Final Team Project: Expose to research paper

Jiade Lin, Guolong Luo and Fengzhang Du

Prof: Chialing Tsai

Due: December 11th

COCO dataset

COCO is a large-scale object detection segmentation, and captioning dataset. COCO has features: Object segmentation, Recognition in context, Superpixel stuff segmentation, 330K images with over 200K labeled, and more.

In the COCO dataset, there are two types of image collections Common object categories and Non-iconic image Collection. With 80 categories with instance-level annotation, COCO is very often used by machine learning training for image segmentation. COCO has several annotation types: for object detection, keypoint detection, stuff segmentation, panoptic segmentation, densepose, and image captioning. The annotations are stored using JSON.

```
{
   "info"              : info,
   "images"            : [image],
   "annotations"       : [annotation],
   "licenses"          : [license],
}

info{
   "year"              : int,
   "version"           : str,
   "description"       : str,
   "contributor"       : str,
   "url"               : str,
   "date_created"      : datetime,
}

image{
   "id"                : int,
   "width"             : int,
   "height"            : int,
   "file_name"         : str,
   "license"           : int,
   "flickr_url"        : str,
   "coco_url"          : str,
   "date_captured"     : datetime,
}

license{
   "id"                : int,
   "name"              : str,
   "url"               : str,
}
```

Object Detection

Each object instance annotation contains a series of fields, including the category id and segmentation mask of the object. The segmentation format depends on whether the instance represents a single object or a collection of objects

```
annotation{
   "id"                : int,
   "image_id"          : int,
   "category_id"       : int,
   "segmentation"      : RLE or [polygon],
   "area"              : float,
   "bbox"              : [x,y,width,height],
   "iscrowd"           : 0 or 1,
}

categories[{
   "id"                : int,
   "name"              : str,
   "supercategory"     : str,
}]
```

Literature survey

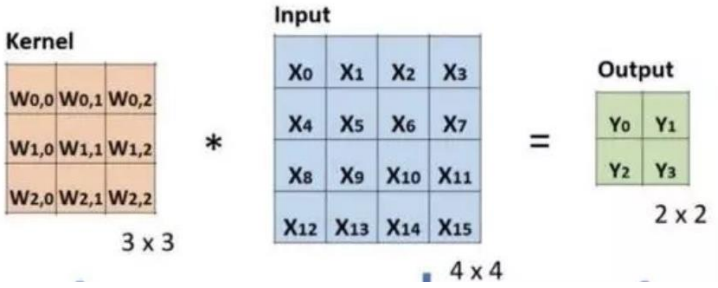
The paper we going to talk about is Smoothed Dilated Convolutions for Improved Dense Prediction by Zhengyang Wang and Shuiwang Ji. The main idea of this paper is using Smoothed Dilated convolution to improve the preformation of DCNNs with dilated convolution.

First let's take a look what is the convolution:

As we talk in class, in the simplest sentence, convolution uses a filter that goes through an input image and produces an output image. We would get different outputs for the same input image depending on the different types of filters.

Why convolution is importance for deep learning?

First it can help reduce the size of training data. As the sample show below, the size of input image is 4x4 pixels, after applying the 3x3 Kernel(filter), the result image is only 2x2.



What's more, in the real model, we are going to use convolution more than once.

For example, if our model has three convolution blocks and three 5x5 convolutions in each convolution block, we will have nine convolutions in total. For each 5x5 convolution, we reduce the image size 4x4, so totally we will reduce the image size 36x36. You may say for an image with a high number of pixels, it is relatively less, but we can also increase the filter size.

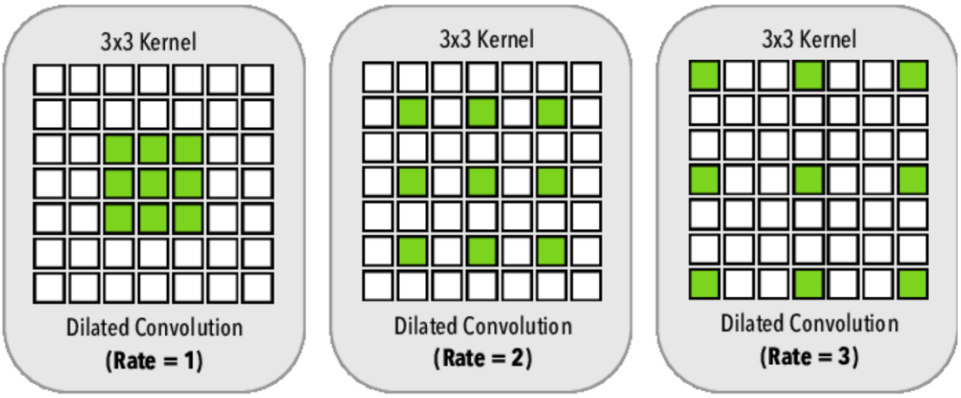
Second, more importance is by doing convolution. We can take out the important features from the image. For example, suppose we are trying to make a deep learning mode to classify clothes in the sample below. Our purpose is to let the computer learn what's a cloth, so the features like color and an input image label are no longer important. Instead, we need to pay more attention to the edge or the shape of the cloth. By doing the convolution, we can reduce the noise in the import image and let the model train faster and better:

Sample of input image

Sample of output image by convolution

What is Dilated convolution(also called Atrous convolution)?

Still, by the simplest sentence, Dilated convolution is just a special convolution whose filter has space. We use one extra parameter to control the space between points like these:



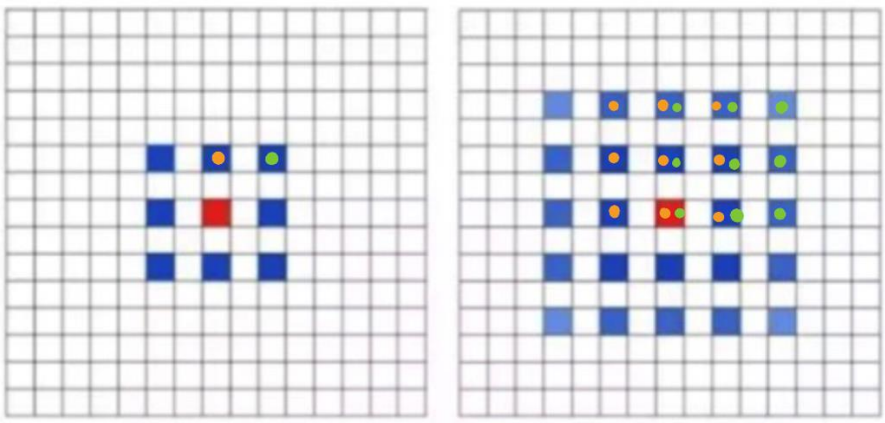
What's the benefit of Dilated Convolution?

1. Dilated convolution can generally improve the performance in semantic segmentation. The reason is with same number of pixels points, dilated convolution can get information from larger areas compare with normal convolution.
2. It doesn't influence the benefit of convolution because except adding space, the filter is as same as normal convolution so it still can reduce the noise.

What's the problem of Dilated Convolution?

The Gridding Effect is the biggest problem for dilated convolution, since we are adding space in the convolution kernel, as the sample show below, the pixels in the space area is never been use, the result is it will cause local information loss. Especially for the pixel-level dense prediction it's a huge problem.

Result after dilated convolution **original image**

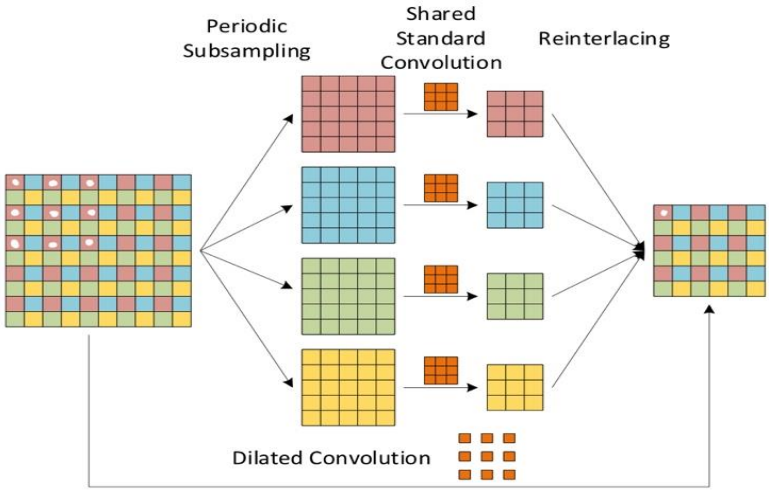


How the paper improves Dilated convolution?

In the paper, they first separate the dilated convolution into three steps:

a. periodically subsample the input image
b. doing regular convolution to these subgroup images
c. mixed the subgroups to make the output image

as the image show below, although it been separate into 3 steps the result is same as apply dilated filter (look the sample pixel I label with white point)

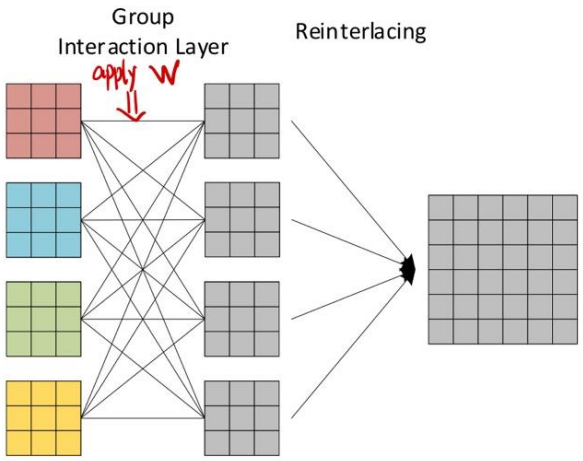


By this separate, we can clearly see the problem of dilated convolution: in the output image, the pixels next to each other have different colors, which means they come from a different subgroup of the original images. It causes the gridding, or we can say that the local features between neighbors are not united in the output image.

The paper come up with two solutions for this:

a. Smoothed Dilated Convolutions by group Interaction Layers

In this method, the author adds an extra Group Interaction Layer in the decomposition step as image show below:

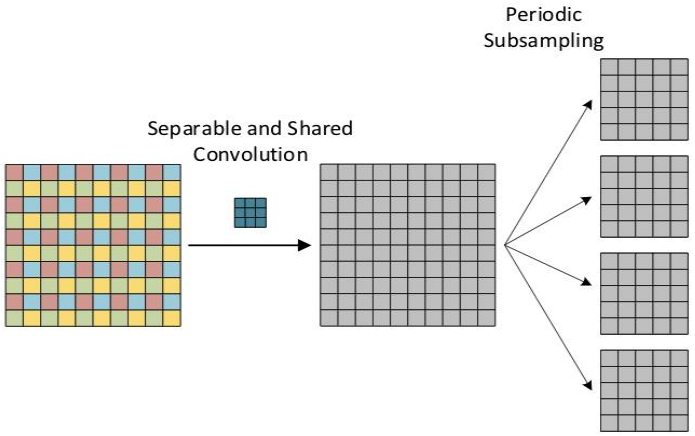


In the Group Interaction Layer, rather than directly mixing the image from subgroups, they use a Weight Matrix W to create new subgroup images and then mix them to create the final output. When the final image comes, the neighbor pixels will have more connection and will reduce the gridding problem

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1,r^d} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2,r^d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{r^d,1} & w_{r^d,2} & w_{r^d,3} & \cdots & w_{r^d,r^d} \end{bmatrix}$$

b. Smoothed Dilated Convolutions by Separable and Shared Convolution

In the last method, we add one extra layer at step 3 of dilated convolution to mix the sub-images. Here, we also try to mix the images, but we do it in the first step before the separate sub-images. As the image is shown below, the author proposes separable and shared (SS) convolution before subsampling the image. In this way, the local information can be incorporated before periodic so in the output image, pixels have more connection to nearby pixels and reduce gridding.



In my understanding, it's just like doing a smooth for the image before the sampling. And Because the original image is already smoothed so the pixels already conation information from neighbors, so after dilated Convolution, they will keep have information from neighbors, which reduces Gridding's influence.

What is the result from paper?

In the paper, the author tares the same model but with different convolution methods and the models in both ASCAL VOC 2020 dataset and MS-COCO data set to compare how the three two methods work. There is the result of the paper:

Models	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	mIoU
DeepLabv2	93.8	85.9	38.8	84.8	64.3	**79.0**	93.7	85.5	**91.7**	34.1	83.0	57.0	**86.1**	83.0	81.0	85.0	58.2	83.4	**48.2**	**87.2**	74.0	75.1
Multigrid	93.6	85.4	38.9	82.2	66.9	76.6	93.2	85.3	90.7	35.7	82.5	53.7	83.1	**84.2**	82.2	84.6	56.9	84.3	45.6	85.5	73.1	74.5
G Interact	93.7	**86.9**	**39.6**	84.1	**68.9**	76.4	**93.8**	86.2	**91.7**	**36.1**	**83.7**	55.3	85.7	84.0	82.2	84.9	**59.5**	**85.7**	46.5	85.0	73.0	**75.4**
SS Conv	**93.9**	86.7	39.5	**86.2**	68.1	77.3	**93.8**	**86.4**	91.5	35.4	83.2	**59.0**	85.2	83.6	**82.4**	**85.2**	57.3	82.1	45.8	86.1	**75.2**	**75.4**

It looks like both the two method mention in the paper does a better job comparing with DeepLabV2 and Multigrid models. But we know DeepLabV3+, which we original ask for project improve a lot, So it makes me kind of interesting how the DeepLabV3+ doing the convolution (this will be shown later after how we try to test out the result of the paper)

What is the result from my trying?

First, I am surprised by the speed of machine learning development. When I found the paper, it's in late 2018, so I thought the paper should be new enough. And the idea of the paper did give me a good understanding of deep learning. But when I try to test out their source code, I find out I am totally wrong. Only two years later, some of the libraries they use are unsupported now. If I try to run their code with the newer version of the libraries, there will be lots of bugs. I spend a lot of time trying to fix their code but still can't make it work perfectly. Even the python version they use is out of date. So, I decided to try out the papers' idea based on my understanding.

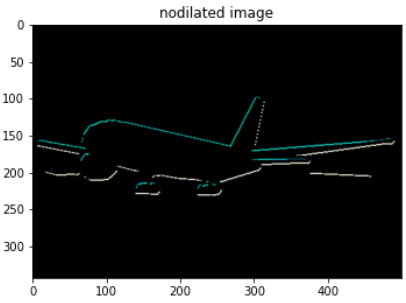
The key idea of the paper actually just applies an extra smooth step before or after the dilated convolution. And as we know, the convolution actually just apply filter to the image. So, I try out compare the different result of directly apply dilated convolution or smooth first then apply dilated convolution.

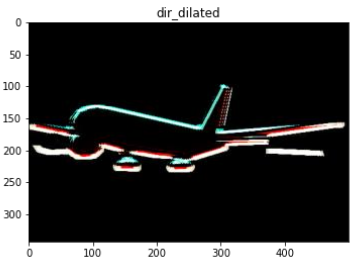
```
#dircvtly apply dilated convolution
res_dir_dilated = cv2.filter2D(img,-1,fil_Dilated)
plt.title('dir_dilated')
plt.imshow(res_dir_dilated)
plt.savefig("dir_dilated_image")
pylab.show()

#apply a smooth convolution before apply dilated convolution
res_smooth = cv2.filter2D(img,-1,fil_smooth )
res_smooth_dilated = cv2.filter2D(res_smooth,-1,fil_Dilated)
plt.title('smooth_dilated')
plt.imshow(res_smooth_dilated)
plt.savefig("smooth_dilated_image")
pylab.show()
```

There is the kernels we used, we choose an edge detection kernel, it's easy to see that dilated convolution actually just adding space to common kernel.

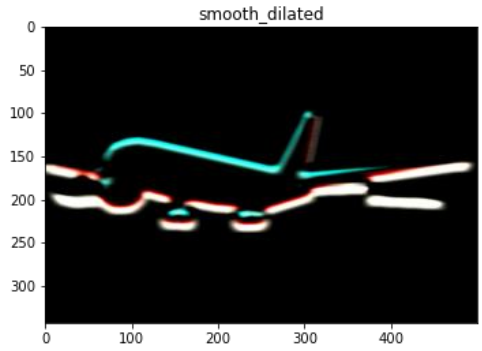
```
fil_smooth = np.ones((9,9),np.float32)/(9*9)    #smooth filter

fil_original = np.array([[ 1,1,-1],             #without dilated
                         [ 0,0,0],
                         [ -1,-1,-1]])

fil_Dilated = np.array([[ 1,0,0,0,1,0,0,0,1],   #dilated with space 3
                        [ 0,0,0,0,0,0,0,0,0],
                        [ 0,0,0,0,0,0,0,0,0],
                        [ 0,0,0,0,0,0,0,0,0],
                        [ 0,0,0,0,0,0,0,0,0],
                        [ 0,0,0,0,0,0,0,0,0],
                        [ 0,0,0,0,0,0,0,0,0],
                        [ 0,0,0,0,0,0,0,0,0],
                        [ -1,0,0,0,-1,0,0,0,-1]])
```

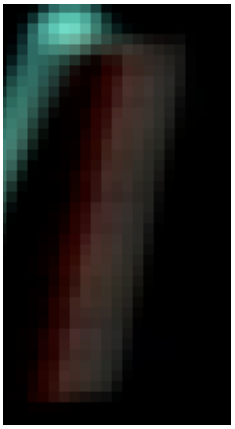
Below is the compare of using dilated convolution and without using dilated convolution:



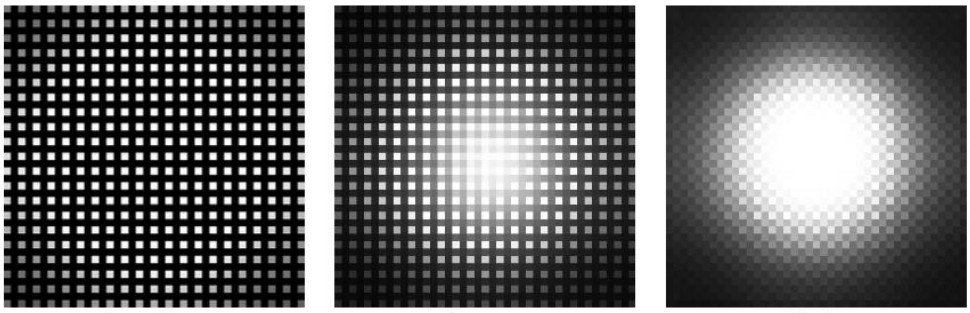


It can see that with get same number of pixels from image (both 3x3), the convolution with dilated seems give us more information. But we can see the problems of dilated by zoom the image bigger, because of the space we add in dilated, some pixels seem loss.
And if we doing smoothing before the dilated convolution its going to like:





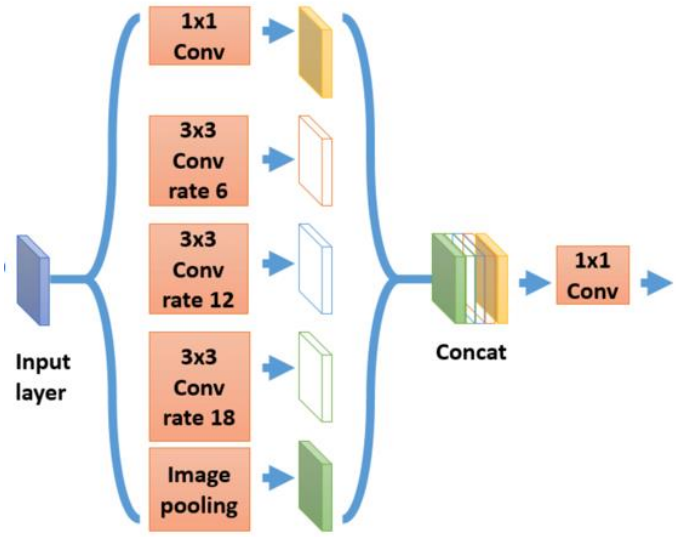
The result keeps the benefit of getting more information and don't cause discontinues at the same time. It's kind of support the result from the paper.
Again, it's quite unlucky we don't find a way to make the paper's code work but we can see the result from paper which do a better show of how powerful their methods work:



It can see that the baseline (directly dilated) separate the pixels but with their two method it will makes the output image's pixels not separated that much. Which means their method can help reduce the Gridding influence form dilated convolution.

How Deeplabv3+ handle with Dilated convolution. (partial research from paper Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation, the entire paper refers to too much methods which we can't get)

As talk above, Dilated convolution can provide a model better view of the image with same computation, So Deeplav3+ also using Dilated convolution. But in order to avoid the Gridding problem we talk above, let come up with a method called ASPP. A s the image shows below, different with our paper who did smooth to the image, they apply couple dilated convolution with different rate(space) to the image, to get serval sub-image, and then combine all the sub-image to build the output.



It's kind of smart, because since the space is different, the sub-images will get information from different location of the original image and as a result the output image will not losing information

The second paper we have read is *PointRend: Image Segmentation as Rendering*. This is a Paper published on 16 Feb 2020 by Alexander kirillov, Yuxin Wu, Kaiming He, and Ross Girshick.

PointRend treats segmentation as a rendering problem, using a segmentation strategy to adaptively select a set of non-uniform points and calculate labels. PointRend can be incorporated into common instance segmentation meta-architectures such as Mask R-CNN and semantic segmentation meta-architectures such as FCN.

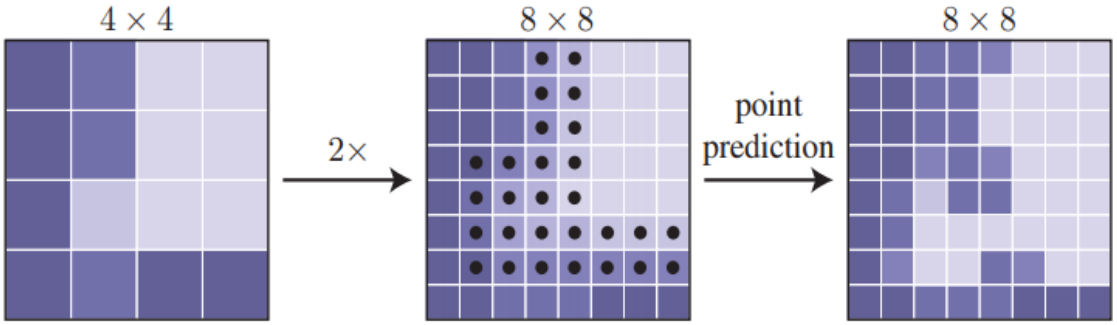
In this paper, they analogize image segmentation in computer vision to image rendering in computer graphics. Rendering algorithms in computer graphics output a regular grid of pixels. Now, we can think of an image as a continuous entity, and the segmentation output, which is a regular grid of predicted labels, is "rendered" from it. The entity is encoded in the network's feature maps and can be accessed at any point by interpolation. Based on this analogy they can do image segmentation using point representations. PointRend is general and can be flexibly integrated into existing semantic and instance segmentation systems.

The overall process can be divided into three steps:

1. **Point selection strategy:**
   Chooses small number of points to make predictions.
2. **Point-wise feature representation**
   Features computed by bilinear interpolation
3. **Point head**
   Multi-layer perceptron to predict the labels

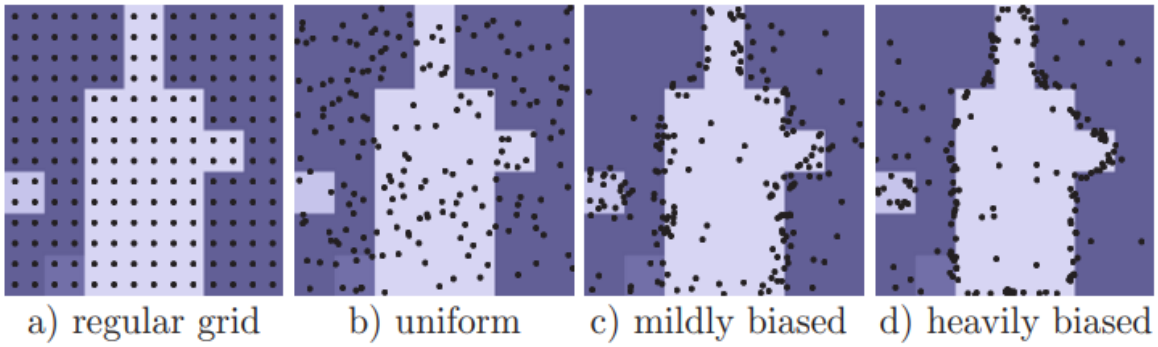
Point selection

The core of point render is the idea of flexibly and adaptively selecting points in the image plane at which to predict segmentation labels. they use adaptive subdivision in computer graphics to find a point where has a high chance that the value is significantly different from its neighbors



The above image is an example of adaptive subdivision step. A prediction on a 4 by 4 grid upsampled by 2x using bilinear interpolation. Then pointRend predicts the N most ambiguous points to recover detail on the finer grid. Repeated the process until the desired grid resolution is achieved

PointRend also needs to select points at which to construct point-wise features for training the point head. Let make N the value for how many points we want on the grid. The sampling strategy selects N points on a feature map to train. There are three principles they used in the paper *over generation*, *importance sampling*, and *coverage*.



over generation candidate points by randomly sampling points and importance sampling to focus on points with uncertain coarse predictions, and remaining points are sampled from a uniform distribution.

Point-wise Representation

At selected points, by combining fine-grained and coarse prediction features, point-wise features will be constructed.

1. Fine-grained features:

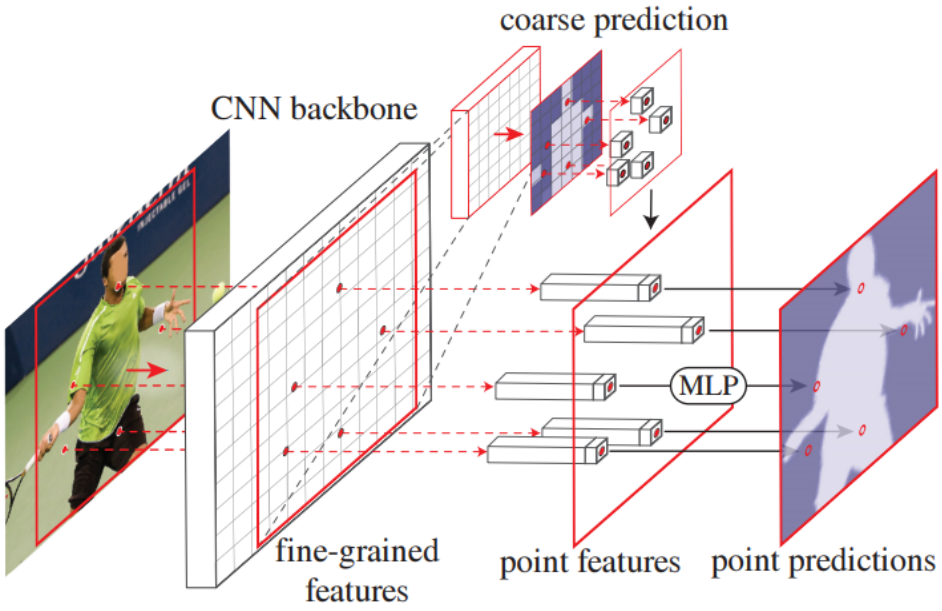
   From ResNet, we need to extract a single feature map to allow PointRend to render fine segmentation details for fine-grained features and extract a feature vector at each sampled point from CNN feature maps.

2. Coarse prediction features:

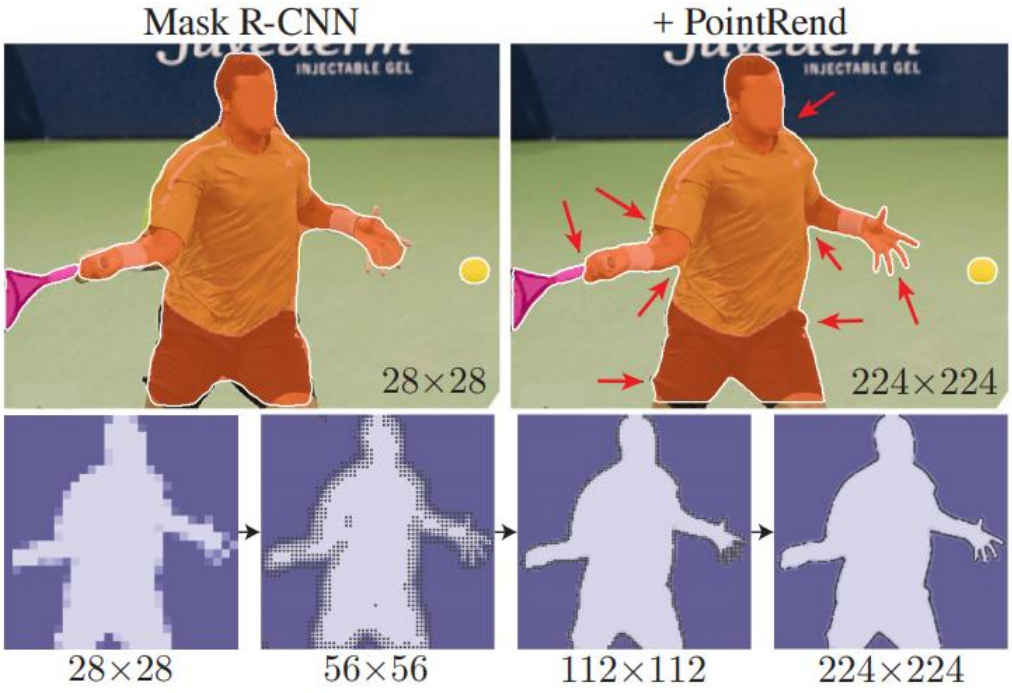
   By design, the coarse resolution needs to provide a more globalized context. Simultaneously, the channels convey the semantic classes because fine-grained features do not contain region-specific information and overlap instance and may only contain relatively low-level information.

Point Head

A small neural network is trained to predict a label from this point-wise feature representation independently for each point. Using the prediction we can sample a set of points they would like to refine it for this points we extract corresponding features for backbones (ResNet, CNN, etc) intermediate representation and the coarse prediction using bilinear interpolation.



Above is a full illustration of PointRender process. A standard network (CNN, resNet) for instance segmentation takes an input image and yields a coarse mask prediction for each detected object. To refine the coarse mask, PointRend selects a set of points and makes prediction for each point independently with a small MLP. The MLP uses interpolated features computed at these points from a fine-grained feature map of backbone CNN and from the coarse prediction mask. The coarse mask features enable the MLP to make different prediction at a single point that is contained by two or more boxes.

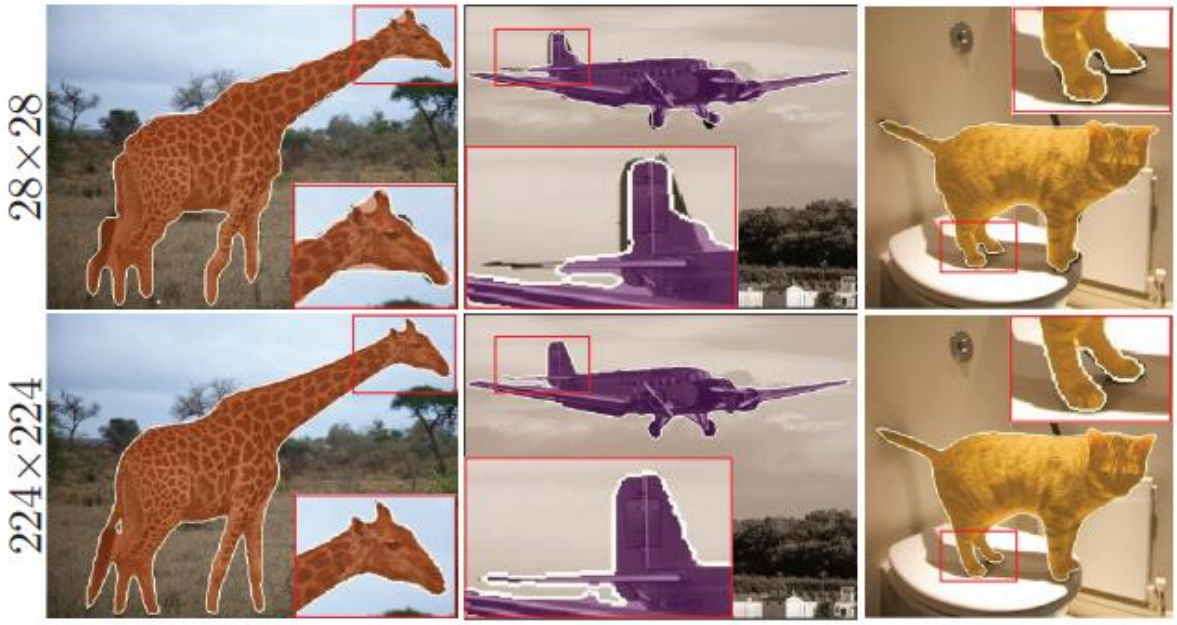


Above is an illustration of Instance segmentation with PointRend. The use of the subdivision strategy enables us to use a higher resolution image to work. unlike traditional R-CNN limited to a lower resolution, 28 by 28 in this example, using PointRend we can work on 224 by 224 images. PointRend yields significantly more detailed results. During inference, PointRend iterative computes its prediction. Each step applies bilinear upsampling in smooth regions and makes higher resolution predictions at a small number of adaptively selected points that are likely to lie on object boundaries.

Experiments

mask head	output resolution	COCO AP	COCO AP*	Cityscapes AP
4× conv	28×28	35.2	37.6	33.0
PointRend	28×28	36.1 (+0.9)	39.2 (+1.6)	35.5 (+2.5)
PointRend	224×224	**36.3** (+1.1)	**39.7** (+2.1)	**35.8** (+2.8)

The expreiments use Mask R-CNN with a ResNet-50 + FPN backbone. The default mask head in Mask R-CNN is a region-wise FCN, denote by "4X conv". PointRend outperforms the standard 4× conv mask head both quantitively and qualitatively. Higher output resolution leads to more detailed predictions.



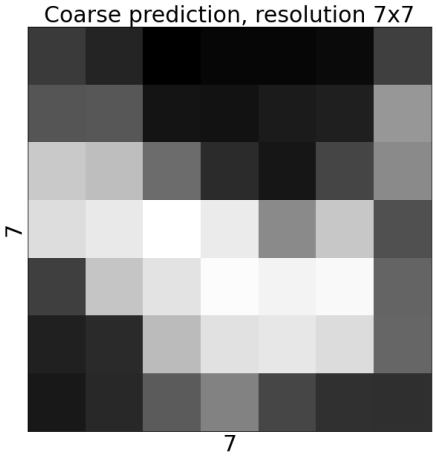
In this figure we can see the boundary of object is much clearer with PointRend (224*224) segmentation.

The figure above shows Mask R-CNN with standard head vs. Mask R-CNN with pointRend. As we can see, the pointRend prediction is much sharper on detecting boundaries. Each red arrow shows the most significant area where it shows differences.

Our own experiment

Because our team memebrs have different hardware configurations, to avoid package installation problems and to focus on the project per se, we decided to use google colab as the platform to collaboration.

1. The first step is to select points to inference and train



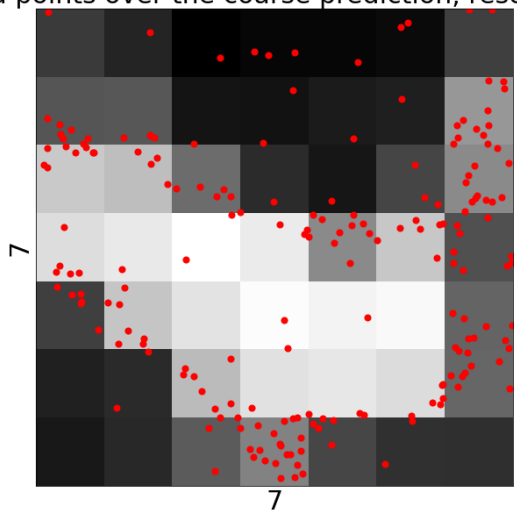

```
# Get bounding box predictions first to simplify the code.
detected_instances = [x["instances"] for x in model.inference(batched_inputs)]
[r.remove("pred_masks") for r in detected_instances]  # remove existing mask predictions
pred_boxes = [x.pred_boxes for x in detected_instances]

# Run backbone.
images = model.preprocess_image(batched_inputs)
features = model.backbone(images.tensor)

# Given the bounding boxes, run coarse mask prediction head.
mask_coarse_logits = model.roi_heads._forward_mask_coarse(features, pred_boxes)
```

Code adapted from “PointRend” in Detectron2 Tutorial

MLP can combine this mask point with the feature map of backbone CNN to computed features that can be interpolated.

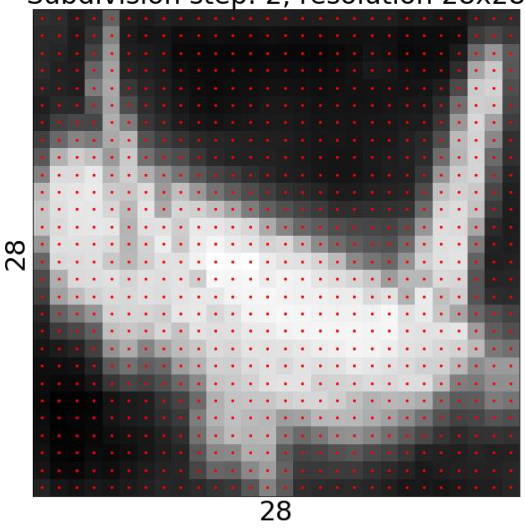
After the previous step, we need to select points where coarse prediction is uncertain to train PoinRend head

Sampled points over the coarse prediction, resolution 7x7

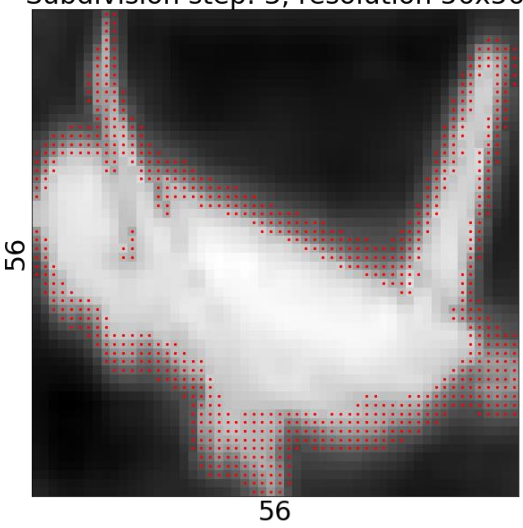



```python
# Select points given a corse prediction mask
point_coords = get_uncertain_point_coords_with_randomness(
  mask_coarse_logits,
  lambda logits: calculate_uncertainty(logits, pred_classes),
  num_points=num_points,
  oversample_ratio=oversample_ratio,
  importance_sample_ratio=importance_sample_ratio
)
```

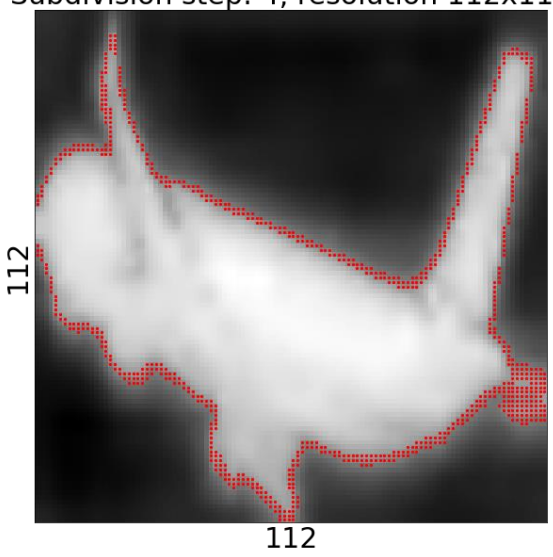
Subdivision step: 2, resolution 28x28



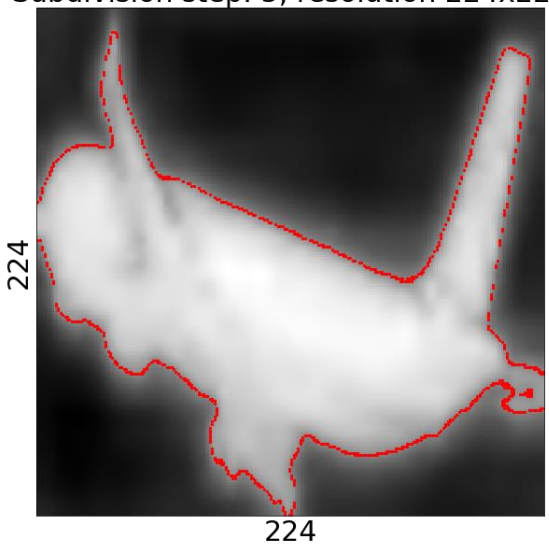
Subdivision step: 3, resolution 56x56



Subdivision step: 4, resolution 112x112



Subdivision step: 5, resolution 224x224



Above is a visualization of the Subdivision technique in the process. As we said before, the PointRend method's key is using the subdivision technique to render an anti-aliased, high-resolution image efficiently.

On top is mask-RCNN + PoinRend that detected sharp corners.

On the bottom is traditional mask-RCNN method that only detects rough boundary of an object.

In conclusion, PointRend can dramatically improve the segmentation accuracy compare to the traditional segmentation method. As we can see form the result, the model with PoinRend augmentation can detect a much more clear outline of the object as well as using much less computation to compute on image with larger resolutions.