Jun 5 · 11 min read

# DeepGlobe Challenge: Three Papers from Neuromation Accepted!

We have great news: we've got not one, not two, but **three** papers accepted to the DeepGlobe workshop at the top computer vision conference, CVPR 2018! This is a big result for us: it shows that our team is able to compete with the very best and get to the main conferences in our field.

Today, we present one of the solutions that got accepted, and it is my great pleasure to present the driving force behind this solution, one of our deep learning researchers in St. Petersburg, with whom we have co-authored this post. Please meet Sergey Golovanov, an experienced competitive data scientist whose skills have been instrumental for the DeepGlobe challenge:


Sergey Golovanov Researcher, Neuromation

## The DeepGlobe Challenge

Satellite imagery is a treasure trove of data that can yield many exciting new applications in the nearest future. Over the last decades, dozens of satellites from various agencies such as NASA, ESA, or DigitalGlobe that sponsored this competition, have collected terabytes upon terabytes of data.

At the same time, satellite imagery has not yet become the target of much research in computer vision and deep learning. There are few large-scale publicly available datasets, and data labeling is always a bottleneck for segmentation tasks. The DeepGlobe Challenge is designed to bridge this gap, bringing high-quality and at the same time labeled satellite imagery to everyone; see this paper by DeepGlobe organizers for a more detailed description of the dataset.

By virtue of data science competitions, organizers try to draw the attention of AI researchers and practitioners to specific problems or whole problem domains and thereby spur the development of new models and algorithms in this field. To attract the deep learning community to analyzing satellite imagery, DeepGlobe presented three tracks with different tasks highly relevant for satellite image processing: road extraction, building detection, and land cover classification. Here are three samples of labeled data for these tasks:

Image source

All of these tasks are formulated as semantic segmentation problems; we have already written about segmentation problems before but will also include a reminder below.

A part of our team from the Neuromation Labs at St. Petersburg, Russia, took part in two of the three tracks: building detection and land cover classification. We took the third place in building detection and fourth and fifth places in land cover classification (see the leaderboard) and got three papers accepted for the workshop! In this post, we will

explain in detail the solution that we prepared for the building detection track.

## Semantic segmentation

Before delving into the details of our solution, let us first discuss the problem setting itself. *Semantic segmentation* of an image is a partitioning of the image into separate groups of pixels, areas corresponding to certain objects, and at the same time classifying what is the type of object in every area (see also our previous post on this subject). That is, the problem looks something like this:

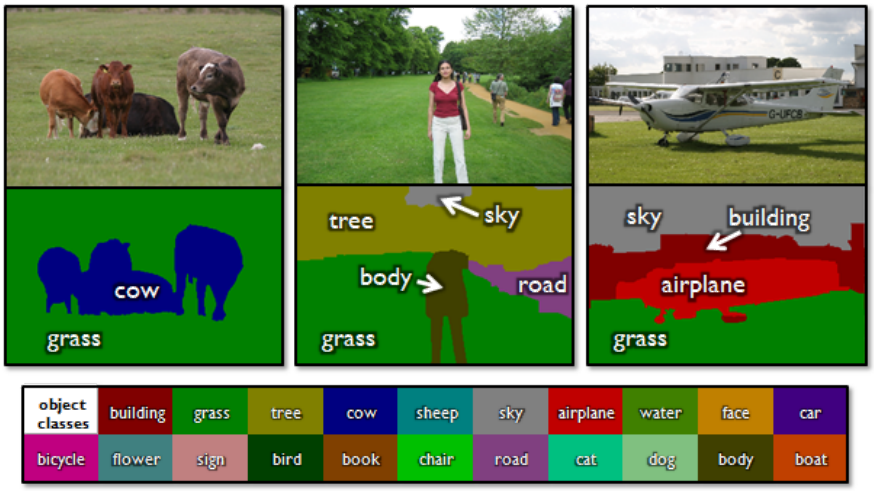



Image source

## Deep Learning for Segmentation: Basic Architecture

We have spoken about convolutional neural networks and characteristics of convolutions many times in our previous posts (for example, see here, here or here), so we will not discuss them in too many details and will go straight to the architectures.

The most popular and one of the most effective neural network architectures for semantic segmentation is U-Net and its extensions (there are plenty of modifications, which is always a good sign for the basic architecture as well). The architecture itself consists of an encoder and a decoder; as you can see in the figure below, U-Net is very aptly named:

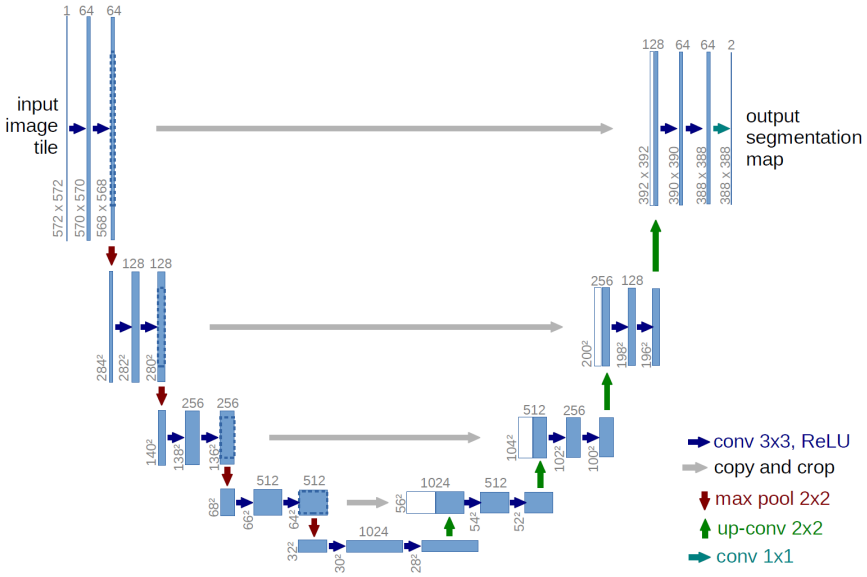



Image source

The encoder creates compressed image representations (feature maps), extracting multiscale features and thereby implicitly taking into account local context information within certain neighborhoods on the image. Real life problems often have not too much labeled data, so usually encoders in U-Net and similar architectures use *transfer learning*, that is, use a classifier pre-trained on ImageNet without the last layer of classification (i.e., only with convolutional layers) as the encoder. Actually, this is useful even if data is plentiful: this usually increases the rate of convergence of the model and improves segmentation results, even in domains that do not overlap with ImageNet such as segmentation of cell nuclei or, in our case, satellite imagery.

The decoder takes as input the feature maps obtained from the encoder and constructs a segmentation map, gradually increasing the resolution for more accurate localization of object boundaries. The main novel feature of U-Net that gives the architecture its form and name are the skip-connections that let the decoder “peek” into intermediate, higher-resolution representations from the encoder, combining them with the outputs of the corresponding level of the decoder. Otherwise, a decoder in U-Net is usually just a few layers of convolutions and deconvolutions.

## Deep Learning for Segmentation: Loss Functions

There is one more interesting remark about segmentation as a machine learning problem. On the surface, it looks like a classification problem: you have to set a class for every pixel in the image. However, if you treat it simply as a classification problem (i.e., start using per-pixel cross-entropy as the objective function to optimize the model) it won't work too well.

The problem with this approach is that it does not capture the spatial connections between the pixels: classification problems are independent for every pixel, and the objective function has no idea that a single pixel in the middle of the sea cannot really be a lawn even if it turns out to be green. This will lead to small holes appearing on segmentation results and very complicated boundaries between different classes.

To solve this problem, we have to balance the standard cross-entropy with some other loss function which is developed specifically for segmentation. We won't go into the mathematical details here, but the 2017 approach here has been to add the average DICE loss which allows to optimize the value of IoU (Intersection over Union). However, in order to strike the right balance for clear boundaries and absence of holes, one has to to choose the coefficients between these two losses very carefully.

The 2018 approach, rapidly growing in popularity, is the Lovász-Softmax loss (and a similar Lovász-Hinge loss), which serves as a differentiable surrogate for intersection-over-union. We have also used the Lovász-Softmax loss in another DeepGlobe paper of ours, devoted to the land cover classification challenge, so we will concentrate on the architectures here and perhaps return to discuss the loss functions in a later post. In any case, our experiments on this challenge have also shown that both DICE and Lovász-Softmax losses give tangible increases in segmentation quality.

## Deep Learning for Segmentation: Beyond the Basics

There are plenty of architectures based on the principles described above; see, e.g., this repository of links on semantic segmentation. But we would like to pay special attention to two very interesting ideas that have already proven to be very effective in practice: Atrous Spatial Pyramid Pooling with Image Pooling and ResNeXt-FPN.

The basic idea of the first approach is to use several parallel atrous convolutions (dilated convolutions) and image pooling at the end of the encoder, which are eventually combined through a 1x1 convolution. An *atrous convolution* is a convolution where there is some distance between the elements of the kernel, called *rate*, like here:

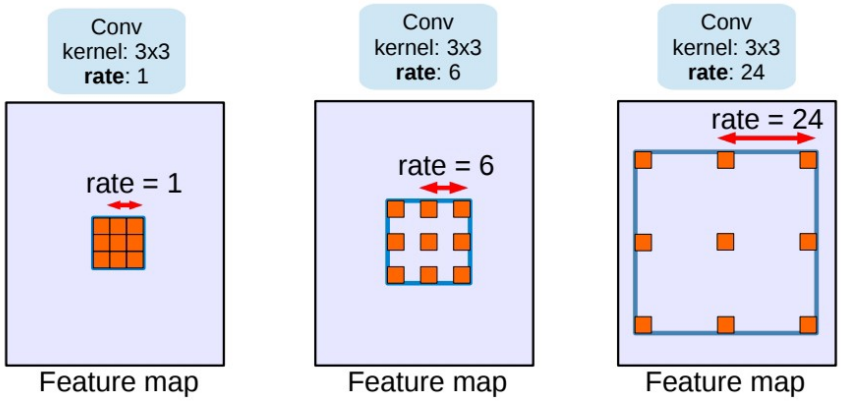
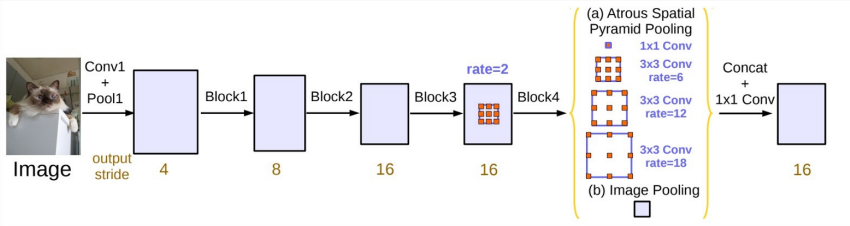



Image source

Image pooling simply means averaging over the entire feature map. This architecture effectively extracts multiscale features and then uses their combination to actually produce the segmentation. In this case, the encoder is made shallow, with a finite size of the feature maps, 8 or 16 times smaller than the input image size:




This leads to feature maps of higher resolution. A good rule of thumb here is to use as simple a decoder as possible since each layer of the decoder is only required to increase the image resolution by a factor of

two, and thanks to the U-Net architecture and spatial pyramid pooling it has quite a lot of input information to do it.

ResNeXt-FPN is basically the Feature Pyramid Network model with ResNeXt, a modern architecture commonly used for object detection (for example, in Faster-RCNN), but adapted for segmentation. Again, the architecture consists of an encoder and a decoder. However, now for segmentation we use feature maps from each decoder level, not just from the last layer with highest resolution.

Since these maps have different sizes, they are resized (to match the largest) and then combined together:

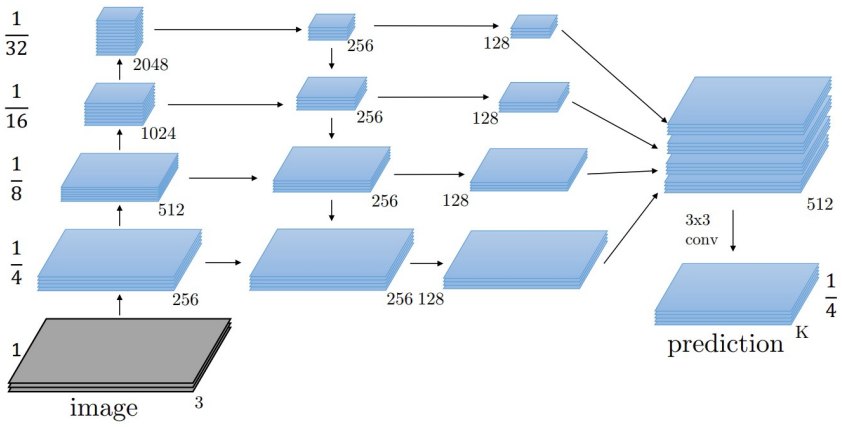



Image source

This architecture has long been known to work very well for segmentation, taking first place in the COCO 2017 Stuff Segmentation Task.

## DeepGlobe Building Detection Challenge

The task in the Building Detection Challenge is, rather surprisingly, to detect buildings. At this point an astute reader might wonder why we need to solve this problem at all. After all, there are a lot of cartographic services where these buildings are already labeled. In the world where you can go to Google Maps and find your city mapped out to the last detail, how is building detection a challenge?

Well, the astute reader would be wrong. Yes, such services exist, but usually they label buildings manually. First of all, this means that

labeling costs *a lot* of money, and cartographic services run into the main bottleneck of modern AI: they need lots of real data that can be produced only by manual labeling.

Moreover, this is not a one-time cost: you cannot label the map of the world once and forget about it. New buildings are constructed all the time, and old ones are demolished. Satellites keep circling the Earth and producing their images pretty automatically, but who will keep re-labeling them? A high-quality detector would help solve these problems.

But cartography is not the only place where such a detector would be useful. Analysis of urbanization in an area, which relies on the location of the buildings, could be useful for realtor, construction, and insurance companies, and, in fact, ordinary people. Another striking application of building detection, one of the most important in our opinion, is disaster relief: when one needs to find and evaluate destructed buildings as fast as possible to save lives, any kind of automation is priceless.

Let us now go back to the challenge. Satellite images were selected from the SpaceNet dataset provided by DigitalGlobe. Images have 30cm per pixel resolution (which is actually a pretty high resolution when you think about it) and have been gathered by the WorldView-3 satellite. The organizers chose several cities—Las Vegas, Paris, Shanghai, and Khartoum—for the challenge. Apart from color photographs (which means three channels for the standard RGB color model), SpaceNet also contains eight additional spectral channels which we will not go into; suffice it to say that satellite imagery contains much, much more than meets the eye.

How does one evaluate the quality of a detector? For evaluation, the organizers proposed to use the classical F1-score:

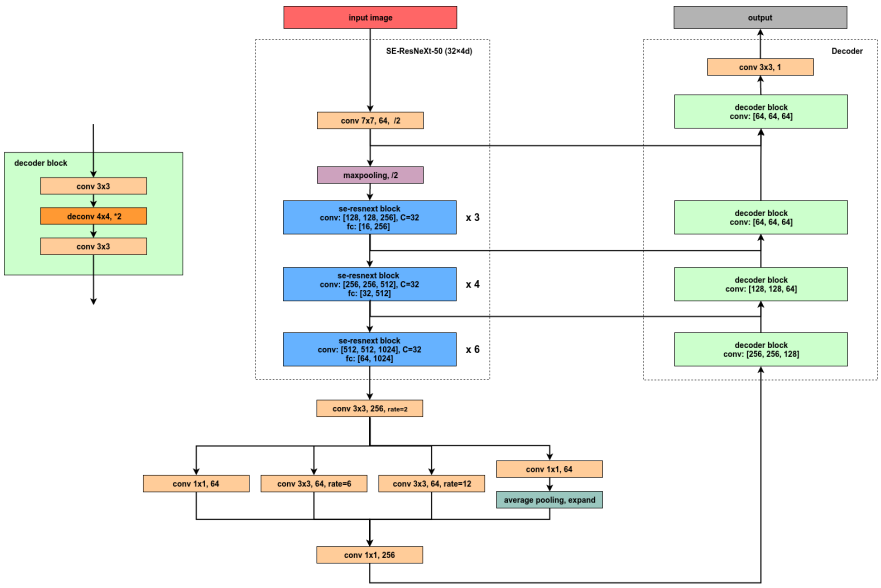
$$F1 = 2\frac{precision \cdot recall}{precision + recall} = 2\frac{\frac{TP}{M} \cdot \frac{TP}{N}}{\frac{TP}{M} + \frac{TP}{N}} = 2\frac{TP}{N+M}$$

Here TP (true positive) is the number of correctly detected polygons of buildings, N is the number of existing real building polygons (unique

for every building), and M is the number of buildings in the solution. A building proposal (a polygon produced by the model) is considered to be a true positive if the real building polygon that has the largest IoU (Intersection over Union) with the proposal has IoU greater than 0.5; otherwise the proposal is a false positive.

In our solution for the segmentation of buildings, we used an architecture similar to U-Net. As the encoder, we used the SE-ResNeXt-50 pretrained on ImageNet. We chose it because this classifier is of high enough quality and does not require too much memory, which is important to maintain a large batch size during training.

To the encoder, we added Atrous Spatial Pyramid Pooling with Image Pooling. The decoder also contains four blocks, each of which is a sequence of convolution, deconvolution, and another convolution. Besides, following the U-Net idea we added skip-connections from the encoder at each level of the decoder. The full architecture is shown in the figure below.




With this model, we did our first experiments and looked at the results… only to find that a good architecture is not enough to produce a good result. The main problem was that in many situations, buildings were clustered very close together. This dense placement made it very difficult for the model to distinguish individual buildings: it usually decided to simply lump them all together into one uber-building. And this was very bad for the competition score (and for the model's results

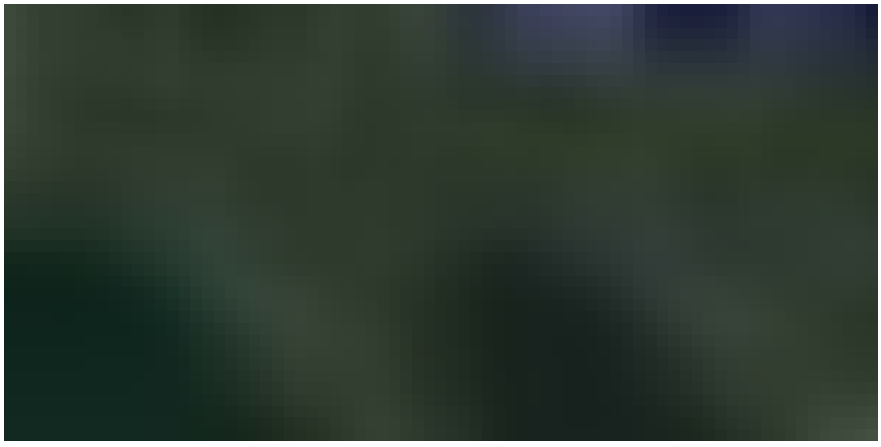
in general) because the score was based on identifying specific buildings rather than classifying pixels.

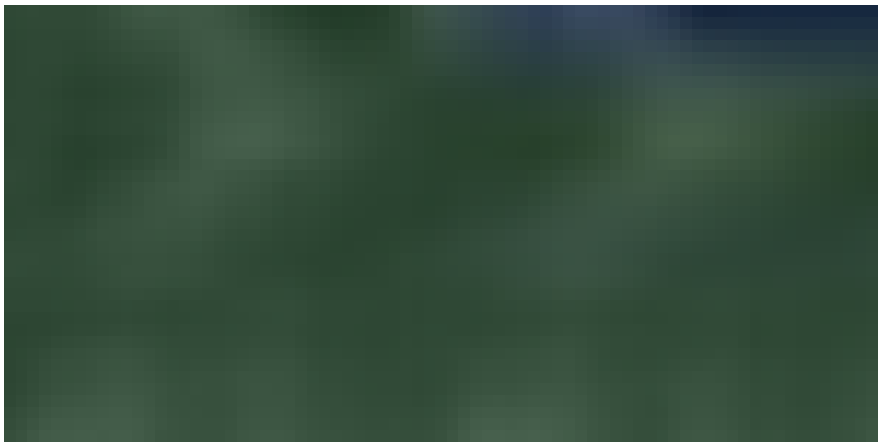
To fix this problem, we needed to do *instance segmentation*, i.e., learn to separate instances of the same class in the segmentation map. After trying several ways of separating the buildings, we decided on a simple but quite effective solution: the watershed algorithm. Since the watershed algorithm needs an initial approximation of the instances (in the form of markers), in addition to the binary segmentation mask our neural network also predicted the normalized pixel distance of the building to its boundary ("energy"). The markers were obtained by binarizing this energy with a threshold. In addition, we increased the input size of images by a factor of two, which allowed to construct more precise segmentation.

As we explained above, we used the sum of binary cross-entropy and the Lovász-Hinge loss function as the objective for the binary mask and the mean squared error for energy. The model was trained on 256x256 crops of input RGB images. We used standard augmentation methods: rotations by multiples of 90 degrees, flips, random scaling, changes in brightness and contrast. We sampled images in the batch based on the value of the loss that was observed on them, so that images with larger error would appear in a training batch more often.

## Results and conclusion

Et voila! In the end, our solution produced building detection of quite good quality:





These animated GIFs show binary masks of the buildings on the left and predicted energy on the right. Naturally, the current detection, even state of the art models such as this one, is still not perfect and we still have a lot to do, but it appears that this quality is already quite sufficient for industrial use.

Let's wrap up: we have discussed in detail one of our papers accepted for the DeepGlobe CVPR workshop. There are two more for land cover classification, i.e., for segmenting satellite images into classes like "forest", "water", or "rangeland"; maybe we will return to them in a later post. Congratulations to everyone who has worked on these models and papers: Alexander Rakhlin, Alex Davydow, Rauf Kurbanov, and Aleksey Artamonov! We have a great team here at Neuromation Labs, and we are sure there are many more papers, competitions, and industrial solutions to come. We'll keep you posted.

*Sergey Golovanov*
*Researcher, Neuromation*

*Sergey Nikolenko*
*Chief Research Officer, Neuromation*