**REFERENCES:**

[1] G. Cybenko, “Approximation by superposition of a sigmoidal function,” Math. ControlSignals Systems, 1989.

[2] F. Chollet. *Deep Learning with Python*. Manning, Shelter Island, NY, USA, 1st edition, 2017

[3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning (Adaptive Computation and Machine Learning Series*. MIT Press, Cambridge, MA, USA, 2016.

[4] <https://www.ibm.com/cloud/learn/gradient-descent>

[5] Sebastian Ruder. *An overview of gradient descent optimization algorithms*

[6] <https://builtin.com/data-science/gradient-descent>

*[7] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell. Understanding convolution for semantic segmentation. In IEEE Winter Conference on Applications of Computer Vision (WACV), pages 1451 - 1460, March 2018.*

*[8] Kunihiko Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,”* Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.*

*[9]* [*https://cs231n.github.io/convolutional-networks/*](https://cs231n.github.io/convolutional-networks/)

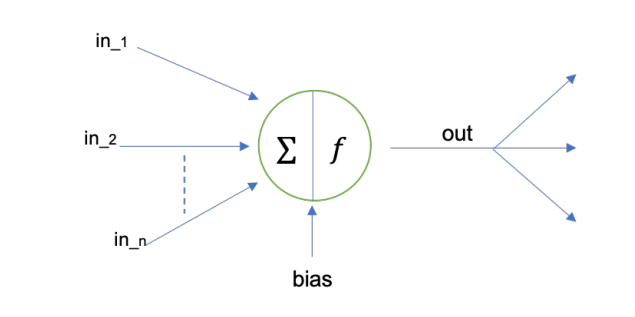
**Background Knowledge**

1. **Artificial Neural Networks (ANN)**

The concept of Artificial Neural Networks was built on the inspire of human brain biology characteristics, Cybenko [1] proved that in a neural network a single hidden layer containing a finite number of neurons is capable of approximating any continuous function to any desired precision. it included a group of machine learning models that are able to “learn” to execute or perform some specific tasks based on supplied examples. In general, these networks were created by a group of units or nodes connected each other like a simulating version of neurons in human brain, which called artificial neurons.

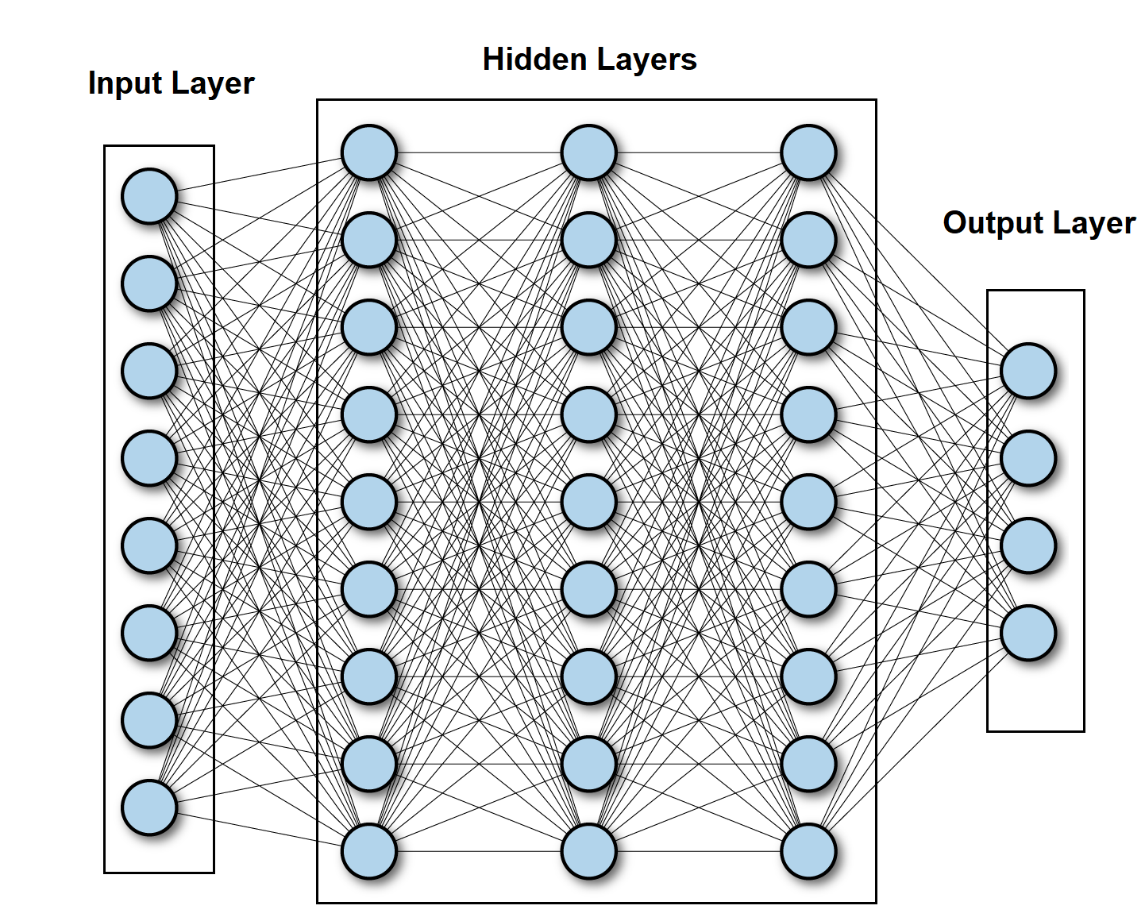
* 1. **Neuron**

Analog electrical signals flow inside biological neurons, while digital values present as the signals for artificial neurons. Each artificial neuron has an output being computed by using a function of the sum of the input signal values with generally added bias, the connection between the neurons as known as edges, in each connection contains an adjustable model parameter named weight that can be “learned” through a process called “model training”. The strength of the synaptic will determines the interaction’s level between the neurons, this function combines the neurons, bias and weights in the connections called “activation function”, and there are several activation functions that mean there is not only one fixed activation function for ANN, and we can choose the activation function for the network’s output. These functions will be mentioned later in this thesis.



* 1. **Layer**

In typical, all neurons are linked directly in network, on the other hand, they are grouped into subsets of neurons known as layers, which are generally connected in a sequential way based on the activation function in every node to compute nodes’ outputs from node’s inputs. In detail, the neurons in a layer are connected to a set of neurons in previous layer and to another one in next layer. Because of the modeling of ANNs as a connection of a neurons’ structure in layer, neurons’ outputs at a certain layer become neurons’ inputs at the following layer, which networks are understood as feedforward neural networks. There are no loops in present layer, therefore the inputs are always sent forward. As a result, the neurons into a layer cannot be connected with each other. When all the neurons in a layer are connected to all other ones in previous layer and following layer, it is called “Fully Connected Layers”, this state of the connection is kept in whole network as a series of fully connected layers that will be called “Fully Connected Neural Network”.



There is an existing of 3 kinds of layer inside a network, they are input layer, hidden layers and output layer.

Input layer represent for the input of the network, the number of nodes as known as neurons are size of the input that call “input dimension”. The hidden layers lay between input lay and output layer, there are so many hidden layers in a network, however there is only one input layer and output layer, and the output layer represent for the output of the network. In addition, the bias contains in input and output layer, not in output layer. Although simple decisions are made at the first layer based on the input, sophisticated judgments are already made in the second layer depending on decisions made in the first layer. As one moves deeper into the network, more difficult and abstract judgments emerge. The phrase deep neural networks refer to a network with numerous hidden layers.

* 1. **Forward and Backward (back propagation) steps**

As the mentioning before, the signals travel from input layer to output layer traversing hidden layers, which operations performed over the values received from the nodes in previous layers (in all layers but in the input layer) and whole activation function results are sent forwards to the next layer till the output layer. However, it is not the end of process. After getting the result of output layer, there is a backwards named back-propagation applying in here as known as training step for ANN. The algorithm is used to effectively train a neural network through a technique called chain rule. In simple terms, after each forward pass through a network, backpropagation performs a backward pass while adjusting the model’s parameters (weights and biases), which helps minimize the error for network’s result, it is called “loss function” or “cost function” which is the evaluation between predicted output and expected output at the final point of forward step. Basically, this function calculates the gradient of the loss function to each parameter in network respectively that means it work from the top layer (output layer) to the bottom layer (input layer) to calculate the contribution of each parameter in that value [2], which update the weights until reaching the minimum values. Furthermore, the key point of calculating the minimum value for loss function is gradient descent, which is a process of determining the modifications needed to minimize the loss and optimize the model to a specific evaluation measure by calculating the derivative of the loss function [3].

* 1. **Gradient Descent:**

Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks [4]. It is built on a convex function and iteratively changes its parameters to minimize a given function to its local minimum. On the other words, Gradient descent begins by defining the initialized parameter values, and then iteratively adjusts the values to minimize the given cost-function. Deeply, a gradient is a function’s derivative with several input variables. In terms of mathematics, a gradient is known as the function’s slope, and it simply measures the change in all weights in relation to the change in error. The steeper the slope and the faster a model can learn, the higher the gradient. However, if the slope is zero, the model will stop learning. A gradient is a partial derivative with regard to its inputs.

Imaging that a blindfolded man wants to hike down to the bottom of a valley from the top of a hill with as few steps as possible. He might begin going down the hill by taking large steps in the sharpest direction, which he can do as long as he is not close to the bottom. However, as he gets closer to the bottom, his steps will become smaller and smaller in order to prevent overshooting it. The gradient can be used to mathematically characterize this process. The learning rate η determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley [5].

To reach the local minimum for gradient descent, we should choose a relevant value for the learning rate, which is not both too low and too high. This is very significant because if the steps are too large, it could fail to reach the local minimum since it bounces between the convex function of gradient descent. Besides, gradient descent will eventually reach the local minimum if we set the learning rate to a very low value, but it may take a lot of time [6]. So, the key point is choosing an optimal learning rate that helps model converge to the minimum value without spending much time.



Gradient descent has three forms that differ in the amount of data used to compute the gradient of the objective function. We choose between the accuracy of the parameter update and the time it takes to complete an update based on the amount of data.

* + 1. **Batch Gradient Descent**

Batch gradient descent calculates the sum of the error for each point in a training set and updates the model only after all training instances have been evaluated. This procedure is known as a training epoch. Though this batching supplies a computation efficiency, it can still take a long time to execute big training datasets because it must still keep all of the data in memory [5]. Batch gradient descent also frequently yields a stable error gradient and convergence, although that convergence point isn't always the best, locating the local minimum rather than the global minimum.

* + 1. **Stochastic Gradient Descent**

Stochastic gradient descent (SGD) performs a training epoch for each example in the dataset and updates the parameters of each training example at a time. They are easier to remember because you only need to hold one training example [5]. In spite of the fact that these frequent updates provide more detail and speed, they can lead in computational efficiency losses when it be compared to batch gradient descent. Despite of ability of noisy gradients production, these frequent updates can also aid in escape the local minimum and locating the global one.

* + 1. **Mini-batch Gradient Descent**

Mini-batch gradient descent combines batch gradient descent with stochastic gradient descent ideas. It divides the training dataset into small batches and updates each of those batches. This method strikes a balance between batch gradient descent's computing efficiency and stochastic gradient descent's speed [5].

Besides, gradient descent has its own set of challenges such as: Local minima and saddle points, vanishing and exploding gradients.

* Local minima resemble global minima in shape with the slope of the cost function increasing on each side of the current location.
* Saddle points happen when the negative gradient exists just on one side of the point, reaching a local maximum on one side and a local minimum on the other. Its name was inspired by horse's saddle;
* Vanishing gradients happens when the gradient is too small. The gradient continues to shrink as we move backwards during backpropagation, leading earlier levels of the network to learn more slowly than later layers. When this occurs, the weight parameters are updated until they become inconsequential, i.e., 0, resulting in an algorithm that is no longer learning.
* Exploding gradients occurs when the gradient becomes too big, resulting in an unstable model. In this instance, the model weights will get excessively huge and will be represented as “NaN” finally. One approach to this problem is to use a dimensionality reduction technique, which can help to reduce model complexity.

1. **Convolutional Neural Network**

Convolutional Neural Networks (CNNs), like ANNs, being inspired by the structure of the human visual cortex [8]. CNNs, which are a type of neural network typically utilized for computer vision tasks in deep learning, include convolutional layers, pooling layers and fully connected layers. These networks are identical to traditional neural networks, except that instead of generic matrix multiplication, a convolution kernel is used in one or more of their layers [3]. These networks have the role of reduction the images into a form which is easier to process, without losing features which are critical for getting a good prediction. Moreover, CNNs have ability to take image tensors as an input, identify which image characteristics or features are essential for classification or differentiation, and output these classifications [1, 7].

Therefore, the basic structure of a Convolutional Network in order starting from a Convolutional Layer or so-called convolution process (do not count the initial input layer), has the effect of picking out the 2-dimensional characteristics of the image input. Specifically, the features of the input original image are passed through the “convolutional layer”, at which there is a mechanism built up by matrix multiplication between the input image known as a matrix and one or more matrices with a certain size (usually smaller than the input matrix) being called “filters”, this matrix multiplication will be shifted by rows and columns with a certain rule that will give generate one or more new matrices based on the number of filter layers, the product from this process is called feature maps which contain the attributes of the original input image.

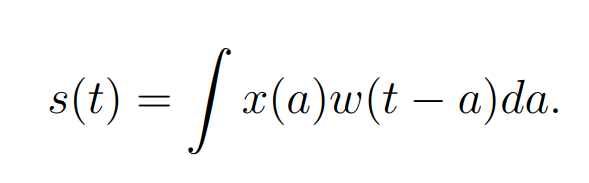
Followed by a pooling layer to help the network synthesize the features of the image in a deeply specific way (max pooling) or more objectively general (average pooling or mean pooling) from the output of the previous layer. Next, the size of the image has been significantly reduced that is very meaningful for memory in process of training model. Although the dimensions are decreased, the number of channels or feature maps are still kept unchanged. So, this process is as a compositing to reduce the size of the input image feature in length and width. At last, when the input size has been reduced to a reasonable value, the 2-dimensional image will be converted to a one-dimensional (one-dimensional vector) or known as "flatten", at which this process is implemented as a Fully Connected Layer, the number of units in this layer will correspond to the number of classes that defined for the network to distinguish as known as the final output of the network with a specific activation function for specific final result.

In general, using CNNs because of their capacity to learn translation-invariant patterns as well as spatial hierarchies utilizing appropriate filters. Learning translation invariant patterns means that once a pattern is learned in one area of a given image, the network will be able to recognize it anywhere else in the image. This improves image processing efficiency by requiring fewer training samples to learn generalizable representations [1].

Spatial hierarchies show how earlier layers in the network can learn tiny local patterns and grow into bigger patterns composed of these tiny patterns in the next layers, which also improves the network's ability to learn complicated and abstract concepts of visualization. In addition, this also allows these networks to deliver class predictions with far higher accuracy than vectorizing a complex image with pixel dependencies throughout. Therefore, CNNs have to be used for any image classification task so as to track these crucial properties and correlations between features [1].

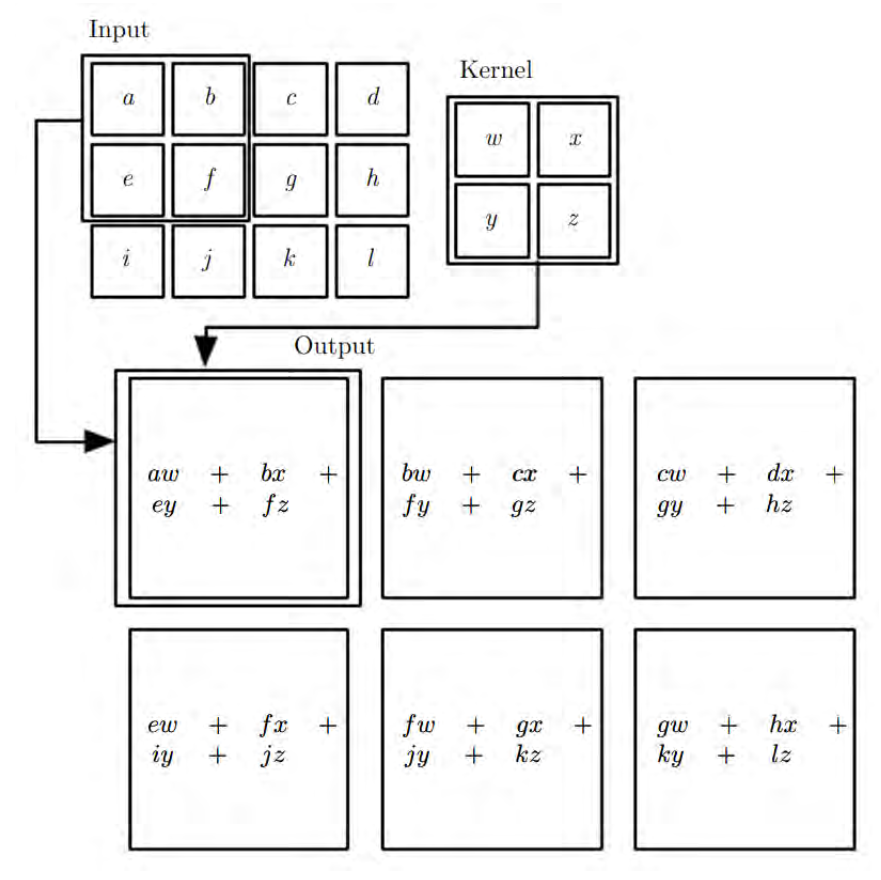
* 1. **Convolution Layer**

Convolution is an operation that produces a new function by combining two functions of a real-valued argument. Let s(t) is the output estimate function based on time t, x(a) is the age-based input position function, and w(a) is the weighting function that prioritizes recent measurements. This is the general formula for convolution utilizing this defined function:

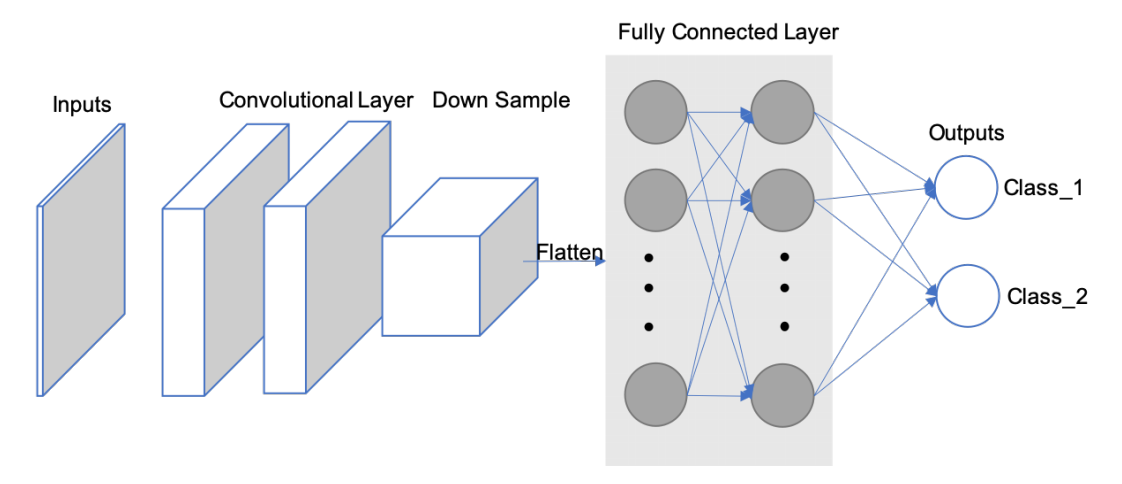


In detail, the input is the function x(a), the kernel is the weighting function w(a), and the output s(t) is the feature map in the CNN. the CNNs use convolutions that are performed over two-dimensional tensors that are made up of the input images' height, width, and channels of colour. These techniques take out patches and alter the input to build a feature map with varied depth. One of the two critical criteria for defining these convolutions is the size of these extracted patches. The depth of the output feature map is the second parameter to consider. This is the number of filters built by the layer that can encode diverse features of the input data.

2D Convolutions are computed by moving a square window of defined height and width across all pixels of the input feature map, which connect to each local region or receptive field of the image corresponding to the size of the applied filters. For images, the input feature map has three dimensions: height, width and colour bands, which commonly correspond to red, green, and blue. When these feature maps perform 2D convolution, two dimensional patches of surrounding features are created. These patches are then turned into a one-dimensional vector reflecting the output depth via a convolution kernel. After that, the vectors are reassembled spatially into a two-dimensional output map that corresponds to all points in the input map. The convolution process is depicted in the figure below.



In other words, a feature map is the output of a convolutional layer, which is typically a three-dimensional tensor with dimensions width, height, and depth. To process inputs or feature maps, the convolutional layer employs a filter, similar to the filtering procedure in picture pre-processing. The primary distinction is that in standard image pre-processing, the filter weights (values) are hand-crafted, whereas in CNNs, during training, data helps they learn. Alternatively, the receptive field in the input feature map corresponds to the output of a convolutional layer, and the size of the receptive field is governed by the kernel size and dilation. It is not essential to flatten images while using CNNs. However, flattening is required to feed the features maps, being the outputs of convolutional layers, to the fully connected layers. Fully connected layers are the same as the ANNs mentioned above, however ANN is a catch-all term for all artificial neural networks, including CNNs. Fully connected layers understand the linear and non-linear correlations between extracting features and classifying each sample into many abstract classes. This figure shows a simple CNN architecture.



The size of the output volume is controlled by hyperparameters which are depth, stride, and zero-padding.

The first component of the output volume decision is depth, which is the number of filters desiring to utilize, each learning to search for something different in the input. For example, if the first Convolutional Layer receives the raw image as an input, distinct neurons along the depth dimension may fire in the presence of different oriented edges or colour blobs. A column of depth is a group of neurons being all staring at the same region of the input [9].

Stride aids in picture and video data compression tuning. The stride of the convolution is a characteristic which can contribute to a different output size in CNNs, which is a convolution operation parameter that defines the distance between patches derived from the input feature map. With a stride of 2, the output feature map's width and height are down-sampled by a factor of two with no padding. For example, if the stride of a neural network is set to 1, step by step, the filter will make a reflection onto the matrix of input, which move one pixel or one unit at a time from left to right in the row till the end of horizontal point and top to bottom in the column at the end of the matrix. Because the size of the filter influences the encoded output volume, stride is frequently a positive integer rather than a fraction or decimal.

It is sometimes useful to pad the input volume with zeros around the border. This zero-size padding's is a hyperparameter. The great thing about zero padding is that it allows to tweak the spatial size of the output volumes, which most typically using it to precisely preserve the spatial size of the input volume so as to be same of the input and output width and height [9].

Combination of these components will give out the formula for calculating the size of Convolution layer:

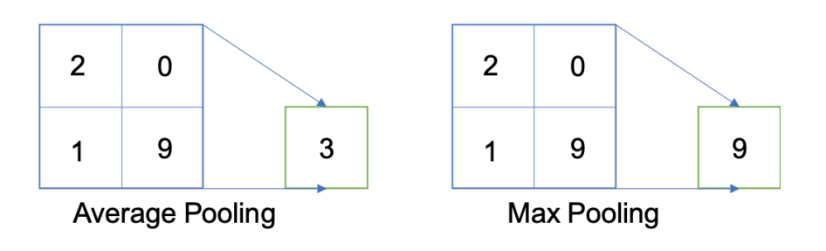
***[(N – F + 2P) / S + 1] x [(N – F + 2P) / S + 1] x C’***

The spatial dimension of the output volume may be calculated as a function of the input volume size ***(N)*** which represent for the dimension by width and height of input image, the receptive field size of the Conv Layer neurons ***(F)*** as the dimension of filter by width and height, the amount of zero padding placed the border for input image ***(P)***, the positive integer number of strides with which they are applied ***(S)***, and the number of filter ***(C’)***.

* 1. **Pooling Layer**

There are also pooling layers between convolutional layers. They are used to down sample the feature maps and reduce the number of parameters. The later layers after feature extraction will need a large number of parameters because the depth is determined by the number of channels in the later layers often increasing exponentially. That increases the number of parameters and the amount of computation in the neural network. Therefore, to reduce the computational load we will need to reduce the dimensions of the input matrix block or reduce the number of layer units. Since each unit would be a representative result of applying a filter to find a specific feature, reducing the number of units would not be feasible. Reducing the input matrix block size by finding a representative value for each spatial region that the filter passes through will not change the main contours of the image but reduce the size of the image. Hence the matrix reduction process is applied.

Average pooling and max pooling are the two most commonly used types of pooling layers helping make this reduction purpose. As the suggestion of their names, average pooling takes out the average (mean) of the values on the feature map corresponding to a kernel size. Besides, max pooling calculates the maximum value in the neighbourhood corresponding to the kernel size position on the feature map. Both of them travel through the whole feature map by a stride size without overlapping local region position. In addition, the pooling layers do not have any parameters which need to be learned during training and the stride in the convolutional layer is able to also be used to replace the pooling layers. Figure below shows the difference between these two pooling types.



* 1. **Batch Normalization:**

Batch normalization (BN) is another important component of a CNN architecture. Similar to normalizing the inputs images, BN normalizes the feature maps at the output of a convolutional layer. BN subtracts the batch mean and divides the values in the feature map by batch standard deviation so that the distribution of feature maps will be the same. Consequently, BN reduces the value shift in hidden layers. BN provides more freedom on the initialization of kernel weights. In addition, higher learning rates can be used, since BN makes sure that there is no activation diverging to very large values. Furthermore, it also has some regularization effects like a dropout layer, since it adds some noise to the feature maps. Both BN and dropout layers will act differently during training and evaluation.

In deep learning, we want the function of probability density (FPD) of the model to be as close to the true real data distribution as possible. However, we only have limited amount of training data available. Thus, the goal becomes to make FPD of model as close to FPD of train as possible, by training on a dataset that represents the real-world application domain. To achieve better representation, a sufficiently large training dataset is needed. When the training dataset is not large enough, there will be a large error between FPD of real and FPD of train. In other words, even though we achieve to obtain a model FPD close to FPD of train, it will not be able to perform well with FPD of test which is a sample of FPD of real. This is called the overfitting problem. When overfitting happens, model will fail to fit future observations. To avoid overfitting, dropout is used, which randomly chooses some neurons and ignores them in the training process. Incoming and outgoing edges to a dropped-out node are also removed. By avoiding to train with all nodes, dropout decreases the problem of overfitting, and also improves the training speed. The components of a classic convolutional layer are shown in Table below