Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Avaliação de Modelos YOLO para detectar Percevejos em Plantações de Soja

Ronald Cesar Dias de Oliveira

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Díbio Leandro Borges

Brasília
2023

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Avaliação de Modelos YOLO para detectar Percevejos em Plantações de Soja

Ronald Cesar Dias de Oliveira

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Díbio Leandro Borges (Orientador)
CIC/UnB

Profa. Roberta Barbosa Oliveira
CIC/UnB

Prof. Wilson Henrique Veneziano
CIC/UnB

Prof. João Luiz Azevedo de Carvalho
Coordenador do Curso de Engenharia da Computação

Brasília, 05 de Fevereiro de 2023

Dedicatória

Eu dedico esse trabalho primeiramente a *Deus*, que me deu a vida e vitalidade para que eu conseguisse conquistar esse objetivo de vida e aos meus pais, *Júlio César e Silvana* por terem me criado, influenciado positivamente na criação da minha personalidade e por todo o apoio dado durante toda a minha vida.

Agradecimentos

Eu agradeço a meu supervisor, *Prof. Dibio Leandro Borges* que me ajudou e me inspirou a continuar estudando essa área incrivel, agradeço ao *Dr. Edson Hirose* e à *Embrapa* por terem proporcionado os materiais necessários para que eu pudesse desenvolver esse trabalho. Quero agradecer aos meus Pais *Júlio Cesar e Silvana* por terem sido meus melhores amigos e dado força em todos os momentos que eu passei por dificuldades. Por último mas não menos importante, agradeço aos meus amigos que estiveram comigo durante essa jornada, espero ter feito diferença na vida deles, assim como eles fizeram na minha e aos amigos que tive que deixar de lado para focar nos estudos.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Devido à crescente demanda por técnicas de agricultura de precisão para melhor aproveitamento dos recursos disponíveis em campo, a aplicação de algoritmos de visão computacional é uma solução que trás uma ampla vantagem para auxiliar a detecção de pestes, possibilitando um tratamento mais eficaz e menos prejudicial à saude ambiental. Com base nessa afirmativa, este trabalho consiste em avaliar o modelo YOLO4 e comparar com as versões posteriores YOLOR, YOLOv5 e YOLOv7, que utilizam redes neurais convolucionais para detecção de objetos em tempo real. Estes foram treinados e utilizados para detectar Percevejos em imagens e vídeos em uma plantação de soja. Assim, utilizando métricas estatísticas, foram feitas considerações para definir o mais adequado modelo de acordo com a situação presente. Por meio de parceria com a Embrapa, foi utilizada uma base de dados que através de técnicas de aumento de dados foi possível transformar 77 imagens em 600. Dessa forma, o melhor modelo foi o YOLOv4 ao obter 99% de mAP@0.50, 99% de Precisão e 99% de Revocação em 60 imagens de validação.

Palavras-chave: R-CNN, FPN, YOLO, Visão Computacional, Agricultura de Precisão, Deep Learning

Abstract

Due to the growing demand for precision agriculture techniques to better utilize available resources in the field, the application of computer vision algorithms is a solution that brings a wide advantage to help the detection of pests, enabling a more effective and less harmful treatment to environmental health. Based on this statement, this work consists of evaluating the model YOLO4 and comparing to the posterior versions YOLOR, YOLOv5 and YOLOv7, which use convolutional neural networks for object detection in real time. These were trained and used to detect brown bugs on images and videos of a soybean plantation. Thus, it was used statistical metrics considerations to define the most appropriate model according to the present situation. Through a partnership with Embrapa, a database was used which, through data augmentation techniques it was possible to transform 77 images into 600. The best model was YOLOv4, obtaining 99% of mAP@0.50, 99% of Precision and 99% of Recall on 60 validation images.

Keywords: R-CNN, FPN, YOLO, Computer Vision, Precision Agriculture, Deep Learning

Sumário

1 Introdução **1**
1.1 Objetivo Geral 1
1.2 Objetivos Específicos 2
1.3 Organização do Trabalho 2

2 Fundamentação Teórica **3**
2.1 Rede Neural 3
2.2 Aprendizado Supervisionado 4
2.3 Rede Neural Convolucional 5
2.4 Rede Convolucional baseada em Regiões e suas Derivações 6
2.5 Rede Piramidal de Atributos 8
2.6 YOLO 9
2.7 YOLOv4 11
2.8 Métricas de Avaliação 13
2.8.1 Matriz de Confusão 13
2.8.2 Precisão 14
2.8.3 Revocação 14
2.8.4 mAP 15
2.9 Revisão de Literatura 15

3 Materiais e Métodos **17**
3.1 Base de Dados 17
3.2 Modelos Utilizados 17
3.3 Pré-Processamento 18
3.4 Configuração 21
3.5 Equipamento Utilizado 23
3.6 Funções Customizadas 24

4 Resultados e Discussão **25**
4.1 Treinamento 25

4.1.1 YOLOv4 26
4.1.2 YOLOR 28
4.1.3 YOLOv5 30
4.1.4 YOLOv7 32
4.2 Aplicação do Modelo 34
4.3 Discussão 36

5 Conclusão **38**
5.1 Trabalhos Futuros 39

References **40**

Lista de Figuras

2.1 Arquitetura de uma Rede Neural. (Fonte: Adaptada de [1]) 4

2.2 Arquitetura de uma CNN que contém uma imagem de entrada, camadas de convolução, camada de *pooling* e uma *fully connected layer*, gerando o resultado final. (Fonte: Adaptada de [2]). 5

2.3 Arquitetura da R-CNN que contém as etapas de entrada de imagem, extração das propostas de região, redimensionamento do mapa de atributos e computação dos recursos da CNN e classificação das regiões. (Fonte: Adaptada de [2]). 7

2.4 Arquitetura da Fast R-CNN, contendo uma imagem de entrada onde atributos de uma *Deep ConvNet* e uma projeção de RoI dessa imagem compõem um Conv feature map e para cada RoI, utiliza-se a *RoI* pooling layer para agrupar os atributos que são assim detectados e classificados.. 7

2.5 Arquitetura da Faster R-CNN, contendo uma imagem que passa por *Conv Layers* e na RPN, são geradas as *Proposals*, podendo assim serem classificadas. (Fonte: Adaptada de [3]). 8

2.6 Comparação entre as Arquiteturas Piramidais de extração dos Mapas de Atributos. a) Redimensionamentos de imagem e convoluções sem redimensionamento em cada nível, onde são feitas previsões nos mapas de atributos. b) Convoluções com redimensionamento dos mapas de atributos, fazendo previsão no último nível. c) Convoluções com redimensionamento dos mapas de atributos, fazendo previsões em cada nível. d) Convoluções com redimensionamento no caminho de baixo pra cima, aumento do mapa de atributos no caminho de cima para baixo com conexões laterais e concatenação entre cada nível, as previsões são feitas em cada nível. (Fonte: Adaptada de [4]) 9

2.7 Visão geral da arquitetura do método YOLO, contendo uma imagem que é dividida em uma grade no formato S × S, gerando de forma simultânea as *bounding boxes* (Caixas de delimitação) com sua *confidence* (confiança) e o *Class probability map* (Mapa de Probabilidades de Classe), permitindo assim a detecção final. (Fonte: Adaptada de [5]) 11
2.8 Arquitetura de detectores de objetos de um e dois estágios (Fonte: Adaptada de [6]) 12
2.9 Na Matriz de confusão, em verde estão as predições corretas do modelo, em vermelho as predições incorretas. 14

3.1 Amostra da Base de Dados contendo um Percevejo.. 18
3.2 Anotação das coordenadas de localização do Percevejo.. 18
3.3 Código de separação da base de dados. 19
3.4 Código para a obtenção do caminho das imagens. 19
3.5 Código para Data Augmentation. 20
3.6 Amostras da base de dados, em que a imagem da esquerda é a imagem original e a da esquerda foi manipulada pela rotação em sentido anti-horário, teve seu brilho alterado e preenchida por meio da cópia dos *pixels* dos vizinhos mais próximos. 21
3.7 Parâmetros do arquivo de configuração *cfg* do YOLOv4 22
3.8 Configuração do arquivo *.yaml* do YOLOR. 23
3.9 Configuração dos parâmetros do YOLOv5. 23

4.1 Resultados do treinamento sem *data augmentation*. 26
4.2 Resultados do treinamento com *data augmentation*. 27
4.3 Inferência que detectou um dos percevejos do casal, provinda dos pesos do treinamento sem *data augmentation*. 27
4.4 Inferência que detectou dois dos percevejos do casal, provinda dos pesos do treinamento com *data augmentation*. 28
4.5 Resultados do treinamento sem *data Augmentation*. 28
4.6 Inferência que detectou dois dos percevejos do casal, provinda dos pesos do treinamento sem *data augmentation*. 29
4.7 Resultados do treinamento com *data Augmentation*. 29
4.8 Inferência que detectou dois dos percevejos do casal, provinda dos pesos do treinamento com *data augmentation*. 30
4.9 Resultados do treinamento sem *data augmentation*. 30
4.10 Resultados do treinamento com *data Augmentation*. 31

4.11 Inferência que detectou dois dos percevejos do casal, provinda dos pesos do treinamento sem *data augmentation*. 31
4.12 Inferência que detectou dois dos percevejos do casal, provinda dos pesos do treinamento com *data augmentation*. 32
4.13 Resultados do treinamento sem *data augmentation*. 32
4.14 Inferência que detectou um dos percevejos do casal, provinda dos pesos do treinamento sem *data augmentation*. 33
4.15 Resultados do treinamento com *data augmentation*. 33
4.16 Inferência que detectou dois dos percevejos do casal, provinda dos pesos do treinamento com *data augmentation*. 34
4.17 Inferência que detecta um Percevejo na plantação. 35
4.18 Inferência que detecta dois Percevejos na plantação. 35
4.19 Inferência que detecta um casal de percevejos na plantação. 36
4.20 Inferência que detecta apenas 1 percevejo do casal de percevejos na plantação. 36

Lista de Tabelas

4.1 Configurações de Treinamento dos modelos YOLO. 25
4.2 Resultados treinamento sem *data augmentation*. 34
4.3 Resultados treinamento com *data augmentation*. 34

Lista de Abreviaturas e Siglas

CNN Convolutional Neural Network.

Fast R-CNN Fast Region based Convolutional Network.

Faster R-CNN Faster Region based Convolutional Network.

FC Fully Connected.

FPN Feature Pyramidal Network.

GPU Graphics Process Unit.

IoU Intersection over Union.

R-CNN Region based Convolutional Neural Network.

RoI Regions of Interest.

RPN Region Proposal Network.

SS Selective Search.

YOLO You Only Look Once.

YOLOR You Only Learn One Representation.

YOLOv2 You Only Look Once version 2.

YOLOv3 You Only Look Once version 3.

YOLOv4 You Only Look Once version 4.

YOLOv5 You Only Look Once version 5.

YOLOv7 You Only Look Once version 7.

Capítulo 1

Introdução

A Agricultura Brasileira evoluiu consideravelmente nos últimos 40 anos devido ao emprego das tecnologias [7]. O Brasil saiu do status de importador para provedor de alimentos, se tornando o maior produtor de soja do mundo [8]. Essas tecnologias foram desenvolvidas e empregadas por meio de investimento em pesquisas, porém, ainda há muito o que melhorar. De acordo com a Embrapa [7], cerca de 100 milhões de hectares de terra estão totalmente improdutivas, devido ao uso inadequado de agrotóxicos, má irrigação, entre outras más práticas. Estima-se que a população mundial vai crescer para 8,5 bilhões de pessoas até 2030 [9], com isso, a demanda de grãos vai consequentemente aumentar. Dessa forma, uma estratégia que vem sendo utilizada é a aplicação de técnicas de agricultura de precisão [10], aproveitando o máximo possível dos recursos e tratando diversas formas de adversidades como o controle da umidade do solo, tratamento de doenças e controle de pestes. Visando a diminuição e até um futuro desuso do agrotóxico como controle de pestes, o proposto trabalho busca formas eficazes de detectar, em tempo real [1], a praga que mais assola as plantações de soja: o *Euschistus Heros* (Percevejo Marrom). Tendo como referência trabalhos relacionados [11, 12], foi selecionado o YOLOv4 [6] para avaliar em comparação prática com três versões posteriores e assim aplicar o que obteve melhor resultado na contagem de percevejos.

1.1 Objetivo Geral

O objetivo desse trabalho foi fazer uma avaliação comparativa entre algumas entre as mais atuais tecnologias no ramo do *Deep Learning* (aprendizado de máquina profundo), permitindo a escolha de um modelo que visa aplicar técnicas de visão computacional para detectar percevejos em tempo real, diretamente nas plantações de soja.

[1] Termo utilizado para definir a não percepção aos olhos humanos, utilizando a taxa de 30 quadros por segundo como referência

1.2 Objetivos Específicos

Para isso, serão feitas comparações entre os mais atuais modelos de deteção de objetos, a fim de observar o que melhor atende à demanda de detecção em tempo real de percevejos em campo, visto que há uma dificuldade relativa aos diversos elementos presentes em plantações como insetos, manchas sobre as folhas, diferentes fases de iluminação e situações climáticas.

1.3 Organização do Trabalho

No Capítulo 2 será abordada a fundamentação teórica, uma breve análise da evolução dos algoritmos de visão computacional no âmbito do aprendizado profundo de máquina, Posteriormente, no Capítulo 3 serão apresentados os materiais e métodos utilizados e implementados, Já no Capítulo 4 serão apresentados os resultados obtidos juntamente com sua respectiva análise e a discussão sobre o que foi alcançado por meio dos modelos utilizados e no Capítulo 5 serão apresentados a conclusão do que foi alcançado com o trabalho e quais são as próximas etapas a serem desenvolvidas, dando continuidade ao mesmo.

Capítulo 2

Fundamentação Teórica

Um sistema de detecção de objetos em tempo real utiliza técnicas aprimoradas de aprendizado profundo de máquina para localizar e classificar objetos em uma fonte de entrada (foto ou vídeo). A fim de, teoricamente, situar o leitor sobre o que foi alcançado por meio das pesquisas, serão apresentados conceitos a serem utilizados em parte das técnicas nas demais sessões.

2.1 Rede Neural

Rede Neural, segundo a *IBM Cloud Education* [1], trata-se de uma rede interconectada de neurônios que simulam o funcionamento do cérebro humano. É uma técnica que faz parte de uma subárea do *Machine Learning* (Aprendizado de Máquina), o *Deep Learning* (Aprendizado Profundo). Na Figura 2.1 é possível observar o esboço da arquitetura de uma rede neural, onde os círculos tratam-se dos neurônios e as linhas que interconectam os neurônios, os pesos. A camada mais a esquerda trata-se da *input layer* (camada de entrada) e a última da direita a *output layer* (camada de saída). Essas camadas intermediárias são as *hidden layers* (camadas ocultas), as quais costumam ser responsáveis por extrair os atributos da *input layer* da rede e aplicar operações para tratar esses atributos, passando adiante para a obtenção do resultado final na *output layer*. O grande objetivo de uma rede neural é aprender a reconhecer padrões dos dados de entrada e fazer previsões em cima de novos dados, para isso, treina-se a rede até chegar no objetivo desejado. A fim de obter a atualização dos pesos dos neurônios para reduzir os erros cometidos em previsões da rede, o treinamento consiste em uma primeira passagem pelos neurônios da rede para posteriormente utilizar técnicas de atualização dos pesos como a *Backpropagation* (retropropagação) [13] que usa uma técnica como o *Gradient Descent* (Gradiente Descendente) para calcular qual será o novo valor de cada peso da rede até ser possível prever, diante de uma entrada, qual o valor da saída com o mínimo de erro

possível. O *Gradient Descent* consiste em calcular a derivada de erro com relação a cada peso e atualizar o valor do mesmo por meio de uma redução escalar negativa sobre o peso do valor do erro calculado. Com relação ao termo *Deep*, a nomenclatura se refere a profundidade das redes neurais que com o uso de *hidden layers*, passaram a ter 3 ou mais camadas.

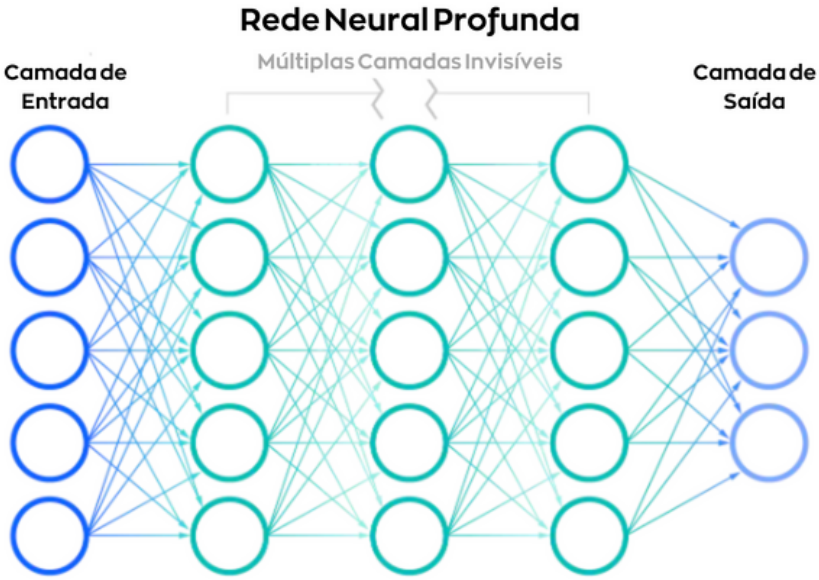

Figura 2.1: Arquitetura de uma Rede Neural. (Fonte: Adaptada de [1])

2.2 Aprendizado Supervisionado

De Acordo com Tom Mitchell [14], um programa aprende de uma expência E, por meio de alguma tarefa T e alguma medida de performance P, se a performance em T, medido por P, melhora a experiência em E. Dessa forma, no contexto do aprendizado supervisionado, as redes neurais que serão apresentadas a seguir, utilizam essa premissa para aprender. A experiência diz respeito ao conhecimento prévio da rede neural, tratando-se assim dos pesos utilizados nos neurônios. Cada iteração do treinamento sobre a rede neural refere-se a tarefa a ser realizada. As anotações sobre as imagens utilizadas no treinamento da rede, cuja base de imagens contém essas anotações das coordenadas da localização dos objetos em cada imagem, servem de parâmetros de comparação entre o obtido e o esperado para a medida de performance do programa. Dessa forma, cada rede neural utiliza diferentes estratégias para melhorar sua performance nos treinamentos e os modelos são avaliados através de métricas estatísticas. Alguns dos modelos e métricas utilizadas para avaliação dos mesmos serão mais detalhados posteriormente nesse mesmo Capítulo.

2.3 Rede Neural Convolucional

Em 1980 foi criada a primeira *Convolutional Neural Network* (CNN) [15] ou Rede Neural Convolucional em tradução livre, por Yan Le Cunn, Cientista da Computação e hoje Professor na New York University (NYU). Uma CNN trata-se de uma rede neural que possui camadas de Convolução, camadas de *Pooling* (agrupamento) e uma *Fully Connected layer* (FC layer), Camada Totalmente Conectada em tradução livre, para Classificação. Convolução é uma operação matemática amplamente usada no processamento de sinais, tratando-se de uma multiplicação de duas funções no domínio da frequência, gerando uma terceira função que é o resultado de como uma das funções modifica o formato da outra. No processamento de imagens, a convolução é responsável por extrair atributos das imagens que de acordo com o filtro que é utilizado na convolução, é possível obter características como bordas, contornos, texturas, entre outras. Essas características contém informações que permitem por meio de outras camadas da rede neural, como a de *pooling*, fazer uma seleção desses atributos, agrupando as informações consideradas mais relevantes para aquela rede e por fim fazer a classificação utilizando a FC. Um exemplo de CNN pode ser vista na Figura 2.2.

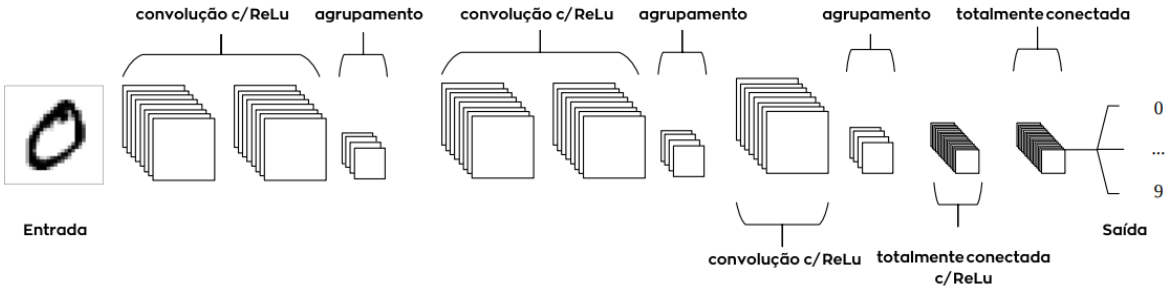

Figura 2.2: Arquitetura de uma CNN que contém uma imagem de entrada, camadas de convolução, camada de *pooling* e uma *fully connected layer*, gerando o resultado final. (Fonte: Adaptada de [2]).

Apesar da CNN ter sido capaz de reconhecer dígitos escritos a mão de forma eficiente, as *ConvNets* [1] não tiveram um espaço de aplicação amplo na época de sua implementação, devido a necessidade massiva de dados de treinamento e recursos computacionais, fazendo com que o sistema não fosse escalável para tarefas mais complexas. Apenas com a criação da *AlexNet* [16] que ganhou o concurso *ImageNet* [17] em 2012, que a área renomeada para

[1] Os termos CNN e ConvNets serão utilizados indiscriminadamente ao longo do texto.

Deep Learning, teve uma atenção dos pesquisadores e ascendeu nas pesquisas, gerando um grande avanço para área por meio das técnicas que foram desenvolvidas ao longo dos anos posteriores.

2.4 Rede Convolucional baseada em Regiões e suas Derivações

Com a criação da *Region based Convolutional Neural Network* (R-CNN) [2], Rede Neural Convolucional baseada em Regiões, em tradução livre, a Localização e Classificação dos objetos subiu para outro patamar. Antes, as *ConvNets* classificavam as imagens que continham apenas um tipo de objeto, o processo de localização dos objetos fazia parte de outro grupo de algoritmos de visão computacional. Como a diversidade de objetos pode ser muito ampla e deseja-se detectar diferentes objetos em uma mesma imagem, se viu necessário separar as regiões das imagens em que esses objetos podem estar localizados e utilizar a CNN para classificar essas regiões. Dessa forma, foi criada uma arquitetura unificada para localizar e classificar os objetos em imagens. Para definir as regiões prováveis de localização dos objetos, foi proposto o uso de 2000 regiões, essas regiões foram chamadas de *Region Proposals* (Propostas de Região). Na época, um método muito utilizado foi o *OverFeat* [18], que utiliza uma abordagem de deslize de janela em multiescalas para obter a localização dos objetos sem a mudança de dimensões da imagem, desenhando-se *Bounding Boxes* (Caixas de Delimitação) sobre a localização dos objetos na imagem e utilizando a CNN apenas para classificar aquele objeto. O método R-CNN em contrapartida utilizou o *Selective Search* (SS) [19], Procura Seletiva em tradução livre, [19], para agrupar 2000 regiões em segmentos de acordo com a similaridade entre os *pixels* da imagem em que está sendo aplicado. Para cada proposta de região, é extraído um vetor de atributos através de uma CNN, sendo assim classificado. Mesmo sendo cerca de 9 vezes mais lento, o método R-CNN superou por uma grande margem a acurácia da detecção de objetos do *OverFeat*. A lentidão é devido ao processo do SS ser lento e se tratar de um algoritmo fixo, ou seja, não há aprendizado ao longo do seu uso no treinamento. Uma visão geral da arquitetura do método R-CNN está esboçada na Figura 2.3.

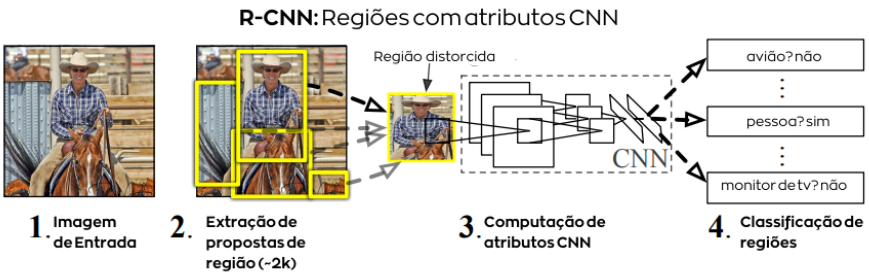

Figura 2.3: Arquitetura da R-CNN que contém as etapas de entrada de imagem, extração das propostas de região, redimensionamento do mapa de atributos e computação dos recursos da CNN e classificação das regiões. (Fonte: Adaptada de [2]).

A Fast R-CNN [20] baseou-se nas limitações da R-CNN com relação a grande quantidade de regiões para cada imagem e a lentidão para fazer a detecção dos objetos. A estratégia foi usar como entrada um imagem, juntamente de um conjunto de Propostas de Regiões, selecionado-se uma CNN para gerar um *convolutional feature map* (mapa convolucional de atributos) extraído da imagem e por meio das propostas de regiões são extraidos, através da *Regions of Interest Pooling Layer* (RoI Pooling Layer), Camada de Agrupamento de Regiões de Interesse em tradução livre, vetores fixos de atributos, usados para alimentar uma sequência de FCs, responsáveis pelo processo de regressão das *bounding boxes*, ou seja, determinar os valores das coordenadas das *bounding boxes* e qual a classificação daqueles objetos. Uma visão geral da arquitetura da Fast R-CNN pode ser vista na Figura 2.4.

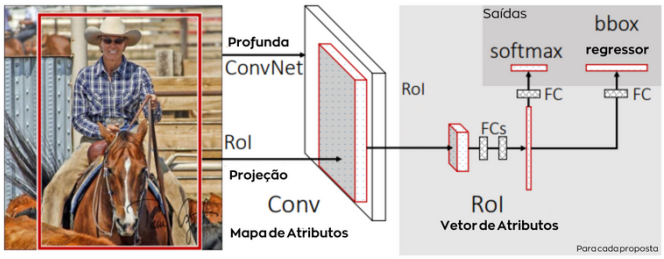

Figura 2.4: Arquitetura da Fast R-CNN, contendo uma imagem de entrada onde atributos de uma *Deep ConvNet* e uma projeção de RoI dessa imagem compõem um Conv feature map e para cada RoI, utiliza-se a *RoI* pooling layer para agrupar os atributos que são assim detectados e classificados. (Fonte: [20]).

A fim de acelerar o processo de obtenção das regiões, na Faster R-CNN [3], mesmo de arquitetura similar à Fast R-CNN com relação a obtenção do mapa de atributos por meio da CNN, ao invés de utilizar o *Selective Search* para obtenção das regiões implementou a *Region Proposal Network* (RPN), Rede de Proposta de Região em tradução livre. A RPN

é responsável por fazer a predição ao mesmo tempo da *bounding box* e a pontuação de objetividade (objectness score) utilizando o conceito de *anchors* (âncoras) que tratam-se de *bounding boxes* de diferentes escalas e razões. Após essa etapa, as propostas de regiões são utilizadas na mesma abordagem da tecnologia anterior, Fast R-CNN. Com isso, foi possível compartilhar atributos provindos das convoluções entre a extração desses atributos e a rede de detecção, de forma a permitir a atualização dos pesos de todas as camadas em tempo de treinamento, criando-se assim uma Rede Neural Convolucional Unificada baseada em Regiões. Esse compartilhamento de atributos fez com que as propostas de região se tornassem praticamente *cost-free* (livres de custo), acelerando consideravelmente a deteção de objetos. Sua arquitetura pode ser vista na Figura 2.5.

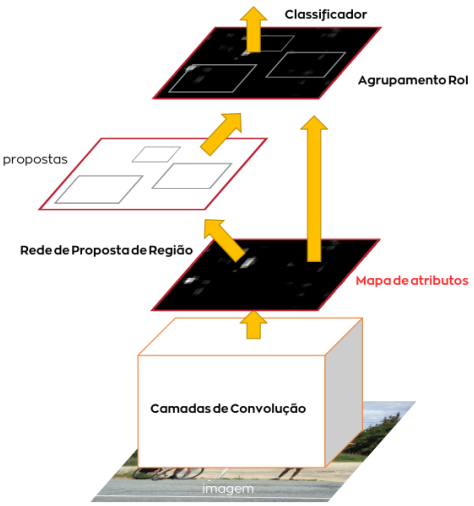

Figura 2.5: Arquitetura da Faster R-CNN, contendo uma imagem que passa por *Conv Layers* e na RPN, são geradas as *Proposals*, podendo assim serem classificadas. (Fonte: Adaptada de [3]).

2.5 Rede Piramidal de Atributos

A *Feature Pyramidal Network* (Rede Piramidal de Atributos) [4] é uma técnica para melhor aproveitar as informações dos atributos extraidos das imagens, melhorando tanto os métodos de propostas de regiões como a RPN, quanto na classificação das regiões como a Fast R-CNN. Consistindo nos caminhos *bottom-up* e *top-down with lateral connections* (de baixo para cima e de cima para baixo com conexões laterais), a extração de atributos de uma imagem de entrada utiliza o conceito de *Pyramid of Images* (piramide de imagens), que apesar de poderoso, é um método que aumenta o tempo de computação e o espaço necessário para armazenar os resultados.

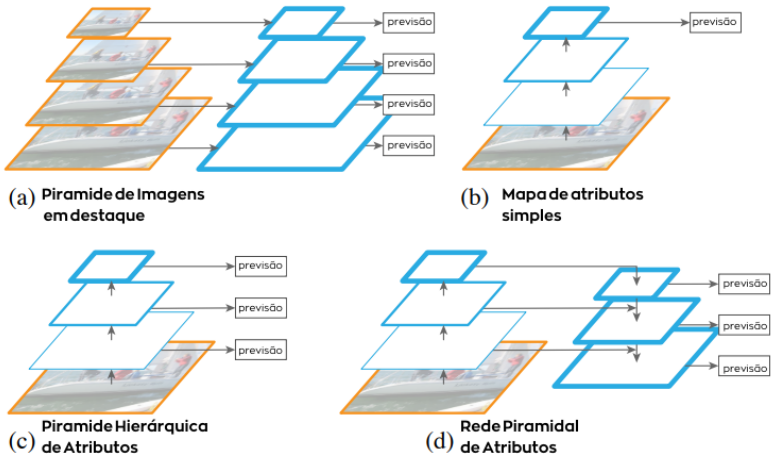

Figura 2.6: Comparação entre as Arquiteturas Piramidais de extração dos Mapas de Atributos. a) Redimensionamentos de imagem e convoluções sem redimensionamento em cada nível, onde são feitas previsões nos mapas de atributos. b) Convoluções com redimensionamento dos mapas de atributos, fazendo previsão no último nível. c) Convoluções com redimensionamento dos mapas de atributos, fazendo previsões em cada nível. d) Convoluções com redimensionamento no caminho de baixo pra cima, aumento do mapa de atributos no caminho de cima para baixo com conexões laterais e concatenação entre cada nível, as previsões são feitas em cada nível. (Fonte: Adaptada de [4])

Na Figura 2.6 é possível ver a comparação entre métodos de extração de atributos piramidais (a), (b), (c), para o utilizado na FPN (d). Sabendo que as convoluções costumam redimensionar o tamanho da imagem de entrada, o caminho *bottom-up* reduz o tamanho (*down-sampling*) para obter melhores informações semânticas da imagem, ou seja, informações relevantes para classificação dos objetos, já o caminho *top-down* com conexões laterais produz um aumento de tamanho (*up-sampling*) aumentando-se assim a resolução dos atributos. As conexões laterais servem para a obtenção de informações mais relevantes para localização dos objetos, visto que os primeiros níveis da piramide, possuem informações mais fracas semânticamente e mais fortes de localização espacial. Dessa forma, os mapas de atributos são concatenados e esse processo é iterado até produzir o melhor mapa de atributos.

2.6 YOLO

YOLO [5] é um método desenvolvido para detectar objetos utilizando como slogan a frase "*You Only Look Once*" (Você apenas olha uma vez). Foi uma abordagem que fez com que o processo de propostas de região e classificação de objetos fosse simultânea. O principal objetivo desse método foi transformar a deteção de objetos em apenas um problema de

regressão, dessa forma, a arquitetura foi desenhada para unificar os dois estágios até então utilizados. Considerando a forma de treinamento supervisionado, o método YOLO consiste em dividir uma imagem de entrada em uma grade $S \times S$, onde S é um número escolhido, cada célula (quadrado da grade), proveniente da divisão, que possui o centro do objeto a ser detectado em seu interior, é responsável por detectar aquele objeto. Dessa forma, cada célula é responsável por fazer 2 tipos de previsões, uma para *bounding boxes* e suas respectivas pontuações de confiança e outra para um mapa de probabilidades do tipo de classe daquele objeto. No primeiro caso a célula faz uma previsão de B caixas de delimitação, esse número B de caixas pode ser escolhido. Cada *bounding box* possui 5 previsões, x e y são as coordenadas do centro da caixa, w e h são a largura e espessura da caixa e por fim o valor de *confidence* (confiança) calculada através da fórmula $Pr(Object) * IOU^{\text{pred}}_{\text{truth}}$, onde $Pr(Object)$ trata-se da probabilidade da caixa de delimitação conter o objeto e $IOU^{\text{pred}}_{\text{truth}}$ trata-se da *Intersection Over Union* (interseção sobre união) entre a *bounding box* e a *ground truth box* (caixa verdade), que foi delimitada previamente por meio das anotações como citado na seção 2.2. Já no segundo caso, a célula faz a previsão de C probabilidades condicionais de classes através da fórmula $Pr(Class_i|Object)$, que trata-se da probabilidade do objeto ser de cada classe C, onde C é a quantidade de classes que deseja-se classificar. O conjunto de probabilidades independe do número de *bounding boxes*, apenas a quantidade de classes que define o cálculo. Uma visão geral dessa arquitetura pode ser observada na Figura 2.7. Apesar de velocidade ser uma das características mais relevantes desse método, a acurácia para detecção de pequenos objetos ou objetos que aparecem em grupos ou em diferentes proporções, diminuiu. Dessa forma, assim como nos métodos citados anteriormente, as limitações foram objetos de pesquisas, a cada versão de lançamento de modelos posteriores, alguma dessas debilidades foram tratadas e outras foram geradas. Apesar de ser uma proposta que revolucionou a forma de detectar objetos, alguns dos conceitos até então utilizados nos modelos anteriores de dois estágios foram resgatados para aprimorar essa ideia de detecção em um estágio e as versões posteriores propuseram novos métodos para diminuir cada vez mais a troca entre velocidade de detecção e acurácia dos modelos.

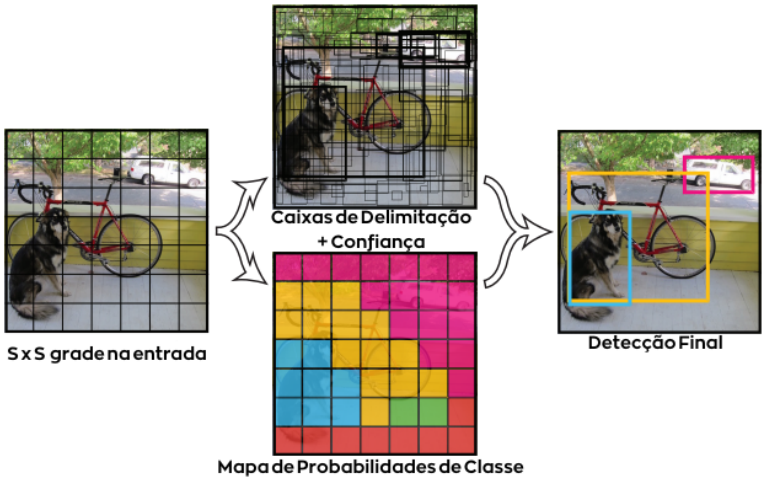

Figura 2.7: Visão geral da arquitetura do método YOLO, contendo uma imagem que é dividida em uma grade no formato S × S, gerando de forma simultânea as *bounding boxes* (Caixas de delimitação) com sua *confidence* (confiança) e o *Class probability map* (Mapa de Probabilidades de Classe), permitindo assim a detecção final. (Fonte: Adaptada de [5])

2.7 YOLOv4

A arquitetura de detectores de objetos de um estágio e dois estágios pode ser vista na Figura 2.8, onde a *backbone* (Espinha Dorsal) é responsável pela extração de atributos, o *neck* (Pescoço) é responsável por aprimorar os atributos entre a *backbone* e a *head* (Cabeça) que trata-se do detector, na *sparse prediction* (Previsão Esparsa), acontece a detecção em dois estágios, utilizando os atributos da *neck* e da *dense prediction* como sendo o *convolutional feature map* e as *region proposals* respectivamente. Os métodos YOLO utilizam a abordagem de apenas um estágio.

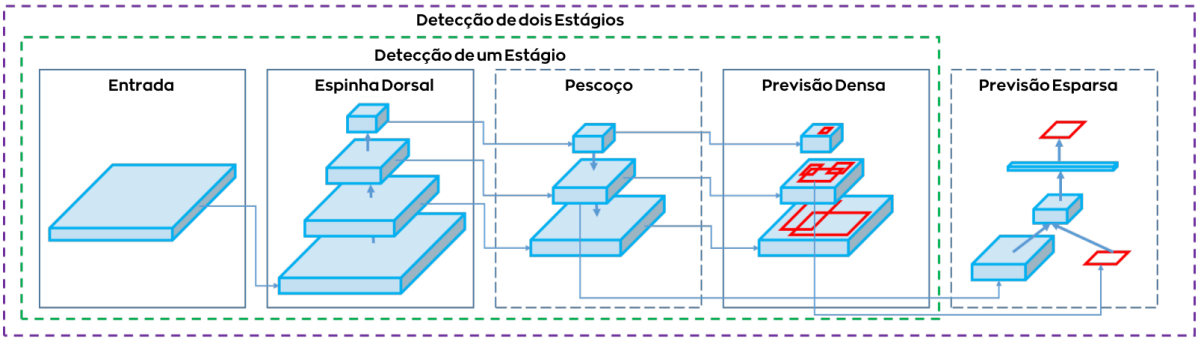

Figura 2.8: Arquitetura de detectores de objetos de um e dois estágios (Fonte: Adaptada de [6])

As versões anteriores, YOLOv2 [21] e YOLOv3 [22], trouxeram melhorias relevantes. Ao invés de seguir a escolha da VGG-16 [23] como extrator de atributos, como os demais métodos contemporâneos à YOLOv2, foi escolhido a Darknet [24], rede neural *open source* (código aberto) escrita na linguagem de programação C , que foi customizada para conter 19 camadas de convolução, sendo chamada de Darknet 19. Uma das recuperações de conceitos dos métodos anteriores foi o uso das *anchors*, utilizado na RPN para definir as *region proposals* 2.4. Para serem utilizadas nas convoluções, ao invés de fazer a predição das *bounding boxes* diretamente, as *anchors* são usadas para fazer as predições das *bounding boxes*. A fim de melhorar a localização de pequenos objetos, um refinamento dos atributos extraídos foi aproveitar informações dos mapas de atributos de camadas anteriores de convolução através da *passthrough layer* (camada de passagem) concatenando níveis menos refinados com níveis mais refinados dos mapas de convolução, similar à construção da FPN na seção 2.5. A Darknet sofreu mudanças na passagem da versão 2 para a versão 3, passando a ter 53 camadas de convolução, visando a maior acurácia do modelo. Porém, essa evolução trouxe uma melhora relativa na deteção de objetos pequenos e piora na performance dos objetos médios e grandes. A YOLOv4 [6], buscando evoluir as versões anteriores, é a prova de que é possível criar um detector de objetos rápido, preciso e robusto, apenas com uma *Graphics Process Unit* (GPU), Unidade Gráfica de Processamento em tradução livre. Através de estudos aprofundados com relação aos métodos anteriores, foi verificado que algumas das combinações do uso de técnicas se mostravam boas apenas para alguns modelos. Dois conceitos importantes que foram definidos por meio da divisão proposta entre etapas da deteção de objetos foi a *Bag of Freebies* (Saco de Presentes) e *Bag of Specials* (Saco de Especiais).

Bag of Freebies trata-se de um conjunto de métodos responsáveis pelo uso de estratégias de treinamento ou por aumentar de alguma forma o custo de treinamento para

aumentar a acurácia do modelo. Esses métodos podem ser aplicados na *backbone* ou então no detector. A grande parte desses métodos utiliza o conceito de *Data Augmentation* (Aumento de Dados) [25], que são mudanças nas imagens para gerar versões modificadas dessas imagens. Algumas dessas mudanças consistem em distorções geométricas como mudança de escala, cortes, inversões e rotações ou fotométricas como mudança de brilho, contraste, matiz, saturação e ruídos. Devido ao uso de *data augmentation*, novas situações são apresentadas para a rede neural, permitindo assim a criação de um sistema mais robusto.

Bag of Specials é também um conjunto de métodos que podem ser aplicados na *backbone* e no detector, porém, atuam no tempo pós treinamento, ou seja, na utilização do modelo para previsões, chamado de inferência. Esses métodos têm como objetivo aumentar a performance de detecção, adicionando tempo de inferência.

A partir desses conceitos, os pesquisadores que desenvolveram a YOLOv4, visando construir um bom modelo que precisa de apenas uma GPU, fizeram algumas implementações e modificações nos métodos já existentes. Alguns dos mais relevantes foi a implementação de técnicas de *data agumentation* como *Mosaic* (mosaico), que mistura 4 imagens em uma só e o *Self-Adversarial Training* (Treinamento Auto-Adversário), baseia-se em 2 estágios em que no primeiro estágio a rede altera a imagem ao invés dos pesos, criando a ilusão para a rede de que não existe aquele objeto naquela imagem, no segundo estágio a rede é treinada para detectar um objeto na imagem modificada. YOLOv4 ainda é um modelo eficiente e plenamente utilizado devido a sua velocidade e precisão.

2.8 Métricas de Avaliação

2.8.1 Matriz de Confusão

Matriz de Confusão, Figura 2.9, trata-se de uma tabela para avaliar a performance de modelos utilizando os conceitos de quantificar quantas foram as predições verdadeiras e falsas comforme a base de dados, para isso são usadas as nomenclaturas *True Positive* (TP) , Verdadeiro Positivo, *True Negative* (TN), Verdadeiro Negativo, *False Positive* (FP), Falso Positivo e *False Negative* (FN), Falso Negativo. Os verdadeiros são relativos ao elemento a ser observado, já os falsos são o restante dos elementos.

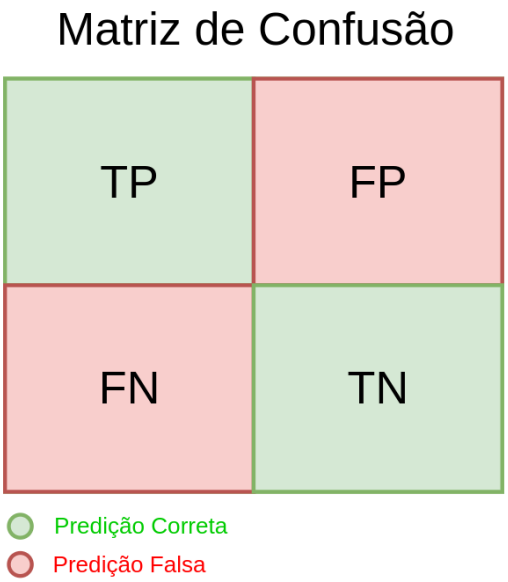

Figura 2.9: Na Matriz de confusão, em verde estão as predições corretas do modelo, em vermelho as predições incorretas.

A partir da Matriz de Confusão, por meio das métricas Precisão e Revocação, é possível avaliar a qualidade do modelo.

2.8.2 Precisão

A Precisão é medida através da Equação 2.1 e mensura no modelo quantos elementos foram corretamente classificados, levando em conta todos os elementos que foram avaliados como verdadeiros, o elemento desejado e os demais que foram falsamente apontados como verdadeiros.

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

2.8.3 Revocação

Já a Revocação, medida através da Equação 2.2, mensura no modelo quantos elementos foram corretamente classificados, levando em conta todos do elemento desejado que foram corretamente classificados e todos também do elemento desejado que não foram corretamente classificados.

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

2.8.4 mAP

O mAP é uma métrica de avaliação amplamente utilizada no *Deep Learning*, pois, além de avaliar os modelos por meio da Precisão e Revocação, fazendo uma média entre as precisões para diferentes valores de revocação, trás resultados de acordo com a acurácia entre N classes a serem analisadas, para isso utiliza o conceito de IoU, que é o limiar mínimo definido para consideração das demais métricas, ou seja, para uma previsão ser considerada, a sua *bounding box* precisa ter um valor de no mínimo uma porcentagem definida de IoU para ser considerada. Esses limiares, por convenção, são definidos como mAP@0.5 (50% de IoU) e mAP@0.95 (95% de IoU). Medida através da Equação 2.3. faz uma média entre as precisões para diferentes valores de revocação,

$$mAP = \frac{1}{N} \cdot \sum_{i=1}^{N} AP_i \tag{2.3}$$

2.9 Revisão de Literatura

Um dos objetivos da agricultura hoje é minimizar a perda de produtividade devido à infestação de pestes. Por meio dessa adversidade, o *Pest-Yolo Framework* [11] foi desenvolvido baseado em uma união de conceitos da YOLOv3, YOLOv4, FPN e agregação de caminhos para detectar pestes em um sistema tempo real, propondo uma boa troca entre acurácia e eficiência. Sua arquitetura baseia-se em 3 etapas:

Estágio de extração de atributos que utilizam uma CNN com a adição do conceito de *Squeeze and Excitation* (Apertamento e Excitação) para diminuir a perda de informações explícitas das convoluções (que normalmente fazem com que as informações fiquem implícitas).

Aprimoramento de atributos, utilizando a FPN e uma rede de agregação de caminhos, incluindo estágio cruzado de fusão múltipla de atributos para manter uma conexão cruzada entre níveis semânticamente mais fracos (que passaram por poucas convoluções), contendo mais informações relativas à localização dos objetos, à níveis semânticamente mais fortes.

Detecção alvo, baseado no uso da cabeça da YOLOv3 que utiliza três diferentes escalas para camadas de detecção, onde cada camada prevê a pontuação de objetividade da *bounding box* relativa à regressão logística.

Outra aplicação da detecção em tempo real de pestes na agricultura é realizada na agricultura de precisão de pomares de avelã por meio do *Yolo-Based Pest Detection System* [12], desenvolvido para detectar insetos e aprimorar o tratamento das plantações. Os insetos que são capturados por uma armadilha de cola servem de objeto de detecção para o sistema que diferencia os insetos de outros elementos como folhas. A arquitetura do sistema é baseada no uso da YOLOv4 que utiliza uma CNN com conexão parcialmente

cruzada e uma camada de agrupamento espacial piramidal como *backbone* para extração de atributos, a agregação de caminhos é o pescoço que coleta diferentes razões de mapas de atributos e a cabeça da YOLOv3 para fazer previsões (localização das *bounding boxes* e das probabilidades de classes).

O treinamento do sistema baseia-se na redução da perda por meio da função de perda da média quadrada ou por funções de perda baseadas na IoU. O aprimoramento proporcionado pelo *data augmentation* foi investigado durante o estudo e foi utilizado imagens RGB-D para adquirir informações de espessura que também ajudam na distinção entre pequenas folhas que têm o formato de um inseto pequeno mas possui diferente espessura.

Dessa forma, o ponto de partida desse trabalho foi utilizar a YOLOv4, assim como [11] e [12] para detectar insetos nas plantações, mas diferentemente dos trabalhos citados, o desafio foi levado à detecção diretamente sobre as folhas da plantação e verificando o desempenho ao comparar não só através das métricas estatísticas sobre o próprio modelo mas também entre as versões posteriores, YOLOR [26], YOLOv5 [27], YOLOv7 [28].

Capítulo 3

Materiais e Métodos

Os métodos utilizados na comparação entre os modelos YOLO e os materiais utilizados nos treinamentos serão descritos a seguir.

3.1 Base de Dados

Todas as imagens utilizadas nos treinamentos, validação e testes foram obtidas por meio da parceria com o Dr. Edson Hirose [1], Agrônomo da Embrapa, Pesquisador A da Embrapa-Soja na região Centro-Oeste do Brasil e trabalha no setor de Núcleos Temáticos. Para obtenção das imagens foi utilizado um celular que possui o modelo de câmera LG-H870, que capturou 77 imagens a 4160 × 3120 *pixels* de resolução e 3 videos de 1920 × 1080 *pixels* de resolução a uma taxa de 30 quadros por segundo. Para a anotação das coordenadas dos Percevejos sobre o treinamento supervisionado 2.2, foi utilizado o *software open source labelImg* [2] desenvolvido na linguagem de programação *Python*. Uma amostra da base de dados pode ser vista na Figura 3.1, e as anotações de suas coordenadas, utilizando o *labelImg*, na Figura 3.2.

3.2 Modelos Utilizados

A fim de comparação, conforme motivação anterior, foram escolhidos os Modelos YOLOv4 [3], YOLOR [4], YOLOv5 [5] e YOLOv7 [6].

[1] Link para acesso ao Currículo do Dr. Edson <http://lattes.cnpq.br/7892692886810778>. Acessado em 05/02/2023

[2] Link para acesso ao repositório do software <https://github.com/heartexlabs/labelImg>. Acessado em 05/02/2023

[3] Link do repositório <https://github.com/AlexeyAB/darknet>. Acessado em 05/02/2023

[4] Link do repositório <https://github.com/WongKinYiu/yolor>. Acessado em 05/02/2023

[5] Link do repositório <https://github.com/ultralytics/yolov5>. Acessado em 05/02/2023

[6] Link do repositório <https://github.com/WongKinYiu/yolov7>. Acessado em 05/02/2023

Figura 3.1: Amostra da Base de Dados contendo um Percevejo..

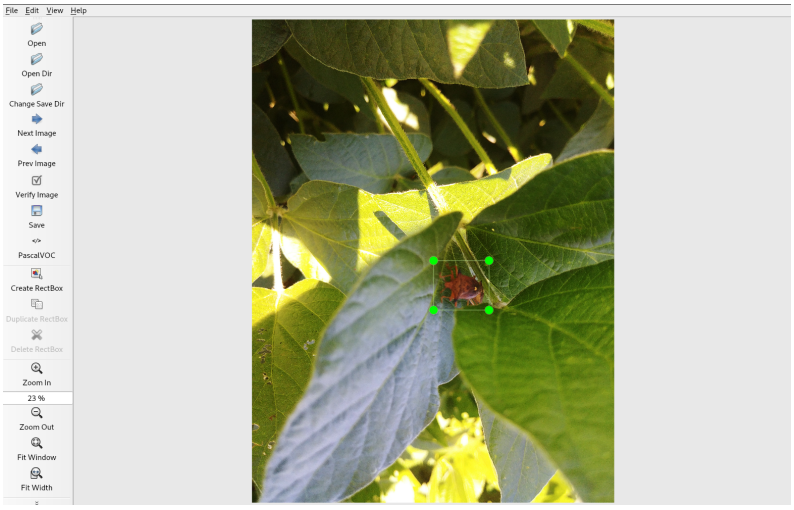

Figura 3.2: Anotação das coordenadas de localização do Percevejo..

3.3 Pré-Processamento

Para treinar os modelos, foi necessário separar a base de dados em imagens para treinamento, validação e teste. A divisão foi feita no formato 80% para treinamento, 10% para validação e 10% para testes. O código implementado pode ser visto na Figura 3.3. A biblioteca *splitfolders* possui a função *ratio* que busca a base de dados no diretório *dataset*, utiliza a *seed* para aleatoriamente separar as imagens nos sub-diretórios *train*, *val*, *test* (por padrão) copiadas para o diretório *obj*. Apenas para o YOLOv4, foi necessário criar um arquivo de extensão *txt* para obter o caminho de cada imagem. O código imple-

```python
import splitfolders


splitfolders.ratio("dataset", output="obj",
    seed=1337, ratio=(.8,.1,.1), group_prefix=None, move=False)
```

Figura 3.3: Código de separação da base de dados.

```python
import glob, os

current_dir = 'data'

file_train = open('train.txt', 'w')
file_val = open('val.txt', 'w')
file_test = open('test.txt', 'w')

yolov4_data_format = ['train', 'val', 'test']

for data in yolov4_data_format:
    for pathAndFilename in glob.iglob(os.path.join(current_dir + '/' + data
                                                   + '/images' ,"*")):
        title, ext = os.path.splitext(os.path.basename(pathAndFilename))
        if data == 'train':
            file_train.write("data/obj" + "/" + title + ext + "\n")
        elif data == 'val':
            file_val.write("data/obj" + "/" + title + ext + "\n")
        elif data == 'test':
            file_test.write("data/obj" + "/" + title + ext + "\n")
            break
```

Figura 3.4: Código para a obtenção do caminho das imagens.

mentado para essa exigência, pode ser visto na Figura 3.4. Também implementadas em *Python*, foram utilizadas as bibliotecas *glob* e *os*. O código consistiu em criar 3 arquivos *txt* para conter o caminho das imagens já separadas previamente em seus devidos diretórios. As funções *join* de escopo *path* que fazem parte de *os*, juntamente com *iglob* de *glob* listam todos os nomes dos arquivos do diretório especificado e a função *splitext* também de *path* separa o nome do arquivo e sua extensão, dessa forma, o *for loop* itera sobre todas as imagens e escreve respectivamente seus caminhos. Já nos demais modelos, os arquivos de configuração foram suficientes para buscar os caminhos das imagens de treinamento, validação e testes.

Por mais que *data augmentation* esteja presente no treinamento do YOLOv4, referenciado na Seção 2.7, é possível também utilizar essa técnica para aumentar a quantidade

de imagens no pré-processamento. Devido a quantidade pequena de imagens na base de dados, foi necessário utilizar técnicas de *data augmentation*, que consiste em fazer distorções geométricas e fotométricas nas imagens como rotações, espelhamento, mudanças de brilho, saturação, entre outras, para gerar imagens alteradas, dando a impressão para a rede neural que são novas imagens. Para isso, foi implementado o código que pode ser visto na Figura 3.5.

```
import os
from os import listdir
import tensorflow.keras.preprocessing.image as kpi

dataGenerator = kpi.ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.1,
        height_shift_range=0.1,
        shear_range=0.2,
        zoom_range=0.1,
        brightness_range =(0.3,0.8),
        horizontal_flip=True,
        vertical_flip=True,
        fill_mode='nearest')


def dataset_augmentation(filename):
    image = kpi.load_img(filename)
    image_array = kpi.img_to_array(image)
    image_array= image_array.reshape((1,) + image_array.shape)

    i = 0
    for batch in dataGenerator.flow(image_array, batch_size=1,
                                    save_to_dir='augmented_dataset',
                                    save_prefix='brown-bugs', save_format='jpg'):
        i+=1
        if i>6:
            break
    return

dir = './data/images'
for image in os.listdir(dir):
    if image.endswith(".jpg"):
        dataset_augmentation(dir + '/' + image)
```

Figura 3.5: Código para Data Augmentation.

As bibliotecas *os* e *tensorflow.keras* de *Python* foram utilizadas no desenvolvimento do método *dataset_augmentation*. O método *ImageDataGenerator* que está no módulo de pré-processamento de imagem da biblioteca *tensorflow.keras*, é responsável por de-

terminar quais serão os tipos de *data augmentation*. A fim de obter uma variedade de imagens considerando as situações em que os percevejos podem se encontrar em campo foram selecionados as alterações através de rotações, mudanças de largura e tamanho, cisalhamento, zoom, brilho, rotações horizontais e verticais e preenchimento por meio da cópia dos *pixels* mais próximos. Os métodos *load_img*, *img_to_array* e *reshape* serviram para carregar a imagem, transformar em vetor e remodelar os canais RGB da imagem para um *array*. Dessa forma, foi utilizado esse vetor que contem os *pixels* da imagem como parâmetro para a função *dataGenerator*. Por fim, cada imagem do diretório de imagens foi inserido com suas distorções no diretório *dataset-augmentation*. É possível observar uma amostra da imagem na Figura 3.6 que foi manipulada a partir da imagem real do percevejo.

Figura 3.6: Amostras da base de dados, em que a imagem da esquerda é a imagem original e a da esquerda foi manipulada pela rotação em sentido anti-horário, teve seu brilho alterado e preenchida por meio da cópia dos *pixels* dos vizinhos mais próximos.

3.4 Configuração

Para configurar o YOLOv4, foi necessário mudar parâmetros de sua arquitetura para customizar o treinamento para apenas 1 classe, visto que o modelo foi implementado e validado utilizando a base de dados COCO [29] que possui 80 classes. Foi selecionado o modelo de 416 $\times$ 416 *pixels* de dimensão, por ser mais leve e permitir treinamentos utilizando *batches* [7] maiores. Para isso foi necessário mudar os parâmetros relativos ao número de classes e filtros que são calculados através do número de classes no arquivo de extensão *cfg* usado, esses parâmetros podem ser vistos na Figura 3.7. Os demais parâmetros foram inalterados. Por requisitos do modelo houve a necessidade de criar 2 arquivos com extensões *data* e *names*. Contendo respectivamente o número de classes,

[7] Quantidades de amostras por época.

os caminhos para encontrar os arquivos *txt* que foram criados anteriormente, o diretório *backup* para guardar os pesos da rede durante o treinamento e o nome da classe do objeto a ser localizado. A rede utilizou pesos pré-treinados [8] na base de dados COCO.

Figura 3.7: Parâmetros do arquivo de configuração *cfg* do YOLOv4

A configuração da YOLOR, foi similar ao YOLOv4, porém, houve necessidade de alterar os parâmetros do arquivo de configuração, a configuração dos caminhos das imagens consistiu em especificar os diretórios que contem as imagens e suas respectivas anotações. Diferentemente do arquivo *names* do YOLOv4, o arquivo utilizado foi de extensão *yaml*, sua configuração pode ser vista na Figura 3.8. O modelo escolhido foi o de 448 × 448 pixels, por ser o menor e mais rápido modelo disponível, assim como na YOLOv4.

Para configurar o YOLOv5, assim como o YOLOR, foi necessário configurar o arquivo *yaml* com seus devidos caminhos para as imagens, porém a configuração dos parâmetros da arquitetura da rede, diferentemente dos anteriores, foi apenas preciso mudar apenas o número de classes. A extensão utilizada ao invés de *.cfg* como nas anteriores foi a *.yaml*. O modelo escolhido foi o de 640 × 640 pixels, por ter apresentado melhores resultados ao se comparar com o menor modelo de 448 × 448.

Por fim, a configuração do YOLOv7 foi idêntica ao YOLOv5 mas o modelo escolhido foi o de 448 × 448 pixels, por ser o menor e mais rápido dos modelos, assim como nas versões 4 e R.

[8]Link de *download* dos pesos pré-treinados <`https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137`>

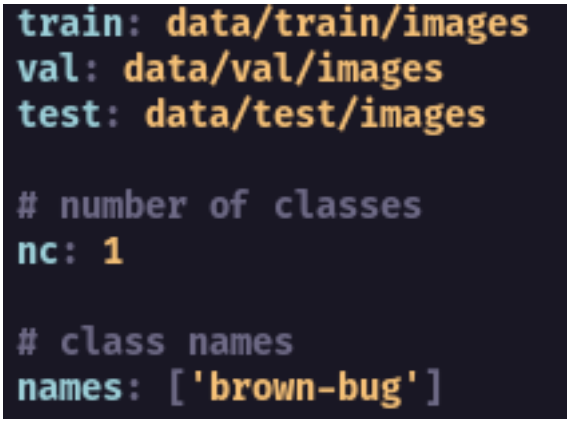

Figura 3.8: Configuração do arquivo *.yaml* do YOLOR.

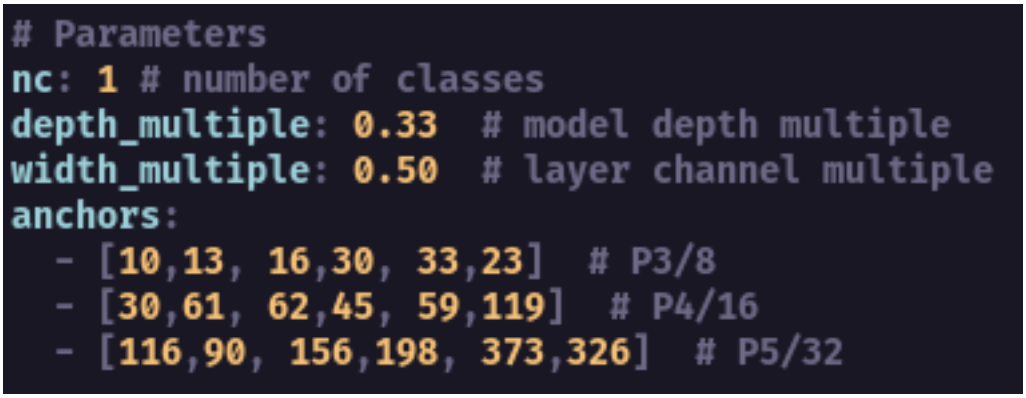

Figura 3.9: Configuração dos parâmetros do YOLOv5.

3.5 Equipamento Utilizado

Os treinamentos ocorreram em máquina local, as configurações do computador utilizado foram: Unidade Central de Processamento Intel I7 com 16 Giga Bytes de memória RAM e Unidade Gráfica de Processamento Gráfico NVIDIA GeForce RTX 3070 com 8 Giga Bytes de memória VRAM. A estratégia de treinamento de todos os modelos consistiu em utilizar pesos pré-treinados no COCO e a partir das configurações definidas anteriormente, obter a atualização dos pesos por meio das épocas de treinamento. No Capítulo 4, os resultados dos treinamentos serão detalhados.

3.6 Funções Customizadas

A partir do modelo escolhido, é possível aplicar funções customizadas. O *software* [9] *open source* possui algumas funções implementadas para utilizar os pesos pré treinados nos modelos, como por exemplo para fazer contagens de objetos, que foi utilizada pelo sistema.

Para isso, uma das etapas necessárias consiste em converter os pesos do modelo YOLOv4 para o formato *tensorflow* utilizado pelo *software*, possibilitando o uso das funções para inferências de imagens ou vídeos.

[9]Link do repositório <`https://github.com/theAIGuysCode/yolov4-custom-functions`>

Capítulo 4

Resultados e Discussão

Visto que a detecção de percevejos em campo é um desafio, devido a quantidade de elementos que podem conter em plantações, todos os materiais e métodos definidos no Capítulo 3, assim como as configurações e parâmetros de treinamento foram utilizados para obter o melhor resultado possível. A seguir serão apresentados com mais detalhes os resultados para cada modelo utilizado em cada treinamento, a comparação entre eles e por fim será feita uma discussão.

4.1 Treinamento

Para o YOLOv4, os treinamentos demoraram uma média de 8 horas e foram utilizadas cerca de 6 mil épocas. Para o YOLOR, demoraram cerca de 1 hora e foram utilizadas cerca de 150 épocas. Para o YOLOv5 os treinamentos demoraram cerca de 30 minutos e foram utilizadas cerca de 200 épocas. Por fim, no YOLOv7 os treinamentos demoraram cerca de 1 hora e meia e foram treinadas cerca de 400 épocas. As informações dos treinamentos podem ser vistas na Tabela 4.1.

Tabela 4.1: Configurações de Treinamento dos modelos YOLO.

Versão	Tempo de Treinamento (horas)	Quantidade de Épocas
YOLOv4	8	6000
YOLOR	1	150
YOLOv5	0.5	100
YOLOv7	1	250

No geral, os treinamentos sem o aumento na base de dados apresentaram uma instabilidade perceptível nos gráficos 4.1, 4.5, 4.9, 4.13. Com a evolução das tecnolgias para cada versão posterior, mais instável o treinamento foi, devido a serem modelos que apre-

sentam técnicas mais refinadas e com isso precisam de muitas imagens. Mas, aplicando-se a técnica de *data augmentation* desenvolvida e detalhada no Capítulo 3, os treinamentos apresentaram uma estabilidade maior e os resultados consequentemente foram melhores também, podendo ser vistos nos gráficos 4.2, 4.7, 4.10, 4.15. Em termos comparativos, os cálculos das métricas dos treinamentos utilizaram bases diferentes, no sem *data augmentation* foram utilizadas 60 imagens para treinamento e 8 imagens para validação, já no com *data augmentation* foram 480 imagens para treinamento e 60 para validação, dessa forma, os valores dos cálculos do segundo modelo mostram uma maior confiabilidade.

4.1.1 YOLOv4

Para a versão YOLOv4, por meio da Figura 4.1, nota-se que no cálculo do mAP, houve uma grande instabilidade, variando bruscamente ao longo das iterações. Diferentemente da Figura 4.2 que apresentou estabilidade com pequenas mudanças após as 1600 iterações. Com isso, a inferência das imagens em campo mostram essa disparidade. Na Figura 4.3, só foi possível localizar 1 percevejo do casal de percevejos com uma confiança de 71%. Já na Figura 4.4, o casal de percevejos foi localizado com confianças de 96% e 100%.

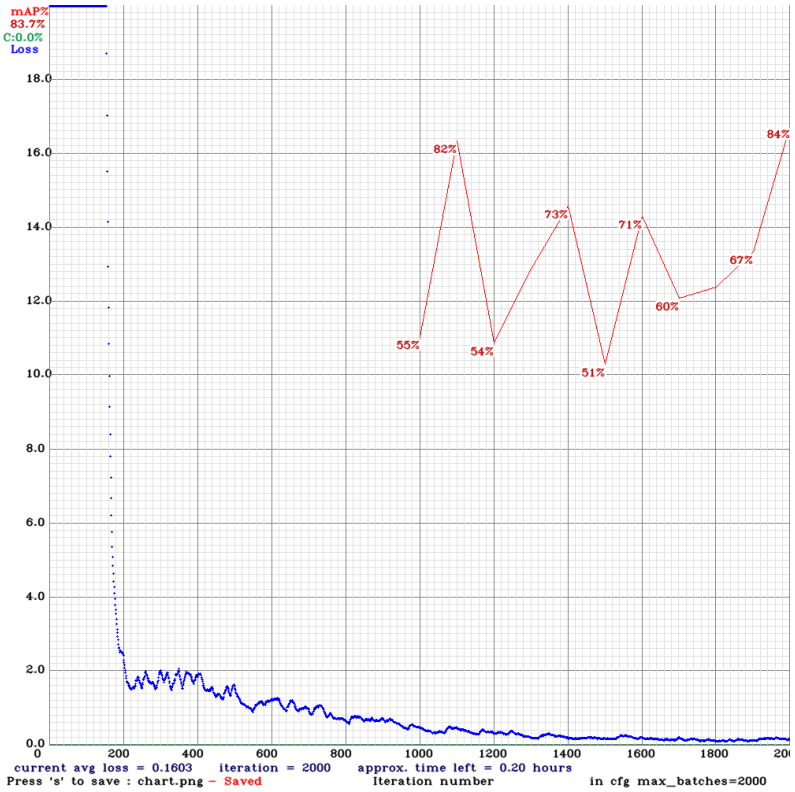

Figura 4.1: Resultados do treinamento sem *data augmentation*.

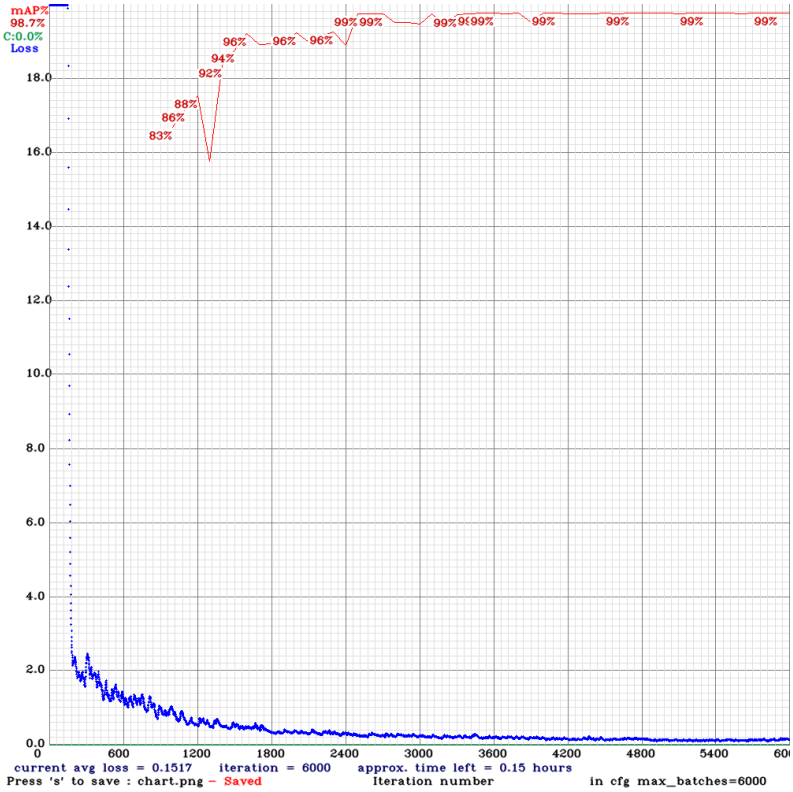

Figura 4.2: Resultados do treinamento com *data augmentation*.

Figura 4.3: Inferência que detectou um dos percevejos do casal, provinda dos pesos do treinamento sem *data augmentation*.

Figura 4.4: Inferência que detectou dois dos percevejos do casal, provinda dos pesos do treinamento com *data augmentation*.

4.1.2 YOLOR

No YOLOR os gráficos apresentam mais informações, os gráficos de *Box* e *val Box* são relativos ao erro por regressão da caixa de delimitação em treinamento e em validação respectivamente. Já a *Objectness* e *val Objectness* são relativos a confiança da presença do objeto em treinamento e validação respectivamente. Os gráficos de *Classification* são relativos a classificação entre as classes, como só possui 1 classe, não há informações para esse gráfico. Os demais são as métricas Precisão, mAP@0.5, Revocação e mAP@0.95, já relatados anteriormente no Capítulo 2.

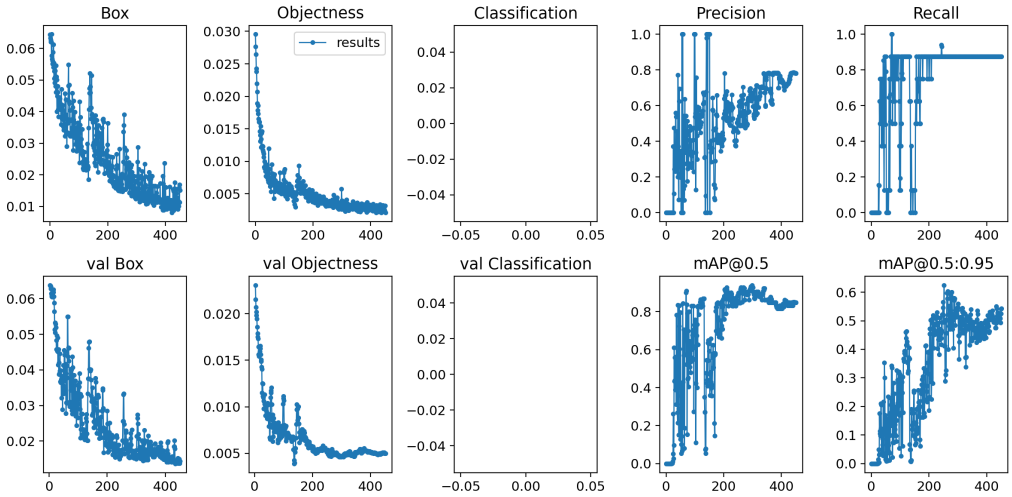

Figura 4.5: Resultados do treinamento sem *data Augmentation*.

Assim como no YOLOv4, houve instabilidade no treinamento sem *data augmentation* que é possível ser visto através da intermitência nos cálculos das perdas e das métricas, Figura 4.5. Já no treinamento com *data Augmentation*, há uma maior estabilidade, Figura 4.7. Porém, é possível observar nas Figuras 4.6 e 4.8 que a inferência utilizando os pesos obtidos pelos respectivos treinamentos foi similar, com pequena vantagem para o segundo modelo.

Figura 4.6: Inferência que detectou dois dos percevejos do casal, provinda dos pesos do treinamento sem *data augmentation*.

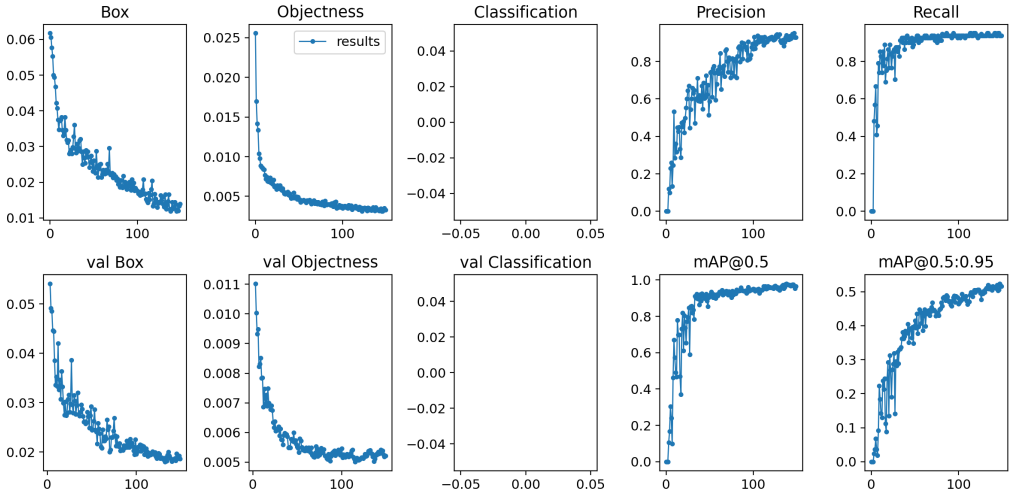

Figura 4.7: Resultados do treinamento com *data Augmentation*.

Figura 4.8: Inferência que detectou dois dos percevejos do casal, provinda dos pesos do treinamento com *data augmentation*.

4.1.3 YOLOv5

Com a versão YOLOv5, a instabilidade do treinamento sem *data augmentation* foi similar ao modelo anterior, Figura 4.9, assim como a maior estabilidade do treinamento com *data augmentation*, Figura 4.10, os *outliers* (valores atípicos) nos cálculos das métricas mAP@0.50, revocação e mAP@0.95 foram relativos ao estouro do tempo de limite máximo pré definido para essas treinamento dessas iterações, resultando em valores baixos nos respectivos cálculos, mas logo após, os valores seguiram a estabilidade. As inferências que podem ser vistas nas Figuras 4.11 e 4.12 mostram um grau de confiança maior no segundo modelo.

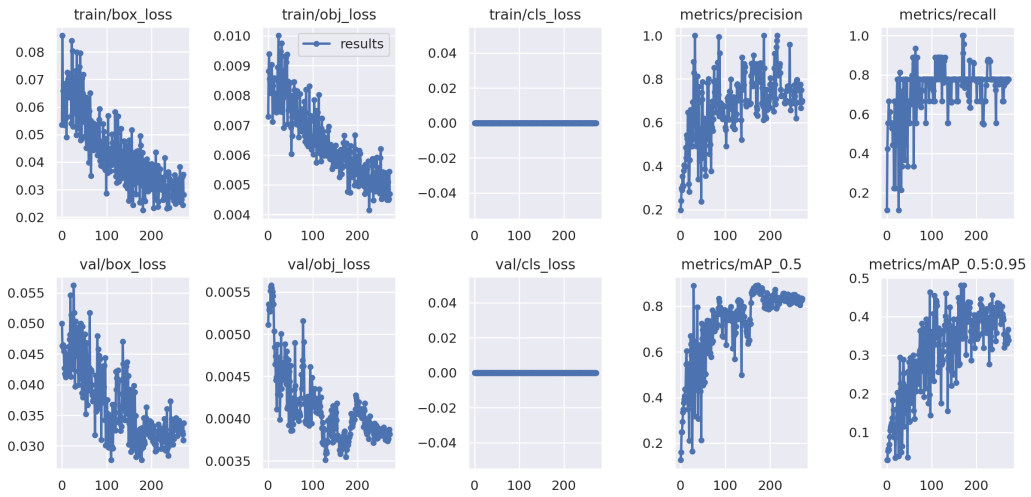

Figura 4.9: Resultados do treinamento sem *data augmentation*.

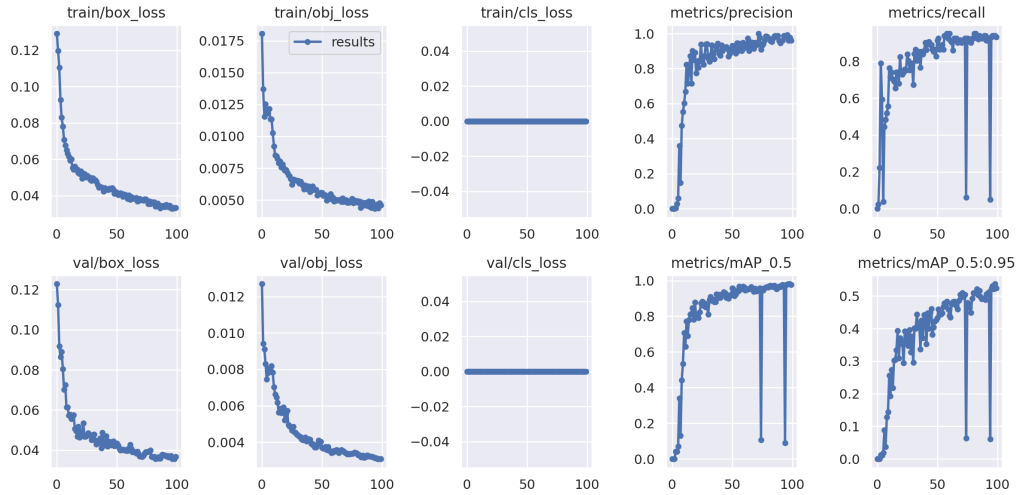

Figura 4.10: Resultados do treinamento com *data Augmentation*.

Figura 4.11: Inferência que detectou dois dos percevejos do casal, provinda dos pesos do treinamento sem *data augmentation*.

Figura 4.12: Inferência que detectou dois dos percevejos do casal, provinda dos pesos do treinamento com *data augmentation*.

4.1.4 YOLOv7

Por fim, o treinamento da versão YOLOv7 foi a que apresentou maior instabilidade no treinamento com *data augmentation*, a intermitência nos cálculos é possível de ser vista na Figura 4.13 e da mesma forma que os anteriores, o treinamento com *data augmentation* seguiu uma maior estabilidade, Figura 4.15. As inferências mostram essa disparidade entre os treinamentos, Figuras 4.14 e 4.16.

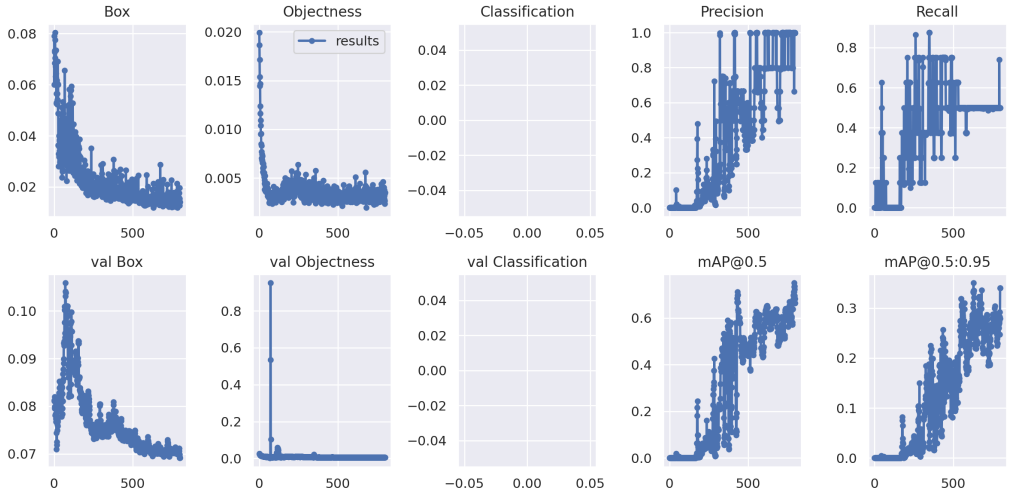

Figura 4.13: Resultados do treinamento sem *data augmentation*.

Figura 4.14: Inferência que detectou um dos percevejos do casal, provinda dos pesos do treinamento sem *data augmentation*.

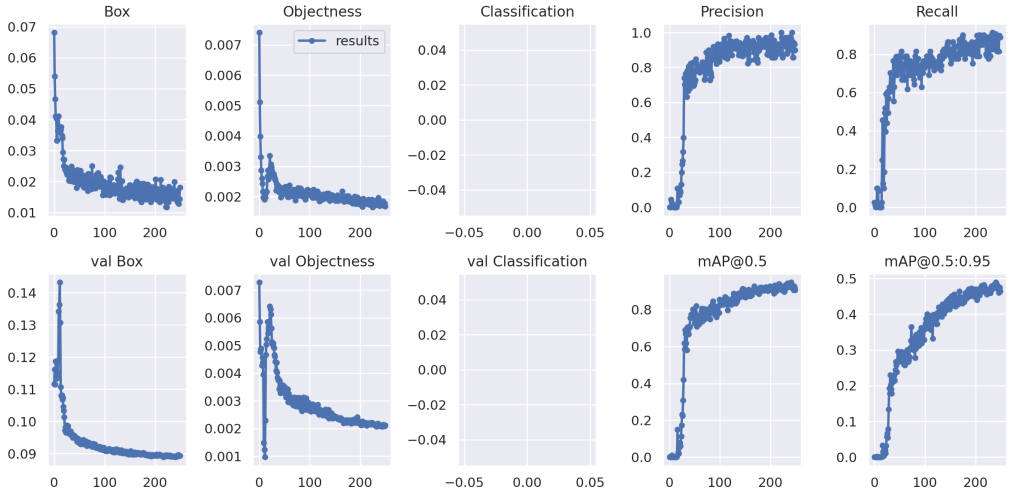

Figura 4.15: Resultados do treinamento com *data augmentation*.

Figura 4.16: Inferência que detectou dois dos percevejos do casal, provinda dos pesos do treinamento com *data augmentation*.

A fim de organizar todos os resultados, as Tabelas 4.2 e 4.3 contem os resultados respectivos do treinamento com *data augmentation* e sem *data augmentation*.

Tabela 4.2: Resultados treinamento sem *data augmentation*.

Versão	mAP@0.5	Precisão	Revocação
YOLOv4	90%	75%	60%
YOLOR	84%	75%	85%
YOLOv5	82%	78%	80%
YOLOv7	59%	65%	49%

Tabela 4.3: Resultados treinamento com *data augmentation*.

Versão	mAP@0.5	Precisão	Revocação
YOLOv4	99%	99%	99%
YOLOR	96%	93%	94%
YOLOv5	99%	95%	99%
YOLOv7	94%	91%	91%

4.2 Aplicação do Modelo

A fim de transferir os pesos obtidos pelo modelo que apresentou melhores resultados, foi utilizado o software descrito na Seção 3.6. Aplicando-se a função de contagem de objetos, os resultados obtidos podem ser vistos nas Figuras 4.17, 4.18. As imagens referenciadas

tratam-se de recortes de tela de um vídeo que foi utilizado na plantação real, nota-se que quando a folha cobre o percevejo na Figura 4.18, o modelo perde a informação e o contador de percevejos diminui de forma esperada.

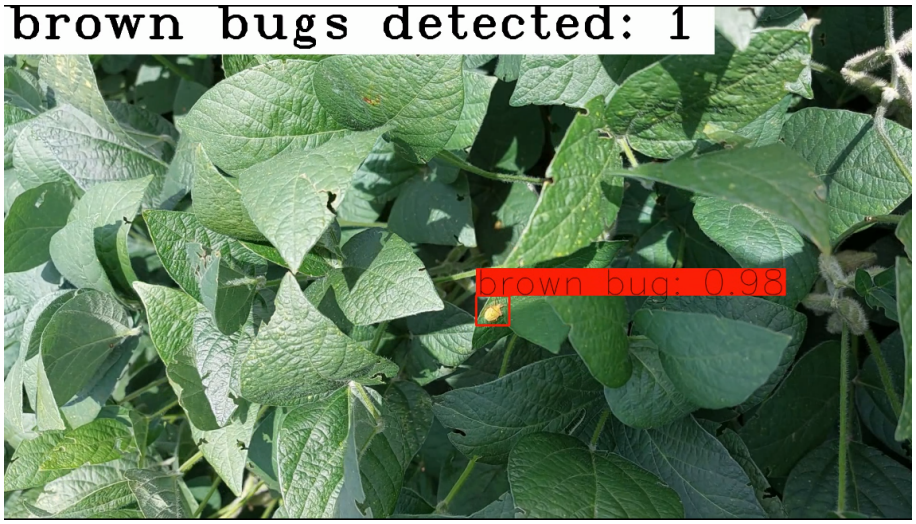

Figura 4.17: Inferência que detecta um Percevejo na plantação.

Figura 4.18: Inferência que detecta dois Percevejos na plantação.

Porém, foi observado também uma debilidade no modelo para detectar casais de percevejos. Visto que as imagens fornecidas em grande maioria foi relativa a percevejos solitários. Nas Figuras 4.19 e 4.20 é possível observar que mesmo existindo dois percevejos na imagem, no primeiro caso que a imagem está mais próxima, o modelo de forma eficaz detecta ambos os percevejos, mas na segunda imagem, só consegue detectar um deles.

Figura 4.19: Inferência que detecta um casal de percevejos na plantação.

Figura 4.20: Inferência que detecta apenas 1 percevejo do casal de percevejos na plantação.

4.3 Discussão

Muitos são os modelos que foram desenvolvidos para a tarefa de detecção de objetos, os critérios escolhidos nesse trabalho consistiram na detecção em tempo real e disponibilidade dos recursos computacionais, pois, todos os modelos foram treinados em máquina local, dessa forma, apenas uma GPU foi utilizada, o esforço dos pesquisadores foi o desenvolvimento de um método robusto que precisasse de apenas uma GPU. Com isso, foi selecionado como principal modelo desse trabalho o YOLOv4, as versões posteriores foram apenas em critério de avaliação prática, visto que suas arquiteturas não foram descritas na Seção 2. Os resultados do treinamento sem *data augmentation*, Tabela 4.2, mostraram

a dependência de uma grande quantidade de imagens para treinar os modelos. Devido a dificuldade de obter um número grande e diverso de imagens reais em pouco tempo, com diferentes iluminações, ângulos, quantidade de percevejos por imagem e situações adversas, foi necessário o uso das técnicas de *data augmentation*, como relatado anteriormente. Dessa forma, os resultados do treinamento com *data augmentation*, Tabela 4.3, foram consideravelmente superiores. Por meio das observações, verificou-se que a detecção de percevejos no campo utilizando esses modelos é eficaz, pois todos eles produzem inferências em tempo real e com uma acurácia relativamente alta. Apesar de mais novos, as versões posteriores ao YOLOv4 mostraram piores resultados, considerando esse banco de imagens e as condições do ambiente que os objetos precisam ser detectados, porém, vale a menção de que a grande desvantagem observada pelo modelo YOLOv4 é com relação ao tempo de treinamento, Tabela 4.1, mostrando extremamente mais lento do que os demais.

Capítulo 5

Conclusão

Diante da atual situação da agricultura brasileira, possuindo mais de 100 milhões de hectares de terras completamente improdutivas [7] por meio de más práticas de plantações e uso de agrotóxicos, técnicas de agricultura de precisão são necessárias para melhor aproveitamento dos recursos e aumento da qualidade da safra. Dessa forma, através desse trabalho, foi possível avaliar alguns entre os mais atuais modelos para Deteção de Objetos em tempo real. Permitindo assim a detecção dos percevejos em fotos ou vídeos para auxiliar o combate à peste que mais assola as plantações de soja no Brasil o Percevejo Marrom.

Devido à indisponibilidade de bases de dados pela internet com imagens desse inseto, foram utilizadas imagens captadas diretamente em uma plantação de soja localizada na Embrapa do Distrito Federal por meio de uma parceria com o Dr. Agrônomo Edson Hirose. Diante da pouca diversidade de imagens na base de dados disponibilizada, foram utilizadas técnicas de processamento de imagem para aumentar a quantidade possível de situações em que são encontradas percevejos. A técnica *Data Augmentation*, mesmo sendo utilizada de forma automática em todos os modelos para treinamento e detecção, não foi suficiente para trazer bons resultados com as 77 imagens utilizadas inicialmente. Portanto, foi necessário implementar um algoritmo para aplicar essas técnicas e multiplicar a quantidade original de imagens, resultando em 600 imagens. Dessa forma, mesmo diante dessas adversidades, foi possível obter resultados satisfatórios.

Em suma, através dos treinamentos, foi possível verificar que o modelo que obteve melhores resultados, levando em conta o número de imagens utilizadas, foi o YOLOv4. Com isso, os pesos do modelo foram convertidos e utilizados para contar o números de ocorrências de percevejos em imagens e vídeos. O que se mostrou eficaz para essa primeira aplicação, permitindo um mapeamento de foco de percevejos na plantação através de uma quantificação desses insetos em uma área determinada de varredura.

5.1 Trabalhos Futuros

A fim de incrementar os resultados obtidos por esse trabalho, algumas das ideias consistem:

- Aumentar a base de dados, para que a comparação entre os modelos seja mais equalitária e avaliar a necessidade de escolha do modelo a ser aplicado;
- Conversão desse modelo para uso em sistemas embarcados, podendo ser acoplado a drones ou equipamentos que se movimentam de forma controlada ou automática.

References

[1] Education, IBM Cloud: *Redes neurais*. https://www.ibm.com/br-pt/cloud/learn/neural-networks. Last accessed 17 August 2020. ix, 3, 4

[2] Girshick, Ross B., Jeff Donahue, Trevor Darrell e Jitendra Malik: *Rich feature hierarchies for accurate object detection and semantic segmentation*. CoRR, abs/1311.2524, 2013. http://arxiv.org/abs/1311.2524. ix, 5, 6, 7

[3] Ren, Shaoqing, Kaiming He, Ross B. Girshick e Jian Sun: *Faster R-CNN: towards real-time object detection with region proposal networks*. CoRR, abs/1506.01497, 2015. http://arxiv.org/abs/1506.01497. ix, 7, 8

[4] Lin, Tsung-Yi, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan e Serge J. Belongie: *Feature pyramid networks for object detection*. CoRR, abs/1612.03144, 2016. http://arxiv.org/abs/1612.03144. ix, 8, 9

[5] Redmon, Joseph, Santosh Kumar Divvala, Ross B. Girshick e Ali Farhadi: *You only look once: Unified, real-time object detection*. CoRR, abs/1506.02640, 2015. http://arxiv.org/abs/1506.02640. x, 9, 11

[6] Bochkovskiy, Alexey, Chien-Yao Wang e Hong-Yuan Mark Liao: *Yolov4: Optimal speed and accuracy of object detection*. CoRR, abs/2004.10934, 2020. https://arxiv.org/abs/2004.10934. x, 1, 12

[7] Embrapa: *Trajetória da agricultura brasileira*, 2018. https://www.embrapa.br/visao/trajetoria-da-agricultura-brasileira, Last accessed 2018. 1, 38

[8] Agriculture, United States Department of: *World soybean production 2022/2023*, 2022. Last accessed December, 2022. 1

[9] CNN: *Estimativa do crescimento populacional*, 2022. https://www.cnnbrasil.com.br/saude/populacao-mundial-se-aproxima-de-8-bilhoes-numero-deve-ser-atingido-na-t Last accessed November, 2022. 1

[10] Bongiovanni, Rodolfo e James Lowenberg-DeBoer: *Precision agriculture and sustainability*. Precision Agriculture, 5:359–387, agosto 2004. 1

[11] Tang, Zhe, Zhengyun Chen, Fang Qi, Lingyan Zhang e Shuhong Chen: *Pest-yolo: Deep image mining and multi-feature fusion for real-time agriculture pest detection*. Em *2021 IEEE International Conference on Data Mining (ICDM)*, páginas 1348–1353, 2021. 1, 15, 16

[12] Lippi, Martina, Niccolò Bonucci, Renzo Fabrizio Carpio, Mario Contarini, Stefano Speranza e Andrea Gasparri: *A yolo-based pest detection system for precision agriculture*. Em *2021 29th Mediterranean Conference on Control and Automation (MED)*, páginas 342–347, 2021. 1, 15, 16

[13] Academy, Data Science: *Deep Learning Book*. 2022. `https://www.deeplearningbook.com.br`. 3

[14] Mitchell, Tom M.: *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997, ISBN 978-0-07-042807-2. `https://www.worldcat.org/oclc/61321007`. 4

[15] O'Shea, Keiron e Ryan Nash: *An introduction to convolutional neural networks*. CoRR, abs/1511.08458, 2015. `http://arxiv.org/abs/1511.08458`. 5

[16] Krizhevsky, Alex, Ilya Sutskever e Geoffrey E Hinton: *Imagenet classification with deep convolutional neural networks*. Em Pereira, F., C.J. Burges, L. Bottou e K.Q. Weinberger (editores): *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`. 5

[17] Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg e Li Fei-Fei: *Imagenet large scale visual recognition challenge*. CoRR, abs/1409.0575, 2014. `http://arxiv.org/abs/1409.0575`. 5

[18] Girshick, Ross B., Jeff Donahue, Trevor Darrell e Jitendra Malik: *Rich feature hierarchies for accurate object detection and semantic segmentation*. CoRR, abs/1311.2524, 2013. `http://arxiv.org/abs/1311.2524`. 6

[19] Uijlings, Jasper, K. Sande, T. Gevers e A.W.M. Smeulders: *Selective search for object recognition*. International Journal of Computer Vision, 104:154–171, setembro 2013. 6

[20] Girshick, Ross B.: *Fast R-CNN*. CoRR, abs/1504.08083, 2015. `http://arxiv.org/abs/1504.08083`. 7

[21] Redmon, Joseph e Ali Farhadi: *YOLO9000: better, faster, stronger*. CoRR, abs/1612.08242, 2016. `http://arxiv.org/abs/1612.08242`. 12

[22] Redmon, Joseph e Ali Farhadi: *Yolov3: An incremental improvement*. arXiv, 2018. 12

[23] Simonyan, Karen e Andrew Zisserman: *Very deep convolutional networks for large-scale image recognition*. Em *International Conference on Learning Representations*, 2015. 12

[24] Redmon, Joseph: *Darknet: Open source neural networks in c*. `http://pjreddie.com/darknet/`, 2013–2016. 12

[25] Perez, Luis e Jason Wang: *The effectiveness of data augmentation in image classification using deep learning*. CoRR, abs/1712.04621, 2017. http://arxiv.org/abs/1712.04621. 13

[26] Wang, Chien-Yao, I-Hau Yeh e Hong-Yuan Mark Liao: *You only learn one representation: Unified network for multiple tasks*. CoRR, abs/2105.04206, 2021. https://arxiv.org/abs/2105.04206. 16

[27] seekFire: *Yolov5 architecture*. https://github.com/ultralytics/yolov5/issues/280. Last accessed 3 June 2020. 16

[28] Wang, Chien Yao, Alexey Bochkovskiy e Hong Yuan Mark Liao: *Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*, 2022. https://arxiv.org/abs/2207.02696. 16

[29] Lin, Tsung-Yi, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár e C. Lawrence Zitnick: *Microsoft COCO: common objects in context*. CoRR, abs/1405.0312, 2014. http://arxiv.org/abs/1405.0312. 21