Silhoutte Generation using Semantic Segmentation

-By Ashish Gupta

An overview of semantic image segmentation.

- Image segmentation is a computer vision task in which we label specific regions of an image according to what's being shown.
- *"What's in this image, and where in the image is it located?"*
- More specifically, the goal of semantic image segmentation is to label *each pixel* of an image with a corresponding ***class*** of what is being represented. Because we're predicting for every pixel in the image, this task is commonly referred to as **dense prediction**

predict

Person
Bicycle
Background

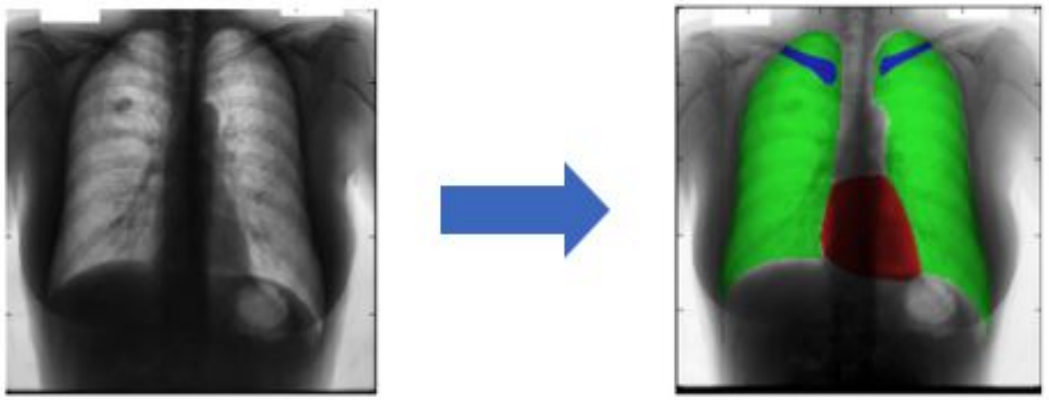
One important thing to note is that we're not separating *instances* of the same class; we only care about the category of each pixel. In other words, if you have two objects of the same category in your input image, the segmentation map does not inherently distinguish these as separate objects. There exists a different class of models, known as *instance segmentation* models, which *do* distinguish between separate objects of the same class. Segmentation models are useful for a variety of tasks, including:

Autonomous vehicles

We need to equip cars with the necessary perception to understand their environment so that self-driving cars can safely integrate into our existing roads.

Medical image diagnostics

Machines can augment analysis performed by radiologists, greatly reducing the time required to run diagnositic tests.

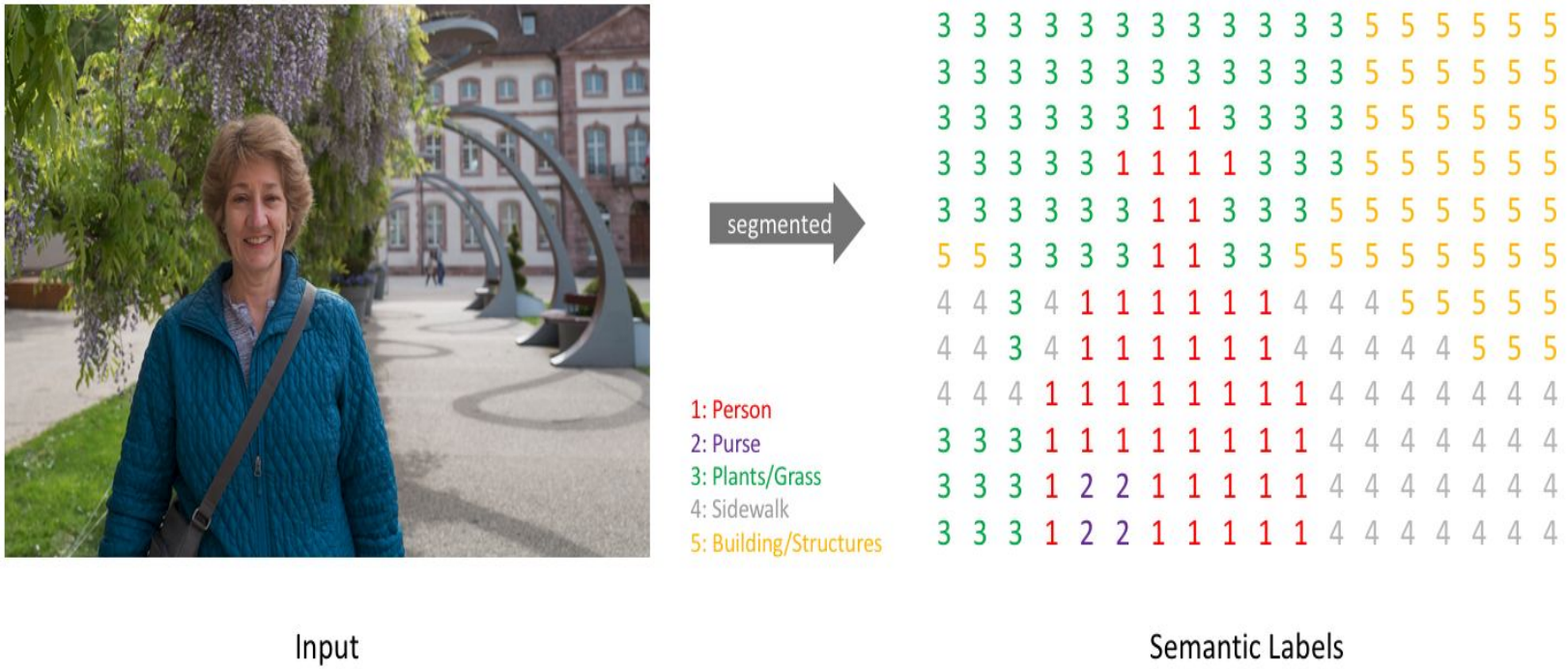


Input Image

Segmented Image

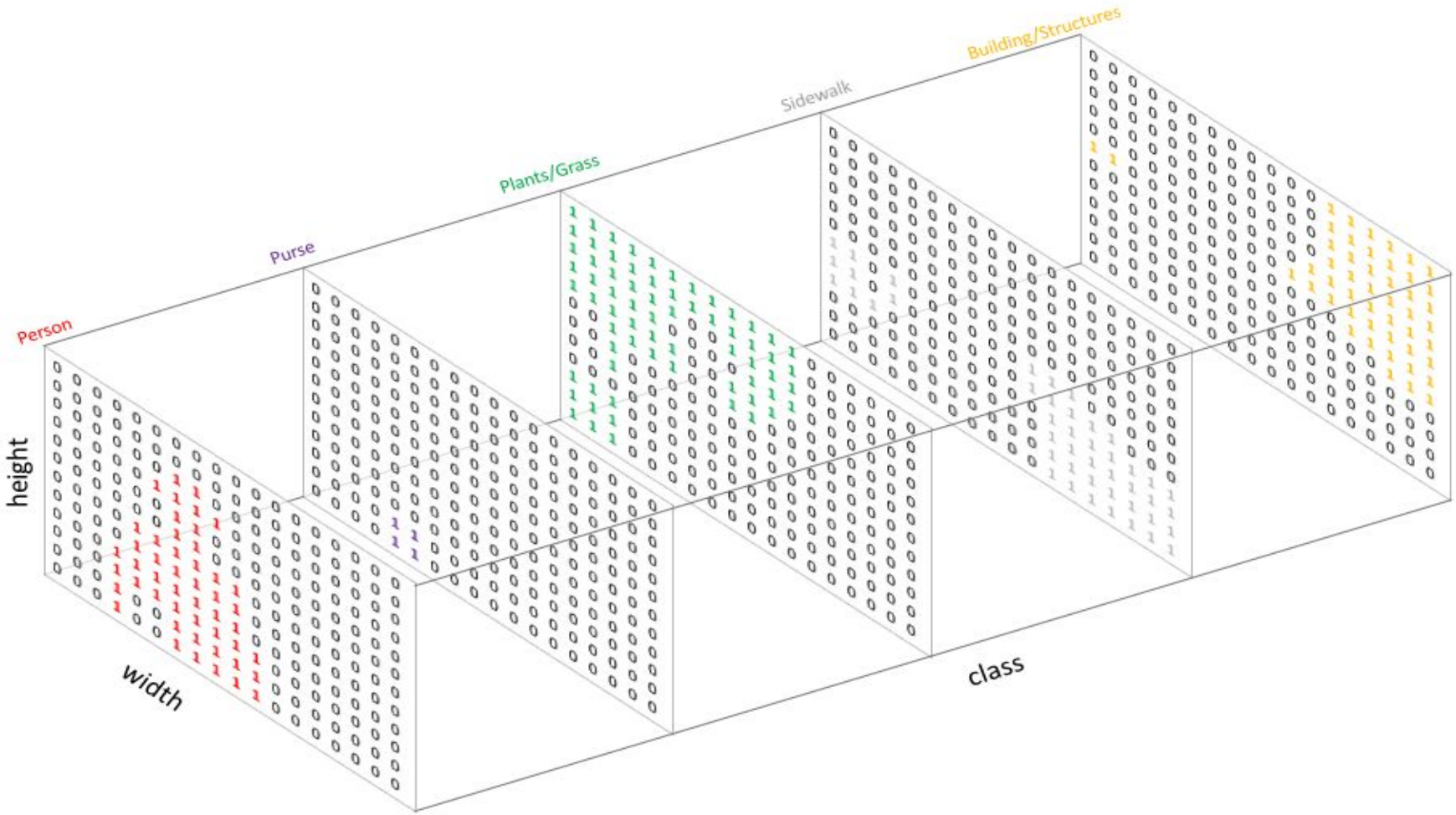
Representing the task

Simply, our goal is to take either a RGB color image (height×width×3height×width×3) or a grayscale image (height×width×1height×width×1) and output a segmentation map where each pixel contains a class label represented as an integer (height×width×1height×width×1).



Note: For visual clarity, I've labeled a low-resolution prediction map. In reality, the segmentation label resolution should match the original input's resolution.

Similar to how we treat standard categorical values, we'll create our **target** by one-hot encoding the class labels - essentially creating an output channel for each of the possible classes.

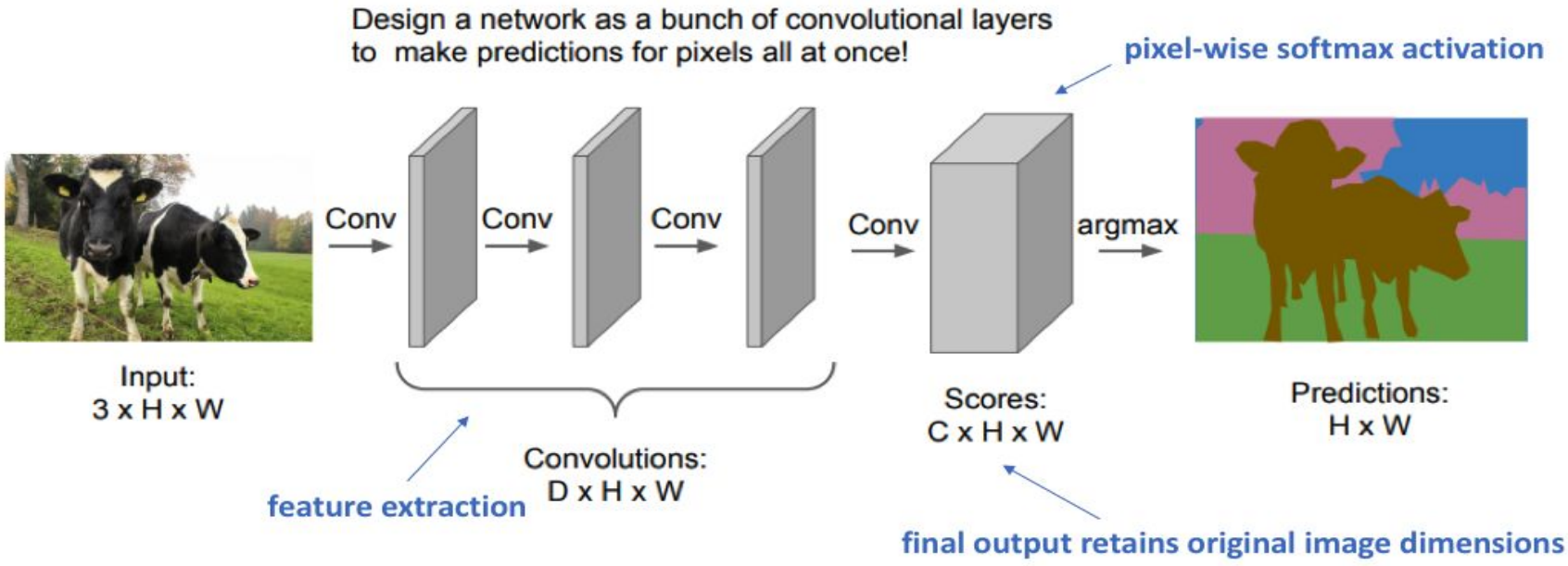


A prediction can be collapsed into a segmentation map (as shown in the first image) by taking the `argmax` of each depth-wise pixel vector.
We can easily inspect a target by overlaying it onto the observation.

When we overlay a *single channel* of our target (or prediction), we refer to this as a **mask** which illuminates the regions of an image where a specific class is present.

Constructing an architecture

A naive approach towards constructing a neural network architecture for this task is to simply stack a number of convolutional layers (with `same` padding to preserve dimensions) and output a final segmentation map. This directly learns a mapping from the input image to its corresponding segmentation through the successive transformation of feature mappings; however, it's quite computationally expensive to preserve the full resolution throughout the network.



Downside: Preserving image dimensions throughout entire network will be computationally expensive.

Recall that for deep convolutional networks, earlier layers tend to learn low-level concepts while later layers develop more high-level (and specialized) feature mappings. **In order to maintain expressiveness, we typically need to increase the number of feature maps (channels) as we get deeper in the network.**

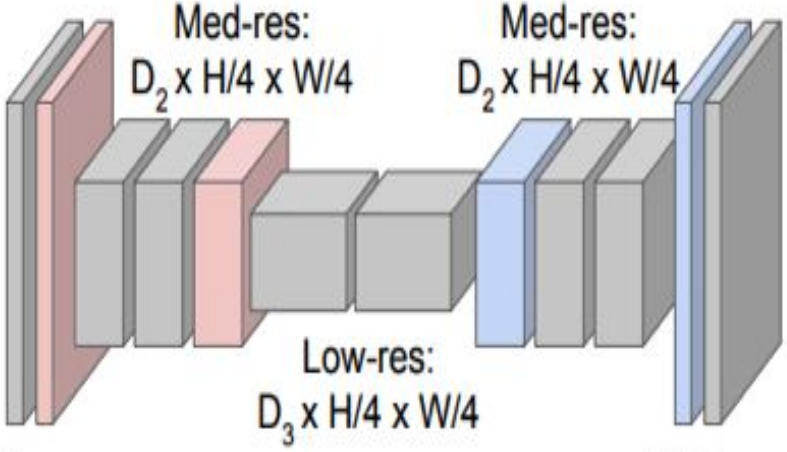
This didn't necessarily pose a problem for the task of image classification, because for that task we only care about *what* the image contains (and not where it is located). Thus, we could alleviate computational burden by periodically downsampling our feature maps through pooling or strided convolutions (ie. compressing the spatial resolution) without concern.

However, for image segmentation, we would like our model to produce a *full-resolution* semantic prediction.

One popular approach for image segmentation models is to follow an **encoder/decoder structure** where we *downsample* the spatial resolution of the input, developing lower-resolution feature mappings which are learned to be highly efficient at discriminating between classes, and the *upsample* the feature representations into a full-resolution segmentation map.

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Input:
3 x H x W



High-res:
D1 x H/2 x W/2

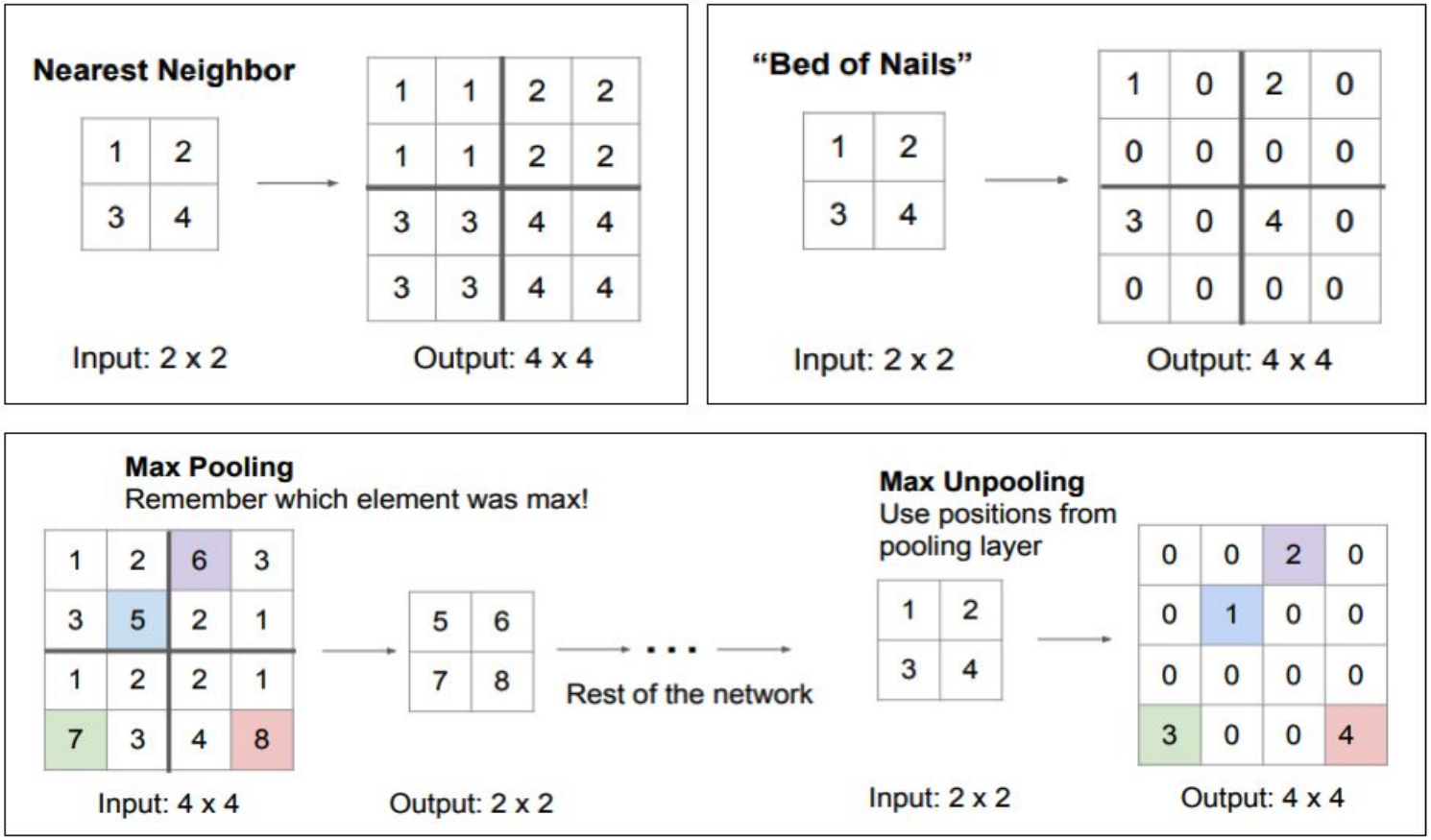
High-res:
D1 x H/2 x W/2

Predictions:
H x W

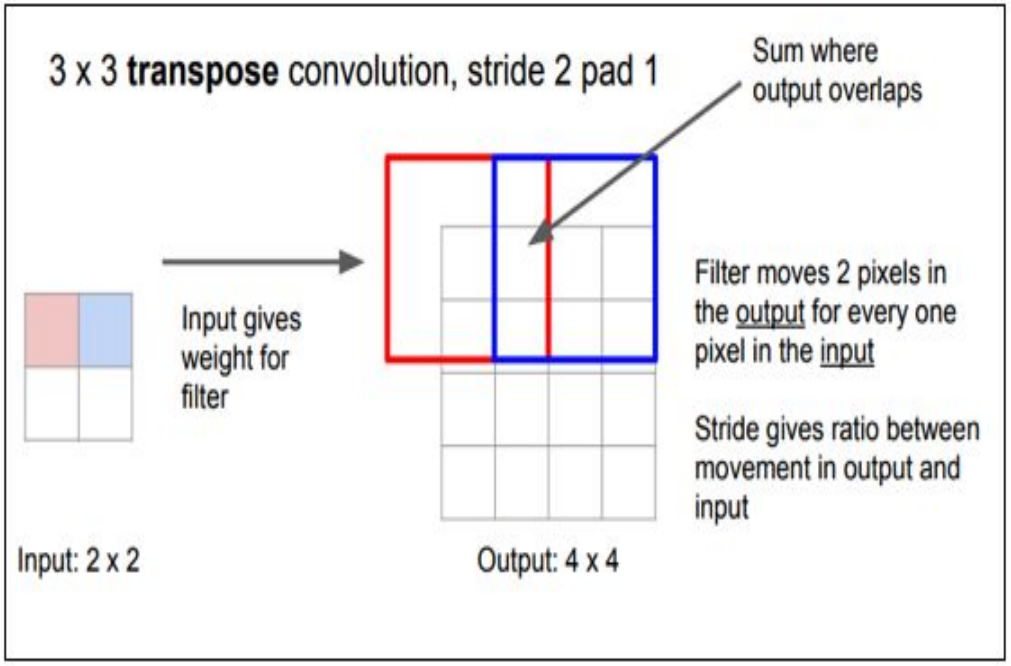
Solution: Make network deep and *work at a lower spatial resolution* for many of the layers.

Methods for upsampling

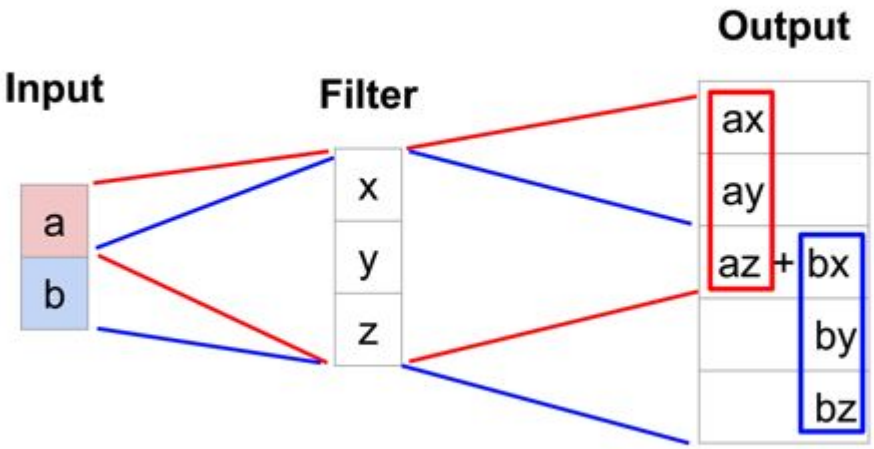
There are a few different approaches that we can use to *upsample* the resolution of a feature map. Whereas pooling operations downsample the resolution by summarizing a local area with a single value (ie. average or max pooling), "unpooling" operations upsample the resolution by distributing a single value into a higher resolution.



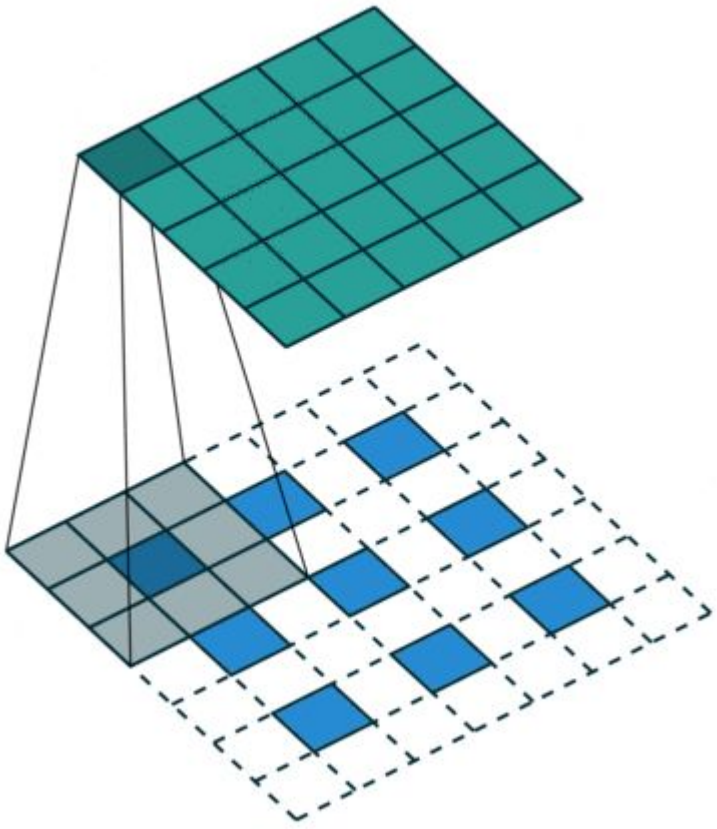
However, **transpose convolutions** are by far the most popular approach as they allow for us to develop a *learned upsampling*.



Whereas a typical convolution operation will take the dot product of the values currently in the filter's view and produce a single value for the corresponding output position, a transpose convolution essentially does the opposite. For a transpose convolution, we take a single value from the low-resolution feature map and multiply all of the weights in our filter by this value, projecting those weighted values into the output feature map.

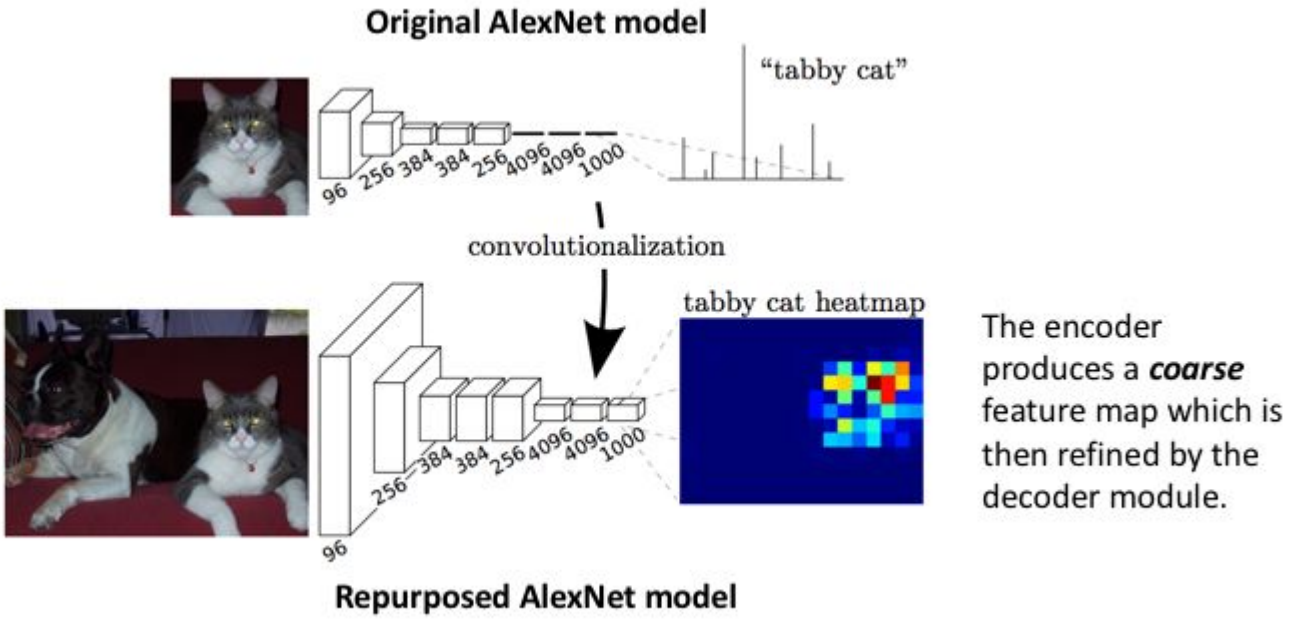


For filter sizes which produce an overlap in the output feature map (eg. 3x3 filter with stride 2 - as shown in the below example), the overlapping values are simply added together. Unfortunately, this tends to produce a checkerboard artifact in the output and is undesirable, so it's best to ensure that your filter size does not produce an overlap.

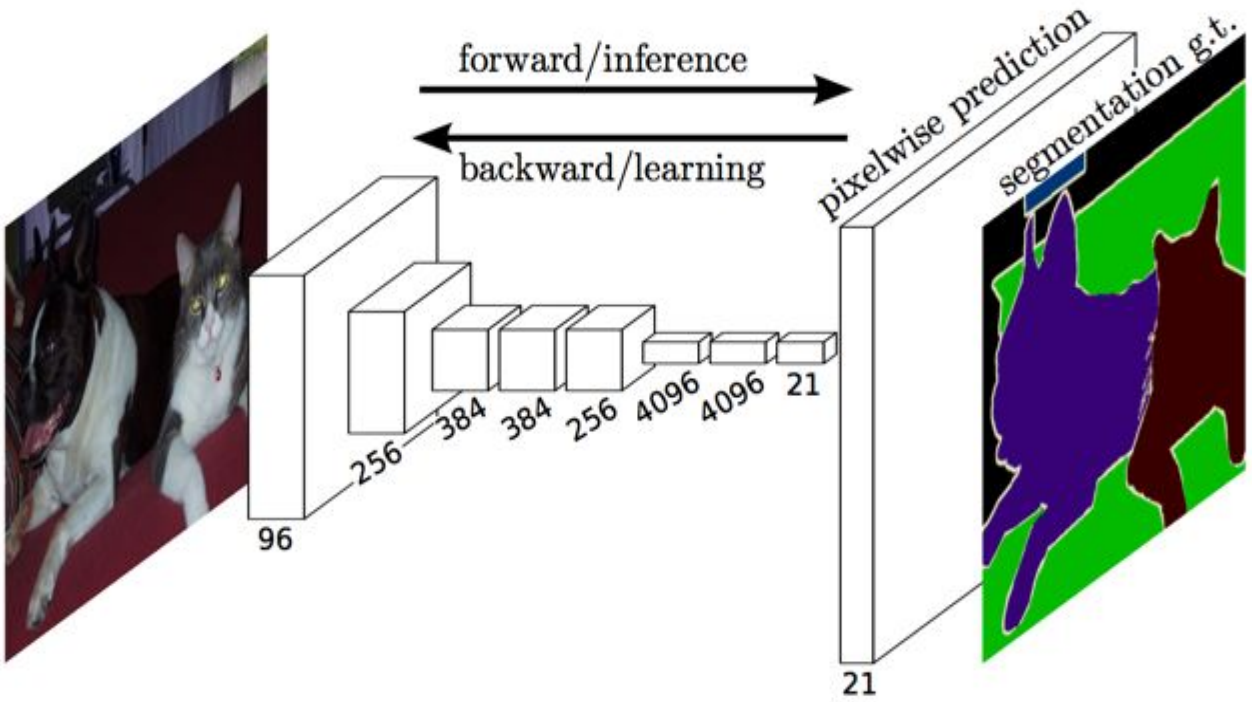


Fully convolutional networks

The approach of using a "fully convolutional" network trained end-to-end, pixels-to-pixels for the task of image segmentation was introduced by Long et al. in late 2014. The paper's authors propose adapting existing, well-studied *image classification* networks (eg. AlexNet) to serve as the encoder module of the network, appending a decoder module with transpose convolutional layers to upsample the coarse feature maps into a full-resolution segmentation map.



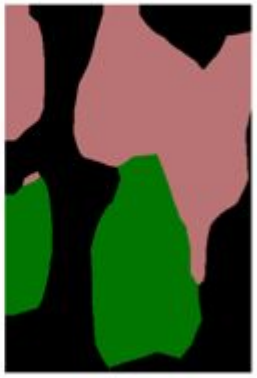
The full network, as shown below, is trained according to a pixel-wise cross entropy loss.



However, because the encoder module reduces the resolution of the input by a factor of 32, the decoder module **struggles to produce fine-grained segmentations** (as shown below).

Ground truth target

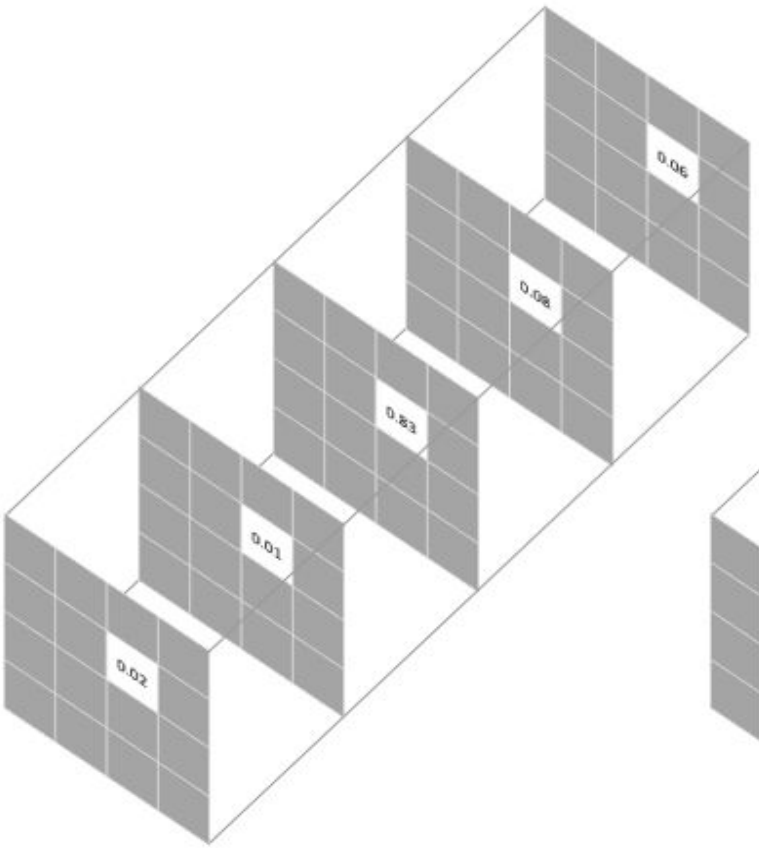
Predicted segmentation



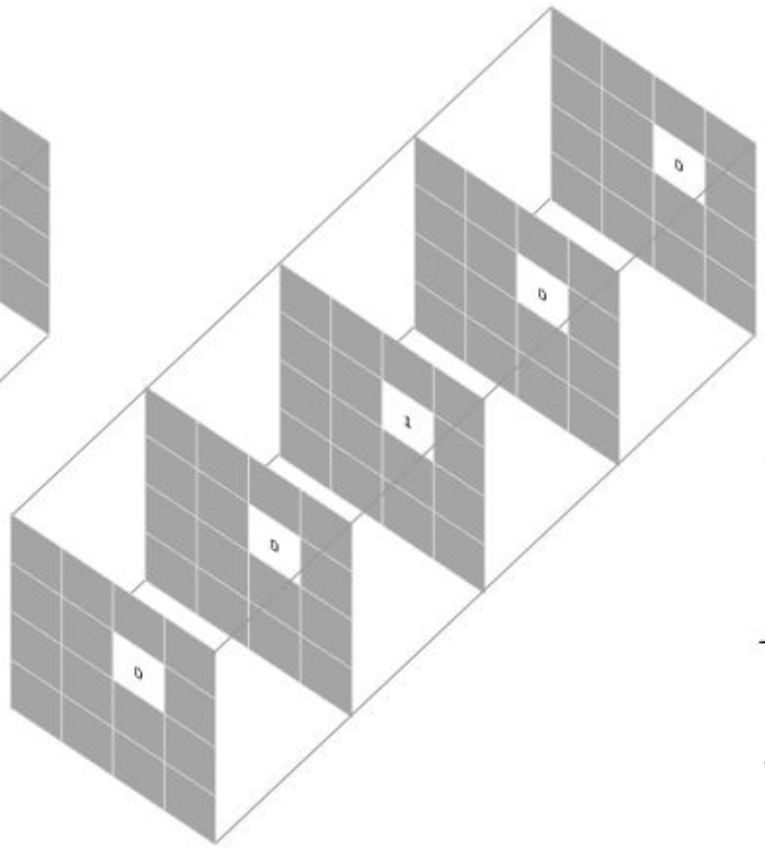
*Semantic segmentation faces an inherent tension between semantics and location: global information resolves **what** while local information resolves **where**... Combining fine layers and coarse layers lets the model make local predictions that respect global structure.* — *Long et al.*

Defining a loss function

The most commonly used loss function for the task of image segmentation is a **pixel-wise cross entropy loss**. This loss examines *each pixel individually*, comparing the class predictions (depth-wise pixel vector) to our one-hot encoded target vector.



Prediction for a selected pixel



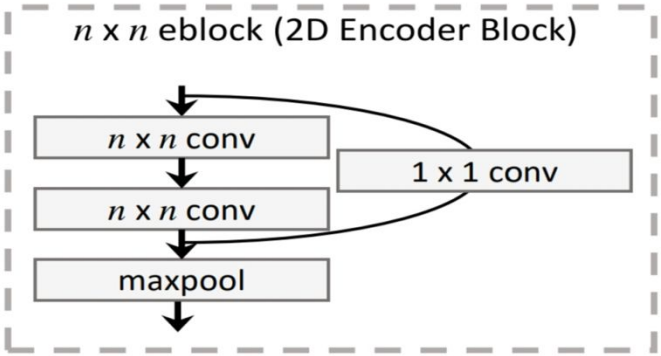
Target for the corresponding pixel

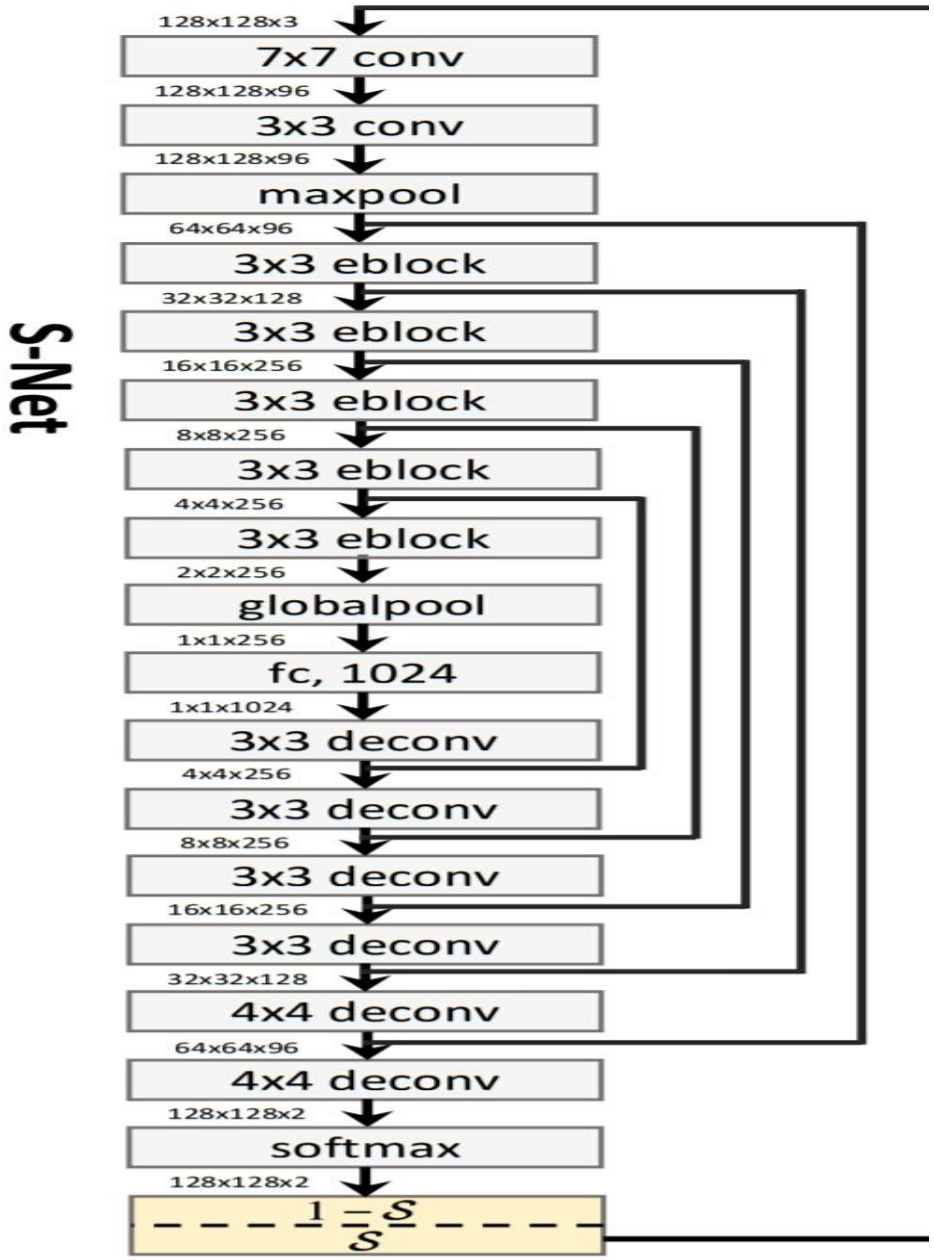
Pixel-wise loss is calculated as the log loss, summed over all possible classes

$$-\sum_{classes} y_{true} \log\left(y_{pred}\right)$$

This scoring is repeated over all **pixels** and averaged

Because the cross entropy loss evaluates the class predictions for each pixel vector individually and then averages over all pixels, we're essentially asserting equal learning to each pixel in the image. This can be a problem if your various classes have unbalanced representation in the image, as training can be dominated by the most prevalent class. Long et al. (FCN paper) discuss weighting this loss for each **output channel** in order to counteract a class imbalance present in the dataset.
Meanwhile, Ronneberger et al. (U-Net paper) discuss a loss weighting scheme for each **pixel** such that there is a higher weight at the border of segmented objects. This loss weighting scheme helped their U-Net model segment cells in biomedical images in a *discontinuous* fashion such that individual cells may be easily identified within the binary segmentation map.


n x n eblock (2D Encoder Block)
n x n conv
n x n conv
maxpool
1 x 1 conv


S-Net
128x128x3
7x7 conv
128x128x96
3x3 conv
128x128x96
maxpool
64x64x96
3x3 eblock
32x32x128
3x3 eblock
16x16x256
3x3 eblock
8x8x256
3x3 eblock
4x4x256
3x3 eblock
2x2x256
globalpool
1x1x256
fc, 1024
1x1x1024
3x3 deconv
4x4x256
3x3 deconv
8x8x256
3x3 deconv
16x16x256
3x3 deconv
32x32x128
4x4 deconv
64x64x96
4x4 deconv
128x128x2
softmax
128x128x2
1 − S
S

Reference-
https://www.jeremyjordan.me/semantic-segmentation/

THANK YOU .