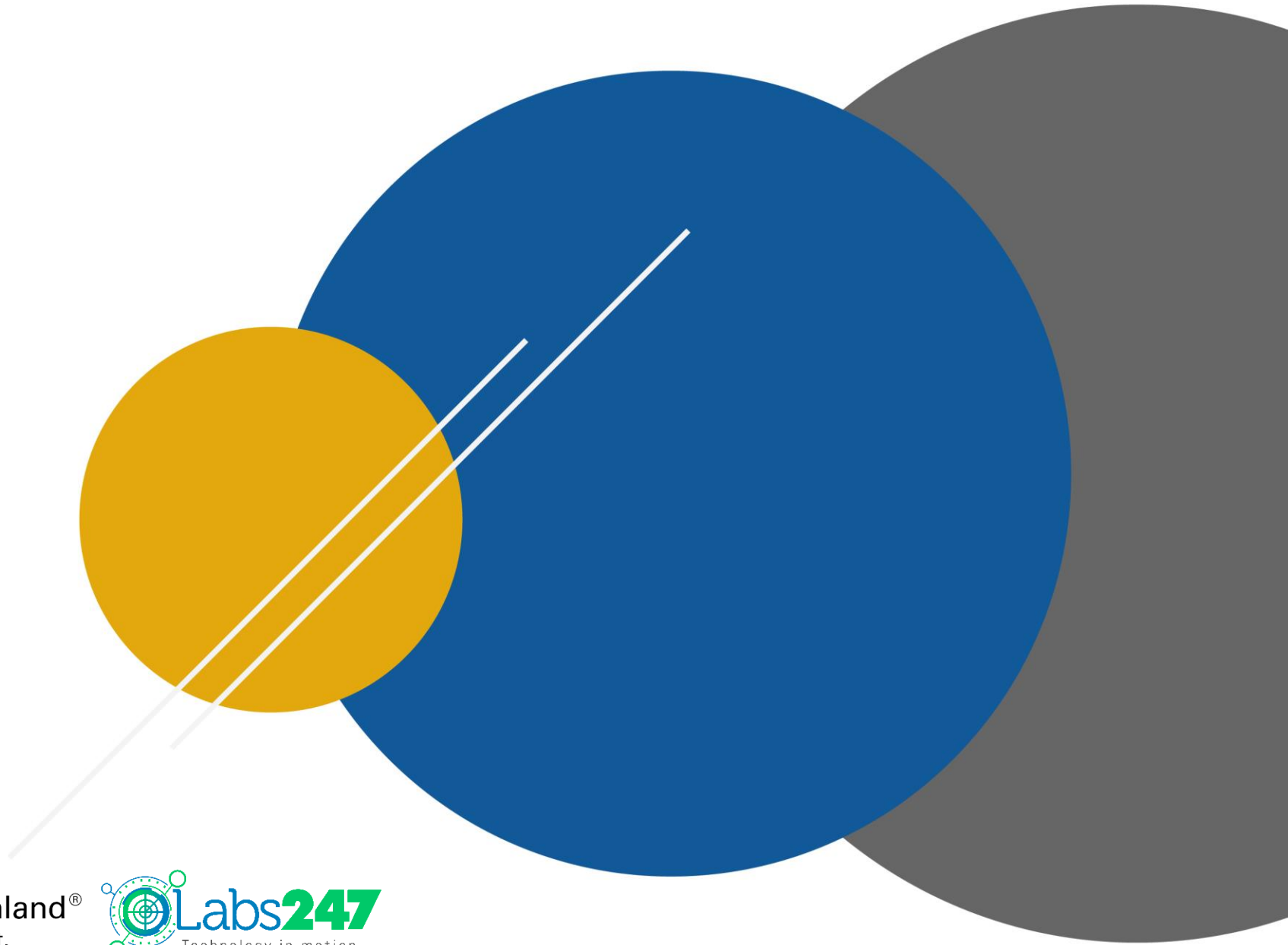


# Deep Learning

## An Introduction



# What You Will Learn

---

- Definition of Deep Learning
- Use case of Deep Learning
- Refreshing basic concept of Neural Network
- Convolution Neural Network Concept
- CNN implementation for image recognition
- Transfer Learning in Convolution Neural Network



# How This Course is Delivered

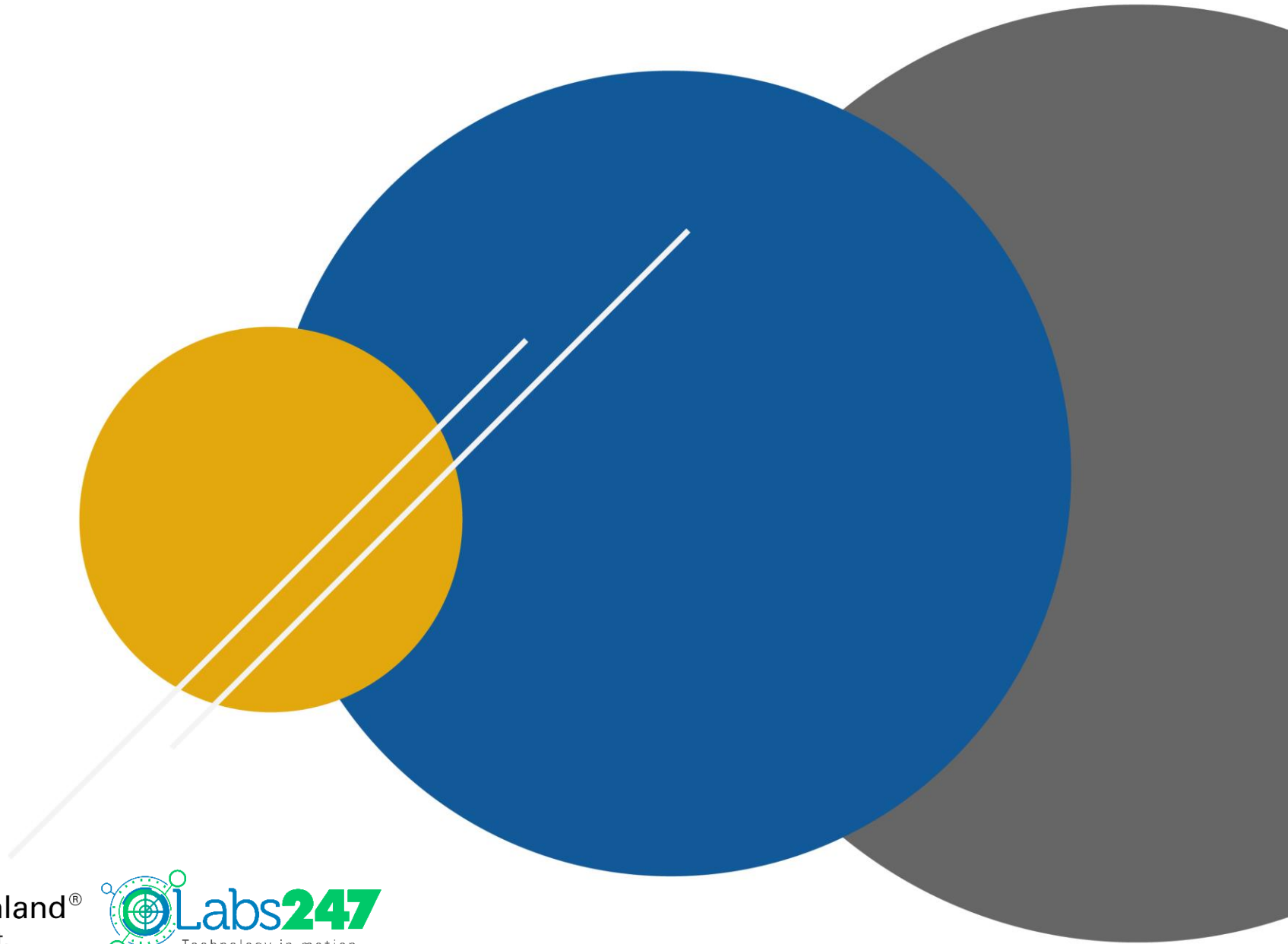
---

- Combining the basic theory by focusing on implementation using Python, Tensorflow and Keras.
- The basic explanation of the theory and algorithm is not explained using in depth mathematical and statistical approach, to make it easier for participants who do not have statistics or mathematics background



# CHAPTER 01

## Introduction

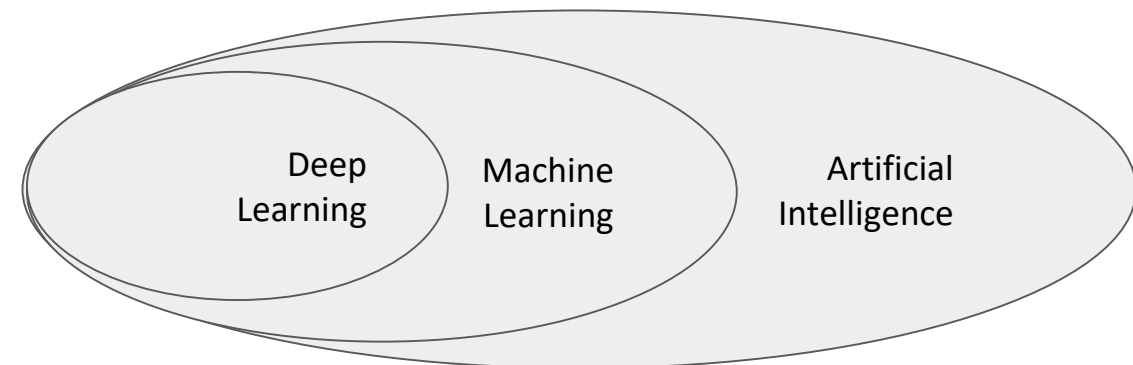


**TÜVRheinland®**  
Precisely Right.



# What is Deep Learning ?

- Machine learning algorithms based on learning multiple levels (i.e deep) of representation/ abstraction (1)
- Learning algorithms derive meaning out of data by using a hierarchy of multiple layers of units (neurons)
- Each neuron/node computes a weighted sum of its inputs and the weighted sum is passed through a nonlinear function, each layer transforms input data in more and more abstract representations
- Learning = find optimal weights from data



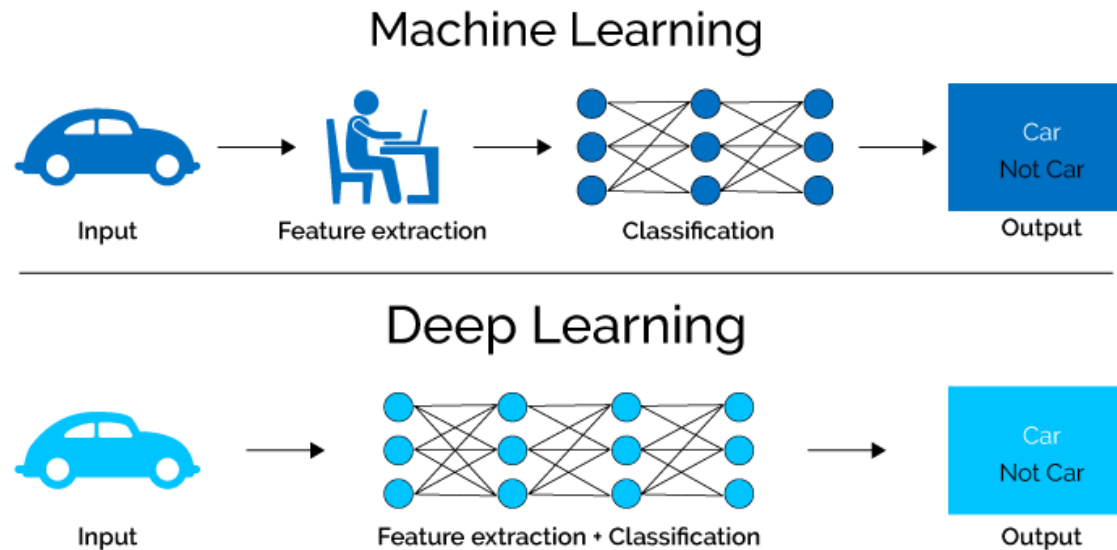
# Why Deep Learning

---

- Manually designed features are often over-specified, incomplete and difficult to design and validate. Learned Features are easy to adapt, fast to learn
- Deep learning provides a very flexible, (almost?) universal, learnable framework for representing world, visual and linguistic information.
- Insufficiently deep architectures can be exponentially inefficient
- Distributed representations are necessary to achieve non-local generalization



# Deep Learning vs Classical Machine Learning



- In classical machine learning, most of the features used require identification of domain experts
- Deep networks scale much better with more data than classical ML algorithms
- Deep learning techniques can be adapted to different domains and applications far more easily than classical ML



# Why Now?

---

- Exponential data growth (and the ability to Process Structured & Unstructured data)
- Faster & open distributed systems (Hadoop, Spark, TensorFlow, ...)
- Faster machines and multicore CPU/GPUs
- New and better models, algorithms, ideas:
  - Better, more flexible learning of intermediate representations
  - Effective end-to-end joint system learning
  - Effective learning methods for using contexts and transferring between tasks

"The analogy to deep learning is that the rocket engine is the deep learning models and the fuel is the huge amounts of data we can feed to these algorithms." - Andrew Ng





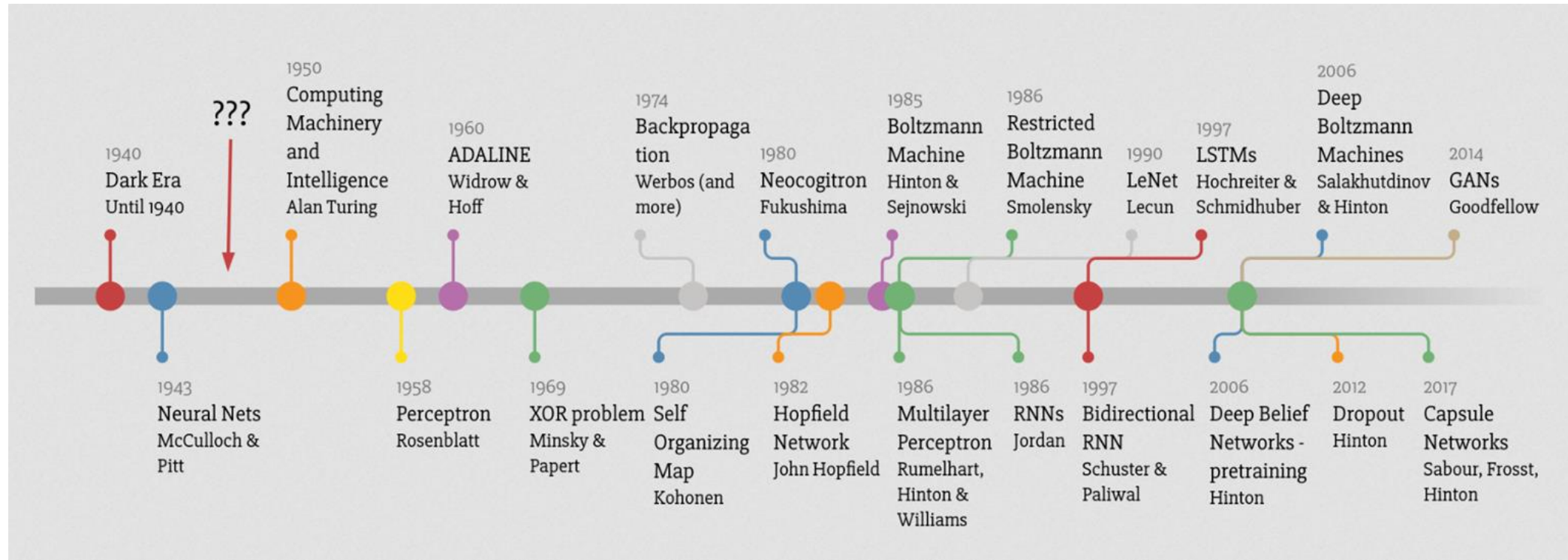
# When To Use Deep Learning ?

---

- If the data size is large
- Need to have high end infrastructure to train in reasonable time
- Complex feature introspection
- Complex problems such as image classification, natural language processing, and speech recognition



# Deep Learning Timeline



# Use Case

## Self-Driving Cars

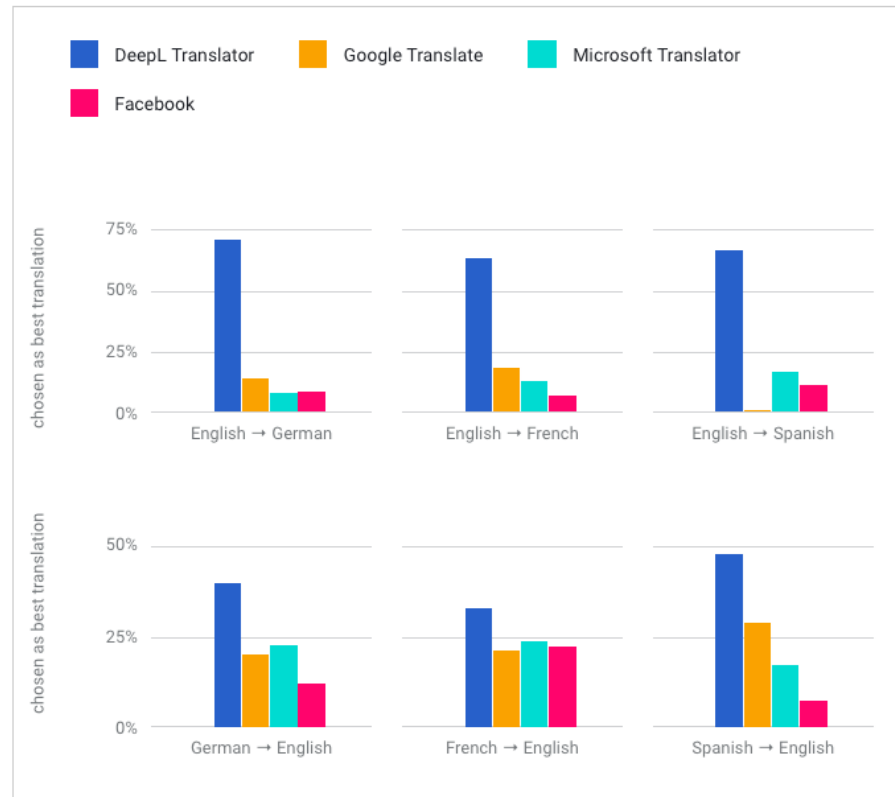


- Uber first announced its intentions to amass a fleet of automatic cars in February 2015
- The cars themselves were packed with around 20 cameras, seven lasers, a GPS, radar and lidar, a technology that measures the distance reached by outgoing lasers so cars can “see” and interpret the action around them.
- In March 2018, an Uber self-driving car in Tempe, Arizona struck a pedestrian who was walking outside of a crosswalk at night.
- Waymo—formerly the Google self-driving car project—paid driverless taxi service could launch in December 2018



# Use Case

## Machine Translation



- DeepL Translator is a translation service launched in August 2017 by DeepL GmbH
- Promising to deliver 3x better results compared to Microsoft Translator and Google Translate



# Use Case

## Colorization Images



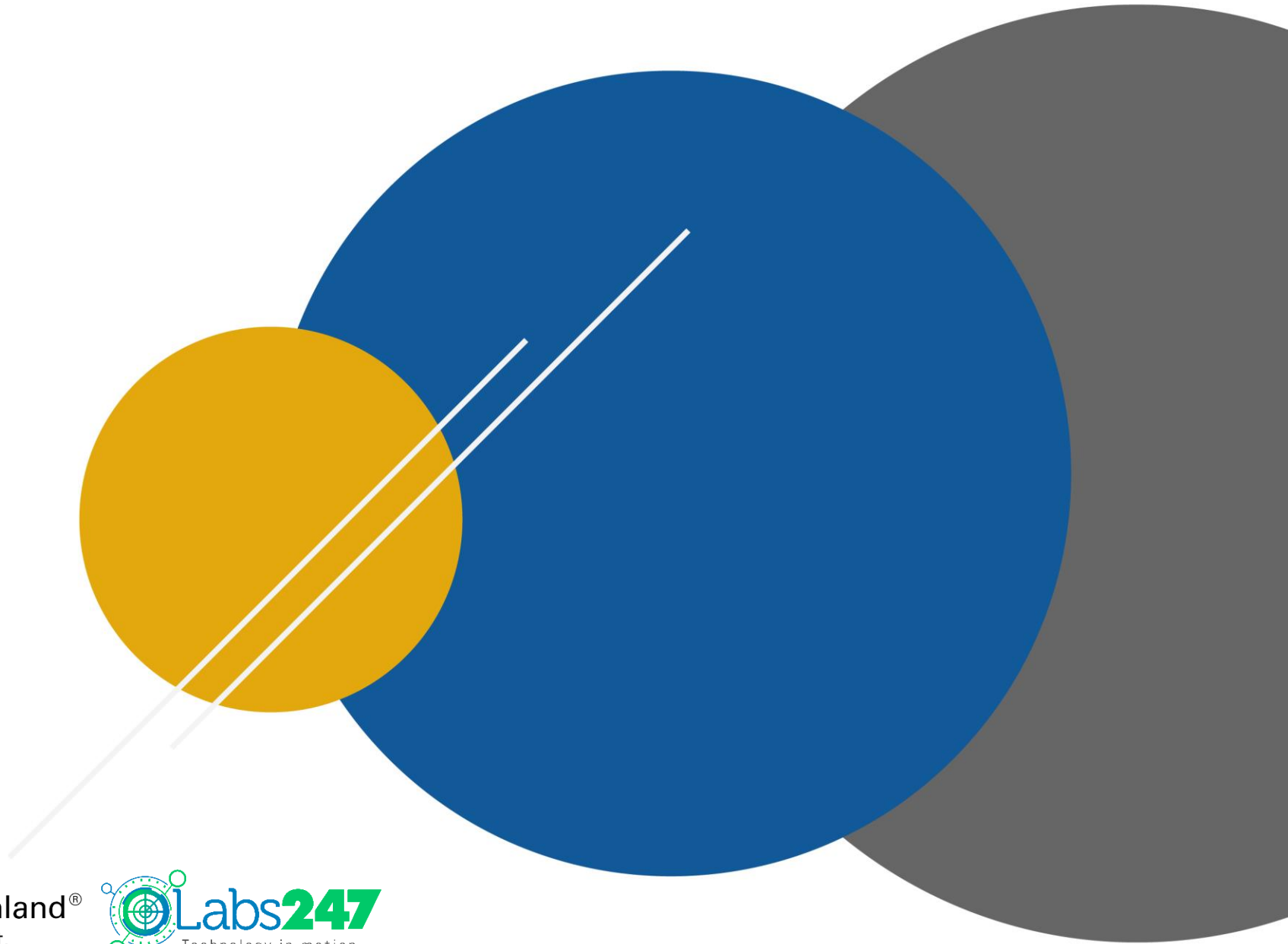
- Paper : ColorUNet: A Convolutional Classification Approach to Colorization
- A final project of Computer Vision courses at Stanford University
- Using Convolution Neural Network method classification to colorize grayscale images.



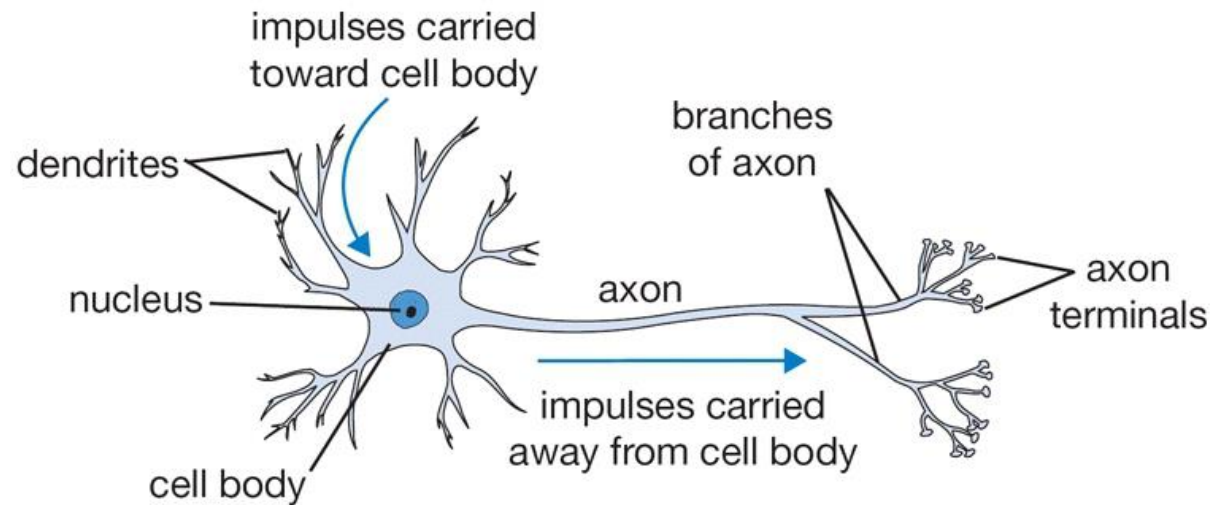


# CHAPTER 02

## Artificial Neural Network



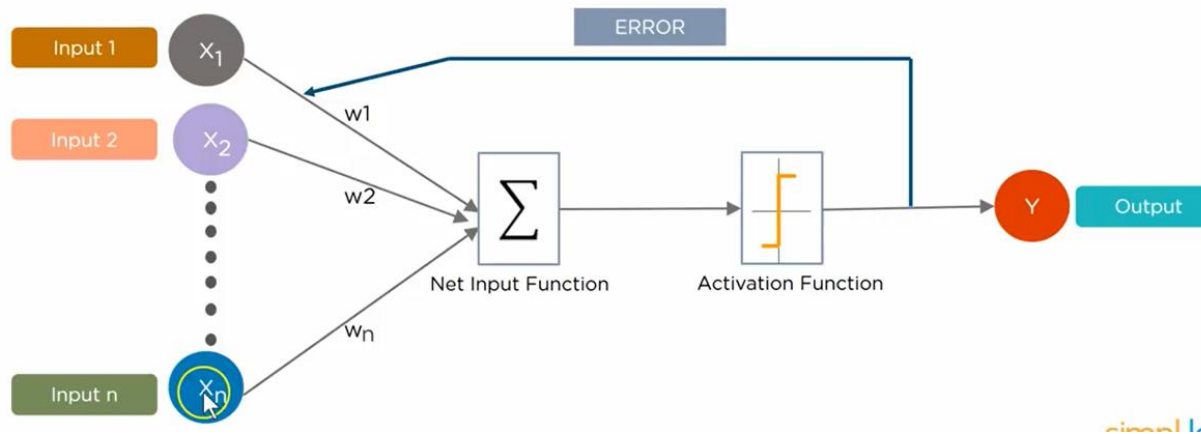
# Anatomy of A Neuron







- Dendrite receives signal from other nucleus
- Cell Body sums all the inputs
- Axon pass message away from cell body to other neuron



# Component Of Artificial Neural Networks



-  Input
-  Weight
-  Bias
-  Activation Function

$$\text{Net input} = x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 = \sum x_i \cdot w_i$$

Or using simple matrix multiplication

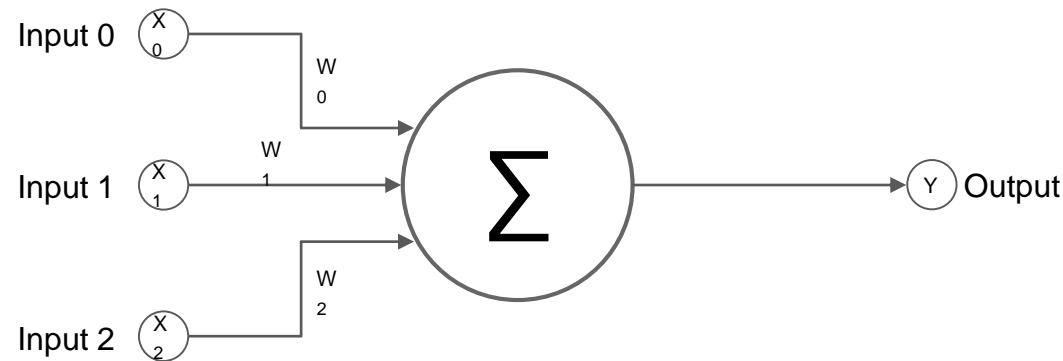
$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3$$





# How It Work

## Perceptron



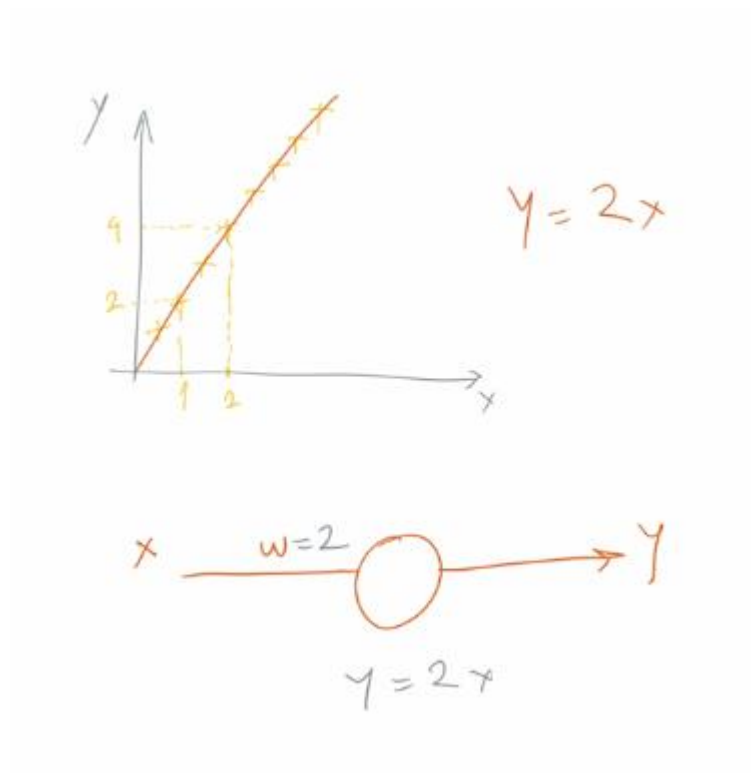
$$\begin{aligned}
 y &= (x_0 w_0) + (x_1 w_1) + (x_2 w_2) \\
 &= \sum_i x_i w_i \\
 &= [w_0 \ w_1 \ w_2] \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}
 \end{aligned}$$

- Perceptron is the basic part of a neural network, which represents a single neuron
- A neuron is a computational unit that calculates a piece of information based on weighted input parameters
- Inputs accepted by the neuron are separately weighted.
- Weights are real numbers expressing the importance of the respective inputs to the output and can be adjusted during learning phase



# How It Work

## Bias



- A bias value allows you to shift the activation function to the left or right, which may be critical for successful learning.
- Changes in weight change the steepness of the curve, while bias shifts the entire curve so that it is more suitable.
- Bias only affects the output value, it does not interact with the actual input data
- Bias value can be adjusted during learning phase

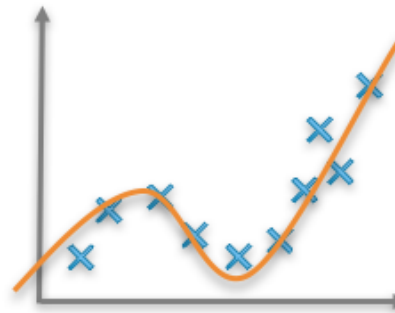
$$y = \sum_i x_i w_i + b$$



# Activation Functions



Linear function



Non-linear function

- Decides whether a neuron should be activated or not by calculating weighted sum and adding bias with it.
- Introduce non-linearity into the output of a neuron → making it capable to learn and perform more complex tasks
- If we do not use activation function, the output signal produced is only a linear function



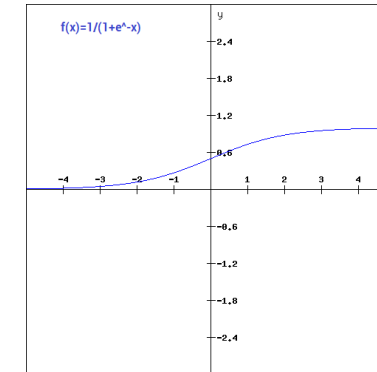
# Activation Functions

$$y = f(\sum_i x_i w_i + b)$$

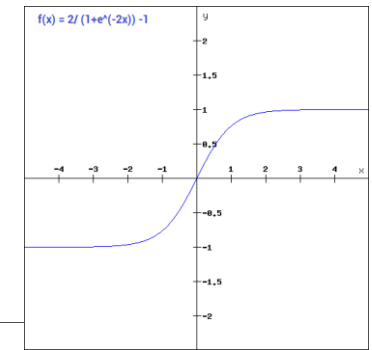
## Commonly used activation functions:

- Sigmoid function :  $A = \frac{1}{1+e^{-x}}$
- Tanh function :  $\tanh(x) = \frac{2}{1+e^{-2x}} - 1$
- ReLU (Rectified Linear Unit) :  $A(x) = \max(0, x)$
- Softmax function : converts an array of values into an array of probabilities (0 - 1)

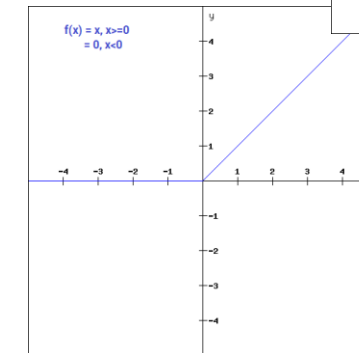
$$\text{softmax}(n) = \frac{\exp n_i}{\sum \exp n_i}$$



Sigmoid function



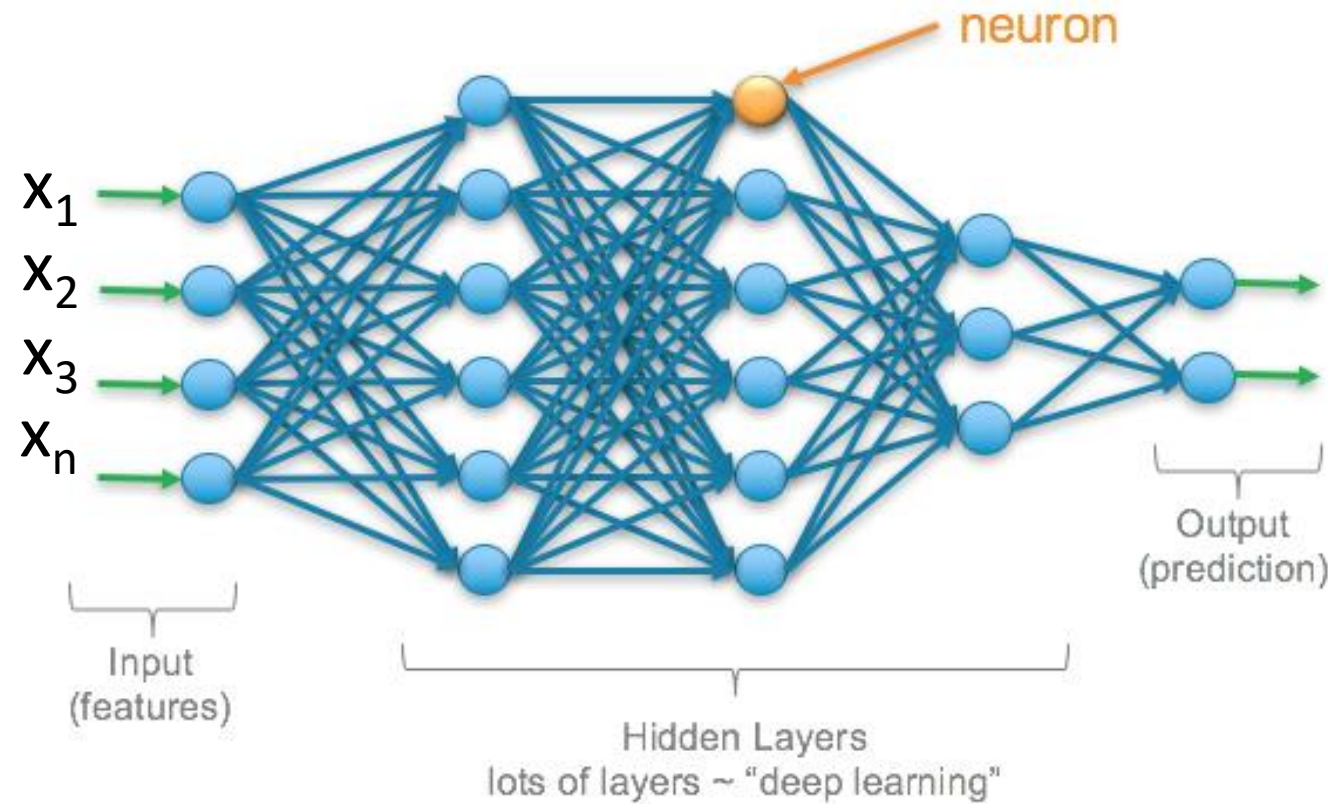
tanh function



ReLU function



# Multilayer Perceptron



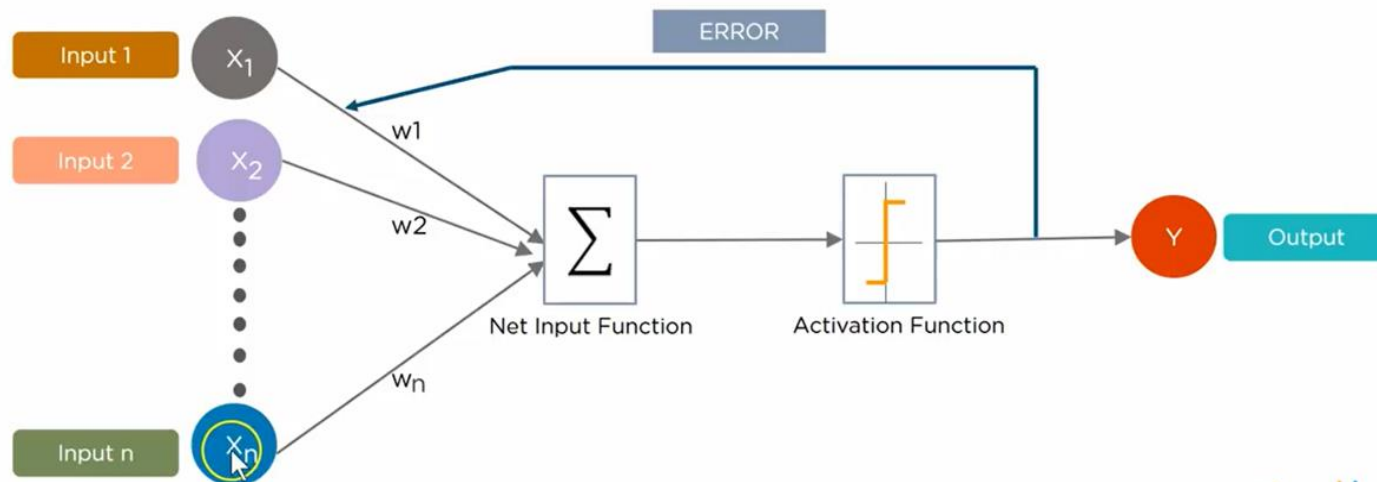
$$y_1 = P(\text{cat} | x)$$

$$y_2 = P(\text{dog} | x)$$



# Learning Rule

## 2 Steps of Learning



### Forward propagation

- weights and bias are initialised randomly
- calculate forward (from input layer to output layer) to get the output
- compare it with the real value to get the error → using lost function or cost function

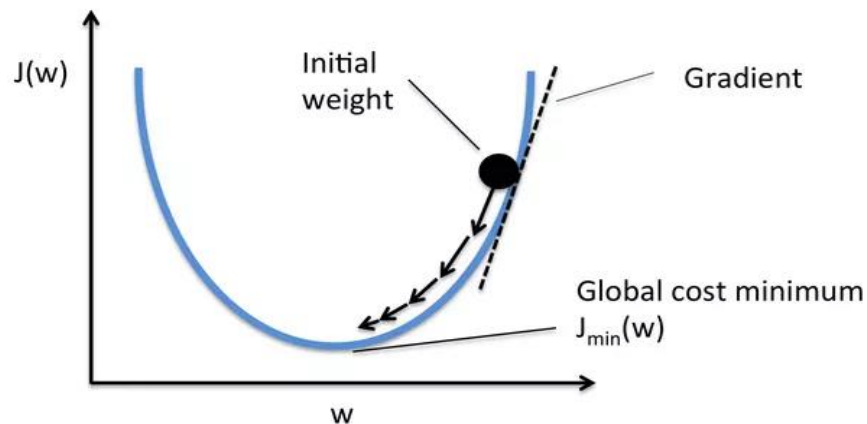
### Backward propagation

propagate backwards and do:

- finding the derivative of the lost function with respect to each weight
- update the weight by subtracting this value from the weight value.



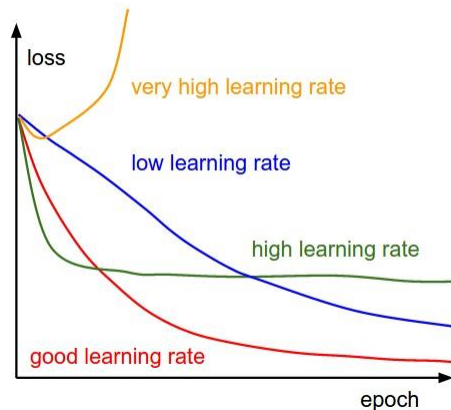
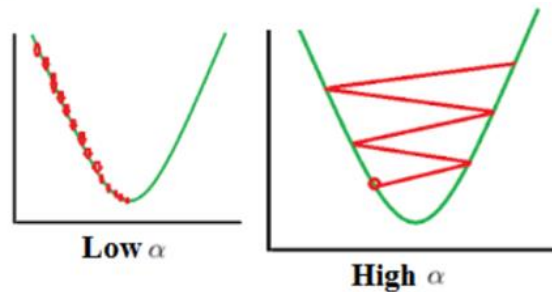
# Loss Function and Gradient Descent



- Loss function : a method of evaluating how well your algorithm models your dataset.
- Learning = minimizing loss function
- How = change the weights\* by calculating the gradient (i.e. the partial derivative of loss function with respect to weights)
- Since we want the value to be minimum  $\rightarrow$  gradient descent. This is the simplest and most popular optimization method for deep neural network
- Other methods : Newton's method, Conjugate Gradient, Quasi-Newton, Levenberg-Marquardt algorithm.



# Learning Rate



the effects of different learning rates

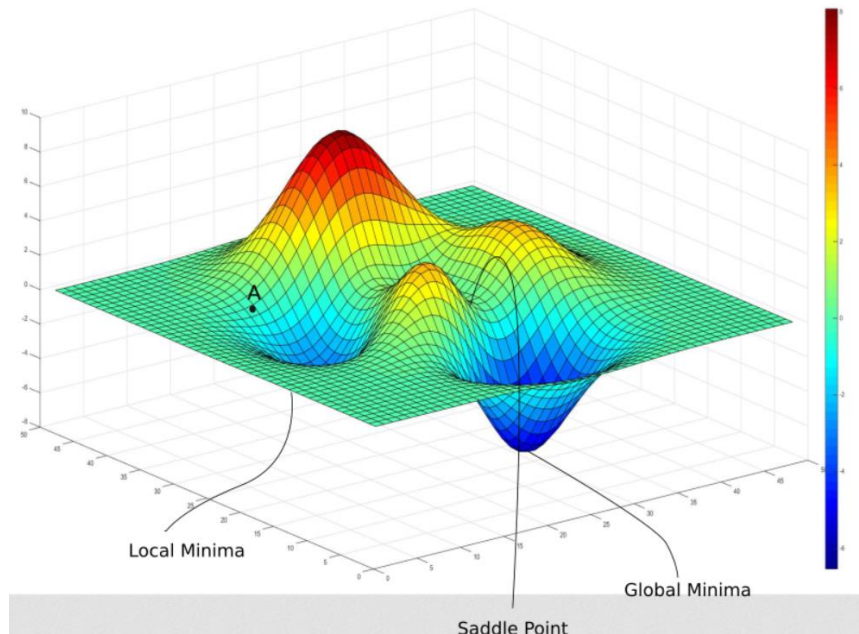
- The gradient told us the direction to change the weight.
- How much we must change is determined by another hyperparameter called learning rate or  $\alpha$
- Picking the value of  $\alpha$  is crucial
  - too small  $\rightarrow$  training process too slow
  - too big  $\rightarrow$  diverging away from the minimum / overshoot
- We can pick  $\alpha$  manually or using optimization technique that utilize adaptive learning rate (e.g. Adagrad, RMSProp, Adam)
- For example:

$$w' = w - \alpha (\partial E / \partial w) \rightarrow \alpha \text{ is learning rate}$$





# Gradient Descent Implementation



Challenges in Gradient Descent

- When we should update the weight? → there are 3 types of gradient descent implementation
- Batch Gradient Descent : process all training dataset in a single batch, calculate the error and update the weight accordingly.
- Stochastic Gradient Descent : randomized the data, and update the weights for each training instance.
- Minibatch Gradient Descent : splits the training dataset into small batches, calculate error and update weights for each small batches.



# Overfitting and Regularization



- The complicated nature of deep neural network makes it prone to overfitting.
- A model is overfit if it performs well on training data but not generalize well on new unseen data
- Regularization is any modification we make to the learning algorithm that is intended to reduce the generalization error, but not its training error (Ian Goodfellow)
- The purpose is to control model complexity and reduce overfitting



# Most Common Regularization Technique

## Weight penalty L2 & L1

Loss = Loss + Regularization

L2 → Loss = Loss +  $\alpha \sum w^2$

L1 → Loss = Loss +  $\alpha \sum w$

## Dropout

- At each training iteration a dropout layer randomly removes some nodes in the network along with all of their incoming and outgoing connections.

## Early stopping

- Interrupting the training procedure once model's performance on a validation set gets worse

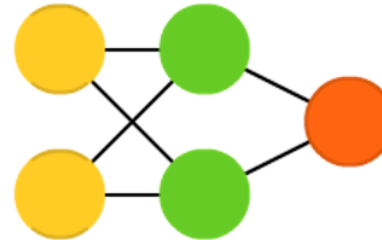
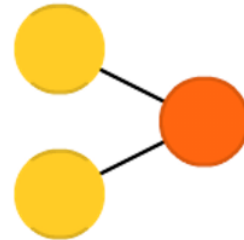
## Dataset augmentation

- Use bigger dataset by using synthetic data



# Feed forward neural networks (FF or FFNN) and perceptrons (P)

---

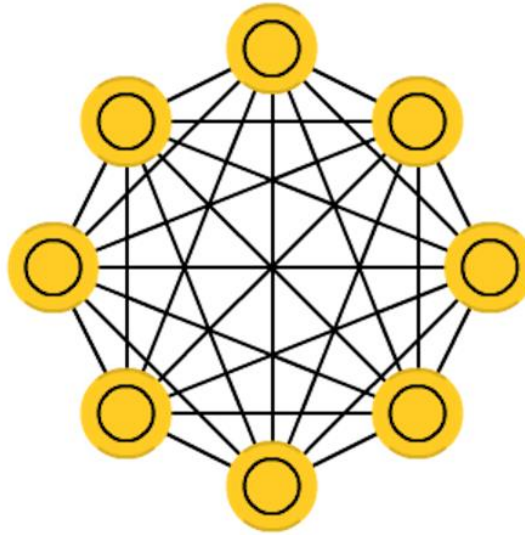


Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain."  
Psychological review 65.6 (1958)



# Hopfield network (HN)

---

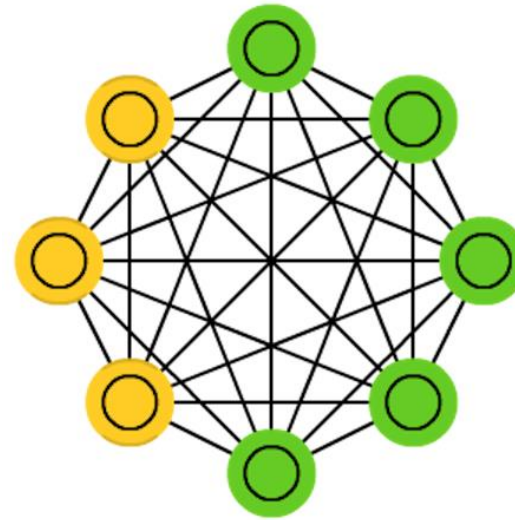


Hopfield, John J. "Neural networks and physical systems with emergent collective computational abilities." Proceedings of the national academy of sciences 79.8 (1982): 2554-2558



# Boltzmann machines (BM)

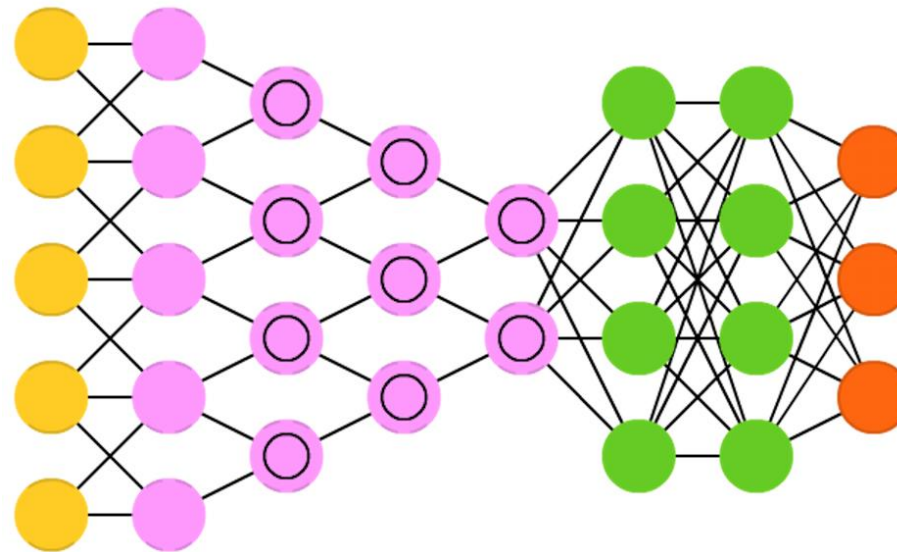
---



Hinton, Geoffrey E., and Terrence J. Sejnowski. "Learning and relearning in Boltzmann machines." *Parallel distributed processing: Explorations in the microstructure of cognition 1* (1986): 282-317



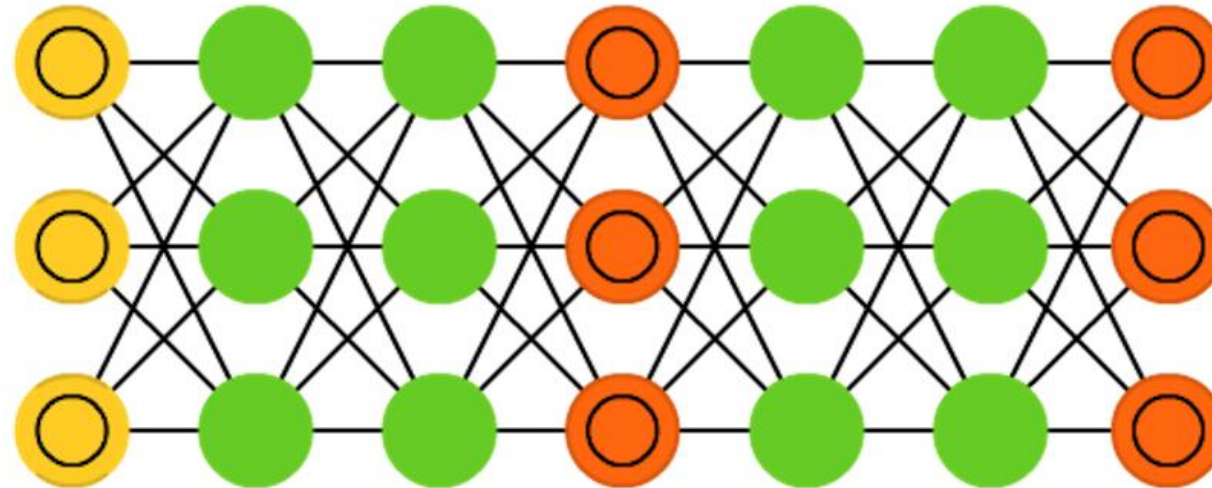
# Convolutional neural networks (CNN)



LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.



# Generative adversarial networks (GAN)



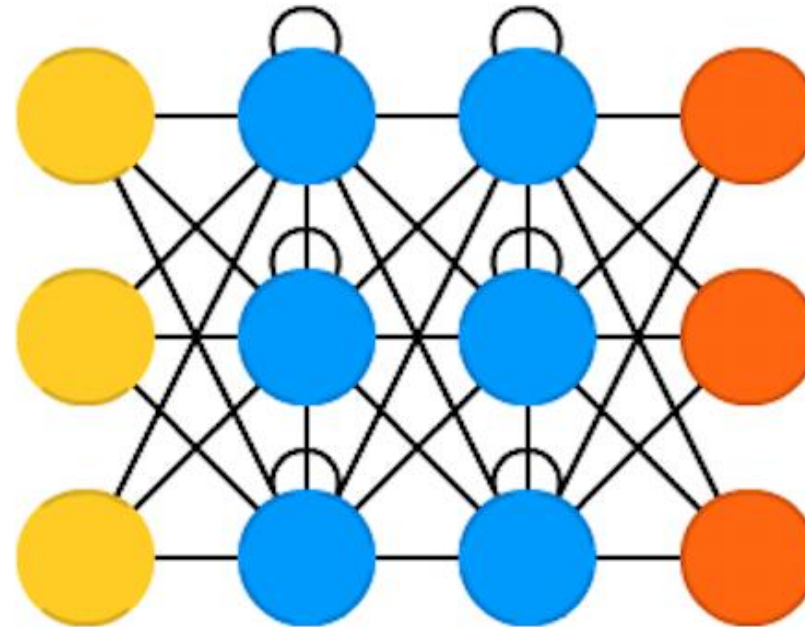
Goodfellow, Ian, et al. "Generative adversarial nets." Advances in Neural Information Processing Systems. 2014.





# Recurrent neural networks (RNN)

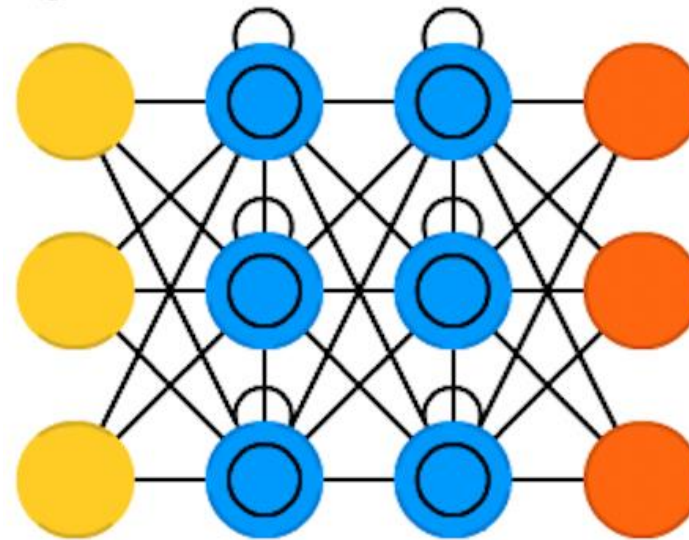
---



Elman, Jeffrey L. "Finding structure in time." Cognitive science 14.2 (1990): 179-211



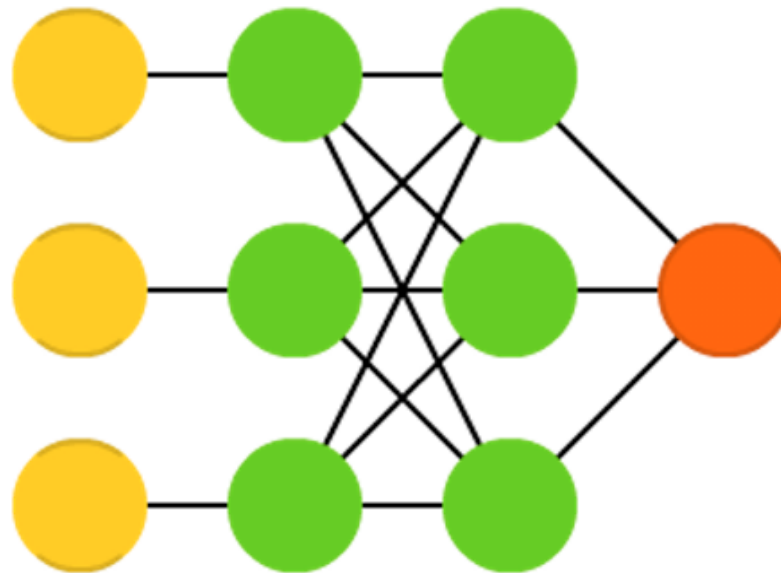
# Long / short term memory (LSTM)



Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780



# Support vector machines (SVM)

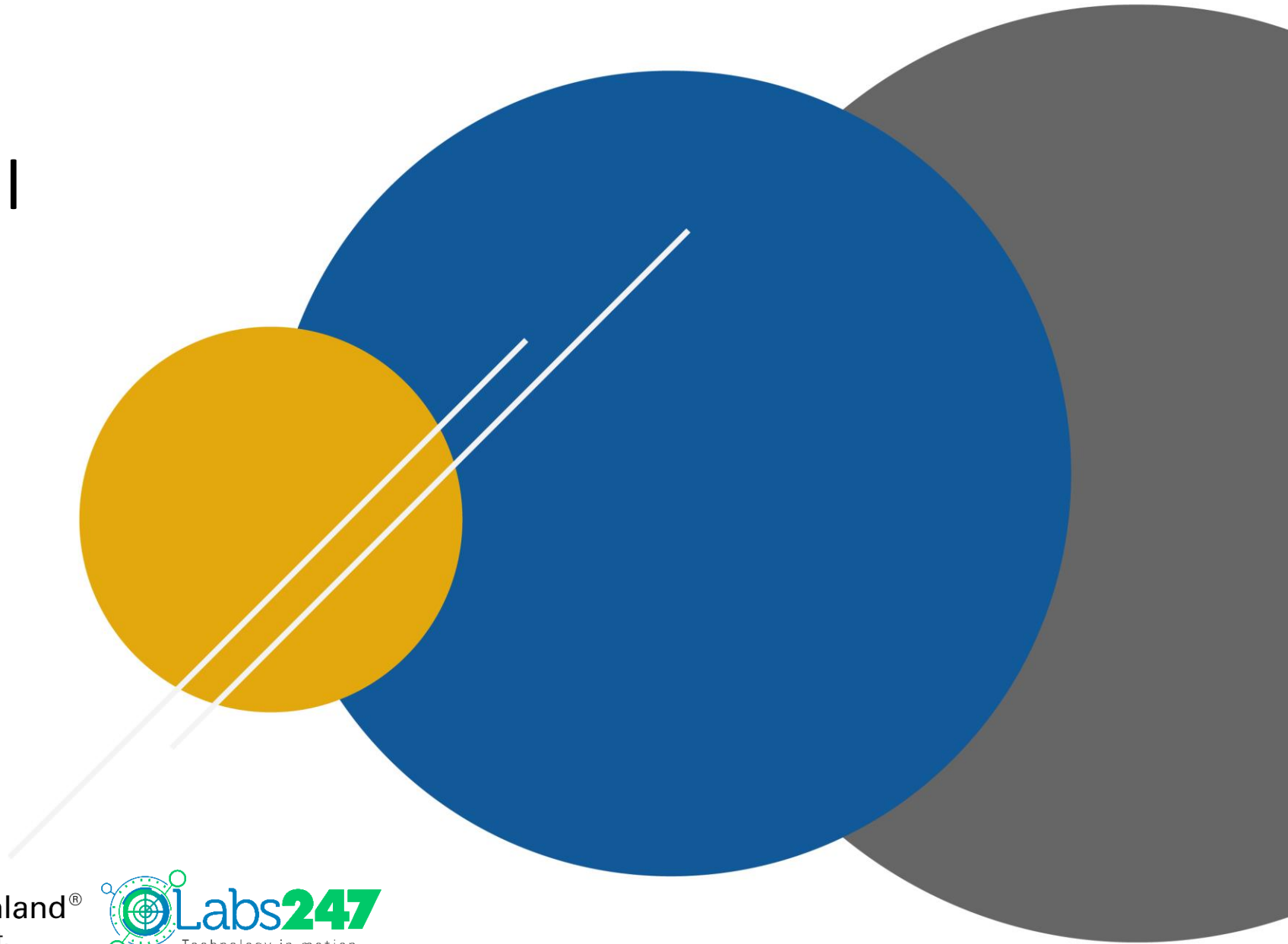


Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20.3 (1995): 273-297

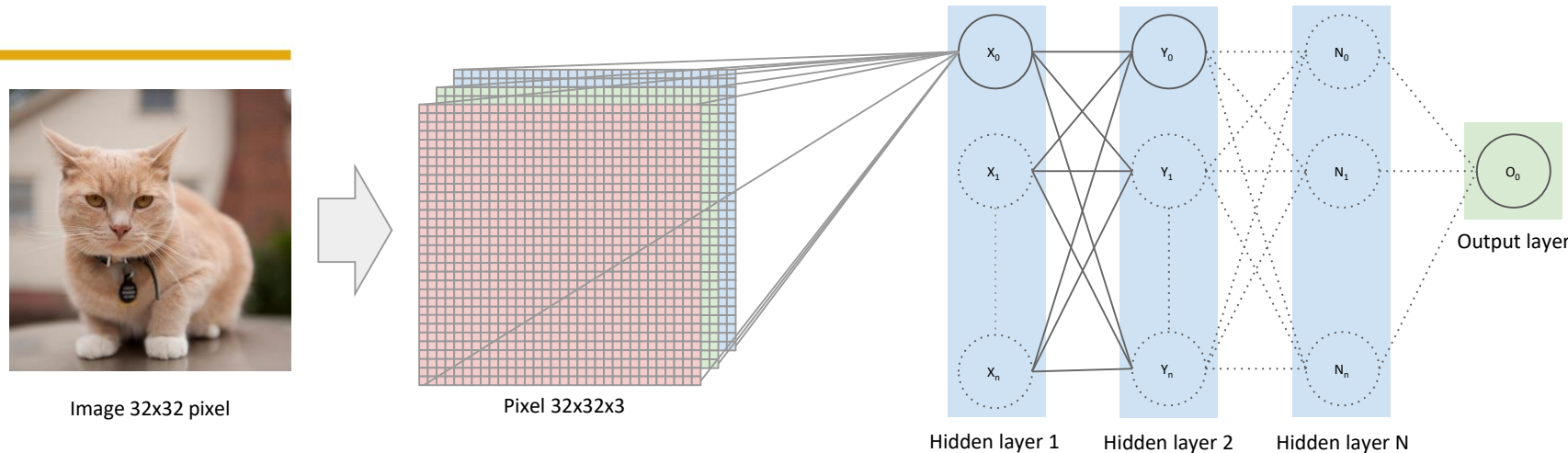


## CHAPTER 03

# Convolution Neural Network



# Image Processing in ANN



- Regular Neural Nets don't scale well to full images

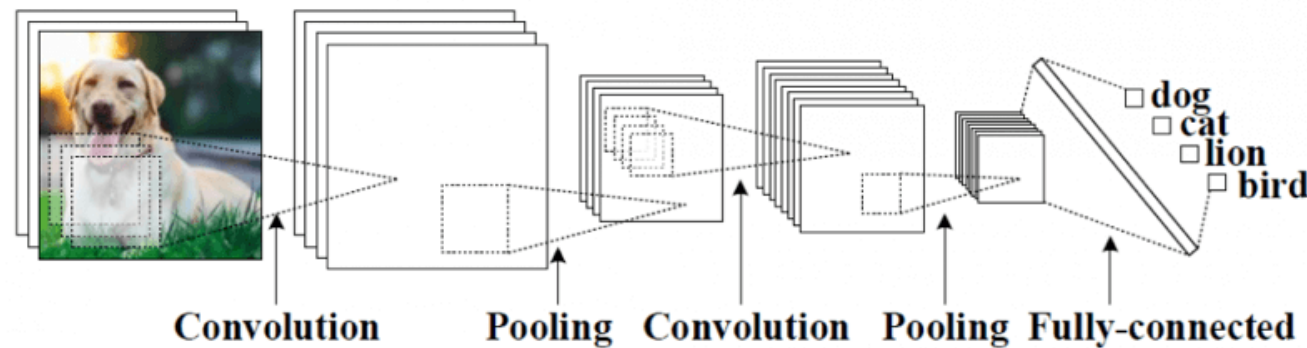
E.g : CIFAR-10 images size 32x32x3 (32x32, 3 color channels - RGB)  $\rightarrow 32 \cdot 32 \cdot 3 = 3072$  weights in a single neuron in a first hidden layer

For 200x200x3 image  $\rightarrow 200 \cdot 200 \cdot 3 = 120,000$  weights

- Parameters would add up quickly  $\rightarrow$  quickly lead to overfitting.
- In order to reduce the number of parameters, it is used by using the convolution method



# Convolutional Neural Network Overview

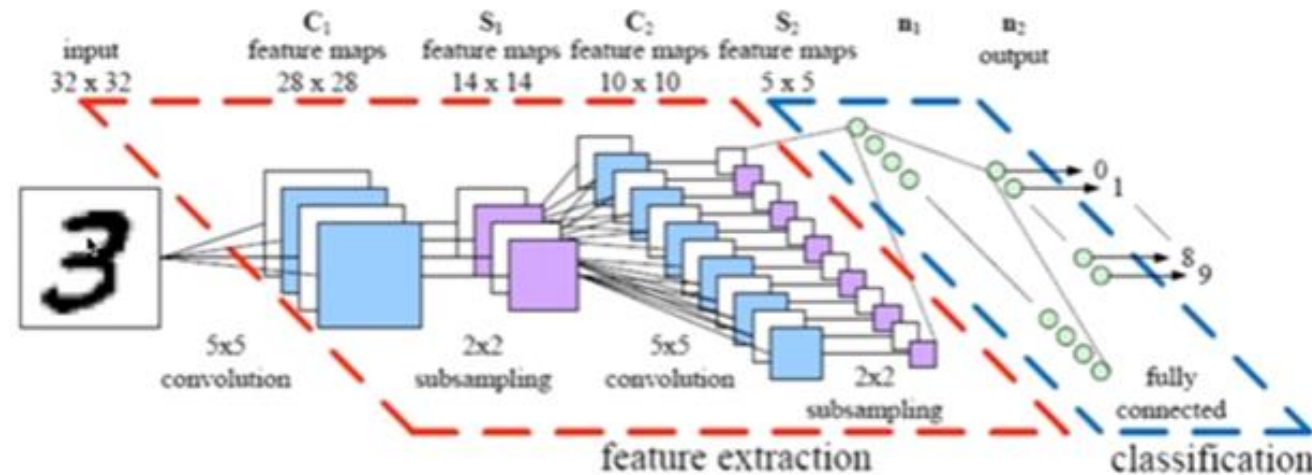


- ○ A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network.
- ○ in 1998, Convolutional Neural Networks were introduced in a paper by Bengio, Le Cun, Bottou and Haffner.
- ○ Their first Convolutional Neural Network was called LeNet-5 and was able to classify digits from hand-written numbers.

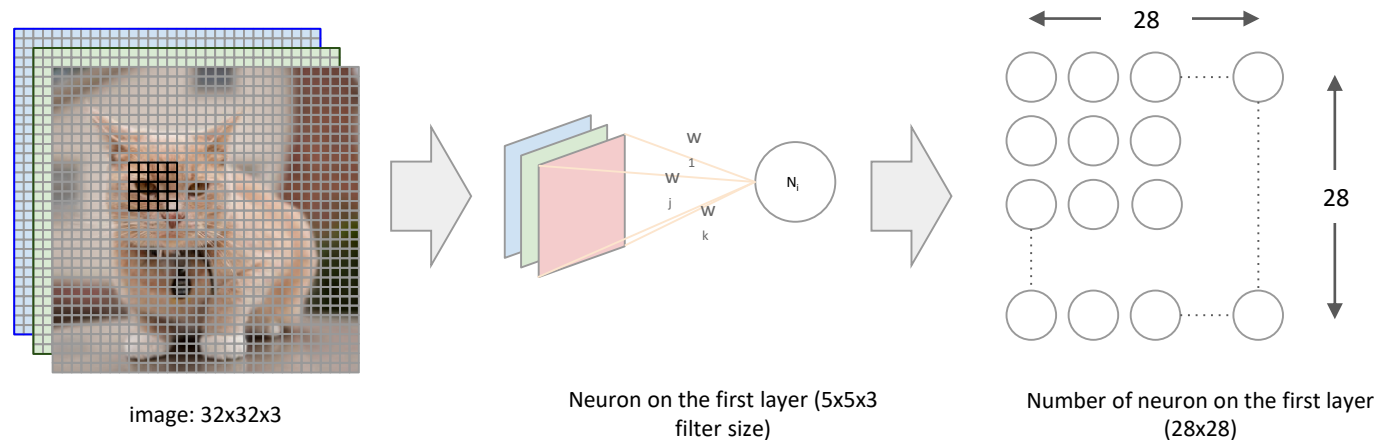


# CNN Layers

- ○ The common types of layers in CNN architecture : Convolution - Pooling - Fully Connected
  - Convolution layer is the core that does most of the computation
  - Pooling layer performs dimensionality reduction and controls overfitting. Commonly puts between the convolution layers. The Convolution and Pooling layer acts as feature extraction part
  - Fully Connected layer generally ends the CNN. It acts as classification part



# Convolution Layer



- The first layer of CNN is always convolution layer
- Convolution layer consist of a set of learnable filters (also called kernel, or weight)
- Filter is a small array of number, and covers all the input depth.

For the CIFAR-10 data, we may have a 5x5x3 filter (5x5 matrix for 3 color channels)

Each neuron will have weights to a [5x5x3] region in the input volume →  $5*5*3 = 75$  weights → this is what we will

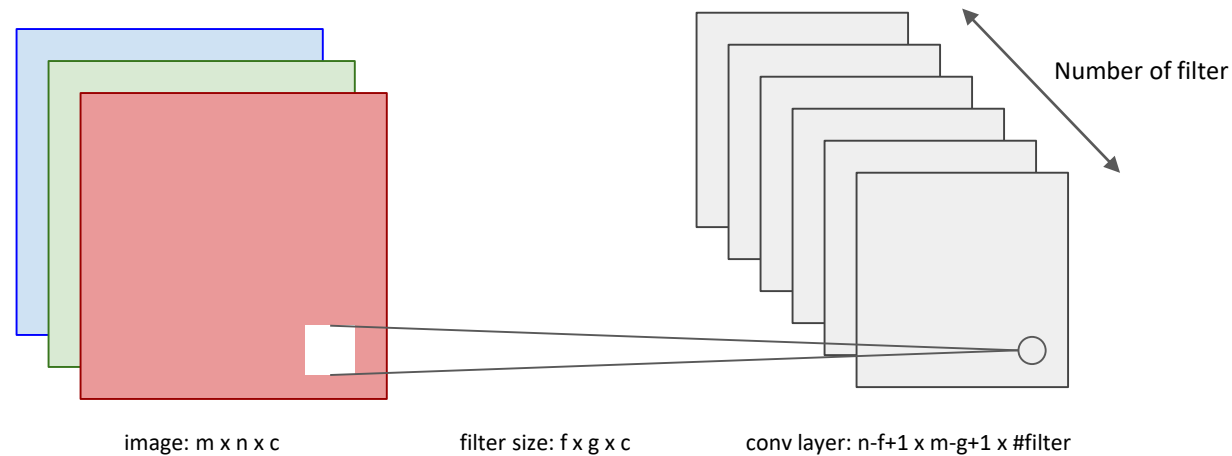
learn

- The filter will convolve around the image, performing dot product operation between the filter and the part of the image it convolves with





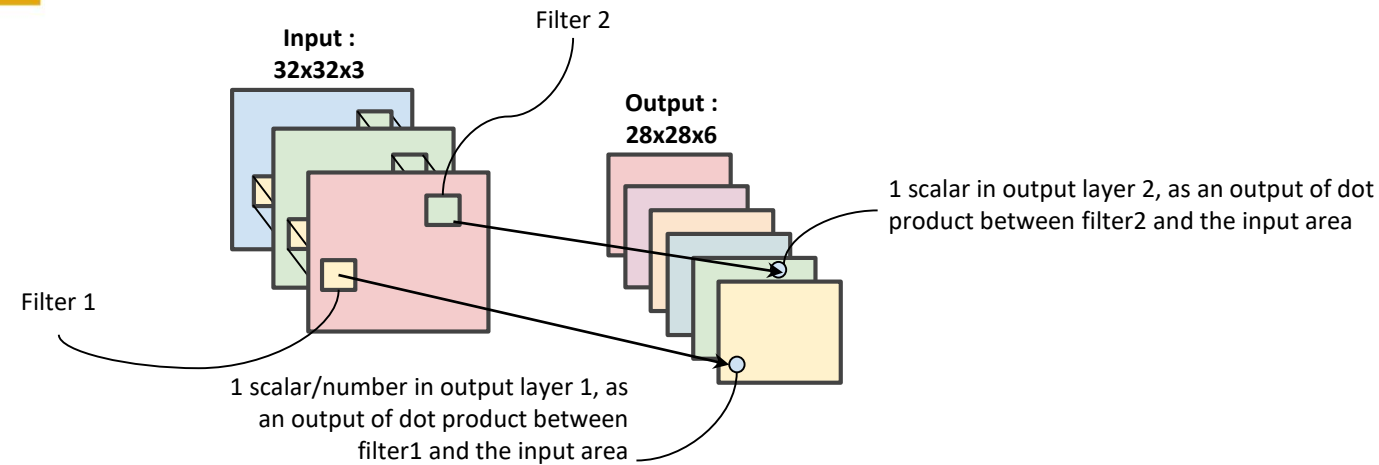
# Convolution Layer Hyperparameter



- There are 3 hyperparameters to set in the conv layer: number of filter, stride, and padding
  - Filter size
  - Number of filter
  - Stride
  - Padding



# Number of Filters



- The number of filters called depth column (or fibre) → note that it's different from input depth.
- We may have multiple filter for a conv layer, each layer learns different features (e.g. straight edges, blobs of color, curves, etc.)
- The 3rd dimension of output layer of a conv layer = the number of filters
- If we use 6 filters of  $5 \times 5 \times 3$  for our example, the output will be  $28 \times 28 \times 6$  (6 layers of  $28 \times 28$ )



# Stride

- Stride = how much we shift the filter at a time. The bigger the stride, the smaller the output.
- Formula to calculate output width and height =  $(W-F)/S+1$ , where  $W$  = input size (width or height),  $F$  = filter size (width or height), and  $S$  = stride
- Stride is normally set in a way so that the output volume is an integer and not a fraction.
- For stride = 3, the output will be  $(32-5)/3+1 = 10 \times 10$
- Example for 7x7 input with 3x3 filter :

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

Image

4		

Convolved  
Feature

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

Image

4		

Convolved  
Feature

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

Image

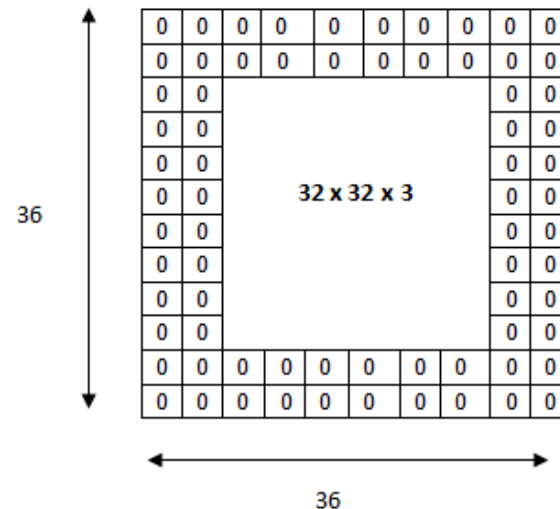
4	3	

Convolved  
Feature



# Padding

- Used to preserve the size of the output.
- The formula :  $(W - F + 2P) / S + 1$ , where P = padding
- If we want to preserve the output spatial size to 32x32 with our 5x5 filter, we can use padding=2 and stride=1, so output =  $(32 - 5 + 4) + 1 = 32$

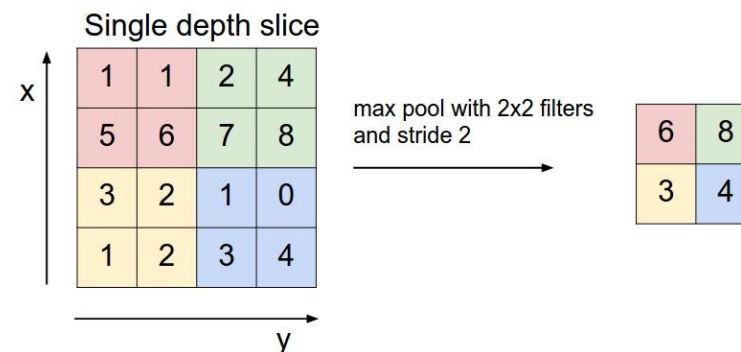


The input volume is 32 x 32 x 3. If we imagine two borders of zeros around the volume, this gives us a 36 x 36 x 3 volume. Then, when we apply our conv layer with our three 5 x 5 x 3 filters and a stride of 1, then we will also get a 32 x 32 x 3 output volume.



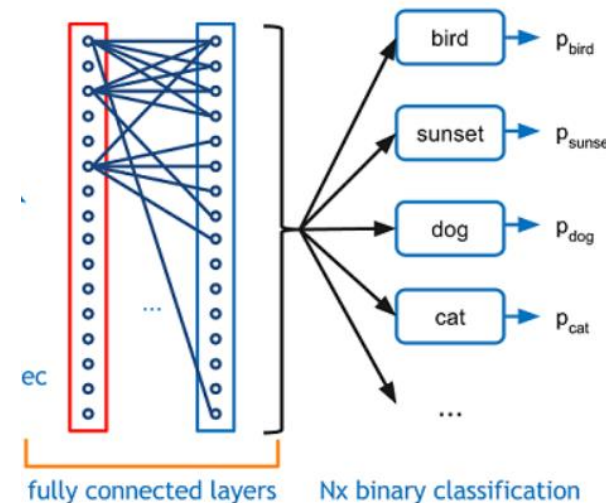
# Pooling Layer

- Also called downsampling layer.
- Reduce the spatial size to reduce the amount of parameters and computation, and to control overfitting.
- It operates independently on every layer/ depth slice of the input → output depth = input depth
- The most common is : 2x2 with stride = 2. Other common setting is  $F=3 \times 3$  ,  $S=2$  (overlapping pooling).
- Most common operation = max pooling. Other less common operation = average pooling



# Fully Connected Layer

- Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural networks.
- Usually put at the end of the CNN architecture to perform the classification.
- FF layer outputs an N dimensional vector where N is the number of classes that the program has to choose from.
- Many newer CNN architectures doesn't use Pooling and/or Fully Connected Layer



# CIFAR-10 Dataset

- The dataset : CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes, commonly used to train machine learning and computer vision algorithms.
- The problem : image classification
- Input format : 32x32x3 arrays of numbers (32x32 RGB image)
- Output : the probability of the image being a certain class (e.g. 0.80 for cat, 0.15 for dog, 0.05 for bird, etc)



What We See

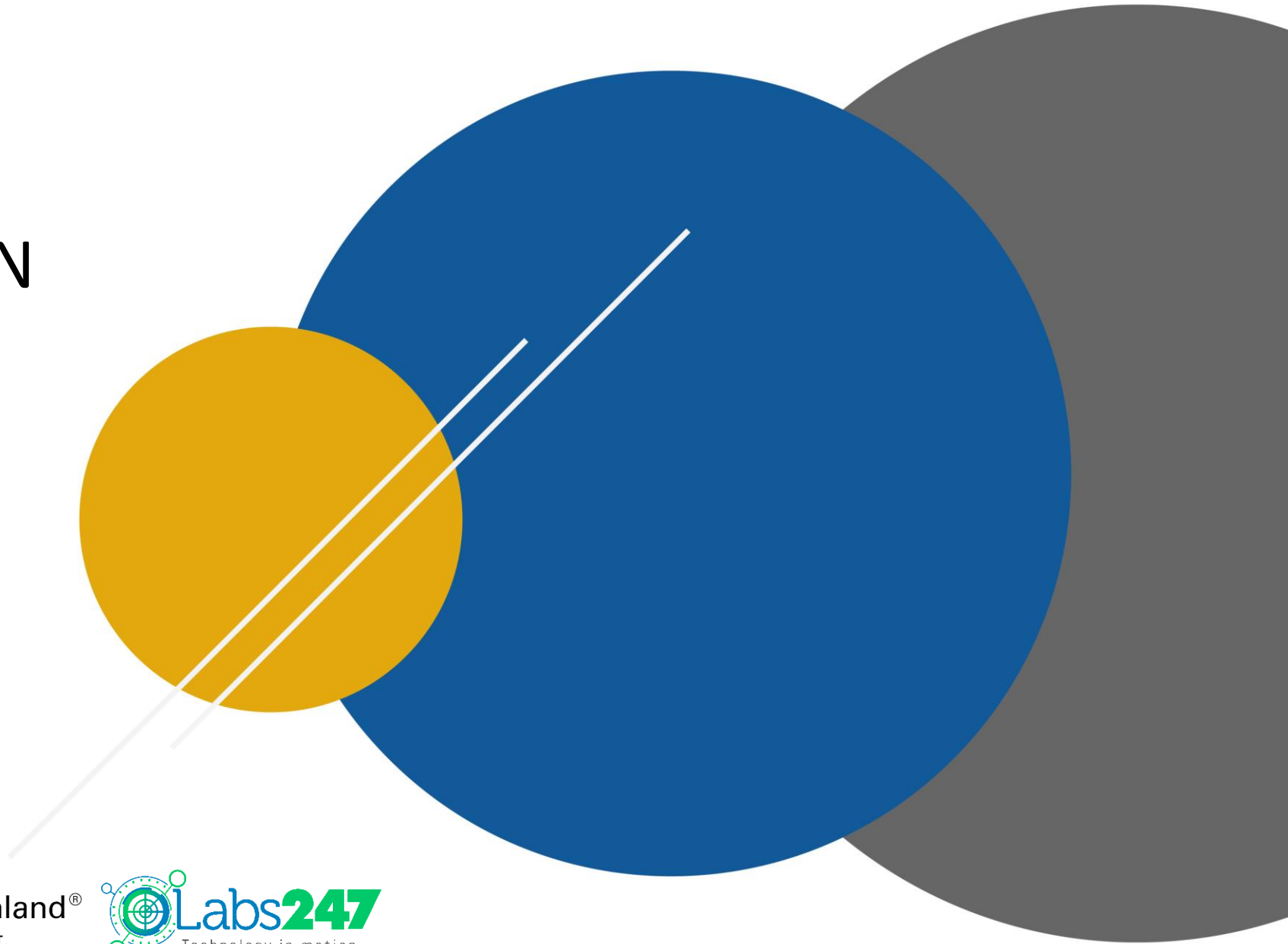
```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 31 08
49 49 39 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 34 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 39 48 61 43 52 01 89 19 67 48
```

What Computers See



# LAB 01

## Train Your First CNN





# Lab Description

---

What you will learn:

- Coding by using basic tensorflow and keras framework
- Creating simple Convolution Neural Network model
- Training and testing CNN model

• Anaconda

• Python 3

- numpy
- pandas
- Matplotlib
- scipy
- jupyter
- tensorflow
- keras

Requirement :



# STEP 01

## Execute Notebook

---

- Open notebook : labs01-firstcnn.ipynb in Jupyter notebook
- Follow the instructor to run notebook
- Instructor will explain step by step process



# Performance Tuning

---

Performance Improvements :

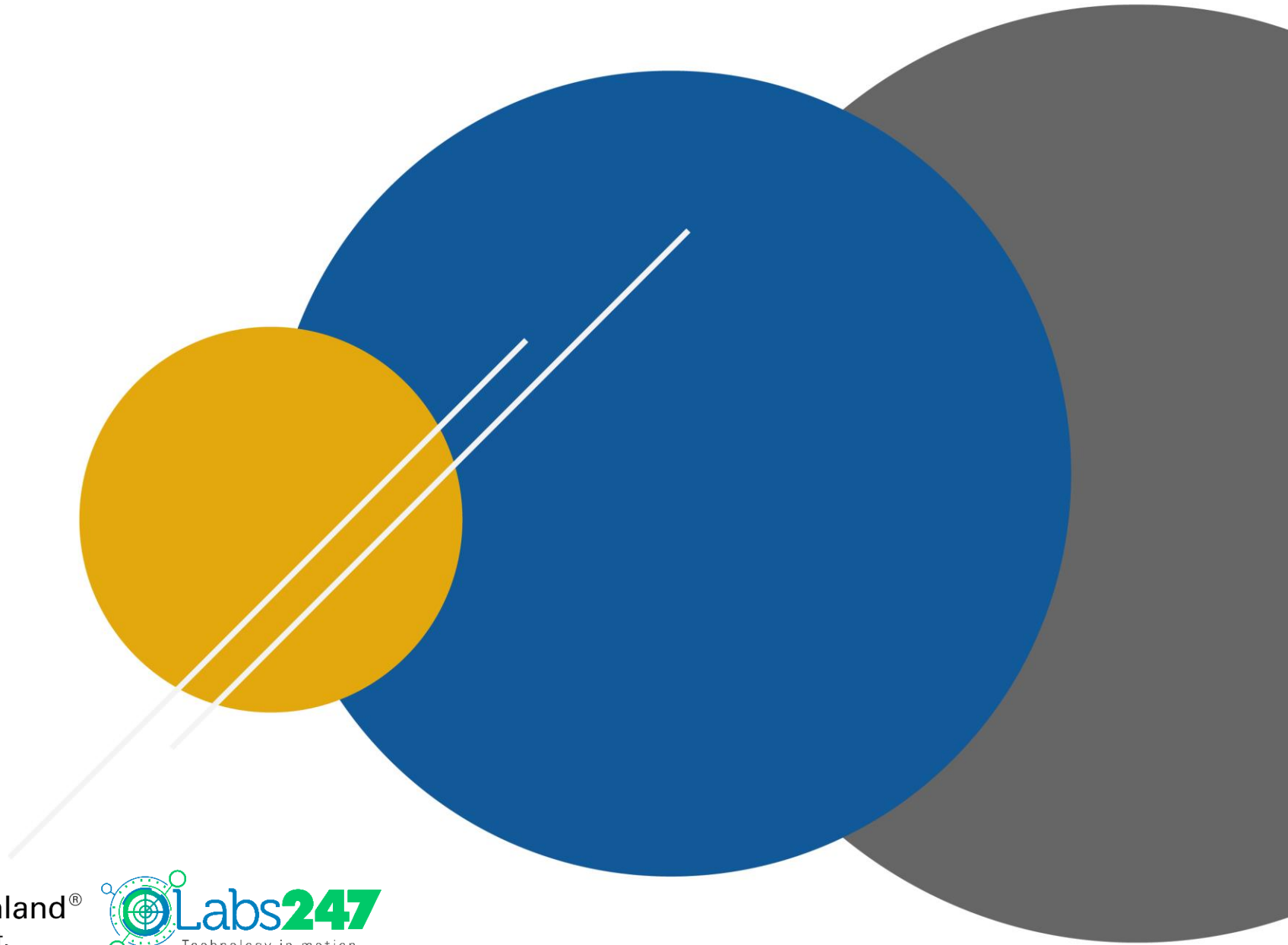
To achieve the best performances we may:

- Fine Tune Hyper-Parameters : Hyper-parameters are the variables which are set before training and determine the network structure & how the network is trained. (eg : learning rate, batch size, number of epochs). Fine tuning can be done by : Manual Search, Grid Search, Random Search...
- Improve Pre-Processing : Better pre-processing of input data can be done as per the need of your dataset like removing some special symbols, numbers, stopwords and so on ...
- Use Dropout Layer : Dropout is regularization technique to avoid overfitting (increase the validation accuracy) thus increasing the generalizing power.



# CHAPTER 04

## Transfer Learning



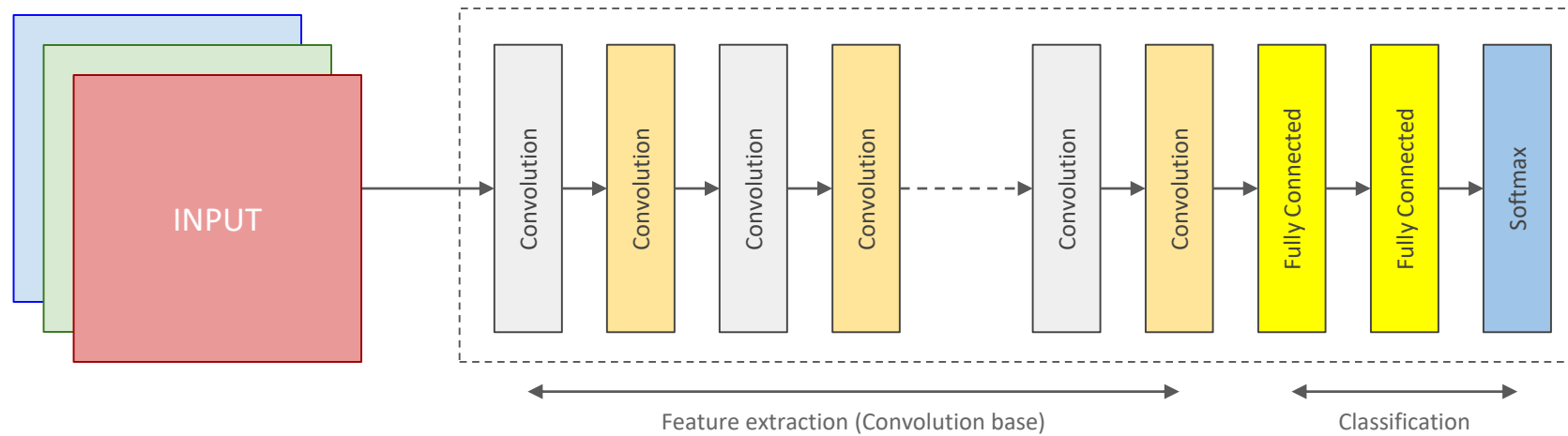
# Transfer Learning

---

- A machine learning technique where a model trained on one task is re-purposed on a second related task → use pre trained models
- Why :
- Small / insufficient training dataset → avoid overfitting
- Cut-off training time
- There are many open source models trained on a very large dataset : ImageNet, COCO, KITTI, etc.
- We can download some open source implementation of the neural network, not only the code, but also the model and weights that have already trained.



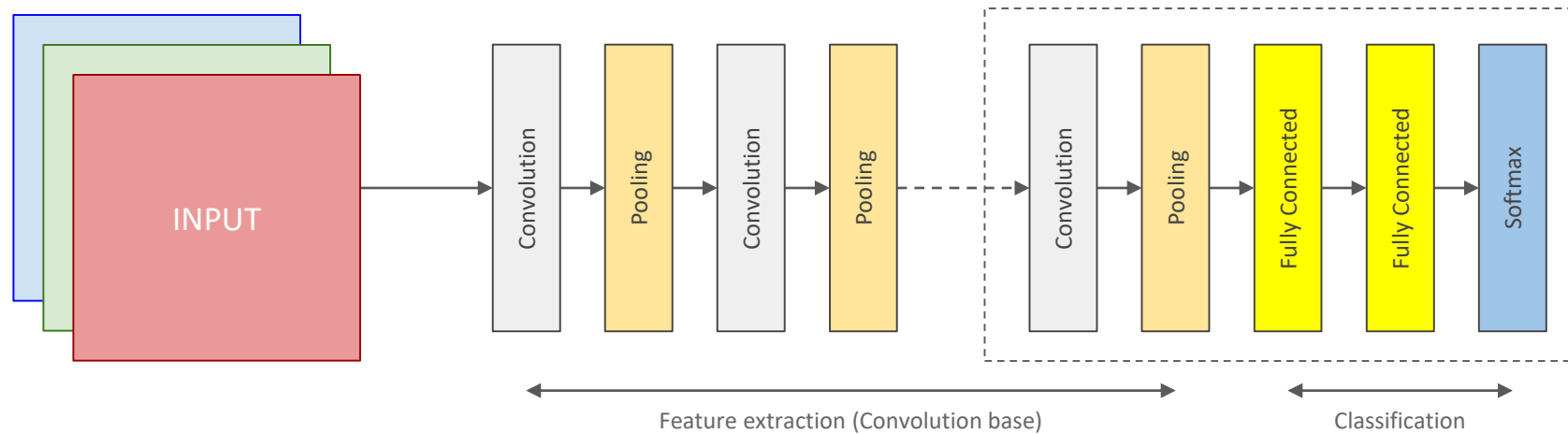
# Train The Entire Model



- Large dataset
- Customize softmax as needed
- Re-train all network by using trained weight as initial value



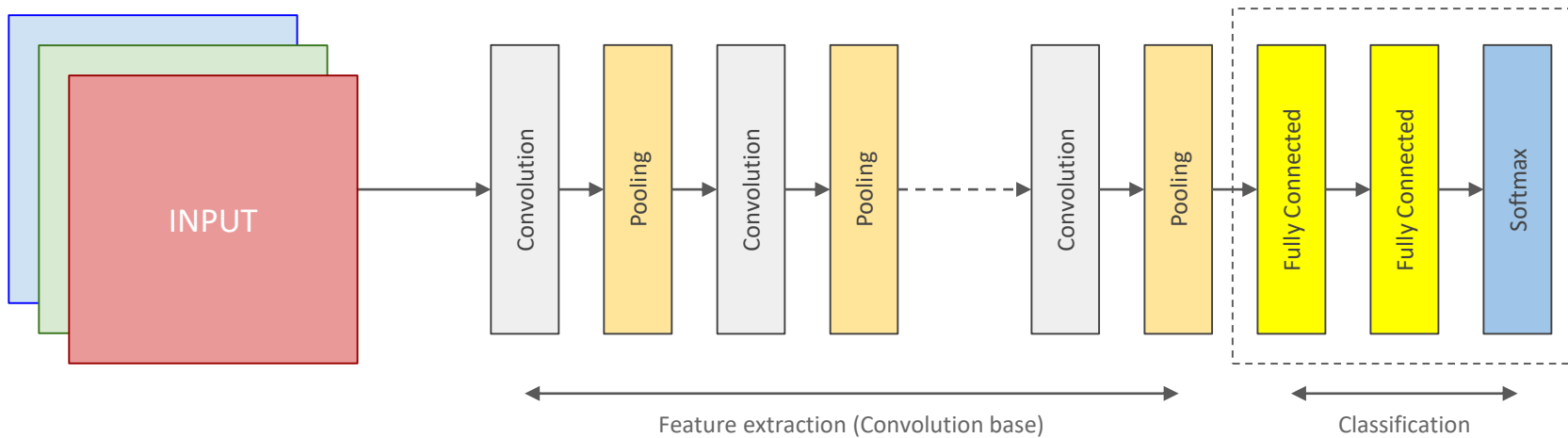
# Train Partial Layer



- 🔧 Large dataset
- 🔧 Customize softmax as needed
- 🔧 Retraining some convolution layer and classification layer
- 🔧 Freeze other layer



# Train Top Layer

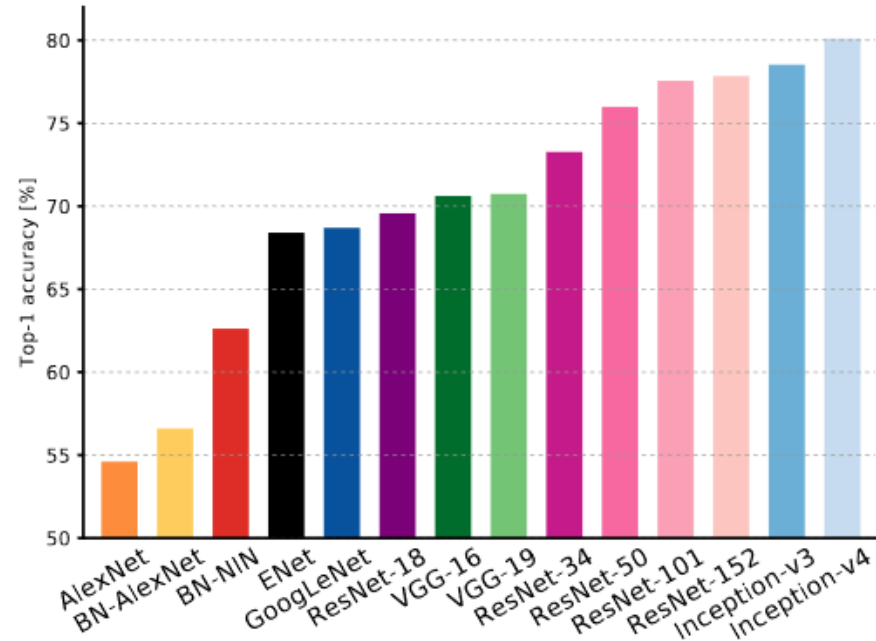


- Small dataset
- Customize softmax as needed
- Freeze all feature layer
- Retrain classification





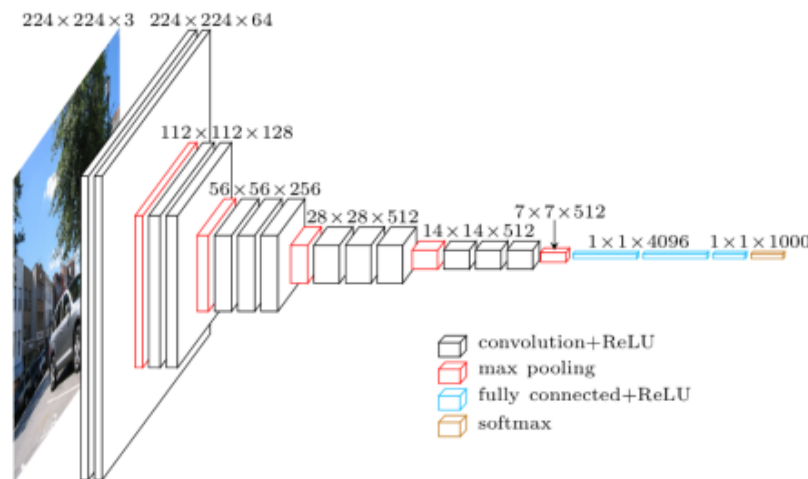
# CNN - State of The Art Architectures



- There are many of models shared by community
- Chart above is benchmark various model by using imagenet dataset
- Source: An Analysis of Deep Neural Network Models for Practical Applications, Alfredo Canziani, et al



# CNN - VGG 16 & 19

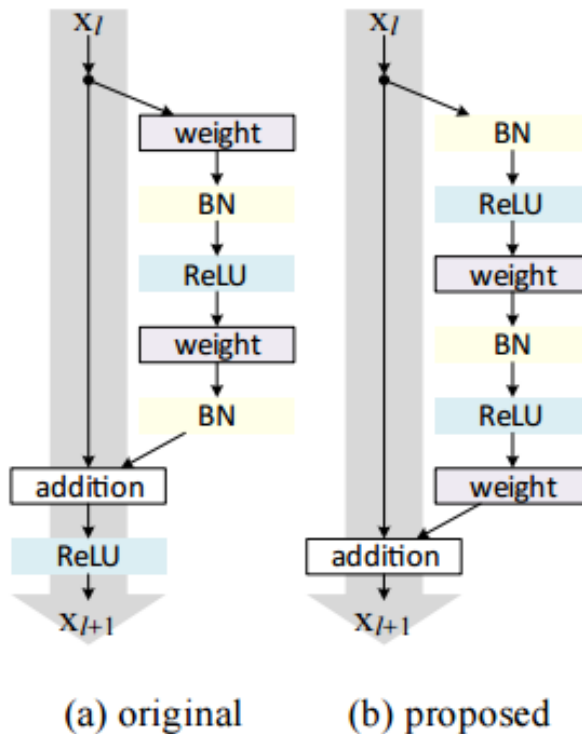


ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

- Simonyan and Zisserman (2014), Very Deep Convolutional Networks for Large Scale Image Recognition.
- Characterized by its simplicity, using 3×3 convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. Two fully-connected layers, each with 4,096 nodes are then followed by a softmax classifier (above).
- The “16” and “19” stand for the number of weight layers in the network (columns D and E)



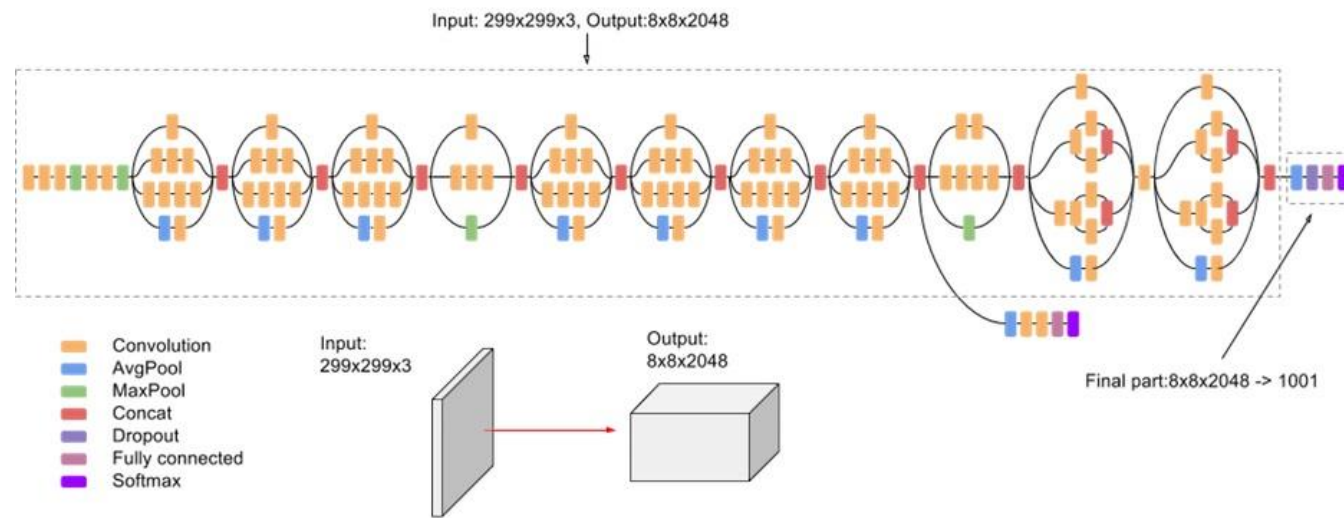
# CNN - ResNet



- He et al. (2015) , Deep Residual Learning for Image Recognition and (2016) Identity Mappings in Deep Residual Networks
- Relies on micro-architecture modules (also called “network-in-network architectures”). are then followed by a softmax classifier (above).
- Much deeper than VGG16 and VGG19, the model size is substantially smaller due to the usage of global average pooling rather than fully-connected layers — this reduces the model size down to 102MB for ResNet50 (50 layers).



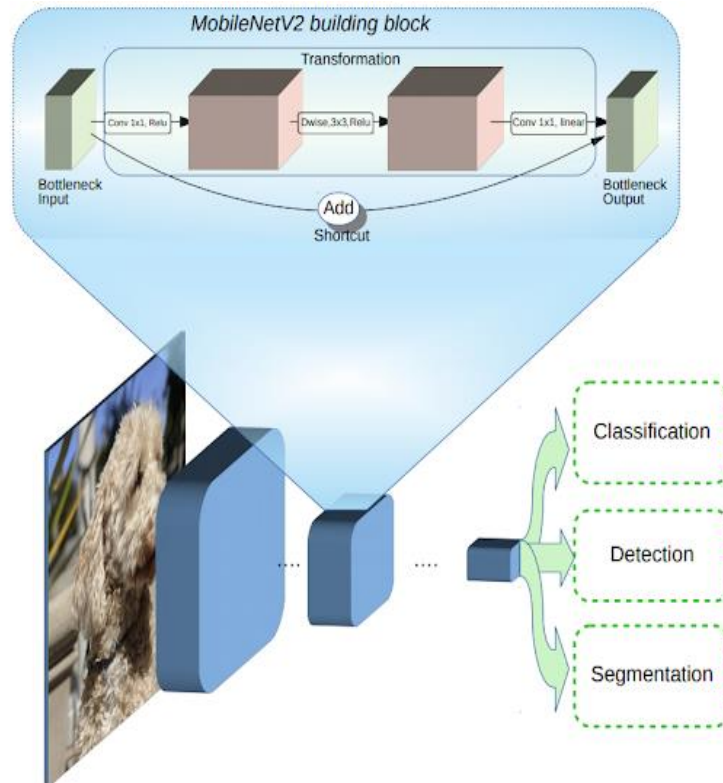
# CNN - Inception



- Szegedy et al. (2014), Going Deeper with Convolutions: and Inception V3 (2015) Rethinking the Inception Architecture for Computer Vision
- The goal of the inception module is to act as a “multi-level feature extractor” by computing  $1 \times 1$ ,  $3 \times 3$ , and  $5 \times 5$  convolutions within the same module of the network — the output of these filters are then stacked along the channel dimension before being fed into the next layer in the network.



# CNN - MobileNet V2

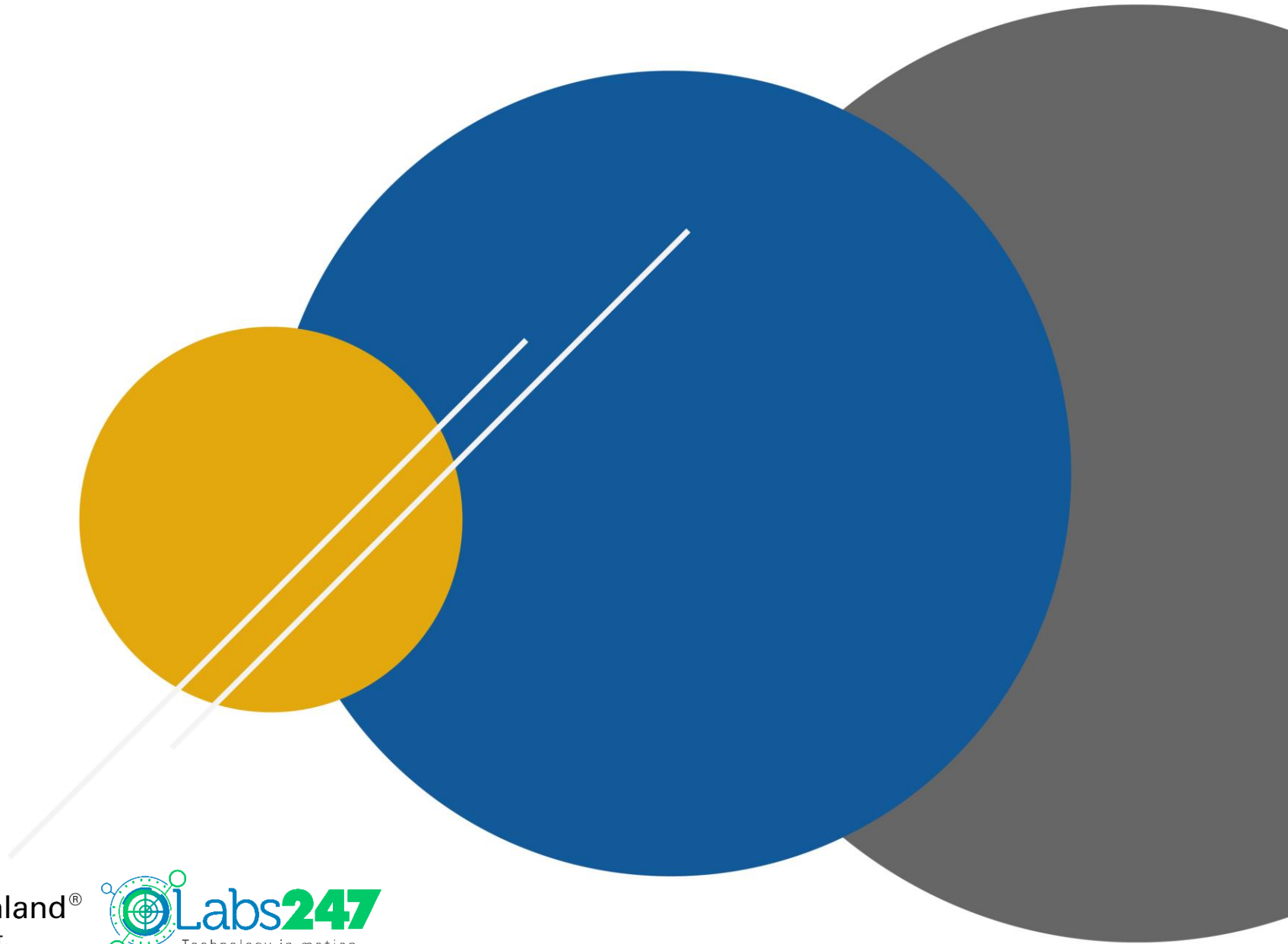


- Sandler et. al. (2018) MobileNetV2: Inverted Residuals and Linear Bottlenecks
- general purpose computer vision neural networks designed with mobile devices in mind to support classification, detection and more.
- very effective feature extractor for object detection and segmentation.



## LAB 02

# Transfer Learning



# Lab Description

---

## What you will learn:

- Using VGG19 model
- Retrain VGG19 model using Caltech 101 dataset

## Requirement :

- Anaconda
- Python 3
  - numpy
  - pandas
  - Matplotlib
  - scipy
  - jupyter
  - tensorflow
  - keras



# STEP 01

## Execute Notebook

---

- Open notebook : labs02-transferlearning.ipynb in Jupyter notebook
- Follow the instructor to run notebook
- Instructor will explain step by step process

