# CS167: Machine Learning

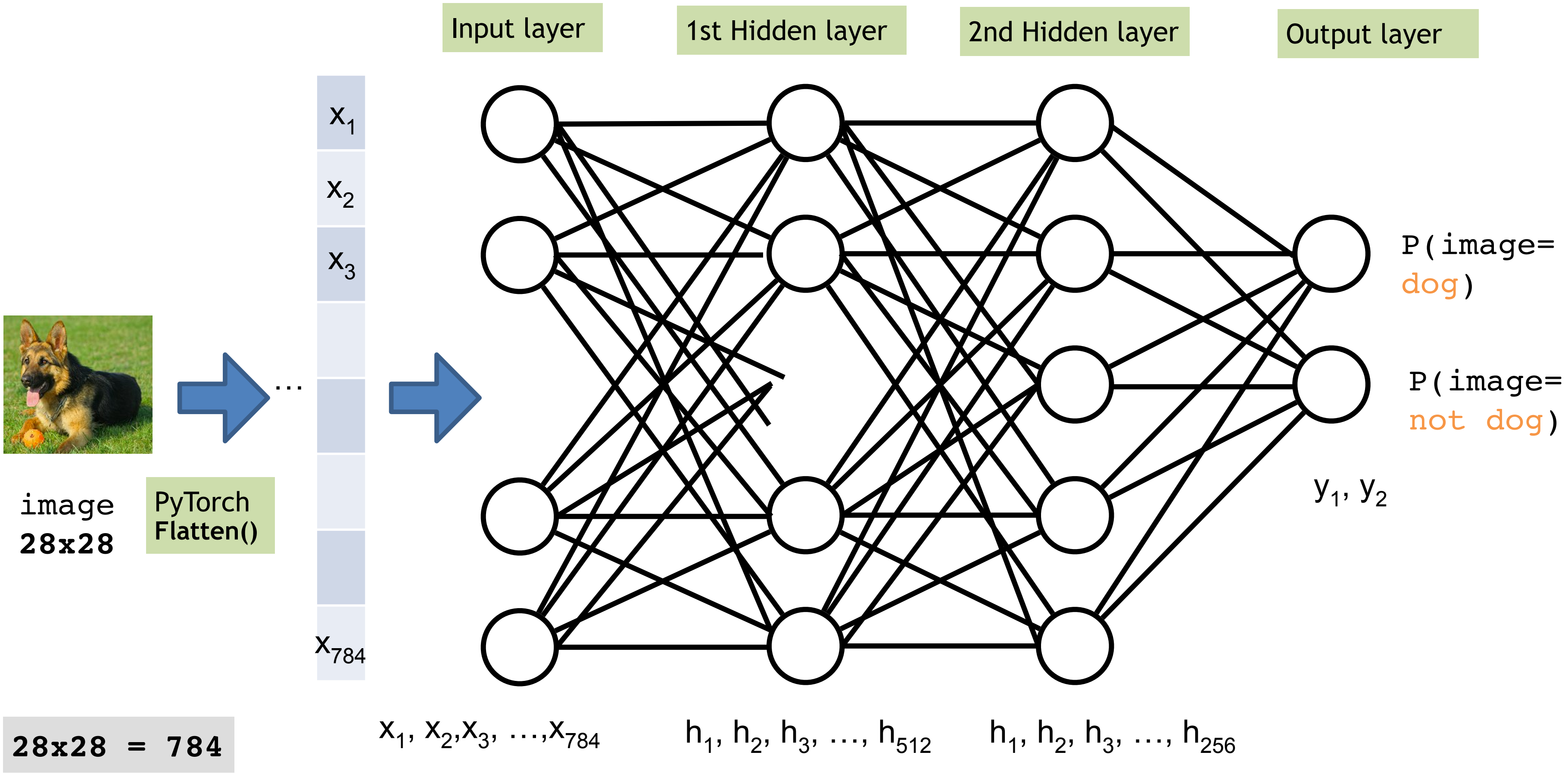
## Deep Learning
## Convolutional Neural Network (CNN)

Tuesday, April 16th, 2024

Drake UNIVERSITY

# Recap: Multilayer Perceptron (MLP)

- A **multilayer perceptron** is the simplest type of neural network. It consists of perceptrons (aka nodes, neurons) arranged in layers



Input layer

1st Hidden layer

2nd Hidden layer

Output layer

$x_1$

$x_2$

$x_3$

…

$x_{784}$

image
**28x28**

**PyTorch Flatten()**

P(image= dog)

P(image= not dog)

$y_1$, $y_2$

**28x28 = 784**

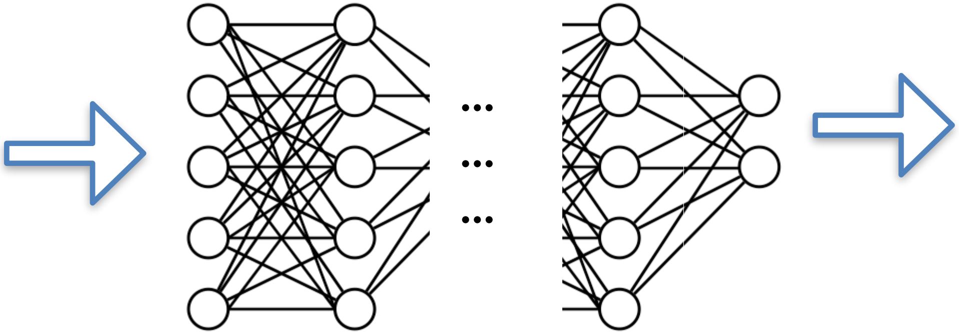
$x_1$, $x_2$,$x_3$, …,$x_{784}$

$h_1$, $h_2$, $h_3$, …, $h_{512}$

$h_1$, $h_2$, $h_3$, …, $h_{256}$

# Recap: MLP Summary

- MLPs are effective in finding non-linear patterns in the training data
  - can be applied to **regression** or **classification**.
  - **backpropagation** tunes the weights over a neural network using **gradient descent** to iteratively reduce the error in the network
  - **overfitting** the training data is common and is important to avoid
  - the following parameters should be tuned when using MLPs:
    - number of epochs
    - structure of the network (depth, width)
    - activation function
    - eta (learning rate)



# Recap: List of PyTorch Functions We Need

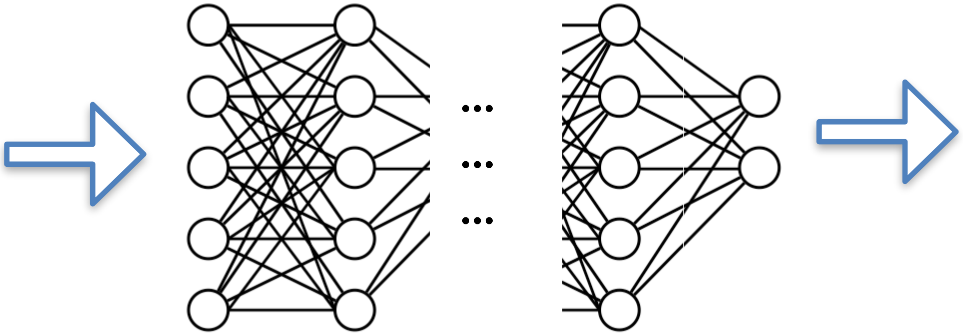
- nn.Linear()
  creates the dense connections between two adjacent layers (*left layer* and *right layer*)
  just provide **#neurons_left_layer** and **#neurons_right_layer**
- nn.ReLU()
- nn.Softmax()
- nn.flatten()
- nn.Sequential()
- nn.CrossEntropyLoss()
- torch.optim.SGD

- Let's jump into the notebook for a detailed discussion.

# Poll: MLP Summary

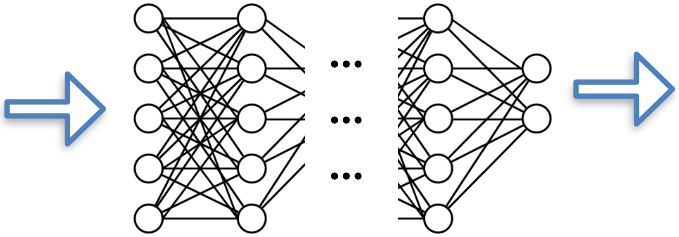
- Finish the MLP poll below:

https://forms.gle/ho5ffcRQ9CAF3oBy9



# What’s Next?

- A **multilayer perceptron** (MLP) is the simplest type of neural network. It consists of perceptrons (aka nodes, neurons) arranged in layers



- A **multilayer perceptron** (MLP) is just the tip of the iceberg; plenty of other neural network variants exist.

# Today's Agenda

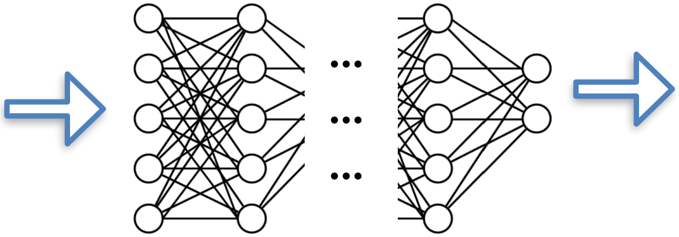
- Deep Learning

- Convolutional Neural Network (CNN)
  - Convolution operation
  - Nonlinearity
  - Pooling operation
  - CNN: convolutional layer + nonlinearity + pooling layer

# Deep Learning

**Deep learning** is a subset of machine learning that relies primarily on neural networks, and most of what is considered AI today is accomplished with deep learning eg,

- recognizing, finding, and enumerating objects in an image
- changing contents of an image
- language translation
- audio/speech translation
- new content generation (eg, computer code, art, music)
- recommendation systems
- autonomous driving
- and numerous others



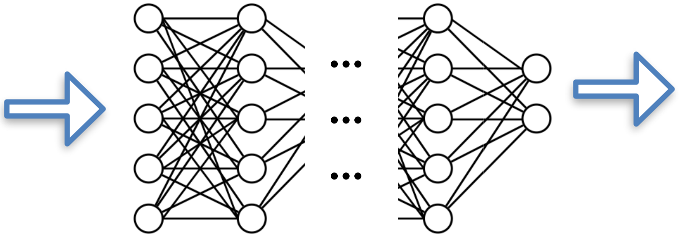
# Traditional Machine Learning Models vs. Deep Learning Models

## Traditional machine learning models

- k-Nearest Neighbor (k-NN)
- Decision Trees (DT)
- Random Forests (RF)
- Support Vector Machines (SVM)

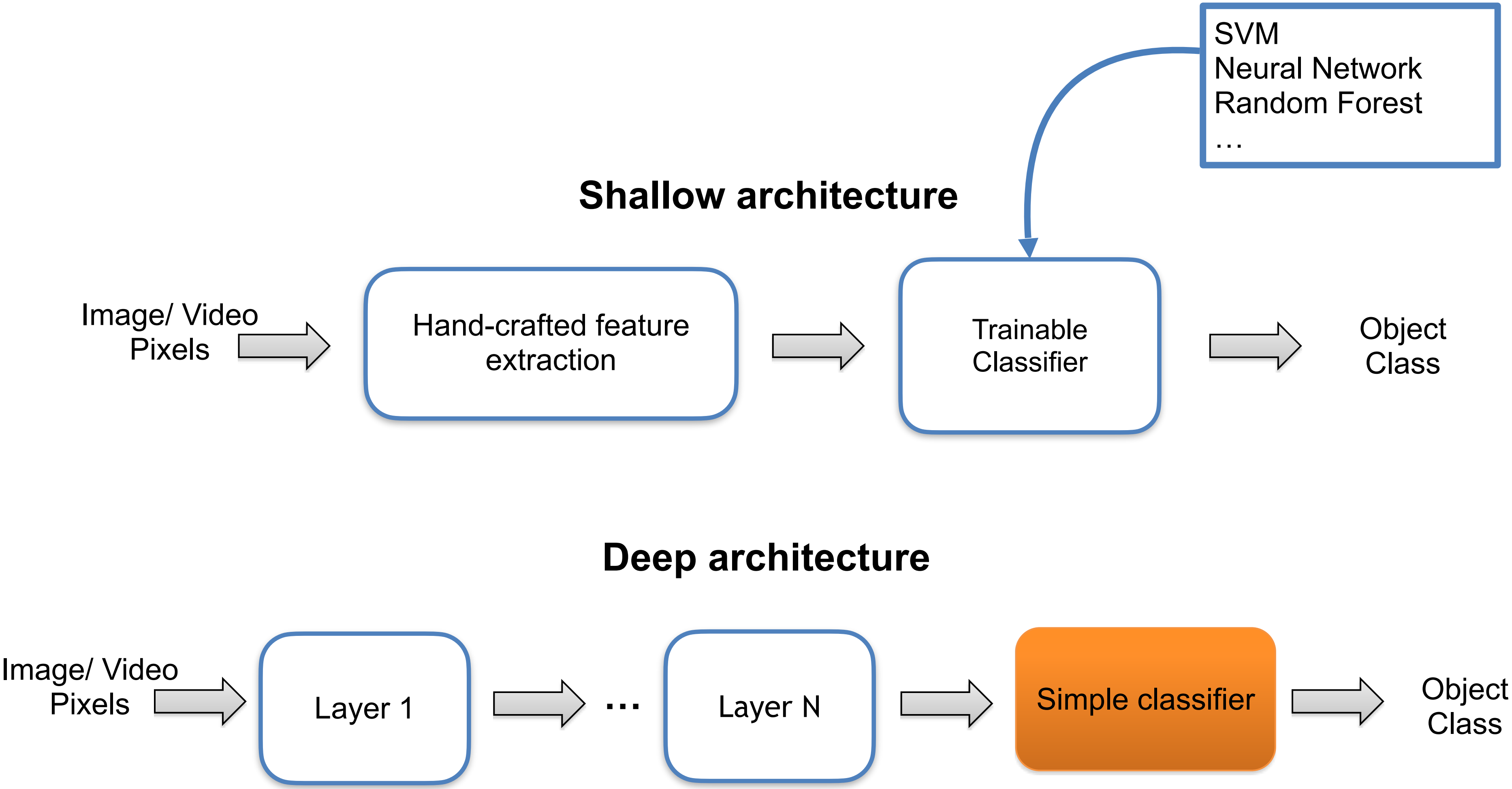
## Deep learning models

- Multilayer Perceptrons (MLP)
  - simplest
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
  - Vanilla RNN, LSTM, GRU
- Transformer
- Generative Adversarial Network (GAN)
- Variational Auto Encoder (VAE)
- Graph Neural Network (GNN)



- These models are neural network variants
- Effective in finding non-linear patterns in the training data

# Traditional Machine Learning Models vs. Deep Learning Models



# Deep Learning Models

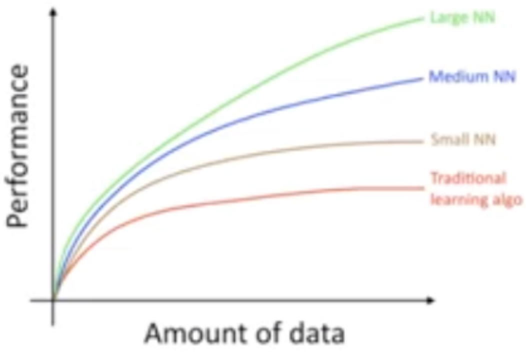
- **Convolutional Neural Network (CNN)**
  - good for computer vision (CV) tasks ie, that AI domain that deals with image data understanding

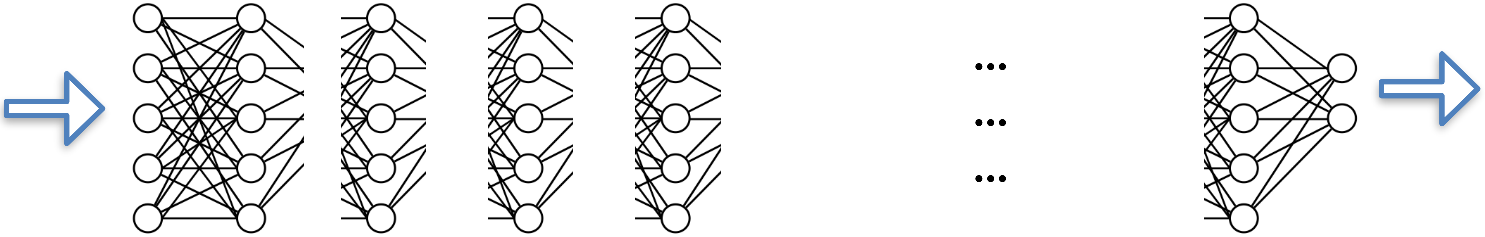
- **Recurrent Neural Network (RNN, LSTM, GRU)**
  - Good for natural language processing (NLP) tasks, AI domain that deals with textual data understanding

- **Transformer**
  - rising star DL model; it had its inception in Natural Language Processing domain but is now gradually taking over all other AI domains such as Computer Vision, Audio/Speech, Robotics

# Deep Learning Models

- With more data and larger, deeper neural networks (containing more parameters), they tend to learn better, thereby improving performance

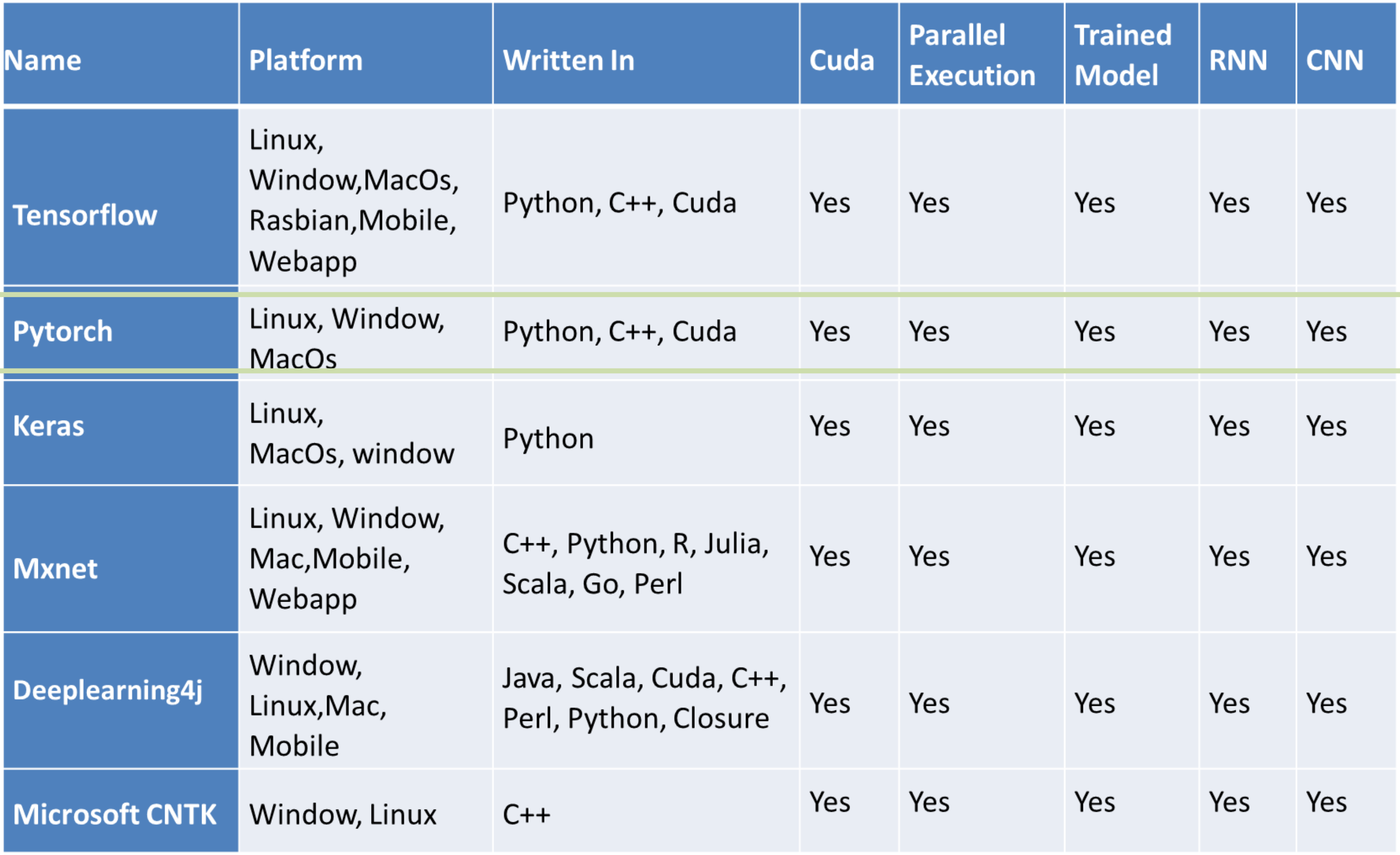




Deeper Neural Network

# Deep Learning Frameworks

Deep Learning Frameworks: libraries that make implementing deep learning easier--building models, training them, visualizing the data and training process, saving/loading models, utilizing GPU, etc.



| Name | Platform | Written In | Cuda | Parallel Execution | Trained Model | RNN | CNN |
|---|---|---|---|---|---|---|---|
| Tensorflow | Linux, Window,MacOs, Rasbian,Mobile, Webapp | Python, C++, Cuda | Yes | Yes | Yes | Yes | Yes |
| Pytorch | Linux, Window, MacOs | Python, C++, Cuda | Yes | Yes | Yes | Yes | Yes |
| Keras | Linux, MacOs, window | Python | Yes | Yes | Yes | Yes | Yes |
| Mxnet | Linux, Window, Mac,Mobile, Webapp | C++, Python, R, Julia, Scala, Go, Perl | Yes | Yes | Yes | Yes | Yes |
| Deeplearning4j | Window, Linux,Mac, Mobile | Java, Scala, Cuda, C++, Perl, Python, Closure | Yes | Yes | Yes | Yes | Yes |
| Microsoft CNTK | Window, Linux | C++ | Yes | Yes | Yes | Yes | Yes |

Reference: https://www.kaggle.com/getting-started/156399

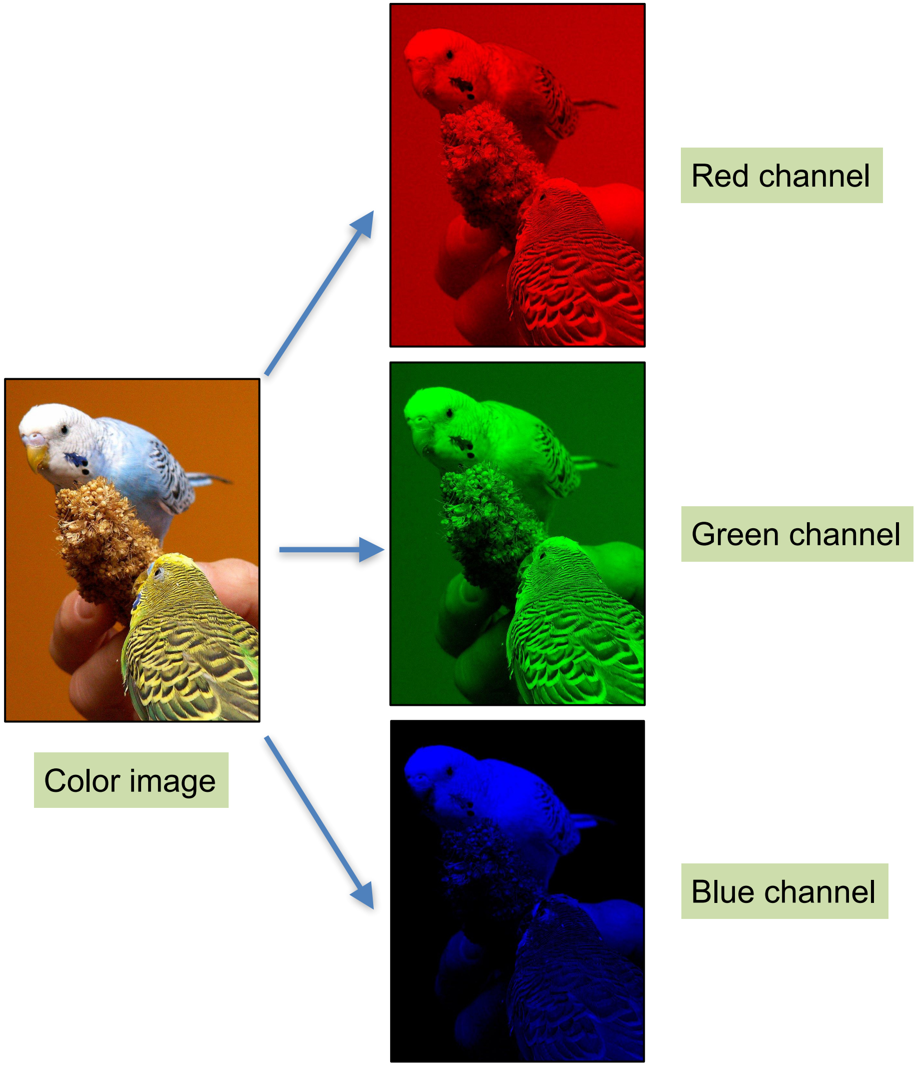
# Today’s Agenda

- Deep Learning
- Convolutional Neural Network (CNN)
  - Convolution operation
  - Nonlinearity
  - Pooling operation
  - CNN: convolutional layer + nonlinearity + pooling layer

# Convolutional Neural Network (CNN)

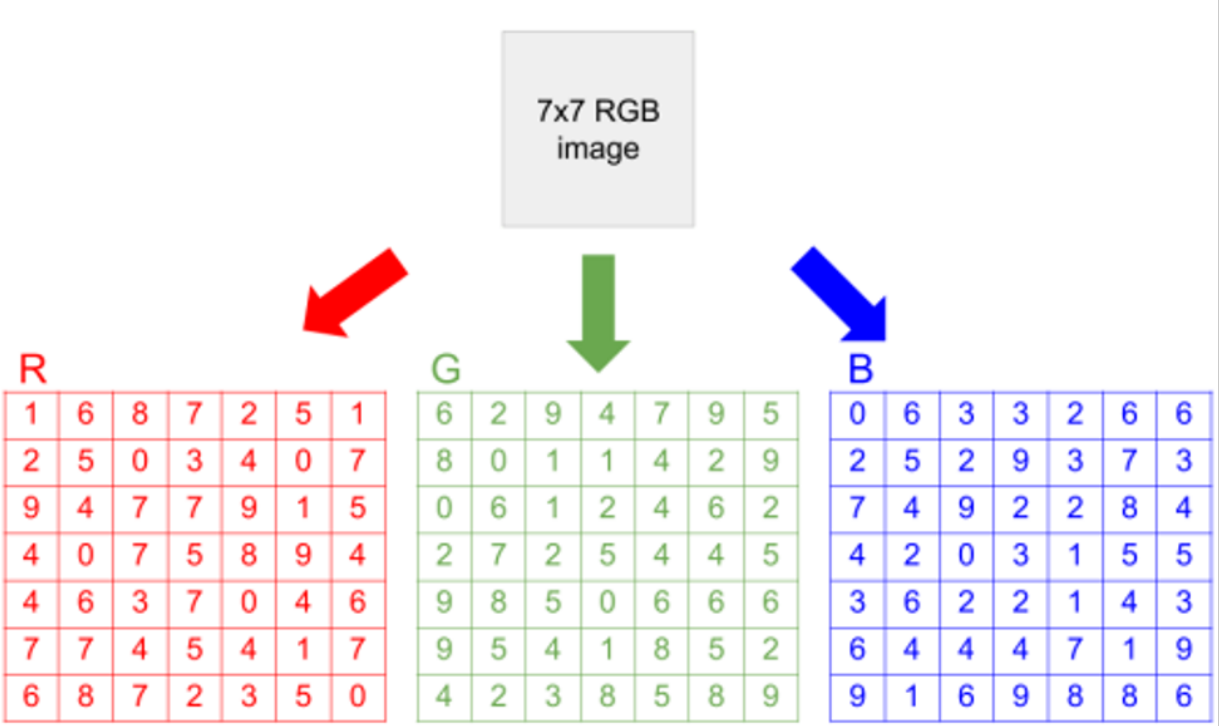
- A **convolutional neural network** that applies convolutional filters on grid-like input such as a image

- Image data is represented as a two-dimensional grid of pixels, either grayscale (monochromatic) or color (RBG)
    - each pixel corresponds to one or multiple numeric values: if it's grayscale, it is one number, if it's color, it corresponds to 3 numbers (a red, a green and a blue value)



# Convolutional Neural Network (CNN)

- A **convolutional neural network** that applies **convolutional filters** on grid-like input such as a image



R

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 6 | 8 | 7 | 2 | 5 | 1 |
| 2 | 5 | 0 | 3 | 4 | 0 | 7 |
| 9 | 4 | 7 | 7 | 9 | 1 | 5 |
| 4 | 0 | 7 | 5 | 8 | 9 | 4 |
| 4 | 6 | 3 | 7 | 0 | 4 | 6 |
| 7 | 7 | 4 | 5 | 4 | 1 | 7 |
| 6 | 8 | 7 | 2 | 3 | 5 | 0 |

G

| | | | | | | |
|---|---|---|---|---|---|---|
| 6 | 2 | 9 | 4 | 7 | 9 | 5 |
| 8 | 0 | 1 | 1 | 4 | 2 | 9 |
| 0 | 6 | 1 | 2 | 4 | 6 | 2 |
| 2 | 7 | 2 | 5 | 4 | 4 | 5 |
| 9 | 8 | 5 | 0 | 6 | 6 | 6 |
| 9 | 5 | 4 | 1 | 8 | 5 | 2 |
| 4 | 2 | 3 | 8 | 5 | 8 | 9 |

B

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 6 | 3 | 3 | 2 | 6 | 6 |
| 2 | 5 | 2 | 9 | 3 | 7 | 3 |
| 7 | 4 | 9 | 2 | 2 | 8 | 4 |
| 4 | 2 | 0 | 3 | 1 | 5 | 5 |
| 3 | 6 | 2 | 2 | 1 | 4 | 3 |
| 6 | 4 | 4 | 4 | 7 | 1 | 9 |
| 9 | 1 | 6 | 9 | 8 | 8 | 6 |

# Convolutional Neural Network (CNN)

- A **convolutional neural network** that applies **convolutional filters** on grid-like input such as a image

Nearby pixels are more strongly related than distant ones.

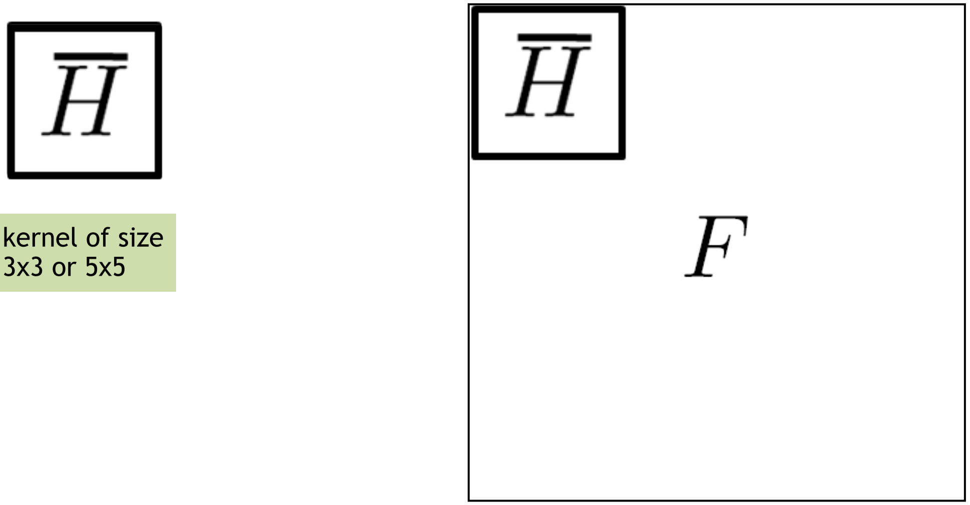
Objects are built up out of smaller parts.

- In order to capture the local dependence of images, we use **convolutional filters.** A convolutional filter, aka kernel:
  - is smaller than the input data (usually 3x3 or 5x5 or 7x7)
  - uses dot product multiplication between a piece of the input that is the size of the filter and the filter
  - scans over the image from the upper left to the bottom right

# Convolution Operation

- What does a convolution operation do?

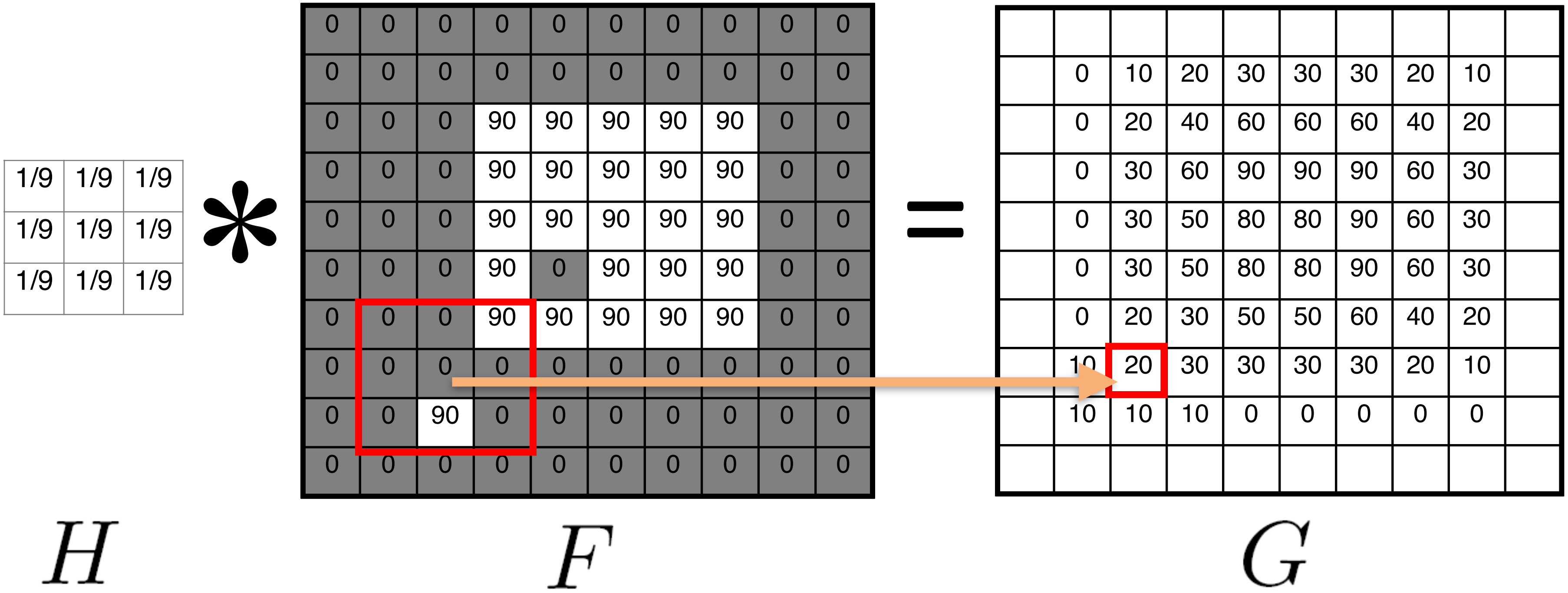
- In an ideal convolution operation, a kernel is "flipped" (horizontally and vertically) and then it is applied through the image (from left to right, and top to bottom)



kernel of size 3x3 or 5x5

# Convolution Operation

- What does a **convolution operation** do?



|  |  |  |
|---|---|---|
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |  |  |  |
|  | 0 | 10 | 20 | 30 | 30 | 30 | 20 | 10 |  |
|  | 0 | 20 | 40 | 60 | 60 | 60 | 40 | 20 |  |
|  | 0 | 30 | 60 | 90 | 90 | 90 | 60 | 30 |  |
|  | 0 | 30 | 50 | 80 | 80 | 90 | 60 | 30 |  |
|  | 0 | 30 | 50 | 80 | 80 | 90 | 60 | 30 |  |
|  | 0 | 20 | 30 | 50 | 50 | 60 | 40 | 20 |  |
|  | 10 | 20 | 30 | 30 | 30 | 30 | 20 | 10 |  |
|  | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 |  |
|  |  |  |  |  |  |  |  |  |  |

kernel of size 3x3 units

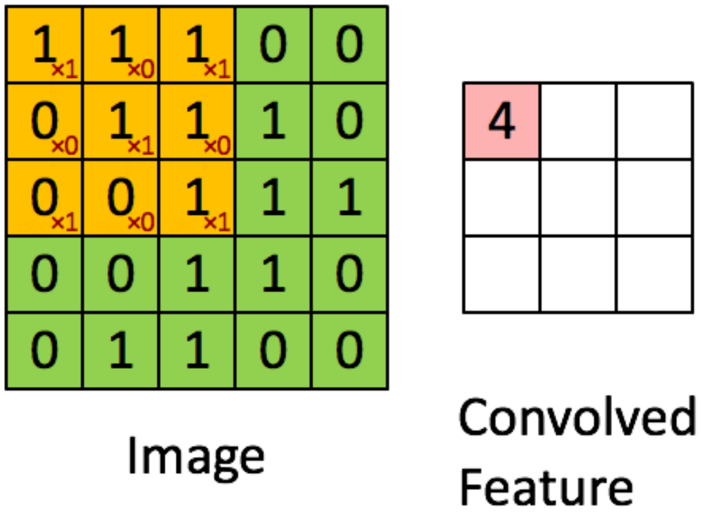
image of 10x10 units

convolved result of 10x10 units

# Convolution Operation

- What does a **convolution operation** do?

- **convolution operation can be achieved with a series of dot products between portions of input feature map and a convolution filter (kernel) weights**



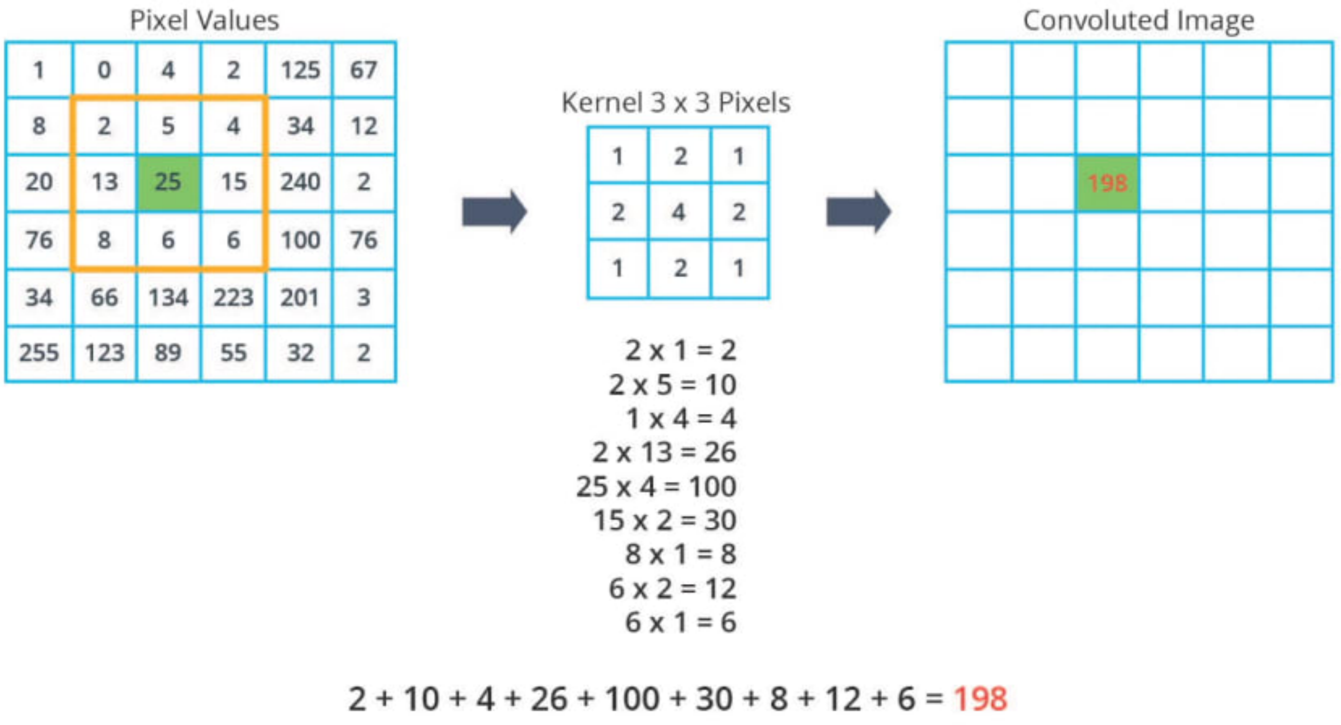
1*1 + 1*0 + 1*1 +
0*0 + 1*1 + 1*0 +
0*1 + 0*0 + 1*1 = 4

Another visualization shows a yellow convolution filter applied to a green image, resulting in the convolved feature

# Convolution Operation

- What does a convolution operation do?

- **convolution operation can be achieved with a series of dot products between portions of input feature map and a convolution filter (kernel) weights**



Pixel Values

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 4 | 2 | 125 | 67 |
| 8 | 2 | 5 | 4 | 34 | 12 |
| 20 | 13 | 25 | 15 | 240 | 2 |
| 76 | 8 | 6 | 6 | 100 | 76 |
| 34 | 66 | 134 | 223 | 201 | 3 |
| 255 | 123 | 89 | 55 | 32 | 2 |

Kernel 3 x 3 Pixels

| | | |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 4 | 2 |
| 1 | 2 | 1 |

2 x 1 = 2
2 x 5 = 10
1 x 4 = 4
2 x 13 = 26
25 x 4 = 100
15 x 2 = 30
8 x 1 = 8
6 x 2 = 12
6 x 1 = 6

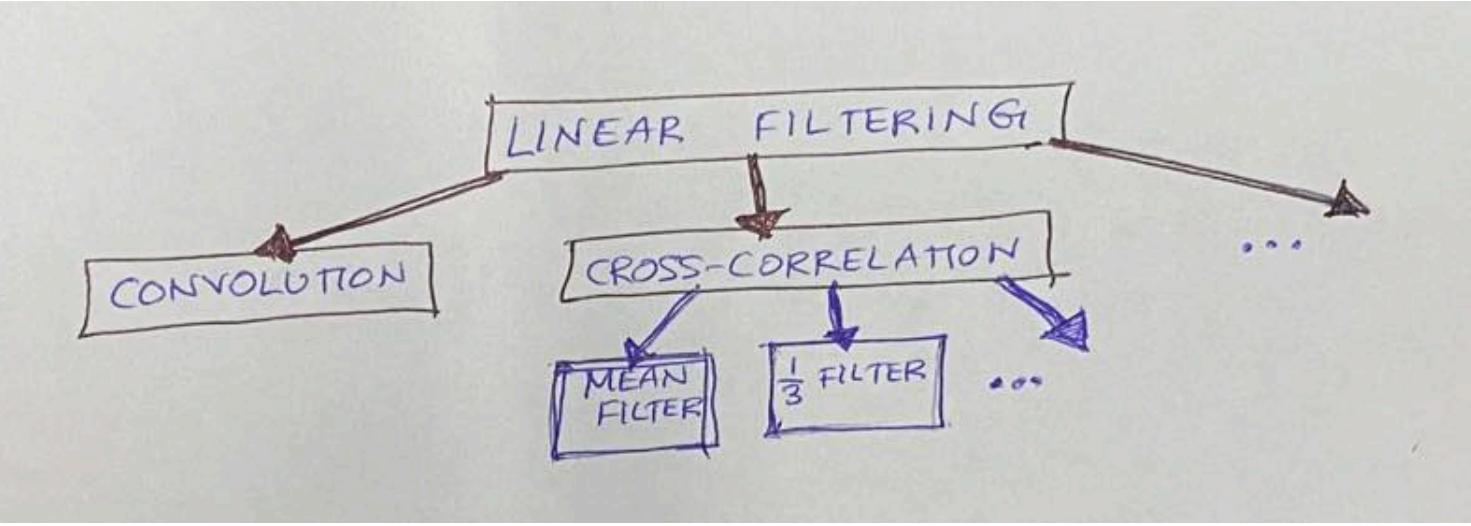
Convoluted Image

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | 198 | | | |
| | | | | | |
| | | | | | |
| | | | | | |

2 + 10 + 4 + 26 + 100 + 30 + 8 + 12 + 6 = 198

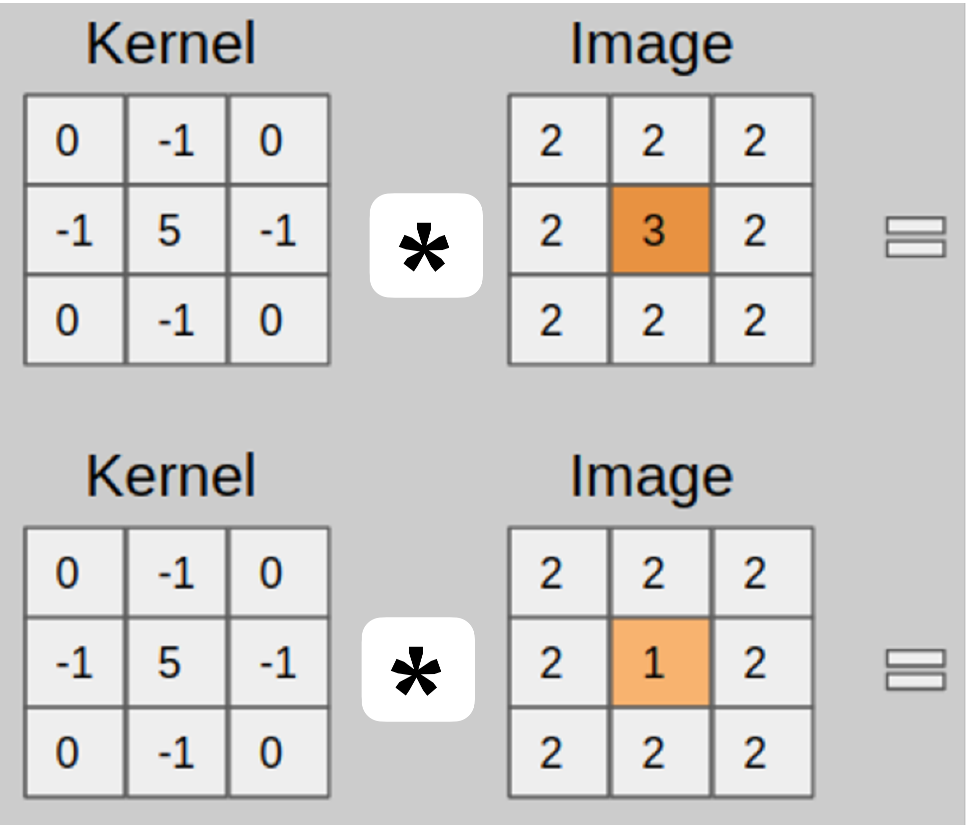
# Convolution Operation

- **Convolution operation** falls within a more general form operation call **linear-filtering**
  - replace each pixel by a linear-combination of its neighbors



# Group Exercise

- find the result of the **convolution operation** below



Kernel

| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 5 | -1 |
| 0 | -1 | 0 |

*

Image

| | | |
|---|---|---|
| 2 | 2 | 2 |
| 2 | 3 | 2 |
| 2 | 2 | 2 |

=

Kernel

| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 5 | -1 |
| 0 | -1 | 0 |

*

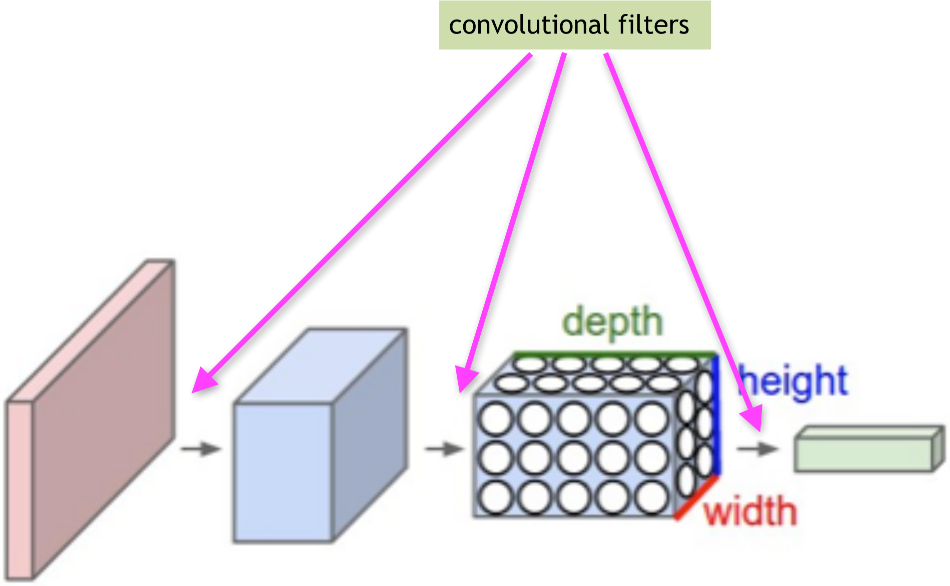
Image

| | | |
|---|---|---|
| 2 | 2 | 2 |
| 2 | 1 | 2 |
| 2 | 2 | 2 |

=

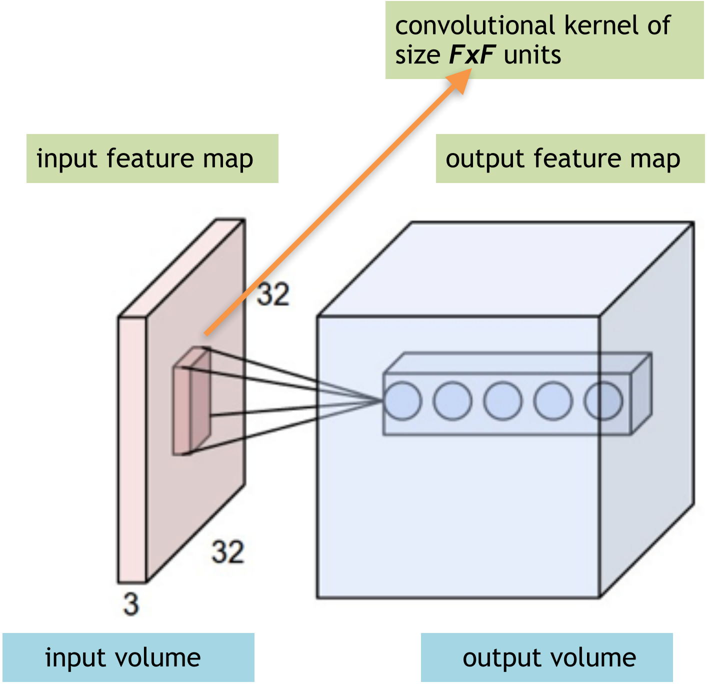
# Convolutional Neural Network (CNN)

- A **convolutional neural network (CNN)** is a neural network with specialized connectivity structure



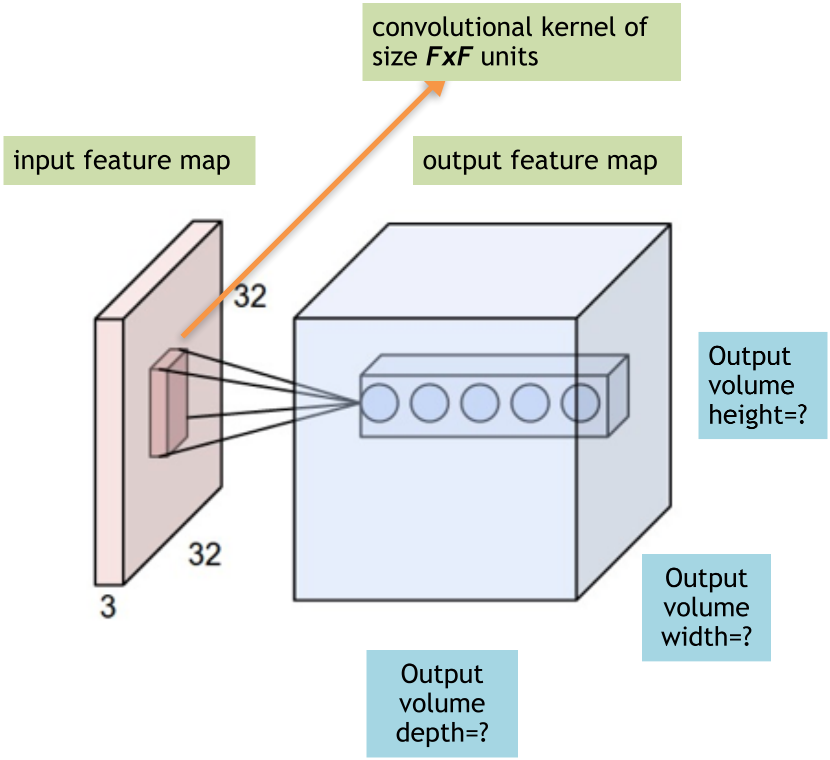
- Every layer of a CNN transforms the input volume to an output volume of neuron activations. The red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels)

# Convolutional Neural Network (CNN)



- Weights correspond to the filter (kernel) values
- **Convolutional neural network** can **learn** their own filters!
    - We do not need to provide the values inside the kernel

# CNN: How to calculate the output volume size?



convolutional kernel of size ***FxF*** units

input feature map

output feature map

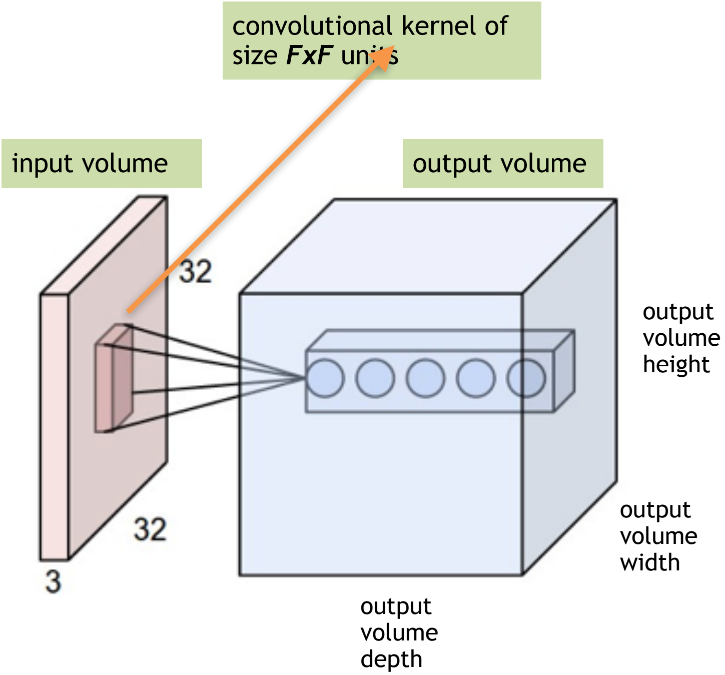
Output volume height=?

Output volume width=?

Output volume depth=?

# CNN: How to calculate the output volume size?

- An input volume has size ***(WxWx3)***, eg, (227, 227, 3)
- Filter size/receptive field is ***(FxF)***, eg, (11x11)
- Spatial Stride ***S***, eg, ***S***=4
- Padding size ***P***, eg, ***P***=0
- Number of filters ***K***, eg, ***K***=96



convolutional kernel of size ***FxF*** units

input volume

output volume

output volume height

output volume width

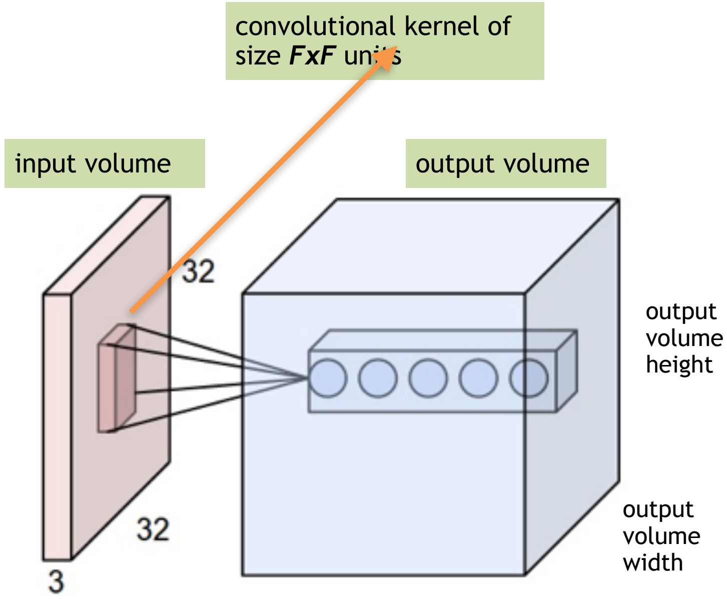
output volume depth

- Size of the <u>output volume width</u> and <u>output volume height</u> as a function of ***W, F, S,***and ***P*** as follows:

output volume width/ height $= \boxed{\frac{(W - F + 2P)}{S} + 1} = \frac{(227 - 11 + 2*0)}{4} + 1 = 54+1 = 55$

# How to calculate the output volume size?

- An input volume has size $(\mathbf{W_1 \times H_1 \times D_1})$
  - Filter size/receptive field is *(FxF)*
  - Spatial stride size ***S***
  - Padding size ***P***
  - Number of filters ***K***



- Spatial sizes of the output volume $(W_2 \times H_2 \times D_2)$

$$W_2 = \frac{(W_1 - F + 2P)}{S} + 1$$

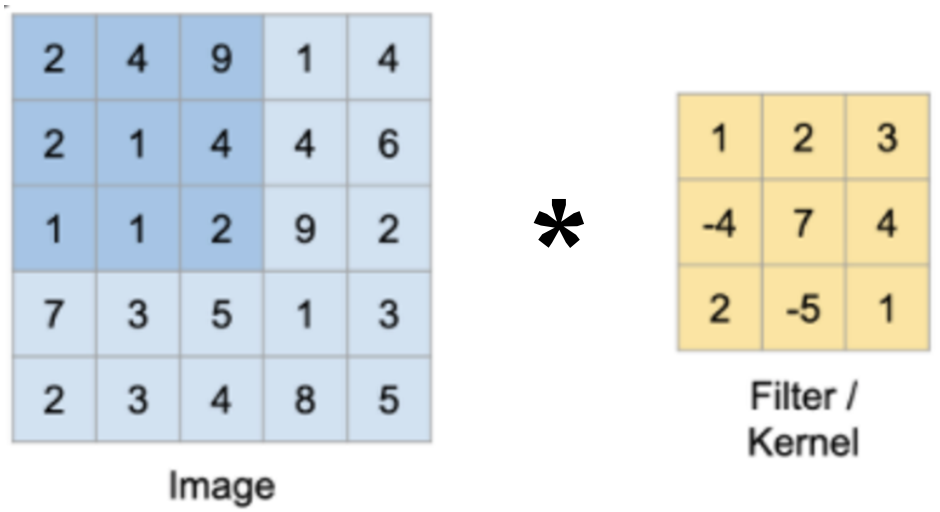
$$H_2 = \frac{(H_1 - F + 2P)}{S} + 1$$

$$D_2 = K$$

- Number of filter weight parameters $= (F \times F \times D_1) \times K$
- Number of bias parameters = ***K***

# Group Exercise

- What will the size of the output of the following convolution be?
  - (5x5x1) * (3x3)



| | | | | |
|---|---|---|---|---|
| 2 | 4 | 9 | 1 | 4 |
| 2 | 1 | 4 | 4 | 6 |
| 1 | 1 | 2 | 9 | 2 |
| 7 | 3 | 5 | 1 | 3 |
| 2 | 3 | 4 | 8 | 5 |

Image

*

| | | |
|---|---|---|
| 1 | 2 | 3 |
| -4 | 7 | 4 |
| 2 | -5 | 1 |

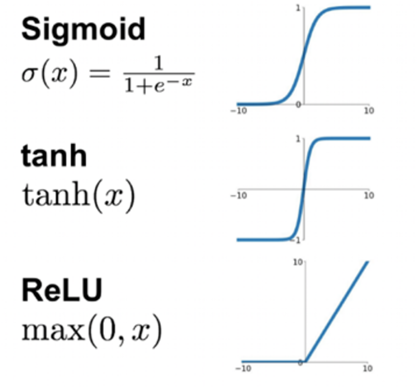
Filter / Kernel

# Today's Agenda

- Deep Learning

- Convolutional Neural Network (CNN)
  - Convolution operation
  - Nonlinearity
  - Pooling operation
  - CNN: convolutional layer + nonlinearity + pooling layer

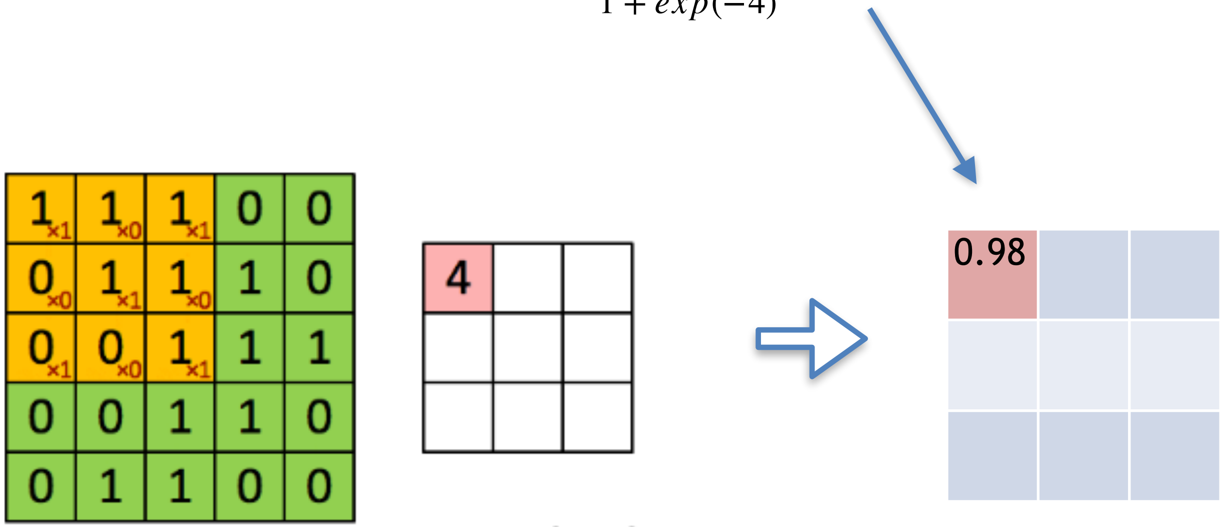
# Nonlinear Function



- Just like an MLP, each convolutional output goes through a non-linear function such as Sigmoid, Tanh, or Rectified Linear Unit (ReLU)

$$convolution = 1*1+1*0+1*1+0*0+1*1+1*0+0*1+0*0+1*1 = 4$$

$$Sigmoid(4) = \frac{1}{1+exp(-4)} = 0.98$$



| | | | | |
|---|---|---|---|---|
| 1 ×1 | 1 ×0 | 1 ×1 | 0 | 0 |
| 0 ×0 | 1 ×1 | 1 ×0 | 1 | 0 |
| 0 ×1 | 0 ×0 | 1 ×1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Image

| | | |
|---|---|---|
| 4 | | |
| | | |
| | | |

Convolved Feature

Sigmoid

| | | |
|---|---|---|
| 0.98 | | |
| | | |
| | | |

# Today’s Agenda

- Deep Learning

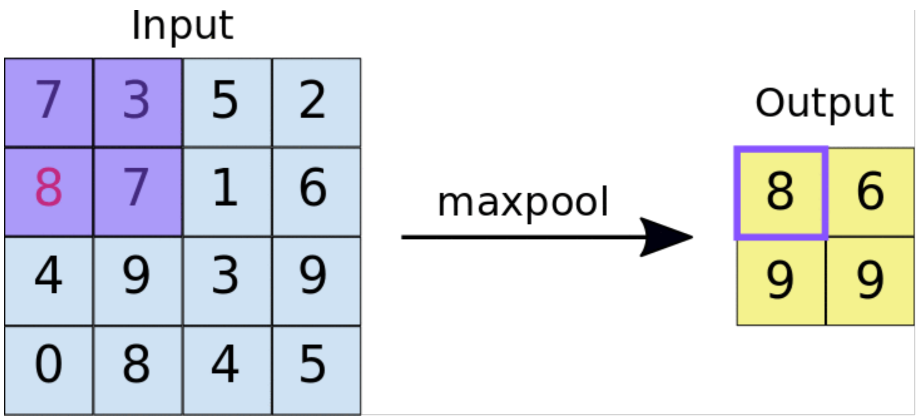
- Convolutional Neural Network (CNN)
  - Convolution operation
  - Nonlinearity
  - Pooling operation
  - CNN: convolutional layer + nonlinearity + pooling layer

# Pooling Operation

- Image data can get computationally inefficient, really quickly. To avoid this, we often toss in a layer that helps us to **summarize** and **downsample** the data

- In classical CNN, we find another useful operation called **pooling operation**

- A common pooling operation is **max pooling**, and its goal is to locally summarize the convolution. It performs something like a convolution, but rather than taking the dot product, it takes the maximum element in the filter area



Input

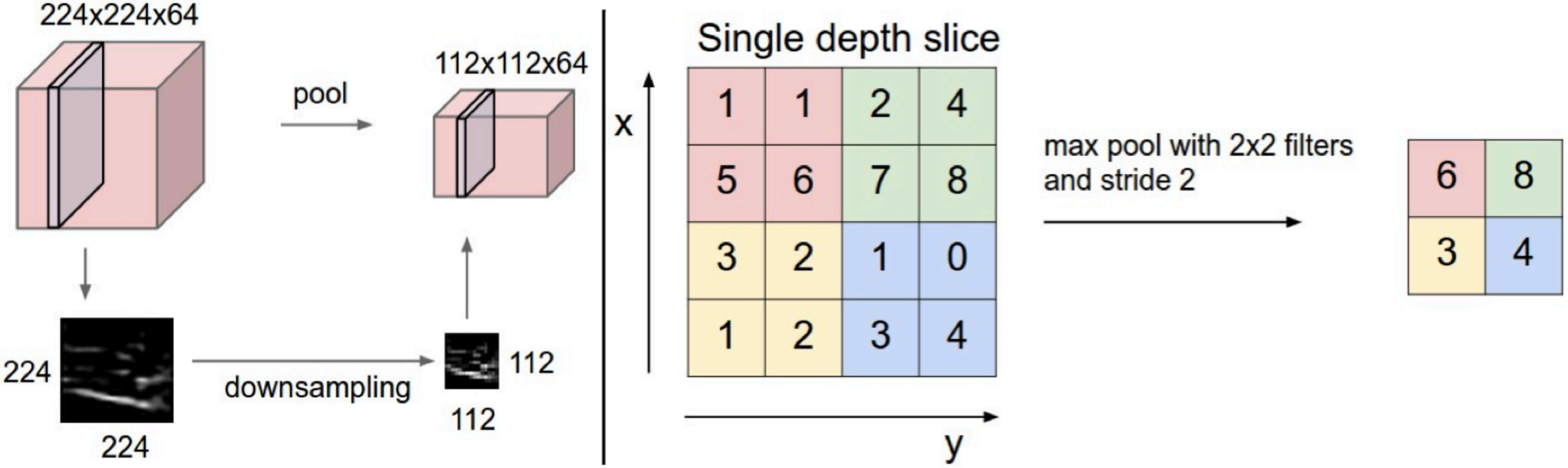
| | | | |
|---|---|---|---|
| 7 | 3 | 5 | 2 |
| 8 | 7 | 1 | 6 |
| 4 | 9 | 3 | 9 |
| 0 | 8 | 4 | 5 |

Output

| | |
|---|---|
| 8 | 6 |
| 9 | 9 |

# Pooling Operation

- Pooling operation downsamples the volume spatially, **independently in each depth slice of the input volume**
- Besides max pooling, other pooling operations include: **sum pooling**, average pooling



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

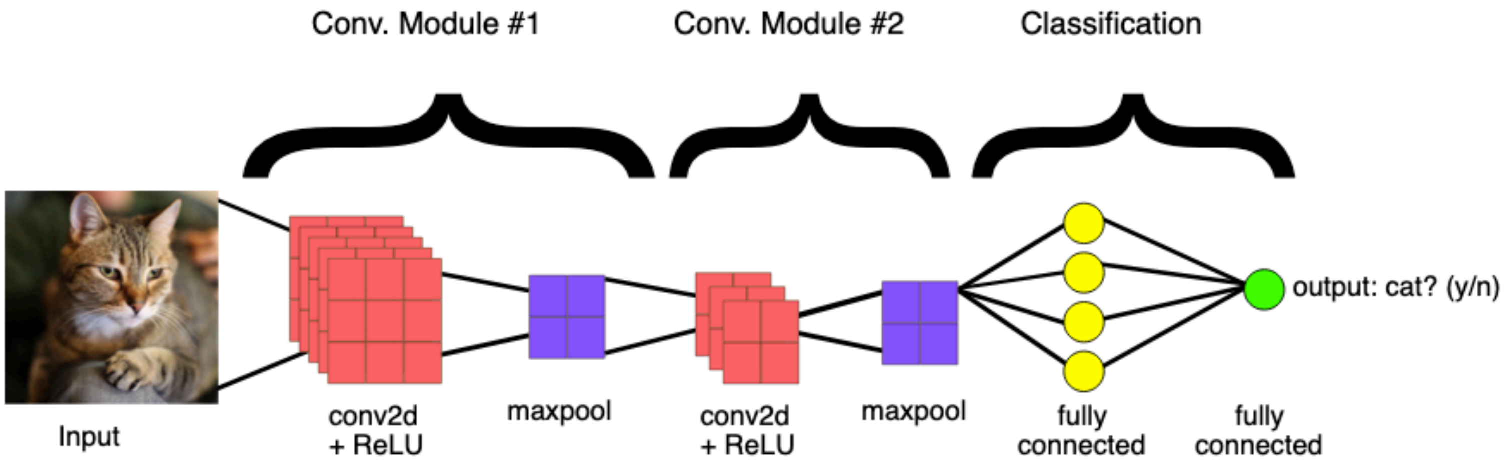
| | |
|---|---|
| 6 | 8 |
| 3 | 4 |

# Today’s Agenda

- Deep Learning
- Convolutional Neural Network (CNN)
    - Convolution operation
    - Nonlinearity
    - Pooling operation
    - CNN: convolutional layer + nonlinearity + pooling layer

# CNN: A Composition of Convolutional Layers

- We've talked about **image data**, **convolutions**, **nonlinearity**, **max pooling**, and how they are related to some computer vision tasks. Let's connect the dots
  - input is an image (in this case a color image, so 3 channels—red, green, and blue)
  - there are several filters, not just one.
  - `Conv2D` layers with `ReLU` are often followed by `maxpool`
  - towards the end of the model, we switch to fully connected (`Dense`) layer
  - We have as many output nodes as we have classes to predict



Reference