FULLY CONNECTED NETWORK

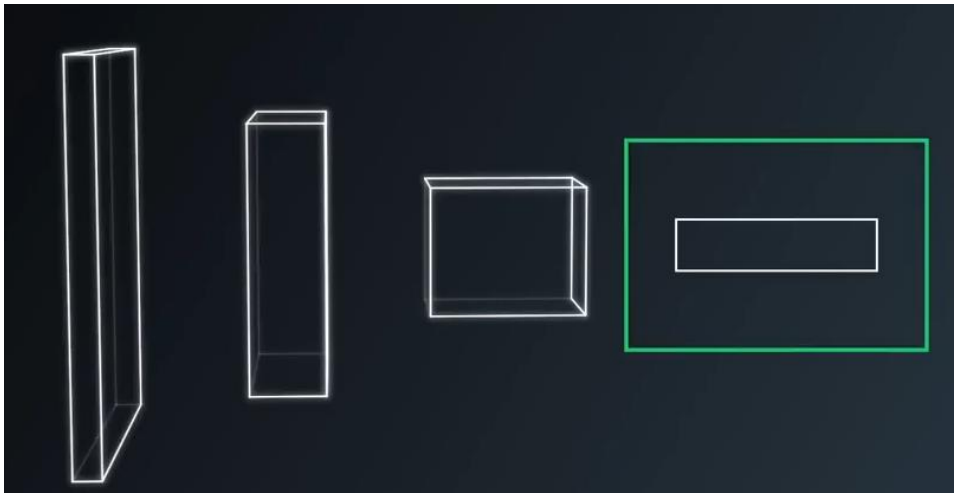
We will cover fully convolutional networks,

1- **Object detection**
2- **Semantic segmentation**
3- **Inference optimizations**

Finally, you will apply this concept and technique to train and supervise semantic segmentation network and test result on automotive video.

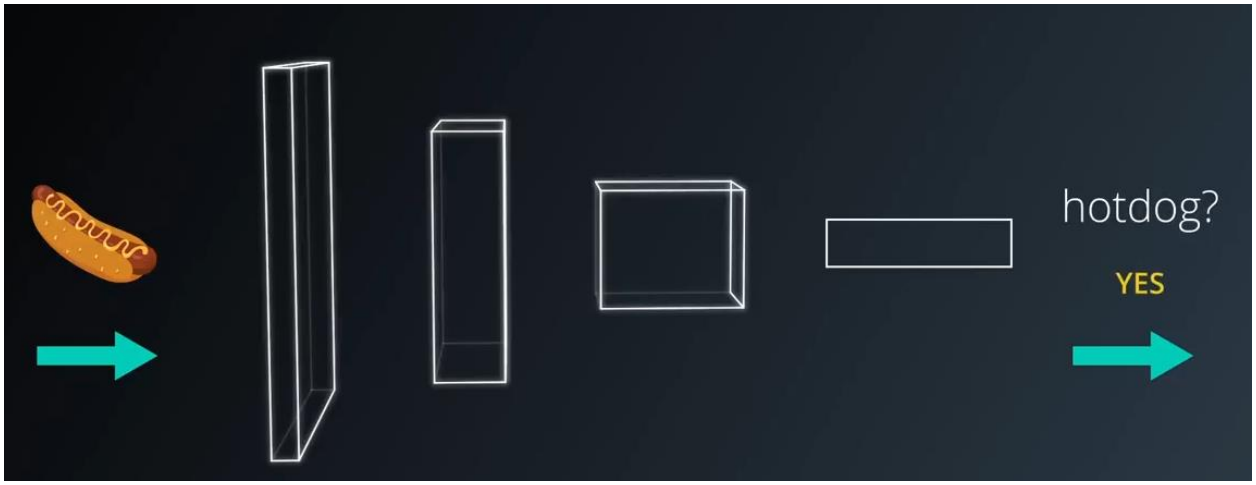
The network will be study the leading edge of research into deep learning for autonomous driving.

Series of convolutional neural network followed by fully connected layers and ultimately a soft max activation function.



This is a great architecture for classification task like

1- Is this picture is of a hotdog? But you want to to change our task ever so slightly

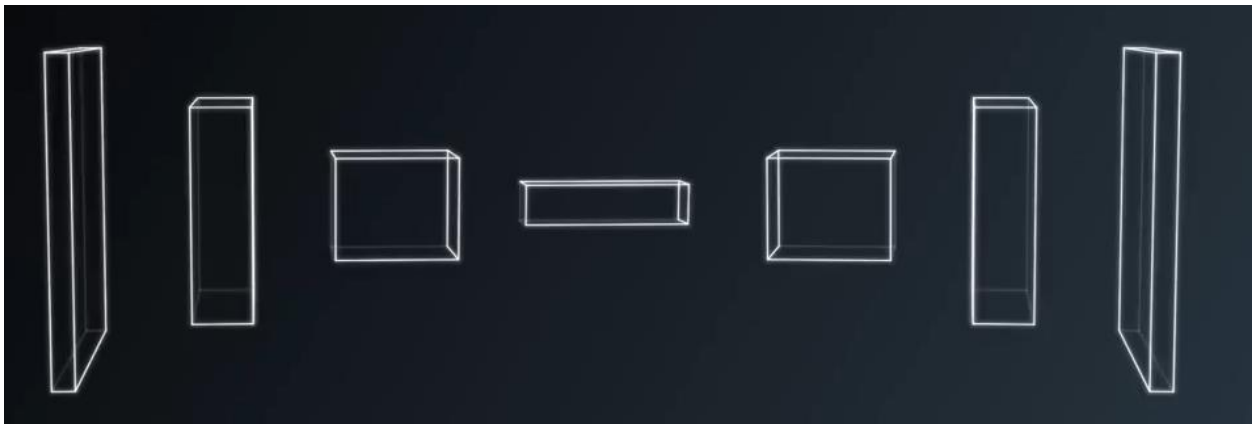


You want to answer the questions? Where in this picture is the hotdog?

FULLY CONNECTED NETWORK

The question is much more difficult to answer since fully connected layers don`t preserve spatial information.

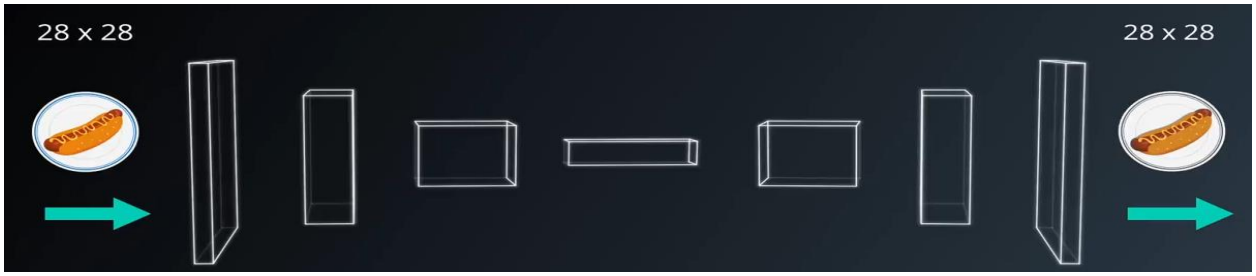
But turn out, If you change the C from connected convolutional, we can integrate convolutions directly into the layer to create fully convolutional network or FCN`s in short.



FCN`s help us answer where is the hotdog question.

Because while doing the convolution they preserve the spatial information throughout the entire network.

Additionally, since convolutional operations Fundamentally don`t care about the size of the input.

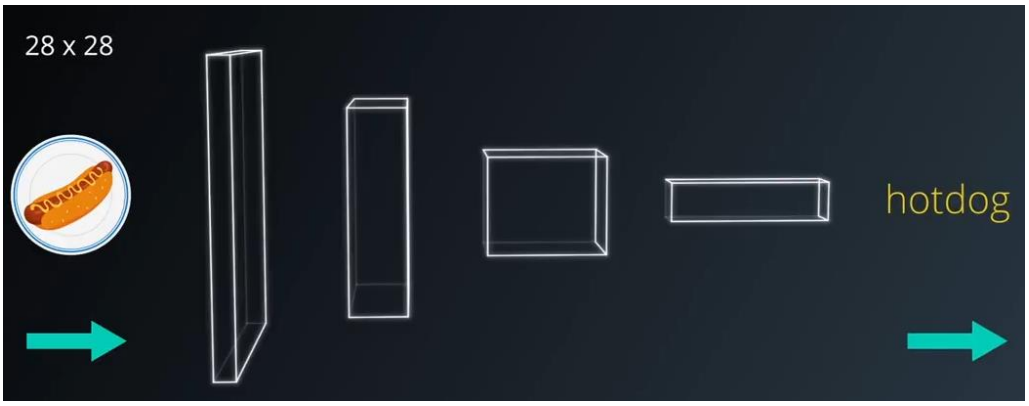


A fully convolutional network will work on images of any size.

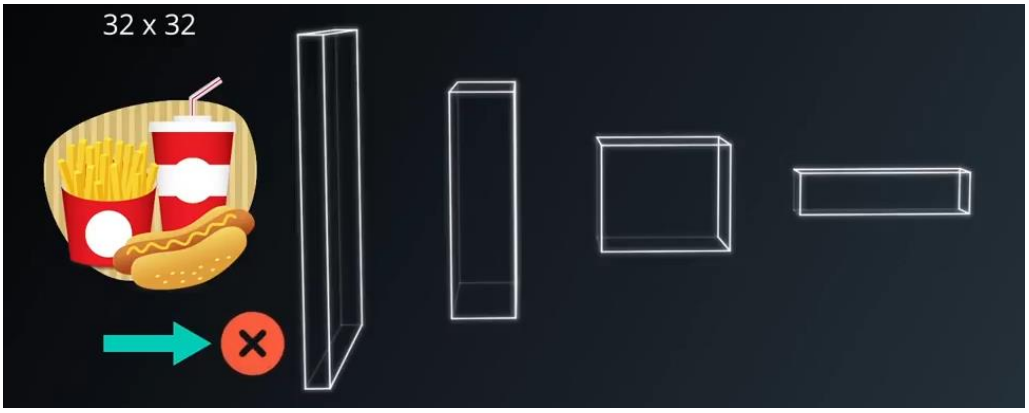
FULLY CONNECTED NETWORK

In classic convolutional network with fully connected final layers the size of input is constrained by the size if the fully connected layers.

Passing different size images through the same sequence of convolutional layers and flatting the final outputs.



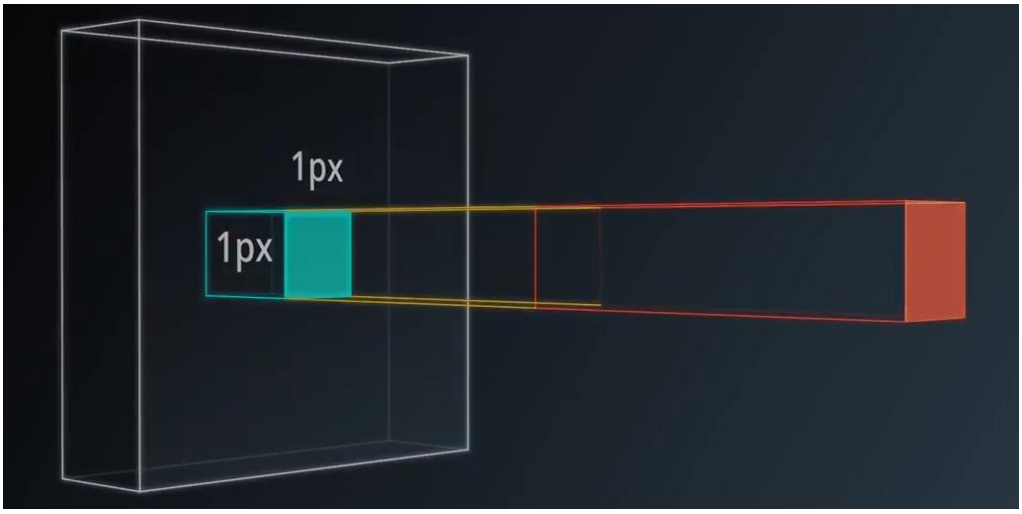
The outputs would be of different sizes, which doesn`t put very well for matrix multiplications.



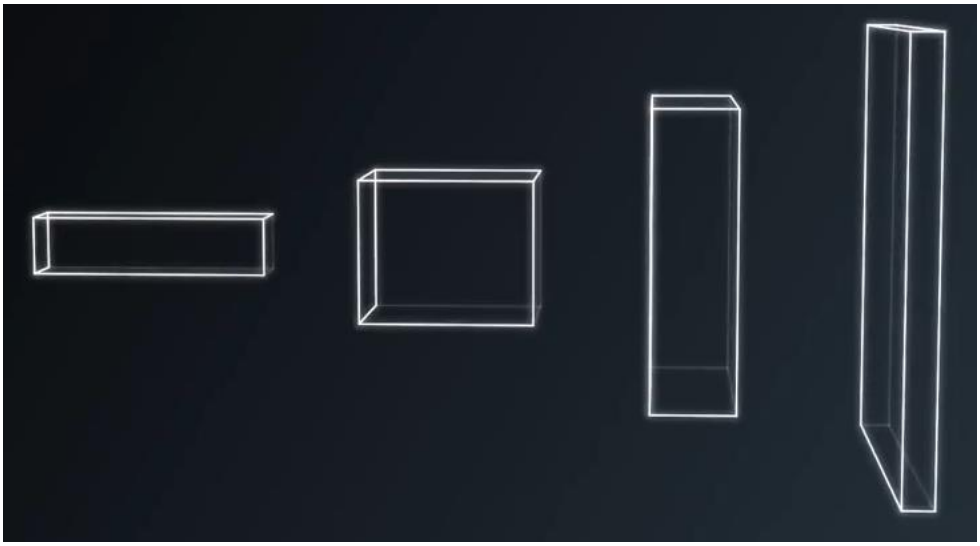
The fully convolutional neural network achieve state of art in computer vision tasks, such as semantic segmentations.

FCNs take advantage of three special techniques

1- **Replace fully connected layer with one convolutional layers,**

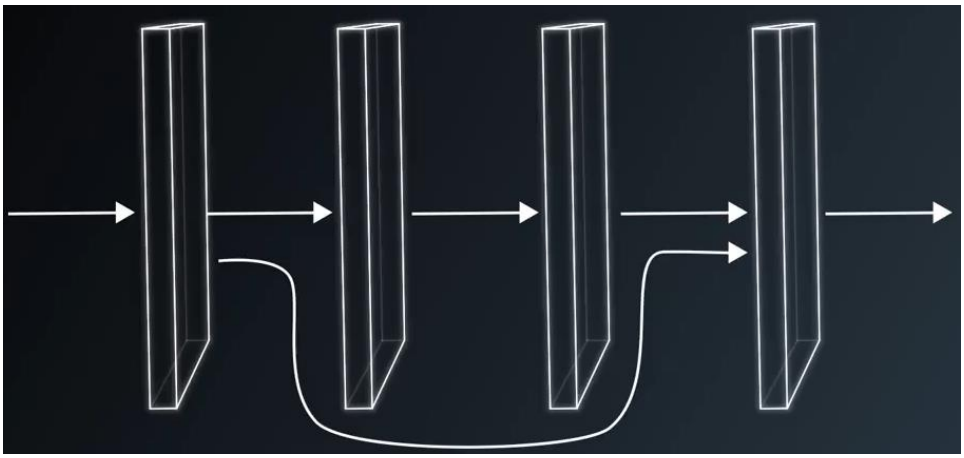


2- two up sampling through the use of transposed convolutional layers



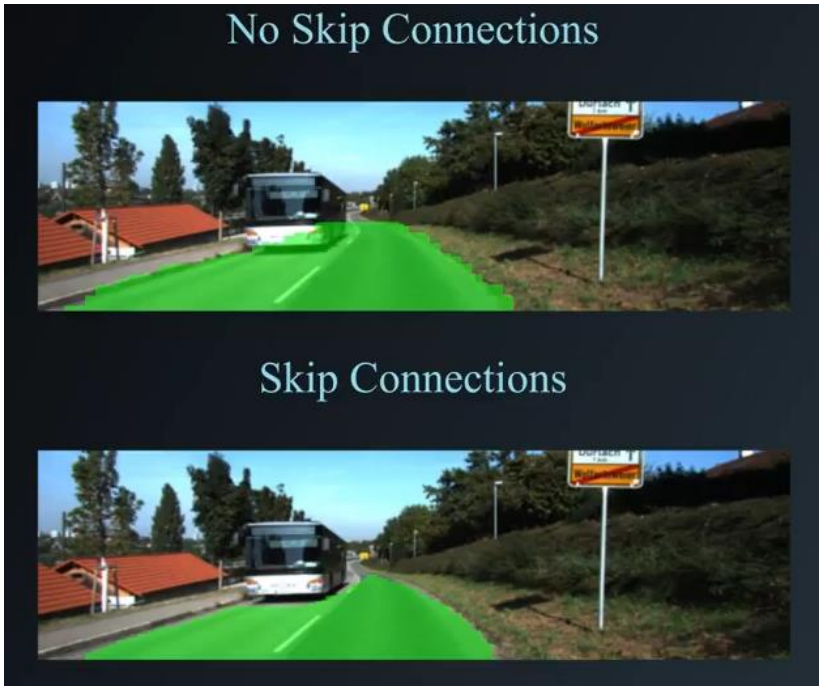
3- three skip connections

These skip connections allow the network to use information from multiple resolutions scales,



As a result, the network is able to make precise segmentation decisions.

We will discuss these techniques details shortly.



Structurally an FCN usually compromised of two parts

1- **encoder**
2- **decoder**

the encoder is a series of convolutional layers like VGG and ResNet.

The goal of the encoder is to extract features from the image.

The decoder up-scales the output of the encoder such that it`s same size the original image.

Thus, it results in segmentation or prediction of each individual pixel in the original image.

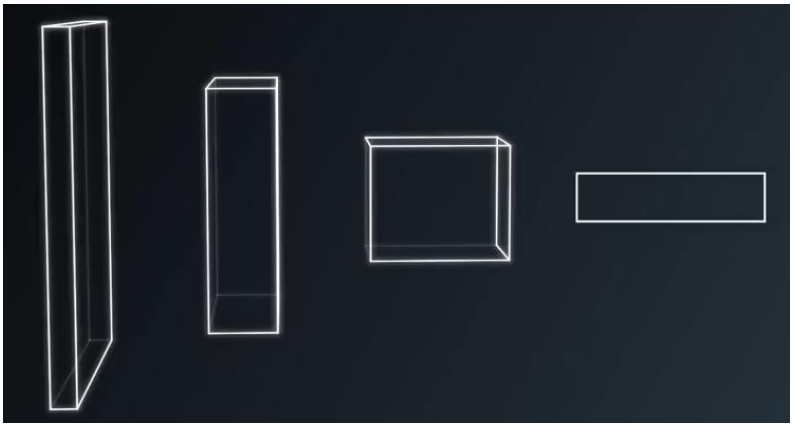
Fully connected 1x1 convolution

This will result in the output value with the tensor will remain 4d instead of flatting to 2D, so spatial information will be preserved.

This may sound daunting but it`s actually simpler than you might think.

At the core, we `re still doing the same math as before.

FULLY CONNECTED NETWORK

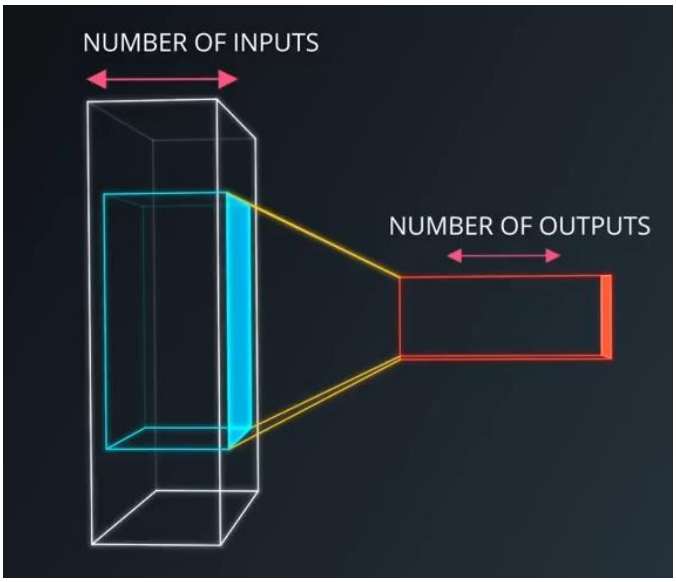


Recall, the output of the convolution operation is result of sweeping the kernel over the input with the sliding window and performing element wise multiplication and summation.

One way to think about this is number of kernels is equivalent to the number of the outputs in a fully connected layer.

Similarly, the number of weights in each kernel Is equivalent to the number of inputs in the fully connected layer.

Effectively, this turns convolutions into matrix multiplications with spatial information.



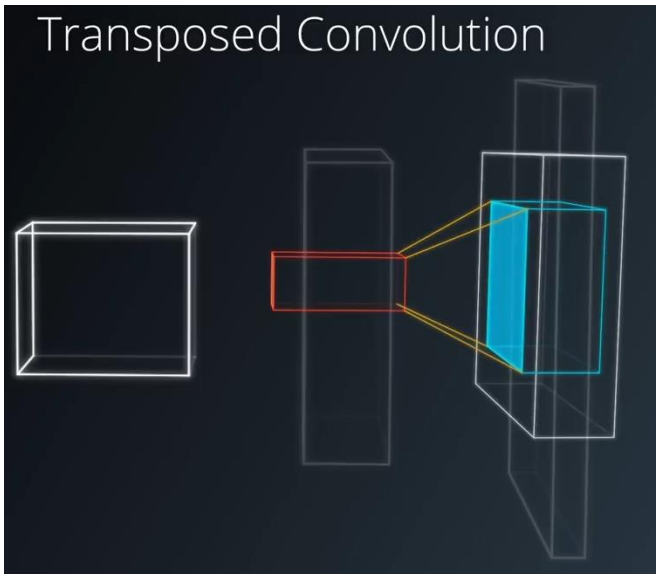
Now we using the second technique we can create decoder of FCN`s using transposed convolution.

A transposed convolution is essentially a reverse convolution in which the forward and the backward passes are swapped.

Hence, we call it transpose convolution

Some people may call it deconvolution because it undoes the previous convolution.

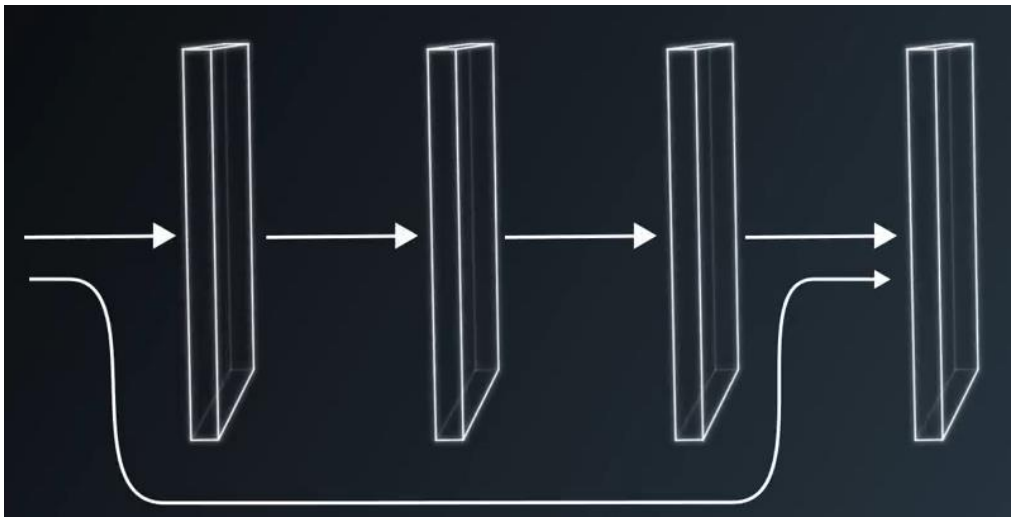
Since all we are doing is swapping the order of forward and backward passes the math actually exactly the same as what we done earlier



The property of differentially is thus retain and training is simply the same as pervious neural network.

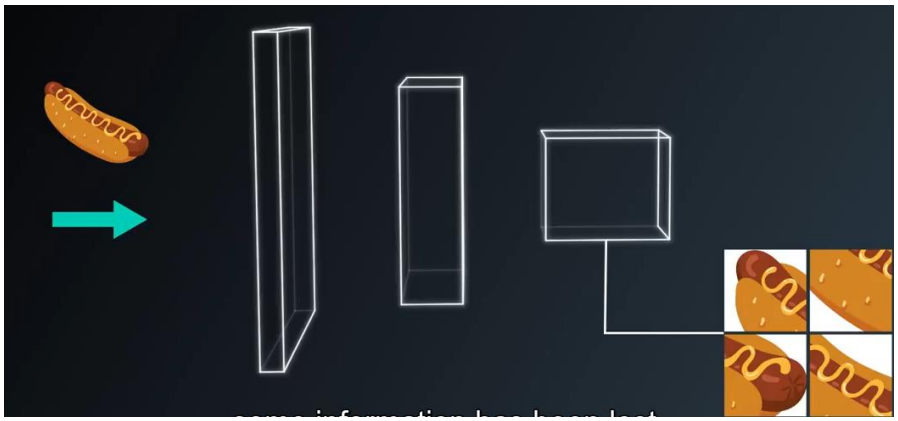
Skip connection

The special technique is fully convolution on network use is the skip connection.

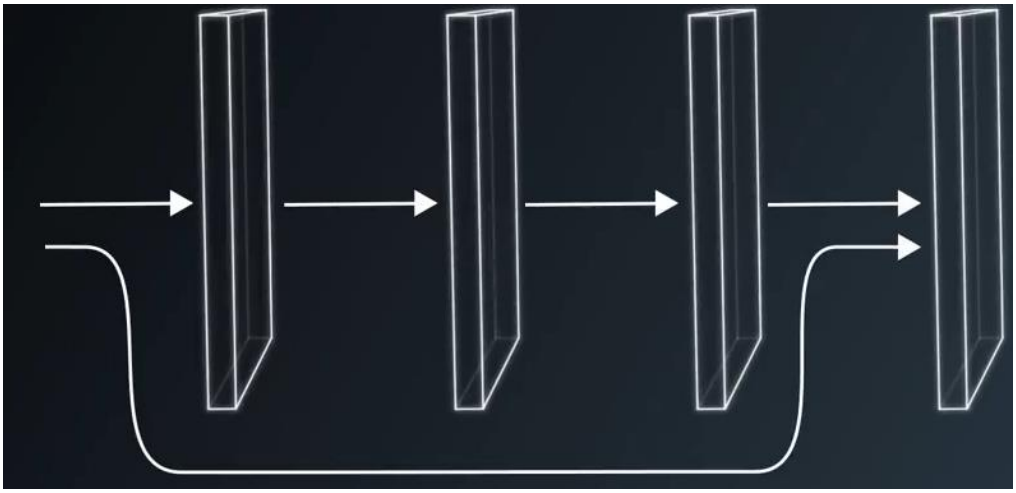


One effect of convolutions or encoding in general is you narrow down the scope by looking closely at some picture and lose the bigger picture as a result.

So even if we were to decode the output the output of the encoder back to the original image size, some information has been lost.

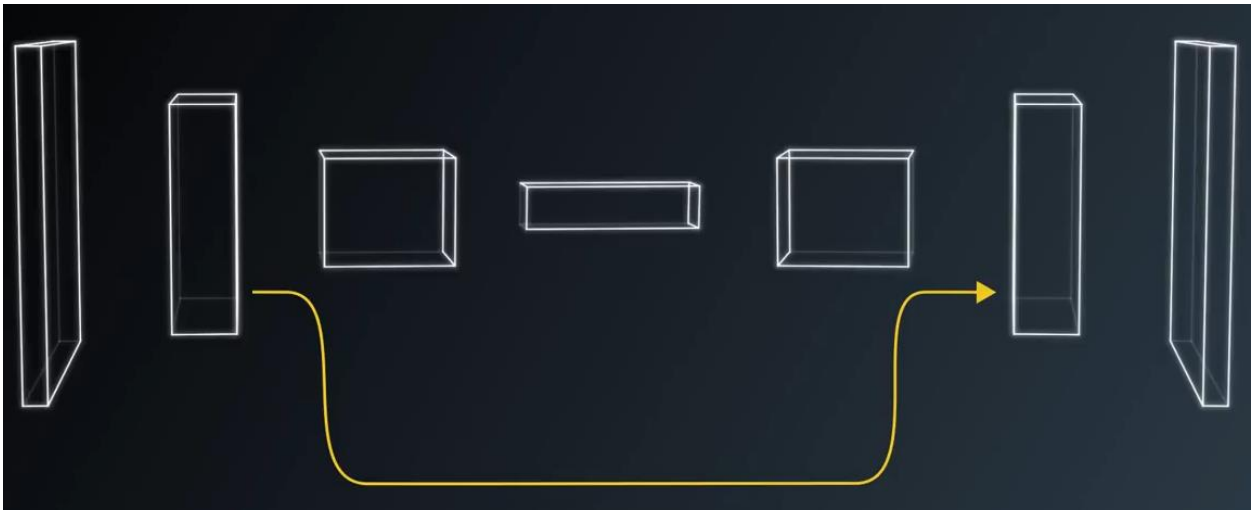


Skip connections are a way of retaining the information easily.

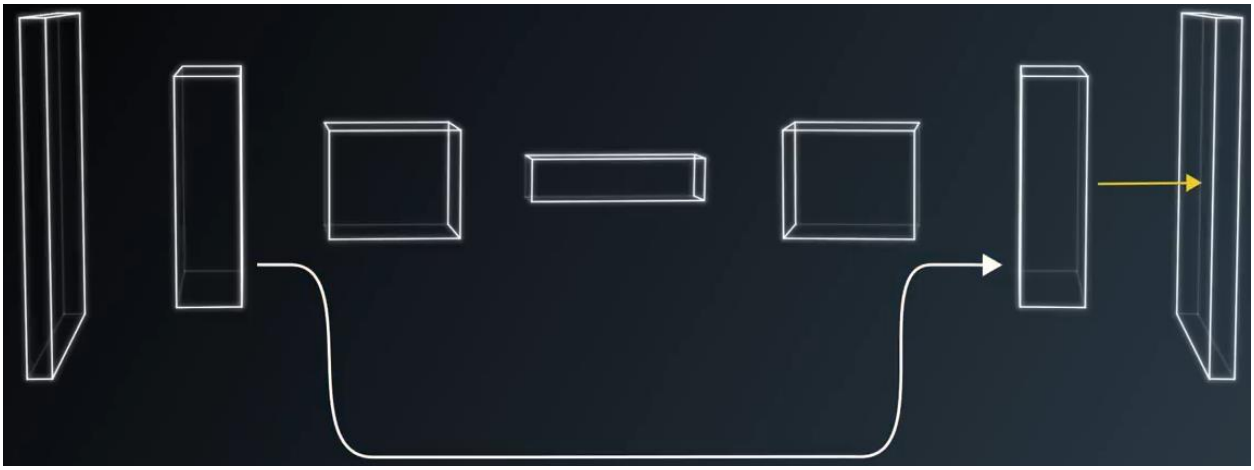


The way skip connections work is by connecting the output of one layer to a non-adjacent layer.

Here, the output of the pooling layer from the encoders combine with the current layers output using element-wise addition operation.



The result is bent into the next layer.

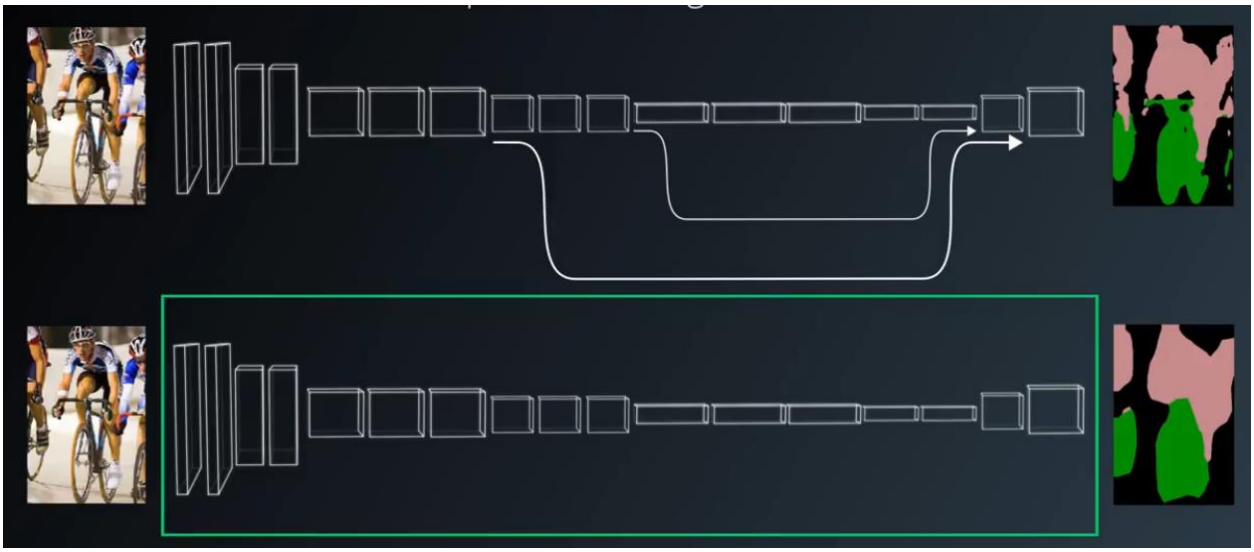


These skip connections allow the network to use information from multiple resolutions.

As result, the network is able to make more precise segmentation decisions.

This is empirically shown in the following comparison between the FCN-8 ARCHTIRECUTER

Which has two skip connections and the FCN-32 ARCHTIRECUTER Which has zero skip connection.



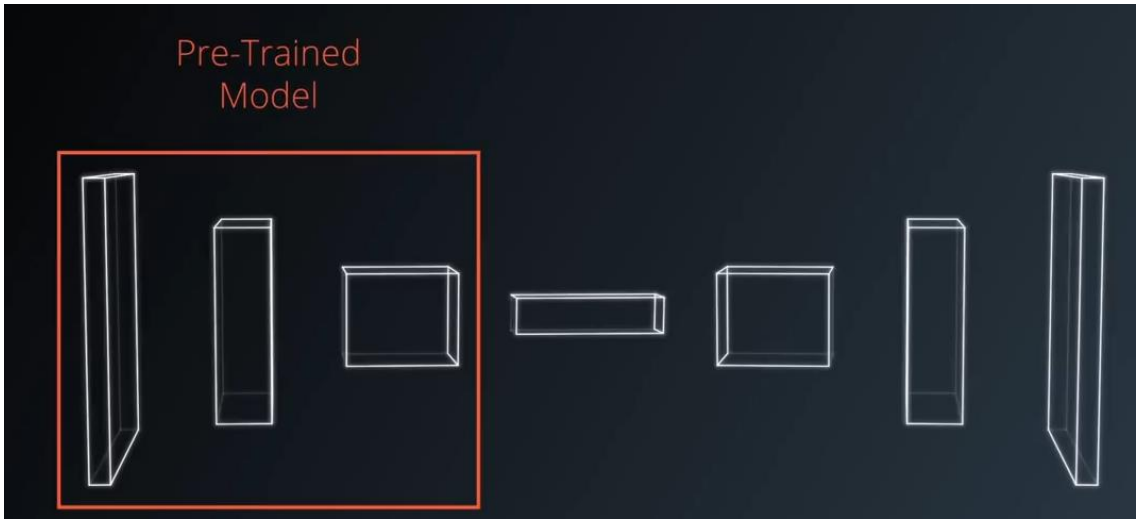
An FCN that have two components, the encoder and the decoder.

1- **We mentioned the encoder extracts features that will later be used the decoder.**

This is sound familiar to transfer learning.

In fact, we can borrow technique from transfer learning to accelerate the training of our FCNs.

It`s common for the encoder to be pre-trained on ImageNet.

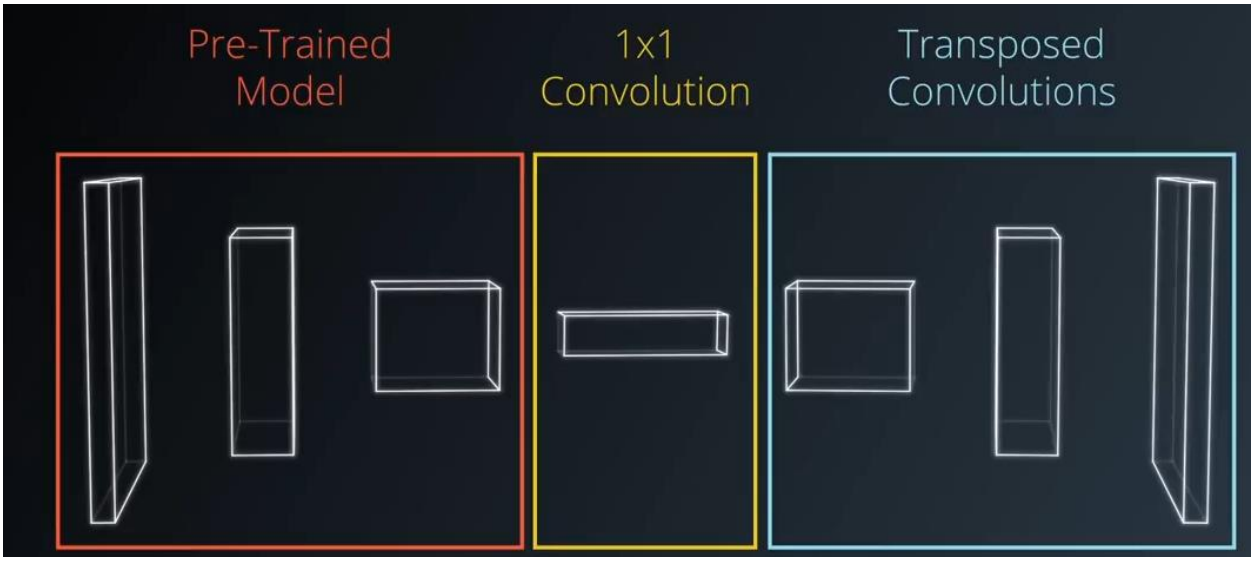


VGG and ResNet are popular choices, as example

By applying the first special technique of one by one convolutional layer conversion, we can complete the encoder portion of the FCN.

The encoder is followed by the decoder, which uses a second special technique of transposed convolutional layer to up sample the image.

Then the skip connection via the third special technique is added.



Be careful not to add too many skip connections, through It can lead to the explosion in the size of your model.

For example, when using VGG-16 as the encoder, only the third and the fourth pooling layers are typically used for skip connections.

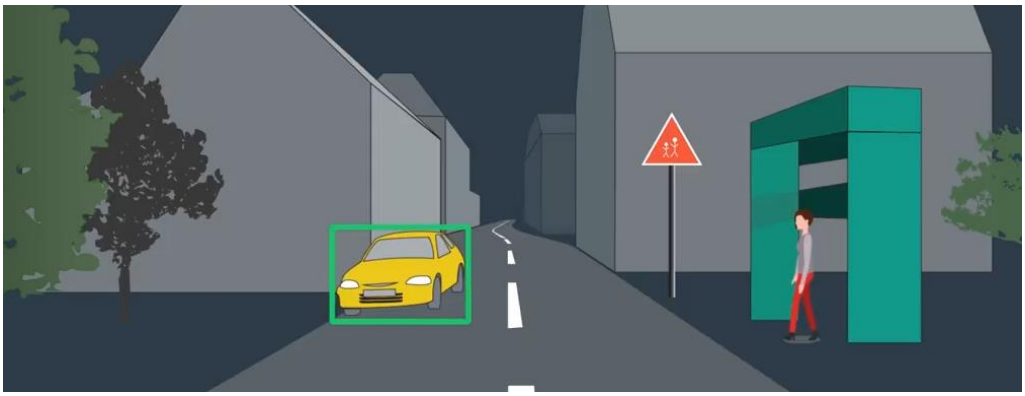
After all that, we can now train the model end to end.

Boundary box

First object detection and for that, we can use bounding box, there are a similar method of scene understating compared to segmentation.

In neural Network, just has to figure out

1- **Where an object is and draw a type box around it.**



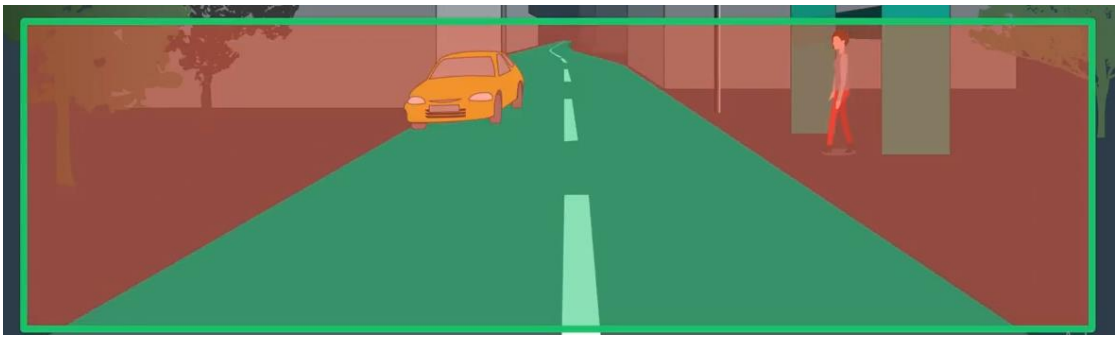
There is already great open source state of the art solutions, such as YOLO and SSD models.

These models perform extremally well even at high frame per second.

There are useful for detecting different objects such as cars, people, traffic light and other objects in the scene.

However, burning boxes have their limits.

Imagine drawing and bounding box around a curvy road, the forest, or the sky this quickly becomes problematic or even impossible to convey the true shape of an object. At best, the bounding box can only hope to achieve partial seen understanding.



FULLY CONNECTED NETWORK

Semantic Segmentation is a task of assigning meaning to part of object.

There can be done at the pixel level where we assign.

Each pixel to a target class such as road, car, pedestrian, sign, or any number of other classes.

Semantic segmentation helps us derive valuable information about in every pixel in the image rather than just slicing sections into boundary boxes.

This is field known as scene understanding and it`s particularly relevant to autonomous vehicle.

Full scene understanding help with perception, which enables vehicles to make decisions.

Semantic Segmentation

In semantic segmentation, we want to input an image into a neural network and we want to output a category for **every pixel** in this image. For example, for the below image of a couple of cows, we want to look at every pixel and decide: is that pixel part of a cow, the grass or sky, or some other category?

Small image of cows in a field.

We'll have a discrete set of categories, much like in a classification task. But instead of assigning a single class to an image, we want to assign a class to every pixel in that image. So, how do we approach this task?

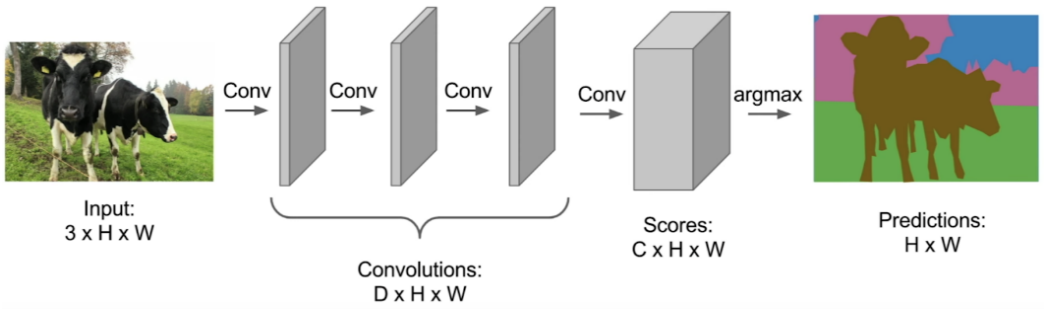
Fully-Convolutional Network (FCN) Approach

If your goal is to preserve the spatial information in our original input image, you might think about the simplest solution: simply never discard any of that information; never downsample/maxpool and don't add a fully-connected layer at the end of the network

We could use a network made entirely of convolutional layers to do this; something called a fully convolutional neural network. **A fully convolutional neural network preserves spatial information.**

This network would take in an image that **has true labels** attached to each pixel, so every pixel is labeled as grass or cat or sky, and so on. Then we pass that input through a stack of convolutional layers that preserve the spatial size of the input (something like 3x3 kernels with zero padding). Then, the final convolutional layer outputs a tensor that has dimensions CxHxW, where C is the number of categories we have.

FULLY CONNECTED NETWORK



FCN architecture.

Predictions

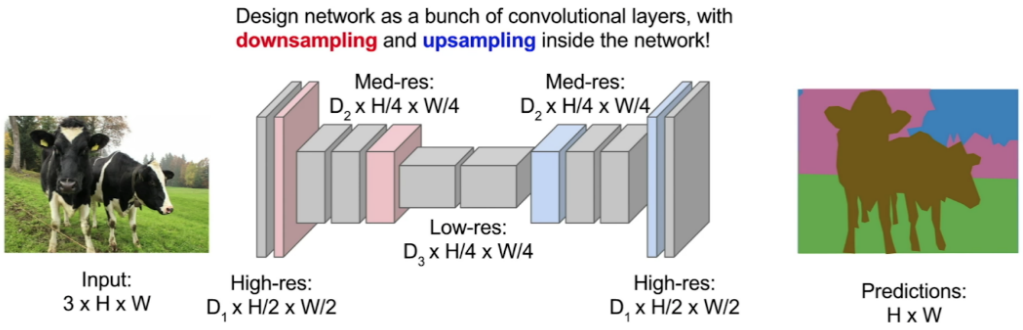
This output Tensor of predictions contains values that classify every pixel in the image, so if we look at a single pixel in this output, we would see a vector that looks like a classification vector -- with values in it that show the probability of this single pixel being a cat or grass or sky, and so on. **We could do this pixel level classification** all at once, and then train this network by assigning a loss function to each pixel in the image and **doing backpropagation as usual**. So, if the network makes an error and classifies a single pixel incorrectly, it will go back and adjust the weights in the convolutional layers until that error is reduced.

Limitations of This Approach

- t's very expensive to label this data (you have to label every pixel), and
- It's computationally expensive to maintain spatial information in each convolutional layer

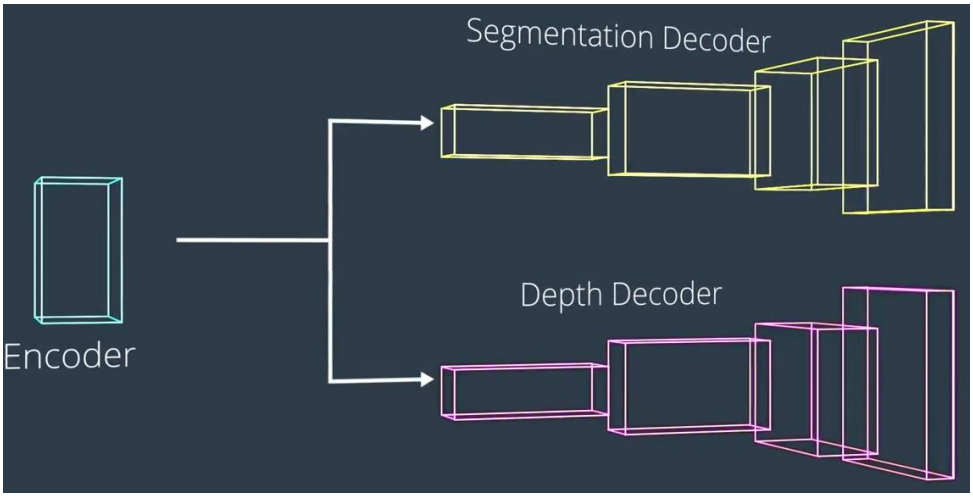
Downsampling/Upsampling

Instead, what you usually see is an architecture that uses downsampling of a feature map and an upsampling layer to reduce the dimensionality and, therefore, the computational cost, of moving forward through these layers in the middle of the network.



So, what you'll see in these networks is a couple of convolutional layers followed by downsampling done by something like maxpooling, very similar to a simple image classification network. Only, this time, in the second half of the network, we want to increase the spatial resolution, so that our output is the same size as the input image, with a label for every pixel in the original image.

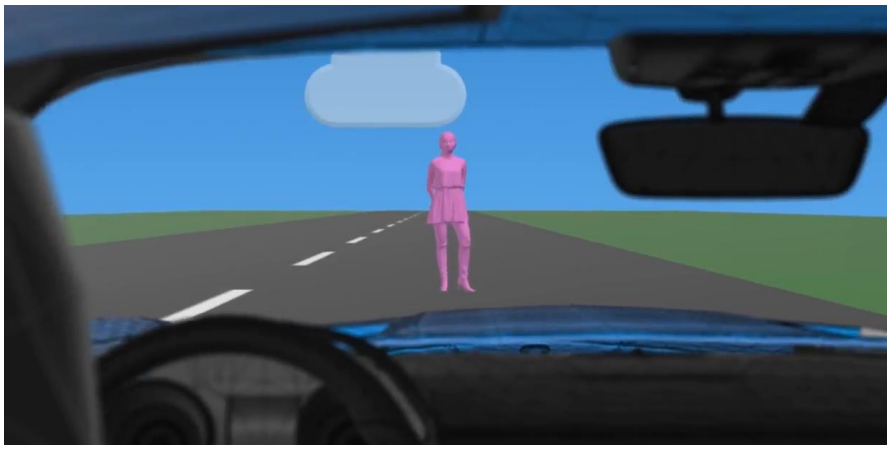
One approach to seen understanding to train multiple decoder.



Each decoder trains on a separate task. We might have one decoder for

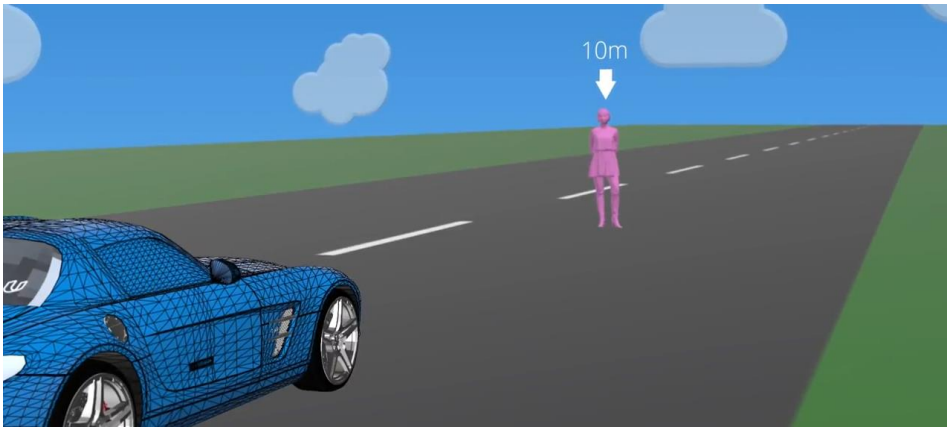
1- **segmentation**
2- **Another for depth measurement**

This way we can have a **single network** which not only predicts the class of a pixel but additionally how far away it is.



You can imagine using this information to reconstruct a rich 3D scene, kind of like how we human do.

This is in fact, the next step to **visual perception**. For now, though we will keep things relatively simple focused just on **semantic segmentation**.

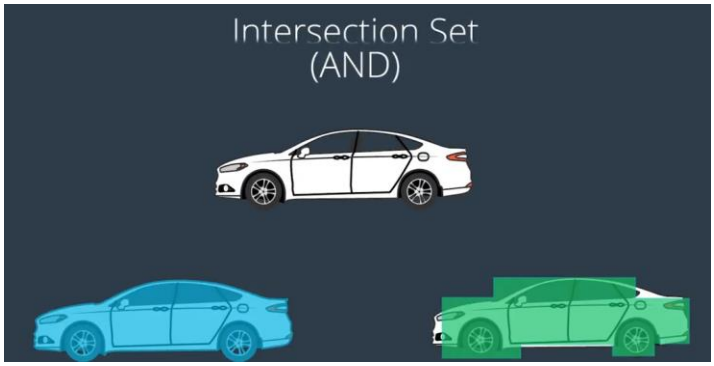


Intersection Union Unit , in short IOU Metric commonly use to measure the performance of a model on the semantic segmentation task.

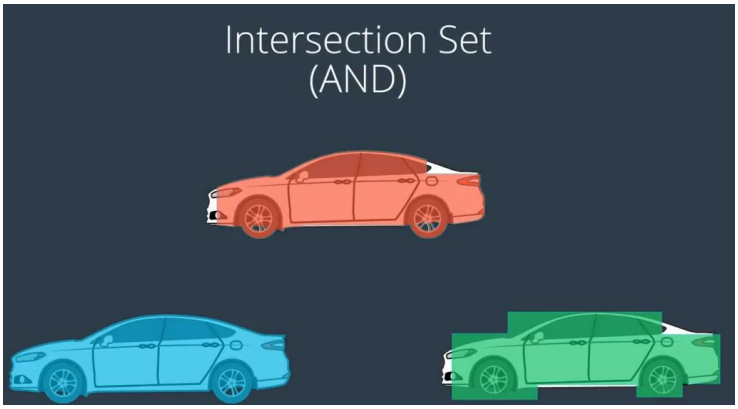
It is literally just the intersection set divided by the union set.

$$\frac{\text{Intersection Set}}{\text{Union Set}}$$

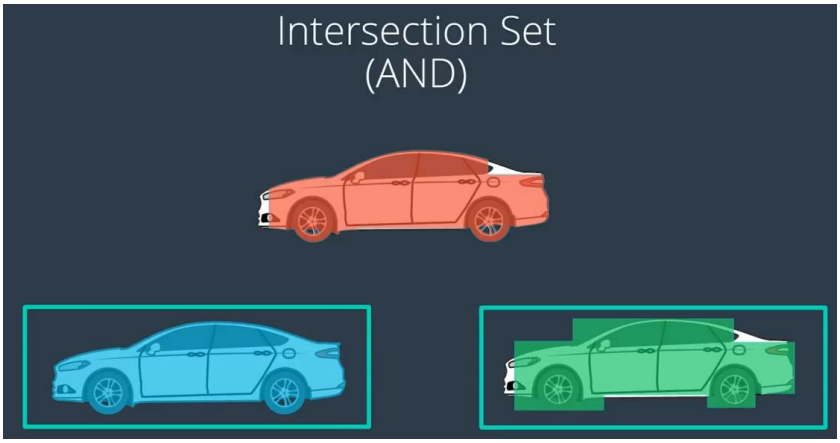
Intersection of two sets is an AND operation.



If it exists in both sets, then we put it into the intersection set.

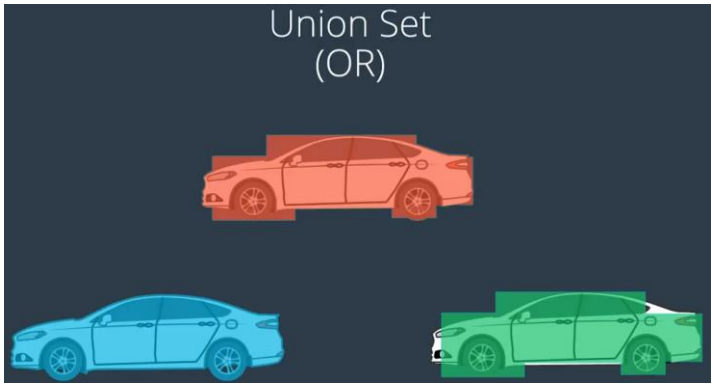


For each class, the intersection is defined as the number of pixels that are both truly part of that class and are classified as part of that class by the network.

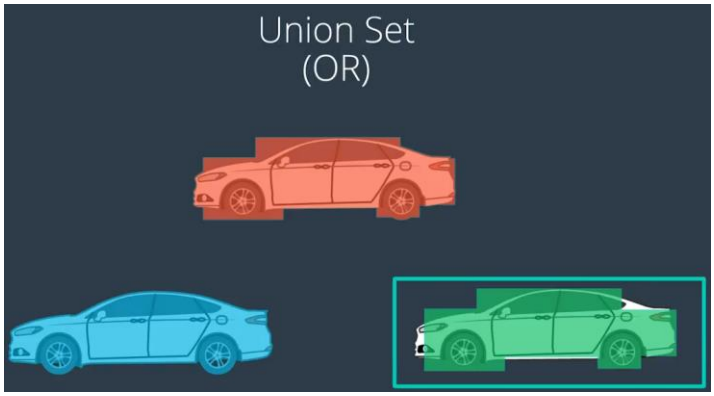


Union of two set is a OR operation

If it exists in at least one of the two sets, then we put it into the union set.



The union is defined as the number of pixels that are truly part of that class plus the number of pixels that are classified as part of that class by the network.



So, the intersection set should always be smaller or equal to the union set.

The ratio then tells us the overall performance per pixel, per class.

Since this intersection set is divided by the union set the ratio will always be less than or equal to one.

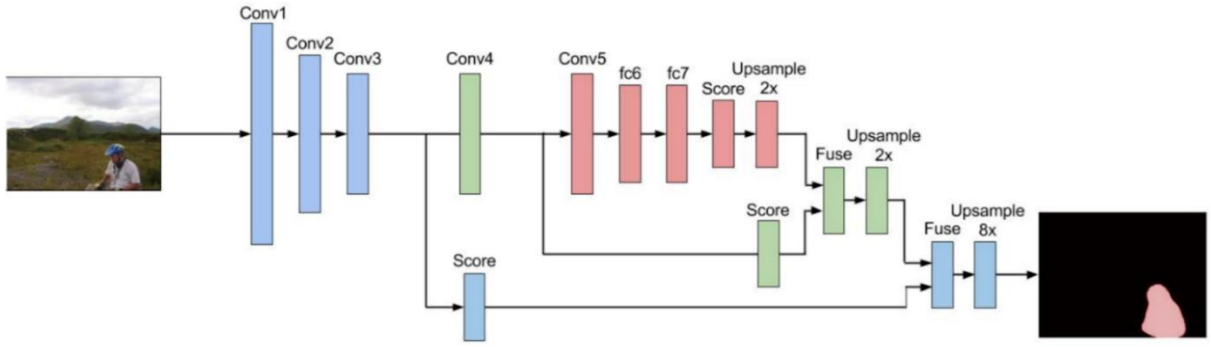
We can go even further and calculate the mean IOU for a network, Which is just the average of all the IOU for all the classes.

This give you an idea of how well it handles all different classifications for every single pixel.

TensorFlow we can use matrix mean IOU function and call it directly.

FCN-8, Architecture

Let's focus on a concrete implementation of a fully convolutional network. We'll discuss the FCN-8 architecture developed at Berkeley. In fact, many FCN models are derived from this FCN-8 implementation. The encoder for FCN-8 is the VGG16 model pretrained on **ImageNet for classification**. The fully-connected "score" layers are replaced by 1-by-1 convolutions. The code for convolutional score layer like looks like:




```
self.score = nn.Conv2d(input_size, num_classes, 1)
```

There are skip connections and various upsampling layers to keep track of a variety of spatial information. In practice this network is often broken down into the **encoder** portion with parameters from VGG net, and a decoder portion, which includes the upsampling layers.

FCN-8, Decoder Portion

To build the decoder portion of FCN-8, we'll upsample the input, that comes out of the convolutional layers taken from VGG net, to the original image size. The shape of the tensor after the final convolutional transpose layer will be 4-dimensional: (batch_size, original_height, original_width, num_classes).

To define a transposed convolutional layer for upsampling, we can write the following code, where 3 is the size of the convolving kernel:

```
self.transpose = nn.ConvTranspose2d(input_depth, num_classes, 3, stride=2)
```

The transpose convolutional layers increase the height and width dimensions of the 4D input Tensor. And you can look at the [PyTorch documentation, here].

Skip Connections

The final step is adding skip connections to the model. In order to do this we'll combine the output of two layers. The first output is the output of the current layer. The second output is the output of a layer further back in the network, typically a pooling layer.

In the following example we combine the result of the previous layer with the result of the 4th pooling layer through elementwise addition (`tf.add`).

```
# the shapes of these two layers must be the same to add them
input = input + pool_4
```

We can then follow this with a call to our transpose layer.

```
input = self.transpose(input)
```

We can repeat this process, upsampling and creating the necssary skip connections, until the network is complete!

FCN-8, Classification & Loss

The final step is to define a loss. That way, we can approach training a FCN just like we would approach training a normal classification CNN.

In the case of a FCN, the goal is to assign each pixel to the appropriate class. We already happen to know a great loss function for this setup, **cross entropy loss!**

Remember the output tensor is **4D** so before we perform the loss, we have to reshape it **to 2D**, **where each row represents a pixel and each column a class**. From here we can just use **standard cross entropy loss**.

That's it, we now have an idea of how to create an end-to-end model for semantic segmentation. Check out the original paper to read more specifics about FCN-8.

Semantic segmentation is the cutting edge of perception in autonomous Vehicles and beyond.

This is a tremendous skill to demonstrate to employers and to use on the job.

Even better, you can on this knowledge to construct faster and more powerful segmentation network.

FULLY CONNECTED NETWORK

You will learn about inference optimizations, which accelerate the deployed model in production.

You see, even after training there`s still a lot we can do to improve performance.