Modeling

In modeling discussion, we will discuss two key aspects.
1. SOTA Segmentation Models: What are the state-of-the-art segmentation models and their architectures?
2. Transfer Learning: How can you use these models to train a better segmenter for the self-driving car data?

We'll cover the following

SOTA segmentation models

Machine learning in general and deep learning, in particular, have progressed a lot in the domain of computer vision-based applications during the last decade. The models enlisted in this section are the most commonly used deep neural networks that provide state-of-the-art (SOTA) results for *object detection and segmentation tasks*. These tasks form the basis for the self-driving car use case.

FCN

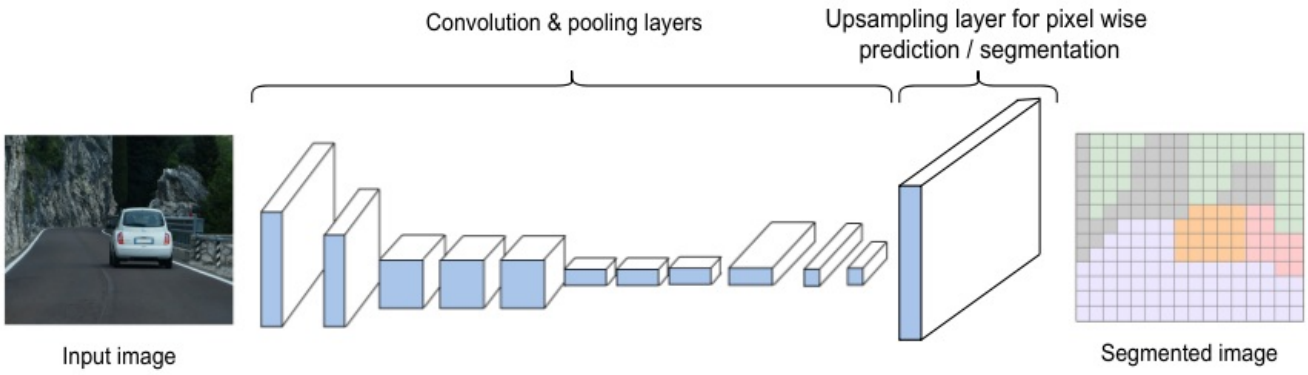
Fully convolutional networks (FCNs) are one of the top-performing networks for semantic segmentation tasks.

> Segmentation is a dense prediction task of pixel-wise classification.

A typical FCN operates by fine-tuning an image classification CNN and applying pixel-wise training. It first compresses the information using multiple layers of convolutions and pooling. Then, it up-samples these feature maps to predict each pixel's class from this compressed information.

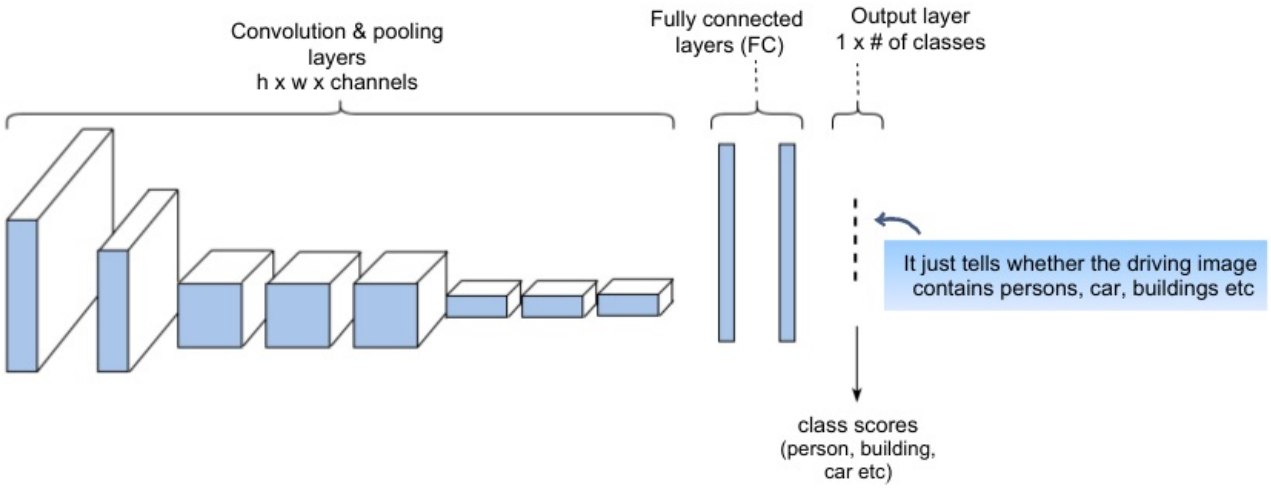
FCN architecture

Let's see how this FCN is made from a CNN trained for image classification



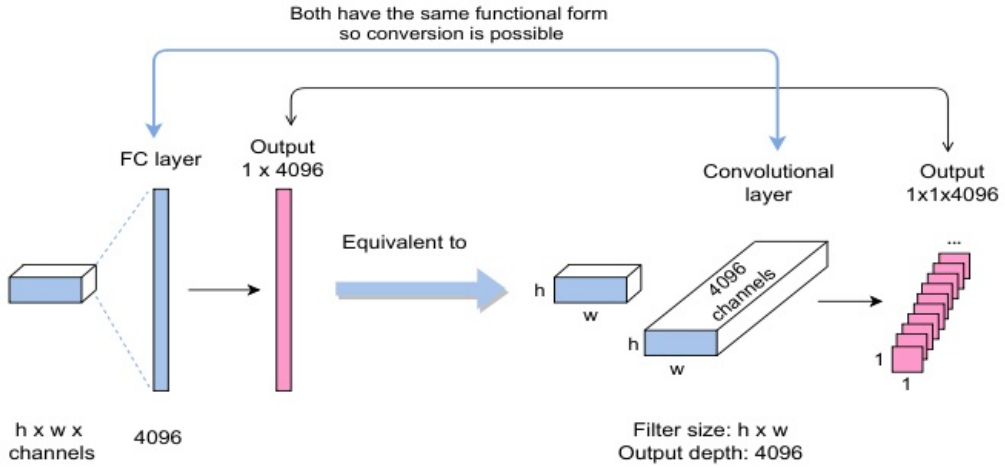
FCN Architecture

CNN trained for image classification



A typical architecture of a CNN-based object classifier

Converting Fully connected (FC) layers of CNN to Convolutional layers

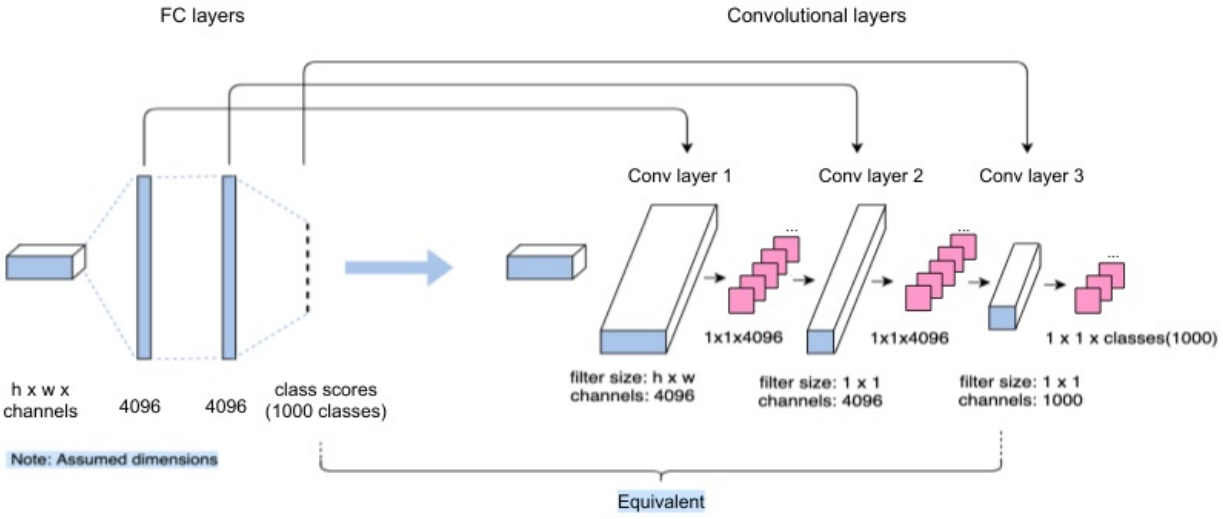


Note: Assumed dimensions

FC layers in CNN can be viewed as convolutions with kernels that cover their entire input regions

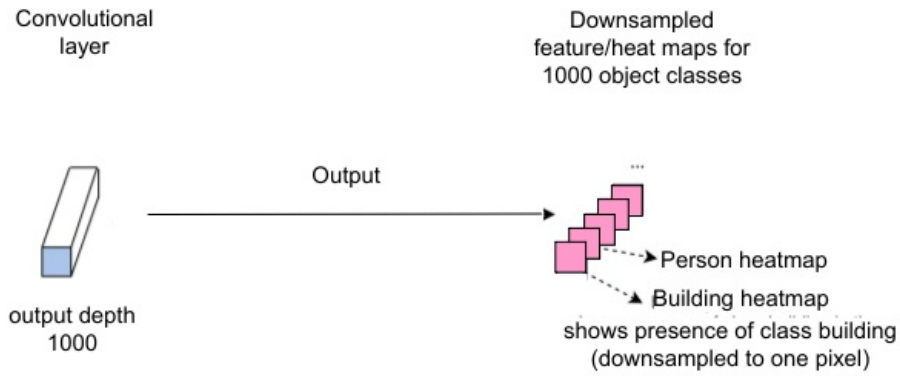
FC layers are equivalent to convolutions with kernels that cover their entire input regions

Converting FC layers to convolutional layers form the
basis of the dense (pixel-wise) prediction aka segmentation



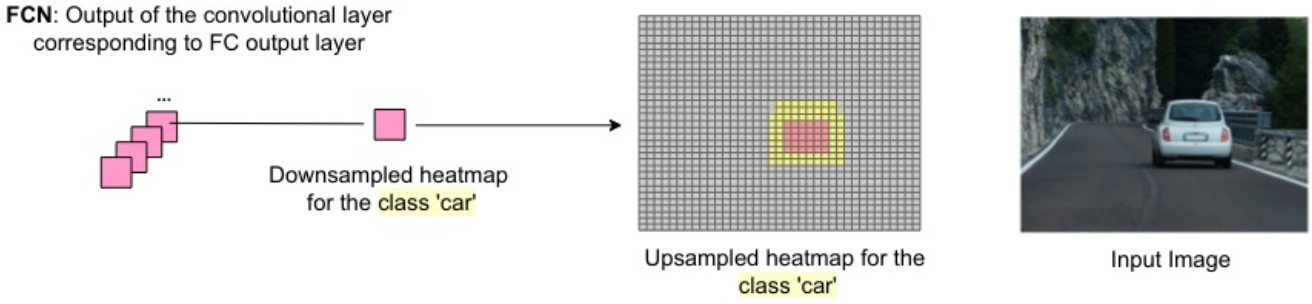
FC layers replaced with equivalent convolutional layers that cover entire input regions

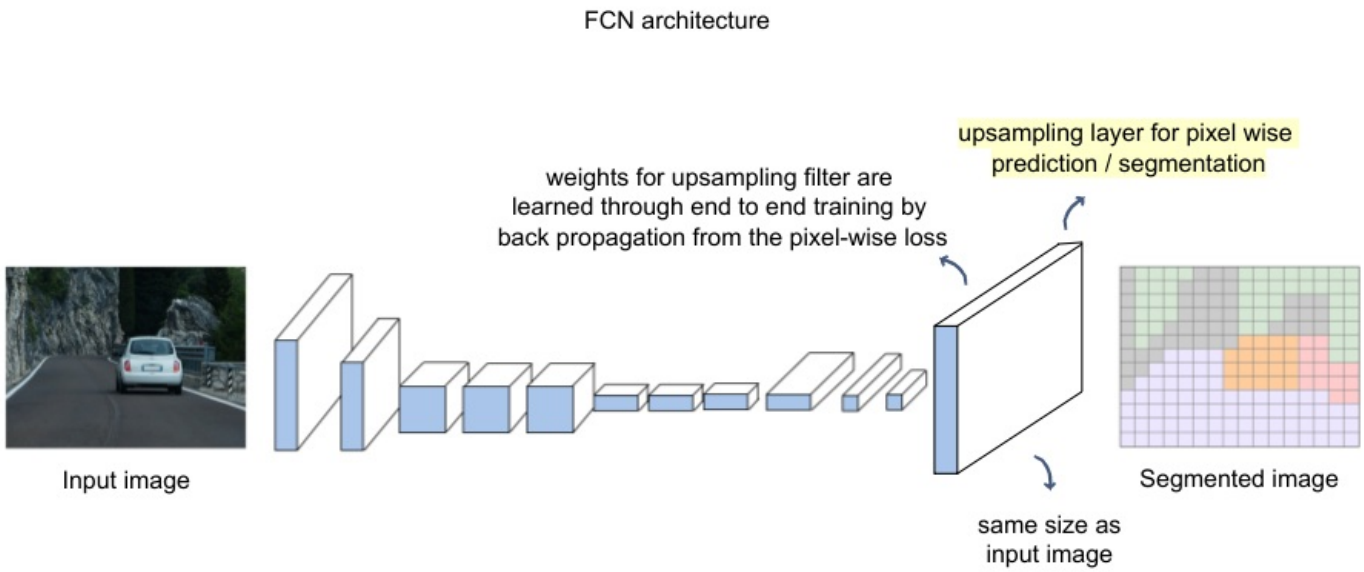
We need to upsample the feature maps to obtain pixel wise segmentation equal to the size of input image



Equivalent of Convolutional layer to downsampled feature maps

We need to upsample the heatmaps to obtain pixel wise segmentation same as the size of the input image

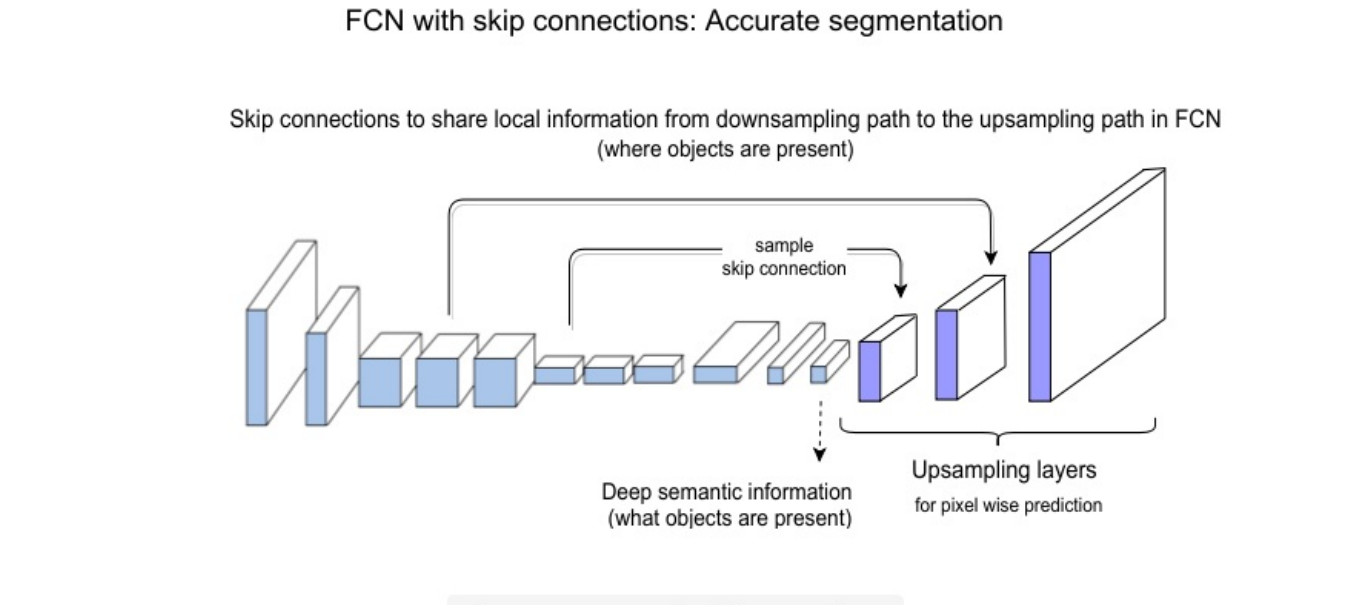




Upsampling layer is added at the end of the convolution and pooling process to obtain pixel wise segmentation from downsampled feature/heat maps

7 of 7

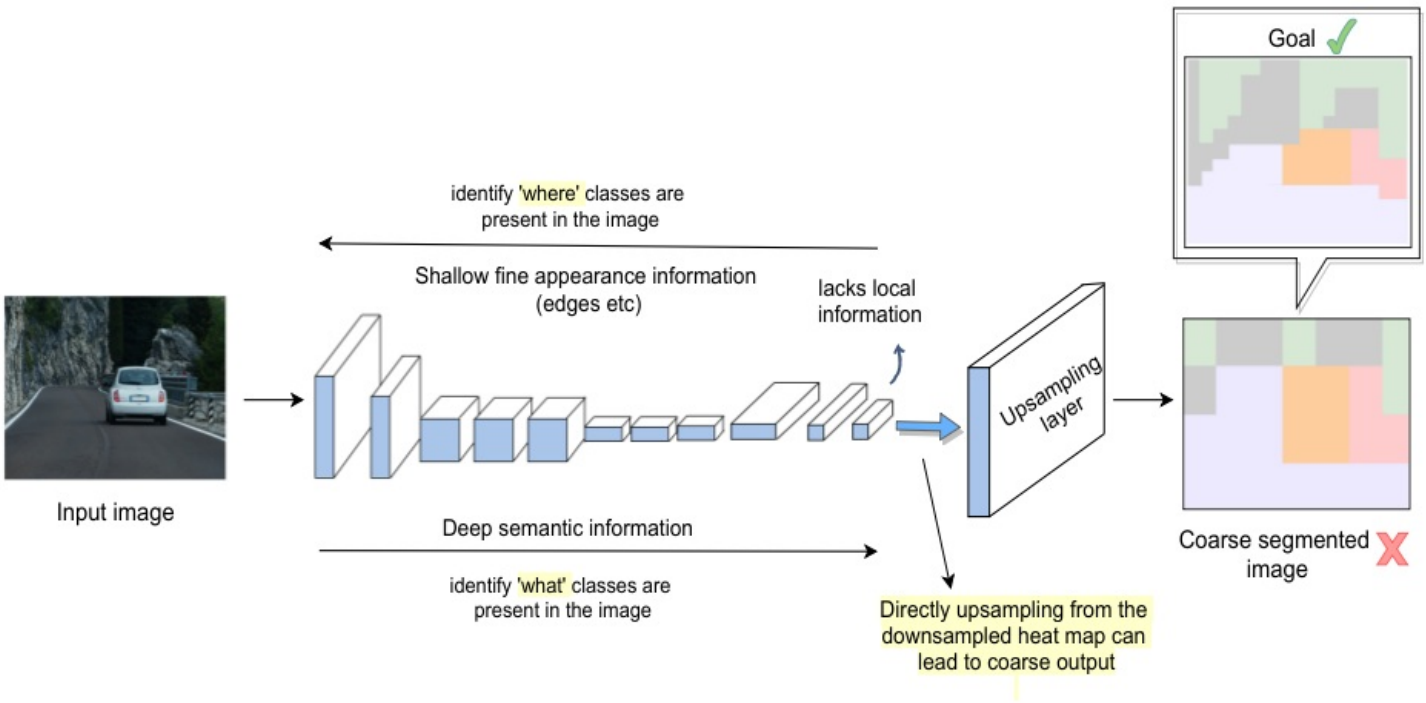
The convolutional layers at the end (instead of the fully connected layers) also allow for dynamic input size. The output pixel-wise classification tends to be coarse, so you make use of skip connections to achieve good edges. The initial layers capture the edge information through the deepness of the network. You use that information during our upsampling to get more refined segmentation.



Accurate segmentation: Skip connections

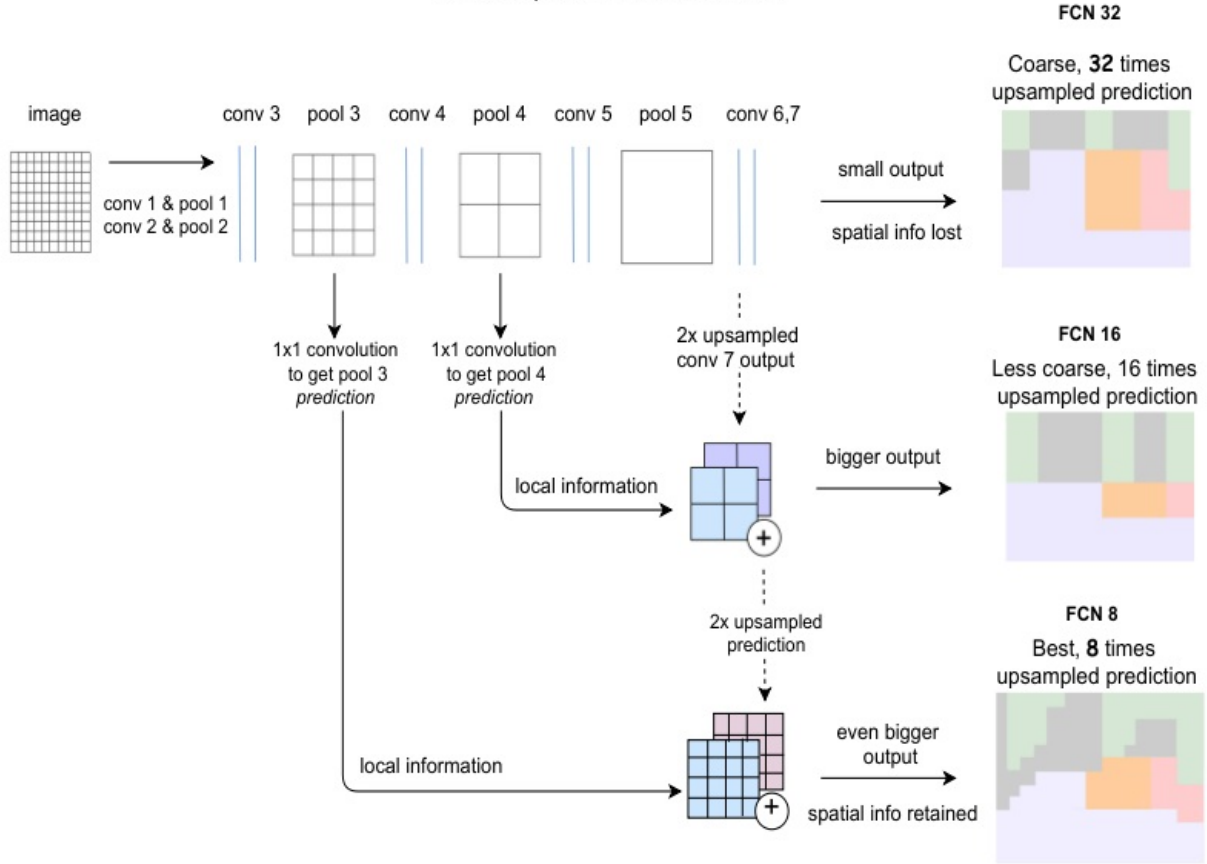
1 of 3

FCN without skip connections: Coarse segmentation



Coarse segmentation: No skip connection

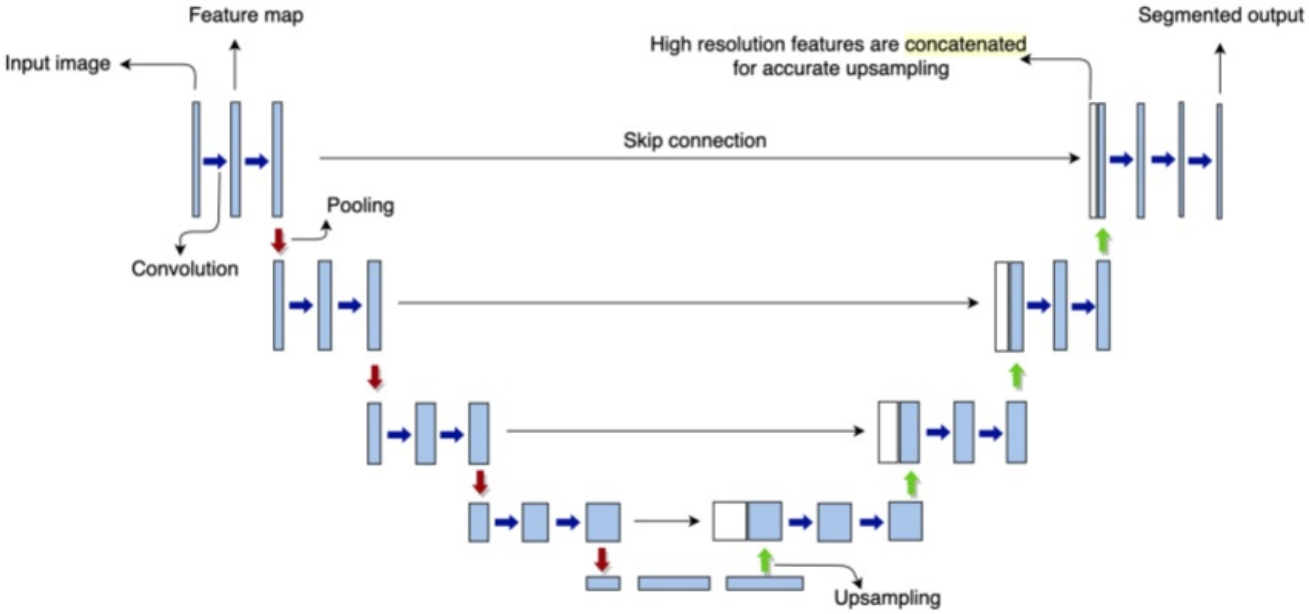
How skip connections work



How skip connections work

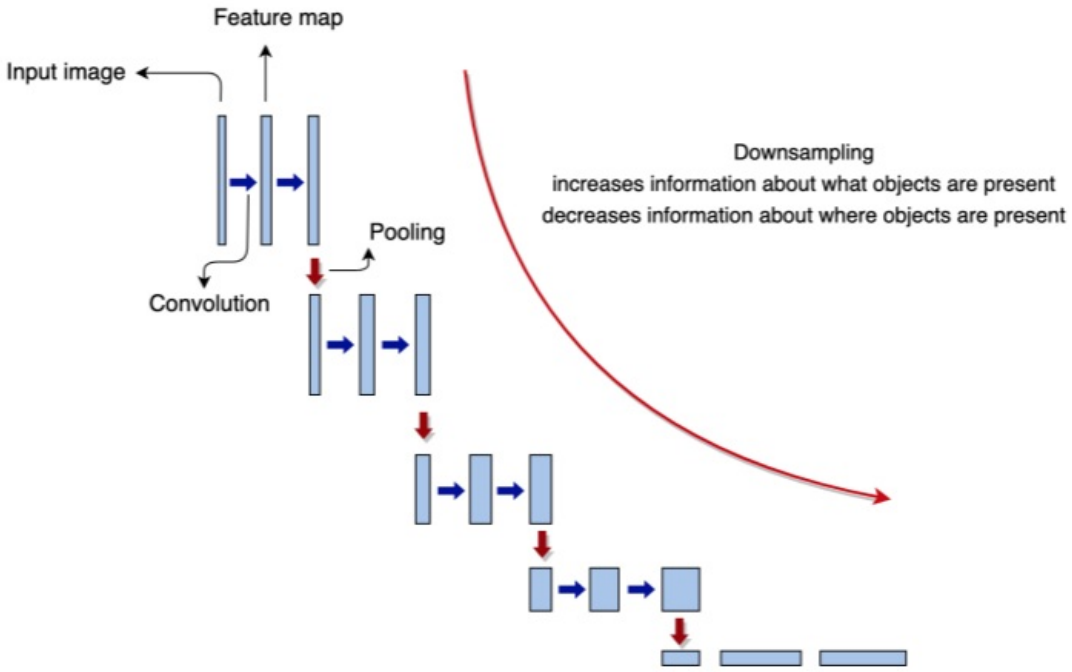
U-Net

U-Net is commonly used for semantic segmentation-based vision applications. It is built upon the FCN architecture with some modifications. The architectural changes add a powerful feature in the network to require less training examples. The overall architecture can be divided into two halves. The first half down-samples the convolutional features through the pooling operation. The second half up-samples the feature maps to generate the output segmentation maps. The upsampling process makes use of skip connections to concatenate high-resolution features from the downsampling portion. This allows for accurate upsampling.



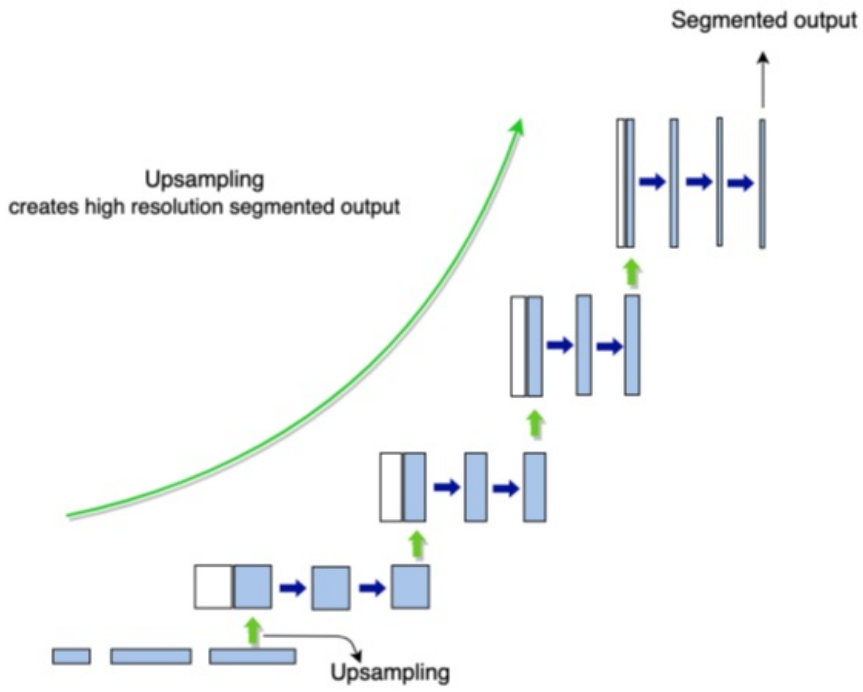
U-Net architecture: like an FCN it does not have FC layers

1 of 4



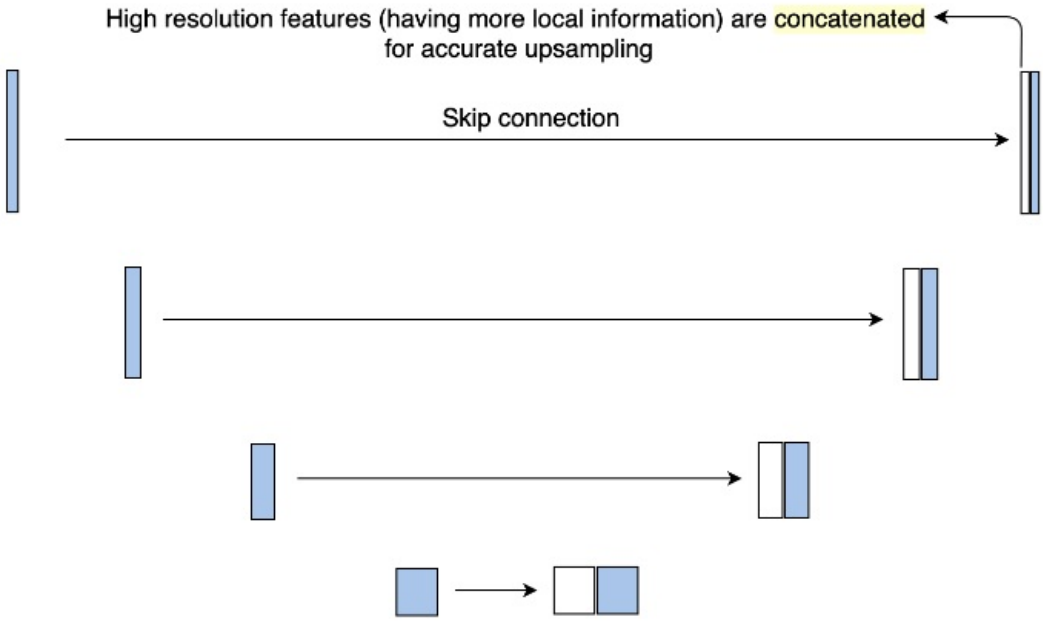
Downsampling

2 of 4



Upsampling

3 of 4



Using skip connections during upsampling allows strong semantic features at each resolution scale

4 of 4

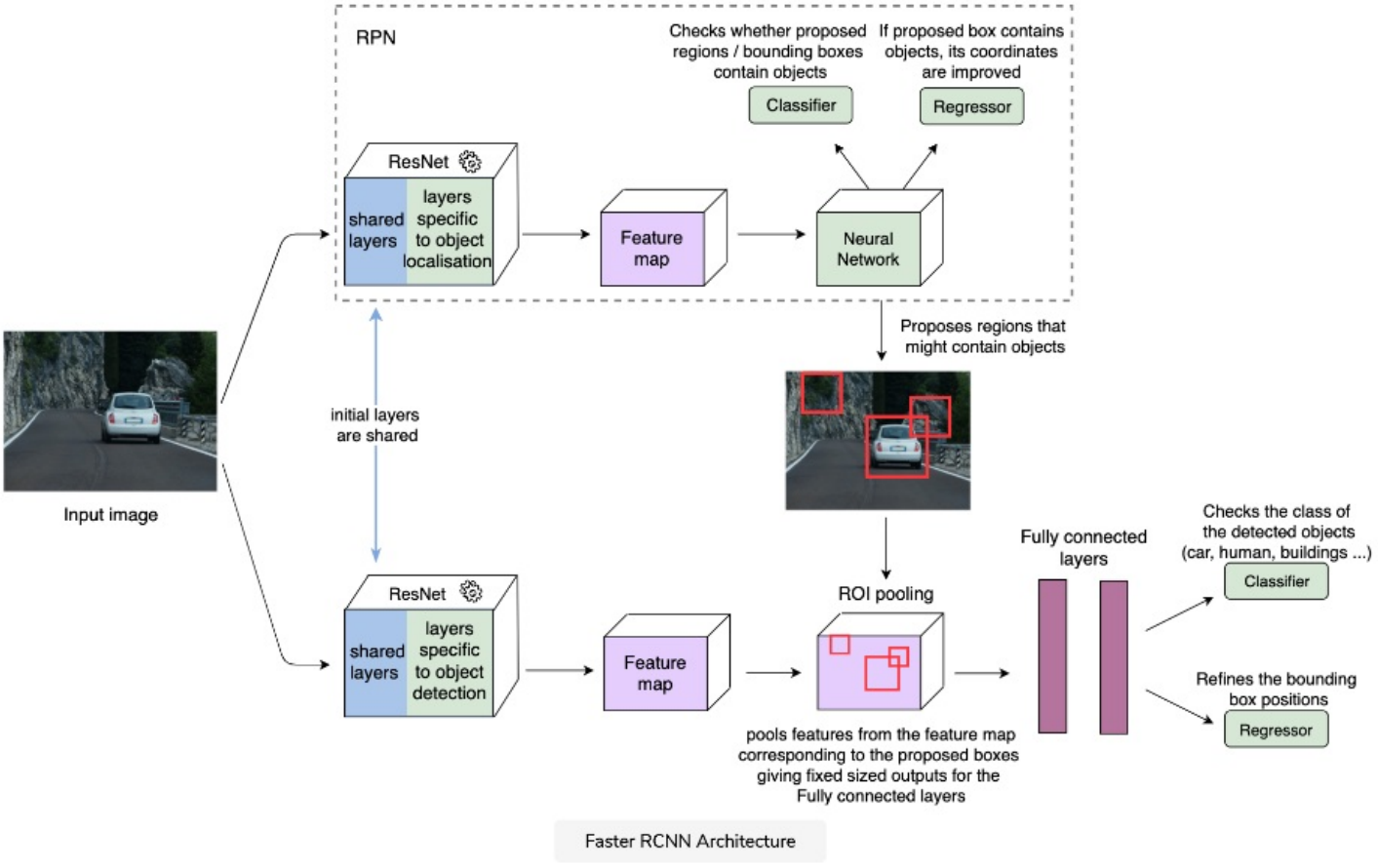
Mask R-CNN

Mask R-CNN is a powerful deep neural network that is widely used for many real-world instance segmentation applications. It combines the best of both the worlds: Faster R-CNN for object detection and localization and FCN for pixel-wise instance segmentation of objects.

Minimize

Faster R-CNN

It is a state-of-the-art deep neural network that proposes regions in the image which contain different objects by drawing bounding boxes around them. The objects are detected by a fully-convolutional **Region Proposal Network (RPN)** that generates predictions on image regions through a sliding window fashion. RPN receives image features from a backbone CNN-based image classifier, e.g., ResNet. The following figure shows the block diagram architecture of faster R-CNN.



Faster RCNN Architecture

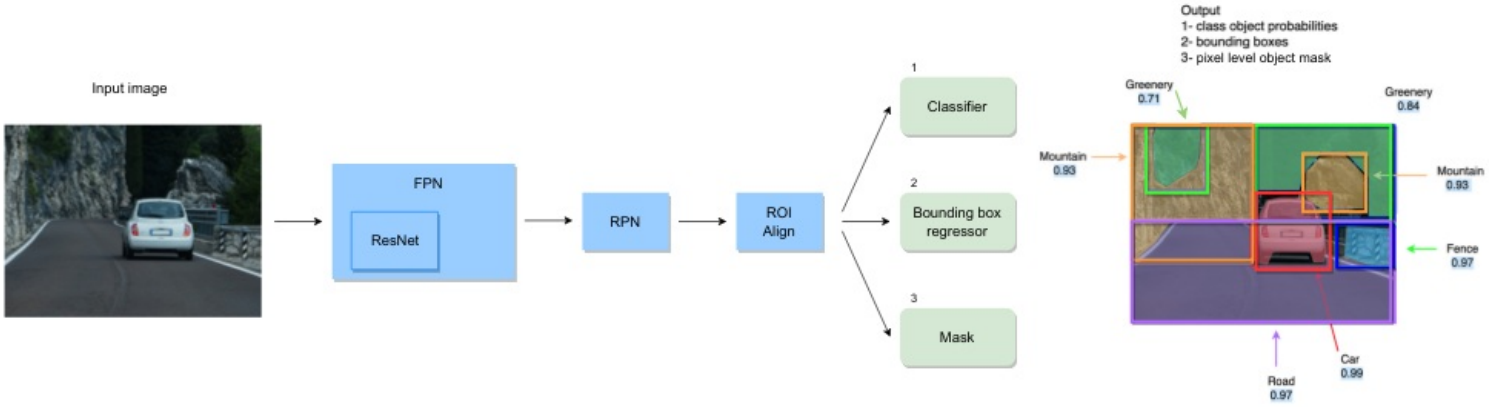
Mask R-CNN architecture is constructed by using a convolutional backbone of a powerful CNN classifier (e.g. ResNet) followed by a Feature Pyramid Network (FPN). FPN extracts feature maps from the input image at different scales, which are fed to the Region Proposal Network (RPN). RPN applies a convolutional neural network over the feature maps in a sliding-window fashion to predict region proposals as bounding boxes that will contain class objects. These proposals are fed to the RoI Align layer that extracts the corresponding ROIs (regions of interest) from the feature maps to align them with the input image properly. The ROI pooled outputs are fixed-size feature maps that are fed to parallel heads of the Mask R-CNN.

To reduce the computational cost, Mask R-CNN has three parallel heads to perform

1. Classification

2. Localization
3. Segmentation

The output layer of the classifier returns a discrete probability distribution of each object class. Localisation is performed by the regressor whose output layer generates the four bounding-box coordinates. The third arm consists of an FCN that generates the binary masks of the predicted objects. The following is a block diagram of the Mask R-CNN.



Mask RCNN

> It is worth mentioning that most of the deep vision architectures are not inherently designed to preserve sensitivity for different scales, sizes and rotations of the objects/images. The pooling operation in these networks is only able to capture the translational variance of the labeled objects/images. Hence, it is critical to cover the variation in scale and rotation of your training images and keep the testing image size similar to the image size in your training dataset.

Transfer learning

Let's answer the question of how to utilize the discussed SOTA segmentation models to build one for the self-driving car.

Transfer learning is a widely used technique in the field of deep learning: utilizing pre-trained powerful deep neural networks (DNNs) for weak or small datasets. When you kickstart a deep learning project with a small ground-truth (human-annotated) training data, it is often challenging to scale the performance of your model on a standard-sized large training dataset. Since increasing the size of human-annotated data can be time-consuming, transfer learning helps take advantage of a powerful deep neural network that is pre-trained on a different but large human-annotated dataset that is similar in nature to the dataset in hand.

> We use transfer learning to build upon learned knowledge from one dataset to improve learning in the dataset.
>
> The trick of transfer learning can be explained through the functionality of the human vision system, which is the basis of convolutional and pooling layers in deep networks. There is a striking similarity between the low and high-level feature detectors (simple and complex neurons) of the primate vision

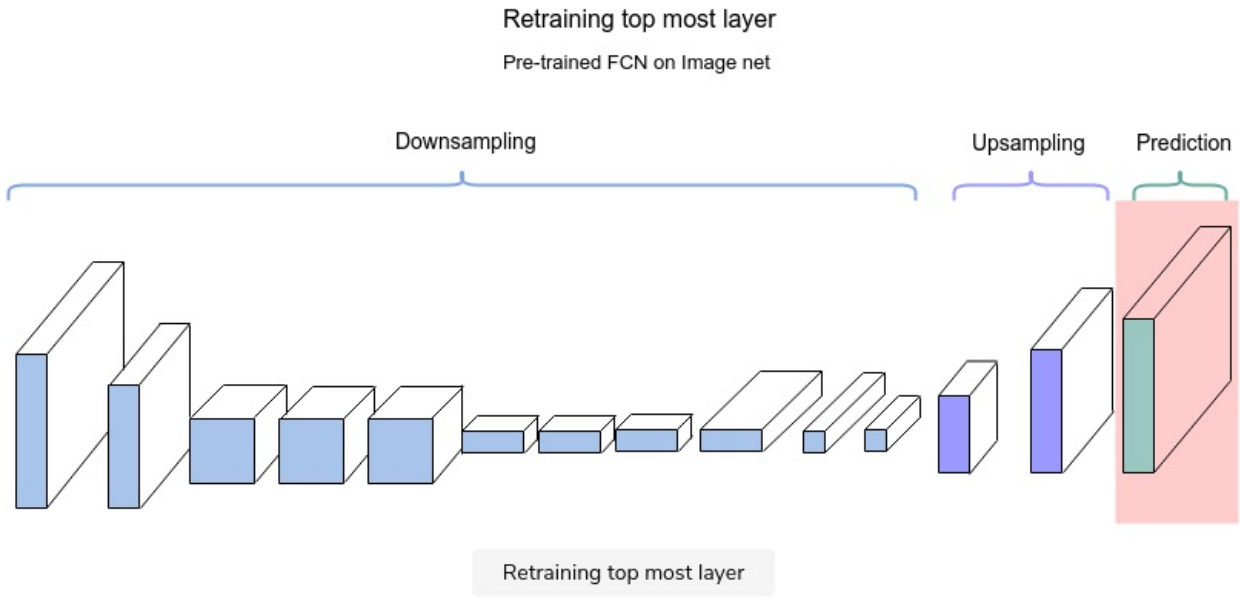
> system and convolutional and pooling operations in deep networks.

In a trained model, the low-level feature detectors are fine-tuned to filter high-frequency visual information (edges), which are mostly common in natural images. For example, you can transfer the learned edge detectors of a model trained to detect cats to detect dogs. For more similar objects/images, this similarity can also be correlated in high-level feature detectors. In this regard, freezing the layers of a trained deep neural network in transfer learning helps take advantage of the learned feature detectors from a large and rather similar natural imaging dataset.

Assume that, from the above-mentioned SOTA models, you want to utilise a pre-trained FCN (on ImageNet dataset) for performing segmentation for your self-driving vehicle. Let's see three different approaches where you can apply transfer learning.

Retraining topmost layer

You can take advantage of the large features' bank in the ImageNet FCN and re-train it for your smaller driving images dataset. In this case, you need to update the final pixel-wise prediction layer in the pre-trained FCN (i.e., replace the classes in ImageNet trained model with classes in driving image set) and retrain the final layer through an optimizer while freezing the remaining layers in the FCN.



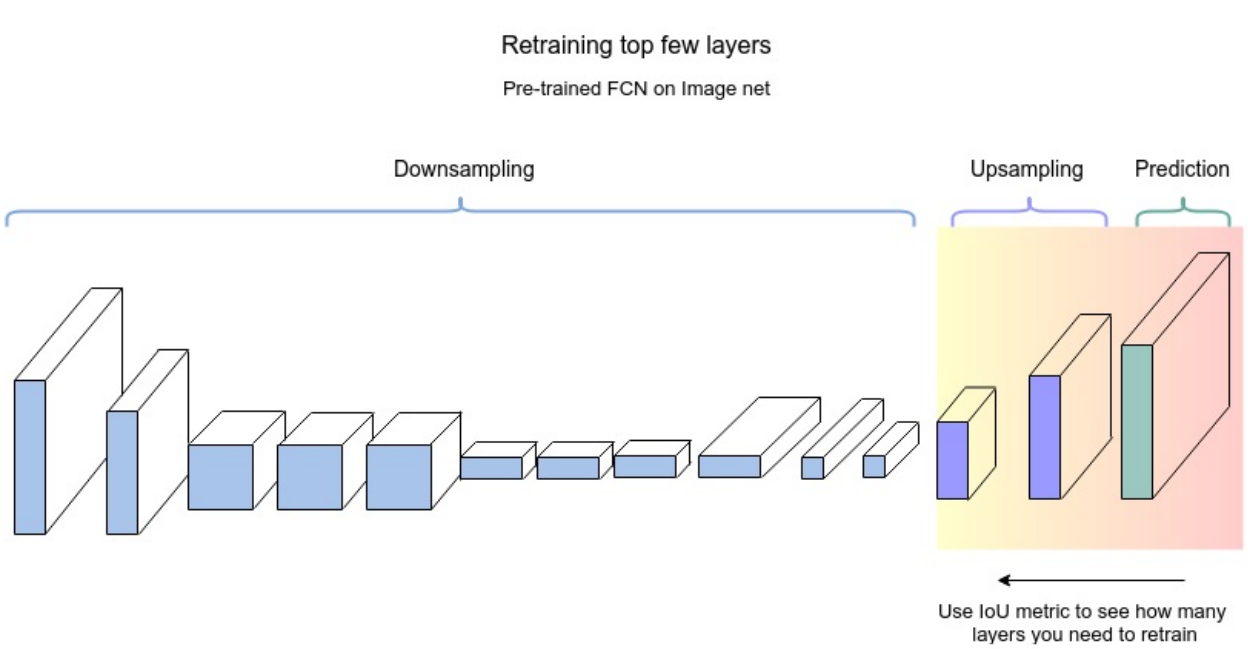
Retraining top most layer

This approach makes the most sense when the driving images data is limited and/or you believe that the current learned layers capture the information that you need for making a prediction.

Retraining top few layers

Similarly, in cases where you have a medium-sized driving images dataset available, you can also try retraining some deeper upsampling layers to improve the accuracy of your segmenter.

You can *experiment with how many upsampling layers you need to retrain* by looking at the *IoU metric* to see if retraining any further improves the performance of the segmentation model.



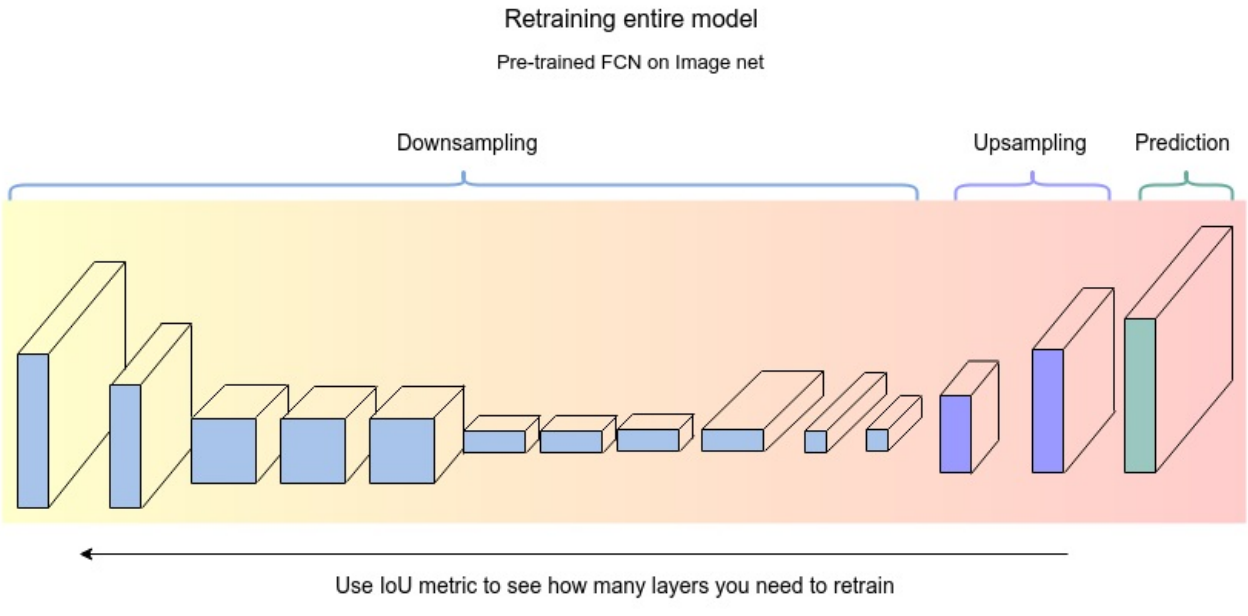
Retraining top few layers

This approach makes the most sense when you have a decent amount of driving images data. Also, shallow layers generally don't need training because they are capturing the basic image features, e.g., edges, which don't need retraining.

Retraining entire model

As the size of your dataset increase, you can consider retraining even more layers or retraining the entire model. Once again, you can use the IoU metric to evaluate the optimal number of layers whose retraining can increase model performance.

Generally, retraining the entire network is laborious and time-consuming. It is usually done only if the dataset under consideration has completely different characteristics from the one on which the network was pre-trained.



Retraining entire model

On top of choosing or designing the optimal DNN architecture, one can also be creative in adding some extra

features in the network to handle the data bias and variance issues. For instance, adding a regularizer (e.g., L1/L2) after a DNN architecture, making a hyper-parameter sweep for fine-tuning and experimenting with different optimizers (e.g., Adam and SGD) during training can further enhance the performance of the model.

← Back
Training Data Generation

Next →
Problem Statement

Mark as Completed

Report an Issue Ask a Question