OEAW AI SUMMER SCHOOL

DEEP LEARNING III

Convolutional Neural Networks

Michael Widrich, Johannes Brandstetter
Institute for Machine Learning

JKU
Institute for
Machine Learning

Lecture III: Convolutional Neural Networks

Basics of Convolutional Neural Networks (ConvNets/CNNs)

Neural Nets for Image Recognition

- ImageNet Large Scale Visual Recognition Competition (ILSVRC)
  - 1.2M images, 1,000 different classes
  - Yearly challenge to find the “State of the Art” in image recognition

Neural Nets for Image Recognition

- ImageNet Large Scale Visual Recognition Competition (ILSVRC)
  - 1.2M images, 1,000 different classes
  - Yearly challenge to find the “State of the Art” in image recognition
  - ILSVRC 2012: Won by the only CNN-based solution

Neural Nets for Image Recognition

- ImageNet Large Scale Visual Recognition Competition (ILSVRC)
  - 1.2M images, 1,000 different classes
  - Yearly challenge to find the “State of the Art” in image recognition
  - ILSVRC 2012: Won by the only CNN-based solution
  - ILSVRC 2013: Best 5 participants were all CNNs (9 of the top 10 were CNNs)

Neural Nets for Image Recognition

- ImageNet Large Scale Visual Recognition Competition (ILSVRC)
  - 1.2M images, 1,000 different classes
  - Yearly challenge to find the “State of the Art” in image recognition
  - ILSVRC 2012: Won by the only CNN-based solution
  - ILSVRC 2013: Best 5 participants were all CNNs (9 of the top 10 were CNNs)
  - ILSVRC 2014: Everyone uses CNNs

Neural Nets for Image Recognition

- ImageNet Large Scale Visual Recognition Competition (ILSVRC)
  - 1.2M images, 1,000 different classes
  - Yearly challenge to find the “State of the Art” in image recognition
  - ILSVRC 2012: Won by the only CNN-based solution
  - ILSVRC 2013: Best 5 participants were all CNNs (9 of the top 10 were CNNs)
  - ILSVRC 2014: Everyone uses CNNs
- CNNs are useful whenever there is “local structure” in the data:
  - Pixel data
  - Audio data
  - Voxel data
  - ...

Basic considerations for Image(-like) Data

- Images are extremely high-dimensional:
  - 250 x 250 pixels * 3 color channels = 187.5k input dimensions
- Pixels that are near each other are highly correlated.
- Same basic patches (e.g. edges, corners) appear on all positions of the image.
- Often invariances to certain variations are desired (e.g. translation invariance).

Receptive Field

- Pixels that are near each other are highly correlated.
- Same basic patches (e.g. edges, corners) appear on all positions of the image.

Receptive Field

- Pixels that are near each other are highly correlated.
- Same basic patches (e.g. edges, corners) appear on all positions of the image.
- We can detect those basic patches by only viewing a small part of the image.

$\rightarrow$ We can use a network with a small receptive field!

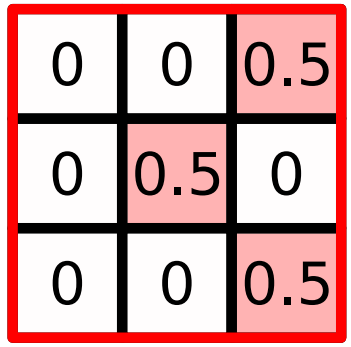
Receptive Field

- Pixels that are near each other are highly correlated.
- Same basic patches (e.g. edges, corners) appear on all positions of the image.
- We can detect those basic patches by only viewing a small part of the image.

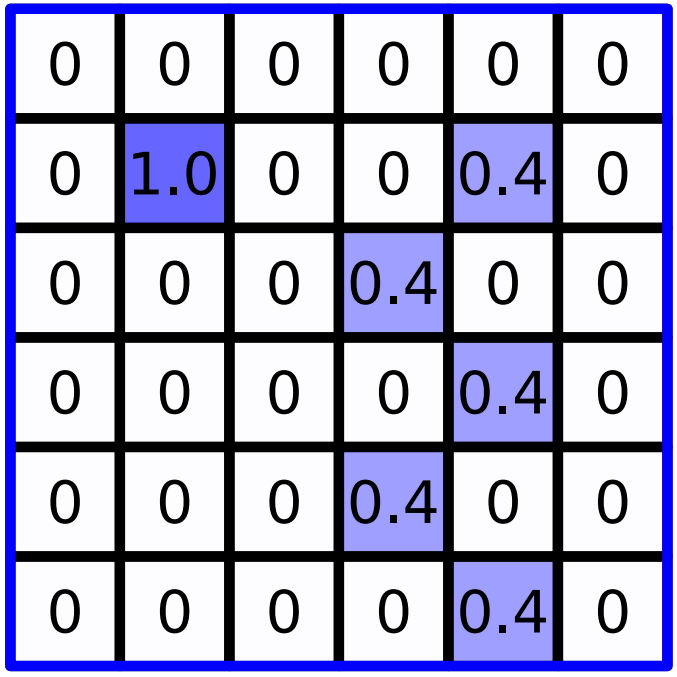
$\rightarrow$ We can use a network with a small receptive field!

- Receptive field: Connect network to patch of image using weight matrix (=kernel, filter)

Receptive Field

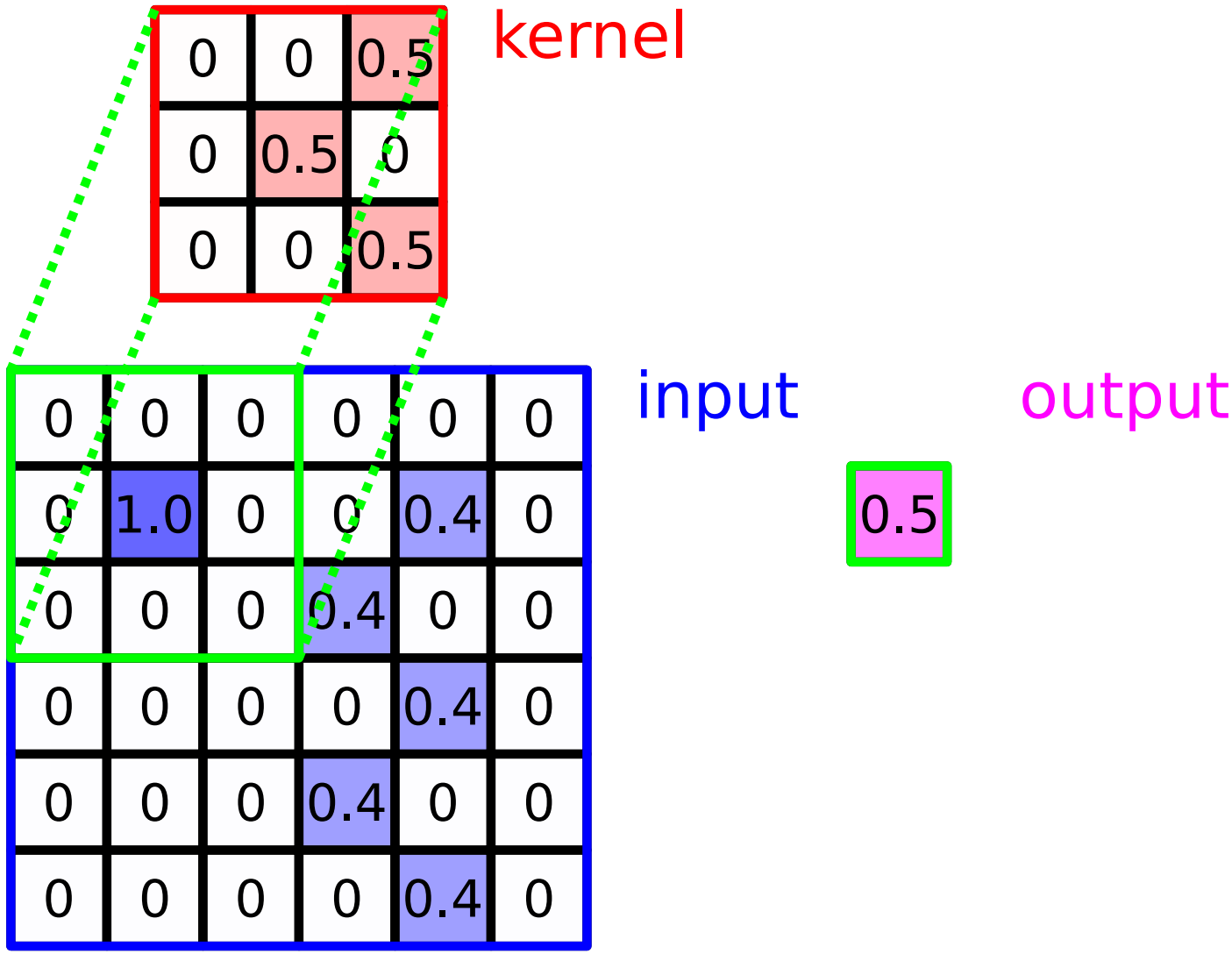


kernel



input

Receptive Field



Weight sharing

- Same basic patches (e.g. edges, corners) appear on all positions of the image.
- Often invariances to certain variations are desired (e.g. translation invariance).

Weight sharing

- Same basic patches (e.g. edges, corners) appear on all positions of the image.
- Often invariances to certain variations are desired (e.g. translation invariance).

$\rightarrow$ We can re-use the receptive field at all positions of the image!

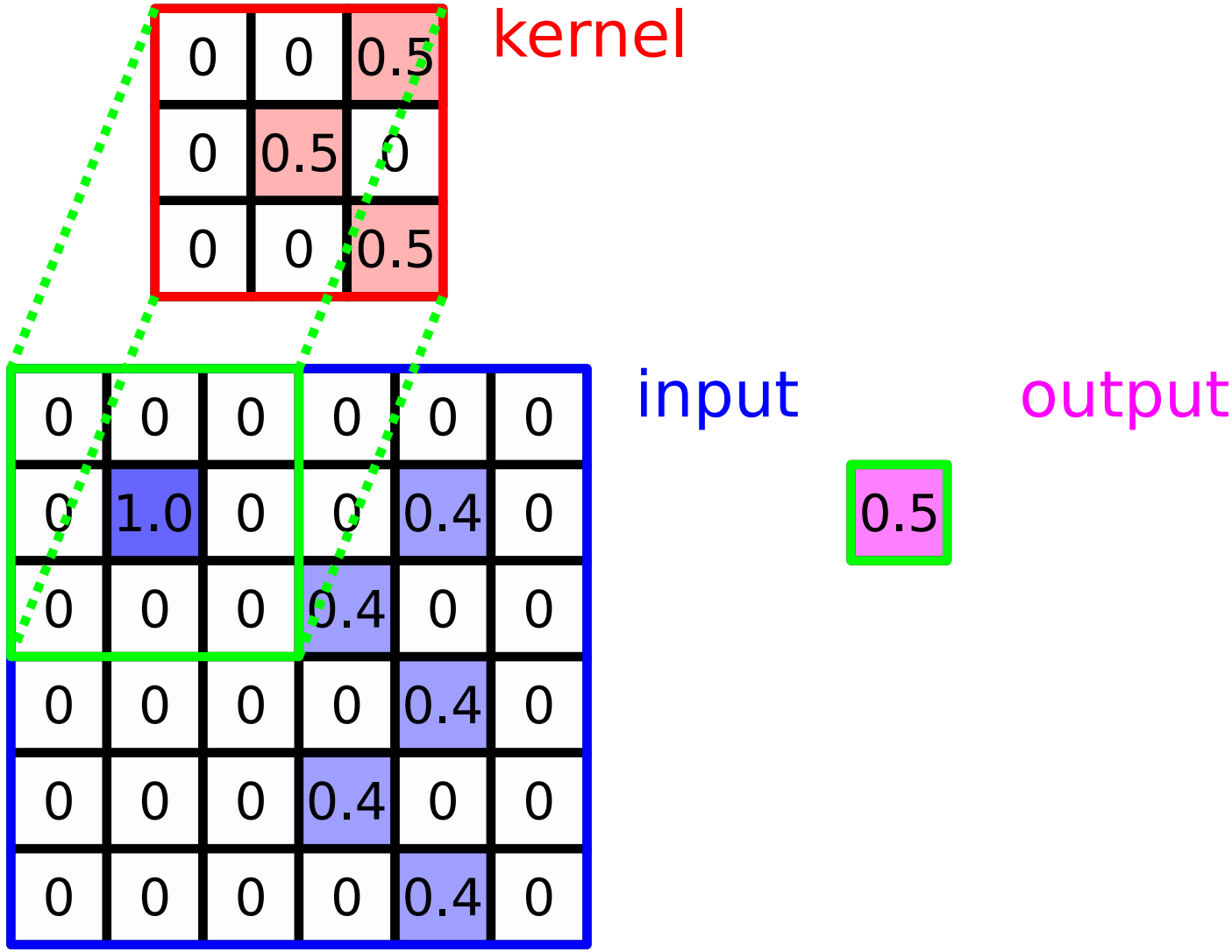
Weight sharing

- Same basic patches (e.g. edges, corners) appear on all positions of the image.
- Often invariances to certain variations are desired (e.g. translation invariance).

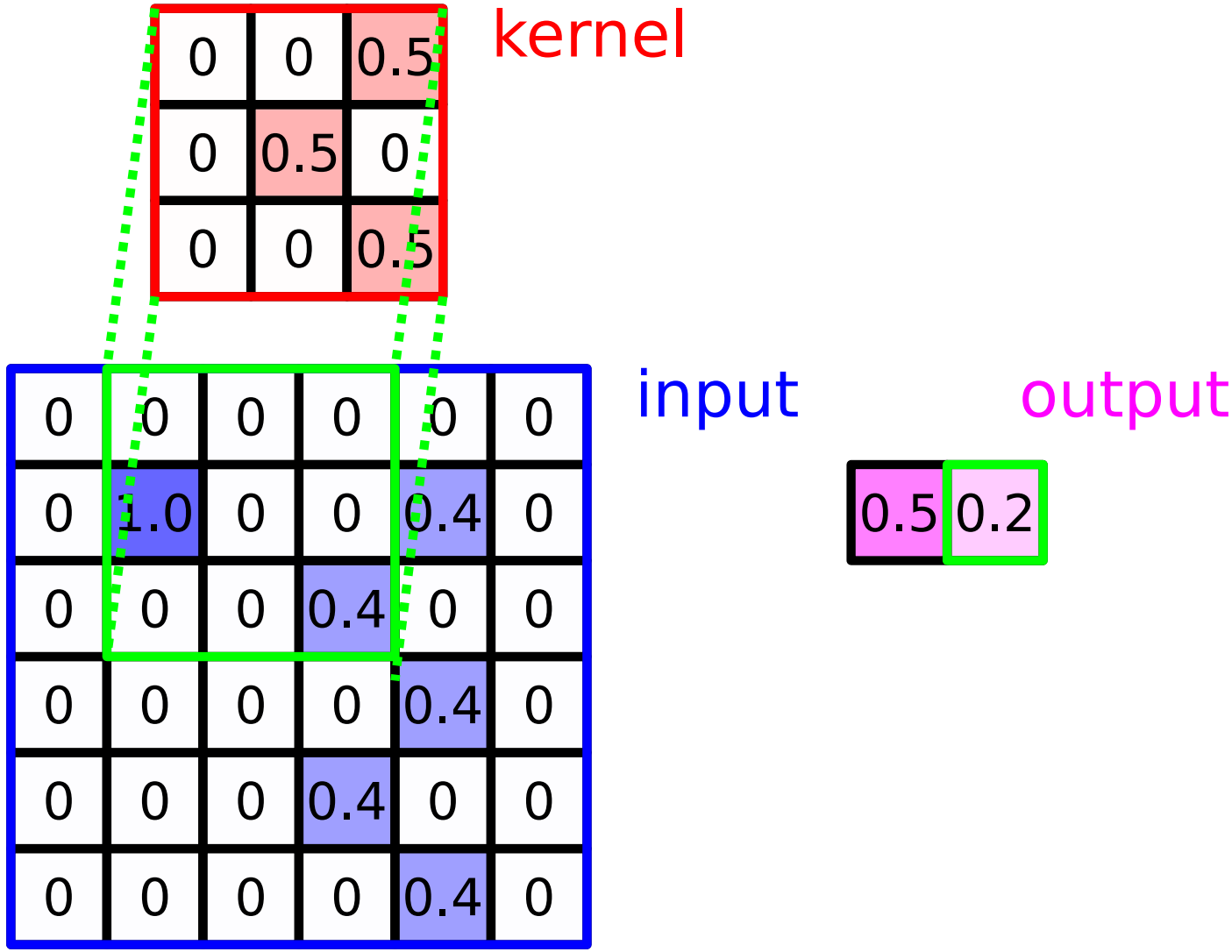
→ We can re-use the receptive field at all positions of the image!

- Re-using the kernel weight matrix is called weight sharing.
  - We apply our kernel to all image positions while keeping the weights the same.

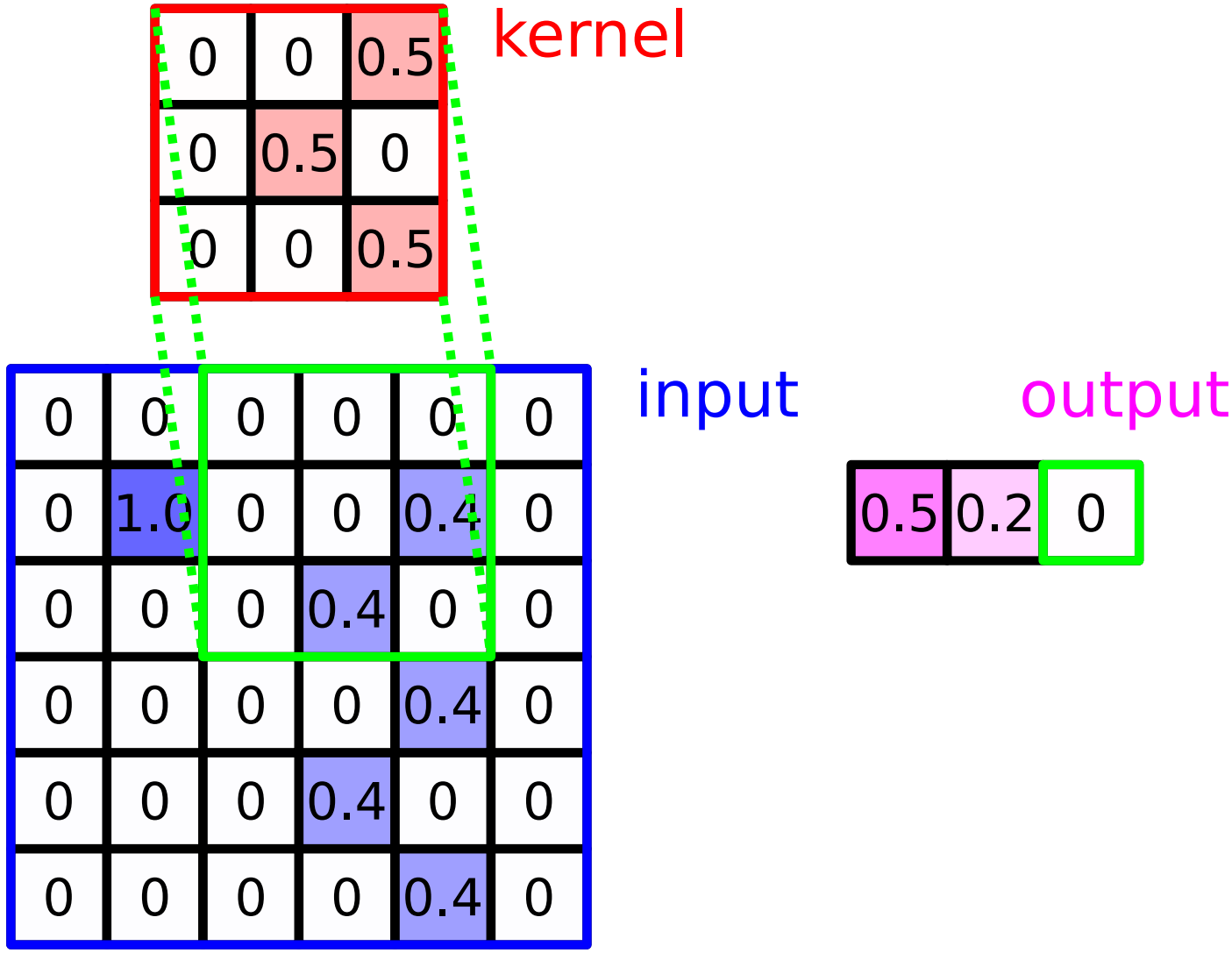
Weight sharing



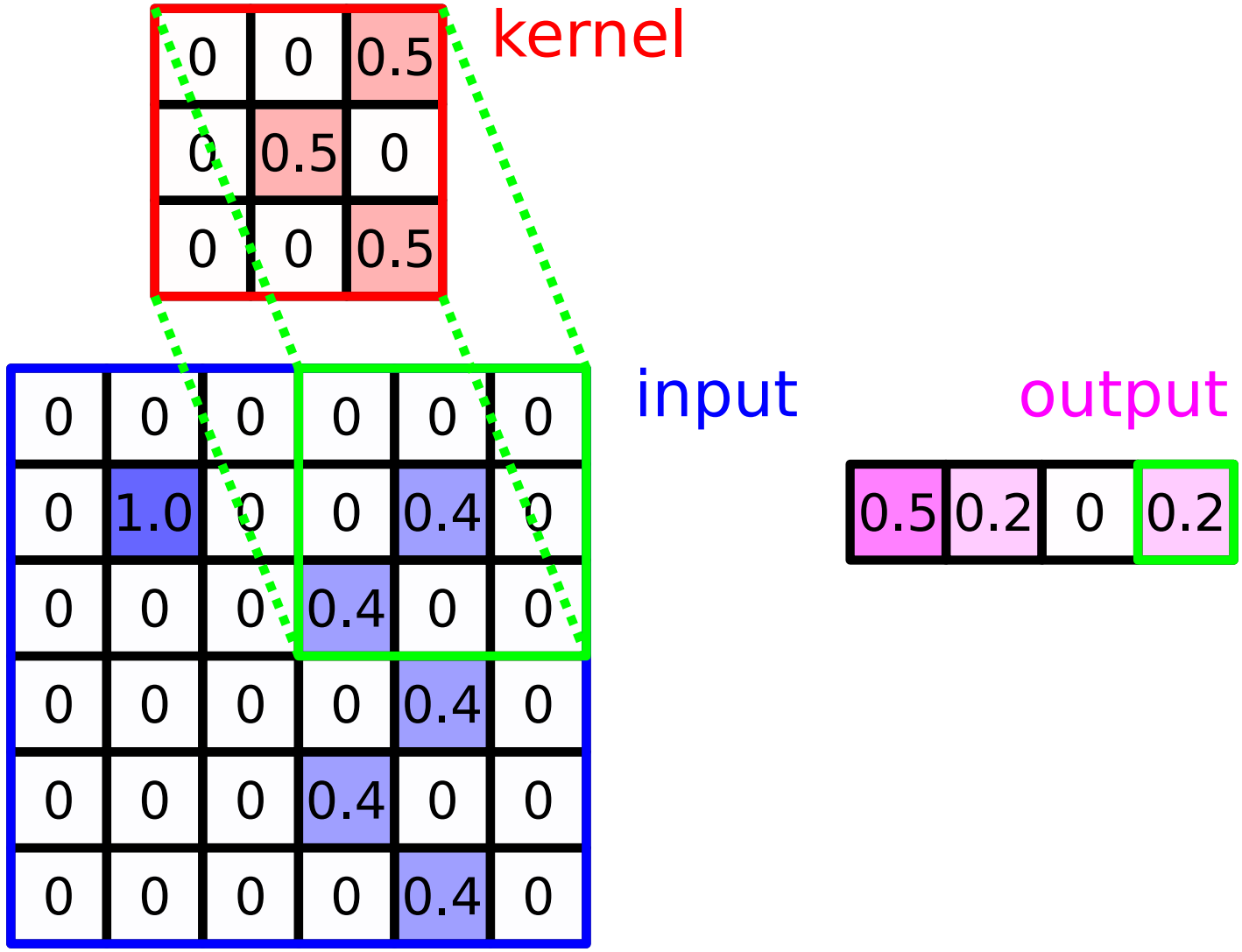
Weight sharing



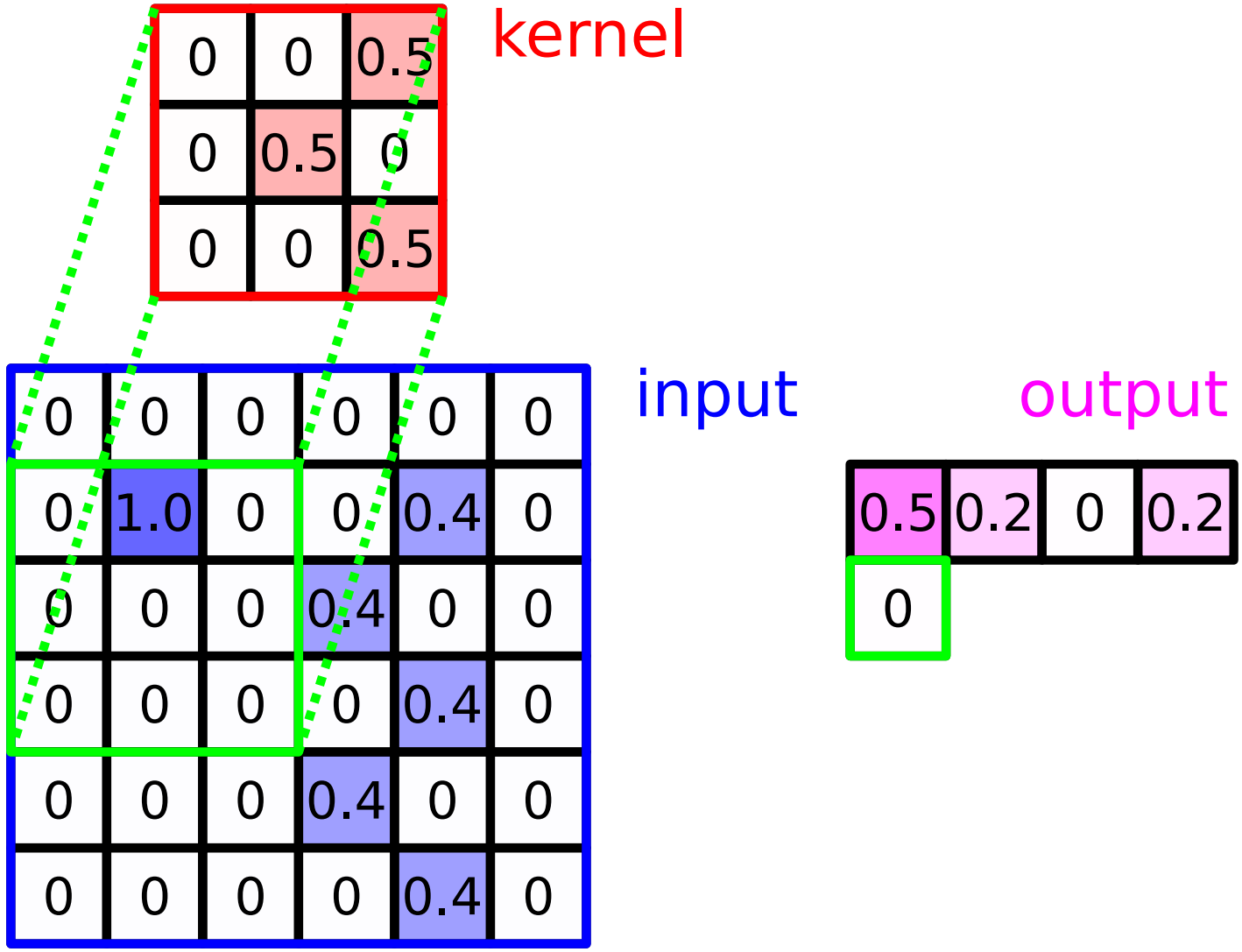
Weight sharing



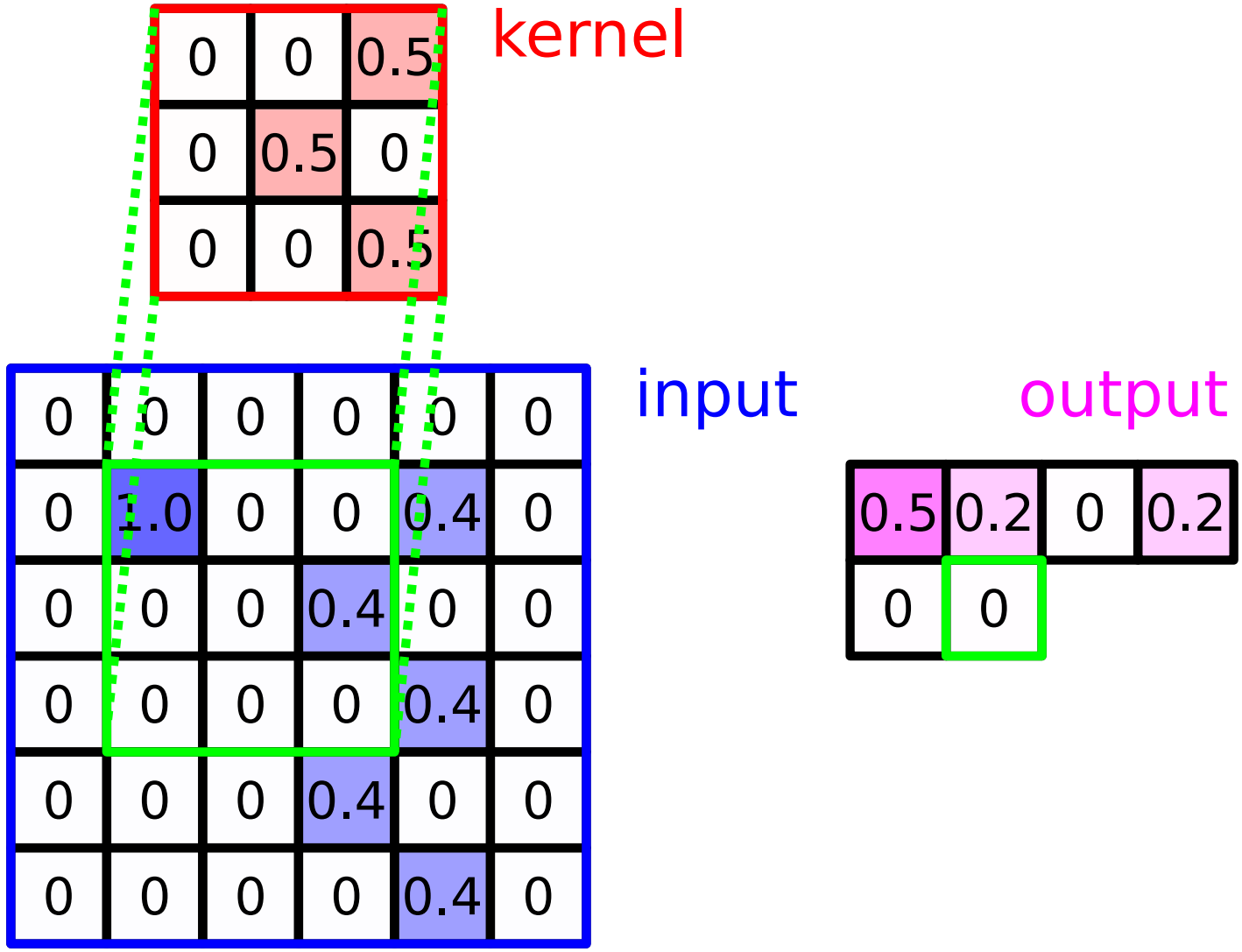
Weight sharing



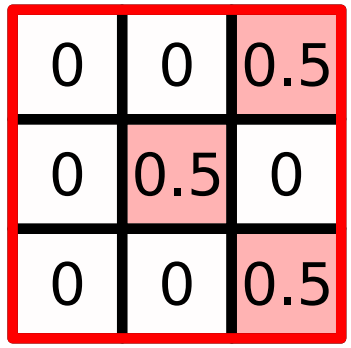
Weight sharing



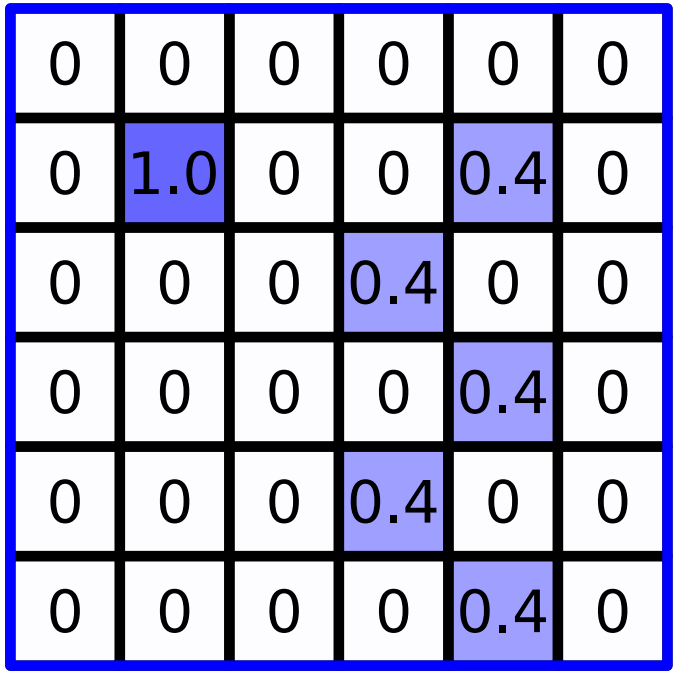
Weight sharing



Weight sharing

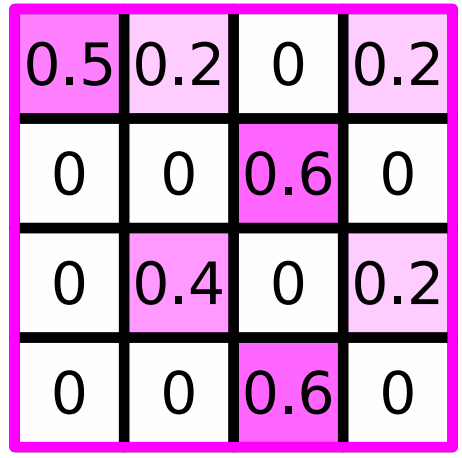


kernel



input

output



Convolution

- Applying the kernel W to all image positions can be viewed as convolution.
  - Discrete convolution: $\mathbf{X} * \mathbf{W}$
- We get an output value for each time we apply the kernel.
- We apply an activation function to those outputs afterwards (usually relu).

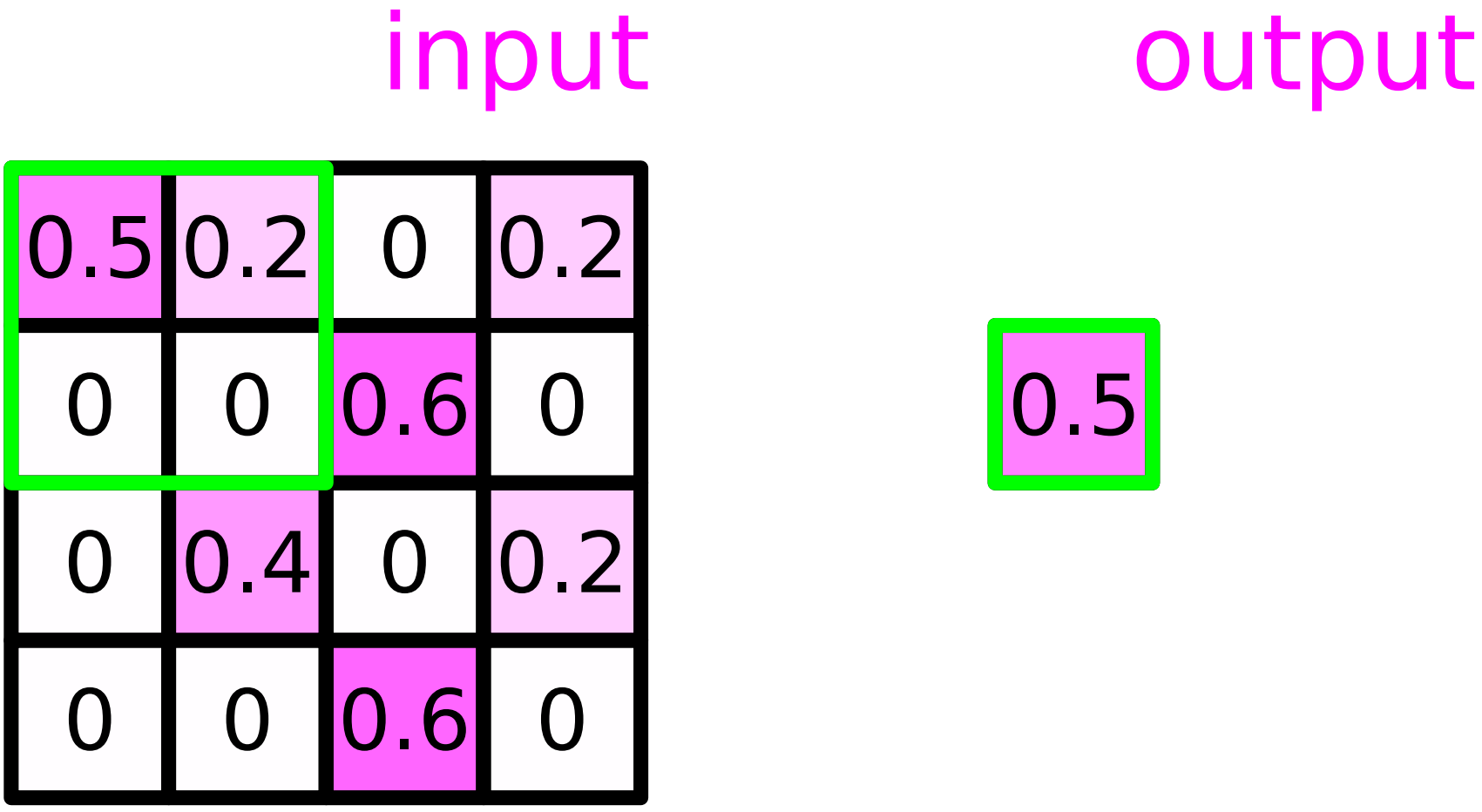
Convolution

- Applying the kernel W to all image positions can be viewed as convolution.
  - Discrete convolution: $\mathbf{X} * \mathbf{W}$
- We get an output value for each time we apply the kernel.
- We apply an activation function to those outputs afterwards (usually relu).
- Applying the kernel and the activation function is one layer in a Convolutional Neural Network (CNN).

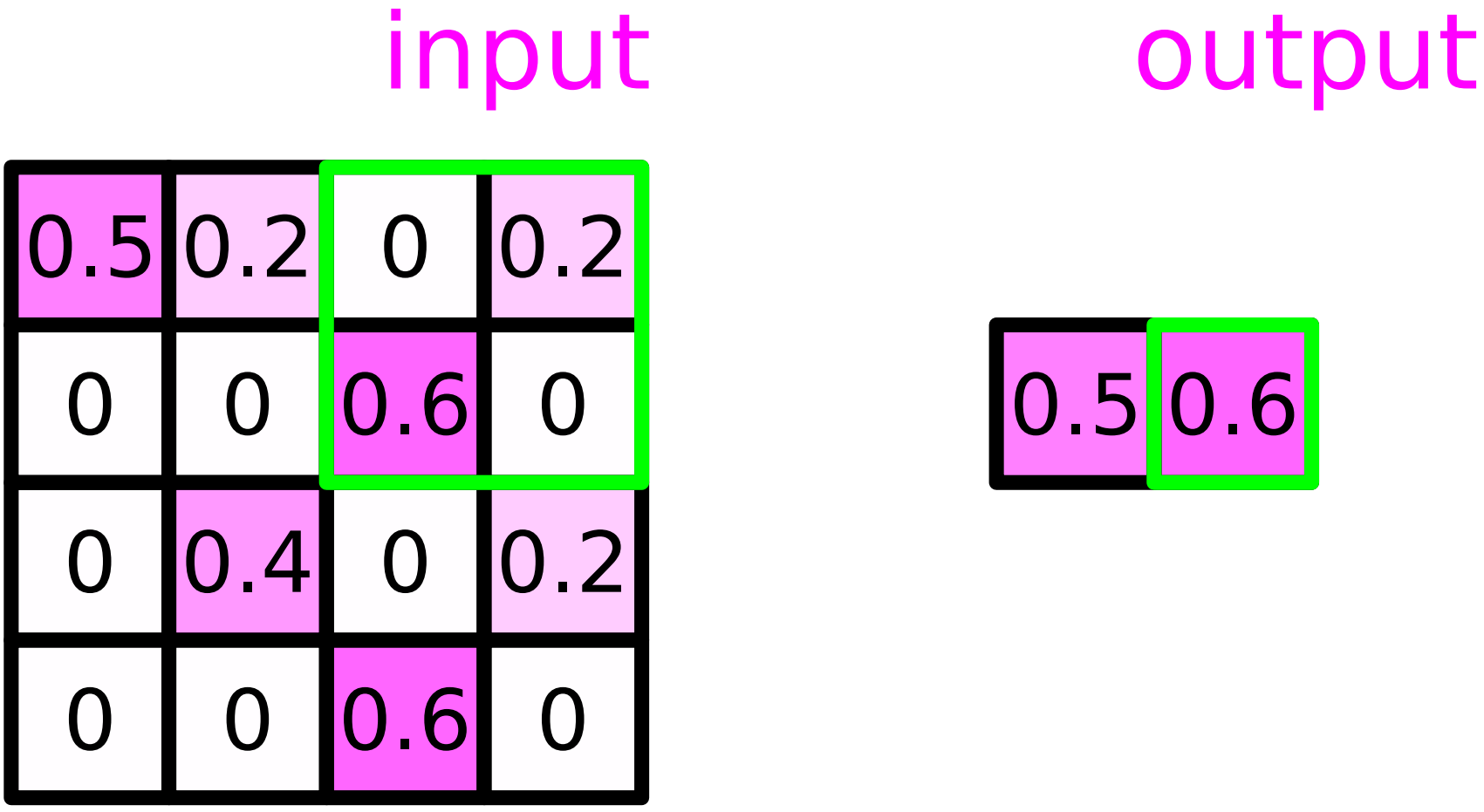
Subsampling ("Pooling")

- We can subsample an image to produce a smaller image.
- We can do the same to the result of a convolution.
- This is called **pooling**.
- There are different ways to perform pooling. Most popular:
  - Average Pooling: take the average value in a $k \times k$ field
  - Max Pooling: take the maximum value in a $k \times k$ field
  - N-Max Pooling: take the mean over the n maximum values in a $k \times k$ field
- Pooling will lead to loss of information.
- Pooling will increase receptive field through depth of network.

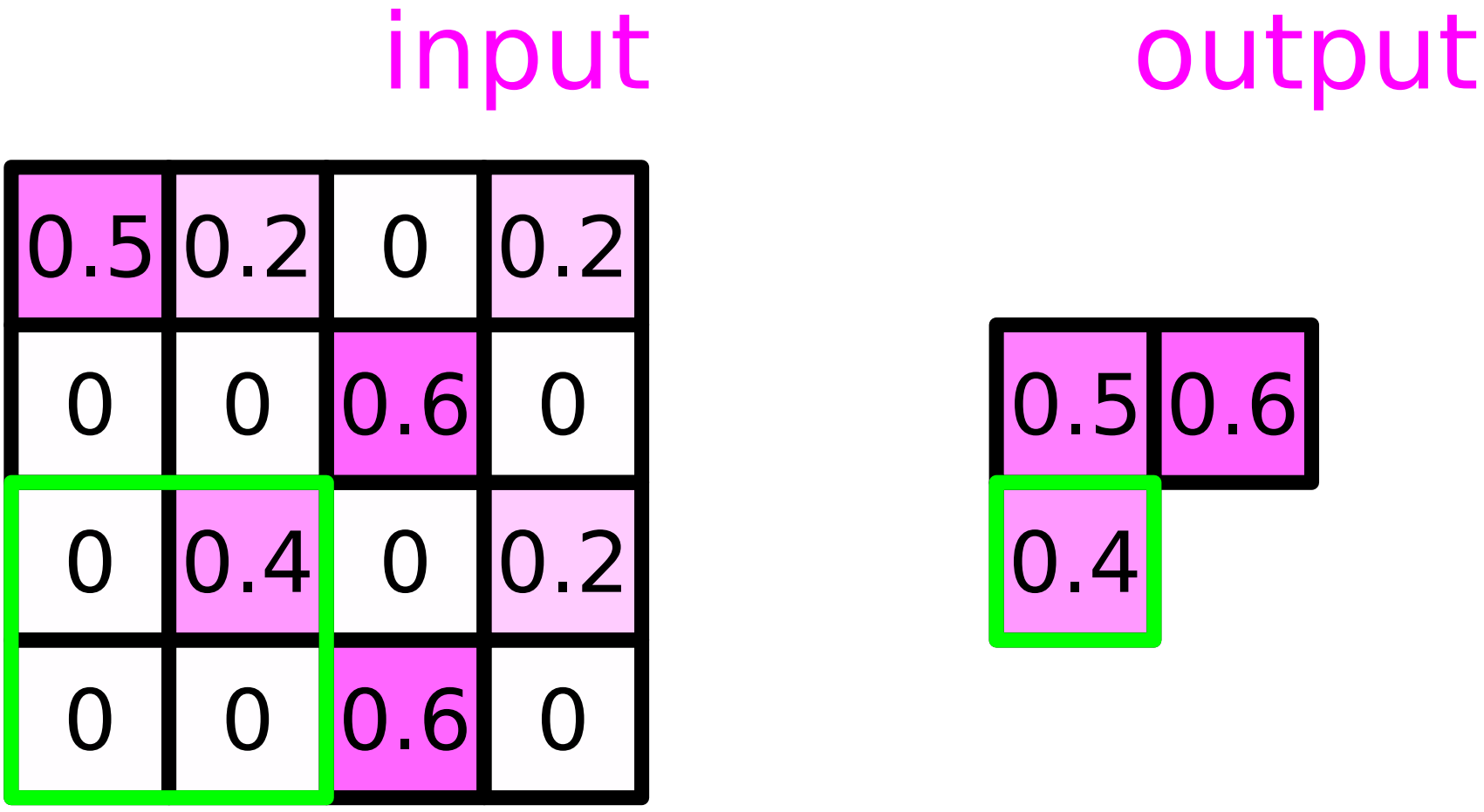
Max Pooling



Max Pooling



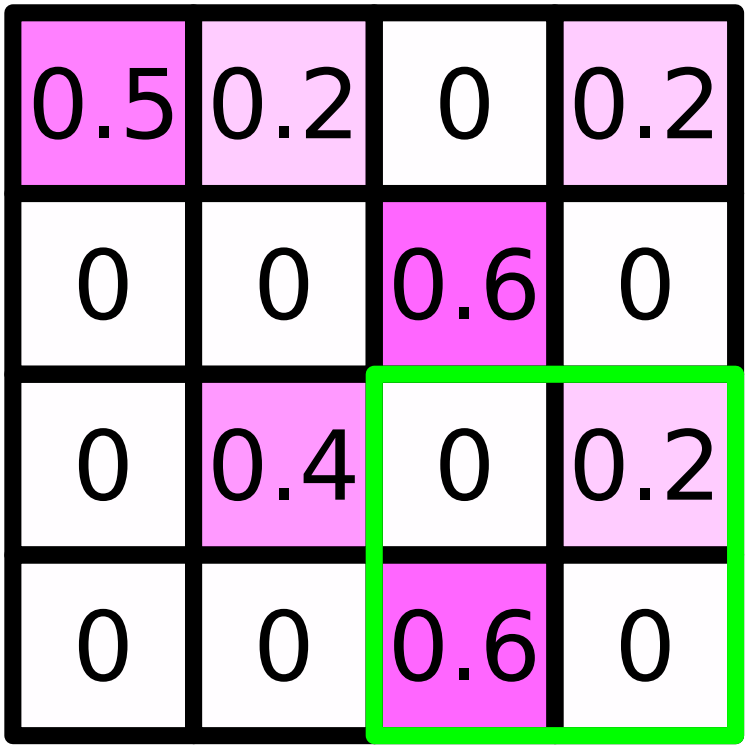
Max Pooling

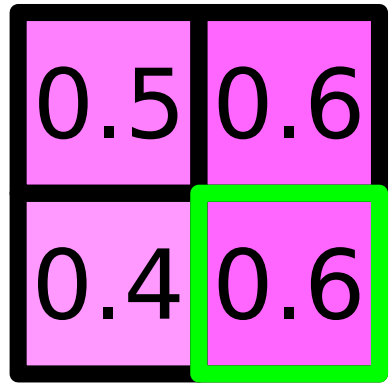


Max Pooling

input

output

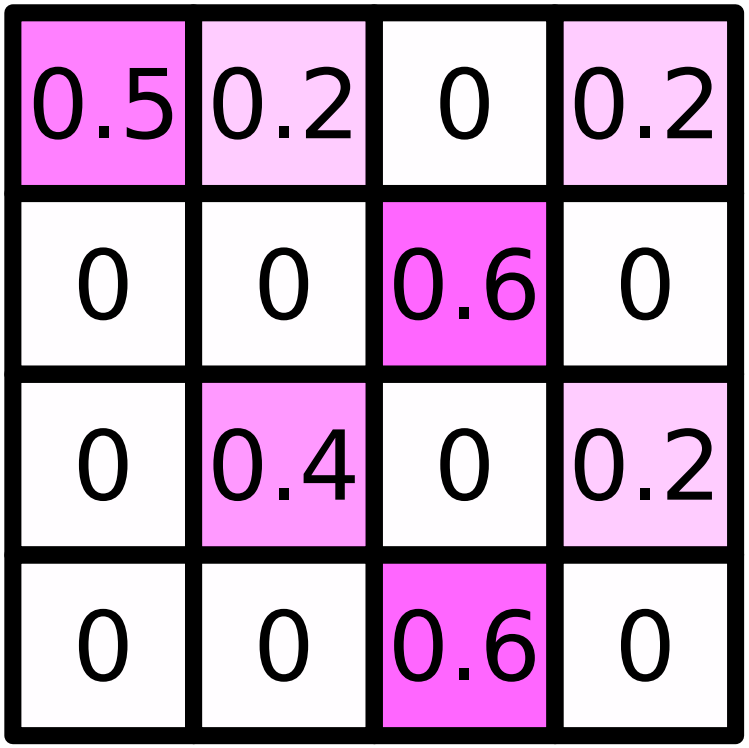


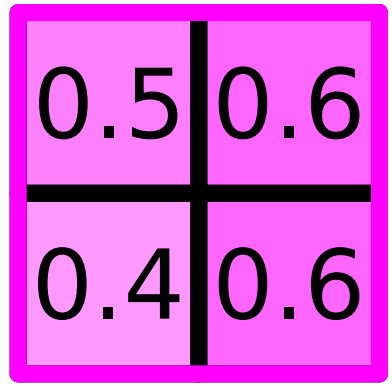


Max Pooling

input

output



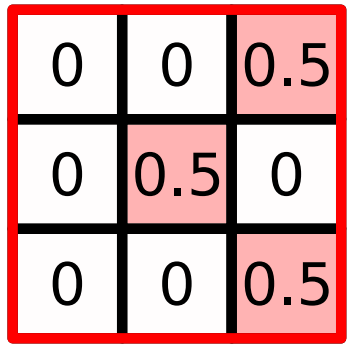


Padding

- Padding can be used to keep the input and output size the same.

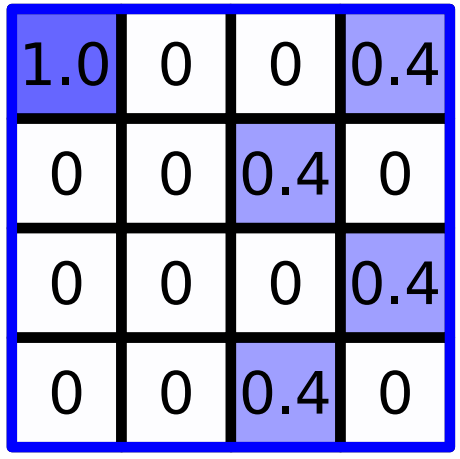
  Zero-Padding: Add zeros at borders of input
  Repeat-Padding: Duplicate the border values
  Other padding methods: Mean, weighted sum, . . .
- Can be applied to input image and feature maps between layers to keep original input size.

Zero-Padding

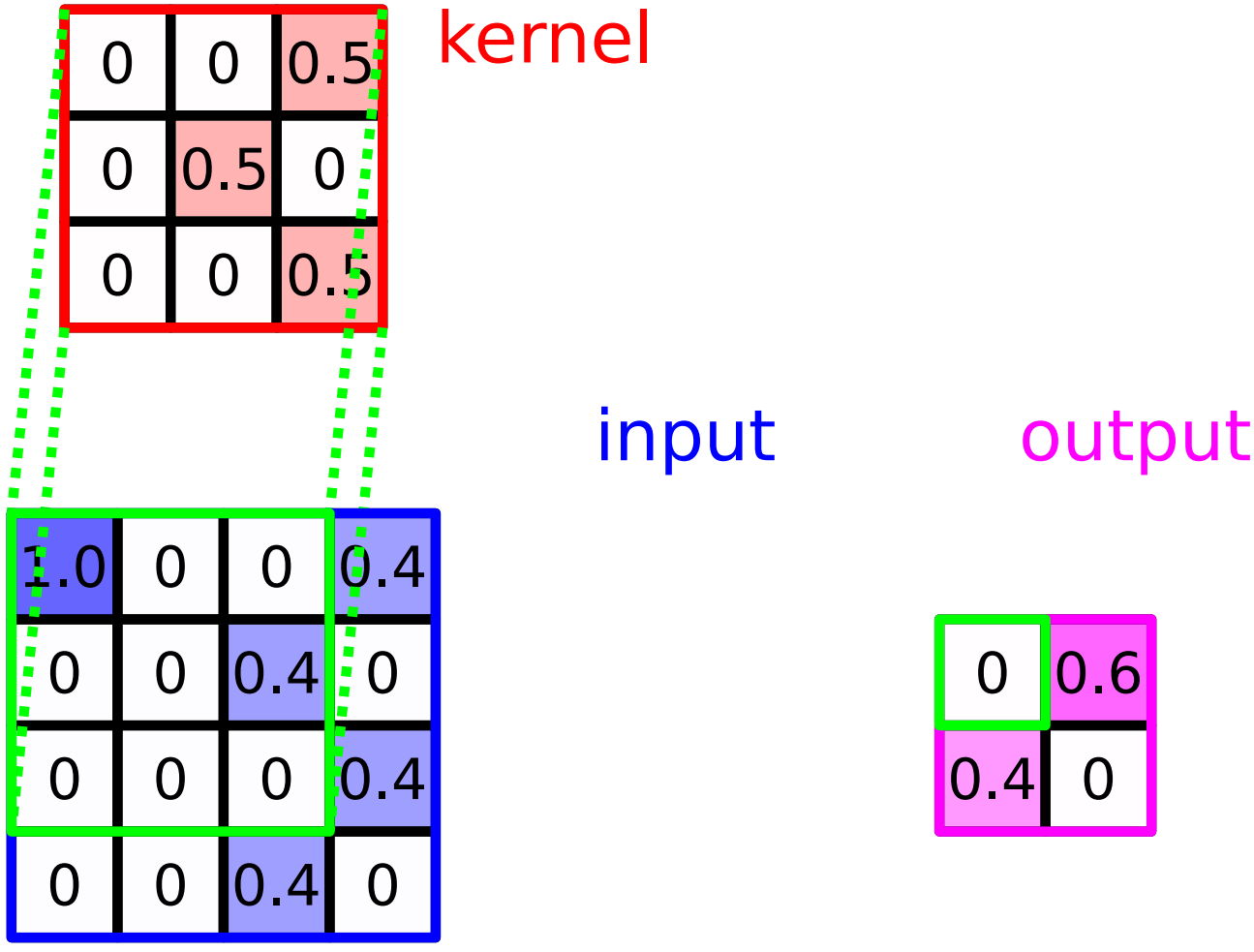


kernel

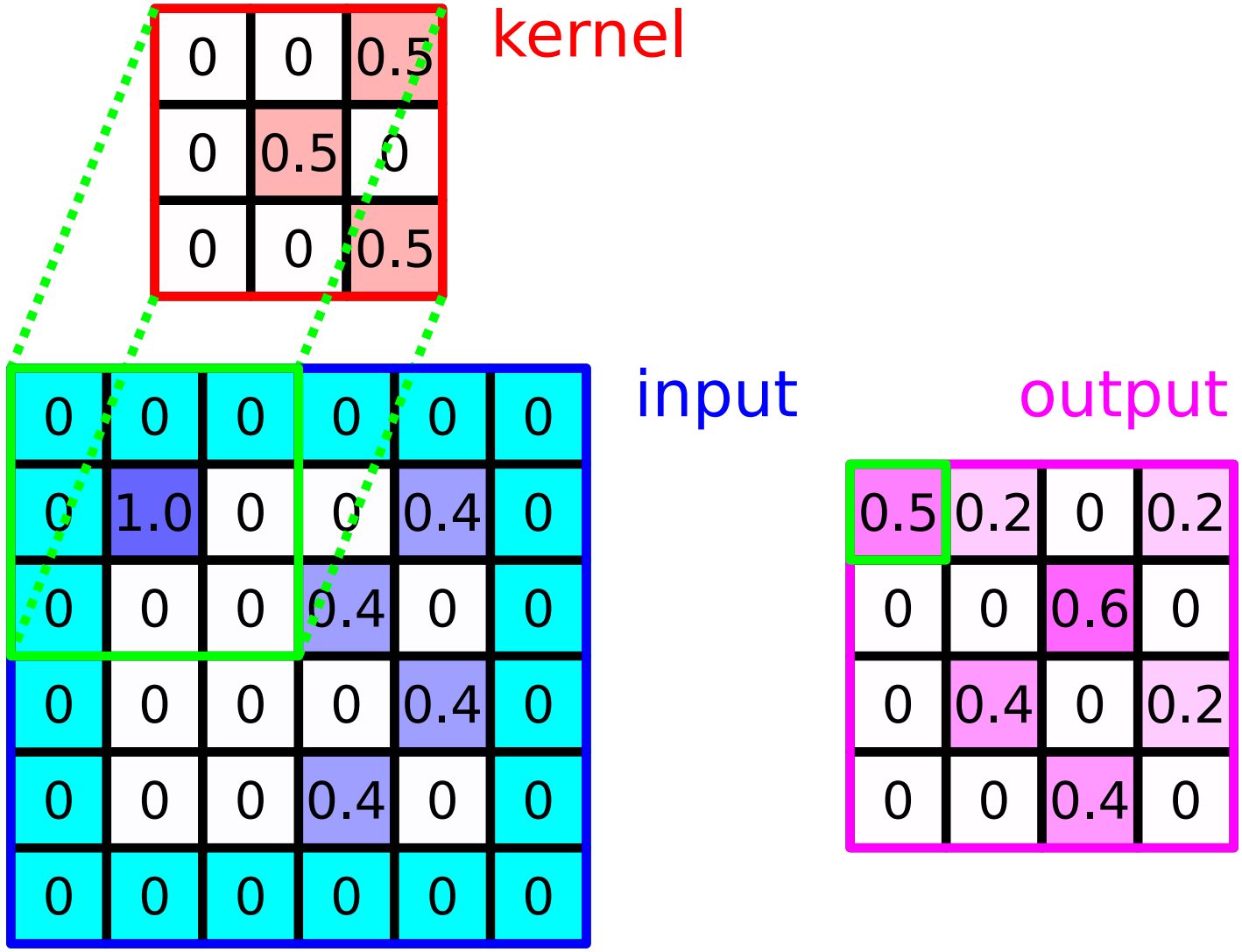
input



Zero-Padding



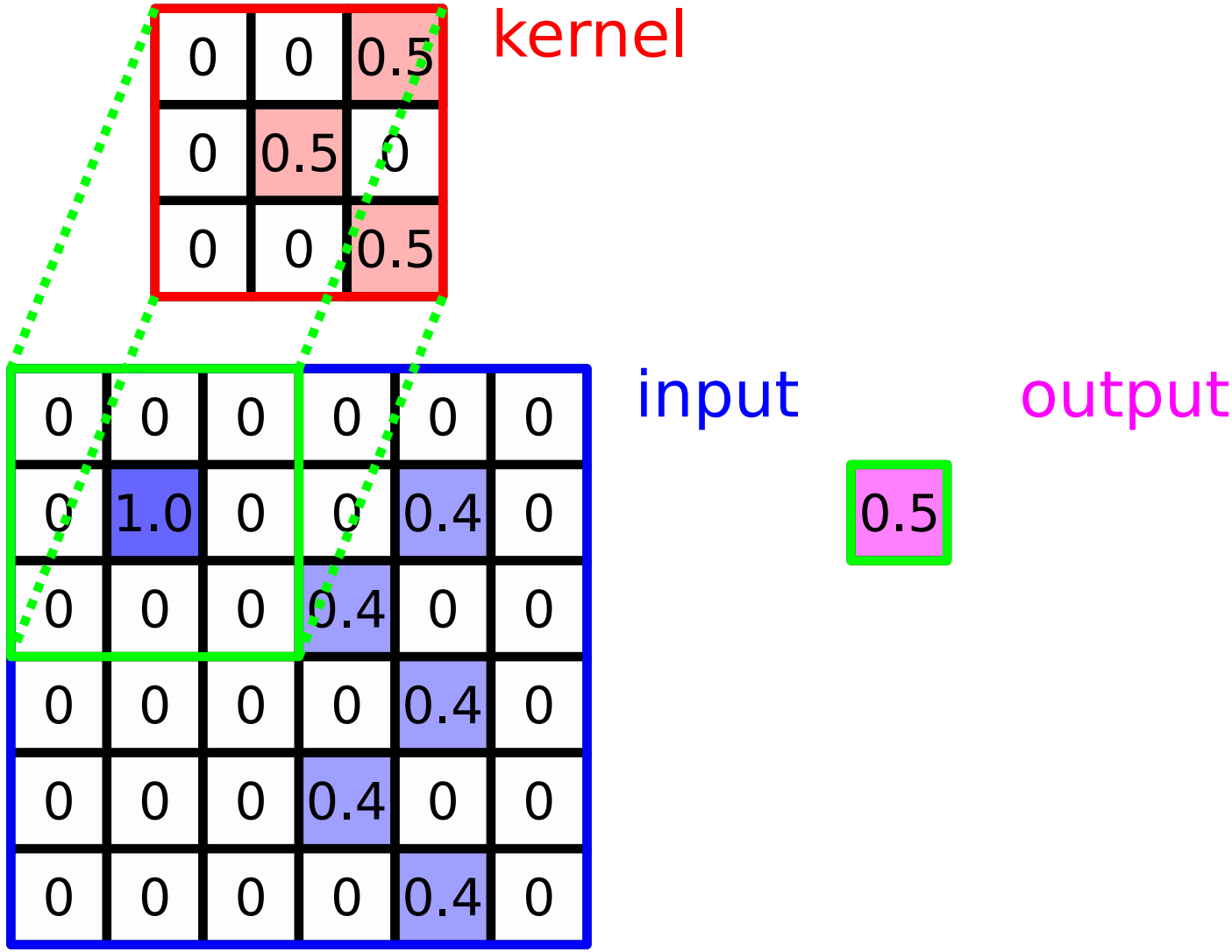
Zero-Padding



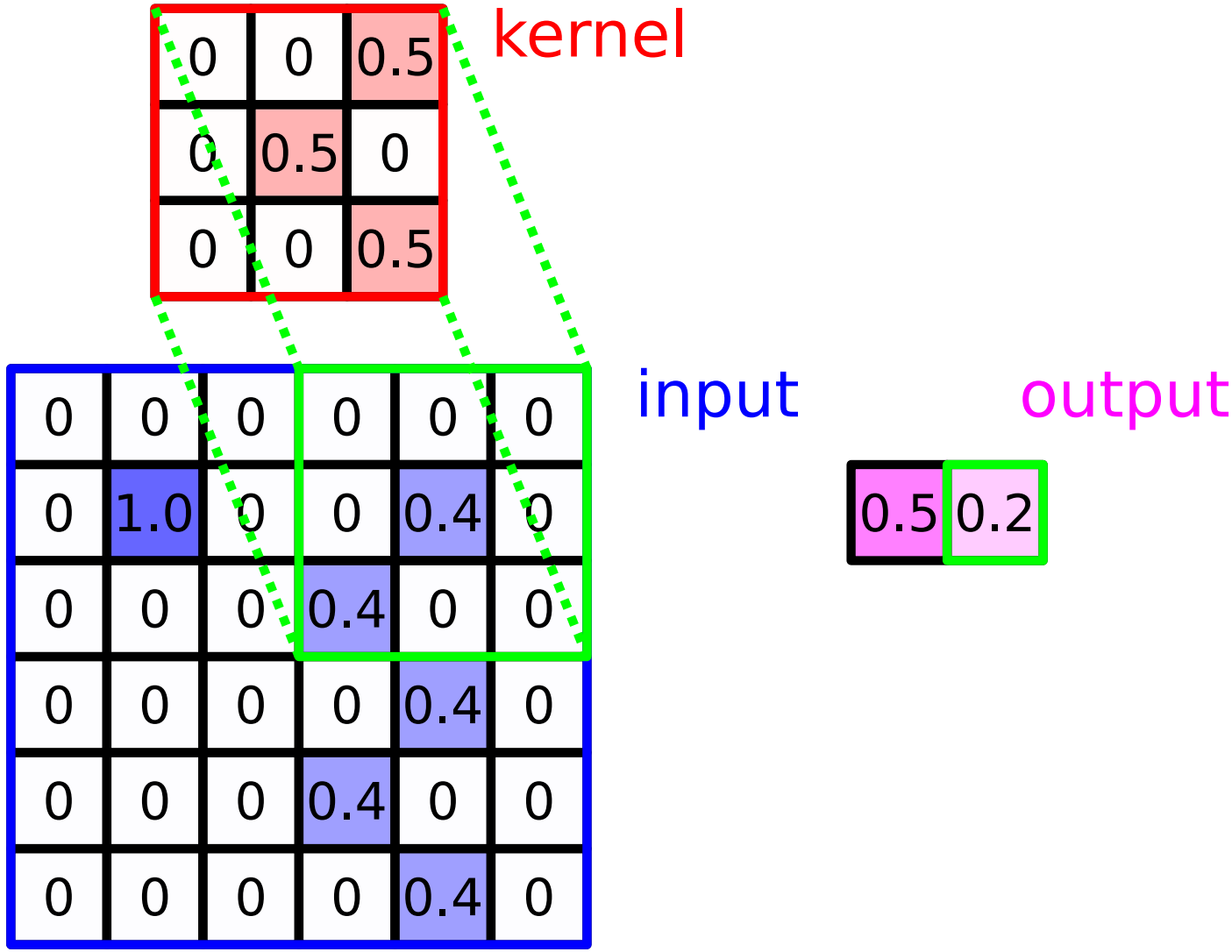
Striding

- Striding controls how much the filters are moved.
- The smaller the stride the more the receptive filters overlap.
- Striding is used for downsampling of images.
- stride > 1 will lead to loss of information.
- stride > 1 will increase receptive field through depth of network.

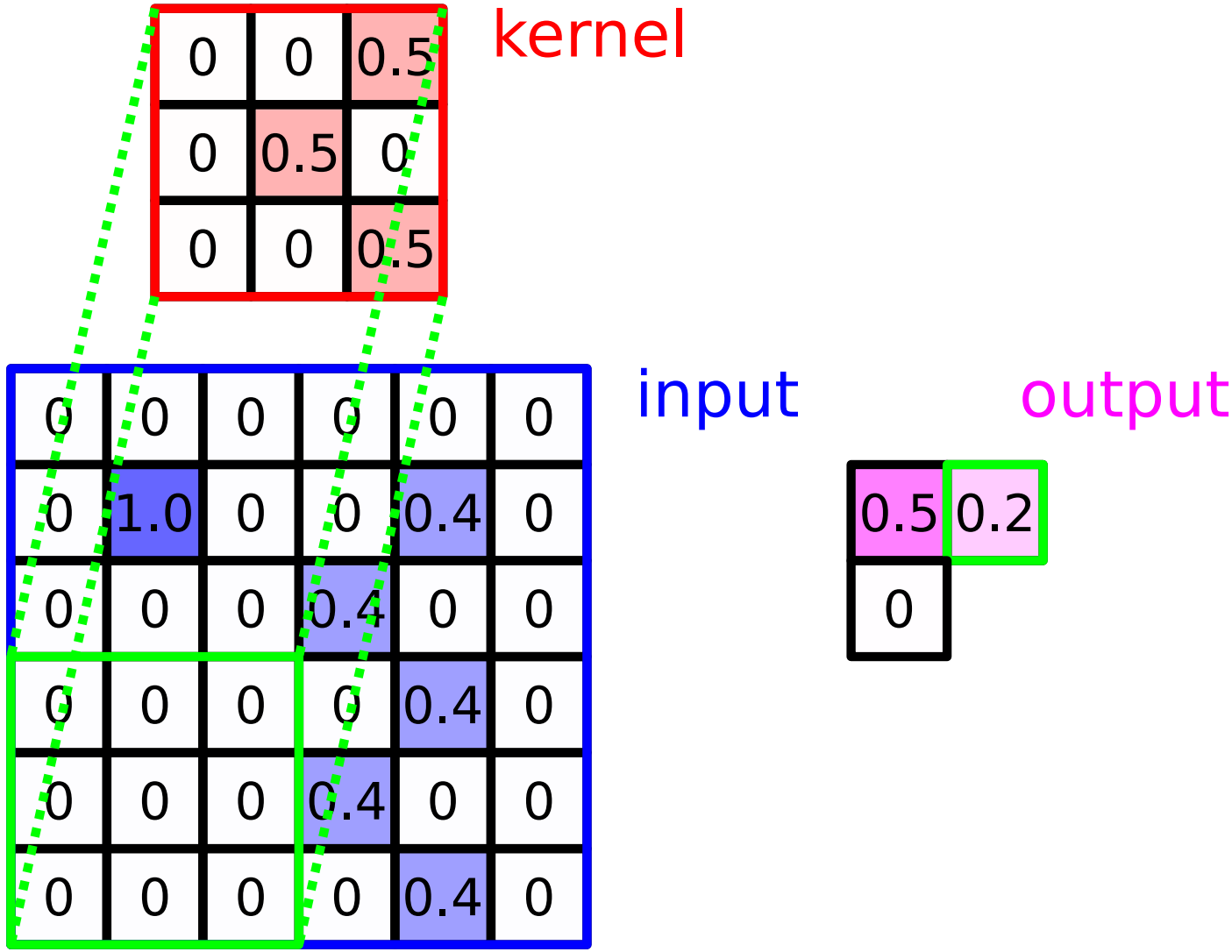
Striding: stride=3



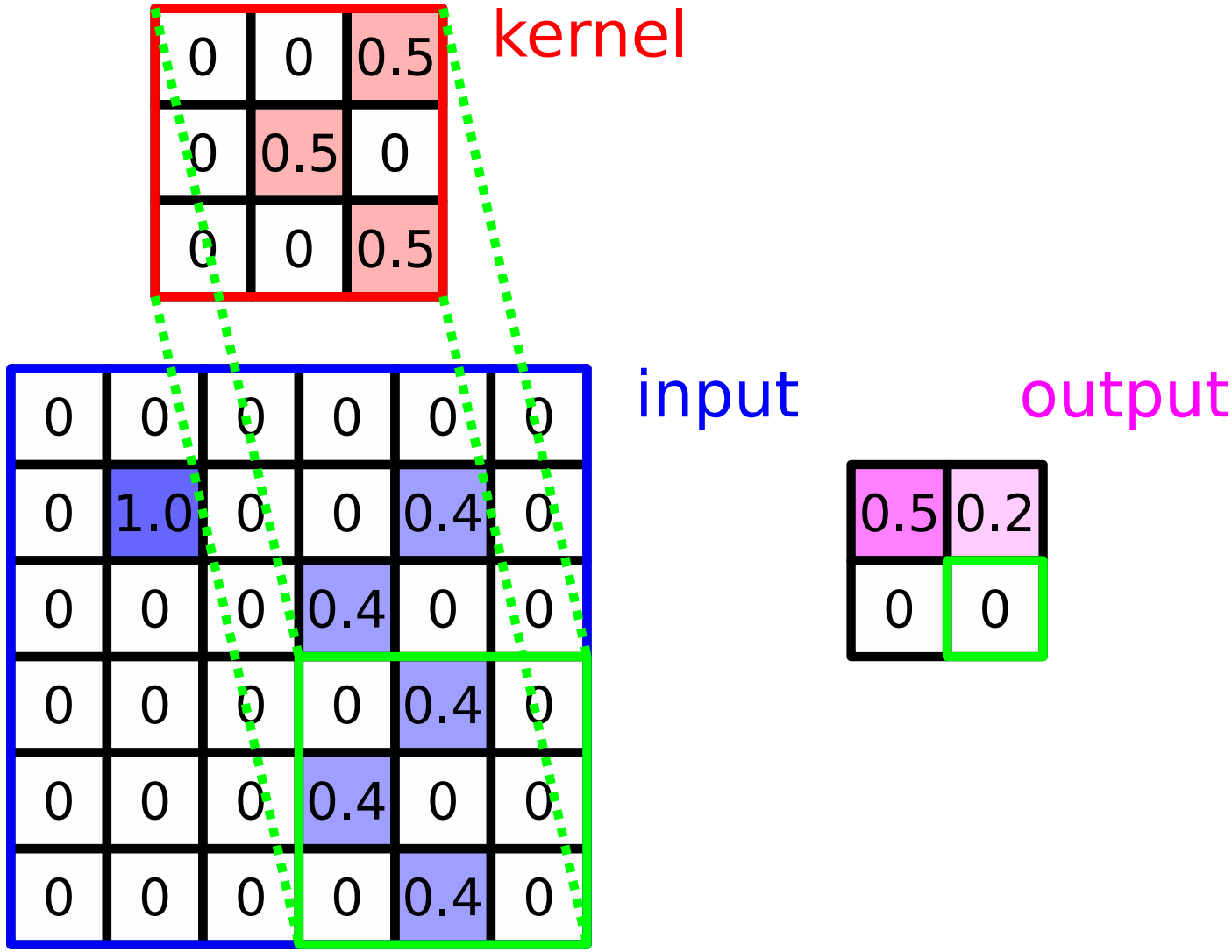
Striding: stride=3



Striding: stride=3



Striding: stride=3



Striding: stride=3

0	0	0.5
0	0.5	0
0	0	0.5

kernel

0	0	0	0	0	0
0	1.0	0	0	0.4	0
0	0	0	0.4	0	0
0	0	0	0	0.4	0
0	0	0	0.4	0	0
0	0	0	0	0.4	0

input

output

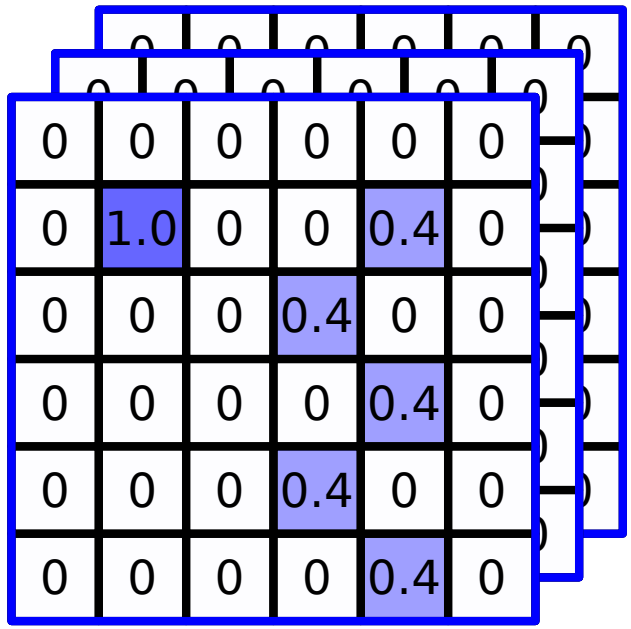
0.5	0.2
0	0

Inputs in CNNs

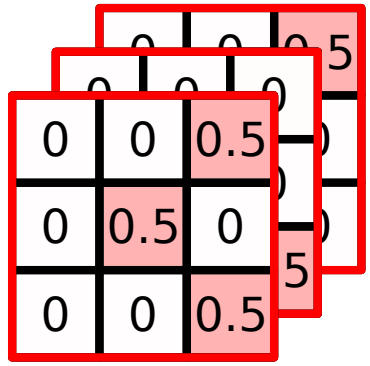
- RGB images have 3 channels for red, green, blue.
- The shape of an RGB image would be (width, height, 3).
- Channels are also called depth or feature maps.
- We need to make sure our kernel matches the number of channels.

Inputs in CNNs

input (3 channels)

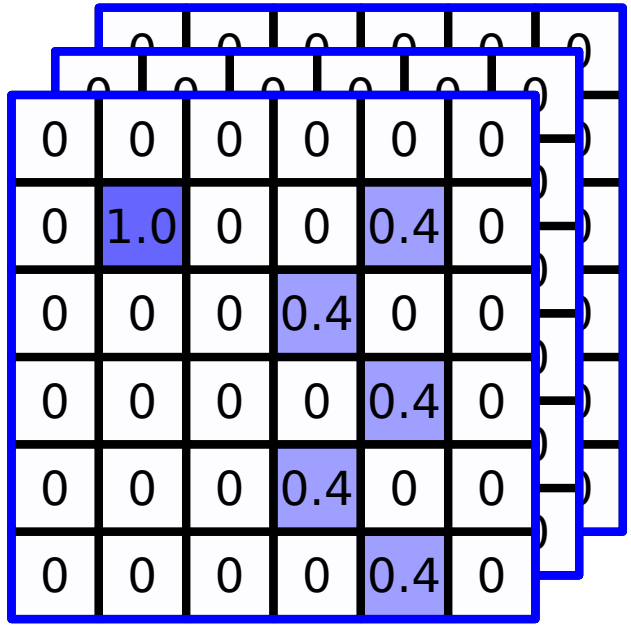


Inputs in CNNs

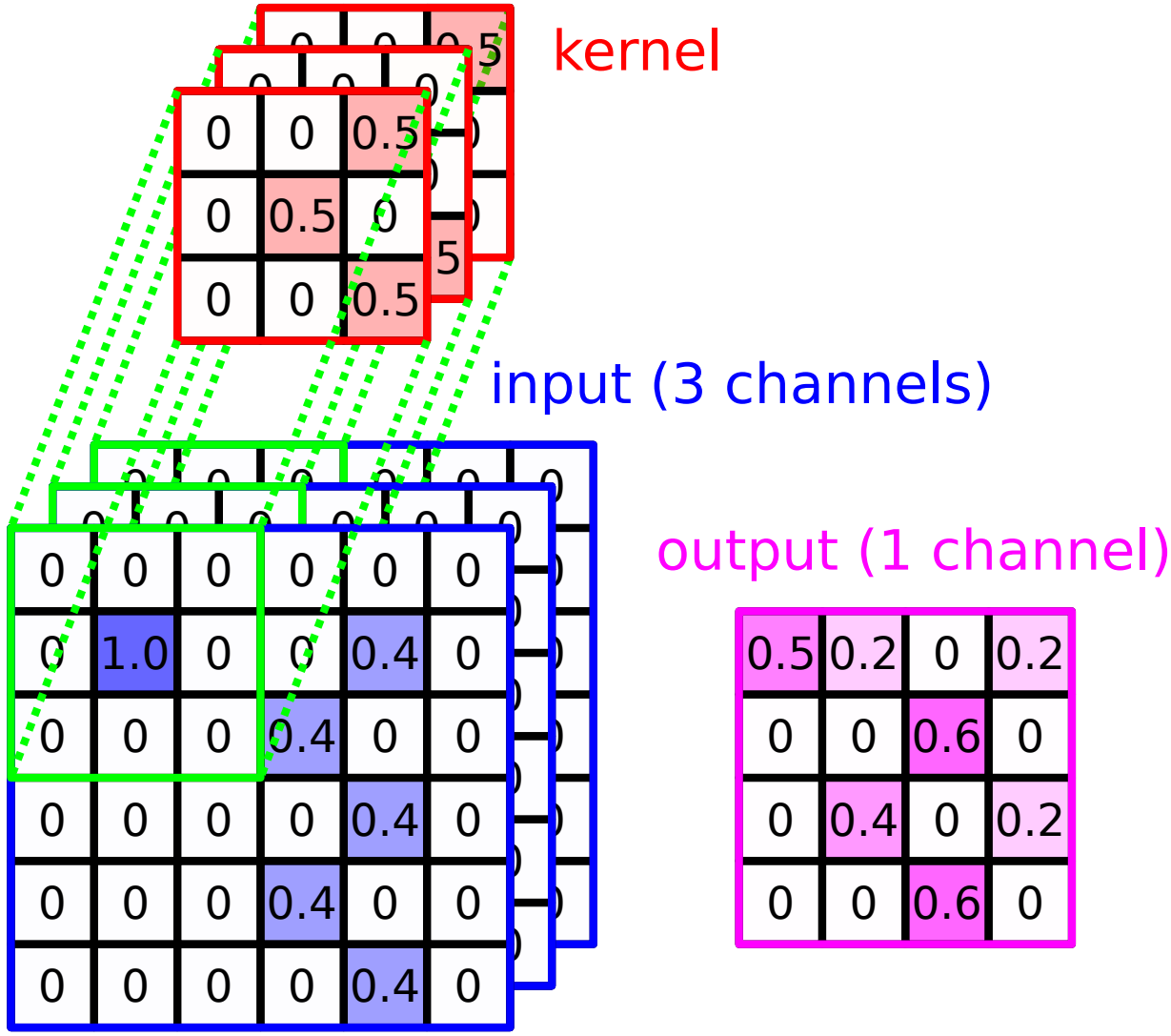


kernel

input (3 channels)



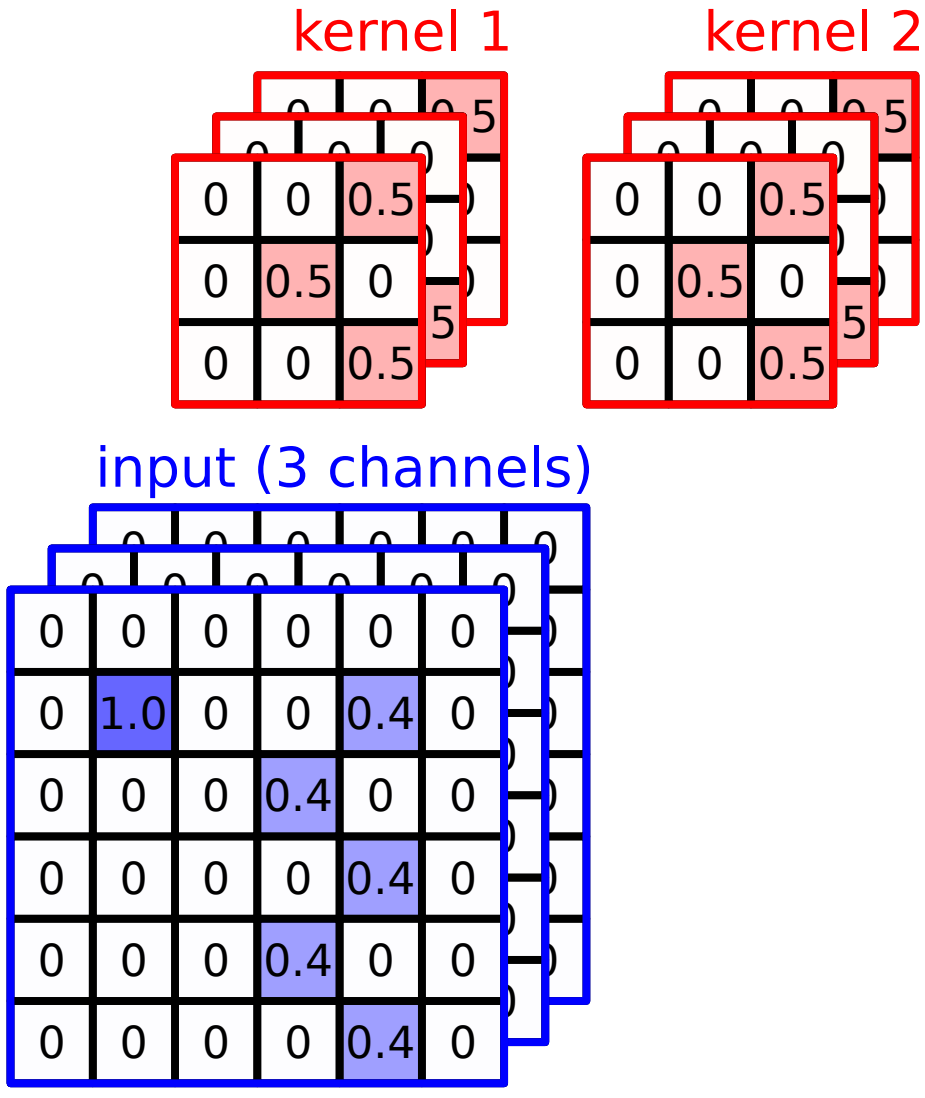
Inputs in CNNs



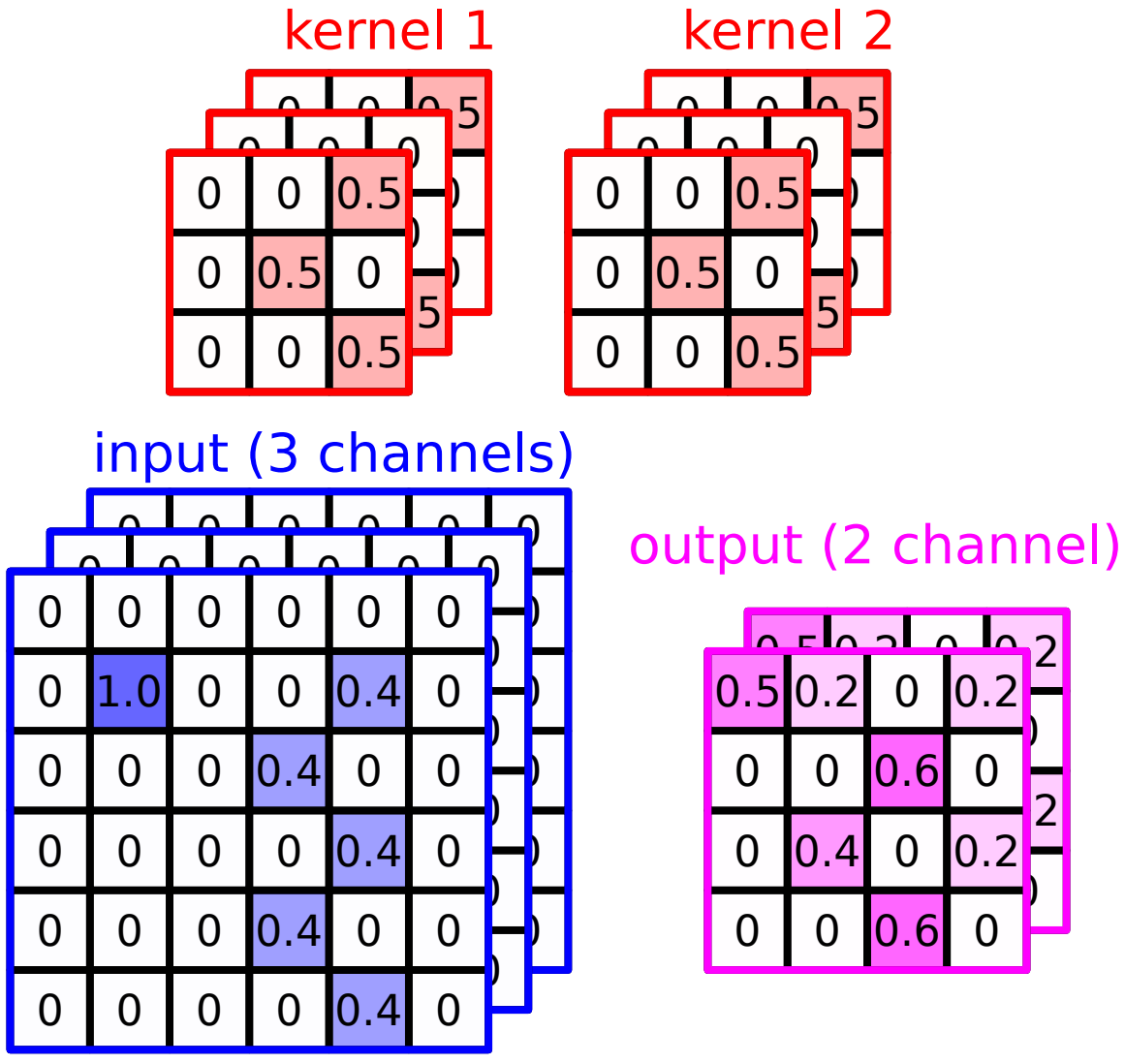
Outputs in CNNs

- We usually want to apply multiple kernels to the image.
- For each kernel, we create a channel in the CNN output.

Outputs in CNNs



Outputs in CNNs



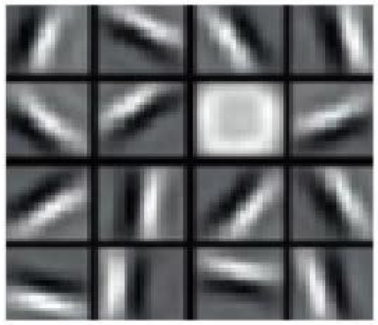
Multiple Levels of Convolutions

A typical CNN architecture has several layers of:

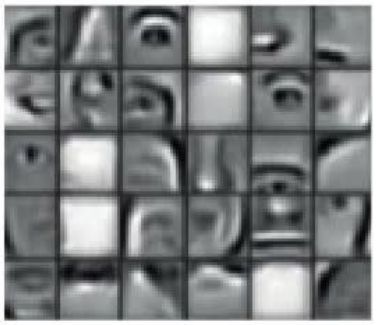
1. Convolution
2. Nonlinearity
3. Pooling (optional)

- Each receptive field produces a new channel/feature map for the next layer.
- Convolution operates on multiple receptive fields and multiple feature maps.
- The complexity of detected features tends to increase layer by layer (first feature map does edge detection, later ones combine it to complex shapes).

Multiple Levels of Convolutions



1. EDGES



2. FEATURES

3. FACES

4. FULL FACE

DeepFace: Facial recognition system used by Facebook since 2015

[Source https://www.simplilearn.com/deep-learning-tutorial]

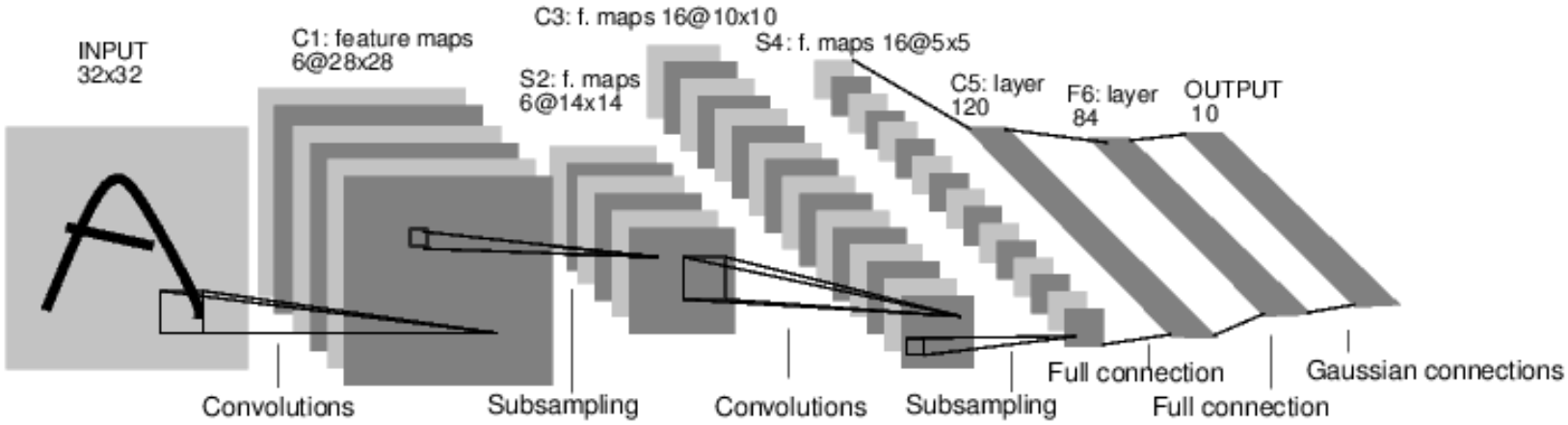
Pytorch implementation CNNs

```
# CNN architecture (two conv layers followed by two fully connected layers)
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0)
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        #Size changes from (1, 28, 28) to (20, 24, 24)
        x = F.relu(self.conv1(x))
        #Size changes from (20, 24, 24) to (20, 12, 12)
        x = F.max_pool2d(x, 2, 2)
        #Size changes from (20, 12, 12) to (50, 8, 8)
        x = F.relu(self.conv2(x))
        #Size changes from (50, 8, 8) to (50, 4, 4)
        x = F.max_pool2d(x, 2, 2)
        #Size changes from (50, 4, 4) to (50*4*4)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

Example CNN Architectures

LeNet



Gradient-based learning applied to document recognition, LeCun, Bottou, Bengio, Haffner, Proceedings of the IEEE,1998

- First famous CNN
- Basic design still valid today

AlexNet

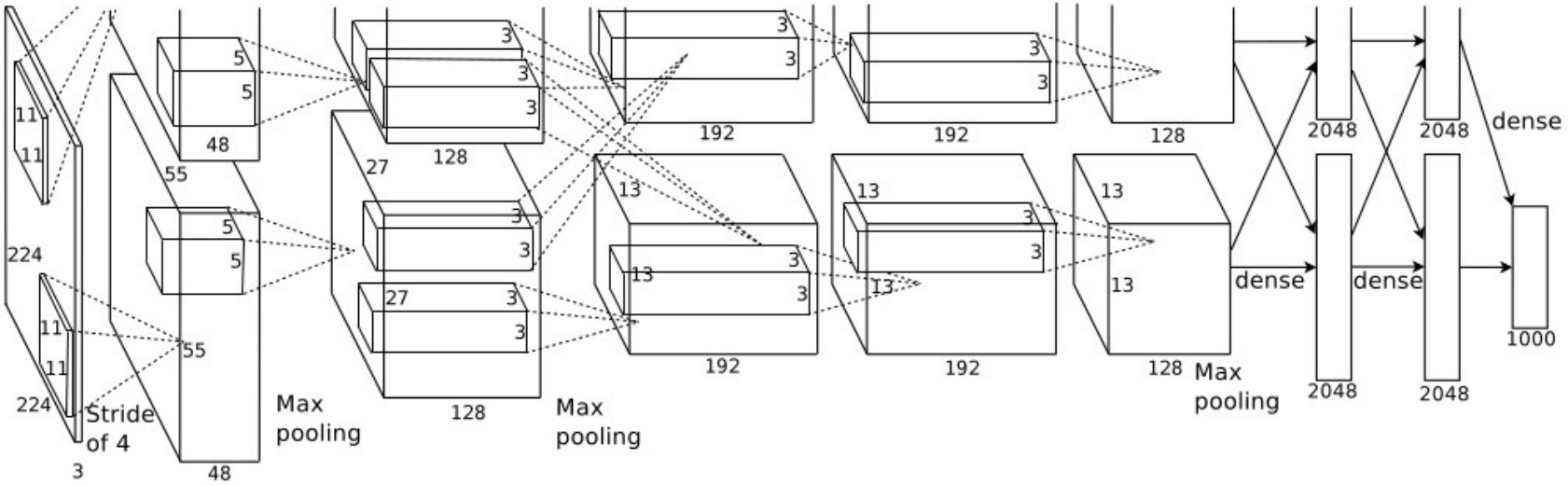


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

- Won ILSVRC 2012 (using ReLUs + Dropout + GPUs) by a landslide
- After Kriezhevsky et al. won ILSVRC 2012, "everyone" started using CNNs for image tasks.

ImageNet Classification with Deep Convolutional Neural Networks, Kriezhevsky, Sutskever, Hinton, NIPS 2012

ResNet

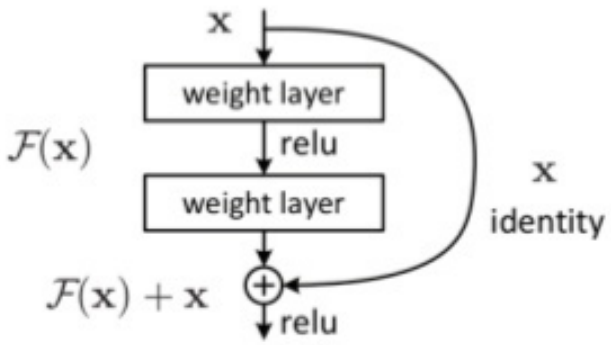


Figure 2. Residual learning: a building block.

- One of the current standard architectures
- Up to 1k layers

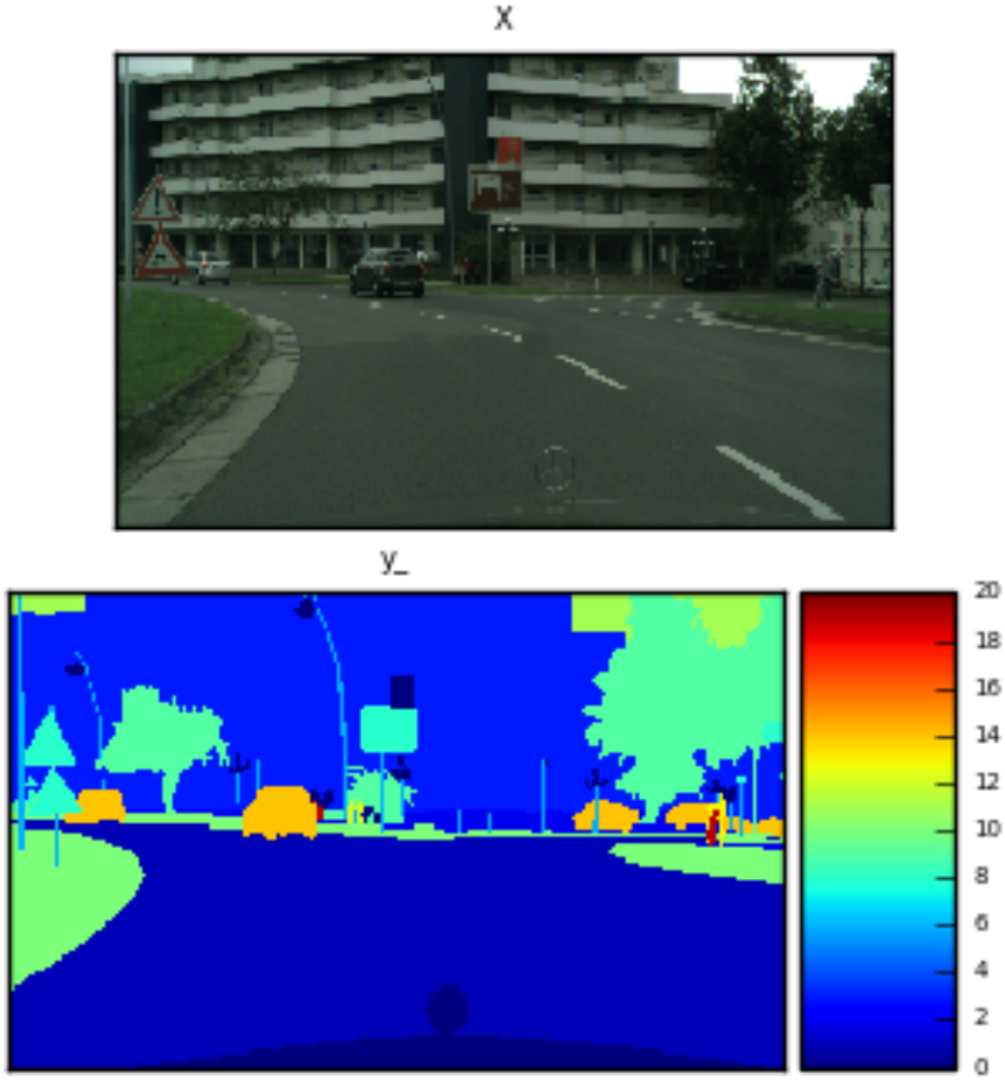
Deep Residual Learning for Image Recognition, He et al., CVPR 2016

CNN Hints and Applications

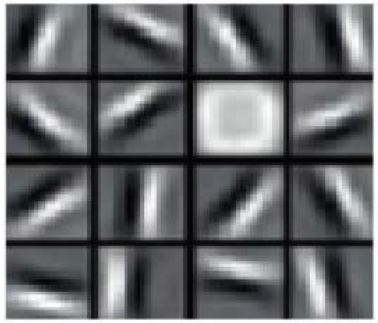
Common tricks for image classification

- Dataset augmentation: Create new images from existing ones
  - Mirroring
  - Translation/Rotation
  - Scaling
  - Color-channel adaptations
  - Image distortions
  - ...
- Use pretrained nets (Transfer Learning)
- Be aware of overfitting to background/colors/artifacts
- Use striding/pooling to reduce dimensionalty and increase number of filters

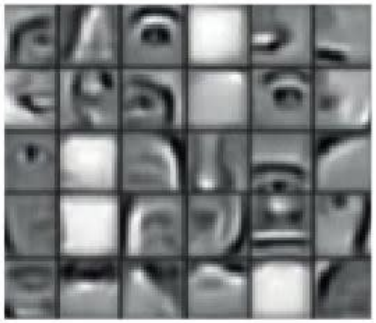
Semantic Segmentation: Self-Driving Cars



Facial Recognition



1. EDGES



2. FEATURES

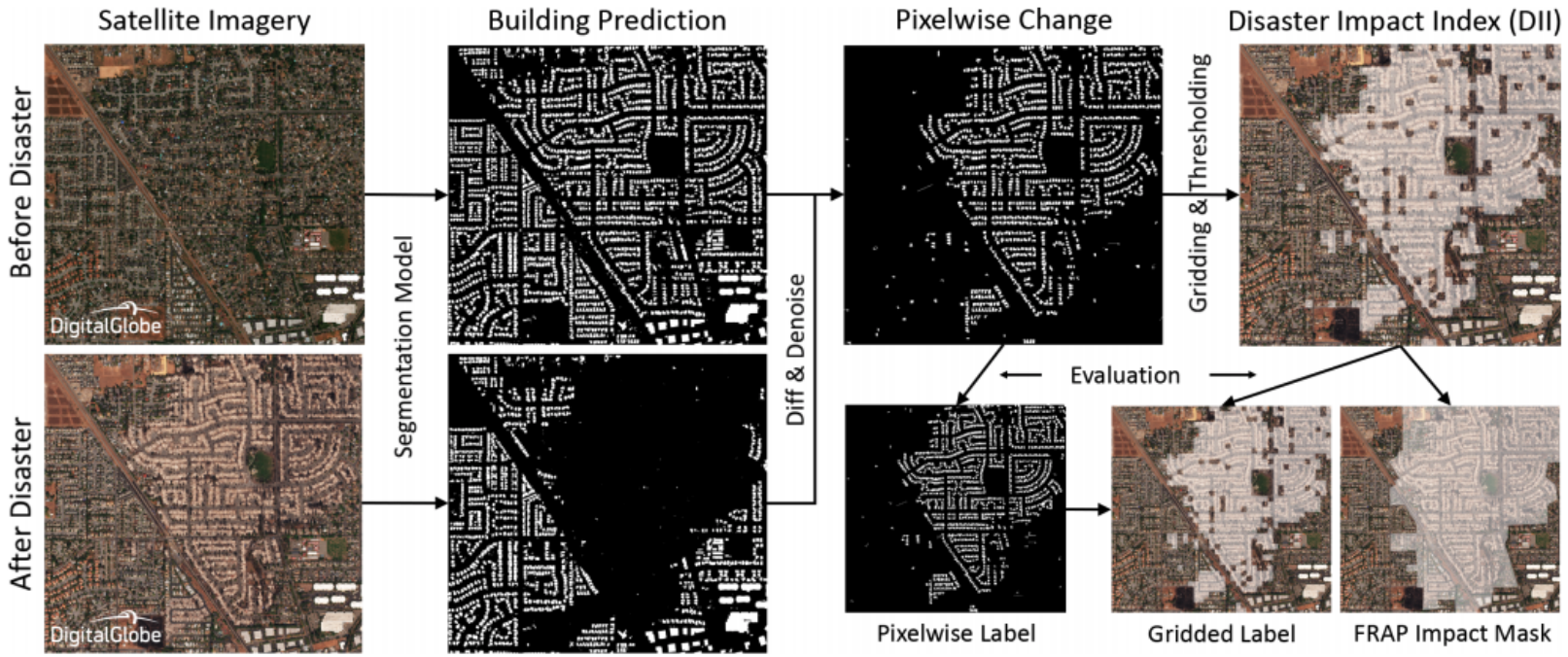
3. FACES

4. FULL FACE

DeepFace: Facial recognition system used by Facebook since 2015

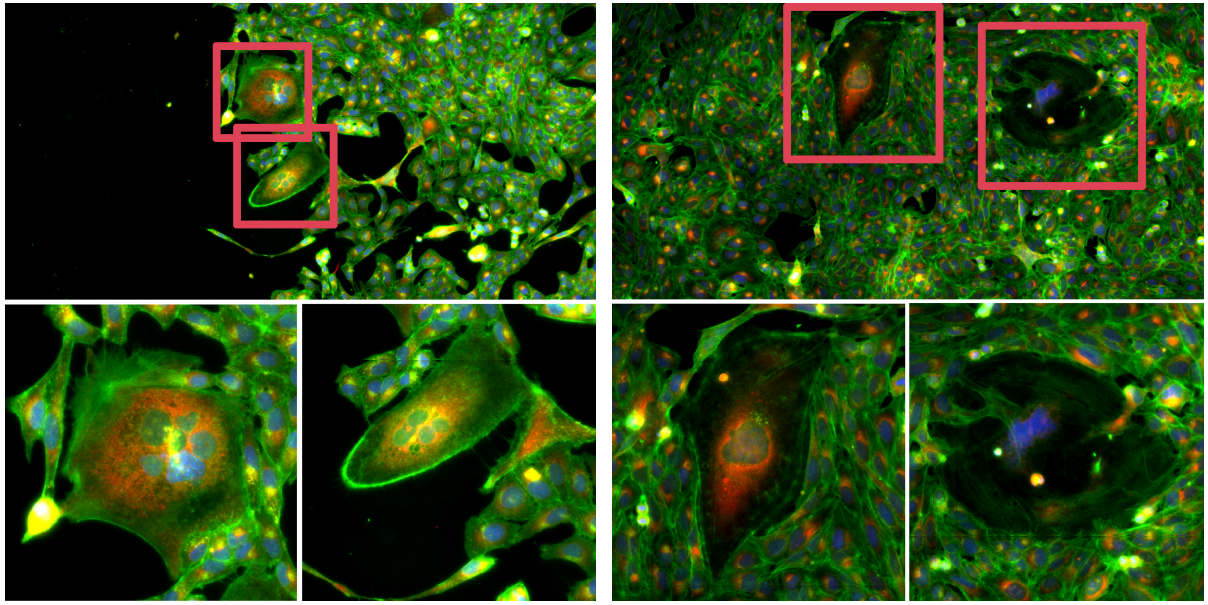
[Source https://www.simplilearn.com/deep-learning-tutorial]

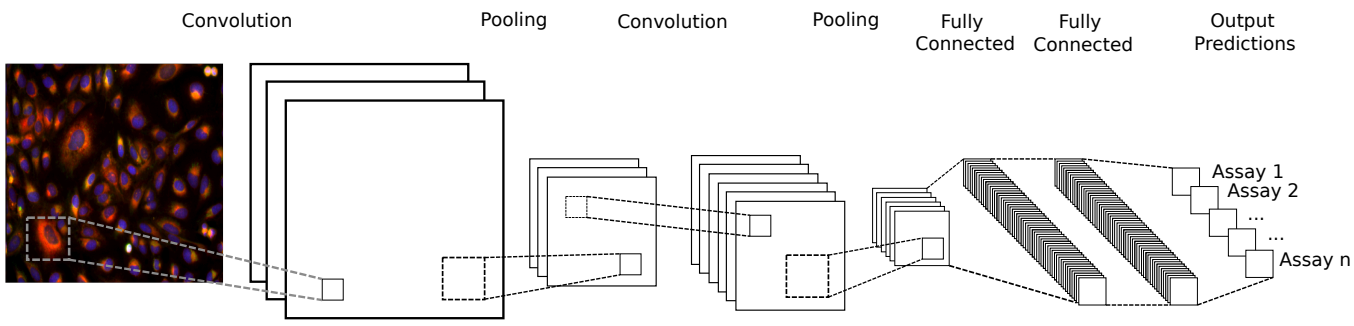
Semantic Segmentation: Satellite Images



Damage estimate after natural disasters [Source: Facebook/CrowdAI]

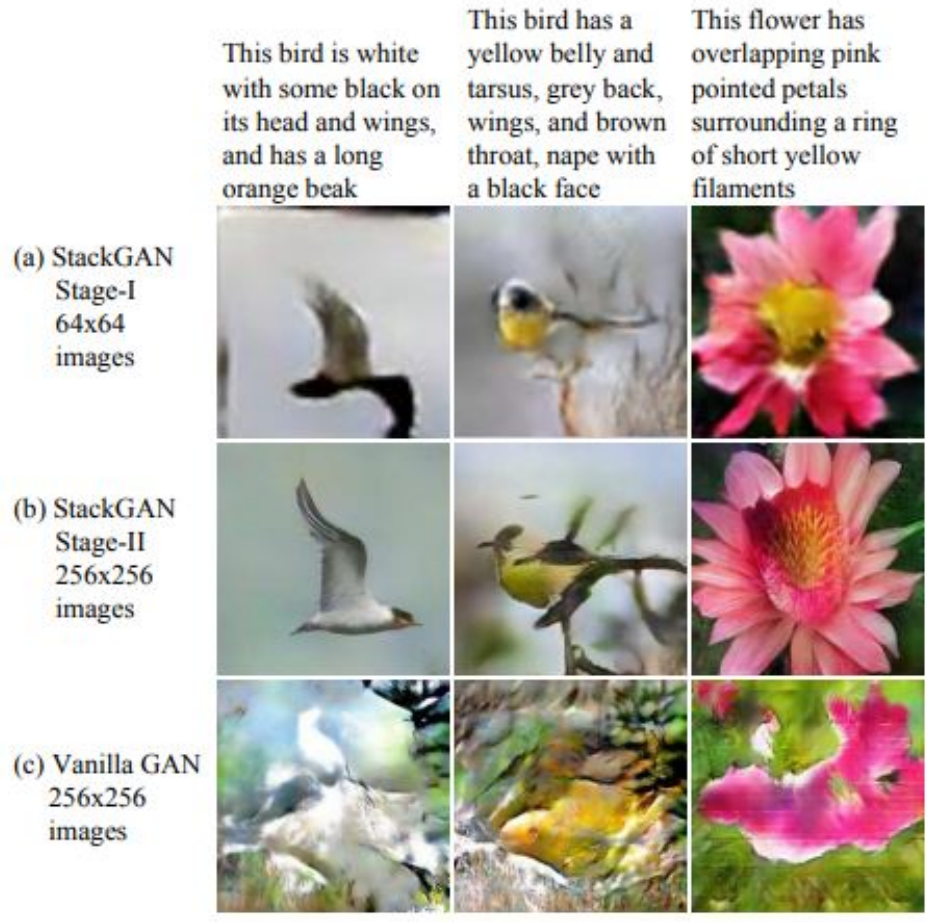
High-Resolution Image Classification





[Source: Accurate prediction of biological assays with high-throughput microscopy images and convolutional networks, Hofmarcher & Rumetshofer & Clevert & Hochreiter & Klambauer, JCIM 2019]

Image Generation (GANs)



[Source: StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks, Zhang & Xu & Li & Zhang & Huang & Wang & Metaxas, ICCV 2017]

Summary

Summary

- State of the Art in all kinds of vision tasks
- Network is convolved over image
  - Image is divided into sub-images and processed by network with shared weights
- Spatial dimensionality can be reduced by striding or pooling
- Very large networks with many tweaks
- Dataset augmentation can be crucial