Detecção de frutos por imagem com redes neurais convolutivas: um tutorial

10

Image-based fruit detection using convolutional neural networks: a tutorial

Thiago Teixeira Santos[1]

[1] *Pesquisador, Embrapa Agricultura Digital, Campinas (SP), Brasil, thiago.santos@embrapa.br*

RESUMO

Na agricultura digital, a detecção de frutos por imagem é uma forma automática e não destrutiva para monitoramento de culturas. Ao se realizar uma varredura por imagem dos pomares, é possível caracterizar sua variabilidade espacial, fornecendo dados para uma gestão agrícola adequada segundo os preceitos da agricultura de precisão. Nos últimos anos, as redes neurais se estabeleceram como uma técnica eficaz para uso em campo, ambiente não controlado no qual aplicações de visão artificial eram impraticáveis. Neste breve tutorial, abordamos os principais passos envolvidos no treinamento de redes neurais artificiais para detecção de frutos, com exemplos em pomicultura e viticultura com código disponível, facilmente extensíveis a outras culturas.

Palavras-chave: agricultura de precisão; agricultura digital; rede; Embrapa; Brasil.

ABSTRACT

In the realm of digital agriculture, fruit detection by image analysis has emerged as an automated and non-destructive approach to crop monitoring. The use of image scanning techniques in orchards, allows the characterization of spatial variability, providing valuable data for implementing appropriate agricultural management practices aligned with the principles of precision agriculture. Recently, neural networks have established themselves as an effective technique for field applications, where uncontrolled environments previously rendered traditional computer vision applications impractical. This concise tutorial delves into the key steps involved in training artificial neural networks for fruit detection, providing illustrative examples in apple production and viticulture accompanied by readily accessible code, which can be easily extended to other crops.

Keywords: precision agriculture; digital agriculture; network; Embrapa; Brazil.

1 INTRODUÇÃO

A detecção de frutos e espigas em imagens é um componente importante em diversas aplicações agrícolas, como previsão de safra (Nuske et al., 2014; Olenskyj et al., 2022), melhoramento genético de cultivares (Jiang; Li, 2020) e colheita automatizada (Silwal et al., 2017; Williams et al., 2020). Com o advento de sistemas de aprendizado de máquina baseados em *redes neurais convolutivas*, capazes de realizar detecção de objetos em imagens naturais de forma acurada, a detecção de frutos e outras estruturas de plantas em campo se tornou viável e largamente acessível (Figura 1). Proposta em 2015 como uma das primeiras redes neurais capazes de detectar objetos com alta precisão, a arquitetura Faster R-CNN (Ren et al., 2015) foi rapidamente adotada na detecção de frutos, em 2016 (Sa et al., 2016; Stein et al., 2016), para pimentões, melões, morangos e mangas. O mesmo ocorreu com a arquitetura YOLO e suas variações (Redmon; Farhadi, 2017), largamente empregada na detecção de frutos como maçãs (Tian et al., 2019), laranjas (Mirhaji et al., 2021) e uvas (Santos et al., 2020). Diversas outras arquiteturas vêm sendo empregadas na detecção de frutos na agricultura ao longo dos anos, apesar de uma aparente hegemonia das arquiteturas Faster R-CNN, YOLO (e derivados) e Mask R-CNN (He et al., 2017).

https://doi.org/10.4322/978-65-86819-38-0.1000091

Outrora restritas a grupos de pesquisa especializados em visão computacional, atualmente as redes neurais convolutivas são de fácil utilização, tanto em seu treinamento (sua adaptação ao objeto de interesse pretendido) como em sua implantação, mesmo em sistemas embarcáveis. Arquiteturas para detecção de objetos encontram-se disponíveis em bibliotecas de software para redes neurais como Keras, PyTorch e MXNet e também em bibliotecas especializadas em visão computacional como OpenCV, OpenMMLab e GluonCV[1]. Porém, mesmo com seu uso facilitado em bibliotecas modernas, alguns princípios devem ser bem entendidos pelo público interessado em aplicar tais ferramentas.

Este tutorial se destina a pesquisadores, estudantes, fabricantes de máquinas e desenvolvedores em *Agtechs* interessados em ferramentas baseadas em imagem capazes de detectar frutos, espigas, flores, folhas ou outras estruturas de interesse em campo, casas de vegetação, estufas ou laboratórios, caracterizando assim a variabilidade espacial encontrada na cultura. Exemplos de código e documentação que acompanham este tutorial estão disponíveis em um repositório público[2] e devem ser acompanhados conjuntamente a este texto. Duas bases de dados públicas para detecção de frutos serão utilizadas neste tutorial, a WGISD (Santos et al., 2020), produzida para a viticultura na Embrapa, e a MinneApple (Häni et al., 2020), produzida para a pomicultura na Universidade de Minnesota, EUA. Os exemplos no tutorial empregam a MMDetection[3], biblioteca para detecção de objetos implementada em PyTorch. Na data de redação deste tutorial (setembro de 2022), a MMDetection apresentava implementações para as principais arquiteturas neurais em detecção de objetos. Seu uso se baseia em *arquivos de configuração* que definem a origem dos dados, a arquitetura utilizada e o processo de treinamento, bem como todos os parâmetros envolvidos. Trata-se de uma metodologia organizada para gerir o fluxo de trabalho necessário ao desenvolvimento de *modelos* baseados em redes neurais para detecção.

Este tutorial *não* visa ser uma introdução ao *deep learning* (aprendizado de representações por redes neurais), sobretudo devido às limitações de espaço. Os leitores interessados em uma introdução ao *deep learning* com a devida apresentação de seus fundamentos, mas ainda com foco em exemplos práticos de código, podem encontrar em Chollet (2021) um material mais adequado. Já os leitores interessados em uma introdução teórica e matemática detalhada poderão encontrá-la em Bengio et al. (2017).

2 VISÃO GERAL DO FLUXO DE TRABALHO

O método empregado neste tutorial pertence à área do *aprendizado de máquina supervisionado*. No aprendizado de máquina, ao invés de desenvolvermos um programa explicitamente codificado para uma determinada tarefa, desenvolvemos um *modelo*. Tal modelo é um programa indiretamente desenvolvido por um processo de *treinamento*, no qual são apresentados os dados de entrada e os resultados esperados. Um algoritmo de treinamento é empregado para ajustar o modelo, aproximando os resultados produzidos dos resultados esperados. Os resultados produzidos pelo modelo são chamados usualmente de *predições*, enquanto que os resultados esperados são chamados de *ground-truth*[4]. O aprendizado é dito *supervisionado* pela disponibilidade do *ground-truth*, utilizado no ajuste do modelo.

Os modelos apresentam certa estrutura que, na maioria dos sistemas de aprendizado, não se altera nem no treinamento, nem na implantação. No caso de redes neurais, a estrutura é chamada de *arquitetura* e define como vários módulos de processamento transmitem dados de uns para os outros. Esses módulos possuem *parâmetros* que são ajustados durante o processo de treinamento, buscando aproximar as predições do *ground-truth*. O algoritmo empregado no treinamento de redes neurais é chamado de *backpropagation*: ele utiliza o erro entre a predição e o *ground-truth* para, de trás para frente, ir ajustando os parâmetros do modelo, desde os últimos módulos de processamento até os módulos iniciais. O modelo é então uma arquitetura combinada a um conjunto de parâmetros. Uma mesma arquitetura pode, com um certo conjunto de parâmetros, realizar predições da localização de maçãs em pomares e, com um conjunto diferente de parâmetros, produzir predições para a detecção de cachos de uva em imagens de videiras. Conjuntos diferentes de parâmetros podem ser obtidos pelo mesmo processo de treinamento, utilizando conjuntos de dados diferentes: uma base de dados para maçãs ou uma base de dados com cachos de uva (Figura 1).

[1] Keras, keras.io; PyTorch, pytorch.org; MXNet, mxnet.apache.org; OpenCV, opencv.org; OpenMMLab, openmmlab.com; e GluonCV, cv.gluon.ai.

[2] tutfrutdet, github.com/thsant/tutfrutdet.

[3] MMDetection, mmdetection.readthedocs.io.

[4] Embora possa ser traduzido por verdade de campo, o termo ground-truth é mais utilizado na comunidade técnica.

Figura 1. Duas variações do problema de detecção de frutos (aqui, cachos de uva): (i) detecção de frutos (localização dos frutos por caixas delimitadoras retangulares); e (ii) segmentação de instâncias, a variação mais desafiadora, que consiste na detecção e na atribuição dos píxeis pertencentes a cada fruto. Cada cor representa um objeto/instância diferente. Figura adaptada de Santos et al. (2020).

Especialistas em redes neurais vêm propondo diferentes arquiteturas ao longo dos anos, buscando resolver os mais variados problemas em inteligência artificial no geral e em visão computacional em particular. Arquiteturas que têm maior repercussão acabam sendo incorporadas a bibliotecas de software, como a MMDetection. Inovações no processo de treinamento (variações do algoritmo de *backpropagation*) também são incorporadas às bibliotecas. Dessa forma, aqueles que desejam utilizar um detector de objetos em seus problemas específicos não precisam necessariamente desenvolver sua própria arquitetura de redes neurais, mas conduzir um processo de treinamento a partir de um conjunto de dados apropriadamente *anotado*, isto é, dados de entrada acompanhados do *ground-truth* que representa os resultados esperados.

No processo de treinamento, a base de dados, formada pelos dados de entrada (imagens no caso) e pelo *ground-truth* (a localização dos frutos), é fornecida ao processo de treinamento em *lotes* (*batches*). Isto se deve a limitações computacionais: na maior parte dos casos é inviável carregar a base de dados inteira na memória do sistema e obter todas as predições para os parâmetros correntes antes de ajustá-los. O treinamento é então realizado em lotes, com ajustes de parâmetros pelo algoritmo de *backpropagation* ao final do processamento de cada lote. Quando o processo de treinamento finaliza todos os lotes, dizemos que ele completou uma época. Para obter boas predições, o processo de treinamento pode levar dezenas ou centenas de épocas. O progresso do processo é medido pela diminuição do erro, isto é, a diferença acumulada entre as predições e o *ground-truth*. Quanto menor o erro, mais próximas as predições do modelo estão dos resultados esperados.

Arquiteturas modernas de redes neurais apresentam *milhões* de parâmetros. Ajustá-los é um processo computacionalmente custoso, especialmente para processadores de propósito geral como CPUs. Hardware especialmente construído para computação distribuída, como GPUs, conseguem realizar os processos de predição e *backpropagation* de forma mais eficiente. Assim, o treinamento de modelos com redes neurais profundas é realizado utilizando-se esse tipo de processador. Quando um modelo já treinado é utilizado para realizar predições, dizemos que está sendo utilizado para *inferência*. Embora a inferência também seja realizada mais rapidamente em GPUs, dependendo da aplicação o tempo gasto por CPUs em inferência pode ser aceitável, o que aumenta o número de dispositivos nos quais o modelo pode ser disponibilizado. O conteúdo deste tutorial pode ser replicado tanto em CPUs quanto em GPUs, mas recomenda-se fortemente o uso de GPUs, quando possível, no processo de treinamento.

Este tutorial apresentará um fluxo de trabalho completo para detecção de frutos em imagens, incluindo:

criação de uma base de dados anotada – como produzir bases de dados de imagens e anotações para detecção de frutos, que ferramentas podem ser utilizadas na anotação e como essas anotações são armazenadas em arquivos;

carga de lotes – como realizar a leitura da base de dados de modo a produzir os lotes necessários ao treinamento da rede e como utilizar ferramentas de *aumentação* de dados de modo melhorar a generalização da detecção e promover a invariância a condições vistas em situações de campo;

configuração de uma arquitetura de rede neural – como utilizar arquiteturas disponíveis, adaptando-as ao problema de detecção de frutos;

treinamento – como efetuar o treinamento e monitorar seu progresso, além de selecionar e armazenar os melhores parâmetros encontrados para o modelo; e

avaliação e inferência – como avaliar o desempenho do modelo e utilizá-lo na inferência, ou seja, na detecção de frutos em outras imagens.

3 BASE DE DADOS E ANOTAÇÃO

As imagens devem ser obtidas nas condições mais próximas possíveis da condição final de operação. Isso inclui: (i) as cultivares de interesse, por exemplo, *Fuji* e *Gala* na pomicultura ou *Syrah* e *Cabernet Sauvignon* na viticultura; (ii) as condições de iluminação na operação: nublado e ensolarado, manhã e tarde, noite com iluminação artificial; (iii) as características dos sistemas de imageamento: modelos de câmeras, resolução de imagem e a posição da câmera em relação aos frutos. A ideia central é que a base de dados reflita a *diversidade* encontrada no ambiente de operação: bases que apresentam uma *distribuição* similar a da condição de operação produzem modelos superiores.

Quantas imagens são necessárias? Vários frutos podem ser observados em uma única imagem, de forma que cada imagem pode apresentar dezenas de exemplos dos padrões visuais assumidos pelos frutos (Figura 1). Dessa forma, poucas centenas de imagens podem ser suficientes para produzir detectores acurados. Além disso, as técnicas de aumentação que veremos a seguir permitem que mais variabilidade seja disponibilizada ao treinamento para a mesma base de dados, aumentando a diversidade de exemplos fornecidos à rede durante o treinamento. Contudo, o tamanho da base depende da necessidade de exemplificar as diversas situações às quais o sistema será exposto: quanto maior a variabilidade de condições, maior a necessidade de imagens. Como referência, a WGISD apresenta 4.431 cachos de uva em 300 imagens, enquanto que a MinneApple apresenta mais de 40 mil frutos em mil imagens.

Selecionadas as imagens, procede-se para o processo de anotação. Na detecção de objetos, há tradicionalmente dois tipos de anotações empregadas. As *caixas delimitadoras* (*bounding boxes*) são anotações retangulares que se ajustam a cada fruto, como visto na Figura 1 (centro). Para cada fruto, basta o registro das extremidades da caixa retangular, geralmente as coordenadas do canto superior esquerdo e do canto inferior direito do retângulo que melhor se ajusta ao fruto (Figura 2). Esse tipo de anotação, com o uso de ferramenta adequada, é realizado de forma rápida: dezenas de frutos podem ser anotados em poucos minutos. A principal desvantagem é que na área retangular não há o registro de quais píxeis fazem parte do fruto, informação que pode ser de interesse em determinadas aplicações. Já as *máscaras* (Figura 1, à direita), identificam quais píxeis da imagem pertencem aos frutos, mas têm a desvantagem de serem mais demoradas e trabalhosas de serem anotadas. Diversas ferramentas para anotação estão disponíveis, tanto na forma de aplicações *desktop* quanto aplicações *web*. Diversas são disponibilizadas com código aberto, como VIA (VGG Image Anotator), CVAT (Figura 2), Diffgram e LabelImg.[5] Algumas ferramentas (como CVAT e Diffgram, por exemplo) permitem que a tarefa de anotação seja compartilhada por um time de múltiplos anotadores. Há também ferramentas de anotação oferecidas comercialmente, geralmente na forma de aplicações *web*.

Essas ferramentas são capazes de exportar as anotações para diversos formatos de arquivo. No caso da detecção de objetos, um formato[6] popular é o adotado na Common Objects in Context (COCO) (Lin et al., 2014), uma base de dados criada para pesquisa e desenvolvimento em detecção de objetos, composta por mais de 200.000 imagens com objetos em 80 categorias diferentes. Nesse formato, as anotações são representadas na forma de arquivos do tipo JSON, um formato de arquivo textual estruturado largamente utilizado em diversas aplicações. Dada a importância da MSCOCO, a maioria das bibliotecas de software voltadas a redes neurais fornecem utilitários para leitura e carga de dados nesse formato. Outra vantagem é que diversas ferramentas de anotação são capazes de importar dados desse formato, o que pode ser especialmente útil na hora de editar e revisar bases previamente anotadas.

Exemplo: base de dados anotada. No repositório de código, veja wgisd/README.md e minneapple/README.md.

4 AUMENTAÇÕES

A *aumentação* é o processo de produzir mais dados de treinamento artificialmente, aplicando transformações aleatorizadas em imagens existentes na base de treinamento, aproveitando suas anotações. Tais transformações são operações de processamento de imagens que produzem imagens de aparência similar e possíves alterações do objeto em estudo. É importante que tais transformações produzam imagens que sejam plausíveis, similares àquelas que poderiam ser encontradas na operação normal: por exemplo, imagens levemente borradas são similares às produzidas por lentes ligeiramente fora do foco, enquanto imagens clareadas remetem a imagens obtidas em am-

[5] VIA, www.robots.ox.ac.uk/~vgg/software/via; CVAT, www.cvat.ai; Diffgram, diffgram.com; e LabelImg, github.com/heartexlabs/labelImg.

[6] Common Objects in Context, formato cocodataset.org/#format-results.

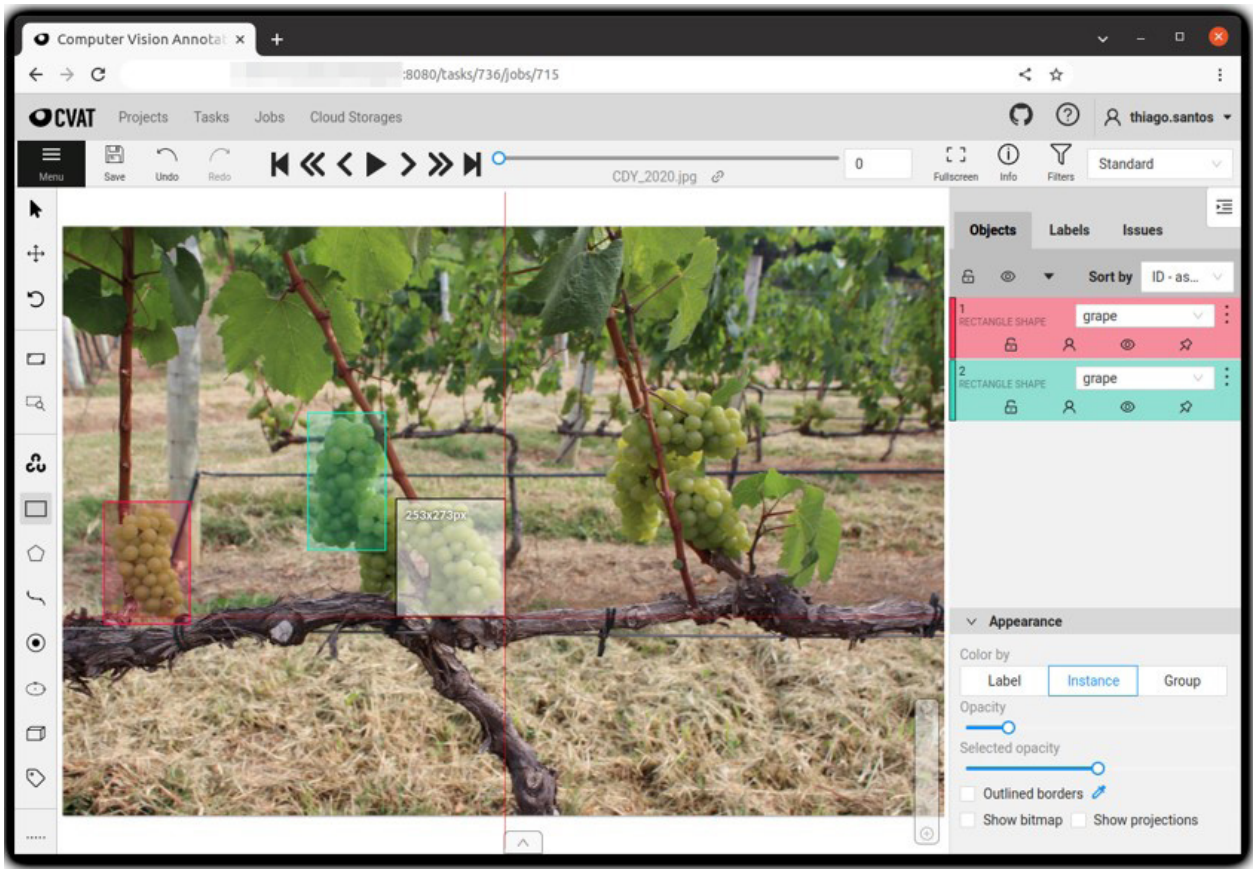

Figura 2. Exemplo de ferramenta de anotação: CVAT. Cachos de uva estão sendo anotados com caixas delimitadoras (*bounding boxes*).

biente mais iluminado ou com maior exposição da câmera. No caso de frutos, transformações podem ser aplicadas para remeter a diferentes condições de iluminação, como diferenças de horário, tempo nublado ou efeitos causados por reflexos solares e umidade. As imagens também podem ser espelhadas horizontalmente. O princípio a ser seguido é o da *invariância a transformações*: se o modelo pode encontrar um fruto na imagem original, ele também deve ser capaz de encontrar o mesmo fruto na imagem transformada. Se maçãs são detectadas em uma imagem, o modelo deve ser capaz de fazer o mesmo em uma versão espelhada da imagem, por exemplo. Existem bibliotecas de software especialmente dedicadas à aumentação, como Albumentations[7] e imgaug[8] – galerias de imagens exibindo exemplos das várias transformações disponíveis encontram-se disponíveis na documentação dessas bibliotecas. Todas as transformações disponíveis na Albumentations podem ser diretamente utilizadas nos pipelines de detecção de objetos da MMDetection.

Exemplo: carga de lotes. No repositório de código, veja wgisd/wgisd.py.

5 ARQUITETURAS

As arquiteturas para detecção de objetos costumam ter três conjuntos de módulos componentes, chamados de *backbone*, *neck* e *head* (espinha dorsal, pescoço e cabeça, em tradução livre). *Backbones*, baseados em redes convolutivas, são essencialmente bancos de filtros que aplicam transformações[9] às imagens de entrada. O primeiro módulo aplica transformações diretamente nos píxeis da imagem, enquanto os módulos seguintes aplicam novos bancos de filtros aos resultados produzidos pelos módulos anteriores. A intuição é que essas transformações combinem informações e ressaltem as características mais importantes para o problema de detecção, por exemplo, salientando elementos que compõem os objetos a serem detectados. As redes convolutivas realizam a composição das características, de forma que os módulos finais estão

[7] Albumentations, albumentations.ai.
[8] Imgaug, imgaug.readthedocs.io.
[9] Não confundir com as transformações realizadas pelo processo de aumentação durante o treinamento das redes.

combinando características mais simples aprendidas nos módulos anteriores. Nos exemplos no repositório, há dois *backbones* em uso: ResNet-50 (He et al., 2016) e EfficientNet-B3 (Tan; Le, 2019).

Enquanto o *backbone* aprende transformações da imagem capazes de construir uma *representação* dela especialmente adaptada para a tarefa em questão (daí o termo *aprendizado de representações*), cabe à *head* receber essa representação da imagem e realizar a tarefa pretendida. No caso da detecção de objetos, uma tarefa é determinar, a partir da representação, quais as caixas delimitadoras para cada fruto. Porém, redes podem ter múltiplas tarefas a resolver, o que significa que podem ser *multi-head*, ou seja, incluir várias *heads* que podem enviar informação umas para as outras. No caso da segmentação de instâncias, outra tarefa é a determinação de uma máscara que identifica quais píxeis dentro da caixa delimitadora pertencem ao fruto, logo encontraremos uma *mask head* na arquitetura. Cada *head* produz uma saída (*output*) que é comparada a anotação por uma função de erro (*loss function*), que avalia numericamente a discrepância entre o predito e o *ground-truth*. Os erros das diversas *heads* são integrados em um valor unificado e o processo de *backpropagation* irá ajustar os parâmetros do modelo de modo a diminuir esse erro total[10].

Ligando o *backbone* e o *head*, é comum encontrarmos um *bottleneck*, ou simplesmente *neck*, um conjunto de módulos dedicados a ajustar a saída do *backbone* (uma arquitetura de propósito geral) à entrada do *head*, geralmente realizando operações de redução de dimensionalidade (como a redução do número de canais). Quando estamos adaptando redes a um problema particular de detecção de objetos (por exemplo, detecção de maçãs), geralmente basta configurar a *head* para que ela se adapte ao problema em questão. No caso da Mask RCNN e da RetinaNet (Lin et al., 2017), empregadas nesse tutorial, a única adaptação necessária é informar o número de classes. Muitas implementações partem da MSCOCO e por isso vêm configuradas para as 80 classes existentes naquela base de dados. No caso dos arquivos de configuração da MMDetect, a alteração é realizada de modo simples, pela edição do número de classes no arquivo de configuração do modelo. Todos as demais configurações do modelo são *herdadas* das *configurações base* da MMDetect.

[10] O erro deve ser uma função diferenciável dos parâmetros, ou seja, deve ser possível determinar um gradiente, a direção na qual os parâmetros devem ser modificados de forma a diminuir o erro. O algoritmo de backpropagation é baseado em um algoritmo de descida de gradiente.

Exemplo: configuração de uma arquitetura de rede neural. No repositório de código, o arquivo wgisd/mask_rcnn_r50_fpn_1x_wgisd.py exibe a configuração de uma rede Mask R-CNN que se utiliza da ResNet 50 como *backbone* e de uma rede FPN como *neck*. Todas a configurações que não foram diretamente especificadas são *herdadas* das configurações base encontradas em mask_rcnn/mask_rcnn_r50_fpn_1x_coco.py, oriunda da implementação original da MMDetection. De forma similar, em wgisd/retinanet_effb3_fpn_crop896_wgisd.py encontramos a configuração para uma rede RetinaNet que utiliza a EfficientDet B3 como *backbone* e novamente a FPN como *neck*. Exemplos similares podem ser encontrados para a MinneApple (maçãs).

6 TREINAMENTO

No treinamento, *ciclos* (*steps*) constituídos pelas etapas de predição, cômputo do erro e ajuste de parâmetros são realizados sucessivamente. Uma amostra de imagens (e suas anotações) é selecionada e aumentações são realizadas, fornecendo variações diferentes da mesma imagem a cada vez que ela é incluída na amostra. Tal amostra forma um lote. O modelo (a arquitetura e os valores atuais dos parâmetros) recebe o lote e realiza predições, no nosso caso, a localização dos frutos. As predições são comparadas ao *ground-truth* e o erro é utilizado no algoritmo de *backpropagation* para ajustar os parâmetros, o que conclui o ciclo.

O treinamento é repetido até atingirmos *convergência*, isto é, um estado no qual o modelo atinge um certo nível de desempenho limite, no qual novos ciclos de treinamento com a mesma base de dados não produzem mais melhorias nas predições. A Figura 3 exibe os valores do erro no conjunto de treinamento e da precisão média no conjunto de validação para uma rotina de treinamento de um modelo Mask R-CNN para a WGISD. Observamos que antes mesmo de atingirmos 10.000 lotes, o erro no conjunto de treinamento decresce lentamente, enquanto a precisão para as imagens de validação atinge um *platô*. Examinando o platô mais atentamente, ele apresenta uma inclinação que sugere que a precisão está *diminuindo* com o passar dos ciclos de treinamento. Isto sinaliza um fenômeno de *superajuste* (*overfitting*), no qual o modelo passa a se ajustar demasiadamente aos dados de treinamento ao custo de seu poder de *generalização*, isto é, sua capacidade de apresentar boas predições em imagens fora do conjunto de treinamento. Informalmente, podemos fazer uma analogia com um estudante que decora questões e apresenta bons resultados em provas, mas exibe um mal desempenho quando questões similares, mas diferentes das decoradas, são apresentadas. Logo, nosso objetivo no treinamento é atingir um ponto de convergência, mas evitar o *overfitting*.

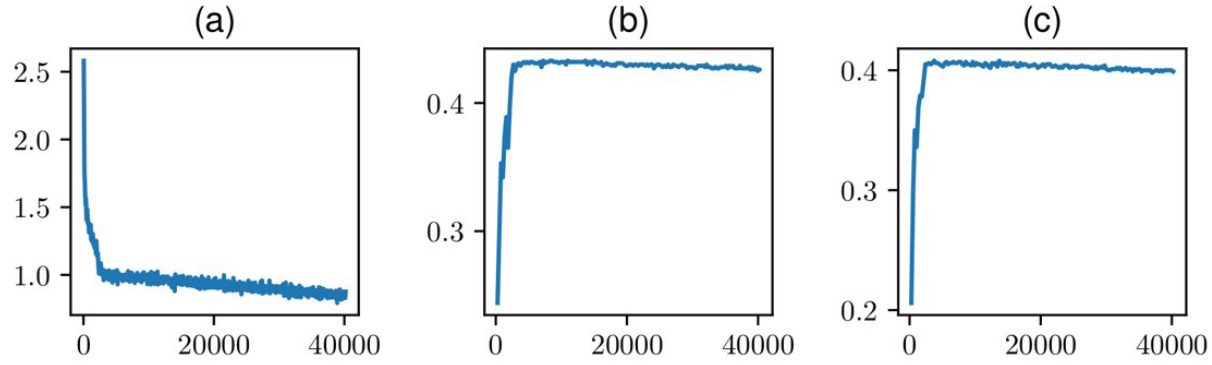

Figura 3. Métricas obtidas durante o treinamento. Em todos os três gráficos, o eixo das abscissas representa o número de lotes processados. (a) erro do treinamento (*training loss*); (b) mAP das caixas delimitadoras para o conjunto de validação; (c) mAP das máscaras para o conjunto de validação.

Gráficos como os vistos na Figura 3 são empregados durante o processo de treinamento para monitorarmos a convergência do modelo e possíveis problemas no processo. Ferramentas como TensorBoard[11] são utilizadas para visualização desses gráficos e outras informações produzidas pelo processo de treinamento.

Exemplo: treinamento. No repositório de código, veja README.md para orientações sobre o processo de treinamento, incluindo seu monitoramento com TensorBoard.

7 PRÉ-TREINAMENTO E TRANSFERÊNCIA DE APRENDIZADO

Como o modelo é capaz de realizar predições para o *primeiro* ciclo de treinamento, em que nenhum ajuste dos parâmetros do modelo foi realizado? Os parâmetros são inicializados aleatoriamente, o que irá inevitavelmente produzir níveis altos de erro nos primeiros ciclos. Conforme os parâmetros são ajustados, o erro diminui e podemos observar frequentemente quedas vertiginosas do erro já nos primeiros ciclos do treinamento.

Porém, há uma maneira mais eficiente de inicializarmos os parâmetros de uma rede, sobretudo os parâmetros do *backbone*, responsável pela extração de características. A *transferência de aprendizado* inicializa o modelo com parâmetros aprendidos em grandes bases de dados de propósito geral (Lin et al., 2014) e disponibilizados publicamente. Essa técnica de inicialização diminui o tempo de treinamento e possibilita que bons níveis de acurácia na detecção de objetos sejam atingidos com o uso de bases de dados de tamanho menor. Os primeiros módulos de um *backbone* aprendem características básicas, comuns à maioria das tarefas visão computacional, como bordas, linhas e formas. Isto permite que um *backbone* treinado em uma base como a MSCOCO produza representações úteis a tarefas de detecção particulares, como detecção de uvas ou maçãs, por exemplo. Nos exemplos disponíveis no repositório, utilizamos *backbones* pré-treinados, carregados automaticamente pela MMDetection.

8 AVALIAÇÃO E INFERÊNCIA

O conceito central na avaliação é que um objeto é considerado detectado se uma porcentagem considerável de sua área foi encontrada pelo detector, conceito expresso por uma métrica chamada de *interseção sobre união* (*intersection over union*, IoU). Outro ponto a ser observado é que a maioria dos detectores de objeto fornece, associado à caixa delimitadora de cada predição, um valor no intervalo de 0 a 1 que indica a *confiança* (*score*) naquela detecção: valores mais próximos de 1 indicam maior confiança de que aquela predição corresponde a um objeto.

Especificados valores mínimos de confiança e IoU, é possível estabelecer os *falsos positivos* (FPs) e os *falsos negativos* (FNs) de uma predição, respectivamente as predições de frutos que não existem no *ground-truth* (FPs) e frutos do *ground-truth* não encontrados pelo detector (FNs). As métricas de avaliação de desempenho de um detector são definidas sobre os valores de FP e FN. Como é possível variar a confiança e o IoU mínimos ou, ainda, estabelecer intervalos de interesse para esses dois valores, a avaliação do desempenho de detectores de objetos acaba apresentando diversos detalhes técnicos que, por limitação de espaço, não serão apresentados neste texto, mas encontram-se detalhados no repositório.

[11] TensorBoard, www.tensorflow.org/tensorboard.

Figura 4. Detecção de maçãs por um modelo Mask R-CNN. (a) imagem de entrada; (b) predições obtidas pelo modelo treinado.

Finalmente, os parâmetros que apresentaram melhor avaliação podem ser armazenados, junto com a arquitetura, formando um modelo para detecção que pode ser utilizado em aplicações que realizarão *inferência*: a detecção de frutos em novas imagens. A Figura 4 exibe um exemplo de inferência para segmentação de instâncias de maçã com um modelo Mask R-CNN.

Exemplo: avaliação e inferência. No repositório de código, veja README.md para orientações sobre as ferramentas de avaliação e inferência disponíveis na MMDetection.

9 CONSIDERAÇÕES FINAIS

Redes neurais artificiais para detecção de objetos em imagens, particularmente frutos, são capazes de atingir elevados níveis de acurácia, mesmo em condições de campo, onde há enorme diversidade de iluminação, condições de imageamento e variabilidade de fenótipos. O emprego dessas redes não está mais limitado a especialistas em visão computacional: é agora técnica reconhecida na agricultura digital para caracterização de talhões e sua variabilidade espacial. Graças a bibliotecas modernas para visão computacional como MMDetection, técnicos que tenham familiaridade com programação podem produzir modelos baseados em redes neurais para detecção de frutos ou outras estruturas de interesse como flores, folhas e espigas. Na literatura técnico-científica, detectores de frutos baseados em arquiteturas neurais como Faster R-CNN, Mask R-CNN e YOLO, todas disponíveis na MMDetection (e em outras bibliotecas), têm sido empregados com bons níveis de acurácia. O repositório de software que acompanha este tutorial apresenta exemplos práticos na viticultura e na pomicultura, que podem ser adaptados a outras culturas, ou, ainda, a casos particulares delas.

AGRADECIMENTOS

O autor agradece os auxílios recebidos da Embrapa, pelo projeto 01.14.09.001.05.04, e da Fundação de Amparo à Pesquisa do Estado de São Paulo (Fapesp), em conjunto com a IBM, pelo auxílio 2017/19282-7.

REFERÊNCIAS

BENGIO, Y.; GOODFELLOW, I.; COURVILLE, A. **Deep learning**. Cambridge: MIT press, 2017. Disponível em: hhttps://www.deeplearningbook.orgi. Acesso em: 22 out. 2023.

CHOLLET, F. **Deep learning with python, second edition**. Shelter Island: Manning, 2021. Disponível em: hhttps://www.manning.com/books/deep-learning-with-python-second-edition. Acesso em: 22 out. 2023.

HE, K.; GKIOXARI, G.; DOLLAR, P.; GIRSHICK, R. Mask R-CNN. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION (ICCV), 2017, Venice. **Procceedings [...]**. Venice: IEEE, 2017. DOI: https://doi.org/10.1109/ICCV.2017.322.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR). 2016, Las Vegas. **Procceedings [...]**. Las Vegas: IEEE, 2016. p. 770-778. DOI: https://doi.org/10.1109/CVPR.2016.90.

HÄNI, N.; ROY, P.; ISLER, V. MinneApple: a benchmark dataset for apple detection and segmentation. **IEEE Robotics and Automation Letters**, v. 5, n. 2, p. 852-858, 2020. DOI: https://doi.org/10.1109/LRA.2020.2965061.

JIANG, Y.; LI, C. Convolutional neural networks for image-based high-throughput plant phenotyping: a review. **Plant Phenomics**, v. 2020, p. 4152816, 2020. DOI: https://doi.org/10.34133/2020/4152816.

LIN, T.-Y.; GOYAL, P.; GIRSHICK, R.; HE, K.; DOLLAR, P. Focal loss for dense object detection. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION (ICCV), 2017, Venice. **Procceedings [...]**. Venice: IEEE, 2017. DOI: https://doi.org/10.1109/TPAMI.2018.2858826.

LIN, T.-Y.; MAIRE, M.; BELONGIE, S.; BOURDEV, L.; GIRSHICK, R.; HAYS, J.; PERONA, P.; RAMANAN, D.; ZITNICK, C. L.; DOLLÁR, P. **Microsoft COCO: common objects in context.** arXiv, 2014. Disponível em: https://arxiv.org/abs/1405.0312i. Acesso em: 22 out. 2023.

MIRHAJI, H.; SOLEYMANI, M.; ASAKEREH, A. Fruit detection and load estimation of an orange orchard using the yolo models through simple approaches in different imaging and illumination conditions. **Computers and Electronics in Agriculture**, v. 191, p. 106533, 2021. DOI: https://doi.org/10.1016/j.compag.2021.106533.

NUSKE, S.; WILSHUSEN, K.; ACHAR, S.; YODER, L.; NARASIMHAN, S.; SINGH, S. Automated visual yield estimation in vineyards. **Journal of Field Robotics**, v. 31, n. 5, p. 837-860, 2014. DOI: https://doi.org/10.1002/rob.21541.

OLENSKYJ, A. G.; SAMS, B. S.; FEI, Z.; SINGH, V.; RAJA, P. V.; BORNHORST, G. M.; EARLES, J. M. End-to-end deep learning for directly estimating grape yield from ground-based imagery. **Computers and Electronics in Agriculture**, v. 198, p. 107081, 2022. ISSN 0168-1699. DOI: https://doi.org/10.1016/j.compag.2022.107081.

REDMON, J.; FARHADI, A. YOLO9000: Better, Faster, Stronger. In: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR). 2017, Venice. **Procceedings [...]**. Venice: IEEE, 2017. DOI: https://doi.org/10.1109/CVPR.2017.690.

REN, S.; HE, K.; GIRSHICK, R.; SUN, J. Faster R-CNN: towards real-time object detection with region proposal networks. In: ORTES, C.; LAWRENCE, N.; LEE, D.; SUGIYAMA, M.; GARNETT, R. (eds.). **Advances in neural information processing systems**. Curran Associates, Inc., 2015. Disponível em: https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf. Acesso em: 22 out. 2023.

SA, I.; GE, Z.; DAYOUB, F.; UPCROFT, B.; PEREZ, T.; MCCOOL, C. Deepfruits: a fruit detection system using deep neural networks. **Sensors**, v. 16, n. 8, p. 1222, 2016. DOI: https://doi.org/10.3390/s16081222.

SANTOS, T. T.; SOUZA, L. L.; SANTOS, A. A.; AVILA, S. Grape detection, segmentation, and tracking using deep neural networks and three-dimensional association. **Computers and Electronics in Agriculture**, v. 170, p. 105247, 2020. DOI: https://doi.org/10.1016/j.compag.2020.105247.

SILWAL, A.; DAVIDSON, J. R.; KARKEE, M.; MO, C.; ZHANG, Q.; LEWIS, K. Design, integration, and field evaluation of a robotic apple harvester. **Journal of Field Robotics**, v. 34, n. 6, p. 1140-1159, 2017. DOI: https://doi.org/10.1002/rob.21715.

STEIN, M.; BARGOTI, S.; UNDERWOOD, J. Image based mango fruit detection, localisation and yield estimation using multiple view geometry. **Sensors**, v. 16, n. 11, p. 1915, 2016. DOI: https://doi.org/10.3390/s16111915.

TAN, M.; LE, Q. EfficientNet: Rethinking model scaling for convolutional neural networks. In: 36TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 2019, Long Beach. **Proceedings [...]**. Long Beach: PMLR, 2019. v. 97, p. 6105-6114. Disponível em: https://proceedings.mlr.press/v97/tan19a.html. Acesso em: 22 out. 2023.

TIAN, Y.; YANG, G.; WANG, Z.; WANG, H.; LI, E.; LIANG, Z. Apple detection during different growth stages in orchards using the improved yolo-v3 model. **Computers and Electronics in Agriculture**, v. 157, p. 417-426, 2019. DOI: https://doi.org/10.1016/j.compag.2019.01.012.

WILLIAMS, H.; TING, C.; NEJATI, M.; JONES, M. H.; PENHALL, N.; LIM, J. Y.; SEABRIGHT, M.; BELL, J.; AHN, H. S.; SCARF, A.; MACDONALD, B. Improvements to and large-scale evaluation of a robotic kiwifruit harvester. **Journal of Field Robotics**, v. 37, n. 2, p. 187-201, 2020. DOI: https://doi.org/10.1002/rob.21890.