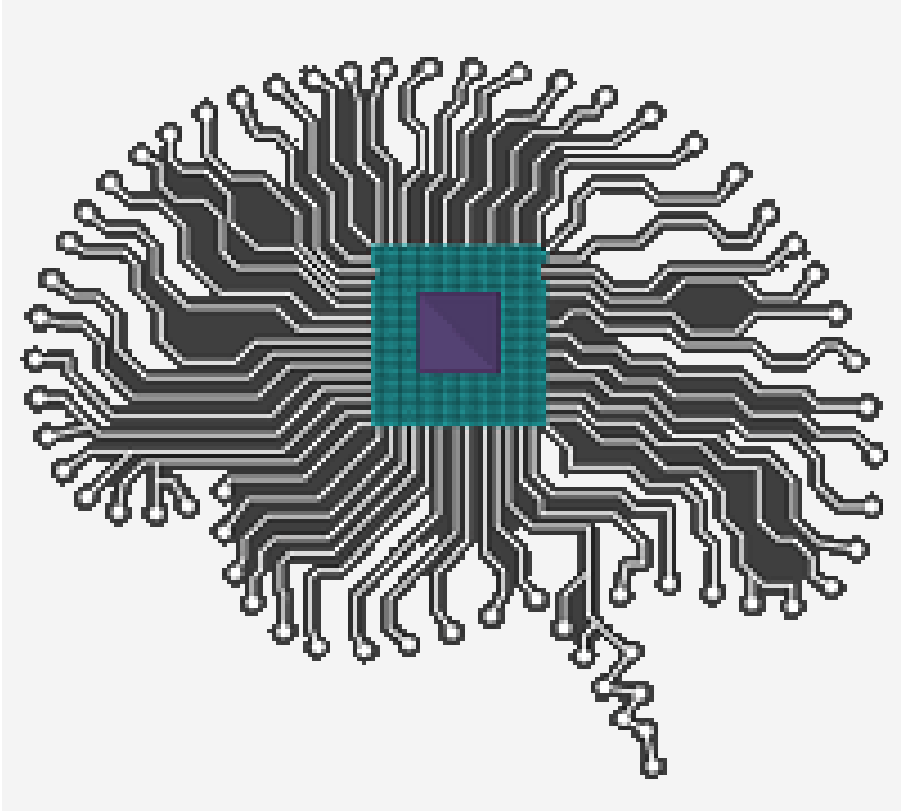


Artificial Neural Networks

By: Sajad Ahmadian

Kermanshah University of Technology

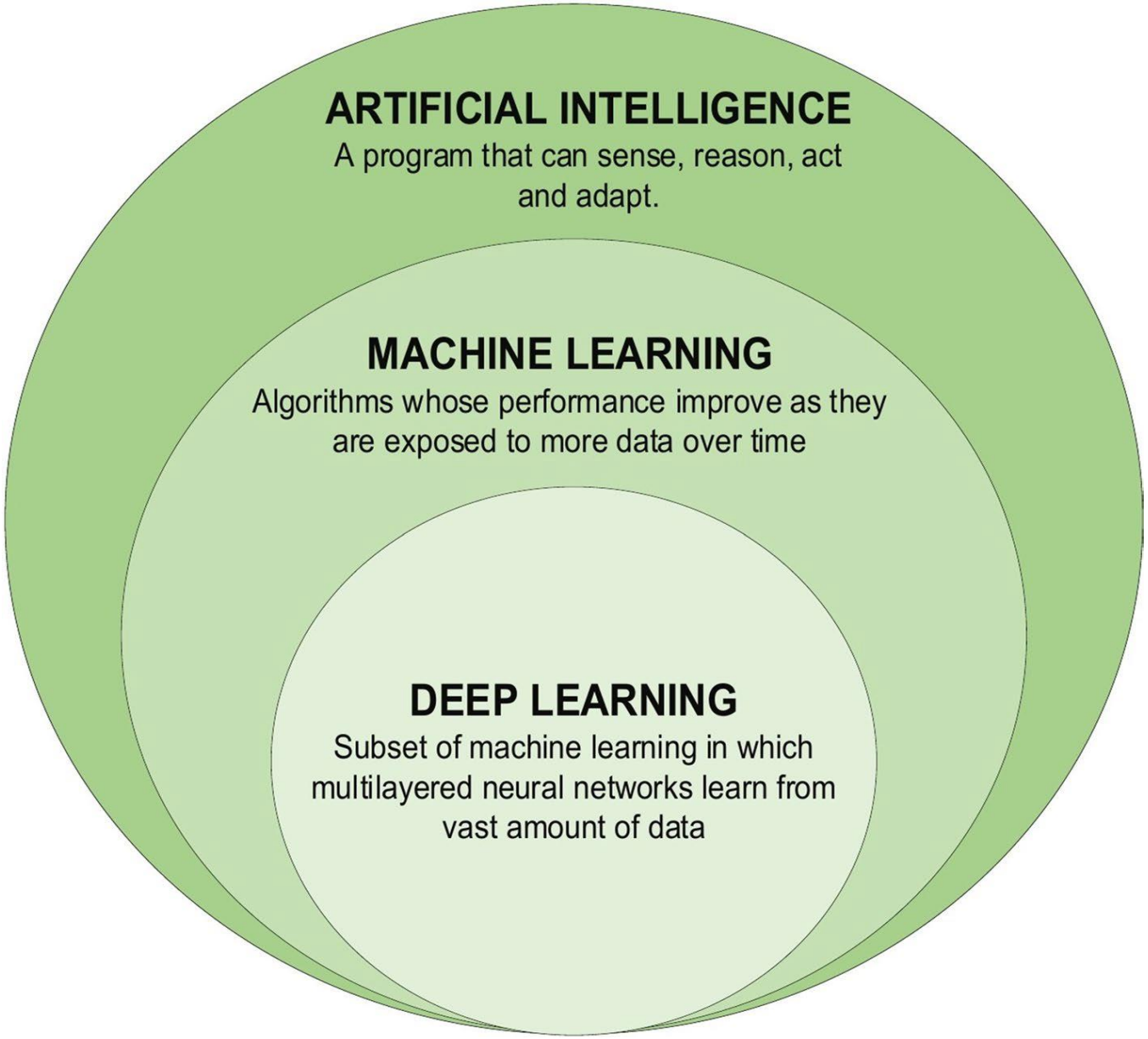
Deep learning

- In the last few years, the deep learning (DL) computing paradigm has been deemed the Gold Standard in the machine learning (ML) community.
- Moreover, it has gradually become the most widely used computational approach in the field of ML.
- One of the benefits of DL is the ability to learn massive amounts of data.
- More importantly, DL has outperformed well-known ML techniques in many domains, e.g., cybersecurity, natural language processing, bioinformatics, robotics and control, medical information processing, etc.

Deep learning

- Another name for DL is representation learning (RL).
- The effectiveness of an ML algorithm is highly dependent on the integrity of the input-data representation.
- A suitable data representation provides an improved performance when compared to a poor data representation.
- Feature extraction is achieved in an automatic way throughout the DL algorithms.
- These algorithms have a multi-layer data representation architecture, in which the first layers extract the low-level features while the last layers extract the high-level features.

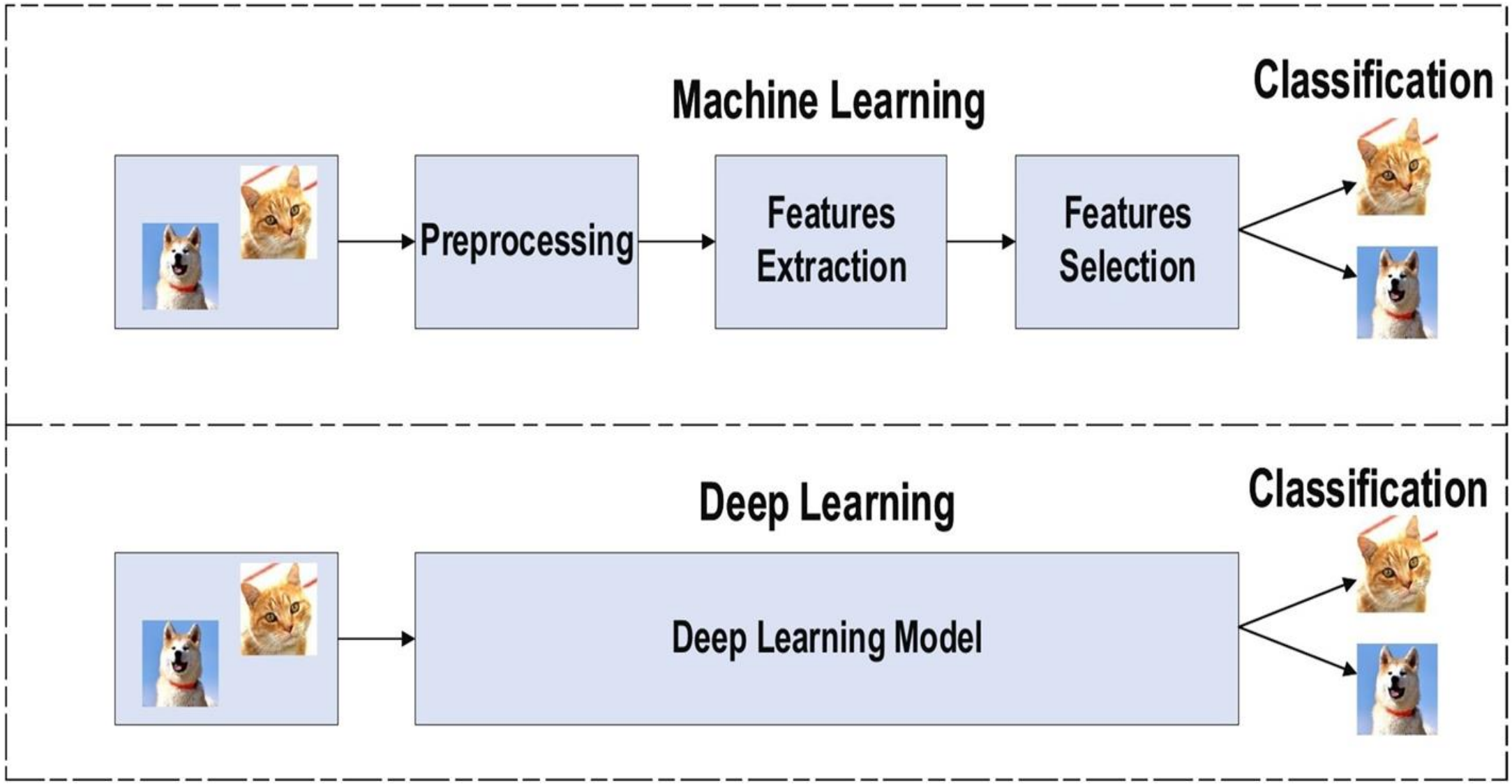
Deep learning family



The difference between deep learning and traditional machine learning

- Achieving the classification task using conventional ML techniques requires several sequential steps, specifically pre-processing, feature extraction, feature selection, learning, and classification.
- Feature selection has a great impact on the performance of ML techniques. Biased feature selection may lead to incorrect discrimination between classes.
- DL has the ability to automate the learning of feature sets for several tasks, unlike conventional ML methods.
- DL enables learning and classification to be achieved in a single shot.

The difference between deep learning and traditional machine learning



When to apply deep learning

1. Cases where human experts are not available.
2. Cases where humans are unable to explain decisions made using their expertise (language understanding, medical decisions, and speech recognition).
3. Cases where the problem solution updates over time (price prediction, stock preference, weather prediction, and tracking).
4. Cases where solutions require adaptation based on specific cases (personalization, biometrics).
5. Cases where size of the problem is extremely large and exceeds our inadequate reasoning abilities (sentiment analysis, calculation webpage ranks).

Why deep learning?

1. **Universal Learning Approach:** Because DL has the ability to perform in approximately all application domains, it is sometimes referred to as universal learning.
2. **Robustness:** In general, precisely designed features are not required in DL techniques. Instead, the optimized features are learned in an automated fashion related to the task under consideration. Thus, robustness to the usual changes of the input data is attained.
3. **Generalization:** Different data types or different applications can use the same DL technique.
4. **Scalability:** DL is highly scalable. ResNet was invented by Microsoft, comprises 1202 layers and is frequently applied at a supercomputing scale.

Classification of DL approaches

- **Deep supervised learning**
  - This technique deals with labeled data.
  - When considering such a technique, there is a collection of inputs and resultant outputs (x_t, y_t).
  - The purpose is to learn a function $\hat{y}_t = f(x_t)$ to predict the label (class) of a given input data x_t.
- **Deep semi-supervised learning**
  - The learning process is based on semi-labeled datasets.
  - The labels for a portion of whole data are known but others do not have any labels.
- **Deep unsupervised learning**
  - This technique makes it possible to implement the learning process in the absence of available labeled data (i.e. no labels are required).
- **Deep reinforcement learning**
  - Reinforcement Learning operates on interacting with the environment, while supervised learning operates on provided sample data.

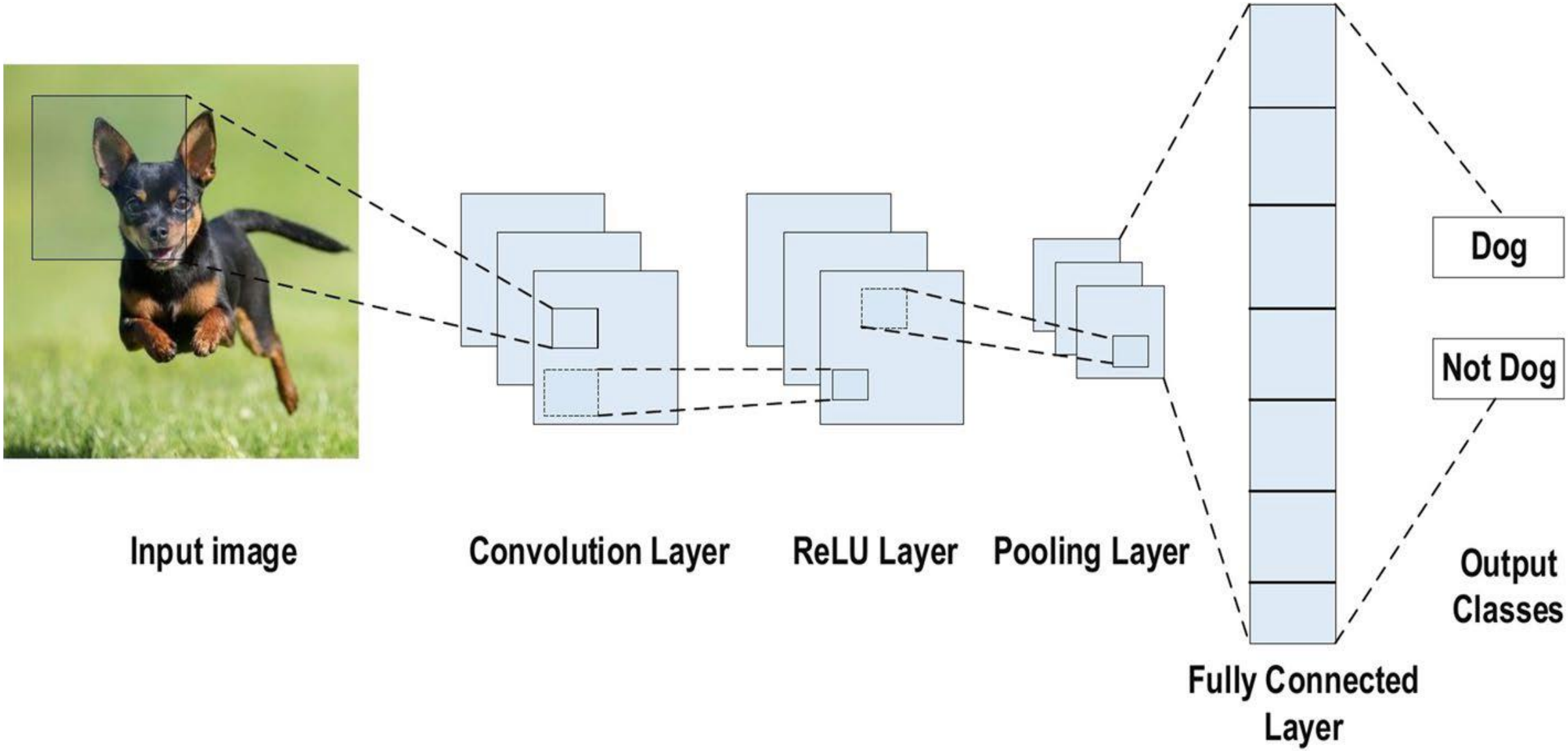
Convolutional Neural Networks

- Convolutional neural network (CNN or ConvNet) is a special type of multilayer neural network or deep learning architecture inspired by the visual system of living beings.
- CNN is very much suitable for different fields of computer vision and natural language processing.
- CNN has deep feed-forward architecture and has amazing generalizing ability as compared to other networks with fully connected layers, it can learn highly abstracted features of objects especially spatial data and can identify them more efficiently.
- A deep CNN model consists of a finite set of processing layers that can learn various features of input data (e.g. image) with multiple level of abstraction.
- The initiatory layers learn and extract the high level features (with lower abstraction), and the deeper layers learn and extract the low level features (with higher abstraction).

Convolutional Neural Networks

- **Why Convolutional Neural Networks is more considerable over other classical neural networks in the context of computer vision?**
  - One of the main reason for considering CNN in such case is the weight sharing feature of CNN, that reduces the number of trainable parameters in the network, which helped the model to avoid overfitting and as well as to improve generalization.
  - In CNN, the classification layer and the feature extraction layers learn together, that makes the output of the model more organized and makes the output more dependent to the extracted features.
  - The implementation of a large network is more difficult by using other types of neural networks rather than using Convolutional Neural Networks.

An example of CNN architecture for image classification



Network Layers in CNN

- **Convolutional Layer**
  - Convolutional layer is the most important component of any CNN architecture.
  - It contains a set of convolutional kernels (also called filters), which gets convolved with the input image (N-dimensional metrics) to generate an output feature map.
  - CNN uses a set of multiple filters in each convolutional layer so that each filter can extract different types of features.

Network Layers in CNN

- **What is a kernel?**
  - A kernel can be described as a grid of discrete values or numbers, where each value is known as the weight of this kernel.
  - During the starting of training process of a CNN model, all the weights of a kernel are assigned with random numbers (different approaches are also available for initializing the weights).
  - Then, with each training epoch, the weights are tuned and the kernel learned to extract meaningful features.

0	1
-1	2

Example of a 2×2 kernel

Network Layers in CNN

- **What is Convolution Operation?**
  - Unlike other classical neural networks (where the input is in a vector format), in CNN, the input is a multi-channeled image (e.g. for RGB image, it is 3 channeled and for Gray-Scale image, it is single channeled).

RGB

Gray-Scale

Network Layers in CNN

- **What is Convolution Operation?**
  - To understand the convolution operation, we take a gray-scale image of 4×4 dimension and a 2×2 kernel with randomly initialized weights.
  - In convolution operation, we take the 2×2 kernel and slide it over all the complete 4×4 image horizontally as well as vertically and along the way we take the dot product between kernel and input image by multiplying the corresponding values of them and sum up all values to generate one scaler value in the output feature map.
  - This process continues until the kernel can no longer slide further.

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

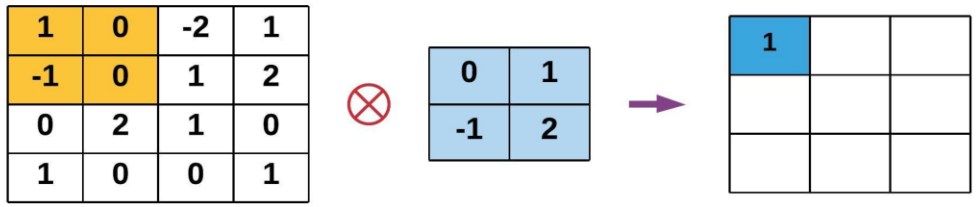
An 4×4 Gray-Scale image

0	1
-1	2

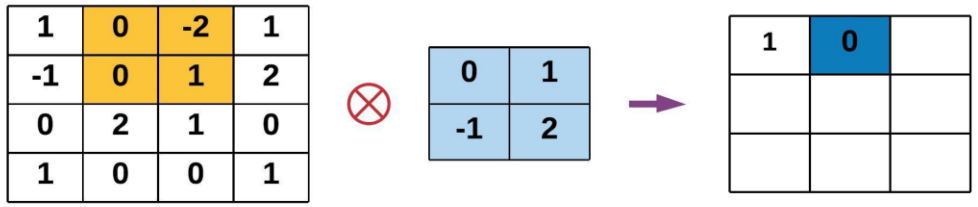
A kernel of size 2×2

Illustrating the first 5 steps of convolution operation

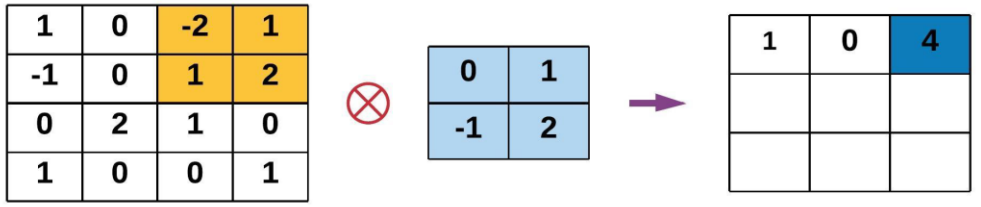
Step-1



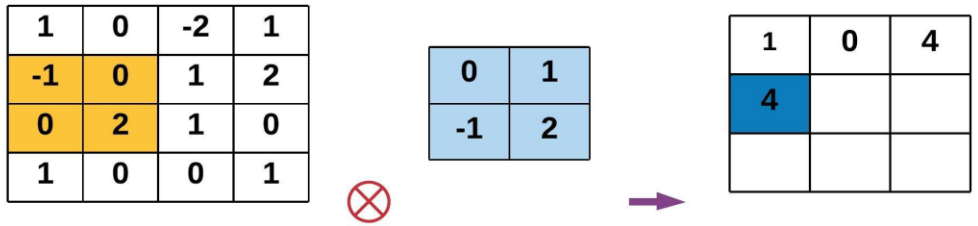
Step-2



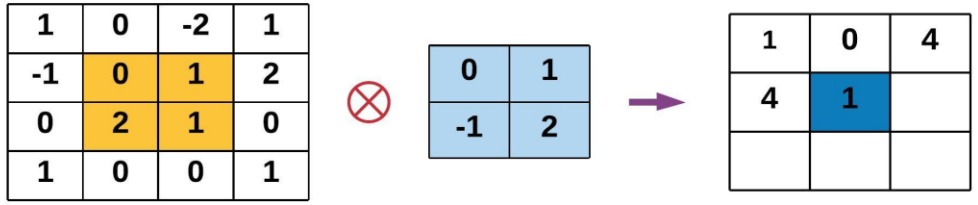
Step-3



Step-4



Step-5



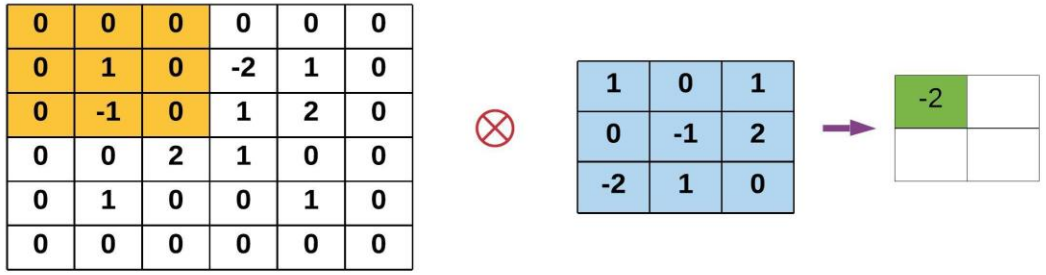
1	0	4
4	1	1
1	1	2

The final feature map after the complete convolution operation

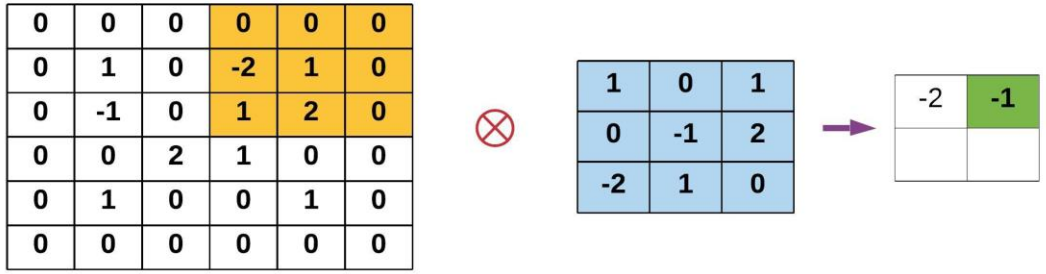
Network Layers in CNN

- In the above example, we apply the convolution operation with no padding to the input image and with stride (i.e. the taken step size along the horizontal or vertical position) of 1 to the kernel.
- But we can use other stride value (rather than 1) in convolution operation. The noticeable thing is if we increase the stride of the convolution operation, it resulted in lower-dimensional feature map.
- The padding is important to give border size information of the input image more importance, otherwise without using any padding the border side features are gets washed away too quickly.
- The padding is also used to increase the input image size, as a result the output feature map size also gets increased.

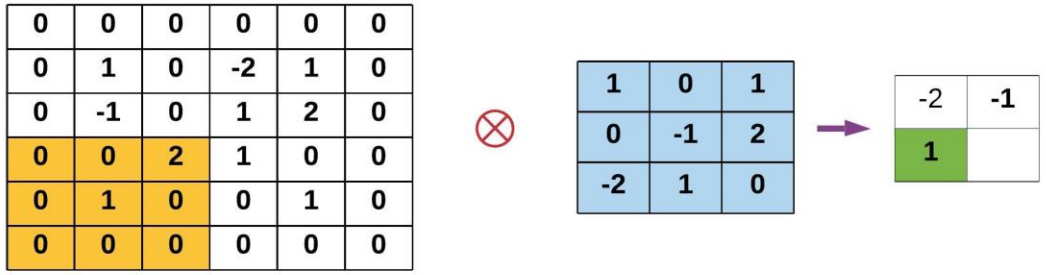
Step-1



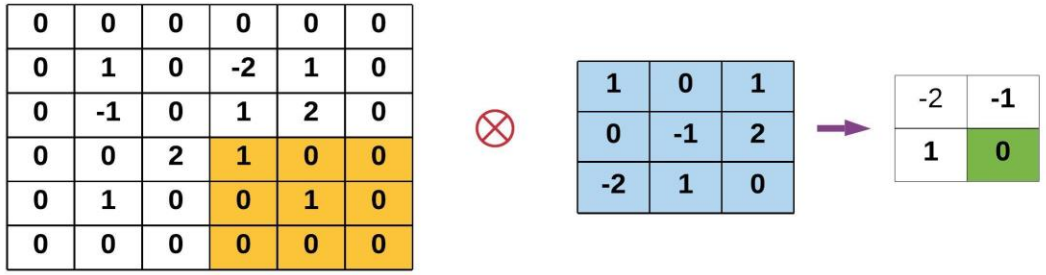
Step-2 :



Step-3 :



Step-4 :



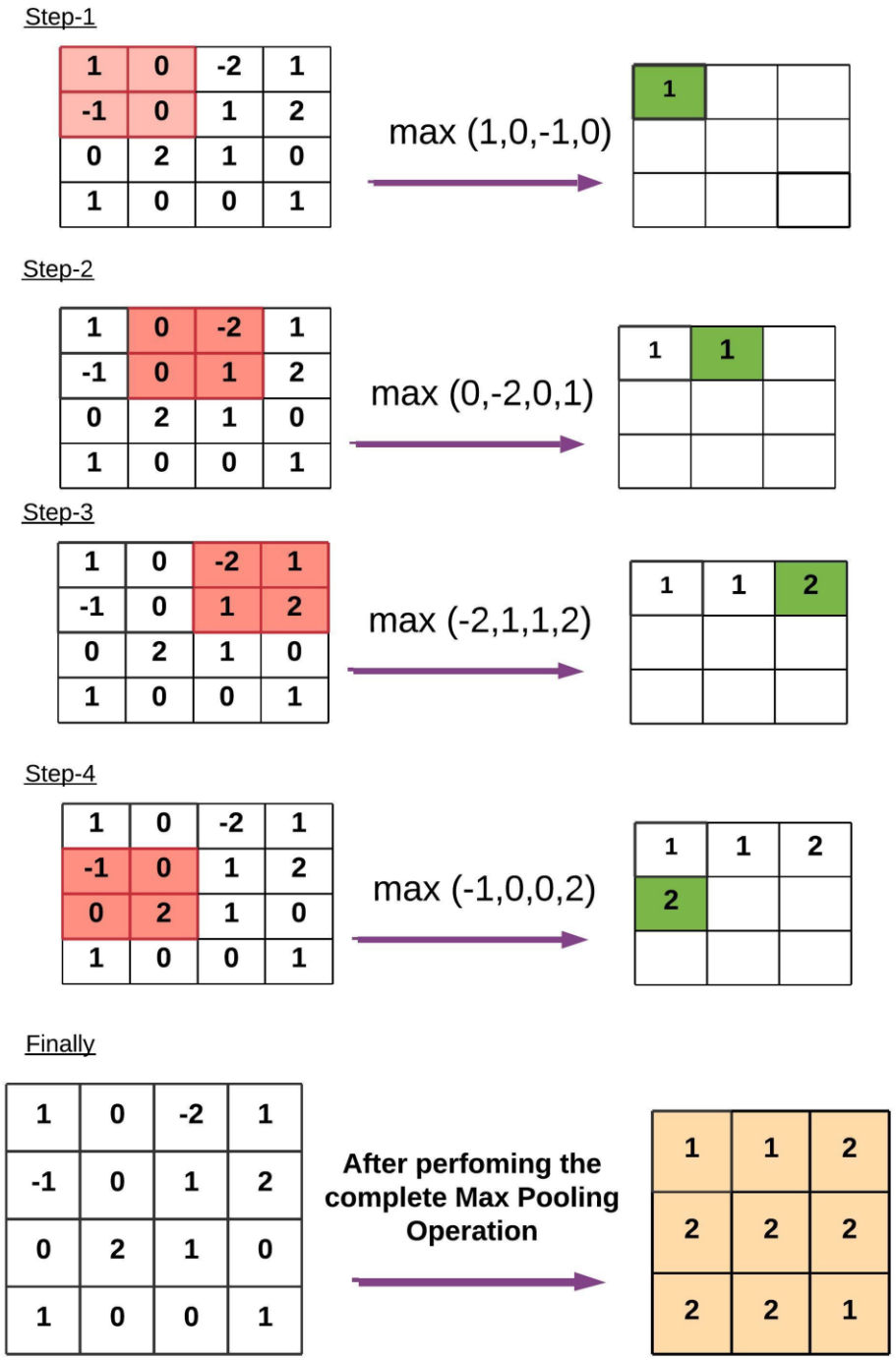
An example by showing the convolution operation with Zero-padding and 3 stride value.

Network Layers in CNN

- **Main advantages of convolution layers:**
  - **Sparse Connectivity:** In a fully connected neural network, each neuron of one layer connects with each neuron of the next layer but in CNN, small number of weights are present between two layers. As a result, the number of connections or weights we need is small, and the amount of memory to store those weights is also small, so it is memory efficient. Also, the dot (.) operation is computationally cheaper than matrix multiplication.
  - **Weight Sharing:** In CNN, no dedicated weights are present between two neurons of adjacent layers instead of all weights work with each and every pixel of the input matrix. Instead of learning new weights for every neuron, we can learn a set of weights for all inputs and this drastically reduces the training time as well as the other costs.

Network Layers in CNN

- **Pooling Layer:**
  - The pooling layers are used to sub-sample the feature maps (produced after convolution operations), i.e. it takes the larger size feature maps and shrinks them to lower sized feature maps.
  - While shrinking the feature maps it always preserve the most dominant features (or information) in each pool steps.
  - The pooling operation is performed by specifying the pooled region size and the stride of the operation, similar to convolution operation.
  - There are different types of pooling techniques used in different pooling layers such as max pooling, min pooling, average pooling, gated pooling, tree pooling, etc. Max Pooling is the most popular and mostly used pooling technique.
  - The main drawback of pooling layer is that it sometimes decreases the overall performance of CNN. The reason behind this is that pooling layer helps CNN to find whether a specific feature is present in the given input image or not without caring about the correct position of that feature.



An example of max-pooling operation, where the size of the pooling region is 2*2 and the stride is 1.

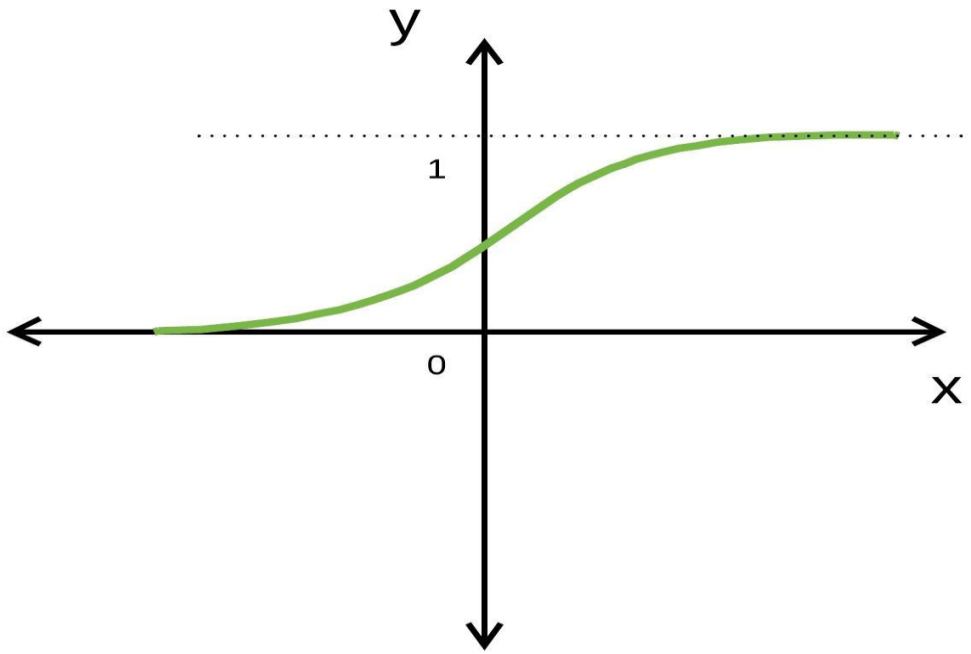
Activation Functions (Non-Linearity)

- The main task of any activation function in any neural network-based model is to map the input to the output, where the input value is obtained by calculating the weighted sum of neuron's input and further adding bias with it (if there is a bias).
- In CNN architecture, after each learnable layers (layers with weights, i.e. convolutional and fully connected layers) non-linear activation layers are used.
- The non-linearity behavior of those layers enables the CNN model to learn more complex things and manage to map the inputs to outputs non- linearly.
- The important feature of an activation function is that it should be differentiable in order to enable error backpropagation to train the model.

Activation Functions (Non-Linearity)

- **Sigmoid:** The sigmoid activation function takes real numbers as its input and bind the output in the range of [0,1]. The curve of the sigmoid function is of `S' shaped. The mathematical representation of sigmoid is:

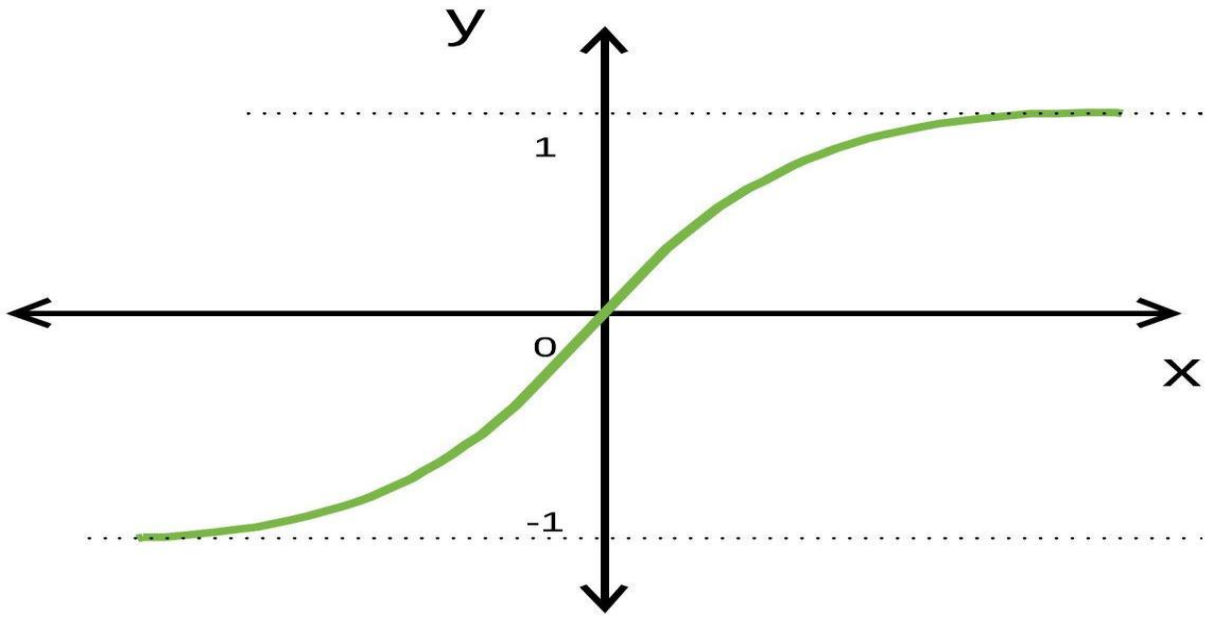
$$f(x)_{sigm} = \frac{1}{1+e^{-x}}$$



Activation Functions (Non-Linearity)

- **Tanh:** The Tanh activation function is used to bind the input values (real numbers) within the range of [-1, 1]. The mathematical representation of Tanh is:

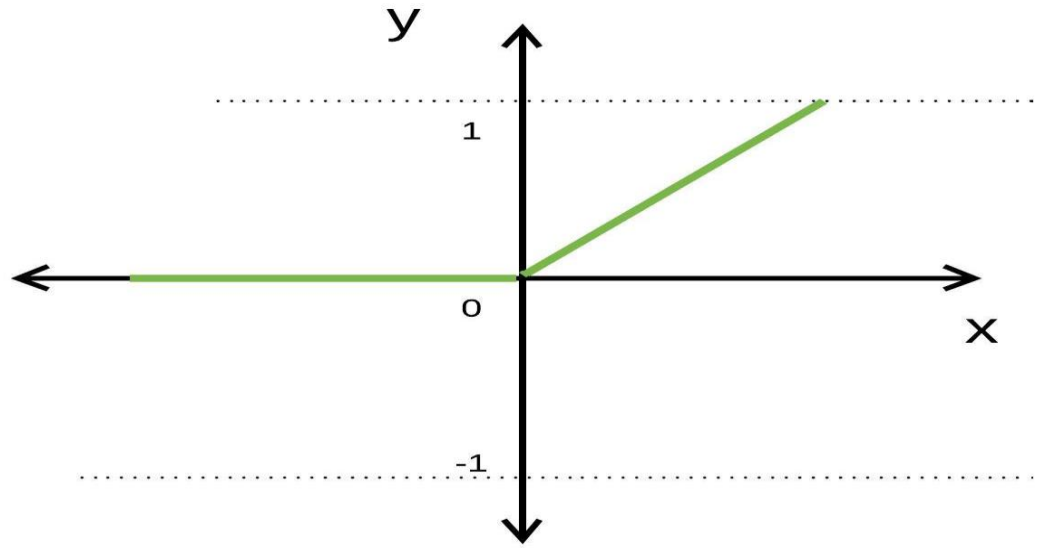
$$f(x)_{tanh} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Activation Functions (Non-Linearity)

- **ReLU:** The Rectifier Linear Unit (ReLU) is the most commonly used activation function in Convolutional Neural Networks. It is used to convert all the input values to positive numbers. The mathematical representation of ReLU is:

$$f(x)_{ReLU} = \max(0, x)$$



Activation Functions (Non-Linearity)

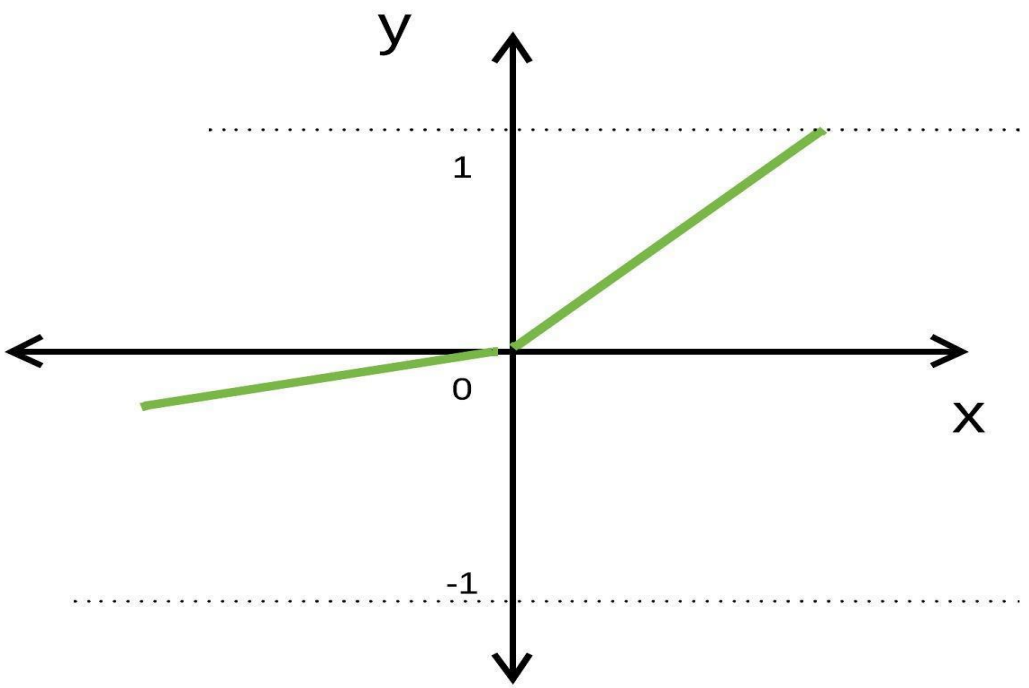
- The advantage of ReLU is that it requires very minimal computation load compared to others.
- But sometimes there may occur some major problems in using ReLU activation function:
  - For example, consider a larger gradient is flowing during error back-propagation algorithm, and when this larger gradient is passed through a ReLU function, it may cause the weights to be updated in such a way that the neuron never gets activated again.
  - This problem is known as the Dying ReLU problem.
  - To solve these types of problems there are some variants of ReLU, some of them are discussed below.

Activation Functions (Non-Linearity)

- **Leaky ReLU:** Unlike ReLU, a Leaky ReLU activation function does not ignore the negative inputs completely, rather than it down-scaled those negative inputs. Leaky ReLU is used to solve Dying ReLU problem. The mathematical representation of Leaky ReLU is:

$$f(x)_{LeakyReLU} = \begin{cases} x, & \text{if } x > 0 \\ mx, & x \leq 0 \end{cases}$$

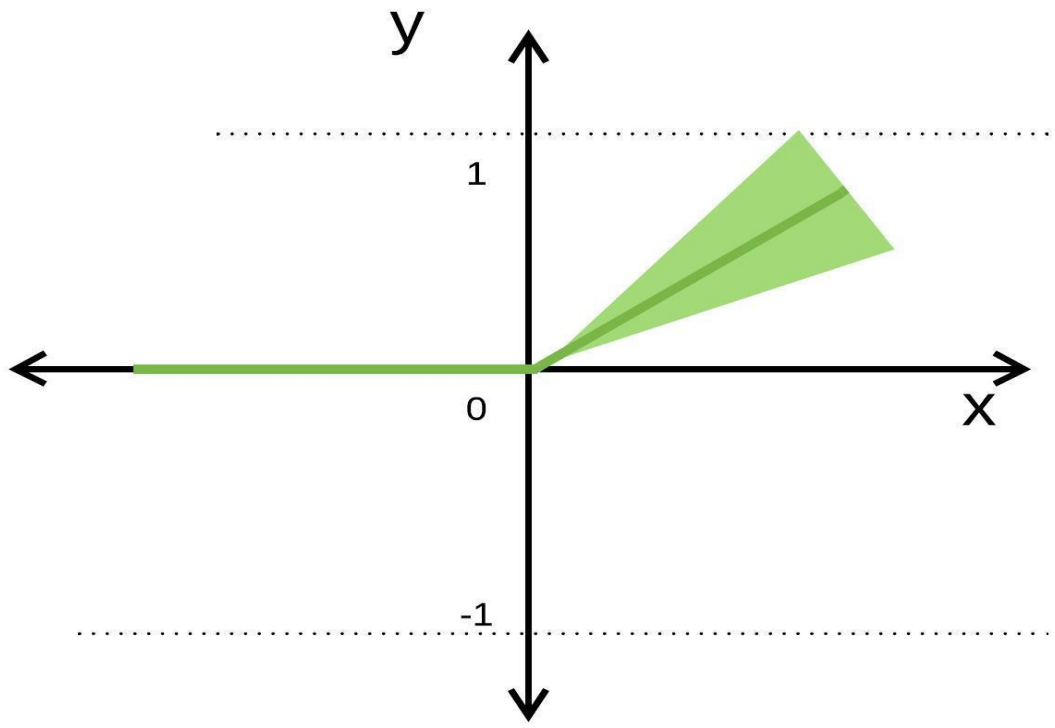
where m is a constant, called leak factor and generally it set to a small value (like 0.001).



Activation Functions (Non-Linearity)

- **Noisy ReLU:** Noisy ReLU uses Gaussian distribution to make ReLU noisy. The mathematical representation of Noisy ReLU is:

$$f(x)_{NoisyReLU} = \max(0, x + Y), \quad with\ Y \sim N(0, \sigma(x))$$



Activation Functions (Non-Linearity)

- **Parametric Linear Units:** It is almost similar to Leaky ReLU, but here the leak factor is tuned during the model training process. The mathematical representation of Parametric Linear Units is:

$$f(x)_{ParametricLinearUnits} = \begin{cases} x, & \text{if } x > 0 \\ ax, & x \leq 0 \end{cases}$$

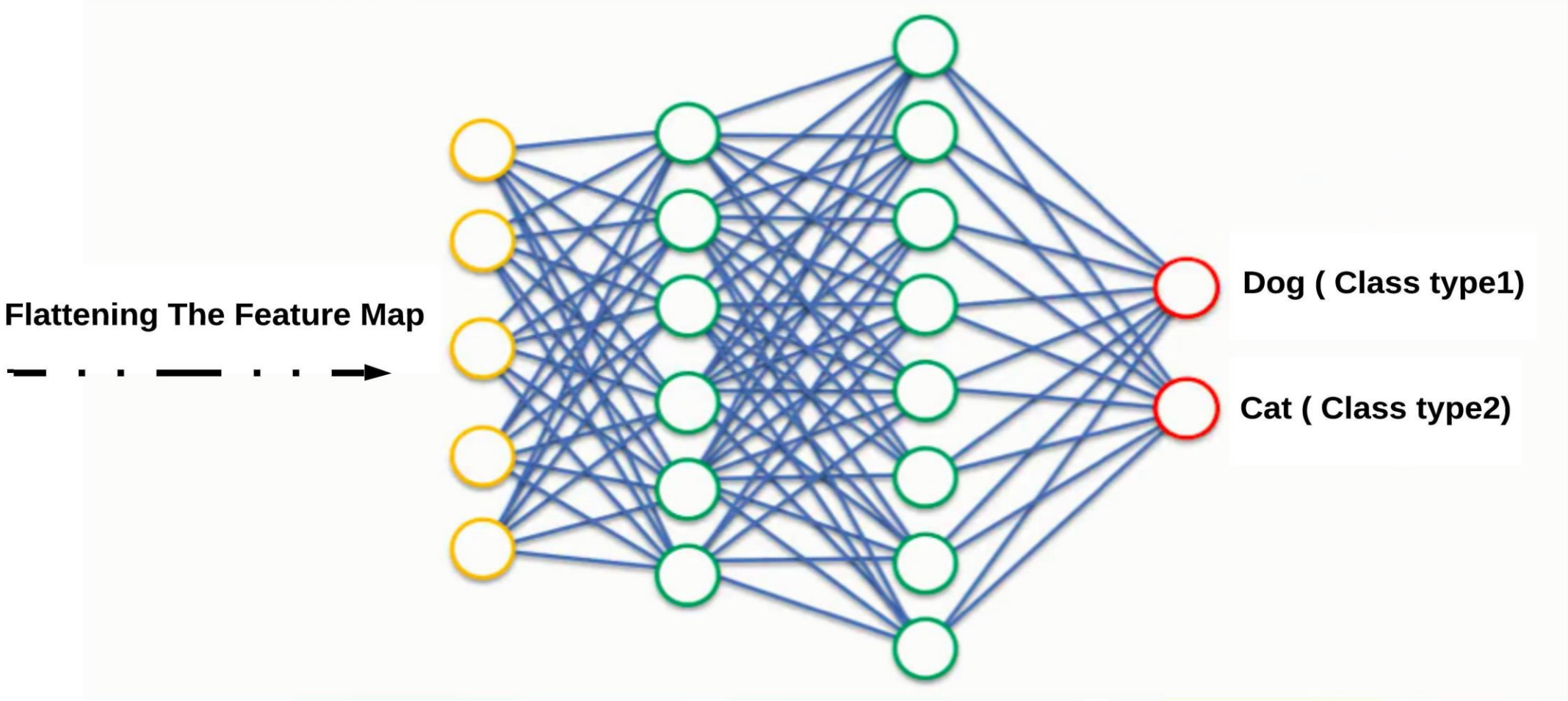
where a is a learnable weight.

Network Layers in CNN

- **Fully Connected (FC) Layer:**
  - Usually the last part (or layers) of every CNN architecture (used for classification) consists of fully-connected layers, where each neuron inside a layer is connected with each neuron from it's previous layer. The last layer of Fully-Connected layers is used as the output layer (classifier)of the CNN architecture.
  - The Fully-Connected Layers are type of feed-forward artificial neural network (ANN) and it follows the principle of traditional multi-layer perceptron neural network (MLP).
  - The FC layers take input from the final convolutional or pooling layer, which is in the form of a set of metrics (feature maps) and those metrics are flattened to create a vector and this vector is then fed into the FC layer to generate the final output of CNN

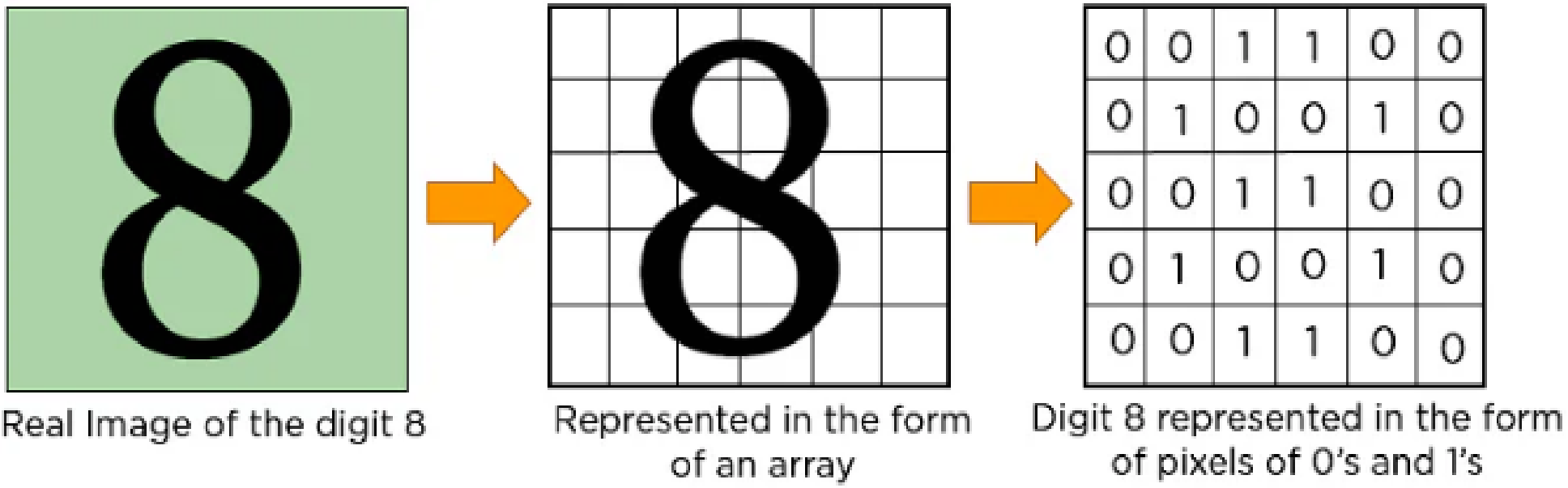
Network Layers in CNN

- The architecture of Fully Connected Layers:



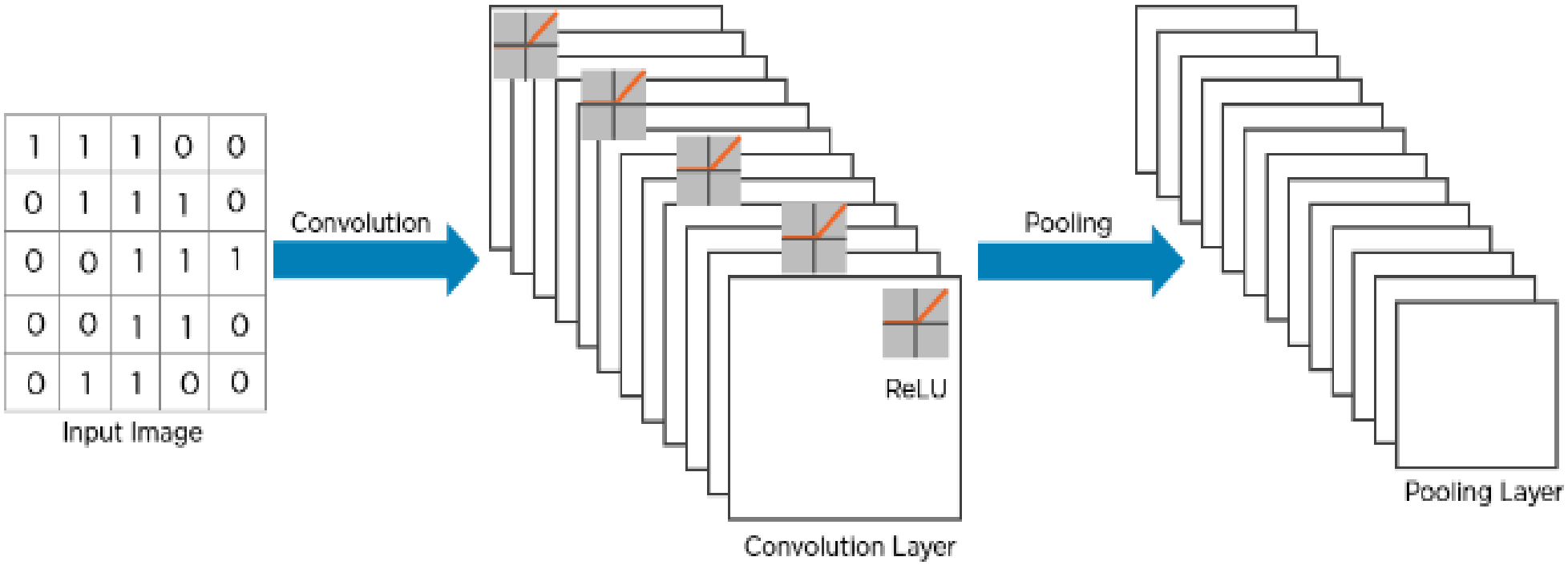
Further Discussion

- In CNN, every image is represented in the form of an array of pixel values.



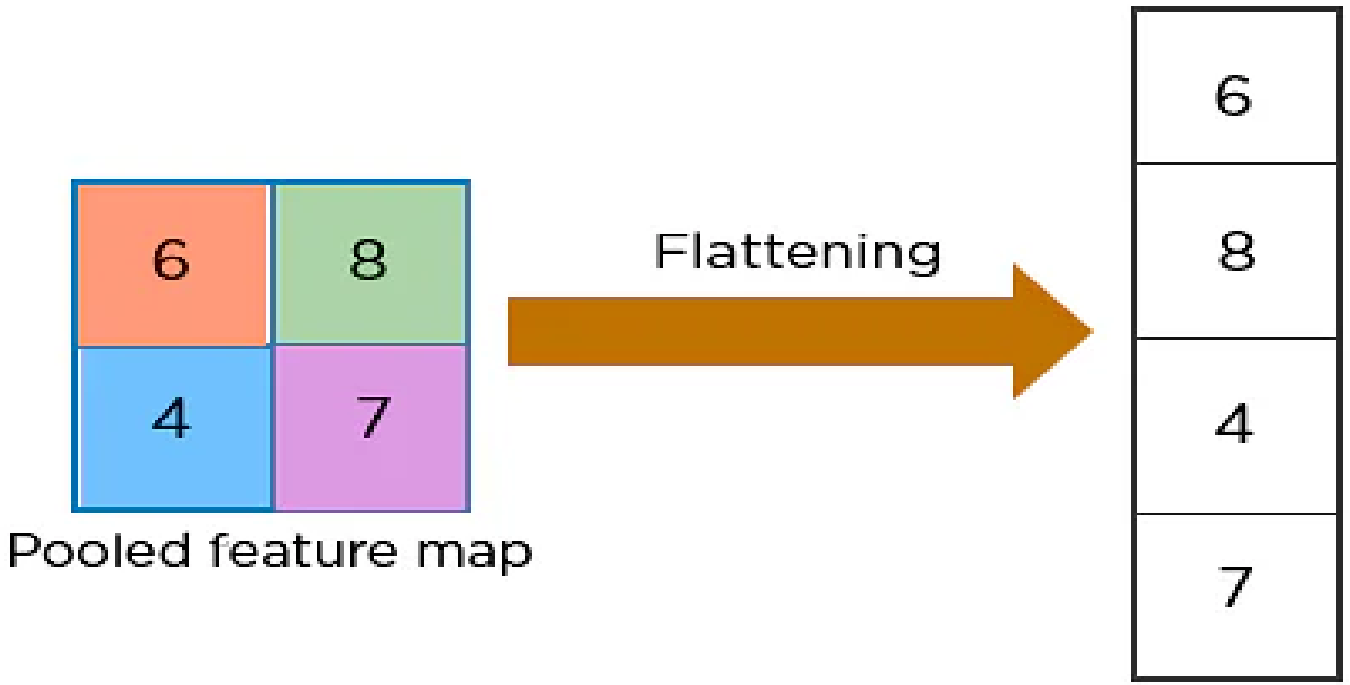
Further Discussion

- Here's how the structure of the convolution neural network looks so far:



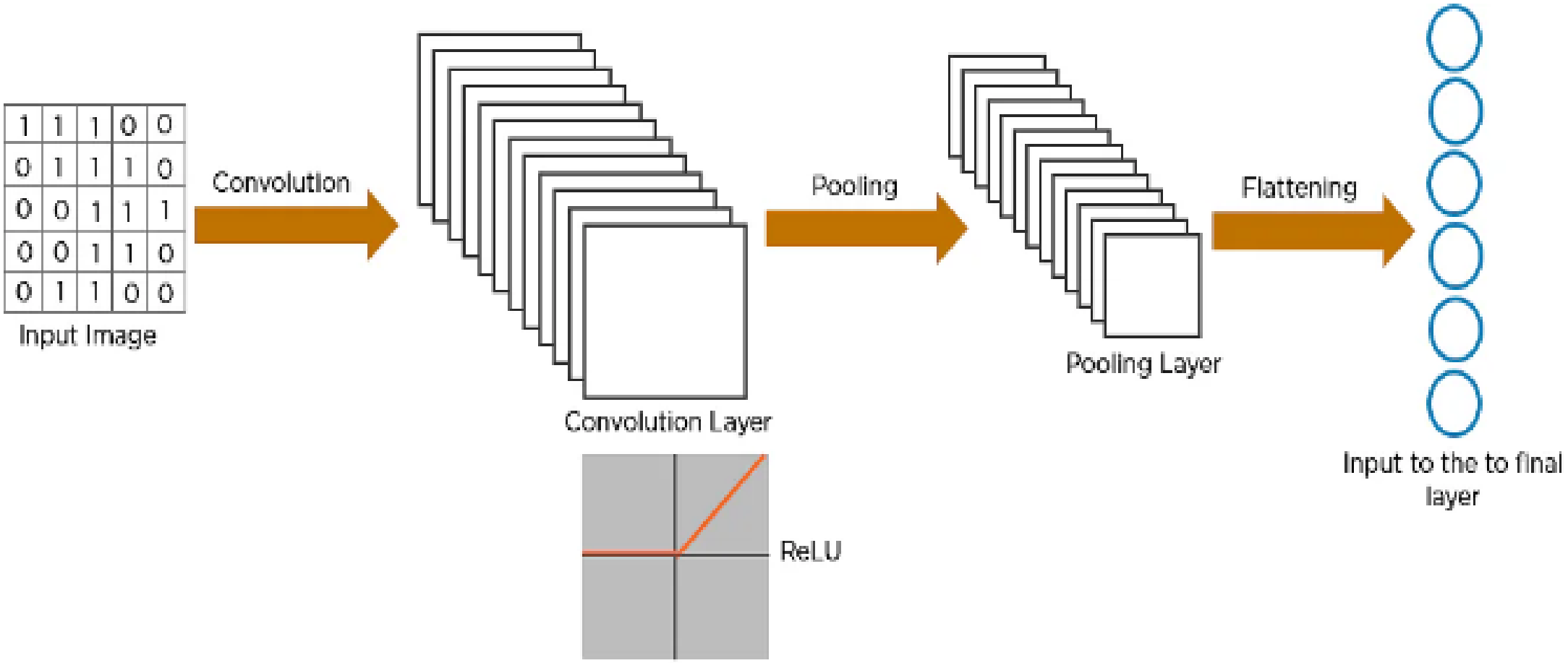
Further Discussion

- The next step in the process is called flattening. Flattening is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector.

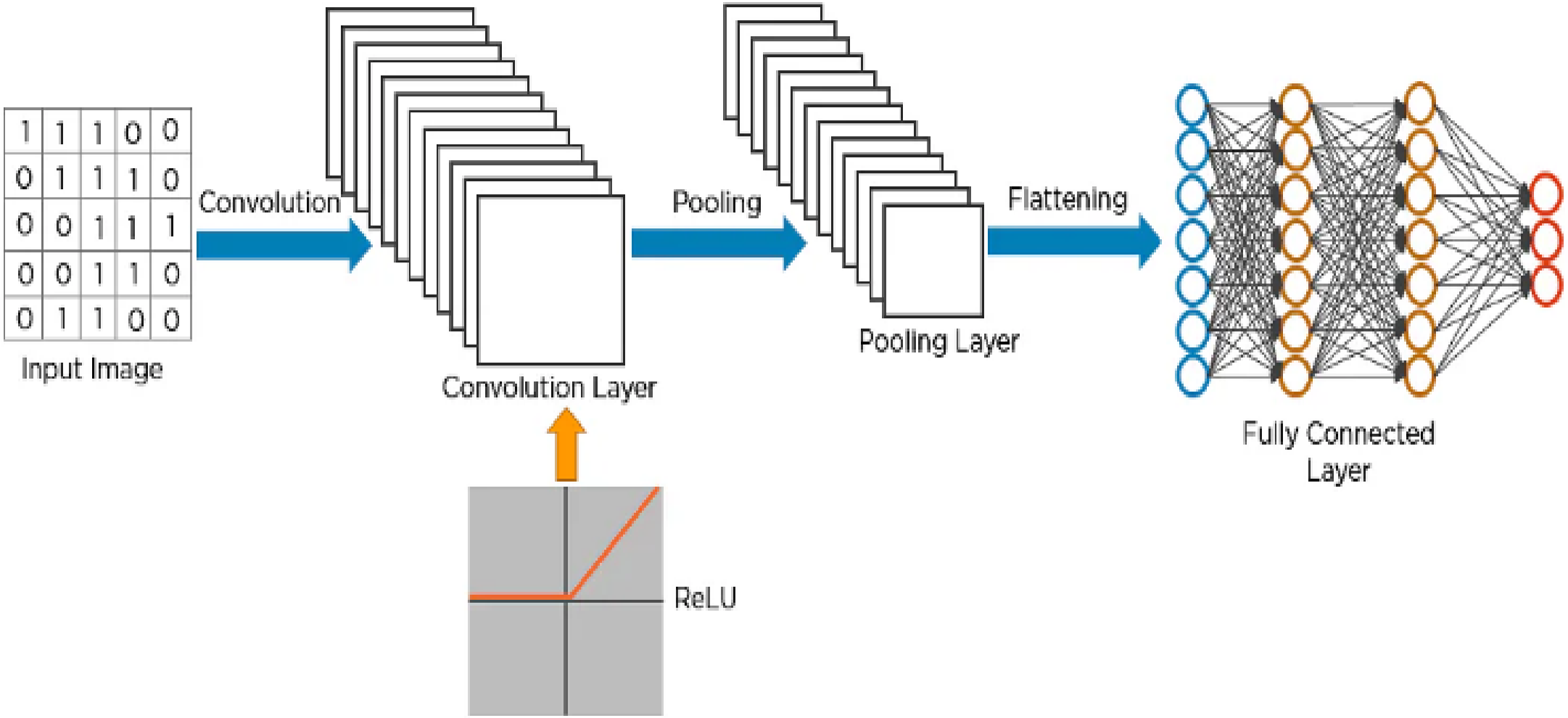


Further Discussion

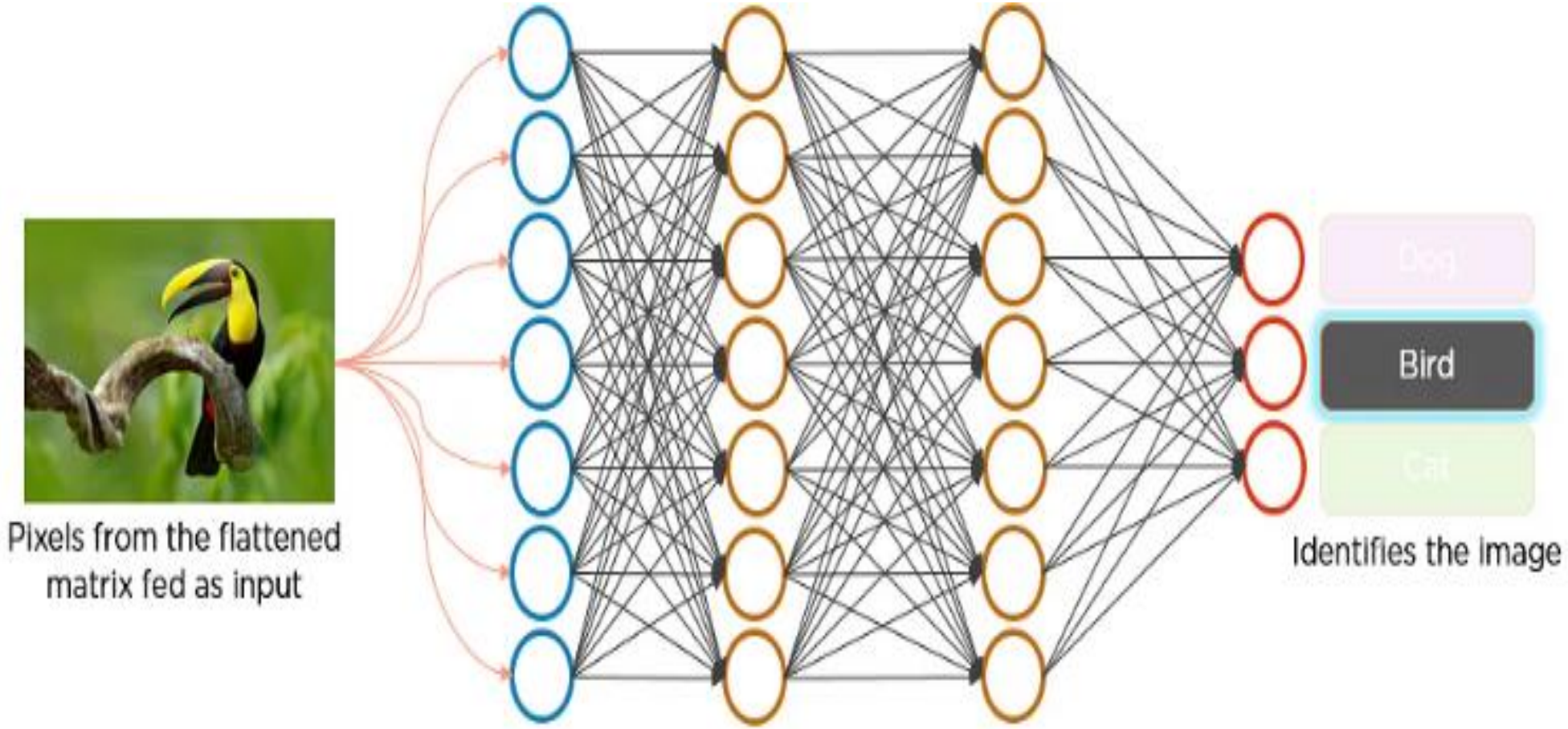
- The flattened matrix is fed as input to the fully connected layer to classify the image.



Further Discussion



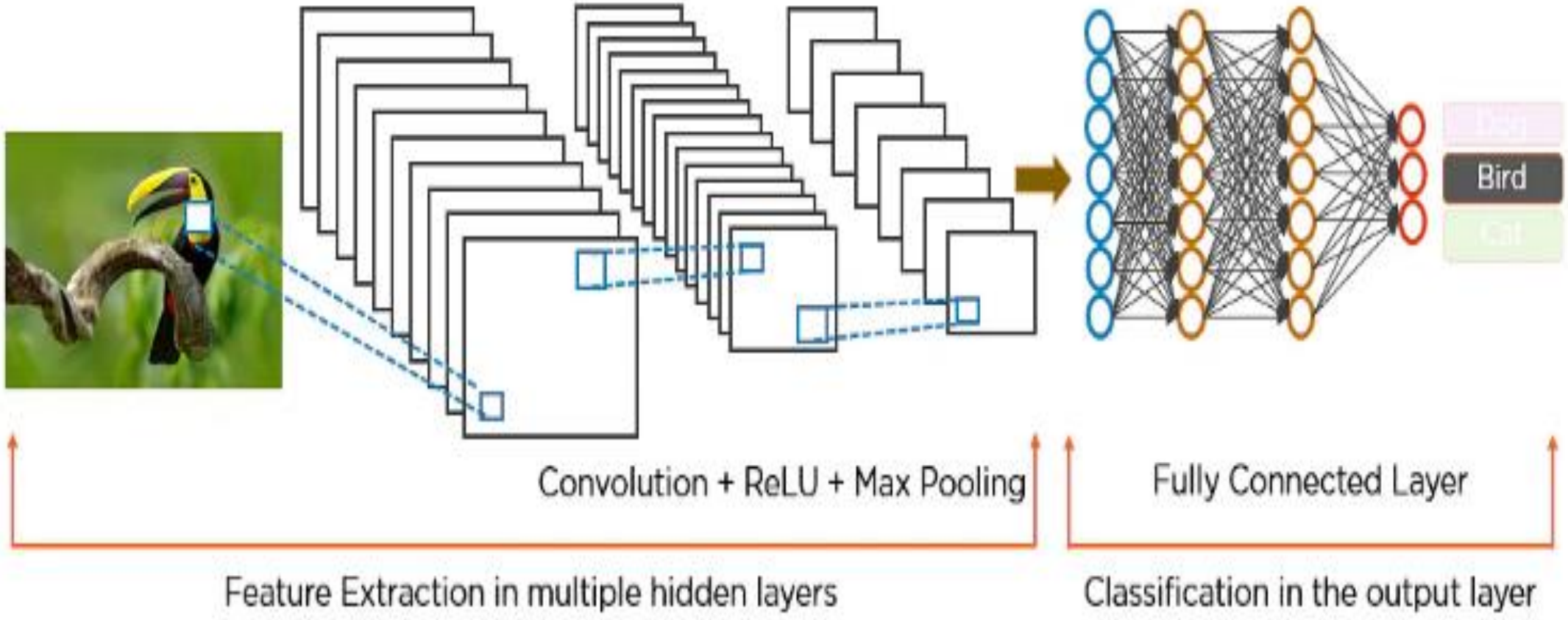
Further Discussion



Further Discussion

- Here's how exactly CNN recognizes a bird:
  1. The pixels from the image are fed to the convolutional layer that performs the convolution operation
  2. It results in a convolved map
  3. The convolved map is applied to a ReLU function to generate a rectified feature map
  4. The image is processed with multiple convolutions and ReLU layers for locating the features
  5. Different pooling layers with various filters are used to identify specific parts of the image
  6. The pooled feature map is flattened and fed to a fully connected layer to get the final output

Further Discussion



Loss Functions

- The last layer of every CNN architecture (classification-based) is the output layer, where the final classification takes place.
- In this output layer, we calculate the prediction error generated by the CNN model over the training samples using some Loss Function.
- This prediction error tells the network how off their prediction from the actual output, and then this error will be optimized during the learning process of the CNN model.
- The loss function uses two parameters to calculate the error, the first parameter is the estimate output of the CNN model (also called the prediction) and the second one is the actual output (also known as the label).

Cross-Entropy or Soft-Max Loss Function

- Cross-entropy loss, also called log loss function is widely used to measure the performance of CNN model, whose output is the probability $p \in \{0,1\}$.
- It uses softmax activations in the output layer to generate the output within a probability distribution, i.e., $p, y \in R^N$, where p is the probability for each output category and y denotes the desired output and the probability of each output class can be obtained by:

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}$$

- where N is the number of neurons in the output layer and e^{a_i} denotes each unnormalized output from the previous layer in the network. Now finally, cross-entropy loss can be defined as :

$$H(p, y) = -\sum_i y_i \log(p_i)$$

$$\text{where } i \in [1, N]$$

Euclidean Loss Function

- The Euclidean loss also called mean squared error (MAE) is widely used in regression problems.
- The MAE between the predicted output $p \in R^N$ and the actual output $y \in R^N$ in each neuron of the output layer of CNN is defined as $H(p, y) = (p - y)^2$.
- So, if there are N neurons in the output layer then, the estimate euclidean loss is defined as:

$$H(p, y) = \frac{1}{2N} \sum_{i=1}^{N} (p_i - y_i)^2$$

Hinge Loss Function

- The Hinge loss function is widely used in binary classification problems. It is used in “maximum-margin" based classification problem, most notably for support vector machines (SVMs).
- Here, the optimizer tries to maximize the margin between two target classes. The hinge loss is defined as:

$$H(p, y) = \sum_{i=1}^{N} \max(0, m - (2y_i - 1)p_i)$$

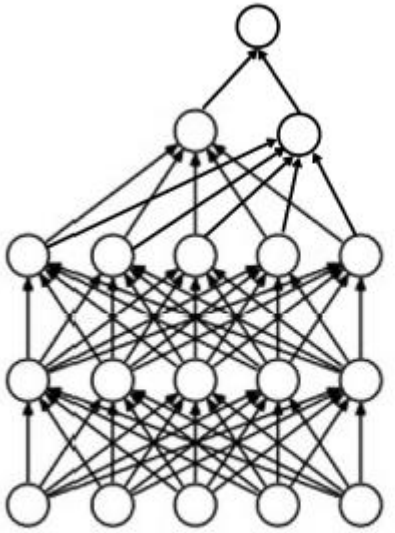
- where m is the margin which is normally set equal to 1, p_i denotes the predicted output and y_i denotes the desired output.

Regularization to CNN

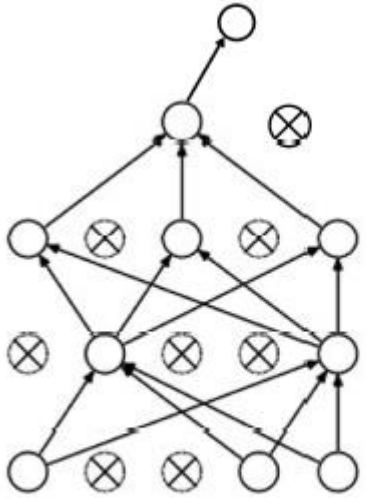
- The core challenge of deep learning algorithms is to adapt properly to new or previously unseen input, drawn from the same distribution as training data, the ability to do so is called generalization.
- The main problem for a CNN model to achieve good generalization is over-fitting.
- When a model performs exceptionally well on training data but it fails on test data (unseen data), then this type of model is called over-fitted.
- The opposite is an under-fitted model, that happens when the model has not learned enough from the training data and when the model performs well on both train and test data, then these types of models are called just-fitted model.
- Regularization helps to avoid over-fitting by using several intuitive ideas.

Regularization to CNN

- **Dropout:**
  - Dropout is one of the most used approach for regularization. Here we randomly drop neurons from the network during each training epoch.
  - By dropping the units (neurons) we try to distribute the feature selection power to all the neurons equally and we forced the model to learn several independent features.
  - Dropping a unit or neuron means, the dropped unit would not take part in both forward propagation or backward propagation during the training process. But in the case of testing process, the full-scale network is used to perform prediction.



An normal Neural network



After applying dropout

Regularization to CNN

- **Drop-Weights:**
  - It is very much similar to dropout. The only difference is instead of dropping the neurons, here we randomly drop the weights (or connections between neurons) in each training epoch.
- **The l^2 Regularization:**
  - The l^2 regularization or "weight decay" is one of the most common forms of regularization. It forces the network's weights to decay towards zero (but not equal to zero) by adding a penalty term equal to the "squared magnitude" of the coefficient to the loss function.
  - It regularizes the weights by heavily penalize the larger weight vectors.
  - This is done by adding $\frac{1}{2}\lambda\|W\|^2$ to the objective function, where λ is a hyper-parameter, which decides the strength of penalization and $\|W\|$ denotes the matrix norm of network weights.

Regularization to CNN

- **The l^2 Regularization:**
  - Consider a network with only a single hidden layer and with parameters W. If there are N neurons in the output layer and the prediction output and the actual output are denoted by y_n and p_n where $n \in [1, N]$. Then the objective function:

$$CostFunction = loss + \frac{1}{2}\lambda\|w\|^2$$

- **The l^1 Regularization:**
  - The l^1 regularization is almost similar to the l^2 regularization and also widely used in practice, but the only difference is, instead of using “squared magnitude" of coefficient as a penalty, here we use the absolute value of the magnitude of coefficients as a penalty to the loss function. So the objective function with l^1 regularization as:

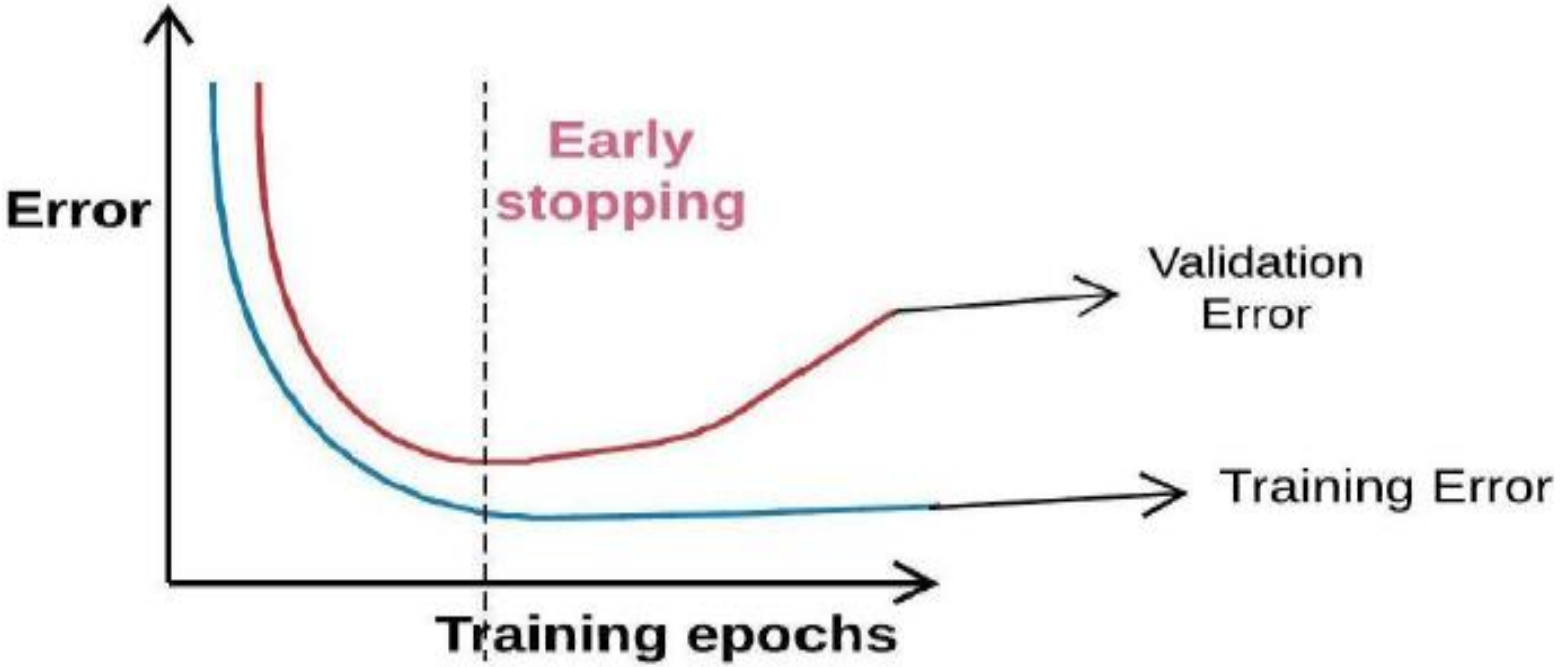
$$CostFunction = loss + \lambda\|w\|$$

Regularization to CNN

- **Data Augmentation:**
  - The easiest way to avoid overfitting is to train the model on a large amount of data with several varieties. This can be achieved by using data augmentation, where we use several techniques to artificially expand the training dataset size.
- **Early Stopping:**
  - In early stopping, we keep a small part (maybe 20% to 30%) of the train dataset as the validation set which is then used for Cross-Validation purposes, where we evaluate the performance of the trained model over this validation set in each training epochs.
  - Here we use a strategy that stops the training process when the performance on the validation set is getting worse with the next subsequent epoch.
  - As the validation error gets increased, the generalization ability of the learned model also gets decreased.

Regularization to CNN

- **Early Stopping:**



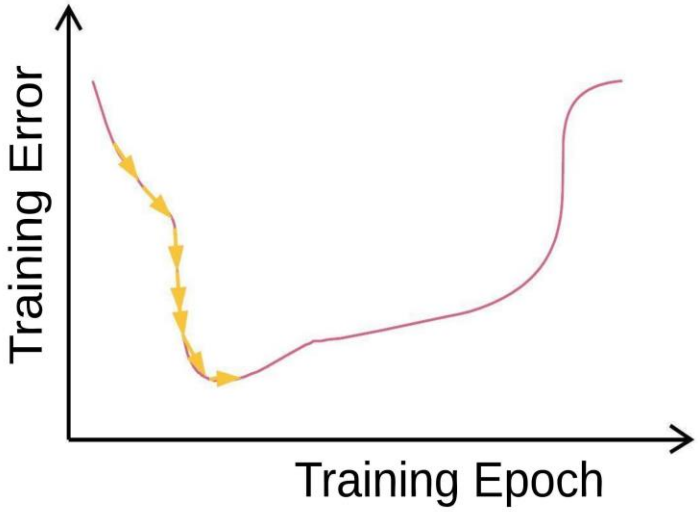
A tipical illustration of early stopping approach during network training.

Optimizer selection

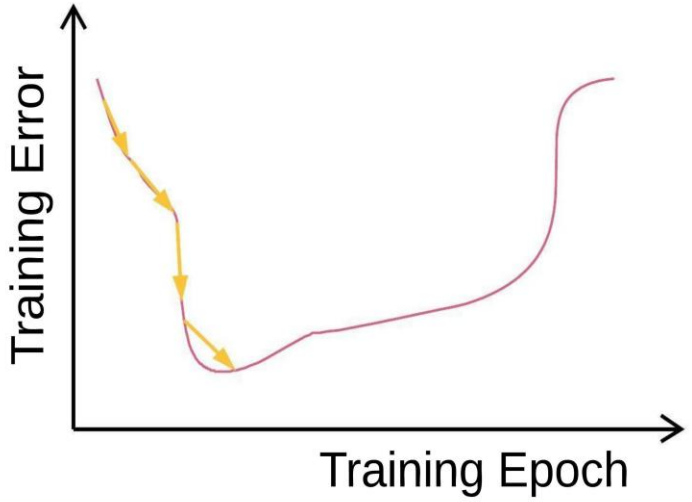
- The learning process includes two major things, first one is the selection of the learning algorithm (Optimizer) and the next one is to use several improvements (such as momentum, Adagrad, AdaDelta) to that learning algorithm in order to improve the result.
- In case of learning to a CNN model, the gradient-based learning methods come as a natural choice.
- To reduce the error, the model parameters are being continuously updated during each training epoch and the model iteratively search for the locally optimal solution in each training epoch.

Optimizer selection

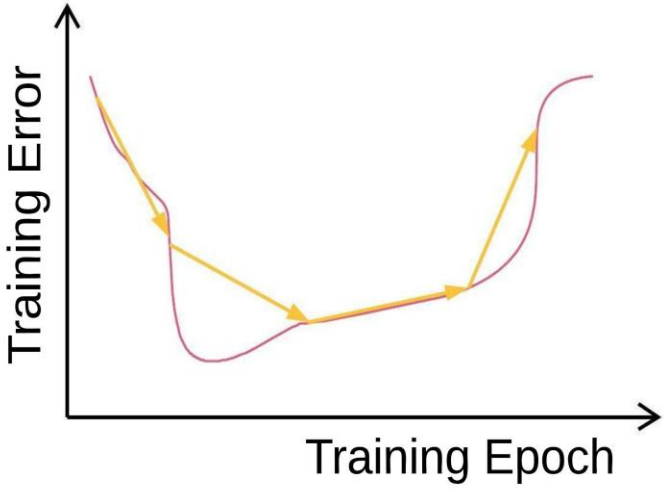
- The size of parameter updating steps is called the “learning rate" and a complete iteration of parameter update which includes the whole training dataset once is called a “training epoch".
- Although the learning rate is a hyper-parameter, but we need to choose it so carefully that, it does not affect the learning process badly.



Very low LR



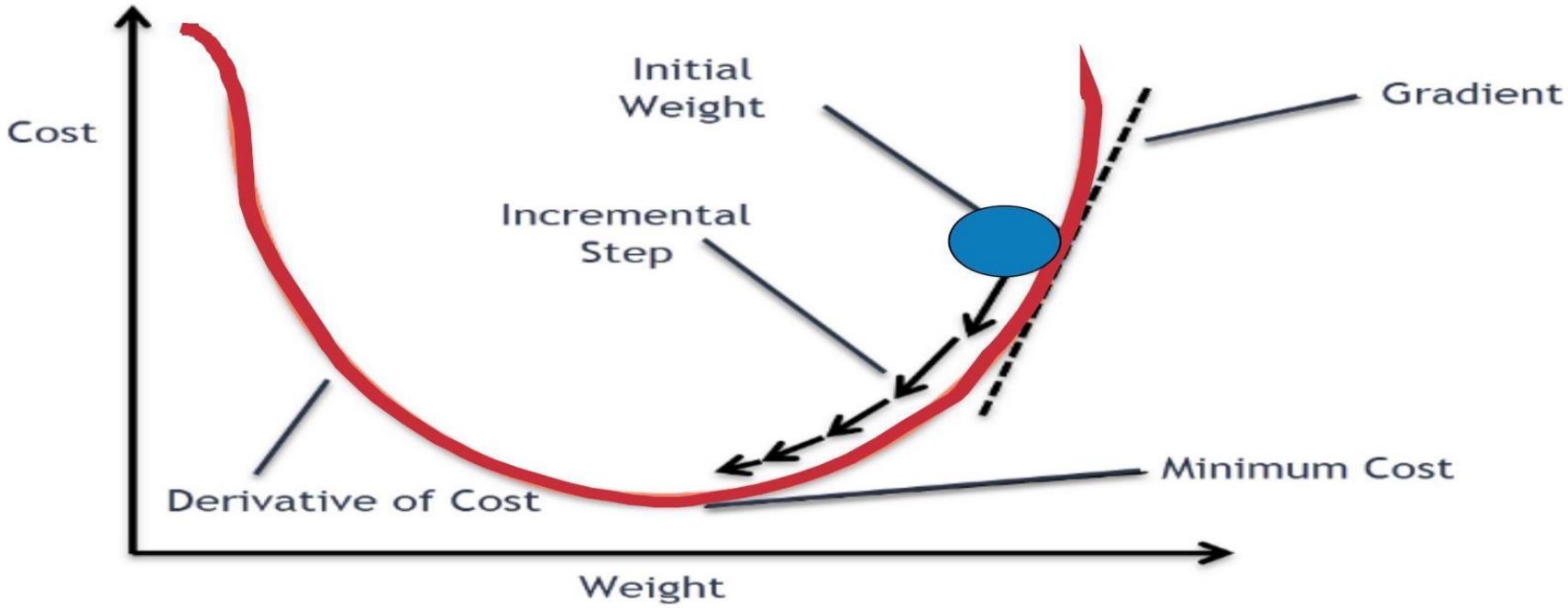
Good LR



Very large LR

Gradient Descent or Gradient-Based Learning Algorithm

- The gradient descent algorithm updates the model parameters continuously during each training epoch in order to reduce the training error.
- For updating those parameters in correct way, it first calculates the gradient of the objective function by using first-order derivative with respect to the model's parameters, and then to minimize the error, it updates the parameter in the opposite direction of the gradient.



Gradient Descent or Gradient-Based Learning Algorithm

- This parameter updating process is done during back-propagation of the model, where the gradient at each neuron is back propagate to all the neurons belonging form it's previous layer. This operation can be mathematically represented as:

$$w_{ij}^{t} = w_{ij}^{t-1} - \Delta w_{ij}^{t}$$

$$\Delta w_{ij}^{t} = \eta * \frac{\partial E}{\partial w_{ij}}$$

where w_{ij}^{t} denotes the final weight in current t'th training epoch, w_{ij}^{t-1} denotes the weight in previous $(t-1)$'th training epoch, η is the learning rate, E is the prediction Error.

- There exist a number of variants of the gradient-based learning algorithm.

Gradient Descent or Gradient-Based Learning Algorithm

- **Batch Gradient Descent:**
  - In Batch Gradient descent, the parameters of the network are updated only once after passing the whole training dataset through the network.
  - That is, it computes the gradient on the entire training set and then updates the parameters using this gradient.
  - With Batch gradient descent, the CNN model produces more stable gradient and also converges faster for small-sized datasets.
  - It also needs fewer resources, because the parameters are updated only once for each training epoch. But if the training dataset becomes large, then it takes more time to converge and it may converge in local optimal solution.

Gradient Descent or Gradient-Based Learning Algorithm

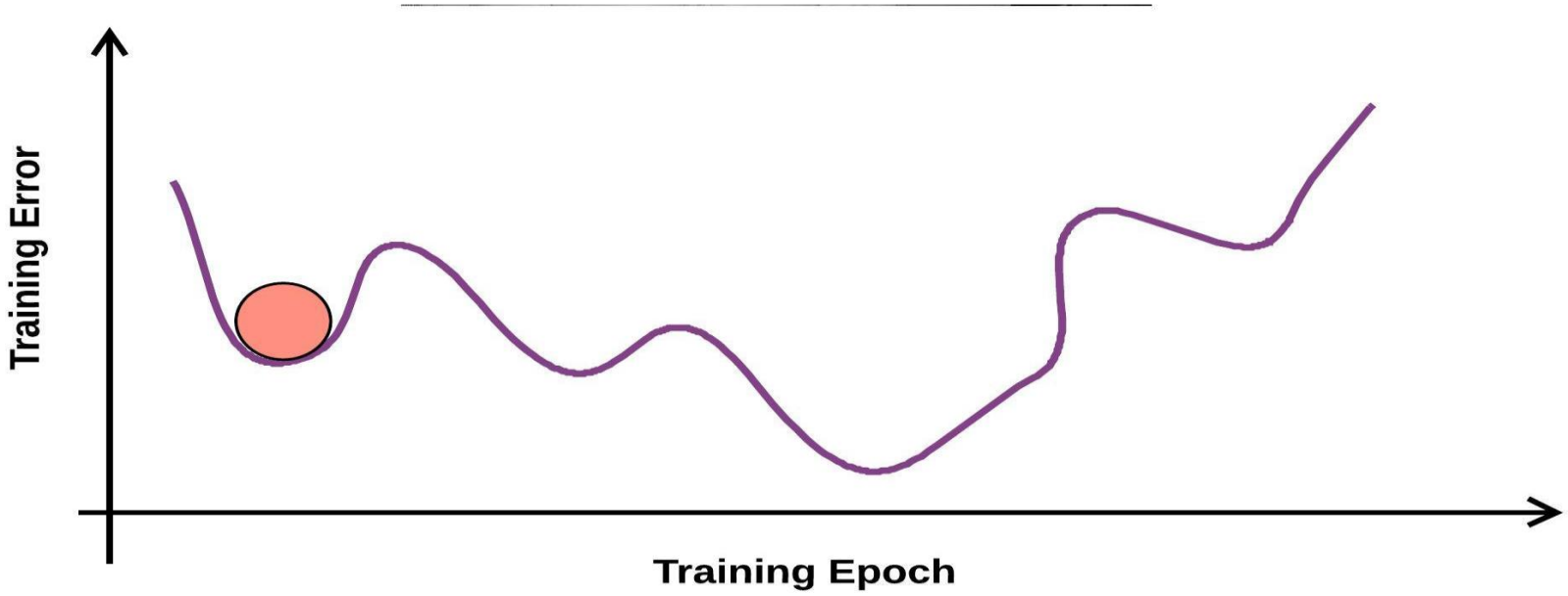
- **Stochastic Gradient Descent (SGD):**
  - Unlike Batch Gradient descent, here the parameters are updated for each training sample separately.
  - Here it is recommended to randomly shuffle the training samples in each epoch before training.
  - The benefit of using it over Batch Gradient descent is that it converges much faster in case of large training dataset and it is also memory efficient.
  - But the problem is, due to frequent updates it takes very noisy steps towards the solution that make the convergence behavior very unstable.

Gradient Descent or Gradient-Based Learning Algorithm

- **Mini Batch Gradient Descent:**
  - Here we divide the training examples into a number of mini-batches in non-overlapping manner, where each mini-batch can be imagined as the small set of samples and then update the parameters by computing the gradients on each mini-batch.
  - It carries the benefit of both Stochastic Gradient Descent and Batch Gradient Descent by mixing them.
  - It was more memory efficient, more computationally efficient and also has a stable convergence.

Gradient Descent or Gradient-Based Learning Algorithm

- **Momentum:**
  - Momentum is a technique used in the objective function of neural networks, which improves both training speed and accuracy by adding the gradient calculated at the previous training step weighted by a parameter λ called the momentum factor.
  - The major problem of Gradient-Based learning algorithm is that it easily stuck in a local minima instead of global minimum, this mostly happens when the problem has non-convex solution space (or surface).



Gradient Descent or Gradient-Based Learning Algorithm

- **Momentum:**
  - To solve this issue, we use the momentum along with the learning algorithm. It can easily mathematically expressed as:

$$\Delta w_{ij}^{t} = (\eta * \frac{\partial E}{\partial w_{ij}}) + (\lambda * \Delta w_{ij}^{t-1})$$

where Δw_{ij}^{t} is weight increment in current t'th training epoch, η is the learning rate and λ is the momentum factor and (Δw_{ij}^{t-1}) is weight incrementation in previous $(t-1)$'th training epoch.

  - The value of momentum factor is staying in between 0 and 1 that increases the step size of weight update towards minima, for minimizing the error.
  - The large value of momentum factor helps the model to converge faster and the very lower value of momentum factor can not avoid local minima.
  - But if we use both LR and momentum factor value as high, it may also miss global minima by jumping over it.

Gradient Descent or Gradient-Based Learning Algorithm

- **AdaGrad:**
  - AdaGrad or adaptive learning rate method performs larger updates for infrequent parameters (by using larger learning rate value) and smaller updates for frequent parameters (by using smaller learning rate value).
  - It is done by dividing the learning rate of each parameter with the sum of square of all the past gradients for each parameter W_{ij} in each training epoch t.
  - In practice AdaGrad is very useful, especially in case of sparse gradients or when we have sparse training data for a large scale neural networks. The update operation can easily mathematically expressed as:

$$w_{ij}^{t} = w_{ij}^{t-1} - \frac{\eta}{\sqrt{\sum_{k=1}^{t} {\delta_{ij}^{k}}^{2} + \in}} * \delta_{ij}^{t}$$

where w_{ij}^{t} is the weight in current t'th training epoch for parameter w_{ij} , w_{ij}^{t-1} is the weight in previous $(t-1)$ 'th training epoch for parameter w_{ij} , δ_{ij}^{t} is the local gradient of parameter w_{ij} in t 'th epoch , δ_{ij}^{t-1} is local gradient of parameter w_{ij} in $(t-1)$'th epoch, η is the learning rate and $\in$ is a term contains very small value to avoid dividing by zero.

Gradient Descent or Gradient-Based Learning Algorithm

- **AdaDelta:**
  - AdaDelta can be imagined as the extension of AdaGrad. The problem with AdaGrad is that, if we train the network with many large training epochs (t), then the sum of square of all the past gradients becomes large, as a result, it almost vanishes the learning rate.
  - To solve this issue, the adaptive delta (AdaDelta) method divides the learning rate of each parameter with the sum of square of past k gradients (instead of using all the past gradients, which is done in the case of AdaGrad) for each parameter W_{ij} in each training epoch t. The update operation can easily mathematically expressed as:

$$w_{ij}^{t} = w_{ij}^{t-1} - \frac{\eta}{\sqrt{\sum_{m=(t-k+1)}^{t} \delta_{ij}^{m2} + \in}} * \delta_{ij}^{t}$$

Gradient Descent or Gradient-Based Learning Algorithm

- **RMSProp:**
  - Root Mean Square Propagation (RMSProp) is also designed to solve the Adagrad's radically diminishing learning rates problem as discussed before.
  - It tries to resolve Adagrad's issue by using a moving average over past squared gradient $E[\delta^2]$. The update operation can easily mathematically expressed as:

$$w_{ij}^{t} = w_{ij}^{t-1} - \frac{\eta}{\sqrt{\mathbf{E}[\delta^2]^t}} * \delta_{ij}^{t}$$

$$And,$$

$$\mathbf{E}[\delta^2]^t = \gamma \mathbf{E}[\delta^2]^{t-1} + (1-\gamma)(\delta^t)^2$$

  - where γ can be set to 0.9, with a good default initial learning rate value like 0.001.

Gradient Descent or Gradient-Based Learning Algorithm

- **Adaptive Moment Estimation (Adam):**
  - Adam is another learning strategy, which calculates adaptive LR for each parameter in the network and it combines the advantages of both Momentum and RMSprop by maintaining the both exponential moving average of the gradients (as like Momentum) and as well as the exponential moving average of the squared gradients (as like RMSprop). So the formulas for those estimators are as:

$$\mathbf{E}[\delta]^t = \gamma_1 \mathbf{E}[\delta^2]^{t-1} + (1-\gamma_1)[\delta^t]$$

$$\mathbf{E}[\delta^2]^t = \gamma_2 \mathbf{E}[\delta^2]^{t-1} + (1-\gamma_2)(\delta^t)^2$$

  - $E[\delta]^t$ is the estimate of the first moment (the mean) and $E[\delta^2]^t$ is the estimate of the second moment (the uncentered variance) of the gradients.

Gradient Descent or Gradient-Based Learning Algorithm

- **Adaptive Moment Estimation (Adam):**
  - Since at initial training epoch the both estimates are set to zero, they can remain biased towards zero even after many iterations, especially when γ_1, γ_2 are very small. To counter this issue, the estimates are calculated after bias-correction and the final formulas for those estimators become as:

$$\widehat{\mathbf{E}[\delta]^t} = \frac{\mathbf{E}[\delta]^t}{(1-(\gamma_1)^t)}$$

$$\widehat{\mathbf{E}[\delta^2]^t} = \frac{\mathbf{E}[\delta^2]^t}{(1-(\gamma_2)^t)}$$

  - So the parameter update operation in Adam finally can easily mathematically expressed as:

$$w_{ij}^t = w_{ij}^{t-1} - \frac{\eta}{\sqrt{\widehat{\mathbf{E}[\delta^2]^t} + \in}} * \widehat{\mathbf{E}[\delta]^t}$$

  - Adam is more memory efficient than others and also needs less computational power.