.10: Arquiteta simplificada de uma RNC.

2.6.1 Camadas Convolutiva

A camada convolutiva é responsável por extrair características (*feature extractors*) de imagem e formas um mapa de características (*feature map*), proveniente de sucessivos processos/camadas de convolução, função de ativação e subamostragem (*pooling*) [18], [30].

As imagens digitais são representações bidimensionais (matrizes) e cada elemento da matriz representa a posição e intensidade luminosa dela.

As imagens coloridas do formato RGB, exemplificada Figura 2.11, na são representadas em três matrizes, onde cada matriz representa um canal de cor, vermelho (*red*), verde (*green*) e azul (*blue*). Então, as entradas (imagens) da camada convolutiva são representadas como tensores de ordem três, isto é $I(h,\ w,\ c)\ = (\ R_{h,\ w},\ G_{h,w}, B_{h,w})$ sendo h altura, w largura e c o canal de cor.

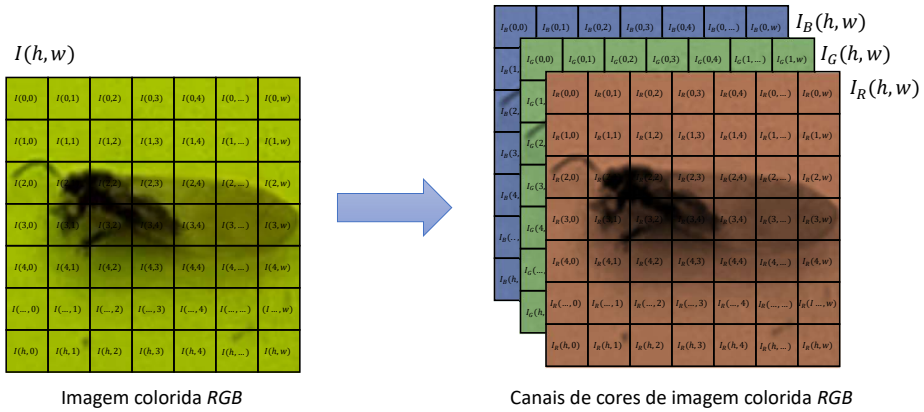


Figura 2.11: Imagem colorida no formato RGB.

Convolução

O termo "convolucional" inserido RNC indica que a rede emprega operações matemáticas chamadas de convolução. Esta por sua vez, é um tipo específico de operação linear [28]. A Figura 2.12 mostra o processo de convolução da camada convolutiva de uma maneira simplificada, que será explicado e exemplificado a seguir, para o melhor entendimento.

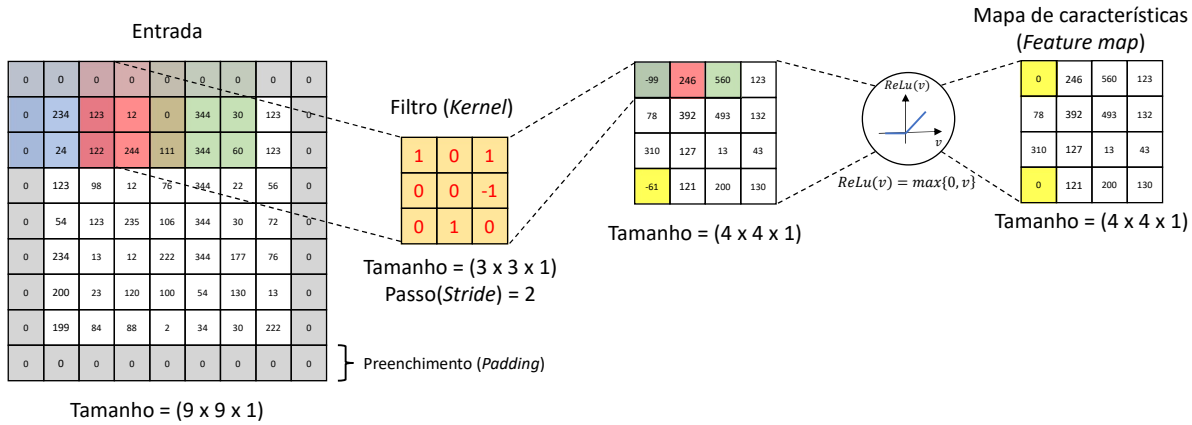


Figura 2.12: Processo simplificado de convolução da camada convolutiva.

Mas antes de propriamente ser realizado a convolução, é necessário realizar um preenchimento (*Padding*) nas bordas da imagem $I_{7,7,1}$, aumentando seu tamanho para $I_{9,9,1}$. Desta forma, o resultado da operação não irá reduzir significativamente o tamanho e não acontecerá um truncamento durante a varredura. O preenchimento utilizado será de zeros (*Zero-Padding*).

As operações serão realizadas em um tensor de ordem um. Existem várias formas diferentes de realizar o processo de convolução, dependendo da configuração que foi definida. Neste exemplo, é definido um filtro (*Kernel*) de tamanho (3 x 3 x 1), com um passo (*Stride*) igual a 2. A ideia é o filtro "varrer" toda a imagem em passos de dois em dois. Importante notar que o filtro funciona como um **campo receptivo**, extraindo características locais, uma vez que uma característica seja extraída, sua localização exata se torna menos importante desde que a sua posição em relação a outras características seja aproximadamente preservada [18].

Podemos definir matematicamente Equação 2.25 a convolução,

$$S(h,w) \;=\; (I * K)(h,w) = \sum_{i=s-1}^{m+s-1} \sum_{j=s-1}^{n+s-1} I(h+i, w+j) K(i+1-s, j+1-s) \qquad (2.25)$$

no qual $I_{h,w}$ representa a imagem de entrada, $K_{m,n}$ representa o filtro (*kernel*), s, o passo e $S_{h,w}$ é a saída da operação convolucional.

Como resultado de todas as convoluções da entrada com o filtro, teremos um tensor

do tamanho (4 x 4 x 1) Figura 2.12.

É necessário agora passar esses dados por uma função de ativação com o objetivo de trazer a não-linearidades ao sistema. Neste exemplo foi utilizado a função de ativação ReLU, essa função zera todos os valores negativos do tensor. Por fim, o tensor resultante é um mapa de características (*feature map*).

Importante destacar que, apesar de ter sido exemplificado a operação de um único filtro em um tensor de ordem um, a operação de convolução é feitar em tensores de ordem três, e são vários filtro realizando a convolução gerando assim vários mapas de características.

Subamostragem (*Pooling*)

Após o processo (camada) de convolução será realizada um processo de subamostragem(*pooling*), sendo ela feita por uma camada computacional que reduz a resolução do mapa de características. Esta operação tem como foco reduzir a sensibilidade da saída do mapa de características em relação ao deslocamento e as outras distorções [18].

Seu processo se assemelha com o processo de convolução. Do mesmo modo que na convolução existe um filtro com um passo que "varre" o tensor, realizando os cálculos e em alguns casos uma função de ativação, a subamostragem possui um filtro(*pool*), o qual realiza cálculos de media local (*averege-pooling*) ou de valor máximo local (*max-pooling*).

A Figura 2.13 mostra o processo de subamostragem (*pooling*) no mapa de características resultante da operação de convolução. O filtro usado foi de tamanho (2 x 2) e no passo 2, normalmente, os valores de passo de tamanho são igual para não acontecer sobreposição das regiões calculadas.

Sucessivas operações de convolução e subamostragem na camada convolutiva criam um efeito "bipiramidal". Isto é, em cada operação, o numero de mapas de características é aumentado, enquanto a resolução espacial é reduzida quando comparadas com a camada anterior [18].

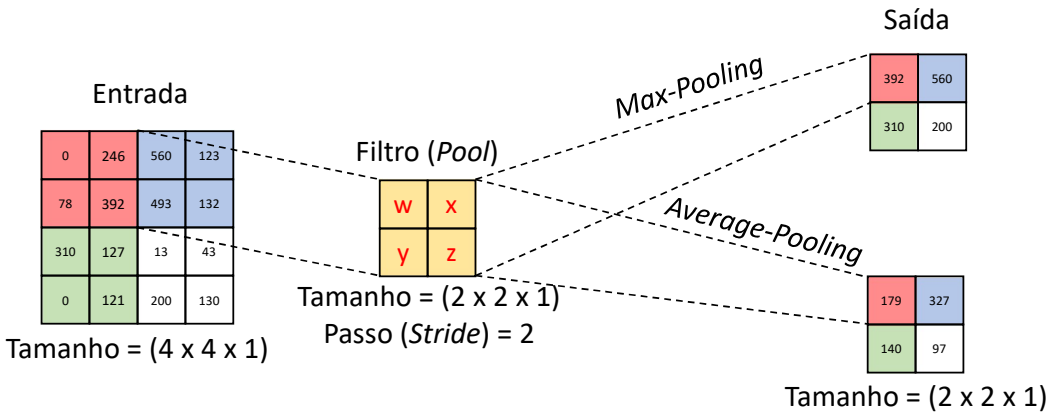


Figura 2.13: Processo de subamostragem simplificado.

2.6.2 Camada Totalmente Conectada

Os dados de saída da camada convolutiva passam por um processo de "achatamento", alterando a ordem do tensor para de apenas uma ordem. Assim está pronto para que os dados se tornem de entrada da camada totalmente conectada. Sendo esta a última grande camada [18], [30].

A camada totalmente conectada é uma rede PMC com a capacidade de realizar classificações [18], [30]. Então todas aquelas características que foram discutidas na (Seção 2.4.3), como camadas ocultas e algoritmos de retro-propagação de erro serão utilizadas nesta camada.

2.6.3 Treinamento de uma Rede Neural Convolucional

Estruturada toda a RNC, agora é preciso que ela consiga classificar objetos em imagens, mas é improvável que isso aconteça antes do treinamento. Isso se dá, devido ao fato que os milhares de parâmetros livres da rede que não estejam ajustando. Então o processo de treinamento de uma RNC é basicamente ajustar os parâmetros livres (**mapa de características, filtro, pesos**) a partir de algum algoritmo de aprendizagem.

Imaginemos a construção de um classificador de insetos. Primeiro, como se trata de uma aprendizagem supervisionada é preciso coletar um grande conjunto de dados de insetos. Cada um desses, por sua vez, deve ser rotulado conforme sua categoria, formando

assim, um grande conjunto de BDA. Tendo esse grande conjunto de BDA é então iniciado o processo de treinamento.

Durante o treinamento, a rede recebe uma imagem na entrada e produz uma saída na forma de um vetor de pontuação, uma para cada categoria. Essa pontuação é usada para medir uma função de erro ou perda (*loss function*), na qual mede a distância entre as pontuações de saída e o padrão desejado. Então o algoritmo de aprendizagem procura minimizar os erros obtidos pela a rede, modificando seus parâmetros ajustáveis [27].

Existem duas abordagens diferentes quando a frequência para os ajustes dos parâmetros, *on-line* e por lote (*batch*). Na online os parâmetros são ajustados após a apresentação de cada imagem do conjunto de treinamento. Em contra partida, por lote, os parâmetros são atualizados após todas as imagens do conjunto de treinamento terem sido apresentadas [25].

O algoritmo retro-propagação procura diminuir o erro, ajustando os pesos e bias, para que eles correspondam às coordenadas dos pontos mais baixos da superfície de erro. Para isso, ele utiliza o método do gradiente descendente. O gradiente de uma função é o caminho de uma curva na direção e sentido em que a função tem a taxa de variação máxima, exemplificada na Figura 2.14. Isso garante que a rede caminhe pela superfície na direção em que diminuir o erro [25].

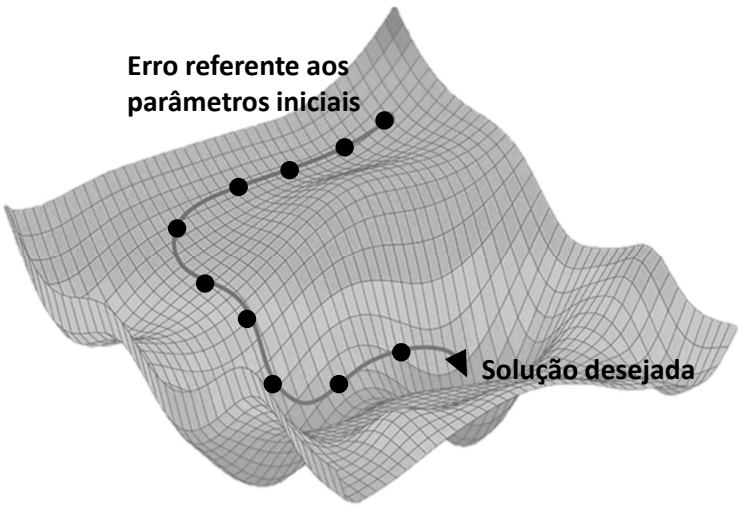


Figura 2.14: Superfície de erro [25], [31].

Na prática, utiliza-se o método do **Gradiente Descendente Estocástica (GDE)**. Isso consiste em mostrar o vetor de entrada para alguns exemplos, calcular o gradiente médio desses exemplos, e ajustar os pesos de acordo. O processo é repetido em vários pequenos conjuntos de treinamento (*mini-batch*) até que a média da função de erro diminua. Esse procedimento simples geralmente encontra valores bons de pesos em um curto espaço de tempo comparado com técnicas mais elaboradas [27].

Alguns problemas podem acontecer durante o processo de treinamento, sendo que um dos mais importantes é o *overfitting*. Esse ocorre quando, após um certo ciclo de treinamento, a rede memorizou os padrões de treinamento, gravando suas peculiaridades e ruídos.

Uma forma de evitá-lo é dividindo o conjunto de treinamento em conjunto de treino e conjunto de teste. O conjunto de treino é utilizado na modificação dos pesos e o conjunto de teste é utilizado para estimar a capacidade de generalização da rede durante o processo de aprendizagem [25].

Após vários ciclos (épocas), a tendência é a rede ter aprendido a classificar os insetos e é natural surgir a dúvida sobre em que momento se deve encerrar o treinamento.

Existem vários métodos ou critérios de paragem: pode ser encerrada após um número de ciclos definido; após um valor de erro estiver a baixo de um valor determinado ou encerrar quando a porcentagem de classificação correta estiver acima de uma constante, assim como outros que o usuário definir como critério [25].

2.7 Aprendizagem por Transferência

Em muitas RNCPs treinadas com imagens, acontece um fenômeno curioso e comum entre as redes: nas primeiras camadas da rede, aprende-se a extrair características semelhantes (bordas, curvas), mas não são especificadas para um conjunto ou tarefa em particular. Sendo que as características mais definidas são aprendidas apenas nas últimas camadas da rede [32].

Há duas estratégias principais de aprendizagem por transferência, o ajuste fino e transferência de características. No ajuste fino, uma RNCP pré-treinada é ligeiramente modificada e ajustada aos parâmetros para trabalhar com a tarefa de interesse, normalmente essa técnica é utilizada quando existe uma semelhança de tarefas.

Já a segunda estratégia é a transferência de características, envolve uma RNCP que tenha sido pré-treinada numa tarefa genérica de classificação de objetos e é exposta a novas imagens de interesse, sendo que as informação sobre a sua percepção dessas imagens é utilizada em uma nova rede, a qual realizará o treino de um sistema de identificação, alterando preferencialmente as camadas inicia e final da rede [3].

A ideia fundamental da aprendizagem por transferência é reaproveitar redes neurais já treinadas para alguma tarefa especifica e reajustar ela para realizar a tarefa desejada para economizar poder computacional e tempo de treinamento.

No caso deste trabalho, faz-se uso de um conjunto de arquiteturas de RNCP pré-treinadas com o BDA composto por mais de 330 mil imagens de 91 categorias (rótulos), chamada de *Microsoft* COCO *dataset* [33]. A arquitetura será reconfigurada para o BDA de insetos e retreinada, fazendo com que apenas as camadas iniciais e finais da RNCP sejam ajustadas. Além disso, a aprendizagem será feita por transferência de características.

2.8 Arquiteturas de Redes Neurais Convolucionais Profunda

A arquitetura de RNCP é uma pilha de camadas de módulos simples, sujeitos a aprendizagem. Muitos desses módulos calculam o mapeamentos de entrada-saída. Cada módulo da pilha transforma sua entrada, aumentando a seletividade e invariância da representação, mediante várias camadas não lineares. Um sistema pode implementar funções intrínsecas de suas entradas que são simultaneamente sensíveis a detalhes minuciosos e insensíveis a grandes variações irrelevantes [27].

Este trabalho se propôs a usar dois tipos de meta-arquitetura moderna de redes neurais

convolucionais, *Faster* R-CNN e SSD.

A principal contribuição da *Faster* R-CNN é a introdução da Rede de Regiões Propostas (RRP), que permite propostas regionais quase sem custos de processamento [6]. Já a da SSD foi a detecção de múltiplas referências e resoluções que melhoram significativamente a precisão de detecção de um detector de uma etapa, especialmente para alguns pequenos objetos [8].

2.8.1 *Faster Region - Convolution Neural Network*

Fast R-CNN [34] tem reduzido o tempo de execução de detecção ao utilizar algoritmos de proposta de região para se fazer hipóteses de localização de objetos. *Faster* R-CNN é a mesclagem da rede *Fast* R-CNN com uma RRP que compartilha características convolucionais de imagens completas com a rede de detecção, permitindo assim,propostas de região quase sem custo, criando uma única arquitetura de rede [6].

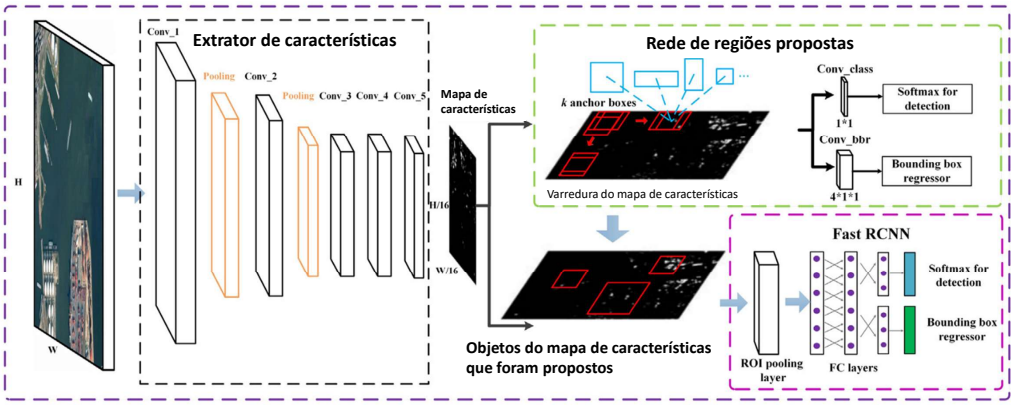


Figura 2.15: *Faster* R-CNN [35].

A detecção dos objetos e sua classificação acontece em duas etapas, como mostrado na Figura 2.15. Na primeira etapa, as imagens são processadas num extrator de características, e os mapas de características da última camadas convolucionais do extrator são usadas em uma rede de regiões propostas, que funciona usando uma técnica de molduras "âncoras" as quais variam sua altura e largura e alternam o valor de escala, varrendo todo o mapa de características, exemplificada na Figura 2.16, assim é feita uma detecção

de regiões de interesse independente da classe do objeto, somente diferenciando o que é fundo de objeto [36].

Na segunda etapa, as regiões de interesse do mapa de características selecionadas são padronizadas em relação de tamanho, num processo chamado *Region of Interest Pooling* (RoIP) e então passam por uma *Fast* R-CNN que aproveita atributos que foram extraídos na primeira etapa. Por fim, é feita a classificação dos objetos [35].

Esta arquitetura teve um grande ganho de performance relacionado a velocidade comparada com as arquiteturas anteriores [6].

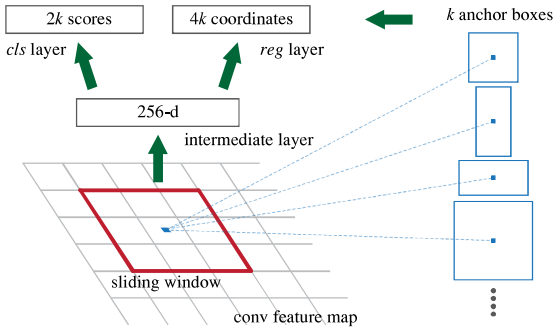


Figura 2.16: Molduras "ancoras"[6].

2.8.2 *Single Shot multibox Detector*

SSD é um método de detecção de objetos usando uma única RNCP do tipo *feed-forward* em um único estágio [7].

As camadas iniciais da rede são baseadas em uma arquitetura padrão utilizada para classificação de imagens de alta qualidade, após isso, é adicionado uma estrutura adicional de rede, conforme mostrado na Figura 2.17.

Ela discretiza o espaço de saída das caixas-delimitadoras em um conjunto de caixas-padrão em diferentes proporções e escalas por localização do mapa de características. No momento da previsão, a rede gera pontuações para a presença de cada categoria de objeto em cada caixa-padrão e produz ajustes na caixa para melhor corresponder à forma do objeto [7].

A rede combina previsões de vários mapas de características com diferentes resoluções para lidar naturalmente com objetos de vários tamanhos [7].

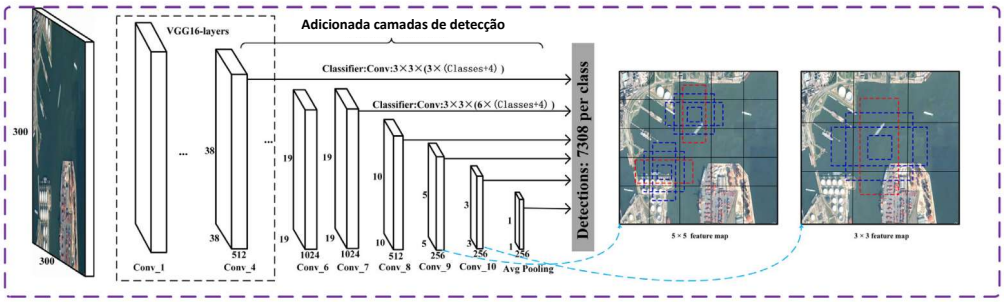


Figura 2.17: Arquitetura SSD [35].

2.9 Extratores de Características

Em todas as meta-arquiteturas de RNC, é aplicado um extrator de características convolutivo à imagem de entrada, para obter características de alto nível. Então, a escolha de um extrator de características é crucial para o sucesso do detector de insetos, uma vez que o número de parâmetros e tipos de camadas afeta diretamente a memória, a velocidade e o desempenho [36].

A seguir será mostrado duas redes: *Inception*-v2 e a *ResNet*-50, as quais constituem a base estrutural dos extratores de características usados neste trabalho, com estratégias distintas de funcionamento.

A principal contribuição do extrator de características *Inception*-v2 foi a introdução dos módulos *Inception*, o que ocasiona um aumento tanto na largura quando na profundidade da rede. Esse aumento é notado principalmente na largura, com seus filtros de tamanhos diferentes funcionando em paralelo. Quanto à *ResNet*-50, pode-se destacar sua importancia na introdução de estruturas de aprendizagem residual, chamadas de "*Identity Shortcut Connections*", as quais funcionam como "atalhos" para RNCP, incorrendo em ainda mais profundidade da rede.

2.9.1 *Inception*-v2

Inception é uma família de modelos de RNCPs, propostas pela Google Inc. desde 2014. A ideia geral da rede *inception* é aumentar tanto a largura como a profundidade da rede, utilizando "módulos *inception*" Figura 2.18, na qual são empilhados uns sobre os outros [8].

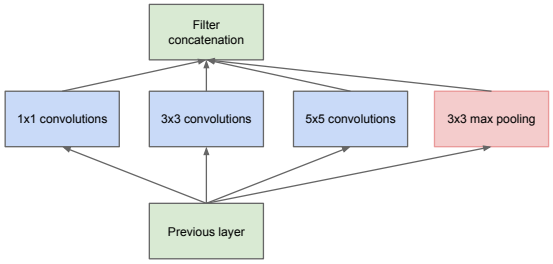


Figura 2.18: Módulo *Inception*-v1 [37].

O "modulo *inception*" realiza convoluções de filtros de tamanhos diferentes (1 x 1, 3 x 3, 5 x 5) e subamostragem (3 x 3 *max-pooling*) paralelamente, deixando-a mais larga, a qual ocasiona também uma diminuição no número de parâmetros e facilitando o treinamento [37].

Como as características de maior abstração são capturadas nas camadas superiores, espera-se que a as concentrações espaciais diminuam, sugere-se então, que as proporções dos filtros convolucionais devem aumentar à medida que avançamos em camadas mais altas [37].

A Figura 2.19 é mostrado a rede *GoogLeNet*, que possui 9 módulos *Inception* empilhados, totalizando 22 camadas [37].

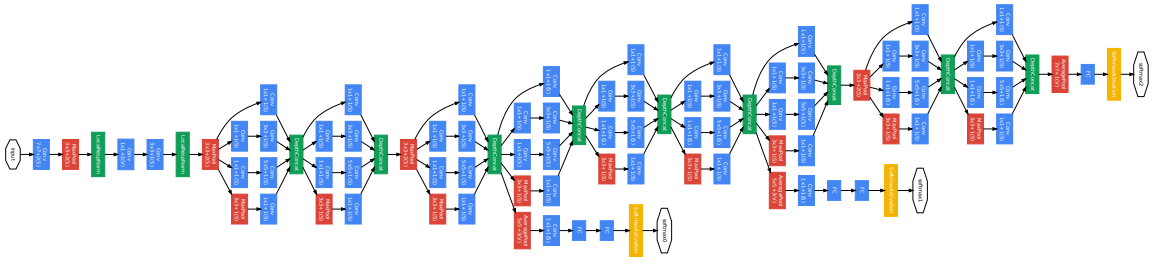


Figura 2.19: Rede *GoogLeNet* [37].

Inception-v2 é a evolução direta da *Inception*-v1, trazendo melhorias e ganho de performance. Suas principais contribuição foram a introdução da convolução fatorizada nos módulos *Inception* Figura 2.20, e a normalização dos lotes (*batch-normalization*) [8].

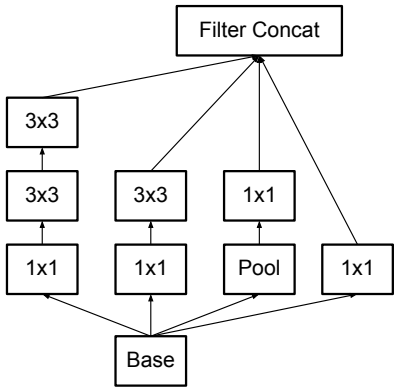


Figura 2.20: Módulo *Inception*-v2 [38].

2.9.2 *ResNet*-50

Depois do ressurgimento das RNCs em 2012, a cada modelo novo apresentado aumentava-se a sua complexidade e profundidade [8]. Em teoria, quando maior a profundidade de uma rede, maior será a acuracidade na classificação. Porém, RNCs são difíceis de treinar e sua precisão poderia ficar saturada [39].

Em 2015, foi proposto o modelo de redes residual profundas (*Residual Neural Network (ResNet)*), um novo tipo de arquitetura de RNCP, substancialmente mais profunda (até 152 camadas) compradas com as anteriores. Este tem o foco de facilitar o treinamento, reformulando suas camadas e introduzindo uma estrutura de aprendizado residual profundo [8].

A *ResNet* venceu várias competições de visão computacional aquele ano, incluindo *ImageNet*, COCO *detection*, entre outras [8].

A ideia das estruturas de aprendizagem residual profundo é adicionar "atalhos" entre blocos de camadas convolucionais, chamados de "*Identity Shortcut Connections*" Figura 2.21.

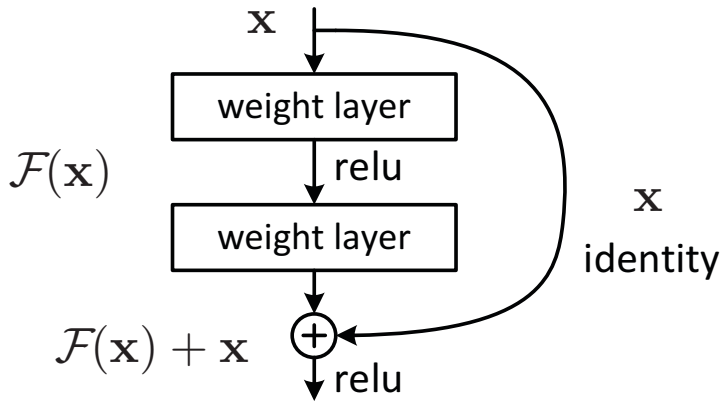


Figura 2.21: Estrutura de *Shortcut Connections* [39].

As conexões de atalho, pulando uma ou mais camadas, simplesmente executam o mapeamento de identidade e suas saídas são adicionadas às saídas das camadas empilhada, mas sem adicionar parâmetros extras e nem complexidade computacional. Dessa forma, a rede empilha camadas que não fazem nada [39].

Por tanto, a rede *ResNet*-50 é estruturada em 50 camadas distribuídas em 4 blocos de camadas convolucionais. Cada bloco possui um trio de camadas convolutivas juntamente com a estrutura de *Shortcut Connection*, que são repetidas, respectivamente em cada bloco em uma configuração de (3, 4, 6, 3), como é mostrado na Figura 2.22 [39].

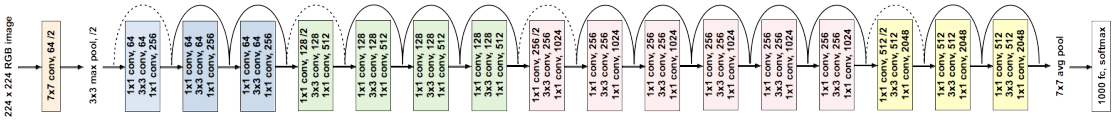


Figura 2.22: Arquitetura *ResNet*-50 [40].

Capítulo 3

Especificação do Sistema de Classificação Automática de Insetos

No Capítulo 3, primeiramente para contextualizar, será exposto um estudo de caso de monitoramento de pragas, com ênfase nos métodos tradicionais de identificação e classificação dos insetos. O objetivo será expor os principais problemas (Seção 3.1). Como modo de resolução destas questões negativas para uma maior eficiência, será apresentado a metodologia proposta na seção de Identificação e Classificação Automática de Insetos (Seção 3.2).

Posteriormente, serão apresentadas as ferramentas (Seção 3.3) e detalhamento dos métodos (Seção 3.4) utilizados para o estudo, implementação e desenvolvimento do identificador, classificador e contador automático de insetos. Para isso, faz-se uso dos recursos de aprendizagem profunda, mais precisamente, dos processos de treinamento das arquiteturas *Faster* R-CNN e SSD com os extratores de características *Inception*-v2 e *ResNet*-50, com os BDAs criados.

3.1 Monitoramento de Pragas em Citros

Recentemente, mais precisamente no ano de 2014 encontrou-se em Portugal o inseto *Trioza erytreae*. Este consiste numa praga cítrica muito perigosa já que pode transmitir a doença *Huanglongbing (Citros Greening)* [2].

Esta praga pode ocasionar uma doença considerada sem cura e ataca as culturas de citros (laranja, limão, tangerina), fazendo com que os frutos produzidos sejam amargos, rijos, disformes e incomestíveis. Seu efeito é extremamente danoso, pois normalmente representa a morte da planta após a infecção [2].

Apesar de ainda não ter sido detectada a doença em Portugal, o monitoramento da praga é de grande relevância para a tomada de decisão sobre estratégias de contingência pois visa antecipar uma possível disseminação da doença na Europa [2].

Atualmente, realiza-se um monitoramento em algumas plantações de citros, estas localizadas em uma região perto da cidade de Porto, Vairão. Este tem interesse em identificar e contabilizar cinco espécies de insetos, *Trioza erytreae*, *Syrphidae*, *Coccinellidae*, *Chrysopidae* e *Philaenus*.

Embora apenas o *Trioza erytreae* seja o único inseto vetor da doença *huanglongbing*, os outros quatro insetos são indicadores da presença da *Trioza erytreae*. Na tabela 3.1 é ilustrado os insetos e algumas informações importantes quanto as características morfológicas que auxilia a identificação.

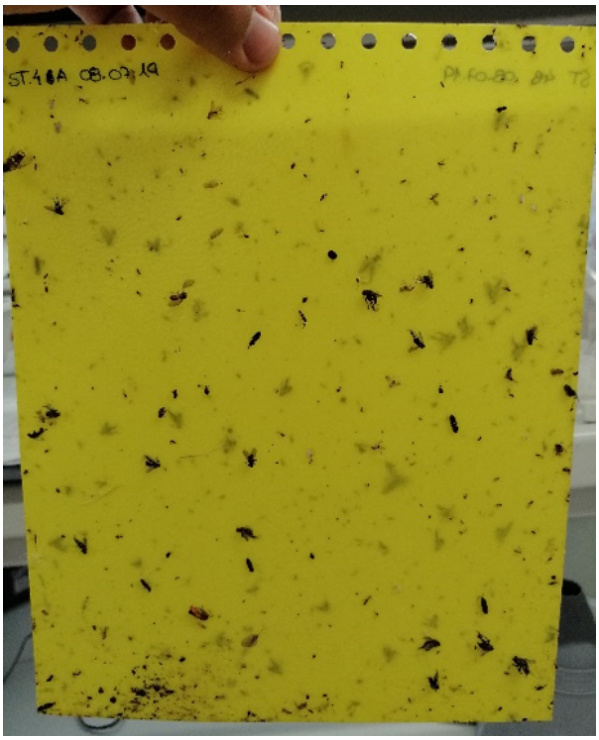
O levantamento dos dados de monitoramento é feito a partir da inspeção das armadilhas instaladas em pontos pré-estabelecidos na plantação. As armadilhas são do tipo adesiva de cor amarela, a Figura 3.1a ilustra como são as armadilhas. Elas são constituídas de um retângulo plástico de cor amarela, os quais possuem uma cola sem cheiro, sendo que esta não resseca ou sai com a chuva. Os insetos são atraídos pela cor amarela e ficam presos nas armadilhas ao se colidirem.

A inspeção das armadilhas é feita por uma pessoa, normalmente um especialista em insetos, que identifica e classifica os insetos e preenche uma planilha manualmente. A Figura 3.1b ilustra a planilha onde são realizadas as anotações dos insetos que estão

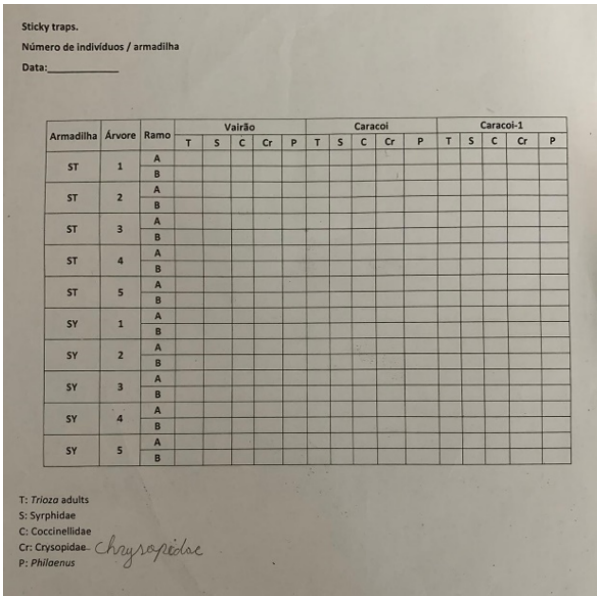
Tabela 3.1: Descrição e características morfológicas das pragas em culturas de citros.

Insetos	Descrição
	*Trioza erytreae* é um inseto picador-sugador da vasta família dos *Psylloidea*. São facilmente reconhecidos na fase adulta pela estrutura e veneração das asas anteriores, cores verde claro e castanho escuro e com 4 mm de tamanho médio [2].
	*Syrphidae* é uma família de moscas. Nem sempre é fácil identificá-las, principalmente, devido a semelhança com as abelhas. São caracterizadas por apenas um par de asas (diferente de abelhas que possuem dois pares), manchas ou faixas amarelas sobre um fundo da cor escura, e pelos densos cobrindo o corpo [41].
	*Coccinellidae* é uma família de insetos popularmente conhecida como Joaninha. Geralmente tem o corpo redondo e colorido, com pintas pretas, tamanho médio de 0.8 a 18 mm e superfície do corpo coberta com pequenos pelos [42].
	*Chrysopidae* é uma família de insetos *Neurópteros*. São insetos delicados, de tamanho médio, com dois pares de asas translucidas 6 e 34 mm de comprimento. Seu corpo é usualmente verde e brilhante, possui olhos compostos e antenas longas [43].
	*Philaenus* é uma espécie de inseto pertencente à família dos *Aphrophoridae*. A espécie pode atingir um comprimento de 5 a 7 mm. Geralmente são amarelados, castanhos ou pretos, com manchas mais brilhantes sobre um fundo escuto [44].

sendo monitorados na plantação de citros. As informações importantes destacadas nos dados analisados são: data, localização (armadilha, árvore, ramo), e os número de cada inseto.



(a) Armadilhas amarela adesiva.



Sticky traps.
Número de indivíduos / armadilha
Data:_____

Armadilha	Árvore	Ramo	Vairão					Caracoi					Caracoi-1				
			T	S	C	Cr	P	T	S	C	Cr	P	T	S	C	Cr	P
ST	1	A															
		B															
ST	2	A															
		B															
ST	3	A															
		B															
ST	4	A															
		B															
ST	5	A															
		B															
SY	1	A															
		B															
SY	2	A															
		B															
SY	3	A															
		B															
SY	4	A															
		B															
SY	5	A															
		B															

T: *Trioza* adults
S: Syrphidae
C: Coccinellidae
Cr: Crysopidae Chrysopidae
P: *Philaenus*

(b) Planilha de identificação e classificação de insetos das armadilhas.

Figura 3.1: Método de levantamentos dos dados de monitoramento.

Ao realizar uma análise dessa prática ou modo de execução da identificação e classificação dos insetos ou propriamente o monitoramento das pragas, é possível identificar alguns problemas e processos que poderiam ser melhorados.

Destaca-se os seguintes problemas:

- **Identificação dos insetos**: Devido ao fato que a contagem e a identificação dos insetos são realizadas manualmente por intermédio de um especialista em insetos, essa tarefa, representa um método trabalhoso e suscetível a erros.

- **Anotação dos dados**: Da mesma forma que a identificação dos insetos, a anotação dos dados de monitoramento também é manual, sendo assim, trabalhosa e suscetível

a erros. É possível ainda destacar um outro ponto, quando se deseja transcrever os dados para um histórico de monitoramento que pode dar prosseguimento aos erros elencados anteriormente.

- **Demanda de profissionais**: Entre os problemas, pode-se destacar a demanda por profissionais capacitados para a realização do processo de monitoramento dos insetos, ou seja, o recurso humano é escasso.

3.2 Identificação e Classificação Automática de Insetos

Para o desenvolvimento da solução de um identificador e classificador automático de insetos, especificamente, os: *Trioza erytreae*, *Syrphidae*, *Coccinellidae*, *Chrysopidae* e *Philaenus*, é proposta uma sequência de procedimentos dividida em 4 principais etapas, sendo cada uma delas com seus respectivos métodos de execução. Para uma melhor compreensão, a Figura 3.2 demostra a sequência de procedimentos.

A primeira consiste na realização de estratégias na aquisição das imagens. A segunda relaciona aos métodos de criação do BDA e como eles estão estruturados. A terceira tem como razão a aprendizagem, nesta etapa, especificamente, é onde a RNCP aprenderá o que é um inseto em um processo de treinamento.

Na última etapa, ocorre a criação de um aplicativo, o qual será responsável por processar as fotografias das armadilhas e trará como resultado a classificação, contabilização, visualização dos insetos e, posteriormente, gerará um histórico de identificação na forma de um banco de dados.

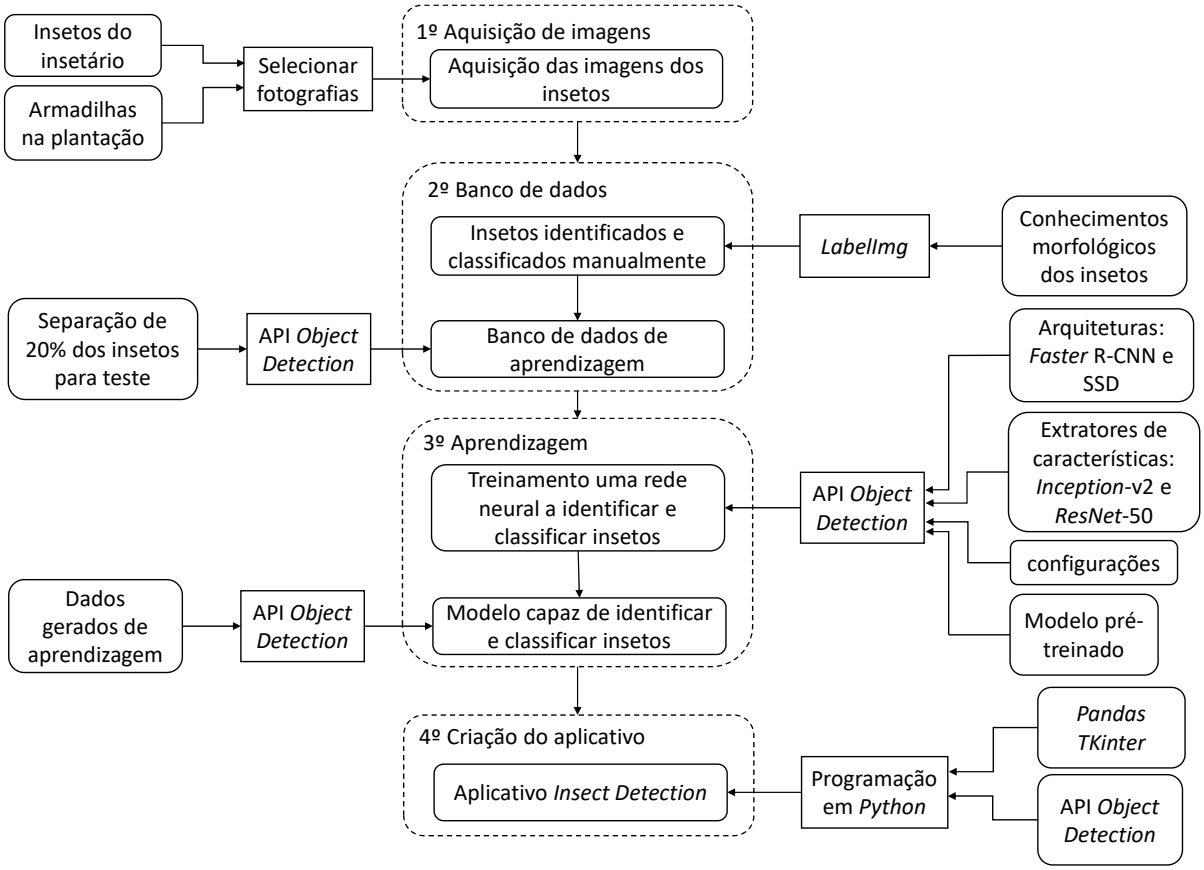


Figura 3.2: Sequência de métodos para o desenvolvimento da solução.

3.3 Ferramentas para o Estudo e Implementação

Para realizar os procedimentos, descritos na sessão anterior, é necessário utilizar um conjunto de ferramentas que irá compor todo um ecossistema de desenvolvimento e implementação da solução.

A escolha do *hardware* e dos *softwares* foi pensada para ser acessível e barato de implementar. Por isso foi escolhido um *hardware* modesto e *softwares* de código aberto (*open source*).

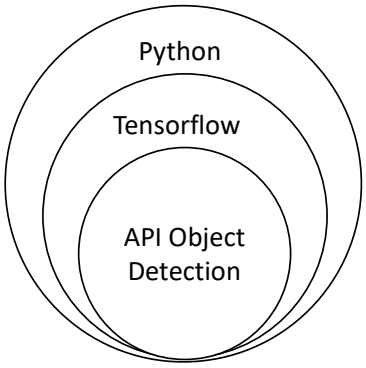
3.3.1 *Hardware*

O *hardware* utilizado será o computador pessoal, com o sistema operacional Windows 10. As características técnicas do computador são: Processador: Intel® Core™ i7-4500U (4M de cache, até 3,00 GHz), Memória RAM: 16 GB DDR3L, Placa de vídeo: NVIDIA GeForce 840M 2 GB, Armazenamento: SSD 480 GB.

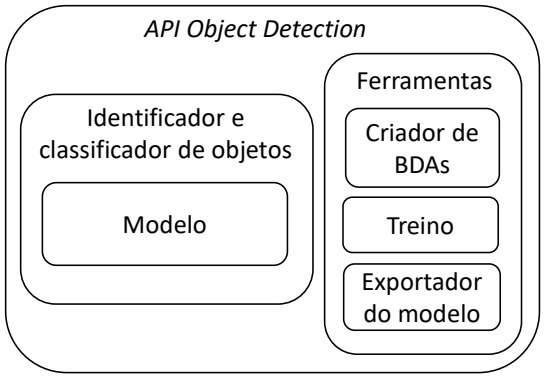
3.3.2 *Softwares*

Para construção, treinamento e implementação das RNCPs será utilizado um *framework*, de código aberto chamado API *Object Detection* [36], [45]. Esta API é construída em cima do pacote *Tensorflow* (para o treinamento das RNCPs será usado a versão 1.15, e a utilização do modelo treinado é na versão 2.0), que por sua vez é escrita em linguagem de programação *Python* (versão 3.7), a Figura 3.3a exemplifica a hierarquia do ecossistema.

A API possui uma grande gama de recursos e ferramentas que auxiliam de forma simplificada e indutiva a criação de soluções para a localização e identificação de objetos em imagens Figura 3.3b. Três ferramentas da API serão utilizadas durante o desenvolvimento do projeto: Criador de BDA (responsáveis pela criação de arquivos no formado



(a) Hierarquia do ecossistema.



(b) Recursos e ferramentas da API *Object Detection*.

Figura 3.3: Ecossistema de software para o desenvolvimento.

$TFRecord$); Algoritmo de treinamento (gera dados de aprendizagem, troca rápida de arquiteturas e extratores de características); e Exportador do modelo treinado. Utilizará, como recurso principal, o carregamento e funcionamento do modelo treinado [36].

Algumas ferramentas e pacotes auxiliares serão utilizadas durante algumas etapas de desenvolvimento, como, por exemplo, *Labeling* [46], que é uma ferramenta de anotação de rótulos em imagens. Esta será utilizada no processo de criação do BDA (as anotações são salvas em xml do formato $PASCAL\ VOC$).

Outra ferramenta auxiliar muito importante é o *TensorBoard* [47], que é uma subferramenta do *Tensorflow*. Ela irá fornecer uma visualização gráficas dos dados de aprendizagem durante a etapa de treinamento.

Quanto aos pacotes auxiliares, podemos destacar dois, *Tkinter* [48], que é pacote padrão do *Python* para trabalhar com interfaces gráficas, que será utilizado na etapa de criação do Aplicativo. E por último, o pacote *Pandas* [49] será utilizado para criar as estruturas de *dataframes*, que possui como função a criação e exportação dos dados coletados a partir das identificações dos insetos.

3.4 Detalhamento dos Métodos de Implementação e Desenvolvimento

Nesta seção, será explicada cada umas das etapas propostas para o desenvolvimento do identificador e classificador automáticos de insetos. Importante destacar que a ordem de desenvolvimento apresentada nesta sessão, deve ser respeitada para que este trabalho seja replicado com sucesso.

3.4.1 Aquisição de Imagens das Armadinhas

A etapa de aquisição de imagem será a peça fundamental para o desenvolvimento do identificador e classificador de insetos. As imagens terão como forma de aquisição fotografias dos insetos dispostos nas armadilhas autocolantes, conforme é mostrado na Figura 3.4. Serão necessárias muitas fotografias de armadilhas em boa qualidade, as quais devem conter os insetos de interesse em diferentes posições e tamanhos. Outro ponto importante é existir uma homogeneidade na quantidade de cada inseto, por exemplo, se forem fotografados 150 *Triozas erytreae*, então os outros insetos devem ter aproximadamente essa mesma quantidade.

Figura 3.4: Fotografia das armadilhas autocolantes.

3.4.2 Banco de Dados de Aprendizagem

O BDA é considerado a base de conhecimento que as RNCPs utilizarão para aprender. No nosso caso, requer-se que a RNCP aprenda o que são insetos e como identificá-los. Para isso, é preciso criar o referido BDA [18].

A proposta consiste em utilizar fotografias dos insetos mediante uso das armadilhas que encontram-se na plantação de citros e identificar e classificar manualmente os insetos provenientes dos conhecimentos morfológicos. A Tabela 3.1 é um exemplo de conhecimentos necessários para a realização adequada do processo. Pode-se constatar que estas tarefas podem ser feitas por intermédio de um especialista com o auxilio de um programador.

O software utilizado para realizar essa tarefa será o *Labelimg*. É preciso separar BDA em BDA de treino e BDA de teste, que deverá ter uma relação de 20% para teste e 80% para treino aproximadamente [25]. Devido ao fato que API Object Detection trabalha um sistema especifico de BDA é necessário realizar uma conversão dos arquivos do BDA, de imagens em *jpg* para um sistema de arquivos *TFrecord* [50]. Para uma melhor compreensão, a Figura 3.5 exemplifica em forma de fluxograma esse processo.

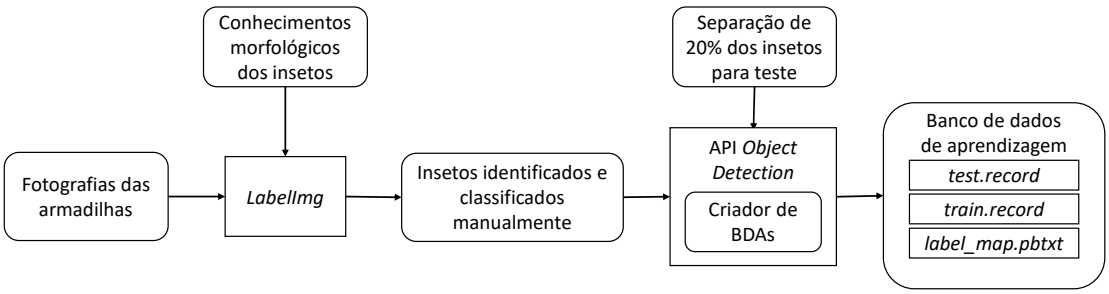


Figura 3.5: Processo de criação do BDA.

3.4.3 Processo de Aprendizagem

Após os BDA prontos, a próxima etapa é a aprendizagem. É nessa etapa que a RNCP irá assimilar o que são os insetos e como classificá-los, isso mediante um processo conhecido como **treinamento**.

O treinamento será integralmente realizado na API *Object Detection*, sendo que ao final de cada processo de aprendizagem é feita uma exportação dos dados gerados do treinamento em um **modelo** de inferência capaz de fazer a identificação e classificação dos insetos. A Figura 3.6 representa o diagrama do processo de aprendizagem proposto, o qual resulta nos modelos pretendidos.

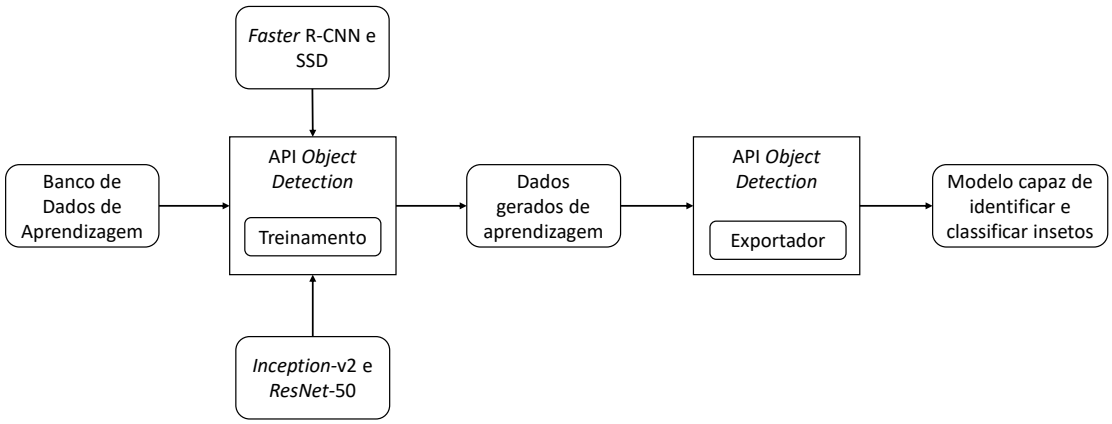


Figura 3.6: Processo de aprendizagem.

Uma das finalidades desta etapa consiste em treinar diferentes tipos de arquiteturas de RNCPs com os BDAs. Com estes treinamentos é possível realizar uma comparação para determinar qual dos modelos apresenta os melhores resultados. Pra isso, foi escolhido duas meta-arquiteturas: *Faster* R-CNN e SSD, assim como dois extratores de características: *Inception*-v2 e ResNet-50. A escolha dessa configuração baseia-se pelos modelos pré-treinados disponíveis na API *Object Detection*, como também pelo fato de representarem as arquiteturas mais modernas (maiores valores de COCO mAP) [36], [45].

Esse processo terá quatro configurações diferentes de treinamento, *Faster* R-CNN com *Inception*-v2, *Faster* R-CNN com *ResNet*-50, SSD com *Inception*-v2 e SSD com *ResNet*-50.

3.4.4 Aplicação Gráfica para Monitoramento de Pragas em Citros

O aplicativo será a **implementação** do modelo resultante da última etapa em uma **interface gráfica de usuário (*Graphical User Interface* (GUI))** amigável, o qual será denominado *Insect Detection*. Este deve cumprir as seguintes funções exemplificadas no diagrama da Figura 3.7.

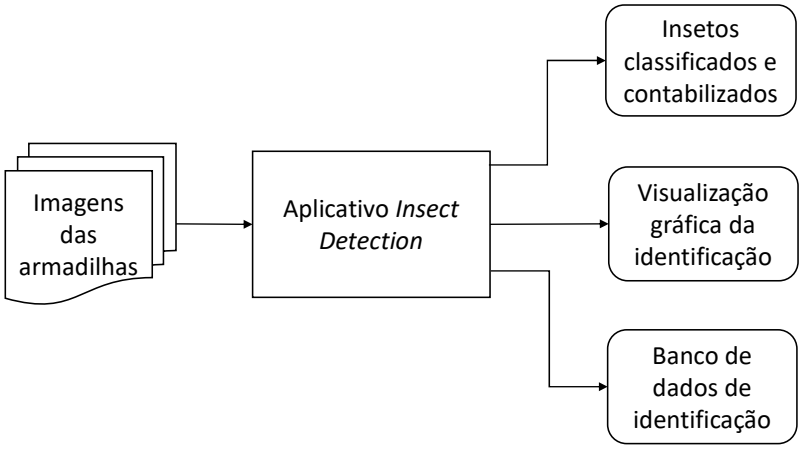


Figura 3.7: Funções do aplicativo

Conforme o usuário seleciona as imagens que ele deseja processar para identificar e classificar os insetos, o aplicativo processará as mesmas e então mostrará, em forma de visualização gráfica, a contabilização da identificação e classificação dos insetos.

Todos os resultados processados serão arquivados na forma de um banco de dados (data, localização das armadilhas, contabilização dos insetos).

O processo de desenvolvimento do aplicativo *Insect Detection* está representando na forma de um fluxograma na Figura 3.8.

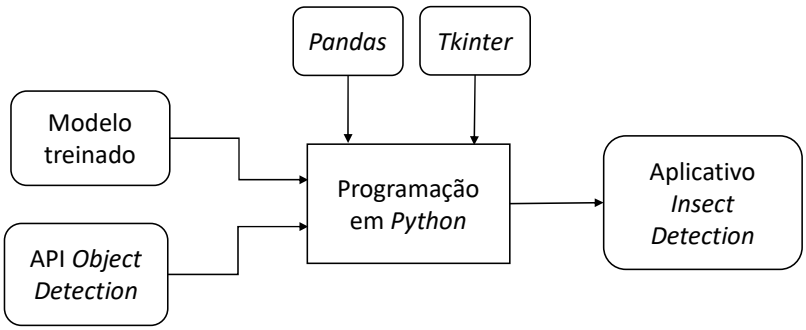


Figura 3.8: Desenvolvimento do aplicativo

O seu desenvolvimento será feito em *Python* e terá como base a API *Object detection* assim como o adicional de alguns pacotes, por exemplo, o *TKinter* para a interface gráfica e o *Pandas* para trabalhar com banco de dados.

Capítulo 4

Desenvolvimento do *Insect Detection*

Conforme já descrito no decorrer deste trabalho, o objetivo principal consiste em desenvolver uma ferramenta com a habilidade de identificar, classificar e contabilizar os insetos, esta realizada de forma automática, rápida, e precisa. Para isso, utiliza-se recursos de aprendizagem profunda.

Neste capítulo, será demonstrado como realizou-se o desenvolvimento da aplicação gráfica, e também a devida implementação das metodologias apresentadas no Capítulo 3.

Ocorre nesse ponto, a divisão em duas etapas. A primeira tem como foco criar um modelo com a capacidade de identificar e classificar cinco tipos de insetos e realizar os estudo de desempenho de treinamento de diferentes arquiteturas de RNCPs (Seções 4.1, 4.2 e 4.3). Já a segunda objetiva a representação de como embarcar o modelo que identifica e classifica insetos em uma interface gráfica amigável, na forma de uma aplicação, aqui chamada de *Insect Detection* (Seção 4.4).

4.1 Aquisição de Imagens das Armadilhas

A parte inicial do desenvolvimento inicia-se com a coleta de imagens para compor o BDA. Essas imagens são provenientes de fotografias das armadilhas autocolantes.

Como explicado anteriormente na (Sessão 3.4.1) do Capítulo 3, existem vários requisitos estabelecidos que devem ser cumpridos na aquisição das imagens, as quais passarão a compor o BDA.

4.1.1 Armadilhas da Plantação

A primeira amostra foi composta por 494 fotografias das armadilhas autocolantes, as quais foram adquiridas diretamente na plantação ou nos laboratórios do IPB. Uma pré-seleção das fotografias foi realizada para selecionar aquelas que contivessem os insetos de interesse, a Figura 4.1 exemplifica essa ação. As que não possuíam esse objeto de relevância, quais sejam os insetos que constituem o foco do estudo, foram descartadas. Dessa maneira, ao final, selecionou-se, ao total, 93 fotografias.

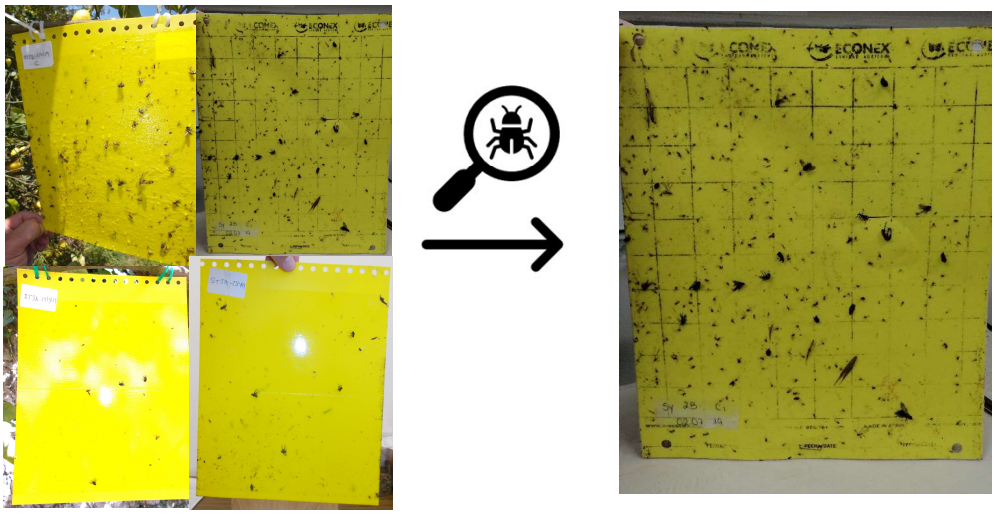


Figura 4.1: Pré-seleção das fotografias das armadilhas.

4.1.2 Insetos do Insetário

Como o número de fotografias e de exemplos de alguns insetos da primeira amostra foram baixos, necessitou-se realizar uma segunda coleta de fotografias. Como estratégia de aquisição para complementação dessas imagens insuficientes colhidas na primeira amostra, coletou-se os insetos de interesse no insetário do IPB. Como modo de operação, ilustrada na Figura 4.2, estes foram colocados nas armadilhas e posteriormente fotografados, sendo que realizou-se mudanças tanto nas disposições dos insetos e quanto na posição dos mesmos.

Assim, criou-se uma nova amostra com a certeza que todas as fotografias tinham os insetos de interesse, não sendo necessário, deste modo, realizar uma pré-seleção. Ao total foram realizada 48 fotografias.

Importante salientar não havia disposição dos inseto *Syrphidae* no insetário. Desta forma, não foi possível a criação de uma nova amostra contendo esse tipo de inseto especificamente.

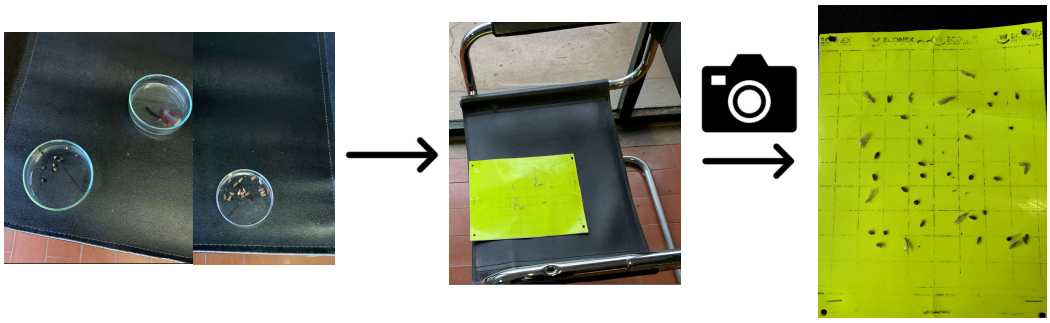


Figura 4.2: Fotografias dos insetos coletados e dispostos nas armadilhas.

4.2 Banco de Dados de Aprendizagem

Recapitulando, BDA é formado por vários exemplos (fotografias) de insetos identificados. As imagens, por sua vez, serão divididas em duas parcelas de BDA, sendo uma para treino e outra para teste. Os dados estabelecidos neste processo (contidos no BDA) devem ser convertidos para o formato *TFRecord*.

4.2.1 Caracterização da Base de Conhecimento

Inúmeros processos antecedentes são necessários para a criação do BDA, como por exemplo, a identificação e classificação manual dos insetos. É nessa fase que são adicionadas representações à sua base de conhecimento, para que posteriormente, na etapa de treinamento, elas aprendam o que são os insetos e como identifica-los.

Identificação e classificação manual dos insetos

Com o auxílio da Tabela 3.1 e de um especialista em insetos, assim como a utilização do *software Labelimg*, como o processo exemplificado na Figura 4.3. Realizou o precedimento de identificação e classificação manualmente dos insetos manualmente na primeira e segunda amostra.

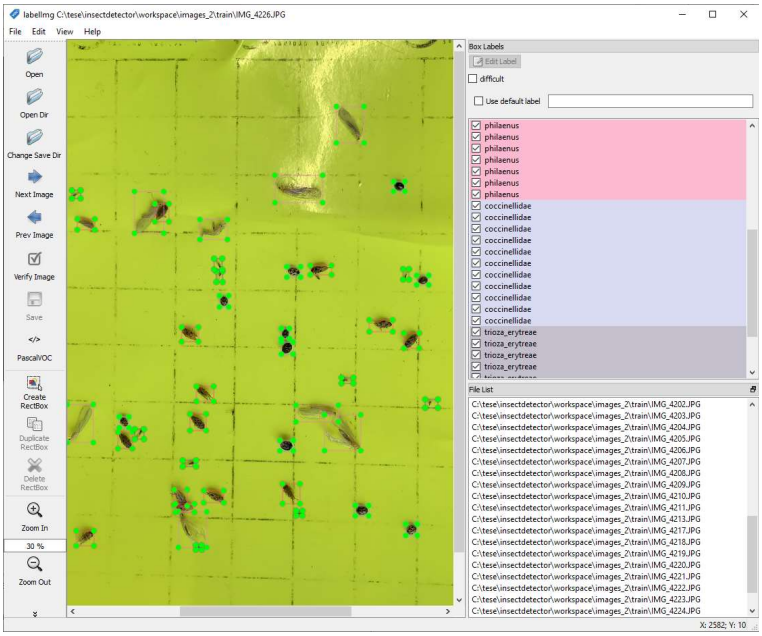


Figura 4.3: Ferramenta *Labelimg* para classificação manual dos insetos.

Gerou-se, dessa forma, uma série de dados com informações (coordenadas e rótulos) dos insetos nas fotografias. Em outras palavras, foi adicionada a representações a base de conhecimento.

Os números de cada tipo de inseto inclusos em cada uma das amostras serão apresentados posteriormente na (Sessão 5.1.1) do Capítulo 5.

Separação dos dados de treino

A separação dos BDAs em dados de treino e de teste é muito importante para o sucesso da aprendizagem. E seguindo a orientação de vários trabalhos realizados, os dados deverão ser dispostos na proporção aproximada de: 80% para treino e 20% para teste [6], [51]–[53].

Tendo como ponto de partida as duas amostras caracterizadas na etapa antecedente, efetivou-se a criação de três diferentes BDAs. O primeiro e o segundo BDA são compostos mediante a utilização da primeira e segunda amostra, respectivamente. Já o terceiro, a sua composição foi feita com a junção de ambas.

A relação dos insetos do BDA separados em BDA de treino e teste para as diferentes amostras podem ser encontradas na (Sessão 5.1.2).

4.2.2 Criação do Banco de Dados de Aprendizagem

Como mencionado anteriormente (Sessão 3.4.2), é necessário converter as imagens dos BDAs e dos dados(coordenadas e rótulos) para um sistema de arquivo específico (*TFRecord*). Esse procedimento é utilizado para que a API *Object Detection* consiga realizar o processo de aprendizagem.

A principal características desse sistema de arquivo é armazenar dados em uma sequência de registros binários, o que acarreta um ganho de velocidade e performance durante a etapa de treinamento [50].

Apesar de existir algoritmos próprios de conversão (*create_pascal_tf_record.py* e *create_pet_tf_record.py*) na API, estes acabam por ser muito genéricos e demandam a realizção de muitas alterações para funcione com o BDA de imagens do trabalho, e por esta razão utiliza-se um algoritmo modificado e unificado (*generate_tfrecord.py*) para a conversão [54].

Como resultado da conversão, cada BDA será composto por apenas três arquivos:

train.record, *test.record* e *label_map.pbtxt*. Sendo que o último arquivo corresponde a uma lista (rótulos) dos insetos que compõem o BDA.

As imagens e os BDAs do projeto estão disponíveis para download no repositório *Gitlab*, assim como o link de *download* está disponível no Apêndice C. Já os algoritmos apresentados nessa sessão podem ser consultados no Apêndice B.1.

4.3 Processo de Aprendizagem com Diferentes Arquiteturas

A segmentação do processo de aprendizagem é composta por três etapas, ilustrada na Figura 4.4, as quais devem ser executada em sequência, conforme a seguinte ordem, **Configurações**, **Treinamento** e **Exportação de modelo treinado**. No decorrer desta seção serão explicada e aprofundadas cada uma dessas etapas.

Ao final desse processo de aprendizagem resultará em um modelo capaz de identificar e classificar insetos para cada configuração de arquitetura e BDA utilizados.

4.3.1 Configurações dos Modelos

As configurações são compostas de **Arquivos de entradas** e **Parâmetros**. Os arquivos de entrada são formados pelos BDAs (*train.record*, *test.record* e *label_map.pbtxt*).

Em "parâmetros" serão encontradas todas as configurações de treinamento e os modelos pré-treinados. Indubitavelmente, os ajustes dessas configurações representam uma peça fundamental para o processo do treinamento. Na Figura 4.5 é mostrado a estruturação das configurações (*configurações.config*) dividida em cinco grupos: *model*, *train_config*, *train_input_reader*, *eval_config* e *eval_input_reader*.

Os parâmetros do *model*, como meta-arquitetura e extrator de características estão correlacionados, de forma direta, com todos os parâmetros do *train_config* e também com a métrica usada na avaliação do *eval_config*.

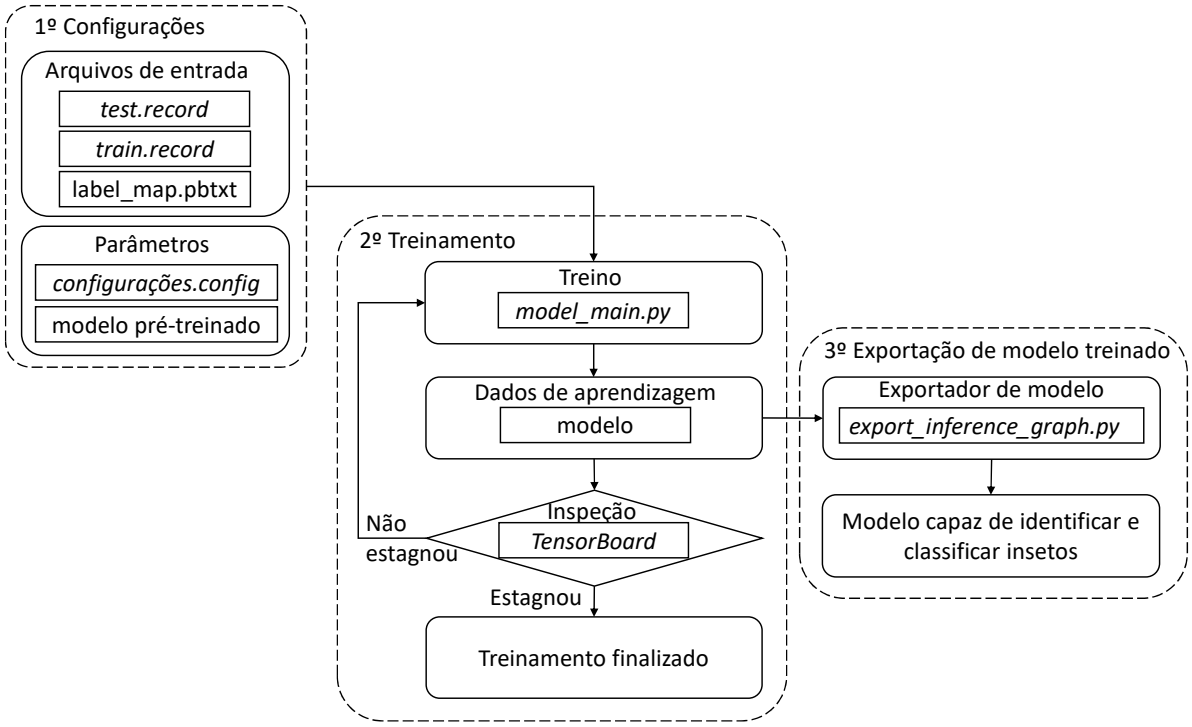


Figura 4.4: Processos de aprendizagem.

Já os parâmetros que estão relacionados com os arquivos de entrada, isto é, os parâmetros de *train_input_reader* e *eval_input_reader* são diretamente correlacionadas com o parâmetro do número de conjunto de dados de avaliação do *eval_config*.

Como já dito (Sessão 3.4.3), foram definidos vários tipos de configurações de treinamento. Essas levaram em conta os três diferentes BDAs, duas meta-arquiteturas e dois extratores de características. Na Tabela 4 são dispostas todas as configurações de treino propostas.

Tabela 4.1: Configurações de treinamento.

Treino	**Arquivos de entrada**	**Meta-arquitetura**	**Extrator de características**	**Modelo pré-treinado**
model-1	BDA 1	*Faster* R-CNN	*Inception*-v2	faster_rcnn_inception_v2_coco_2018_01_28
model-2	BDA 1	*Faster* R-CNN	*ResNet*-50	faster_rcnn_resnet50_coco_2018_01_28
model-3	BDA 1	SSD	*Inception*-v2	ssd_inception_v2_coco_2018_01_28
model-4	BDA 1	SSD	ResNet-50	ssd_resnet50_v1_fpn_coco14_sync_2018_07_03
model-5	BDA 2	*Faster* R-CNN	*Inception*-v2	faster_rcnn_inception_v2_coco_2018_01_28
model-6	BDA 2	*Faster* R-CNN	ResNet-50	faster_rcnn_resnet50_coco_2018_01_28
model-7	BDA 2	SSD	*Inception*-v2	ssd_inception_v2_coco_2018_01_28
model-8	BDA 2	SSD	ResNet-50	ssd_resnet50_v1_fpn_coco14_sync_2018_07_03
model-9	BDA 3	*Faster* R-CNN	*Inception*-v2	faster_rcnn_inception_v2_coco_2018_01_28
model-10	BDA 3	*Faster* R-CNN	ResNet-50	faster_rcnn_resnet50_coco_2018_01_28
model-11	BDA 3	SSD	*Inception*-v2	ssd_inception_v2_coco_2018_01_28
model-12	BDA 3	SSD	ResNet-50	ssd_resnet50_v1_fpn_coco14_sync_2018_07_03

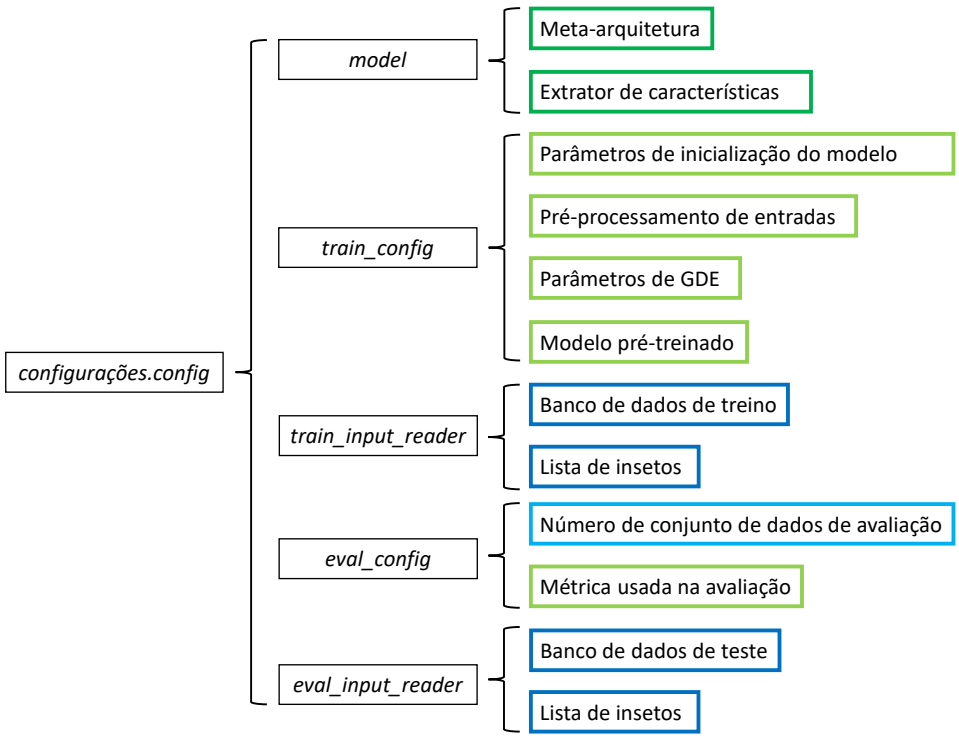


Figura 4.5: Estrutura de configuração de treinamento.

No Apêndice C encontra-se disponível todas as configurações de todos os treinos usados nesse projeto.

4.3.2 Processo de Treinamento

Com as configurações já estabelecidas, a próxima etapa da aprendizagem consiste no processo de treinamento. Cada uma das 12 configurações definidas na Tabela 4.1 irá passar por um processo de treinamento e consequentemente cada uma delas criará um modelo com diferentes performances relacionadas a classificação e identificação de insetos.

O processo de treinamento é executado no *model_main.py* que também faz parte da API. Devido ao hardware utilizado, o processo de treinamento irá demorar várias horas para ser finalizado. Esse fato ocorre por causa do alto poder computacional requisitado para esta operação.

Durante o processo, cada treino irá criar e armazenar **dados de aprendizagem**. Esses dados podem ser inspecionados (ou monitorados) pela ferramenta *TensorBoard*. dessa

forma, será possível analisar, em tempo real, a performance de treinamento, conforme é mostrado na Figura 4.6. Isso sera feito mediante um conjunto de dados métricos que ela fornece, como por exemplo: dados de épocas (ciclos), dados de referência entrada/-saída, precisão de classificação, precisão de localização, além de outras métricas definidas nas configurações. No Apêndice B.1 é possível encontrar o algorítimo de treinamento *model_main.py*.

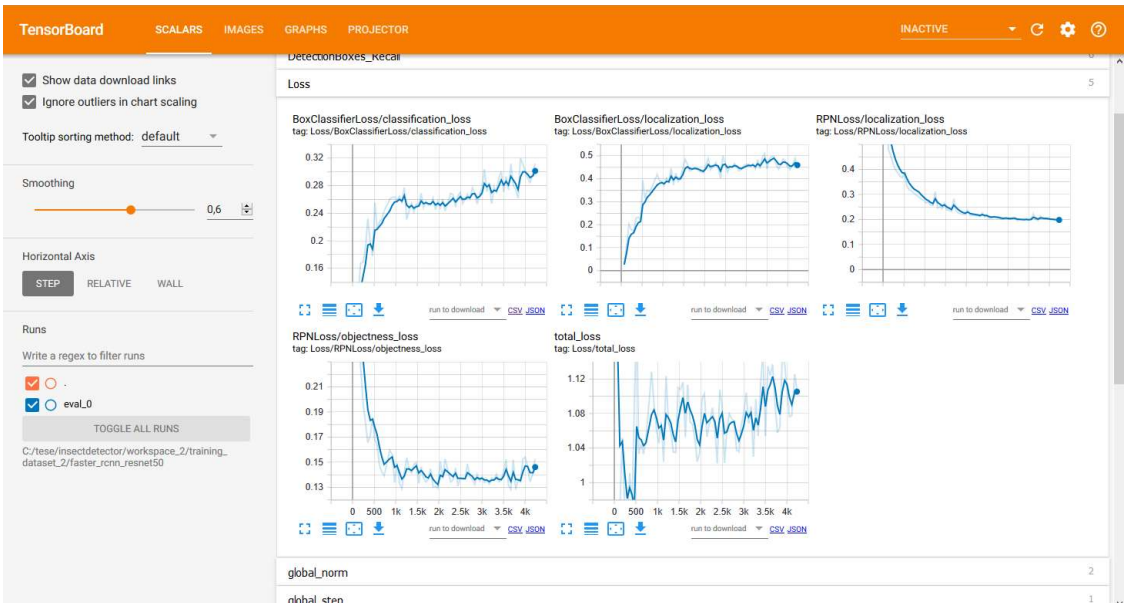


Figura 4.6: *TensorBoard* usado para avaliação dos treinos dos modelos em tempo real.

O critério de finalização do processo de treinamento ocorre quando os valores métricos de treinamento estagnarem, isto é, não se alteram significativamente em função das épocas (ciclos). Nesse momento, especificamente, acontece a aprendizagem da classificação e localização dos insetos. Como critério para finalização do processo de treinamento foi considerado a estagnação da métrica *Total Loss*, que é definida como a função erro internamente realizada no processo de aprendizagem.

Ao final dessa etapa, que começou desde o processo de aquisição de imagens das armadilhas e terminando com os treinos dos modelos com as diferentes configurações arquiteturas. Gerou-se uma grande quantidade de dados e informações dos treinos das arquiteturas.

Dessa forma, foi possível realizar o estudo de desempenho de treinamento de diferentes arquiteturas (*Faster* R-CNN e SSD com *Inception*-v2 e *ResNet*-50) com os diferentes BDA, usadas para identificar e classificar das pragas de citros. Os resultados desses estudos podem ser encontrados na sessão Resultados intermediários (Sessão 5.1)

4.3.3 Modelo de Identificação e Classificação dos Insetos

A última etapa do processo de aprendizagem é a exportação dos dados que foram gerados na etapa de treinamento. Com isso, será possível estabelecer um modelo capaz de identificar e classificar os insetos, sendo que um dos treinos resultará em um modelo diferente. Esse procedimento é realizado no algorítimo *export_inference_graph.py* (Apêndice B.3), que também faz parte da API.

Após a exportação, o modelo será composto de:

- *saved_model/*: um diretório contendo o formato do modelo gravado do modelo exportado;
- *frozen_inference_graph.pb*: um *frozen graph* formado do modelo exportado;
- *model.ckpt.**: os pontos de verificação do modelo usados para exportar;
- *checkpoint*: um arquivo especificando para restaurar arquivos de pontos de verificação incluídos;
- *pipeline.config*: arquivo de configuração para o modelo exportado

Todos os modelos do projeto estão disponíveis para download no repositório *Gitlab*, e o link de *download* estão disponíveis no Apêndice C.

4.4 Aplicativo *Insect Detection*

Todas as etapas anteriores foram realizadas para que no final um modelo eficaz fosse criando, desempenhando de forma correta a identificação e classificação dos insetos. Para desenvolver a plataforma objeto desse trabalho, utilizou-se esse modelo de identificação e classificação mediante uma aplicação amigável e de fácil utilização pelo usuário. Seu funcionamento se dará da seguinte forma conforme a Figura 3.7. O usuário seleciona as imagens que deseja a fim de identificar e classificar os insetos. O aplicativo, por sua vez, utilizando o modelo previamente treinado irá processar as imagens escolhidas, estabelecendo algumas informações: visualização gráfica e contabilização da identificação e classificação dos insetos. Essas informações são guardadas num banco de dados juntamente com informações de data e localização.

4.4.1 Arquitetura do Aplicativo

O aplicativo foi desenvolvido em *Python* integrado com a API *Object Detection*. Fez-se necessário também, o uso de alguns pacotes auxiliares, tais como: *TKinter*, *PIL* e *Pandas* Figura 4.7.

O *TKinter* foi utilizado para criar a interface de usuário, bem como a visualização gráfica e recursos de manipulação das imagens para processamento.

PIL teve como função tratar a imagem após ela passar pelo modelo de identificação e classificação dos insetos.

Pandas foi usado para criar a estrutura de bando de dados e para o mesmo ser exportado para um arquivo *csv*, o qual pode ser carregado pelo próprio aplicativo exibindo um histórico. Cabe salientar aqui que *csv* possui a característica de ser intercambiável entre várias plataformas diferentes de análise de dados, e assim sendo, representa um arquivo universal tendo uma alta gama de aplicação .

O algorítimo (*app_insect_detector.py*) do aplicativo *Insect Detection* está disponível no Apêndice B.4.

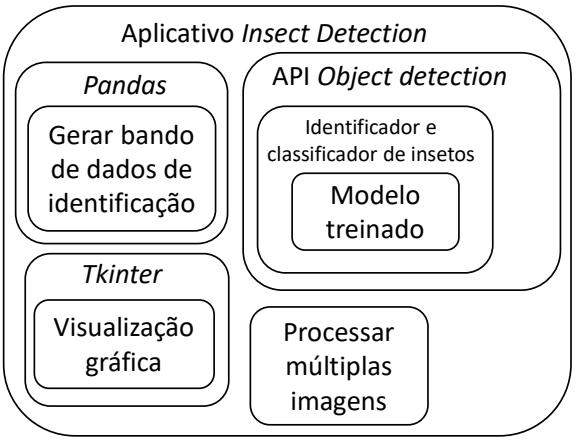


Figura 4.7: Estrutura do Aplicativo *Insect Detection*.

4.4.2 Interface Gráfica de Usuário (GUI) do *Insect Detection*

A interface gráfica de Usuário (GUI) do *Insect Detector* resultante é mostrado na Figura 4.8.

No canto esquerdo superior fica a parte que seleciona as imagens que serão analisadas. Clicando no botão *Select Images* é aberto uma janela Figura 4.9, a qual apresentará uma lista de imagens.

Logo abaixo existem dois botões, *Detect Insect* e *all files process*. O primeiro serve para fazer a detecção dos insetos da imagem selecionada dentre as dispostas na lista, já o segundo botão serve para fazer a detecção de todos as imagens selecionada. Quando este último botão é clicado, um alerta é mostrado, informando que o processo pode tomar tempo e perguntando se deseja continuar conforme demonstrado na Figura 4.10.

A visualização da identificação dos insetos é mostrada à direita, a qual contém a imagem completa, sendo os insetos condidos nela, delimitados mediante retângulos. É fornecida uma legenda com o nome do inseto e a porcentagem de acuracidade da identificação tal como apresentado na Figura 4.8. Juntamente com a visualização, é mostrado, logo abaixo, o nome e o número dos insetos identificados na imagem.

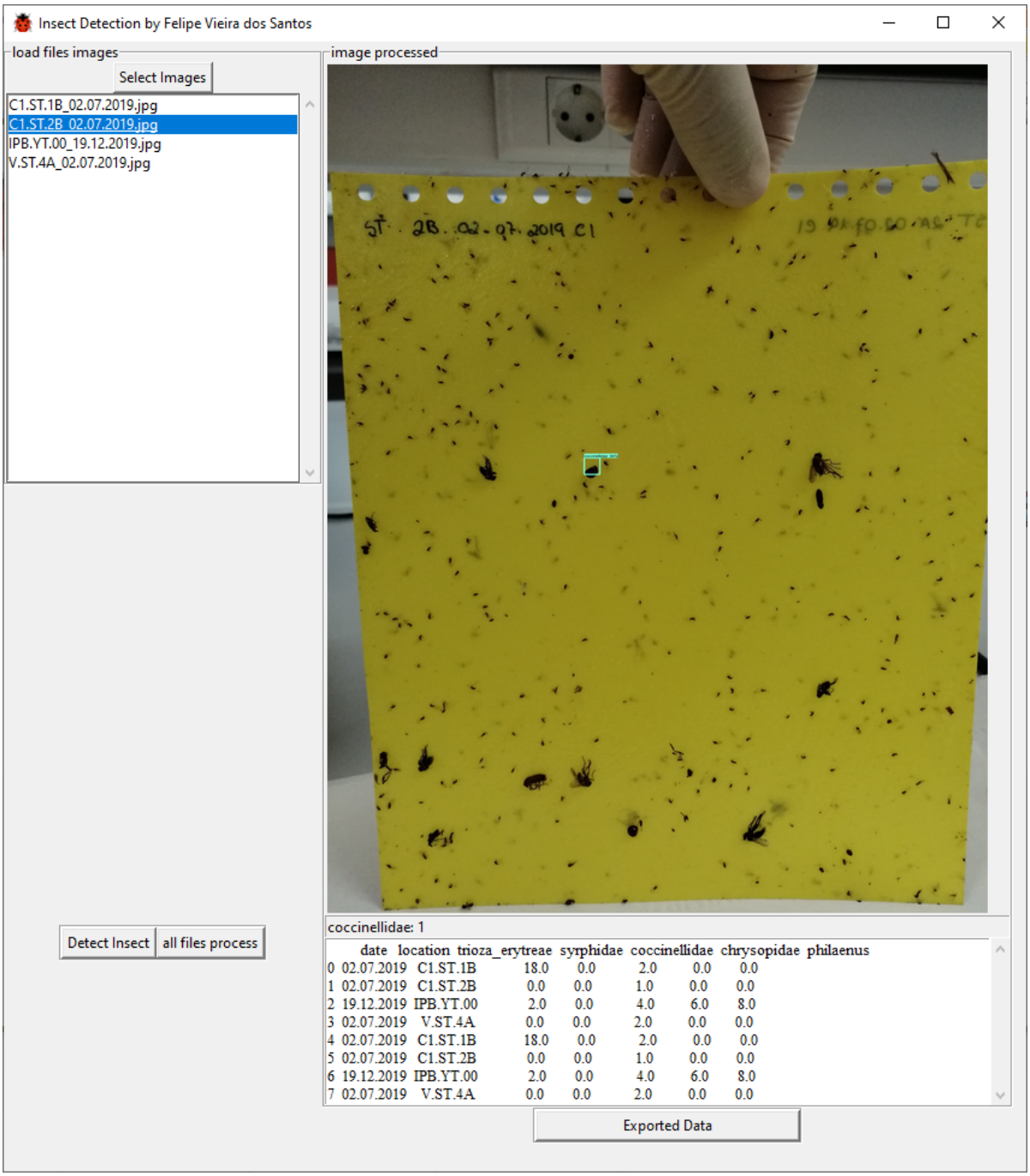


Figura 4.8: Interface Gráfica de Usuário (GUI) do *Insect Detector*.

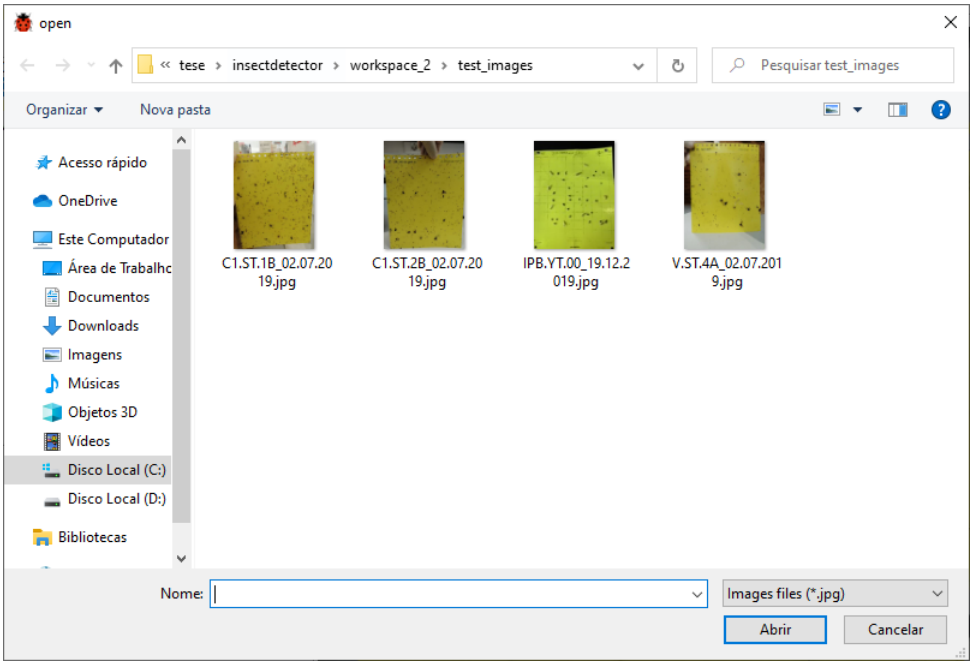


Figura 4.9: Janela de seleção de imagens.

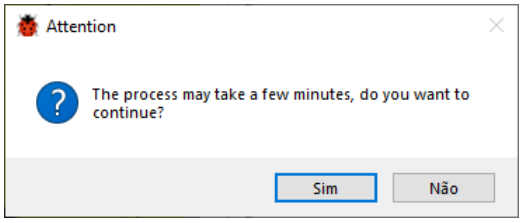


Figura 4.10: Alerta de processo de múltiplos arquivos.

A última definição visual, que se encontra na esquerda inferior do aplicativo, disponibiliza uma lista de dados de identificação e um botão *Exported Data*. Nessa lista é possível visualizar todos os dados de cada imagem analisada, tais como: localização, data, número de identificação de cada inseto. Já o botão tem a utilidade de salvar os dados de identificação, assim como um arquivo *csv*. Caso exista um banco de dados de identificação, os mesmos serão carregados quando se abrir o aplicativo.

Capítulo 5

Resultados e Discussões

Neste capítulo, será feita a divisão em duas partes, sendo a primeira direcionada à apresentação dos dados, resultados e discussões do estudo de desempenho de treinamento de diferentes arquiteturas (*Faster* R-CNN e SSD com *Inception*-v2 e *ResNet*-50) com os diferentes BDAs usadas para desenvolver o identificador e classificador das pragas de citros (Sessão 5.1).

Já na segunda parte, o foco passa a ser a demonstração dos resultados e discussões gerados ao se realizar a análise das capacidades do aplicativo desenvolvido *Insect Detection* (Sessão 5.2).

5.1 Resultados do Estudo de Desempenho das Diferentes Arquiteturas

Os resultados obtidos mediante o estudo de desempenho do treinamento das diferentes arquiteturas têm como teor a análise dos dados iniciais que foram usados como fator decisivo para a escolha de estratégias utilizadas nas etapas do desenvolvimento da aplicação gráfica *Insect Detection*.

Primeiramente, serão apresentados os resultados e discussões relativos à criação dos

BDAs (Sessões 5.1.1 e 5.1.2). Na sequência, será explicado brevemente as métricas utilizadas para estas avaliações (Sessão 5.1.3), e assim, finalmente, serem apresentados os resultados e discussões das avaliações dos processos de aprendizagem realizados (Sessão 5.1.4).

5.1.1 Atributos e Quantidade dos Exemplares de Insetos nas Amostras

Muitas fotografias foram produzidas no decorrer do processo de monitoramento de pragas nas plantações de citros. Para a aquisição da primeira amostra, entretanto nem todos puderam ser utilizadas. Esse fato ocorreu devido muitas das fotografias não conterem os insetos de interesse para o estudo, provavelmente decorrente do ciclo sazonal dos insetos. Isso pode ser comprovado mediante os seguintes dados: das 494 fotografias da primeira amostra, foram aproveitadas apenas 93 fotografias, sendo assim, obteve-se um aproveitamento de apenas 18%.

Outro ponto que merece ênfase, é quanto a qualidade das fotografias, mais precisamente a degradação dos insetos que estavam nas armadilhas. Muito dos insetos já estavam em processo de decomposição, dessa forma, acabaram perdendo características morfológicas, por exemplo, a tonalidade verde do inseto *Triosa erytreae* desapareceu.

Como solução a esse obstáculo (número reduzido de fotografias e degradação dos insetos), coletou-se uma segunda amostra de fotografias, conforme foi mostrado na Sessão 4.1.2 (Insetos do Insetário), as quais, em comparação a gama de fotografias da primeira, é considerado menor, sendo de apenas 48. Entretanto, todas as fotos tiradas possuíam os insetos de interesse. Isso gerou um aproveitamento de 100%, não sendo necessário, desta maneira, fazer uma pré-seleção das mesmas.

Na Tabela 5.1 está disposto a distribuição do número de insetos para cada amostra respectivamente.

É possível constatar que na primeira amostra ocorre uma deficiência relativa ao número de algumas espécies insetos, como, *Chrysopidae*, *Philaenus* e *Syrphidae*, os quais somados,

Tabela 5.1: Insetos identificados nas amostras.

Insetos	**Primeira (93 fotografias)**	**Segunda (48 fotografias)**
Trioza erytreae	265	402
Coccinellidae	147	230
Chrysopidae	16	198
Philaenus	4	372
Syrphidae	5	0
Total:	**437**	**1202**

totalizam apenas 25 exemplos. Esse número é muito baixo se comparado, por exemplo, com o inseto *Trioza erytreae*, o qual possui 265 exemplos. Desta forma, é nítido que homogeneidade, necessária, relativa à quantidade dos insetos, não pode ser alcançada com a utilização apenas da primeira amostra.

Com a segunda amostra, foi possível resolver parcialmente os problemas apresentados, devido a estratégia de aquisição de fotografias das armadilhas com apenas os insetos de interesse. Por exemplo, o número total de insetos que compõem a segunda amostra é de 1202, muito maior que os 437 disposto na primeira.

Os insetos *Coccinellidae*, *Chrysopidae* e *Philaenus* tiveram um aumento bem significativo. Do total de 167 (primeira amostra) obteve-se um "salto"para 800 (segunda amostra). Porém, ainda nota-se a ausência do inseto *Syrphidae* e assim sendo, pode-se afirmar que o critério da homogeneidade entre os insetos foi somente parcialmente resolvido.

5.1.2 Diferentes Banco de Dados de Aprendizagem

O primeiro e o segundo BDA são formados mediante o uso da primeira e segunda amostra respectivamente. Já o terceiro é constituído pela junção de ambas. A escolha de fazer uso de diferentes BDAs tem como escopo poder realizar uma comparação entre eles. Desta forma, torna-se possível verificar se realmente existiu uma melhora no treinamento, assim como constatar se a identificação e classificações obteve níveis mais elevados de sucesso.

Nas Tabelas 5.2–5.4 é mostrado a relação de exemplos relativos ao treino e teste da distribuição dos insetos para os BDAs.

Essa divisão torna evidente a principal deficiência da primeira amostra Tabela 5.2, sendo que os *Philaenus* e *Syrphidae* tiveram, cada um, apenas um único exemplar para o teste.

Tabela 5.2: Disposição dos insetos do BDA 1.

	Banco de Dados de Aprendizagem 1 (93 fotografias)			
Insetos	**Classificados**	**Treino**	**Teste**	**% Teste**
Trioza erytreae	265	214	51	19,25
Coccinellidae	147	117	30	20,41
Chrysopidae	16	13	3	18,75
Philaenus	4	3	1	25,00
Syrphidae	5	4	1	20,00
Total:	**437**	**351**	**86**	**19,68**

Quanto ao segundo BDA Tabela 5.3, os insetos tiveram uma melhor divisão em treino e teste, excluindo o inseto *Syrphidae* da análise. Salienta-se que o tipo de inseto com menor taxa de exemplos para teste é o da família dos *Chrysopidae*, com 40 exemplares.

Tabela 5.3: Disposição dos insetos do BDA 2.

	Banco de Dados de Aprendizagem 2 (48 fotografias)			
Insetos	**Classificados**	**Treino**	**Teste**	**% Teste**
Trioza erytreae	402	321	81	20,15
Coccinellidae	230	186	44	19,13
Chrysopidae	198	158	40	20,20
Philaenus	372	304	68	18,28
Syrphidae	0	0	0	0,00
Total:	**1202**	**969**	**233**	**19,38**

O terceiro BDA Tabela 5.4, com base nos dados, é classificado como “o melhor entres dois mundos”, sendo composto dos dois primeiros BDAs, cada um com suas características. Apesar disso, o problema do inseto da família dos *Syrphidae* se manteve mediante a deficiência de exemplos.

Tabela 5.4: Disposição insetos do BDA 3.

	Banco de Dados de Aprendizagem 3 (141 fotografias)			
Insetos	**Classificados**	**Treino**	**Teste**	**% Teste**
Trioza erytreae	667	535	132	19,79
Coccinellidae	377	303	74	19,63
Chrysopidae	214	171	43	20,09
Philaenus	376	307	69	18,35
Syrphidae	5	4	1	20
Total:	**1639**	**1320**	**319**	**19,46**

Vale ressaltar que, durante o processo manual de identificação e classificação dos insetos, a utilização do software *Labelimg* foi um sucesso pois proporcionou uma organização clara nas divisões dos insetos para BDAs.

5.1.3 Métricas de Avaliação de Aprendizagem

Para um uma melhor compreensão, será realizada uma breve explicação a respeito dos conceitos métricos que serão posteriormente apresentados.

As avaliações de desempenho baseiam-se em alguns conceitos fundamentais, dos quais pode-se destacar a precisão (*precision*) que é responsável por medir de forma precisa as classificações, ou seja, a porcentagem de suas classificações que está correta, e a revogação (*recall*), que representa a frequência em que o seu classificador encontra os exemplos de uma classe. Precisão e Revogação podem ser definidas matematicamente Equações 5.1 e 5.2.

$$precisão = \frac{VP}{VP + FP} \tag{5.1}$$

$$revogação = \frac{VP}{VP + FN} \tag{5.2}$$

Em que, VP: verdadeiro positivo, FP: falso positivo e FN: falso negativo.

Os resultados que serão apresentados a seguir, foram coletados pela ferramenta *TensorBoard*, e as métricas utilizadas são amplamente adotadas por várias competições de reconhecimento e detecção de objetos [33], [55].

Por tanto, três métricas de avaliação de desempenho serão utilizados:

mAP (*Mean Average Precision*) é definida como a porcentagem média do valor médio de precisão, ela é baseada na curva de revogação-precisão como métrica de desempenho, quanto maior melhor.

AR@1 (*Mean Average Recall*) é definida como a porcentagem média do valor médio de revogação em todas as imagens com no máximo 1 detecção, quanto maior melhor.

Total Loss é definida como a função erro internamente realizada no processo de aprendizagem. O valor de *Loss* informa distância do valor da predição em relação ao valor esperado, portanto, quando menor, melhor.

5.1.4 Avaliação dos Processos de Aprendizagem

Os resultados do treinamento serão apresentados respeitando a ordem cronológicas dos treinos, ou seja, primeiro serão apresentados os resultados de treinamento do BDA 1, e assim por diante. Como descrito anteriormente, foram definidas duas meta-arquiteturas e dois extratores de características que, combinados entre si, resultaram em quatro configurações diferentes de treino. Portanto, cada um dos BDAs passa pelos quatro diferentes processos de aprendizagem. Desta forma, observa-se cada um gerou uma série de resultados diversos.

O primeiro resultado mostrado, diz respeito à possibilidade de realizar o processo de aprendizagem e o quão longe em ciclos (épocas) chegaram no treino.
Na Tabela 5.5 demonstra-se os 12 treinos com as suas respectivas configurações e valores de ciclos (épocas).

É possível observar que os treinos com a arquitetura SSD e o extrator de característica

Tabela 5.5: Épocas (ciclos) de cada treino.

Configurações			Aprendizagem	
BDA	**Meta-aquitetura**	**Extrator de características**	**Treino**	**Épocas**
BDA 1	*Faster* R-CNN	*Inception*-v2	*model*-1	30080
BDA 1	*Faster* R-CNN	*ResNet*-50	*model*-2	5057
BDA 1	SSD	*Inception*-v2	*model*-3	10760
BDA 1	SSD	*ResNet*-50	*model*-4	0
BDA 2	*Faster* R-CNN	*Inception*-v2	*model*-5	31135
BDA 2	*Faster* R-CNN	*ResNet*-50	*model*-6	5280
BDA 2	SSD	*Inception*-v2	*model*-7	10417
BDA 2	SSD	*ResNet*-50	*model*-8	0
BDA 3	*Faster* R-CNN	*Inception*-v2	*model*-9	31660
BDA 3	*Faster* R-CNN	*ResNet*-50	*model*-10	4848
BDA 3	SSD	*Inception*-v2	*model*-11	10050
BDA 3	SSD	*ResNet*-50	*model*-12	0

ResNet-50 falharam no processo de treinamento, e por isso o valor de épocas é 0, na API *Object Detection* foi mostrando a seguinte mensagem de erro "`OP_REQUIRES failed at cwise_ops_common.cc:82 : Resource exhausted: OOM when allocating tensor with shape[104857600] and type float on /job:localhost/replica:0/task:0/device:CPU:0 by allocator mklcpu`". Esse erro ocorre devido as capacidades limitadas do hardware, o qual não conseguiu nem ao menos carregar a grande arquitetura que é a SSD com a *ResNet*-50.

Os treinos para os três BDAs foram encerrados seguindo um certo padrão de épocas, por exemplo, *Faster* R-CNN com *Inception*-v2 tiveram em torno de 30 mil ciclos, já o *Faster* R-CNN *ResNet*-50 tiveram 5 mil ciclos e SSD *Inception*-v2 com 10 mil ciclos. Estes valores foram definidos quando ocorreu estagnação da métrica *Total Loss* durante o processo de treinamento.

Resultados de treino para o BDA 1

O BDA 1 apresentou os seguintes resultados e desempenho, mostrado nas Figuras 5.1 – 5.3. O *model*-1 alcançou 0,0142 de mAP, 0,0146 de AR@1 e 0,3203 de *Total Loss*, na

a configuração de treinamento *Faster* R-CNN *Incepition*-v2. O *model*-2 alcançou 0,0124 de mAP, 0,0086 de AR@1 e 0,2685 de *Total Loss*, na configuração de treinamento *Faster* R-CNN *ResNet*-50. E por último, o *model*-3 alcançou 0,0360 de mAP, 0,0366 de AR@1 e 18.9140 de *Total Loss*, na a configuração de treinamento SSD *Inception*-v2.

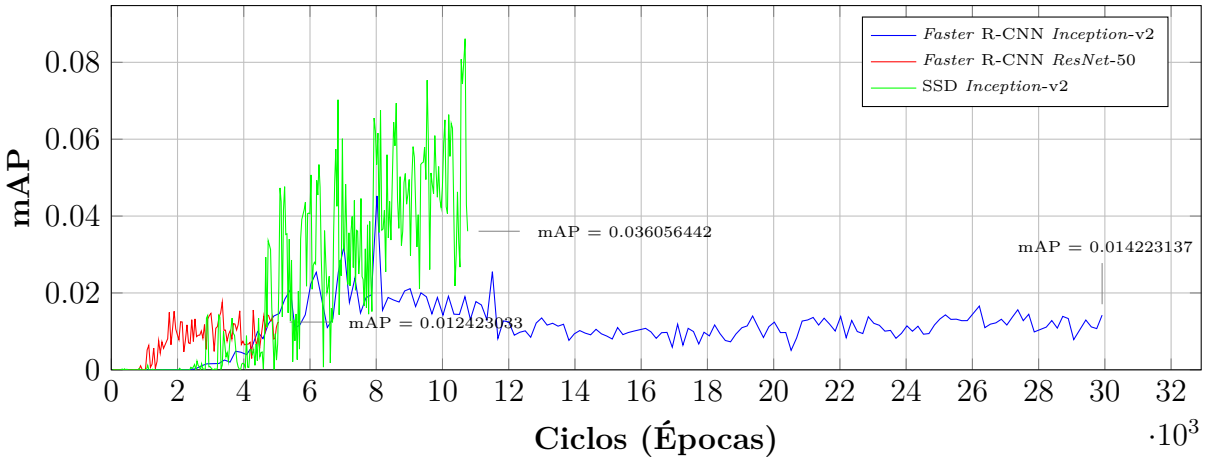


Figura 5.1: mAP em diferentes arquiteturas para BDA 1.

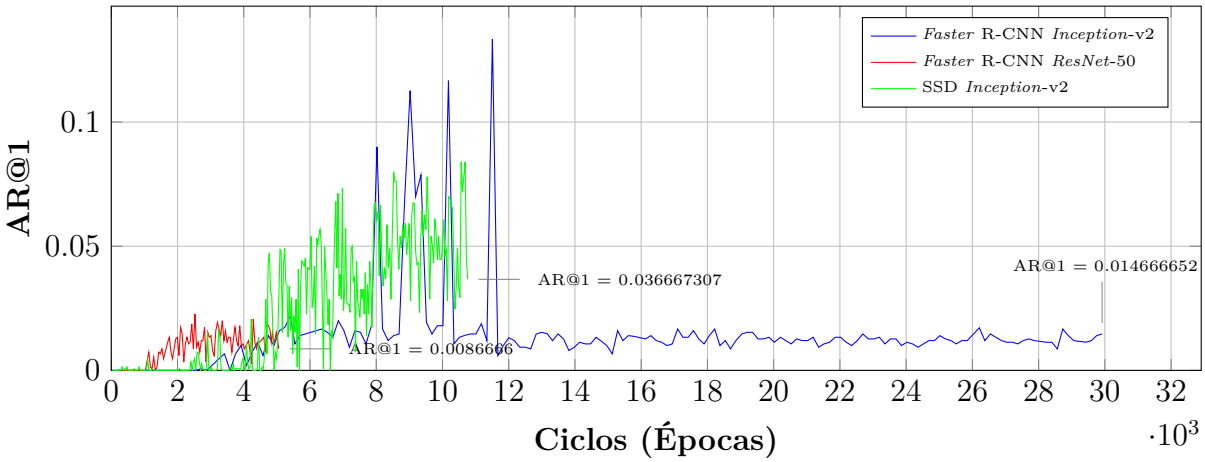


Figura 5.2: AR@1 em diferentes arquiteturas para BDA 1.

Os treinos *model*-1 e *model*-2, ambos com *Faster* R-CNN tiveram resultados semelhantes nas métricas de avaliação: diferença de 0,0018 mAP e 0,052 de *Total Loss*. Vale ressaltar que o *model*-2 que usava o extrator de característica *ResNet*-50 alcançou os valores das métricas com um menor número de ciclos.

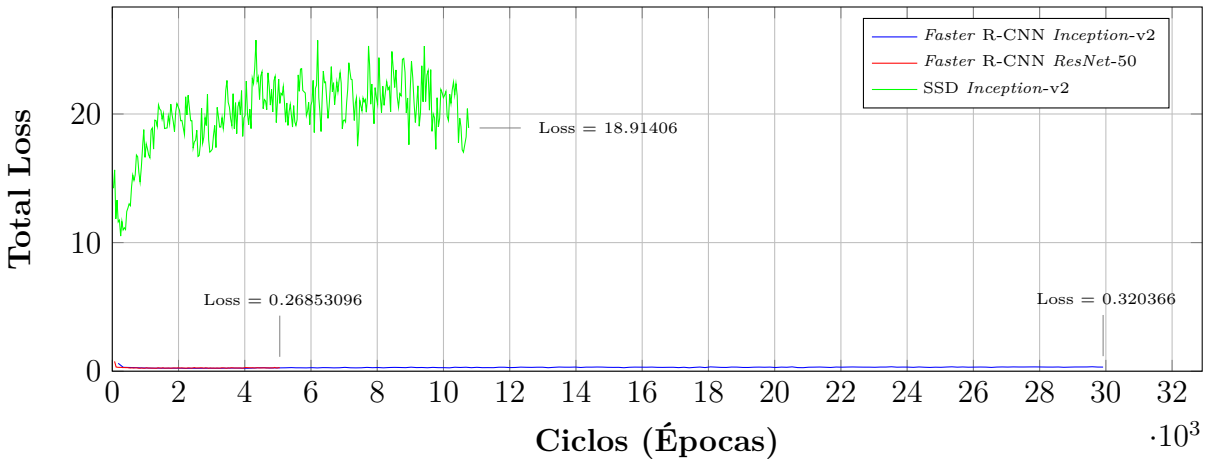


Figura 5.3: *Total Loss* em diferentes arquiteturas para BDA 1.

No *model*-3, percebe-se uma grande diferença dos resultados, tanto o valor de 0,0360 mAP teve um aumento de 170,6% em relação a média dos valores mAP do *model*-1 e *model*-2, que foram de 0,0133 mAP, quanto o valor de 18,9140 *Total Loss* teve um aumento gigantesco de 6287.8% comparados à média dos resultados dos treinos anteriores, a qual foi de 0,2960 *Total Loss*.

Resultados de treino para o BDA 2

Para o BDA 2, as Figuras 5.4 – 5.6, mostra o desempenho e o resultado dos treinos. O *model*-5 alcançou 0,3437 de mAP, 0.0348 de AR@1 e 1.2603 de *Total Loss*, na a configuração de treinamento *Faster* R-CNN *Incepition*-v2. O *model*-6 alcançou 0.3202 de mAP, 0.0359 de AR@1 e 1.2506 de *Total Loss*, na a configuração de treinamento *Faster* R-CNN *ResNet*-50. E por último, o *model*-7 alcançou 0.2602 de mAP, 0.0319 de AR@1 e 5,9406 de *Total Loss*, na a configuração de treinamento SSD *Inception*-v2.

De um modo geral, os resultados dos treinos que usaram o BDA 2 obtiveram um melhor desempenho quando comparados aos treinos do BDA 1. Por exemplo, a mAP na configuração *Faster* R-CNN *Incepition*-v2, teve uma melhora significativa (24 vezes superior), saltando de 0.01422 mAP para 0.3437 mAP.

Constata-se também que o comportamento das métricas em função dos ciclos ocorreu de forma mais constante, ou seja, não averiguou-se grandes saltos (picos-vales) durantes

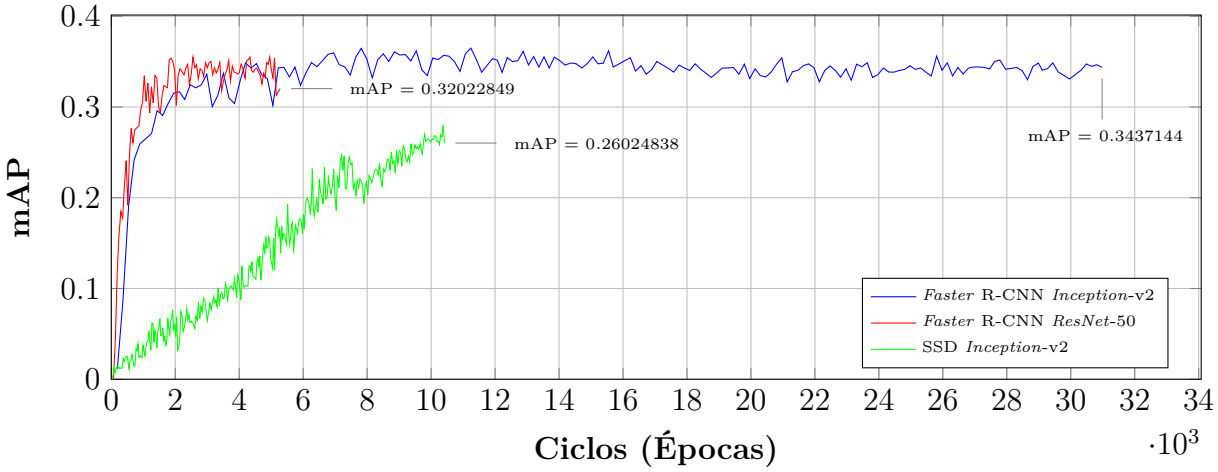


Figura 5.4: mAP em diferentes arquiteturas para BDA 2.

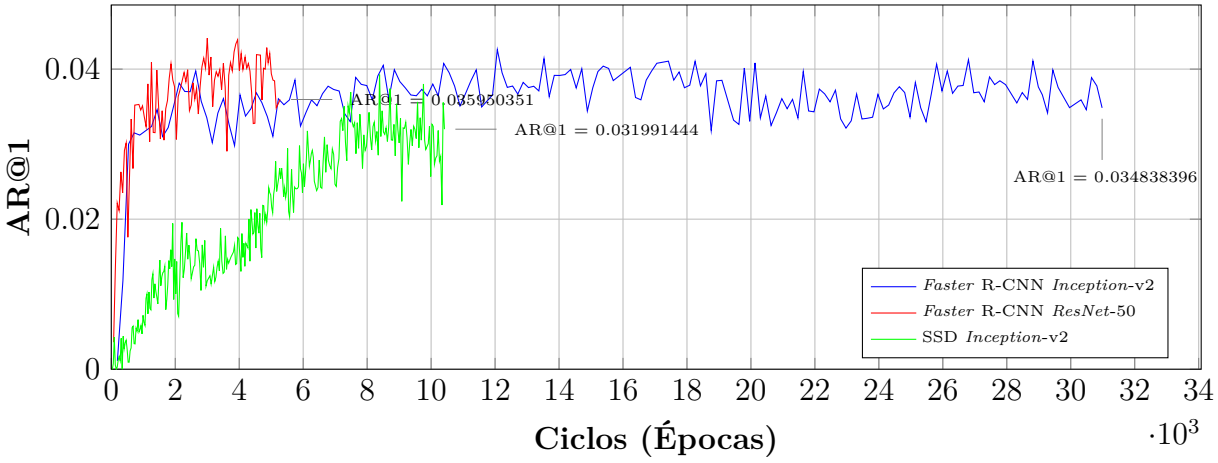


Figura 5.5: AR@1 em diferentes arquiteturas para BDA 2.

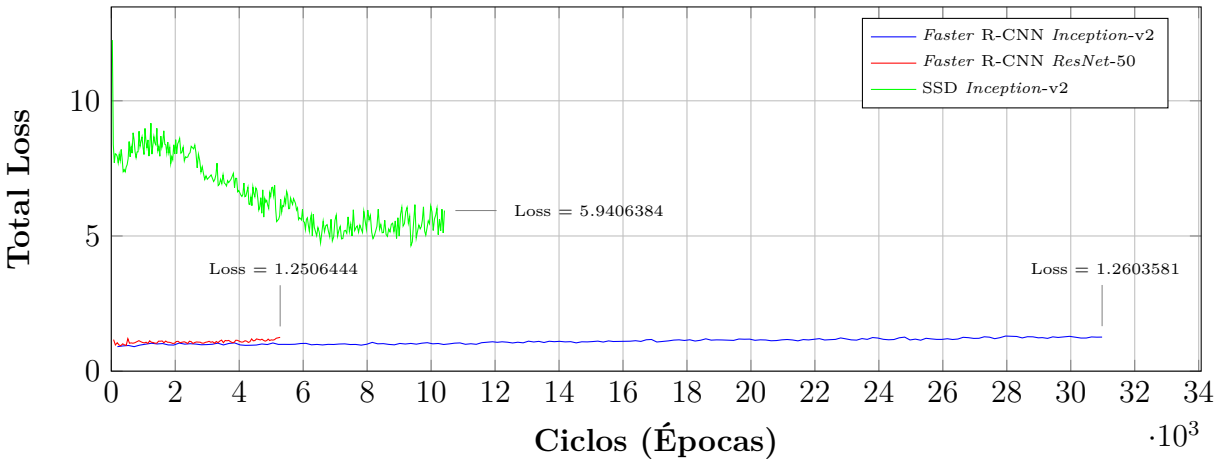


Figura 5.6: *Total Loss* em diferentes arquiteturas para BDA 2.

os ciclos em comparação com o BDA 1.

Resultados de treino para o BDA 3

Por último, BDA 3 (junção dos dois primeiros BDAs), corresponde as Figuras 5.7 – 5.9, as quais mostram o desempenho e o resultado dos treinos.

O *model*-9 alcançou 0,2251 de mAP, 0,0282 de AR@1 e 0,5920 de *Total Loss*, na a configuração de treinamento *Faster* R-CNN *Incepition*-v2. O *model*-10 alcançou 0,2129 de mAP, 0,0290 de AR@1 e 0,6049 de *Total Loss*, na a configuração de treinamento *Faster* R-CNN *ResNet*-50. E por último, o *model*-11 alcançou 0,1985 de mAP, 0,0274 de AR@1 e 13,0055 de *Total Loss*, na a configuração de treinamento SSD *Inception*-v2.

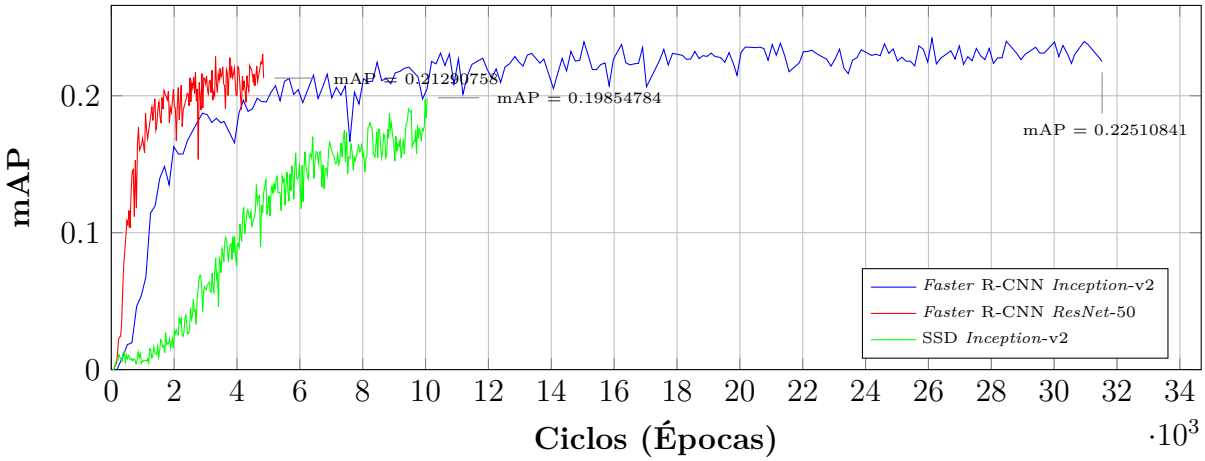


Figura 5.7: mAP em diferentes arquiteturas para BDA 3.

Ao comparar o maior valor de mAP entre as arquiteturas do primeiro BDA, o qual correspondia a 0,0360 na configuração de treino SSD *Inception*-v2 e o maior valor mAP do BDA 3 de 0,2251 na configuração de treino *Faster* R-CNN *Incepition*-v2, constata-se um aumento de 6 vezes relacionado à precisão.

Quando realiza-se a análise do *Total Loss*, os valores nas configurações dos *Faster* R-CNN *Incepition*-v2 e *Faster* R-CNN *ResNet*-50 constata-se uma piora, tendo o valor da média aumentado de 0,2944 para 0,5985. Porém, quando uma comparação é feita entre os resultados do BDA 1 com o BDA 3 da métrica de *Total Loss* na configuração de treino SSD *Inception*-v2 observa-se uma melhora de 45,4%, sendo esta de 18,91 para 13,005.

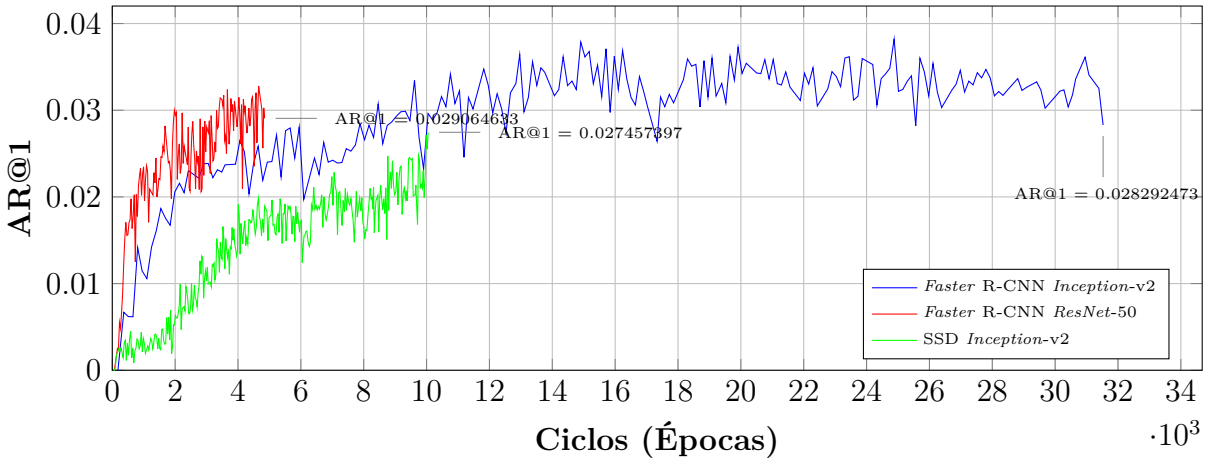


Figura 5.8: AR@1 em diferentes arquiteturas para BDA 3.

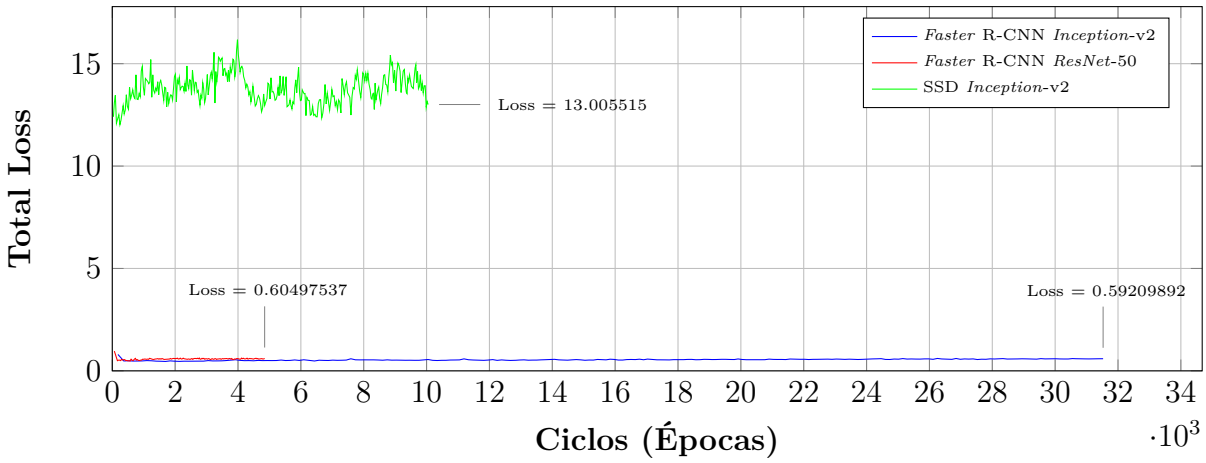


Figura 5.9: *Total Loss* em diferentes arquiteturas para BDA 3.

5.1.5 Comparação Quantitativa dos Processos de Aprendizagem

Para uma visualização clara dos valores métricos dos treinos, a Tabela 5.6 apresenta a disposição desses valores em função das arquiteturas e BDA. Dentre os dados dispostos, a comparação dos resultados obtidos com os treinos que foram realizados com o BDA 1 com os do BDA 3, objetiva validar a o pressuposto inicial, que estabelecia que a inclusão da segunda amostra (BDA 2) na primeira amostra (BDA 1), traria melhoras consideráveis nos modelos treinados.

Tabela 5.6: Valores métricos dos treinos em função das arquiteturas e BDA.

Arquiteturas	BDA	modelos	Ciclos	mAP	AR@1	*Total Loss*
Faster R-CNN com *Inception-v2*	BDA 1	*model*-1	30080	0,0142	0,0146	0,3203
	BDA 2	*model*-5	31135	0,3437	0,0348	1,2603
	BDA 3	*model*-9	31660	0,2251	0,0282	0,592
Faster R-CNN com *ResNet*-50	BDA 1	*model*-2	5057	0,0124	0,0086	0,2685
	BDA 2	*model*-6	5280	0,3202	0,0359	1,2506
	BDA 3	*model*-10	4848	0,2129	0,029	0,6049
SSD com *Inception*-v2	BDA 1	*model*-3	10760	0,036	0,0366	18,914
	BDA 2	*model*-7	10417	0,2602	0,0319	5,9406
	BDA 3	*model*-11	10050	0,1985	0,0274	13,0055

Ao observar o comportamento das médias do valor mAP e AR@1 para todos os BDAs, é possível constatar que a precisão e revogação são as variáveis dependentes, relacionadas com o valor de ciclos (épocas). Desta forma, naturalmente, diz-se que quando maior o número de ciclos, maior será a precisão e revogação do modelo treinado.

Mediante o exposto, é possível afirmar que o valor de média do valor médio de precisão (mAP) e o valor média do valor médio de revogação (AR@1) de todas as comparações pressupostas, isto é, *model*-1 com *model*-9, *model*-2 com *model*-10 e *model*-3 com *model*-11, nitidamente melhoraram significativa. Por exemplo, *model*-1 com 0,0142 mAP e 0,0146 AR@1 e *model*-9 com 0,2251 mAP e 0,0282 AR@1, progrediram 58% em relação ao mAP e 93%, ao AR@1.

Portanto, é evidente que a inclusão da segunda amostra na primeira amostra (para compor o BDA 3) culminou em melhores resultados relativos à precisão da identificação e classificação dos insetos.

Diante disso, os *model*-9, 10 e 11 foram escolhidos para constituírem o aplicativo *Insect Detection*. Essa opção, deve-se ao fato terem sido treinados com o BDA 3 e assim, como demonstrado acima, os que obtiveram os resultados mais promissores.

5.2 Resultados do *Insect Detection*

O aplicativo *Insect Detection* com seu *layout* e as suas funcionalidades, foram anteriormente apresentados na Sessão 4.4.2 do Capítulo 4. Nesse momento, aborda-se os resultados e discussões das capacidades e performance de aplicativo desenvolvido. Em outras palavras, se ele é capaz de identificar, classificar e contabilizar os insetos a partir de fotografias de armadilhas autocolantes das plantações de citros, bem como a de gerar um banco de dados de identificação.

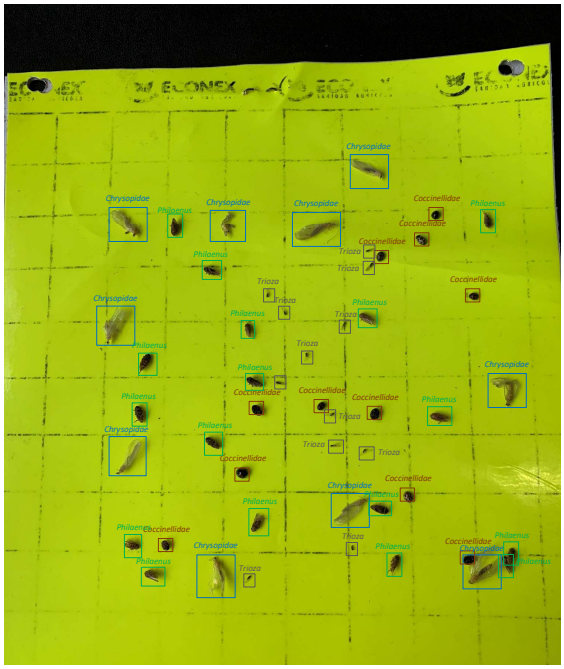
E após essa verificação, analisa-se a viabilidade da a substituição do especialista em insetos pela aplicação *Insect Detection*.

5.2.1 Identificação e Contabilização dos Insetos

Para avaliar o desempenho na tarefa de identificação, classificação e contagem dos insetos, foram selecionadas quatro imagens para teste, as quais passaram por um processo manual de identificação e classificação, com retângulos coloridos indicando os tipos insetos, como demonstrado na Figura 5.10. Essas não constituem nenhum dos BDAs, assim tem-se a garantia da capacidade de generalização do modelo treinado, já que não compuseram o processo de aprendizagem.

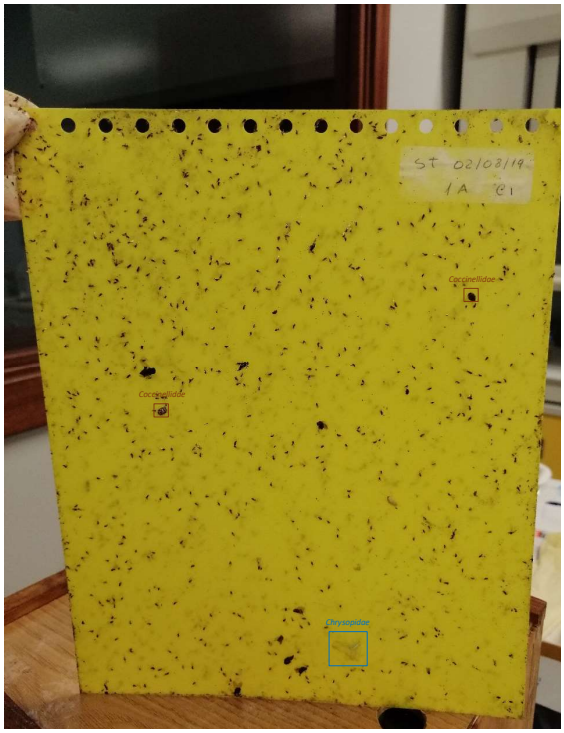
As imagens usadas para o teste estão disponíveis para download no repositório *Gitlab*, assim como o link de *download* está disponível no Apêndice C.

A primeira imagem de teste ilustrada na Figura 5.10a consiste numa fotografia criada nos mesmos moldes daquelas que fazia parte da amostra do insetário. Dessa forma, elas possuem uma grande variedade do insetos de interesse, possuem 12 *Triozas erytreae*, 11 *Coccinellidae*, 8 *Chrysopidae* e 17 *Philaenuse*.



(a) Fotografia da armadilha do insetário.

(b) Fotografia da armadilha 1 da Plantação.



(c) Fotografia da armadilha 2 da Plantação.

(d) Fotografia da armadilha 3 da Plantação.

Figura 5.10: Imagens selecionadas para teste.

A segunda, terceira e quarta imagens ilustrada nas Figuras 5.10b à 5.10d são fotografias das armadinhas das plantações de citros. Assim, representam um teste prático de identificação, classificação e contagem de insetos. Entretanto, o número real de cada inseto dessas últimas fotografias não é exato, isso, em virtude, principalmente, do número elevado de *Triozas erytreae*, sendo que cada imagem possui mais de 150 exemplares (não foram feitos retângulos coloridos indicando as *Triozas* devido ao número elevado), e por isso é dado a importância a outros insetos. Por exemplo, na Figura 5.10b, tem-se 1 *Coccinellidae*. Na Figura 5.10c possui 2 *Coccinellidae* e 1 *Chrysopidae*. E por último, a Figura 5.10b tem 1 *Chrysopidae*.

Desse modo, cada uma das imagem foram processadas no aplicativo *Insect Detection* usando os *model*-9, 10 e 11, desempenhando a tarefa de identificar, classificar e contar o número de insetos. A aplicação pode facilmente alterar o modelo para realizar a inferência nas imagens, ou seja, troca rápida de modelos para teste.

Assim sendo, a Tabela 5.7 apresenta os resultados desse processamento, demonstrando os números e porcentagem de identificação de cada inseto nas imagens de teste em função dos modelos.

Tabela 5.7: Resultados dos *model*-9, 10 e 11 para as imagens de teste.

Imagens de teste	**Modelos**	***Triozas erytreae***		***Coccinellidae***		***Chrysopidae***		***Philaenus***		**Precisão média**
Fotografia do insetário	manual	12	-	11	-	8	-	17	-	-
	model-9	0	0%	3	27,27%	8	100%	9	52,94%	45,05%
	model-10	0	0%	7	63,64%	8	100%	5	55,56%	54,80%
	model-11	2	16,67%	4	36,36%	6	75%	8	47,06%	43,77%
Fotografia da armadilha 1	manual	+100	-	1		0	-	0	-	-
	model-9	0	0%	1	100%	1	0%	1	0%	25%
	model-10	0	0%	1	100%	0	0%	0	0%	25%
	model-11	0	0%	1	100%	0	0%	0	0%	25%
Fotografia da armadilha 2	manual	+100	-	2	-	1	-	0	-	-
	model-9	0	0%	1	50%	0	0%	0	0%	12,50%
	model-10	0	0%	2	100%	1	100%	0	0%	50%
	model-11	0	0%	0	0%	0	0%	0	0%	0%
Fotografia da armadilha 3	manual	+100	-	0	-	1	-	0	-	-
	model-9	0	0%	1	0%	1	100%	0	0%	25%
	model-10	0	0%	0	0%	0	0%	0	0%	0%
	model-11	0	0%	1	0%	0	0%	0	0%	0%

Analisando os resultados, de um modo geral, pode-se dizer que a precisão média ficou em torno de 25,51%. Este têm muita semelhança com valores métricos da média do valor médio de precisão (mAP) encontrados na etapa de treinamento dos modelos.

Entretendo, analisando individualmente alguns resultados, aponta-se alguns dados relevantes, como: em todos os modelos a performance de identificação dos insetos *Coccinellidae* (53,05% de precisão média), *Chrysopidae* (52% de precisão média) e *Philaenus* (51,85% de precisão média) foram o que tiveram melhores resultados.

Em contrapartida, o inseto *Triozas erytreae* foi o que apresentou os piores resultados, com apenas 1,38% de precisão média em todos os modelos e todas as imagens de teste, e apenas pontuando no *model*-11 na primeira imagem de teste com 16,67% de precisão.

É possível apontar alguns fatores que podem ser a causa dessa situação indesejada, tais como: a má qualidade das fotografias (quanto a iluminação, desfoque na imagem, perda do espectro de cores, degradação e posicionamento dos insetos) inclusas no BDA e nas imagens de teste, o baixo número de exemplos de insetos no BDA, número baixo de ciclos (épocas) de treino, a escolha da estagnação da métrica de *Total Loss* como critério de finalização de treino, ou até a própria limitação das arquiteturas de RNCP (*Faster* R-CNN) usadas nesse trabalho, que não funciona bem com pequenos insetos (*Triozas erytreae*).

Um ponto que merece destaque, graças ao bom desempenho apresentado é o relativo à análise da meta-arquitetura SSD com o extrator de característica *Inception*-v2. Esta obteve melhores resultados na identificação de insetos de pequenos quando comparada às outras arquiteturas analisadas.

Como os resultados encontrados foram muito abaixo do esperado, à primeira instância, não é viável a substituição de um especialista capacitado pela aplicação na tarefa de identificação, classificação e contagens dos insetos. Mas, como a aplicação *Insect Deteccion* pode facilmente alterar o modelo para realizar a identificação, classificação e contagem nos insetos nas imagens, consequentemente, o modelo pode ser aprimorado e importado na aplicação, de modo a ir melhorando a precisão.

Após esta explanação, para melhorar a precisão dos modelos, é preciso realizar todos os treinos novamente, porém levando em conta todos os apontamentos realizados acima. Em outras palavras, melhorar o BDA e treiná-las usando a arquitetura SSD ou outras mais modernas, como por exemplo *Mask* R-CNN, as quais, atualmente, estão obtendo

ótimos resultados de precisão (COCO mAP) [56].

5.2.2 Banco de dados de identificação

O aplicativo *Insect Detection* tem como função desempenhar as tarefas de criação de um banco de dados de identificação e exibir um histórico das últimas identificações. Quanto a isso, o aplicativo foi um sucesso e isso ocorreu devido à atualização da biblioteca *Pandas*, a qual inseriu o recurso de *dataframe* (uma estrutura de dados), recurso de muita importância nas nas análises de grandes dados.

Após selecionadas uma série de imagens para serem processadas, como foi mostrado na Figura 4.8, é importante salientar a grande relevância que os nomes dos ficheiros das imagens devem conter as informações de data e localização (*C1.ST.1B_02.07.2019.jpg*).

Nota-se que o aplicativo foi capaz de processá-las em sua totalidade e realizou também a exibição das informações em uma janela contendo: data, localização e número de cada inseto identificado.

Caso o usuário queira, é possível salvar os dados processados, para isso, é necessário clicar no botão "*Exported Data*". Assim, os dados são salvos em um arquivo *csv*, chamado de *imagesinsectdata.csv*. O resultado visual desse banco de dados de monitoramento pode ser observado na Tabela 5.8.

Tabela 5.8: Banco de dados (*imagesinsectdata.csv*).

```
date,location,trioza_erytreae,syrphidae,coccinellidae,chrysopidae,philaenus
02.07.2019,C1.ST.1B,18.0,0.0,2.0,0.0,0.0
02.07.2019,C1.ST.2B,0.0,0.0,1.0,0.0,0.0
19.12.2019,IPB.YT.00,2.0,0.0,4.0,6.0,8.0
02.07.2019,V.ST.4A,0.0,0.0,2.0,0.0,0.0
```

Capítulo 6

Conclusões

O desenvolvimento deste trabalho proporcionou a criação de uma ferramenta automática para o monitoramento de insetos, em especial, aqueles que têm a capacidade de ocasionar a doença *huanglongbing* (*Citros Greening*) em culturas de citros.

A solução foi proporcionada mediante o desenvolvimento da aplicação chamada de *Insect Detection*. Esta tem como base meios de aprendizagem profunda, mais precisamente de RNCPs nas arquiteturas *Faster* R-CNN e SSD com os extratores de características *Inception*-v2 e *ResNet*-50.

Para o processo de identificação e classificação dos insetos, utilizou-se de fotografias das armadilhas dispostas na plantação localizada em Vairão (Portugal). Os dados obtidos dessa coleta foram registados em uma base da dados de monitoramento, os quais contém informações relativas à data, localização e número de insetos que foram identificados.

Portanto, o trabalho teve como propósito criar uma ferramenta que tivesse a capacidade de solucionar os problemas existentes no método tradicional de monitoramento de pragas e para isso, obter maior velocidade, uma organização mais satisfatória nos dados de monitoramento e também deixar de exigir a presença de um especialista em insetos para realizar manualmente a identificação e classificação. Porém, desses propósitos alguns aspectos não foram alcançados.

Durante a etapa de coleta de imagens para formar o BDA surgiram os primeiros desafios. O baixo número de alguns exemplos e a degradação de insetos foram um grande

obstáculo a ser superado, pois este fato pode prejudicar, e muito, os resultados durante a etapa de treinamento da RNCP. Como solução implementada, criou-se amostras a partir de insetos selecionados para complementar a base original, e assim, estabelecer uma melhora dos resultados de treinamento.

De um modo geral, as RNCPs são ótimas para identificar e classificar objetos e o presente trabalho teve como objetivo analisá-las na prática. Para isso, verificou-se o desempenho em diferentes arquiteturas de rede a identificação e classificação de insetos e o resultado obtido demonstrou claramente a dificuldade de se identificar insetos muito pequenos (*Trioza erytreae*) localizados entre insetos maiores como o (*Coccinellidae*, o *Chrysopidae* e o *Philaenus*).

Com os teste de identificação e classificação, obtive-se os seguintes resultados de precisão para cada inseto: 1,38% de *Trioza erytreae*, 53,05% de *Coccinellidae*, 52% de *Chrysopidae* e 51,85% de *Philaenus*. E precisão média geral de aproximadamente 25,51%. Assim sendo, à primeira instância, não se mostrou viável a substituição de especialista na tarefa de identificar, classificar e contar os insetos pela aplicação, devido ao valor muito baixo.

Diversos fatores plausíveis podem ocasionar esses resultados negativos, tais como: o baixo número de exemplos de insetos, a má qualidade das fotografias (degradação dos insetos) inclusas no BDAs, número baixo de ciclos (épocas) de treino (devido ao tempo necessário) ou até a própria limitação das arquiteturas de RNCPs usadas nesse trabalho.

Destaca-se no trabalho a contribuição do estudo do desempenho das diferentes arquiteturas moderas de RNCPs para realizar identificação e classificação dos insetos. Sobretudo, neste ponto, cabe enfatizar a meta-arquitetura SSD pois esta apresentou resultados promissores (os melhores desta pesquisa), principalmente na identificação de insetos de pequeno tamanho. Salienta-se também a grande importância das estratégias de criação de BDA para a realização do treinamento das RNCPs.

Diante do exposto, é nítido que este trabalho representa apenas o começo de uma linha de investigação com grande potencial. Deixando registrado para trabalhos futuros, diversas contribuições. Com os estudos realizados é possível determinar que alguns ajustes devem obter resultados mais eficientes e assim viabilizar a aplicação *Insect Detection* para

o uso em monitoramento de pragas de citros.

Este é o caso do BDA, o qual pode ser aperfeiçoado de diversas maneiras, como maior número de fotografias, melhor qualidade (insetos menos degradados), maior número de exemplos de insetos. Outro ajuste é o referente ao aprimoramento de estratégias de otimização (*data augmentation*, pré-processamento), assim como aumentar o número de ciclos (épocas) dos treinos.

A aplicação *Insect Deteccion* pode facilmente alterar o modelo, o qual realizar a tarefa de identificação, classificação e contagem nos insetos nas imagens. Consequentemente, o modelo pode ser aprimorado e importado, assim sendo, torna-se interessante estudar e verificar o desempenho de identificação e classificação de insetos em outras arquiteturas de rede, como por exemplo, a *Mask* R-CNN, a qual, atualmente, estão obtendo ótimos resultados na identificação de objetos [56].

Cabe também para um trabalho futuro, a transferência da aplicação para um micro-computador *Raspberry pi*, com módulos de comunicação *Global System for Mobile Communications* (GSM) e câmeras. O qual será instalado diretamente na armadilha, deste modo, tornando-se possível fazer uma previsão da dinâmica das populações dos insetos de acordo com os dados em tempo real.

Bibliografia

[1] B. P. Santos e A. Alberto, "Indústria 4.0: desafios e oportunidades", pp. 111–124, 2018.

[2] G. E. Cocuzza, U. Alberto, E. Hernández-Suárez, F. Siverio, S. Di Silvestro, A. Tena e C. Rapisarda, "A review on Trioza erytreae (African citrus psyllid), now in mainland Europe, and its potential risk as vector of huanglongbing (HLB) in citrus", *Journal of Pest Science*, vol. 90, n.º 1, 2017, ISSN: 16124758. DOI: 10.1007/s10340-016-0804-1.

[3] M. Valan, K. Makonyi, A. Maki, D. Vondráček e F. Ronquist, "Automated Taxonomic Identification of Insects with Expert-Level Accuracy Using Effective Feature Transfer from Convolutional Networks", *Systematic Biology*, vol. 68, n.º 6, pp. 876–895, nov. de 2019, ISSN: 1076836X. DOI: 10.1093/sysbio/syz014.

[4] Z. Liu, J. Gao, G. Yang, H. Zhang e Y. He, "Localization and Classification of Paddy Field Pests using a Saliency Map and Deep Convolutional Neural Network", *Scientific Reports*, vol. 6, fev. de 2016, ISSN: 20452322. DOI: 10.1038/srep20410.

[5] S. Lim, S. Kim e D. Kim, "Performance Effect Analysis for Insect Classification using Convolutional Neural Network", *Proceedings - 7th IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2017*, vol. 2017-Novem, n.º November, pp. 210–215, 2018. DOI: 10.1109/ICCSCE.2017.8284406.

[6] S. Ren, K. He, R. Girshick e J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", *IEEE Transactions on Pattern Analysis*

and Machine Intelligence, vol. 39, n.º 6, pp. 1137–1149, 2017, ISSN: 01628828. DOI: 10.1109/TPAMI.2016.2577031. arXiv: 1506.01497.

[7] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu e A. C. Berg, "SSD: Single Shot MultiBox Detector", dez. de 2015. DOI: 10.1007/978-3-319-46448-0_2. arXiv: 1512.02325.

[8] Z. Zou, Z. Shi, Y. Guo e J. Ye, "Object Detection in 20 Years: A Survey", pp. 1–39, 2019. arXiv: 1905.05055.

[9] Y. Zhong, J. Gao, Q. Lei e Y. Zhou, "A vision-based counting and recognition system for flying insects in intelligent agriculture", *Sensors (Switzerland)*, vol. 18, n.º 5, pp. 1–38, 2018, ISSN: 14248220. DOI: 10.3390/s18051489.

[10] J. G. Ribeiro, D. Y. Marinho e J. W. M. Espinosa, "Agricultura 4.0: Desafios À Produção De Alimentos E Inovações Tecnológicas", *Simpósio De Engenharia De Produção - Sienpro*, pp. 1–7, 2018.

[11] M. Martineau, D. Conte, R. Raveaux, I. Arnault, D. Munier, G. Venturini, M. Martineau, D. Conte, R. Raveaux, I. Arnault e D. Munier, "A survey on image-based insect classification To cite this version : HAL Id : hal-01441203", 2017.

[12] A. Krizhevsky, I. Sutskever e G. E. Hinton, "ImageNet classification with deep convolutional neural networks", *Communications of the ACM*, vol. 60, n.º 6, pp. 84–90, jun. de 2017, ISSN: 15577317. DOI: 10.1145/3065386.

[13] K. Simonyan e A. Zisserman, "Very deep convolutional networks for large-scale image recognition", em *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, International Conference on Learning Representations, ICLR, 2015. arXiv: 1409.1556.

[14] J. Redmon, S. Divvala, R. Girshick e A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", jun. de 2015. arXiv: 1506.02640.

[15] T. S. Huang, "Computer Vision: Evolution and Promise", *Report*, 1997.

[16] L. G. Roberts, "Machine perception of three-dimensional solids", n.º January 1963, 1963. URL: http://dspace.mit.edu/handle/1721.1/11589.

[17] D. H. Hubel e T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex", *The Journal of Physiology*, vol. 160, n.º 1, pp. 106–154, 1962, ISSN: 14697793. DOI: 10.1113/jphysiol.1962.sp006837.

[18] S. Haykin, *Neural Networks and Learning Machines, 3/E*. Pearson Education India, 2010.

[19] X. Feng, Y. Jiang, X. Yang, M. Du e X. Li, *Computer vision algorithms and hardware implementations: A survey*, nov. de 2019. DOI: 10.1016/j.vlsi.2019.07.005.

[20] S. J. Russell e P. Norvig, *Artificial Intelligence A Modern Approach; PearsonEducation*. 2003, p. 1151, ISBN: 9780136042594. DOI: 10.1017/S0269888900007724. arXiv: 9809069v1 [arXiv:gr-qc].

[21] V. Rajaraman, "JohnMcCarthy - Father of artificial intelligence", *Resonance*, vol. 19, n.º 3, pp. 198–207, abr. de 2014, ISSN: 0973712X. DOI: 10.1007/s12045-014-0027-9.

[22] G. F. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Addison-Wesley, 2005, ISBN: 9780321263186.

[23] A. P. Sage, *Concise Encyclopedia of Information Processing in Systems & [and] Organizations*. Pergamon press New York, NY, 1990.

[24] S. Shalev-Shwartz e S. Ben-David, *Understanding machine learning: From theory to algorithms*. 2013, vol. 9781107057, pp. 1–397, ISBN: 9781107298019. DOI: 10.1017/CBO9781107298019.

[25] A. d. P. Braga, A. C. P. d. L. F. Carvalho e T. B. Ludermir, *Redes Neurais Artificiais : Teoria e Aplicações*. LTC Editora, 2007, p. 226, ISBN: 9788521615644.

[26] W. S. McCulloch e W. Pitts, "A logical calculus of the ideas immanent in nervous activity", *The Bulletin of Mathematical Biophysics*, vol. 5, n.º 4, pp. 115–133, dez. de 1943, ISSN: 00074985. DOI: 10.1007/BF02478259.

[27] Y. Lecun, Y. Bengio e G. Hinton, "Deep learning", *Nature*, vol. 521, n.º 7553, pp. 436–444, 2015, ISSN: 14764687. DOI: 10.1038/nature14539.

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

[29] *Understand the Softmax Function in Minutes - Data Science Bootcamp - Medium*. URL: https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d (acedido em 23/03/2020).

[30] L. Yann e B. Yoshua, "Convolutional Networks for Images, Speech, and Time-Series", vol. 4, n.º April 2016, pp. 2571–2575, 1995, ISSN: 1098-7576. DOI: 10.1109/IJCNN.2004.1381049. arXiv: arXiv:1011.1669v3.

[31] *Coding Deep Learning for Beginners — Linear Regression (Part 3): Training with Gradient Descent*. URL: https://towardsdatascience.com/coding-deep-learning-for-beginners-linear-regression-gradient-descent-fcd5e0fc077d (acedido em 07/04/2020).

[32] J. Yosinski, J. Clune, Y. Bengio e H. Lipson, "How transferable are features in deep neural networks?", *Advances in Neural Information Processing Systems*, vol. 4, n.º January, pp. 3320–3328, 2014, ISSN: 10495258. arXiv: 1411.1792.

[33] T. Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár e C. L. Zitnick, "Microsoft COCO: Common objects in context", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8693 LNCS, n.º PART 5, pp. 740–755, 2014, ISSN: 16113349. DOI: 10.1007/978-3-319-10602-1_48. arXiv: 1405.0312.

[34] R. Girshick, "Fast R-CNN", *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, pp. 1440–1448, 2015, ISSN: 15505499. DOI: 10.1109/ICCV.2015.169. arXiv: 1504.08083.

[35] Z. Deng, H. Sun, S. Zhou, J. Zhao, L. Lei e H. Zou, "Multi-scale object detection in remote sensing imagery with convolutional neural networks", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 145, pp. 3–22, 2018, ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2018.04.003.

[36] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama e K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors", *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-January, pp. 3296–3305, 2017. DOI: 10.1109/CVPR.2017.351. arXiv: 1611.10012.

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke e A. Rabinovich, "Going deeper with convolutions", em *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, IEEE Computer Society, out. de 2015, pp. 1–9, ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7298594. arXiv: 1409.4842.

[38] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens e Z. Wojna, "Rethinking the Inception Architecture for Computer Vision", em *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, IEEE Computer Society, dez. de 2016, pp. 2818–2826, ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.308. arXiv: 1512.00567.

[39] K. He, X. Zhang, S. Ren e J. Sun, "Deep residual learning for image recognition", em *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, IEEE Computer Society, dez. de 2016, pp. 770–778, ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.90. arXiv: 1512.03385.

[40] M. M. Leonardo, T. J. Carvalho, E. Rezende, R. Zucchi e F. A. Faria, "Deep Feature-Based Classifiers for Fruit Fly Identification (Diptera: Tephritidae)", *Proceedings - 31st Conference on Graphics, Patterns and Images, SIBGRAPI 2018*, n.º May 2019, pp. 41–47, 2019. DOI: 10.1109/SIBGRAPI.2018.00012.

[41] *Mic-UK: Hoverflies - All About Hoverflies*. URL: http://www.microscopy-uk.org.uk/mag/indexmag.html?http://www.microscopy-uk.org.uk/mag/artmay07/cd-hoverflies.html (acedido em 20/04/2020).

[42] I. Hodek e A. Honêk, *Ecology of coccinellidae*. Springer Science & Business Media, 2013, vol. 54.

[43] G. S. Albuquerque, "(Neuroptera : Chrysopidae)", n.º January 2009, 2014.

[44] *Philaenus spumarius - Wikiwand*. URL: https://www.wikiwand.com/en/Philaenus%7B%5C_%7Dspumarius (acedido em 20/04/2020).

[45] *models/research/object_detection at master · tensorflow/models · GitHub*. URL: https://github.com/tensorflow/models/tree/master/research/object%7B%5C_%7Ddetection (acedido em 16/04/2020).

[46] *GitHub - tzutalin/labelImg: LabelImg is a graphical image annotation tool and label object bounding boxes in images*. URL: https://github.com/tzutalin/labelImg (acedido em 09/01/2020).

[47] *TensorBoard | TensorFlow*. URL: https://www.tensorflow.org/tensorboard (acedido em 16/04/2020).

[48] M. Lutz, *Programming Python: 4th edition*, sér. Nutshell handbook. O'Reilly Media, Incorporated, 2011, ISBN: 9780596158101.

[49] *pandas - Python Data Analysis Library*. URL: https://pandas.pydata.org/ (acedido em 16/04/2020).

[50] *TFRecord and tf.Example | TensorFlow Core*. URL: https://www.tensorflow.org/tutorials/load_data/tfrecord (acedido em 09/01/2020).

[51] D. Xia, P. Chen, B. Wang, J. Zhang e C. Xie, "Insect detection and classification based on an improved convolutional neural network", *Sensors (Switzerland)*, vol. 18, n.º 12, pp. 1–12, 2018, ISSN: 14248220. DOI: 10.3390/s18124169.

[52] Y. Li, H. Wang, L. M. Dang, A. Sadeghi-Niaraki e H. Moon, "Crop pest recognition in natural scenes using convolutional neural networks", *Computers and Electronics in Agriculture*, vol. 169, n.º July 2019, p. 105 174, 2020, ISSN: 01681699. DOI: 10.1016/j.compag.2019.105174.

[53] Z. Reitermanová, "Data Splitting", 2010.

[54] *GitHub - datitran/raccoon_dataset: The dataset is used to train my own raccoon detector and I blogged about it on Medium*. URL: https://github.com/datitran/raccoon_dataset (acedido em 09/01/2020).

[55] M. Everingham, L. Van Gool, C. K. Williams, J. Winn e A. Zisserman, "The pascal visual object classes (VOC) challenge", *International Journal of Computer Vision*, vol. 88, n.º 2, pp. 303–338, 2010, ISSN: 09205691. DOI: 10.1007/s11263-009-0275-4.

[56] K. He, G. Gkioxari, P. Dollar e R. Girshick, "Mask R-CNN", em *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, Institute of Electrical e Electronics Engineers Inc., dez. de 2017, pp. 2980–2988, ISBN: 9781538610329. DOI: 10.1109/ICCV.2017.322. arXiv: 1703.06870.

Apêndice A

Proposta Original do Projeto

Proposta de Dissertação / Projeto
Mestrado em Engenharia Industrial
2018/2019

Título Provisório: Sensorização e monitorização da quantidade de insetos na cultura da oliveira

---

Aluno de mestrado: Felipe Vieira dos Santos

Orientador: Paulo Leitão
Co-Orientador: José Barbosa e José Alberto Pereira (ESA)
Co-Orientador (UTFPR): Frederic Conrad Janzen

Principais objetivos a atingir:
O objetivo deste trabalho consiste no estudo e desenvolvimento de uma solução que visa a monitorização remota e de forma autónoma da quantidade de insetos em oliveiras, uma vez que estes podem condicionar o seu normal desenvolvimento. O trabalho irá utilizar tecnologias de aquisição de imagem, de Internet das Coisas e de análise de dados de forma a adquirir, processar e enviar a informação para um servidor central.

Resultados esperados:
Pretende-se desenvolver uma solução assente em sensorização que permita a recolha de dados relativos à quantidade de insetos numa oliveira. Esta sensorização deverá estar assente em dispositivos de Internet das Coisas, permitindo embeber inteligência nos dispositivos locais, assim como permitir o envio dos dados recolhidos para sistemas remotos de forma a que possam ser monitorizados.

Caracterização do trabalho
A utilização de tecnologias TIC permite automatizar o processo de monitorização da quantidade de insetos em oliveiras e assim reduzir os custos associados ao processo tradicional que requer a deslocação periódica de um técnico ao olival. Em particular, as tecnologias da Internet das Coisas estão cada vez mais avançadas em termos de capacidade de processamento, consumo energético e envio de informação. A sensorização em ambientes remotos, como os que se encontram em ambientes agrícolas, é assim facilitada, sendo os dados recolhidos por estes sensores e facilmente enviados para locais remotos, permitindo que os seus utilizadores possam tomar decisões com base na informação recolhida.
Deste modo, a presente proposta pretende desenvolver uma abordagem para a sensorização da quantidade de insetos em oliveiras, o envio de dados para um sistema central e a sua posterior monitorização dinâmica, permitindo a geração de alertas e relatórios. Esta solução deve também atender aos requisitos de autonomia, permitindo que a solução a apresentar possa funcionar em modo isolado da rede elétrica.

Calendarização das fases do trabalho:
O desenvolvimento da presente proposta de trabalho será realizado através da execução das seguintes etapas:

1. Familiarização com o problema de monitorização da quantidade de insetos em oliveiras e do estado da arte no domínio (M1-M2)
2. Desenvolvimento do sistema de sensorização (por exemplo recorrendo à aquisição de imagem) da quantidade de insetos em oliveiras (M1 – M4)
3. Desenvolvimento da conetividade do sistema de sensorização para transmissão de dados (M3 – M4)
4. Desenvolvimento do sistema de monitorização dinâmica e remotas (M4 – M7)
5. Testes e validação do protótipo (M7 – M10)
6. Escrita da dissertação e defesa final do trabalho (M10 - M11)

Palavras-chave: IoT, sensorização, instrumentação, monitorização.

Infraestruturas a utilizar:
Este trabalho será desenvolvido no laboratório de Automação, Controlo e Robótica (LCAR).

Declaro que aceito realizar este projeto de dissertação segundo o plano apresentado

_____
(O aluno de mestrado)

Declaro que aceito orientar a preparação deste projeto de dissertação segundo o plano apresentado

_____
(O orientador)

Declaro que aceito orientar a preparação deste projeto de dissertação segundo o plano apresentado

_____
(O co-orientador)

Apêndice B

Principais algoritmo em *Python*

B.1 *generate_tfrecord.py*

```
"""
Usage:
  # From tensorflow/models/
  # Create train data:
  python generate_tfrecord.py --csv_input=data/train_labels.csv
↪  --output_path=train.record

  # Create test data:
  python generate_tfrecord.py --csv_input=data/test_labels.csv
↪  --output_path=test.record
"""
from __future__ import division
from __future__ import print_function
from __future__ import absolute_import

import os
import io
import pandas as pd
import tensorflow as tf

from PIL import Image
from object_detection.utils import dataset_util
from collections import namedtuple, OrderedDict
```

```
flags = tf.app.flags
flags.DEFINE_string('csv_input', '', 'Path to the CSV input')
flags.DEFINE_string('output_path', '', 'Path to output TFRecord')
flags.DEFINE_string('image_dir', '', 'Path to images')
FLAGS = flags.FLAGS


## TO-DO replace this with label map
# def class_text_to_int(row_label):
#     if row_label == 'trioza_erytreae':
#         return 1
#     elif row_label == 'syrphidae':
#         return 2
#     elif row_label == 'coccinellidae':
#         return 3
#     elif row_label == 'chrysopidae':
#         return 4
#     elif row_label == 'philaenus':
#         return 5
#     else:
#         None

#TO-DO replace this with label map
def class_text_to_int(row_label):
    if row_label == 'trioza_erytreae':
        return 1
    elif row_label == 'coccinellidae':
        return 2
    elif row_label == 'chrysopidae':
        return 3
    elif row_label == 'philaenus':
        return 4
    else:
        None




def split(df, group):
    data = namedtuple('data', ['filename', 'object'])
    gb = df.groupby(group)
```

```
    return [data(filename, gb.get_group(x)) for filename, x in zip(gb.groups.keys(),
    ↪  gb.groups)]


def create_tf_example(group, path):
    with tf.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:
        encoded_jpg = fid.read()
    encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size

    filename = group.filename.encode('utf8')
    image_format = b'jpg'
    xmins = []
    xmaxs = []
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []

    for index, row in group.object.iterrows():
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
        classes_text.append(row['class'].encode('utf8'))
        classes.append(class_text_to_int(row['class']))

    tf_example = tf.train.Example(features=tf.train.Features(feature={
        'image/height': dataset_util.int64_feature(height),
        'image/width': dataset_util.int64_feature(width),
        'image/filename': dataset_util.bytes_feature(filename),
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(encoded_jpg),
        'image/format': dataset_util.bytes_feature(image_format),
        'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
        'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
        'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
        'image/object/class/label': dataset_util.int64_list_feature(classes),
    }))
    return tf_example
```

```
def main(_):
    writer = tf.python_io.TFRecordWriter(FLAGS.output_path)
    path = os.path.join(FLAGS.image_dir)
    examples = pd.read_csv(FLAGS.csv_input)
    grouped = split(examples, 'filename')
    for group in grouped:
        tf_example = create_tf_example(group, path)
        writer.write(tf_example.SerializeToString())

    writer.close()
    output_path = os.path.join(os.getcwd(), FLAGS.output_path)
    print('Successfully created the TFRecords: {}'.format(output_path))


if __name__ == '__main__':
    tf.app.run()
```

Argumentos para *generate_tfrecord.py*

```
    # BDA 1 de treino
    --csv_input=images_1\train_labels.csv
    --image_dir=images_1\train
    --output_path=traning_demo_1\annotations\train.record

    # BDA 1 de teste
    --csv_input=images_1\test_labels.csv
    --image_dir=images_1\test
    --output_path=traning_demo_1\annotations\test.record

    # BDA 2 de treino
    --csv_input=images_2\train_labels.csv
    --image_dir=images_2\train
    --output_path=traning_demo_2\annotations\train.record

    # BDA 2 de teste
    --csv_input=images_2\test_labels.csv
    --image_dir=images_2\test
    --output_path=traning_demo_2\annotations\test.record

    # BDA 3 de treino
```

```
--csv_input=images_3\train_labels.csv
--image_dir=images_3\train
--output_path=traning_demo_2\annotations\train.record

# BDA 3 de teste
--csv_input=images_3\test_labels.csv
--image_dir=images_3\test
--output_path=traning_demo_3\annotations\test.record
```

B.2 *model_main.py*

Argumentos para *model_main.py*

```
# BDA 1 na configuração Faster R-CNN Inception-v2
--pipeline_config_path=C:/tese/insectdetector/workspace_2/training_dataset_1/faster_rcnn_inception_2/faster_rcnn_inception_v2_coco.config
--model_dir=C:/tese/insectdetector/workspace_2/training_dataset_1/faster_rcnn_inception_2
--num_train_steps=200000
--sample_1_of_n_eval_examples=1
--alsologtostderr

 # BDA 1 na configuração Faster R-CNN ResNet-50
--pipeline_config_path=C:/tese/insectdetector/workspace_2/training_dataset_1/faster_rcnn_resnet50/faster_rcnn_resnet50_coco.config
--model_dir=C:/tese/insectdetector/workspace_2/training_dataset_1/faster_rcnn_resnet50
--num_train_steps=200000
--sample_1_of_n_eval_examples=1
--alsologtostderr

# BDA 1 na configuração SSD Inception-v2
--pipeline_config_path=C:/tese/insectdetector/workspace_2/training_dataset_1/ssd_inception_v2/ssd_inception_v2_coco.config
--model_dir=C:/tese/insectdetector/workspace_2/training_dataset_1/ssd_inception_v2
--num_train_steps=200000
--sample_1_of_n_eval_examples=1
--alsologtostderr

# BDA 1 na configuração SSD ResNet-50
--pipeline_config_path=C:/tese/insectdetector/workspace_2/training_dataset_1/ssd_resnet50_v1_fpn/ssd_resnet50_v1_fpn_shared_box_predictor_640x640_coco14_sync.config
--model_dir=C:/tese/insectdetector/workspace_2/training_dataset_1/ssd_resnet50_v1_fpn
--num_train_steps=200000
--sample_1_of_n_eval_examples=1
--alsologtostderr

# BDA 2 na configuração Faster R-CNN Inception-v2
--pipeline_config_path=C:/tese/insectdetector/workspace_2/training_dataset_2/faster_rcnn_inception_2/faster_rcnn_inception_v2_coco.config
```

```
--model_dir=C:/tese/insectdetector/workspace_2/training_dataset_2/faster_rcnn_inc
↪  eption_2
--num_train_steps=200000
--sample_1_of_n_eval_examples=1
--alsologtostderr

 # BDA 2 na configuração Faster R-CNN ResNet-50
--pipeline_config_path=C:/tese/insectdetector/workspace_2/training_dataset_2/fast
↪  er_rcnn_resnet50/faster_rcnn_resnet50_coco.config
--model_dir=C:/tese/insectdetector/workspace_2/training_dataset_2/faster_rcnn_res
↪  net50
--num_train_steps=200000
--sample_1_of_n_eval_examples=1
--alsologtostderr

# BDA 2 na configuração SSD Inception-v2
--pipeline_config_path=C:/tese/insectdetector/workspace_2/training_dataset_2/ssd_
↪  inception_v2/ssd_inception_v2_coco.config
--model_dir=C:/tese/insectdetector/workspace_2/training_dataset_2/ssd_inception_v2
--num_train_steps=200000
--sample_1_of_n_eval_examples=1
--alsologtostderr

# BDA 2 na configuração SSD ResNet-50
--pipeline_config_path=C:/tese/insectdetector/workspace_2/training_dataset_2/ssd_
↪  resnet50_v1_fpn/ssd_resnet50_v1_fpn_shared_box_predictor_640x640_coco14_sync.
↪  config
--model_dir=C:/tese/insectdetector/workspace_2/training_dataset_2/ssd_resnet50_v1
↪  _fpn
--num_train_steps=200000
--sample_1_of_n_eval_examples=1
--alsologtostderr

# BDA 3 na configuração Faster R-CNN Inception-v2
--pipeline_config_path=C:/tese/insectdetector/workspace_2/training_dataset_3/fast
↪  er_rcnn_inception_2/faster_rcnn_inception_v2_coco.config
--model_dir=C:/tese/insectdetector/workspace_2/training_dataset_3/faster_rcnn_inc
↪  eption_2
--num_train_steps=200000
--sample_1_of_n_eval_examples=1
--alsologtostderr

 # BDA 3 na configuração Faster R-CNN ResNet-50
```

```
--pipeline_config_path=C:/tese/insectdetector/workspace_3/training_dataset_2/fast
↪  er_rcnn_resnet50/faster_rcnn_resnet50_coco.config
--model_dir=C:/tese/insectdetector/workspace_2/training_dataset_3/faster_rcnn_res
↪  net50
--num_train_steps=200000
--sample_1_of_n_eval_examples=1
--alsologtostderr

# BDA 3 na configuração SSD Inception-v2
--pipeline_config_path=C:/tese/insectdetector/workspace_2/training_dataset_3/ssd_
↪  inception_v2/ssd_inception_v2_coco.config
--model_dir=C:/tese/insectdetector/workspace_2/training_dataset_3/ssd_inception_v2
--num_train_steps=200000
--sample_1_of_n_eval_examples=1
--alsologtostderr

# BDA 3 na configuração SSD ResNet-50
--pipeline_config_path=C:/tese/insectdetector/workspace_2/training_dataset_3/ssd_
↪  resnet50_v1_fpn/ssd_resnet50_v1_fpn_shared_box_predictor_640x640_coco14_sync.
↪  config
--model_dir=C:/tese/insectdetector/workspace_2/training_dataset_3/ssd_resnet50_v1
↪  _fpn
--num_train_steps=200000
--sample_1_of_n_eval_examples=1
--alsologtostderr
```

B.3 *export_inference_graph.py*

Argumentos para *export_inference_graph.py*

```
# model-1
--input_type=image_tensor \
--pipeline_config_path=training_dataset_1/faster_rcnn_inception_2/faster_rcnn_inc
↪  eption_v2_coco.config
↪  \
--trained_checkpoint_prefix=training_dataset_1/faster_rcnn_inception_2/model.ckpt
↪  -30080
↪  \
--output_directory=trained_inference_graphs_dataset_1/output_inference_graph_fast
↪  er_rcnn_inception_2

# model-2
--input_type=image_tensor \
--pipeline_config_path=training_dataset_1/faster_rcnn_resnet50/faster_rcnn_resnet
↪  50_coco.config
↪  \
--trained_checkpoint_prefix=training_dataset_1/faster_rcnn_resnet50/model.ckpt-50
↪  57
↪  \
--output_directory=trained_inference_graphs_dataset_1/output_inference_graph_fast
↪  er_rcnn_resnet50

# model-3
--input_type=image_tensor \
--pipeline_config_path=training_dataset_1/ssd_inception_v2/ssd_inception_v2_coco.
↪  config
↪  \
--trained_checkpoint_prefix=training_dataset_1/ssd_inception_v2/model.ckpt-10760 \
--output_directory=trained_inference_graphs_dataset_1/output_inference_graph_ssd_
↪  inception_v2

# model-5
--input_type=image_tensor \
--pipeline_config_path=training_dataset_2/faster_rcnn_inception_2/faster_rcnn_inc
↪  eption_v2_coco.config
↪  \
--trained_checkpoint_prefix=training_dataset_2/faster_rcnn_inception_2/model.ckpt
↪  -31135
↪  \
```

```
--output_directory=trained_inference_graphs_dataset_2/output_inference_graph_fast
↪  er_rcnn_inception_2

# model-6
--input_type=image_tensor \
--pipeline_config_path=training_dataset_2/faster_rcnn_resnet50/faster_rcnn_resnet
↪  50_coco.config
↪  \
--trained_checkpoint_prefix=training_dataset_2/faster_rcnn_resnet50/model.ckpt-52
↪  80
↪  \
--output_directory=trained_inference_graphs_dataset_2/output_inference_graph_fast
↪  er_rcnn_resnet50

# model-7
--input_type=image_tensor \
--pipeline_config_path=training_dataset_2/ssd_inception_v2/ssd_inception_v2_coco.
↪  config
↪  \
--trained_checkpoint_prefix=training_dataset_2/ssd_inception_v2/model.ckpt-10417 \
--output_directory=trained_inference_graphs_dataset_2/output_inference_graph_ssd_
↪  inception_v2

# model-9
--input_type=image_tensor \
--pipeline_config_path=training_dataset_3/faster_rcnn_inception_2/faster_rcnn_inc
↪  eption_v2_coco.config
↪  \
--trained_checkpoint_prefix=training_dataset_3/faster_rcnn_inception_2/model.ckpt
↪  -31660
↪  \
--output_directory=trained_inference_graphs_dataset_3/output_inference_graph_fast
↪  er_rcnn_inception_2

# model-10
--input_type=image_tensor \
--pipeline_config_path=training_dataset_3/faster_rcnn_resnet50/faster_rcnn_resnet
↪  50_coco.config
↪  \
--trained_checkpoint_prefix=training_dataset_3/faster_rcnn_resnet50/model.ckpt-48
↪  48
↪  \
```

```
--output_directory=trained_inference_graphs_dataset_3/output_inference_graph_fast_
↪  er_rcnn_resnet50

# model-11
--input_type=image_tensor \
--pipeline_config_path=training_dataset_3/ssd_inception_v2/ssd_inception_v2_coco._
↪  config
↪  \
--trained_checkpoint_prefix=training_dataset_3/ssd_inception_v2/model.ckpt-10050 \
--output_directory=trained_inference_graphs_dataset_3/output_inference_graph_ssd_
↪  inception_v2
```

B.4 *app_insect_detector.py*

```
#imports to tkinter works
import tkinter as tk
from PIL import ImageTk
from tkinter import filedialog, messagebox
from functools import partial

#imports to tensorflow and insect detector works
import pathlib

import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image
from IPython.display import display


#Import the object detection module.
from object_detection.utils import ops as utils_ops
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util


#import to works export csv data
import pandas as pd

try:
    df = pd.read_csv('C:/tese/insectdetector/workspace_2/app
    ↪  data/database/imagesinsectdata.csv')
except:
    df = pd.DataFrame(
        {'date': [], 'location': [], 'trioza_erytreae': [], 'syrphidae': [],
        ↪  'coccinellidae': [], 'chrysopidae': [],
         'philaenus': []})
```

```
else:
    pass


if "models" in pathlib.Path.cwd().parts:
    while "models" in pathlib.Path.cwd().parts:
        os.chdir('..')


utils_ops.tf = tf.compat.v1

# Patch the location of gfile
tf.gfile = tf.io.gfile

#Loader
def load_model(model_name):
    model_dir = 'C:/tese/insectdetector/workspace_2/trained_inference_graphs_dataset_
    ↪  3/output_inference_graph_ssd_inception_v2'
    model_dir = pathlib.Path(model_dir)/"saved_model"

    model = tf.saved_model.load(str(model_dir))
    model = model.signatures['serving_default']

    return model


# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = 'C:/tese/insectdetector/workspace_2/dataset_3/label_map.pbtxt'
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
↪  use_display_name=True)


def run_inference_for_single_image(model, image):
    image = np.asarray(image)
    # The input needs to be a tensor, convert it using `tf.convert_to_tensor`.
    input_tensor = tf.convert_to_tensor(image)
    # The model expects a batch of images, so add an axis with `tf.newaxis`.
    input_tensor = input_tensor[tf.newaxis, ...]

    # Run inference
    output_dict = model(input_tensor)

    # All outputs are batches tensors.
```

```
    # Convert to numpy arrays, and take index [0] to remove the batch dimension.
    # We're only interested in the first num_detections.
    num_detections = int(output_dict.pop('num_detections'))
    output_dict = {key: value[0, :num_detections].numpy()
                   for key, value in output_dict.items()}
    output_dict['num_detections'] = num_detections

    # detection_classes should be ints.
    output_dict['detection_classes'] =
    ↪  output_dict['detection_classes'].astype(np.int64)

    # Handle models with masks:
    if 'detection_masks' in output_dict:
        # Reframe the the bbox mask to the image size.
        detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
            output_dict['detection_masks'], output_dict['detection_boxes'],
            image.shape[0], image.shape[1])
        detection_masks_reframed = tf.cast(detection_masks_reframed > 0.5,
                                           tf.uint8)
        output_dict['detection_masks_reframed'] = detection_masks_reframed.numpy()

    return output_dict

def show_inference1(model, image_path):
    global labelimg
    global img2
    global df
    # the array based representation of the image will be used later in order to
    ↪  prepare the
    # result image with boxes and labels on it.
    image_np = np.array(Image.open(image_path))
    # Actual detection.
    output_dict = run_inference_for_single_image(model, image_np)
    # Visualization of the results of a detection.
    vis_util.visualize_boxes_and_labels_on_image_array(
        image_np, output_dict['detection_boxes'],
        output_dict['detection_classes'],
        output_dict['detection_scores'],
        category_index,
        instance_masks=output_dict.get('detection_masks_reframed', None),
        use_normalized_coordinates=True,
        line_thickness=8)
```

```
    try:
        trioza_erytreae = vis_util.get_number_insect()['trioza_erytreae']
    except:
        trioza_erytreae = 0

    try:
        syrphidae = vis_util.get_number_insect()['syrphidae']
    except:
        syrphidae = 0

    try:
        coccinellidae = vis_util.get_number_insect()['coccinellidae']
    except:
        coccinellidae = 0

    try:
        chrysopidae = vis_util.get_number_insect()['chrysopidae']
    except:
        chrysopidae = 0

    try:
        philaenus = vis_util.get_number_insect()['philaenus']
    except:
        philaenus = 0


    data = [
        {'location': image_path[image_path.rfind('/')+1:image_path.rfind('_')],
         'date': image_path[image_path.rfind('_') + 1:image_path.rfind('.')],
         'trioza_erytreae': int(trioza_erytreae),
         'syrphidae': int(syrphidae),
         'coccinellidae': int(coccinellidae),
         'chrysopidae': int(chrysopidae),
         'philaenus': int(philaenus)}
    ]

    df = df.append(data, ignore_index=True)
    #dfinsect['text'] = df.to_string()
    print(df.to_string())

def show_inference2(model, image_path):
    global labelimg
```

```
    global img2
    global df
    # the array based representation of the image will be used later in order to
    ↪  prepare the
    # result image with boxes and labels on it.
    image_np = np.array(Image.open(image_path))
    # Actual detection.
    output_dict = run_inference_for_single_image(model, image_np)
    # Visualization of the results of a detection.
    vis_util.visualize_boxes_and_labels_on_image_array(
        image_np, output_dict['detection_boxes'],
        output_dict['detection_classes'],
        output_dict['detection_scores'],
        category_index,
        instance_masks=output_dict.get('detection_masks_reframed', None),
        use_normalized_coordinates=True,
        line_thickness=8)

    display(Image.fromarray(image_np))
    #script teste de plotar imagem
    #img2p = Image.open("C:/tese/insectdetector/models/research/object_detection/inse
    ↪  tos.jpg")
    img2p = Image.fromarray(image_np)
    #n = 1
    #[imageSizeWidth, imageSizeHeight] = img1p.size
    #newImageSizeWidth = int(imageSizeWidth * n)
    #newImageSizeHeight = int(imageSizeHeight * n)

    img2 = ImageTk.PhotoImage(img2p.resize((newImageSizeWidth, newImageSizeHeight),
    ↪  Image.ANTIALIAS))

    labelimg.grid_forget()
    labelimg = tk.Label(frameimage, image=img2, anchor=tk.N)
    labelimg.grid(row=0, column=0)
    statusbar['text'] = str(vis_util.get_number_insect()).replace('{',
    ↪  '').replace('}', '').replace("'", '')



model_name = 'output_inference_graph_v1'
detection_model = load_model(model_name)
```

```
#codigo para criação da interface grafica

window1 = tk.Tk()
window1.title('Insect detector by Felipe Vieira dos Santos')
window1.iconbitmap('C:/tese/insectdetector/workspace_2/app data/ladybug.ico')
#window1.geometry('1280x720+200+200')
window1.geometry('1024x940+100+50')

def openfn():
    filename = filedialog.askopenfilename(multiple=True, title='open',
    ↪  filetypes=[("Images files", "*.jpg")])
    return filename


TEST_IMAGE_PATHS = list()
def open_img():
    global TEST_IMAGE_PATHS, PATH_TO_TEST_IMAGES_DIR
    path_images = openfn()
    print(path_images)
    caminho_imagens = str(path_images[0][0:str(path_images[0]).rfind('/')])
    PATH_TO_TEST_IMAGES_DIR = pathlib.Path(caminho_imagens)
    TEST_IMAGE_PATHS = path_images

    lb2.delete(0, tk.END)
    for path in path_images:
        lb2.insert(tk.END, path[path.rfind('/')+1:])



def execucao():
    response = messagebox.askquestion("Attention", "The process may take a few
    ↪  minutes, do you want to continue?")
    if response == 'yes':
        for image_path in TEST_IMAGE_PATHS:
            show_inference1(detection_model, image_path)
            print(vis_util.get_number_insect())
    else:
        pass
    dfinsect.delete("1.0", tk.END)
    dfinsect.insert(tk.END, df.to_string())

def indivitual():
```

```
    show_inference2(detection_model,
    ↪  TEST_IMAGE_PATHS[0][0:TEST_IMAGE_PATHS[0].rfind('/')+1] + lb2.get(tk.ACTIVE))


def exporteddate():
    df.to_csv('C:/tese/insectdetector/workspace_2/app
    ↪  data/database/imagesinsectdata.csv', index=None, header=True)


frameload = tk.LabelFrame(window1, text="load files images")
frameload.grid(row=0, column=0, sticky=tk.N)

lb1 = tk.Button(frameload, text='Select Images', command=open_img)
lb1.pack()

barraderolagemy = tk.Scrollbar(frameload)
barraderolagemy.pack(side=tk.RIGHT, fill=tk.Y)

lb2 = tk.Listbox(frameload, yscrollcommand=barraderolagemy.set, height=20, width=40)
lb2.pack(side=tk.LEFT, fill=tk.BOTH)

barraderolagemy.config(command=lb2.yview)


frameprocessor = tk.LabelFrame(window1)
frameprocessor.grid(row=1, column=0)

bt1 = tk.Button(frameprocessor, text='Detect Insect', command=indivitual)
bt1.pack(side=tk.LEFT)
btt = tk.Button(frameprocessor, text='all files process', command=execucao)
btt.pack(side=tk.LEFT)



frameimage = tk.LabelFrame(window1, text='image processed')
frameimage.grid(row=0, column=1, rowspan=2)

img1p = Image.open("C:/tese/insectdetector/workspace_2/app data/back.png")
n = 0.9
[imageSizeWidth, imageSizeHeight] = img1p.size

newImageSizeWidth = int(imageSizeWidth * n)
```

```
newImageSizeHeight = int(imageSizeHeight * n)
img1 = ImageTk.PhotoImage(img1p.resize((newImageSizeWidth, newImageSizeHeight),
↪  Image.ANTIALIAS))

labelimg = tk.Label(frameimage, image=img1, anchor=tk.N)
labelimg.grid(row=0, column=0, columnspan=2)

statusbar = tk.Label(frameimage, text="number of insects", bd=1, relief=tk.SUNKEN,
↪  anchor=tk.W)
statusbar.grid(row=1, column=0, sticky=tk.W+tk.E, columnspan=2)

rolagemy = tk.Scrollbar(frameimage)
rolagemy.grid(row=2, column=1, sticky=tk.NE+tk.SE)

dfinsect = tk.Text(frameimage, yscrollcommand=rolagemy.set, height=9, bd=1,
↪  relief=tk.SUNKEN, font=("Times New Roman", 10))
dfinsect.grid(row=2, column=0, sticky=tk.W+tk.E)
rolagemy.config(command=dfinsect.yview)

isempty = df.empty
if isempty is not True:
    dfinsect.insert(tk.END, df.to_string())

print(isempty)
frameexport = tk.LabelFrame(window1)
frameexport.grid(row=2, column=1)

bt2 = tk.Button(frameexport, text='Exported Data', width=30, command=exporteddate)
bt2.pack()




window1.mainloop()
```

Apêndice C

Parâmetros de treinamento

C.1 Configurações para Treinos do BDA 1

***Faster* R-CNN *Inception*-v2**

```
# Faster R-CNN with Inception v2, configuration for MSCOCO Dataset.
# Users should configure the fine_tune_checkpoint field in the train config as
# well as the label_map_path and input_path fields in the train_input_reader and
# eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find the fields that
# should be configured.


model {
  faster_rcnn {
    num_classes: 5
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
    feature_extractor {
      type: 'faster_rcnn_inception_v2'
      first_stage_features_stride: 16
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
```

```
    scales: [0.25, 0.5, 1.0, 2.0]
    aspect_ratios: [0.5, 1.0, 2.0]
    height_stride: 16
    width_stride: 16
  }
}
first_stage_box_predictor_conv_hyperparams {
  op: CONV
  regularizer {
    l2_regularizer {
      weight: 0.0
    }
  }
  initializer {
    truncated_normal_initializer {
      stddev: 0.01
    }
  }
}
first_stage_nms_score_threshold: 0.0
first_stage_nms_iou_threshold: 0.7
first_stage_max_proposals: 300
first_stage_localization_loss_weight: 2.0
first_stage_objectness_loss_weight: 1.0
initial_crop_size: 14
maxpool_kernel_size: 2
maxpool_stride: 2
second_stage_box_predictor {
  mask_rcnn_box_predictor {
    use_dropout: false
    dropout_keep_probability: 1.0
    fc_hyperparams {
      op: FC
      regularizer {
        l2_regularizer {
          weight: 0.0
        }
      }
      initializer {
        variance_scaling_initializer {
          factor: 1.0
          uniform: true
          mode: FAN_AVG
```

```
            }
          }
        }
      }
    }
    second_stage_post_processing {
      batch_non_max_suppression {
        score_threshold: 0.0
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 300
      }
      score_converter: SOFTMAX
    }
    second_stage_localization_loss_weight: 2.0
    second_stage_classification_loss_weight: 1.0
  }
}

train_config: {
  batch_size: 1
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        manual_step_learning_rate {
          initial_learning_rate: 0.0002
          schedule {
            step: 900000
            learning_rate: .00002
          }
          schedule {
            step: 1200000
            learning_rate: .000002
          }
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  gradient_clipping_by_norm: 10.0
  fine_tune_checkpoint: "C:/tese/insectdetector/workspace_2/pre_trained_model/faster_
  ↪  rcnn_inception_v2_coco_2018_01_28/model.ckpt"
```

```
  from_detection_checkpoint: true
  # Note: The below line limits the training process to 200K steps, which we
  # empirically found to be sufficient enough to train the COCO dataset. This
  # effectively bypasses the learning rate schedule (the learning rate will
  # never decay). Remove the below line to train indefinitely.
  num_steps: 200000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_1/train.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_1/label_map.pbtxt"
}

eval_config: {
  num_examples: 86
  # Note: The below line limits the evaluation process to 10 evaluations.
  # Remove the below line to evaluate indefinitely.
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_1/test.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_1/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

Faster R-CNN *ResNet*-50

```
# Faster R-CNN with Resnet-50 (v1), configuration for MSCOCO Dataset.
# Users should configure the fine_tune_checkpoint field in the train config as
# well as the label_map_path and input_path fields in the train_input_reader and
# eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find the fields that
# should be configured.
```

```
model {
  faster_rcnn {
    num_classes: 5
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
    feature_extractor {
      type: 'faster_rcnn_resnet50'
      first_stage_features_stride: 16
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
        scales: [0.25, 0.5, 1.0, 2.0]
        aspect_ratios: [0.5, 1.0, 2.0]
        height_stride: 16
        width_stride: 16
      }
    }
    first_stage_box_predictor_conv_hyperparams {
      op: CONV
      regularizer {
        l2_regularizer {
          weight: 0.0
        }
      }
      initializer {
        truncated_normal_initializer {
          stddev: 0.01
        }
      }
    }
    first_stage_nms_score_threshold: 0.0
    first_stage_nms_iou_threshold: 0.7
    first_stage_max_proposals: 300
    first_stage_localization_loss_weight: 2.0
    first_stage_objectness_loss_weight: 1.0
    initial_crop_size: 14
    maxpool_kernel_size: 2
    maxpool_stride: 2
```

```
    second_stage_box_predictor {
      mask_rcnn_box_predictor {
        use_dropout: false
        dropout_keep_probability: 1.0
        fc_hyperparams {
          op: FC
          regularizer {
            l2_regularizer {
              weight: 0.0
            }
          }
          initializer {
            variance_scaling_initializer {
              factor: 1.0
              uniform: true
              mode: FAN_AVG
            }
          }
        }
      }
    }
    second_stage_post_processing {
      batch_non_max_suppression {
        score_threshold: 0.0
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 300
      }
      score_converter: SOFTMAX
    }
    second_stage_localization_loss_weight: 2.0
    second_stage_classification_loss_weight: 1.0
  }
}

train_config: {
  batch_size: 1
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        manual_step_learning_rate {
          initial_learning_rate: 0.0003
          schedule {
```

```
            step: 900000
            learning_rate: .00003
          }
          schedule {
            step: 1200000
            learning_rate: .000003
          }
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  gradient_clipping_by_norm: 10.0
  fine_tune_checkpoint: "C:/tese/insectdetector/workspace_2/pre_trained_model/faster_
  ↪  rcnn_resnet50_coco_2018_01_28/model.ckpt"
  from_detection_checkpoint: true
  # Note: The below line limits the training process to 200K steps, which we
  # empirically found to be sufficient enough to train the pets dataset. This
  # effectively bypasses the learning rate schedule (the learning rate will
  # never decay). Remove the below line to train indefinitely.
  num_steps: 200000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_1/train.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_1/label_map.pbtxt"
}

eval_config: {
  num_examples: 86
  # Note: The below line limits the evaluation process to 10 evaluations.
  # Remove the below line to evaluate indefinitely.
  max_evals: 10
}

eval_input_reader: {
```

```
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_1/test.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_1/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

SSD R-CNN *Inception*-v2

```
# SSD with Inception v2 configuration for MSCOCO Dataset.
# Users should configure the fine_tune_checkpoint field in the train config as
# well as the label_map_path and input_path fields in the train_input_reader and
# eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find the fields that
# should be configured.

model {
  ssd {
    num_classes: 5
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    anchor_generator {
      ssd_anchor_generator {
```

```
    num_layers: 6
    min_scale: 0.2
    max_scale: 0.95
    aspect_ratios: 1.0
    aspect_ratios: 2.0
    aspect_ratios: 0.5
    aspect_ratios: 3.0
    aspect_ratios: 0.3333
    reduce_boxes_in_lowest_layer: true
  }
}
image_resizer {
  fixed_shape_resizer {
    height: 300
    width: 300
  }
}
box_predictor {
  convolutional_box_predictor {
    min_depth: 0
    max_depth: 0
    num_layers_before_predictor: 0
    use_dropout: false
    dropout_keep_probability: 0.8
    kernel_size: 3
    box_code_size: 4
    apply_sigmoid_to_scores: false
    conv_hyperparams {
      activation: RELU_6,
      regularizer {
        l2_regularizer {
          weight: 0.00004
        }
      }
      initializer {
        truncated_normal_initializer {
          stddev: 0.03
          mean: 0.0
        }
      }
    }
  }
}
```

```
feature_extractor {
  type: 'ssd_inception_v2'
  min_depth: 16
  depth_multiplier: 1.0
  conv_hyperparams {
    activation: RELU_6,
    regularizer {
      l2_regularizer {
        weight: 0.00004
      }
    }
    initializer {
      truncated_normal_initializer {
        stddev: 0.03
        mean: 0.0
      }
    }
    batch_norm {
      train: true,
      scale: true,
      center: true,
      decay: 0.9997,
      epsilon: 0.001,
    }
  }
  override_base_feature_extractor_hyperparams: true
}
loss {
  classification_loss {
    weighted_sigmoid {
    }
  }
  localization_loss {
    weighted_smooth_l1 {
    }
  }
  hard_example_miner {
    num_hard_examples: 3000
    iou_threshold: 0.99
    loss_type: CLASSIFICATION
    max_negatives_per_positive: 3
    min_negatives_per_image: 0
  }
```

```
      classification_weight: 1.0
      localization_weight: 1.0
    }
    normalize_loss_by_num_matches: true
    post_processing {
      batch_non_max_suppression {
        score_threshold: 1e-8
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 100
      }
      score_converter: SIGMOID
    }
  }
}

train_config: {
  batch_size: 24
  optimizer {
    rms_prop_optimizer: {
      learning_rate: {
        exponential_decay_learning_rate {
          initial_learning_rate: 0.004
          decay_steps: 800720
          decay_factor: 0.95
        }
      }
      momentum_optimizer_value: 0.9
      decay: 0.9
      epsilon: 1.0
    }
  }
  fine_tune_checkpoint: "C:/tese/insectdetector/workspace_2/pre_trained_model/ssd_inc
  ↪  eption_v2_coco_2018_01_28/model.ckpt"
  from_detection_checkpoint: true
  # Note: The below line limits the training process to 200K steps, which we
  # empirically found to be sufficient enough to train the pets dataset. This
  # effectively bypasses the learning rate schedule (the learning rate will
  # never decay). Remove the below line to train indefinitely.
  num_steps: 200000
  data_augmentation_options {
    random_horizontal_flip {
    }
```

```
  }
  data_augmentation_options {
    ssd_random_crop {
    }
  }
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_1/train.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_1/label_map.pbtxt"
}

eval_config: {
  num_examples: 86
  # Note: The below line limits the evaluation process to 10 evaluations.
  # Remove the below line to evaluate indefinitely.
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_1/test.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_1/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

SSD R-CNN *ResNet*-50

```
# SSD with Resnet 50 v1 FPN feature extractor, shared box predictor and focal
# loss (a.k.a Retinanet).
# See Lin et al, https://arxiv.org/abs/1708.02002
# Trained on COCO, initialized from Imagenet classification checkpoint

# Achieves 35.2 mAP on COCO14 minival dataset. Doubling the number of training
# steps to 50k gets 36.9 mAP

# This config is TPU compatible
```

```
model {
  ssd {
    inplace_batchnorm_update: true
    freeze_batchnorm: false
    num_classes: 5
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
        use_matmul_gather: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    encode_background_as_zeros: true
    anchor_generator {
      multiscale_anchor_generator {
        min_level: 3
        max_level: 7
        anchor_scale: 4.0
        aspect_ratios: [1.0, 2.0, 0.5]
        scales_per_octave: 2
      }
    }
    image_resizer {
      fixed_shape_resizer {
        height: 640
        width: 640
      }
    }
```

```
box_predictor {
  weight_shared_convolutional_box_predictor {
    depth: 256
    class_prediction_bias_init: -4.6
    conv_hyperparams {
      activation: RELU_6,
      regularizer {
        l2_regularizer {
          weight: 0.0004
        }
      }
      initializer {
        random_normal_initializer {
          stddev: 0.01
          mean: 0.0
        }
      }
      batch_norm {
        scale: true,
        decay: 0.997,
        epsilon: 0.001,
      }
    }
    num_layers_before_predictor: 4
    kernel_size: 3
  }
}
feature_extractor {
  type: 'ssd_resnet50_v1_fpn'
  fpn {
    min_level: 3
    max_level: 7
  }
  min_depth: 16
  depth_multiplier: 1.0
  conv_hyperparams {
    activation: RELU_6,
    regularizer {
      l2_regularizer {
        weight: 0.0004
      }
    }
    initializer {
```

```
          truncated_normal_initializer {
            stddev: 0.03
            mean: 0.0
          }
        }
        batch_norm {
          scale: true,
          decay: 0.997,
          epsilon: 0.001,
        }
      }
      override_base_feature_extractor_hyperparams: true
    }
    loss {
      classification_loss {
        weighted_sigmoid_focal {
          alpha: 0.25
          gamma: 2.0
        }
      }
      localization_loss {
        weighted_smooth_l1 {
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    normalize_loss_by_num_matches: true
    normalize_loc_loss_by_codesize: true
    post_processing {
      batch_non_max_suppression {
        score_threshold: 1e-8
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 100
      }
      score_converter: SIGMOID
    }
  }
}

train_config: {
```

```
  fine_tune_checkpoint: "C:/tese/insectdetector/workspace_2/pre_trained_model/ssd_res
  ↪  net50_v1_fpn_shared_box_predictor_640x640_coco14_sync_2018_07_03/model.ckpt"
  batch_size: 64
  sync_replicas: true
  startup_delay_steps: 0
  replicas_to_aggregate: 8
  num_steps: 25000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        cosine_decay_learning_rate {
          learning_rate_base: .04
          total_steps: 25000
          warmup_learning_rate: .013333
          warmup_steps: 2000
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_1/train.record"
  }
```

```
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_1/label_map.pbtxt"
}

eval_config: {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
  num_examples: 86
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_1/test.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_1/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

C.2 Configurações para Treinos do BDA 2

Faster R-CNN *Inception*-v2

```
# Faster R-CNN with Inception v2, configuration for MSCOCO Dataset.
# Users should configure the fine_tune_checkpoint field in the train config as
# well as the label_map_path and input_path fields in the train_input_reader and
# eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find the fields that
# should be configured.


model {
  faster_rcnn {
    num_classes: 4
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
    feature_extractor {
      type: 'faster_rcnn_inception_v2'
      first_stage_features_stride: 16
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
        scales: [0.25, 0.5, 1.0, 2.0]
        aspect_ratios: [0.5, 1.0, 2.0]
        height_stride: 16
        width_stride: 16
      }
    }
    first_stage_box_predictor_conv_hyperparams {
      op: CONV
      regularizer {
        l2_regularizer {
          weight: 0.0
        }
      }
      initializer {
        truncated_normal_initializer {
          stddev: 0.01
        }
```

```
    }
  }
  first_stage_nms_score_threshold: 0.0
  first_stage_nms_iou_threshold: 0.7
  first_stage_max_proposals: 300
  first_stage_localization_loss_weight: 2.0
  first_stage_objectness_loss_weight: 1.0
  initial_crop_size: 14
  maxpool_kernel_size: 2
  maxpool_stride: 2
  second_stage_box_predictor {
    mask_rcnn_box_predictor {
      use_dropout: false
      dropout_keep_probability: 1.0
      fc_hyperparams {
        op: FC
        regularizer {
          l2_regularizer {
            weight: 0.0
          }
        }
        initializer {
          variance_scaling_initializer {
            factor: 1.0
            uniform: true
            mode: FAN_AVG
          }
        }
      }
    }
  }
  second_stage_post_processing {
    batch_non_max_suppression {
      score_threshold: 0.0
      iou_threshold: 0.6
      max_detections_per_class: 100
      max_total_detections: 300
    }
    score_converter: SOFTMAX
  }
  second_stage_localization_loss_weight: 2.0
  second_stage_classification_loss_weight: 1.0
}
```

```
}

train_config: {
  batch_size: 1
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        manual_step_learning_rate {
          initial_learning_rate: 0.0002
          schedule {
            step: 900000
            learning_rate: .00002
          }
          schedule {
            step: 1200000
            learning_rate: .000002
          }
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  gradient_clipping_by_norm: 10.0
  fine_tune_checkpoint: "C:/tese/insectdetector/workspace_2/pre_trained_model/faster_
  ↪  rcnn_inception_v2_coco_2018_01_28/model.ckpt"
  from_detection_checkpoint: true
  # Note: The below line limits the training process to 200K steps, which we
  # empirically found to be sufficient enough to train the COCO dataset. This
  # effectively bypasses the learning rate schedule (the learning rate will
  # never decay). Remove the below line to train indefinitely.
  num_steps: 200000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_2/train.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_2/label_map.pbtxt"
```

```
}

eval_config: {
  num_examples: 233
  # Note: The below line limits the evaluation process to 10 evaluations.
  # Remove the below line to evaluate indefinitely.
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_2/test.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_2/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

Faster R-CNN *ResNet*-50

```
# Faster R-CNN with Resnet-50 (v1), configuration for MSCOCO Dataset.
# Users should configure the fine_tune_checkpoint field in the train config as
# well as the label_map_path and input_path fields in the train_input_reader and
# eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find the fields that
# should be configured.

model {
  faster_rcnn {
    num_classes: 4
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
    feature_extractor {
      type: 'faster_rcnn_resnet50'
      first_stage_features_stride: 16
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
        scales: [0.25, 0.5, 1.0, 2.0]
```

```
    aspect_ratios: [0.5, 1.0, 2.0]
    height_stride: 16
    width_stride: 16
  }
}
first_stage_box_predictor_conv_hyperparams {
  op: CONV
  regularizer {
    l2_regularizer {
      weight: 0.0
    }
  }
  initializer {
    truncated_normal_initializer {
      stddev: 0.01
    }
  }
}
first_stage_nms_score_threshold: 0.0
first_stage_nms_iou_threshold: 0.7
first_stage_max_proposals: 300
first_stage_localization_loss_weight: 2.0
first_stage_objectness_loss_weight: 1.0
initial_crop_size: 14
maxpool_kernel_size: 2
maxpool_stride: 2
second_stage_box_predictor {
  mask_rcnn_box_predictor {
    use_dropout: false
    dropout_keep_probability: 1.0
    fc_hyperparams {
      op: FC
      regularizer {
        l2_regularizer {
          weight: 0.0
        }
      }
      initializer {
        variance_scaling_initializer {
          factor: 1.0
          uniform: true
          mode: FAN_AVG
        }
```

```
          }
        }
      }
    }
    second_stage_post_processing {
      batch_non_max_suppression {
        score_threshold: 0.0
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 300
      }
      score_converter: SOFTMAX
    }
    second_stage_localization_loss_weight: 2.0
    second_stage_classification_loss_weight: 1.0
  }
}

train_config: {
  batch_size: 1
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        manual_step_learning_rate {
          initial_learning_rate: 0.0003
          schedule {
            step: 900000
            learning_rate: .00003
          }
          schedule {
            step: 1200000
            learning_rate: .000003
          }
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  gradient_clipping_by_norm: 10.0
  fine_tune_checkpoint: "C:/tese/insectdetector/workspace_2/pre_trained_model/faster_
  ↪  rcnn_resnet50_coco_2018_01_28/model.ckpt"
  from_detection_checkpoint: true
```

```
  # Note: The below line limits the training process to 200K steps, which we
  # empirically found to be sufficient enough to train the pets dataset. This
  # effectively bypasses the learning rate schedule (the learning rate will
  # never decay). Remove the below line to train indefinitely.
  num_steps: 200000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_2/train.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_2/label_map.pbtxt"
}

eval_config: {
  num_examples: 233
  # Note: The below line limits the evaluation process to 10 evaluations.
  # Remove the below line to evaluate indefinitely.
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_2/test.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_2/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

SSD R-CNN *Inception*-v2

```
# SSD with Inception v2 configuration for MSCOCO Dataset.
# Users should configure the fine_tune_checkpoint field in the train config as
# well as the label_map_path and input_path fields in the train_input_reader and
# eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find the fields that
# should be configured.
```

```
model {
  ssd {
    num_classes: 4
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    anchor_generator {
      ssd_anchor_generator {
        num_layers: 6
        min_scale: 0.2
        max_scale: 0.95
        aspect_ratios: 1.0
        aspect_ratios: 2.0
        aspect_ratios: 0.5
        aspect_ratios: 3.0
        aspect_ratios: 0.3333
        reduce_boxes_in_lowest_layer: true
      }
    }
    image_resizer {
      fixed_shape_resizer {
        height: 300
        width: 300
      }
    }
```

```
box_predictor {
  convolutional_box_predictor {
    min_depth: 0
    max_depth: 0
    num_layers_before_predictor: 0
    use_dropout: false
    dropout_keep_probability: 0.8
    kernel_size: 3
    box_code_size: 4
    apply_sigmoid_to_scores: false
    conv_hyperparams {
      activation: RELU_6,
      regularizer {
        l2_regularizer {
          weight: 0.00004
        }
      }
      initializer {
        truncated_normal_initializer {
          stddev: 0.03
          mean: 0.0
        }
      }
    }
  }
}
feature_extractor {
  type: 'ssd_inception_v2'
  min_depth: 16
  depth_multiplier: 1.0
  conv_hyperparams {
    activation: RELU_6,
    regularizer {
      l2_regularizer {
        weight: 0.00004
      }
    }
    initializer {
      truncated_normal_initializer {
        stddev: 0.03
        mean: 0.0
      }
    }
```

```
        batch_norm {
          train: true,
          scale: true,
          center: true,
          decay: 0.9997,
          epsilon: 0.001,
        }
      }
      override_base_feature_extractor_hyperparams: true
    }
    loss {
      classification_loss {
        weighted_sigmoid {
        }
      }
      localization_loss {
        weighted_smooth_l1 {
        }
      }
      hard_example_miner {
        num_hard_examples: 3000
        iou_threshold: 0.99
        loss_type: CLASSIFICATION
        max_negatives_per_positive: 3
        min_negatives_per_image: 0
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    normalize_loss_by_num_matches: true
    post_processing {
      batch_non_max_suppression {
        score_threshold: 1e-8
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 100
      }
      score_converter: SIGMOID
    }
  }
}

train_config: {
```

```
  batch_size: 24
  optimizer {
    rms_prop_optimizer: {
      learning_rate: {
        exponential_decay_learning_rate {
          initial_learning_rate: 0.004
          decay_steps: 800720
          decay_factor: 0.95
        }
      }
      momentum_optimizer_value: 0.9
      decay: 0.9
      epsilon: 1.0
    }
  }
  fine_tune_checkpoint: "C:/tese/insectdetector/workspace_2/pre_trained_model/ssd_inc
  ↪  eption_v2_coco_2018_01_28/model.ckpt"
  from_detection_checkpoint: true
  # Note: The below line limits the training process to 200K steps, which we
  # empirically found to be sufficient enough to train the pets dataset. This
  # effectively bypasses the learning rate schedule (the learning rate will
  # never decay). Remove the below line to train indefinitely.
  num_steps: 200000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    ssd_random_crop {
    }
  }
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_2/train.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_2/label_map.pbtxt"
}

eval_config: {
  num_examples: 233
  # Note: The below line limits the evaluation process to 10 evaluations.
```

```
  # Remove the below line to evaluate indefinitely.
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_2/test.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_2/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

SSD R-CNN *ResNet*-50

```
# SSD with Resnet 50 v1 FPN feature extractor, shared box predictor and focal
# loss (a.k.a Retinanet).
# See Lin et al, https://arxiv.org/abs/1708.02002
# Trained on COCO, initialized from Imagenet classification checkpoint

# Achieves 35.2 mAP on COCO14 minival dataset. Doubling the number of training
# steps to 50k gets 36.9 mAP

# This config is TPU compatible

model {
  ssd {
    inplace_batchnorm_update: true
    freeze_batchnorm: false
    num_classes: 4
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
```

```
    ignore_thresholds: false
    negatives_lower_than_unmatched: true
    force_match_for_each_row: true
    use_matmul_gather: true
  }
}
similarity_calculator {
  iou_similarity {
  }
}
encode_background_as_zeros: true
anchor_generator {
  multiscale_anchor_generator {
    min_level: 3
    max_level: 7
    anchor_scale: 4.0
    aspect_ratios: [1.0, 2.0, 0.5]
    scales_per_octave: 2
  }
}
image_resizer {
  fixed_shape_resizer {
    height: 640
    width: 640
  }
}
box_predictor {
  weight_shared_convolutional_box_predictor {
    depth: 256
    class_prediction_bias_init: -4.6
    conv_hyperparams {
      activation: RELU_6,
      regularizer {
        l2_regularizer {
          weight: 0.0004
        }
      }
      initializer {
        random_normal_initializer {
          stddev: 0.01
          mean: 0.0
        }
      }
```

```
      batch_norm {
        scale: true,
        decay: 0.997,
        epsilon: 0.001,
      }
    }
    num_layers_before_predictor: 4
    kernel_size: 3
  }
}
feature_extractor {
  type: 'ssd_resnet50_v1_fpn'
  fpn {
    min_level: 3
    max_level: 7
  }
  min_depth: 16
  depth_multiplier: 1.0
  conv_hyperparams {
    activation: RELU_6,
    regularizer {
      l2_regularizer {
        weight: 0.0004
      }
    }
    initializer {
      truncated_normal_initializer {
        stddev: 0.03
        mean: 0.0
      }
    }
    batch_norm {
      scale: true,
      decay: 0.997,
      epsilon: 0.001,
    }
  }
  override_base_feature_extractor_hyperparams: true
}
loss {
  classification_loss {
    weighted_sigmoid_focal {
      alpha: 0.25
```

```
          gamma: 2.0
        }
      }
      localization_loss {
        weighted_smooth_l1 {
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    normalize_loss_by_num_matches: true
    normalize_loc_loss_by_codesize: true
    post_processing {
      batch_non_max_suppression {
        score_threshold: 1e-8
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 100
      }
      score_converter: SIGMOID
    }
  }
}

train_config: {
  fine_tune_checkpoint: "C:/tese/insectdetector/workspace_2/pre_trained_model/ssd_res
  ↪  net50_v1_fpn_shared_box_predictor_640x640_coco14_sync_2018_07_03/model.ckpt"
  batch_size: 64
  sync_replicas: true
  startup_delay_steps: 0
  replicas_to_aggregate: 8
  num_steps: 25000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
```

```
      overlap_thresh: 0.0
    }
  }
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        cosine_decay_learning_rate {
          learning_rate_base: .04
          total_steps: 25000
          warmup_learning_rate: .013333
          warmup_steps: 2000
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_2/train.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_2/label_map.pbtxt"
}

eval_config: {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
  num_examples: 233
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_2/test.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_2/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

C.3 Configurações para Treinos do BDA 3

Faster R-CNN *Inception*-v2

```
# Faster R-CNN with Inception v2, configuration for MSCOCO Dataset.
# Users should configure the fine_tune_checkpoint field in the train config as
# well as the label_map_path and input_path fields in the train_input_reader and
# eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find the fields that
# should be configured.


model {
  faster_rcnn {
    num_classes: 5
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
    feature_extractor {
      type: 'faster_rcnn_inception_v2'
      first_stage_features_stride: 16
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
        scales: [0.25, 0.5, 1.0, 2.0]
        aspect_ratios: [0.5, 1.0, 2.0]
        height_stride: 16
        width_stride: 16
      }
    }
    first_stage_box_predictor_conv_hyperparams {
      op: CONV
      regularizer {
        l2_regularizer {
          weight: 0.0
        }
      }
      initializer {
        truncated_normal_initializer {
          stddev: 0.01
        }
```

```
    }
  }
  first_stage_nms_score_threshold: 0.0
  first_stage_nms_iou_threshold: 0.7
  first_stage_max_proposals: 300
  first_stage_localization_loss_weight: 2.0
  first_stage_objectness_loss_weight: 1.0
  initial_crop_size: 14
  maxpool_kernel_size: 2
  maxpool_stride: 2
  second_stage_box_predictor {
    mask_rcnn_box_predictor {
      use_dropout: false
      dropout_keep_probability: 1.0
      fc_hyperparams {
        op: FC
        regularizer {
          l2_regularizer {
            weight: 0.0
          }
        }
        initializer {
          variance_scaling_initializer {
            factor: 1.0
            uniform: true
            mode: FAN_AVG
          }
        }
      }
    }
  }
  second_stage_post_processing {
    batch_non_max_suppression {
      score_threshold: 0.0
      iou_threshold: 0.6
      max_detections_per_class: 100
      max_total_detections: 300
    }
    score_converter: SOFTMAX
  }
  second_stage_localization_loss_weight: 2.0
  second_stage_classification_loss_weight: 1.0
}
```

```
}

train_config: {
  batch_size: 1
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        manual_step_learning_rate {
          initial_learning_rate: 0.0002
          schedule {
            step: 900000
            learning_rate: .00002
          }
          schedule {
            step: 1200000
            learning_rate: .000002
          }
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  gradient_clipping_by_norm: 10.0
  fine_tune_checkpoint: "C:/tese/insectdetector/workspace_2/pre_trained_model/faster_
  ↪  rcnn_inception_v2_coco_2018_01_28/model.ckpt"
  from_detection_checkpoint: true
  # Note: The below line limits the training process to 200K steps, which we
  # empirically found to be sufficient enough to train the COCO dataset. This
  # effectively bypasses the learning rate schedule (the learning rate will
  # never decay). Remove the below line to train indefinitely.
  num_steps: 200000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_3/train.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_3/label_map.pbtxt"
```

```
}

eval_config: {
  num_examples: 319
  # Note: The below line limits the evaluation process to 10 evaluations.
  # Remove the below line to evaluate indefinitely.
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_3/test.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_3/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

Faster R-CNN *ResNet*-50

```
# Faster R-CNN with Resnet-50 (v1), configuration for MSCOCO Dataset.
# Users should configure the fine_tune_checkpoint field in the train config as
# well as the label_map_path and input_path fields in the train_input_reader and
# eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find the fields that
# should be configured.

model {
  faster_rcnn {
    num_classes: 5
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
    feature_extractor {
      type: 'faster_rcnn_resnet50'
      first_stage_features_stride: 16
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
        scales: [0.25, 0.5, 1.0, 2.0]
```

```
    aspect_ratios: [0.5, 1.0, 2.0]
    height_stride: 16
    width_stride: 16
  }
}
first_stage_box_predictor_conv_hyperparams {
  op: CONV
  regularizer {
    l2_regularizer {
      weight: 0.0
    }
  }
  initializer {
    truncated_normal_initializer {
      stddev: 0.01
    }
  }
}
first_stage_nms_score_threshold: 0.0
first_stage_nms_iou_threshold: 0.7
first_stage_max_proposals: 300
first_stage_localization_loss_weight: 2.0
first_stage_objectness_loss_weight: 1.0
initial_crop_size: 14
maxpool_kernel_size: 2
maxpool_stride: 2
second_stage_box_predictor {
  mask_rcnn_box_predictor {
    use_dropout: false
    dropout_keep_probability: 1.0
    fc_hyperparams {
      op: FC
      regularizer {
        l2_regularizer {
          weight: 0.0
        }
      }
      initializer {
        variance_scaling_initializer {
          factor: 1.0
          uniform: true
          mode: FAN_AVG
        }
```

```
        }
      }
    }
  }
  second_stage_post_processing {
    batch_non_max_suppression {
      score_threshold: 0.0
      iou_threshold: 0.6
      max_detections_per_class: 100
      max_total_detections: 300
    }
    score_converter: SOFTMAX
  }
  second_stage_localization_loss_weight: 2.0
  second_stage_classification_loss_weight: 1.0
 }
}

train_config: {
  batch_size: 1
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        manual_step_learning_rate {
          initial_learning_rate: 0.0003
          schedule {
            step: 900000
            learning_rate: .00003
          }
          schedule {
            step: 1200000
            learning_rate: .000003
          }
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  gradient_clipping_by_norm: 10.0
  fine_tune_checkpoint: "C:/tese/insectdetector/workspace_2/pre_trained_model/faster_
  ↪  rcnn_resnet50_coco_2018_01_28/model.ckpt"
  from_detection_checkpoint: true
```

```
  # Note: The below line limits the training process to 200K steps, which we
  # empirically found to be sufficient enough to train the pets dataset. This
  # effectively bypasses the learning rate schedule (the learning rate will
  # never decay). Remove the below line to train indefinitely.
  num_steps: 200000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_3/train.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_3/label_map.pbtxt"
}

eval_config: {
  num_examples: 319
  # Note: The below line limits the evaluation process to 10 evaluations.
  # Remove the below line to evaluate indefinitely.
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_3/test.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_3/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

SSD R-CNN *Inception*-v2

```
# SSD with Inception v2 configuration for MSCOCO Dataset.
# Users should configure the fine_tune_checkpoint field in the train config as
# well as the label_map_path and input_path fields in the train_input_reader and
# eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find the fields that
# should be configured.
```

```
model {
  ssd {
    num_classes: 5
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    anchor_generator {
      ssd_anchor_generator {
        num_layers: 6
        min_scale: 0.2
        max_scale: 0.95
        aspect_ratios: 1.0
        aspect_ratios: 2.0
        aspect_ratios: 0.5
        aspect_ratios: 3.0
        aspect_ratios: 0.3333
        reduce_boxes_in_lowest_layer: true
      }
    }
    image_resizer {
      fixed_shape_resizer {
        height: 300
        width: 300
      }
    }
```

```
box_predictor {
  convolutional_box_predictor {
    min_depth: 0
    max_depth: 0
    num_layers_before_predictor: 0
    use_dropout: false
    dropout_keep_probability: 0.8
    kernel_size: 3
    box_code_size: 4
    apply_sigmoid_to_scores: false
    conv_hyperparams {
      activation: RELU_6,
      regularizer {
        l2_regularizer {
          weight: 0.00004
        }
      }
      initializer {
        truncated_normal_initializer {
          stddev: 0.03
          mean: 0.0
        }
      }
    }
  }
}
feature_extractor {
  type: 'ssd_inception_v2'
  min_depth: 16
  depth_multiplier: 1.0
  conv_hyperparams {
    activation: RELU_6,
    regularizer {
      l2_regularizer {
        weight: 0.00004
      }
    }
    initializer {
      truncated_normal_initializer {
        stddev: 0.03
        mean: 0.0
      }
    }
```

```
        batch_norm {
          train: true,
          scale: true,
          center: true,
          decay: 0.9997,
          epsilon: 0.001,
        }
      }
      override_base_feature_extractor_hyperparams: true
    }
    loss {
      classification_loss {
        weighted_sigmoid {
        }
      }
      localization_loss {
        weighted_smooth_l1 {
        }
      }
      hard_example_miner {
        num_hard_examples: 3000
        iou_threshold: 0.99
        loss_type: CLASSIFICATION
        max_negatives_per_positive: 3
        min_negatives_per_image: 0
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    normalize_loss_by_num_matches: true
    post_processing {
      batch_non_max_suppression {
        score_threshold: 1e-8
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 100
      }
      score_converter: SIGMOID
    }
  }
}

train_config: {
```

```
  batch_size: 24
  optimizer {
    rms_prop_optimizer: {
      learning_rate: {
        exponential_decay_learning_rate {
          initial_learning_rate: 0.004
          decay_steps: 800720
          decay_factor: 0.95
        }
      }
      momentum_optimizer_value: 0.9
      decay: 0.9
      epsilon: 1.0
    }
  }
  fine_tune_checkpoint: "C:/tese/insectdetector/workspace_2/pre_trained_model/ssd_inc
  ↪  eption_v2_coco_2018_01_28/model.ckpt"
  from_detection_checkpoint: true
  # Note: The below line limits the training process to 200K steps, which we
  # empirically found to be sufficient enough to train the pets dataset. This
  # effectively bypasses the learning rate schedule (the learning rate will
  # never decay). Remove the below line to train indefinitely.
  num_steps: 200000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    ssd_random_crop {
    }
  }
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_3/train.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_3/label_map.pbtxt"
}

eval_config: {
  num_examples: 319
  # Note: The below line limits the evaluation process to 10 evaluations.
```

```
  # Remove the below line to evaluate indefinitely.
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_3/test.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_3/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

SSD R-CNN *ResNet*-50

```
# SSD with Resnet 50 v1 FPN feature extractor, shared box predictor and focal
# loss (a.k.a Retinanet).
# See Lin et al, https://arxiv.org/abs/1708.02002
# Trained on COCO, initialized from Imagenet classification checkpoint

# Achieves 35.2 mAP on COCO14 minival dataset. Doubling the number of training
# steps to 50k gets 36.9 mAP

# This config is TPU compatible

model {
  ssd {
    inplace_batchnorm_update: true
    freeze_batchnorm: false
    num_classes: 5
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
```

```
    ignore_thresholds: false
    negatives_lower_than_unmatched: true
    force_match_for_each_row: true
    use_matmul_gather: true
  }
}
similarity_calculator {
  iou_similarity {
  }
}
encode_background_as_zeros: true
anchor_generator {
  multiscale_anchor_generator {
    min_level: 3
    max_level: 7
    anchor_scale: 4.0
    aspect_ratios: [1.0, 2.0, 0.5]
    scales_per_octave: 2
  }
}
image_resizer {
  fixed_shape_resizer {
    height: 640
    width: 640
  }
}
box_predictor {
  weight_shared_convolutional_box_predictor {
    depth: 256
    class_prediction_bias_init: -4.6
    conv_hyperparams {
      activation: RELU_6,
      regularizer {
        l2_regularizer {
          weight: 0.0004
        }
      }
      initializer {
        random_normal_initializer {
          stddev: 0.01
          mean: 0.0
        }
      }
```

```
      batch_norm {
        scale: true,
        decay: 0.997,
        epsilon: 0.001,
      }
    }
    num_layers_before_predictor: 4
    kernel_size: 3
  }
}
feature_extractor {
  type: 'ssd_resnet50_v1_fpn'
  fpn {
    min_level: 3
    max_level: 7
  }
  min_depth: 16
  depth_multiplier: 1.0
  conv_hyperparams {
    activation: RELU_6,
    regularizer {
      l2_regularizer {
        weight: 0.0004
      }
    }
    initializer {
      truncated_normal_initializer {
        stddev: 0.03
        mean: 0.0
      }
    }
    batch_norm {
      scale: true,
      decay: 0.997,
      epsilon: 0.001,
    }
  }
  override_base_feature_extractor_hyperparams: true
}
loss {
  classification_loss {
    weighted_sigmoid_focal {
      alpha: 0.25
```

```
          gamma: 2.0
        }
      }
      localization_loss {
        weighted_smooth_l1 {
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    normalize_loss_by_num_matches: true
    normalize_loc_loss_by_codesize: true
    post_processing {
      batch_non_max_suppression {
        score_threshold: 1e-8
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 100
      }
      score_converter: SIGMOID
    }
  }
}

train_config: {
  fine_tune_checkpoint: "C:/tese/insectdetector/workspace_2/pre_trained_model/ssd_res
  ↪  net50_v1_fpn_shared_box_predictor_640x640_coco14_sync_2018_07_03/model.ckpt"
  batch_size: 64
  sync_replicas: true
  startup_delay_steps: 0
  replicas_to_aggregate: 8
  num_steps: 25000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
```

```
      overlap_thresh: 0.0
    }
  }
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        cosine_decay_learning_rate {
          learning_rate_base: .04
          total_steps: 25000
          warmup_learning_rate: .013333
          warmup_steps: 2000
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_3/train.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_3/label_map.pbtxt"
}

eval_config: {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
  num_examples: 319
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "C:/tese/insectdetector/workspace_2/dataset_3/test.record"
  }
  label_map_path: "C:/tese/insectdetector/workspace_2/dataset_3/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

Apêndice D

Link para *Gitlab* do Projeto

Link para *Gitlab*, com todos os arquivos do projeto, incluindo: imagens, BDAs, códigos em *Python* e o Aplicativo *Insect Detection*:

url = https://gitlab.estig.ipb.pt/a39316/insect-detection.git

Apêndice E

Publicações

Santos, F., Leitão, P., "Machine learning applied to sensing and monitoring insects in olive trees", 6° Encontro de Jovens Investigadores do IPB 2019. Bragança

Aprendizado de máquina aplicado na sensorização e monitoramento de insetos na cultura de oliveiras

Felipe V. dos Santos; Paulo Leitão[1]; José Barbosa; José A. Pereira; Frederic C. Janzen

[1]pleitao@ipb.pt, Instituto Politécnico de Bragança, Portugal

Resumo

O monitoramento de pragas em plantações possui extrema relevância e importância para avaliar e determinar a situação das pragas em uma lavoura. Por meio dele, avalia-se os danos e realiza-se o planejando da forma correta para aplicação de defensivos. Atualmente é realizado um monitoramento de uma plantação de oliveiras localizada nas proximidades da cidade de Porto mediante o uso de armadilhas autocolantes instaladas em pontos pré-estabelecidos. Esse processo possui como foco contar e identificar cinco tipos de insetos: trioza erytreae, syrphidae, coccinellidae, chrysopidae e philaenus. O modo tradicional de monitoração é realizado por intermédio de um especialista em insetos, o qual é incumbido da tarefa de identificar e contar os insetos manualmente. Essa prática ou modo de execução, constitui assim, num método moroso e trabalhoso. Nesse sentido, o presente trabalho se propõe a estudar e elaborar uma solução por intermédio de técnicas de aprendizagem de máquina e processamento de imagem com o objetivo de identificar e monitorar as cinco espécies de insetos. Utiliza-se, para esta finalidade, de redes neurais convolucionais capazes de reconhecer objetos a partir da extração de características de imagens, tendo como processo de aprendizagem a compilação de uma grande amostra de exemplos de insetos já classificados. Deste modo, inicialmente foi preciso fotografar as armadilhas e especificar os insetos acima mencionados para alimentação e posterior treinamento da rede neural. Para validar a proposta será feita a comparação da solução desenvolvida em contraste com a performance e precisão do modelo de identificação dos insetos tradicional.

Palavras-chave: aprendizagem de máquina; rede neurais convolucionais; oliveiras; monitorização; processamento de imagens; insetos.