UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE RUSSAS

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

WILLIANA LUZIA SOUSA LEITE

TRABALHO FINAL - VISÃO COMPUTACIONAL

RUSSAS

2020

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

AP	*Average Precisison*
CNN	Rede Neural Convolucional ou *Convolutional Neural Network*
Fast-RCNN	*Fast Region-based Convolutional Neural Network*
Faster-RCNN	*Faster Region-based Convolutional Neural Network*
IoU	*Intersection Over Union*
mAP	*mean Average Precisison*
Mask-RCNN	*Mask Region-based Convolutional Neural Network*
RoI	*Region of Interest*
RPN	*Region Proposal Networks*
YOLO	*You Only Look Once*

SUMÁRIO

1 PROPOSTA

Este trabalho tem como objetivo realizar a detecção frutos separadas em duas classes: maçãs saudáveis e maçãs danificadas. Para tal, será utilizado a arquitetura YOLO e o framework Mask-RCNN para detecção de objetos, para fins de comparação e análise dos resultados obtidos.

1.1 Justificativa e Objetivos

Considerado em muitos artigos como o estado da arte em detecção de objetos, o YOLO oferece uma rede robusta, com alta performance e um ótimo desempenho em relação aos outros frameworks encontrados na literatura. Dessa forma, investigando as características da base de dados escolhida para este trabalho, como por exemplo tamanho dos objetos, utilizar o YOLO parece ser uma escolha sensata.

Analisando o site oficial, disponível em: <https://pjreddie.com/darknet/yolo/>, em um primeiro momento, é apresentado uma explicação sucinta de como o modelo funciona e também fornece um tutorial aparentemente fácil de como utilizar o framework. Na tentativa de realizar o tutorial fornecido pelo site oficial e ao mesmo tempo adaptando-o para a base de dados utilizada neste trabalho, diversos problemas ao longo da execução surgiram, desde problemas com ambientes, até capacidade de processamento da máquina.

Estes problemas impulsionaram uma busca por outras formas de utilizar a arquitetura YOLO, sem precisar usar diretamente o framework oficial. A maneira mais simples encontrada, seria utilizando a biblioteca ImageAI, que permite utilizar a arquitetura do YOLO para detecção de objetos, disponível no link: <https://imageai.readthedocs.io/en/latest/customdetection/>, onde é apresentado um tutorial simples de como utilizá-la.

Como o YOLO apresenta uma maneira diferente de lidar com o problema da detecção de objetos, que seria baseada em regressão, neste ponto me pareceu relevante analisar e comparar os resultados obtidos desta técnica em comparação com outra. A técnica escolhida para comparar os resultados foi a detecção baseada em regiões, para tal será utilizado a rede Mask-RCNN, também considerada o estado da arte para algoritmos de detecção e segmentação de objetos baseados em regiões. Para a implementação deste trabalho será utilizado a biblioteca mrcnn, que pode ser encontrada neste link: <https://github.com/matterport/Mask_RCNN>.

2 BASE DE DADOS

A base de dados possui 713 imagens, de ambas as classes, disponibilizada em um arquivo compactado chamado: *apple_detection_dataset*, de tamanho 5,67 MB. O *dataset*, após descompactado, possui a seguinte estrutura padrão:

1. *Train*: 563 imagens de tamanho variado na pasta images e 563 arquivos de anotações referente as imagens de treino.
2. *Validation*: 150 imagens de tamanho variado na pasta images e 150 arquivos de anotações referente as imagens de treino.

Esta configuração da base de dados é a forma como os dados originalmente foram organizados, mas nada impede que sejam redistribuídas com proporções diferentes para fins de investigação dos resultados.

Na Figura 1, a esquerda é apresentado um exemplo de imagem retirado dos dados de treino e na direita é apresentado um exemplo de imagem retirado dos dados de teste.

Figura 1 – Exemplo de imagens existentes no *dataset*.

Fonte: Base de dados de treinamento.

Fonte: Base de dados de validação e teste.

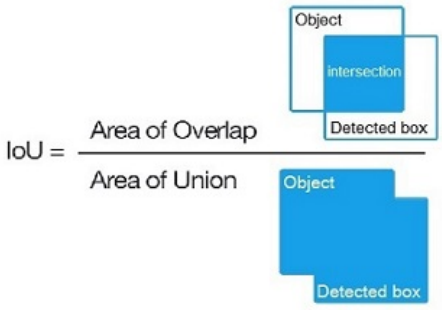
3 YOLO

Conforme explicado em Chablani (2017), a rede YOLO modela o problema da detecção de objetos como um único problema de regressão, diretamente dos *pixels* da imagem para as coordenadas das caixas delimitadoras e as probabilidades das classes. Segundo os criadores Redmon *et al.* (2016), o YOLO divide a imagem de entrada em uma grade *SS*, se o centro de um objeto cair em uma célula da grade, essa célula é responsável por detectar esse objeto.

Cada célula da grade prevê caixas delimitadoras B e pontuações de confiança para essas caixas. Essas pontuações de confiança refletem o grau de confiança do modelo em determinar que a caixa contém o objeto e também a precisão que ele acha que a caixa prevê. Formalmente, definimos confiança como $Pr(Objeto)IoU$. Se nenhum objeto existir nessa célula, as pontuações de confiança devem ser zero. Caso contrário, queremos que a pontuação de confiança seja igual à intersecção sobre a união entre a caixa prevista e a verdade básica (*ground truth*).

O IoU é calculado como a relação entre a intersecção da caixa delimitadora prevista com a caixa delimitadora correta sobre a união das duas caixas delimitadoras, na Figura 2 há uma representação do cálculo

Figura 2 – Cálculo do IoU.



Fonte: Elaborado pelo Autor (2019).

Cada caixa delimitadora consiste em 5 previsões: x, y, w, h e a confiança. As coordenadas (x, y) representam o centro da caixa em relação aos limites da célula da grade. A largura e a altura são previstas em relação à imagem inteira. Finalmente, a previsão de confiança representa o IoU entre a caixa prevista e qualquer caixa *ground truth*. Cada célula da grade também prevê probabilidades de classe condicional C, $Pr(Classi|Object)$.

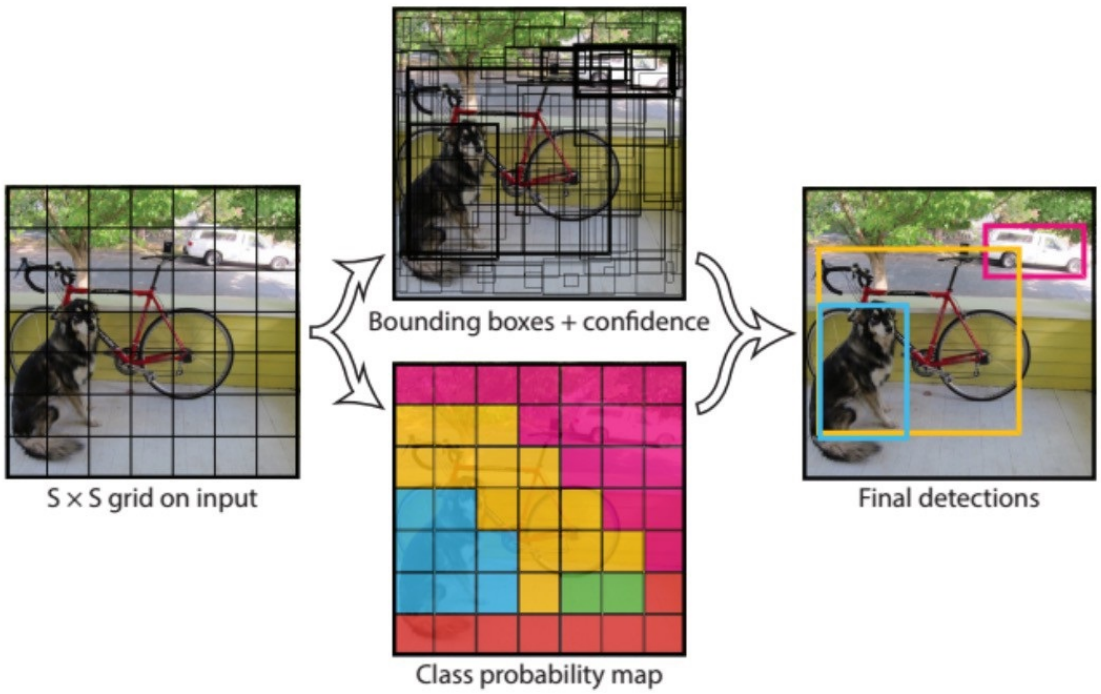
Essas probabilidades são condicionadas na célula da grade que contém um objeto.

Prevemos apenas um conjunto de probabilidades de classe por célula da grade, independentemente do número de caixas B. No momento do teste, é multiplicado as probabilidades da classe condicional e as previsões de confiança da caixa individual, ficando dessa forma:

$$Pr(Classi|Objeto) * Pr(Objeto) * IoU = Pr(Classi) * IoU$$

Com isso, as pontuações de confiança específicas de classe para cada caixa são encontradas. Essas pontuações refletem tanto a probabilidade dessa classe aparecer na caixa quanto a caixa prevista se encaixar no objeto. Na Figura 3 é apresentado um modelo simplificado do funcionamento do YOLO.

Figura 3 – Representação simplificada do modelo YOLO.



Fonte: Redmon *et al.* (2016)

4 MASK-RCNN

A rede *Mask Region-based Convolutional Neural Network* (Mask-RCNN) é a união da rede de detecção de objetos Faster-RCNN, mais a segmentação de instância. A diferença entre as duas redes é que a Mask-RCNN possui uma ramificação adicional para prever máscaras de segmentação em cada *Region of Interest* (RoI) de *pixel* a *pixel*. O escopo deste trabalho se reduz apenas a detecção de objetos, logo, apenas o módulo de detecção de objetos da rede Mask-RCNN será relevante para a pesquisa. O módulo de detecção de objetos do Mask-RCNN, como dito anteriormente, nada mais é que a rede Faster-RCNN, portanto para compreender a parte teórica da detecção de objetos com a rede em questão, somente é necessário compreender o funcionamento da rede Faster-RCNN.

Uma Faster-RCNN é dividida basicamente em 2 módulos, sendo o primeiro módulo composto por uma rede *Region Proposal Networks* (RPN) e o segundo módulo composto pelo detector que utiliza as regiões propostas do módulo anterior para fazer as detecções. Na Figura 4 é apresentado a estrutura da rede Faster-RCNN.

4.1 RPN

O primeiro módulo é responsável pela geração de propostas de regiões. Segundo Ren *et al.* (2015a) o RPN ao receber como entrada uma imagem de qualquer tamanho sua saída deve ser um conjunto de propostas regiões. De acordo com os autores, a rede RPN funciona da seguinte forma: primeiro a imagem é enviada a uma Rede Neural Convolucional ou *Convolutional Neural Network* (CNN) para extração de características, como resultado desta etapa um mapa de recursos é gerado. A partir do mapa de recursos o RPN aplica uma janela deslizante para cada ponto do mapa, onde para cada janela se propõe no máximo k objetos.

Para definir o tamanho das caixas delimitadoras, que são essencialmente 4 coordenadas que formam retângulos envolvendo um determinado objeto, o RPN utiliza o conceito de âncoras que são caixas delimitadoras de referência que por padrão, possuem 3 escalas e 3 proporções, produzindo k = 9 âncoras em cada posição de deslizamento.

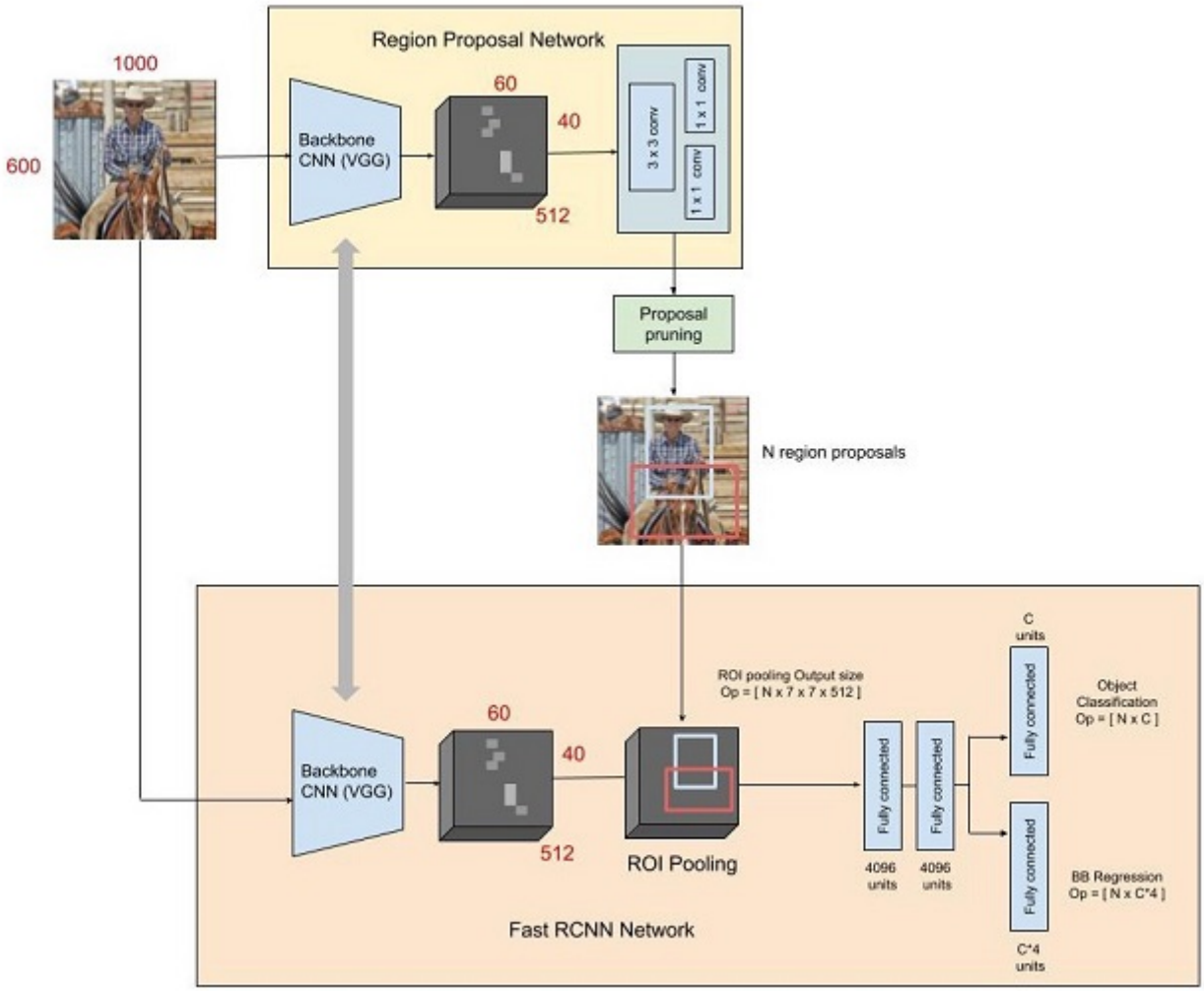
4.2 Detector

O módulo de detecção utiliza a rede *Fast Region-based Convolutional Neural Network* (Fast-RCNN) para realizar a tarefa de detecção. De acordo com Ren *et al.* (2015b) o

detector Fast-RCNN funciona da seguinte forma: dado um conjunto de propostas de regiões de uma determinada imagem, chamadas de RoI, estas propostas são passadas para uma CNN, para extração de recursos/características da imagem. Os recursos extraídos são enviados para a camada de *RoI Pooling* onde cada RoI é agrupada em um mapa de características de tamanho fixo.

Após a aplicação da camada *RoI Pooling*, as informações são mapeadas para um vetor de recurso que servirá como entrada para duas camadas totalmente conectadas. A rede possui dois vetores de saída por RoI: a primeira aplica a função Softmax, para obter a probabilidade de uma RoI pertencer a determinada classe e a segunda saída é responsável pela regressão das coordenadas das caixas delimitadoras por classe.

Figura 4 – Arquitetura detalhada da Faster-RCNN, uma rede unificada para detecção de objetos.



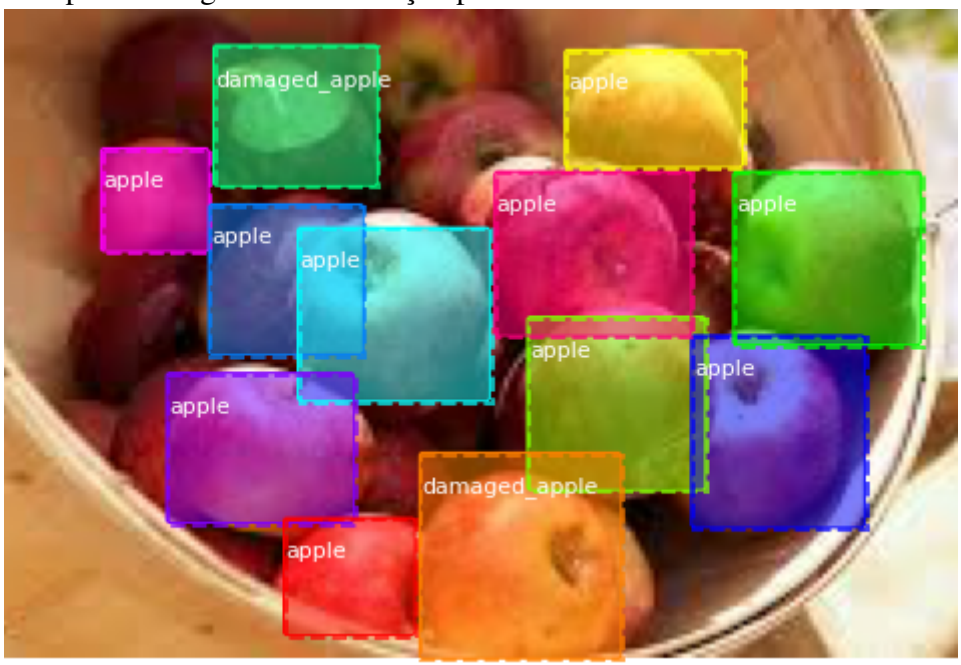
Fonte: Ananth (2019)

5 ANALISANDO RESULTADOS

Antes de investigar os resultados obtidos no modelo, é importante analisar a qualidade das informações da base de dados. Este ponto é importante visto que a qualidade da base de dados influência diretamente no desempenho do modelo. A base de dados utilizada, possui uma quantidade baixa de dados em relação a quantidade utilizada em outros algoritmos de visão computacional.

Outra característica relevante da base de dados pode ser observada na Figura 5, onde evidentemente algumas maçãs não estão marcadas. Com isso pode-se esperar que os algoritmos tenham uma dificuldade além do esperado no aprendizado, visto que, dados parcialmente corretos e com uma baixa quantidade de informações gerem naturalmente uma confusão para os algoritmos.

Figura 5 – Exemplo de imagem com marcação parcialmente correta.



Fonte: Base de dados.

Como a base de dados é pequena, apesar de não ser uma boa prática, os dados de validação e teste são iguais, para ambos os casos. Os dois modelos, executaram o treinamento ao longo de 50 épocas, com algumas configurações ligeiramente diferentes.

A métrica utilizada para avaliar os dois algoritmos foi a *mean Average Precisison* (mAP). A métrica mAP é definida como a média de todas as AP's calculadas, onde a AP é calculada para cada imagem, e para cada classe. Portanto, uma mAP é definida como:

$$MAP = \frac{\sum_{n=1}^{N} AP(n)}{N}$$

Onde N é a quantidade total de dados a serem analisados.

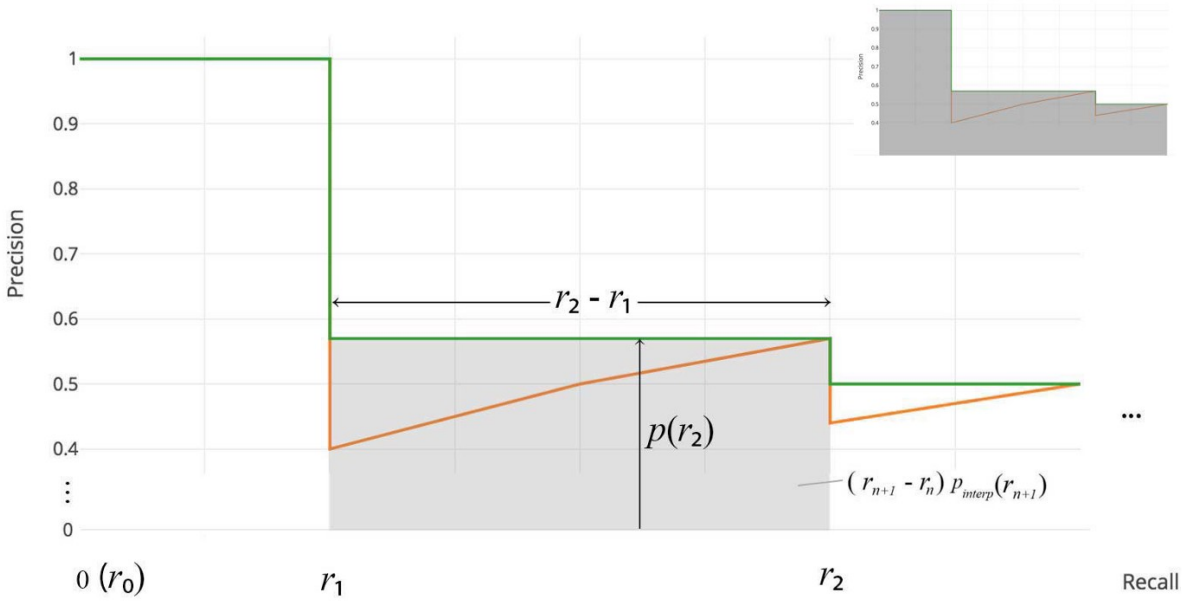
O AP é calculado da seguinte forma:

$$AP = \sum (r_{n+1} - r_n)\, p_{interp}\,(r_{n+1})$$

$$p_{interp}(r_{n+1}) = \underset{\tilde{r} \geq r_{n+1}}{max}\; p(\tilde{r})$$

Na Figura 6 é apresentado um exemplo de representação do AP.

Figura 6 – Exemplo de representação do AP.



Fonte: Ananth (2019)

5.1 Resultados YOLO

O YOLO foi treinado, seguindo a estrutura de arquivos padrão da base de dados, como foi apresentado no Capítulo 2, separando 563 imagens para treino e 150 para validação e teste.

Os resultados obtidos após o treinamento do modelo, são apresentados a seguir:

Tabela 1 – Resultados da detecção de acerolas.

Conjunto	mAP-Geral	mAP-Apple	mAP-Damaged-Apple	IoU
Teste	74.1%	82.7%	65.5%	0.5

Fonte: Elaborado pelo Autor (2020).

Ao analisar os resultados, pode-se observar que o modelo teve bastante dificuldade em detectar precisamente maçãs danificadas, este é um comportamento esperado, visto que a principal dificuldade do algoritmo seria identificar pequenas deformidades/características que tornam uma maçã danificada. No entanto, por se tratar de um objeto relativamente grande nas imagens, era de se esperar uma precisão maior. Dito isso uma possível melhoria nestes resultados poderia acontecer caso aumentasse o número de imagens na base de dados, e com isso houvesse algumas modificações nos hiperparâmetros da rede.

5.2 Resultados Mask-RCNN

O Mask-RCNN foi treinado utilizando 491 imagens para treino e 222 para validação e teste. Note que fiz um balanceamento das imagens diferente, em busca de melhores resultados.

Os resultados obtidos após o treinamento do modelo, são apresentados a seguir:

Tabela 2 – Resultados da detecção de acerolas.

Conjunto	mAP-Geral	mAP-Apple	mAP-Damaged-Apple	IoU
Teste	74.6%	70.1%	79.0%	0.5

Fonte: Elaborado pelo Autor (2020).

Ao analisar os resultados, pode-se observar que o modelo teve um resultado bastante semelhante ao obtido com o YOLO. O modelo ainda continuou apresentando dificuldade em detectar precisamente maçãs danificadas, como no YOLO, este é um comportamento esperado, visto que a principal dificuldade do algoritmo seria identificar pequenas deformidades/características que tornam uma maçã danificada. No entanto, por se tratar de um objeto relativamente grande nas imagens, era de se esperar uma precisão maior. Dito isso uma possível melhoria nestes resultados poderia acontecer caso aumentasse o número de imagens na base de dados, e com isso houvesse algumas modificações nos hiperparâmetros da rede.

5.3 Conclusão

Ao investigar os resultados do Mask-RCNN, com o novo balanceamento da quantidade de imagens de treino e validação/teste, pode-se observar que o aumento da quantidade de imagens na etapa de validação, repercutiu no desempenho do modelo que conseguiu aprender mais sobre a classe *damaged_apple*, este comportamento pode ser justificado, dado que quanto maior a qualidade dos dados de validação maior a capacidade do modelo avaliar seu aprendizado durante o treinamento. Com a redistribuição das imagens, o modelo equilibrou o seu aprendizado entre as classe.

De forma geral, os dois algoritmos tiverem desempenhos semelhantes, o que de fato teve maior discrepância entre os mesmos foi a capacidade de aprendizado de cada classes especificamente. No YOLO, a disparidade de precisão entre as classes foi muito alta em comparação com o Mask-RCNN. Em termos gerais, acredito que as diferenças entre precisão das redes podem ser mais evidenciadas com o aumento da quantidade de imagens na base de dados, onde as redes podem apresentar verdadeiramente suas reais capacidades de aprendizado.

Este projeto possuí sua implementação disponível no Github, através deste repositório: <https://github.com/WillianaLeite/apple-detection>, com somente as implementações disponíveis, como também está disponível em uma pasta Drive no seguinte link: <https://drive.google.com/drive/folders/1Y9C7GWX9KeLoGXEcMq9rFdJatD6nCO4K?usp=sharing>, onde também apresenta todas as implementações, com suas respectivas bases de dados.

REFERÊNCIAS

ANANTH, S. **Faster R-CNN for object detection**. 2019. Disponível em: <https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46>.

CHABLANI, M. **YOLO — You only look once, real time object detection explained**. 2017. Disponível em: <https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006>.

REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. You only look once: Unified, real time object detection. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 779–788.

REN, S.; HE, K.; GIRSHICK, R.; SUN, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2015. p. 91–99.

REN, S.; HE, K.; GIRSHICK, R.; SUN, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2015. p. 91–99.