Artificial Intelligent
Deep Learning

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

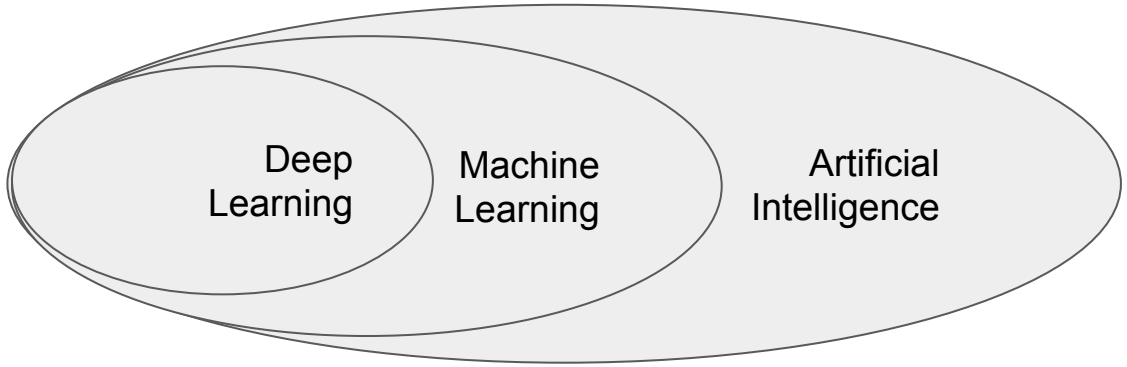
Agenda

- 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

Chapter 1
Introduction

Deep Learning Definition

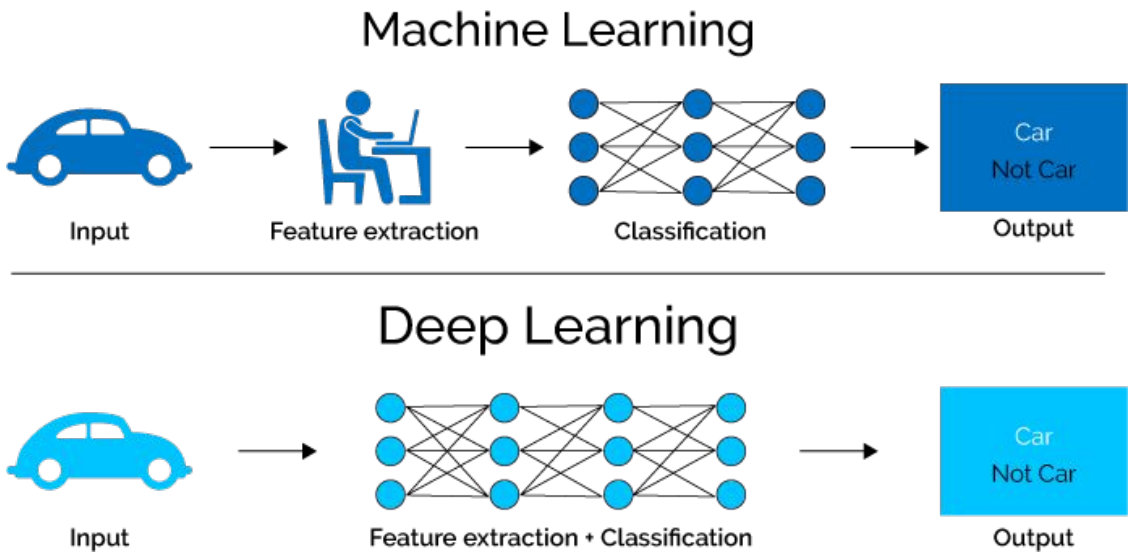
- Machine learning algorithms based on learning multiple levels (i.e deep) of representation/ abstraction [1]
- Learning algorithms derive meaning out of data by using a hierarchy of multiple layers of units (neurons).
- Each neuron/node computes a weighted sum of its inputs and the weighted sum is passed 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?

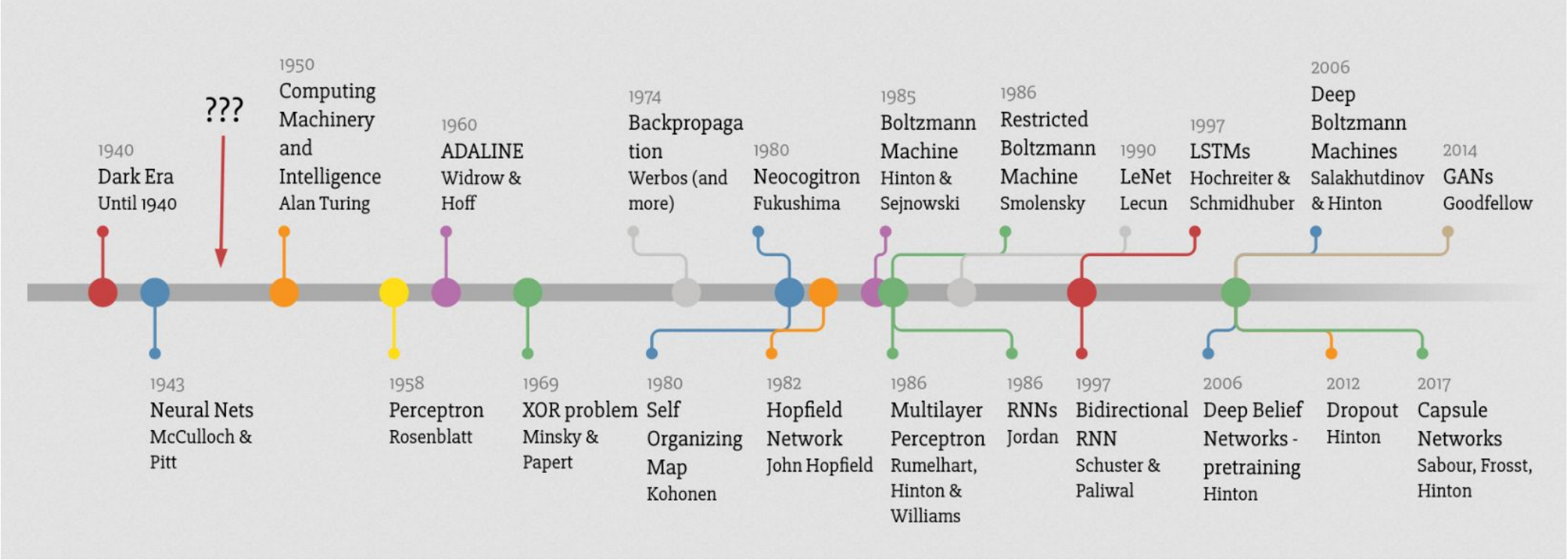
- Exponential data growth (and the ability to Process Structured & Unstructured data)
- Faster & open distributed systems (Hadoop, Spark, TensorFlow, ...)
- Faster machines and multicore CPU/GPUs
- New and better models, algorithms, ideas:
  - Better, more flexible learning of intermediate representations
  - 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

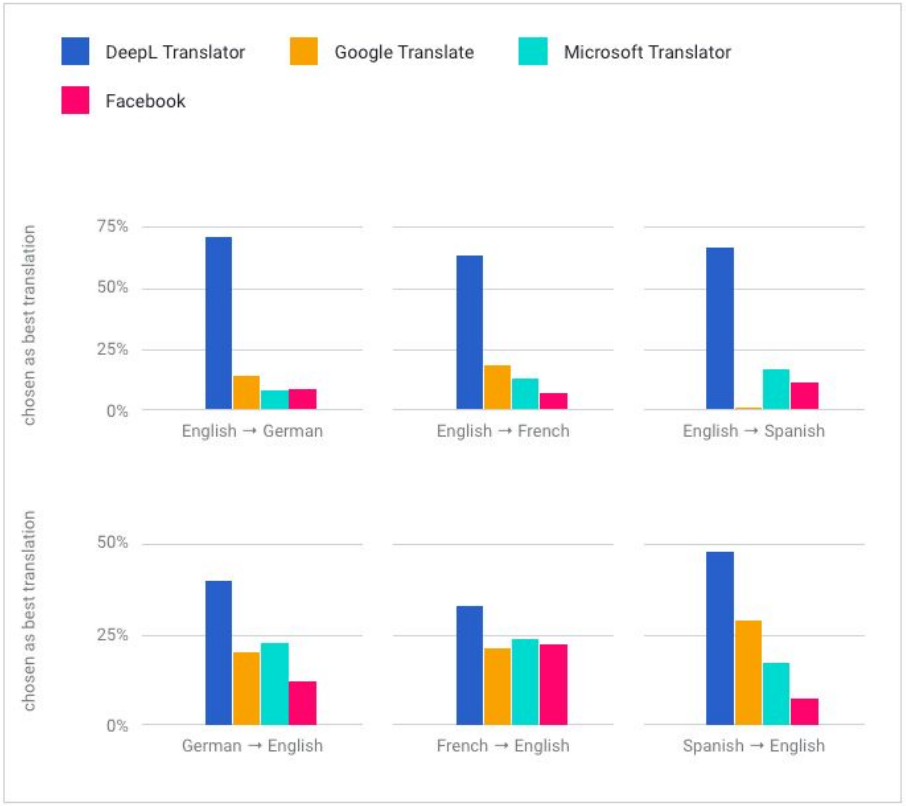
Deep Learning
Timeline



Use Case
Self Driving Car

- 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



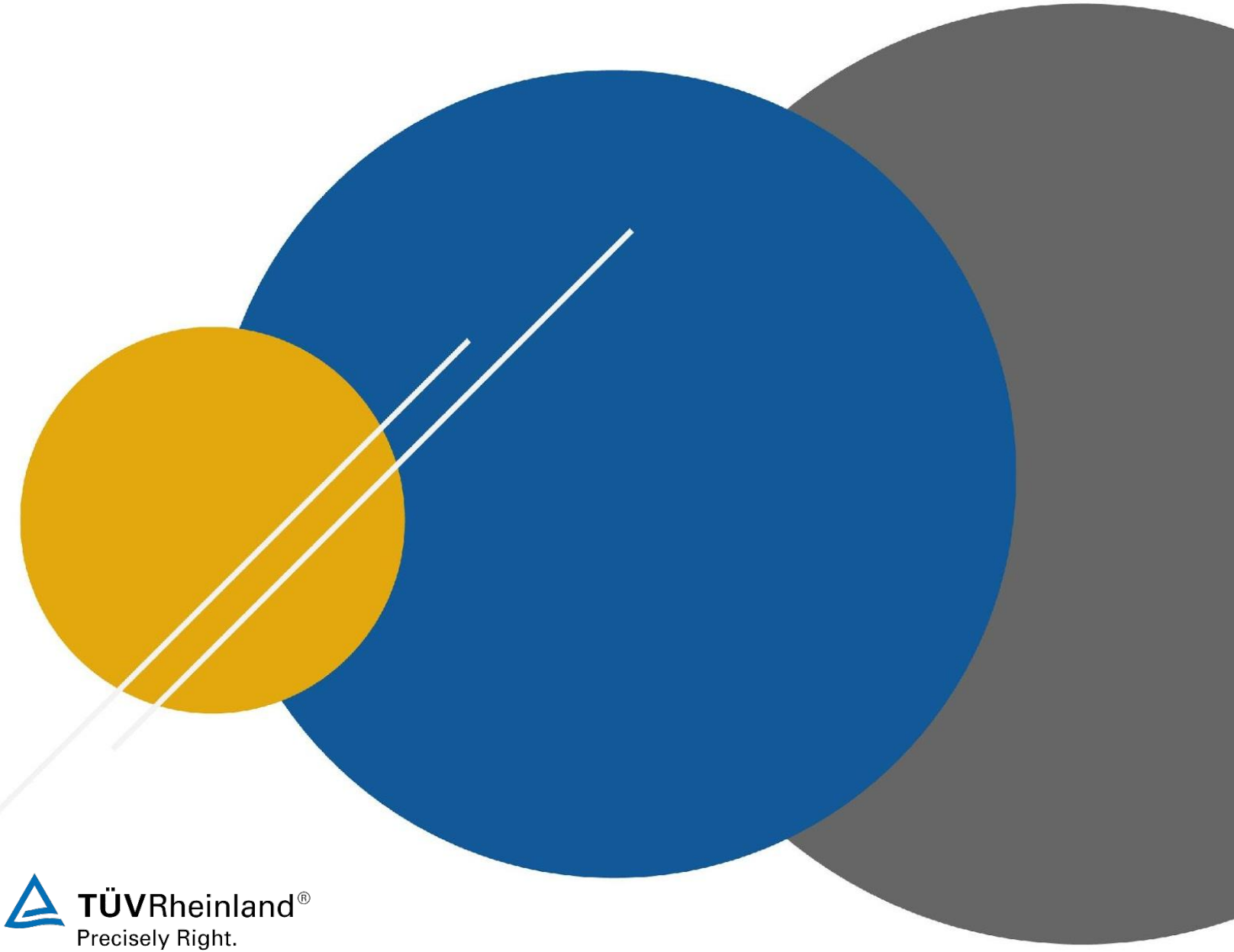
- DeepL Translator is a translation service launched in August 2017 by DeepL GmbH
- Promising to deliver 3x better results compared to Microsoft Translator and Google Translate

Use Case

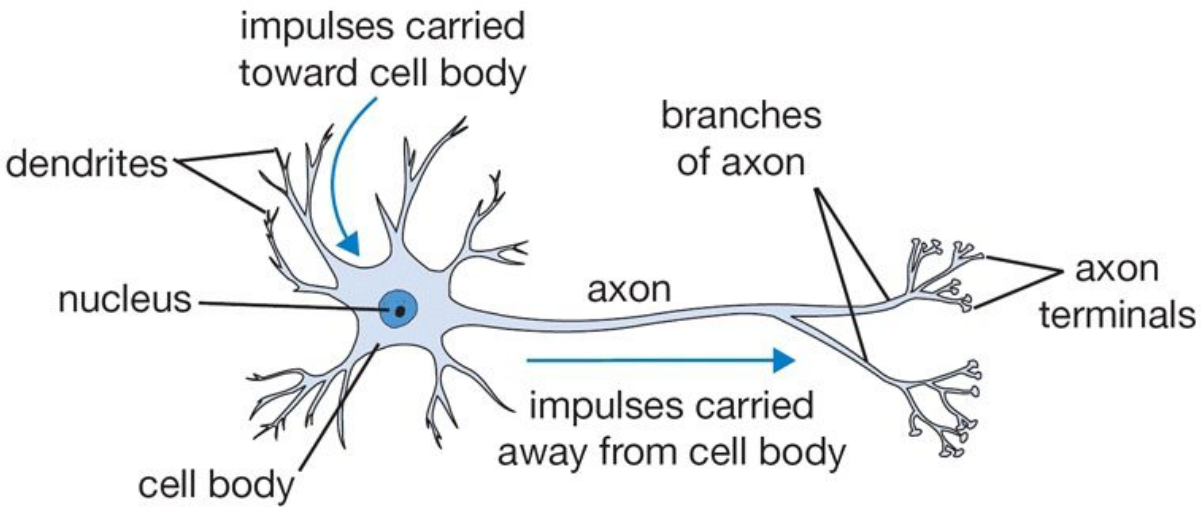
Colorization Images

- 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 2
Artificial Neural Network

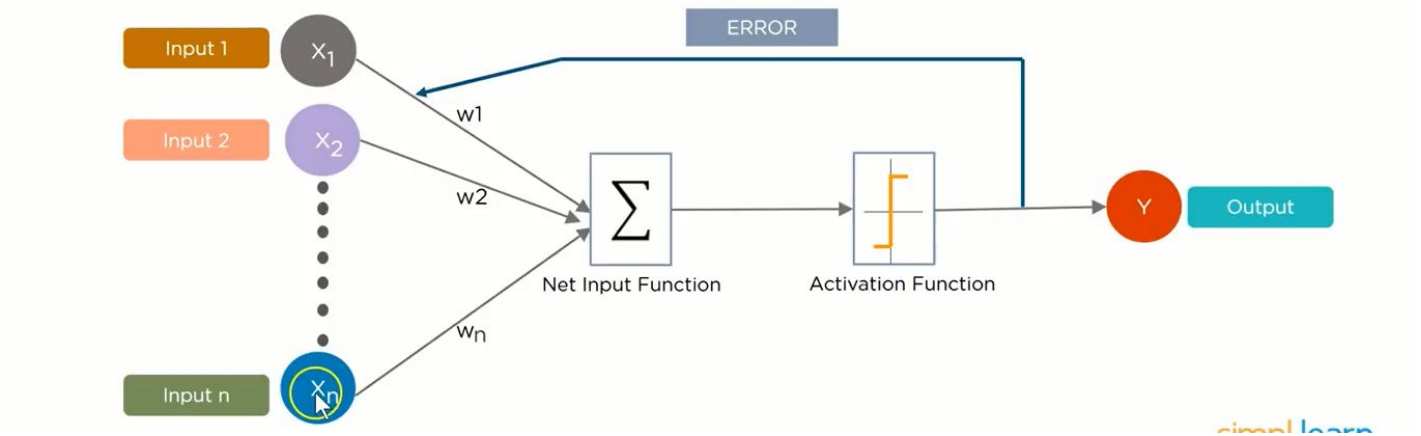


Anatomy of A Neuron



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

Component Of Artificial Neural Networks



- Input
- Weight
- Bias
- Activation Function

Net input = $x_1.w_1 + x_2.w_2 + x_3.w_3 = \sum x_i.w_i$

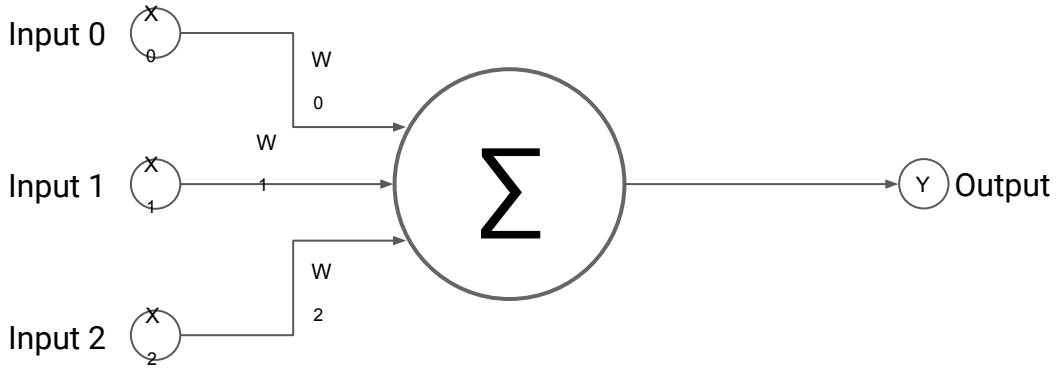
Or using simple matrix multiplication

[w1 w2 w3] [x1] $= x_1.w_1 + x_2.w_2 + x_3.w_3$

X2

x3

How It Works
Perceptron



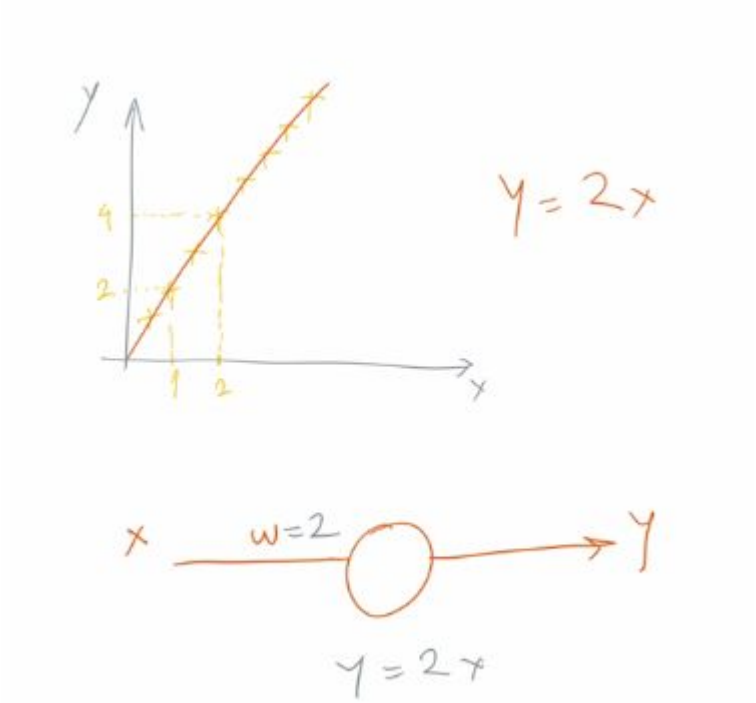
$$y = (x_0 w_0) + (x_1 w_1) + (x_2 w_2)$$

$$= \sum_i x_i \; w_i$$

$$= [w_0 \; w_1 \; w_2] \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

- 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

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

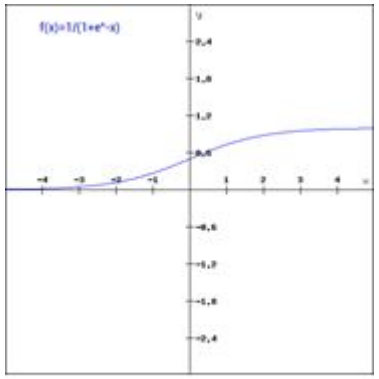
Linear function

Non-linear function

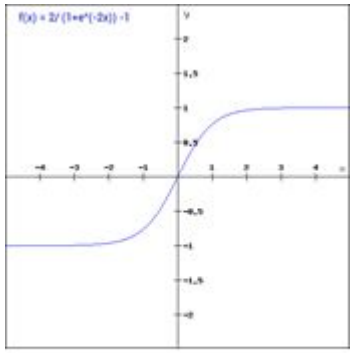
- Decides whether a neuron should be activated or not by calculating weighted sum and adding bias with it.
- Introduce non-linearity into the output of a neuron → making it capable to learn and perform more complex tasks
- If we do not use activation function, the output signal produced is only a linear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

Activation Functions

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



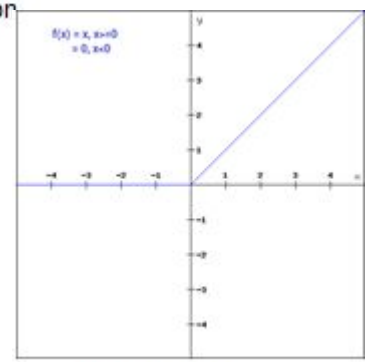
Sigmoid function



tanh function

Commonly used activation functions:

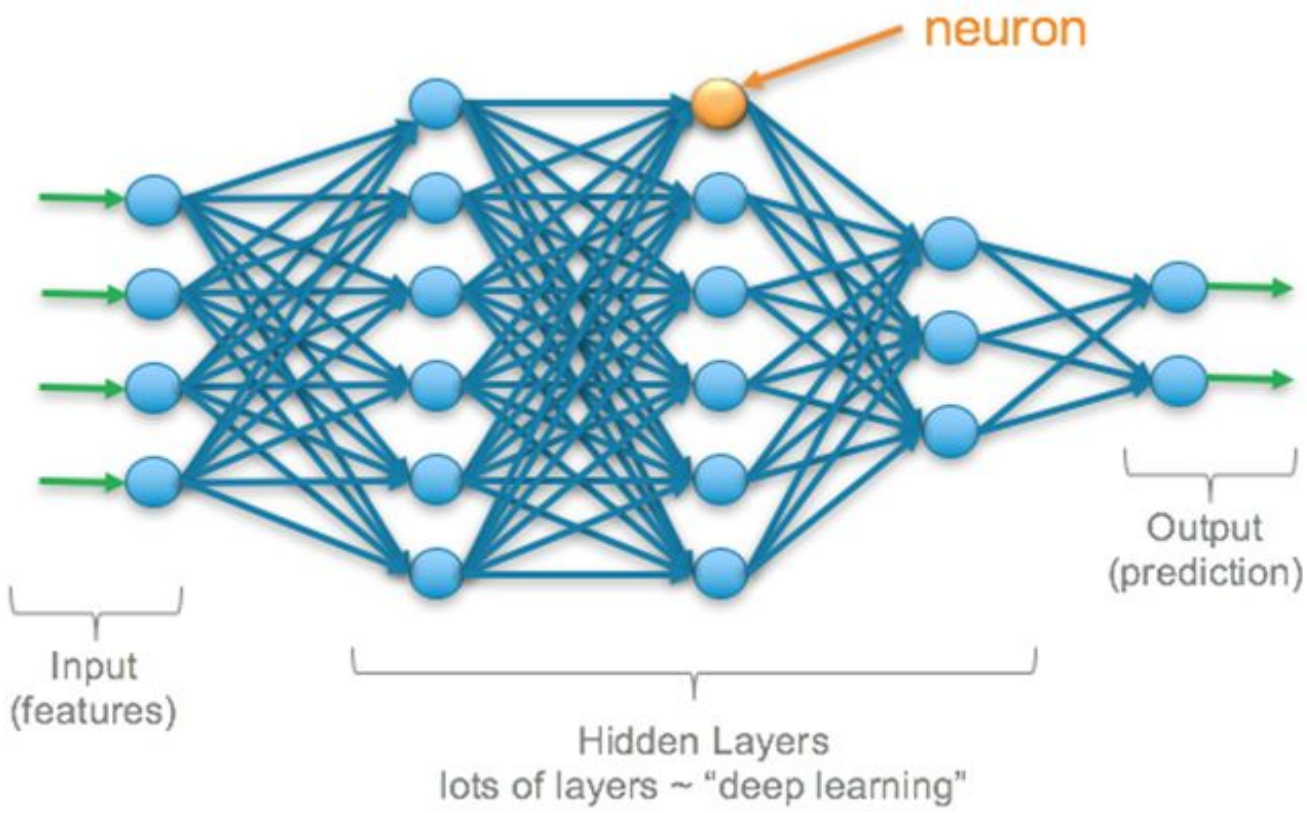
- Sigmoid function : $A = \frac{1}{1+e^{-x}}$
- Tanh function : $tanh(x) = \frac{2}{1+e^{-2x}} - 1$
- ReLU (Rectified Linear Unit) : A(x) = max(0,x)
- Softmax function : converts an array of values into an array of probabilities (0 - 1)



ReLu function

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

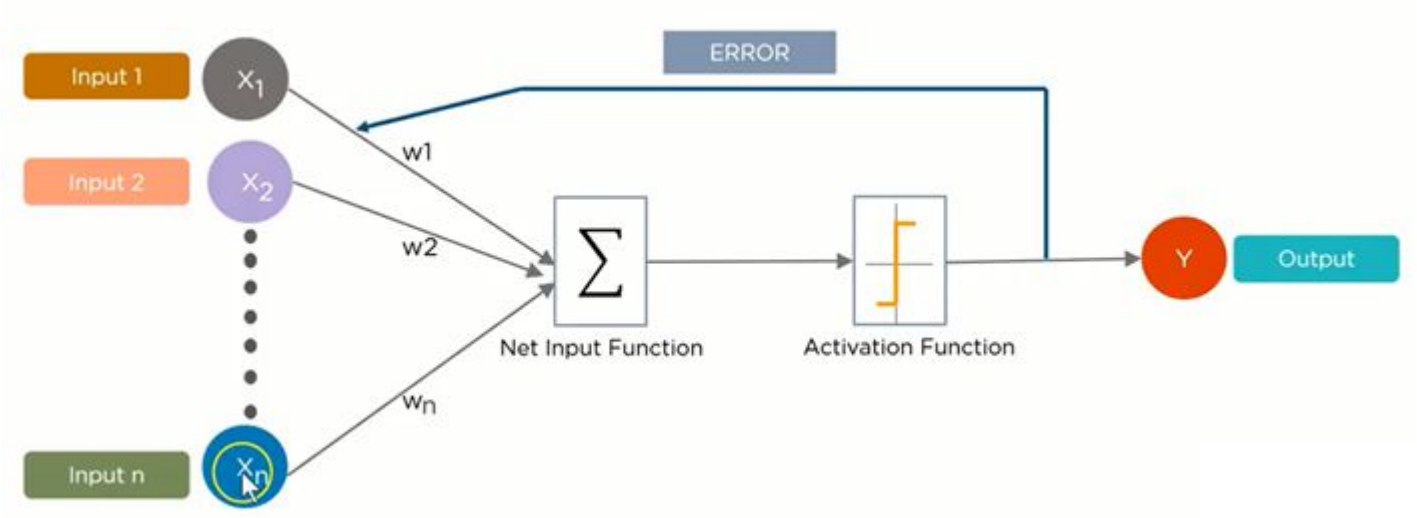
Multilayer Perceptron



y1 = P (cat|x)

y2 = P (dog|x)

Learning Rule - 2 Steps of Learning



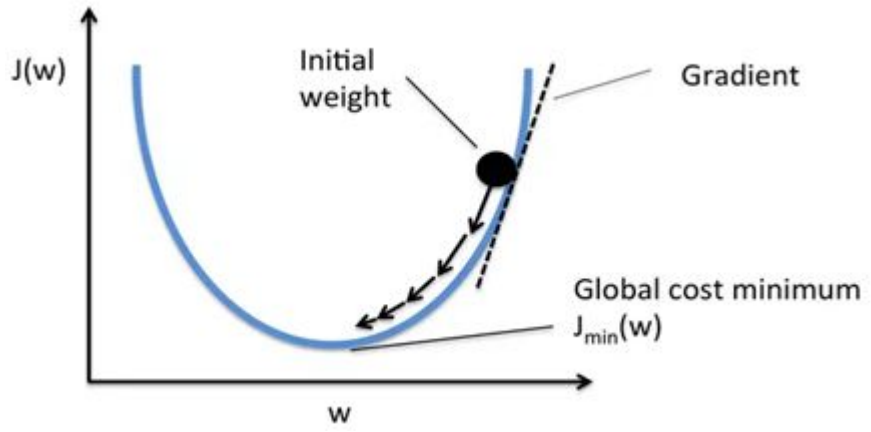
Forward propagation

- 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

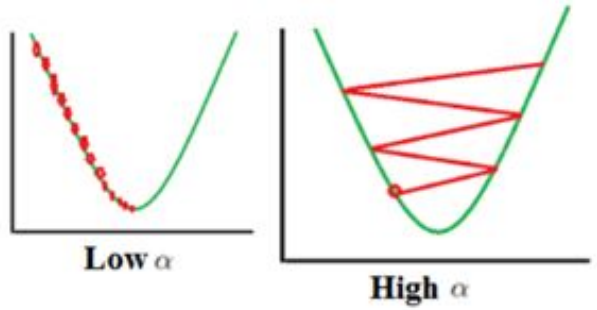
- finding the **derivative of the lost function** with respect to each weight
- update the weight by subtracting this value from the weight value.

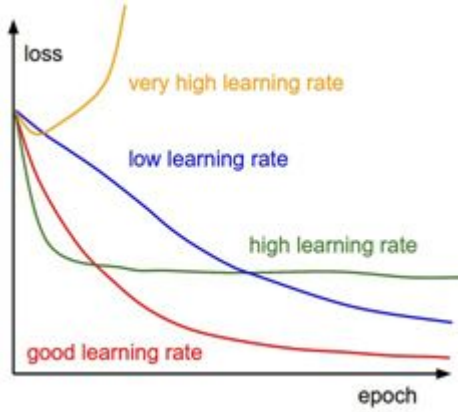
Loss Function and Gradient Descent



- 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 → **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





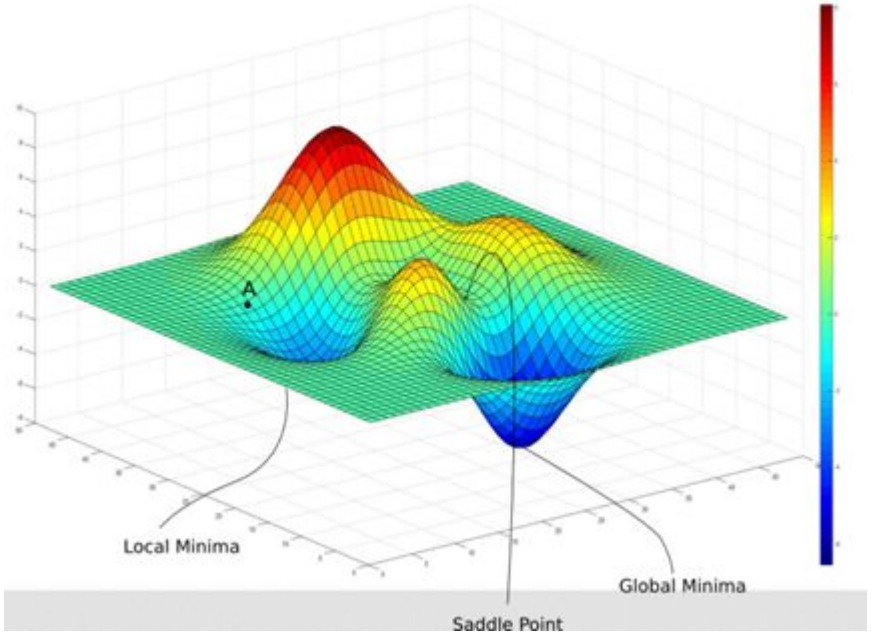
- The gradient told us the direction to change the weight.
- How much we must change is determined by another hyperparameter called learning rate or α
- Picking the value of α is crucial

  too small → training process too slow

  too big → diverging away from the minimum / overshoot
- We can pick α manually or using optimization technique that utilize adaptive learning rate (e.g. Adagrad, RMSProp, Adam)
- For example:

  $w' = w - \alpha \, (\partial E / \partial w)$ → α is learning rate

Gradient Descent Implementation

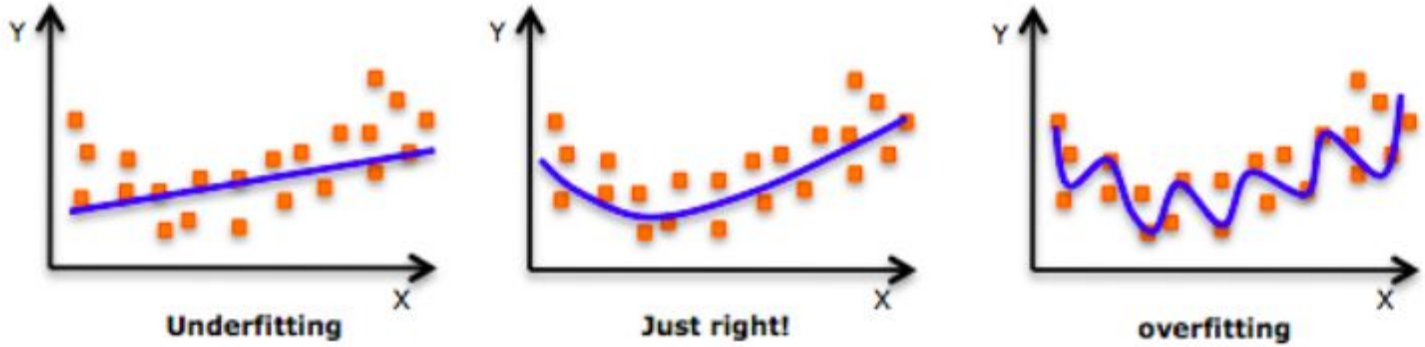


Challenges in Gradient Descent

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 + α ∑w2

  L1 → Loss = Loss + α ∑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

Chapter 3
Convolution Neural Network

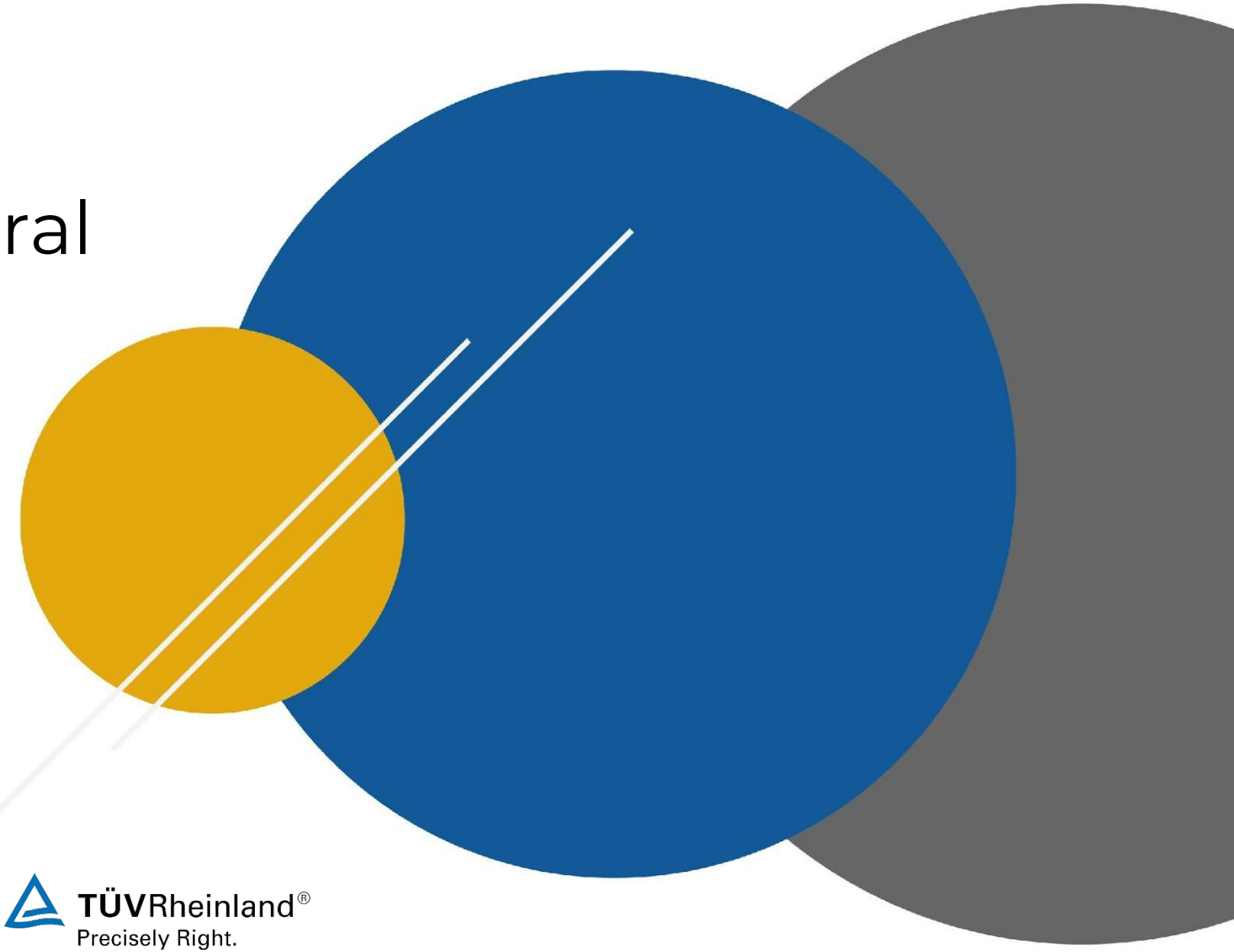
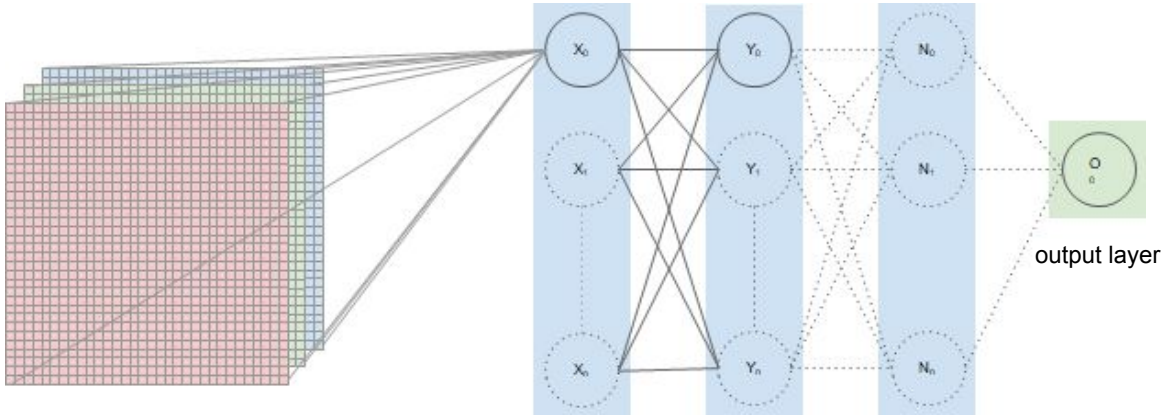


Image Processing in ANN

image: 32 x 32 pixel



pixel: 32 x 32 x 3 hidden layer 1 hidden layer 2 hidden layer N

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

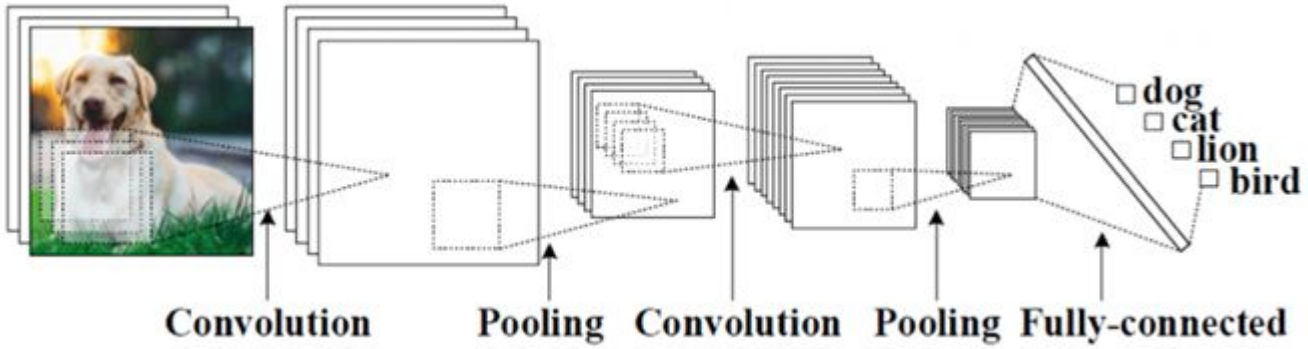
  E.g : CIFAR-10 images size 32x32x3 (32x32, 3 color channels - RGB)

  → 32*32*3 = 3072 weights in a single neuron in a first hidden layer

  For 200x200x3 image → 200*200*3 = **120,000** weights

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

Convolutional Neural Network Overview



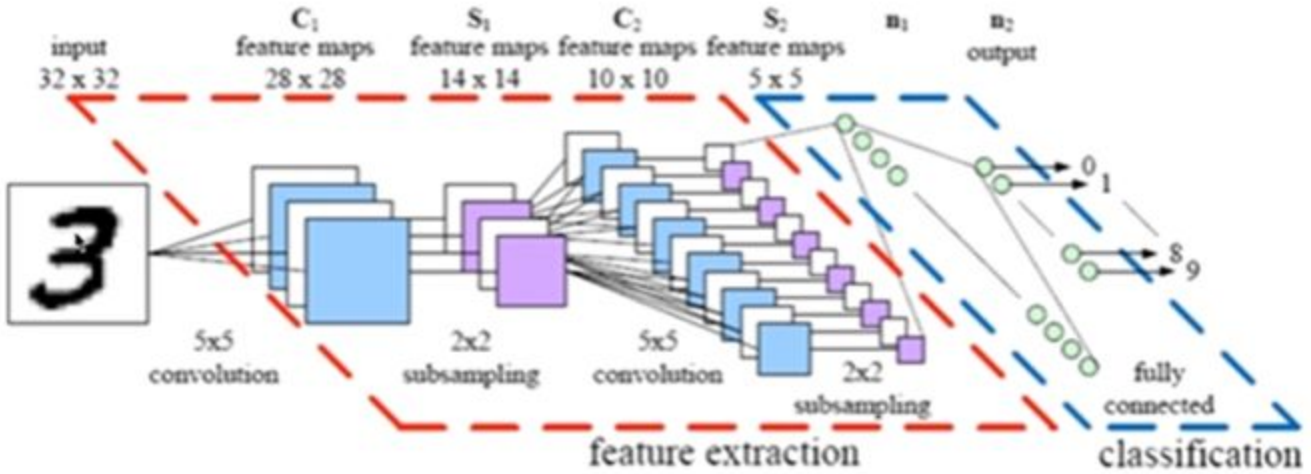
- A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network
- in 1998, Convolutional Neural Networks were introduced in a paper by Bengio, Le 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

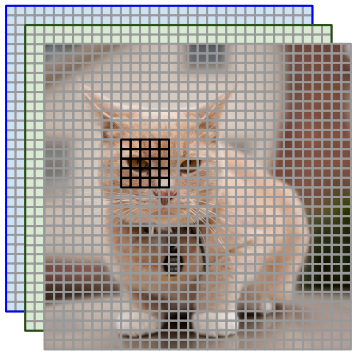
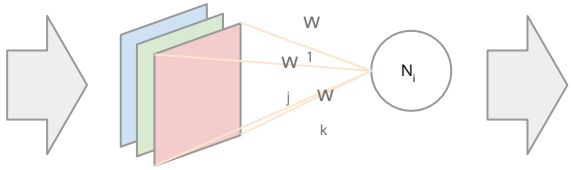
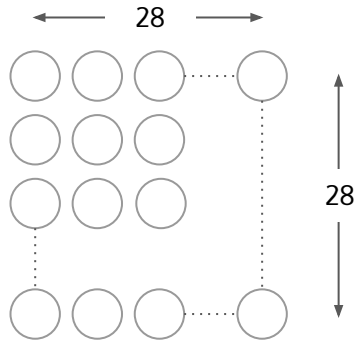


image: 32x32x3



Neuron on the first layer
(5x5x3 filter size)



Number of neuron on the first layer
(28x28)

- 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.

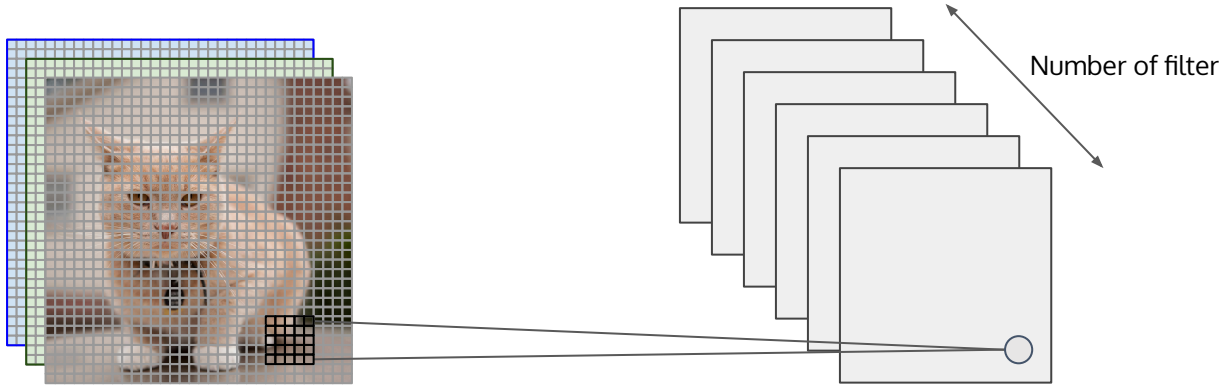
  For the CIFAR-10 data, we may have a **5x5x3** filter (5x5 matrix for 3 color channels)

  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

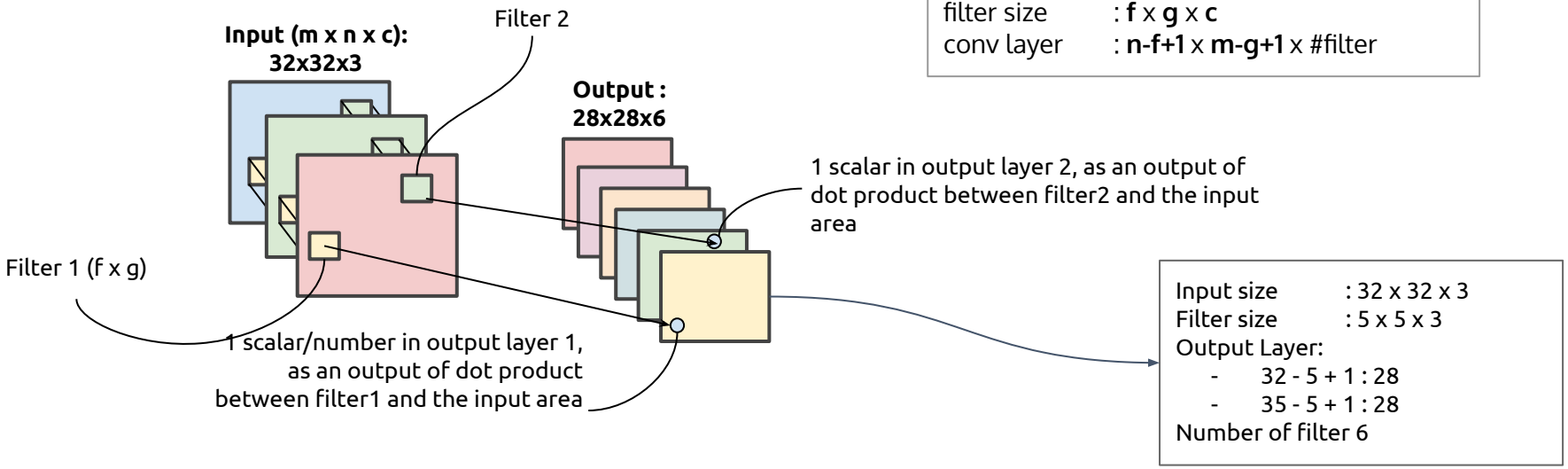
Convolution Layer Hyperparameter



Hyperparameters to set in the convolution layer:

- Filter size
- Number of filter
- Stride
- Padding

Number of Filters



input	: **m** x **n** x **c**
filter size	: **f** x **g** x **c**
conv layer	: **n-f+1** x **m-g+1** x #filter

Input size : 32 x 32 x 3
Filter size : 5 x 5 x 3
Output Layer:
- 32 - 5 + 1 : 28
- 35 - 5 + 1 : 28

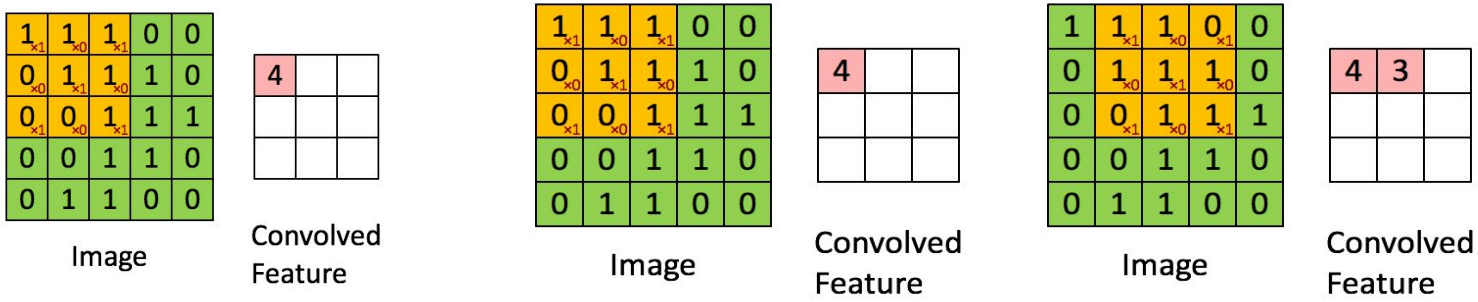
Number of filter 6

- 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 5x5x3 for our example, the output will be 28x28x6 (6 layers of 28x28)

Stride

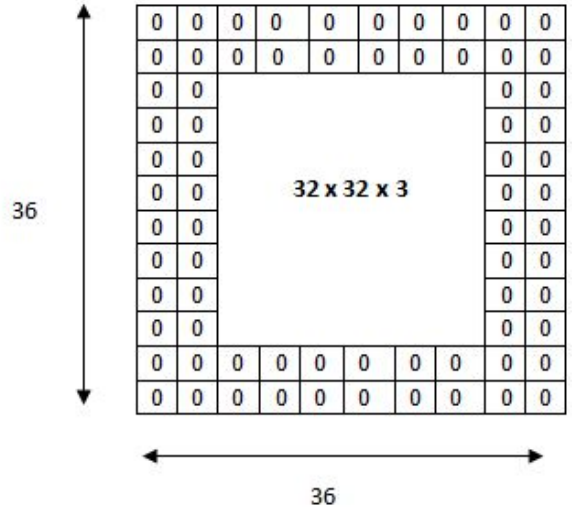
- Stride = how much we shift the filter at a time. The bigger the stride, the smaller the output.
- Formula to calculate output width and height = **(W−F)/S + 1**

  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 = 10x10
- Example for 7x7 input with 3x3 filter :



Padding

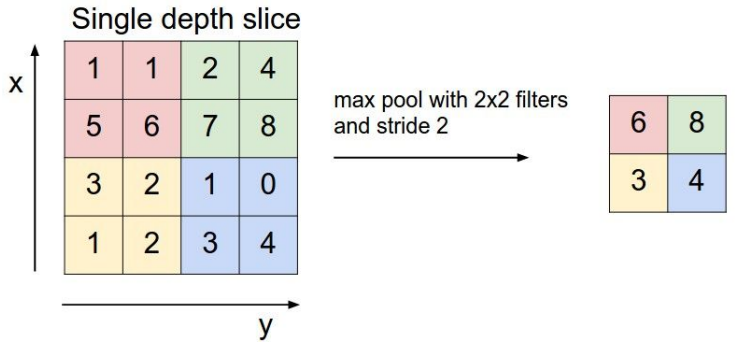
- Used to preserve the size of the output.
- The formula : (W − F + 2P) / S + 1, where P = padding
- If we want to preserve the output spatial size to 32x32 with our 5x5 filter, we can use padding=2 and stride=1, so output = (32 - 5 + 4) + 1 = 32



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 x3 output volume.

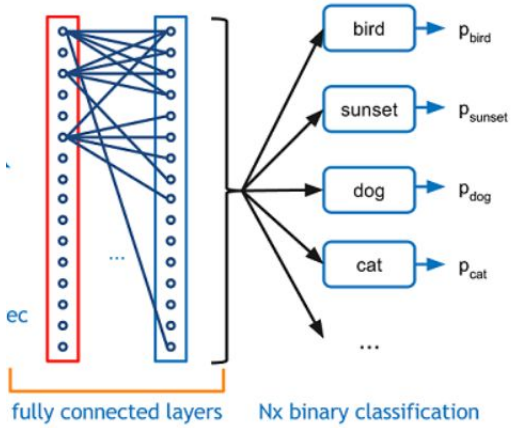
Pooling Layer

- 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=3x3 ,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



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

Requirement:

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

Lab 01
Architecture