YOLACT++

Better Real-Time Instance Segmentation
TL;DR Version

ABSTRACT

YOLACT is:

- fully-convolutional model
- evaluated on a single Titan Xp
- trained on a single GPU

The improvements stem from breaking instance segmentation into two **parallel** tasks:

1. generating a set of prototype masks (a dictionary of non-local prototype masks over the entire image)
2. predicting per-instance mask coefficients (a set of linear combination coefficients per instance)

The results (instance masks) are obtained by **linearly combining** the prototypes with the mask coefficients. For each instance, linearly combine the prototypes using the corresponding coefficients and then crop with a predicted bounding box.

Even though it is fully convolutional, YOLACT learns to **localize instances** in a translation variant manner, as a result of the **prototype's emergent behaviour**.

The network uses **Fast NMS** as opposed to regular **Non-Max Suppression (NMS)**. It has a **12ms** speed gain with only a marginal performance penalty.

Improvements implemented in YOLACT++ over YOLACT are:

- deformable convolutions into the backbone network
- better anchor scales and aspect ratios to improve the prediction head
- novel fast mask re-scoring branch

1. INTRODUCTION

The speedup from YOLACT mainly comes from forgoing an explicit localization step thus making it faster than two-stage detectors, but also from splitting the segmentation into two parallel tasks.

The localization is **learned implicitly through the emergent behaviour** of the generated prototypes.

By segmenting in this way the network learns how to **localize instance masks** on its own, where visually, spatially and semantically similar instances appear different in the prototypes.

Because the number of prototype masks is independent of the number of categories, YOLACT learns a distributed representation. Each instance is segmented with a combination of prototypes that are shared across categories.

Therefore the **prototype space** contains prototypes that have the following functions:

- spatially partition the image
- localize instances
- detect instance contours
- encode position-sensitive directional maps
- combine some of the functions mentioned above

The advantages of **deformable convolutions** added into the backbone network are:

- provide more flexible feature sampling
- strengthens the capability of handling instances with different scales, aspect ratios and rotations

The **prediction heads** are optimized using better anchor scales and aspect ratio choices for larger object recall.

The **novel fast mask re-scoring branch** results in a decent performance boost with only marginal speed overhead.

2. RELATED WORK

Mask-RCNN

- two-stage instance segmentation method:
    1. generates candidate regions-of-interest (ROIs)
    2. classifies and segments the generated ROIs
- follow-ups on Mask-RCNN try to improve:
    - accuracy by enriching the FPN network
    - addressing the incompatibility between a mask's confidence score and it's localization accuracy
- overhead introduced by the second stage prevents these from achieving real-time speeds (30 fps or more)

One-Stage Instance Segmentation

- generates position-sensitive maps combined with position-sensitive pooling
- combines semantic segmentation logits with direction prediction logits
- are slow because they still require re-pooling or other non-trivial computations (e.g mask voting)

Real-time Instance segmentation

There are a few alternatives but they lack accuracy

- Straight to Shapes
    - 30 fps - Pascal SBD 2012
    - 10.9 fps - CityScapes

- Box2Pix
  - 35 fps - KITTI
- Mask R-CNN
  - 13.5 fps - COCO

Prototypes

Prototypes such as textons and visual words are mostly used to represent features, as opposed to YOLACT, which uses them to assemble the masks for instance segmentation.

Therefore YOLACT is learning prototypes that are specific for each image rather than global prototypes shared over the entire dataset.

3. YOLACT

The goal is to add a mask branch to an existing one-stage object detection model but without adding an explicit feature localization step.

More details on the two-pronged process:

- the first branch uses an **FCN** to produce a set of image-sized **prototype masks** that does not depend on any one instance.
- The second branch adds an extra head to the object detection branch to predict a vector of **mask coefficients** for each anchor that encodes an instance's representation in the prototype space
- for each instance that survives **NMS** a mask is constructed for it by linearly combining the work of these two branches.
- the linear combination can be implemented using a simple **matrix multiplication** that is accelerated using the GPU.

Rationale

The reason for this approach of instance segmentation is that masks are spatially coherent (i.e. pixels close to each other are likely to be part of the same instance). While convolutional layers naturally take advantage of the spatial coherence, fully connected layers do not.

Since the one-stage detectors use fully connected layers to produce class and box coefficients for each anchor, the problem stated above must be addressed.

Mask R-CNN uses a localization step (i.e RoI-Align) to preserve the spatial coherence of the features while also allowing the mask to be a conv layer output. This involves waiting for the first stage RPN (Region Proposal Network) to propose localization candidates, therefore introducing a significant overhead to the inference process.

Breaking the problem in two **parallel** parts makes sense because:

- fc layers are good at producing semantic vectors => mask coefficients
- conv layers are good at producing spatially coherent masks => prototype masks

This approach preserves both the spatial coherence and the speed of the backbone detector.

3.1 Prototype Generation

The prototype generation branch (protonet) predicts a set of *k* **prototype masks** for the entire image.

It is implemented as an FCN whose last layer has *k* channels (one for each prototype) and it is attached to a backbone **feature layer**.

The prototype layer has no explicit loss on the prototypes. All supervision comes from the final mask loss, after the assembly part (the matrix multiplication with the coefficients).

Important design choices:

- taking protonet from deeper backbone features produces more robust masks
- higher resolution prototypes result in both higher quality masks and better performance on smaller objects
- the protonet's output is unbounded, allowing the network to produce large activations for prototypes that have higher confidence levels.
- protonet is followed by ReLU

3.2 Mask Coefficients

The prediction head is formed out of three branches (one extra compared to typical anchor-based object detectors):

- one branch to predict *c* **class confidences**
- another branch to predict **4 bounding box regressors**
- a third branch that predicts *k* **mask coefficients** (one for each prototype)

A *tanh* function is applied over the k mask coefficients for two reasons:

- it allows for prototype subtraction from the final mask
- produces more stable outputs (compared to **not** adding a **nonlinearity**)

3.3 Mask Assembly

Final masks are produced according to $M = \sigma(PC^T)$. Where :

- P is an $h \times \omega \times k$ matrix of prototype masks
- C is an $n \times k$ matrix of mask coefficients for n instances surviving **NMS**
- σ is a sigmoid nonlinearity

This basic linear combination allows the mask assembly process to be fast.

Losses

Three losses are used to train the model:

- classification loss: L_{cls} with weight 1
- box regression loss: L_{box} with weight 1.5
- mask loss: L_{mask} with weight 6.125

L_{cls} and L_{box} are defined in the same way as in SSD (Single Shot multibox Detector).

Mask loss is computed as a pixel-wise binary cross entropy betweem assembled masks M and the ground truth masks M_{gt}: L_{mask} = BCE(M, M_{gt}).

Cropping Masks

During evaluation, the final masks are cropped with the predicted bounding box. Specifically, zero is assigned to pixels outside of the box region.

During training the mask is cropped with the ground truth bounding box, and L_{mask} is divided by the ground truth box area to preserve small objects in the prototypes.

3.4 Emergent Behaviour

Because FCN's are translation invariant some models have an explicit translation variance such as directional maps or position-sensitive re-pooling.

The only translation variance added in YOLACT is cropping the final mask with the predicted bounding box. YOLACT learns how to localize instances on its own via different activations in its prototypes. Because the backbone (ResNet) is inherently translation variant the property also translates to YOLACT. Many generated prototypes are observed to activate only on certain partitions of the image; they only activate on objects that are located on one side of an implicitly learned boundary. Because of this, the network can distinguish between different (even overlapping) instances of the same semantic class.

Being learned objects, the prototypes can be compressed by combining the functionality of multiple prototypes into one. The mask coefficient branch can learn which situation calls for which type of functionality.

This behaviour explains why the model does not degrade in performance when k is as low as 32 but also why increasing the number of prototypes is ineffective (also predicting coefficients proves to be a difficult task).

4. BACKBONE DETECTOR

Speed is prioritized as well as feature richness since predicting coefficients is a difficult task that requires good features to do well.

YOLACT Detector

Default feature backbone is ResNet-101 with FPN with base image size of *550 X 550*

The aspect ratio of the image is not preserved in order to get consistent evaluation times per image.

The FPN has slight alterations, similar to RetinaNet, in the way the FPN is produced but also regarding anchor scales and aspect ratios.

Compared to RetinaNet the prediction head in YOLACT is lighter and faster.

To **train** the box regressor and **encode** box regression coordinates, *smooth-*L_1 loss is applied (similar to SSD).

Softmax Cross-Entropy with *c* positive labels and 1 background label is used to train class prediction.

Train examples are selected using OHEM with 3:1 neg:pos ratio.

Even though *focal loss* is used in RetinaNet, it is not viable to use it here.

5. OTHER IMPROVEMENTS

This either increases the speed with little performance impact or increases performance with no speed penalty.

5.1 Fast NMS

Duplicate predictions are suppressed using NMS, after the assembly step.

In previous works, NMS is performed sequentially which introduces significant overhead.

Fast NMS fixes this issue by enabling every instance to be kept or discarded not sequentially, but in parallel. This is done by allowing already-removed detections to suppress other detections.

Fast NMS can be implemented entirely using standard GPU-accelerated matrix operations.

Being a more relaxed version of NMS, Fast NMS tends to remove slightly too many boxes. Yet this trade-off is done in favour of speed increase.

Fast NMS is 11.8 ms faster than standard NMS Cython implementation while only reducing performance by 0.1 mAP.

5.2 Semantic Segmentation Loss

Applying extra losses during train time, which are not executed during testing is a way to increase feature richness without sacrificing speed.

Thus *semantic segmentation loss* is applied on the feature space using layers that are only evaluated during training.

To create predictions during training a 1 X 1 *conv* layer with *c* output channels is attached to the largest feature map in the backbone (P3 in Fig. 2).

Since each pixel can be assigned to more than one class sigmoid with *c* channels is used instead of softmax with *c+1* channels.

This loss if given a eight of 1 and results in a +0.4 mAP boost.

6. YOLACT++

YOLACT so far seems to be viable for real-time applications and only consumes ~1500 MB of VRAM even when using a ResNet-101 backbone.

Several performance improvements are explored with YOLACT++ such as:

- an efficient and fast mask re-scoring network (re-ranks the mask according to their mask quality)
- improve backbone with deformable convolutions (makes the feature sampling align better with instances)
- better choices for the detection anchors to increase recall

6.1 Fast Mask Re-Scoring Network

Mask Scoring R-CNN indicates that there is a discrepancy in the model's classification confidence and the quality of the predicted masks.

To better correlate class confidence with mask quality Mask Scoring R-CNN learns to regress the predicted mask to its IoU with ground-truth.

Fast Mask Re-Scoring Network is following the same principle.

It is implemented as a 6-Layer Fully Convolutional Network (FCN) with one ReLU per conv layer and a final global pooling layer. It takes as input YOLACT's cropped mask prediction (before thresholding) and outputs the mask IoU for the category predicted by the classification head and the corresponding class confidence.

Differences between Fast Mask Re-Scoring Network and Mask Scoring R-CNN:

1. input is only the mask at full image size (with zeros outside the predicted box region)
2. there are no *fully-connected* layers

Speed considerations:

- adding Fast Mask Re-Scoring to YOLACT with ResNet-101 has a 1.2 ms penalty => fps drop from 34.4 to 33
- adding Mask Scoring R-CNN into YOLACT with ResNet-101 has 28 ms penalty => fps drop from 34.4 to 17.5

The speed difference mainly comes from MS R-CNN's usage of RoI align operation, its *fc* layers and the feature concatenation in the input.

6.2 Deformable Convolution with Intervals

They replace rigid, grid sampling used in conventional conv nets with free-form sampling.

YOLACT replaces the 3x3 convolution layer in each ResNet block with a 3x3 deformable convolution layer from C3 to C5 (in Fig. 2).

Speed and performance considerations:

- +1.8 mAP gain
- 8 ms speed overhead

Boost is due to:

1. DCN strengthens the ability to handle instances with different scales, rotations and aspect ratios by aligning properly to the target instances
2. YOLACT, as a single-shot method, does not have a re-sampling process.

To further speed up the inference process less deformable convolutions in various configurations can be used:

1. in the last 10 ResNet blocks
2. in the last 13 ResNet blocks
3. in the last 3 ResNet stages with an interval of 3 (i.e. skipping 2 ResNet blocks in between => 11 deformable layers)
4. in the last 3 ResNet stages with an interval of 4 (i.e. skipping 3 ResNet blocks in between => 8 deformable layers)

The best choice seems to be the interval of 3 which cuts down the speed overhead by 5.2 ms to 2.8 ms and only has a 0.2 mAP drop compared to not having an interval.

6.3 Optimized Prediction Head

Being based off an anchor-based backbone, choosing the right anchor configuration (scales, aspect ratios) is important.

Two variations are tried:

1. keeping the scales unchanged while increasing the aspect ratio from [1, 1/2, 2] to [1, 1/2, 2, 1/3, 3]
2. keeping the aspect ratios unchanged while increasing the scales per FPN level ([1x, $2^{1/3}$x, $2^{2/3}$x])

Using the second configurations seems to yield the best speed vs performance trade-off.

7. RESULTS

Results are reported on MS COCO and Pascal 2012 SBD.

MSCOCO:

- train: `train2017`
- evaluate: `val2017` and `test-dev`

7.1 Implementation Details

All models are trained with batch size 8, on one GPU, using ImageNet pre-trained weights. This is a sufficient batch size to use batch norm on, so the pre-trained batch norm is left unfrozen and not extra batch norm layers are added.

ImageNet training:

- SGD for 800k iterations
- initial learning rate: 10^{-3}
  - divided by 10 at 280k, 600k, 700k and 750k iterations
- weight decay: $5x10^{-4}$

- momentum: 0.9
- all data augmentations used in SSD

Pascal training:

- SGD for 120k iterations
- initial learning rate: 10^{-3}
  - divided by 10 at 60k and 100k iterations
- weight decay: $5x10^{-4}$
- momentum: 0.9
- all data augmentations used in SSD
- multiply the anchor scales by 4/3 (because objects tend to be larger)

Training time on Titan Xp:

- 4-6 days on COCO
- less than 1 day on Pascal

7.2 Mask Results

The tests are conducted with no test-time augmentations.

All the times are calculated on a single Titan Xp.

YOLACT-550 shows competitive instance segmentation performance:

- 3.8x faster than previous SOTA
- IoU threshold results:
  - 50%: gap between YOLACT-550 and Mask R-CNN is 9.5 AP
  - 75%: gap between YOLACT-550 and Mask R-CNN is 6.6 AP
  - 95%: gap between YOLACT-550 and Mask R-CNN is 0.3 AP

Further evaluation with alternate configurations shows that lowering the image size results in a large decrease in performance, demonstrating that instance segmentation naturally demands larger images.

If higher speeds are preferable it is suggested that a different backbone should be used (ResNet-50, DarkNet-53) instead of lowering the image size. These configurations perform much better than YOLACT-400, while only being slightly slower.

7.3 Mask Quality

Masks are of higher quality than Mask R-CNN and FCIS because:

- they are created directly from the original features with a final size of 138x138
- there is no re-pooling to transform and potentially misalign the features

7.4 Temporal Stability

Training is only done on static images. There is no temporal smoothing.

The masks generated by YOLACT are more temporally stable than others because of the higher quality and because they do not depend on any region proposal.

Even if the model predicts different boxes across frames, the prototypes are not affected, yielding much more temporally stable masks.

7.6 Box Results

YOLACT achieves similar results to YOLOv3 at similar speeds while not using additional improvements shown in YOLOv2 or YOLOv3 (multi-scale training, anchor boxes).

Improvements come mostly from using an FPN and training with masks.

7.7 YOLACT++ Improvements

The optimized anchor choice directly improves the recall of box predictions and boosts the backbone detector.

The deformable convolutions help with better feature sampling by aligning the sampling positions with the instances of interest and by handling changes in scale, rotation and aspect ratio better.

Those two improvements result in 3.4 mAP boost for ResNet-101 and 4.2 mAP boost for ResNet-50.

In addition to this, the Fast Mask Re-Scoring Network is faster than MS R-CNN by 26.8 ms yet it still improves YOLACT by 1 mAP.

8. DISCUSSION

Most errors, that bring down the overall performance score of YOLACT are mistakes in the detector: misclassification, box misalignment, etc.

But there also are 2 typical errors that happen because of YOLACT's mask generation algorithm.

Localization Failure

If there are multiple objects, very close to each other, the network can fail to localize each object in its own prototype.

YOLACT++ partially addresses this by introducing more anchors that cover more scales, and by making use of deformable convolutions for better feature sampling.

Leakage

Because the masks are cropped after assembly there is no attempt to suppress noise outside of the cropped (bounding box) region.

If the bounding box is not properly identified the noise can interact with the instance mask, creating some "leakage" outside the cropped region.

Because the network doesn't need to localize far away instances (because the cropping will take care of it) some leakage also happens when objects are further apart from each other.

YOLACT++ partially mitigates these issues with a mask error down-weighting scheme, where masks exhibiting these errors will be ignored or ranked down lower than high-quality masks.

9. CONCLUSION

The key idea regarding YOLACT and YOLACT++ is to predict mask prototypes and per-instance mask coefficients in parallel and linearly combine them to form the final instance masks.

Finally, improvements to the backbone network, better anchor design and a fast mask re-scoring network showed a significant boost of YOLACT++ compared to YOLACT, while still running in real-time.