Faster scale-invariant object detection

Agarwal, Sonu **Gupta, Anant** **Rajput, Shashank**

Abstract

We aim to investigate the feasibility of a scale-invariant object detection model, using a single deep neural network. We aim to achieve this by designing a novel layer which we call as "zoom" layer. The aim of these layers is to determine the scale of the input images, and transform them to a standard scale. This allows further convolution layers to focus on learning high level features for object detection without worrying about different image scales. In addition, they don't have to deal with noisy or background part of the image. We use statistical methods to calculate the relevant object region from the feature maps. Our approach did not work as expected, and we try to highlight some of the issues we faced during training, and potential things to try.

1 Introduction

Object detection includes two tasks: object localization and object classification. The problem of object localization is quite complex and has many challenges. Object could be anywhere in the image and they could be of any size. Also, there could be multiple objects in the image, each of which need to be correctly localized and classified.

There have been several recent works on object detection using deep neural network. Many of these methods deal with the problem of object localization in a similar fashion. They consider all possible bounding boxes of pre-defined sizes and aspect ratios within some pre-defined constraints to limit the number of bounding boxes. Then they calculate some score at the end of the network, based on which they accept or reject a probably object region. The number of bounding boxes is usually very large, and leads to wasted computation and increased time complexity.

R-CNN model proposed by Girshick et al. uses selective search method to propose 20k bounding boxes [1]. Then raw image corresponding to each of these bounding boxes are processed through entire architecture. Ross Girschick then proposed a Fast R-CNN network, in which feature maps obtained after the sequence of convolution networks are shared across each of these 20k bounding boxes [2]. Only fully connected networks are now processed for all bounding boxes. Following up on these works, Ren et al. introduced the idea of region proposal network (RPN) in their Faster R-CNN model [3]. They introduced a RPN after the convolution network which calculated the probability of being any object in a given bounding box. They used the set of k anchors by which they calculated the objectness score for these k bounding boxes for each of the location of feature map. Lately there has been some attempts to do perform object localization and object classification in an integrated manner [4][5]. But they still use the concepts of anchors and scan through whole image at some stage to create multiple bounding boxes.

As it can be intuitively seen, this approach can be inefficient and may lead to slower framer per second (FPS) task rate and might not be useful for real time applications, as well as power-constrained mobile devices.

We aim to investigate the possibility of designing a network which can localize and classify the object, irrespective of its size or location in the image, without considering many bounding boxes at any stage of the network. We propose a new architecture which uses a zoom-in approach to vary scale continuously, and only maintains a single region of interest per layer corresponding to the current

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

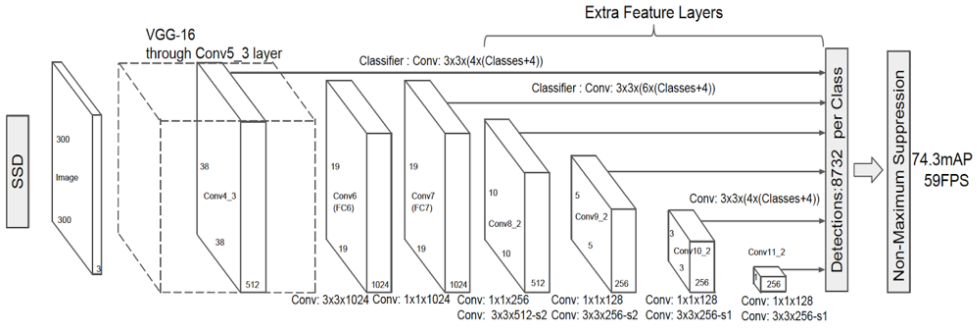
zoom level. We have simplified this problem to a single object scenario but this can be extended to multiple object as discussed in future works section.

In current standard architecture, CNN and max pooling are there which works in hierarchical manner. In our architecture, we plan to insert a zoom-in layer between CNN feature map and max-pooling layer. In standard architectures, region of interest depends on filter size which are fixed while we plan to make it variable by changing the amount of zoom in our zoom layer. Also, zooming allows the network to trim off parts of the image which don't have the object progressively, so that subsequent layers don't have to deal with them.

Our goal can be summarized as: To generate a scale-invariant representation at the end of our zoom-in layers, such that it corresponds to a cropped version of the input image in which the target object has been centered and scaled appropriately.

2 Comparison

The network used by state-of-the-art model SSD looks like:



In the SSD, the early network layers are based on a standard architecture used for high quality image classification (truncated before any classification layers), which they call the base network. Then they add auxiliary structure to the network to produce detections with the following key features: Multi-scale feature maps for detection: They add convolutional feature layers to the end of the truncated base network. These layers decrease in size progressively and allow predictions of detections at multiple scales. The convolutional model for predicting detections is different for each feature layer. Convolutional predictors for detection: Each added feature layer can produce a fixed set of detection predictions using a set of convolutional filters. For a feature layer of size mxn with p channels, the basic element for predicting parameters of a potential detection is a 3 x 3 x p small kernel that produces either a score for a category, or a shape offset relative to the default box coordinates. At each of the m x n locations where the kernel is applied, it produces an output value. Default boxes and aspect ratios: They associate a set of default bounding boxes with each feature map cell, for multiple feature maps at the top of the network.

So SSD basically, predicts an output for every cell of each feature map at different scales. We aim to create a network which doesn't need to predict outputs for so many different combinations of the input. Instead, we expect that once we have applied our "crop-and-zoom", layer to the input image, an existing network should be able to classify the image and fine-tune the bounding box (especially for an image with a single object). We shouldn't need feature maps at different scales and also we shouldn't need to calculate outputs for each cell of the feature map.

3 Methodology

As described earlier, we use a ground-up approach to bounding box proposal. The network consists of alternating convolution and zoom layers. Instead of using explicit pooling layers, we added strides in our convolution layers [6]. The convolution layers output increasingly higher order feature maps, which are used by the zoom layers to decide where and how much to "zoom". The initial layers will learn to be scale invariant, but only work with low level features. As the depth increases, the layers will start learning higher level features, but also become less resistant to scale changes. Thus, the

hope is that the lower layers do a good enough job of coarse zooming, leaving the higher layers to do the actual object detection in the uniform scale images.

3.1 Zoom layer

The zoom layer computes 2 things:
1. The new image center
2. How much to zoom into the new center (distance from the center upto new image boundaries)
These values are then used to crop the input feature maps to the required area around the new center, and finally resize the cropped images to the dimensions of the input. The new image center μ is given by the mean of the positions, weighted by the mean feature map activations:

$$\mu_x = \sum_{i=1}^{n} ic_i / \sum_{i=1}^{n} c_i$$

$$\mu_y = \sum_{i=1}^{n} ir_i / \sum_{i=1}^{n} r_i$$

where c_i and r_i are the means of the ith column and ith row respectively, of the mean feature map $\bar{F}$. Here $\bar{F} = \sum w_i F_i$, a weighted mean of the feature maps. The weights are trainable parameters of the network.
Similarly, the distance from the center d is computed using a weighted standard deviation of the positions:

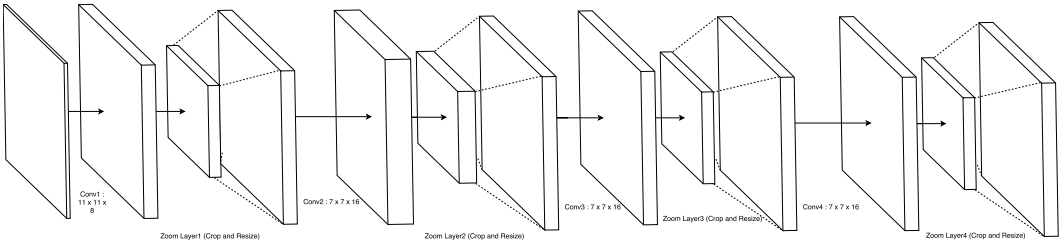
$$d_x = \sum_{i=1}^{n} (i - \mu_x)^2 c_i / \sum_{i=1}^{n} c_i$$

$$d_y = \sum_{i=1}^{n} (i - \mu_y)^2 r_i / \sum_{i=1}^{n} r_i$$

d_x and d_y are multiplied by a factor f, which is a trainable weight for each zoom layer, and controls the relative zooming action of each layer. Figure 1 illustrates this zooming process. To ensure that a batch of images can be processed together as a Tensor, we actually crop a square image of side length $max(d_x, d_y)$, and black out the extra length in the new image.

3.2 Network architecture

Following the alternating convolution and zoom layers, we should ideally use more layers that consider many different bounding boxes, and compute scores for each of them. Then we should do an end-to-end training of the whole network. To save time, however, we trained only the zoom network using the loss function described in the next section. This had the added benefit of highlighting the (non) feasibility of the zoom layers (training issues).



3.3 Loss function(s)

To train the zooming part of the network alone, we compute the loss in the following way. We take a weighted sum of the output feature maps, normalize it, map it to the original image dimensions with the cropped out area of the original image set to zero. We then subtract it from a 0-1 mask denoting the ground truth bounding box, and compute the norm. The idea is that the final zoomed in image should contain activations at the location of the object. Instead of calculating the explicit bounding

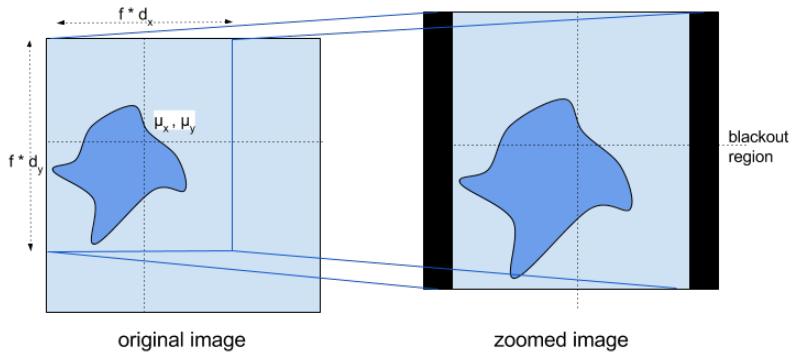


Figure 1: The left image shows a feature map with high activations (dark blue) and low activations (light blue). μ_x, μ_y, d_x and d_y are calculated as explained, and the cropping boundary is determined. Since d_x might not be equal to d_y and we always maintain square images, the zoomed image can have its borders blacked out.

box using subsequent layers, we compute this simpler loss to get an idea about the working of the zoom layers.

Calculating loss

procedure LOSS($cnn\_feature\_maps, ground\_truth\_bounding\_box$)
  $summed\_feature\_map \leftarrow \sum cnn\_feature\_maps_i$
  $normalized\_feature\_map \leftarrow$ zero_one_normalization($summed\_feature\_map$)
  $zeroed\_feature\_map \leftarrow$ zero_out_cropped_region($normalized\_feature\_map$)
  $rescaled\_feature\_map \leftarrow$ rescale_to_original_dimensions($zeroed\_feature\_map$)
  $ground\_truth\_mask \leftarrow$ compute_zero_one_mask($ground\_truth\_bounding\_box$)
  $loss \leftarrow$ frobenius_norm($ground\_truth\_mask - rescaled\_feature\_map$)
  return $loss$

3.4 Other architectures tried

3.4.1 Object localization without zooming

One of the simpler architectures that we tried was having a series of CNN and pooling layers to reduce the size of the image and then find the mean and standard deviation of locations of activations in the small image. The idea was that the CNN would get trained to give activations only where the object is / or give activation on its edges. Thus the mean location of activations would approximately be the center of the object and the standard deviation of the location of activations would be proportional to the size of the object. We thus calculated the bounding box using mean and standard deviation. The

loss was calculated as the L2 norm of the difference between the actual and the predicted coordinates of the bounding box corners.

Network details -

Unfortunately, the network did not learn to give activations on/near the object of interest. We think that the reason that CNN kernels weren't able to learn properly was because we were training them to identify the whole bounding box as an object. However, the objects don't actually occupy the whole bounding box, they only occupy a fraction of that. If we computed the loss only on the exact region occupied by the object, instead of the bounding boxes, we think the network might have learnt what we expected.

4 Experiments

4.1 Dataset details

Our main objective was to create a model that runs quickly (at run time) and is good enough to remove area where there is no object of interest, therefore we planned to work on images that contained only one object, so that the task now simplifies to removing the background (Once we remove the background whatever is left is the object of interest). Also, with this assumption, many of the calculations like mean location and area spanned by the object becomes simpler.

We took the pascal VOC dataset and selected only those images which had been tagged with only one object. There were 20 classes of objects and around 3000 images in the training dataset as well as the validation dataset.

4.2 Framework details

We used the Google' Object detection framework, which is open sourced (`https://github.com/tensorflow/models/tree/master/research/object_detection`) and uses tensorflow. The framework contains implementations of some of the basic as well as state of the art object detection and classification models.

4.3 Source code

The source code can be found at `https://github.com/shashankrajput/models`. We cloned the original framework and our main code changes are in the `/research/object_detection/meta_architectures/ssd_meta_arch.py` path. We did not create a new class (we were facing some issues in incorporating a new class in the framework), instead we replaced the original logic of the ssd based meta architecture with our architecture.

5 Results

We tried training multiple variants of our network, with different configurations and hyperparameter settings. In all cases, the training seemed to get stuck at points corresponding to trivial bounding boxes, like complete image or empty bounding box. Since we didn't get a non-trivial solution by training, we are not reporting any results. In the next section, we try to explain why training was not successful.

6 Analysis

6.1 Bounding boxes equal to full image

Our zoom network has 2 sets of weights: those associated with the convolutions, and those associated with the zooming factor. When we tried training the zoom network, the zoom factors learnt were so large that each layer proposed the whole image as the new image. The problem was that once the zoom factor becomes large enough, the gradient of the loss with respect to the factor becomes zero (since increasing the factor further doesn't change the output image, it still remains the whole uncropped image). To remedy this, we could have tried to make the cropping factor a smooth function

of the standard deviation (with no sharp "cut-off" when the factor reaches the image boundary), so that the zoom factor always remains trainable. But this is more of a hack than a solution, and maybe there is a better way.

6.2 Convolution result zero

We also noticed that many convolution outputs were coming out to be exactly zero. This is a standard problem that CNNs with ReLU activations face. The solution that is normally used is leaky ReLUs [7]. However, since we use the convolution outputs as weights for mean and standard deviation calculations in the zoom layers, our convolutions outputs have to be positive. Therefore, we used sigmoid instead. This solved the zero convolutions problem.

6.3 Standard deviations not discriminative enough

Our whole idea relies on the assumption that it is possible to compute zooming factors and centers using means and standard deviations of activations alone. However, what we observed was that means hover around the image center, and standard deviations are similar to what a randomly distributed activation map would produce. This could either mean that our convolutions didn't learn feature representations adequately and are just random numbers, or it could be a more fundamental issue that feature maps learnt by convolutions simply cannot be discriminated sufficiently using means and standard deviations alone. See clustering in future work.

6.4 Loss/idea may not be compatible with training CNNs

Even if our network can potentially represent a function that detects objects, it might be very hard to train. The issues about exploding zoom factors and vanishing convolutions could be due to local minima in the loss function. One thing that could be tried is training layers individual zoom layers separately, by defining loss functions for each layer.

7 Future work / investigation

Although the 3 techniques implemented by us did not seem to work well, an approach that might work is if we do clustering on CNN's output to detect the objects. We can think of this as each cluster representing an object. If we can integrate such a clustering with CNN's output and jointly train the network, it might be faster than current object localization techniques (if we use a fast kind of clustering algorithm). Density based clustering would be a good option to identify objects. We did explore this, but we did not find an existing or come up with a new clustering algorithm that could be implemented in the tensorflow framework (basically a clustering algorithm which could be integrated with the CNN network and compatible with backpropagation algorithm). Thus we need to find a clustering algorithm that can spatially cluster pixels in an image and uses only matrix operations. To summarize clustering might give a better representation, but we don't know of an algorithm for clustering that could work with CNNs.

After exploring the different aspects of our project, we realized that we started with a much complex dataset (dataset containing images of different objects). Since we proposed a completely new layer, it might have been better to try to prove the working of this new architecture on the dataset of a single object, so that we can single out the problem of object detection and not worry about classification initially. If it was possible to learn the network for single object, it could be extended to the dataset with images of different objects.

Our proposed architecture was based on a heuristic to compute the new image center and scale. Some work is needed to justify why such a representation should be expected to work, and find out other minimal architectures to identify salient regions in the images. Some work is also needed to study trainability of such networks, stability of gradients, and robustness of the optimization problem.

References

[1] Girshick, R., Donahue, J., Darrell, T. and Malik, J., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. *In Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 580-587).

[2] Girshick, R., 2015. Fast r-cnn. *In Proceedings of the IEEE international conference on computer vision* (pp. 1440-1448).

[3] Ren, S., He, K., Girshick, R. and Sun, J., 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. *In Advances in neural information processing systems* (pp. 91-99).

[4] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y. and Berg, A.C., 2016, October. Ssd: Single shot multibox detector. *In European conference on computer vision* (pp. 21-37). Springer, Cham.

[5] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You only look once: Unified, real-time object detection. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 779-788).

[6] Springenberg, J.T., Dosovitskiy, A., Brox, T. and Riedmiller, M., 2014. Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806.

[7] http://cs231n.github.io/neural-networks-1/ (Dying ReLU problem)