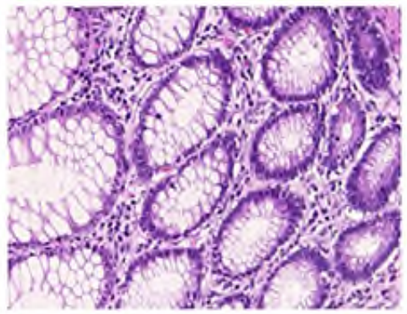
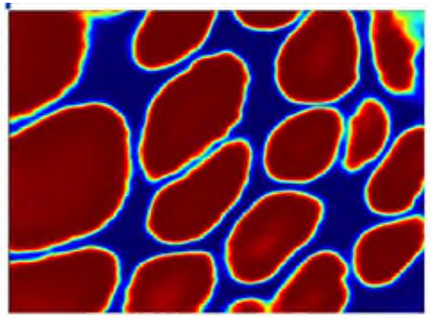
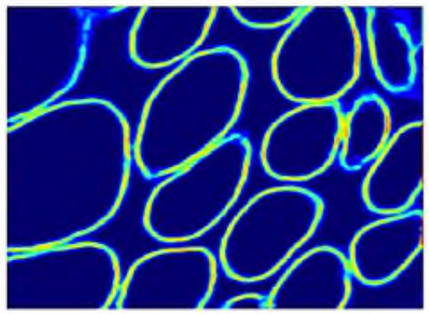
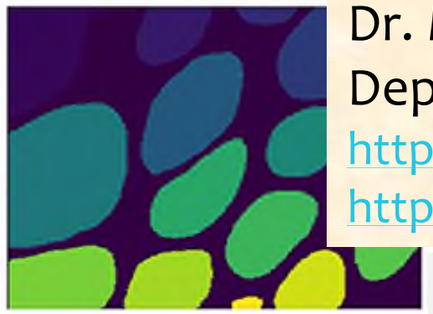
CS867 Computer Vision

Fall 2021

Lecture 13-14 : Semantic Segmentation

Dr. Muhammad Moazam Fraz
Department of Computing, NUST SEECS
http://seecs.nust.edu.pk/faculty/mmfraz.html
http://vision.seecs.edu.pk

Today's Class : Learning Outcomes

The students will learn about

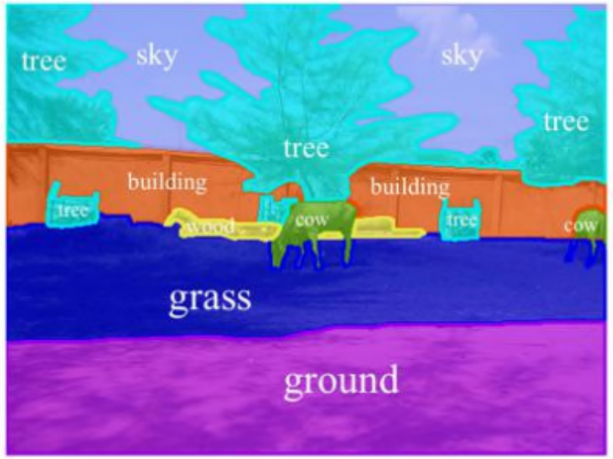
- Semantic Segmentation
- Fully Convolutional Network
- Transposed Convolutions
- U-Net Architecture

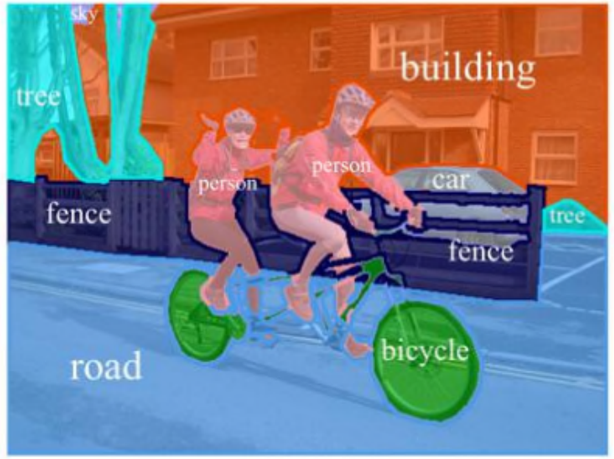
Today's Class

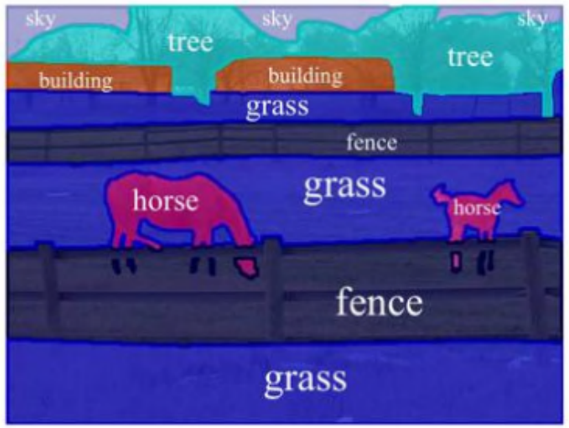
What is semantic segmentation?

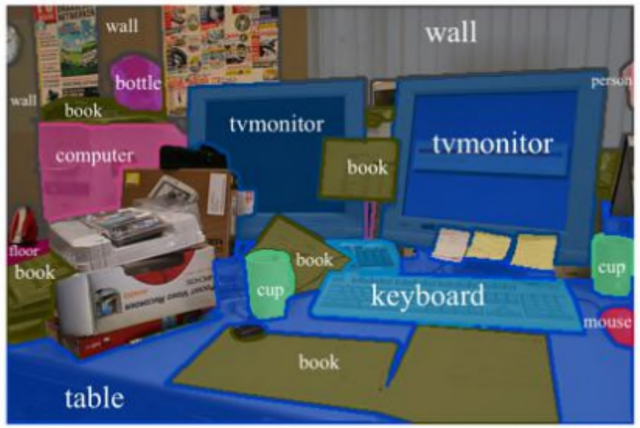
1. Idea: recognizing, understanding what's in the image in pixel level.
2. A lot more difficult

   (Most of the traditional methods cannot tell different objects.)









"Two men riding on a bike in front of a building on the road. And there is a car."

What is semantic segmentation?

1. Idea: recognizing, understanding what's in the image in pixel level.
2. A lot more difficult

   (Most of the traditional methods cannot tell different objects.)

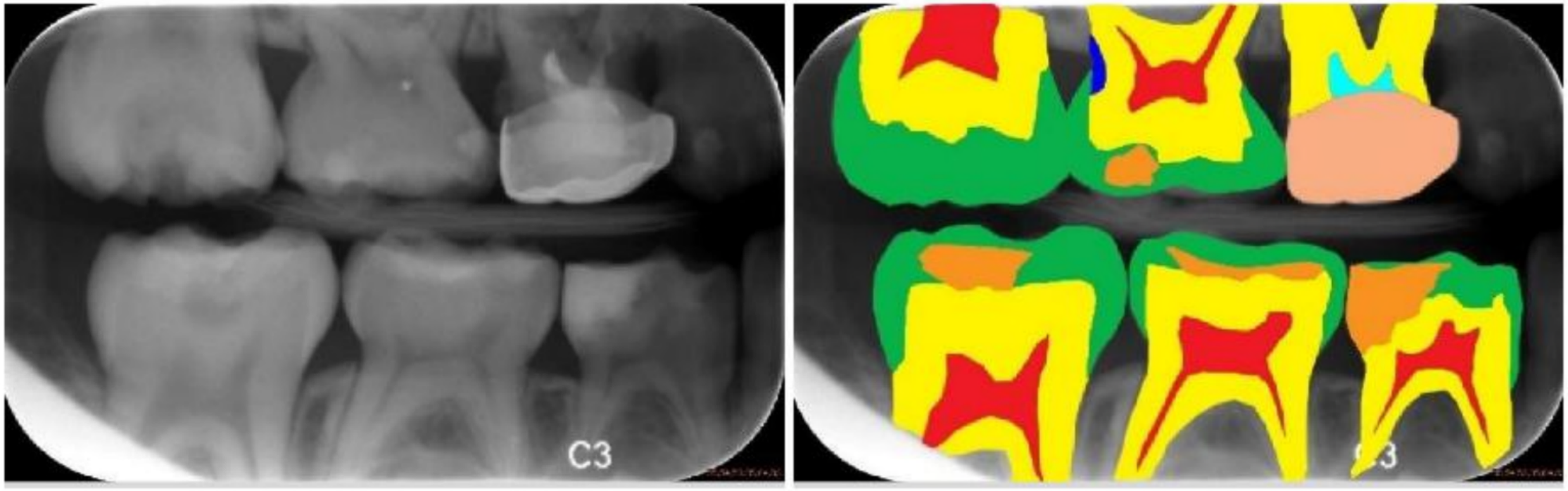
No worries, even the best ML researchers find it very challenging.

3. Output: regions with different (and limited number of) classes
   1. COCO detection challenge: 80 classes.
   2. PASCAL VOC challenge: 21 classes

Why semantic segmentation?

1. robot vision and understanding
2. autonomous driving

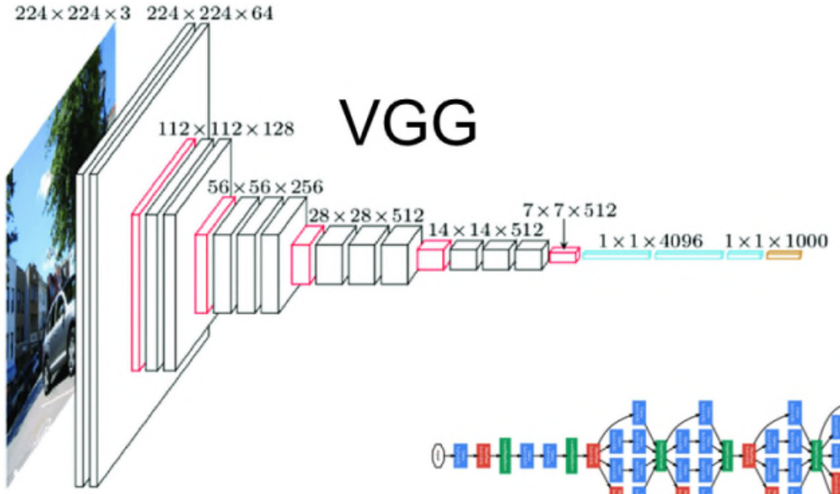
1. Why semantic segmentation?
    3. medical purposes (ISBI Challenge)



Semantic segmentation before deep learning

1. Clustering methods
2. Motion & Interactive Segmentation
3. Partial differential equation-based methods
   - **Level-set methods**
   - **Parametric methods (Active Contours, Snakes)**
   - **Fast marching methods**
4. Graph partitioning methods
   - **Markov random fields**
5. Watershed transformation

So far: Image Classification



Class Scores

Cat: 0.9

Dog: 0.05

Car: 0.01

...

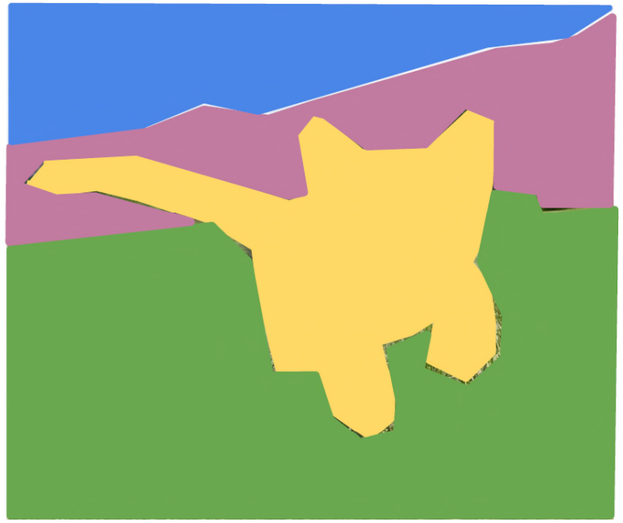
Computer Vision Tasks: Semantic Segmentation

Classification

CAT

No spatial extent

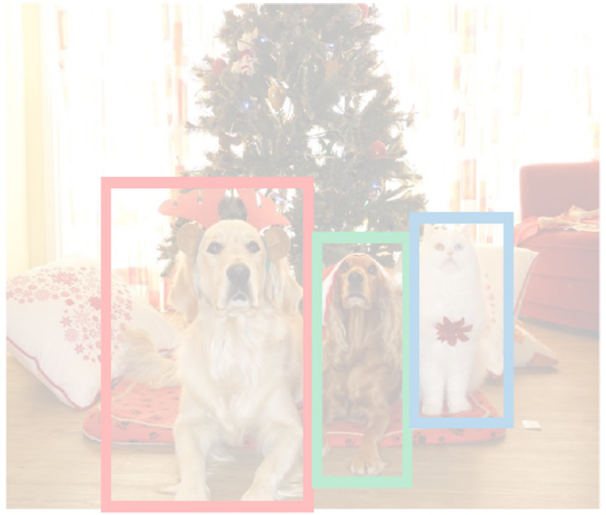
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection



DOG, DOG, CAT

Instance Segmentation

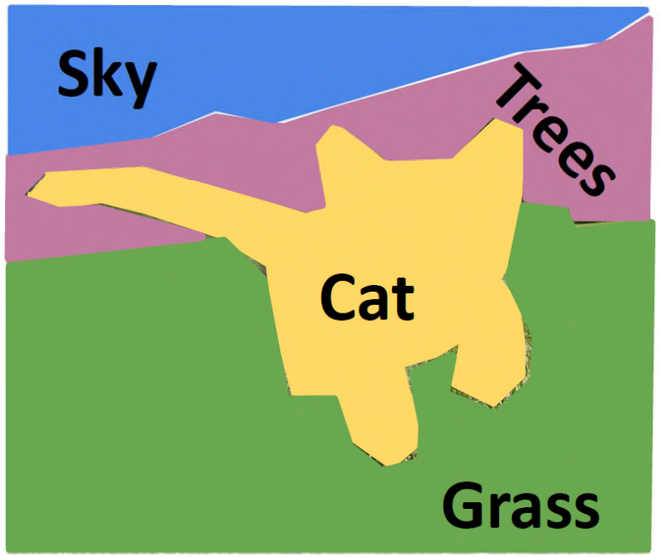
DOG, DOG, CAT

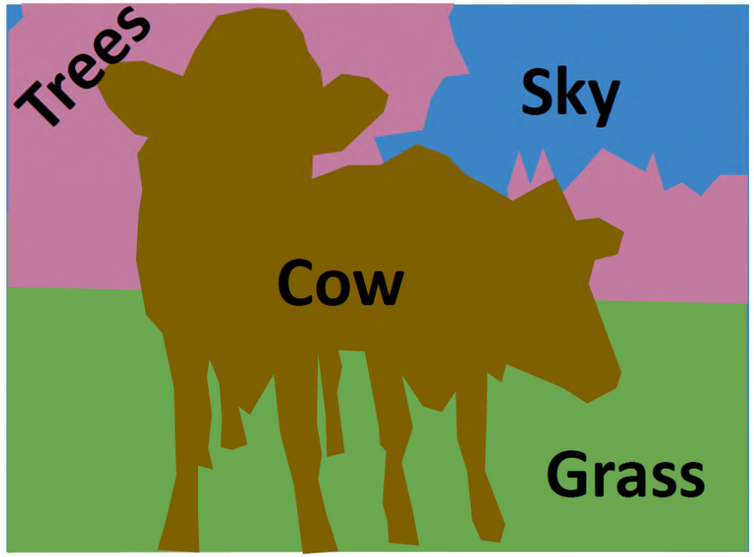
Multiple Objects

Semantic Segmentation

Label each pixel in the image with a category label

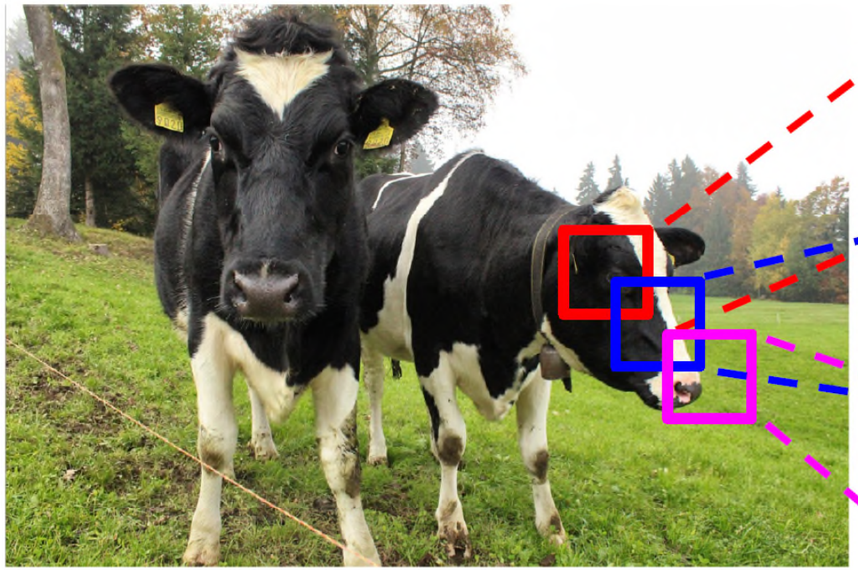
Don’t differentiate instances, only care about pixels



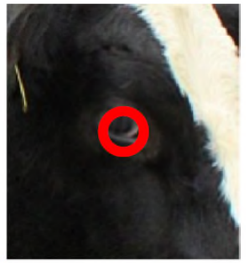


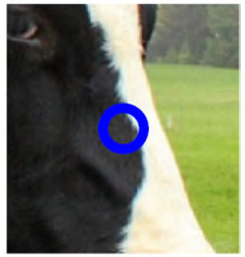
Semantic Segmentation Idea: Sliding Window

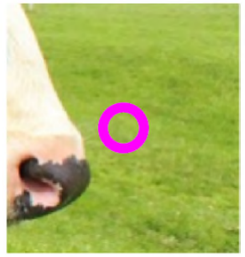
Full image



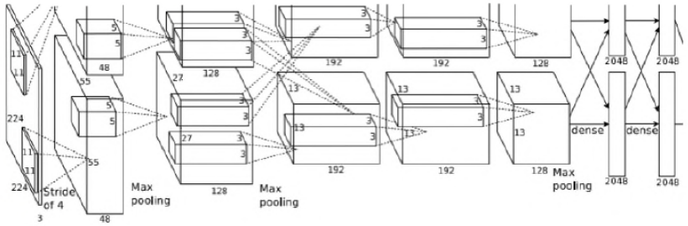
Extract patch



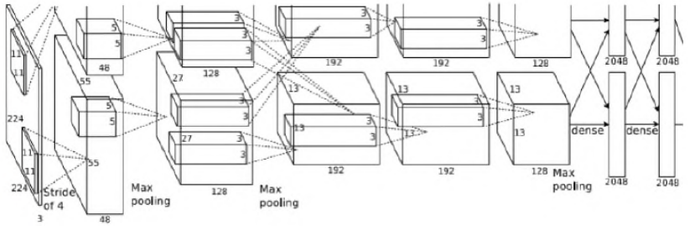




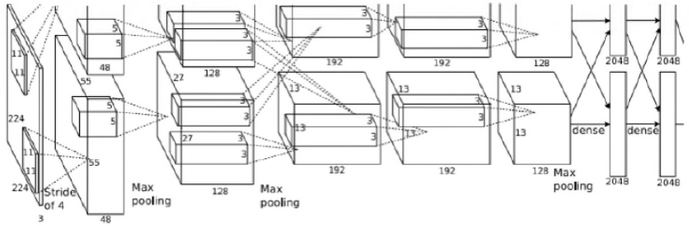
Classify center pixel with CNN



Cow



Cow

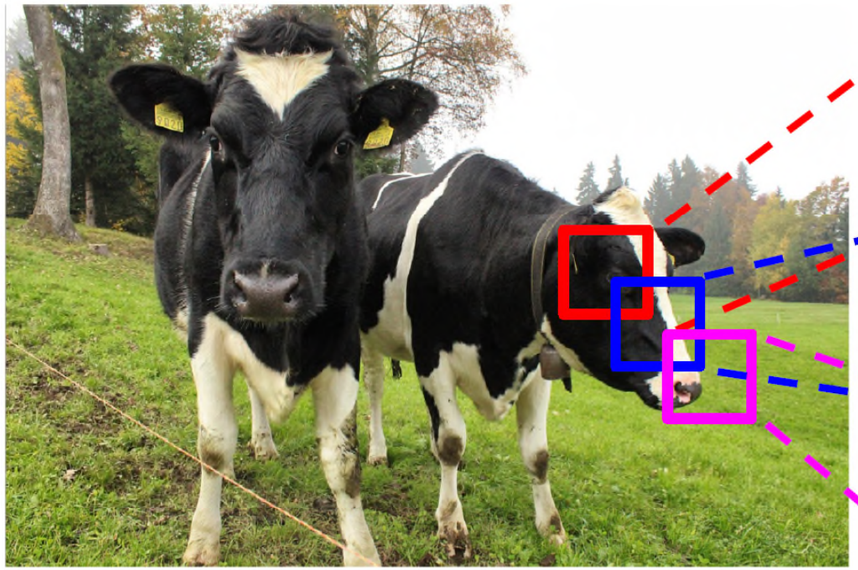


Grass

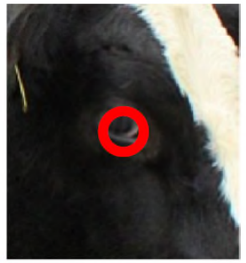
Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

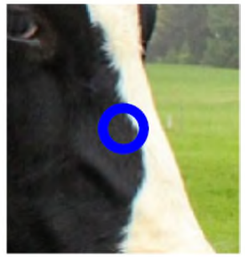
Semantic Segmentation Idea: Sliding Window

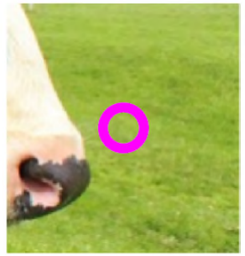
Full image



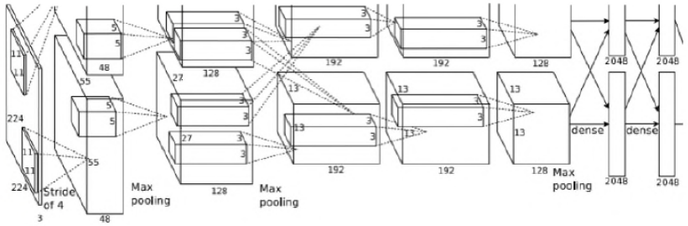
Extract patch



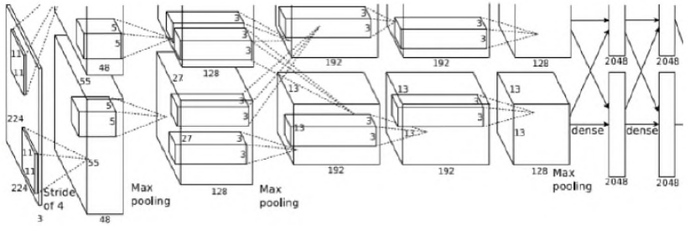




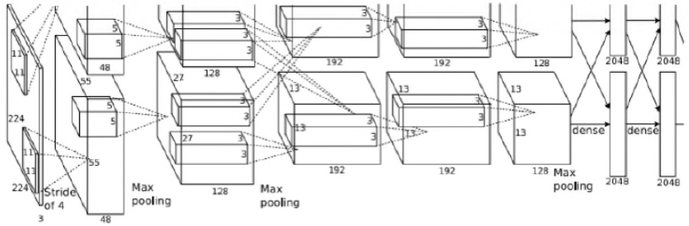
Classify center pixel with CNN



Cow



Cow



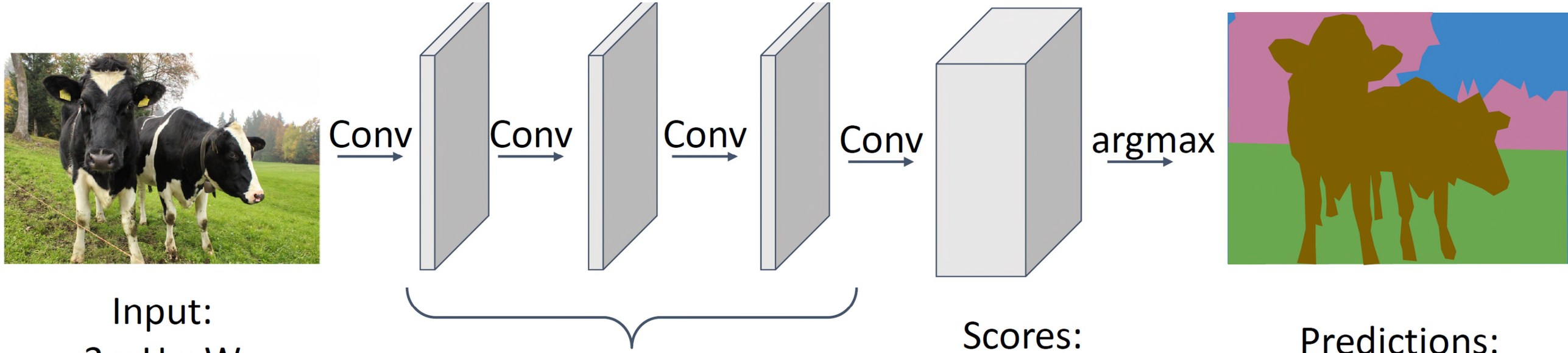
Grass

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

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Semantic Segmentation: Fully Convolutional Network

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!



Loss function: Per-Pixel cross-entropy

Long et al, “Fully convolutional networks for semantic segmentation”, CVPR 2015

Semantic Segmentation: Fully Convolutional Network

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!

Input:

Conv → Conv → Conv → Conv → argmax

volutions:
x H x W

Scores:
C x H x W

Predictions:
H x W

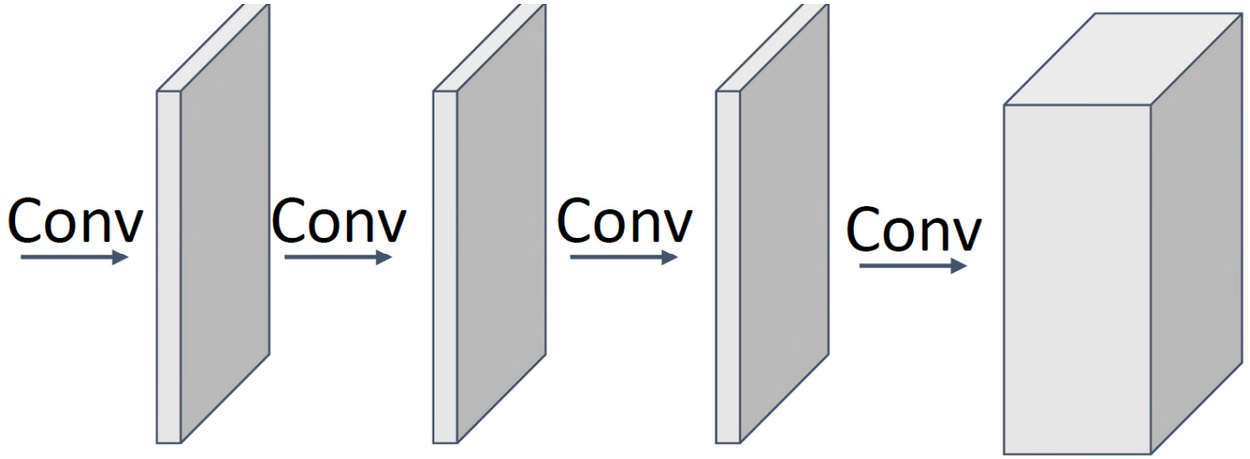
Problem #1: Effective receptive field size is linear in number of conv layers: With L 3x3 conv layers, receptive field is 1+2L

Loss function: Per-Pixel cross-entropy

Long et al, "Fully convolutional networks for semantic segmentation", CVPR 2015

Semantic Segmentation: Fully Convolutional Network

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!



Input:

volutions:
x H x W

ons:
V

ntropy

Problem #1: Effective receptive field size is linear in number of conv layers: With L 3x3 conv layers, receptive field is 1+2L

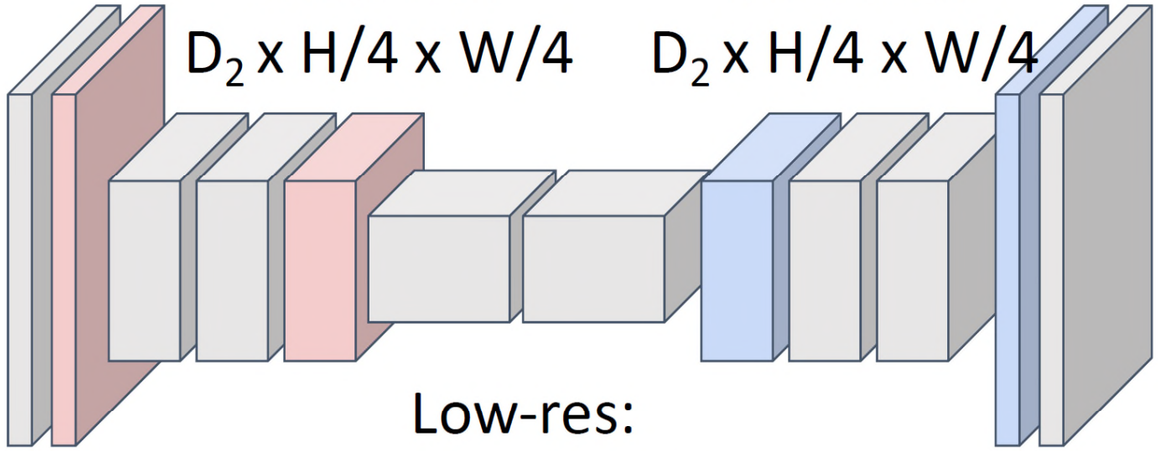
Problem #2: Convolution on high res images is expensive! Recall ResNet stem aggressively downsamples

Long et al, "Fully convolutional networks for semantic segmentation", CVPR 2015

Semantic Segmentation: Fully Convolutional Network

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

Input:
3 x H x W



Predictions:
H x W

Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015
Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

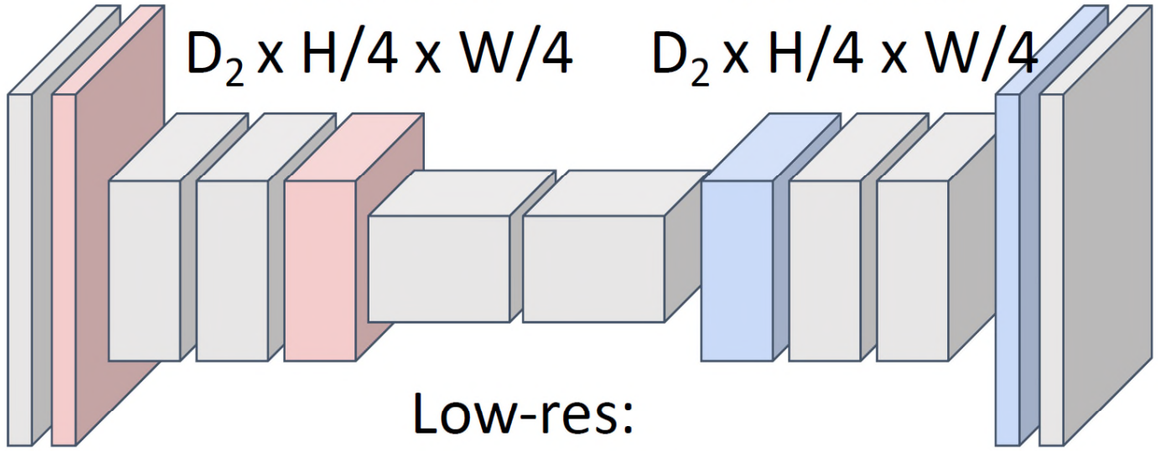
Semantic Segmentation: Fully Convolutional Network

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

Downsampling: Pooling, Strided convolution

Upsampling: ???

Input:
3 x H x W



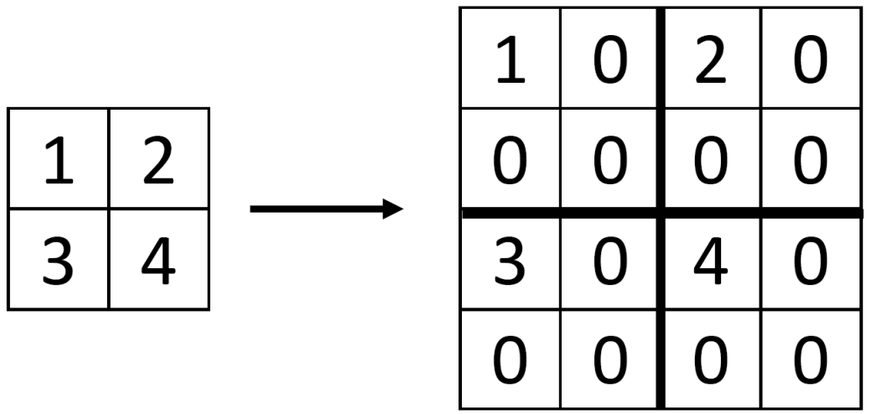
Predictions:
H x W

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Semantic Segmentation: Fully Convolutional Network

In-Network Upsampling: “Unpooling”

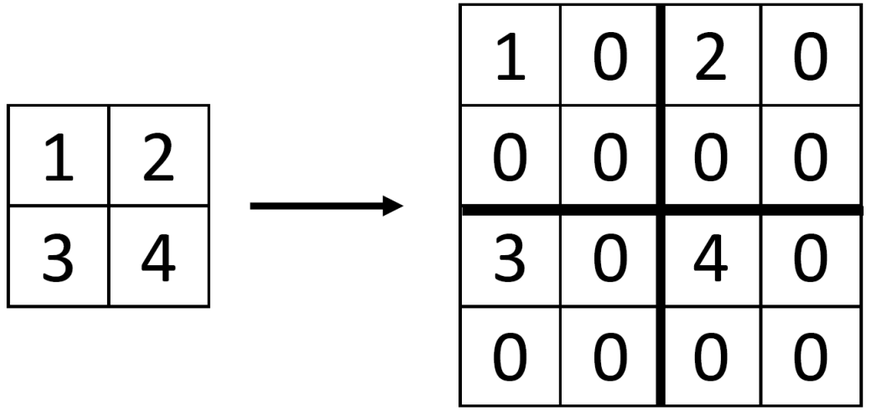
Bed of Nails



Semantic Segmentation: Fully Convolutional Network

In-Network Upsampling: "Unpooling"

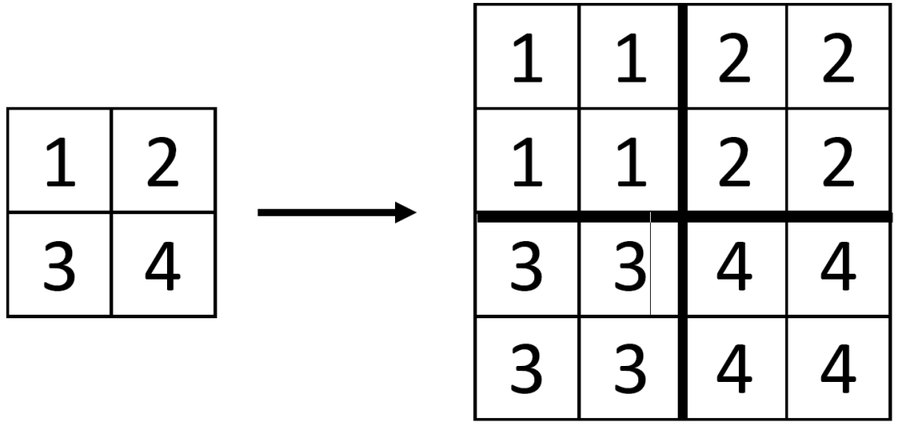
Bed of Nails



Input
C x 2 x 2

Output
C x 4 x 4

Nearest Neighbour

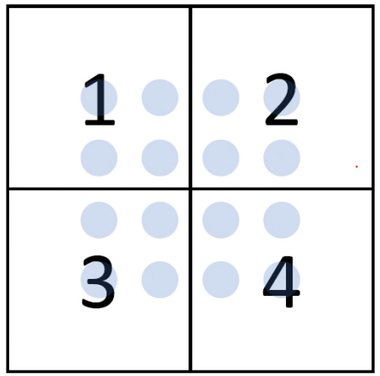


Input
C x 2 x 2

Output
C x 4 x 4

Semantic Segmentation: Fully Convolutional Network

In-Network Upsampling: "Bicubic Interpolation"



0.68	1.02	1.56	1.89
1.35	1.68	2.23	2.56
2.44	2.77	3.32	3.65
3.11	3.44	3.98	4.32

Input: C x 2 x 2

Output: C x 4 x 4

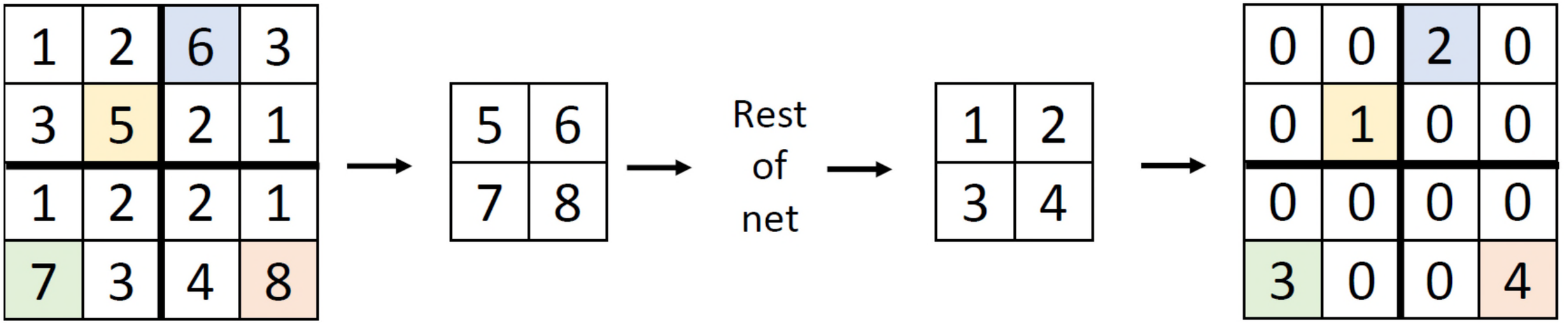
Use **three** closest neighbors in x and y to construct **cubic** approximations
(This is how we normally resize images!)

Semantic Segmentation: Fully Convolutional Network

In-Network Upsampling: "**Max Unpooling**"

Max Pooling: Remember which position had the max

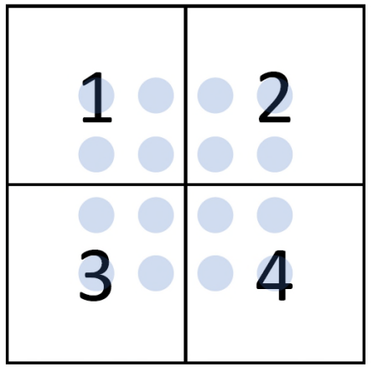
Max Unpooling: Place into remembered positions



Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Semantic Segmentation: Fully Convolutional Network

In-Network Upsampling: "Bilinear Interpolation"



1.00	1.25	1.75	2.00
1.50	1.75	2.25	2.50
2.50	2.75	3.25	3.50
3.00	3.25	3.75	4.00

Input: C x 2 x 2

Output: C x 4 x 4

$$f_{x,y} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - i|) \max(0, 1 - |y - j|) \quad i \in \{\lfloor x \rfloor - 1, \ldots, \lceil x \rceil + 1\} \quad j \in \{\lfloor y \rfloor - 1, \ldots, \lceil y \rceil + 1\}$$

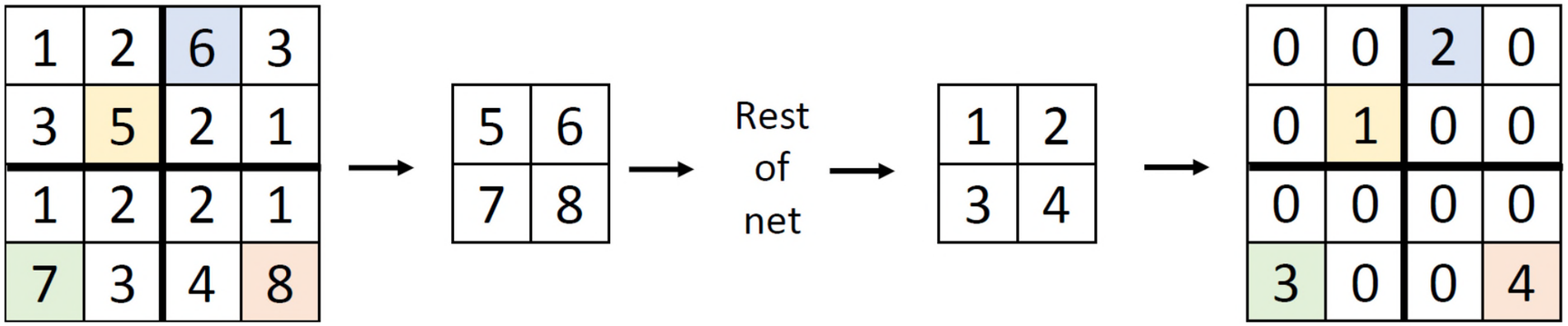
Use two closest neighbors in x and y to construct linear approximations

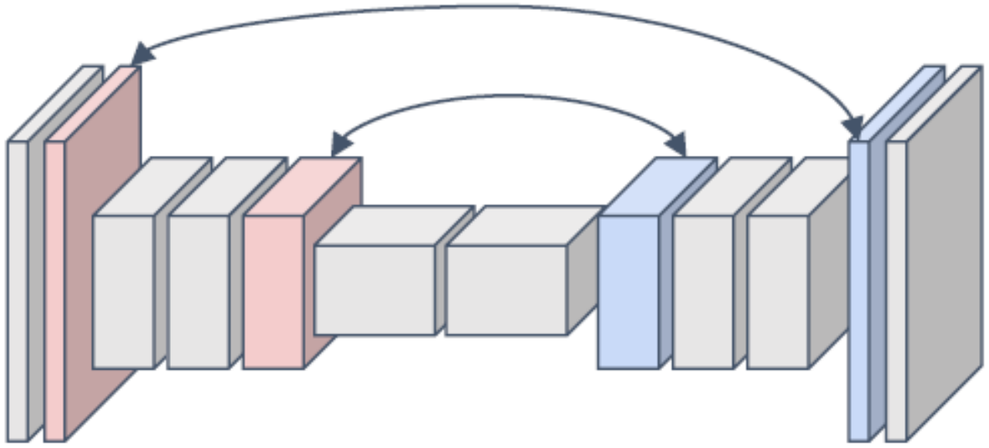
Semantic Segmentation: Fully Convolutional Network

In-Network Upsampling: "**Max Unpooling**"

Max Pooling: Remember which position had the max

Max Unpooling: Place into remembered positions





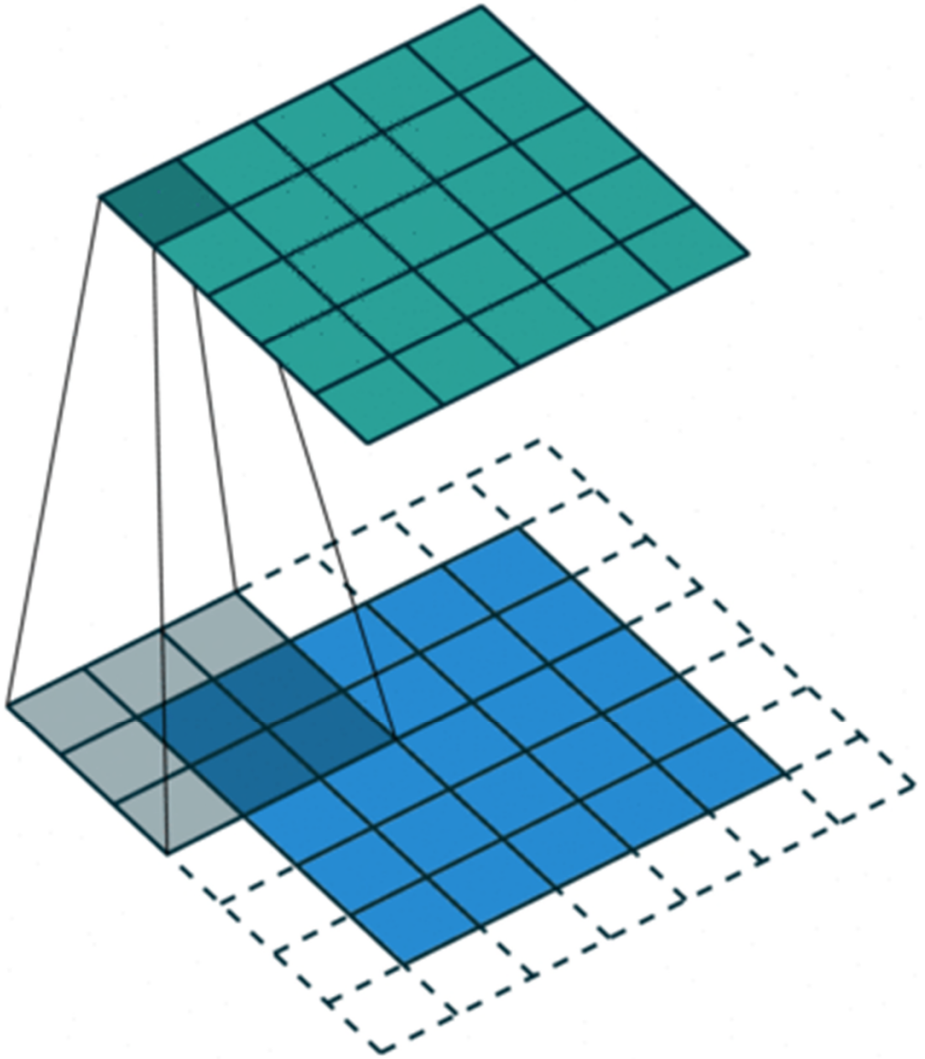
Pair each **downsampling** layer with an **upsampling** layer

Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Up-sampling

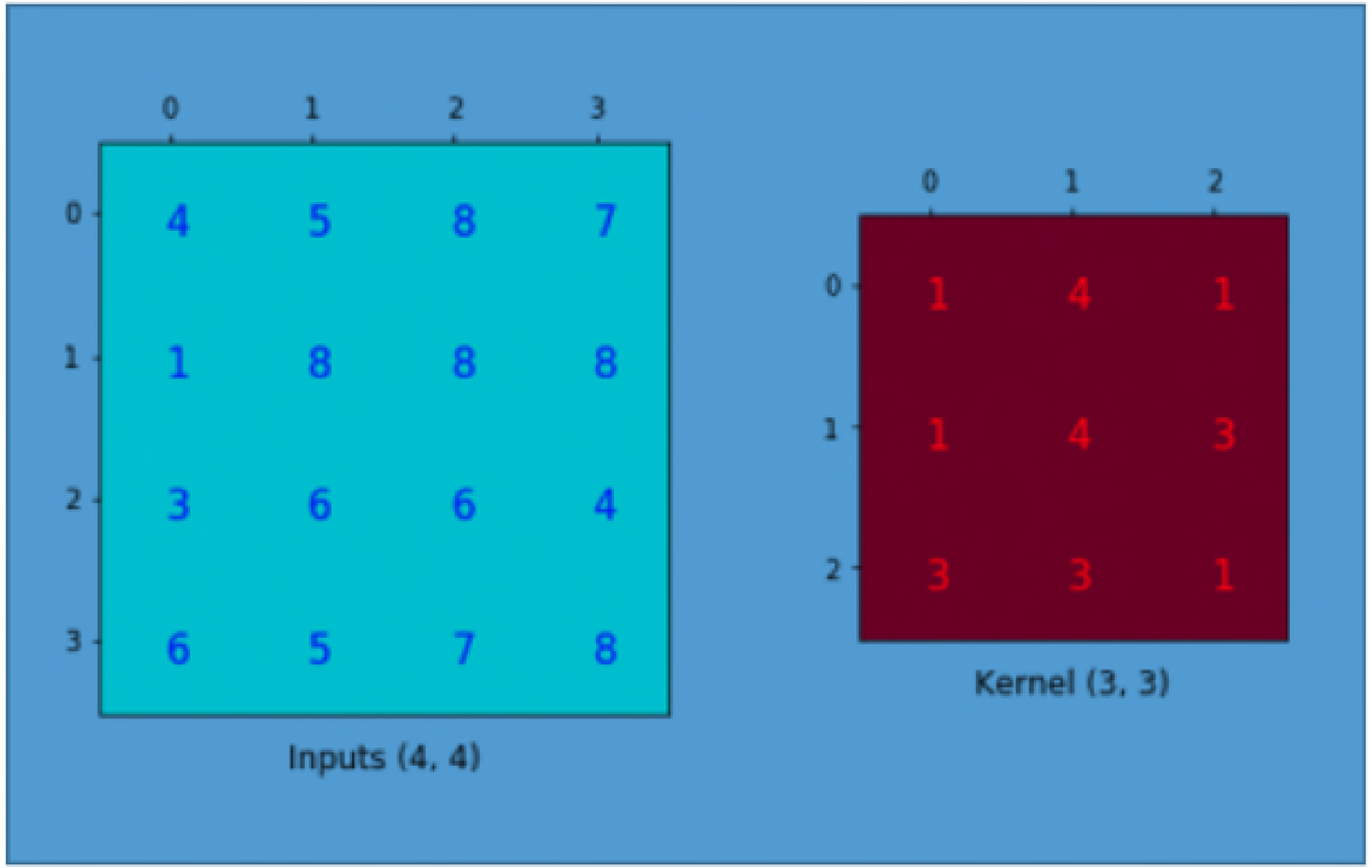
- There are various methods to conduct up-sampling operation:
- Interpolation
  - Nearest neighbor interpolation
  - Bi-linear interpolation
  - Bi-cubic interpolation
  - All these methods involve some interpolation method which we need to chose when deciding a network architecture.
  - It is like a **manual feature engineering** and there is **nothing that the network can learn about.**
- Convolutions
  - Transposed Convolutions
  - Dilated Convolutions

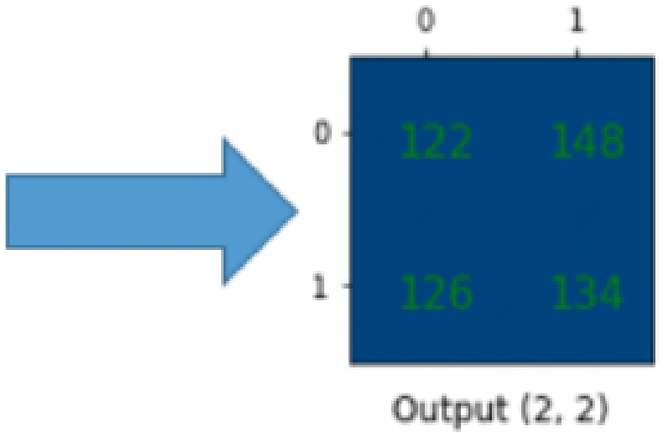
Normal convolutions



2D convolution using a kernel size of 3, stride of 1 and padding

Normal convolutions



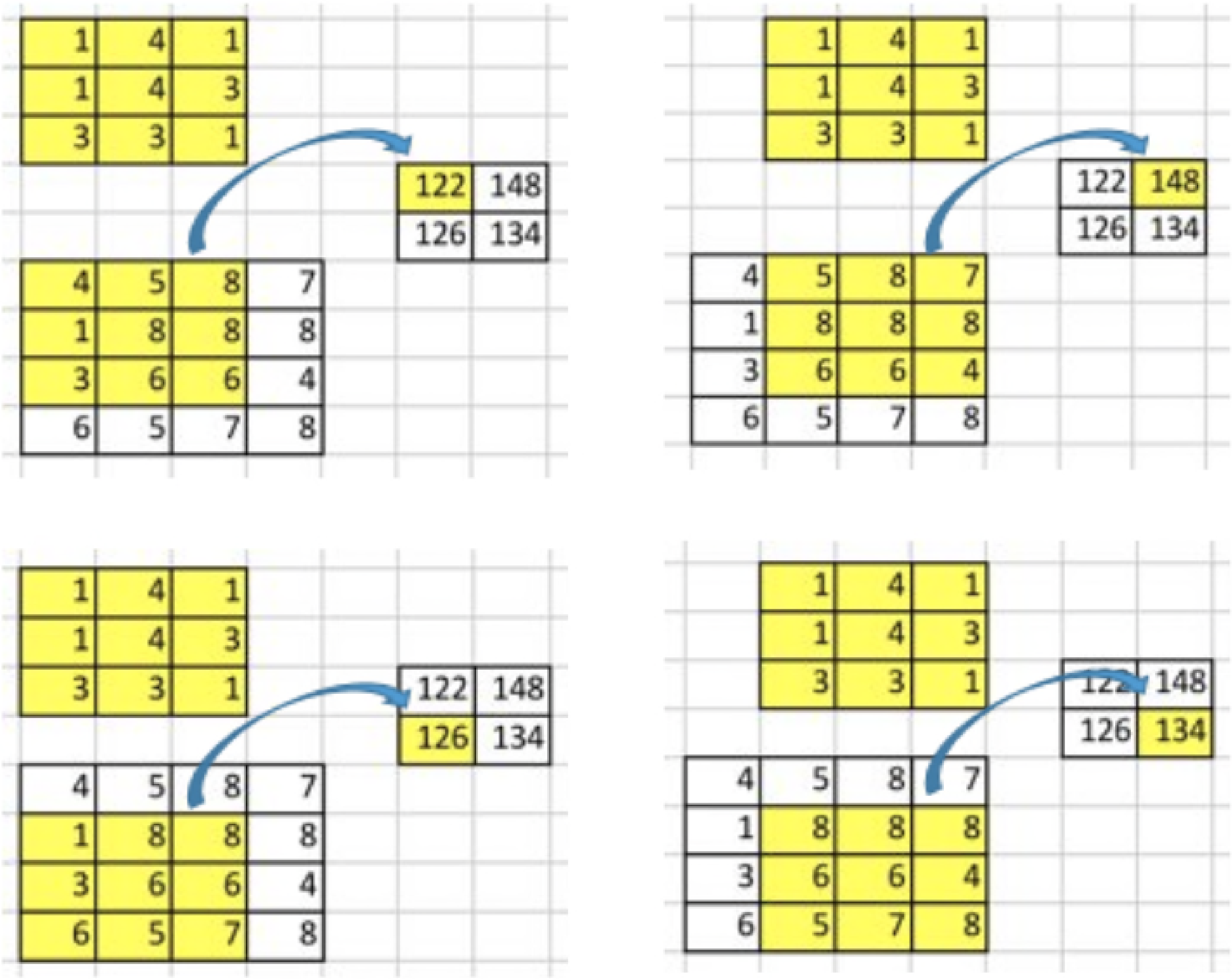


the output is a 2x2 matrix

we have a 4x4 matrix and apply a convolution operation on it with a 3x3 kernel, with no padding, and with a stride of 1

The convolution operation calculates the sum of the element-wise multiplication between the input matrix and kernel matrix.

Since we have no padding and the stride of 1, we can do this only 4 times. Hence, the output matrix is 2x2.



Normal convolutions : One important point

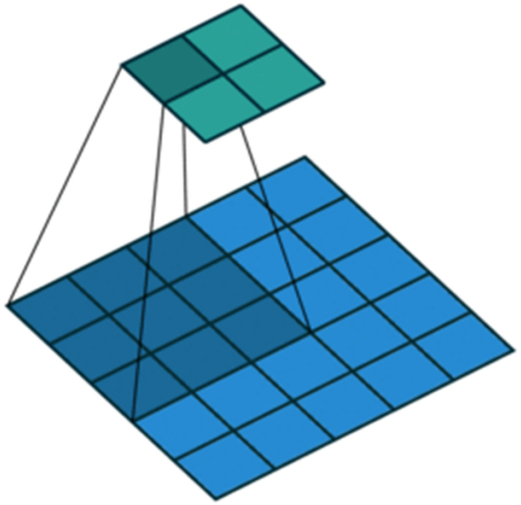
- One important point of such convolution operation is that the positional connectivity exists between the input values and the output values.

- For example, the top left values in the input matrix affect the top left value of the output matrix.

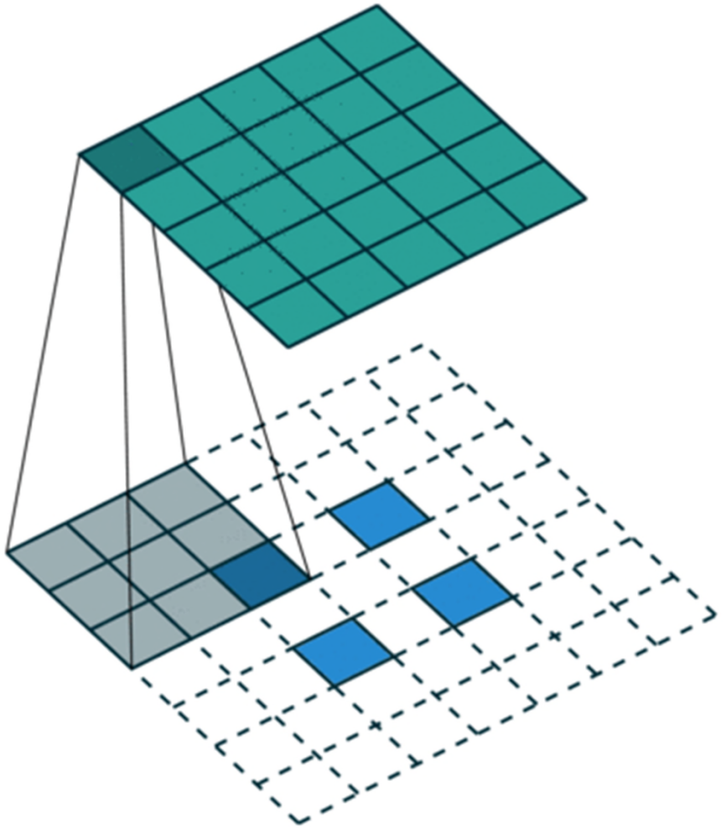
- More concretely, the 3x3 kernel is used to connect the 9 values in the input matrix to 1 value in the output matrix.

- **A convolution operation forms a many-to-one relationship.** Let's keep this in mind as we need it later on.

2D convolution with no padding, stride of 2 and kernel of 3



Transposed 2D convolution with no padding, stride of 2 and kernel of 3



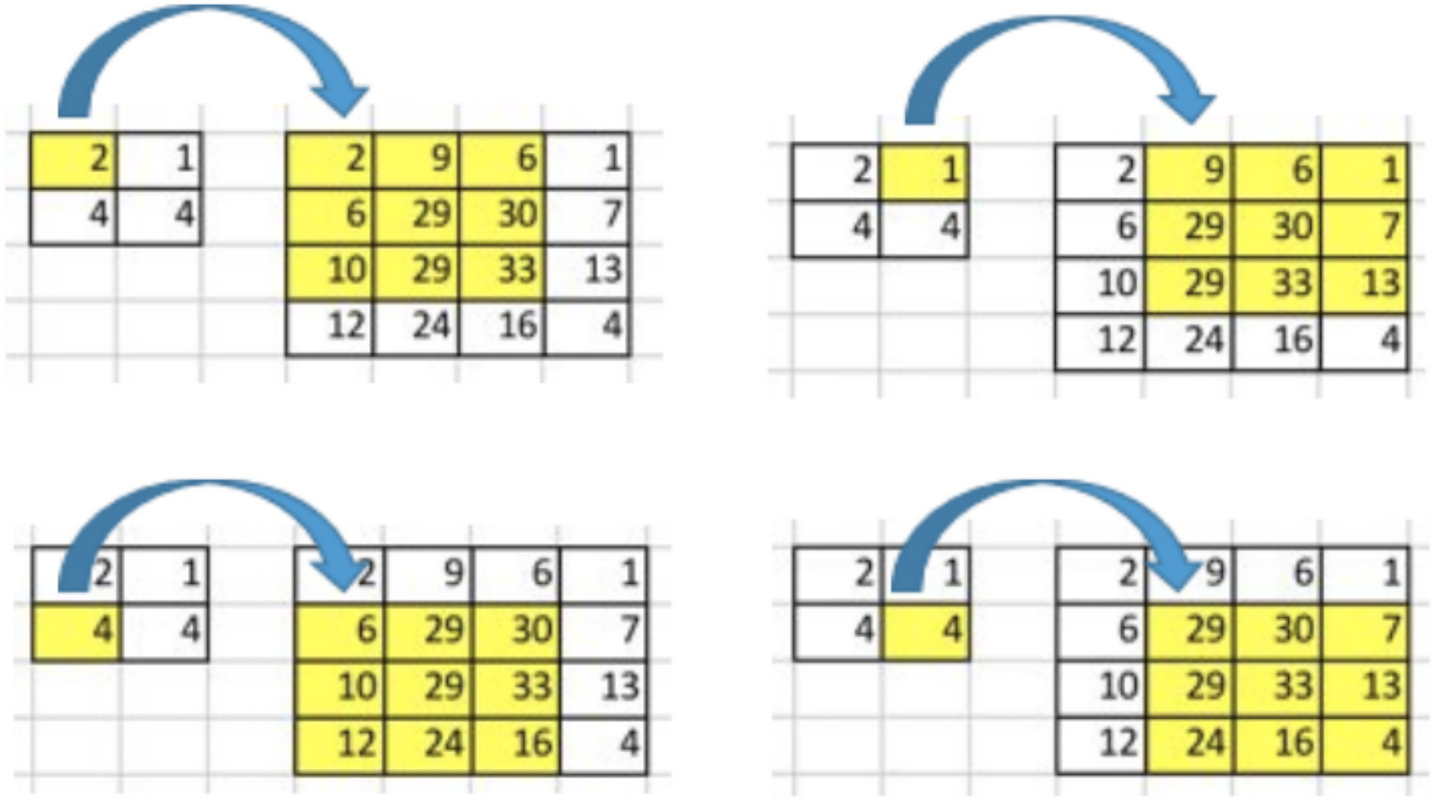
It guarantees that the output will be a 5x5 image as well, while still performing a normal convolution operation.

To achieve this, we need to perform some fancy padding on the input.

We want to associate 1 value in a matrix to 9 values in another matrix. It's a **one-to-many relationship**.

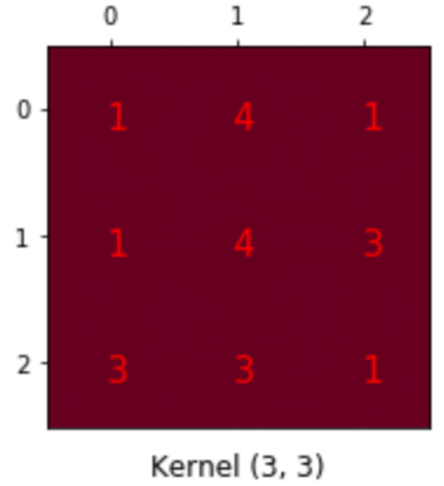
This is like going backward of convolution operation, and it is the core idea of **transposed convolution**

For example, we up-sample a 2x2 matrix to a 4x4 matrix.
The operation maintains the 1-to-9 relationship



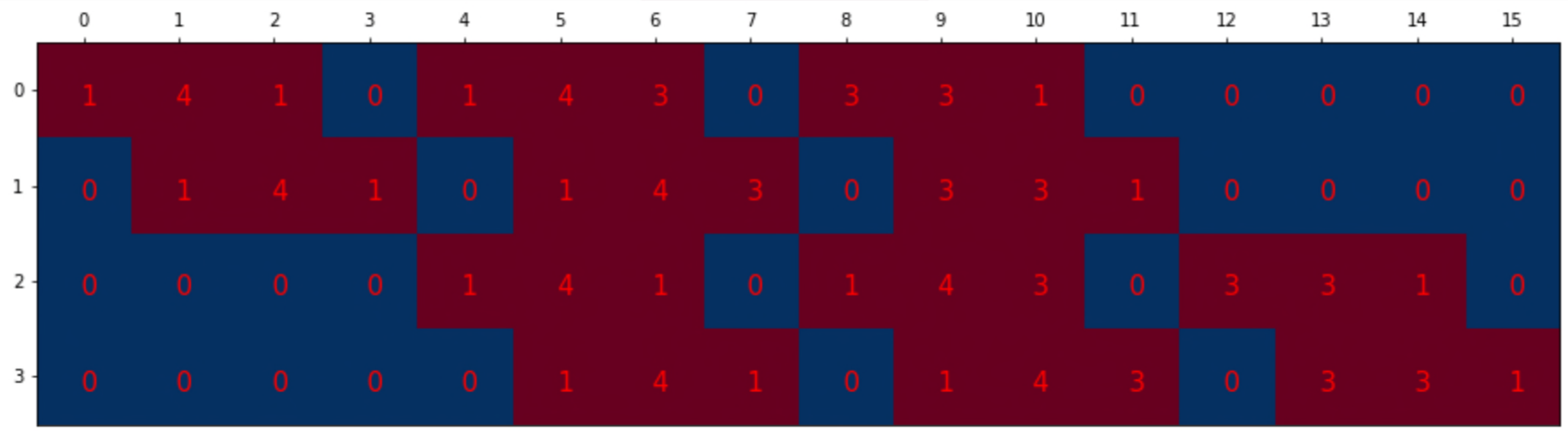
Transposed Convolutions

we need to define the **convolution matrix** and the **transposed convolution matrix**



Kernel (3, 3)

We rearrange the 3x3 kernel into a 4x16 matrix as

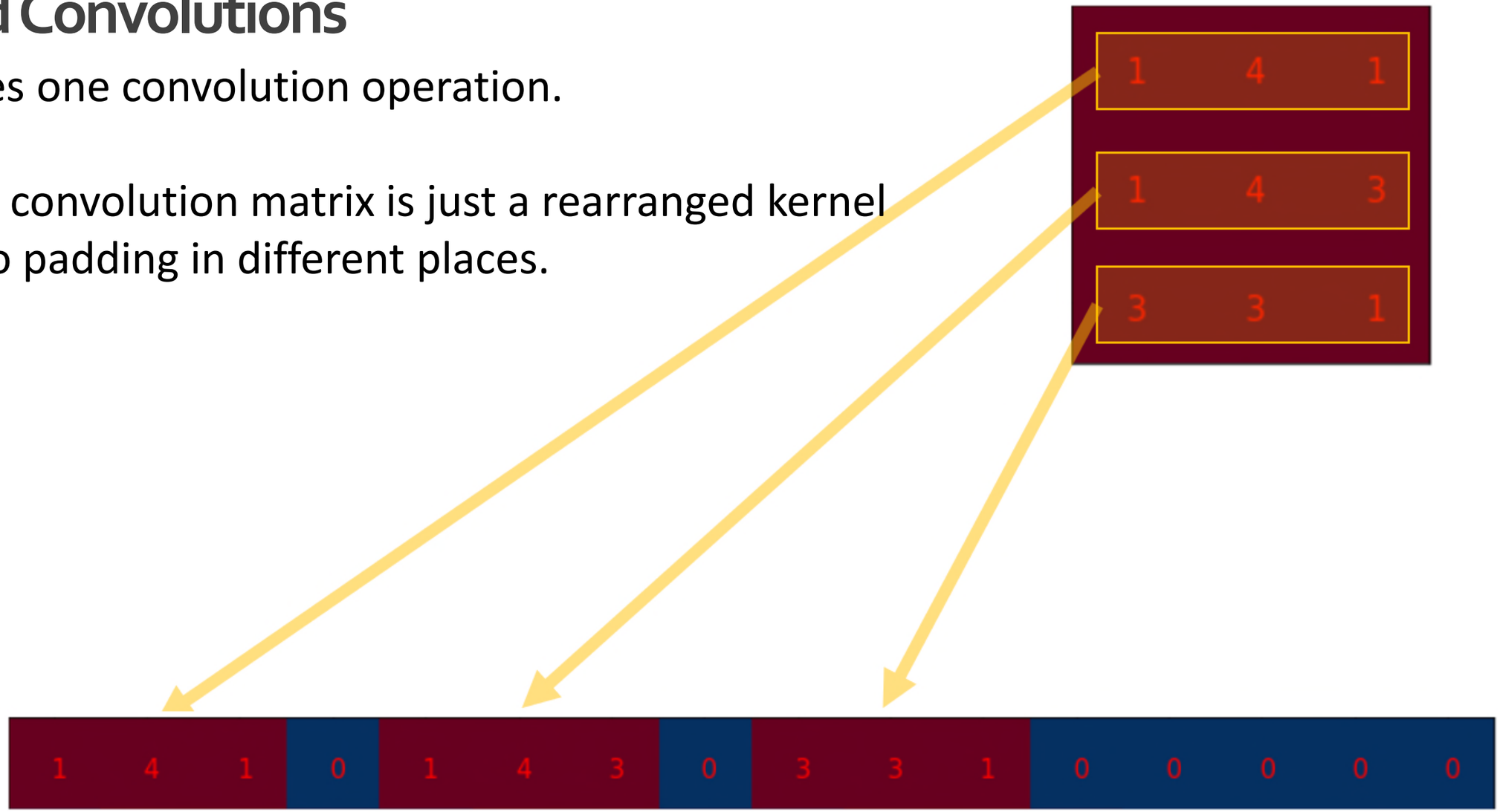


Convolution Matrix (4, 16)

Transposed Convolutions

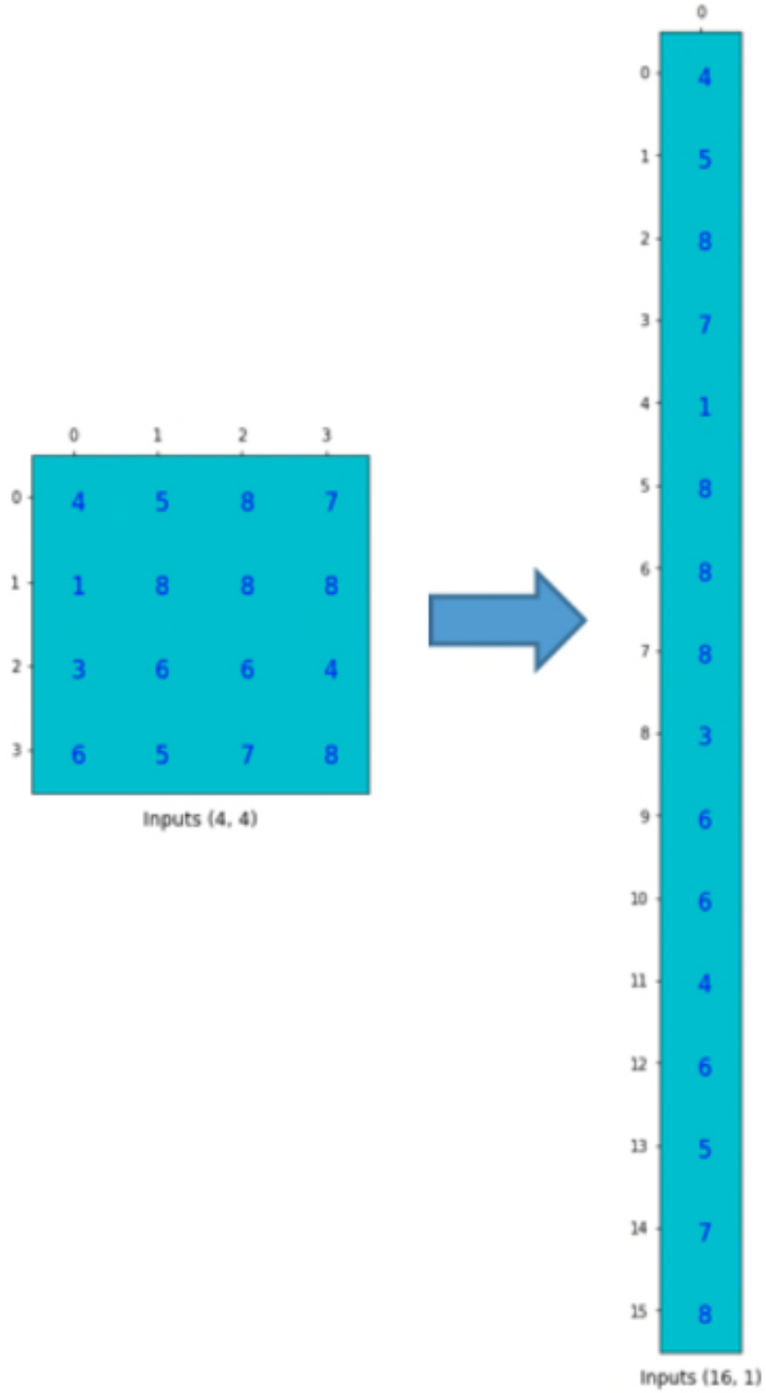
Each row defines one convolution operation.

Each row of the convolution matrix is just a rearranged kernel matrix with zero padding in different places.

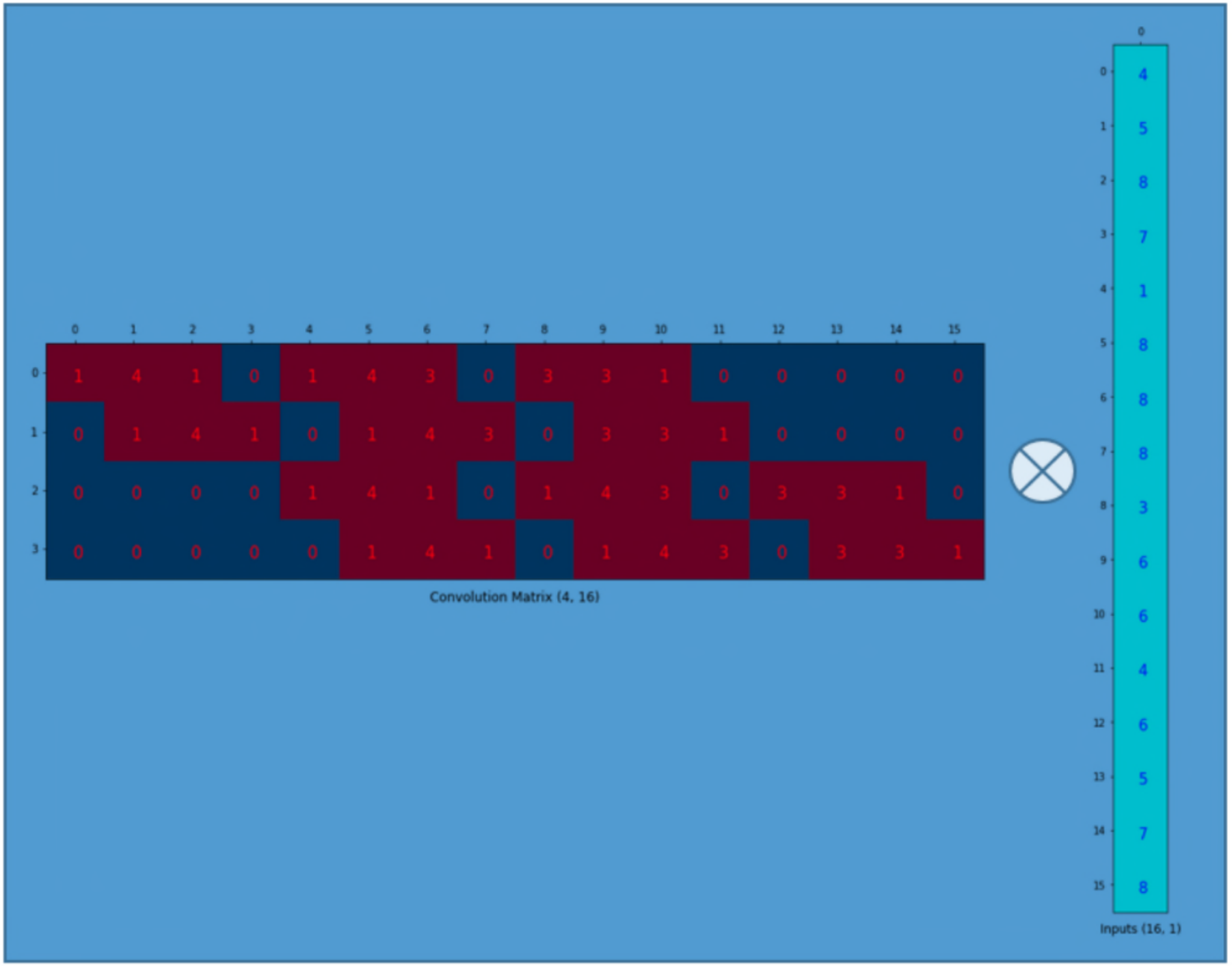


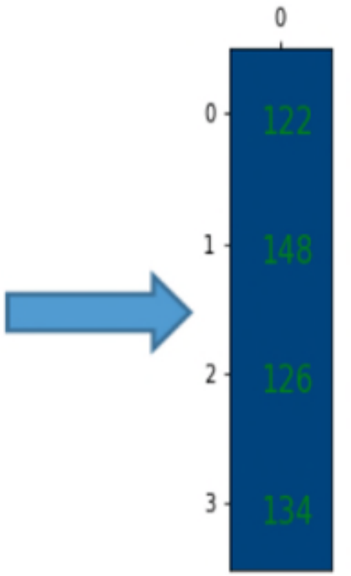
Transposed Convolutions

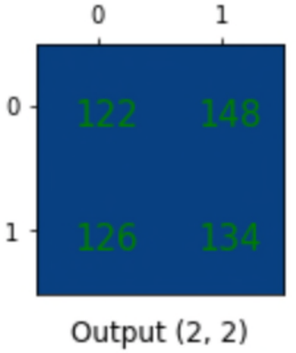
We flatten the input matrix (4x4) into a column vector (16x1)



We can matrix-multiply the 4x16 convolution matrix with the 16x1 input matrix (16 dimensional column vector)


Convolution Matrix (4, 16)
Inputs (16, 1)

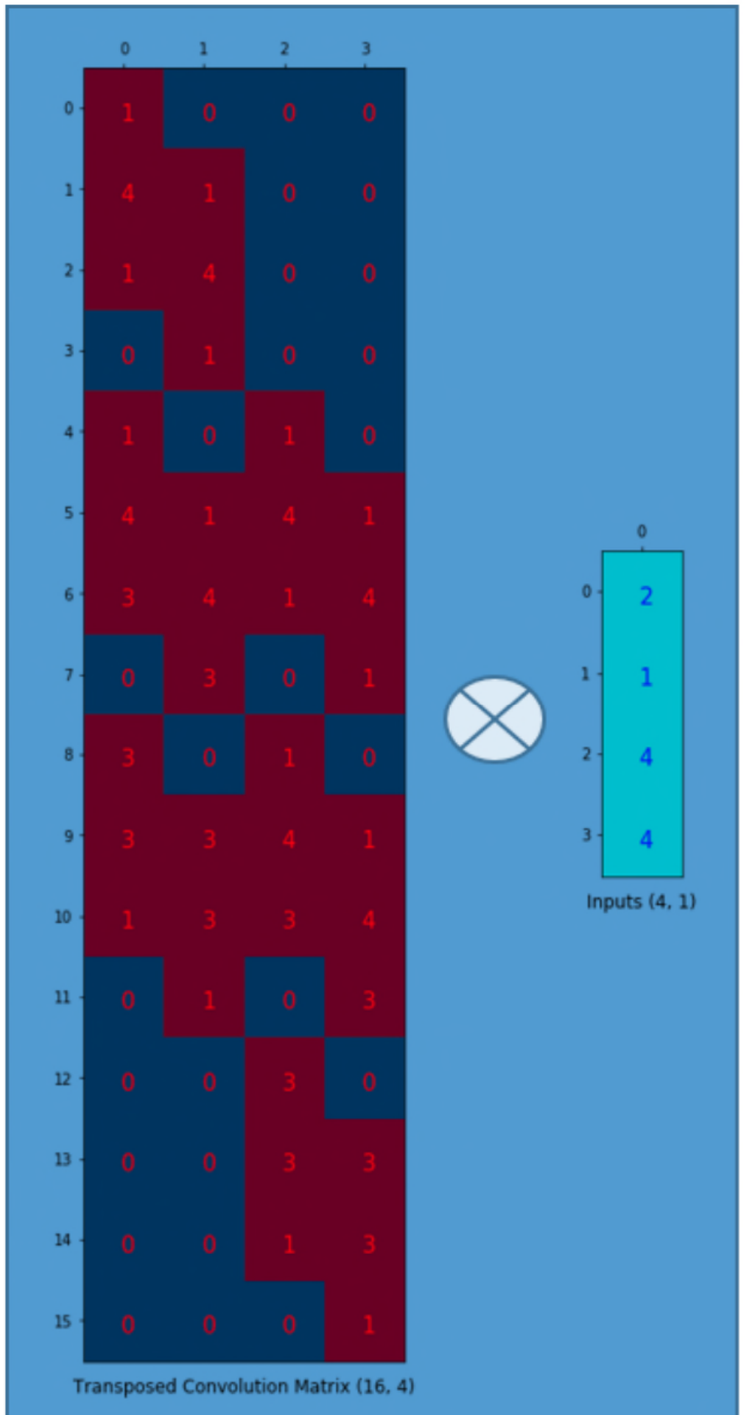

Output (4, 1)

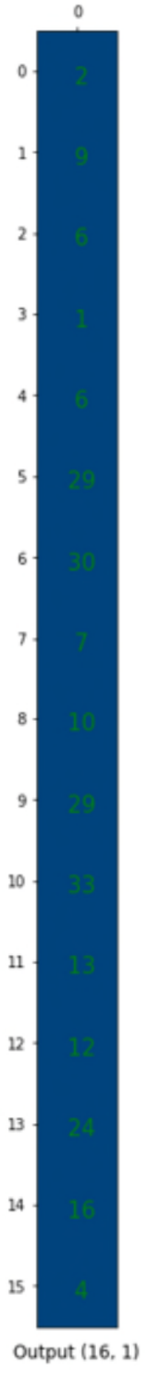

Output (2, 2)

Transposed Convolutions

The point is that with the convolution matrix, you can go from 16 (4x4) to 4 (2x2) because the convolution matrix is 4x16.
Then, if you have a 16x4 matrix, you can go from 4 (2x2) to 16 (4x4)

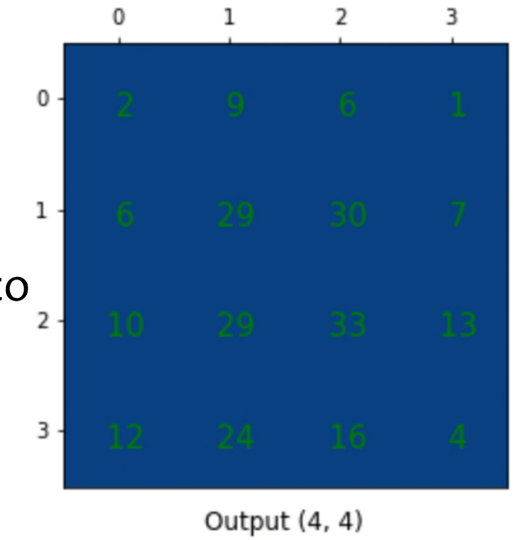
- We transpose the convolution matrix C (4x16) to C^T (16x4).
- We can matrix-multiply C^T (16x4) with a column vector (4x1) to generate an output matrix (16x1)
- The transposed matrix connects 1 value to 9 values in the output.





We have just up-sampled a smaller matrix (2x2) into a larger one (4x4).
The transposed convolution maintains the 1 to 9 relationship because of the way it lays out the weights.

The output can be reshaped into 4x4



The actual weight values in the matrix does not have to come from the original convolution matrix. What's important is that the weight layout is transposed from that of the convolution matrix.

Transposed Convolutions

- Even though it is called the transposed convolution, it does not mean that we take some existing convolution matrix and use the transposed version.

- The main point is that the association between the input and the output is handled in the backward fashion compared with a standard convolution matrix (one-to-many rather than many-to-one association).

One caution: the transposed convolution is the cause of the checkerboard artifacts in generated images. This article* recommends an up-sampling operation (i.e., an interpolation method) followed by a convolution operation to reduce such issues. If your main objective is to generate images without such artifacts, it is worth reading the paper to find out more.

*Deconvolution and Checkerboard Artifacts
https://distill.pub/2016/deconv-checkerboard/

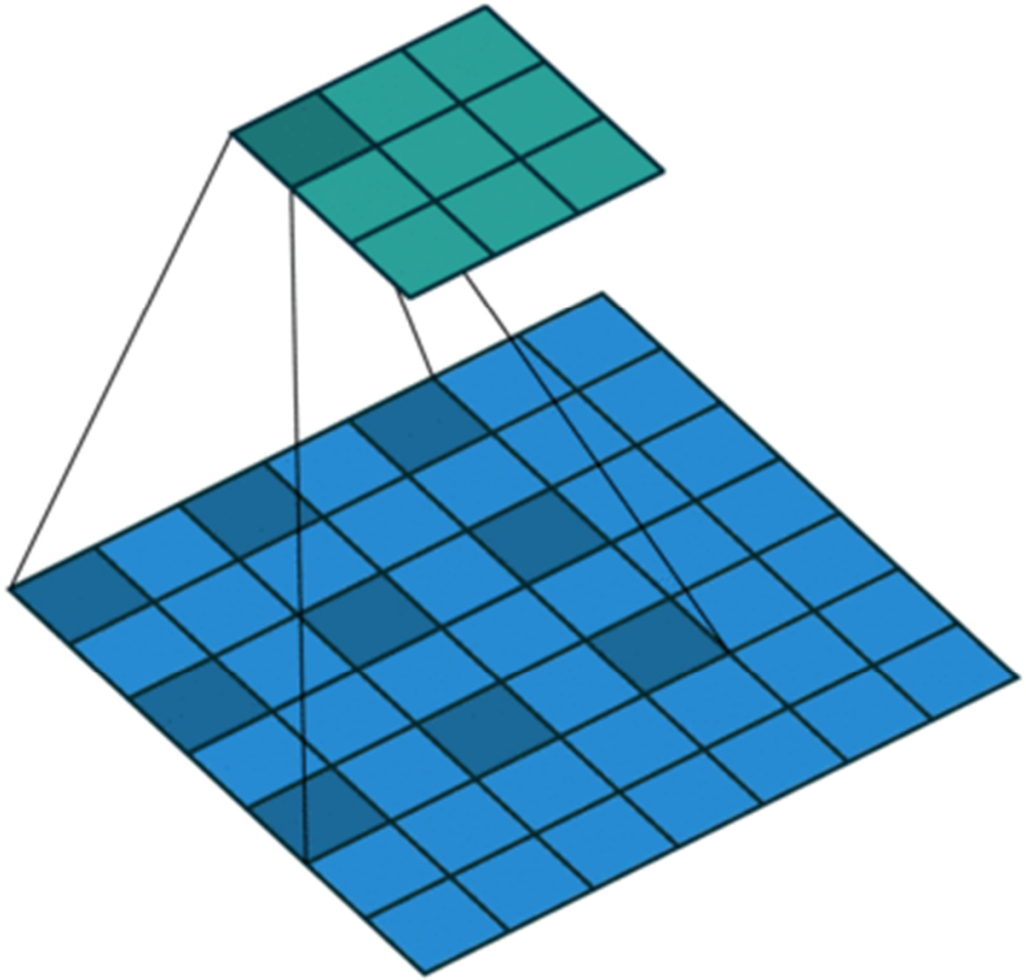
Dilated convolutions

- Dilated convolutions introduce another parameter to convolutional layers called the dilation rate.

- This defines a spacing between the values in a kernel. A 3x3 kernel with a dilation rate of 2 will have the same field of view as a 5x5 kernel, while only using 9 parameters.
- (Imagine taking a 5x5 kernel and deleting every second column and row).

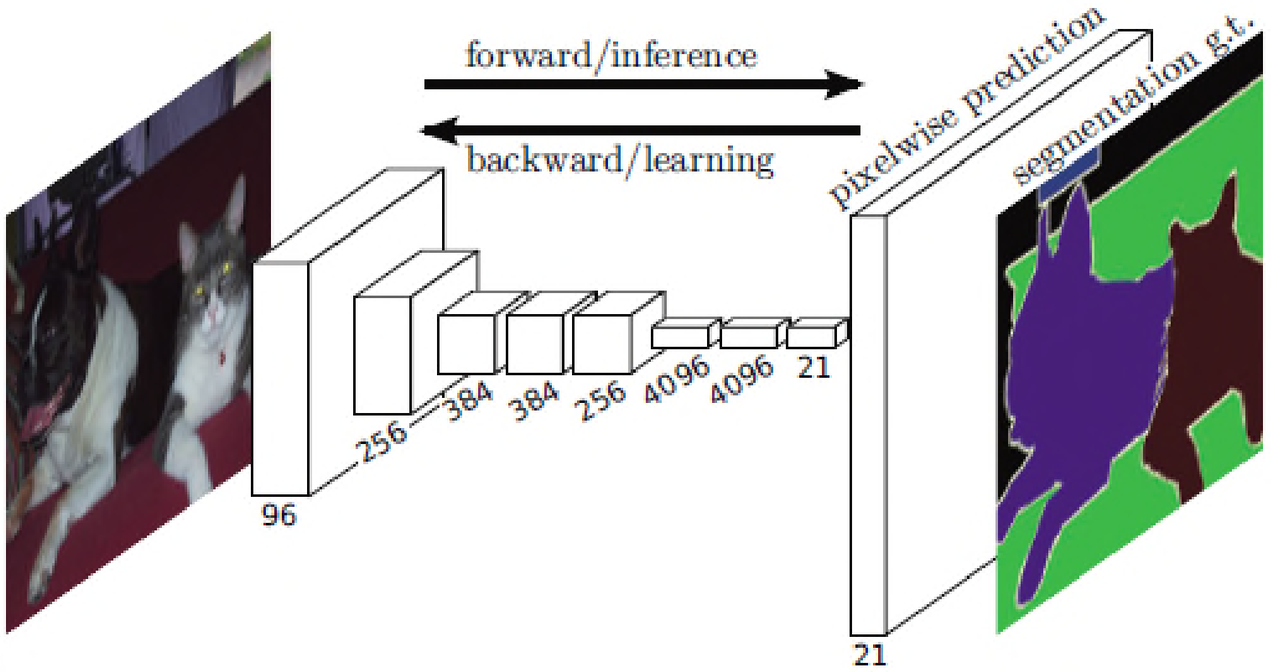
- This delivers a wider field of view at the same computational cost. Dilated convolutions are particularly popular in the field of real-time segmentation.

- Use them if you need a wide field of view and cannot afford multiple convolutions or larger kernels.



2D Atrous convolution using a 3x3 kernel with a dilation rate of 2 and no padding

Back to FCN


forward/inference
backward/learning
pixelwise prediction
segmentation g.t.
96
256
384
384
256
4096
4096
21
21

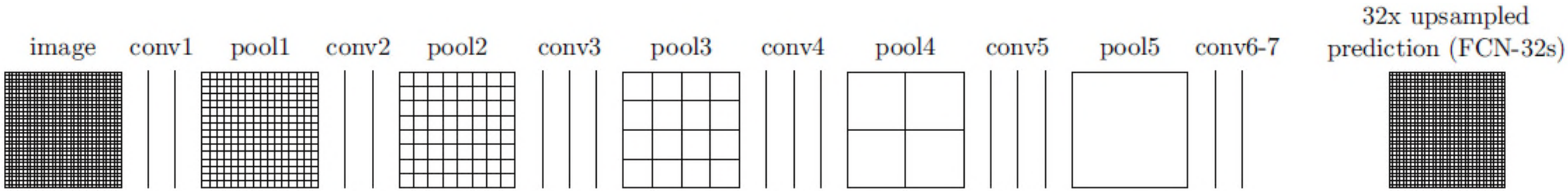
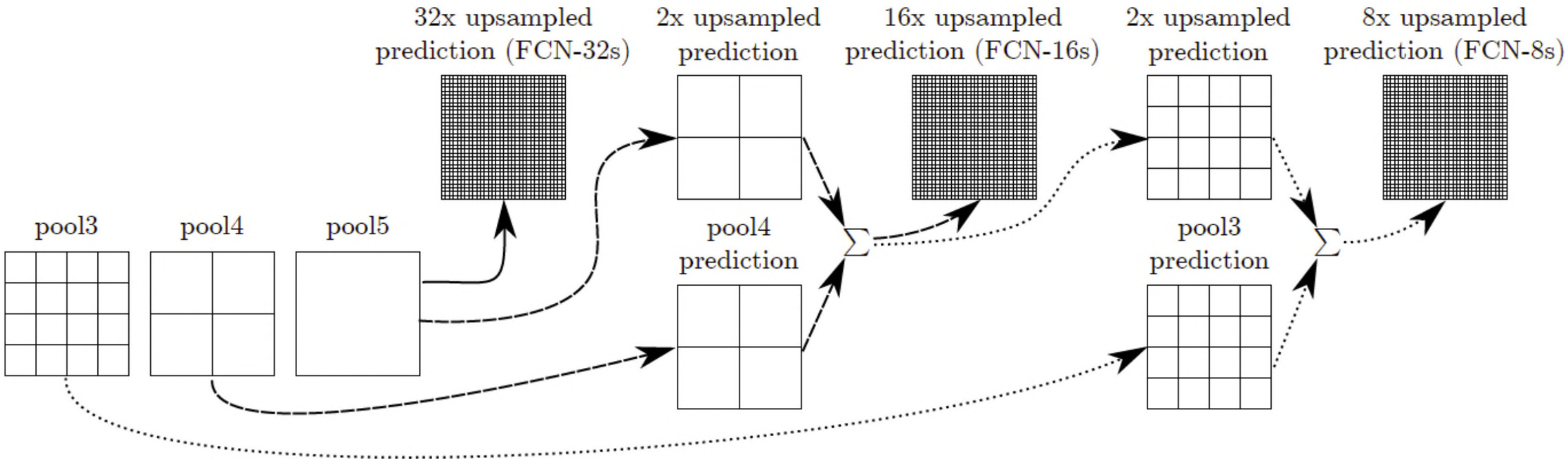


image
conv1
pool1
conv2
pool2
conv3
pool3
conv4
pool4
conv5
pool5
conv6-7
32x upsampled prediction (FCN-32s)

FCNs

- Deep features can be obtained when going deeper, spatial location information is also lost when going deeper.
- That means output from shallower layers have more location information.
- If we combine both, we can enhance the result.
- To combine, we fuse the output (by element-wise addition):



FCNs

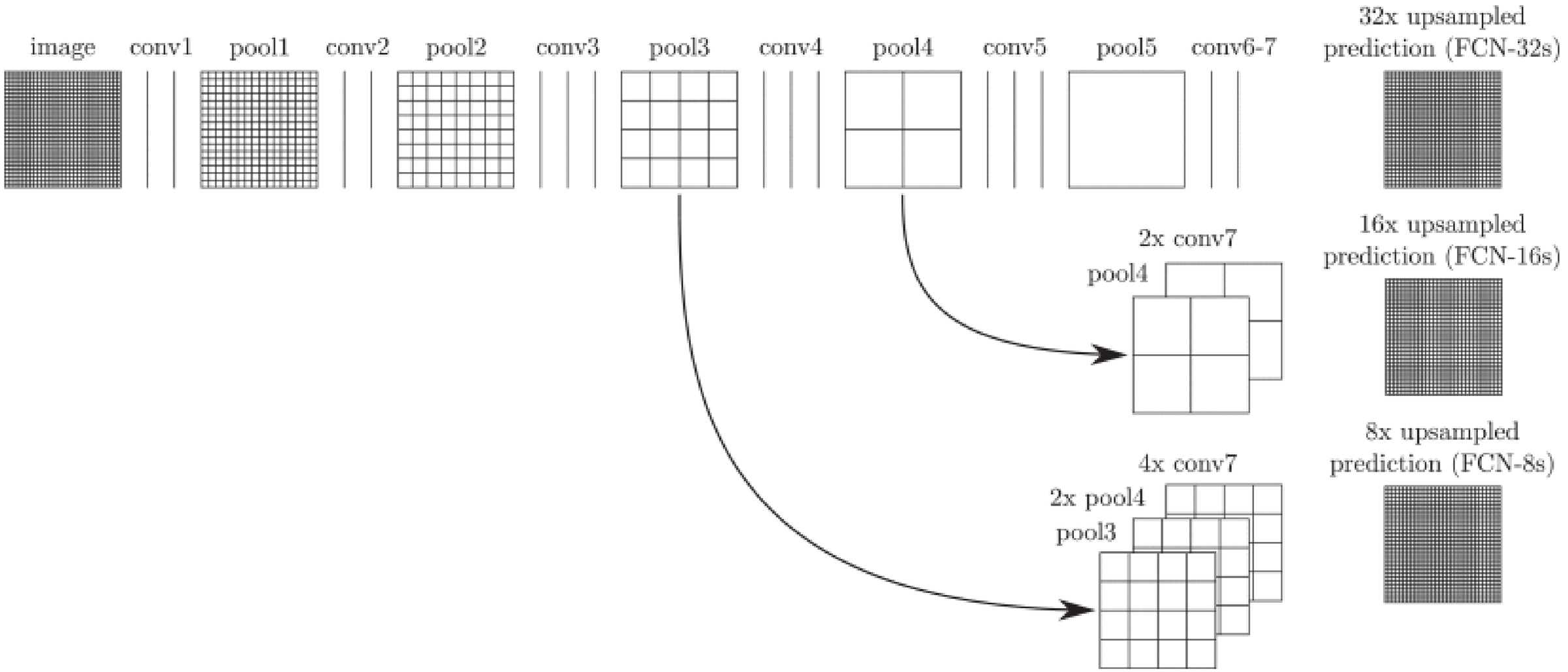
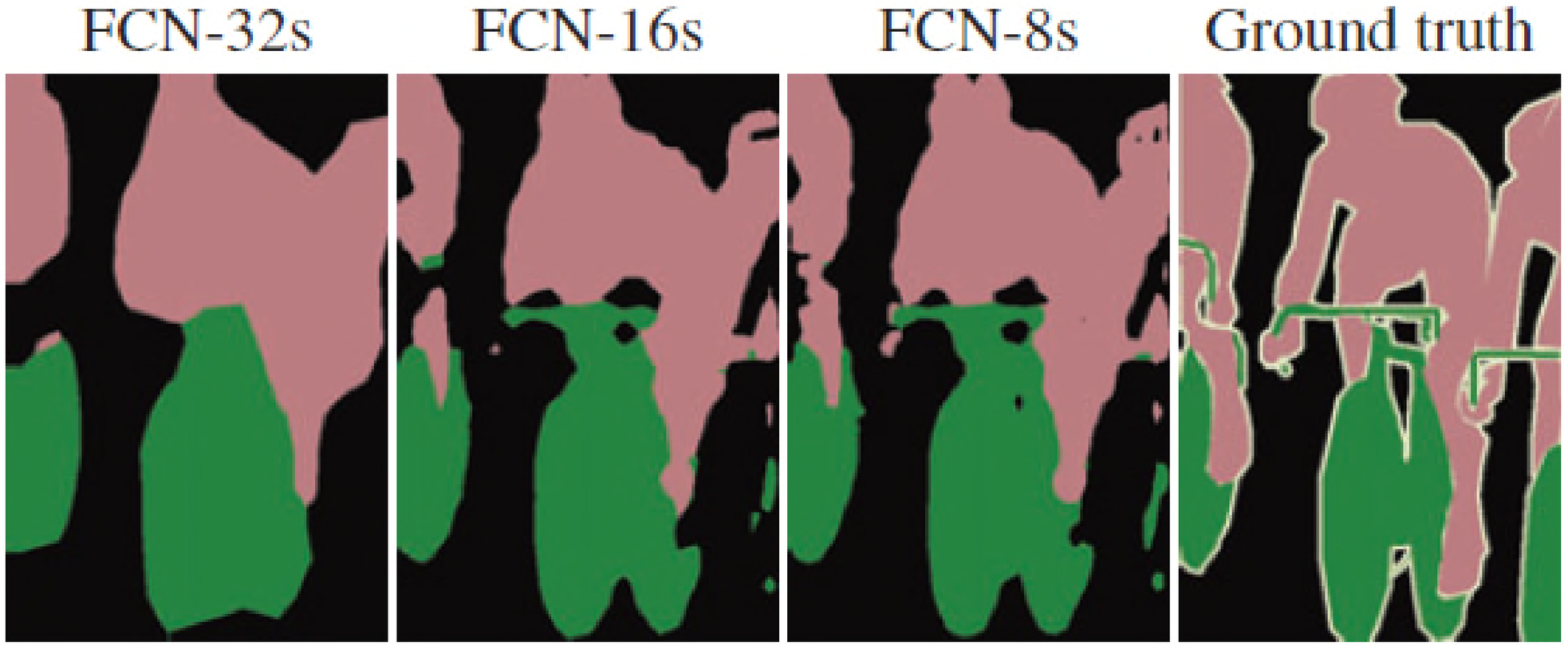


image
conv1
pool1
conv2
pool2
conv3
pool3
conv4
pool4
conv5
pool5
conv6-7
32x upsampled prediction (FCN-32s)
2x conv7
pool4
16x upsampled prediction (FCN-16s)
4x conv7
2x pool4
pool3
8x upsampled prediction (FCN-8s)

Visual Demonstration



Layers in Keras

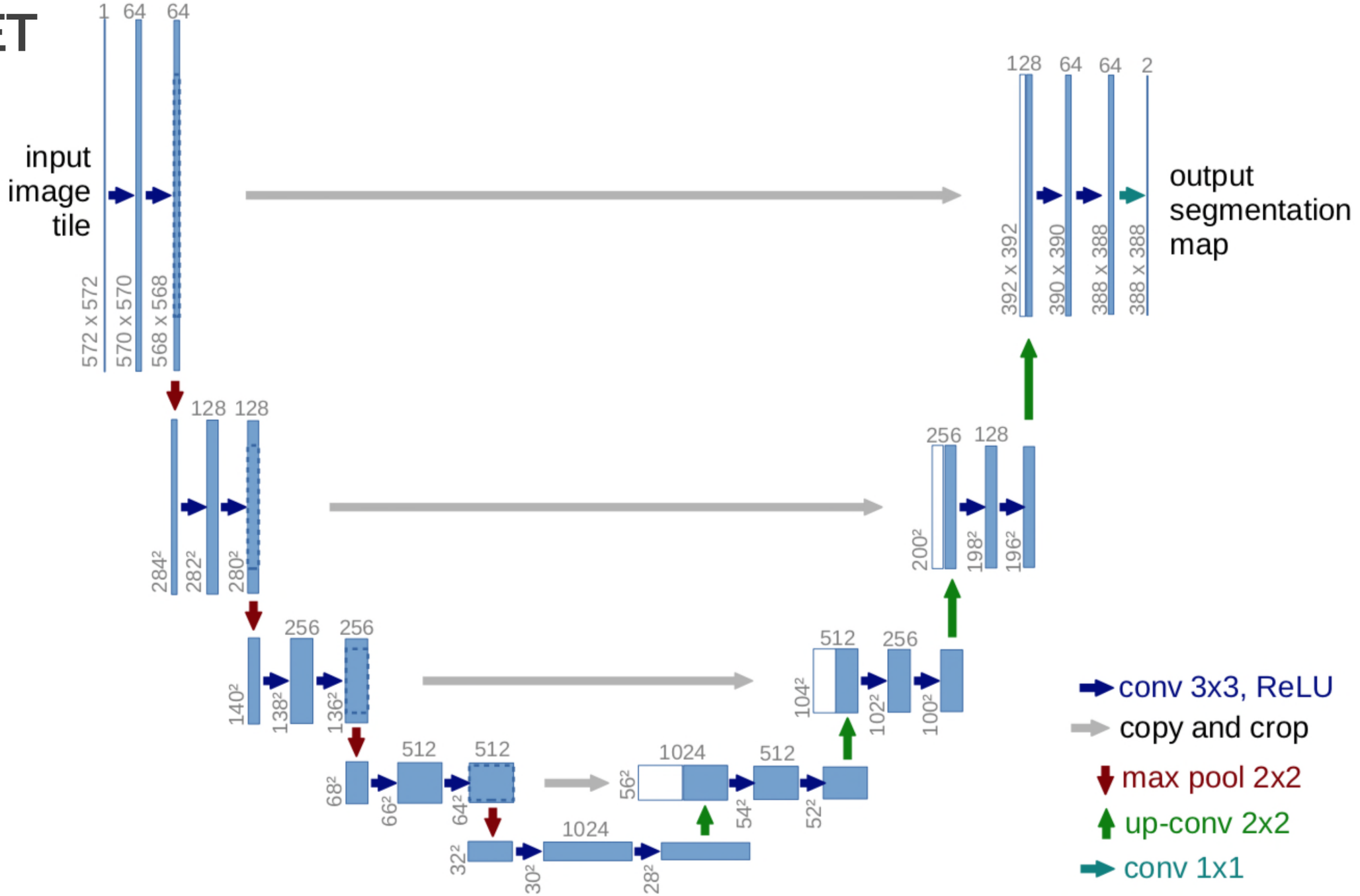
UpSampling2D is just a simple scaling up of the image by using nearest neighbour or bilinear upsampling, so nothing smart.

Advantage is it's cheap.

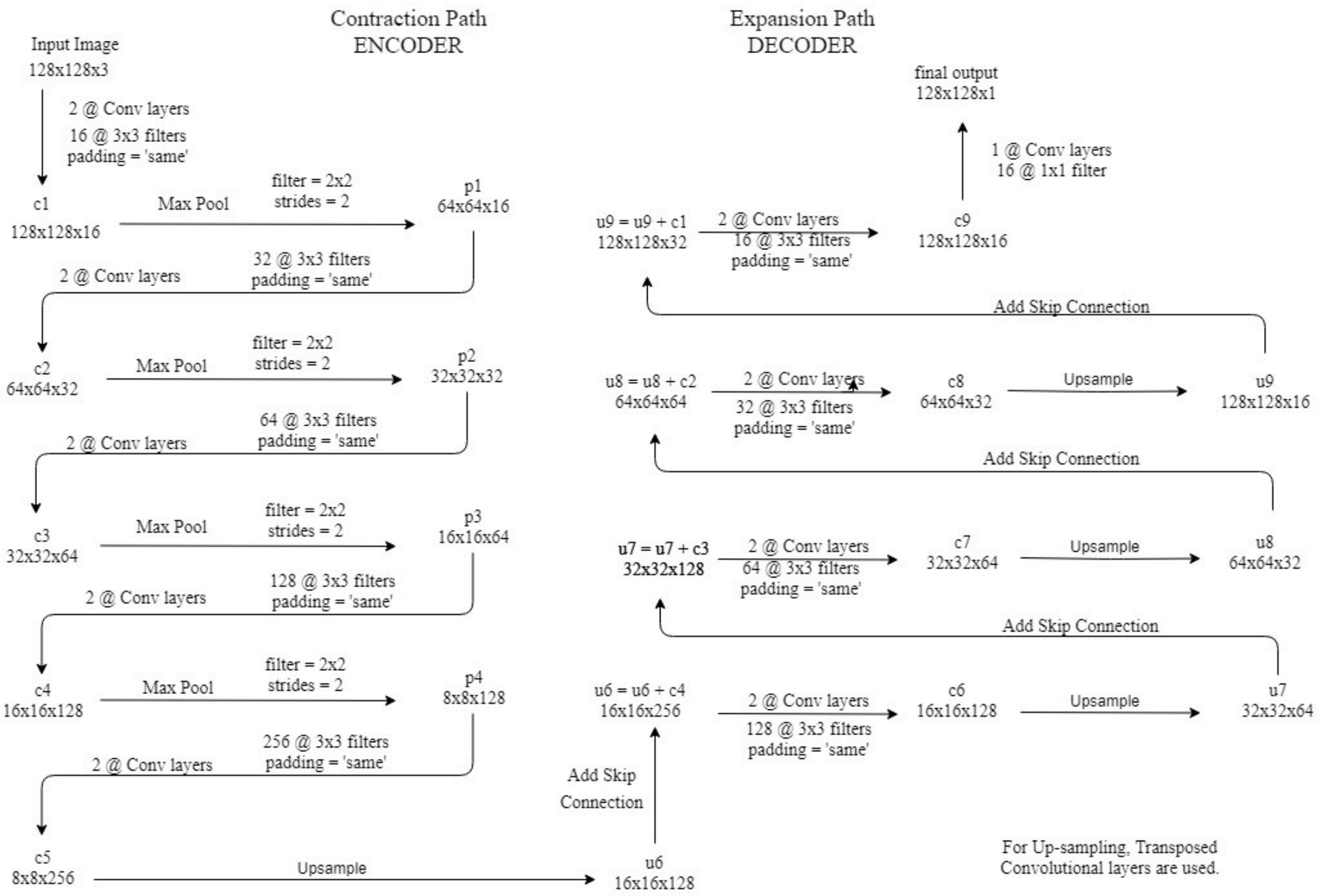
Conv2DTranspose is a convolution operation whose kernel is learnt (just like normal conv2d operation) while training your model

Interesting: https://www.kaggle.com/mpalermo/remove-grideffect-on-generated-images/notebook

UNET


input image tile
572 x 572
570 x 570
568 x 568
1 64 64
128 128
284²
282²
280²
256 256
140²
138²
136²
512 512
68²
66²
64²
1024
32²
30²
28²
1024
512
56²
54²
52²
512 256
104²
102²
100²
256 128
200²
198²
196²
128 64 64 2
392 x 392
390 x 390
388 x 388
388 x 388
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

UNET


Contraction Path
ENCODER
Expansion Path
DECODER
Input Image
128x128x3
2 @ Conv layers
16 @ 3x3 filters
padding = 'same'
c1
128x128x16
Max Pool
filter = 2x2
strides = 2
p1
64x64x16
2 @ Conv layers
32 @ 3x3 filters
padding = 'same'
c2
64x64x32
Max Pool
filter = 2x2
strides = 2
p2
32x32x32
2 @ Conv layers
64 @ 3x3 filters
padding = 'same'
c3
32x32x64
Max Pool
filter = 2x2
strides = 2
p3
16x16x64
2 @ Conv layers
128 @ 3x3 filters
padding = 'same'
c4
16x16x128
Max Pool
filter = 2x2
strides = 2
p4
8x8x128
2 @ Conv layers
256 @ 3x3 filters
padding = 'same'
c5
8x8x256
Upsample
u6
16x16x128
Add Skip
Connection
u6 = u6 + c4
16x16x256
2 @ Conv layers
128 @ 3x3 filters
padding = 'same'
c6
16x16x128
Upsample
u7
32x32x64
Add Skip Connection
u7 = u7 + c3
32x32x128
2 @ Conv layers
64 @ 3x3 filters
padding = 'same'
c7
32x32x64
Upsample
u8
64x64x32
Add Skip Connection
u8 = u8 + c2
64x64x64
2 @ Conv layers
32 @ 3x3 filters
padding = 'same'
c8
64x64x32
Upsample
u9
128x128x16
Add Skip Connection
u9 = u9 + c1
128x128x32
2 @ Conv layers
16 @ 3x3 filters
padding = 'same'
c9
128x128x16
1 @ Conv layers
16 @ 1x1 filter
final output
128x128x1
For Up-sampling, Transposed Convolutional layers are used.
The parameters for each Transpose Convolution are such that, the height and width of the image are doubled while the depth (no. of channels) is halved

UNET

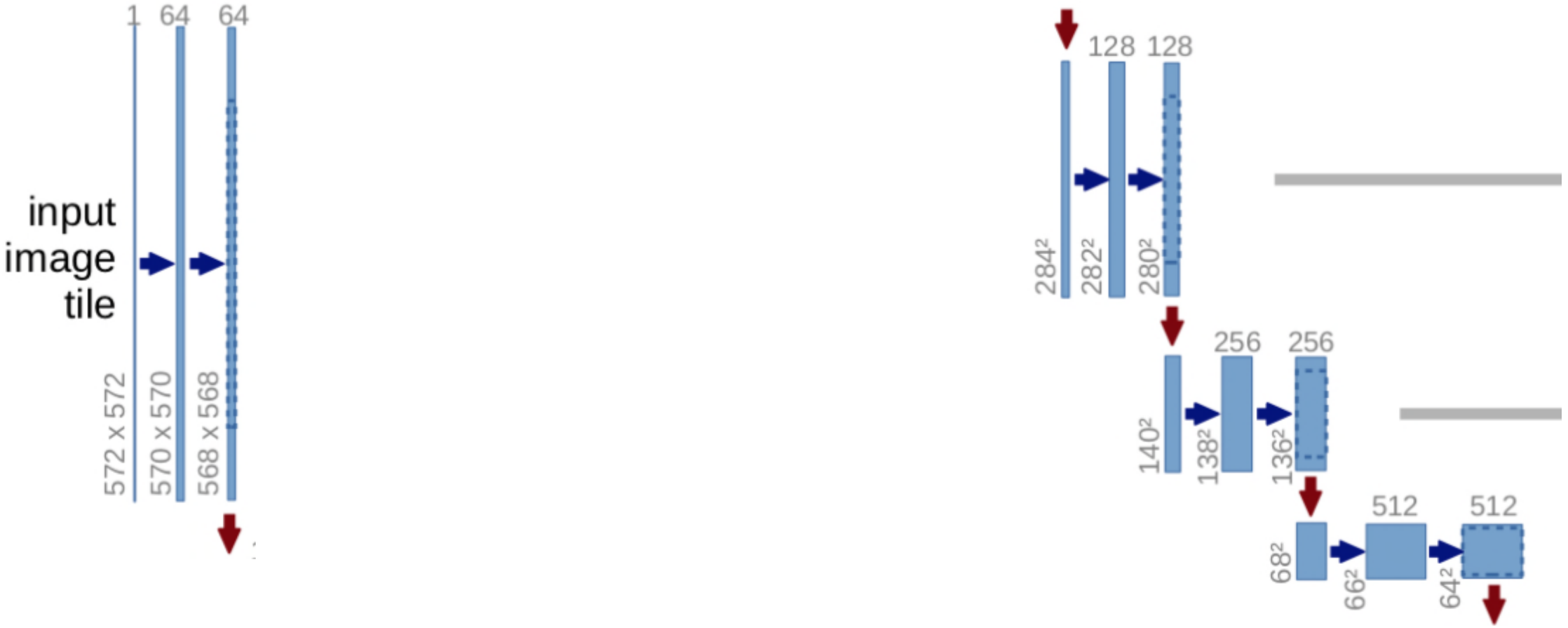
- 2@Conv layers means that two consecutive Convolution Layers are applied
- c1, c2, …. c9 are the output tensors of Convolutional Layers
- p1, p2, p3 and p4 are the output tensors of Max Pooling Layers
- u6, u7, u8 and u9 are the output tensors of up-sampling (transposed convolutional) layers
- The **left hand side is the contraction path (Encoder)** where we apply regular convolutions and max pooling layers.
- In the Encoder, the size of the image gradually reduces while the depth gradually increases. Starting from 128x128x3 to 8x8x256
- This basically means the network learns the "WHAT" information in the image, however it has lost the "WHERE" information
- **The right hand side is the expansion path (Decoder)** where we apply transposed convolutions along with regular convolutions
- In the decoder, the size of the image gradually increases and the depth gradually decreases. Starting from 8x8x256 to 128x128x1
- Intuitively, the Decoder recovers the "WHERE" information (precise localization) by gradually applying up-sampling
- To get better precise locations, at every step of the decoder we use **skip connections** by concatenating the output of the transposed convolution layers with the feature maps from the Encoder at the same level:
  u6 = u6 + c4
  u7 = u7 + c3
  u8 = u8 + c2
  u9 = u9 + c1
  After every concatenation we again apply two consecutive regular convolutions so that the model can learn to assemble a more precise output

U Net

On a high level, we have the following relationship:
Input (128x128x1) => Encoder =>(8x8x256) => Decoder =>Ouput (128x128x1)

Implementation: https://github.com/hlamba28/UNET-TGS

Explanation:
https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

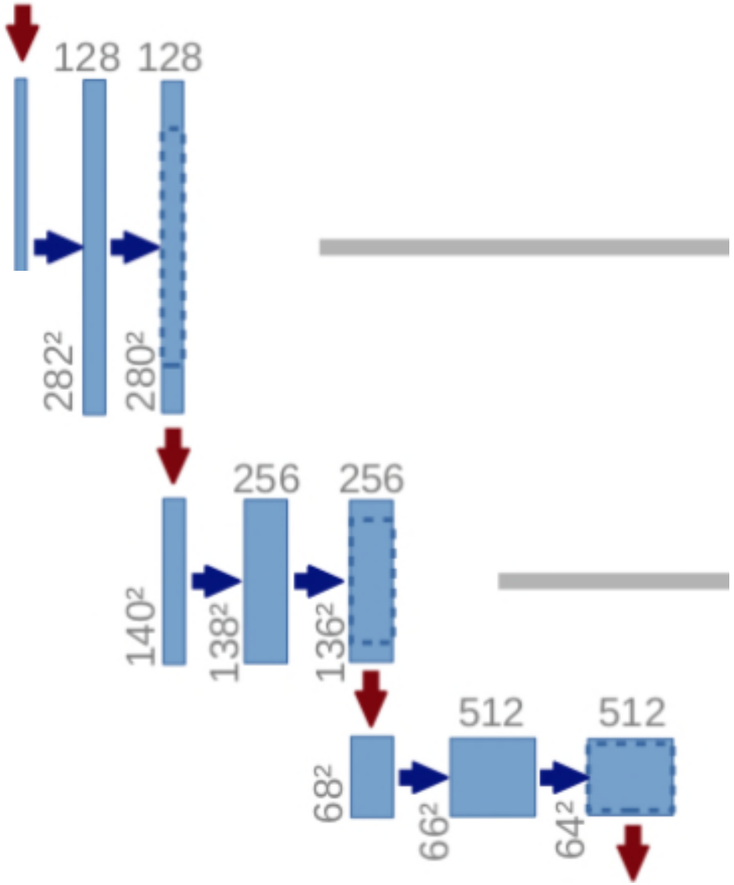



```
conv1 = Conv2D(start_neurons * 1, (3, 3), activation="relu", padding="same")(input_layer)
conv1 = Conv2D(start_neurons * 1, (3, 3), activation="relu", padding="same")(conv1)
pool1 = MaxPooling2D((2, 2))(conv1)
pool1 = Dropout(0.25)(pool1)
```

```
conv2 = Conv2D(start_neurons * 2, (3, 3), activation="relu", padding="same")(pool1)
conv2 = Conv2D(start_neurons * 2, (3, 3), activation="relu", padding="same")(conv2)
pool2 = MaxPooling2D((2, 2))(conv2)
pool2 = Dropout(0.5)(pool2)

conv3 = Conv2D(start_neurons * 4, (3, 3), activation="relu", padding="same")(pool2)
conv3 = Conv2D(start_neurons * 4, (3, 3), activation="relu", padding="same")(conv3)
pool3 = MaxPooling2D((2, 2))(conv3)
pool3 = Dropout(0.5)(pool3)

conv4 = Conv2D(start_neurons * 8, (3, 3), activation="relu", padding="same")(pool3)
conv4 = Conv2D(start_neurons * 8, (3, 3), activation="relu", padding="same")(conv4)
pool4 = MaxPooling2D((2, 2))(conv4)
pool4 = Dropout(0.5)(pool4)
```



Reading Material

- **Papers:**
  - **A guide to convolution arithmetic for deep learning by** Vincent Dumoulin, Francesco Visin at https://arxiv.org/abs/1603.07285

  - [2015 CVPR] [FCN]
    Fully Convolutional Networks for Semantic Segmentation
  - [2017 TPAMI] [FCN]
    Fully Convolutional Networks for Semantic Segmentation

  - **Understanding transposed convolutions**
  - https://www.machinecurve.com/index.php/2019/09/29/understanding-transposed-convolutions/

Reading Material

- Understanding UNET : Implementation
- https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

- Line by Line Explanation
- https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5

Acknowledgments

Various contents in this presentation have been taken from different books, lecture notes, and the web. These solely belong to their owners, and are here used only for clarifying various educational concepts. Any copyright infringement is *not* intended