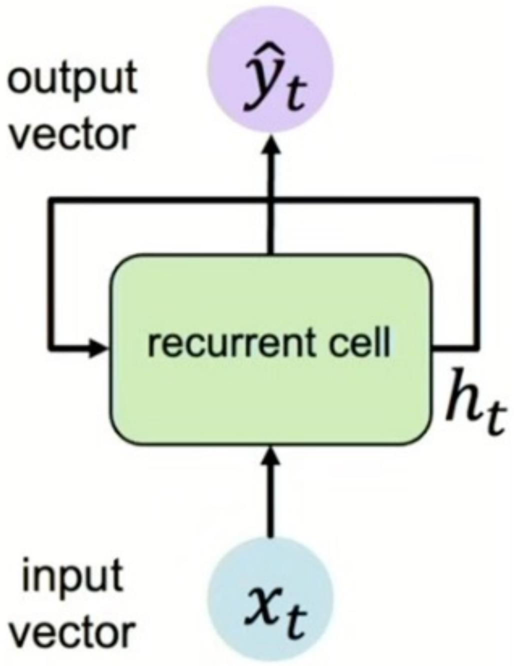
STATS 507
Data Analysis in Python

Week14: CNN in PyTorch

Dr. Xian Zhang

Recap: Sequential Modeling with RNN



Update hidden state (cell state)

past memory

$$h_t = \tanh(W_{hh}^T h_{t-1} + W_{xh}^T x_t)$$

Hidden State | Activation function | Previous hidden state | Input vector

$$\hat{y}_t = W_{hy}^T h_t$$

Note: in RNN, we use the SAME function and set of parameters at every time step.

Slide Credit: Ava Amini

Recap: LSTM

i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell
o: Output gate, How much to reveal cell
g: Gate gate (?), How much to write to cell

At each time step: we introduce 4 **gates** and a new **state** -> cell state

LSTM

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h \right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \tanh(c_t)$$

$$i_t = \sigma(x_t W_{xi} + h_{t-1} W_{hi}) + b_i$$

$$f_t = \sigma(x_t W_{xf} + h_{t-1} W_{hf}) + b_f$$

[0,1]

$$o_t = \sigma(x_t W_{xo} + h_{t-1} W_{ho}) + b_o$$

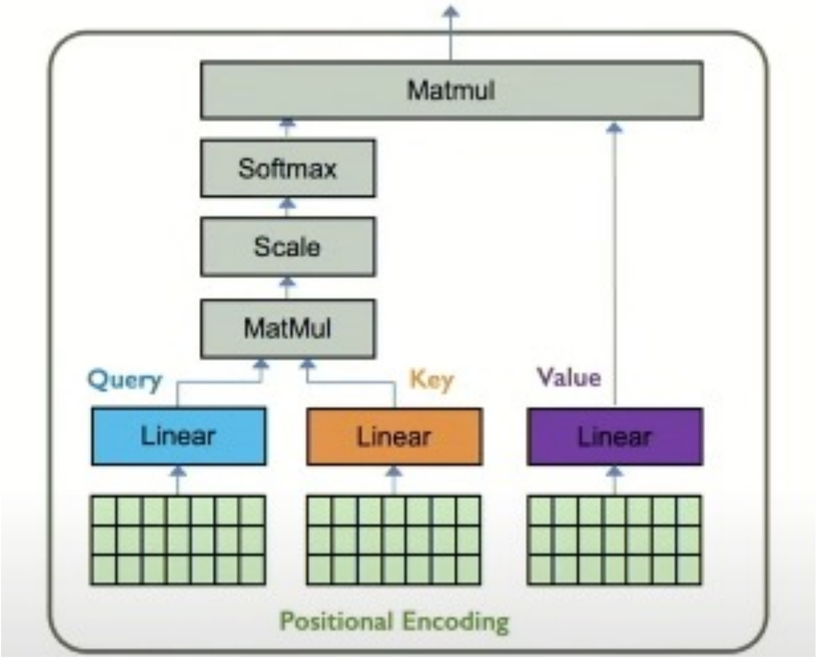
$$g_t = tanh(x_t W_{xg} + h_{t-1} W_{hg}) + b_g$$

[-1,1]

Learned weights!

Slide Credit: Justin Johnson

Recap: Attention and transformer



1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

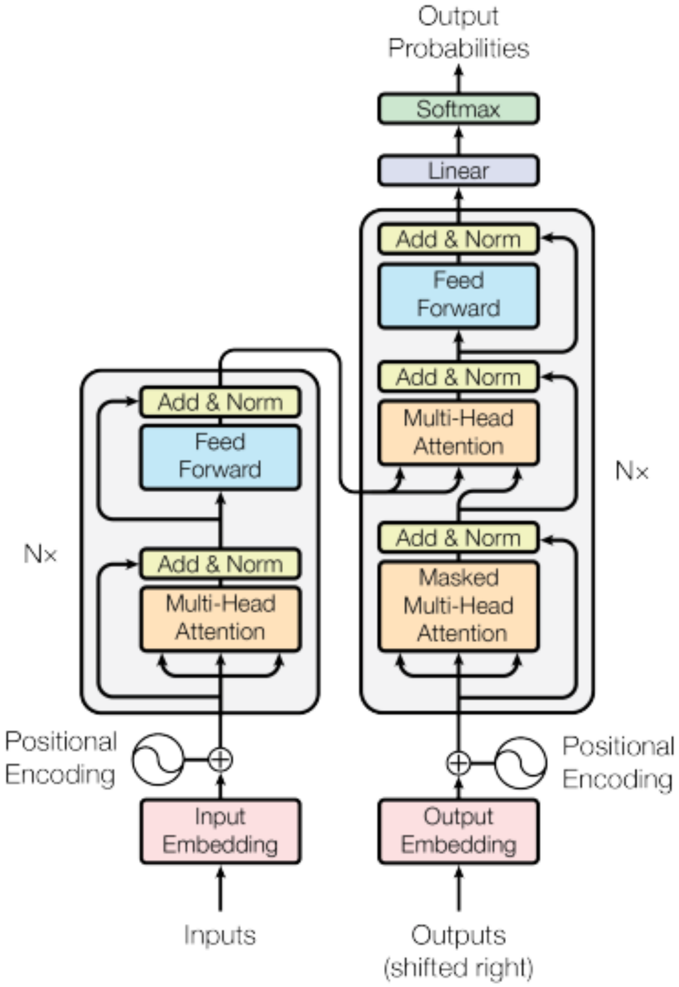
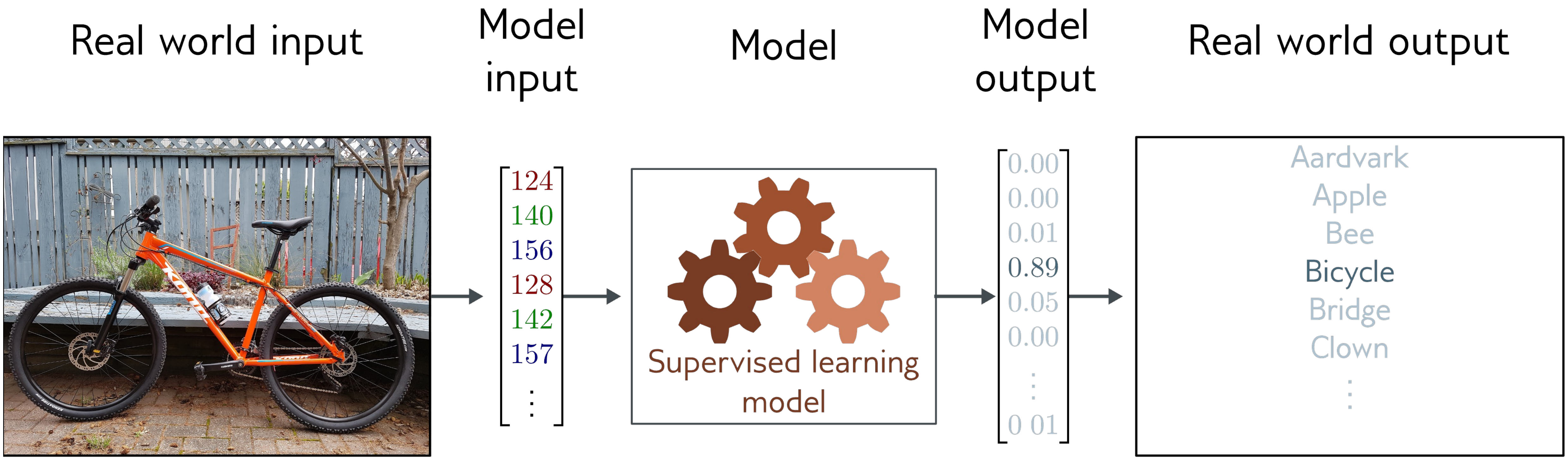


Figure 1: The Transformer - model architecture.

Today: teach machine to **see** with CNN

Vision: to see what is where.

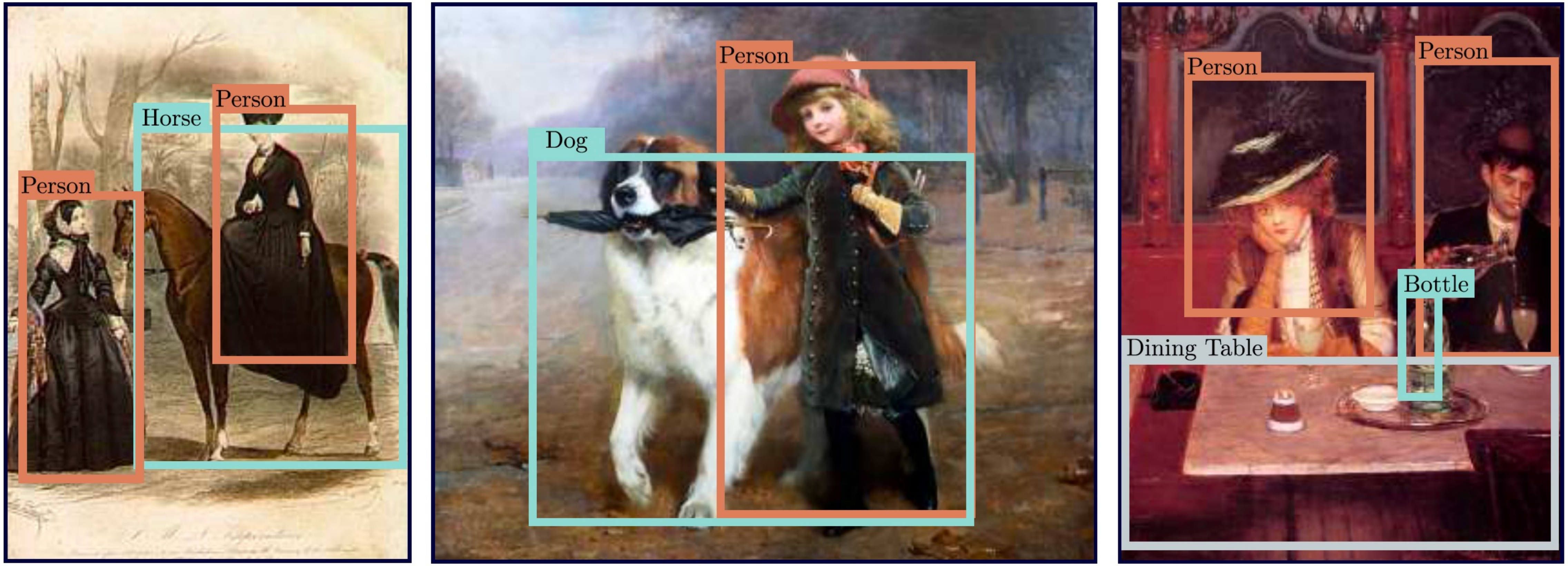
Image classification



- Multiclass classification problem (discrete classes, >2 possible classes)
- Convolutional network

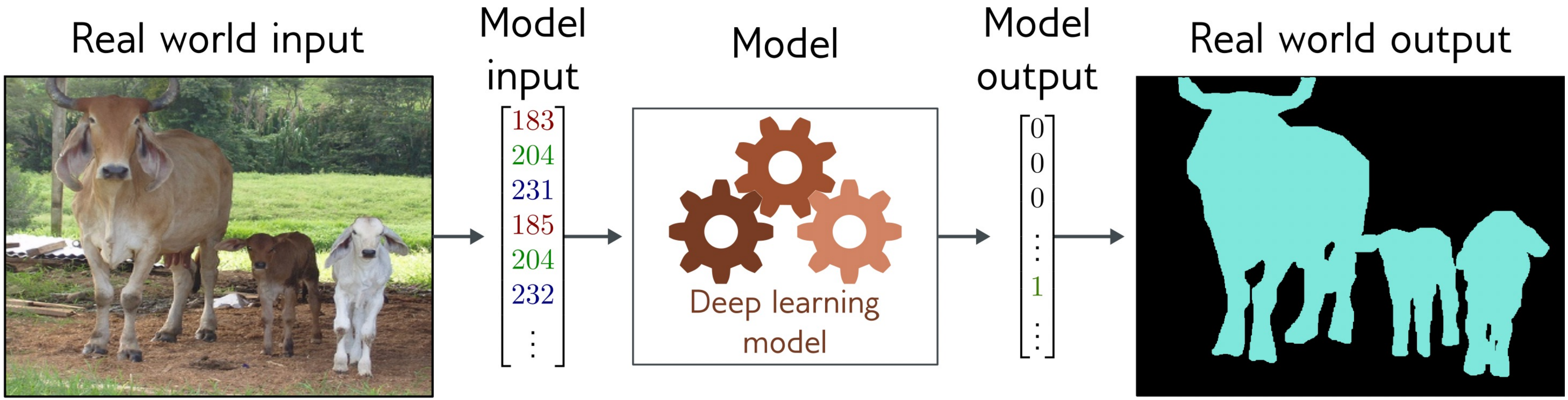
Slide Credit: Simon Prince

Object detection



Slide Credit: Simon Prince

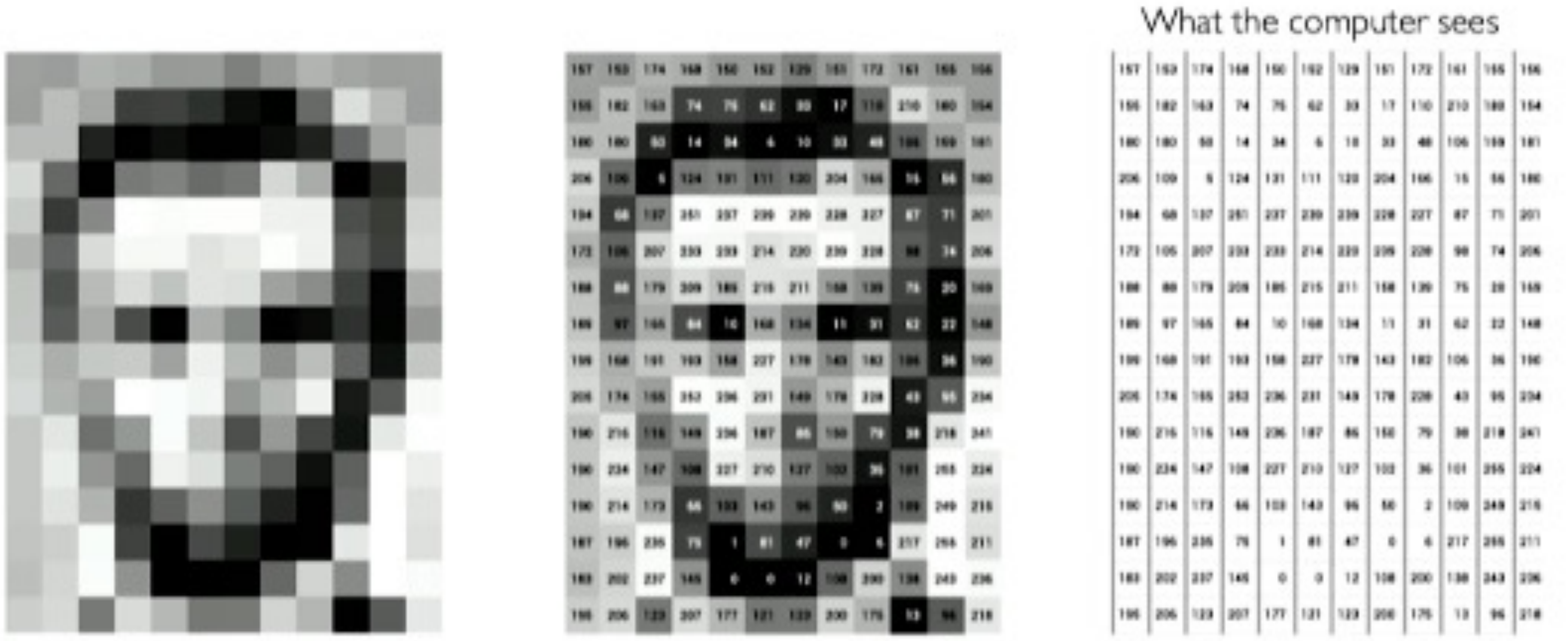
Image segmentation



- Multivariate binary classification problem (many outputs, two discrete classes)
- Convolutional encoder-decoder network

Slide Credit: Simon Prince

How can we teach a machine to see?



An image is just a matrix of numbers [0,255]!
i.e., 1080x1080x3 for an RGB image

- Manual Feature Extraction
- Through Learning

Slide Credit: Alexander Amini

Manual Feature Extraction

Problem?

How can we do better?

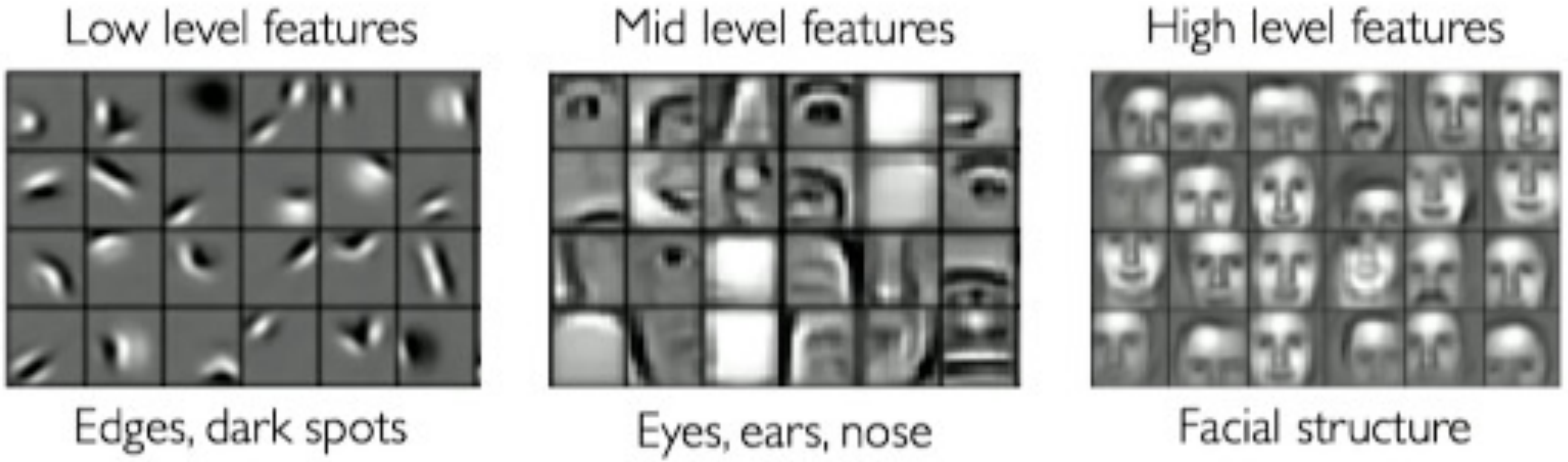
- Human face feature: eyes, noses, mouths…
- Detect small features first and then do the classify task
- Types of features are very **difficult** to define

Slide Credit: Alexander Amini

Learning with Neural Network

What we want: detect the features in image **automatically**

Ideally can learn a hierarchy of features.

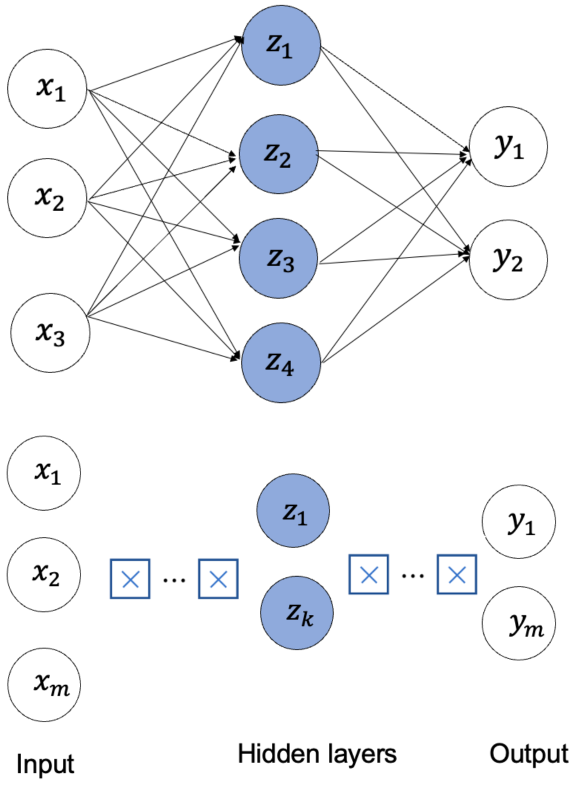


Slide Credit: Alexander Amini

Learning with MLP

Input	Fully Connected Neural Network	Problem?

- 2D image
- Vector of pixel values (**flattened**…)



- No Spatial information preserved
- Many many parameters…

How can we use the **spatial structure** in the input?

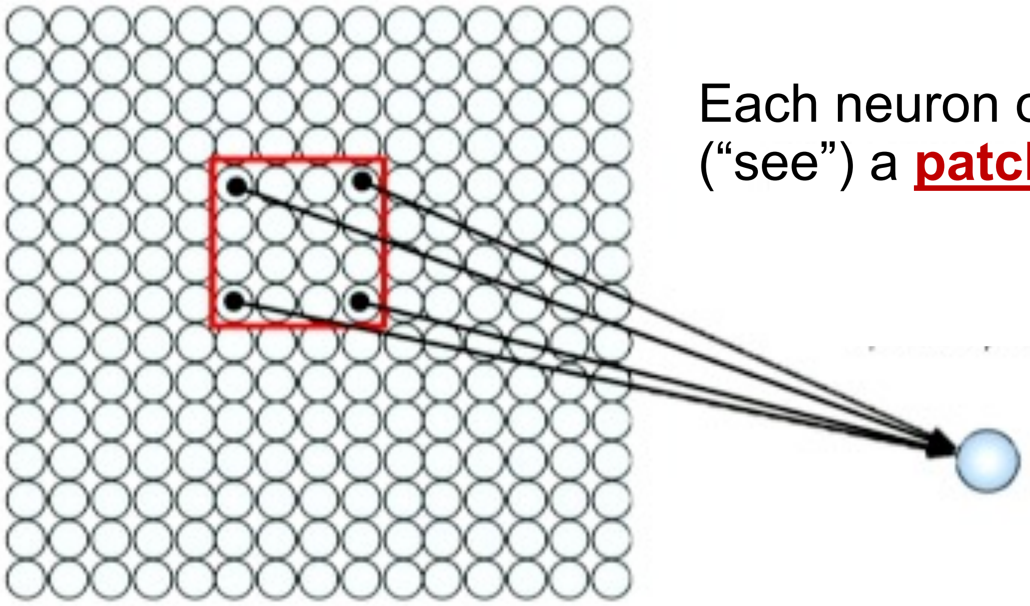
1. CNN basics

2. torchvision: vision in PyTorch

Learning with spatial structure: patching

Input	How does the perceptron act on the input?

- 2D image
- Array of pixel values
- **No flattening…**

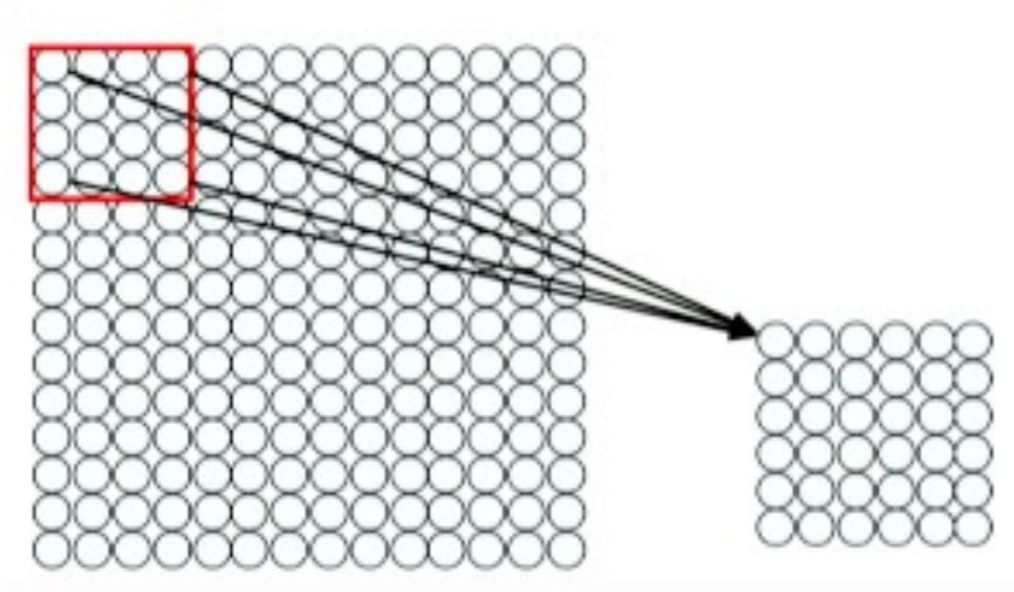


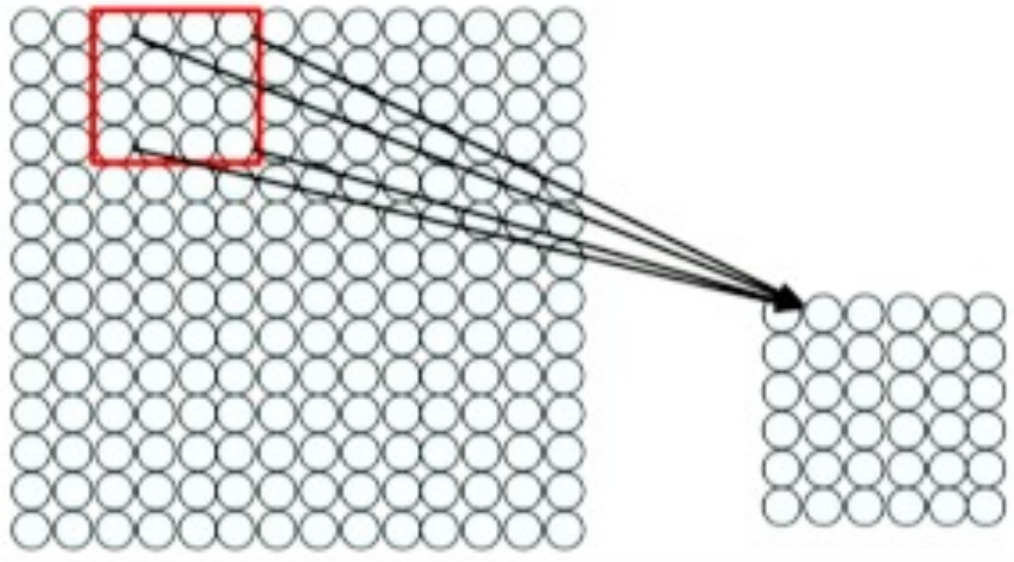
Each neuron only connects ("see") a **patch** of the input

Slide Credit: Alexander Amini

Sliding to see different patches

Connect patches in input layer to a single neuron in subsequent layer, using a sliding window to define connections.





Feature map

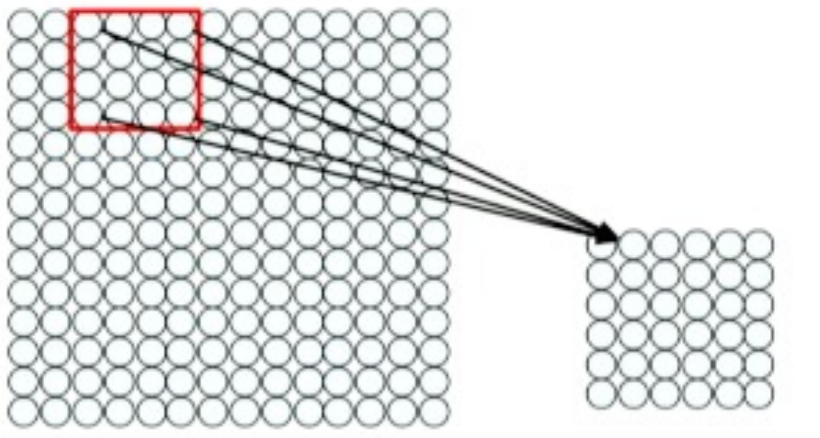
Same filter (weights) for different local regions across the image.

Slide Credit: Alexander Amini

Extract features with convolution

This allows us to extract local features with spatial info

- Apply a set of weights – a filter – to extract local features
- Use multiple filters to extract different local features
- Spatially share parameters of each filter

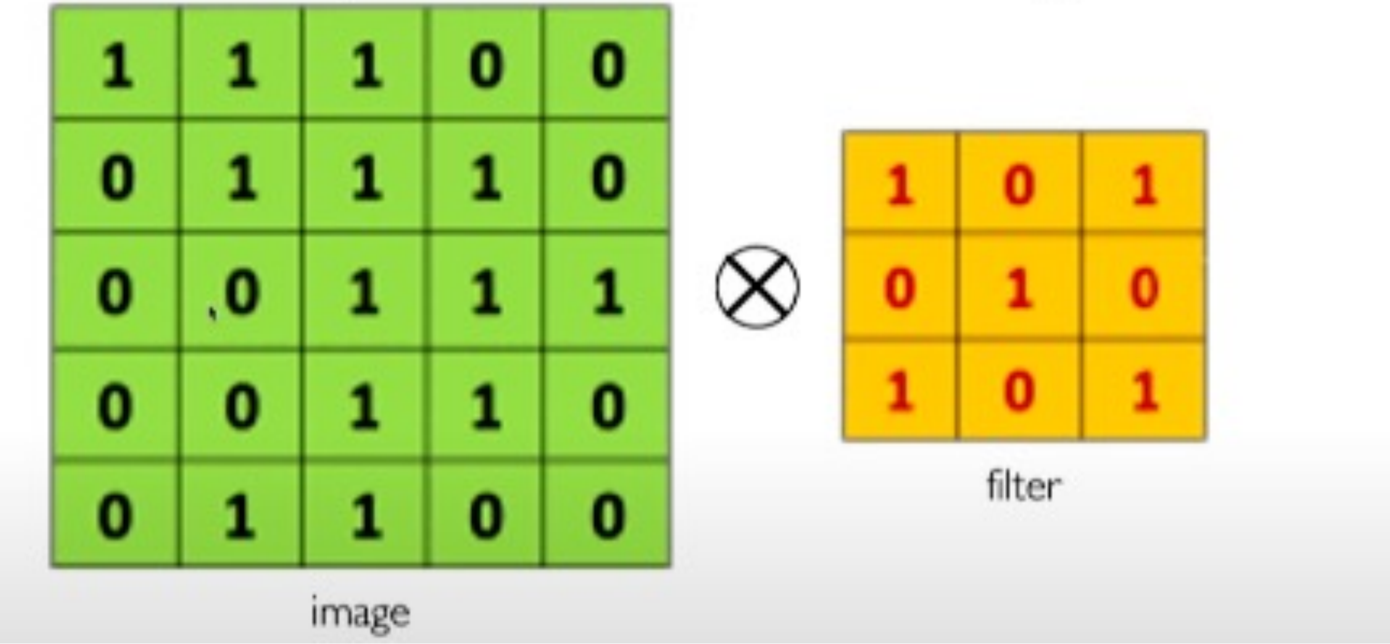


- Filter of 4*4: 16 weights
- Apply the same filter to 4*4 patches in input
- Shift by 2 pixels for next patch

Slide Credit: Alexander Amini

The Convolution Operation

Suppose we want to compute the convolution of a 5 * 5 image and a 3*3 filter..

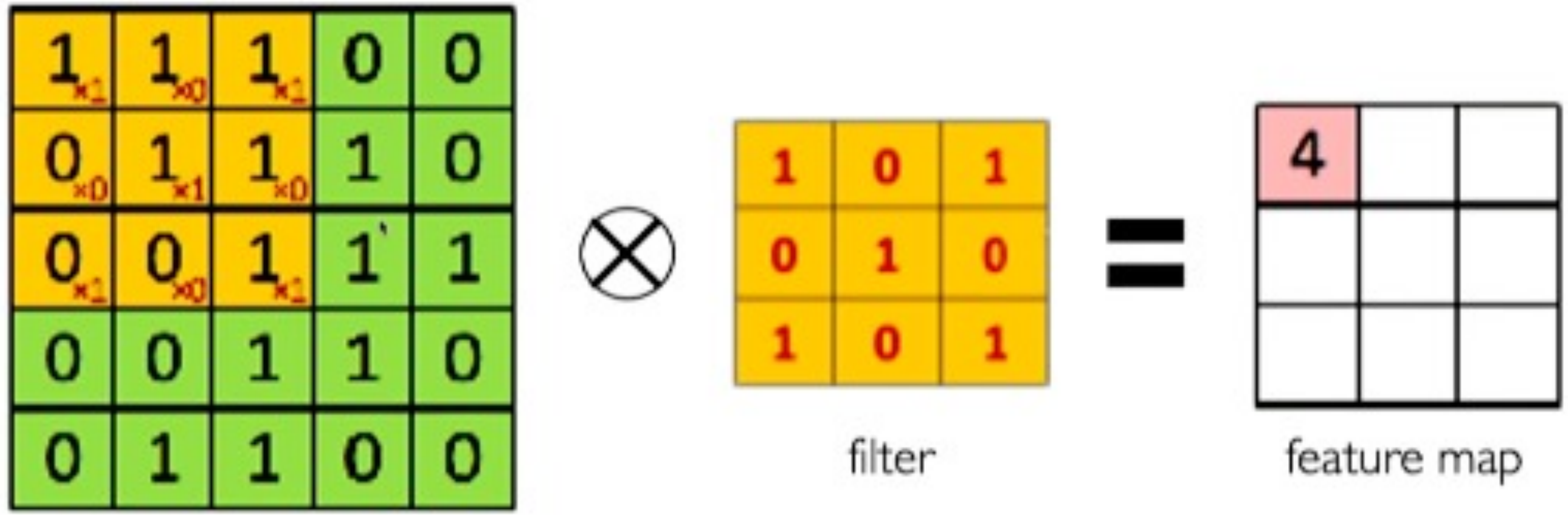


We slide the same 3*3 filter over the input image, element-wise multiply, and add the outputs to represent the ”local” feature…

Slide Credit: Alexander Amini

The Convolution Operation

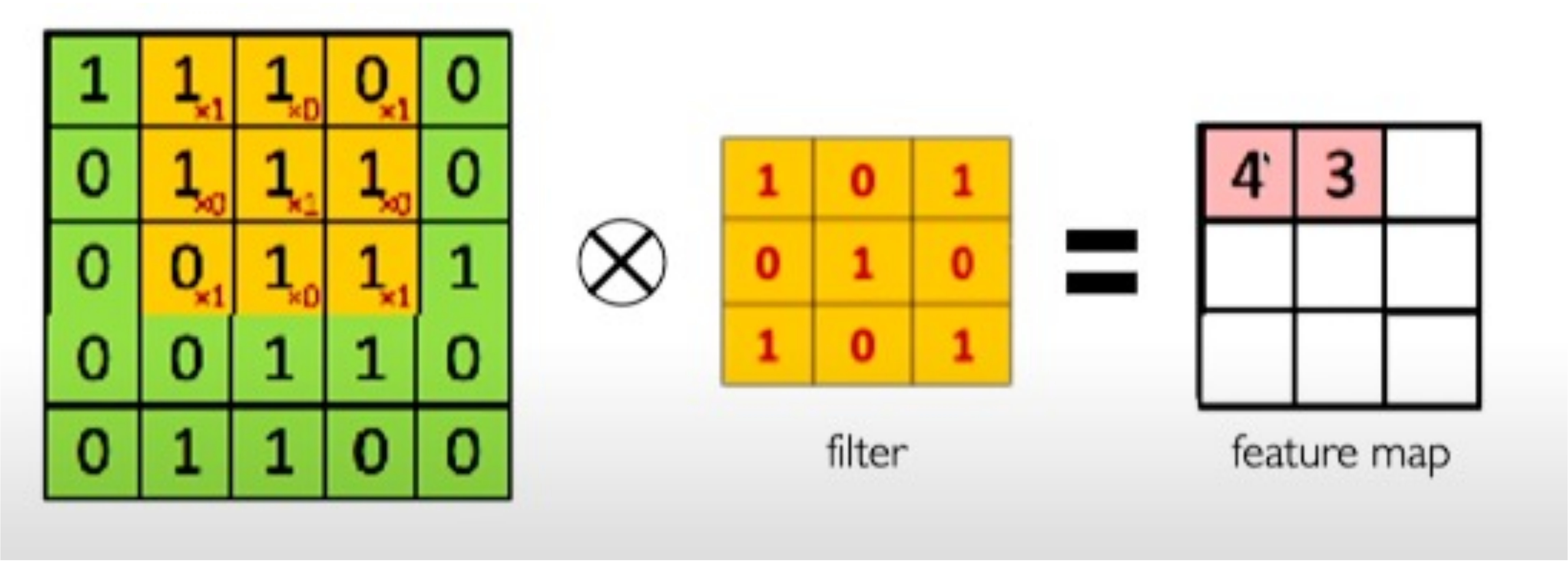
We slide the same 3*3 filter over the input image, element-wise multiply, and add the outputs to represent the ”local” feature…



Slide Credit: Alexander Amini

The Convolution Operation

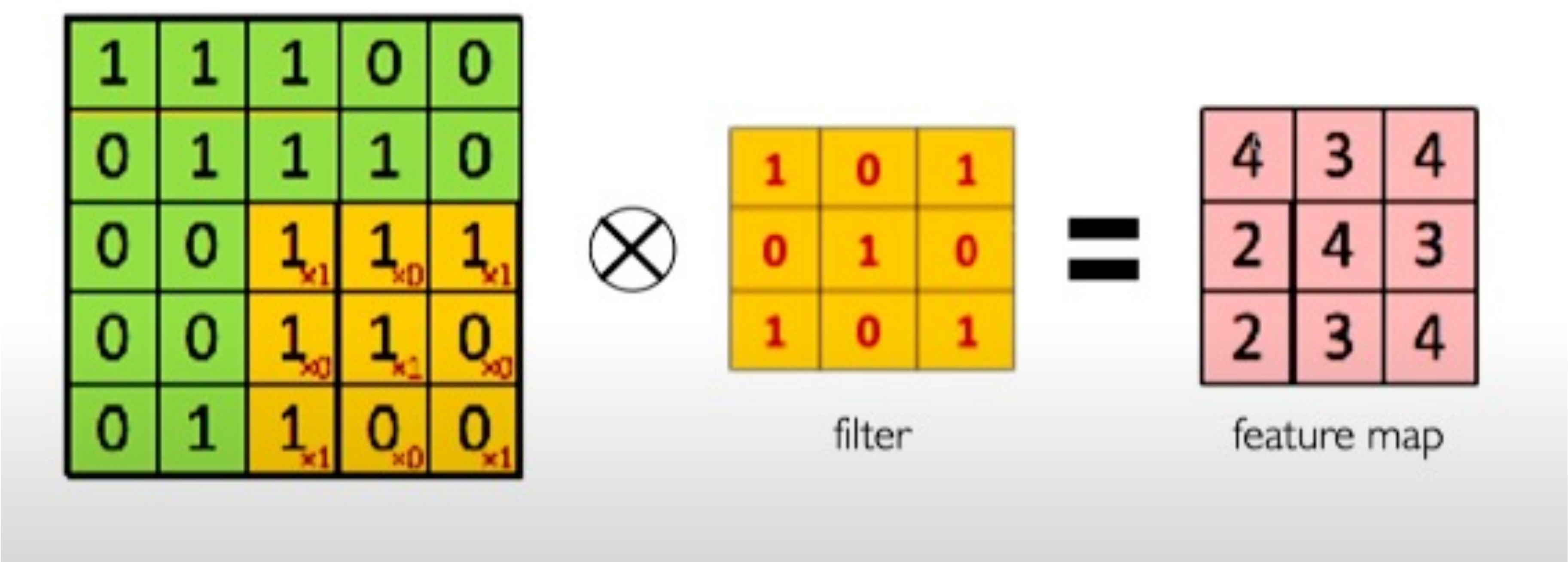
We slide the same 3*3 filter over the input image, element-wise multiply, and add the outputs to represent the ”local” feature…



Slide Credit: Alexander Amini

The Convolution Operation

We slide the same 3*3 filter over the input image, element-wise multiply, and add the outputs to represent the ”local” feature…



Slide Credit: Alexander Amini

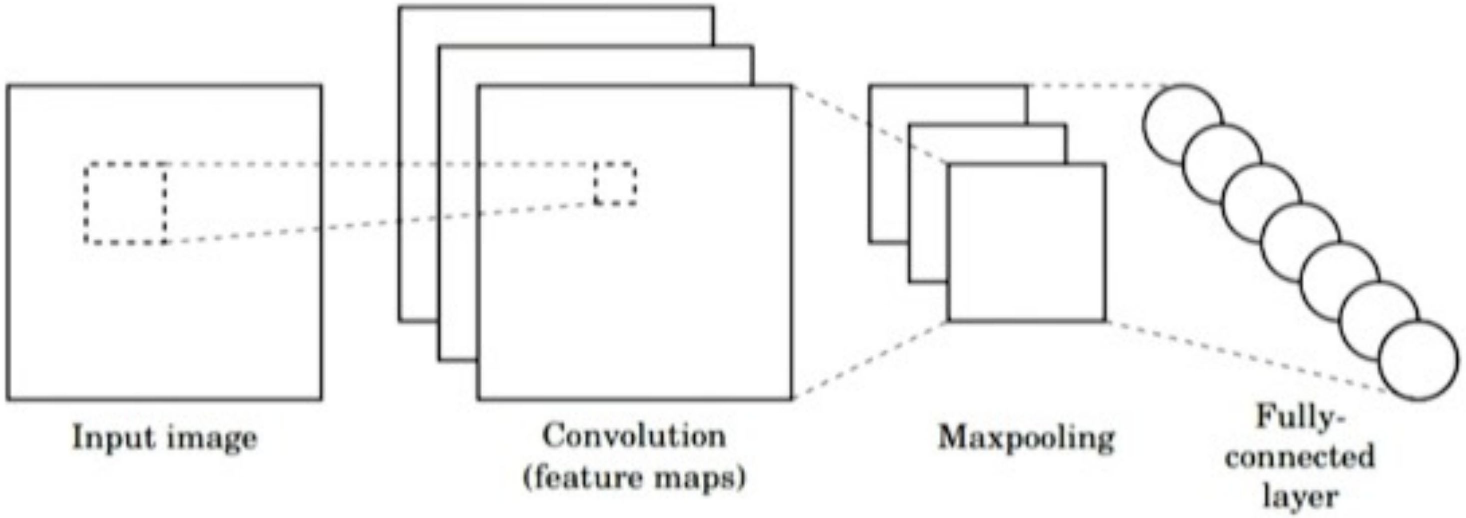
Feature maps with different filters

We can extract different local features with different filters.

One filter -> One feature maps…

Slide Credit: Alexander Amini

From Convolution to CNN

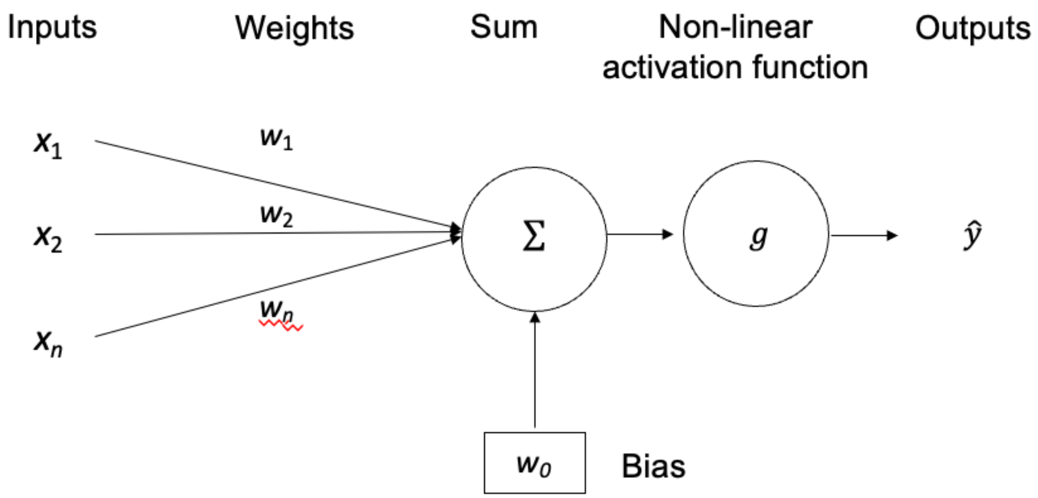


Three main operations:

1) Use convolution and filters to extract different feature maps
2) <u>Nonlinearity</u>: Often Relu
3) **<u>Pooling</u>**: downsample operation on each feature map.

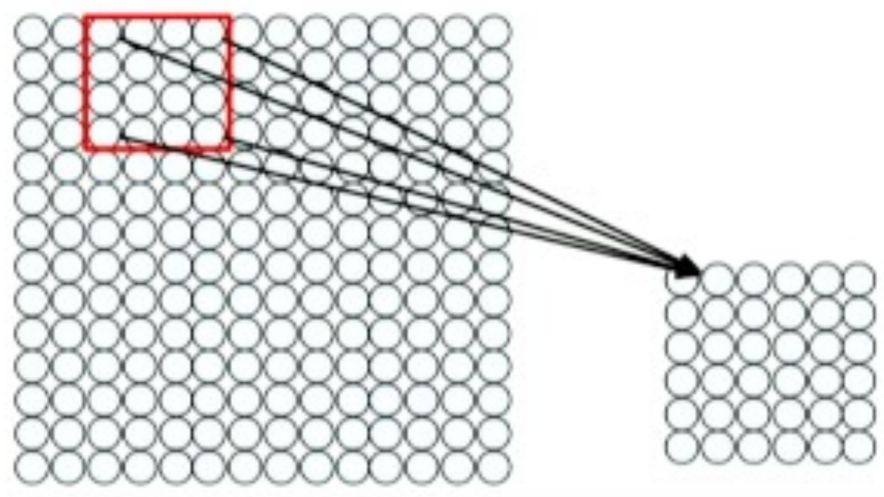
Slide Credit: Alexander Amini

Convolutional layers: to extract local features



$$\hat{y} = g(w_0 + \sum_{i=1}^{n} x_i w_i) = g(z)$$

For a neuron on a CNN layer:
1) Takes 2d Input from a patch
2) Compute weighted sum
3) Add bias



For a 4*4 filter
For neuron (p,q)

$$g(\sum_{i=1}^{4}\sum_{j=1}^{4} w_{ij}\, x_{i+p} x_{j+q} + w_0)$$

Slide Credit: Alexander Amini

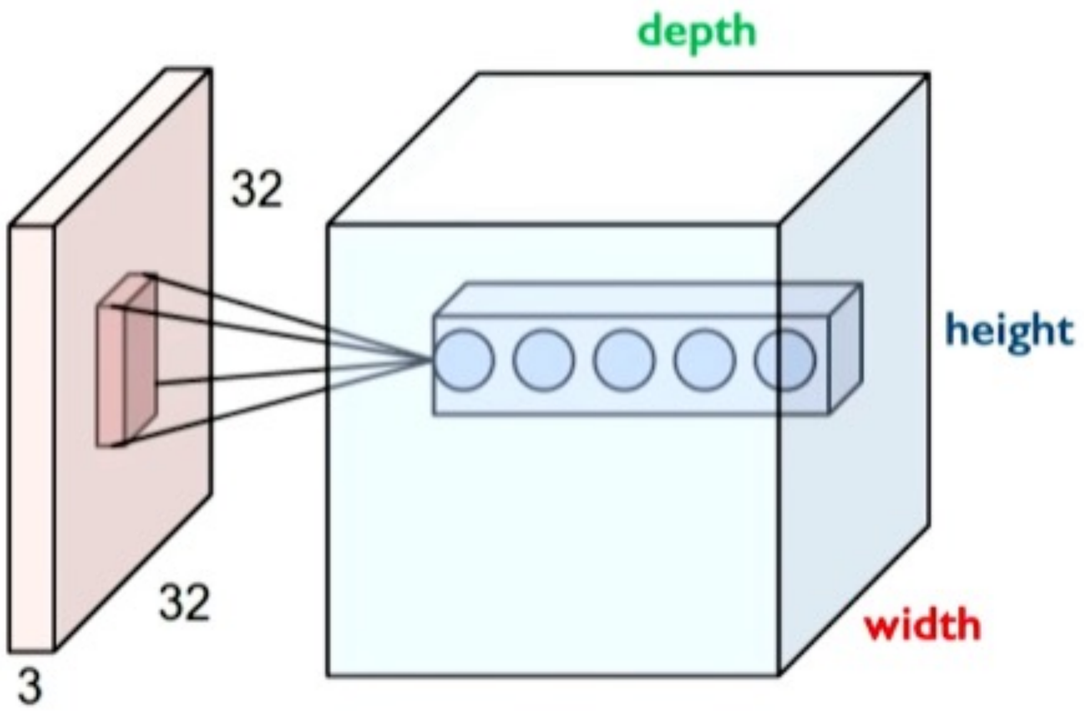
Convolutional layers in Pytorch

Conv2d

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

Parameters

- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int* or *tuple*) – Size of the convolving kernel
- **stride** (*int* or *tuple*, *optional*) – Stride of the convolution. Default: 1
- **padding** (*int*, *tuple* or *str*, *optional*) – Padding added to all four sides of the input. Default: 0
- **dilation** (*int* or *tuple*, *optional*) – Spacing between kernel elements. Default: 1
- **groups** (*int*, *optional*) – Number of blocked connections from input channels to output channels. Default: 1
- **bias** (*bool*, *optional*) – If `True`, adds a learnable bias to the output. Default: `True`
- **padding_mode** (*str*, *optional*) – `'zeros'`, `'reflect'`, `'replicate'` or `'circular'`. Default: `'zeros'`

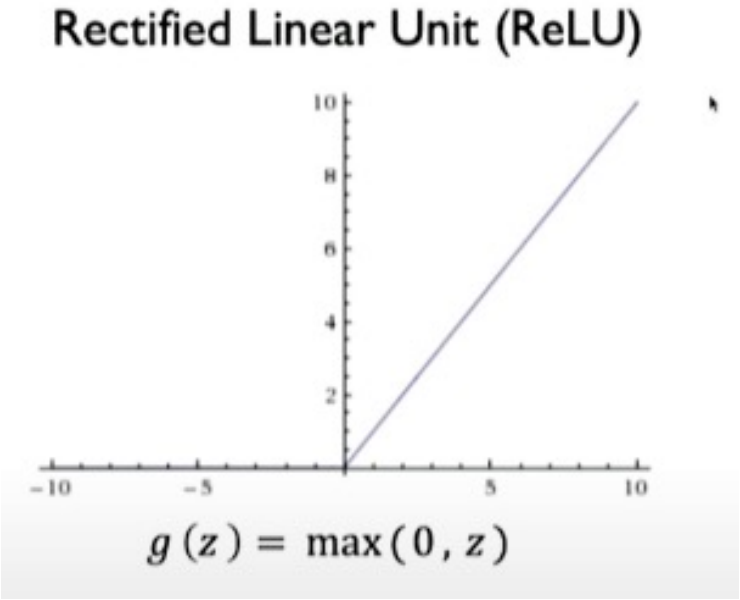


Depth: number of filters/feature maps

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

Introducing non-linearity

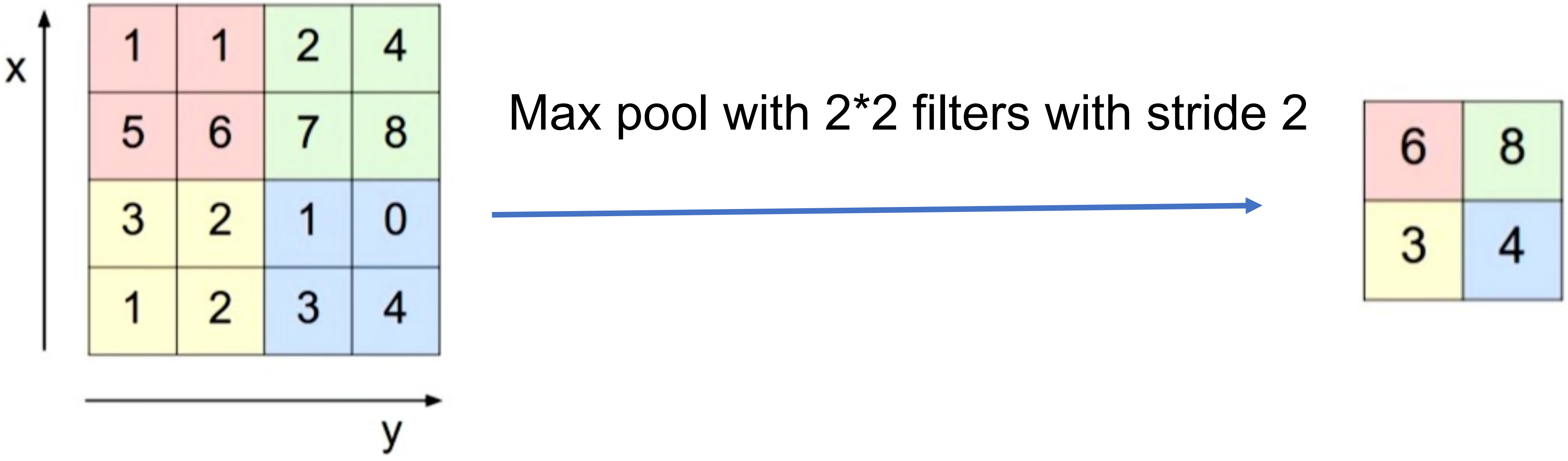
Apply after every convolutional operation (after convolutional layers)



Slide Credit: Alexander Amini

Pooling

Downsample and still preserve spatial structure?



https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

Slide Credit: Alexander Amini

Put it all together: in-class practice

Define a CNN for CIFAR-10 images classification using PyTorch

https://www.cs.toronto.edu/~kriz/cifar.html

1. CNN basics

2. torchvision: Vision in PyTorch

Torchvision overview

Torchvision is a library within PyTorch for **image** and **video** processing

https://pytorch.org/vision/stable/index.html

Dataset and Dataloader

Torchvision provides many built-in datasets in the torchvision.datasets module, as well as utility classes for building your own datasets.

Step 1: Load the datasets using torchvision.datasets
Step 2: Pass the dataset to a torch.utils.data.DataLoader which can load (and batch) multiple samples in parallel

```
batch_size = 4

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)
```

https://pytorch.org/vision/stable/datasets.html

Torchvision io: read/write image/video

The torchvision.io module provides utilities for decoding and encoding images and videos.

Read a JPEG or a PNG image into a 3d RGB tensor.

It will decode images straight into (uint8 [0, 255]) image tensors

Image	Video
• read_image	• read_video
• decode_image	• read_video_timestamps
• encode_jpeg	• write_video
• decode_jpeg	• VideoReader
• write_image	

https://pytorch.org/vision/stable/io.html

Optionally convert the image to the desired format…

Torchvision models/pretrained weights

The torchvision.models subpackage contains definitions of models for addressing different tasks, including:

- image classification
- pixelwise semantic segmentation
- object detection
- instance segmentation
- person keypoint detection
- video classification
- …

Use pre-defined model

```
from torchvision.models import resnet50, ResNet50_Weights

# Using pretrained weights:
resnet50(weights=ResNet50_Weights.IMAGENET1K_V1)
resnet50(weights="IMAGENET1K_V1")
resnet50(pretrained=True)  # deprecated
resnet50(True)  # deprecated

# Using no weights:
resnet50(weights=None)
resnet50()
```

Use pretrained weights to initialize a model

```
# Step 1: Initialize model with the best available weights
weights = ResNet50_Weights.DEFAULT
model = resnet50(weights=weights)
```

https://pytorch.org/vision/stable/models.html

Torchvision transform

Transforms can be used to transform or augment data for training or inference of different tasks.

- Rotate
- Crop
- Convert RGB to grayscale
- …

```
import torchvision.transforms as transforms
```

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
# In this case:
# mean = (0.5, 0.5, 0.5) - one value for each RGB channel
# std = (0.5, 0.5, 0.5) - one value for each RGB channel
```

Can use nn.sequential to compose operations

```
transforms = torch.nn.Sequential(
    CenterCrop(10),
    Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
)
```

https://pytorch.org/vision/stable/transforms.html

In class practice solution

```
import torch.nn as nn
import torch.nn.functional as F


class Net(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


net = Net()
```

Other things

Final project proposal due today!

HW8 due this week.

Coming next:

Last lecture in class.

Come to class if you need my help for final project.