

# Deep Learning

# Deep Learning

- Subset of machine learning
- Neural networks with three or more layers
- Imitates how humans process data and make decisions
- Exponential growth in the last few years
- Powered by advances in large-scale data processing and inference

# Applications of Deep Learning

- Natural language processing (NLP)
- Speech recognition and synthesis
- Image recognition
- Self-driving cars
- Customer experience, healthcare, and robotics

# **Scope and Prerequisites**

# Deep Learning: Background

- Vast domain with a variety of tools and technologies
- Multiple courses exist for concepts and implementations
- Math requirements and coverage depth
- Tools—understanding and depth

# Scope of the Course

- Educate students about the basic concepts of deep learning
- Focus on what happens behind-the-scenes
- Use Keras for implementation—mask complexity
- Minimal math coverage, deeper topics omitted
- Simple examples to get started with deep learning

# Course Prerequisites

- Machine learning concepts and technologies
- Python programming, notebooks
- Keras and TensorFlow
- scikit-learn, text processing with NLTK

# Deep Learning

- Subset of machine learning
- Neural networks with three or more layers
- Imitates how humans process data and make decisions
- Exponential growth in the last few years
- Powered by advances in large-scale data processing and inference

# Applications of Deep Learning

- Natural language processing (NLP)
- Speech recognition and synthesis
- Image recognition
- Self-driving cars
- Customer experience, healthcare, and robotics

# Linear Regression

# Linear Regression

- A linear model
- Relationship between two or more variables
- Predict dependent variable based on independent variable
- Slope and intercept models the relationship

y	Dependent Variable
x	Independent Variable
a	Slope
b	Intercept

$$y = ax + b$$
$$y = a_1x_1 + a_2x_2 \dots + a_nx_n + b$$

# Building a Linear Regression Model

- Find values for slope and intercept
- Use known values of x and y (multiple samples)
- Multiple independent variables make it complex

$$5 = 2a + b, 9 = 4a + b$$

$$b = 5 - 2a$$

$$9 = 4a + 5 - 2a$$

$$a = (9-5)/2 = 2$$

$$b = 5 - 2*2 = 1$$

# Logistic Regression

- A binary model
- Relationship between two or more variables
- Output is 0 or 1

y	Dependent Variable
x	Independent Variable
a	Slope
b	Intercept
f	Activation Function

$$y = f(ax + b)$$

$$y = f(a_1x_1 + a_2x_2 \dots + a_nx_n + b)$$

# Deep Learning Process

- Deep learning is a complex and iterative process
- Starts with random initialization and works towards the right values by trial and error

$$10 = 3a + b$$

Trial	a	b	$3a + b$	Error
1	1	1	4	6
2	4	3	15	-5
3	2	2	8	2
4	3	2	11	-1
5	2	3	9	1
6	2	4	10	0



# What Is a Perceptron?

- An algorithm for supervised learning for binary classification
- Resembles a cell in the human brain
- A single cell neural network
- Based on logistic regression

# Perceptron Formula

$$y = f(a_1x_1 + a_2x_2 \dots + a_nx_n + b)$$



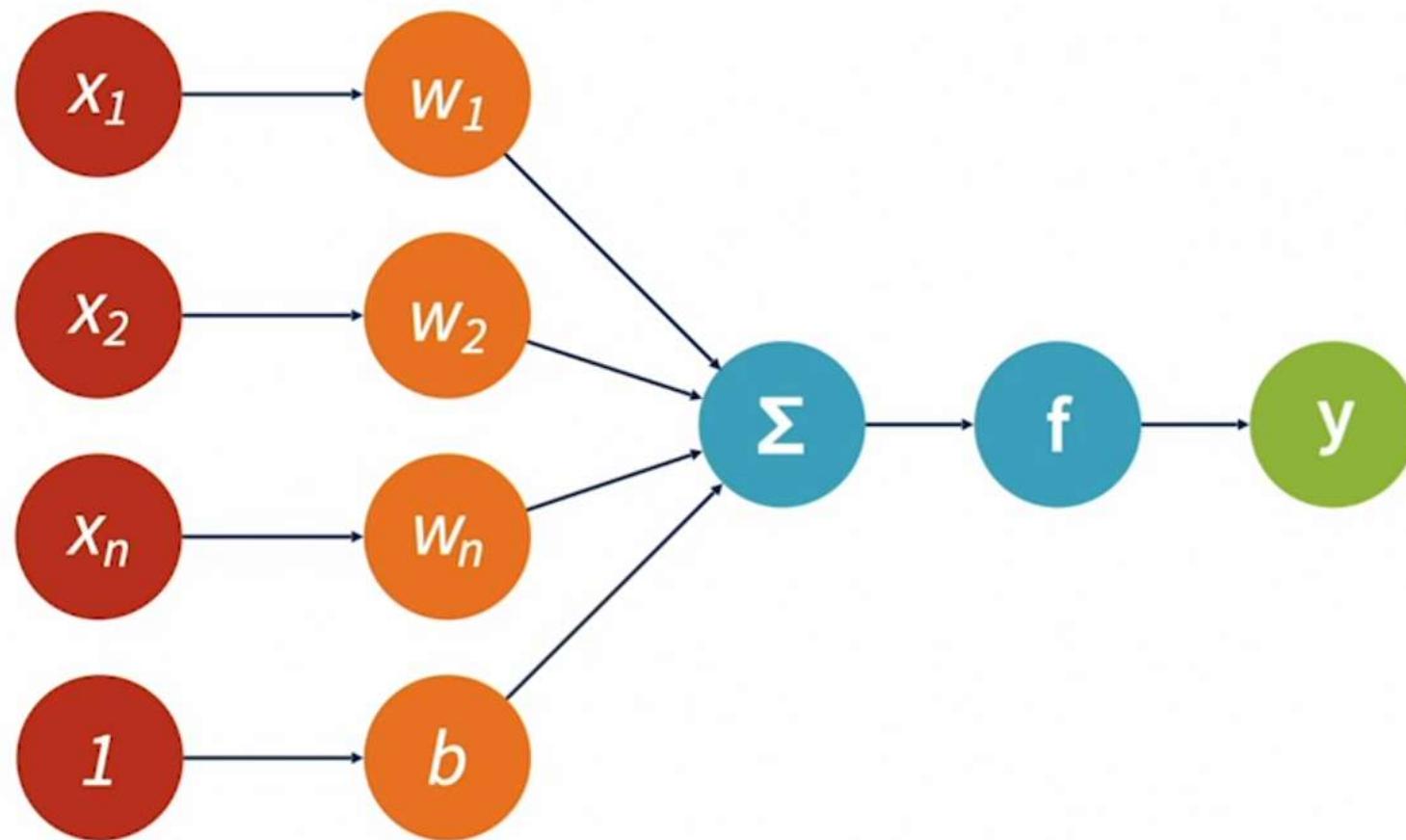
$$y = f(w_1x_1 + w_2x_2 \dots + w_nx_n + b)$$

w	Weight
b	Bias
f	Activation Function

## Activation Function Example

1 if value > 0  
0 otherwise

# Perceptron



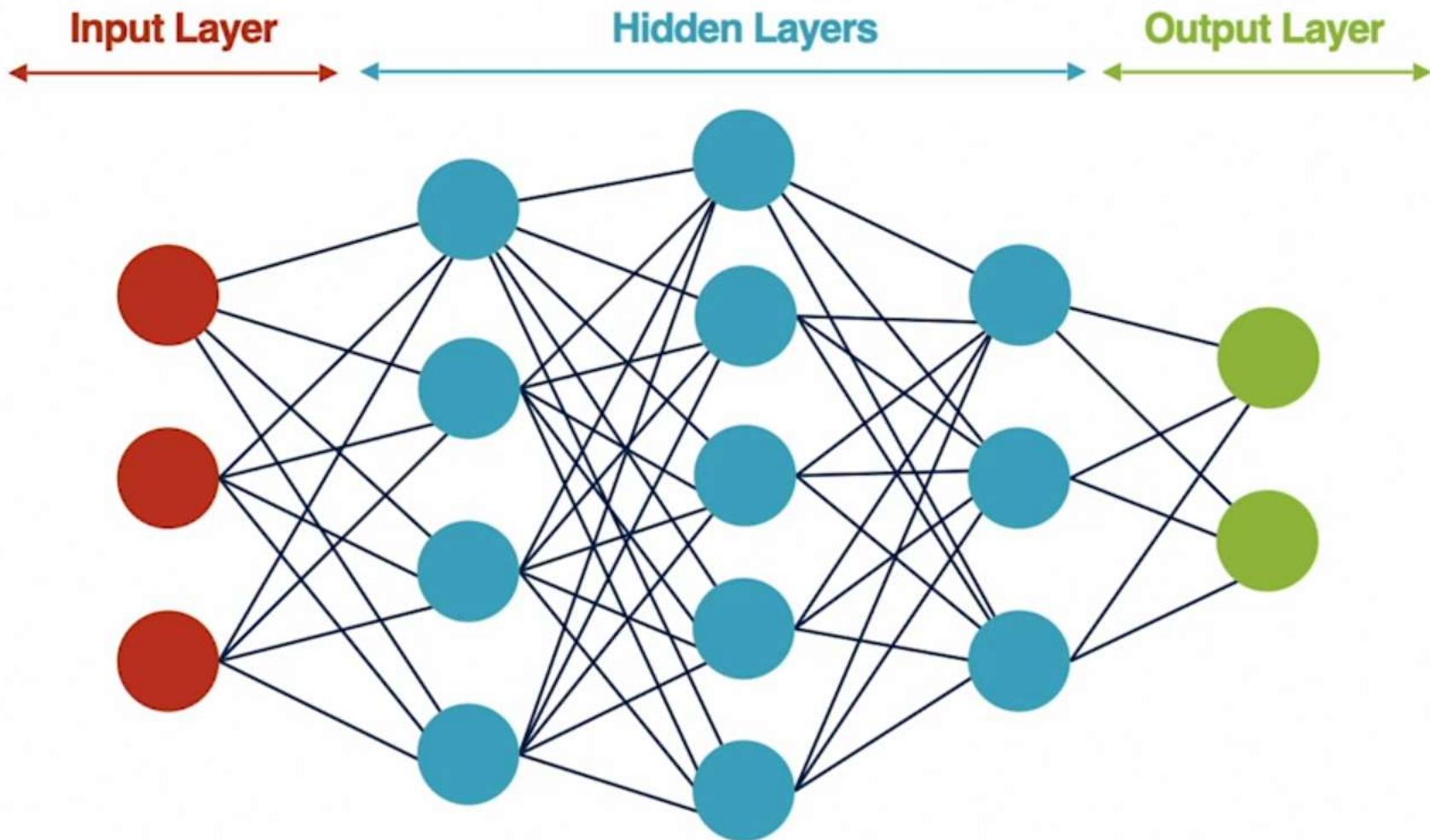
# **Artificial Neural Network (ANN)**

A network of perceptrons, modeled after the  
human brain

# What Is an Artificial Neural Network?

- Perceptrons are called *nodes* in the neural network
- Nodes organized as layers
- Each node has weights, biases, and activation functions
- Each node is connected to all nodes in the next layer

# Artificial Neural Network



# How ANN Works

- Inputs (independent variables) are sent from the input layer
- Inputs passed on to the nodes in the next hidden layer
- Each node computes its output based on its weights, biases, and activation functions
- Node output is then passed on as inputs to the next layer

# **Training an ANN**

# What Does Training Mean?

A model is represented by parameters and hyperparameters.

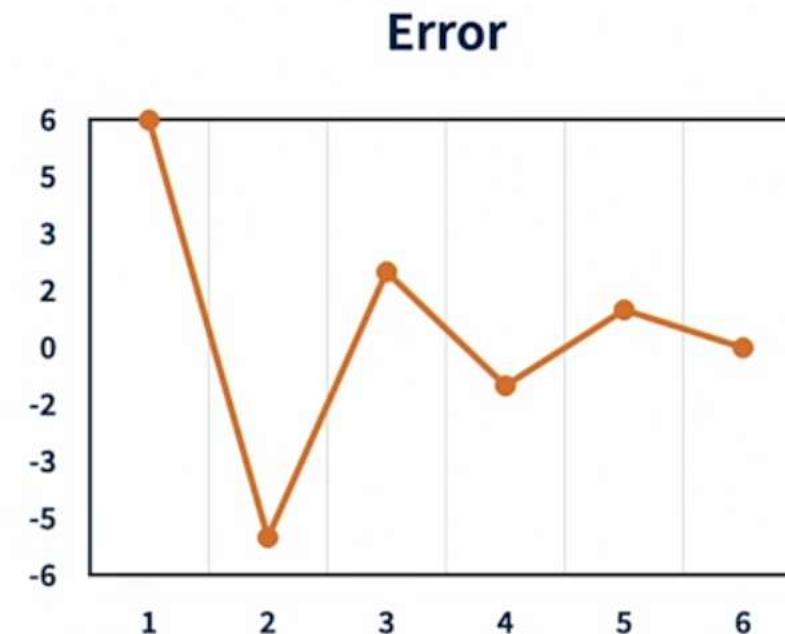
- Weights, biases, nodes, layers, layers, etc.

Training a model means determining the best possible values for the parameters and hyperparameters that maximize accuracy.

Inputs, weights, and biases might be n-dimensional arrays.

# Recall the Linear Regression Analogy

Trial	a	b	$3a + b$	Error
1	1	1	4	6
2	4	3	15	-5
3	2	2	8	2
4	3	2	11	-1
5	2	3	9	1
6	2	4	10	0



# Training Process

- Use training data (known values of inputs and outputs)
- Create network architecture with intuition
- Start with random values for weights and biases
- Minimize error in predicting known outputs from inputs

# Training Process

- Adjust weights and biases until error is minimized
- Improve model by adjusting layers, node counts, and other hyper parameters

# The Input Layer

# Vectors

The input to deep learning is a vector of numeric values

A vector is a tuple of one or more values

- E.g., ( 1.0, 2.1, 3.5)

Defined usually as a NumPy array

Represents the feature variables for prediction

# Samples and Features

The diagram illustrates the concept of samples and features in machine learning. It shows a table of employee data with four columns: Employee ID, Age, Salary, and Service. The first row represents the feature names, while the subsequent three rows represent individual employees (samples). Arrows point from the column headers to the 'Features' label, and arrows point from the data rows to the 'Samples' label.

Employee ID	Age	Salary	Service
1001	28	40,000	5
1002	47	60,000	20
1003	35	55,000	10

**Features**

**Samples**

# Input Preprocessing

Features need to be converted to numeric representations.

Input Type	Preprocessing Needed
Numeric	Centering and scaling
Categorical	Integer encoding, one-hot encoding
Text	TF-IDF, embeddings
Image	Pixels – RGB representation
Speech	Time series of numbers

# Samples to Input Vectors

Raw Data

Age	Salary	Service
28	40,000	5
47	60,000	20
35	55,000	10

Centered and Scaled

x1	x2	x3
-1.10	-1.37	-1.07
1.32	0.98	1.34
-0.21	0.39	-0.27

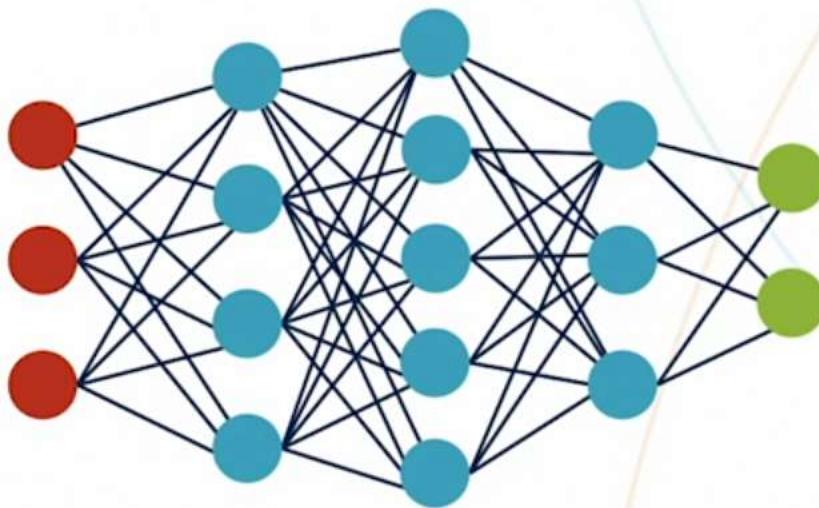
Transposed

x1	-1.1	1.32	-0.21
x2	-1.37	0.98	0.39
x3	-1.07	1.34	-0.27

# **Hidden Layers**

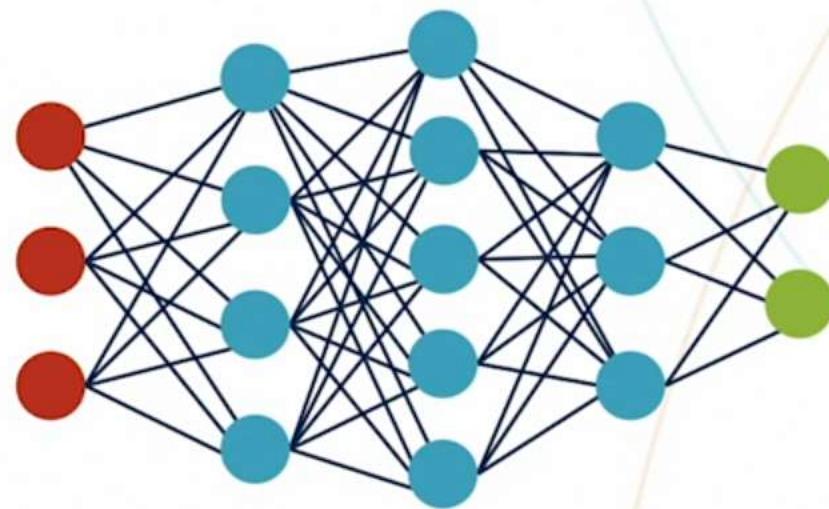
# Hidden Layers

- An ANN can have one or more hidden layers
- Each hidden layer can have one or more nodes (or perceptron) (count in  $2n$ )
- A neural network architecture is defined by the number of layers and nodes



# Inputs and Outputs

- The output of each node in the previous layer becomes the input for each node in the current layer (fully connected)
- Each node produces one output that is forwarded to the next layer



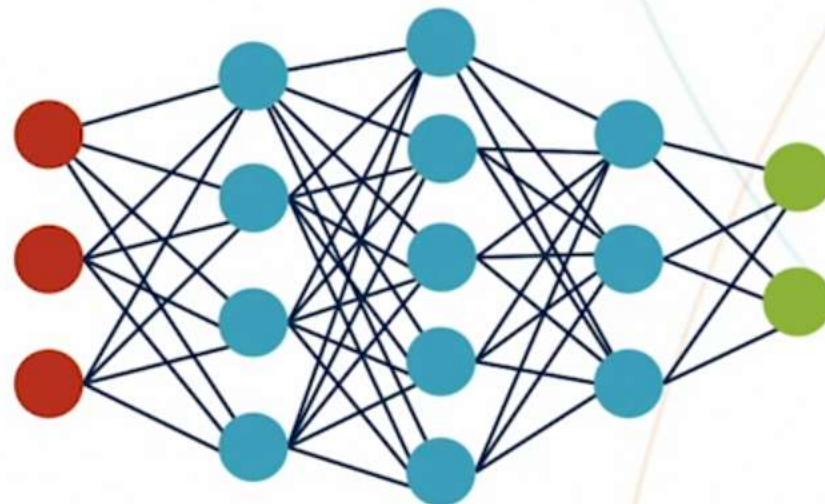
# Determining Hidden Layer Architecture

Each node *learns* something about the feature-target relationship

More nodes and layers mean

- Web applications on AWS
- Analytics on GCP

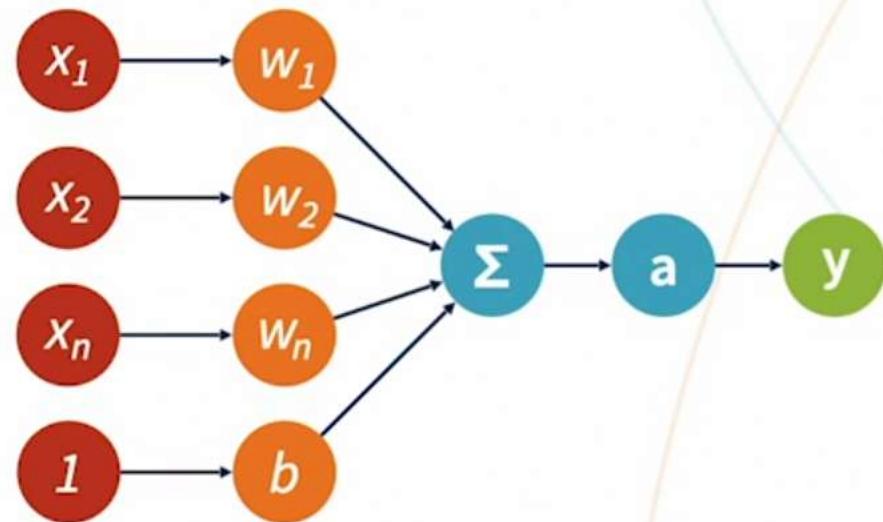
Architecture decided by experimentation



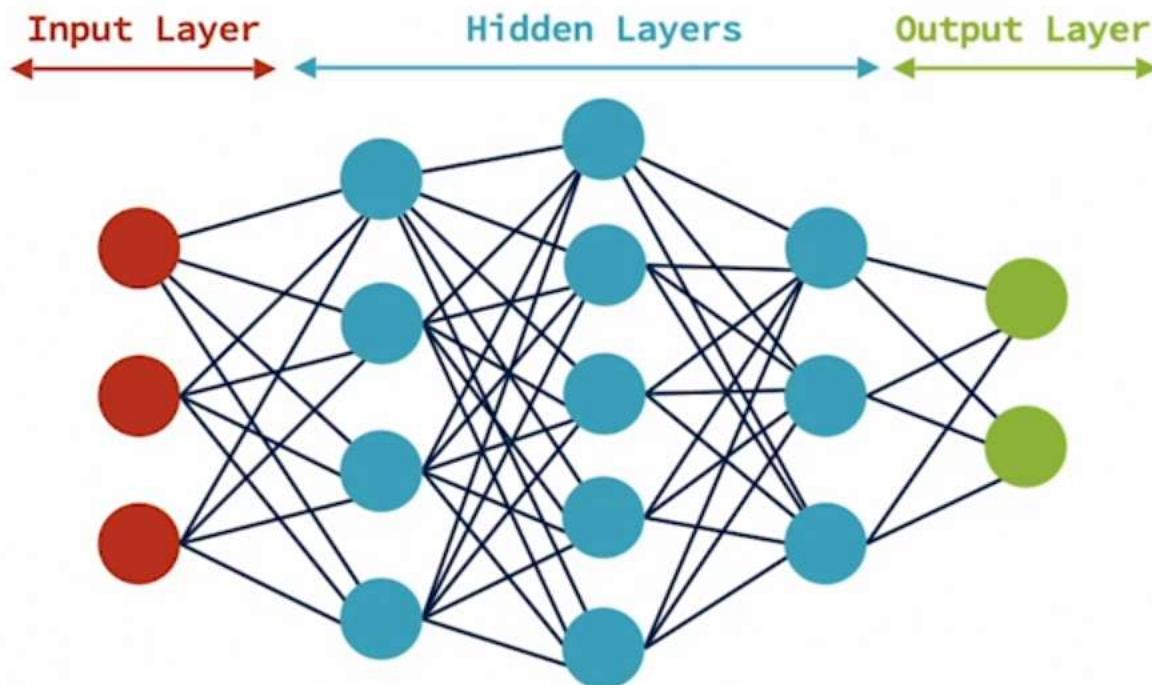
# **Weights and Biases**

# Weights and Biases Overview

- Weights and biases represent the trainable parameters in an ANN
- Numeric values
- Each input for each node has a weight associated with it
- Each node has a bias associated with it

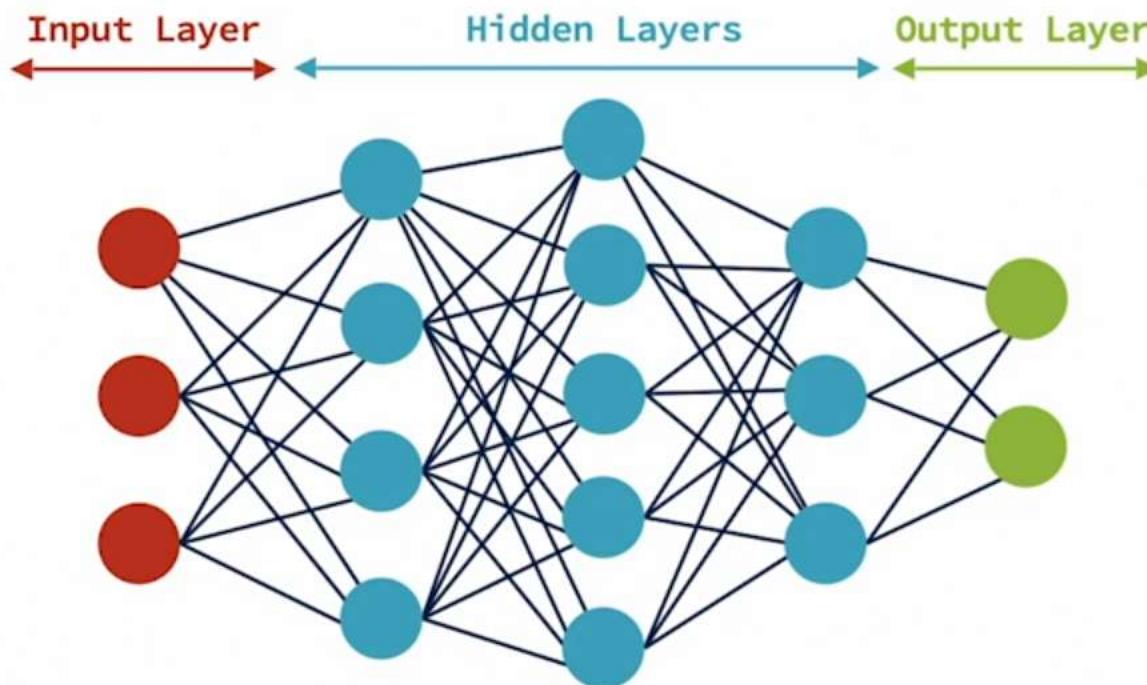


# Weights and Biases for a Layer



Layer	Inputs	Nodes	Weights	Biases
HL 1	3	4	12	4
HL 2	4	5	20	5
HL 3	5	3	15	3
Output	3	2	6	2
<b>Total</b>			<b>53</b>	<b>14</b>

# Computing Outputs: Hidden Layer 2



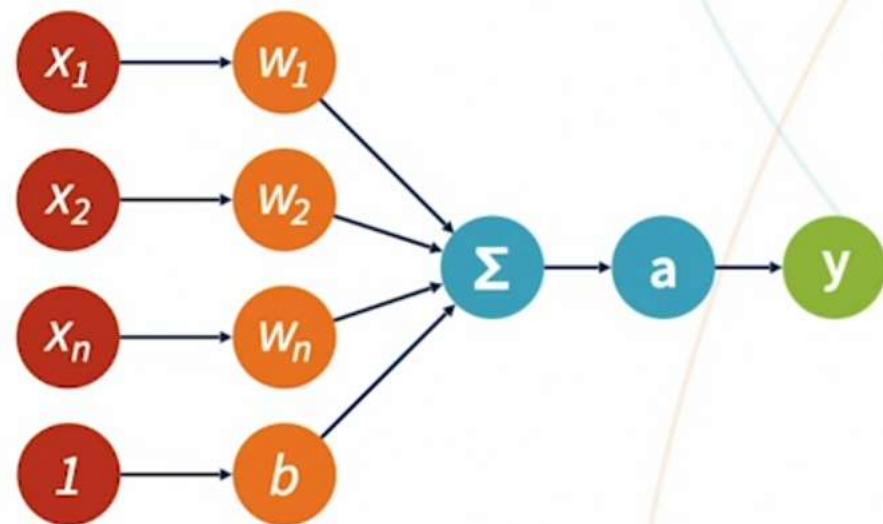
$$\begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{34} & w_{43} \\ w_{14} & w_{24} & w_{34} & w_{44} \\ w_{15} & w_{25} & w_{35} & w_{45} \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}$$

$$W^T X + B = Y$$

# Activation Functions

# Activation Functions

- Determines which nodes propagate information to the next layer
- Filters and normalizes
- Converts output to nonlinear
- Critical in learning patterns



# Popular Activation Functions

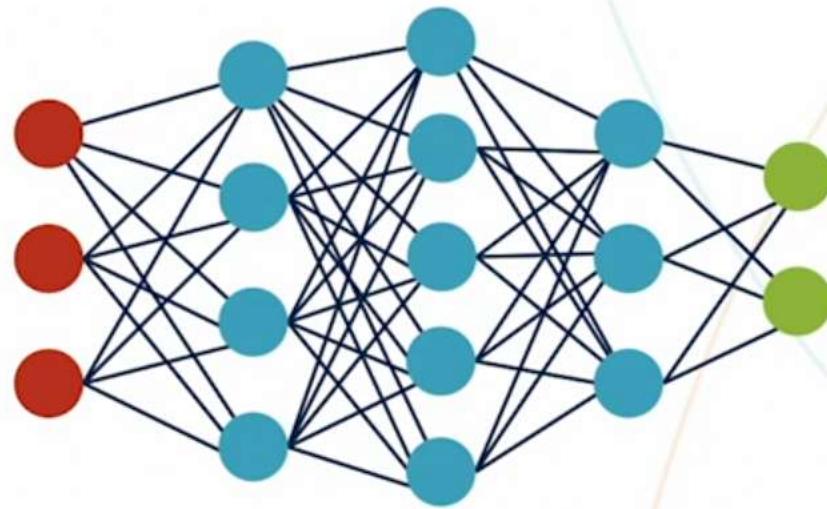
Activation Function	Output
Sigmoid	0 to 1
Tanh	-1 to +1
Rectified Linear Unit (ReLU)	0 if $x < 0$ ; $x$ otherwise
Softmax	Vector of probabilities, with sum=1

Choice depends on problem and experimentation.

# The Output Layer

# Output Layer

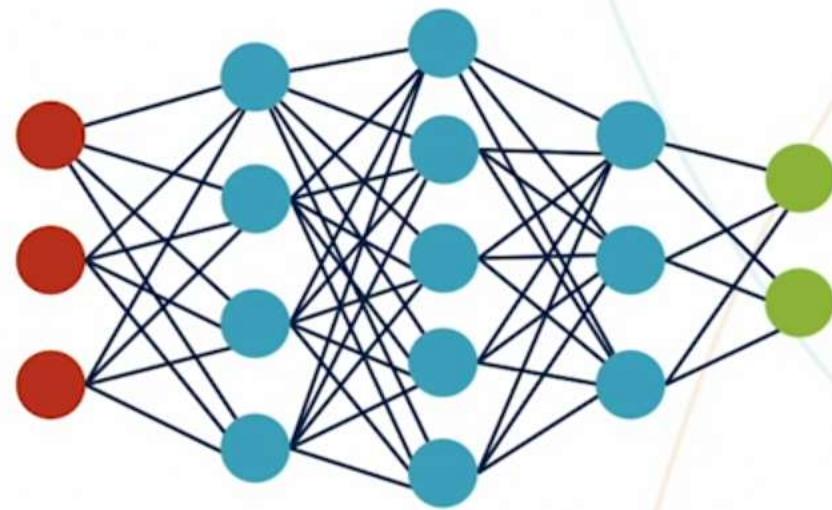
- One layer of output, produces desired Y
- Has its own weights and biases
- Softmax activation used for classification problems
- May need postprocessing to convert to business values



# Output Layer Size

Size depends on the problem

- 1 for binary classification
- $n$  for a  $n$ -class classification
- 1 for regression problems
- Vary based on other problem domains



# **Setup and Initialization**

# Input Preprocessing

	Sample 1	Sample 2	Sample 3	Sample 4
Feature 1	x11	x21	x31	x41
Feature 2	x12	x22	x32	x42
Feature 3	x13	x23	x34	x43
Feature 4	x14	x24	x34	x44
Feature 5	x15	x25	x35	x45
Target	y1	y2	y3	y4

# Splitting Input

- Training set: Used to fit the parameters
- Validation set: Used for model selection/tuning
- Test set: Used to measure the final model performance
- Usual split: 80:10:10

# **Model Architecture and Hyper Parameters**

## **Select values for the model**

- Layers and nodes in the layer, activation functions
- Hyper parameters

## **Selection criteria**

- Initial selection based on intuition/reference
- Adjustment based on results

# Weights Initialization

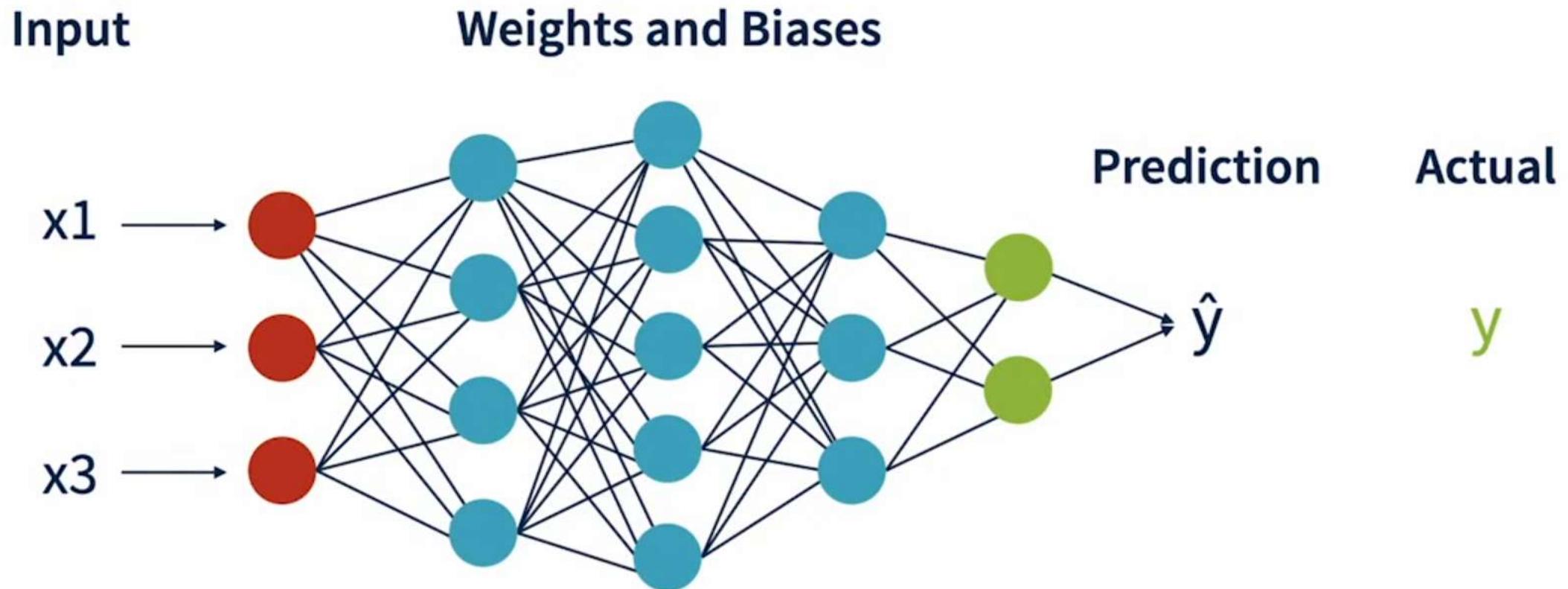
- All weights and bias parameters need to be initialized to some value before we start training
- Zero initialization: Initialize to zeros, not recommended
- Random initialization: Initialize to random values from a standard normal distribution (mean = 0, SD = 1)

# Forward Propagation

# Inputs, Targets, and Predictions

	Sample 1	Sample 2	Sample 3	Sample 4
Feature 1	x <sub>11</sub>	x <sub>21</sub>	x <sub>31</sub>	x <sub>41</sub>
Feature 2	x <sub>12</sub>	x <sub>22</sub>	x <sub>32</sub>	x <sub>42</sub>
Feature 3	x <sub>13</sub>	x <sub>23</sub>	x <sub>34</sub>	x <sub>43</sub>
Target	y <sub>1</sub>	y <sub>2</sub>	y <sub>3</sub>	y <sub>4</sub>
Prediction	$\hat{y}_1$	$\hat{y}_2$	$\hat{y}_3$	$\hat{y}_4$

# Forward Propagation: 1 Sample

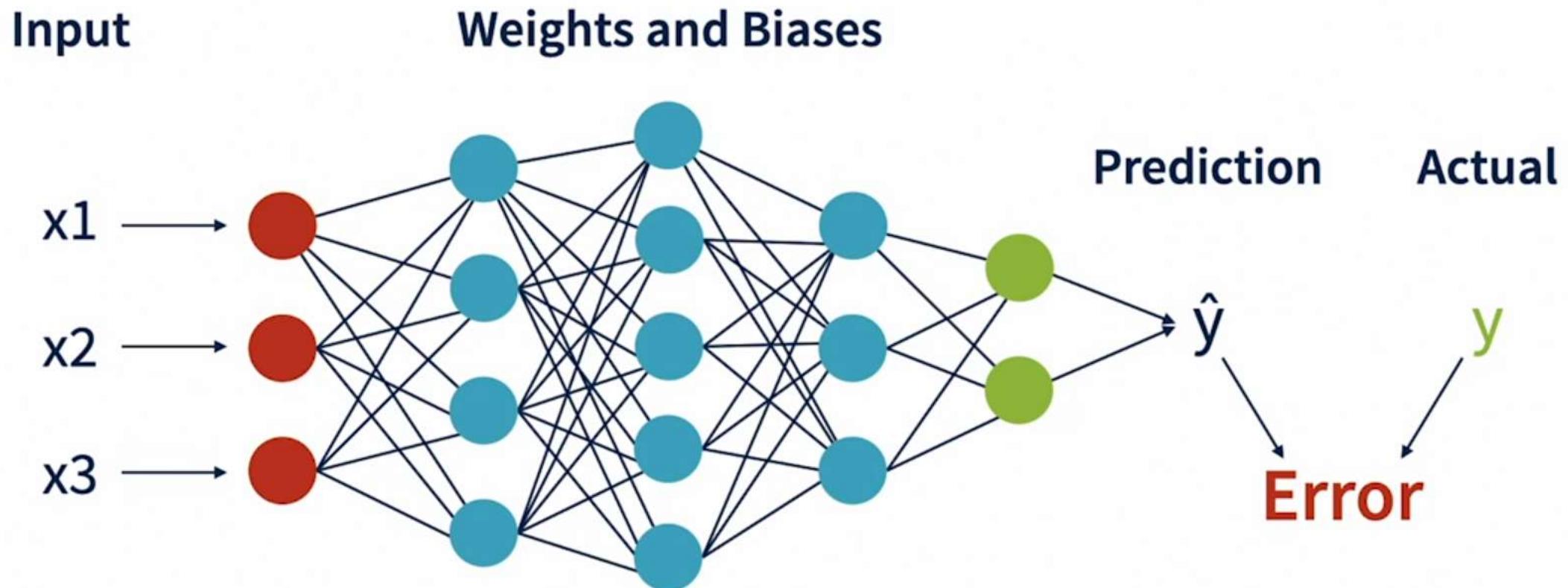


# Forward Propagation: All Samples

- Send each sample through the neutral network and obtain the value of  $\hat{y}$
- Repeat for all samples and collect a set of  $\hat{y}$
- Compare the values of  $\hat{y}$  to  $y$  to obtain error rates

# **Measuring Accuracy and Error**

# Error in Prediction



# Loss and Cost Function

- A loss function measures the prediction error for a single sample
- A cost function measures the error across a set of samples

# Popular Cost Functions

Cost Functions	Applications
Mean Square Error ( MSE )	Regression
Root Mean Square Error ( RMSE )	Regression
Binary Cross Entropy	Binary classification
Categorical Cross Entropy	Multi-class classification

# Measuring Accuracy

- Send a set of samples through the ANN and predict outcome
- Estimate the prediction error between the predicted outcome and expected outcome using a cost function
- Use back propagation to adjust weights based on the error value

# Back Propagation

# Why Back Propagate?

- Each node in a neural network contributes to the overall error in prediction (differing contributions)
- A node's contribution is driven by its weights and bias
- Weights and biases need to be adjusted to lower the error contribution by each node

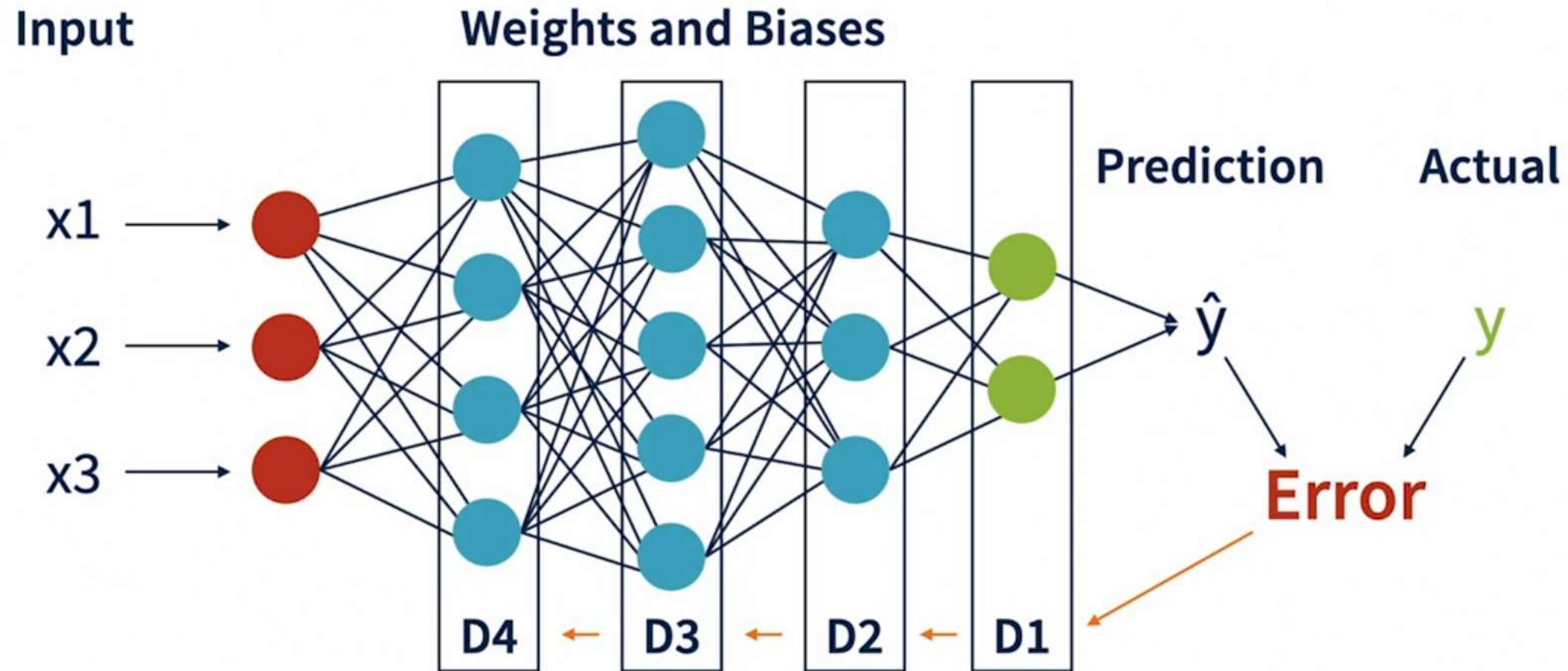
# How Back Propagation Works

- Back propagation works in reverse of the forward propagation
- Start from the output layer
- Compute a delta value based on the error found
- Apply the delta to adjust the weights and biases in the layer

# How Back Propagation Works

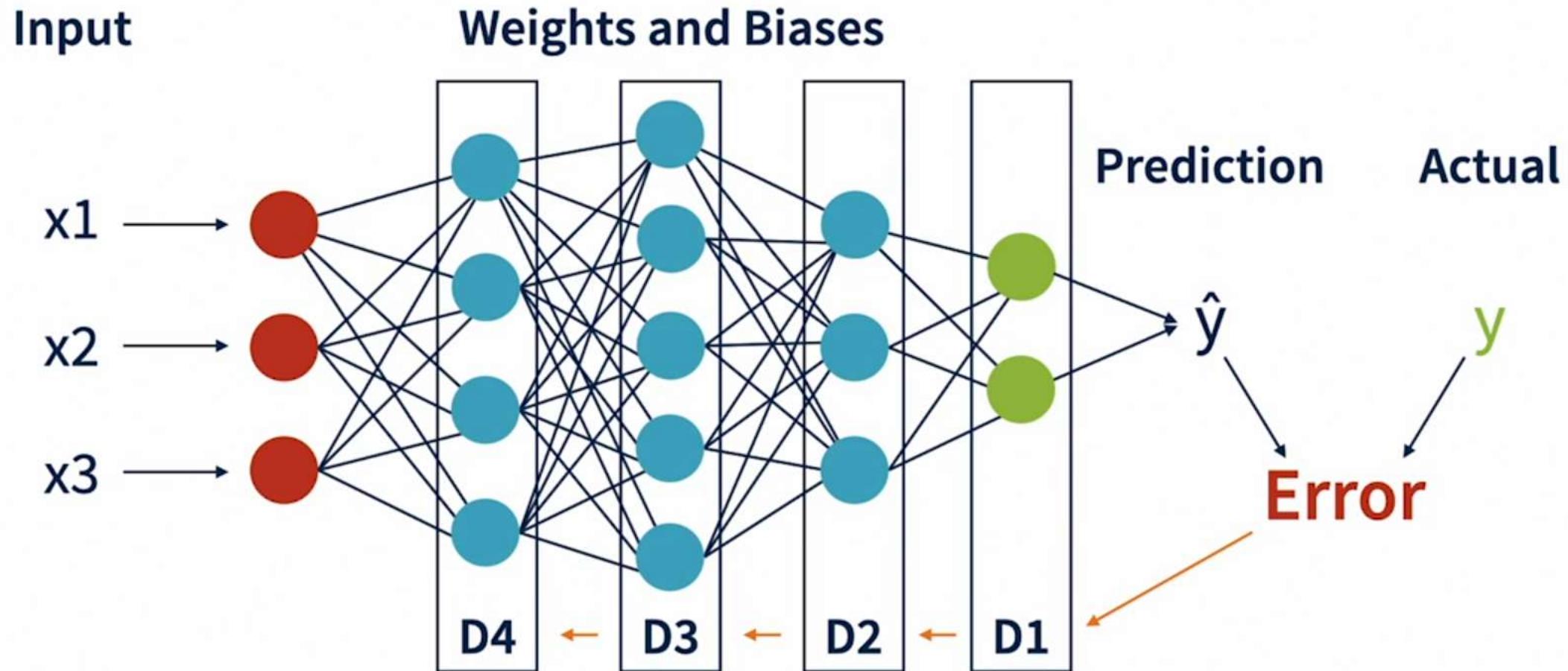
- Derive a new error value
- Back propagate the new error to the previous layer and repeat

# Back Propagation



# Gradient Descent

# Back Propagation



# Gradient Descent Process

Repeat the learning process.

- Forward propagation
- Estimate error
- Backward propagate
- Adjust weights and biases



# Batches and Epoch

# Batch

- A set of samples sent through ANN in a single pass
- The training data set can be divided into one or more batches
- Training data is sent to the ANN one batch at a time
- Cost estimated and parameters updated one batch at a time

# Batch

- Batch gradient descent: batch size = training set size
- Mini-batch gradient descent: batch size < training set size
- Typical batch sizes are 32, 64, 128, etc.

# Epoch

- The number of times the entire training set is sent through the ANN
- An epoch has one or more batches
- The training process completes when all epoch is complete
- Epoch sizes can be higher to achieve better accuracy

# Epoch and Batch Example

- Training set size = 1000, batch size = 128, epoch = 50
- Batches per epoch =  $\text{ceil} ( 1000 / 128 ) = 8$
- Total iterations (passes) through ANN =  $8 * 50 = 400$
- Batch size and epoch are hyperparameters that can be tuned to improve model accuracy

# **Validation and Testing**

# Learning Process: Validation

- During learning, the predictions are obtained for the same data that is used to train the parameters (weights and biases)
- After each epoch and corresponding parameter updates, the model can be used to predict for the validation data set
- Accuracy and/or loss can be measured and investigated
- Model can be fine-tuned and learning process repeated based on results

# Evaluation

- After all fine-tuning is completed and final model obtained, the test data set can be used to evaluate the model
- Results obtained with test data set is used to measure the performance of the model

**What is an ANN model and what does it contain?**

# ANN Model

## Parameters

- Weights
- Biases

## Hyperparameters

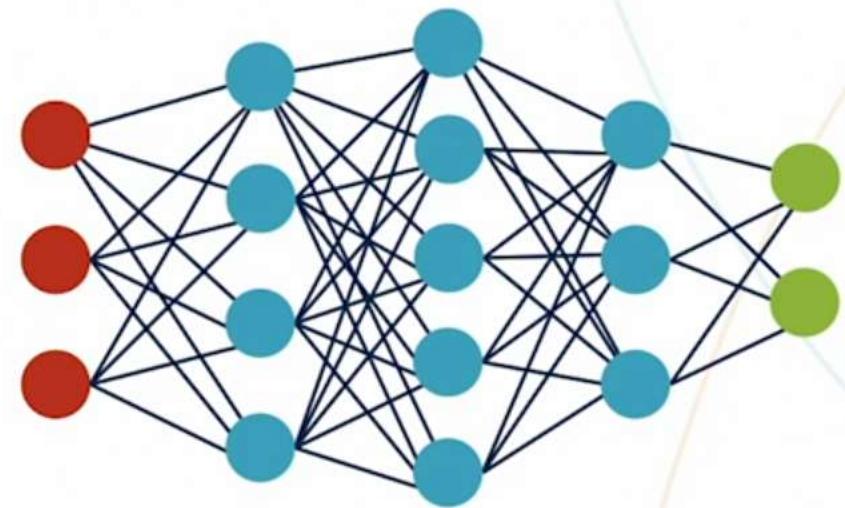
- Number of layers, nodes in each layer, activation function
- Cost functions, learning rate, optimizers
- Batch size, epoch

# Prediction Process

Preprocess and prepare inputs

Pass inputs to the first layer

- Compute  $Y$  using weights, biases, activation
- Pass to the next layer



Repeat process until output layer

Postprocess output for predictions

# The Iris Classification Problem

# Feature Variables

- Sepal length
- Sepal width
- Petal length
- Petal width





Build an ANN model to predict species.

# Image Recognition



# Keras

- Keras is a high-level library for building neural networks in Python with only a few lines of code



# A Simple Neural Network

```
model = keras.models.Sequential()  
model.add(Dense(3, input_dim=2, activation='relu'))  
model.add(Dense(3, activation='relu'))  
model.add(Dense(1))  
model.compile(optimizer='adam', loss='mse')
```

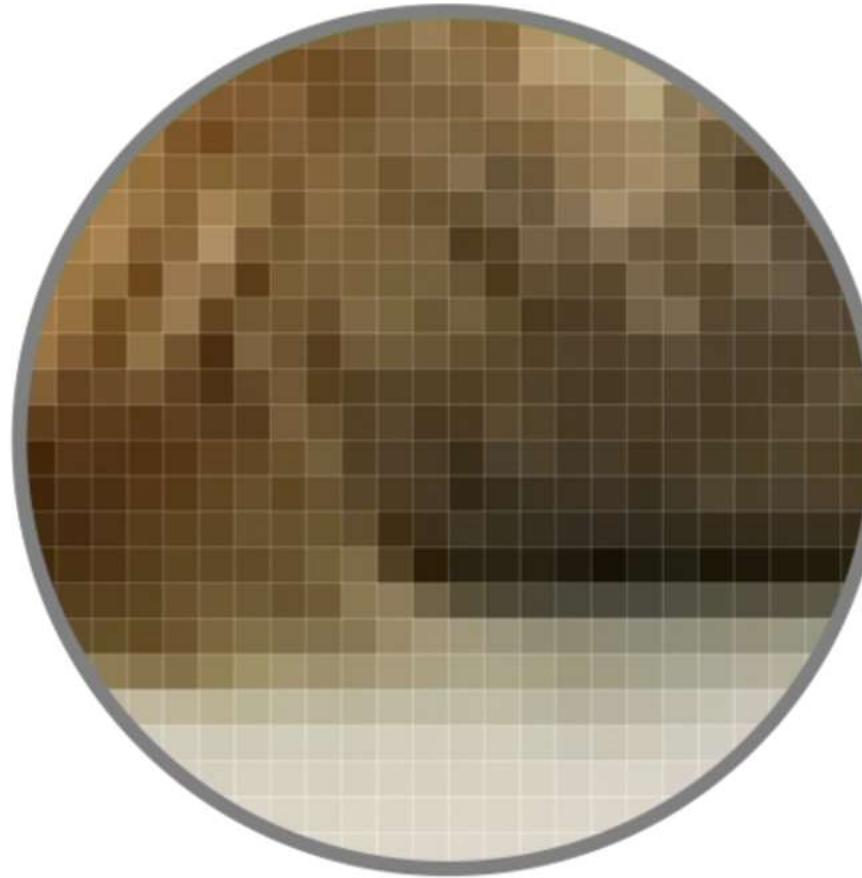
# Using Images as Input to a Neural Network



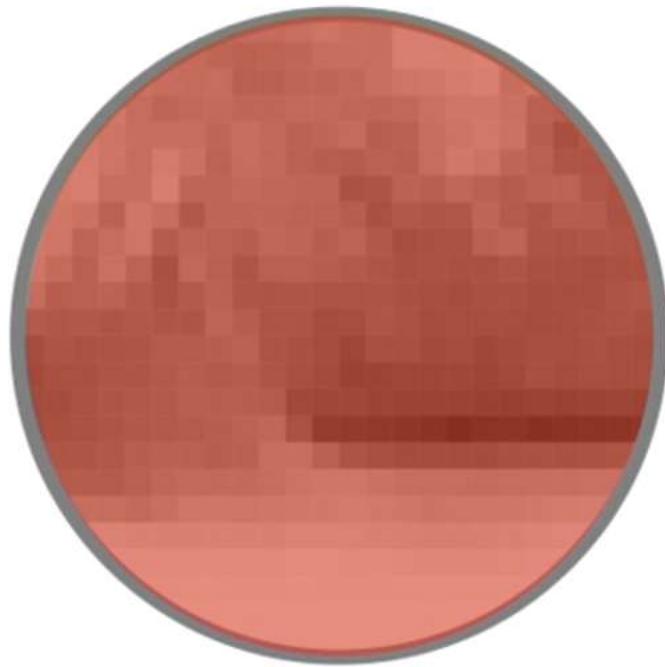
---

Digital Images

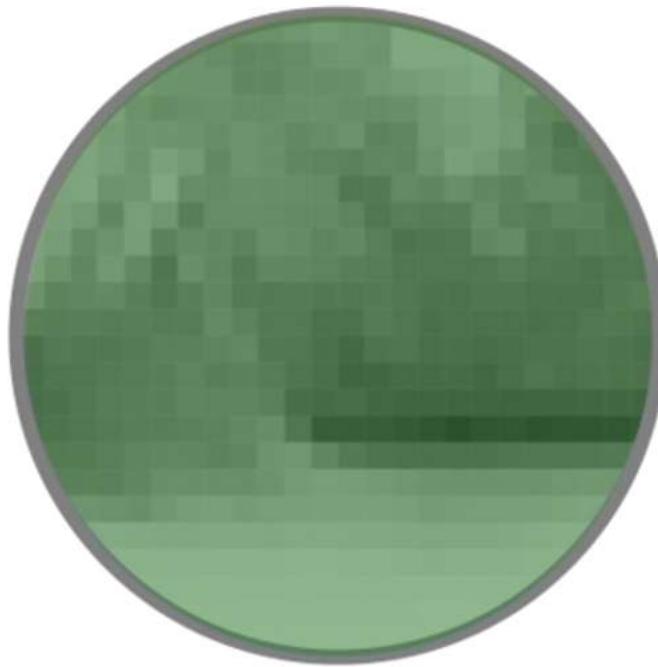
# Digital Images



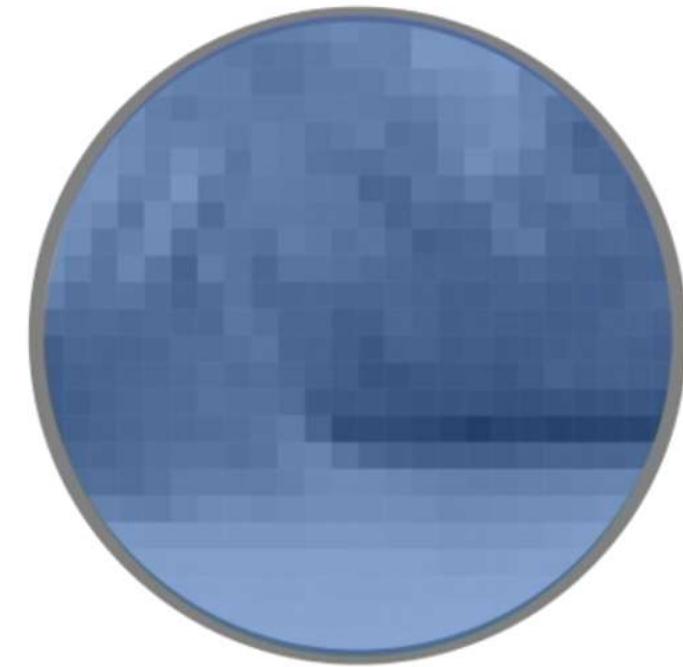
# Digital Images



Red channel



Green channel



Blue channel

# Red Channel

- Red channel =

161	162	163	146	140	128	137	146	142	141	149	159	148	145	180	183	192	198	186	156	138	142	116	101
170	164	166	152	149	145	141	131	122	124	131	145	148	145	184	184	179	188	178	176	149	128	108	104
174	149	155	167	156	144	141	142	121	121	132	132	135	149	159	166	173	184	148	160	148	114	118	105
173	147	158	169	142	128	136	139	132	130	137	139	133	131	136	151	161	165	152	121	114	112	104	96
159	156	173	159	144	145	142	132	143	135	123	131	141	114	120	147	157	168	155	114	88	104	107	96
157	164	175	160	143	170	155	124	143	135	122	116	112	122	118	142	168	157	132	119	100	96	107	107
164	172	177	143	146	181	130	128	138	133	130	128	93	97	126	118	134	119	111	123	126	102	92	99
167	167	155	122	163	148	105	131	130	131	126	126	115	90	106	101	102	131	109	100	113	112	93	92
170	161	127	143	160	111	125	122	114	134	129	115	124	105	86	89	92	121	129	97	95	101	98	93
165	141	119	153	124	89	135	119	98	120	119	115	108	101	91	92	90	90	101	90	88	86	90	91
142	114	120	124	107	92	108	122	98	96	98	95	94	97	91	85	83	89	89	83	85	85	96	102
107	102	104	101	106	101	100	127	114	97	97	83	84	99	92	82	85	84	84	82	84	91	95	89
82	97	93	94	98	109	109	115	123	95	91	82	70	87	88	79	82	75	80	84	85	89	86	82
79	89	90	98	100	108	115	107	112	100	88	81	64	67	72	72	74	69	77	88	83	87	86	86
80	83	92	101	104	107	117	115	116	109	76	72	75	68	64	64	65	62	61	64	63	67	69	65
80	87	93	99	105	107	110	113	124	133	92	54	53	51	48	41	39	41	43	37	40	41	43	48
82	98	99	104	116	121	121	114	121	145	147	119	97	87	83	83	83	89	91	91	93	96	103	
95	109	111	110	119	134	138	137	138	143	158	164	161	155	147	143	142	141	143	144	147	151	152	153
134	142	146	142	138	150	161	163	159	162	167	169	171	169	165	158	160	162	165	168	173	176	176	177
188	186	187	190	187	185	185	187	186	185	188	188	188	187	183	179	181	184	187	192	196	195	193	195
199	200	204	205	204	203	203	204	203	203	207	205	203	204	200	197	198	199	201	204	205	203	205	204
207	208	208	209	208	208	210	210	209	209	211	209	209	209	210	210	212	211	210	211	211	212	212	211
213	213	212	212	213	214	214	213	213	214	213	213	214	214	215	215	214	216	214	215	215	214	214	215

# Red Channel

- Red channel =

161	162	163	146	140	128	137	146	142	141	149	159	148	145	180	183	192	198	186	156	138	142	116	101
170	164	166	152	149	145	141	131	122	124	131	145	148	145	184	184	179	188	178	176	149	128	108	104
174	149	155	167	156	144	141	142	121	121	132	132	135	149	159	166	173	184	148	160	148	114	118	105
173	147	158	169	142	128	136	139	132	130	137	139	133	131	136	151	161	165	152	121	114	112	104	96
159	156	173	159	144	145	142	132	143	135	123	131	141	114	120	147	157	168	155	114	88	104	107	96
157	164	175	160	143	170	155	124	143	135	122	116	112	122	118	142	168	157	132	119	100	96	107	107
164	172	177	143	146	181	130	128	138	133	130	128	93	97	126	118	134	119	111	123	126	102	92	99
167	167	155	122	163	148	105	131	130	131	126	126	115	90	106	101	102	131	109	100	113	112	93	92
170	161	127	143	160	111	125	122	114	134	129	115	124	105	86	89	92	121	129	97	95	101	98	93
165	141	119	153	124	89	135	119	98	120	119	115	108	101	91	92	90	90	101	90	88	86	90	91
142	114	120	124	107	92	108	122	98	96	98	95	94	97	91	85	83	89	89	83	85	85	96	102
107	102	104	101	106	101	100	127	114	97	97	83	84	99	92	82	85	84	84	82	84	91	95	89
82	97	93	94	98	109	109	115	123	95	91	82	70	87	88	79	82	75	80	84	85	89	86	82
79	89	90	98	100	108	115	107	112	100	88	81	64	67	72	72	74	69	77	88	83	87	86	86
80	83	92	101	104	107	117	115	116	109	76	72	75	68	64	64	65	62	61	64	63	67	69	65
80	87	93	99	105	107	110	113	124	133	92	54	53	51	48	41	33	41	43	37	40	41	43	48
82	98	99	104	116	121	121	114	121	145	147	119	97	87	83	83	83	89	91	91	93	96	103	
95	109	111	110	119	134	138	137	138	143	158	164	161	155	147	143	142	141	143	144	147	151	152	153
134	142	146	142	138	150	161	163	159	162	167	169	171	169	165	158	160	162	165	168	173	176	176	177
188	186	187	190	187	185	185	187	186	185	188	188	188	187	183	179	181	184	187	192	196	195	193	195
199	200	204	205	204	203	203	204	203	203	207	205	203	204	200	197	198	199	201	204	205	203	205	204
207	208	208	209	208	208	210	210	209	209	211	209	209	209	210	210	212	211	211	211	212	212	211	211
213	213	212	212	213	214	214	213	213	214	213	213	214	213	214	214	215	215	214	216	214	214	215	215
213	213	214	214	216	215	217	217	215	216	216	216	217	217	215	216	216	215	215	217	217	216	216	217

# Full Image

- Image (Three color channels) =

174	170	171	161	162	163	146	140	128	137	146	142	141	149	159	148	145	180	183	192	198	186	156	138	142	116	101	117	133	117	18	101				
171	172	171	170	164	166	152	149	145	141	131	122	124	131	145	148	145	184	184	179	188	178	176	149	128	108	104	120	110	93	95	79				
173	171	181	174	149	155	167	156	144	141	142	121	121	132	132	135	149	159	166	173	184	148	160	148	114	118	105	95	87	97	95	79				
172	168	188	173	147	158	169	142	128	136	139	132	130	137	139	133	131	136	151	161	165	152	121	114	112	104	96	96	102	102	74	83				
153	165	182	159	156	173	159	144	145	142	132	143	135	123	131	141	114	120	147	157	168	155	114	88	104	107	96	93	93	88	99	88				
144	175	176	157	164	175	160	143	170	155	124	143	135	122	116	112	122	118	142	168	157	132	119	100	96	107	107	92	79	82	78	72				
155	181	164	164	172	177	143	146	181	130	128	138	133	130	128	93	97	126	118	134	119	111	123	126	102	92	99	97	85	89	67	66				
177	172	167	167	167	155	122	163	148	105	131	130	131	126	126	115	90	106	101	102	131	109	100	113	112	93	92	92	96	107	71	73				
180	168	171	170	161	127	143	160	111	125	122	114	134	129	115	124	105	86	89	92	121	129	97	95	101	98	93	95	102	107	82	91				
169	163	162	165	141	119	153	124	89	135	119	98	120	119	115	108	101	91	92	90	90	101	90	88	86	90	91	100	100	83	89	92				
157	157	162	142	114	120	124	107	92	108	122	98	96	98	95	94	97	91	85	83	89	89	83	85	85	85	96	102	95	93	84	85	67			
146	151	138	107	102	104	101	106	101	100	127	114	97	97	83	84	99	92	82	85	84	84	82	84	91	95	89	86	94	91	77	68				
125	121	92	82	97	93	94	98	109	109	115	123	95	91	82	70	87	88	79	82	75	80	84	85	89	86	82	86	89	86	78	75				
103	92	79	79	89	90	98	100	108	115	107	112	100	88	81	64	67	72	72	74	69	77	88	83	87	86	86	88	84	73	69	53				
82	76	76	80	83	92	101	104	107	117	115	116	109	76	72	75	68	64	64	65	62	61	64	63	67	69	65	68	66	66	71	69				
70	74	74	80	87	93	99	105	107	110	113	124	133	92	54	53	51	48	41	33	41	43	37	40	41	43	48	52	52	55	58	56				
72	82	81	82	98	99	104	116	121	121	114	121	145	147	119	97	87	83	83	83	83	89	91	91	93	96	103	110	111	118	47	50				
80	82	82	95	109	111	110	119	134	138	137	138	143	158	164	161	155	147	143	142	141	143	144	147	151	152	153	156	157	157	06	114				
141	132	126	134	142	146	142	138	150	161	163	159	162	167	169	171	169	165	158	160	162	165	168	173	176	176	177	178	179	180	55	154				
191	192	188	188	186	187	190	187	185	185	187	186	185	188	188	188	187	183	179	181	184	187	192	196	195	193	195	197	197	197	76	176				
195	198	201	199	200	204	205	204	203	203	204	203	207	205	203	204	200	197	198	199	201	204	205	203	205	204	204	206	206	93	193					
204	204	207	207	208	208	209	208	208	210	210	209	209	211	209	209	210	210	212	211	210	211	211	212	212	211	213	212	02	202						
212	212	211	213	213	212	212	213	214	214	213	214	213	214	214	215	215	214	216	214	214	215	214	214	215	215	216	216	216	217	216	216	216	216	216	
215	215	215	213	213	214	214	216	215	217	217	215	216	216	216	217	217	215	216	216	215	217	217	216	216	216	217	216	216	216	216	216	216	216	216	
206	207	208	207	207	208	208	210	208	209	211	210	210	210	210	211	210	210	211	210	210	211	210	209	211	211	210	210	211	210	210	210	210	210	210	
195	195	194	194	193	194	193	194	194	195	194	195	193	195	195	196	196	197	196	196	197	195	195	196	197	197	196	196	197	196	196	197	196	196	195	195

## Number of Neural Net Input Nodes

- For a small 256x256 pixel image
- $256 \times 256 \times 3 = 196,608$  input nodes required
- And that's just in the input layer!

# Working with Image Data

- Images are computationally intensive
- Image recognition systems tend to use small image sizes
- Images are scaled down to that smaller size before being fed into a neural network

# Recognizing Image Contents with a Neural Network

# Training Data Set

## "8"



61.png



84.png



110.png



128.png



179.png



181.png



184.png



226.png



257.png



260.png



266.png



268.png



338.png



344.png



355.png



373.png

## Not "8"



34.png



51.png



54.png



75.png



100.png



104.png



111.png



115.png



118.png



135.png



154.png



156.png



165.png



170.png



187.png



199.png

# Training Phase



# Training Phase



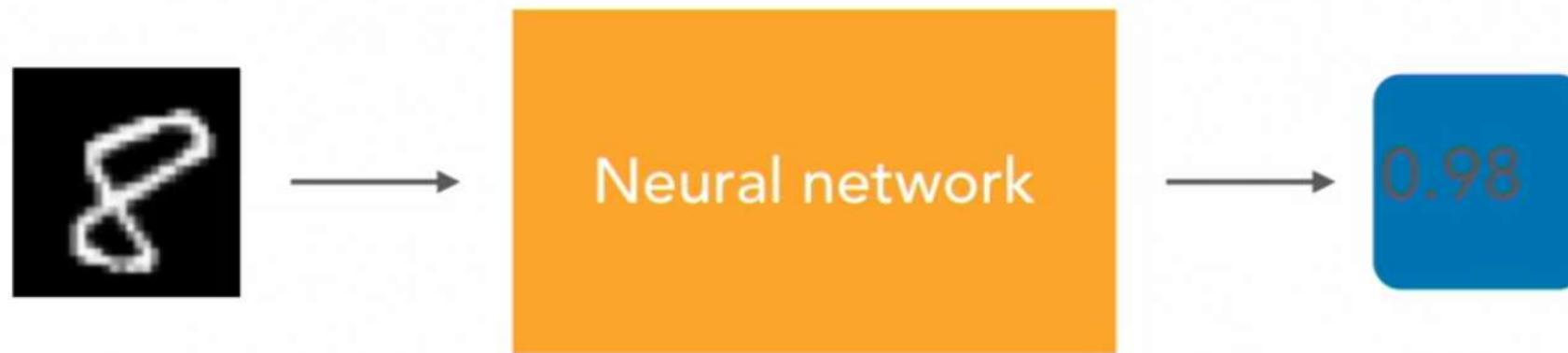
# Training Phase



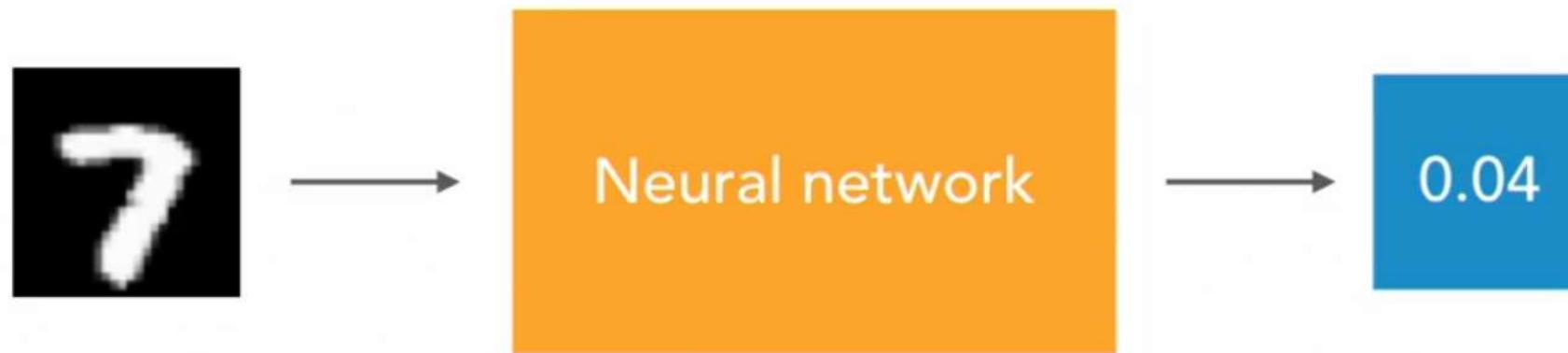
# Inference Phase



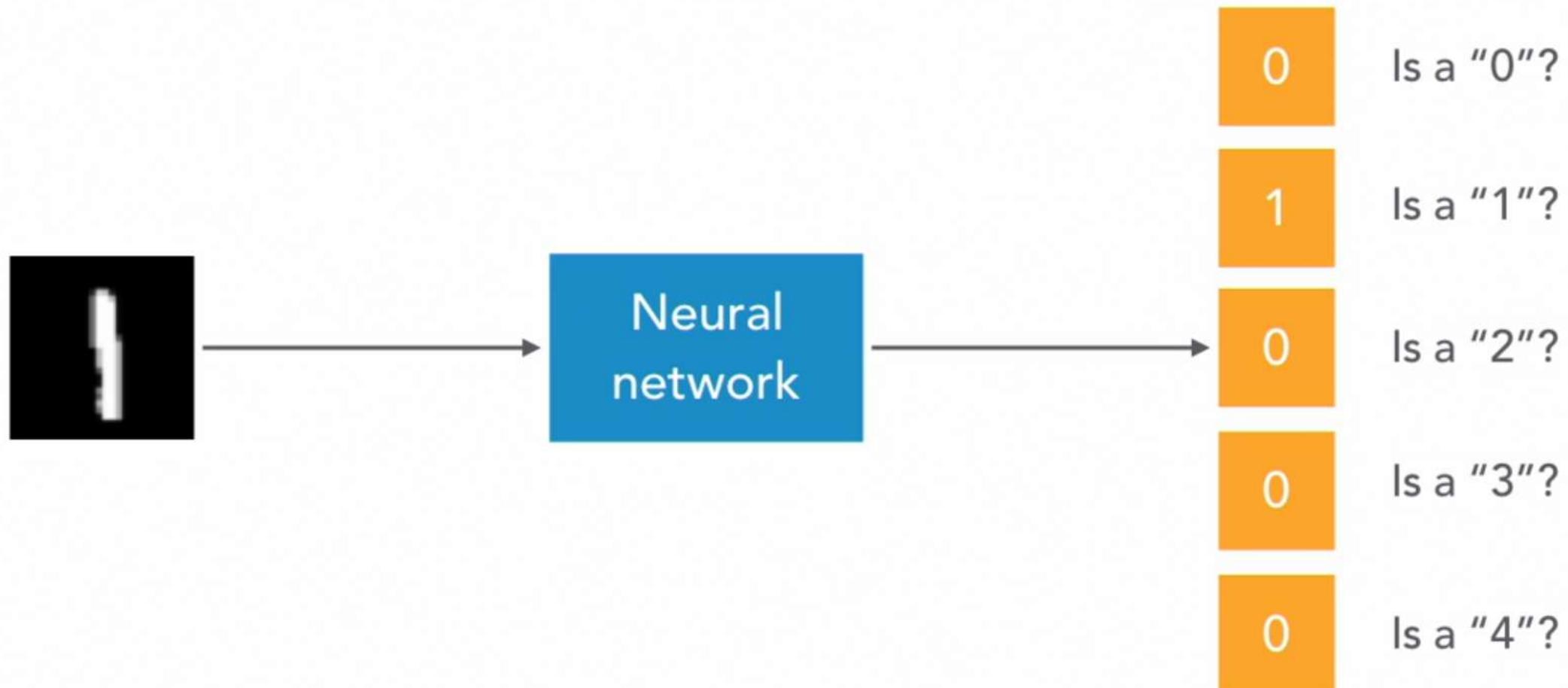
# Inference Phase



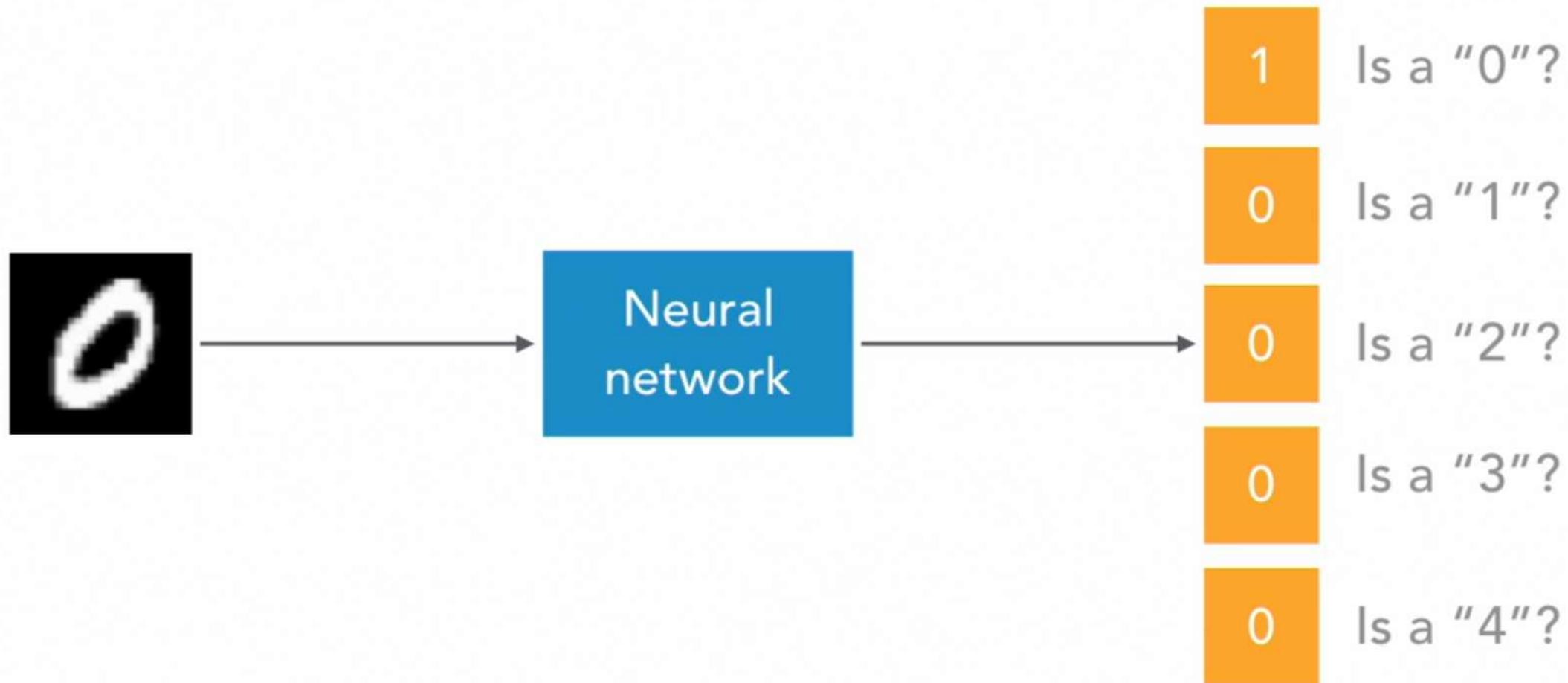
# Inference Phase



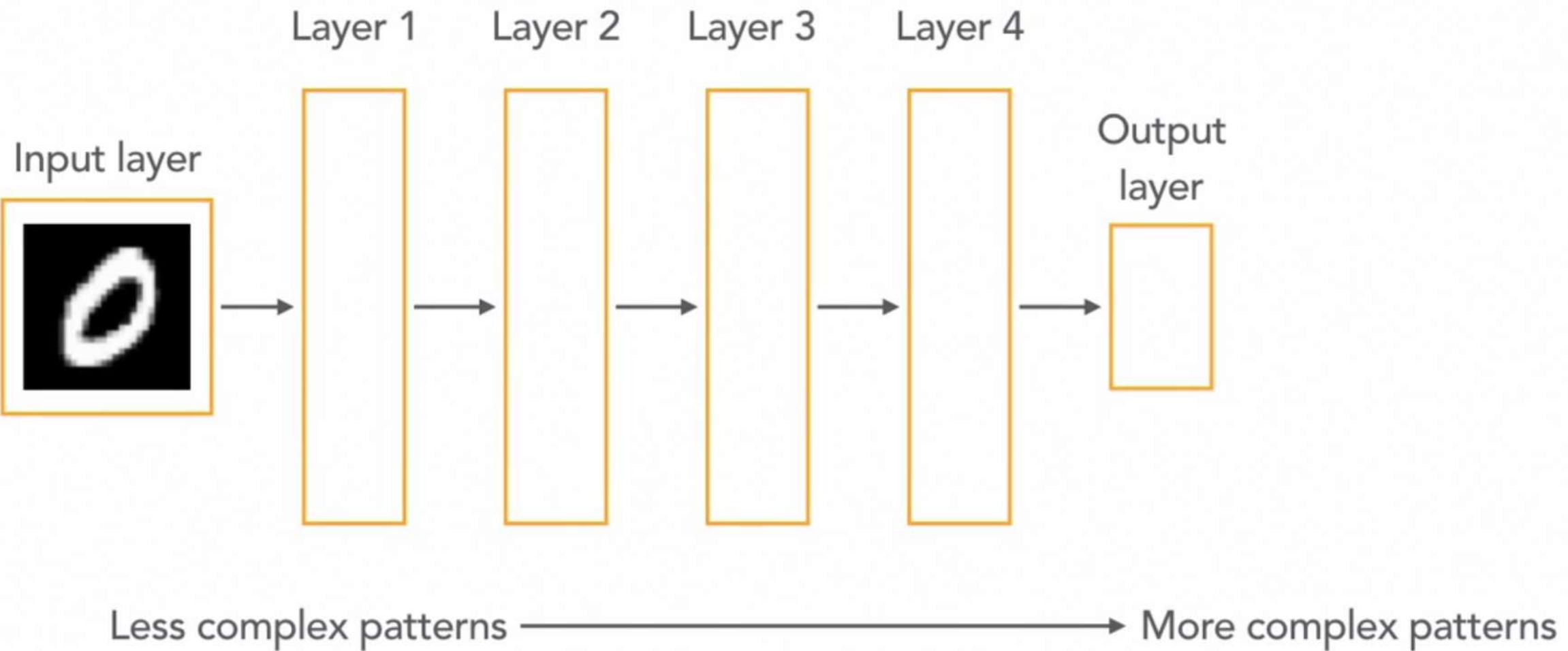
# Training Phase



# Training Phase



# Deep Neural Networks



# Adding Convolution for Translational Invariance

# Traditional Image Classification



# Traditional Image Classification



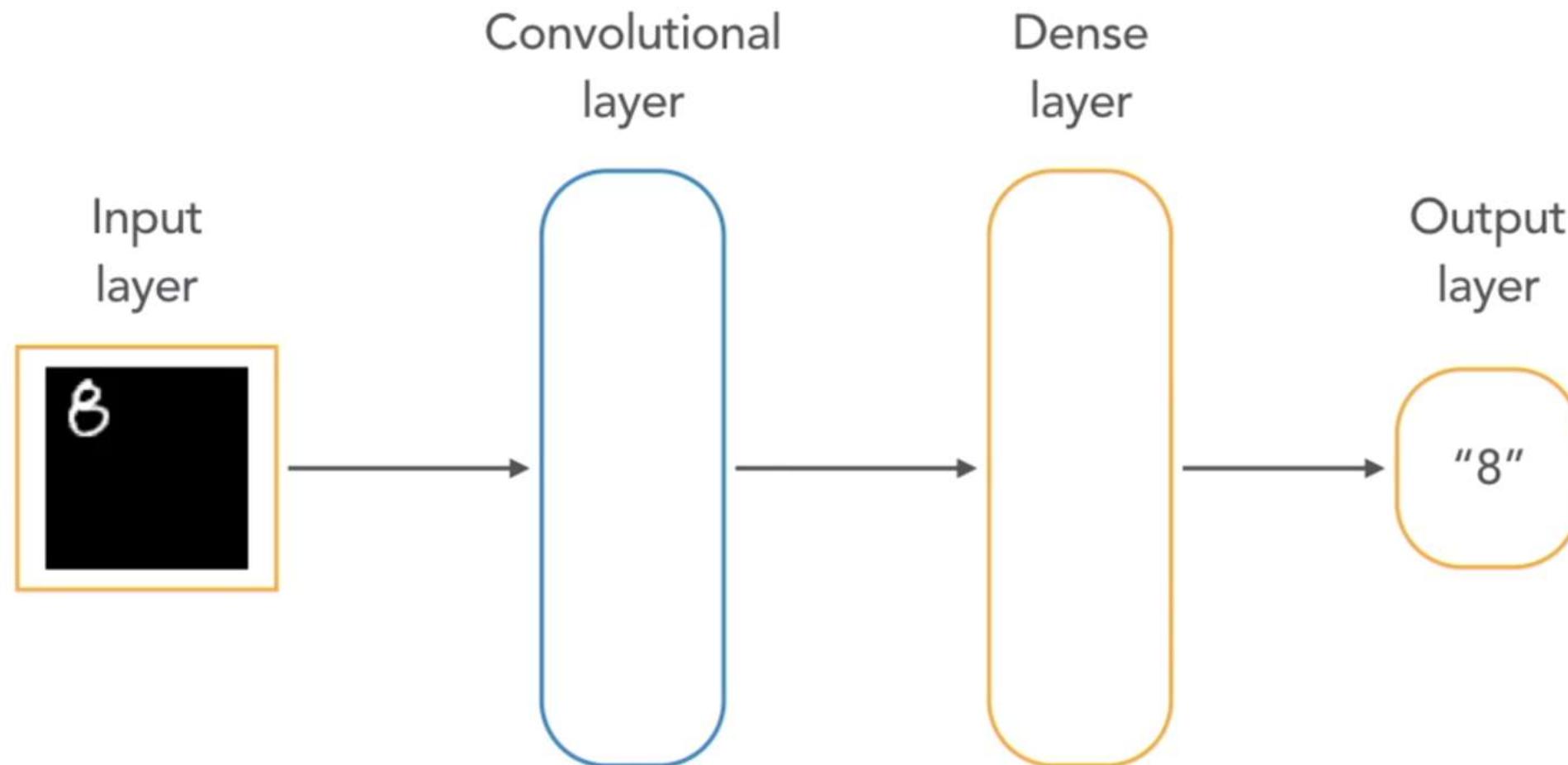


---

## Translation invariance

When a machine learning model can recognize an object no matter whether it is moved (or *translated*) in the image

# Convolutional Neural Network



## Convolution: Step 1

Input image

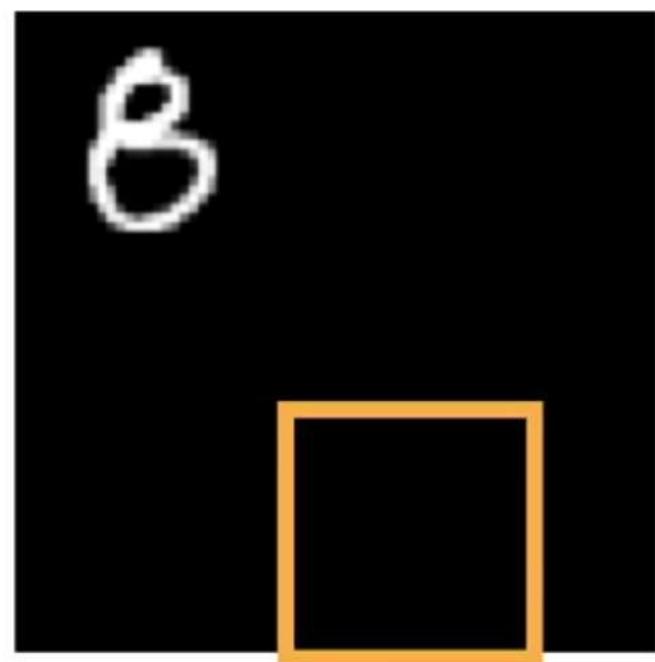
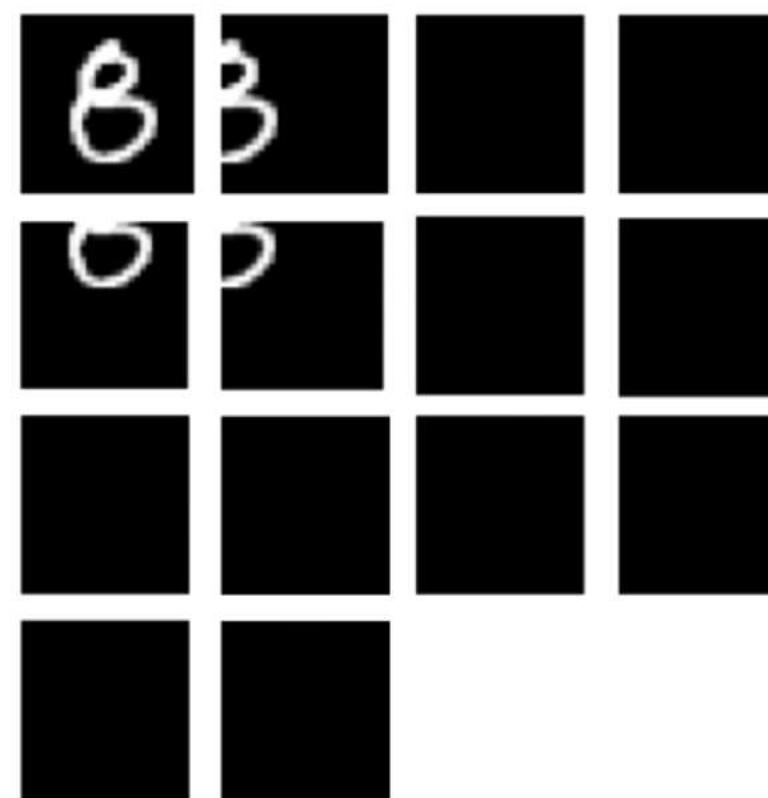


Image tiles



# Convolution: Step 1

Input image

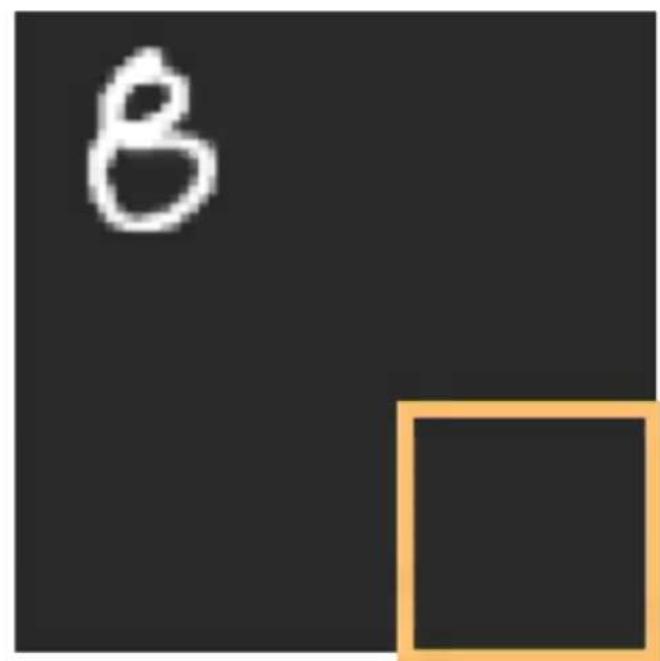
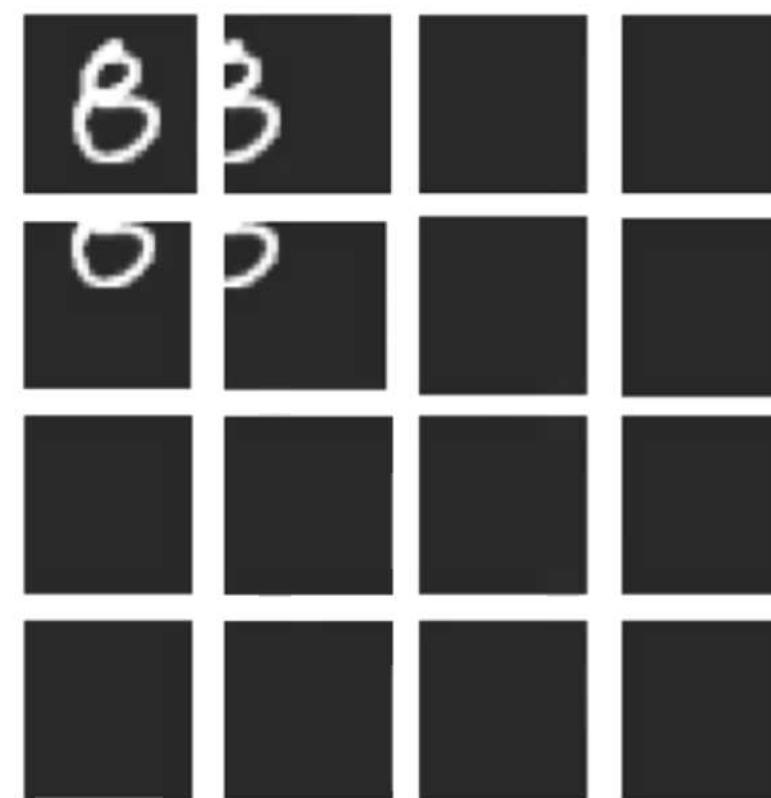
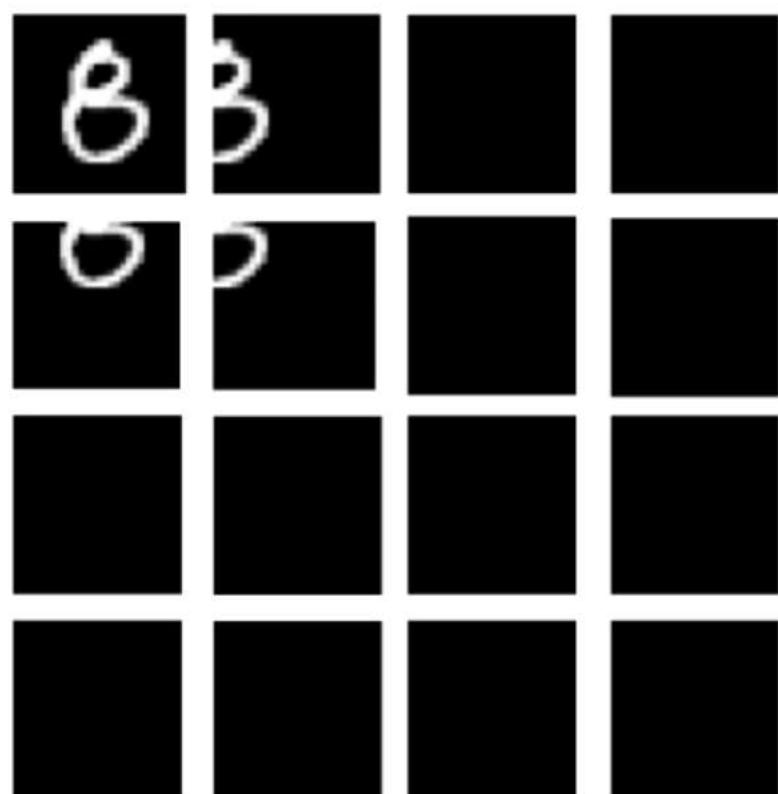


Image tiles



## Convolution: Step 2

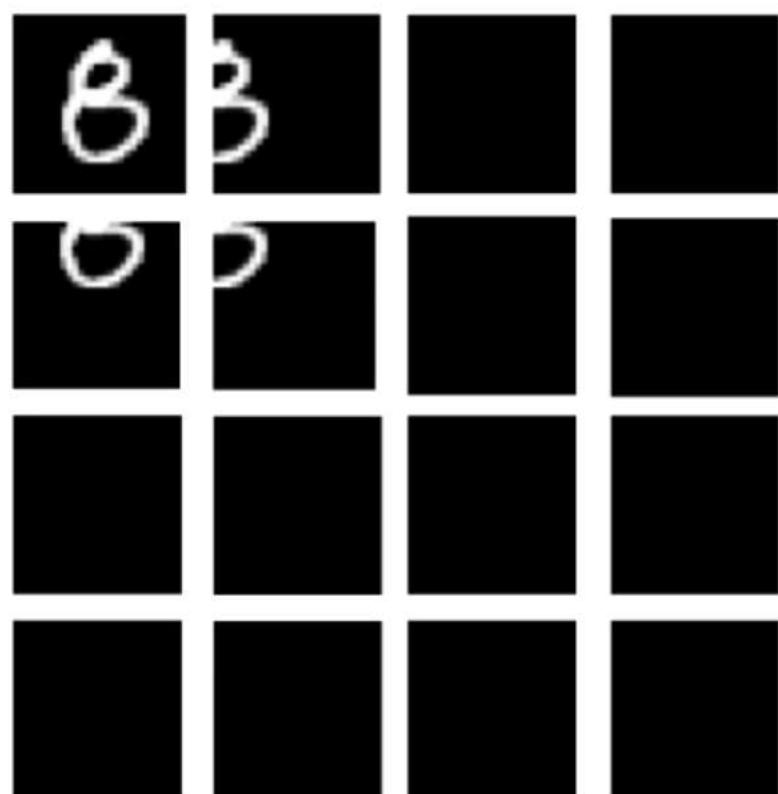
Image tiles



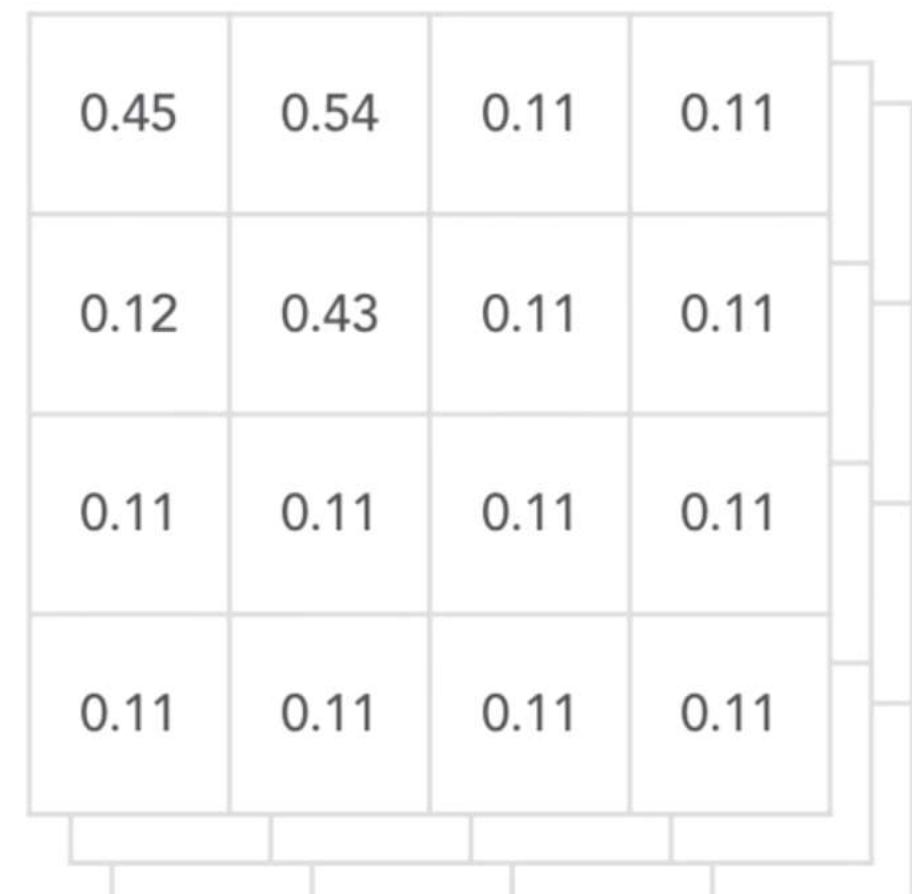
0.12	0.85	0.0	0.0
0.76	0.321	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

## Convolution: Step 3

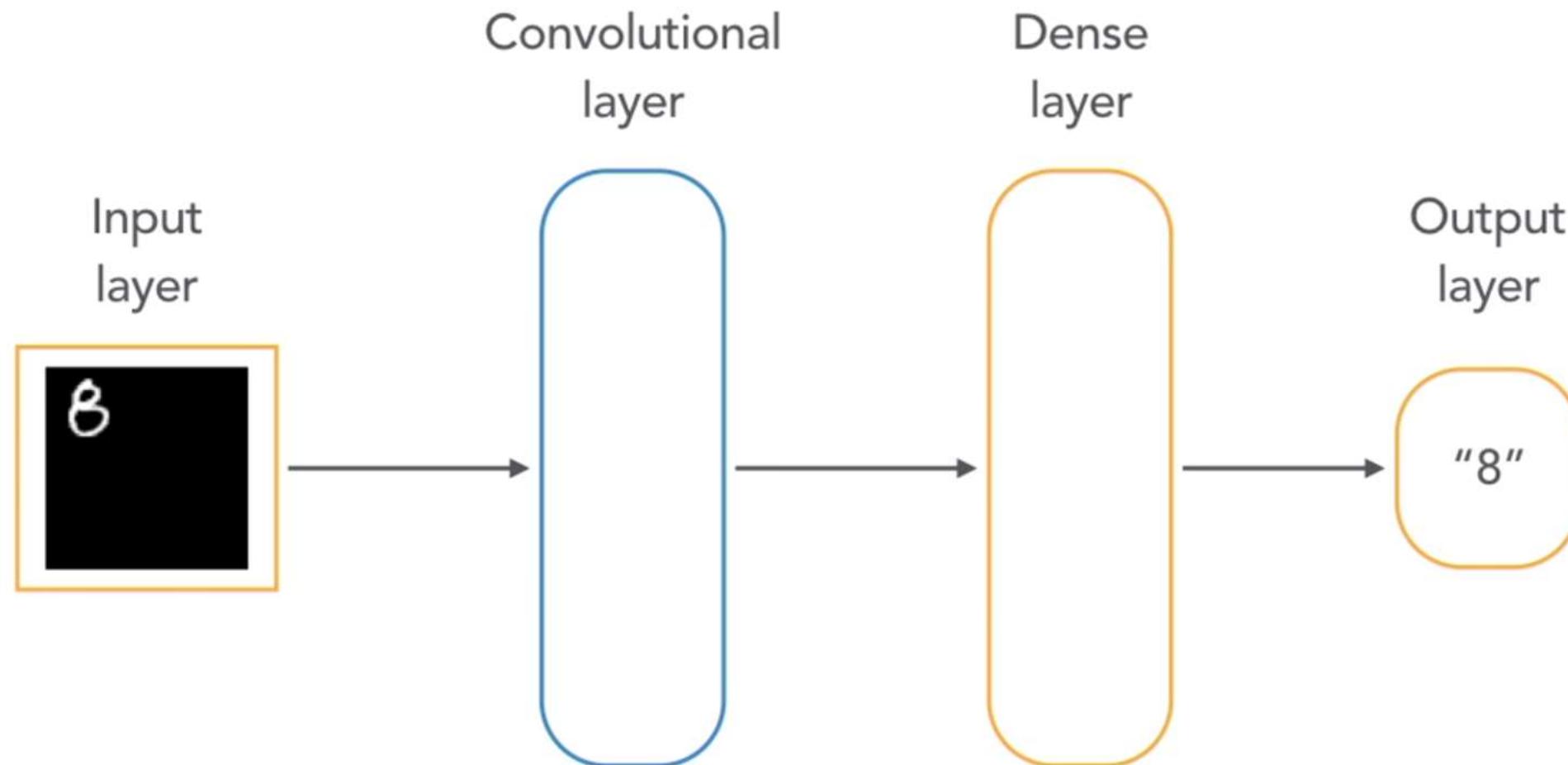
Image tiles



Neural  
network  
layer

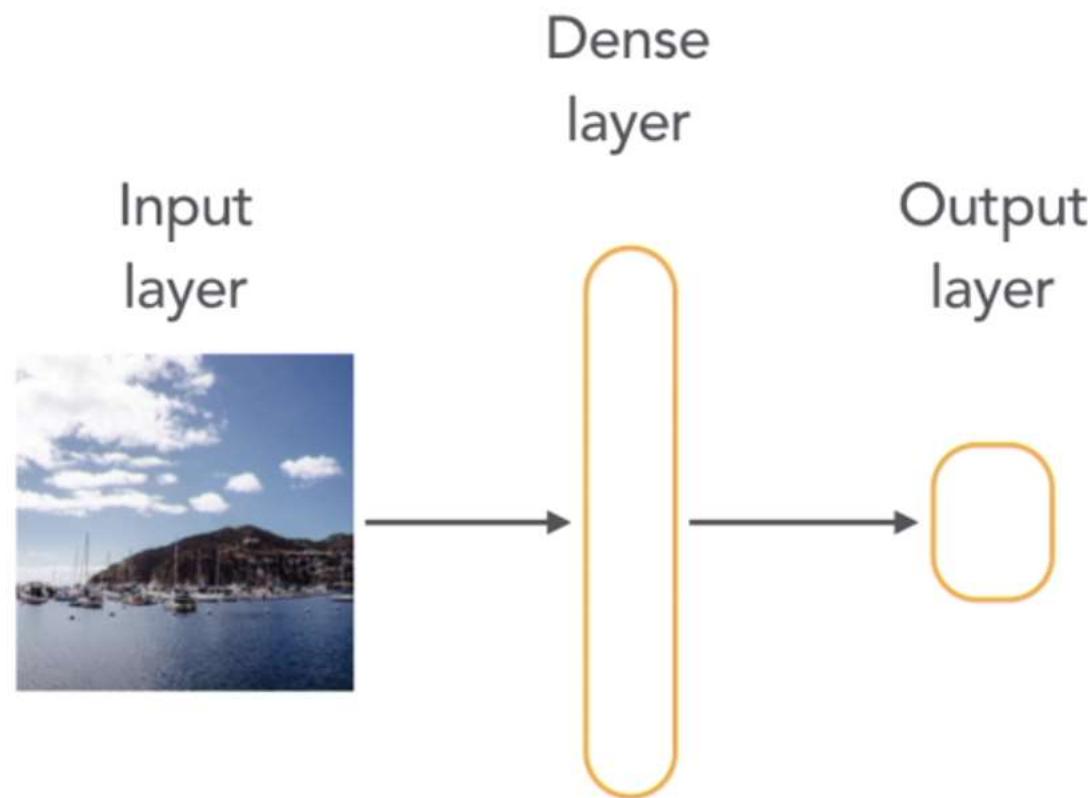


# Convolutional Neural Network

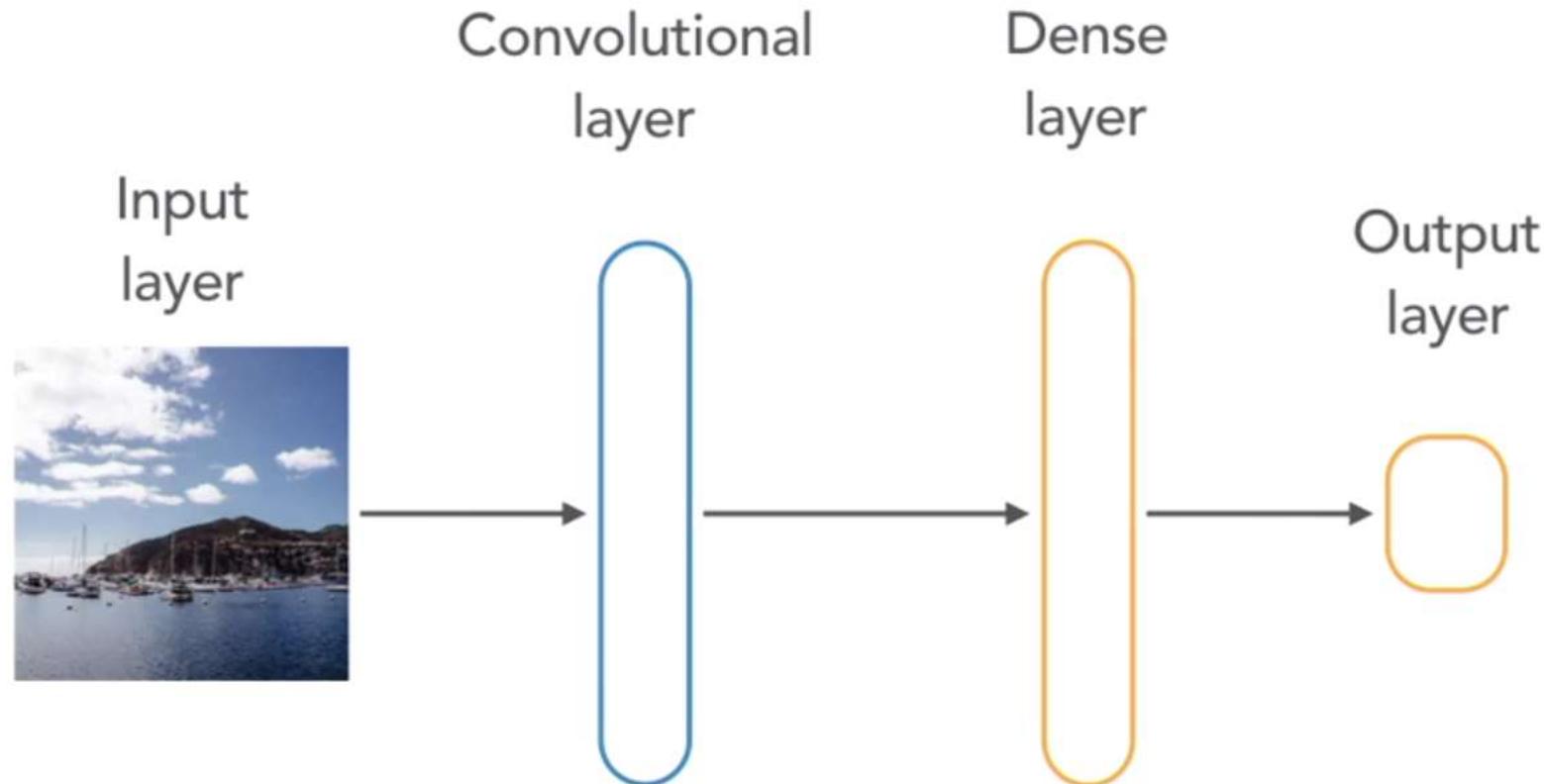


# Designing a Neural Network Architecture for Image Recognition

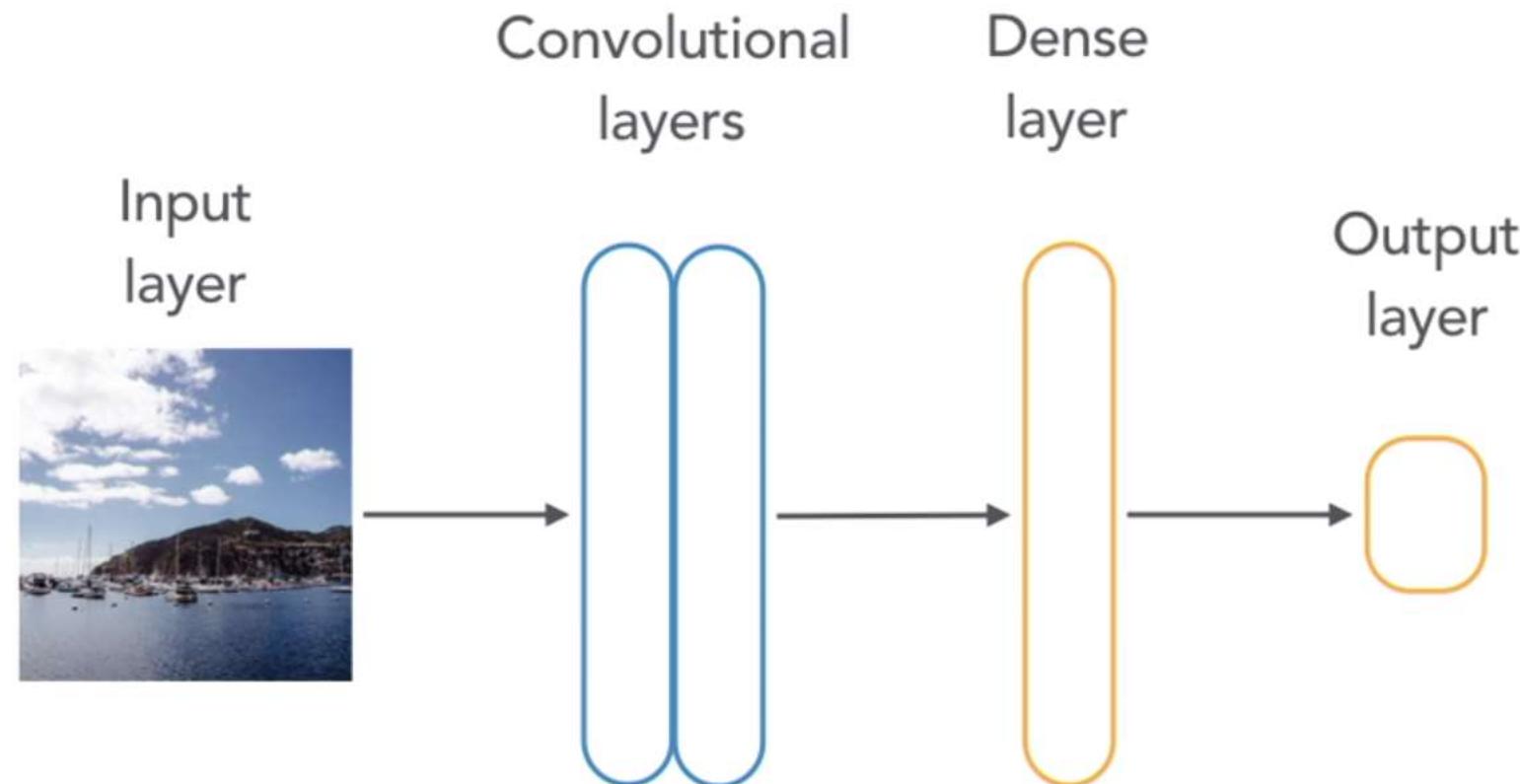
# Basic Neural Network



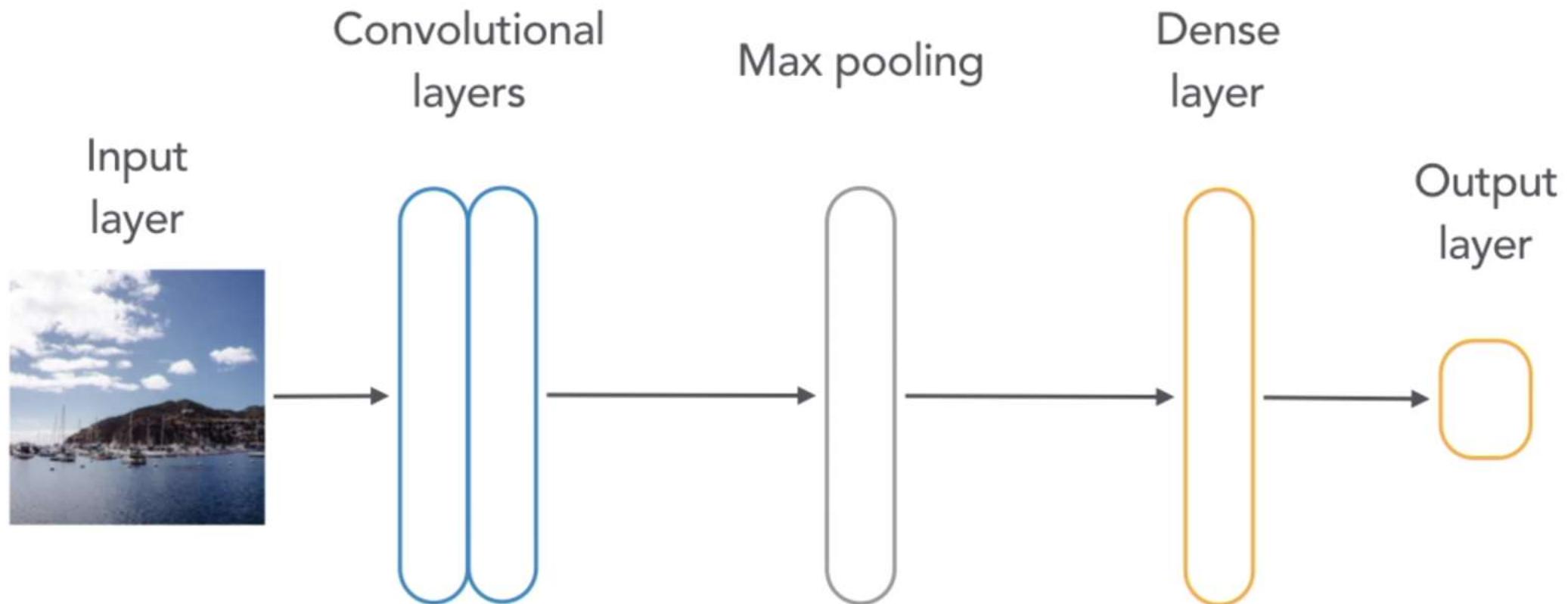
# Convolutional Neural Network



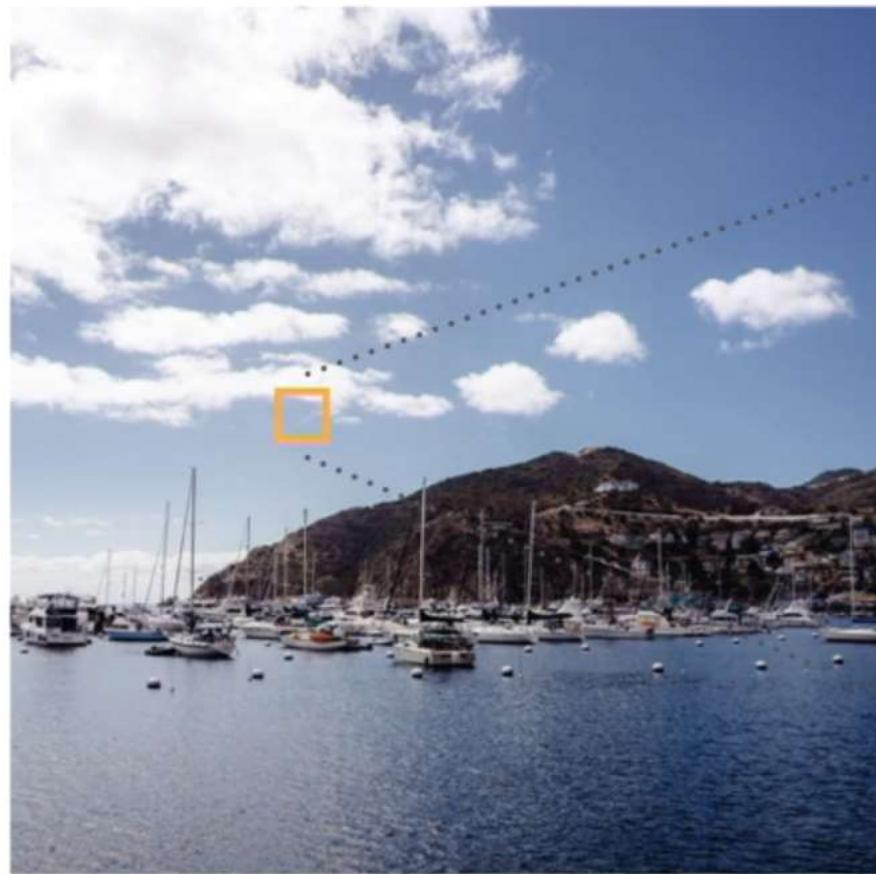
# Convolutional Neural Network



# Convolutional Neural Network

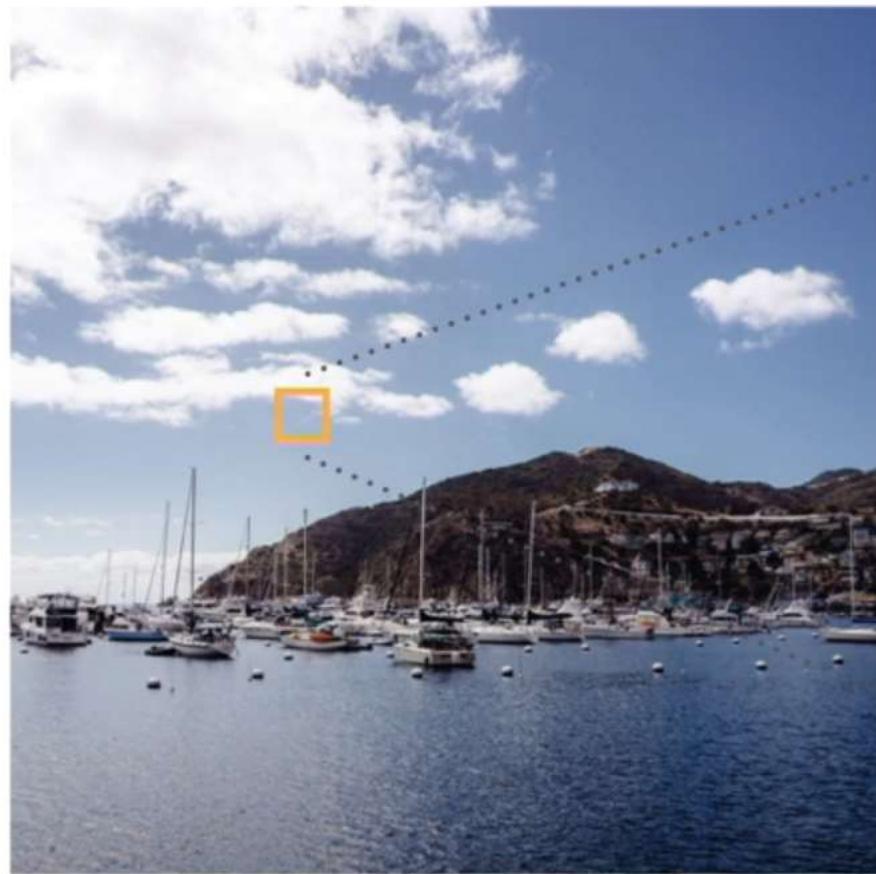


# Max Pooling



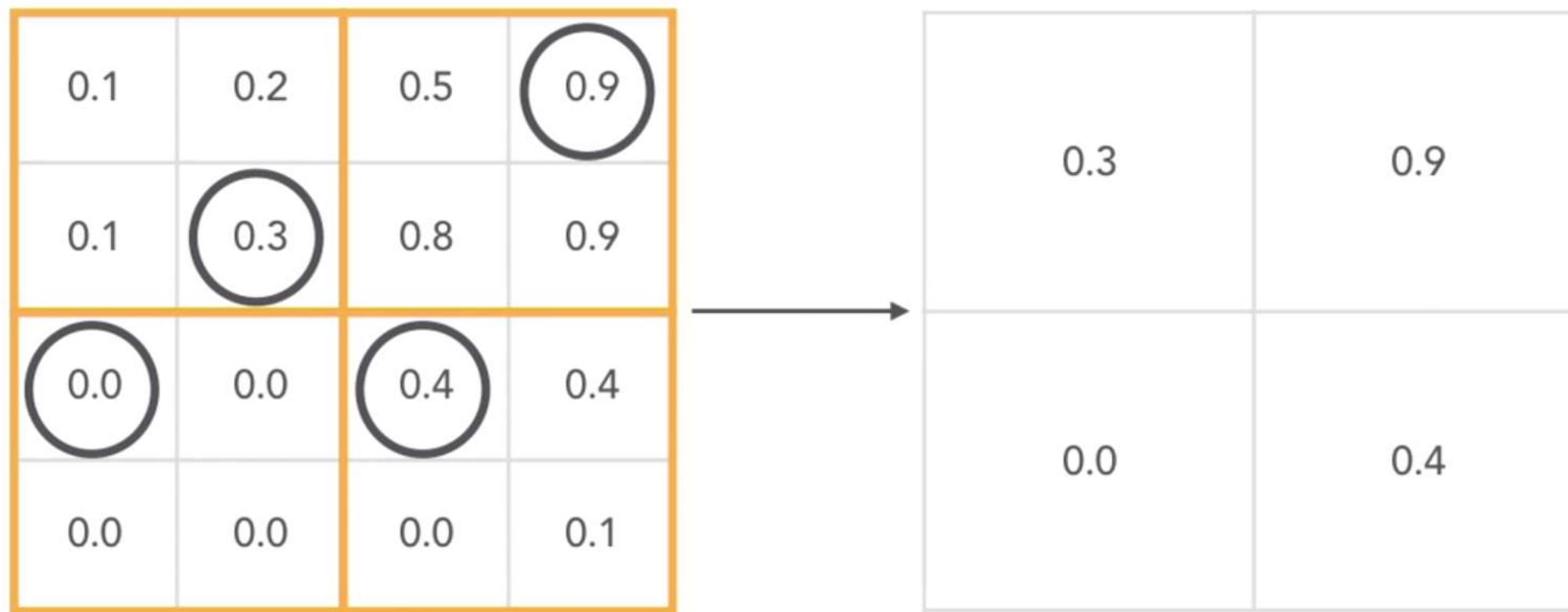
0.1	0.2	0.5	0.9
0.1	0.3	0.8	0.9
0.0	0.0	0.4	0.4
0.0	0.0	0.0	0.1

# Max Pooling

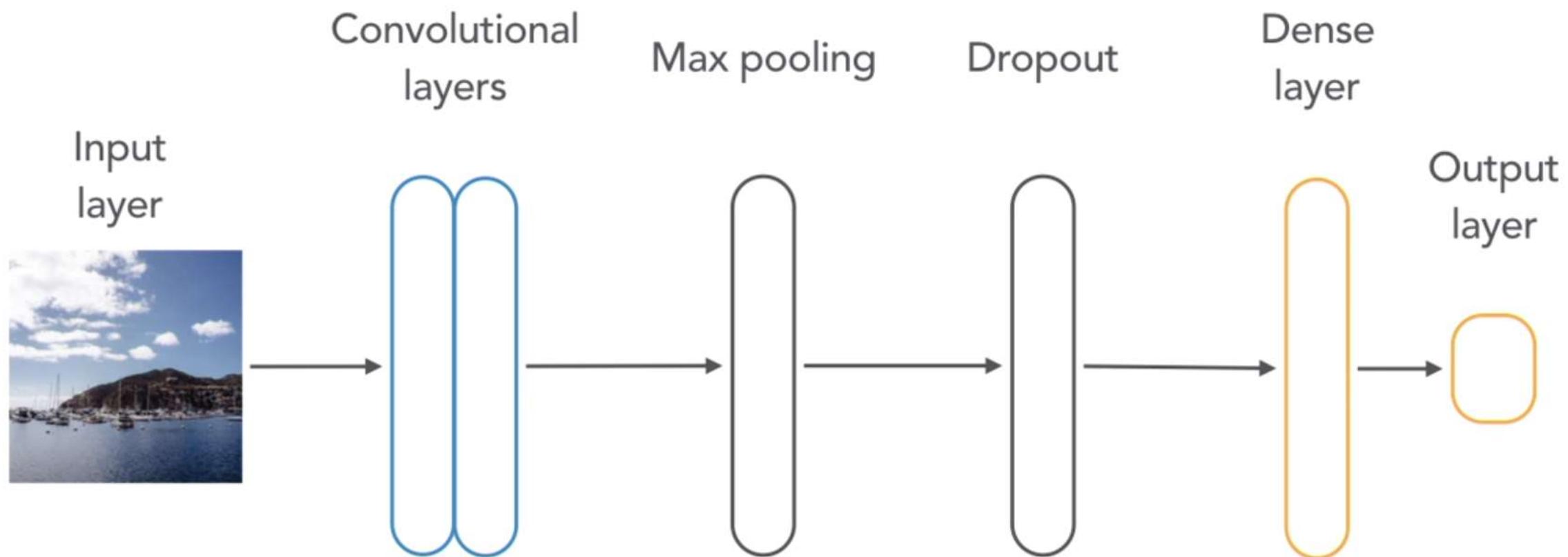


0.1	0.2	0.5	0.9
0.1	0.3	0.8	0.9
0.0	0.0	0.4	0.4
0.0	0.0	0.0	0.1

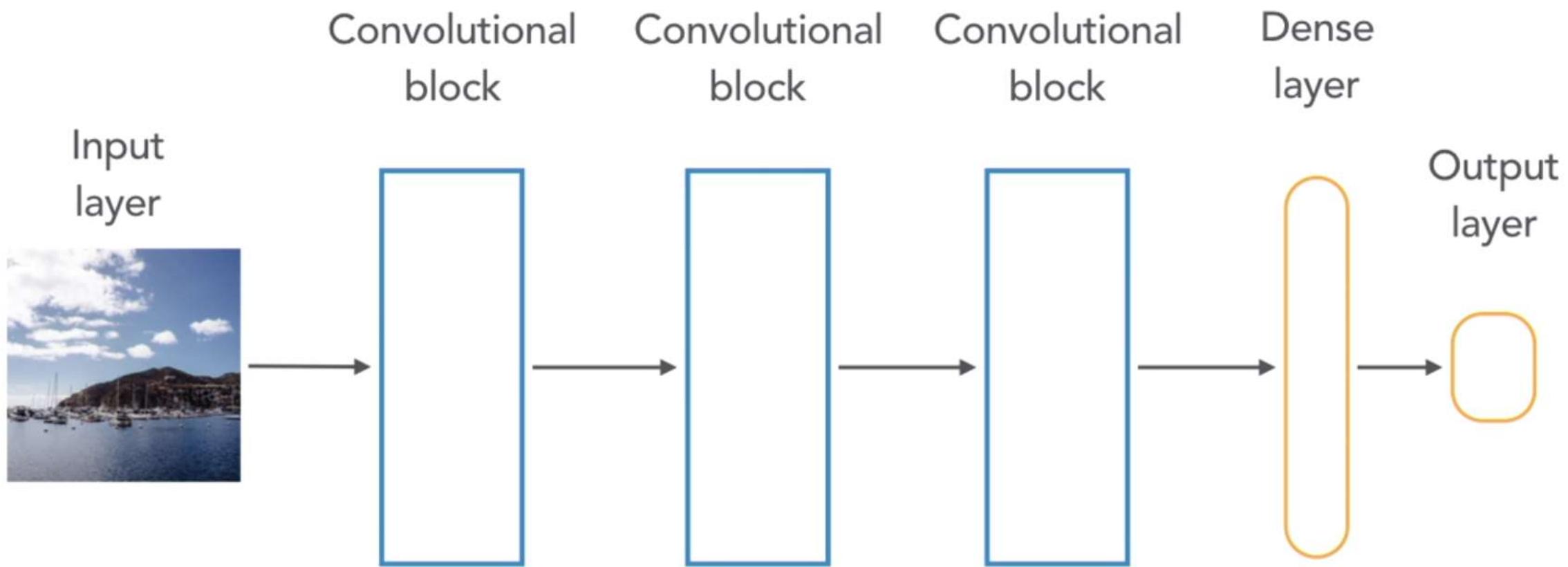
# Max Pooling

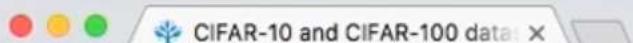


# Convolutional Neural Network



# Convolutional Neural Network





< Back to Alex Krizhevsky's home page

The CIFAR-10 and CIFAR-100 are labeled subsets of the [80 million tiny images](#) dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

## The CIFAR-10 dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:

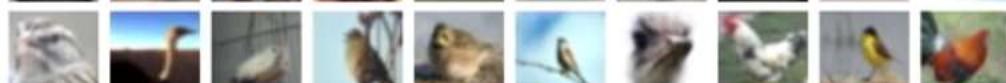
**airplane**



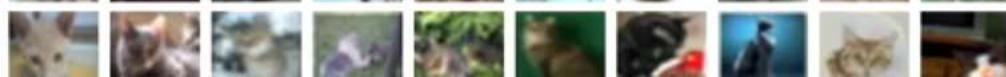
**automobile**



**bird**



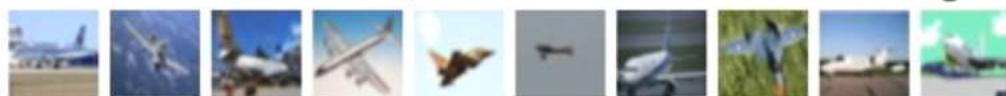
**cat**



 CIFAR-10 and CIFAR-100 data:  Secure | <https://www.cs.toronto.edu/~kriz/cifar.html>

Here are the classes in the dataset, as well as 10 random images from each:

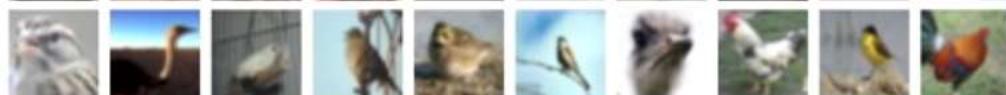
**airplane**



**automobile**



**bird**



**cat**



**deer**



**dog**



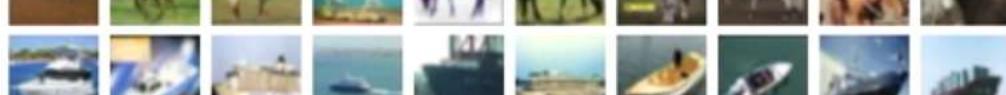
**frog**



**horse**



**ship**



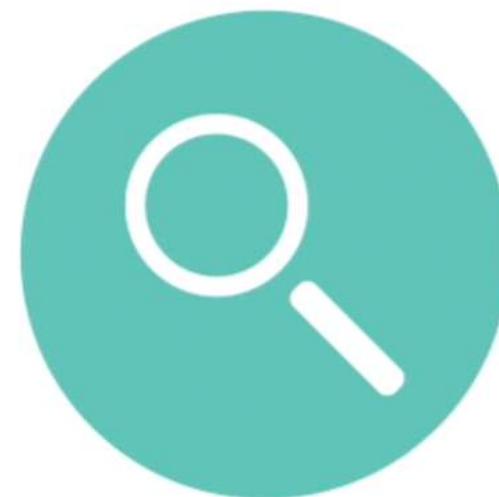
**truck**



The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

# Exploring Your Data Set

- Look through the data by hand
- Check for obvious errors
- Verify that the data makes sense



# CIFAR-10 Specifications

- Images are 32x32 pixels, with 2 color channels
- 10 different types, or classes, of objects
- 60,000 total images
- 5,000 training images and 1,000 test images per class

































