UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

GABRIEL EUGENIO BRITO

UM ESTUDO COMPARATIVO DE MODELOS DE SEGMENTAÇÃO DE OBJETOS

CURITIBA

2023

GABRIEL EUGENIO BRITO

UM ESTUDO COMPARATIVO DE MODELOS DE SEGMENTAÇÃO DE OBJETOS

A comparative study of object segmentation models

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Sistemas de Informação do Curso de Bacharelado em Sistemas de Informação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Rodrigo Minetto

Coorientadora: Profª. Leyza Baldo Dorini

CURITIBA

2023

GABRIEL EUGENIO BRITO

UM ESTUDO COMPARATIVO DE MODELOS DE SEGMENTAÇÃO DE OBJETOS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Sistemas de Informação do Curso de Bacharelado em Sistemas de Informação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 27/junho/2023

Rodrigo Minetto
Doutorado
Universidade Tecnológica Federal do Paraná (UTFPR)

Ricardo Dutra da Silva
Doutorado
Universidade Tecnológica Federal do Paraná (UTFPR)

Egon Sulivan Stefani
Graduação
Universidade Tecnológica Federal do Paraná (UTFPR)

CURITIBA

2023

RESUMO

Segmentação de objetos é a tarefa de identificar, com precisão de pixel, todos os objetos de uma imagem. Ela tem diversas aplicações no mundo real, como sistemas de visão para carros autônomos e sistemas de monitoramento. O objetivo deste trabalho é comparar três abordagens diferentes para solucionar esse problema – Mask R-CNN, Yolact e SOLO –, todas relativamente recentes e baseadas em *deep learning*, e apresentar alguns pontos que possam ser explorados em pesquisas futuras. Ao executar os modelos em 200 imagens do *dataset* OpenImages, mostramos que o SOLO alcança os melhores resultados em relação à qualidade, com 42.4 de precisão média (AP), aproximadamente 4 pontos acima do segundo colocado, e o Yolact, em relação à velocidade, com aproximadamente metade do tempo de inferência do segundo colocado. Também foi possível constatar algumas limitações na área, como a falta de dados anotados no *dataset* e casos que são ignorados durante a avaliação (mais especificamente, objetos de classes que não possuem anotações em uma determinada imagem são ignorados durante o cálculo do AP). Todo o código está disponível em https://github.com/GabrielEug2/tcc-instance-segm. Esperamos que estes pontos possam ser explorados mais a fundo em pesquisas futuras, para avançar a pesquisa na área ainda mais.

Palavras-chave: segmentação de instâncias; detecção de objetos; segmentação de objetos; visão computacional.

ABSTRACT

Object segmentation is the task of identifying, with pixel precision, all the objects on an image. It has several applications in the real world, such as vision systems for autonomous cars and monitoring systems. The goal of this paper is to compare three different approaches of solving that problem – Mask R-CNN, Yolact and SOLO –, all relatively recent and based on deep learning, and present some points that can be explored in future researches. By executing the models on 200 images of the OpenImages dataset, we showed that SOLO achieves the best results in terms of quality, with 42.4 of average precision (AP), about 4 points ahead of the runner-up, and Yolact, in terms of speed, with approximately half the inference time of the second place. It was also possible to observe some limitations in the area, such as the lack of annotated data on the dataset and cases that are ignored during the evaluation (more specifically, objects of classes that do not have annotations on a given image are ignored durante the AP calculations). All the code is available at https://github.com/GabrielEug2/tcc-instance-segm. Hopefully those points can be further explored in future researches, to advance the research on the area even more.

Keywords: instance segmentation; object detection; object segmentation; computer vision.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Visão Computacional é um ramo da Inteligência Artificial (IA) que busca dar aos computadores a habilidade de descrever e interpretar imagens. Humanos são capazes de realizar esta tarefa com facilidade, mas para um computador até mesmo uma tarefa simples, como contar quantas pessoas aparecem em uma foto, costumava ser um grande desafio (SZELISKI, 2010, p. 3-5). Diversos problemas são estudados em paralelo nessa área, como classificação de imagens, reconhecimento facial e detecção de objetos, e ideias propostas para um campo frequentemente resultam em avanços nos demais.

Um problema que ganhou atenção recentemente é a chamada segmentação de instâncias, cujo objetivo é detectar e segmentar com precisão todos os objetos em uma imagem (HE *et al.*, 2017, p. 1). Ter algoritmos capazes de realizar segmentação de instâncias em um nível próximo ao de humanos avançaria muito a forma como máquinas interagem com seu ambiente (PINTO, 2017, p. 2). Um carro autônomo, por exemplo, poderia verificar se está a uma distância segura de um pedestre ou outro veículo, e ajustar sua velocidade e direção de acordo. Um robô assistente poderia olhar ao redor e identificar se um objeto solicitado se encontra ao seu alcance. Geração de legendas para imagens, sistemas de monitoramento e destaque de contornos para deficientes visuais são mais algumas das possíveis aplicações para estes algoritmos (Liang *et al.*, 2018; LIU *et al.*, 2018; ROMERA-PAREDES; TORR, 2015). Entretanto, essa é uma tarefa extremamente difícil, pois é preciso lidar, entre outras coisas, com a diversidade nas formas dos objetos, fronteiras oclusas por outros elementos da imagem e variedade de textura e cor em um mesmo objeto (CHEN; LIU; YANG, 2015; Liang *et al.*, 2018).

Diversos trabalhos foram publicados ao longo dos anos, com diferentes arquiteturas e abordagens para o problema. Muitos deles, como o Mask R-CNN (HE *et al.*, 2017) e o SOLO (WANG *et al.*, 2019), que serão discutidos aqui, apresentam resultados promissores em *datasets* com dezenas de classes de objetos, como o *Common Objects in Context*, comumente identificado pela sigla COCO (LIN *et al.*, 2014). Esses modelos alcançam entre 30 e 40 pontos na métrica utilizada pelos pesquisadores, que vai de 0 até 100. Pode não parecer muito, mas analisando a saída dos modelos é possível perceber que eles são surpreendentemente bons. Eles são capazes de detectar objetos dos mais variados tipos, inclusive em imagens com dezenas de objetos. Contudo, mesmo que os resultados divulgados sejam bons, sempre há espaço para melhorias.

1.1 Objetivos

O objetivo geral deste trabalho é analisar o desempenho de três modelos de segmentação de instâncias – Mask R-CNN (HE *et al.*, 2017), Yolact (BOLYA *et al.*, 2019b) e SOLO (WANG *et al.*, 2019) –, e apresentar as limitações observadas, tanto nos modelos em si quanto nos demais fatores envolvidos (datasets, métricas).

Quanto aos objetivos específicos:

- Analisar o desempenho dos modelos com diversas imagens (objetos conhecidos, objetos pequenos, objetos desconhecidos) de uma base similar à que eles foram treinados.
- Criar hipóteses que justifiquem o que foi observado.
- Sugerir tópicos a serem explorados em pesquisas futuras, baseado no que foi observado.

1.2 Estrutura do trabalho

Este trabalho está estruturado da seguinte forma. O Capítulo 2 apresenta alguns conceitos básicos, necessários para um melhor entendimento desse trabalho. O Capítulo 3 contém uma visão geral de trabalhos relacionados ao tema. O Capítulo 4 apresenta a metodologia utilizada na pesquisa, detalhando o que foi feito em cada etapa. O Capítulo 5 traz uma breve explicação de como os modelos selecionados funcionam. Os resultados dos experimentos são discutidos no Capítulo 6. Por fim, no Capítulo 7, são apresentadas as conclusões do trabalho.

2 REFERENCIAL TEÓRICO

2.1 Imagens digitais

Uma imagem pode ser definida como uma função multidimensional $f(x,y)$, na qual x e y são coordenadas espaciais em um plano, e o valor de f em qualquer ponto representa a cor naquele lugar em específico. No caso de imagens "preto e branco", os valores dessa função representam a intensidade ou nível de cinza naquele ponto. Já para imagens coloridas, estes valores representam cores em um determinado espaço de cores (GONZALEZ; WOODS, 2006, p. 23, 423-424, 446-447).

Se os valores de x, y e $f(x,y)$ são todos discretos, é possível representar essa imagem digitalmente usando um vetor bidimensional de tamanho $M \times N$, conforme a Equação 1. Cada posição (x,y) nesta matriz é chamada de *pixel* (GONZALEZ; WOODS, 2006, p. 23-24, 77-78).

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(M-1,0) & f(M-1,1) & \dots & f(M-1,N-1) \end{bmatrix} \tag{1}$$

2.2 Aprendizado de Máquina

Aprendizado de máquina é um ramo da IA que busca produzir nos computadores a habilidade de aprender. Diversos algoritmos e modelos de aprendizagem de máquina foram propostos ao longo do tempo, como árvores de decisão e redes neurais (RUSSELL; NORVIG, 2016).

Existem diversos tipos de aprendizado, mas o foco de pesquisa neste trabalho é o chamado aprendizado supervisionado. De forma geral, ele funciona da seguinte forma:

1. Primeiro, é necessário ter um conjunto de dados rotulados, que são pares indicando a saída esperada para uma determinada entrada. A ideia é que o computador irá usar estes exemplos para criar regras mais genéricas, que possam ser usadas futuramente com outros dados.

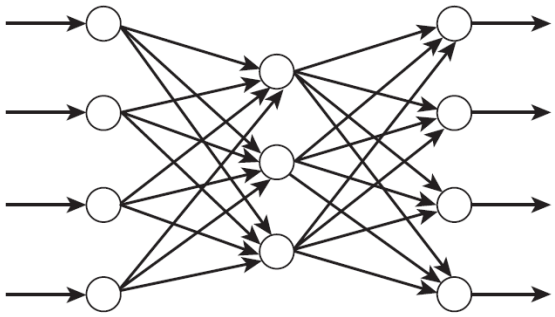
2. Os dados são divididos em dois grupos: dados de treinamento e dados de teste. Isso é importante, pois permite verificar se o modelo aprendeu, de fato, a relação entre entrada e saída, ou se apenas decorou a resposta certa para os dados de treinamento.

3. O modelo aprende a produzir a saída esperada para os dados de treinamento, sem nunca ver os dados de teste.

4. O modelo é testado com esse outro conjunto de dados, não vistos até então. Se os resultados forem satisfatórios, o modelo está pronto para ser usado. Caso contrário, são feitas modificações no processo de aprendizado ou no modelo em si, e ele passa por um novo teste.

Um modelo que se tornou muito popular nos últimos anos são as chamadas redes neurais. Inspiradas no sistema nervoso biológico, estas redes são compostas de diversas unidades chamadas neurônios. Cada neurônio recebe um determinado número de entradas, as processa e produz uma saída, que é propagada para outros neurônios através de conexões direcionais. Cada conexão possui um peso que indica a força e o sinal da conexão, e esses pesos são levados em consideração durante o processamento (RUSSELL; NORVIG, 2016, p. 727-728). Estas redes são normalmente organizadas em camadas, como na Figura 1.

Figura 1 – Uma rede neural simples com três camadas.



Fonte: (COPPIN, 2004, p. 301).

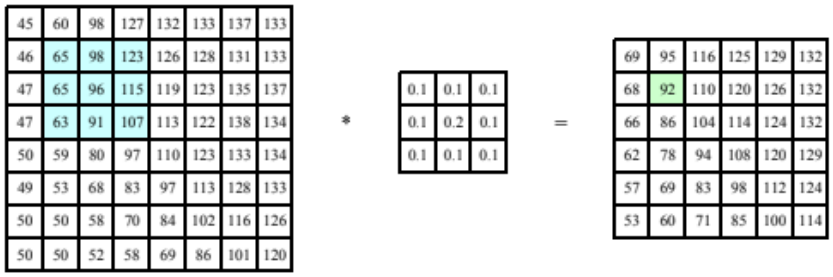
Existem vários tipos de redes neurais, mas a que nos interessa para este trabalho são as chamadas Redes Neurais Convolucionais (frequentemente chamada de CNNs, do inglês *Convolutional Neural Network*). Elas são um tipo especializado de redes neurais usadas para processar, entre outros tipos de dados, imagens (GOODFELLOW; BENGIO; COURVILLE, 2016, p. 326). Popularizadas por LeCun *et al.* (1998), estas redes foram inicialmente projetadas para serem usadas em classificação de imagens, mas com o passar do tempo diversas outras arquiteturas surgiram, o que possibilitou seu uso em outros problemas de visão computacional. Mas antes de falar mais a respeito das CNNs, primeiro é preciso entender o que é uma convolução.

No contexto de imagens, uma convolução pode ser entendida como o processo de movimentação de uma máscara sobre uma imagem e do cálculo, para cada pixel, da soma dos produtos entre a máscara e os *pixels* da imagem cobertos por ela (GONZALEZ; WOODS, 2006, p. 127-128, 168-172). Esse processo pode ser visto na Figura 2.

Formalmente, a convolução de um filtro w de tamanho $m \times n$ em uma imagem f produz uma segunda imagem, g, conforme a Equação 2,

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t) \tag{2}$$

Figura 2 – Exemplo de convolução com uma máscara de tamanho 3×3. A máscara no meio é aplicada sobre a imagem à esquerda, um pixel de cada vez, produzindo uma nova imagem, à direita. O pixel em verde é o resultado da aplicação da máscara sobre a região em azul.

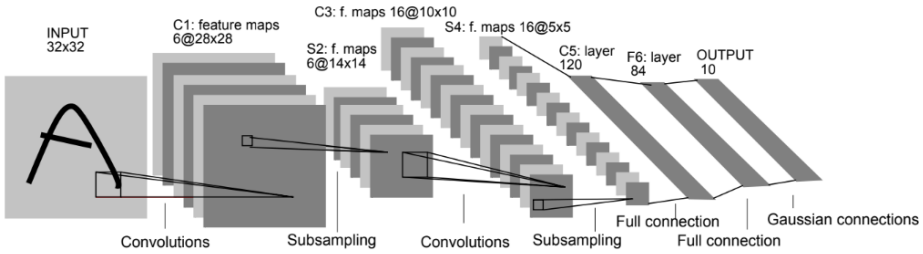


Fonte: (SZELISKI, 2010, p. 112).

tal que $a = (m-1)/2$ e $b = (n-1)/2$.

Uma CNN completa pode ser vista na Figura 3. As camadas de convolução realizam diversas convoluções em paralelo e produzem os chamados mapas de características (*feature maps*), que indicam onde uma determinada característica é mais presente na imagem. As primeiras camadas extraem características visuais simples, como bordas e cantos, e nas camadas seguintes elas são combinadas para detectar características de alto nível. Também existem as camadas de *pooling* ou *sub-sampling*, que reduzem o tamanho dos mapas de ativação, reduzindo assim a sensibilidade a deslocamentos e distorções (LECUN *et al.*, 1998, p. 6).

Figura 3 – Uma rede neural convolucional para reconhecimento de dígitos. Após as camadas de convolução e pooling são utilizadas camadas densamente conectadas para classificar, com base nas características extraídas, a imagem em uma das 10 classes possíveis.



Fonte: (LECUN *et al.*, 1998, p. 7).

2.3 Segmentação de Instâncias e Problemas Relacionados

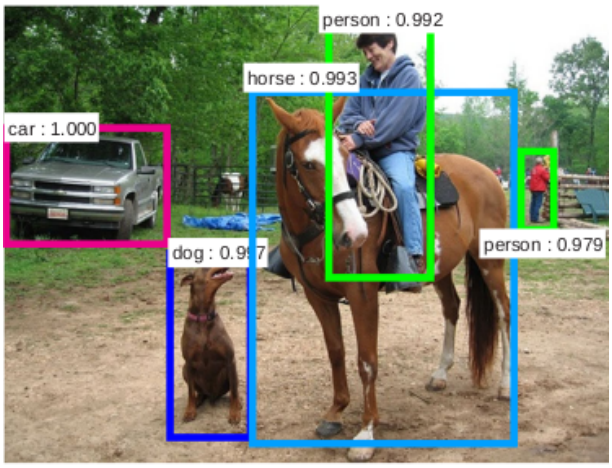
Entre os diversos problemas estudados na área de visão computacional, existe um, segmentação de instâncias, que é o foco deste trabalho. Mas para entendê-lo completamente, primeiro é necessário falar sobre alguns problemas similares que influenciaram no seu surgimento.

Classificação de imagens, às vezes chamada de *reconhecimento de objetos*, é a tarefa de determinar quais classes de objetos estão presentes em uma imagem (RUSSAKOVSKY *et al.*, 2014, p. 1). Aqui, não há a preocupação com a localização dos objetos, apenas com a

existência deles na imagem. Por exemplo, para a Figura 4, a saída esperada seria simplesmente "{pessoa, cavalo, carro, cão}".

Detecção de objetos se refere a obter não só a classe, mas também a localização dos objetos contidos em uma imagem (SZEGEDY; TOSHEV; ERHAN, 2013, p. 1). A localização é dada pelas coordenadas do menor retângulo que envolve o objeto, chamado de *bounding box*. A Figura 4 mostra um exemplo dessa definição.

Figura 4 – Exemplo de saída esperada de um algoritmo de detecção de objetos. Os valores indicam a confiança da classificação.



Fonte: (REN *et al.*, 2015, p. 3).

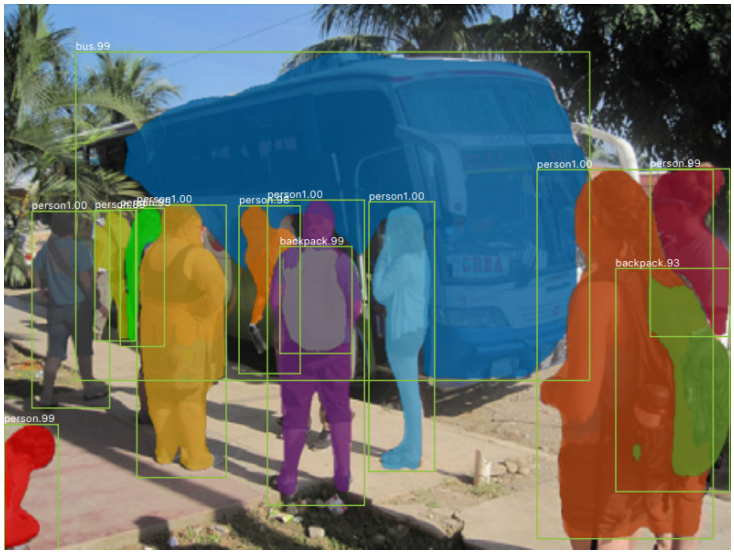
Segmentação semântica, por sua vez, é o nome dado à tarefa de classificação de imagens a nível de pixel (CHEN *et al.*, 2014, p. 1), ou seja, para cada pixel deve ser atribuída uma classe indicando do que ele é parte. Um exemplo de saída esperada pode ser visto na Figura 5.

Figura 5 – Exemplo de saída esperada de um algoritmo de segmentação semântica. Cada cor representa uma classe. O vermelho, por exemplo, indica a classe "pessoa".

Fonte: (XU *et al.*, 2018, p. 2).

Finalmente, **segmentação de instâncias**, ou *segmentação semântica ao nível de instância*, é a tarefa de detectar todos os objetos em uma imagem e, para cada instância, marcar os *pixels* que pertencem à ela (HARIHARAN *et al.*, 2014, p. 2). A Figura 6 mostra um exemplo de saída esperada. Comparada à detecção de objetos, aqui há uma preocupação maior com a localização deles, e diferente da segmentação semântica, é necessário distinguir entre as diferentes instâncias de objetos.

Figura 6 – Exemplo de saída esperada de um algoritmo de segmentação de instâncias. Note que pessoas diferentes aparecem em cores diferentes, ou seja, as instâncias devem ser diferenciadas.



Fonte: (HE *et al.*, 2017, p. 2).

3 TRABALHOS RELACIONADOS

Detecção de objetos, segmentação semântica e segmentação de instâncias são problemas intimamente relacionados. Ideias propostas para um, frequentemente são adaptadas para os demais, e portanto esta seção irá abranger trabalhos referentes à todos eles.

3.1 Detecção de Objetos

Grande parte dos trabalhos em detecção de objetos seguem a mesma estrutura: primeiro são geradas várias propostas de região para a imagem, que são regiões com altas chances de conterem um objeto, e depois estas regiões são classificadas. Os detectores de dois estágios, como são chamados, parecem ter origem no trabalho de Girshick *et al.* (2013), no qual os autores introduziram o R-CNN, um modelo que combina propostas de região e CNNs.

Diversas melhorias para esse modelo foram propostas com o passar do tempo. Girshick (2015) modificou a arquitetura de forma a computar as características da imagem inteira e depois apenas extrair as características referentes a cada região dos mapas de características, o que trouxe ganhos significativos na velocidade de treinamento e teste. Ren *et al.* (2015) propuseram uma nova forma de gerar as propostas de região, a *Region Proposal Network* (RPN), que tornou o modelo ainda mais rápido devido ao compartilhamento de características entre ela e a rede de detecção. Dai *et al.* (2016) introduziram os *position-sensitive score maps*, uma técnica que permitiu reduzir o número de computações por região, e melhorou ainda mais a velocidade do modelo. Lin *et al.* (2016) propuseram o uso de pirâmides de características, uma técnica já utilizada em detectores de objetos antes da popularização das redes neurais, que resultou em detecções mais precisas. Cai e Vasconcelos (2017) propuseram uma arquitetura com múltiplos estágios em que são usados valores crescentes de IoU (consulte a subseção 4.3.1 para o significado deste termo), forçando detecções cada vez mais precisas.

Quanto aos métodos de geração de propostas, existem alguns, como os de Carreira e Sminchisescu (2012), Uijlings *et al.* (2013) e Dai *et al.* (2016). A busca seletiva, de Uijlings *et al.* (2013), por exemplo, segmenta a imagem em diversas regiões pequenas e então combina regiões similares para formar mais propostas. Os outros funcionam de forma parecida, mas após sua introdução o RPN se tornou o método mais comum por se adequar bem à estrutura dos modelos.

Existem também modelos que dispensam a etapa de geração de propostas, chamados de detectores de um estágio. Uma estratégia comum nesse tipo de detector é usar um conjunto predefinido de *bounding boxes*, chamadas de *anchors*, e treinar o modelo para ajustá-las aos objetos através de regressão. O YOLO (REDMON *et al.*, 2015), o SSD (LIU *et al.*, 2015) e o RetinaNet (LIN *et al.*, 2017), por exemplo, funcionam assim. Embora estes modelos sejam mais rápidos do que os baseados em geração de propostas, seus resultados são, no geral, inferiores. Outra estratégia é resolver o problema de forma similar a segmentação semântica, realizando

predições por pixel, como no trabalho de Tian *et al.* (2019). Dessa forma, não é necessário computar propostas de região nem definir os parâmetros das *anchors*.

3.2 Segmentação Semântica

Abordagens recentes de segmentação semântica derivam do trabalho de Long, Shelhamer e Darrell (2014), no qual os autores propuseram uma rede neural totalmente convolucional, ou FCN (do inglês *Fully Convolutional Network*). Estas redes são capazes de realizar predições a nível de pixel para imagens de tamanhos arbitrários, devido à ausência de camadas densamente conectadas. As primeiras camadas extraem características, enquanto as últimas são responsáveis pelo *upsampling* para que a saída tenha o formato esperado. Ronneberger, Fischer e Brox (2015) expandiram esta arquitetura em um trabalho envolvendo imagens biomédicas. Sua principal contribuição foi adicionar mais camadas de *upsampling*, tornando a rede simétrica.

Chen *et al.* (2016) propuseram soluções para alguns problemas referentes ao uso de CNNs em segmentação semântica, como a baixa resolução das características decorrente das operações de *pooling*, que dificulta a realização de segmentações precisas. Entre outras contribuições, os autores mostraram que é possível gerar mapas de ativação com maior resolução utilizando convoluções *atrous*, um tipo de convolução no qual os filtros são ampliados (*upsampled*).

3.3 Segmentação de Instâncias

Quase todos os trabalhos em segmentação de instâncias são baseados em trabalhos de detecção de objetos. Em um dos primeiros trabalhos da área, Hariharan *et al.* (2014) se inspiraram no R-CNN para desenvolver uma abordagem parecida para segmentação de instâncias. Assim como aconteceu em detecção, a estratégia de usar propostas de região obteve bons resultados e fez com que essa abordagem se tornasse extremamente popular nos anos seguintes.

Dai, He e Sun (2015), por exemplo, propuseram um modelo em cascata composto de três redes, a primeira responsável por produzir propostas de região, a segunda por segmentar os objetos presentes nestas regiões, e a última por classificar cada instância. Li *et al.* (2016) adaptaram o conceito de *position sensitive score maps* usado em trabalhos de detecção para a tarefa de segmentação. He *et al.* (2017) modificaram um detector de objetos, incluindo um ramo responsável por produzir uma máscara de segmentação. Este modelo, denominado Mask R-CNN, é provavelmente o mais bem estabelecido atualmente, já que frequentemente é usado para comprovar resultados de novos trabalhos. Liu *et al.* (2018) fizeram algumas melhorias na arquitetura do Mask R-CNN, de forma a garantir um fluxo melhor de informação na rede. Chen

et al. (2019) exploraram diversas arquiteturas combinando o Mask R-CNN e o Cascade R-CNN, e o resultado foi um modelo que progressivamente refina as máscaras de segmentação.

Recentemente houveram trabalhos apresentando soluções mais rápidas, capazes de realizar segmentação “em tempo real” (a 30 quadros por segundo). Estes modelos utilizam detectores de um estágio, evitando o custo computacional referente à geração de propostas de região. Bolya *et al.* (2019b), por exemplo, desenvolveram um modelo que computa diversos protótipos de máscaras para uma imagem, e então os combina com as saídas produzidas por um detector para criar a máscara de cada instância. Mais tarde, em Bolya *et al.* (2019a), os autores propuseram algumas modificações no modelo com o intuito de melhorar o desempenho sem comprometer a velocidade. Lee e Park (2019), por sua vez, propuseram um modelo no qual as saídas produzidas por um detector são passadas individualmente por um módulo de atenção espacial, responsável por destacar os *pixels* mais informativos de uma região da imagem, e então usadas para segmentar os objetos detectados.

Outro trabalho interessante é o de Wang *et al.* (2019), no qual os autores abordam a segmentação de instâncias de uma perspectiva diferente. Da mesma forma que o detector YOLO, a imagem é divida em células, e cada célula é responsável por detectar, ou nesse caso, segmentar, os objetos cujo centro pertencem a ela. Em um trabalho seguinte, Wang *et al.* (2020) mostraram que essa ideia também pode alcançar bons resultados quando a preocupação é com a velocidade.

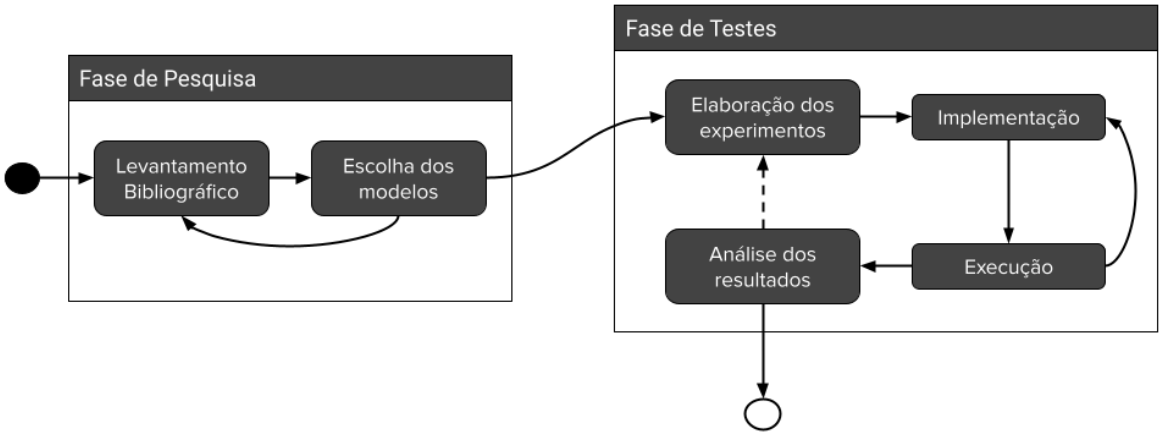
Algumas outras abordagens também foram exploradas, mas não são citadas com muita frequência. Park e Berg (2015) e Romera-Paredes e Torr (2015), por exemplo, utilizam redes recorrentes para segmentar uma instância por vez. Pinto (2017) e Arnab e Torr (2017) tentam combinar as saídas de modelos de detecção de objetos e segmentação semântica. Brabandere, Neven e Gool (2017) e Liang *et al.* (2018), por sua vez, utilizam técnicas de *clustering* para agrupar pixels pertencentes a uma mesma instância.

4 METODOLOGIA

4.1 Visão geral

A Figura 7 mostra as etapas seguidas no desenvolvimento deste trabalho.

Figura 7 – Fluxograma representando as etapas seguidas no trabalho.



Fonte: Autoria própria.

Primeiro foi feito o **levantamento bibliográfico**, ou seja, a busca por publicações e outros materiais relacionados ao tema em questão. Inicialmente foram utilizadas palavras-chave como “visão computacional” e “detecção de objetos”, para afunilar a área que seria estudada. Depois, conforme novos termos eram introduzidos, como “segmentação semântica” e “segmentação de instâncias”, a busca passou a incluí-los também. No fim, os termos mais relevantes foram “segmentação de instâncias” e os termos mencionados nos trabalhos da área, como “R-CNN” e “FCN / Fully Convolutional Network”. A maioria dos trabalhos podem ser encontrados nas bases *arXiv* e *IEEE*, com algumas exceções encontradas através do motor de busca Google. No geral, todos os textos podem ser encontrados seguindo essa linha de pesquisa.

Logo no início dessa etapa, percebeu-se que não seria possível propôr um modelo no nível dos atuais, devido à alta complexidade do problema e às limitações presentes, tanto em tempo quanto em recursos computacionais. Ao invés disso, optou-se por analisar alguns modelos existentes, e levantar possíveis limitações deles. Os modelos escolhidos foram: Mask R-CNN, pelo grande número de citações e resultados reportados; YOLACT, pelo grande número de citações e preocupação com a velocidade; e SOLO, por ser uma ideia diferente e não muito explorada, sem *bounding boxes* como as anteriores. Mais detalhes sobre esses modelos podem ser encontrados no Capítulo 5.

Em seguida ocorreu a etapa de **elaboração dos experimentos**, na qual foi definido como os modelos seriam testados e avaliados. Os detalhes desses experimentos podem ser vistos nas próximas seções.

Implementação é a escrita de código para possibilitar os experimentos, realizados na etapa seguinte, de **execução**. Como a maioria dos modelos publicados faz uso da linguagem Python, optou-se por utilizar esta mesma linguagem para os testes e experimentos. Inicialmente considerou-se utilizar o ambiente Google Colab, por conta da GPU que o serviço oferece, mas no fim optou-se por utilizar o computador pessoal do autor. Embora tenha especificações inferiores às da máquina oferecida pelo Colab, a execução de testes se provou mais fácil em uma máquina local. Outro ponto que influenciou nessa decisão foi a curiosidade em ver como os modelos em questão rodam em um computador sem placas gráficas de alto desempenho, ou quão acessíveis eles são para um público em geral. Todo o código está disponível em https://github.com/GabrielEug2/tcc-instance-segm.

Por fim, ocorreu a etapa de **análise**, cujo objetivo é extrair conclusões dos resultados obtidos. Quando ideias complementares surgiam nesse ponto, por exemplo, avaliar as imagens individualmente, novos experimentos eram realizados, o que explica o ciclo na Figura 7.

4.2 Dados

O treinamento inicial dos modelos, realizados pelos autores dos artigos originais, foi feito no COCO (LIN *et al.*, 2014). Esse dataset tem 1.5 milhão de objetos anotados, de 80 classes diferentes, e é geralmente o padrão para trabalhos na área. Basicamente, os autores treinam os modelos em um conjunto de dados, disponíveis para o público em geral, rodam a inferência em outro conjunto de dados, o conjunto de testes, cujas anotações não estão disponíveis para o público, e então enviam as predições para esse conjunto de testes para um servidor, gerenciado pelos autores do dataset, para que a avaliação seja feita.

Como o modelos foram treinados nos dados públicos do COCO e não há como trabalhar diretamente com as anotações de testes, já que essas são mantidas em segredo, foi necessário utilizar outro conjunto de dados para os testes. Optou-se por utilizar um subconjunto de dados do OpenImages (KUZNETSOVA *et al.*, 2020), um dataset que tem pouco mais de 2.7 milhões de objetos anotados, de 350 classes diferentes, na sua versão atual. Por questões de tempo e recursos computacionais, os modelos **não** foram treinados nesses dados. Os pesos das redes neurais se referem apenas ao treinamento feito no COCO, pelos autores dos modelos. Obviamente, para a análise, foram consideradas somente classes em comum entre os dois datasets.

Foram selecionadas 200 imagens contendo pelo menos um objeto de uma das 10 classes mais comuns no COCO. Após filtrar apenas as classes em comum entre os datasets, obteve-se um total de 306 objetos anotados, de 13 classes. Estas classes adicionais apenas estavam presentes nas anotações adquiridas e, como elas eram avaliáveis (isto é, previsíveis pelos modelos), optou-se por mantê-las, ao invés das 10 classes originais. A distribuição de classes desses objetos pode ser vista na Tabela 1.

Tabela 1 – Distribuição de classes nos dados de teste. N é o número de ocorrências.

Classe	N	Classe	N	Classe	N
avião (airplane)	2	tigela (bowl)	5	ovelha (sheep)	11
pássaro (bird)	42	carro (car)	39	semáforo (traffic light)	6
livro (book)	29	bolsa (handbag)	6	caminhão (truck)	43
garrafa (bottle)	64	cavalo (horse)	6	vaso (vase)	4
		pessoa (person)	49		

Fonte: Autoria própria (2023).

4.3 Métricas

4.3.1 Qualidade

Em problemas de classificação é relativamente fácil dividir as predições entre certas e erradas. Em detecção e segmentação, no entanto, esta separação não é tão simples pelo fato de ser extremamente improvável que o modelo produza *exatamente* a saída esperada. É necessário definir uma medida de qualidade, algo que descreva o quão boa ou ruim é uma determinada segmentação, para depois categorizá-la entre certa e errada.

Uma dessas medidas, que é amplamente usada pelos pesquisadores, é a interseção sobre união, ou IoU (do inglês, *Intersection over Union*), apresentada na Equação 3 (REZATOFIGHI *et al.*, 2019). Ela produz valores entre 0 e 1, que representam o quão bem a segmentação se encaixa no objeto real.

$$\text{IoU} = \frac{\text{área de interseção}}{\text{área de união}} \tag{3}$$

No caso de detecção de objetos, é computado o IoU entre a *bounding box* produzida pelo modelo e as *bounding boxes* reais da mesma classe. Se o IoU máximo for maior do que um limite (por exemplo, 0.5), a detecção é considerada certa. Se for menor do que o limite ou não houverem *bounding boxes* reais da mesma classe, a detecção é considerada errada. Na segmentação de instâncias o procedimento é o mesmo, mas considerando as máscaras de segmentação ao invés das *bounding boxes*.

Uma vez definido quais detecções/segmentações estão corretas, é possível computar diversas métricas para avaliar o desempenho dos modelos. A mais adotada pelos trabalhos da área é o mAP, que será explicado na sequência.

Primeiro, suponha um limiar de IoU, digamos, 0.5, e um limite de confiança na classificação, digamos, 0.6. Não precisa ser o mesmo número, já que esses dois conceitos são independentes (confiança se refere à classificação, e IoU, à sobreposição das máscaras). Para cada classe, separadamente, as detecções/segmentações são categorizadas entre:

- **Verdadeiros positivos** (VP): objetos que aparecem na imagem de entrada e que foram detectados pelo modelo.

- **Falsos positivos** (FP): objetos que foram detectados pelo modelo, mas que não aparecem na imagem de entrada.
- **Falsos negativos** (FN): objetos que aparecem na imagem de entrada, mas que não foram detectados pelo modelo.

Com isso, é possível calcular a precisão (*precision*) e a revocação (*recall*). A precisão, dada pela Equação 4, indica a proporção de predições positivas que de fato são positivas. O *recall*, dado pela Equação 5, indica a proporção de casos positivos que foram preditos como tal (DAVIS; GOADRICH, 2006, p. 2-3). Aqui, a precisão representa a proporção de objetos detectados que realmente existem, e o *recall* representa a proporção de objetos reais que foram de fato detectados.

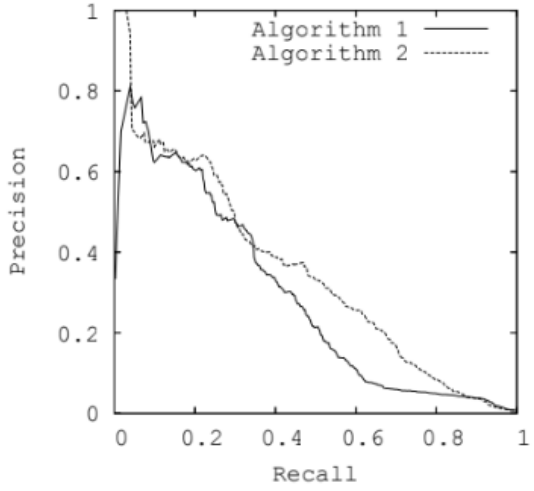
$$\text{Precisão} = \frac{\text{VP}}{\text{VP + FP}} \tag{4}$$

$$\textit{Recall} = \frac{\text{VP}}{\text{VP + FN}} \tag{5}$$

O ideal é ter um modelo onde tanto a precisão quanto o *recall* são altos, já que os demais casos são normalmente piores. *Precisão alta* e *recall baixo* significa que o seu modelo está deixando de prever muitos objetos a favor da precisão. *Recall alto* e *precisão baixa*, por outro lado, significa que o modelo está identificando muitos objetos, mas também errando muitos objetos.

Diferentes limiares de confiança resultam em diferentes números de objetos em cada categoria (VP, FP, FN), e por consequência, diferentes valores de precisão e *recall*. Com estes valores, pode-se montar uma curva de *precision-recall* (Figura 8).

Figura 8 – Exemplo de uma curva de *precision-recall*.



Fonte: (DAVIS; GOADRICH, 2006, p. 2).

Embora esta curva ajude a comparar diferentes modelos, uma forma muito mais simples é através de um valor numérico, a área sob esta curva, chamada de **precisão média** ou simplesmente AP (do inglês *Average Precision*). A forma exata para computar essa média varia, mas normalmente envolve algum tipo de interpolação para suavizar a curva. Aqui, foi utilizado a interpolação de 101 pontos da curva, que é o padrão utilizado no código de avaliação.

Finalmente, após computar o AP de cada classe, é computada a média do AP de todas as classes, o chamado mAP (*mean Average Precision*), resultando assim em um único valor que indica o desempenho do modelo em uma determinada imagem ou *dataset*. A métrica utilizada nos trabalhos vai além, calculando a média entre 10 limiares de IoU, para recompensar modelos com melhor localização. É comum os pesquisadores da área se referirem ao mAP apenas como AP, já que as tarefas de detecção e segmentação são, por natureza, multi-classe.

Os autores frequentemente detalham várias métricas em diversas variações na arquitetura dos modelos, mas aqui, como o objetivo é fazer uma análise mais genérica, somente as principais serão discutidas. Dos trabalhos originais, foi avaliado o AP padrão, que é o AP para 10 limiares de IoU, e o AP para diferentes escalas de objetos, da melhor variação de cada modelo, usando o AP padrão como critério para a seleção. Para o subconjunto do OpenImages, foi avaliado o AP padrão, o número de predições e o número de verdadeiros positivos, falsos positivos e falsos negativos. O AP em diferentes escalas não foi avaliado porque, já que essa é uma característica “intrínseca” dos modelos, por assim dizer, ditada pela arquitetura e pelos dados em que eles foram treinados, os resultados devem ser os mesmos do que os reportados pelos autores.

Além das métricas no *dataset*, foram computadas algumas métricas por imagem. Mais especificamente, para cada imagem, foi computado o AP padrão e uma lista de verdadeiros positivos, falsos positivos e falsos negativos, para visualização. As imagens então foram ordenadas por AP médio (entre os três modelos) e pela diferença entre o melhor e o pior modelo. As análises focaram principalmente nos casos mais extremos:

- **Média alta, diferença baixa**: imagens em que todos os modelos tiveram uma boa performance.
- **Média baixa, diferença baixa**: imagens em que todos os modelos tiveram uma má performance.
- **Diferença alta**: imagens em que a maioria dos modelos tiveram uma performance similar, mas que algum se sobressaiu, positiva ou negativamente.

4.3.2 Velocidade

Para a velocidade, foram considerados tanto o tempo de inferência reportado pelos autores dos modelos, em uma GPU, quanto no computador pessoal do autor dessa pesquisa,

em uma CPU. O objetivo, além de verificar qual modelo é o mais rápido, é ver como os modelos rodam em um computador comum, não projetado especificamente para aprendizado de máquina.

4.3.3 Referencia rápida

Em resumo, a lista abaixo descreve todas as métricas utilizadas e os seus significados:

- **AP**: mAP, calculado sobre 10 limiares de IoU (de 0.5 a 0.95, em intervalos de 0.05).
- **AP**^{small}: AP padrão considerando apenas objetos pequenos (com área inferior a 32^2). Normalmente mencionado nos trabalhos como AP^S.
- **AP**^{medium}: AP padrão considerando apenas objetos médios (com área entre 32^2 e 96^2). Normalmente mencionado nos trabalhos como AP^M.
- **AP**^{large}: AP padrão considerando apenas objetos grandes (com área superior a 96^2). Normalmente mencionado nos trabalhos como AP^L.
- **N**$_{obj}$: número de objetos preditos pelo modelo, independente desses objetos existirem ou não.
- **VP**: número de verdadeiros positivos, objetos preditos que existem nas anotações.
- **FP**: número de falsos positivos, objetos preditos que não existem nas anotações.
- **FN**: número de falsos negativos, objetos anotados que não foram preditos.
- **Velocidade**: tempo médio de inferência por imagem.
- **FPS**: número de imagens que um modelo consegue processar por segundo.

5 MODELOS AVALIADOS

Esta seção apresenta uma descrição simplificada dos modelos utilizados no trabalho. O objetivo aqui é apenas fornecer ao leitor uma noção de como os modelos funcionam. Para mais detalhes sobre os tópicos mencionados, consulte os trabalhos originais.

5.1 Mask R-CNN

Proposto por He *et al.* (2017), o Mask R-CNN é uma extensão do Faster R-CNN, o modelo de detecção de objetos de Ren *et al.* (2015).

O Faster R-CNN, basicamente, funciona assim:

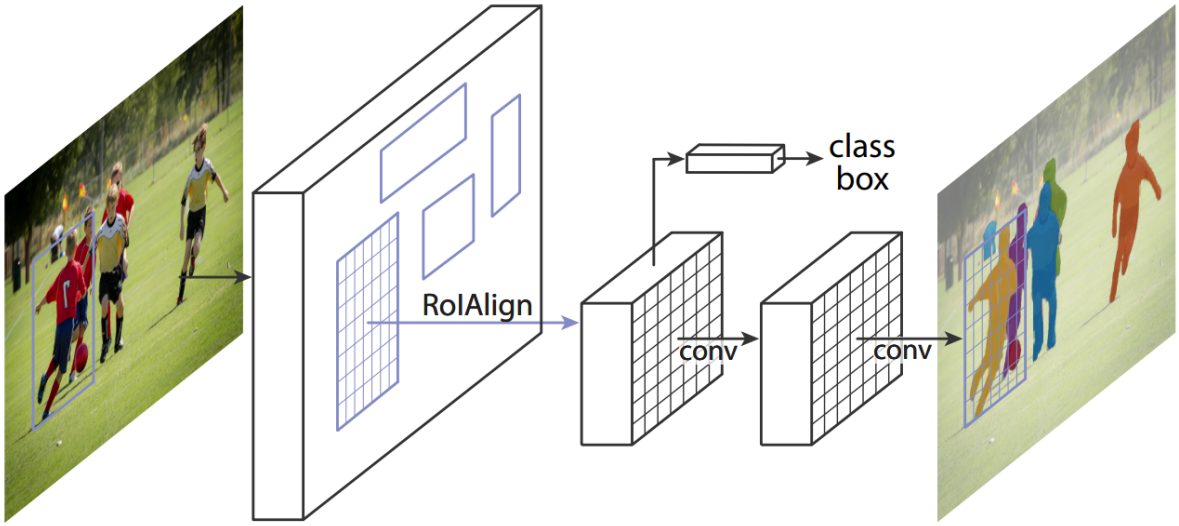
- O modelo recebe uma imagem como entrada e computa características dessa imagem, com uma CNN – isso é comum em praticamente todos os modelos de todas as áreas de visão computacional.
- Ele então usa essas características para obter um conjunto de regiões/*bounding boxes* com alta probabilidade de conter um objeto. Esse processo é feito pela chamada *Region Proposal Network*, ou RPN, para abreviar. Basicamente, o que essa rede faz é "adaptar" (regredir) um conjunto de *bounding boxes* pré-definidas, chamadas de *anchors*, e então classificá-las com base em sua "objetividade" (o quão provável é que exista um objeto naquela região).
- Então, daqui em diante, cada região é processada individualmente. O modelo extrai características dessa região dos mapas computados anteriormente, em um passo chamado de *RoI pooling*, e então usa essas características para: A) classificar o objeto; e B) refinar a *bounding box* para se adequar melhor ao objeto.

O Mask R-CNN é praticamente a mesma coisa, mas com um terceiro ramo no último passo para produzir a máscara de segmentação. Um diagrama pode ser visto na Figura 9. Esse ramo é uma rede totalmente convolucional (FCN) que prediz uma máscara para cada classe. Depois da classificação, apenas a máscara daquela respectiva classe é mantida. O "recorte" das características para cada região também é um pouco diferente, já que, para gerar as máscaras, são necessário mapas de características mais precisos para cada região, mas a ideia ainda é a mesma: obter características sobre essa região.

5.2 YOLACT

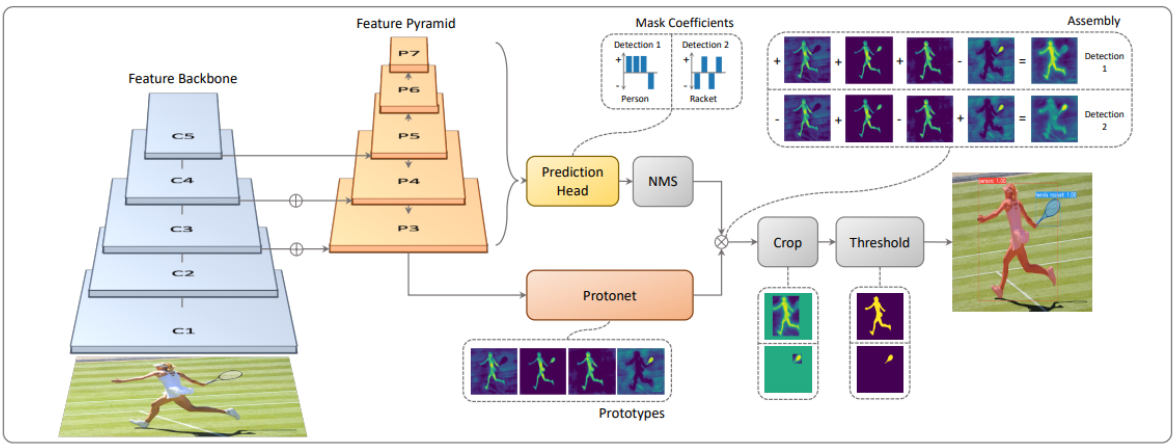
Proposto por Bolya *et al.* (2019b), o YOLACT é um modelo projetado para velocidade. Um diagrama pode ser visto na Figura 10. Basicamente, ele:

Figura 9 – Visão geral da arquitetura do Mask R-CNN. Resumidamente, ele recebe uma imagem como entrada, computa características e um conjunto de regiões de interesse e então, para cada região, classifica e segmenta o objeto.



Fonte: (HE *et al.*, 2017).

Figura 10 – Visão geral da arquitetura do Yolact. Resumidamente, ele recebe uma imagem, computa características e então combina máscaras geradas para a imagem inteira com a saída de um detector de objetos para gerar as máscaras individuais.



Fonte: (BOLYA *et al.*, 2019b), (BOLYA *et al.*, 2019a).

- Recebe uma imagem e computa características com uma CNN.
- Então, em paralelo:
  - Gera um conjunto de “máscaras protótipo” para a imagem, usando uma rede totalmente convolucional (FCN). Essas máscaras capturam diversas características da imagem. Por exemplo, uma pode ativar objetos no lado esquerdo da imagem, ou direito, outra bordas de objetos e assim por diante.
  - Detecta os objetos da imagem usando um detector de um estágio. Para cada objeto, também é gerado um conjunto de coeficientes, basicamente indicando o quanto de cada máscara protótipo é necessário para “construir” o objeto.
- Por fim, para cada detecção, ele multiplica as máscaras protótipo pelos coeficientes, recortando só a área demarcada pela *bounding box* predita, para gerar as máscaras finais.

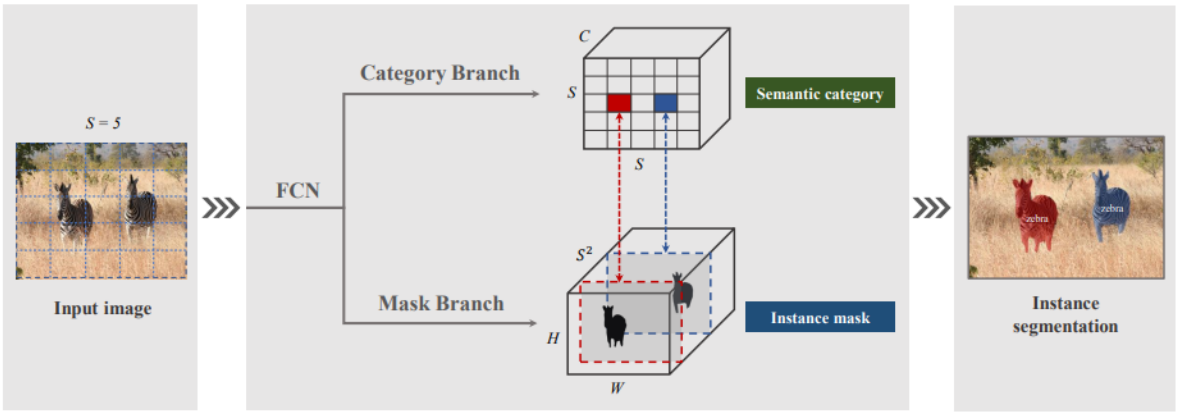
Mais tarde, os autores disponibilizaram outra versão do modelo, o YOLACT++ (BOLYA *et al.*, 2019a), com diversas melhorias na arquitetura e resultados melhores. Essas melhorias não serão discutidas aqui, porque a ideia geral do modelo ainda é a mesma. Elas são apenas otimizações, não mudanças significativas na arquitetura.

5.3 SOLO

Proposto por (WANG *et al.*, 2019), o SOLO é um modelo que tenta segmentar objetos *diretamente*, sem o uso de propostas de região e *anchors*, como nas abordagens anteriores. Um diagrama pode ser visto na Figura 11. Basicamente, ele divide a imagem em N células, e cada célula é então responsável por prever o objeto cujo centro pertence à esta célula. Na prática, o que acontece é que a saída tem N saídas concatenadas, cada uma referente a uma área da imagem (uma célula). A forma como eles introduzem a “variação espacial” (a habilidade de cada saída corresponder a uma parte da imagem, e não à imagem inteira) é adicionando as coordenadas dos *pixels* ao vetor de entrada – tornando a localização das células uma característica a ser aprendida, de certa forma. Algumas camadas aprendem a detectar os objetos da primeira célula, outras da segunda e assim por diante.

Assim como o Yolact, os autores mais tarde disponibilizaram outra versão do modelo, o SOLOv2 (WANG *et al.*, 2020), com diversas melhorias, mas nada que mude a ideia geral do modelo. Essas melhorias não serão discutidas aqui.

Figura 11 – Visão geral da arquitetura do SOLO. Resumidamente, ele recebe uma imagem, divide ela (conceitualmente) em uma grade de células, e então cada célula prediz a classe e a máscara do objeto cujo centro pertence à esta célula.



Fonte: (WANG *et al.*, 2019).

6 RESULTADOS

6.1 Qualidade a nível de *dataset*

A Tabela 2 apresenta o resultado dos modelos no conjunto de testes do COCO, ou seja, os resultados reportados pelos autores dos modelos nos trabalhos originais. É possível perceber que os três modelos são bem similares em termos de performance. O Yolact tem os resultados mais baixos porque os autores priorizam velocidade ao invés de performance, mas ele ainda alcança resultados competitivos. O Mask R-CNN é, no geral, melhor do que o Yolact, exceto para objetos grandes, onde parece ser levemente inferior a ele. Surpreendentemente, o SOLO, o modelo menos mencionado entre os três, pelo menos durante o levantamento bibliográfico dessa pesquisa, alcança os melhores resultados entre eles. Especialmente para objetos médios e grandes, onde podemos ver um grande diferença em AP, 5 pontos aproximadamente, o que é um intervalo bastante considerável.

Tabela 2 – Resultados no COCO, reportados pelos autores dos modelos. O AP é normalizado para [0,100], em vez do tradicional [0,1], por conveniência, como é feito na maioria dos trabalhos da área.

Modelo	**Variação**	**AP**	**AP**^{small}	**AP**^{medium}	**AP**^{large}
Mask R-CNN	ResNeXt-101-FPN	37.1	16.9	39.9	53.5
Yolact	YOLACT-550+ – R-101-FPN	34.6	11.9	36.8	55.1
SOLO	SOLOv2 – Res-DCN-101-FPN	41.7	18.0	45.0	61.6

Fonte: Trabalhos originais. He *et al.* (2017), Bolya *et al.* (2019a) e Wang *et al.* (2020).

Como as abordagens são bem diferentes, é difícil apontar exatamente *por que* o SOLO é melhor. Uma hipótese é que esse ganho seja resultado da concatenação das coordenadas dos *pixels* na entrada, algo que só o SOLO faz. Talvez essa etapa permita a rede aprender mais características de cada região da imagem, o que resulta em máscaras mais precisas. Pesquisas futuras podem comprovar ou refutar essa teoria, adicionando essa funcionalidade a outros modelos e verificando se isso melhora os resultados.

A Tabela 3, por sua vez, apresenta o resultado dos modelos no subconjunto do OpenImages, descrito no Capítulo 4. É importante notar que essas **não** são as melhores variações de cada modelo, por uma questão de recursos – tanto computacionais (o computador utilizado não ser capaz de rodar as arquiteturas mais profundas) quanto de tempo (para algumas variações era necessária a instalação de componentes adicionais, que atrasariam os experimentos). As variações utilizadas estão descritas na tabela, seguindo a mesma nomenclatura dos trabalhos originais.

Como esperado, uma performance similar, já que os *datasets* são parecidos em termos de variedade de imagens e apenas as classes em comum estão sendo avaliadas (lembre-se, não houve *fine-tuning*, então os modelos só podem prever as classes do COCO). Aqui, no entanto, podemos ver alguns detalhes interessantes, como o fato de que o Mask R-CNN faz *bem*

Tabela 3 – Resultados no subconjunto de 200 imagens selecionadas do OpenImages. A soma de VP e FP não totaliza o N_{obj} porque este valor se refere a *todos* os objetos que o modelo previu, independente de ser uma classe avaliável (presente no subconjunto do OpenImages) ou não.

Modelo	Variação	AP	N_{obj}	VP	FP	FN
Mask R-CNN	ResNet-50-FPN	37.1	1486	224	1037	82
Yolact	YOLACT-550 R-101-FPN	31.7	610	163	373	143
SOLO	SOLOv2 Res-50-FPN	42.4	559	180	325	126

Fonte: Autoria própria (2023).

mais predições do que outros modelos. Talvez isso seja resultado do alto número de propostas geradas pela RPN, mas seriam necessários mais experimentos para chegar a uma conclusão.

Outro ponto relevante é o alto número de falsos positivos, em todos os modelos. Claro, parte do motivo pode ser intrínseco aos modelos e como eles foram treinados, mas, depois de investigar as anotações, notou-se que diversas imagens desse dataset não estão anotadas corretamente – não no sentido de máscaras ruins, que não correspondem aos objetos, mas de *falta de máscaras*. Objetos que claramente *deveriam* estar anotados, não estão. A Figura 12 tem alguns exemplos disso. Também seria interessante verificar os resultados por classe, mas, por questões de tempo, esta análise teve que ser cortada do trabalho.

Figura 12 – Exemplos de imagens mal anotadas no OpenImages. À esquerda é possível ver as anotações, com diversos objetos não anotados. À direita, um exemplo de predições para essas imagens, para comparação. Ou as imagens não estão anotadas corretamente, ou seja, é um problema *no dataset*, ou a API fornecida pelos mantenedores do dataset tem algum erro, e não é capaz de baixar todas as anotações de algumas imagens.

Fonte: Autoria própria (2023).

6.2 Velocidade

A Tabela 4 mostra o desempenho dos modelos conforme reportado pelos autores, nos trabalhos originais. Sabendo que essas GPUs são relativamente similares em termos de performance (obviamente existem diferenças, mas isto está além do escopo dessa pesquisa), nós podemos ver o quão grande é a diferença entre Yolact e os demais modelos. O Mask R-CNN é, de longe, o mais lento, provavelmente por conta da etapa de geração de propostas, onde centenas de possíveis objetos são detectados, para então serem classificados e segmentados. O SOLO evita essa etapa, como mencionado no Capítulo 5, mas parece ter outra etapa de processamento que o atrasa. Os autores discutem algumas variações na segunda versão do modelo (WANG *et al.*, 2020), mas para alcançar velocidades mais altas é necessário sacrificar um pouco da qualidade nos resultados (ou seja, um AP mais baixo).

Tabela 4 – Velocidade de inferência para a melhor variação de cada modelo.

Modelo	Variação	GPU	Velocidade
Mask R-CNN	ResNet-101-FPN	Nvidia Tesla M40 GPU	~200ms (5 fps)
Yolact	YOLACT-550++ – R-101-FPN	Titan Xp	~36ms (~27 fps)
SOLO	SOLOv2 – Res-DCN-101-FPN	V100	~100ms (10 fps)

Fonte: Trabalhos originais. He *et al.* (2017), Bolya *et al.* (2019a) e Wang *et al.* (2020).

A Tabela 5 mostra o desempenho em uma CPU AMD FX-8300 8-Core 3.30 GHz, 8GB RAM, um computador doméstico. Como esperado, a inferência é *bem* mais lenta do que em uma GPU, mas não *tanto* ao ponto de ser inviável, dependendo da aplicação. No caso desta pesquisa, por exemplo, onde o objetivo era apenas realizar a inferência em um pequeno conjunto de imagens, é totalmente possível executar modelos de segmentação de instâncias em um computador comum. Para *treinar* um modelo, no entanto, uma GPU seria praticamente uma necessidade, já que, como os autores reportam, treinar os modelos no COCO, usando uma GPU, leva alguns dias. Sem uma GPU, isso levaria semanas, se não meses. Felizmente existem opções envolvendo computação em nuvem, como o Google Colab, mencionado anteriormente, para facilitar o acesso a tais GPUs, se necessário.

Em relação à performance, aqui podemos notar a diferença entre as GPUs utilizadas nos trabalhos originais. Quando executados no mesmo computador, o Mask R-CNN tem uma performance similar ao SOLO, ao contrário do que a Tabela 4 sugere. A velocidade do Yolact, por outro lado, se mantém, mostrando que ele é, de fato, a abordagem mais rápida.

Tabela 5 – Velocidade de inferência, em uma máquina com CPU, para as variações selecionadas.

Modelo	Variação	Velocidade
Mask R-CNN	ResNet-50-FPN	10s (0.1 fps)
Yolact	YOLACT-550 R-101-FPN	4s (0.25 fps)
SOLO	SOLOv2 Res-50-FPN	11s (0.09 fps)

Fonte: Autoria própria (2023).

6.3 Análises de imagens específicas

Nesta seção, as cores da primeira imagem de cada exemplo são aleatórias – elas simplesmente representam as anotações do *dataset*. As cores nas demais imagens, referente ao resultado dos modelos, representam a categoria em que elas foram colocadas durante a avaliação. Verde significa que é um verdadeiro positivo (VP), ou seja, uma previsão correta, que corresponde a uma anotação. Laranja, que é um falso positivo (FP), uma previsão que não corresponde a nenhuma anotação e, portanto, errada. Por fim, os objetos em azul são falsos negativos (FN), anotações que não correspondem a nenhuma das previsões do modelo, e portanto, também erradas.

6.3.1 Média alta, diferença baixa

Primeiro, alguns casos em que todos os modelos obtiveram um bom desempenho. Como podemos ver na Figura 13, casos simples, com um ou dois objetos, são facilmente segmentados pelos modelos. Na Figura 14, no entanto, nós começamos a ver um problema. Os modelos predizem mais objetos do que os que estão anotados no *dataset*. Se você olhar atentamente, verá que uma grande parte desses objetos estão tecnicamente certos, eles só não estão anotados e, portanto, são considerados falsos positivos.

Uma pergunta que pode surgir aqui é: "Esses falsos positivos não deveriam reduzir o AP? Como ele ainda é 1.0?". A resposta tem a ver com como o AP é calculado. Cada classe é avaliada separadamente. Quando não há anotações de uma determinada classe (nesse caso, *pessoa/person* e *caminhonete/truck*), é impossível calcular o *recall* para essas classes (divisão por zero). O que acaba acontecendo é que esses casos são ignorados durante o cálculo do AP. A Figura 15 mostra outro exemplo disso, dessa vez com as classes *pessoa/person* e *bolsa/handbag*. A bolsa está anotada, então é levada em consideração durante o cálculo do AP, mas a pessoa não está, o que cria essa situação inusitada de AP máximo, mesmo com falsos positivos.

Isso é mais um problema com o dataset em si, já que esses objetos deveriam estar anotados, mas nós também podemos olhar para isso do ponto de vista da métrica. Um falso positivo ainda é um **falso** positivo. Algo que não deveria acontecer. Talvez seja interessante incluir esses casos no cálculo do AP em pesquisas futuras, para ser capaz de diferenciar um modelo que só prediz os verdadeiros positivos de um que faz o que foi descrito aqui (verdadeiros positivos de classes anotadas e falsos positivos de classes não-anotadas).

Figura 13 – Resultados no subconjunto do OpenImages - Exemplo 1. Consulte o início da seção 6.3 para o significado das cores.

(a) Anotações **(b) Mask R-CNN – AP 1.0**

(c) Yolact – AP 1.0 **(d) Solo – AP 1.0**

Média: 1.0 Diferença entre o melhor e o pior: 0.0

Fonte: Autoria própria (2023).

Figura 14 – Resultados no subconjunto do OpenImages - Exemplo 2. Consulte o início da seção 6.3 para o significado das cores.

(a) Anotações **(b) Mask R-CNN – AP 1.0**

(c) Yolact – AP 1.0 **(d) Solo – AP 1.0**

Média: 1.0 Diferença entre o melhor e o pior: 0.0

Fonte: Autoria própria (2023).

Figura 15 – Resultados no subconjunto do OpenImages - Exemplo 3. Consulte o início da seção 6.3 para o significado das cores.

(a) Anotações **(b) Mask R-CNN – AP 1.0**

(c) Yolact – AP 1.0 **(d) Solo – AP 1.0**

Média: 1.0 Diferença entre o melhor e o pior: 0.0

Fonte: Autoria própria (2023).

6.3.2 Média baixa, diferença baixa

Conforme mencionado na seção anterior, a falta de anotações limita bastante a avaliação. A maioria dos casos onde os modelos tem baixa performance, pelo menos nesse *dataset*, são uma consequência da falta de anotações. A Figura 16 é um ótimo exemplo disso. Nós podemos ver que todos os modelos foram capazes de detectar diversos carros na imagem e, no caso do Mask R-CNN e do SOLO, algumas pessoas também. Os carros não afetam o AP porque, como foi explicado na seção anterior, classes que não estão anotadas em uma imagem são ignoradas, mas as pessoas afetam, porque há uma pessoa anotada. Toda pessoa categorizada como um falso positivo ou um falso negativo reduz o AP final, resultando em um baixo desempenho de todos os modelos.

Figura 16 – Resultados no subconjunto do OpenImages - Exemplo 4. Consulte o início da seção 6.3 para o significado das cores.

(a) Anotações **(b) Mask R-CNN – AP 0.083**

(c) Yolact – AP 0.0 **(d) Solo – AP 0.0**

Média: 0.028 Diferença entre o melhor e o pior: 0.083

Fonte: Autoria própria (2023).

Nessa imagem também é possível observar o “bias de mais predições” do Mask R-CNN, mencionado na seção 6.1. Ele previu bem mais objetos do que os outros dois, 24, para ser exato, o que é um pouco mais do que o dobro do que os outros modelos previram.

Como grande parte das imagens que se encaixam nessa categoria são resultado da falta de anotações, esse será o único exemplo discutido nessa seção. Talvez pesquisas futuras possam realizar experimentos similares em outros *datasets*, para encontrar casos em que todos os modelos tem dificuldade em prever, mas isso está além do escopo dessa pesquisa. Outro fator que poderia ajudar na avaliação é agrupar classes similares, como *carro/car* e *caminho-*

nete/truck, ambas previsíveis pelos modelos, ou *copo/glass*, que é uma classe previsível pelos modelos, e *taça/wine glass*, uma classe presente nas anotações.

6.3.3 Diferença alta

Essa seção apresenta algumas imagens nas quais houve discrepância entre os modelos, que podem ser um indício de algum problema com a sua arquitetura ou *design*.

A Figura 17 é um exemplo onde apenas o SOLO teve baixo desempenho. Ele de alguma forma conseguiu prever a pessoa dentro do carro, mas não o carro. Durante a pesquisa, não foi possível encontrar nenhuma explicação para este ocorrido. Parece ser um caso isolado, já que o SOLO obteve os melhores resultados tanto no COCO quanto no subconjunto do OpenImages.

Figura 17 – Resultados no subconjunto do OpenImages - Exemplo 5. Consulte o início da seção 6.3 para o significado das cores.

(a) Anotações **(b) Mask R-CNN – AP 1.0**

(c) Yolact – AP 1.0 **(d) Solo – AP 0.0**

Média: 0.666 Diferença entre o melhor e o pior: 1.0

Fonte: Autoria própria (2023).

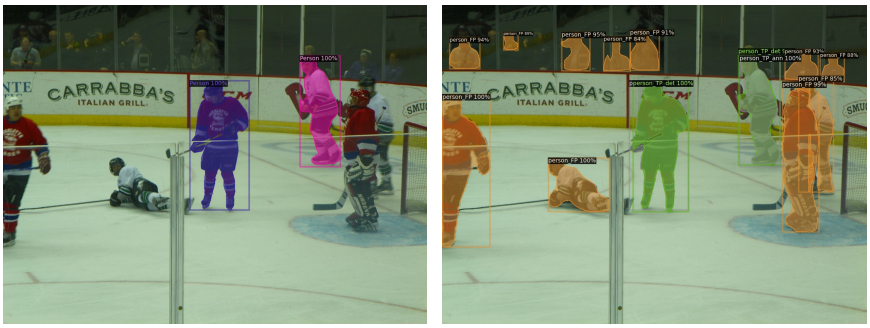
Um detalhe interessante que podemos ver nessa imagem é que todos os modelos conseguiram prever a pessoa *dentro* do carro. Parte do motivo pode ser atribuído ao fato de pessoa ser uma classe “fácil” de prever – tem uma alta representatividade no COCO, possui características relativamente fáceis de serem identificadas, como boca, nariz, etc –, mas ainda assim, é impressionante. Durante as etapas iniciais desta pesquisa, acreditava-se que a oclusão seria um grande obstáculo que os modelos teriam que resolver, mas os resultados mostram que ela

não é *tão* limitante quanto se havia imaginado. Claro, diversos autores propõem formas de mitigar esse problema, mas, mesmo sem elas, os modelos conseguem obter resultados razoáveis.

Em seguida, na Figura 18, uma imagem em que apenas o Yolact teve baixo desempenho. Aqui nós podemos ver, novamente, o problema da falta de anotações. O Yolact detectou diversas pessoas na imagem, mas não as duas que estão anotadas, então seu resultado foi zero. Quanto ao motivo dele não ter predito essas duas pessoas, talvez as camadas responsáveis por gerar as máscaras protótipos tenham divido as pessoas em dois objetos, por conta do capacete. Como podemos ver na imagem, o Yolact só conseguiu prever uma pessoa usando o capacete, enquanto o Mask R-CNN e o SOLO previram todas. Seria necessário analisar os mapas de ativação da rede para ver o que eles capturaram.

Figura 18 – Resultados no subconjunto do OpenImages - Exemplo 6. Consulte o início da seção 6.3 para o significado das cores.

(a) Anotações **(b) Mask R-CNN – AP 0.652**



(c) Yolact – AP 0.0 **(d) Solo – AP 0.628**

Média: 0.427 Diferença entre o melhor e o pior: 0.652

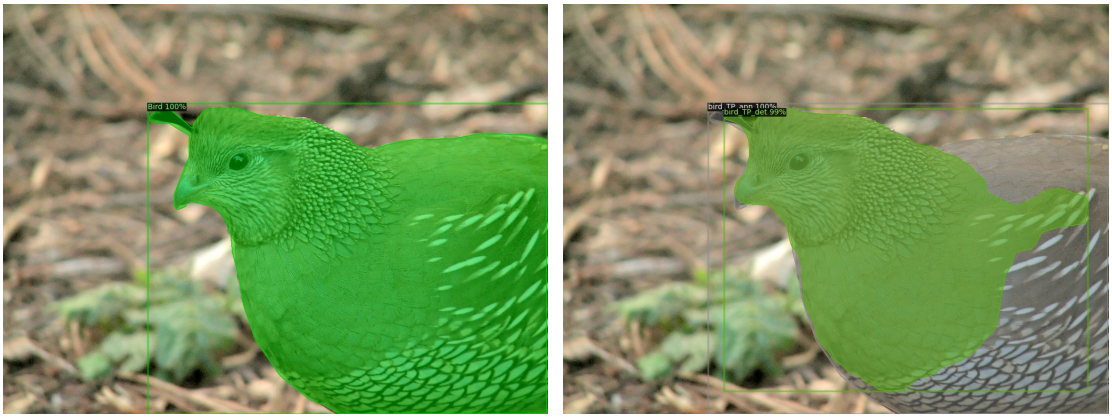
Fonte: Autoria própria (2023).

Na Figura 19 temos um exemplo onde o Mask R-CNN obteve baixo desempenho. Ele conseguiu *detectar* o pássaro, mas a máscara não se adequa muito bem ao objeto real. Não foi possível encontrar nenhuma justificativa para o ocorrido. Uma coisa interessante, no entanto, é como o AP padrão ajuda a diferenciar esses casos. O AP do Mask R-CNN nessa imagem não foi zero, o que significa que, para pelo menos um dos limiares de IoU, a segmentação foi considerada correta. Entretanto, como o resultado final é a média de vários limiares, o resultado foi menor do que os demais modelos, que produziram uma máscara mais fiel.

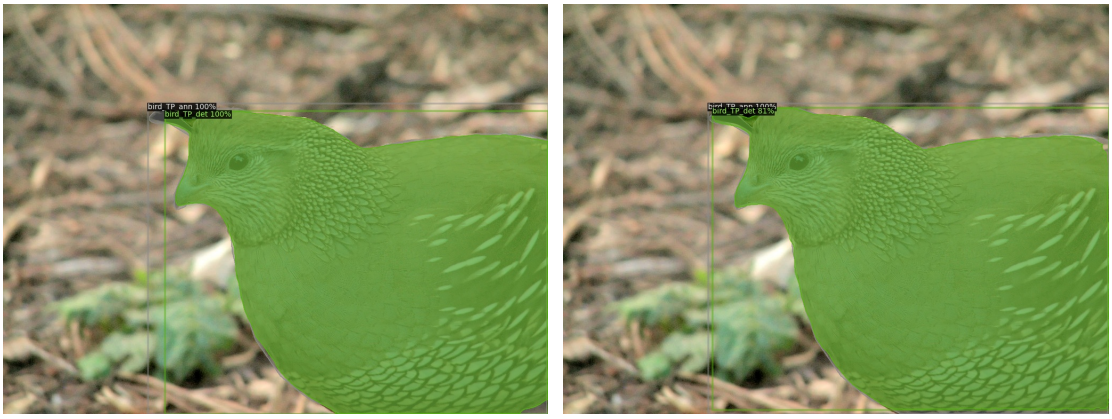
Um dos objetivos iniciais desta pesquisa era encontrar casos em que as segmentações dos modelos variam bastante, mas como os resultados mostram, os três modelos são bem

Figura 19 – Resultados no subconjunto do OpenImages - Exemplo 7. Consulte o início da seção 6.3 para o significado das cores.

(a) Anotações **(b) Mask R-CNN – AP 0.3**



(c) Yolact – AP 1.0 **(d) Solo – AP 1.0**



Média: 0.766 Diferença entre o melhor e o pior: 0.7

Fonte: Autoria própria (2023).

Figura 20 – Resultados no subconjunto do OpenImages - Exemplo 8. Consulte o início da seção 6.3 para o significado das cores.

(a) Anotações **(b) Predições do SOLO.**

Fonte: Autoria própria (2023).

similares em termos de performance. Seria necessário uma análise bem mais profunda dos resultados para detectar algum padrão, como, digamos, o modelo X ter uma tendência maior a prever máscaras maiores do que os objetos (*overshooting*), menores (*undershooting*) ou algo parecido.

Por fim, outra observação, dessa vez das predições originais dos modelos, não das visualizações de VPs/FPs/FNs. Na Figura 20, podemos ver que o SOLO classificou esse objeto como sendo uma moto em vez de um carro. O que acaba acontecendo durante o cálculo do AP é que, a moto será ignorada, porque não há anotações de motos na imagem, e o carro será um falso negativo, resultando em um AP baixo. Um ponto a se pensar aqui é "Esse AP deveria ser *tão* baixo?".

Mesmo que a *segmentação* do SOLO seja boa, o fato da *classificação* estar errada impacta *muito* no AP final. Talvez esse fator poderia ser incorporado nas métricas, de alguma forma? Digamos, além de computar o AP para cada classe, incluir um AP genérico, sem levar em consideração as classes? Obviamente é melhor se um modelo conseguir prever tanto a classe quanto a localização do objeto, mas, dependendo do caso, pode fazer sentido dar mais peso para uma dessas coisas, como para carros autônomos. No final das contas, não importa tanto se ele classifica algo como *carro* ou *caminhonete*. A parte relevante é que ele detectou **alguma coisa**, e o carro deve *evitar* essa coisa. Talvez existam algumas variações da métrica que são capazes de lidar com estes casos, mas o que o levantamento bibliográfico desta pesquisa mostrou é que a maioria dos trabalhos só utiliza o AP padrão para a avaliação.

7 CONCLUSÕES

Segmentação de instâncias é um problema difícil. Ao longo dos anos, pesquisadores da área fizeram diversos avanços incríveis, mas, como foi dito na introdução, sempre há espaço para melhorias. O objetivo desse trabalho era realizar a comparação de três arquiteturas bem estabelecidas na área e identificar pontos nos quais os modelos pudessem ser melhorados. Como a pesquisa mostrou, os modelos são todos similares em termos de performance, com o SOLO levemente à frente. Eles ainda estão longe de serem perfeitos, mas considerando a dificuldade do problema, isso é, no mínimo, esperado. Surpreendentemente, o que acabou sendo encontrado foram limitações não nos modelos, mas no *dataset* e na métrica utilizada, o AP.

Durante a etapa final da pesquisa, notou-se que outros autores também apontaram problemas similares, particularmente em relação ao AP. Os autores do YOLO, na terceira versão do modelo (REDMON; FARHADI, 2018), já apresentaram alguns casos onde o AP produz resultados conflitantes, mostrando, como foi feito aqui, que o AP utilizado não é a métrica mais adequada para o problema. Eles sugerem se livrar do AP por classe e calcular apenas o AP global, sem levar em consideração as classes, ou, outra possível melhoria, calcular o AP por imagem e depois a média entre as imagens. Outro trabalho, mais recente (JENA *et al.*, 2023) também discute algumas ideias de como melhorar essa métrica.

Conforme a pesquisa avançar na área, e mais e mais pesquisadores perceberem as limitações do AP utilizado hoje, talvez outra métrica surja para substituí-lo. Novas ideias surgem todos os dias, de maneiras de melhorar os modelos existentes, até a criação de novos. Anotações mais precisas também ajudariam no desenvolvimento dos modelos, embora o processo de anotação seja complexo por si só. Espera-se que este trabalho possa servir como uma introdução para a área, para novos pesquisadores, e uma fonte de ideias, para os existentes.

Por fim, algumas reflexões sobre aprendizado de máquina, no geral. Um grande problema dessa área é que é relativamente difícil entrar nela. O treinamento, em particular, de modelos que envolvem redes neurais, como os que foram apresentados aqui, praticamente requer uma GPU de alto desempenho, e essas GPUs não são baratas. Felizmente, nos anos recentes, essas tecnologias estão se tornando mais acessíveis para o público em geral, com serviços como o Google Colab. Talvez com os avanços recentes na IA, como o ChatGPT e o DALL·E, uma nova onda de pesquisadores surja, não só em visão computacional, mas em todas as áreas da IA. Será interessante ver onde isso nos levará. Como o mundo irá mudar com a IA.

REFERÊNCIAS

ARNAB, A.; TORR, P. H. S. Pixelwise instance segmentation with a dynamically instantiated network. **CoRR**, abs/1704.02386, 2017. Disponível em: http://arxiv.org/abs/1704.02386.

BENENSON, R.; POPOV, S.; FERRARI, V. Large-scale interactive object segmentation with human annotators. In: **CVPR**. [*S.l.*: *s.n.*], 2019.

BOLYA, D. *et al.* **YOLACT++: Better Real-time Instance Segmentation**. 2019.

BOLYA, D. *et al.* YOLACT: real-time instance segmentation. **CoRR**, abs/1904.02689, 2019. Disponível em: http://arxiv.org/abs/1904.02689.

BRABANDERE, B. D.; NEVEN, D.; GOOL, L. V. Semantic instance segmentation with a discriminative loss function. **CoRR**, abs/1708.02551, 2017. Disponível em: http://arxiv.org/abs/1708.02551.

CAI, Z.; VASCONCELOS, N. Cascade R-CNN: delving into high quality object detection. **CoRR**, abs/1712.00726, 2017. Disponível em: http://arxiv.org/abs/1712.00726.

Carreira, J.; Sminchisescu, C. Cpmc: Automatic object segmentation using constrained parametric min-cuts. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 34, n. 7, p. 1312–1328, 2012.

CHEN, K. *et al.* Hybrid task cascade for instance segmentation. **CoRR**, abs/1901.07518, 2019. Disponível em: http://arxiv.org/abs/1901.07518.

CHEN, L. *et al.* Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. **CoRR**, abs/1606.00915, 2016. Disponível em: http://arxiv.org/abs/1606.00915.

CHEN, L.-C. *et al.* Semantic image segmentation with deep convolutional nets and fully connected crfs. **arXiv preprint arXiv:1412.7062**, 2014.

CHEN, Y.-T.; LIU, X.; YANG, M.-H. Multi-instance object segmentation with occlusion handling. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [*S.l.*: *s.n.*], 2015. p. 3470–3478.

COPPIN, B. **Artificial Intelligence Illuminated**. USA: Jones and Bartlett Publishers, Inc., 2004. ISBN 0763732303.

DAI, J. *et al.* Instance-sensitive fully convolutional networks. **CoRR**, abs/1603.08678, 2016. Disponível em: http://arxiv.org/abs/1603.08678.

DAI, J.; HE, K.; SUN, J. Instance-aware semantic segmentation via multi-task network cascades. **CoRR**, abs/1512.04412, 2015. Disponível em: http://arxiv.org/abs/1512.04412.

DAI, J. *et al.* R-FCN: object detection via region-based fully convolutional networks. **CoRR**, abs/1605.06409, 2016. Disponível em: http://arxiv.org/abs/1605.06409.

DAVIS, J.; GOADRICH, M. The relationship between precision-recall and roc curves. In: **Proceedings of the 23rd International Conference on Machine Learning**. New York, NY, USA: Association for Computing Machinery, 2006. (ICML '06), p. 233–240. ISBN 1595933832. Disponível em: https://doi.org/10.1145/1143844.1143874.

GIRSHICK, R. Fast r-cnn. In: **The IEEE International Conference on Computer Vision (ICCV)**. [*S.l.*: *s.n.*], 2015.

GIRSHICK, R. B. *et al.* Rich feature hierarchies for accurate object detection and semantic segmentation. **CoRR**, abs/1311.2524, 2013. Disponível em: http://arxiv.org/abs/1311.2524.

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing (3rd Edition)**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN 013168728X.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [*S.l.*]: MIT Press, 2016. http://www.deeplearningbook.org.

HARIHARAN, B. *et al.* Simultaneous detection and segmentation. **CoRR**, abs/1407.1808, 2014. Disponível em: http://arxiv.org/abs/1407.1808.

HE, K. *et al.* Mask R-CNN. **CoRR**, abs/1703.06870, 2017. Disponível em: http://arxiv.org/abs/1703.06870.

JENA, R. *et al.* **Beyond mAP: Towards better evaluation of instance segmentation**. 2023.

KUZNETSOVA, A. *et al.* The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. **IJCV**, 2020.

LECUN, Y. *et al.* Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, Taipei, Taiwan, v. 86, n. 11, p. 2278–2324, 1998.

LEE, Y.; PARK, J. **CenterMask : Real-Time Anchor-Free Instance Segmentation**. 2019.

LI, Y. *et al.* Fully convolutional instance-aware semantic segmentation. **CoRR**, abs/1611.07709, 2016. Disponível em: http://arxiv.org/abs/1611.07709.

Liang, X. *et al.* Proposal-free network for instance-level object segmentation. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 40, n. 12, p. 2978–2991, 2018.

LIN, T. *et al.* Feature pyramid networks for object detection. **CoRR**, abs/1612.03144, 2016. Disponível em: http://arxiv.org/abs/1612.03144.

LIN, T. *et al.* Focal loss for dense object detection. **CoRR**, abs/1708.02002, 2017. Disponível em: http://arxiv.org/abs/1708.02002.

LIN, T. *et al.* Microsoft COCO: common objects in context. **CoRR**, abs/1405.0312, 2014. Disponível em: http://arxiv.org/abs/1405.0312.

LIU, S. *et al.* Path aggregation network for instance segmentation. **CoRR**, abs/1803.01534, 2018. Disponível em: http://arxiv.org/abs/1803.01534.

LIU, W. *et al.* SSD: single shot multibox detector. **CoRR**, abs/1512.02325, 2015. Disponível em: http://arxiv.org/abs/1512.02325.

LONG, J.; SHELHAMER, E.; DARRELL, T. Fully convolutional networks for semantic segmentation. **CoRR**, abs/1411.4038, 2014. Disponível em: http://arxiv.org/abs/1411.4038.

PARK, E.; BERG, A. C. Learning to decompose for object detection and instance segmentation. **CoRR**, abs/1511.06449, 2015. Disponível em: http://arxiv.org/abs/1511.06449.

PINTO, J. **Masknet: An Instance Segmentation Algorithm**. 2017. Tese (Doutorado) — Chalmers University of Technology, 2017. Disponível em: http://publications.lib.chalmers.se/records/fulltext/250417/250417.pdf.

REDMON, J. *et al.* You only look once: Unified, real-time object detection. **CoRR**, abs/1506.02640, 2015. Disponível em: http://arxiv.org/abs/1506.02640.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. **CoRR**, abs/1804.02767, 2018. Disponível em: http://arxiv.org/abs/1804.02767.

REN, S. *et al.* Faster r-cnn: Towards real-time object detection with region proposal networks. In: CORTES, C. *et al.* (Ed.). **Advances in Neural Information Processing Systems 28**. Curran Associates, Inc., 2015. p. 91–99. Disponível em: http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf.

REZATOFIGHI, S. H. *et al.* Generalized intersection over union: A metric and A loss for bounding box regression. **CoRR**, abs/1902.09630, 2019. Disponível em: http://arxiv.org/abs/1902.09630.

ROMERA-PAREDES, B.; TORR, P. H. S. Recurrent instance segmentation. **CoRR**, abs/1511.08250, 2015. Disponível em: http://arxiv.org/abs/1511.08250.

RONNEBERGER, O.; FISCHER, P.; BROX, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. **CoRR**, abs/1505.04597, 2015. Disponível em: http://arxiv.org/abs/1505.04597.

RUSSAKOVSKY, O. *et al.* Imagenet large scale visual recognition challenge. **CoRR**, abs/1409.0575, 2014. Disponível em: http://arxiv.org/abs/1409.0575.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence: a modern approach**. [*S.l.*]: Malaysia; Pearson Education Limited„ 2016.

SZEGEDY, C.; TOSHEV, A.; ERHAN, D. Deep neural networks for object detection. In: **Advances in neural information processing systems**. [*S.l.*: *s.n.*], 2013. p. 2553–2561.

SZELISKI, R. **Computer Vision: Algorithms and Applications**. 1st. ed. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN 1848829345, 9781848829343.

TIAN, Z. *et al.* FCOS: fully convolutional one-stage object detection. **CoRR**, abs/1904.01355, 2019. Disponível em: http://arxiv.org/abs/1904.01355.

UIJLINGS, J. R. *et al.* Selective search for object recognition. **International journal of computer vision**, Springer, v. 104, n. 2, p. 154–171, 2013.

WANG, X. *et al.* **SOLO: Segmenting Objects by Locations**. 2019.

WANG, X. *et al.* **SOLOv2: Dynamic, Faster and Stronger**. 2020.

XU, Y.-S. *et al.* Dynamic Video Segmentation Network. **CoRR**, abs/1804.00931, 2018. Disponível em: http://arxiv.org/abs/1804.00931.