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

# Identificação de Moscas da Espécie *Bemisia tabaci* em Armadilhas Adesivas Amarelas Usando Aprendizado de Máquina

Vasco Rodrigues Monteiro Neto

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Orientador
Prof. Dr. Marcus Vinicius Lamar

Brasília
2021

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Computação — Licenciatura

Coordenador: Prof. Dr. Wilson Henrique Veneziano

Banca examinadora composta por:

Prof. Dr. Marcus Vinicius Lamar (Orientador) — CIC/UnB
Prof.ª Dr.ª Roberta Barbosa Oliveira — CIC/UnB
Prof. Dr. Vinícius Ruela Pereira Borges — CIC/UnB

**CIP — Catalogação Internacional na Publicação**

Monteiro Neto, Vasco Rodrigues.

Identificação de Moscas da Espécie *Bemisia tabaci* em Armadilhas Adesivas Amarelas Usando Aprendizado de Máquina / Vasco Rodrigues Monteiro Neto. Brasília : UnB, 2021.
76 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2021.

1. mosca branca, 2. aprendizado de máquina, 3. redes neurais, 4. visão computacional

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

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

# Identificação de Moscas da Espécie *Bemisia tabaci* em Armadilhas Adesivas Amarelas Usando Aprendizado de Máquina

Vasco Rodrigues Monteiro Neto

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Prof. Dr. Marcus Vinicius Lamar (Orientador)
CIC/UnB

Prof.ª Dr.ª Roberta Barbosa Oliveira
CIC/UnB

Prof. Dr. Vinícius Ruela Pereira Borges
CIC/UnB

Prof. Dr. Wilson Henrique Veneziano
Coordenador do Curso de Computação — Licenciatura

Brasília, 25 de Maio de 2021

# Dedicatória

Dedico este trabalho ao meus pais, Maria Aparecida e Sebastião, que nunca me deixaram faltar nada e sem os quais eu não estaria aqui hoje. Também dedico à minha companheira, Danyelle, minha fonte de suporte e inspiração.

# Agradecimentos

Agradeço aos colegas e professores que cruzaram o meu caminho ao longo dessa caminhada, em especial ao Prof. Marcus Vinicius Lamar pela paciência, boa vontade e sapiência para me orientar ao longo desse projeto. Também não poderia deixar de agradecer ao Laboratório de Ecologia de Insetos do Instituto de Ciências Biológicas da UnB pelo suporte.

# Resumo

A *Bemisia tabaci*, popularmente conhecida como mosca branca, é considerada hoje a maior praga agrícola do mundo, tanto pelo seu potencial em servir como vetor de diversas fitopatologias quanto por sua distribuição cosmopolita. O objetivo do presente trabalho é usar a visão computacional para propor e comparar o desempenho de classificadores, utilizando aprendizado de máquina, no processo de identificação e contagem da mosca branca em imagens de armadilhas adesivas amarelas tradicionais. O objetivo é otimizar o tempo levado em tal processo, que quando feito de forma manual ainda que por um especialista, pode vir a demandar muito tempo. Um modelo de baixo custo computacional como o *K-Nearest Neighbors*, apresentou uma acurácia superior a 94% na classificação dos insetos, enquanto no melhor resultado, utilizando um modelo de Rede Neural Convolucional, mais complexo, a acurácia foi de aproximadamente 95,55%. Dessa forma, alguns dos modelos aqui propostos se mostraram eficientes para solucionar o problema da contagem de moscas brancas em armadilhas adesivas, podendo ser incrementado futuramente usando classificadores mais complexos.

**Palavras-chave:** mosca branca,aprendizado de máquina, redes neurais, visão computacional

# Abstract

Bemisia tabaci, popularly known as the whitefly, is today considered the biggest agricultural pest in the world, both for its potential to serve as a vector for various phytopathologies and also for its cosmopolitan distribution. The objective of the present work is to use computer vision techniques to propose and compare the performance of classifying models, using machine learning and neural networks, in the process of counting and recognize whiteflies in images of traditional yellow sticky traps, in order to optimize the time taken in such a process, that when done manually even by a specialist, it can be very time consuming. A low cost computational model such as KNN showed an accuracy greater than 94% in the classification of these insects, while in the best result, using a more complex CNN model, the accuracy was approximately 95.55%. Some of our proposed models have shown to be efficient in solving the problem of counting whiteflies in sticky traps, however can be increased in the future using more complex classifier models.

**Keywords:** whitefly, machine learning, neural networks, computer vision

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Abreviaturas e Siglas

**ADAM** (do Inglês, *Adaptive Moment Estimation*).

**CNN** Redes Neurais Convolucionais (do Inglês, *Convolutional Neural Networks*).

**KNN** K Vizinhos Mais Próximos (do Inglês, *K-Nearest Neighbors*).

**ML** Aprendizado de Máquina (do Inglês, *Machine Learning*).

**MLP** Perceptron Multicamadas (do Inglês, *Multilayer Perceptron*).

**ReLU** Unidade Linear Retificada (do Inglês, *Rectified Linear Units*).

**RNA** Redes Neurais Artificiais.

**YOLO** You Only Look Once.

# Capítulo 1

# Introdução

A mosca-branca *Bemisia tabaci*, mostrado na Figura 1.1, é um pequeno inseto da ordem Hemiptera considerado uma praga agrícola em escala global [19]. São insetos sugadores de seiva, que podem se alimentar de mais de 600 especies de plantas [20], além de atuarem como vetores de diversos tipos de fito-vírus. Esses fatores renderam à espécie o título de "praga do século" [21], causando, apenas no Brasil, um prejuízo de mais de 700 milhões de dólares ao ano [22].

Figura 1.1: *Bemisia tabaci* [1]

Um meio tradicionalmente usado para realizar a detecção da presença desse inseto em uma localidade é o uso de armadilhas adesivas amarelas, ilustradas na Figura 1.2. Essa ferramenta é utilizada dentro e fora do ambiente acadêmico para auxiliar nos processos

de, além da detecção, identificação e monitoramentos dessa praga. De fácil manuseio, a armadilha é colocada no espaço da área de cultivo próximo as plantas, geralmente presa a um suporte, e deixada no local por um tempo determinado pelo responsável. Ao longo desse período os insetos são atraídos pela coloração amarela e acabam ficando grudados na armadilha, que pode ser usada em estufas, plantações e áreas urbanas [23].

Figura 1.2: Armadilhas adesivas amarelas [2]

Dado a crescente melhoria no poder computacional dos processadores e dispositivos de *hardware* em geral, técnicas de processamento de imagens e de aprendizagem de máquina evoluem em concomitância, tanto devido a viabilização do manejo de grandes quantidades de dados [24], tanto pela natureza multidisciplinar de sua aplicabilidade propriamente

dita, sendo empregue em áreas como a medicina, agricultura, topografia, segurança e monitoramento, carros autodirigidos, detecção de fraudes e diversos outros campos. Dentre as "áreas quentes" da Ciência da Computação voltada para a aprendizagem de máquina/inteligência artificial, está o domínio da visão computacional, área que, dentre outras coisas, busca caracterizar, interpretar e gerar alguma informação por meio da análise de imagens e vídeos [25].

O presente trabalho busca aplicar técnicas de visão computacional e aprendizado de máquina no processamento e análise de imagens de armadilhas adesivas amarelas.

## 1.1 Motivação

A armadilha adesiva amarela, como ferramenta bastante difundida na entomologia em geral, apenas captura os insetos. A identificação e contagem dos mesmos dependem de algum indivíduo apto para a tarefa, incumbência que pode demandar, em uma armadilha de $25 \times 20$ cm com mais de 100 moscas, em média 30 minutos, segundo estudantes de pós-graduação do Laboratório de Ecologia de Insetos da Universidade de Brasília. Dado esse problema, a proposta desse trabalho é criar uma ferramenta usando visão computacional, que consiga, de forma eficiente e eficaz, fazer a identificação para fins de contagem do total de moscas-brancas em uma foto/imagem de armadilha adesiva amarela de forma automatizada.

## 1.2 Objetivos

O objetivo principal deste trabalho é criar uma metodologia, e desenvolvê-la em *software* utilizando a ferramenta MATLAB, que identifique os insetos da espécie *Bemisia tabaci* em imagens de armadilhas adesivas amarelas, empregando para isso métodos de processamento de imagens e modelos de aprendizado de máquina treinados através de um banco de dados de armadilhas em que os insetos foram previamente identificados.

Portanto, afim de alcançar este propósito, também devemos realizar algumas etapas precedentes à detecção propriamente dita, e são elas: tratamento, segmentação e estruturação dos dados que serão usados para o treinamento das redes neurais artificiais.

## 1.3 Estrutura do documento

Este trabalho possui 5 capítulos:

- O primeiro sendo essa introdução, onde o problema, a motivação e os objetivos do trabalho são apresentados.

- O segundo capítulo é composto pela fundamentação teórica deste trabalho, apresentando os conceitos usados como base para o desenvolvimento do mesmo.
- No terceiro relata-se como os dados são tratados antes de estarem prontos para uso nos modelos de classificação e como estes foram construídos.
- O quarto capítulo mostra os resultados obtidos pelos modelos propostos.
- O quinto e último abarca uma síntese do trabalho e uma análise dos resultados obtidos através da metodologia proposta.

# Capítulo 2

# Fundamentação Teórica

Este capítulo trata dos conceitos teóricos que fundamentam esse trabalho, além de referências a trabalhos correlatos. Iniciaremos com os conceitos de Aprendizado de Máquina, Redes Neurais e algumas de suas variações, além de explicar o que caracteriza o treinamento desses tipos de modelos. Por fim, apresentamos trabalhos relacionados com este, desde alguns que utilizam a mesma tecnologia em contextos diferentes até modelos mais especialistas que visam a classificação da mesma espécie de inseto.

## 2.1 Aprendizado de Máquina

Na programação tradicional, o programador escreve um algoritmo que determina o funcionamento de determinado programa no qual o mesmo só vai gerar algum tipo de resposta de acordo com as regras pré-delimitadas pelo código. No campo do Aprendizado de Máquina (do Inglês, *Machine Learning*) (ML) essa organização é rearranjada, em que fica a cargo do programador fornecer os dados e as respostas esperadas à partir deles, e ao computador a incumbência de criar as regras[26]. Ou seja, à partir e um conjunto significativo de dados, a máquina deve detectar e reconhecer os padrões por detrás dos dados, e assim criar modelos que possuem aplicabilidade em outros dados. Esse processo é chamado de treinamento e é a maneira com que o programa aprende o valor de saída, de forma que seja capaz de generalizar e fazer predições com eficácia, que é mensurada pelo erro (diferença) entre as saídas do sistema e a resposta desejada. De acordo com a maneira que o algoritmo realiza esse treinamento, eles podem ser categorizados como:

- Aprendizagem supervisionada: O algoritmo de treinamento utiliza dados pré classificados, em que para cada elemento (ou elementos) de entrada existe um valor já conhecido de saída. Normalmente, os dados já rotulados são sub-divididos em um conjunto de treino e um conjunto, geralmente menor, de teste, que será usado para

verificar o grau de precisão do modelo, ou seja, o quão bem consegue predizer a saída esperada.

- Aprendizagem não-supervisionada: O algoritmo de treinamento não utiliza dados previamente rotulados e o modelo deve inferir quais são as informações e estruturas escondidas nos mesmos.
- Aprendizagem por reforço: é um modelo baseado em recompensar a melhor saída, em que o *feedback* do modelo determinará se o mesmo será recompensado. Se difere do aprendizado supervisionado pois o sistema não aprende diretamente com os dados rotulados, mas por um processo de tentativa e erro que favorece o melhor resultado.

## 2.2 Redes neurais artificiais

Considerada por muitos autores um subgrupo do ML [27], as Redes Neurais Artificiais (RNA's) são modelos computacionais que emulam a morfologia das células nervosas animais, no qual os dendritos de um neurônio recebem impulsos elétricos e transmitem para a célula contígua passando pelo corpo do axônio e resultando em uma sinapse, como mostrado na Figura 2.1. Nas RNA's o que forma a estrutura são neurônios artificiais, agrupados em camadas que se comunicam com as camadas adjacentes.

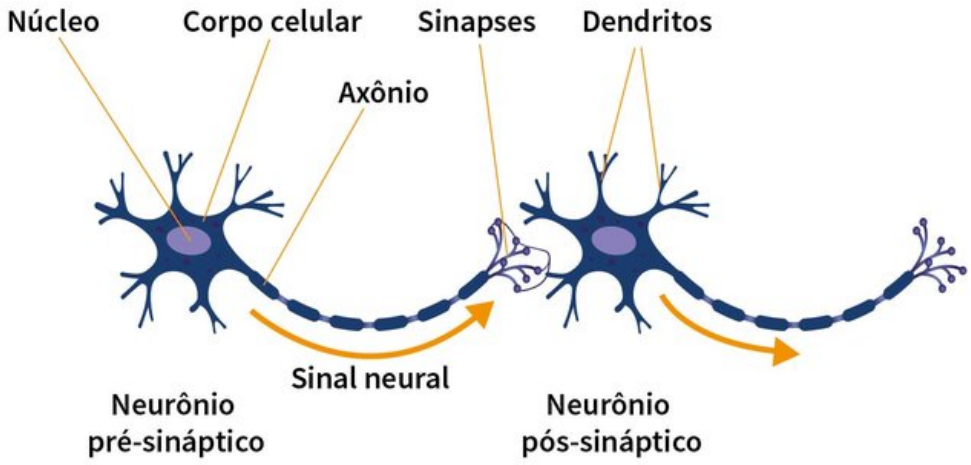



Figura 2.1: Fluxo de um impulso nervoso [3].

Um modelo tradicional de RNA que será usado nas redes implementadas nesse trabalho é o *feedfoward*. Este modelo é estruturado de forma que os dados fluem em uma única direção da entrada até a saída, sendo que o *output* dos neurônios de uma determinada camada são usados como *input* da camada subsequente.

A topologia básica de uma rede *feedforward* consiste em uma camada de entrada, uma ou mais camadas intermediárias, também chamadas de camadas escondidas, e uma camada de saída, como é mostrado na Figura 2.2.

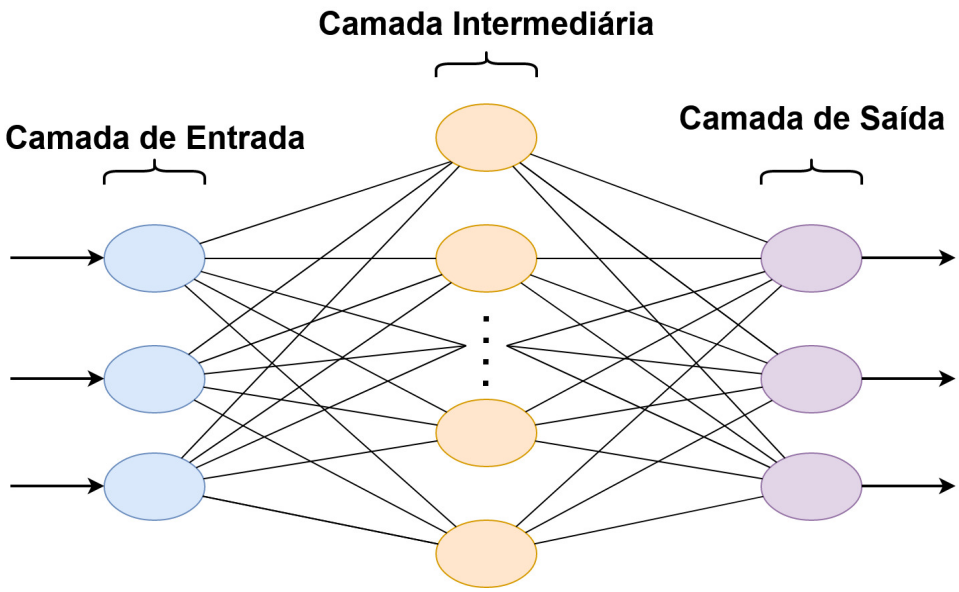



Figura 2.2: Exemplo de uma rede neural *feedforward*.

Embora as RNA's venham ganhando muito espaço devido a sua aplicabilidade voltada para inúmeros campos do conhecimento humano como engenharia, medicina, ecologia, agricultura, mineração, meteorologia, artes e muitas outras áreas [28], a base teórica para essa tecnologia data desde a década de 40, sendo que o primeiro modelo de RNA voltado para lidar com o problema de reconhecimento de padrões foi de fato desenvolvido por Frank Rosenblatt em 1958, modelo chamado de "O perceptron de Rosenblatt" [29].

### 2.2.1 Estrutura do neurônio

A teoria por trás do neurônio de uma RNA surgiu antes do conceito da rede propriamente dita. O primeiro neurônio artificial foi desenvolvido em 1943 por Warren McCulloch, um neurocientista, e Walter Pitts, um especialista em lógica e cognição [30]. Nesse modelo, Figura 2.3, tanto os *inputs* quanto *outputs* são valores booleanos, que eram processados por uma função $g$ e de acordo com o resultado desta, uma outra função $f$ determina a saída. Esse neurônio possui uma lógica simples, porém apesar de ter sido desenvolvido a mais de 70 anos, o fundamento se assemelha aos modelos atuais.

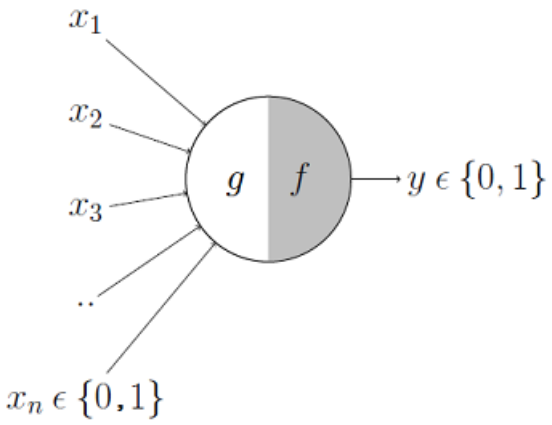



Figura 2.3: Neurônio artificial de McCulloch-Pitts [4].

Nas RNA's modernas, o neurônio recebe um vetor de entradas $x$ e aplica em uma função $g$ tal que

$$g(x_1, x_2, ..., x_i) = \sum_{n=1}^{i} W_n x_n + b_n, \tag{2.1}$$

em que $W$ representa um vetor de pesos e $b$ um viés, valores que são atualizados durante o treinamento da rede. O resultado da função dada pela Equação 2.1 é então usado como entrada para uma função de ativação não-linear $f(\cdot)$, que gera a saída do neurônio.

Seguem alguns exemplos de funções de ativação e uma breve descrição do seu comportamento:

- Função ReLU: Unidade Linear Retificada (do Inglês, *Rectified Linear Units*) é uma função que, dada uma entrada $x$, retorna $x$ se o mesmo for maior ou igual a 0 e retorna 0 caso contrário, de acordo com

$$ReLu(x) = \max(0, x); \tag{2.2}$$

- Função sigmoide (*Logsig*): Equação não linear que dado um número $x$ de entrada, retorna um valor real entre [0,1]. Observe na Equação 2.3 que, quanto maior o $x$, mais próximo de 1 está a saída e quando menor mais próximo de 0, representada de forma que

$$\sigma(x) = \frac{1}{1 + e^{-x}}; \tag{2.3}$$

- Função Tangente Hiperbólica (*Tansig*): Equação não linear que dado um número $x$ de entrada, retorna um valor real entre [-1,1]. Funciona de forma semelhante a Função Sigmoide, como se observa na equação

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}; \tag{2.4}$$

- Função Softmax: É uma função que converte um vetor de números em um vetor de probabilidades. Normalizando esses valores para um conjunto de números entre [0,1] e cuja soma total é 1, de forma que

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}. \tag{2.5}$$

Existem várias funções de ativação usadas atualmente, porém são descritas nessa seção apenas aquelas que são relevantes para o propósito deste trabalho.

### 2.2.2 Treinamento

A eficácia de uma rede neural é determinada por uma função de custo, que representa o erro entre o valor predito pelo modelo e o valor desejado. Existem diferentes definições para cálculo do erro, uma usada tradicionalmente é a do erro quadrático médio [31], descrita na função de custo $C$, tal que

$$C(W, b) = \left(\frac{1}{n}\right) \sum_{i=1}^{n} \left(y_i(x) - y_i'(x)\right))^2, \tag{2.6}$$

em que

$$y_i(x) = W \cdot x + b, \tag{2.7}$$

em que $n$ é o total de amostras, $y$ o valor desejado e $y'$ o valor de saída calculado pela rede.

Como podemos perceber, a função de custo e, por conseguinte, o desempenho de uma rede neural depende diretamente do valor dos pesos $W$ e do viés $b$ de cada neurônio. Para calcular os valores ótimos desses parâmetros que minimizem a função custo da rede, pode ser utilizado o algoritmo de otimização gradiente descendente estocástico para atualização dos parâmetros até se encontrar o menor valor da função $C(W, b)$, mostrado na Figura 2.4.

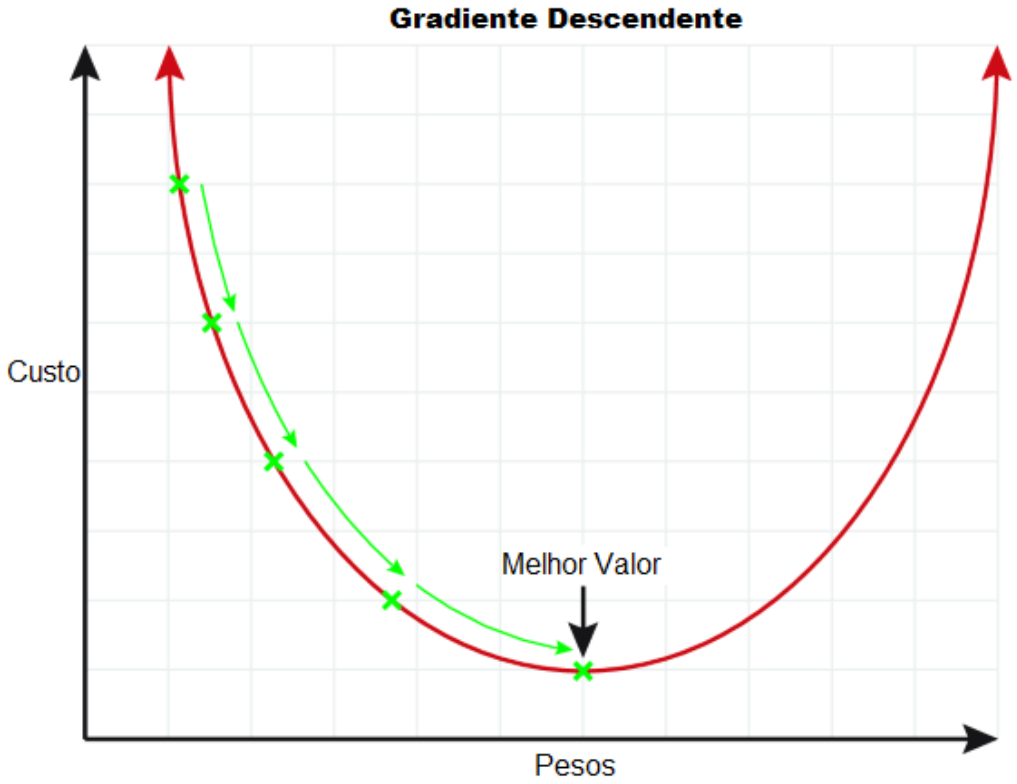



Figura 2.4: Gradiente descendente (adaptado de [5]).

Dado um conjunto inicial de dados de treino, nos parâmetros $W$ e $b$ de cada neurônio são atribuídos valores arbitrários, em que após uma época (em inglês, *epoch*), ou seja, depois que todos os dados de entrada são apresentados à rede, no sentido da camada de entrada para a camada de saída (*foward pass* Figura 2.5), é utilizado a técnica de retropropagação do erro (do inglês, *backpropagation*) para atualizar os valores dos pesos e vieses.

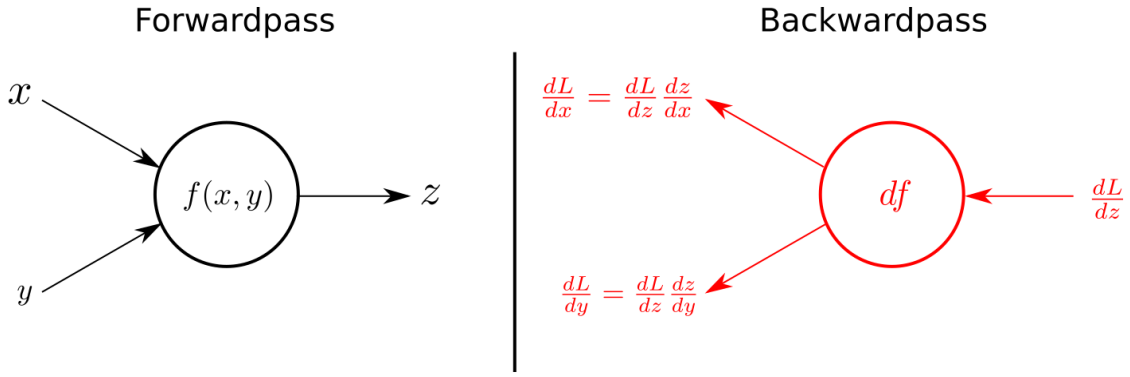



Figura 2.5: *Foward pass* e *Backward pass* [6].

Nesse processo, os pesos são recalculados a partir do fluxo inverso da rede, no sentido da camada de saída para a de entrada (*backward pass*, Figura 2.5), utilizando a regra da cadeia para sucessivamente obter as derivadas parciais da função de custo $C$ [32]. Os parâmetros $W$ e $b$ são atualizados de acordo com as Equações (2.8) e (2.9), respectivamente, em que o valor $\alpha$ representa a taxa de aprendizado,

$$W_{i+1} = W_i - \alpha \frac{\partial C(W)}{\partial W_i} \tag{2.8}$$

e

$$b_{i+1} = b_i - \alpha \frac{\partial C(b)}{\partial b_i}. \tag{2.9}$$

Esse mesmo processo em que os pesos são atualizados de acordo com a função de custo é realizado até que seus respectivos valores convirjam.

O método de otimização por gradiente descendente estocástico apesar de ser muito utilizado tem suas limitações. O algoritmo pode não funcionar bem caso a base de dados de treino fique muito grande e passe a não convergir [33], ou pode ocorrer do gradiente não ser direcionado para um mínimo local e não global, além de ser muito dependente do ajuste de sua taxa de aprendizado.

O algoritmo de otimização usado nas redes implementadas nesse trabalho será o ADAM (do Inglês, *Adaptive Moment Estimation*) [34], um método que altera dinamicamente os parâmetros da rede utilizando o primeiro e segundo momento dos gradientes, dados respectivamente por

$$P_i = \beta_1 P_{i-1} + (1 - \beta_1) \nabla C_i(W_{i-1}), \tag{2.10}$$

$$Q_i = \beta_2 Q_{i-1} + (1 - \beta_2) \nabla C_i(W_{i-1})^2, \tag{2.11}$$

em que $\beta_1$ e $\beta_2$ são os pesos dos gradientes do neurônio $i$.

Após o calculo de $P_i$ e $Q_i$, é aplicado uma correção a cada um dos momentos, dados por

$$\hat{P}_i = \frac{P_i}{1 - (\beta_1)^i}, \tag{2.12}$$

$$\hat{Q}_i = \frac{Q_i}{1 - (\beta_2)^i}, \tag{2.13}$$

e por fim a atualização dos pesos é dada por

$$W_i = W_{i-1} - \alpha \frac{\hat{P}_i}{\sqrt{\hat{Q}_i + \epsilon}}, \tag{2.14}$$

em que $\alpha$ corresponde à taxa de aprendizado.

O processo descrito é o que caracteriza o treinamento de uma rede neural propriamente dito.

### 2.2.3 Redes Perceptron Multicamadas

O modelo Perceptron Multicamadas (do Inglês, *Multilayer Perceptron*) (MLP) [35] consiste em uma rede do tipo *feedfoward* com uma camada de entrada, uma ou mais camadas escondidas e uma camada de saída, como vemos na Figura 2.6. O número de neurônios em cada camada pode variar de acordo com a necessidade do problema e, com exceção da primeira, todas as camadas aplicam uma função de ativação não-linear.

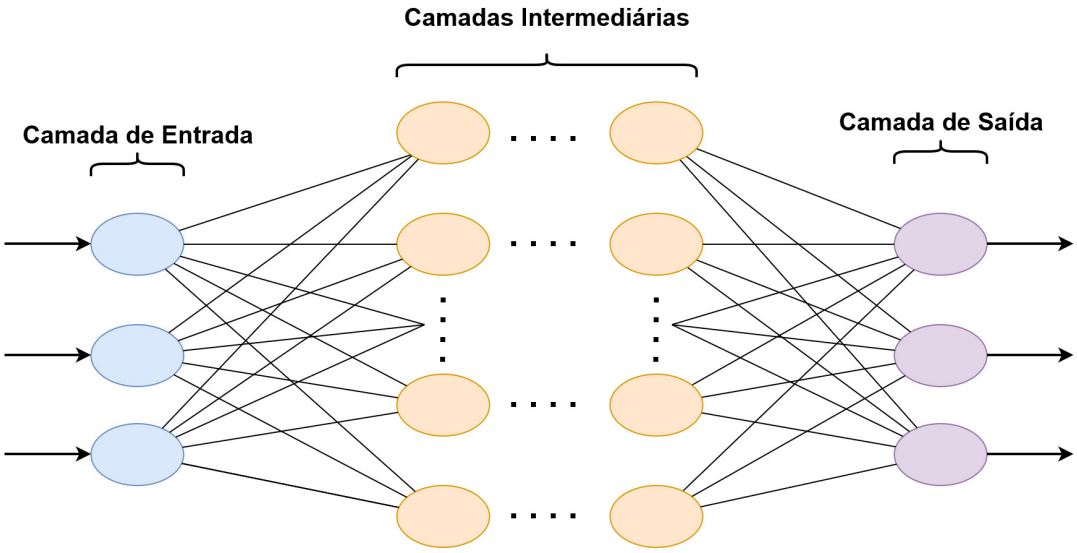



Figura 2.6: Perceptron Multicamadas.

A MLP se difere de um perceptron simples por conseguir realizar a distinção de dados que não são linearmente separáveis.

### 2.2.4 Redes Neurais Convolucionais

A rede MLP tem inúmeras aplicações porém possui limitações, como por exemplo na análise de um dado que possui uma natureza bidimensional como uma imagem, que computacionalmente é representada na forma de uma matriz. Uma rede MLP até consegue receber uma imagem como *input*, mas antes tem de transformar a matriz bi ou tri-dimensional em um vetor unidimensional, em que cada elemento do vetor corresponderá a um neurônio de entrada do perceptron. O problema dessa abordagem é que ela ignora a natureza multidimensional do objeto de entrada, causando assim uma perda de informação.

As Redes Neurais Convolucionais (do Inglês, *Convolutional Neural Networks*) (CNN) correspondem a um modelo *feedfoward* que lida com a natureza multidimensional da entrada, sendo ideal para trabalhar com imagens. Nesse tipo de rede, a estrutura análoga ao neurônio será composta de um filtro, ou *kernel*, que possui o mesmo número de dimensões do objeto de entrada, esse filtro percorre o objeto realizando o processo de convolução e possibilitando o aprendizado da hierarquia espacial do mesmo.

A estrutura geral de uma CNN, mostrada na Figura 2.7, possui uma sequência de blocos de camadas de convolução que aplicam funções de ativação e de redução da dimensionalidade (*pooling*), e no final da rede possuem camadas totalmente conectadas que então direcionam os dados a saída da rede em que a última camada aplica uma função de ativação para gerar o *output*.

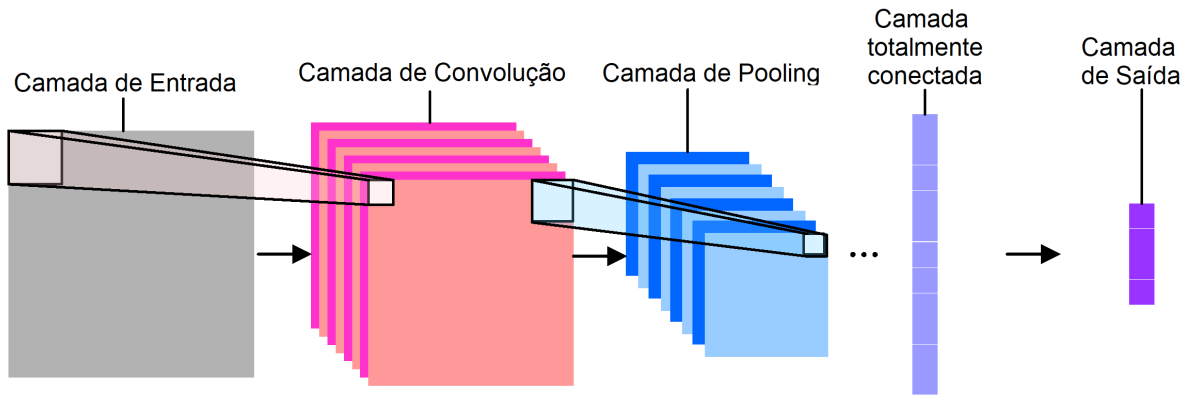



Figura 2.7: Estrutura básica da Rede Neural Convolucional (adaptado de [7]).

Abaixo é apresentada uma descrição mais detalhada do funcionamento de cada uma dessas estruturas:

- Camadas Convolucionais: nessa camadas, um ou mais kernels, obrigatoriamente menores que a matriz de entrada, aplicam operações de multiplicação entre os componentes do *kernel* e da entrada, elemento a elemento ao longo de toda a matriz, em que dado uma matriz de entrada $mat_1$ e um kernel $k_1$ de dimensões $m$ e $n$, o resultado da convolução $conv_1$ é calculado[36] como

$$conv_1[m,n] = mat_1[m,n] * k_1[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} mat_1[i,j] \cdot k_1[m-i, n-j]. \quad (2.15)$$

  Na Figura 2.8 podemos observar como a convolução é realizada na prática, em que a matriz na extremidade esquerda da figura é a matriz de entrada, a matriz adjacente

a ela o *kernel*, e o valor destacado na matriz na extremidade direita o valor de saída da convolução;

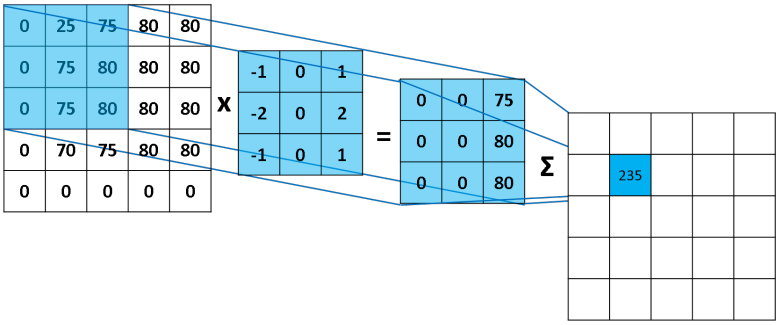



Figura 2.8: Exemplo de uma convolução entre matrizes [8].

- Camadas de *Pooling*: são camadas de redução da dimensionalidade do plano, tendo como função diminuir o custo computacional da saída da convolução e gerar uma saída que represente um mapa de características condensadas de uma sub-região. Existem vários tipos de redução de dimensionalidade, os mais comuns são o *max pooling*, em que aplica-se um filtro que retira o valor máximo das sub-regiões não sobrepostas da matriz inicial, e o *average pooling*, que tem como saída o valor médio da sub-região do filtro. Na Figura 2.9 observamos como cada um dos *poolings* funciona;

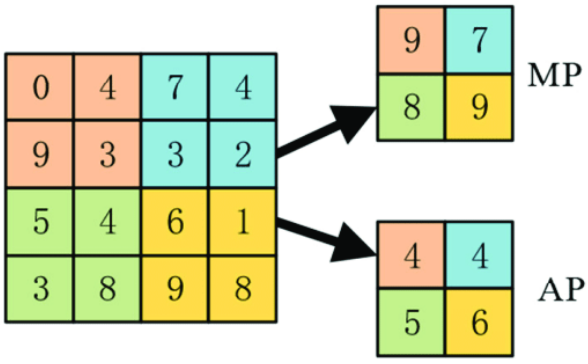



Figura 2.9: *Average Pooling* (AP) e *Max Pooling* (MP) [9].

- Camada Totalmente Conectada: camada na qual cada neurônio se conecta a todos os neurônios de saída da última camada de *pooling*, que tem a saída transformada em um vetor unidimensional e cada elemento do mesmo é multiplicado por um peso e somado a um viés.

- Camada de saída: gera um *output* aplicando ao vetor recebido da última camada totalmente conectada uma função de ativação, escolhida de acordo com o objetivo da rede.

## 2.3 Estado da arte

O uso de de técnicas de visão computacional pelas áreas da biologia, agricultura e áreas afins já é bem difundido e serve de ferramenta para realização de propósitos variados. Como em Anand Koirala *et al.* [37], em que foram testados sistemas usando redes CNN do tipo You Only Look Once (YOLO) [38] e CNN recursiva para classificação do estágio de floração em mangueiras utilizando fotografias das árvores retiradas a nível do solo. Ou em Enrique Apolo *et al.* [39] em que foi utilizado redes neurais para fazer a estimação do tamanho das frutas e a aferição do rendimento de plantações de laranjas, utilizando drones como forma de aquisição das imagens.

Em casos mais semelhantes a este trabalho, no qual são propostos sistemas de detecção de insetos, podemos encontrar casos como o desenvolvido em Suk-Ju Hong *et al.* [10], em que o objetivo era a identificação de três espécies de traças: *Helicoverpa assulta* (Figura 2.10a), *Spodoptera litura* (Figura 2.10b) e *Spodoptera exigua* (Figura 2.10c). Os dados foram coletados com uma câmera de resolução $2448 \times 2048$, posteriormente o *ground truth* foi definido com quatro classes, as três traças e uma rotulada como "desconhecida", utilizada para outras espécies de insetos.

(a) (b) (c)

Figura 2.10: Espécies de traças classificadas em Suk-Ju Hong *et al.* [10].

A Figura 2.11 mostra o fluxograma do processo proposto no trabalho, no qual foram testados sete classificadores diferentes (Faster R-CNN ResNet 101, FasterR-CNN ResNet 50, Faster R-CNN Inception v.2, R-FCN ResNet 101, Retinanet ResNet 50, RetinanetMobile v.2, e SSD Inception v.2), em que no melhor resultado possível se obteve uma acurácia de 98,06%, 77,59% e 95,11% na classificação das três espécies de traça respectivamente.

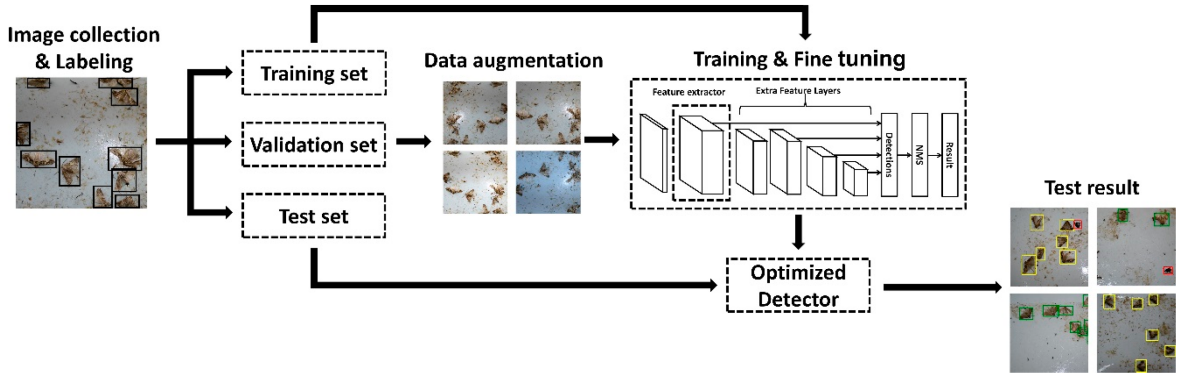
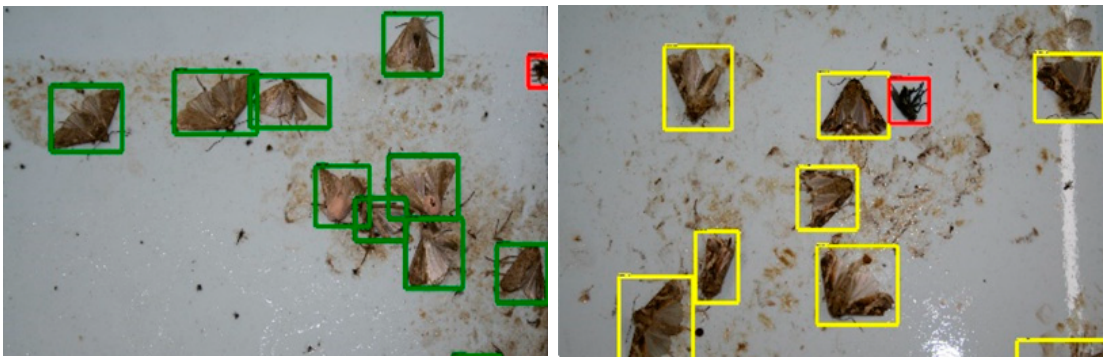



Figura 2.11: Fluxograma do processo de classificação proposto em Suk-Ju Hong *et al.* [10].

Podemos observar as saídas da rede de melhor performance em duas imagens diferentes, na Figura 2.12a e na Figura 2.12a.



(a) (b)

Figura 2.12: Exemplos da saída da rede de melhor performance em Suk-Ju Hong *et al.* [10].

Devido ao impacto da mosca-branca no cenário da agricultura mundial, existem diversos trabalhos que se utilizam de técnicas de visão computacional intentando a identificação da mesma. Em Jeremy Tusubira *et al.* [11] foi comparado o desempenho de dois modelos na identificação de moscas-brancas em folhas de mandioca. Os desenvolvedores desse projeto utilizaram um classificador *Haar Cascade* e uma rede *Faster-RCNN ResNet 101*, em que os modelos foram treinados com 2000 imagens, como na Figura 2.13, de resolução $4000 \times 1920$ píxeis.

Figura 2.13: Folha de mandioca com infestação de mosca-branca [11].

Nesse trabalho, com o classificador *Haar Cascade* se obteve uma precisão de 83%, e no outro modelo, *Faster-RCNN ResNet 101*, uma acurácia de 98%. Na Figura 2.14 pode ser visto um exemplo da saída do modelo *Haar Cascade* tendo como entrada a Figura 2.13.

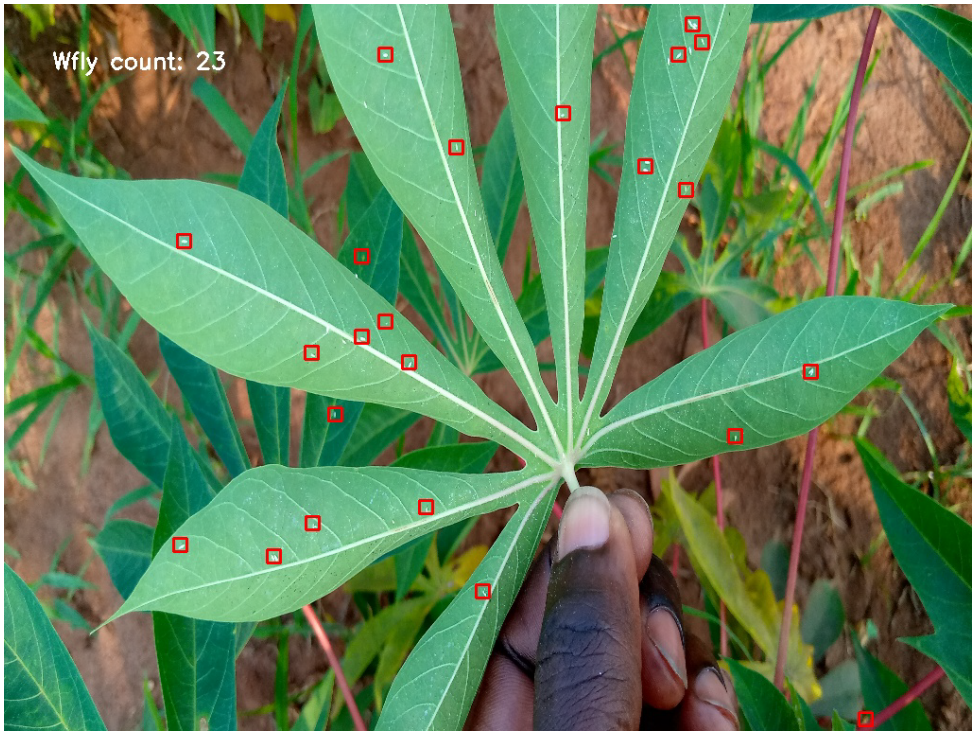



Figura 2.14: Saída do classificador [11].

O modelo supracitado apresenta uma robustez maior do que os modelos propostos nesse trabalho devido ao fato das imagens de entrada apresentarem maior complexidade, pois possuem um grau de homogeneidade muito menor do que as que usaremos, em que todas possuem o mesmo padrão: um fundo amarelo correspondente a armadilha e os insetos sobrepostos.

Mencionando trabalhos semelhantes, ou seja, usando imagens com o mesmo padrão visando a identificação da mosca-branca, podemos mencionar Cho *et al.* [12], em que o processo de identificação é baseado em um sistema que primeiro segmenta as imagens fundamentado no tamanho médio das moscas, Figura 2.15a, em que cada segmento é classificado como uma possível mosca-branca. Após a segmentação, a cor original dos segmentos é restaurada, Figura 2.15b, e feita a classificação baseada nos objetos que estiverem dentro de um espectro de cor pré-determinado usando o valor médio dos píxeis de mosca, observado na Figura 2.15c.

(a) (b) (c)

Figura 2.15: Processo de classificação em Cho *et al.* [12].

O sistema proposto apresenta uma acurácia na detecção de 94,56% de um total de 305 moscas distribuídas entre 10 imagens de 600dpi. Salienta-se que esse trabalho é implementado visando a detecção de outras duas pragas além da mosca-branca: afídios e tripes.

Outro modelo semelhante é o desenvolvido por Nieuwenhuizen *et al.* [13], em que se implementa uma *Faster R-CNN* usando como entrada o mesmo banco de dados que usaremos neste trabalho, 225 imagens de armadilhas adesivas amarelas pré-rotuladas de $5184 \times 3456$ obtidas usando o *Scoutbox* [40], subdivididas em segmentos de $1720 \times 1220$ de resolução como entrada da rede. O *ground truth* dos dados é feito identificando a mosca-branca e outros dois insetos predadores da mesma: *Macrolophus pygmaeus* e *Nesidiocoris*. Nesse modelo, obteve-se uma acurácia de 59% na identificação das moscas, porém foi visto que foram detectadas 242 moscas que não constavam no *ground truth*, e após a revisão e

reestruturação do mesmo, foi atingido uma acurácia de 87,4%. Um exemplo de saída da rede pode ser observada na Figura 2.16.


Figura 2.16: Saída do classificador [13].

Um segundo conjunto de 90 imagens de $4608 \times 3456$ píxeis fotografadas com smartfones foi utilizado como complemento do modelo. As imagens foram subdivididas em 6 segmentos de mesmo tamanho, no qual foi contado o total de insetos nas armadilhas, porém não foi feito o *ground truth*. Ao final, do total de 5574 moscas, 1592 *Macrolophus* e 26 *Nesidiocoris* identificadas manualmente, o classificador identificou 5521, 390 e 9 insetos, respectivamente. Se considerarmos apenas a mosca-branca, obteve-se uma precisão de 99%.

Neste capítulo foi mostrado alguns conceitos teóricos necessários para o entendimento dos modelos propostos nesse trabalho, além de alguns exemplos de aplicações que utilizam tecnologias semelhantes em aplicações com um contexto parecido com o nosso problema. No próximo capítulo será mostrado a metodologia do sistema proposto nesse trabalho, explicando desde as etapas mais basais, como a aquisição dos dados, até a aplicação dos modelos de classificação.

# Capítulo 3

# Sistema Proposto

Neste capítulo, definiremos como implementaremos a aplicação para detecção e classificação das moscas-brancas.

A metodologia proposta é dividida em cinco etapas principais, descritas na Figura 3.1:

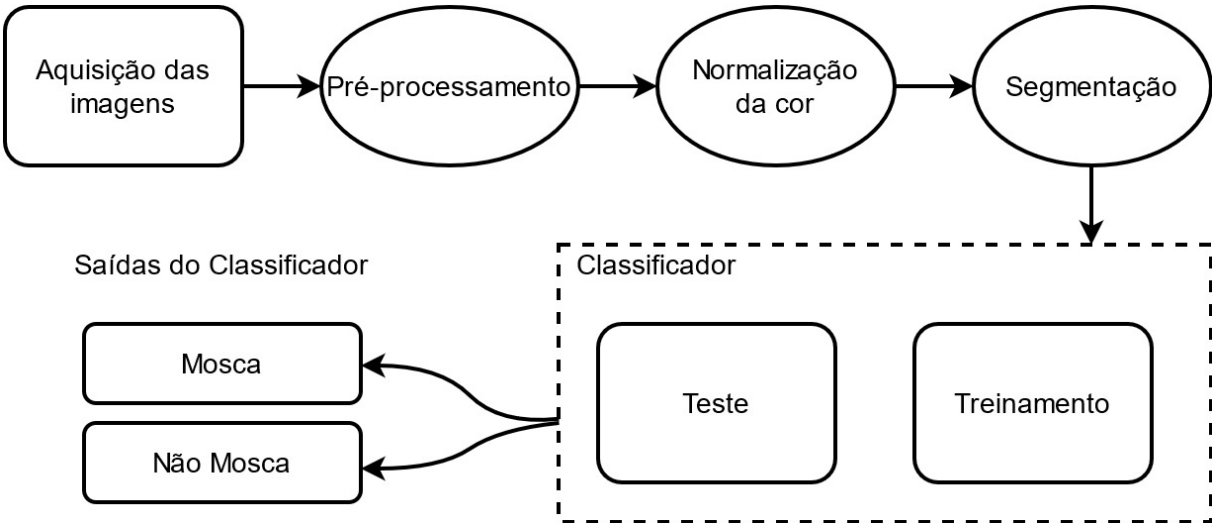



Figura 3.1: Diagrama do fluxo do trabalho.

Primeiramente, é feito um pré-processamento nas imagens, etapa que consiste no isolamento da área correspondente somente a armadilha e da retirada de um etiqueta de identificação, presente em todas as imagens do nosso banco, como podemos observar na Figura 3.2. Em seguida, é realizada a normalização da cor da armadilha e depois a segmentação, processo que consiste em gerar uma imagem binária, apenas píxeis pretos e brancos, com as possíveis regiões correspondentes as moscas.


Figura 3.2: Exemplo de uma imagem do banco de dados [14].

Foram escolhidas 21 imagens aleatórias, que contenham pelo menos uma mosca branca, dentro do banco de dados, para serem usadas para o treinamento dos modelos identificadores das moscas.

## 3.1 Pré-processamento das imagens

Nesta seção, descreveremos os passo tomados no pré-processamento das imagens brutas, com o intuito de melhorar e homogeneizar o aspecto visual das imagens, afim de otimizar as chances de classificação e o bom desempenho dos algoritmos testados na tentativa de resolução do nosso problema.

### 3.1.1 Isolar a armadilha

As imagens que formam o banco de dados não são compostas apenas da imagem isolada da armadilha adesiva. Parte da estrutura na qual a armadilha foi colocada sobreposta para ser fotografada também aparece, portando a primeira parte do pré-processamento é focada em ilhar apenas a parte da imagem que nos interessa.

A Figura 3.3 mostra um diagrama do processo, onde primeiro é selecionado o pixel modal, que é usado na binarização, que é feita usando a distância euclidiana entre o modal e todos os píxeis da imagem. Em seguida é realizada a identificação do maior segmento oriundo da imagem binária, e posteriormente feito um recorte na imagem baseado na coordenada dessa maior região.

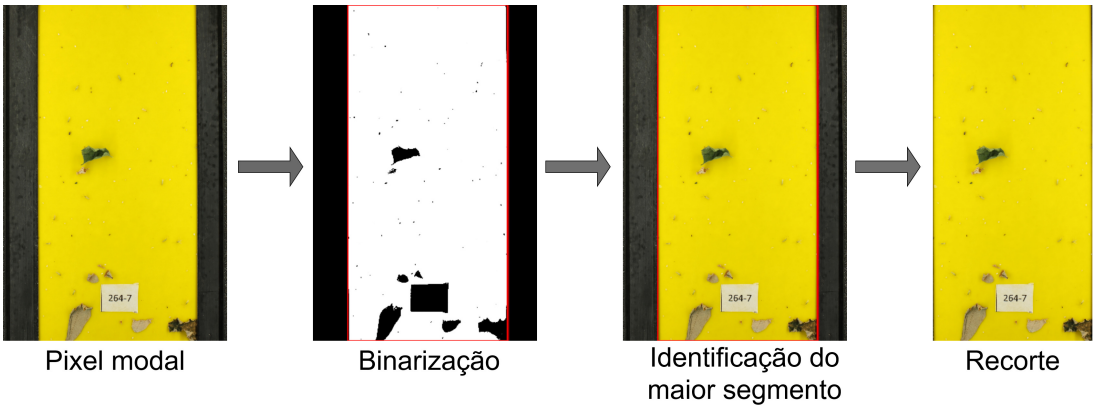



Figura 3.3: Fluxo do processo de isolar a região da armadilha [14].

Para esse intuito, é criada uma matriz de zeros do mesmo tamanho da imagem original, a seguir é calculada a cor modal $Moda = (M_R, M_G, M_B)$ da imagem colorida (espera-se alguma tonalidade de amarelo) e, em seguida, é calculada a distância euclidiana entre a moda e cada pixel da imagem $P(x, y) = (R(x, y), G(x, y), B(x, y))$, de forma que a distância $DE$ do pixel na posição $(x, y)$ e a cor modal é dada por

$$DE(x, y) = \sqrt{(M_R - R(x, y))^2 + (M_G - G(x, y))^2 + (M_B - B(x, y))^2}, \tag{3.1}$$

portanto, se o valor dessa distância entre o pixel na imagem de coordenada $(x, y)$ for menor que um *threshold* $Th_1$, o pixel de mesma coordenada $(x, y)$ da matriz de zeros é atribuído o valor 1, caso contrário permanece zero.

Ao final desse processo, criamos uma matriz lógica que corresponde a uma imagem binária (Figura 3.4), onde '1' representa um pixel totalmente branco e '0' um pixel totalmente preto.

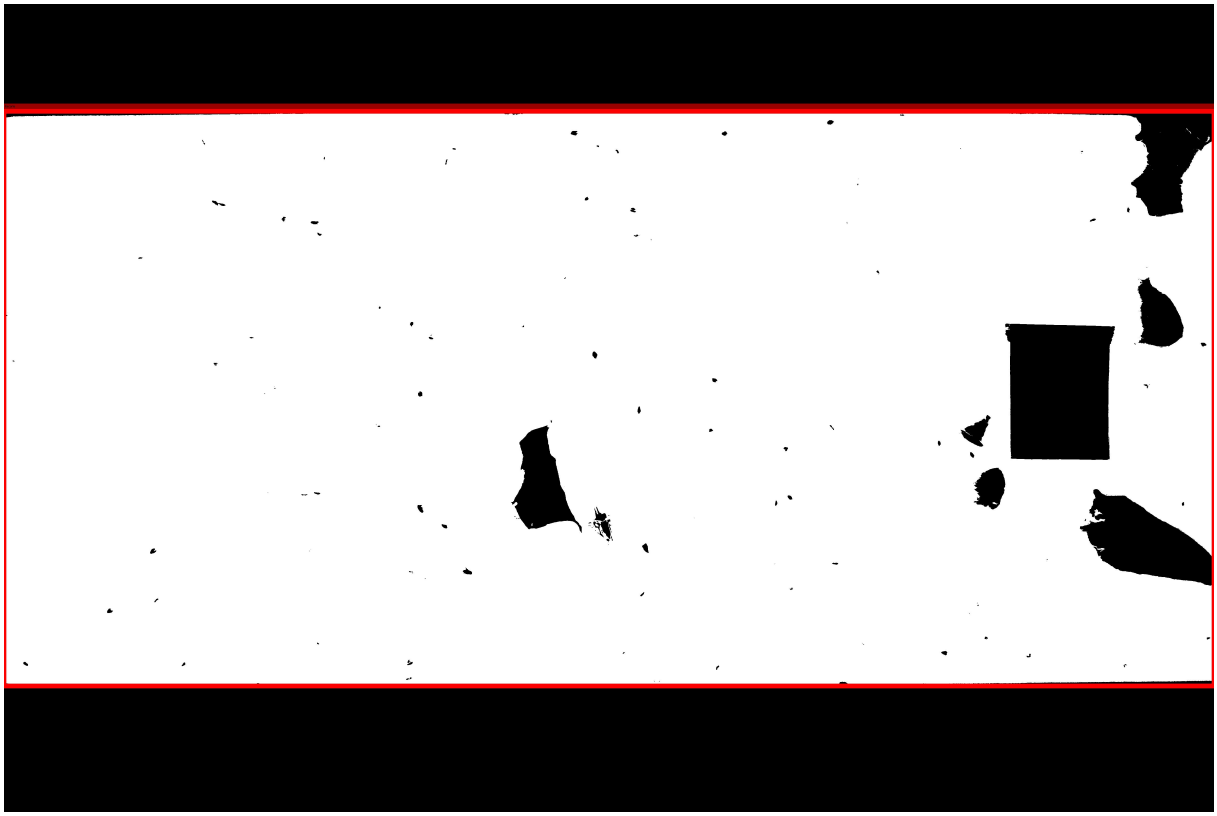


Figura 3.4: Região da armadilha, em vermelho, na imagem binária.

Usando um algoritmo de inundação [41] para detectar todas as regiões conectadas na imagem binarizada, extraímos as seguintes propriedades de cada região conectada, ou seja, formada apenas por píxeis brancos, da imagem binária: Área, dada pela quantidade total de píxeis, e Bounding Box, as coordenadas do menor quadrilátero que engloba toda a região. Com essas informações, fazemos um recorte na imagem original, Figura 3.5, utilizando as coordenadas do quadrilátero de maior área dentre as regiões conectadas.


Figura 3.5: Região da armadilha, em vermelho, na imagem colorida.

Por fim, o recorte resultante desse processo é uma área correspondente apenas pela armadilha amarela, como observado na Figura 3.6.


Figura 3.6: Recorte da região da armadilha.

### 3.1.2 Retirar a etiqueta

As imagens das armadilhas possuem uma etiqueta de identificação retangular, sendo que em algumas imagens essa etiqueta é feita com um pedaço de papel branco colocado sobre a armadilha, Figura 3.7a, e em outras foram inseridas digitalmente com algum *software* de manipulação de imagens, Figura 3.7b, sendo todas as etiquetas da cor branca.


(a) Etiqueta de papel.


(b) Etiqueta inserida digitalmente.

Figura 3.7: Dois tipos de etiqueta nas armadilhas.

Na Figura 3.8, observa-se um diagrama do processo de retirada da etiqueta, no qual primeiramente é feito a binarização baseada na distância euclidiana do pixel branco. Em seguida é identificada a maior região oriunda dessa binarização, e feito a pintura dessa região para o valor do pixel modal, como mostrado na imagem na extrema direita do diagrama.

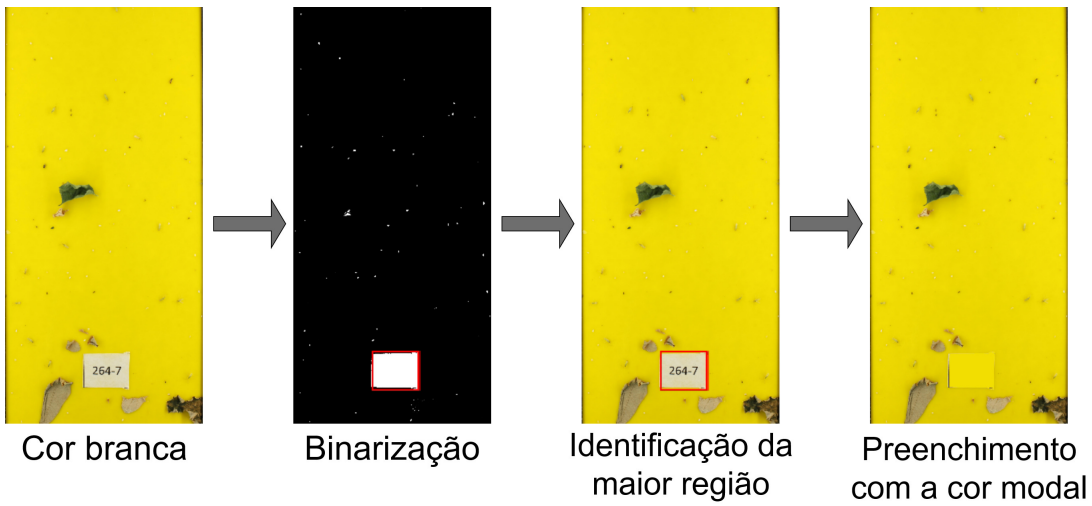



Figura 3.8: Fluxo do processo de retirada da etiqueta.

Para a retirada da etiqueta então, é preciso encontrar a maior região conectada que possua a cor branca 'pura', RGB(255, 255, 255), ou próximo à dela. Assim como no

primeiro passo, é criada uma matriz de zeros do mesmo tamanho da imagem original, e em seguida calculamos a distância euclidiana tridimensional entre o branco 'puro' e cada pixel da imagem, se o valor dessa distância for menor que um *threshold* $Th_2$, o elemento de mesma coordenada da matriz de zeros é definido atribuído o valor 1, caso contrário permanece zero.

Depois de gerar essa imagem binarizada, Figura 3.9, de maneira semelhante ao feito para a segmentação da armadilha amarela, é usado um algoritmo de inundação [41] para detectar o maior segmento dentro da mesma, e retirada a posição $(x, y)$ de cada pixel dentro dessa região, como observa-se na região do retângulo vermelho na Figura 3.10.

Figura 3.9: Em destaque, região da etiqueta de identificação na imagem binarizada.

Finalmente, retornamos na imagem colorida, e os píxeis com as mesmas coordenadas $(x, y)$ retirados anteriormente na imagem binarizada, Figura 3.9, são substituídos pelo valor do pixel modal da imagem, conforme observado na Figura 3.11.


Figura 3.10: Em destaque, região da etiqueta de identificação na imagem colorida.

Figura 3.11: Etiqueta substituída pela cor modal.

A imagem obtida no final desse processo contém apenas os elementos naturais capturados pela armadilha, isto é, insetos de diferentes espécies e outros elementos, que por conta de intempéries, ficaram presos na armadilha.

### 3.1.3 Normalização da cor

Fazendo uma análise dos histogramas da intensidade de cada canal de cor das imagens usadas no pré-processamento, observa-se um padrão em que a maior parte dos píxeis no canal R e G tem um valor alto (próximo de 255), e baixo no canal B (próximo de 0),

o que era de se esperar já que a maior parte da composição da imagem é um tom de amarelo. O problema é que as diferentes imagens do banco de dados não possuem uma iluminação homogênea, algumas estão muito claras, outras muito escuras. Este efeito fica nítido ao ver a extensão dos valores dos píxeis em cada histograma. A variação da maior parte da intensidade do canal B de uma imagem é de 0 a 30, em outra, de 0 a 100. Nos outros canais de cor, apesar do padrão ser o mesmo entre as imagens, também existem pequenas variações nos valores. Exemplos dessas diferenças em quatro imagens diferentes são mostradas na Figura 3.12.

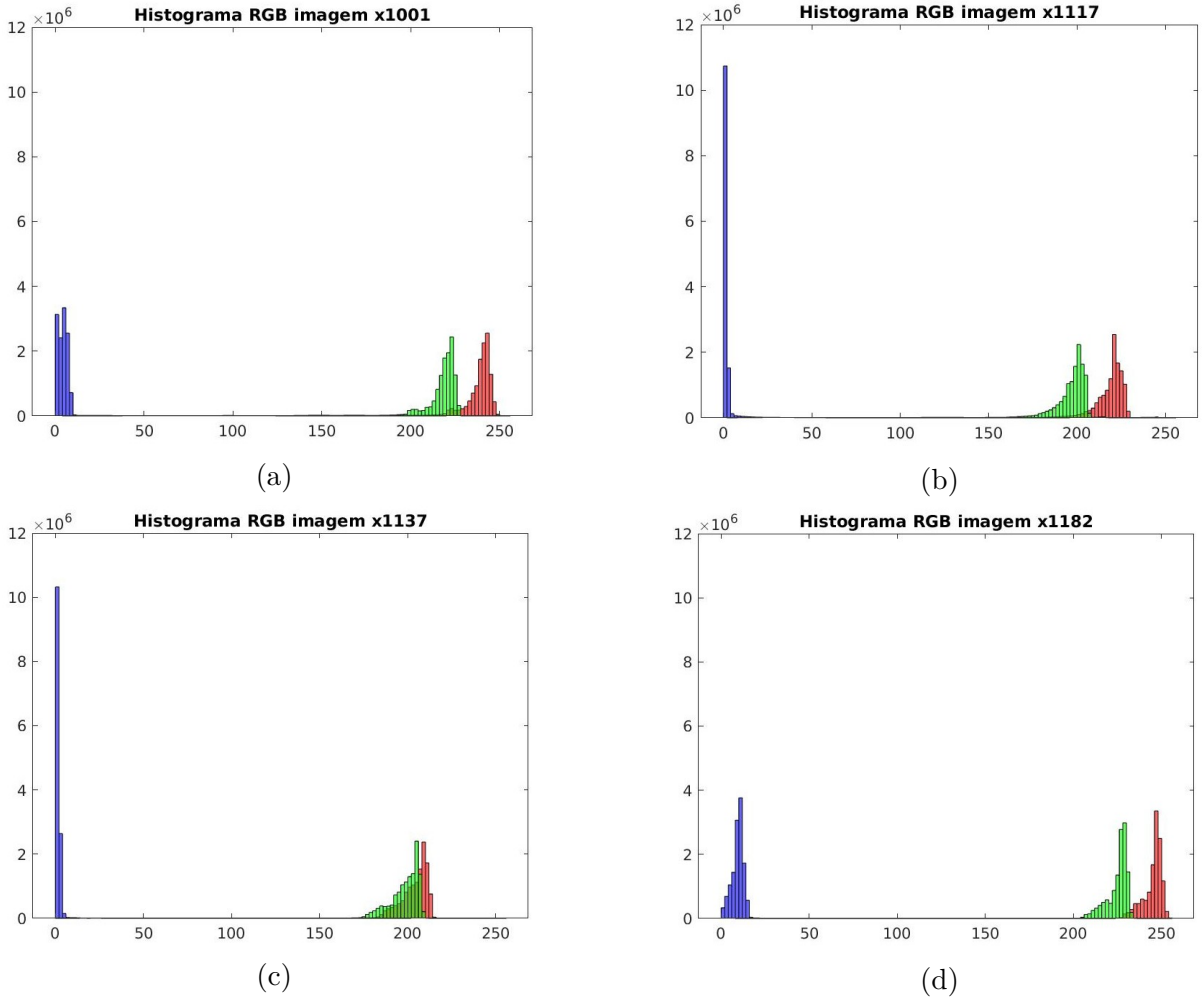



Figura 3.12: Histogramas de cor de quatro imagens diferentes: Imagem x1001(a), x1117(b), x1137(c) e x1182(d).

O foco desse trabalho é a identificação de moscas-brancas que, como o próprio nome já insinua e visto na Figura 1.1, apresentam-se como objetos que possuem em seus três canais de cor RGB um alto valor, se aproximando do branco puro na maioria dos casos.

Fazendo uma normalização 'tradicional', ou seja, apenas esticando a extensão dos valores do pixel em cada canal de forma que o ponto de menor intensidade seja 0 e o

maior seja 255, método também chamado de *contrast stretching* [42], apresentado na Figura 3.13.

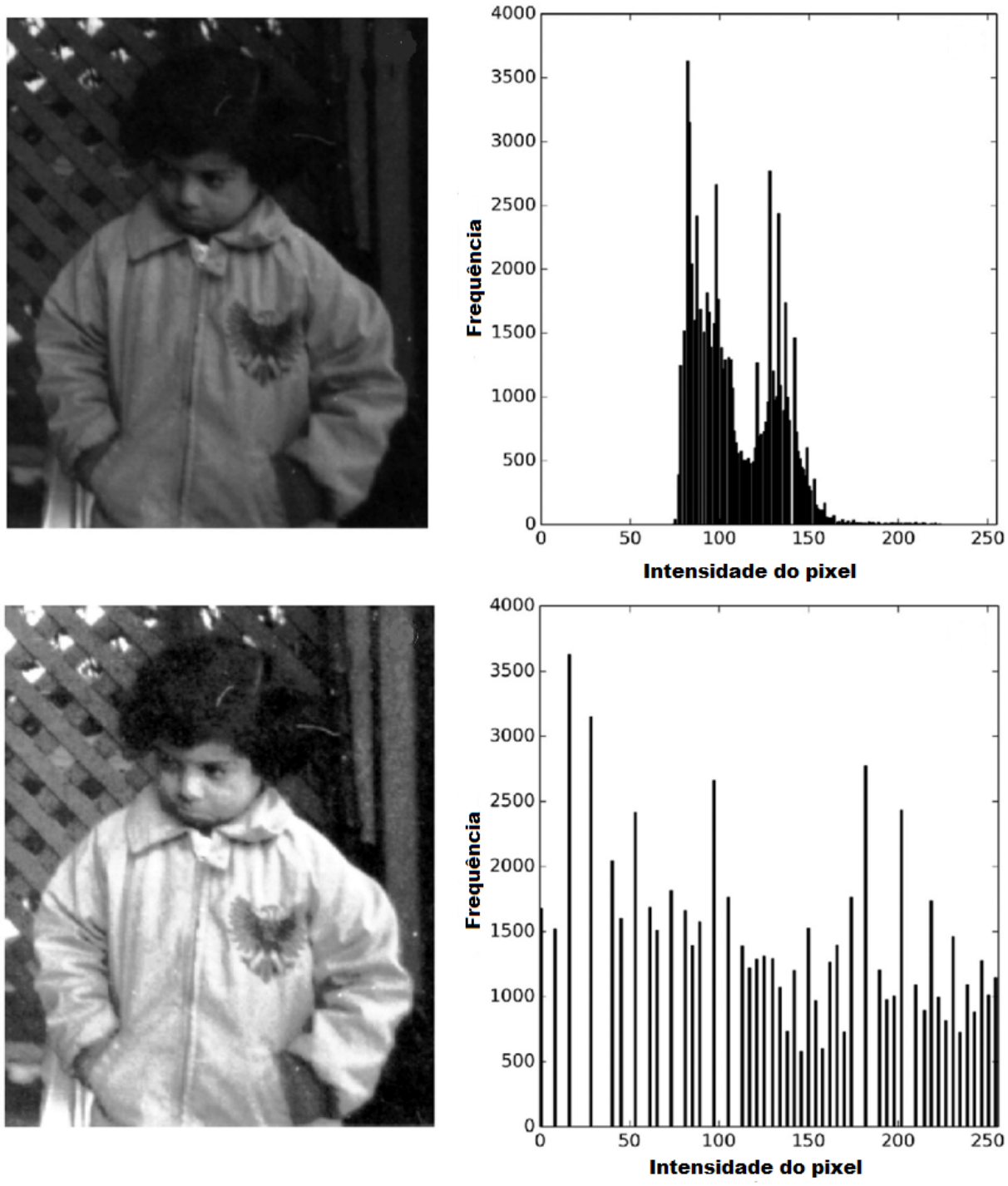



Figura 3.13: Normalização do canal de cor esticando os valores através do canal (adaptado de [15]).

No desenvolvimento do trabalho, foi constatado que as imagens com um canal de cor de extensão muito curto, ficam excessivamente claras, fazendo que boa parte da imagem

se torne uma mancha clara, como observa-se na Figura 3.14, ou algo que dificulta a visualização de moscas-brancas.

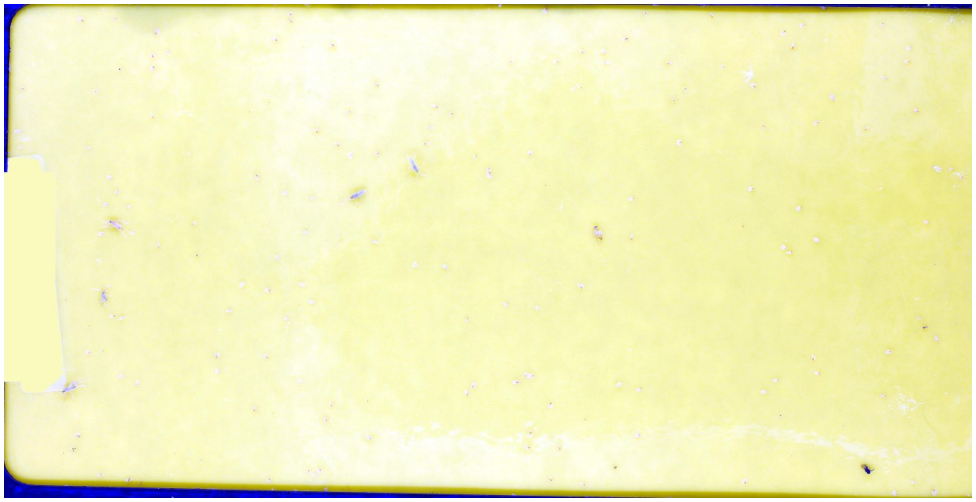


Figura 3.14: Exemplo do *contrast stretching* padrão em algumas imagens

Como descrito nos parágrafos anteriores, o valor modal dos píxeis da imagem são um tom de amarelo (R alto, G alto, B baixo) e o objeto a ser identificado um tom branco (R alto, G alto, B alto), portanto, podemos notar que o que de fato vai gerar a diferença do contraste da mosca com a armadilha é o valor do canal B, primordialmente.

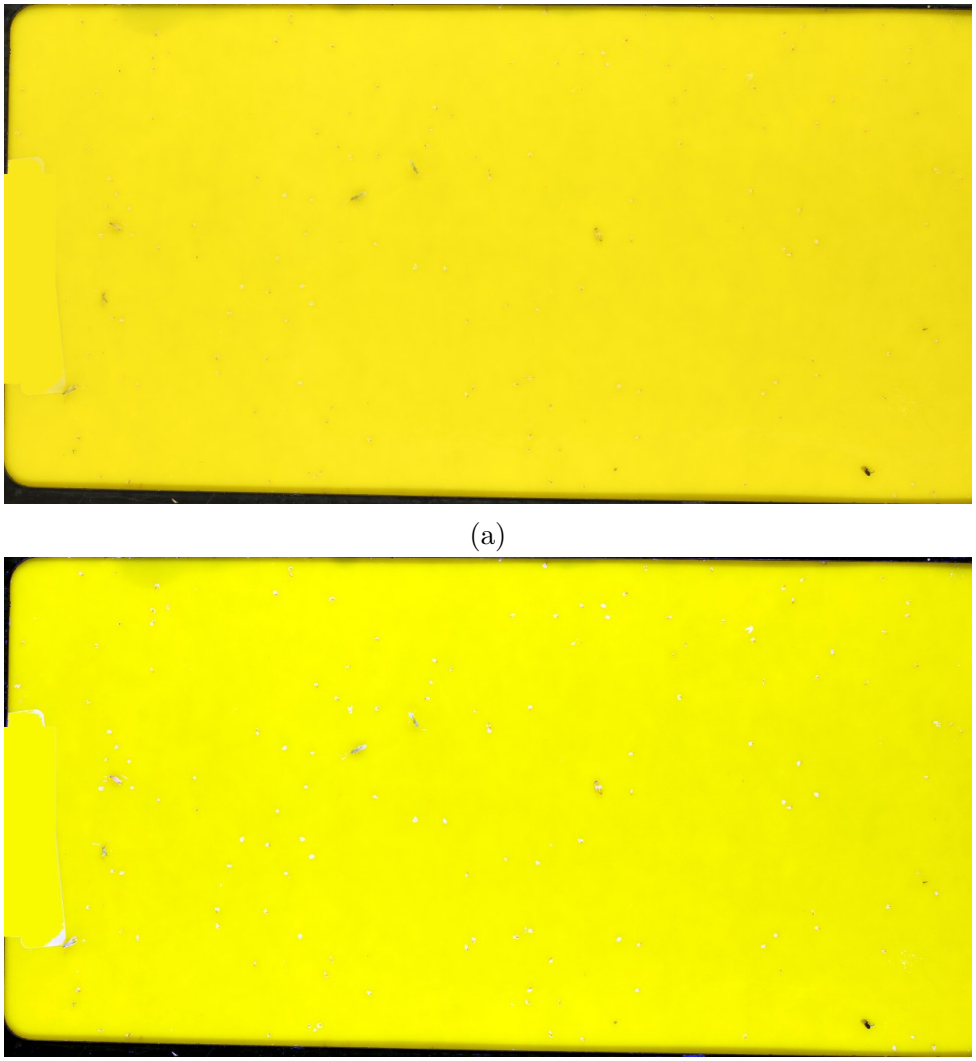
O MATLAB possui a função nativa `stretchlim` [43], que retorna um vetor de dois elementos com o limite inferior e superior de uma matriz de cor, sendo que por *default* a função exclui os 1% dos píxeis de menor e maior intensidade, e é usada pra auxiliar na normalização da cor.

Após uma série de testes, chegamos a um tipo de normalização baseada em um *contrast stretching* que, na matriz de cor B, sendo o limite inferior $l_I$ e o superior $l_S$, a $Moda$ e o desvio padrão $\sigma$, definimos

$$l_I = Moda + 2\sigma \tag{3.2}$$

e $l_S$ como o valor imediatamente inferior à porção que representa 1% do total de píxeis de maior valor.

Nos canais R e G, o limite inferior é delimitado pelo valor acima de 1% dos píxeis de menor intensidade e 1% abaixo dos de maior, ou seja, o *default* da função `stretchlim`. Os píxeis abaixo do limite são substituídos por 0 e os acima por 255.



(a)

(b)

Figura 3.15: Resultado do processo de normalização proposto, em que (a) é imagem original e (b) a Imagem normalizada.

Na Figura 3.15 vemos o resultado desse processo. Percebe-se a crucialidade da normalização da cor para este trabalho ao se comparar com a Figura 3.14, já que ambas são a mesma imagem aplicando normalizações diferentes. Na 3.14 é difícil conseguir identificar onde estão as moscas, já na 3.15b as mesmas são visualizadas com mais nitidez e contrastando melhor com o fundo amarelo.

## 3.2 Segmentação

Para segmentar a imagem, ou seja, tentar isolar as partes que são moscas. Primeiro, selecionamos manualmente uma amostra de $N$ píxeis pertencentes somente à moscas contidas

nas imagens, e em seguida calculamos o valor médio de cada matriz RGB, para se obter o valor de cor média de uma mosca.

Em seguida, criamos uma matriz de zeros do mesmo tamanho da imagem e em sequência calculamos a distância euclidiana tridimensional entre o valor do pixel médio de mosca $PM{=}(\mu_R, \mu_G, \mu_B)$ e cada pixel da imagem. Se o valor dessa distância for menor que um *threshold* $Th_3$, o elemento de mesma coordenada da matriz de zeros é definido como 1, caso contrário permanece zero, gerando uma imagem binária com segmentos, como observamos na Figura 3.16, onde as regiões que são candidatas à moscas foram destacadas com o quadrilátero vermelho.

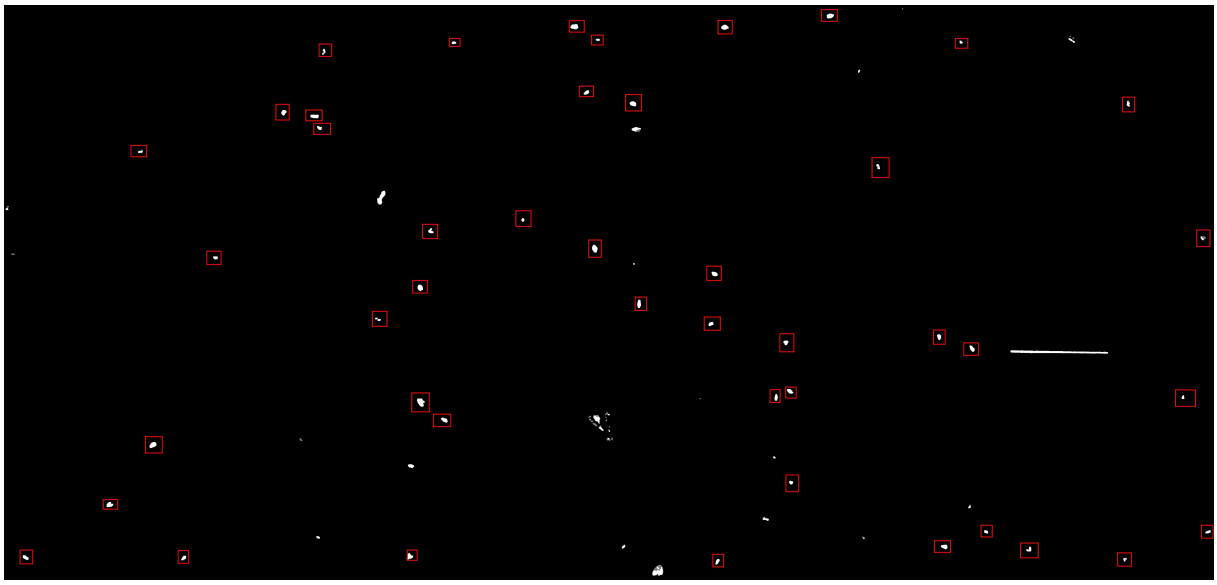


Figura 3.16: Imagem segmentada - Em destaque as regiões que são candidatas à serem moscas.

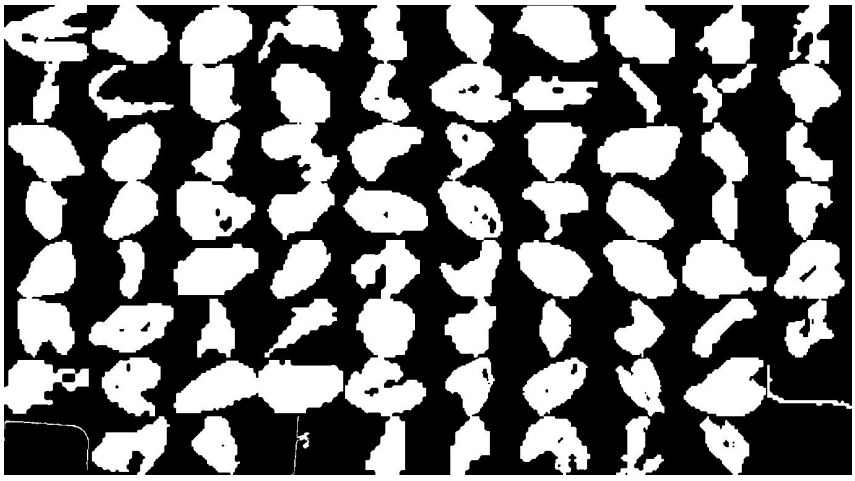


Figura 3.17: Regiões segmentadas da Figura 3.16 em detalhe (moscas e não-moscas).

Na Figura 3.17 são mostrados, com mais detalhes, os objetos resultantes da segmentação na Figura 3.16. Regiões muito pequenas, menores que 20 píxeis, não são inseridas no grupo de segmentos computados, afim de evitar que o número de segmentos que não fossem mosca fosse muito superior ao número de moscas, pois foi observado que não existiriam moscas-brancas com a área total de píxeis inferior a 20.

## 3.3 Extração das propriedades morfológicas dos segmentos

Com a imagem segmentada, usamos a função `regionprops` [16] do MATLAB para extrair propriedades relacionadas a morfologia de cada região, mosca e não-mosca. Esta função retorna outras características, porém depois de testes empíricos, apenas as seguintes propriedades relacionadas à morfologia de cada segmento foram relevantes ao trabalho.

- **Area**: Número total de píxeis da região;
- **BoundingBox**: Posição e tamanho do menor quadrilátero que engloba a região (Figura 3.18). Retornado como um array BB=$[x, y, x_{length}, y_{length}]$, onde o primeiro e o segundo elemento são, respectivamente, as coordenadas $(x, y)$ do vértice superior esquerdo do retângulo, a terceira posição, $x_{length}$, representa a largura e a quarta, $y_{length}$, a altura do quadrilátero;

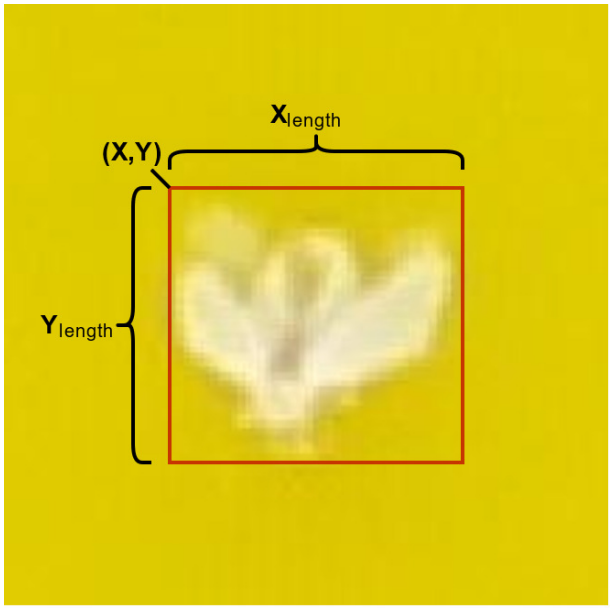



Figura 3.18: BoudingBox.

- **MajorAxisLength**: Tamanho (em píxeis) do maior eixo da elipse que possui o mesmo segundo momento que a região (Figura 3.19).
- **MinorAxisLength**: Tamanho (em píxeis) do menor eixo da elipse que possui o mesmo segundo momento que a região (Figura 3.19);

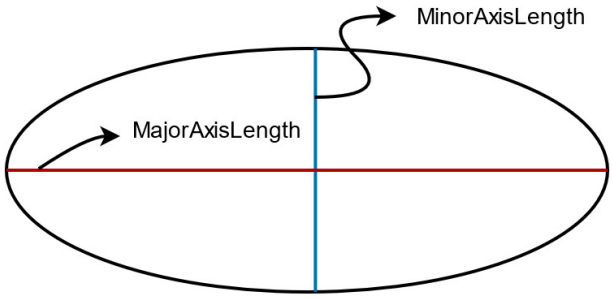



Figura 3.19: MajorAxisLength e MinorAxisLength.

- **Eccentricity**: É a razão entre o foco da elipse e o comprimento do maior eixo da mesma. Na prática, se esse campo for 1, a região é uma linha, se for 0 é um círculo.
- **Orientation**: Na Figura 3.20, angulo entre a linha do maior eixo da elipse e o eixo horizontal da região;

(a) (b)

Figura 3.20: (a) Representação do maior eixo da elipse e (b) o ângulo entre ele e o eixo horizontal [16].

- **Image**: Recorte da região segmentada Figura 3.21, armazenada como uma matriz binária;

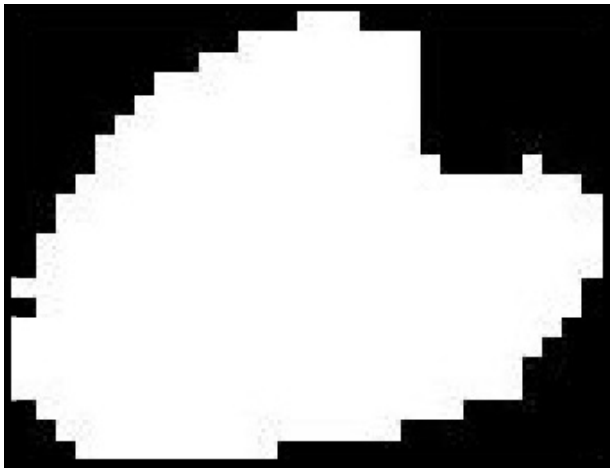


Figura 3.21: Image.

- **ConvexArea**: Número de píxeis do menor polígono convexo que engloba a região;
- **Solidity**: Proporção de píxeis do menor polígono convexo que engloba a região que também pertencem a mesma;
- **Extent**: Razão entre o total de píxeis da região e o total de píxeis do BoundingBox;
- **Perimeter**: Número de píxeis ao redor da região;

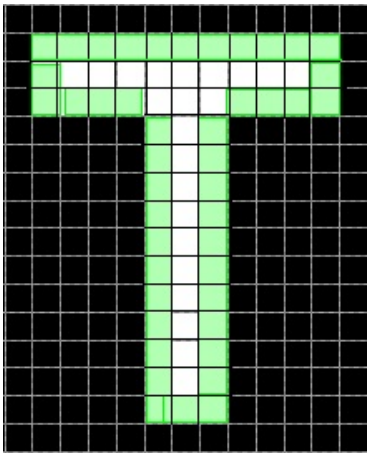


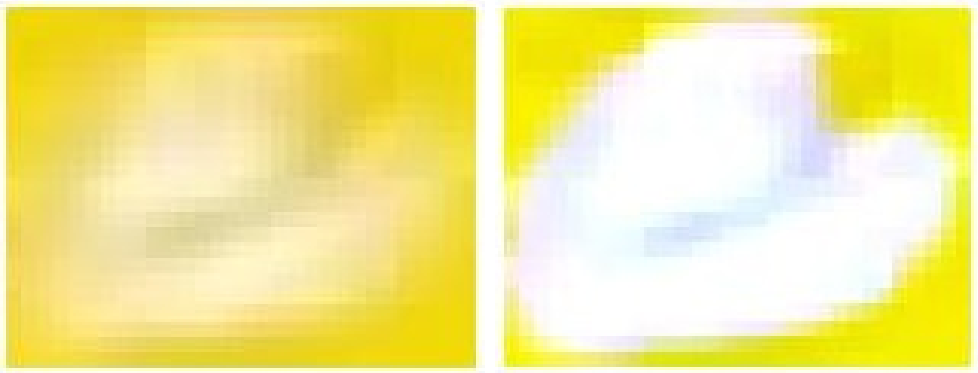
Figura 3.22: Perímetro [16].

Extraímos um total de 11 medidas morfológicas das regiões segmentadas, sendo que dessas, usaremos como parte do vetor de características que será usado como entrada em modelos classificadores apenas os dados escalares, ou seja: *Area, MajorAxisLength, MinorAxisLength, Eccentricity, ConvexArea, Solidity, Extent* e *Perimeter*. As variáveis *BoundingBox*, *Orientation* e *Image* serão usadas para auxílio em outras tarefas, como veremos posteriormente.

### 3.3.1 Extração de informações relacionadas a cor

Na seção anterior, extraímos um total de 11 medidas morfológicas das regiões segmentadas, sendo que dessas, usaremos como variáveis de entrada em modelos classificadores apenas os dados escalares, ou seja: *Area, MajorAxisLength, MinorAxisLength, Eccentricity, ConvexArea, Solidity, Extent* e *Perimeter*, as variáveis *BoundingBox*, *Orientation* e *Image* serão usadas para auxílio em outras tarefas, como veremos posteriormente.

Afim de incrementar o nosso vetor de características e extrair o máximo possível de informação acerca das regiões segmentadas, usamos as coordenadas do *BoundingBox* para recortar o objeto, na imagem original (sem normalização da cor) e na imagem normalizada. Na Figura 3.23 vemos um exemplo de regiões análogas nas duas conformações de cor.



(a) Recorte da imagem original (b) Recorte da imagem normalizada

Figura 3.23: Recorte de uma mosca usando a coordenada do *BoundingBox*.

Em seguida, separamos a matriz numérica que compõem os três canais de cores R, G e B e calculamos o PCA (*Principal Component Analysis*)[44] dos elementos dessas respectivas matrizes.

A componente principal resultante do PCA em cada canal de cor indica essa nova variável, o coeficiente de iluminação [45]. O MATLAB possui uma função chamada `illumpca` [46] que realiza esse cálculo, retornando um vetor de 3 elementos caso a entrada seja uma imagem colorida, os coeficientes de iluminação de cada canal RGB.

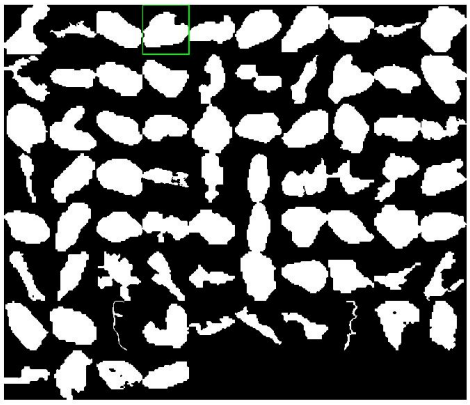
Esse processo foi aplicado tanto nos objetos retirados da imagem original, quanto naqueles da imagem normalizada, criando assim 6 novas variáveis, 3 coeficientes de iluminação do recorte da imagem original, que são: $Illum_R - ImgOriginal$, $Illum_G - ImgOriginal$, $Illum_B - ImgOriginal$; e 3 da cor normalizada, que denominaremos: $Illum_R - ImgNormalizada$, $Illum_G - ImgNormalizada$ e $Illum_B - ImgNormalizada$.

### 3.3.2 Rotulação manual dos segmentos

Após extraídas todas as propriedades descritas nas seções anteriores, precisamos determinar se cada uma das regiões segmentadas é uma mosca ou não, ou seja, criaremos os rótulos (*labels*) de cada uma das amostras.

Para a tarefa de classificação manual, escrevemos uma função simples para auxiliar, onde usamos as coordenadas do *BoundingBox* de cada região segmentada para desenhar um quadrilátero na imagem original colorida destacando-a, além de um quadro com todos os segmentos dessa imagem na forma binária para usar de auxílio se necessário. Podendo assim fazer a análise visual da região do segmento e determinar a correspondência dentro do recorte à uma mosca ou não.

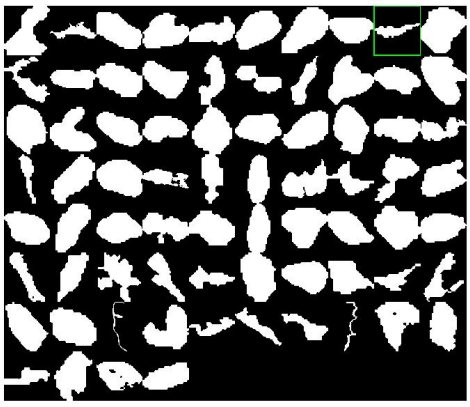

(a) Região destacada na imagem colorida



(b) Região destacada dentre as binárias

Figura 3.24: Exemplo da classificação de uma região que é uma mosca.


(a) Região destacada na imagem colorida



(b) Região destacada em uma grade com todas as regiões segmentadas da imagem

Figura 3.25: Exemplo da classificação de uma região que não é uma mosca.

Se o segmento destacado da imagem for de fato uma mosca-branca, como na Figura 3.24, recebe o valor 1, caso contrário, como na Figura 3.25, é atribuído o valor 0. Criamos um nova coluna no final da tabela chamada `'mosca'` e conforme cada região vai sendo classificada, inserimos nessa nova variável o respectivo valor (0 ou 1).

Todo esse processo de classificação foi auxiliado por um aluno de pós-graduação do LECOI (Laboratório de Ecologia de Insetos da UnB).

## 3.4 Classificação das regiões

Esta seção descreve os algoritmos usados na classificação das regiões obtidas pelas etapas de segmentação descritas anteriormente.

### 3.4.1 Validação cruzada

Antes da descrição dos métodos usados na classificação, visando uma melhor avaliação na capacidade de generalização dos modelos, salienta-se que para todos os algoritmos que serão usados adiante no trabalho, para treinamento e teste da nossa base de dados foi utilizado o método de validação cruzada *k-fold* com $k = 10$ *folds* [47].

Para a execução do *k-fold*, dividimos os dados em 10 porções, ou seja, cada um desses subconjuntos corresponde a 10% do total de amostras. Feita essa separação, é feito 10 iterações do modelo no qual os dados usados para teste serão as frações de 10% e os 90% restante usadas para o treinamento do modelo. A acurácia final do modelo $AccTotal$ é medida pela média das acurácias do conjunto de testes $Acc(i)$ em cada *fold* $i$, conforme

$$AccTotal = \frac{1}{10}\sum_{i=1}^{10}(\text{Acc(i)}), \tag{3.3}$$

onde Acc($i$) corresponde à medida de acurácia do modelo na iésima iteração do *k-fold*.

### 3.4.2 Normalização dos dados

Cada conjunto de variáveis é normalizado antes de serem usados nos classificadores, de forma que os valores de cada característica (*feature*) estejam distribuídos no intervalo $[0, 1]$. Dada uma variável $X$, seu valor $\bar{X}$ após a normalização será dado por

$$\bar{X} = \frac{X - X_{min}}{X_{máx} - X_{min}}, \tag{3.4}$$

normalizando os dados sem distorcer as proporções nos intervalos dos valores [48].

## 3.5 Classificadores

Nessa seção serão descritos os classificadores avaliados para a implementação deste trabalho.

### 3.5.1 KNN

O KNN (do inglês, *K-Nearest Neighbors*) [49] é um algoritmo de aprendizagem supervisionada usado para classificação de objetos de acordo com a semelhança com os **K** objetos mais próximos a ele (vizinhos), onde, a classe dos objetos com maior proximidade, calculado usando algum tipo de medida de distância (tipicamente distância euclidiana), determina a classe do elemento. No exemplo da Figura 3.26, podemos observar como o algoritmo funciona tendo apenas duas variáveis, dado um $K$ igual a 3 e 5, onde nesse exemplo o elemento (estrela) seria classificado como da Classe B se $K = 3$, pois dos 3 vizinhos mais próximos, 2 são dessa classe. Caso $K = 5$, o elemento será classificado como pertencente a Classe A, pois dos 5 vizinhos mais próximos, 3 são dessa classe.

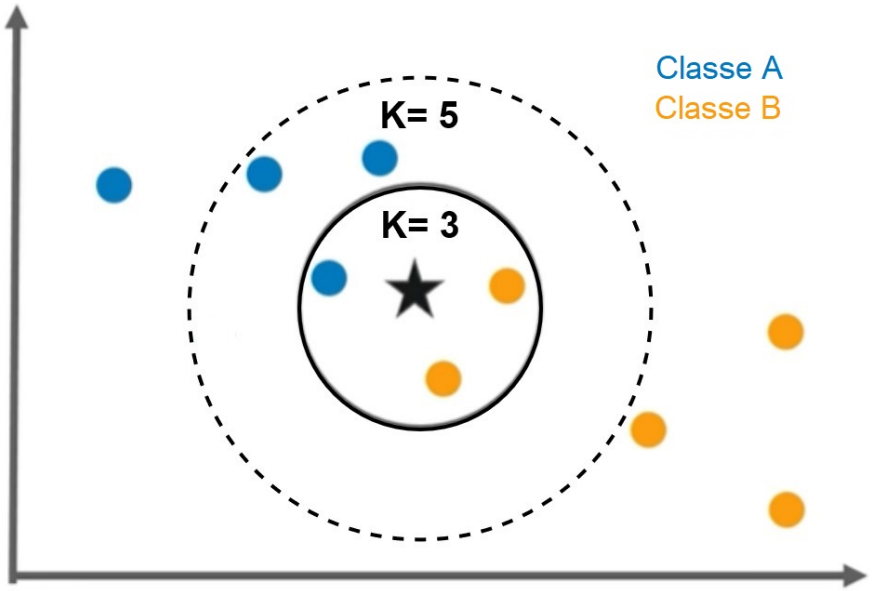



Figura 3.26: Exemplo de um KNN com K=3 e K=5.(adaptado de [17]).

Para que apenas uma classe seja escolhida, o valor de $K$ deve ser sempre um número ímpar, para impedir a possibilidade de um elemento ter o mesmo número de vizinhos pertencentes a duas ou mais classes diferentes.

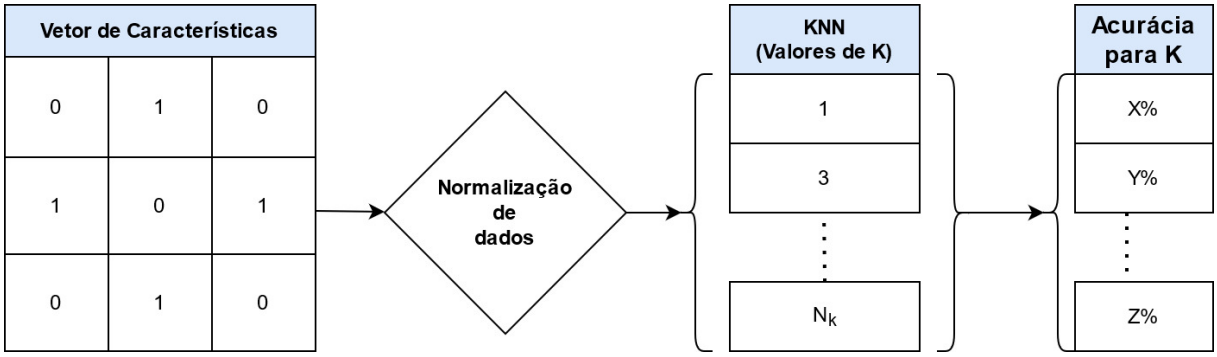



Figura 3.27: Fluxo da implementação do KNN.

Afim de descobrir um número $K$ de vizinhos que otimize a acurácia, implementamos um KNN para todo número ímpar entre 1 e $N_k$. O diagrama da Figura 3.27 mostra esse fluxo de execução.

### 3.5.2 Multilayer Perceptron

Implementaremos duas redes neurais MLP, uma que usa como parâmetros de entrada as variáveis do vetor de características e outra que usa a matriz do recorte na forma de um vetor unidimensional de tamanho **D**, tal que $D = [L \times A \times d]$, onde $L$ é a largura, $A$ a altura e $d$ o número de dimensões do espaço de cores da imagem.

#### Implementação com o vetor de características

Essa implementação da rede MLP conta com 14 neurônios na camada de entrada, uma para cada variável da tabela, $N$ neurônios na única camada escondida utilizando a função de ativação *Tansig* e 2 na camada de saída, um para cada classe, aplicando a função de ativação *Softmax* e outra implementação utilizando a *Logsig*.

#### Implementação com os recortes

Nessa implementação, transforma-se a imagem em um vetor unidimensional que conta com $D$ neurônios de entrada, N neurônios na única camada escondida aplicando a função de ativação *Tansig* e 2 na camada de saída aplicando a função de ativação *Softmax*.

### 3.5.3 CNN Rasa

Implementaremos um classificador CNN em que nessa rede, usaremos como dados de entrada os recortes das imagens segmentadas, sendo uma rede rasa, com no máximo 5 blocos de camadas de Convolução + ReLU + Pooling, seguido de uma camada totalmente

conectada e uma camada final de saída que aplica a função de ativação *Softmax*. Cada camada de convolução conta com $n$ kernels de tamanho [2×2].

# Capítulo 4

# Resultados Obtidos

Neste capítulo serão apresentados os resultados obtidos nos processos prévios a classificação dos dados, como a geração do vetor de características, além da análise dos resultados dos modelos propostos.

O *software* utilizado em todas as etapas do projeto foi o MATLAB versão R2018b. Todos os testes foram feitos usando um computador pessoal com um processador Intel Core i7-4510u CPU 2,6GHz, memória 8GB e sistema operacional Windows 10 64 bits. O código fonte pode ser encontrado no link: `https://github.com/VascoMonteiroNeto/MoscaBranca-Classificador-TG`.

A acurácia de cada modelo é medida pela proporção de objetos classificados corretamente, de acordo com as duas possíveis classes: mosca e não-mosca. Usaremos uma matriz de confusão, Figura 4.1, como parâmetro de medida da performance.

| | | **Detectada** | |
|---|---|---|---|
| | | **Sim** | **Não** |
| **Real** | **Sim** | Verdadeiro Positivo (VP) | Falso Negativo (FN) |
| | **Não** | Falso Positivo (FP) | Verdadeiro Negativo (VN) |

Figura 4.1: Matriz de confusão [18].

A acurácia final é medida de forma que

$$Acurácia = \frac{VP + VN}{VP + VN + FP + FN}, \tag{4.1}$$

em que os 4 parâmetros representam:

- Verdadeiro positivo (VP): item classificado corretamente como mosca.

- Verdadeiro negativo (VN): item classificado corretamente como não-mosca.
- Falso positivo (FP): item classificado erroneamente como mosca.
- Falso negativo (FN): item classificado erroneamente como não-mosca.

## 4.1 Thresholds

No Capítulo 3, definimos o uso de 3 *thresholds* $Th_1$, $Th_2$ e $Th_3$, nos processos de isolar a armadilha (Seção 3.1.1), retirar a etiqueta (Seção 3.1.2) e segmentação (Seção 3.2), respectivamente. Através de testes empíricos, o valor foram definidos como $Th_1 = 100$, $Th_2 = 100$ e $Th_3 = 50$.

Para auxílio na definição do *threshold* $Th_3$, foram extraídos 130 píxeis pertencentes a moscas-brancas dentro das imagens. A Tabela 4.1 mostra a média e o desvio padrão de cada canal de cor dos píxeis de mosca.

Tabela 4.1: Pixel de mosca-branca.

| Média e desvio padrão do pixel de mosca-branca | | |
|---|---|---|
| Canal R | Canal G | Canal B |
| Média: 236<br>Desvio padrão: 17,6 | Média: 234<br>Desvio padrão: 18,7 | Média: 245<br>Desvio padrão: 18,2 |

Além de servir como apoio na definição dos *thresholds*, a tabela mostra o perfil de cor acerca do objeto que buscamos classificar nesse trabalho.

## 4.2 Ground Truth

Usaremos um banco de dados composto de 283 imagens de armadilhas adesivas amarelas tradicionais [14]. Todas as imagens possuem $3456 \times 5184$ píxeis.

Apesar do banco de dados possuir um *ground truth*, ao iniciar os trabalhos, percebemos que o mesmo não era confiável, pois deixava de identificar algumas moscas em certas imagens.

Por esse motivo, o *ground truth* do banco foi complementado utilizando o software *labelimg* [50], o mesmo que já tinha sido usado para fazer o *ground truth* prévio. O procedimento foi feito com o auxílio de um estudante de pós-graduação do Laboratório de Ecologia de Insetos da UnB.

Para criação do nosso vetor de características que será usado como *input* de modelos, foram processadas 21 imagens aleatórias dentro do banco, com pelo menos uma mosca, nas quais foram obtidas 1430 regiões oriundas da segmentação.

A partir dessas regiões, foram computadas as seguintes propriedades, descritas na Seção 3.3, para inserção no vetor de características: *Area*, *MajorAxisLength*, *MinorAxisLength*, *Eccentricity*, *ConvexArea*, *Solidity*, *Extent* e *Perimeter*.

Além destas, também são inseridas no vetor os coeficientes de iluminação descritos na Seção 3.3.1: $Illum_R - ImgOriginal$, $Illum_G - ImgOriginal$, $Illum_B - ImgOriginal$, $Illum_R - ImgNormalizada$, $Illum_G - ImgNormalizada$ e $Illum_B - ImgNormalizada$.

A estrutura final do vetor de características das regiões pode ser observada na Tabela 4.2. Ao todo, ele é composto por 14 variáveis relacionadas a cada um dos objetos oriundos da segmentação.

A estrutura do vetor de características das regiões pode ser observada na Tabela 4.2.

Tabela 4.2: Vetores de características das regiões segmentadas(sem normalização).

| Variável | Região 1 | Região 2 | Região 3 | ... | Região 1430 |
|---|---|---|---|---|---|
| Area | 134 | 203 | 325 | ... | 81 |
| MajorAxisLength | 21,362 | 35,672 | 28,333 | ... | 13,643 |
| MinorAxisLength | 10,820 | 9,795 | 14,863 | ... | 8,137 |
| Eccentricity | 0,862 | 0,961 | 0,851 | ... | 0,802 |
| ConvexArea | 176 | 278 | 342 | ... | 91 |
| Solidity | 0,761 | 0,730 | 0,950 | ... | 0,890 |
| Extent | 0,525 | 0,527 | 0,650 | ... | 0,692 |
| Perimeter | 54 | 78 | 67 | ... | 34 |
| $Illum_R - ImgNormalizada$ | 0,665 | 0,574 | 0,635 | ... | 0,634 |
| $Illum_G - ImgNormalizada$ | 0,635 | 0,604 | 0,631 | ... | 0,630 |
| $Illum_B - ImgNormalizada$ | 0,391 | 0,551 | 0,444 | ... | 0,446 |
| $Illum_R - ImgOriginal$ | 0,730 | 0,698 | 0,705 | ... | 0,744 |
| $Illum_G - ImgOriginal$ | 0,647 | 0,657 | 0,648 | ... | 0,665 |
| $Illum_B - ImgOriginal$ | 0,218 | 0,281 | 0,287 | ... | 0,056 |

Já a Tabela 4.3 mostra um exemplo de *input* dos modelos que recebem de entrada os recortes. As entradas serão matrizes tridimensionais, caso o recorte seja colorido, ou matrizes bidimensionais, caso seja uma imagem em escala de cinza ou binária.

Tabela 4.3: Tabela com os recortes das regiões segmentadas.

| **Variável** | Região 1 | Região 2 | Região 3 | ... | Região 1430 |
|---|---|---|---|---|---|
| Imagem | $18 \times 16 \times 3$ | $12 \times 36 \times 3$ | $21 \times 26 \times 3$ | ... | $70 \times 32 \times 3$ |

Todas as regiões segmentadas foram classificadas manualmente como descrito na Seção 3.3.2. Dos 1430 objetos, 507 são da classe mosca e 923 são da classe não-mosca.

## 4.3 KNN

Na implementação do algorítimo KNN, afim de encontrar o melhor número possível de K, ou seja, o número de vizinhos em que teremos o melhor resultado, o algorítimo foi testado com todos os números ímpares entre 1 e 31.

Como parâmetros de entrada do modelo foram utilizadas as 14 variáveis do vetor de características das 1430 amostras, além de usado o método de validação cruzada *k-fold* com k=10 *folds*, e da normalização de cada variável da tabela para um alcance entre 0 e 1.

Tabela 4.4: Tabela com o número de vizinhos e os respectivos resultados.

| **Número de vizinhos (k)** | **Acurácia** |
|---|---|
| 1 | 92,80% |
| 3 | 94,34% |
| 5 | 94,34% |
| 7 | 94,13% |
| 9 | 94,27% |
| 11 | 94,13% |
| 13 | 94,06% |
| 15 | 93,78% |
| 17 | 93,71% |
| 19 | 93,78% |
| 21 | 93,22% |
| 23 | 93,57% |
| 25 | 93,36% |
| 27 | 93,15% |
| 29 | 93,43% |
| 31 | 92,94% |

Podemos observar na Tabela 4.4, que nesse modelo os valores ótimos para K são 3 e 5 vizinhos, empatados com o mesmo valor de acurácia 94,34%.

## 4.4 Multilayer Perceptron

Na implementação da rede MLP, salienta-se que foram testadas várias entradas diferentes, além de variações quanto às funções de ativação e quanto ao número de neurônios nas camadas de entrada e escondida, porém o número de neurônios na camada de saída é o mesmo em todas as redes testadas, ou seja, o número de classes, mosca e não-mosca.

As redes MLP foram treinadas por 1500 épocas, utilizando a função de entropia cruzada para cálculo do custo e ADAM como algorítimo de otimização.

### 4.4.1 Implementação com o vetor de características

Nessa implementação, usamos como parâmetros de entrada da rede os valores da tabela escalar, ou seja, ela tem 14 neurônios de entrada e dois na camada de saída. Na camada intermediária $N_{Escondida}$, iniciamos os testes com um número de neurônios igual a 6, representando a raiz quadrada da multiplicação do número de neurônios na primeira, $N_{Entrada}$, e última camada $N_{Saida}$:

$$N_{Escondida} = \sqrt{N_{Entrada} \times N_{Saida}}. \tag{4.2}$$

Em seguida foi testado o dobro deste valor, e posteriormente com 20 e 30 neurônios.

Além da variação do número de neurônios na camada escondida, foi testado diferentes funções de ativação na última camada. A Tabela 4.5 mostra os resultados usando a função *softmax* e a Tabela 4.6 usando a função *logsig*.

Tabela 4.5: Implementação usando *softmax* como função de ativação na camada de saída

| **Número de neurônios** | **Acurácia** |
|---|---|
| 6 | 93,01% |
| 12 | 95,10% |
| 20 | 95,17% |
| 30 | 94,20% |

Tabela 4.6: Implementação usando *logsig* como função de ativação na camada de saída

| **Número de neurônios** | **Acurácia** |
|---|---|
| 6 | 84,90% |
| 12 | 92,31% |
| 20 | 94,48% |
| 30 | 91,26% |

Observou-se que o melhor resultado foi obtido com 20 neurônios na camada escondida e utilizando a função *softmax* na saída.

### 4.4.2 Implementação com os recortes

Nessa seção, descreve-se como foi realizada a implementação da MLP usando como parâmetros de entrada o recorte transformado em um vetor unidimensional.

Para todos os testes, os recortes foram normalizados para um tamanho médio de mosca, ou seja, a largura e altura média, resultando em recortes de 26 por 30 píxeis. Isso significa que, ao ser transformado em um vetor unidimensional, o recorte das imagens binarizadas e em preto e branco tem um tamanho de $780 \times 1$. Já as imagens coloridas no espaço RGB, como são matrizes tridimensionais, o vetor resultante tem um tamanho de $2340 \times 1$.

#### Imagem binarizada

Essa rede possui 780 neurônios na camada de entrada, e foram testados modelos com 150 e 300 neurônios na única camada escondida aplicando a função de ativação *logsig* e dois na camada de saída empregando a função *softmax*. O dado de entrada é a imagem binária, Figura 4.2 composta apenas por píxeis totalmente brancos ou pretos.

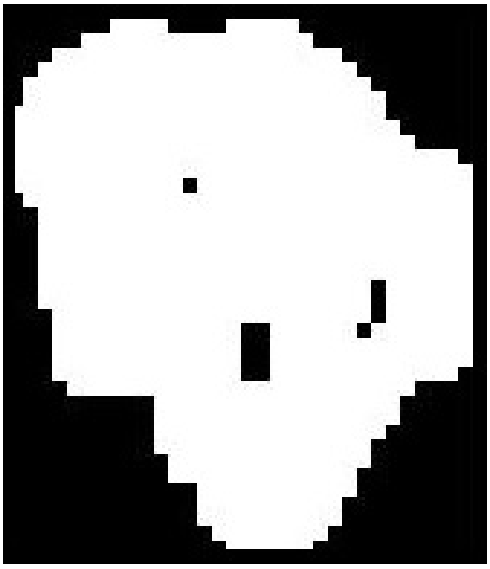


Figura 4.2: Exemplo de entrada da imagem binarizada.

Tabela 4.7: Acurácia do recorte binário

| **Número de neurônios** | **Acurácia** |
|---|---|
| 150 | 76,36% |
| 300 | 76,29% |

Percebemos pela Tabela 4.7, que esse tipo de entrada possui uma acurácia muito baixa. Também deve-se salientar que a acurácia dos dados de treino chegavam a 100%, indicando que houve especialização.

### Imagem em escala de cinza

Os parâmetros aplicados nessa rede são os mesmos da seção anterior. A única diferença foi que ao invés de um vetor composto por valores binários de 0's e 1's, como se trata de uma imagem em escala de cinza, presentada na Figura 4.3, cada elemento do vetor possui um valor inteiro entre 0 e 255.

Figura 4.3: Exemplo de entrada da imagem em tons de cinza.

Tabela 4.8: Acurácia do recorte em escala de cinza

| **Número de neurônios** | **Acurácia** |
|---|---|
| 150 | 77,76% |
| 300 | 77,13% |

Como podemos observar na Tabela 4.8, a acurácia foi semelhante a da Tabela 4.7.

### Imagem colorida

Nessa rede, o número de neurônios na camada de entrada é de 2340, além de ser testado uma variação com 450 neurônios na camada intermediária. Quanto a cor, foram usados os recortes da imagem normalizada, Figura 4.4.

Figura 4.4: Exemplo de entrada da imagem normalizada.

Tabela 4.9: Acurácia do recorte colorido

| **Número de neurônios** | **Acurácia** |
|---|---|
| 150 | 83,57% |
| 300 | 84,13% |
| 450 | 81,19% |

A acurácia mostrada na Tabela 4.9 foi superior as outras duas versões testadas, porém inferior aos outros modelo classificadores testados.

## 4.5 CNN

Nesse modelo, foram implementadas redes convolucionais rasas, ou seja, com no máximo 10 camadas de convolução, dado o tamanho pequeno do recorte. Vale ressaltar que, quando nos referciamos a camada de convolução, se trata de um bloco formado por uma convolução, seguida de uma normalização em lote (*Batch normalization*) e uma camada subsequente aplicando a função de ativação ReLU.

As redes CNN foram treinadas por 300 épocas, utilizando a função de entropia cruzada para cálculo do custo e ADAM como algorítimo de otimização. O tipo de redução de dimensionalidade (*pooling*) utilizado foi o *maxpooling* com passo (*stride*) igual a 2. Já na camada de convolução, o passo é igual a 1 e o preenchimento foi o *same padding*, o que implica dizer que a matriz de saída da camada possui a mesma dimensão da de

entrada. Em todas as redes implementadas nesse modelo, foram aplicados filtros (*kernels*) de tamanho $2 \times 2$, seja nas camadas de convolução ou de redução de dimensionalidade.

No bloco de saída, todas as redes possuem uma camada totalmente conectada com duas saídas seguida de um bloco de classificação aplicando a função *softmax*.

### 4.5.1 Definindo a normalização de tamanho

Quanto à normalização do tamanho do recorte que é usado como *input* da rede convolucional, são definidos dois tipos que chamaremos $Norm_1$ e $Norm_2$. Na $Norm_1$, os recortes são normalizados para o valor do tamanho médio das regiões classificadas como mosca, obtendo o tamanho de $26 \times 30$, observado na Figura 4.5a.

Já na $Norm_2$, a normalização é baseada na maior largura e altura possível de mosca, ou seja, se esses valores são os maiores possíveis, nenhum recorte de mosca tem deformações causadas pela normalização que ocorre usando, por exemplo, a média do tamanho. Para realizar essa normalização, criamos uma matriz de zeros e tamanho da maior largura e altura possíveis de mosca encontrada nos dados, no nosso caso 48 de largura por 53 de altura, e inserimos o recorte dentro dessa matriz de forma que o pixel do canto superior esquerdo do recorte sobreponha o pixel do canto superior esquerdo dessa nova matriz. O restante da matriz com valor 0, ou seja, as partes não cobertas pelo recorte sobreposto são setados para o valor de cor dado pela média entre os píxeis dos 4 cantos do recorte, que tendem a ter a cor da armadilha, como apresentado na Figura 4.5b. Os recortes que não são moscas, passam pelo mesmo processo, porém, caso algum tenha a largura maior que 48 ou altura superior a 53 píxeis, é feito somente a alteração do tamanho como em uma normalização comum.

(a) Recorte usando como parâmetro de normalização a média do recorte de mosca.

(b) Recorte usando como parâmetro de normalização a maior altura e largura de mosca possível.

Figura 4.5: Exemplo da diferença do recorte com os dois tipos de normalização do tamanho.

Portanto, quanto a normalização do tamanho, as imagens poderão variar de acordo com o tipo: $Norm_1 = 26 \times 30$ e $Norm_2 = 48 \times 53$.

Nas camadas de *pooling*, a partir de uma imagem de dimensões $W_1 \times H_1$, se o tamanho da dimensão do filtro for $F$ e o *stride* $S$, a saída será uma matriz de tamanho $W_2 \times H_2$ em que

$$W_2 = \frac{(W_1 - F)}{S + 1} \tag{4.3}$$

$$H_2 = \frac{(H_1 - F)}{S + 1} \tag{4.4}$$

que apenas poderá ter uma saída válida quando $W_1$ e $H_1$ forem maiores que $F$, ou seja, é isso que condiciona a aplicação de um *pooling*. De acordo com o tamanho do filtro que usamos, $F = 2$, e o *stride* que é $S = 2$, o número máximo de *poolings* aplicados em um recorte do tipo $Norm_1$ é 4, e do $Norm_2$ é 5.

Além da implementação de dois tipos diferentes de normalização do tamanho dos recortes, também foi utilizado dois tipos de padrão de cor diferentes: usando o recorte com a cor da foto original e da foto com a cor normalizada, descrita na Seção 3.1.3.

### 4.5.2 Imagem em escala de cinza

Nessa rede, usamos como entrada os recortes em escala de cinza normalizados pelo tipo $Norm_1$, que é o tamanho médio das moscas.

A primeira camada de convolução possui 8 neurônios, a segunda 16 e da terceira em diante todas possuem 32 neurônios, em que cada camada de convolução é seguida por uma camada de *pooling*.

Tabela 4.10: Acurácia da CNN do recorte em escala de cinza

| **Número de camadas de convolução** | **Acurácia** |
|---|---|
| 1 | 87,27% |
| 2 | 88,46% |
| 3 | 87,62% |
| 4 | 88,81% |
| 5 | 86,15% |

Como podemos observar na Tabela 4.10, com 4 camadas de convolução se obteve o melhor resultado para esse tipo de rede.

### 4.5.3 Recortes com a cor normalizada

Na Tabela 4.11 são mostrados os resultados da rede com uma estrutura em que cada camada de *pooling* fica posicionada entre duas camadas de convolução. A normalização usada foi do tipo $Norm_1$.

Tabela 4.11: Implementação usando recortes $26 \times 30$ e cor normalizada.

| **Número de camadas de convolução** | **Acurácia** |
|---|---|
| 1 | 87,34% |
| 2 | 88,25% |
| 3 | 89,23% |
| 4 | 90,28% |

Após esse limite, foram inseridas camadas extras de convolução no início da rede. Por exemplo, a estrutura da rede com 6 camadas de convolução possui duas camadas de convolução no início da rede antes da primeira camada de *pooling*, a rede com 7 camadas de convolução ao todo, possui 3 camadas de convolução antes da primeira camada de *pooling*, e assim sucessivamente. Observamos os resultados na Tabela 4.12.

Tabela 4.12: Implementação usando as convoluções extras no início da rede

| **Número de camadas de convolução** | **Acurácia** |
|---|---|
| 5 | 90,77% |
| 6 | 90,14% |
| 7 | 90,07% |
| 8 | 89,58% |

No modelo no qual os dados de entrada são os recortes normalizados de acordo com $Norm_2$, o número máximo de camadas de *pooling* é 5. Seguindo a mesma estrutura supracitada em que cada camada de *pooling* fica no meio de duas de convolução, obtemos os resultados representados na Tabela 4.13.

Tabela 4.13: Implementação usando recortes $48 \times 53$ e cor normalizada.

| **Número de camadas de convolução** | **Acurácia** |
|---|---|
| 1 | 89,30% |
| 2 | 90,07% |
| 3 | 90,21% |
| 4 | 92,73% |
| 5 | 93,57% |

Na Tabela 4.14 observamos o resultado inserindo camadas extras de convolução no início da rede, em que não houve um ganho de acurácia com a inserção das mesmas.

Tabela 4.14: Implementação usando as convoluções extras no início da rede.

| **Número de camadas de convolução** | **Acurácia** |
|---|---|
| 6 | 93,11% |
| 7 | 92,94% |
| 8 | 93,01% |
| 9 | 93,36% |
| 10 | 92,45% |
| 11 | 92,87% |

### 4.5.4 Recortes com a cor original

Nessa seção, mostraremos os resultados da implementação seguindo os mesmos parâmetros da anterior, porém foram usados os recortes das imagens originais, sem passar pelo processo de normalização da cor, como exemplificado na Figura 4.6.

Figura 4.6: Exemplo de entrada da imagem original.

A Tabela 4.15 mostra o resultado da CNN utilizando os recortes normalizados de acordo com $Norm_1$, em que cada camada de *pooling* fica entre duas camadas de convolução e a Tabela 4.16 com as convoluções extras no início da rede.

Tabela 4.15: Implementação usando recortes $26 \times 30$ e cor original.

| **Número de camadas de convolução** | **Acurácia** |
|---|---|
| 1 | 89,02% |
| 2 | 91,12% |
| 3 | 93,08% |
| 4 | 92,66% |

Tabela 4.16: Implementação usando as convoluções extras no início da rede.

| **Número de camadas de convolução** | **Acurácia** |
|---|---|
| 5 | 92,80% |
| 6 | 92,03% |
| 7 | 93,15% |
| 8 | 93,08% |

Os resultados do modelo em que os dados de entrada são os recortes normalizados de acordo com $Norm_2$ e o número máximo de camadas de *pooling* é 5 são mostrados na Tabela 4.17 seguido dos resultados com as camadas extras de convolução no começo da rede retratados na Tabela 4.18.

Tabela 4.17: Implementação usando recortes $48 \times 53$ de largura e a imagem original.

| **Número de camadas de convolução** | **Acurácia** |
|---|---|
| 1 | 90,98% |
| 2 | 90,91% |
| 3 | 91,96% |
| 4 | 94,34% |
| 5 | 95,55% |

Tabela 4.18: Implementação usando as convoluções extras no início da rede.

| **Número de camadas de convolução** | **Acurácia** |
|---|---|
| 6 | 95,52% |
| 7 | 94,76% |
| 8 | 94,97% |
| 9 | 94,76% |
| 10 | 94,13% |
| 11 | 92,87% |

Na Figura 4.7, observamos um gráfico comparativo entre os modelos aplicados nessa seção. De acordo com a legenda, os classificadores correspondem aos modelos:

1. Modelo utilizando inputs tamanho $Norm_2$ e cor original.
2. Modelo utilizando inputs tamanho $Norm_1$ e cor original.
3. Modelo utilizando inputs tamanho $Norm_2$ e cor normalizada.
4. Modelo utilizando inputs tamanho $Norm_1$ e cor normalizada.
5. Modelo utilizando inputs tamanho $Norm_1$ e cor em escala de cinza.

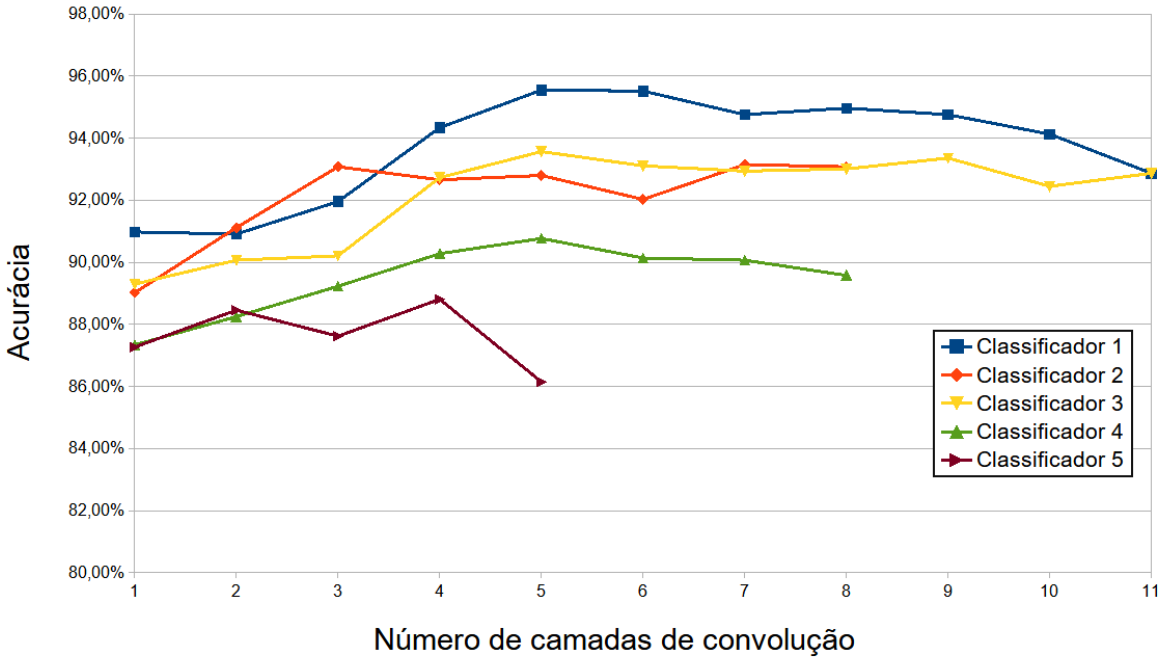



Figura 4.7: Comparação entre as CNN's.

Na Figura 4.7, o eixo $Y$ representa a acurácia e o $X$ o número de camadas de convolução em cada classificador. Os modelos que utilizaram a normalização do tamanho do tipo $Norm_2$ apresentaram um melhor desempenho, e também é mostrado que aumentar o número de camadas de convolução não faz com que o classificador CNN tenha uma melhor acurácia.

## 4.6 Análise dos resultados

Observa-se que os modelos em que os dados de entrada são as imagens originais obtiveram um desempenho melhor do que os que utilizaram a imagem normalizada. Sendo a normalização da cor importante para a segmentação, porém não tão eficaz para a classificação propriamente dita. Os modelos em que a entrada foi a imagem em escala de cinza ou binária não se mostraram eficientes.

A Tabela 4.19 mostra os melhores resultados obtidos com os classificadores KNN, MLP e CNN.

Tabela 4.19: Comparação das técnicas

| **Classificador** | **Melhor Acurácia** |
|---|---|
| KNN | 94,34% |
| MLP | 95,17% |
| CNN | 95,55% |

Observa-se que o melhor resultado foi obtido usando o modelo CNN, apesar da performance dos outros não terem ficado com uma precisão muito menor. Há de se salientar que o modelos utilizando a CNN são mais complexos que os outros, pelo fato do modelo da rede em si possuir um número maior de parâmetros e hiper-parâmetros.

Conclui-se que os modelos em que os dados de entrada são as imagens originais obtiveram melhor acurácia do que os que utilizaram a imagem normalizada, sendo a normalização da cor importante para a segmentação, porém os recortes normalizados não são tão eficazes para a classificação quanto os que possuem a cor original. Os modelos em que a entrada foi a imagem em escala de cinza não se mostraram eficientes.

# Capítulo 5

# Conclusão

Esse trabalho propôs uma metodologia de software para identificar moscas da espécie *Bemisia tabaci* em imagens de armadilhas adesivas amarelas entomológicas tradicionais, com o objetivo de acelerar o processo de contagem das mesmas, ação tipicamente feita de forma manual por um indivíduo apto.

Foram testados 3 modelos de classificadores: KNN, MLP e CNN, onde cada modelo foi testado usando dados de entrada variados, dados escalares oriundos do processo de segmentação e o próprio recorte dos segmentos, afim de testar qual seria o mais eficaz no processo de identificação da mosca-branca. Além dos diferentes tipos de dados de entrada, foram realizados testes usando variações dentro dos próprios modelos, alterando parâmetros dentro da rede, tais como o número de neurônios e número de camadas de convolução.

No melhor resultado obtido, conseguimos uma acurácia de 95,55%, apresentando um melhor resultado que em Cho *et al.* [12], com 94,56%(porém usando uma base de dados diferente) e um pouco menor do que em Nieuwenhuizen *et al.* [13], que utiliza a mesma base de dados, onde se chegou a obter uma precisão de 99% na classificação de moscas-brancas, porém usando uma rede CNN mais complexa.

## 5.1 Limitações

Afim de documentação das etapas do trabalho, salienta-se que foi feita a tentativa de implementação de modelos mais complexos de redes convolucionais, como uma YOLO [51] e uma *Faster R-CNN* [52] usando como dados de entrada a imagem original no tamanho natural ou fragmentos da mesma, porém o impedimento para a implementação desses modelos foi o poderio computacional disponível, dado a nossa limitação de hardware.

## 5.2 Trabalhos futuros

Uma proposição de trabalho futuro seria o uso de modelos de redes mais complexas e potentes, dado a disponibilidade de um maior poder computacional, para a resolução do problema da contagem da mosca-branca em armadilhas adesivas amarelas.

Além disso, os classificadores deste trabalho visavam a identificação de apenas uma espécie de inseto, seria interessante o desenvolvimento de modelos que classifiquem outras espécies, pragas ou não, que também são objetos de estudo e pesquisa ao redor do mundo.

# Referências


[1] Syngenta Digital. Como o clima favorece a multiplicação da mosca branca? https://pordentrodoagro.com.br/mosca-branca-clima-soja-algodao/. Online; Acesso: 12-11-2020. x, 1

[2] Ali Express. De doble cara, trampa adhesiva amarilla para insectos de plantas voladoras. https://es.aliexpress.com/i/4000121651443.html. Online; Acesso: 4-11-2020. x, 2

[3] Leandro Marques. Mecanismo de ação dos inseticidas neonicotinóides, organofosforados e carbamatos. https://elevagro.com/materiais-didaticos/mecanismo-de-acao-dos-inseticidas-neonicotinoides-organofosforados-e-carbamatos/. Online; Acesso: 10-3-2021. x, 6

[4] Akshay L Chandra. McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron. https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1. Online; Acesso: 4-11-2020. x, 8

[5] Frank La. How Do Neural Networks Learn? https://docs.microsoft.com/en-us/archive/msdn-magazine/2019/april/artificially-intelligent-how-do-neural-networks-learn. Online; Acesso: 1-10-2020. x, 10

[6] Frederik Kratzert. Understanding the backward pass through Batch Normalization Layer. https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html. Online; Acesso: 3-3-2021. x, 10

[7] Min Peng, Chongyang Wang, Tong Chen, e Guangyuan Liu. NIRFaceNet: A Convolutional Neural Network for Near-Infrared Face Identification. *Information*, 7(4), 2016. https://www.mdpi.com/2078-2489/7/4/61. x, 13

[8] Rob Robinson. Convolutional Neural Networks - Basics. https://mlnotebook.github.io/post/CNN1/. Online; Acesso: 20-05-2021. x, 14

[9] Shuihua Wang, Chaosheng Tang, Junding Sun, Jingyuan Yang, Chenxi Huang, Preetha Phillips, e Yu-Dong Zhang. Multiple Sclerosis Identification by 14-Layer Convolutional Neural Network With Batch Normalization, Dropout, and Stochastic Pooling. *Frontiers in Neuroscience*, 12:818, 11 2018. x, 14

[10] Suk-Ju Hong, Sang-Yeon Kim, Eungchan Kim, Chang Lee, Jung-Sup Lee, Dong-Soo Lee, Jiwoong Bang, e Ghiseok Kim. Moth Detection from Pheromone Trap Images Using Deep Learning Object Detectors. *Agriculture*, 10:170, 05 2020. x, 15, 16

[11] Jeremy Francis Tusubira, Solomon Nsumba, Flavia Ninsiima, Benjamin Akera, Guy Acellam, Joyce Nakatumba, Ernest Mwebaze, John Quinn, e Tonny Oyana. Improving In-field Cassava Whitefly Pest Surveillance with Machine Learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 68–69, 2020. x, 16, 17

[12] J. Cho, J. Choi, M. Qiao, C.W. Ji, H.Y. Kim, K.B. Uhm, e Tae-Soo Chon. Automatic Identification of Whiteflies, Aphids and Thrips in Greenhouse based on Image Analysis. *International Journal of Mathematics and Computers in Simulation*, 1:46–53, 01 2007. x, 18, 58

[13] Ard Nieuwenhuizen, J. Hemming, e H.K. Suh. Detection and classification of insects on stick-traps in a tomato crop using Faster R-CNN. *Research Gate*, 01 2018. `http://edepot.wur.nl/463457`. x, 18, 19, 58

[14] A.T. (Ard) Nieuwenhuizen, J. (Jochen) Hemming, D. (Dirk) Janssen, H.K. (Hyun) Suh, L. (Lien) Bosmans, V. (Vincent) Sluydts, N. (Nathalie) Brenard, E. (Estefanía) Rodríguez, e M.D.M. (Maria del Mar) Tellez. Raw data from Yellow Sticky Traps with insects for training of deep learning Convolutional Neural Network for object detection. `https://data.4tu.nl/articles/dataset/Raw_data_from_Yellow_Sticky_Traps_with_insects_for_training_of_deep_learning_Convolutional_Neural_Network_for_object_detection/12707066/1`, Mar 2019. x, 21, 22, 43

[15] Alexander Toet e Tirui Wu. Efficient contrast enhancement through log-power histogram modification. *Journal of Electronic Imaging*, 23:063017, 12 2014. xi, 29

[16] MathWorks. regionprops: Measure properties of image regions. `https://www.mathworks.com/help/images/ref/regionprops.html`. Online; Acesso: 1-10-2020. xi, 33, 34, 35

[17] AI-ML. K-nearest neighbor algorithm. `http://www.aimlfront.com/knn-algorithm.php`. Online; Acesso: 12-03-2020. xi, 39

[18] Vitor Rodrigues. Métricas de Avaliação: acurácia, precisão, recall... quais as diferenças? `https://medium.com/@vitorborbarodrigues/métricas-de-avaliaç~ao-acurácia-precis~ao-recall-quais-as-diferenças-c8f05e0a513c`. Online; Acesso: 12-03-2020. xi, 42

[19] Surapathrudu Kanakala e Murad Ghanim. Global Genetic Diversity and Geographical Distribution of Bemisia tabaci and its Bacterial Endosymbionts. *PLOS ONE*, 14(3):1–21, 03 2019. `https://doi.org/10.1371/journal.pone.0213946.1`

[20] Jesús Navas-Castillo, Elvira Fiallo-Olivé, e Sonia Sánchez-Campos. Emerging Virus Diseases Transmitted by Whiteflies. *Annual Review of Phytopathology*, 49(1):219–248, 2011. `https://doi.org/10.1146/annurev-phyto-072910-095235.1`

[21] Robert L. Gilbertson, Ozgur Batuman, Craig G. Webster, e Scott Adkins. Role of the Insect Supervectors Bemisia tabaci and Frankliniella occidentalis in the Emergence and Global Spread of Plant Viruses. *Annual Review of Virology*, 2(1):67–93, 2015. https://doi.org/10.1146/annurev-virology-031413-085410. 1

[22] C. M. Oliveira, A. M. Auad, S. M. Mendes, e M. R. Frizzas. Economic Impact of Exotic Insect Pests in Brazilian Agriculture. *Journal of Applied Entomology*, 137(1-2):1–15, 2013. https://onlinelibrary.wiley.com/doi/abs/10.1111/jen.12018. 1

[23] Delia Pinto-Zevallos e Irene Vänninen. Yellow sticky traps for decision-making in whitefly management: What has been achieved? *Crop Protection*, 47:74–84, 05 2013. 2

[24] Wei Pan, Zhanhuai Li, Yansong Zhang, e Chuliang Weng. The New Hardware Development Trend and the Challenges in Data Management and Analysis. *Data Science and Engineering*, 3:1–14, 09 2018. 2

[25] Andreas Schierwagen. *Vision as Computation, or: Does a Computer Vision System Really Assign Meaning to Images?*, pages 579–587. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. https://doi.org/10.1007/978-3-642-56585-4_37. 3

[26] François Chollet. *Deep Learning with Python*. Manning, November 2017. 5

[27] Nikita Silaparasetty. Machine Learning vs. Deep Learning. In *Machine Learning Concepts with Python and the Jupyter Notebook Environment: Using TensorFlow 2.0*, pages 57–65. Springer, 09 2020. https://doi.org/10.1007/978-1-4842-5967-2_4. 6

[28] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, e Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018. http://www.sciencedirect.com/science/article/pii/S2405844018332067. 7

[29] Frank Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6):386–408, 1958. http://dx.doi.org/10.1037/h0042519. 7

[30] Warren Mcculloch e Walter Pitts. A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943. 7

[31] Claude Sammut e Geoffrey I. Webb, editors. *Mean Squared Error*, pages 653–653. Springer US, Boston, MA, 2010. https://doi.org/10.1007/978-0-387-30164-8_528. 9

[32] Paul Munro. *Backpropagation*, pages 73–73. Springer US, Boston, MA, 2010. 11

[33] Hong Chen, Yi Tang, Luoqing Li, Yuan Yuan, Wei Liu, e Yuanyan Tang. Error Analysis of Stochastic Gradient Descent Ranking. *IEEE Transactions of Cybernetics*, 43:898–909, 06 2013. 11

[34] Diederik Kingma e Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014. 11

[35] Popescu Marius, Valentina Balas, Liliana Perescu-Popescu, e Nikos Mastorakis. Multilayer Perceptron and Neural Networks. *WSEAS Transactions on Circuits and Systems*, 8, 2009. 12

[36] C. C. Jay Kuo. Understanding convolutional neural networks with a mathematical model, 2016. 13

[37] Anand Koirala, Kerry Walsh, Zhenglin Wang, e Nicholas Anderson. Deep Learning for Mango (Mangifera Indica) Panicle Stage Classification. *Agronomy*, 12 2019. 15

[38] Joseph Redmon, Santosh Divvala, Ross Girshick, e Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016. 15

[39] O. Enrique Apolo, Jorge Martínez Guanter, Gregorio Egea, P. Raja, e Manuel Pérez-Ruiz. Deep learning techniques for estimation of the yield and size of citrus fruits using a UAV. *European Journal of Agronomy*, 115:126030, 04 2020. 15

[40] Agrocares. Scoutbox specifications. `https://www.agrocares.com/en/products/scoutbox/`. Online; Acesso: 3-3-2021. 18

[41] Serge Beucher e Fernand Meyer. *The Morphological Approach to Segmentation: The Watershed Transformation*, volume Vol. 34, page 433–481. CRC Press, 01 1993. 23, 26

[42] E. Davies. *Computer and Machine Vision: Theory, Algorithms, Practicalities*. Academic Press, 01 2012. 29

[43] MatWorks. stretchlim: Find limits to contrast stretch image. `https://www.mathworks.com/help/images/ref/stretchlim.html`. Online; Acesso: 1-10-2020. 30

[44] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417–441, 1933. 36

[45] Dongliang Cheng, Dilip Prasad, e Michael Brown. Illuminant estimation for color constancy: Why spatial-domain methods work and the role of the color distribution. *Journal of the Optical Society of America A*, 31, 04 2014. 36

[46] MathWorks. illumpca: Estimate illuminant using principal component analysis (PCA). `https://www.mathworks.com/help/images/ref/illumpca.html`. Online; Acesso: 1-10-2020. 36

[47] Trevor Hastie, Robert Tibshirani, e Jerome Friedman. The Elements Of Statistical Learning. *Aug, Springer*, 1, 01 2001. 38

[48] Chris Albon. Machine learning with python cookbook - chapter 4.- handling numerical data. `https://www.oreilly.com/library/view/machine-learning-with/9781491989371/ch04.html`. Online; Acesso: 12-11-2020. 38

[49] Padraig Cunningham e Sarah Jane Delany. k-Nearest Neighbour Classifiers. https://arxiv.org/abs/2004.04523, 2020. 39

[50] TzuTa Lin. LabelImg : a graphical image annotation tool and label object bounding boxes in images. https://github.com/tzutalin/labelImg. Online; Acesso: 3-4-2021. 43

[51] Joseph Redmon, Santosh Divvala, Ross Girshick, e Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 06 2015. 58

[52] Shaoqing Ren, Kaiming He, Ross Girshick, e Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39, 06 2015. 58