Sergey Mushi... • (3rd in this Competition) • 10 months ago • Options • Reply 0

Great post and solution, guys!

One thing I notice as very unusual -- do you really used 4 patches per batch? Why so little?

And also, do you have breakdown to classes? If so, could you share? (it is very interesting to compare with our results!)

arnowaczy... • (4th in this Competition) • 10 months ago • Options • Reply 1

Yes, it was 4 256x256 patches per batch. We used GPU with 8GB memory. It should be enough to train with larger batch size but because of some implementation details we quickly got to memory limit. When I think of it now, I wish I hadn't optimized the code for larger batch size.

After competition we found simple way to train models with arbitrarily large batch size in Pytorch. Because in Pytorch you control when optimizer does updates and when gradients are zeroed out, the algorithm is straightforward:

```
1 zero out gradients values
2 run a couple of times forward pass and backword pass
(sample new batch at each time) - gradients are automatically
accumulated
3 call optimizer which does weights update
4 Repeat 1->2->3
```

It's definitely something I will try in further competitions. It may be slow, but good for convergence.

Our results per class in final submission:

```
classid,private,public
1,0.06295,0.07791
2,0.02311,0.01732
3,0.04618,0.07758
4,0.04410,0.04219
5,0.06445,0.05492
6,0.08549,0.07358
7,0.08887,0.09295
```

```
8,0.01817,0.05559
9,0.01414,0.03328
10,0.01140,0.01590
total,0.45886,0.54122
```

Sergey Mu... • (3rd in this Competition) • 10 months ago • Options • Reply 1

Thanks for sharing! Some of you classes are extremely good, like cars (given how hard it to detect) or crops (almost perfect detection) It is strange that you hit memory limits with such small batch. I have 8Gb GPU too and experimented with 224*224*20 patches (it is a bit smaller than 256*256*20) and batch about 24 using Keras+Theano. However, great finding how to accumulate gradient in PyTorch, it is definitely worth knowing!

CRaymond • (119th in this Competition) • 10 months ago • Options • Reply 0

Thanks so much for sharing. I would like to ask for your up-conv step, is it a repeating of nearby pixel or is a real 3x3 kernel learned for upsampling? Thanks a lot.

arnowaczy... • (4th in this Competition) • 10 months ago • Options • Reply 0

It's transposed convolution with 3x3 kernel which is learned to upsample image. Here you'll find details for UPCONV layer: https://deepsense.io/wp-content/uploads/2017/04/architecture_details.png

CRaymond • (119th in this Competition) • 9 months ago • Options • Reply 0

Thx a lot for reply. I use Keras, and I found there was not 3x3 upsamle kernel, it only had an upsampling kernel to duplicate pixel value. So this is by PyTorch or you implemented this in Keras? For learning purpose, I am thinking to reimplement your idea and see how well can I score. Thanks.

For "Alignment was necessary to remove shifts between channels", do you think OpenCV "MOTION_TRANSLATION " (http://www.learnopencv.com/image-alignment-ecc-in-opencv-c-python/) is sufficient? Also, after alignment, there will be some missing contents for some channels, how do you handle this? Many thanks for sharing knowledge.

Michal • (4th in this Competition) • 9 months ago • Options • Reply 1

First, all the images were upsampled to the resolution of the RGB image. Then we did our registration with translation only, using OpenCV "MOTION_TRANSLATION".

Also, we registered both of the multispectral images to the panspectral one, then panspectral to RGB and then we composed translations to have all images aligned with the RGB ones. This type of registration seemed to be less likely to diverge than registering directly to RGB, although we still had divergence for a few images (these were just a few fairly homogenous ones so we just kept misaligned images, but maybe some more intelligent backup would be better).

Once registered, the images were aligned and we concatenated all the channels. The missing parts after alignment were just filled with the image average value for each channel.

arnowaczy... • (4th in this Competition) • 9 months ago • Options • Reply 1

Final architecture was implemented in PyTorch. For convolutional upsampling we used this: http://pytorch.org/docs/nn.html#convtranspose2d but Keras has something similar: https://keras.io/layers/convolutional/#conv2dtranspose

Alex • 8 months ago • Options • Reply 0

Thanks for sharing! Did you guys do any preprocessing to the images (smoothing, blurring, boosting contrast, etc.) before feeding them in?

arnowaczy... • (4th in this Competition) • 8 months ago • Options • Reply 1

We normalized each input channel independently to have a zero mean and unit variance. This approach requires some pre-computed mean and std per channel. You can get this numbers in a couple of ways:

1) for each input image compute its own mean and std and use them during normalization

2) compute mean and std for entire dataset and use always the same number or any input image

3) compute mean and std per each 5x5 region and normalize input images by region statistics to which image belongs to (I think it worked the best in our case)