



Semantic Segmentation

Janhavi Khindkar

Semantic Segmentation

1. In contrast to Image classification which assigns single class to input image... Segmentation assigns classes pixel wise.
2. Semantic Segmentation assigns same class to near by objects without assigning boundaries whereas instance segmentation assigns different classes to nearby same object
3. Common datasets: MSCOCO and Pascal VOC

Semantic Segmentation



Classification + Localization



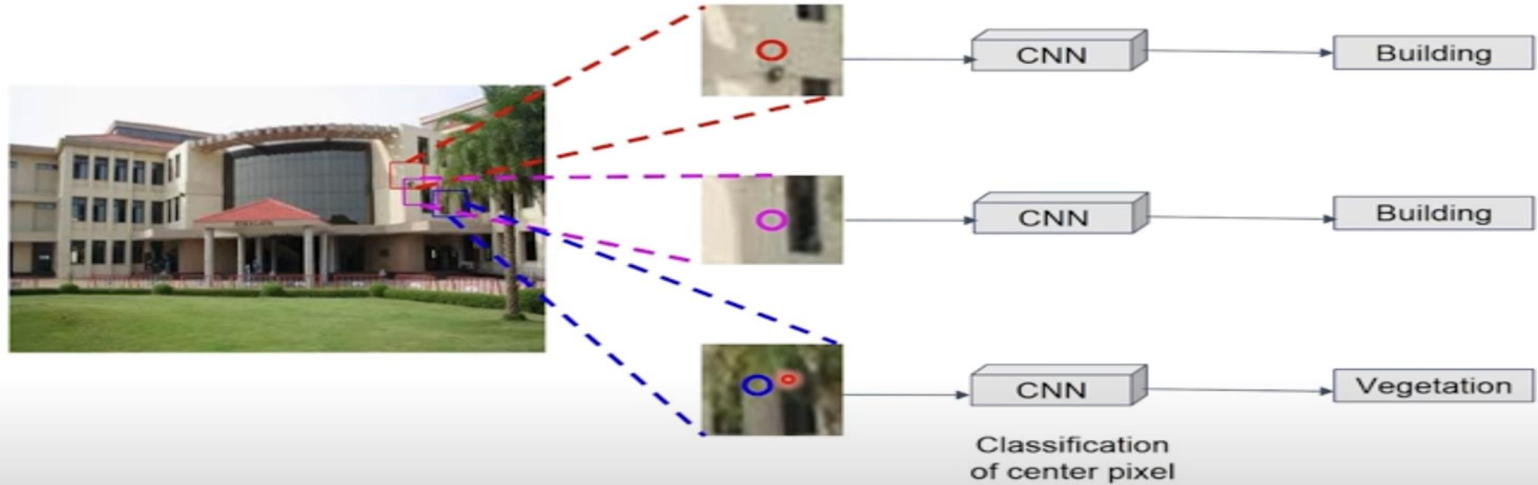
Object Detection



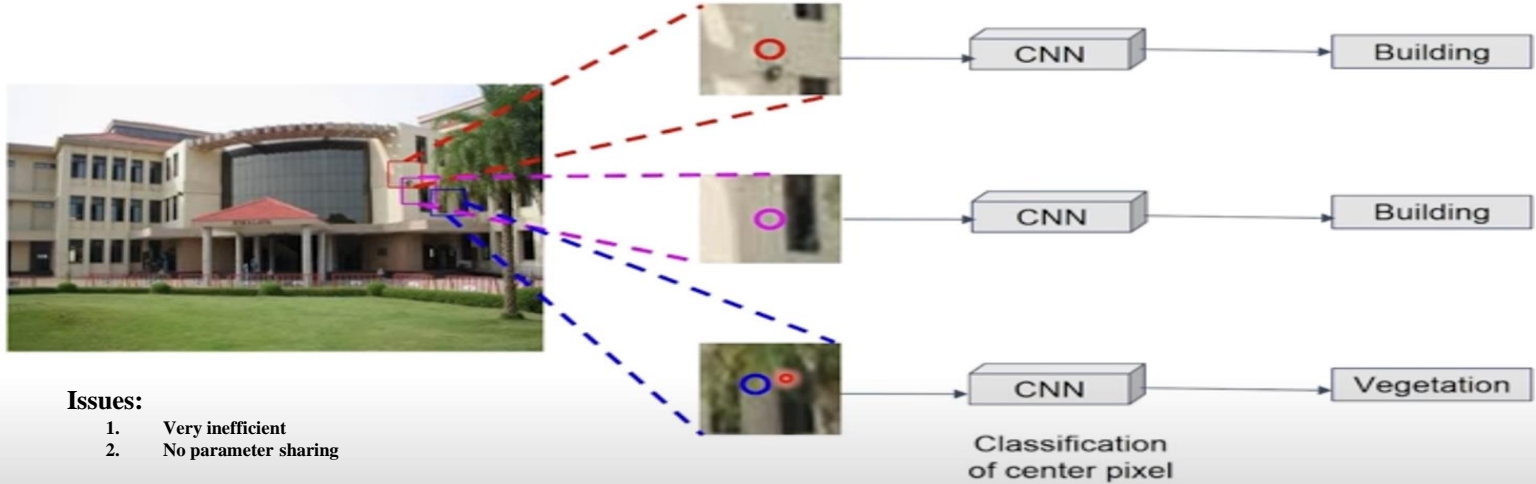
Instance Segmentation



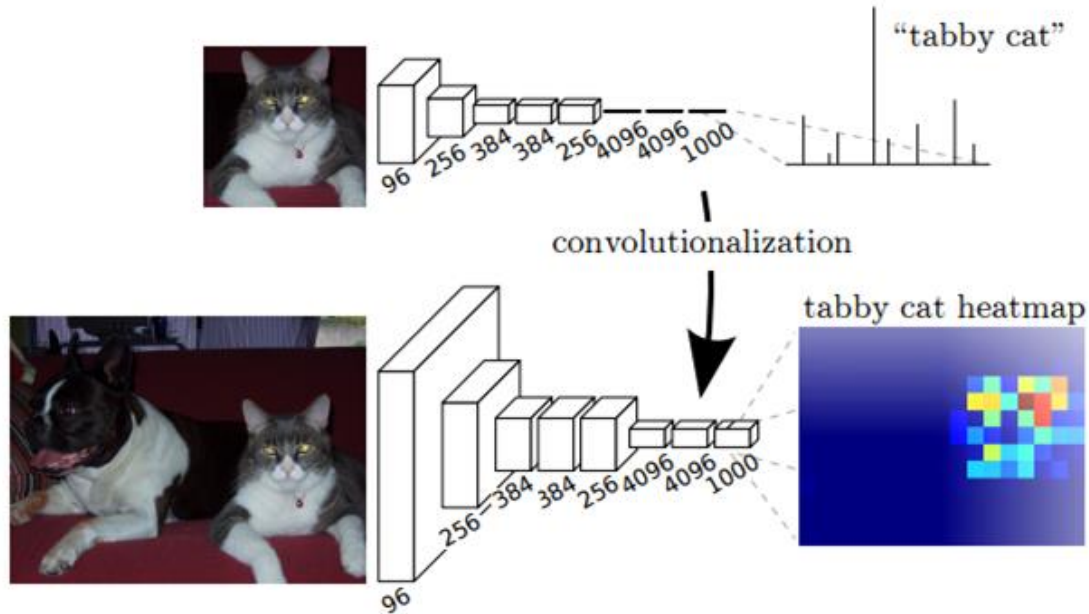
Overlapping patches (Naive method)



Overlapping patches Issues



Fully convolutional Network

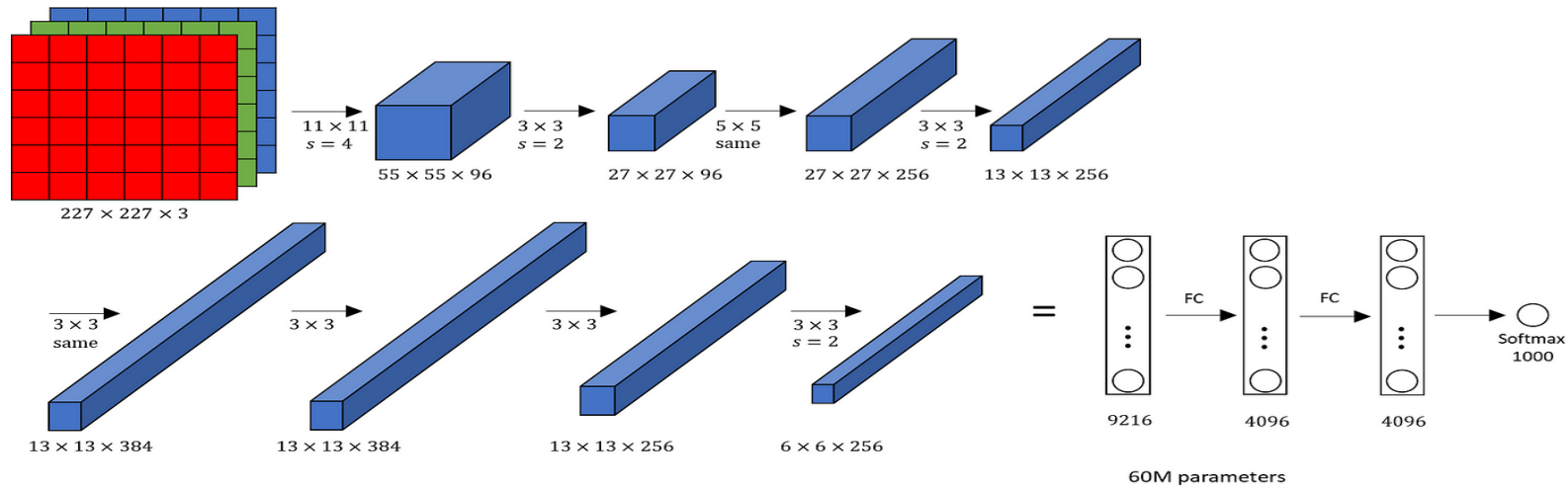


1. Use the similar architecture of that of Image classification models, just replacing the dense layers by 1×1 conv layers

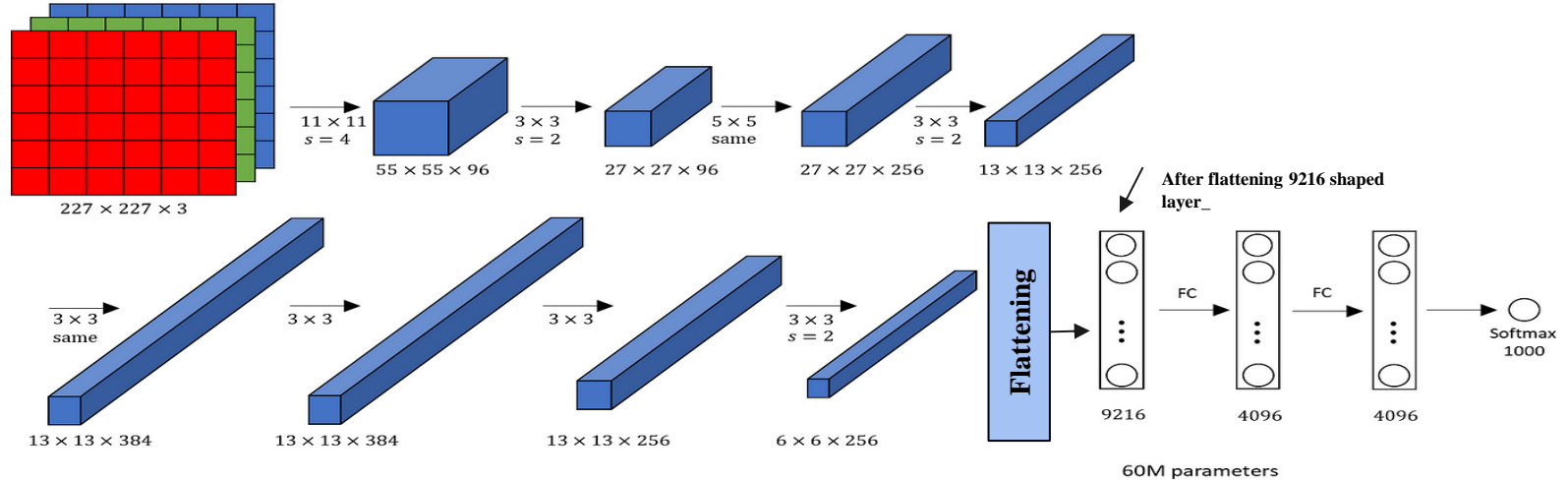
2. The advantage of using conv layers is in dense layers we have to compulsory match the input and have all inputs of same size,

3. But in conv layers that is not a compulsion

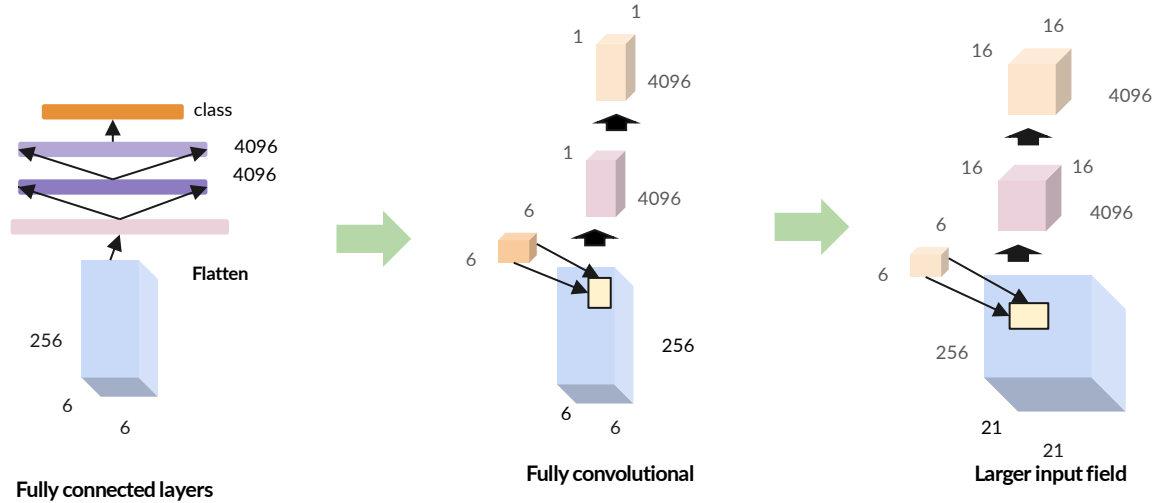
Disadvantages of Dense layers



Disadvantages of Dense layers

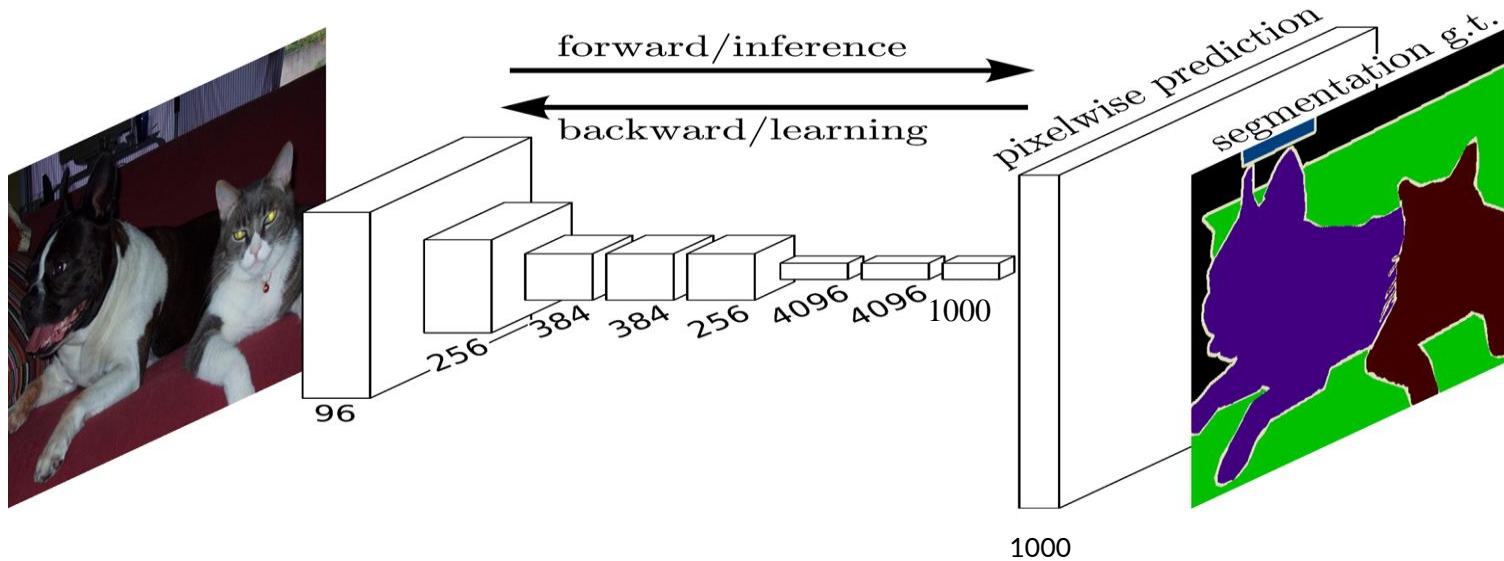


Fully connected to Fully convolutional

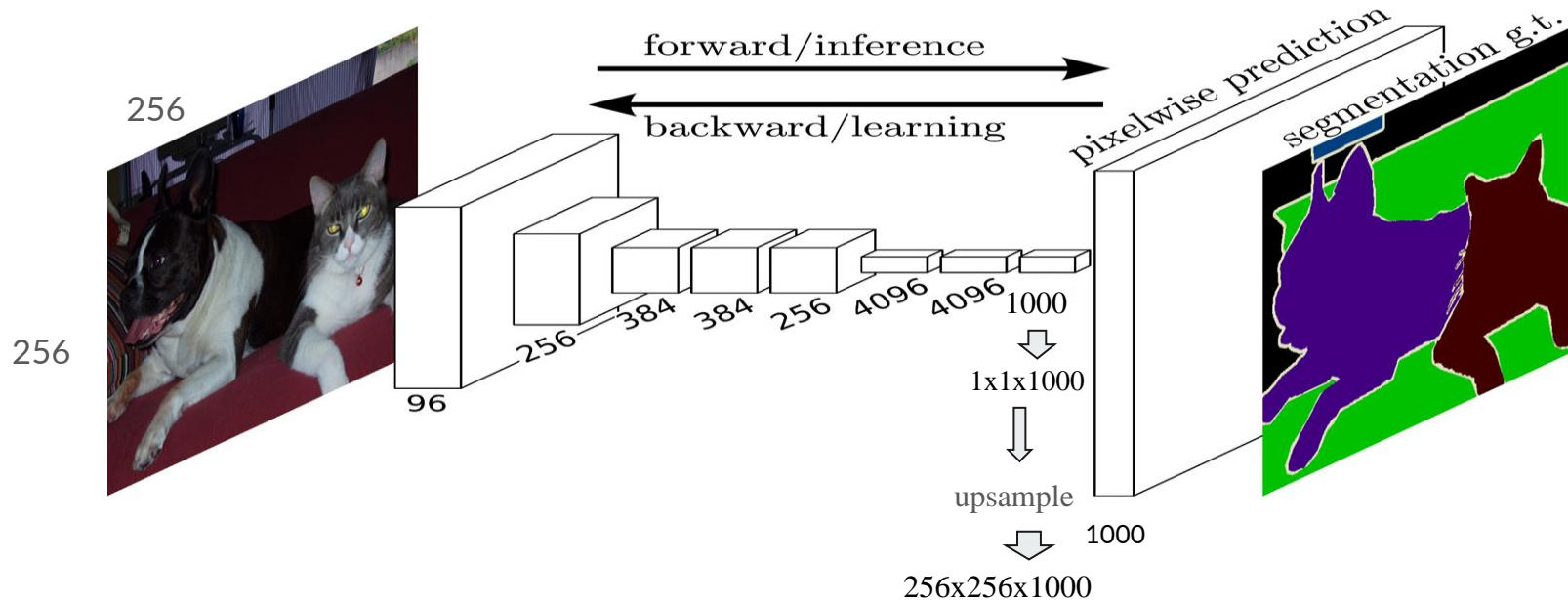


Output size = $(F - K + 2P) / S + 1$
F= 6 K=6 P=0 S=1
Output = $(6-6+0)/1+1=1$

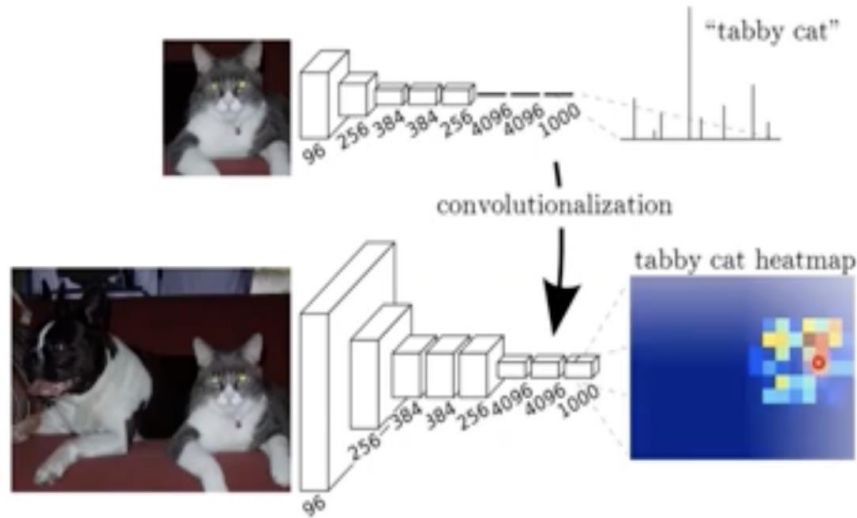
FCN continued



FCN continued



Changes from classification to segmentation



1. From classification 1000 classes of fully connected network we went to 1000 filter maps
2. These final maps can we considered heatmaps of each pixel being that particular class
3. But since final heatmap is smaller than original dimension we would need to upscale the final resolutions



Steps to be taken for segmentation

1. Use pretrained networks of classification for feature extraction for segmentation
2. Replace the fully connected layers with fully convolutional layers
3. Upsample the final activations we get to match the original input image to get proper segmentation masks
4. Use skip connection from earlier layers to improve segmentation accuracy

**How to achieve Upsampling of
filter maps???**

Using Nearest Neighbour

8	6
6	9

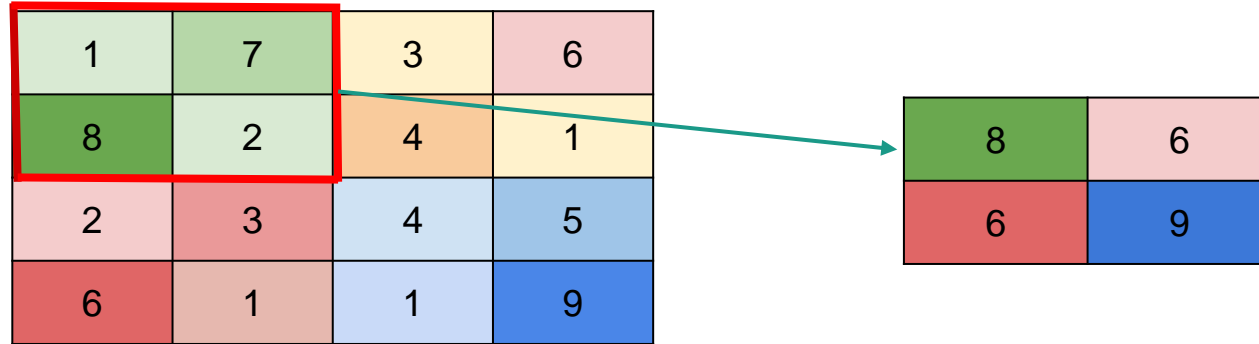


8	8	6	6
8	8	6	6
6	6	9	9
6	6	9	9

(stride 2, kernel size 2)

Solving upsampling

Lets get recap of downsampling using max pooling, by using stride of 2 and 2x2 as the kernel size



Solving upsampling

Lets get recap of downsampling using max pooling, by using stride of 2 and 2x2 as the kernel size

1	7	3	6
8	2	4	1
2	3	4	5
6	1	1	9

Max pooling

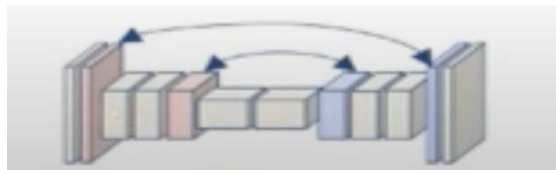


(1,0)	(0,3)
(3,0)	(3,3)

8	6
6	9

Store index of highest value
to use at the time of
upsampling

Using opp of max pooling



1	7	3	6
8	2	4	1
2	3	4	5
6	1	1	9



8	6
6	9



Intermediate
layers



1	7
9	5

(1,0)	(0,3)
(3,0)	(3,3)

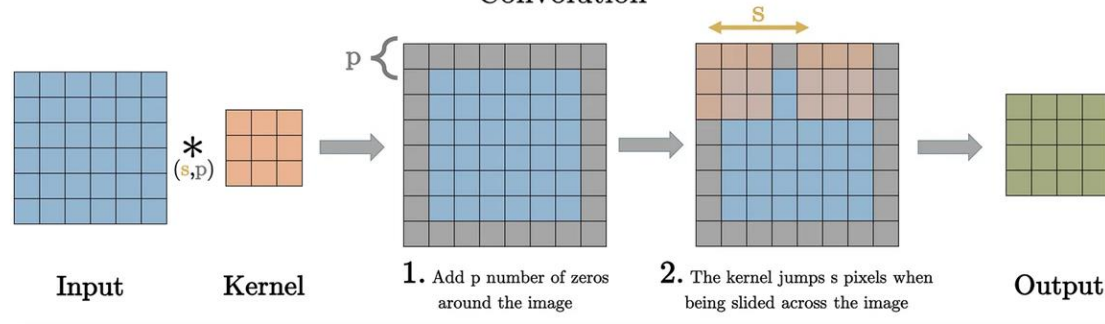
0	0	0	7
1	0	0	0
0	0	0	0
9	0	0	5

Learnable Upsampling - Transpose Convolution

Standard Convolutional Layer: A standard convolutional layer on an input of size $i \times i$ is defined by the following two parameters.

Padding (p): The number of zeros padded around the original input increasing the size to $(i+2*p) \times (i+2*p)$

Stride (s): The amount by which the kernel is shifted when sliding across the input image. The figure below shows





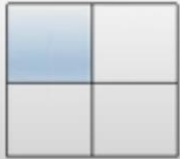
Transpose convolution

1. A transposed convolutional layer, on the other hand, is usually carried out for upsampling i.e. to generate an output feature map that has a spatial dimension greater than that of the input feature map.
2. Just like the standard convolutional layer, the transposed convolutional layer is also defined by the padding and stride.

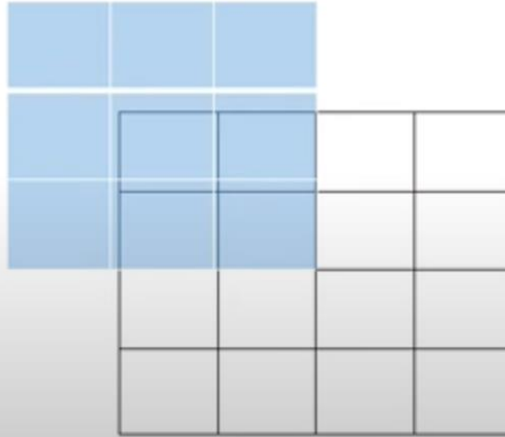
Transpose convolution

3x3 convolution, stride of 2 and a pad of 1

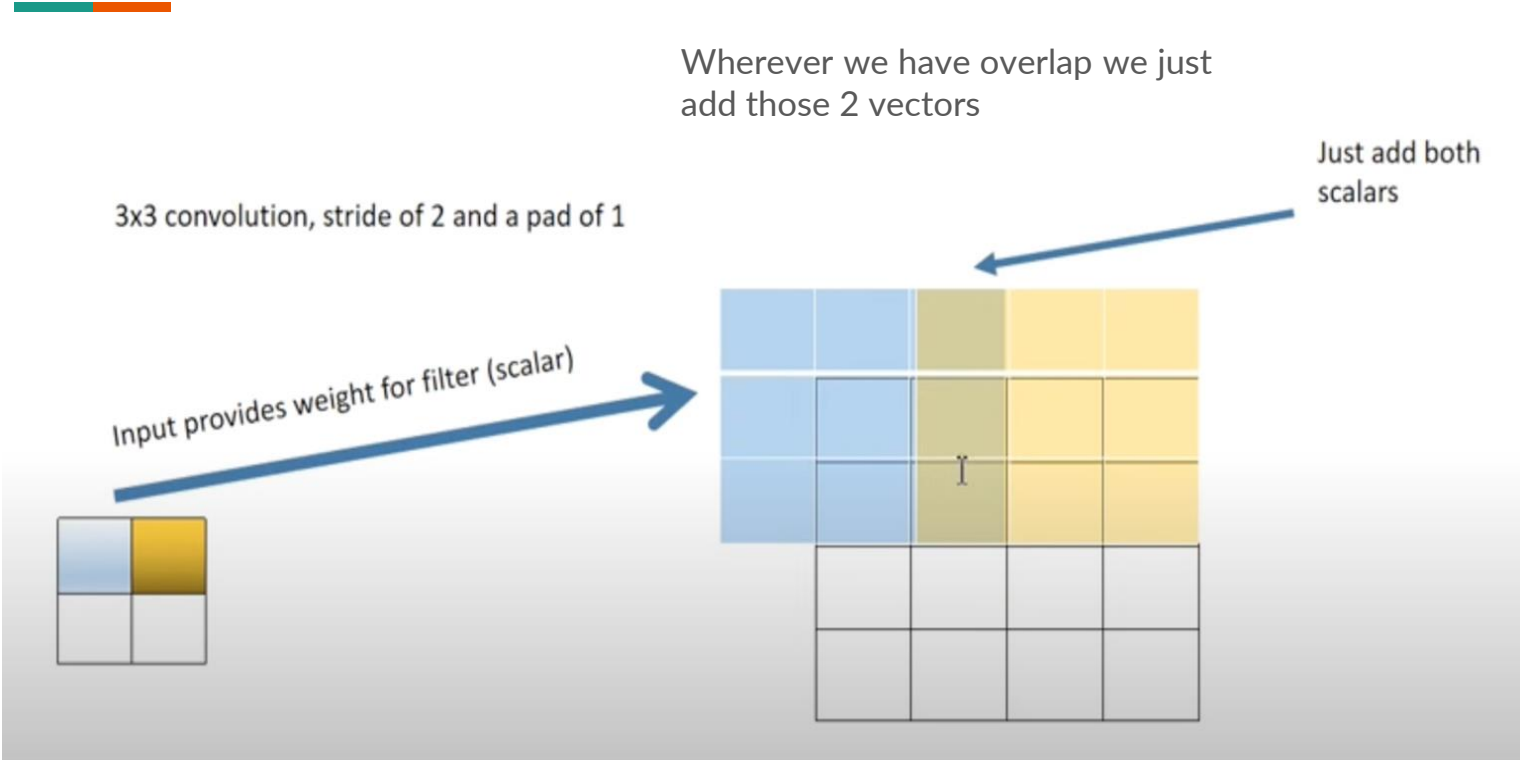
Input provides weight for filter (scalar)



Trying to learn this filter by using the input cell as weight



Continued...



Transposed Convolutions

2x2 convolution, stride of 1 and a pad of 0

$$4 * 3 = 12$$

$$2 * 1 = 2$$

$$12 + 2 = 14$$

6	14	4
2	17	21
0	1	5

Output

2	4
0	1

Input

3	1
1	5

Conv
Kernel

6	2	
2	10	

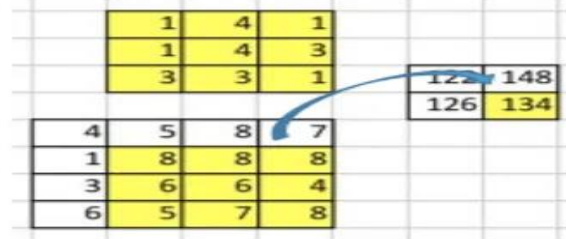
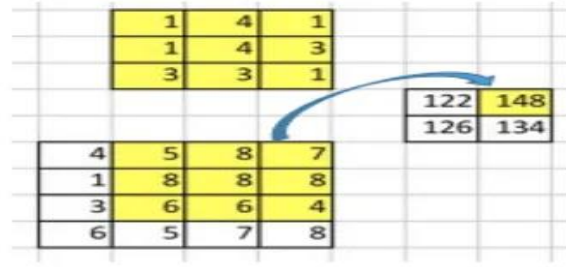
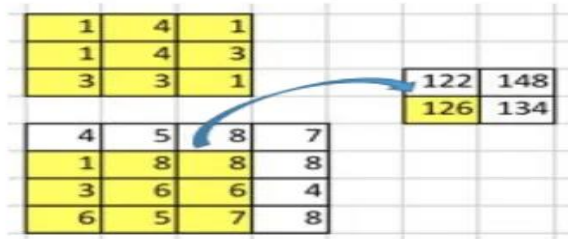
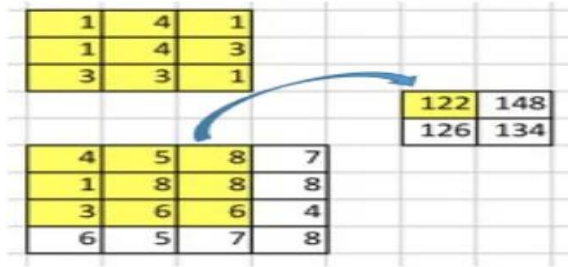
	12	4
	4	20

0	0	
0	0	

	3	1
	1	5

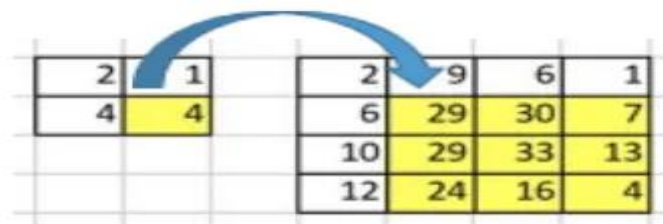
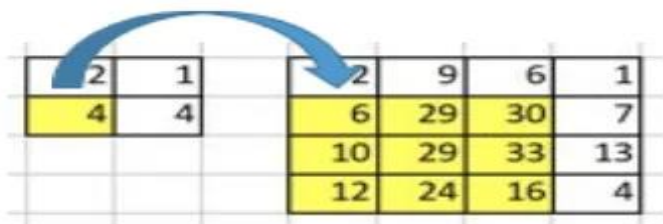
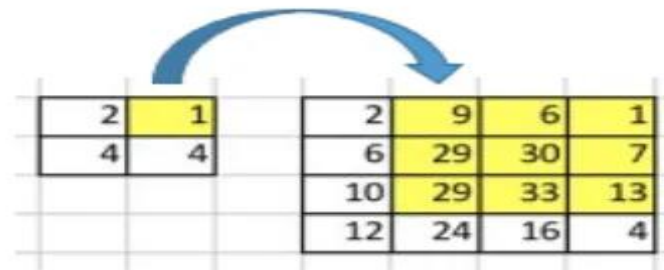
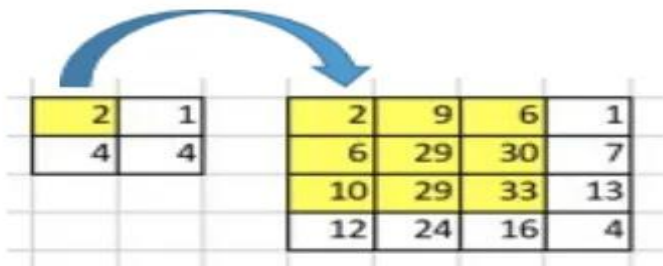
Transpose convolutions intuition

Transpose convolution



1. Consider this convolution operation on 4x4 input with 3x3 filter and stride of 1 and no padding
2. We go from 4x4 filter go 2 x2 filter
3. 3x3 filter connects 9 values in input to 1 value in output
4. Now suppose we want to do opposite i.e 1 value in input is connected to 9 places in output

Transpose convolution



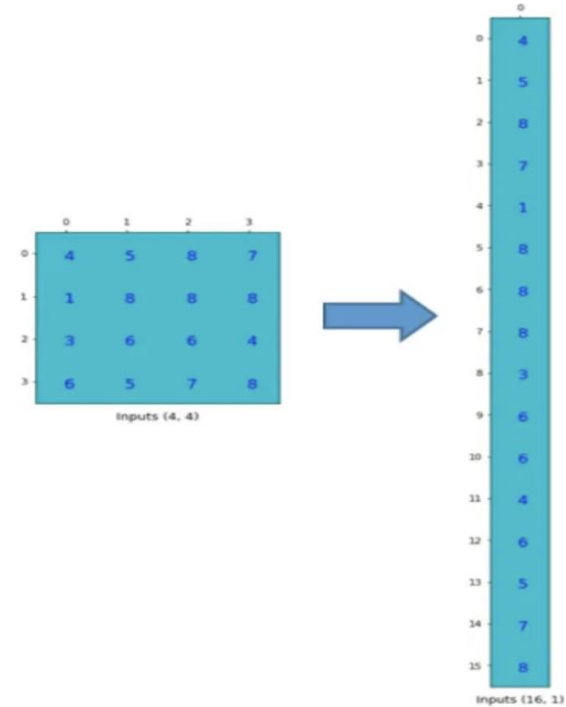


Matrix representation of convolution

1. We can express a convolution operation using a matrix. It is nothing but a kernel matrix rearranged so that we can use matrix multiplication to conduct convolution operations.
2. First we transform the kernel in form of matrix and then the original matrix is transformed as well

Matrix representation of convolution

1. We can express a convolution operation using a matrix.
2. First we flatten the input matrix to get row vector
3. Then we transform the kernel in form of matrix
4. Matrix multiply the flatten input vector with kernel matrix



Flattening the input matrix (4,4) to (16,1)

Understanding convolution by matrix multiplication

	0	1	2
0	1	4	1
1	1	4	3
2	3	3	1

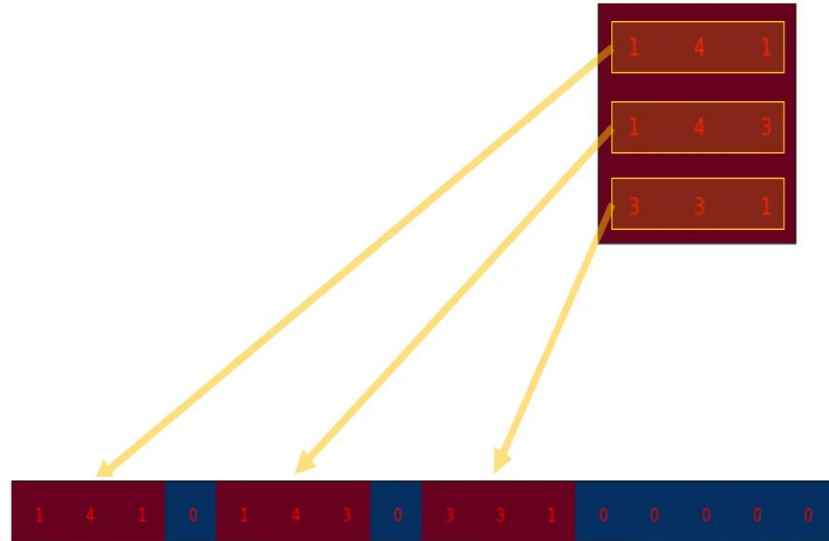
Kernel (3, 3)



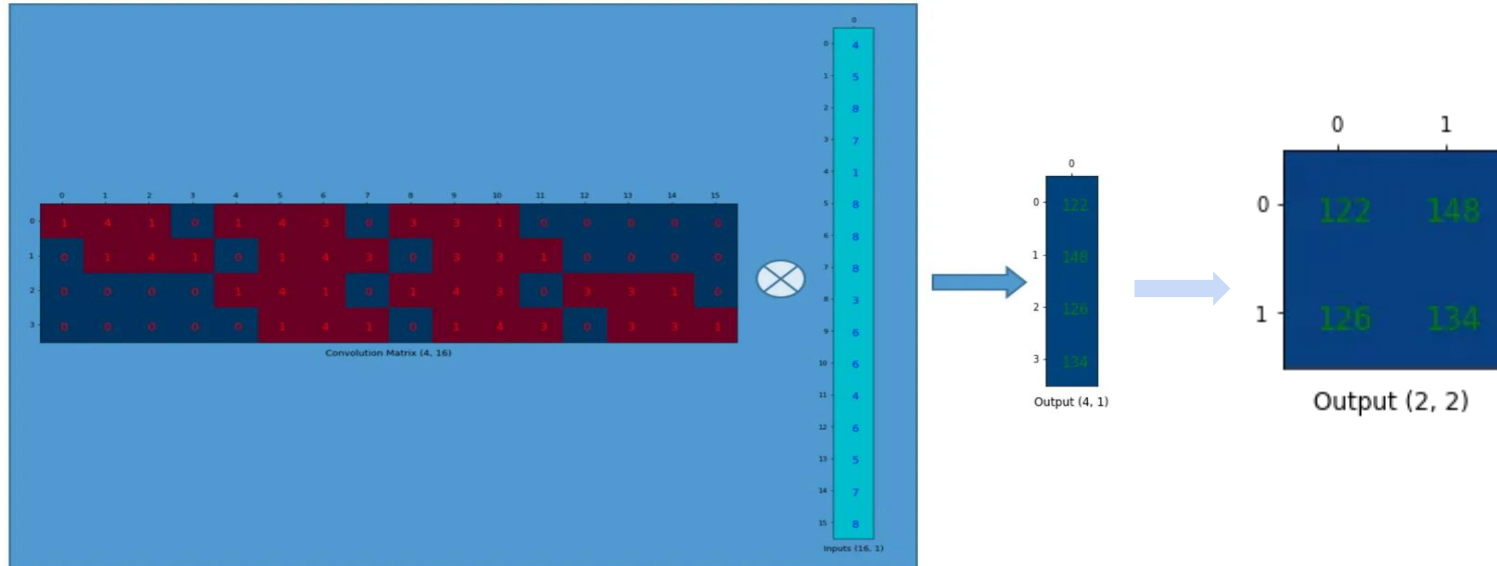
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	4	1	0	1	4	3	0	3	3	1	0	0	0	0	0
1	0	1	4	1	0	1	4	3	0	3	3	1	0	0	0	0
2	0	0	0	0	1	4	1	0	1	4	3	0	3	3	1	0
3	0	0	0	0	0	1	4	1	0	1	4	3	0	3	3	1

Convolution Matrix (4, 16)

This is the convolution matrix. Each row defines one convolution operation. If you do not see it, the below diagram may help. Each row of the convolution matrix is just a rearranged kernel matrix with zero padding in different places.



Multiply them to get output





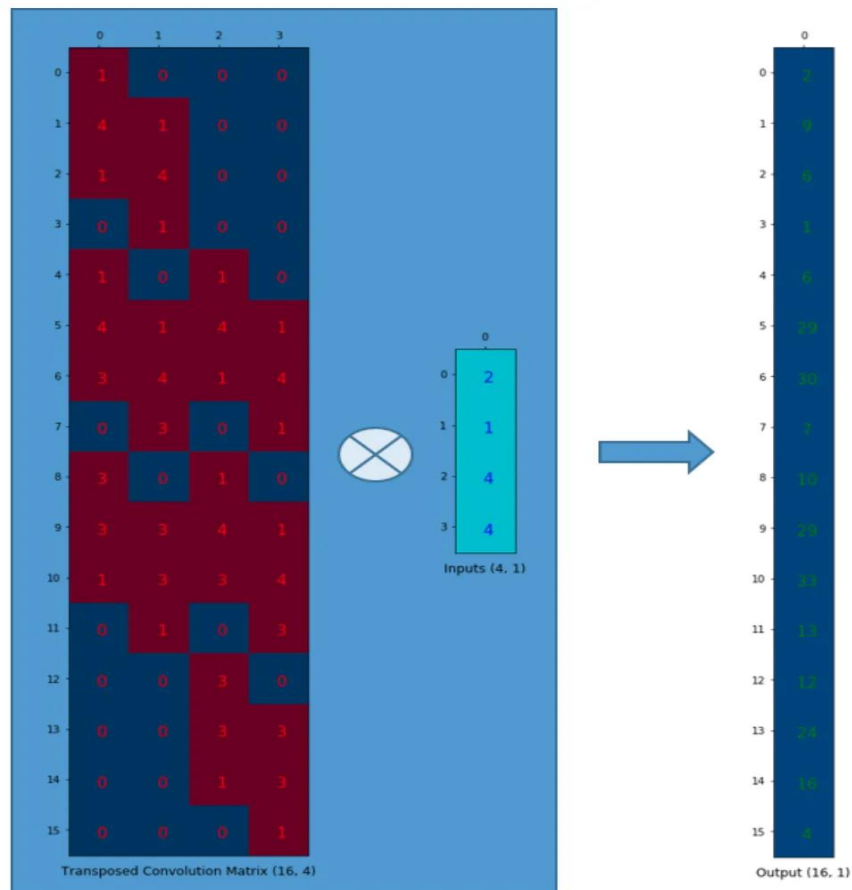
Intuition

In short, a convolution matrix is nothing but rearranged kernel weights, and a convolution operation can be expressed using the convolution matrix. So what? The point is that with the convolution matrix, you can go from $16 (4 \times 4)$ to $4 (2 \times 2)$ because the convolution matrix is 4×16 . Then, if you have a 16×4 matrix, you can go from $4 (2 \times 2)$ to $16 (4 \times 4)$.

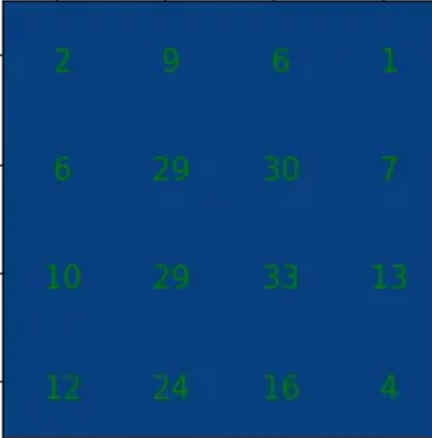
Transpose convolution

We want to go from 4 (2x2) to 16 (4x4). So, we use a 16x4 matrix. But there is one more thing here. We want to maintain the 1 to 9 relationship.

Suppose we transpose the convolution matrix C (4x16) to C.T (16x4). We can matrix-multiply C.T (16x4) with a column vector (4x1) to generate an output matrix (16x1). The transposed matrix connects 1 value to 9 values in the output.



Can be transformed to get 4x4 matrix



	0	1	2	3
0	2	9	6	1
1	6	29	30	7
2	10	29	33	13
3	12	24	16	4

Output (4, 4)

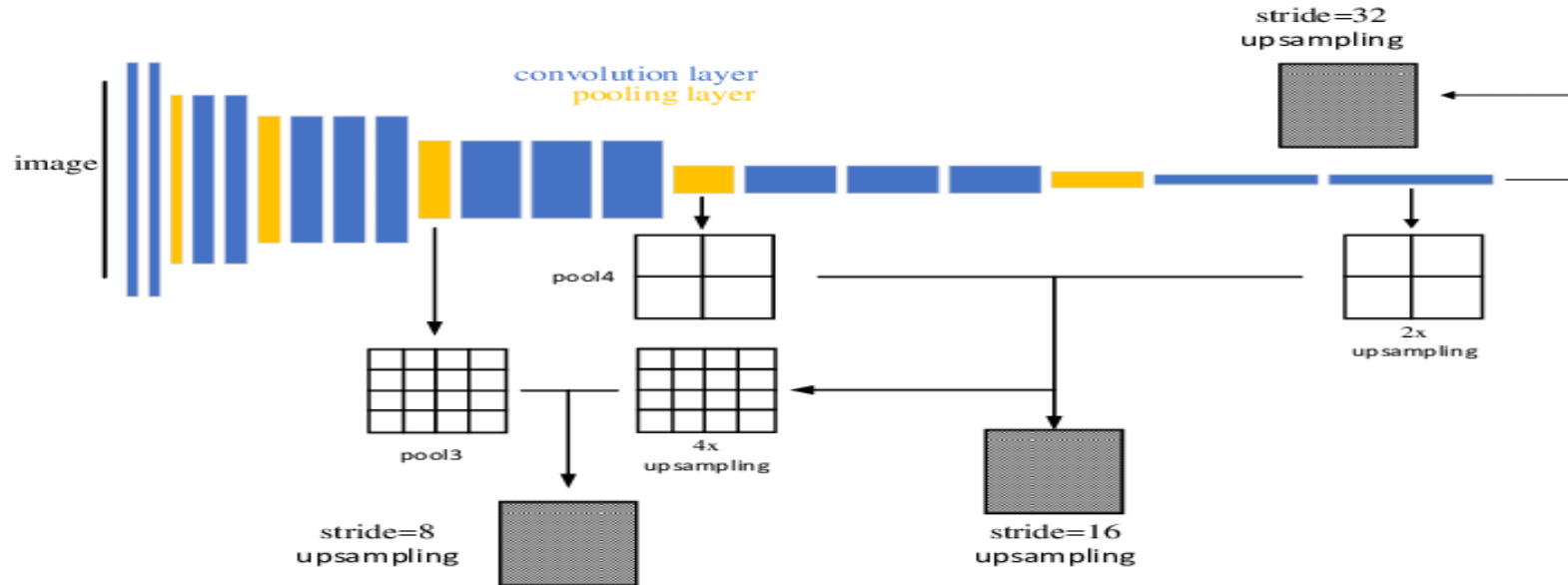


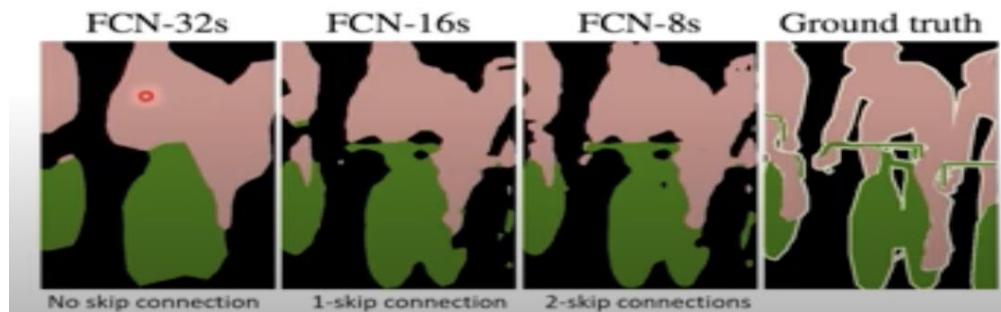
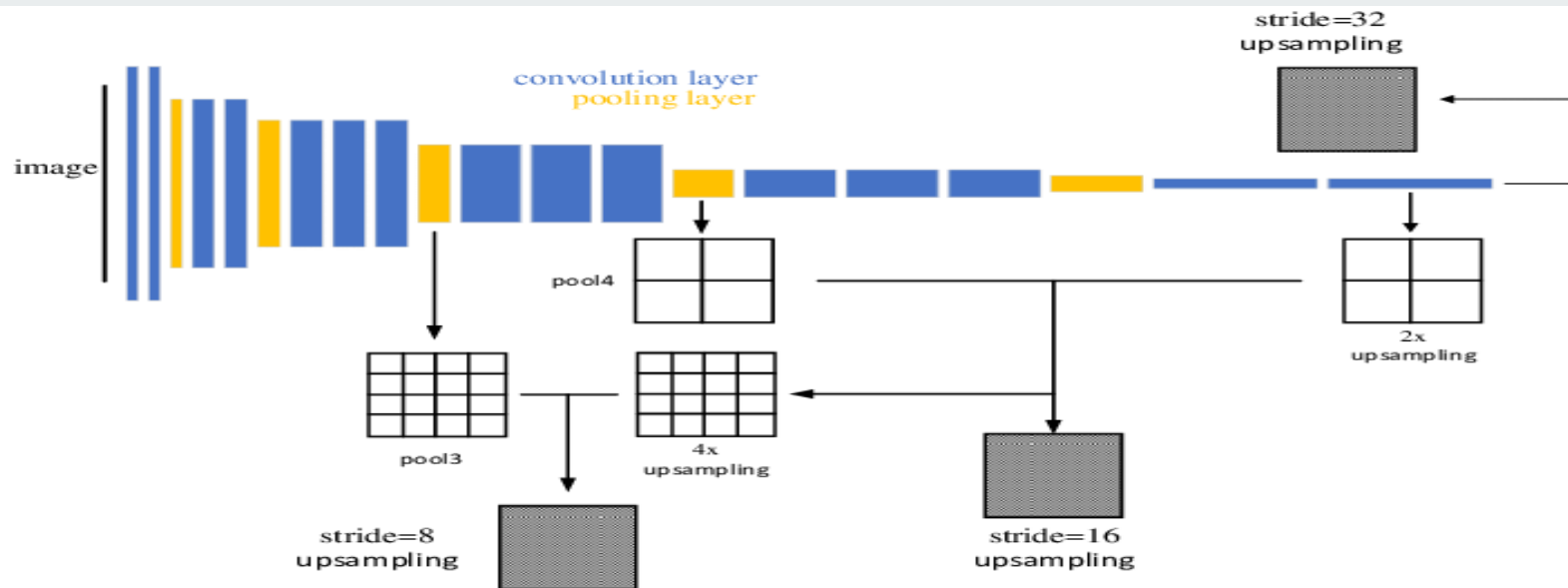
Steps to be taken for segmentation

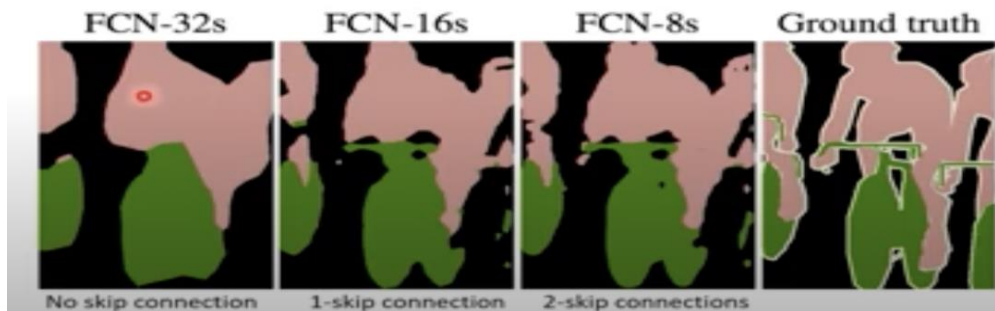
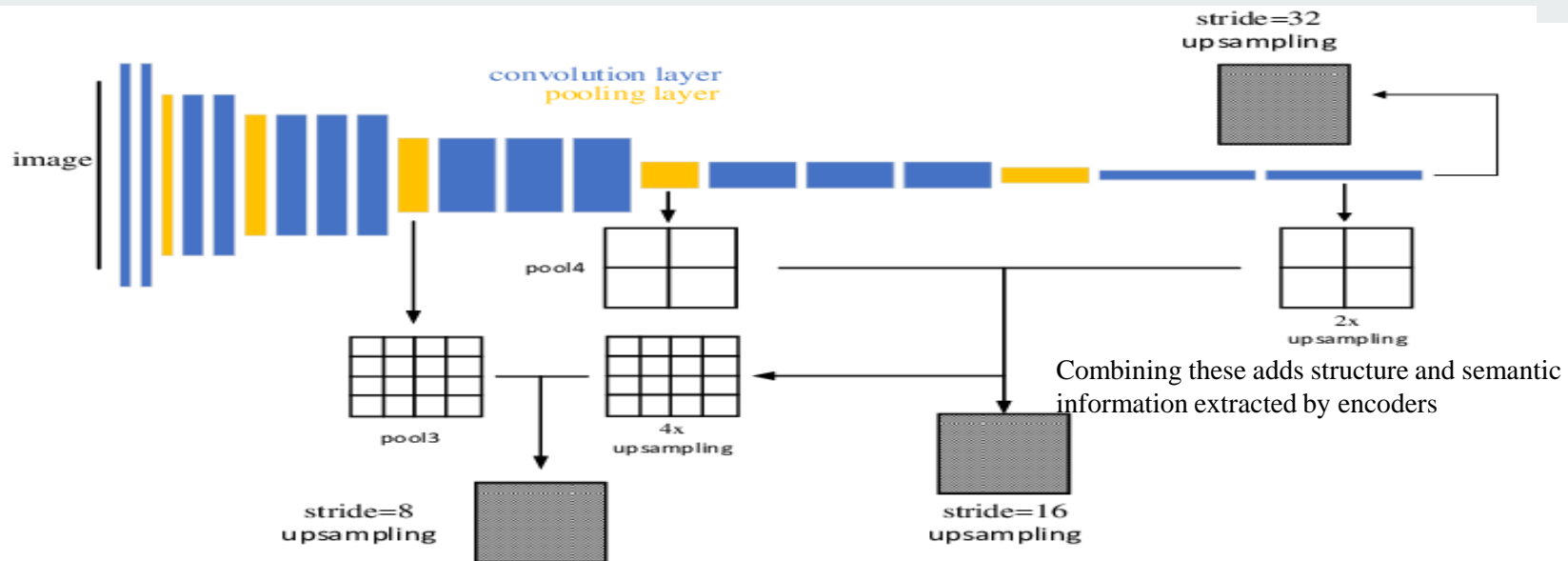
1. Use pretrained networks of classification for feature extraction for segmentation
2. Replace the fully connected layers with fully convolutional layers
3. Upsample the final activations we get to match the original input image to get proper segmentation masks
4. Use skip connection from earlier layers to improve segmentation accuracy

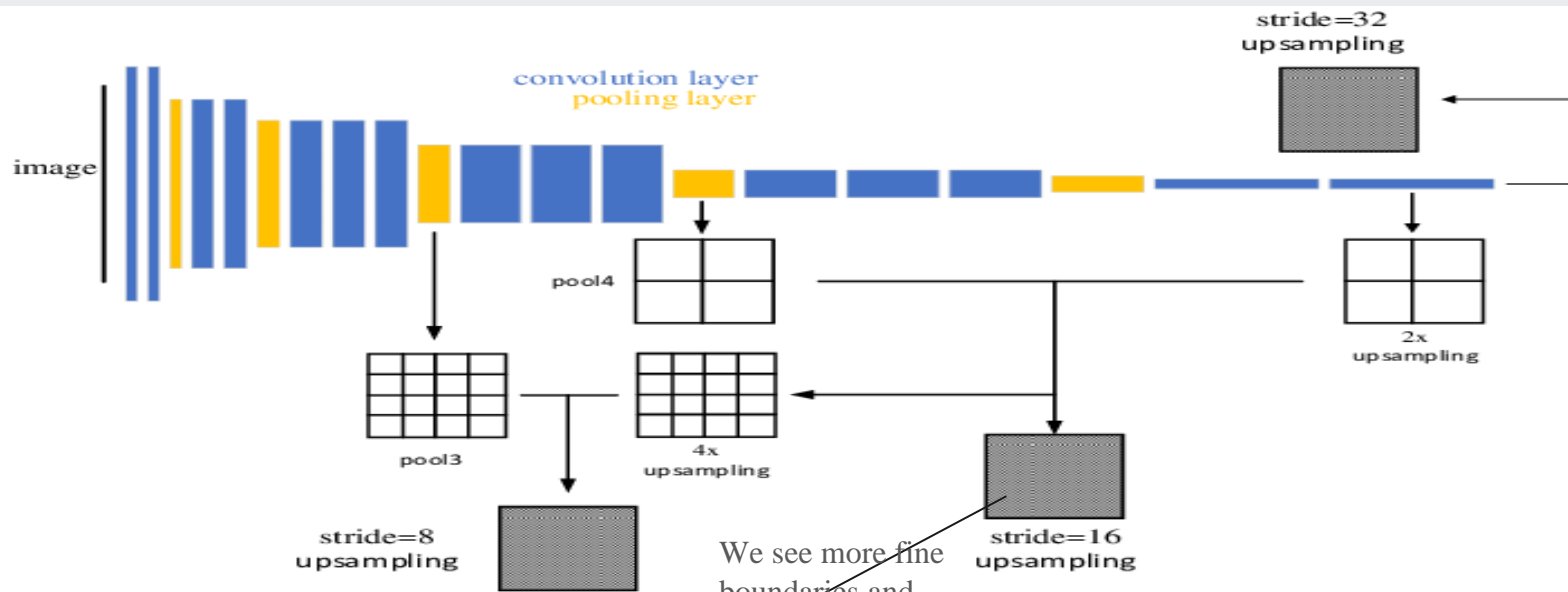
**How to use skip connections with
upsampling???**

Look at FCN network

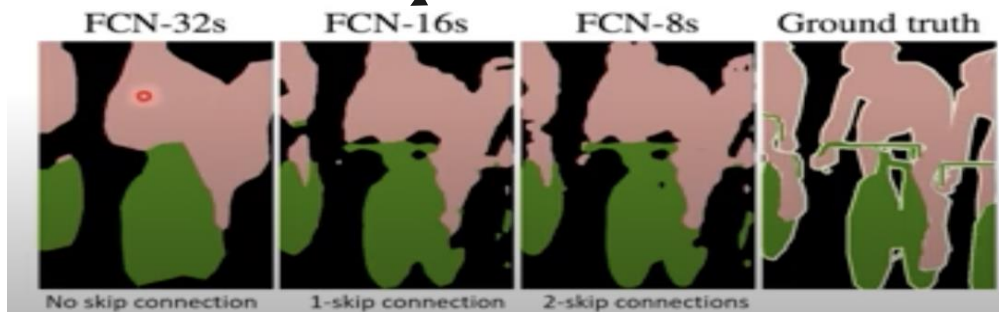








We see more fine boundaries and structures

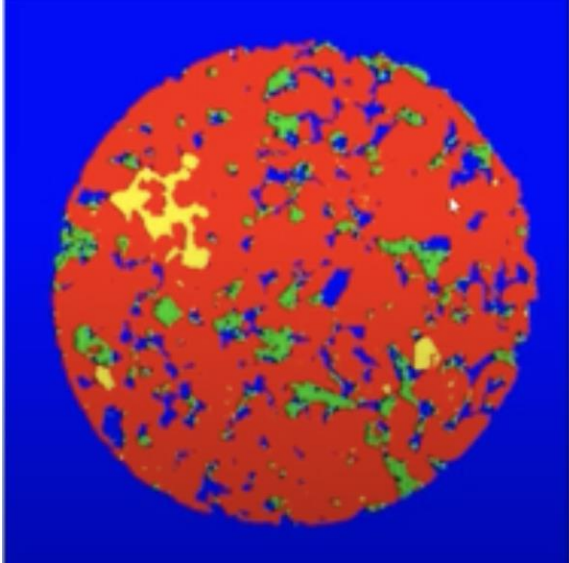




Training and Loss in segmentation

1. Images are sent through and we generate the segmentation masks which are softmax score per pixel per class
2. We calculate sparse categorical cross entropy between generated masks and the original mask pixel wise and average the loss across pixels to get final loss.
3. In our case the predicted mask is $(128, 128, 3)$ and original is $(128, 128, 1)$
4. The parameters are updated by using backpropagation and same is continued till the loss is minimised

Dice score and IOU

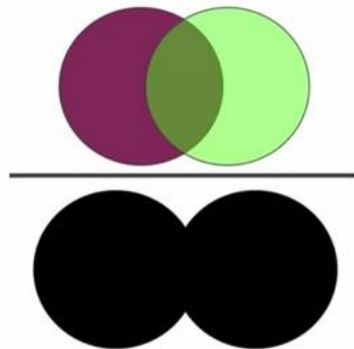


Why accuracy can't be a good measure for segmentation tasks :

1. Accuracy is inefficient when having class imbalance
2. Inaccuracy of minority classes is overshadowed by accuracies of majority classes
3. Here if u see red and blue are well represented and green and yellow are not

IOU

$$\text{IOU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



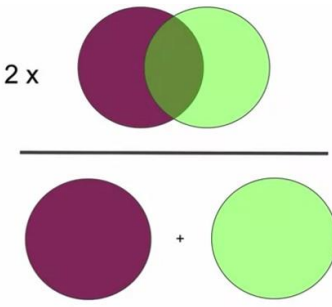
1. IOU in keras is calculated using

Keras' definition:

$$\text{IOU} = \text{true_positive} / (\text{true_positive} + \text{false_positive} + \text{false_negative})$$

1. We average the IOU across classes but it is also good to check individual IOU to check how the performance is class wise

Dice score

$$\text{Dice Score} = 2 \times \frac{\text{Area of Overlap}}{\text{Combined Area}}$$


1. It tries to calculate area of overlap and divides it by total area

```
loss = 1 - (2 * sum(y_true * y_pred)) / (sum(y_true) + sum(y_pred))
```

1. In keras it is calculated using above formula as loss but score will be 1-loss



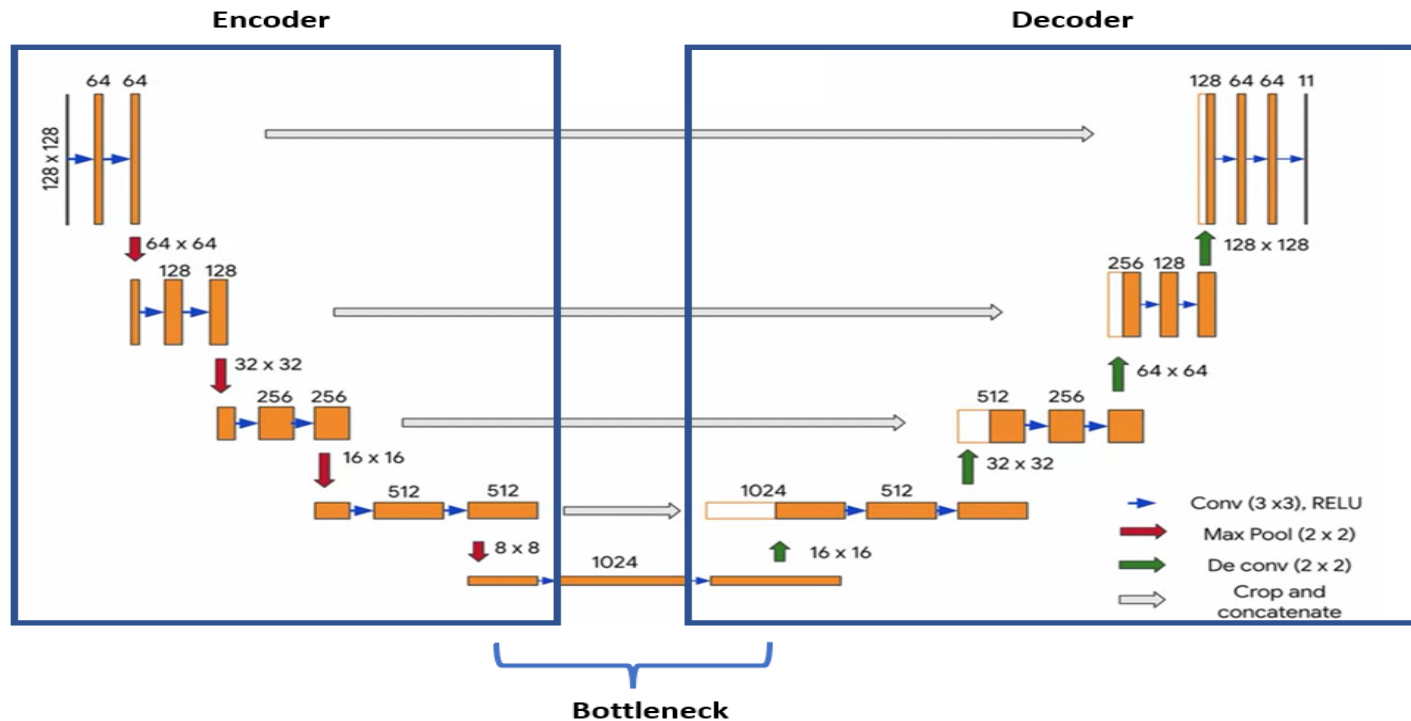
UNet theory

UNET is an architecture developed by Olaf Ronneberger et al. for Biomedical [Image Segmentation](#) in 2015 at the University of Freiburg, Germany.

It is one of the most popularly used approaches in any semantic segmentation task today.

It is a fully convolutional neural network that is designed to learn from fewer training samples

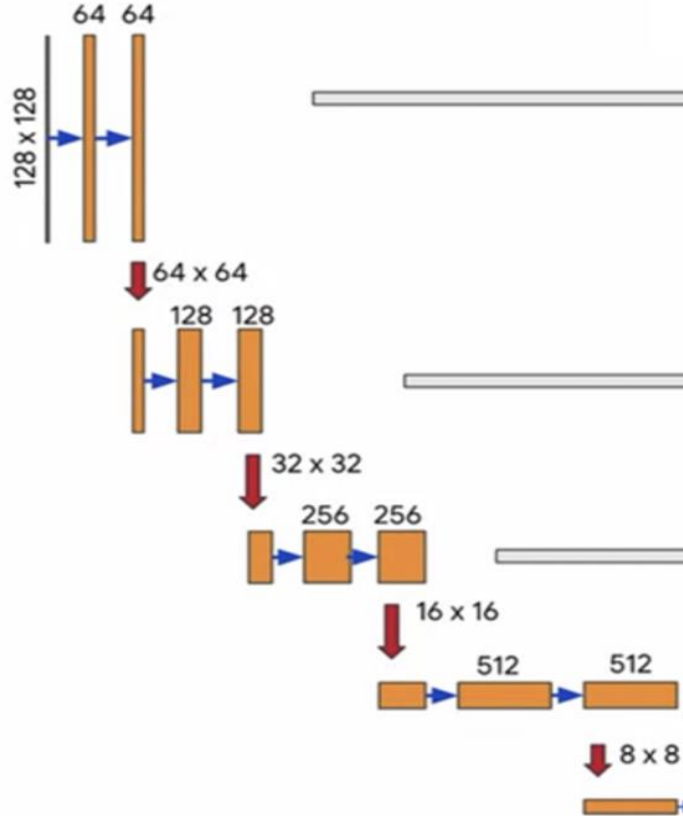
Unet Architecture





Unet continued...

1. UNET is a U-shaped encoder-decoder network architecture, which consists of four encoder blocks and four decoder blocks that are connected via a bridge.
2. The encoder network (contracting path) half the spatial dimensions and double the number of filters (feature channels) at each encoder block.
3. Likewise, the decoder network doubles the spatial dimensions and half the number of feature channels.



Encoder Block

1. The encoder network acts as the feature extractor and learns an abstract representation of the input image through a sequence of the encoder blocks.
2. Each encoder block consists of two 3x3 convolutions, where each convolution is followed by a ReLU (Rectified Linear Unit) activation function.
3. The output of the ReLU acts as a skip connection for the corresponding decoder block.
4. Next, follows a 2x2 max-pooling, where the spatial dimensions (height and width) of the feature maps are reduced by half. This reduces the computational cost by decreasing the number of trainable parameters.



Skip connections

Skip Connections (or *Shortcut Connections*) as the name suggests *skips some of the layers in the neural network and feeds the output of one layer as the input to the next layers*

Skip Connections were introduced to solve different problems in different architectures. In the case of ResNets, skip connections solved the *degradation problem* that we addressed earlier whereas, in the case of Unet, it ensured feature reusability

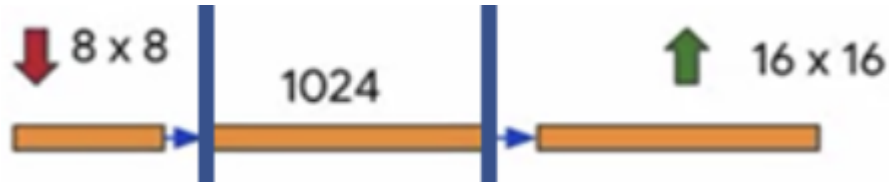
Skip connections in Unet

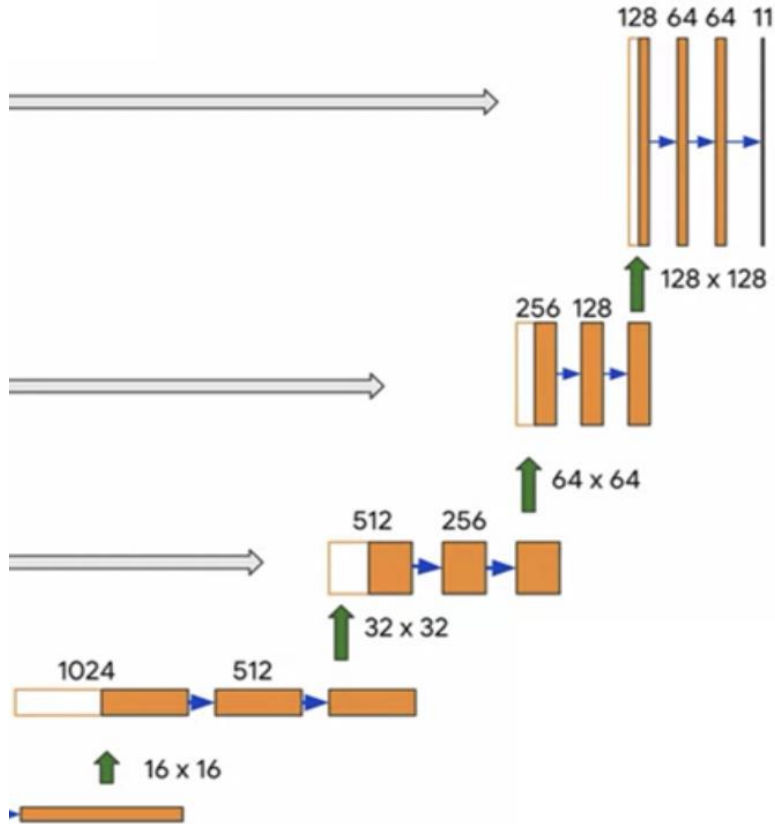
1. Skip connections are connections made between earlier relu layers from encoder side to later upsampled layers in decoder side.
2. As we go through a lot of layers some of the feature information is lost and mostly the class level information is present for the decoder side
3. Thus skip connections provide additional information that helps the decoder to generate better semantic features by concatenating the semantic level features with the previous layers decoder features
4. This makes the U-Nets use *fine-grained* details learned in the encoder part to construct an image in the decoder part.



Bridge

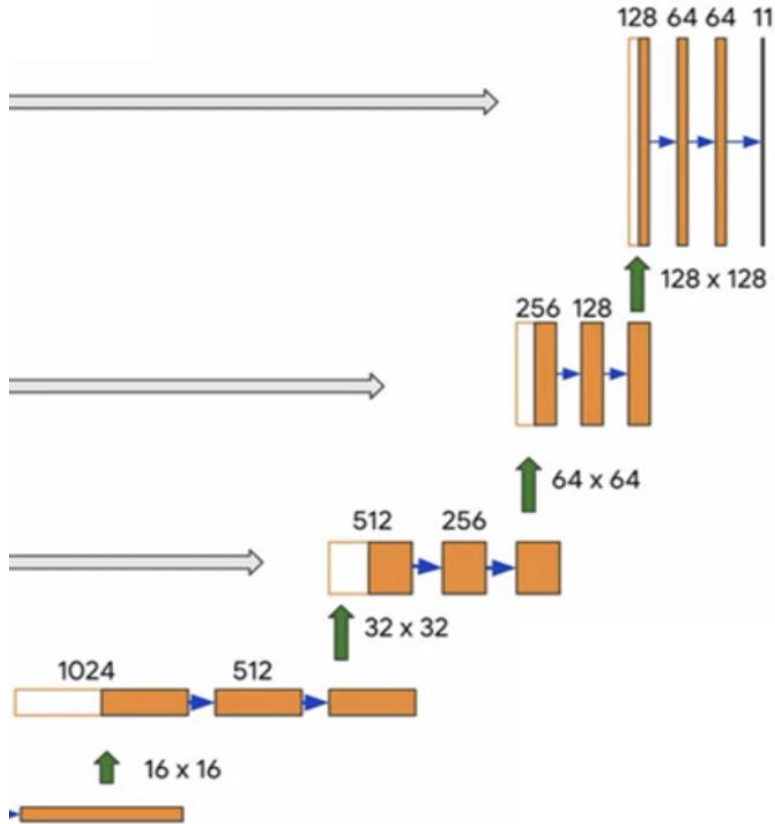
The bridge connects the encoder and the decoder network and completes the flow of information. It consists of two 3×3 convolutions, where each convolution is followed by a ReLU activation function





Decoder

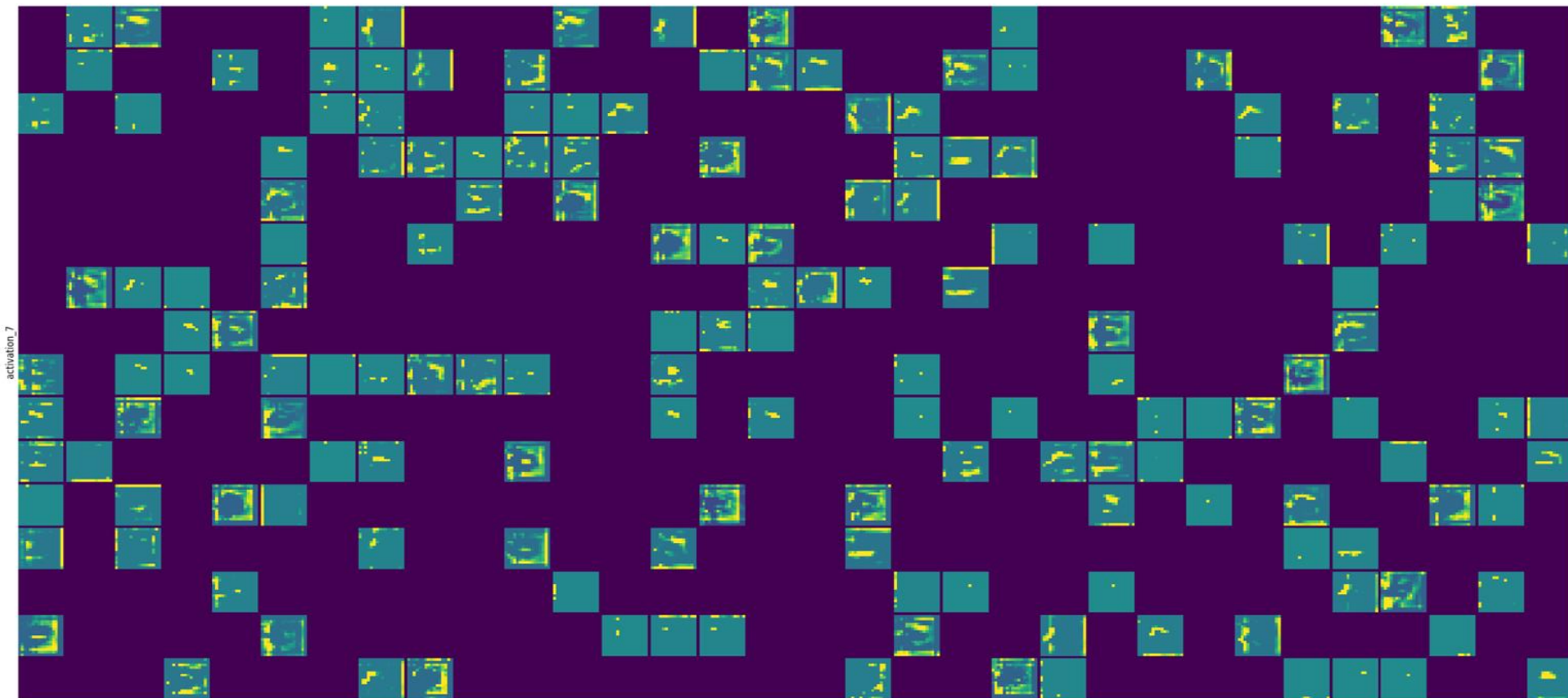
1. The decoder network is used to take the abstract representation and generate a semantic segmentation mask.
2. The decoder block starts with a 2x2 transpose convolution.
3. Next, it is concatenated with the corresponding skip connection feature map from the encoder block



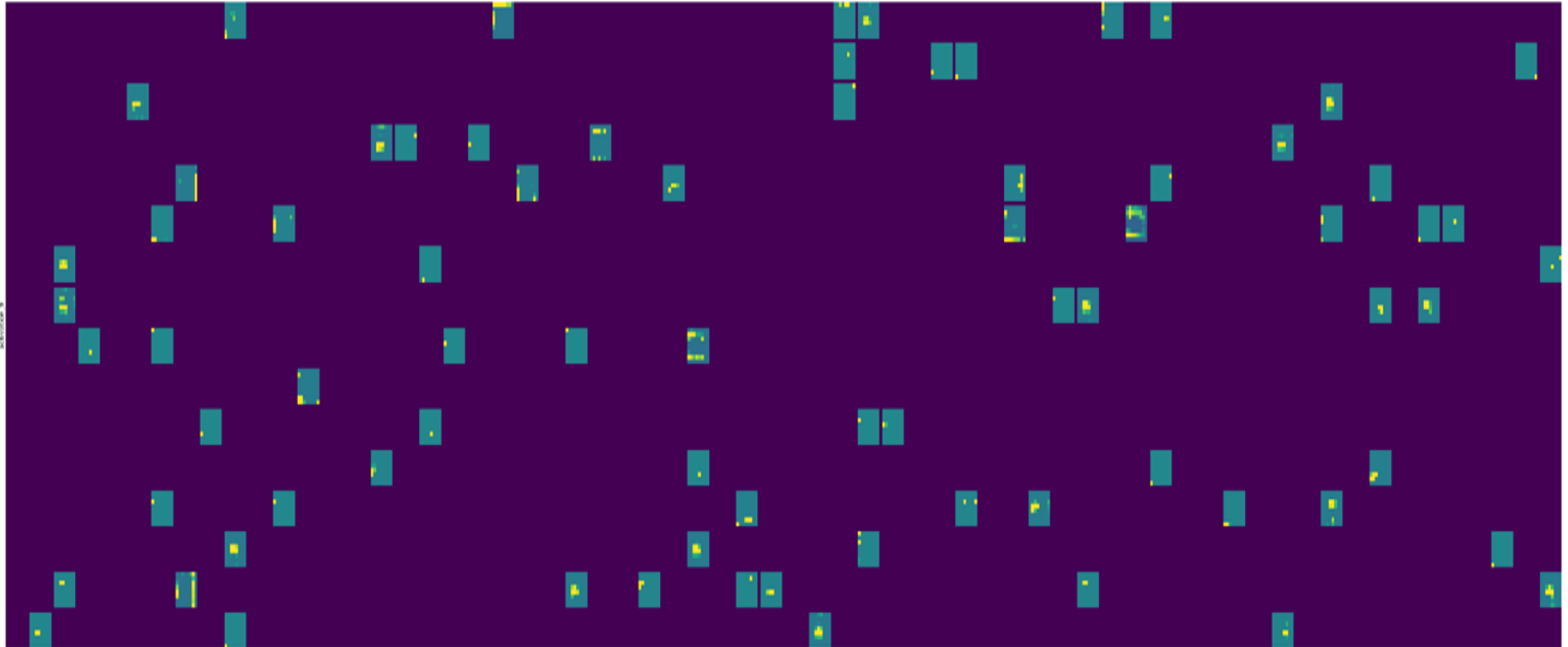
Decoder

4. These skip connections provide features from earlier layers that are sometimes lost due to the depth of the network.
5. After that, two 3x3 convolutions are used, where each convolution is followed by a ReLU activation function
6. The output of the last decoder passes through a 1x1 convolution with sigmoid activation. The sigmoid activation function gives the segmentation mask representing the pixel-wise classification.

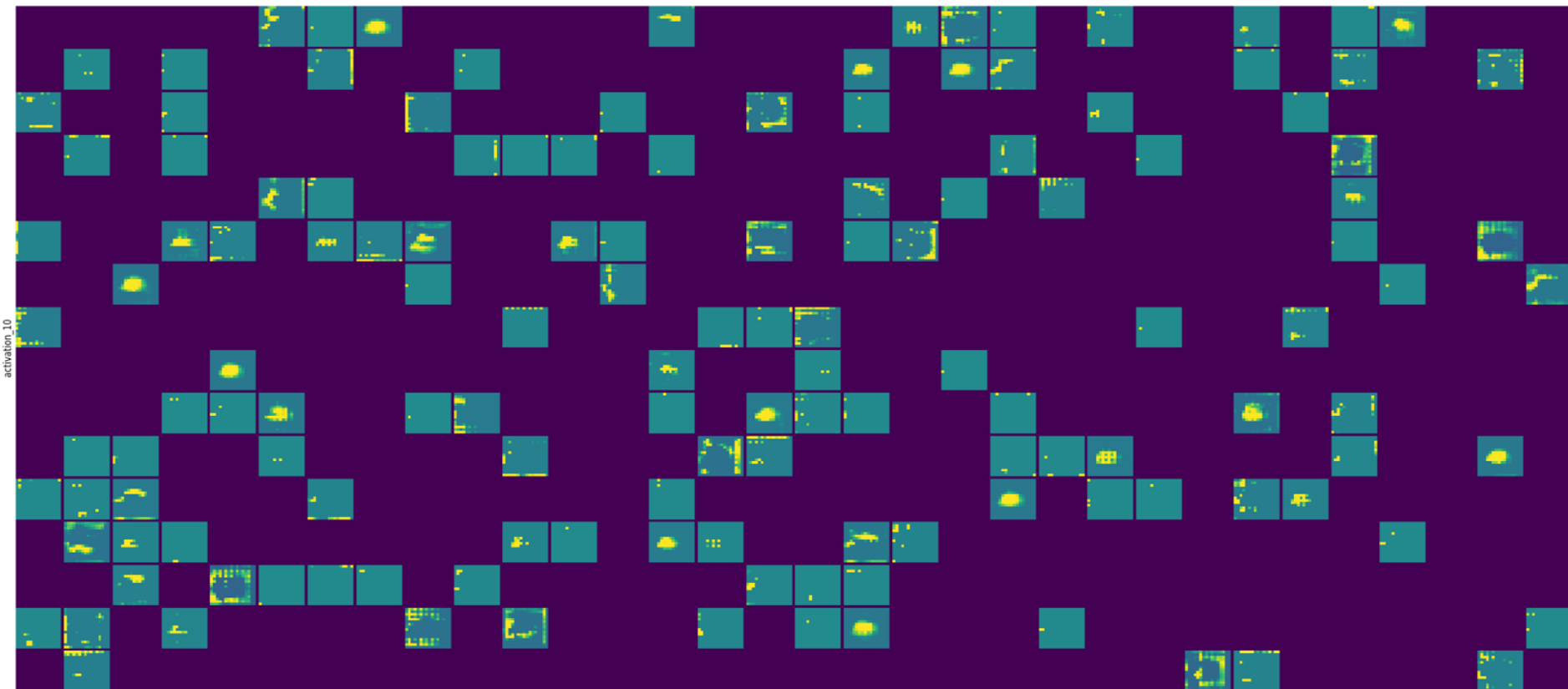
Help of skip connections: Encoder output



Corresponding decoder output: without encoder connection

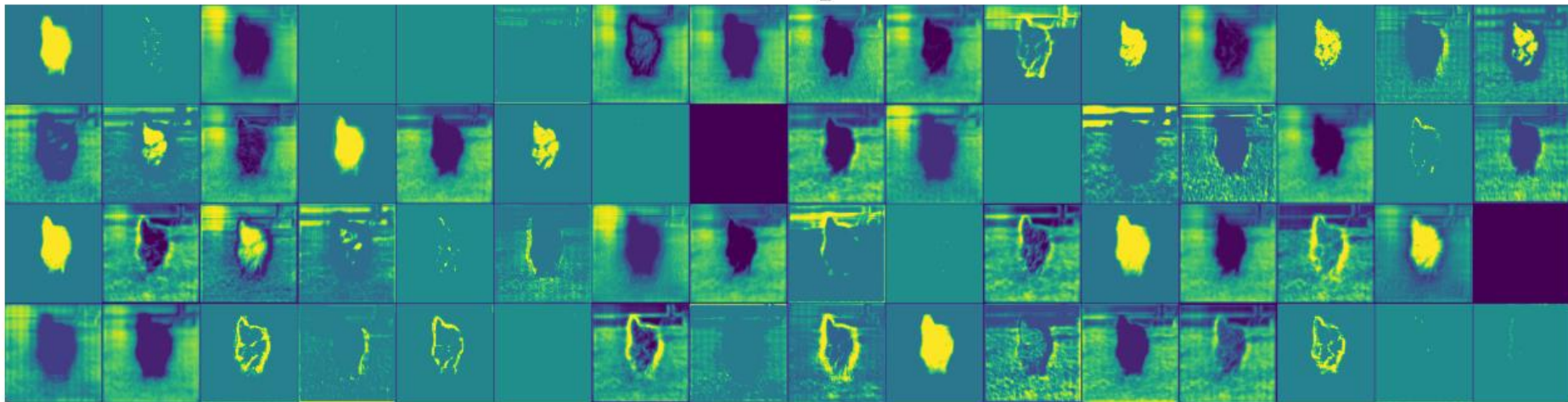


Output after concatenating this 2 and passing to conv



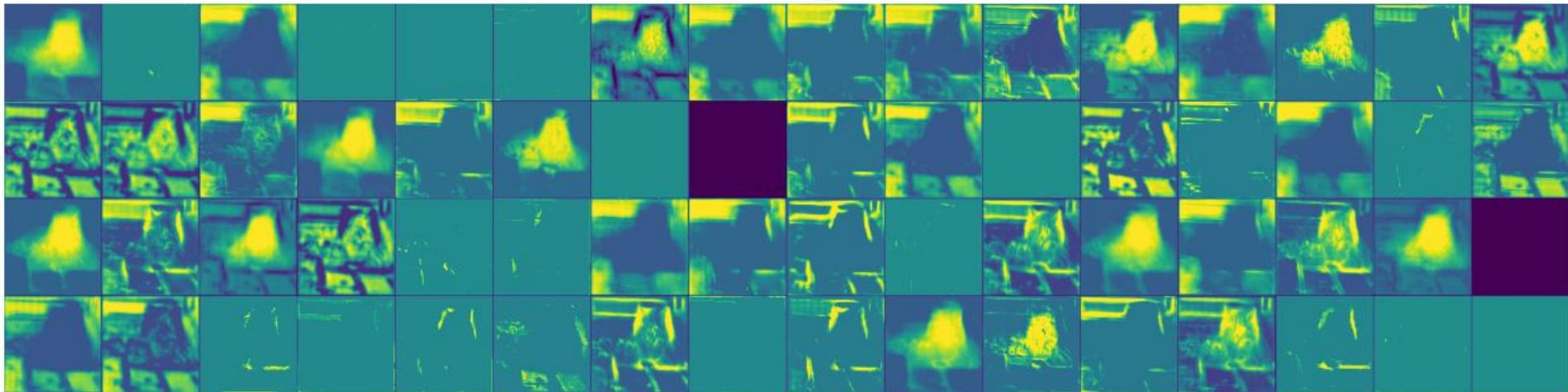
Final features before softmax for dog image

activation_17

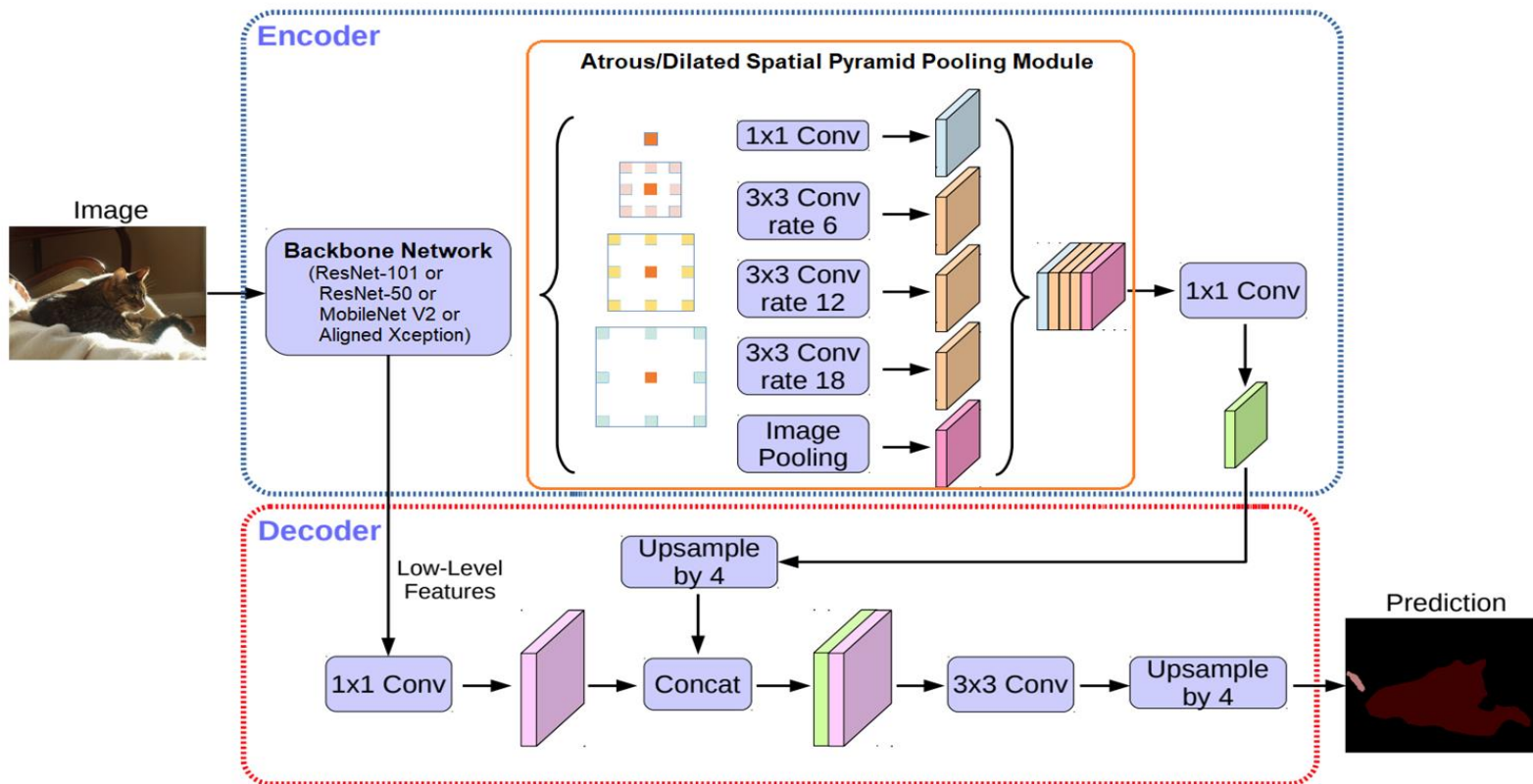


Final feature of cat image before softmax

activation_17



Deeplabv3+





Issues with other models

1. The feature extraction in encoder down-samples the input a lot ... so the final segmentation masks are coarse in nature
2. Objects of different scale can be present in single image.... The models didn't work well with such cases
3. Boundary detection was not refined

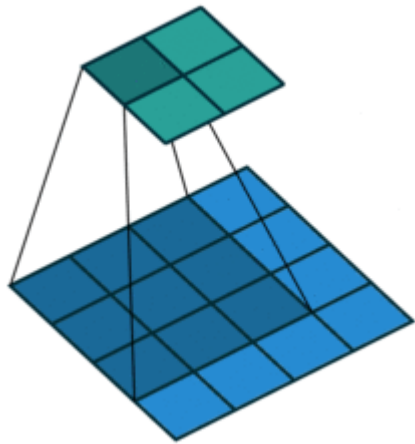


Deeplab solutions

To solve the issues mentioned in previous slides deeplab introduces following novel ideas

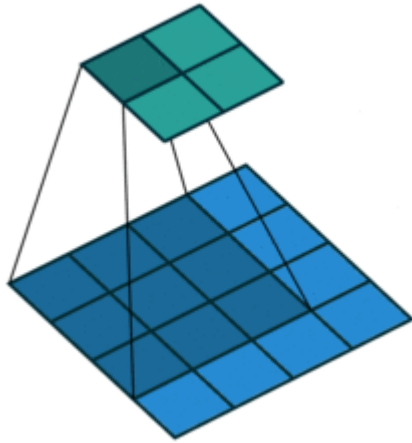
1. Atrous convolutions to handle feature reductions
2. Atrous Spatial Pyramid Pooling (ASPP): to handle various objects scale
3. Upsampling with skip connections from encoder side(Similar to Unet)

Convolutions Revisit



1. We have a kernel(weights of the conv layer) that is slid over the input feature map and at each location, element-wise multiplication followed by a summation of the products is performed to obtain a scalar value
2. The convolutional filter detects a particular feature by **sliding** over the input feature map, i.e, it looks for that feature at each location.

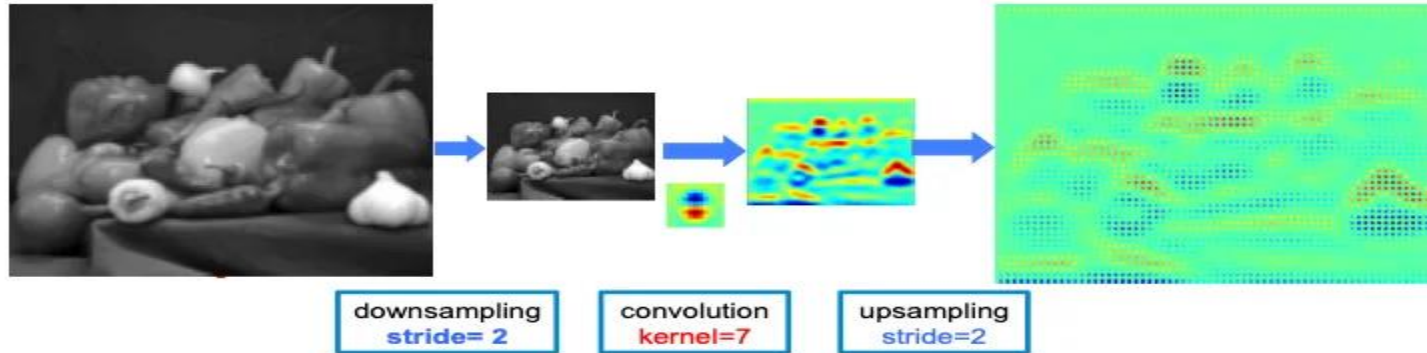
Receptive field



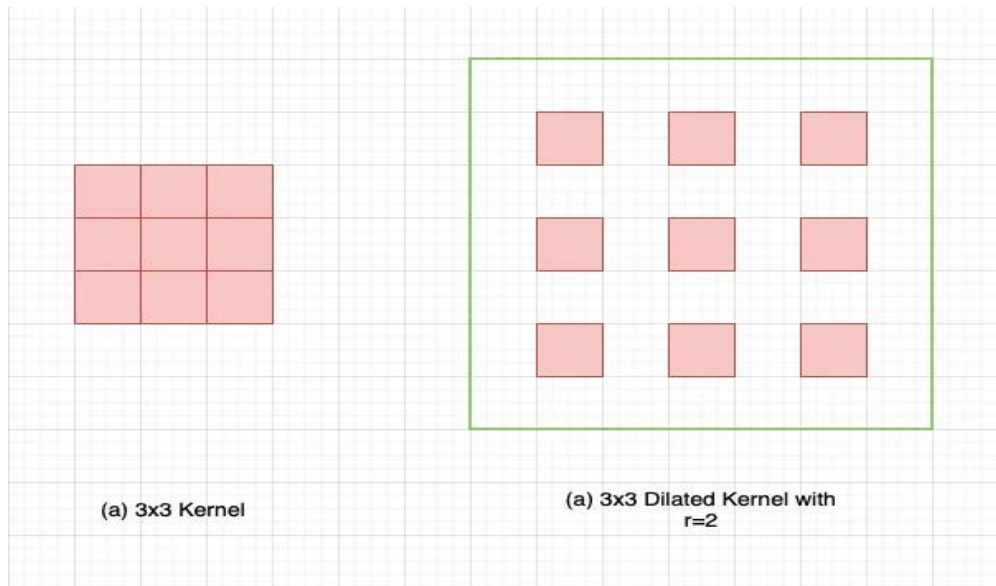
1. Receptive Field is defined as the size of the region of the input feature map that produces each output element.
2. Here we have a receptive field of 3×3 .
3. Higher receptive fields look at greater regions producing fine and dense feature maps

Problems with normal convolutions with pooling

1. Deep CNNs use a combination of Convolutions and max-pooling. This has the disadvantage that, at each step, the spatial resolution of the feature map is halved
2. Implanting the resultant feature map onto the original image results in sparse feature extraction
3. Upsampling and imposing the feature map on the image shows that the responses correspond to only 1/4th of the image locations(Sparse feature extraction).

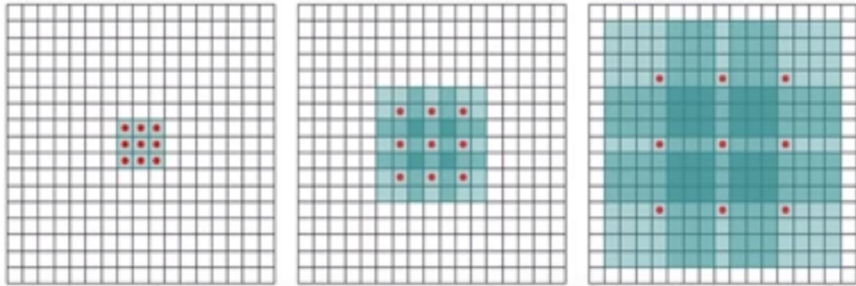


Atrous/Dilated convolutions



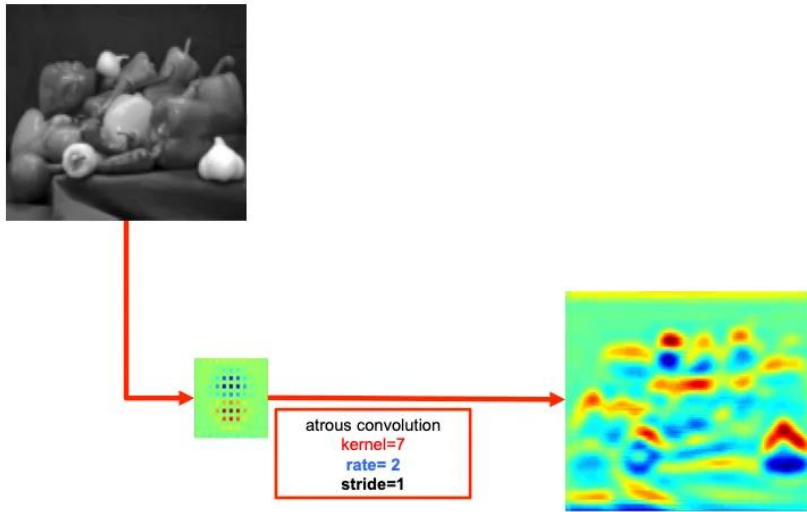
1. Atrous(Dilated) convolution fixes this problem and allows for dense feature extraction.
2. This is achieved a new parameter called **rate(r)**.
3. Atrous convolution is akin to the standard convolution except that the weights of an atrous convolution kernel are spaced r locations apart
4. If $r=1$ then its a normal convolution operation

Dilated convolutions



By controlling the rate parameter, we can arbitrarily control the receptive fields of the conv. layer. This allows the conv. filter to look at larger areas of the input (receptive field) without a decrease in the spatial resolution or increase in the kernel size.

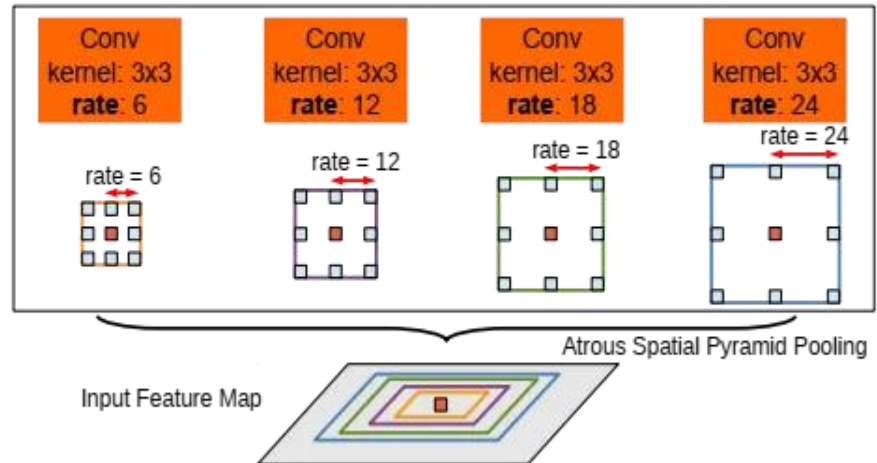
Dilated convolutions advantages



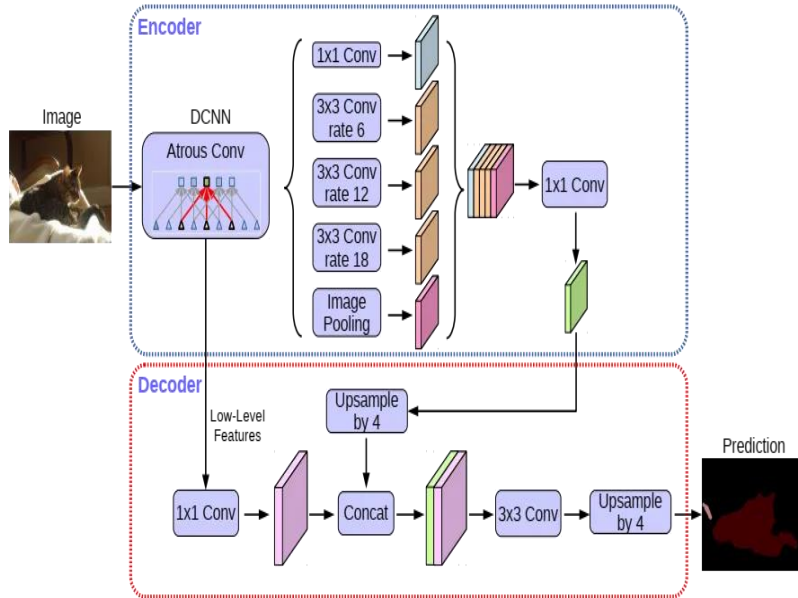
1. Using dilated convolutions we get denser maps
2. It has higher receptive field keeping same number of feature maps and similar feature size
3. Gets global context by seeing more area
4. Helps in detecting and segmenting larger objects
5. Leads to dense segmentation maps as while reconstructing in decoder we concat these feature maps with decoder output

Atrous Spatial Pyramid Pooling (ASPP)

1. As we discussed a lot of different scale objects can be present in single input
2. To handle this we use ASPP.
3. The idea is to apply multiple atrous convolution with different sampling rates to the input feature map, and fuse together.
4. As objects of the same class can have different scales in the image, ASPP helps to account for different object scales which can improve the accuracy.

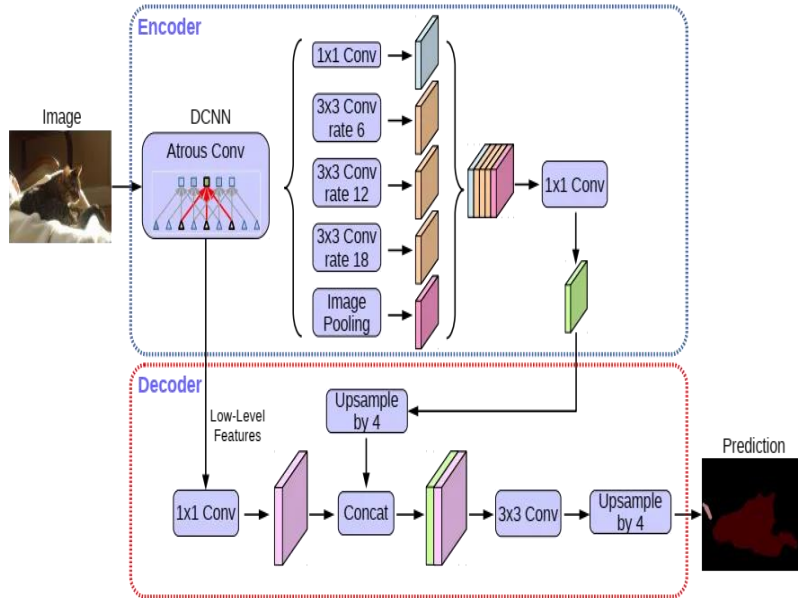


DeeplabV3+ architecture explained



1. Feature extractor like resnet or xception are used to extract features
2. In the paper they have mentioned pooling operations are replaced by depth-wise separable convolution.
3. These extracted features are sent through ASPP module to generate scale invariant features and we use padding = same to keep same size as that of feature extractor output
4. the concatenated feature maps are sent to pointwise convolutions to reduce the number of channels.
5. The feature extractor (or ASPP output) is such that it is downsampled by 16.
6. Now in decoder we try to upsample the feature map by factor of 16 to generate the final segmentation mask output

DeeplabV3+ architecture explained



6. instead of using up-sampling with a factor of 16, the encoded features are first up-sampled by a factor of 4 and concatenated with corresponding low level features from the encoder module having the same spatial dimensions

7. After concatenation, a few 3 x 3 convolutions are applied and the features are up-sampled by a factor of 4. This gives the output of the same size as that of the input image.



Reference for transpose convolutions and segmentation

1. [Cs231n lecture](#)
2. [Transpose convolution](#)
3. [Colab notebook](#)
4. [Understanding skip connections](#)
5. [Deeplab Architecture](#)