Final Project - 896974

Applying Super-Resolution on Object Detection Dataset

Submitted by David Guriel, 211481130 to Prof. Gal Chechik & Dvir Samuel

Bar-Ilan University, 2021

Link to Research Proposal: *https://docs.google.com/document/d/1nvLB_P0zkf0lqOk4_fypE9R3RB13Z-H_q9ZmKhpWMMk/edit*.

1. The Learning Problem

In this project I chose to discuss the learning problem of applying an Image Super-Resolution model to Object-Detection datasets. My goal is to improve the performance of an Object-Detection model on the higher-resolution images. The comparison will be to the results of a pretrained model.

The examined dataset I chose is COCO-Detection. Note also that since they are massive and huge, I sampled 5000 images of each dataset. The performance is evaluated using MSE Loss (Pixel-by-Pixel comparison). Moreover, for the Super-Resolution process, I use PSNR (Peak Signal-to-Noise Ratio), which is a common measure for Image Reconstruction:

$$PSNR = 10 \cdot log_{10}\left(\frac{MAX_I^2}{MSE}\right)$$

I conjecture that the resolution-increasing will sharpen the edges of the objects with respect to their background, especially for small or far objects, and thus results in better results in the Object-Detection task.

I will pay special attention to the performance on such objects, as the greatest improvement is expected to occur at them.

Note: I did not implement the project's idea on PASCAL-VOC, because I could not manage to work with its data. I implemented my idea only on the COCO dataset.

2. Previous Solutions to the Problem

In 2015, Dong et. al have shown that Super-Resolution Convolutional-Neural-Network (SRCNN) achieves better results than the traditional methods and dominates them.

My project is based on the project from Stanford's CS-229, Spring 2020, which implemented Dong et. al's architecture. I used one of the implementations of that network, and did not train it from scratch, i.e. used a pretrained SRCNN network.

As far as I know, there are no attempts to tackle the Object-Detection task with Super-Resolution methods. Therefore, I was not sure which results to expect.

3. Experimental Methods

The code is detailed in the Colab notebook attached. The order of the following parts does not correspond to the order of the code.

1. Using a Pre-Trained SR Model

The model is presented in models.py. It consists of 3 layers of 2-dimensional Convolution, separated by ReLU layers. In a sense, this network is not very complicated in terms of the architectures we saw in the classes. However, apparently it is enough for achieving higher resolution-improvement than the traditional methods.

I used an implementation from GitHub for the pretrained model (see reference below), specifically the pretrained weights file *srcnn_x2.pth* and the test file *test.py*, through which I generated the "enhanced" COCO dataset.

2. Loading the COCO Dataset

I used the dataset MS-COCO of val-2017, which consists of 5000 images. When I used the entire dataset, even just loading it to the memory took a couple of hours, so I decided to settle with a more feasible size – 550 images. In addition, I found out there are few images that do not have any objects annotated, and thus considered as "have no objects to detect", at least by COCO's annotation. Therefore, I filtered all those images (after a lot of time wasted on finding why this bug happens).

3. Applying SRCNN on COCO

Once the preprocessing is done, I applied the SRCNN network on the COCO images. I also printed the PSNR measure of every image. One can see that they range between 30 to 40. A short script shows the average and std: $\mu = 35.22, \sigma = 3.6$. The resulting images are generated in the folder *coco_images_enhanced*.

Note: although in the proposal I suggested reshaping the images to have the same dimensions, eventually both the SRCNN and Faster R-CNN networks allowed the inputs to be of different sizes. Thus, in general the dimensions remained mostly the same. Yet, the resampling in the SRCNN networks lead to minor changes in the dimensions (at most adding/subtracting a pixel in each dimension).

4. Finetuning Faster R-CNN on the Enhanced Dataset

For that part, I used mostly the code from the third assignment, which is also found in the Torchvision tutorial (see link below). However, I had to make some adjustments in the Dataset class definition, in order for it to fit to my dataset (and not the Penn-Fudan). This included manually calculating the masks (although I did not use it), handling degenerate bounding-boxes (which probably occurred due to the minor dimension changes in the images).

I used 50 images for the test set, that is a train/test ratio of ~0.9/0.1. I chose to use SGD optimizer with an lr scheduler with $\gamma = 0.1$. The network was trained for 5 epochs. I used the standard helper functions of the tutorial of Torchvision.

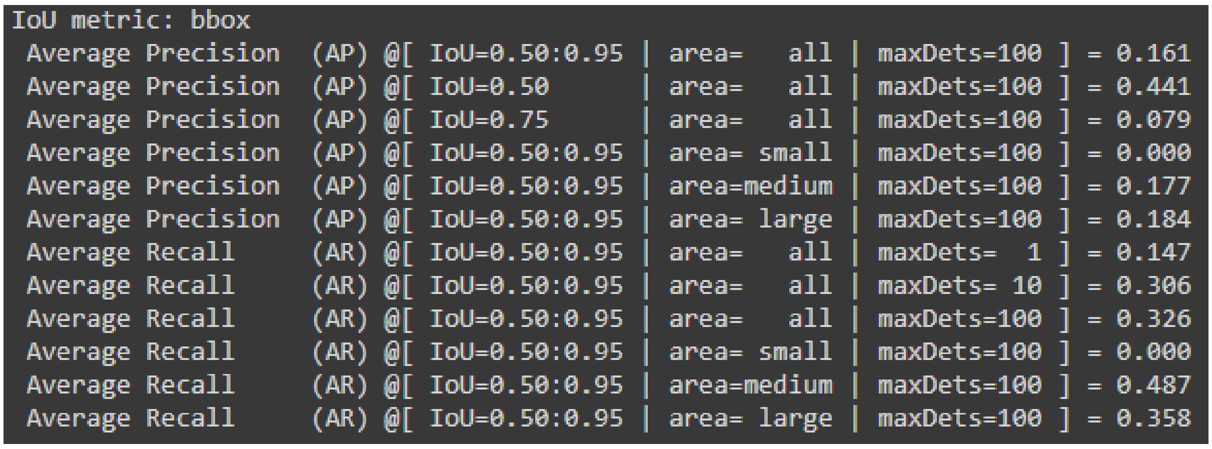
4. Results and Analysis

Below the section of training the network (finetuning) at the Colabnotebook, I added some examples to demonstrate the visual results of the network. One can notice 2 major issues:

- All the classifications are of "person", that is the network identifies all the objects only as persons.
- The model detects either the same object multiple times, or detects areas that do not have objects (which results in high False-Positive error). I conjecture that this is also related to the Region Proposals mechanism of Faster R-CNN.

Assuming the code is correct and has no bugs, the consequences of the results are that performing Super-Resolution causes an Adversarial effect - changing the resolution confuses the network.

The numerical results of the training quite correlate with that:



```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.161
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.441
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.079
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.177
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.184
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.147
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.306
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.326
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.487
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.358
```

One can observe the performance for small objects (the 'area' column) - the Average Precision and Recall equal to 0. This means that unlike what we expected, the Super-Resolution harmed the performance on them, and did not" allow better precision".

However, on objects with IoU=0.50, the results are more promising: my network achieved Average Precision of 44%. For comparison, the original Faster R-CNN network achieved Average Precision of 42.7% (see reference to the paper below), or 34.9%, depending on the dataset's version.

In conclusion, Super-Resolution results in no improvement for small objects, but for large ones it does have some positive effect.

5. Bibliography

1. The paper from the Stanford course: http://cs229.stanford.edu/proj2020spr/report/Garber_Grossman_Johnson-Yu.pdf
2. The original paper of Super-Resolution CNN: https://arxiv.org/abs/1501.00092
3. The Faster R-CNN paper: https://arxiv.org/abs/1506.01497
4. COCO – website: https://cocodataset.org/#detection-2020
5. The format of the COCO Detection dataset: https://cocodataset.org/#format-data
6. Object Detection in PyTorch: https://pytorch.org/docs/stable/torchvision/models.html#object-detection-instance-segmentation-and-person-keypoint-detection
7. Link for the implementation of SR-CNN I used: https://github.com/yjn870/SRCNN-pytorch.git
8. Pytorch - Torchvision tutorial, including Fine-tuning a pretrained model: https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html
9. Comparing performances on COCO-dataset: https://paperswithcode.com/sota/object-detection-on-coco (Faster R-CNN has 34.9% Average Precision)