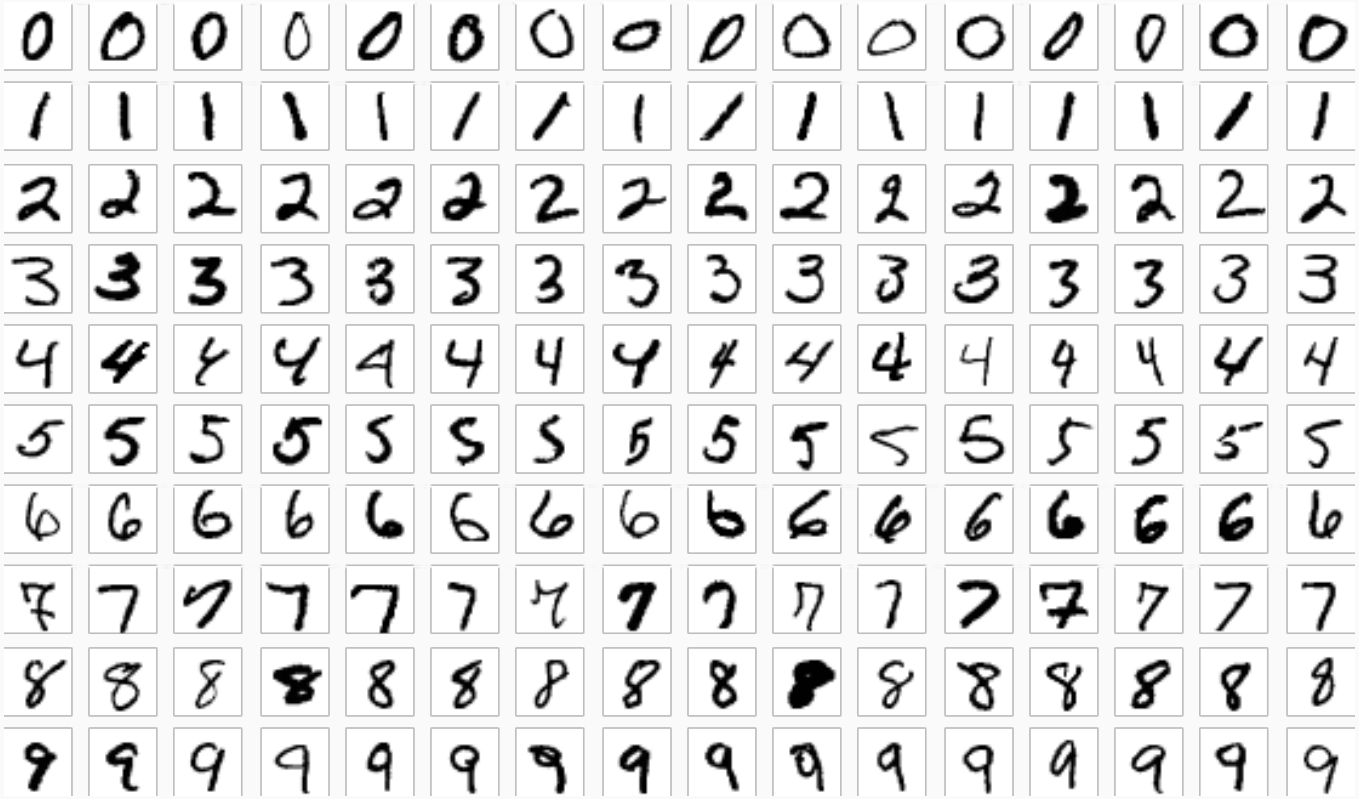
Computer Vision with Convolutional Neural Networks

Sheikh Azizul Hakim

Compilation Time: Saturday 16th November, 2024, 03:29

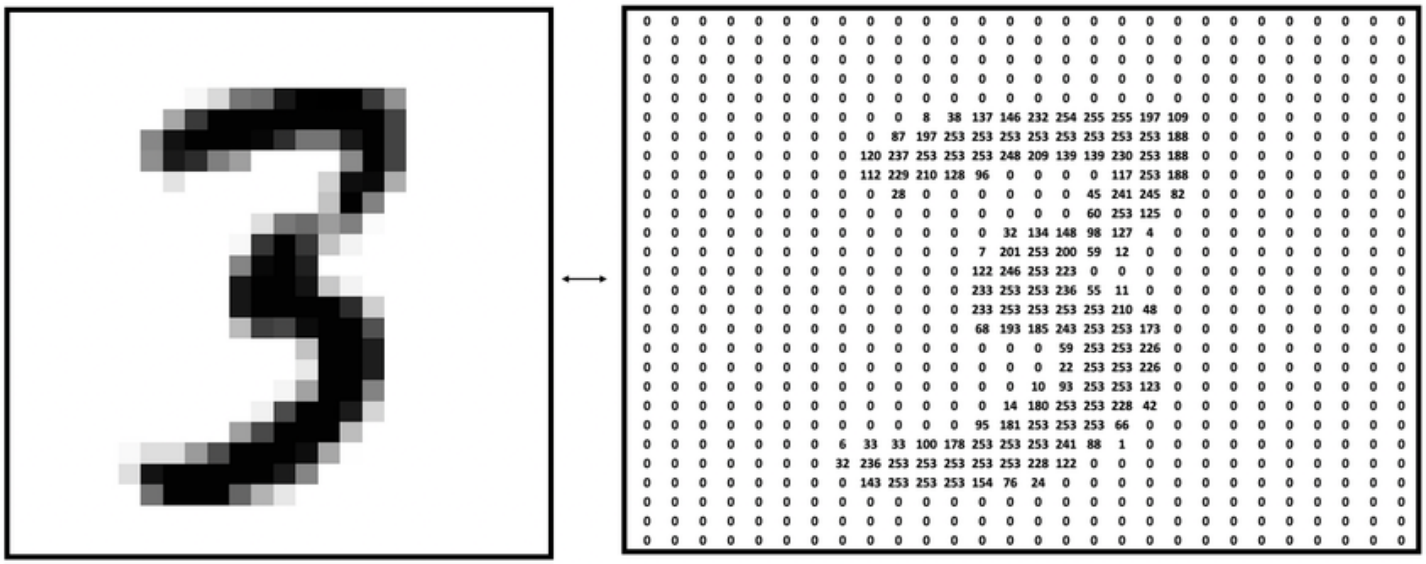
Department of CSE, BUET

A Rudimentary Vision Problem



Given the image of a handwritten digit, identify it.

Images are just tensors of numbers



Idea I

1. Flatten the image

Idea I

1. Flatten the image
2. Feed a vanilla neural network

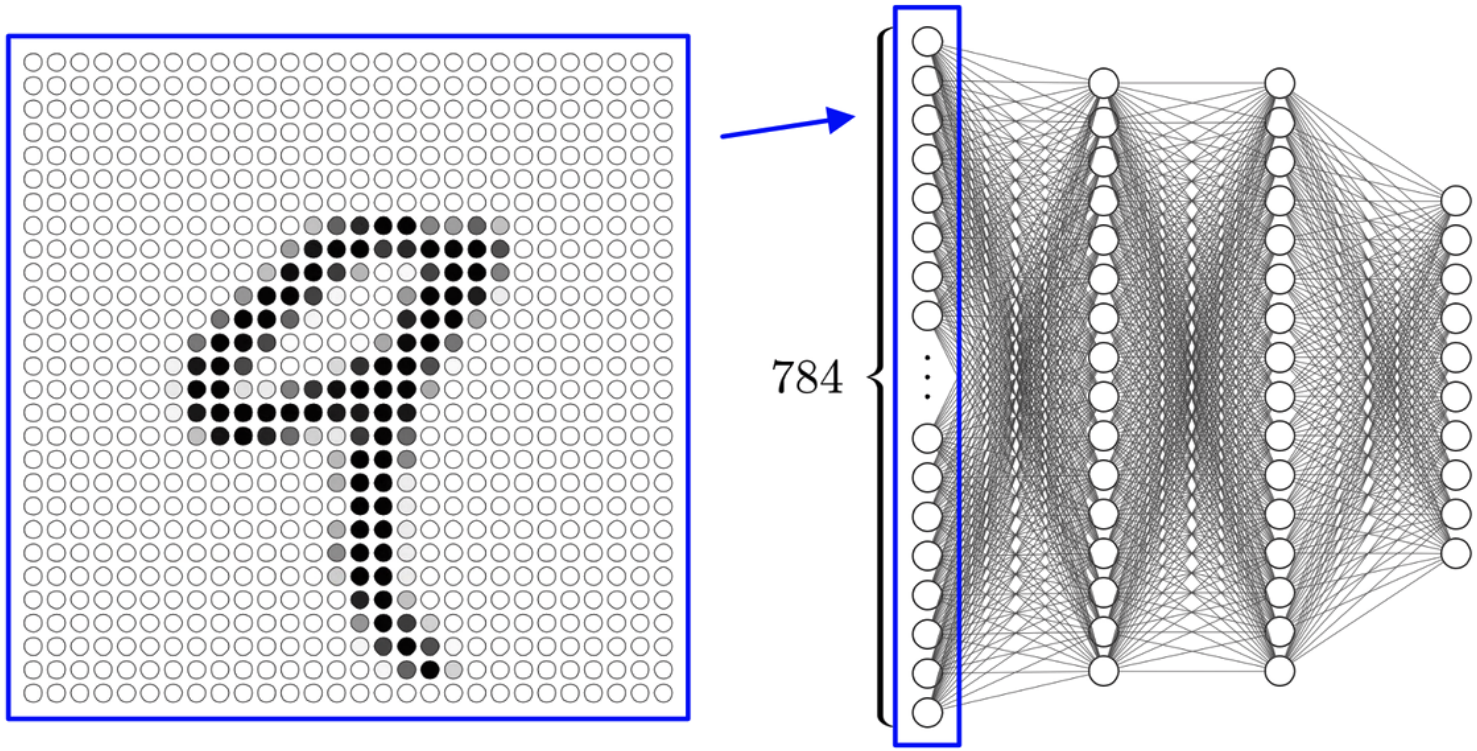
Idea I

1. Flatten the image
2. Feed a vanilla neural network
3. Get $> 95\%$ accuracy

Idea I

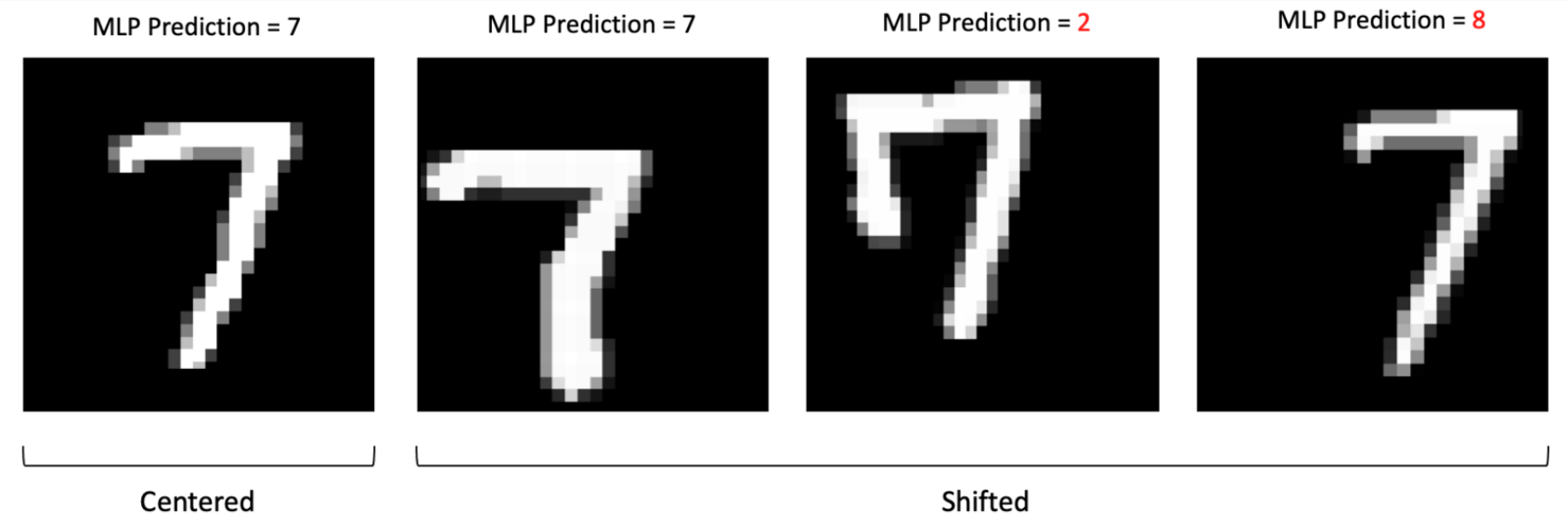
1. Flatten the image
2. Feed a vanilla neural network
3. Get $> 95\%$ accuracy
4. End of course

Idea I



What is the problem?

MLP has no knowledge of `spatial invariance`



What Waldo looks like **does not depend upon** where Waldo is located.

Intuition I

In the earliest layers, our network should respond similarly to the same patch, regardless of where it appears in the image.

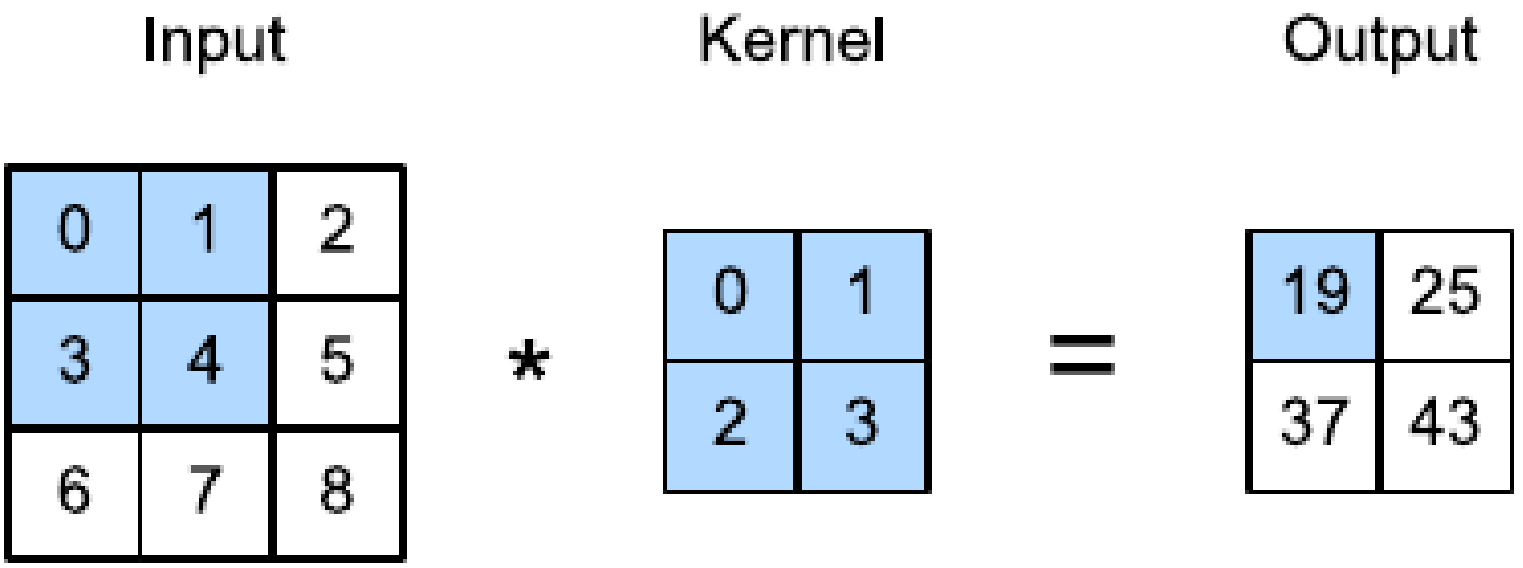
Intuition II

The earliest layers of the network should focus on local regions, without regard for the contents of the image in distant regions.

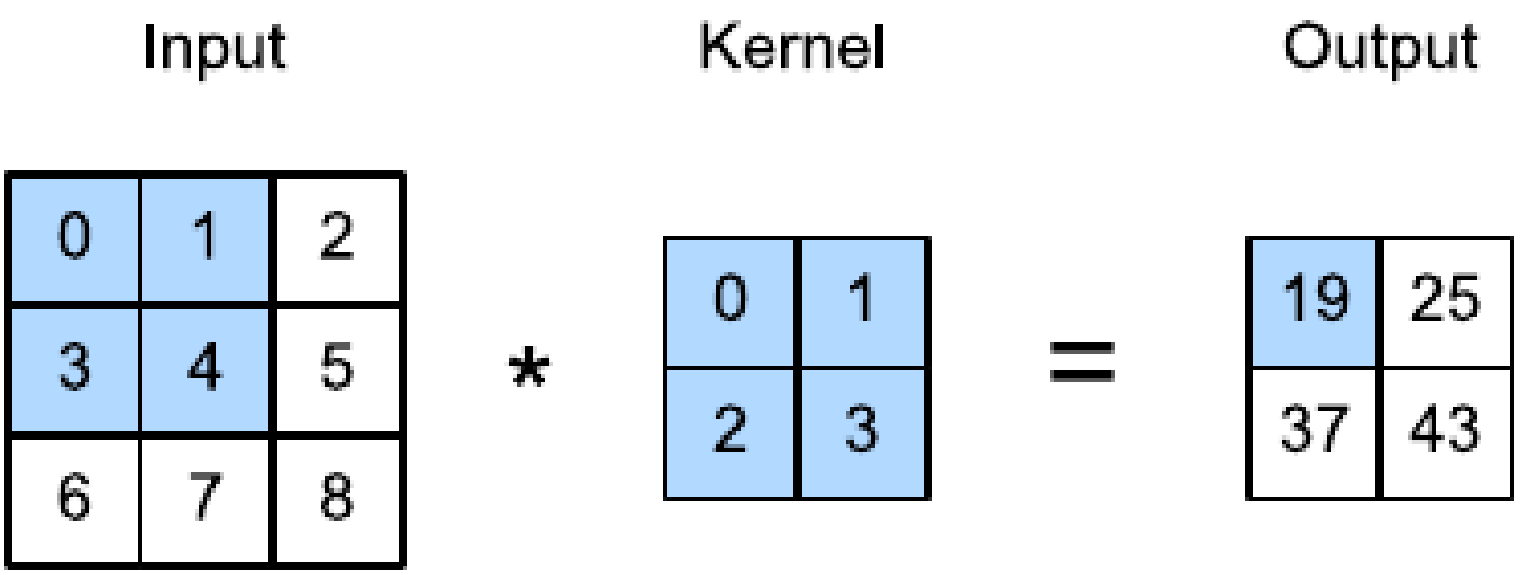
Intuition III

As we proceed, deeper layers should be able to capture longer-range features of the image, in a way similar to higher level vision in nature.

Convolution Operation



Convolution Operation



$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$
$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$
$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$
$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$

Convolution Code in Pytorch

```
import torch
from torch import nn

def conv2d(X, K):
    """Compute 2D Convolution"""
    h, w = K.shape
    Y = torch.zeros((X.shape[0] - h + 1, X
        .shape[1] - w + 1))
    for i in range(Y.shape[0]):
        for j in range(Y.shape[1]):
            Y[i, j] = (X[i:i + h, j:j + w]
                * K).sum()
    return Y
```

Backpropagation of Convolution

Click this link for details (self study).

Backpropagation of Convolution

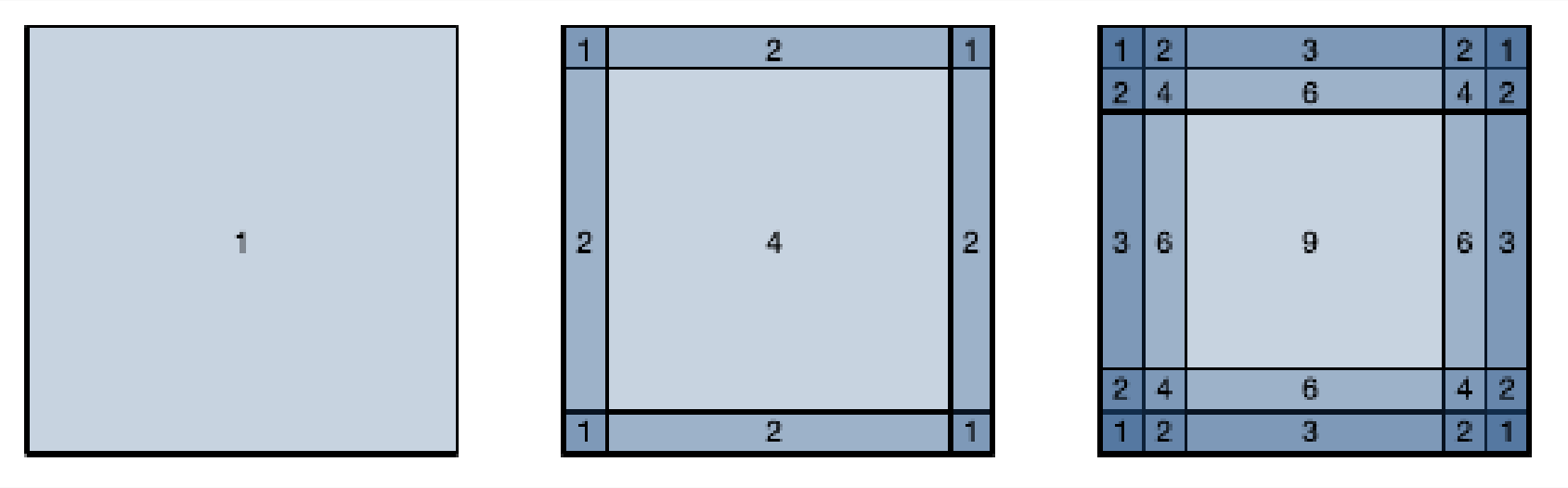
Click this link for details (self study).

In frameworks like PyTorch, defining the forward operation is sufficient because the autograd system automatically computes gradients during backpropagation, eliminating the need for manual gradient calculations in most cases.

Convolution Code in Pytorch

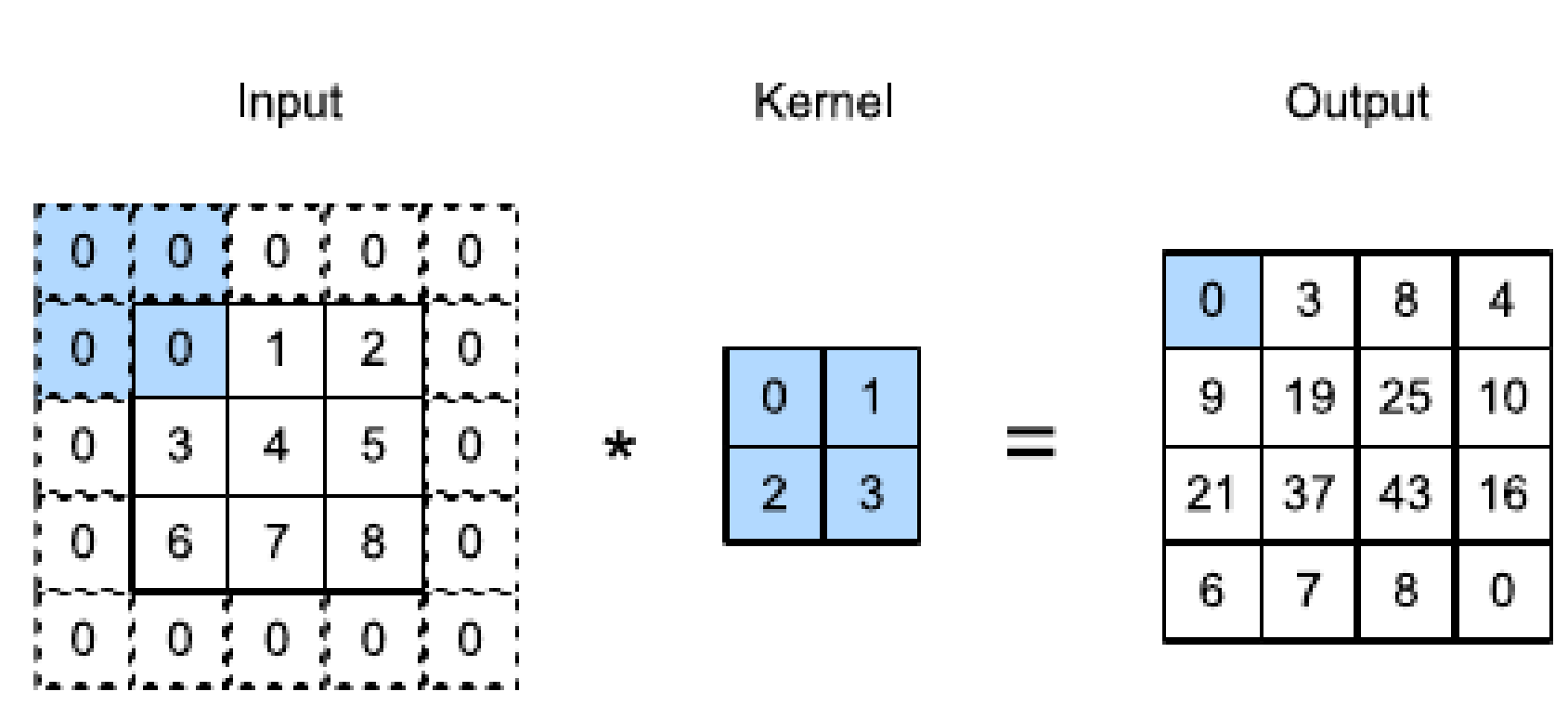
```
class Conv2D(nn.Module):
    def __init__(self, krnl_sz):
        super().__init__()
        self.weight = nn.Parameter(torch.
            rand(krnl_sz))
        self.bias = nn.Parameter(torch.
            zeros(1))

    def forward(self, x):
        return conv2d(x, self.weight) +
            self.bias
```

Cornel Pixels Relatively Unutilized



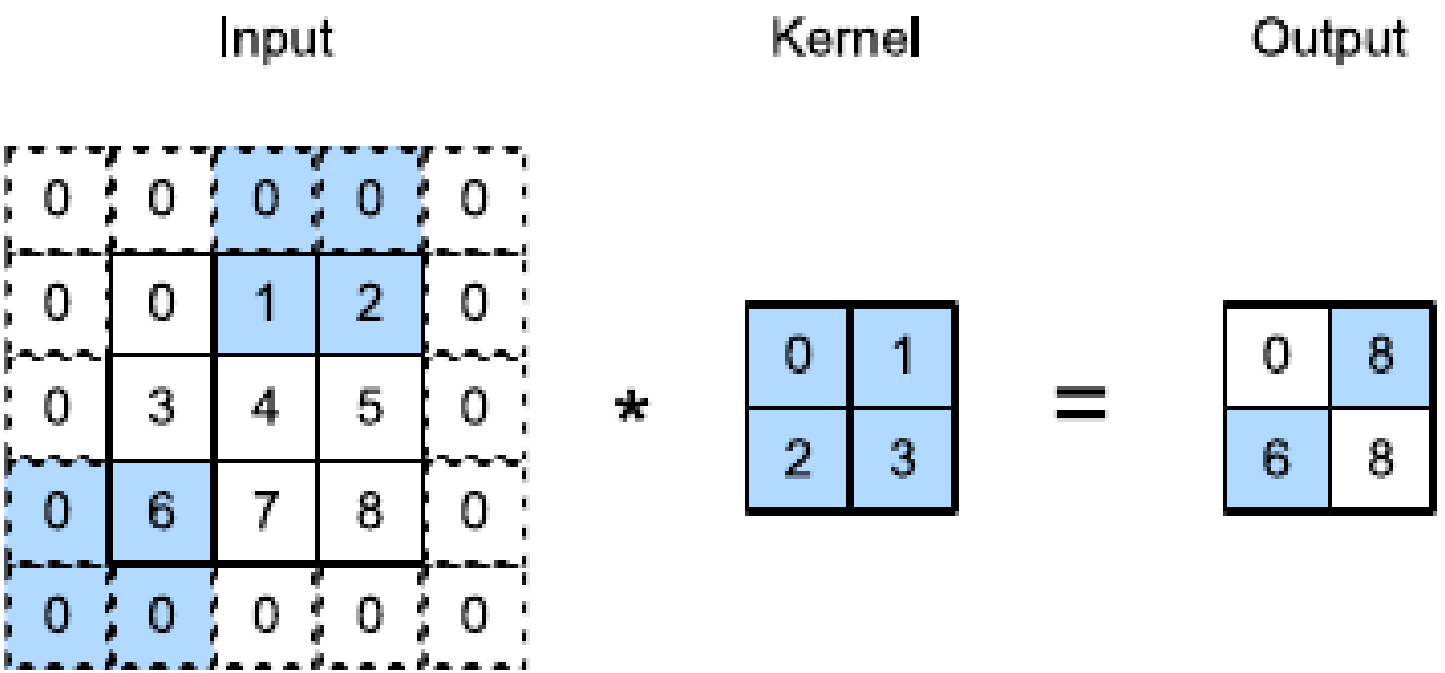
Pixel utilization for convolutions of size $1 \times 1, 2 \times 2, and 3 \times 3$ respectively.

Padding



An example of zero padding. Another option is to pad with the nearby pixel values.

Stride



Cross-correlation with strides of 3 and 2 for height and width, respectively.

Output Shape of Convolutional Layer

$$\text{Output Shape} = \left\lfloor \frac{n_h - k_h + p_h + s_h}{s_h} \right\rfloor \times \left\lfloor \frac{n_w - k_w + p_w + s_w}{s_w} \right\rfloor$$

where:

n_h : Input height

n_w : Input width

k_h : Kernel height

k_w : Kernel width

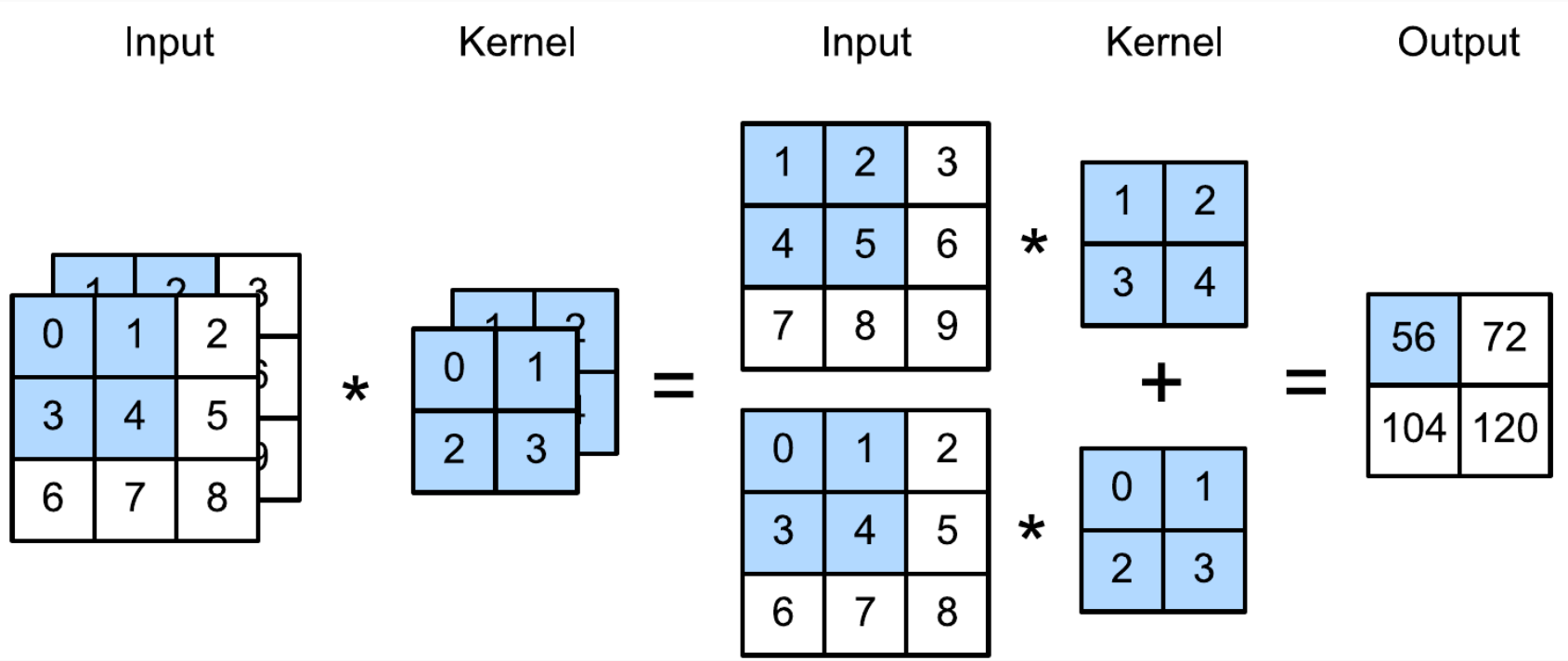
p_h : Padding height

p_w : Padding width

s_h : Stride height

s_w : Stride width

Multiple Input Channels



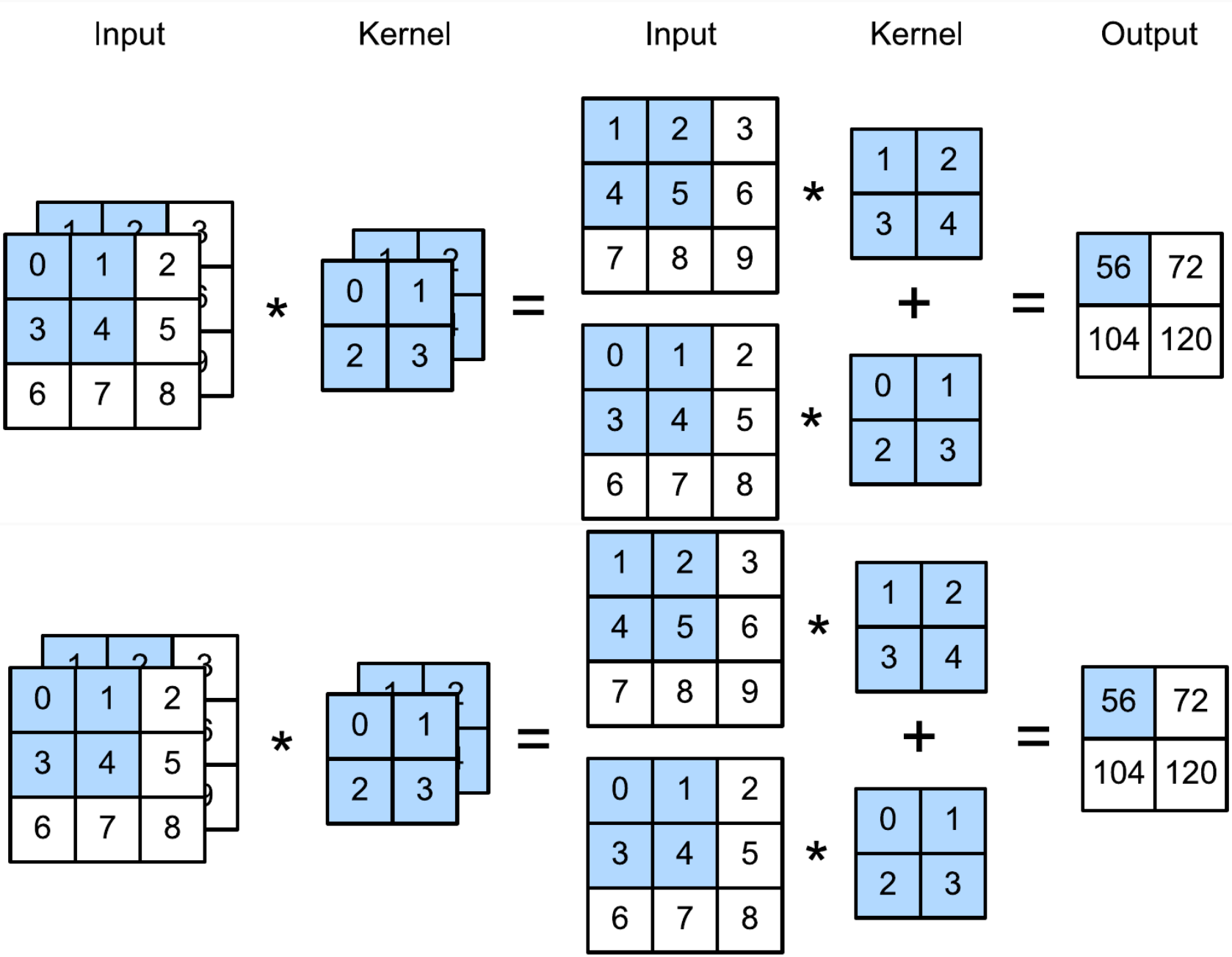
Processing an image with n channels needs a kernel with n channels.

Regardless of the number of input channels, so far we always ended up with one output channel.

Regardless of the number of input channels, so far we always ended up with one output channel.

How to produce multiple output channels?

Multiple Output Channels

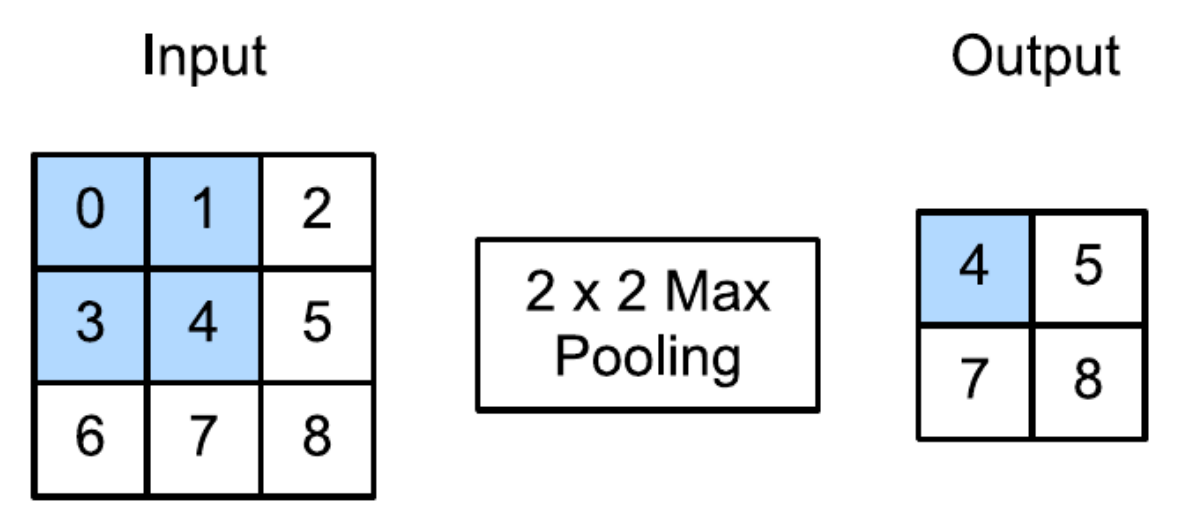


Processing an image with n channels needs a kernel with n channels.

Processing an image with n channels needs a kernel with n channels.

The number of output chanels is the number of kernels.

Pooling



Two pooling methods are popular: max pooling (shown above) and average pooling.

We can also apply padding and stride with pooling.

Does a pooling layer have a kernel of weights?

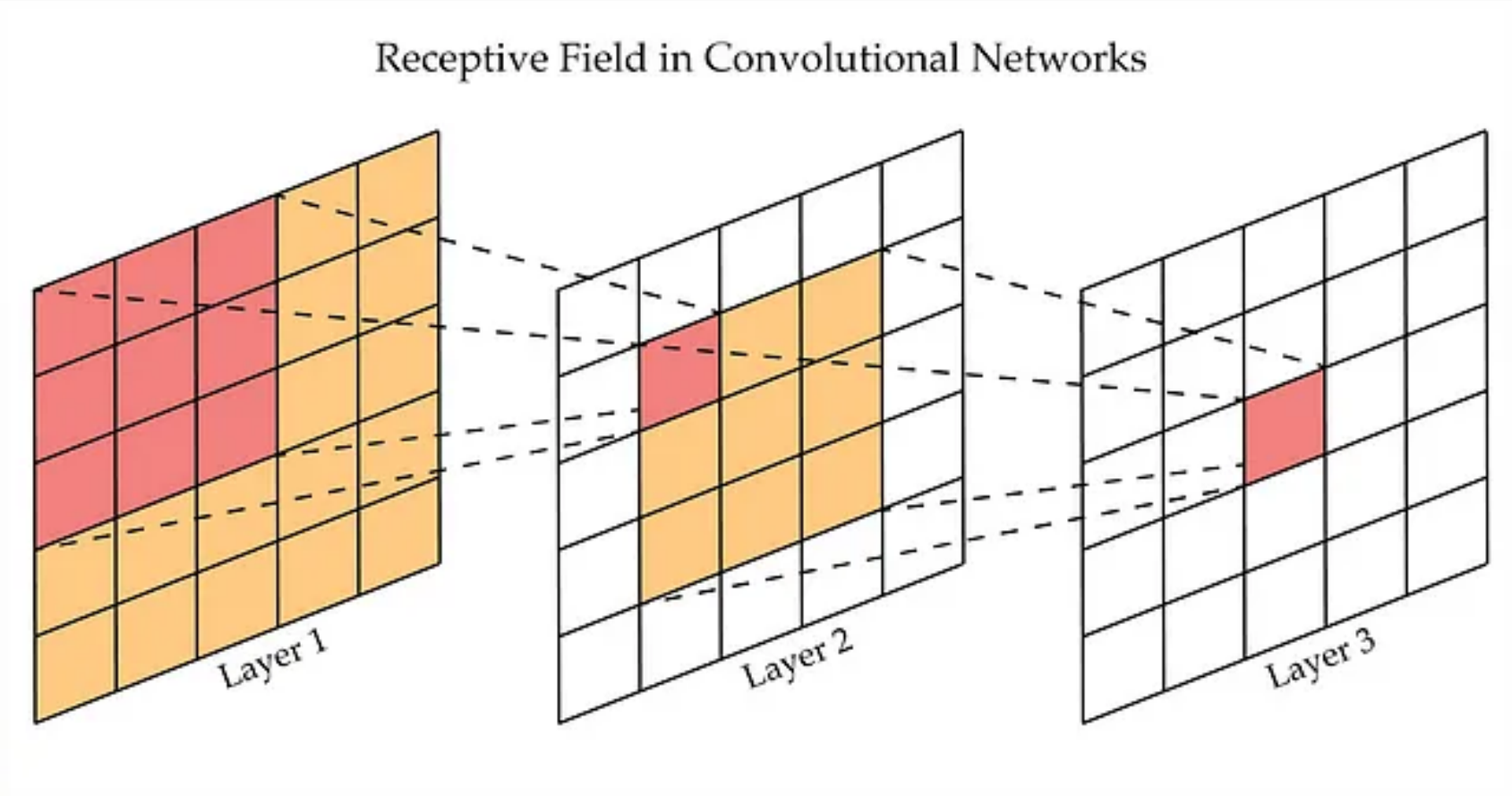
Does a pooling layer have a kernel of weights?

No. Pooling layers, unlike convolutional layers, do not have learnable parameters.

Benfits of Pooling

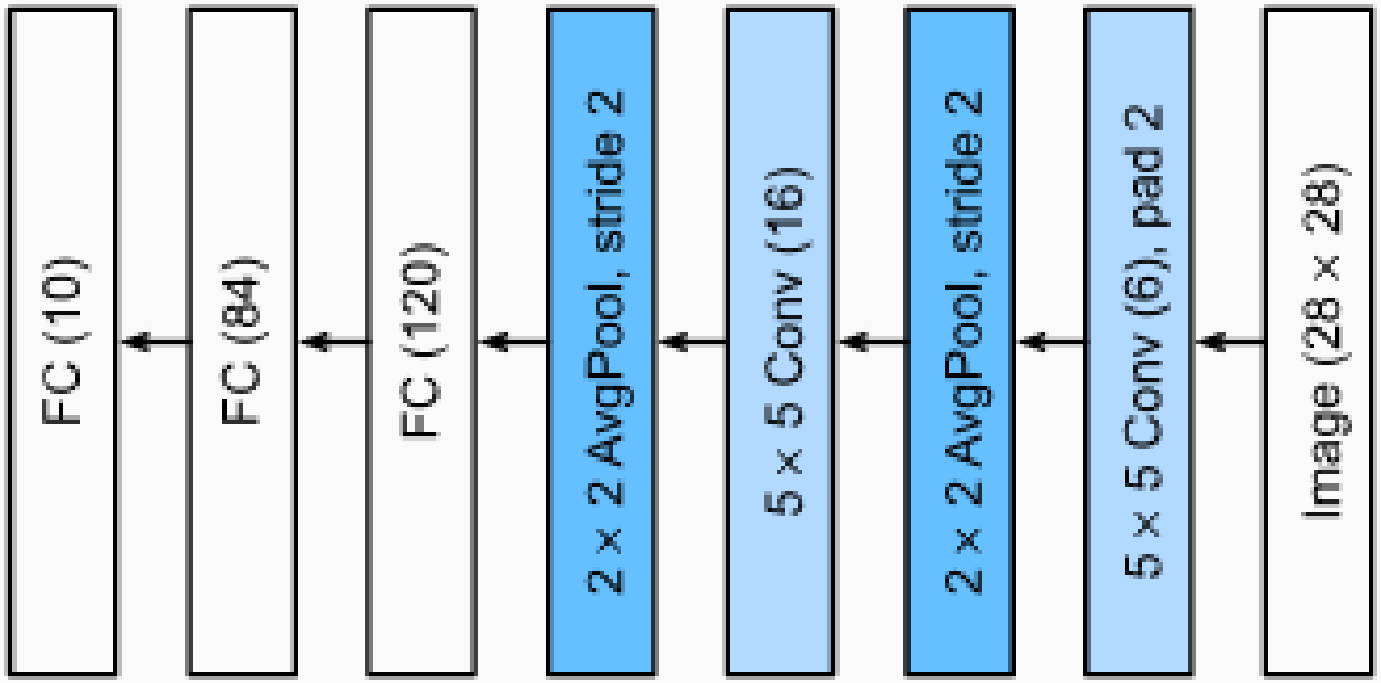
1. Spatially downsampling representations
2. Mitigating the sensitivity of convolutional layers to location

Feature Map



The deeper we go in the network, the larger the receptive field (relative to the input) to which each hidden node is sensitive. Reducing spatial resolution accelerates this process, since the convolution kernels cover a larger effective area.

LeNet: An Early CNN Architecture



```
class LeNet(nn.Module):
  def __init__(self, lr=0.1
          , num_classes=10):
    self.net = nn.Sequential(
        nn.LazyConv2d(6, krnl_sz=5, padding
            =2),
        nn.Sigmoid(),
        nn.AvgPool2d(krnl_sz=2, stride=2),
        nn.LazyConv2d(16, krnl_sz=5),
        nn.Sigmoid(),
        nn.AvgPool2d(krnl_sz=2, stride=2),
        nn.Flatten(),
        nn.LazyLinear(120), nn.Sigmoid(),
        nn.LazyLinear(84), nn.Sigmoid(),
        nn.LazyLinear(num_classes)
    )
```

Recommended Reading

https://d2l.ai/chapter_convolutional-neural-networks/index.html

CNN Architectures

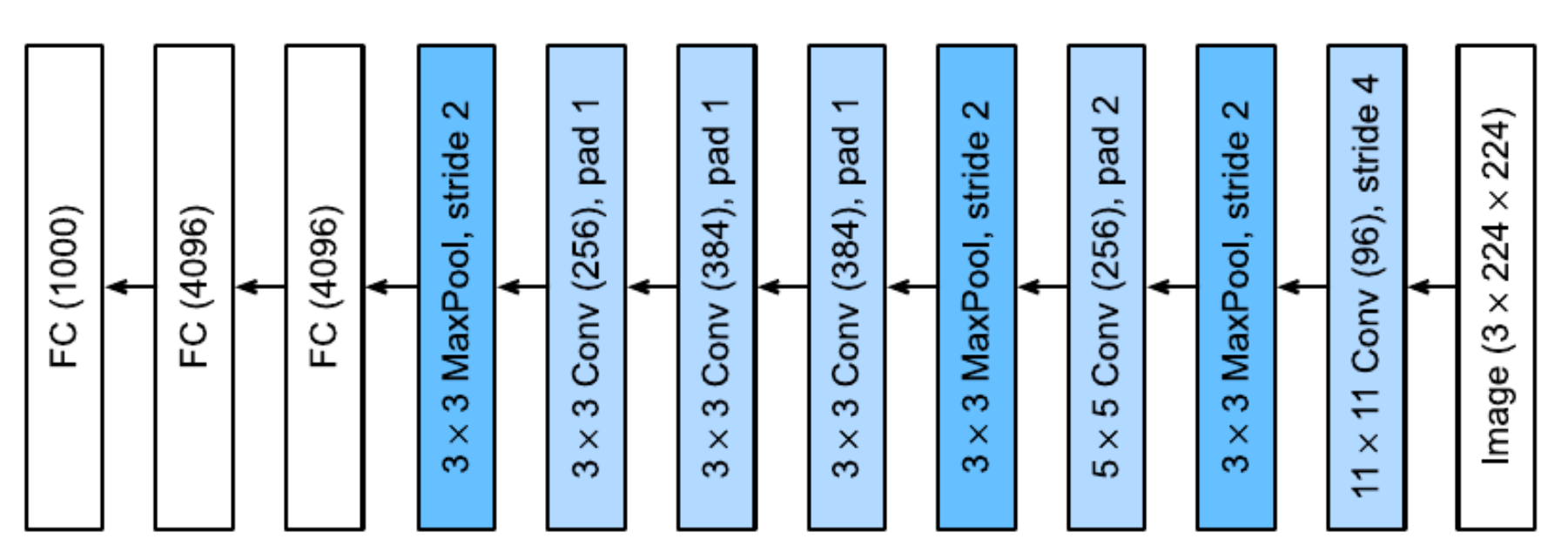
Computer Vision in `the bad old days`

1. Obtain an interesting dataset.
2. Preprocess the dataset with hand-crafted features based on some knowledge of optics, geometry, and other analytic tools.
3. Feed the data through a standard set of feature extractors, or any number of other hand-tuned pipelines.
4. Dump the resulting representations into your favorite classifier, likely a linear model or kernel method, to train a classifier.

Computer Vision in `the bad old days`

While machine learning researchers believed their field was important, beautiful, thriving, rigorous, and eminently useful, computer vision researchers justifiably believed that a slightly bigger or cleaner dataset or a slightly improved feature-extraction pipeline mattered far more to the final accuracy than any learning algorithm.

AlexNet: the first `modern` CNN Architecture



Representation learning

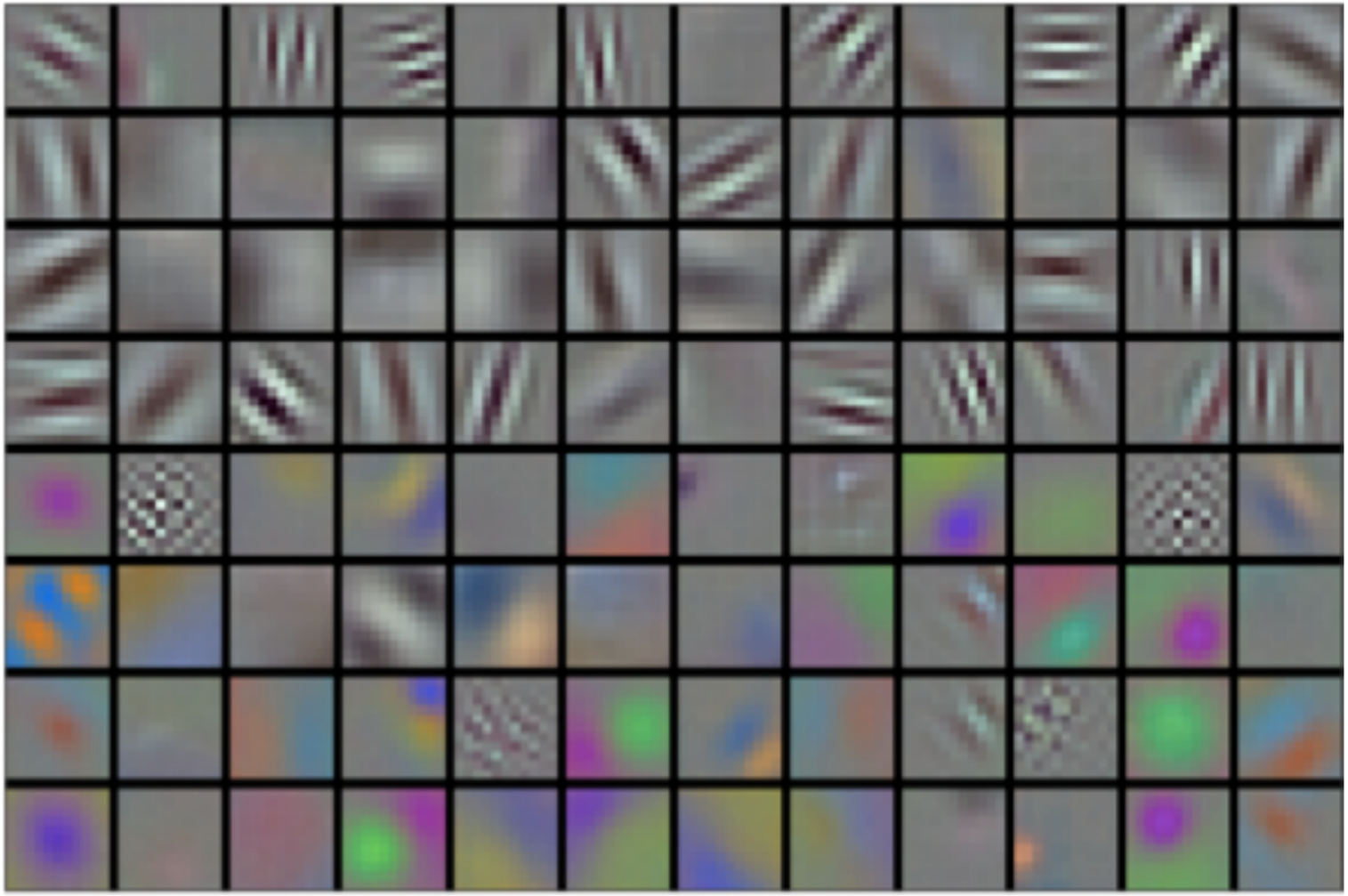


Image filters learned by the first layer of AlexNet.

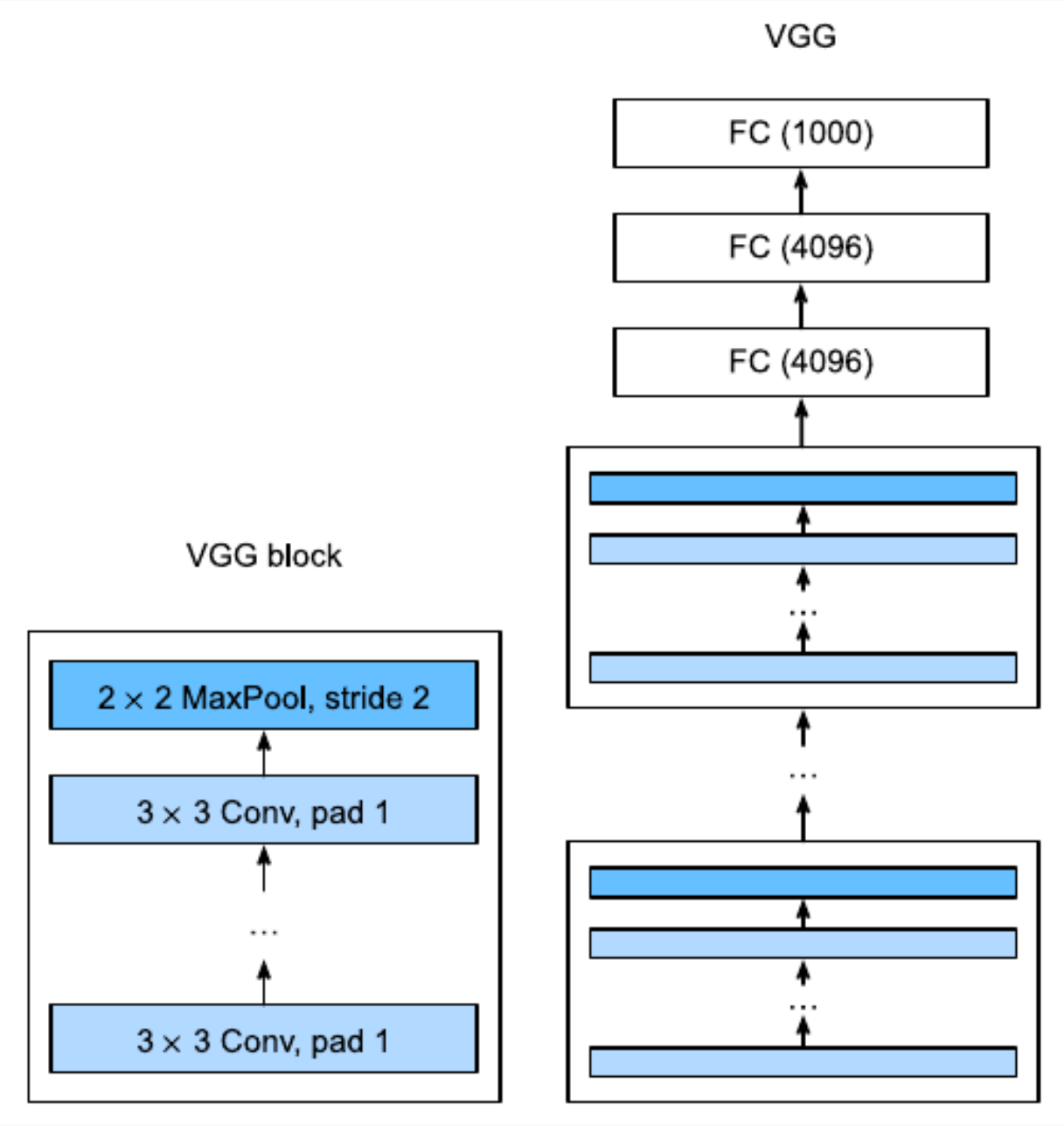
AlexNet (2012) and its precursor LeNet (1995) share many architectural elements.

Why did it take so long?

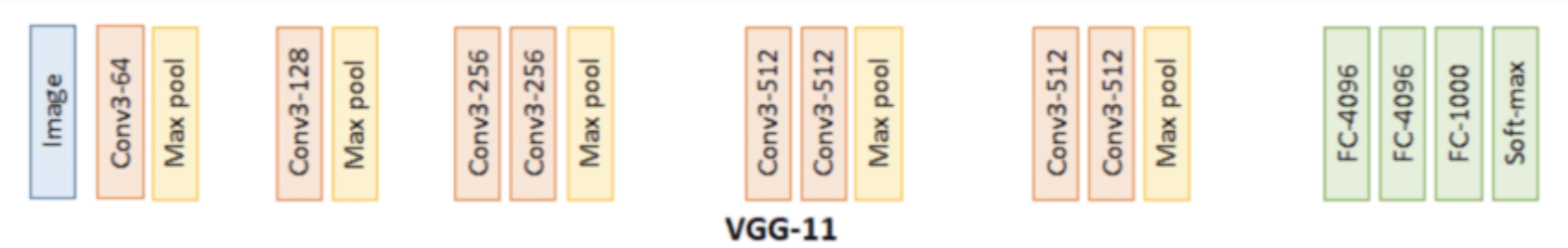
Missing Ingredients

- Data [ImageNet Dataset]
- Hardware [GPUs]

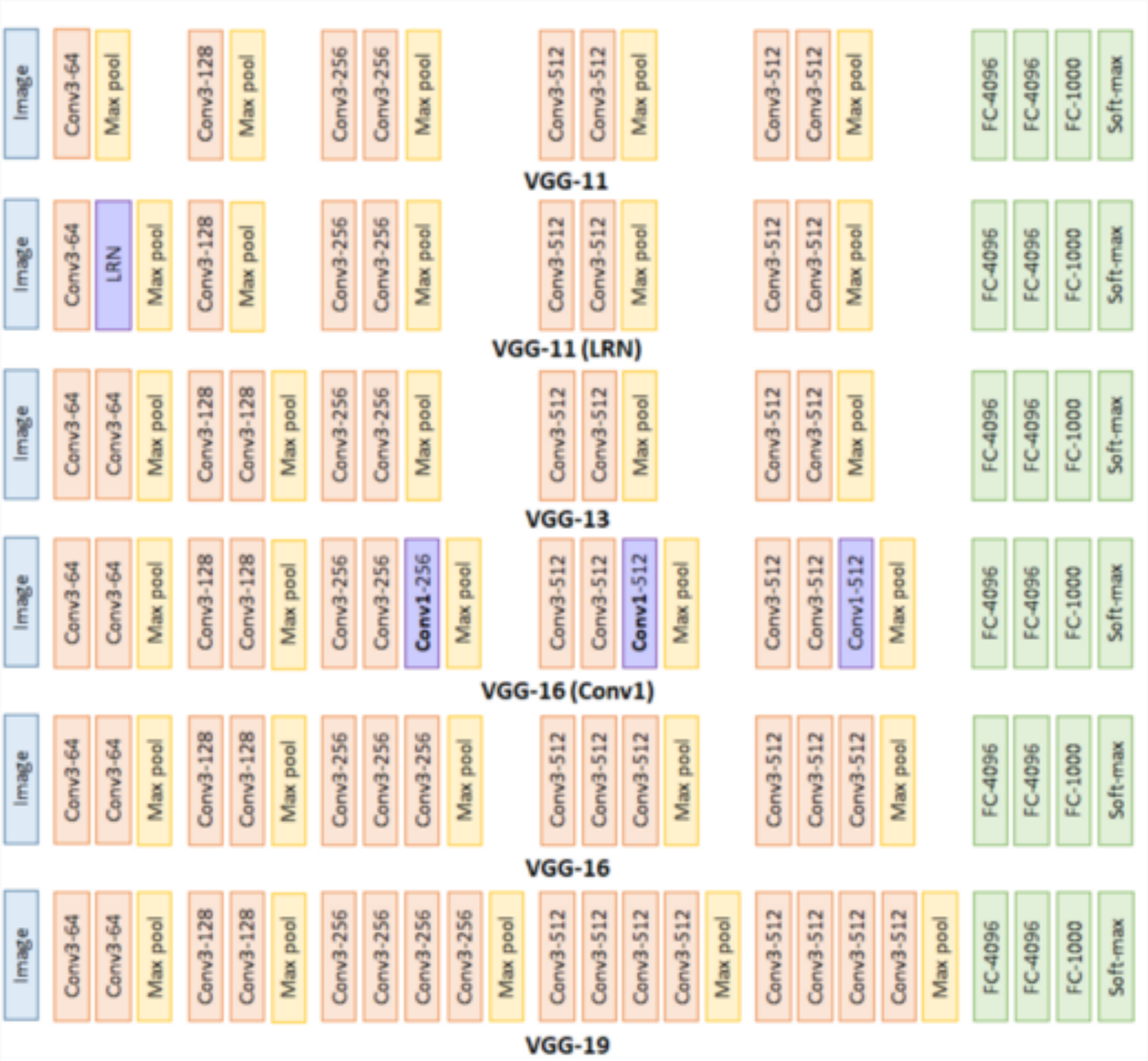
VGG


VGG
FC (1000)
FC (4096)
FC (4096)
VGG block
2 × 2 MaxPool, stride 2
3 × 3 Conv, pad 1
3 × 3 Conv, pad 1

VGG-11 Architecture



Other VGG Architectures



Network in Network (NiN)

Some issues of previous architectures:

- Fully connected layers at the end of AlexNet/VGG are huge.
- It's not possible to introduce fully connected layers earlier.

```
def nin_block(out_channels, krnl_sz,
    strides, padding):
    return nn.Sequential(
        nn.LazyConv2d(out_channels,
            krnl_sz, strides, padding),
        nn.ReLU(),
        nn.LazyConv2d(out_channels,
            krnl_sz=1),
        nn.ReLU(),
        nn.LazyConv2d(out_channels,
            krnl_sz=1),
        nn.ReLU()
    )
```

```
def nin_block(out_channels, krnl_sz,
    strides, padding):
    return nn.Sequential(
        nn.LazyConv2d(out_channels,
            krnl_sz, strides, padding),
        nn.ReLU(),
        nn.LazyConv2d(out_channels,
            krnl_sz=1),
        nn.ReLU(),
        nn.LazyConv2d(out_channels,
            krnl_sz=1),
        nn.ReLU()
    )
```

The resulting 1×1 convolution can be thought as a fully connected layer acting independently on each pixel location

```python
class NiN(d2l.Classifier):
  def __init__(self, lr=0.1, n_cls=10):
    self.net = nn.Sequential(
    nin_block(96, krnl_sz=11, strides=4,
       padding=0),
    nn.MaxPool2d(3, stride=2),
    nin_block(256, krnl_sz=5, strides=1,
       padding=2),
    nn.MaxPool2d(3, stride=2),
    nin_block(384, krnl_sz=3, strides=1,
       padding=1),
    nn.MaxPool2d(3, stride=2),
    nn.Dropout(0.5),
    nin_block(n_cls, krnl_sz=3, strides=1,
        padding=1),
    nn.AdaptiveAvgPool2d((1, 1)),
    nn.Flatten())
```

NiN

NiN discards fully connected layers altogether.

Rather, NiN uses a NiN block with a number of output channels equal to the number of label classes, followed by a global average pooling layer, yielding a vector of logits.

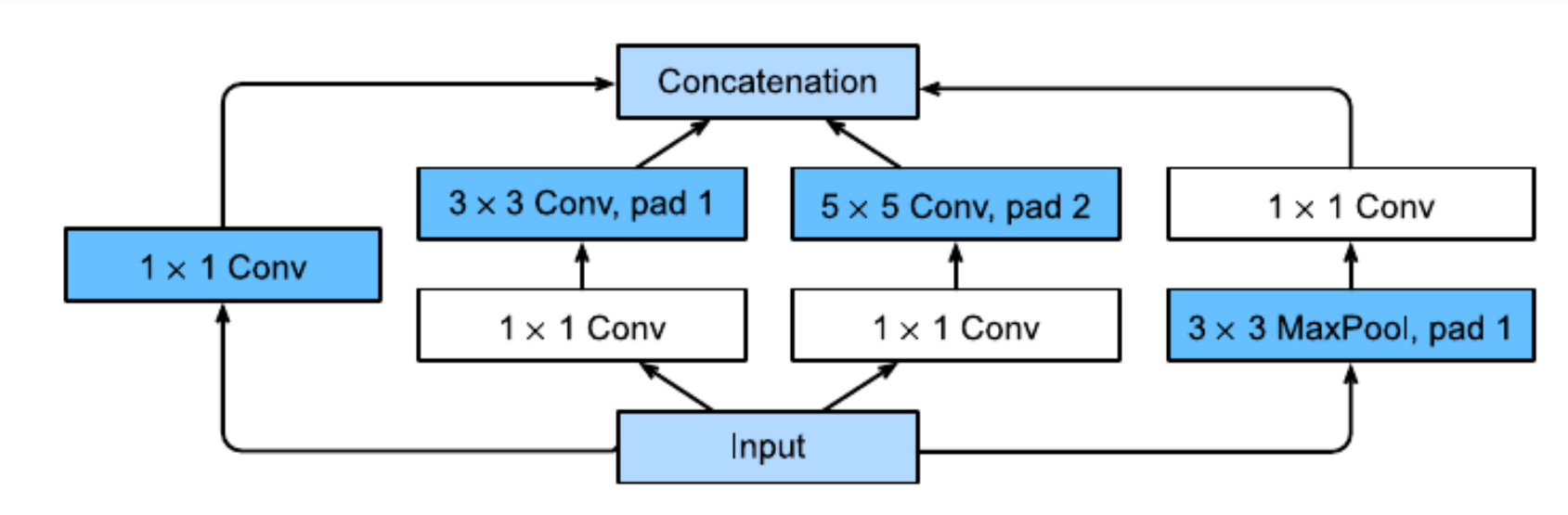
NiN

```
NiN().layer_summary((1, 1, 224, 224))
```

```
Sequential output shape:     torch.Size([1, 96, 54, 54])
MaxPool2d output shape:      torch.Size([1, 96, 26, 26])
Sequential output shape:     torch.Size([1, 256, 26, 26])
MaxPool2d output shape:      torch.Size([1, 256, 12, 12])
Sequential output shape:     torch.Size([1, 384, 12, 12])
MaxPool2d output shape:      torch.Size([1, 384, 5, 5])
Dropout output shape:        torch.Size([1, 384, 5, 5])
Sequential output shape:     torch.Size([1, 10, 5, 5])
AdaptiveAvgPool2d output shape:      torch.Size([1, 10, 1, 1])
Flatten output shape:        torch.Size([1, 10])
/home/d2l-worker/miniconda3/envs/d2l-en-release-1/lib/python3.9/site-packages/
↪torch/nn/modules/lazy.py:178: UserWarning: Lazy modules are a new feature␣
↪under heavy development so changes to the API or functionality can happen at␣
↪any moment.
  warnings.warn('Lazy modules are a new feature under heavy development '
```

Which kernel size is to be used? $1 \times 1,\ 3 \times 3,\ 5 \times 5,\ \ldots$?

Inception Block

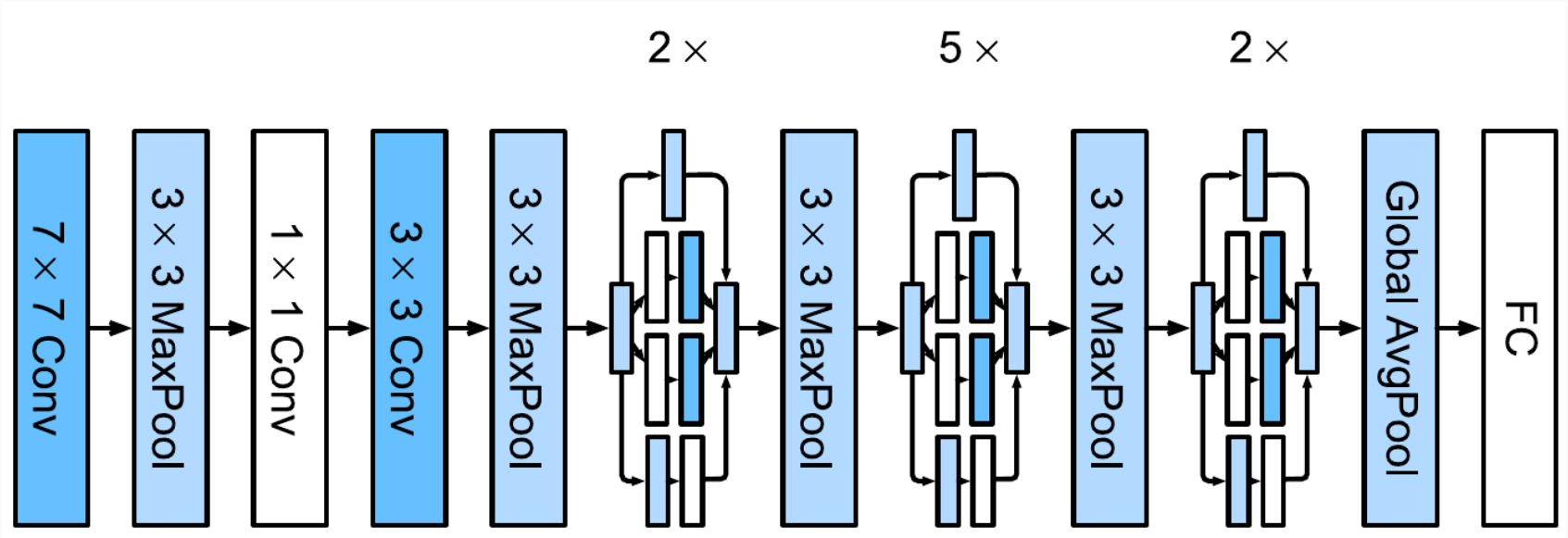


Inception Block

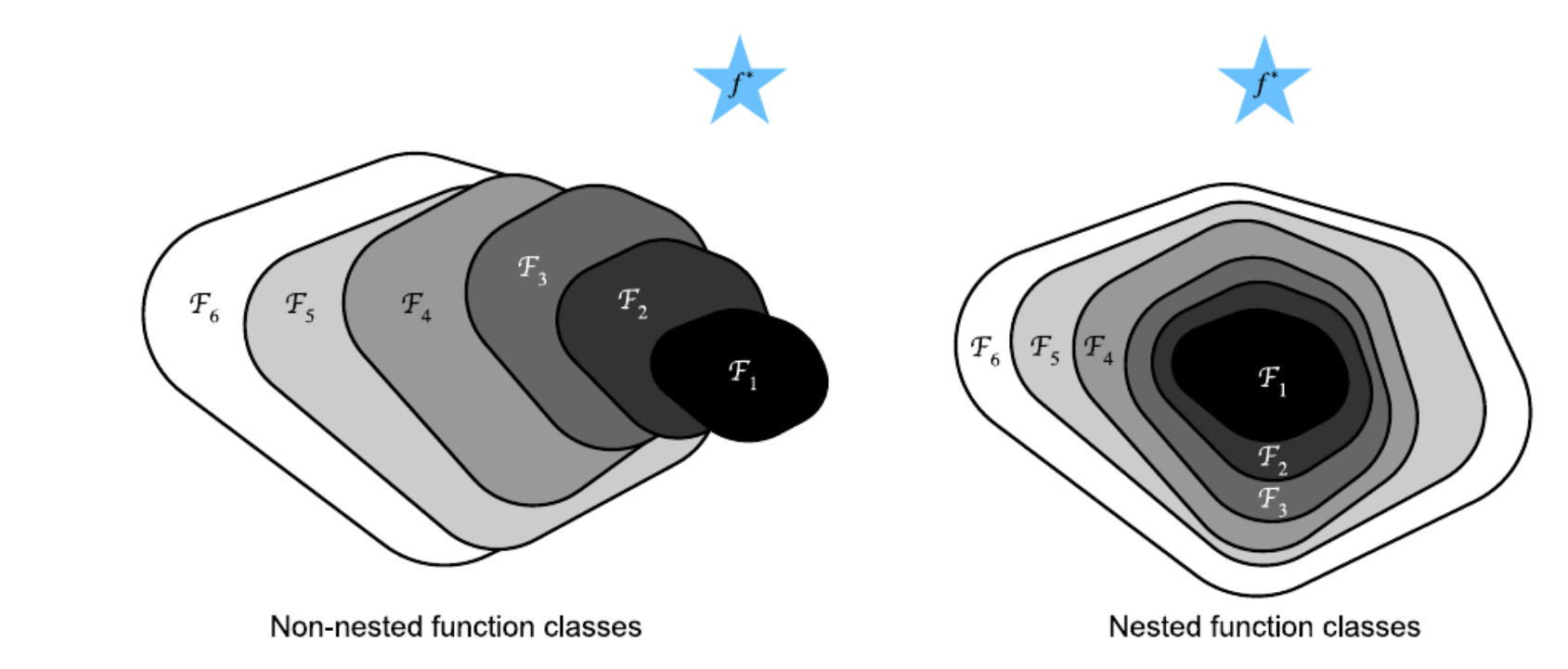
To gain some intuition for why this network works so well, consider the combination of thefilters. They explore the image in a variety of filter sizes. This means that details at different extents can be recognized efficiently by filters of different sizes.

At the same time, we can allocate different amounts of parameters for different filters.

GoogLeNet Model

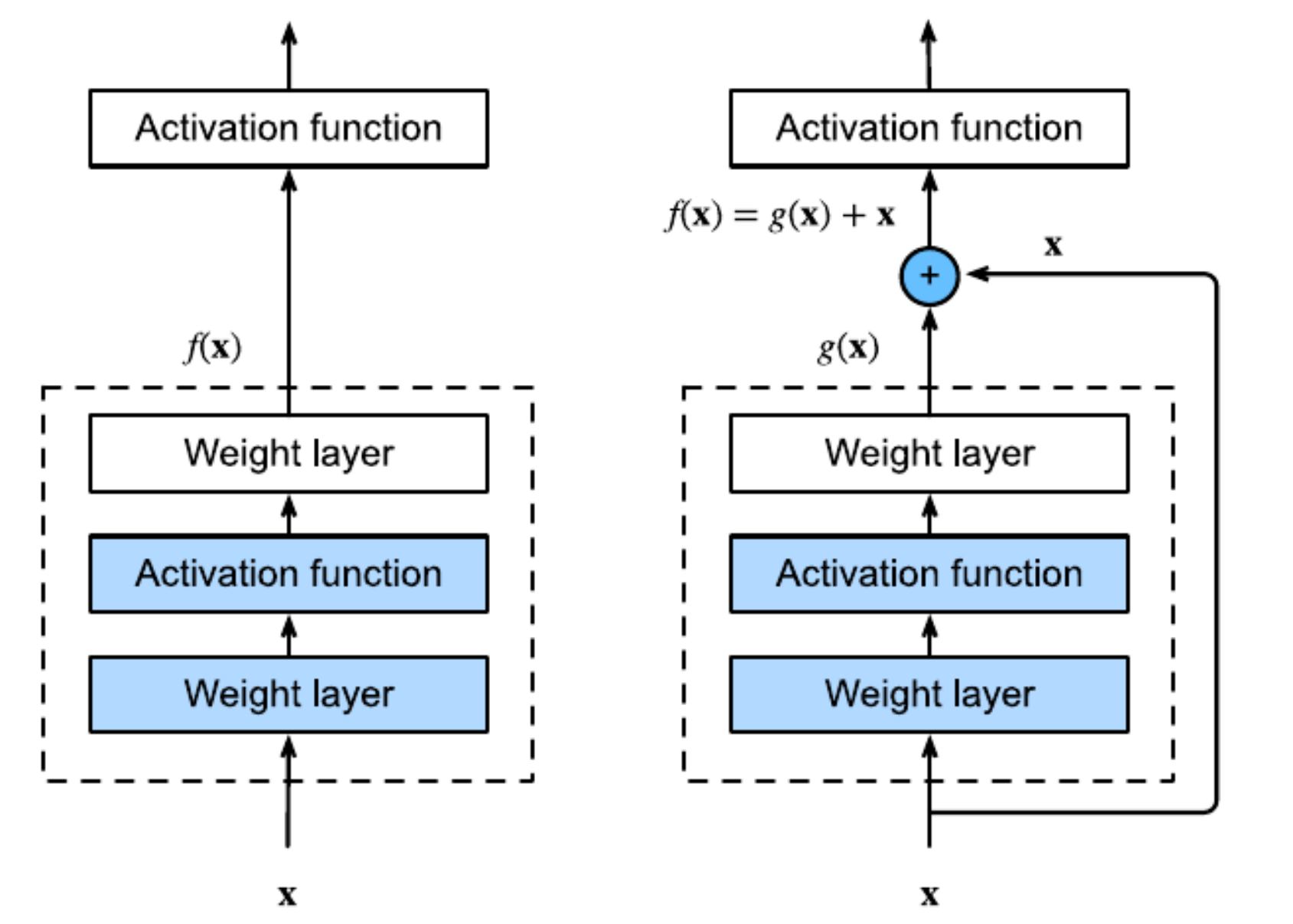


Function Classes

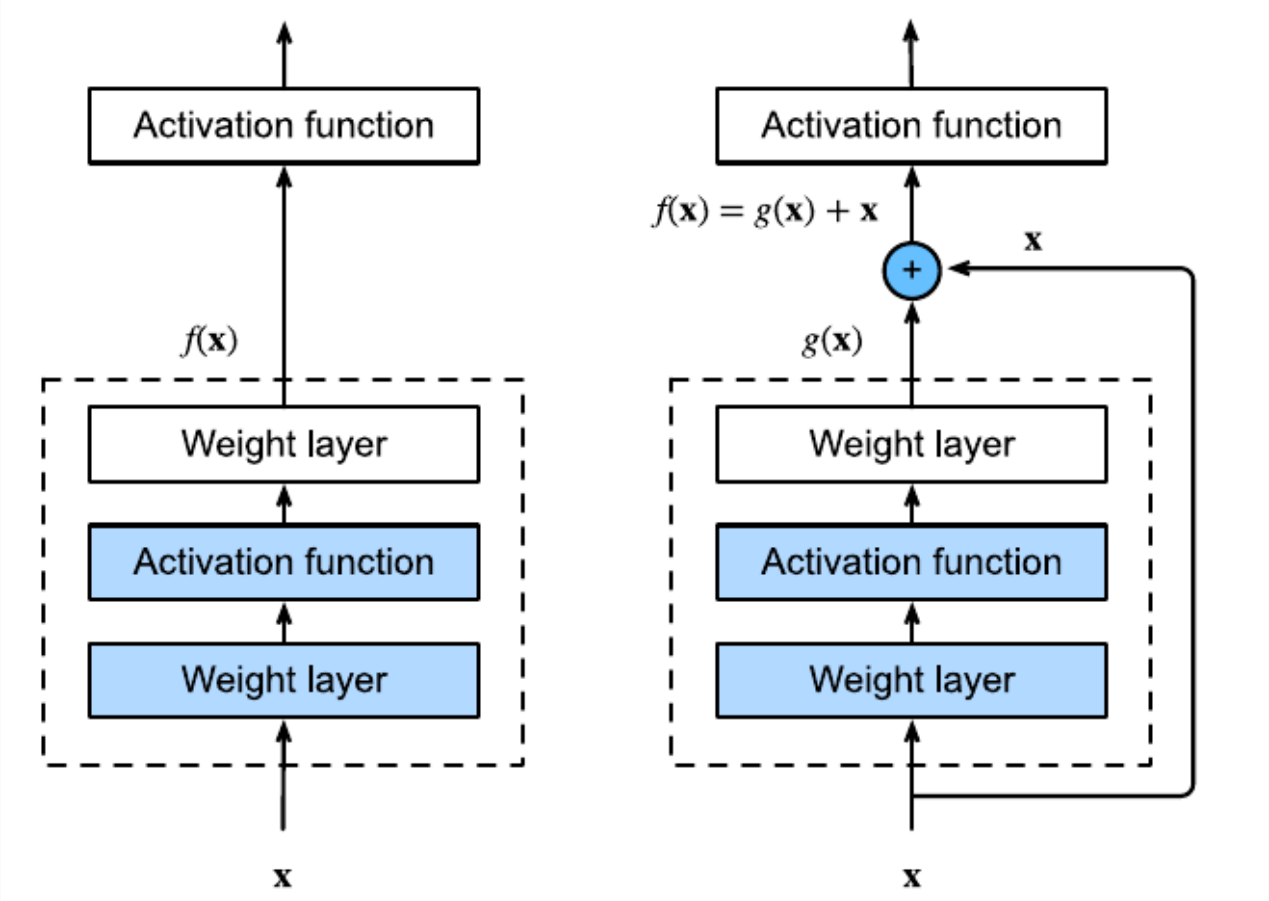


For non-nested function classes, a larger (indicated by area) function class does not guarantee to get closer to the truth function (f^*). This does not happen in nested function classes.

Residual Block

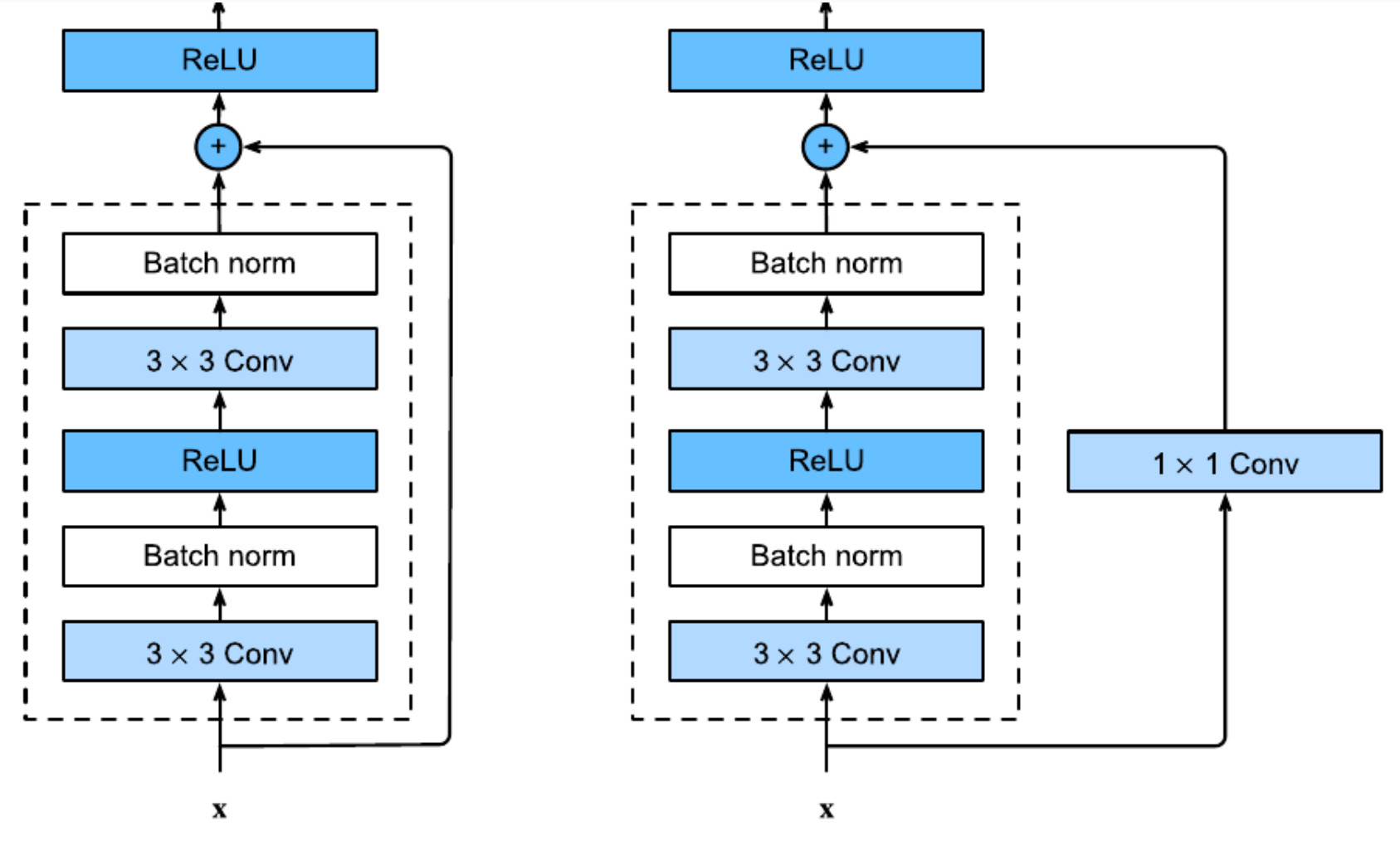


Residual Block



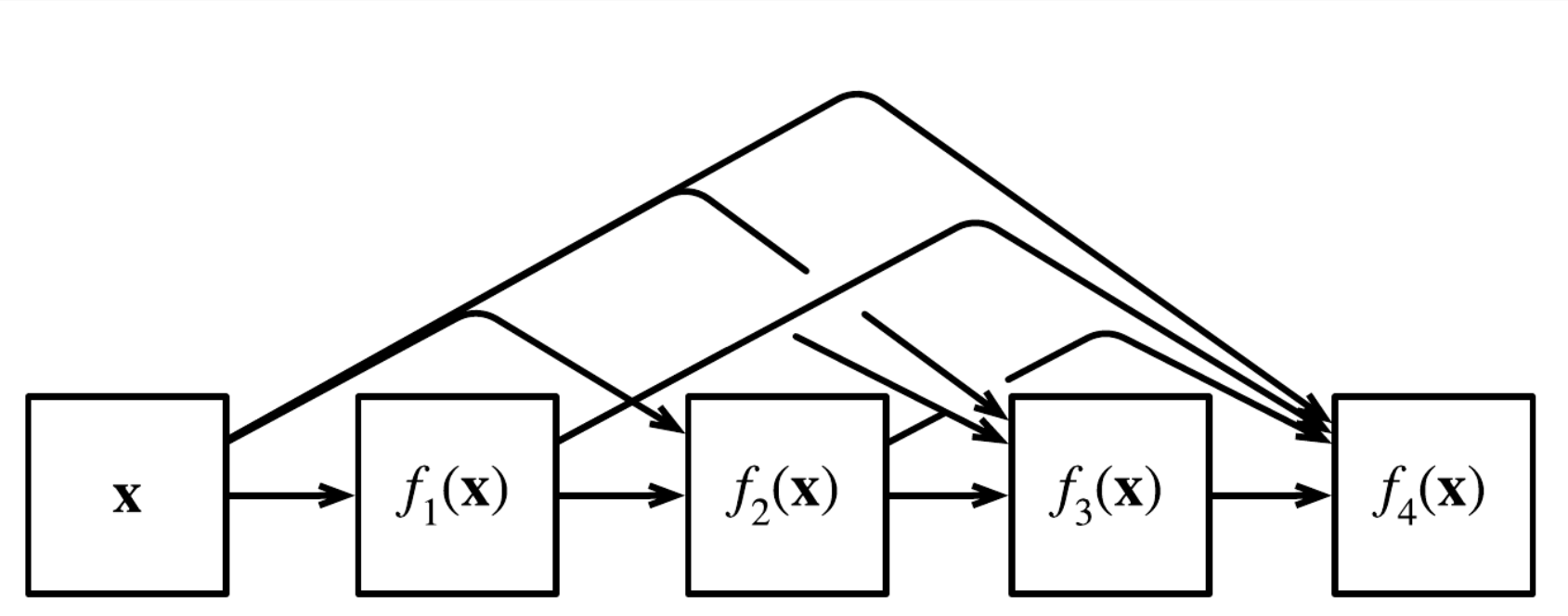
In a regular block (left), the portion within the dotted-line box must directly learn the mapping $f(x)$. In a residual block (right), the portion within the dotted-line box needs to learn the residual mapping $g(x) = f(x) - x$, making the identity mapping $f(x) = x$ easier to learn.

Residual Block



ResNet block with and without 1×1 convolution, which transforms the input into the desired shape for the addition operation.

DenseNet



Batch Normalization

Motivation I

It is common to normalize input before feeding into ML models.

Motivation I

It is common to normalize input before feeding into ML models.

- Normalize input features to have zero mean and unit variance.

Motivation I

It is common to normalize input before feeding into ML models.

- Normalize input features to have zero mean and unit variance.
- Rescale vectors to unit length, possibly zero mean per observation (e.g. spatial sensor data).

Motivation I

It is common to normalize input before feeding into ML models.

- Normalize input features to have zero mean and unit variance.
- Rescale vectors to unit length, possibly zero mean per observation (e.g. spatial sensor data).

Might a corresponding normalization step inside a deep network be beneficial?

Motivation II

The variables in intermediate layers may take values with widely varying magnitudes, which may hamper learning:

- whether along the layers from input to output
- across units in the same layer
- over time due to our updates to the model parameters.

Motivation III

Deeper networks are more complex and prone to overfitting.

$\Longrightarrow$ Regularization is crucial.

Batch Normalization

$$\mathrm{BN}(x) = \gamma \cdot \frac{x - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

Here, $\hat{\mu}_B$ is the sample mean and $\hat{\sigma}_B$ is the sample standard deviation of the minibatch B.

$$\hat{\mu}_B = \frac{1}{|B|} \sum_{x \in B} x$$

$$\hat{\sigma}_B^2 = \frac{1}{|B|} \sum_{x \in B} (x - \hat{\mu}_B)^2 + \boxed{\epsilon}$$

After applying standardization, the resulting minibatch has zero mean and unit variance.

Batch Normalization

$$\text{BN}(x) = \gamma \cdot \frac{x - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

Both *scale parameter* γ and *shift paramter* β have the same shape as the input x and are learned during model training.

Batch Normalization

$$\text{BN}(x) = \gamma \cdot \frac{x - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

Both *scale parameter* γ and *shift paramter* β have the same shape as the input x and are learned during model training.

During **training mode**, $\hat{\mu}_B$ and $\hat{\sigma}_B$ are calculated *per minibatch*.

Batch Normalization

$$\text{BN}(x) = \gamma \cdot \frac{x - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

Both *scale parameter* γ and *shift paramter* β have the same shape as the input x and are learned during model training.

During **training mode**, $\hat{\mu}_B$ and $\hat{\sigma}_B$ are calculated *per minibatch*.

During **inference mode**, $\hat{\mu}_B$ and $\hat{\sigma}_B$ are calculated *across the entire dataset*.

BN for Fully Connected Layers

$$\mathbf{h} = \phi(\text{BN}(\mathbf{W}\mathbf{x} + \mathbf{b}))$$

$$\mathbf{h} = \text{BN}(\phi(\mathbf{W}\mathbf{x} + \mathbf{b}))$$

Both are supported in literature.

Batch Normalization in Convolutional Layers

We apply the operation on a per-channel basis across all locations.

Compatible with our assumption that the specific location of a pattern within an image was not critical for the purpose of understanding.

Layer Normalization

Applied to one observation at a time.

Layer Normalization

Applied to one observation at a time.

Consequently both the offset (mean) and the scaling factor (standard deviation) are scalars.

BN vs LN: An Example from Claude

https://claude.site/artifacts/
90831008-687b-4435-af77-369b1b5a6888

BN vs LN: Another Example from Claude

https://claude.site/artifacts/
80fbf4a8-436e-42c2-b773-a201a34afe43

Things to Explore

https://paperswithcode.com/area/computer-vision

Recommended Reading

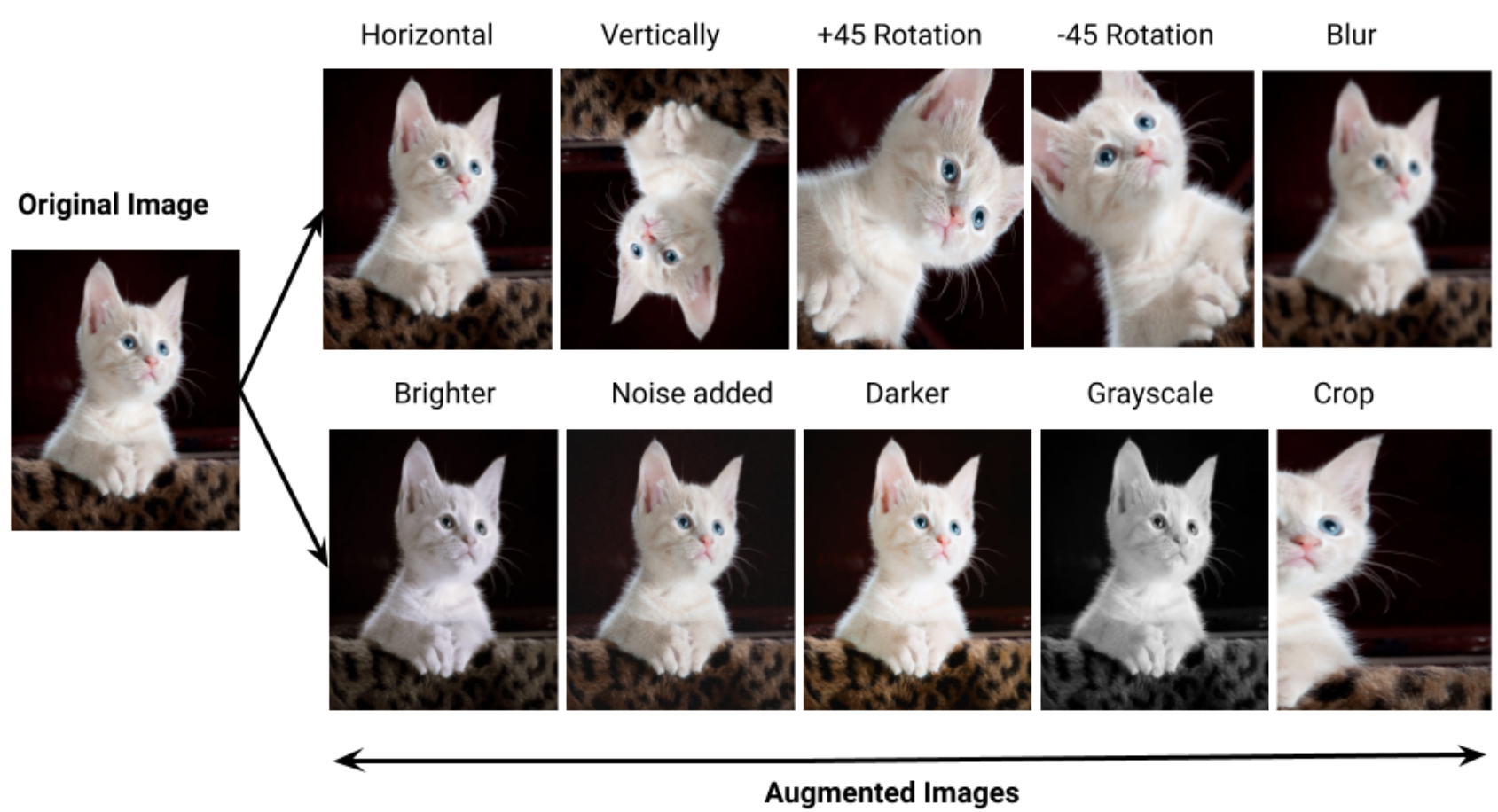
https://d2l.ai/chapter_convolutional-modern/index.html

Computer Vision Techniques

Image Augmentation

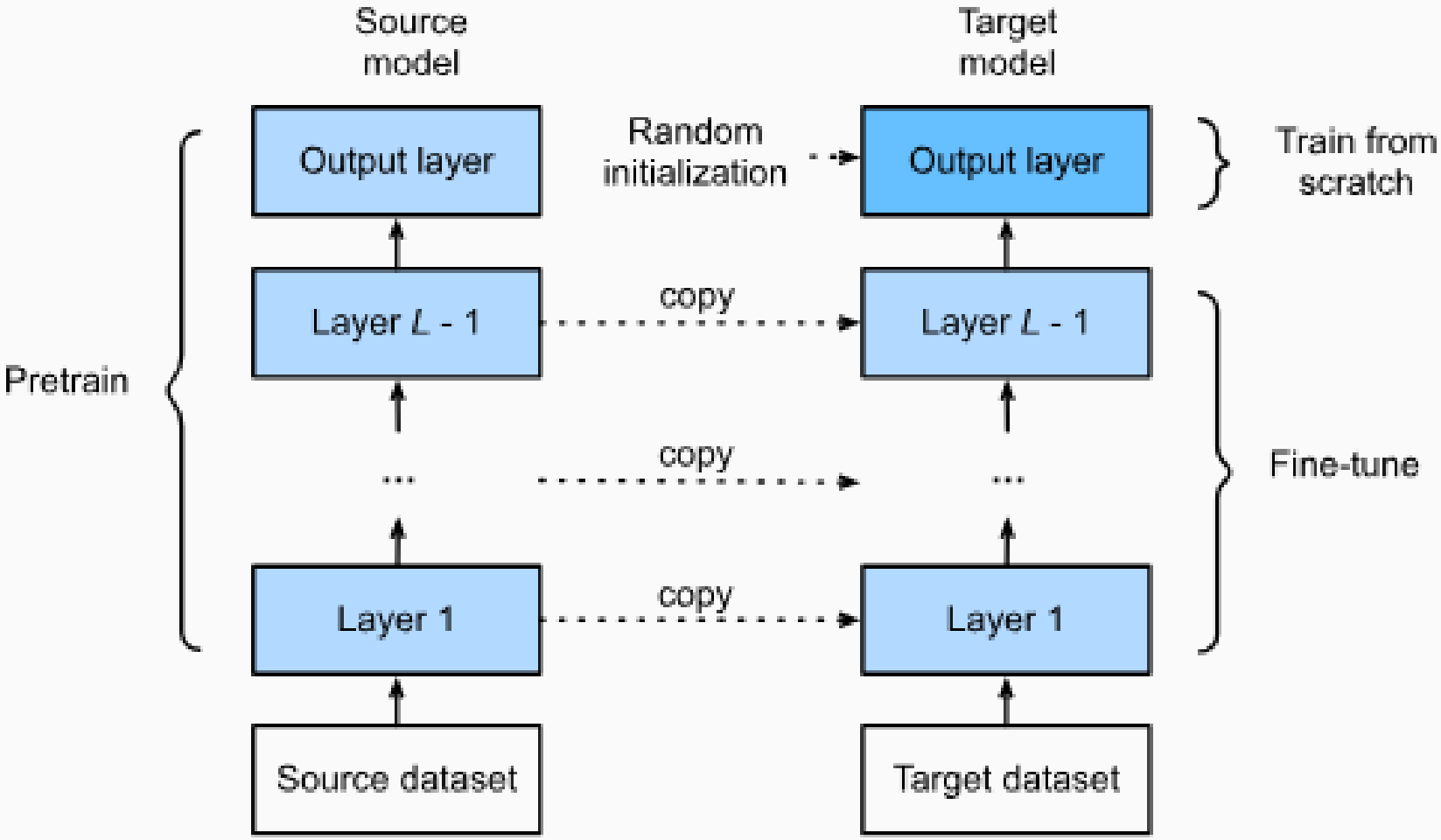
Generates *similar* but *distinct* training examples after a series of *random* changes to the training images, thereby expanding the size of the training set.

As a consequence, model is often less reliant on particular artifacts, leading to better generalization.


Horizontal
Vertically
+45 Rotation
-45 Rotation
Blur
Original Image
Brighter
Noise added
Darker
Grayscale
Crop
Augmented Images

Fine Tuning

Apply *transfer learning* to transfer the knowledge learned from the source dataset to the target dataset.

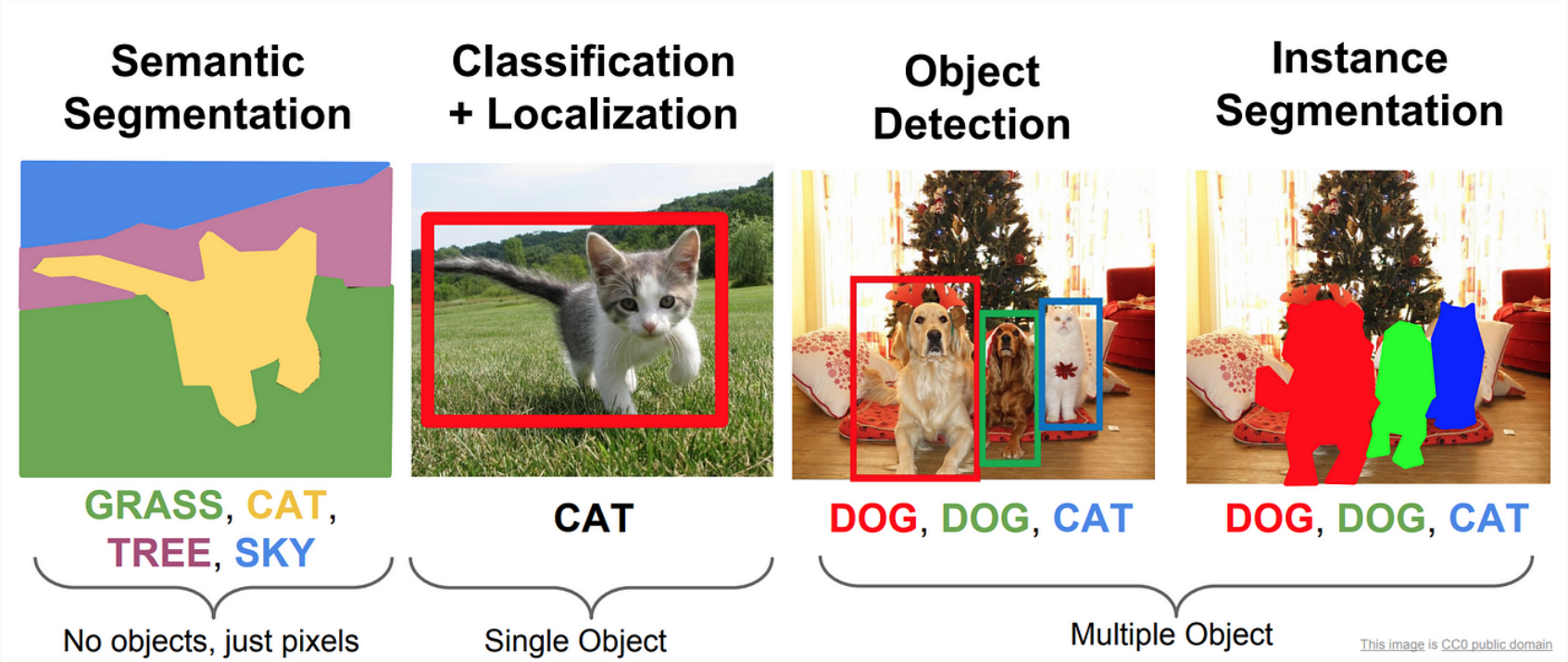

Source model
Target model
Output layer
Random initialization
Output layer
Train from scratch
Layer L - 1
copy
Layer L - 1
Pretrain
...
copy
...
Fine-tune
Layer 1
copy
Layer 1
Source dataset
Target dataset

Fine Tuning in Pytorch

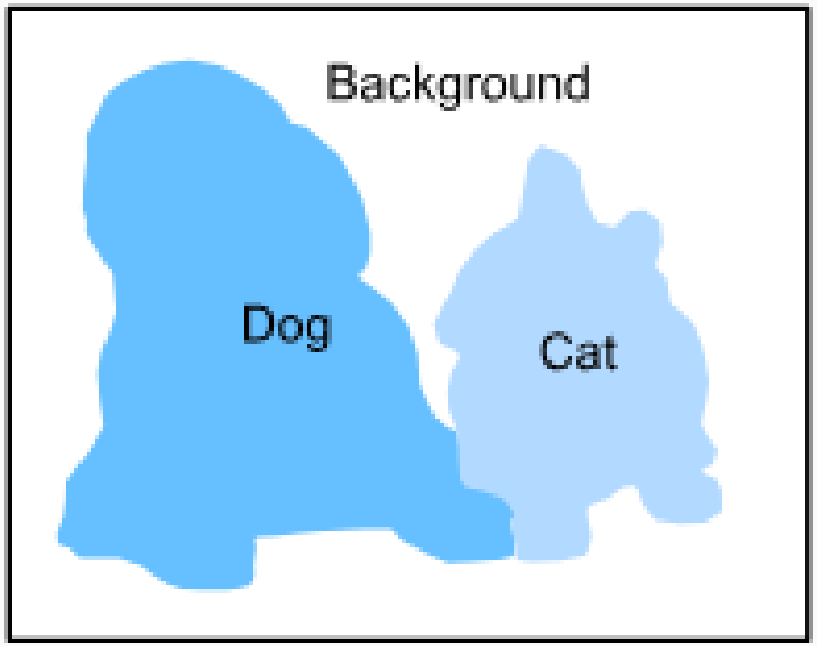
```
new_net = torchvision.models.resnet18()
new_net.fc = nn.Linear(new_net.fc.
    in_features, num_classes)
```

Collection of Pretrained Models

https://github.com/huggingface/pytorch-image-models


Semantic Segmentation
Classification + Localization
Object Detection
Instance Segmentation
GRASS, CAT, TREE, SKY
CAT
DOG, DOG, CAT
DOG, DOG, CAT
No objects, just pixels
Single Object
Multiple Object
This image is CC0 public domain

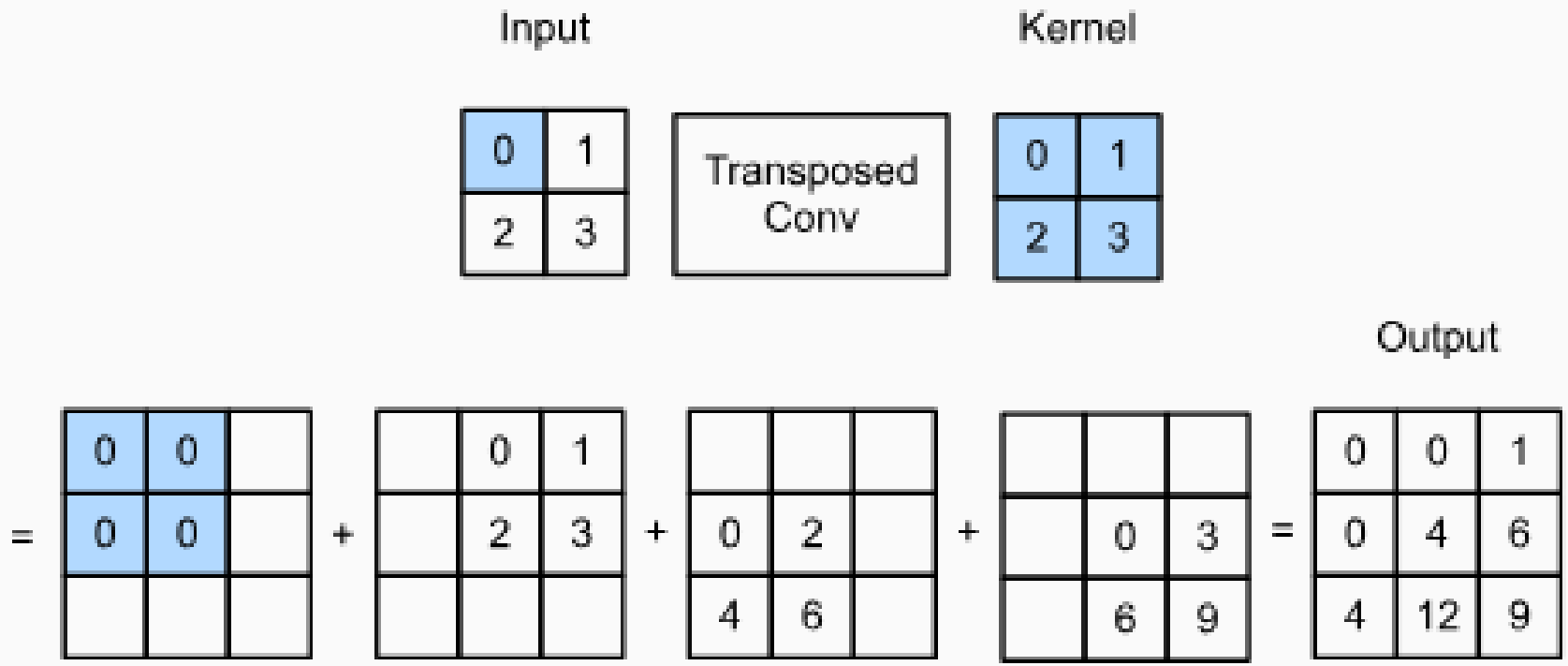
Semantic Segmentation



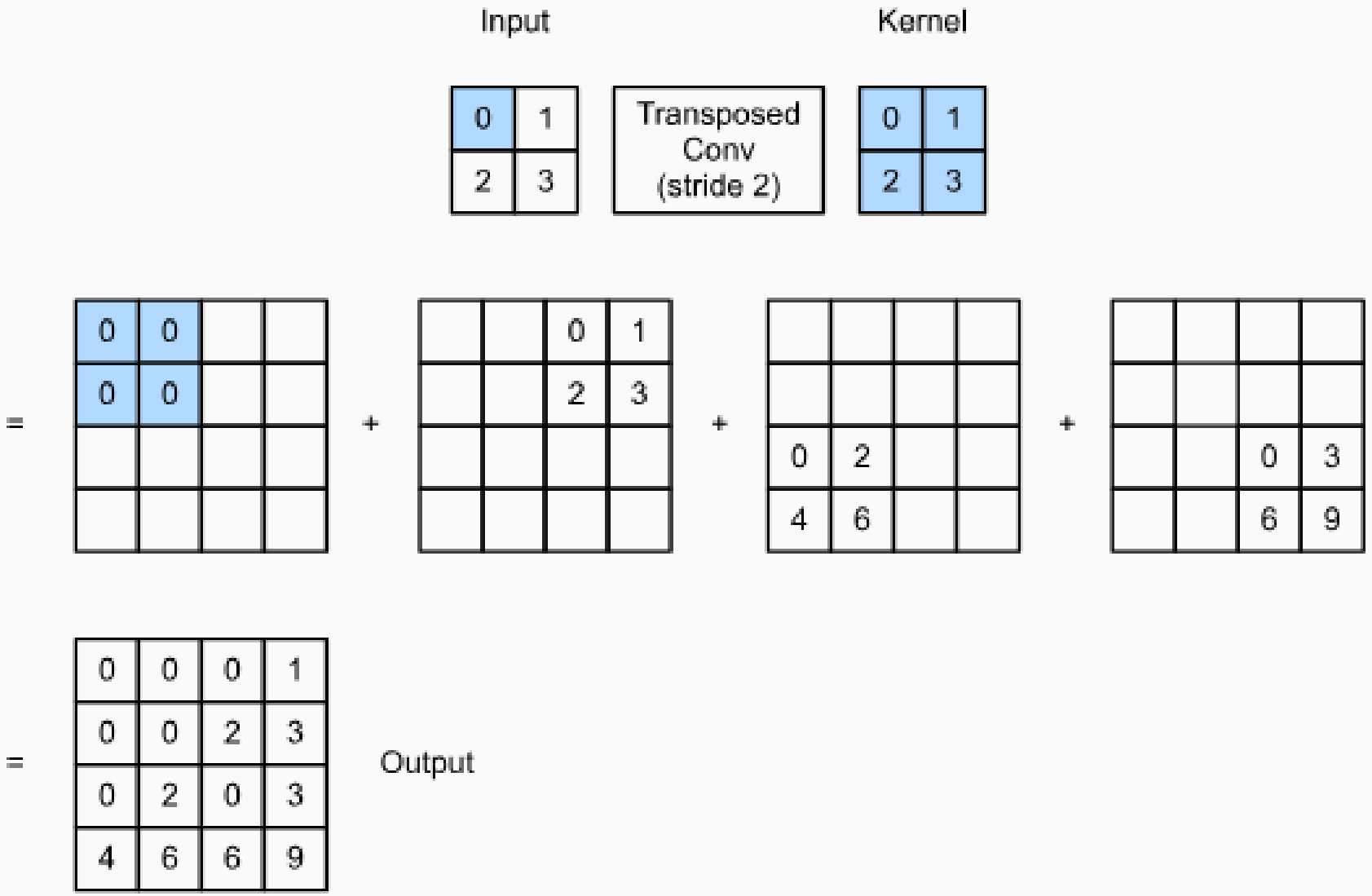
Each convolutional layer decreases resolution.

How can we get back to the original resolution?

Transposed Convolution / Upconvolution



Transposed Convolution (Stride)



Transposed Convolution (Padding)

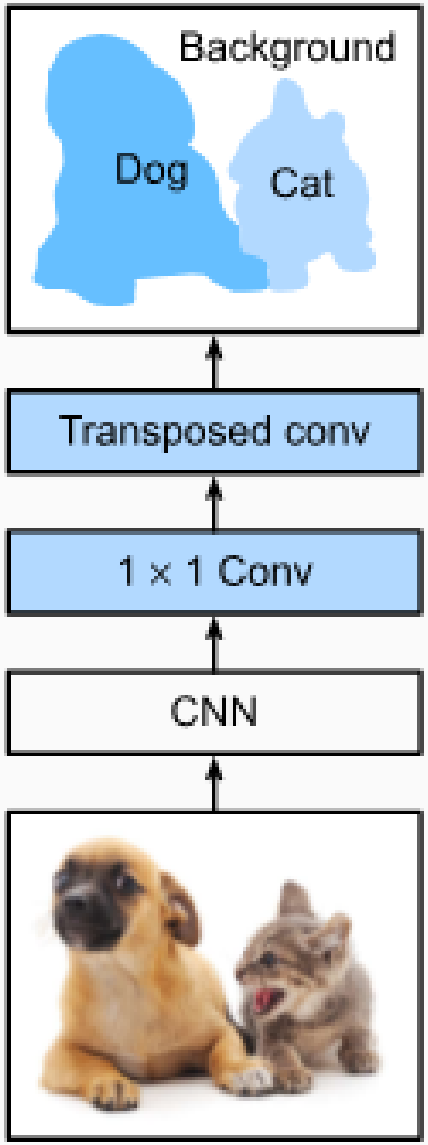
Different from in the regular convolution where padding is applied to input, it is applied to **output** in the transposed convolution. For example, when specifying the padding number on either side of the height and width as 1, the first and last rows and columns will be **removed** from the transposed convolution output.

Self Study

Why is transposed convolution called "transposed"?

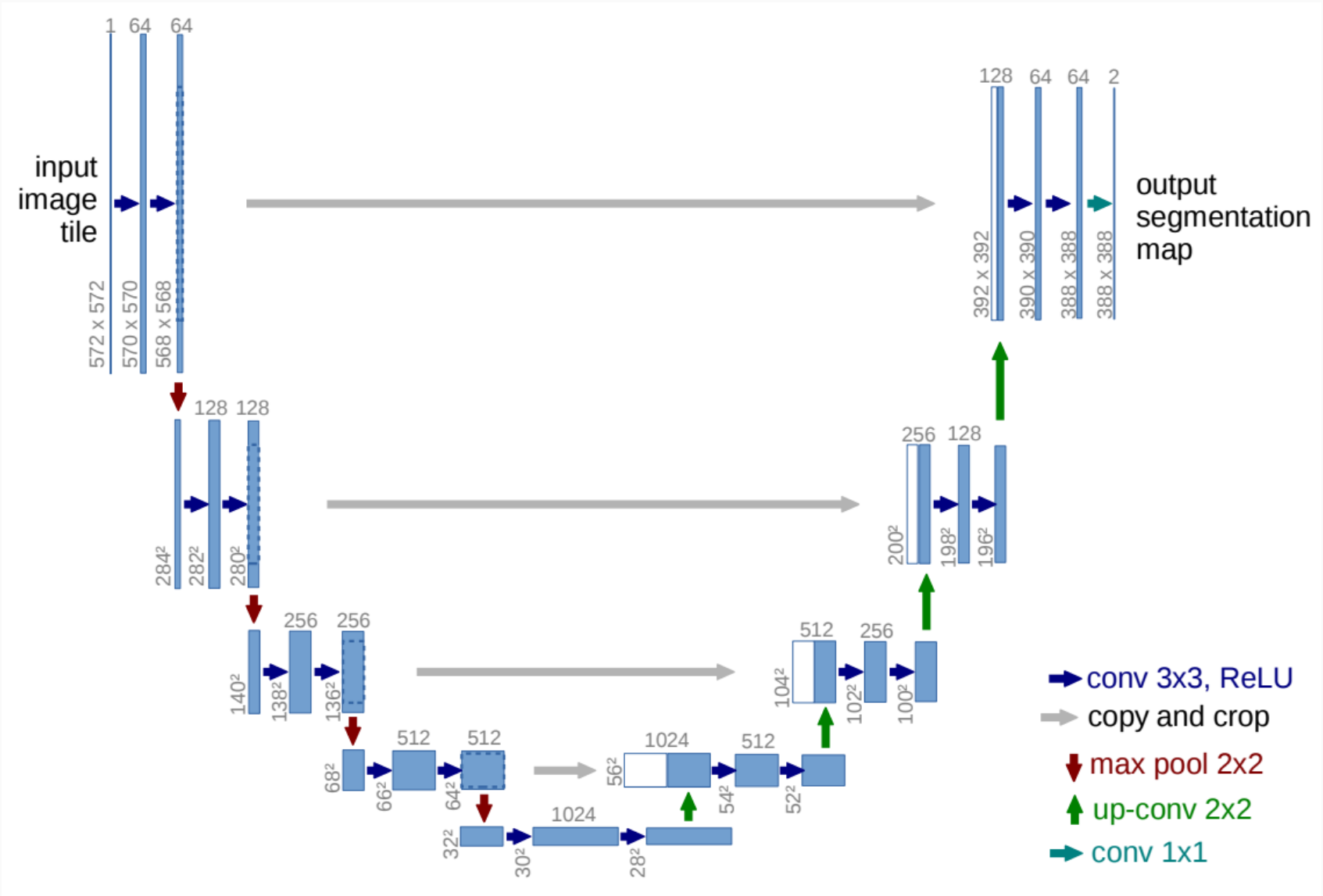
As in all, if we feed X into a convolutional layer f to output $Y = f(X)$ and create a transposed convolutional layer g with the same hyperparameters as f except for the number of output channels being the number of channels in X, then $g(Y)$ will have the same shape as X.

Fully Convolutional Network (FCN)



Here we can used pretrained CNN models to facilitate feature extraction.

UNet: Biomedical Image Segmentation



Variations of UNet

https://github.com/NITR098/Awesome-U-Net

Recommended Reading

https://d2l.ai/chapter_computer-vision/index.html