Introduction to Deep Learning for Computer Vision

Adhyayan '23 - ACA Summer School
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

Lecture 8

Course Logistics

- **Quiz 2** is scheduled for **tomorrow**: **Thursday (22.06.2023)**
- Time: **Thursday 9 PM to Friday 9 AM. (12 hours)**
- Rules: **Same as Quiz 1**
- Syllabus: Upto today's Lecture (**Lecture 8**).

Object Detection!

Object Classification v Object Detection

Classification

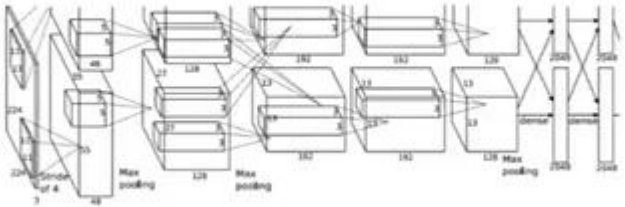
Predicted Output: <Class Name>

Detection

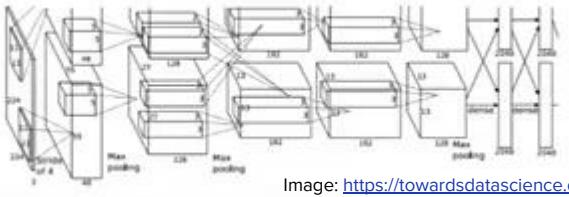
Predicted Output: (<Class Name>, x_1, y_1, x_2,y_2)

Challenges in Object Detection

- An image might contain more than one object.
- We don't know how many!
- Hence, the output layer is not fixed.
- This is where vanilla CNN is limited.



CAT: (x, y, w, h)

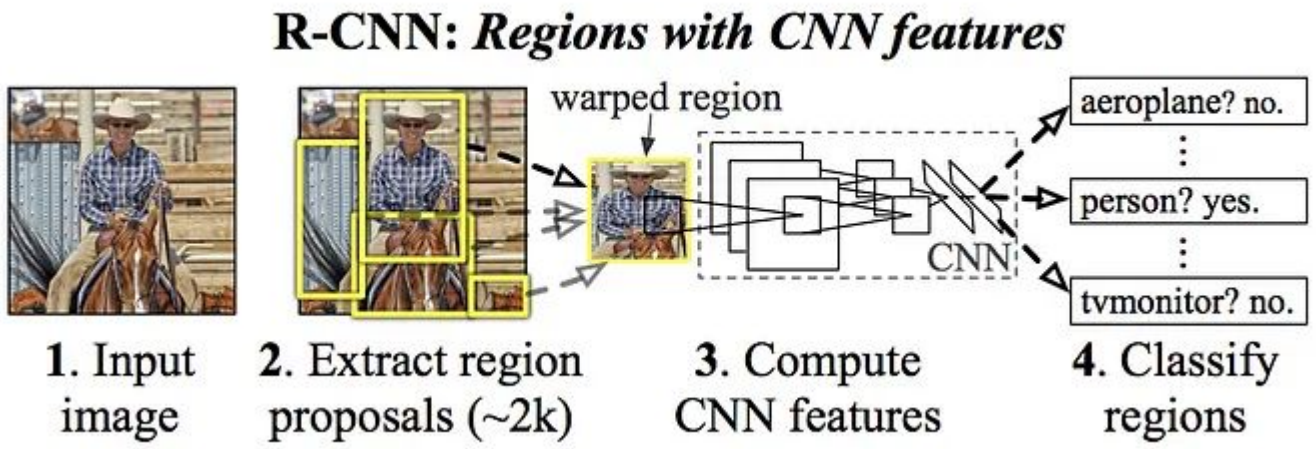


DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
....

Image: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

R-CNN

- Use selective search algorithm to extract 2000 regions from image.
- These 2000 region proposals are warped and fed into a CNN.
- CNN outputs a 4096-length vector that is fed into an SVM.
- Also predicts bounding boxes.



R-CNN

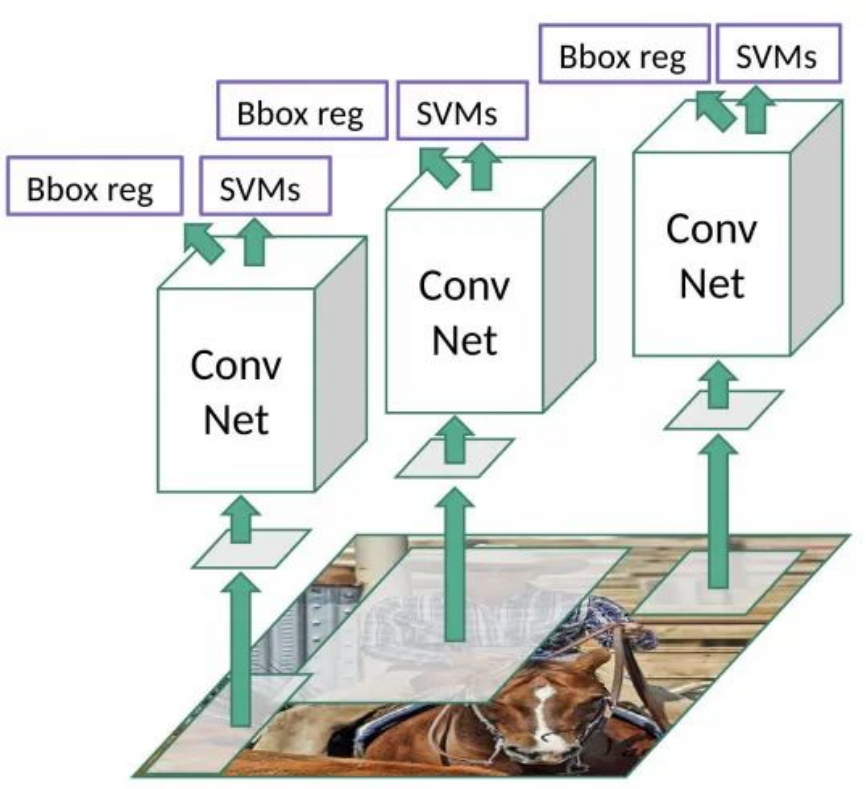
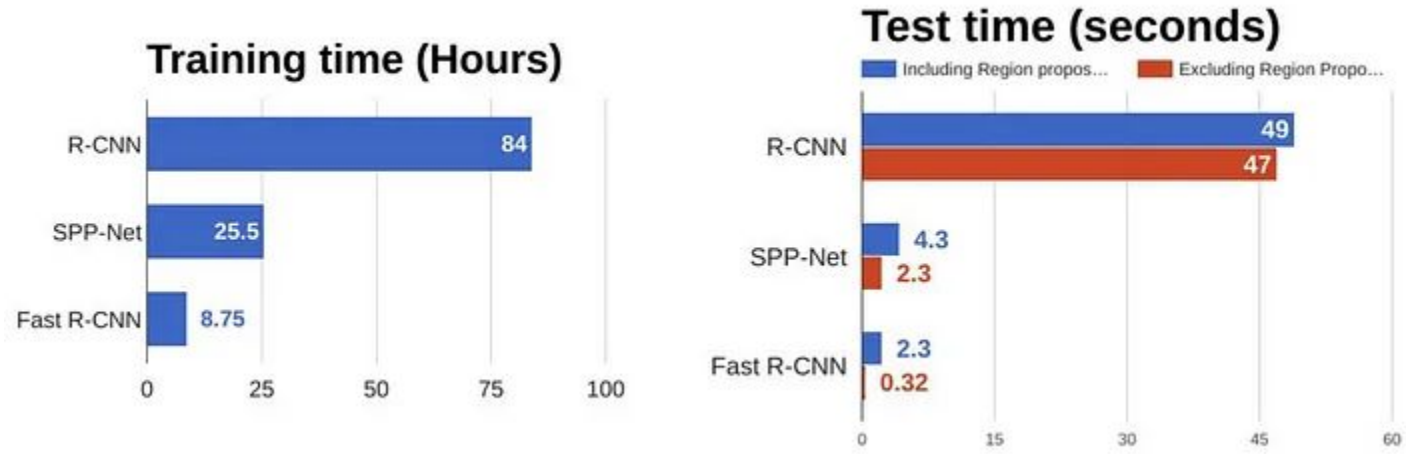


Image: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

Problems with R-CNN

- It still takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.
- It cannot be implemented real time as it takes around 47 seconds for each test image.
- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

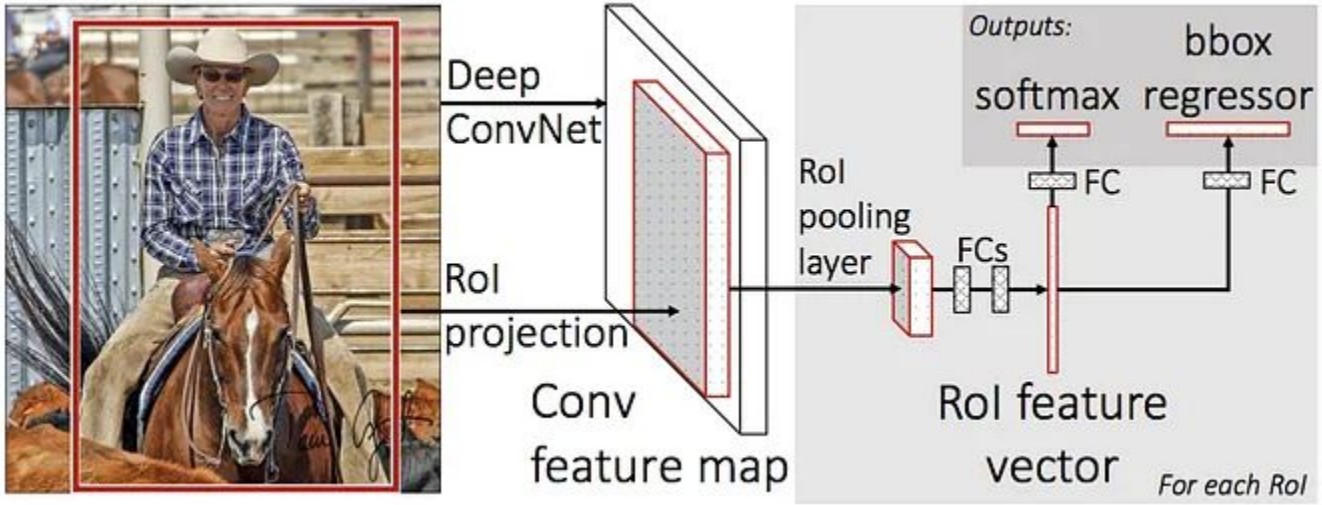
Comparison with R-CNN



- The reason “Fast R-CNN” is faster than R-CNN is because you don’t have to feed 2000 region proposals to the convolutional neural network every time.
- Instead, the convolution operation is done only once per image and a feature map is generated from it.

Image: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

Fast R-CNN



- Instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map.
- From the convolutional feature map, we identify the region of proposals and the rest is similar to R-CNN.

Image: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

Faster R-CNN

- Got rid of the selective search algorithm.
- Region Proposals will now be “learned”.
- A separate network will predict region proposals.
- Faster R-CNN achieved real-time object detection!

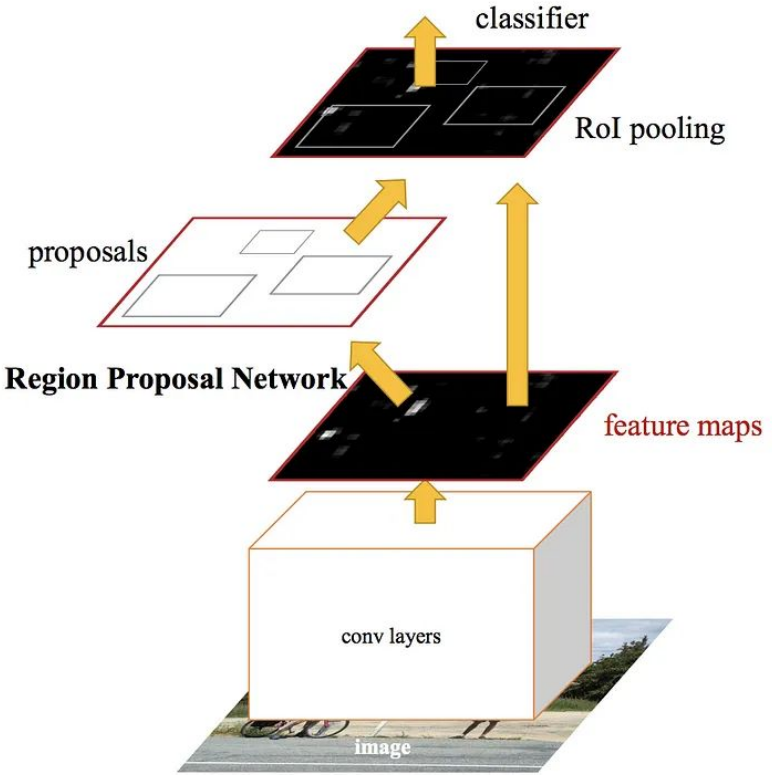


Image: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

Comparison with R-CNN and Fast R-CNN

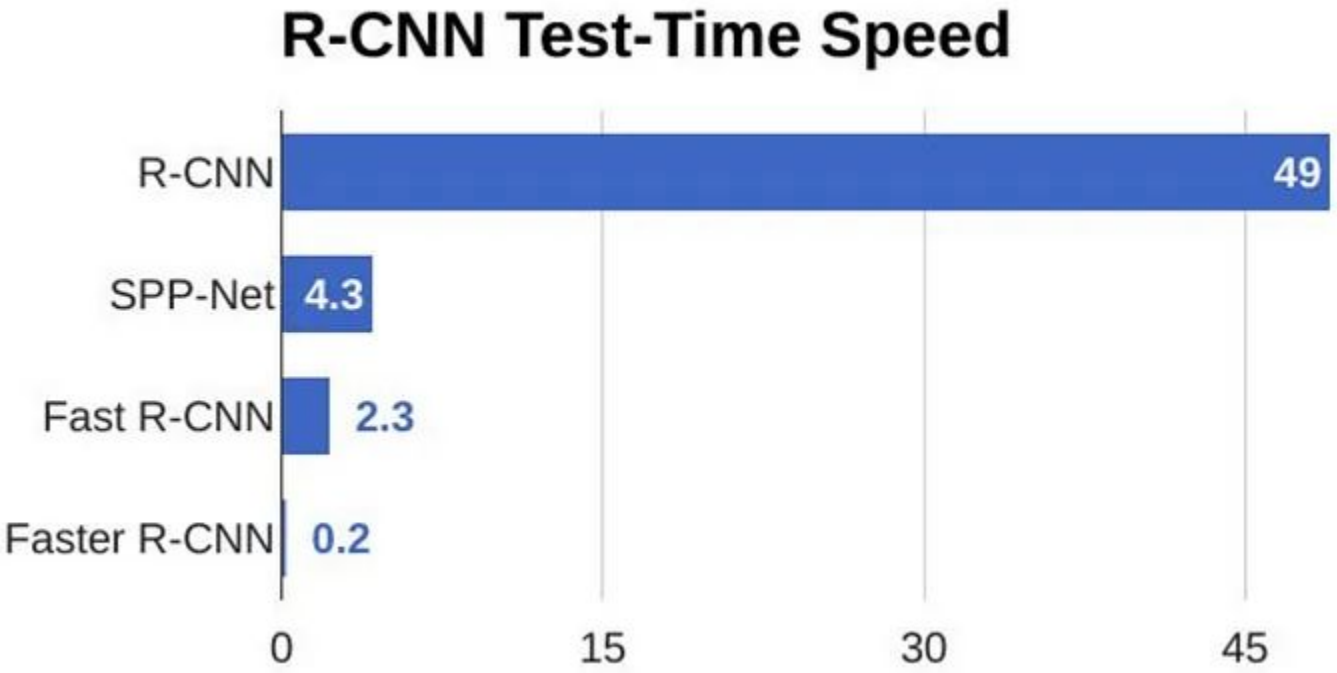


Image: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

YOLO - You Only Look Once!

- One CNN predicts bounding boxes and class probabilities.
- Previous methods looked at parts of the image with high probabilities of containing the object.
- Yolo looks at the whole image just once speeding up the process 10 times!

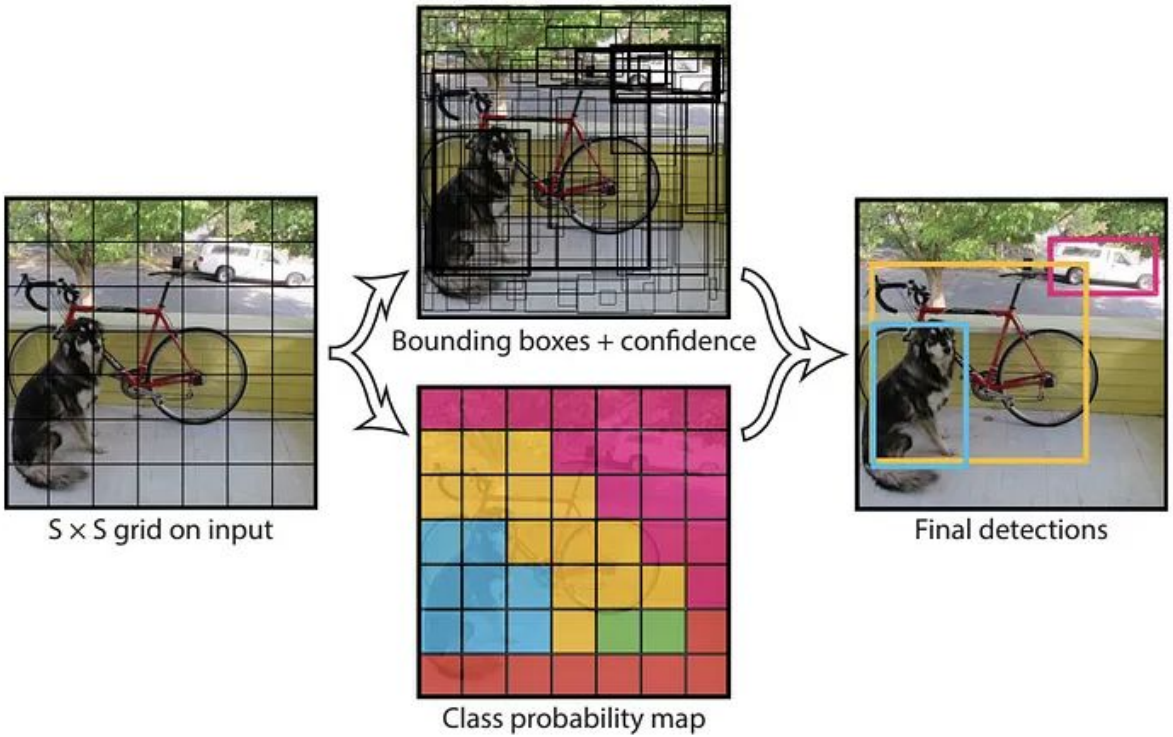


Image: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

YOLO - How does it work?

- Split an image into SxS grid.
- Take m bounding boxes within each grid.
- For each bounding box, output:
  - Class Probability
  - Offset values of bounding box
- Select bounding boxes having the class probability above a threshold.

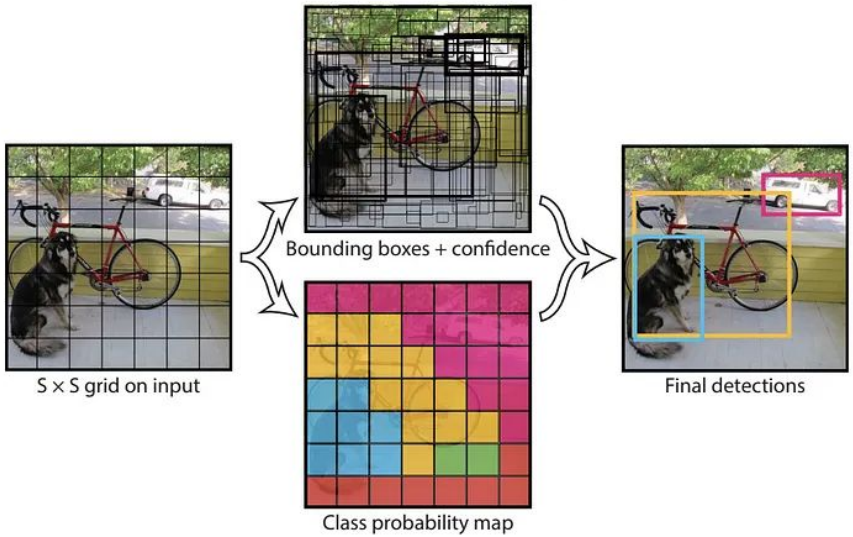
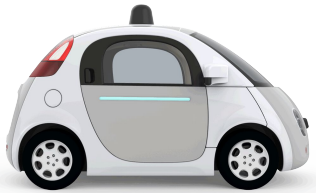


Image: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

Comparison of YOLO

	Pascal 2007 mAP	Speed	
DPM v5	33.7	.07 FPS	14 s/img
R-CNN	66.0	.05 FPS	20 s/img



⅓ Mile, 1760 feet

Comparison of YOLO

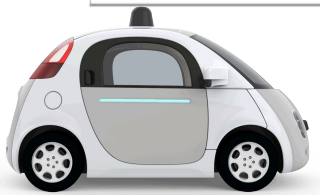
	Pascal 2007 mAP	Speed	
DPM v5	33.7	.07 FPS	14 s/img
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img

Comparison of YOLO

	Pascal 2007 mAP	Speed	
DPM v5	33.7	.07 FPS	14 s/img
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img
Faster R-CNN	73.2	7 FPS	140 ms/img

Comparison of YOLO

	Pascal 2007 mAP	Speed	
DPM v5	33.7	.07 FPS	14 s/img
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img
Faster R-CNN	73.2	7 FPS	140 ms/img
YOLO	69.0	45 FPS	22 ms/img

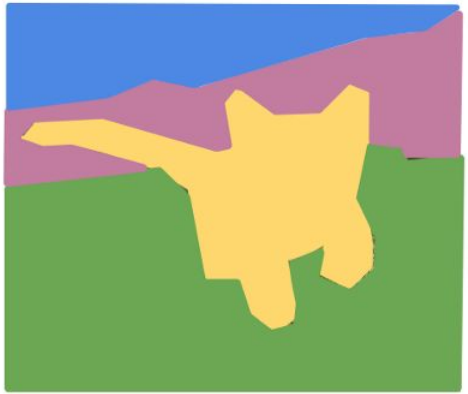


2 feet

Segmentation!

Computer Vision Tasks

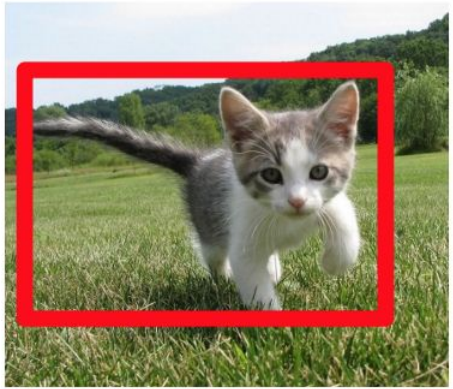
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

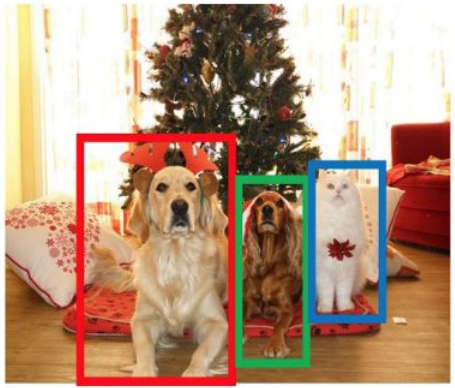
Classification + Localization



CAT

Single Object

Object Detection



DOG, DOG, CAT

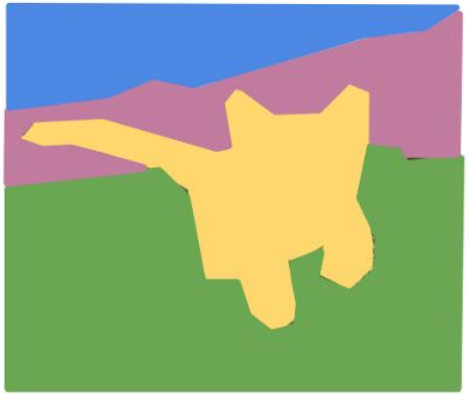
Instance Segmentation

DOG, DOG, CAT

Multiple Object

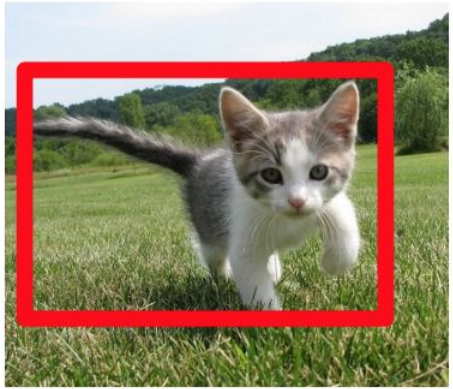
Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Semantic Segmentation



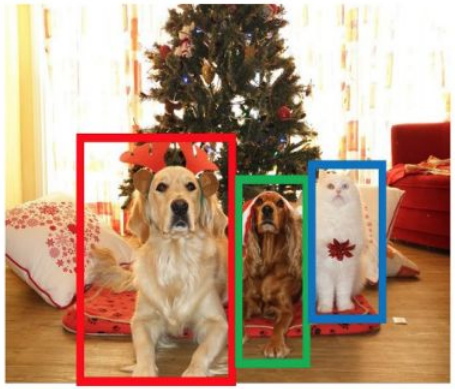
GRASS, CAT, TREE, SKY

No objects, just pixels



CAT

Single Object



DOG, DOG, CAT

DOG, DOG, CAT

Multiple Object

Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Semantic Segmentation

- Label each pixel in the image with a category label.
- Don’t differentiate instances, only care about pixels.

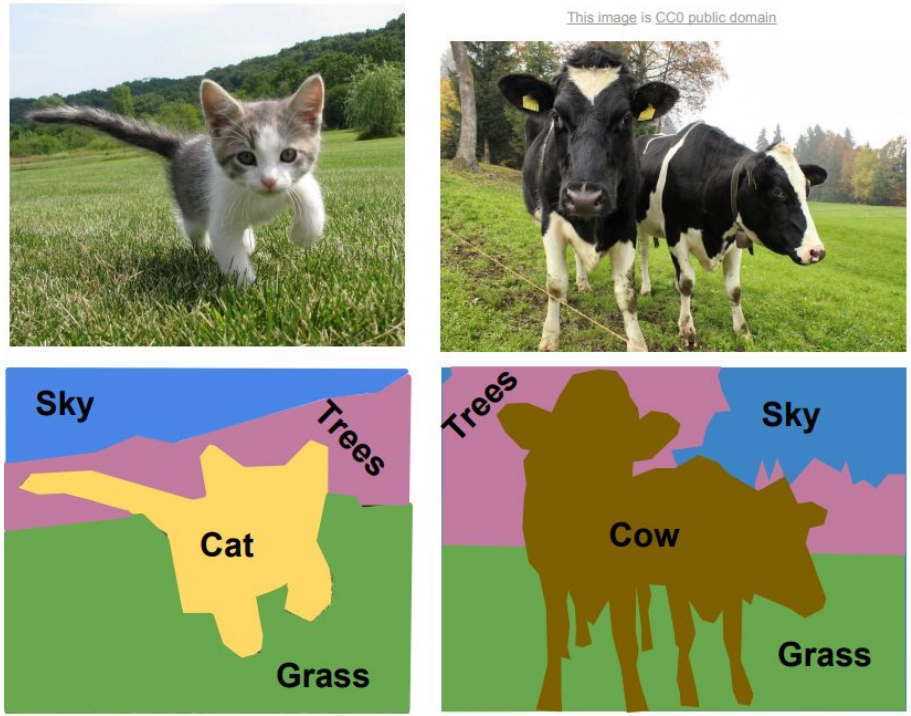


Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Idea: Sliding Window

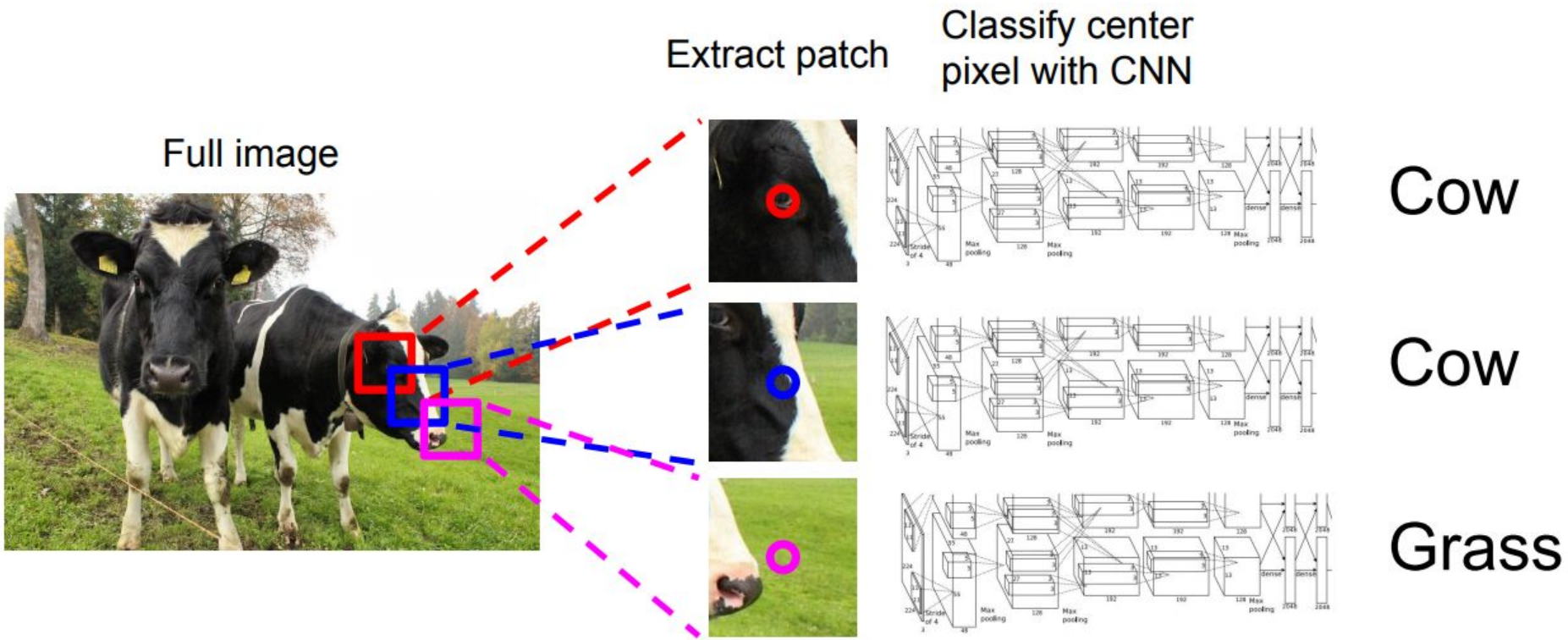
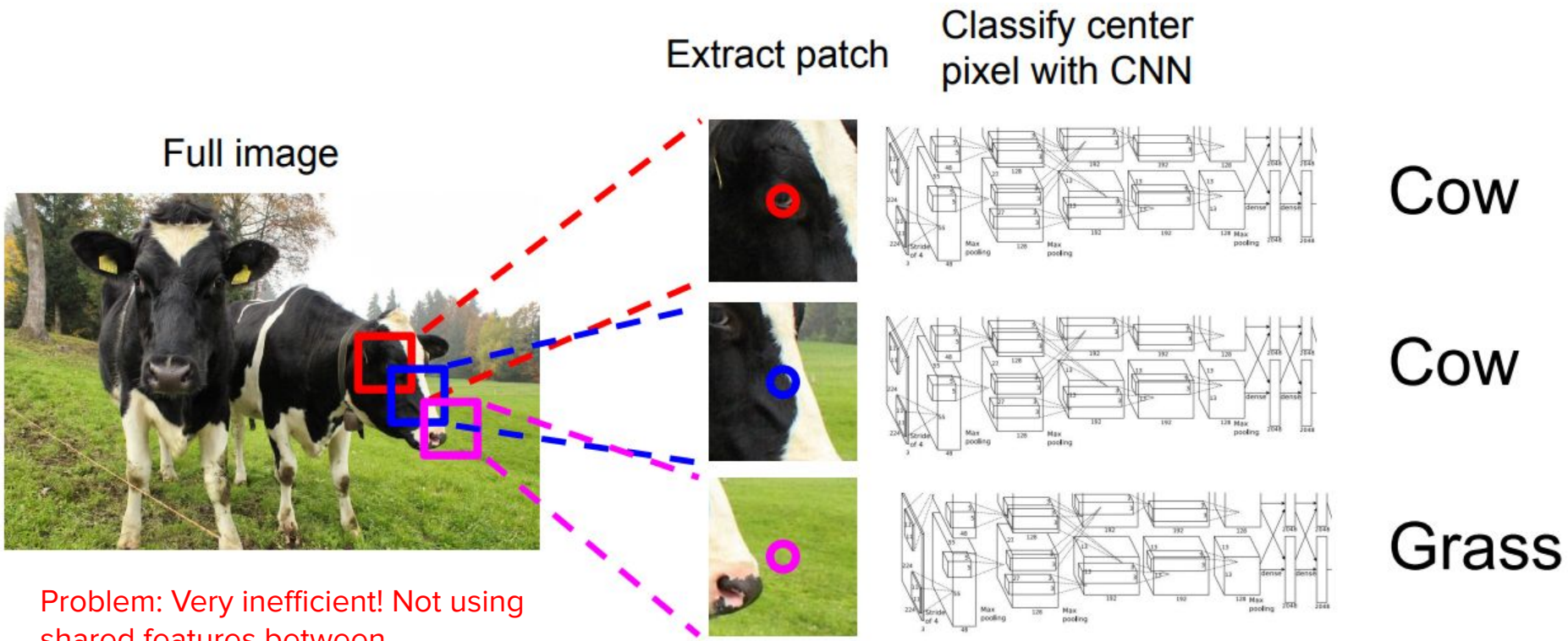


Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Idea: Sliding Window



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

Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Idea: Fully Convolutional

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

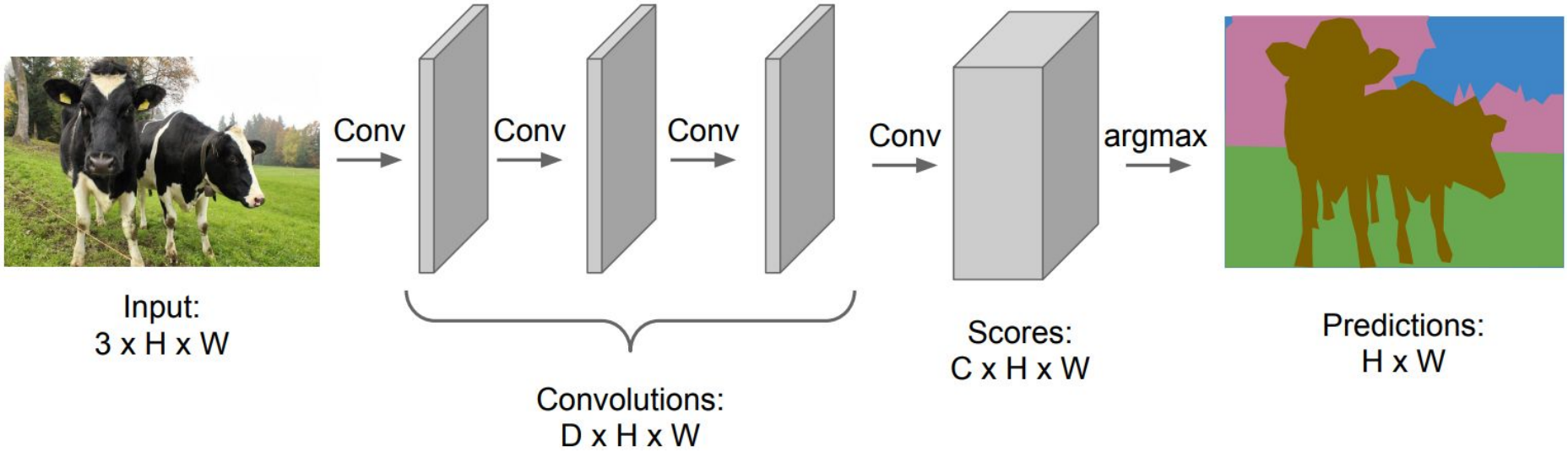
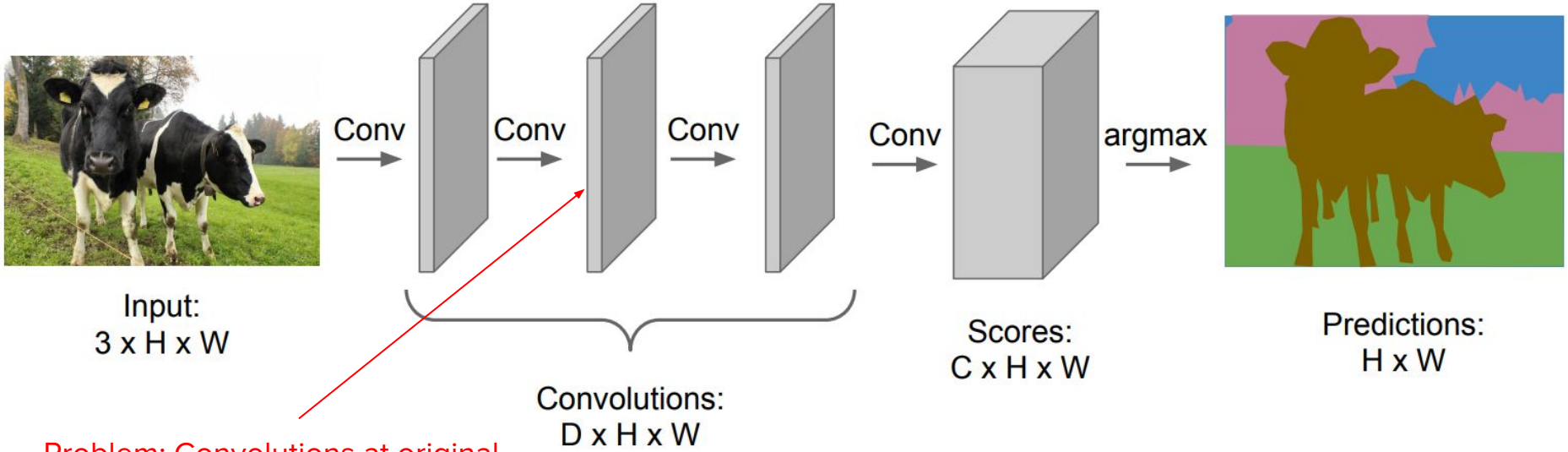


Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Idea: Fully Convolutional

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



Problem: Convolutions at original image resolution will be very expensive!

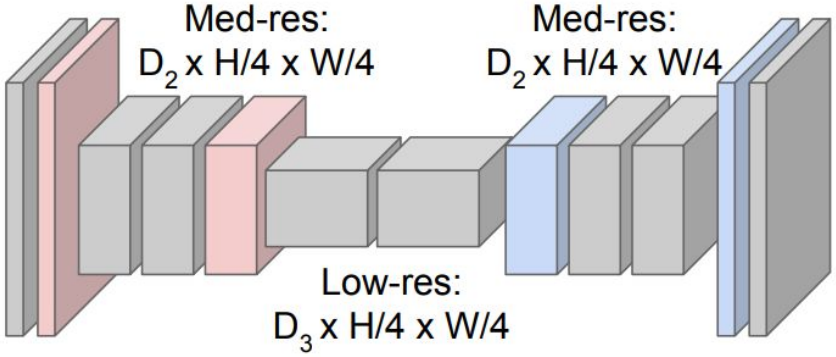
Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Idea: Fully Convolutional

Downsampling:
Pooling, Strided Convolution

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

Input:
3 x H x W



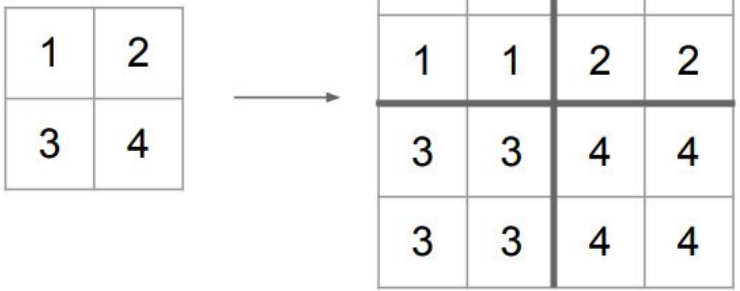
Upsampling:
???

Predictions:
H x W

Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

In-Network Upsampling: Unpooling

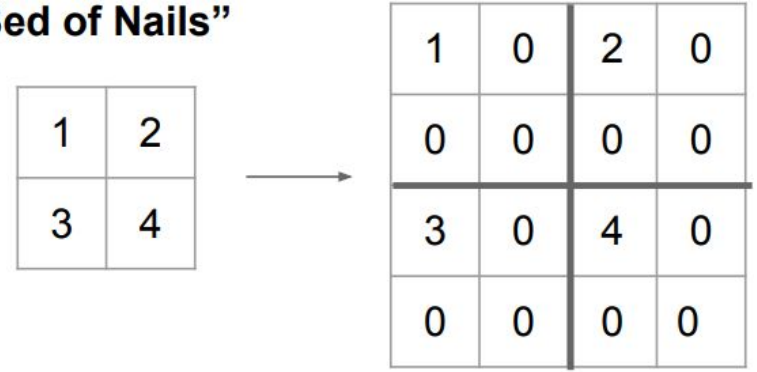
Nearest Neighbor



Input: 2 x 2

Output: 4 x 4

“Bed of Nails”



Input: 2 x 2

Output: 4 x 4

Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

In-Network Upsampling: Max Unpooling

Max Pooling
Remember which element was max!

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

Input: 4 x 4

5	6
7	8

Output: 2 x 2

Rest of the network

Max Unpooling
Use positions from pooling layer

1	2
3	4

Input: 2 x 2

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

Output: 4 x 4

Corresponding pairs of downsampling and upsampling layers

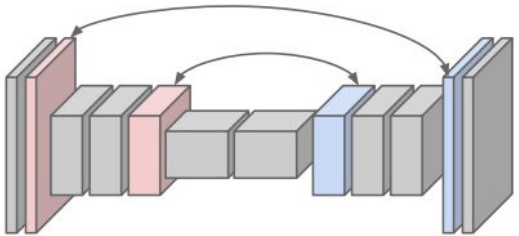
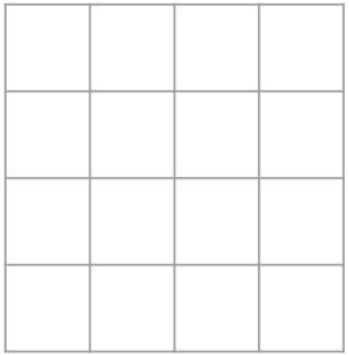


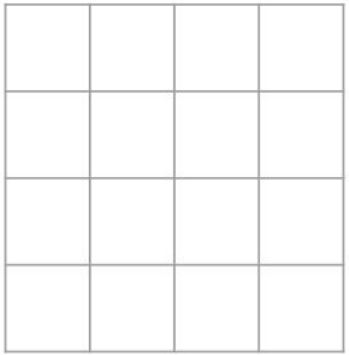
Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Learnable Upsampling: Transpose Convolution

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



Input: 4 x 4



Output: 4 x 4

Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Learnable Upsampling: Transpose Convolution

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

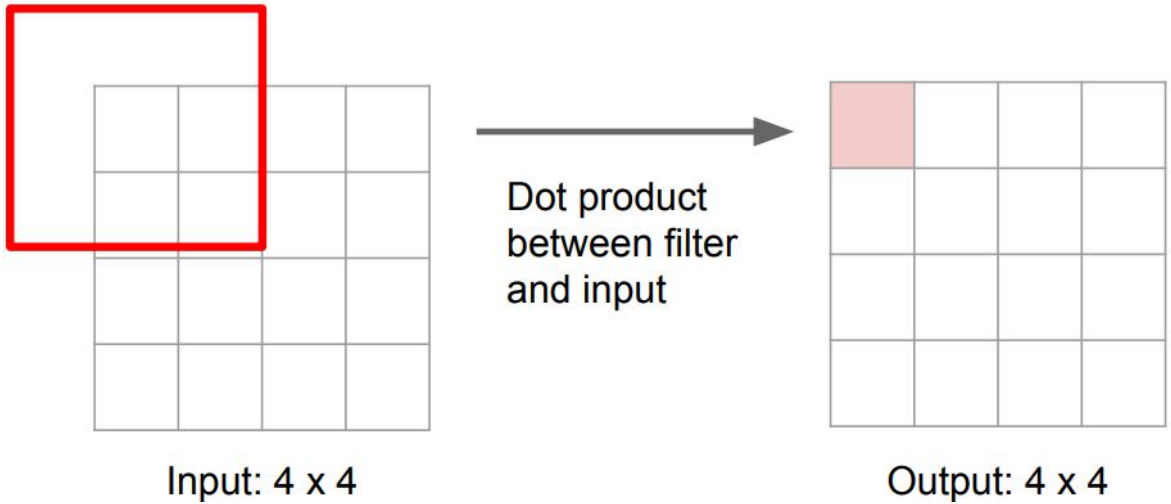


Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Learnable Upsampling: Transpose Convolution

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

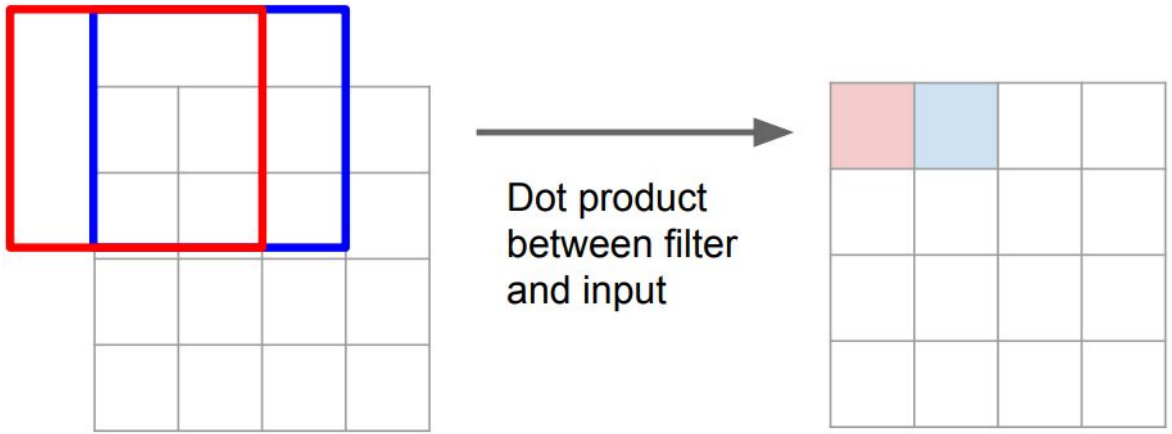
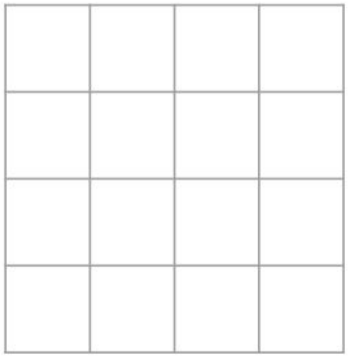


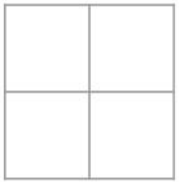
Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Learnable Upsampling: Transpose Convolution

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



Input: 4 x 4



Output: 2 x 2

Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Learnable Upsampling: Transpose Convolution

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

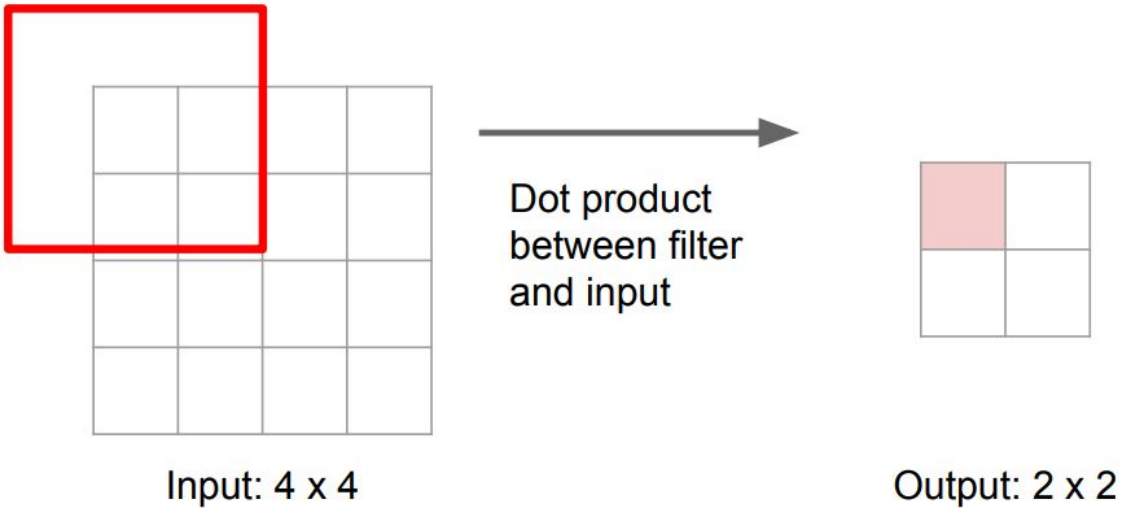
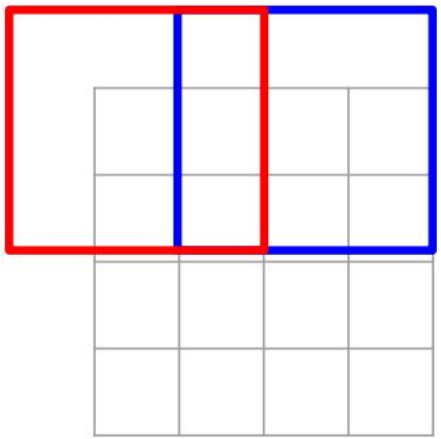


Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

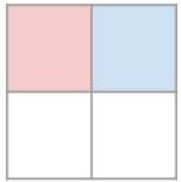
Learnable Upsampling: Transpose Convolution

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



Input: 4 x 4

Dot product between filter and input



Output: 2 x 2

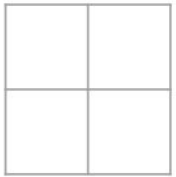
Filter moves 2 pixels in the input for every one pixel in the output

Stride gives ratio between movement in input and output

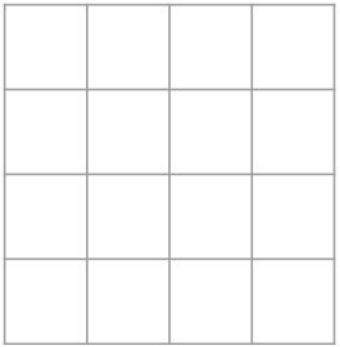
Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Learnable Upsampling: Transpose Convolution

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



Input: 2 x 2



Output: 4 x 4

Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Learnable Upsampling: Transpose Convolution

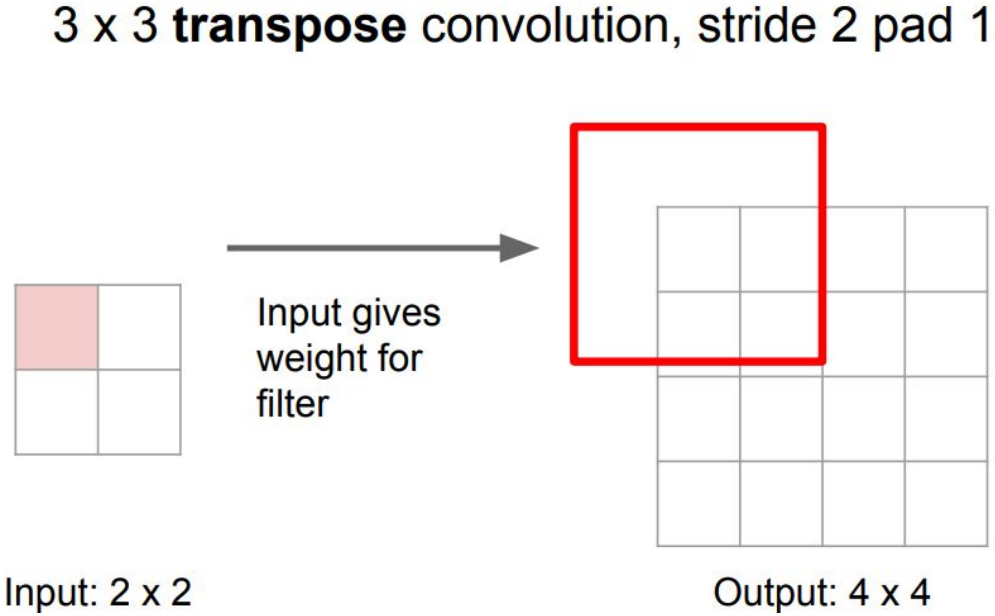


Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Learnable Upsampling: Transpose Convolution

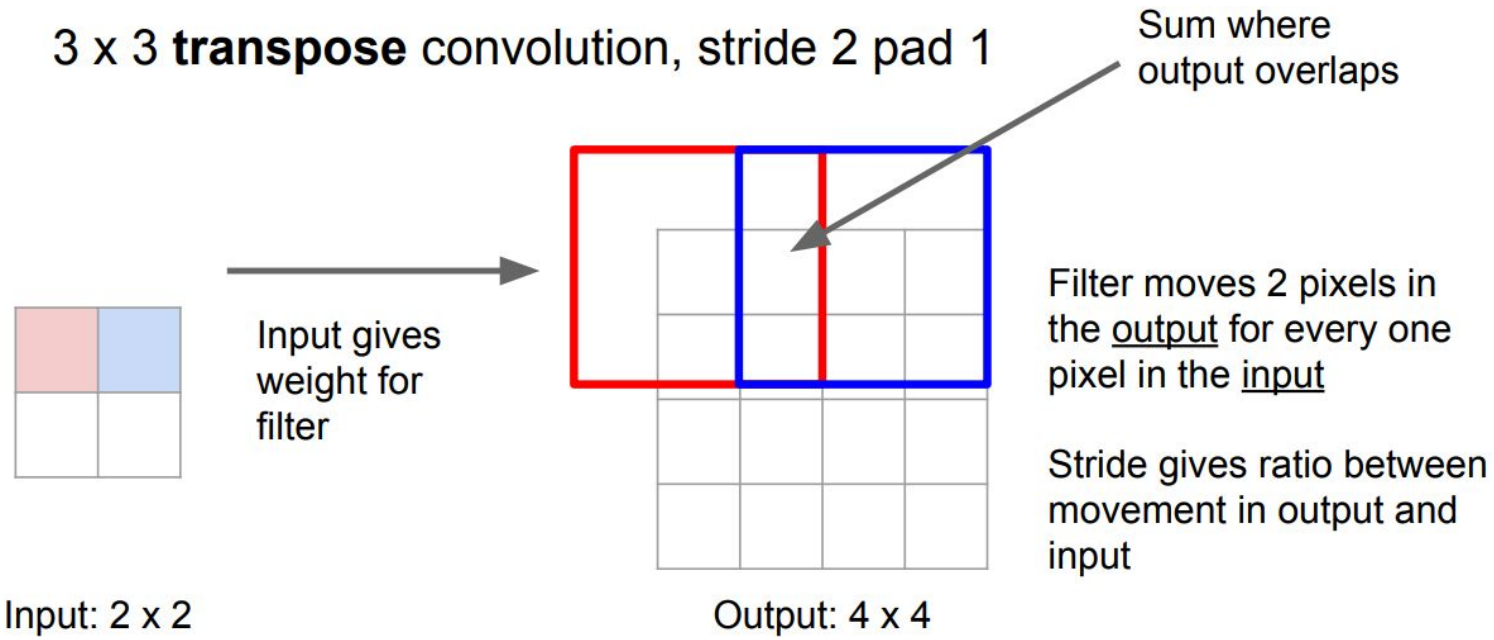


Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Learnable Upsampling: Transpose Convolution

Other names:
-Deconvolution (bad)
-Upconvolution
-Fractionally strided convolution
-Backward strided convolution

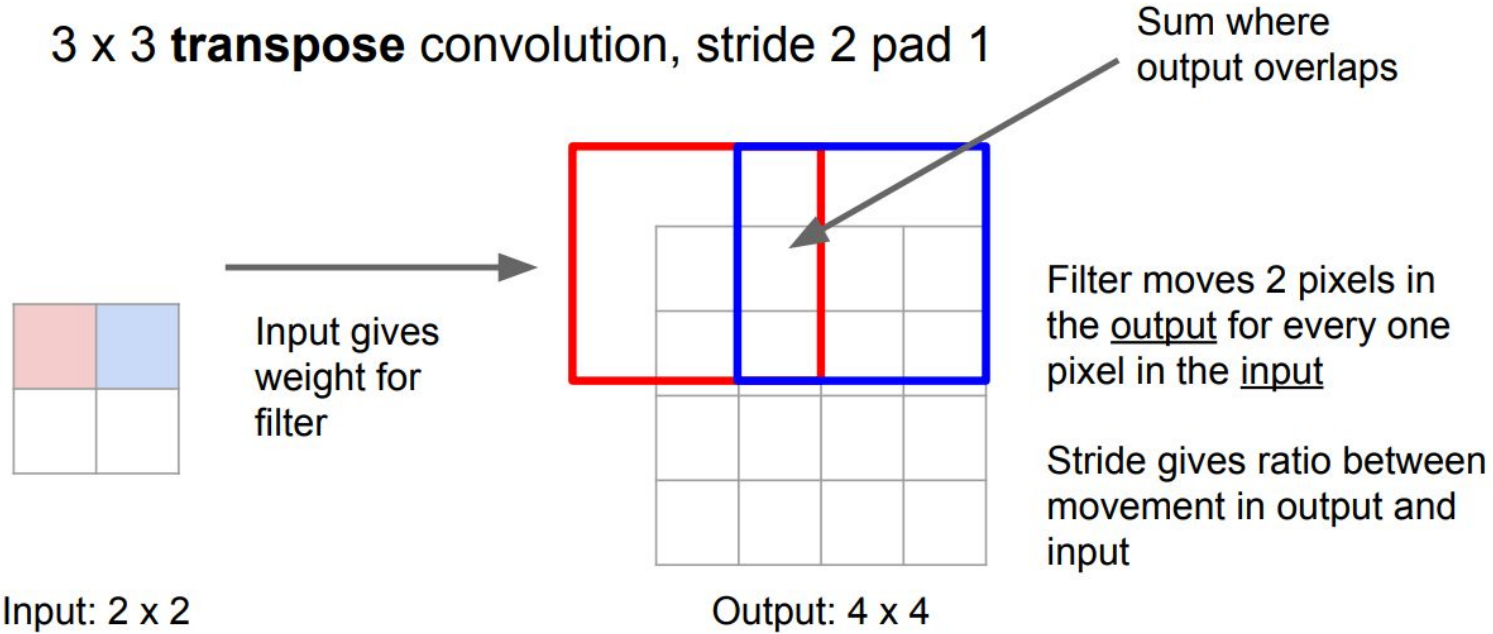


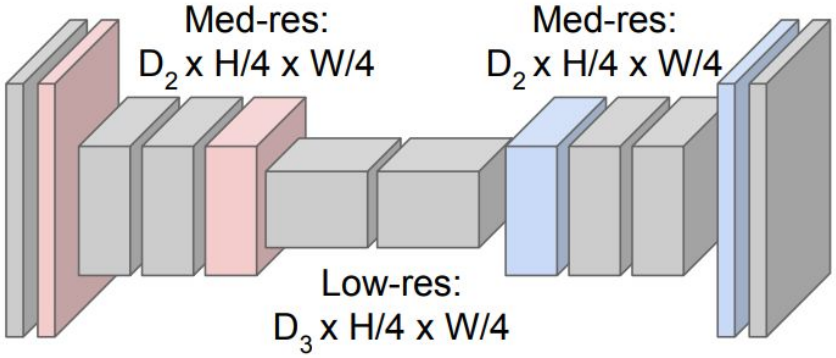
Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Idea: Fully Convolutional

Downsampling:
Pooling, Strided 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

Upsampling:
Unpooling or strided transpose convolution

Predictions:
H x W

Image: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

U-Net

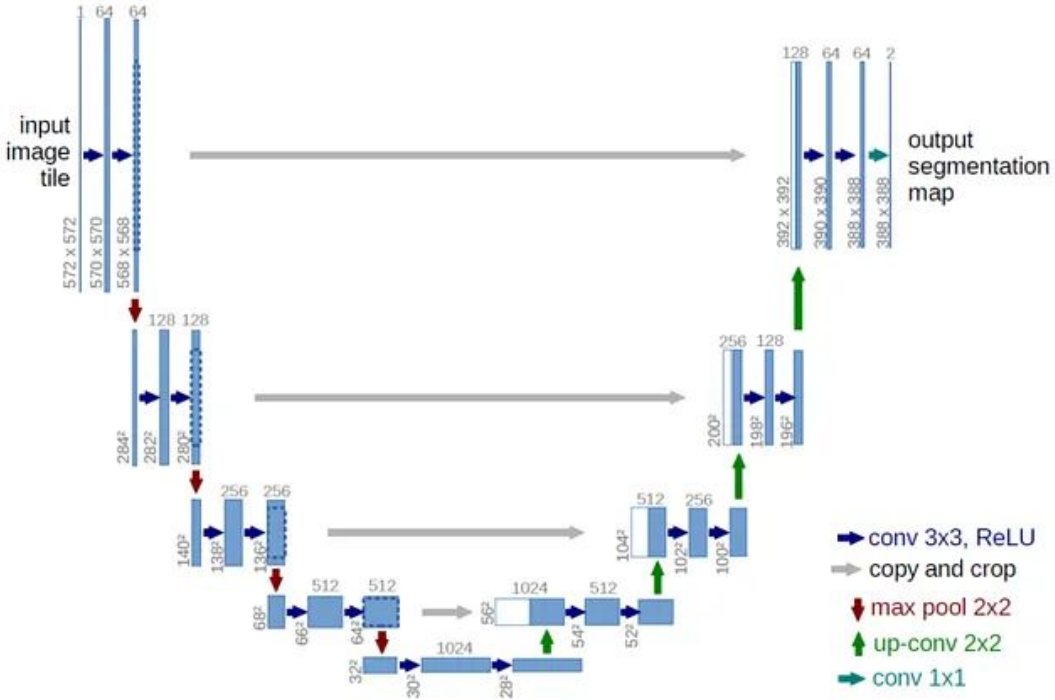


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

U-Net

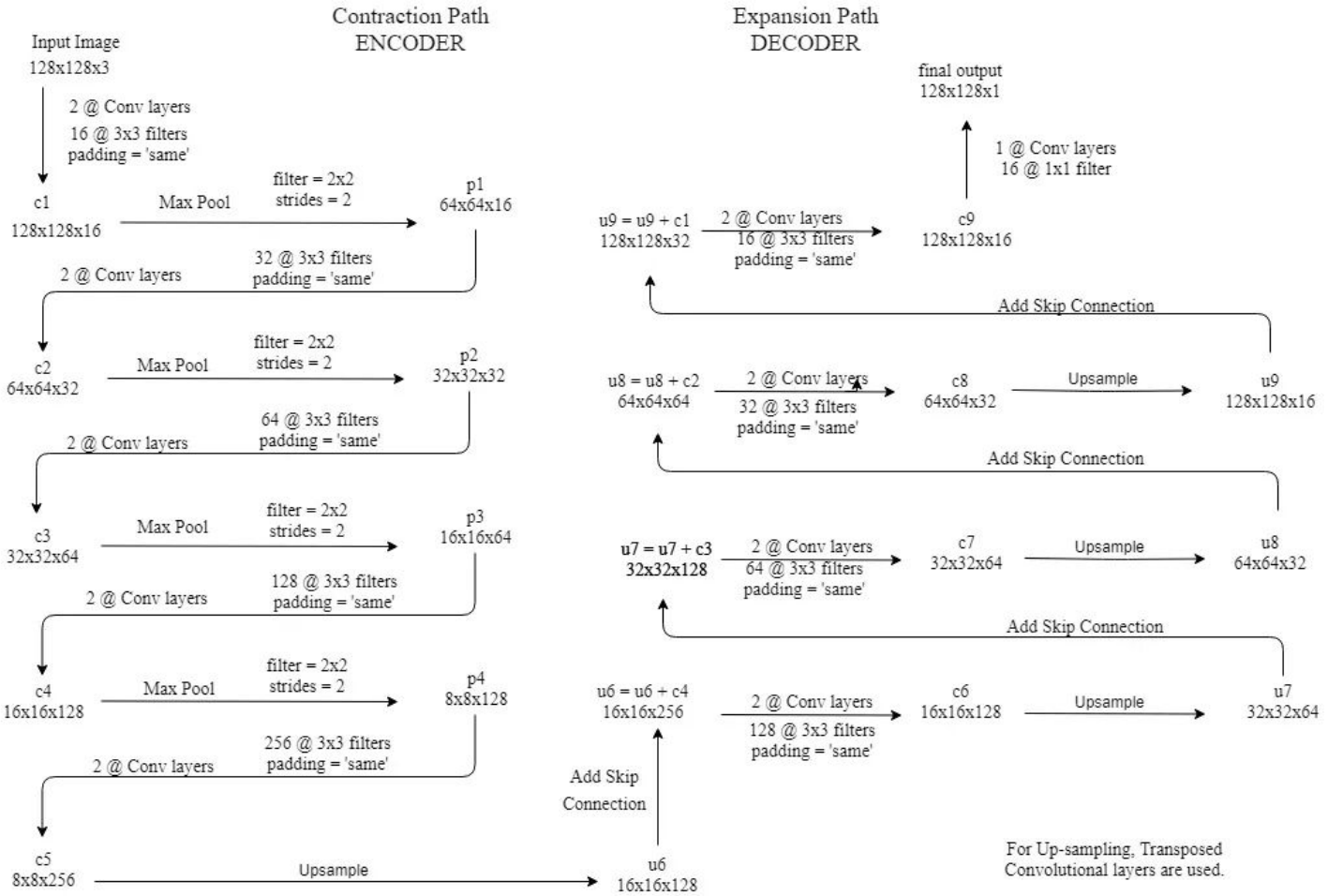


Image: https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

U-Net: Encoder - The Downsampler!

- 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

U-Net: Decoder - The Upsampler!

- 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
- This is what gives the architecture a symmetric U-shape, hence the name UNET
- On a high level, we have the following relationship:
- Input (128x128x1) => Encoder =>(8x8x256) => Decoder =>Output (128x128x1)

Next Lecture: Assorted Topics!