Semantic Segmentation with Deep Learning

A guide and code

George Seif Follow
Sep 19, 2018 · 11 min read ★

Semantic Segmentation

What is semantic segmentation?

Most people in the deep learning and computer vision communities understand what image classification is: we want our model to tell us what single object or scene is present in the image. Classification is very coarse and high-level.

Many are also familiar with object detection, where we try to *locate and classify multiple objects* within the image, by drawing bounding boxes around them and then classifying what's in the box. Detection is mid-level, where we have some pretty useful and detailed information, but it's still a bit rough since we're only drawing bounding boxes and don't really get an accurate idea of object *shape*.

Semantic Segmentation is the most informative of these three, where we wish to classify *each and every pixel in the image*, just like you see in the gif above! Over the past few years, this has been done entirely with deep learning.

In this guide, you'll learn about the basic structure and workings of semantic segmentation models and all of the latest and greatest state-of-the-art methods.

If you'd like to try out the models yourself, you can checkout my Semantic Segmentation Suite, complete with TensorFlow training and testing code for many of the models in this guide!

GeorgeSeif/Semantic-Segmentation-Suite

Semantic Segmentation Suite in TensorFlow. Implement, train, and test new Semantic Segmentation models easily! ...

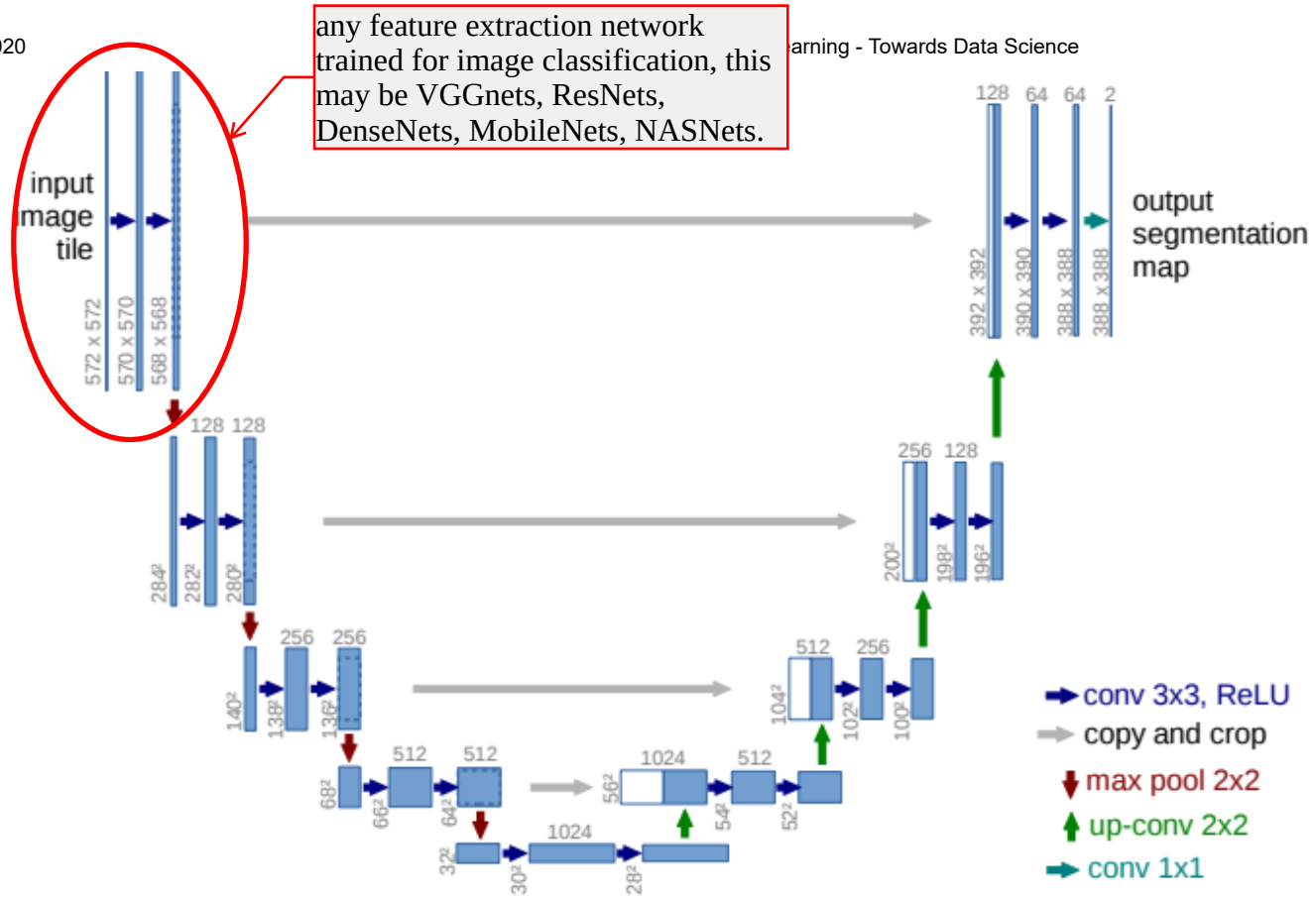
github.com

Basic structure

The basic structure of semantic segmentation models that I'm about to show you is present in all state-of-the-art methods! This makes it very easy to implement different ones, since almost all of them have the same underlying backbone, setup, and flow.

The U-Net model has a great illustration of this structure. The left side of the model represents **any feature extraction network trained for image classification.** This includes networks like VGGNet, ResNets, DenseNets, MobileNets, and NASNets! You can really use anything you want there.

The main thing when selecting your classification network for feature extraction is to keep in mind the tradeoffs. Using a very deep ResNet152 will get you great accuracy but won't be nearly as fast as a MobileNet. The tradeoffs that appear when applying those networks to classification also appear when using them for segmentation. The important thing to remember is that **these backbones will be major drivers when designing / selecting your segmentation network, and I can't stress that enough.**



U-Net model for segmentation

Once those features are extracted they are then further processed at *different scales*. The reason for this is two-fold. Firstly, your model will very likely encounter objects of many different sizes; processing the features at different scales will give the network the capacity to handle those different sizes.

Second, when performing segmentation there is a tradeoff. If you want good classification accuracy, then you'll definitely want to process those high level features from later in the network since they are more discriminative and contain more useful semantic information. On the other hand, if you only process those deep features, you won't get good localisation because of the low resolution!

The recent state-of-the-art methods have all followed the above structure of feature extraction followed by multi-scale processing. As such, many are quite easy to implement and train end-to-end. Your selection of which one to use will depend on your need for accuracy vs speed/memory, as all have been trying to come up with new methods of addressing this tradeoff, while maintaining efficiency.

In the following walkthrough of the state-of-the-art I'm going to focus on the latest methods, since these will be the most useful to the most readers after understanding the basic structure above. We will walkthrough in rough chronological order, which also roughly corresponds to the advancing of the state-of-the-art.

State-of-the-art walkthrough

Full-Resolution Residual Networks (FRRN)

The FRRN model is a very clear example of the multi-scale processing technique. It accomplishes this using 2 separate streams: the residual stream and the pooling stream.

We want to process those semantic features for higher classification accuracy, so FRRN progressively processes and downsamples the features maps in the pooling stream. At the same time, it processes the feature maps at *full resolution* in the residual stream. So the pooling stream handles the high-level semantic information (for high classification accuracy) and the residual stream handles the low-level pixel information (for high localisation accuracy)!

Now since we are training an end-to-end network, we don't want these 2 streams to be totally disconnected. So after each max pooling, FRRN does some joint processing of the features maps from the 2 streams to combine their information.

FRRN model structure

Pyramid Scene Parsing Network (PSPNet)

The FRRN did a great job of directly performing the multi-scale processing. But doing heavy processing at *every scale* is quite computationally intensive. Moreover, FRRN does some processing at full resolution, very slow stuff!

The PSPNet proposes a clever way to get around this by using *multiple scales of pooling*. It starts off with a standard feature extraction network (ResNet, DenseNet etc) and takes the features of the third downsampling for further processing.

To get the multi-scale information, PSPNet applies 4 different max pooling operation with 4 different window sizes and strides. This effectively captures feature information from 4 different scales without the need for heavy individual processing of each one! We simply do a lightweight convolution on each one after, upsample so each feature map is at the same resolution, and concatenate them all.

Voila! We've combined multi-scale feature maps without apply many convolutions on them!

All of this is done on the lower resolution feature maps for high-speed. At the end, we upscale the output segmentation map to the desired size using bilinear interpolation . This technique of upscaling only after all processing is done is present in many state-of-the-art works.

PSPNet model structure

The One Hundred Layers Tiramisu (FCDenseNet)

If there's one awesome trend that deep learning has brought about, it's awesome research paper names! The Hundred Layers Tiramisu FCDenseNet (sounds delicious!) uses a similar structure as the U-Net architecture we saw before. The main contribution is the clever use of of dense connections similar to that of the DenseNet classification model.

This really emphasises the strong trend in computer vision where the feature extraction frontend is the main backbone for performing well on any other task. Thus, the first place to look for accuracy gains is often your feature extraction frontend.

FCDenseNet model structure

Rethinking Atrous Convolution (DeepLabV3)

The DeepLabV3 and is yet another clever way to do multi-scale processing, this time **without increasing parameters**.

This model is very lightweight. We again start with a feature extraction frontend, taking the features of the 4th downsampling for further processing. This resolution is very low (16 times smaller than the input) and thus it would be great if we can just process here! The tricky part is that at such a low resolution, it's hard to get good localisation due to poor pixel accuracy.

That's where DeepLabV3's main contribution comes in, with the clever use of the *Atrous* convolutions. Regular convolutions can only handle very local information since the weights are always right next to each other. For example, in a standard 3x3 convolution, the distance between one weight to any other one is only a single step / pixel.

With atrous convolutions we will directly increase the spacing between the convolution weights, without actually increasing the *number* of weights in the operation. So we're still using a 3x3 with 9 total parameters, we're just spacing the weights we multiply with further apart! The distance between each weight is called the *dilation rate*. The diagram of the model below illustrates this idea well.

When we use a low dilation rate, we will process very local / low-scale information. When we use a high dilation rate, we process more global / high-scale information. Thus, the DeepLabV3 model mixes atrous convolutions with different dilation rates to capture multiscale information.

The technique of upscaling at the end after all processing as was explained for PSPNet is done here as well.

DeepLabV3 model structure

Multi-Path Refinement Networks (RefineNet)

We saw before how the FRRN did a great job of directly combining information from multiple resolutions and combining them all together. The drawback was that the

processing at such a high resolution was computationally intensive and we still had to process and combine those features with the low resolutions ones too!

The RefineNet model says that we don't need to do this. When we run our input image through the feature extraction network, we naturally get multi-scale feature maps after each downsampling.

RefineNet then processes these multi-resolution feature maps in a bottom up fashion to combine the multiscale information. First, each feature map is processed independently. Then as we upscale we combine the low-resolution feature map with the higher resolution one, doing some further processing on both of them together. Thus, the multi-scale feature maps are processed both independently and together. The whole process moves from left to right in the diagram below.

The technique of upscaling at the end after all processing as was explained for PSPNet and DeepLabV3 is done here as well.

RefineNet model structure

Large Kernel Matters (GCN)

Previously, we saw how the DeepLabV3 model used atrous convolutions with different dilation rates to capture multi-scale information. The tricky part with this is that we can only process a single scale at a time and then have to combine them later. An atrous convolution with a rate of 16 for example won't do well with local information and must

later be combined with information from a convolution with a much smaller rate in order to perform well in semantic segmentation.

Thus, in previous methods the multi-scale processing happens separately first and then the results are combined together. It would make more sense if we could get the multi-scale information in one shot.

To do this, the Global Convolutional Network (GCN) cleverly proposes to use large one-dimensional kernels instead of square ones. With square convolutions like 3x3, 7x7, etc, we can't make them too big without taking a massive speed and memory consumption hit. One dimensional kernels on the otherhand scale much more efficiently and we can make them quite big without slowing down the network too much. The paper even went all the way up to a size of 15!

The important thing you have to be sure to do is balance the horizontal and vertical convolutions. In addition, the paper does use small 3x3 convolutions with a low filter count for efficient refinement of anything the one dimensional convs might miss.

GCN follows the same style as previous works by processing each scale from the feature extraction frontend. Because of the efficiency of the one-dimensional convolutions, GCN performs processing on all scales all the way up to the full resolution, rather than staying small and upscaling afterwards. This allows for constant refinement of the segmentation as we scale up, rather than the possible bottle-necking that may occur due to staying at a lower resolution.

GCN model structure

DeepLabV3+

The DeepLabV3+ model, as the name suggests, is a quick extension of DeepLabV3, borrowing some conceptual ideas from the advances that came before it. As we saw before, there is a potential bottleneck if we simply wait to upscale with bilinear interpolation at the end of the network. In fact, the original DeepLabV3 model upscaled by x16 at the end!

To address this, DeepLabV3+ proposes to add an intermediate decoder module on top of the DeepLabV3. After processing via DeepLabV3, the features are then upsampled by x4. They are then further processed along with the original features from the feature extraction frontend, before being upscaled again by x4. This lightens the load from the end of the network and provides a shortcut path from the feature extraction frontend to the near end of the network.

DeepLabV3+ model structure

CVPR and ECCV 2018

The networks we went through in the previous section represent the bulk of the techniques you'll need to know to do Semantic Segmentation! Much of the things released this year in the computer vision conferences have been minor updates and small bumps in accuracy, not extremely critical to getting going. For thoroughness, I provide a quick review of their contributions here for anyone who's interested!

Image Cascade Network (ICNet) — Uses deep supervision and runs the input image at different scales, each scale through their own subnetwork and progressively combining the results

Discriminative Feature Network (DFN) — Uses deep supervision and attempts to process the smooth and edge portions of the segments separately

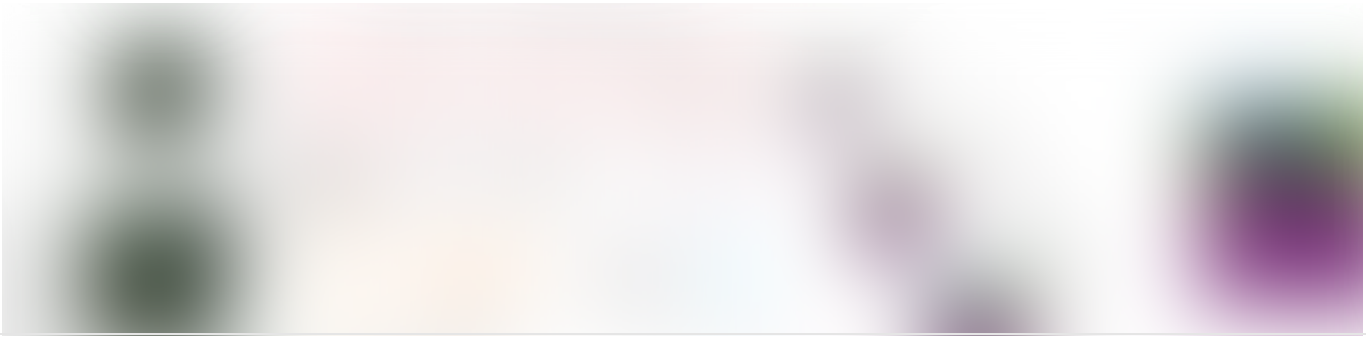
DenseASPP — Combines dense connections with atrous convolutions

Context Encoding — Leverages global context to increase accuracy by adding a channel attention module, which triggers attention on certain feature maps based on a newly designed loss function. The loss is based on a network branch which predicts which classes are present in the image (i.e higher level global context).

Dense Decoder Shortcut Connections — Use dense connections in the decoding stage for higher accuracy (previously only done during feature extraction / encoding)

Bilateral Segmentation Network (BiSeNet) — Has 2 branches: one is deep for getting semantic information, while the other does very little / minor processing on the input image as to preserve the low-level pixel information

ExFuse — Uses deep supervision and explicitly combines the multi-scale features from the feature extraction frontend *before* processing, in order to ensure multi-scale information is processed together at all levels.



ICNet model structure

TL;DR or how to do semantic segmentation

- Be aware of classification net tradeoffs. Your classification network is your main driver for processing features and most of your gains / losses will come from here
- Process at multiple scales and combine the information together
- Multiscale pooling, atrous convs, large one-dimensional convs are all good for semantic segmentation
- You don't need to do lots of processing at a high resolution. Do most of it at low res for speed, then upscale and do light processing at the end if necessary
- Deep supervision can bump up your accuracy a bit (trickier to setup training though)

. . .

Like to learn?

Follow me on twitter where I post all about the latest and greatest AI, Technology, and Science! Connect with me on LinkedIn too!

Recommended Reading

Want to learn more about Deep Learning? The ***Deep Learning with Python*** book will teach you how to do *real* Deep Learning with the easiest Python library ever: Keras!

And just a heads up, I support this blog with Amazon affiliate links to great books, because sharing great books helps everyone! As an Amazon Associate I earn from qualifying purchases.