AIL 862

Lecture 5

Does Deeper Network Perform Better?

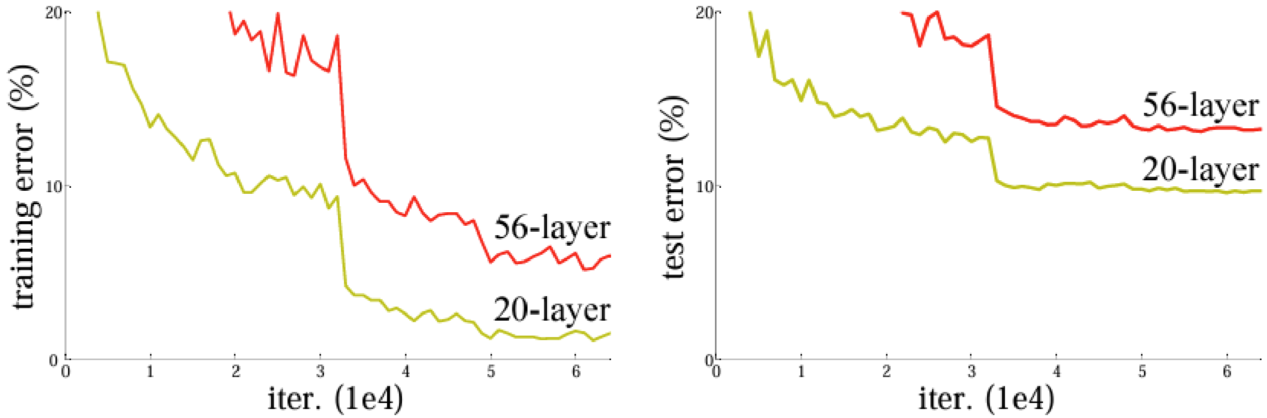
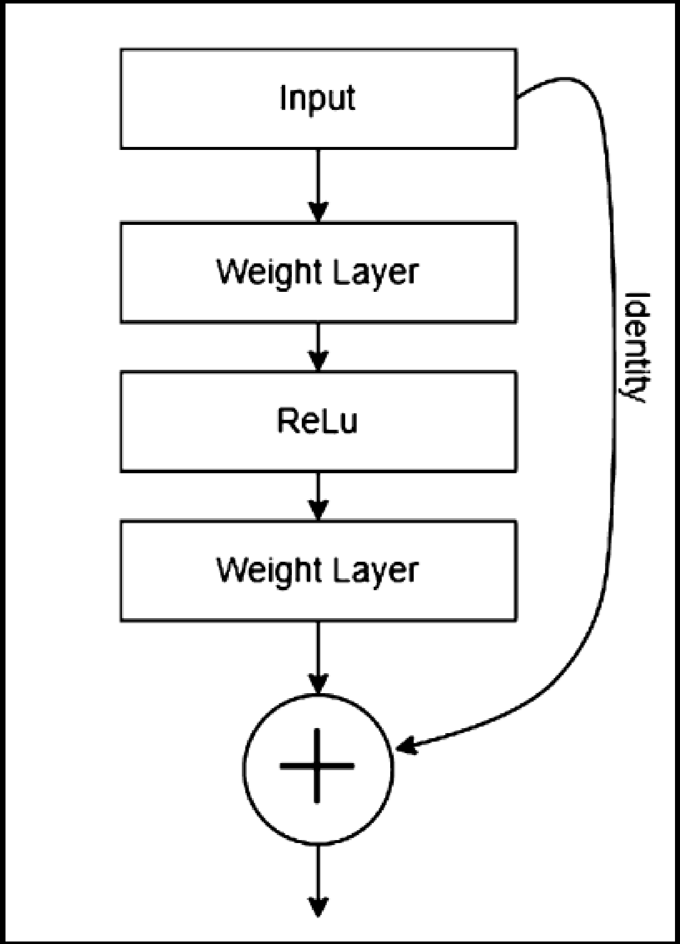


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Deep Residual Learning for Image Recognition, 2015

Residual Block



```python
def forward(self, x: Tensor) -> Tensor:
    identity = x

    out = self.conv1(x)
    out = self.bn1(out)
    out = self.relu(out)

    out = self.conv2(out)
    out = self.bn2(out)

    out += identity
    out = self.relu(out)

    return out
```

https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation.

Deep Residual Learning for Image Recognition, 2015

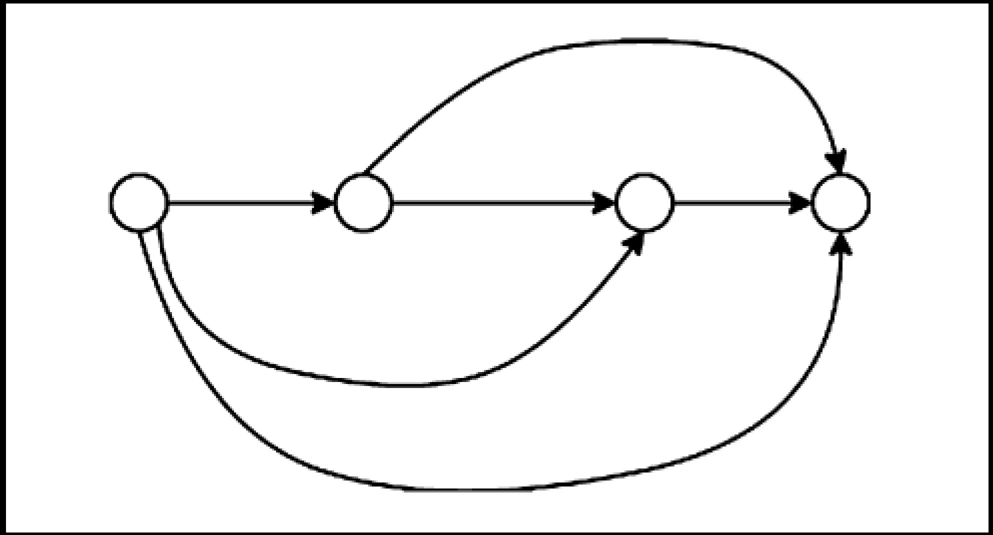
DenseNet

- Uses the idea that more connections between layers may enhance learning.

- Each layer receives input not only from the preceding layer but also from all preceding layer. Thus, each layer has direct access to the feature maps generated by all preceding layers.

- The network is divided into several densely connected blocks. Layers situated between these blocks are termed transition layer

Dense Connection



Dense Connection

$x = H([x_0, x_1, \ldots, x_{l-1}])$

$[x_0, x_1, \ldots, x_{l-1}]$ refers to the concatenation of the feature-maps produced in layers $0, \ldots, l-1$

$H(\cdot)$ is defined as a composite function of three consecutive operations: batch normalization, a ReLU and a 3×3 convolution

Transition Layers

The transition layers consist of a batch normalization layer and an 1×1 convolutional layer followed by a 2×2 average pooling layer

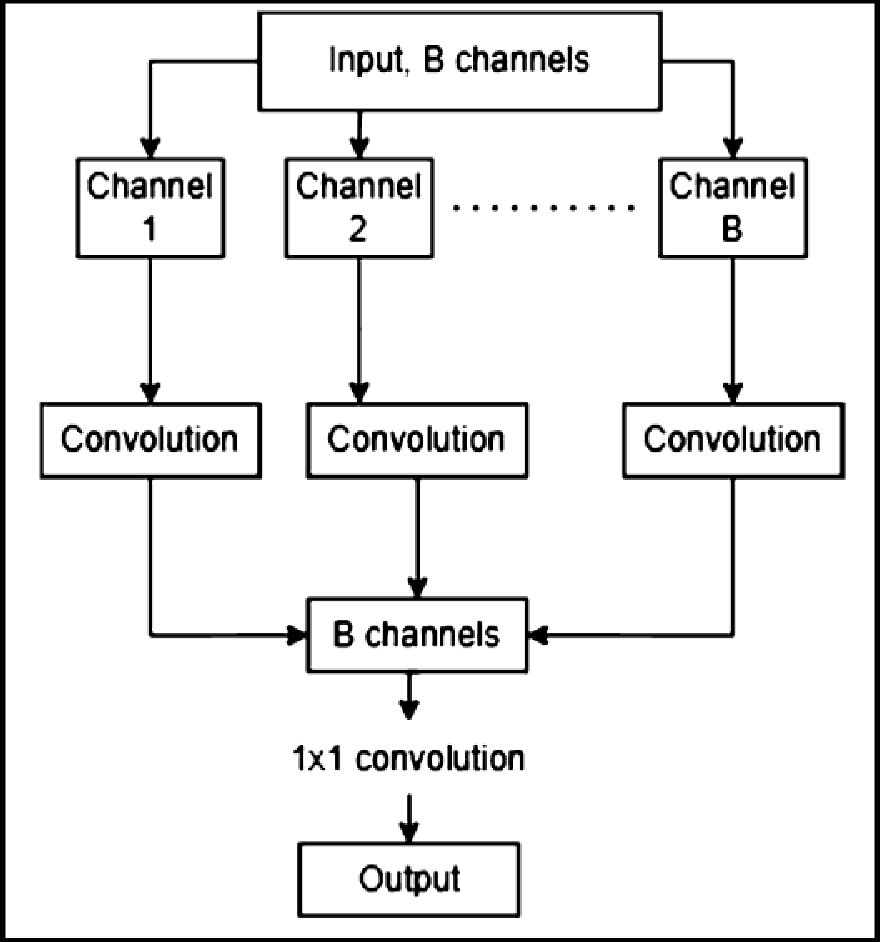
Growth Rate

- If each function H produces k feature maps, it follows that the l-th layer has $k\_0 + k \times (l-1)$ input feature-maps, where k_0 is the number of channels in the input layer.
- Densenet typically uses small k value, e.g., k=12
- k is called growth rate

MobileNet – Depthwise Separable Convolution

In standard convolutions, input features are filtered and combined into output features in a single operation. In contrast, depthwise separable convolutions divide this process into two distinct layers—one for spatial filtering and the other for channel-wise combination

MobileNet – Depthwise Separable Convolution



Reduction factor - ?

Image Segmentation

- ***Split image into different regions***
  Different regions usually cover the whole image
  - Different regions usually do not overlap
- ***Similarity predicate***
  Satisfied by each region
  - Not satisfied by union of different regions
- ***Can be subjective***
  Depending on how we define the notion of similarity

Before Deep Learning

- ***Region growing***
  - Start with one pixel of a potential region and try to grow until pixels being compared are too dissimilar

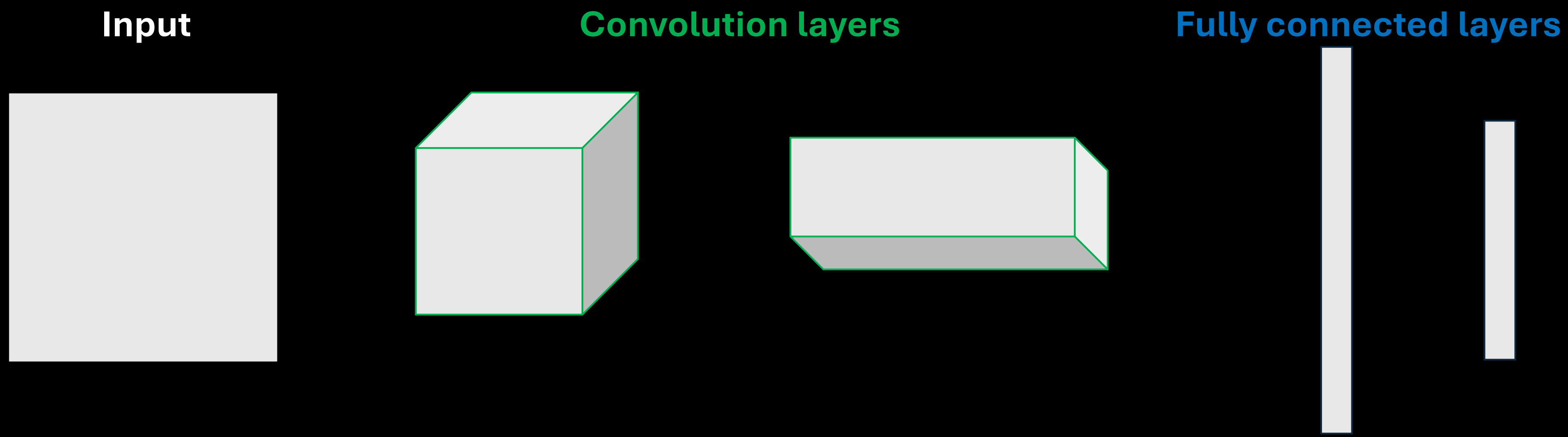
- ***Clustering***

- ***Split and merge***

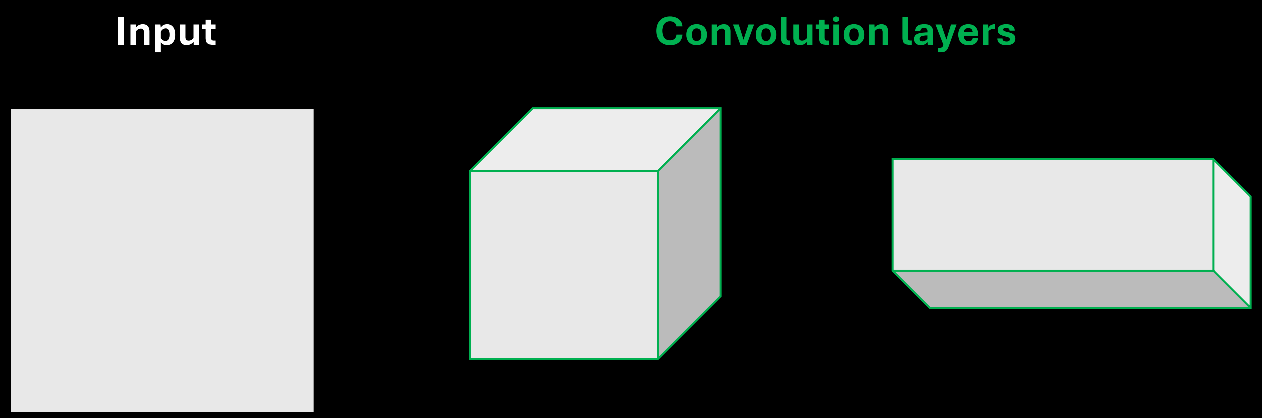
Input Space for Clustering

- Color
- Texture features

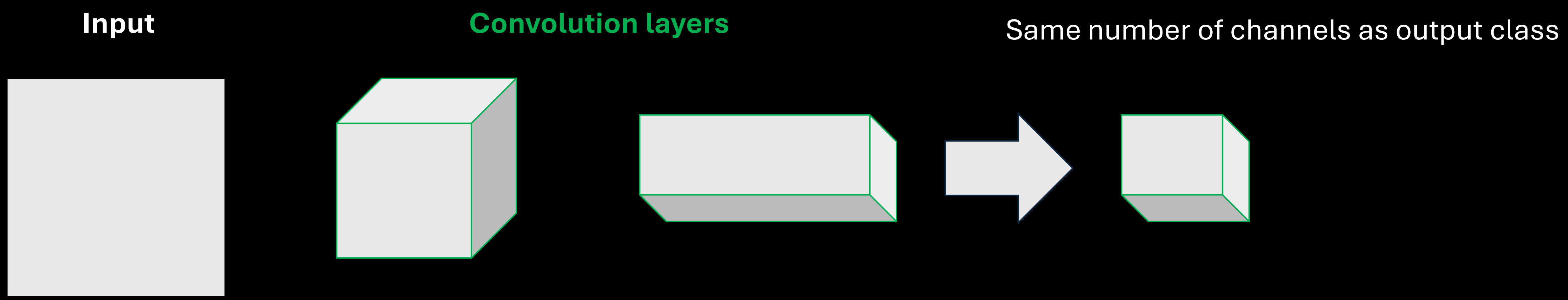
Typical Classification Network



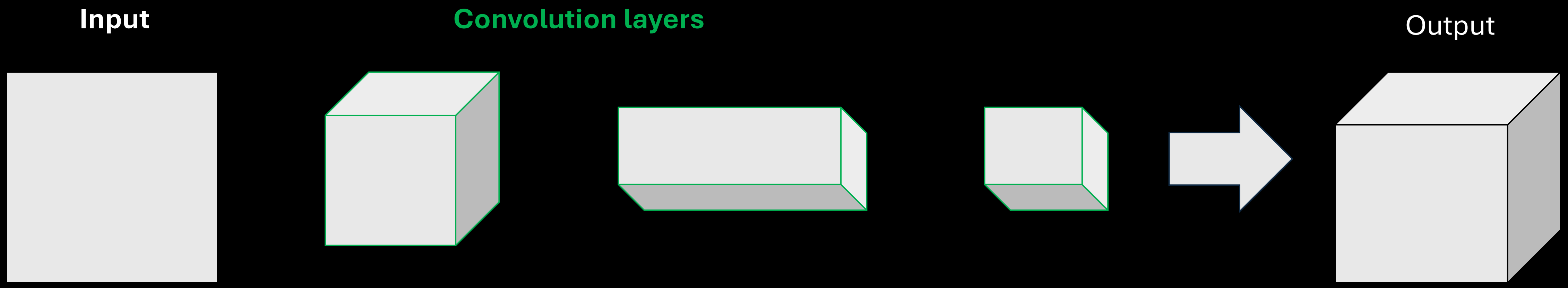
Fully Convolutional



Fully Convolutional



Fully Convolutional



Classifier to Semantic Segmentation

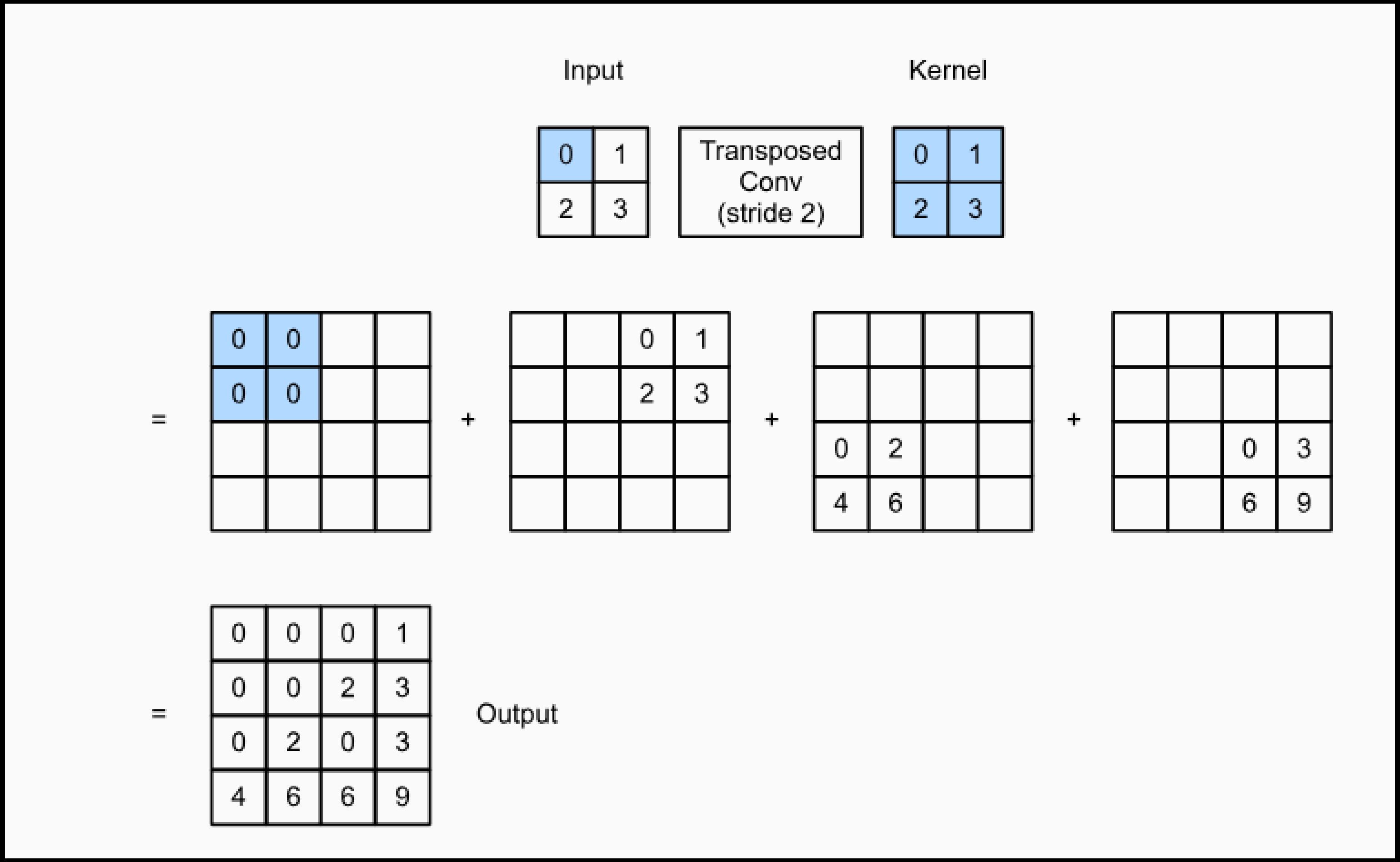
Convolutionalize the classification architectures: AlexNet, VGGNet

Remove classification layer

Use 1x1 convolution with required number of channel dimensions and upsample

Or replace the last step with transposed convolution.

Transposed Convolution



Transposed Convolution

```
# Input
input = torch.tensor([[0.0, 1.0], [2.0, 3.0]])
#Kernel
kernel1 = torch.tensor([[[0.0, 1.0], [2.0, 3.0]]])
kernel2 = torch.tensor([[[4.0, 1.0], [2.0, 3.0]]])

# Redefine the shape in 4 dimension
input = input.reshape(1, 1, 2, 2)
kernel1 = kernel1.reshape(1, 1, 2, 2)
kernel2 = kernel2.reshape(1, 1, 2, 2)
```

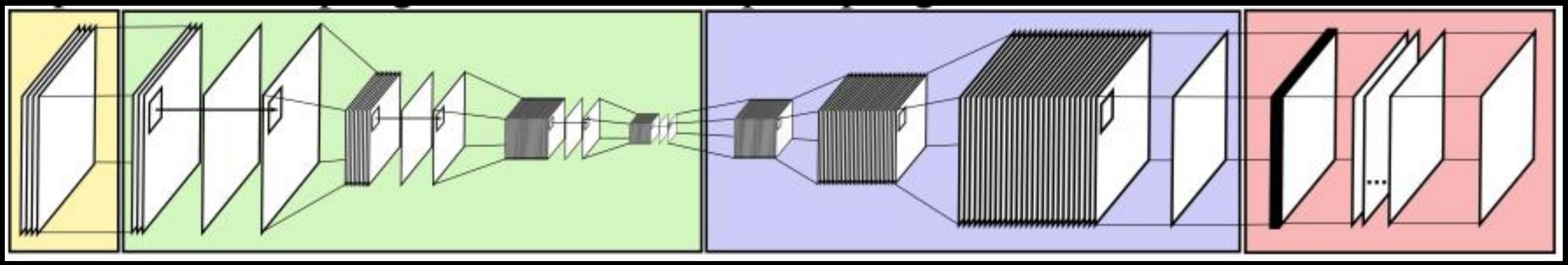
Transposed Convolution

```
# Transpose convolution Layer
transpose = nn.ConvTranspose2d(in_channels =1,
                               out_channels =1,
                               kernel_size=2,
                               stride = 2,
                               padding=0,
                               bias=False)

# Initialize Kernel
transpose.weight.data = kernel1
# Output value
output = transpose(input)

print(output)


# Initialize Kernel
transpose.weight.data = kernel2
# Output value
output = transpose(input)

print(output)
```

Transposed Convolution

```
tensor([[[[0., 0., 0., 1.],
          [0., 0., 2., 3.],
          [0., 2., 0., 3.],
          [4., 6., 6., 9.]]]], grad
tensor([[[[ 0.,  0.,  4.,  1.],
          [ 0.,  0.,  2.,  3.],
          [ 8.,  2., 12.,  3.],
          [ 4.,  6.,  6.,  9.]]]],
```

CNN for Semantic Segmentation of EO Images



Dense Semantic Labeling of Subdecimeter Resolution Images With Convolutional Neural Networks, 2017

FCN

```python
def forward(self, x):
    output = self.pretrained_net(x)
    x5 = output['x5']  # size=(N, 512, x.H/32, x.W/32)
    x4 = output['x4']  # size=(N, 512, x.H/16, x.W/16)

    score = self.relu(self.deconv1(x5))
    score = self.bn1(score + x4)
    score = self.bn2(self.relu(self.deconv2(score)))
    score = self.bn3(self.relu(self.deconv3(score)))
    score = self.bn4(self.relu(self.deconv4(score)))
    score = self.bn5(self.relu(self.deconv5(score)))
    score = self.classifier(score)

    return score  # size=(N, n_class, x.H/1, x.W/1)
```

https://github.com/pochih/FCN-pytorch/blob/master/python/fcn.py