LEAP

AI@PLAKSHA

Introduction to Neural Networks

Day 1; 13th April 2024

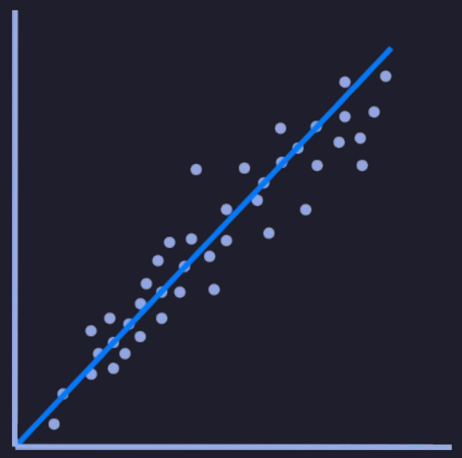
Learning & Exploration in AI Practices

LEAP
AI@PLAKSHA

Regression vs Classification

01

Regression	Classification
Predict continuous values	Predicts discrete values
Linear and Non-Linear Regression	Binary or Multi-Class Classifier
Maps input/s to continuous output variable	Maps input/s to discrete output variable
Linear Regression, etc.	Logistic Regression, etc.
Find best fit line	Find decision boundary



LEAP
AI@PLAKSHA

Loss Functions

02

$$L = \frac{1}{2n}\sum_{i=1}^{n}\left(y_i - y_{predicted_i}\right)^2$$

RMSE Loss

$$L = -(\hat{y} \cdot \log(y) + (1 - \hat{y}) \cdot \log(1 - y))$$

Binary Cross Entropy Loss

Loss functions are **a method of evaluating how well your algorithm models your dataset.**

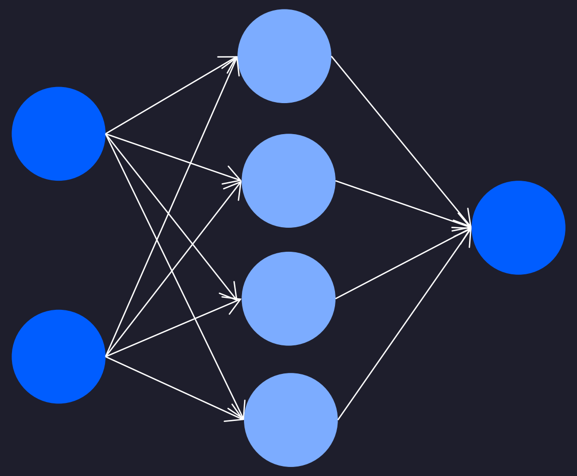
If your predictions are totally off, your loss function will output a higher number. If they're pretty good, it'll output a lower number.

LEAP
AI@PLAKSHA

What is a Neural Network?

Neural network is a type of machine learning model inspired by the human brain. Just like our brains, a neural network is made up of ***neurons***.

Neural networks, and brains both **receive input**, have a **layered structure**, and are formed of simple **computation** units.



Why do we need them?

1 Learns **complex patterns** in data that humans struggle to identify.

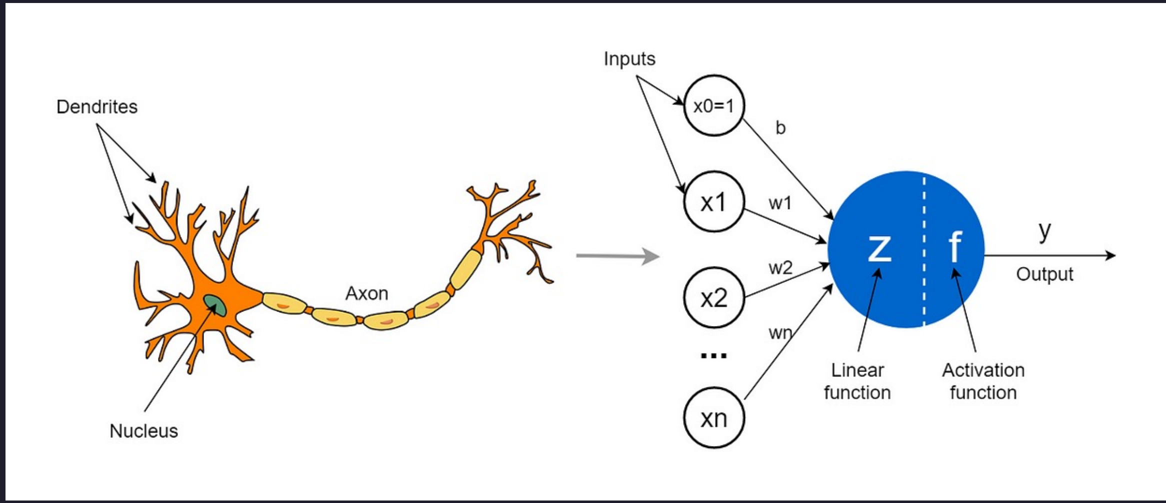
2 **Extracts “hidden” features** from raw data, eliminating the need for us to do so.

3 Multi-dimensionality!

A single perceptron model

Simplest form of a neural network.

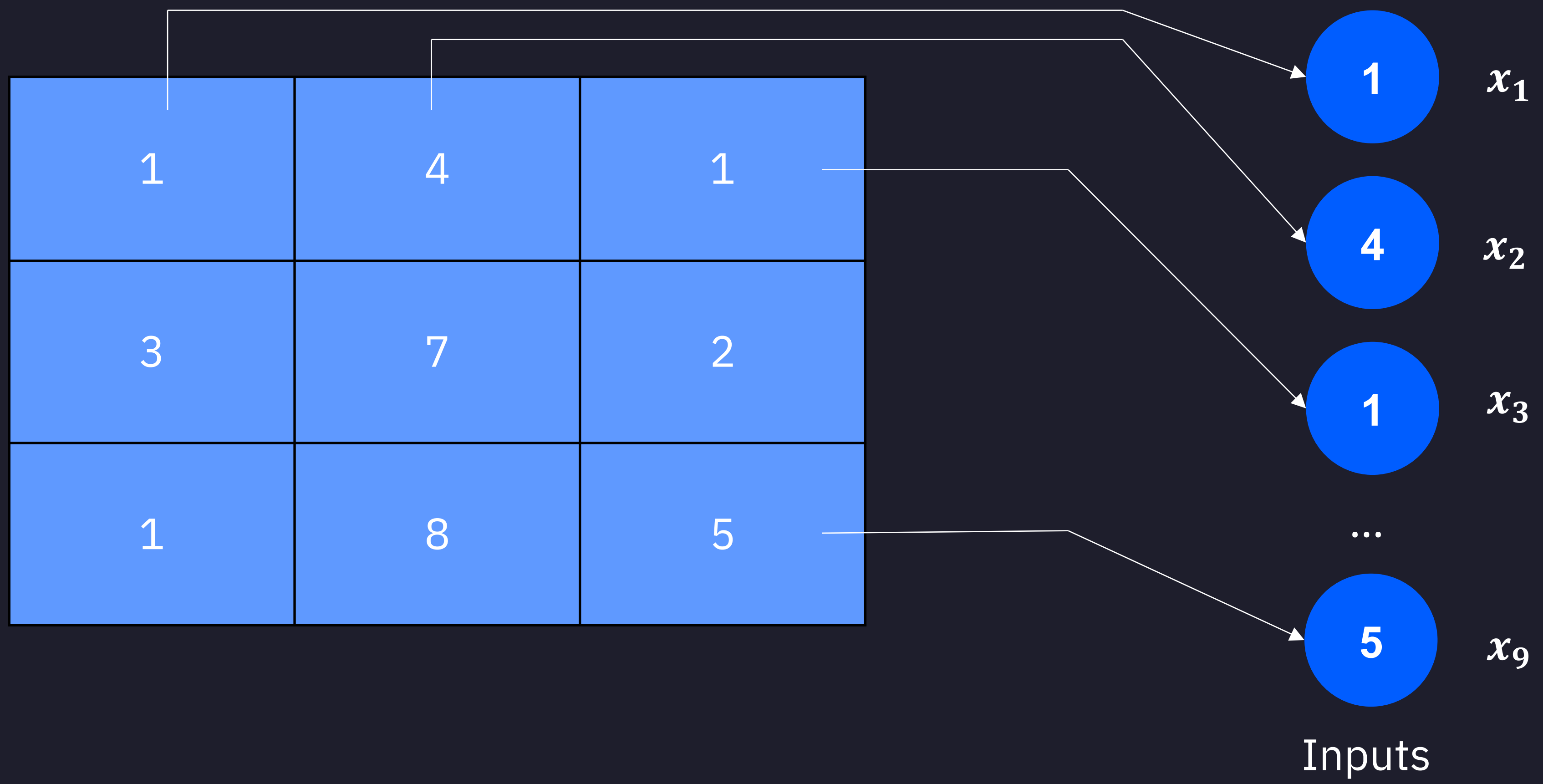
- Consists of a single neuron with adjustable weights.
- The weighted sum of the inputs is applied to the threshold function.
- Finally, we get a binary output classifying inputs into one of the two classes.



Source: The Concept of Artificial Neurons (Perceptrons) in Neural Networks by Rukshan Pramoditha | Towards Data Science

Inputs

06



LEAP
AI@PLAKSHA

Weights & Biases

Weights

Weights are like the "strength" of the connections between the input values and the perceptron.

Each input has a weight associated with it.

Higher weights = more influence
lower weights = less influence.

Bias

The bias is a single number added to the weighted sum of the inputs.

The bias can shift the activation of the perceptron up or down, making it more or less likely to "fire".

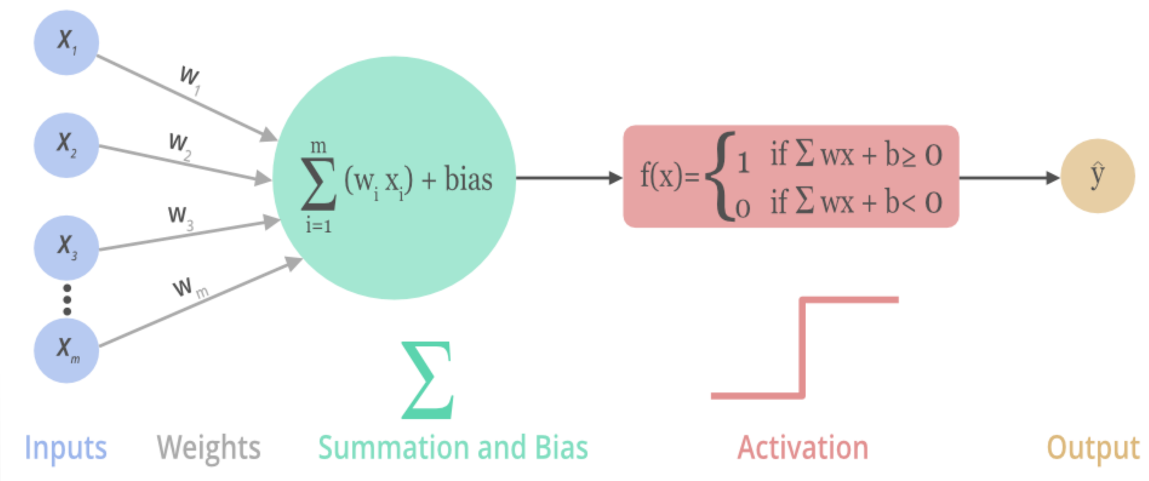
It acts as a threshold.

Activation Function

What?
They decide whether a neuron should be activated or not.

When?
Applied to every linear function within the neural network.

Why?
They help convert inputs from the input space to the output space.



Source: **Understanding Activation Functions in Depth | Geeks for Geeks**

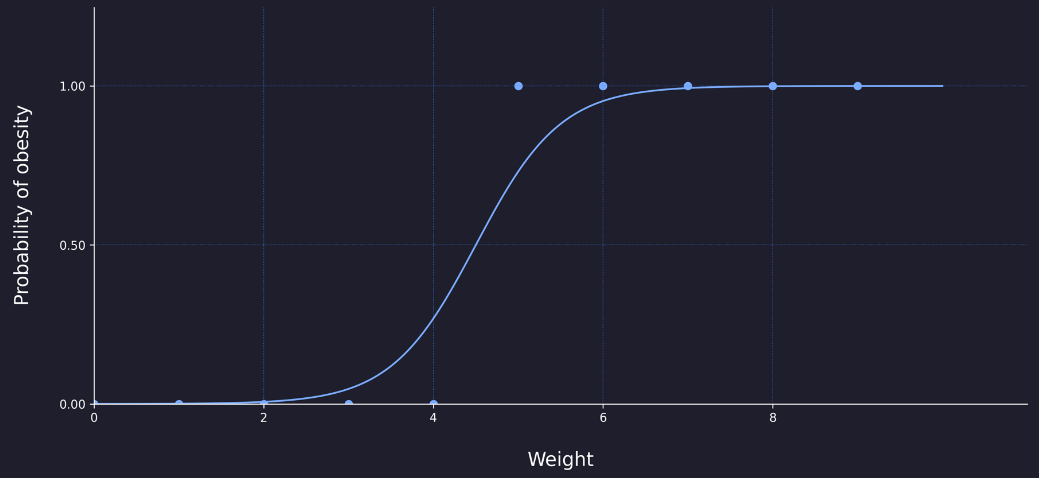
LEAP
AI@PLAKSHA

Common Activation Functions 09

Sigmoid

Squashes input values to a range between 0 and 1.

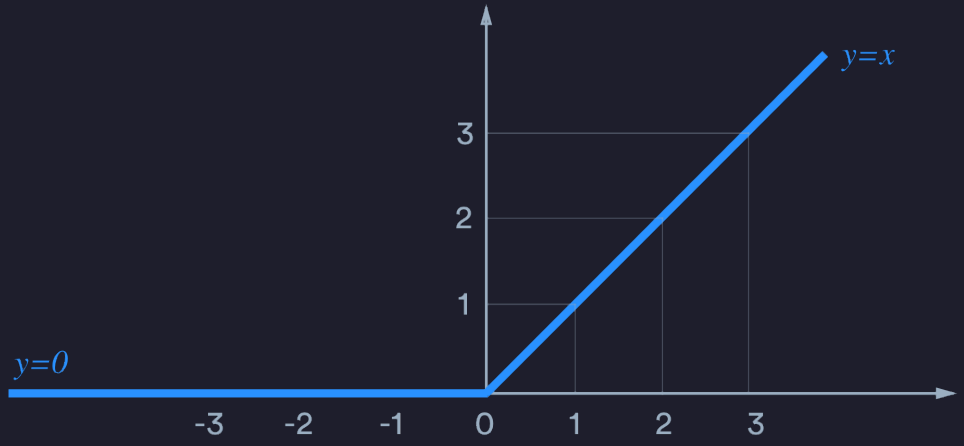
$$f(x) = \frac{1}{1 + e^{-x}}$$



ReLU (Rectified Linear Unit)

Returns the input directly if it is positive; otherwise zero.

$$f(x) = \max(0, x)$$



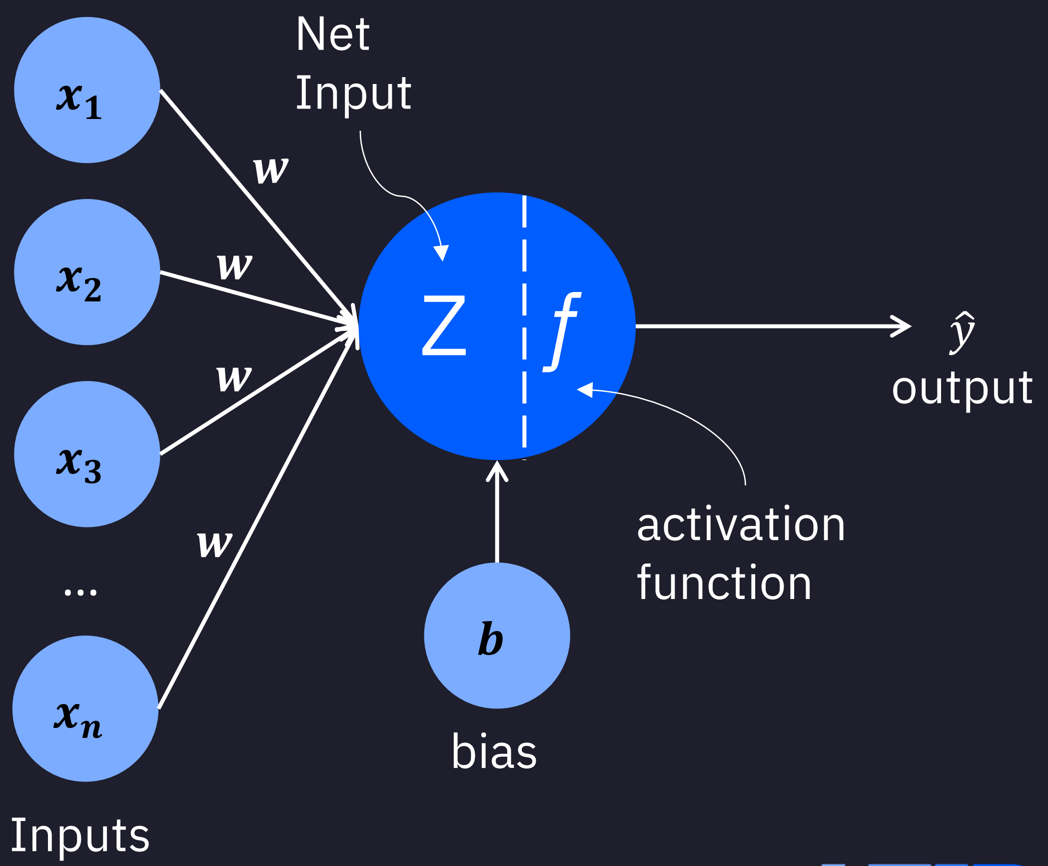
All together.

$$z = x_1 . w_1 + x_2 . w_2 + \cdots + x_n . w_n + b$$
$$= \sum_0^n x_i . w_i + b$$
$$= \boldsymbol{x}^T . \boldsymbol{w} + b$$

$$\hat{y} = \sigma\,(z)$$

$$\hat{y} = \begin{cases} 0, z \leq 0 \\ z, z > 0 \end{cases}$$

$z = Net\ input$
$\sigma = Activation\ function (ReLU)$
$\hat{y} = Predicted\ output$



Loss

$$\hat{y}(p) = \begin{cases} 0, z \le 0 \\ 1, z > 0 \end{cases} \quad \textit{predicted output}$$

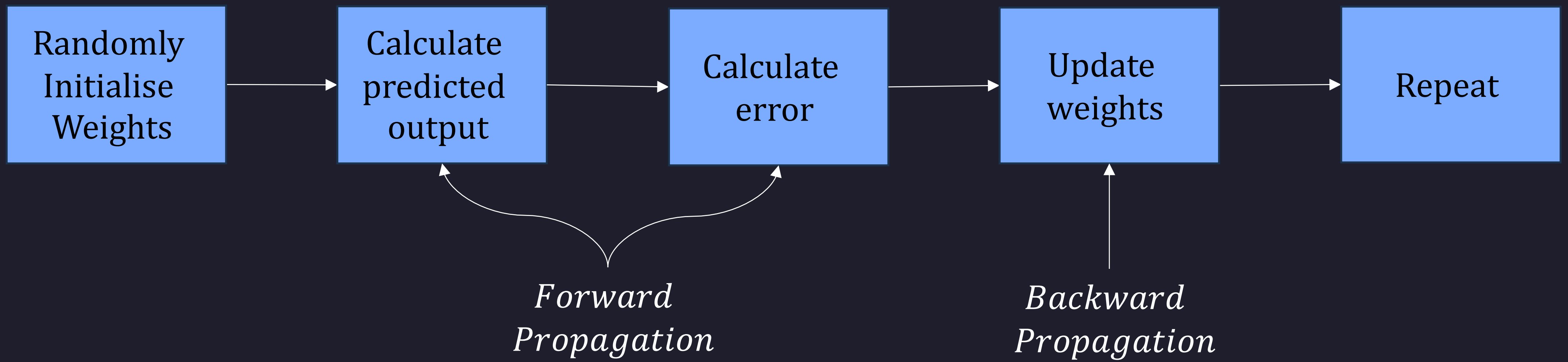
$$y(p) = \textit{desired output}$$

$$e(p) = \hat{y}(p) - y(p)$$
$$= \textit{error aka loss}$$

$$\text{Cost} = \text{J} = \frac{1}{2n}\sum_{i=1}^{n}(e(i))^2$$

$$L = -(\hat{y} \cdot \log(y) + (1 - \hat{y}) \cdot \log(1 - y))$$

Learning

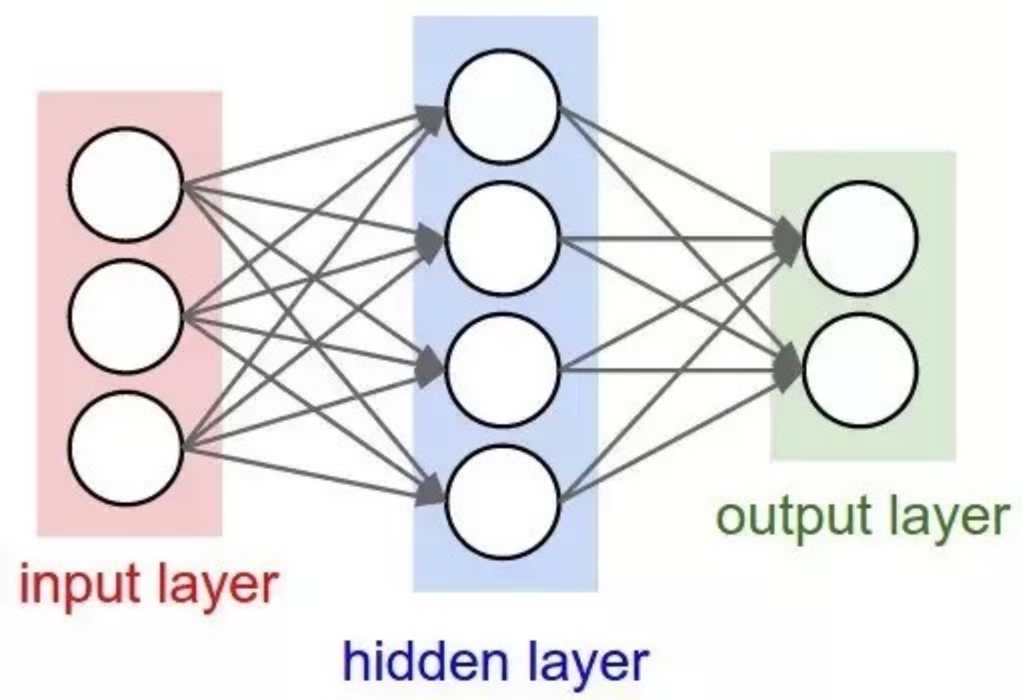


Let's take a look: https://xnought.github.io/backprop-explainer/

An extension: Multi-Layered Neural Networks

Multi-Layered Neural Networks

They not only contain *multiple* neurons, but also *multiple layers*.



Source: https://compphysics.github.io/MachineLearning/doc/pub/cnn/html/cnn.html

Input Layer: This is the input that is fed into the neural network

Hidden Layers: Perform all the necessary calculations.

Output Layer: Predicts out final output.

Fully connected: Every neuron in one layer is connected to every neuron in the next layer and has its own sets of weights and bias.

Number of neurons in each layer 15

Input Layer

- Neurons = input size
- Input = grayscale image of size 28x28 pixels.
- Input layer = 784 neurons (28 x 28).

Hidden Layer

- Layers = you decide
- Neurons = up to you too!
- Complex problems require more hidden layers.

Output Layer

- Neurons = no. of classes
- Input = circle, square or triangle.
- No. of neurons = 3

LEAP
AI@PLAKSHA

16

Inferencing
Forward Propagation

Forward Propagation

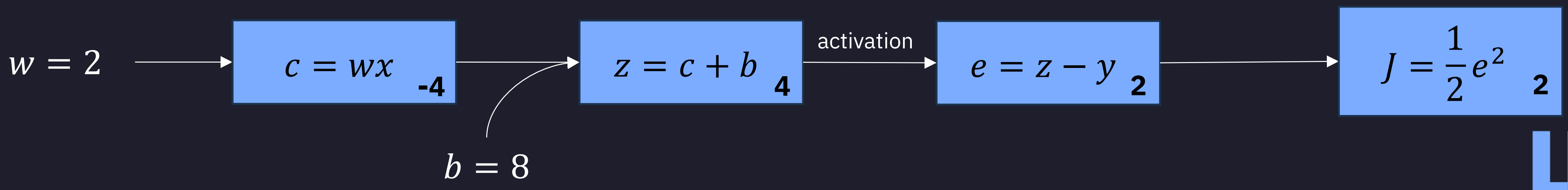
Forward propagation is where input data is fed through a network, in a forward direction, to generate an output. The data is accepted by hidden layers and processed (by the activation functions), and moves to the successive layer.

Let's look at an example of a small neural network...

A Small Neural Network

$$
\begin{aligned}
w &= 2\\
x &= -2\\
b &= 8\\
y &= 2\\
z &= wx + b\\
y &= g(z) = z
\end{aligned}
$$

$$J(w, b) = \frac{1}{2}(y \; - y_{actual})^2$$



Backward Propagation
Computing Gradients

But what is a gradient?

The **gradient** is a fancy word for derivative, or the rate of change of a function. It's a vector (a direction to move) that:

- Points in the direction of greatest increase of a function
- Is zero at a local maximum or local minimum (because there is no single direction of increase)

  So the gradient tells us how sensitive the model is to a change in that weight

Source: https://web.stanford.edu/class/cs224n/

What does the gradient tell us?

Let a cost function $J(w) = w^2$.
Say $w = 3, J(w) = 3^2 = 9$

If we increase w by a tiny amount $\epsilon = 0.001$, how does $J(w)$ change?

$$w = 3 + 0.001 \qquad \text{(If w} \uparrow 0.001\text{)}$$
$$J(w) = w^2 = 9.006001 \qquad (then\ J(w) \uparrow 6 * 0.001)$$

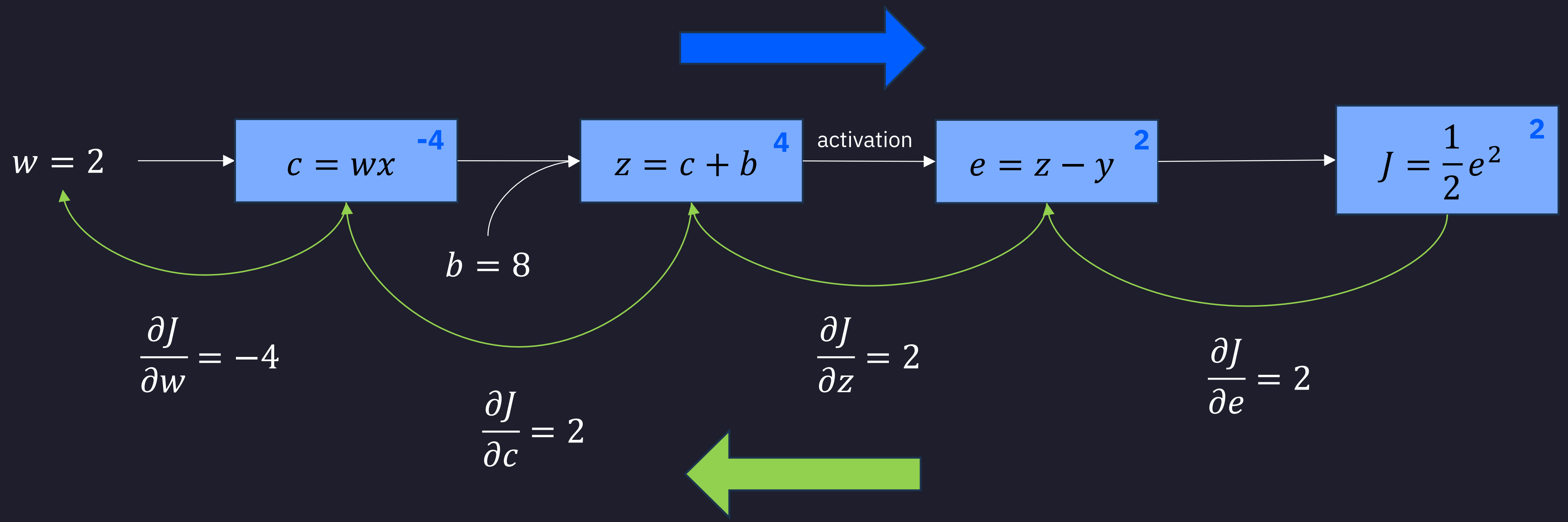
$$\frac{dJ(w)}{dw} = 2w = 6$$

Optimizing the gradient

$$w_{n+1} = w_n - \alpha \frac{\partial J(w, b)}{\partial w}$$

Learning Rate

Back to our small neural network



$w = 2$
$x = -2$
$b = 8$
$y = 2$

The above is using the **chain rule**.

Let's take a look: https://xnought.github.io/backprop-explainer/

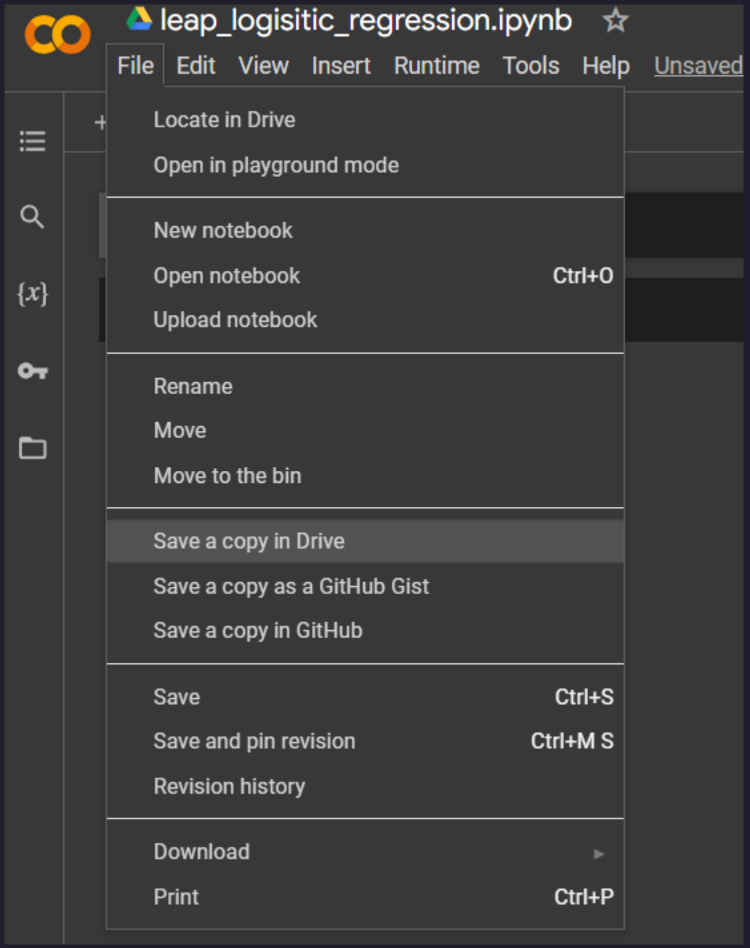
Hyperparameters

Everything "configurable" before training.

- Number of neurons per hidden layer
- Number of hidden layers
- Which algorithm to use for optimizing gradients
- Activation functions for every neuron
- Learning rate
- ...

Let’s make our own!

- Go to leap-ai.tech.
- Click the Colab Notebook for Day 1.
- Go to File->Save a copy in Drive!



The Problem

In a *fully connected neural network*, there is an exponential increase in the number of parameters.

Example:

- Input layer of 784 neurons (28x28 pixel grayscale image)
- Hidden layer also with 784 neurons
- 784 neurons x 784 weights per neuron = 614,656 weights
- Plus 784 biases
- Total parameters: 615,440

All this for just one layer! Oh no!

Now imagine the layer count for **higher resolutions** and **RBG** images.

The Solution?
Convolutional Neural Networks.

It's time to TEST you.

See you tomorrow
for the CNN Project!

LEAP
AI@PLAKSHA

LEAP

AI@PLAKSHA

Convolutional Neural Networks

Day 2; 14th April 2024

Learning & Exploration in AI Practices

LEAP
AI@PLAKSHA

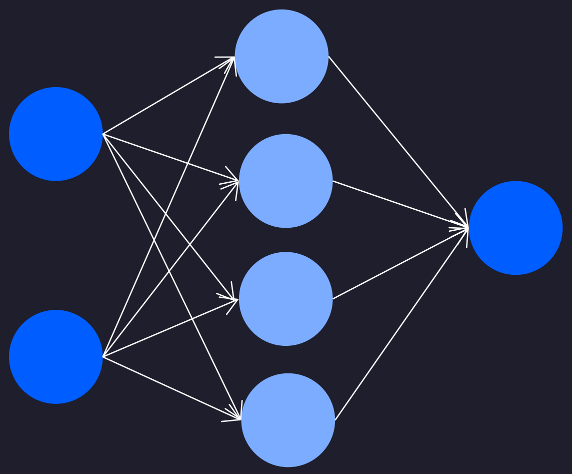
Recap on Neural Networks

01

Randomly Initialize Weights → Calculate Predicted Output → Calculate Loss → Update Weights → Iterate until suitably low loss

Forward Propagation

Backward Propagation



Remember the problem?

02

In a *fully connected neural network*, there is an **exponential increase** in the number of parameters.

A huge number of parameters (600,000+) for only a 28x28 input image! Imagine how this will scale. This leads to **overfitting** due to too many parameters.

The solution was **Convolutional Neural Networks (CNNs)**.

03

Before that,
Let’s meet the Dataset

🌸🌻🌹

LEAP
AI@PLAKSHA

How computers see

Computers C using a Matrix of numbers.

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	30
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

28x28x1

LEAP
AI@PLAKSHA

The dataset we will use.

05





The dataset we will use.

We are using a dataset of Flower Images from Kaggle.

The dataset has about **500 labeled images** for each of **5 classes**: **Daisy**, **Dandelion**, **Rose**, **Sunflower**, and **Tulips**.

Let's explore this **data**.

Why the split?

We need to make a **test** set out of our dataset.

- All available data is not used for training.
- Testing always on *unseen, 'new' data.*
- Can check model performance in the "general" case!

Train	*0.8*	*Test*	*0.2*

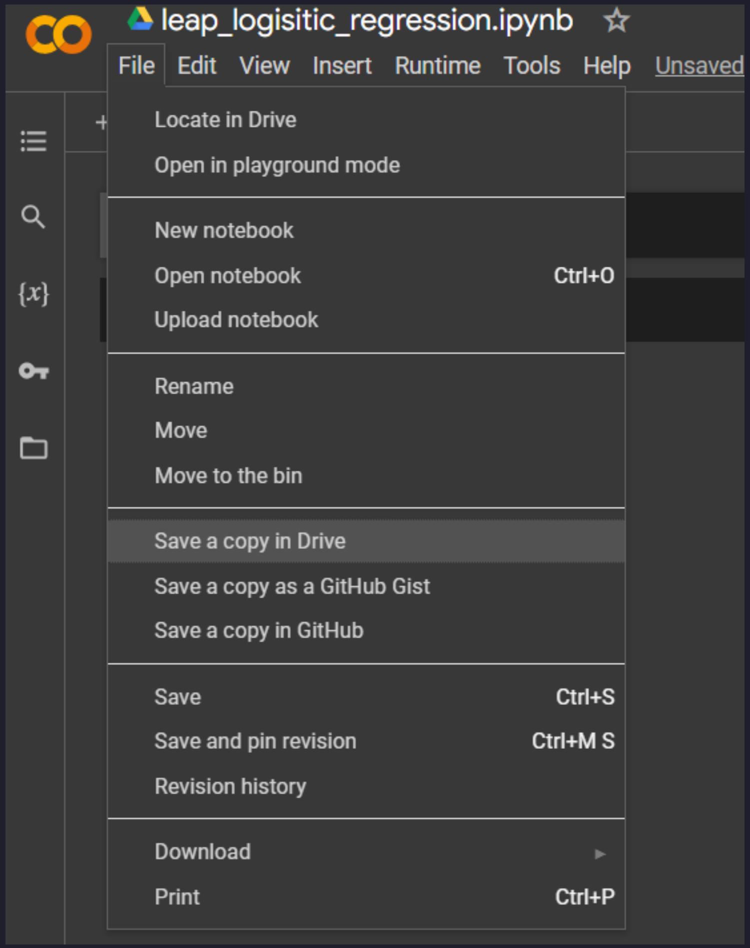
Things to keep in mind.

For Public Datasets, keep in mind:

- **Understand Data**: Analyze dataset structure and check for data imbalance. Explore your dataset!
- **Preprocess**: Remove ***duplicates***, handle ***missing values***, ***outliers***, and ***normalize data*** into one standard format.
- **Ethical Use**: Adhere to ***data policies***, respect ***privacy***, and follow ***ethical guidelines***.

Let’s explore and clean!

- Go to leap-ai.tech.
- Click the Colab Notebook for Day 2.
- Go to File->Save a copy in Drive!



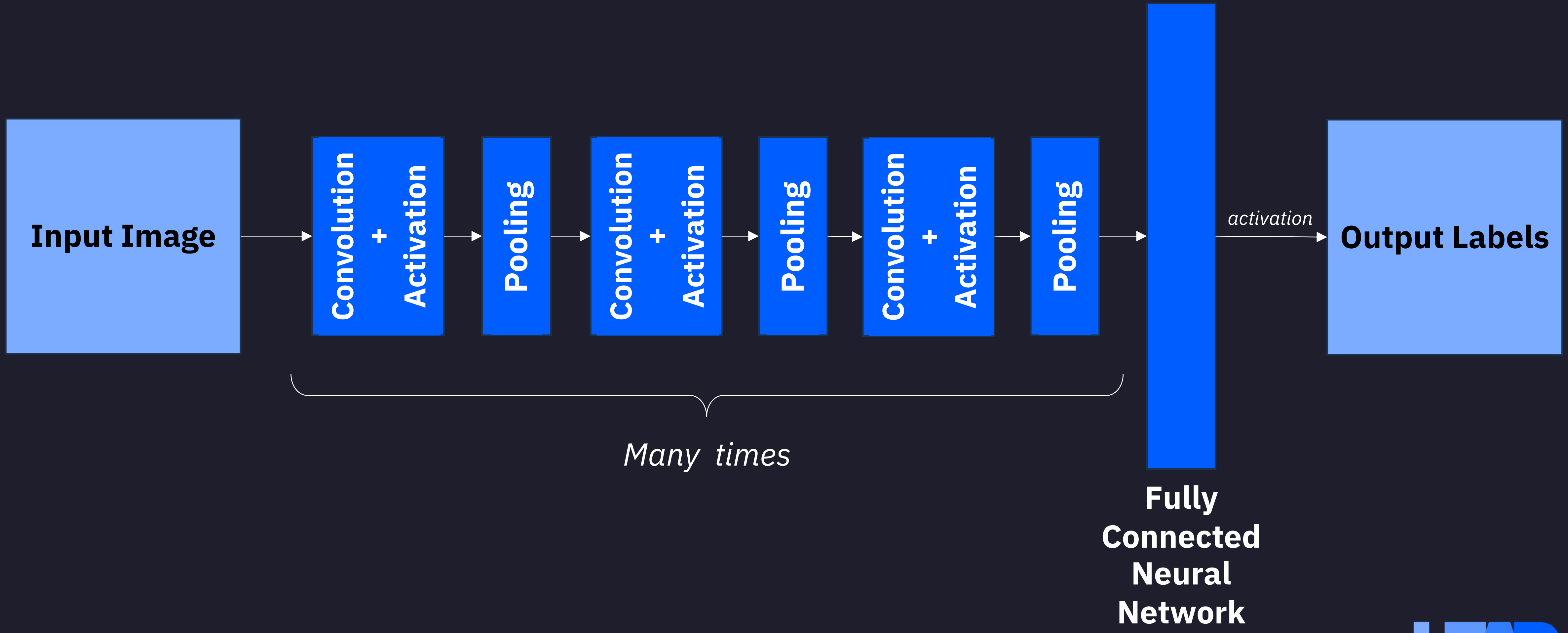
LEAP AI@PLAKSHA

09

CNN

Source: “Chris” X.com/@gtx281

CNNs for Classification: Outline



Part One: Convolution

Source: https://www.youtube.com/watch?v=KuXjwB4LzSA

Convolution is a method to “highlight” features in an image. It “filters” the image and discards unnecessary information.

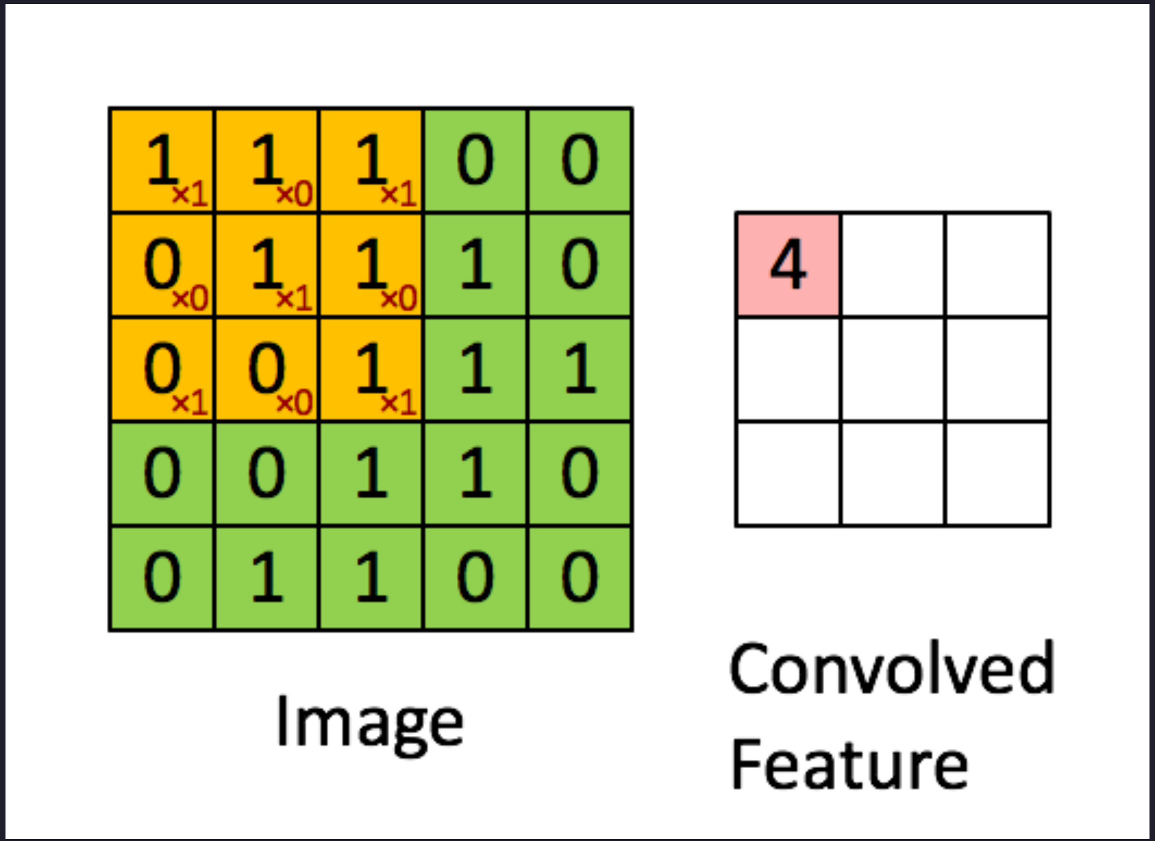
Part One: Convolution

Convolution mathematically combines two matrices (images) using a sliding window (kernel).

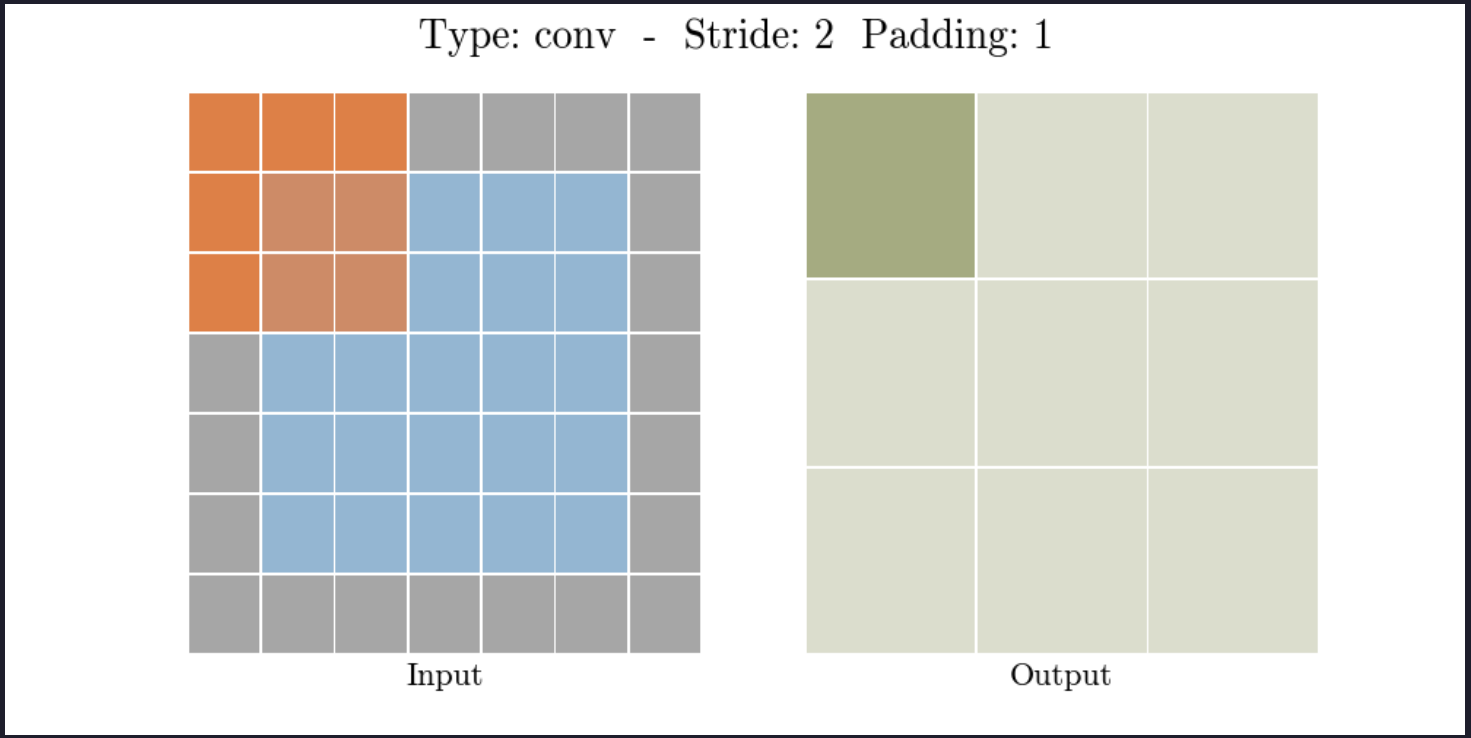
Naturally, you can see how it might need **padding** when it is applied near edges.

Stride can also be used to “skip” a few regions. This makes our output even smaller!

Part One: Convolution



Part One: Convolution



Output Size!

- As a function of input size,

$$Output\ Size = \frac{W\ - F\ - 2P}{S} + 1$$

- **W** is the input size
  F is the filter size
  P is the padding (usually auto-defined!)
  S is the stride
- Also, S = 1 and P = (F – 1)/2 ensure Input Size = Output Size.

This was all in the two dimensions of the image. What about the spatial arrangement?

How are they connected within?

- Each neuron is connected to a **local region** instead of all the neurons in the previous layer. This is a hyperparameter called “filter size.”

  Example: Let input size be **32x32x3**.
  Then, connectivity with **5x5x3** filter size: Each neuron has **75 weights** and of course, 1 bias parameter. Number of neurons will be **32x32x3**.

- **Depth**: The number of filters to use per layer! Changing this will change the output dimension in the depth dimension.

  Example: Let input size be **32x32x3**.
  Now, I have a layer with **depth=12** and **stride=1**. Output size will be 32x32x12! **12288 neurons**

But, the problem was numbers...

Input=32x32x3, depth=12, stride=1, filter=5x5x3 leads to:
12288 neurons, each with 76 parameters... **9,33,888 parameters**

The solution: **Depth Slices**

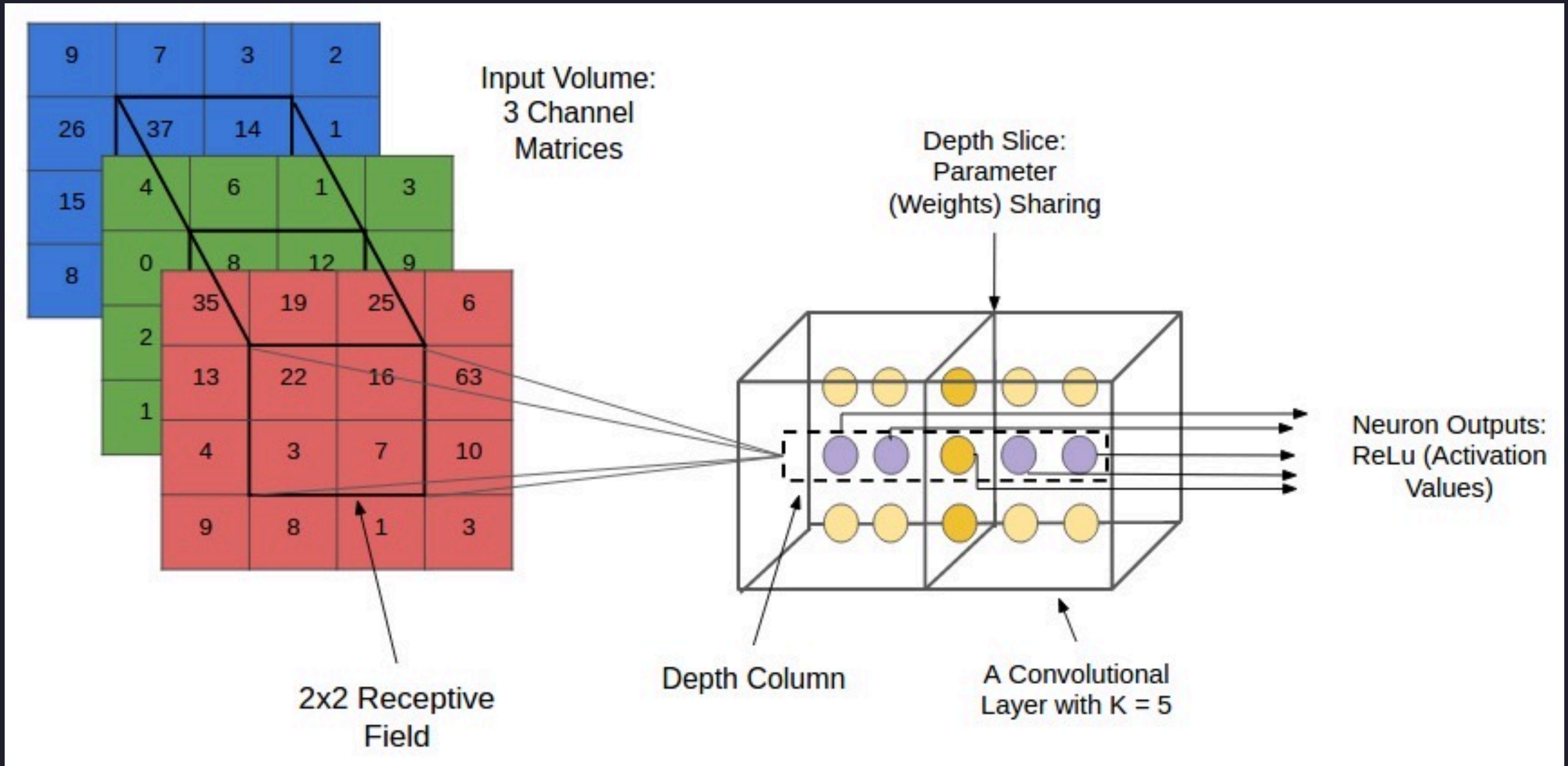
A 32x32x12 output has 12 “depth” slices.
Neurons in the same depth slice use the same weights and biases.

So, with the same parameters above, new shapes:

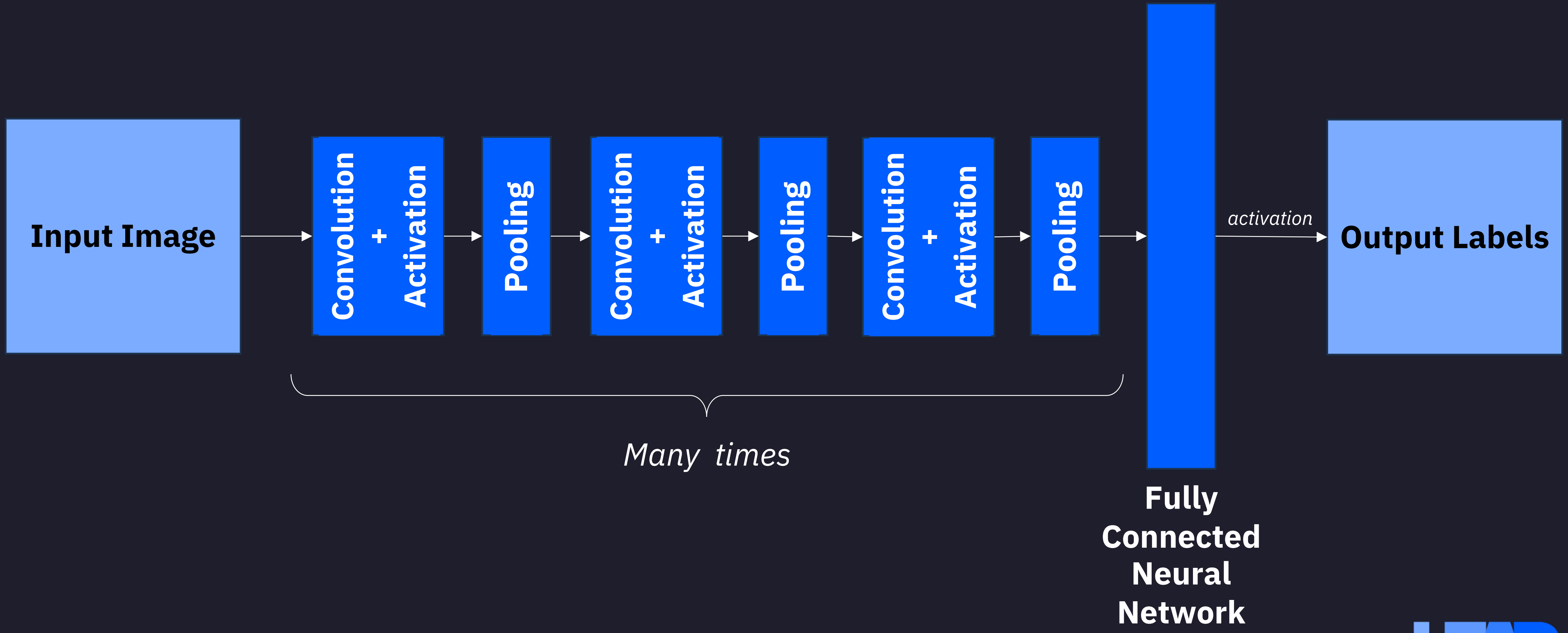
Still 12288 neurons, but parameters are only 5x5x3x12 + 12 = **912**.

Based on the assumption that a feature at (x_1, y_1) is also useful at (x_2, y_2).
Why is this reasonable? When can it fail?

Illustrative Specimen



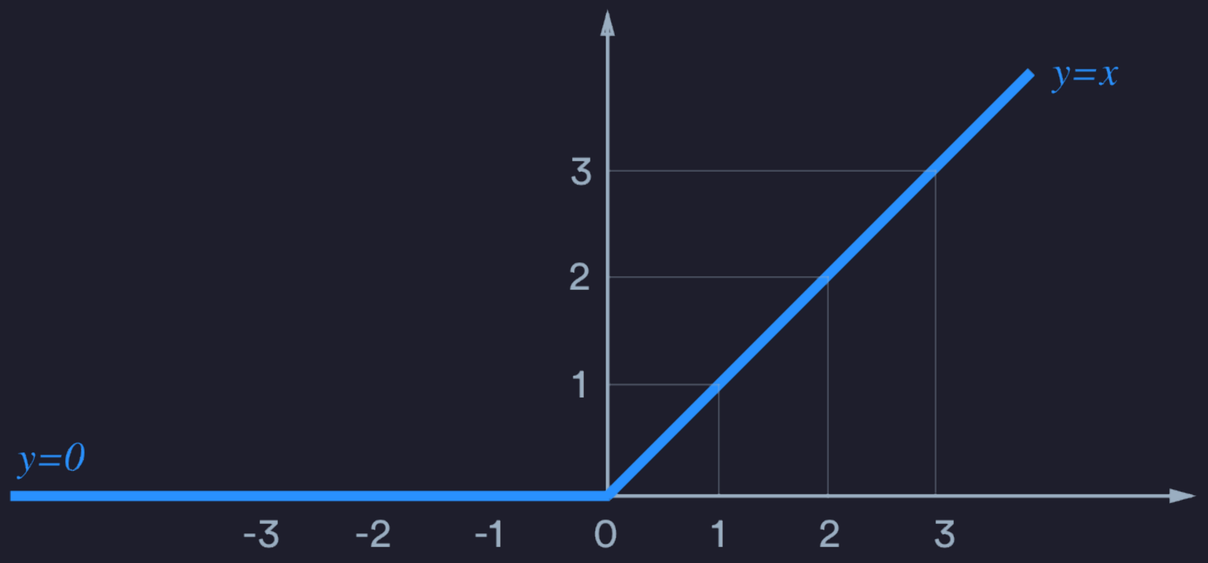
CNNs for Classification: Outline



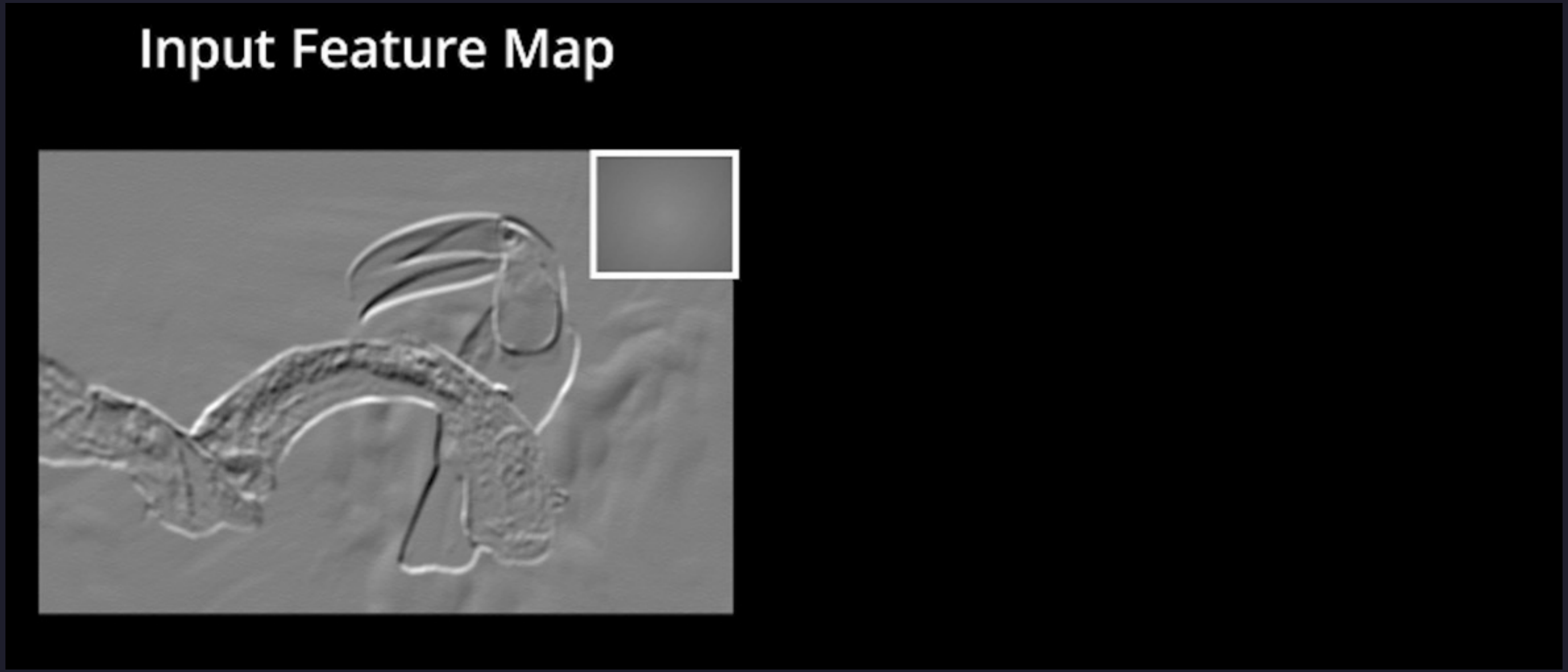
Part Two: Activation Function

After the convolution, the output goes through an activation function. *Do you remember why this was?*

A common function used is **ReLU (Rectified Linear Unit)**.

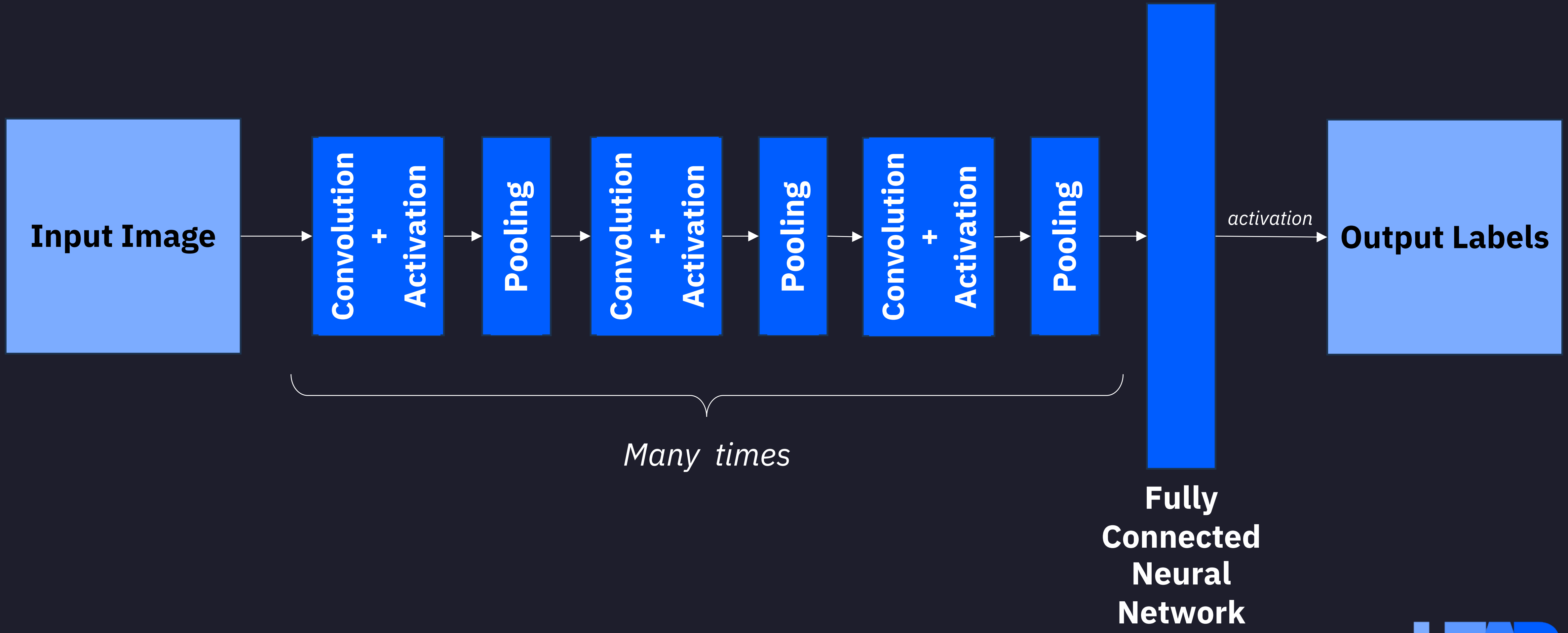


Part Two: Activation Function



An Illustration

CNNs for Classification: Outline



Part Three: Pooling

A method to reduce size of the image to **compress** the data to a smaller scale, while **preserving** the broader and more general patterns created by applying the kernel. Also reduces parameters!

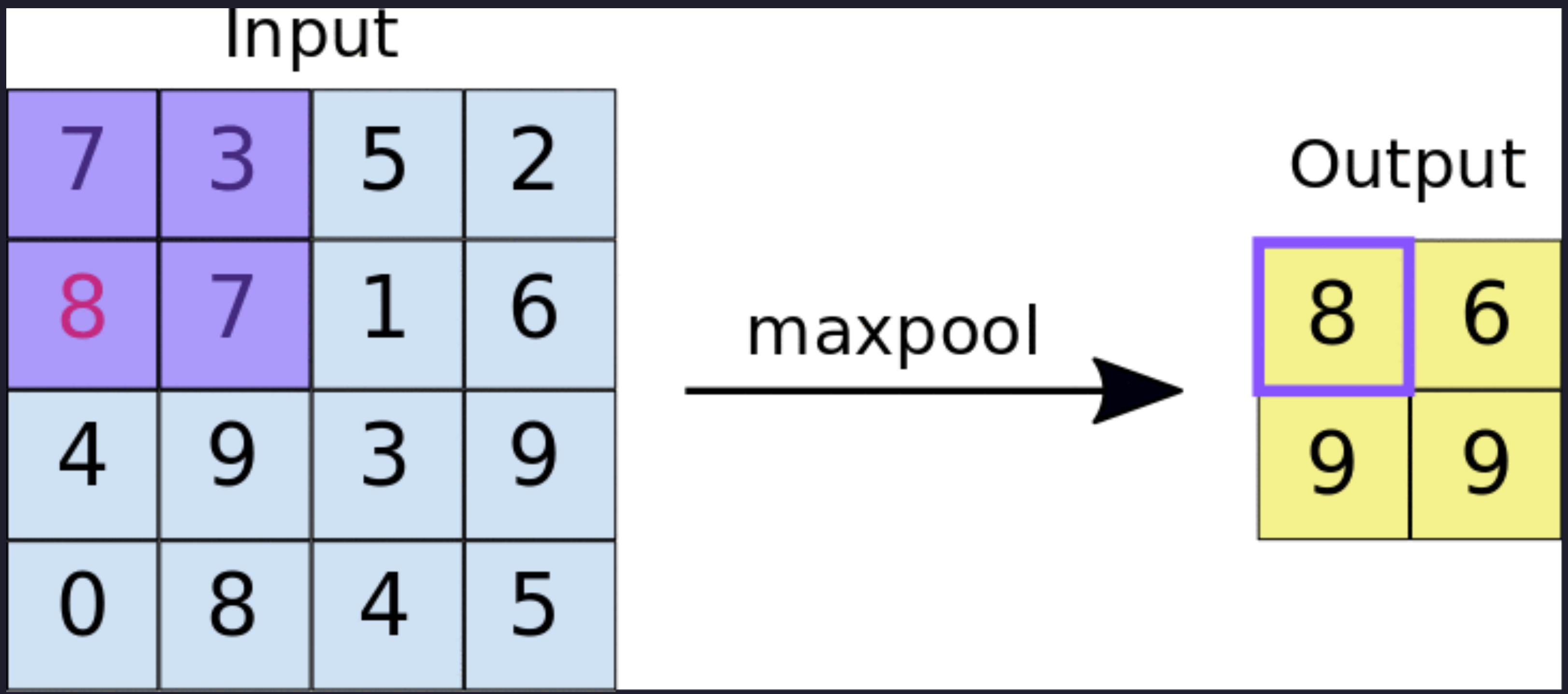
Also works based on a sliding window.

Types: **MeanPool**, **MaxPool**, **SumPool**, etc.

Common: F=3,S=2 and F=2, S=2.

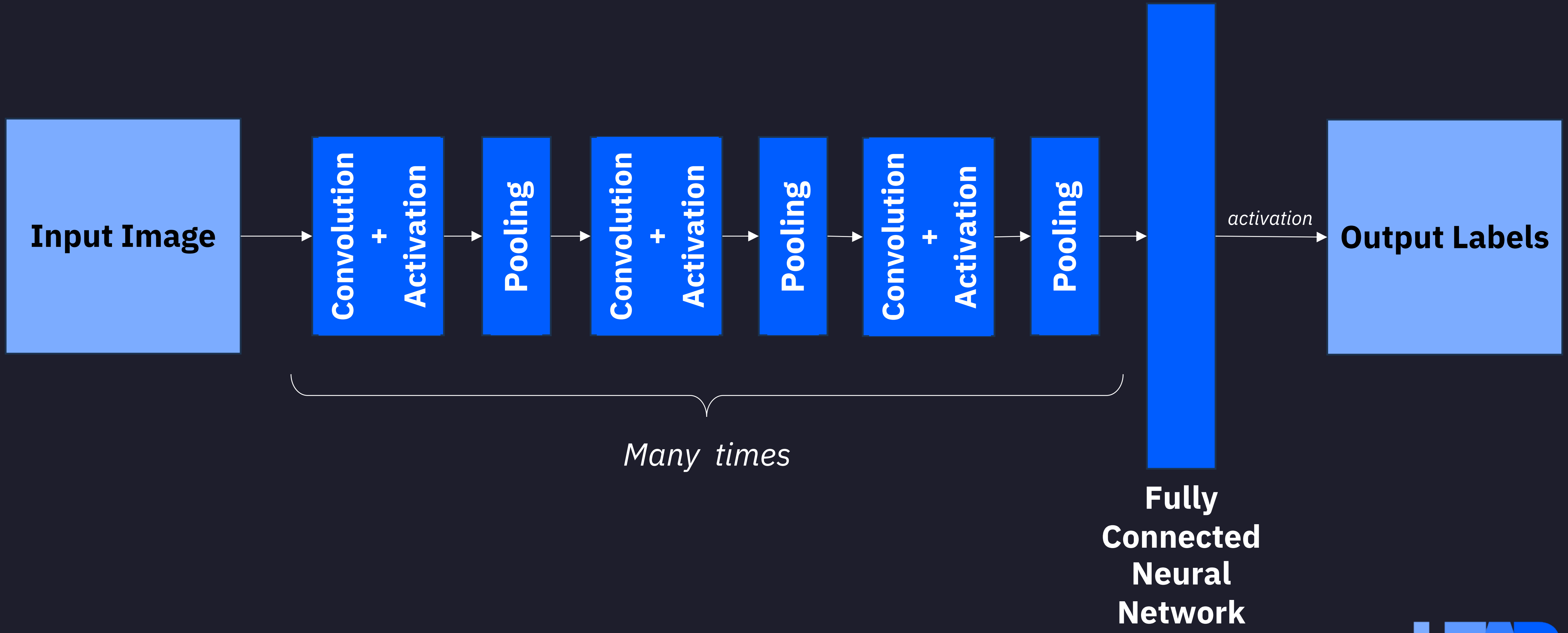
It might be that future CNNs will feature very few to no pooling layers.

An Illustrative



LEAP
AI@PLAKSHA

CNNs for Classification: Outline



Part Four: Fully Connected Network

Once we go through a few Conv + Pooling layer groups, we flatten the image into a **1-Dimensional array**.

It then goes into a Neural Network which can have several layers. This is a **“simple” ML Model** that uses the features we have extracted using convolution as input to give our desired Classification outputs. We saw this yesterday.

Best Practices

- Input layer should be divisible by 2 many times. *Can you think of why?*
- Use small filters for the **Conv** layers. Maximum 5x5! Use a Stride of 1, leaving downsizing to **Pool** layers only!
- F=2, S=2 for the Pooling layers!

Can you see how all the above makes our lives easier?

We really like TESTING you.

All Talk, No Play?

Let’s design our own Convolutional Network.

Testing
(the model this time)

Confusion Matrix

Instances where the model correctly predicts the negative class (0) when the actual class is indeed negative (0).

Instances where the model incorrectly predicts the positive class (1) when the actual class is negative (0). Also known as a Type I error or false alarm.

Predicted Classes

Actual Classes	-ve	+ve
-ve	**TN**	**FP**
+ve	**FN**	**TP**

TN - True Negative
FP - False Positive
FN - False Negative
TP - True Positive

Instances where the model incorrectly predicts the negative class (0) when the actual class is positive (1). Also known as a Type II error or miss.

Instances where the model correctly predicts the positive class (1) when the actual class is indeed positive (1).

Evaluation Metrics

Accuracy is a metric used to evaluate classification models. It measures the ratio of correctly predicted instances to the total instances.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

For example, If you have 100 instances in your dataset and your model correctly predicts 85 of them, the accuracy would be 85%.

LEAP
AI@PLAKSHA

Other Evaluation Metrics

Precision $= \dfrac{\text{Correct “true” predictions}}{\text{All “true” predictions}} = \dfrac{\text{True Positives}}{\text{True Positives+False Positives}}$

Recall (Sensitivity) $= \dfrac{\text{Correct “true” predictions}}{\text{Actually “true” values}} = \dfrac{\text{True positives}}{\text{True Positives+False Negatives}}$

F1 Score = Harmonic Mean(Precision, Recall) $= 2 * \dfrac{\text{Precision X Recall}}{\text{Precision+ Recall}}$

Refer to the post-workshop content for a detailed explanation.

LEAP
AI@PLAKSHA

And we are done!

P.S. Check out the Post Workshop Content!

LEAP

AI@PLAKSHA