Universidade Estadual de Feira de Santana

Programa de Pós-Graduação em Computação Aplicada

Classificação do inseto *Diaphorina citri* utilizando *Deep Learning*

Mário Lúcio Gomes de Queiroz Pierre Júnior

Feira de Santana

2019

Universidade Estadual de Feira de Santana

Programa de Pós-Graduação em Computação Aplicada

Mário Lúcio Gomes de Queiroz Pierre Júnior

Dissertação apresentada à Universidade Estadual de Feira de Santana como parte dos requisitos para a obtenção do título de Mestre em Computação Aplicada.

Orientadora: Dr.ª Michele Fúlvia Angelo

Feira de Santana

2019

Ficha Catalográfica - Biblioteca Central Julieta Carteado - UEFS

P677

Pierre Júnior, Mário Lúcio Gomes de Queiroz

Classificação do inseto Diaphorina citri utilizando Deep Learning / Mário Lúcio Gomes de Queiroz Pierre Júnior. – 2019.

69 f.: il.

Orientadora: Michele Fúlvia Angelo.

Dissertação (mestrado) – Universidade Estadual de Feira de Santana, Programa de Pós-graduação em Computação Aplicada, Feira de Santana, 2019.

1. Aprendizado profundo. 2. *Deep Learning*. 3. Redes Convolucionais Neurais. 4. *Huanglongbin*. 5. *Greening*. 6. *Diaphorina citri*. 7. Classificação automatizada. I. Angelo, Michele Fúlvia, orient. II. Universidade Estadual de Feira de Santana. III. Título.

CDU: 004.65:632.75

Luis Ricardo Andrade da Silva - Bibliotecário - CRB-5/1790

Mário Lúcio Gomes de Queiroz Pierre Júnior

Classificação do inseto Diaphorina citri utilizando Deep Learning

Dissertação apresentada à Universidade Estadual de Feira de Santana como parte dos requisitos para a obtenção do título de Mestre em Computação Aplicada.

Feira de Santana, 03 de setembro de 2018

BANCA EXAMINADORA

Dra. Michele Fúlvia Angelo (Orientadora)
Universidade Estadual de Feira de Santana

Dr. Angelo Conrado Loula
Universidade Estadual de Feira de Santana

Dr. Antônio Lopes Apolinário Junior
Universidade Federal da Bahia

Abstract

Brazil is one of the main producers of orange juice, exporting 98% of its production. Phytosanitary problems cause loss of production and difficulty in exporting. Among existing pests is the industry's biggest concern that Huanglongbing, also known as Greening. This disease reduces the quality of the fruit, disrupts the development of the plant and reduces its productivity. The main spreader of the disease-causing bacteria is the Diaphorina citri, which is 2 to 3 mm in length. For the control of the problem, one method is to capture from yellow traps and the count of the insects that insects to later adjust the dosage of the insecticides to be applied. This way of monitoring is an important component in the prevention, the detection and counting of the insect that causes the disease is carried out manually and this process is determinant for more effective insecticide applications. This research had as general objective the use of the methodology of Deep Learning with Neural Convolutional Networks (CNN) in the classification of the insect Diaphorina citri in digitized images of adhesive traps, in order to streamline the identification process and improve the accuracy results in the recognition of the insect. To do this, it was necessary to create a database with the images of the traps after digitized and to analyze the results obtained in the classification of Diaphorina citri using models of distinct architectures with deep learning approach. For automated classification, three architectures of Neural Convolutional Networks (CNN) were tried. After evaluation and application of statistical tests to compare the results of the LeNet, AlexNet, Inception architectures, the Inception model applied to the set of test samples for generalization of the model presented an average accuracy of 99.51% accuracy in the cross-validation and 99.37% in the validation with the final test set in the classification of the insect Diaphorina citri.

Keywords: Deep Learning, Convolutioin Neural Network, Huanglongbing, Diaphorina citri.

Resumo

O Brasil é um dos principais produtores de suco de laranja, exportando 98% do que produz. E problemas fitossanitários causam perda na produção e dificuldade de exportação. Dentre as pragas existentes está a *Huanglongbing*, também conhecida por *Greening*. Esta doença reduz a qualidade do fruto, atrapalha o desenvolvimento da planta e reduz sua produtividade. O principal inseto propagador da bactéria causadora da doença é o psilídeo *Diaphorina citri*, que tem 2 a 3 mm de comprimento. Para o controle do problema, um método usado consiste na captura com armadilhas amarelas e contagem dos insetos para ajuste posterior da dosagem dos inseticidas a serem aplicados. Essa forma de monitoramento é um componente importante na prevenção, na detecção e contagem do inseto causador da doença, e é realizada de maneira manual sendo esse processo determinante para aplicações mais efetivas de inseticidas. Esta pesquisa, teve como objetivo geral o uso da metodologia de Aprendizado Profundo (*Deep Learning*) com Redes Convolucionais Neurais (CNN) na classificação do psilídeo *Diaphorina citri* em imagens digitalizadas de armadilhas adesivas, como forma de agilizar o processo de identificação e melhorar os resultados de precisão no reconhecimento do psilídeo. Para isso, foi necessário criar um banco de dados com as imagens das armadilhas após digitalizadas e analisar os resultados obtidos na classificação do *Diaphorina citri* utilizando modelos de arquiteturas distintos com abordagem de aprendizado profundo. Para a classificação automatizada, foram experimentadas três arquiteturas de Redes Convolucionais Neurais (CNN). Após avaliação e aplicação de testes estatísticos para comparar os resultados das arquiteturas *LeNet*, *AlexNet*, *Inception*, o modelo *Inception* aplicado ao conjunto de amostras de teste para generalização do modelo apresentou uma média de acurácia de 99.51% na validação cruzada e 99.37% na validação com o conjunto de teste final na classificação do inseto *Diaphorina citri*.

Palavras-chave: *Deep Learning*, Redes Convolucionais Neurais, *Huanglongbin*, *Diaphorina citri*.

Prefácio

Esta dissertação de mestrado foi submetida à Universidade Estadual de Feira de Santana (UEFS) como requisito parcial para a obtenção do grau de Mestre em Computação Aplicada.

A dissertação foi desenvolvida dentro do Programa de Pós-Graduação em Computação Aplicada (PGCA) tendo como orientadora a Prof.ª Dr.ª **Michele Fúlvia Angelo**.

Agradecimentos

Agradeço a Deus por todas as oportunidade que me são oferecidas, por ter saúde e pela possibilidade de agregar os conhecimentos de grande valor adquiridos durante esse período do curso.

Aos meu pais Mário e Sileide pelo grande amor ofertado, o incentivo à educação de qualidade e pelo grande esforço feito para nunca deixar que nada faltasse em nossa casa.

Às minhas irmãs Marcelle e Mayara pelo convívio sempre harmonioso e pelo carinho fraterno sempre presente em nossos encontros.

Ao meu sobrinho Martin que mesmo nos momentos mais difíceis, seus vídeos e fotos foram e são um sopro de alegria.

À minha esposa Hélia, grande incentivadora e responsável por grandes mudanças em minha vida, muito obrigado.

À professora Michele meus sinceros agradecimento por toda a dedicação e disponibilidade em ajudar e tornar possível a conclusão deste trabalho.

Aos professores do PGCA que contribuíram durante essa caminhada.

Ao Fundecitrus por disponibilizar as amostras utilizadas na criação do banco de imagens.

Sumário

Abstract i

Resumo ii

Prefácio iii

Agradecimentos iv

Sumário vi

Lista de Publicações vii

Lista de Tabelas ix

Lista de Figuras xi

Lista de Abreviações xii

Lista de Símbolos xiii

1 Introdução 1
- 1.1 Objetivo . . . 3
  - 1.1.1 Objetivos específicos . . . 3
- 1.2 Contribuições . . . 3
- 1.3 Organização do Trabalho . . . 3

2 Revisão bibliográfica 4
- 2.1 *Huanglongbing*(HLB) . . . 4
- 2.2 *Diaphorina citri* . . . 5
- 2.3 Armadilhas adesivas amarelas . . . 7
- 2.4 *Machine Learning* . . . 9
- 2.5 *Deep Learning* . . . 11
- 2.6 *Redes Neurais Convolucionais* . . . 12
  - 2.6.1 Camada Convolucional . . . 13
  - 2.6.2 Camada de *Pooling* . . . 15

2.6.3 Camada Completamente Conectada 19
2.7 Treinamento 19
2.7.1 Regularização 22
2.7.2 *Dropout* 22
2.8 *Dataset Augmentation* 24
2.9 *Arquiteturas de Redes Convolucionais* 24
2.9.1 *LeNet* 25
2.9.2 *AlexNet* 26
2.9.3 *Inception* 27
2.10 Avaliação dos Classificadores 31
2.10.1 Validação Cruzada 32
2.10.2 Testes estatísticos 33
2.11 Trabalhos Relacionados 35

3 Métodos e experimentos **39**
3.1 Criação do Banco de Imagem das Armadilhas Adesivas Amarelas . . 39
3.1.1 *Dataset Augmentation* 40
3.2 Classificação utilizando a abordagem proposta por Melo [Melo 2016] . 42
3.3 Classificação utilizando Redes Neurais Convolucionais 42
3.3.1 Arquitetura *LeNet* 45
3.3.2 Arquitetura AlexNet 46
3.3.3 Arquitetura *Inception* v3 47

4 Resultados e Discussões **49**
4.1 Experimentação utilizando a abordagem proposta por Melo [Melo 2016] 49
4.2 Experimentação da Arquitetura *LeNet* 51
4.3 Experimentação da Arquitetura *AlexNet* 53
4.4 Experimentação da Arquitetura *Inception* 56
4.4.1 Discussão dos Resultados 57

5 Considerações Finais **61**
5.1 Pesquisas Futuras 62

Referências Bibliográficas **63**

Lista de Publicações

[Pierre Junior et al. 2017] Pierre Junior, M. L. G. Q., Angelo, M. F., e Miranda, M. P. (2017). Identificação do Diaphorina Citri em Imagens de Armadilhas Adesivas Amarelas Digitalizadas. Anais da ERBASE - Escola de Computação Bahia-Alagoas-Sergipe.

Lista de Tabelas

2.1 Exemplo de uma arquitetura CNN. 18
2.2 Dados de exemplo para aplicação do teste de Wilcoxon para 10 domínios. 34

3.1 Base de imagens de insetos. 41
3.2 Parâmetros do modelo LeNet. 46
3.3 Parâmetros do modelo AlexNet. 47
3.4 Parâmetros do modelo *Inception* v3. 48

4.1 Resultado da classificação utilizando a metodologia proposta por Melo [Melo 2016] com validação cruzada 10-*fold*. 50
4.2 Resultado da classificação utilizando a metodologia proposta por Melo [Melo 2016] com validação cruzada 2-*fold* x 5. 50
4.3 Resultados da generalização da metodologia proposta por Melo [Melo 2016] utilizando SVM. 51
4.4 Resultados gerados após treinamento do modelo *LeNet*. 51
4.5 Resultados da generalização do modelo *LeNet* com amostras do conjunto reservado para ajuste de parâmetros, utilizando o otimizador *Adam*. 53
4.6 Resultados da generalização do modelo *LeNet* com amostras do conjunto reservado para ajuste de parâmetros, utilizando o otimizador *RMSprop*. 53
4.7 Resultados gerados após treinamento do modelo AlexNet. 54
4.8 Resultados da generalização do modelo com amostras do conjunto reservado para ajuste de parâmetros. 54
4.9 Resultados da generalização do modelo com amostras do conjunto reservado para ajuste de parâmetros. 55
4.10 Matriz de confusão da segunda execução de validação cruzada com o otimizador *Adam*. 55
4.11 Resultados gerados após treinamento do modelo *Inception*. 56
4.12 Resultados da generalização do modelo com amostras do conjunto reservado para ajuste de parâmetros. 56
4.13 Resultados da generalização do modelo com amostras do conjunto reservado para ajuste de parâmetros. 57
4.14 Matriz de confusão da primeira e quinta execução da validação cruzada com o otimizador *RMSprop*. 57

4.15 Resultado do teste de *Nemenyi* com os valores de acurácia. 58
4.16 Resultado do teste de *Nemenyi* com os valores de precisão. 59
4.17 Matriz de confusão da segunda execução de validação cruzada com o otimizador *Adam*. 60
4.18 Resultados da generalização do modelo com o conjunto de amostras de teste. 60

Lista de Figuras

2.1 Exemplo de folhas com perda da coloração verde e com nervuras mais grossas. 5
2.2 Exemplo de fruto incompletamente maduros e com sementes escurecidas. 6
2.3 Frutos desformes afetados pela doença. 6
2.4 Psilídeo vetor da HLB 6
2.5 Exemplares do psilídeo *Diaphorina citri* 7
2.6 Armadilha adesiva amarela utilizada na captura do psilídeo, cedida pelo FUNDECITRUS (Fundo de Defesa da Citricultura). 8
2.7 Armadilha adesiva na copa das árvores com o destaque para o inseto. 8
2.8 Machine Learning como campo da IA. 9
2.9 Visualização de uma camada CNN com a organização de seus neurônios em três dimensões (largura, altura, profundidade). 13
2.10 Convolução com *kernel* 3x3 e *kernel* 1x1. 14
2.11 Computando os valores de saída da convolução. 15
2.12 Computando os valores de saída da convolução com aplicação de *padding*. 16
2.13 Redução do volume espacial da imagem de entrada. 16
2.14 Aplicação de *maxpoling* com filtro de 2x2 e *stride* 2. 17
2.15 Exemplo de uma camada completamente conectada. 19
2.16 O cálculo de erro é usado como *feedback* pelo otimizador para o ajuste de peso da rede. 21
2.17 A) Rede sem aplicação de *dropout*. B) Exemplo de uma rede *thinned* produzia aplicando *dropout*. 23
2.18 Arquitetura modelo LeNet 25
2.19 Arquitetura do modelo AlexNet 26
2.20 Módulo *Inception* 27
2.21 Estrutura de rede *Inception v1* 28
2.22 Estrutura de rede *Inception v3* 28
2.23 Módulo A da *Inception v3* 29
2.24 Equivalência de uma convolução 3 x 1 seguida por uma convolução 1 3 com uma rede utilizando convolução 3 x 3 30
2.25 Figura a) representando o módulo B; e b) representando o módulo C da rede *inception* v3. 30

2.26 Exemplo de uso do processo de validação cruzada de K subconjuntos 33

3.1 a) Imagem original com parte de uma armadilha. b)A mesma imagem, porém com a marcação circular identificando o inseto *Diaphorina* na armadilha. 40
3.2 Imagens do *Diaphorina citri* com tamanhos distintos e com ruídos contidos nas armadilhas. 40
3.3 A esquerda, imagem original da extração de insetos da armadilha. A direita, imagens criadas pelo processo de aumento de base, a partir da imagem original. 41
3.4 70% das amostras para o conjunto de desenvolvimento e 30% para o conjunto de teste. 42
3.5 Etapas detalhas do processo de classificação da metodologia de [Melo 2016]. 43
3.6 Rotulagem das imagens. 44
3.7 Divisão do conjunto de desenvolvimento e aplicação do *k-Fold*. 44
3.8 Arquitetura do modelo LeNet. 45
3.9 Arquitetura do modelo AlexNet. 46

4.1 Gráfico do desvio padrão em função da acurácia com o otimizador *Adam*. 52
4.2 Gráfico do desvio padrão em função da acurácia com o otimizador *RMSprop*. 52
4.3 Gráfico do desvio padrão em função da acurácia com o otimizador *Adam*. 54
4.4 Gráfico do desvio padrão em função da acurácia com o otimizador *RMSprop*. 55

Lista de Abreviações

Abreviação	**Descrição**
CNN	Convolucional Neural Network
HLB	Huanglongbing
SURF	Speeded Up Robust Features
SIFT	Scale-Invariant Feature Transform
SVM	Support Vector Machine
IA	Inteligência Artificial
ANN	Artificial Neural Network
HOG	Histogram of Oriented Gradients
ReLU	Rectified Linear Unit
MLP	Multilayer perceptron
MNIST	Modified National Institute of Standards and Technology
RMSProp	Root Mean Square Propagation
RBF	Basis Function
BoF	Bag of Features
FUNDECITRUS	Fundo de Defesa da Citricultura
ILSVRC	ImageNet Large Scale Visual Recognition Competition

Lista de Símbolos

Símbolo	Descrição
δ	Delta

Capítulo 1

Introdução

Notadamente o Brasil é um país voltado ao agronegócio, e o setor tem importante participação no Produto Interno Bruto (PIB) brasileiro e na balança comercial. É o maior fornecedor mundial de frutas cítricas *in natura* e outros subprodutos [Neves et al. 2010].

As exportações de suco de laranja devem passar de 2,1 milhões de toneladas em 2015 para 2,4 milhões de toneladas ao final do período de 2024/2025. Isso representa um aumento de 16,9% na quantidade exportada [Brasil 2015]. O Brasil detém 50% da produção mundial de suco de laranja, exporta 98% do que produz e consegue 85% de participação no mercado mundial. Portanto, de cada cinco copos de suco de laranja consumidos no mundo, três são produzidos no Brasil [Neves et al. 2010].

No entanto, apesar de toda importância, os problemas fitossanitários causam grandes perdas econômicas aos produtores e tornam mais difíceis a exportação do suco de laranja brasileiro [Nava et al. 2007]. Pragas e doenças foram responsáveis pela erradicação de 40 milhões de árvores nesta década. A mortalidade saltou de 4% para preocupantes 7,5%. Essas doenças foram responsáveis por perdas de quase 80 milhões de caixas por ano. Uma das preocupações mais sérias do setor é o *Huanglongbing* (HLB), também conhecida como *Greening*, que avança com extrema rapidez [Neves et al. 2010].

O *Grenning* é a pior doença de citrus, e tem como características a clorose (entupimento dos vasos responsáveis por levar água e nutrientes da raiz para a copa da planta) e a redução e perda da qualidade do fruto deixando-o com sabor amargo [Halbert e Manjunath 2004].

O teor de acidez dos frutos aumenta e diminuem os sólidos solúveis, reduzindo a qualidade do suco. O sistema radicular também é afetado, apresentando pouco desenvolvimento [Bové 2006]. A queda de frutos sintomáticos é também observada em plantas infectadas pela doença. A proporção de frutos caídos aumenta conforme a progressão do *Greening* [Bassanezi et al. 2006].

Sem métodos de cura, o manejo da doença é realizado por meio do plantio de mudas sadias, inspeções para identificação e eliminação das plantas sintomáticas e o controle químico, que é a maneira mais utilizada pelo citricultor para reduzir a população do psilídeo [Yamamoto et al. 2009].

O principal inseto propagador da bactéria causadora da doença do HLB é o psilídeo *Diaphorina citri* [Fundecitrus 2016]. O *Diaphorina citri* é um pequeno inseto sugador que mede de 2 a 3mm de comprimento, de cor cinza e manchas escuras nas asas [Gallo et al. 2002]. Devem ser realizadas pulverizações periódicas para o controle do inseto vetor, *Diaphorina citri*, com o intuito de reduzir a probabilidade de ocorrência de novas infecções [Belasque Junior et al. 2009].

O uso incorreto de inseticida pode causar surto de pragas secundárias, tornar o psilídeo resistente e aumentar o custo da produção. No entanto, o emprego de defensivos agrícolas juntamente com as inspeções periódicas, aumentam a quantidade de plantas recuperadas e reduzem a necessidade de eliminar árvores produtoras [Belasque Junior et al. 2009].

A tomada de decisão para a pulverização é realizada por meio do monitoramento, que pode ser realizado com cartões adesivos ou por avaliação visual realizada por um profissional capacitado [Leonardo 2014]. Segundo Belasques (2009), para o emprego racional dos defensivos agrícolas, é necessário conhecer a população de psilídeos na lavoura, e um dos métodos utilizados é a inspeção visual. As armadilhas adesivas amarelas são utilizadas como forma de atração e captura de insetos e é um método de inspeção visual.

As armadilhas devem ser instaladas no perímetro da propriedade e colocada nas plantas para a captura efetiva dos *Diaphorina citri* [Miranda et al. 2011]. Um dos grandes problemas na utilização desse método de inspeção visual é que as armadilhas atraem outros insetos além do *Diaphorina citri* [Yamamoto et al. 2009]. Após dias de exposição no pomar, as armadilhas adesivas são retiradas e levadas aos laboratórios. Através da identificação e contagem manual dos psilídeos encontrados, realizada por técnicos treinados, são realizados cálculos para se estimar a população da praga no cultivo do citros [Leonardo 2014].

No trabalho de Melo [Melo 2016] foram experimentados métodos computacionais para a extração de características juntamente com algoritmos de aprendizado de máquina tendo como objetivo a identificação e classificação automatizada do *Diaphorina citri* em imagens de microscopia. A abordagem utilizou a combinação dos extratores SURF (*Speeded Up Robust Features*)/SIFT (*Scale-Invariant Feature Transform*), BoF (*Bag of Features*) com características do *Diaphorina citri* e SVM (*Support Vector Machine*). Portanto, com o algoritmo de aprendizado de máquina SVM foi alcançado um resultado de acurácia no processo de validação cruzada de 98,17% e teve 2,54% como desvio padrão e a acurácia do teste final de generalização de modelo foi de 99,14%, sendo superior ao medido na pesquisa de Leonardo [Leonardo 2014], que avaliou a eficiência no processo de contagem manual do inseto.

O monitoramento é um importante componente do manejo integrado de pragas, pois indica o momento correto para a tomada de decisão, evitando aplicações desnecessárias de inseticidas e, consequentemente o desequilíbrio biológico [Yamamoto et al. 2009]. Portanto, metodologias que automatizem esse processo de identificação e melhorem os resultados de acurácia são importantes para um processo eficiente de controle e uso racional de inseticidas.

1.1 Objetivo

O objetivo geral deste trabalho é medir e analisar o desempenho de classificação do inseto *Diaphorina citri* a partir de imagens de armadilhas adesivas amarelas digitalizadas, através do uso de modelos de Redes Neurais Convolucionais (*Convolutional Neural Network - CNN*).

1.1.1 Objetivos específicos

Os objetivos específicos desta pesquisa são:

- Criar um banco de imagens digitalizadas dos insetos *Diaphorina citri* contidos nas armadilhas adesivas amarelas;
- Analisar os resultados obtidos na classificação do *Diaphorina citri* utilizando a abordagem de aprendizado profundo;
- Comparar os resultados da classificação do *Diaphorina citri* através da abordagem de aprendizado profundo com a abordagem proposta por Melo (2016).

1.2 Contribuições

O uso da metodologia de aprendizado profundo, que permite a instrução de sistemas computacionais através de experiências e amostragem de dados, juntamente com a criação de um *dataset* de imagens, possibilitará a automatização do processo de identificação e classificação do *Diaphorina citri* com maior confiabilidade e rapidez.

Este trabalho torna-se válido, pois o método atual de reconhecimento do inseto demanda custo e tempo de avaliação para categorização dos insetos, que é realizada de maneira manual. A automatização computacional poderá dar mais confiabilidade nos resultados de classificação do psilídeo e proporcionará maior auxílio na tomada de decisão quanto ao uso eficiente de inseticidas.

1.3 Organização do Trabalho

No capítulo 2 é apresentada a revisão bibliográfica com os conceitos e trabalhos relacionados a esta pesquisa. No capitulo 3 é especificada a metodologia utilizada neste trabalho. No capítulo 4 são apresentados e discutidos os resultados obtidos. E finalmente no capítulo 5 é apresentada a conclusão do trabalho e as sugestões de futuros trabalhos.

Capítulo 2

Revisão bibliográfica

Este capítulo tem como objetivo apresentar os principais conceitos relacionados a este trabalho. Assim, a seguir será apresentada a doença dos citros *Huanglongbing* juntamente com seu inseto vetor *Diaphorina citri*; o artefato utilizado na captura dos psilídeos; técnicas computacionais que serão aplicadas ao logo deste trabalho para a classificação do inseto, bem como os trabalhos relacionados a esta pesquisa.

2.1 *Huanglongbing* (HLB)

O *Huanglongbing* ou *Greening* é uma destrutiva doença na cultura de citros que traz uma grande ameaça aos produtores mundiais, e que aos poucos está invadindo novas áreas da citricultura [Nava et al. 2007]. Presente de forma endêmica nos continentes asiático e africano há várias décadas, essa doença foi recentemente detectada nos dois principais países produtores de citros, Brasil e Estados Unidos [Teixeira et al. 2005][Hall 2009].

Antes da detecção do *Huanglongbing* no país, eram conhecidas duas variantes da bactéria causadora da enfermidade: *Candidatus Liberibacter africanus* e a *Candidatus Liberibacter asiaticus*. Porém, foi identificada a bactéria denominada *Candidatus Liberibacter americanus*, atualmente, somente encontrada no Brasil [Fundecitrus 2016][Bové 2006].

Em todos os lugares onde tem aparecido a doença, a produção de citros tem sido comprometida com a perda de milhões de árvores [Brlansky et al. 2012]. O HLB não provoca a morte das plantas, porém com o tempo ficam debilitadas e improdutivas. Observações de pomares afetados em diferentes regiões citrícolas do mundo revelam que pomares inteiros podem tornar-se inviáveis economicamente entre sete e dez anos após o aparecimento da primeira planta sintomática, se medidas de controle não são adotadas [Gottwald et al. 2007].

Uma vez detectada a presença de plantas sintomáticas, as mesmas devem ser eliminadas de forma compulsória e às expensas do proprietário [Hall 2009]. Todas as plantas básicas (obtidas a partir de processo de melhoramento), matrizes (plantas utilizadas na produção de mudas) ou de borbulheiras (viveiro de mudas) devem ser eliminadas [Brasil 2008].

Em chinês, *Huanglongbing* significa "doença do ramo amarelo". As folhas das árvores infectadas exibem neste ramo perda da coloração verde e apresentam-se parcialmente amareladas, sem delimitação entre as cores. Esse é o sintoma mais característico da doença, observado em todos os locais afetados. As folhas de ramos sintomáticos podem apresentar-se curvadas, de tamanho reduzido, com nervuras mais grossas e escurecidas [Belasque Junior et al. 2009]. Apresentam também folhas com manchas amareladas e deformações, sementes abortadas, árvores com poucos frutos e apresentando amargor. Porém, a semelhança desses sintomas com deficiência de nutrientes ou outras doenças, dificulta seu diagnóstico [Brlansky et al. 2012], Figura 2.1.

Causam a queda precoce de frutos, a diminuição dos frutos sintomáticos, assimétricos, incompletamente maduros e com a região estilar mantendo-se verde, diferentemente de frutos de ramos sadios. Apresentam-se mais ácidos e com menores valores de Brix, ratio, porcentagem de suco e sólidos solúveis, reduzindo a qualidade do suco [Bassanezi et al. 2006], conforme Figura 2.2 e Figura 2.3

Figura 2.1: Exemplo de folhas com perda da coloração verde e com nervuras mais grossas.
[Fundecitrus 2016]

2.2 *Diaphorina citri*

Considerado por muitos anos como sendo uma praga secundária no Brasil [Gallo et al. 2002], por causar danos diretos na cultura de citrus pela sucção de seiva e deformação foliar [Hall 2009], o *Diaphorina citri* tornou-se uma praga importante por propagar as bactérias transmissoras do HLB [Parra et al. 2010]. Este inseto é um pequeno sugador que mede de 2 a 3mm de comprimento, de cor cinza

Figura 2.2: Exemplo de fruto incompletamente maduros e com sementes escurecidas.
[Fundecitrus 2016]

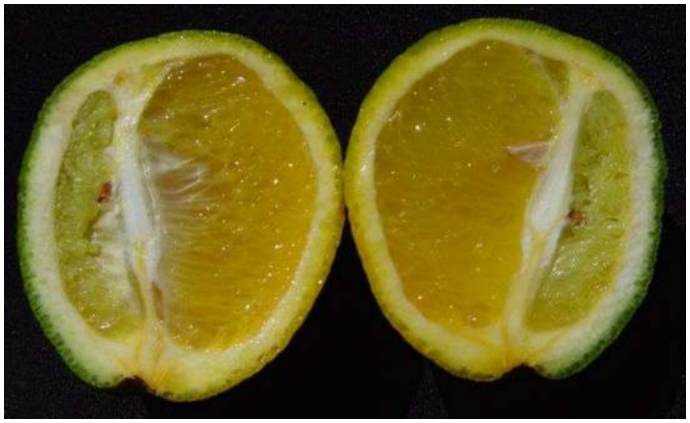

Figura 2.3: Frutos desformes afetados pela doença.
[Fundecitrus 2016]

e manchas escuras nas asas. Os ovos são colocados agrupados entre as folhas novas das plantas, possuem coloração amarela, formato alongado e medem 0.3mm de comprimento [Gallo et al. 2002].

O psilídeo é encontrado praticamente em todo o Brasil e ainda não existe cura para as plantas, pois não se conhece a resistência genética. Regulamentar a produção de mudas de citros, com maior fiscalização, controles severos [Brasil 2011] e ações direcionadas para a conscientização de produtores são de extrema importância na prevenção e controle da doença [Girardi et al. 2011].

O uso de inseticidas modernos pode dar resultados efetivos no controle da praga [Catling et al. 1970], porém, o uso incorreto pode causar surto de pragas secundárias, tornar o psilídeo resistente e aumentar o custo da produção. O emprego de defensivos agrícolas juntamente com as inspeções periódicas, aumentam a quantidade de plantas recuperadas e reduzem a necessidade de eliminar árvores produtoras. Portanto, o controle químico deve ser empregado de maneira racional e através do

Figura 2.4: Psilídeo vetor da HLB
[Fundecitrus 2016]

monitoramento da população do *Diaphorina citri*, por meio de armadilhas adesivas [Belasque Junior et al. 2009].

Figura 2.5: Exemplares do psilídeo *Diaphorina citri*
Fonte: http://www.fundecitrus.com.b

2.3 Armadilhas adesivas amarelas

Segundo Miranda [Miranda et al. 2011], o monitoramento do *Diaphorina citri* pode ser realizado por meio de:

- inspeção visual;
- amostragem por batidas no ramo, procedimento no qual o inspetor utiliza uma prancheta com papel branco quadriculado, que é posicionada abaixo do ramo, o qual é golpeado três vezes, contando-se posteriormente o número de psilídeos adultos sobre o papel branco;
- armadilhas adesivas dispostas na parte superior da copa das plantas.

Como estratégia de monitoramento do *Diaphorina citri*, utiliza-se armadilhas adesivas amarelas espalhadas no campo, Figura 2.2 [Girardi et al. 2011]. As armadilhas adesivas, como método de monitoramento do psilídeo, mostrou-se mais eficiente que outras estratégias de monitoramento, pois consegue capturar maior

quantidade de insetos adultos, do que as outras alternativas de monitoramento [Yamamoto et al. 2009]. É uma ferramenta importante para estimar a densidade do psilídeo na lavoura de citrus [Hall 2009]. O monitoramento através da utilização das armadilhas indica a atividade de *Diaphorina citri* dentro da propriedade, demonstrando quais são os locais de maior ocorrência e a época de migração de psilídeos provenientes de áreas vizinhas [Miranda et al. 2011], Figura 2.6.

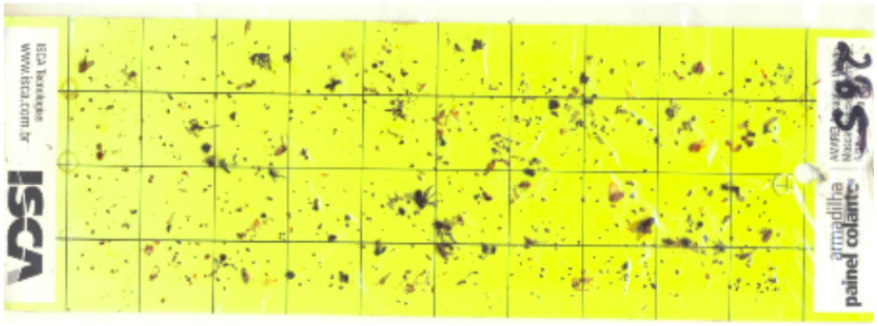

Figura 2.6: Armadilha adesiva amarela utilizada na captura do psilídeo, cedida pelo FUNDECITRUS (Fundo de Defesa da Citricultura).

Figura 2.7: Armadilha adesiva na copa das árvores com o destaque para o inseto. Fonte: [Fundecitrus 2016]

As armadilhas devem ser colocadas no terço superior da copa da árvore, de forma que fiquem bem visíveis ao inseto. A instalação deve ser feita, de preferência, nas plantas da borda do talhão e na bordadura da propriedade [Girardi et al. 2011], Figura 2.7. Após dias expostas, as armadilhas são coletas e levadas ao laboratório para ser realizada a identificação e contagem, de forma manual, dos insetos. Com a identificação e contagem, é possível calcular a população estimada do inseto na região e auxiliar na tomada de decisão referente a aplicação de defensivos agrícolas no combate ao psilideo [Leonardo 2014].

As inspeções para a identificação do inseto demandam custo e tempo dos inspetores para avaliação [Miranda et al. 2011], pois além da dificuldade de seu reconhecimento devido ao tamanho do psilídeo, as armadilhas adesivas atraem outros insetos [Gallo et al. 2002] [Yamamoto et al. 2009].

2.4 *Machine Learning*

O *Machine Learning* ou Aprendizado de Máquina é um campo dentro da Inteligência Artificial (IA), Figura 2.8, onde algoritmos são capazes de aprender através de exemplos [Géron 2017] e que tem como objetivo a construção e desenvolvimento de modelos computacionais que podem adaptar-se e aprender a partir de experiências [Mitchell et al. 1997]. Segundo Samuel [Samuel 1959], é o campo de estudo que dá aos computadores a capacidade de aprender sem ser explicitamente programado.

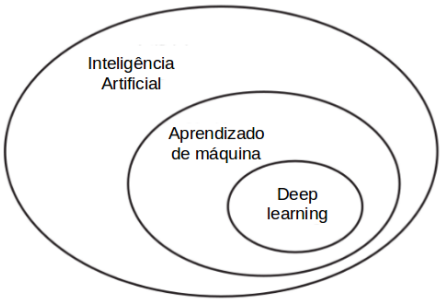

Figura 2.8: Machine Learning como campo da IA.
[Chollet 2017]

Segundo Mitchell (1997), um programa de computador aprende com a experiência E em relação a alguma tarefa T e alguma medida de desempenho P, se o seu desempenho em T, medido por P, melhora com a experiência E, sendo por exemplo:

- T: reconhecer e classificar palavras escritas à mão dentro de imagens;
- P: porcentagem de letras classificadas corretamente;
- E: uma base de dados de palavras manuscritas com classificações.

O aprendizado de máquina é indicado para problemas em que as soluções existentes exigem muitos ajustes manuais ou longas listas de regras: um algoritmo de aprendizado de máquina geralmente pode simplificar o código e ter um melhor desempenho. É também aplicado em problemas complexos para os quais não há uma boa solução usando uma abordagem tradicional, como por exemplo na área de reconhecimento de voz [Géron 2017].

Os sistemas de aprendizado de máquina podem ser classificados de acordo com o tipo de supervisão que recebem durante a etapa de treinamento. Podendo ser

divididos em: supervisionado; não-supervisionado; e por reforço [Géron 2017]. A forma mais comum de aprendizagem de máquina é a aprendizagem supervisionada [LeCun et al. 2015].

Na aprendizagem supervisionada, o algoritmo aprende através de dados de entrada rotulados e também com as informações de saída da rede. Assim, quando são fornecidas entradas de dados com saídas desconhecidas essa função de mapeamento é capaz de predizer as informações de saída [Russell e Norvig 2013]. Este tipo de aprendizagem faz com que o treinamento com os dados de entrada na *machine learning* tenha um alvo ou resultados esperados [Harrington 2012]. Uma tarefa típica de aprendizado supervisionado é a classificação. O filtro de *spam* é um bom exemplo disso: é treinado com muitos exemplos de *e-mails* junto com sua classe (*spam* ou não *spam*), e deve aprender como classificar os novos *e-mails* que serão recebidos [Géron 2017].

Dentre os algoritmos de aprendizado supervisionado estão: *Artificial Neural Network* (ANN); *Support Vector Machine* (SVM); *k-Nearest Neighbors* (KNN)[Kotsiantis et al. 2007]. Sendo a classificação uma das tarefas mais utilizadas pela aprendizagem supervisionada [Hackeling 2017].

Na aprendizagem supervisionada, o processo de aprendizagem ocorre sob a tutela de um "professor". No entanto, no paradigma conhecido como aprendizagem não supervisionada, não há supervisão durante o processo de aprendizagem. Isso quer dizer que não há dados de entrada rotulados para a função a ser aprendida pela rede. É o tipo de aprendizado de máquina no qual é dado um conjunto de exemplos de entrada, mas sem conhecimento do conjunto de saída, ou seja, não se conhece a classe que os atributos do conjunto de exemplos pertencem. A rede desenvolve a capacidade de formar representações internas para codificar recursos da entrada e, assim, criar novas classes automaticamente [Haykin et al. 2009].

Na aprendizagem por reforço, a aprendizagem se baseia em estratégias de séries de reforços que podem ser estımulos positivos ou negativos, ou seja, recompensas ou punições [Russell e Norvig 2013]. O sistema de aprendizagem, chamado de agente neste contexto, pode observar o ambiente, selecionar e executar ações e obter recompensas como retorno ou penalidades na forma de recompensas negativas. Neste tipo de aprendizagem o sistema deve aprender por si mesmo qual é a melhor estratégia para obter a maior recompensa ao longo do tempo [Géron 2017].

Para o processo de classificação utilizando o aprendizado supervisionado, é necessária a divisão do conjunto de dados em: conjunto destinado ao treinamento, sendo este o principal conjunto de informações fornecidas ao algoritmo para a construção do modelo e geração de hipóteses; e o conjunto de dados de teste que será utilizado para avaliar o modelo construído a partir do conjunto de treinamento. É importante que os dados usados para o treinamento não sejam os mesmo utilizados para a validação [Monard e Baranauskas 2003] e o conjunto de testes é completamente separado do processo de aprendizado [Sarkar et al. 2018].

É recomendada a divisão do conjunto de dados em 3 partes: um conjunto de treinamento; um conjunto de validação; e um conjunto de testes final. As amostras do conjunto de treinamento e do conjunto de validação são utilizadas para treinamento e para a validação do modelo. Após a seleção do modelo a ser utilizado, novamente são experimentados os conjuntos de treinamento e validação, e é apresentado o conjunto de teste, com dados não apresentados anteriormente, para avaliação da real generalização do modelo de aprendizagem [Hackeling 2017].

O desafio central no aprendizado de máquina é ter bom desempenho em novas entradas, antes inéditas e não apenas nas amostras em que nosso modelo foi treinado. E essa capacidade do modelo executar bem com o conjunto de dados de teste é chamada de generalização [Goodfellow et al. 2016]. A maneira de saber se o modelo de aprendizado irá generalizar para novos dados é experimentá-lo com novos dados e para isto será necessário colocar o modelo em produção e monitorar seu desempenho [Géron 2017].

A taxa de erro apresentada ao utilizar novas amostras é chamada de erro de generalização e, ao avaliar o modelo com o conjunto de testes, é possível obter uma estimativa desse erro. Esse valor informa o desempenho do modelo com dados que não foram utilizados durante o processo de treinamento do algoritmo. Se o erro de treinamento for baixo (seu modelo comete alguns erros no conjunto de treinamento), mas o erro de generalização com o conjunto de teste for alto, isso quer dizer que seu modelo está super ajustando (*overfitting*) aos dados de treinamento. O *overfitting* acontece quando o modelo tem bom desempenho durante o treinamento, mas expõe muitos erros de classificação ao ser apresentado a um novo conjunto de dados, apresentando um intervalo muito grande entre o erro de treinamento e o erro de teste. Enquanto que o *underfitting* ocorre quando o modelo não consegue obter um valor de erro suficientemente baixo no conjunto de treinamento [Géron 2017] [Goodfellow et al. 2016].

2.5 *Deep Learning*

Deep Learning ou Aprendizagem Profunda é uma abordagem de *Machine Learning* que permite que sistemas computacionais aprendam através de experiência e amostras de dados [Goodfellow et al. 2016]. Esse método de aprendizado permite que modelos computacionais compostos de múltiplas camadas de processamento aprendam representações de dados com múltiplos níveis de abstração. Esses métodos melhoraram drasticamente o estado da arte no reconhecimento de voz e reconhecimento visual de objetos [LeCun et al. 2015].

Na Aprendizagem Profunda, o algoritmo é alimentado com dados brutos que descobrem automaticamente representações, com múltiplos níveis de representação, obtidos pela composição de módulos simples, mas não lineares, que transformam a representação em um nível (começando com a entrada bruta) em outro nível mais

alto e ligeiramente mais abstrato [Goodfellow et al. 2016]. Para que as representações sejam aprendidas, é necessária uma sequência profunda de estágios na transformação destes dados, sendo fundamental a apresentação de amostras de exemplo [Chollet 2017].

A transformação implementada por uma camada é parametrizada por seus pesos, desta forma, o aprendizado significa encontrar valores para os pesos de todas as camadas, de maneira que a rede mapeie corretamente a entrada de exemplos para seus destinos associados [Chollet 2017]. A *deep learning* descobre uma estrutura complexa em grandes conjuntos de dados usando o algoritmo de *Backpropagation* para indicar como uma máquina deve alterar seus pesos [LeCun et al. 2015].

Para tarefas de classificação, as camadas mais altas de representação amplificam aspectos da entrada que são importantes para a discriminação e suprimem variações irrelevantes. Uma imagem, por exemplo, vem na forma de uma matriz de valores de pixel, e os recursos aprendidos na primeira camada de representação normalmente representam a presença ou ausência de bordas em orientações e locais específicos na imagem. Na segunda camada da rede, normalmente se detecta arranjos particulares de bordas, independentemente de pequenas variações de posições. A terceira camada pode agrupar combinações que correspondem a partes de objetos, e as camadas subsequentes detectariam objetos como combinações dessas partes. O aspecto chave da aprendizagem profunda é que ela aprende através de amostras de dados usando um procedimento de propósito geral para o aprendizado [LeCun et al. 2015].

Entre as várias arquiteturas de aprendizagem profunda, as *Convolucional Neural Network* (CNN) vêm apresentando grandes avanços no reconhecimento de fala e reconhecimento de objetos [LeCun et al. 2015].

2.6 *Redes Neurais Convolucionais*

As Redes Neurais Convolucionais (*Convolutional Neural Network* - CNN) estão no centro dos avanços na aprendizagem profunda [Dumoulin e Visin 2016]. São um tipo especializado de rede Perceptron Multicamadas (MLP) [Goodfellow et al. 2016] que ganharam muito impulso nos últimos anos, especialmente no campo do reconhecimento de imagens [Joshi 2017].

O nome Rede Neural Convolucional indica que uma Rede Neural Artificial (RNA) emprega uma operação matemática chamada convolução, que é um tipo especializado de operação linear. As redes convolucionais são simplesmente redes neurais que usam a convolução no lugar da multiplicação geral de matrizes em pelo menos uma de suas camadas [Goodfellow et al. 2016].

Uma das características dessa rede são em relação as propriedades estatísticas de uma imagem que são invariantes à translação (deslocamento). Por exemplo, uma

foto de um gato permanece uma foto de um gato mesmo que um *píxel* seja deslocado para a direita da imagem. As CNNs levam em conta essa propriedade compartilhando parâmetros em vários locais da imagem com o mesmo atributo sendo calculado em diferentes locais na entrada. Isso significa que se pode encontrar um gato com o mesmo detector, quer o gato apareça na coluna i ou coluna $i + 1$ na imagem [Goodfellow et al. 2016].

As CNNs foram utilizadas nos anos noventa para resolver as tarefas de reconhecimento de caracteres e no início dos anos 2000 foram aplicadas com grande sucesso na área de visão computacional com a detecção, segmentação e reconhecimento de objetos e regiões em imagens [LeCun et al. 2015].

Atualmente, é bastante difundida devido a um trabalho mais recente, quando uma CNN profunda foi usada para vencer o desafio de classificação de imagem *ImageNet* [Krizhevsky et al. 2012]. Desta maneira, provando a eficácia de sua aplicação no reconhecimento de imagens e também alcançando grandes resultados no reconhecimento de fala e reconhecimento facial [LeCun et al. 2015] [Deng et al. 2014].

A arquitetura típica de uma Rede Neural Convolucional é estruturada em uma série de estágios. O primeiro estágio é composto por duas camadas responsáveis pelo processo de extração de características: Camadas Convolucionais, que processa as entradas considerando campos receptivos locais; a Camada de *Pooling*, que reduz a dimensionalidade espacial das representações; e o segundo estágio sendo composto de uma Camada Totalmente Conectada *(Fully Connected)*, que tem por função atuar como um classificador. A saída da CNN define a probabilidade da imagem pertencer a uma das classes para qual a rede foi treinada [LeCun et al. 2015].

As redes convolucionais são projetadas para processar dados tridimensionais. Desta forma, suas camadas possuem neurônios dispostos em 3 dimensões: largura, altura e profundidade [Karpathy 2018], conforme Figura 2.9. Portanto, são projetadas para processar dados que vêm na forma de múltiplos *arrays*, por exemplo, uma imagem colorida composta por três matrizes contendo intensidades de pixel nos três canais de cores [LeCun et al. 2015]. As dimensões das imagens mudam conforme passam pelas camadas da rede convolucional até gerar um único vetor com uma série de probabilidades na camada de saída, sendo uma probabilidade para cada possível classe de saída [Karpathy 2018].

2.6.1 Camada Convolucional

Convolução é uma operação matemática entre duas funções f e g, produzindo uma terceira função, que pode ser interpretada como uma função modificada de f. No processamento de imagens, onde esta é determinada como uma função bidimensional, a convolução é útil para detecção de bordas, extração de atributos e outras aplicações [Parker 2010].

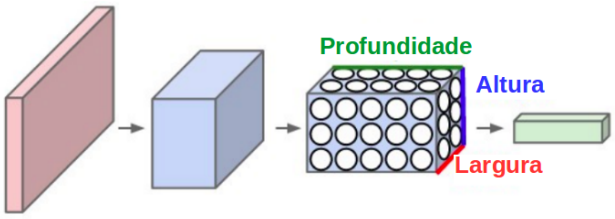

Figura 2.9: Visualização de uma camada CNN com a organização de seus neurônios em três dimensões (largura, altura, profundidade).
Fonte: http://cs231n.github.io/convolutional-networks/pool

Na camada convolucional um conjunto de filtros (*kernels*) são aplicados em uma imagem e percorrem sequencialmente os dados desta entrada, sendo cada região da imagem processada pelo filtro chamado de campo receptivo *(receptive field)*. Cada filtro é aplicado aos valores de *pixel* da imagem, levando em consideração os canais de cores, por uma janela deslizante que são matrizes de pesos e calculam o produto do ponto entre o *pixel* do filtro e o *pixel* de entrada. Isso resultará em um mapa de ativação bidimensional do filtro chamado mapa de atributos (*features map*). Durante este processo, os filtros são ajustados automaticamente para que sejam ativados ao detectarem características relevantes como arestas, curvas e bordas [Skansi 2018].

Os valores resultantes após a operação de convolução passam por uma função de ativação, e a mais comum é a função ReLU *(Rectified Linear Units)* para introduzir a não linearidade em uma CNN [Krizhevsky et al. 2012] [Dumoulin e Visin 2016]. A ReLU é uma função de ativação muito popular definida como $f(x) = max\ (0,\ x)$ onde x é a entrada para um neurônio. É destinada a zerar valores negativos, que é zero quando $x < 0$ e depois linear com declive 1 quando $x > 0$ [Chollet 2016].

Cada camada convolucional pode ter vários filtros aplicados ao dado de entrada e este processo por ser repetido gerando quantos mapas de atributos forem necessários [Dumoulin e Visin 2016]. Os mapas gerados pelos diversos filtros são empilhados formando um tensor cuja profundidade é igual ao número de filtros aplicados. Esse tensor será oferecido como entrada para a próxima camada da rede [Ponti e da Costa 2018].

A aplicação dos filtros é realizada por uma janela deslizante que percorre no sentido horizontal toda a dimensão de uma imagem e pode ser especificado o passo *(*stride) com o qual o filtro é deslizado. Quando o passo é definido como 1, os filtros percorrem um pixel por vez. Quando o *stride* é 2, os filtros saltam 2 *pixels* de uma vez enquanto percorre a dimensão da imagem. Quanto maior for o passo, menor será o volume de saída da camada [Karpathy 2016].

A Figura 2.10 exemplifica o processo de aplicação de filtros em uma imagem. Os quadrados na cor azul com as dimensões 7 x 7 representam a imagem de entrada na camada convolucional. Dois tamanhos de filtros são utilizados neste exemplo, um filtro com tamanho 3 x 3 e outro filtro com tamanho 1 x 1, com os dois utilizando

o valor de *stride* 1 resultando em um janela deslizante percorrendo 1 *pixel* por vez em toda dimensão da imagem. Os quadrados na cor verde representam o mapa de atributo gerado pelo processo de convolução. Um mapa de atributo menor com 5 x 5 de dimensão, devido o uso de filtro 3 x 3, e outro mapa de atributo com a dimensão igual ao tamanho da imagem por ter utilizado um filtro 1 x 1.

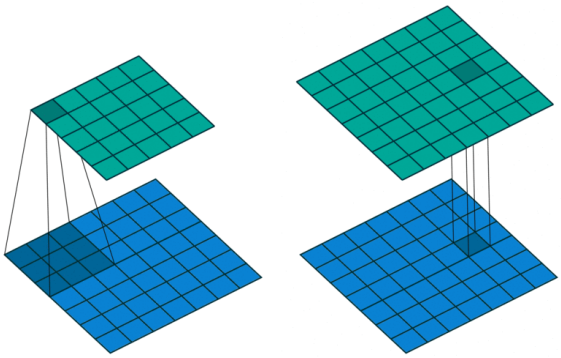

Figura 2.10: Convolução com *kernel* 3x3 e *kernel* 1x1.
Fonte: https://iamaaditya.github.io/2016/03/one-by-one-convolution

Um exemplo da aplicação da camada convolucional com valores discretos é apresentado na Figura 2.11. O quadrado na cor azul com dimensão 5 x 5 representa a imagem tendo ao centro o valor de cada *pixel*, enquanto que o quadrado sobreposto na cor azul escuro representa o tamanho do filtro na dimensão 3 x 3 com seus valores apresentados no canto inferior de cada *pixel*. Na cor verde com dimensão 3 x 3 está representado o mapa de atributo gerado. Ao passo de 1 *pixel* por vez da janela deslizante são calculados os produtos entre os valores do campo receptivo da imagem com os valores do filtro e ao final é realizada a soma desses valores. Nesse exemplo o resultado desse primeiro cálculo resultou no valor 12, apresentado no primeiro quadrante na cor verde escuro do mapa de atributos. Caso o resultado da soma seja negativo, a função ReLU zera esse valor. Esse processo apresentado é repetido a cada passo da janela deslizante.

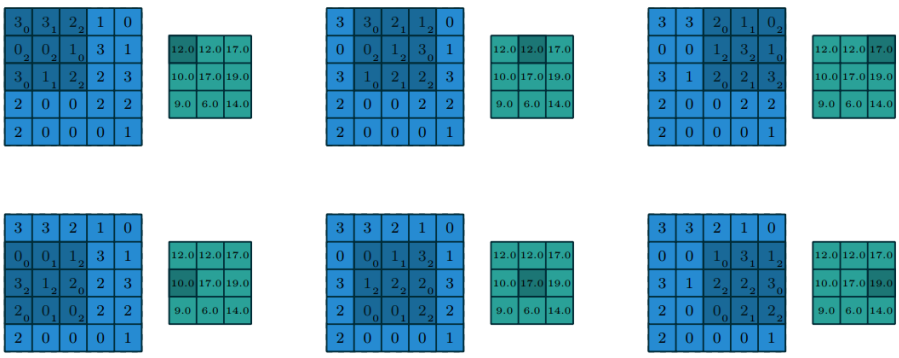

Figura 2.11: Computando os valores de saída da convolução.
Fonte: [Dumoulin e Visin 2016].

Existem duas formas de tratar as bordas das imagens durante esse processo de aplicação de filtros. Podem ser utilizados o *valid padding* ou *zero padding*. Com o *valid padding* as bordas da imagem não são ultrapassadas pela borda do filtro, enquanto que no *zero padding* ocorre o preenchimento com zero todas as vezes que o filtro ultrapassar as bordas da imagem de modo a ter a saída com o mesmo tamanho e largura da imagem original. A Figura 2.12 demonstra a utilização de *zero padding* na geração dos mapas de atributos.

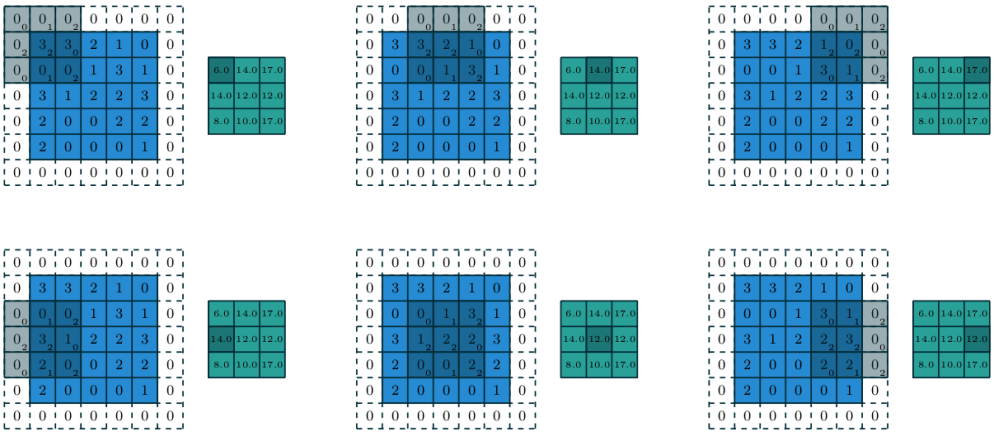

Figura 2.12: Computando os valores de saída da convolução com aplicação de *padding*.
Fonte: [Dumoulin e Visin 2016].

2.6.2 Camada de *Pooling*

Uma camada típica de uma rede convolucional consiste em três etapas. No primeiro estágio, a camada executa várias convoluções em paralelo para produzir um conjunto de ativações lineares. No segundo estágio, cada ativação linear é executada através de uma função de ativação não linear, como a função ReLU e este estágio é às vezes chamado de estágio do detector. No terceiro estágio, é usada uma função de agrupamento (*pooling*) para modificar ainda mais a saída da camada [Goodfellow et al. 2016].

As camadas de *pooling* são inseridas entre as camadas de convolução e seu objetivo é reduzir progressivamente a dimensão espacial do volume de entrada (*downsampling*), e consequentemente reduzir o custo computacional da rede e evitar *overfitting* [Karpathy 2018]. A Figura 2.13 mostra a redução da dimensionalidade da imagem em metade de seu tamanho original.

Após cada camada de convolução é aplicada uma camada de subamostragem (*subsampling*), que nada mais é que uma coleta de amostras de cada mapa de característica. Estas amostragens podem ser realizadas obtendo-se a soma, tirando-se a média,

selecionando-se o maior (*maxpooling*) ou menor (*minpooling*) valor da região em análise, o que produz uma sumarização do mapa de características [Haykin et al. 2009].

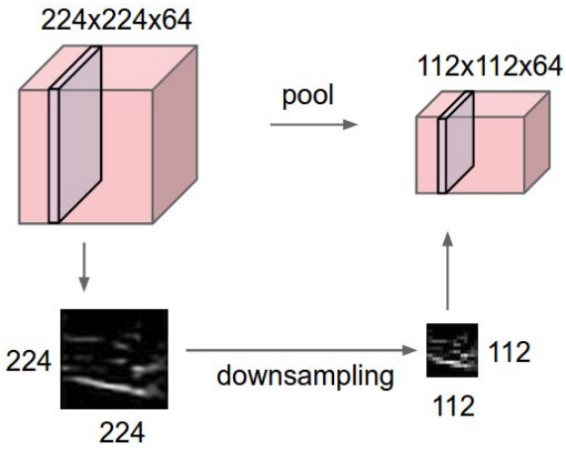

Figura 2.13: Redução do volume espacial da imagem de entrada.
Fonte: http://cs231n.github.io/convolutional-networks/pool

Essa redução é importante por uma questão de agilidade no treinamento, mas principalmente para criar invariância espacial. A camada de *pooling* funciona agrupando um conjunto de dados, por exemplo: a entrada é dividida em janelas ou *pool size* de 2 x 2 com *stride* com o valor 2. Os *strides* são os passos que uma janela deslizante percorre a imagem verticalmente, neste caso 2 *pixels* por vez e de cada uma é selecionado um valor para os representar. Essa escolha pode ser feita por diversas funções, porém a mais utilizada é a função de *maxpooling*, que consiste em dividir a entrada e produzindo o valor máximo de cada *patch*. A ideia da função *maxpooling* é que informações importantes em uma imagem raramente estão contidas em *pixels* adjacentes, e é frequentemente contida em *pixels* mais escuros [Skansi 2018]. É importante mencionar que a camada de *pooling* não reduz a profundidade da entrada, ela apenas reduz a altura e a largura de um mapa [Vargas et al. 2016][Dumoulin e Visin 2016]. Na Figura 2.14 é aplicada a função *maxpooling* e no primeiro quadrante na cor rosa é selecionado o maior valor. O processo é repetido a cada passo da janela deslizante e resulta na redução da dimensão da imagem em 50% de seu tamanho original contendo somente os *pixels* mais significativos.

A camada de *pooling* ajuda a tornar a representação aproximadamente invariante e aumenta a cobertura de características. E a cada camada convolucional ou de *pooling*, a quantidade de mapas de características é incrementada enquanto que a resolução espacial é diminuída, comparando com as outras camadas anteriores correspondentes [Haykin et al. 2009].

A Tabela 2.1 representa um exemplo de arquitetura de uma CNN composta de 3 camadas convolucionais e 2 camadas de *pooling* com aplicação em classificação de imagem. Foi utilizado como exemplificação uma imagem com as dimensões de (28, 28, 3), sendo os valores de 28 *pixels* x 28 *pixels* a altura e largura da imagem e o

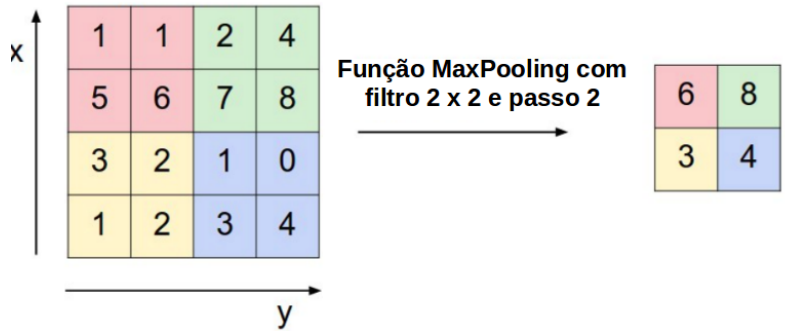

Figura 2.14: Aplicação de *maxpoling* com filtro de 2x2 e *stride* 2.
Fonte: http://cs231n.github.io/convolutional-networks/pool

valor 3 representando os canais RGB. Os parâmetros e valores resultantes de cada camada são apresentados da seguinte maneira: :

- Na primeira camada convolucional (Conv 1) da rede são aplicados 32 filtros com tamanho de *kernel* de dimensão 3 x 3, que após processo de convolução tem como saída o tensor de dimensão (26, 26, 32), que corresponde a 32 mapas de características com dimensão de 26 x 26.
- A camada de *Pooling* 1 com aplicação da função *maxpolling* e filtro de dimensão 2 x 2, produz como saída um tensor (13, 13, 32).
- Na segunda camada convolucional (Conv 2) de rede são aplicados 64 filtros com tamanho de *kernel* de dimensão 3 x 3, que após processo de convolução tem como saída o tensor de dimensão (11, 11, 64), que corresponde a 64 mapas de características com dimensão de 11 x 11.
- A camada de *Pooling* 2 com aplicação da função *maxpolling* e filtro de dimensão 2 x 2, produz como saída um tensor (5, 5, 64).
- Na terceira camada convolucional (Conv 3) de rede são aplicados 64 filtros com tamanho de *kernel* de dimensão 3 x 3, que após processo de convolução tem como saída o tensor de dimensão (3, 3, 64), que corresponde a 64 mapas de características com dimensão de 3 x 3.

Tabela 2.1: Exemplo de uma arquitetura CNN.

Camada da Rede	**Saída da camada**	**Parametros**
Conv1	(26, 26, 32)	320
Pooling 1	(13, 13, 32)	0
Conv2	(11, 11, 64)	18496
Pooling 2	(5, 5, 64)	0
Conv3	(3, 3, 64)	36928

Os valores de parâmetros também apresentados na Tabela 2.1 referem-se a quantidade de parâmetros a serem aprendidos em cada camada e que esses valores au-

mentam conforme o tamanho da rede. Esses parâmetros são calculados utilizando a quantidade de filtros da camada juntamente com as dimensões deste filtro, a quantidade de canais da imagem e o valor de *bias*, associado com cada valor do vetor recebido da camada anterior [Ponti e da Costa 2018].

$$Conv1 - (32 * (3 * 3 * 3 + 1) = 320$$

$$Conv2 - (64 * (3 * 3 * 32 + 1) = 18.496$$

$$Conv3 - (64 * (3 * 3 * 64 + 1) = 36.928$$

O envio dos dados das camadas de extração de características para a camada de classificação é feito através da camada *flatten*. Sua atribuição é passar uma matriz de características 3D para um vetor de características e desta forma, alimentar a camada completamente conectada. É realizado o cálculo do produto dos valores do tensor de saída da última camada $Conv3 - (3 * 3 * 64) = 576$, resultado em um vetor de característica de tamanho 576.

2.6.3 Camada Completamente Conectada

Dentro da etapa de classificação são usadas camadas completamente conectadas, onde cada neurônio fornece uma conexão completa com todos os mapas de atributos aprendidos emitidos a partir das camadas de convolução anteriores [Yosinski et al. 2014].

A entrada de dados para a primeira camada totalmente conectada é o conjunto de todos os mapas de atributos resultantes da etapa de extração de atributos e são passados para as camadas ocultas [Karpathy 2018]. As camadas ocultas são compostas de neurônios artificiais onde será aplicada uma função de ativação não linear que irá propagar ou não as informações com base nos estímulos recebidos da camada anterior multiplicado pelos pesos obtidos durante a etapa de treinamento [Haykin et al. 2009].

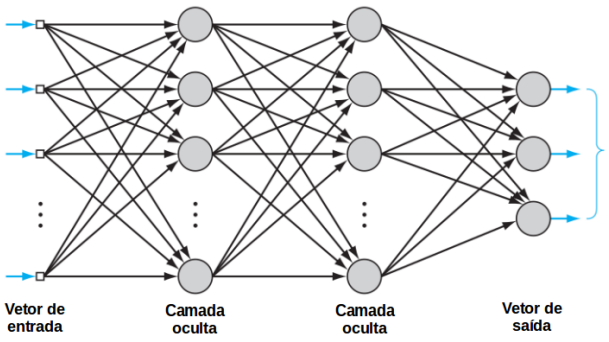

Figura 2.15: Exemplo de uma camada completamente conectada. [Haykin et al. 2009]

A Figura 2.15 exemplifica a etapa de classificação de uma CNN com: um vetor com as características detectadas nas camadas anteriores como entrada de dados na camada totalmente conectada; 2 camadas ocultas de neurônios responsáveis pelo treinamento feito com os ajustes de parâmetros e *bias* conforme o valor de custo; e uma camada de saída que irá representar uma distribuição de probabilidades dos resultados de classificação por meio de uma função de regressão logística.

Neurônios em uma camada completamente conectada têm conexões completas com todas as ativações na camada anterior, como nas redes neurais regulares. Suas ativações podem, portanto, serem calculadas com uma multiplicação de matrizes seguida por um deslocamento de polarização. A camada completamente conectada corresponde a uma rede MLP (*Multilayer Perceptron*) tradicional [Karpathy 2016].

2.7 Treinamento

As MLPs são frequentemente aplicadas a problemas de aprendizagem supervisionada. O treinamento desta rede envolve ajustes de parâmetros e *bias* com base na avaliação do valor de custo. O valor de custo é calculado pela função de custo que compara os rótulos das previsões da rede com os rótulos das classes verdadeiras e calcula uma pontuação de custo [Chollet 2017]. Esse cálculo de custo é usado como um *feedback* para ajuste nos valores de peso, com objetivo de redução deste valor [Haykin et al. 2009].

O ajuste de peso é uma atribuição da função de otimização que implementa a retropropagação (*backpropagation*) [Chollet 2017]. O processo de *backpropagation* consiste na propagação dos dados, onde a transmissão dos dados é feita para frente em um único sentido com os neurônios recebendo as informações da camada anterior e enviando para a próxima camada, e a retropropagação permite que as informações ao final da propagação retornem alimentando a entrada da rede atualizando os pesos com objetivo de minimização do valor de custo do modelo [Goodfellow et al. 2016][Haykin et al. 2009].

Com uma função de custo definida, precisa-se então ajustar os parâmetros de forma que o custo seja reduzido. Para isso, em geral, usa-se o algoritmo do Gradiente Descendente em combinação com o método de *backpropagation*, que permite obter o gradiente para a sequência de parâmetros presentes na rede usando a regra da cadeia [Ponti e da Costa 2018].

O Gradiente descendente é um algoritmo de otimização iterativo usado para encontrar os valores de parâmetros de pesos e *bias* que minimizam uma função de custo [Goodfellow et al. 2016]. Utiliza o método de aprendizado em lote (*batch*) onde o modelo é treinado usando todos os dados de treinamento disponíveis de uma só vez [Sarkar et al. 2018]. Devido ao grande número de parâmetros a serem aprendidos por uma CNN, torna-se inviável a utilização deste algoritmo ao executá-lo para to-

dos os dados da base de treinamento, devido ao seu elevado custo computacional [Ponti e da Costa 2018].

Já o Gradiente Descendente Estocástico é um método que oferece aproximações ao Grandiente Descendente, e utiliza amostras aleatórios dos dados ao invés de utilizar todos os dados existentes para o treinamento, estes lotes aleatórios de dados são chamados de *mini-batches* [Goodfellow et al. 2016]. Após executar diversas iterações (sendo que cada iteração irá adaptar os parâmetros usando as instâncias no *mini-batch* atual), espera-se obter uma aproximação do método do Gradiente Descendente [Ponti e da Costa 2018].

Os algoritmos *Adam* e *RMSProp* também otimizam a busca por um valor mínimo da função de custo. *Adam*, derivado de "momentos adaptativos", é um algoritmo de otimização baseado em gradiente de primeira ordem de funções estocásticas objetivas, embasado em estimativas adaptativas de momentos de menor ordem [Kingma e Ba 2014]. Empiricamente, o algoritmo *RMSProp* demonstrou ser um algoritmo de otimização efetivo e prático para redes neurais profundas. Atualmente, é um dos métodos de otimização que é empregado rotineiramente por praticantes de aprendizado profundo.

O final da camada totalmente conectada produz um vetor de dimensões K onde K é o número de classes que a rede será capaz de prever. Este vetor contém as probabilidades de uma imagem pertencer a determinada classe. Essas probabilidades são calculadas por uma função de ativação que pode ser a função *softmax* ou *sigmóide*.

A função de ativação *softmax* é basicamente uma generalização da função logística, que representará uma distribuição de probabilidade sobre n possíveis resultados de duas ou mais classes, garantindo que a soma de todas as probabilidades de saída da MLP resulte em 1 [Sarkar et al. 2018]. A função de ativação *sigmóide*, com gráficos em forma de S, é uma função de ativação mais comumente aplicada às redes neurais com no máximo duas classes de classificação. Essa função aplicada a rede só poderá aprender a prever valores entre 0 e 1 [Haykin 2007].

O processo de treinamento de uma rede totalmente conectada é representado na Figura 2.16 sequencialmente:

- Inicio do processo de propagação com a entrada dos dados na primeira camada totalmente conectada da rede com pesos iniciais aleatórios. Aplicação de uma função linear utilizando os dados de entrada juntamente com os valores de pesos e *bias*;
- O fluxo de propagação segue enviando os dados para a próxima camada da rede;
- Ao chegar na última camada da rede, é aplicada uma função de ativação que irá apresentar a probabilidade dos valores de entrada pertencerem a uma classe;
- Valores da predição são comparados com os rótulos das classes através da aplicação de uma função de custo que irá apresentar a distância entre a predição

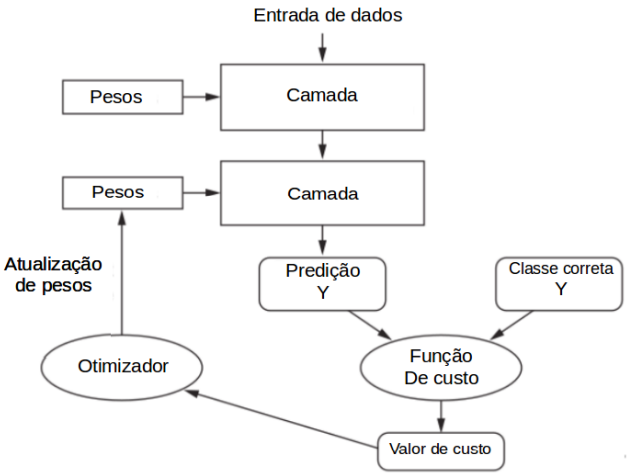

Figura 2.16: O cálculo de erro é usado como *feedback* pelo otimizador para o ajuste de peso da rede.
Fonte: [Chollet 2017]

realizada e a classe correta através do valore de custo.

- Caso o desempenho da rede não seja o suficiente, é iniciado o processo de retropropagação com os valores retornando a rede por meio de um algoritmo de otimização atualizando os pesos da rede com o objetivo de minimizar o valor de custo do modelo.

A diferença fundamental entre uma camada totalmente conectada e uma camada de convolução é a seguinte: camadas totalmente conectadas aprendem padrões globais em seu espaço de recursos de entrada (por exemplo, para um dígito MNIST (*database of handwritten digits*), padrões envolvendo todos os *pixels*), enquanto camadas de convolução aprendem padrões locais [Chollet 2017].

2.7.1 Regularização

Para avaliar um modelo é necessária a utilização e divisão dos dados em três conjuntos: conjunto de treinamento; validação; e teste. Treina-se a rede com os dados de treinamento e avalia este modelo com os dados de validação. Estando o modelo pronto, apresentando baixo valor de custo, é realizada a verificação final com o conjunto de dados de teste [Chollet 2017].

Um problema central no aprendizado de máquina é como criar um algoritmo que funcione bem não apenas nos dados de treinamento, mas também com novas amostras de entrada do conjunto de teste. Algumas estratégias são projetadas para contribuir na redução do valor de custo do modelo durante o treinamento da rede. Essas estratégias são conhecidas como regularização [Goodfellow et al. 2016].

A regularização ajuda a controlar a capacidade do modelo utilizado, garantindo melhorias nas classificações corretas em dados que não foram utilizadas durante a etapa de treinamento e isso é chamado de generalização. A não aplicação de técnicas de regularização pode fazer com que o classificador aprenda padrões específicos para o conjunto de treinamento, mas que tornam-se irrelevantes para novos dados. Portanto, a rede pode acabar memorizando os dados de treinamento e este fenômeno é chamado de superajustamento *overffiting*, desta forma perdendo a capacidade de generalizar [Haykin et al. 2009] [Chollet 2017].

2.7.2 *Dropout*

A técnica de regularização *dropout* fornece um método computacionalmente barato, mas poderoso de regularizar uma ampla família de modelos [Goodfellow et al. 2016]. Esse procedimento pode ser considerado uma forma de regularização pois, ao desativar aleatoriamente neurônios, perturba os *feature maps* gerados a cada iteração por meio da redução da complexidade das funções [Ponti e da Costa 2018].

O motivo pelo qual aplica-se o *dropout* é reduzir o *overfitting*, alterando explicitamente a arquitetura da rede no momento do treinamento. A queda aleatória de conexões garante que nenhum nó na rede seja responsável por "ativar" quando apresentado um determinado padrão. A aplicação desta técnica de regularização consiste em desativar aleatoriamente (definindo como zero) vários recursos de saída da camada durante o treinamento. Como exemplo, uma determinada camada normalmente retornaria um vetor [0.2, 0.5, 1.3, 0.8, 1.1] para uma determinada amostra de entrada durante o treinamento. Após aplicar o *dropout*, este vetor terá algumas entradas zero distribuídas aleatoriamente: por exemplo, [0, 0.5, 1.3, 0, 1.1]. A taxa de abandono é a fração dos recursos que são zerados; normalmente é definido entre 0.2 e 0.5 [Chollet 2017].

A aplicação de *dropout* a uma rede neural resulta em uma rede "*thinned*". A *thinned* consiste em uma rede que as unidade de neurônios se mantiveram ativas após aplicação de *dropout*. Uma rede neural com n unidades pode ser vista como uma coleção de 2^n possíveis redes neurais "*thinned*". Todas essas redes compartilham pesos para que o número total de parâmetros ainda seja O (n^2) ou menor. Para cada apresentação de cada caso de treinamento, uma nova rede "*thinned*" é amostrada e treinada [Srivastava et al. 2014].

Exemplificando o funcionamento de uma rede com aplicação *dropout*, a imagem a) da Figura 2.17 representa uma rede padrão onde durante o processo de treinamento os dados passam pelos neurônios e ao final retornam através da retropropagação alterando os pesos. A imagem b) representa a aplicação do *dropout* onde neurônios são desativados e os dados passam pela rede e retornam os resultados por *backpropagation*. Este processo é repetido, primeiramente restaurando os neurônios desativados, depois escolhendo um novo subconjunto aleatório de neurônios a serem desativados, e realizando o treinamento e atualização dos pesos na rede.

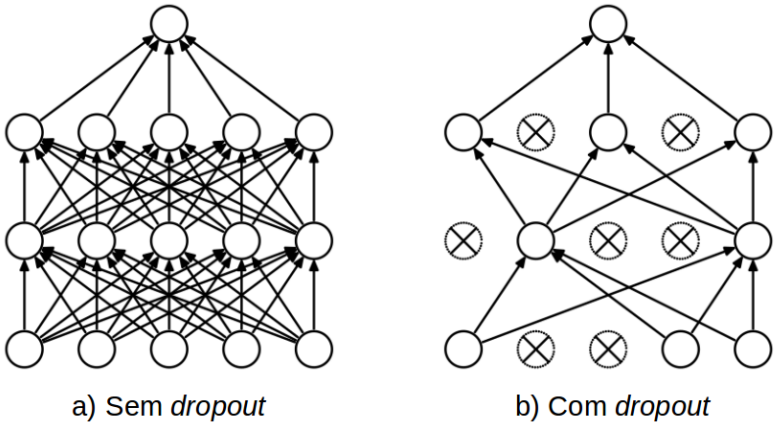

Figura 2.17: A) Rede sem aplicação de *dropout*. B) Exemplo de uma rede *thinned* produzia aplicando *dropout*.
[Srivastava et al. 2014]

Após o processo de treinamento e na apresentação do conjunto de testes, como novas amostras não utilizadas no treinamento, nenhuma unidade é descartada; ao invés disso, os valores de saída da camada são reduzidos por um fator igual à taxa de desistência, para equilibrar o fato de que mais unidades estão ativas do que no tempo de treinamento [Srivastava et al. 2014].

2.8 *Dataset Augmentation*

A melhor maneira de fazer um modelo de aprendizado de máquina generalizar melhor é treiná-lo em mais dados [Goodfellow et al. 2016]. Mesmo quando os dados são de baixa qualidade, os algoritmos podem realmente ter um melhor desempenho, contanto que dados úteis possam ser extraído pelo modelo do conjunto de dados original [Perez e Wang 2017].

O *Data augmentation* ou aumento de dados é outra maneira pela qual é possível reduzir o *overfitting*, superajustamento em modelos, onde aumenta-se a quantidade de dados de treinamento usando informações apenas nos dados de treinamento. O campo de aumento de dados não é novo e, de fato, várias técnicas de aumento de dados foram aplicadas a problemas específicos. As principais técnicas se enquadram na categoria de deformação de dados, que é uma abordagem que procura aumentar diretamente os dados de entrada para o modelo [Perez e Wang 2017].

Essa técnica de *data augmentation* é particularmente efetiva para problemas de classificação como reconhecimento de objetos [Goodfellow et al. 2016]. Para a tarefa de classificação de imagem, alguns métodos de transformação de imagem são empregados para gerar novas amostras a partir do conjunto de treinamento [Shijie et al. 2017]. Os métodos de transformação de imagem mais usados são:

- *Flipping*: inversão ou espelhamento da imagem no sentido horizontal e vertical.
- *Rotation*: possibilita a rotação da imagem em sentidos aleatórios ou nos ângulos 90 °, 180 ° e 270 ° já predefinidos.
- *Shifting*: Nesse critério a imagem é deslocada para a esquerda ou para a direita e o intervalo de tradução e o comprimento do passo podem ser especificados manualmente para alterar a localização do conteúdo da imagem.
- *Noise*: possibilita a inserção de ruídos aleatórios nos canais RGB da imagem.
- *Cropping*: viabiliza o recorte de uma parte da imagem original e seu redimensionamento da imagem recortada, para uma resolução específica.
- *Resize*: as imagens possuem dimensões distintas e este parâmetro realiza a padronização de dimensionalidade de imagem.

2.9 *Arquiteturas de Redes Convolucionais*

A arquitetura refere-se a estrutura geral da rede: quantas unidades de camadas deve ter e como essas unidades devem ser conectadas umas às outras. A maioria das arquiteturas de rede organiza essas camadas em uma estrutura de cadeia, com cada camada sendo uma função da camada que a precedeu. Nessas arquiteturas, as principais considerações a serem feitas são referentes a escolha da profundidade da rede e a largura de cada camada [Goodfellow et al. 2016].

Como exemplo, uma rede com até uma camada oculta pode ser suficiente para se ajustar ao conjunto de treinamento. Redes mais profundas muitas vezes são capazes de usar menos unidades por camada e muito menos parâmetros e por várias vezes generalizam para o conjunto de testes, mas também são frequentemente mais difíceis de otimizar. A arquitetura de rede ideal para uma tarefa deve ser encontrada por meio de experimentação guiada pelo monitoramento do erro do conjunto de validação [Goodfellow et al. 2016].

A seguir serão apresentadas, seguindo a ordem cronológica e tamanho estrutural, as arquiteturas de redes convolucionais: *Lenet*, *AlexNet*, e *Inception*.

2.9.1 *LeNet*

A arquitetura *LeNet* foi introduzida por Lecun [LeCun et al. 1998] com o objetivo de reconhecimento óptico de caracteres (OCR). Modelo pioneiro que introduziu rede neural convolucional para classificação de imagens.

A *LeNet* é composta por 7 camadas, sem contar a entrada, com todas contendo parâmetros treináveis (pesos). A entrada é uma imagem de 32 x 23 *pixels*. Isso é

significativamente maior que o maior caractere no banco de dados MNIST (*Modified National Institute of Standards and Technology*), base de dados de dígitos manuscritos. A razão é que é desejável que características distintivas de potencial, tais como pontos finais ou ângulos de traço possam surgir no centro do campo receptivo dos detectores de características de nível mais alto [LeCun et al. 1998].

A figura 2.18 representa uma rede *LeNet* composta de duas camadas convolucionais que aplica 20 e 50 filtros respectivamente com dimensão de 5 x 5, seguidas de duas camadas de *pooling*, uma camada totalmente conectada com 500 neurônios e uma camada de ativação. A *LeNet* é direta e pequena (em termos de memória ocupada), tornando-se um bom exemplo para o entedimento dos conceitos básicos das CNNs [Sarkar et al. 2018].

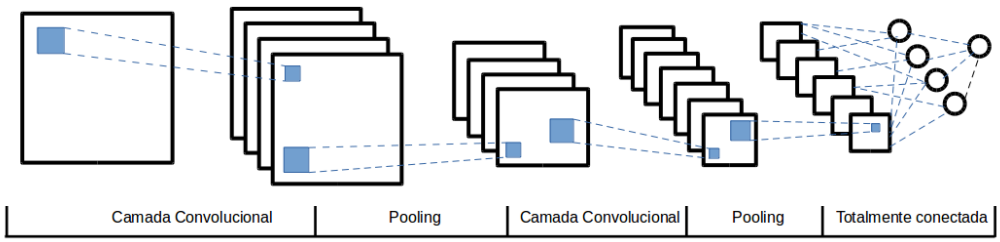

Figura 2.18: Arquitetura modelo LeNet
[Sarkar et al. 2018]

O número de parâmetros de uma rede convolucional está relacionado aos valores a serem aprendidos em todos os filtros nas camadas convolucionais [Ponti e da Costa 2018]. No caso do modelo *LeNet* o total de parâmetros é de 60 mil.

2.9.2 *AlexNet*

AlexNet foi desenvolvido por Alex Krizhevsky [Krizhevsky et al. 2012] para competir na ILSVRC (*ImageNet Large-Scale Visual Recognition Challenge*). ImageNet é um conjunto de dados de mais de 15 milhões de imagens de alta resolução rotuladas, pertencentes a aproximadamente 22.000 categorias e que promove a competição que seleciona o melhor modelo de visão computacional.

A arquitetura geral é bastante semelhante a rede *LeNet-5*, embora este modelo seja consideravelmente maior. Recebe imagens com a dimensão de 227 x 227 x 3 e possui as seguintes camadas: a primeira camada aplica 96 filtros com dimensão de 11 x 11 seguida de uma camada com função *maxpooling*; a segunda camada aplica 256 filtros com dimensão de 5 x 5 e é seguida de uma camada com função *maxpooling*; a terceira camada com 384 filtros com dimensão de 3 x 3; a quarta camada com 384 filtros com dimensão de 3 x 3; a quinta camada aplica 256 filtros com dimensão de 3 x 3, seguida de uma cada *maxpooling*; duas camadas totalmente conectadas com 4096 neurônios; e uma camada de ativação com 1000 classes de saída.

Em seu artigo Krizhevsky [Krizhevsky et al. 2012] utiliza o treinamento da rede em múltiplas GPU (*Graphics Processing Unit*), conforme Figura 2.19. O processo funciona paralelizando o processamento em duas GPUs pois elas podem ler e gravar na memória do outro diretamente, sem passar pela memória da máquina *host*. O esquema de paralelização empregado coloca metade dos *kernels* (ou neurônios) em cada GPU, com um truque adicional: as GPUs se comunicam apenas em certas camadas.

A arquitetura *AlexNet* apresenta 60 milhões de parâmetros, aumentando significativamente os valores a serem aprendidos e também o custo computacional, comparando com os 60 mil do modelo *LeNet*.

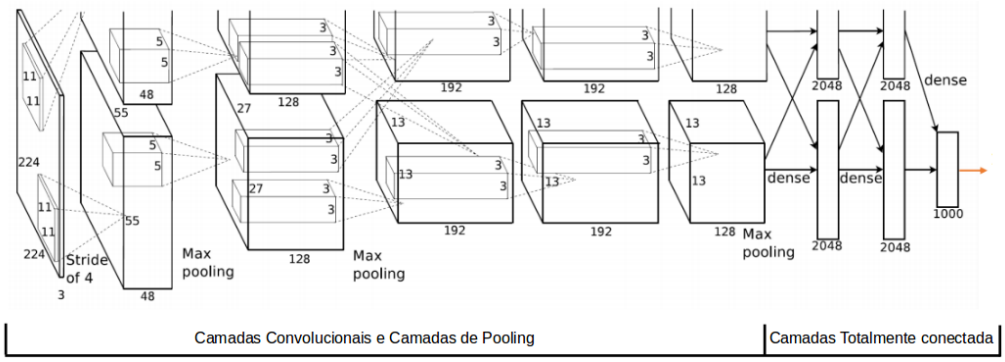

Figura 2.19: Arquitetura do modelo AlexNet
[Krizhevsky et al. 2012]

2.9.3 *Inception*

Rede desenvolvida por engenheiros da *Google* que propuseram o modelo conhecido como *GoogLeNet* e que venceu a competição ILSVRC no ano de 2014, e que posteriormente foi definida pelo nome de *Inception*. Esta arquitetura consiste em uma pilha de módulos que parecem ser pequenas redes independentes, divididas em vários ramos paralelos [Chollet 2017]. Este modelo oi introduzido por Szegedy [Szegedy et al. 2015] e tem o objetivo de atuar como um extrator em vários níveis da imagem, realizando convoluções com três diferentes tamanhos de *kernel*, 1 x 1, 3 x 3 e 5 x 5, e desta forma, abranger elementos de interesse que apresentam variados tamanhos e posições em uma imagem.

A Figura 2.20 apresenta a forma mais básica do módulo de *Inception* que tem de três a quatro ramificações começando com uma convolução de 1 x 1, seguida por uma convolução 3 x 3 e 5 x 5 e terminando com a concatenação dos recursos resultantes. Essa configuração ajuda a rede a aprender separadamente os recursos espaciais e os recursos do canal, o que é mais eficiente do que aprendê-los em conjunto. Na Figura 2.21 é possível visualizar a estrutura completa da primeira versão da rede como

a sequência de 9 módulos *Inception v1* em sequência, destacadas com retângulos pontilhados, que resultam em 27 camadas.

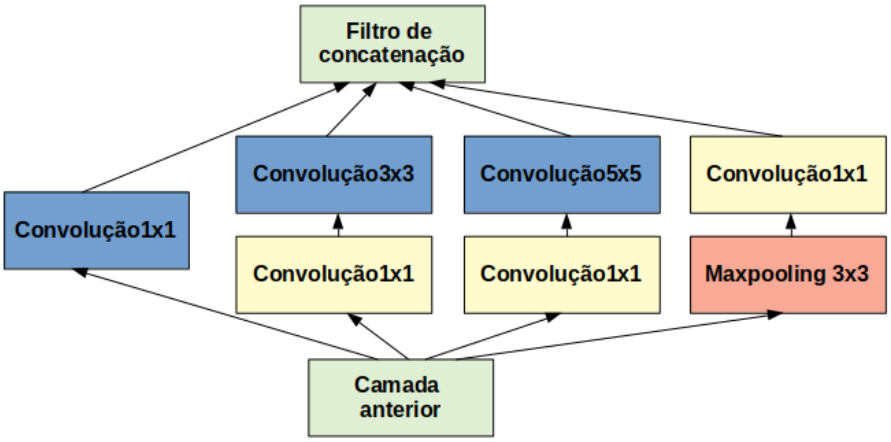

Figura 2.20: Módulo *Inception*
[Szegedy et al. 2015]

Figura 2.21: Estrutura de rede *Inception v1*
[Szegedy et al. 2015]

Muitos dos ganhos da rede *GoogLeNet* [Szegedy et al. 2015] surgem de um uso muito generoso através da redução de dimensão da imagem, que pode ser visto como um caso especial do uso de convoluções fatorizantes de uma maneira computacionalmente eficiente. Porém, qualquer redução no custo computacional desta maneira implica na redução de número de parâmetros.

No artigo [Szegedy et al. 2016] são propostas as versões *Inception v2* e *Inception v3* objetivando explorar maneiras de ampliação das redes de de modo a utilizar a computação de forma mais eficiente posível por meio de convoluções fatorizadas e aplicação de modos de regularização mais agressiva. Diferentemente da primeira versão que possuia somente 1 módulo *Inception*, na terceira versão do modelo serão aplicados 3 módulos que foram representados na Figura 2.22 como módulos A, B e C. Na versão três da arquitetura *Inception* a ideia é melhor desempenho sem que as convoluções alteram drasticamente as dimensões da entrada, pois reduzir demais as

dimensões pode causar perda de informações. Usando métodos de fatoração inteligentes, as convoluções podem se tornar mais eficientes em termos de complexidade computacional.

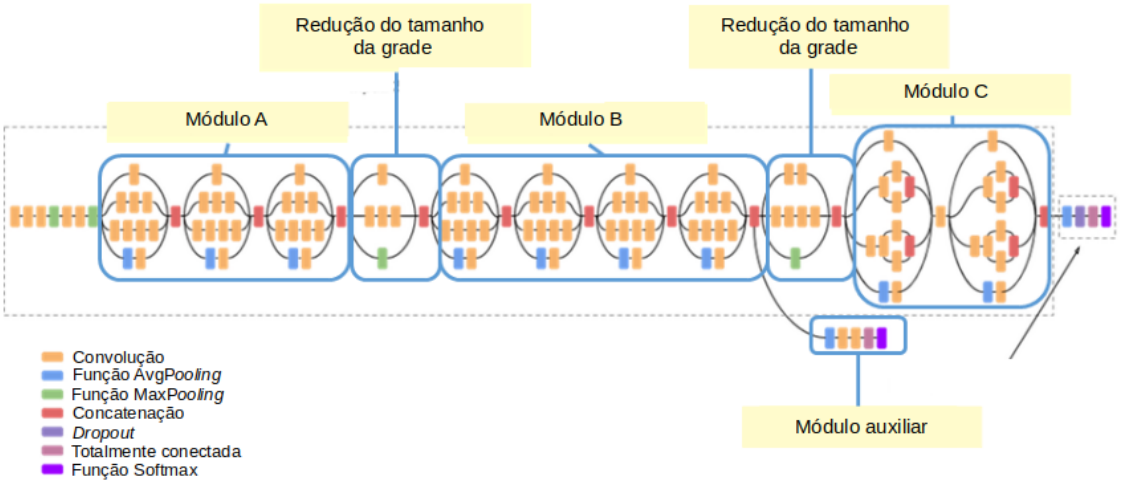

Figura 2.22: Estrutura de rede *Inception v3*
Fonte: https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

No módulo *inception* v1, Figura 2.20 é realizada uma convolução 5 x 5, enquanto que no módulo A, Figura 2.23, é feita a substituição a convolucção 5 x 5 por duas operações de convolução 3 x 3 para melhorar a velocidade computacional. Embora isso possa parecer contra-intuitivo, uma convolução de 5 x 5 é 2.78 vezes mais cara computacionalmente do que uma convolução de 3 x 3. Portanto, o empilhamento de duas convoluções 3 x 3 leva a um aumento no desempenho da rede [Szegedy et al. 2016].

Para a versão atualizada da rede, verificou-se que os resultados poderiam ser melhores se utilizasse convoluções assimétricas. Por exemplo, usar uma convolução 3 x 1 seguida por uma convolução 1 x 3 é equivalente a deslizar uma rede de duas camadas com o mesmo campo receptivo que em uma convolução 3 x 3, como mostra a Figura 2.24. Em teoria, é possível substituir qualquer convolução n x n por uma convolução de 1 x n seguida de uma convolução n x 1 com a economia computacional aumentando drasticamente à medida que n cresce. Essa solução não funcionou bem nas camadas iniciais, porém com resultados muito bons em tamanhos médios de grade usando convoluções 1 x 7 seguidas de convoluções 7 x 1, conforme Figura a) 2.25. A proposta do módulo C é o aumento da dimensionalidade sendo aplicado após os módulos A e B pois é o local um local crítico onde se produz representação esparsa de alta dimensionalidade, pois a taxa de processamento local (por 1 x 1 convoluções) é aumentada em comparação com a agregação espacial. Representações dimensionais mais altas são mais fáceis de processar localmente dentro de uma rede. Aumentar as ativações por bloco em uma rede convolucional permite mais recursos e mais rapidez no treinamento [Szegedy et al. 2016].

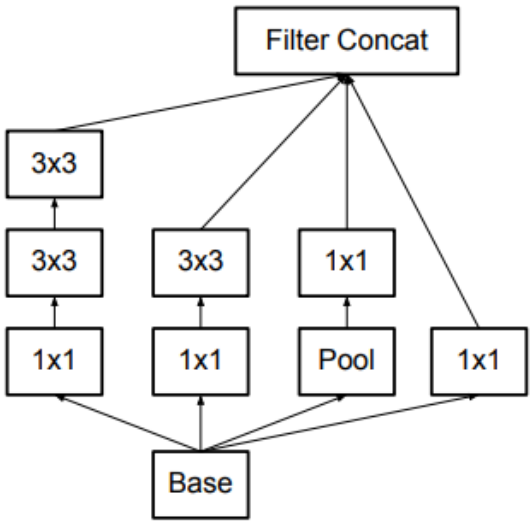

Figura 2.23: Módulo A da *Inception v3*
[Szegedy et al. 2016]

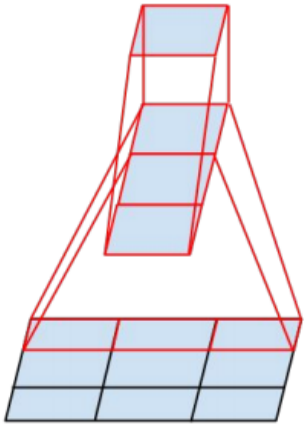

Figura 2.24: Equivalência de uma convolução 3 x 1 seguida por uma convolução 1 3 com uma rede utilizando convolução 3 x 3
[Szegedy et al. 2016]

É possível verificar na Figura 2.22 a identificação de módulo auxiliar que na verdade é um módulo de classificador auxiliar que atua como um regularizador. Isso é suportado pelo fato de que o classificador principal da rede tem um desempenho melhor se o ramo secundário for normalizado por lotes ou tiver uma camada de *dropout*. Isso também fornece uma evidência de suporte fraca para a conjetura de que a normalização em lote atua como um regularizador. É também utilizado módulos para reduzir o tamanho da grade dos mapas de atributos como forma de reduzir o custo computacional da rede [Szegedy et al. 2016].

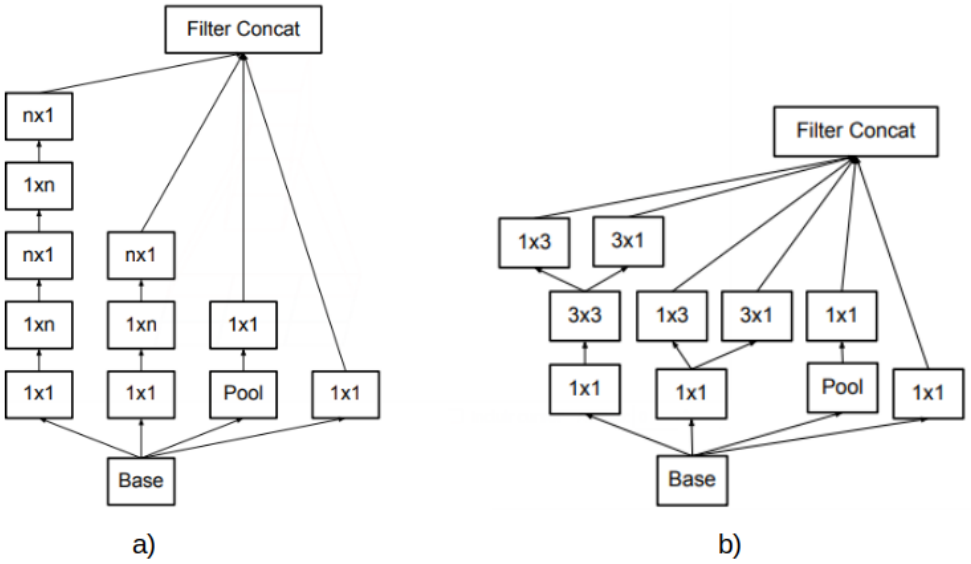

Figura 2.25: Figura a) representando o módulo B; e b) representando o módulo C da rede *inception* v3.
[Szegedy et al. 2016]

2.10 Avaliação dos Classificadores

Quando os modelos são desenvolvidos, seu desempenho é avaliado no conjunto de validação e é selecionado o modelo com o melhor desempenho para a aplicação das amostras de teste final. Para esta seleção são aplicados métodos que avaliam o processo de aprendizagem, como por exemplo acompanhar as medidas de precisão do modelo [Sarkar et al. 2018].

Em um modelo de classificação existem 4 saídas possíveis: o número de classificações corretas para a classe positiva (verdadeiro-positivo), que é representado por TP (*True Positive*); o TN (*True Negative*) que é o número de classificações corretas para a classe negativa (verdadeiro-negativo); o FP (*False Positive*) que é o número de classificações incorretas para a classe negativa (falso-positivo); e o FN (*False Negative*) que é o número de classificações incorretas para a classe positiva (falso-negativo) [Fawcett 2006].

Dados os resultados de saída de um modelo de classificação é possível construir uma matriz de confusão 2 x 2 (colunas e linhas) que gera uma base de várias medidas de avaliação do modelo como a acurácia, precisão, sensibilidade e o *F1-Score* [Fawcett 2006][Sarkar et al. 2018].

A acurácia é a proporção de exemplos para os quais o modelo produz a saída correta. E também pode-se obter informações equivalentes medindo a taxa de erro, a distorção de exemplos para os quais o modelo produz uma saída incorreta

[Goodfellow et al. 2016]. Essa métrica é uma das medidas mais populares de avaliação de classificadores e é definida como acurácia global ou proporção de previsões corretas do modelo. A fórmula para calcular a acurácia, através da matriz de confusão é:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.1}$$

Além da acurácia, outras medidas padrões utilizadas na avaliação de desempenho são: precisão, a sensibilidade e o *F1-Score*.

A precisão (*precision*), também conhecida como valor preditivo positivo, é outra métrica que pode ser derivada da matriz de confusão. Calcula a quantidade de previsões feitas que são realmente corretas ou relevantes de todas as previsões baseadas na classe positiva. A fórmula para precisão é a seguinte:

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

Um modelo com alta precisão identificará uma fração maior da classe positiva em comparação a um modelo com menor precisão.

Recall, também conhecida como sensibilidade, é uma medida de um modelo para identificar a porcentagem de pontos de dados relevantes. É definido como o número de instâncias da classe positiva que foram corretamente previstas. Isso também é conhecido como taxa de acertos, cobertura ou sensibilidade [Sarkar et al. 2018]. A fórmula para *recall* é:

$$Recall = \frac{TP}{TP + FN} \tag{2.3}$$

O *recall* se torna uma medida importante do desempenho do classificador em cenários em que se quer capturar o maior número de instâncias de uma determinada classe, mesmo quando ela aumenta os falsos positivos.

F1-Score é uma métrica que é a média harmônica de precisão e recuperação e ajuda a otimizar um classificador para precisão balanceada e desempenho de recordação. Em alguns casos é necessária uma otimização balanceada de precisão e recuperação [Sarkar et al. 2018].

A fórmula para a *F1-Score é*:

$$F1Score = \frac{2x(Precision)x(Recall)}{Precision + Recall} \tag{2.4}$$

2.10.1 Validação Cruzada

As medidas descritas anteriormente não são suficientes para avaliar um modelo de aprendizagem, pois ao dividir os conjuntos de amostras de validação e teste, é desconsiderada uma quantidade significativa de dados que poderiam ser usados para refinar o processo de modelagem. Além disso, outro ponto importante é que se o erro do modelo de uma única iteração for considerado como o erro geral, um erro grave estará sendo cometido. Ao invés disso, o que se deve fazer é obter uma medida média de erro, construindo várias iterações do mesmo modelo. Portanto, reconstruir o modelo com o mesmo conjunto de dados não resultará mudanças em seu desempenho [Sarkar et al. 2018].

Alguns procedimentos alternativos permitem o uso de todos os exemplos na estimativa do erro médio de teste, ao preço do custo computacional aumentado. Esses procedimentos baseiam-se na ideia de repetir o treinamento e o cálculo de testes em diferentes subconjuntos escolhidos aleatoriamente ou em divisões do conjunto de dados original. Um desses procedimentos é o *K-fold Cross Validation* ou validação cruzada que tem como ideia obter diferentes divisões de conjuntos de treinamento e validação (diferentes observações em cada conjunto, a cada vez) usando alguma estratégia e, em seguida, construir várias iterações de cada modelo nessas diferentes divisões [Sarkar et al. 2018].

Na validação cruzada é feita a divisão dos dados em k subconjuntos iguais. Em seguida, realizada as k iterações de aprendizado; em cada rodada, *1/k* dos dados são mantidos como um conjunto de testes e os exemplos restantes são usados como dados de treinamento. A pontuação média do conjunto de testes das k iterações deve então ser uma estimativa melhor do que uma única pontuação, conforme Figura 2.26. Os valores populares para k são 5 e 10 - o suficiente para fornecer uma estimativa estatisticamente provável de ser precisa, com um custo de 5 a 10 vezes mais tempo de computação [Russell e Norvig 2013].

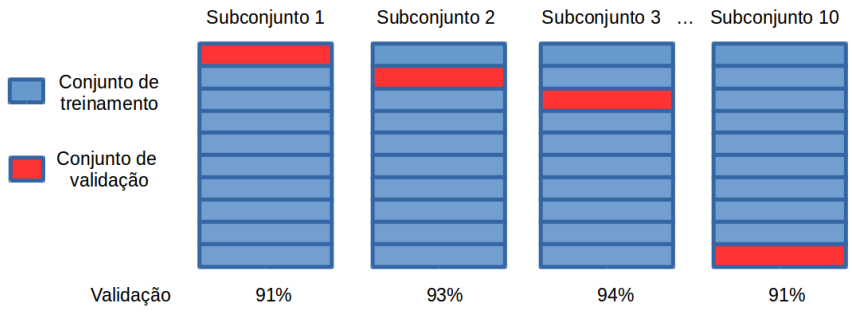

Figura 2.26: Exemplo de uso do processo de validação cruzada de K subconjuntos
Fonte:https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6

O erro médio nessas divisões é então relatado como o erro do modelo em questão e a decisão final é feita nessa métrica de erro médio. Essa estratégia tem um efeito

considerável no erro estimado de cada modelo, pois garante que o erro médio seja uma aproximação do erro do modelo em dados realmente novos (conjunto de testes) e também poderíamos aproveitar o conjunto de dados de treinamento completo para a construção do modelo [Sarkar et al. 2018].

2.10.2 Testes estatísticos

Para analisar se existem diferenças nas técnicas aplicadas a cada algorítmo de classificação utilizando aprendizado de máquina com validação cruzada, é necessário avaliar, através de testes estatísticos, os resultados de acurácia, *recall*, precisão e *F1-Score*.

Embora o desempenho de diferentes classificadores possa ser mostrado como diferente em conjuntos de dados especificados, é necessário confirmar se as diferenças observadas são estatisticamente significativas e não meramente coincidentes. Desta maneira, um teste estatístico irá avaliar a diferença de desempenho entre classificadores quanto a questões de significância. O teste de significância estatística permite que os pesquisadores passem para avaliações mais precisas de significância dos resultados obtidos [Japkowicz e Shah 2011].

O teste estatístico de hipóteses, por vezes referido apenas como teste de hipóteses, desempenha um papel muito importante na inferência estatística. Faz parte do ramo da estatística conhecido como análise de dados confirmatória. Assim, como o nome sugere, a principal preocupação de sua aplicação é confirmar se o comportamento observado é representativo do comportamento verdadeiro [Japkowicz e Shah 2011].

Para o teste de hipóteses, assume-se uma hipótese nula, denominada de H_0. É necessário determinar a hipótese contrária a hipótese nula, definida como H_1, que será verdadeira, caso H_0 seja falsa. A hipótese nula pode ser a hipótese de que os classificadores sejam iguais, não apresentando diferença significativa. É necessário definir um nível de significância (α), que corresponde à probabilidade de rejeitar H_0 quando H_0 for verdadeiro (erro Tipo I) ou não rejeitar H_0 quando H_0 for falso (erro Tipo II). Em seguida, a estatística do teste e o *P-value* são calculados. O *P-value* representa a probabilidade de significância, caso o *P-value* seja menor que o valor de (α), o resultado é considerado como significativo e a hipótese nula é rejeitada [Johnson 1999].

Dentre os vários testes estatísticos estão os Teste de Wilcoxon e o Teste de Friedman. Ambos são teste não paramétricos, que significa que a distribuição das informções não apresentam um padrão de distribuição, sendo distribuídas de forma livre.

O Teste de Wilcoxon é do tipo não paramétrico usado para comparar estatisticamente dois algoritmos aplicados a um único domínio de amostras de teste ou dois algoritmos aplicados a múltiplus domínios [Japkowicz e Shah 2011]. Este teste utiliza o método de ranqueamento, isto é, a pontuação *1, 2, 3 ... n* é substituída

pelos dados numéricos reais, a fim de se obter uma ideia aproximada rápida da significância das diferenças entre os experimentos [Wilcoxon 1945].

Tabela 2.2: Dados de exemplo para aplicação do teste de Wilcoxon para 10 domínios.

Domínio	**C1**	**C2**	**C1 - C2**	**|C1 - C2|**	**Rank 1**	**Rank 2**
1	0.9643	0.9933	-0.0301	0.0301	3	-3
2	0.7342	0.8134	-0.0792	0.0792	6	-6
3	0.7230	0.9151	-0.1921	0.1921	8	-8
4	0.7170	0.6616	+0.0554	0.0554	5	+5
5	0.7167	0.7167	0	0	–	–
6	0.7436	0.7708	-0.0272	0.0272	2	-2
7	0.7063	0.6221	+0.0842	0.0842	7	+7
8	0.8321	0.8063	+0.0258	0.0258	1	+1
9	0.9822	0.9358	+0.0464	0.0464	4	+4
10	0.6962	0.9990	-0.3028	0.3028	9	-9

A Tabela 2.2 [Japkowicz e Shah 2011] exemplifica o método do teste de Wilcoxon com os dados distribuídos da seguinte maneira: a primeira coluna representa os 10 domínios utilizados no teste; as colunas C1 e C2 apresentam os resultados de acurácia de dois classificadores distintos; a coluna C1 - C2 apresenta a subtração das acurácia dos dois classificadores; a coluna |C1 - C2| com os valores absolutos da subração sem sinal; e as colunas Rank 1 e Rank 2 apresentam o ranqueamento dos valores resultantes da subtração das acurácias, sendo que o Rank 2 replica os sinais apresentados na coluna C1 - C2.

A soma dos valores por sinais é computada, produzindo os valores: W_{s1}=17, como somatório do ranqueado com valores positivos; e W_{s2}=28, como resultado do somatório do ranqueamento dos valores negativos. A estatística Wilcoxon é calculada como $T_{wilcoxon} = min(W_{s1}, W_{s2})$, com o valores resultante de $T_{wilcoxon} = 17$.

Comparando o valor crítico de V_α, Se $V_\alpha >=$ ao $T_{wilcoxon}$ a hipótese nula H_0 é rejeitada de que o desempenho dos dois classificadores é o mesmo, no nível de confiança α. Para $n = 10 - 1$, valor é subtraído devido o domínio 5 ter apresentado valor 0 na subtração entre C1 e C2, graus de liberdade e $\alpha = 0.005$, $V = 8$ para o teste de *one-sided*, o valor V é definido por uma tabela de valores críticos que compara a quantidade de domínios utilizados.

Um teste de hipóteses estatísticas é chamado *one-sided* se os valores que podem rejeitar a hipótese nula estiverem contidos em uma única ponta da distribuição de probabilidade. Ou seja, esses valores são todos menores do que o limite do teste (também conhecido como o valor crítico do teste) ou acima do limite, mas não de ambos. Por outro lado, um *two-sided* permite rejeitar a hipótese nula, levando em conta ambas as caudas da distribuição de probabilidade [Japkowicz e Shah 2011].

O valor de V deve ser maior que o $T_{wilcoxon}$ para rejeitar a hipótese. Desde $17 > 8$, não se pode rejeitar a hipótese de que o desempenho do C1 é igual ao do C2 no nível 0.005.

2.11 Trabalhos Relacionados

Nesta seção serão abordados trabalhos que utilizaram métodos de aprendizado de máquina na classificação de pragas, como: Identificação de gêneros de formigas usando um conjunto de redes neurais convolucionais [Marques et al. 2018]; Pesquisa sobre detecção e reconhecimento de imagens de pragas de insetos com base em métodos bio-inspirados [Deng et al. 2018]; Detecção de insetos de grãos armazenados usando aprendizado profundo [Shen et al. 2018]; Classificação automática do *Diaphorina citri* em imagens de microscopia [Melo 2016]; Localização e classificação de pragas de arrozal usando um mapa de saliência e rede neural convolucional profunda [Liu et al. 2016]; Detecção automática de traças a partir de imagens de armadilhas para o manejo de pragas [Ding e Taylor 2016].

No trabalho de Marques [Marques et al. 2018] foi aplicada Redes Neurais Convolucionais para identificar gêneros de formigas, explorando a diversidade de múltiplas perspectivas de cabeça, dorso e perfil da mesma amostra de formiga, utilizando de imagens de formigas no maior banco de dados *on-line* sobre biologia de formigas, o *AntWeb*. Devido a estas diferenças, foi decidido fazer máquinas de aprendizado específicas para cada uma delas, que foram chamadas de classificadores de propósito específico. Com isso foi proposto utilizar três classificadores, um para cada visão. A primeira proposta foi utilizar o modelo *AlexNet*, que possui cinco camadas convolucionais, para a extração dos *features maps*, e duas camadas totalmente conectadas para a classificação. A segunda proposição aplicou o método de *transfer learning*, que usa dados pré-treinados do *ImageNet* para melhorar o desempenho individual dos classificadores, com uma arquitetura de rede baseada no modelo *AlexNet*. Na última abordagem proposta, extraíram as saídas da penúltima camada de uma CNN da *AlexNet* treinada com o conjunto de dados *imagenet.org* e aplicaram essas saídas como vetores de recurso para o treinamento de uma SVM. O desempenho alcançado pelos classificadores é diversificado o suficiente para promover uma redução no erro de classificação geral quando eles são combinados, alcançando uma taxa de precisão de mais de 80% na classificação top-1 e uma precisão de mais de 90% no top-3.

No trabalho de Deng [Deng et al. 2018] foi proposto um sistema baseado na visão humana para o reconhecimento de pragas e insetos. Na detecção do objeto de interesse (ROI do inglês *Region of Interest*) foi utilizado mapas de saliência, através da deteminação da frequencia com que ocorrem nas imagens. Para extrair os recursos invariantes na representação das pragas, foi aplicado o modelo hierárquico bio-inspirado e o modelo X (HMAX), aplicado juntamento com o algoritmo *Scale Invariant Feature Transform* (SIFT). Após a extração de característica, a classificação foi realizada pelo algorítmo de aprendizado de máquina SVM, com o modelo

Radial Basis Function (RBF). Resultados experimentais mostraram que o método de detecção poderia detectar a região do objeto em imagens de ambientes naturais complexos. O método proposto demonstrou bom desempenho de reconhecimento com uma precisão de 85,5%, e fornece uma nova ideia e método para a rápida detecção e reconhecimento de pragas de insetos.

No trabalho de Shen [Shen et al. 2018] foi desenvolvido um método de detecção e identificação de insetos armazenados em grãos com aplicação de redes neurais profundas. O método foi criado baseando-se no *Faster R-CNN*, que pode ser usado para identificar os insetos misturados sob diferentes condições de iluminação. Foi criada uma base de imagens dos insetos *Cryptoleste Pusillus(S.), Sitophilus Oryzae(L.), Oryzaephilus Surinamensis(L.), Tribolium Confusum(Jaquelin Du Val), Rhizopertha Dominica(F.) and Lasioderma Serricorne(F.)* contendo 739 amostras. Para incrementar o conjunto de treinamento objetivando maior generalização da rede e prevenir *overfitting*, foi utilizado o recurso de *data augmentation* com: adição de ruídos, ajustes aleatório de brilho, contraste e nitidez da imagem. Após este processo, a base de imagem foi aumentada em 12 vezes. Os insetos na imagem foram contornados com uma caixa azul, de forma manual, para serem utilizados no treinamento da rede. A detecção da região de interesse foi feita por *Region Proposal Network* (RPN) e rede neural convolucional para obter as áreas de interesse com mais rapidez e precisão através da aceleração da Unidade de Processamento Gráfico (GPU, do inglês *Graphics Processing Unit*). A rede *Inception* foi usada para extrair os mapas de características de cada imagem, depois o RPN retornou as coordenadas das áreas que poderiam ter insetos nos mapas de atributos. Para a classificação, foi aplicada a ROI *Pooling Layer*, que mapeou a caixa candidata para o mapa de atributos que foi a saída da sétima estrutura da rede *Incepction*. A proposta obteve bons resultados de precisão média (mAP) tanto na detecção como na classificação dos insetos.

No trabalho de Melo [Melo 2016] foram experimentados e combinados métodos computacionais para a detecção e extração de características ORB (*Oriented FAST and Rotated BRIEF*), SIFT (*Scale-Invariant Feature Transform*), SURF (Speeded-Up Robust Features), BRISK (*Binary Robust Invariant Scalable Keypoints*) e FREAK(*Fast Retina Keypoint*). Utilizou-se o algoritmo de agrupamento de características *Mini Batch K-Means*, que é um tipo de aprendizagem de máquina não supervisionada que forma automaticamente grupos (*clusters*) de características similares e os métodos de aprendizagem de máquina K-Vizinhos Mais Próximos (KNN) e o SVM na classificação do inseto *Diaphorina citri*. Foi criado um banco de imagens com 1152 imagens adquiridas por microscópio (imagens cedidas pelo FUNDECITRUS) com uma ampliação de imagem fixa de 0.67x, sendo 576 do *Diaphorina citri* e o restante de imagens com outros insetos. Após estudos, foram definidas a abordagem com os detectores e extratores de características SURF/SIFT e a quantidade de 700 grupamentos de caraterísticas do algoritmo *Mini Batch K-Means* e em seguida, as bolsas de características utilizadas como entrada para os procedimentos computacionais de classificação do inseto. O SVM foi selecionado como classificador por apresentar melhores resultados comparados com os obtidos pelo KNN. A abordagem

com o extrator SURF/SIFT, BoF com características do *Diaphorina citri* e SVM com núcleo RBF (*Radius Basis Function*), obteve a maior acurácia no processo de validação cruzada com 98,17% e teve 2,54% como desvio padrão. A acurácia do teste final de generalização de modelo foi de 99,14%. Portanto, o extrator SURF/SIFT juntamente com o algoritmo de aprendizado de máquina SVM alcançou um resultado superior ao medido na pesquisa de Leonardo [Leonardo 2014] que avaliou a eficiência no processo de contagem manual. Com a geração das bolsas de caracterısticas e livros de código, esses dados serviram de entrada para os algoritmos de classificação SVM. Foi utilizada também a validação cruzada *K-Fold* em 10 partes iguais. A validação cruzada foi realizada em 10 iterações 10-*fold* e dividiu a base em 90% pra o desenvolvimento e 10% para o teste final.

No trabalho de Liu [Liu et al. 2016] foi proposta a localização visual e classificação de pragas agrícolas calculando um mapa de saliência e aplicando a aprendizagem da rede neural convolucional profunda. No conjunto de imagens utilizado, os insetos geralmente ocupavam regiões de cores com alto contraste em relação ao seu *background*, regiões de saliência, e por esse motivo foi aplicada a abordagem de detecção de região saliente baseada em contraste global para a detecção de objetos. Com os mapas de saliência computados, foi aplicado o algoritmo GrabCut para a segmentação dos insetos, resultando no recorte de todas as imagens e apresentadas para a entrada da Rede Neural Convolucional. A Arquitetura de rede utilizada foi com base no *AlexNet*, contendo 8 camadas, as cinco primeiras chamadas de camadas convolucionais intercaladas por *max-pooling* e as duas últimas camadas totalmente conectadas,e a camada de classificação final aceita o vetor de representação das camadas anterior para o reconhecimento. Foi realizada a comparação com outros métodos. A precisão média da exatidão da rede baseada na *AlexNet* atingiu 0,834. Combinando isso com a localização baseada em mapa de saliência, os modelos propostos obtiveram desempenho muito melhor, 0,923 e 0,951. A proposta demonstrou a eficácia no uso da abordagem baseada em mapas de saliência em conjunto com as redes de aprendizado profundo para localizar objetos de insetos em imagens naturais.

Ding [Ding e Taylor 2016] propuseram um *pipeline* de detecção baseado em janela deslizante e a aplicação de aprendizado profundo para identificar algum tipo de praga em armadilhas adesivas. Para o objetivo proposto foi criado um banco de imagens de armadilhas adesivas, usadas para a captura de insetos. Foi usada uma CNN baseada na estrutura do modelo *Lenet5*, que consiste em: duas camadas convolucionais; duas camadas *max-pooling*, para reduzir o número de parâmetros livres e introduzir invariância na rede, e duas camadas totalmente conectadas (*fully connected*) que são redes neurais *feed-forward*. Para uma melhor performance de generalização do modelo proposto, foi aplicado o recurso de *data augmentation*, que incorpora transformações geométricas, rotações e espelhamentos para incremento da base de imagens utilizada no treinamento da rede. Tiveram como resultado a produção de 72 imagens criadas para cada amostra original. A entrada de dados na rede proposta foi executada em diferentes tamanhos de imagem: 21x21, 28x28, 35x35, 42x42 e 49x49. A CNN com entrada de imagens com o tamanho 21x21 alcançou o melhor desempenho no

nível de objeto, que detecta traças individuais. E a CNN com entrada de imagens com tamanho de 35x35 teve o melhor desempenho para a detecção de presença ou não de traça. Na base de imagens, algumas mariposas são bloqueadas por outras, o que dificulta sua identificação e se excluídas, apresentaram melhoras na classificação de precisão, aumentando de 0,931 para 0,934 e a média de *log* diminuiu de 0,099 para 0,0916. Os experimentos demonstraram a eficácia do método proposto em um conjunto de dados de traça *codling*.

Capítulo 3

Métodos e experimentos

Para o desenvolvimento deste trabalho foram necessárias as seguintes etapas: Criação do Banco de Imagens das Armadilhas Adesivas Amarelas, Classificação do Inseto *Diaphorina citri* e a Avaliação dos Classificadores. A seguir serão descritas cada uma das etapas.

3.1 Criação do Banco de Imagem das Armadilhas Adesivas Amarelas

O banco de imagens das armadilhas adesivas foi criado com amostras digitalizadas e disponibilizadas pelo FUNDECITRUS. Ao todo foram cedidas 220 amostras de armadilhas adesivas amarelas digitalizadas com as dimensões padronizadas de 10200 *pixels* x 14040 *pixels* e com o tamanho médio de 14 MB, sendo 110 armadilhas com a presença do *Diaphorina citri* e outras 110 armadilhas sem a presença do *Diaphorina citri*.

Para a validação do banco de imagens todas as armadilhas digitalizadas pelo FUNDECITRUS foram disponibilizadas de forma duplicada. Sendo uma imagem original da armadilha e a outra uma cópia da mesma armadilha, porém com marcação circular, realizada pelo corpo técnico do Fundo de Defesa da Citricultura, identificando o *Diphorina citri*, conforme apresentado na Figura 3.1. Também foi disponibilizada a informação de quantidade de insetos *Diaphorina* em cada armadilha, sendo esta bastante variável possuindo desde de um único exemplar até a excepcionalidade de mais de 90 psilídeos.

O recorte dos insetos foi feito de forma manual através do software *ImageJ*, que é uma plataforma para análise de imagens científicas. Com tamanhos variados de insetos, o processo de recorte resultou em: arquivos de imagem dos insetos com tamanhos variados; posições distintas dos insetos no quadrante do recorte; e imagens do inseto

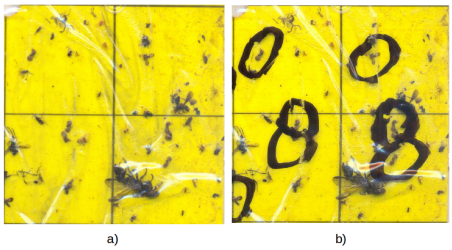

Figura 3.1: a) Imagem original com parte de uma armadilha. b)A mesma imagem, porém com a marcação circular identificando o inseto *Diaphorina* na armadilha.

com ruídos contidos na armadilhas como sujeiras, reflexo do processo de digitalização e as linhas divisórias da armadilha, conforme Figura 3.2.

Posteriormente, foi realizada a verificação, identificação e separação das amostras em diretórios distintos nomeados como *"Diaphorina"* e "outros". Toda a identificação e separação foi realizada manualmente conforme dados fornecidos pelo FUNDECITRUS, onde sabia-se a quantidade de *Diaphorina citri* em cada armadilha e quais insetos eram o psilídeo alvo desta pesquisa.

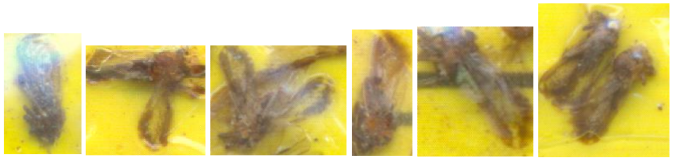

Figura 3.2: Imagens do *Diaphorina citri* com tamanhos distintos e com ruídos contidos nas armadilhas.

A quantidade final de *Diphorina citri* após o processo de recorte manual foi de 551 amostras e a de outros insetos foi de 2.743 amostras. Para igualar as quantidades de imagens, foi estabelecida a quantidade padrão de 551 amostras de insetos *Diphorina citri* e outras 551 imagens do outros insetos que foram selecionadas de forma aleatória.

3.1.1 *Dataset Augmentation*

O método mais fácil e mais comum para reduzir o *overfitting* ou superajustamento em dados de imagem é aumentar artificialmente o conjunto de dados usando transformações de preservação de rótulo, produzindo-se imagens transformadas a partir de imagens originais [Krizhevsky et al. 2012].

Para a tarefa de classificação de imagem, alguns métodos de transformação de imagem são empregados para gerar novas amostras a partir do conjunto de treinamento [Shijie et al. 2017]. Para este trabalho foi utilizado o pacote *Augmentor* para auxiliar o aumento e a geração artificial de imagem para tarefas de aprendizado de

máquina. Foram aplicados métodos de: rotação de 90° e 180° nas amostras do conjunto; a inversão das entradas horizontalmente ou espelhamento das imagens; deslocamento aleatório no sentido horizontal e vertical para treinar sua rede de maneira a lidar com objetos fora do centro da imagem; distorções nas imagens, porém mantendo sua proporção; e redimensionamento para gerar imagens nas dimensões necessárias para cada arquitetura de rede utilizada.

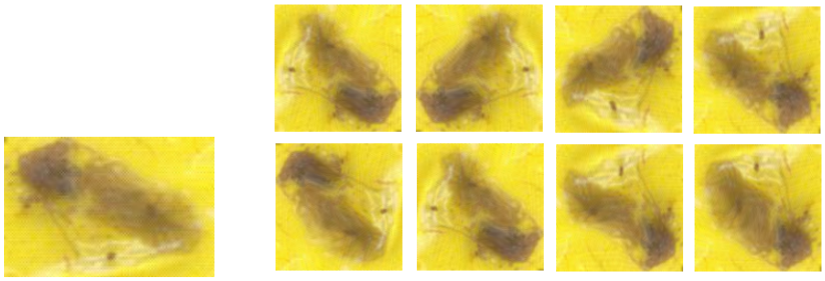

Figura 3.3: A esquerda, imagem original da extração de insetos da armadilha. A direita, imagens criadas pelo processo de aumento de base, a partir da imagem original.

Esse processo de ampliação da base de imagem foi executado 3 vezes para gerar imagens redimensionadas em 32 x 32, 224 x 224 e 299 x 299. A base de imagens teve uma ampliação de 1102 amostras iniciais, divididas em igual quantidade para *Diaphorina citri* e outros insetos, para 8354 amostras, também divididas em igual quantidade para as duas categorias, resultando no aumento de 7.5 vezes o tamanho original da base de imagens, conforme Tabela 3.1.

Tabela 3.1: Base de imagens de insetos.

Quantidade de imagens	**Diaphorina**	**Outros**	**Total**
Quantidade inicial	551	551	1102
Data Augmentation	4177	4177	8354

Ao final do processo, foi estabelecido o *dataset* com 8354 imagens dividido em: um conjunto para treinamento, usado para aprender os parâmetros; conjunto de validação, usado para estimar o erro de generalização durante ou após o treinamento; e o conjunto de dados para teste, usado para a validação final da rede [Goodfellow et al. 2016].

A imagem original e as imagens geradas pelo processo de *data augmentation* permaneceram no mesmo conjunto de imagens. Para exemplificar, a Figura 3.3 mostra uma imagem original do *Diaphorina citri* e oito imagens geradas artificialmente, através de métodos de transformação de imagem, a partir dela e todas foram agrupadas em um mesmo conjunto de imagens, não ficando distribuídas em mais de um dos agrupamentos criados.

Para este trabalho o *dataset* de imagens foi dividido com 70% das amostras destinadas para o conjunto de desenvolvimento e 30% das amostras para o teste final, conforme Figura 3.4.

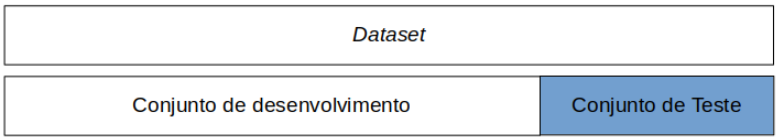

Figura 3.4: 70% das amostras para o conjunto de desenvolvimento e 30% para o conjunto de teste.

3.2 Classificação utilizando a abordagem proposta por Melo [Melo 2016]

Conforme metodologia utilizada no trabalho de Melo [Melo 2016] Figura 3.5, foi realizada uma classificação aplicando o aprendizado de máquina SVM (*Support Vector Machine*) com as amostras do *dataset* criado através do processo de *Data augmentation* nas dimensões de 224 *pixels* x 224 *pixels*. No trabalho citado, foram classificadas imagens do inseto *Diaphorina citri* adquiridas por microscopia.

Foram utilizados os algoritmos SIFT e SURF para a detecção e extração de características. Finalizado o processo de extração e detecção, foram criados livros de código (*codebooks*) com o algoritmo *Mini Batch K-Means* e bolsas de características (*Bag of Features*), organizadas em 700 agrupamentos de atributos, Figura 3.5.

Com a geração das bolsas de características e livros de código, esses dados serviram de entrada para os algoritmos de classificação *Vector Machine Learning* (SVM). Foi utilizado também a validação cruzada *2-Fold* x 5 aplicando a divisão do conjunto de desenvolvimento conforme estabelecido na Figura 3.7.

3.3 Classificação utilizando Redes Neurais Convolucionais

Neste trabalho foi proposta uma abordagem para a classificação do *Diaphorina citri* com *Deep Learning* utilizando Redes Neurais Convolucionais. A escolha por esse método de classificação foi devido ao seu grande sucesso, melhorando o estado da arte em visão computacional, reconhecimento de objetos e detecção de objetos [LeCun et al. 2015].

Os experimentos com os modelos de arquitetura foram feitos utilizando valores de parâmetros semelhantes.

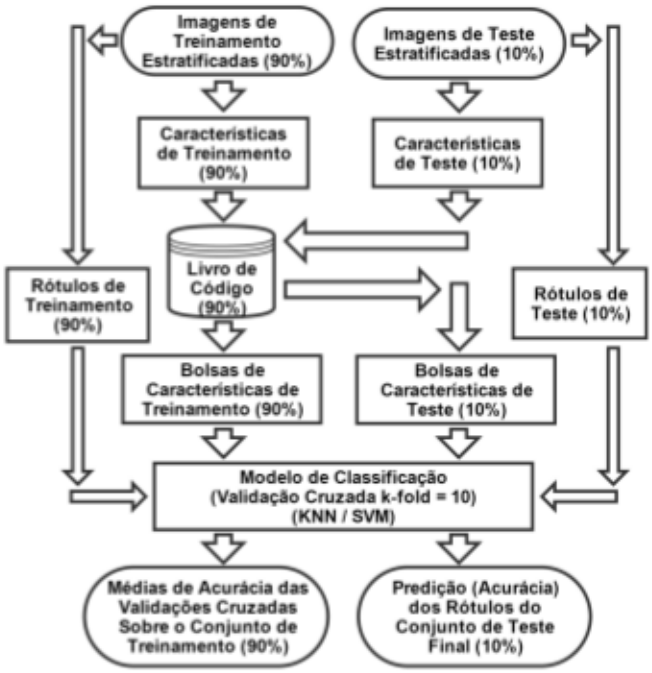

Figura 3.5: Etapas detalhas do processo de classificação da metodologia de [Melo 2016].

Foi definido o valor de 50 *epochs* de iterações do algoritmo, após avaliações empíricas foi detectado que o algoritmo não apresentava melhorias significativas após 50 iterações, e com 32 *batch size* correspondente ao grupo de imagens selecionadas para o treinamento da rede. O uso de pequenos valores de *batch size* alcançam melhor estabilidade de treinamento e desempenho de generalização, para um dado custo computacional sendo que os melhores os melhores resultados foram obtidos com tamanhos de lote m = 32 ou menores [Masters e Luschi 2018].

Utilizou-se a função *categorical cross entropy* para o cálculo do valor de erro e a função de ativação *softmax* para calcular as probabilidades de pertencimento de uma amostra para as duas classes definidas. Primeiro o algoritmo de classificação foi executado aplicando o algoritmo de otimização *Adam* e posteriormente o otimizador *RMSprop*.

Para os experimentos de classificação com os modelos de redes CNN foi aplicado o *dataset* de imagens criado pelo processo de *Data augmentation*, e que foi redimensionado conforme a necessidade e configuração de cada arquitetura utilizada. Todas as amostras foram rotuladas pelo algoritmo que atribuiu o rótulo 1 para as amostras do *Diaphorina citri* e para as amostras dos outros insetos foi atribuído o rótulo 0, como mostra a Figura 3.6.

O conjunto com as amostras de desenvolvimento criado, conforme representação da Figura 3.4, foi novamente dividido para o emprego do processo de validação cruzada e apresentado na Figura 3.7. Os 70% das imagens destinadas ao conjunto de

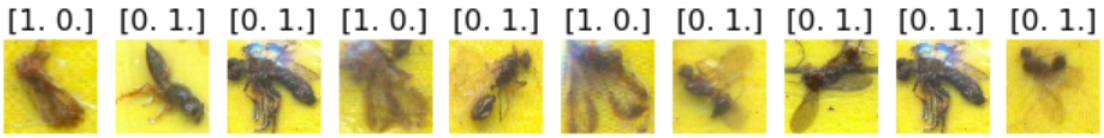

Figura 3.6: Rotulagem das imagens.

desenvolvimento foram subdivididas em três conjuntos. 90% destas imagens foram divididas em partes iguais para o conjunto de treinamento e validação e os 10% restantes das amostras foram destinadas para os testes de parâmetros do conjunto de desenvolvimento. Para a validação cruzada utilizou-se o *k-fold*, com o k=2 sendo executado 5 vezes, de acordo com proposta de Dietterich [Dietterich 1998] que sugeriu a substituição de validação cruzada de *10-fold* pelo *2-fold* x 5 com objetivo de melhorar a estabilidade do teste e como forma de reduzir o custo computacional. Os 10% restantes de amostras foram utilizadas para testes de parâmetros do conjunto de desenvolvimento.

No primeiro momento do processo de validação cruzada 2-*fold* x 5, um conjunto de amostras foi utilizado para o treinamento e outro conjunto para a validação. Em seguida os conjuntos foram trocados e o conjunto que anteriormente foi usado para o treinamento passou a ser aplicado para a validação, enquanto que o conjunto de validação passou a ser o conjunto de treinamento. Nesse processo foi utilizado um pequeno conjunto de teste para analisar o desempenho de cada classificador com imagens que não foram utilizadas durante o treinamento.

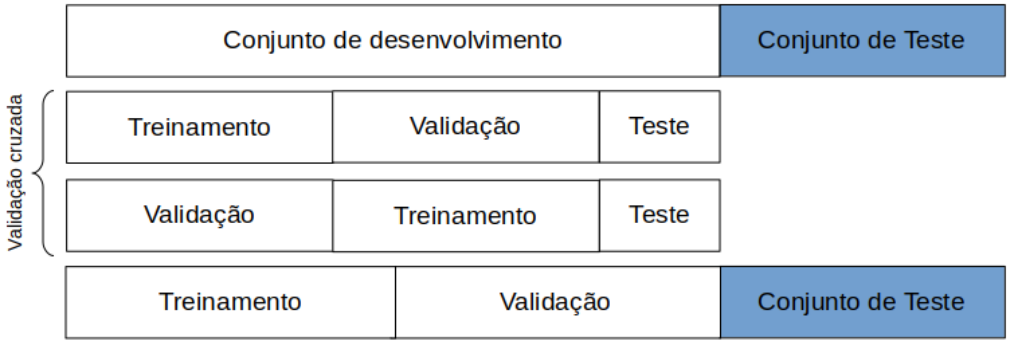

Figura 3.7: Divisão do conjunto de desenvolvimento e aplicação do *k-Fold*.

Para qualificar os resultados alcançados com os modelos propostos, foram utilizados teste de significância estatísticas, que comparam algoritmos de classificação permitindo avaliações mais precisas, sobre o valor de acurácia. Esse teste estatístico foi aplicado a todos os modelos de redes utilizadas neste trabalho. Ao final de todo esse processo de treinamento utilizando a validação cruzada, os modelos que apresentaram melhor desempenho, conforme resultados de testes estatísticos, foram executados novamente e apresentado o conjunto de teste, com amostras não utilizadas em nenhum processo anterior, e para a validação final do classificador.

3.3.1 Arquitetura *LeNet*

Foi utilizada a arquitetura *LeNet*, pioneira em aplicação de redes convolucionais em classificação de imagem, na identificação e classificação do inseto *Diaphorina citri*. Nesta rede utilizou-se o *dataset* com o redimensionamento das imagens para 32 *pixels* x 32 *pixels* devido ao tamanho padrão de imagem utilizado como entrada de dados neste modelo. A rede *LeNet* é composta de 7 camadas distribuídas, sendo que as camadas convolucionais e de *pooling* correspondem a etapa de extração de características do modelo e as camadas totalmente conectadas são responsáveis pela classificação, Figura 3.8.

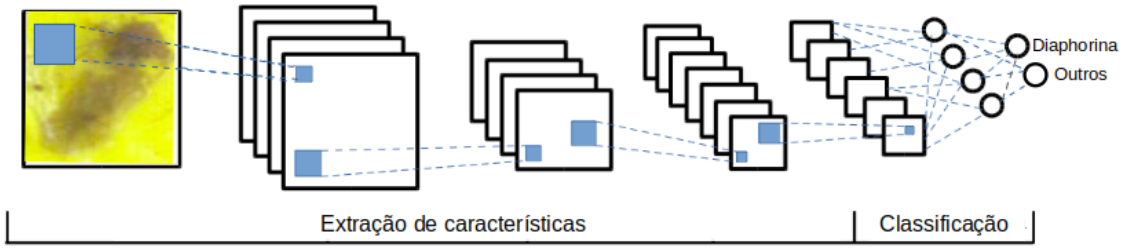

Figura 3.8: Arquitetura do modelo LeNet.
[Chollet 2017]

Conforme Tabela 3.2, entrada da rede é realizada através de uma imagem com dimensão 32 x 32 x 3, que correspondem a altura x largura e a quantidade de canais da imagem, que neste caso será 3 por usar o padrão de cores RGB. Na primeira camada convolucional (Conv 1) foram aplicados 20 filtros com a dimensão de 5 x 5. Os valores após a primeira camada convolucional correspondem a dimensionalidade do espaço de saída, nesse caso foram computados 20 filtros nos dados de entrada da rede. E que cada um desses 20 canais de saída terão um *grid* de valores de 28 x 28 que são um mapa de resposta do filtro, sendo que o número de canais é controlado pela quantidade de filtros. A aplicação da função *maxpooling* resultou na saída da camada com 14 x 14 x 20, desta forma reduzindo em 50% a dimensão da imagem.

Passadas as informação resultantes da primeira camada de *pooling* para a camada Conv 2 e executado o processo convolucional com aplicação de 50 filtros, obteve-se como saída da camada o resultado 14 x 14 x 50 e encaminhado novamente para a execução da função *maxpooling* que reduziu a dimensão dos dados para 7 x 7 x 50.

Para encaminhar os mapas de atributos extraídos pelas camadas anteriores para a camada totalmente conectada (TC), foi realizado pela camada *Flatten* o procedimento de transformação dos dados de matriz para vetor. A TC é uma camada oculta com 500 neurônios e a saída da rede com as probabilidades de pertencimento a uma classe é executada pela função de ativação *Softmax*. O modelo *LeNet* apresenta como padrão um vetor com 10 posições referentes a 10 classes, porém essa saída da rede foi modificada para duas classes correspondentes a classificação entre *Diaphorina citri* e outros insetos.

Tabela 3.2: Parâmetros do modelo LeNet.

Tipo de camada	**Saída da camada**	**Filtro**	***Stride***
Entrada	32 x 32 x 3		
Conv 1	28 x 28 x 20	20, 5 x 5	2 x 2
Pooling 1	14 x 14 x 20		2 x 2
Conv 2	14 x 14 x 50	50, 5 x 5	
Pooling 2	7 x 7 x 50		2 x 2
TC	500		
TC	2		

O número de parâmetros de uma rede convolucional está relacionado aos valores a serem aprendidos em todos os filtros nas camadas convolucional [Ponti e da Costa 2018]. No caso do modelo LeNet o total de parâmetros foi de 60.000.

3.3.2 Arquitetura AlexNet

A Alexnet foi desenvolvida por Krizhevsky [Krizhevsky et al. 2012] para reconhecimento de objetos diversos. Sua arquitetura é composta por 8 camadas, sendo 5 camadas convolucionais e 3 camadas totalmente conectadas, Figura 3.9. Para a execução deste modelo foi utilizado o *dataset* com imagens redimensionadas para 227x227 que é a dimensão de entrada para a rede AlexNet e a compilação da rede de forma sequencial em uma única GPU, portanto sem utilizar o paralelismo proposto por Krizhevsky.

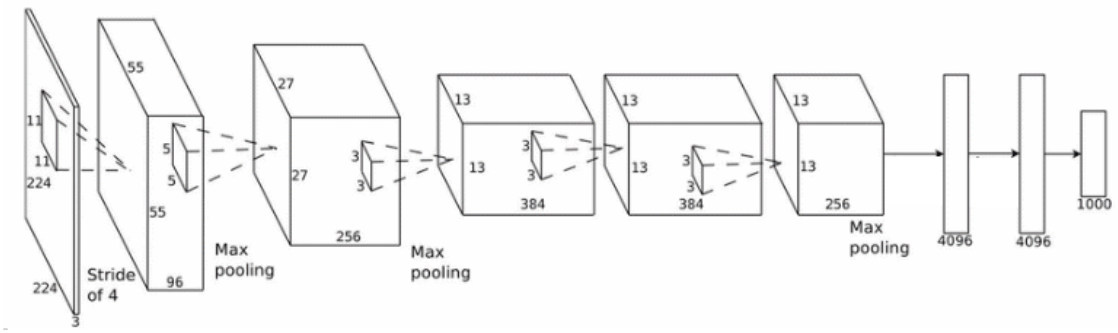

Figura 3.9: Arquitetura do modelo AlexNet.
[Krizhevsky et al. 2012]

A Tabela 3.3 apresenta as camadas e parâmetros utilizados nesta arquitetura. A primeira camada convolucional da Alexnet aplica 96 filtros na imagem de entrada com tamanho 11x11 com passo (*stride*) definido em 4, que corresponde a passagem do filtro pela imagem passando 4 *pixels* por vez, o que ocasiona a redução da di-

mensionalidade da imagem para 55 x 55 x 96. É aplicada a função *maxpooling* que irá reduzir ainda para 27 x 27 x 96 com o uso do filtro 3 x 3 com passos de 2.

A segunda camada convolucional toma como entrada a saída da primeira camada convolucional e a filtra com 256 núcleos de tamanho 5 x 5 e aplica a função *maxpooling* reduzindo a dimensão da amostra para 13 x 13 x 256. As terceira, quarta e quinta camadas convolucionais estão conectadas umas às outras com somente uma camada de *pooling* após a camada Conv 5. A terceira camada executa o processo convolucional com 384 filtros de tamanho 3 x 3 ligados às saídas da segunda camada convolucional. A quarta camada convolucional também aplica 384 filtros de tamanho 3 x 3, e a quinta camada convolucional aplica 256 filtros de tamanho 3 x 3 seguida de uma camada de *pooling*.

As camadas totalmente conectadas possuem duas camadas ocultas conectadas com 4096 neurônios cada e com uma camada de saída com a função de ativação *Softmax* com 1000 classes. A saída da função de ativação foi modificada para que se pudesse ter como resultado as duas classes *Diaphorina citri* e outros.

Tabela 3.3: Parâmetros do modelo AlexNet.

Tipo de camada	**Saída da camada**	**Filtro**	***Stride***
Entrada	227 x 227 x 3		
Conv 1	55 x 55 x 96	96, 11 x 11	4 x 4
Pooling 1	27 x 27 x 96		3 x 3
Conv 2	27 x 27 x 256	256, 1 x 1	
Pooling 2	13 x 13 x 256		2 x 2
Conv 3	13 x 13 x 384	384, 1 x 1	
Conv 4	13 x 13 x 384	384, 1 x 1	
Conv 5	13 x 13 x 256	256, 1 x 1	
Pooling 3	6 x 6 x 256		2 x 2
TC 1	4096		
TC 2	4096		
TC 3	2		

3.3.3 Arquitetura *Inception* v3

A arquitetura de rede *Inception* foi desenvolvida com base na arquitetura *GoogLeNet* vista no ILSVRC 2014 e tem com objetivo reduzir o custo computacional na classificação de imagens de forma precisa, utilizando o método de aprendizado profundo. Esta arquitetura apresenta a sequência de módulos *Inception* A, B e C resultando em 42 camadas de rede. Embora apresente uma grande quantidade de camadas, o custo de computação é apenas cerca de 2.5 maior que o da *GoogLeNet* e ainda é muito mais eficiente do que a arquitetura de rede *VGGNet* [Szegedy et al. 2015]. Foi

utilizado o *dataset* com imagens redimensionadas para 299 x 299 que é a dimensão de entrada para a rede *Inception* v3.

Tabela 3.4: Parâmetros do modelo *Inception* v3.

Tipo de camada	**Saída da camada**	**Filtro**	***Stride***
Entrada	299 x 299 x 3		
Conv 1	142 x 142 x 32	32, 3 x 3	2 x 2
Conv 2	147 x 147 x 32	32, 3 x 3	1 x 1
Pooling	1 x 1 x 2048	2048, 8 x 8	
Conv 3	71 x 71 x 80	80, 3 x 3	2 x 2
Conv 4	35 x 35 x 192	192, 3 x 3	1 x 1
3x *Incepion*	35 x 35 x 288	Módulo A	
5x *Incepion*	17 x 17 x 768	Módulo B	
2x *Incepion*	8 x 8 x 1280	Módulo C	
Pooling	56 x 56 x 256	256, 3 x 3	
TC 1	1000		
TC 2	2		

Os valores após a primeira camada Convolucional correspondem a dimensionalidade do espaço de saída, nesse caso foram computados 32 filtros nos dados de entrada da rede. E que cada um desses 32 canais de saída terão um *grid* de valores de 142 x 142 que são um mapa de resposta do filtro, como mostra a Tabela 3.4. Nesse modelo as camadas de *pooling* não correspondem a mesma quantidade de camadas convolucionais, como ocorre no modelo de arquitetura anterior e aparecerá somente após a segunda camada convolucional e também depois da aplicação do módulo *inception* C que irá reduzir a dimensão da amostra.

Outra diferença para os modelos anteriores é a utilização dos módulos *incpetion* A, B e C que tem por objetivo a melhoria do desempenho usando métodos de fatoração inteligentes para serem mais eficientes e com um reduzido custo computacional. Como apresentado na Tabela 3.4, o módulo A é aplicado 3 vezes, o módulo B é executado 5 vezes e ao final 2 vezes é executado o módulo C. O modelo possui uma camada totalmente conectada com 1000 neurônios e na saída da rede foi aplicada a função *softmax* para a classificação da rede.

O módulo *GoogleNet* empregou apenas 5 milhões de parâmetros, o que representou uma redução de 12 vezes em relação ao modelo antecessor, *AlexNet*, que usou 60 milhões de parâmetros. Além disso, o modelo *VGGNet* emprega cerca de 3x mais parâmetros que o *AlexNet* [Szegedy et al. 2016].

Capítulo 4

Resultados e Discussões

Neste capıtulo serão apresentados os resultados obtidos na classificação do *Diaphorina citri*. O primeiro experimento foi realizado aplicando a abordagem proposta por Melo [Melo 2016] e na sequência foram executados os experimentos com 3 arquiteturas distintas de redes de aprendizado profundo, conforme metodologia apresentada no capítulo anterior. Para cada experimento, as medidas de acurácia, precisão, *recall*, *F1-score* e o custo computacional, representado pelo tempo de processamento, foram analisadas e os testes estatísticos aplicados. As análises dos resultados de classificação também estarão neste capítulo.

Utilizou-se durante o desenvolvimento deste trabalho o conjunto de dados apresentado no Capítulo 3. Cada um dos modelos de arquitetura apresenta diferentes características em relação as dimensões das imagens utilizadas como entrada do algoritmo, desta forma, foi necessário realizar o redimensionamento das imagens para cada uma das arquiteturas.

Os experimentos realizados neste trabalho foram executados com as seguintes configuração de *Hardware* e *software*: Processador Intel® Core™ i7-7500U CPU @ 2.70GHz x 4; memória de 8GB, DDR4, 2400MHz; Disco rígido de 1TB (5400 RPM); Placa de vídeo *NVIDIA® GeForce® MX150* de 4GB, GDDR5; Sistema operacional Ubuntu 16.04 LTS; *PyCharm Community* 2016.3, como Ambiente de Desenvolvimento Integrado (IDE); *Python 2.7.12; Keras* 2.2.0; *TensorFlow* 1.9.0, *Jupyter Notebook*.

4.1 Experimentação utilizando a abordagem proposta por Melo [Melo 2016]

Em seu trabalho, Melo [Melo 2016] utilizou imagens adquiridas por microscopia para a criação de seu banco de imagens, sendo um processo distinto do realizado durante

a criação do banco de imagens desta pesquisa, que utilizou imagens digitalizadas diretamente das armadilhas adesivas amarelas.

Ao executar o algoritmo com os parâmetros aplicados no trabalho de Melo [Melo 2016] no *dataset* criado com imagens digitalizadas, o resultado foi de 49.95% de acurácia, sendo muito inferior ao resultado encontrado quando aplicado ao *dataset* de imagens obtidas por microscopia. Após análise do código e também da metodologia descrita no trabalho citado, foi constatada a possibilidade de alteração na quantidade do número de *clusters* (agrupamentos) que foi utilizado o valor de 700. Os *clusters* agrupam coisas similares encontradas em imagens, sendo encargo do algoritmo *Mini Batch K-Means* a formação destes agrupamentos.

No processo de redução de agrupamentos e analisando a reposta de saída, foram testados empiricamente os valores de 250, 100 e 50 *clusters* e chegou-se na medida reduzida de 50 *clusters* que apresentou 97.18% de acurácia no processo de validação cruzada *10-fold* para a classificação do *Diaphorina citri* em imagens digitalizadas, conforme Tabela 4.1, sendo este o melhor resultado de acurácia obtido.

Tabela 4.1: Resultado da classificação utilizando a metodologia proposta por Melo [Melo 2016] com validação cruzada 10-*fold*.

Clusters	**Acurácia**	**Recall**	**Precision**	**F1-Score**	**Validação Cruzada**
50	97.99%	98.80%	97.23%	98.01%	97.18%
100	80.40%	99.80%	71.92%	83.60%	80.91%
250	51.56%	100%	50.82%	67.39%	51.12%

Definindo-se a quantidade de agrupamentos em 50 *clusters*, o algoritmo de classificação foi novamente executado e desta vez sendo aplicado o procedimento proposto de validação cruzada 2-*fold* x 5.

Tabela 4.2: Resultado da classificação utilizando a metodologia proposta por Melo [Melo 2016] com validação cruzada 2-*fold* x 5.

Execução	**Validação Cruzada**	**TempExc**
1	94.05% (+/- 0.16%)	6.25 min
2	95.67% (+/- 0.9%)	6.10 min
3	94.49% (+/- 0.19%)	6 min
4	95.02% (+/- 0.08%)	6.40 min
5	95.53% (+/- 0.11%)	6.25 min

As Tabelas 4.2 e 4.3 apresentam os valores resultantes da classificação do *Diaphorina citri* com a proposta de abordagem de Melo [Melo 2016] modificando-se a quantidade de *clusters* e o processo de validação cruzada. Na coluna Execução em ambas tabelas estão as 5 execuções ordenadas do algoritmo de classificação e que apresentaram a média de 6.20 min de duração para cada execução, conforme apresenta a coluna

TempExc. Os valores de acurácia da Tabela 4.3 do processo de generalização com validação 2-*fold* x 5 apresentaram proximidade com o valor de acurácia da Tabela 4.1 com a validação *10-fold*.

Tabela 4.3: Resultados da generalização da metodologia proposta por Melo [Melo 2016] utilizando SVM.

	SVM			
Execução	**Acurácia**	**Recall**	**Precisão**	**F1-Score**
1	97.99%	98.80%	97.23%	98.01%
2	97.21%	97.97%	96.41%	97.18%
3	97.57%	98.36%	96.78%	97.57%
4	98.18%	98.00%	98.39%	98.19%
5	98.07%	98.19%	97.99%	98.09%

4.2 Experimentação da Arquitetura *LeNet*

Os experimentos com o modelo de rede *LeNet* foram feitos utilizando valores de parâmetros semelhantes ao aplicados aos outros modelos. Para esta arquitetura o *dataset* de imagens foi redimensionado para 32 x 32 x 3, dimensão padrão de entrada de dados do modelo. A Tabela 4.4 apresenta os valores de acurácia do processo de validação cruzada 2-*fold* x 5. Na primeira coluna está a ordem em que as 5 execuções do algoritmo foram realizadas. Os valores da acurácia da validação cruzada (AValC) foram calculados como a média de cada execução do algoritmo juntamente com o valor de desvio padrão. A coluna (TempExc) corresponde ao tempo de duração de cada execução do procedimento de validação cruzada do algoritmo.

Tabela 4.4: Resultados gerados após treinamento do modelo *LeNet*.

	Adam		**RMSprop**	
Execução	**AValC**	**TempExc**	**AValC**	**TempExc**
1	96.86% (+/- 0.64%)	36.78 min	96.43 % (+/- 0.27%)	39.02 min
2	96.69% (+/- 0.17%)	36.67 min	96.86% (+/- 0.51%)	38.42 min
3	97.30% (+/- 0.09%)	41.76 min	96.60% (+/- 0.49%)	37.47 min
4	95.87% (+/- 0.87%)	36.68 min	96.48% (+/- 0.20%)	36.69 min
5	96.60% (+/- 0.67%)	42.57 min	96.69% (+/- 0.75)	36.61 min

As Figuras 4.1 e 4.2 apresentam os gráficos com o desvio padrão em função da acurácia encontrada com o processo de validação cruzada 2-*fold* x 5 e realizado para o algoritmo, utilizando os otimizadores *Adam* e *RMSprop*. Os valores plotados no gráfico foram ordenados do maior valor de acurácia para o menor. No eixo x estão representados os valores de acurácia, enquanto que no eixo y o intervalo dos valores

calculados de desvio padrão. Os valores de desvio padrão apresentados revelam uma pequena dispersão dos resultados variando de 0.09% a 0.87%.

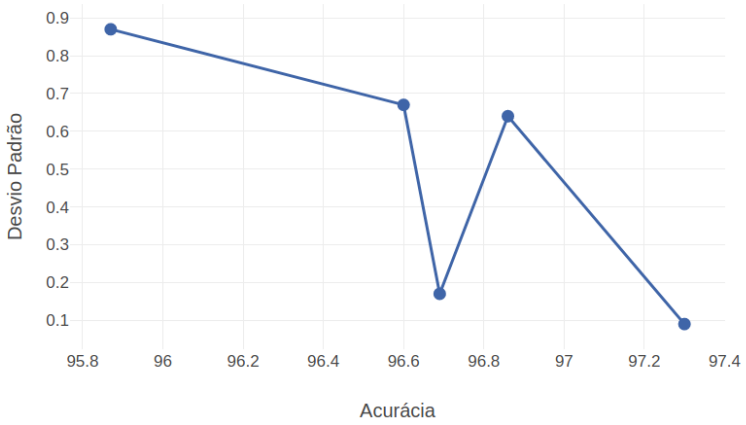

Figura 4.1: Gráfico do desvio padrão em função da acurácia com o otimizador *Adam*.

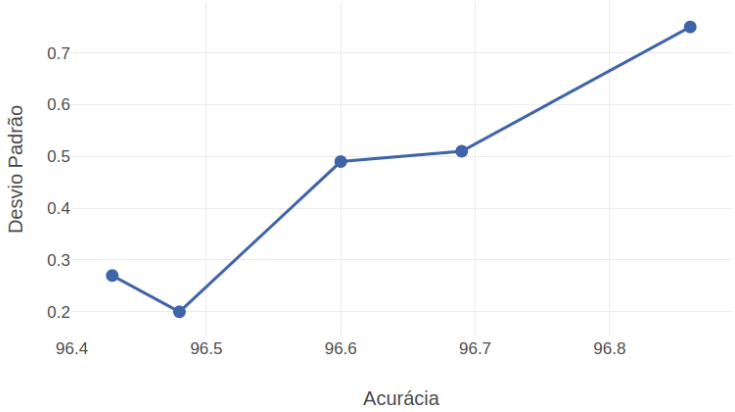

Figura 4.2: Gráfico do desvio padrão em função da acurácia com o otimizador *RMSprop*.

Após a obtenção dos valores de validação de cada execução do processo de validação cruzada, foi apresentado ao algoritmo o conjunto de amostras não utilizado no processo de treinamento e que foi reservado para os testes de ajustes do modelo e escolha do melhor modelo. Para cada uma das 5 execuções do 2-*fold* foram obtidos os valores de acurácia, *recall*, precisão e *f1-score*. As Tabelas 4.5 e 4.6 apresentam os resultados de generalização do modelo com suas medidas de avaliação para o conjunto de amostras de teste de parâmetros, com a utilização dos otimizadores *Adam* e *RMSprop*.

Comparando-se os valores expostos nas Tabelas 4.4, 4.5 e 4.6 é possível detectar a proximidade dos valores de acurácia obtidos por processo de validação cruzada,

Tabela 4.5: Resultados da generalização do modelo *LeNet* com amostras do conjunto reservado para ajuste de parâmetros, utilizando o otimizador *Adam*.

	Adam			
Execução	**Acurácia**	**Recall**	**Precisão**	**F1-Score**
1	97.89%	99.67%	96.24%	97.92%
2	97.64%	100%	95.50%	97.70%
3	97.93%	99.83%	96.17%	97.97%
4	97.56%	99.91%	95.42%	97.62%
5	97.93%	99.67%	96.31%	97.96%

Tabela 4.6: Resultados da generalização do modelo *LeNet* com amostras do conjunto reservado para ajuste de parâmetros, utilizando o otimizador *RMSprop*.

	RMSprop			
Execução	**Acurácia**	**Recall**	**Precisão**	**F1-Score**
1	97.32%	99.91%	94.98%	97.39%
2	98.41%	100%	96.93%	98.44%
3	98.21%	99.50%	96.91%	98.19%
4	98.13%	99.91%	96.47%	98.16%
5	97.60%	99.91%	95.50%	97.96%

com os dados de treinamento, e os resultados de generalização do modelo com as imagens do conjunto de ajustes. Desta forma, é possível evidenciar que a rede teve desempenho próximo nos dois casos, eliminando a possibilidade de superajustamento do modelo, mesmo não apresentando estratégias de regularização.

4.3 Experimentação da Arquitetura *AlexNet*

Os experimentos realizados com o modelo *AlexNet* foram com os valores de parâmetros já definidos para todas as arquiteturas do trabalho, conforme apresentado na seção 3.2. A Tabela 4.7 apresenta os valores de acurácia do processo de validação cruzada 2-*fold* x 5. Na primeira coluna está a ordem em que as 5 execuções do algoritmo foram realizadas. Os valores da acurácia da validação cruzada (AValC) foram calculados através da média de cada execução do algoritmo e está apresentada juntamente com o valor de desvio padrão. O tempo de execução (TempExc) do algoritmo apresentou significativo aumento no treinamento da rede, comparando-se com os tempos do modelo *LeNet*, com uma média de 12 horas de execução de cada 2-*fold* x 5, porém justificável, já que no modelo anterior a quantidade de parâmetros a serem aprendidos pelos filtros convolucionais foram de 60 mil, enquanto que no modelo *AlexNet*, devido a uma maior quantidade de camadas convolucionais, a quantia de parâmetros foi de 60 milhões, aumentando o custo computacional.

Tabela 4.7: Resultados gerados após treinamento do modelo AlexNet.

	Adam		RMSprop	
Execução	**AValC**	**TempExc**	**AValC**	**TempExc**
1	97.62% (+/- 0.09%)	12:50 hrs	98.53 % (+/- 0.24%)	12 hrs
2	97.62% (+/- 0.15%)	12:55 hrs	97.33% (+/- 0.26%)	12:07 hrs
3	98.09% (+/- 0.13%)	12:58 hrs	97.47% (+/- 0.12%)	11:53 hrs
4	97.80% (+/- 0.27%)	12:47 hrs	98.00% (+/- 0.24%)	12:15 hrs
5	97.41% (+/- 0.42%)	12:50 hrs	98.13% (+/- 0.19%)	12:04 hrs

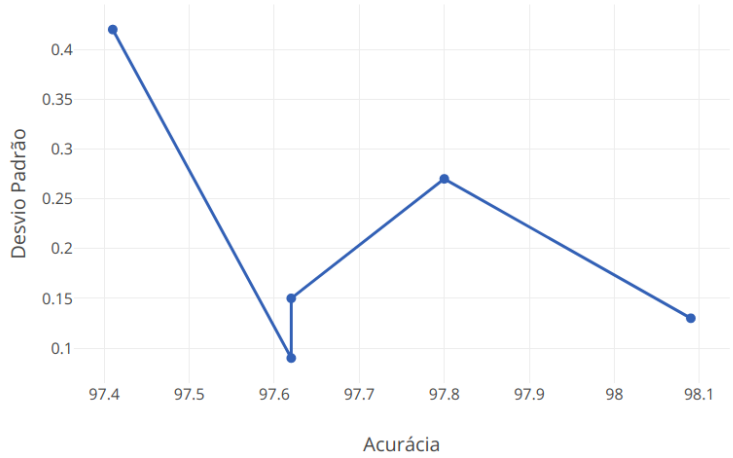

Figura 4.3: Gráfico do desvio padrão em função da acurácia com o otimizador *Adam*.

Tabela 4.8: Resultados da generalização do modelo com amostras do conjunto reservado para ajuste de parâmetros.

	Adam			
Execução	**Acurácia**	**Recall**	**Precisão**	**F1-Score**
1	96.40%	94.52%	98.22%	96.33%
2	80.47%	65.06%	94.05%	76.92%
3	95.71%	99.65%	92.38%	95.88%
4	97.95%	98.28%	97.60%	97.94%
5	97.09%	96.30%	97.95%	97.11%

As Tabelas 4.8 e 4.9 apresentam os resultados de generalização do modelo com suas medidas de avaliação para o conjunto de amostras de teste de parâmetros, com a aplicação dos otimizadores *Adam* e *RMSprop*. Os resultados de acurácia apresentaram valores de acertos próximos para os dois otimizadores, porém pode-se destacar, de maneira negativa, a segunda execução do processo de validação cruzada com o otimizador *Adam*, que apresentou 80.47% de acertos na classificação, 120 insetos da classe Outros foram indicados de forma errônea como sendo da classe

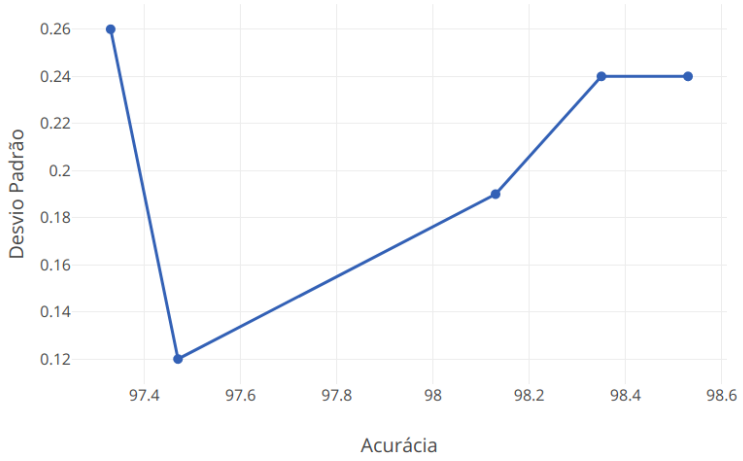

Figura 4.4: Gráfico do desvio padrão em função da acurácia com o otimizador *RMSprop*.

Tabela 4.9: Resultados da generalização do modelo com amostras do conjunto reservado para ajuste de parâmetros.

	RMSprop			
Execução	**Acurácia**	**Recall**	**Precisão**	**F1-Score**
1	98.63%	99.31%	97.95%	98.632%
2	97.95%	97.30%	98.63%	97.96%
3	97.43%	96.91%	97.92%	97.41%
4	97.77%	97.94%	97.60%	97.77%
5	98.12%	97.95%	98.29%	98.12%

dos *Diaphorina* e 12 psilídeos foram classificados como Outros, reduzindo assim o indicador de acurácia, conforme Tabela 4.10.

Os valores presentes na Tabela 4.9 com o otimizador *RMSprop* indicam uma redução de amostras classificadas incorretamentes em relação ao valores com a otimização *Adam* e também em relação aos dados da rede *LeNet*, o que indica uma melhoria na qualidade da classificação com o uso do modelo *AlexNet*.

Tabela 4.10: Matriz de confusão da segunda execução de validação cruzada com o otimizador *Adam*.

	Diaphorina	**Outros**
Diaphorina	280	12
Outros	102	190

4.4 Experimentação da Arquitetura *Inception*

Nesta seção foram realizados experimentos com o modelo *Inception*. Esta arquitetura da *GoogLeNet* foi projetada para ter um bom funcionamento mesmo sob restrição de memória e de processamento, pois emprega 5 milhões de parâmetros representando uma redução de 12 vezes a quantidade de parâmtros utilizados pela *AlexNet*. A Tabela 4.11 mostra os valores de acurácia do processo de validação cruzada 2-*fold* x 5. Na primeira coluna está a ordem de execução do algoritmo de validação cruzada e na coluna seguinte os valores de acurácia deste processo (AValC) juntamente com o valor de desvio padrão. Na última coluna estão os tempos de execução (TempExc) do algoritmo.

Tabela 4.11: Resultados gerados após treinamento do modelo *Inception*.

	Adam		**RMSprop**	
Execução	**AValC**	**TempExc**	**AValC**	**TempExc**
1	99.65% (+/- 0.12%)	26:06 hrs	99.82% (+/- 0.06%)	25:56 hrs
2	99.80% (+/- 0.08%)	26:00 hrs	98.76% (+/- 0.10%)	26:16 hrs
3	99.78% (+/- 0.10%)	25:57 hrs	97.47% (+/- 0.12%)	26:01 hrs
4	99.84% (+/- 0.05%)	26:03 hrs	98.35% (+/- 0.24%)	26:00 hrs
5	99.56% (+/- 0.15%)	26:15 hrs	98.13% (+/- 0.19%)	26:06 hrs

Um ponto que chama atenção em relação aos dados da Tabela 4.11 é o tempo de execução do algoritmo que apresentou em média 26 horas de execução, aumento significativo em relação aos modelos *LeNet* e *AlexNet*, apresentados anteriormente.

Porém, apesar do aumento do custo computacional o modelo *Inception* apresentou números importantes em relação a taxa de acurácia tanto no processo de validação cruzada como também na generalização do modelo ao ser apresentado dados que não haviam sido utilizados durante o processo de treinamento.

Tabela 4.12: Resultados da generalização do modelo com amostras do conjunto reservado para ajuste de parâmetros.

	Adam			
Execução	**Acurácia**	**Recall**	**Precisão**	**F1-Score**
1	99.48%	99.31%	99.65%	99.48%
2	99.66%	99.66%	99.66%	99.66%
3	99.48%	99.31%	99.65%	99.48%
4	99.66%	99.32%	100%	99.66%
5	99.14%	98.98%	99.32%	99.15%

As Tabelas 4.12 e 4.13 apresentam os resultados de generalização do modelo *Inception* com suas medidas de avaliação para o conjunto de amostras de teste de

Tabela 4.13: Resultados da generalização do modelo com amostras do conjunto reservado para ajuste de parâmetros.

	RMSprop			
Execução	**Acurácia**	**Recall**	**Precisão**	**F1-Score**
1	99.82%	100 %	99.65%	99.82%
2	99.66%	100 %	99.32%	99.66%
3	99.49%	99.66%	99.32%	99.49%
4	99.49%	98.98%	100%	99.49%
5	99.82%	100%	99.65%	99.82%

parâmetros, com a aplicação dos otimizadores *Adam* e *RMSprop*. Os resultados de acurácia apresentaram valores de acertos próximos para os dois otimizadores, porém com o maior valor de acurácia sendo apresentado com o otimizador *RMSprop* na primeira e na quinta execução do processo de validação cruzada com 99.82%, consequentemente apresentando poucos erros na classificação do *Diaphorina citri*, conforme apresentado na Tabela 4.10 com a matriz de confusão.

Tabela 4.14: Matriz de confusão da primeira e quinta execução da validação cruzada com o otimizador *RMSprop*.

	Diaphorina	**Outros**
Diaphorina	291	1
Outros	0	292

4.4.1 Discussão dos Resultados

Neste trabalho foi aplicado o método de aprendizagem profunda através de modelos de Redes Convolucionais Neurais (CNN) distintos para a classificação do inseto *Diaphorina citri* em imagens digitalizadas de armadilhas adesivas amarelas, com o objetivo de comparar os resultados obtidos por cada modelo empregado e aferir qual arquitetura apresenta melhores resultados na classificação do psilídeo.

Para avaliar a qualidade dos dados de classificação apresentados neste trabalho foram considerados os resultados atingidos com o uso da metodologia de Melo [Melo 2016] com o SVM (*Support Vector Machine*) juntamente com o *dataset* criado neste trabalho. Com a alteração de parâmetro como a redução na quantidade de bolsas de características de 700 para 50 *clusters* obteve-se com esta metodologia um resultado de 97.18% de validação cruzada *10-fold* e de 97.99% de acurácia na generalização. Com a definição do valor de *clusters* em 50, a metodologia de Melo [Melo 2016] foi novamente executada, porém utilizando a validação cruzada *2-fold* x 5, e apresentou a média de 97.80% de acurácia, conforme valores presentes na Tabela 4.3. Desta forma, os valores obtidos pela metodologia empregada por Melo [Melo 2016]

foram usados para comparar os valores de classificação encontrados empregando a metodologia de aprendizado profundo.

Foram aplicados os modelos de redes neurais convolucionais *LeNet*, *AlexNet* e *Inception*. Para avaliar os resultados obtidos, foi aplicado o teste não paramétrico de *Friedman* para comparar múltiplos algoritmos em vários domínios. O resultado do teste apresentou o *p-value* de 0.04057, sendo este valor significante já que o *p-value* é menor que o nível de confiança estabelecido em 0.05. Portanto, é rejeitada a hipótese nula H_0 que afirma serem iguais os classificadores, não apresentado diferença significativa em seus resultados.

Ao ser rejeitada a hipótese nula faz-se necessário a aplicação de um teste *Post Hoc* para o teste de *Friedman*, este teste compara um a um os algoritmos e determina onde está a diferença entre eles. Para o teste *Post Hoc* pode ser usado o teste *Nemenyi* [Japkowicz e Shah 2011]. As Tabelas 4.15 e 4.16 apresentam o resultado do teste de *Nemenyi* que compara em pares os resultados dos experimentos com o objetivo de apresentar qual modelo obteve melhor resultado na classificação do *Diaphorina citri*. Na linha superior e primeira coluna estão os nomes dos modelos executados com os indicadores 1 para o uso do otimizador *Adam* e 2 para o otimizador *RMSprop*.

Para a Tabela 4.15 foram utilizados os valores de acurácia resultantes dos classificadores. Foram destacados em negrito os valores do teste estatístico menores que o nível de confiança 0.05, que apresentaram diferença significante rejeitando-se a hipótese nula de igualdade entre os classificadores. O modelo *Inception* com os otimizadores *Adam* e *RMSprop*, representados na tabela como *Incep 1* e *Incep 2*, apresentou diferença significativa ao ser comparado com os resultados do modelo *LeNet*, modelo *AlexNet* e com os resultados obtidos com a metodologia proposta por Melo [Melo 2016] utilizando SVM.

Tabela 4.15: Resultado do teste de *Nemenyi* com os valores de acurácia.

	LeNet 1	**LeNet 2**	**Alex 1**	**Alex 2**	**Incep 1**	**Incep 2**
LeNet 2	1.000000					
Alex 1	0.661201	0.098096				
Alex 2	1.000000	1.000000	0.098096			
Incep 1	**0.002483**	**0.026245**	**0.000030**	**0.026245**		
Incep 2	**0.000704**	**0.008724**	**0.000009**	**0.008724**	1.00000	
SVM	1.00000	1.00000	0.098096	1.00000	**0.026245**	**0.008724**

Observa-se que na Tabela 4.16 foram utilizados os valores de precisão, que é quantidade de previsões feitas que são realmente corretas ou relevantes de todas as previsões baseadas na classe positiva, extraídos dos modelos e em sua maioria apresentaram diferenças significativas do modelo *Inception* com os otimizadores *Adam* e *RMSprop*, representados na tabela como *Incep 1* e *Incep 2*, com os outros modelos aplicados. Porém, não apresentando diferença significativa em relação ao modelo de rede *Alex-*

Tabela 4.16: Resultado do teste de *Nemenyi* com os valores de precisão.

	LeNet 1	**LeNet 2**	**Alex 1**	**Alex 2**	**Incep 1**	**Incep 2**
LeNet 2	1.000000					
Alex 1	1.000000	1.000000				
Alex 2	**0.008805**	0.068725	0.141233			
Incep 1	**0.000008**	**0.000055**	**0.000143**	0.068725		
Incep 2	**0.000014**	**0.000106**	**0.000278**	0.112824	1.000000	
SVM	0.141233	0.670793	1.000000	1.00000	**0.003390**	**0.006584**

Net utilizando o otimizador *RMSprop*, conforme valores significantes destacados em negrito.

Com a aplicação do teste de *Nemenyi*, o modelo *AlexNet* com o otimizador *RMSprop* não apresentou diferença significativa com o modelo *Inception*. É importante salientar que o modelo *AlexNet* classificou em média 5.4 *Diaphorina citri* de forma errônea nas execuções de validação cruzada, enquanto que o modelo *Inception* teve média de erro de 1 *Diaphorina citri* classificado errado. Portanto, mesmo que o teste estatístico não rejeite o H_0, a hipótese H_0 pode ser falsa enquadrando-se no erro Tipo II, já que o modelo *Inception* apresentou menor erro ao classificar o *Diaphorina* e considerando os resultados da Tabela 4.15.

Outro ponto importante a ser observado nos resultados é o tempo de execução, dos modelos de rede utilizados, em cada processo de validação cruzada 2-fold x 5. O crescente aumento de tempo tem relação a quanto o valor de filtros aplicados e número de camadas de cada arquitetura. Desta forma, arquitetura *LeNet* apresentou o tempo médio de execução de 37.6 minutos, enquanto que as arquiteturas *AlexNet* e *Inception* apresentaram respectivamente a média de tempo, em cada execução da validação cruzada, de 12 horas e 26 horas.

Apesar do elevado tempo de execução do algoritmo, a arquitetura *Inception* apresentou o melhor resultado na classificação do psilídeo, conforme os dados das Tabelas 4.12 e 4.13, com quase 100% de acerto nas classificação correta do psilídeo, durante o processo de validação cruzada, e baixíssima quantidade de falsos positivos. Portanto, demonstrando maior confiabilidade na classificação correta do inseto *Dipahorina citri*, foi selecionado para a execução final a arquitetura *Inception*. Para esta execução, foram utilizadas as amostras de imagens do conjunto de teste, que ainda não haviam sido utilizadas em nenhum procedimento anterior deste trabalho.

Ao aplicar o conjunto com amostras de teste ao modelo *Inception* foi obtida uma média de acurácia de 99.51% na validação cruzada e 99.37% na validação como resultado da generalização do modelo com o conjunto de teste final, conforme Tabelas 4.17 e 4.18.

Tabela 4.17: Matriz de confusão da segunda execução de validação cruzada com o otimizador *Adam*.

Execução	**Acurácia**	**TempExc**
1	99.28% (+/- 0.08%)	26 hrs
2	99.48% (+/- 0.11%)	25:58 hrs
3	99.64% (+/- 0.13%)	26:05 hrs
4	99.48% (+/- 0.09%)	26 hrs
5	99.68% (+/- 0.27%)	26:03 hrs

Tabela 4.18: Resultados da generalização do modelo com o conjunto de amostras de teste.

	Inception			
Execução	**Acurácia**	**Recall**	**Precisão**	**F1-Score**
1	99.12%	99.44%	98.80%	99.12%
2	99.32%	99.60%	99.04%	99.32%
3	99.56%	99.68%	99.44%	99.56%
4	99.32%	99.60%	99.04%	99.32%
5	99.44%	99.60%	99.28%	99.44%

Capítulo 5

Considerações Finais

Esta pesquisa teve como objetivo geral o uso da metodologia de Aprendizado Profundo (*Deep Learning*) com Redes Convolucionais Neurais (CNN) na classificação do *Diaphorina citri* em imagens digitalizadas de armadilhas adesivas amarelas.

Para a criação de um *dataset* de imagens do *Diaphorina citri* foi realizado o recorte manual do inseto, tendo ao final 1102 amostras, divididas em partes iguais para o inseto alvo da pesquisa e outros insetos presentes nas armadilhas. Foi utilizado o recurso de *Data Augment*, algoritmo de incrementação da base de imagens que aumentou o *dataset* inicial para 8354 imagens, também dividas em partes iguais. Para a validação foram utilizadas informações de quantidade e localização do inseto em cada armadilha adesiva, passadas pelo FUNDECITRUS.

Para a efetiva classificação com aprendizagem profunda, foram experimentadas três arquiteturas distintas de redes convolucionais. Devido às diferentes quantidade de camadas convolucionais e camadas totalmente conectadas, as redes apresentam quantidade distintas de parâmetros a serem aprendidos, o que tem como resultado diferentes tempos de processamento dos algoritmos e melhor qualidade de resultados. O modelo de rede *Inception* apresentou melhores valores de acurácia e menor quantidade de erros na classificação do *Diaphorina citri* e por esse motivo foi aplicado a este modelo o teste final de generalização que resultou em uma média de acurácia de 99.51% na validação cruzada e 99.37% na validação do modelo com o conjunto de teste final.

Baseando-se na metodologia proposta por Melo [Melo 2016], que utilizou o mesmo inseto, porém com o aprendizado de máquina SVM, foram realizados experimentos utilizando o *dataset* criado com imagens digitalizadas. Com a redução de *clusters* para 50 agrupamentos, o algoritmo alcançou 97,18% de acurácia na validação cruzada e 97.99% de acurácia no teste de generalização do modelo. A metodologia utilizada com aprendizado profundo apresentou resultados de acurácia na classificação do *Diaphorina citri* superiores aos resultados da metodologia acima citada.

As contribuições alcançadas nesta dissertação abrem a possibilidade de continuidade

do trabalho e também no desenvolvimento de uma aplicação que identifique e classifique o inseto *Diaphorina citri* de forma automatizada utilizando aprendizagem profunda.

5.1 Pesquisas Futuras

Propõe-se na continuidade desta pesquisa o desenvolvimento de um algoritmo de processamento de imagens para a extração de amostras dos insetos contidos nas armadilhas adesivas, desta maneira automatizando a criação do *dataset* e a obtenção de mais amostras das armadilhas adesivas para o incremento da base de imagens.

Também poderão ser aprimorados os algoritmos das redes convolucionais neurais (CNN), utilizando outros parâmetros de otimização, objetivando a melhoria da acurácia e da outras métrica de avaliação apresentadas neste trabalho de pesquisa. Poderão também ser experimentadas outros modelos de CNN existentes e com capacidade de aprender com o volume de dados apresentados, ou outras metodologias de aprendizado de máquina para classificação.

Referências Bibliográficas

[Bassanezi et al. 2006] Bassanezi, R., Bergamin Filho, A., Amorim, L., e Gottwald, T. (2006). Epidemiology of huanglongbing in são paulo. In *Proceedings of Huanglongbing Greening International Workshop, Ribeirão Preto*, pp. 37.

[Belasque Junior et al. 2009] Belasque Junior, J., Bergamin Filho, A., Bassanezi, R. B., Barbosa, J. C., Fernandes, N. G., Yamamoto, P. T., Lopes, S. A., Machado, M. A., Leite Junior, R. P., Ayres, A. J., et al. (2009). Base científica para a erradicação de plantas sintomáticas e assintomáticas de huanglongbing (hlb, greening) visando o controle efetivo da doença. *Tropical Plant Pathology*, 34(3):137–145.

[Bové 2006] Bové, J. M. (2006). Huanglongbing: a destructive, newly-emerging, century-old disease of citrus. *Journal of plant pathology*, pp. 7–37.

[Brasil 2008] Brasil (2008). Instrução normativa n. 53, de 16 de outubro de 2008). *Diário Oficial da União, Ministério da Agricultura, Pecuária e Abastecimento*.

[Brasil 2011] Brasil (2011). Agenda estratégica citricultura 2010-2015). *Ministério da Agricultura, Pecuária e Abastecimento*.

[Brasil 2015] Brasil (2015). Projeções do agronegócio, brasil 2014/15 a 2024/25 – projeções de longo prazo. *Ministério da Agricultura, Pecuária e Abastecimento*.

[Brlansky et al. 2012] Brlansky, R., Chung, K., e Rogers, M. (2012). 2006 florida citrus pest management guide: Huanglongbing (citrus greening). *UF/IFAS Extension*.

[Catling et al. 1970] Catling, H. et al. (1970). Distribution of the psyllid vectors of citrus greening disease, with notes on the biology and bionomics of diaphorina citri. *FAO Plant Protection Bulletin*, 18(1):8–15.

[Chollet 2016] Chollet, F. (2016). Keras documentation.

[Chollet 2017] Chollet, F. (2017). *Deep learning with python*. Manning Publications Co.

[Deng et al. 2018] Deng, L., Wang, Y., Han, Z., e Yu, R. (2018). Research on insect pest image detection and recognition based on bio-inspired methods. *Biosystems Engineering*, 169:139–148.

[Deng et al. 2014] Deng, L., Yu, D., et al. (2014). Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387.

[Dietterich 1998] Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923.

[Ding e Taylor 2016] Ding, W. e Taylor, G. (2016). Automatic moth detection from trap images for pest management. *Computers and Electronics in Agriculture*, 123:17–28.

[Dumoulin e Visin 2016] Dumoulin, V. e Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

[Fawcett 2006] Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874.

[Fundecitrus 2016] Fundecitrus (2016). *Greening - medidas essencias de controle*.

[Gallo et al. 2002] Gallo, D., Nakano, O., Silveira Neto, S., Carvalho, R. P. L., e Baptista, G. C. d. (2002). *Entomologia agrícola*. Number 632.7 E61e. Fundacao de Estudos Agrários Luiz de Queiroz,.

[Géron 2017] Géron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. "O'Reilly Media, Inc.".

[Girardi et al. 2011] Girardi, E., do Nascimento, A., Barbosa, F. F. L., de Andrade, E., Astua, J. d. F., Barbosa, C. d. J., Sanches, N., Stuchi, E., Fancelli, M., Santos Filho, H., et al. (2011). Guia de identificação do huanglongbing (hlb, ex-greening) dos citros. *Embrapa Mandioca e Fruticultura-Fôlder/Folheto/Cartilha (INFOTECA-E)*.

[Goodfellow et al. 2016] Goodfellow, I., Bengio, Y., Courville, A., e Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

[Gottwald et al. 2007] Gottwald, T. R., da Graça, J. V., Bassanezi, R. B., et al. (2007). Citrus huanglongbing: the pathogen and its impact. *Plant Health Progress*, 6(1):1–18.

[Hackeling 2017] Hackeling, G. (2017). *Mastering Machine Learning with scikit-learn*. Packt Publishing Ltd.

[Halbert e Manjunath 2004] Halbert, S. E. e Manjunath, K. L. (2004). Asian citrus psyllids (sternorrhyncha: Psyllidae) and greening disease of citrus: a literature review and assessment of risk in florida. *Florida entomologist*, 87(3):330–353.

[Hall 2009] Hall, D. G. (2009). An assessment of yellow sticky card traps as indicators of the abundance of adult diaphorina citri (hemiptera: Psyllidae) in citrus. *Journal of Economic Entomology*, 102(1):446–452.

[Harrington 2012] Harrington, P. (2012). Machine learning in action. *Shelter Island, NY: Manning Publications Co.*

[Haykin 2007] Haykin, S. (2007). *Redes neurais: princípios e prática.* Bookman Editora.

[Haykin et al. 2009] Haykin, S. S., Haykin, S. S., Haykin, S. S., e Haykin, S. S. (2009). *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:.

[Japkowicz e Shah 2011] Japkowicz, N. e Shah, M. (2011). *Evaluating learning algorithms: a classification perspective.* Cambridge University Press.

[Johnson 1999] Johnson, D. H. (1999). The insignificance of statistical significance testing. *The journal of wildlife management*, pp. 763–772.

[Joshi 2017] Joshi, P. (2017). *Artificial Intelligence with Python.* Packt Publishing.

[Karpathy 2016] Karpathy, A. (2016). Cs231n convolutional neural networks for visual recognition. *Neural networks*, 1.

[Karpathy 2018] Karpathy, A. (2018). Convolutional neural networks for visual recognition.

[Kingma e Ba 2014] Kingma, D. P. e Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[Kotsiantis et al. 2007] Kotsiantis, S. B., Zaharakis, I., e Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24.

[Krizhevsky et al. 2012] Krizhevsky, A., Sutskever, I., e Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105.

[LeCun et al. 2015] LeCun, Y., Bengio, Y., e Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

[LeCun et al. 1998] LeCun, Y., Bottou, L., Bengio, Y., e Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Leonardo 2014] Leonardo, A. (2014). *Otimização da leitura de cartão adesivo amarelo para o monitoramento de adultos de Diaphorina citri Kuwayama (Hemiptera: Liviidae) 28 f.* PhD thesis, Dissertação de Mestrado. Araraquara, SP: Fundo de Defesa da Citricultura.

[Liu et al. 2016] Liu, Z., Gao, J., Yang, G., Zhang, H., e He, Y. (2016). Localization and classification of paddy field pests using a saliency map and deep convolutional neural network. *Scientific reports*, 6:20410.

[Marques et al. 2018] Marques, A. C. R., Raimundo, M. M., Cavalheiro, E. M. B., Salles, L. F., Lyra, C., e Von Zuben, F. J. (2018). Ant genera identification using an ensemble of convolutional neural networks. *PloS one*, 13(1):e0192011.

[Masters e Luschi 2018] Masters, D. e Luschi, C. (2018). Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*.

[Melo 2016] Melo, J. L. d. S. (2016). Classificação automática do diaphorina citri em imagens de microscopia. Master's thesis, Dissertação de Mestrado, Universidade Estadual de Feira de Santana, Feira de Santana.

[Miranda et al. 2011] Miranda, M., Yamamoto, P., e Noronha JR, N. d. (2011). Utilização de cartões adesivos para monitoramento de diaphorina citri. *Citricultura Atual*, 81:8–9.

[Mitchell et al. 1997] Mitchell, T. M. et al. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877.

[Monard e Baranauskas 2003] Monard, M. C. e Baranauskas, J. A. (2003). Conceitos sobre aprendizado de máquina. *Sistemas Inteligentes-Fundamentos e Aplicações*, 1(1):32.

[Nava et al. 2007] Nava, D., Torres, M., Rodrigues, M., Bento, J., e Parra, J. (2007). Biology of diaphorina citri (hem., psyllidae) on different hosts and at different temperatures. *Journal of Applied Entomology*, 131(9-10):709–715.

[Neves et al. 2010] Neves, M. F., Trombin, V. G., Milan, P., Lopes, F. F., Cressoni, F., e Kalaki, R. (2010). O retrato da citricultura brasileira. *Ribeirão Preto: CitrusBR*.

[Parker 2010] Parker, J. R. (2010). *Algorithms for image processing and computer vision*. John Wiley & Sons.

[Parra et al. 2010] Parra, J. R. P., Lopes, J. R. S., Torres, M. G., Nava, D. E., e Paiva, P. E. B. (2010). Bioecologia do vetor diaphorina citri e transmissão de bactérias associadas ao huanglongbing. *Citrus Research and Technology*, 31(1):37–51.

[Perez e Wang 2017] Perez, L. e Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*.

[Ponti e da Costa 2018] Ponti, M. A. e da Costa, G. B. P. (2018). Como funciona o deep learning. *arXiv preprint arXiv:1806.07908*.

[Russell e Norvig 2013] Russell, S. e Norvig, P. (2013). *Inteligência Artificial. 3a Edição*. Editora Elsevier.

[Samuel 1959] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229.

[Sarkar et al. 2018] Sarkar, D., Bali, R., e Sharma, T. (2018). *Practical Machine Learning with Python*. Springer.

[Shen et al. 2018] Shen, Y., Zhou, H., Li, J., Jian, F., e Jayas, D. S. (2018). Detection of stored-grain insects using deep learning. *Computers and Electronics in Agriculture*, 145:319–325.

[Shijie et al. 2017] Shijie, J., Ping, W., Peiyi, J., e Siping, H. (2017). Research on data augmentation for image classification based on convolution neural networks. In *Chinese Automation Congress (CAC), 2017*, pp. 4165–4170. IEEE.

[Skansi 2018] Skansi, S. (2018). *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Springer.

[Srivastava et al. 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., e Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[Szegedy et al. 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., e Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

[Szegedy et al. 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., e Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826.

[Teixeira et al. 2005] Teixeira, C. D., Saillard, C., Eveillard, S., Danet, J. L., Ayres, A., Bové, J., et al. (2005). 'candidatus liberibacter americanus', associated with citrus huanglongbing (greening disease) in são paulo state, brazil. *International Journal of Systematic and Evolutionary Microbiology*, 55(Pt 5):1857–1862.

[Vargas et al. 2016] Vargas, A. C. G., Paes, A., e Vasconcelos, C. N. (2016). Um estudo sobre redes neurais convolucionais e sua aplicação em detecçãoo de pedestres. In *Proceedings of the XXIX Conference on Graphics, Patterns and Images*, pp. 1–4.

[Wilcoxon 1945] Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83.

[Yamamoto et al. 2009] Yamamoto, P. T., Fellipe, M. R., Sanches, A. L., Coelho, J. H., Garbim, L. F., e Ximenes, N. L. (2009). Eficácia de inseticidas para o manejo de diaphorina citri kuwayama (hemiptera: Psyllidae) em citros. *BioAssay*, 4.

[Yosinski et al. 2014] Yosinski, J., Clune, J., Bengio, Y., e Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pp. 3320–3328.