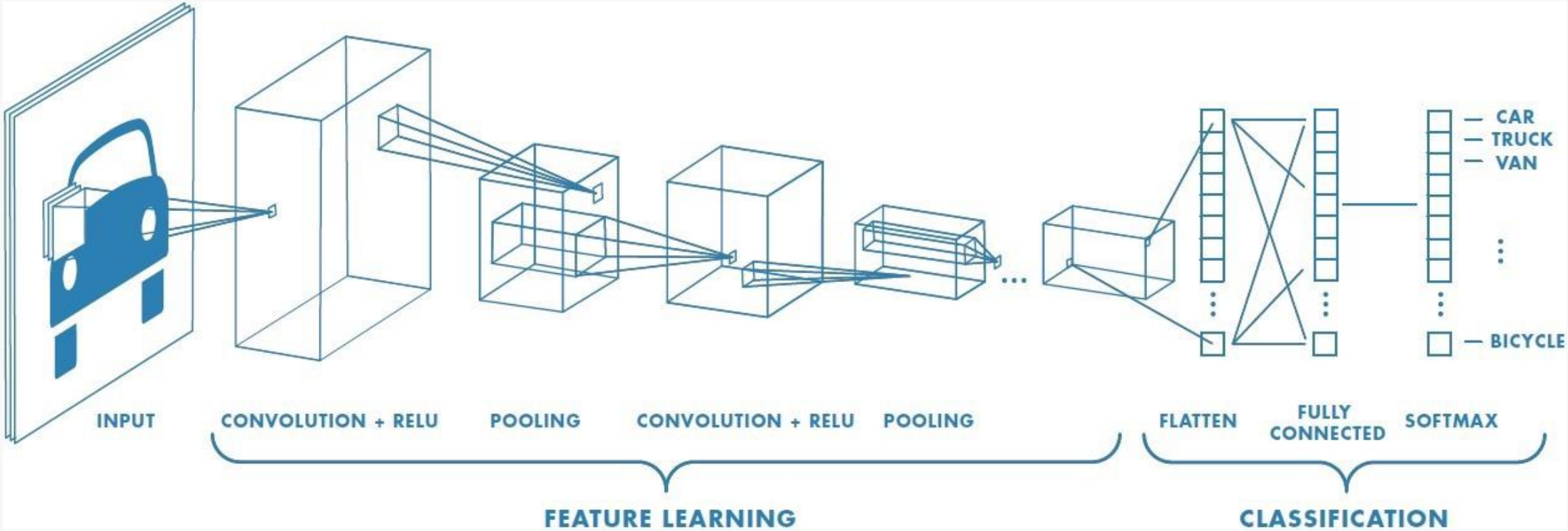
Deep Learning for Computer Vision

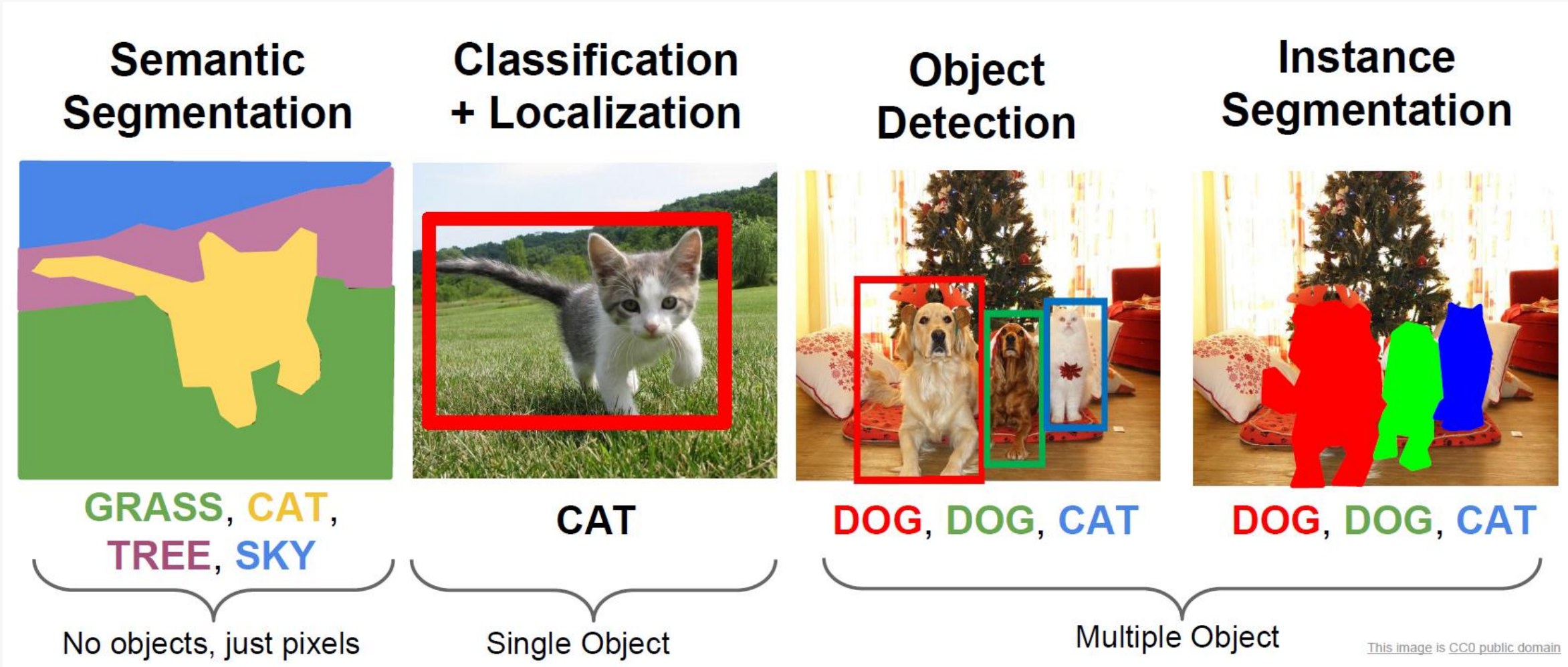
Ahmed Hosny Abdel-Gawad

Senior AI/CV Engineer

So far: Image Classification



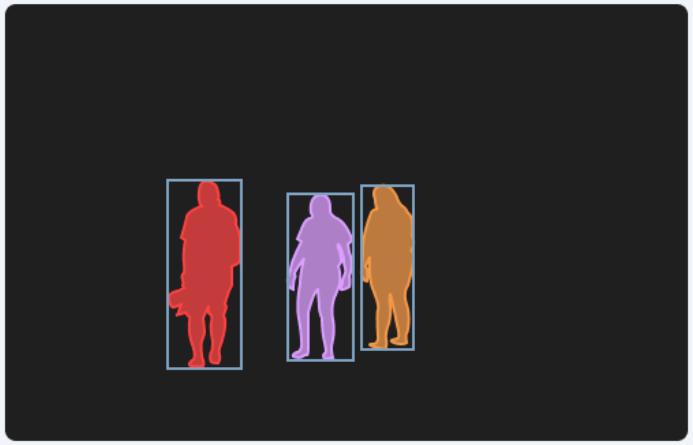
Other CV Tasks



Types of Image Segmentation tasks

(a) Image

(b) Semantic Segmentation



(c) Instance Segmentation

(d) Panoptic Segmentation

Semantic segmentation segments out a broad boundary of objects belonging to a particular class.

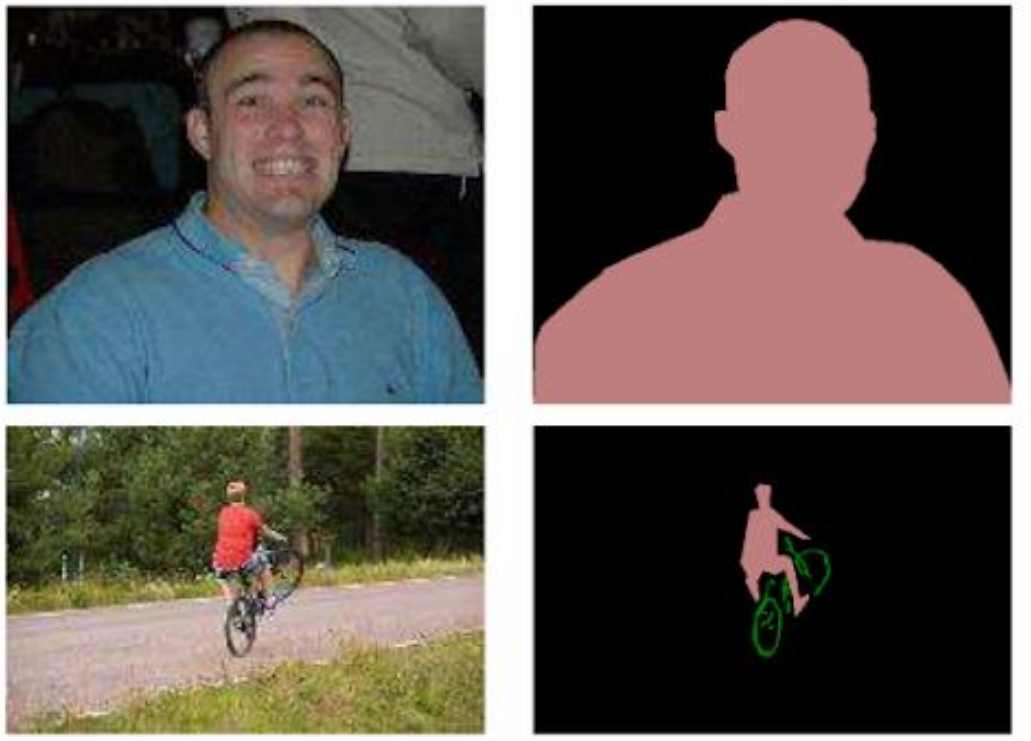
Instance segmentation provides a segment map for each object it views in the image, without any idea of the class the object belongs to.

Panoptic segmentation gives us the segment maps of all the objects of any particular class present in the image.

Semantic Segmentation

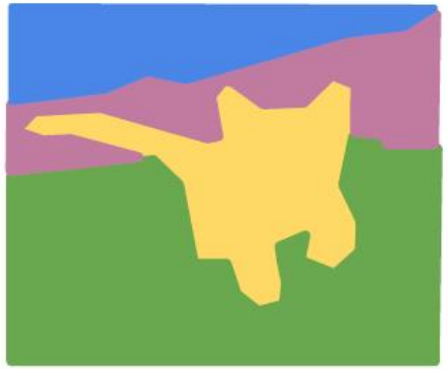
Semantic Segmentation

We know an image is nothing but a collection of pixels. **Semantic segmentation** is the process of classifying each pixel in an image belonging to a certain class and hence can be thought of as a classification problem per pixel.



NOTE: We're not separating instances of the same class; We only care about the category of each pixel.

Semantic Segmentation: The Problem



GRASS, CAT,
TREE, SKY, ...

Paired training data: for each training image,
each pixel is labeled with a semantic category.

?

At test time, classify each pixel of a new image.

Semantic Segmentation: The Goal

Simply, our goal is to take either a RGB color image ($height \times width \times 3$) or a grayscale image ($height \times width \times 1$) and output a segmentation map ($height \times width \times 1$) where the pixel value ($from\ 0\ to\ 255$) of the input image is transformed into a class label value ($0, 1, 2, \ldots\ n$).

segmented

1: Person
2: Purse
3: Plants/Grass
4: Sidewalk
5: Building/Structures

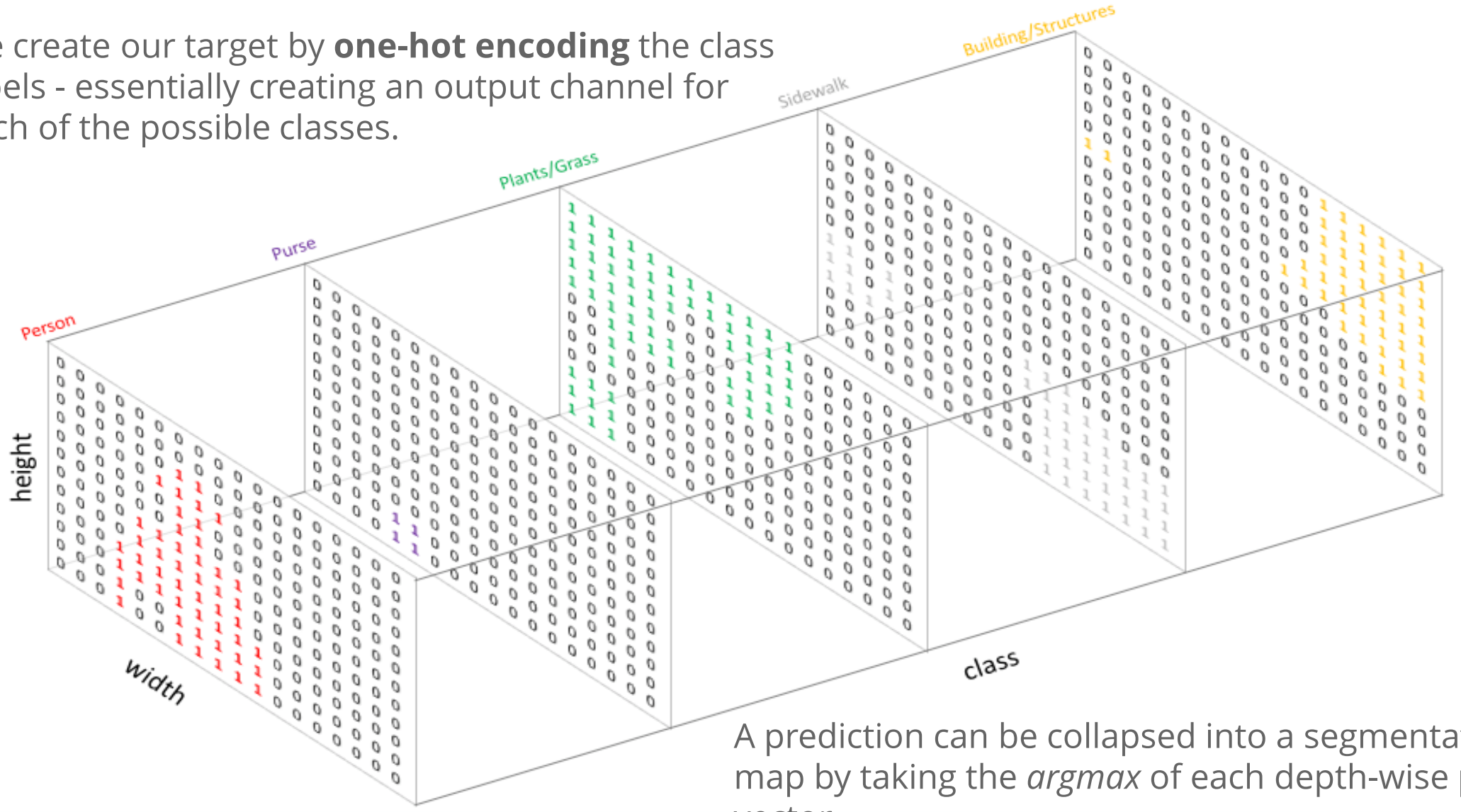
3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	1	1	1	1	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	5	5	5	5	5	5	5
5	5	3	3	3	3	1	1	3	3	5	5	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	4	4	5	5	5
4	4	4	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4

Input

Semantic Labels

Semantic Segmentation: The Label

We create our target by **one-hot encoding** the class labels - essentially creating an output channel for each of the possible classes.



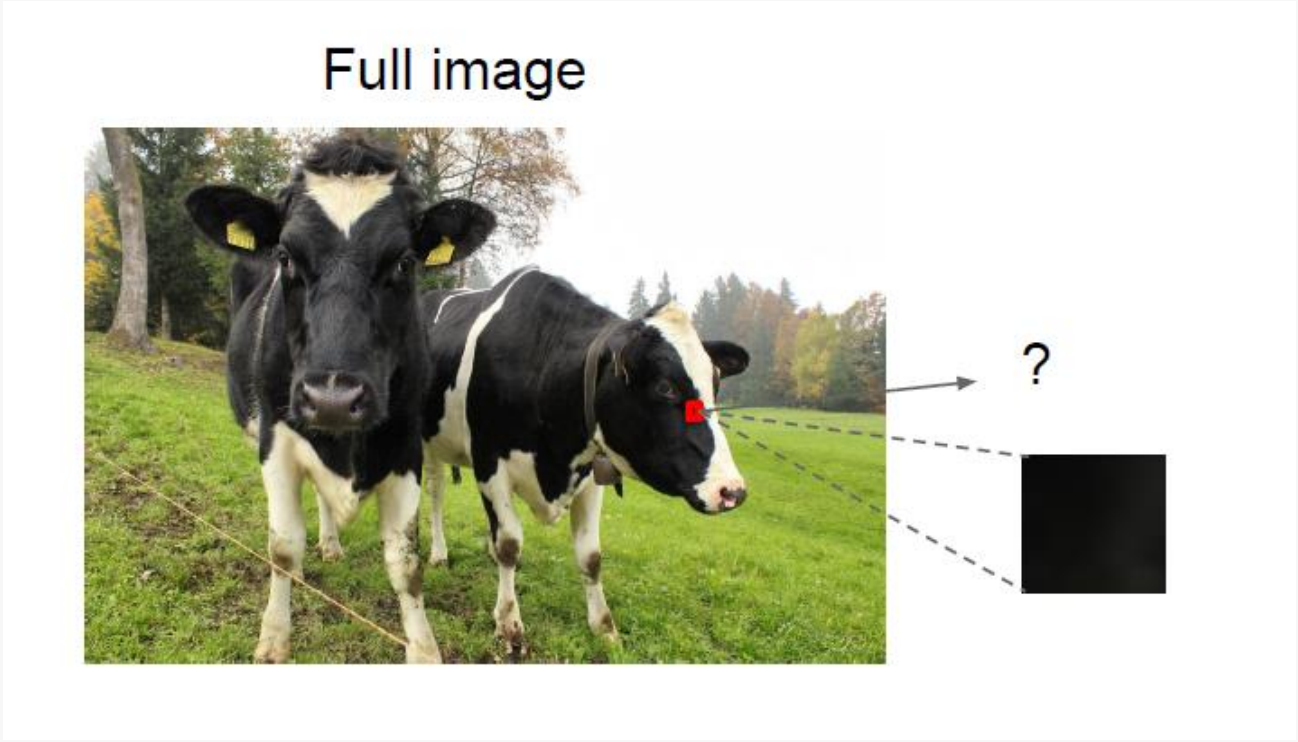
A prediction can be collapsed into a segmentation map by taking the *argmax* of each depth-wise pixel vector.

Overview of Semantic Segmentation

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.

0: Background/Unknown
1: Person
2: Purse
3: Plants/Grass
4: Sidewalk
5: Building/Structures

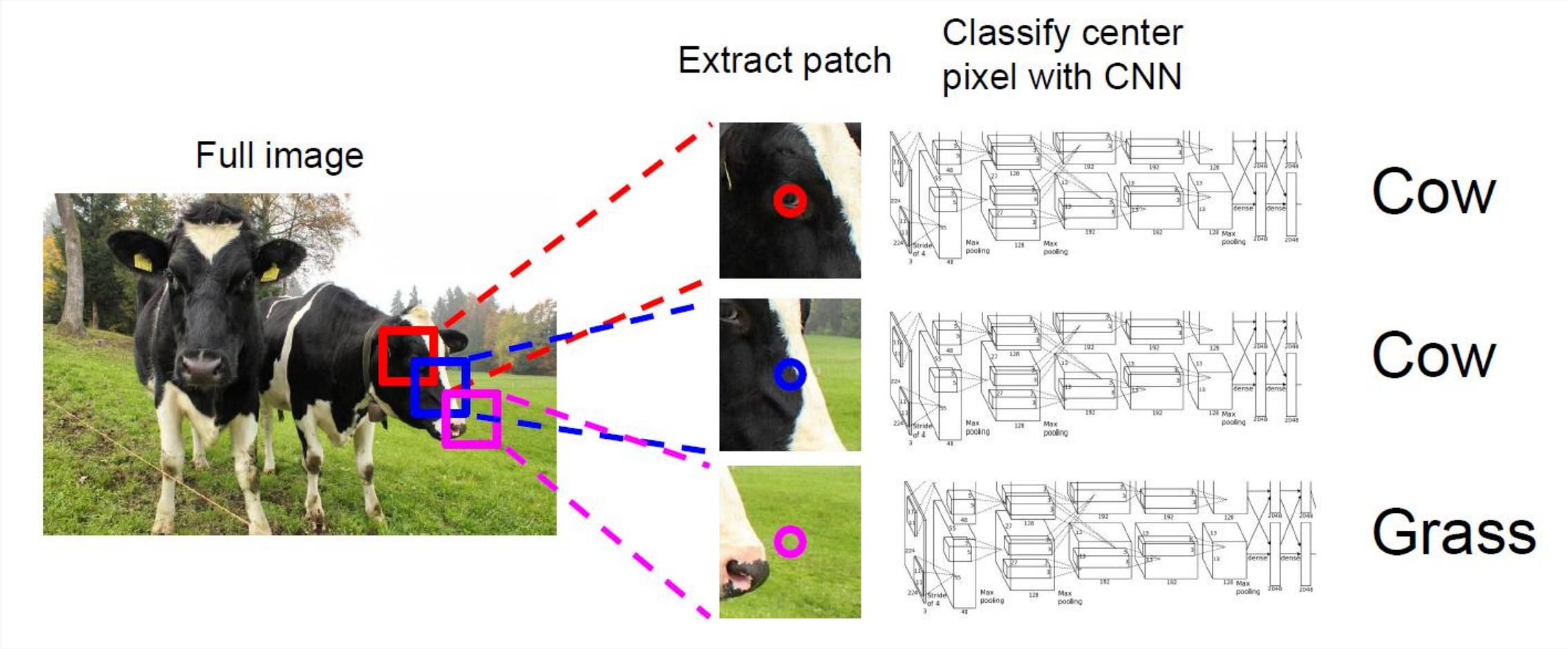
Semantic Segmentation Idea: Sliding Window



Impossible to classify without context

Q: how do we include context?

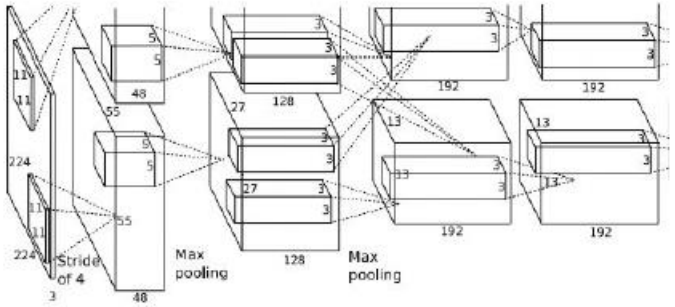
Semantic Segmentation Idea: Sliding Window



Problem: Very inefficient!
Not reusing shared features between overlapping patches.

Semantic Segmentation Idea: Convolution

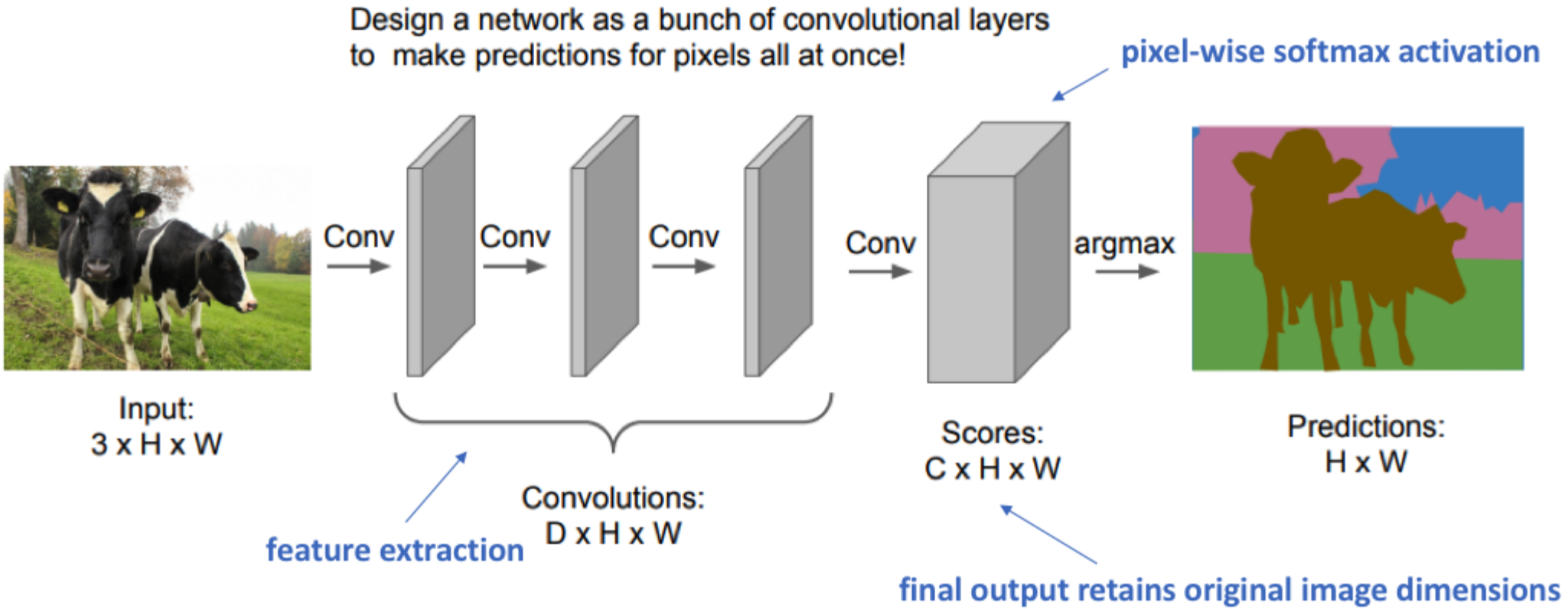
Full image



An intuitive idea: encode the entire image with conv net, and do semantic segmentation on top.

Problem: Classification architectures often reduce feature spatial sizes to go deeper, but semantic segmentation requires the output size to be the same as input size.

Semantic Segmentation Idea: Fully Convolution



Problem: Preserving original image resolution throughout entire network will be computationally expensive.

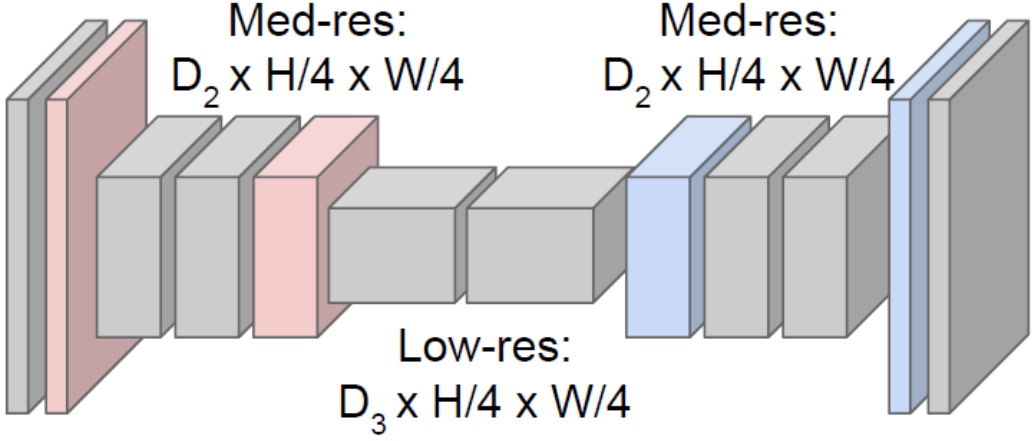
Semantic Segmentation Idea: Fully Convolution

Downsampling:
Pooling, strided convolution

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

Upsampling:
???

Input:
3 x H x W



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

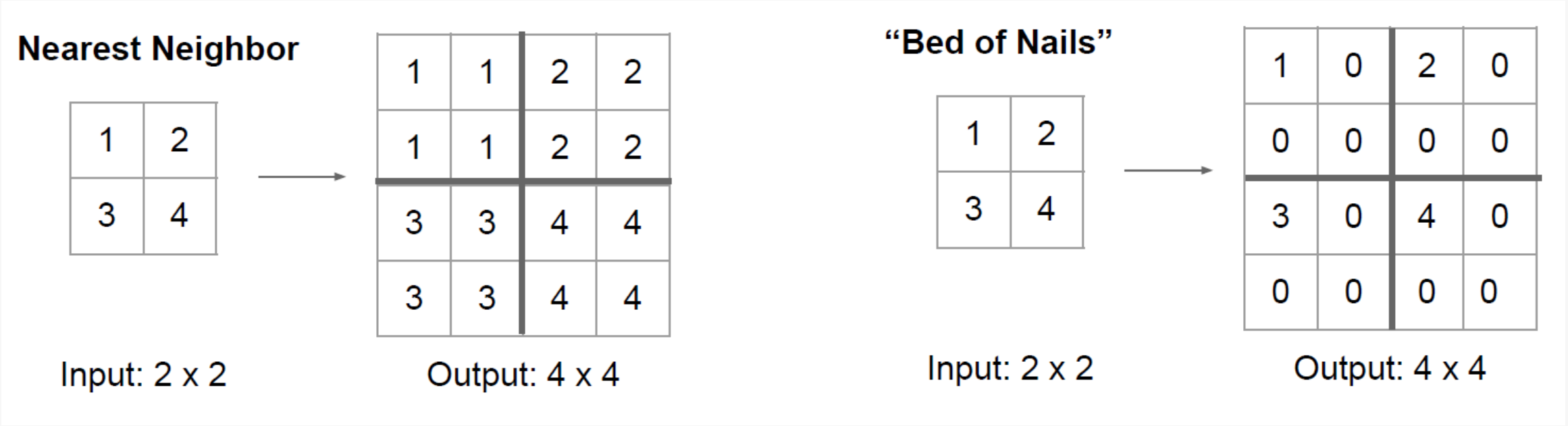
High-res:
D1 x H/2 x W/2

C x H x W

Predictions:
H x W

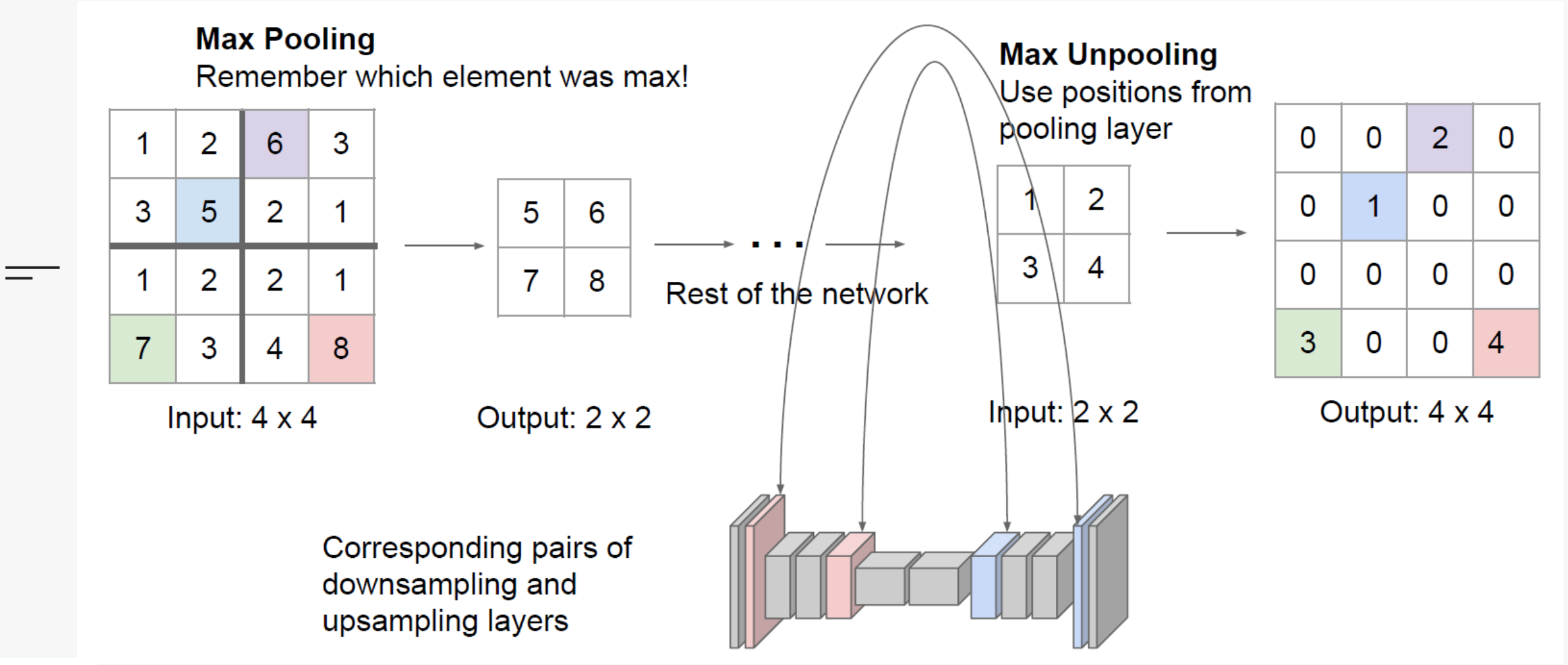
Solution: Make network deep and work at a lower spatial resolution for many of the layers.

In-Network Upsampling: Unpooling



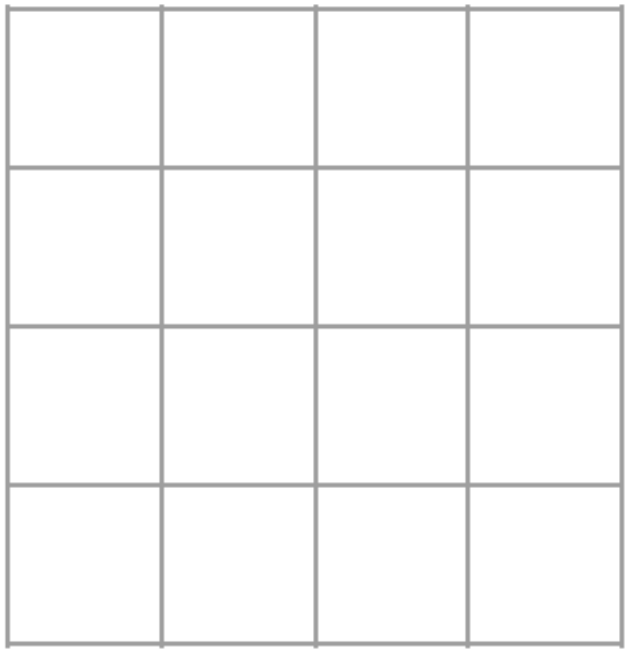
Whereas **pooling** operations **downsample** the resolution by summarizing a local area with a single value (i.e. average or max pooling), **unpooling** operations **upsample** the resolution by distributing a single value into a higher resolution.

In-Network Upsampling: Max Unpooling

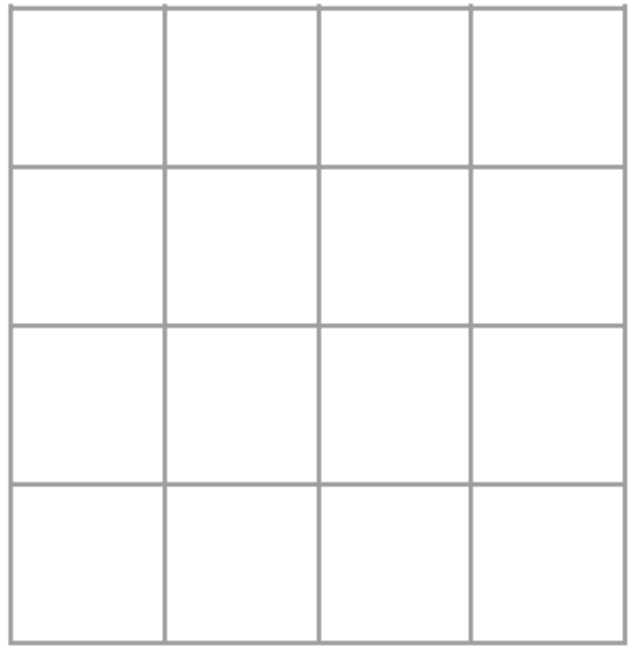


Learnable Upsampling

Recall: Normal 3 x 3 convolution, stride 1 pad 1



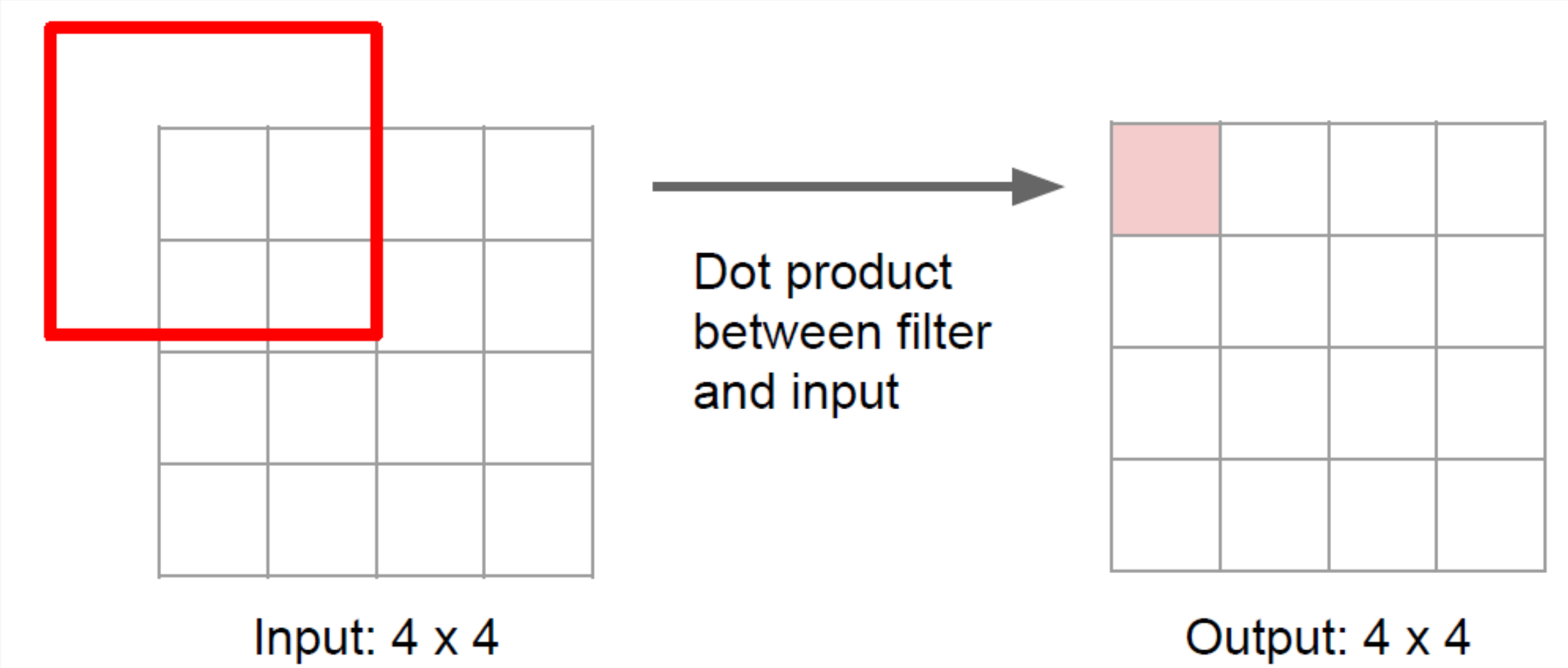
Input: 4 x 4



Output: 4 x 4

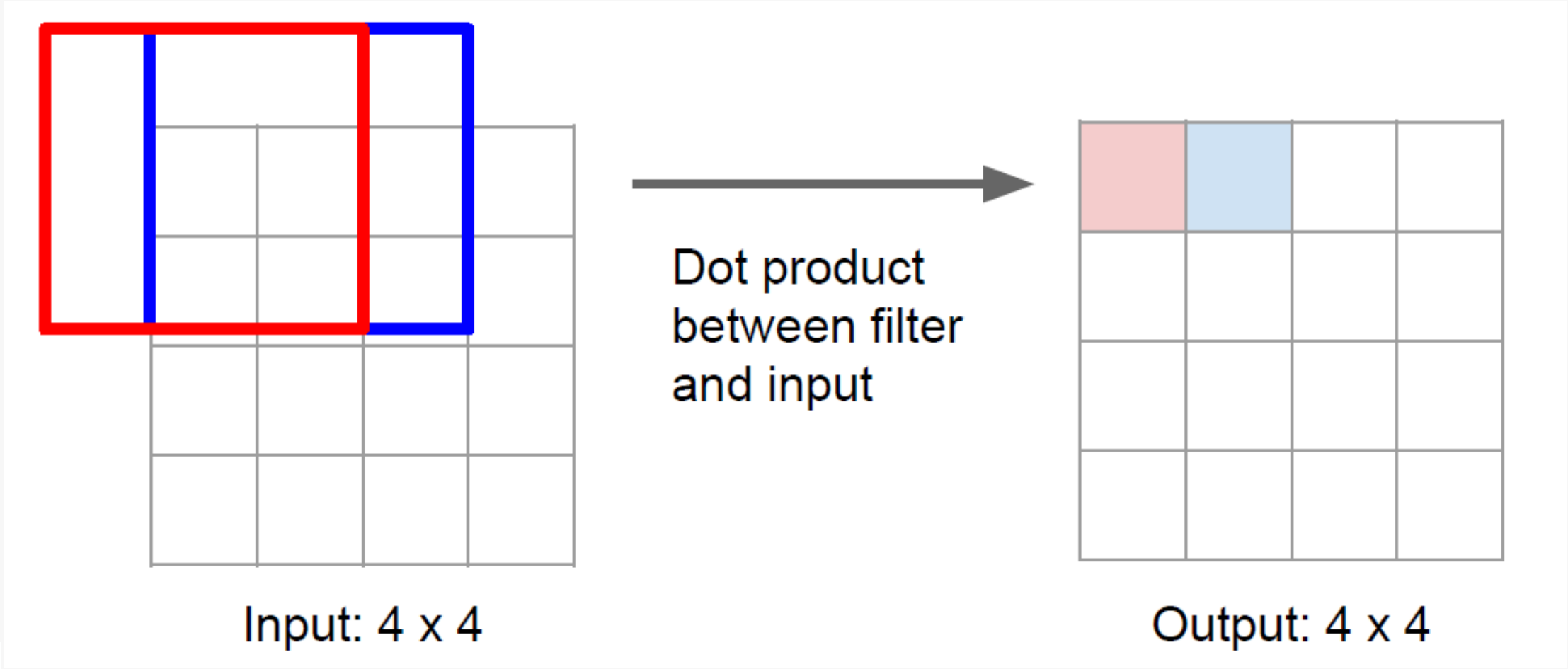
Learnable Upsampling

Recall: Normal 3 x 3 convolution, stride 1 pad 1



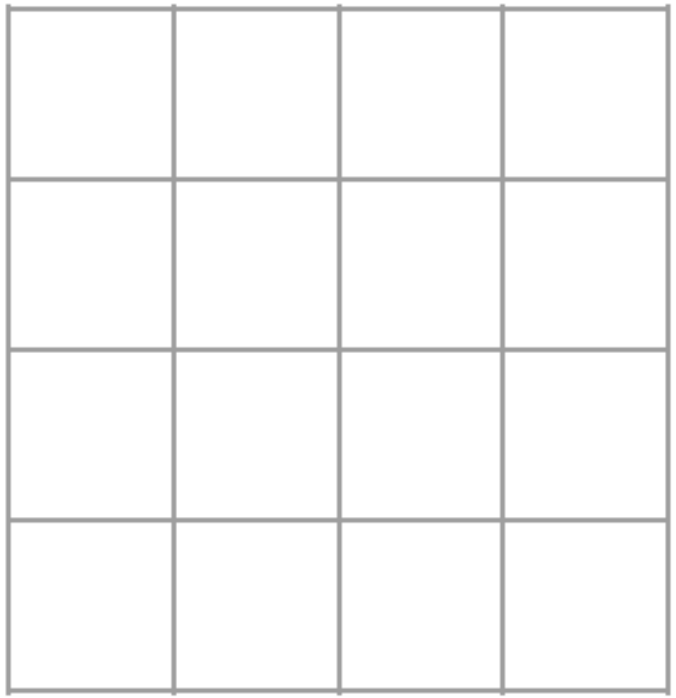
Learnable Upsampling

Recall: Normal 3 x 3 convolution, stride 1 pad 1

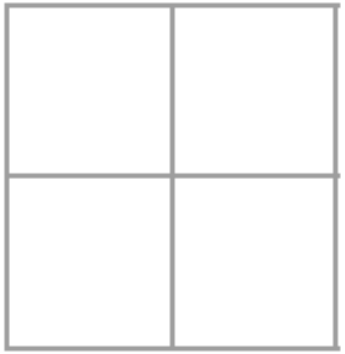


Learnable Upsampling

Recall: Normal 3 x 3 convolution, **stride 2** pad 1



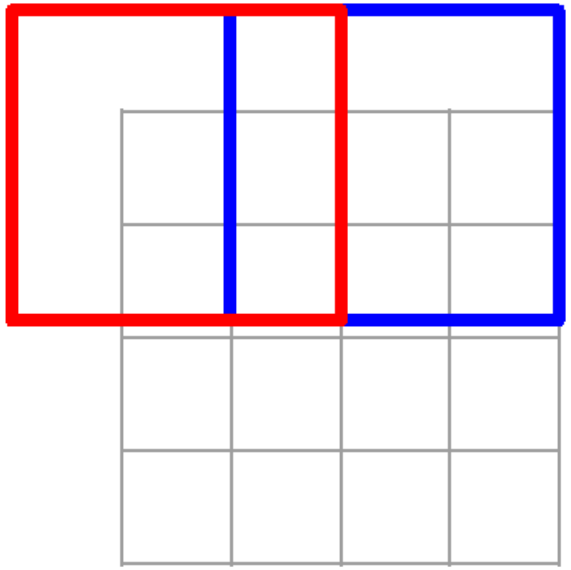
Input: 4 x 4



Output: 2 x 2

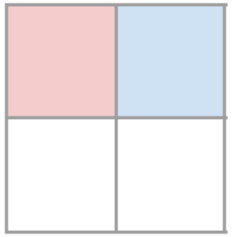
Learnable Upsampling

Recall: Normal 3 x 3 convolution, **stride 2** pad 1



Input: 4 x 4

Dot product between filter and input



Output: 2 x 2

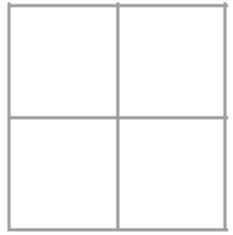
Filter moves 2 pixels in the input for every one pixel in the output

Stride gives ratio between movement in input and output

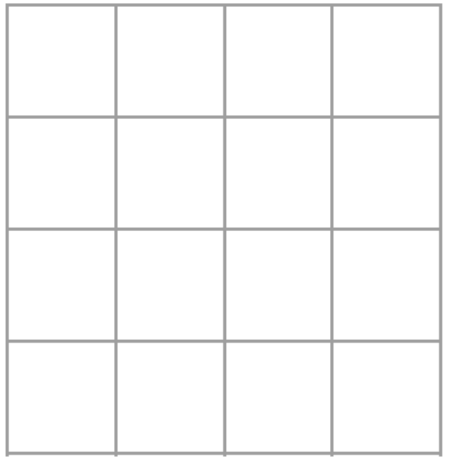
We can interpret strided convolution as “learnable downsampling”.

Learnable Upsampling: Transposed Convolution

3 x 3 **transpose** convolution, stride 2 pad 1



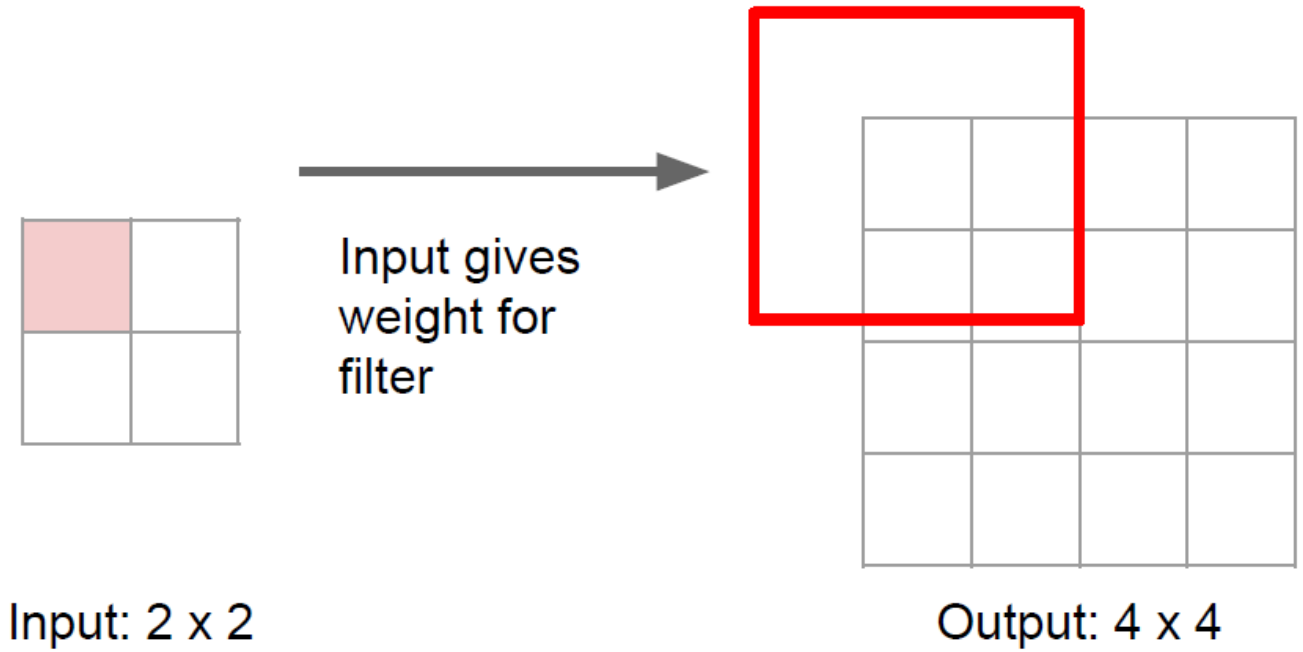
Input: 2 x 2



Output: 4 x 4

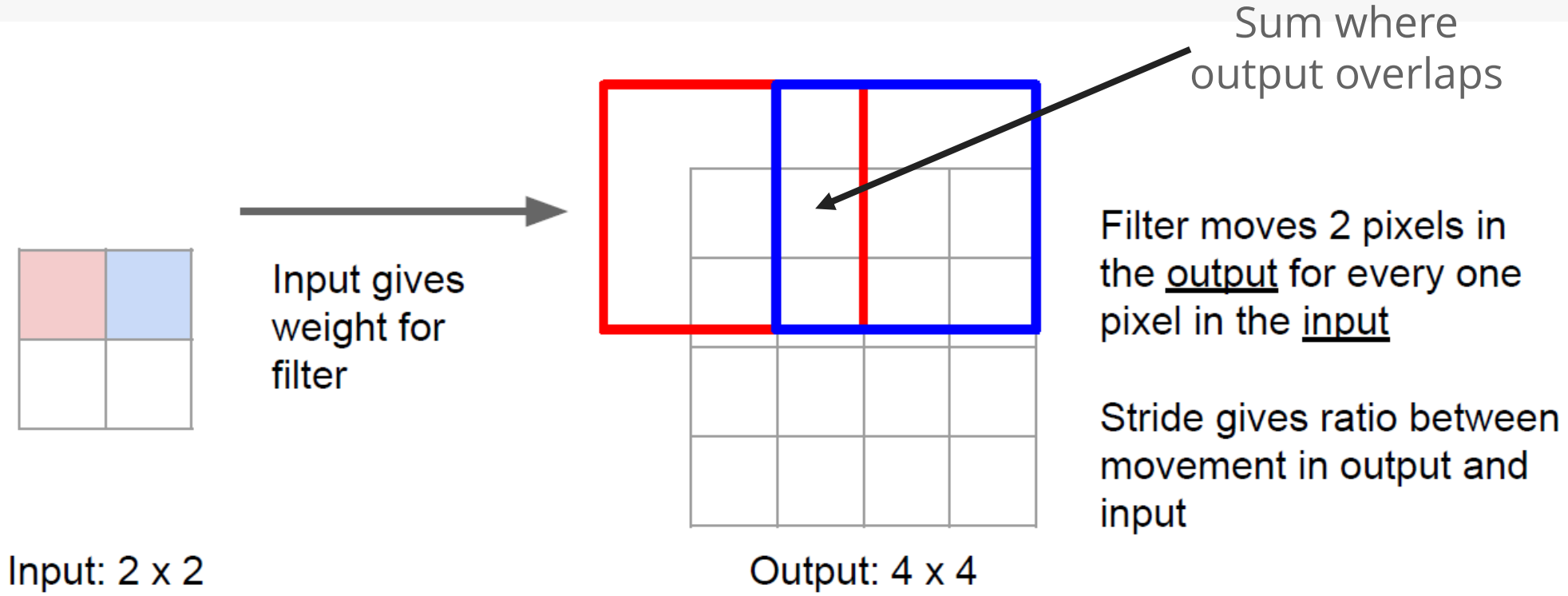
Learnable Upsampling: Transposed Convolution

3 x 3 **transpose** convolution, stride 2 pad 1

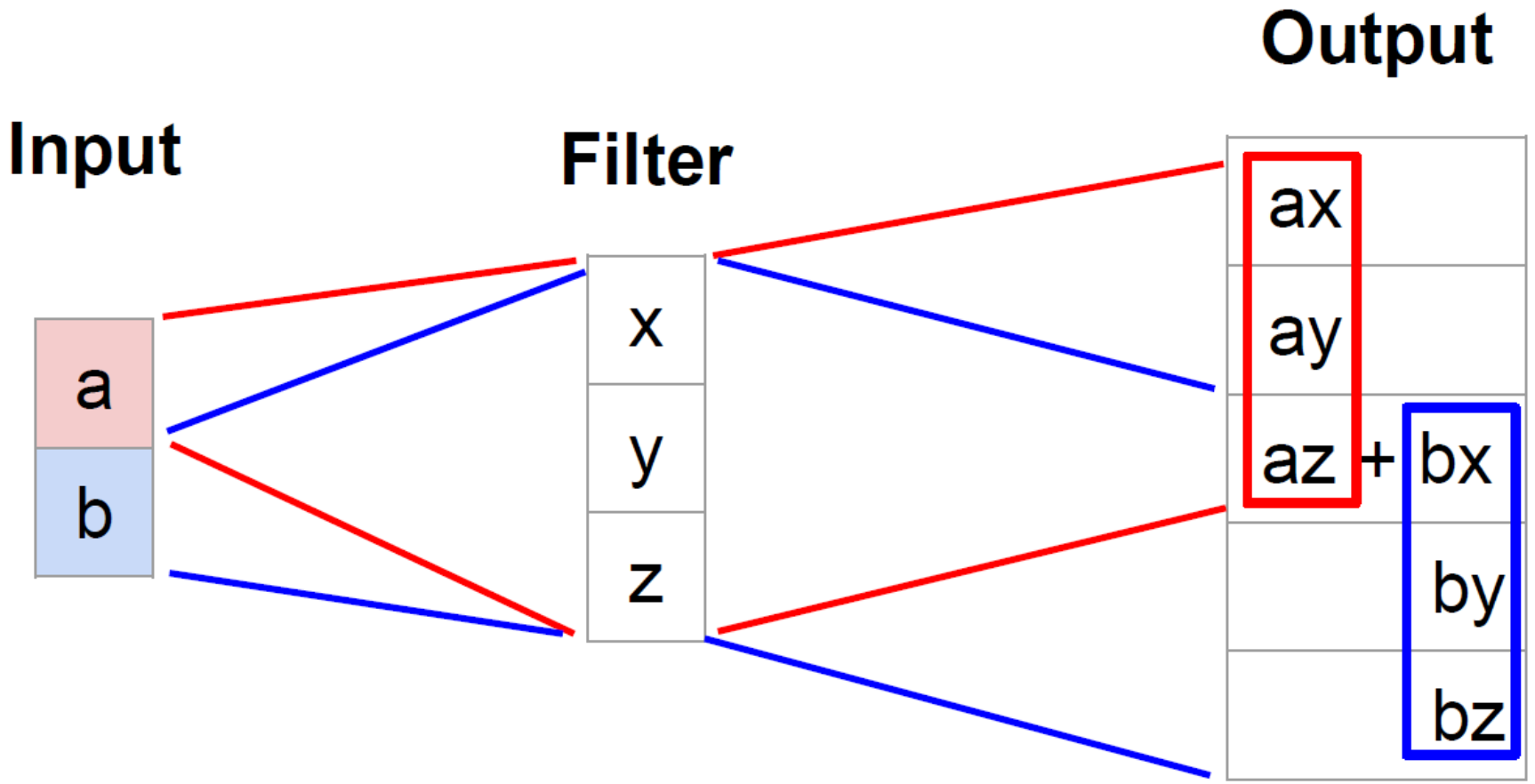


Learnable Upsampling: Transposed Convolution

3 x 3 **transpose** convolution, stride 2 pad 1



Learnable Upsampling: 1D Example

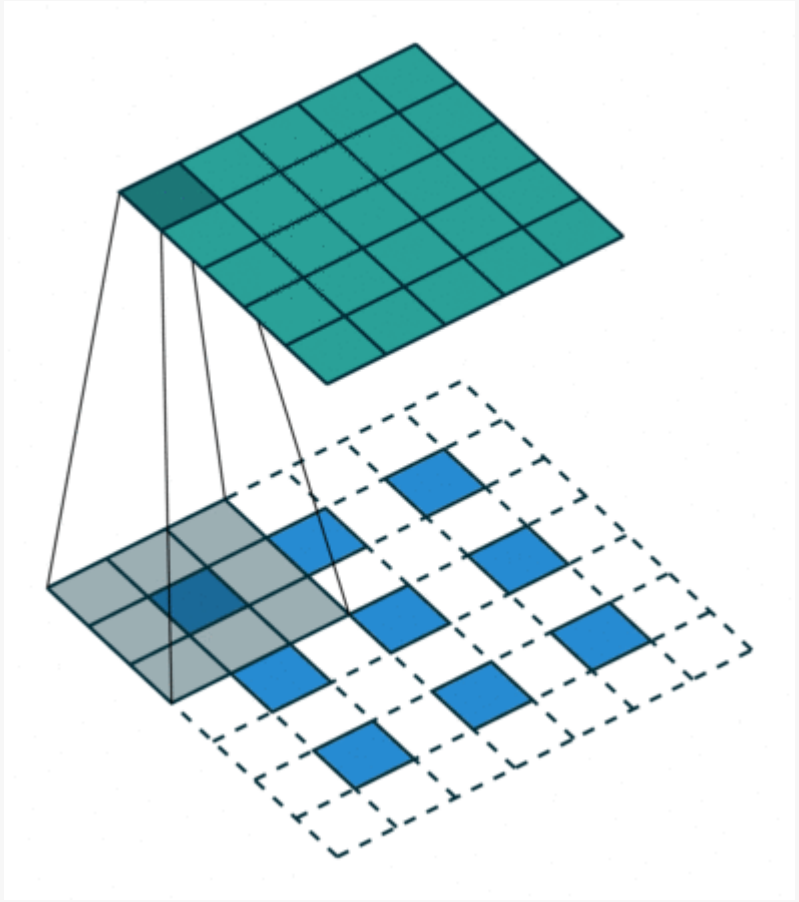


Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

Learnable Upsampling: 1D Example

For filter sizes which produce an overlap in the output feature map (e.g. 3x3 filter with stride 2), 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.



Convolution as Matrix Multiplication: 1D Example

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & 0 & x & y & x & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=2, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

Example: 1D transpose conv, kernel size=3, stride=2, padding=0

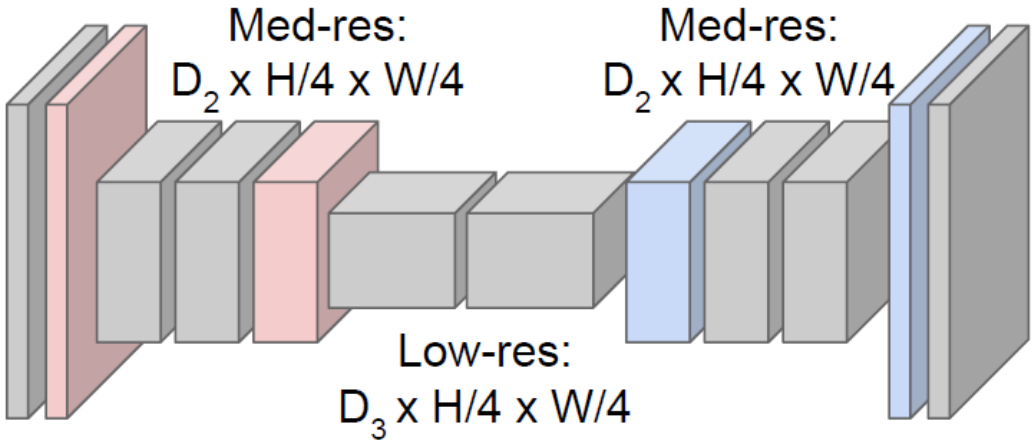
Semantic Segmentation Idea: Fully Convolution

Downsampling:
Pooling, strided convolution

Upsampling:
Unpooling or strided transpose convolution

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

Input:
3 x H x W



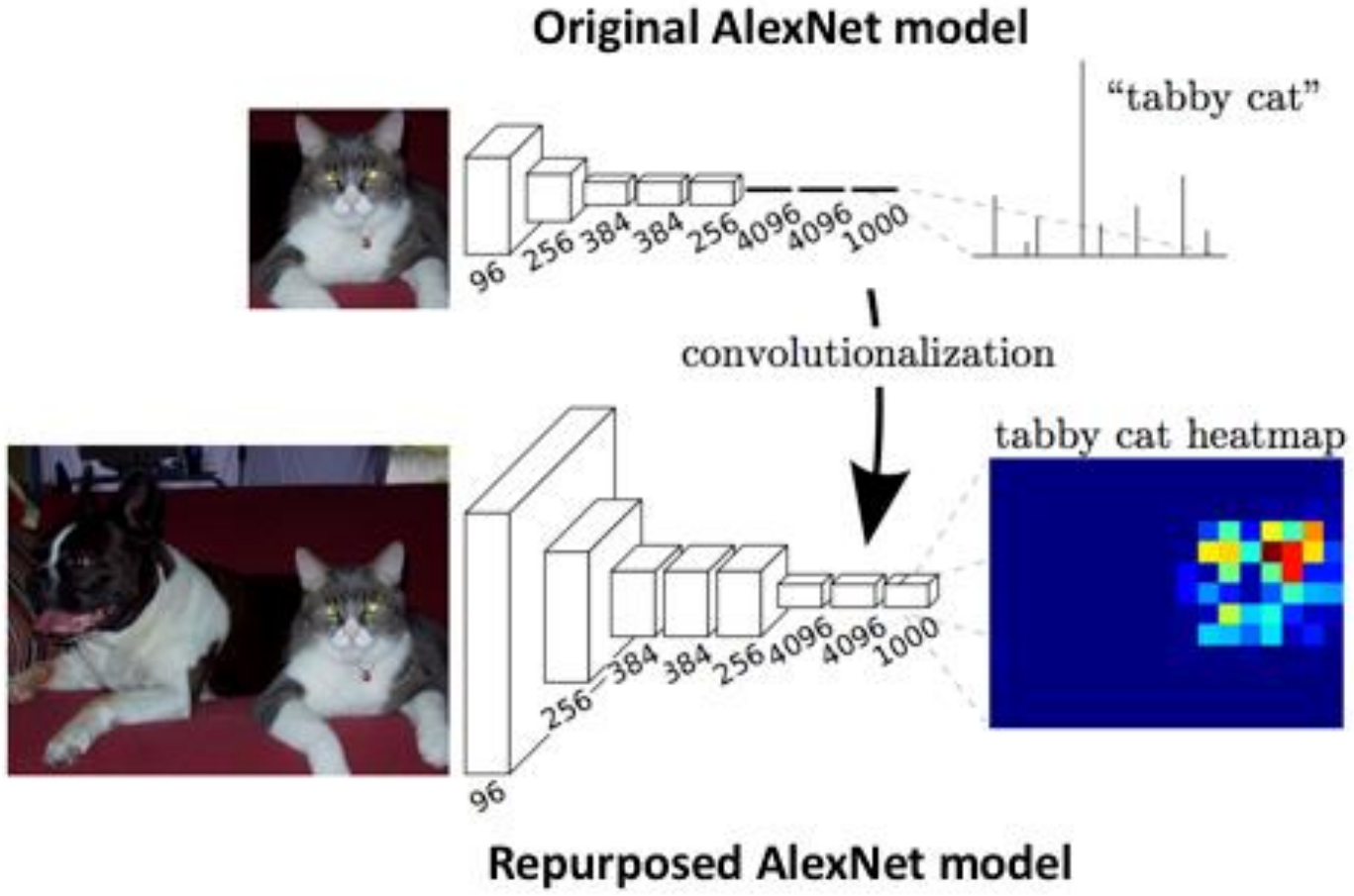
High-res:
D1 x H/2 x W/2

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

C x H x W

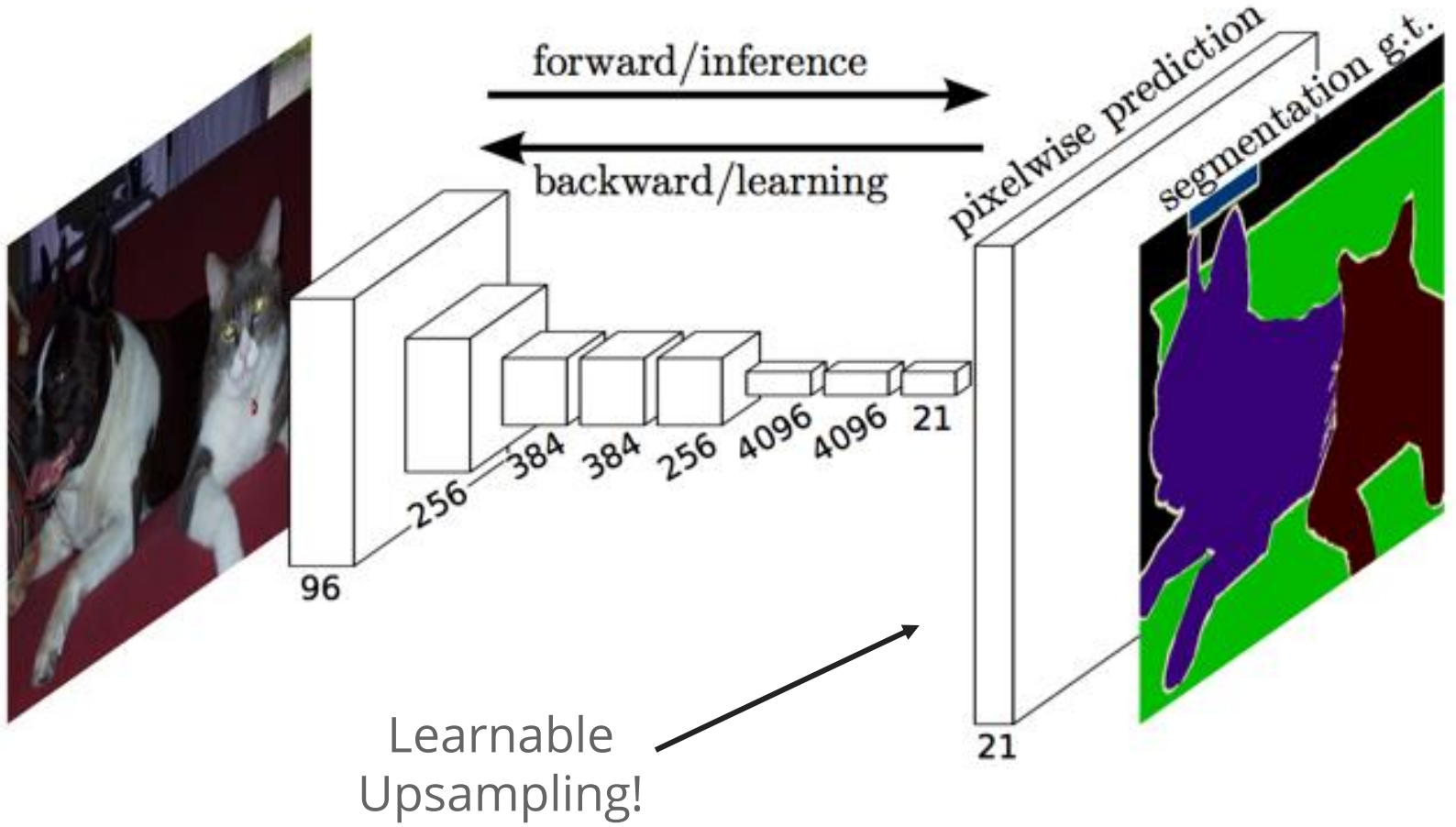
Predictions:
H x W

Semantic Segmentation Idea: Fully Convolution



The encoder produces a ***coarse*** feature map which is then refined by the decoder module.

Semantic Segmentation Idea: Fully Convolution

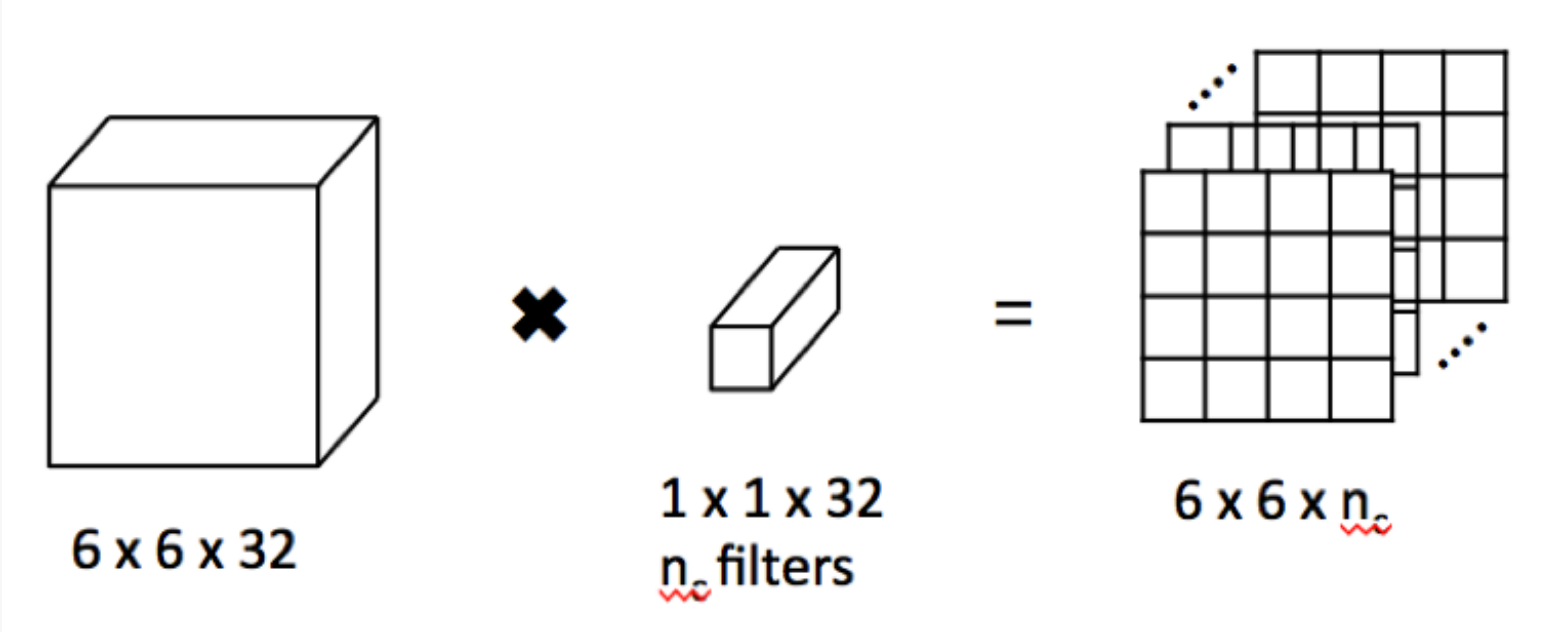


How to Get Class Label Heatmap?

By applying, a 1x1 convolution, we can shrink the number of feature map.

The $1x1$ convolution looks at each 36 $(6x6)$ different positions, and takes the element-wise product between 32 numbers on the left and 32 number in the.

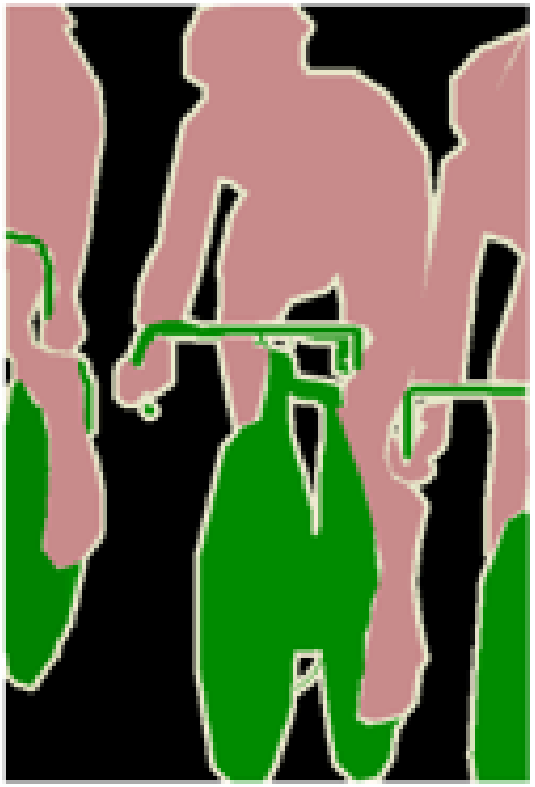
If we apply "nc" number of $1x1$ convolutional filters, then the output will be, in our example, $6x6xnc$.



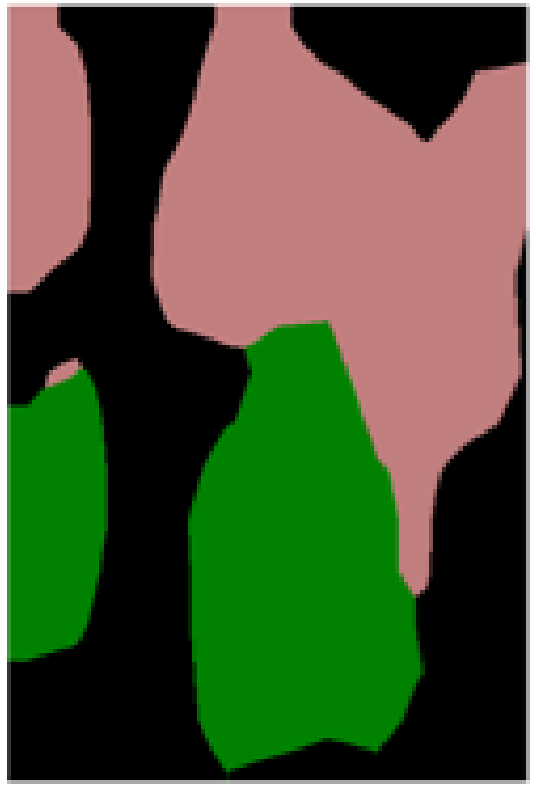
Semantic Segmentation Idea: Fully Convolution

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**.

Ground truth target

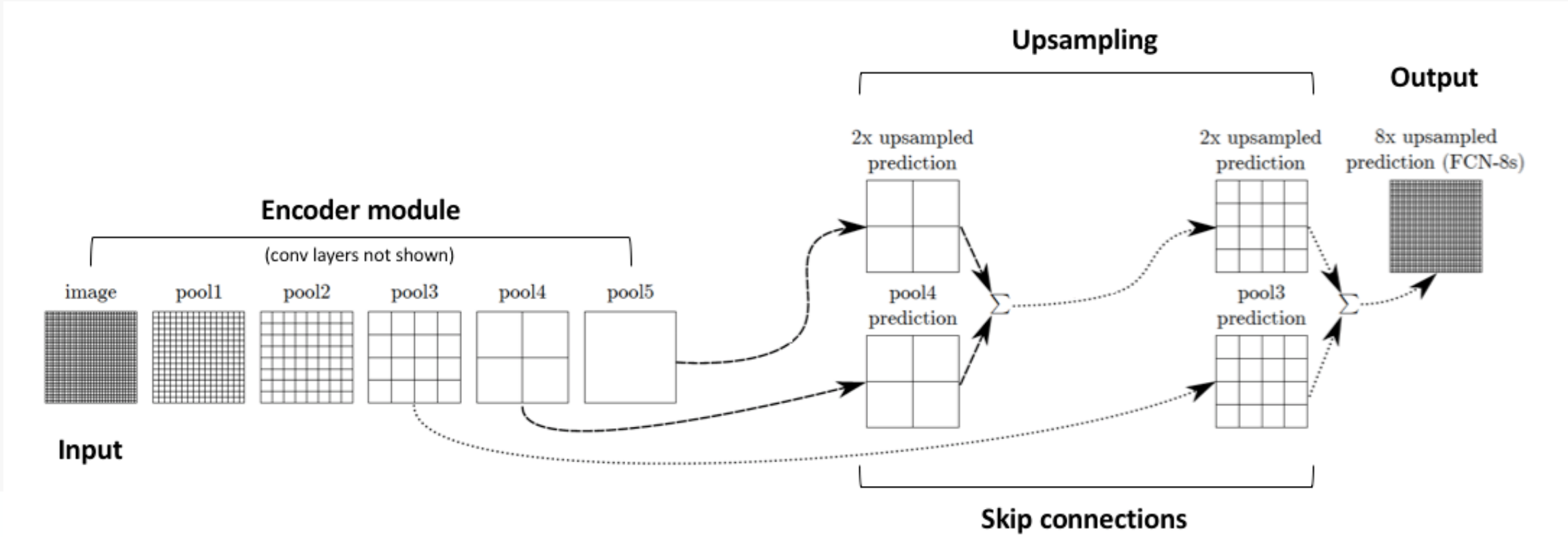


Predicted segmentation



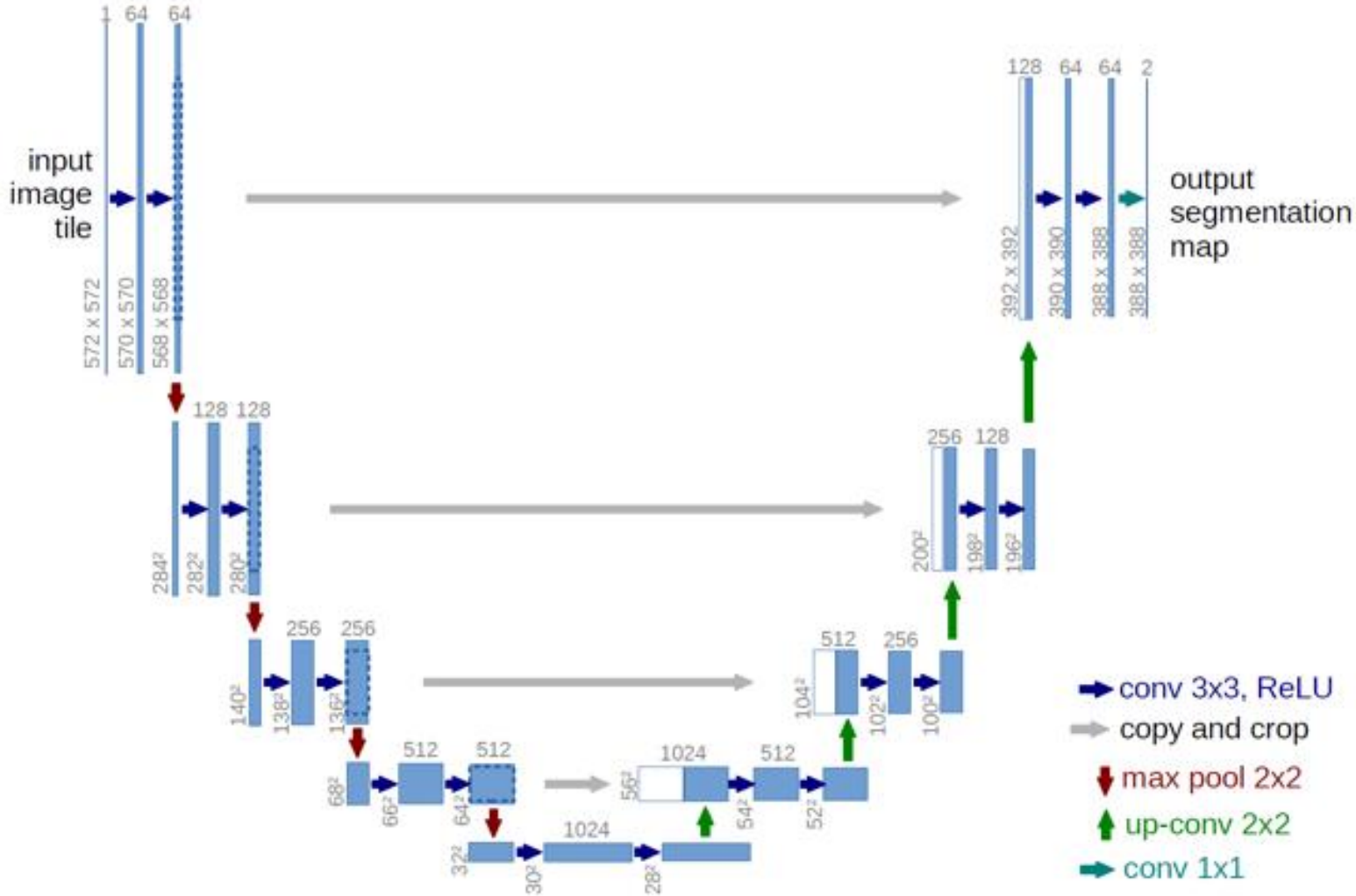
Semantic Segmentation Idea: Adding Skip Connections

Adding ***"skip connections"*** from earlier layers (prior to a downsampling operation) and summing these two feature maps should provide the necessary detail in order to reconstruct accurate shapes for segmentation boundaries.



Semantic Segmentation Architecture: UNet

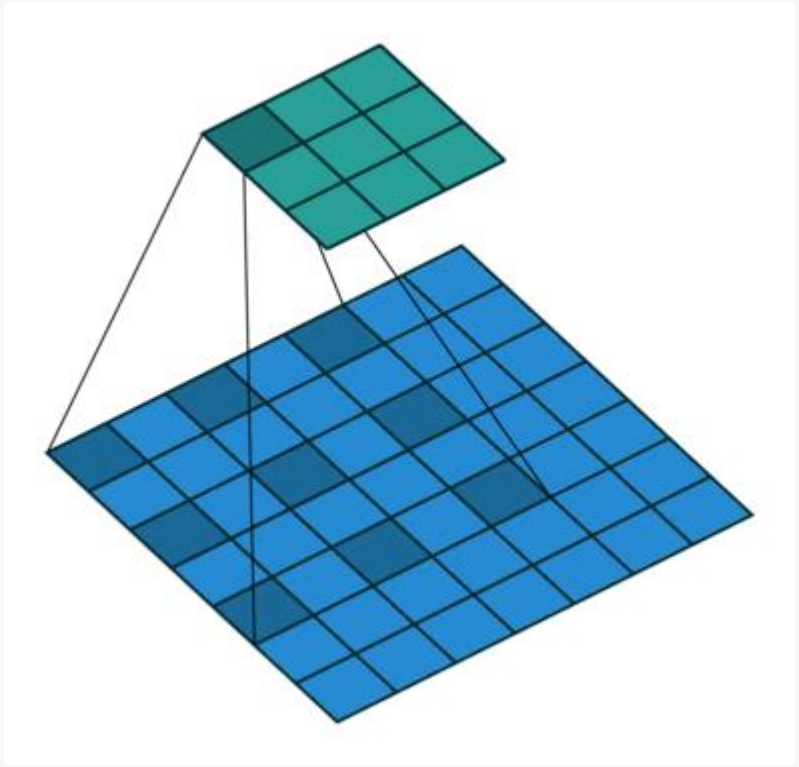
U-Net ,which first introduced skip connections in Deep Learning as a solution for the loss of information observed in downsampling layers of typical encoder-decoder networks.



Semantic Segmentation Idea: Dilated/Atrous Convolutions

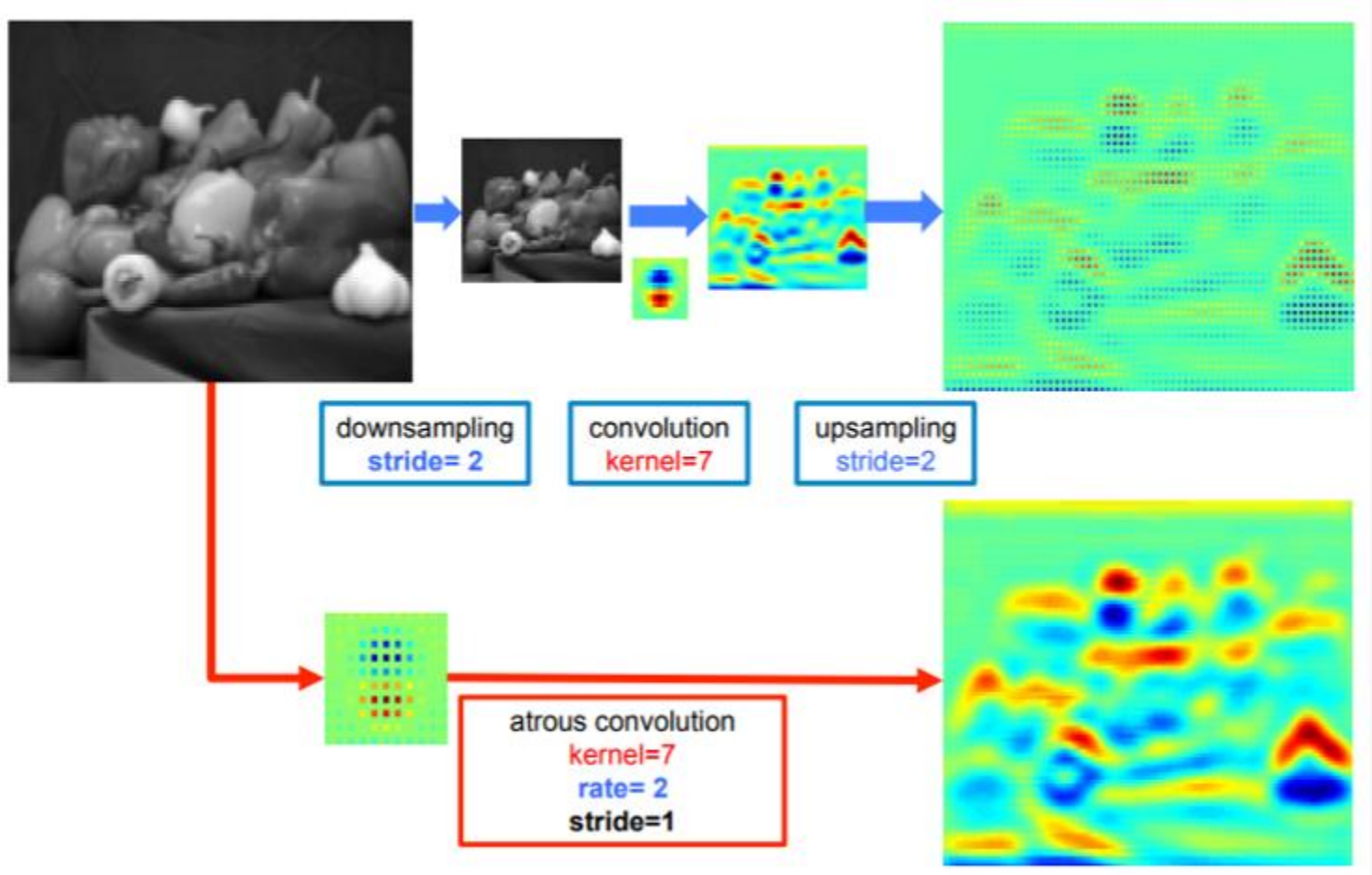
Dilated convolutions provide an alternative approach towards gaining a wide field of view while preserving the full spatial dimension.

Assume k is the dilation rate. If $k = 1$, the convolution will be normal. IF $k = 2$, then we skip one pixel per input.



Semantic Segmentation Idea: Dilated/Atrous Convolutions

Atrous convolution achieves a **denser** representation than the normal one.



Semantic Segmentation Architecture: DeepLab

DeepLab made use of **atrous convolutions** replacing simple pooling operations and preventing significant information loss while downsampling.

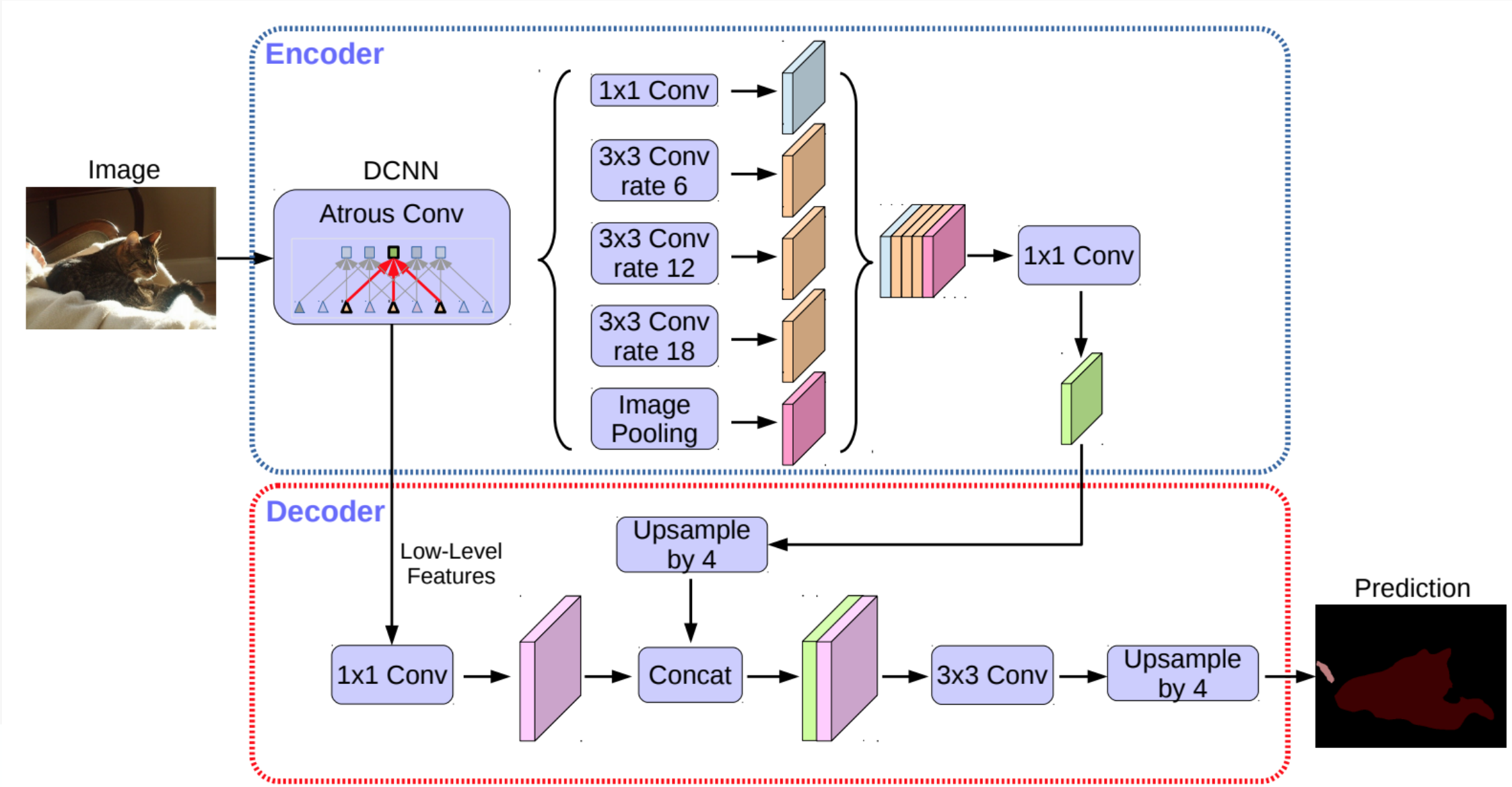
DeepLab V2 used **Atrous Spatial Pyramid Pooling (ASPP)**.

The idea was to apply multiple atrous convolutions with different sampling rates and concatenate them together to get greater accuracy.

Conditional Random Field (CRF) operates a post-processing step and tries to improve the results produced to define shaper boundaries.

It works by classifying a pixel based not only on it's label but also based on other pixel labels.

Semantic Segmentation Architecture: DeepLab



Defining a Loss Function

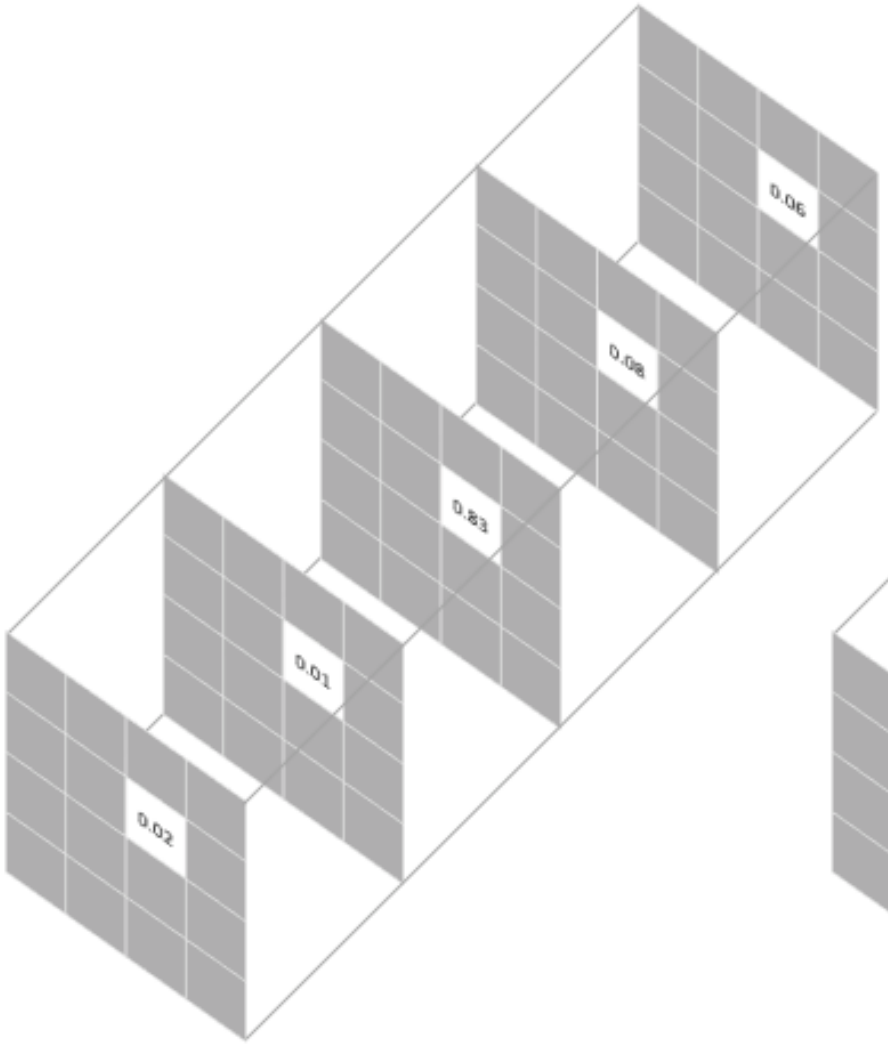
Recall that semantic segmentation is a classification task where the final result yields segmentation maps based on the class label.

Since semantic segmentation is a classification task, we conclude that loss functions will be somewhat similar to what has been used in general classification tasks.

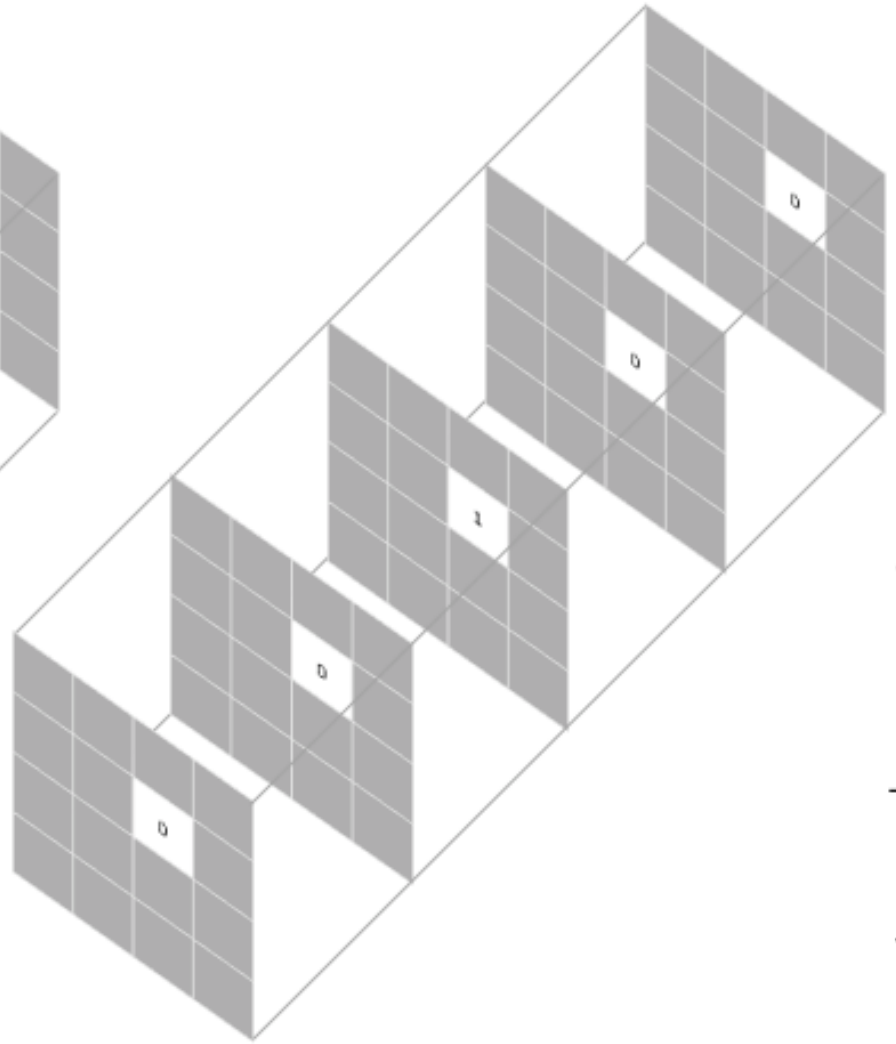
The most commonly used loss function for the task of image segmentation is a **pixel-wise cross entropy loss**.

This loss examines each pixel individually, comparing the class predictions (depth-wise pixel vector) to our one-hot encoded target vector.

Pixel-Wise Cross Entropy Loss



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

Balanced Cross Entropy Loss

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.

Balanced Cross-Entropy loss adds a weighting factor to each class, which is represented by the Greek letter alpha, [0, 1].

Alpha could be the inverse class frequency or a hyper-parameter that is determined by cross-validation. The alpha parameter replaces the actual label term in the Cross-Entropy equation.

$$BalancedCrossEntropy = -\sum_{i=1}^{i=n} \alpha_i log_b(p_i)$$

Focal Loss

Focal loss focuses on the examples that the model gets wrong rather than the ones that it can confidently predict, ensuring that predictions on hard examples improve over time rather than becoming overly confident with easy ones.

In the case of the misclassified sample, the $\boldsymbol{pt}$ is small, making the modulating factor approximately or very close to $\mathbf{1}$.

As the confidence of the model increases, that is, $\boldsymbol{pt} \rightarrow \mathbf{1}$, modulating factor will tend to $\mathbf{0}$, thus down-weighting the loss value for well-classified examples.

$$\mathrm{FL}(p_\mathrm{t}) = -\alpha_\mathrm{t}(1 - p_\mathrm{t})^\gamma \log(p_\mathrm{t}).$$

Dice Loss

Dice Loss is based on the Dice coefficient, which is essentially a measure of overlap between two samples. This measure ranges from $\mathbf{0}$ to $\mathbf{1}$ where a Dice coefficient of $\mathbf{1}$ denotes perfect and complete overlap.

$$Dice = \frac{2\,|A \cap B|}{|A| + |B|}$$

Dice Loss

We can approximate $|A \cap B|$ as the element-wise multiplication between the prediction and target mask, and then sum the resulting matrix.

$$|A \cap B| = \begin{bmatrix} 0.01 & 0.03 & 0.02 & 0.02 \\ 0.05 & 0.12 & 0.09 & 0.07 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \xrightarrow{\text{element-wise multiply}} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix} \xrightarrow{\text{sum}} 7.41$$

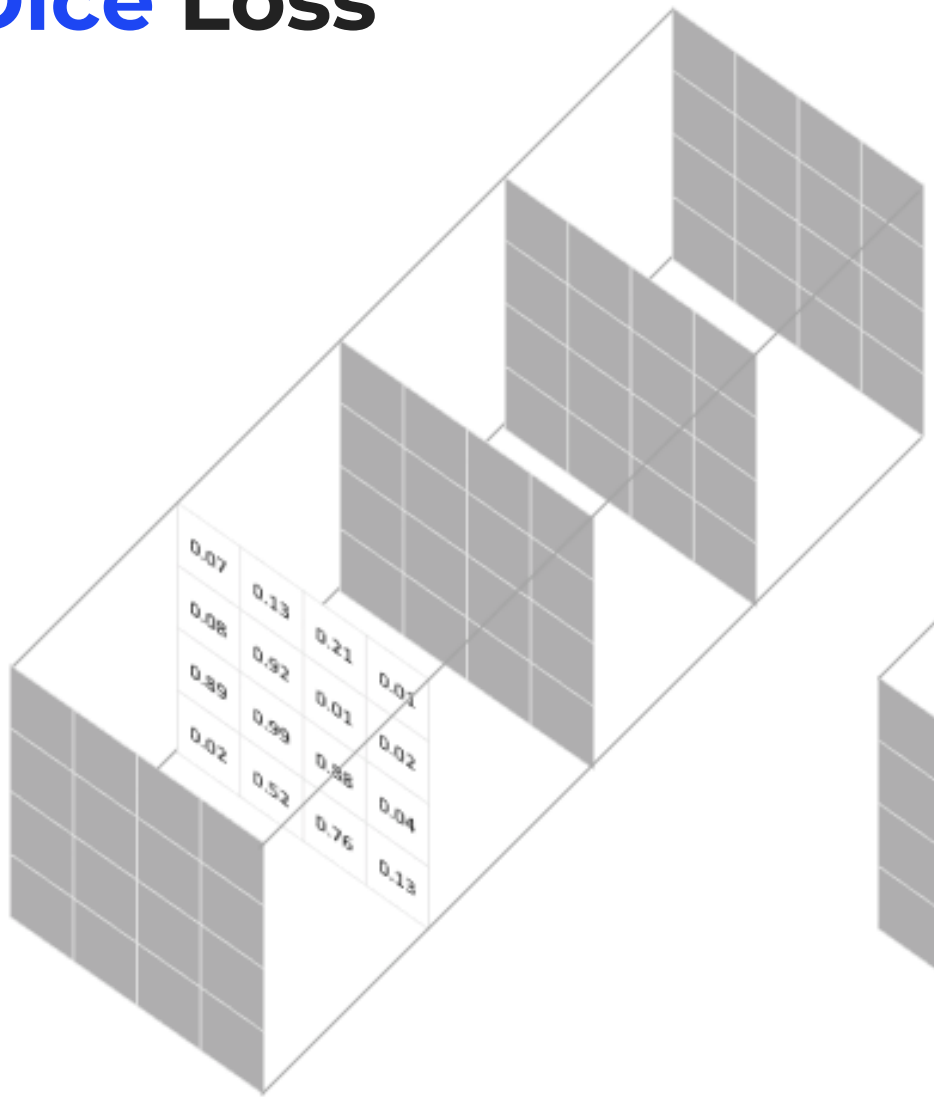
prediction target

In order to quantify $|A|$ and $|B|$, some researchers use the simple sum whereas other researchers prefer to use the squared sum for this calculation.

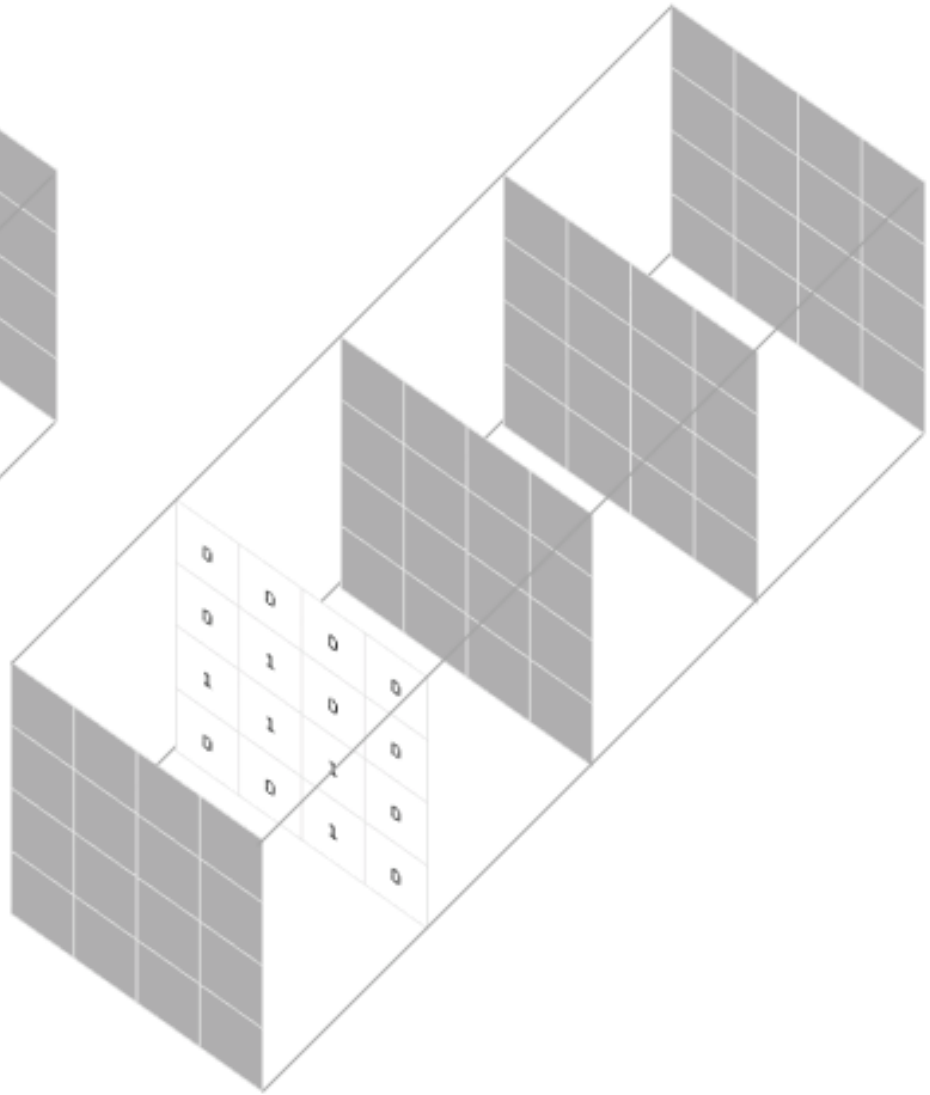
$$|A| = \begin{bmatrix} 0.01 & 0.03 & 0.02 & 0.02 \\ 0.05 & 0.12 & 0.09 & 0.07 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix}^{2\ (optional)} \xrightarrow{\text{sum}} 7.82$$

$$|B| = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}^{2\ (optional)} \xrightarrow{\text{sum}} 8$$

Dice Loss



Prediction for a selected class



Target for the corresponding class

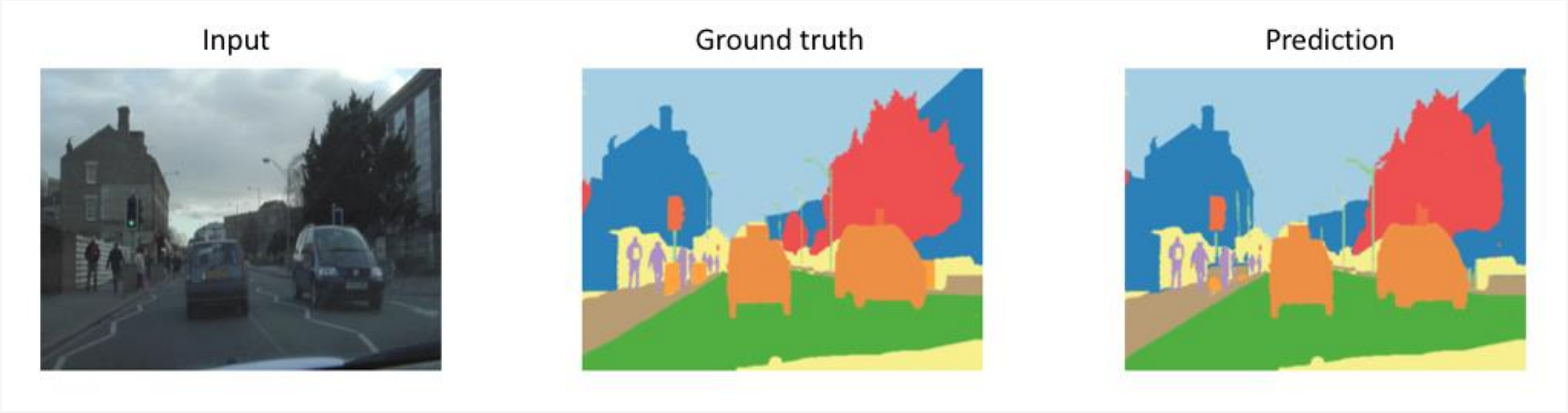
Soft Dice coefficient is calculated for each class mask

$$1 - \frac{2\sum_{pixels} y_{true} y_{pred}}{\sum_{pixels} y_{true}^2 + \sum_{pixels} y_{pred}^2}$$

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

Evaluating Segmentation Models

Recall that the task of semantic segmentation is simply to predict the class of each pixel in an image.



Our prediction output shape matches the input's spatial resolution (width and height) with a channel depth equivalent to the number of possible classes to be predicted. Each channel consists of a binary mask which labels areas where a specific class is present.

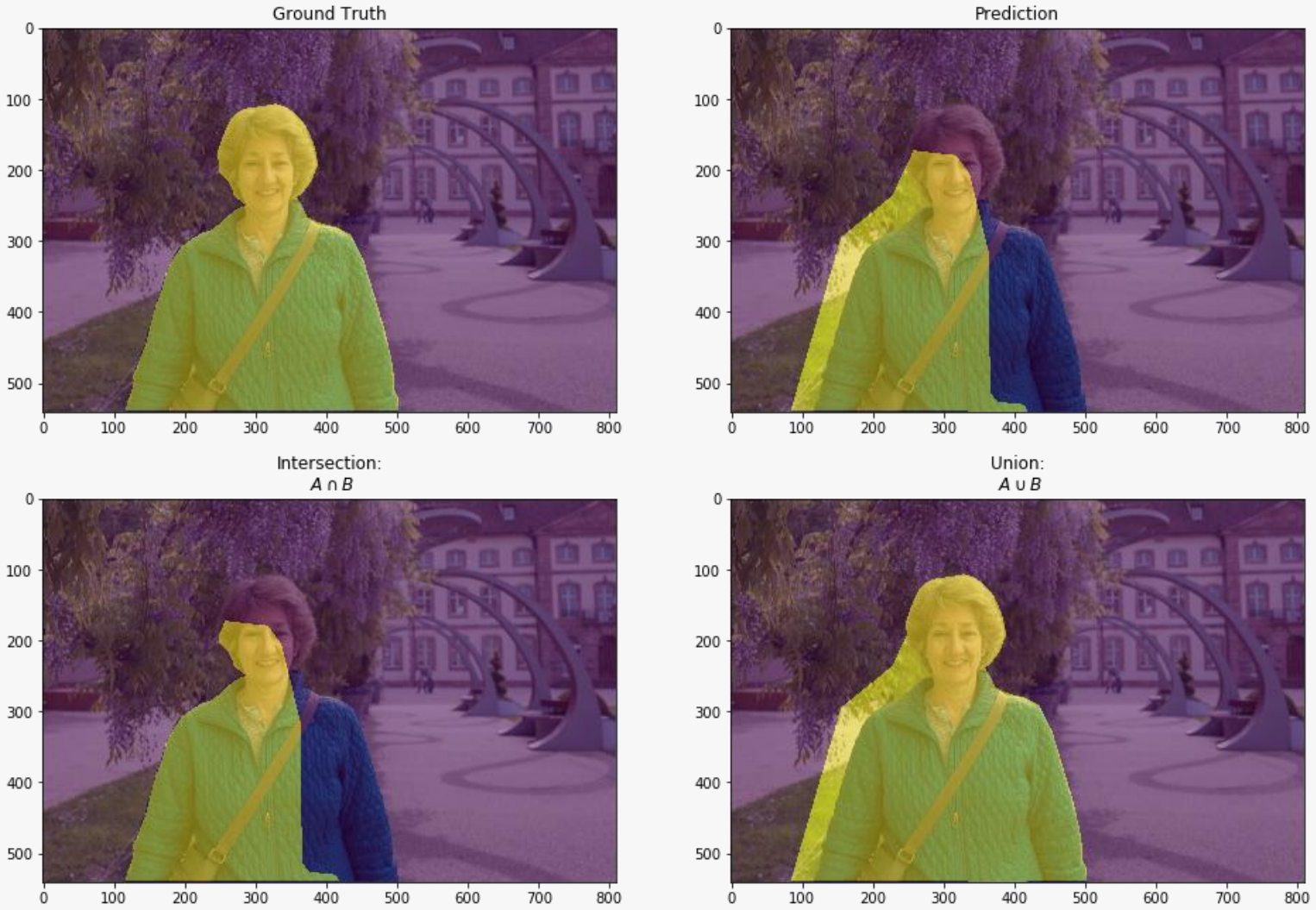
Intersection over Union IoU

The Intersection over Union (IoU) metric, also referred to as the Jaccard index, is essentially a method to quantify the percent overlap between the target mask and our prediction output.

Quite simply, the IoU metric measures the number of pixels common between the target and prediction masks divided by the total number of pixels present across ***both*** masks.

$$IoU = \frac{target \cap predictiont}{target \cup prediction}$$

Intersection over Union IoU



The IoU score is calculated for each class separately and then averaged over all classes to provide a global, mean IoU score of our semantic segmentation prediction.

Pixel Accuracy

An alternative metric is to simply report the percent of pixels in the image which were correctly classified.

The pixel accuracy is commonly reported for each class separately as well as globally across all classes.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

This metric can sometimes provide misleading results when the class representation is small within the image, as the measure will be biased in mainly reporting how well you identify negative case (i.e. where the class is not present).

Links

https://segmentation-modelspytorch.readthedocs.io/en/latest/

https://albumentations.ai/docs/

Thank You!