ALEX ETHEL ALVES MOSCARDINI

UTILIZAÇÃO DE ALGORITMOS DE REDE NEURAL CONVOLUCIONAL REGIONAL DE MÁSCARA (MASK R-CNN) E SEGMENTAÇÃO DE IMAGENS POR COR NA CONTAGEM E CLASSIFICAÇÃO DE FRUTOS DE CAFÉ

LAVRAS - MG
2023

ALEX ETHEL ALVES MOSCARDINI

UTILIZAÇÃO DE ALGORITMOS DE REDE NEURAL CONVOLUCIONAL REGIONAL DE MÁSCARA (MASK R-CNN) E SEGMENTAÇÃO DE IMAGENS POR COR NA CONTAGEM E CLASSIFICAÇÃO DE FRUTOS DE CAFÉ

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Engenharia de Controle e Automação, para obtenção do título de Bacharel.

Prof. Dr. Danton Diego Ferreira

Orientador

Dr. Luiz de Gonzaga Ferreira Júnior

Coorientador

LAVRAS - MG

2023

ALEX ETHEL ALVES MOSCARDINI

UTILIZAÇÃO DE ALGORITMOS DE REDE NEURAL CONVOLUCIONAL REGIONAL DE MÁSCARA (MASK R-CNN) E SEGMENTAÇÃO DE IMAGENS POR COR NA CONTAGEM E CLASSIFICAÇÃO DE FRUTOS DE CAFÉ

USING OF REGION MASK CONVOLUTIONAL NEURAL NETWORK (MASK R-CNN) AND IMAGE SEGMENTATION BY COLOR ALGORITHMS ON COUNTING AND CLASSIFICATION OF COFFEE FRUITS

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Engenharia de Controle e Automação, para obtenção do título de Bacharel.

Prof. Dr. Danton Diego Ferreira
Orientador

Dr. Luiz de Gonzaga Ferreira Júnior
Coorientador

APROVADA em 28 de Novembro de 2023.

Prof. Dr. Danton Diego Ferreira	UFLA
Dr. Luiz de Gonzaga Ferreira Júnior	GONZAGA TREINAMENTOS E CONSULTORIA AGRÍCOLA LTDA

LAVRAS - MG

2023

RESUMO

O objetivo desse trabalho é utilizar uma arquitetura Mask R-CNN para a contagem e classificação do grau de maturação de frutos de café. A contagem e classificação do estágio de maturação do café é um artifício interessante para produtores de café, uma vez que com tais informações, o produtor pode tomar decisões mais assertivas, visando maximizar a colheita dos frutos em graus de maturação que sejam economicamente mais rentáveis. Para atingir tal objetivo, foi reunido uma base de dados contendo imagens dos frutos de café em diferentes graus de maturação. Na sequência foi utilizado o software de marcação “VGG Image Annotator (VIA)” para fazer a anotação de cada fruto utilizando polígonos, tais anotações foram exportadas em um arquivo de formato *json* para posterior processamento. Com isso, foi feito a adaptação de um algoritmo Mask R-CNN em linguagem Python utilizando do recurso de sobrescrita de métodos e classes para trabalhar com a base de dados criada. Utilizando técnicas de transferência de aprendizado, os pesos de uma base de dados pré-estabelecida foram utilizados para treinar uma nova rede neural para classificar as imagens. Com o auxílio de algoritmos de segmentação baseados em intervalos de cores, as imagens foram segmentadas de acordo com os graus de maturação do café, que são: verde, cana, cereja e seco. Posteriormente, uma análise utilizando o indicador MAP (*Mean Average Precision*) foi feita para averiguar os resultados das detecções. O resultado obtido para este indicador foi de 93%, enquanto que para imagens individuais foram obtidos valores de 85% a 100%, indicando, portanto, elevada eficiência e eficácia na detecção. A utilização desses algoritmos mostrou grande potencial na resolução do desafio de detecção do grau de maturação do café, abrindo novas possibilidades no desenvolvimento de tecnologias que atuem nos processos de pré-colheita, colheita e pós-colheita do café. O aprimoramento dos resultados foi verificado na prática, por meio do aumento da base de dados.

Palavras Chave: Maturação do café, contagem do café, classificação do café, colheita do café, redes neurais.

ABSTRACT

The objective of this work is to use a Mask R-CNN architecture to count and classify the degree of ripeness of coffee fruits. Counting and classifying the coffee maturation stage is an interesting device for coffee producers, since with such information, the producer can make more assertive decisions, aiming to maximize the harvest of fruits at maturation levels that are more economically profitable. To achieve this objective, a database containing images of coffee fruits at different degrees of maturation was gathered. The marking software “VGG Image Annotator (VIA)” was then used to annotate each fruit using polygons. These annotations were exported into a json format file for further processing. With this, a Mask R-CNN algorithm was adapted in Python language using the method and class overwriting feature to work with the created database. Using transfer learning techniques, the weights from a pre-established database were used to train a new neural network to classify the images. With the help of segmentation algorithms based on color ranges, the images were segmented according to the coffee's maturity levels, which are: green, cane, cherry and dry. Subsequently, an analysis using the MAP (Mean Average Precision) indicator was carried out to verify the detection results. The result obtained for this indicator was 93%, while for individual images values of 85% to 100% were obtained, therefore indicating high efficiency and effectiveness in detection. The use of these algorithms showed great potential in solving the challenge of detecting the degree of coffee maturation, opening up new possibilities in the development of technologies that act in the coffee pre-harvest, harvest and post-harvest processes. The improvement of results was verified in practice, through the increase in the database.

Key Words: Coffee maturation, coffee counting, coffee classification, coffee harvesting, neural networks.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E TERMOS

MAP – Mean Average Precision (Média das Médias de Precisão)

AP – Average Precision (Média de Precisão)

HSV – Hue Saturation Value

RGB – Red Green Blue

BGR – Blue Green Red

JSON – Java Script Object Anotation (Anotação de Objeto JavaScript)

Mask R-CNN – Mask Region-based Convolutional Neural Network (Rede Neural Convolucional Baseada em Regiões com Máscaras)

Intersection Over Union – Interseção Sobre União

Ground Truth – Anotação

Bound Box – Caixa Delimitadora

Precision – Precisão

Score – Resultado

Config – Configuração

Network – Rede

Path – Diretório, Caminho

Dataset, Database – Conjunto de dados

Image Anotator – Programa anotador de Imagens

Hardware – Peças e componentes eletrônicos que compõe um computador

Software – Programa de computador

Network Configuration – Configuração de Rede

Background – Fundo

JSON – JavaScript Object Notation (Notação de Objeto JavaScript)

C&V – Cana & Verde

CV – Cereja Vermelho

S – Seco

Blur – Borrado

Input – Entrada

Threshold - Limiar

Sumário

1 Introdução

O Brasil é o segundo maior consumidor de café do mundo, e o maior produtor do grão. Somente em 2022, o país exportou 2,2 milhões de toneladas, comercializando o produto com mais de 145 países (MINISTÉRIO DA AGRICULTURA, 2022). Dessa forma, para atender tamanha demanda interna e externa, os produtores necessitam maior produtividade, recorrendo frequentemente à mecanização. Na cafeicultura, principalmente para o processo de colheita, a busca pela redução de custos, melhor desempenho da colheita, maior seletividade de frutos e preservação da lavoura, é muito importante, sendo de interesse tanto dos cafeicultores quanto de empresas fabricantes de máquinas e equipamentos (FERREIRA JÚNIOR et al, 2020).

Uma das dificuldades enfrentadas pelos cafeicultores que mecanizam a colheita é identificar o momento ideal para iniciar a colheita e também colher o máximo de frutos desejados possível. Uma das formas de gerenciar o momento da colheita, é monitorando a maturação dos frutos.

Na lavoura de café é essencial analisar o estágio de maturação do fruto, para assim determinar qual a melhor época de fazer a colheita. Os estágios de maturação são: Verde, cana, cereja amarelo, cereja vermelho, passa e seco. O café seco também é denominado “boia” em determinadas regiões do país.

O fruto “verde” adquire tonalidade verde escuro, enquanto que o fruto “cana” adquire tonalidade verde claro. No fruto “cereja” a tonalidade vai do amarelo ao vermelho escarlate. O fruto “passa” tem tonalidade púrpura e formato geométrico diferenciado, uma vez que ocorre perda de parte de sua umidade e, portanto, de seu volume. No fruto “seco” o volume se esvai quase por completo do fruto, sobrando assim sua semente.

A comunidade de visão computacional tem aprimorado rapidamente os resultados em detecção de objetos e segmentação semântica em um curto espaço de tempo (GIRSHICK, 2013). Novos algoritmos surgem a todo momento aprimorando os resultados e avançando na fronteira do conhecimento.

A colheita seletiva tem ganhado força nos últimos tempos com novas tecnologias que vêm surgindo no mercado. Um exemplo é o novo protótipo denominado Flying Autonomous Robot, ou simplesmente FAR, desenvolvido

pela Israelita Tevel Aerobotics Technologies, que faz o uso de drones equipados com câmeras e um manipulador robótico para fazer a coleta apenas de frutos maduros de pêssegos, nectarinas, ameixas e damascos, preservando os demais frutos até atingirem o ponto ideal para coleta (vide Figura 1).

Os drones são interligados a um veículo terrestre, e o uso de algoritmos de visão computacional fazem o reconhecimento do fruto no estágio ideal de maturação. Assim, uma nova gama de aplicações potenciais pode ser explorada com o estudo das redes neurais e algoritmos de visão computacional. Porém, ressalta-se que esta tecnologia não foi adaptada ao cafeeiro.

Figura 1 – Flying Autonomous Robot.

Fonte: Tevel-Tech (2023).

Nesse contexto, utilizar um algoritmo de alta performance como o Mask R-CNN para identificar o momento ideal para a colheita pode ser uma alternativa ágil, confiável e financeiramente justificável para que o produtor consiga gerenciar melhor a sua colheita. Esse objetivo pode ser alcançado na forma de um aplicativo móvel rodando o algoritmo ou, ainda, com o processamento em tempo real do algoritmo junto a uma câmera acoplada em um maquinário.

2 Referencial Teórico

2.1 Identificação de Frutos de Café

A escala fenológica do café (MORAES, 2008), mostra quatro fases:

- M_1 – frutos de coloração verde, ou seja, sem evidências de alteração na cor.
- M_2 – frutos de coloração verde-cana, ou seja, frutos que já iniciaram a redução na concentração de pigmentos.
- M_3 – frutos em estágio “cereja”, de coloração vermelho-claro e maduros fisiologicamente.
- M_4 – frutos secos, desidratados com coloração externa escura

Para as variedades de cafés com frutos maduros de coloração amarelo, a escala M_1 é a mesma, a M_2 os frutos podem ser de coloração verde-cana ou melão claro; no M_3 os frutos ficam com coloração amarelo claro (vivo) e M_4 o mesmo.

Na colheita do café é importante observar que o grau de maturação dos frutos não é homogêneo, variando até mesmo dentro de uma mesma planta ou ramo. Determinados graus de maturação conferem ao fruto um maior valor comercial no momento da venda. O fruto do tipo “cereja” é considerado o de maior valor agregado, pois confere ao café um sabor característico (FIGUEIREDO, 2021), além de proporcionar maior rendimento final (conhecido por renda do café), ou seja, menor quantidade de frutos para formar uma saca de café beneficiado.

Dessa forma, podemos destacar em duas as modalidades de colheita do café: a plena e a seletiva. Na colheita seletiva os frutos de interesse são coletados diretamente do pé, porém esta é onerosa, de elevado custo e requer mão de obra qualificada e treinada. Com o uso da mecanização, é possível realizar a colheita seletiva utilizando regulagens específicas da colhedora obtendo resultados similares à uma seletiva manual. Já a colheita plena é a colheita objetivando derriçar todos os frutos da planta. Pode ser manual, semi-mecanizada e mecanizada. A variação na maturação dos frutos tem sido um desafio para a prática desse tipo de colheita pois os frutos em diferentes estágios de maturação dificultam a derriça total. A desvantagem da colheita plena, no entanto, é a colheita de grãos de menor valor agregado, devido ao seu grau de maturação.

Uma estratégia para realizar tanto a colheita plena quanto a seletiva, principalmente mecanizada, é determinar, via amostragem e estatística, qual o melhor momento para realizar a colheita, ou seja, em qual momento a safra tem um maior percentual de frutos do tipo cereja.

> Decidir o momento ideal para colher é muito importante e tem relação direta com a qualidade e os defeitos do café. Uma colheita atrasada acarreta maior quantidade de grãos pretos. Caso a colheita seja adiantada, poderá ocorrer maior quantidade de frutos ardidos e verdes, prejudicando também a qualidade final do café. (FERREIRA JÚNIOR et al, 2017).

Uma forma interessante de determinar percentualmente o grau de maturidade dos frutos é emergir uma tábua de 100 furos em uma amostra de café coletada, conforme ilustrado na Figura 2. Assim, por meio da identificação e contagem visual das amostras pode-se obter uma estimativa do percentual de grãos em cada estágio de maturação (vide Figura 3).

Figura 2 – Tábua de separação de frutos.

Fonte: Gonzaga Treinamentos e Consultoria Agrícola Ltda. (2023).

Figura 3 – Estágios de maturação do café.

Fonte: PerfectDailyGrind (2023).

Na sequência vem o desafio da separação e triagem dos frutos. Após a colheita é recomendado que o café seja enviado imediatamente à estrutura de pós colheita, lá ele pode ser submetido ao Abanador, cujo o objetivo é separar as impurezas do café. Seguindo, ele é levado para a Bica de Jogo, onde são separados gravetos e folhas. Então, os frutos podem ser levados ao Lavador, onde recebem uma higienização primária, ocorrendo também a separação de frutos de acordo com sua densidade, acarretando em uma primeira divisão entre frutos de acordo com sua umidade e qualidade (FIGUEIREDO, 2021).

Os frutos cerejas e verdes que geralmente são mais densos podem conter de 55% a 65% de umidade, afundando no lavador. Os frutos do tipo “passa” podem chegar de 40% a 45% de umidade, enquanto os frutos “secos” podem chegar de 15% a 25% de umidade, boiando no lavador (FIGUEIREDO, 2021).

Após o Lavador, os frutos de maior densidade podem ser levados para uma selecionadora eletrônica de cafés conforme ilustração a seguir.

Figura 4 – Separador eletrônico de café cereja.

Fonte: Palini-Alves (2023).

Nesse tipo de máquina, o café passa por uma esteira, onde o grão tem sua cor detectada. Da esteira o grão cai por gravidade em um recipiente de separação, onde existem válvulas injetoras de ar que são acionadas em frações de segundo, soprando ar comprimido de modo a desviarem os grãos verdes para outro bocal de coleta, separando assim os grãos verdes e cana dos grãos cereja vermelho e cereja amarelo.

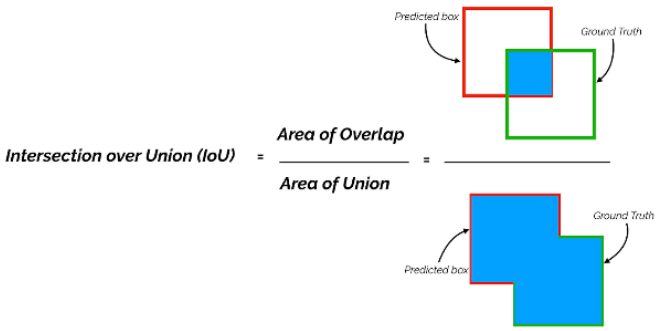
O objetivo deste trabalho é utilizar o algoritmo de Mask R-CNN juntamente com outros algoritmos de segmentação de imagens por cores para realizar a contagem e classificação dos frutos de café em imagens, para que futuramente esses algoritmos possam ser aplicados em soluções tanto para auxiliar na colheita seletiva como na colheita plena do café ou também em sistemas de pós colheita para separação e homogeneização de lotes.

2.2 Métrica MAP

A métrica mais adequada para avaliar o modelo é o Mean Average Precion (MAP), que é uma média aritmética do Average Precision. Para a problemática de detecção de objetos, temos os seguintes indicadores:

- IoU (Intersection Over Union) – é o cociente da área de sobreposição pela área de união de um objeto real (Ground Truth) e seu Bounding Box ou objeto predito.

Figura 5 – IoU (Intersection Over Union).



Fonte: TowardsDataScience (2023).

- Verdadeiro Positivo (True Positive – TP) – quando o objeto está presente no ground truth e o modelo o detecta. É considerado TP quando o IoU é superior a um limiar especificado.
- Falso Positivo (False Positive – FP) – quando o objeto que se encontra no ground truth possui IoU inferior a um limiar e não é encontrado.
- Falso Negativo (False Negative – FN) – quando o objeto se encontra no ground truth e o modelo não o detecta. Portanto, o modelo não gera predição alguma para essa marcação.
- Precisão (Precision) – precisão das predições ou porcentagem de predições corretas. Mede o quão preciso são as predições.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- Recall – capacidade de um modelo de encontrar todos positivos. Mede o quão bem o algoritmo acha todos positivos.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- Precisão Média (AP - Average Precision) – área abaixo da curva Precision X Recall, chamada de curva AP.
- Média das Precisões Médias (MAP - Mean Average Precision) – média das APs das classes. Quanto mais próxima de 1, melhor o modelo.

2.3 Mask R-CNN

O Mask R-CNN é um algoritmo de segmentação de instâncias que oferece segmentação a nível de pixel. O algoritmo pode lidar com inúmeras classes e sobreposições de objetos e é a junção de dois algoritmos que resolvem dois subproblemas: A detecção de objetos e a segmentação semântica (GIRSHICK, 2013).

Os principais componentes do algoritmo são:

- Backbone (Espinha Dorsal) – Utiliza uma arquitetura de rede neural convolucional pré-treinada, como ResNet ou VGG, para extrair características da imagem.
- Região de Proposta (Region Proposal Network - RPN) – Gera propostas de região para possíveis objetos na imagem.
- Cabeça de Detecção – Processa as propostas de região e classifica as caixas delimitadoras associadas a elas.
- Cabeça de Segmentação – Gera máscaras precisas para cada objeto detectado.

O Mask R-CNN tem como saída uma imagem com caixas delimitadoras (bounding boxes), máscaras (masks) e a precisão/pontuação (score) da detecção. As caixas delimitadoras são retângulos que contém o objeto detectado em toda sua extensão de pixels. As máscaras são o cercamento exato de todos os pixels que compõe o objeto, incluindo seus contornos. A precisão é uma medida estatística que indica a probabilidade do objeto de pertencer a determinada classe. Quando consideramos diversas detecções, temos diferentes valores de probabilidade para cada instância encontrada, a junção desses valores forma um conjunto ou intervalo, que pode ser chamado de intervalo de confiança da detecção. O resultado de uma detecção típica de um algoritmo Mask R-CNN é mostrado na Figura 6.

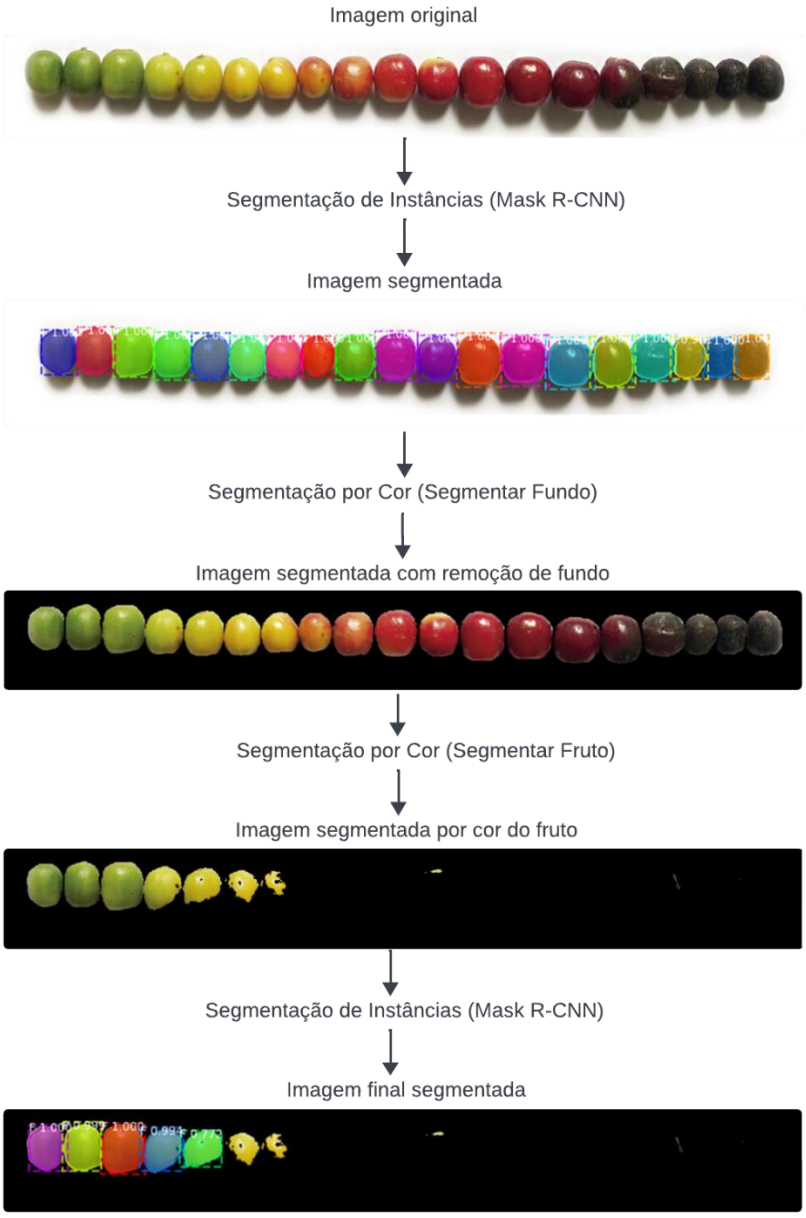
Figura 6 – Saída do Mask R-CNN

Fonte: Alsombra (2023).

2.4 Algoritmo proposto

O processo completo de segmentação do algoritmo proposto no trabalho passa primeiramente por uma segmentação de instâncias que vai segmentar os frutos da imagem. Na sequência é feito uma segmentação de fundo para remover o fundo da imagem, separando gravetos, folhas, cafeeiros e quaisquer outros ruídos da imagem, deixando apenas os frutos. Uma segmentação por cor é feita para segmentar os frutos de determinada coloração e, por fim, é feito uma última segmentação de instâncias para obter os metadados dos frutos segmentados por cor. Todo esse processo é mostrado na Figura 7.

Figura 7 – Algoritmo proposto



Fonte: Do autor (2023).

3 Materiais

3.1 Especificações de Hardware

Grande parte do trabalho foi desenvolvido utilizando o Google Colab. Na plataforma do Google Colab é possível criar notebooks, que misturam linguagem textual com trechos de código de programação em linguagem

Python executados na nuvem por máquinas dedicadas. É possível, ainda, executar comandos básicos de terminal Linux da máquina hospedada.

Todavia, com o aumento da complexidade de computação, do tamanho dos códigos trabalhados e principalmente do tempo de execução dos algoritmos, foi necessário mudar o ambiente de trabalho, uma vez que utilizando a licença gratuita ocorria a desconexão do ambiente de execução depois de algum tempo, acarretando perda de processamento e atraso nos trabalhos.

Assim, software foi instalado localmente em uma máquina para realizar as tarefas de pesquisa e processamento. A máquina utilizada no trabalho tem processador 12th Gen Intel® Core™ i7-12700KF, 32GB de memória DDR5 e processador gráfico NVIDIA GeForce RTX 3060.

É importante ressaltar que a maior parte das máquinas atuais são capazes de realizar as tarefas de processamento, variando apenas a velocidade de execução dos algoritmos e principalmente o treinamento das redes neurais. A NVIDIA GeForce oferece atualmente o melhor suporte para o processamento através de seu plugin “Cudnn” e de sua biblioteca “Cuda”, tendo se destacado no mercado frente as concorrentes.

4 Métodos

4.1 Conjunto de dados

O conjunto de dados utilizado nesse trabalho foi obtido selecionando imagens disponíveis no Google Imagens ou disponibilizadas pela empresa Gonzaga Treinamentos e Consultoria Ltda. São imagens de diversas resoluções mostrando frutos em diversos estágios de maturação, alguns coletados, outros ainda no cafeeiro.

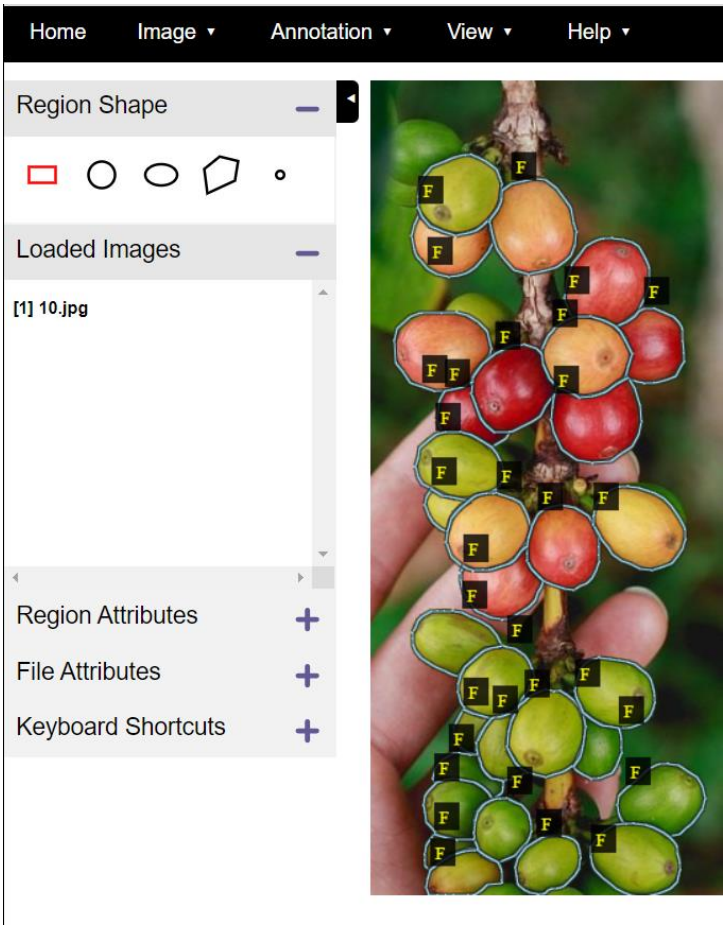
Figura 8 – Imagem aleatória do banco de dados.

Fonte: Arena BR-ME (2023).

Em posse das imagens, iniciou-se o processo de anotação utilizando o software VGG Image Annotator (VIA). O processo consiste em utilizar a ferramenta de polígono e marcar, manualmente, os pontos que fecham o polígono que melhor representa o objeto alvo do treinamento.

Na sequência, esse polígono recebe os chamados atributos de região. Aqui foi utilizado somente um atributo denominado de “label”, sendo todas instâncias dos frutos marcados tendo este atributo como valor “F” de fruto. É possível visualizar como a imagem fica após o processo de anotação dentro do software.

Figura 9 – Anotação dos frutos.



Fonte: Do autor (2023).

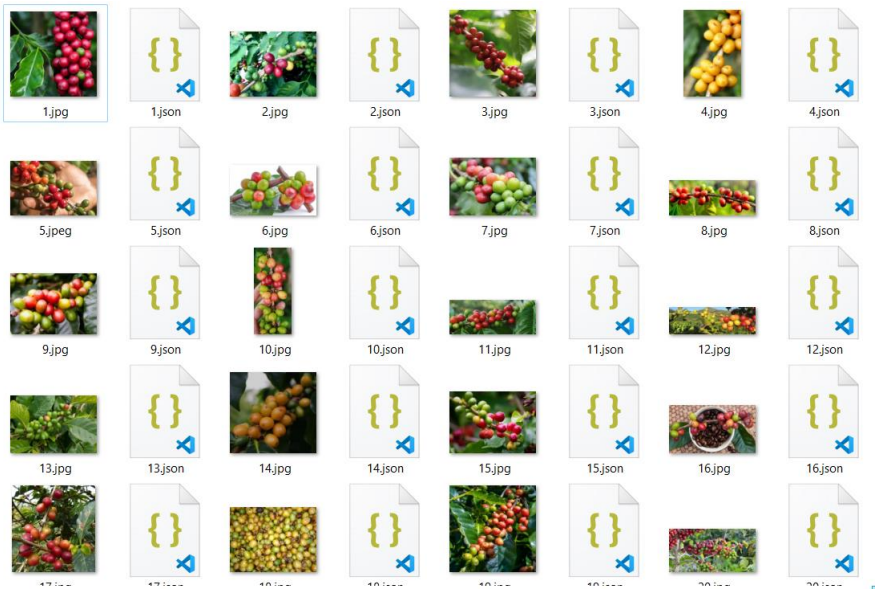
Na sequência, um arquivo de extensão .json é salvo. O formato “json” é muito utilizado para representar conjuntos de dados utilizando a estrutura de dados conhecida como árvore, utilizando o formato chave-valor. Abrindo o arquivo gerado, é possível observar a estrutura da Figura 10.

Figura 10 – Estrutura do arquivo json de anotação.

Fonte: Do autor (2023).

Nessa instância em específico, algumas informações podem ser observadas na estrutura de árvore, tais como o nome do arquivo e seu formato “10.jpg47877”, seu nome “filename”:”10.jpeg”, uma chave “regions” com valores “0” e “1”, indicando dois frutos ou polígonos demarcados. Em outra chave “shape_atributes” o nome do shape em “name”:”polygon”, assim como suas coordenadas x e y, nos campos “all_points_x” e “all_points_y” respectivamente. No campo “region_attributes” outro campo “label” de valor único “F” como discutido anteriormente. Na imagem em questão tem-se, portanto, 2 frutos demarcados, o fruto 0 está com seu “shape_atributes” comprimido para melhor visualização do restante das estruturas. Feito a marcação de todas fotos, 20 delas foram alocados em um diretório “repositorio/train” para fazer o treinamento da rede, enquanto que 6 delas foram armazenadas no diretório “repositorion/val” de validação do treinamento vide Figura 11.

Figura 11 – Repositório de imagens.



Fonte: Do autor (2023).

Foi criado um arquivo denominado “via_region_data.json” reunindo todas marcações individuais feitas dos demais arquivos. Esse processo foi feito para facilitar a marcação, visto a quantidade de imagens envolvidas. Esse será o arquivo que irá ser carregado no código para o treinamento contendo toda base de dados de treinamento vide Figura 12.

Figura 12 – Estrutura geral do arquivo de anotação.

Fonte: Do autor (2023).

4.2 Importação das Bibliotecas

O código começa com a importação das bibliotecas. As bibliotecas utilizadas foram as seguintes:

Figura 13 – Bibliotecas.

```
# Bibliotecas

import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
import json
import random
import time
import skimage.draw
import tensorflow as tf
import mrcnn.utils as utils
import mrcnn.model as modellib
from skimage import io
from skimage.io import imread
from mrcnn.config import Config
from mrcnn import visualize
```

Fonte: Do autor (2023).

- os – Fornece funções para interagir com o sistema operacional, permitindo manipulação de diretórios, arquivos e outras operações relacionadas ao sistema operacional.
- numpy – Biblioteca de computação científica. Fornece arrays multidimensionais e funções matemáticas.
- cv2 – OpenCV. Biblioteca de visão computacional para trabalhos com processamento de imagem, detecção de objetos, reconhecimento de padrões e outras mais.
- matplotlib.pyplot – Biblioteca de visualização de dados que permite trabalhar com gráficos e visualizações interativas.
- json – Biblioteca que permite a leitura e escrita em arquivos json (JavaScript Object Notation).
- skimage.draw – Fornece funções para desenhar formas geométricas em imagens.
- skimage.io – Fornece funções para manipular imagens.
- mrcnn.config – modulo de configuração da biblioteca Mask R-CNN.

- mrcnn.utils – módulo que contém utilitários da biblioteca Mask R-CNN, auxiliando no redimensionamento de imagens, cálculo de sobreposição de bounding boxes e outros.
- mrcnn.visualize – módulo que fornece funções para visualização de resultados de detecção e segmentação de objetos usando Mask R-CNN.
- mrcnn.model – módulo que contém a implementação principal do Mask R-CNN.

4.3 Adaptando o Mask R-CNN

Para adaptar o Mask R-CNN ao conjunto de dados, foi criado uma classe denominada “Detector” quer herda da classe utils.Dataset e sobrescreve os métodos load_object, load_mask e image_reference, vide Figura 14.

Figura 14 – Métodos da classe Detector.

```
19  # Caregamento do dataset
20
21  class Detector(utils.Dataset):
22
23      # dataset_dir: Diretório raiz do dataset
24      # subset: Subconjunto de treinamento(train) ou validação (val)
25
26      def load_object(self, dataset_dir, subset, nome_annotation="via_region_data.json"):
65
66      def load_mask(self, image_id):
87
88      def image_reference(self, image_id):
```

Fonte: Do autor (2023).

O método “load_object” tem como objetivo configurar o número de classes da rede neural, os diretórios de treinamento e validação, carregar as anotações, interpretar as coordenadas x e y dos pontos dos polígonos, e calcular o tamanho da imagem para a geração das máscaras.

Figura 15 – Método “load_object”.

```
# Carregamento do dataset

class Detector(utils.Dataset):

    # dataset_dir: Diretório raiz do dataset
    # subset: Subconjunto de treinamento(train) ou validação (val)

    def load_object(self, dataset_dir, subset, nome_annotation="via_region_data.json"):

        # Adiciona as classes da rede neural
        self.add_class("objetos", 1, "F")

        # Escolhe qual o dataset
        assert subset in ["train", "val"]
        dataset_dir = os.path.join(dataset_dir, subset)

        # Carrega as anotações
        annotations = json.load(open(os.path.join(dataset_dir, nome_annotation)))
        annotations = list(annotations.values())
        annotations = [a for a in annotations if a['regions']]

        contagem = 0

        # Interpreta as coordenadas x e y dos pontos dos polígonos. O "if"
        # tem como função dar compatibilidade a anotações geradas por diferentes
        # versões do VIA.
        for a in annotations:
            if type(a['regions']) is dict:
                polygons = [r['shape_attributes'] for r in a['regions'].values()]
            else:
                polygons = [r['shape_attributes'] for r in a['regions']]

            # Fornece à função load_mask() o tamanho da imagem para que possa haver
            # a conversão dos polígonos em máscaras.
            image_path = os.path.join(dataset_dir, a['filename'])
            image = skimage.io.imread(image_path)
            height, width = image.shape[:2]
            contagem = contagem+1

            self.add_image("objetos",
                image_id=a['filename'], # Nome do arquivo como id da imagem.
                path=image_path,
                width=width, height=height,
                polygons=polygons)

        print("Número de imagens " + subset + ": " + str(contagem))
```

Fonte: Do autor (2023).

O método “load_mask” tem como objetivo gerar as máscaras das instâncias da imagem, converter os polígonos em máscaras bitmap e classificar cada pixel da imagem se ele representa um pixel da classe ou não.

Figura 16 – Método “load_mask”.

```
66      def load_mask(self, image_id):
67
68          # Gera as máscaras das instâncias da imagem.
69          image_info = self.image_info[image_id]
70          if image_info["source"] != "objetos":
71              return super(self.__class__, self).load_mask(image_id)
72
73          # Faz a conversão dos polígonos em uma máscara bitmap de shape [height, width, instance_count].
74          info = self.image_info[image_id]
75          mask = np.zeros([info["height"], info["width"], len(info["polygons"])],
76                          dtype=np.uint8)
77
78          # Classifica cada pixel da imagem como pertencente à classe ou não.
79          for i, p in enumerate(info["polygons"]):
80
81              # Define os índices dos pixels dentro dos polígonos como valor 1 (cor branca).
82              rr, cc = skimage.draw.polygon(p['all_points_y'], p['all_points_x'], mask.shape)
83              mask[rr, cc, i] = 1
84
85          # Retorna a máscara e o array dos ids das classes de cada instância encontrada.
86          return mask.astype(bool), np.ones([mask.shape[-1]], dtype=np.int32)
```

Fonte: Do autor (2023).

Ainda dentro da classe Detector, o método “image_reference”, que retorna o caminho (path) da imagem vide Figura 17. Na sequência é definido o nome do arquivo das anotações e o path do dataset. Dois objetos da classe Detector são criados, cada um executando o método “load_object”.

Figura 17 – Método “image_reference”.

```
88      def image_reference(self, image_id):
89          # Retorna o caminho da imagem.
90          info = self.image_info[image_id]
91          if info["source"] == "objetos":
92              return info["path"]
93          else:
94              super(self.__class__, self).image_reference(image_id)
95
96  annotation = 'via_region_data.json'
97  path_dataset = "/Arquivos/UFLA/TCC/repositorio/"
98
99  dataset_train = Detector()
100 dataset_train.load_object(path_dataset, 'train', annotation)
101 dataset_train.prepare()
102
103 dataset_val = Detector()
104 dataset_val.load_object(path_dataset, 'val', annotation)
105 dataset_val.prepare()
```

Fonte: Do autor (2023).

4.4 Configurando a sessão e a compatibilidade do tensorflow

O trecho permite que não ocorra problemas ao executar o código com versões mais recentes do tensorflow. Uma sessão denominada “sessao” é criada para o treinamento.

Figura 18 – Configuração de sessão e compatibilidade.

```
# Configurando sessão e compatibilidade do tensorflow

from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
config = ConfigProto()
config.gpu_options.allow_growth = True
sessao = InteractiveSession(config=config)

path_raiz = os.path.abspath('/Arquivos/UFLA/TCC/Mask_RCNN-TF2')
path_modelos = os.path.join(path_raiz, 'logs')
path_modelos
```

Fonte: Do autor (2023).

4.5 Configuração da rede neural e carregamento dos pesos

Para configurar a rede neural é criado uma classe NetworkConfig que herda da classe Config. Nessa classe é feito a configuração de alguns parâmetros importantes como o número de passos por época do treinamento, o número de classes (uma para background e outra para frutos) vide Figura 19.

Figura 19 – Configuração da rede neural.

```
# Configuração da rede neural

class NetworkConfig(Config):
  NAME = 'dados'
  IMAGES_PER_GPU = 2
  DETECTION_MAX_INSTANCES = 600
  NUM_CLASSES = 1 + 1
  STEPS_PER_EPOCH = 100
  DETECTION_MIN_CONFIDENCE = 0.10
  USE_MINI_MASK=False
  IMAGE_MIN_DIM = 2048
  IMAGE_MAX_DIM = 2048
  VALIDATION_STEPS = 5

config = NetworkConfig()
config.display()

path_modelos = os.path.join(path_raiz, 'logs')
path_modelos

# Carregamento dos pesos

path_modelo_COCO = os.path.join(path_raiz, 'mask_rcnn_coco.h5')
path_modelo_COCO

if not os.path.exists(path_modelo_COCO):
  utils.download_trained_weights(path_modelo_COCO)
```

Fonte: Do autor (2023).

Os parâmetros de configuração usados foram:

- NAME – Identificador do modelo.
- IMAGES_PER_GPU – Número de imagens que são processadas por cada unidade de processamento gráfico (GPU) a cada iteração durante o treinamento.
- DETECTION_MAX_INSTANCES – Número máximo de instâncias que o modelo pode detectar.
- NUM_CLASSES – Número de classes que o modelo é capaz de detectar. Nesse caso a classe de fundo (representada pelo 0) e a classe de interesse (frutos – F).
- STEPS_PER_EPOCH – Parâmetro que define quantos passos de treinamento são executados por época. Cada passo é uma atualização de pesos. O número total de passos em uma época é determinado pelo número de amostras de treinamento dividido pelo BATCH_SIZE.

- USE_MINI_MASK – Parâmetro que determina se o modelo deve usar máscaras reduzidas ou máscaras completas durante o treinamento. As máscaras reduzidas tem a capacidade de acelerar o treinamento.
- IMAGE_MIN_DIM – Define a dimensão mínima das imagens de entrada durante o treinamento. As imagens de treinamentos são redimensionadas para essa dimensão mínima antes do treinamento.
- IMAGE_MAX_DIM – Define a dimensão máxima das imagens de entrada durante o treinamento. As imagens de treinamento são redimensionadas para esta dimensão máxima antes do treinamento.
- VALIDATION_STEPS – Parâmetro que define quantos passos de validação são executados após cada época durante o treinamento.

4.6 Treinamento da rede neural personalizada

Para treinar a rede neural é feito o carregamento do modelo configurado com o os pesos “coco” na variável “model”. A partir disso é iniciado o treinamento juntamente com o cronômetro que vai marcar o tempo gasto no treinamento na variável “minutos” vide Figura 20.

Figura 20 – Método “carregar_modelo_treinamento”.

```
147  def carregar_modelo_treinamento(config, inicia_com = 'coco', model_path=''):
148    model = modellib.MaskRCNN(mode = 'training', config = config, model_dir=model_path)
149    if inicia_com == 'coco':
150      model.load_weights(path_modelo_COCO, by_name = True, exclude = ['mrcnn_class_logits',
151                                                                      'mrcnn_bbox_fc',
152                                                                      'mrcnn_bbox',
153                                                                      'mrcnn_mask'])
154    elif inicia_com == 'imagenet':
155      model.load_weights(model.get_imagenet_weights(), by_name=True)
156    elif inicia_com == 'last':
157      if model_path == "":
158        model_path = model.find_last()
159      model.load_weights(model_path, by_name=True)
160
161    return model
162
163  model = carregar_modelo_treinamento(config, 'coco')
164
165  # Treinamento
166
167  inicio = time.time()
168  model.train(dataset_train, dataset_val, learning_rate = config.LEARNING_RATE, epochs = 15, layers='heads')
169  fim_treino = time.time()
170  minutos = round((fim_treino - inicio) / 60, 2)
171  print('Tempo de treinamento: ', minutos)
```

Fonte: Do autor (2023).

Ao fim do treinamento, é gerado um modelo de nome “mask_rcnn_dados_0150”.

4.7 Configurando a classe de inferência

Uma vez tendo o modelo treinado é possível carrega-lo para realizar inferências. Primeiramente é criado uma classe “InferenceConfig” que herda de “Config”, e é sobrescrito os parâmetros de hardware e processamento para esta classe. O método “carrega_modelo_teste” cria uma instância da classe de configuração e utiliza essa instância para carregar o modelo na variável model vide Figura 21.

Figura 21 – Classe de inferência e método “carrega_modelo_teste”.

```
# Testes

class InferenceConfig(NetworkConfig):
  GPU_COUNT = 1
  IMAGES_PER_GPU = 1

inference_config = InferenceConfig()

# Carregamento da rede neural

def carrega_modelo_teste(model_path='modelo_treinado'):
  inference_config = InferenceConfig()
  model = modellib.MaskRCNN(mode = 'inference', config=inference_config, model_dir=model_path)
  if model_path == 'modelo_treinado':
    model_path = model.find_last()
  print('Carregando os pesos de', model_path)
  model.load_weights(model_path, by_name=True)
  return model, inference_config

model_teste, inference_config = carrega_modelo_teste('mask_rcnn_dados_0150.h5')
```

Fonte: Do autor (2023).

4.8 Segmentando a imagem quanto à classe

Na sequência, o método “segmentar_imagem” recebe o modelo e a imagem no qual vai realizar a segmentação. A variável retornada contém diversos dados a respeito da segmentação tais como a região de interesse, as máscaras que foram geradas do resultado da segmentação, o id das classes dos objetos encontrados e o score, que é o número de instâncias encontrado durante a segmentação. É definido também um método “mostrar” com o intuito de visualizar qualquer imagem que a este for passado. O comando “size” tem como objetivo determinar o número de instâncias encontradas.

Figura 22 – Método “segmentar_imagem”.

```
# Função de segmentação

def segmentar_imagem(model_teste, img):
  resultados = model_teste.detect([img], verbose = 0)
  r = resultados[0]
  visualize.display_instances(img, r['rois'], r['masks'], r['class_ids'], dataset_val.class_names,
                              r['scores'],
                              figsize=(14, 12))
  return r


# Função de visualização

def mostrar(imagem):
  fig = plt.gcf()
  fig.set_size_inches(16,4)
  plt.imshow(imagem)
  plt.axis('off')
  plt.show

# Imagem teste

imagem = imread('repositorio/val/5.jpg')
resultado = segmentar_imagem(model_teste, imagem)
print("Número de instâncias encontradas na imagem de teste: ", resultado['scores'].size)
```

Fonte: Do autor (2023).

4.9 Utilizando as máscaras para segmentação de fundo

Feito a segmentação, algumas imagens podem conter conteúdo de fundo irrelevante para a análise, o método “segmentacao_fundo”, ilustrado na Figura 23, remove esse conteúdo, deixando apenas a imagem na região das máscaras encontradas durante a segmentação anterior.

Figura 23 – Segmentação de fundo.

```
def segmentacao_fundo(img, mask):
    gray = skimage.color.gray2rgb(skimage.color.rgb2gray(img)) * 0
    # Faz a cópia da imagem original na região das máscaras
    if mask.shape[-1] > 0:
        mask = (np.sum(mask, -1, keepdims=True) >= 1)
        # Substitui os pixels na região das máscara
        resultado = np.where(mask, img, gray).astype(np.uint8)
    else:
        resultado = gray.astype(np.uint8)
    return resultado

segmentado = segmentacao_fundo(imagem, resultado['masks'])
visualize.display_images([segmentado, imagem], cols=2)
mostrar(segmentado)
```

Fonte: Do autor (2023).

4.10 Segmentação por cores

4.10.1 Cana e Verde

Para segmentação dos frutos Cana e Verde foi usado um algoritmo de segmentação de imagem no espaço de cores HSV. O algoritmo faz a conversão da imagem do espaço de cores BGR para HSV, na sequência define um limite inferior e outro superior que determina a cor verde no espaço de cores HSV e cria uma máscara binária de pixels que estão dentro desse intervalo. Assim, é retornada a imagem com os pixels que fazem a interseção com os valores verdadeiros da máscara binária, o código que representa este processo é apresentado na Figura 24.

Figura 24 – Método “segmentar_c_v”

```
237  # Segmentação C e V
238
239  def segmentar_c_v(imagem):
240      # Converte a imagem para o espaço de cores HSV (Hue, Saturation, Value)
241      hsv = cv2.cvtColor(imagem, cv2.COLOR_BGR2HSV)
242
243      # Define os limites para a cor verde no espaço de cores HSV
244      limite_inferior = np.array([35, 50, 50])
245      limite_superior = np.array([95, 255, 255])
246
247      # Criar uma máscara binária para segmentar a cor verde
248      mascara = cv2.inRange(hsv, limite_inferior, limite_superior)
249      imagem_segmentada = cv2.bitwise_and(imagem, imagem, mask=mascara)
250
251      return imagem_segmentada
```

Fonte: Do autor (2023).

Por último é feito a segmentação novamente na imagem para detectar qual os “scores”, ou seja, o número de frutos Cana e Verde do resultado da segmentação por cores. Utilizando o método size na variável é possível observar o número de instâncias de frutos verde e cana. Foi feito também um trecho de código para mostrar o comparativo dessa segunda etapa de segmentação, que está ilustrado na Figura 25.

Figura 25 – Método “segmentar_c_v” continuação.

```
segmentado_c_v = segmentar_c_v(segmentado)
resultado_c_v = segmentar_imagem(model_teste, segmentado_c_v)
print("Número de instâncias C e V encontradas na imagem de teste: ", resultado_c_v['scores'].size)
segmentacao_sobreposicao = cv2.addWeighted(segmentado_c_v, 0.7, segmentado, 0.3, 0)
titulos = ['Segmentação de fundo', 'Segmentação C e V']
imagens = [segmentado, segmentacao_sobreposicao]

fig = plt.gcf()
fig.set_size_inches(18, 10)
for i in range(2):
    plt.subplot(1,2,i+1)
    plt.imshow(imagens[i])
    plt.title(titulos[i])
    plt.xticks([]),plt.yticks([])
```

Fonte: Do autor (2023).

4.10.2 Cereja Vermelho

Um método semelhante ao da Figura 24 foi desenvolvido para a segmentação dos frutos Cereja Vermelho conforme ilustrado na Figura 26. Neste método é considerado a união de dois intervalos do espectro que representam tonalidades do vermelho do claro ao escuro. A combinação das máscaras ocorre por meio da função “cv2.bitwise_or”.

Figura 26 – Método “segmentar_cv”.

```
# Segmentação CV

def segmentar_cv(imagem):
    # Converter a imagem para o espaço de cores HSV
    hsv2 = cv2.cvtColor(imagem, cv2.COLOR_RGB2HSV)

    limite_inferior = np.array([0, 50, 50])
    limite_superior = np.array([20, 255, 255])  # Tons claros
    limite_inferior2 = np.array([140, 50, 50])  # Tons escuros
    limite_superior2 = np.array([180, 255, 255])

    # Criar uma máscara binária para segmentar a cor vermelha na imagem
    mascara1 = cv2.inRange(hsv2, limite_inferior, limite_superior)
    mascara2 = cv2.inRange(hsv2, limite_inferior2, limite_superior2)

    # Combinar as duas máscaras para obter a segmentação completa dos tons de vermelho
    mascara_final = cv2.bitwise_or(mascara1, mascara2)

    # Aplicar a máscara na imagem original para obter apenas os pixels vermelhos
    imagem_segmentada = cv2.bitwise_and(imagem, imagem, mask=mascara_final)

    return imagem_segmentada

segmentado_cv = segmentar_cv(segmentado)
resultado_cv = segmentar_imagem(model_teste, segmentado_cv)
print("Número de instâncias CV encontradas na imagem de teste: ", resultado_cv['scores'].size)
resultado_cv['scores'].shape
segmentacao_sobreposicao = cv2.addWeighted(segmentado_cv, 0.7, segmentado, 0.3, 0)
titulos = ['Segmentação de fundo', 'Segmentação CV']
imagens = [segmentado, segmentacao_sobreposicao]

fig = plt.gcf()
fig.set_size_inches(18, 10)
for i in range(2):
    plt.subplot(1,2,i+1)
    plt.imshow(imagens[i])
    plt.title(titulos[i])
    plt.xticks([]),plt.yticks([])

plt.subplots_adjust(wspace=0.02)
plt.show()
```

Fonte: Do autor (2023).

4.10.3 Seco

Um método semelhante ao da Figura 26 utilizado para segmentar os frutos Secos é mostrado na Figura 27. Tonalidades de marrom e cinza são agrupadas em uma máscara binária para segmentar os pixels nessas faixas.

Figura 27 – Método “segmentar_s”.

```
# Segmentação S

def segmentar_s(imagem):
    # Converter a imagem para o espaço de cores HSV
    hsv = cv2.cvtColor(imagem, cv2.COLOR_BGR2HSV)

    # Definir os limites para tons acinzentados
    limite_inferior_acinzentado = np.array([0, 0, 5])
    limite_superior_acinzentado = np.array([179, 100, 150])

    # Definir os limites para tons de marrom
    limite_inferior_marrom = np.array([0, 20, 0])
    limite_superior_marrom = np.array([60, 255, 150])

    # Criar máscaras binárias para segmentar tons acinzentados e marrom (faixa ampliada)
    mascara_acinzentado = cv2.inRange(hsv, limite_inferior_acinzentado, limite_superior_acinzentado)
    mascara_marrom = cv2.inRange(hsv, limite_inferior_marrom, limite_superior_marrom)

    # Combine as duas máscaras para obter a segmentação completa de tons acinzentados e marrom
    mascara_final = cv2.bitwise_or(mascara_acinzentado, mascara_marrom)

    # Aplique a máscara na imagem original para obter apenas os pixels de tons acinzentados e marrom
    imagem_segmentada = cv2.bitwise_and(imagem, imagem, mask=mascara_final)

    return imagem_segmentada

segmentado_s = segmentar_s(segmentado)
resultado_s = segmentar_imagem(model_teste, segmentado_s)
print("Número de instâncias S encontradas na imagem de teste: ", resultado_s['scores'].size)
resultado_s['scores'].shape
segmentacao_sobreposicao = cv2.addWeighted(segmentado_s, 0.7, segmentado, 0.3, 0)
titulos = ['Segmentação de fundo', 'Segmentação S']
imagens = [segmentado, segmentacao_sobreposicao]

fig = plt.gcf()
fig.set_size_inches(18, 10)
for i in range(2):
    plt.subplot(1,2,i+1)
    plt.imshow(imagens[i])
    plt.title(titulos[i])
    plt.xticks([]),plt.yticks([])

plt.subplots_adjust(wspace=0.02)
plt.show()
```

Fonte: Do autor (2023).

4.11 Indicadores e métricas

O código da Figura 28 faz a análise dos resultados para o banco de dados de validação. Nele é feito a extração dos indicadores do modelo, dentre eles o AP e é calculado a média dos APs para obtenção do MAP que vai gerar o indicador geral do modelo. Também é mostrado o número de instâncias encontradas em cada imagem.

Figura 28 – Cálculo dos indicadores.

```
# Avaliação de resultados

APs = []

for img_id in dataset_val.image_ids:
  img, img_meta, gt_class_id, gt_bbox, gt_mask = modellib.load_image_gt(dataset_val, inference_config, img_id,False)
  resultados = model_teste.detect([img], verbose=0)
  resultado = resultados[0]
  print("Número de instâncias encontradas na imagem ", img_id, " de validação: ", resultado['scores'].size)
  AP, precisions, recalls, overlaps = utils.compute_ap(gt_bbox, gt_class_id, gt_mask, resultado['rois'],
                                                       resultado['class_ids'], resultado['scores'], resultado['masks'])
  APs.append(AP)
  print("AP da imagem", img_id, " de validação: ", AP)
print('MAP: ', np.mean(APs))
```

Fonte: Do autor (2023).

O método “analise” faz a visualização da anotação (ground truth) e da segmentação para efeitos de comparação.

Figura 29 – Método “analise”.

```
def analise(model_teste, img_id, dataset_val, inference_config):

    img_original, img_meta, gt_class_id, gt_bbox, gt_mask = modellib.load_image_gt(dataset_val, inference_config,
                                                                                   img_id, False)


    # Modelo treinado
    resultados = model_teste.detect([img_original], verbose=0)
    resultado = resultados[0]
    visualize.display_instances(img_original, resultado['rois'], resultado['masks'], resultado['class_ids'],
                                dataset_val.class_names, resultado['scores'], figsize = (10,10))

    # Anotação (Ground Truth)
    visualize.display_instances(img_original, gt_bbox, gt_mask, gt_class_id,
                                dataset_val.class_names, figsize=(10, 10))

img_id = random.choice(dataset_val.image_ids)
analise(model_teste, img_id, dataset_val, inference_config)
```

Fonte: Do autor (2023).

5 Resultados

Os resultados foram obtidos a partir da execução do algoritmo com 3 imagens de teste e 5 imagens de validação. As imagens de teste são a 22 e 23 do banco de treinamento e a imagem 5 do banco de validação. Essas imagens foram escolhidas em razão de sua heterogeneidade, representando melhor as classes dos frutos que o algoritmo abrange. Como o processo é estocástico, os resultados podem variar ligeiramente em cada instância de execução, mas convergem para um dado valor específico.

Tabela 1: Conjunto de dados

Conjunto de dados de treinamento	Conjunto de dados de validação	Conjunto de dados de testes
[1, 29]	[1, 6]	3, 22 e 23.

Fonte: Do autor (2023).

5.1 Segmentação de Imagens de Teste

5.1.1 Imagem 5

A segmentação da imagem 5 é mostrada na Figura 30.

Figura 30 – Segmentação da imagem 5.

Fonte: Do autor (2023).

É possível observar que o algoritmo foi capaz de detectar 100% dos frutos, com um intervalo de confiança variando de 99% a 100%.

A Figura 31 mostra a segmentação de fundo da imagem.

Figura 31 – Segmentação de fundo da imagem 5.

Fonte: Do autor (2023).

A segmentação C&V é mostrada na Figura 32.

Figura 32 – Segmentação C&V da imagem 5.

Fonte: Do autor (2023).

O intervalo de confiança variou de 77% a 100% e o número de instâncias encontradas foi 5. A Figura 33 mostra um comparativo da imagem de segmentação de fundo com a imagem de segmentação C e V.

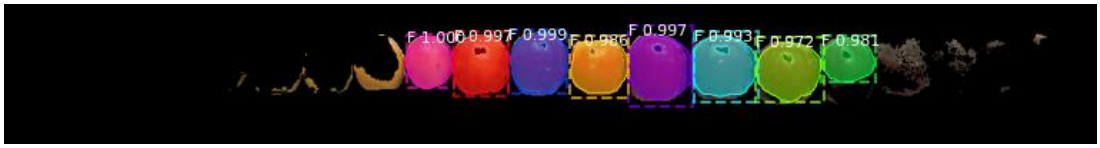
Figura 33 – Comparativo da segmentação C&V da imagem 5.

Fonte: Do autor (2023).

É possível observar que o intervalo de cores definido no algoritmo ocasionou a segmentação de tons de verde claro, beirando ao amarelo, acarretando em um falso positivo do fruto mais à direita, que poderia ser categorizado como cereja amarelo.

A segmentação CV da imagem 5 é mostrada na Figura 34.

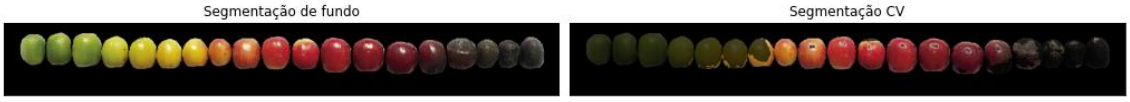
Figura 34 – Segmentação CV da imagem 5.



Fonte: Do autor (2023).

O intervalo de confiança variou de 97% a 100% e o número de instâncias encontradas foi de 8. A Figura 35 mostra um comparativo da imagem de segmentação de fundo com a imagem de segmentação CV.

Figura 35 – Comparativo da segmentação CV da imagem 5.



Fonte: Do autor (2023).

A Segmentação S da imagem 5 é mostrada na Figura 36.

Figura 36 – Segmentação S da imagem 5.

Fonte: Do autor (2023).

O intervalo de confiança variou de 54% a 100% e o número de instâncias encontradas foi de 3.

A Figura 37 mostra um comparativo da imagem de segmentação de fundo com a imagem de segmentação S.

Figura 37 – Comparativo da segmentação S da imagem 5.

Fonte: Do autor (2023).

Um falso negativo foi registrado no fruto mais à esquerda, evidenciando a necessidade de um refinamento ou ajuste nos intervalos ou até mesmo no método “segmentacao_s”.

5.1.2 Imagem 22

A segmentação da imagem 22 é mostrada na Figura 38.

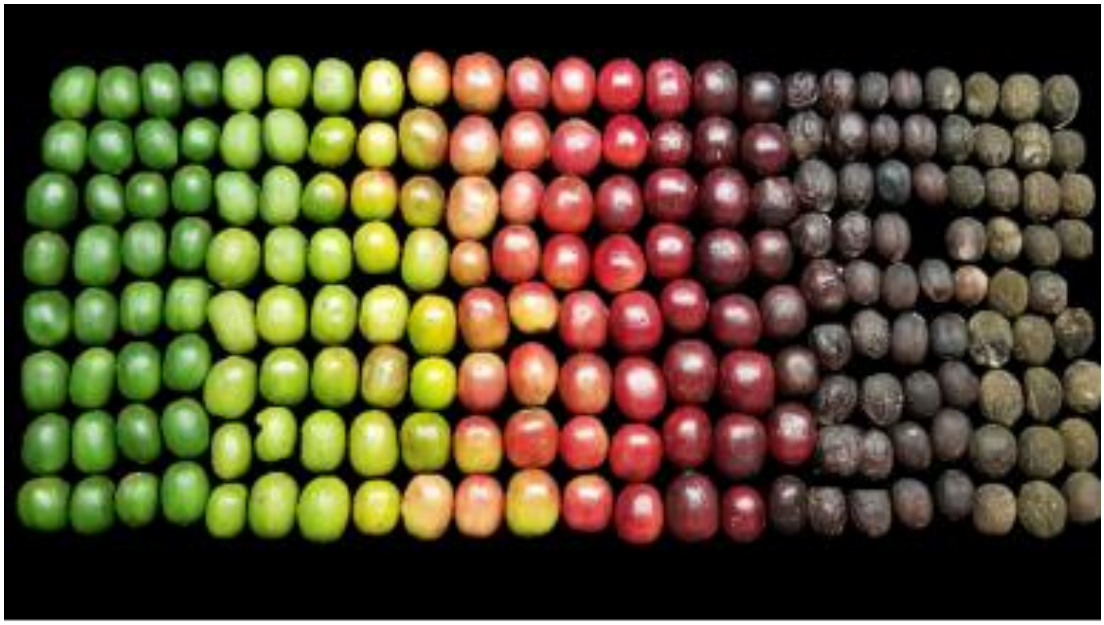
Figura 38 – Segmentação da imagem 22.

Fonte: Do autor (2023).

O algoritmo foi capaz de detectar 99% dos frutos, com um intervalo de confiança variando de 51% a 100%.

A Figura 39 mostra a segmentação de fundo da imagem 22, evidenciando dois frutos não segmentados.

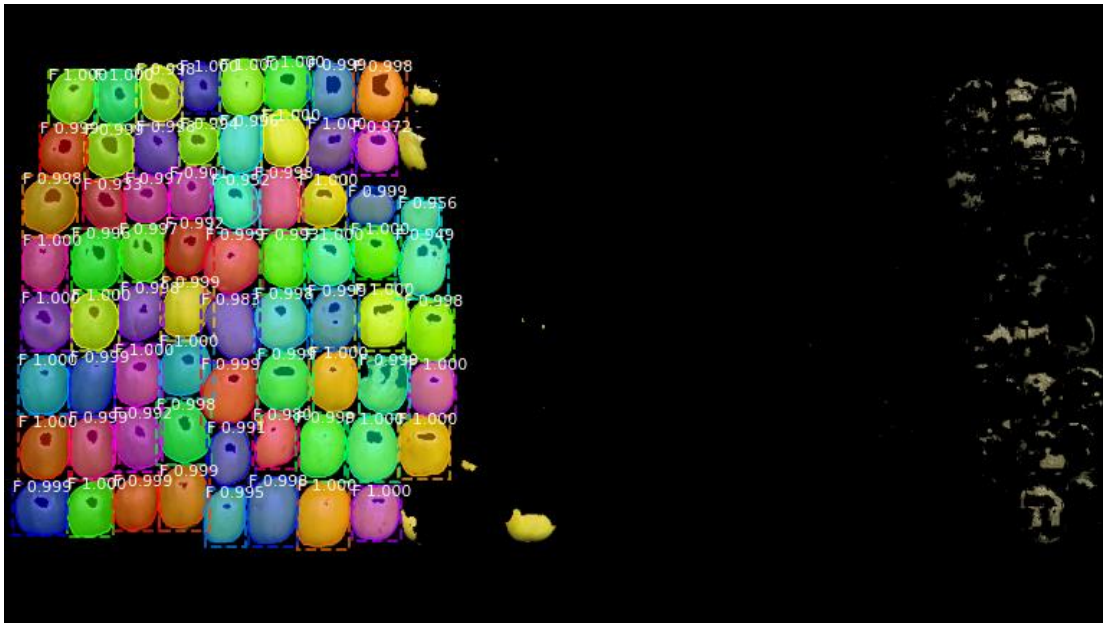
Figura 39 – Segmentação de fundo da imagem 22.



Fonte: Do autor (2023).

A segmentação C&V é mostrada na Figura 40.

Figura 40 – Segmentação C&V da imagem 22.



Fonte: Do autor (2023).

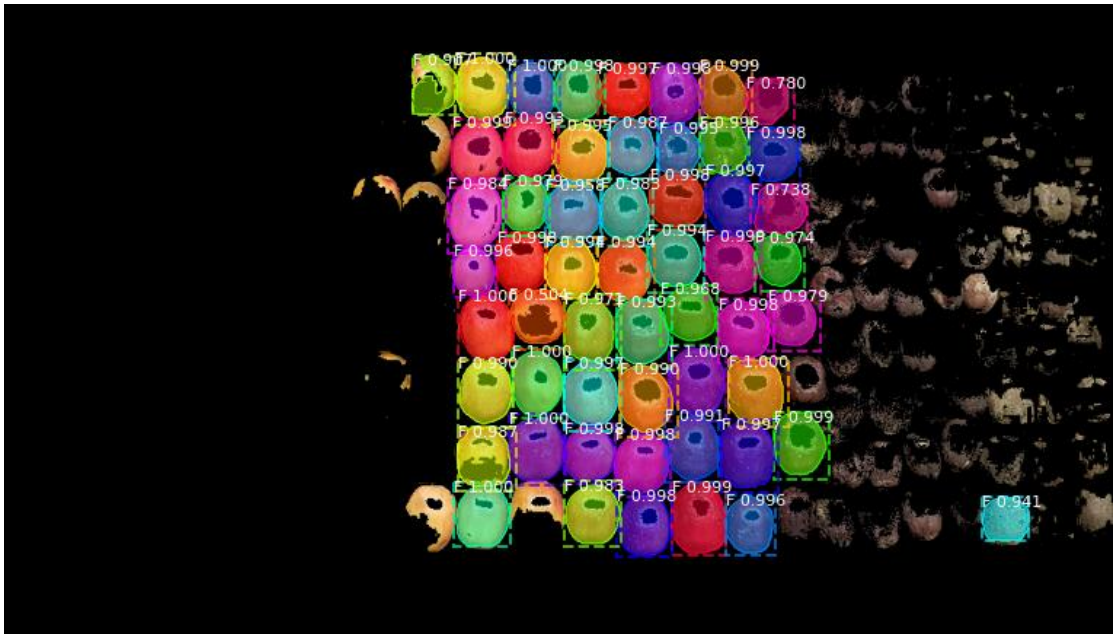
O intervalo de confiança variou de 77% a 100% e o número de instâncias encontradas foi de 69. A Figura 41 mostra um comparativo da imagem de segmentação de fundo com a imagem de segmentação C&V.

Figura 41 – Comparativo da segmentação C&V da imagem 22.

Fonte: Do autor (2023).

A segmentação CV da imagem 22 é mostrada na Figura 42.

Figura 42 – Segmentação CV da imagem 22.



Fonte: Do autor (2023).

O intervalo de confiança variou de 50% a 100% e o número de instâncias encontradas foi de 55. A Figura 43 mostra um comparativo da imagem de segmentação de fundo com a imagem de segmentação CV.

Figura 43 – Comparativo da segmentação CV da imagem 22.

Fonte: Do autor (2023).

A segmentação S da imagem 22 é mostrada na Figura 44.

Figura 44 – Segmentação S da imagem 22

Fonte: Do autor (2023).

O intervalo de confiança variou de 51% a 100% e o número de instâncias encontradas foi de 24. A Figura 45 mostra um comparativo da imagem de segmentação de fundo com a imagem de segmentação S.

Figura 45 – Comparativo da segmentação S da imagem 22.

Fonte: Do autor (2023).

5.1.2 Imagem 23

A segmentação da imagem 23 é mostrada na Figura 46.

Figura 46 – Segmentação da imagem 23.

Fonte: Do autor (2023).

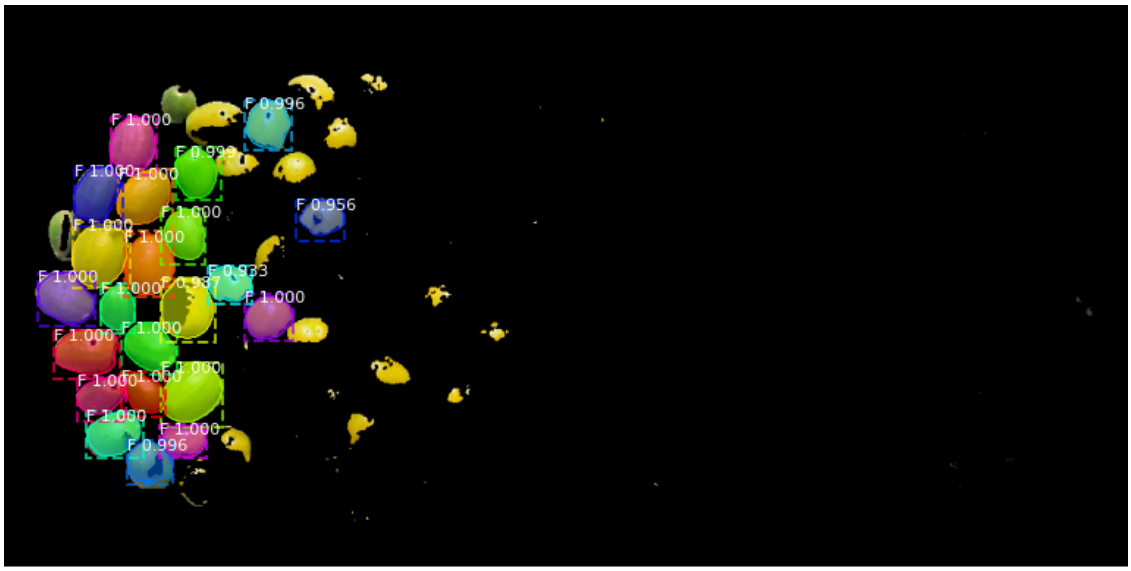
É possível observar que o algoritmo foi capaz de detectar 100% dos frutos, com um intervalo de confiança variando de 64% a 100%. A Figura 47 mostra a segmentação de fundo da imagem.

Figura 47 – Segmentação de fundo da imagem 23.

Fonte: Do autor (2023).

A segmentação C&V é mostrada na Figura 48.

Figura 48 – Segmentação C&V da imagem 23.



Fonte: Do autor (2023).

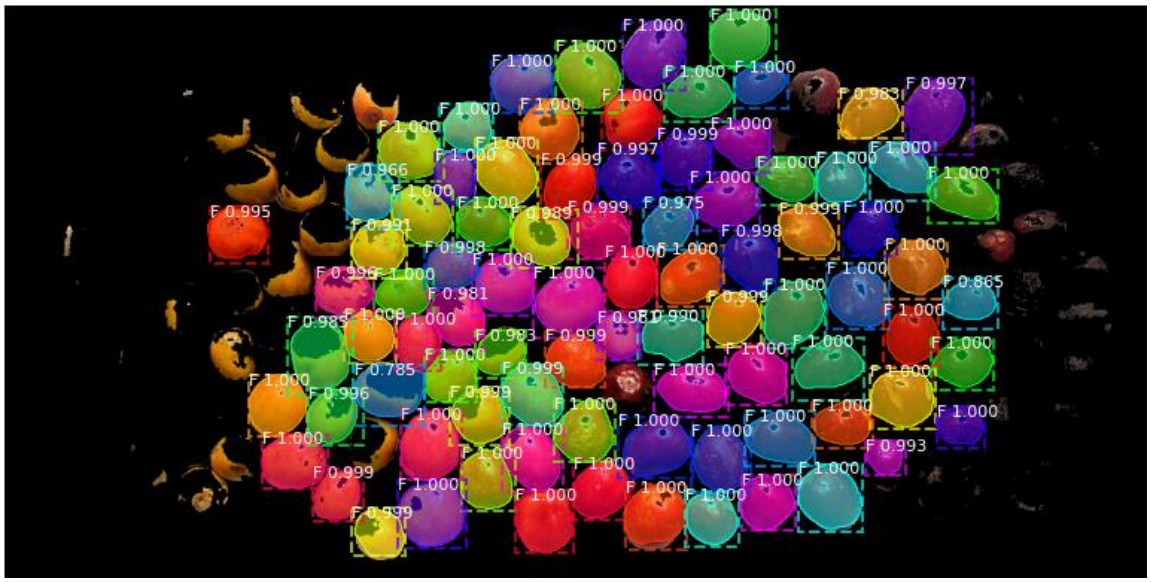
O intervalo de confiança variou de 93% a 100% e o número de instâncias encontradas foi de 22. A Figura 49 mostra um comparativo da imagem de segmentação de fundo com a imagem de segmentação C&V.

Figura 49 – Comparativo da segmentação C&V da imagem 23.

Fonte: Do autor (2023).

A segmentação CV da imagem 23 é mostrada na Figura 50.

Figura 50 – Segmentação CV da imagem 23.



Fonte: Do autor (2023).

O intervalo de confiança variou de 86% a 100% e o número de instâncias encontradas foi de 86. A Figura 51 mostra um comparativo da imagem de segmentação de fundo com a imagem de segmentação CV.

Figura 51 – Comparativo da segmentação CV da imagem 23.

Fonte: Do autor (2023).

A segmentação S da imagem 23 é mostrada na Figura 52.

Figura 52 – Segmentação S da imagem 23.

Fonte: Do autor (2023).

O intervalo de confiança variou de 97% a 100% e o número de instâncias encontradas foi de 15. A Figura 53 mostra um comparativo da imagem de segmentação de fundo com a imagem de segmentação S.

Figura 53 – Comparativo da Segmentação S da Imagem 23.

Fonte: Do autor (2023).

5.2 Resultados das segmentações das imagens de testes

Os resultados das segmentações das imagens de testes podem ser sumarizados na Tabela 2 e na Tabela 3, onde:

- T – Total de frutos.
- $\mathrm{T}_{\mathrm{seg}}$ – Total de frutos segmentados.
- $\mathrm{T}_{\mathrm{seg\%}}$ – Porcentagem do total de frutos segmentados.
- $\mathrm{C\&V}$ – Total de frutos cana e verde.
- $\mathrm{C\&V}_{\mathrm{seg}}$ – Total de frutos cana e verde segmentados.
- $\mathrm{C\&V}_{\mathrm{seg\%}}$ – Porcentagem do total de frutos cana e verde segmentados.
- CV – Total de frutos cereja vermelho.
- $\mathrm{CV}_{\mathrm{seg}}$ – Total de frutos cereja vermelho segmentados.
- $\mathrm{CV}_{\mathrm{seg\%}}$ – Porcentagem do total de frutos cereja vermelho segmentados.
- S – Total de frutos seco.
- $\mathrm{S}_{\mathrm{seg}}$ – Total de frutos seco segmentados.
- $\mathrm{S}_{\mathrm{seg\%}}$ – Porcentagem do total de frutos seco segmentados.

Tabela 2: Comparação dos resultados T e C&V.

Imagem	T	$\mathrm{T}_{\mathrm{seg}}$	$\mathrm{T}_{\mathrm{seg\%}}$	$\mathrm{C\&V}$	$\mathrm{C\&V}_{\mathrm{seg}}$	$\mathrm{C\&V}_{\mathrm{seg\%}}$
3	19	19	100%	4	5	100%
22	196	194	99%	69	69	100%
23	140	140	100%	24	22	92%

Fonte: Do autor (2023).

Tabela 3: Comparação dos resultados CV e S.

Imagem	CV	CV_{seg}	$CV_{seg\%}$	S	S_{seg}	$S_{seg\%}$
3	8	8	100%	4	3	75%
22	61	55	90%	46	24	52%
23	91	86	95%	15	15	100%

Fonte: Do autor (2023).

Esses resultados mostram que o algoritmo tem uma boa taxa detecção, segmentando de 99% a 100% dos frutos e uma boa taxa de detecção dos frutos Cana e Verde, com taxas que vão de 92% a 100%. Já para frutos Cereja Vermelho a taxa varia de 90% a 100%.

Os frutos Secos tem segmentação aquém do esperado, com taxas que variam de 52% a 100%. A principal causa desse fenômeno pode ser a baixa incidência desses frutos no banco de treinamento utilizado, dificultando o processo de aprendizado da rede neural.

O melhoramento dos intervalos de cores utilizados na segmentação também se faz necessário, principalmente nos frutos S, uma vez que é possível verificar visualmente que no processo de segmentação desses frutos ocorre várias crateras ou buracos na máscara gerada, indicando que existem faixas de cores dos pixels não cobertas totalmente pelo método de segmentação.

Como o processo de maturação não é homogêneo, muitos frutos têm dupla coloração, dificultando sua detecção por quaisquer dos métodos de segmentação.

Outra fonte de erros está relacionada com a reflexão total da luz que incide diretamente no fruto na direção da lente de captura da câmera. Essa reflexão gera pequenos círculos brancos que não foram tratados nas funções de segmentação, podendo dificultar a segunda etapa de segmentação por cores. Um bom exemplo dessas reflexões ocorre na Figura 42.

5.3 Resultados das segmentações da base de validação

As execuções referentes às imagens de teste 5, 22 e 23 e do banco de validação e seus resultados obtidos são mostrados na Figura 54, Figura 55 e Figura 56 respectivamente.

Figura 54 – Resultados gerais da primeira execução.

```
Número de instâncias encontradas na imagem de teste:  19
Número de instâncias C e V encontradas na imagem de teste:  5
Número de instâncias CV encontradas na imagem de teste:  8
Número de instâncias S encontradas na imagem de teste:  7
Número de instâncias encontradas na imagem  0  de validação:  25
AP da imagem 0  de validação:  0.8495855609588898
Número de instâncias encontradas na imagem  1  de validação:  47
AP da imagem 1  de validação:  0.9076336375632706
Número de instâncias encontradas na imagem  2  de validação:  61
AP da imagem 2  de validação:  0.9080679532623415
Número de instâncias encontradas na imagem  3  de validação:  10
AP da imagem 3  de validação:  1.0
Número de instâncias encontradas na imagem  4  de validação:  19
AP da imagem 4  de validação:  1.0
MAP:  0.9330574303569005
```

Fonte: Do autor (2023).

Figura 55 – Resultados gerais da segunda execução.

```
Número de instâncias encontradas na imagem de teste:  194
Número de instâncias C e V encontradas na imagem de teste:  69
Número de instâncias CV encontradas na imagem de teste:  55
Número de instâncias S encontradas na imagem de teste:  24
Número de instâncias encontradas na imagem  0  de validação:  25
AP da imagem 0  de validação:  0.8495855609588898
Número de instâncias encontradas na imagem  1  de validação:  47
AP da imagem 1  de validação:  0.9076336375632706
Número de instâncias encontradas na imagem  2  de validação:  61
AP da imagem 2  de validação:  0.9080679532623415
Número de instâncias encontradas na imagem  3  de validação:  10
AP da imagem 3  de validação:  1.0
Número de instâncias encontradas na imagem  4  de validação:  19
AP da imagem 4  de validação:  1.0
MAP:  0.9330574303569005
```

Fonte: Do autor (2023).

Figura 56 – Resultados gerais da terceira execução.

```
 Número de instâncias encontradas na imagem de teste:  140
Número de instâncias C e V encontradas na imagem de teste:  22
Número de instâncias CV encontradas na imagem de teste:  86
Número de instâncias S encontradas na imagem de teste:  15
Número de instâncias encontradas na imagem  0  de validação:  25
AP da imagem 0  de validação:  0.8495855609588898
Número de instâncias encontradas na imagem  1  de validação:  47
AP da imagem 1  de validação:  0.9076336375632706
Número de instâncias encontradas na imagem  2  de validação:  61
AP da imagem 2  de validação:  0.9080679532623415
Número de instâncias encontradas na imagem  3  de validação:  10
AP da imagem 3  de validação:  1.0
Número de instâncias encontradas na imagem  4  de validação:  19
AP da imagem 4  de validação:  1.0
MAP:  0.9330574303569005
```

Fonte: Do autor (2023).

Esses números mostram que a precisão média (AP) na detecção variou de 85% a 100%. Já a média da precisão média (MAP) foi de 93%, o que é um valor consideravelmente alto.

É possível observar que grande parte dos erros no processo de inferência ocorreu em razão dos frutos estarem em segundo plano (fora de foco da câmera), pois estes sofrem o chamado efeito Blur, perdendo parte de seu contorno e definição, dificultando a detecção. É importante ressaltar que estes erros causam a diminuição no valor dos indicadores AP e MAP, pois acarretam acréscimo no número de FP’s.

Na primeira execução foram detectadas 7 instâncias do fruto seco, indicando que houve duas duplas detecções e uma tripla detecção. O melhoramento da detecção pode ser feito ajustando a função de segmentação por cor do fruto do tipo seco, aprimorando o treinamento da rede neural com mais épocas e também com o ajuste dos parâmetros de threshold da rede de inferência. Esses ajustes, no entanto, podem alterar outros aspectos da detecção, como o aumento do número de falsos negativos

6 Conclusões

O resultado obtido para o indicador MAP foi de 93%, enquanto que, para imagens individuais, foram obtidos APs de 85% a 100%, indicando, portanto, elevada eficiência e eficácia na detecção. Grande parte dos erros no processo de inferência ocorreram em razão de desajustes nos intervalos de cores das funções de segmentação por cor. Outra fonte de erros foram as reflexões da cor branca nos frutos que geraram buracos, dificultando assim a segunda segmentação de instâncias. Também vale a pena citar a presença de frutos de dupla coloração, que não são detectados por nenhuma das segmentações de cores. Por fim, ocorreu a presença de alguns frutos em segundo plano (fora de foco da câmera), causando a diminuição no valor dos indicadores AP e MAP, uma vez que alguns não estão marcados no arquivo de notação e foram identificados na segmentação, aumentando assim o número de FPs.

A utilização desses algoritmos mostrou grande potencial na resolução do desafio de detecção do estágio de maturação do café, abrindo novas possibilidades no desenvolvimento de tecnologias que auxiliem nos processos de pré-colheita, colheita e pós-colheita do café. O aprimoramento dos resultados foi verificado na prática, por meio do aumento da base de dados.

Todavia, muito ainda pode ser feito em trabalhos futuros para melhoramento do algoritmo. Seria interessante aumentar a base de dados de treinamento, principalmente no que se refere aos frutos do tipo Seco, de modo a melhorar os resultados de predição desses frutos. Outro desafio seria revisar as funções de segmentação por cores, abrangendo ao máximo possível todas as cores do espectro envolvidas, incluindo o estágio de maturação “Cereja” para cultivares que produzem frutos cereja (maduros) de coloração amarelo, e o estágio de maturação “Passa” (ou sobremaduro).

É possível também refinar o método de segmentação C&V de modo a separar essas duas classes segmentando em tonalidades mais claras e escuras do espectro de cores verde.

É proposto também adaptar o algoritmo para a segmentação em tempo real através de um input de imagem vindo de uma câmera real.

7 Referências

FERREIRA, JÚNIOR et al. Dynamic behavior of coffee tree branches during mechaical harverst. COMPUTERS AND ELETRONICS IN AGRICULTURE, 173: 105415, 2020.

FERREIRA JÚNIOR et al. Characterization of the coffe fruit detachment force in crop subjected to mechanized harvesting. COFFE SCIENCE, V. 13, p. 71-79, 2018.

FERREIRA JÚNIOR et al. "Efeito dominó": Colheita mecânica do café visando qualidade. Revista Cultivar, V. 180, p. 28-31, 2017.

GIRSCHICK et al. “Mask R-CNN”: Introduction, p.1, 2013.

MORAES, H. M. et al. Escala fenológica detalhada da fase reprodutiva de Coffea arabica. Bragantia, Campinas, v. 67, n. 1, p. 257-260, 2008.

PEZZOPANE, J. R. M. et al. Escala para avaliação de estádios fenológicos do cafeeiro arábica. Bragantia, Campinas, v. 62, n.3, p. 499-505, 2003

Ministério da Agricultura e Pecuária – Brasil é o maior produtor mundial e o segundo maior consumidor de café. Disponível em < https://www.gov.br/agricultura/pt-br/assuntos/noticias/brasil-e-o-maior-produtor-mundial-e-o-segundo-maior-consumidor-de-cafe> Acesso em 11/10/2023.

LAPiX. Métricas de Visão Computacional – Mean Average Precision. Disponível em <https://lapix.ufsc.br/ensino/visao/visao-computacionaldeep-learning/visao-computacionalmetricasmean-average-precision/> Acesso em 11/10/2023.

TowardsDataScience. MAP (Mean Average Precision). Disponível em <https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2> Acesso em 10/10/2023.

Arena BR-ME. As variedades de grãos de café. Disponível em < https://arena.brme.com.br/artigos-cafes/as-variedades-de-graos-de-cafe/> Acesso em 26/10/2023.

Perfect Daily Grind. O ponto perfeito da colheita. Disponível em <https://perfectdailygrind.com/pt/2022/06/07/o-ponto-perfeito-da-colheita/> Acesso em 26/10/2023.

Palini & Alves. Selecionadora Eletrônica de Cafés. Disponível em <https://www.palinialves.com.br/produto?produto=250&pa_fullselect> Acesso em 01/10/2023.

Giselle Figueiredo Abreu. Pós-colheita do Café: Como separar os Frutos de Café para Produzir Café Especial. Disponível em < https://www.youtube.com/watch?v=jTtt576mGO0> Acesso em 03/10/2023.

Tevel Tech. Enter A New Era of Harvesting With Tevel's Flying Autonomous Robots™. Disponível em <https://www.tevel-tech.com/> Acesso em 03/10/2023.

Genial Investimentos. NVIDIA (NVDC34) Dominando a Revolução da IA. Disponível em <https://analisa.genialinvestimentos.com.br/acoes/nvidia/nvidia-nvdc34-reuniao-com-diretor-sr-de-ri-stewart-stecker/> Acesso em 23/11/2023.

VGG Image Anotator. Disponível em <https://www.robots.ox.ac.uk/~vgg/software/via/> Acesso em 23/11/2023.

Alsombra - Implementação do Mask R-CNN. Disponível em <https://www.github.com/alsombra/Mask_RCNN-TF2> Acesso em 23/11/2023.

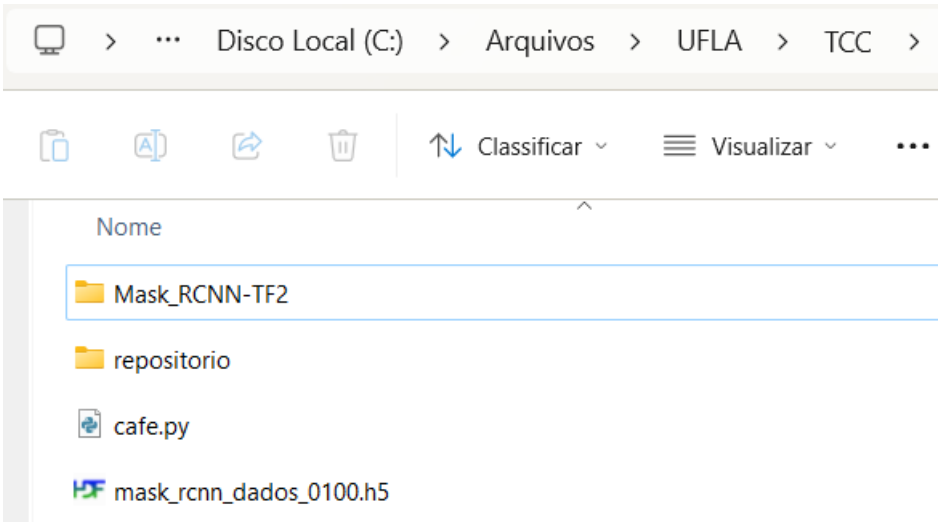
8 Apêndice

8.1 Especificações e Instalação de Software

O Anaconda Python é uma plataforma de código aberto que oferece um ambiente de desenvolvimento e gerenciamento de pacotes para Python. Para prepararmos o ambiente de desenvolvimento que executará os códigos é necessário:

- Acessar https://www.anaconda.com/download e baixar a última versão.
- Realizar a instalação padrão.
- Baixe o arquivo compactado “cafe.zip” no link https://drive.google.com/file/d/12Xe9hqmxHQoU7RgeVcYqzb0ld-GanVO8/view?usp=sharing . Esse arquivo contém o código no formato .py, o repositório de imagens, o algoritmo Mask R-CNN e o arquivo mask_rcnn_coco.h5, necessário para o processo de transferência de aprendizado.
- Descompacte os arquivos no seguinte diretório: C:/Arquivos/UFLA/TCC conforme a Figura

Figura 57 – Arquivos extraídos.

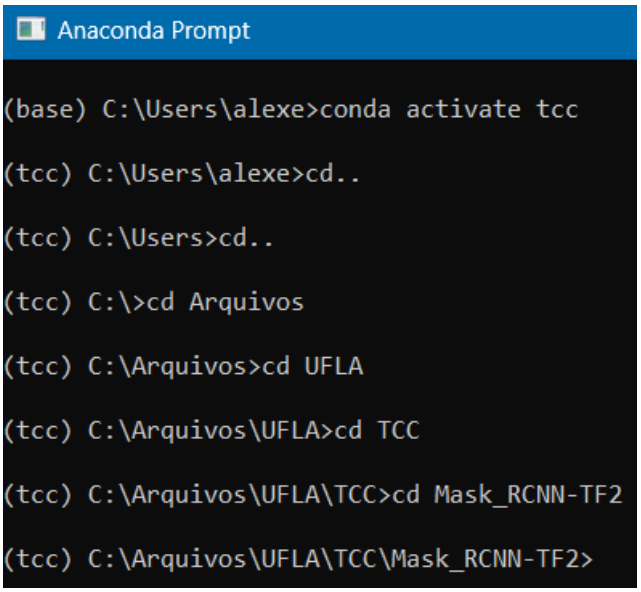


Fonte: Do autor (2023).

- Abra o “Anaconda Prompt”, que é o prompt de comandos da ferramenta Anaconda.

- Digite o comando “*conda create -n tcc python=3.9*”
- Uma mensagem irá aparecer pedindo autorização para instalar os pacotes necessários, digite “*y*” no teclado e aperte a tecla enter.
- Ao final da instalação, digite “conda activate tcc” para ativar o ambiente de desenvolvimento recém criado.
- Utilizando do comando “cd..” para retornar um diretório acima, navegue até o diretório C:/Arquivos/UFLA/TCC/Mask_RCNN-TF2.

Figura 58 – Navegação pelos diretórios.



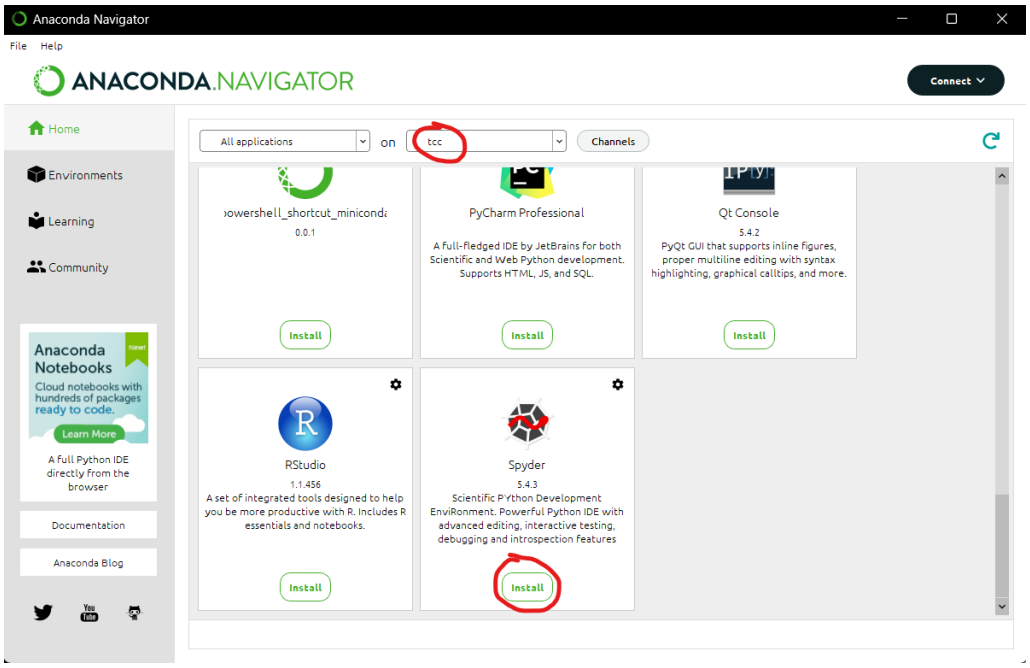
Fonte: Do autor (2023).

- Digite o comando “python -m pip install -r requirements.txt” para instalar as dependências especificadas no arquivo “requirements.txt”.
- Eventualmente, uma mensagem “There was na error checking the latest version of pip” pode ser exibida em amarelo interrompendo a instalação dos pacotes. Caso ocorra, abra o Windows PowerShell e digite o comando: “rm -r $env:LOCALAPPDATA\pip\cache\selfcheck\” para limpar as variáveis locais no cache relacionadas ao pip e volte a tentar a instalação.
- Digite o comando exit() para sair do prompt do python e digite o comando “python setup.py install” para instalar o Mask R-CNN.
- Ao final do processo, entre no console python digitando o comando “python” e digite “import mrcnn”, caso o console vá para a próxima linha

sem mostrar nenhuma mensagem a biblioteca foi importada corretamente e está pronta para ser utilizada. Digite “exit()” e na sequência “exit” denovo para fechar o prompt.

- Abra o Anaconda Navigator, mude a aba de ambiente para tcc e instale o software Spyder e o abra conforme mostra a Figura 59.

Figura 59 – Anaconda Navigator.



Fonte: Do autor (2023).

- Abra o Spyder, carregue o arquivo cafe.py e execute.