Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition

1 Citation

He, Kaiming, et al. "Spatial pyramid pooling in deep convolutional networks for visual recognition." european conference on computer vision. Springer, Cham, 2014.

https://arxiv.org/pdf/1406.4729.pdf

2 Abstract

CNNs have take an input image of fixed size - we build a layer that relaxes this assumption. It takes variable sized inputs and produces fixed size outputs. It improves many popular CNNs and sets state of the art on PASCAL VOC and Caltech-101. It also gives huge speedups on R-CNN for object detection (just compute feature maps once and pool over arbitrary regions). On ImageNet 2014, we get 2nd in detection and 3rd in classification.

3 Introduction

CNNs take a fixed input size, so you need to crop or warp your input image. The reason for the fixed size requirement is because of the fully-connected layers. Our Spatial Pyramid Pooling (SPP) module is put after the last convolutional layer to take a variable size input and turn it into a fixed sized output.

You can also train on images of different sizes. To enable this efficiently, we basically create several copies of the network (for each size) that share parameters and use one for each epoch.

R-CNN (for object detection) is slow because you need to run a CNN for thousands of regions. If we use SPP, we just run the CNN once and pool over the regions. This ends up taking only 0.5 seconds per image.

4 Deep Networks with Spatial Pyramid Pooling

Basically, we create M equal sized bins on the image and max-pool over them to get a kM-length feature vector (k is number of channels of the input). We do this for each pyramid layer.

Notice that if $M = 1$, we are doing global average pooling.

GPU implementations prefer fixed size images. How can we handle this? First consider single-size training. Suppose our $pool_5$ layer produces $a \times a$ feature maps. For a pyramid level of size $n \times n$ bins, we use window size $\lceil \frac{a}{n} \rceil$ and stride $\lfloor \frac{a}{n} \rfloor$. To handle multiple scales, we just take the original image, get our square crop, and resize it to the proper size. We train one network for each input size, make them share parameters, and switch between them epoch by epoch.

5 SPP For Image Classification

Use ImageNet classification dataset. Resize so smallest side has length 256. Take center crop and 4-corner crops. Augment with flips and color altering. Use dropout on fully-connected layers. Drop learning rate when error plateaus.

On an NVIDIA Titan GPU, it takes 2-4 weeks to train our models.

Our basline models are the Zeiler and Fergus fast model (ZF), modified AlexNet, and modified OverFeat.

We use a four-level pyramid with sizes 1×1, 2×2, 3×3, and 6×6.

SPP improves performance. Multi-scale training beats single scale training. Using two scales (180×180 and 224×224) beats randomly sampling scale dimension from $[180, 224]$ probably because 224 is used at test-time.

At test time, using a single image (no crops or flips) does pretty well, but the crops/flips/multi-scales do have a little benefit.

We get great performance on Pascal VOC by taking our pretrained CNNs and using them to extract features for each image (no crops/flips/multi-scales) and training an SVM on the resulting feature vectors. Upscaling the Pascal VOC images helps because the objects are small in the image. We also do well for Caltech-101.

6 SPP-Net for Object Detection

R-CNN generates 2000 candidates with selective search, warps them to a fixed size, extracts features with a CNN, and feeds the resulting vectors to an SVM. The runtime bottleneck is that we need to run the CNN 2000 times for a single image. With SPP-Net, we can extract features on the full image with the CNN (remove the SPP piece). Then, for each region, we pool using a 4-level SPP pyramid. With these feature vectors, we train the SVM.

We find we can improve performance if we consider the input image at multiple scales. This means each proposed region will have multiple possible sizes (depending on how we scale the input image). We choose the scale that brings the region size closest to 224×224. This improves performance.

Fine-tuning also improves performance.

Using bounding box regression to post-process the prediction windows also helps.

We get performance comparable to R-CNN, but we are two orders of magnitude faster.