
# Detecção de landmarks e classificação de subespécies de abelhas através de asas com deep learning

**Walter Betini Sandim Gomes - 40936**

Dissertação apresentada à Escola Superior de Tecnologia e de Gestão de Bragança para obtenção do Grau de Mestre em Sistemas de Informação. No âmbito da dupla diplomação com a Universidade Tecnológica Federal do Paraná

Trabalho orientado por:

Prof. Dr. Pedro João Soares Rodrigues

Prof. Dr. André Pinz Borges (UTFPR)

Profa. Dra. Maria Alice Pinto

Bragança

2018-2019


# Detecção de landmarks e classificação de subespécies de abelhas através de asas com deep learning

**Walter Betini Sandim Gomes - 40936**

Dissertação apresentada à Escola Superior de Tecnologia e de Gestão de Bragança para obtenção do Grau de Mestre em Sistemas de Informação. No âmbito da dupla diplomação com a Universidade Tecnológica Federal do Paraná

Trabalho orientado por:

Prof. Dr. Pedro João Soares Rodrigues

Prof. Dr. André Pinz Borges (UTFPR)

Profa. Dra. Maria Alice Pinto

Bragança

2018-2019

# Agradecimentos

Agradeço a Deus por cada oportunidade que tive.

Agradeço a meus pais por acreditarem em mim mais do que eu mesmo.

Agradeço aos meus orientadores Pedro João, André Pinz e Maria Alice Pinto. Aprendi coisas que jamais acreditei que entenderia, em diversos ângulos inesperados e enriquecedores. Em especial ao Professor Pedro João por sua atenção e dedicação.

Agradeço ao IPB e a UTFPR pela formação e oportunidades que me ofereceram.

Obrigado!

Este trabalho é financiado por Fundos FEDER através do Programa Operacional Competitividade e Internacionalização - COMPETE 2020 e por Fundos Nacionais através da FCT - Fundação para a Ciência e a Tecnologia no âmbito do projeto POCI-01-0145-FEDER-029871.

# Resumo

Com intuito de auxiliar no manutenção de colmeias na apicultura, esse projeto tem como objetivo o desenvolvimento de um classificador automático de subespécies de abelhas. Para isso foi desenvolvido um programa que utiliza das *landmarks* adaptadas de Nawrocka[1] para classificação, porém para realizar o processo de forma automática, foi necessário implementar um detector de objetos capaz de encontrar asas de abelha em uma imagem e um detector de *landmarks* capaz de indentificá-las em uma imagem e então proceder para classificação. O detector de objetos foi capaz de detectar 98% das asas e o detector de *landmarks* obteve foi capaz de detectar todos os *landmarks* em 91% dos casos, com uma precisão de 94% de semelhança com *landmarks* marcados a mão. A classificação por sua vez, apresentou bons resultados com as maiores classes dos datasets(em quantidade de elementos), tendo 92% de precisão com as duas maiores classes e 87% de precisão com as três maiores.

**Palavras-chave:** visão computacional, detecção de *landmarks*, detector de objetos, deep learning.

# Abstract

The aim of this project is to help bee hive maintenance in beekeeping to develop an automatic bee subspecies classifier. For this, a program was developed that uses the landmarks adapted from Nawrocka [1] for classification, but to perform the process automatically, it was necessary to implement an object detector capable of finding bee wings in an image and a detector of landmarks able to identify them in an image and then proceed to classification. The object detector was able to detect 98 % of the wings and the landmarks detector obtained was able to detect all landmarks in 91 % of cases, with a precision of 94 % resemblance to hand-marked landmarks. The classification, in turn, presented good results with the largest classes of datasets, with 92 % accuracy with the two largest classes and 87 % accuracy with the three largest.

**Keywords:** computer vision,landmark detection, object detection, deep learning

# Conteúdo

# Lista de Tabelas

# Lista de Figuras

# Capítulo 1

# Introdução

O mercado de mel tem apresentado um grande crescimento nos últimos anos [2], dentro do processo de produção existem diversos fatores direcionados a manutenção de colônias de abelhas que tem impacto nos custo de produção, qualidade do produto e na biodiversidade de abelhas na região[1], [3].

Colônias de abelhas são muito sujeitas a misturas genéticas e tais misturas podem comprometer a qualidade dos produtos gerados ou mesmo prejudicar a biodiversidade das abelhas. Uma parte muito importante no processo de garantir a qualidade do produto e de manutenção do processo de produção é avaliar constantemente as colônias de abelhas e ter certeza de sua integridade.

Para ter a informação da subespécie que compõe uma colonia existem diversos processos[3], porém, em sua maioria, os processos com boa precisão são caros e demorados. Em função de auxiliar nesse problema, esse projeto automatiza a classificação de subespécies de abelhas, com base na morfologia de asas utilizado por Nawrocka[1], com foco em velocidade, acessibilidade, precisão e redução de custos. Para isso foi necessário implementar técnicas de *machine learning* para deteção de asas, deteção de *landmarks* e modelos de classificação, dessa forma, tornando o processo totalmente automática a partir do recolhimento de imagens das asas.

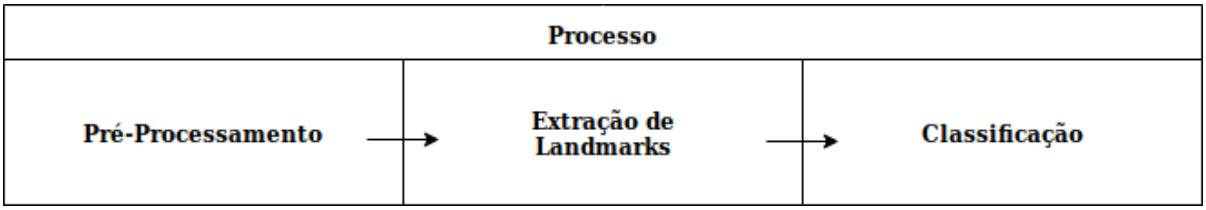


Figura 1.1: Fases do processo.

## 1.1 Enquadramento

Para se identificar a subespécie de uma abelha existem diversos processos que podem ser utilizados, eles podem ser separados em duas categorias, métodos baseados em análise morfológica como os descritos por Ruttner[3] e métodos de análise genética como as de Garnery[4] e de Jensen[5]. Os métodos morfológicos tendem a ter uma amplitude maior de instabilidade, da mesma forma variam mais em sua precisão, onde processos mais complexos (que utilizam uma quantia extensa de características para classificação) tendem a ser mais precisos. Contudo, entre os métodos mais efetivos, nas duas categorias de identificação, eles apresentam resultados bem semelhantes entre si como demonstrado por Alburaki[6].

Como ambos os métodos utilizam de análise de dados e tendem a ter um conjunto muito grande de amostras, implementações automatizadas para classificação de subespécie de abelhas passaram a ser desenvolvidas como as de Nawrocka[1] e Tofilski[7], [8]. O nosso método de identificação das subespécies, ao contrário dos existentes, é completamente autônomo. Ele deteta os *landmarks* nas nervuras das asas (tarefa manual nos outros softwares existentes) e correlaciona as posições relativas desses pontos com a subespécie em causa, identificando-a.

A figura 1.1 mostra as três fases do processo de análise de asas. Pré-processamento para preparar as imagens para os processos futuros, extração de *landmarks* para analisar as imagens e extrair as *landmarks* definidas por 1.2 e classificação é a fase do processo onde os *landmarks* são analisados e classificados.

A figura 1.2 apresenta os 19 pontos de referência(*landmarks*) utilizados utilizados por Nawrocka[1].

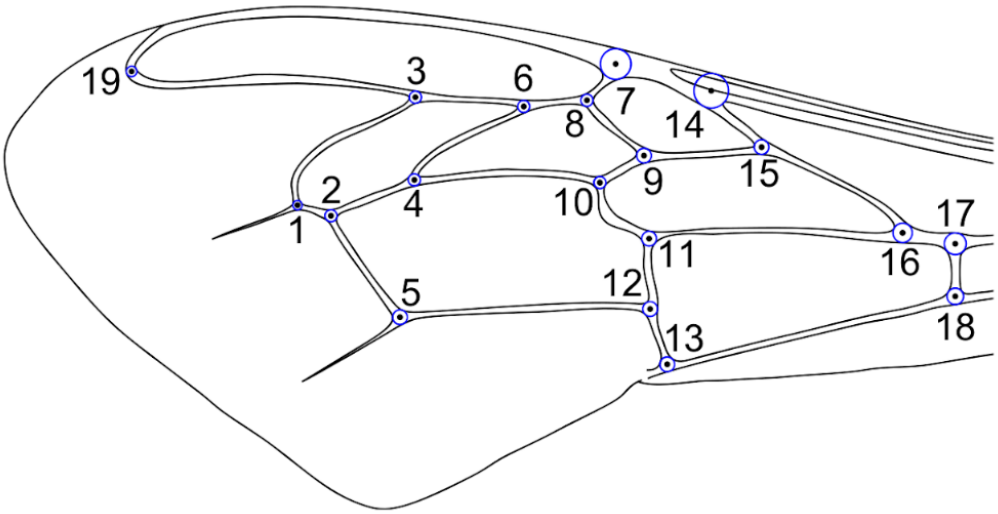


Figura 1.2: Representação dos pontos de referência (*landmarks*)[1]

## 1.2 Objetivos

Este projeto tem como objetivo desenvolver um programa capaz de fazer uma análise sobre a imagem da asa de uma abelha e ser capaz de identificar automaticamente as características utilizadas para classificação[1] e então apresentar respostas de forma clara e rápida.

Com foco no usuário simples, esse projeto não assume que o usuário terá um grande poder de processamento, dessa forma, tem também como objetivo que o programa entregado ao final do desenvolvimento seja leve e tenha uma boa velocidade de análise. Para isso é necessário assegurar baixo consumo de recursos durante o processo, além de também ser necessário que esses processam aconteçam de forma rápida, e ao final, apresentem resultados precisos.

## 1.3 Contribuições

Esse trabalho contém contribuições para o processo de manutenção de colmeias e para o desenvolvimento de detectores de *landmarks*.

O processo de deteção de *landmarks* automático não havia antes sido implementado

para auxiliar na classificação de subespécies de abelhas, tornado o processo muito mais rápido e barato.

Foi possível durante o desenvolvimento desse projeto através de um processo de segmentação semântica de imagem (U-net[9]) realizar uma deteção de *landmarks* com alta precisão, tanto em deteção de pontos quanto em qualidade de deteção. Durante os experimentos com a U-net, também foi possível reconhecer o impacto do tamanho dos kerneis utilizados pela rede durante a deteção de *landmark*.

# Capítulo 2

# Contexto e Tecnologias/Ferramentas

## 2.1 Classificação de abelhas

Uma classificação precisa de subespécies de abelhas só pode ser feita a partir de adquirido o conhecimento da viação genética de uma subespécie, dessa forma gerando a necessidade de processar um grande número de amostras para de fato compreender essa variabilidade [3]. Existem muitas características que podem ser utilizadas para classificação de subespécies e avaliar todas as características possíveis é um processo caro e demorado. A condição ideal é reconhecer subespécies de abelhas com o mínimo possível de características e de tempo gasto.

Existem diversos métodos para realizar a classificação de abelhas [10], dentre eles o método que será abordado por esse projeto é um processo de análise morfológica adaptado por Nawrocka [1].

### 2.1.1 Análise morfológica

Baseia-se de encontrar características que sejam capazes de diferenciar as subespécies entre si. No caso de Nawrocka[1], são utilizados 19 pontos localizados na asas das abelhas, como apresentado na figura 1.2, a proporção desses pontos entre si, em diversos casos é o bastante para diferenciar diversas subespécies de abelhas, como demonstrado por

Nawrocka [1]. Em outros métodos de análise das asas são utilizados o comprimento da asa, sua largura e menos pontos, onde esses pontos podem vir a ser considerados de maior impacto na diferenciação das subespécies[3], [10].

## 2.2 Deteção de *Landmarks*

*Landmark detection* ou também *keypoint detection*, refere-se ao processo de detetar características contidas em uma imagem e ter como resposta sua localização dentro da imagem.

O processo de deteção de *landmark* é comumente utilizado em tarefas como deteção de características em rostos[11] e para criar estimativa de poses[12]. Consiste na observação de uma imagem por um processo, onde a saída desse processo é constituída por um conjunto de *landmark*.

Para realizar a classificação de abelhas (através de imagens de suas asas) existem dificuldades, uma delas sendo o alto consumo de tempo para extrair as características. Com a análise morfológica de Nawroka[1] permite que com coordenadas extraídas de asas de abelhas, sua subespécie possa ser classificada, tornando um processo de deteção de *landmark* automatizado uma grande avanço na acessibilidade do projeto e também em sua velocidade.

## 2.3 Redes Neuronais

As redes neuronais são sistemas inspirados em cérebros animais, compostos de neurônios artificiais e conexões que os relacionam, gerando uma rede de neurônios. A estrutura básica de um neurônio pode ser encontrada no perceptron[13], a implementação de um único neurônio para realizar uma classificação binária, após o processo de treino.

Como apresentado na figura 2.1, o perceptron tem suas entradas associadas ao neurônio através de uma conexão com pesos, dessa forma (um fato que será multiplicado ao valor da entrada), cada característica será aletrada por seu peso correspondente e terá um impacto diferente no resultado. Existe uma constante que também é afetada por um peso próprio.

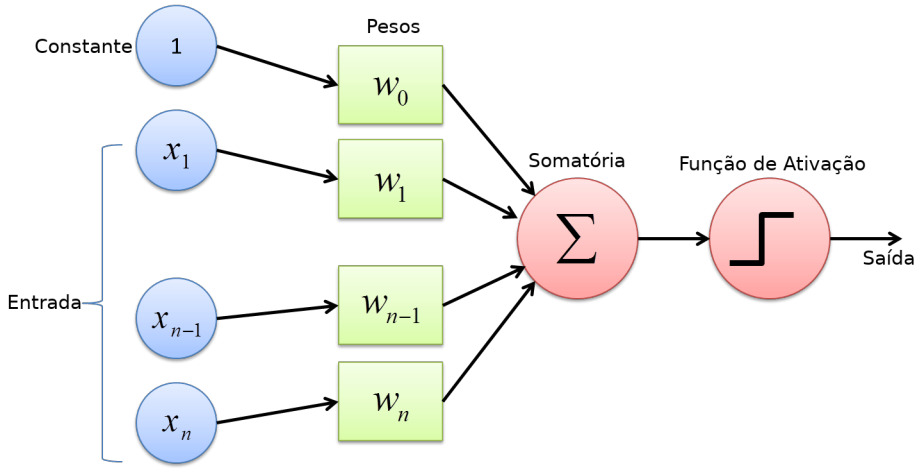


Figura 2.1: Perceptron[13], imagem alterada de (imagem alterada de *Towards Data Science*[14])

Uma vez que as entradas e a constante são alteradas por seus respectivos pesos, seus resultados são somados e então irão passar por uma função de ativação, que por sua vez, analisa o resultado da somatória e entrega a resposta do algoritmo.

A implementação de uma rede neuronal por sua vez, exige outros fatores, pois existe uma rede de neurônios que permitem expandir o processo de aprendizado de uma rede neuronal, dessa forma permitindo que problemas mais complexos sejam resolvidos e também problemas com mais do que duas classes de respostas. A conexão entre neurônios também é feita através da transferência do valor da sua função de ativação manipulado pelo peso da conexão com o próximo neurônio.

O *backpropagation* é o método utilizado para realizar o treinamento de redes neuronais recalculando seus pesos (que são iniciados aleatoriamente)[15].

### 2.3.1 Função de Ativação

A função de ativação tem o trabalho de definir se um neurônio será ou não ativado, determinando o seu valor de output. A figura 2.2 apresenta as duas funções de ativação utilizadas durante a implementação do projeto.

A primeira função (a) apresentada na figura 2.2 é a função sigmoid[16], que tem como

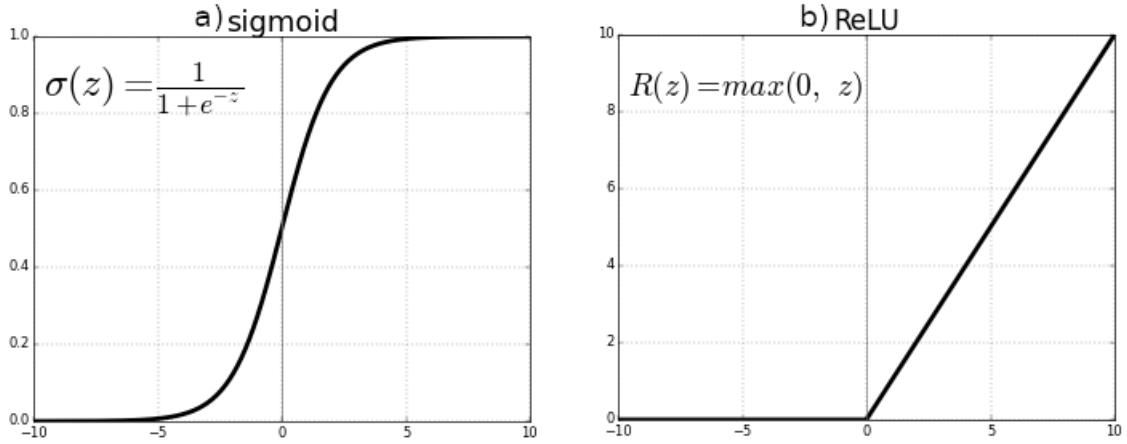


Figura 2.2: Funções de ativação: a) Sigmoid[16] e b) Relu[17] (imagem alterada de *Towards Data Science*[14])

saída um valor entre 0 e 1, onde 1 representa ativação e 0 representa não ativação do neurônio. Para determinar o o valor da função de ativação, o valor de saída $z$ do neurônio é utilizado na equação 2.1. O problema da função sigmoid é a falta de granularidade, onde a função tende a ter respostas ou muito próximas de 0 ou de 1, porém na rede U-net utilizada[9], em sua saída o objetivo é a produção de uma máscara com valores 0 e 1, onde valores entre 0 e 1 não tem grande impacto no resultado, nesse caso, essa característica passa a ser vantajosa para a implementação da rede.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.1}$$

A segunda função (b) da figura 2.2 é a função Relu[17], uma função de ativação muito simples, onde valores positivos se mantém e valores negativos viram zero. Essa função tem tido muita popularidade em camadas convolucionais.

$$R(z) = max(0, z) \tag{2.2}$$

## 2.4 Rede Neuronal Convolucional (*Convolutional Neural Network* - CNN)

Uma CNN tem uma estrutura semelhante a de uma rede neuronal, porém ao invés de neurônios convencionais, uma camada convolucional apresenta kerneis que funcionam como filtros nas imagens, extraindo características. Os valores dos kerneis de uma rede neuronal convolucional são os valores que são treinados, eles funcionam como os pesos da rede neuronal e são atualizados durante o treinamento. Esses pesos definem a característica que será extraída da imagem através do kernel, de forma a cada kernel ser capaz de extrair uma característica, independente de onde ela estiver na imagem, a partir de um conjunto kerneis é possível obter a filtragem de diversas características e então possibilitar a identificação de formas mais complexas dentro de uma imagem através da agregação de diversas camadas neuronais.

A figura 2.3 apresenta uma rede convolucional tradicional. As camadas convolucionais geralmente diminuem o tamanho da imagem devido ao kernel não ser capaz de analisar perfeitamente os cantos das imagens, esse problema pode ser removido adicionando números além da borda. As camadas de *pooling* tem a função de extrair apenas o valor de maior impacto dentro de uma área(*max pooling*), ou a média da região(*average pooling*), reduzindo drasticamente o tamanho da imagem, destacando características encontradas e reduzindo peso de processamento. A camada *Flatten* torna a imagem final dos processos de convolução em um vetor, de forma a permitir que uma rede neuronal analise as características encontradas pelas camadas convolucionais e gere uma saída.

## 2.5 Detector de Objetos

O processo de deteção de objetos tornou-se eficiente com o desenvolvimento do haar-cascade[18], um detector de objetos muito rápido e preciso, que posteriormente viria até a ser utilizado em câmeras digitais para reconhecimento de faces. Com os avanços em *machine learning*, principalmente na área de aprendizagem profunda, com o desenvolvimento

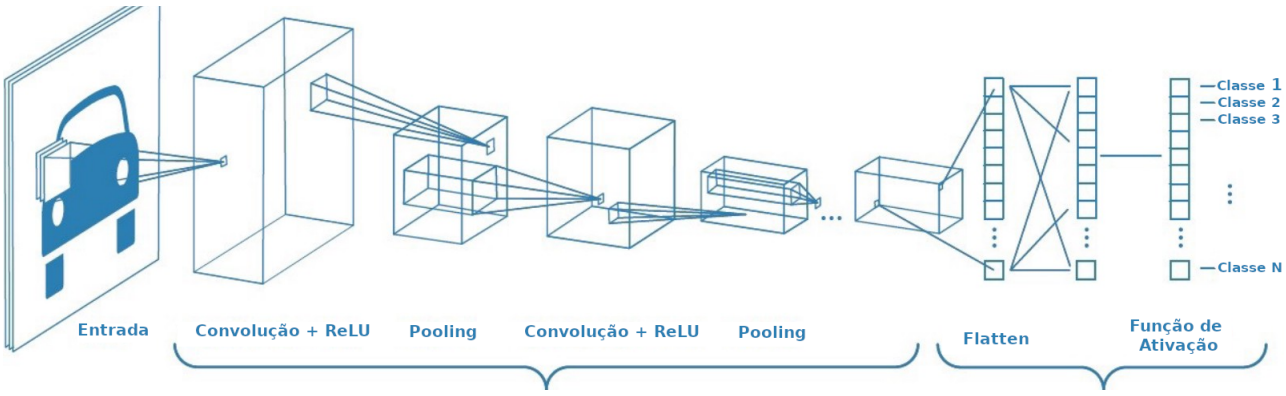


Figura 2.3: Exemplo de rede convolucional (imagem alterada de *Towards Data Science*[14]).

de redes neuronais convolucionais, surgiram detectores mais robustos, capazes de além de detetar objetos, classificá-los, tornando-se capazes de detetar mais de um tipo genérico de objeto e a da apresentação de uma *bounding box* mais precisa que métodos anteriores como o desenvolvido por Girshick [19], que não muito depois, teve implementações mais rápidas que seu projeto inicial[20], [21]. O passo seguinte para deteção de objetos, em conjunto com os já populares aparelhos de celular portáteis, foi o desenvolvimento de modelos baseados em redes neuronais convolucionais, leves e capazes de serem operados em tempo real em uma aparelho portátil, como pode ser visto na mobilenet[22] e no modelo Yolo (*You Only Look Once*)[23], esses atributos são muito importantes dentro dessa pesquisa, considerando que o produto final deve ser capaz de funcionar em computadores menos poder computacional.

### 2.5.1 Interseção sobre a União

Interseção sobre união é uma métrica de avaliação muito comum para avaliar a precisão de detectores de objetos[24], sendo capaz de avaliar a precisão de uma deteção comparando o resultado obtido com o resultado esperado.

Para realizar a avaliação é necessário ter a área e as coordenadas da *bounding box* esperada e a *bounding box* obtida. A partir delas é possível obter a área de interseção e a área de união. A área de interseção é a área da imagem onde ambas as *bounding boxes* se

sobrepõem, a área de união é a área de ambas as *bounding boxes*, ou a soma da área de ambas, menos a área de interseção.

Para calcular a intersecção sobre a união é necessário dividir a área de interseção pela área de união. O resultado será um um número entre 0 e 1, onde 0 significa que não existe interseção e 1 significa que o resultado obtido foi exato.

### 2.5.2 Precisão

Para se avaliar a precisão de um detector de objetos é necessário o uso de uma métrica que descreva a quantidade de acertos e a qualidade deles. Para calcular a quantidade de acertos basta utilizar a equação 2.3, onde a precisão $P$ é igual aos verdadeiros positivos, divididos pela soma dos verdadeiros positivos com os falsos positivos. Um falso positivo é uma deteção falhada e a os verdadeiros positivos são as detecções corretas. Para determinar se uma deteção está correta, existem diversas medidas padronizadas que podem ser utilizadas, como detecções para o tamanho das imagens ou atribuindo a qualidade de uma deteção à sua intersecção sobre a união[24]. Existem dois métodos comuns de avaliar a qualidade de uma deteção com a intersecção sobre a união. Um deles são casos onde a Intersecção tem um valor acima de 0.5, ou seja, 50% do recorte está correto, outro limite comumente utilizado é o de 0.75, onde 75% do recorte contém o objeto alvo[24]. A precisão utilizada nesse projeto se baseia em uma intersecção sobre a união acima de 0.75.

$$P = \frac{VP}{VP + FP} \tag{2.3}$$

## 2.6 U-Net

A U-net é uma rede neuronal completamente convolucional desenvolvida para segmentar características dentro de imagens [9]. Seu intuito original era de segmentar células em imagens e separá-las para poderem ser identificadas separadamente. Devido a sua alta precisão, baixa complexidade, bons desempenhos com pequenas bases de dados e

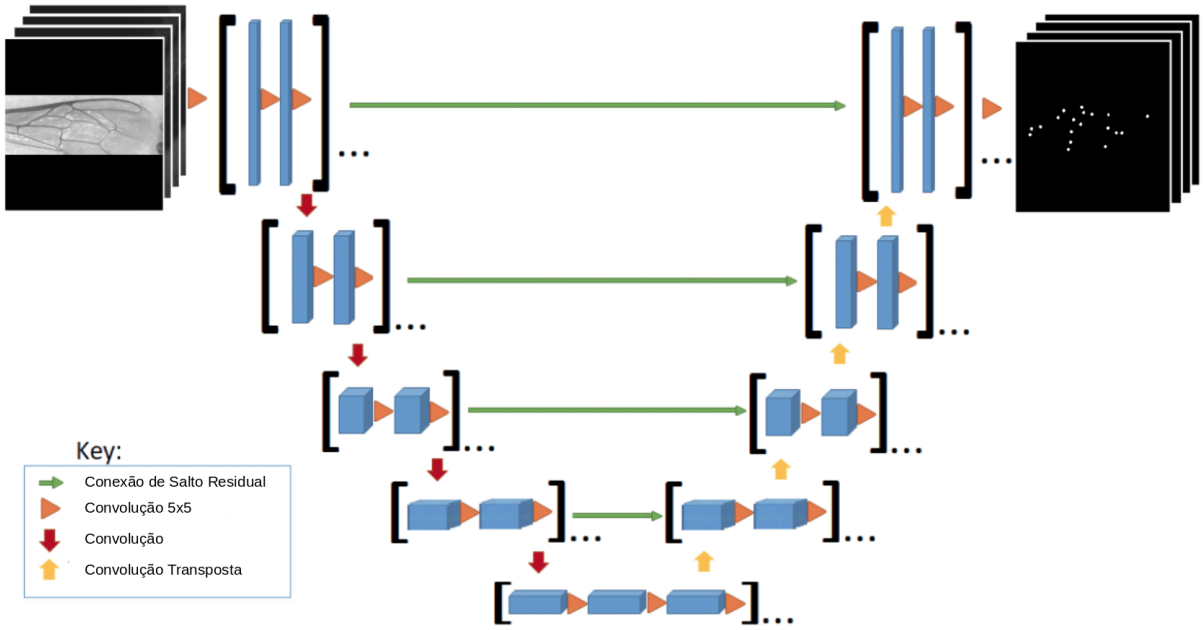


Figura 2.4: Estrutura da U-net utilizada, imagem alterada de Georgiev[25]

forte capacidade em separar as áreas segmentadas (evitar que duas áreas diferentes se misturem)[9].

A U-net foi considerada uma boa opção para o processo de deteção de *landmark*, pois, apesar dela retornar uma imagem, ela é leve, rápida e as *landmark* que serão extraídas não se sobrepõem como mostrado na figura 1.2, desas forma ao analisar a máscara gerada pela U-net, cada elemento da máscara (circulo branco) corresponde a uma *landmark*, com baixa chance dos elementos se unirem devido a estarem bem separados. A figura 2.4 apresenta a estrutura da U-net que apresentou os melhores resultados (com todos os kerneis de tamanho 5x5) e que está a ser utilizada como o modelo final do projeto. Foram testadas outras configurações, alterando o tamanho dos kerneis presentes na rede.

## 2.7 *Principal Component Analysis* (PCA)

O PCA é uma técnica para análise de dados com foco em realizar uma redução de dimensionalidade nos dados e facilitar a compreensão e análise dos dados tentando representá-los de forma mais simples. O custo da redução de dimensionalidade acontece na perda de

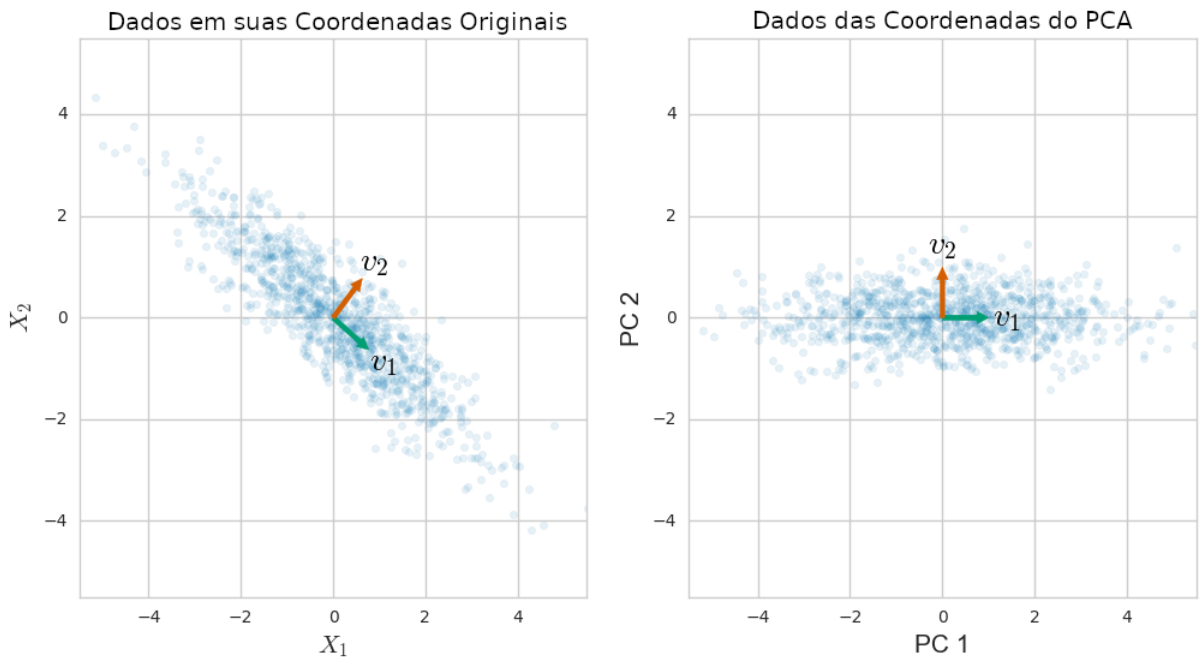


Figura 2.5: Exemplo de ajuste de ângulo com PCA, (imagem alterada de *Towards Data Science*[14])

precisão, onde os dados não são mais representados com 100% de fidelidade[26]. A técnica é comumente utilizada para análise de dados e reconhecimento de padrões em processamento de sinal e de imagens[27].

O PCA utiliza uma transformação ortogonal para converter um conjunto de observações de variáveis (onde é assumido que existe relação entre as variáveis, num conjunto de valores lineares chamados de componentes principais[26]. A quantidade de componentes principais é igual ou menor que a quantidade de variáveis originais. Esses componentes principais servem como representação da distribuição de dados (ou a amplitude de variação das variáveis) e são representados como vetores n dimensionais, onde n é igual a quantidade de variáveis.

No caso de processamento de imagens, o PCA pode ser utilizado para alterar o ângulo de uma imagem rotacionando-a de acordo com seus componentes principais[27].

No caso do problema desse projeto, é possível ver que apesar da classificação ser feita baseada na diferença entre os objetos, a proporção entre as *landmark* é próxima o bastante

para ter componentes principais muito próximos, o que auxilia no ajuste do ângulo dos *landmark*. Com os *landmark* ajustados em ângulo, o processo de ordenar os *landmark* em um vetor torna-se mais fácil, pois é possível ordená-los usando seus eixos como critério, como descrito no capítulo 4.3.4 Processamento de Pontos. Uma ordenação consistente de *landmark* torna-se importante na fase de classificação, onde os algoritmos utilizados precisam de uma entrada padronizada.

## 2.8 Procrustes

A análise procrustes generalizada[28] é comumente utilizada na análise morfológica de asas[29]. Ela ajuda a garantir uma análise de objetos onde os itens que estão sendo comparados encontram-se na posição mais próxima possível de encaixe, o que permite comparações muito mais precisas entre formas.

Uma análise procrustes realiza transformações em um objeto para que ele se encaixe o mais próximo possível com a referência (realizando uma superimposição de um objeto sobre a referência). As alterações realizadas pelo ajuste procrustes são: translação, escala, rotação e reflexão.

Essa comparação de objetos permite que, ao analisar uma asa, ela possa ser superimposta sobre uma referência (que representa o *dataset*), e comparada de forma mais efetiva, garantindo que variáveis como ângulo, tamanho, reflexo e translação não venham a representar um problema na hora de efetuar uma classificação.

Para comparar um objeto com todo um conjunto de objetos, é necessário realizar uma análise procrustes generalizada, que por sua vez acrescenta regras para garantir uma boa referência que represente adequadamente todo o conjunto de objetos[28]:

1. Selecionar um elemento aleatório do *dataset* como referência
2. Ajustar o *dataset* de acordo com a referência
3. Calcular uma média do *dataset* ajustado

4. Se a média do *dataset* ajustado tem uma diferença vetorial com a referência maior que um limite pré determinado, a média torna-se a nova referência e o processo volta para o estágio dois.

### 2.8.1 Transformações Procrustes

Para realizar superposição do objeto sobre a referência são realizadas quatro processos: translação, escala, rotação e uma checagem se reflexos do objeto tem um encaixe mais adequado.

Para sobrepor um objeto sobre outro é necessário identificar a média de seus pontos, de forma a encontrar a média do valor dos eixos e criar um ponto que representa o seu centro, e com esse centro, deslocá-lo para a origem, subtraindo o valor da média dos eixos nos pontos. A equação 2.4 apresenta uma forma de identificar o centro do objeto para uma quantidade $n$ de pontos.

$$\bar{x} = \frac{x_1 + x_2 + ... + x_n}{n}, \bar{y} = \frac{y_1 + y_2 + ... + y_n}{n} \tag{2.4}$$

Para garantir que a escala dos objetos é a mesma é preciso escalá-los de forma eficiente. Para descobrir a escala atual do objeto é possível utilizar a equação 2.5.

$$Escala = \sqrt{\frac{(x_1 - \bar{x})^2 + (y_2 - \bar{y})^2 + ...}{n}} \tag{2.5}$$

A partir da Escala é possível reajustar a escala dos pontos com a equação 2.6, tornando sempre a escala igual a 1. É importante ressaltar que se a a translação já foi feita sobre o objeto, $\bar{x}$ e $\bar{y}$ são iguaia à 0, dessa forma, podendo ser desconsiderados nas equações 2.5 e 2.6, caso a translação ainda não seja feita, o resultado da equação 2.6 já coloca o objeto com seu centro sobre a origem.

$$(\frac{x_1 - \bar{x}}{Escala}, \frac{y_1 - \bar{y}}{Escala}) \tag{2.6}$$

Para realizar a rotação de um ponto por um ângulo $\theta$ basta utilizar a equação 2.7,

porém, para realizar a superposição dos objetos, é preciso garantir o encaixe mais preciso possível. Para descobrir o ângulo adequado para realizar essa rotação a equação 2.8 é utilizada, onde $((x_1, y_1), ...)$ representa o objeto que será rotacionado e $((w_1, z_1), ...)$ representa a referência.

$$(u, v) = (\cos\theta x - \sin\theta x, \cos\theta y + \sin\theta y) \tag{2.7}$$

$$\theta = \tan^{-1}\left(\frac{\sum_{i=1}^{n}(w_i y_i - z_i x_i)}{\sum_{i=1}^{n}(w_i x_i + z_i y_i)}\right) \tag{2.8}$$

### 2.8.2 Distância Procrustes

A distância procrustes[28] é uma medida que avalia a discrepância entre dois objetos, onde, quanto menor o valor do resultado, maior a semelhança entre o objeto e a referência.

Após a análise e a superposição de objetos, a discrepância $D$ pode ser calculada como demonstrado na função 2.9 [30]:

$$D^2 = \sum_{i=1}^{n}((x_i - w_i)^2 + (y_i - z_i)^2) \tag{2.9}$$

## 2.9 Classificadores

Para realizar a classificação de subespécies é importante que o classificador utilizado tenha habilidade em lidar com uma quantidade limitada de informação, devido à quantidade limitada de amostras. Para isso foram escolhidos os classificadores: support-vector machine (SVM), naive bayes e random forest.

### 2.9.1 Support-vector machine

Uma SVM é um algoritmo de *machine learning*, comumente usado para classificação e regressão[31]. Seu funcionamento baseia-se em analisar elementos (classificados) e descrever um hiperplano capaz de separar as classes da melhor forma possível. Um elemento

pode ter n características então ele é um vetor n dimensional, dessa forma, para separar os elementos, é necessário o uso de um hiperplano, capaz de separá-los em n dimensões com uma margem generalizadora o mais ampla possível.

A margem generalizadora é a uma margem entre as classes separadas pelo hiperplano e o hiperplano, uma boa margem permite classificações mais precisas, onde classes não se misturam. Idealmente uma margem generalizadora faz com que o hiperplano fique equidistante e o mais longe possível das classes.

### 2.9.2 Naive Bayes

O Naive Bayes é um modelo de probabilidade condicional na área de *machine learning* supervisionado. Ele analisa as características do *dataset* de treino e gera um modelo que ao analisar as características de novos elementos, ele calcula a probabilidade desse elemento pertencer a uma classe.

A equação 2.10 mostra a função de probabilidade condicional de Bayes[31] onde $P(A|B)$ é a probabilidade de A e B acontecer de A acontecer sabendo que aconteceu antes B, $P(B)$ representa a probabilidade a priori de $B$, ou seja, antes da descoberta de $A$, e $P(B|A)$ representa a probabilidade condicional, ou seja, após a descoberta de $A$.

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \tag{2.10}$$

### 2.9.3 Random Forest

A random forest (árvore aleatória)[32], é uma versão variante da árvore de decisão onde mais de uma árvore é implementada. Durante o treinamento, diversas árvores com configurações diferentes são treinadas e durante o processo de classificação, cada árvore funciona como um voto, onde a classe com mais votos é a classe escolhida como saída da random forest, em casos de regressão, onde o objetivo é prever algo ou gerar um valor baseado em características, a saída do modelo é a média da resposta das árvores.

Uma árvore de decisão por sua vez, para gerar uma classificação analisa o impacto

de cada característica presente no *dataset* de treino. Ao descobrir a característica mais impactante na classificação, o *dataset* é dividido com base nessa característica e ela é removida do *dataset*, gerando ramos (cada ramo representando uma variação da característica selecionada). Esse processo é repetido diversas vezes, criando novos ramos, até um ponto onde pode ser definida uma classe para o elemento, ou acabem características, ou a árvore tenha atingido o limite máximo de profundidade.

O processo de classificação de uma árvore de decisão baseia-se em analisar o elemento e percorrer os ramos da árvore correspondentes a suas características, onde no final desses ramos (na folha), existe a classe respectiva a esse elemento.

## 2.10 Ferramentas

O desenvolvimento do projeto dependeu de diversas ferramentas e bibliotecas para o seu desenvolvimento.

### 2.10.1 MorphoJ

O morphoJ é um programa desenvolvido para análise geométrica morfológica[29]. Ele apresenta características interessantes como uma implementação própria da análise procrustes generalizada e deteção de *outliers*. Ambas essas funções auxiliaram no desenvolvimento do projeto, tanto para comparar resultados da análise procrustes generalizada[28] quanto para deteção de outliers dentro dos resultados obtidos, o que auxiliou a melhorar no processo de ordenação de *landmark* devido a função de visualização de *outliers*.

### 2.10.2 Bibliotecas

Durante o decorrer do projeto, diversas bibliotecas foram utilizadas para auxiliar o seu desenvolvimento.

### OpenCV

O OpenCV [33] é uma biblioteca aberta para manipulação de imagens da qual apresenta uma boa documentação, interface em python e apresenta um desempenho muito bom. A biblioteca Pillow foi considerada, porém, apesar de ser muito mais simples, é também muito mais lenta. Abrir imagens e editá-las com OpenCV apresentou um desempenho aproximadamente 10 vezes melhor que utilizando Pillow.

### Tensorflow

O tensorflow[34] é uma biblioteca criada pela Google para desenvolvimento e pesquisa na área de *machine learning*, com foco em redes neuronais[35]. É uma das ferramentas mais populares da área em conjunto com pytorch[36], Caffe[37] e entre outros. O Keras[38] por sua vez é uma interface de alto nível para implementação de redes neuronais dentro do tensorflow, da qual além de permitir uma implementação simples de diversos conceitos de redes neuronais, apresenta alto desempenho [39].

Contido dentro do tensorflow existe também uma API (*Application programming interface*) para detectores de objetos previamente treinados, dos quais basta realizar transferência de aprendizado para utilizá-los em outros problemas, de forma rápida e efetiva[40].

O tensorflow apresenta uma comunidade muito ativa em comparação com outras bibliotecas de forma a ser mais fácil resolver problemas relacionados ao seu funcionamento e implementação[41].

#### *Object Detection Zoo*

Dentro da API de deteção de objetos do tensorflow[40] é possível encontrar uma sessão dedicada a modelos já testados dentro da plataforma, esses modelos são treinados dentro de diversos *datasets* e disponibilizados já treinados dentro da API.

Para este projeto foram testados alguns detectores de objetos dessa API, dos quais foram previamente treinados dentro do COCO *dataset*[42]. A API também oferece algoritmos para realizar transferência de aprendizado (*transfer learning*) nesses modelos de

forma a tornar o processo mais simples do que a confecção completa de um detector de objetos.

### Scipy

O Scipy é um eco-sistema em python desenvolvido para aplicações matemáticas, cientificas e de engenharia[30]. Dentro desse eco-sistema, quatro pacotes foram utilizados durante o desenvolvimento desse projeto: Numpy, matplotlib, pandas e Ipython.

Para a maior parte dos processos relacionados à matemática e manipulação de vetores n-dimensionais, a biblioteca numpy apresenta funções para lidar com diversos problemas matemáticos e realizar interações entre vetores de forma muito eficiente[43].

O Ipython é um pacote que dá suporte em python para programação interativa, permitindo visualização interativa no terminal[44].

Matplotlib é uma biblioteca de plotagem 2D, que produz desde representações de dados simples até gráficos interativos, o que em conjunto com o Ipython gera um processo rápido de análise de dados e desenvolvimento[45].

Um vez que os dados foram extraídos e precisam ser manipulados e analisados, a biblioteca pandas apresenta diversas funcionalidades que tornam o processo muito mais eficiente[46]. Além de ter suporte para diversos tipos de arquivos, a biblioteca apresenta diversas funções de busca, ordenação e análise de grandes bases de dados.

### Scikit Learn

O scikit learn é uma biblioteca com foco em *machine learning* e suporte no uso de seus algoritmos[47]. Ele apresenta diversas implementações de algoritmos de *machine learning* como SVMs, árvores de decisão, entre outros, mas também apresenta funções que dão suporte a manipulação de datasets com foco no seu uso na área de *machine learning*. Outra vantagem do scikit-learn é o fato de sua implementação funcionar em conjunto com as funções e objetos de biblioteca scipy[30], principalmente o numpy.

# Capítulo 3

# Abordagem/Análise/Modelação

## 3.1 Problemas

A partir da premissa de automatizar completamente o processo de deteção de asas de abelhas foi necessário estabelecer os problemas a serem resolvidos. O projeto foi dividido em duas partes: pré-processamento, deteção de características e classificação.

Para o pré-processamento é necessário realizar a leitura da imagem a ser detetada e padronizá-la no formato da rede que efetuará o pré-processamento. Dentro dessa área é necessário detetar uma ou mais asas dentro da imagem, recortá-las de forma padronizada, escalá-las sem causar deformação para o formato de entrada da rede de deteção de *landmark* e alterar as cores para tons de cinzento.

Na deteção de características é necessário ter garantia da deteção de todos os *landmark* e de sua precisão em relação aos pontos originais, a ordem dos *landmark* extraídos também influencia no processo de deteção de forma onde garantir que os *landmark* são detetados na ordem correta tem um peso importante na avaliação do processo de deteção de características.

Detetar os *landmark* exige muita precisão e é repleta de problemas relacionados ao desbalanceamento de classes. Considerando os *datasets* que estão a ser utilizados, existe um alto desbalanceamento na quantidade de elementos presentes em cada classe, onde

a menor classe apresenta 12 amostras e a maior delas, aproximadamente 3500 amostras. Devido ao baixo número de amostras de algumas classes, o processo de deteção não pode ser resolvido com implementações básicas de redes neuronais, que necessitam de uma grande quantidade de amostras de cada classe[13]. Para resolver esse problema de deteção devem ser utilizados métodos que apresentam maior versatilidade com um número menor de exemplos.

Lidar com variações dentro do *dataset* exige adaptações em diversas partes do desenvolvimento. Dentre as variações mais importantes podem ser encontradas características como, deslocamento (local a ser analisado na imagem não é padronizado), tamanho da imagem, iluminação, ruído, manchas e variação de ângulo. Diversos desses fatores como iluminação, deslocamento e tamanho da imagem podem ser resolvidos com mais facilidade na sessão de pré-processamento, porém, características como ruído, manchas e variação de ângulo tornam-se mais complexas e tornam o processo de deteção de características mais difícil, pois são características difíceis de serem tratadas.

Um método que foi considerado para deteção das subespécies foi o uso de CNNs para deteção direta nas imagens (sem uso de *landmark*), porém como os *datasets* apresentam iluminação e qualidade de imagem muito destoantes entre classes, não é possível ter certeza se um processo de deteção baseado unicamente na imagem é capaz de realizar uma classificação sem ficar tendencioso à iluminação e outras características distintas de cada classe, devido a variações nas imagens não relacionadas às asas, fora isso, ainda existe o problema do desequilíbrio na quantidade de elementos entre as classes, onde existem classes compostas de apenas 20 imagens e outros com mais de 100, o que justifica a inviabilidade dessa abordagem.

## 3.2 Necessidades

Como o objetivo inicial desse trabalho é possibilitar um programa acessível a usuários de baixo nível, algumas características tornam-se necessárias para que o programa possa ter o máximo de acessibilidade possível.

A velocidade é uma grande necessidade do programa, pois a demora em realizar a tarefa pode ser custosa para o usuário, tanto em tempo, quando em gasto de recursos (processamento/energia). Caso o processo de deteção seja mais lento que um processo manual, as vantagem no seu uso são reduzidas drasticamente.

Para garantir seu funcionamento no maior número de computadores possível, é vantajoso que o programa seja leve, tanto ao âmbito de espaço de armazenamento, quanto ao uso de memória em tempo de execução, pois muitos processos de redes neuronais consomem muita memória e poder de processamento, tornando-os inutilizáveis em diversos computadores convencionais.

Para garantir a usabilidade e efetividade do programa, a precisão passa a ser o requerimento mais importante do projeto. Existem, nesse trabalho, diversos pontos com implementação de modelos de *machine learning*(o detector de objetos, a U-net e os classificadores), que podem gerar acumulo de erro, atrapalhando o resultado final de deteção. Para evitar o acumulo de erro é importante a maximização da precisão e métodos que lidem devidamente com as saídas dos modelos.

## 3.3 Descrição

O desenvolvimento do projeto foi dividido em três partes: pré-processamento, extração de *landmark* e análise/classificação de características.

O pré-processamento constitui a preparação das imagens para serem utilizadas pelos modelos de deteção de landmark, tem como entrada um conjunto de imagens e tem como saída um conjunto de asas detetadas em preto e branco em uma imagem 400x400, como é apresentado na figura 4.2.

O processo de extração dos *landmark* utiliza as imagens criadas pelo pré-processamento para detetar os *landmark*. Essa fase do processo possui 3 implementações diferentes, uma onde o processo é realizado apenas com processamento de imagem, sem o uso de redes neuronais, outro com redes neuronais convolucionais como as demonstradas na figura 2.3, e a terceira implementação utiliza variações do modelo U-net[9].

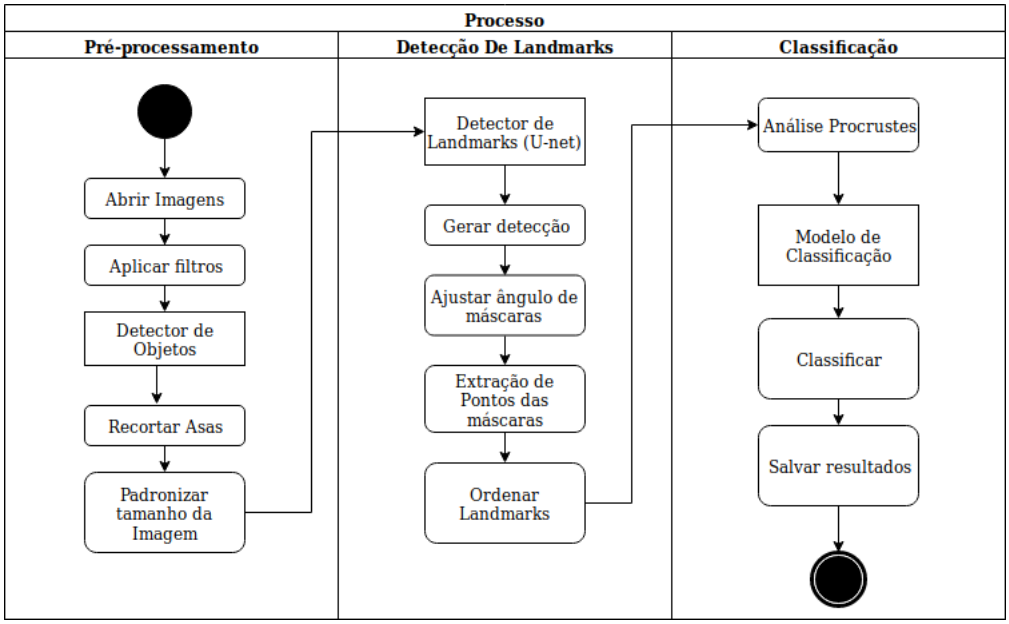


Figura 3.1: Fluxograma do programa final.

Na versão com a U-net implementada, o processo de extração de *landmark* exige que os pontos sejam extraídos da máscara gerada na saída da U-net. A maior vantagem do uso de CNNs mais convencionais é o fato dos *landmark* na saída da rede já estão devidamente ordenados, enquanto nas outras duas versões da implementação (a versão sem redes neuronais e a versão que utiliza a U-net), é necessário realizar um ajuste no ângulo da imagem e realizar uma ordenação nos *landmark*. A saída do processo de extração de *landmark* é um conjunto de detecções bem sucedidas (cada uma composta de 19 *landmark*), já ordenadas.

Para analisar as *landmark* foram implementados diversos modelos, que convenientemente, devido a padronização do Scikit-Learn, as entradas e saídas são padronizadas de forma a troca do modelo, não impactar no resto da estrutura do programa. A saída do processo de análise é um conjunto classificações das subespécies de abelhas, respectivas aos conjunto de detecções que foi utilizado de entrada para os modelos de classificação.

A figura 3.1 apresenta a estrutura final do programa, com a implementação da U-net.

### 3.3.1 Pré-Processamento

O pré-processamento tem o trabalho de preparar os dados para serem analisados pelos processos da Extração de *landmark*. Nessa etapa são implementados filtros que ressaltam características importantes da imagem, ou que removem dados irrelevantes (nesse caso, cores e balanceamento de iluminação), e também é realizado o processo de deteção e recorte de asa nas imagens de entrada, essas imagens então passam para um processo de alteração de escala para padronizá-las com o padrão das entradas dos modelos utilizados.

### 3.3.2 Extração de *landmark*

Para extrair as *landmark* serão testadas três abordagens. A primeira composta de apenas de ferramentas de processamento de imagem, com uso da biblioteca OpenCV[33], a segunda composta de CNNs comumente utilizadas em problemas de deteção de *landmark* e a terceira é constituída na implementação de um modelo de U-net adaptado para o problema atual.

O processo de extração de *landmark* também é constituído de encontrar métricas adequadas para averiguar a precisão dos algoritmos utilizados, de forma a possibilitar uma comparação efetiva entre eles.

### 3.3.3 Análise de Características

A partir da extração de *landmark*, os dados podem ser analisados para realizar a classificação das subespécies. Serão testados diversos modelos já implementados dentro da biblioteca do Scikit-learn[47]. Os modelos que serão utilizados para classificação serão: SVM, Random Forest e Naive Bayles.

# Capítulo 4

# Desenvolvimento/Implementação

O programa desenvolvido é capaz de realizar segmentação em asas de abelhas encontrando os 19 pontos de referência (*landmarks*) baseados nos pontos utilizados por Nawroka[1] para classificação, como apresentado na figura 1.2.

## 4.1 Base de Dados

Para o desenvolvimento foram utilizados dois conjuntos de imagens, um deles (*dataset* 1) composto por 5770 imagens de asas de abelhas catalogadas com os seus pontos de referência (catalogados à mão pela nossa equipa em pesquisas anteriores[48], [49]) pertencentes a duas subespécies (*A. m. iberica* e *A. m. ligustica*) extraídas da região da península ibérica e da região dos Açores.

O segundo *dataset* (*dataset* 2) é composto por 2447 imagens de asas extraídas de diversas regiões da Europa e da África, essas imagens encontram-se majoritariamente sem pontos de referência (apenas 190 delas possuem pontos de referência). Neste último grupo temos 26 subespécies no total, dos quais duas pertencem já estão presentes no *dataset* 1, estas sendo as *Iberienses* e a *Ligustica*. A maior complicação na análise do *dataset* 2 é relacionada a alta variância na qualidade de imagens no *dataset*, muitas imagens são antigas e apresentam uma qualidade de imagem muito baixa, ou mesmo, muita sujeira e ruído na imagem, o que atrapalha no processo de deteção de *landmarks*.

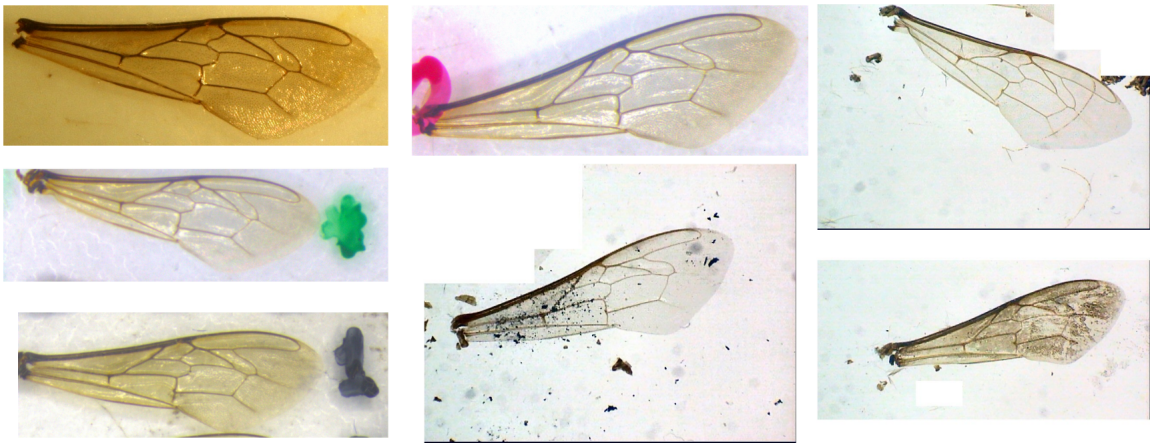

Figura 4.1: Exemplares dos *datasets* 1 e 2.

Os nomes das subespécies presentes no *dataset* são: *Apis mellifera anatoliaca (A. m.)*, *A. m. litorea*, *A. m. scutellata*, *A. m. monticola*, *A. m. iberica*, *A. m. sahariensis*, *A. m. unicolor*, *A. m. syriaca*, *A. m. mellifera*, *A. m. adami*, *A. m. major*, *A. m. cypria*, *A. m. macedonica*, *A. m. jemenitica*, *A. m. intermissa*, *A. m. ligustica*, *A. m. carnica*, *A. m. ruttneri*, *A. m. cecropia*, *A. m. armeniaca*, *A. m. meda*, *A. m. caucasica*, *A. m. sicula*, *A. m. lamarkii* e *A. m. capensis*.

As imagens originais consistem de um conjunto de imagens coloridas com várias asas na mesma imagem e também de imagens com apenas uma asa. De início, as imagens (asas) foram recortadas manualmente e tiveram seus pontos de referência catalogados gerando o conjunto de imagens utilizados para o desenvolvimento/treino do anotador de *landmarks* (segmentador). A figura 4.1 mostra exemplos de ambos os *datasets* (1 e 2), com intuito de descrever a variabilidade das amostras encontradas.

## 4.2 Processo de Pré-processamento

Com as imagens catalogadas foi desenvolvido primeiramente um detector de asas para automaticamente localizar asas em uma imagem e recortá-las, assim facilitando a interação do usuário com o programa e aumentando a precisão do anotador de *landmarks* com imagens mais padronizadas, com intuito de padronizar e melhorar a qualidade de iluminação

Figura 4.2: Exemplos do recorte gerado para o treinamento.

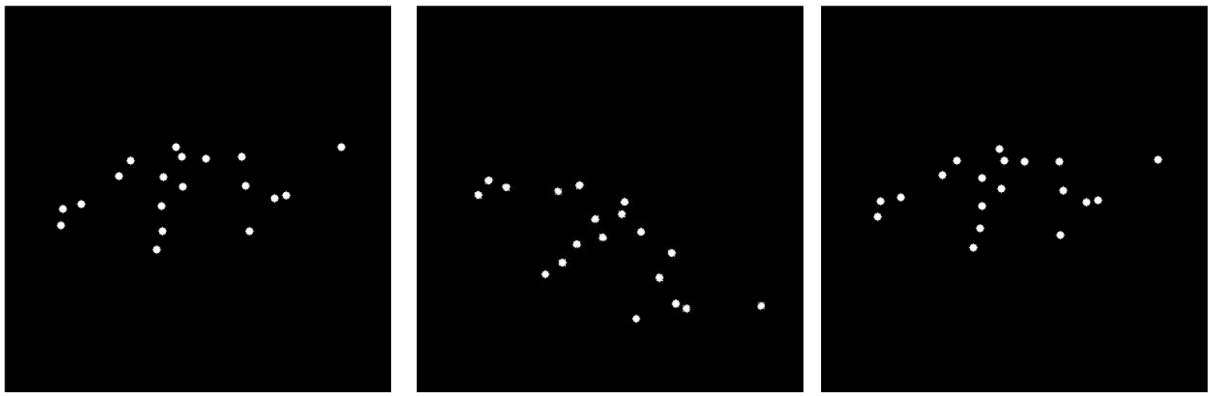

Figura 4.3: Exemplos de máscaras para a U-net.

das imagens, o filtro Clahe[33] foi utilizado em todos os recortes. A figura 4.2 monstra um recorte gerado para o treinamento das redes neuronais de deteção de *landmarks*.

Foram feitas máscaras de tamanho igual aos da entrada onde os pixeis das regiões de interesse ficam brancos e os demais pretos. Essas máscaras servem como saída da rede neuronal (U-net[9]). A figura 4.3 apresenta exemplos de máscaras criadas para o treinamento da U-net.

### 4.2.1 Aumento de Dados

Para detetar os pontos de referência foram testadas e treinadas diversas estruturas de redes neuronais. Ao comparar os resultados foi escolhido o segmentador neuronal U-net[9] que apresentou os melhores resultados. Devido ao *dataset* 2, ser menor e apresentar uma

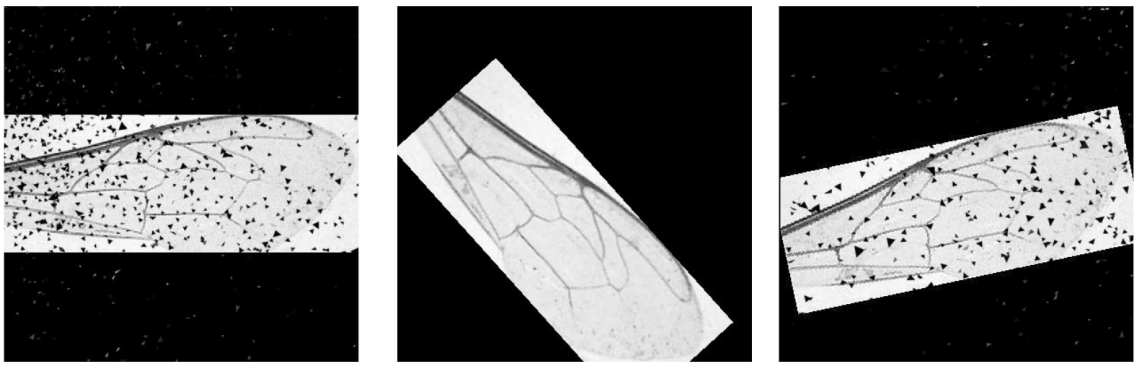

Figura 4.4: Exemplos imagens alteradas

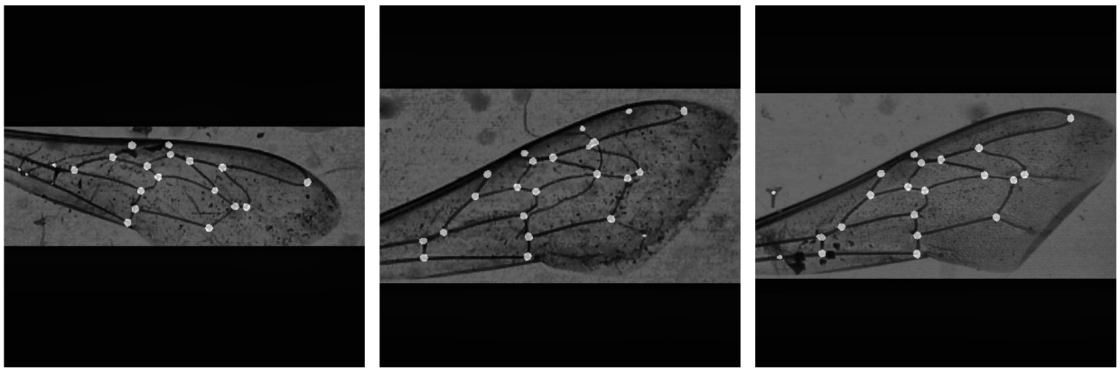

Figura 4.5: Exemplos de erros em imagens com sujeira.

variação visual muito maior que o *dataset* 1, foi necessário expandir artificialmente esse grupo simulando condições visuais típicas desse conjunto de imagens (*data augmentation*). Foram simulados poeira, ruído e alterações drásticas de ângulo na imagem da asa, como pode ser visto na figura 4.4.

Essas alterações criaram uma base de dados muito maior e ajudam o anotador automático de *landmarks* a ser muito mais preciso, porém, em imagens com muita sujeira ainda são perdidos alguns pontos de referência. A figura 4.5 mostra detecções falhas onde a sujeira na imagem distorce uma deteção ou cria detecções a mais do que deveria.

### 4.2.2 Deteção de Asas

Como o programa não pode esperar que a imagem de entrada seja padronizada, o programa deve lidar com imagens variantes em diversos sentidos:

- Tamanho da imagem
- Posição da asa dentro da imagem
- Quantidade de asas em uma única imagem

Para isso foi necessário o uso de um detector de objetos capaz de detetar a existência de uma ou mais asas dentro de uma imagem e suas coordenadas, para assim recortá-las e examiná-las efetivamente.

Nenhum dos *datasets* apresentavam *bounding boxes*, então elas precisaram ser confeccionadas. Utilizando as coordenadas das *landmarks* nas imagens catalogada é possível detetar a posição das asas dentro das imagens. Para detetá-las as *landmarks* são lidas e são separadas duas coordenadas, uma com o maior valor do eixo X e do eixo Y e outra com o menor valor do eixo X e Y, esses valores representam uma *bounding box* mínima onde todas as *landmarks* estão presentes dentro dessa área.

Porém o processo de deteção de objetos precisa de uma margem de erro, para garantir que os recortes sempre irão ter todas as *landmarks*. Essa margem de erro também pode auxiliar no processo de deteção de características, permitindo que a rede veja detalhes ao redor de todas as *landmarks*.

Para realizar os recortes da forma mais adequada possível, foram utilizados diversos modelos de detector de objetos. Os modelos testados foram:

- ssd mobilenet v1 fpn coco[22]
- faster rcnn nas[20]
- faster rcnn inception resnet v2 atrous coco[21]
- YoloV3/Darknet[23]

O modelo que apresentou os os melhores resultados foi a ssd mobilenet v1 fpn coco[22], mantendo bons resultados quando comparado com os outro modelos, porém sendo muito mais rápido. A implementação entregue nos modelos do Tensorflow, para manipulação

de arquivos utiliza uma biblioteca chamada Pillow[50], essa biblioteca, embora apresente funções de alto nível para manipulação de imagens, torna-se ao mesmo tempo consideravelmente mais lenta. Devido a isso, a biblioteca utilizada para abrir imagens foi alterada para o OpenCV[33], que apresentou um desempenho mais rápido.

### 4.2.3 Formato de Imagem

Como as características buscadas não dependem de cor, todas as imagens ao serem utilizadas são convertidas para níveis de cinzento, essa alteração também auxilia na economia de memória. Como as imagens de asas apresentam variação nos ângulos posicionais, as imagens recortadas tendem a ter tamanhos muito diferentes e como a entrada da rede neuronal precisa ser padronizada, após o recorte para segmentação é gerada uma imagem de tamanho padronizado que considera variações de ângulo na asa, colocando um espaço preto extra em imagens que não a preencham completamente. Para manter ainda mais a padronização posicional, a imagem da asa está sempre centrada. A figura 4.2 apresenta a conversão da imagem para o formato de entrada de rede neuronal após o recorte.

Foram testados diversos tamanhos de imagens e a que apresentou melhores resultados como parâmetro de entrada foi 400x400 pixeis. Ao reajustar o tamanho da imagem, não existe deformação no formato da asa, apenas em sua escala, para manter as características de cada subespécie intacta.

## 4.3 Deteção de características

Existiram três abordagens para solucionar a deteção de *landmarks*, a primeira delas baseia-se no uso de abordagens convencionais de processamento de imagem encontradas na biblioteca OpenCV[33], a segunda abordagem é constituída de implementações de redes convolucionais[31] para solucionar o problema, uma abordagem mais atual para os problemas de deteção de *landmarks*, a terceira abordagem é constituída da implementação de uma U-net[9], que apesar de não estar diretamente associada ao problema, apresenta características vantajosas para sua implementação.

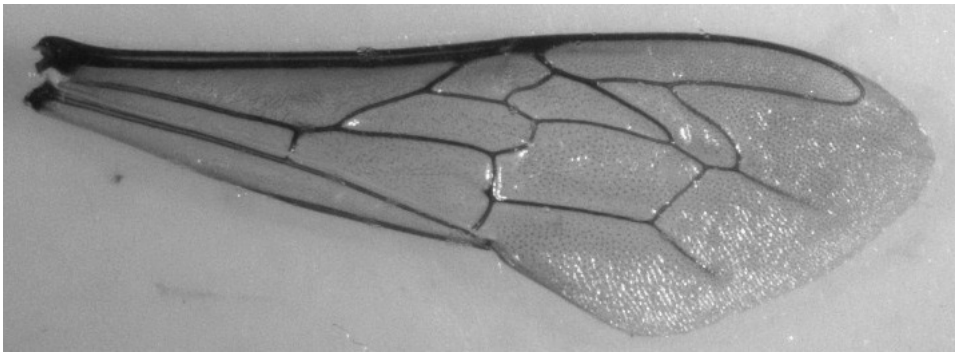

Figura 4.6: Exemplo de imagem do *dataset* em tons de cinzento.

### 4.3.1 deteção de *Landmarks* com Processamento de Imagem

Inicialmente, durante a análise do problema, foram feitos experimentos sem o uso de *deep learning* para averiguar sua viabilidade apenas com técnicas mais clássicas de processamento de imagem.

Para extrair os *landmarks* nesse caso, a primeira abordagem buscou realizar uma separação entre a imagem da asa e do fundo da imagem (*background*), para isso foram utilizados limites(*thresholds*[33]) manualmente nos valores das cores das imagens para realizar a separação, onde de um lado do *threshold* ficaria o fundo e do outro a asa, dessa forma, possibilitando extrair apenas o formato da asa, porém um simples *threshold* manual não foi capaz de criar uma separação adequada.

Em seguida foi utilizado um *threshold* adaptativo, que apesar de demonstrar bons resultados para imagens individuais, não apresentava resultados consistentes ao ser testado com todas as imagens dos *datasets*. A figura 4.7 apresenta um resultado a partir do *threshold* adaptativo.

Considerando a possibilidade de estabelecer uma separação consistente entre asa e o fundo da imagem, foram feitos testes de limiarização nas imagens, para averiguar o deslocamento dos pontos e as características restantes após o processo.

A figura 4.6 apresenta um exemplo do *dataset* em níveis de cinzento.

A figura 4.7 apresenta a saída de um *threshold* adaptativo realizado na figura 4.6 que separa a asa do plano de fundo baseado na variação de cores. Através dela é possível

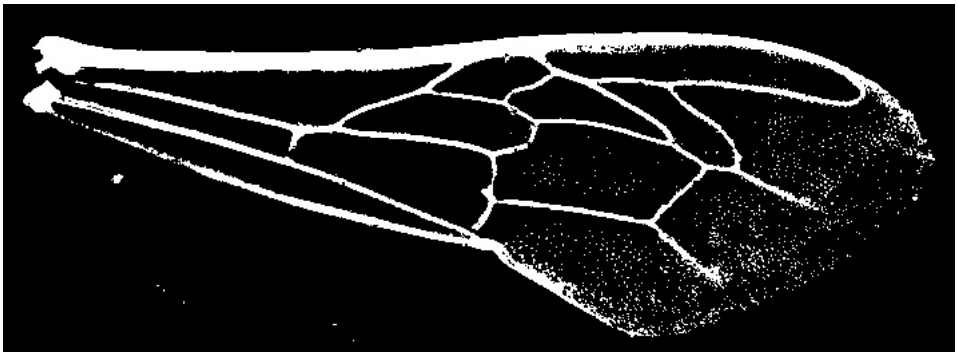

Figura 4.7: Exemplo do uso do *threshold* adaptativo

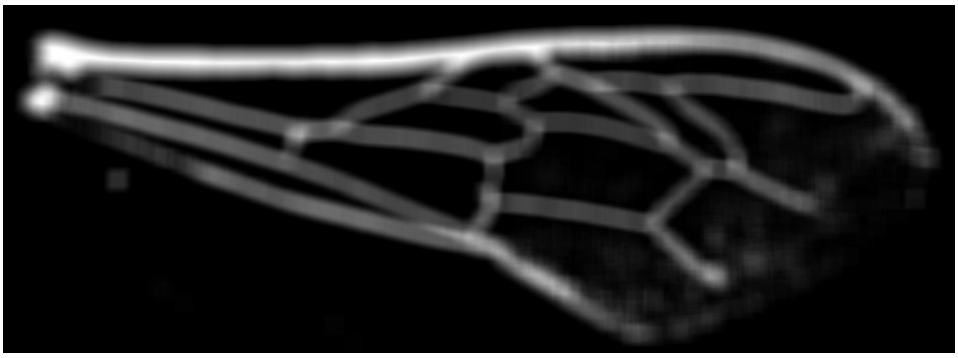

Figura 4.8: Exemplo de imagem ofuscada

ver a existência de diversos pontos brancos devido ao reflexo da luz na imagem original e que diversas conexões entre as nervuras foram desfeitas, podem ser reparadas também manchas fora da asa.

As figuras 4.8 e 4.9 são uma sequência de operações com foco em remover pequenas manchas e reunir as nervuras que foram separadas. Na figura 4.8 foi realizado um filtro 2D[33] que cria uma convolução manual na figura 4.7 criando um efeito de borrão na imagem. A partir da figura 4.8 foi aplicado um *threshold* binário, gerando a figura 4.9 que por sua vez foi capaz de reunir as nervuras e remover algumas manchas, porém acabou engrossando demais as conexões entre nervuras incapacitando uma deteção precisa.

Ao aplicar o medianBlur[33] na figura 4.7 foi possível obter a figura 4.10, que por sua vez tem menos problemas com manchas e ruídos, porém as conexões entre as nervuras ficaram ainda mais comprometidas e nesse estado, utilizar o mesmo processo utilizado nas figuras 4.8 e 4.9 não se mostra tão eficiente quanto anteriormente, devido a remoção do

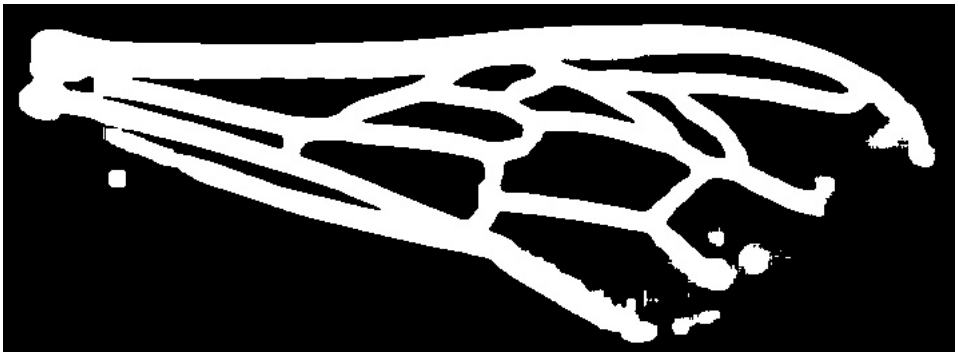

Figura 4.9: Exemplo de *threshold* após ofuscar imagem

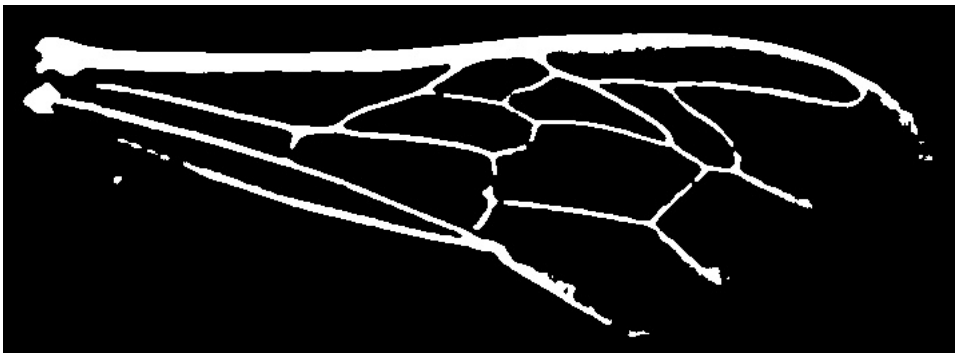

Figura 4.10: Exemplo de remoção de particular através do medianBlur

ruído.

O processo de limiarização[33] tem como objetivo manter apenas a estrutura base da asa, tornando mais fácil sua análise mantendo apenas seu "esqueleto", de forma a imagem manter apenas as formas principais de sua composição, nesse problema, o processo de limiarização tem como objetivo criar o esqueleto da asa e as conexões entre as nervuras seriam próximas dos *landmarks* procurados, tornando a deteção muito mais fácil. A figura 4.11 é o resultado de uma limiarização aplicada à figura 4.10, porém devido a perda na conexão das nervuras, não foi possível estabelecer conexões consistentes na imagem limiarizada, fora isso, o processo de limiarização também criou novamente manchas na imagem.

Nas imagens bem sucedidas nos *thresholds* e nas limiarizações, foi testado o detector de bordas do OpenCV chamado goodfeaturestotrack[33], porém ele geralmente detetam características a mais do que as necessárias e deixam muitas *landmarks* indetetadas. A

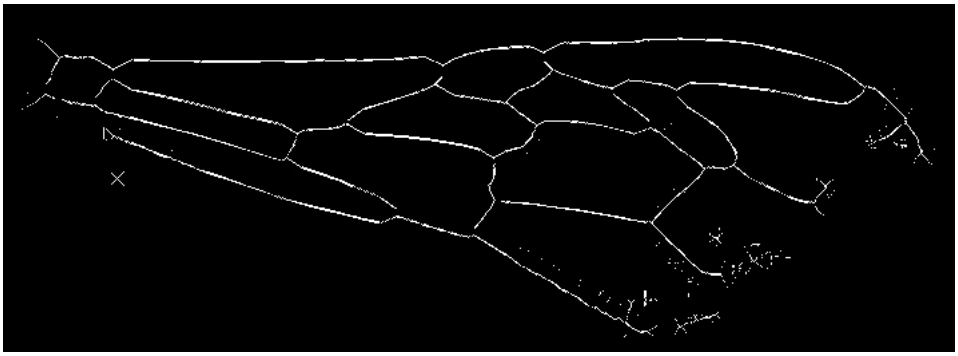

Figura 4.11: Exemplo de limiarização

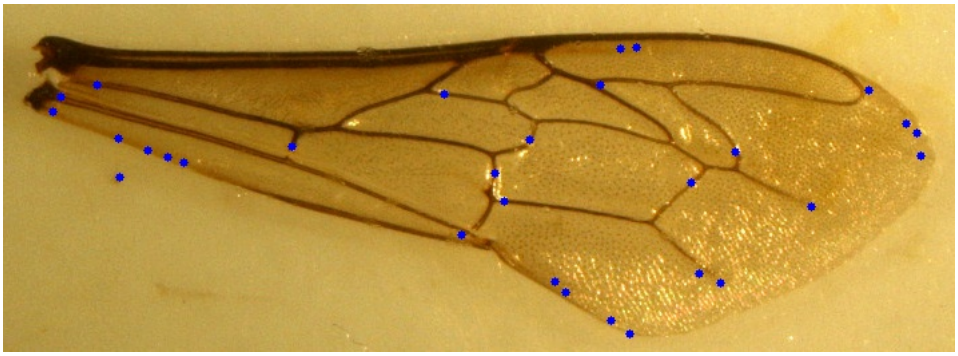

Figura 4.12: Exemplo de deteção a partir da função goodFeaturesToTrack[33]

figura 4.12 apresenta a versão colorida da figuras 4.6 com os pontos detetados pela função goodfeaturestotrack sobre sua versão limiarizada(figura 4.11). É possível reparar que existem pontos corretos que foram detetados, porém os pontos estão majoritariamente deslocados ou não fazem parte dos *landmarks* que devem ser identificados.

### 4.3.2 CNNs para Deteção de *Landmarks*

Uma abordagem muito comum atualmente no processo de reconhecimento facial é o uso de CNNs para detetar *landmarks*. Nesses casos, os modelos utilizados recebem uma imagem como entrada e sua saída são as coordenadas das características. Esse tipo de processo apresenta diversas vantagens sobre os detectores de características do openCV, pois ao determinar a quantidade de *outputs* não existirão detecções que variam em quantidade de elementos e o processo de treinamento permite que a rede não só encontre as características

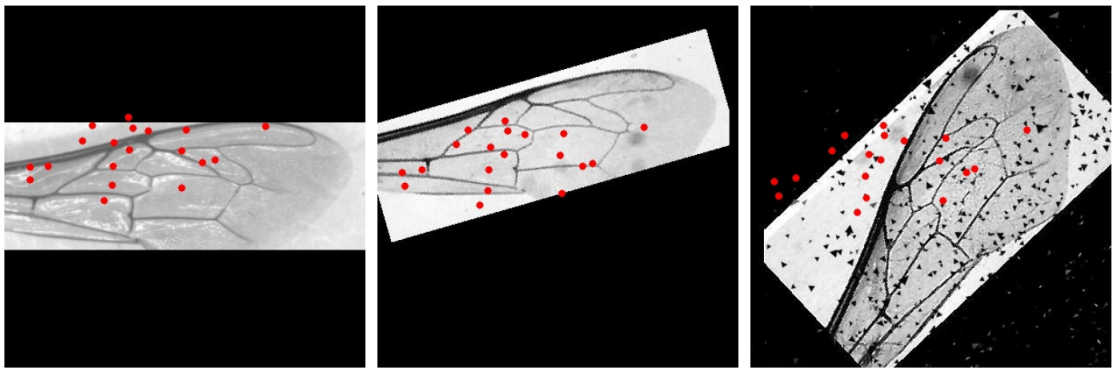

Figura 4.13: Exemplo de deteção com o modelo de CNN que deteta todos os pontos

procuradas como as entrega de forma estruturada, dessa forma, não é necessário ordenar os *landmarks*.

Foram testados dois métodos diferentes para essa abordagem, modelos completos, que analisam toda a imagem e tem como saída todas as *landmarks* buscadas e modelos especialistas, que ao analisar a imagem buscam uma parcela pequena das características, no caso, cada modelo especialista busca por uma *landmark* dentre as 19, gerando 19 modelos.

Os modelos completos tenderam a acertar diversos *landmarks*, porém não conseguiram entregar 19 pontos corretamente na maioria dos casos. Já os modelos especialistas foram muito suscetíveis a *overfitting*.

A figura 4.13 mostra detecções feitas a partir da CNN que tem como objetivo detetar todos os 19 pontos de uma vez, a saída da rede foi plotada sobre a imagem de entrada da rede por razões de visualização. É possível ver que os 19 pontos se assemelham levemente a proporção dos *landmarks* objetivo, porém não correspondem a imagem em ângulo e em proporção, também é possível perceber que a saída da rede para as três imagens, embora diferentes, tem praticamente a mesma saída da rede, o que indica *overffiting*.

A figura 4.14 apresenta a saída de um dos modelos especialistas, plotada sobre a imagem de entrada da rede. Esse modelo em especifico, tem foco em encontrar o *landmark* 19 (que pode ser encontrado na figura 1.2). É possível ver que nesse caso, a saída também se mantém no mesmo local apesar das imagens de entrada serem diferentes.

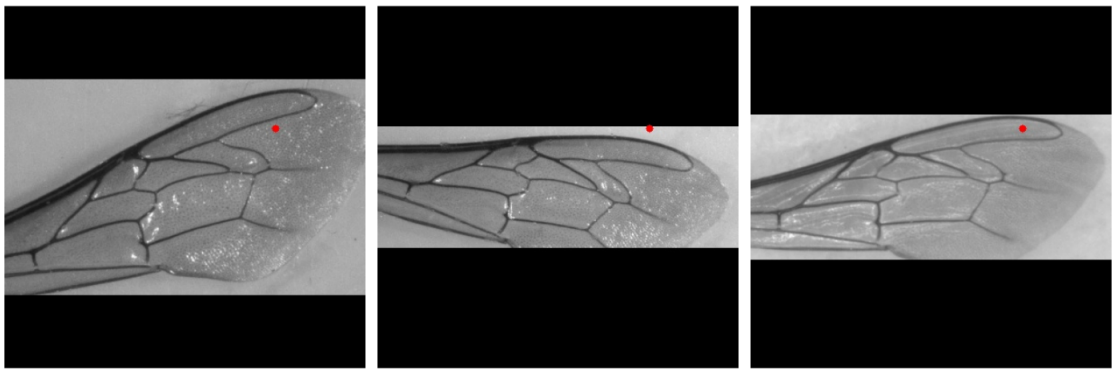

Figura 4.14: Exemplo de deteção com o modelo especialista

Nenhuma dessas redes apresentou resultados confiáveis de forma a possibilitar a implementação desses modelos na versão final do projeto. As redes também ficaram muito pesadas com o passar das implementações, de forma a ficaram consideravelmente mais pesadas ao longo do desenvolvimento dos modelos, de forma a consumir muito mais memória e processamento.

### 4.3.3 U-net

A U-net[9] não é um modelo tradicional em processos de deteção de *landmark*, porém, devido à sua precisão com segmentação de imagens e o fato de ser muito leve, foi considerado como uma possibilidade para solução do problema de deteção de *landmarks*. O modelo foi utilizado com o intuito de destacar as regiões de interesse, para que com outras técnicas fosse possível extrair as *landmarks*. O objetivo inicial era de fazer uma segmentação exata, onde a máscara de saída da rede apresentasse apenas 19 pontos, onde cada um deles fosse uma das *landmarks*, porém os melhores resultados da U-net foram com a segmentação da região em torno das *landmarks* e os pontos sendo extraídos das máscaras posteriormente.

#### Geração de máscaras

Para treinar a U-net é necessário gerar máscaras para serem comparadas com a saída da rede. Cada máscara se refere a uma única imagem no *dataset* e apresenta cor preta

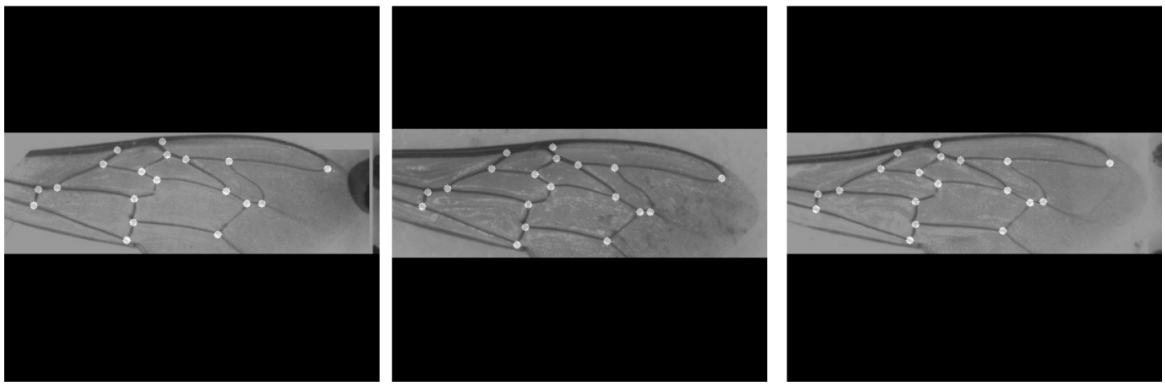

Figura 4.15: Máscara sobreposta a sua imagem correspondente.

e onde as *landmarks* referentes aquela imagem se encontram, existe um círculo branco, essas máscaras servem como a resposta que é esperada pela rede, para sua imagem correspondente. Foram feitas imagens de proporções iguais as da entrada da rede (400x400) pretas(valor igual a zero), onde nas regiões de interesse eram desenhados círculos brancos (valor igual a 1). O tamanho dos círculos influencia diretamente na efetividade da rede, os tamanhos mais adequados de círculos para as máscaras foram dos de raio 3 e 4, onde 4 apresentou melhores resultados.

Círculos de tamanho inferior a 3 eram pequenos demais e passavam a ser ignorados pela rede, círculos de tamanho superior a 4 ficam grandes demais e comprometem a precisão dos pontos detetados, em diversos casos fundindo regiões em uma única deteção, impossibilitando a deteção de todas as 19 características. A figura 4.3 apresenta exemplos de mascaras enquanto a figura 4.15 apresenta a máscara sobreposta a sua imagem original.

### Parâmetros da U-net

Para garantir bons resultados foram feitas alterações em características da U-net para responder de forma adequada à demanda. A implementação original da U-net utilizava kerneis de tamanho 3, porém essa versão tinha um foco maior em detetar bordas[9], para detetar os pontos, kerneis maiores poderiam entregar resultados melhores devido a ter uma visão mais ampla do contexto ao redor do objetivo. Foram testados kerneis de tamanhos 3, 5 e 7, onde a versão com kerneis de tamanho 5 tiveram o melhor resultado.

Embora não sejam adicionadas novas camadas à U-net, a troca de tamanho dos kerneis tem grande impacto no processamento da rede, onde aumentar os kerneis de 3x3 para 5x5, torna a rede em torno de três vezes mais pesada, o que pode vir a ser um problema quando se trata de consumo de memória e velocidade de desempenho.

Para aprimorar um pouco mais o resultados, foram implementados também dois *callbacks*. O *early stopping*[51] com objetivo de encerrar o treinamento do modelo após ele deixasse de aprimorar a sua loss, com intuito de evitar *overfitting* (evitando treinamentos demasiadamente grandes), reduzir o tempo de treinamento, interrompendo o processo assim que for considerado que o modelo convergiu. O *reduce on plateau*[38] foi utilizado com diversos objetivos, pois ao reduzir o tamanho dos passos dados durante o treinamento é possível aprimorar a precisão do algoritmo de forma segura.

Devido as características do problema, onde é preciso destacar uma área muito pequena foi necessário utilizar uma função de *Loss* diferente da original. O objetivo principal da função de *loss* utilizada na implementação original da U-net tem como um de seus maiores objetivos a separação adequada entre os objetos detetados, de forma a impedir que essas detecções se misturem, uma característica secundária para o problema atual, pois como é possível ver nas figuras 1.2 e 4.15, os *landmarks* que são procurados pela rede tem uma distância considerável entre si, dificultando a chance de que se misturem. O maior problema encontrado pelo nosso modelo, é o fato da área segmentada ser muito pequena e a rede tender a ignorá-la. Para resolver esse problema, foi necessária a implementação de uma função de *loss* que utilize de pesos.

### Função de *Loss*

O balanceamento de classes dentro de uma imagem é muito importante para auxiliar o processo de segmentação, nas primeiras implementações do modelo ele tendia a convergir em gerar imagens completamente pretas pois as imagens era desbalanceadas. Uma imagem 400x400 tem 160000 pixeis, e em máscaras com 19 círculos de raio 3, apenas 1687 desses pixeis são brancos (regiões a serem segmentadas) e em máscaras com círculos e raio 4, 3000 pixeis, onde pixeis brancos representam menos de 1% dos pixeis totais nos

dois casos.

Para aumentar a relevância dos pixeis brancos foi necessário o uso de pesos sobre à classe dos pixeis brancos, porém a função de *loss* que estava a ser utilizada (*binary crossentropy*) não apresentava uma implementação própria com pesos que lide com um problema de segmentação. Dessa forma foi necessário gerar uma nova função de *loss*, baseada na anterior que fosse capaz de lidar com pesos. Para isso, foi utilizada uma função de *loss* de dentro do próprio tensorflow que já havia implementação de pesos chamada *weighted categorical crossentropy*, que adiciona pesos à função *categorical crossentropy*, a chamada da função *categorical crossentropy* foi alterada para uma chamada da função *binary crossentropy*, dessa forma, gerando uma nova função(*weighted binary crossentropy*). Com essa nova função de *loss*, a partir de uma boa implementação de pesos, foi possível desenvolver uma U-net capaz de segmentar regiões muito pequenas dentro de uma imagem.

Diversas configurações de pesos foram testadas, a que obteve melhores resultados foi manter os pesos da classe preta, e aumentar em 50 vezes o peso da classe branca.

A equação 4.1 descreve a função de (*binary crossentropy*), sem pesos, onde $N$ corresponde a quantidade de elementos, $y$ corresponde ao valor da classe (0 ou 1) e $p$ à probabilidade de $y$ ocorrer.

$$Loss = -\frac{1}{N}\sum_{i=1}^{n} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i)) \tag{4.1}$$

A alteração na função corresponde a uma adição do peso $w$, como pode ser visto na equação 4.2.

$$Loss = -\frac{1}{N}\sum_{i=1}^{n} (y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_1))) \cdot w_{y_i} \tag{4.2}$$

### 4.3.4 Processamento de Pontos

Uma vez que as *landmarks* foram extraídas é necessário processá-las de forma a ficarem padronizadas. Como a U-net ao segmentar as *landmarks* não as classifica (indicando dentre as 19 *landmarks* detetadas, qual sua classe dentre as *landmarks* da figura 1.2),

é necessário realizar uma extração de pontos da máscara de forma com que fiquem na mesma ordem, para que as entradas dos modelos de classificação fiquem padronizadas corretamente.

Essa padronização é necessária pois os algoritmos de classificação (para as subespécies) utilizados exigem que a entrada seja padronizada, do contrário, não estão comparando características diferentes e estão assumindo que são as mesmas, dessa forma, comprometendo o resultado.

### *Principal Component Analysis* (PCA)

A primeira ação para garantir uma extração padronizada dos *landmarks* é o uso do PCA[26], que pode ser utilizado para identificar o ângulo de distribuição dos valores positivos da máscara gerada pela U-net e com esse ângulo, rotacionar a máscara, padronizando o ângulo de distribuição dos valores.

### Extração de Pontos

A partir das máscaras geradas pela U-net é necessário realizar uma extração dos pontos detetados por ela. Como a máscara gerada pela U-net não é composta apenas pelas coordenadas das *landmarks* e sim por toda região ao redor dela, é necessário detetar as regiões destacadas pela U-net e extrair as coordenadas das regiões segmentadas. Como as máscaras são baseadas em círculos com a região de interesse no centro, a abordagem mais simples é de detetar o centro das áreas segmentadas pela U-net. Para isso foi utilizado o *Blob Detector* implementado na biblioteca OpenCV[33].

Devido a algumas imagens serem muito variantes em características, a U-net mesmo ao localizar alguns pontos, segmenta uma região muito pequena que passa despercebida pelo *Blob Detector*. Para resolver esse problema são utilizadas algumas condições que ajudam a assegurar a precisão dos pontos:

1. Detetar *landmarks* na máscara;

2. Caso sejam 19 *landmarks* detetados, utilizar esses 19 *landmarks*;

3. Caso sejam menos, aplicar dilatação na máscara e averiguar novamente, caso sejam 19, utilizar esses 19 *landmarks*;

4. Se em nenhum dos dois casos foram detetados 19 *landmarks*, essa deteção falhou.

Uma vez com as 19 *landmarks*, é preciso ordená-las de forma a garantir que no processo de classificação, elas sejam comparadas com suas respectivas *landmarks*. Para isso as *landmarks* são colocadas em uma lista, que é ordenada de forma crescente no eixo X. Após a ordenação, caso existam valores do eixo X que sejam muito próximos (distância de 5 pixeis), o eixo Y é utilizado como critério de desempate, onde o maior Y fica na frente no vetor.

A função de buscar *outliers* do MorphoJ[29] foi utilizada como critério para avaliar se ouve troca nos pontos detetados, essa função, após encontrar a posição média de cada ponto em um *dataset*, apresenta aqueles que mais se distanciam da média.

Dentre a deteção de pontos, existiam diversas regiões de *landmarks* onde, devido a proximidade, eles alteravam constantemente em ordem, porém devido ao forte padrão apresentado entre esses pontos, foi possível fazer uma função capaz de analisá-los e reordená-los dentro de um padrão, tornando o *output* muito mais padronizado. A figura 4.16 apresenta uma das imagens e sua discrepância com a média, pode ser visto que os pontos 7, 8 e 9 foram trocados entre si e os pontos 10 e 11, também. As regiões que apresentam maior variação foram: pontos entre 4 e 9, 10 e 12 e entre 14 e 16. Essas regiões são reordenadas novamente (pelo eixo y) separadas da lista original e depois colocados de volta para aprimorar a precisão da ordenação.

Após essa ordenação, no entanto, ainda existiram alguns pontos que eram invertidos com certa frequência, a partir disso, foi feita mais uma ordenação, que dessa vez, olhava dois únicos pontos e comparava-os no eixo y. Os pontos que eram trocados foram: 1 e 2, 14 e 16, e 6 e 9.

O processo de ordenação de pontos pode ser descrito da seguinte forma:

1. Ordenar vetor de pontos em X (ordem crescente);

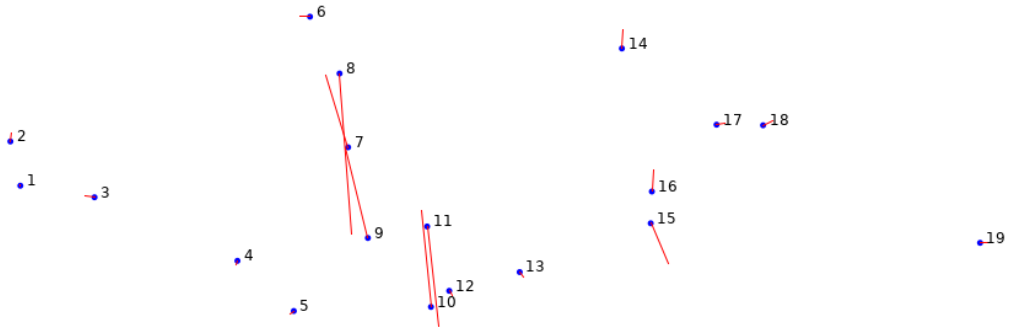


Figura 4.16: Resultado do MorphoJ[29] para deteção de *outliers* (os pontos aparecem de cabeça para baixo no MorphoJ devido ao programa inverter a posição do eixo y em comparação ao openCV e outras bibliotecas em python).

2. Checar *landmark* próximos (proximidade de 5 pixeis) e fazer o desempate no eixo Y (ordem crescente);

3. Selecionar 3 "regiões"(pontos: 4-9, 10-12, 14-16), ordená-las em Y (ordem crescente) e colocá-las de volta ao vetor.

4. Checar pontos trocados (1 e 2, 6 e 9, 14 e 16), ordenar duplas em Y (ordem crescente) e colocar de volta no vetor.

### Análise Procrustes

O processo de análise procrustes[28] é realizado em duas partes do processo. A primeira é a etapa de extração de características, onde com os *landmarks* detetados é realizado uma análise procrustes generalizada para criar uma referência média que represente todo o *dataset*, a partir dessa referência todos os elementos são padronizados de forma que possam ser efetivamente classificados.

A segunda etapa onde a análise procrustes é utilizada é durante o processo de classificação de um novo elemento, onde ele é ajustado com a referência gerada anteriormente e então classificado.

A análise procrustes também foi utilizada para validar a precisão dos *landmarks* extraídos com os originais, utilizando a distância procrustes como critério de medida.

### Avaliação de Precisão

Para avaliar a qualidade dos *landmark*s extraídos foi implementada uma comparação entre o objeto obtido na saída da rede e os objetos marcados manualmente. Para isso, o objeto original é superposto sobre o objeto que é a saída da rede utilizando uma análise procrustes, dessa forma é possível compará-los de forma mais eficiente, o objeto original também precisa passar por um processo de reordenação dos *landmarks*, para garantir que os *landmarks* estão sendo corretamente comparados com seus correspondentes no novo objeto. Com esses dois objetos superpostos, são extraídas características de ambos, essas características são compostas da distância euclidiana de um *landmark* para os outros 18 que compõe o objeto, dessa forma, cada objeto tem 171 características únicas em um grupo de 342 características (considerando repetições). Para descobrir a quantidade de elementos únicos $e$ em um conjunto de n pontos, em uma combinação de k pontos, basta utilizar a equação 4.3[52].

$$e = \frac{n!}{k!(n-k)!} \tag{4.3}$$

Uma vez que as características foram extraídas a semelhança entre elas é calculada com uma divisão dos valores das características, devolvendo um valor entre 0 e 1, representando a relação entre ambos, onde 0 representa nenhuma semelhança e 1 representa 100% de semelhança. Para garantir um número entre 0 e 1, a divisão dos valores das características antes são checados, o maior número entre os dois torna-se o divisor e o menor torna-se o dividendo. Esse método não leva ângulo em consideração.

Para exemplificar o método, considere dois objetos a serem comparados $A$ e $A'$, compostos de 3 pontos cada ($(a, b, c)$ e $(a', b', c')$): $A((1,1),(3,3),(5,3))$ e $A'((1,1.2),(2.9,3),(5,3.5))$.

Cada um desses objetos possui 3 distâncias relativas a seus pontos: $a \to b$, $a \to c$ e $b \to c$. Dessa forma, obtemos $A(a \to b(2.828427), a \to c(4.472136), b \to c(2))$ e $A'(a' \to$

$b'(2.61725), a' \rightarrow c'(4.614109), b' \rightarrow c'(2.158703))$.

Ao calcular a diferença entre as distâncias, obtemos: $D((0,925),(0,969),(0,926))$ com uma média de $0,94$, ou seja, uma semelhança de 94% entre os objetos $A$ e $A'$.

### 4.3.5 Dados gerados

A partir da extração de *landmarks*, diversos arquivos são gerados com intuito de analisá-los em outros programas, para testes de classificadores e avaliação dos resultados obtidos.

As imagens com 19 *landmarks* detetados são salvas em arquivos (no formato .tpf para ser analisados no morphoj) correspondentes as classes (subespécies) da imagem original, também é gerado um arquivo (.csv) com todas as detecções com e sem a análise procrustes generalizada[28].

A análise procrustes generalizada cria uma referência ao grupo de objetos analisados, o programa salva uma referência para cada classe e uma referência geral.

Durante o processamento das imagens, as classes são referenciadas por seu índice dentro do vetor, então é importante saber a qual subespécie cada classe se refere, para isso é também salvo um arquivo que indica o nome da subespécie de cada classe.

## 4.4 Classificação

O processo de classificação foi composto do uso de diversos algoritmos testados. Como muitas classes apresentavam poucos exemplares no dataset, o uso de redes neuronais não torna-se viável para uma classificação efetiva, devido a necessidade de diversos exemplares de cada classe[31].

Os algoritmos testados para classificação foram SVM[31], Naive Bayes[31] e Random Forest[32], todos algoritmos de *machine learning* que apresentam possibilidade de bons resultados com baixa quantidade de elementos no *dataset*.

Foi utilizada para a implementação desses modelos a biblioteca scikit-learn[47], que apresenta modelos desses algoritmos padronizados para facilitar o processo de treinamento, de forma a não ser necessário alterar o formato de entrada em nenhum desses

algoritmos. Uma vez que os algoritmos foram implementados, foram realizados testes com variações em suas características para encontrar os modelos que obtém maior precisão no processo de classificação.

# Capítulo 5

# Testes/Avaliação/Discussão

## 5.1 Detector de Asas

A partir do *dataset* 1 diversos modelos de detectores de objetos foram retreinados através de transferência de aprendizado[53], devido à serem modelos bem estabelecidos, foi apenas necessário realizar a transferência de aprendizado e averiguar quais seriam os mais adequados para solução do problema. Para isso os critérios para averiguar qual o melhor modelo para o problema foram: precisão e velocidade.

A tabela 5.1 apresenta a precisão de cada modelo e sua velocidade em imagens por segundo. É possível observar uma precisão muito alta de todas as redes treinadas, porém, a mais rápida foi a mobilenet[22], e devido a alta precisão apresentada pela rede, ela foi a escolhida para a implementação final do projeto.

| Modelo | Precisão | Imagems por Segundo |
|---|---|---|
| Mobilenet[22] | 97.5% | 24 |
| Faster rcnn nas[20] | 95% | 0.6 |
| Faster rcnn inception resnet v2 atrous coco[21] | 95% | 1.8 |
| YoloV3/Darknet[23] | 90% | 18 |

Tabela 5.1: Detectores de objetos, precisão e velocidade.

## 5.2 Detectores de *Landmarks*

Para criação do detector de *landmarks* foram utilizados diversos métodos, tanto para averiguar sua complexidade quanto para identificar as melhores maneiras de resolvê-lo. Para averiguar o melhor método as métricas utilizadas foram: Quantidade de *landmarks* detetadas, a discrepância entre os pontos detetados e os originais (utilizando a implementação utilizada pela função do procrustes[28] da biblioteca Scipy[30]) e a velocidade.

A figura 5.1 mostra a saída de U-net, e uma segunda versão da imagem, onde a máscara gerada pela U-net fica sobreposta a imagem original, para permitir melhor visualização do resultado obtido. É possível ver que o modelo também aprendeu a lidar com problemas como poeira e alterações de ângulo.

É importante ressaltar a diferença nos resultados em cada um dos *datasets* utilizados. Com o *dataset* 1 (que apresenta imagens em alta qualidade e com maior padronização na imagem), a precisão relacionada a deteção dos 19 *landmarks* é muito alta, com 98,5% (utilizando a melhor rede implementada) das imagens apresentando uma deteção bem sucedida, porém com o *dataset* 2 (com imagens mais antigas, sujeira e ruído), a precisão cai para 78%, o que também dificulta o processo de classificação das classes presentes no *dataset* 2.

### 5.2.1 Resultados: Processamento de Imagem

As implementações sem uso de técnicas de aprendizagem profunda não apresentaram bons resultados, como pode ser visto na figura 4.12, as detecções não indicam os *ladmarks* desejados. O maior motivo da falta de precisão encontra-se na parda de características durante o processo de separação da asa com o fundo da imagem, dessa forma não estabelecendo boas características (bordas) a serem detetadas.

### 5.2.2 Resultados: CNNs

As CNNs implementadas para deteção de *landmarks* também não apresentaram bons resultados, apresentado *overfitting* nos modelos treinados (como demonstrado no capítulo

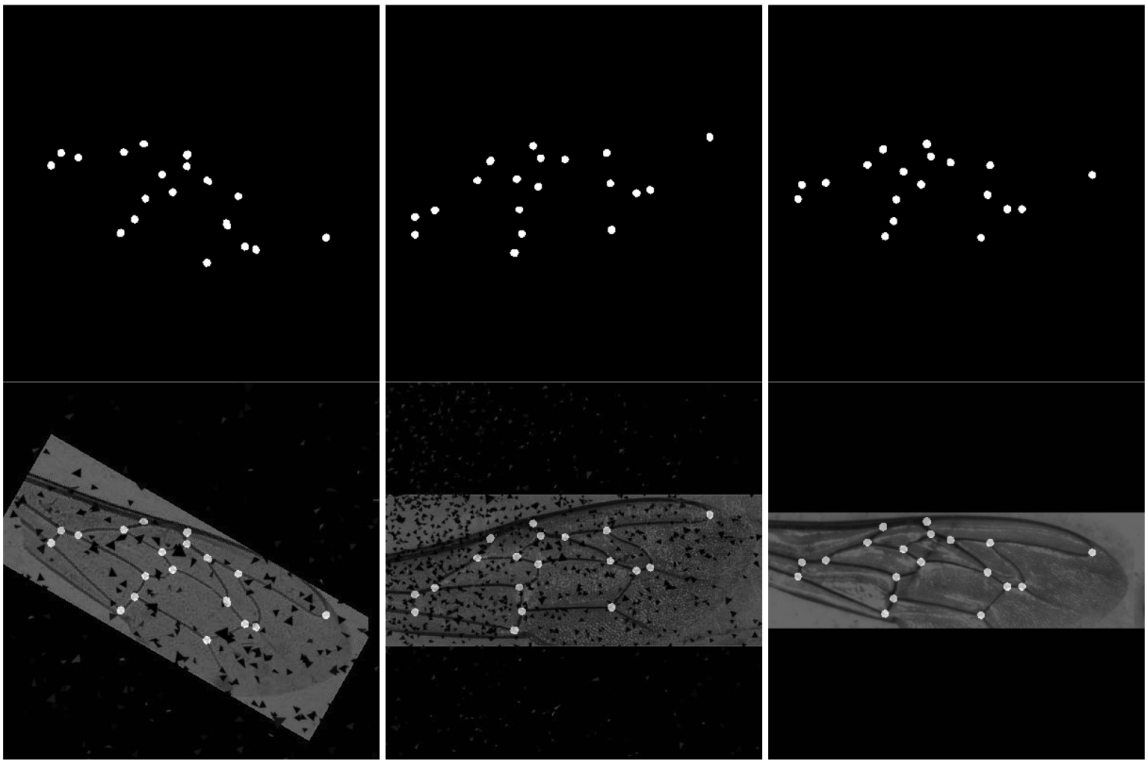

Figura 5.1: Saída da U-net após o treino, acima as máscaras, abaixo as máscaras sobrepostas sobre a imagem original.

4.3.2), majoritariamente devido a poucos dados para o treinamento e ao fato das imagens de entrada estarem padronizadas permitiu a rede a "chutar"os mesmos valores e constantemente conseguir valores muito próximos dos desejados, de forma a não aprender de fato a analisar as características da imagem.

### 5.2.3 Resultados: U-net

A tabela 5.2 apresenta os resultados obtidos a partir dos modelos U-net testados durante o desenvolvimento do projeto. A primeira coluna "Raio", representa o raio da *landmark* utilizado nas máscaras de treinamento da rede, a coluna "Imagens Alteradas"indica se no *dataset* as imagens foram rotacionadas ou deslocadas durante o processo de aumento de dados, "Poeira"corresponde a implementação de poeira artificial e ruído no *dataset*, "Pesos"implica se o modelo foi treinado com a nova função de *loss* e precisão indica o resultado do cálculo de precisão de cada modelo da U-net baseada na contagem de sucessos (imagens com 19 *landmarks*) dividido pelo total de detecções. É possível a partir desse modelo, identificar o impacto das características implementadas, tanto na rede, quanto no pré-processamento. A precisão apresentada nessa imagem é a divisão de verdadeiros positivos, pela quantidade de elementos a serem detetados. Um verdadeiro positivo nesse caso, é considerador correto quando, em uma imagem, ele consegue detetar um quantidade de *landmarks* igual a 19. O modelo com maior precisão apresenta todas as características implementadas no pré-processamento, um kernel de tamanho 5x5 e o raio dos *landmarks* nas máscaras utilizadas durante o treinamento tinham tamanho igual a 4.

A velocidade da U-net implementada é de aproximadamente 90 imagens por segundo.

Para averiguar a qualidade dos pontos foi utilizado o método apresentado no capítulo 4.3.4 Avaliação de Precisão. É possível observar que existe pouca variação na precisão dos *landmarks*, onde a maior precisão corresponde ao *landmark* 19, de 97.5% e a menor precisão, no *landmark* 9, com 90% de precisão, uma diferença de apenas 7.5%.

É importante ressaltar que os *landmarks* apresentados na tabela 5.3 não correspondem (em nomeação) aos números apresentados por Nawrocka na figura 1.2, como a ordenação

| Raio | Imagens Alteradas | Poeira | Kernel | Pesos | Precisão |
|---|---|---|---|---|---|
| 3 | Não | Não | 3x3 | Não | 77.4% |
| 4 | Sim | Não | 3x3 | Não | 86.4% |
| 3 | Sim | Não | 3x3 | Não | 86.8% |
| 3 | Sim | Não | 3x3 | Não | 86.8% |
| 4 | Sim | Não | 5x5 | Não | 81.9% |
| 3 | Sim | Sim | 5x5 | Não | 70.7% |
| 3 | Sim | Sim | 5x5 | Sim | 89.2% |
| **4** | **Sim** | **Sim** | **5x5** | **Sim** | **91.8%** |
| 3 | Sim | Sim | 7x7 | Sim | 83.1% |

Tabela 5.2: Precisão das implementações de U-net.

| ***Landmark*** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Precisão** | 96.8% | 97% | 96.3% | 95.4% | 91.1% | 93.2% | 93.9% | 95% | **90%** | 93.7% |
| ***Landmark*** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **Total** |
| **Precisão** | 92.4% | 92.6% | 93.7% | 94.5% | 93.3% | 93.1% | 95.8% | 96.2% | **97.5%** | **94.3%** |

Tabela 5.3: Precisão individual de cada *landmark* e precisão total a partir da saída da U-net.

dos *landmarks* detetados foi feita na ordem crescente no eixo x, a nomeação de saída dos *landmarks* encontrados ficam numa ordem diferente dos números utilizados por Nawrocka. A imagem que corresponde aos números utilizados na tabela 5.3 são os números da figura 4.16.

## 5.3 Classificação

Durante o processo de classificação existiram empecilhos que não possibilitaram realizar uma boa classificação com todas as 26 classes presentes no *dataset*, majoritariamente devido a baixa quantidade de amostras de algumas classes. Os algoritmos (SVM, random forest e naive bayes) não foram capazes de entregar melhores resultados a partir de uma quantidade elevada de classes, em treinos com mais de 3 classes o algoritmo raramente responde subespécies de fora das duas maiores classes(0 e 1). Em casos onde existe um balanceamento de classes, removendo elementos de classes desproporcionais, os classificadores ainda apresentaram baixa precisão.

Para classes com uma quantidade maior de elementos foi possível obter resultados mais consistentes e de acordo com os resultados de [1]. Utilizando apenas o algoritmo de Naive Bayes[31] nas duas maiores classes, que correspondem a 3518 (do Açores) e 2251(da Península Ibérica) imagens respectivamente, foi possível obter 92% de precisão na em sua classificação.

A implementação da SVM por sua vez, foi capaz de classificar com 87% de precisão com 3 classes (asas de açores, península Ibérica e *A.m. anatoliaca* ) e com 67% de precisão ao adicionar uma quarta classe (*A.m.litorea*).

A implementação da random forest não apresentou bons resultados, sendo capaz apenas de classificar duas classes (asas de açores e da península Ibérica) com uma precisão de aproximadamente 56%.

## 5.4 Velocidade

A implementação final do projeto, com uso da mobilenet e da U-net, tem ótimo desempenho, com uma velocidade de 11 imagens por segundo para realizar o processo completo. É importante ressaltar que a GPU utilizada para realizar o processo é uma GTX 1080 ti, com 11 gigabites de memória.

O uso da U-net, em conjunto com a alteração de ângulo apresenta uma velocidade de 90 imagens por segundo, porém o processo de extrair os pontos das máscaras com o *Blob Detector*[33], tem uma velocidade de 31 imagens por segundo.

A mobile net, em conjunto com a adição de filtros e escala da imagem, teve uma velocidade de 20 imagens por segundo.

# Capítulo 6

# Conclusões

Nesse trabalho foi abordada a produção de um sistema automático para classificação de abelhas através da morfologia das nervuras das asas, é possível ver que o processo de extração de *landmarks* apresentou ótimos resultados de velocidade, consumo de recursos e precisão na deteção de *landmarks*.

Computadores simples, embora não consigam o desempenho de 11 imagens por segundo como apresentado nos nossos resultados, ainda serão capazes de utilizar um processo de maneira eficiente.

A deteção de objetos implementada apresenta resultados eficientes e de acordo com as aspectivas. Todas as implementações apresentaram bons resultados de forma a possibilitar a escolha do modelo que consome a menor quantidade de recursos (mobilenet[22]).

É possível concluir que a extração dos *landmarks* é possível de ser realizada com uma rede U-net[9], embora ela não seja originalmente construída para isso. O processo de extração de *landmarks* através de processos que não fazem uso de redes neuronais apresentam um desempenho muito abaixo do esperado e não apresentam consistência, de forma a não ser uma boa abordagem para a solução do problema. O uso de CNNs para solução do problema (sem considerar o uso da U-net) não apresentou bons resultados, provavelmente devido à pequena quantidade de dados e devido à rede apresentar *overfitting*, a implementação de aumento de dados e o uso de modelos mais robustos pode possibilitar resultados melhores na solução do problema com essa abordagem, porém os modelos tendem a ficar

muito grandes (em consumo de memória) em comparação a U-net. É possível identificar nos resultados que o tamanho do kernel utilizado na unet tem impacto nos resultados da deteção de *landmarks*.

É possível identificar que variações na qualidade da imagem como baixa resolução, ruído, manchas e sujeira são capazes de interferir na deteção de *landmarks*, porém essas características não apresentam um impacto tão grande no processo de deteção de asas.

O processo de classificação, embora em alguns casos apresente bons resultados, ainda tem muito o que ser aprimorado, devido a falta de elementos presentes em algumas das classes (as classes *A. m. major, A. m. sahariensis e A. m. sicula* por exemplo, tem 20 ou menos elemento cada), tornado difícil sua classificação. Algumas subespécies apresentam maior dificuldade em sua classificação, como já pôde ser visto nos dados de Nawrocka[1].

Os dados gerados tanto quanto o código desenvolvido pode ser encontrado no Github pelo link: Github: Beeapp-landmark-detection

## 6.1 Trabalhos futuros

A maior necessidade desse projeto atualmente é o desenvolvimento de uma interface amigável ao usuário simples para garantir um uso acessível para que ele possa de fato ser utilizado.

Existem diversas análises no processo de classificação que ainda podem ser feitas. É interessante realizar testes da relação entre as linhagens de abelhas de forma a possibilitar outras classificações.

Para aprimorar os resultados da deteção de *landmarks* é possível testar outras configurações de redes neuronais, e adicionar aumento de dados no treinamento (nos outros modelos de CNN) de forma a dificultar a possibilidade de *overfitting*. Existem detalhes do pré-processamento que também podem aprimorar os resultados, como no aumento de dados, ao adicionar poeira artificial, alterar a opacidade das manchas de poeira, para simular manchas e regiões que não estão 100% tampadas pela poeira ou ruido das imagens.

# Bibliografia


[1] A. Nawrocka, İ. Kandemir, S. Fuchs e A. Tofilski, "Computer software for identification of honey bee subspecies and evolutionary lineages", *Apidologie*, vol. 49, n.º 2, pp. 172–184, abr. de 2018. DOI: 10.1007/s13592-017-0538-y. URL: https://doi.org/10.1007/s13592-017-0538-y.

[2] N. L. García, "The Current Situation on the International Honey Market", *Bee World*, vol. 95, n.º 3, pp. 89–94, 2018. DOI: 10.1080/0005772X.2018.1483814. eprint: https://doi.org/10.1080/0005772X.2018.1483814. URL: https://doi.org/10.1080/0005772X.2018.1483814.

[3] F. Ruttner, *Biogeography and Taxonomy of Honeybees*, 1st. New York: Springer-Verlag Berlin Heidelberg, 1988, ISBN: 978-3-642-72651-4.

[4] L. Garnery, M. Solignac, G. Celebrano e J. .-.-M. Cornuet, "A simple test using restricted PCR-amplified mitochondrial DNA to study the genetic structure ofApis mellifera L", *Experientia*, vol. 49, n.º 11, pp. 1016–1021, nov. de 1993, ISSN: 1420-9071. DOI: 10.1007/BF02125651. URL: https://doi.org/10.1007/BF02125651.

[5] A. B Jensen, K. Ulyate, J. Boomsma e B. Pedersen, "Varying degrees of Apis mellifera ligustica introgression in protected populations of the black honeybee, Apis mellifera mellifera, in northwest Europe", *Molecular ecology*, vol. 14, pp. 93–106, fev. de 2005. DOI: 10.1111/j.1365-294X.2004.02399.x.

[6] M. Alburaki, S. Moulin, H. Legout, A. Alburaki e L. Garnery, "Mitochondrial structure of Eastern honeybee populations from Syria, Lebanon and Iraq", *Apidologie*,

vol. 42, n.º 5, p. 628, jul. de 2011, ISSN: 1297-9678. DOI: 10.1007/s13592-011-0062-4. URL: https://doi.org/10.1007/s13592-011-0062-4.

[7] Tofilski, Adam, "Using geometric morphometrics and standard morphometry to discriminate three honeybee subspecies", *Apidologie*, vol. 39, n.º 5, pp. 558–563, 2008. DOI: 10.1051/apido:2008037. URL: https://doi.org/10.1051/apido:2008037.

[8] A. Tofilski, *IdentiFly software*, http://drawwing.org/identifly, Accessed: 2019-05-28.

[9] O. Ronneberger, P. Fischer e T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation", em *MICCAI*, 2015.

[10] M. Bouga, C. Alaux, M. Bienkowska, R. Büchler, N. L. Carreck, E. Cauia, R. Chlebo, B. Dahle, R. Dall'Olio, P. D. la Rúa, A. Gregorc, E. Ivanova, A. Kence, M. Kence, N. Kezic, H. Kiprijanovska, P. Kozmus, P. Kryger, Y. L. Conte, M. Lodesani, A. M. Murilhas, A. Siceanu, G. Soland, A. Uzunov e J. Wilde, "A review of methods for discrimination of honey bee populations as applied to European beekeeping", *Journal of Apicultural Research*, vol. 50, n.º 1, pp. 51–84, 2011. DOI: 10.3896/IBRA.1.50.1.06. eprint: https://doi.org/10.3896/IBRA.1.50.1.06. URL: https://doi.org/10.3896/IBRA.1.50.1.06.

[11] O. M. Parkhi, A. Vedaldi e A. Zisserman, "Deep Face Recognition", em *British Machine Vision Conference*, 2015.

[12] K. Sun, B. Xiao, D. Liu e J. Wang, *Deep High-Resolution Representation Learning for Human Pose Estimation*, 2019. eprint: arXiv:1902.09212.

[13] Y. Freund e R. E. Schapire, "Large Margin Classification Using the Perceptron Algorithm", *Machine Learning*, vol. 37, n.º 3, pp. 277–296, dez. de 1999, ISSN: 1573-0565. DOI: 10.1023/A:1007662407062. URL: https://doi.org/10.1023/A:1007662407062.

[14] *Towards Data Science*, https://towardsdatascience.com/, Accessed: 2019-05-20.

[15] R. HECHT-NIELSEN, "III.3 - Theory of the Backpropagation Neural Network**Based on "nonindent" by Robert Hecht-Nielsen, which appeared in Proceedings of the International Joint Conference on Neural Networks 1, 593–611, June 1989. © 1989 IEEE.", em *Neural Networks for Perception*, H. Wechsler, ed., Academic Press, 1992, pp. 65–93, ISBN: 978-0-12-741252-8. DOI: https://doi.org/10.1016/B978-0-12-741252-8.50010-8. URL: http://www.sciencedirect.com/science/article/pii/B9780127412528500108.

[16] A. C. Marreiros, J. Daunizeau, S. J. Kiebel e K. J. Friston, "Population dynamics: Variance and the sigmoid activation function", *NeuroImage*, vol. 42, n.º 1, pp. 147–157, 2008, ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2008.04.239. URL: http://www.sciencedirect.com/science/article/pii/S1053811908005132.

[17] Y. Li e Y. Yuan, "Convergence Analysis of Two-layer Neural Networks with ReLU Activation", em *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan e R. Garnett, eds., Curran Associates, Inc., 2017, pp. 597–607. URL: http://papers.nips.cc/paper/6662-convergence-analysis-of-two-layer-neural-networks-with-relu-activation.pdf.

[18] P. Viola e M. J. Jones, "Robust Real-Time Face Detection", *Int. J. Comput. Vision*, vol. 57, n.º 2, pp. 137–154, mai. de 2004, ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000013087.49260.fb. URL: https://doi.org/10.1023/B:VISI.0000013087.49260.fb.

[19] R. Girshick, J. Donahue, T. Darrell e J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, 2013. eprint: arXiv:1311.2524.

[20] S. Ren, K. He, R. Girshick e J. Sun, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, 2015. eprint: arXiv:1506.01497.

[21] C. Szegedy, S. Ioffe, V. Vanhoucke e A. Alemi, *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*, 2016. eprint: arXiv:1602.07261.

[22] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto e H. Adam, *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, 2017. eprint: arXiv:1704.04861.

[23] J. Redmon e A. Farhadi, *YOLOv3: An Incremental Improvement*, 2018. eprint: arXiv:1804.02767.

[24] C. L. Zitnick e P. Dollár, "Edge Boxes: Locating Object Proposals from Edges", em *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele e T. Tuytelaars, eds., Cham: Springer International Publishing, 2014, pp. 391–405.

[25] N. Georgiev e A. Asenov, "Automatic Segmentation of Lumbar Spine MRI Using Ensemble of 2D Algorithms", em. jan. de 2019, pp. 154–162, ISBN: 978-3-030-13735-9. DOI: 10.1007/978-3-030-13736-6_13.

[26] M. Mudrová e A. Procházka, "PRINCIPAL COMPONENT ANALYSIS IN IMAGE PROCESSING", mar. de 2019.

[27] S. Wold, K. Esbensen e P. Geladi, "Principal component analysis", *Chemometrics and Intelligent Laboratory Systems*, vol. 2, n.º 1, pp. 37–52, 1987, Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists, ISSN: 0169-7439. DOI: https://doi.org/10.1016/0169-7439(87)80084-9. URL: http://www.sciencedirect.com/science/article/pii/0169743987800849.

[28] J. C. Gower, "Generalized procrustes analysis", *Psychometrika*, vol. 40, n.º 1, pp. 33–51, mar. de 1975, ISSN: 1860-0980. DOI: 10.1007/BF02291478. URL: https://doi.org/10.1007/BF02291478.

[29] C. P. KLINGENBERG, "MorphoJ: an integrated software package for geometric morphometrics", *Molecular Ecology Resources*, vol. 11, n.º 2, pp. 353–357, 2011. DOI: 10.1111/j.1755-0998.2010.02924.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1755-0998.2010.02924.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1755-0998.2010.02924.x.

[30] E. Jones, T. Oliphant, P. Peterson et al., *SciPy: Open source scientific tools for Python*, 2001. URL: http://www.scipy.org/.

[31] S. S. Haykin, *Neural networks and learning machines*, Third. Upper Saddle River, NJ: Pearson Education, 2009.

[32] L. Breiman, "Random Forests", *Machine Learning*, vol. 45, n.º 1, pp. 5–32, out. de 2001, ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[33] G. Bradski, "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2000.

[34] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu e X. Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*, 2015. URL: http://download.tensorflow.org/paper/whitepaper2015.pdf.

[35] W. S. McCulloch e W. Pitts, "A logical calculus of the ideas immanent in nervous activity", *The bulletin of mathematical biophysics*, vol. 5, n.º 4, pp. 115–133, dez. de 1943, ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259.

[36] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga e A. Lerer, "Automatic differentiation in PyTorch", 2017.

[37] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama e T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding", *arXiv preprint arXiv:1408.5093*, 2014.

[38] F. Chollet, *Keras*, https://github.com/keras-team/keras, 2015.

[39] K. V. K. A. K. S., "Deep learning with theano, torch, caffe, tensorflow, and deeplearning4J: which one is the best in speed and accuracy?", *Minsk: Publishing Center of BSU*, 2016. URL: http://elib.bsu.by/handle/123456789/158561.

[40] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama e K. Murphy, *Speed/accuracy trade-offs for modern convolutional object detectors*, 2016. eprint: arXiv:1611.10012.

[41] *Stack Overflow*, https://stackoverflow.com/, Accessed: 2019-05-09.

[42] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick e P. Dollár, *Microsoft COCO: Common Objects in Context*, 2014. eprint: arXiv:1405.0312.

[43] S. van der Walt, S. C. Colbert e G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation", *Computing in Science Engineering*, vol. 13, n.º 2, pp. 22–30, mar. de 2011, ISSN: 1521-9615.

[44] F. Pérez e B. E. Granger, "IPython: a System for Interactive Scientific Computing", *Computing in Science and Engineering*, vol. 9, n.º 3, pp. 21–29, mai. de 2007, ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53. URL: https://ipython.org.

[45] J. D. Hunter, "Matplotlib: A 2D graphics environment", *Computing In Science & Engineering*, vol. 9, n.º 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[46] W. McKinney, "Data Structures for Statistical Computing in Python", em *Proceedings of the 9th Python in Science Conference*, S. van der Walt e J. Millman, eds., 2010, pp. 51–56.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot e E. Duchesnay, "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[48] H. Ferreira, C. Machado, C. Costa, T. M. Francoy e M. A. Pinto, "Identification of honey bee populations from the Azores: insights from wing geometric morphometrics", *CIMO - Resumos em Proceedings Não Indexados à WoS/Scopus*, URL: http://hdl.handle.net/10198/18975.

[49] A. R. Lopes, C. J. Neves, H. Ferreira, D. Henriques, A. Quaresma, R. Martín-Hernández, J. Azevedo e M. A. Pinto, "Applying reduce SNP assays for inferring C-lineage introgression patterns in Iberian honeybee populations of the Azores archipelago", *CIMO - Resumos em Proceedings Não Indexados à WoS/Scopus*, URL: http://hdl.handle.net/10198/18979.

[50] wiredfool, A. Clark, Hugo, A. Murray, A. Karpinsky, C. Gohlke, B. Crowell, D. Schmidt, A. Houghton, S. Johnson, S. Mani, J. Ware, D. Caro, S. Kossouho, E. W. Brown, A. Lee, M. Korobov, M. Górny, E. S. Santana, N. Pieuchot, O. Tonnhofer, M. Brown, B. Pierre, J. C. Abela, L. J. Solberg, F. Reyes, A. Buzanov, Y. Yu, eliempje e F. Tolf, *Pillow: 3.1.0*, jan. de 2016. DOI: 10.5281/zenodo.44297. URL: https://doi.org/10.5281/zenodo.44297.

[51] L. Prechelt, "Automatic early stopping using cross validation: quantifying the criteria", *Neural Networks*, vol. 11, pp. 761–767, jun. de 1998. DOI: 10.1016/S0893-6080(98)00010-0. URL: https://doi.org/10.1016/S0893-6080(98)00010-0.

[52] P. Sahoo, *Probability and Mathematical Statistics*. jan. de 2015.

[53] S. J. Pan e Q. Yang, "A Survey on Transfer Learning", *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, n.º 10, pp. 1345–1359, out. de 2010, ISSN: 1041-4347. DOI: 10.1109/TKDE.2009.191.