ML/DL for Everyone with PYTORCH

Lecture 10: Basic CNN

Sung Kim <hunkim+ml@gmail.com> HKUST
Code: https://github.com/hunkim/PyTorchZeroToAll
Slides: http://bit.ly/PyTorchZeroAll
Videos: http://bit.ly/PyTorchVideo

Call for Comments

Please feel free to add comments directly on these slides.

Other slides: http://bit.ly/PyTorchZeroAll

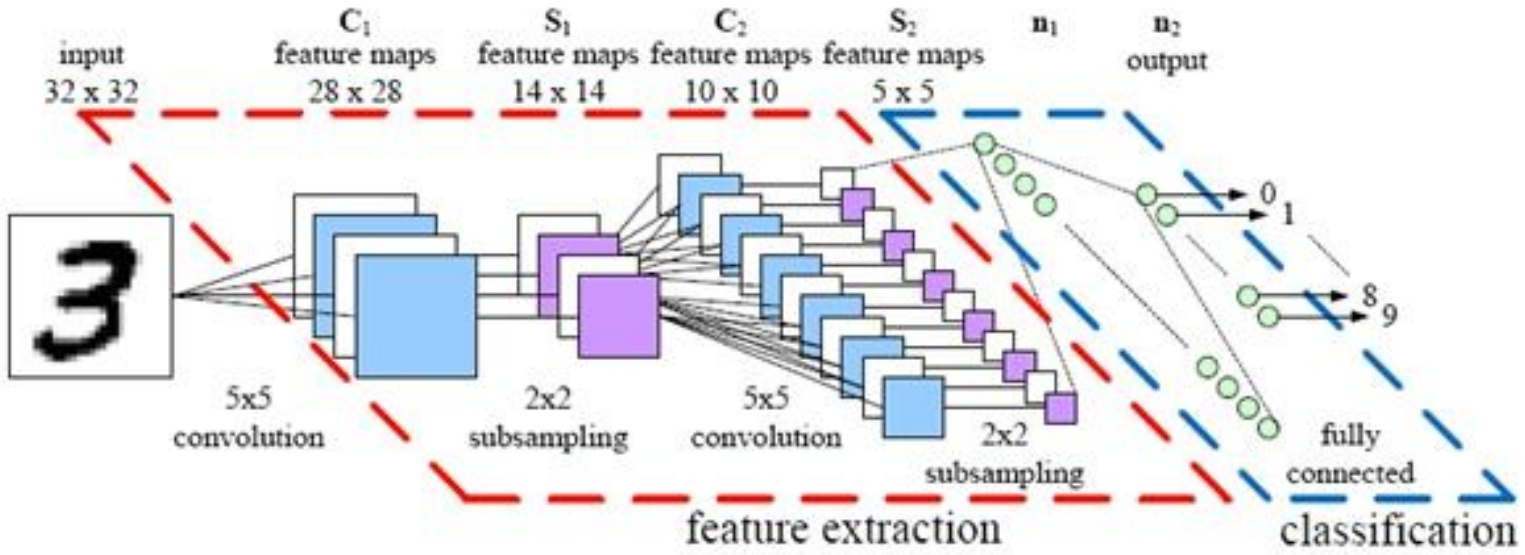
Picture from http://www.tssablog.org/archives/3280

ML/DL for Everyone with PYTORCH

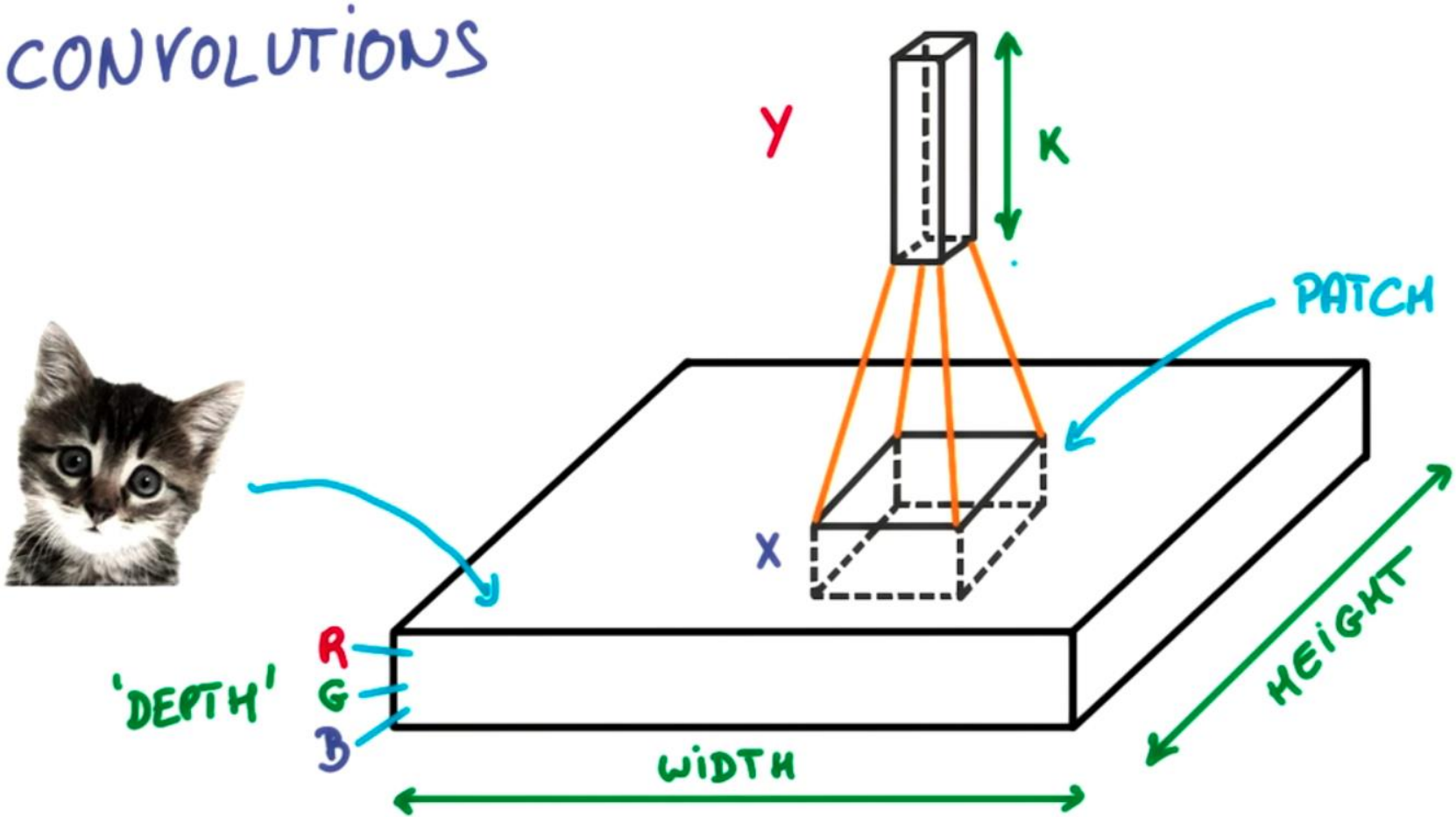
Lecture 10: Basic CNN

Sung Kim <hunkim+ml@gmail.com> HKUST
Code: https://github.com/hunkim/PyTorchZeroToAll
Slides: http://bit.ly/PyTorchZeroAll
Videos: http://bit.ly/PyTorchVideo

CNN

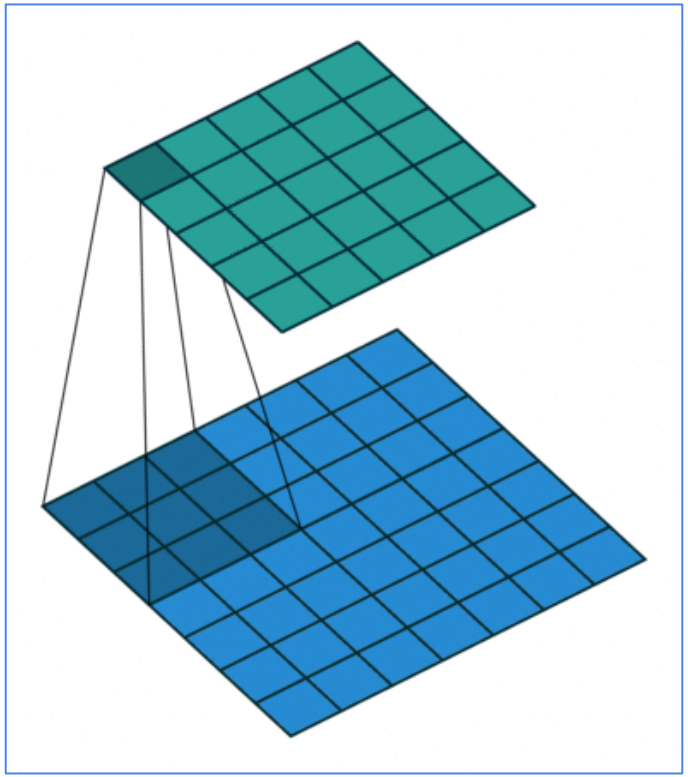


http://parse.ele.tue.nl/cluster/2/CNNArchitecture.jpg



https://ireneli.eu/2016/02/03/deep-learning-05-talk-about-convolutional-neural-network%EF%BC%88cnn%EF%BC%89/

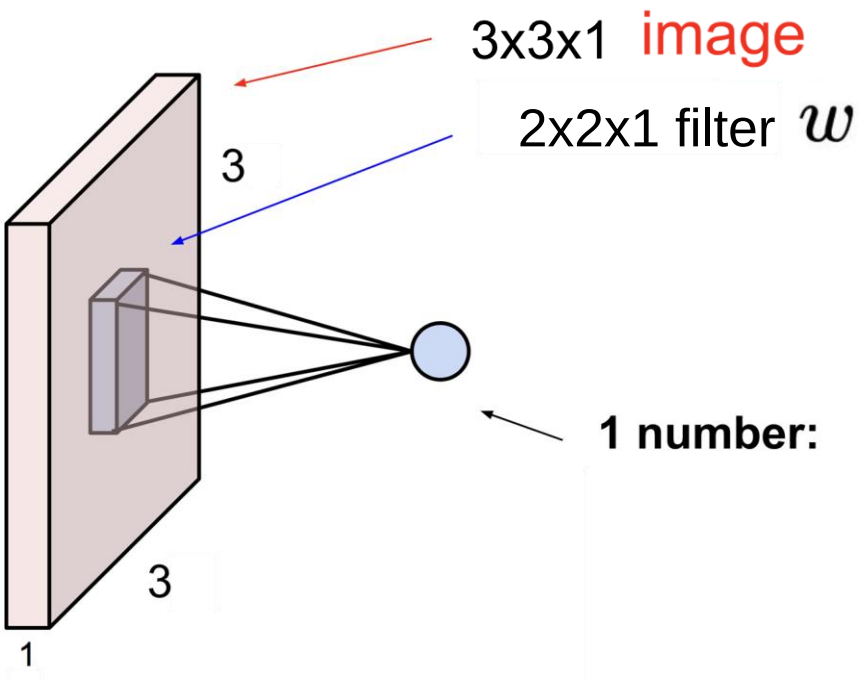
Convolution in Action

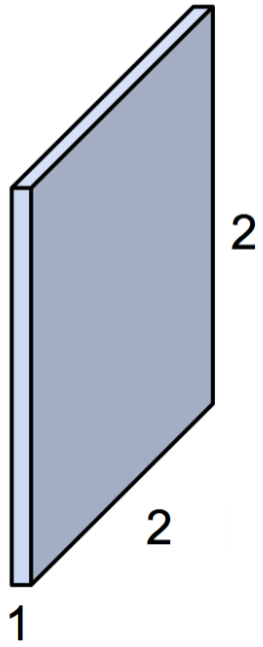


https://github.com/vdumoulin/conv_arithmetic

Simple convolution layer

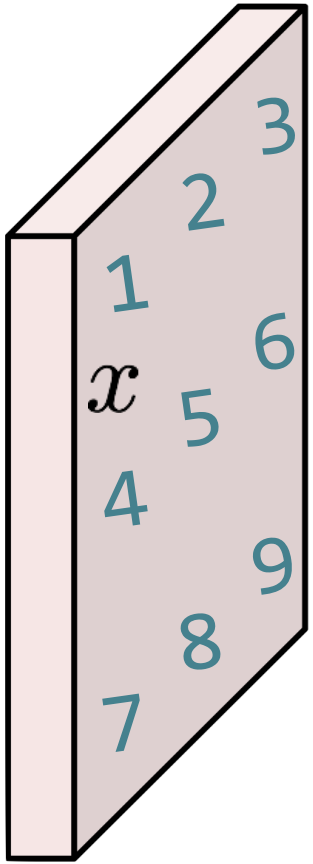
`Stride: 1x1`

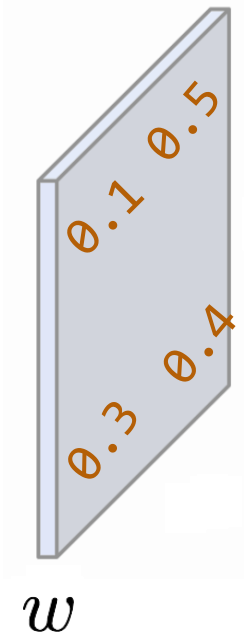


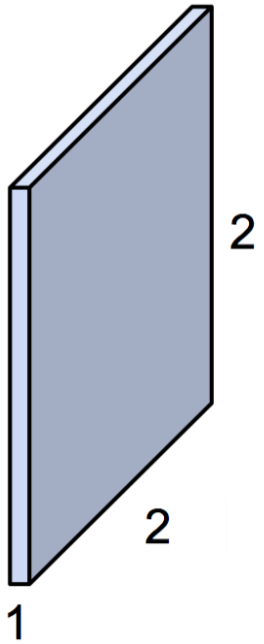


Simple convolution layer

Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, No Padding

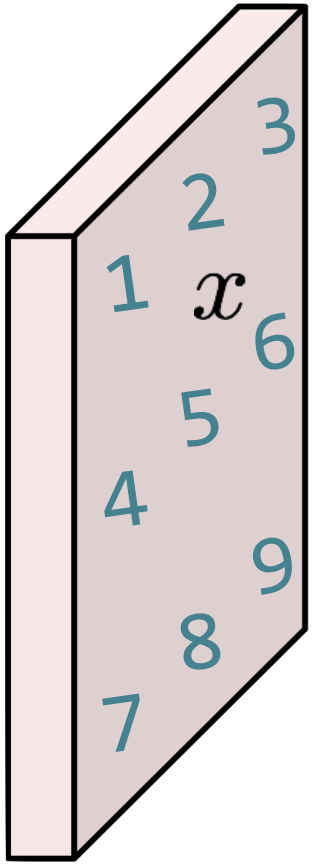


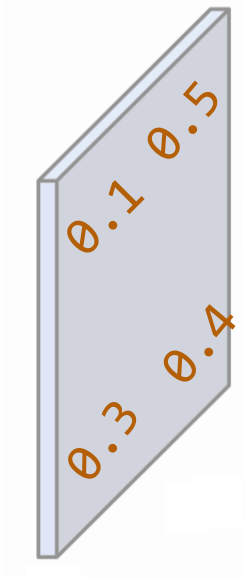


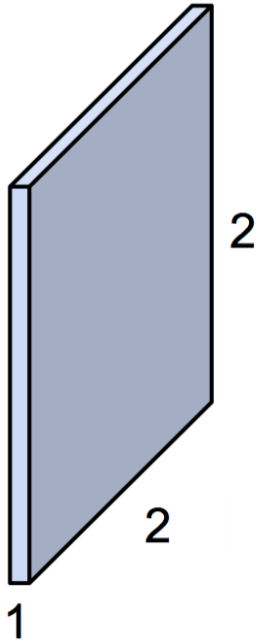


Simple convolution layer

Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, No Padding

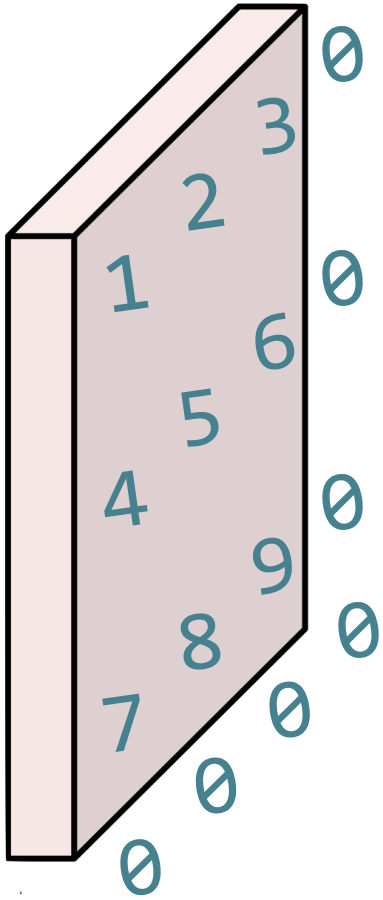


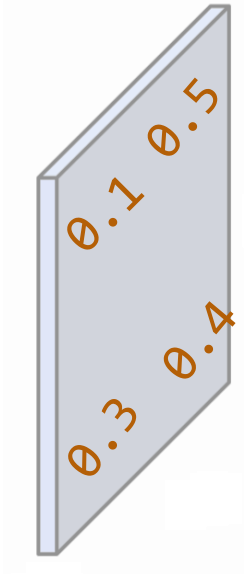


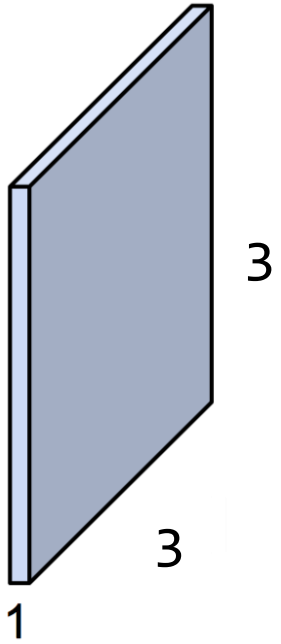


Simple convolution layer

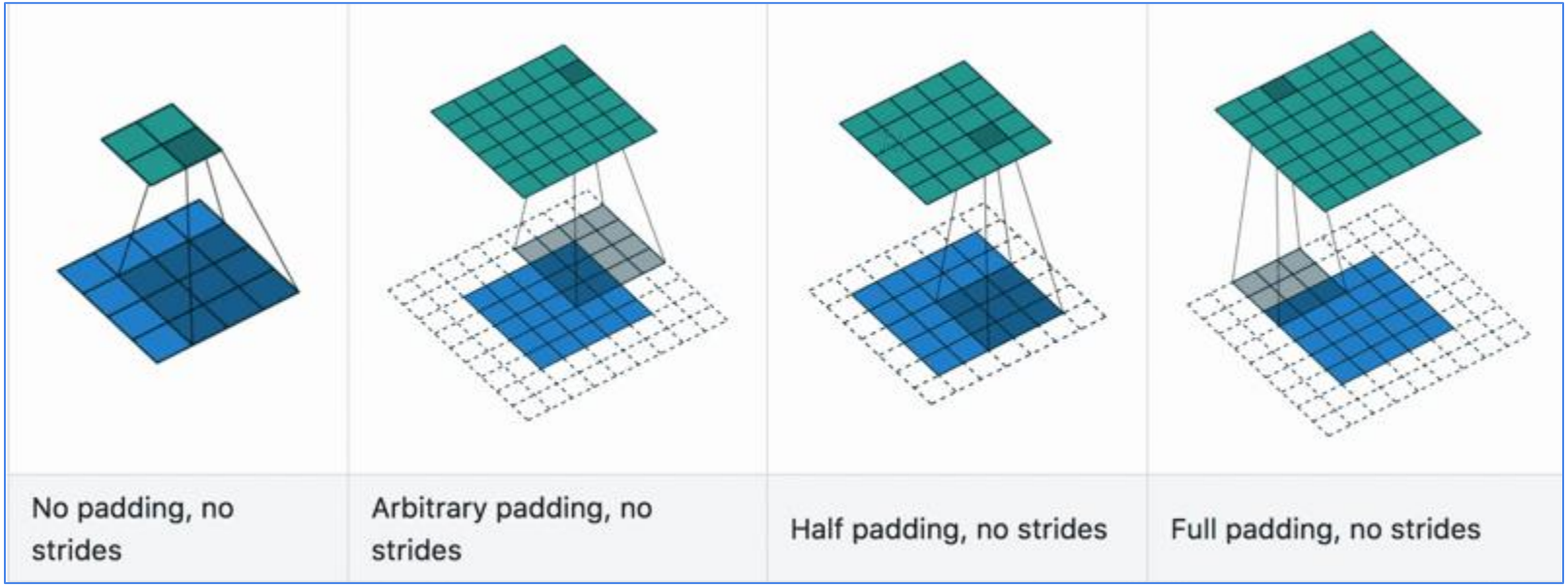
Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, With padding







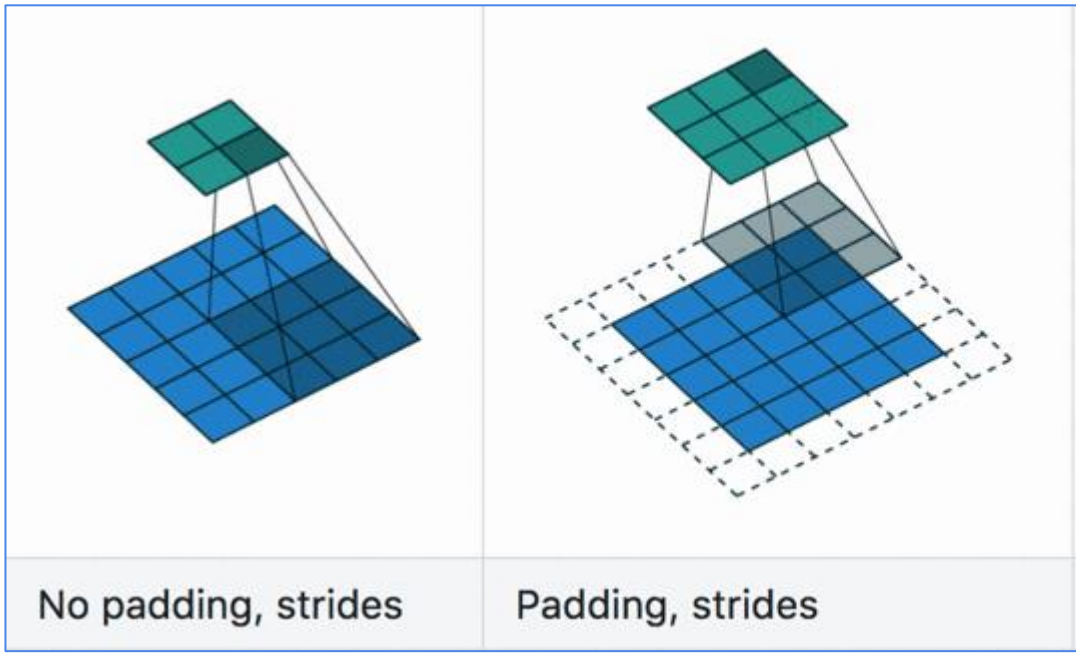
Convolution with padding in Action



No padding, no strides	Arbitrary padding, no strides	Half padding, no strides	Full padding, no strides

https://github.com/vdumoulin/conv_arithmetic

Convolution with stride in Action

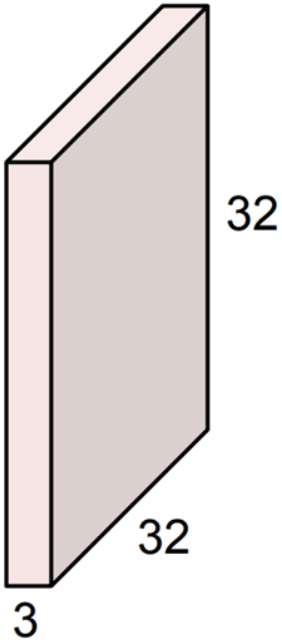


https://github.com/vdumoulin/conv_arithmetic

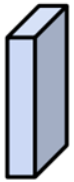
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

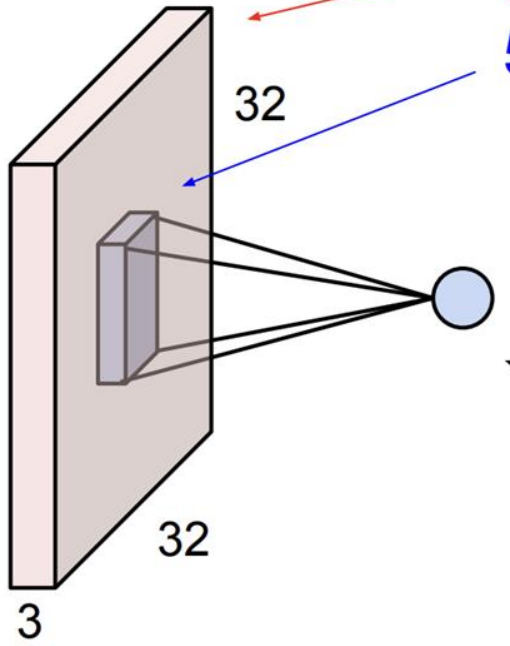


5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

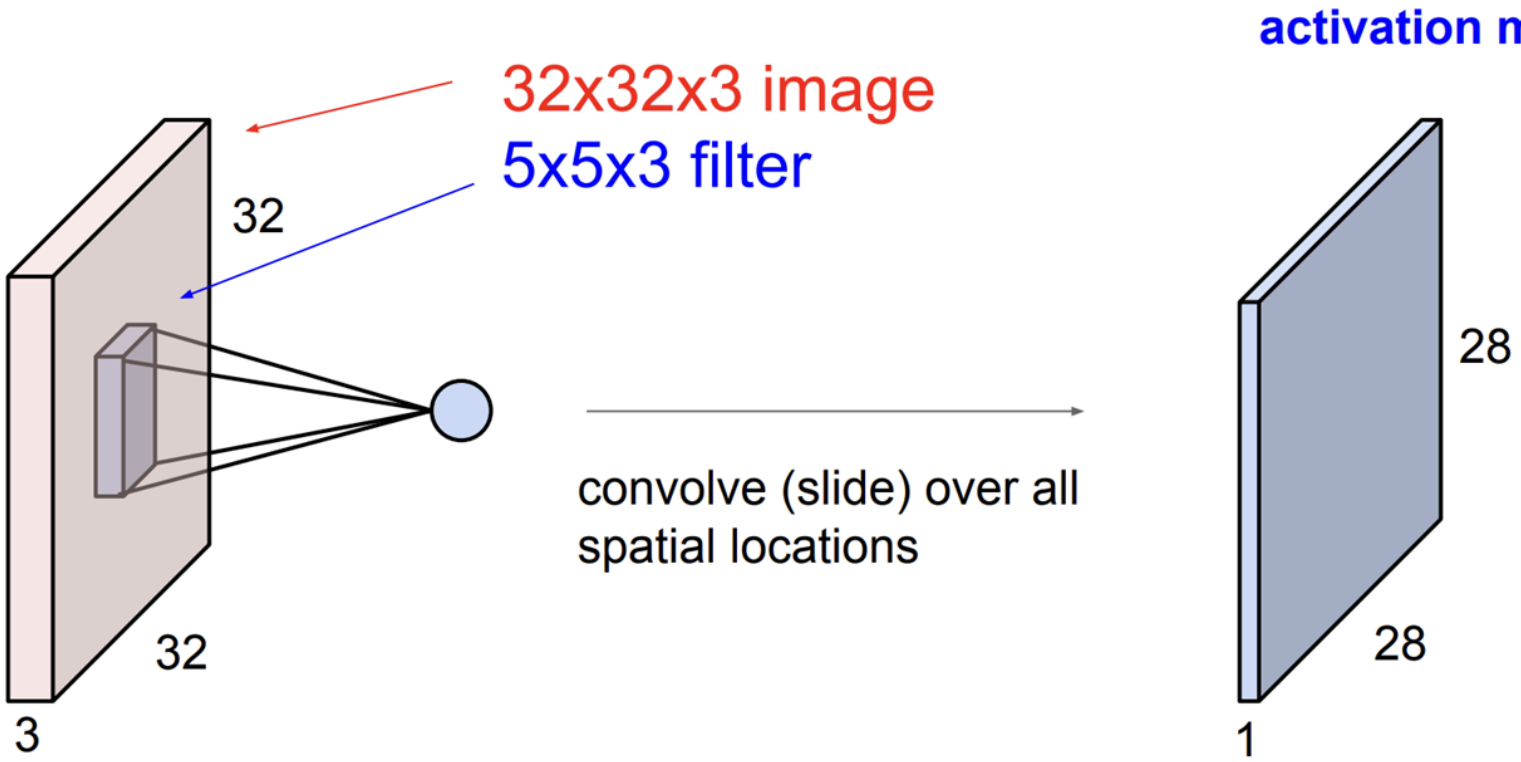
Convolution Layer



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

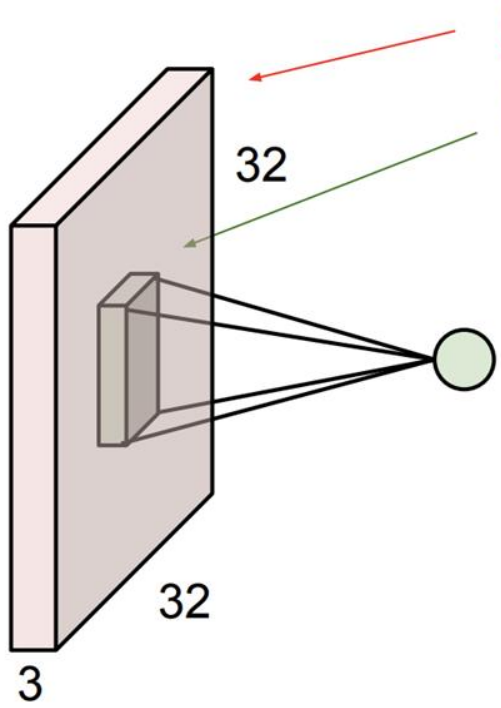
$$w^T x + b$$

Convolution Layer

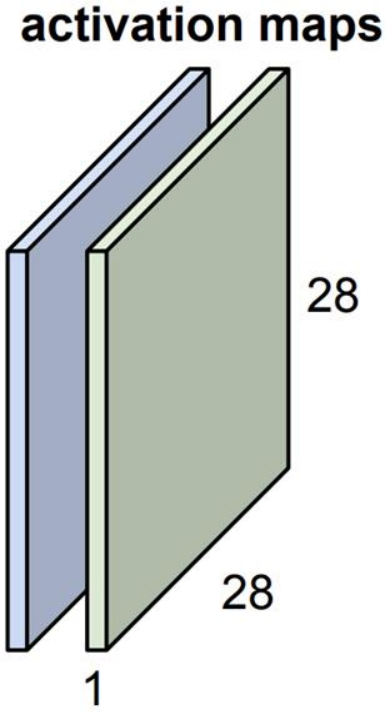


Convolution Layer

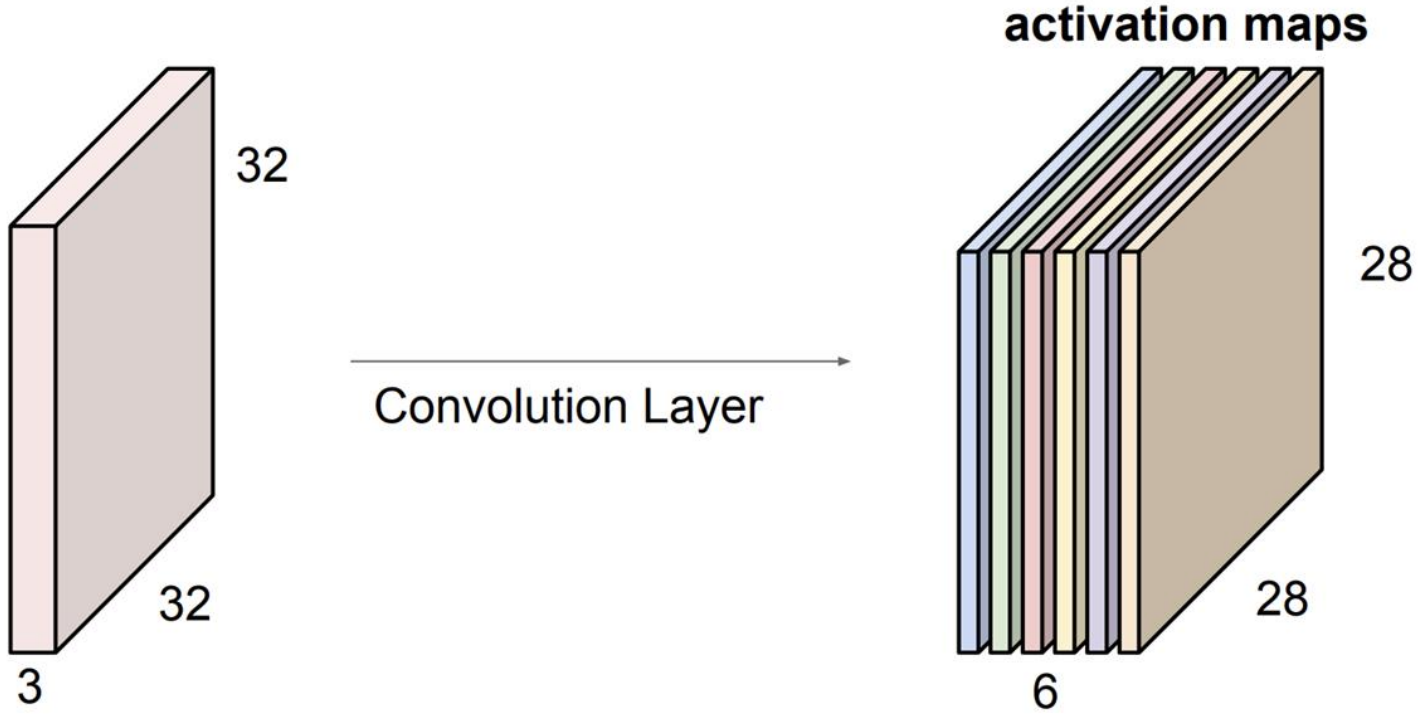
consider a second, green filter



convolve (slide) over all spatial locations

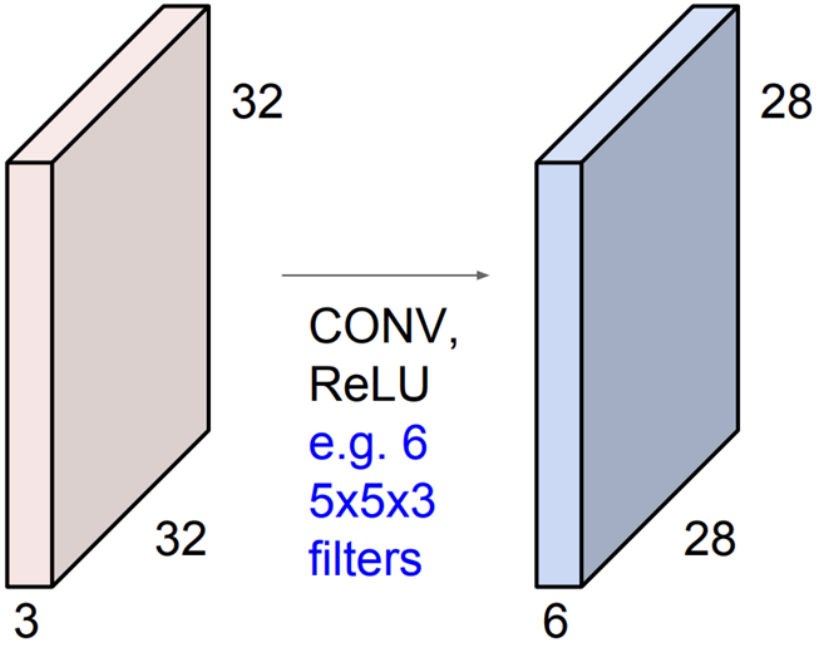


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

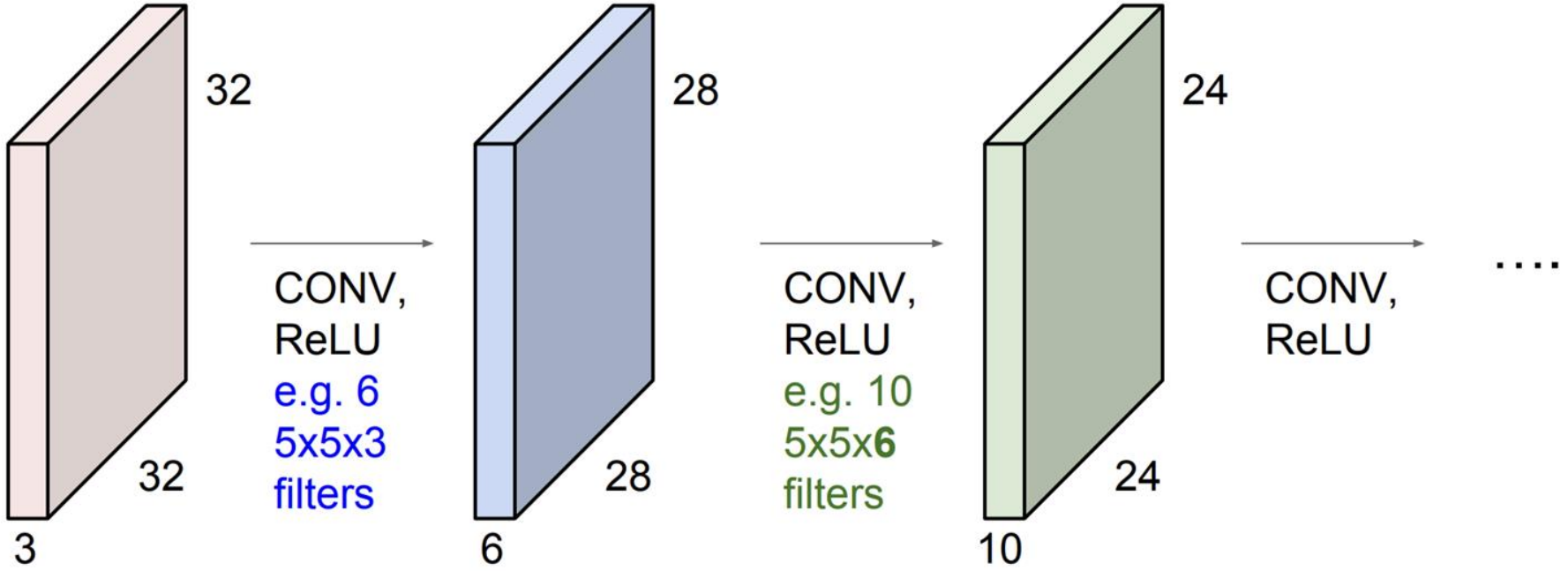


We stack these up to get a "new image" of size 28x28x6!

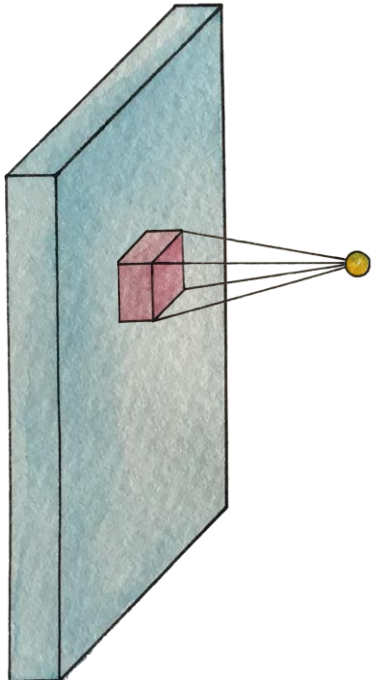
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Max pooling



Single depth slice

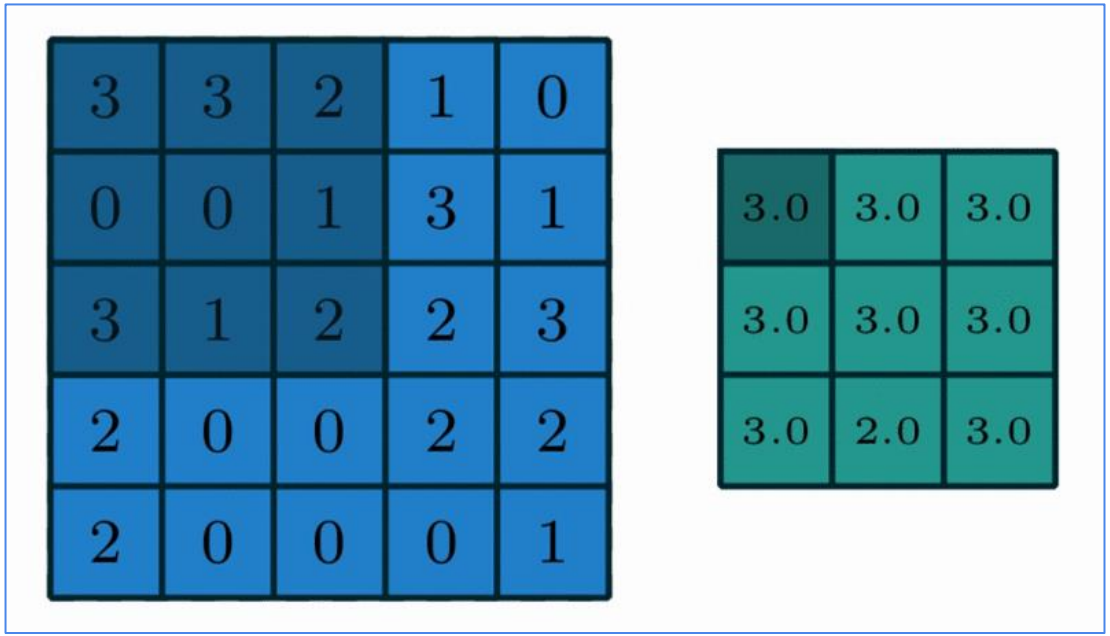
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters
and stride 2

→

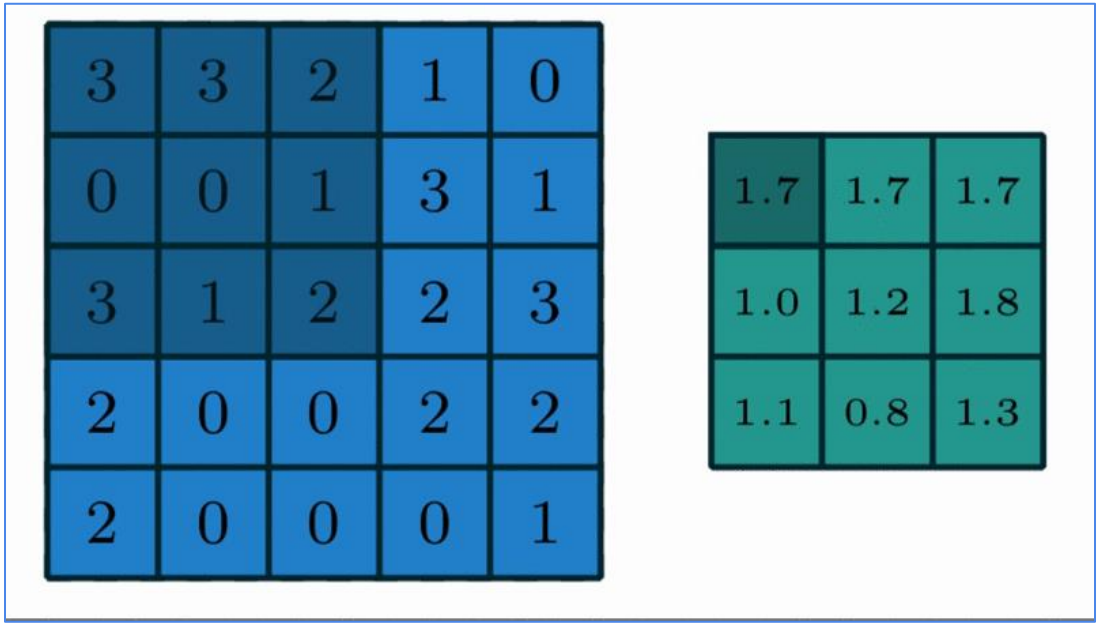
6	8
3	4

Max Pooling in Action



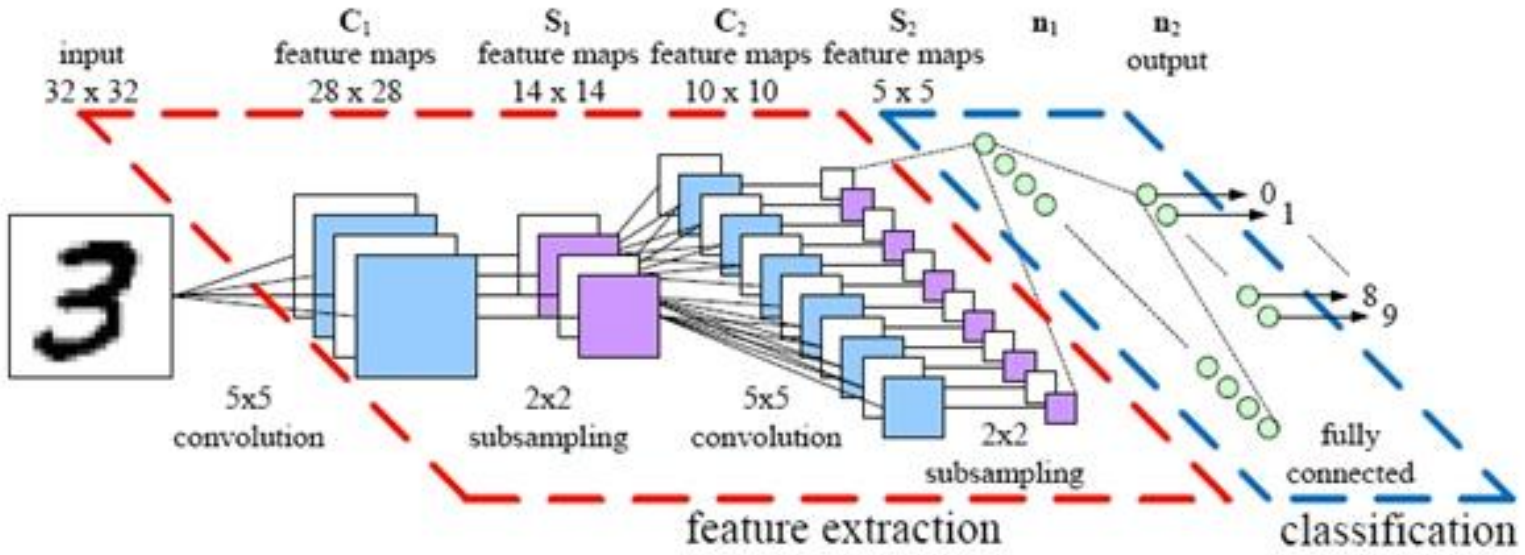
https://github.com/vdumoulin/conv_arithmetic

Avg Pooling in Action



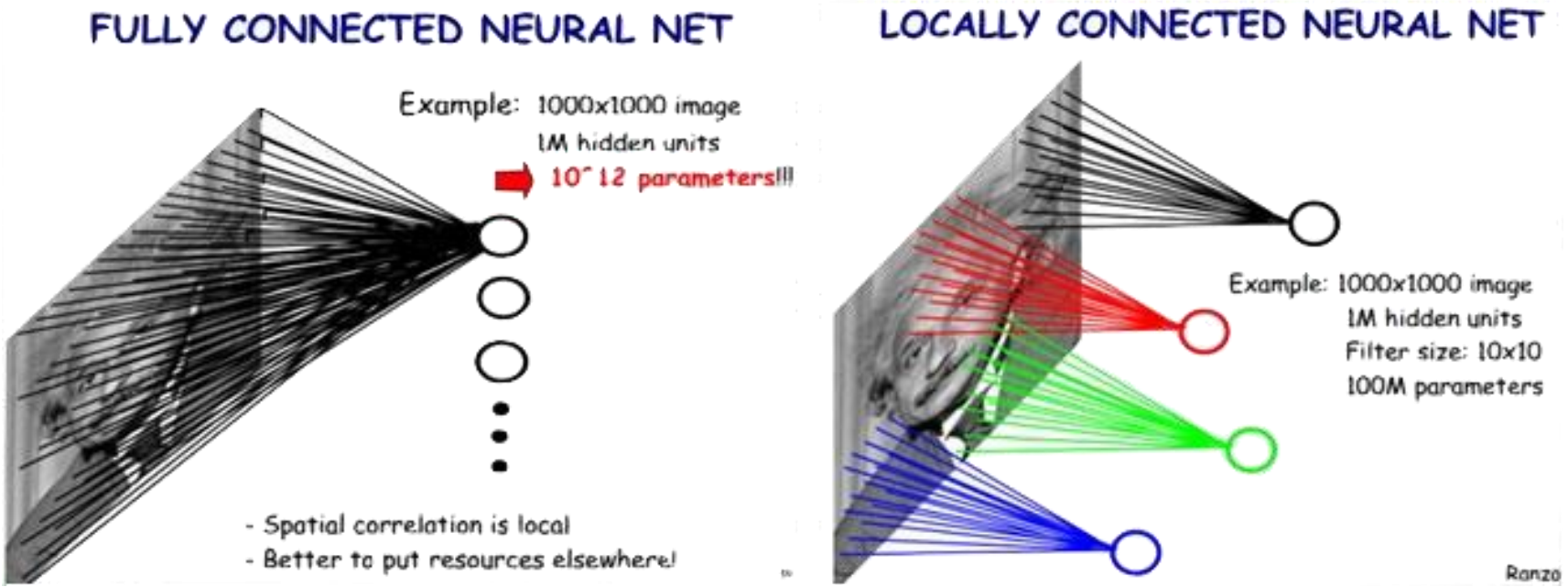
https://github.com/vdumoulin/conv_arithmetic

CNN



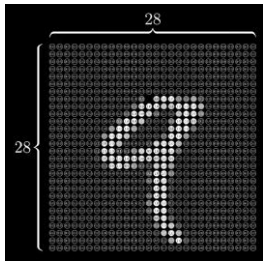
http://parse.ele.tue.nl/cluster/2/CNNArchitecture.jpg

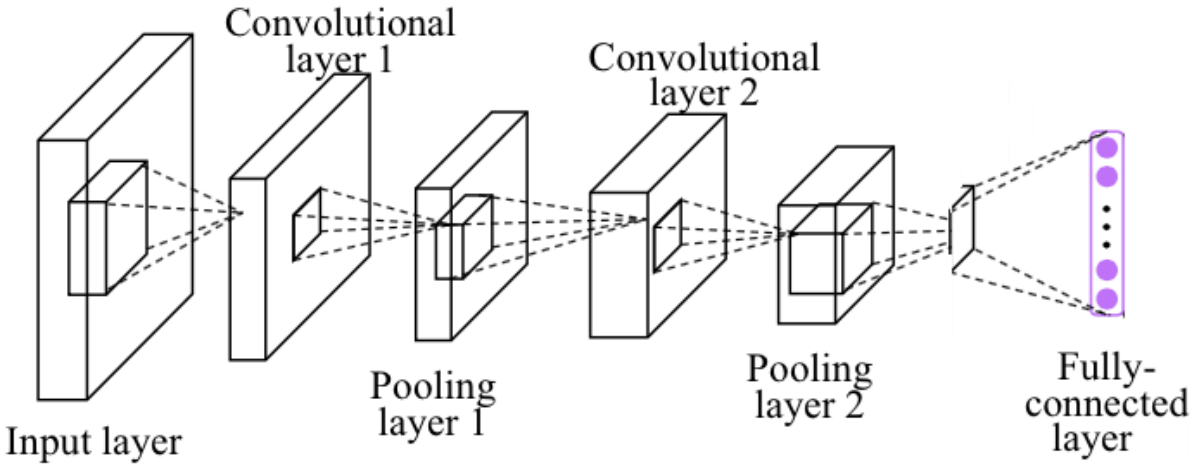
Locally Connected Features



https://ireneli.eu/2016/02/03/deep-learning-05-talk-about-convolutional-neural-network%EF%BC%88cnn%EF%BC%89/

Simple CNN

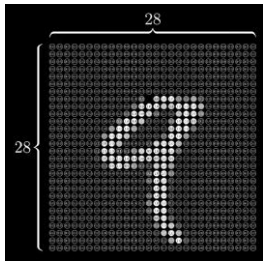

28
28

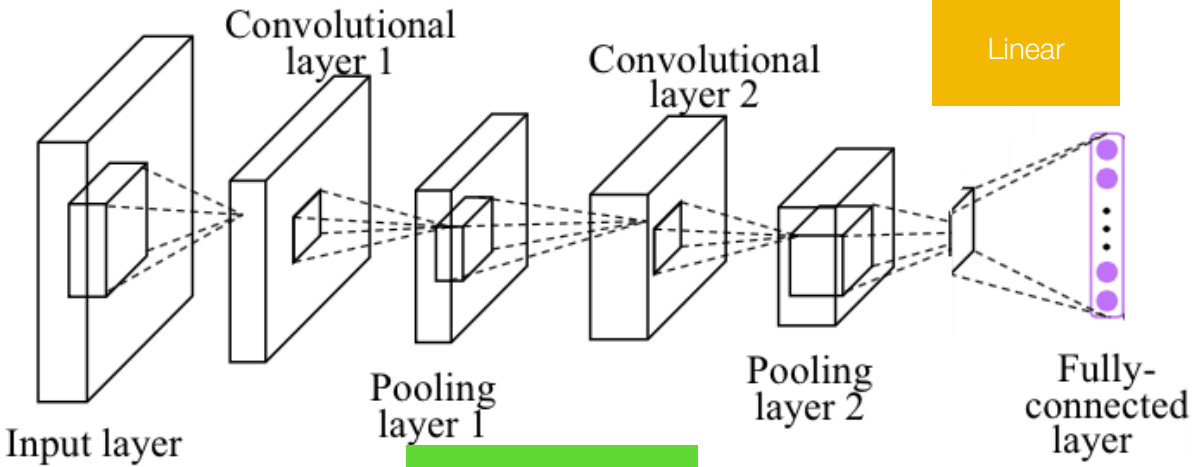

Convolutional layer 1
Convolutional layer 2
Input layer
Pooling layer 1
Pooling layer 2
Fully-connected layer

Simple CNN

```
self.fc = nn.Linear(320, 10)
```

Linear



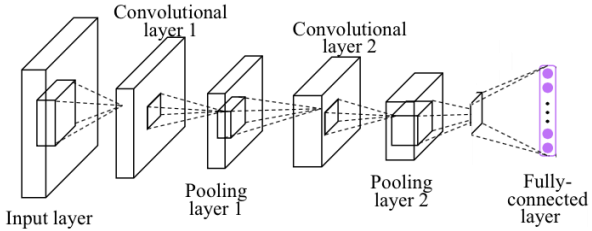


torch.nn.Conv2d (in_channels, out_channels, kernel_siz)

nn.MaxPool2d (kernel_size)

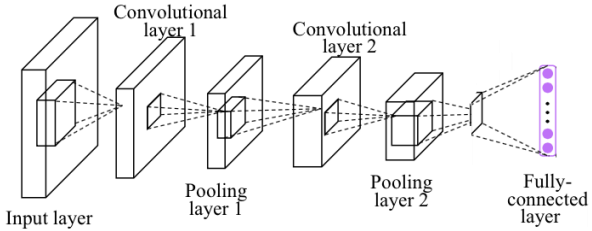
```
self.mp = nn.MaxPool2d(2)
```

```
self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
```



Simple CNN

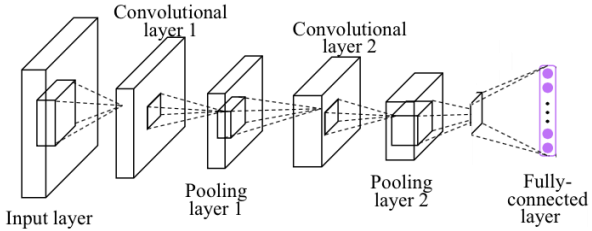
```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(100???, 10) # ??? -> 10

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = F.relu(self.mp(self.conv2(x)))
        x = x.view(in_size, -1) # flatten the tensor
        x = self.fc(x)
        return F.log_softmax(x)
```



Simple CNN

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(100???, 10) # ??? -> 10

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = F.relu(self.mp(self.conv2(x)))
        x = x.view(in_size, -1) # flatten the tensor
        x = self.fc(x)
        return F.log_softmax(x)
```

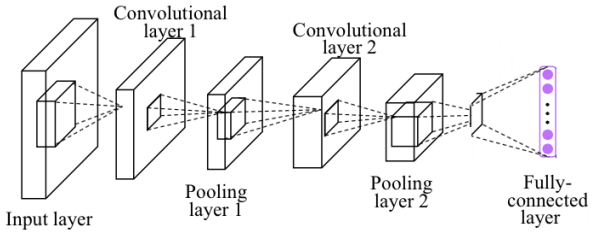
RuntimeError: size mismatch, m1: [64 x **320**], m2: [100 x 10]



Simple CNN

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(320, 10) # 320 -> 10

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = F.relu(self.mp(self.conv2(x)))
        x = x.view(in_size, -1) # flatten the tensor
        x = self.fc(x)
        return F.log_softmax(x)
```

RuntimeError: size mismatch, m1: [64 x **320**], m2: [100 x 10]

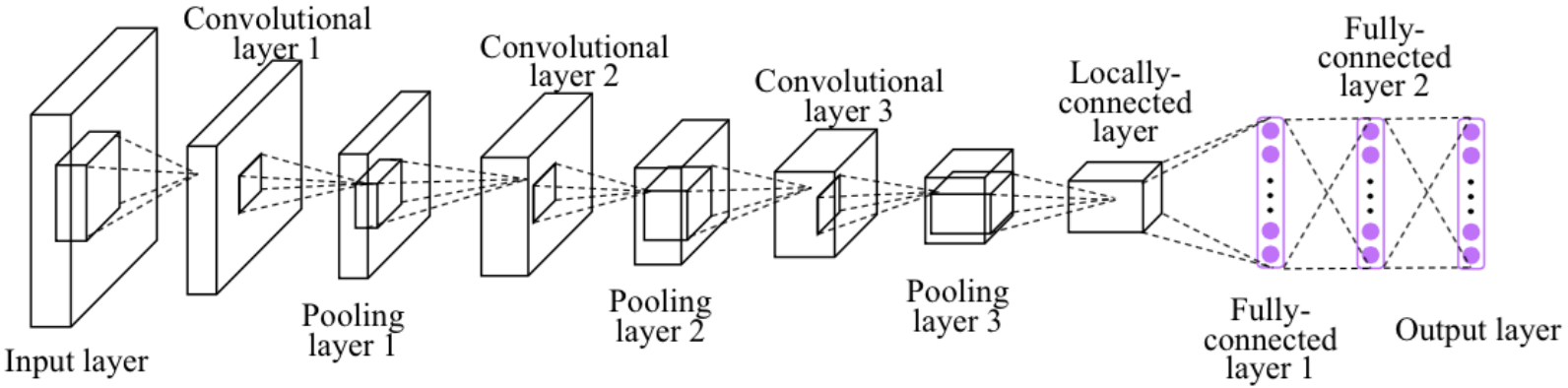


Simple CNN

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(320, 10) # 320 -> 10

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = F.relu(self.mp(self.conv2(x)))
        x = x.view(in_size, -1) # flatten the tensor
        x = self.fc(x)
        return F.log_softmax(x)
```

```
Train Epoch: 9 [46080/60000 (77%)]    Loss: 0.108415
Train Epoch: 9 [46720/60000 (78%)]    Loss: 0.140700
Train Epoch: 9 [47360/60000 (79%)]    Loss: 0.090830
Train Epoch: 9 [48000/60000 (80%)]    Loss: 0.031640
Train Epoch: 9 [48640/60000 (81%)]    Loss: 0.014934
Train Epoch: 9 [49280/60000 (82%)]    Loss: 0.090210
Train Epoch: 9 [49920/60000 (83%)]    Loss: 0.074975
Train Epoch: 9 [50560/60000 (84%)]    Loss: 0.058671
Train Epoch: 9 [51200/60000 (85%)]    Loss: 0.023464
Train Epoch: 9 [51840/60000 (86%)]    Loss: 0.018025
Train Epoch: 9 [52480/60000 (87%)]    Loss: 0.098865
Train Epoch: 9 [53120/60000 (88%)]    Loss: 0.013985
Train Epoch: 9 [53760/60000 (90%)]    Loss: 0.070476
Train Epoch: 9 [54400/60000 (91%)]    Loss: 0.065411
Train Epoch: 9 [55040/60000 (92%)]    Loss: 0.028783
Train Epoch: 9 [55680/60000 (93%)]    Loss: 0.008333
Train Epoch: 9 [56320/60000 (94%)]    Loss: 0.020412
Train Epoch: 9 [56960/60000 (95%)]    Loss: 0.036749
Train Epoch: 9 [57600/60000 (96%)]    Loss: 0.163087
Train Epoch: 9 [58240/60000 (97%)]    Loss: 0.117539
Train Epoch: 9 [58880/60000 (98%)]    Loss: 0.032256
Train Epoch: 9 [59520/60000 (99%)]    Loss: 0.026360

Test set: Average loss: 0.0483, Accuracy: 9846/10000 (98%)
```

Exercise 10-1: Implement CNN more layers



Lecture 11: Advanced CNN