CS4501: Introduction to Computer Vision

Neural Networks + Convolutional Neural Networks

Last Class

- Global Features
- The perceptron model
- Neural Networks – multilayer perceptron model (MLP)
- Backpropagation

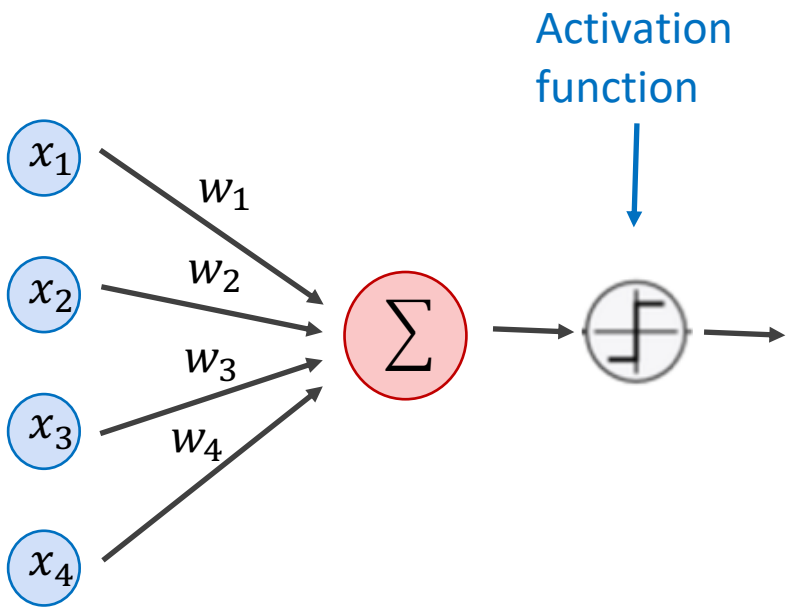
Today’s Class

- Neural Networks – multilayer perceptron model (MLP)
- Backpropagation
- Convolutional Neural Networks

Perceptron Model

Frank Rosenblatt (1957) - Cornell University

$$f(x) = \begin{cases} 1, & \text{if } \sum_{i=0}^{n} w_i x_i + b > 0 \\ 0, & \text{otherwise} \end{cases}$$

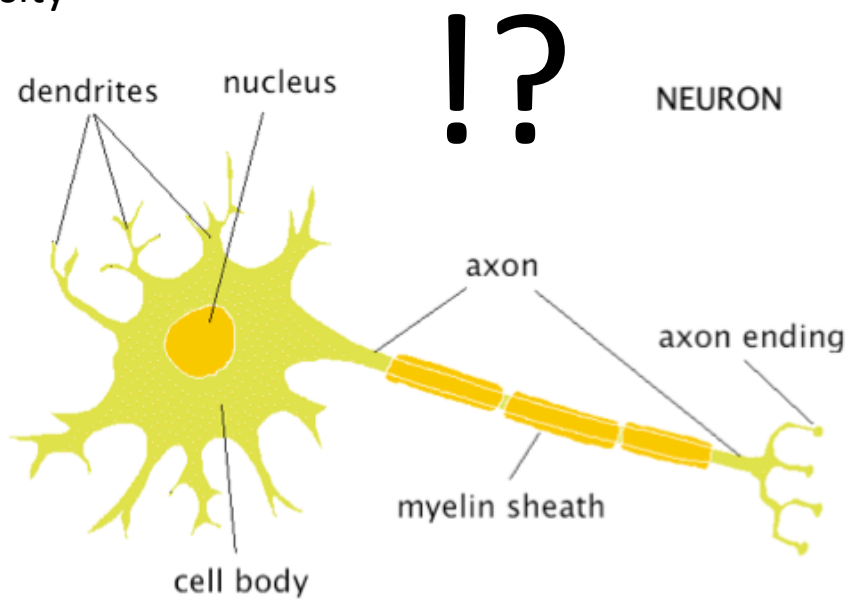


More: https://en.wikipedia.org/wiki/Perceptron

Perceptron Model

Frank Rosenblatt (1957) - Cornell University

$$f(x) = \begin{cases} 1, & \text{if } \sum_{i=0}^{n} w_i x_i + b > 0 \\ 0, & \text{otherwise} \end{cases}$$

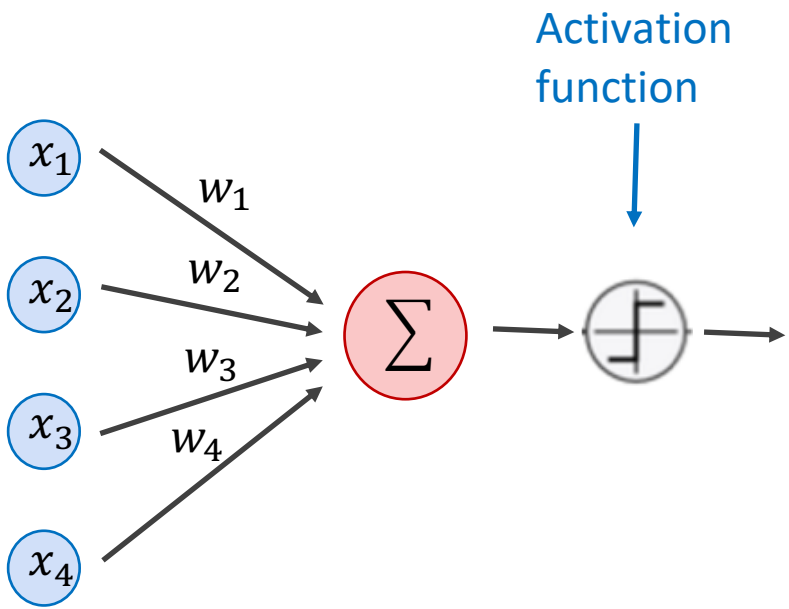


More: https://en.wikipedia.org/wiki/Perceptron

Perceptron Model

Frank Rosenblatt (1957) - Cornell University

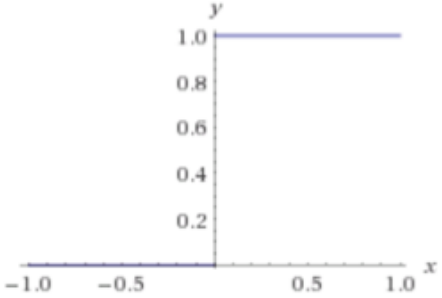
$$f(x) = \begin{cases} 1, & \text{if } \sum_{i=0}^{n} w_i x_i + b > 0 \\ 0, & \text{otherwise} \end{cases}$$



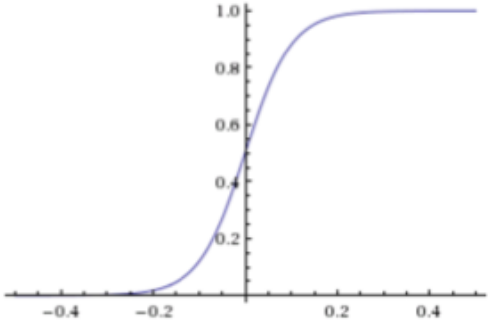
More: https://en.wikipedia.org/wiki/Perceptron

Activation Functions

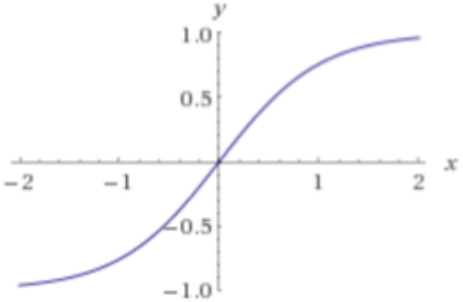
Step(x)



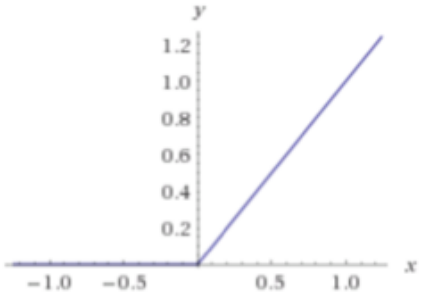
Sigmoid(x)



Tanh(x)



ReLU(x) = max(0, x)

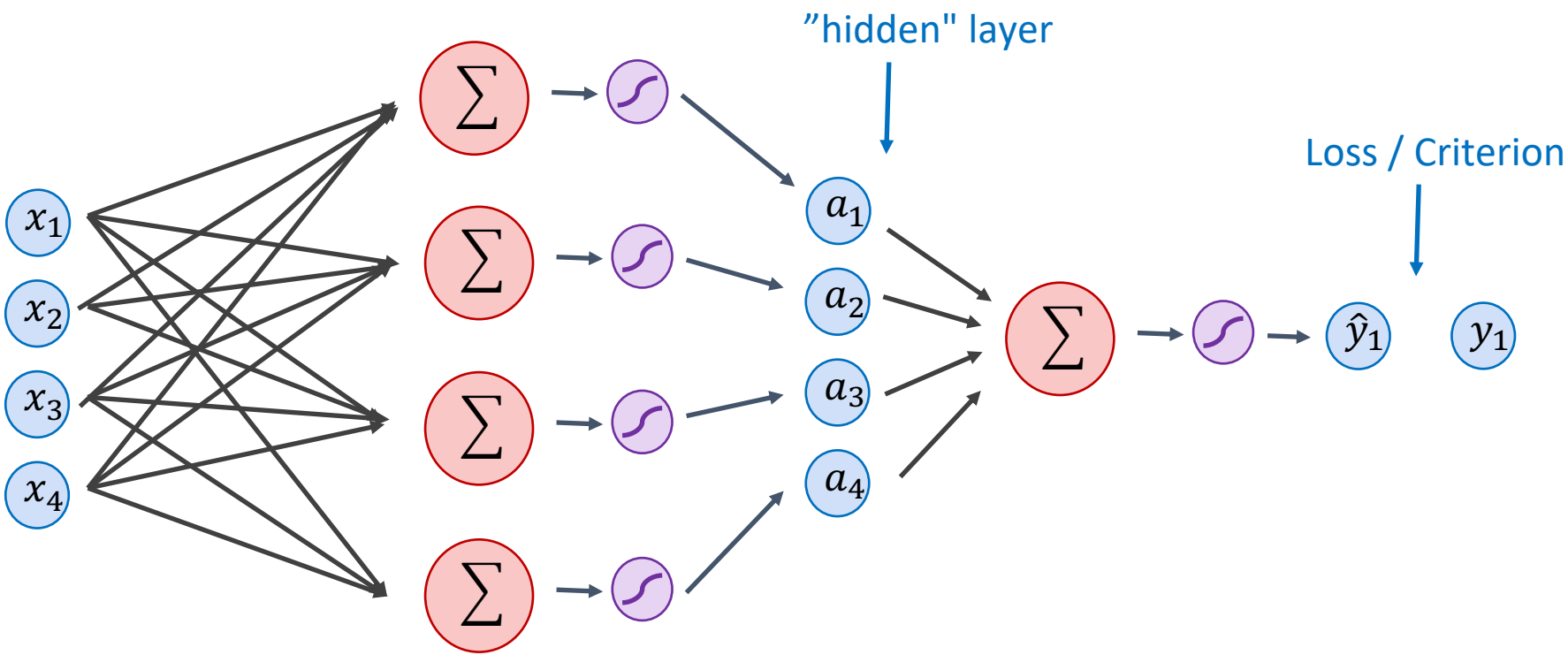


Pytorch - Perceptron

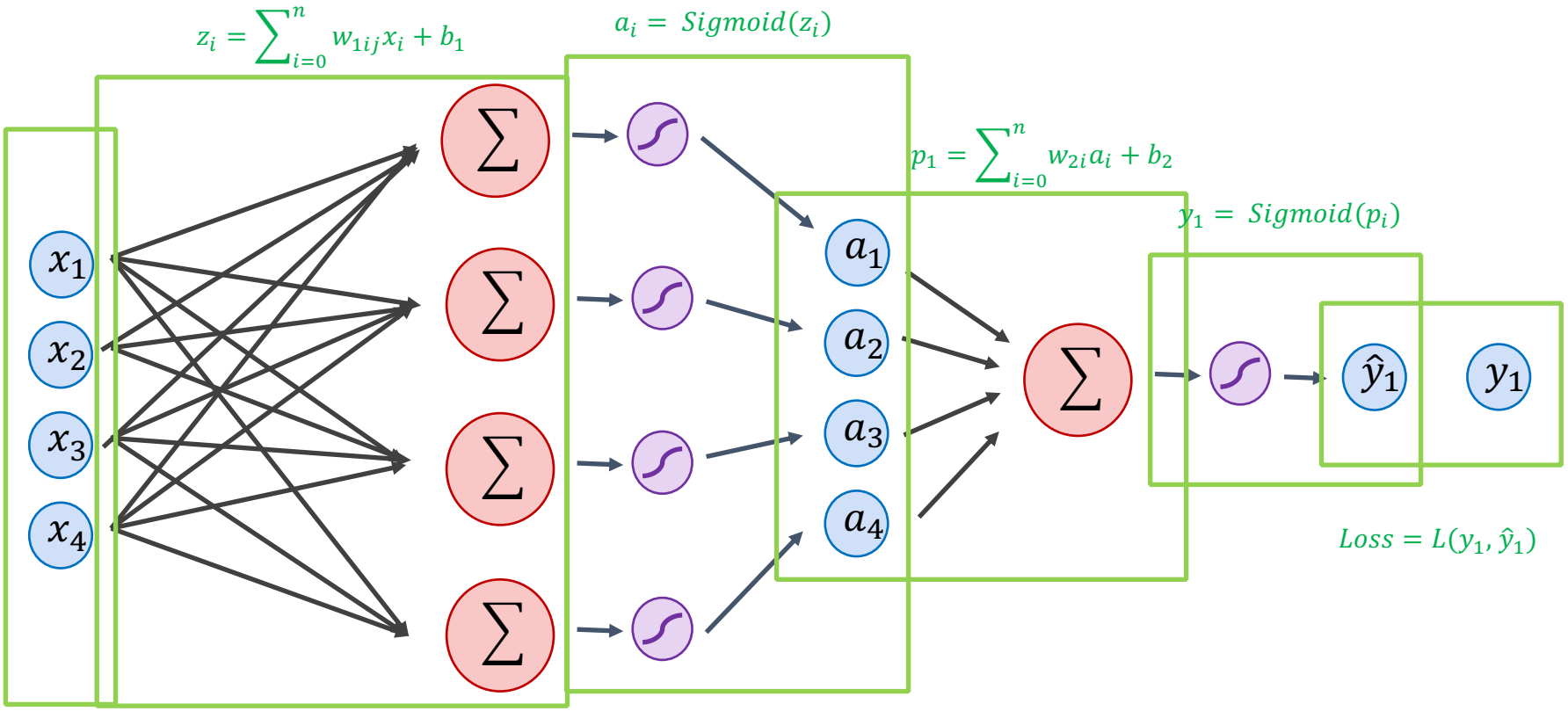
In [5]:
```
import torch
import torch.nn as nn
import torch.autograd

network = nn.Sequential(
    nn.Linear(4, 1),
    nn.Sigmoid()
)

batch_size = 16
input_vector = torch.autograd.Variable(torch.Tensor(batch_size, 4))
predictions = network(input_vector)
print(predictions.size())
```

```
torch.Size([16, 1])
```

Two-layer Multi-layer Perceptron (MLP)



Forward pass



Backward pass

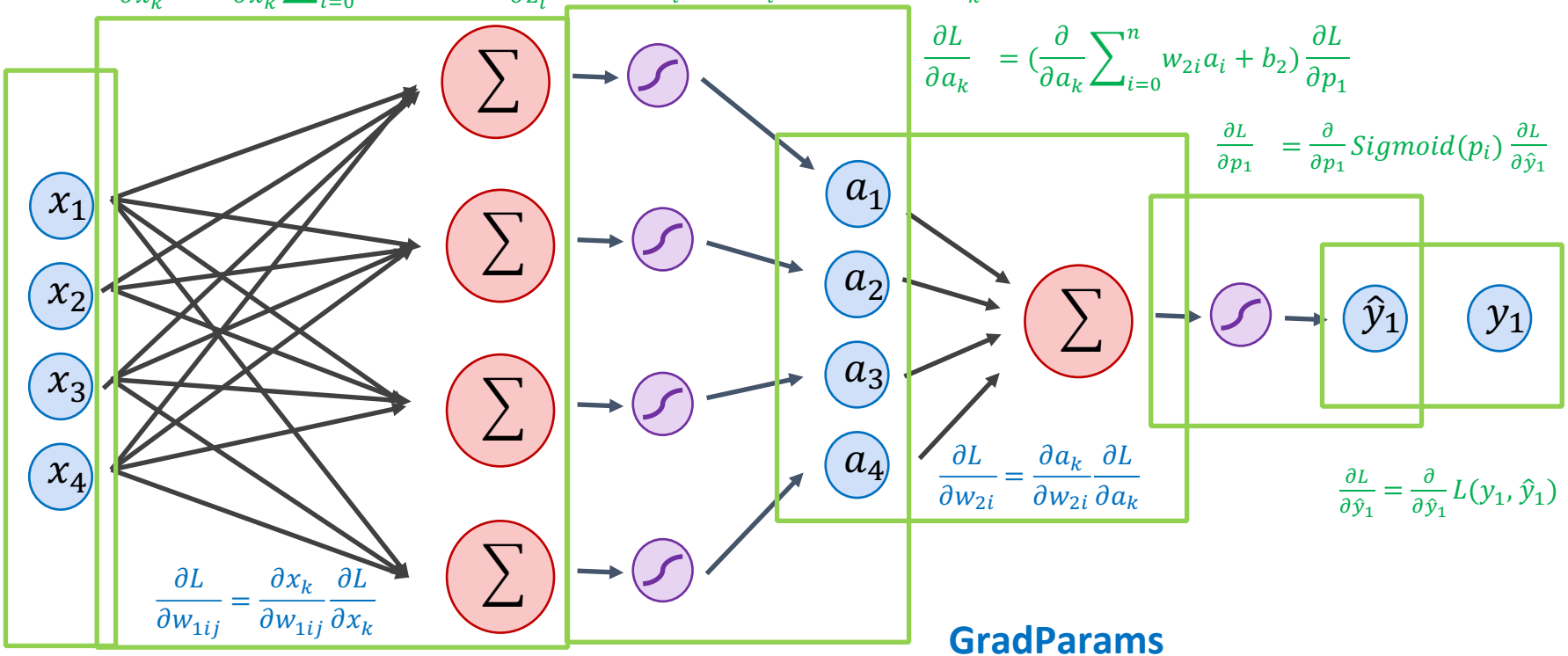
$$\frac{\partial L}{\partial x_k} = \left(\frac{\partial}{\partial x_k}\sum_{i=0}^{n} w_{1ik}x_i + b_k\right)\frac{\partial L}{\partial z_i}$$

$$\frac{\partial L}{\partial z_i} = \frac{\partial}{\partial z_i} Sigmoid(z_i)\frac{\partial L}{\partial a_k}$$

GradInputs

$$\frac{\partial L}{\partial a_k} = \left(\frac{\partial}{\partial a_k}\sum_{i=0}^{n} w_{2i}a_i + b_2\right)\frac{\partial L}{\partial p_1}$$

$$\frac{\partial L}{\partial p_1} = \frac{\partial}{\partial p_1} Sigmoid(p_i)\frac{\partial L}{\partial \hat{y}_1}$$



$$\frac{\partial L}{\partial w_{2i}} = \frac{\partial a_k}{\partial w_{2i}}\frac{\partial L}{\partial a_k}$$

$$\frac{\partial L}{\partial \hat{y}_1} = \frac{\partial}{\partial \hat{y}_1} L(y_1, \hat{y}_1)$$

$$\frac{\partial L}{\partial w_{1ij}} = \frac{\partial x_k}{\partial w_{1ij}}\frac{\partial L}{\partial x_k}$$

GradParams

Pytorch – Two-layer MLP + Regression

In [9]:
```
import torch
import torch.nn as nn
import torch.autograd

network = nn.Sequential(
    nn.Linear(4, 4),
    nn.Sigmoid(),
    nn.Linear(4, 1),
    nn.Sigmoid()
)

batch_size = 16
input_vector = torch.autograd.Variable(torch.Tensor(batch_size, 4))
predictions = network(input_vector)
print(predictions.size())
```

```
torch.Size([16, 1])
```

In [10]:
```
criterion = nn.MSELoss()
loss = criterion(predictions, labels)
```

Pytorch – Two-layer MLP + LogSoftmax

In [16]:
```
import torch
import torch.nn as nn
import torch.autograd

network = nn.Sequential(
    nn.Linear(3072, 512),
    nn.Sigmoid(),
    nn.Linear(512, 10),
    nn.LogSoftmax()
)

batch_size = 16
input_vector = torch.autograd.Variable(torch.Tensor(batch_size, 3072))
predictions = network(input_vector)
print(predictions.size())
```

# of Hidden Units

```
torch.Size([16, 10])
```

In [13]:
```
criterion = nn.NLLLoss()
loss = criterion(predictions, labels)
```

Pytorch – Two-layer MLP + LogSoftmax

In [17]:
```
import torch
import torch.nn as nn
import torch.autograd

network = nn.Sequential(
    nn.Linear(3072, 512),
    nn.Sigmoid(),
    nn.Linear(512, 10),
)

batch_size = 16
input_vector = torch.autograd.Variable(torch.Tensor(batch_size, 3072))
predictions = network(input_vector)
print(predictions.size())
```

```
torch.Size([16, 10])
```

LogSoftmax + Negative Likelihood Loss

In [13]:
```
criterion = nn.CrossEntropyLoss()
loss = criterion(predictions, labels)
```

PYTORCH

master (0.2.0+561fc8d) ▼

Search docs

NOTES

- Autograd mechanics
- Broadcasting semantics
- CUDA semantics
- Extending PyTorch
- Multiprocessing best practices
- Serialization semantics

PACKAGE REFERENCE

- torch
- torch.Tensor
- torch.sparse
- torch.Storage
- torch.nn
- torch.nn.functional
- torch.nn.init

Docs » PyTorch documentation

Edit on GitHub

PyTorch documentation

PyTorch is an optimized tensor library for deep learning using GPUs and CPUs.

Notes

- Autograd mechanics
- Broadcasting semantics
- CUDA semantics
- Extending PyTorch
- Multiprocessing best practices
- Serialization semantics

Package Reference

- torch
- torch.Tensor
- torch.sparse
- torch.Storage
- torch.nn
- torch.nn.functional

Bookmark Opportunity!

SGD training code (Project 4)

```
# Re-initialize the classifier.
classifier = Classifier()

# Let's define the loss function or criterion.
criterion = nn.NLLLoss()

# Number of epochs is the number of times we go over the full training set.
num_epochs = 10

# Learning rate.
learningRate = 0.005

# This is often needed to prepare model for training.
classifier.train()

# Short-cut for the model parameters.
weight = classifier.linear.weight;
bias = classifier.linear.bias;

# Training loop starts.
for epoch in range(0, num_epochs):

  correct = 0
  cumloss = 0
  # Go over all the training data one batch at a time.
  for (i, (x, y)) in enumerate(trainLoader):

    # Flatten the images in the batch to input to the classifier.
    x = Variable(x.view(x.shape[0], 3 * 32 * 32))
    y = Variable(y)

    # Compute predictions under the current model.
    yhat = classifier(x)

    # Compute the loss with respect to this batch.
    loss = criterion(yhat, y)

    # Set to zero gradients computed in previous iteration.
    if weight.grad is not None:
      weight.grad.data.zero_()
      bias.grad.data.zero_()

    # Compute the gradients for the entire model variables,
    # this includes inputs, outputs, and parameters (weight, and bias).
    loss.backward()

    # Now we can update the weight and bias parameters.
    weight.data.add_(-learningRate * weight.grad.data)
    bias.data.add_(-learningRate * bias.grad.data)
```

Convolutional (Neural) Networks

Convolutional Layer

Input image * Weights → Output image

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

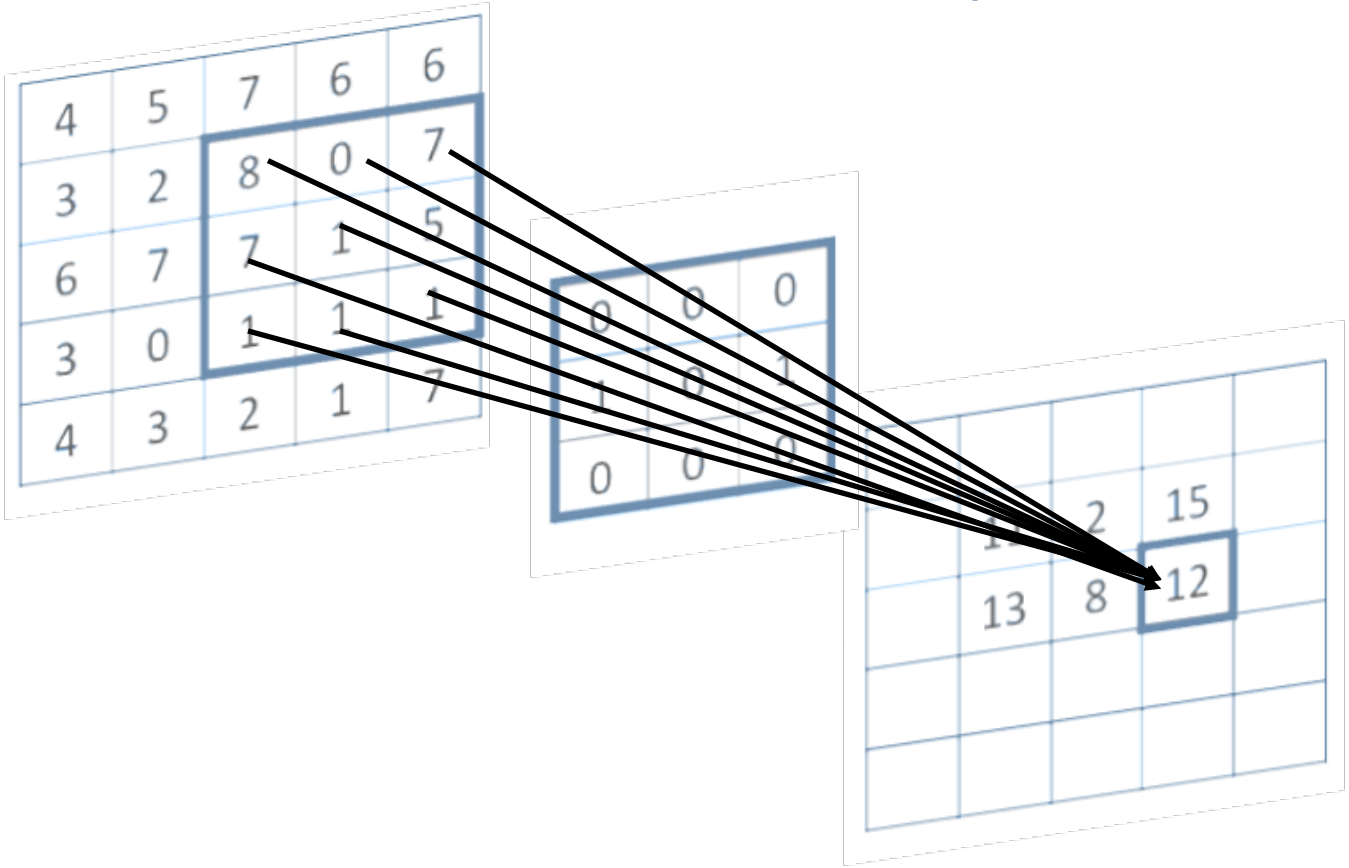
*

0	0	0
1	0	1
0	0	0

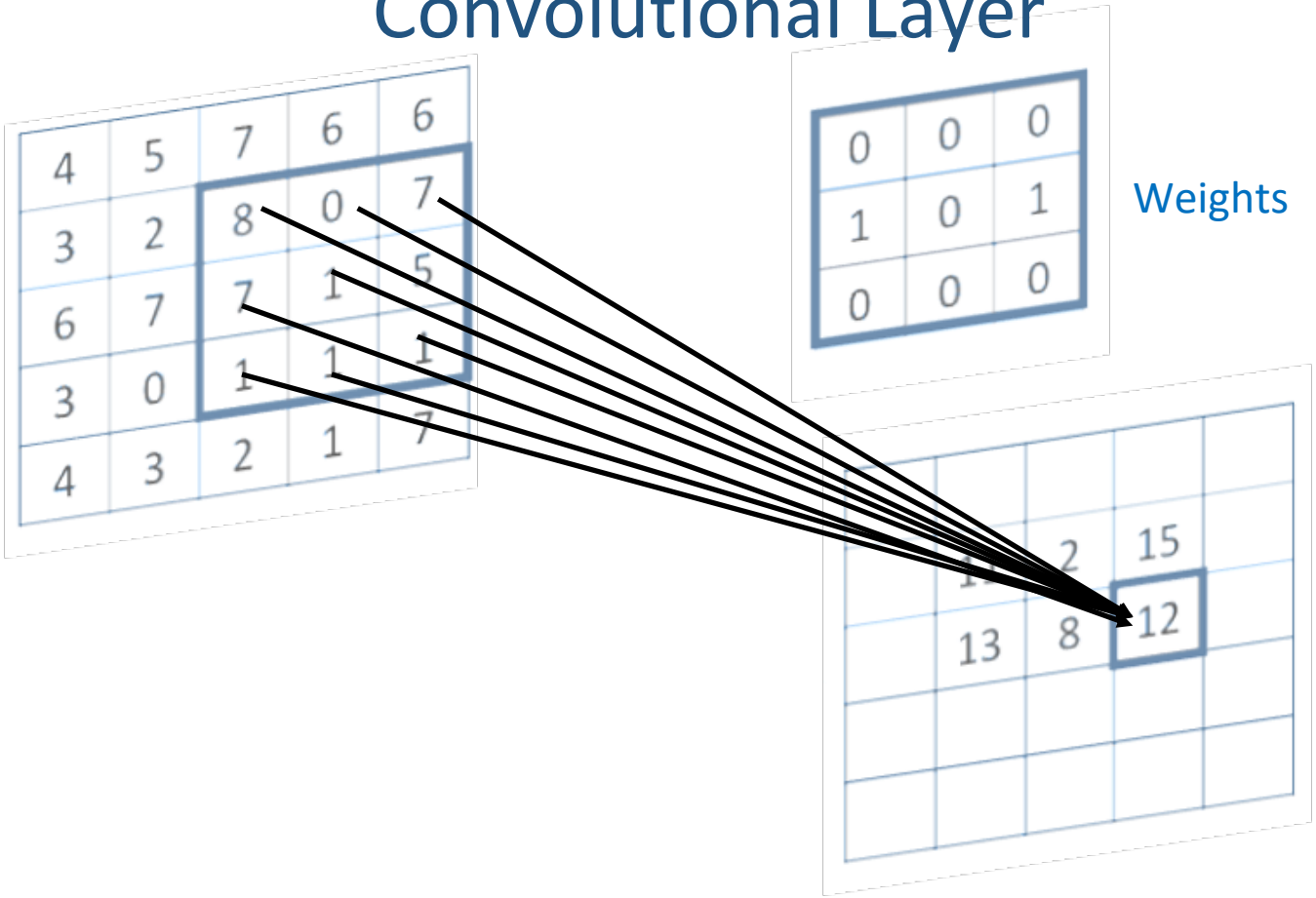
→

	11	2	15	
	13	8	12	

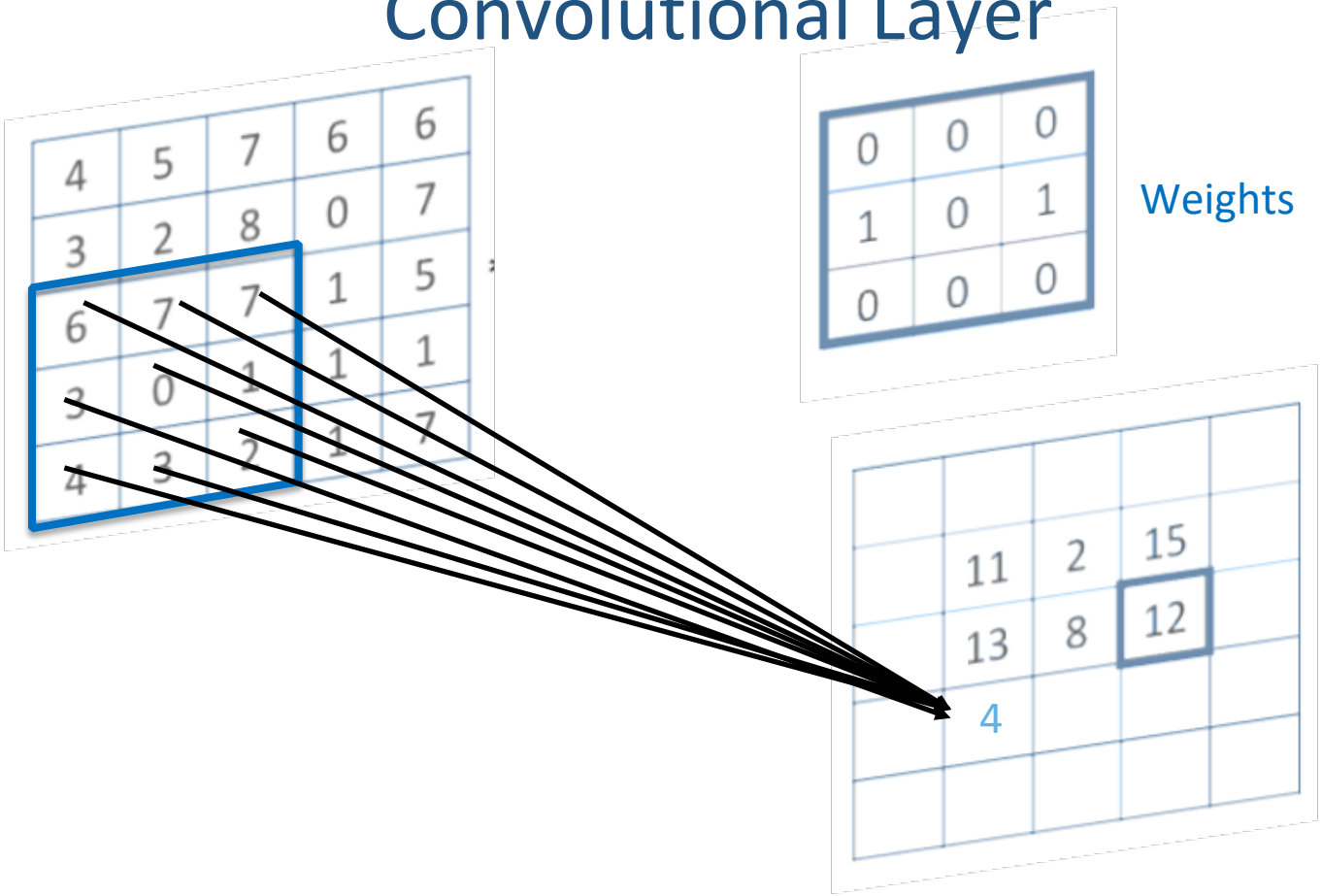
Convolutional Layer



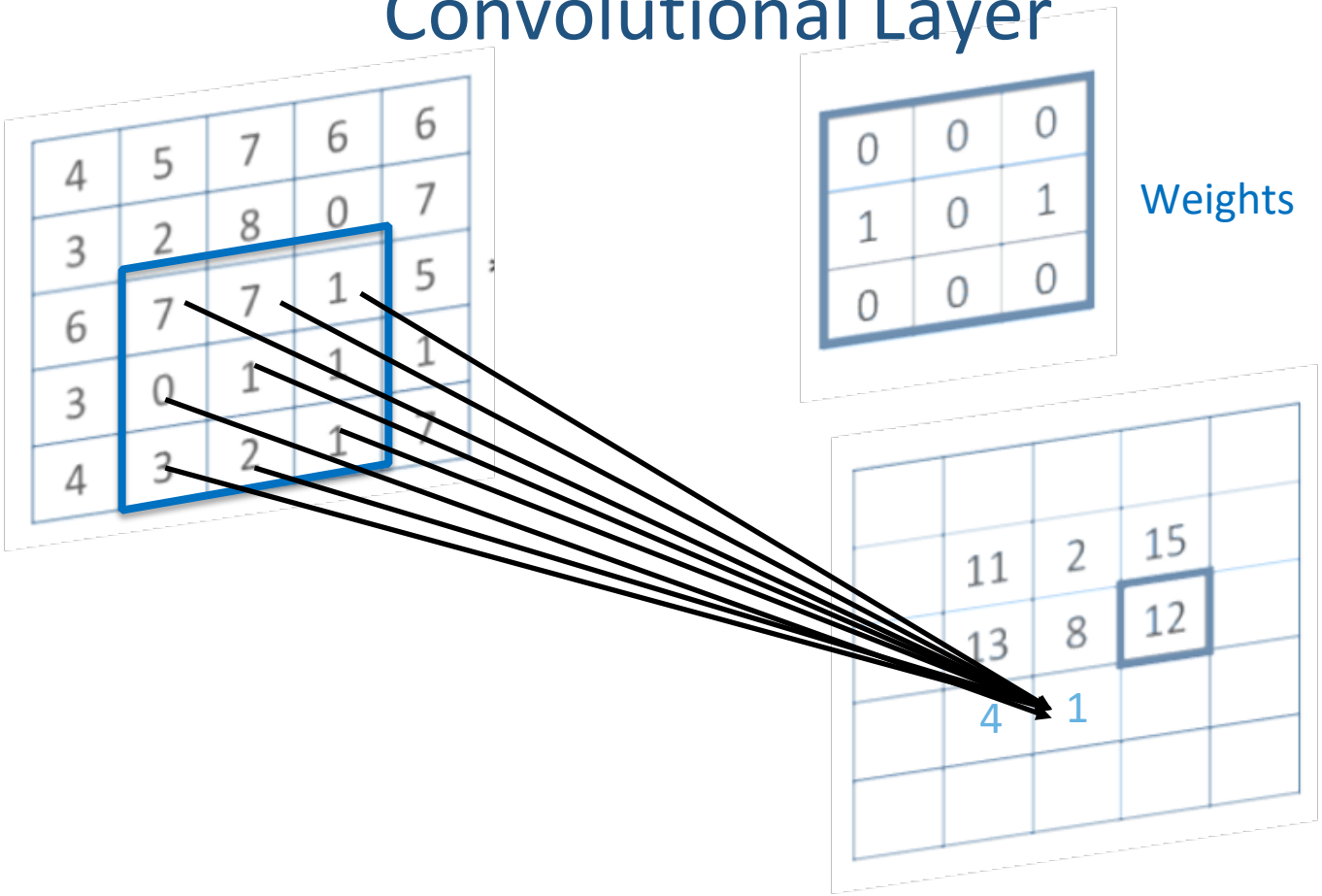
Convolutional Layer



Convolutional Layer



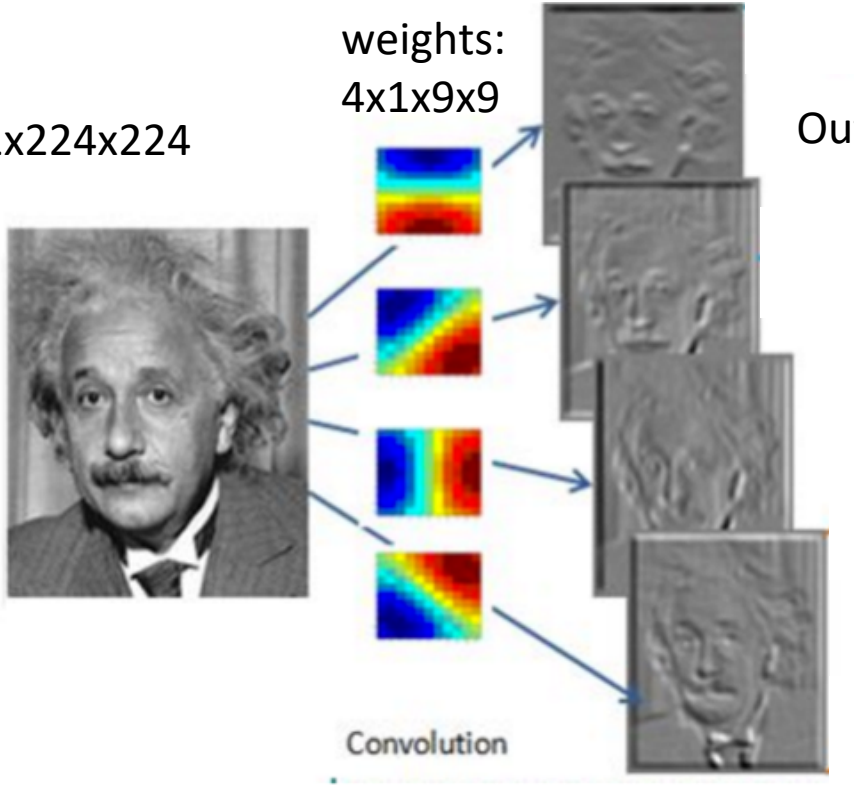
Convolutional Layer



Convolutional Layer (with 4 filters)

Input: 1x224x224

weights:
4x1x9x9



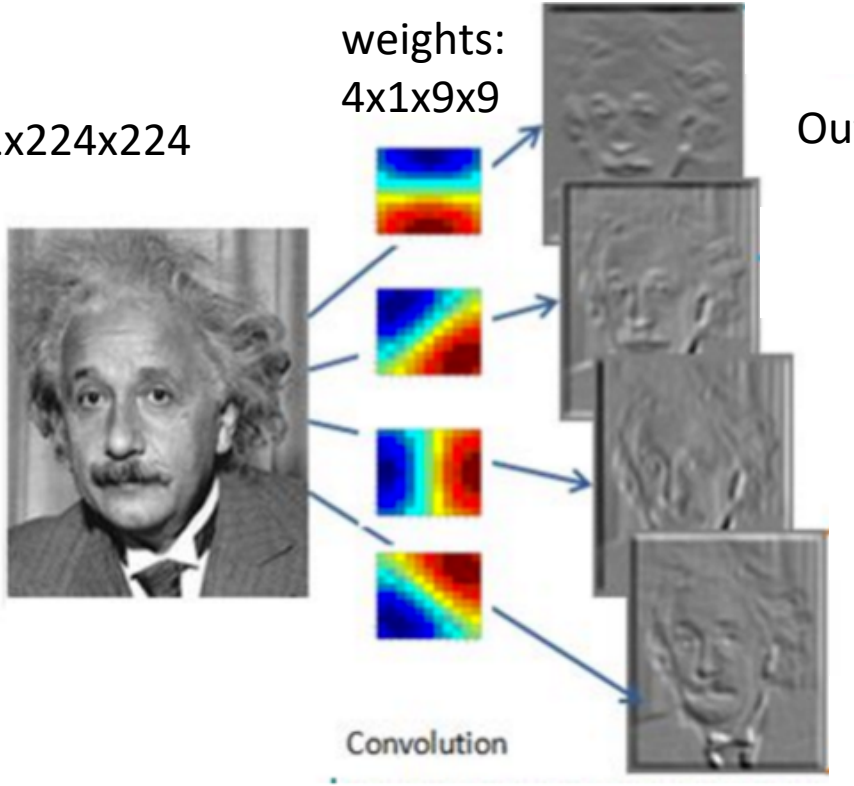
Output: 4x224x224

if zero padding,
and stride = 1

Convolutional Layer (with 4 filters)

Input: 1x224x224

weights:
4x1x9x9



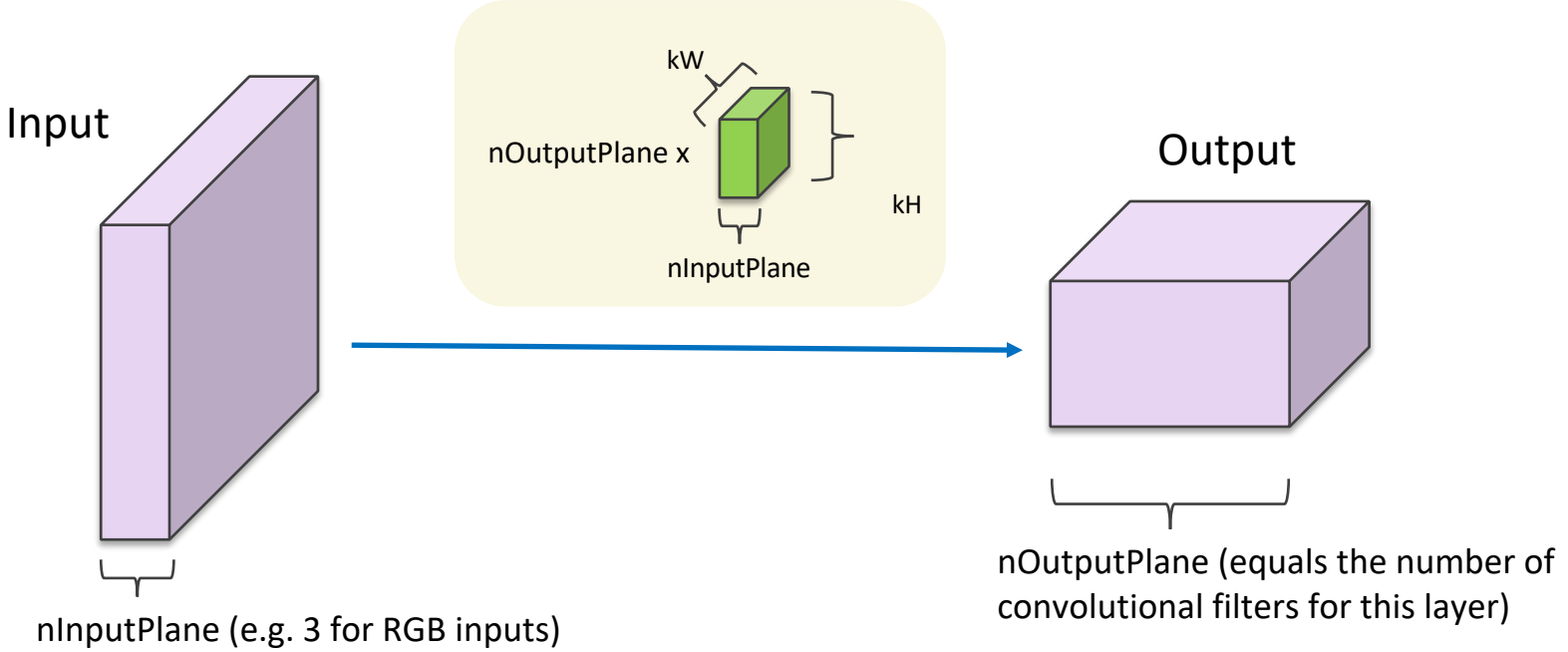
Output: 4x112x112

if zero padding,
but stride = 2

Convolutional Layer in Torch

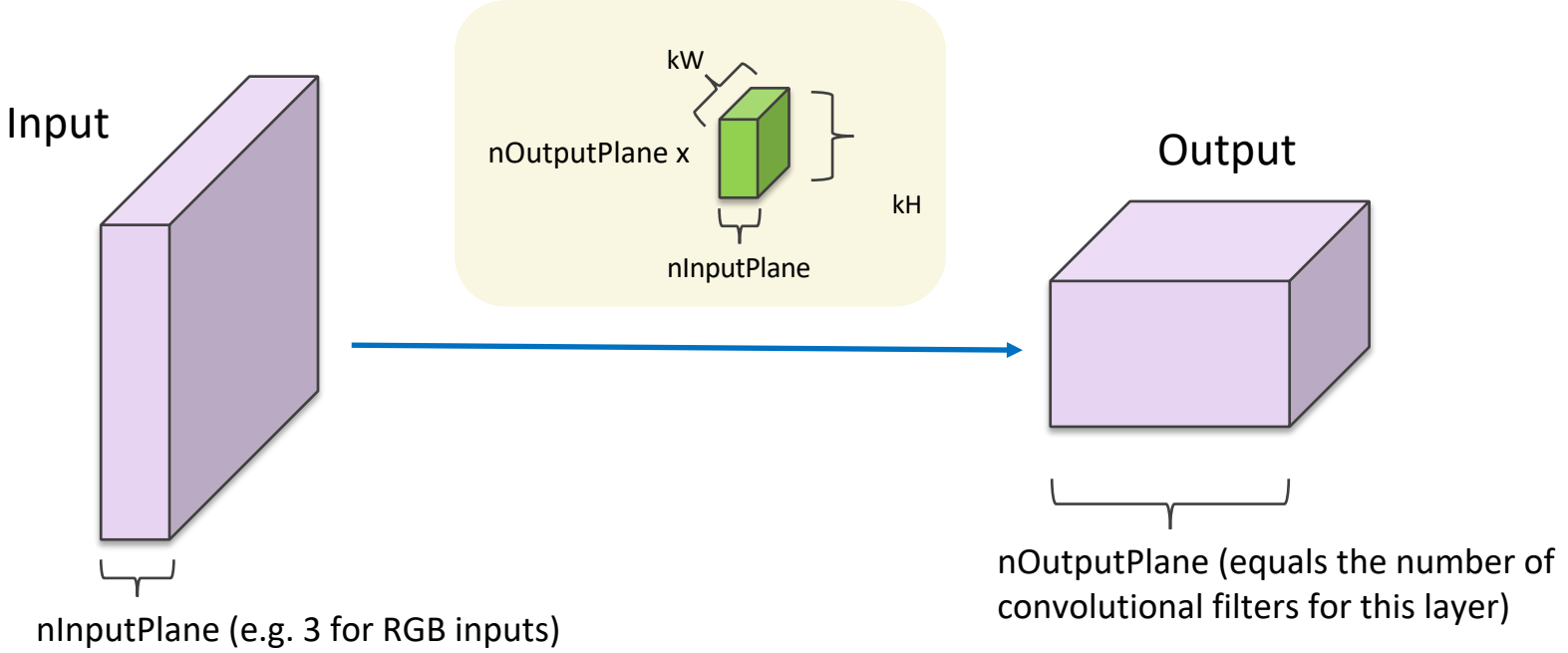
SpatialConvolution

```
module = nn.SpatialConvolution(nInputPlane, nOutputPlane, kW, kH, [dW], [dH], [padW], [padH])
```



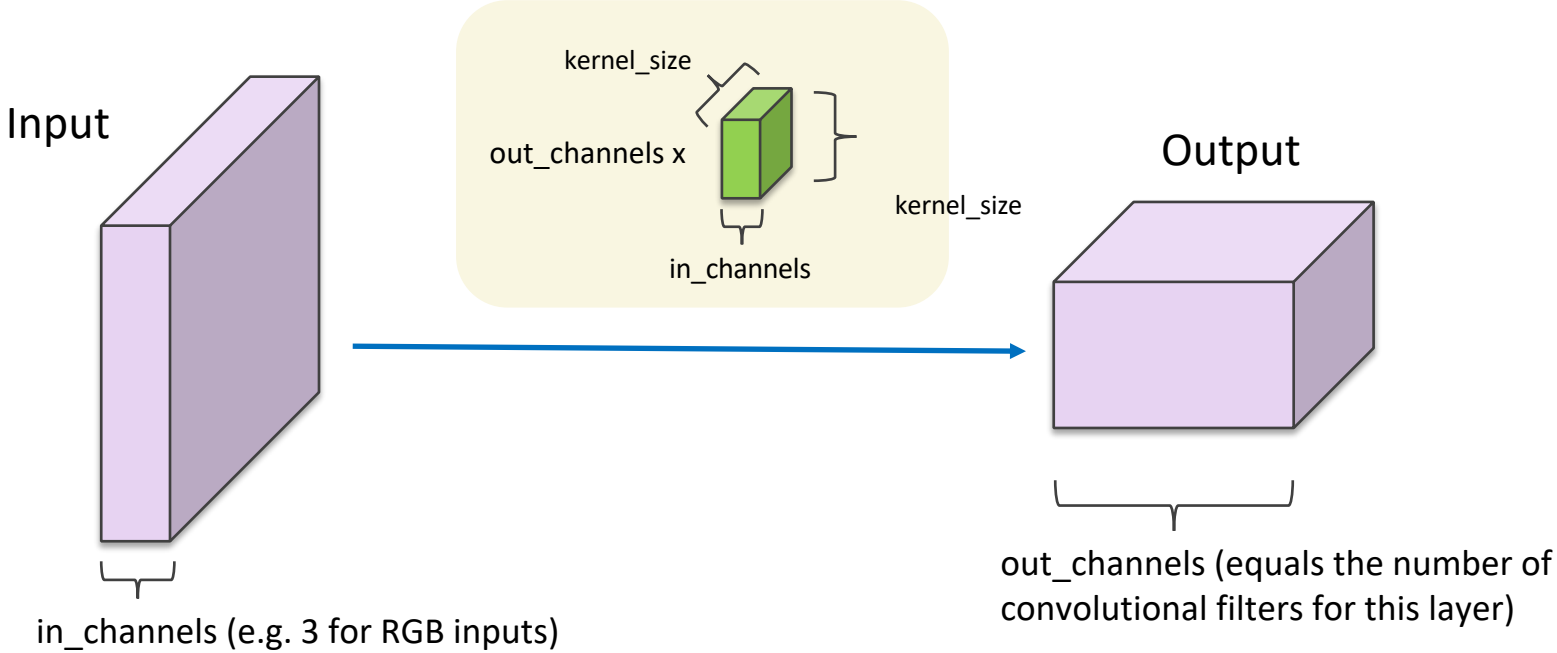
Convolutional Layer in Keras

Convolution2D(nOutputPlane, kW, kH, input_shape = (3, 224, 224), subsample = 2, border_mode = valid)

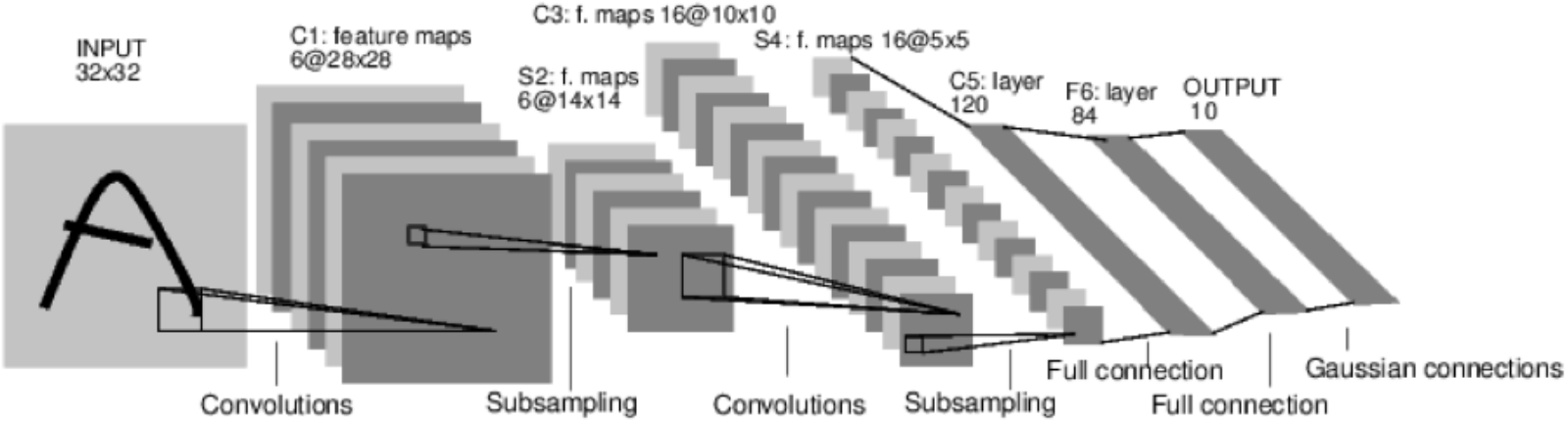


Convolutional Layer in pytorch

class `torch.nn.Conv2d`*(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)* [source]



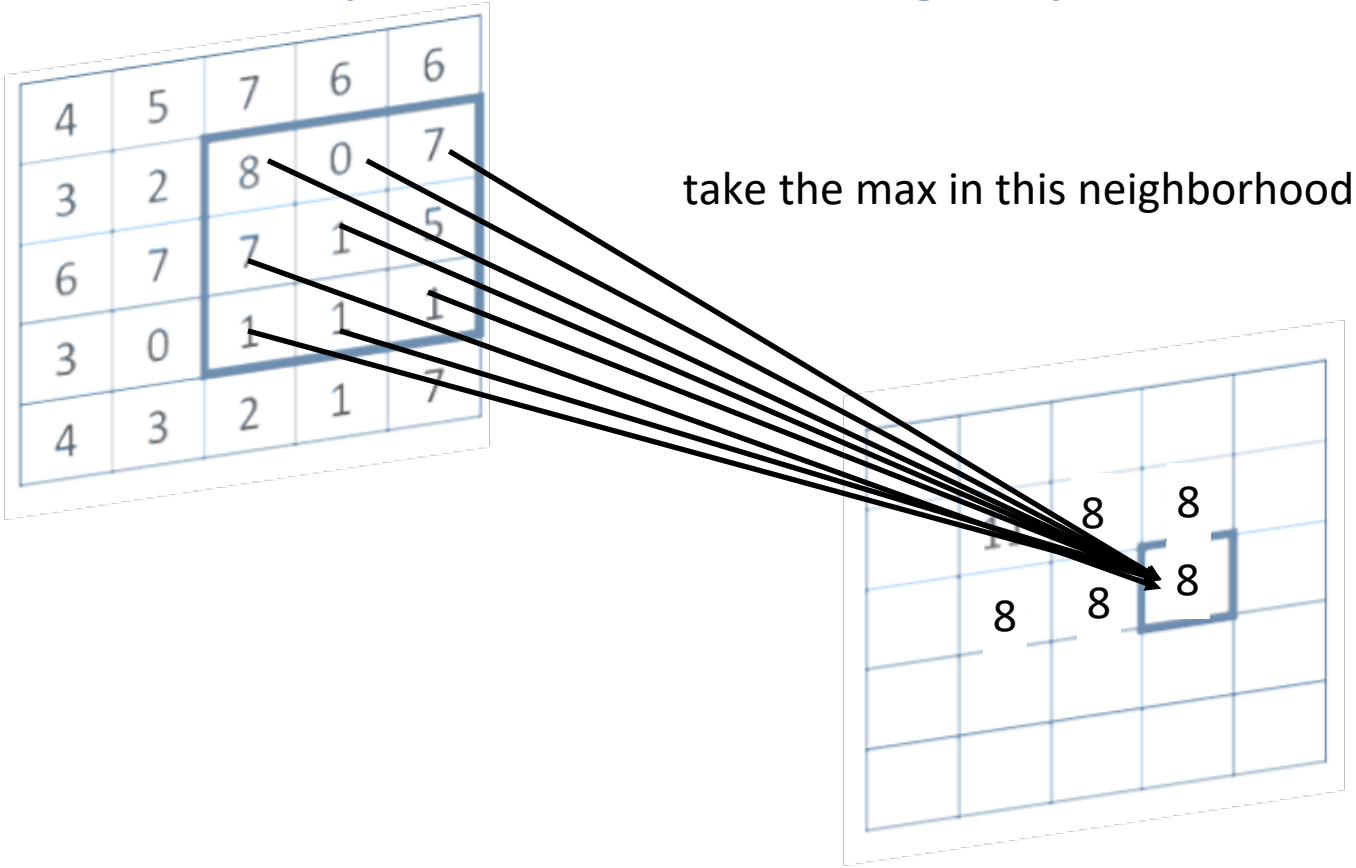
Convolutional Network: LeNet



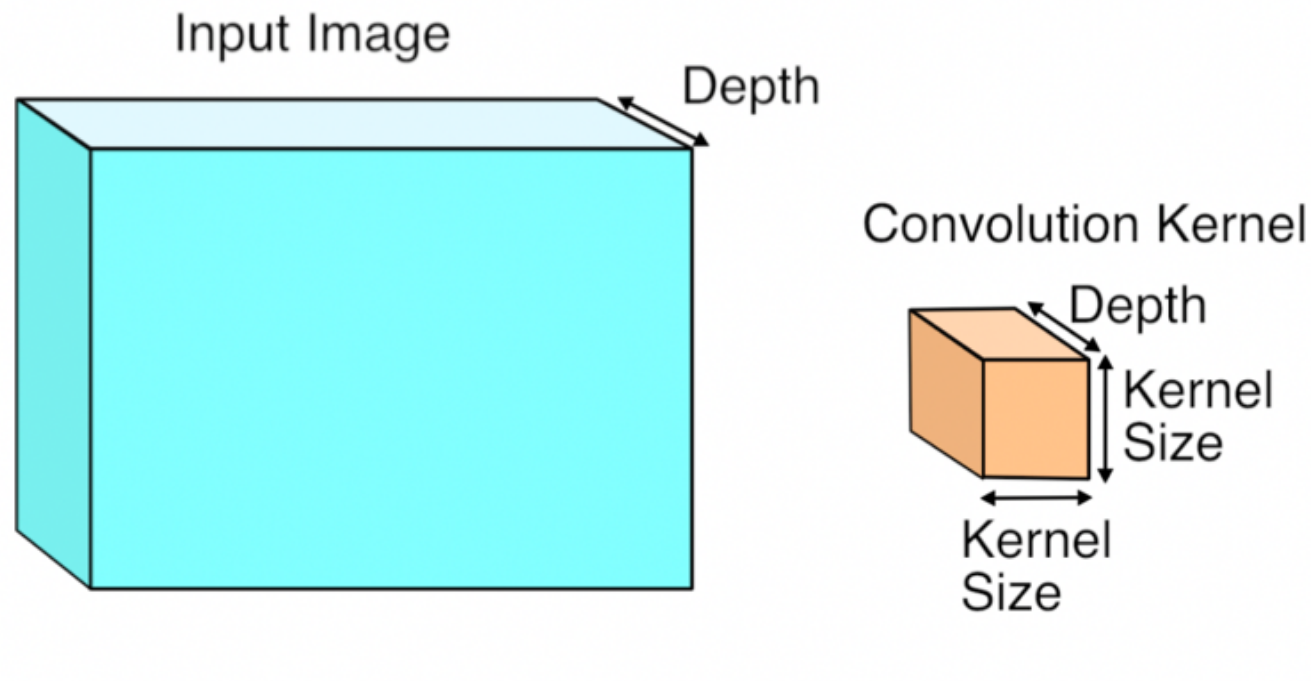
LeNet in Pytorch

```python
# LeNet is French for The Network, and is taken from Yann Lecun's 98 paper
# on digit classification http://yann.lecun.com/exdb/lenet/
# This was also a network with just two convolutional layers.
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        # Convolutional layers.
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)

        # Linear layers.
        self.fc1 = nn.Linear(16*5*5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        out = F.relu(self.conv1(x))
        out = F.max_pool2d(out, 2)
        out = F.relu(self.conv2(out))
        out = F.max_pool2d(out, 2)

        # This flattens the output of the previous layer into a vector.
        out = out.view(out.size(0), -1)
        out = F.relu(self.fc1(out))
        out = F.relu(self.fc2(out))
        out = self.fc3(out)
        return out
```

SpatialMaxPooling Layer

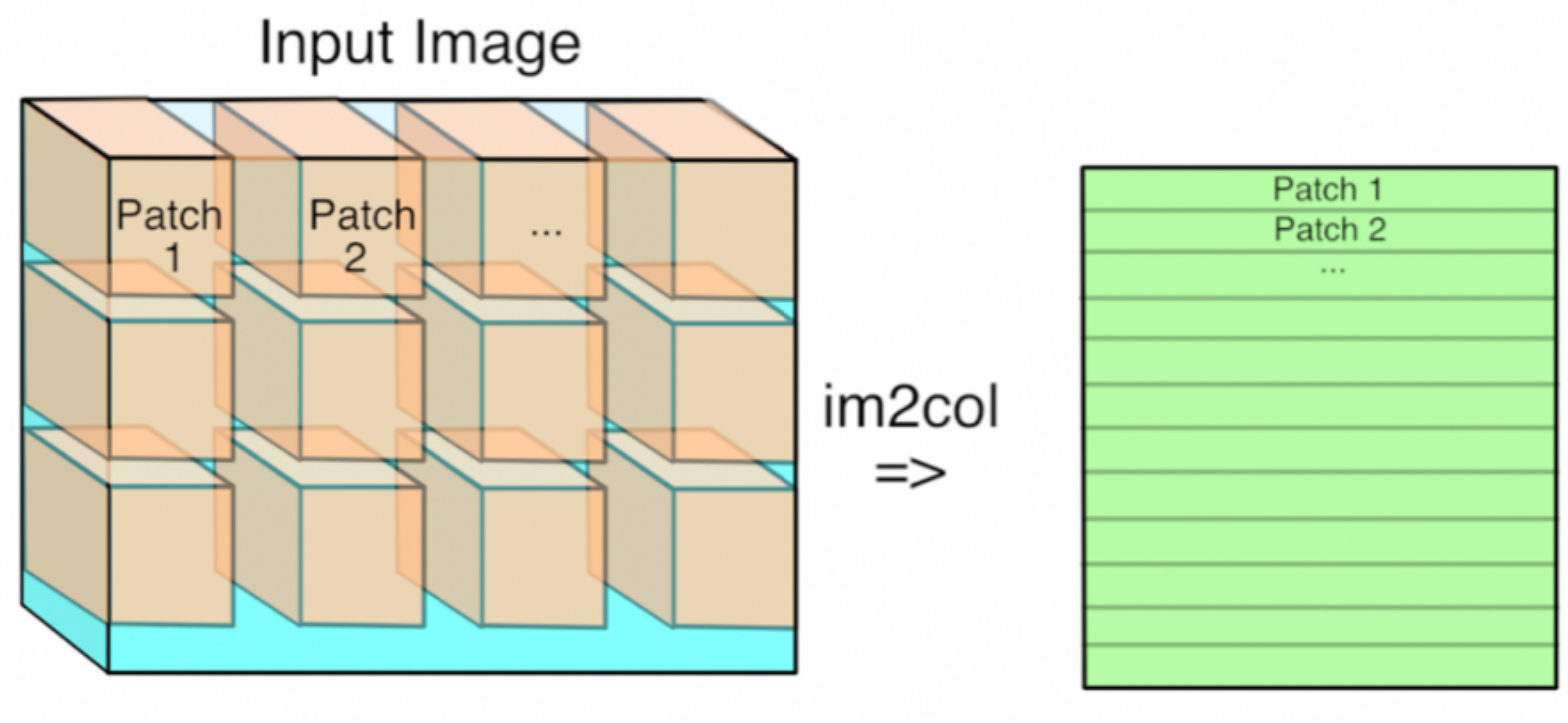


Convolutional Layers as Matrix Multiplication



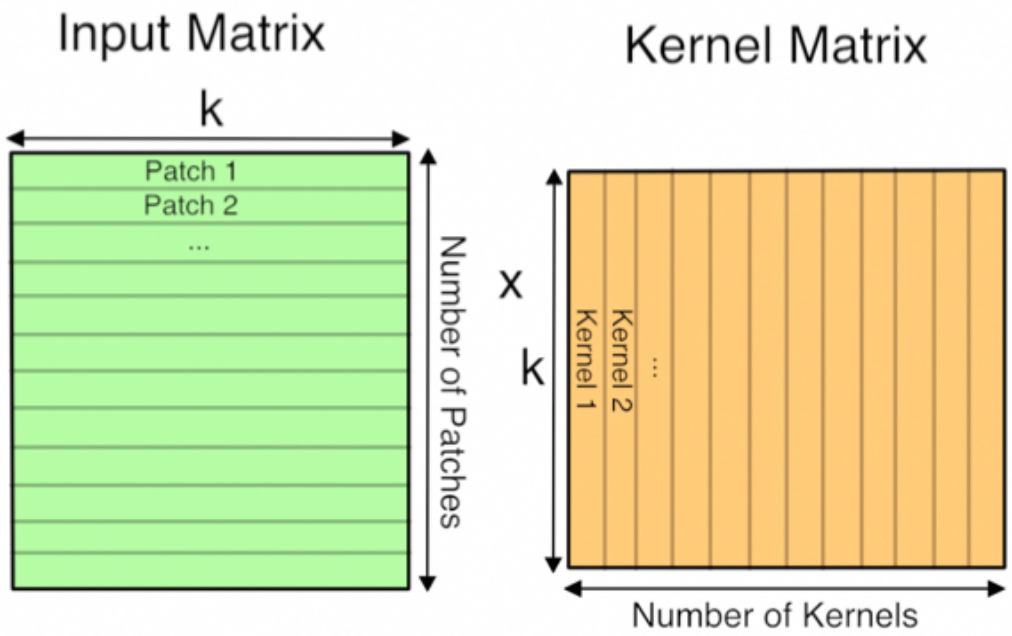
https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/

Convolutional Layers as Matrix Multiplication



https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/

Convolutional Layers as Matrix Multiplication



Pros?
Cons?

https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/

Questions?