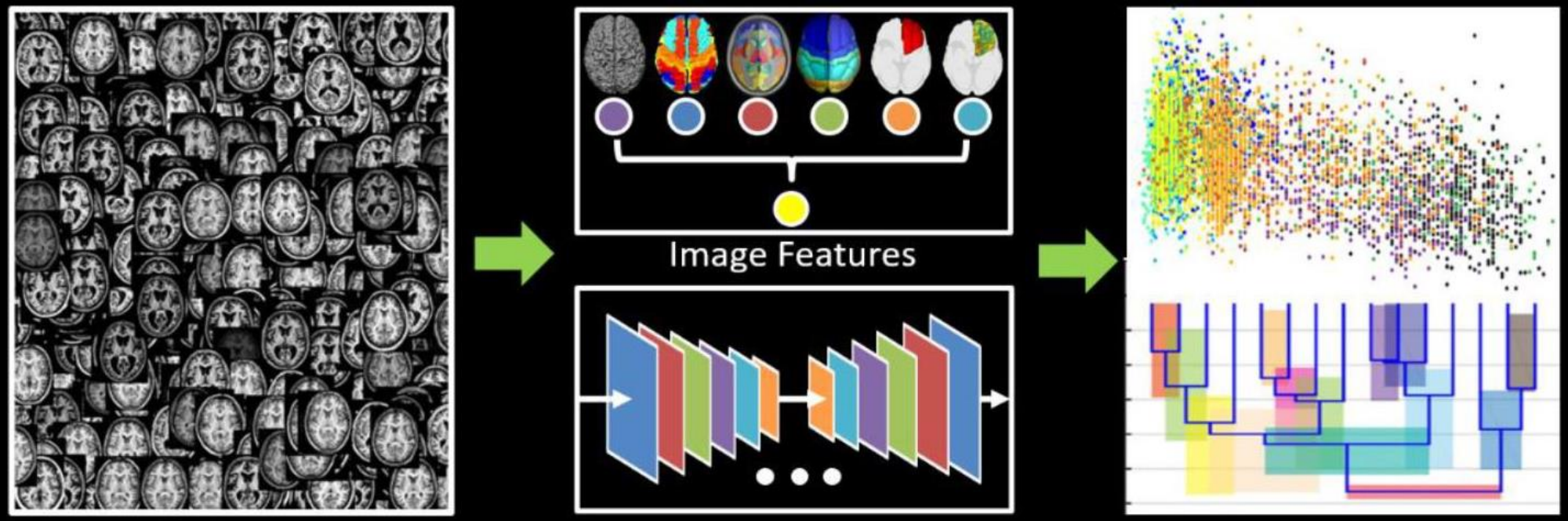



# Deep Image Segmentation

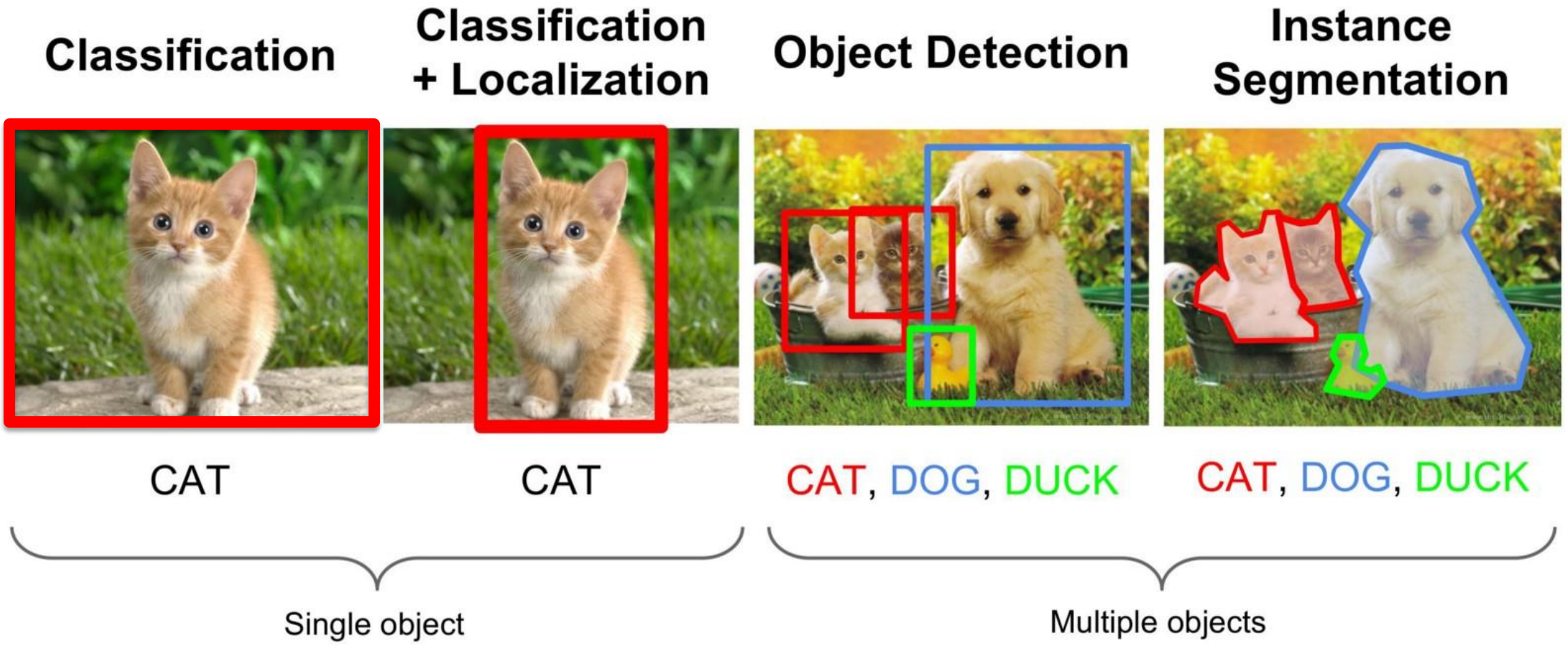
## [Spring 2020 CS-8395 Deep Learning in Medical Image Computing]

Instructor: Yuankai Huo, Ph.D.
Department of Electrical Engineering and Computer Science
Vanderbilt University

# Topics


- Review
- Semantic Segmentation
- U-Net
- Instance Segmentation

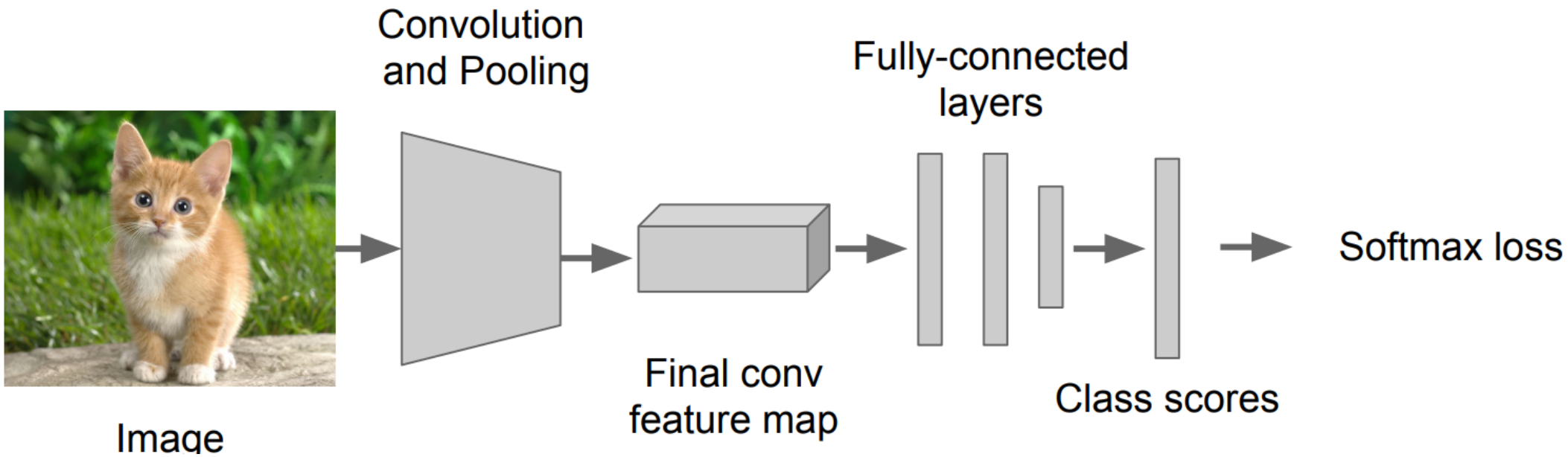
# Localization & Detection




https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852

# Simple Recipe for Classification

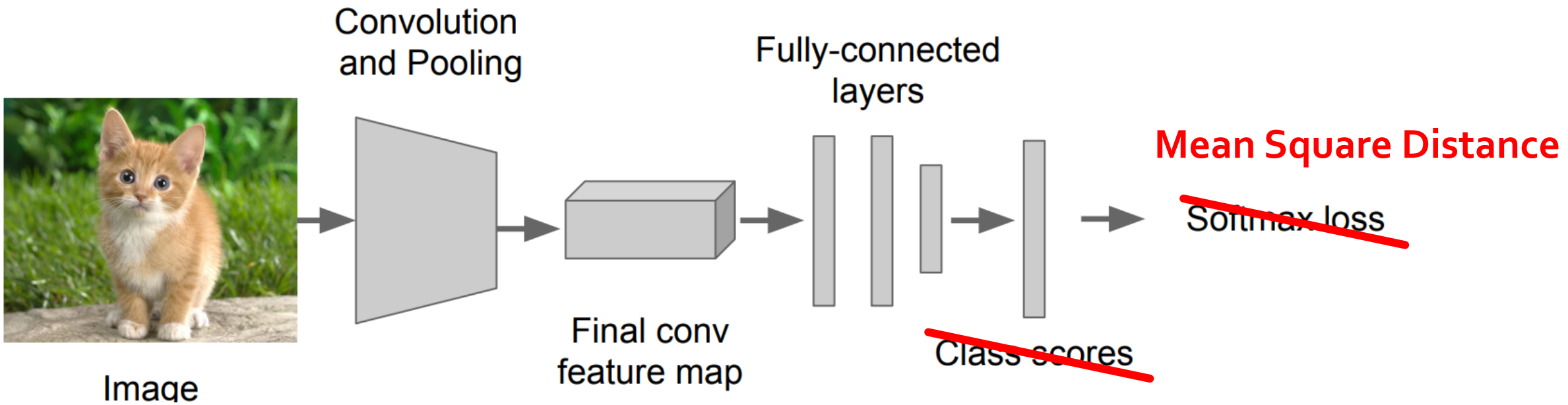
## "CONV-POOL-FC"




http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

# Simple Recipe for Localization

**"CONV-POOL-FC"**




http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

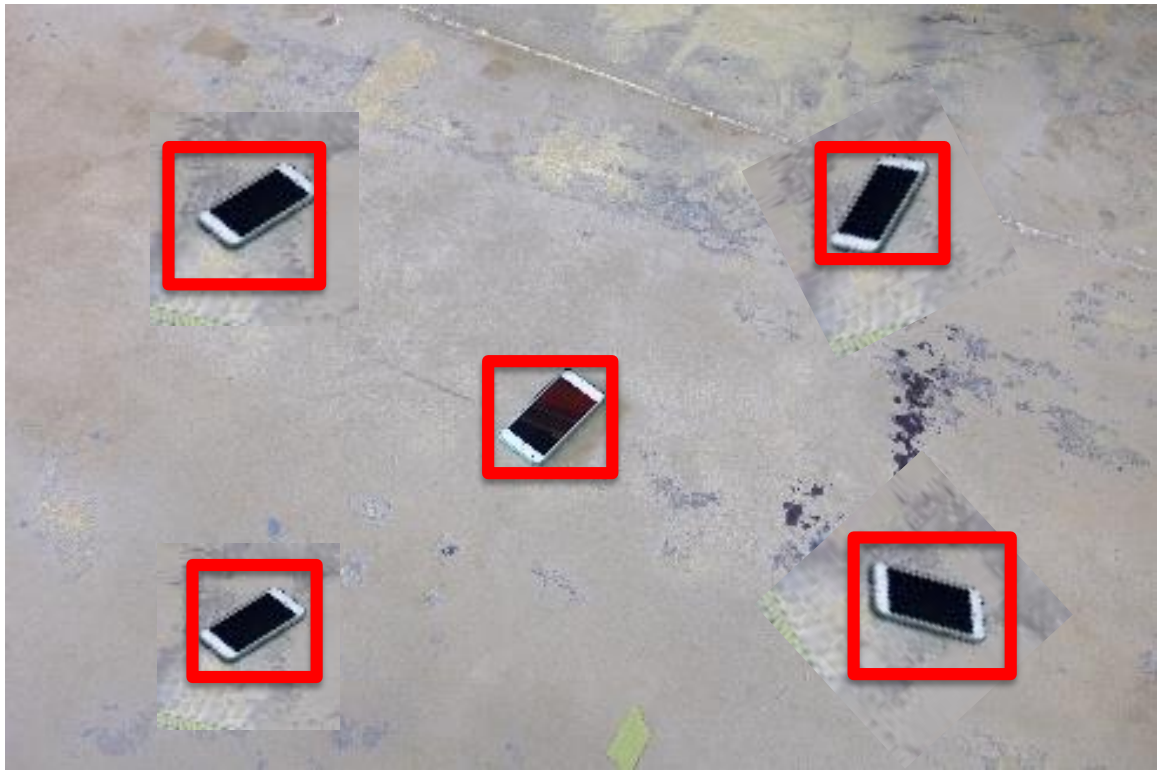
# Single Landmark Detection

Center Point (x, y)

Bounding Box (x1, y1,x2,y2)

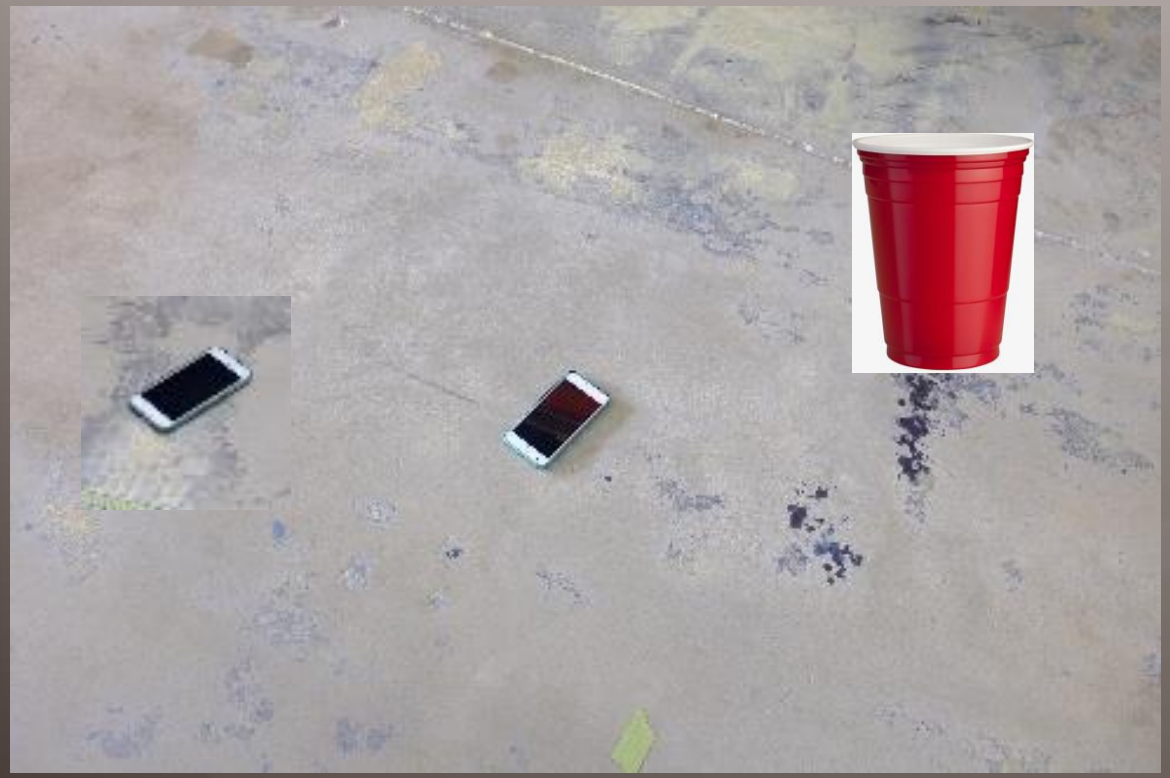
# Multi Landmark Detection

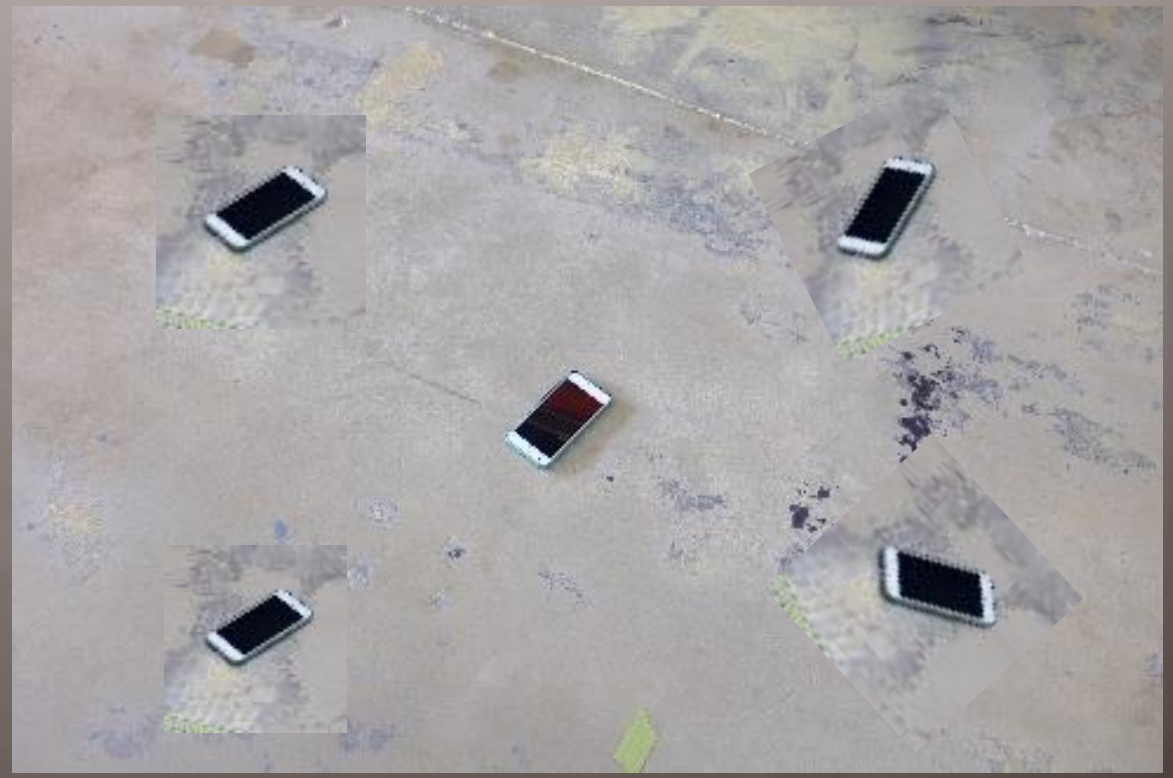
Center Points (x1, y1, x2, y2 …)



Bounding Boxs (x11, y11,x12,y12 x21,y21,x22,y22 …)

# Now, what to do?

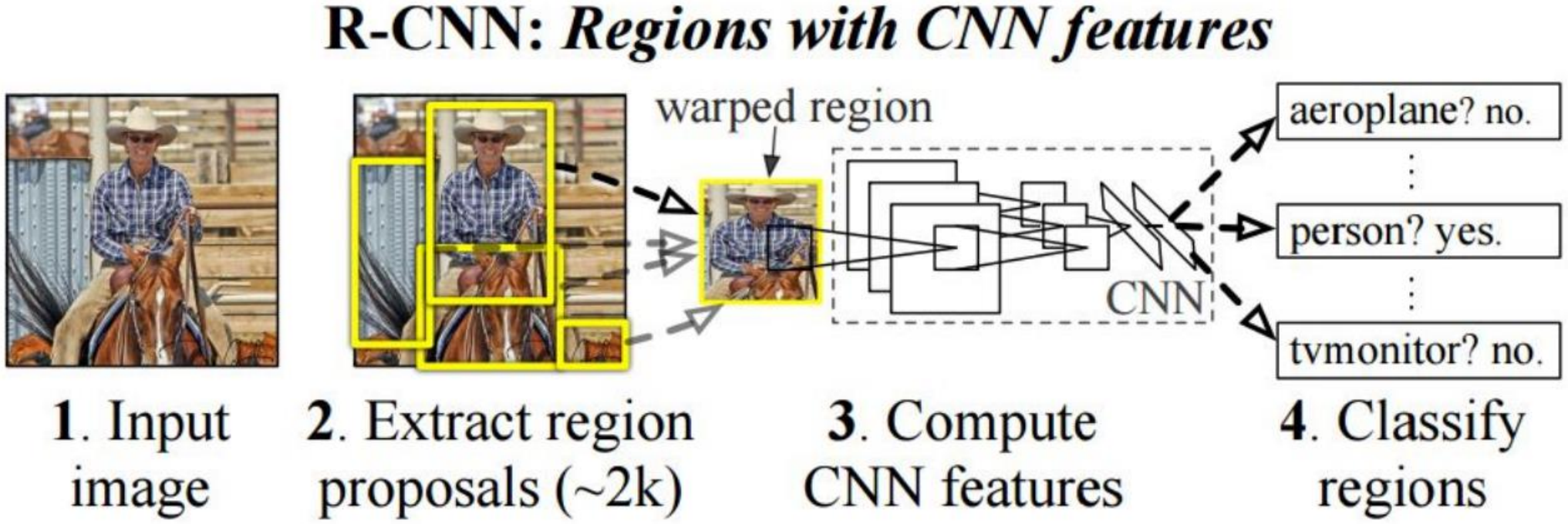




# Idea 1: Sliding Window

**Detect Face**

# R-CNN



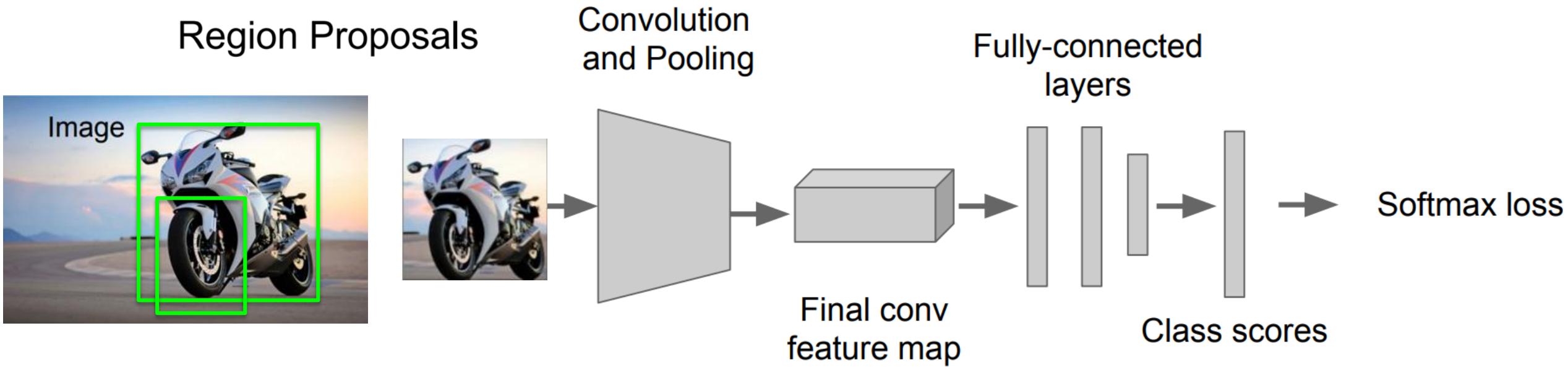

**Stage 1: Region Proposal**

**Stage 2: Classification**

https://arxiv.org/abs/1311.2524

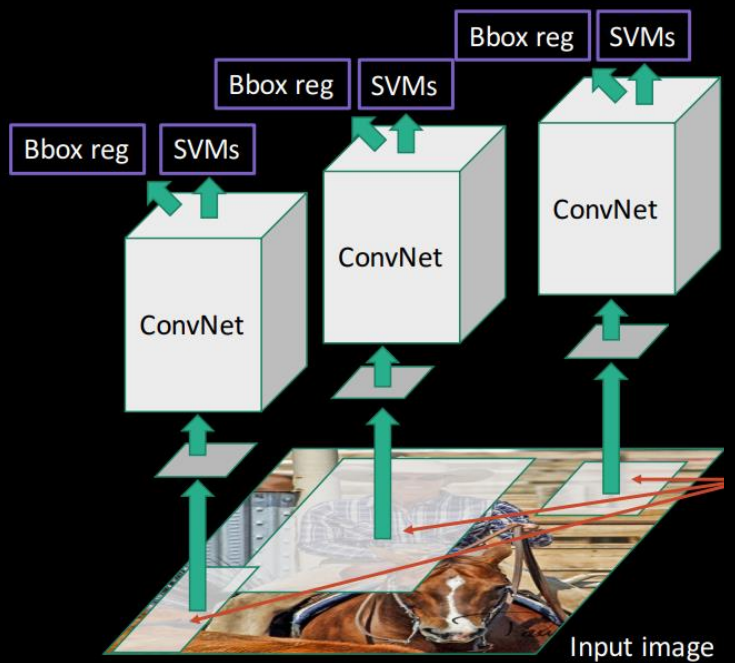
# Detection

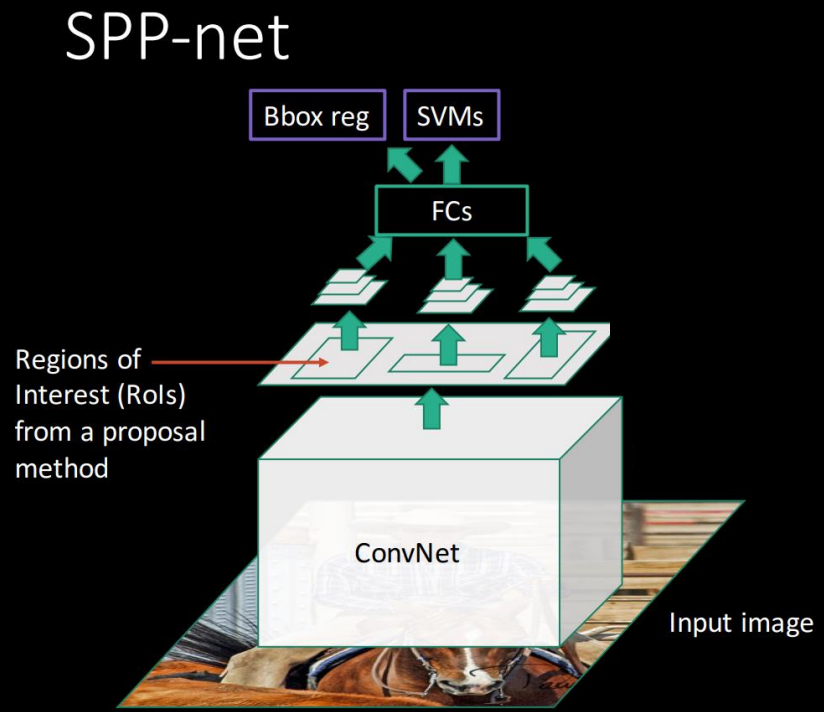
## "CONV-POOL-FC"

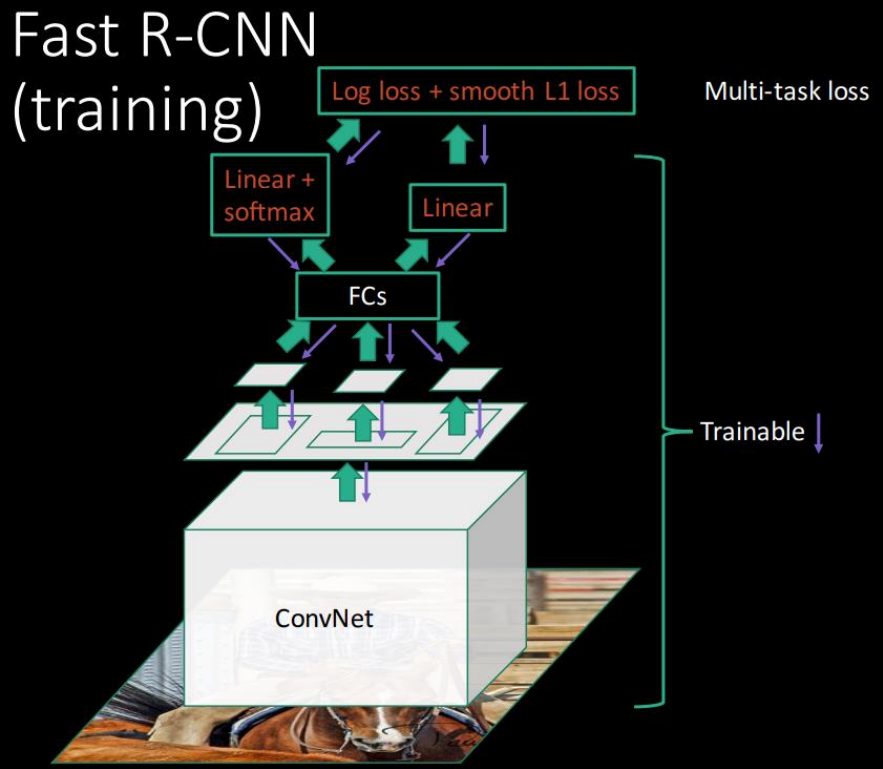



http://slazebni.cs.illinois.edu/spring17/lec07_detection.pdf
https://blog.csdn.net/sum_nap/article/details/80388110
http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

# RCNN → Fast RCNN

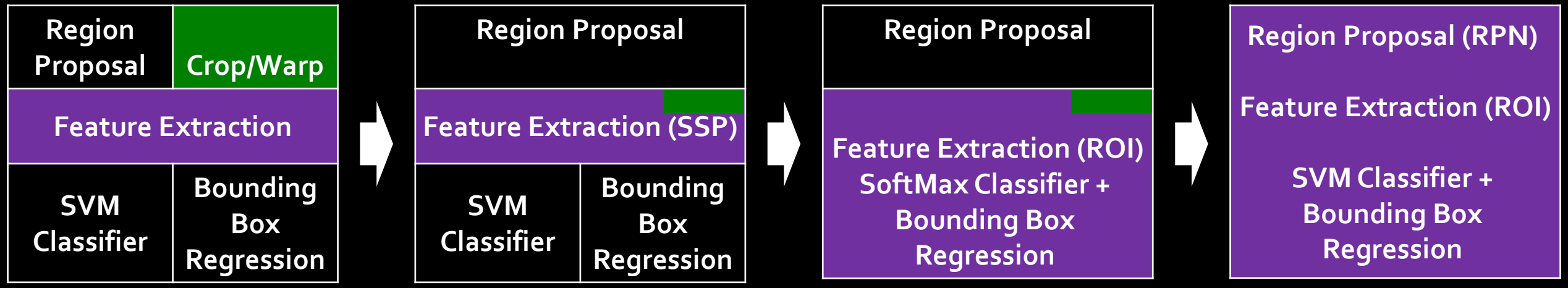


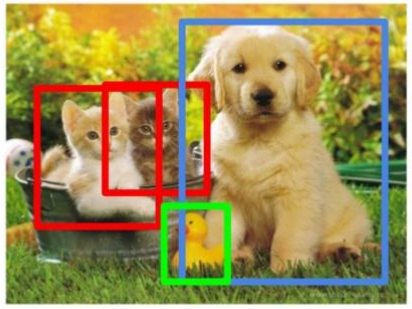
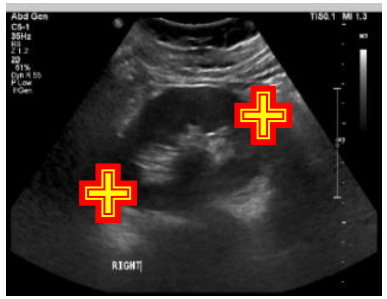
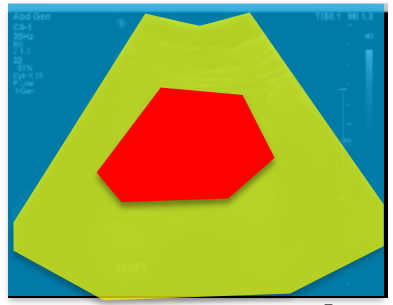

http://www.robots.ox.ac.uk/~tvg/publications/talks/fast-rcnn-slides.pdf

# Summerize



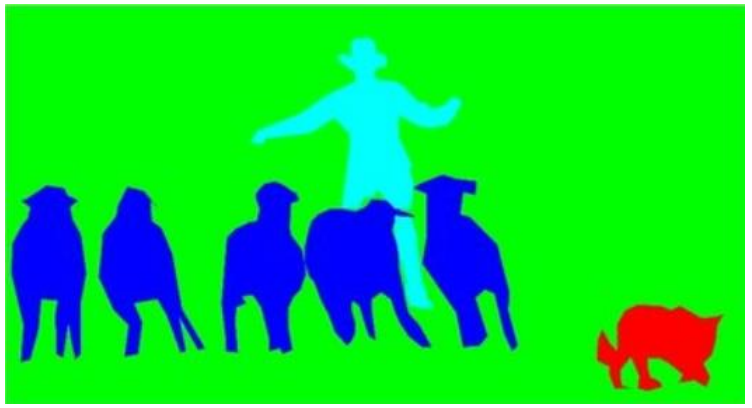

R-CNN SPP-Net Fast R-CNN Faster R-CNN

# Compare with Computer Vision

| | Classification | Classification + Localization | Object Detection | Instance Segmentation | Synthesis |
|---|---|---|---|---|---|
| **Computer Vision** |  CAT |  CAT |  CAT, DOG, DUCK |  CAT, DOG, DUCK |  |
| **Medical Image Computing** |  Input |   Classification |  Detection |  Segmentation |  Synthesis |

https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852

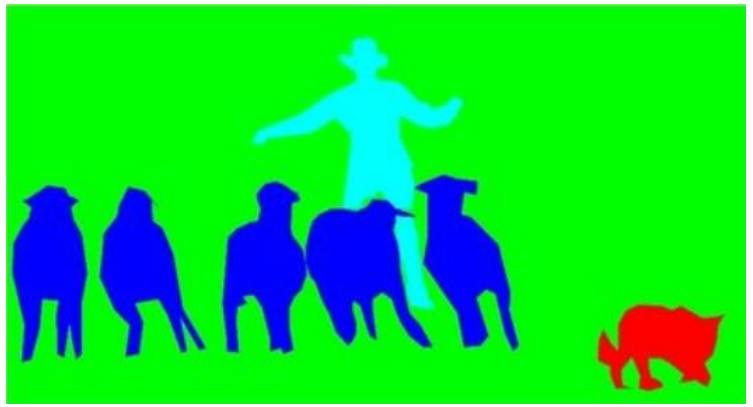
# Segmentation

Semantic Segmentation

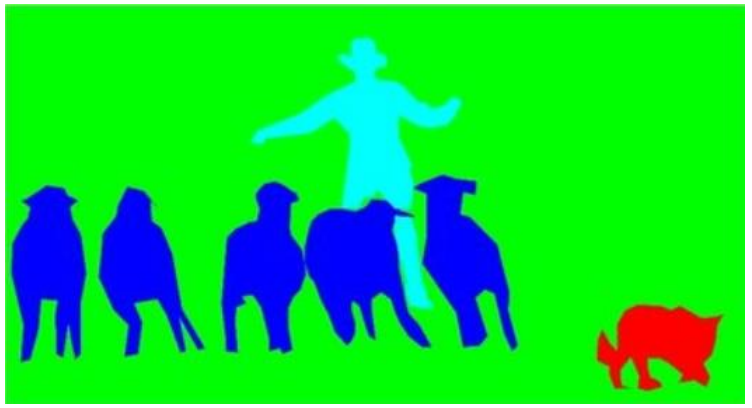
Instance Segmentation

# Segmentation

**Semantic Segmentation**

# Segmentation

Semantic Segmentation

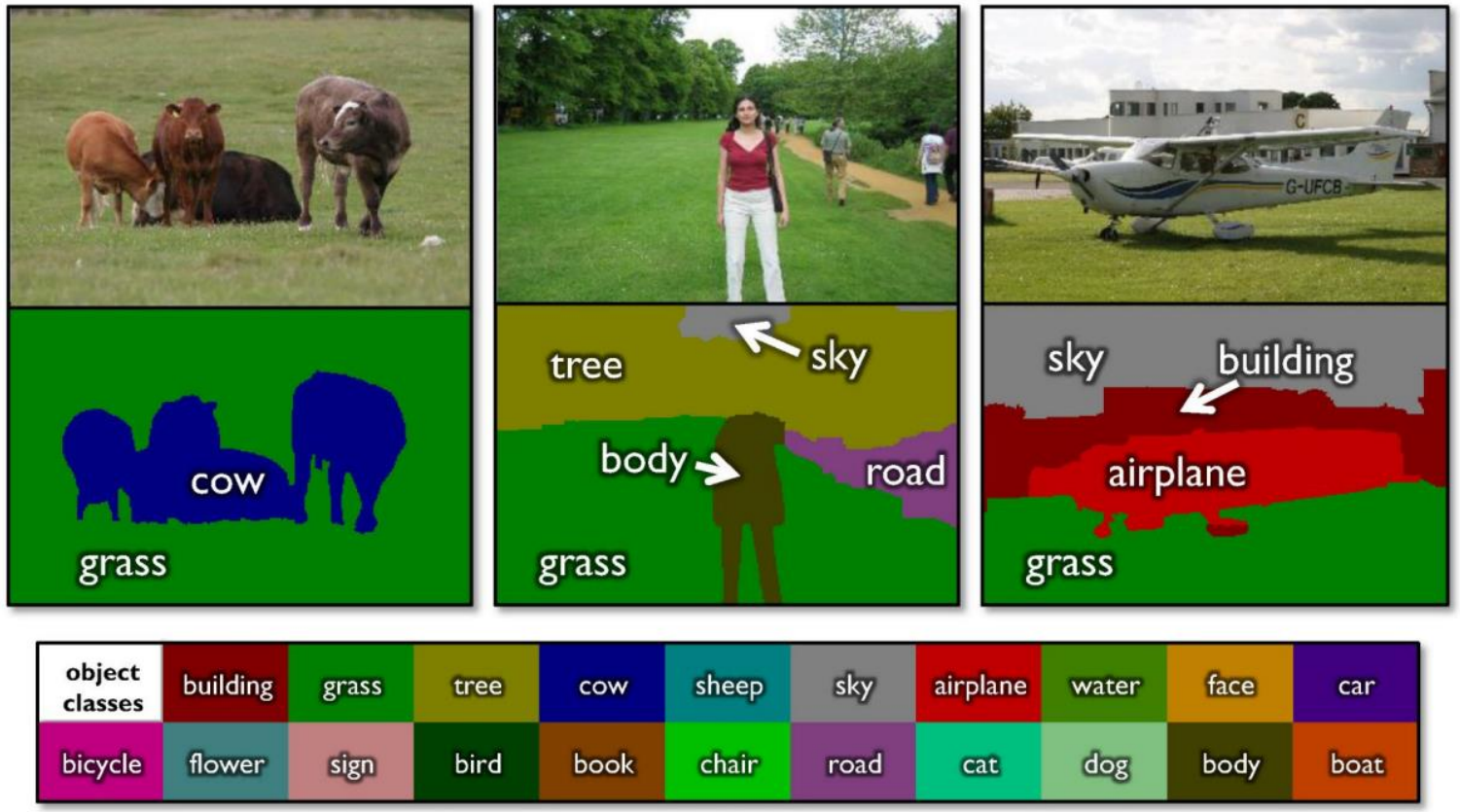
Instance Segmentation

# Semantic Segmentation

Label every pixel!

Don’t differentiate instances (cows)

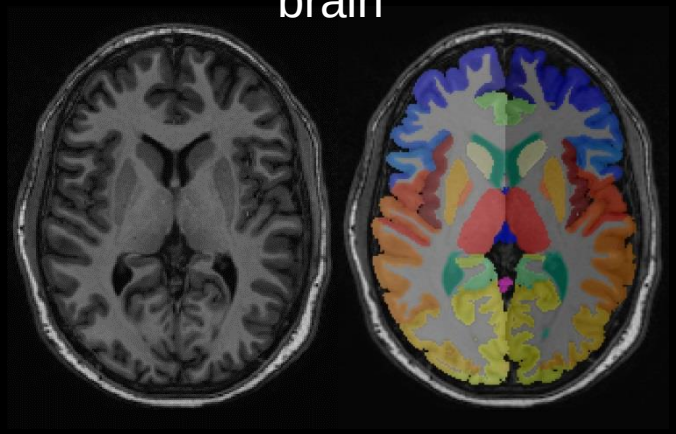
Classic computer vision problem



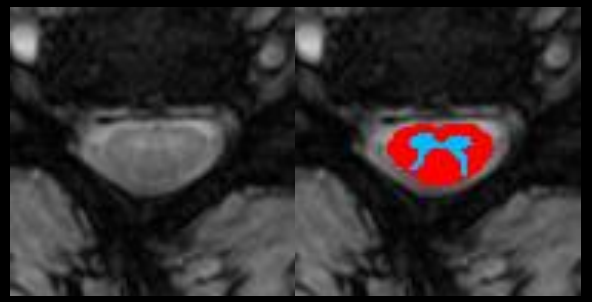

https://telecombcn-dl.github.io/2018-dlmm/

# Medical Image Segmentation

brain



spinal cord

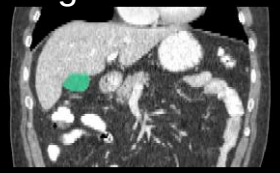


spleen

right kidney

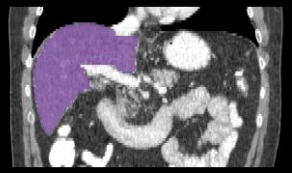
left kidney

gallbladder

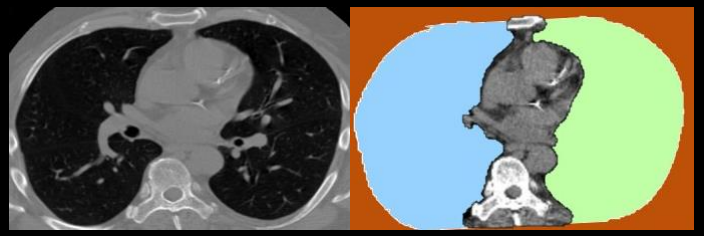


esophagus

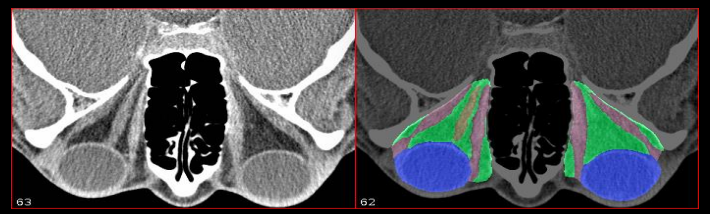
liver



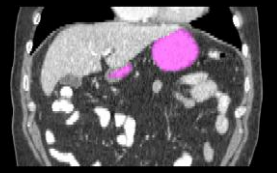
lung



optical nerve



stomach

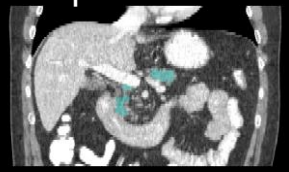


aorta

inferior vena cava

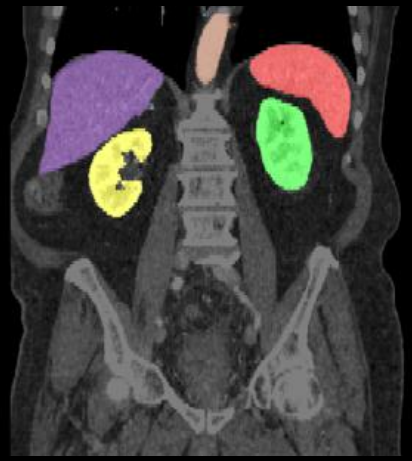
portal vein & splenic vein

pancreas



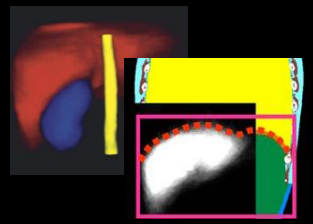
adrenal glands

# History

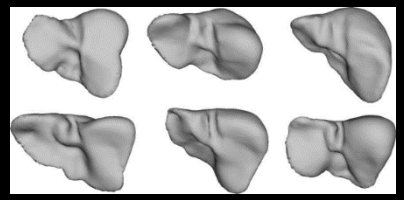


**Automatic Segmentation?**

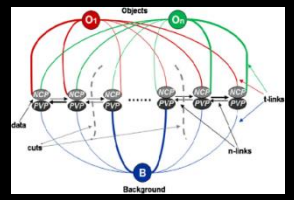
Probabilistic Atlas



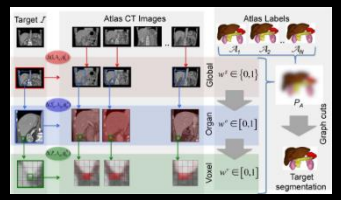
+Statistical Shape Model



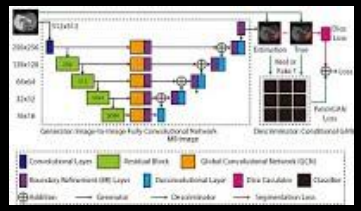
+Graph Cut




Multi-Atlas Label Fusion



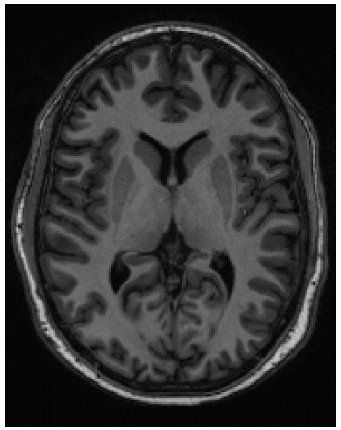

Deep Learning

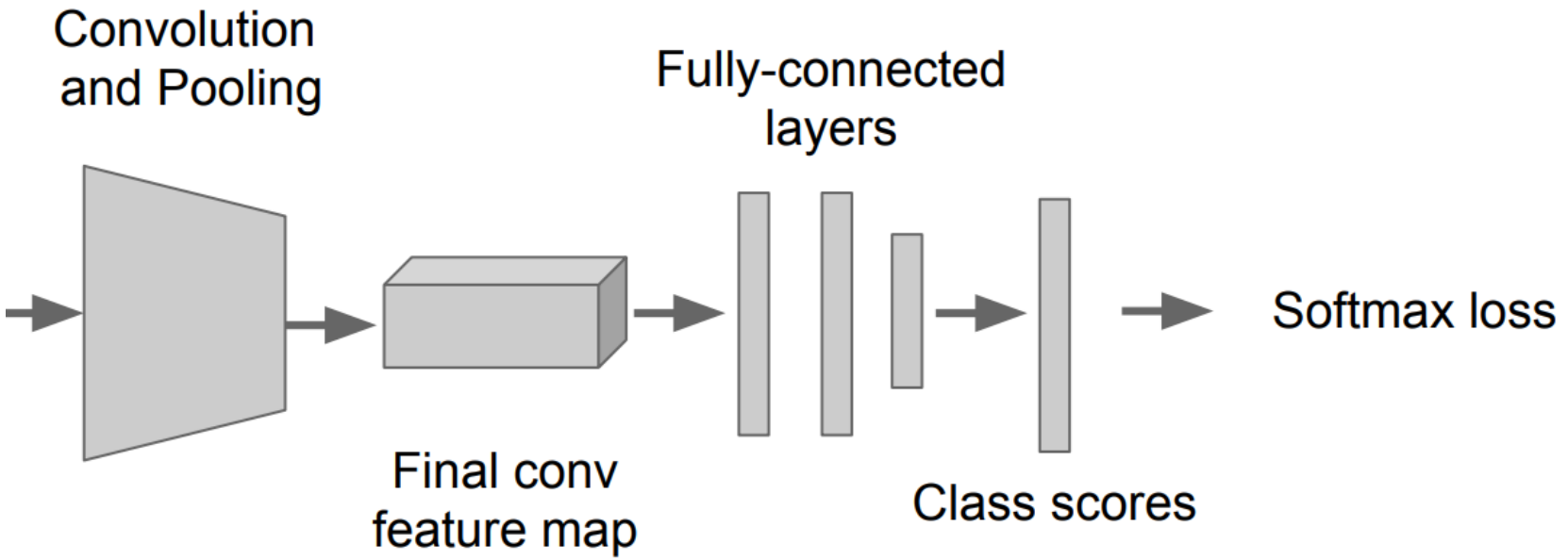


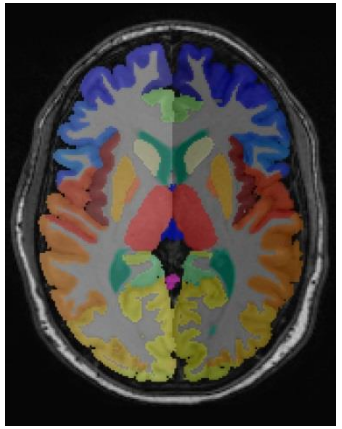
Park et al. (2003)
Zhou et al. (2005)
Shimizu et al. (2007)
Okada et al. (2008)
Heimann et al. (2009)
Bagci et al. (2012)
Chen et al. (2012)
Linguraru et al. (2012)
Wolz et al. (2013)
Xu et al. (2015)

# Segmentation ?

## “CONV-POOL-FC”



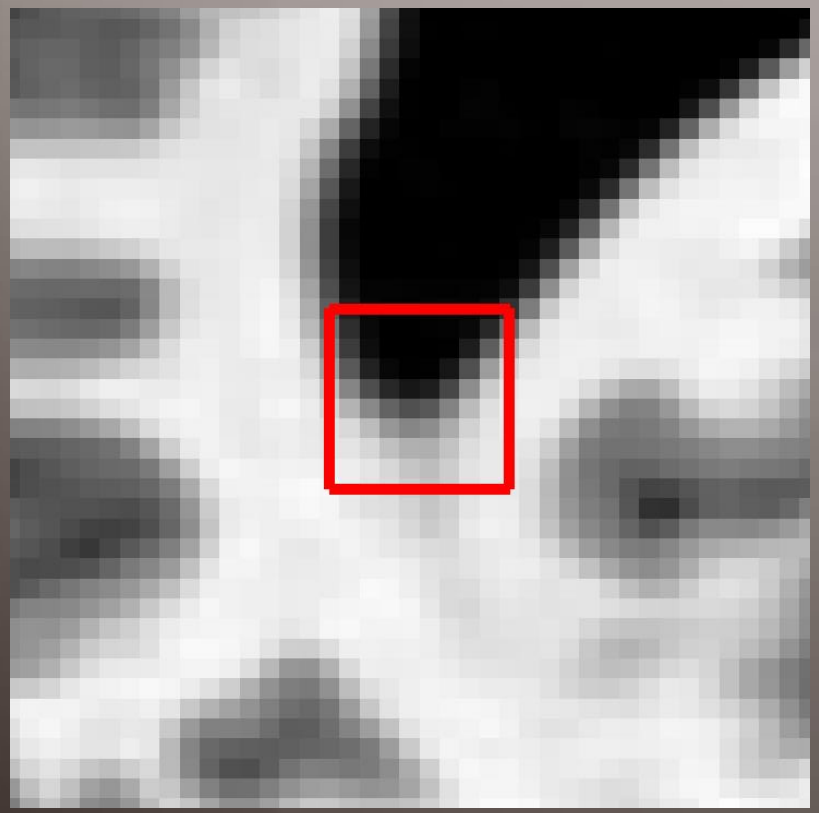


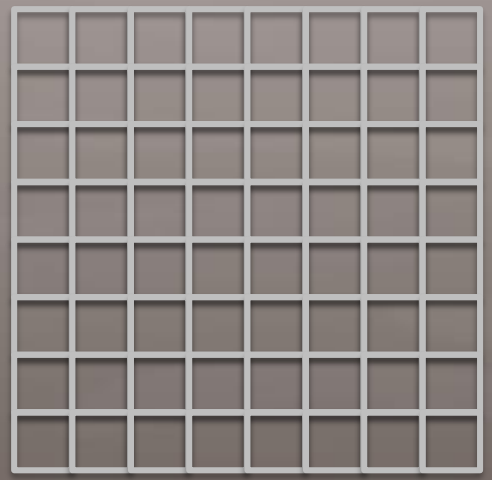



http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

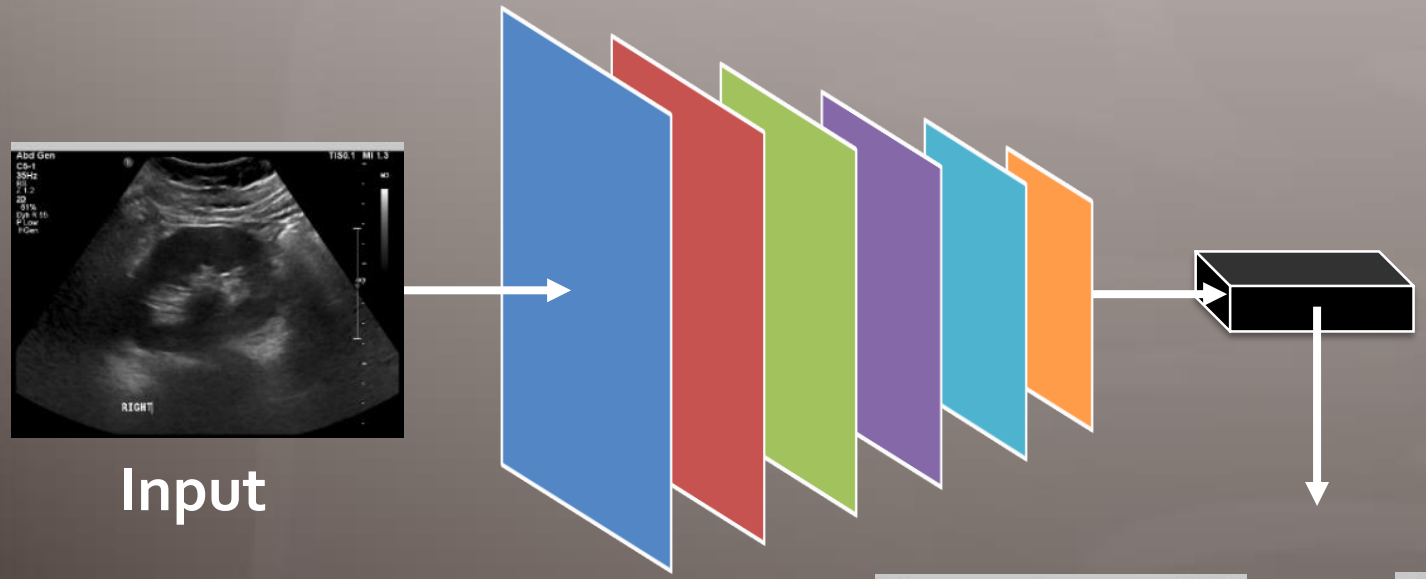
# everything can be solved by classification

# Segmentation is Classification



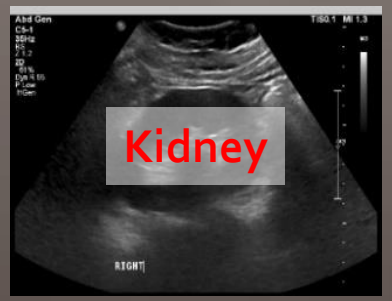


# Classification / Regression

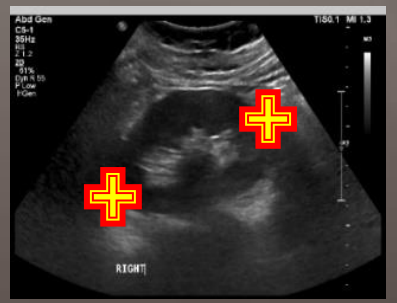


CONV-POOL-FC

Input

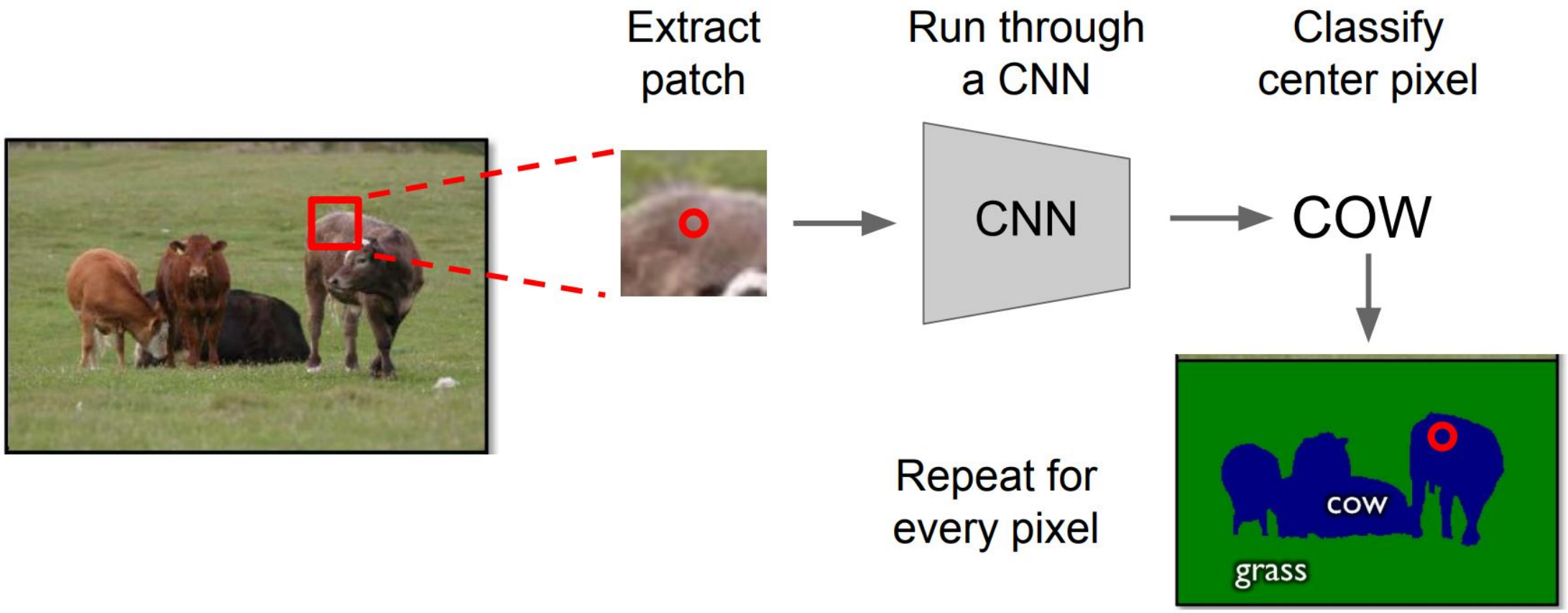



Classification



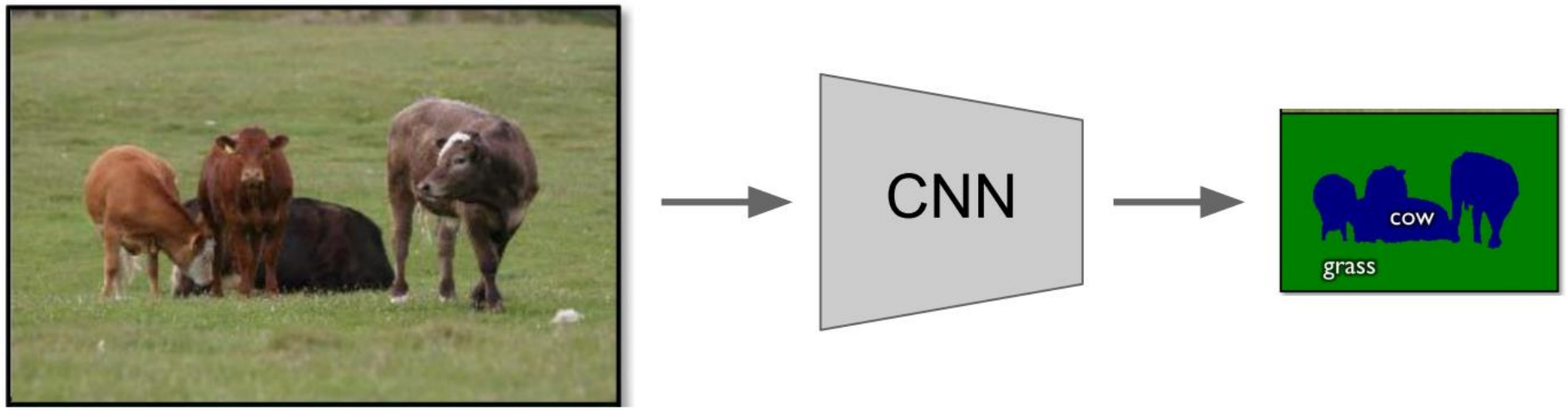
Detection

# Classification to Segmentation



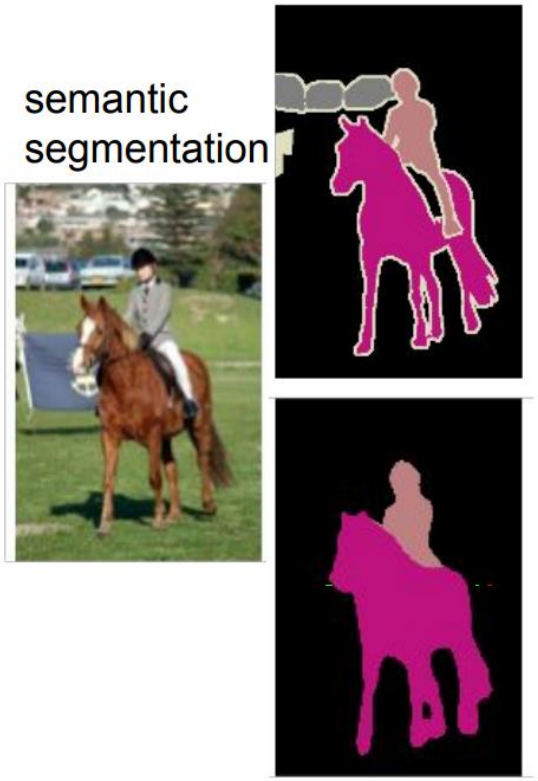
# FCN

Run “fully convolutional” network
to get all pixels at once

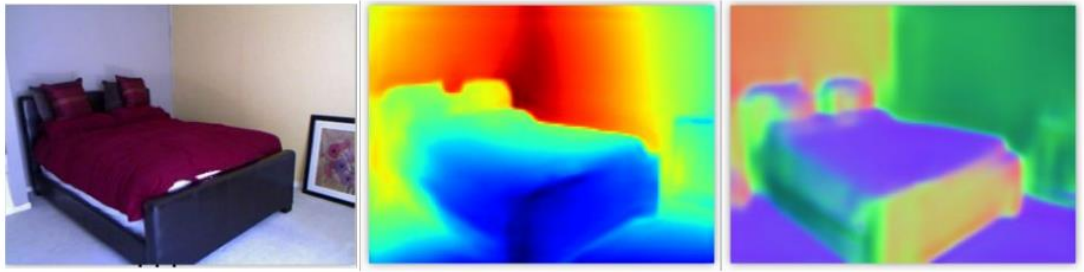


# Pixel In – Pixel Out (Image to Image)

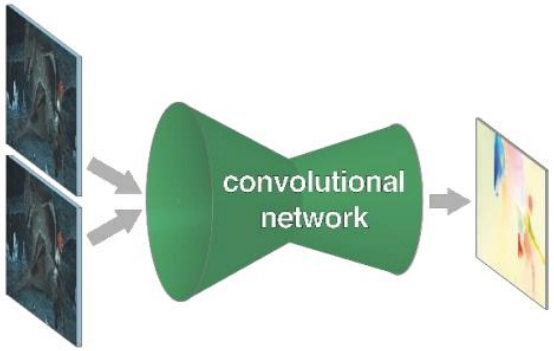
semantic segmentation



monocular depth + normals Eigen & Fergus 2015



colorization
Zhang et al.2016



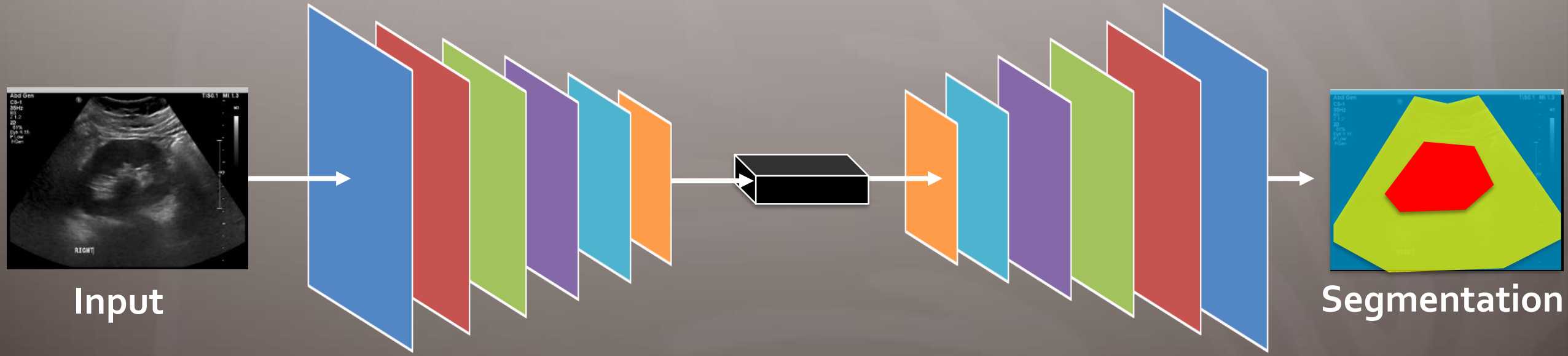

optical flow Fischer et al. 2015

boundary prediction Xie & Tu 2015

2

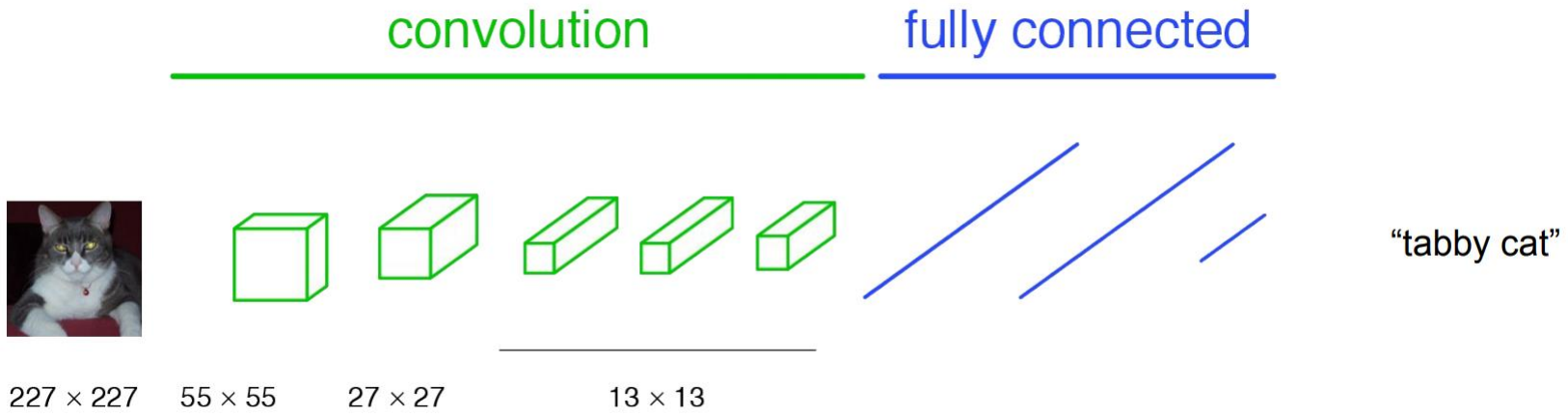
http://www.micc.unifi.it/bagdanov/pdfs/FCN-presentation.pdf

# Segmentation (Image to Image)

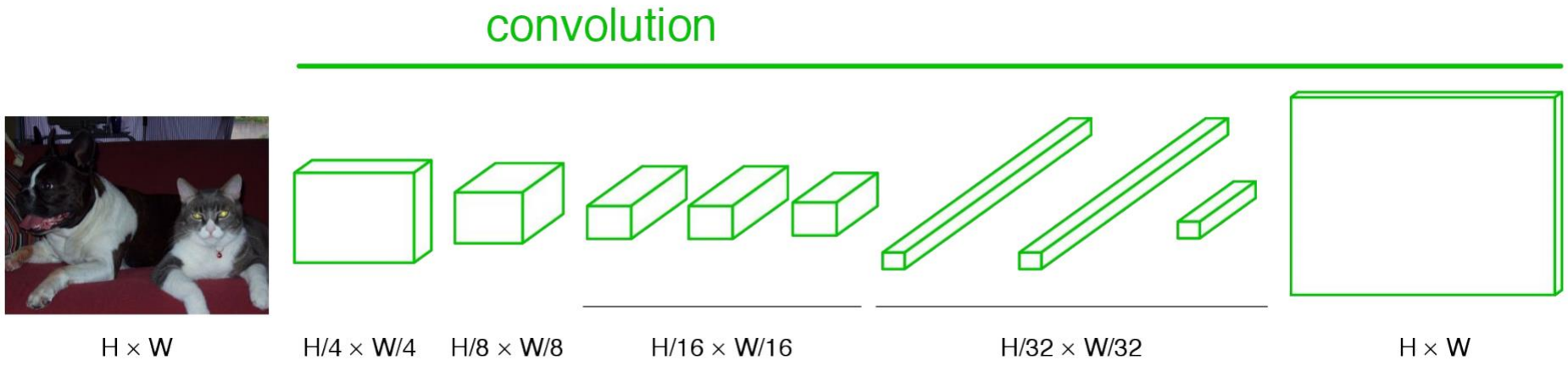


# Change a Classification Network

Classification

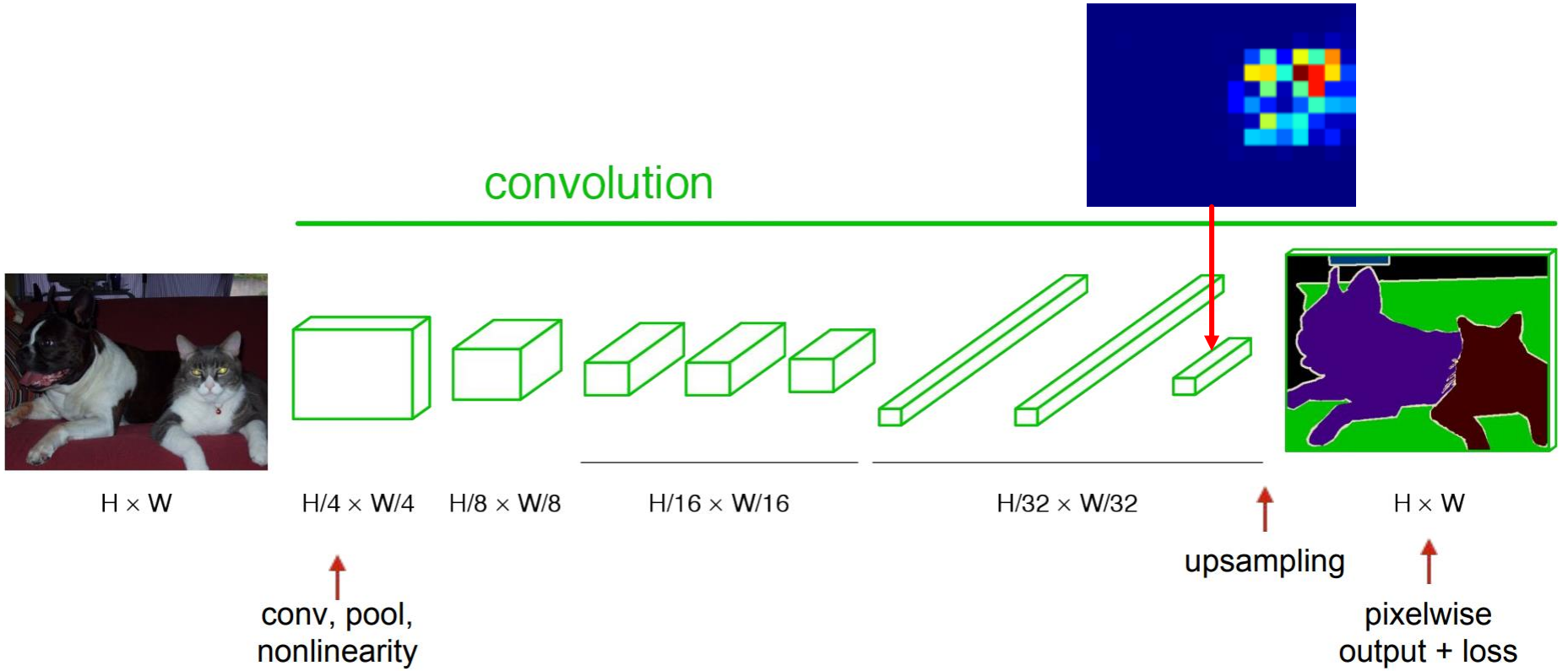



Fully Convolutional



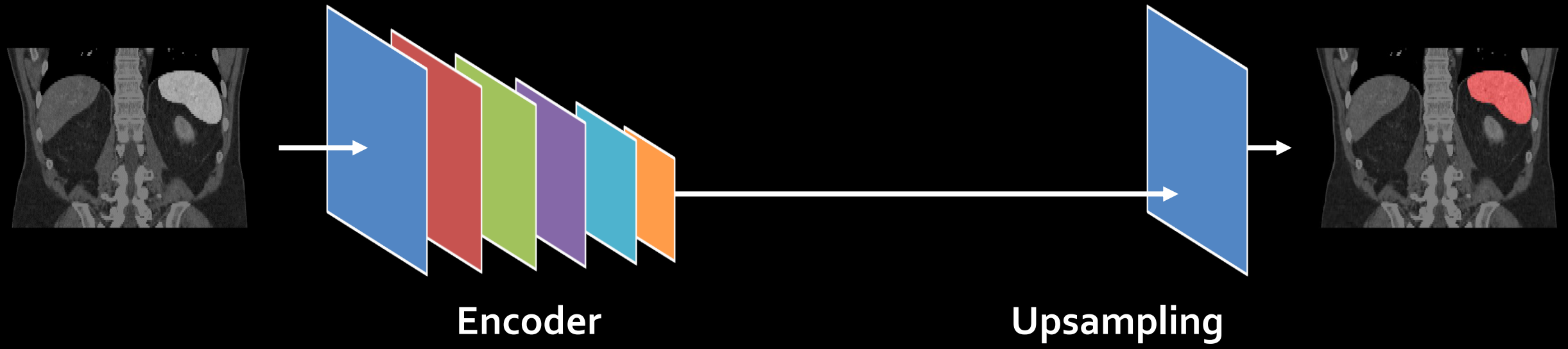

http://www.micc.unifi.it/bagdanov/pdfs/FCN-presentation.pdf

# Naïve Idea

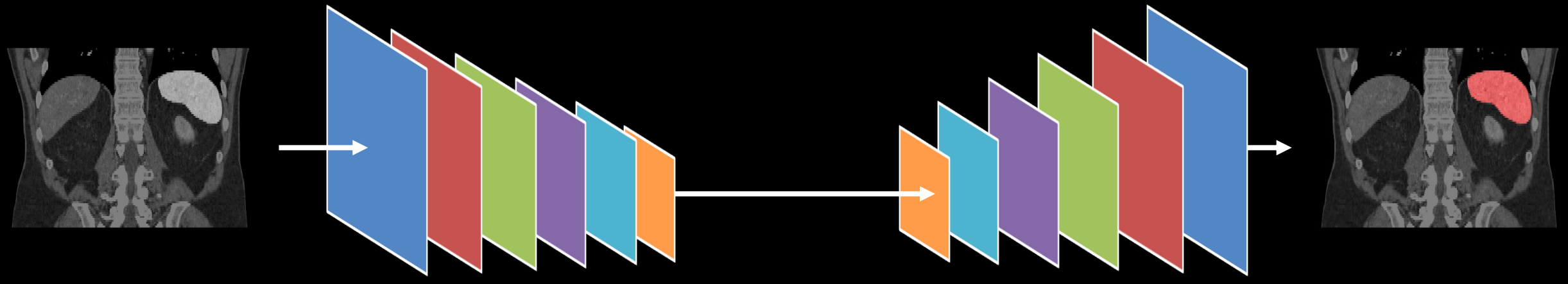



http://www.micc.unifi.it/bagdanov/pdfs/FCN-presentation.pdf

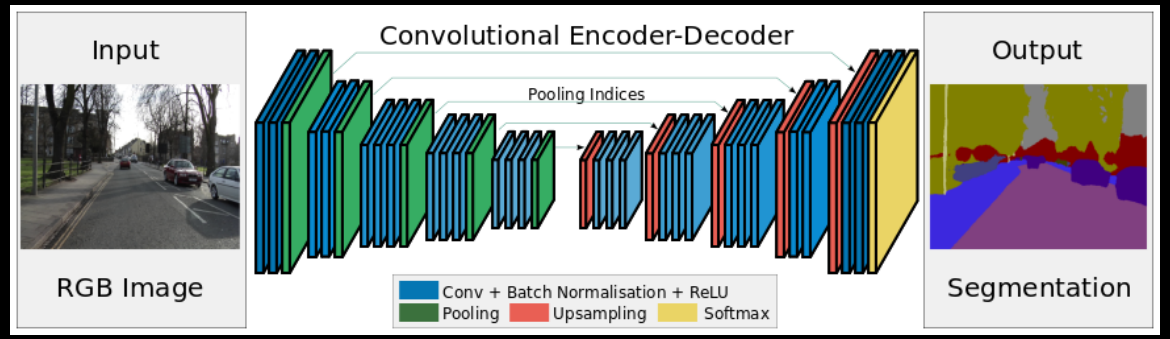
# Fully Convolutional Neural Network



# Encoder – Decoder Strategy

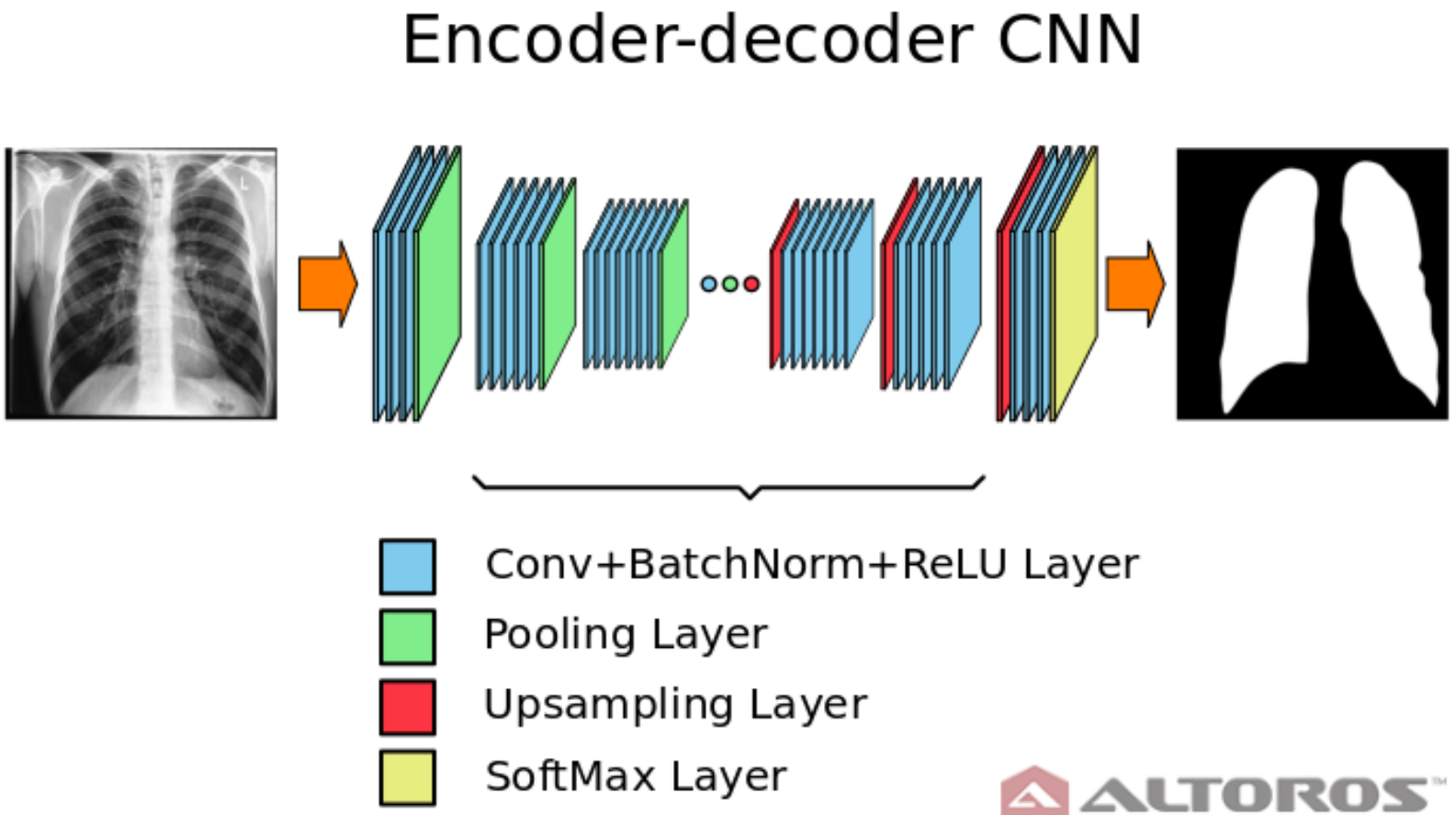



**SegNet:**



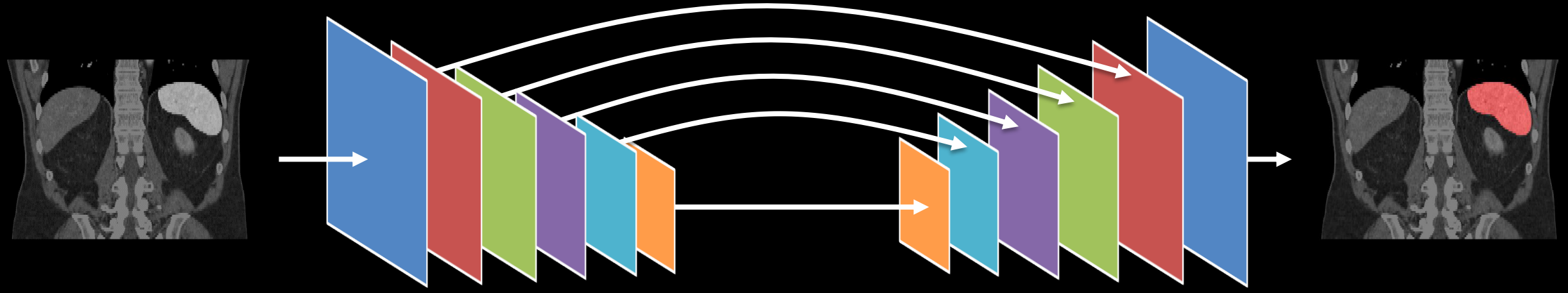

Badrinarayanan, et al. arXiv:1511.00561(2015)

# SegNet




https://blog.altoros.com/experimenting-with-deep-neural-networks-for-x-ray-image-segmentation.html

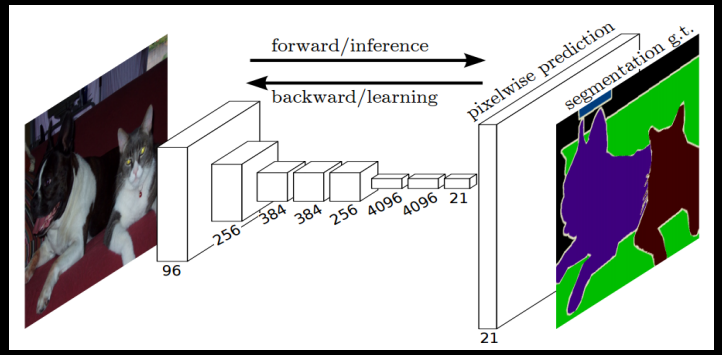
# Fully Convolutional Network (FCN)



Encoder

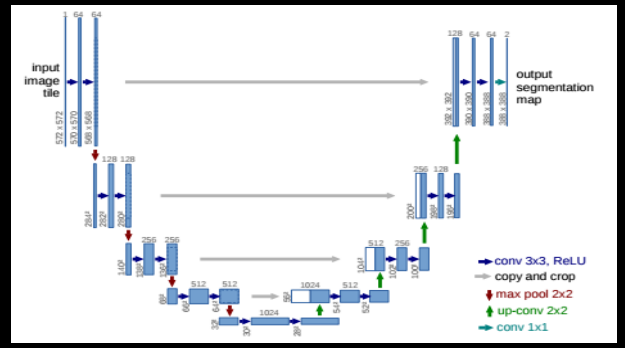
Decoder

**FCN:**



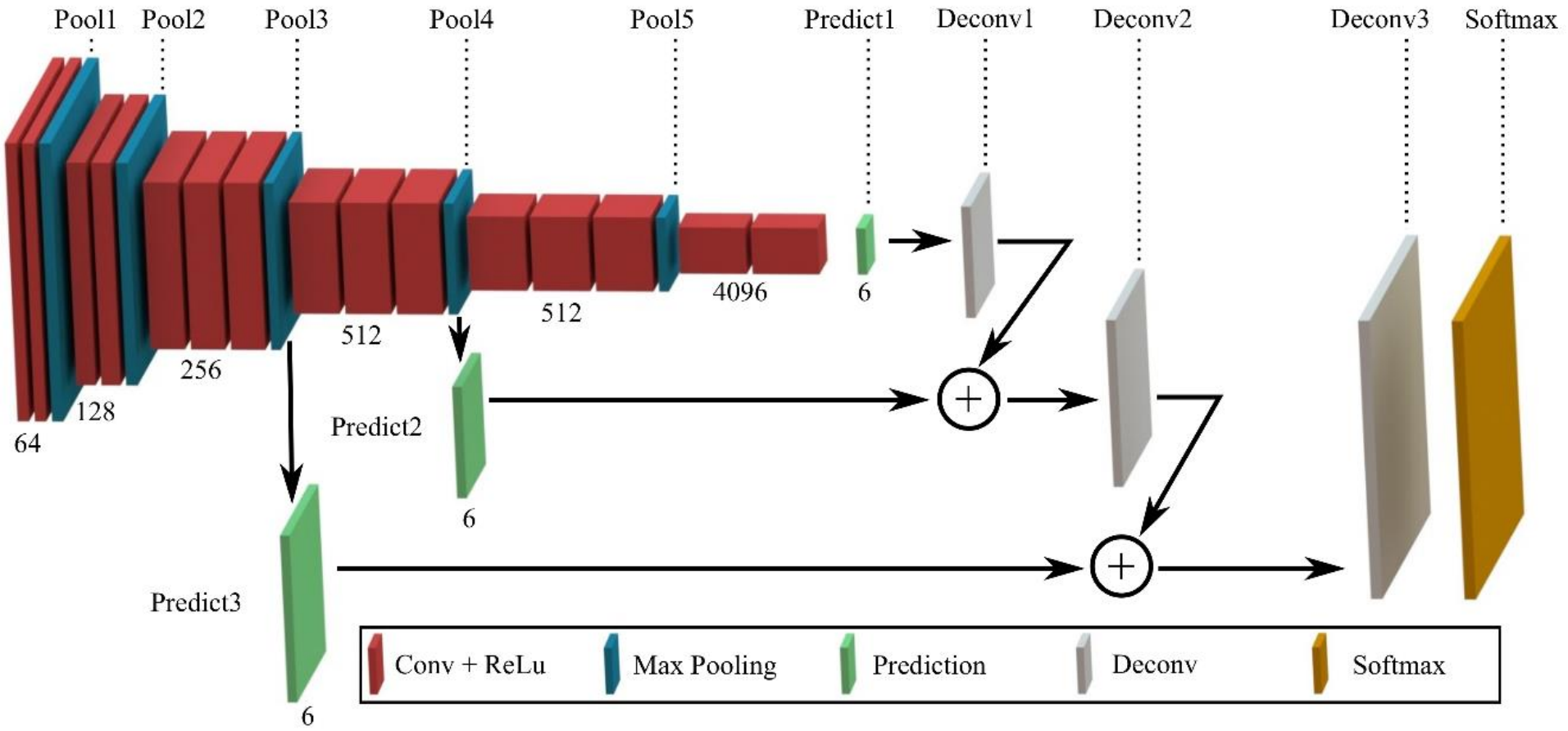

Long, et al. CVPR (2015)

**Unet:**



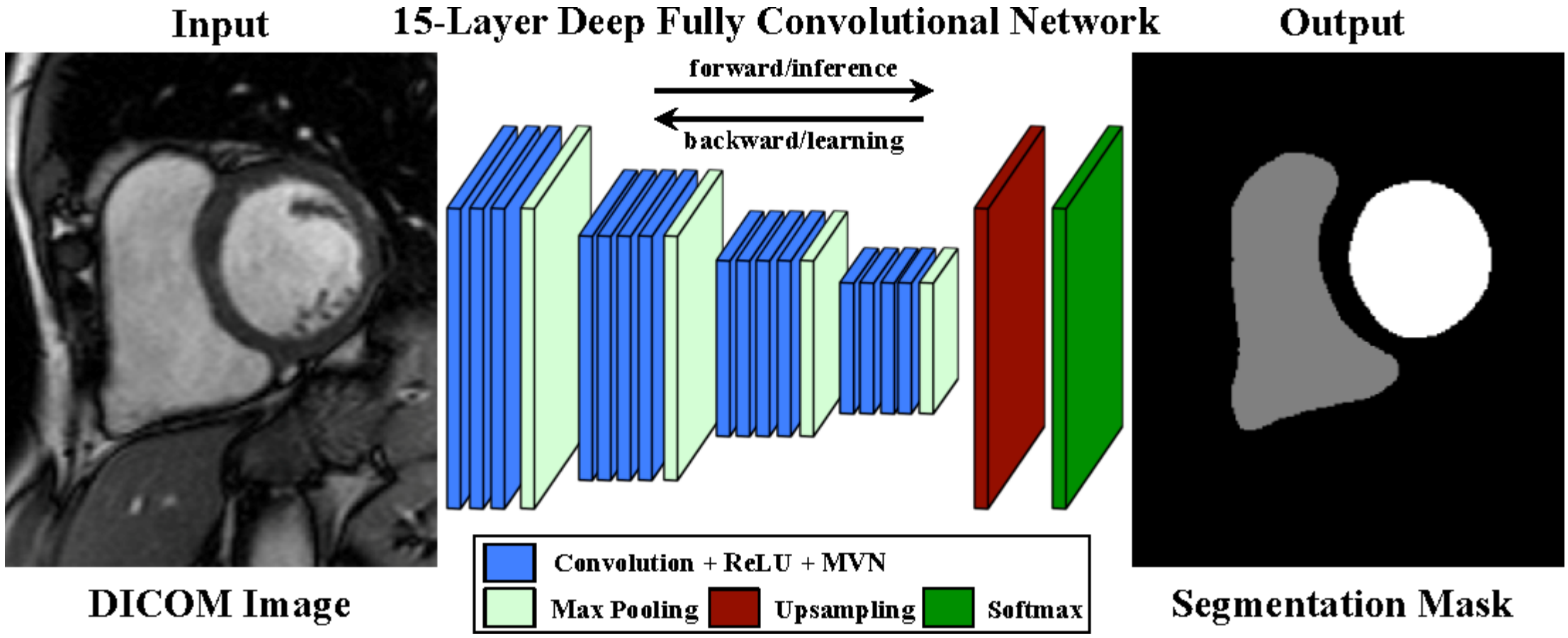

Ronneberger, et al. MICCAI (2015)

# Fully Convolutional Network



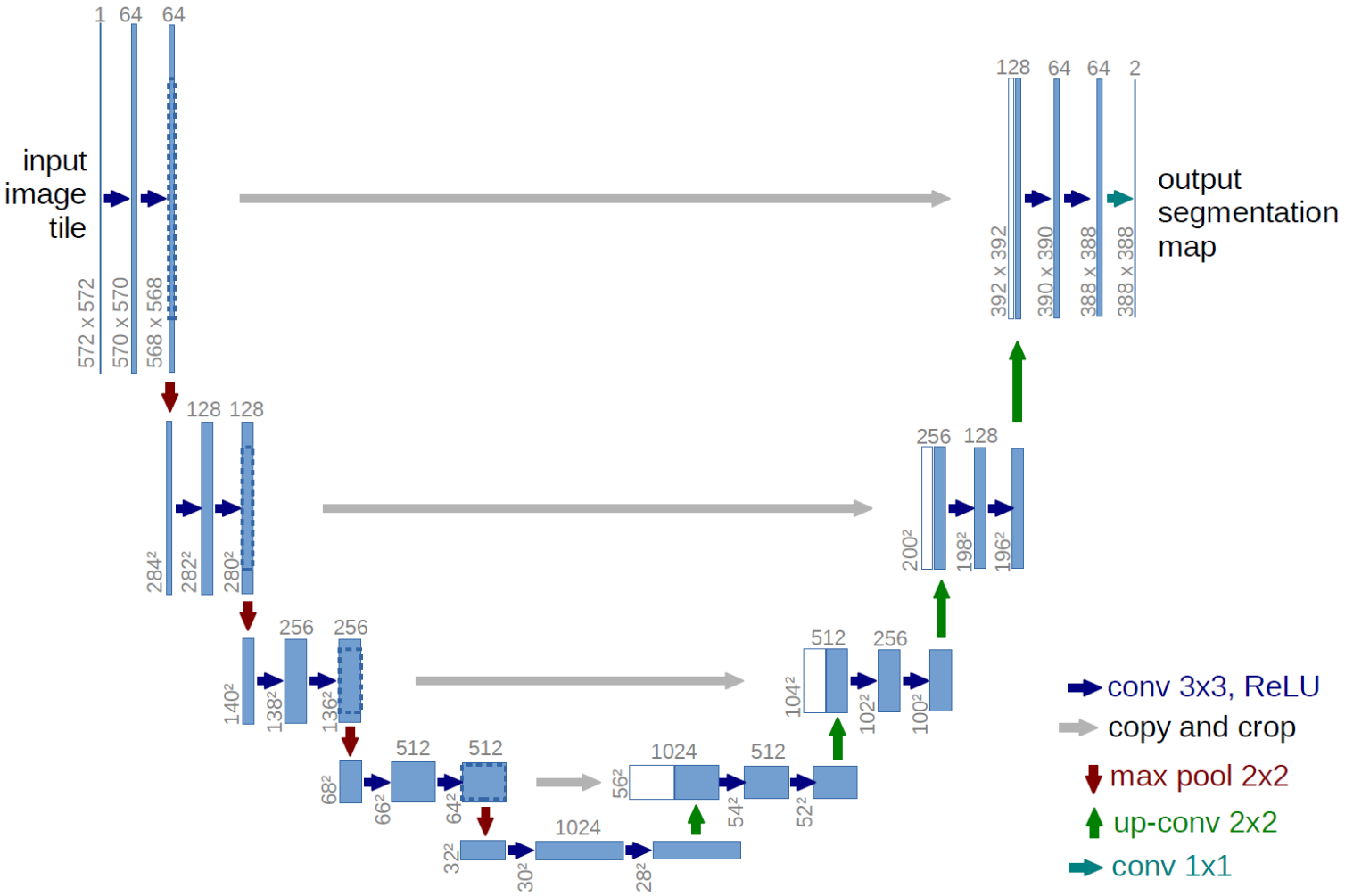

https://medium.com/@wilburdes/semantic-segmentation-using-fully-convolutional-neural-networks-86e45336f99b

# FCN

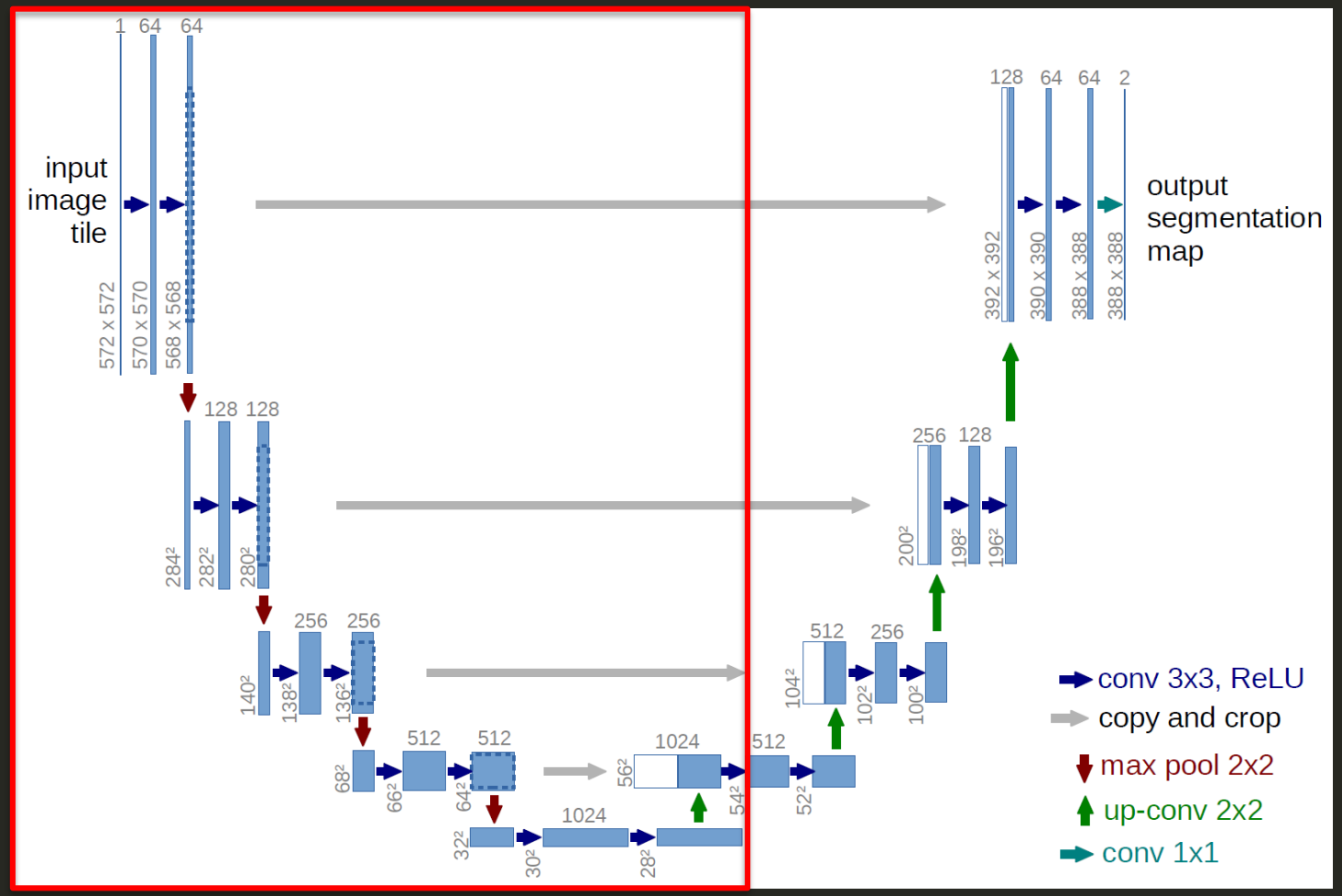



https://arxiv.org/abs/1604.00494

# U-Net

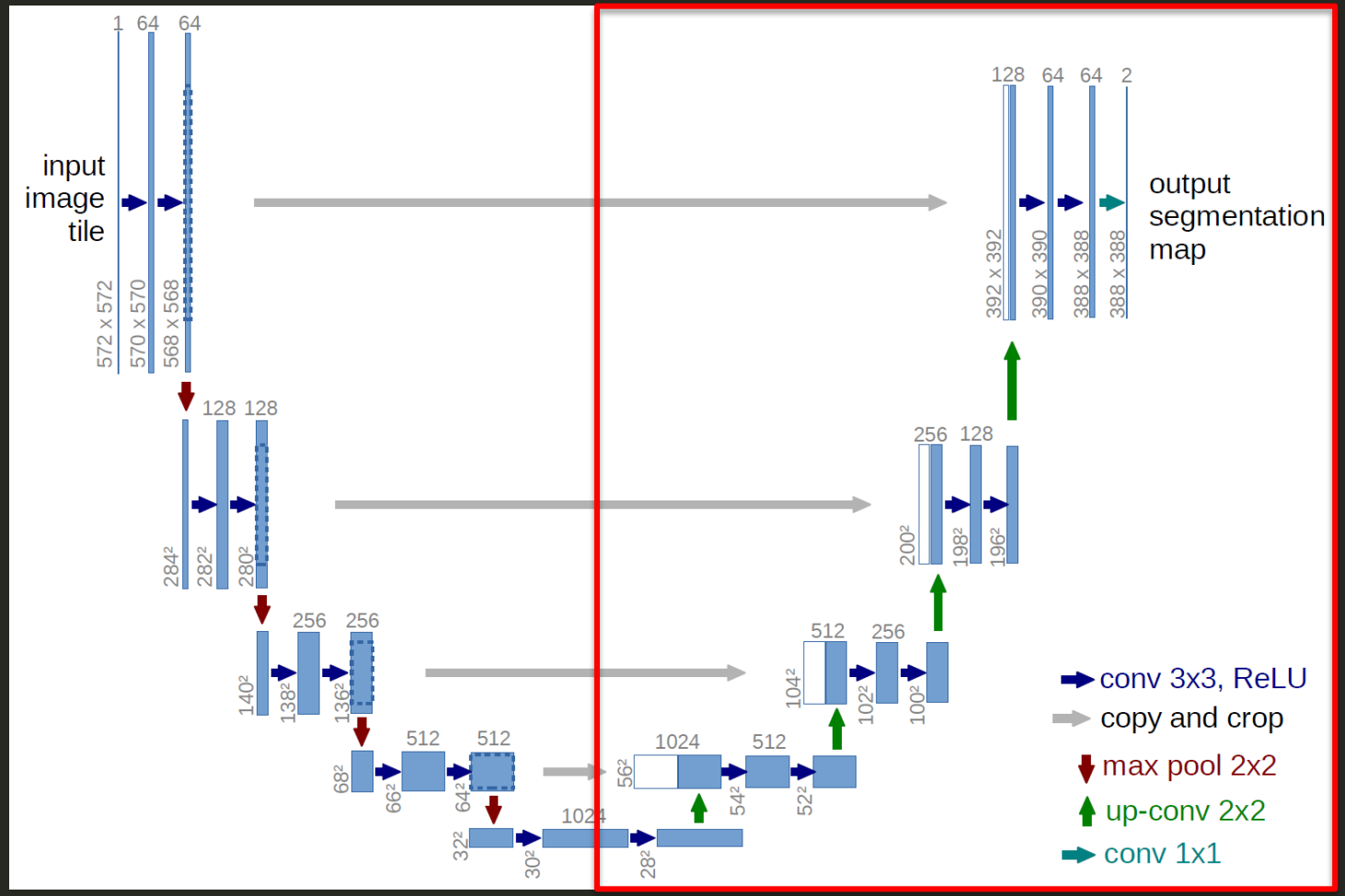

input image tile
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

# Encoder




```
layer1 = F.relu(self.conv1_input(x))
layer1 = F.relu(self.conv1(layer1))

layer2 = F.max_pool2d(layer1, 2)
layer2 = F.relu(self.conv2_input(layer2))
layer2 = F.relu(self.conv2(layer2))

layer3 = F.max_pool2d(layer2, 2)
layer3 = F.relu(self.conv3_input(layer3))
layer3 = F.relu(self.conv3(layer3))

layer4 = F.max_pool2d(layer3, 2)
layer4 = F.relu(self.conv4_input(layer4))
layer4 = F.relu(self.conv4(layer4))

layer5 = F.max_pool2d(layer4, 2)
layer5 = F.relu(self.conv5_input(layer5))
layer5 = F.relu(self.conv5(layer5))
```

# Decoder




```
layer6 = F.relu(self.conv6_up(layer5))
layer6 = torch.cat((layer4, layer6), 1)
layer6 = F.relu(self.conv6_input(layer6))
layer6 = F.relu(self.conv6(layer6))

layer7 = F.relu(self.conv7_up(layer6))
layer7 = torch.cat((layer3, layer7), 1)
layer7 = F.relu(self.conv7_input(layer7))
layer7 = F.relu(self.conv7(layer7))

layer8 = F.relu(self.conv8_up(layer7))
layer8 = torch.cat((layer2, layer8), 1)
layer8 = F.relu(self.conv8_input(layer8))
layer8 = F.relu(self.conv8(layer8))

layer9 = F.relu(self.conv9_up(layer8))
layer9 = torch.cat((layer1, layer9), 1)
layer9 = F.relu(self.conv9_input(layer9))
layer9 = F.relu(self.conv9(layer9))
layer9 = self.final(self.conv9_output(layer9))

return layer9
```

# All Code for U-Net

## Encoder

```
layer1 = F.relu(self.conv1_input(x))
layer1 = F.relu(self.conv1(layer1))

layer2 = F.max_pool2d(layer1, 2)
layer2 = F.relu(self.conv2_input(layer2))
layer2 = F.relu(self.conv2(layer2))

layer3 = F.max_pool2d(layer2, 2)
layer3 = F.relu(self.conv3_input(layer3))
layer3 = F.relu(self.conv3(layer3))

layer4 = F.max_pool2d(layer3, 2)
layer4 = F.relu(self.conv4_input(layer4))
layer4 = F.relu(self.conv4(layer4))

layer5 = F.max_pool2d(layer4, 2)
layer5 = F.relu(self.conv5_input(layer5))
layer5 = F.relu(self.conv5(layer5))
```

## Decoder

```
layer6 = F.relu(self.conv6_up(layer5))
layer6 = torch.cat((layer4, layer6), 1)
layer6 = F.relu(self.conv6_input(layer6))
layer6 = F.relu(self.conv6(layer6))

layer7 = F.relu(self.conv7_up(layer6))
layer7 = torch.cat((layer3, layer7), 1)
layer7 = F.relu(self.conv7_input(layer7))
layer7 = F.relu(self.conv7(layer7))

layer8 = F.relu(self.conv8_up(layer7))
layer8 = torch.cat((layer2, layer8), 1)
layer8 = F.relu(self.conv8_input(layer8))
layer8 = F.relu(self.conv8(layer8))

layer9 = F.relu(self.conv9_up(layer8))
layer9 = torch.cat((layer1, layer9), 1)
layer9 = F.relu(self.conv9_input(layer9))
layer9 = F.relu(self.conv9(layer9))
layer9 = self.final(self.conv9_output(layer9))

return layer9
```

## Parameters

```
self.conv1_input =      nn.Conv2d(1, 64, 3, padding=1)
self.conv1 =            nn.Conv2d(64, 64, 3, padding=1)
self.conv2_input =      nn.Conv2d(64, 128, 3, padding=1)
self.conv2 =            nn.Conv2d(128, 128, 3, padding=1)
self.conv3_input =      nn.Conv2d(128, 256, 3, padding=1)
self.conv3 =            nn.Conv2d(256, 256, 3, padding=1)
self.conv4_input =      nn.Conv2d(256, 512, 3, padding=1)
self.conv4 =            nn.Conv2d(512, 512, 3, padding=1)
self.conv5_input =      nn.Conv2d(512, 1024, 3, padding=1)
self.conv5 =            nn.Conv2d(1024, 1024, 3, padding=1)

self.conv6_up =         nn.ConvTranspose2d(1024, 512, 2, 2)
self.conv6_input =      nn.Conv2d(1024, 512, 3, padding=1)
self.conv6 =            nn.Conv2d(512, 512, 3, padding=1)
self.conv7_up =         nn.ConvTranspose2d(512, 256, 2, 2)
self.conv7_input =      nn.Conv2d(512, 256, 3, padding=1)
self.conv7 =            nn.Conv2d(256, 256, 3, padding=1)
self.conv8_up =         nn.ConvTranspose2d(256, 128, 2, 2)
self.conv8_input =      nn.Conv2d(256, 128, 3, padding=1)
self.conv8 =            nn.Conv2d(128, 128, 3, padding=1)
self.conv9_up =         nn.ConvTranspose2d(128, 64, 2, 2)
self.conv9_input =      nn.Conv2d(128, 64, 3, padding=1)
self.conv9 =            nn.Conv2d(64, 64, 3, padding=1)
self.conv9_output =     nn.Conv2d(64, 2, 1)
```

https://github.com/jonnedtc/U-Net-PyTorch/blob/master/networks.py

# Convolution 2D

**torch.nn.Conv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*)

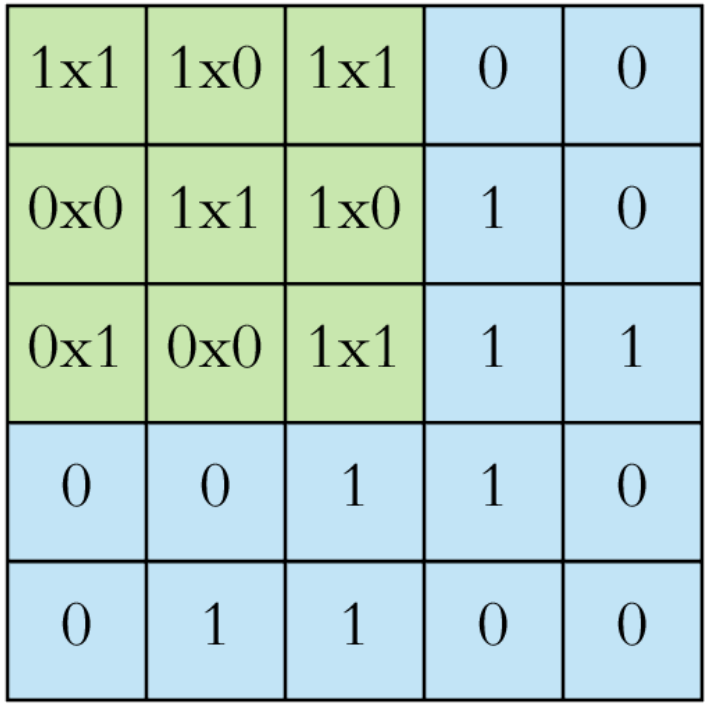
```
layer1 = F.relu(self.conv1_input(x))
layer1 = F.relu(self.conv1(layer1))

layer2 = F.max_pool2d(layer1, 2)
layer2 = F.relu(self.conv2_input(layer2))
layer2 = F.relu(self.conv2(layer2))

layer3 = F.max_pool2d(layer2, 2)
layer3 = F.relu(self.conv3_input(layer3))
layer3 = F.relu(self.conv3(layer3))

layer4 = F.max_pool2d(layer3, 2)
layer4 = F.relu(self.conv4_input(layer4))
layer4 = F.relu(self.conv4(layer4))

layer5 = F.max_pool2d(layer4, 2)
layer5 = F.relu(self.conv5_input(layer5))
layer5 = F.relu(self.conv5(layer5))
```

- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) – Size of the convolving kernel
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1
- **padding** (*int or tuple, optional*) – Zero-padding added to both sides of the input. Default: 0
- **dilation** (*int or tuple, optional*) – Spacing between kernel elements. Default: 1
- **groups** (*int, optional*) – Number of blocked connections from input channels to output channels. Default: 1
- **bias** (*bool, optional*) – If `True`, adds a learnable bias to the output. Default: `True`

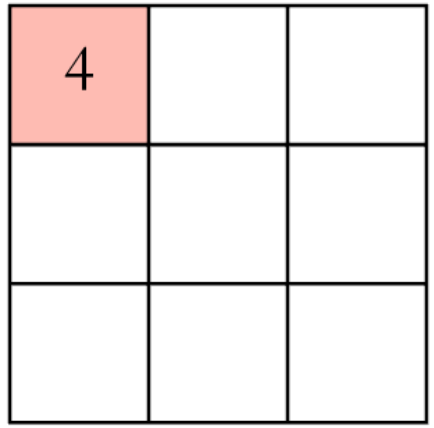
https://pytorch.org/docs/stable/nn.html

# Convolution

**torch.nn.Conv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*)




Input x Filter

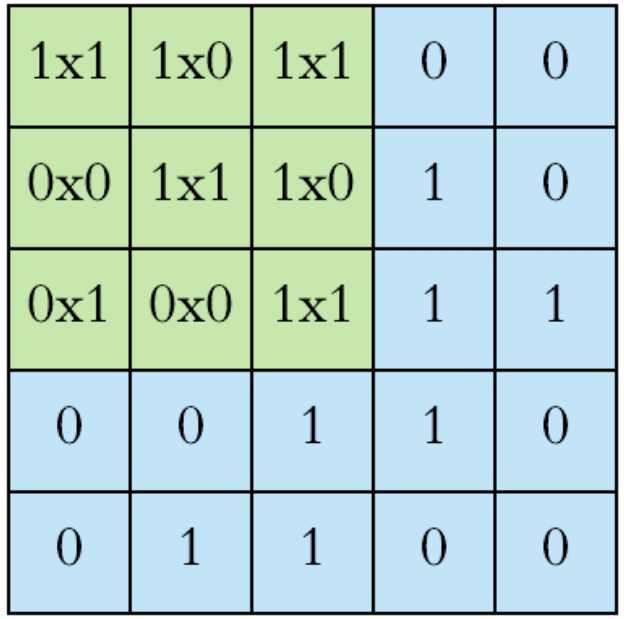


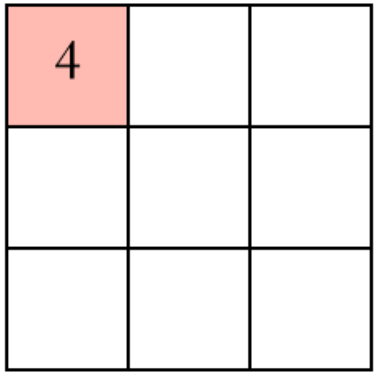

Feature Map

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

# Convolution

**torch.nn.Conv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*)

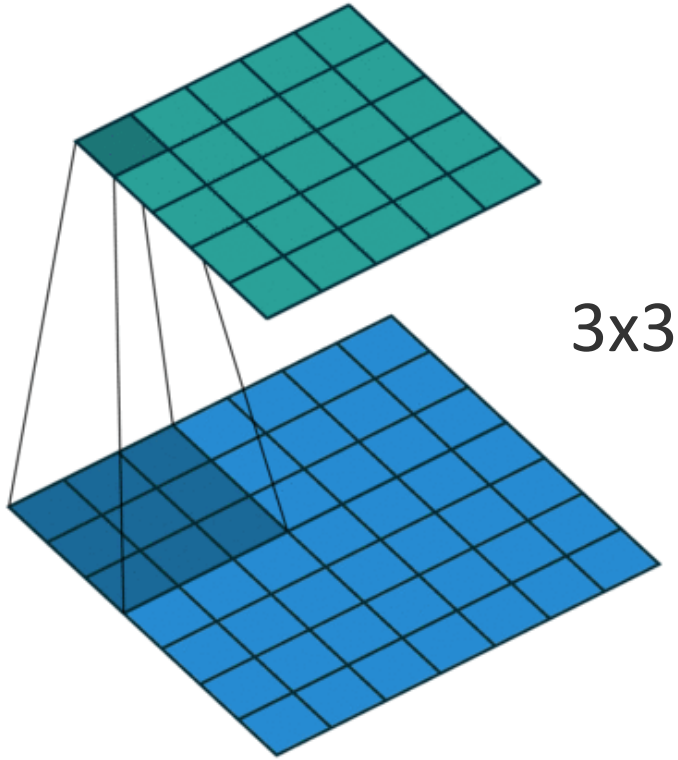


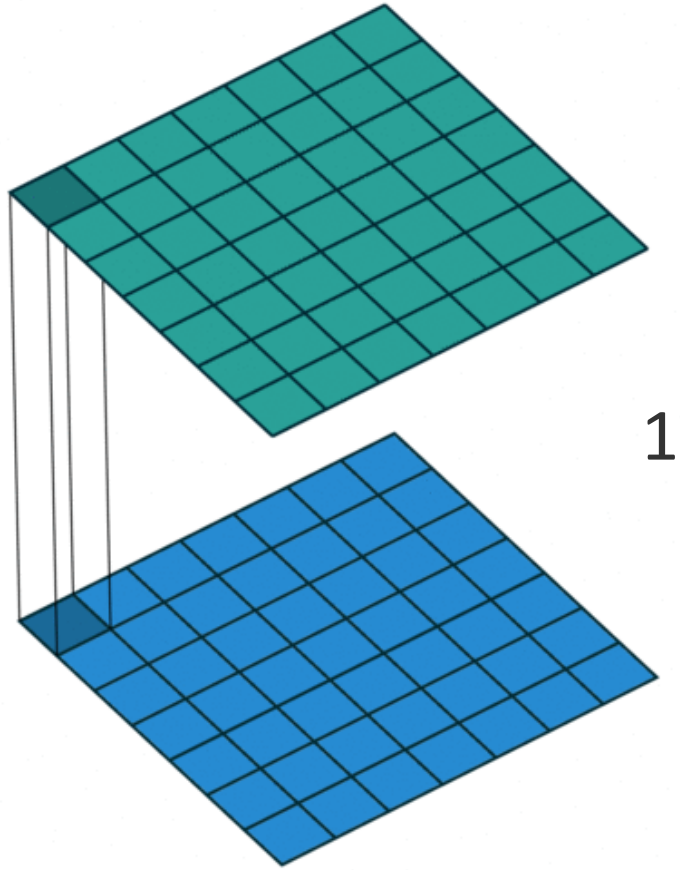

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

# Kernel Size

**torch.nn.Conv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*)



3x3

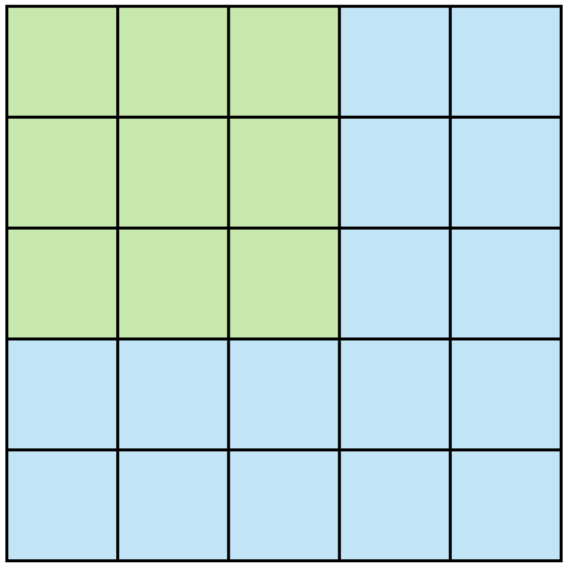


1x1

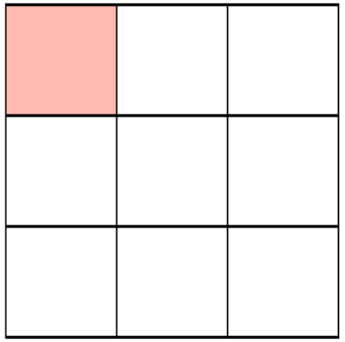
http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html

# Stride

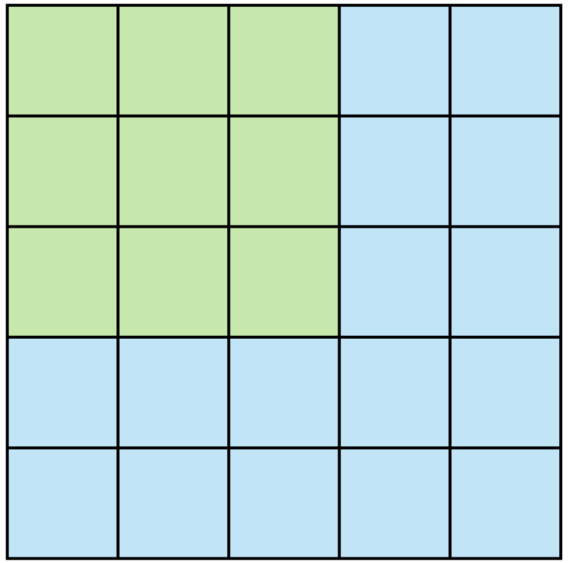
**torch.nn.Conv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*)



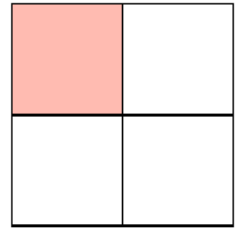
Stride 1



Feature Map



Stride 2

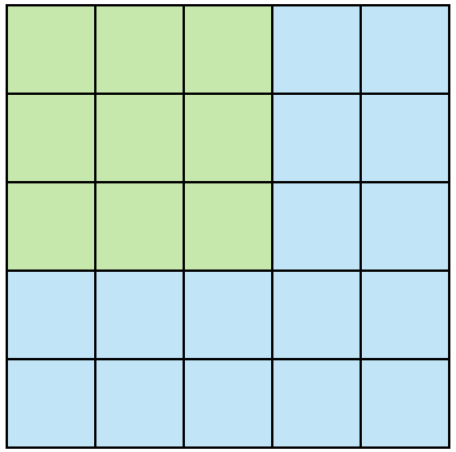


Feature Map

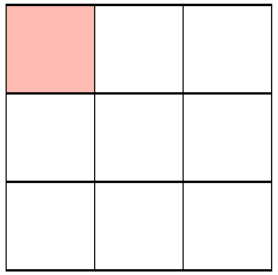
https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

# Padding

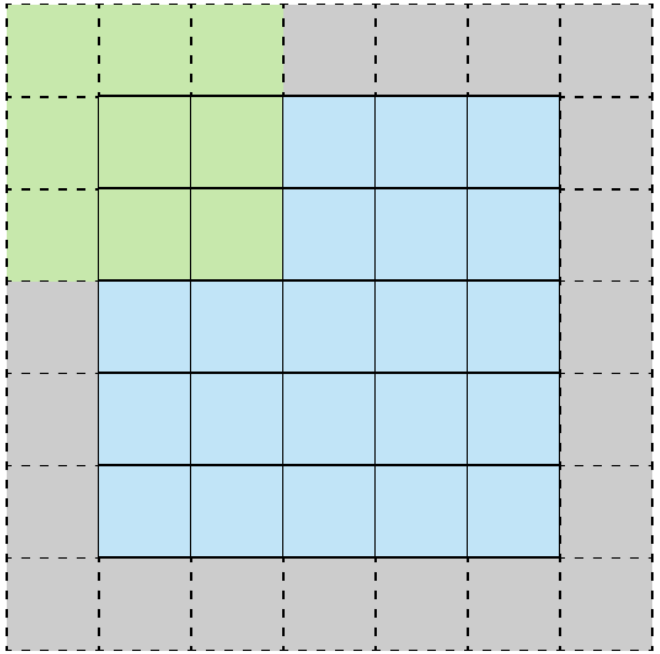
**torch.nn.Conv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*)



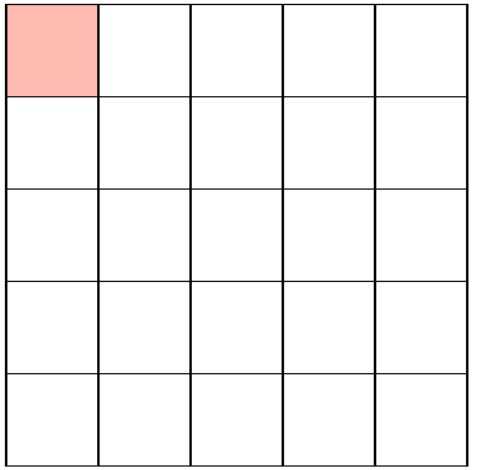
Stride 1



Feature Map



Stride 1 with Padding

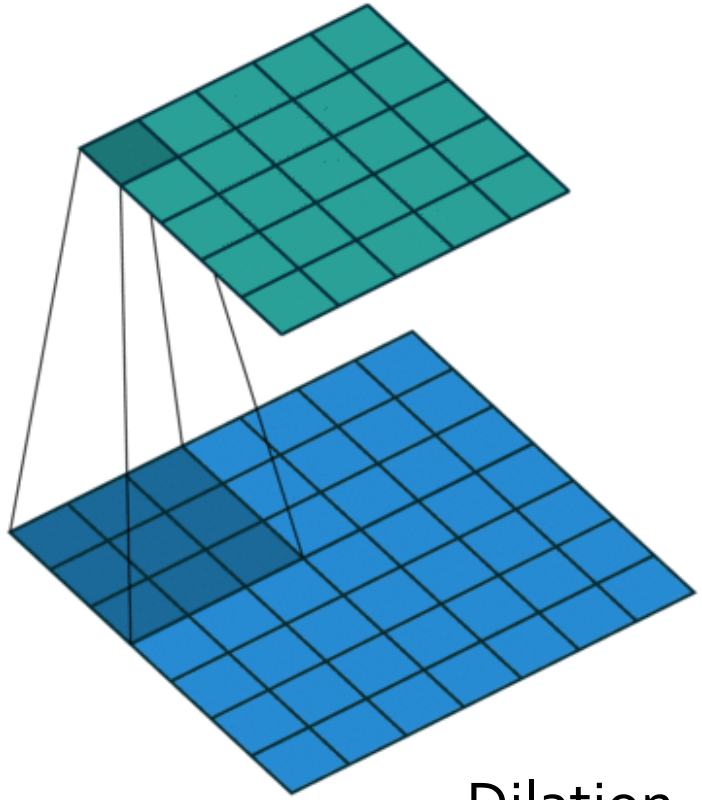


Feature Map

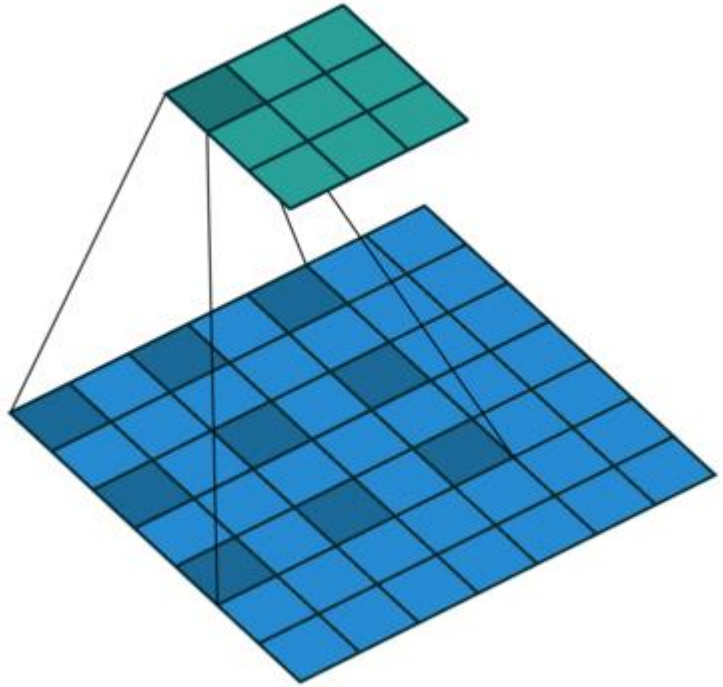
https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

# Dilation (typically default)

**torch.nn.Conv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*)



Dilation = 1

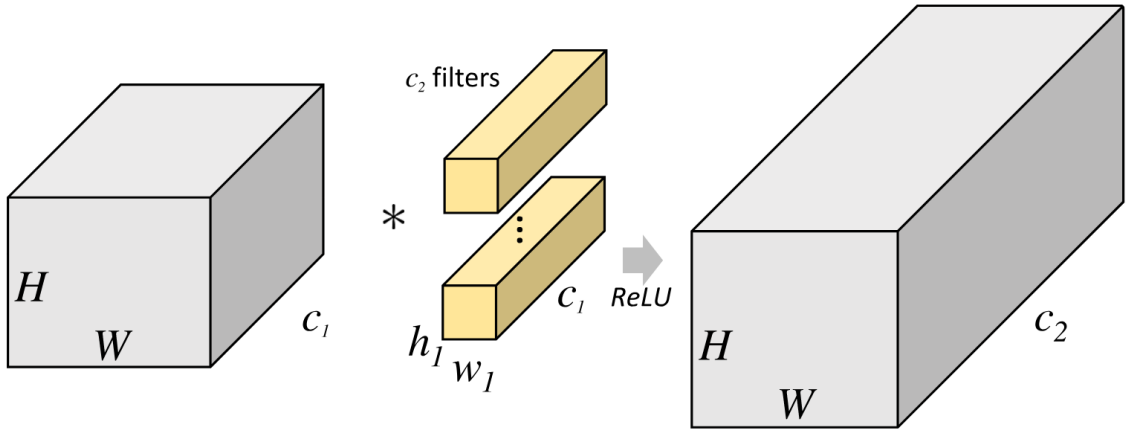


Dilation = 2

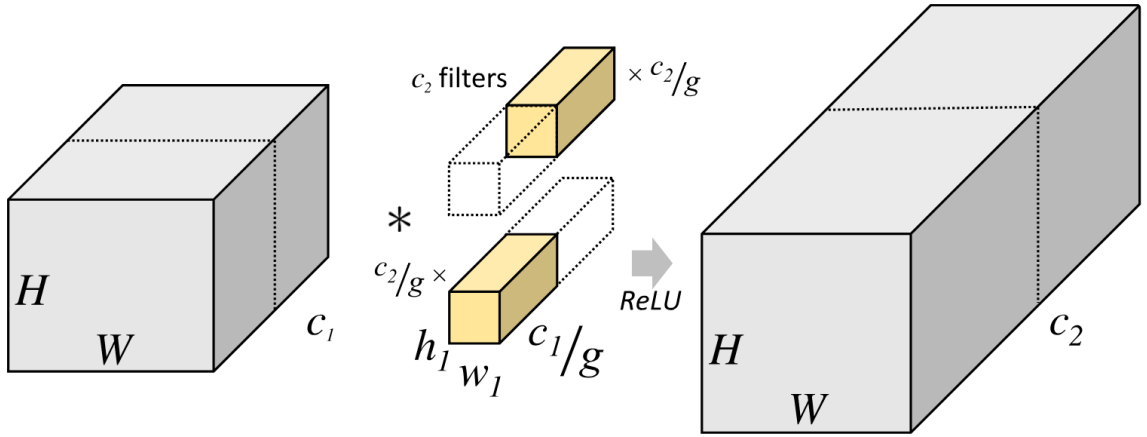
https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md

# Groups (typically default)

**torch.nn.Conv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*)




A normal convolutional layer. Yellow blocks represent learned parameters, gray blocks represent feature maps/input images (working memory).



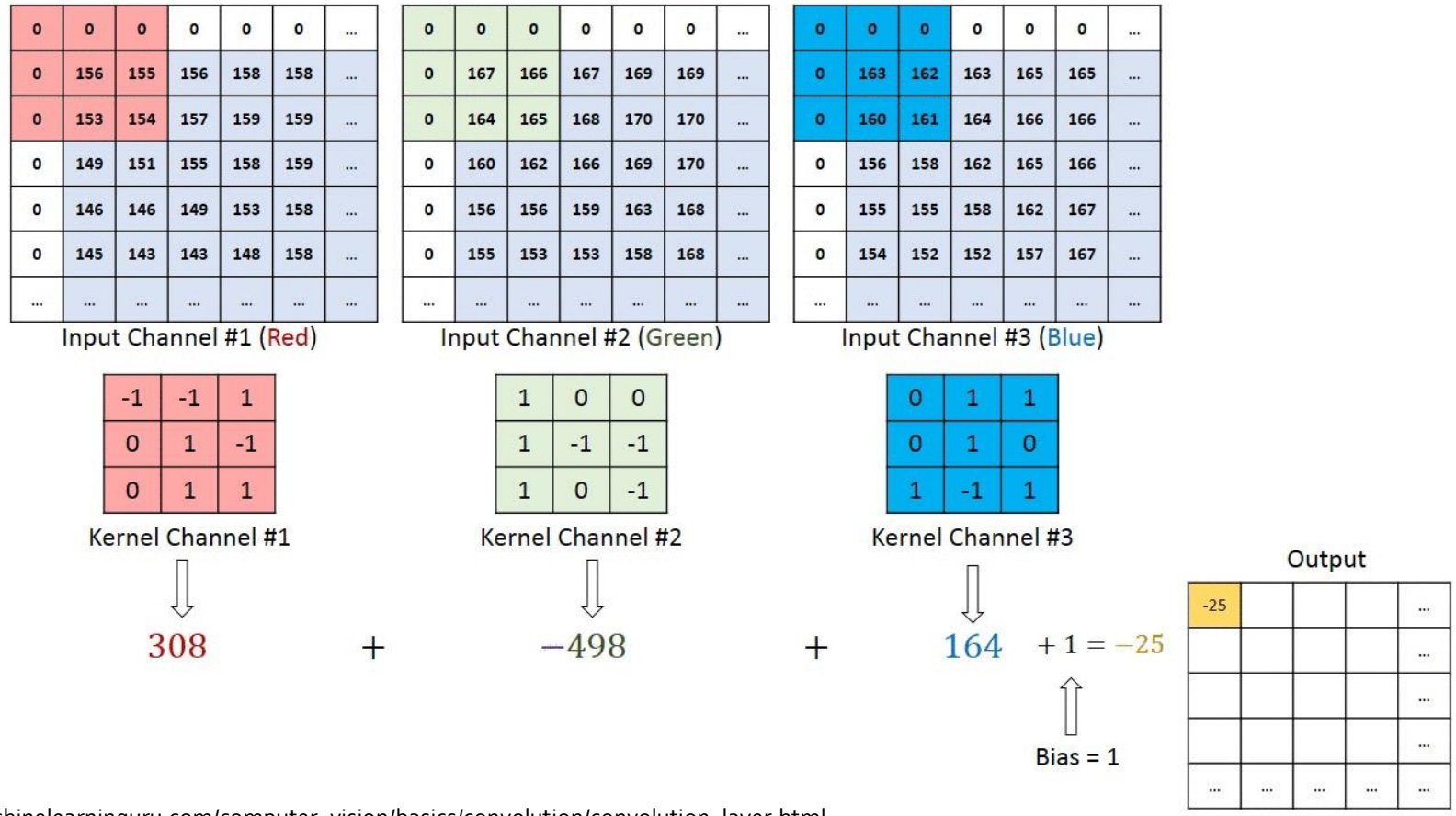

A convolutional layer with 2 filter groups. Note that each of the filters in the grouped convolutional layer is now exactly half the depth, i.e. half the parameters and half the compute as the original filter.

Groups = 1

Groups = 2

# Input Channels

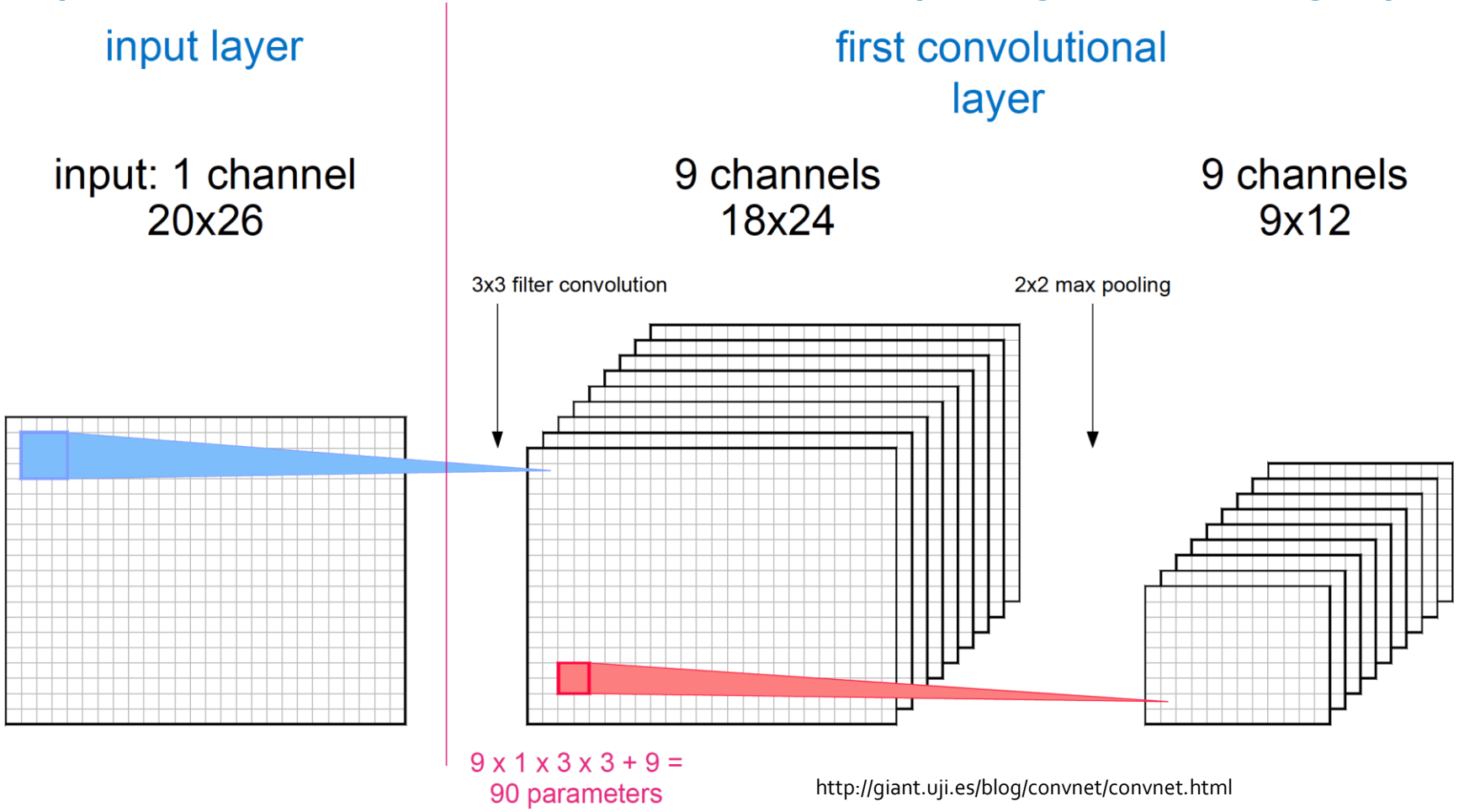
**torch.nn.Conv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*)




http://machinelearninguru.com/computer_vision/basics/convolution/convolution_layer.html

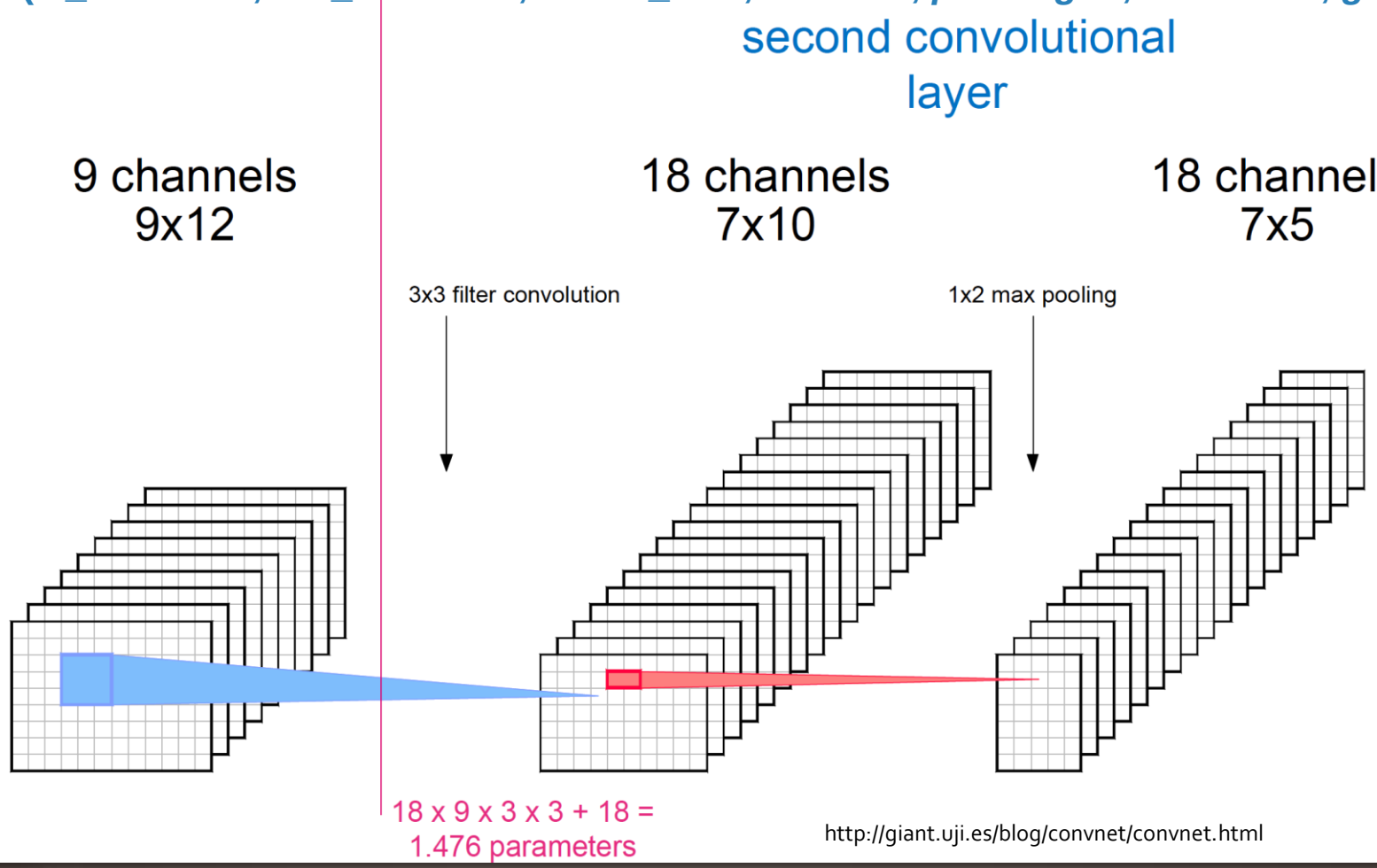
# Output Channels

**torch.nn.Conv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*)



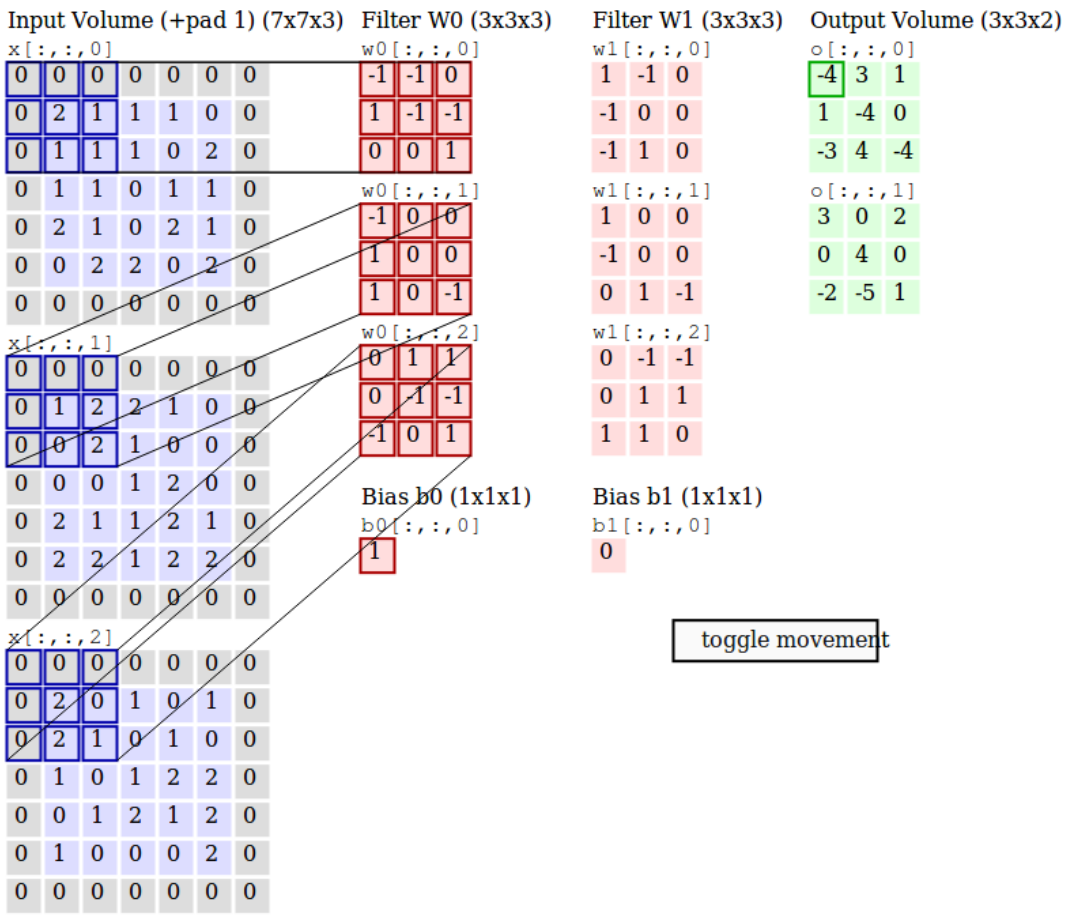
# Multiple Input and Output Channels

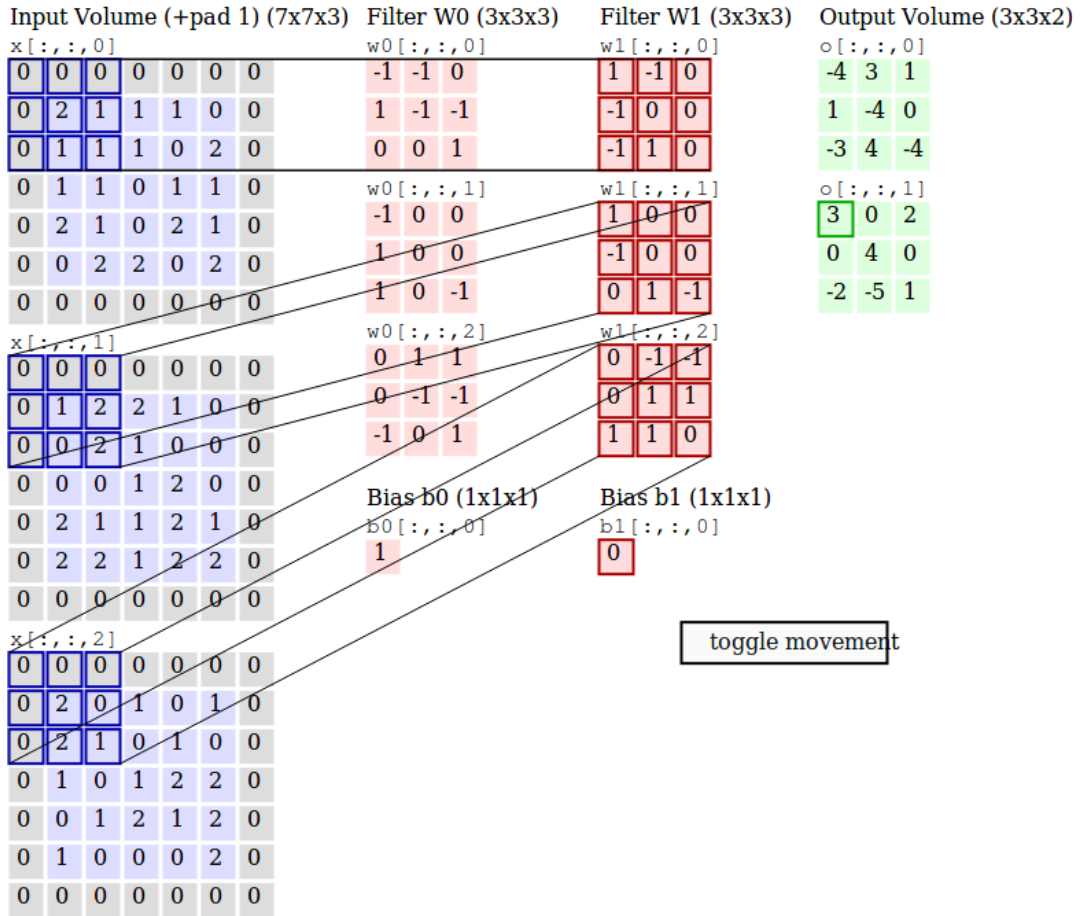
**torch.nn.Conv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*)



# Bias

**torch.nn.Conv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*)

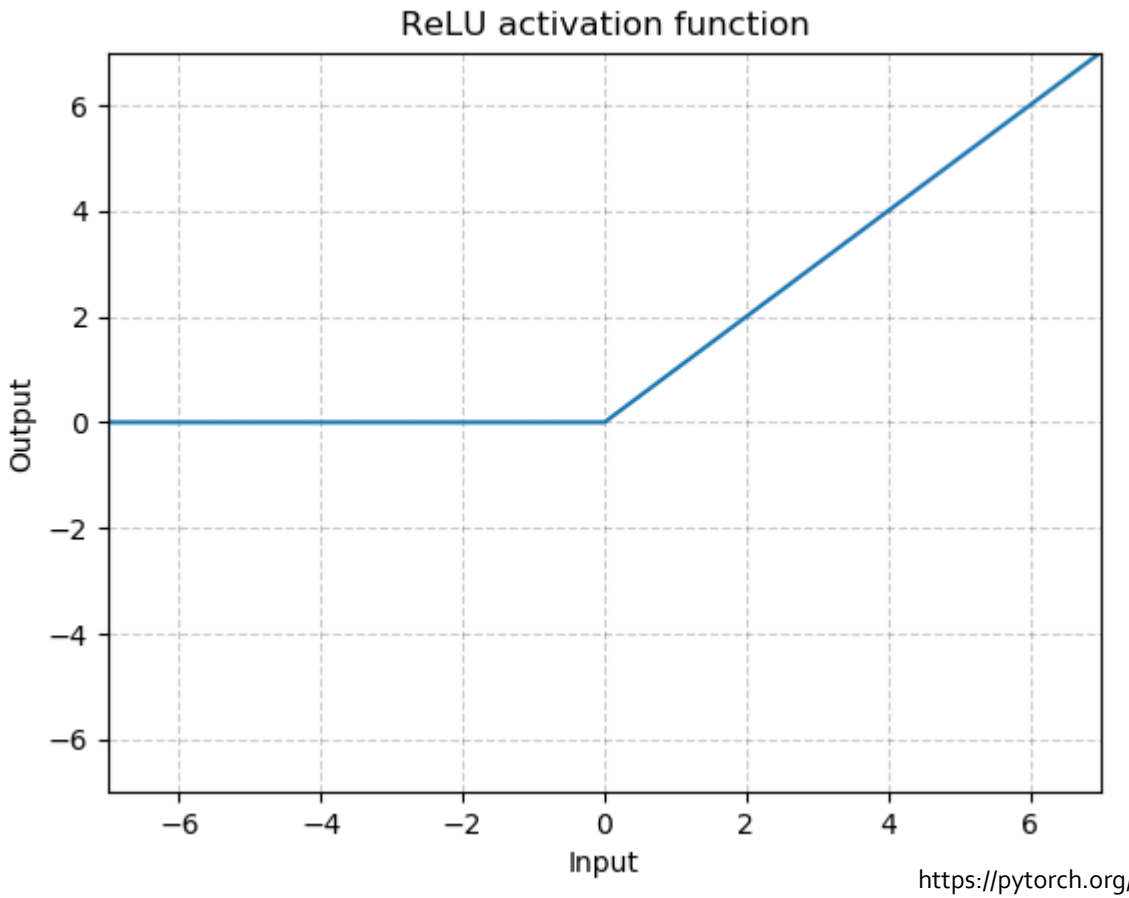


# Relu

**torch.nn.ReLU(inplace=False)**

```
layer1 = F.relu(self.conv1_input(x))
layer1 = F.relu(self.conv1(layer1))

layer2 = F.max_pool2d(layer1, 2)
layer2 = F.relu(self.conv2_input(layer2))
layer2 = F.relu(self.conv2(layer2))

layer3 = F.max_pool2d(layer2, 2)
layer3 = F.relu(self.conv3_input(layer3))
layer3 = F.relu(self.conv3(layer3))

layer4 = F.max_pool2d(layer3, 2)
layer4 = F.relu(self.conv4_input(layer4))
layer4 = F.relu(self.conv4(layer4))

layer5 = F.max_pool2d(layer4, 2)
layer5 = F.relu(self.conv5_input(layer5))
layer5 = F.relu(self.conv5(layer5))
```




https://pytorch.org/docs/stable/nn.html

# Max Pooling

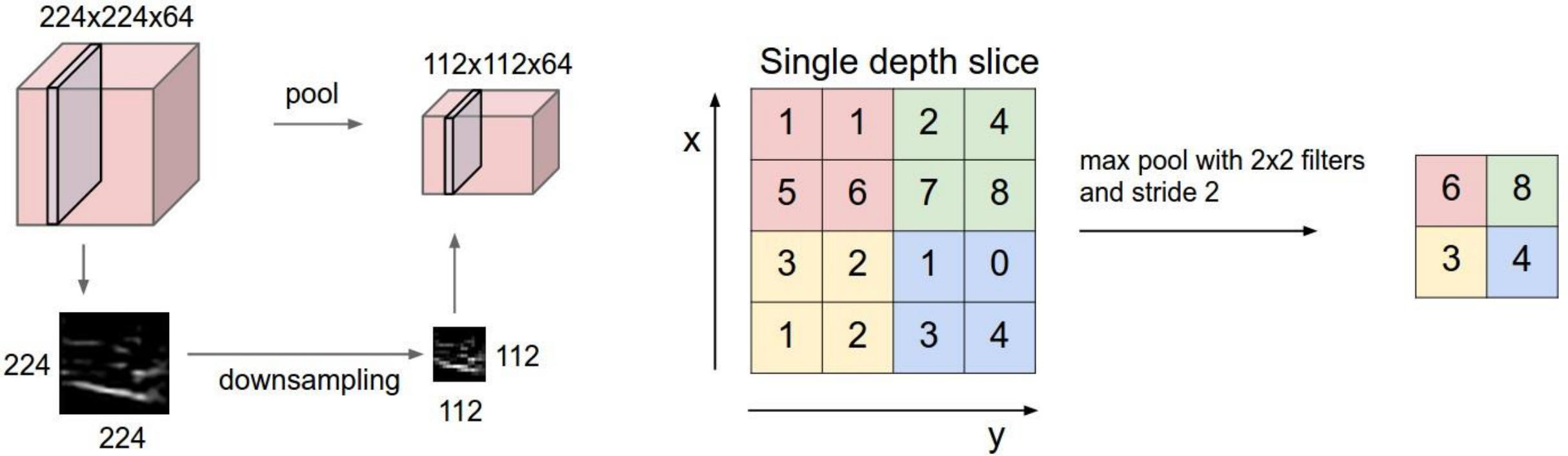
**torch.nn.MaxPool2d**(*kernel_size*, *stride=None*, *padding=0*, *dilation=1*, *return_indices=False*, *ceil_mode=False*)

```
layer1 = F.relu(self.conv1_input(x))
layer1 = F.relu(self.conv1(layer1))

layer2 = F.max_pool2d(layer1, 2)
layer2 = F.relu(self.conv2_input(layer2))
layer2 = F.relu(self.conv2(layer2))

layer3 = F.max_pool2d(layer2, 2)
layer3 = F.relu(self.conv3_input(layer3))
layer3 = F.relu(self.conv3(layer3))

layer4 = F.max_pool2d(layer3, 2)
layer4 = F.relu(self.conv4_input(layer4))
layer4 = F.relu(self.conv4(layer4))

layer5 = F.max_pool2d(layer4, 2)
layer5 = F.relu(self.conv5_input(layer5))
layer5 = F.relu(self.conv5(layer5))
```

- **kernel_size** – the size of the window to take a max over
- **stride** – the stride of the window. Default value is `kernel_size`
- **padding** – implicit zero padding to be added on both sides
- **dilation** – a parameter that controls the stride of elements in the window
- **return_indices** – if `True`, will return the max indices along with the outputs. Useful when Unpooling later
- **ceil_mode** – when True, will use *ceil* instead of *floor* to compute the output shape

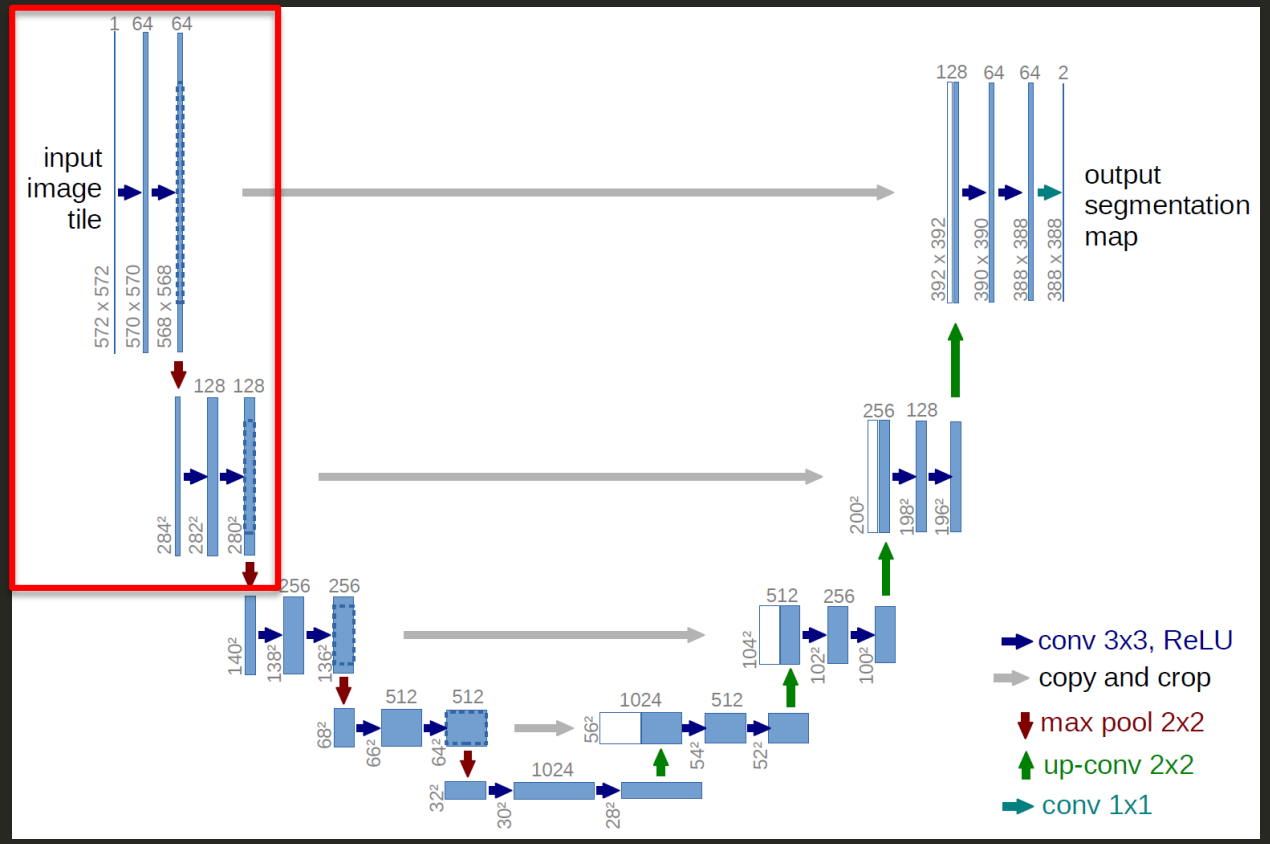
# Max Pooling (kernel size = 2)

**torch.nn.MaxPool2d**(*kernel_size*, *stride=None*, *padding=0*, *dilation=1*, *return_indices=False*, *ceil_mode=False*)




https://github.com/cs231n/cs231n.github.io/blob/master/convolutional-networks.md

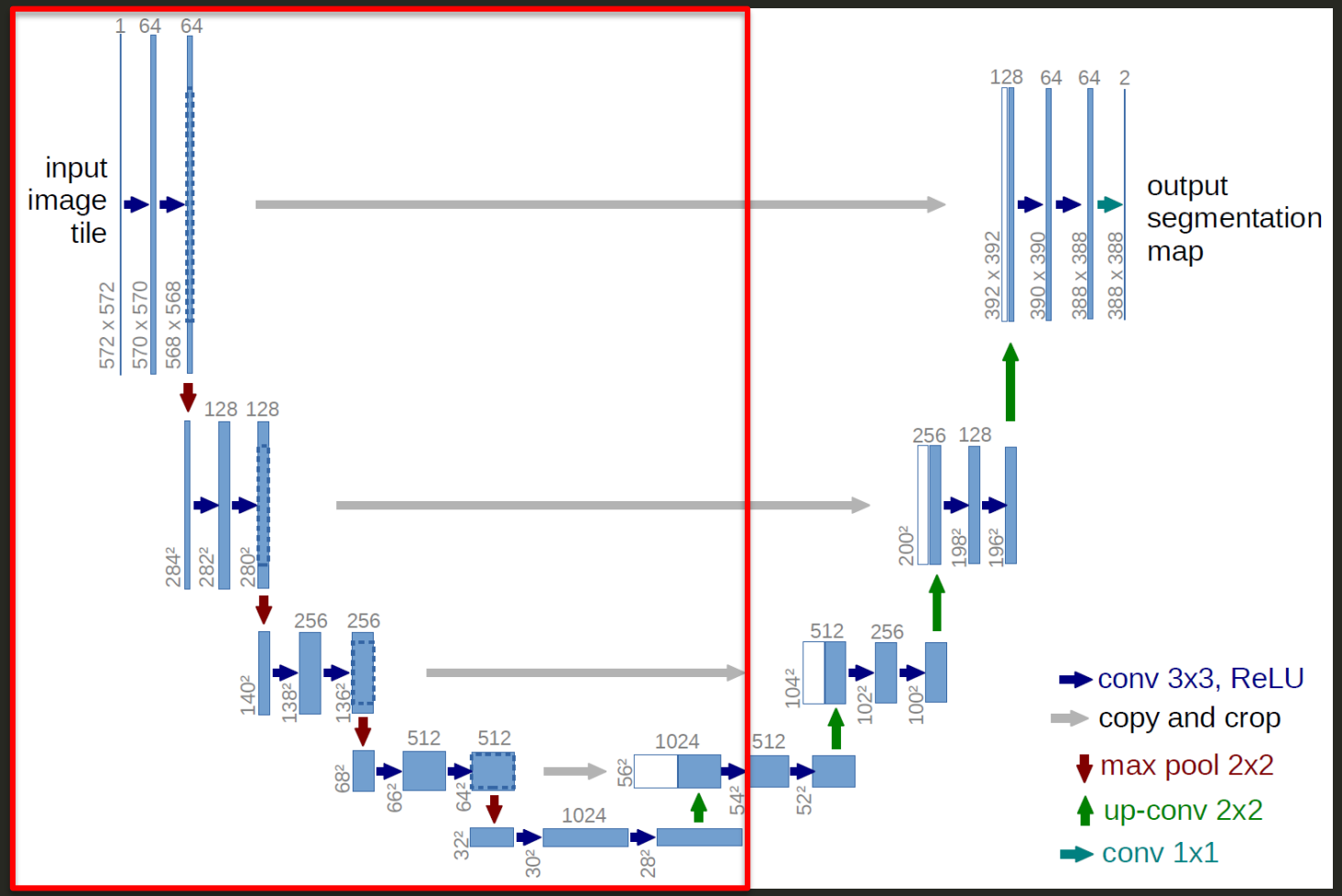
# Encoder




```
layer1 = F.relu(self.conv1_input(x))
layer1 = F.relu(self.conv1(layer1))

layer2 = F.max_pool2d(layer1, 2)
layer2 = F.relu(self.conv2_input(layer2))
layer2 = F.relu(self.conv2(layer2))

layer3 = F.max_pool2d(layer2, 2)
```

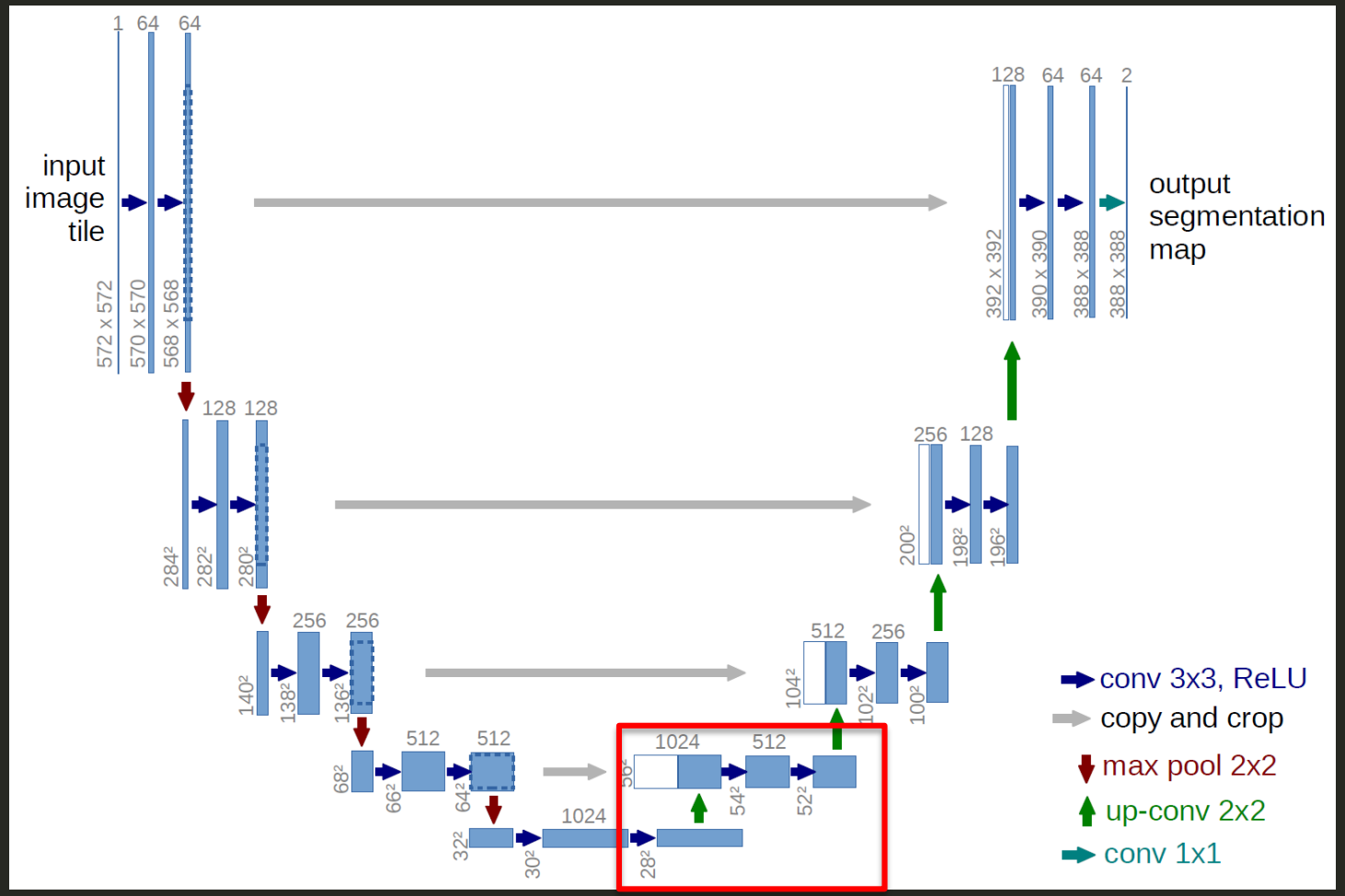
```
self.conv1_input =      nn.Conv2d(1, 64, 3, padding=1)
self.conv1 =            nn.Conv2d(64, 64, 3, padding=1)
self.conv2_input =      nn.Conv2d(64, 128, 3, padding=1)
self.conv2 =            nn.Conv2d(128, 128, 3, padding=1)
```

# Encoder




```
layer1 = F.relu(self.conv1_input(x))
layer1 = F.relu(self.conv1(layer1))

layer2 = F.max_pool2d(layer1, 2)
layer2 = F.relu(self.conv2_input(layer2))
layer2 = F.relu(self.conv2(layer2))

layer3 = F.max_pool2d(layer2, 2)
layer3 = F.relu(self.conv3_input(layer3))
layer3 = F.relu(self.conv3(layer3))

layer4 = F.max_pool2d(layer3, 2)
layer4 = F.relu(self.conv4_input(layer4))
layer4 = F.relu(self.conv4(layer4))

layer5 = F.max_pool2d(layer4, 2)
layer5 = F.relu(self.conv5_input(layer5))
layer5 = F.relu(self.conv5(layer5))
```

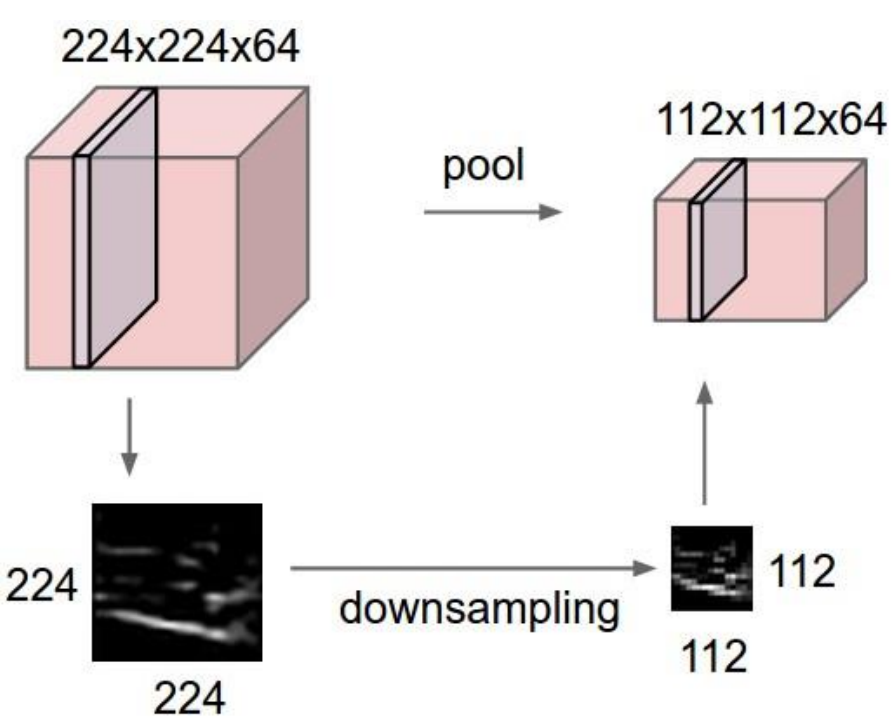
# Decoder



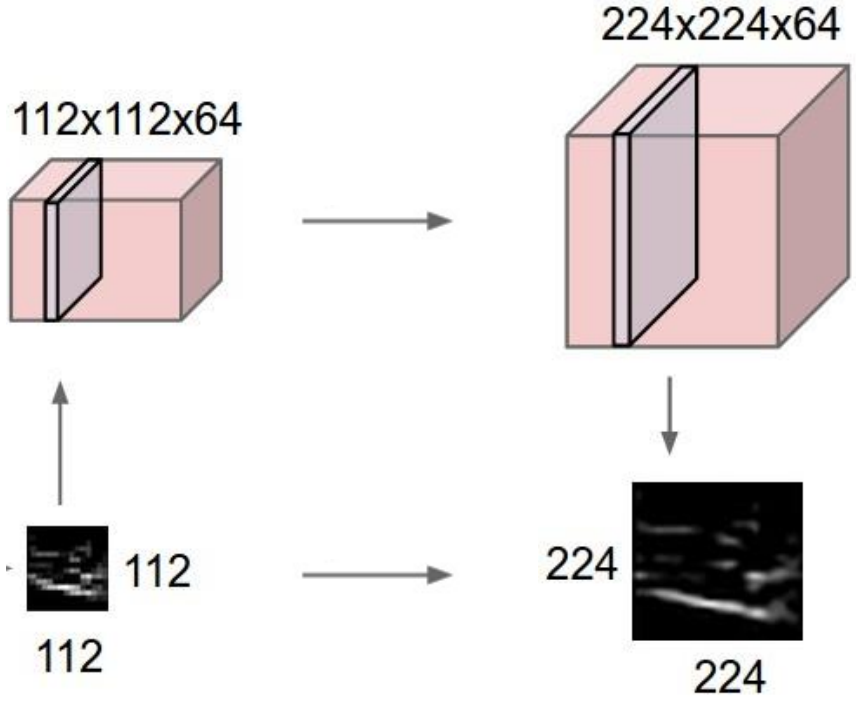

```
layer6 = F.relu(self.conv6_up(layer5))
layer6 = torch.cat((layer4, layer6), 1)
layer6 = F.relu(self.conv6_input(layer6))
layer6 = F.relu(self.conv6(layer6))
```

```
self.conv6_up =        nn.ConvTranspose2d(1024, 512, 2, 2)
self.conv6_input =     nn.Conv2d(1024, 512, 3, padding=1)
self.conv6 =           nn.Conv2d(512, 512, 3, padding=1)
```

# Upsampling




Down Sampling



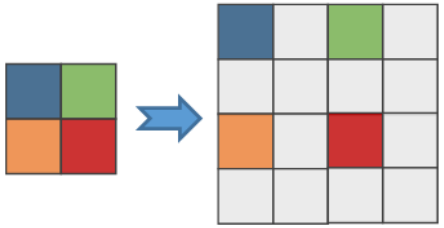

Up Sampling

https://github.com/cs231n/cs231n.github.io/blob/master/convolutional-networks.md

# Unpooling

**torch.nn.MaxUnpool2d(*kernel_size*, *stride=None*, *padding=0*)**

2x2 unpooling:



Input

| | | | | | |
|---|---|---|---|---|---|
| 0.8 | 0 | 3.3 | 0 | 4 | 1 |
| 1.2 | 1.2 | 0 | 2.1 | 0 | 0 |
| 0 | 3 | 0.2 | 3.1 | 5.2 | 0 |

→ Max-pooling [3, 3] → Value: 3.3 5.2 →

Inverse

| | | | | | |
|---|---|---|---|---|---|
| 3.3 | 0 | 0 | 5.2 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

Input

| | | | | | |
|---|---|---|---|---|---|
| 0.8 | 0 | 3.3 | 0 | 4 | 1 |
| 1.2 | 1.2 | 0 | 2.1 | 0 | 0 |
| 0 | 3 | 0.2 | 3.1 | 5.2 | 0 |

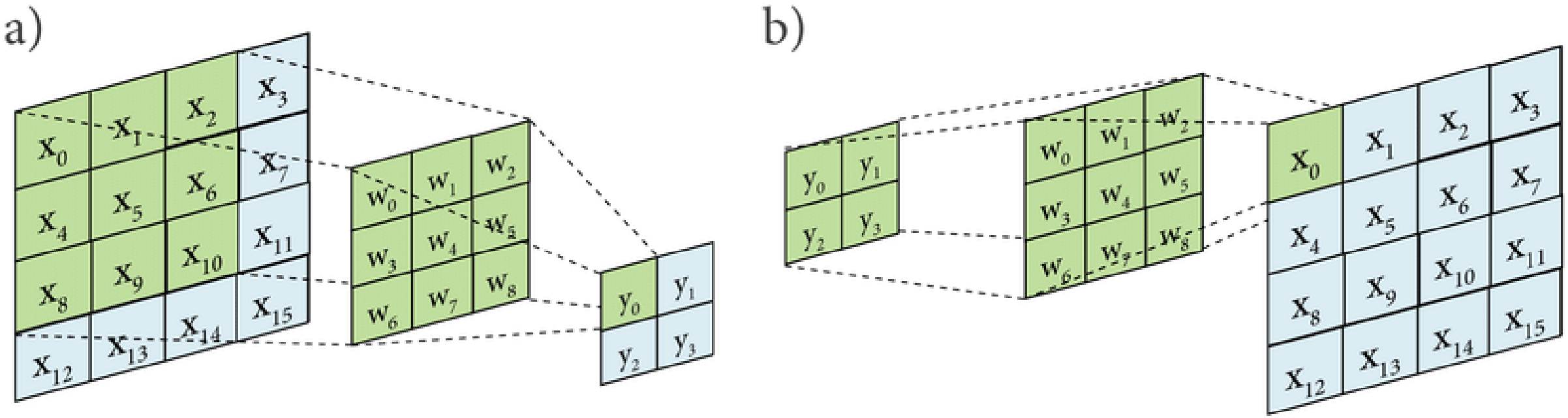
→ Max-pooling [3, 3] → Value: 3.3 5.2, Position: (0,2) (2,1) →

Unpooling

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 3.3 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 5.2 | 0 |

https://stackoverflow.com/questions/36548736/tensorflow-unpooling

# Decoder




https://www.researchgate.net/figure/Example-of-a-discrete-convolution-a-and-equivalent-transposed-convolution-operation-b_fig2_321719286

# Deconvolution (Transpose Convolution)

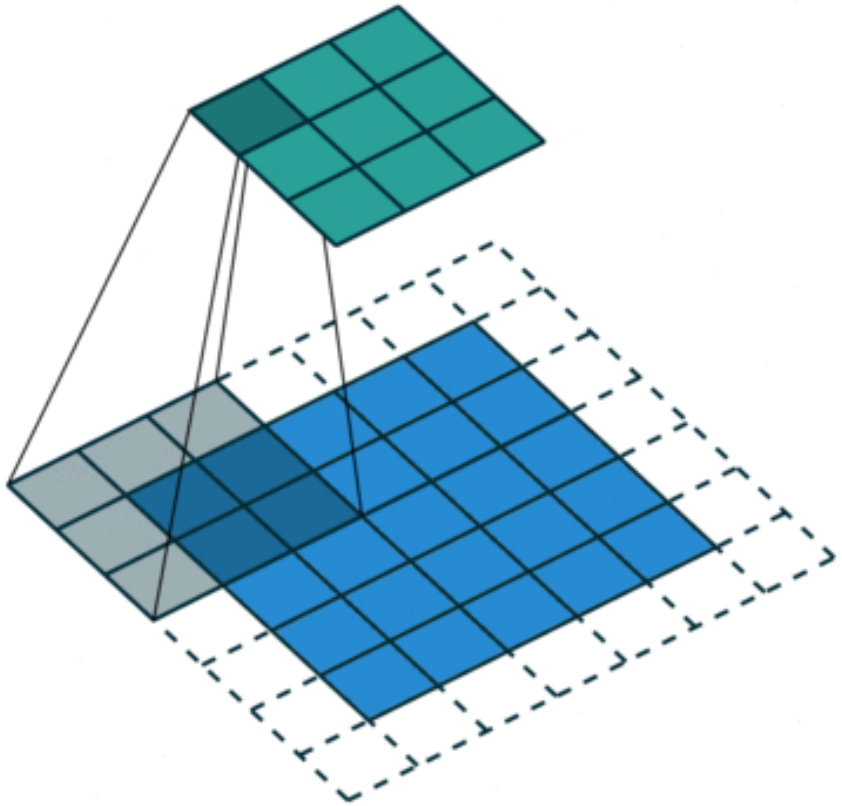
**torch.nn.ConvTranspose2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *output_padding=0*, *groups=1*, *bias=True*, *dilation=1*)

```
layer6 = F.relu(self.conv6_up(layer5))
layer6 = torch.cat((layer4, layer6), 1)
layer6 = F.relu(self.conv6_input(layer6))
layer6 = F.relu(self.conv6(layer6))

layer7 = F.relu(self.conv7_up(layer6))
layer7 = torch.cat((layer3, layer7), 1)
layer7 = F.relu(self.conv7_input(layer7))
layer7 = F.relu(self.conv7(layer7))

layer8 = F.relu(self.conv8_up(layer7))
layer8 = torch.cat((layer2, layer8), 1)
layer8 = F.relu(self.conv8_input(layer8))
layer8 = F.relu(self.conv8(layer8))

layer9 = F.relu(self.conv9_up(layer8))
layer9 = torch.cat((layer1, layer9), 1)
layer9 = F.relu(self.conv9_input(layer9))
layer9 = F.relu(self.conv9(layer9))
layer9 = self.final(self.conv9_output(layer9))

return layer9
```

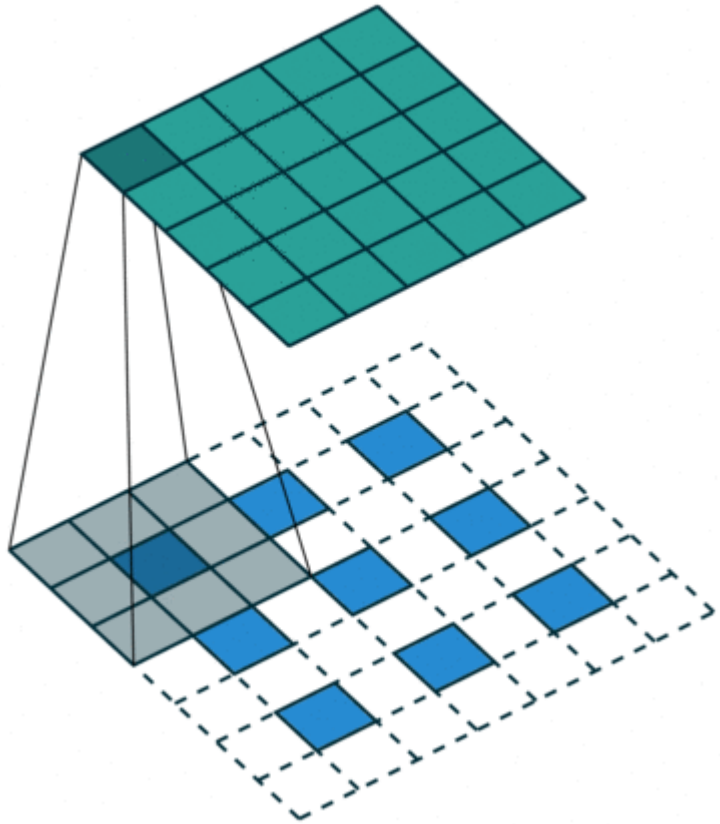
- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) – Size of the convolving kernel
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1
- **padding** (*int or tuple, optional*) – `kernel_size - 1 - padding` zero-padding will be added to both sides of each dimension in the input. Default: 0
- **output_padding** (*int or tuple, optional*) – Additional size added to one side of each dimension in the output shape. Default: 0
- **groups** (*int, optional*) – Number of blocked connections from input channels to output channels. Default: 1
- **bias** (*bool, optional*) – If `True`, adds a learnable bias to the output. Default: `True`
- **dilation** (*int or tuple, optional*) – Spacing between kernel elements. Default: 1

https://pytorch.org/docs/stable/nn.html

# Transpose Convolution



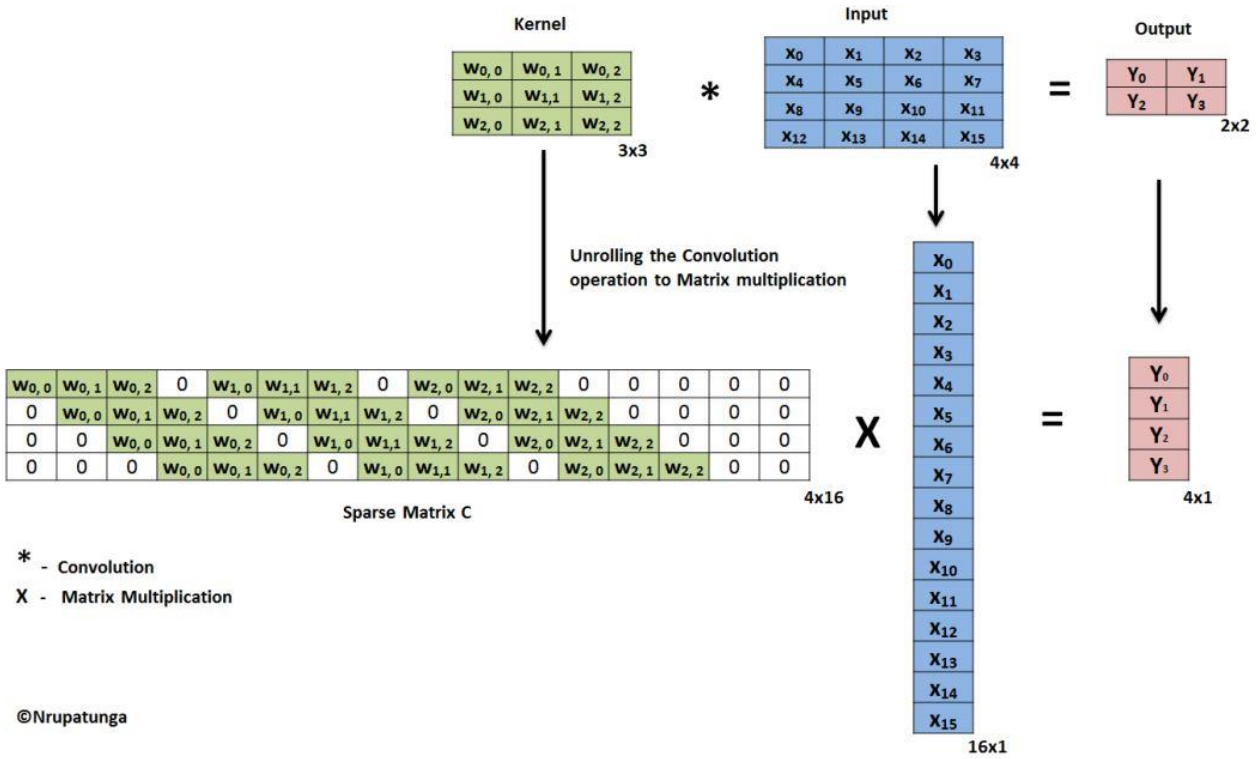
Convolution



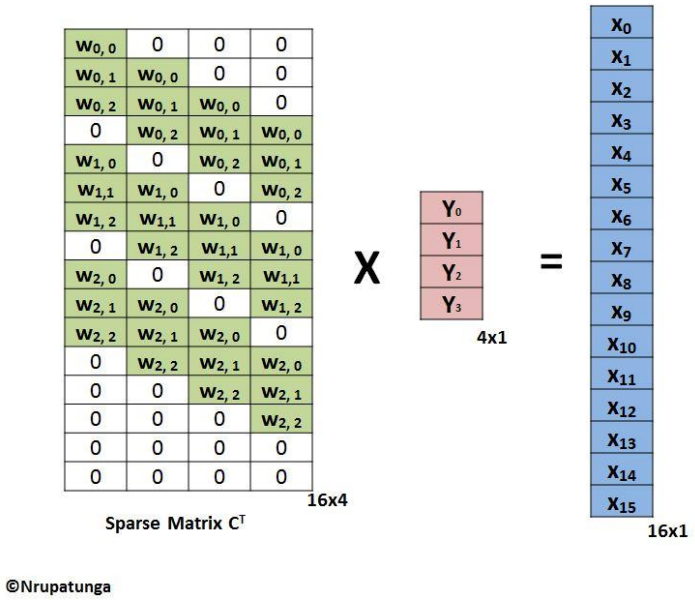
Transpose Convolution

https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md

# Transpose Convolution




Convolution




Transpose Convolution

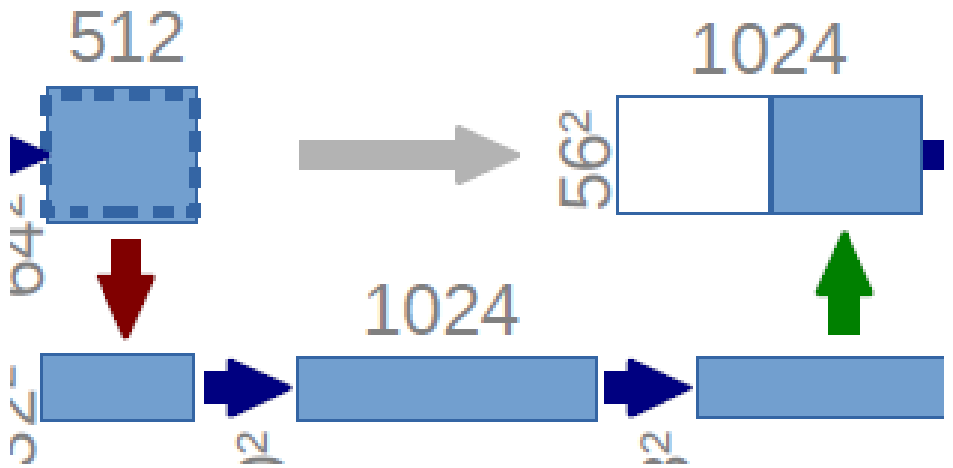
https://nrupatunga.github.io/convolution-2/

# Concatenation

**torch.cat**(*seq*, *dim=0*, *out=None*)

```
layer6 = F.relu(self.conv6_up(layer5))
layer6 = torch.cat((layer4, layer6), 1)
layer6 = F.relu(self.conv6_input(layer6))
layer6 = F.relu(self.conv6(layer6))

layer7 = F.relu(self.conv7_up(layer6))
layer7 = torch.cat((layer3, layer7), 1)
layer7 = F.relu(self.conv7_input(layer7))
layer7 = F.relu(self.conv7(layer7))

layer8 = F.relu(self.conv8_up(layer7))
layer8 = torch.cat((layer2, layer8), 1)
layer8 = F.relu(self.conv8_input(layer8))
layer8 = F.relu(self.conv8(layer8))

layer9 = F.relu(self.conv9_up(layer8))
layer9 = torch.cat((layer1, layer9), 1)
layer9 = F.relu(self.conv9_input(layer9))
layer9 = F.relu(self.conv9(layer9))
layer9 = self.final(self.conv9_output(layer9))

return layer9
```

- **seq** (*sequence of Tensors*) – any python sequence of tensors of the same type. Non-empty tensors provided must have the same shape, except in the cat dimension.
- **dim** (*int*, *optional*) – the dimension over which the tensors are concatenated
- **out** (*Tensor*, *optional*) – the output tensor




https://pytorch.org/docs/stable/nn.html

# All Code for U-Net

## Encoder

```
layer1 = F.relu(self.conv1_input(x))
layer1 = F.relu(self.conv1(layer1))

layer2 = F.max_pool2d(layer1, 2)
layer2 = F.relu(self.conv2_input(layer2))
layer2 = F.relu(self.conv2(layer2))

layer3 = F.max_pool2d(layer2, 2)
layer3 = F.relu(self.conv3_input(layer3))
layer3 = F.relu(self.conv3(layer3))

layer4 = F.max_pool2d(layer3, 2)
layer4 = F.relu(self.conv4_input(layer4))
layer4 = F.relu(self.conv4(layer4))

layer5 = F.max_pool2d(layer4, 2)
layer5 = F.relu(self.conv5_input(layer5))
layer5 = F.relu(self.conv5(layer5))
```

## Decoder

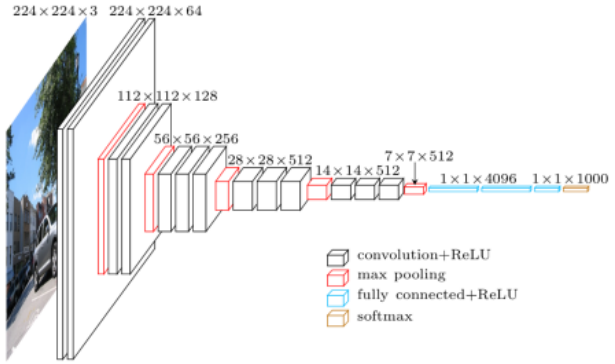
```
layer6 = F.relu(self.conv6_up(layer5))
layer6 = torch.cat((layer4, layer6), 1)
layer6 = F.relu(self.conv6_input(layer6))
layer6 = F.relu(self.conv6(layer6))

layer7 = F.relu(self.conv7_up(layer6))
layer7 = torch.cat((layer3, layer7), 1)
layer7 = F.relu(self.conv7_input(layer7))
layer7 = F.relu(self.conv7(layer7))

layer8 = F.relu(self.conv8_up(layer7))
layer8 = torch.cat((layer2, layer8), 1)
layer8 = F.relu(self.conv8_input(layer8))
layer8 = F.relu(self.conv8(layer8))

layer9 = F.relu(self.conv9_up(layer8))
layer9 = torch.cat((layer1, layer9), 1)
layer9 = F.relu(self.conv9_input(layer9))
layer9 = F.relu(self.conv9(layer9))
layer9 = self.final(self.conv9_output(layer9))

return layer9
```

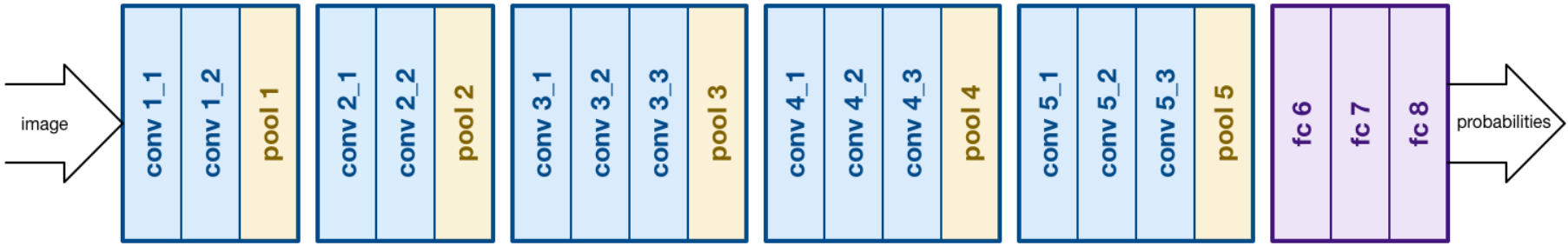
## Parameters

```
self.conv1_input =      nn.Conv2d(1, 64, 3, padding=1)
self.conv1 =            nn.Conv2d(64, 64, 3, padding=1)
self.conv2_input =      nn.Conv2d(64, 128, 3, padding=1)
self.conv2 =            nn.Conv2d(128, 128, 3, padding=1)
self.conv3_input =      nn.Conv2d(128, 256, 3, padding=1)
self.conv3 =            nn.Conv2d(256, 256, 3, padding=1)
self.conv4_input =      nn.Conv2d(256, 512, 3, padding=1)
self.conv4 =            nn.Conv2d(512, 512, 3, padding=1)
self.conv5_input =      nn.Conv2d(512, 1024, 3, padding=1)
self.conv5 =            nn.Conv2d(1024, 1024, 3, padding=1)

self.conv6_up =         nn.ConvTranspose2d(1024, 512, 2, 2)
self.conv6_input =      nn.Conv2d(1024, 512, 3, padding=1)
self.conv6 =            nn.Conv2d(512, 512, 3, padding=1)
self.conv7_up =         nn.ConvTranspose2d(512, 256, 2, 2)
self.conv7_input =      nn.Conv2d(512, 256, 3, padding=1)
self.conv7 =            nn.Conv2d(256, 256, 3, padding=1)
self.conv8_up =         nn.ConvTranspose2d(256, 128, 2, 2)
self.conv8_input =      nn.Conv2d(256, 128, 3, padding=1)
self.conv8 =            nn.Conv2d(128, 128, 3, padding=1)
self.conv9_up =         nn.ConvTranspose2d(128, 64, 2, 2)
self.conv9_input =      nn.Conv2d(128, 64, 3, padding=1)
self.conv9 =            nn.Conv2d(64, 64, 3, padding=1)
self.conv9_output =     nn.Conv2d(64, 2, 1)
```

https://github.com/jonnedtc/U-Net-PyTorch/blob/master/networks.py

# Vgg 16 Classification



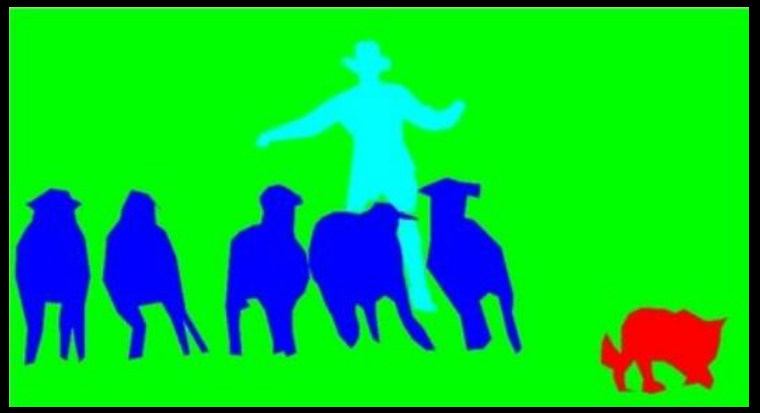

```python
def make_layers(cfg, batch_norm=False):
    layers = []
    in_channels = 3
    for v in cfg:
        if v == 'M':
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        else:

            conv2d = nn.Conv2d(in_channels, v, 3, padding=1)
            if batch_norm:
                layers += [conv2d, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]
            else:
                layers += [conv2d, nn.ReLU(inplace=True)]
            in_channels = v
    return nn.Sequential(*layers)
```

```python
self.classifier = nn.Sequential(
    #fc6
    nn.Linear(512*7*7, 4096),
    nn.ReLU(),
    nn.Dropout(),

    #fc7
    nn.Linear(4096, 4096),
    nn.ReLU(),
    nn.Dropout(),

    #fc8
    nn.Linear(4096, num_classes))
```

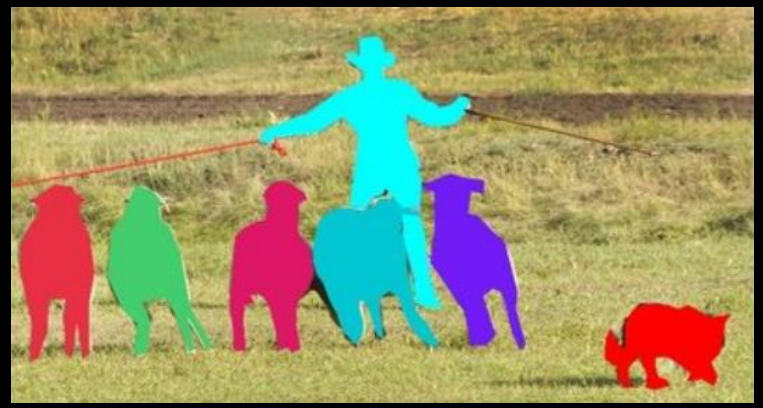
```python
def forward(self, x):
    x = self.features(x)
    x = x.view(x.size(0), -1)
    x = self.classifier(x)
    return x
```

# Instance Segmentation

Semantic Segmentation

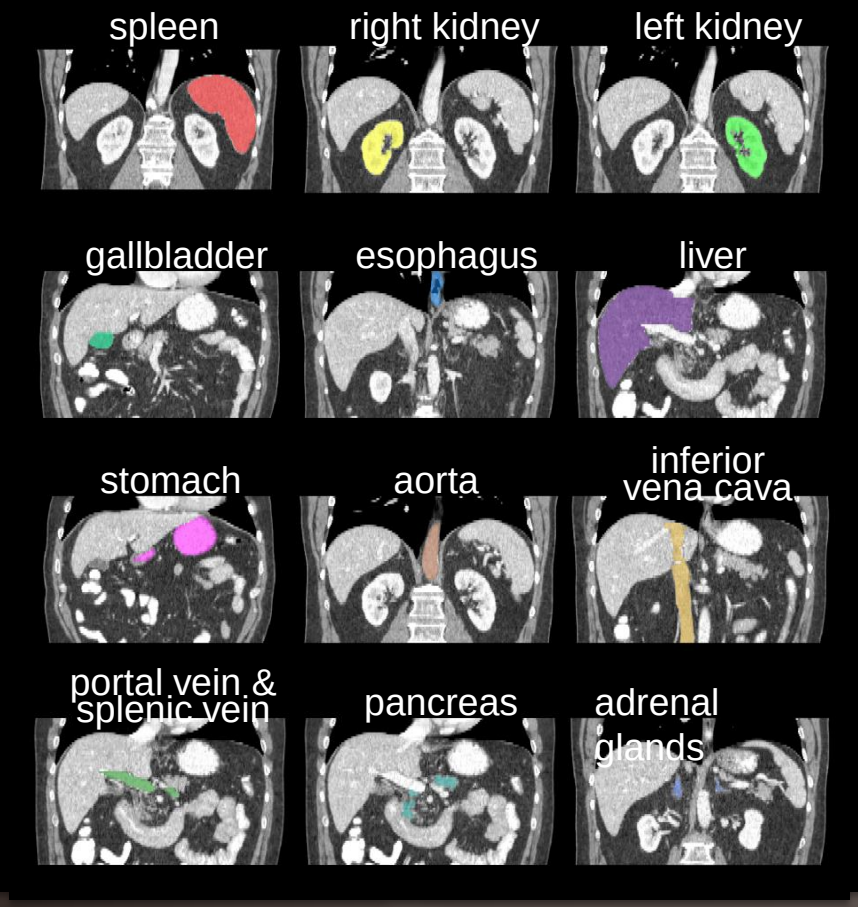
Instance Segmentation

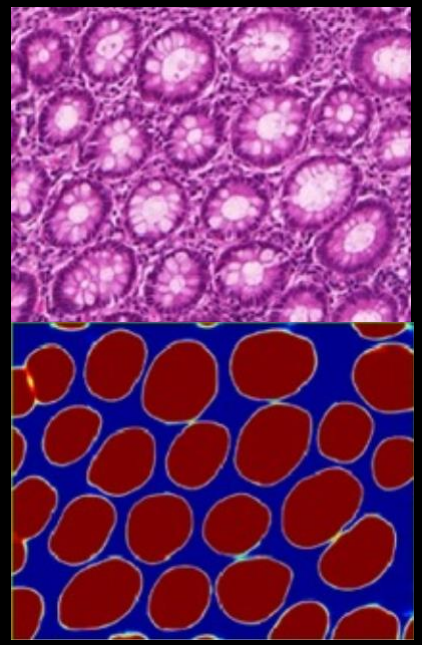



https://arxiv.org/abs/1405.0312

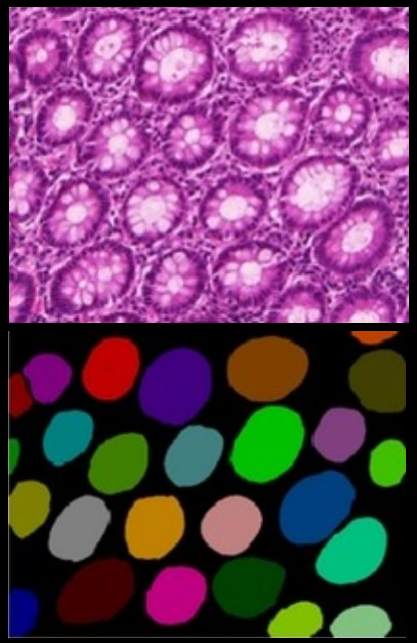
# Instance Segmentation

## Semantic Segmentation



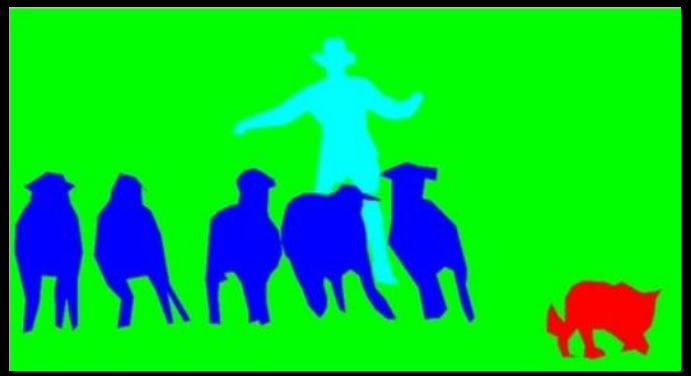
## Instance Segmentation



https://www.sciencedirect.com/science/article/pii/S1361841516302043

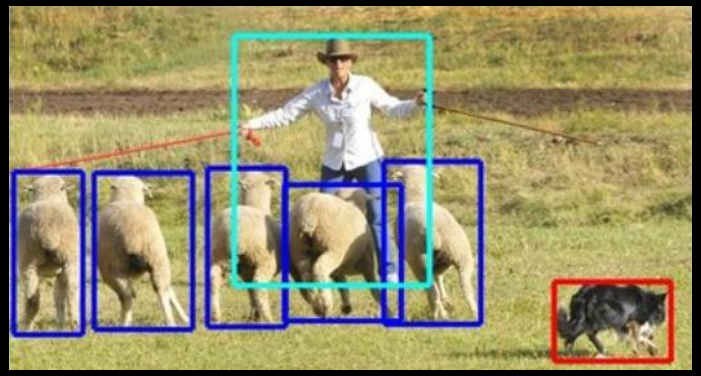
# How To Perform Instance Segmentation

Semantic Segmentation



+

Detection



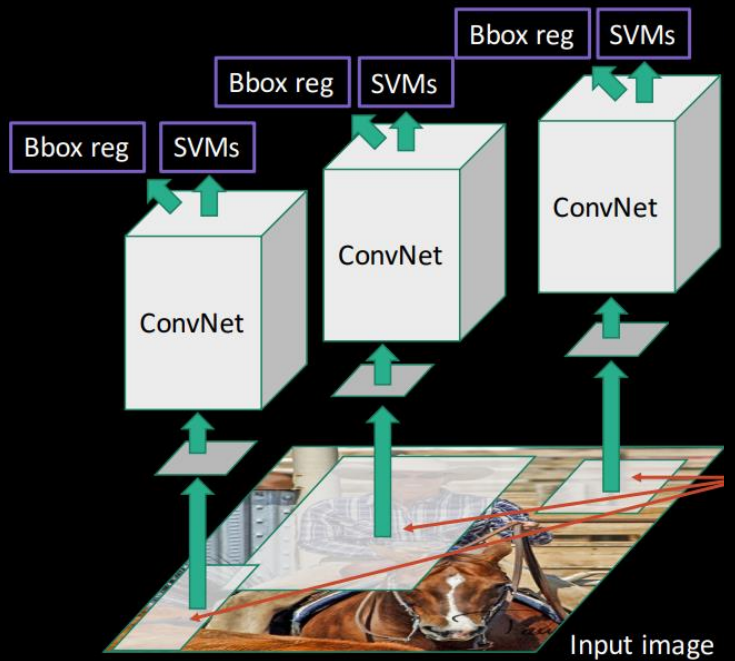
=

Instance Segmentation

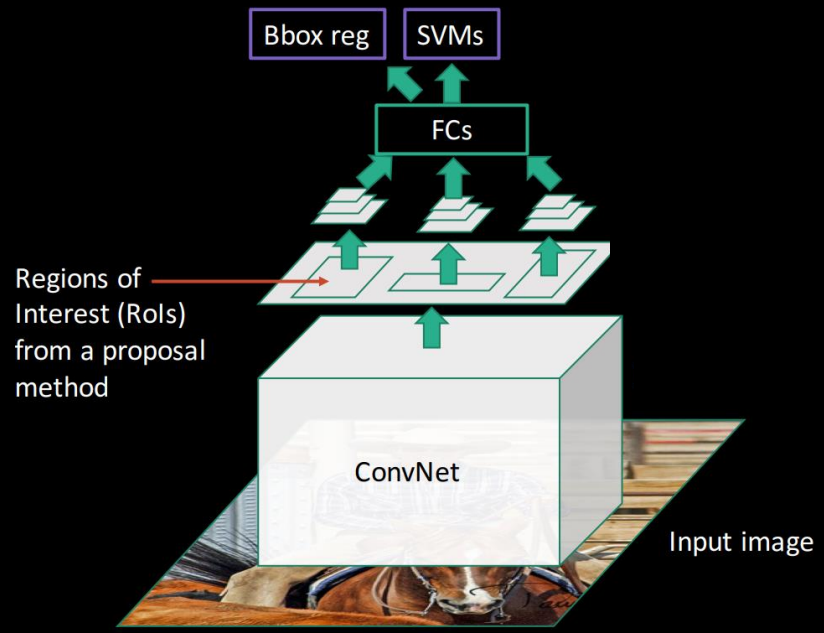
https://arxiv.org/abs/1405.0312

# Detection using RCNN Family

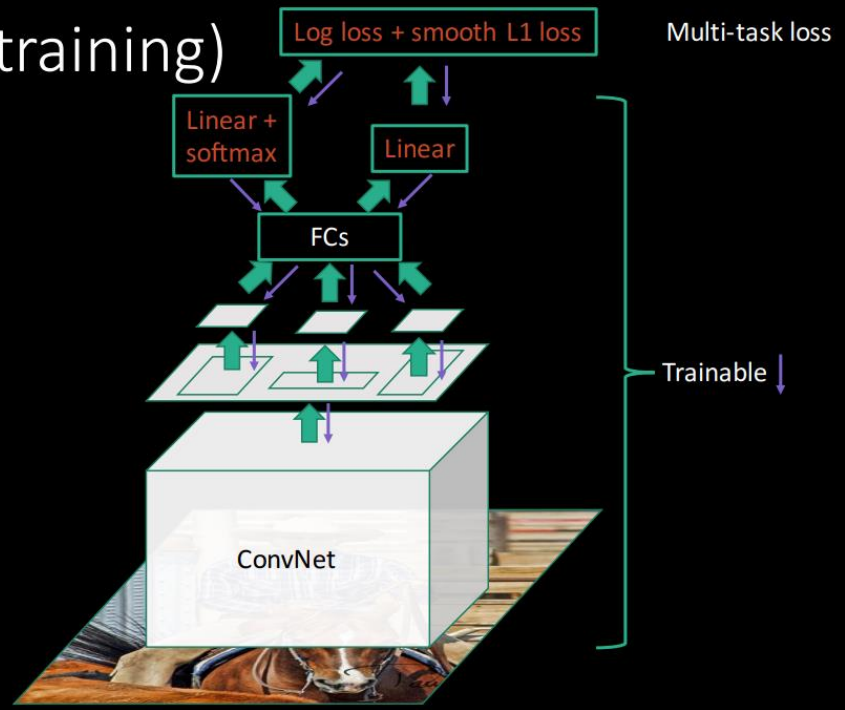
Slow R-CNN




SPP-net

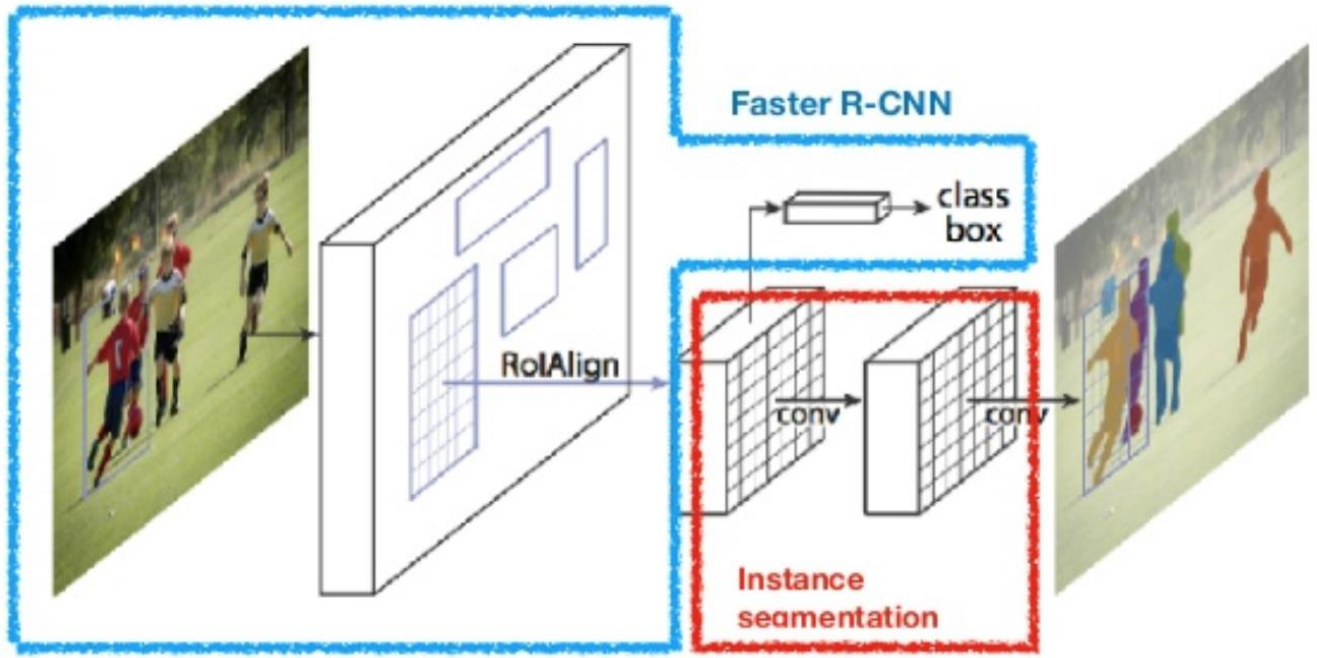


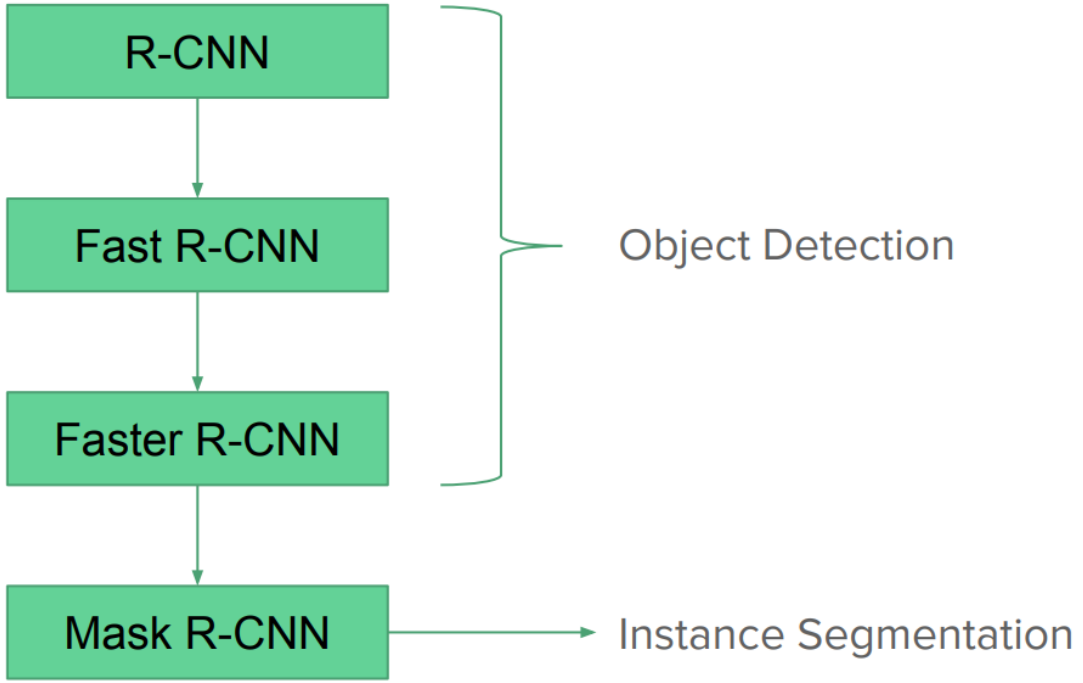

Fast R-CNN (training)




http://www.robots.ox.ac.uk/~tvg/publications/talks/fast-rcnn-slides.pdf

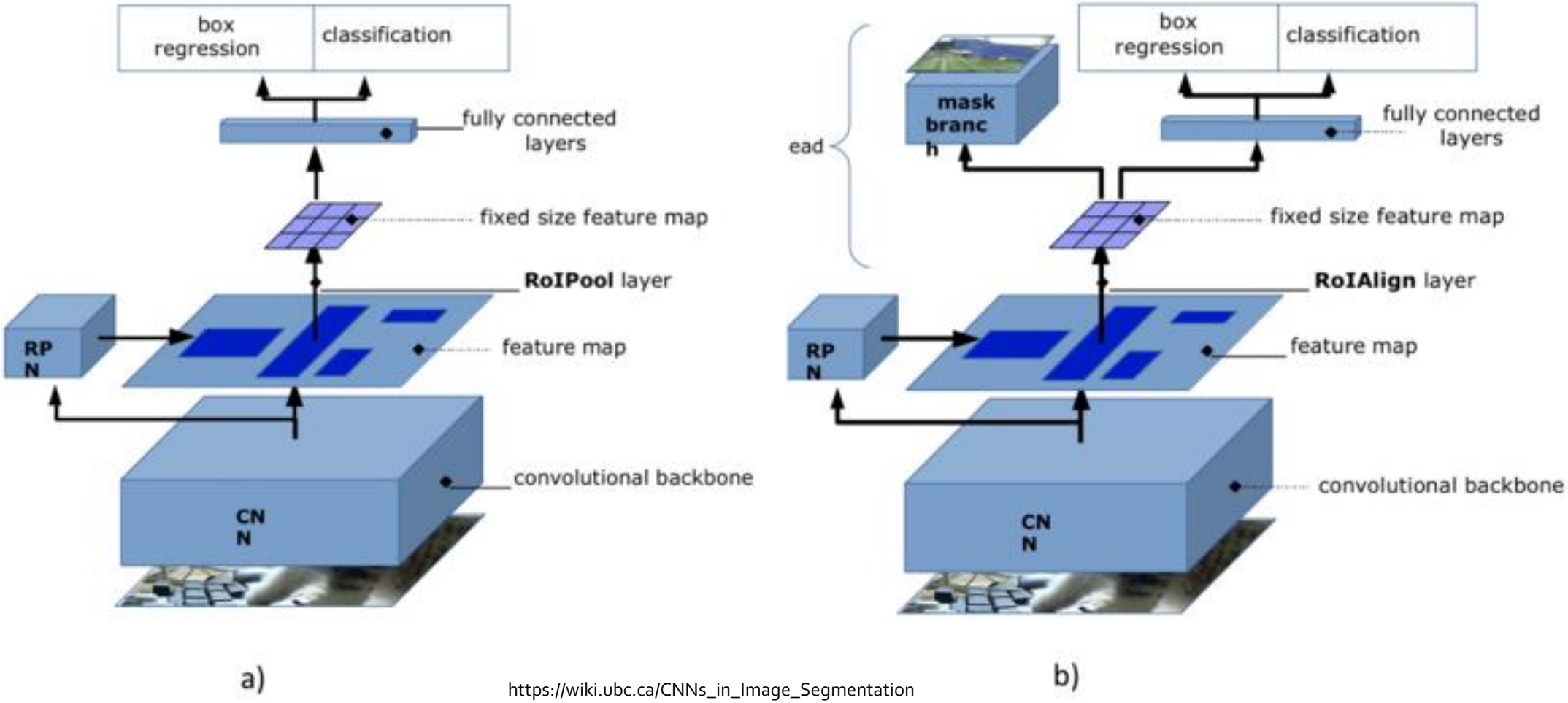
# Mask R-CNN



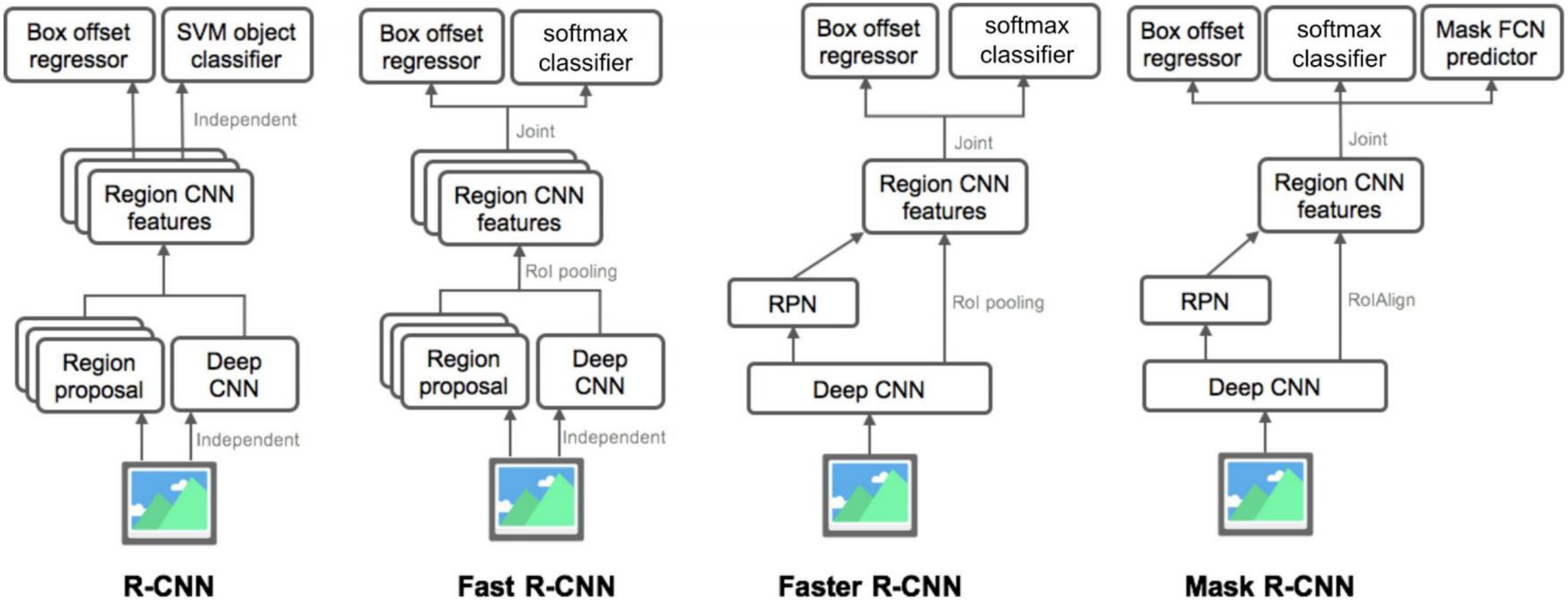

https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html

# Faster R-CNN vs. Mask R-CNN



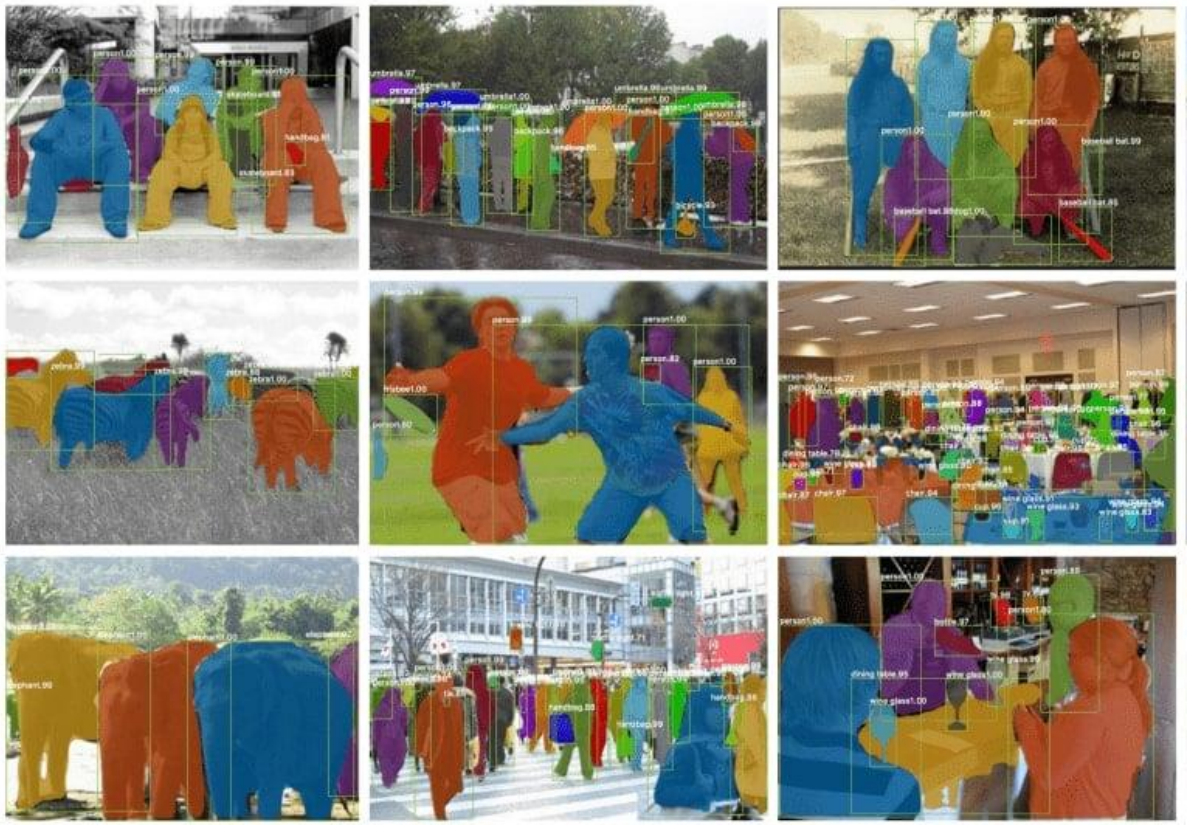

https://wiki.ubc.ca/CNNs_in_Image_Segmentation

# R-CNN Family

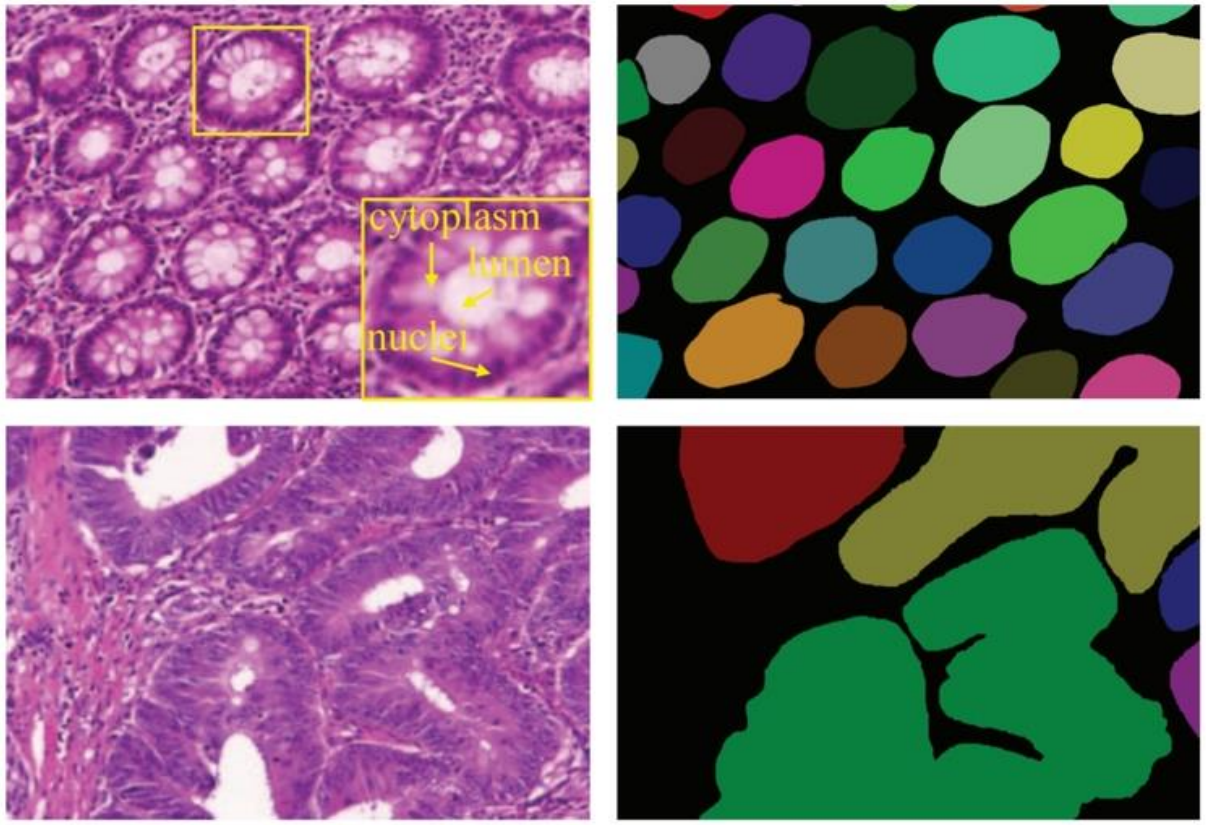



https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html

# Instance Segmentation



https://sigmoidal.io/dl-computer-vision-beyond-classification/




https://www.ocutri.com/medical/?language=en