SSS: Single Shot Segmenter

Daniel Bolya Fanyi Xiao Yong Jae Lee
University of California, Davis
{dbolya, fyxiao, yongjaelee}@ucdavis.edu

Abstract

This is a working draft of research that is currently in progress. While the underlying concepts here will stay the same, the specifics are subject to change.

1. Introduction

Over the past few years, the vision community has made great strides in the task of instance segmentation, in part due to recent advances drawing powerful parallels from the well-established domain of object detection. State-of-the-art instance segmenters like Mask R-CNN [2] and FCIS [4] directly build off of advances in object detection like Faster R-CNN [12] and FPN [5]. However, these methods focus primarily on performance over speed, leaving the scene devoid of instance segmentation parallels to real-time object detectors like SSD [7] and YOLO [10, 11]. In this work, our goal is to fill that gap with a fast, one-stage instance segmenter in the same way that SSD and YOLO fill that gap for object detection.

However, unlike with SSD or YOLO, this is not as simple as taking a more complex two stage approach and stripping it down to only what is necessary. For instance, while Faster R-CNN predicts perfectly good bounding boxes in the first stage and only uses the second stage to refine them, Mask R-CNN has to wait until the second stage to even begin producing masks. This is because mask production is inherently more complex than box prediction, requiring a deep understanding of the spacial relationship between features. Thus, a one-stage method based off of Mask R-CNN would at the very least require repooling to preserve these spacial relationships, which, given the number of anchors in a one-stage model, would add a significant computational overhead.

In our Single Shot Segmenter, we bypass this issue by breaking the problem into two parts: generating a dictionary of prototype masks for the entire image, and predicting a set of coefficients for each instance indicating which prototype masks to combine into a final mask. By doing this, we can preserve the spacial relationship between features in the prototype masks, while also allowing us to leverage the semantic instance-level information in the layers that predict class and box regression coefficients. Moreover, since these two operations can be done completely in parallel, this adds virtually no overhead to the existing SSD backbone we base our model off of.

Another, perhaps even simpler approach applied to instance segmentation in the past would be to first mask out the foreground in an image, and then partition that foreground mask into individual instances. This has the main benefit of forground masks being incredibly easy for an FCN [8] to produce. Yet, the complexity of instance detection makes it challenging to create a robust partitioning algorithm to complete the approach. Despite this, we show that *this partitioning paradigm arises naturally from our model without explicit supervision*. That is, our approach as described above naturally induces a robust and fast foreground partitioning model.

This result is somewhat unexpected because, as [4] points out, fully convolutional networks are not translation variant. This then poses a problem, as instance segmentation (and this partitioning approach) fundamentally requires models to treat pixels differently depending on their location in the image.

However, we observe that this analysis misses a crutial point: modern backbone architectures like Resnet [3] make heavy use of padding to keep the output of conv layers the same size as the input. We posit that the introduction of this consistent border of 0's around every feature map introduces a natural coordinate system within Resnet. Namely, repeated convolution can propagate these padding pixels away from the borders, letting the network know that a pixel is some set distance away from a border. In our model, this results in a very much translation-variant set of partitions, despite its fully-convolutional implementation.

2. Related Work

In progress

3. Single Shot Segmenter

We build our approach as an extension to SSD [7]. We choose SSD primarily because it's an accessible and fast object detector, but our method is general enough to be implemented in any anchor-based object detector sufficiently similar to SSD, including YOLO [10, 11] and even two-stage approaches like Faster R-CNN [12]. However, because speed is our main focus, this paper will only cover an SSD-based model.

3.1. Producing Masks

Our goal is to add a mask branch to an existing one-stage object detection model in the same vein as Mask R-CNN [2]. However, SSD and other sliding-window object detectors produce boxes for multiple thousands of anchors, and it would be infeasible to predict a detailed mask for each one.

Moreover, masks are spatially coherent—that is, pixels close to each other have a similar probability of being part of the mask. While a convolutional (*conv*) layer very naturally takes advantage of this coherence, a fully-connected (*fc*) layer does not. That poses a problem, since these object detectors produce class and box coefficients for each anchor as an output of an *fc* layer. As we show in experiments, simply adding masks to a one-stage model as *fc* outputs is very much insufficient. Two stage approaches like Mask R-CNN have gotten around this problem by using a repooling-type operation, but that would be slow for all the anchors in a one-stage model.

This motivates us to break the problem into two parts, taking into account that *fc* layers are good at producing semantic vectors, while *conv* layers are good at producing spatially coherent masks. Hence, instead of predicting individual masks for each anchor, we predict coefficients that instruct the network how construct a mask from a dictionary of image-level prototypes. Because the coefficients now have some semantic (not spacial) meaning, *fc* layers can very naturally encode them, and because the dictionary of prototypes is per image, not per instance, we can use an FCN [8] to produce them all at once.

This is also much faster than producing a mask for each anchor, because we can wait to construct the masks until after non-maximal suppression (NMS), considerably reducing the number of masks that need to be constructed. Thus, our approach can be decomposed into two entirely parallel tasks: coefficient prediction and prototype generation, as well as a final combine step for the few detections surviving NMS.

Mask Coefficients: Typical anchor-based object detectors have two branches: one to predict class confidences, and the other to predict bounding box regression coefficients. For mask coefficient prediction, we just add a third branch that predicts these mask coefficients.

	Total	S	M	L
	Post-Augmentation			
SSD	43.7%	1.43%	34.2%	92.3%
Ours	57.3%	10.7%	62.9%	94.5%

Table 1. The recall for training boxes that have an IoU overlap of over 0.5 with any anchor. Here we compare the anchors described in the original SSD paper [7] adapted to an image size of 550×550 (7532 anchors total) to our anchors optimized on the post-augmentation set (7558 anchors total). Note that we added back a small number of anchors for some redundancy.

That is, object detectors usually predict 4 bounding box regression coefficients and c class confidences for a total of $4 + c$ outputs per anchor. To predict our mask coefficients, we simply add an extra branch that predicts k coefficients, one for each prototype mask. Thus, all this requires is to increase the number of outputed coefficients from $4 + c$ to $4 + c + k$.

Prototype Generation: The prototype generation branch predicts a set of k prototype masks for the full image. Because prototype generation doesn't depend on the number of anchors, we can implement this as a fully convolutional subnetwork that takes input from some backbone layer and outputs a dictionary of masks.

Constructing Masks: The prototypes and mask coefficients form two separate components, which when combined give the final masks for an instance. In this work, we combine prototypes and their corresponding coefficients by simply multiplying each prototype by its coefficient, and summing the resulting prototypes. This can be implemented efficiently using a single matrix multiplication:

$$M = PC^T \tag{1}$$

where P is an $h \times w \times k$ matrix of prototype masks and C is a $n \times k$ matrix of mask coefficients for n instances surviving NMS and score thresholding.

Note that other, more complicated combination steps are possible. In fact, the assemble step in FCIS [4] could be viewed as just a different (albeit much more complicated) combine step of prototypes and box regression coefficients. However, for this paper we keep it simple (and fast) with a basic linear combination.

3.2. Optimizing Anchors

Good anchors are imperative for a one-stage approach. In a two-stage model, there is an opportunity to correct any mistakes made during the first stage, and the first stage can also let go some false positives that would be filtered out in the second stage. However, a one-stage model only has one shot. Thus, designing anchors becomes a critical issue.

Previous works such as YOLOv2 [10] and DSSD [1] have attempted to improve their anchors by running k-means on the dimensions and the aspect ratios of the boxes in the training data respectively. While this is better than hand-engineering the anchors, the latter approach doesn't optimize the scales of the anchor boxes, while neither approach take into account the stride between each anchor.

Instead, we propose to optimize the anchor parameters directly. What we would like is for each object to have an anchor close to it, specifically with IoU overlap over 0.5 (since that's the threshold we use for training). Thus we can define an objective function by the anchor recall—the percentage of training boxes that have an overlap 0.5 or greater with at least one anchor box. We can then optimize this objective function directly, using the anchor scales and aspect ratios as parameters.

Because this objective function isn't differentiable, we use Powell's method [9] to optimize it. For smoother training, we run Powell's method on the scales and aspect ratios in turns, alternating between them until the total recall converges. Note that in order to prevent overfitting (and for quick and painless optimization), we only train these parameters on a random 10k subset of all 600k boxes in COCO's train2014 [6]. We then do this thrice and select the anchors that obtained the best recall on the entire training set.

This whole process only takes 5 minutes but increases the recall on COCO considerably (see 1). However, note that the goal of this process isn't just to get anchors closer to ground truth, but also to smooth the training process. Thus, we have found it better to optimize the anchor boxes with training data after it has gone through data augmentation. SSD has very harsh data augmentation that reduces the size of a lot of the training examples, so these post-augmentation optimized anchors better help the model during training.

3.3. Faster Non Maximal Suppression

Because anchor-based object detectors produce so many detections, they need to apply Non Maximal Suppression (NMS) to get rid of redundant ones. This is traditionally implemented by first sorting the detected boxes descending by confidence, and then for each detection removing all the detections below it that have an IoU overlap greater than some threshold. For the sake of speed, we consider some alternative ways of doing NMS:

Cross-Class NMS: Traditional NMS is run on each class individually. This is fine in cases where there are not many classes, but the computation time quickly adds up for datasets with a lot of classes. SSD and YOLOv2 both report timing information on VOC Pascal, which has 20 classes. COCO, on the other hand, has 80 classes, a full 4 times that of VOC Pascal. Thus on COCO, this per-class NMS cuts a significant margin into the total running time.

To remedy this, we run our model with cross-class NMS. That is, instead of running NMS for each class individually, first choose the maximum class confidence for each detection, and run NMS using that. Thus, cross-class NMS's computation time doesn't scale with the number of classes like traditional NMS.

Fast NMS: In traditional NMS, if a detection gets removed by a detection with a higher score, it can't then go on to remove other detections with a lower score than it. However, if we relax this constraint, we can implement a very fast vectorized version of NMS:

```
_, idx = scores.sort(0, descending=True)
idx = idx[:top_k]
boxes = boxes[idx]

iou = pairwise_iou(boxes, boxes)
iou.triu_(diagonal=1)

idx_keep = idx[iou.max(dim=0) <=
    iou_threshold]

return boxes[idx_keep]
```

This has the effect of removing slightly too many boxes, but offers a very nice FPS improvement.

4. Results

In progress

5. Conclusion

In progress

References

[1] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg. Dssd: Deconvolutional single shot detector. *arXiv:1701.06659*, 2017. 3

[2] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *ICCV*, 2017. 1, 2

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1

[4] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei. Fully convolutional instance-aware semantic segmentation. In *CVPR*, 2017. 1, 2

[5] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017. 1

[6] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 3

[7] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016. 1, 2

[8] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 1, 2

[9] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162, 1964. 3

[10] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In *CVPR*, 2017. 1, 2, 3

[11] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv:1804.02767*, 2018. 1, 2

[12] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 1, 2