Detecção de Morangos Maduros implementando YOLO

1º Lucas Eduardo Cavalcanti de Oliveira
Centro de Tecnologia e Geociências
Universidade Federal de Pernambuco
Recife, Brasil
leco@cin.ufpe.br

***Abstract*— Este estudo apresenta uma solução inovadora para a detecção de morangos maduros, aplicando a arquitetura de rede neural YOLO (You Only Look Once) em um contexto de agricultura automatizada. Através de uma estratégia modular que incorpora configurações dinâmicas, treinamento adaptativo e avaliação rigorosa, desenvolvemos um modelo altamente eficaz capaz de identificar morangos prontos para a colheita em imagens. Utilizando um conjunto de dados cuidadosamente anotado, o modelo aprende a distinguir características visuais significativas de morangos maduros, otimizando o processo de colheita e classificação pós-colheita.**

Keywords*— *YOLO, detecção de morangos, agricultura automatizada, visão computacional, aprendizado profundo.

I. Introdução

A aplicação do YOLO para detecção de morangos maduros tem potencial significativo em áreas como agricultura automatizada e sistemas de colheita robótica. Ao identificar precisamente a localização e o estágio de maturação dos frutos, tais sistemas podem otimizar o processo de colheita, selecionando apenas os frutos prontos para a colheita, enquanto minimizam o dano aos frutos e plantas. Além disso, essa tecnologia pode ser aplicada para monitoramento de qualidade e classificação automatizada de produtos em pós-colheita, melhorando a eficiência e reduzindo o desperdício. A detecção de objetos, como um pilar central da visão computacional, demanda arquiteturas de aprendizado profundo altamente eficientes e adaptáveis. Este estudo apresenta uma abordagem modular para a construção, treinamento, e avaliação de modelos de detecção de objetos, utilizando uma configuração flexível que permite ajustes rápidos e eficazes em várias etapas do processo de desenvolvimento. Iniciamos com a implementação da classe Config, um núcleo centralizado para a gestão de configurações experimentais, que abrange desde a definição de caminhos de diretórios até a especificação de parâmetros de treinamento e modelo. Essa modularidade é crucial para a adaptação a diferentes conjuntos de dados e objetivos de pesquisa.

II. Metodologia

A. Coleta de Dados

A arquitetura começa com a classe Config, um mecanismo centralizado para gerenciar todas as configurações necessárias do projeto. Isso inclui caminhos para armazenamento de dados e modelos, parâmetros para carregamento e processamento de dados, e especificações de modelos, como arquiteturas e pesos pré-treinados. Esse design facilita a adaptação e escalabilidade do projeto, permitindo ajustes rápidos e eficientes nas configurações sem necessidade de alterações profundas no código.

```
[ ]  class Config:
        def __init__(self):
            self.checkpoint_dir = '/content/drive/MyDrive/checkpoints'
            self.experiment_name = '/content/drive/MyDrive/experiment_name'
            self.data_dir = '/content/drive/MyDrive/datasets'

            self.dirs = self.setup_data_dirs(['train', 'valid', 'test'])

            self.classes = ['strawberry']
            self.num_classes = len(self.classes)

            self.dataloader_params = {
                'batch_size': 4,
                'num_workers': 2
            }

            self.model_params = self.setup_model_params('yolo_nas_l', 'coco')

        def setup_data_dirs(self, phases):
            dirs = {}
            for phase in phases:
                image_dir = f"{self.data_dir}/{phase}/images"
                label_dir = f"{self.data_dir}/{phase}/labels"
                dirs[f"{phase}_images_dir"] = image_dir
                dirs[f"{phase}_labels_dir"] = label_dir
            return dirs

        def setup_model_params(self, model_name, pretrained_weights):
            return {
                'model_name': model_name,
                'pretrained_weights': pretrained_weights
            }

    config = Config()
```

B. Desenvolvimento da rede

O YOLO, sendo uma das arquiteturas de detecção de objetos mais rápidas e eficientes, oferece uma solução ideal para a tarefa em questão. Seu design único, que analisa a imagem como um todo para detectar objetos, permite que opere em tempo real, uma vantagem significativa para aplicações agrícolas onde a rapidez pode ser essencial para processos como a colheita. A decisão de treinar esta rede utilizando o conjunto de dados COCO (Common Objects in Context) é motivada pela diversidade e abrangência deste dataset, que proporciona uma base sólida para o aprendizado de características visuais complexas. Optamos por treinar o modelo por 30 épocas, um número que representa um equilíbrio entre alcançar um desempenho adequado e manter a eficiência computacional. Esse período de treinamento é suficiente para permitir que a rede ajuste seus pesos e aprenda a identificar morangos maduros com precisão, sem incorrer em overfitting ou consumir recursos excessivos. Ajustes e adaptações foram realizados no conjunto de dados COCO para incluir anotações específicas de morangos maduros, garantindo que o modelo aprendesse a reconhecer esse fruto específico em diferentes condições e estágios de maturação.

C. Dataloaders e Processamento de Dados

Funções específicas são usadas para carregar e preparar os datasets de treinamento, validação e teste, garantindo que os dados estejam no formato adequado para o modelo. Isso inclui a organização de imagens e anotações (labels), e a aplicação de transformações necessárias. A modularidade nesta etapa permite a fácil integração de diferentes conjuntos

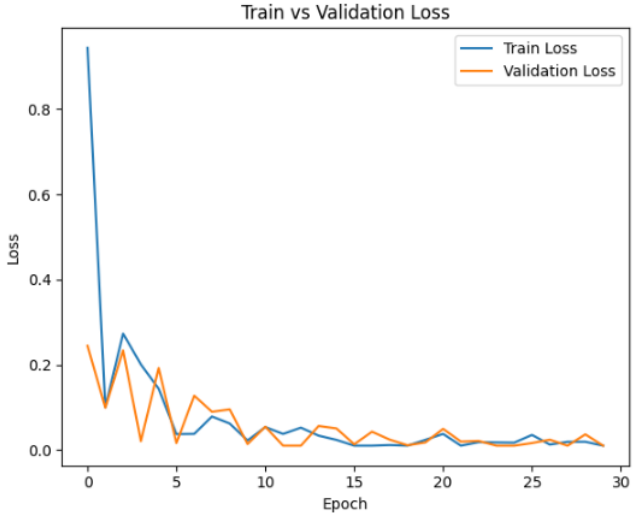
de dados e formatos, aumentando a versatilidade da arquitetura.

D. Treinamento do modelo

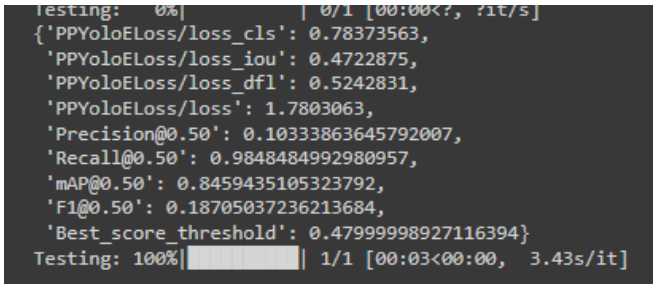
O treinamento é gerenciado por uma instância Trainer, que utiliza as configurações definidas para conduzir o processo de treinamento. Isso inclui a preparação dos dados, a execução das iterações de treinamento, e a validação do modelo. A flexibilidade é novamente um ponto chave, com parâmetros detalhados de treinamento permitindo a customização do processo para atender a objetivos específicos, como a implementação de técnicas de otimização avançadas e ajustes dinâmicos do learning rate.

III. CONCLUSÃO

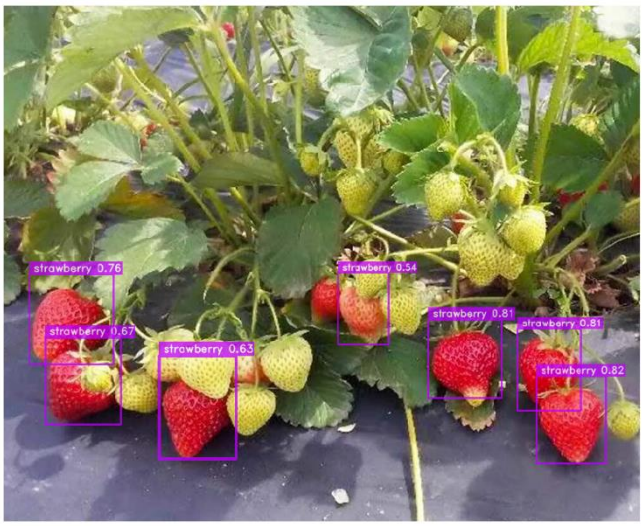
Após o treinamento, o modelo é avaliado utilizando um conjunto de dados de teste, com métricas específicas de detecção de objetos fornecendo uma avaliação quantitativa do desempenho. A capacidade de carregar modelos a partir de checkpoints específicos permite a análise de diferentes estados do modelo, facilitando a identificação da melhor configuração para a tarefa em questão.



Foi avaliado também algumas métricas para entender melhor como foi treinada a rede e como podemos analisar melhor.



Finalmente, a arquitetura permite a aplicação do modelo treinado para fazer previsões em novas imagens, demonstrando a capacidade do modelo de generalizar a partir dos dados de treinamento. A funcionalidade de visualização das previsões ajuda na interpretação dos resultados, permitindo ajustes finos e validação qualitativa do modelo.



A integração da arquitetura YOLO treinada com o COCO para a detecção de morangos maduros é uma demonstração poderosa do potencial da visão computacional na otimização da agricultura. Este projeto não apenas prova a aplicabilidade dessas tecnologias avançadas em ambientes agrícolas, mas também estabelece um precedente para futuras pesquisas e desenvolvimento, potencialmente abrindo caminho para a automação completa de várias outras tarefas agrícolas. À medida que continuamos a explorar e expandir os limites do que é possível com o aprendizado de máquina e a visão computacional, podemos esperar ver mais inovações que trazem eficiências significativas e melhorias na sustentabilidade para a agricultura global.

REFERENCES

[1] "Deep Learning with Python" by François Chollet (Manning Publications; 2017)

[2] "Ultralytics YOLOv8 Documentos" By https://docs.ultralytics.com/pt/models/yolo-nas/

[3] "Deep Learning" by Adam Gibson, Josh Patterson (O'Reilly Media, Inc.; 2017)

[4] https://docs.deci.ai/super-gradients/YOLONAS.html

[5] Super-Gradient Disponível em: https://github.com/Deci-AI/super-gradients/