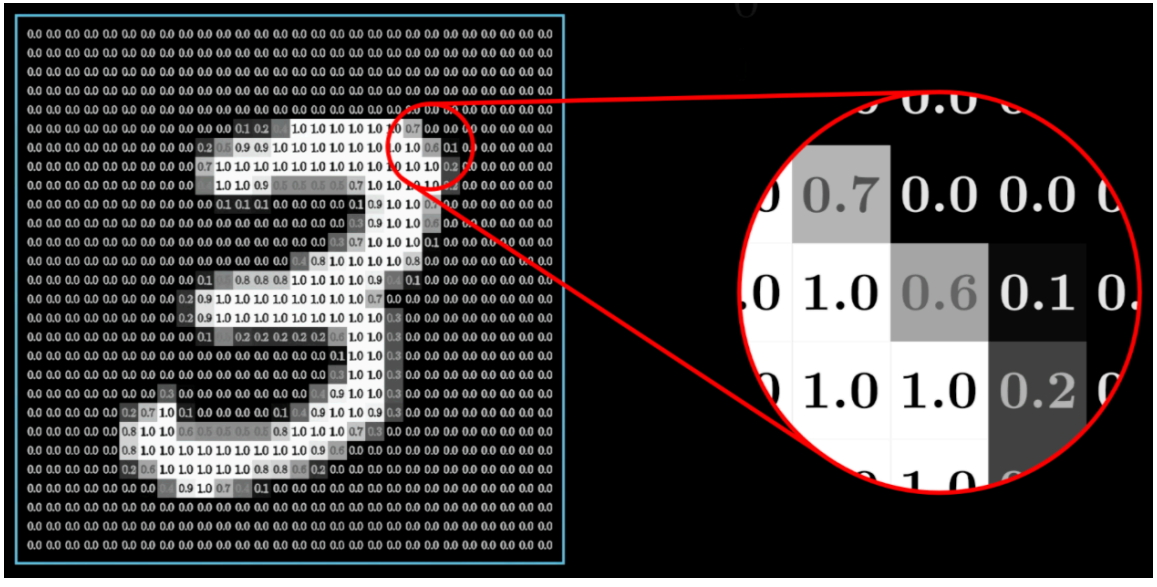
Convolutional Neural Networks

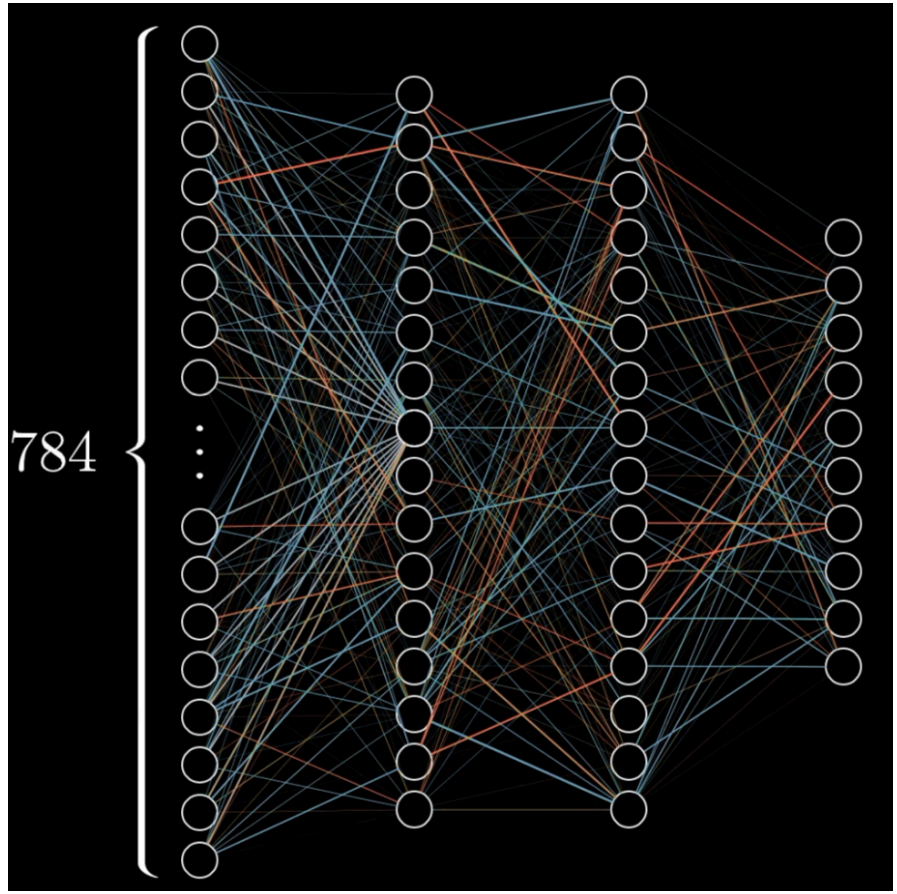
Key ideas from yesterday

- data: features and labels
- models: one and multi-layer feedforward networks
- loss function: cross entropy
- training: minimize loss by stochastic gradient descent
- validation: compute accuracy on test data
- dataset and dataloader: provide a standard interface to access data

Data

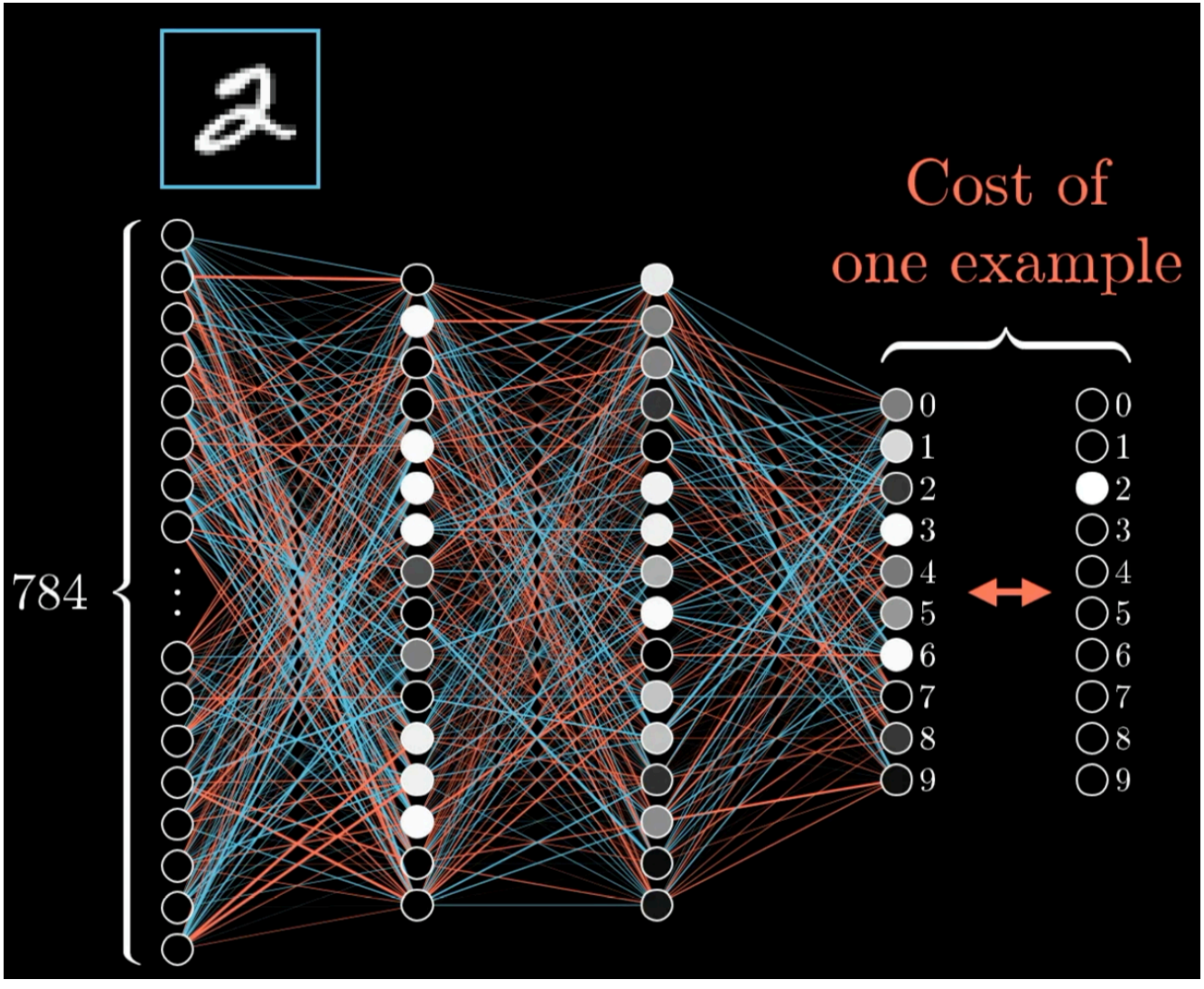


Models



$$\sigma(w_0 a_0 + w_1 a_1 + \cdots + w_{n-1} a_{n-1} + b)$$

Loss and training



Training loop

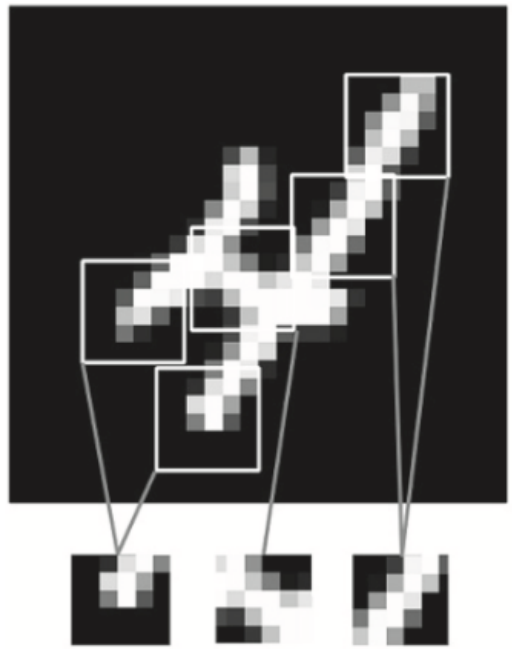
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

for epoch in range(2):  # loop over the dataset multiple times
    running_loss = 0.0
    for i, data in enumerate(train_loader):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        # zero the parameter gradients
        optimizer.zero_grad()
        # forward + backward + optimization step
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        # print statistics
        running_loss += loss.item()
        if i % 1000 == 999:    # print every 1000 mini-batches
            print(f'[{epoch+1}, {i+1}] loss: {running_loss / 1000:.3f}')
            running_loss = 0.0

print('Finished Training')
```

Neural networks for images

- in FNNs, the activations of a layer are $a^{(l+1)} = \sigma(Wa^{(l)} + b)$
- the jth element of the hidden layer has value $a_j^{(l+1)} = \varphi(w_j^T a^{(l)} + b_j)$
- inner product compares input x (or a) to a learned template w_j
- if match is good (large positive inner product), activation is large, indicating jth pattern is present in input
- for images of $H \times W \times C$ ($x \in \mathbb{R}^{HWC}$), this approach will not be spatially invariant
  - weights are not shared across locations
  - and there will be a very large number of them
- we fix this by replacing the multiplications Wx by convolutions $W * x$
  - giving us convolutional neural networks (CNNs)
  - essentially divides images into small 2d patches and compares each one with a small set of weights, or *filters*

Image templates

- identify features occurring in the correct relative locations
- we're generating a feature map through convolution (template matching)



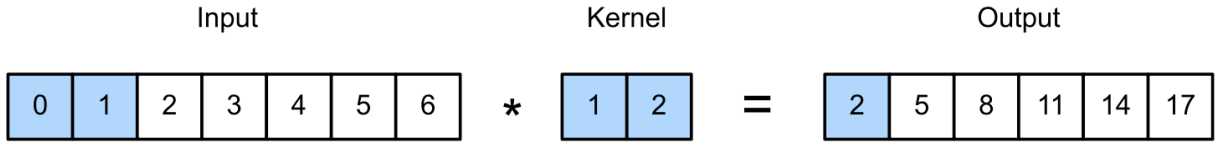
Convolution in 1d

- standard (mathematical) convolution $w * x$

-	-	1	2	3	4	-	-	
7	6	5	-	-	-	-	-	$z_0 = x_0 w_0 = 5$
-	7	6	5	-	-	-	-	$z_1 = x_0 w_1 + x_1 w_0 = 16$
-	-	7	6	5	-	-	-	$z_2 = x_0 w_2 + x_1 w_1 + x_2 w_0 = 34$
-	-	-	7	6	5	-	-	$z_3 = x_1 w_2 + x_2 w_1 + x_3 w_0 = 52$
-	-	-	-	7	6	5	-	$z_4 = x_2 w_2 + x_3 w_1 = 45$
-	-	-	-	-	7	6	5	$z_5 = x_3 w_2 = 28$

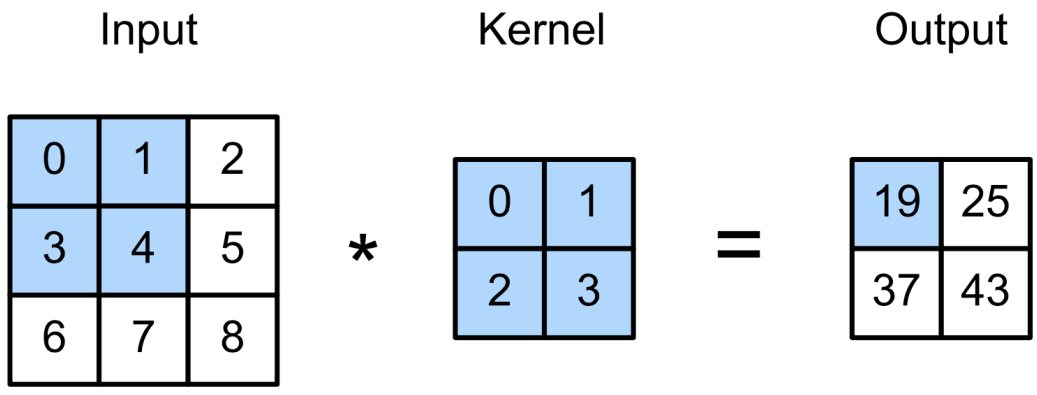
- cross-correlation (also called convolution in ML)

$$(w * x)[i] = \sum_{u=0}^{L-1} w_u x_{i+u}$$

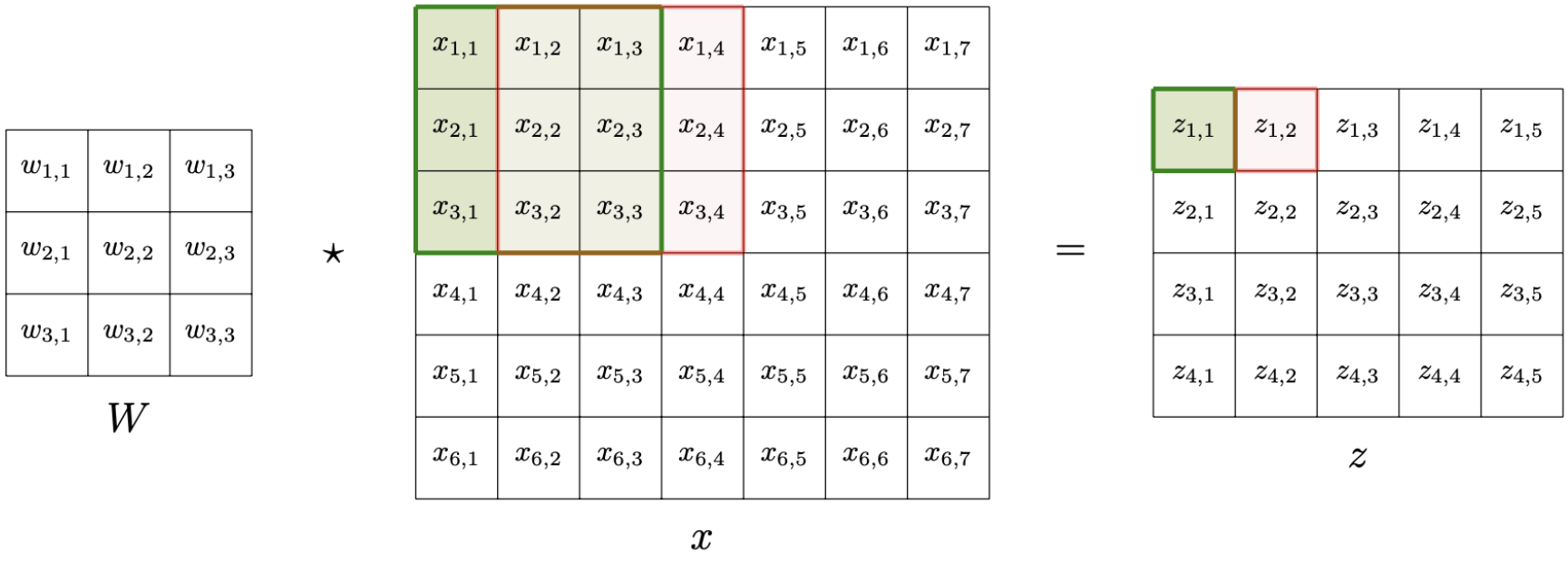


Convolution in 2d

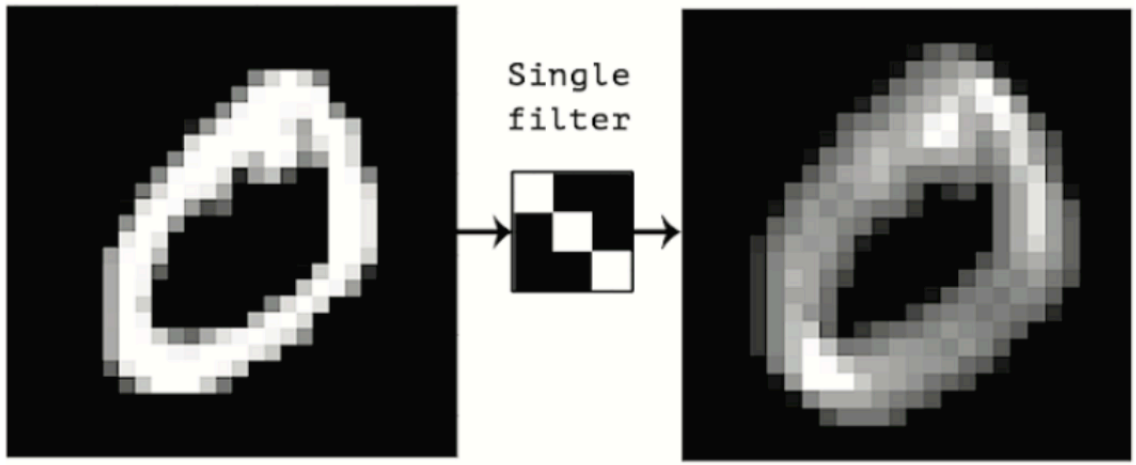
$$(W * X)[i,j] = \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} w_{u,v}\, x_{i+u,j+v}$$



Basic convolution

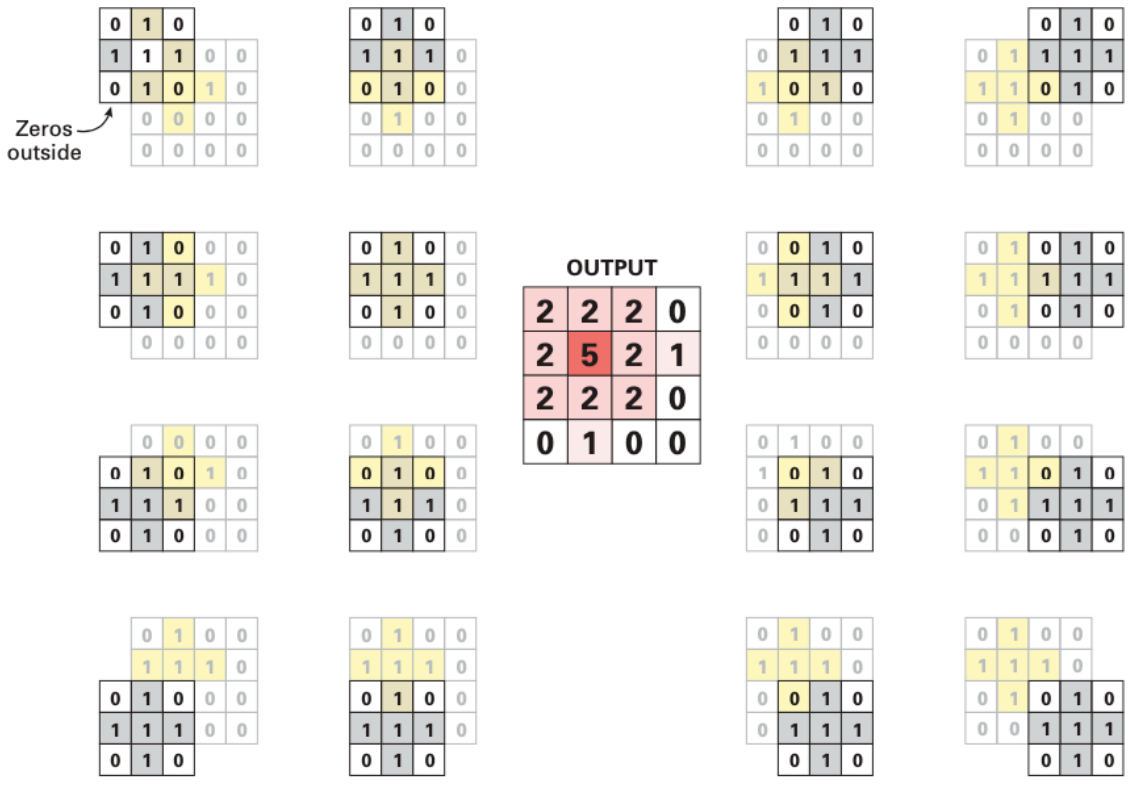


Convolving a 2d image with a 3×3 filter

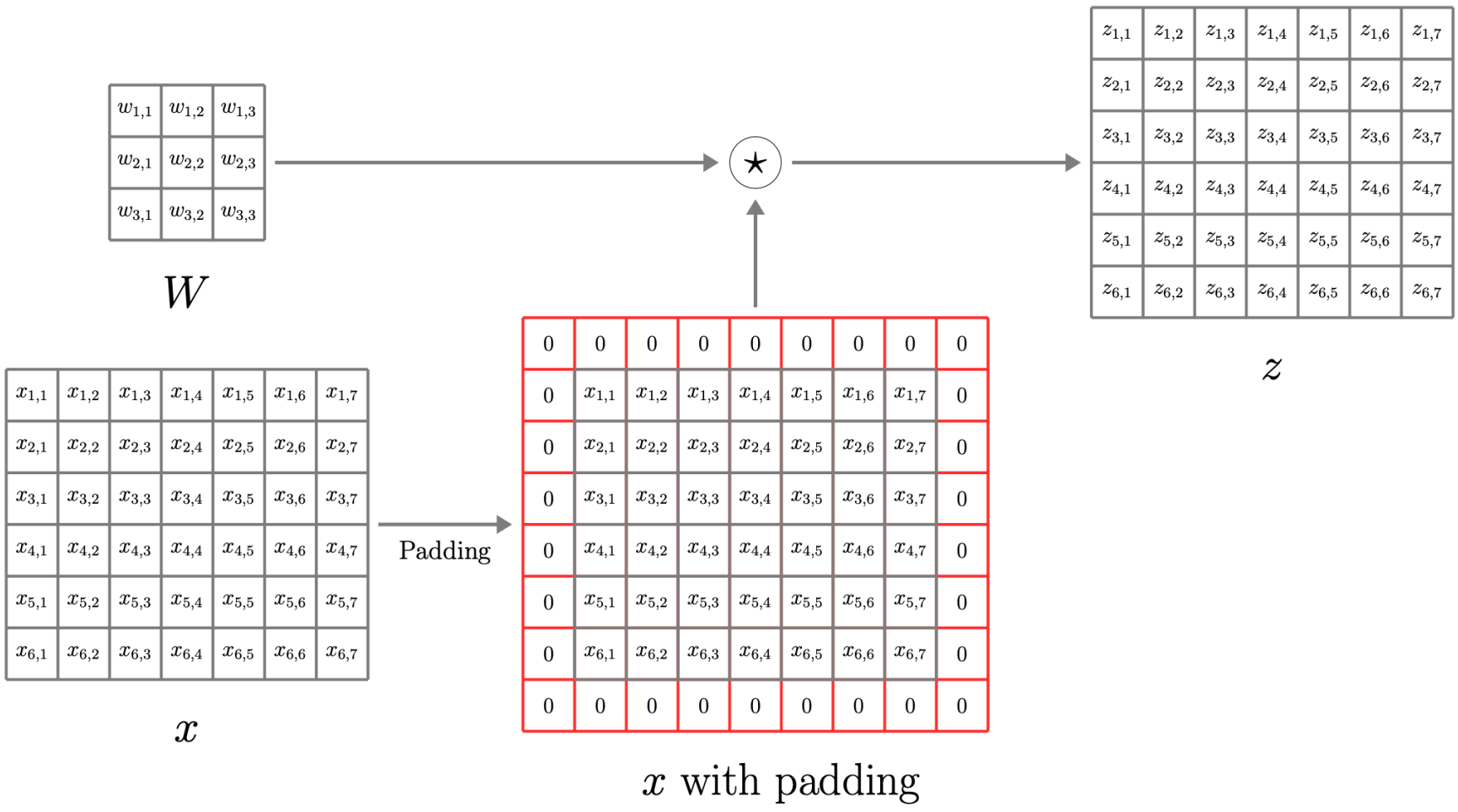


- notice that bright spots in response map correspond to locations in image which contain diagonal lines sloping down and to the right

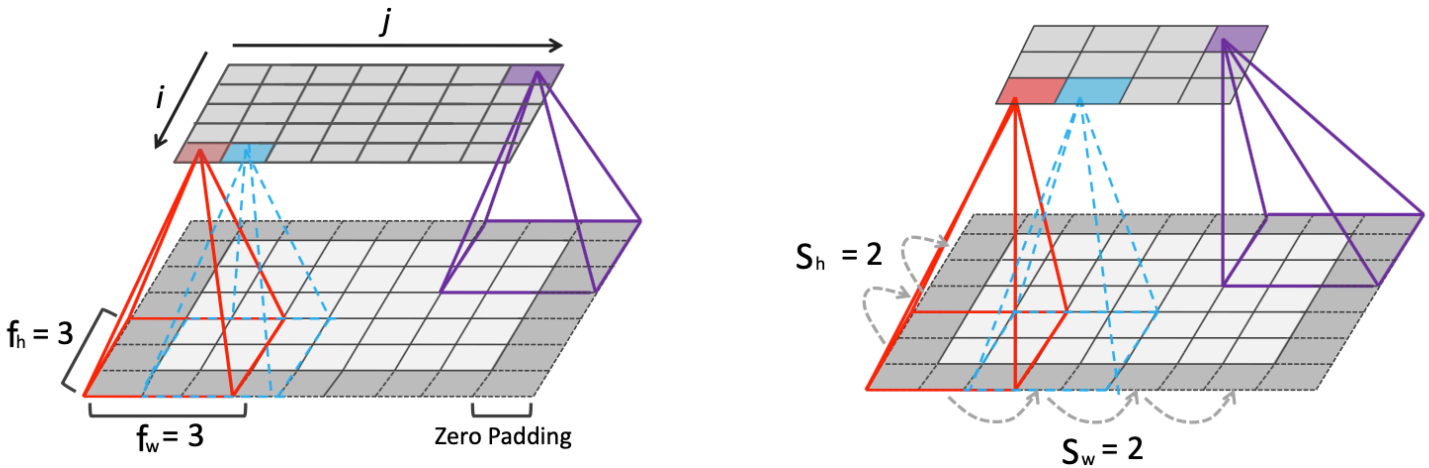
Padding



Padding


$w_{1,1}$ $w_{1,2}$ $w_{1,3}$
$w_{2,1}$ $w_{2,2}$ $w_{2,3}$
$w_{3,1}$ $w_{3,2}$ $w_{3,3}$
W
$\star$
$z_{1,1}$ $z_{1,2}$ $z_{1,3}$ $z_{1,4}$ $z_{1,5}$ $z_{1,6}$ $z_{1,7}$
$z_{2,1}$ $z_{2,2}$ $z_{2,3}$ $z_{2,4}$ $z_{2,5}$ $z_{2,6}$ $z_{2,7}$
$z_{3,1}$ $z_{3,2}$ $z_{3,3}$ $z_{3,4}$ $z_{3,5}$ $z_{3,6}$ $z_{3,7}$
$z_{4,1}$ $z_{4,2}$ $z_{4,3}$ $z_{4,4}$ $z_{4,5}$ $z_{4,6}$ $z_{4,7}$
$z_{5,1}$ $z_{5,2}$ $z_{5,3}$ $z_{5,4}$ $z_{5,5}$ $z_{5,6}$ $z_{5,7}$
$z_{6,1}$ $z_{6,2}$ $z_{6,3}$ $z_{6,4}$ $z_{6,5}$ $z_{6,6}$ $z_{6,7}$
z
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$ $x_{1,7}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$ $x_{2,7}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$ $x_{3,7}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$ $x_{4,7}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$ $x_{5,7}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$ $x_{6,7}$
x
Padding
0 0 0 0 0 0 0 0 0
0 $x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$ $x_{1,7}$ 0
0 $x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$ $x_{2,7}$ 0
0 $x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$ $x_{3,7}$ 0
0 $x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$ $x_{4,7}$ 0
0 $x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$ $x_{5,7}$ 0
0 $x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$ $x_{6,7}$ 0
0 0 0 0 0 0 0 0 0
x with padding

Strided convolution



- with a filter of size $f_h \times f_w$, an image of size $x_h \times x_w$, and padding of size p_h and p_w on each side, the output size is

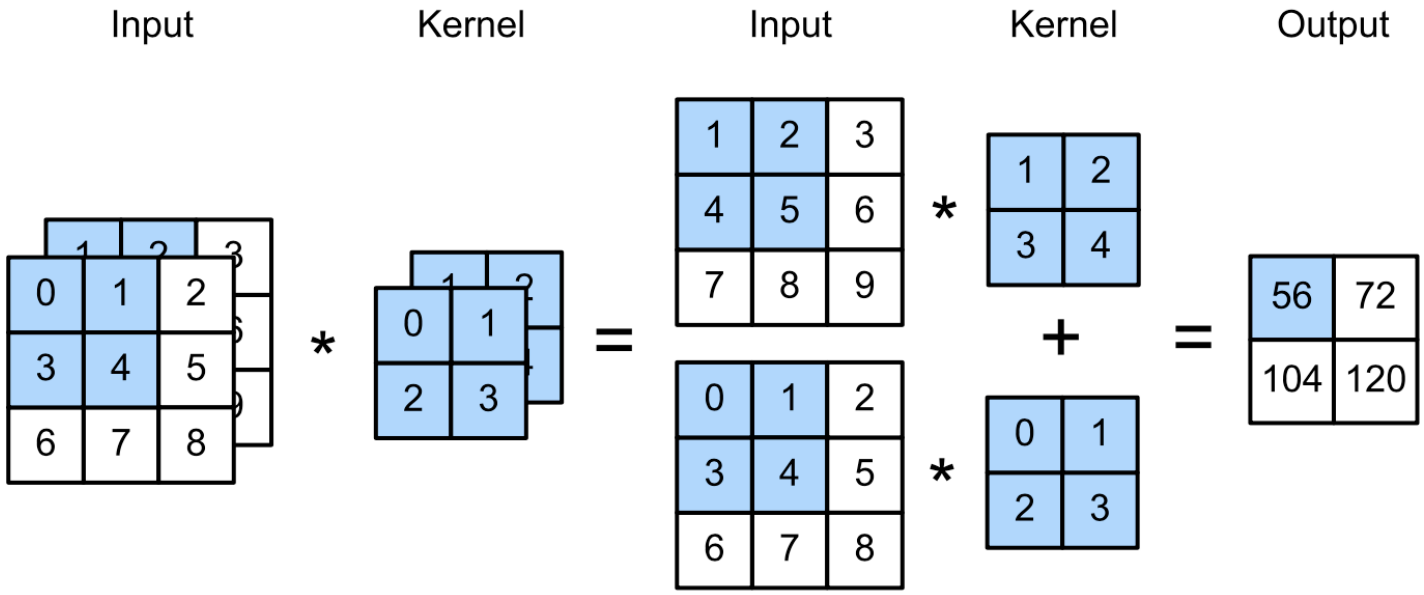
$$(x_h + 2p_h - f_h + 1) \times (x_w + 2p_w - f_w + 1)$$

- if we perform the convolution every s pixels, output size is

$$\left(\left\lfloor \frac{x_h + 2p_h - f_h}{s_h} \right\rfloor + 1 \right) \times \left(\left\lfloor \frac{x_w + 2p_w - f_w}{s_w} \right\rfloor + 1 \right)$$

- called strided convolution

Multiple input channels

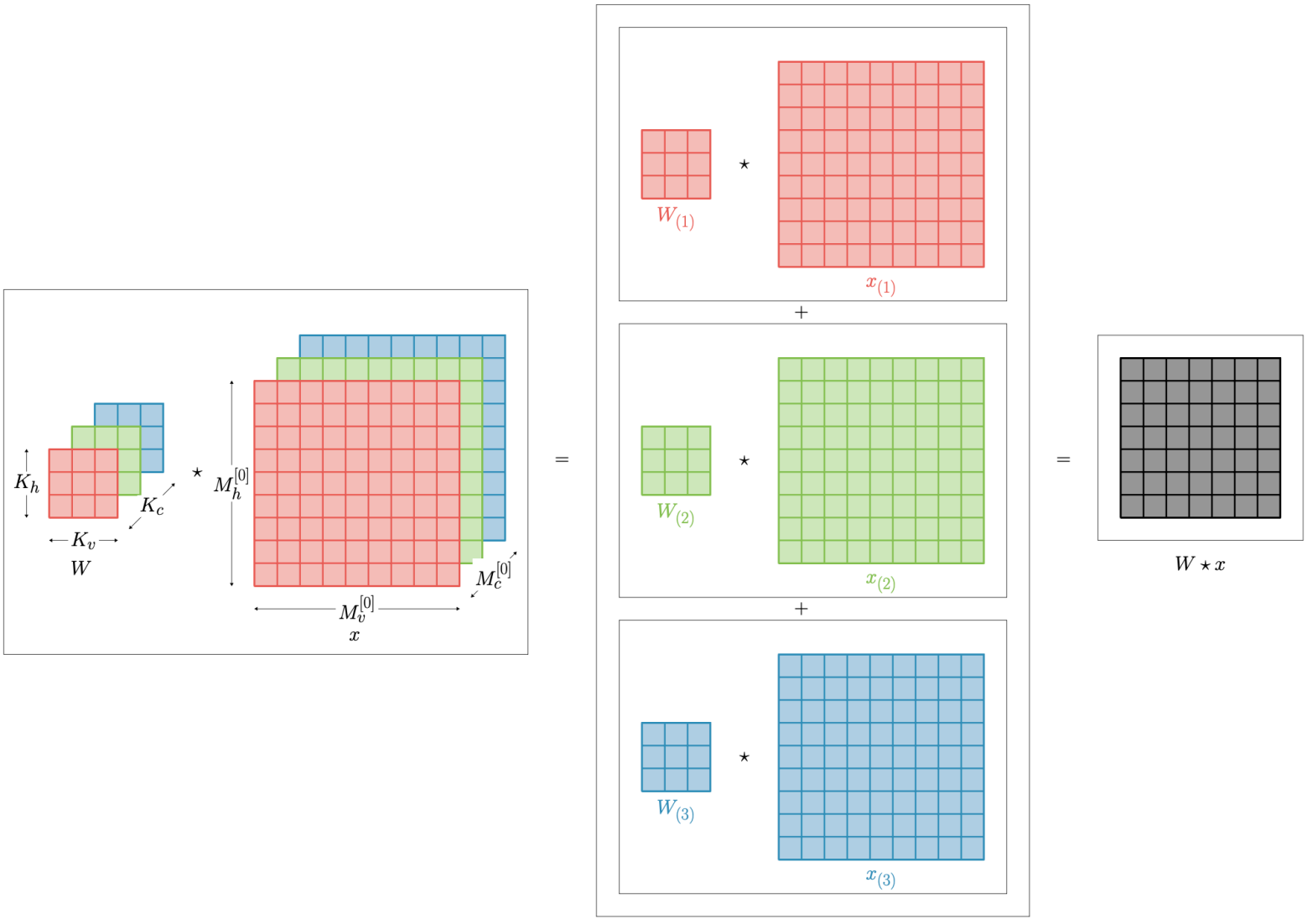


- images often have multiple channels
- in this case W is a 3d weight matrix (tensor), $W \times W \times C$

$$z_{i,j} = \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} \sum_{c=1}^{C-1} w_{u,v,c} x_{si+u,sj+v,c} + b$$

- note: one bias term per channel

Multiple input channels

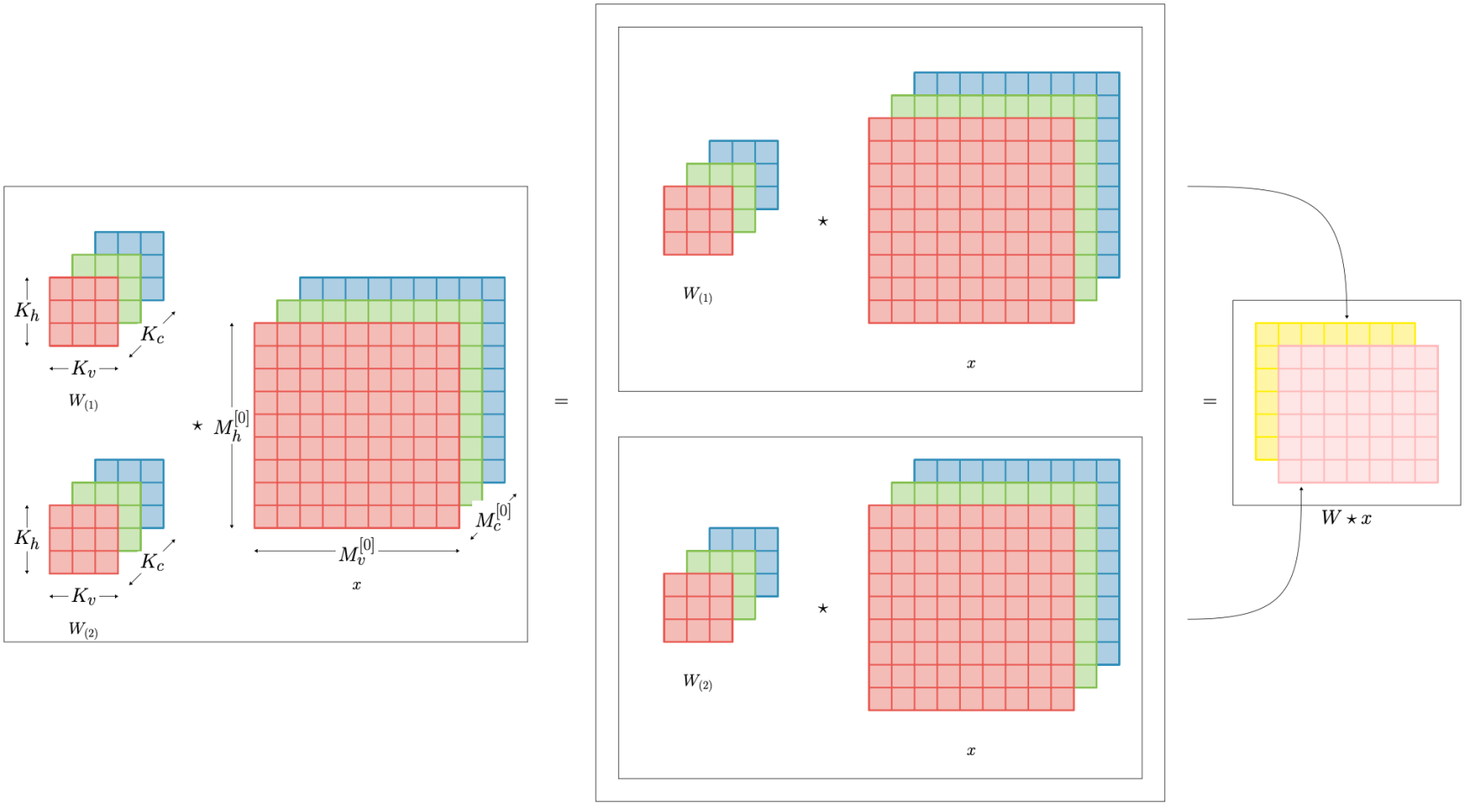


Multiple input and output channels

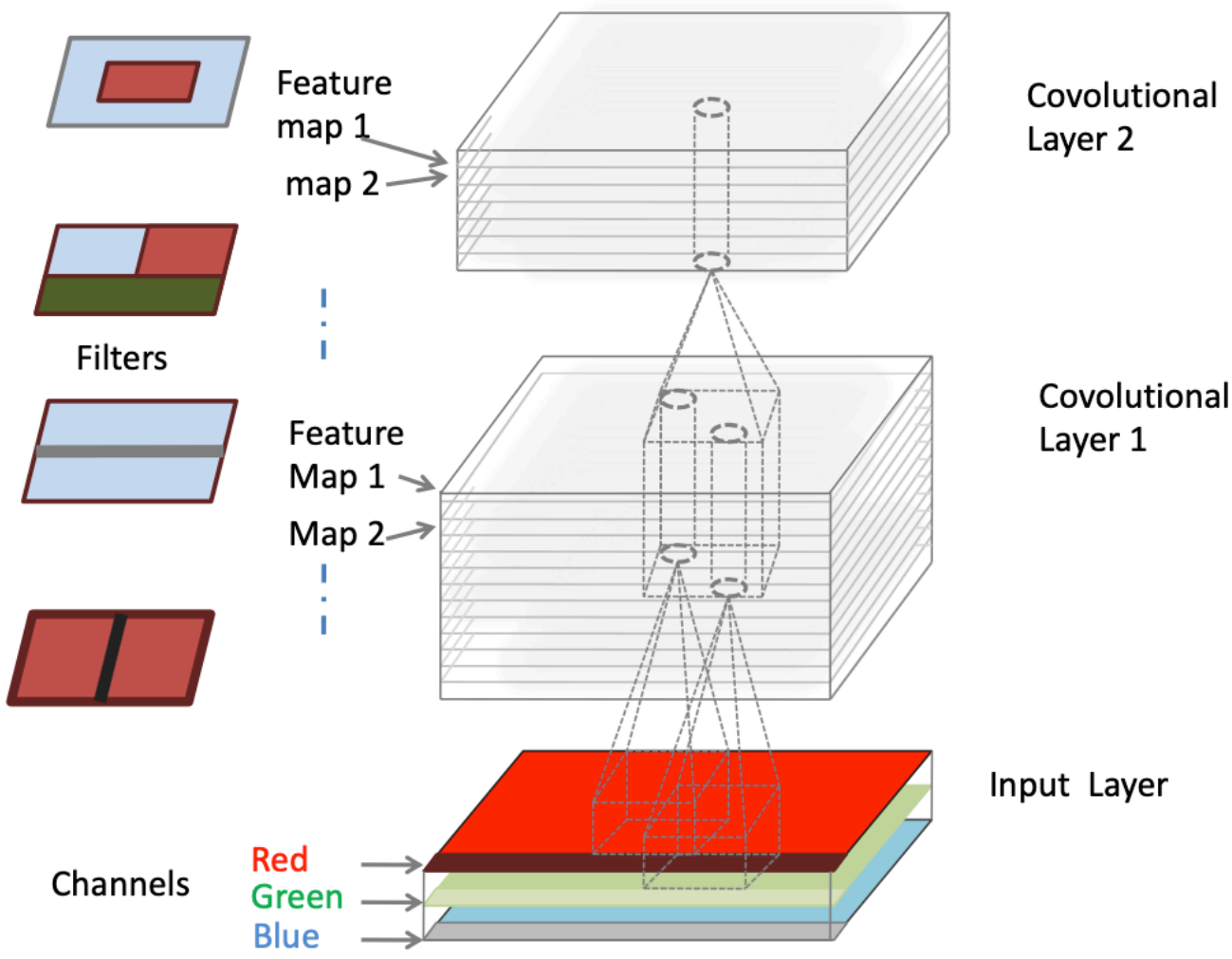
- we also typically have many output channels per layer
- W is now a 4d weight matrix, $H \times W \times C \times D$, for detecting D different feature types
- output is now 3d with D features maps

$$z_{i,j,d} = \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} \sum_{c=1}^{C-1} w_{u,v,c,d} x_{si+u,sj+v,c} + b_d$$

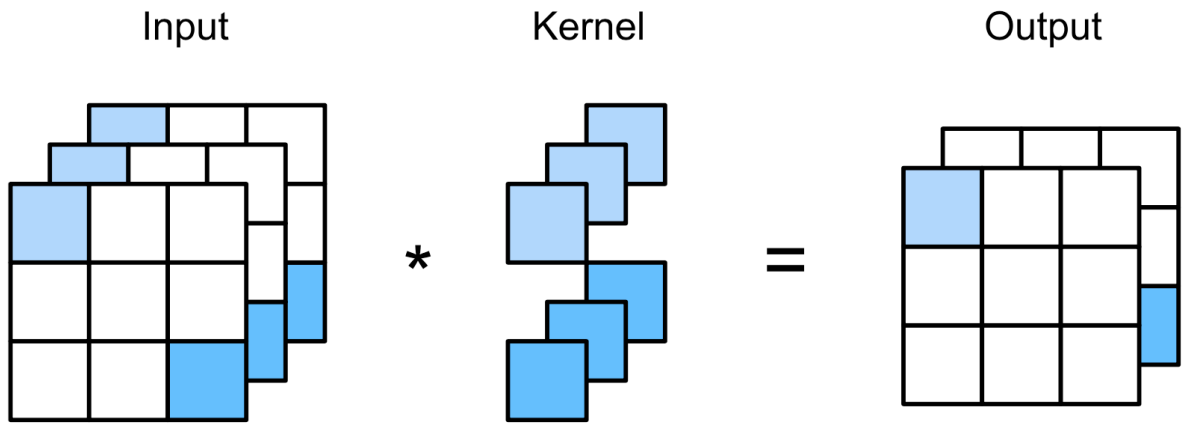
Multiple output channels



CNN example with 2 layers



Pointwise convolution

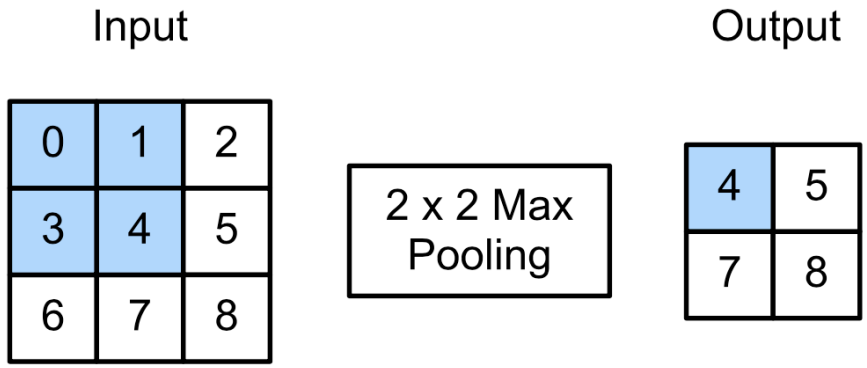


- sometime we just want to take a weighted average of features at a given location
- we can do a 1×1 convolution

$$z_{i,j,d} = \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} \sum_{c=1}^{C-1} w_{c,d} x_{i,j,c} + b_d$$

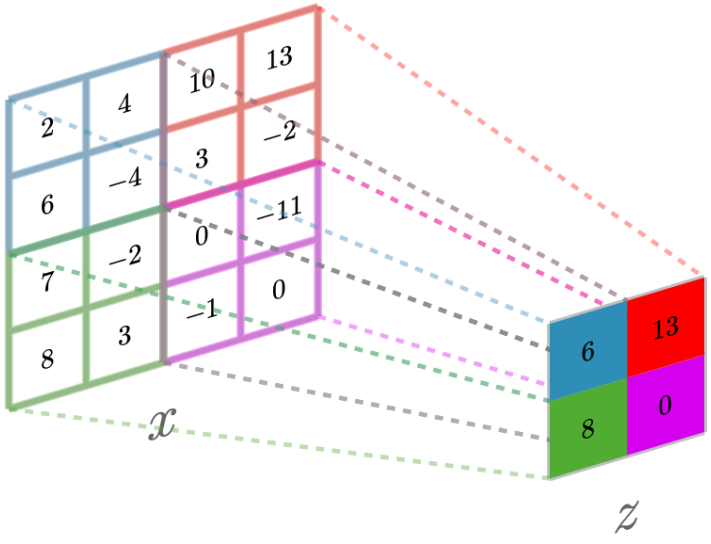
- changed number of channels from C to D without changing spatial dimensions

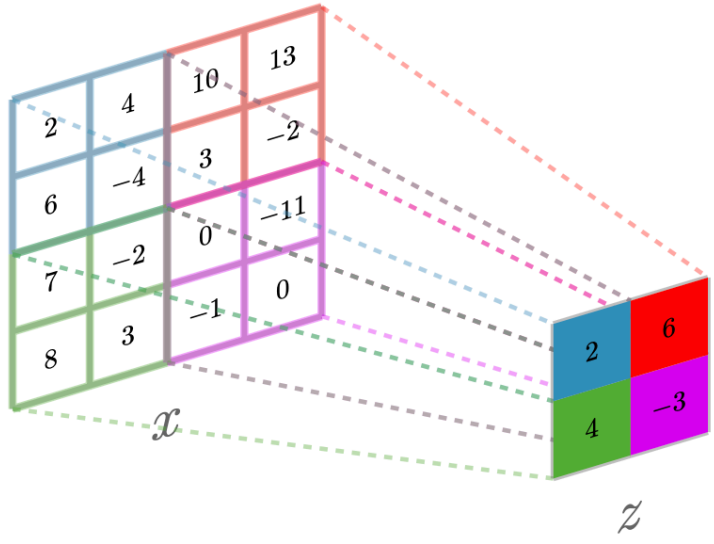
Pooling layers



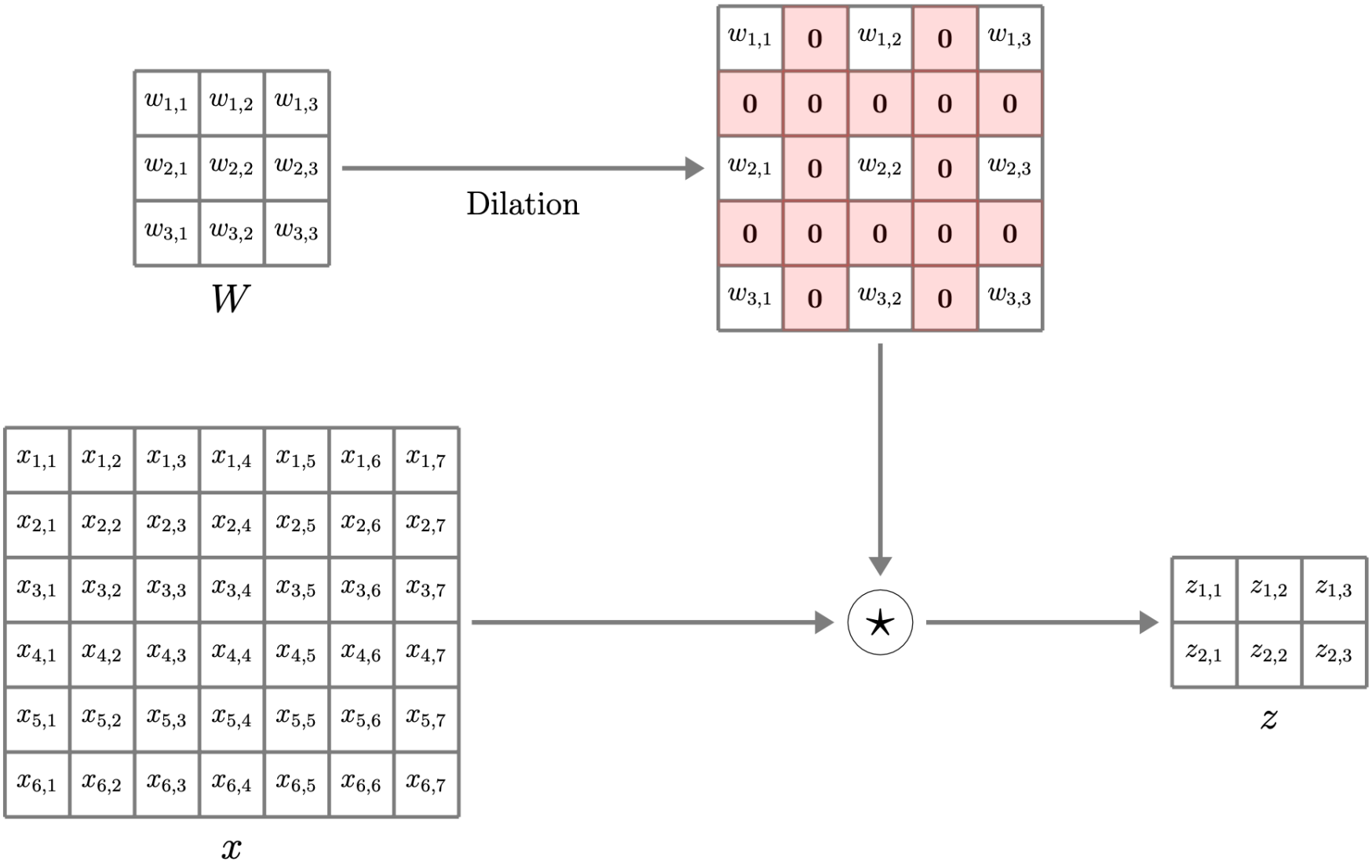
- reduces the spatial dimensionality of the layers
  - each feature channel computed independently
- max pooling computes maximum over its incoming values
- average pooling replaces the max by the mean
- global average pooling averages over all locations in a feature map
  - converts a $H \times W \times D$ feature map into $1 \times 1 \times D$
  - where have seen a 1d version of global average pooling?

Pooling

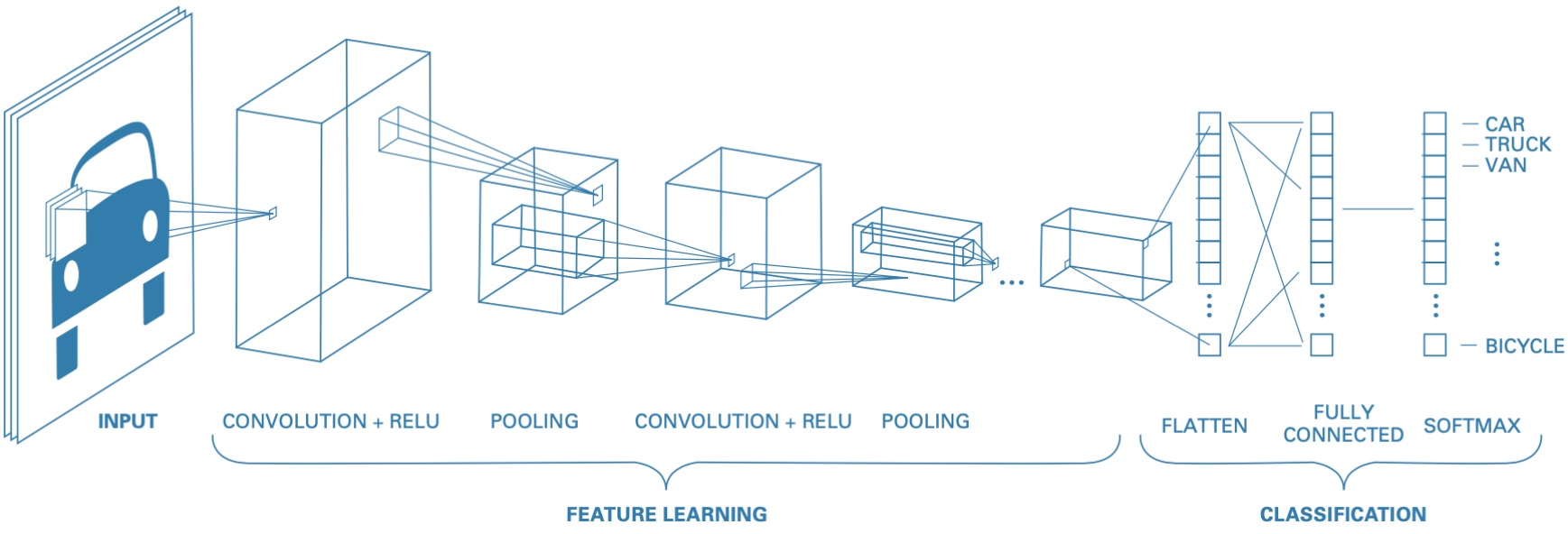

2
4
10
13
6
−4
3
−2
7
−2
0
−11
8
3
−1
0
x
6
13
8
0
z


2
4
10
13
6
−4
3
−2
7
−2
0
−11
8
3
−1
0
x
2
6
4
−3
z

Dilation


$w_{1,1}$ $w_{1,2}$ $w_{1,3}$
$w_{2,1}$ $w_{2,2}$ $w_{2,3}$
$w_{3,1}$ $w_{3,2}$ $w_{3,3}$
W
Dilation
$w_{1,1}$ 0 $w_{1,2}$ 0 $w_{1,3}$
0 0 0 0 0
$w_{2,1}$ 0 $w_{2,2}$ 0 $w_{2,3}$
0 0 0 0 0
$w_{3,1}$ 0 $w_{3,2}$ 0 $w_{3,3}$
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$ $x_{1,7}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$ $x_{2,7}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$ $x_{3,7}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$ $x_{4,7}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$ $x_{5,7}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$ $x_{6,7}$
x
$\star$
$z_{1,1}$ $z_{1,2}$ $z_{1,3}$
$z_{2,1}$ $z_{2,2}$ $z_{2,3}$
z

Putting it all together



Batch normalization layer

- in order to reduce vanishing or exploding gradient problems with SGD, it is often useful to normalize data in every layer
- similar to standardizing input data
- batch normalization: replace activation vector z_n by $\tilde{z}_n$

$$\mu_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{z \in \mathcal{B}} z$$

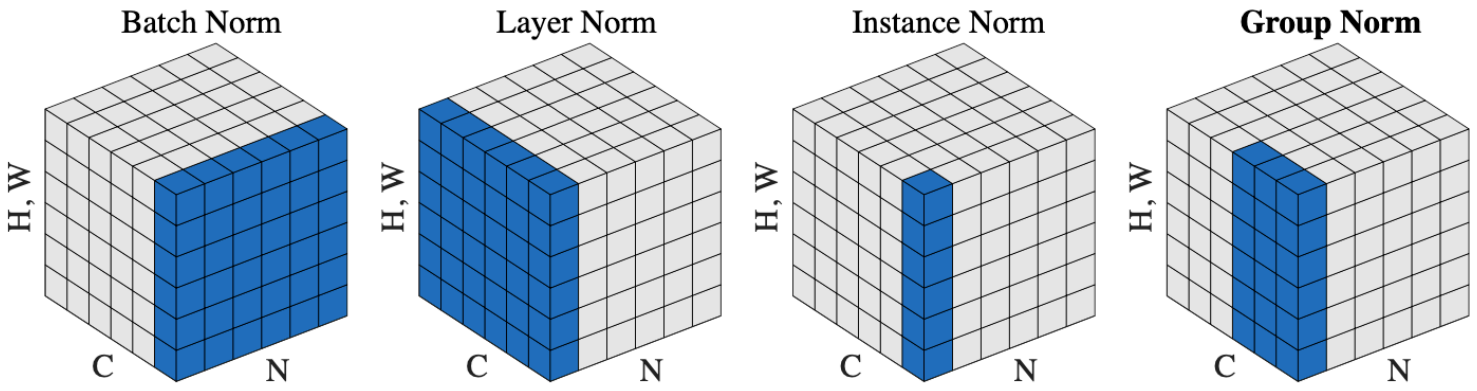
$$\sigma^2_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{z \in \mathcal{B}} (z - \mu_{\mathcal{B}})^2$$

$$\hat{z}_n = \frac{(z - \mu_{\mathcal{B}})}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

$$\tilde{z}_n = \gamma \hat{z}_n + \beta$$

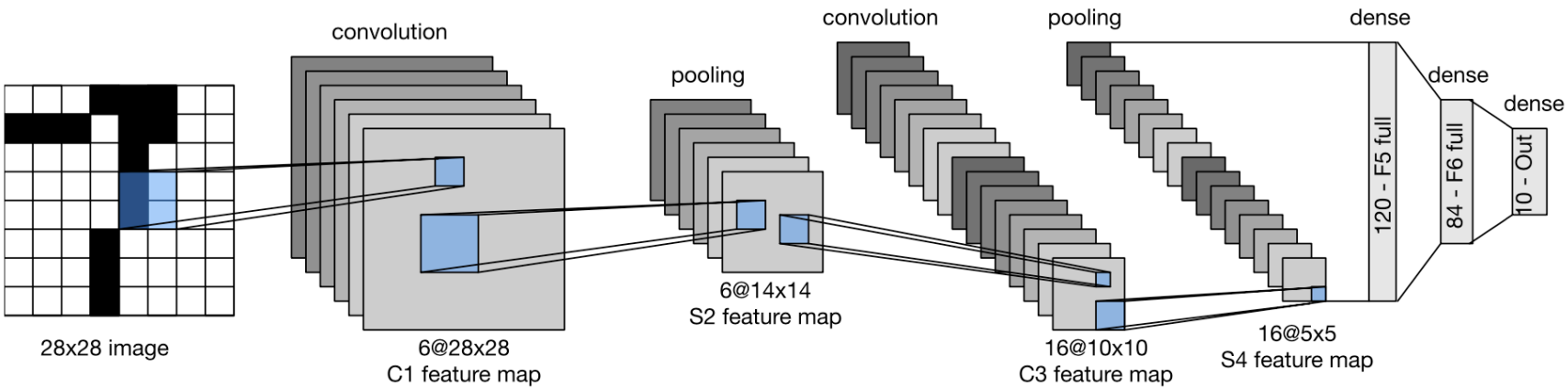
- done dynamically in every layer with every batch
  - trainable parameters: one γ and one β per feature map per layer
  - note: must be held fixed at inference time (post training)

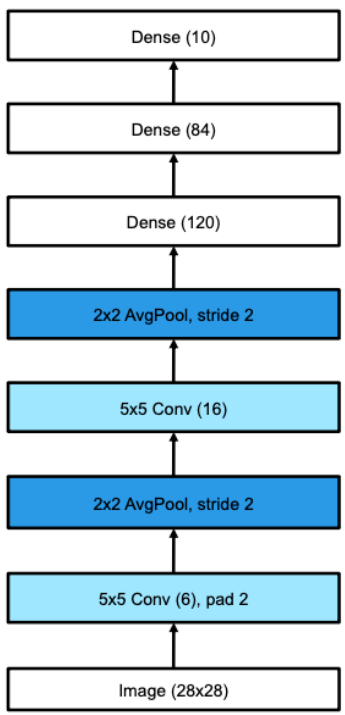
Other kinds of normalizations



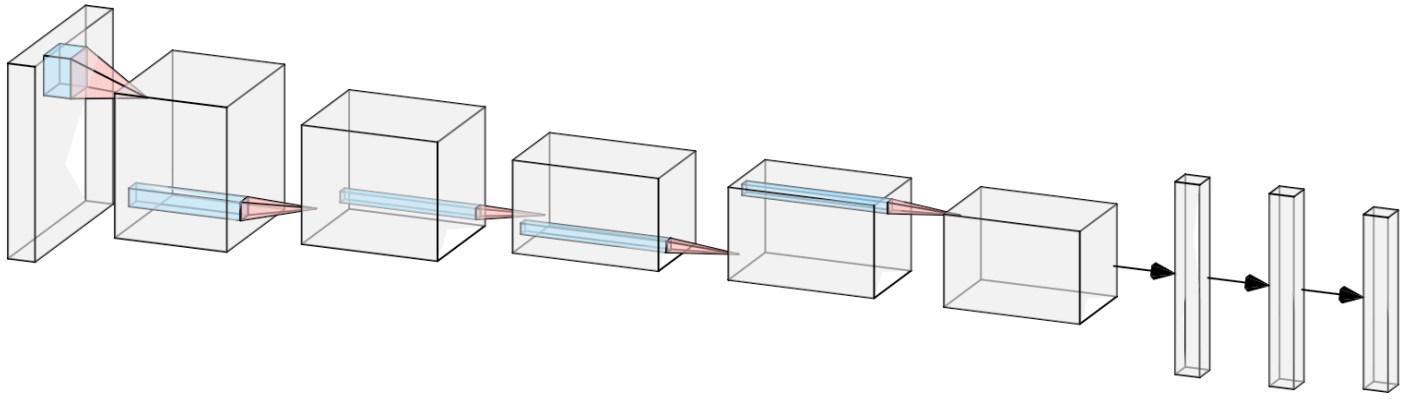
- can be conveniently added as layers in the neural network definition

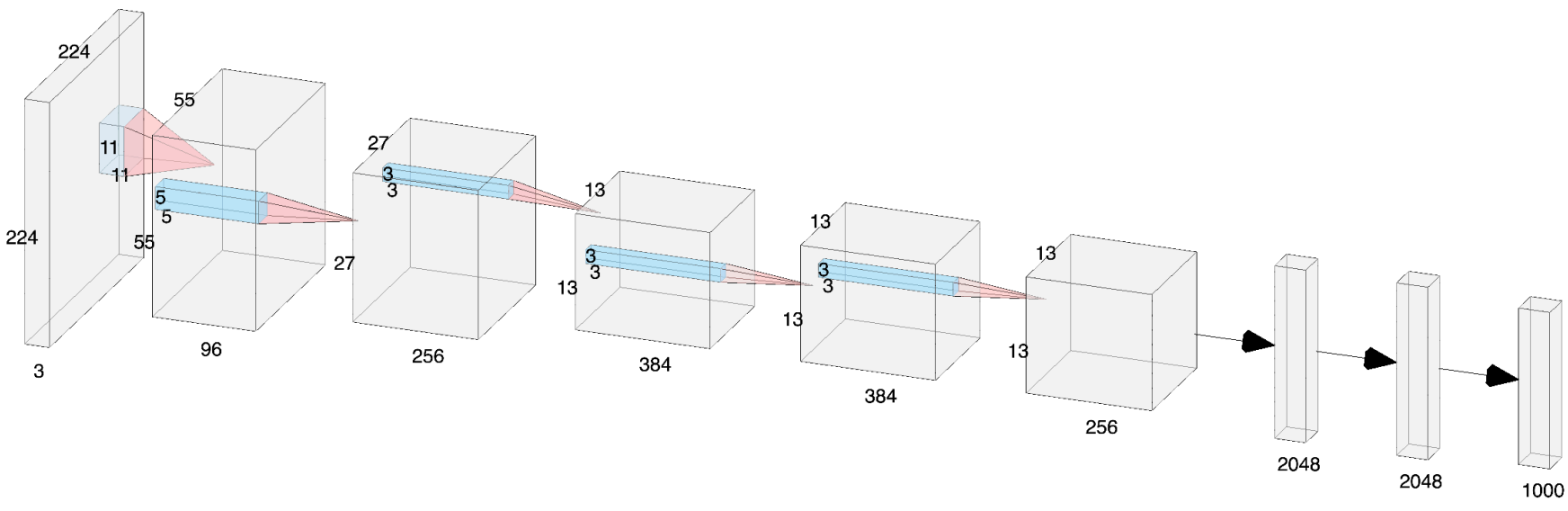
LeNet5


convolution
pooling
convolution
pooling
dense
dense
dense
120 - F5 full
84 - F6 full
10 - Out
28x28 image
6@28x28
C1 feature map
6@14x14
S2 feature map
16@10x10
C3 feature map
16@5x5
S4 feature map

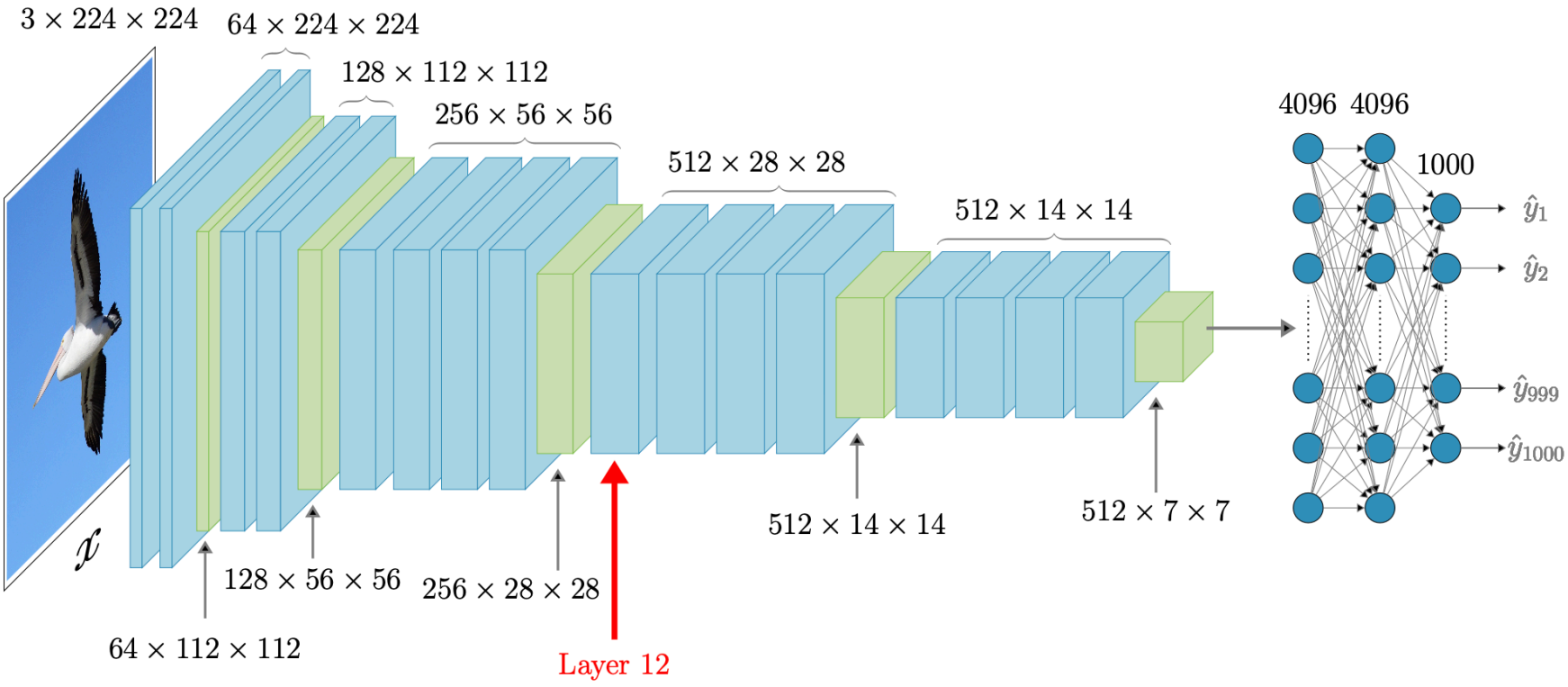

Dense (10)
Dense (84)
Dense (120)
2x2 AvgPool, stride 2
5x5 Conv (16)
2x2 AvgPool, stride 2
5x5 Conv (6), pad 2
Image (28x28)

CNN networks





VGG19


3 × 224 × 224
64 × 224 × 224
128 × 112 × 112
256 × 56 × 56
512 × 28 × 28
512 × 14 × 14
512 × 7 × 7
4096
4096
1000
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_{999}$
$\hat{y}_{1000}$
x
64 × 112 × 112
128 × 56 × 56
256 × 28 × 28
Layer 12
512 × 14 × 14

Layer. No.	Type of Layer	Output Dimension	No. of Neurons	No. of Learned Parameters
0	Input	$3 \times 224 \times 224$	-	-
1	Convolution	$64 \times 224 \times 224$	3,211,264	1,792
2	Convolution	$64 \times 224 \times 224$	3,211,264	36,928
3	Max-pooling	$64 \times 112 \times 112$	802,816	0
4	Convolution	$128 \times 112 \times 112$	1,605,632	73,856
5	Convolution	$128 \times 112 \times 112$	1,605,632	147,584
6	Max-pooling	$128 \times 56 \times 56$	401,408	0
7	Convolution	$256 \times 56 \times 56$	802,816	295,168
8	Convolution	$256 \times 56 \times 56$	802,816	590,080
9	Convolution	$256 \times 56 \times 56$	802,816	590,080
10	Convolution	$256 \times 56 \times 56$	802,816	590,080
11	Max-pooling	$256 \times 28 \times 28$	200,704	0
12	Convolution	$512 \times 28 \times 28$	401,408	1,180,160
13	Convolution	$512 \times 28 \times 28$	401,408	2,359,808
14	Convolution	$512 \times 28 \times 28$	401,408	2,359,808
15	Convolution	$512 \times 28 \times 28$	401,408	2,359,808
16	Max-pooling	$512 \times 14 \times 14$	100,352	0
17	Convolution	$512 \times 14 \times 14$	100,352	2,359,808
18	Convolution	$512 \times 14 \times 14$	100,352	2,359,808
19	Convolution	$512 \times 14 \times 14$	100,352	2,359,808
20	Convolution	$512 \times 14 \times 14$	100,352	2,359,808
21	Max-pooling	$512 \times 7 \times 7$	25,088	0
Flattening to a vector of length 25,088				
22	Fully connected	4,096	4,096	102,764,544
23	Fully connected	4,096	4,096	16,781,312
24	Fully connected	1,000	1,000	4,097,000
			Total: 16,391,656	Total: 143,667,240

CNN in pytorch

```
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=16, kernel_size=5,
            ↪  stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(16, 32, 5, 1, 2),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        self.out = nn.Linear(32 * 7 * 7, 10)  # fully connected layer,
        ↪  output 10 classes

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        # flatten the output of conv2 to (batch_size, 32 * 7 * 7)
        x = torch.flatten(x, start_dim=1)
        output = self.out(x)
        return output
```

CNNs in pytorch

```
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Exercises

- Build and train an FNN and a CNN for MNIST data
- Build and train an FNN and a CNN for CIFAR10 data