Welcome to Computer Vision

Computer Vision

Dr. Muhammad Tahir

DEPARTMENT OF COMPUTER SCIENCE,

FAST-NUCES, Peshawar

Course Details

LECTURES: Monday
& Wednesday

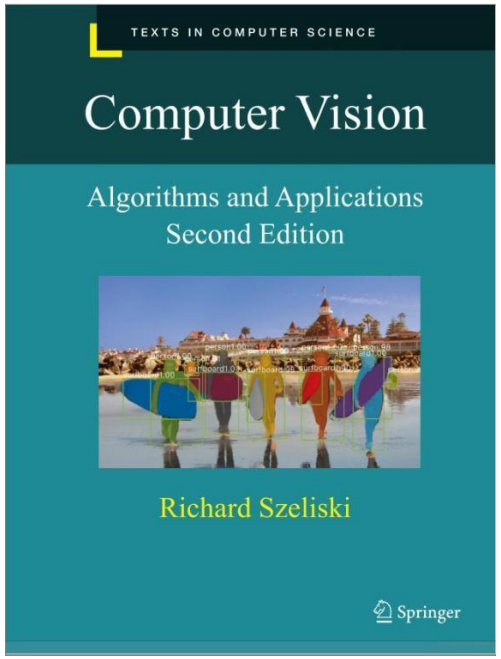
TIMINGS:

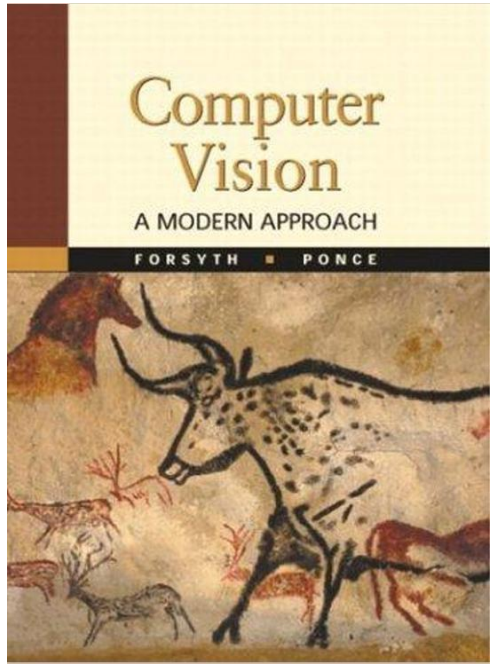
9:30 am – 11:00 am

MY OFFICE:

OFFICE HOURS:

EMAIL: m.tahir@nu.edu.pk





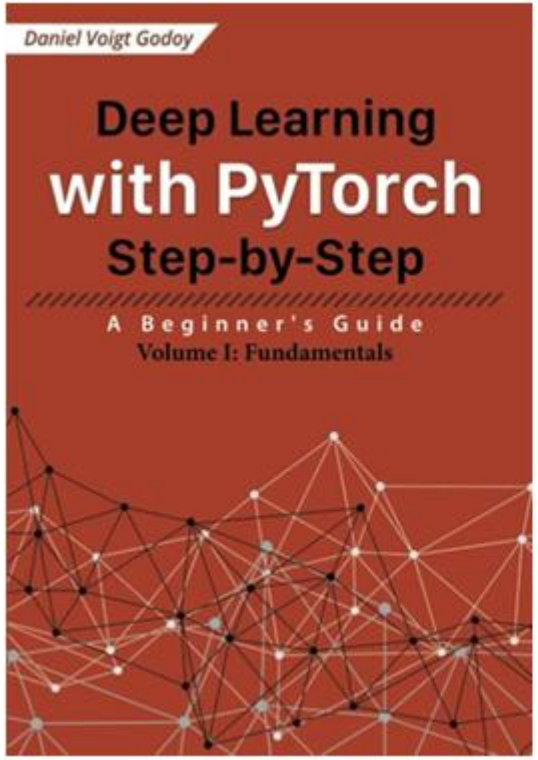
References

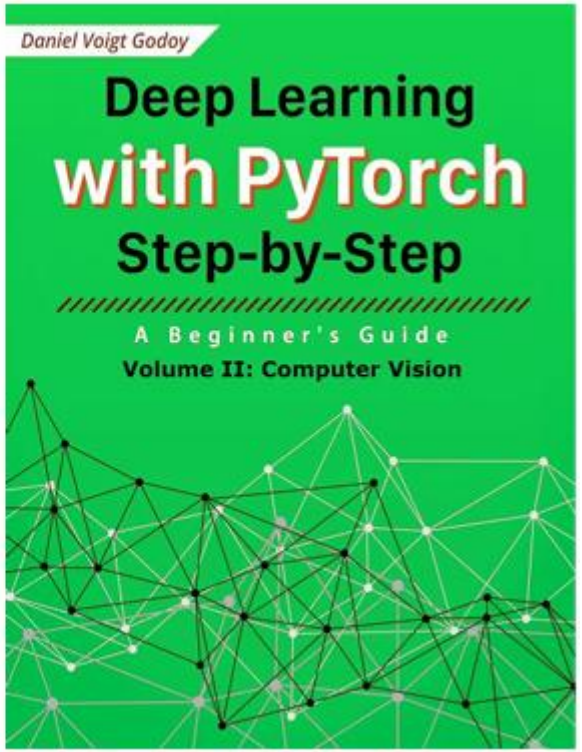
The material in these slides are based on:

1. Rick Szeliski's book: Computer Vision: Algorithms and Applications
2. Forsythe and Ponce: Computer Vision: A Modern Approach

Recommended Books

Deep Learning with PyTorch Step-by-Step by Daniel Voigt Godoy





Course Learning Outcomes

No	CLO (Tentative)	Domain	Taxonomy Level	PLO
1	Understanding basics of Computer Vision: algorithms, tools, and techniques	Cognitive	2	
2	Develop solutions for image/video understanding and recognition	Cognitive	3	
3	Design solutions to solve practical Computer Vision problems	Cognitive	3	

Outline

Deep Learning Fundamentals

[Deep Learning Tutorial - GeeksforGeeks]

Limitations of Hand-Crafted Features

- **Limited Representational Power:** Hand-crafted features are designed based on human intuition and domain knowledge, which may not fully capture the complexity and variability of real-world data (all possible cases).
- **Brittleness:** Hand-crafted features are often sensitive to variations in lighting, viewpoint, and other factors, making them less robust to real-world conditions.
- **Manual Engineering:** Designing hand-crafted features requires expertise and time-consuming experimentation, especially for complex tasks or datasets.
- **Limited Transferability:** Hand-crafted features may not generalize well to new tasks or datasets, requiring re engineering or adaptation.

Deep Learning/AI Advantages

- **Learned Representations:** Deep learning models can automatically learn features from data, potentially capturing complex patterns and relationships that are difficult to hand-craft.
- **End-to-End Learning:** Deep learning models can learn hierarchical representations directly from raw data to perform a task, eliminating the need for manual feature engineering.
- **Flexibility and Adaptability:** Deep learning models can be adapted to various tasks and datasets with minimal changes, making them more versatile.
- **Scalability:** Deep learning models can scale with the size of data and compute resources, potentially achieving better performance with more data.
- **State-of-the-Art Performance:** Deep learning has achieved state-of-the-art performance in many computer vision tasks, such as image classification, object detection, and segmentation

The 2012 revolution

ImageNet Challenge

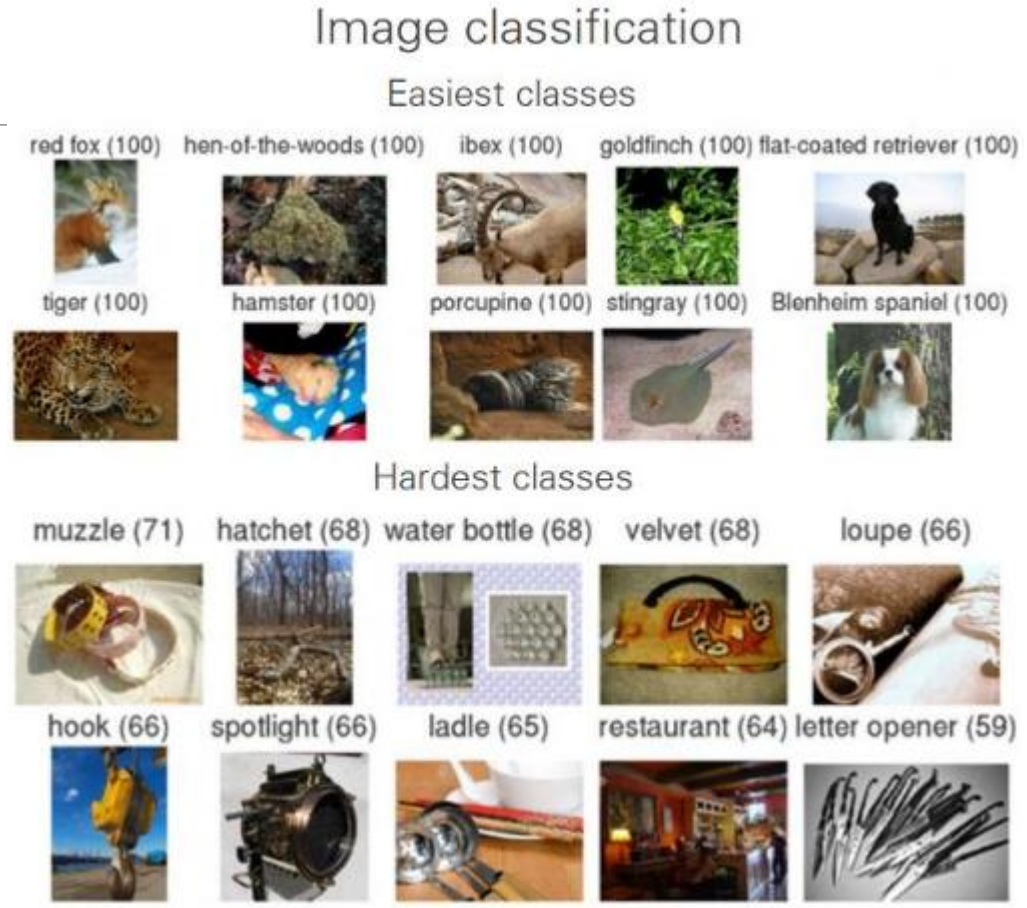
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
  - **1.2M** training images with **1K** categories
  - Measure top-5 classification error

Output
Scale
T-shirt
Steel drum
Drumstick
Mud turtle

✓

Output
Scale
T-shirt
Giant panda
Drumstick
Mud turtle

✗



IMAGENET Large Scale Visual Recognition Challenge

The Image Classification Challenge:

1,000 object classes

1,431,167 images

Output:
Scale
T-shirt
Steel drum
Drumstick
Mud turtle

Output:
Scale
T-shirt
Giant panda
Drumstick
Mud turtle

Russakovsky et al. IJCV 2015

Structured Data

Size	#bedrooms	⋯	Price (1000$s)
1200	2		300
1500	3		400
2000	3		480
⋮	⋮		⋮
3000	4		520

User Region	Ad Id	⋯	Ad revenue($)
USA	1005		0.5
UK	1009		0.3
USA	998		0.5
⋮	⋮		⋮
CAN	2104		0.45

Unstructured Data

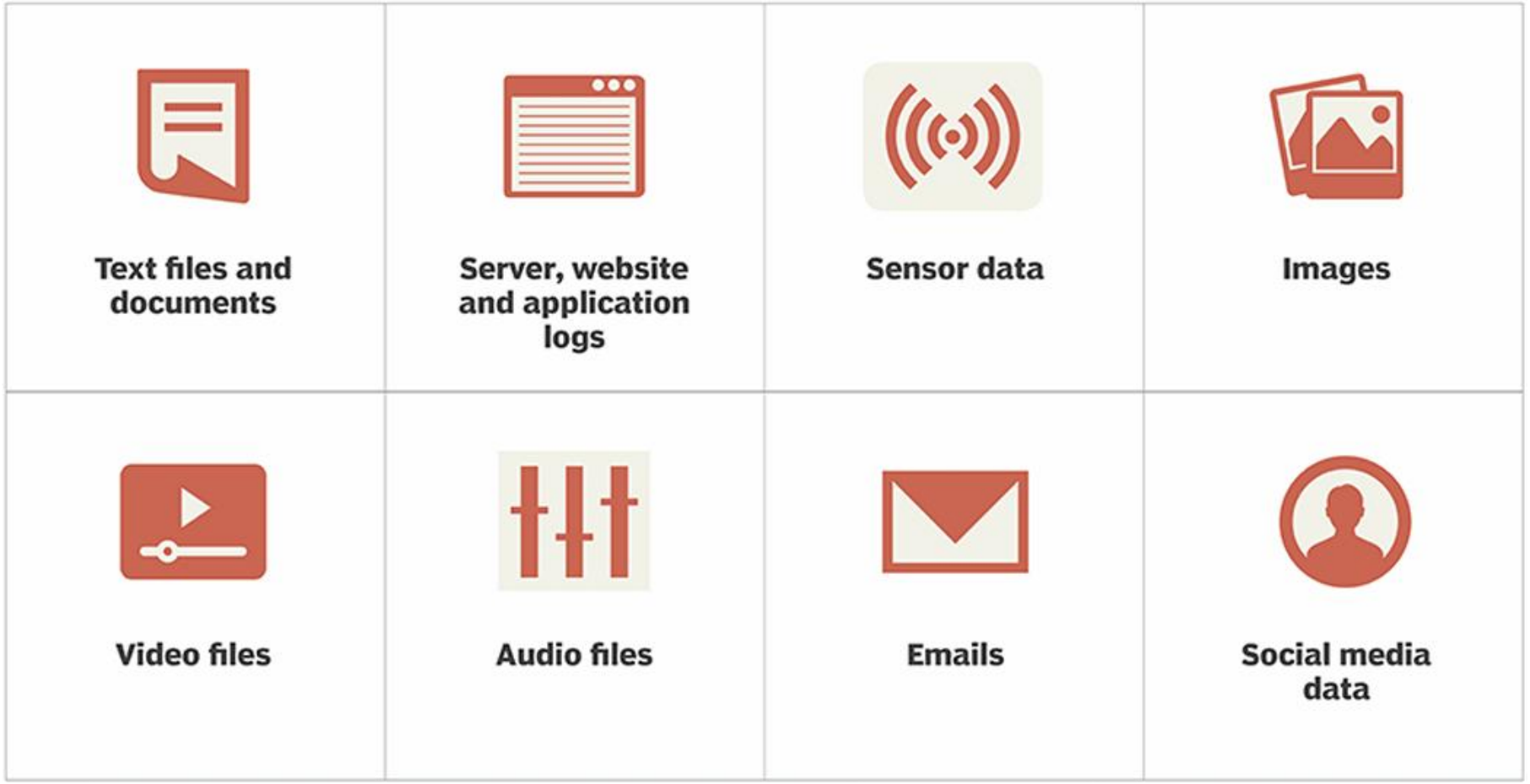
Audio

Image

Once upon a time in the history of...

Text

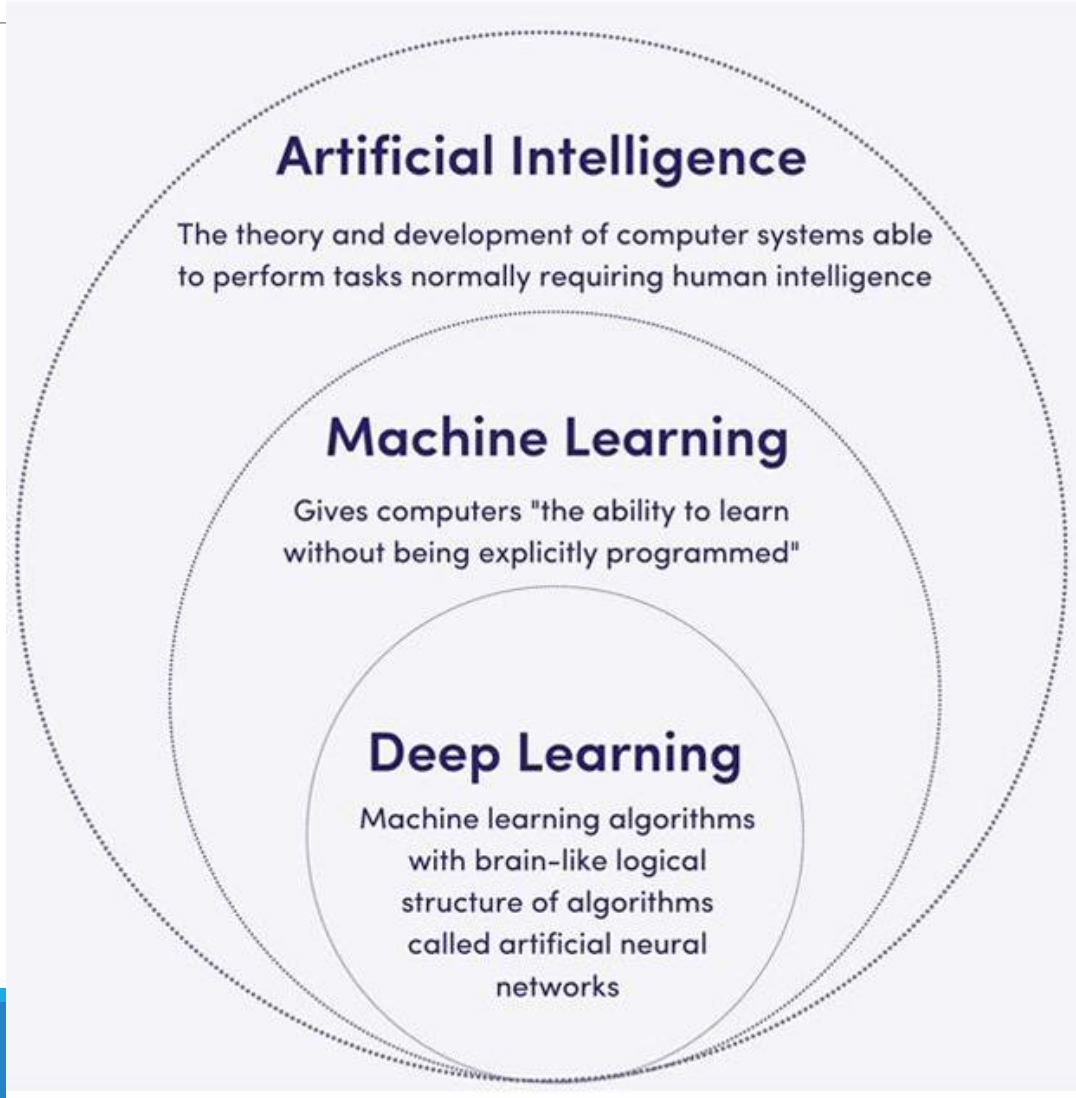
Unstructured Data



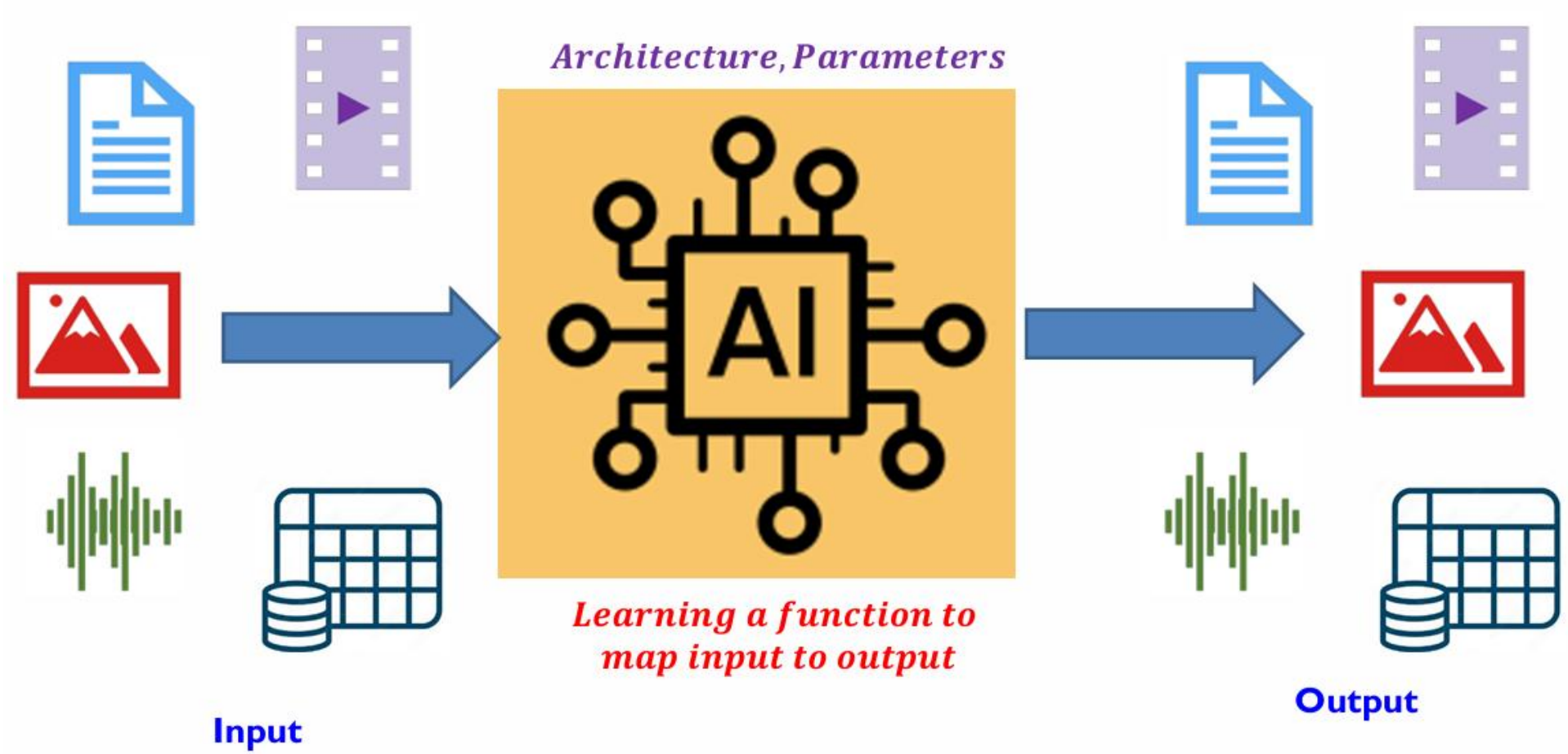
Deep Neural Networks are very good at processing these Unstructured data

Artificial Intelligence (AI)

The term Deep Learning refers to training very large Neural Network

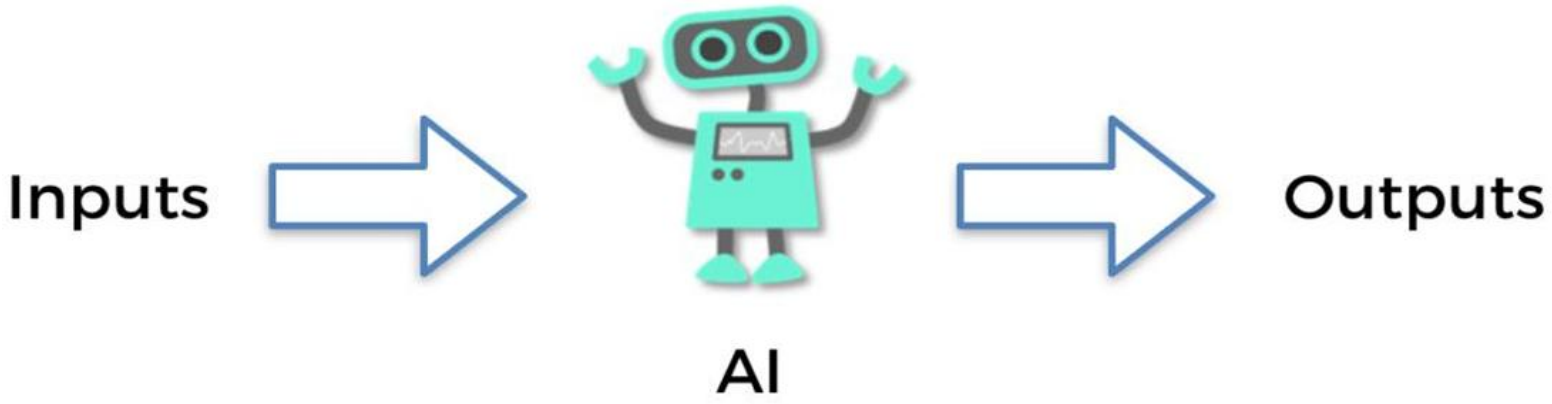


Why AI?

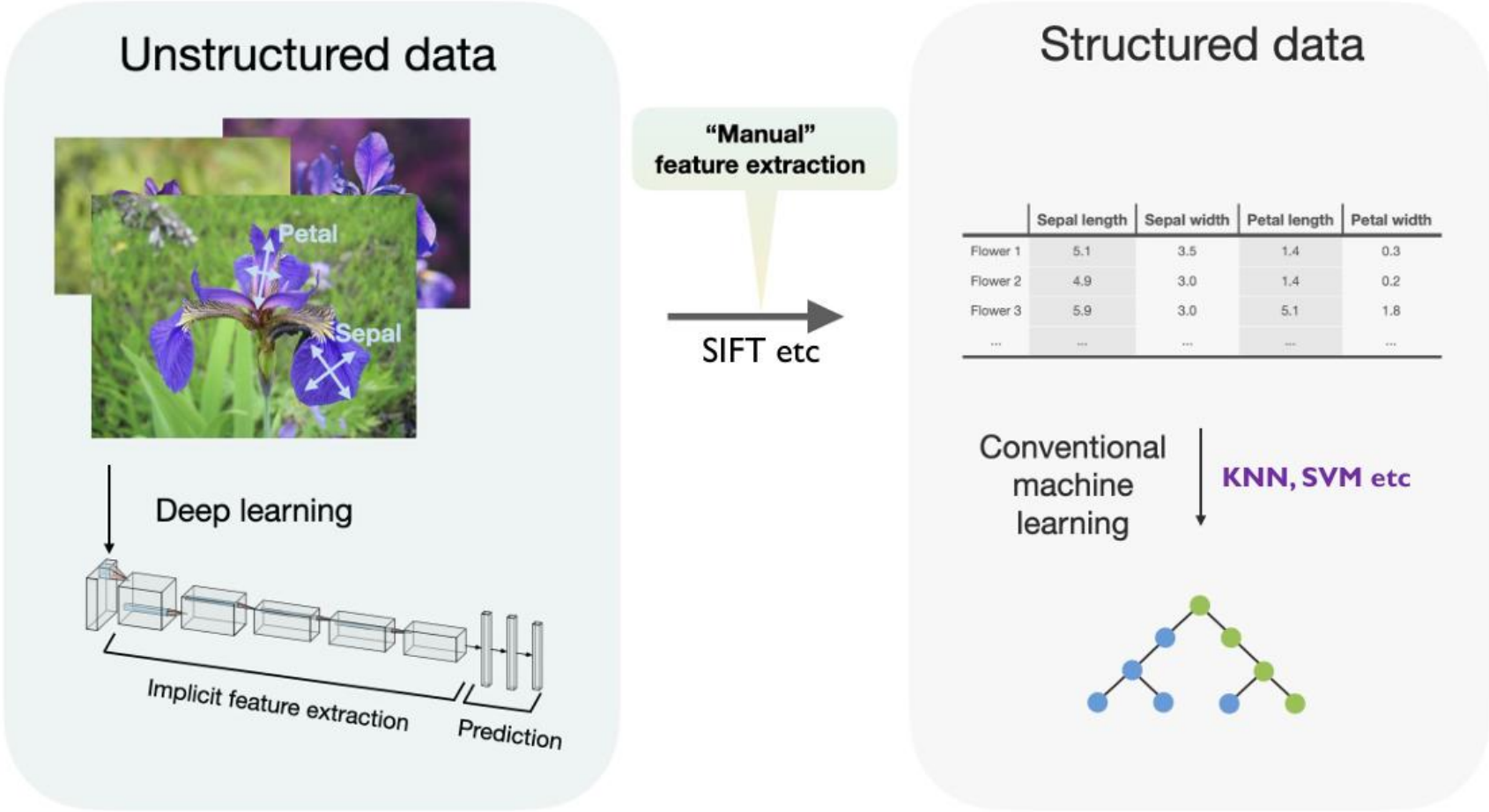

Architecture, Parameters
AI
Learning a function to map input to output
Input
Output

AI Model examples

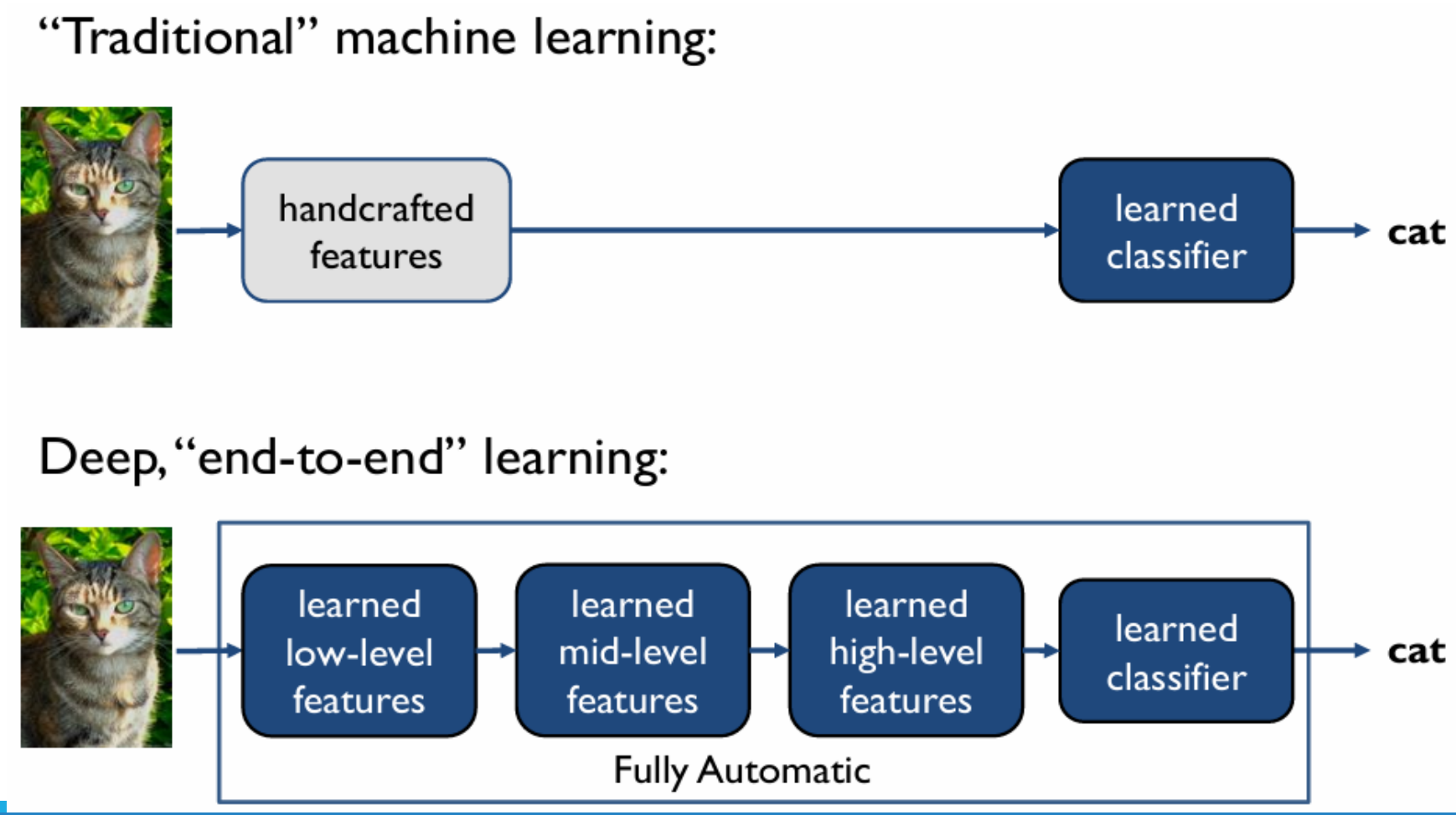
- Text to Text?
- Image to Text?
- Speech to Text?
- Signal to Text?
- Image to Class?
- Signal to Number?



ML vs DL


Unstructured data
Petal
Sepal
Deep learning
Implicit feature extraction
Prediction
"Manual"
feature extraction
SIFT etc
Structured data
Sepal length
Sepal width
Petal length
Petal width
Flower 1
5.1
3.5
1.4
0.3
Flower 2
4.9
3.0
1.4
0.2
Flower 3
5.9
3.0
5.1
1.8
Conventional
machine
learning
KNN, SVM etc

ML vs DL

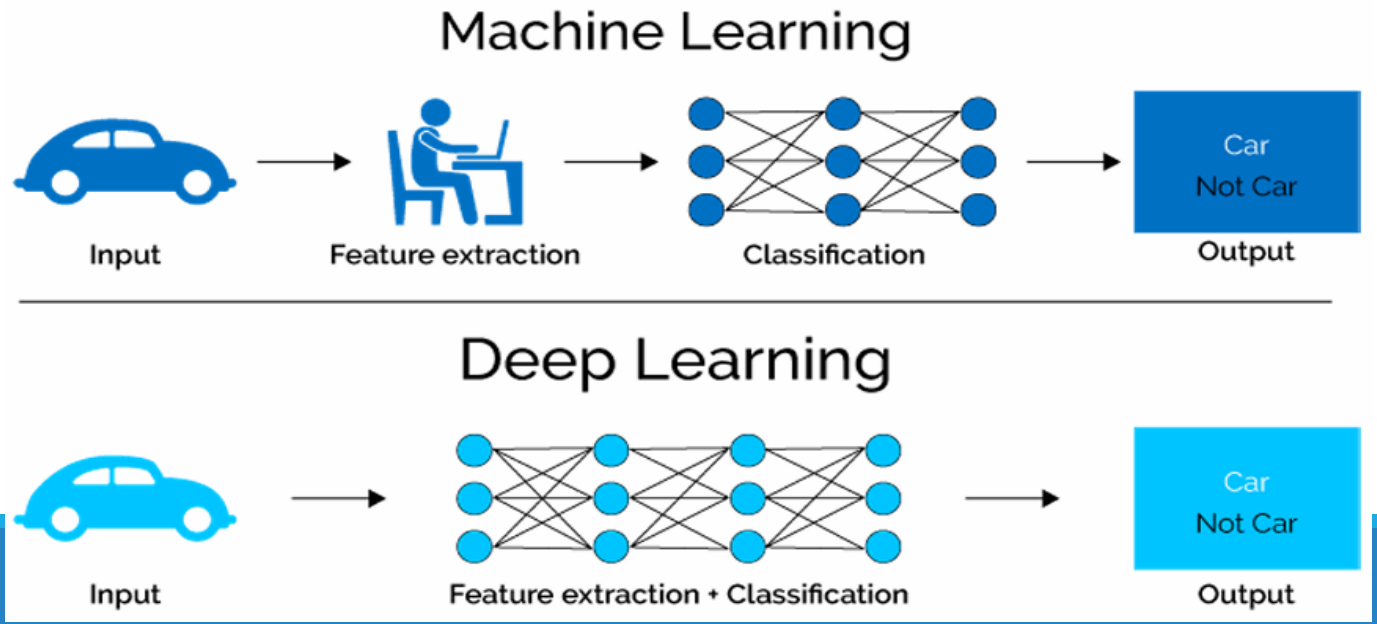

"Traditional" machine learning:
handcrafted features
learned classifier
cat
Deep, "end-to-end" learning:
learned low-level features
learned mid-level features
learned high-level features
learned classifier
cat
Fully Automatic

What is Deep Learning (DL) ?

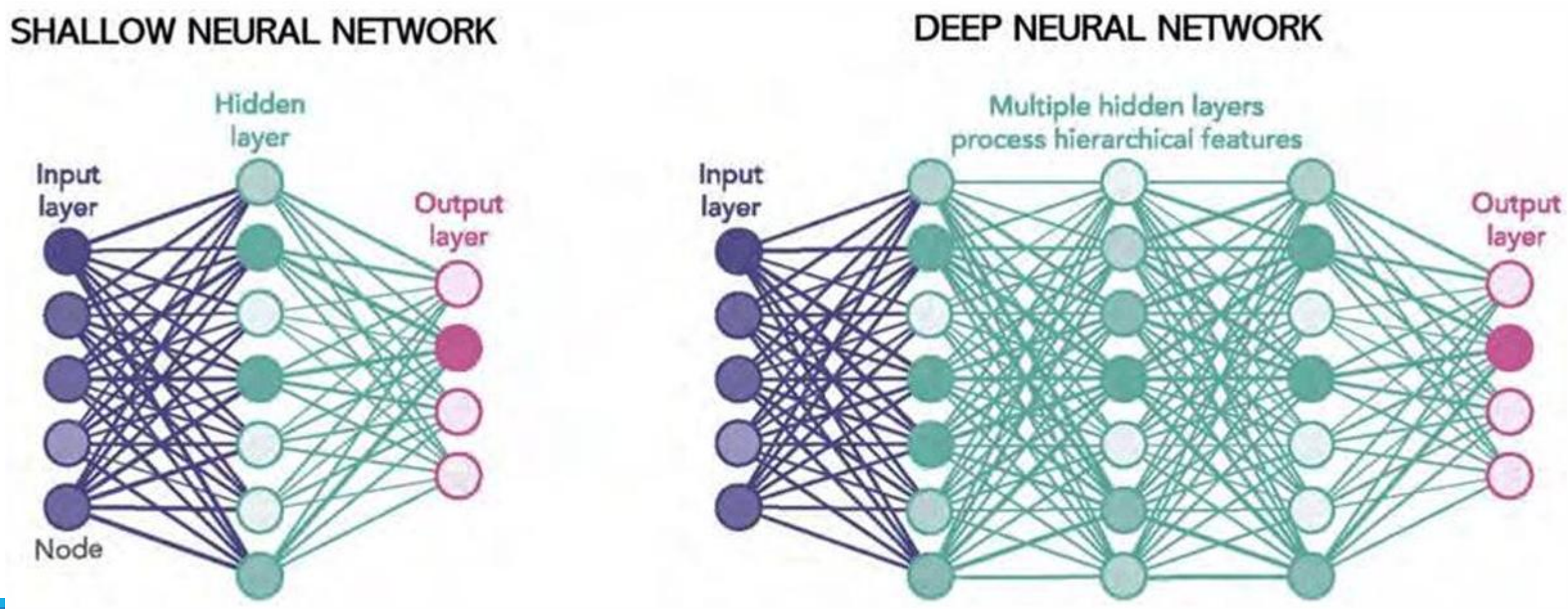
A machine learning subfield of learning representations of data. Exceptional effective at learning patterns.

Deep learning algorithms attempt to learn (multiple levels of) representation by using a hierarchy of multiple layers

If you provide the system tons of information, it begins to understand it and respond in useful ways.



Shallow Vs Deep Neural Networks

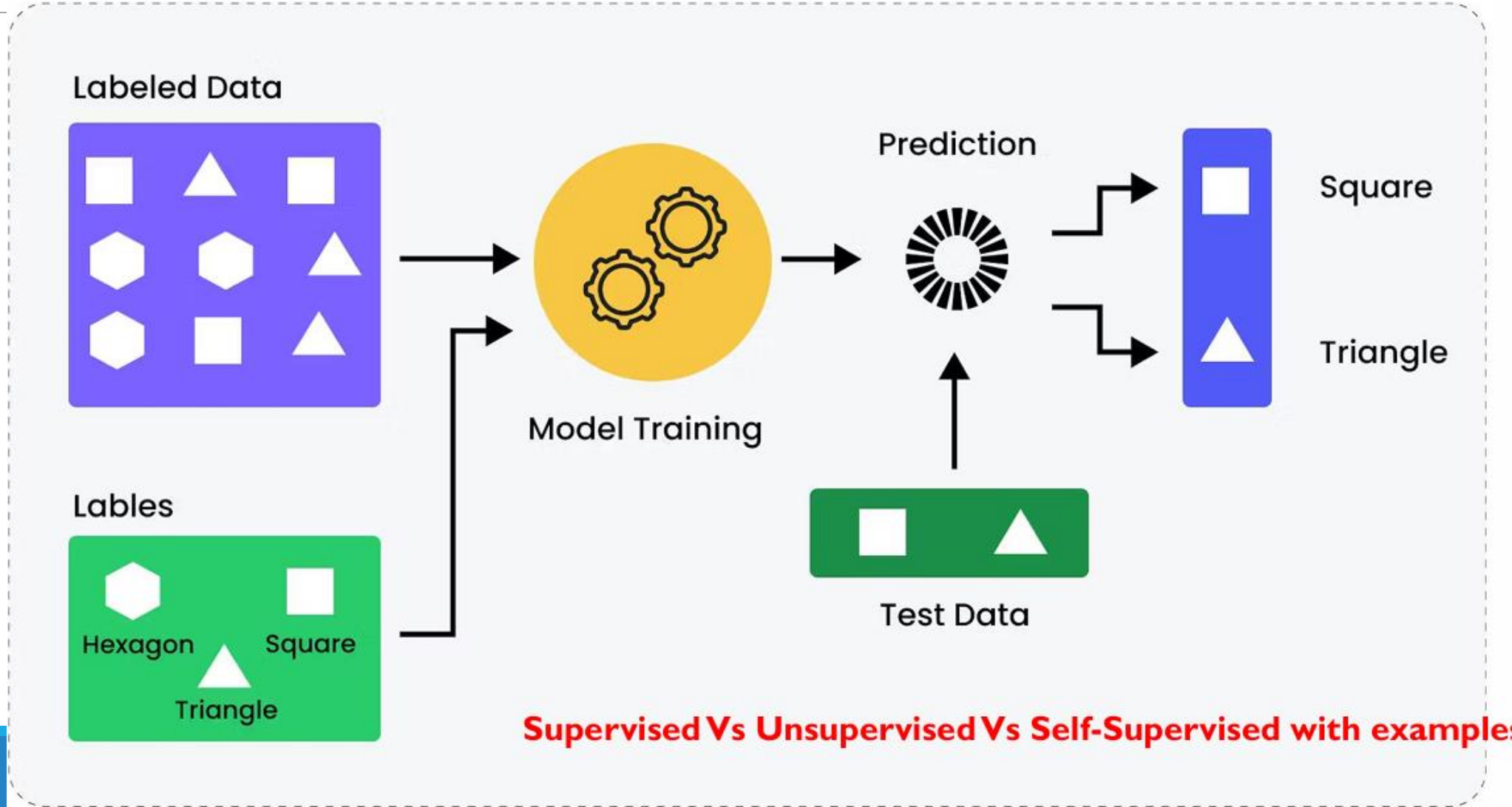


Shallow Vs Deep Neural Networks

Factors	Shallow Neural Network (SNN)	Deep Neural Network (DNN)
Feature Engineering	1. Individual feature extraction process is required. Various features cited in the literature are histogram oriented gradients, speeded up robust features, and local binary patterns.	1. Replace the hand-crafted features and directly work on the entire input. Thus, more practical for complex datasets.
Data Size Dependency	2. Needs a lesser quantity of data.	2. Needs vast volumes of data.

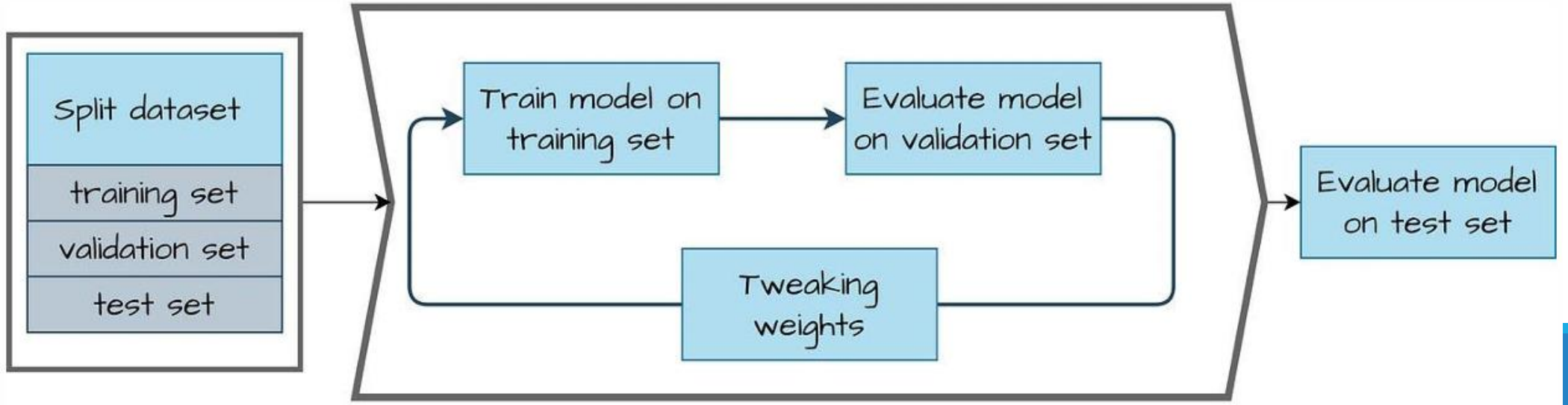
Fundamentals of Deep Learning

AI Pipeline

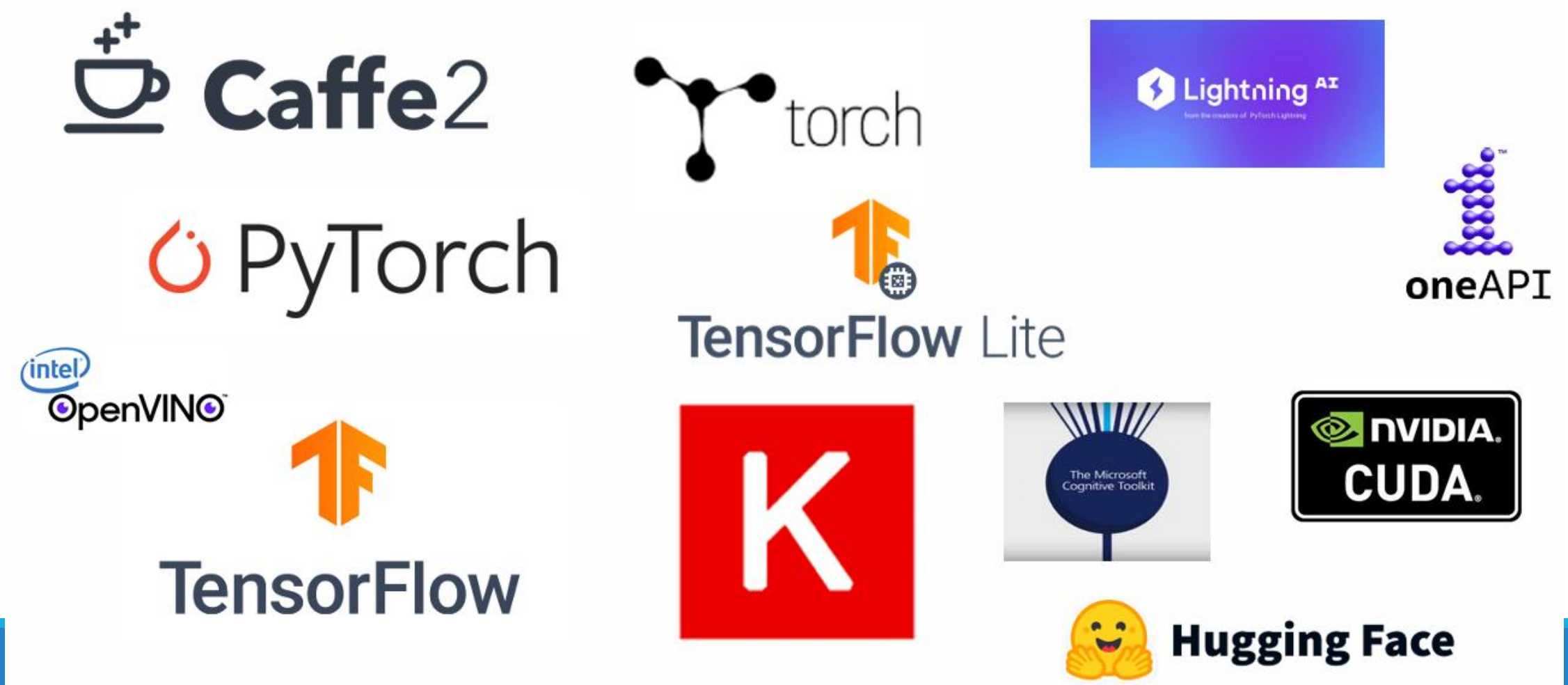

Labeled Data
Lables
Hexagon
Square
Triangle
Model Training
Prediction
Square
Triangle
Test Data
Supervised Vs Unsupervised Vs Self-Supervised with examples?

AI Pipeline

- Huge amount of training data & very large number of parameters
- Train/validate/test data management, Layers of each architecture, loss functions, training algorithms (optimizers), metrics, data augmentation, labeling tools
- Pretrained Models, Transfer Learning
- Requires GPU (CUDA, NVIDIA)
- Demands a completely new set of tools/programming-language/framework



Deep Learning Frameworks



Deep Learning

- **Deep Learning** involves training artificial neural networks to learn from vast amounts of data, simulating the way the human brain processes information.
- The term “deep” refers to the number of layers in the neural network, where each layer learns increasingly abstract representations of the input data.
- Deep Learning is a **machine learning technique** that automates feature extraction and uses multi-layered neural networks to model complex relationships in data.
- Unlike traditional machine learning models, which require manual feature extraction and selection, deep learning can automatically detect patterns and make decisions with minimal human intervention.

Neural Networks: The Building Blocks of Deep Learning

- A **neural network** is a series of algorithms designed to recognize underlying relationships in a set of data through a process that mimics how the human brain operates.
- Neural networks are composed of **neurons** organized in layers, each responsible for different aspects of the learning process.
  - **Input Layer**: The first layer of the neural network where the input data (features) is fed into the model.
  - **Hidden Layers**: Layers between the input and output, where computations occur. These layers extract features and learn representations from the data.
  - **Output Layer**: The final layer, where predictions or classifications are made based on the learned features.

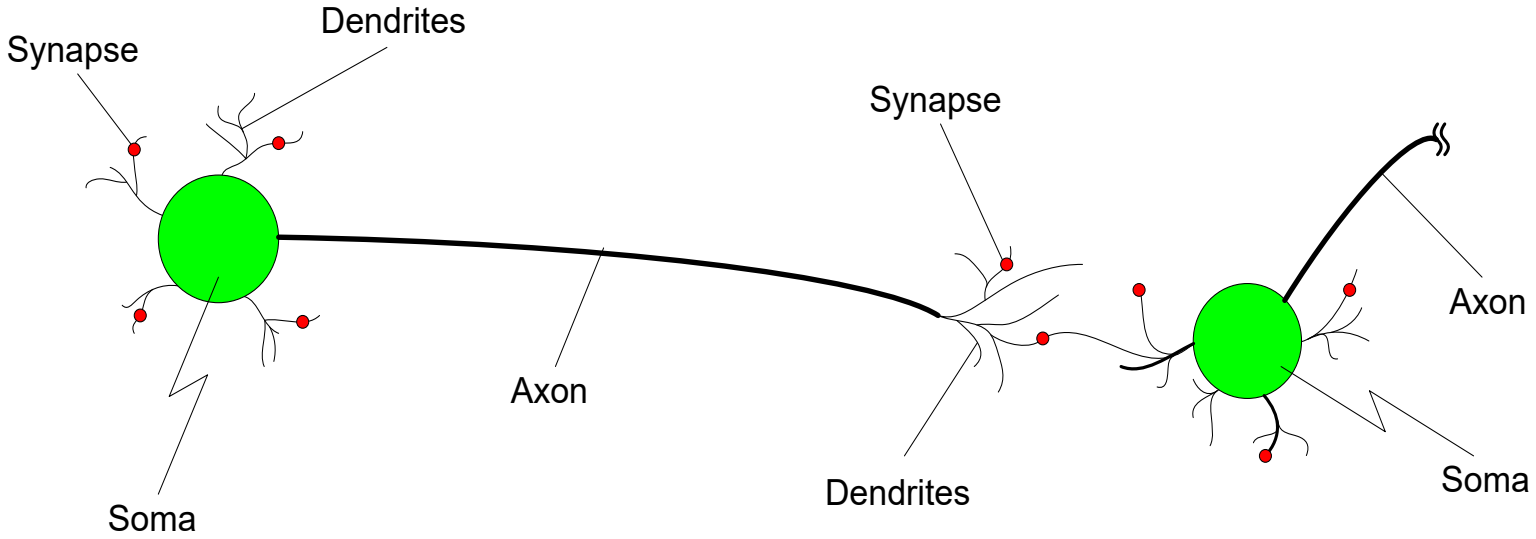
Key Elements of a Neural Network

- **Neurons**: The basic unit of a neural network, representing a single computation.
- **Weights**: Connections between neurons that determine the strength of the signal transmitted.
- **Bias**: An additional parameter that helps adjust the output.
- **Activation Functions**: Non-linear functions that introduce complexity and enable the network to learn from data. Examples include **ReLU (Rectified Linear Unit)**, **Sigmoid**, and **Tanh**.
- Each neuron in a layer receives an input, processes it using a mathematical function (activation function), and passes its output to the next layer.

Biological Neuron

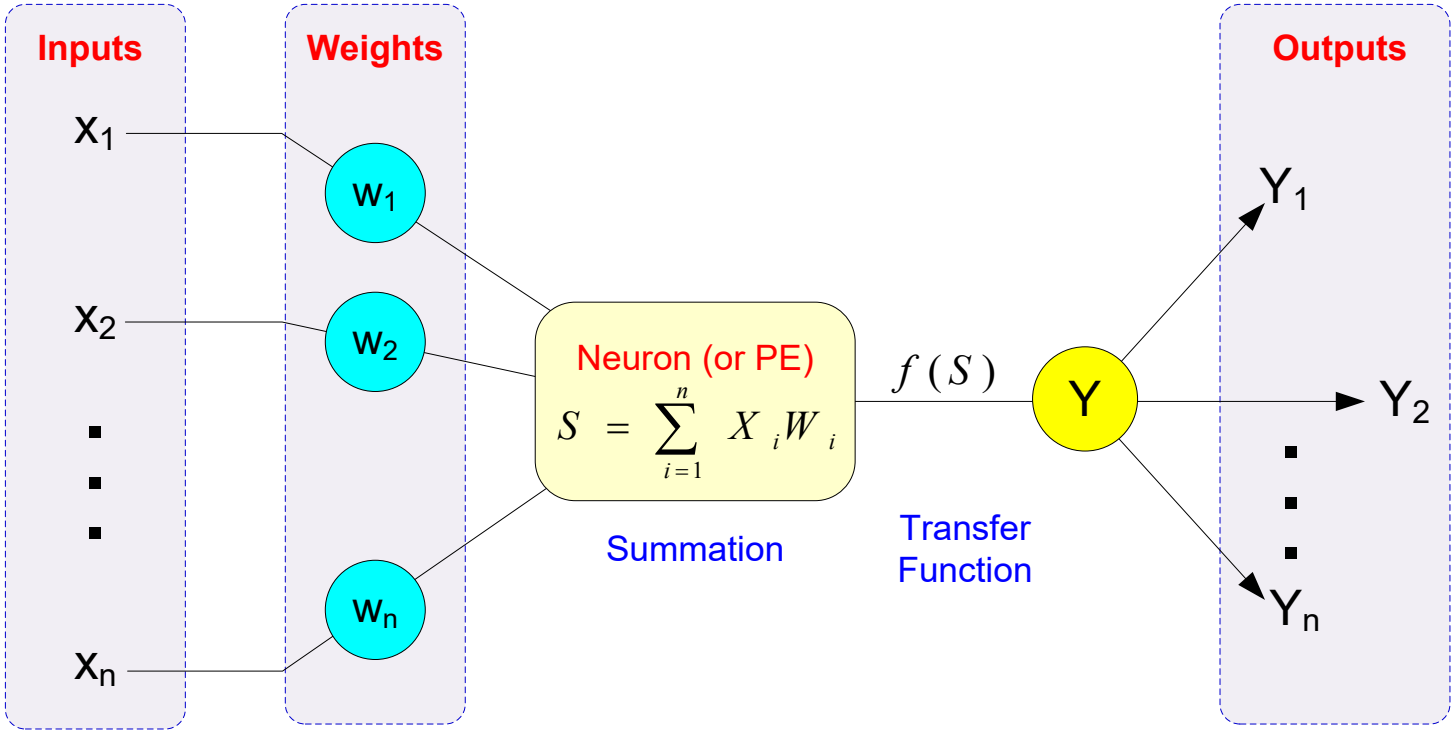
Artificial Neural Networks are inspired by biological neurons and their ability to process information.

An **artificial neuron** is a simplified computational model that mimics the behavior of a biological neuron. It takes inputs, processes them and produces an output.



Two interconnected brain cells (neurons)

Processing Information in ANN

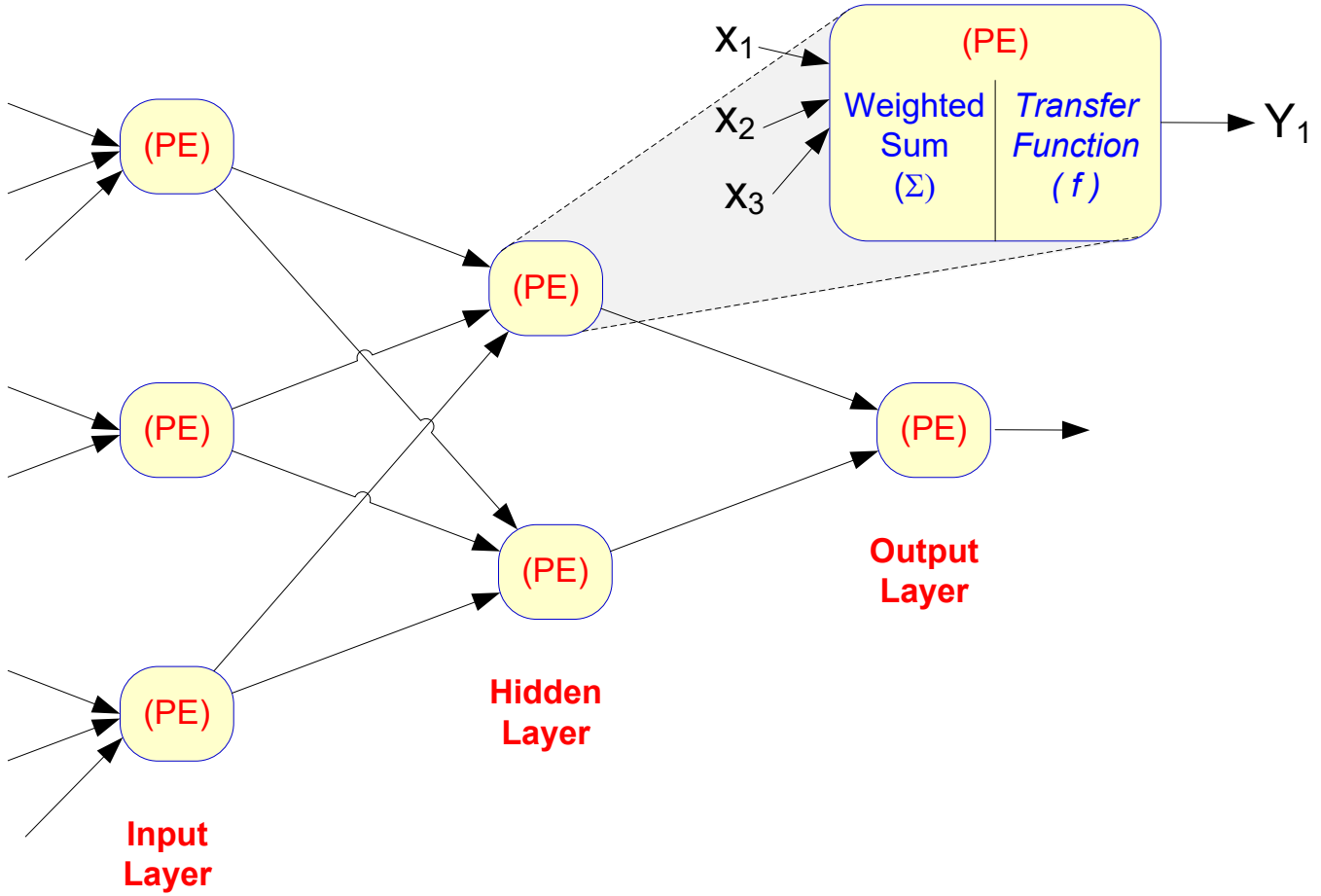


A single neuron (processing element – PE) with inputs and outputs

Biology Analogy

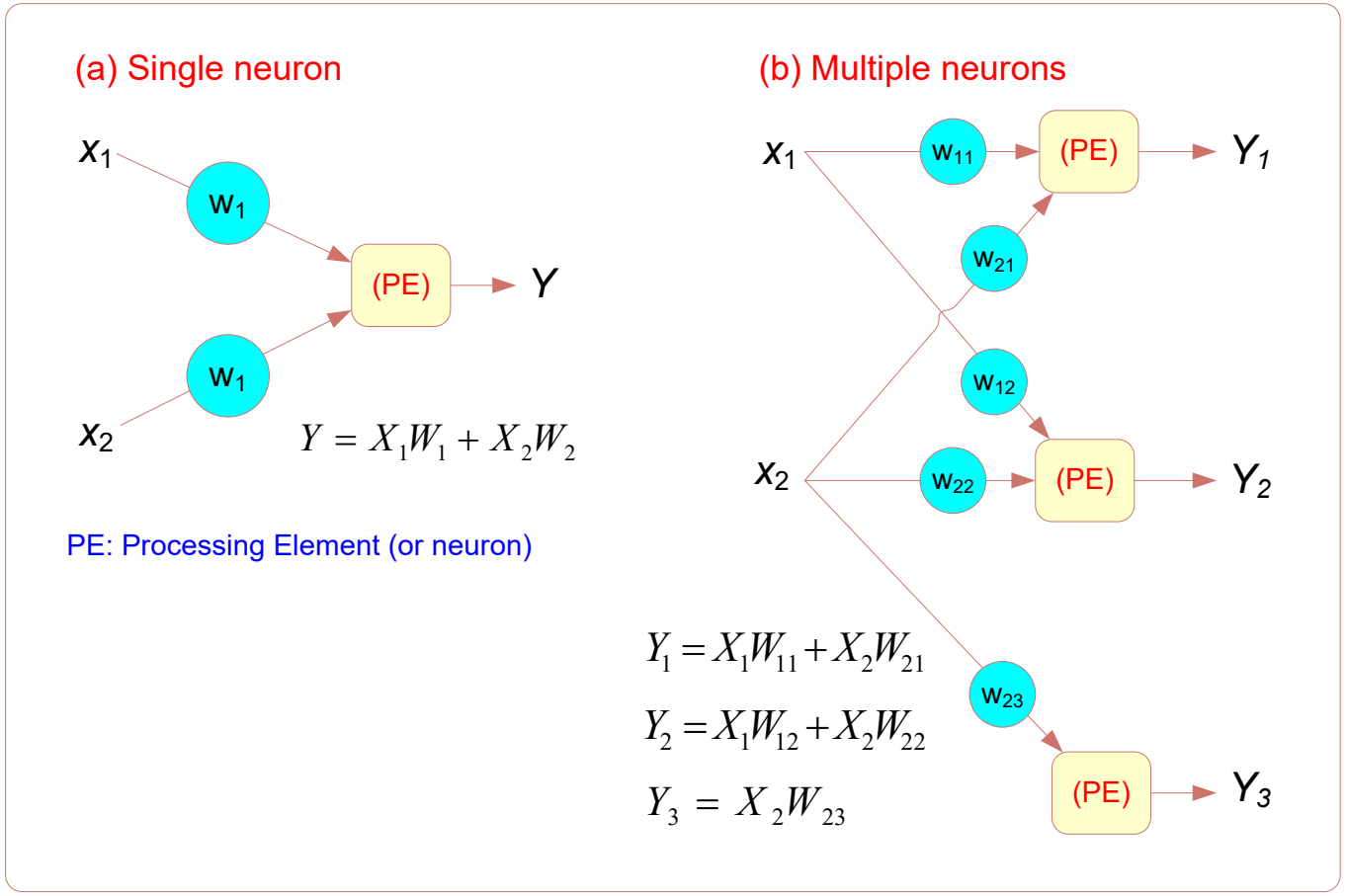
Biological	versus	Artificial	NNs
Soma		Node	
Dendrites		Input	
Axon		Output	
Synapse		Weight	
Slow		Fast	
Many neurons (10^9)		Few neurons (~100s)	

Elements of ANN



Neural Network with One Hidden Layer

Elements of ANN

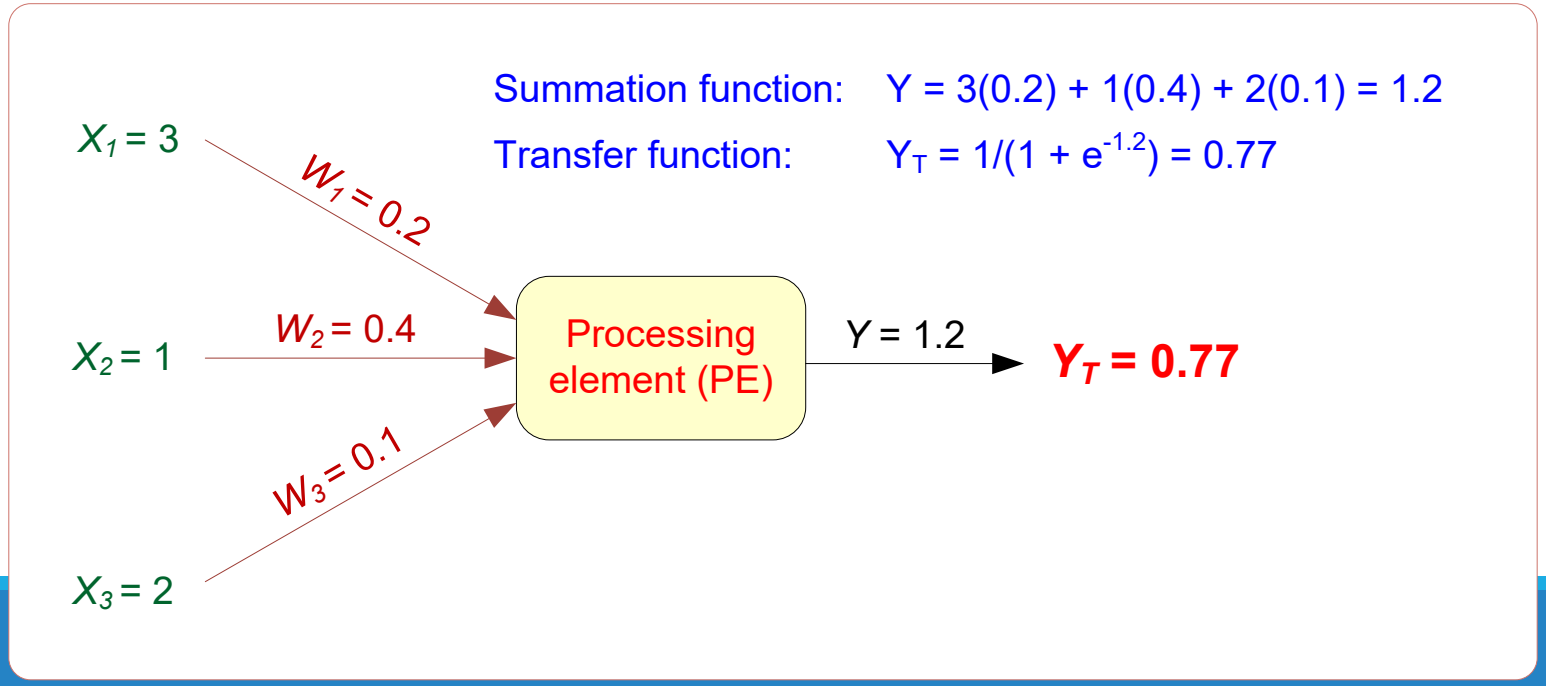


Summation Function for a Single Neuron (a), and Several Neurons (b)

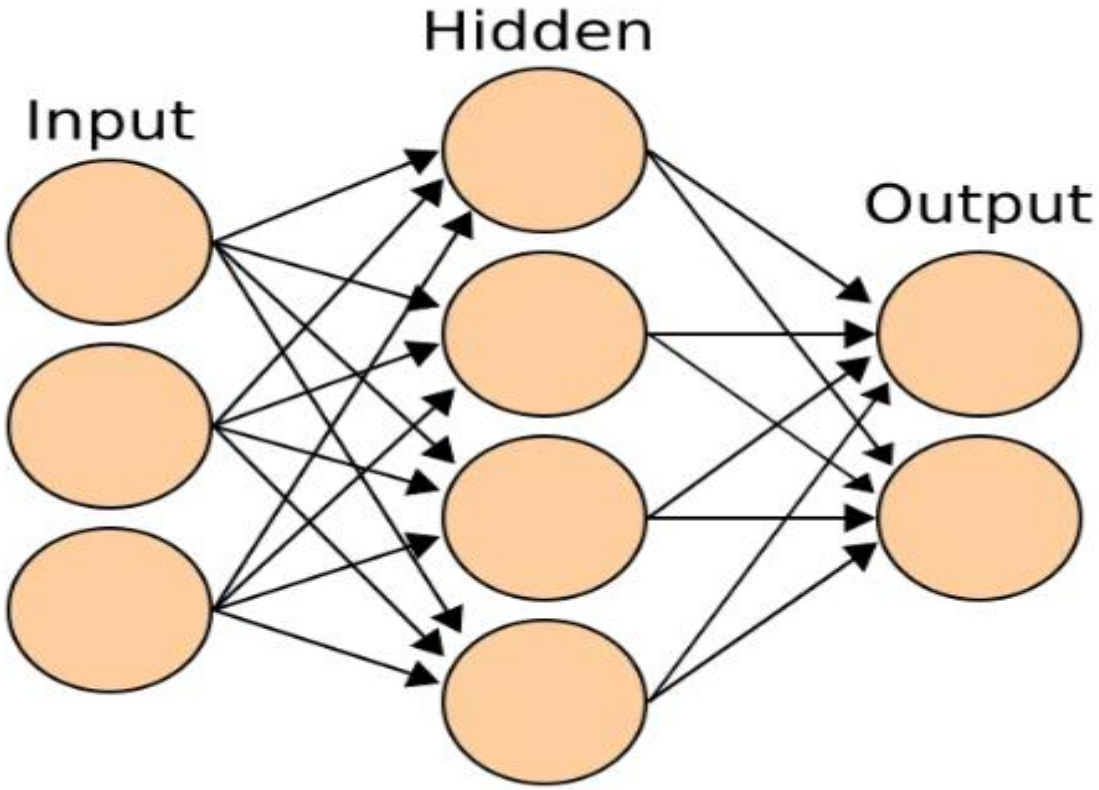
Elements of ANN

Transformation (Transfer) Function

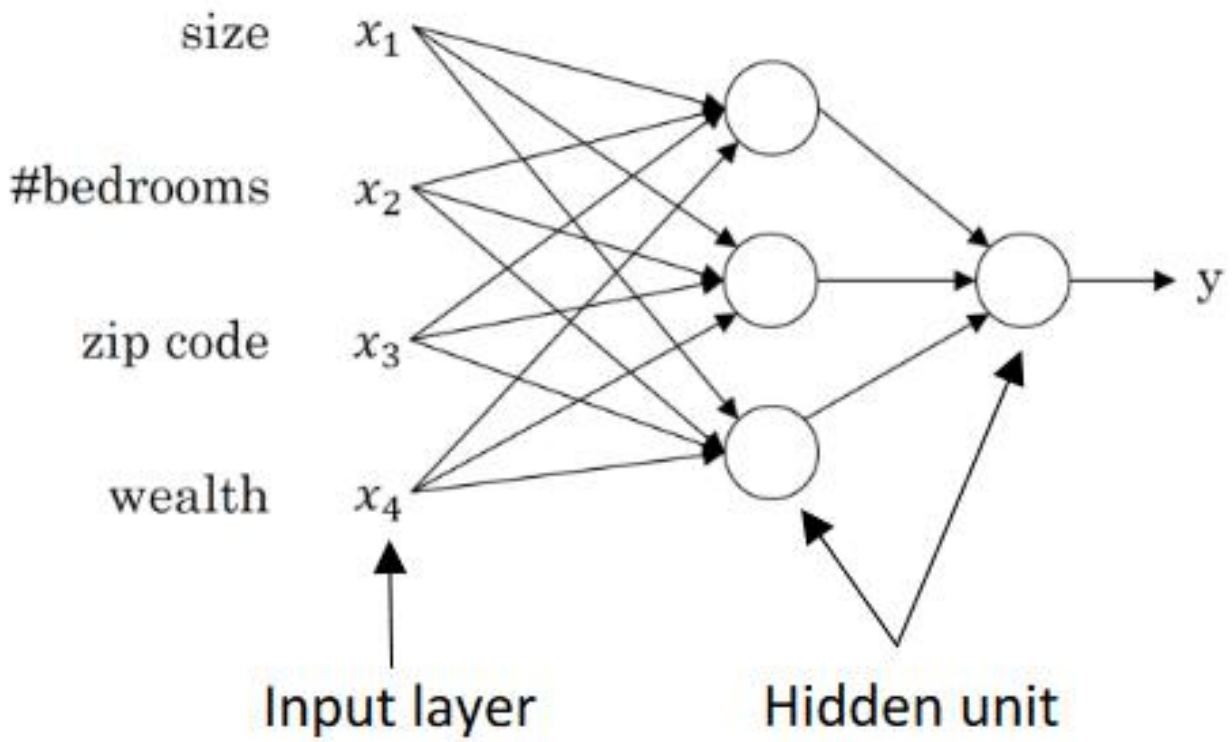
- Linear function
- Sigmoid (logical activation) function [0 1]
- Tangent Hyperbolic function [-1 1]



Basic Neural Network Architecture



Architecture of a standard NN



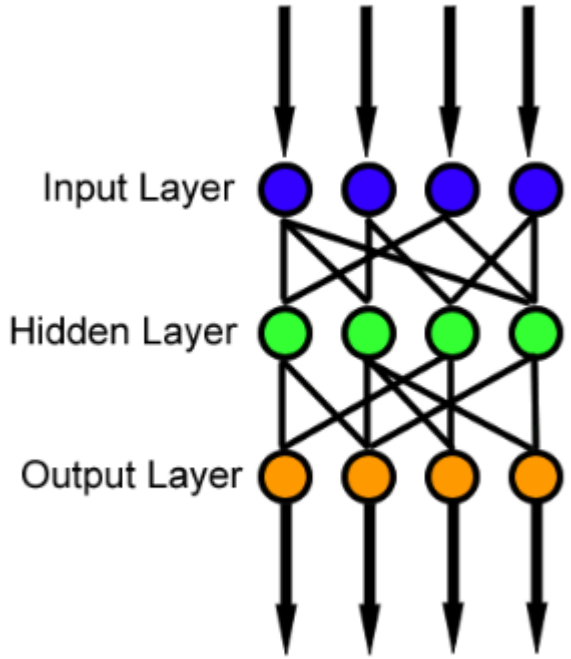
Types of Neural Networks in Deep Learning

Feedforward Neural Networks (FNN)

- Feedforward Neural Networks (FNN) are the most basic type of neural network where data moves in only one direction — from the input layer to the output layer. The data passes through the hidden layers, and no loops or cycles exist.

Example: Predicting house prices based on features like square footage, number of bedrooms, etc.

This diagram represents the simple architecture of a feedforward neural network, where the flow of data is strictly unidirectional.

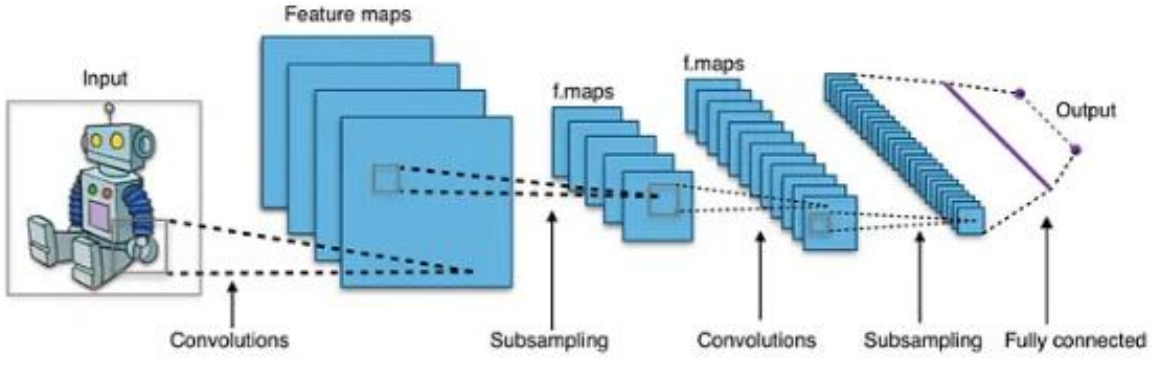


Types of Neural Networks in Deep Learning

Convolutional Neural Networks (CNN)

- CNNs are specialized neural networks primarily used for processing **image data**. CNNs are excellent at recognizing visual patterns like edges, textures, and shapes by using convolutional layers that apply filters (kernels) to the image.
- This makes CNNs widely used in image classification, object detection, and facial recognition tasks. Example: Classifying handwritten digits in the MNIST dataset.

This diagram shows the architecture of a Convolutional Neural Network (CNN), with convolutional and pooling layers designed to detect image features.



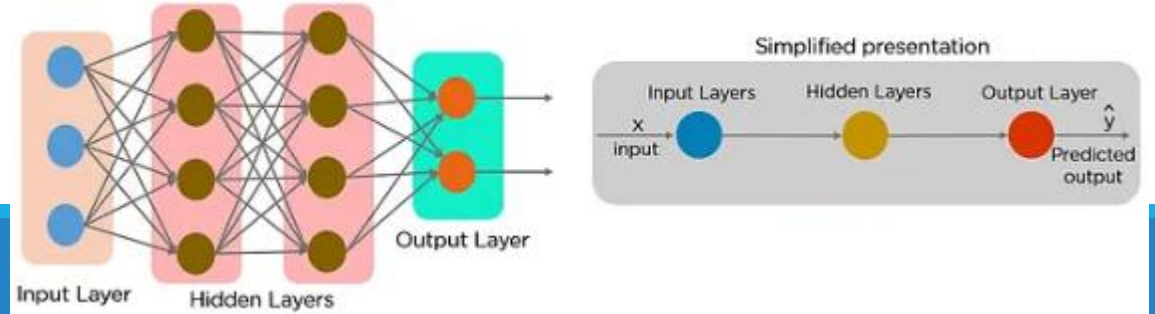
Types of Neural Networks in Deep Learning

Recurrent Neural Networks (RNN)

RNNs are used for processing **sequential data**. Unlike feedforward networks, RNNs have loops in their architecture, allowing information to be passed from one step to the next. This makes them useful for tasks like time series prediction, speech recognition, and natural language processing.

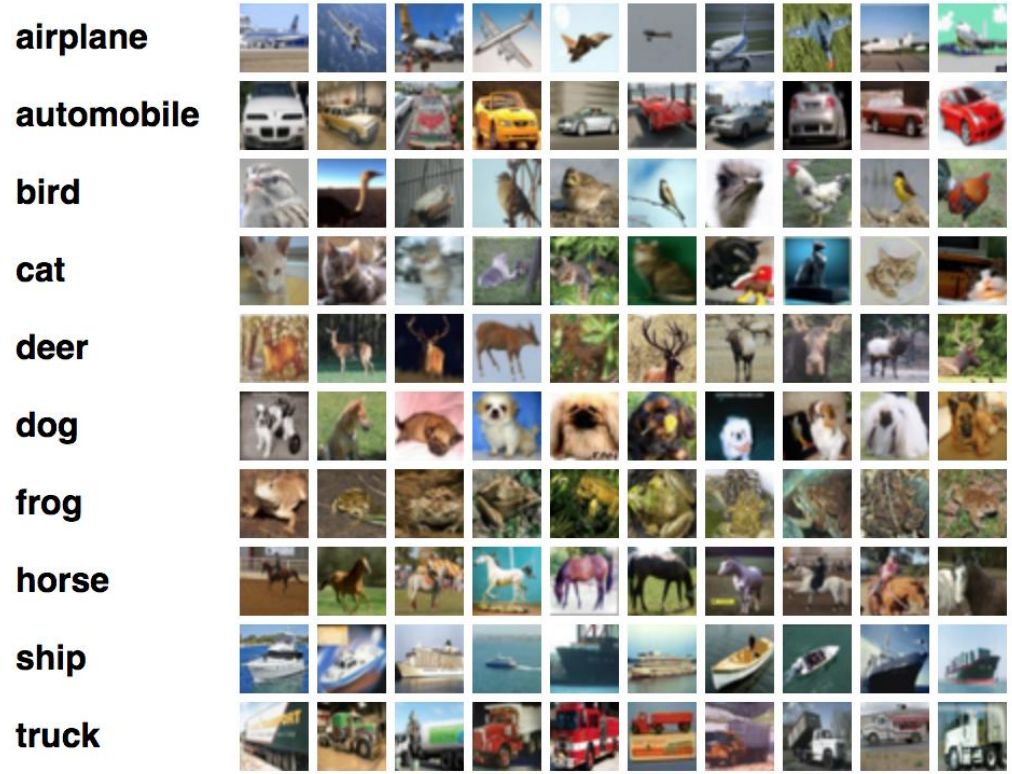
Example: Predicting the next word in a sentence or generating text.

The following diagram illustrates the architecture of an RNN, where the output of each step is fed back into the network for subsequent steps.



Example

Let's build a **neural network** classification algorithm for the CIFAR-10 dataset (10 classes, 32x32 images):



NN algorithm

dog

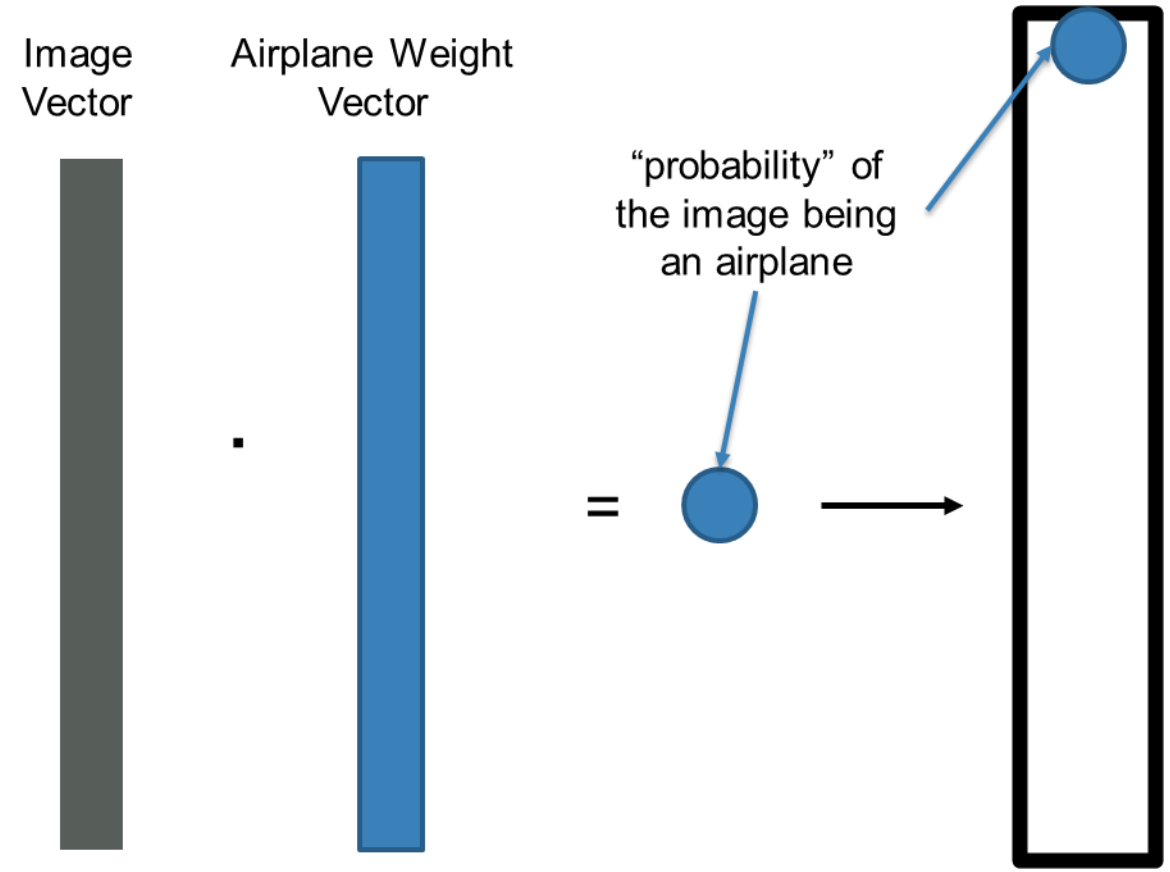


Image Vector
Airplane Weight Vector
“probability” of the image being an airplane

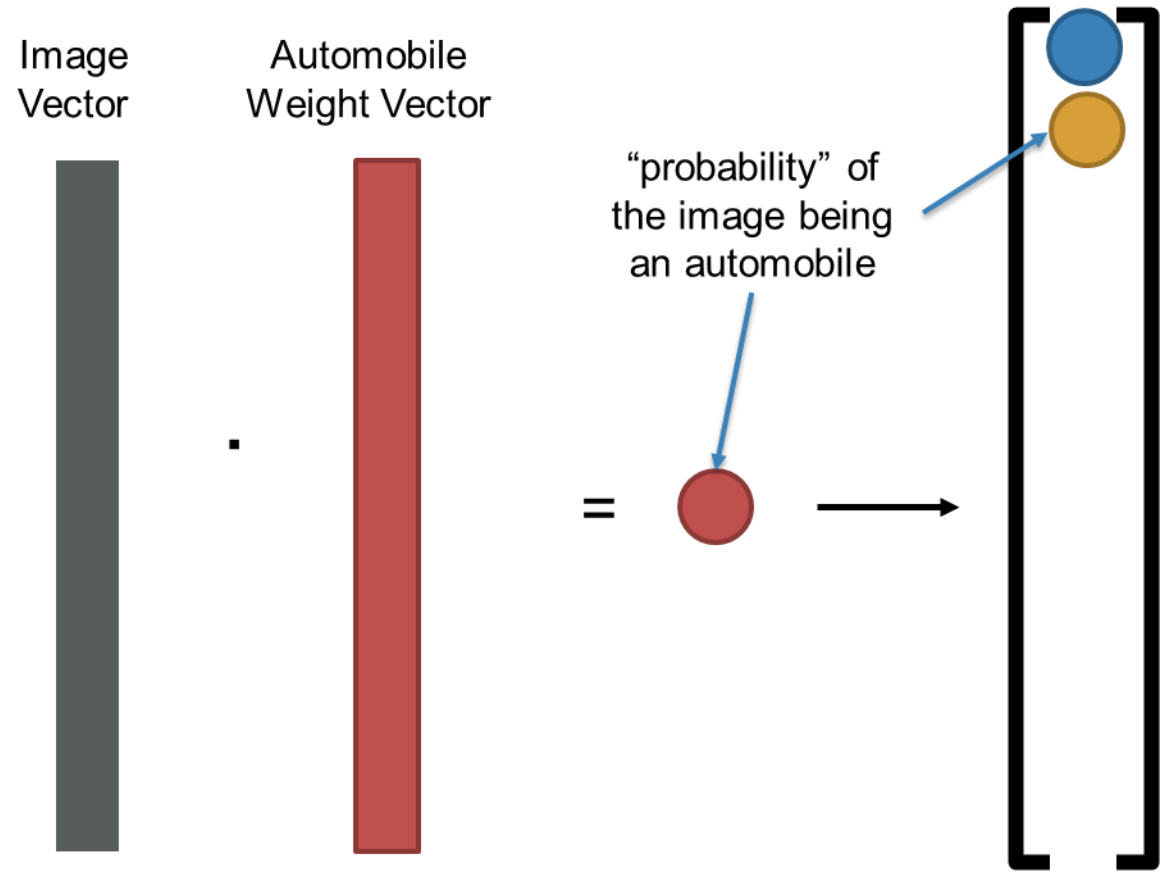


Image
Vector
Automobile
Weight Vector
“probability” of
the image being
an automobile

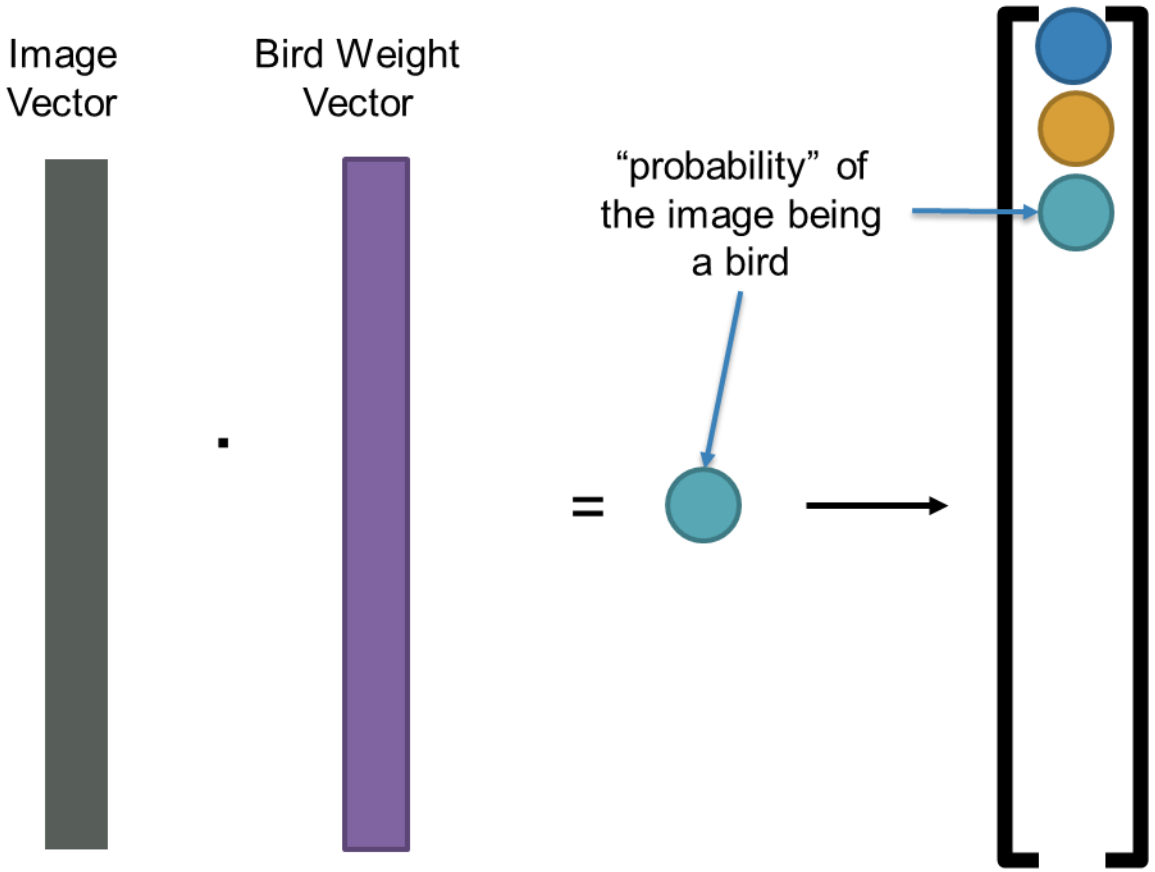


Image Vector
Bird Weight Vector
"probability" of the image being a bird
=

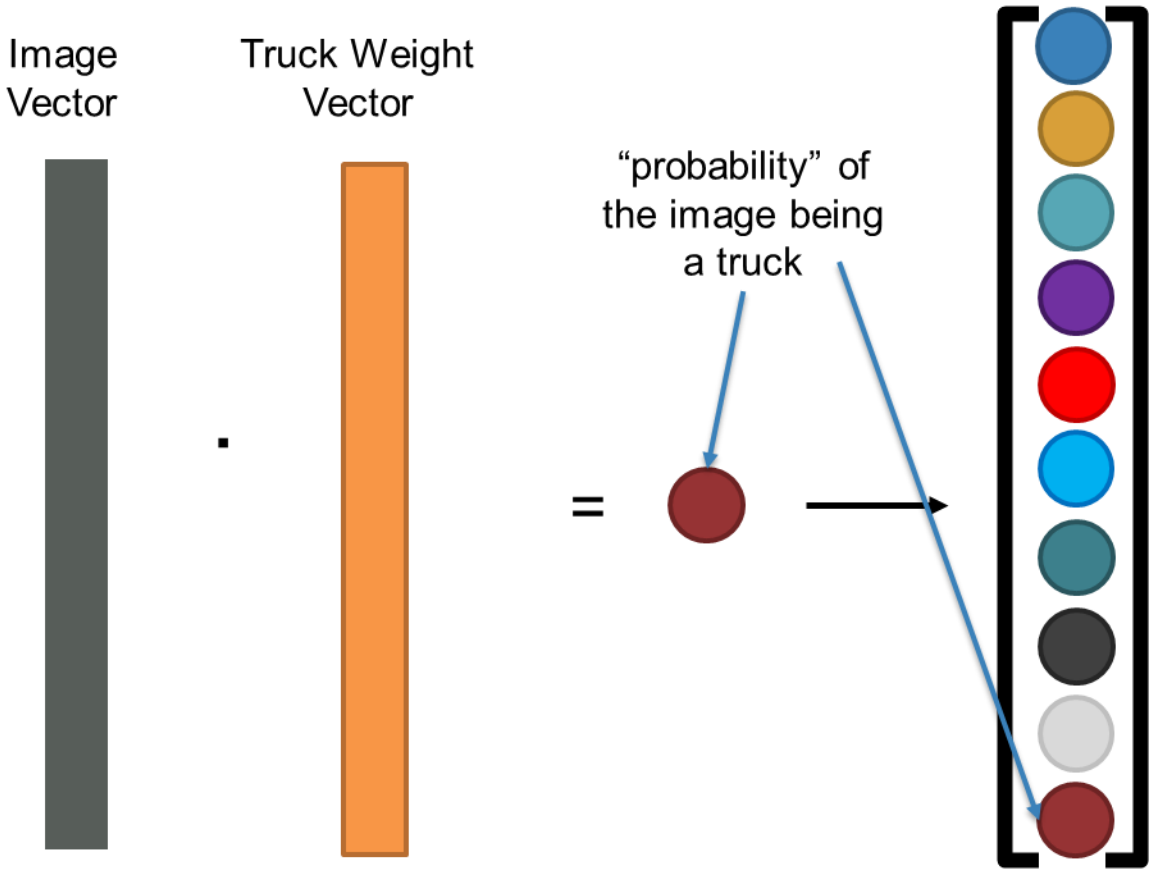
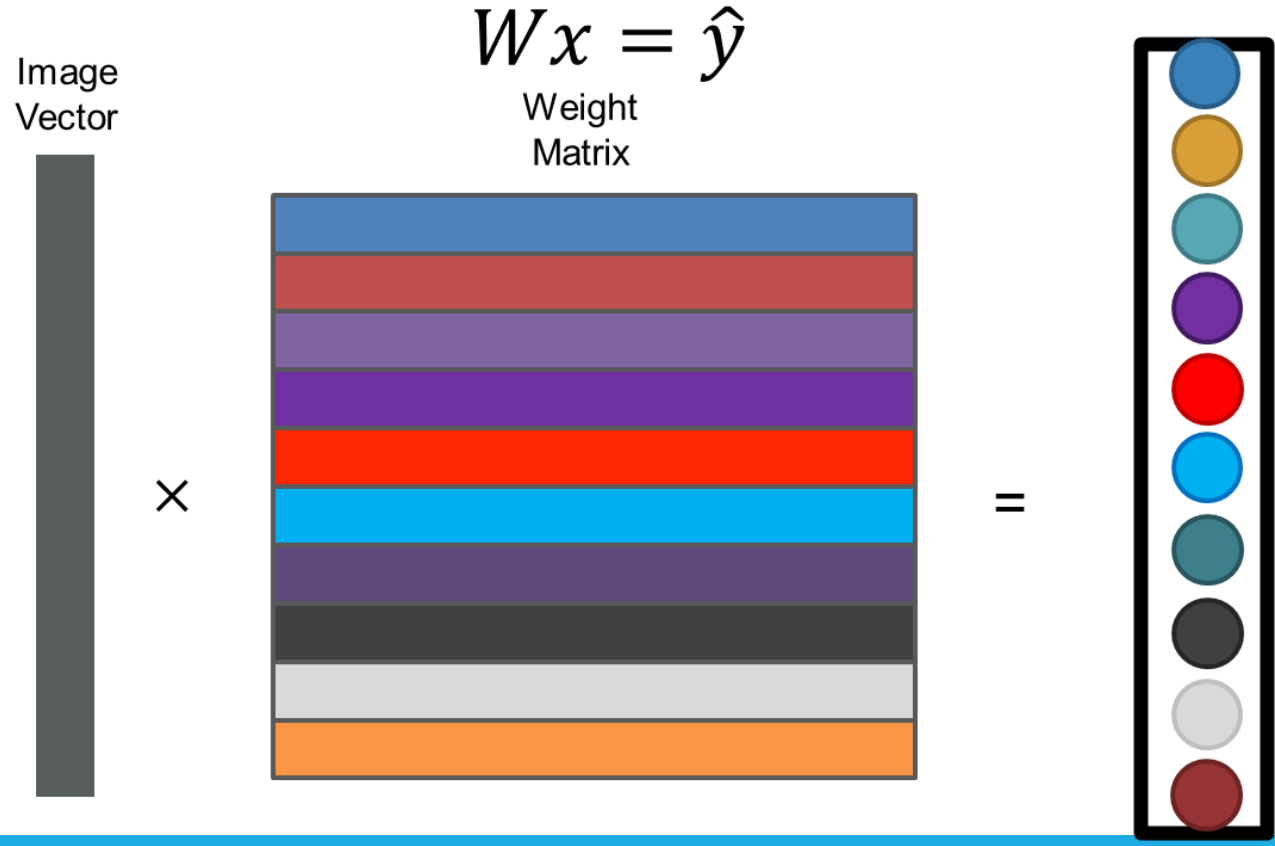


Image Vector
Truck Weight Vector
.
=
"probability" of the image being a truck

W: the (10x1024) matrix of weight vectors

x: the (1024x1) image vector (original image is 32x32 and after converting it into one vector became 1024x1)

$\hat{y}$: the (10x1) vector of class "probabilities"



Numerical Example - Step 1:

An image that's 2×2 pixels $\rightarrow$ flattened into a **4×1 vector** x

3 possible classes $\rightarrow$ so our **weight matrix** W will be **3×4**

Numerical Example - Step 2: Define our data

Image vector x:

$$x = \begin{bmatrix} 2 \\ 1 \\ 0 \\ 3 \end{bmatrix}$$

This represents pixel intensities from the image (flattened).

Weight matrix W:

Each row represents the weights for one class.

$$W = \begin{bmatrix} 1 & 0 & 2 & -1 \\ 0 & 1 & -1 & 2 \\ 2 & -2 & 0 & 1 \end{bmatrix}$$

Numerical Example - Step 3: Compute the scores

We multiply $W \times x$: $\hat{y} = Wx$

$$W = \begin{bmatrix} 1 & 0 & 2 & -1 \\ 0 & 1 & -1 & 2 \\ 2 & -2 & 0 & 1 \end{bmatrix} \qquad x = \begin{bmatrix} 2 \\ 1 \\ 0 \\ 3 \end{bmatrix}$$

Let's compute it row by row:

For Class 1: $(1)(2) + (0)(1) + (2)(0) + (-1)(3) = 2 + 0 + 0 - 3 = -1$

For Class 2: $(0)(2) + (1)(1) + (-1)(0) + (2)(3) = 0 + 1 + 0 + 6 = 7$

For Class 3: $(2)(2) + (-2)(1) + (0)(0) + (1)(3) = 4 - 2 + 0 + 3 = 5$

So, our output vector (scores) is: $\hat{y} = \begin{bmatrix} -1 \\ 7 \\ 5 \end{bmatrix}$

Numerical Example - Step 4: Convert to probabilities (Softmax)

Step 4: Convert to probabilities (Softmax)

$$p_i = \frac{e^{\hat{y}_i}}{\sum_j e^{\hat{y}_j}}$$

Compute:

$$e^{-1} = 0.3679$$
$$e^{7} = 1096.63$$
$$e^{5} = 148.41$$

Sum = 0.3679 + 1096.63 + 148.41 = **1245.41**

Then:

$$p_1 = 0.3679/1245.41 = 0.0003$$
$$p_2 = 1096.63/1245.41 = 0.880$$
$$p_3 = 148.41/1245.41 = 0.119$$

Numerical Example - Step 5: Interpretation

Class	Score $\hat{y}$	Probability p_i
1	-1	0.0003
2	7	0.880
3	5	0.119

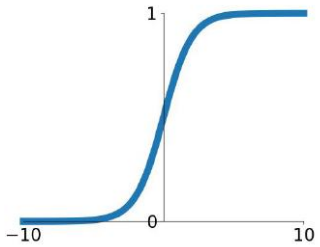
Predicted class = Class 2 (highest probability)

Activation functions

Activation functions

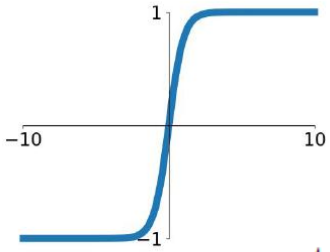
- Activation functions are mathematical equations that determine the output of a neural network.
- The function is attached to each neuron in the network and determines whether it should be activated (“fired”) or not, based on whether each neuron’s input is relevant for the model’s prediction.
- Activation functions also help normalize the output of each neuron to a range between 1 and 0 or between -1 and 1.

Different activation functions



Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

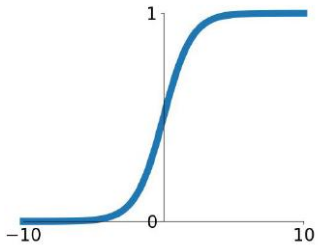


tanh(x)

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

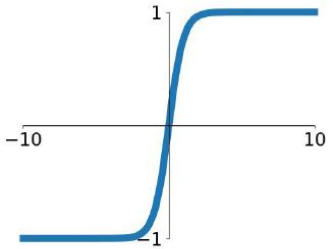
- Saturated neurons "kill" the gradients
  - When the input x is **very large positive** or **very large negative**, both sigmoid and tanh functions **flatten out**.
  - In those flat regions, the **slope ≈ 0**
  - The slope is what backpropagation uses as the **gradient**
  - So, gradients **vanish** → **weights stop updating**
  - This is the **vanishing gradient problem**.

Different activation functions



Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

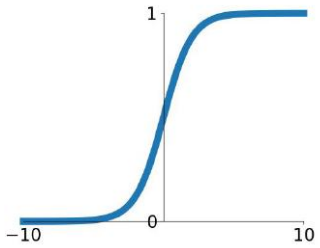


tanh(x)

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

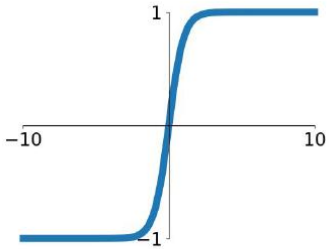
- **Sigmoid outputs are not zero-centered**
  - Sigmoid outputs are between **0 and 1**
  - So activations are **always positive**
  - This causes the next layer's inputs to be **biased in one direction**
  - Makes optimization harder — gradient updates can zigzag inefficiently.
  - **Tanh** improves this: outputs are between **-1 and 1**, which are **zero-centered**.

Different activation functions



Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

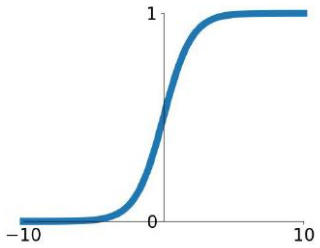


tanh(x)

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

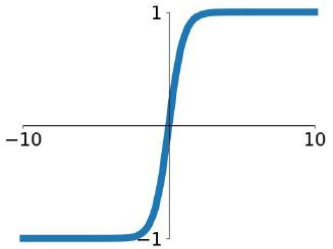
- **exp() is computationally expensive**
  - Both sigmoid and tanh rely on the **exponential function (exp)** in their definitions.
  - e^{-x} is used in sigmoid
  - Both e^{x} and e^{-x} appear in tanh
    So they're **slower to compute** compared to functions like ReLU ().

Different activation functions



Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



tanh(x)

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

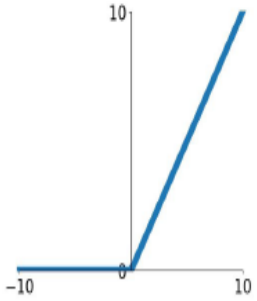
- **"Still kills gradients when saturated"**
  - Even tanh (which is zero-centered) has **flat regions** at both extremes (outputs close to ±1).

    That means when neurons enter these regions, **gradients again become very small**, slowing or halting learning.

Different activation functions

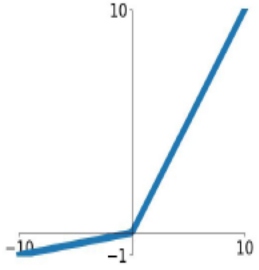
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$



- **Does not saturate**
  - Unlike sigmoid or tanh, ReLU **doesn't flatten** for large x.

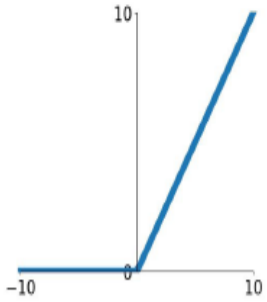
    → Gradient remains **1** for positive inputs, so **no vanishing gradient** on that side.

    → Helps deep networks train effectively.

Different activation functions

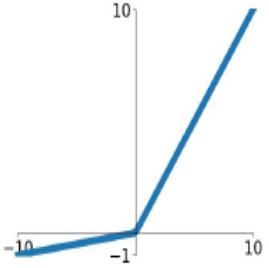
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

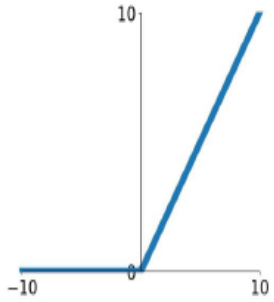


- **Computationally efficient**
  - ReLU uses only a **simple threshold operation** (no exponentials or divisions).
    → Very **fast** to compute, ideal for GPUs and deep nets.
- **Converges much faster**
  - Because ReLU keeps strong gradients for positive inputs, **learning is faster** and convergence occurs in fewer iterations compared to sigmoid/tanh.

Different activation functions

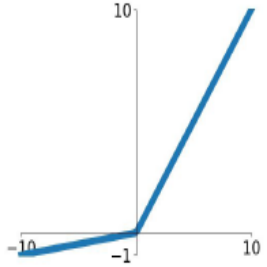
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

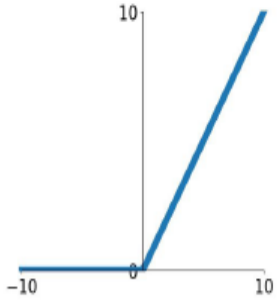


- **Not zero-centered output**
  - Outputs are **always ≥ 0**.
  - That means all activations are **positive**, which can cause the same bias shift issue as sigmoid — but in practice, ReLU still performs extremely well due to its simplicity and speed.
- **Losing half the spectrum**
  - For all **negative inputs**, ReLU outputs **0**. This means that about half of the neurons might be **inactive (output = 0)** at any time.

Different activation functions

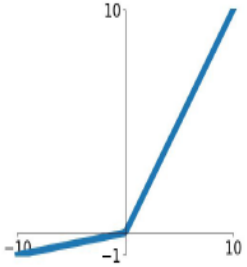
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

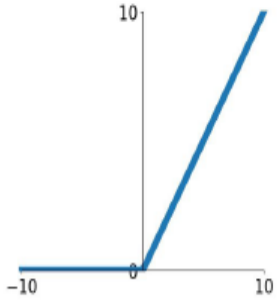


- **Problem: "Dead ReLUs"**
  - When weights are updated such that a neuron's input is **always negative**, its output becomes permanently **0** — and its gradient **0** → it **never recovers**.
    → This neuron is called a **"dead ReLU."**

Different activation functions

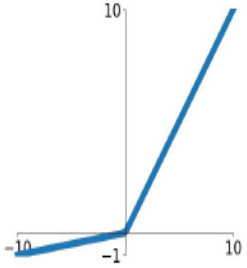
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$



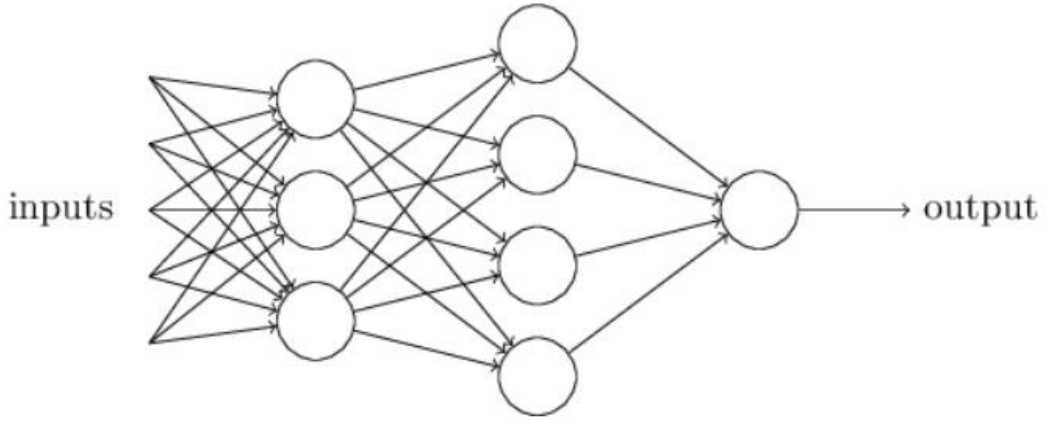
- **Neurons will not die**
  - Leaky ReLU introduces a **small slope** for negative x, instead of making it 0.
    This means:
  - The gradient is **never completely 0**,
  - So neurons **will not "die."**

Deep Neural Networks

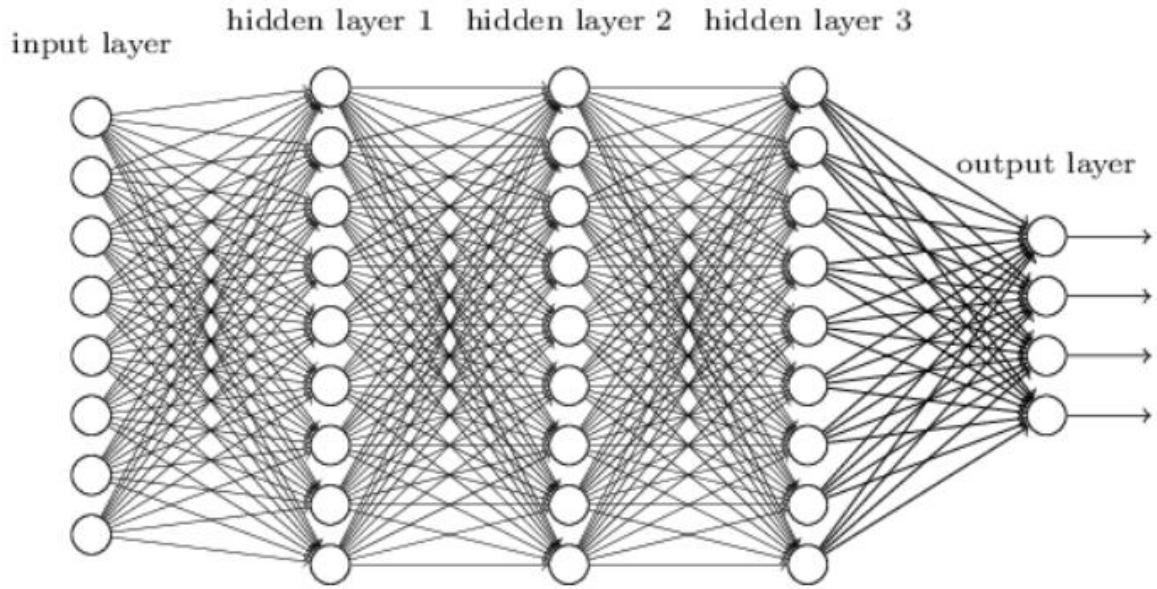
Deep Neural Networks

With more than one hidden layer, e.g.

3-layer fully connected

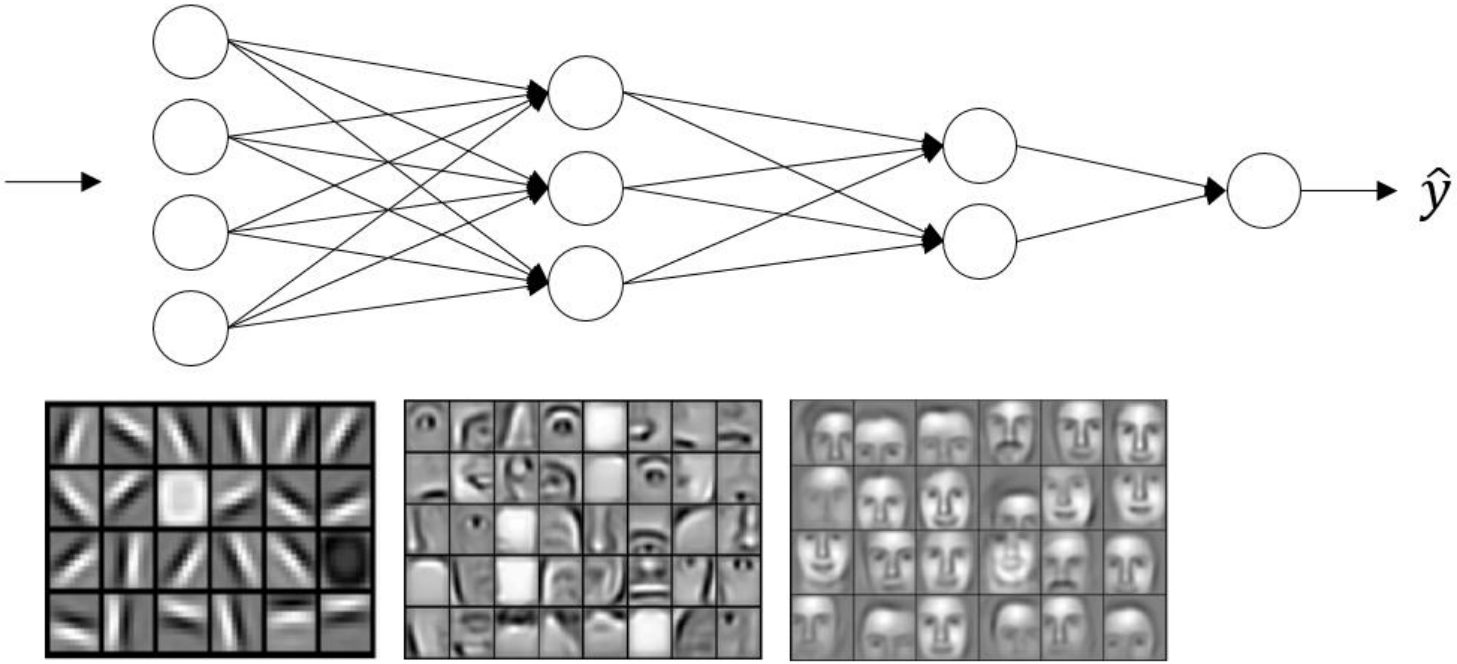


4-layer fully connected



Why Deep Neural Networks

They seem to learn complex structures in stages starting from simple features to complex features



Hyper-parameters

- Learning rate
- Number of layers
- Neuron/layers
- Activation function
- Others like number of iterations, optimization strategy.

Overfitting (Variance)

- Overfitting happens when a model **learns the training data too well**, including **noise and random fluctuations**, instead of learning the **underlying pattern**.

  As a result:

  - Training accuracy is **very high**,
  - But test/validation accuracy is **poor**.
  - So, we need methods to make the model **generalize better**

Dealing With Overfitting (Variance)

- L_p regularization
- Dropout
  - Decide dropout rate {0.5, 0.2,…}
  - Set some activations to be 0 and rescale all the remaining activations
  - Only during training
- Data augmentation
- Early stopping

Dealing With Overfitting (Variance)

More data

- The most natural and effective way to reduce overfitting.
- With more examples, the model can learn the **true distribution** rather than memorizing specific samples.
- However, in practice, collecting more data can be expensive or impossible — so we use other tricks like *data augmentation* (see below).

Dealing With Overfitting (Variance)

Lp Regularization

- This refers to **penalizing large weights** in the model by adding a regularization term to the loss function. $Loss_{total} = Loss_{data} + \lambda \parallel w \parallel_p$

where:

- λ is a hyperparameter controlling penalty strength, p determines the type of regularization.

Common types:

- L_1 **regularization** ($\parallel w \parallel_1 = \sum \mid w_i \mid$) → encourages **sparse weights** (some weights become 0).
- L_2 **regularization** ($\parallel w \parallel_2 = \sum w_i^2$) → discourages large weights but doesn't make them exactly zero (also called **weight decay**).

Dealing With Overfitting (Variance)

Dropout

- During **training**, randomly "drop" (set to 0) some neuron activations with a probability p (the **dropout rate**, e.g., 0.5 or 0.2).
- The network is forced to **not rely too much on any one neuron**.
- The remaining neurons are **rescaled** (usually divided by $1 - p$) to maintain the same expected output.
- **During testing/inference:**
- No dropout — all neurons are active.
- But the weights are scaled appropriately so the output matches expected values.
- *Intuition:* Dropout acts like training **many smaller networks** and averaging their predictions $\rightarrow$ reduces overfitting.

Dealing With Overfitting (Variance)

Data Augmentation

- Artificially increase your dataset by **transforming existing samples** (e.g., rotating, flipping, cropping, adjusting brightness in images).
- Helps the model learn **invariant features** (like object recognition under different orientations).
- Greatly improves generalization, especially in computer vision tasks.

Dealing With Overfitting (Variance)

Early Stopping

- Monitor model performance on **validation data** during training.
- Stop training **when validation loss starts to increase**, even if training loss keeps decreasing.
- Prevents the network from memorizing the training data too long.

Practical issues

- Scaling
- Vanishing/exploding gradients
  - Weight initialization:
  - weights = np.random.randn(dim) * np.sqrt(2/n)), or:
  - weights = np.random.randn(dim) * np.sqrt(1/n)), [Xavier initialization]
- Batch/mini-batch/stochastic gradient descent

Convolutional Neural Networks

Why CNNs

- Used ‘mostly’ with images
- Translation invariance

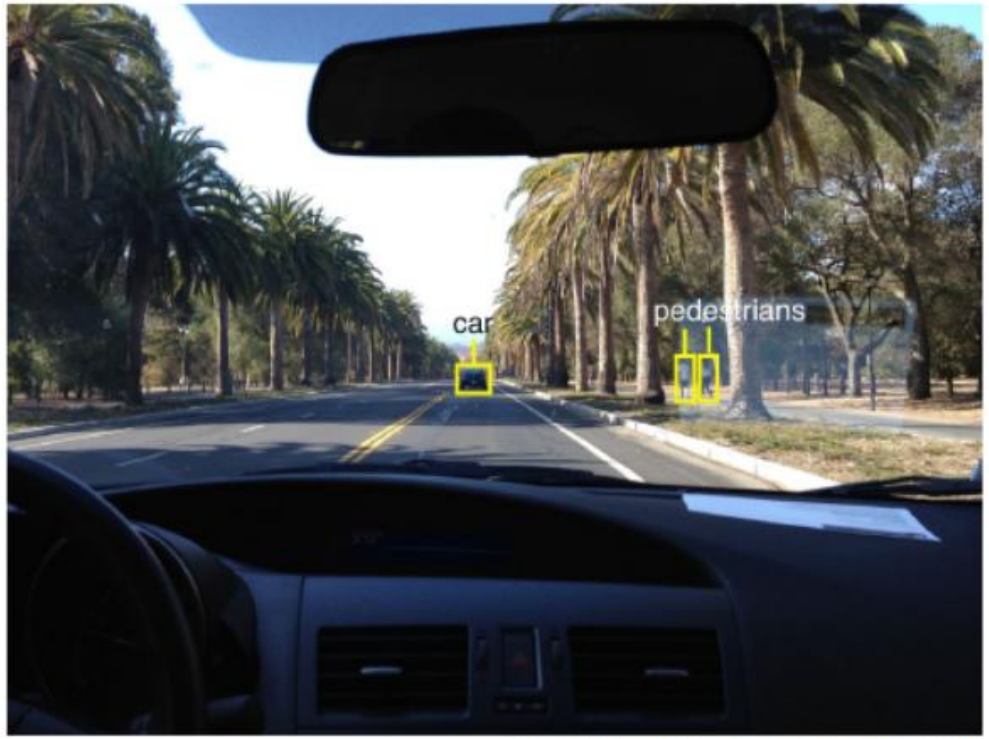
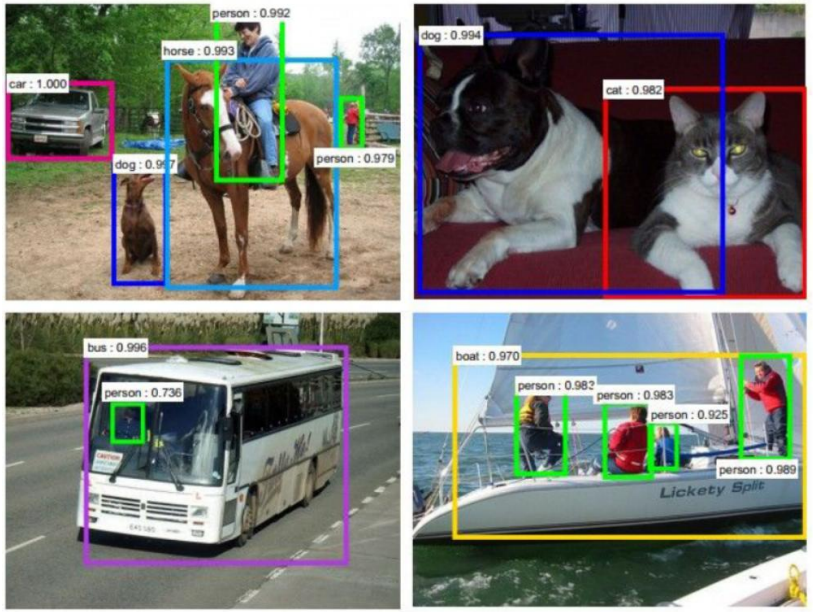


Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars

Fast-forward to today: ConvNets are everywhere

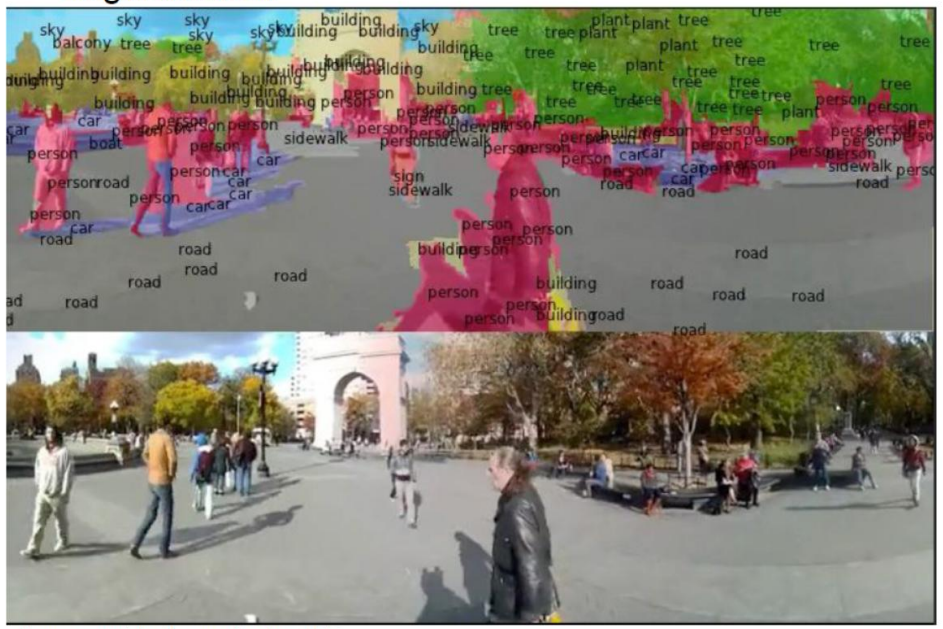
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



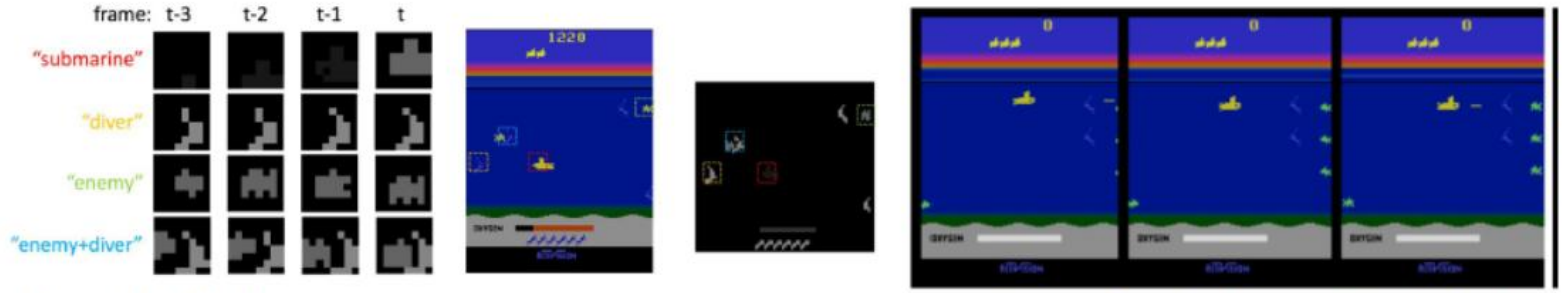
Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

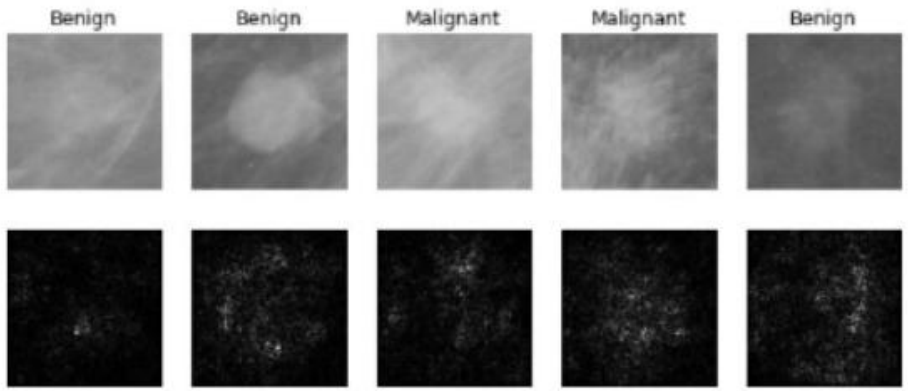
[Toshev, Szegedy 2014]



[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

Fast-forward to today: ConvNets are everywhere



[Levy et al. 2016]

[Dieleman et al. 2014]

[Sermanet et al. 2011]
[Ciresan et al.]

Fast-forward to today: ConvNets are everywhere

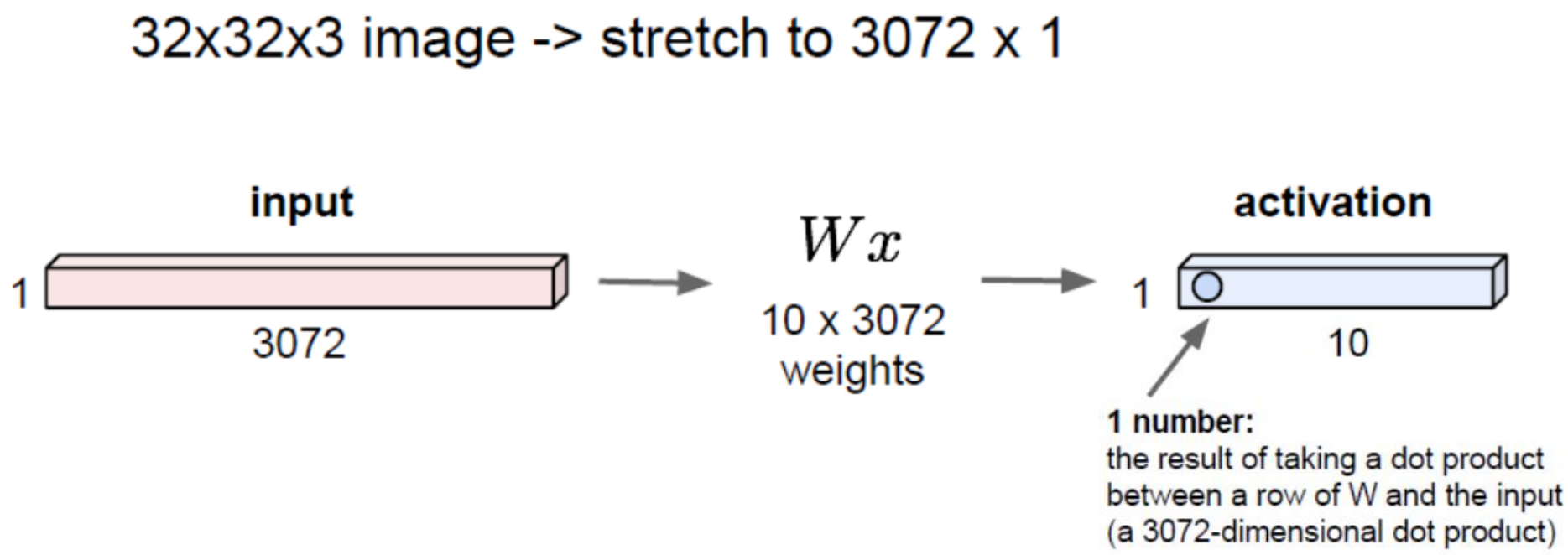
This image by Christin Khan is in the public domain and originally came from the U.S. NOAA.

Whale recognition, Kaggle Challenge

Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.

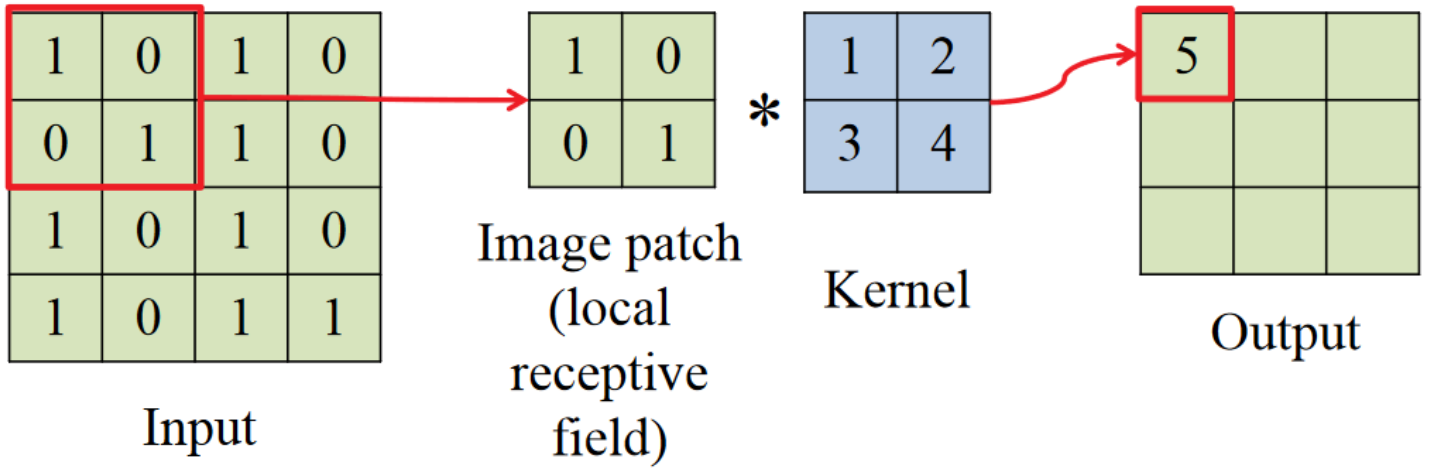
Mnih and Hinton, 2010

Fully Connected Layer



Convolution Layer

A dot product of a kernel (aka filter) and a patch of an image (local receptive field) of the same size.



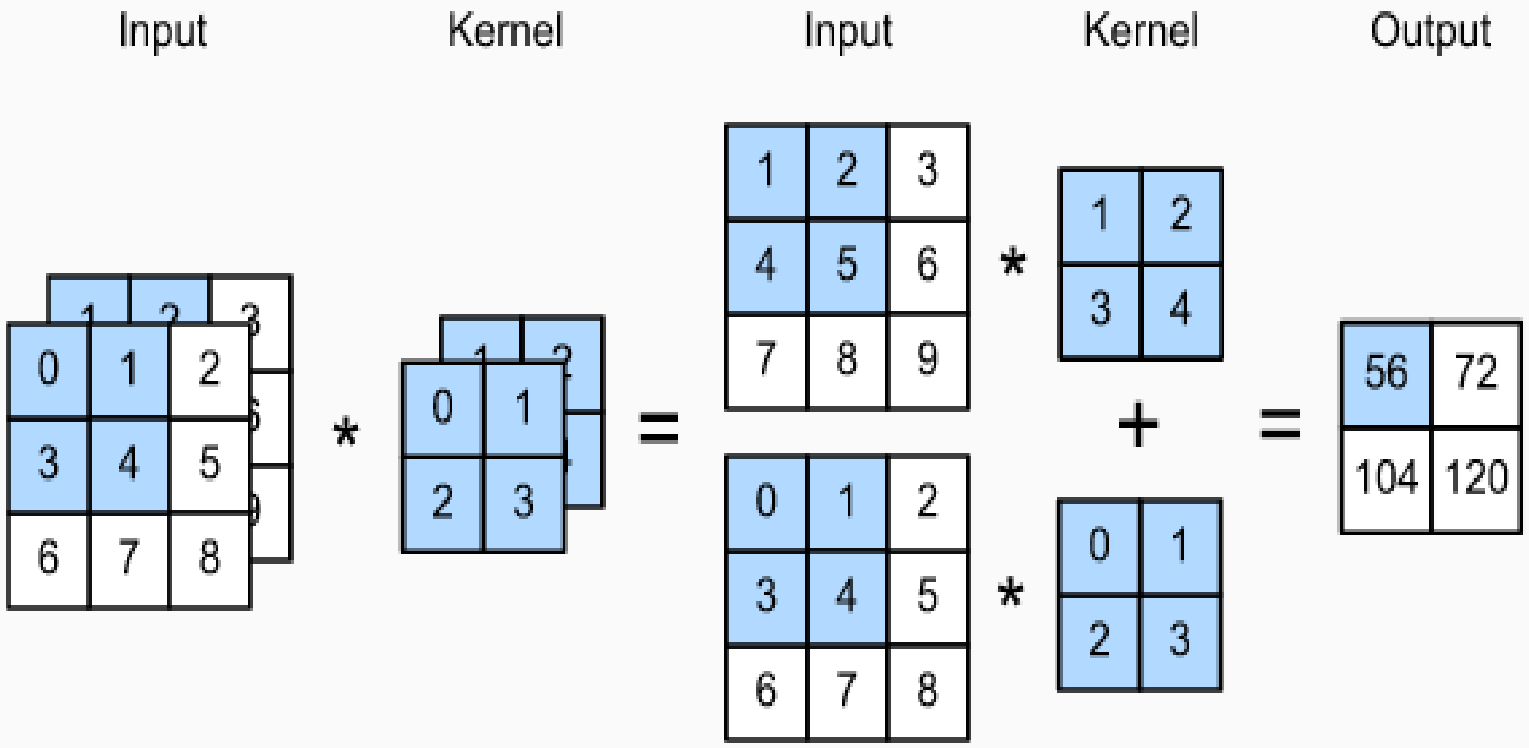
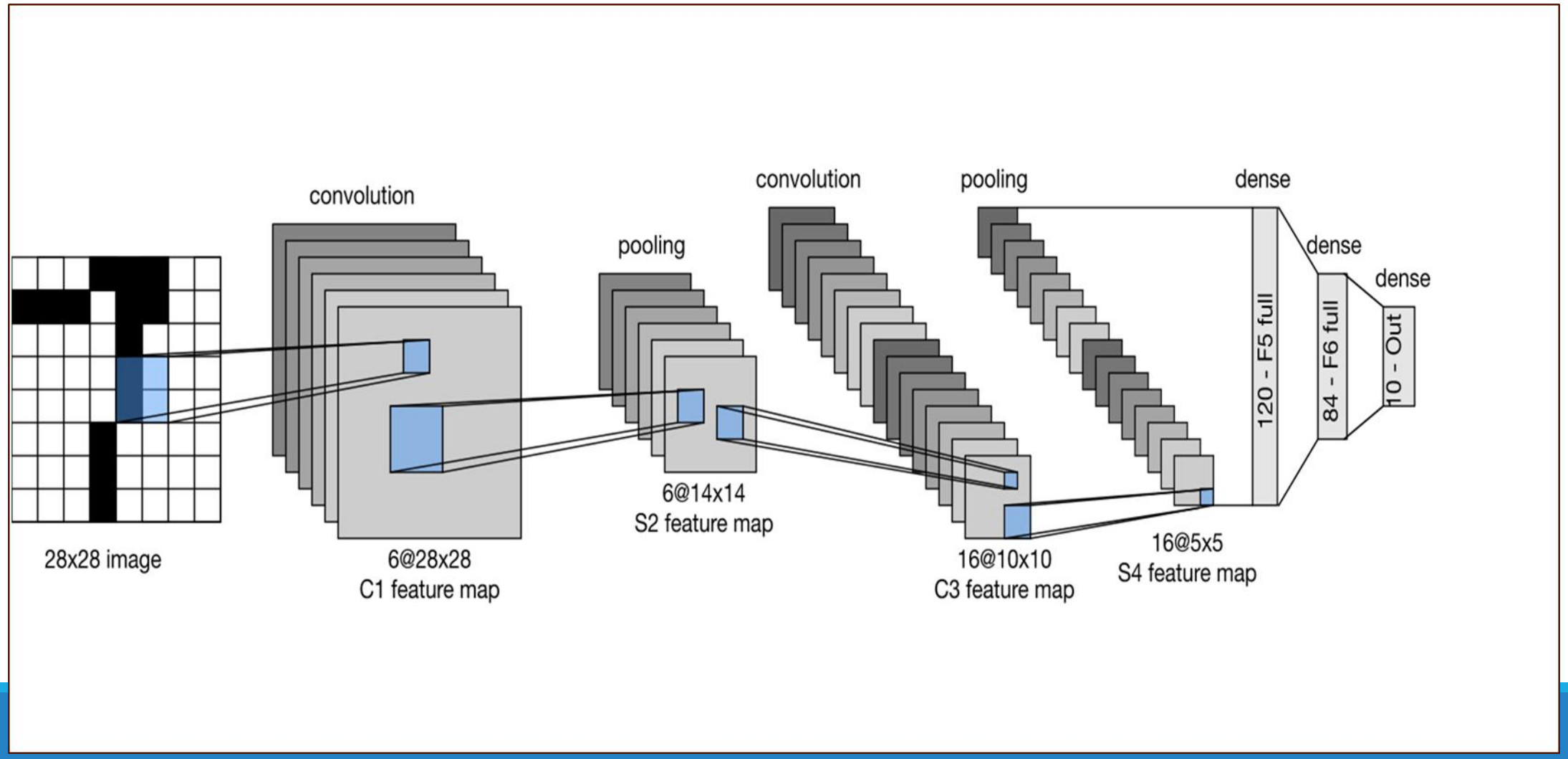


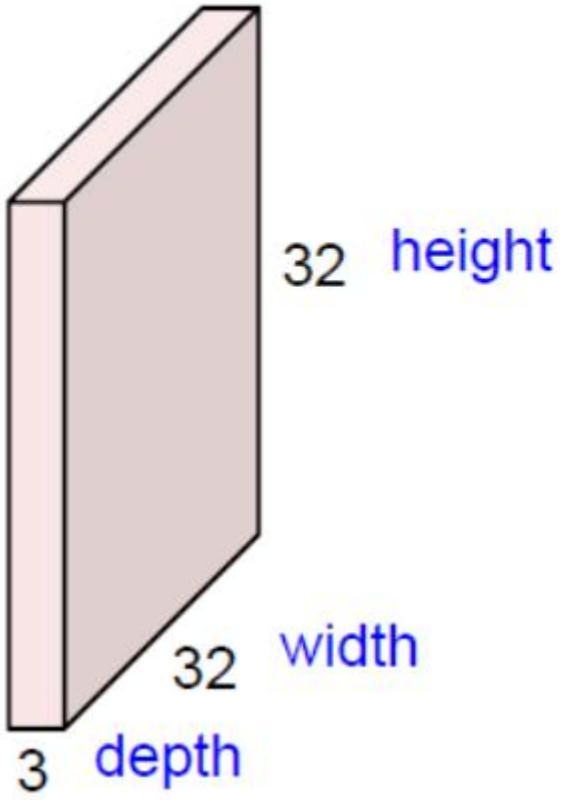
Fig. 6.4.1 Cross-correlation computation with 2 input channels.

CNN

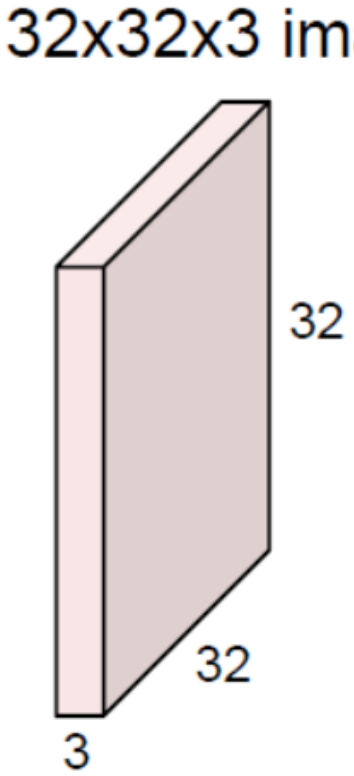

convolution
pooling
convolution
pooling
dense
dense
dense
120 - F5 full
84 - F6 full
10 - Out
28x28 image
6@28x28
C1 feature map
6@14x14
S2 feature map
16@10x10
C3 feature map
16@5x5
S4 feature map

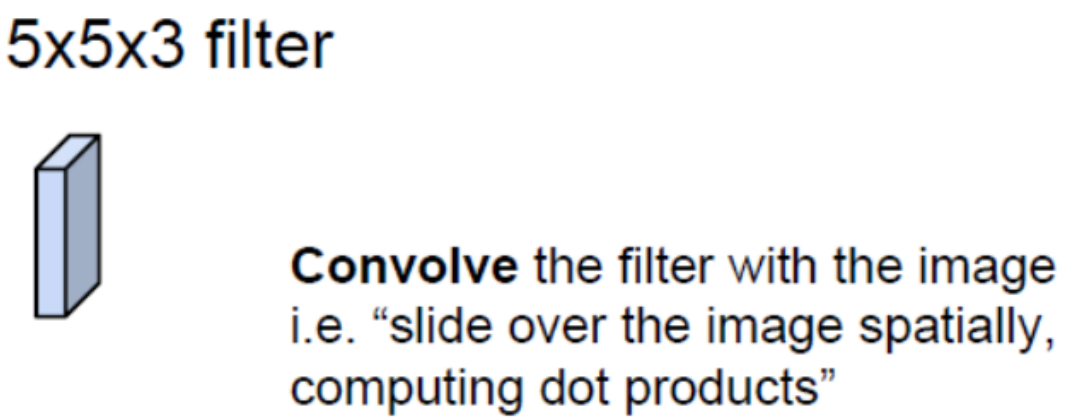
Convolution Layer

32x32x3 image -> preserve spatial structure

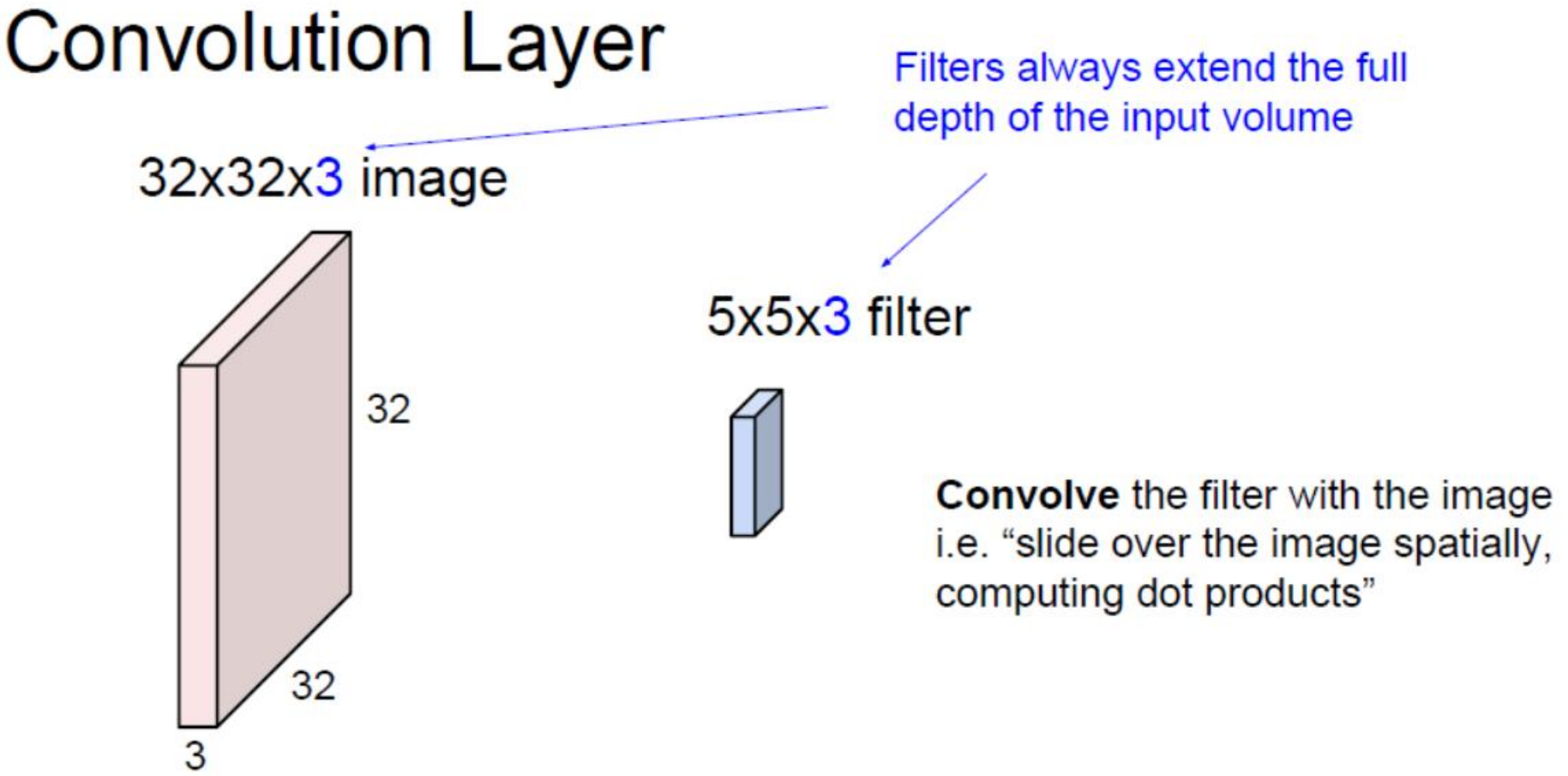


Convolution Layer

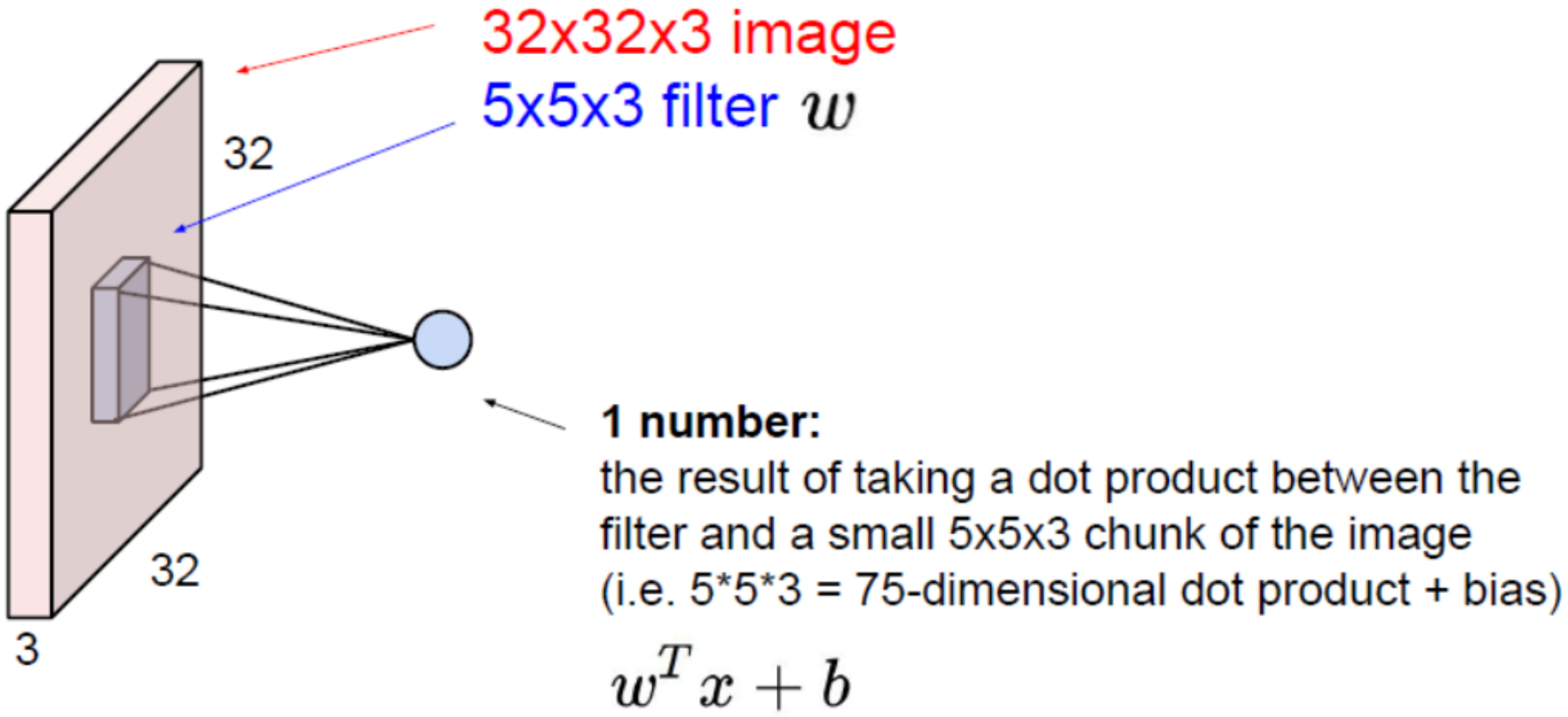




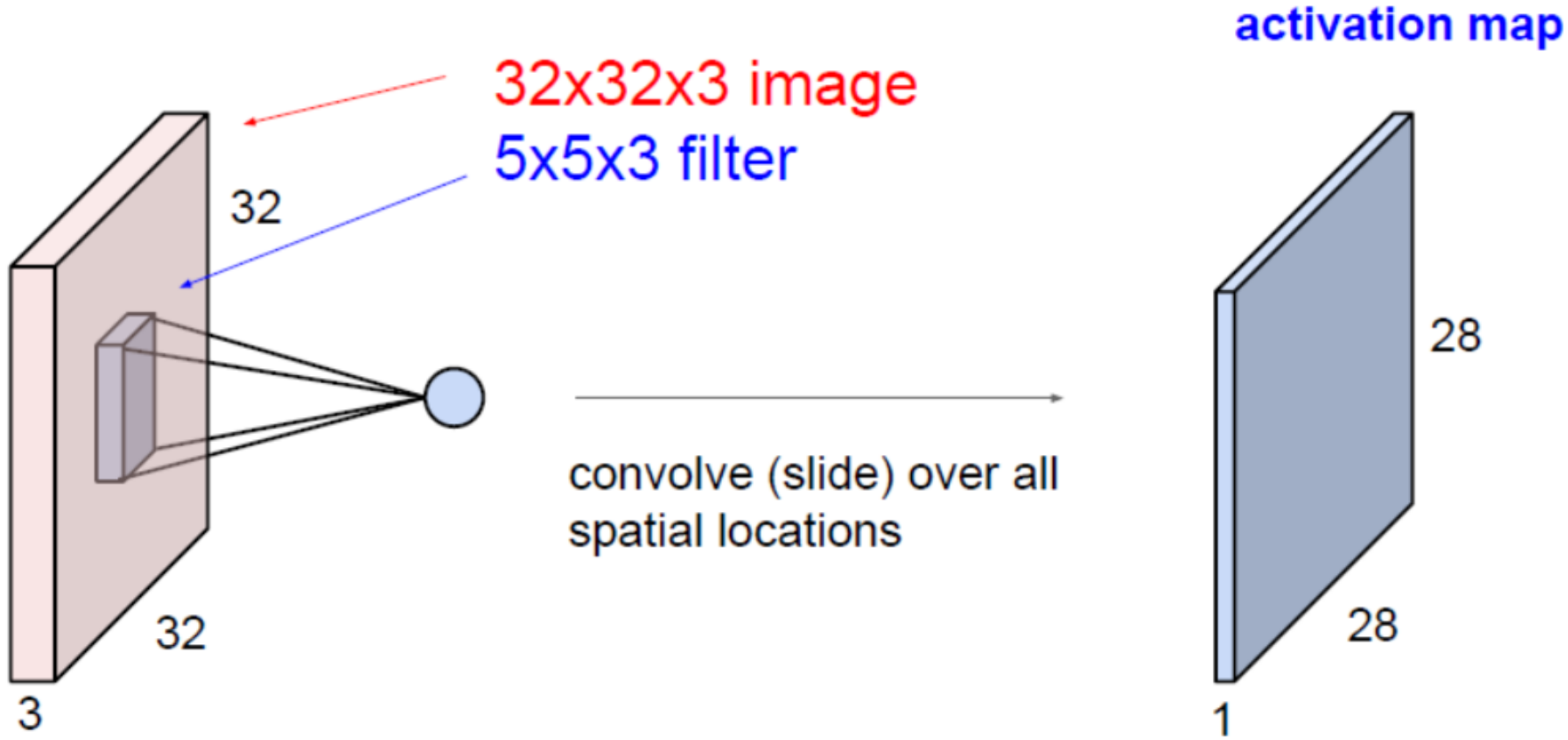
Convolution Layer



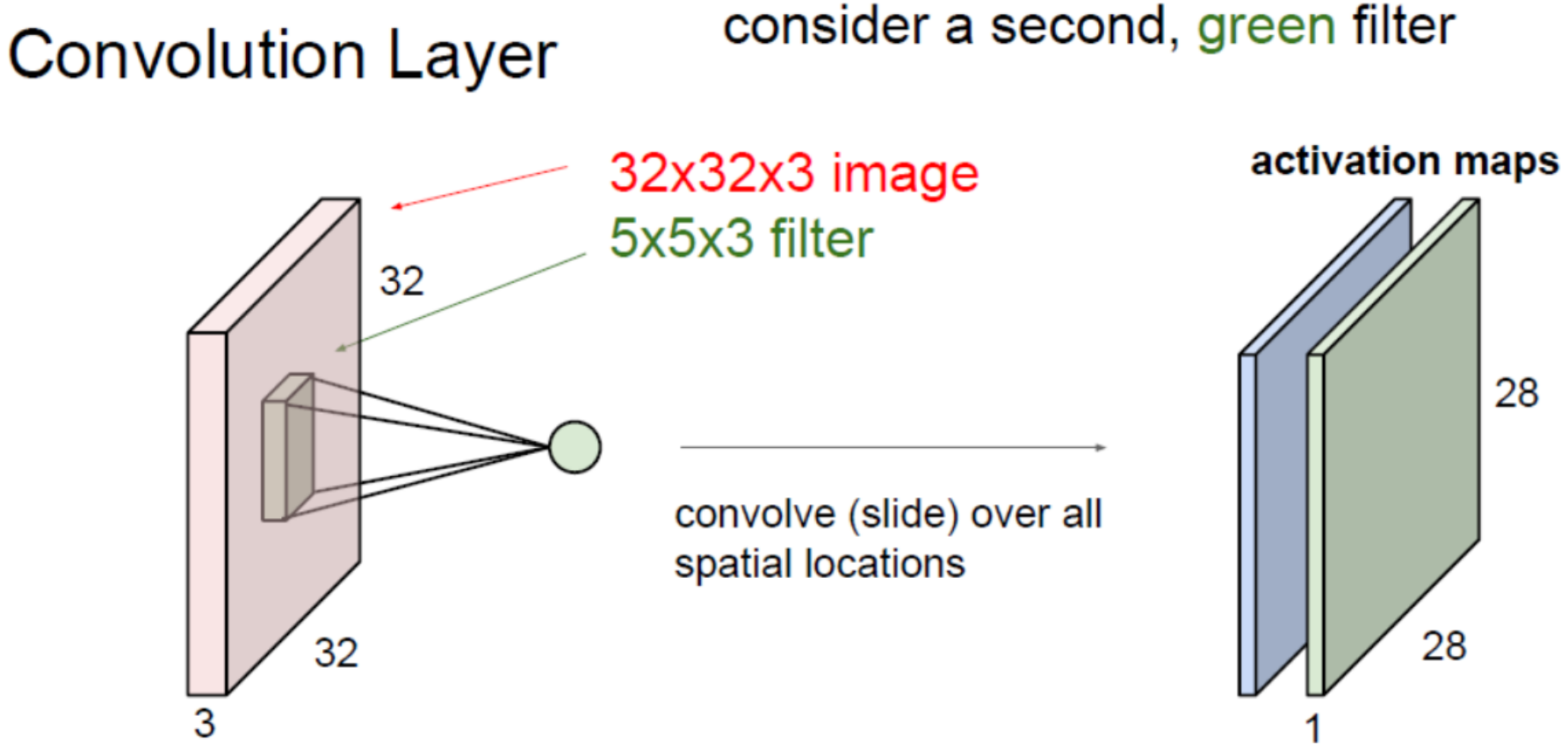
Convolution Layer



Convolution Layer

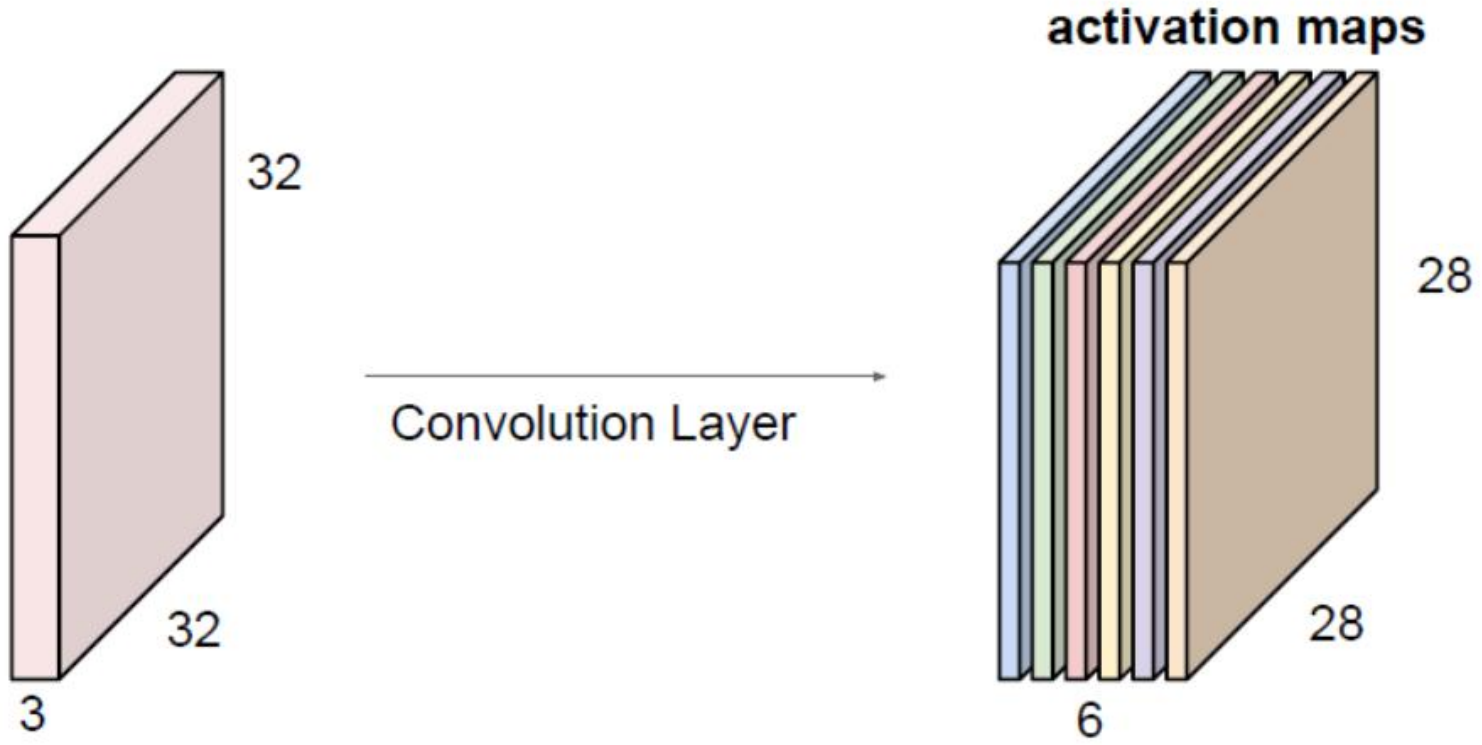


Convolution Layer



Convolution Layer

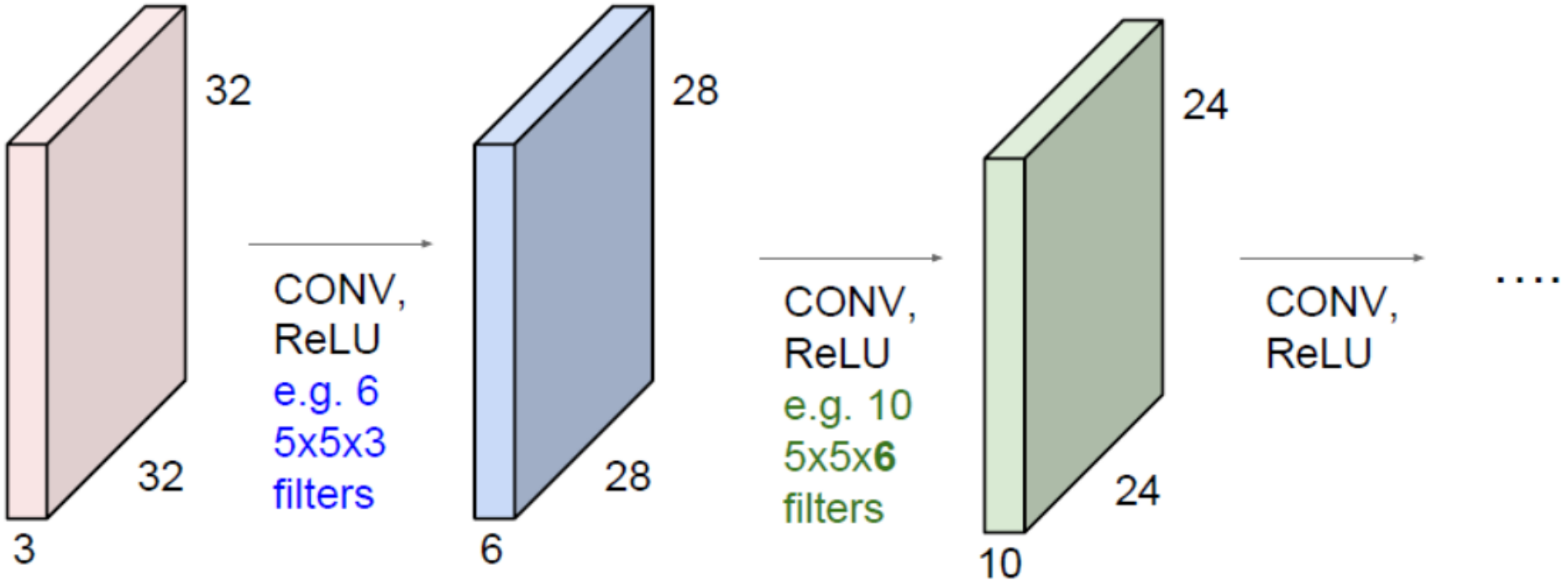
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



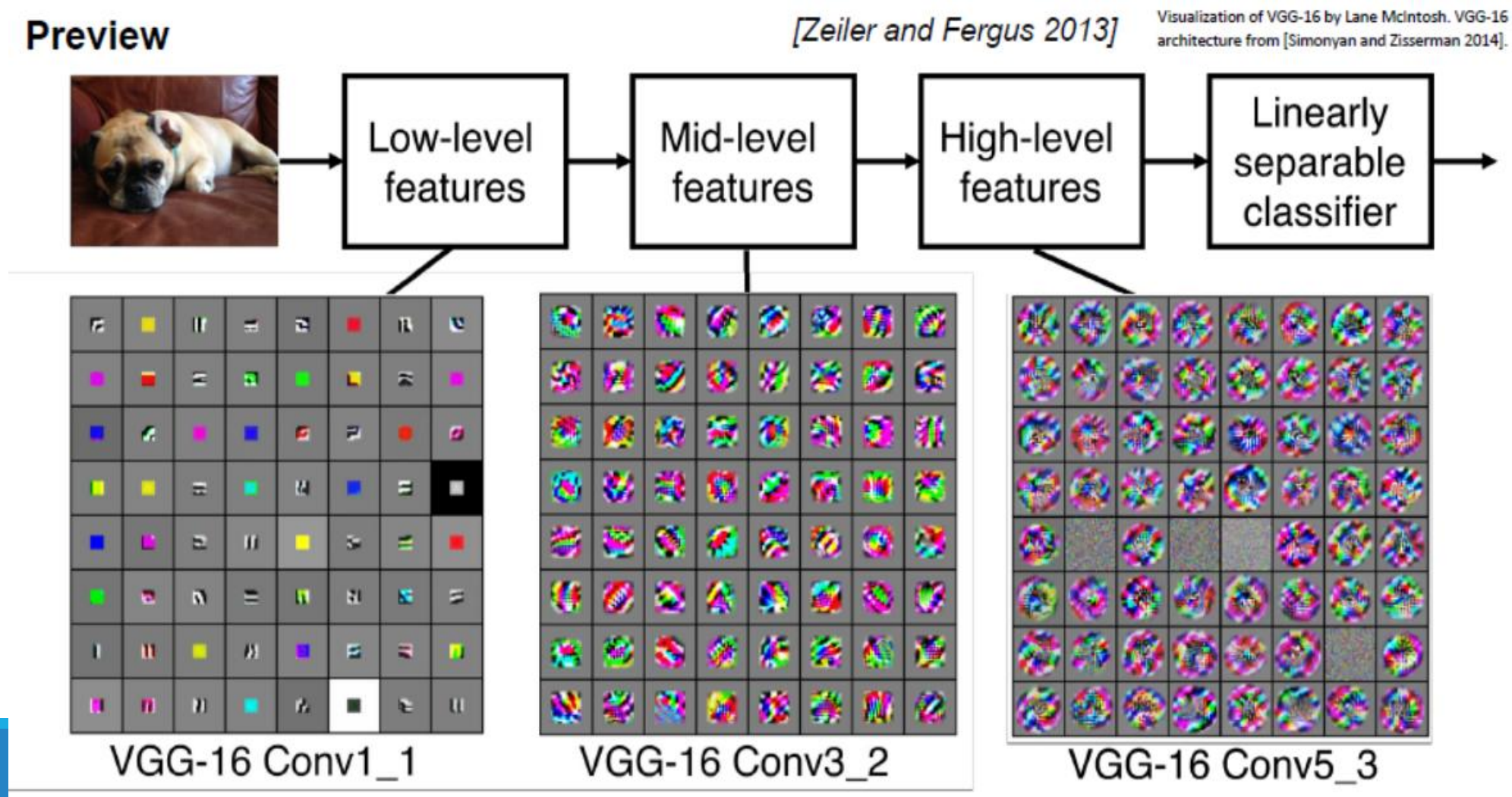
We stack these up to get a "new image" of size 28x28x6!

Convolution Layer

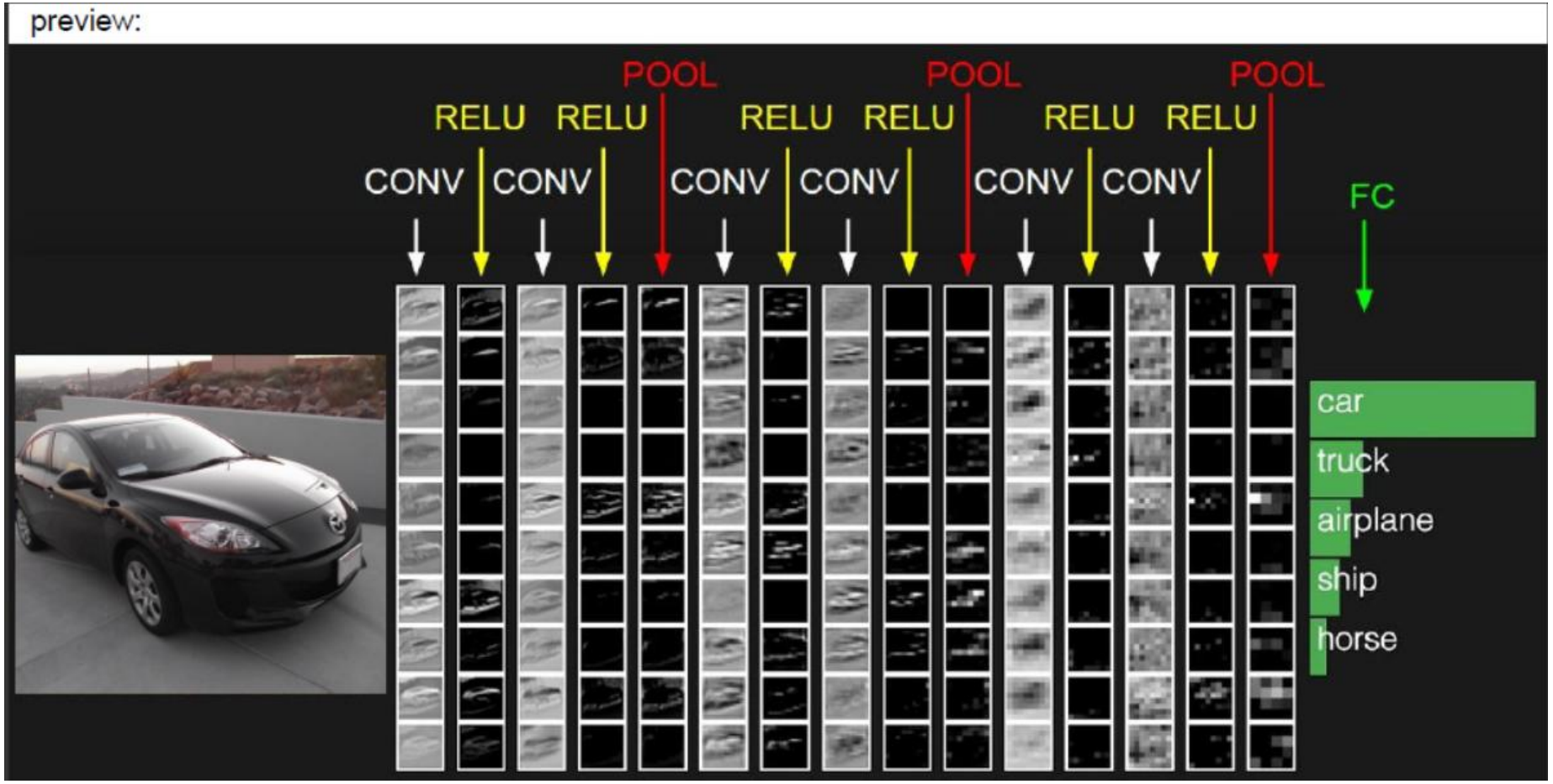
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Convolution Layer



Convolution Layer



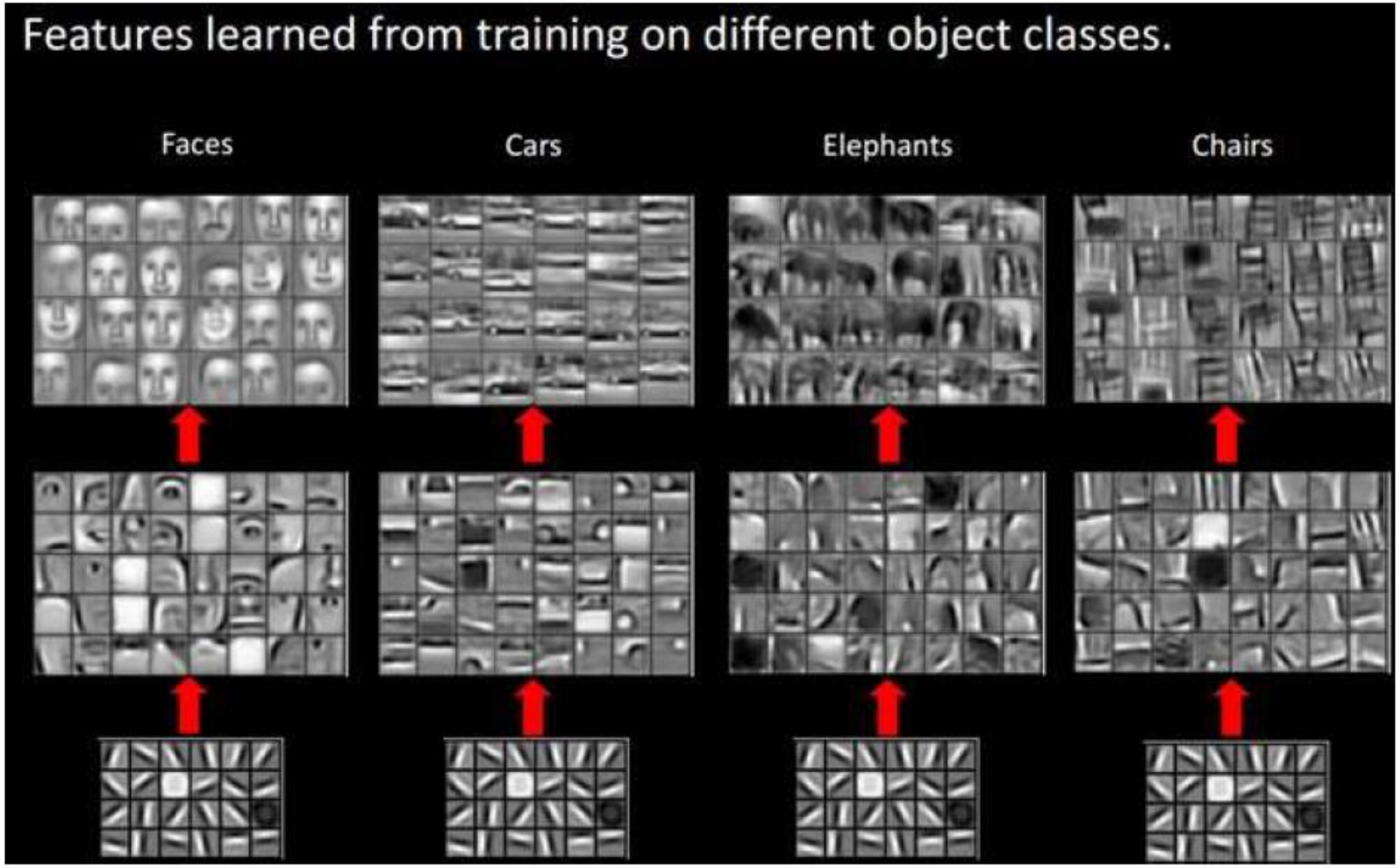
Deeper convolution layers

Initial convolution layers

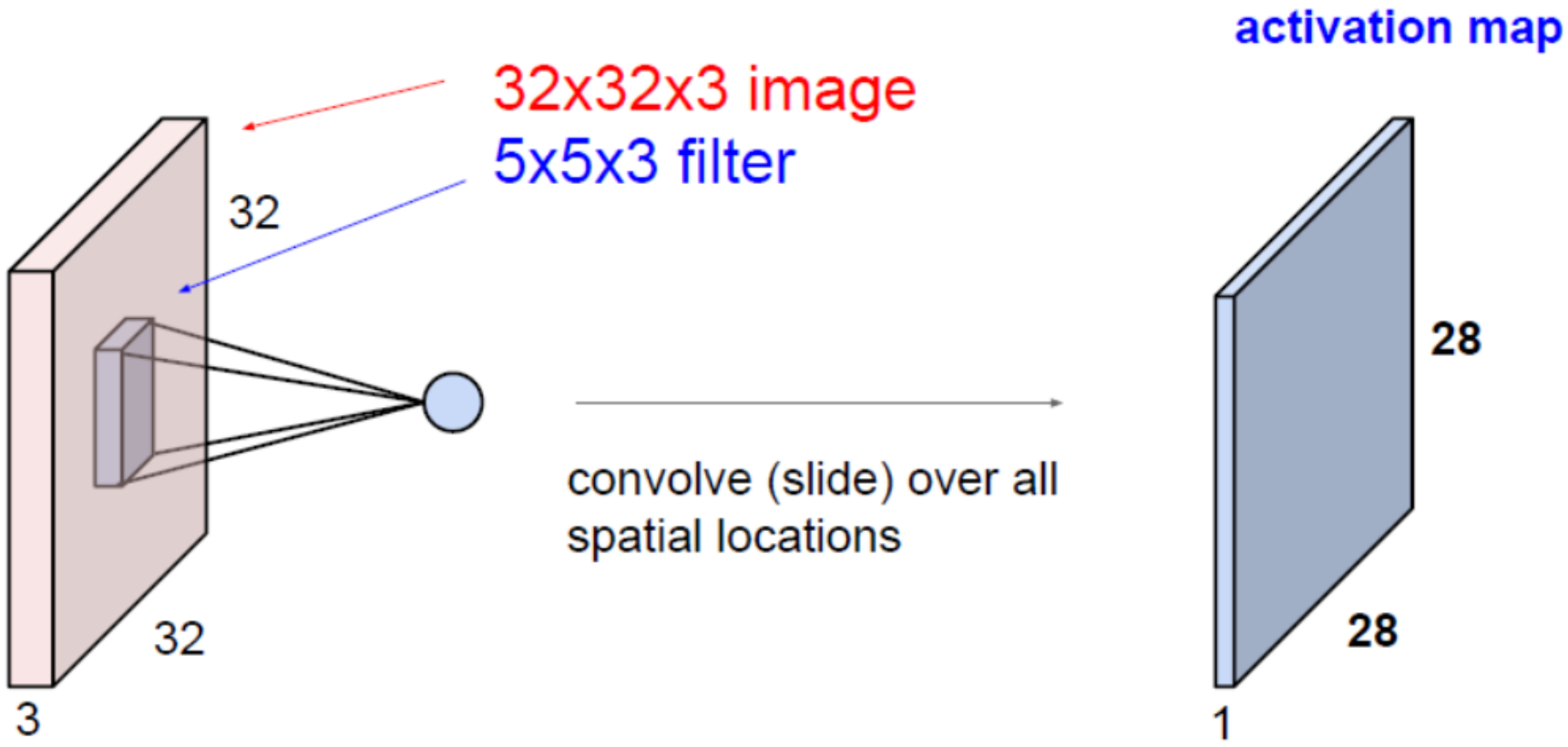
Higher-level features (e.g., objects)

Mid-level features (e.g., object parts, shapes)

Low-level features (e.g., edges)



A closer look at spatial dimensions:



Example

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

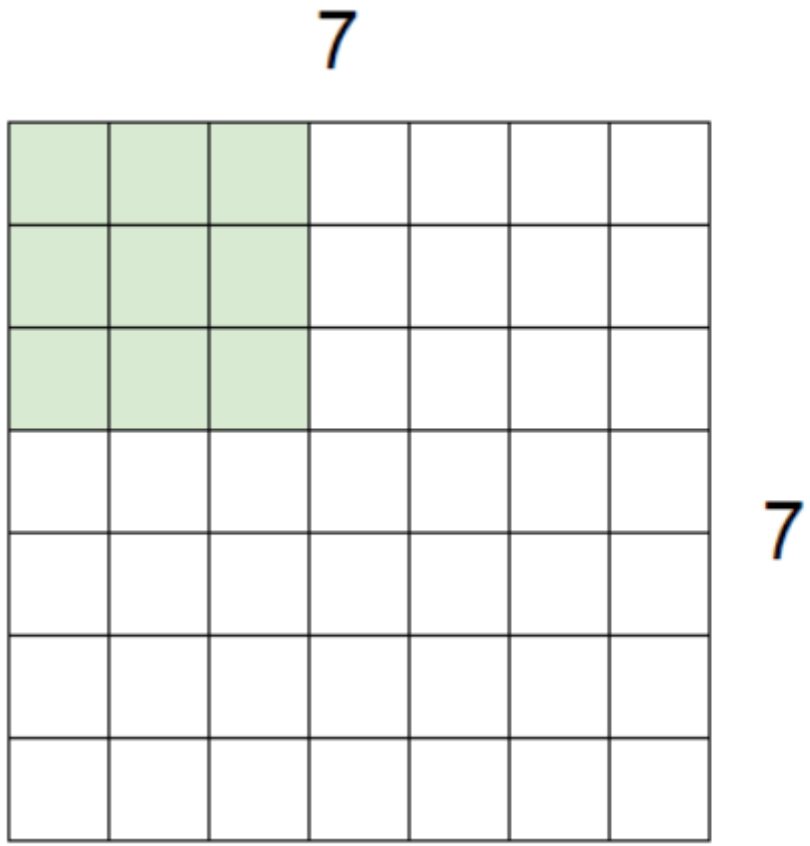
Do the same process for every filter

-1	-1	-1	-1
-1	-1	-2	1
-1	-1	-2	1
-1	0	-4	3

4 x 4 image

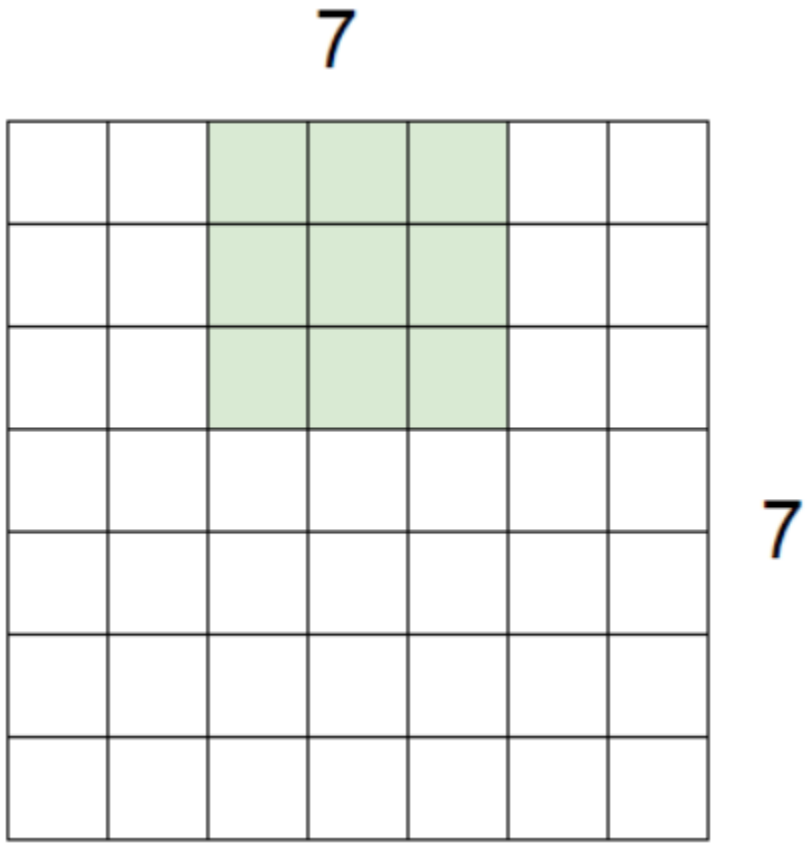
Feature/Activation Map

Convolution with stride 2



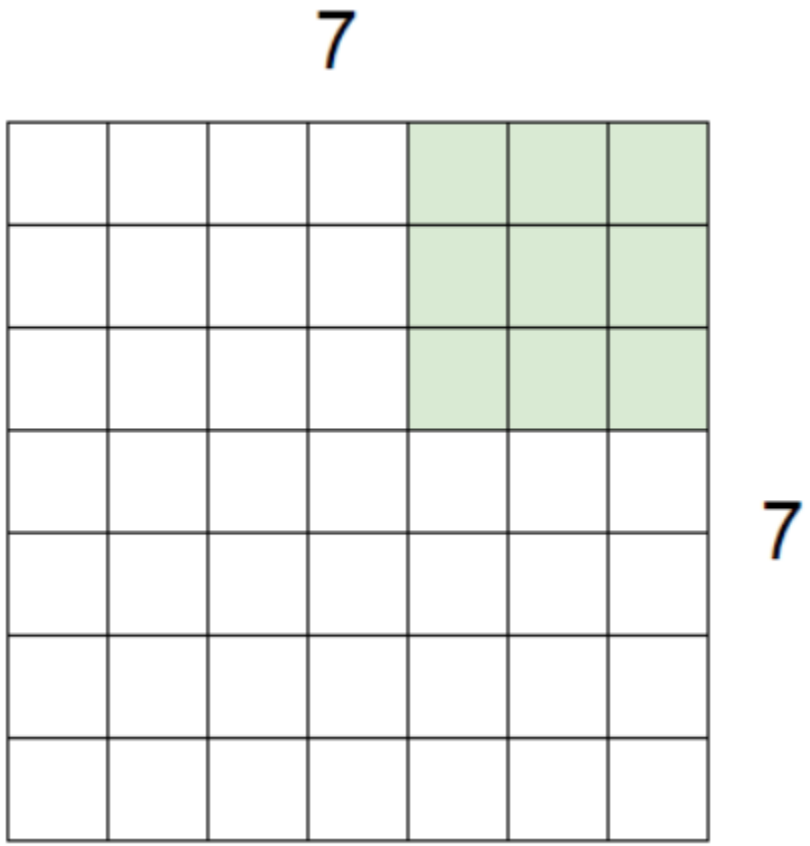
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolution with stride 2



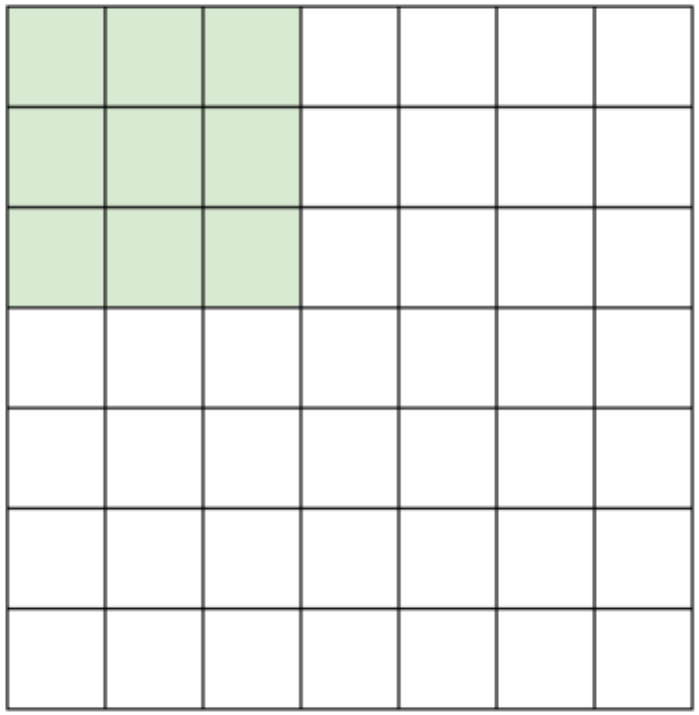
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolution with stride 2



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Convolution with stride 3



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

Convolution Stride – assuming no padding

$N \rightarrow$ Input size (height or width of the image)

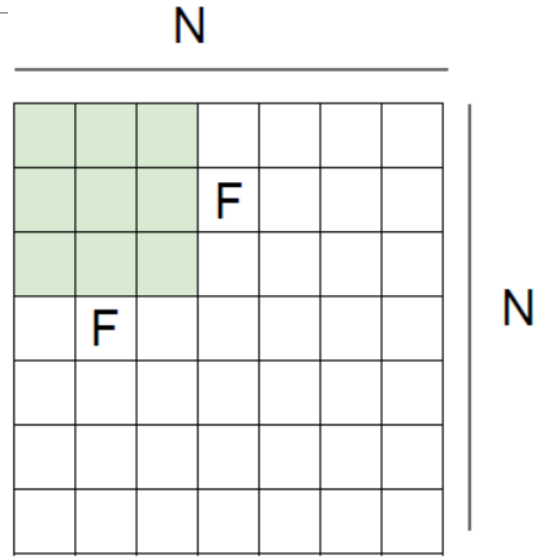
$F \rightarrow$ Filter (kernel) size

Stride → Number of pixels the filter moves (shifts) each time

Formula for Output Size

$$\text{Output } \textit{size} = \frac{(N - F)}{\textit{stride}} + 1$$

This tells how many times the filter fits (and slides) over the input image.



Stride = 1

$$(7 - 3)/1 + 1 = 5$$

So the output feature map will be **5×5**.

➞ The filter moves **one pixel at a time**, so there are **many overlaps** and **dense coverage**.

Convolution Stride – assuming no padding

Stride = 2

$$(7-3)/2+1=3$$

So, the output feature map will be **3×3**.

➞ The filter moves **two pixels at a time**, so fewer positions are covered, and **output shrinks**.

Stride = 3

$$(7-3)/3+1=2.33$$

Since the output size must be an **integer**, it's rounded down to **2**.

➞ This means the filter doesn't fit neatly — the last positions are skipped.

Convolution Stride – assuming no padding

Example comparison

Input size $N = 32$, Filter $F = 5$, Stride $S = 1$:

Without padding (P=0):

$(32 - 5)/1 + 1 = 28$ →Output 28×28

With padding (P=2):

$(32 + 2 \times 2 - 5)/1 + 1 = 32$ →Output 32×32

In practice: Common to zero pad the border

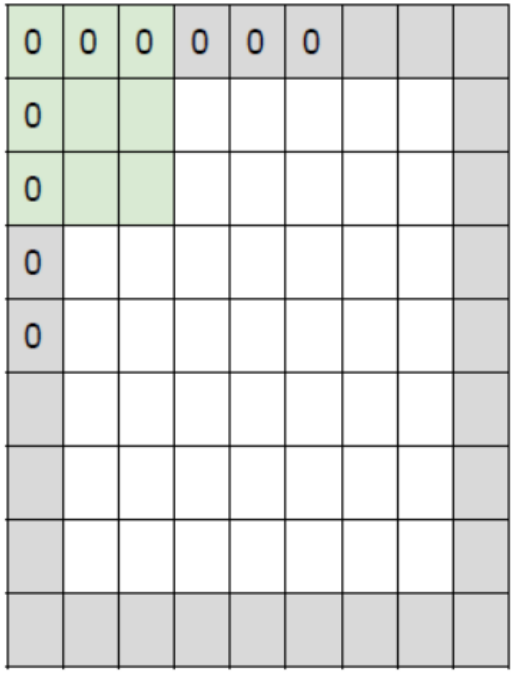
In practice: Common to zero pad the border

Zero padding means adding a border of zeros around the input image before applying convolution.

This is done to:

Preserve spatial size of the output (so it doesn't shrink).

Allow filters to see edge pixels — without losing border information.



Given Example

- **Input:** 7×7
- **Filter:** 3×3
- **Stride:** 1
- **Padding:** 1 pixel border (zeros added all around)

In practice: Common to zero pad the border

Compute Output Size

Use the **general formula** (which includes padding):

$$O = \frac{(N + 2P - F)}{stride} + 1$$

Substitute values:

$$O = \frac{(7 + 2 * 1 - 3)}{1} + 1 = (7 + 2 - 3) + 1 = 7$$

Output = 7×7

So, the output has the **same spatial size** as the input because padding preserved it.

In practice: Common to zero pad the border

Why pad with 1 pixel?
Padding of $(F-1)/2$is a rule of thumb when stride = 1:

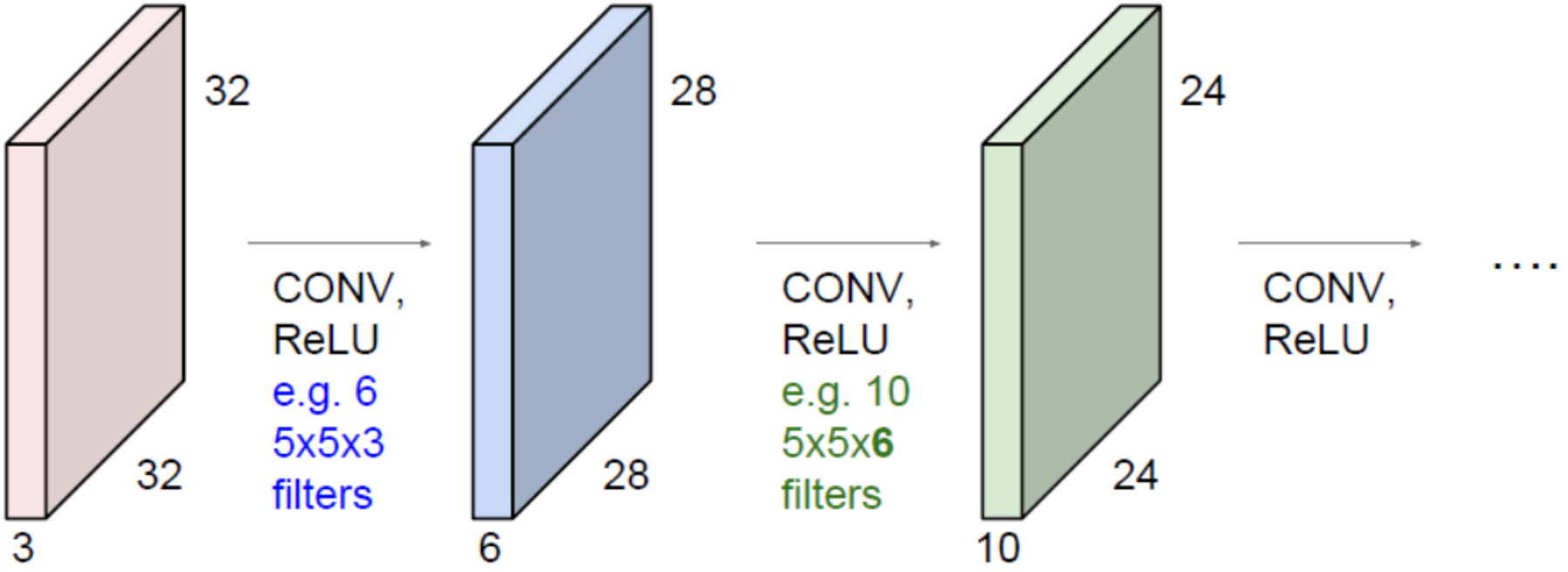
Filter size (F)	**Padding (P)**	**Output size**
3×3	1	same as input
5×5	2	same as input
7×7	3	same as input

Spatial Shrinking in Convolution Layers

Spatial Shrinking in Convolution Layers

Remember back to…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



Spatial Shrinking in Convolution Layers

Why "Shrinking too fast is not good"

If the feature map becomes too small early in the network:

- **Loss of spatial information** — the network can't detect fine details.
- **Less room for deeper layers** — limits learning capacity.
- **Inefficient learning** — fewer activation values to train on.

Hence, shrinking too fast "doesn't work well."

Output Size and Number of Parameters in CNN Layer

Output Size and Number of Parameters in CNN Layer

Given: Input volume: 32 × 32 × 3

32 × 32 → spatial size (height × width)

3 → number of channels (RGB)

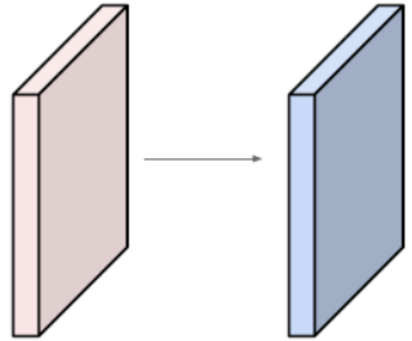
Filters: 10 filters of size 5 × 5

Stride (S): 1

Padding (P): 2

Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Output Size and Number of Parameters in CNN Layer

Output volume size

Formula for output spatial size: $O = \frac{(W+2P-F)}{S} + 1$

- W = input width/height
- P = padding
- F = filter size
- S = stride

Plugging values in: $O = \frac{(32+2*2-5)}{1} + 1 = 32$

So the **output height = output width = 32**.

There are **10 filters**, so the **depth = 10**.

Output volume = 32 × 32 × 10

Output Size and Number of Parameters in CNN Layer

Number of parameters

Each filter has:

$$5 \times 5 \times 3 = 75 \textit{ weights (since 3 input channels)}$$

plus **1 bias** term

→ total **76 parameters per filter**

Since there are **10 filters**:

$$76 \times 10 = 760 \textit{ total parameters}$$

LeNet-5 architecture - An example CNN

LeNet-5

- **LeNet-5** was one of the first CNNs used for **image classification**, specifically to recognize digits (0–9).
- It takes a **28×28 grayscale image** as input and outputs **one of 10 classes** using a **softmax** classifier.

Architecture Breakdown -

1. Input Layer

Input: 28×28×1 image (grayscale)

2. Convolution Layer 1 (Conv1)

Filter size: 5×5

Number of filters: 6

Stride: 1

Padding: 2 (to preserve spatial size)

Output size: 28×28×6

(Purpose: Detects low-level features like edges, corners, and simple textures.)

Architecture Breakdown -

3. Pooling Layer 1 (Pool1)

Type: Average Pooling (2×2, stride 2)

Output size: 14×14×6

(Purpose: Reduces spatial size (downsampling) to make computation faster and features more robust.)

4. Convolution Layer 2 (Conv2)

Filter size: 5×5

Number of filters: 16

Stride: 1

Output size: 10×10×16

(Purpose: Detects higher-level patterns and combinations of previous features.)

Architecture Breakdown -

5. Pooling Layer 2 (Pool2)

Type: Average Pooling (2×2, stride 2)

Output size: 5×5×16

(Purpose: Again reduces size and focuses on more abstract features.)

6. Fully Connected Layer 1 (FC1)

Input: 5×5×16 = 400 units

Output: 120 neurons

(Purpose: Connects the spatial features to dense layers for classification logic.)

Architecture Breakdown -

7. Fully Connected Layer 2 (FC2)

Input: 120 neurons

Output: 84 neurons

(Purpose: Further combines features before final decision.)

8. Output Layer (FC3 + Softmax)

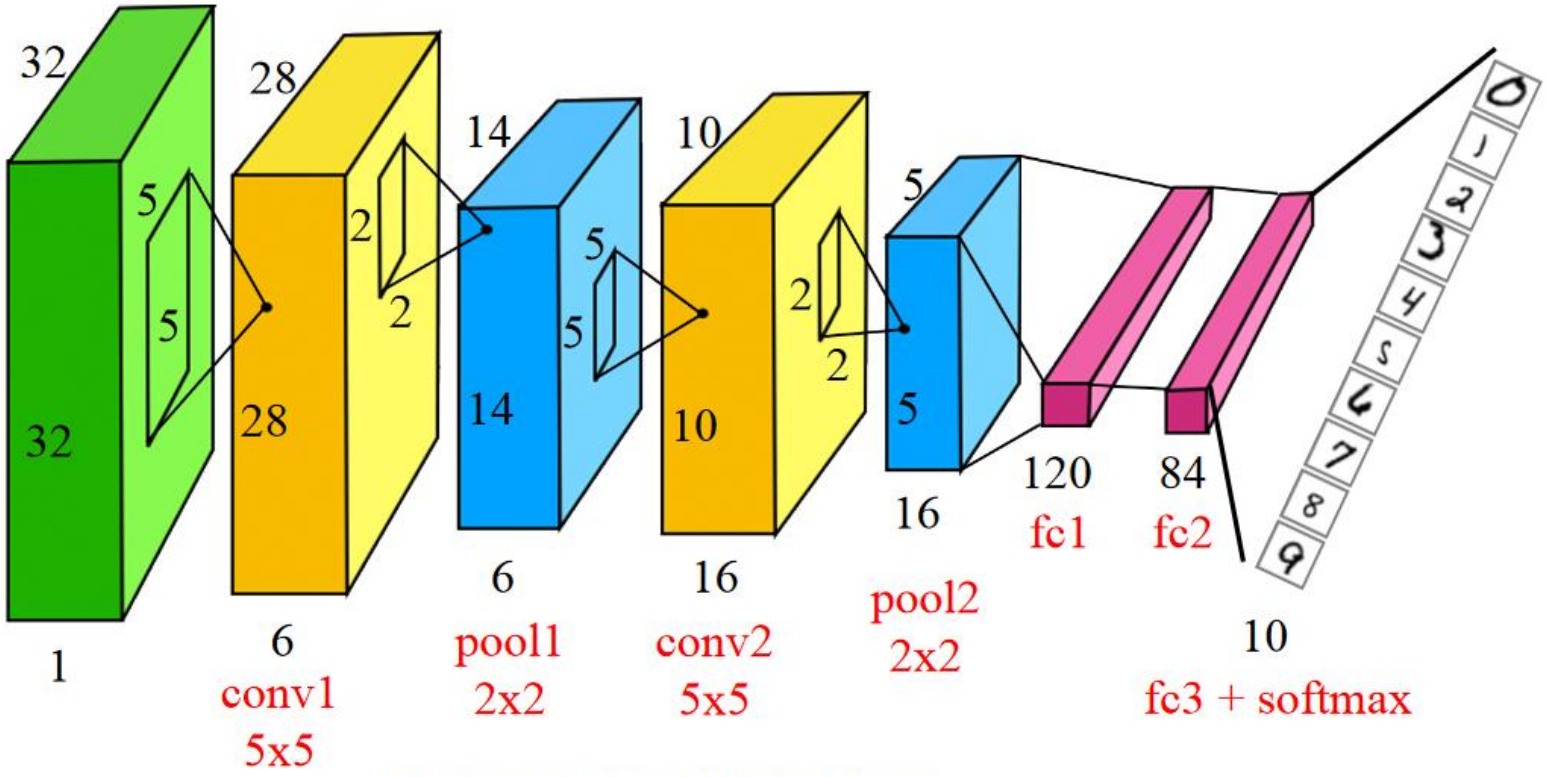
Input: 84 neurons

Output: 10 neurons (one for each digit 0–9)

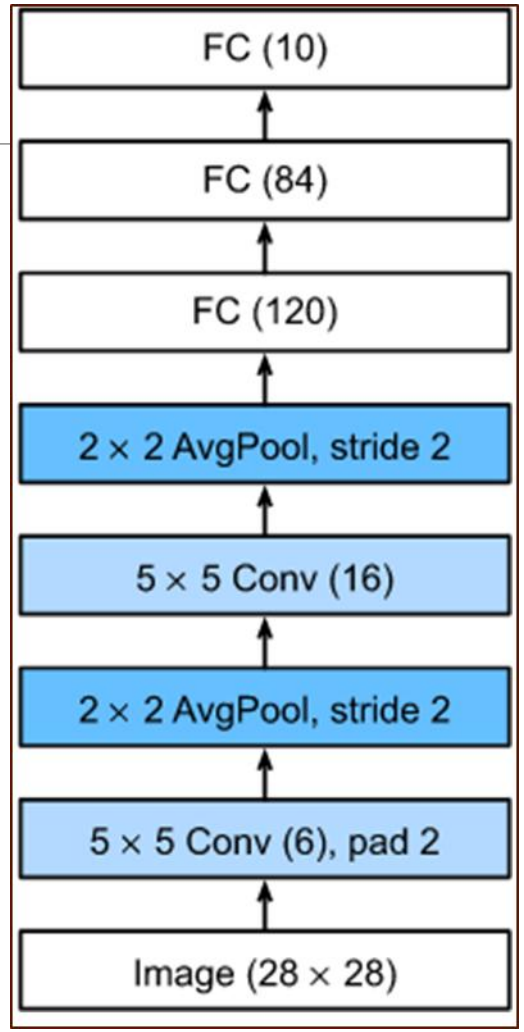
Activation: Softmax

(Purpose: Produces probability distribution across the 10-digit classes.)

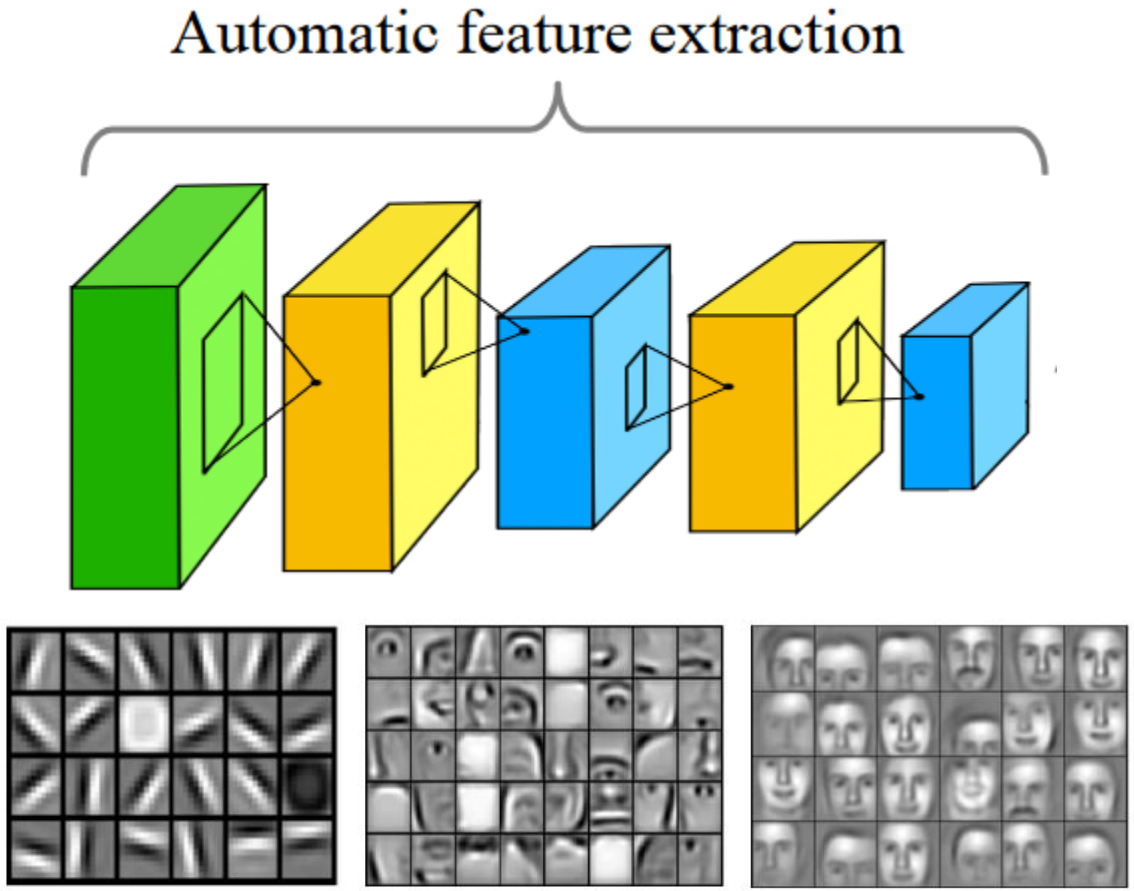
LeNet-5 architecture



http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf



Learning deep representations



http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf

Layer No.	Layer Type	Input Size	Filter / Kernel Size	# of Filters / Maps	Stride	Padding	Output Size	Activation	Purpose / Description
1	**Input**	32×32×1	—	—	—	—	32×32×1	—	Grayscale image (MNIST padded to 32×32)
2	**Conv1**	32×32×1	5×5	6	1	2	28×28×6	tanh	Detects basic edges & patterns
3	**Average Pooling (Subsampling)**	28×28×6	2×2	—	2	0	14×14×6	—	Reduces spatial resolution
4	**Conv2**	14×14×6	5×5	16	1	0	10×10×16	tanh	Extracts higher-level patterns
5	**Average Pooling**	10×10×16	2×2	—	2	0	5×5×16	—	Further reduces size, increases invariance
6	**Conv3 / FC (Flatten)**	5×5×16	—	—	—	—	400	—	Flattened feature vector
7	**Fully Connected (FC1)**	400	—	—	—	—	120	tanh	Learns abstract representations
8	**Fully Connected (FC2)**	120	—	—	—	—	84	tanh	Further abstraction
9	**Output Layer (FC3)**	84	—	—	—	—	10	softmax	Classifies digits 0–9

Flow Summary (shape evolution):

$$32 \times 32 \times 1 \rightarrow 28 \times 28 \times 6 \rightarrow 14 \times 14 \times 6 \rightarrow 10 \times 10 \times 16 \rightarrow 5 \times 5 \times 16 \rightarrow 400 \rightarrow 120 \rightarrow 84 \rightarrow 10$$

The **Conv + Pooling** pattern gradually reduces the spatial dimension but increases feature depth.

The **last three layers (FC1, FC2, FC3)** act like a **traditional neural network** for classification.

tanh activations were used in the original LeNet-5 (modern CNNs often use ReLU).

Padding is used initially to preserve dimensions (from 32×32 to 28×28).

Assignment - 55

Implement LeNet-5

Thank you