Convolutional Neural Networks for Computer Vision

Sachin Mehta

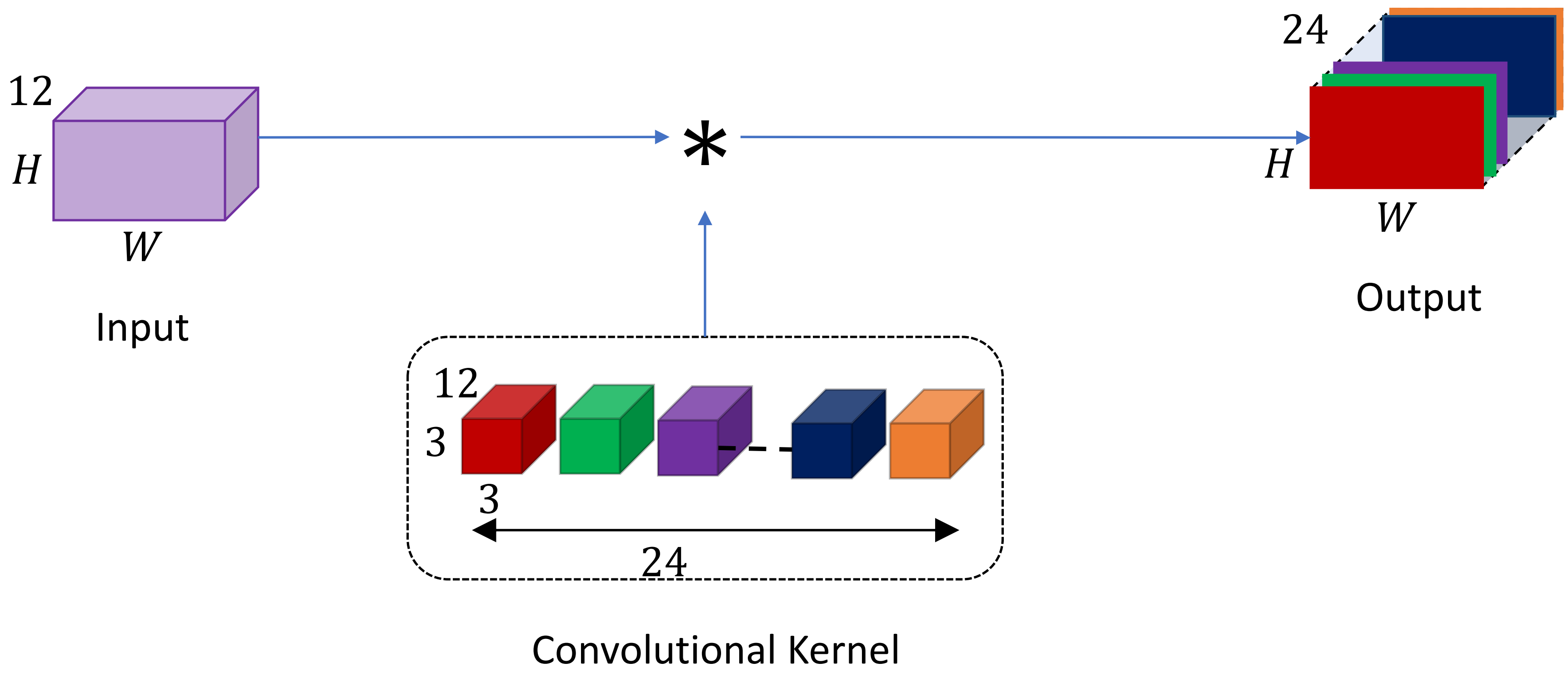
Outline

- Brief overview of convolutions
  - Parameters
  - # of operations
  - Receptive Field
  - Efficient convolutions
- Different Convolutional Units
- Semantic Segmentation
- Some visualizations

Convolutions

Convolution

```python
import torch
from torch import nn

nn.Conv2d(in_channels=12, out_channels=24, kernel_size=3, stride=1, padding=0)
```



How many parameters did we learn?

- Number of parameters:

$$in_{channels} * out_{channels} * kernelHeight * kernelWidth$$

- For our example, we learned 12*24*3*3 = 2,616

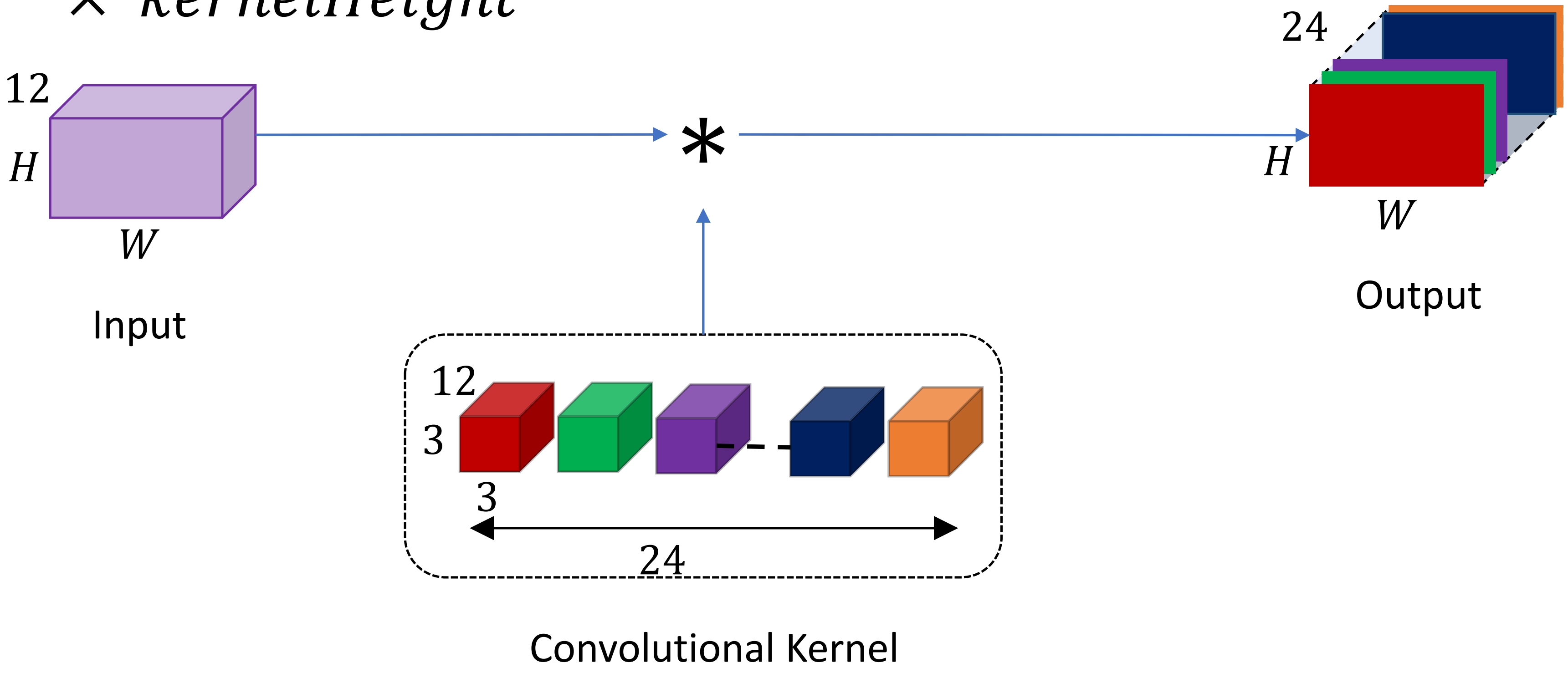
```
import torch
from torch import nn
import numpy as np

layer = nn.Conv2d(in_channels=12, out_channels=24, kernel_size=3, stride=1, padding=0)
print('Weight tensor size: {}'.format(layer.weight.size()))

num_parameters = np.sum([p.numel() for p in layer.parameters()])
print("Number of parameters: {}".format(num_parameters))

# p.numel() multiplies the dimensions of a tensor
```

```
Weight tensor size: torch.Size([24, 12, 3, 3])
Number of parameters: 2616
```

How many operations did we perform?

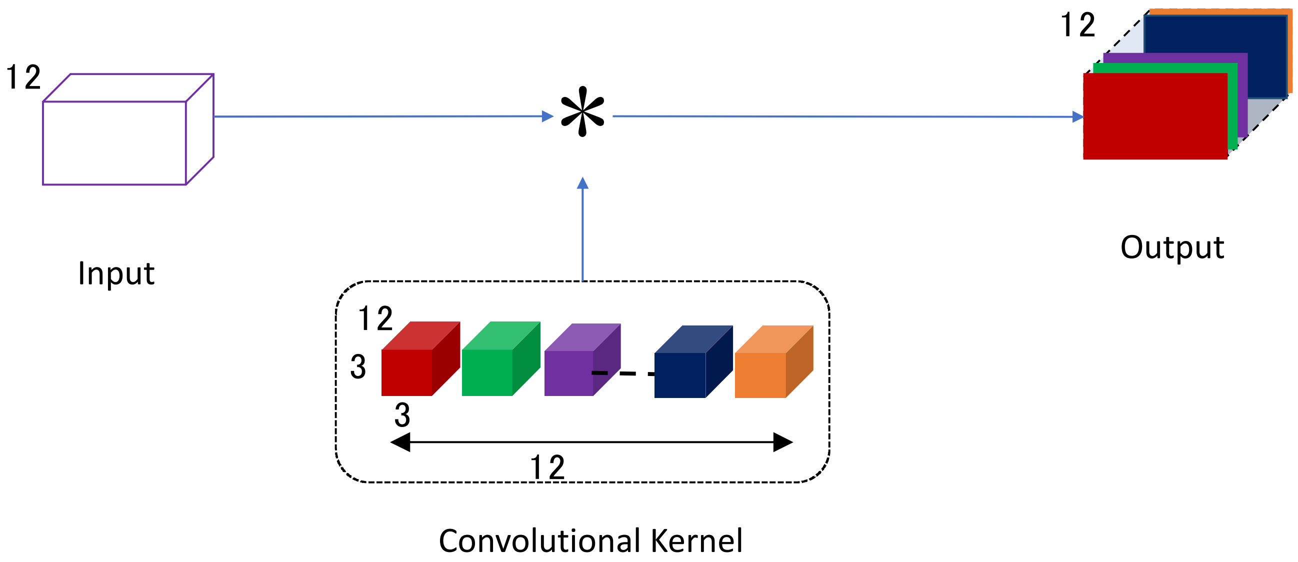
- Number of operations:

$$H \times W \times inChannels \times outChannels \times kernelWidth \times kernelHeight$$



How can we reduce convolutional operations?

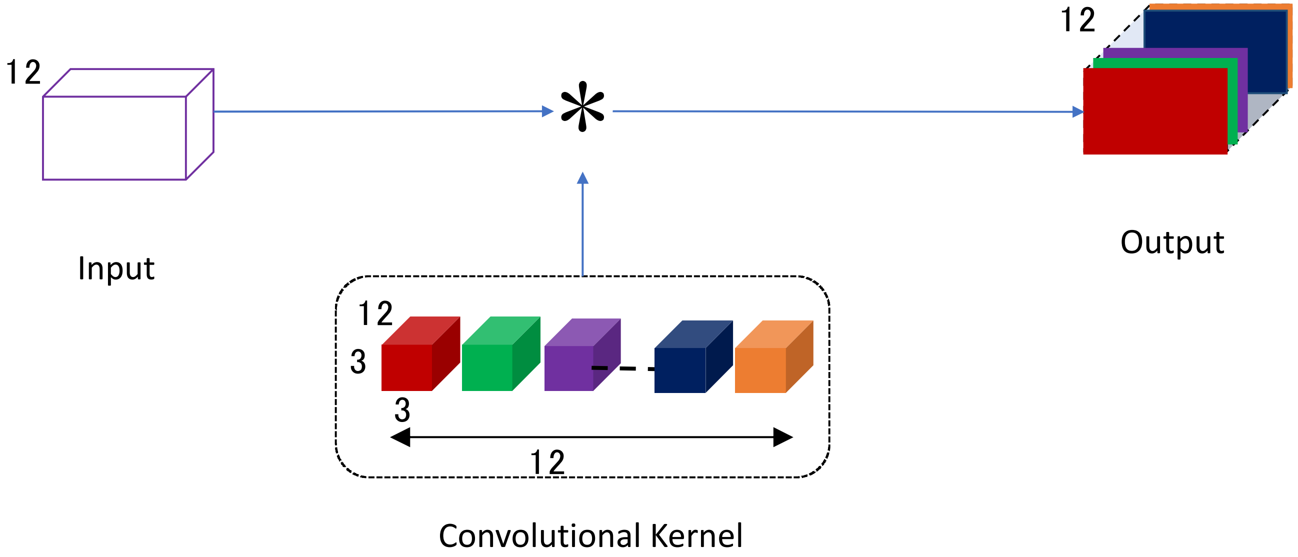
- Apply convolution per channel



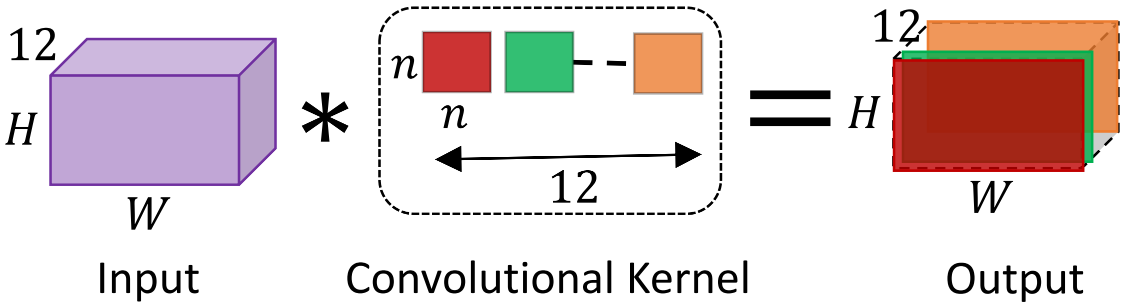
Standard Convolution

How can we reduce convolutional operations?

- Apply convolution per channel



Standard Convolution



Depth-wise Convolution

Standard vs Depth-wise convolution in PyTorch

```python
import torch
from torch import nn
import numpy as np
######================================
### Standard Convolution
######================================
print('Standard Convolution')
standard_conv = nn.Conv2d(in_channels=12, out_channels=12, kernel_size=3, stride=1, padding=0)
# Weight tensor size
print('Weight tensor size: {}'.format(standard_conv.weight.size()))
# number of parameters
num_parameters = np.sum([p.numel() for p in standard_conv.parameters()])
print("Number of parameters: {}".format(num_parameters))
```

```
Standard Convolution
Weight tensor size: torch.Size([12, 12, 3, 3])
Number of parameters: 1308
```

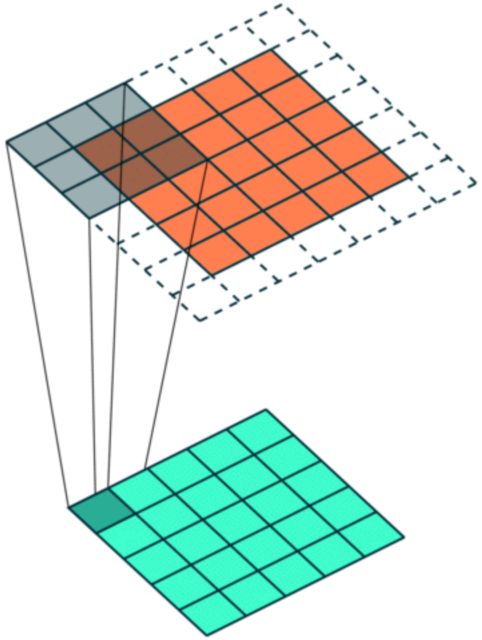
Note the change in the size of weight tensor

```python
######================================
### Depth-wise Convolution
######================================
print('\n')
print('DEPTH-WISE Convolution')
depth_wise_conv = nn.Conv2d(in_channels=12, out_channels=12, kernel_size=3, stride=1, padding=0, groups=12)
# Weight tensor size
print('Weight tensor size: {}'.format(depth_wise_conv.weight.size()))
# number of parameters
num_parameters = np.sum([p.numel() for p in depth_wise_conv.parameters()])
print("Number of parameters: {}".format(num_parameters))
```

```
DEPTH-WISE Convolution
Weight tensor size: torch.Size([12, 1, 3, 3])
Number of parameters: 120
```

Receptive Field

- Region in the input that particular convolutional kernel is looking at to produce an output



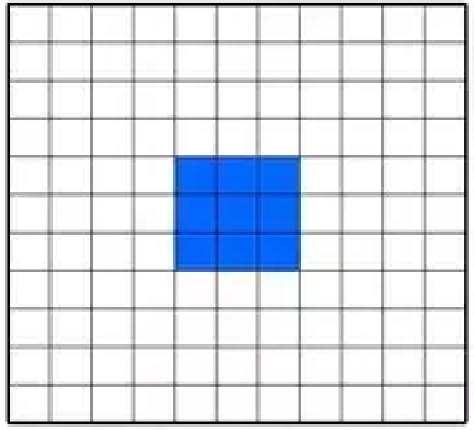
Receptive Field

- Receptive field of the convolutional layer is: ??????

```
import torch
from torch import nn
import numpy as np

layer = nn.Conv2d(in_channels=12, out_channels=24, kernel_size=5, stride=1, padding=0)
print('Weight tensor size: {}'.format(layer.weight.size()))

num_parameters = np.sum([p.numel() for p in layer.parameters()])
print("Number of parameters: {}".format(num_parameters))
```

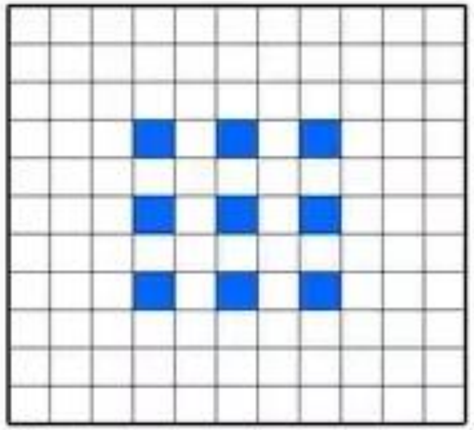
How can we increase the receptive field without increasing parameters?

- Dilated convolution:
  - Insert zeros between kernel elements to increase the receptive field



Standard convolution

- Kernel size: 3x3
- Receptive field: 3x3
- Weights: 9



Dilated convolution

- Kernel size: 3x3
- Receptive field: 5x5
- Weights: 9

Standard vs Dilated convolutions in PyTorch

```
import torch
from torch import nn
import numpy as np
######================================
### Standard Convolution
######================================
print('Standard Convolution')
standard_conv = nn.Conv2d(in_channels=12, out_channels=12, kernel_size=3, stride=1, padding=0)
# Weight tensor size
print('Weight tensor size: {}'.format(standard_conv.weight.size()))
# number of parameters
num_parameters = np.sum([p.numel() for p in standard_conv.parameters()])
print("Number of parameters: {}".format(num_parameters))
```

```
Standard Convolution
Weight tensor size: torch.Size([12, 12, 3, 3])
Number of parameters: 1308
```

- Note there is no change in the tensor size.
- Zeros are added (input elements are skipped) during actual computation.
- This is done for efficiency.

```
######================================
### Dilated Convolution
######================================
print('\n')
print('Dilated Convolution')
dilated_conv = nn.Conv2d(in_channels=12, out_channels=12, kernel_size=3, stride=1, padding=0, dilation=2)
# Weight tensor size
print('Weight tensor size: {}'.format(dilated_conv.weight.size()))
# number of parameters
num_parameters = np.sum([p.numel() for p in dilated_conv.parameters()])
print("Number of parameters: {}".format(num_parameters))
```

```
Dilated Convolution
Weight tensor size: torch.Size([12, 12, 3, 3])
Number of parameters: 1308
```

Convolution Blocks

CNN Structure

- A typical image classification network is a stack of convolutional and pooling layers
- Instead of using a single convolution layer, you can use a convolutional block

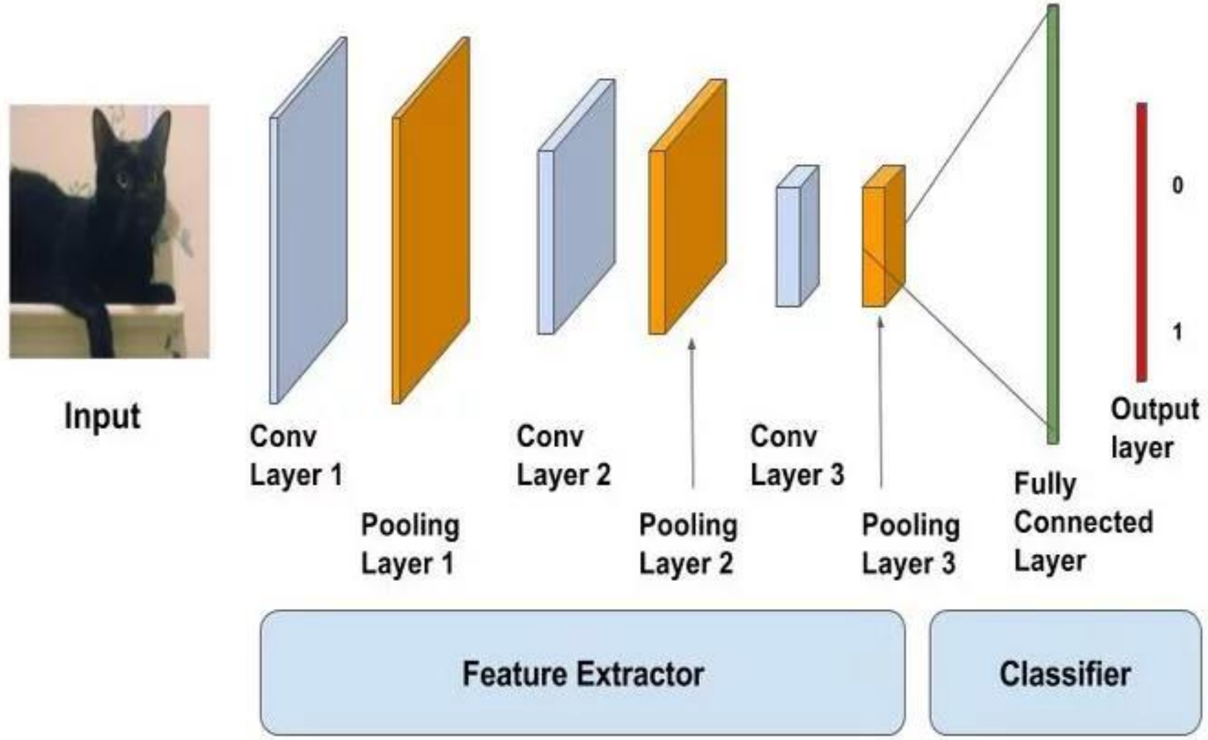
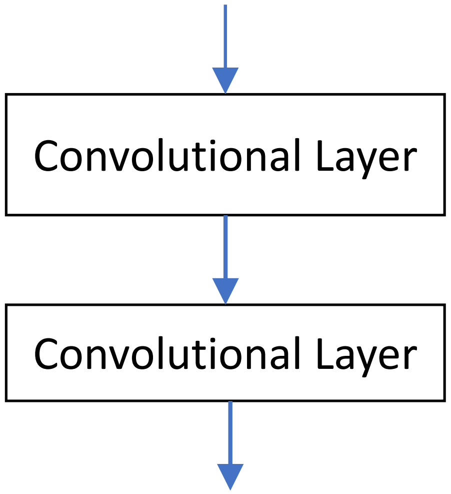


Image source: https://www.learnopencv.com/image-classification-using-convolutional-neural-networks-in-keras/

Convolutional block: VGG

- VGG block usually comprises of a stack of 2 or 3 convolutional layers

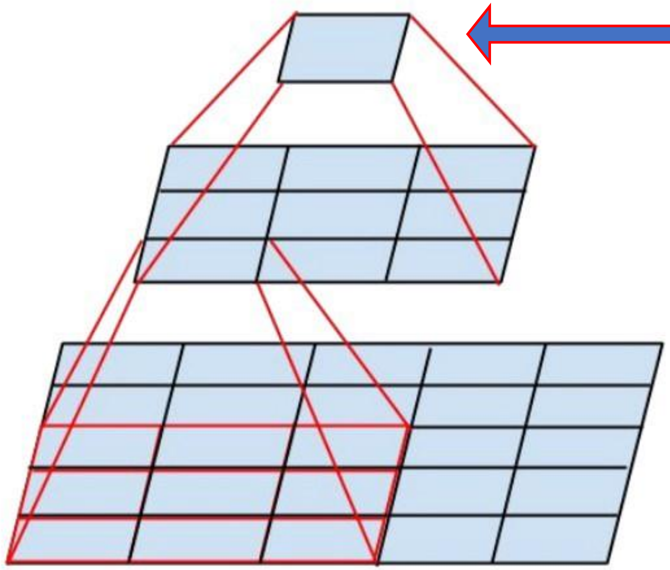


VGG Block

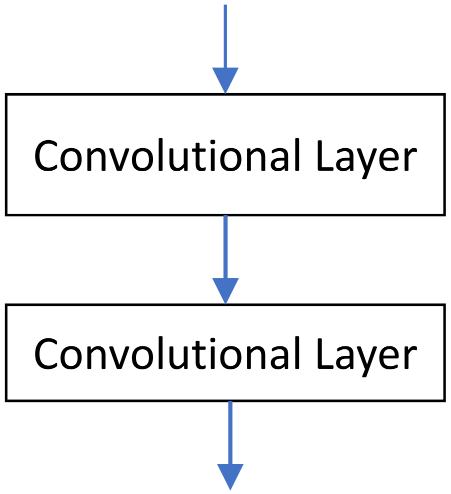
Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition."ICLR (2015).

Convolutional block: VGG

- VGG block usually comprises of a stack of 2 or 3 convolutional layers
- Stacking increases **effective** receptive field while learning fewer parameters



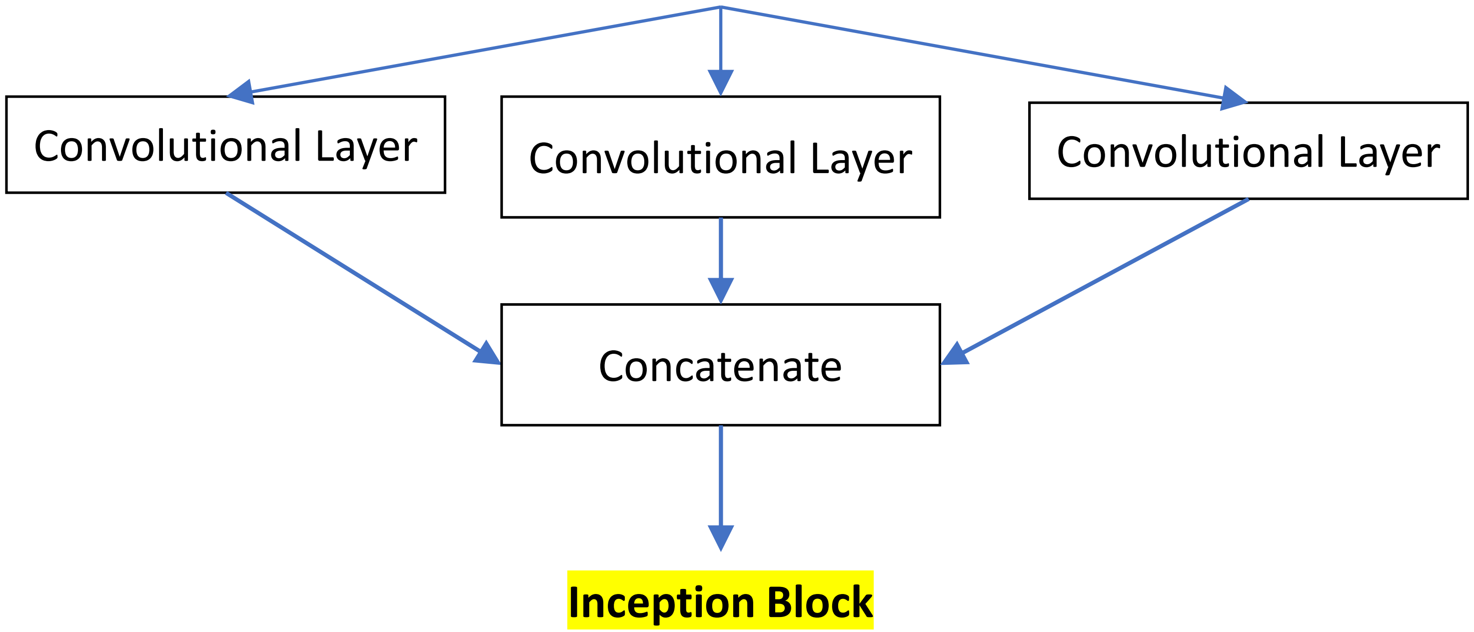
This pixel has information from 5x5 input region



VGG Block

Convolutional Block: Inception

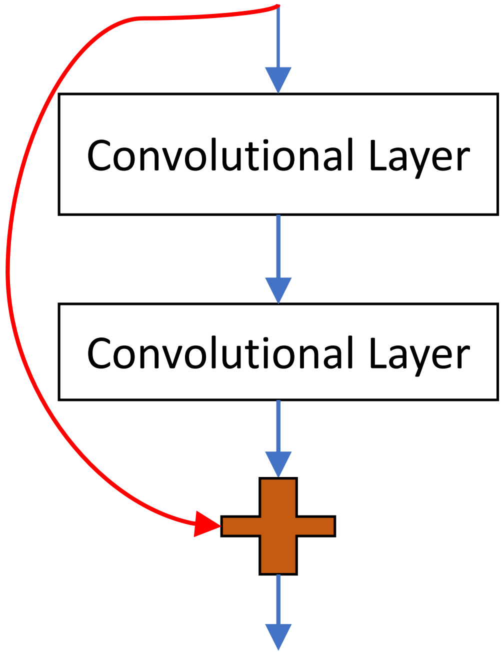
- Instead of stacking, this block process multiple convolutional layers in parallel.



- Szegedy, et al. "Going deeper with convolutions." CVPR 2015. (ImageNet challenge Winner, 2014)
- Szegedy et al. "Rethinking the inception architecture for computer vision", CVPR 2016.

Convolutional Block: ResNet

- Same as the VGG block, but adds input and output
- Improves gradient flow
- Widely used CNN block

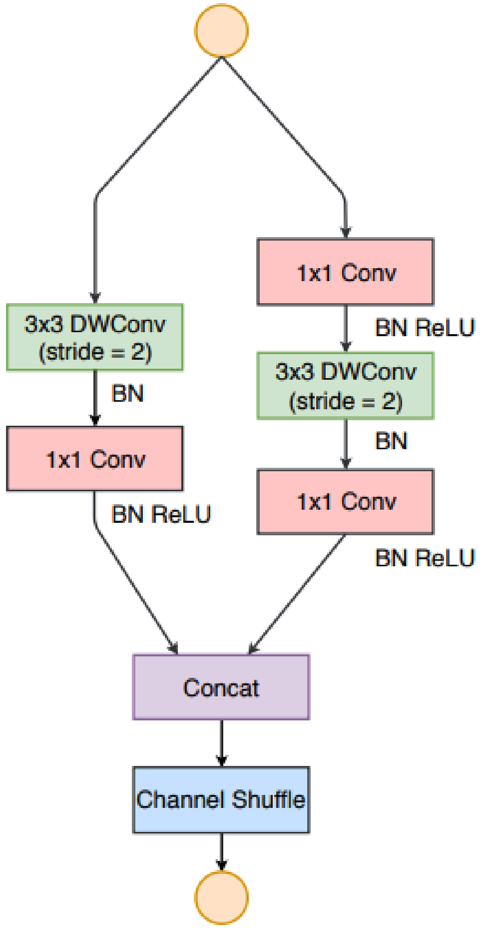


ResNet Block

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." CVPR, 2016 **(Best Paper)**

Convolutional Block: ShuffleNet

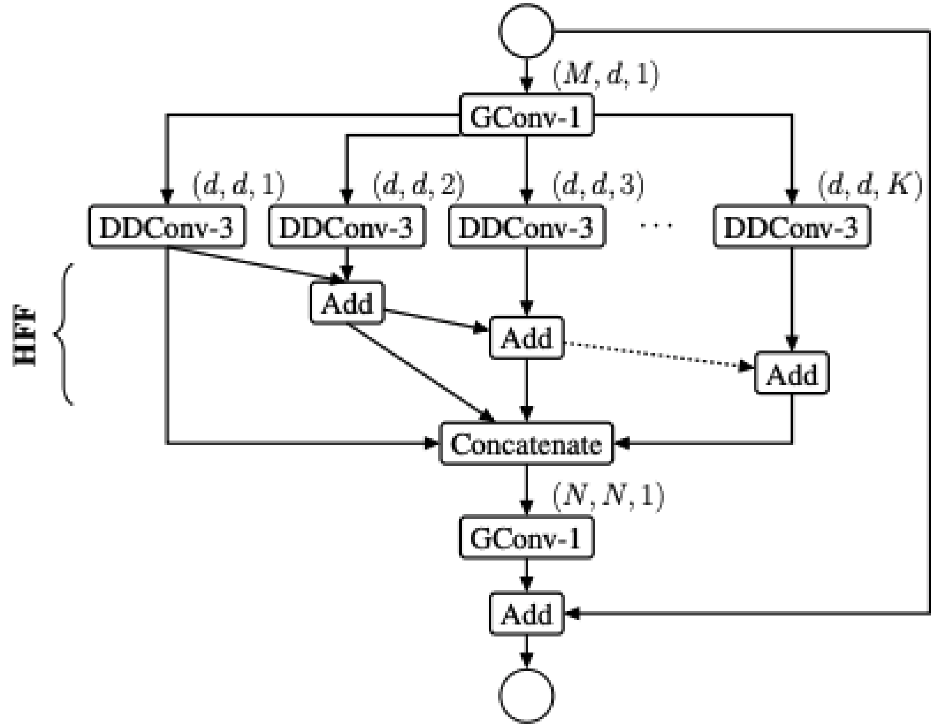
- This block is the same as the ResNet block, but it uses
  - Depth-wise Convolutions
  - Channel-split and channel-shuffle to be efficient



- Zhang et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices.", CVPR, 2018
- Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. "Shufflenet v2: Practical guidelines for efficient cnn architecture design." ECCV, 2018.

Convolutional Block: ESPNet

- This block works on following principle:
  - Reduce
  - Split
  - Transform
  - Merge
- This block uses ***depth-wise dilated convolutions*** (DDConv)
- Looks similar to the Inception block, but it has higher receptive field

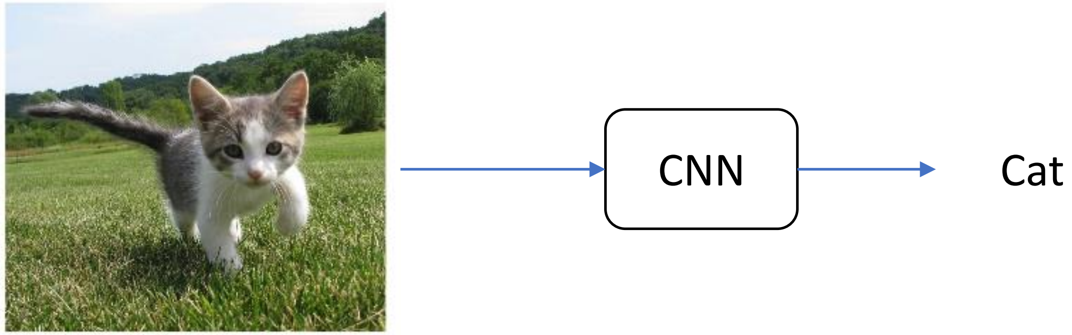


- **Sachin Mehta**, Mohammad Rastegari, Anat Caspi, Linda Shapiro, and Hannaneh Hajishirzi. "Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation." ECCV, 2018
- **Sachin Mehta**, Mohammad Rastegari, Linda Shapiro, and Hannaneh Hajishirzi. "ESPNetv2: A Light-weight, Power Efficient, and General Purpose Convolutional Neural Network." *CVPR, 2019*

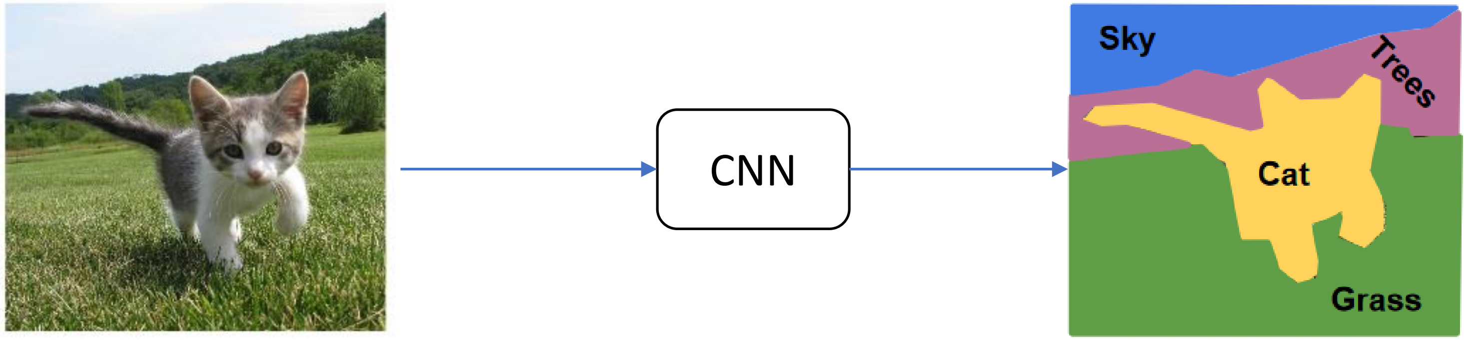
Semantic Segmentation

Semantic Segmentation

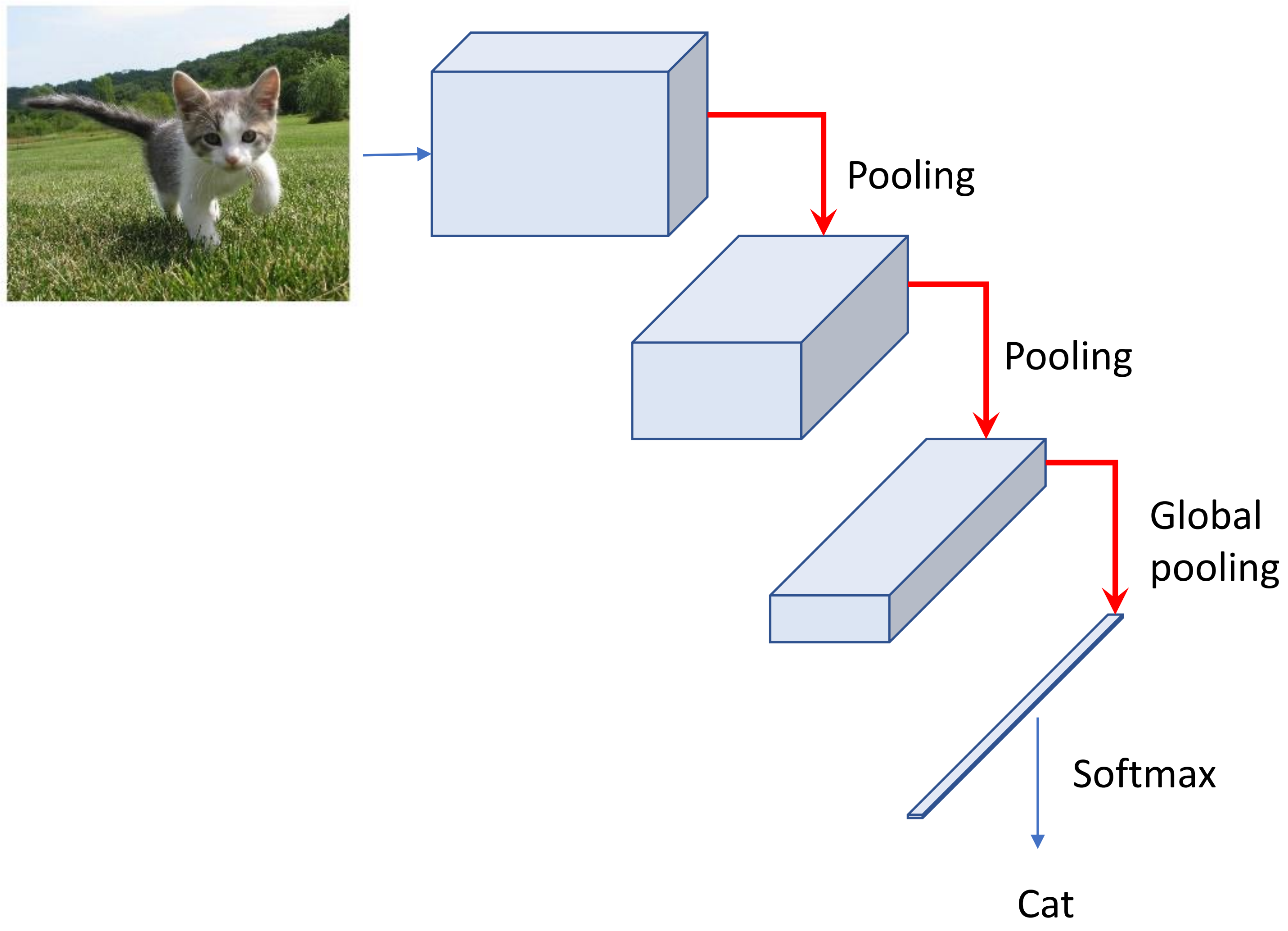
- Image Classification



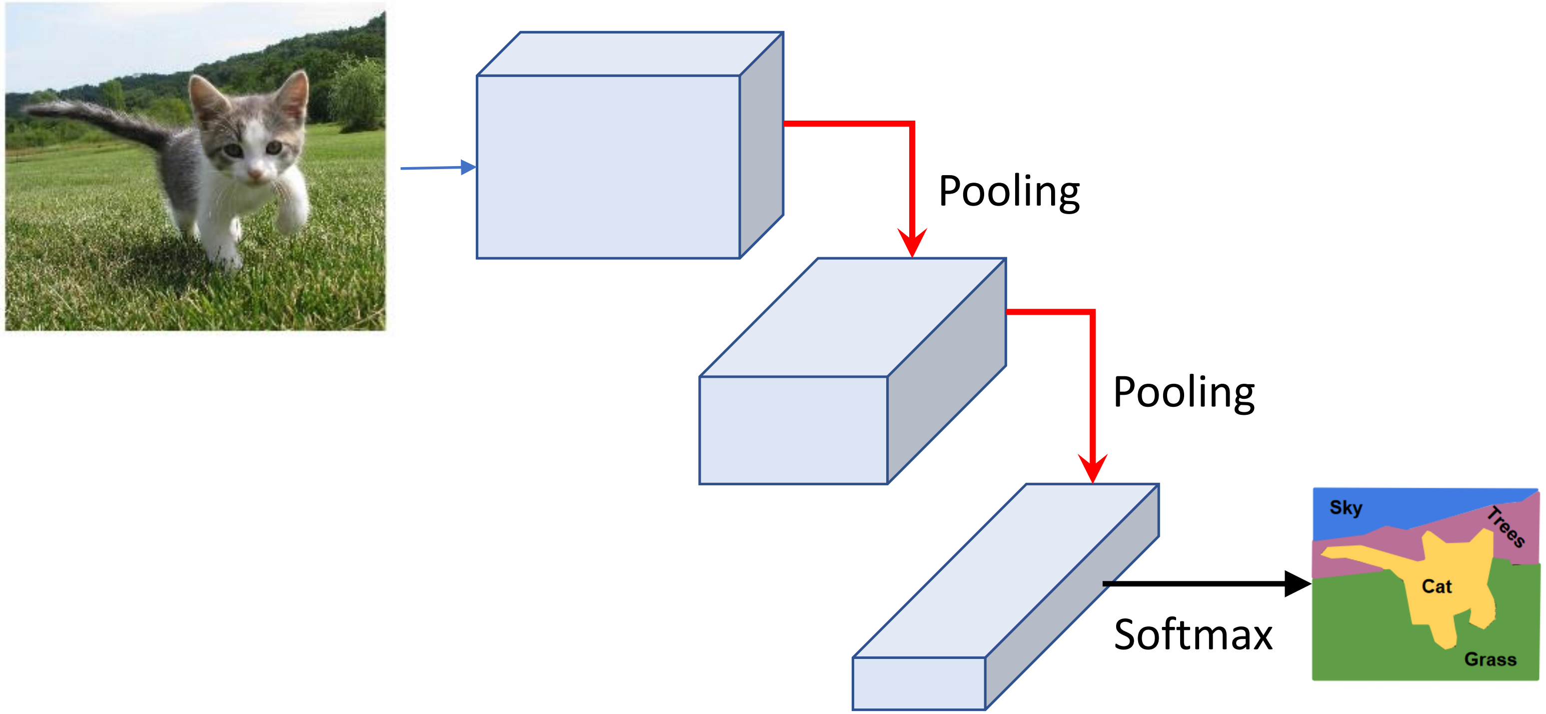
- Semantic segmentation



Segmentation Networks: Vanilla



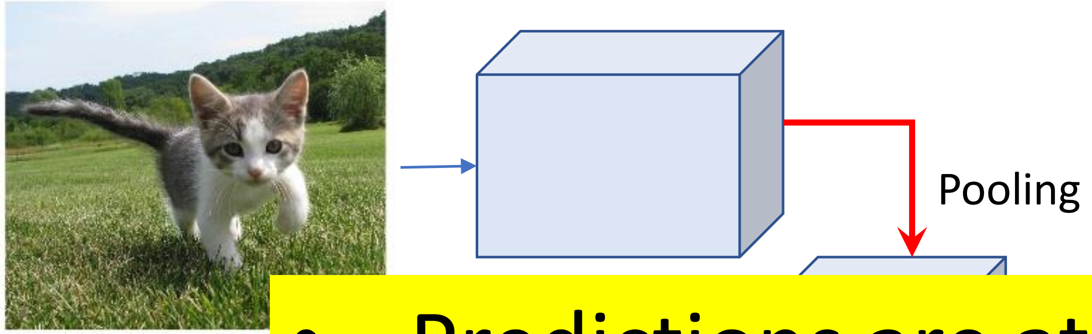
Segmentation Networks: Vanilla



Standard Approach:

- Remove the classification layer
- Finetune for segmentation

Segmentation Networks: Vanilla

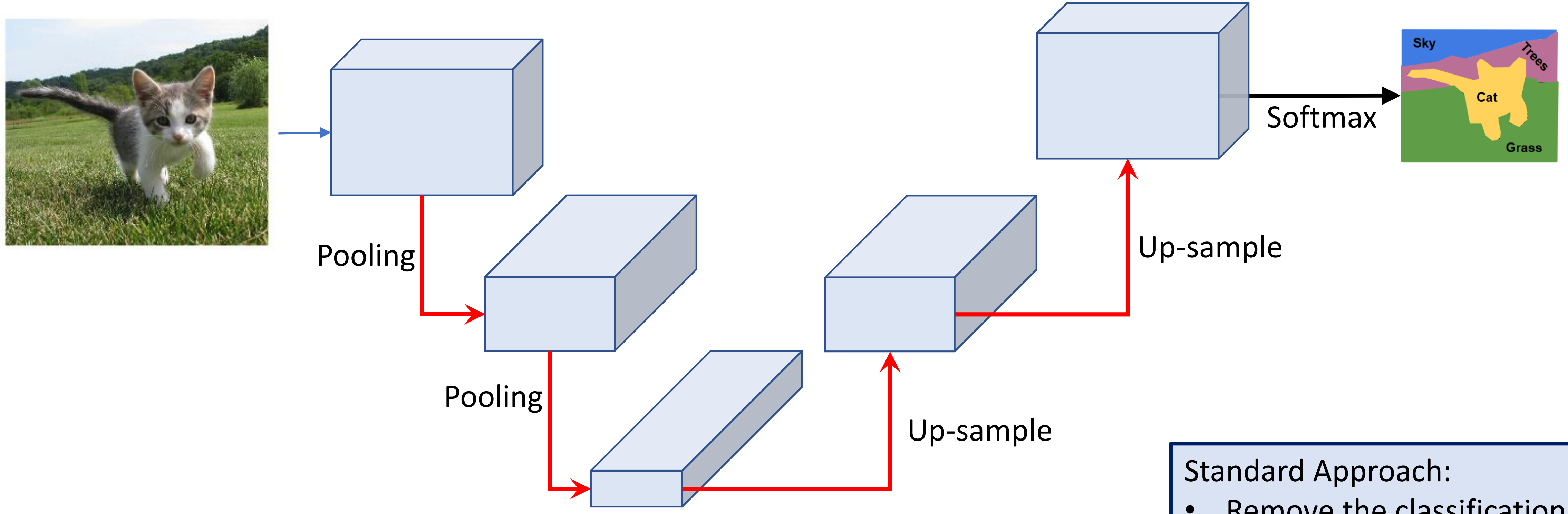


- Predictions are at low resolution
- A lot of information is lost

Can we make it better?

Standard Approach:
- Remove the classification layer
- Finetune for segmentation

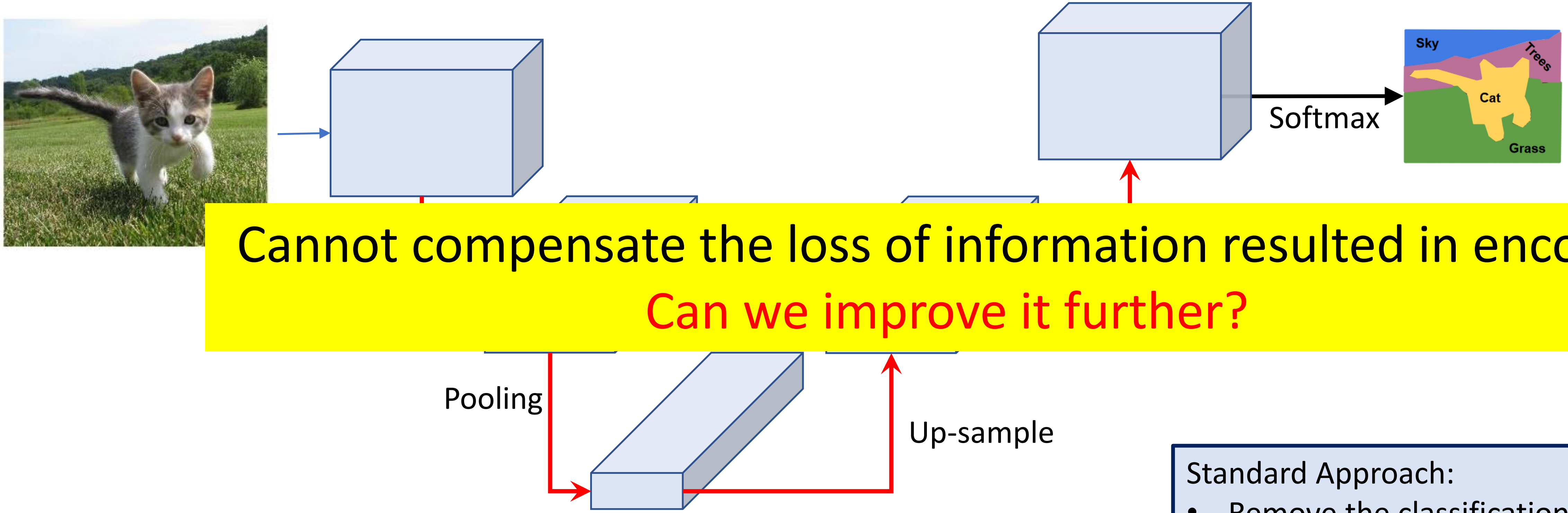
Segmentation Networks: Encoder-Decoder



Standard Approach:

- Remove the classification layer
- Finetune for segmentation
- Add a decoder

Segmentation Networks: Encoder-Decoder



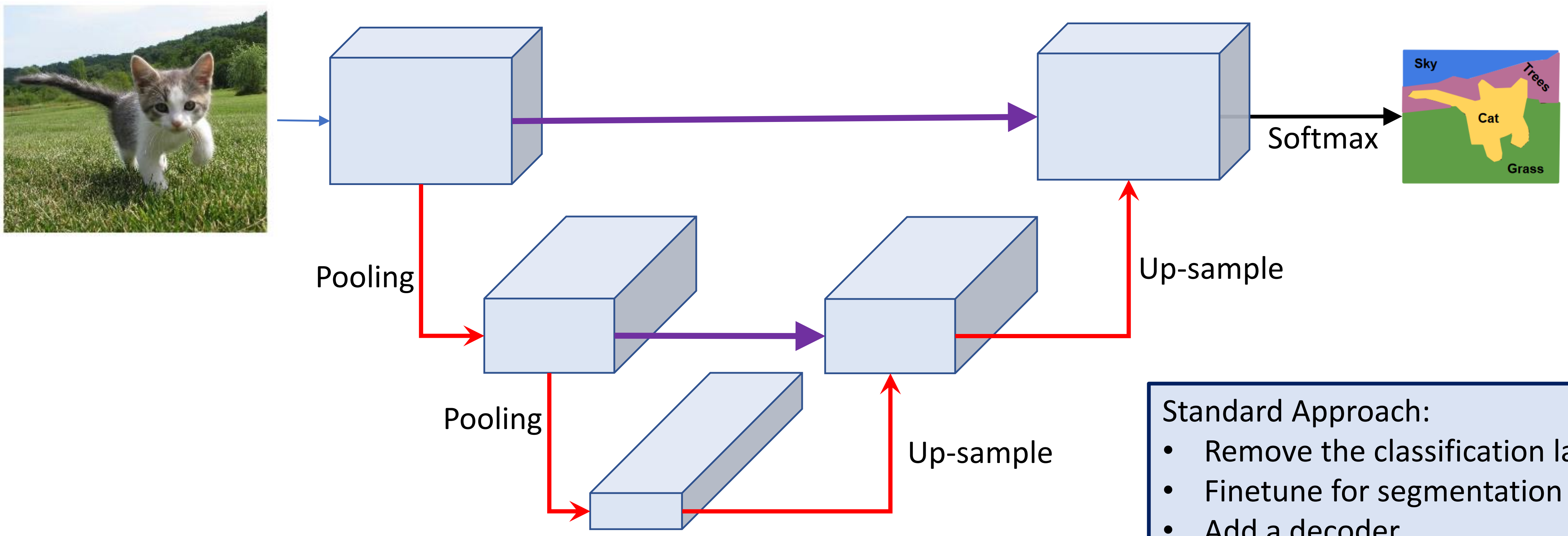
Cannot compensate the loss of information resulted in encoder

Can we improve it further?

Standard Approach:

- Remove the classification layer
- Finetune for segmentation
- Add a decoder

Segmentation Networks: Encoder-Decoder with Skip-connections

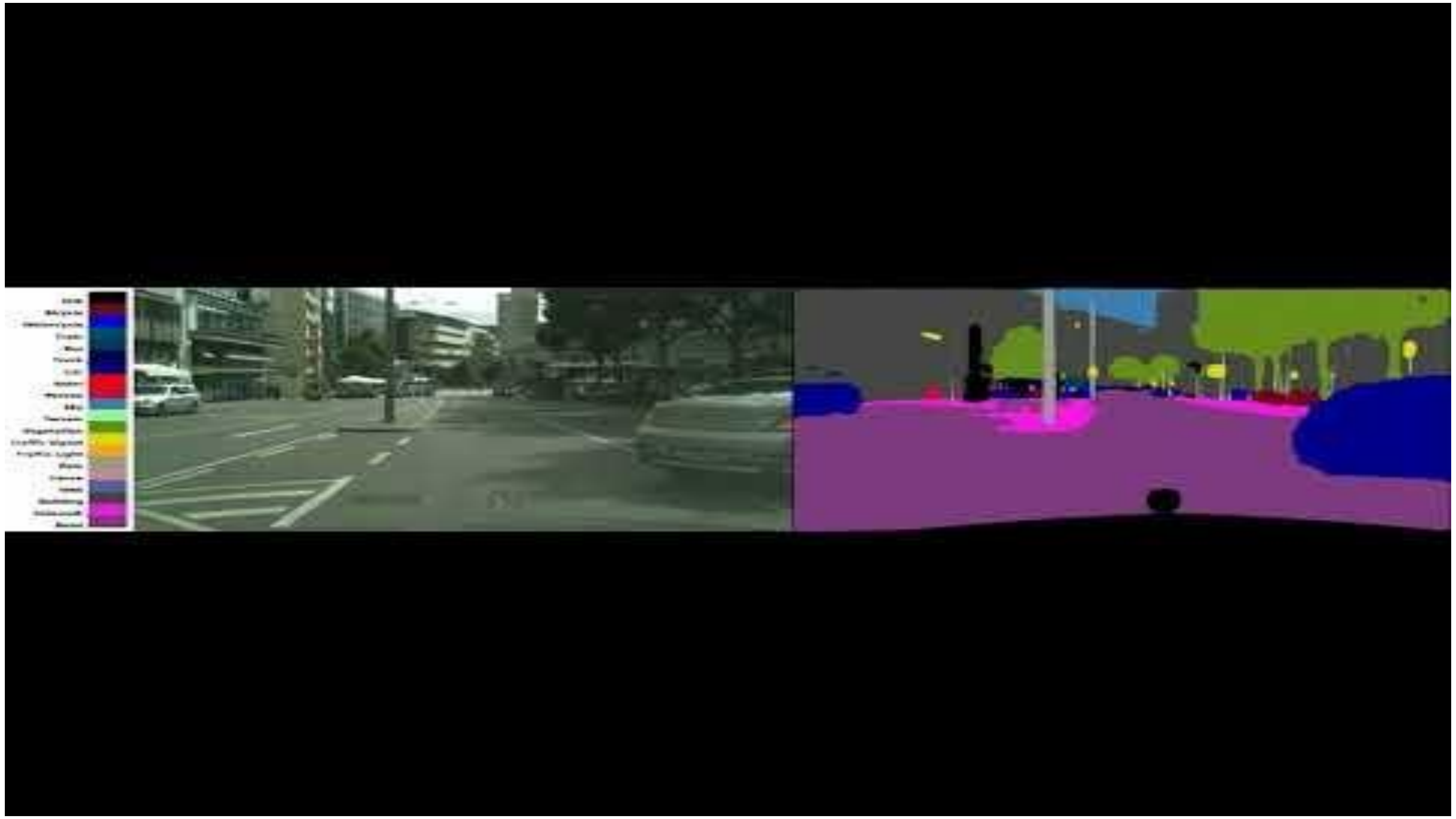


Standard Approach:

- Remove the classification layer
- Finetune for segmentation
- Add a decoder
- Share information between encoder and decoder

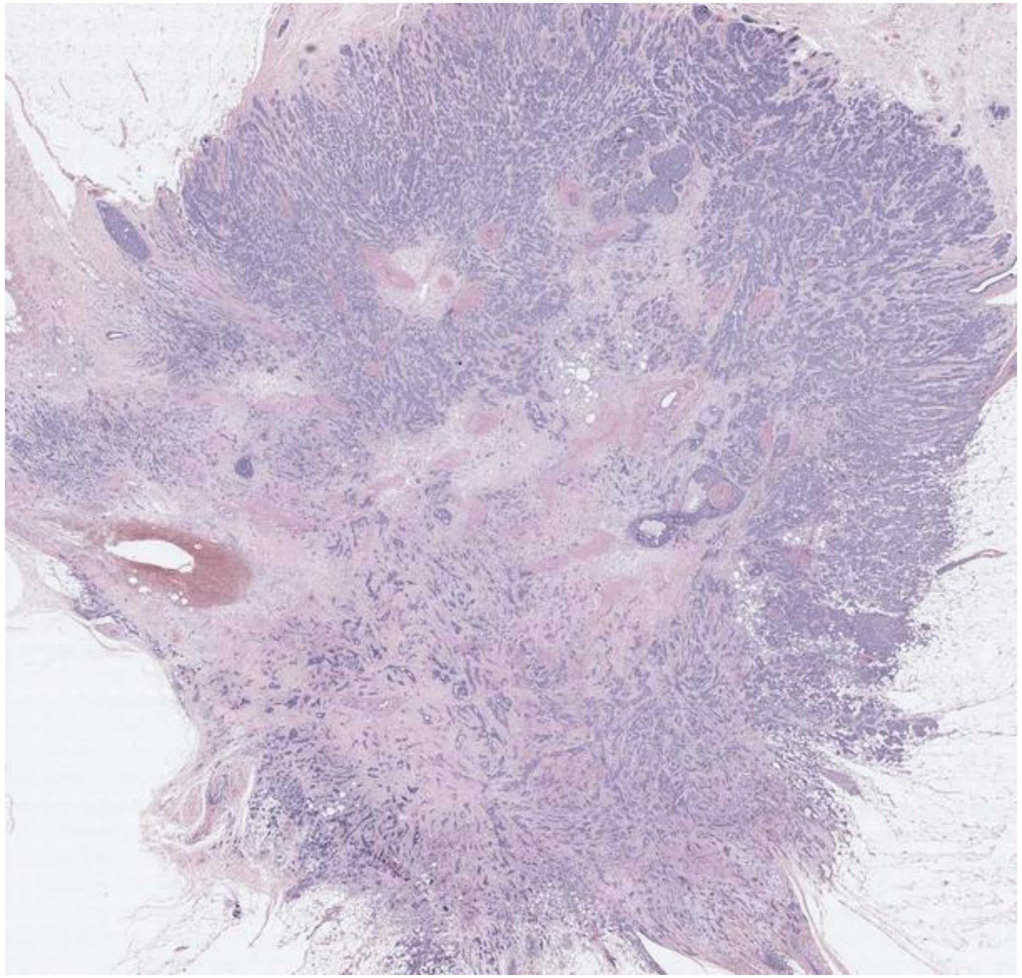
Semantic Segmentation Visualizations on Different Tasks

Semantic Segmentation results



Semantic Segmentation results (Background vs foreground)

Tissue-level segmentation in Breast Biopsy Whole Slide Images



☐ background ■ benign epithelium ☐ normal stroma ■ secretion
■ malignant epithelium ■ desmoplastic stroma ☐ blood ■ necrosis

1. **Sachin Mehta**, Ezgi Mercan, Jamen Bartlett, Donald Weaver, Joann Elmore, and Linda Shapiro. "Learning to segment breast biopsy whole slide images." WACV, 2018.
2. **Sachin Mehta**, Ezgi Mercan, Jamen Bartlett, Donald Weaver, Joann G. Elmore, and Linda Shapiro. "Y-Net: joint segmentation and classification for diagnosis of breast biopsy images." MICCAI, 2018

Tumor Lesion Segmentation in 3D Brain Images

- Enhancing tumor
- Edema
- Necrotic and non-enhancing tumor

Nicholas Nuechterlein*, and **Sachin Mehta***. "3D-ESPNet with Pyramidal Refinement for Volumetric Brain Tumor Image Segmentation." In *International MICCAI Brainlesion Workshop*, 2018. (* Equal contribution)

Thank You!!