



PIZVI COLLEGE OF ENGINEERING

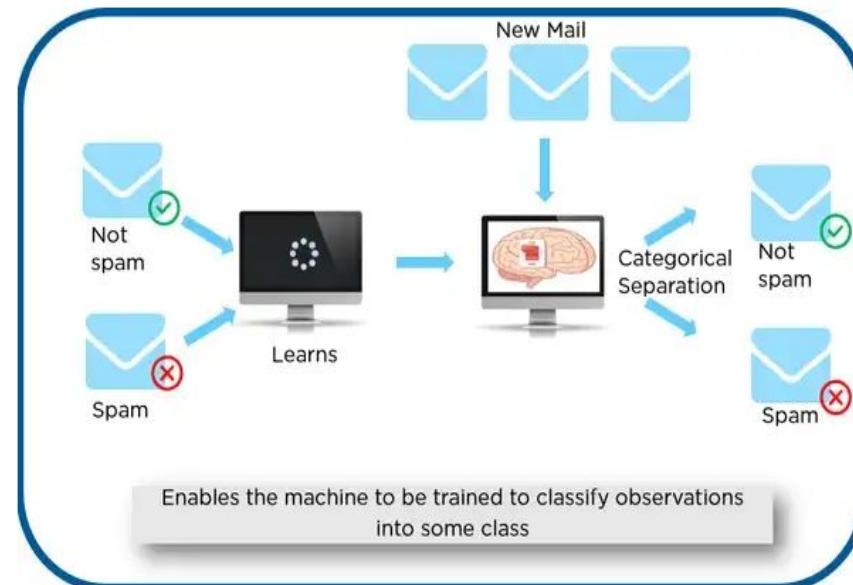
DEEP LEARNING

Topics

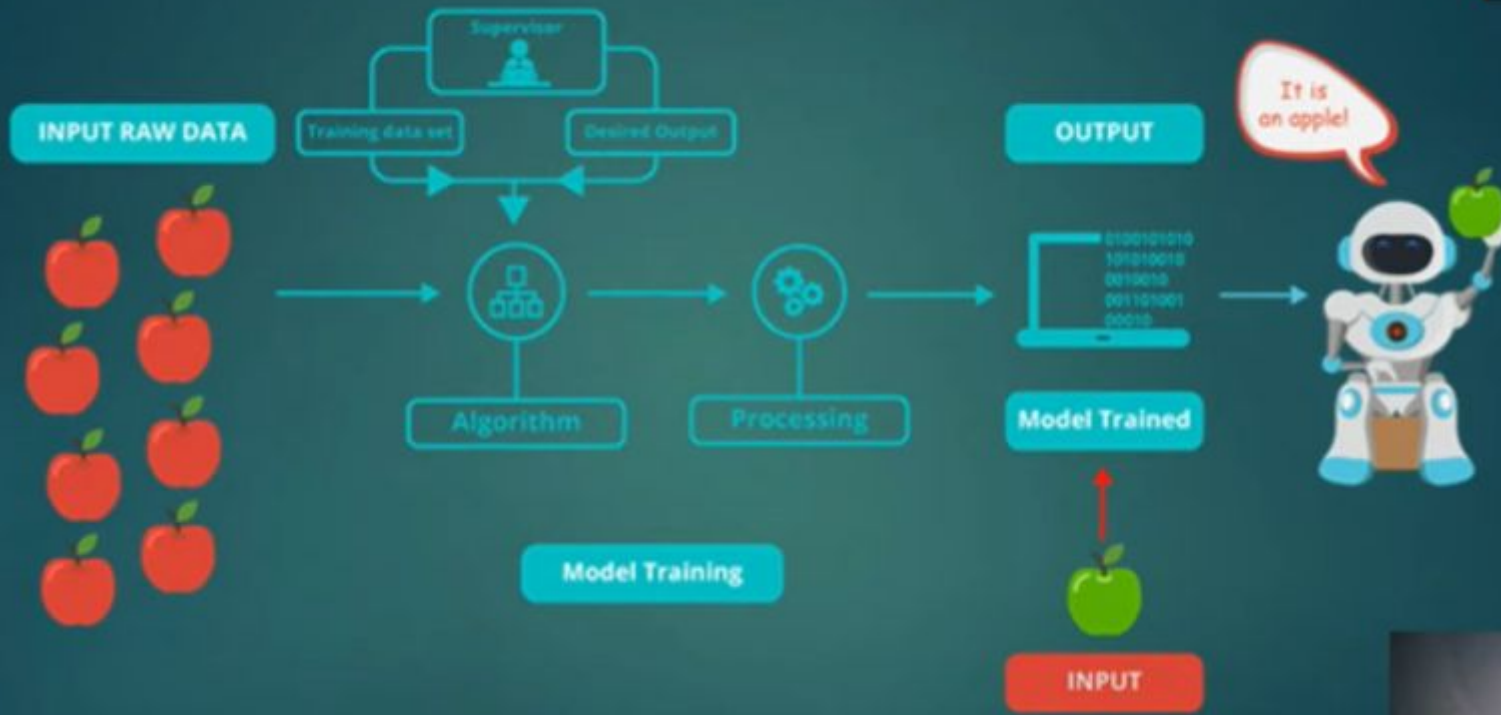
4		Convolutional Neural Networks (CNN): Supervised Learning
	4.1	Convolution operation, Padding, Stride, Relation between input, output and filter size, CNN architecture: Convolution layer, Pooling Layer, Weight Sharing in CNN, Fully Connected NN vs CNN, Variants of basic Convolution function, Multichannel convolution operation, 2D convolution.
	4.2	Modern Deep Learning Architectures: LeNET: Architecture, AlexNET: Architecture, ResNet : Architecture

Supervised learning

- ❖ **SUPERVISED LEARNING**: In supervised learning, models are trained using labelled dataset, where the model learns about each type of data. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.
- ❖ Eg: Data is marked with spam, not spam in training.



EXAMPLE



Steps Involved in Supervised Learning:

- First Determine the type of training dataset
- Collect/Gather the labelled training data.
- Split the training dataset **into training dataset(model learns the task), validation dataset(which model is the best) and test dataset(how good is this model).**
- Determine the input features of the training dataset, which should have enough knowledge so that the model can accurately predict the output.
- Determine the suitable algorithm for the model, such as support vector machine, decision tree, etc.
- Execute the algorithm on the training dataset. Evaluate the accuracy of the model by providing the test set. If the model predicts the correct output, which means our model is accurate.

SUPERVISED LEARNING can be further divided into two types of problems:

Regression (entails the prediction of a class label)

Regression algorithms are used if there is a relationship between the input variable and the output variable. It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc.

Few examples of Regression algorithms under supervised learning:

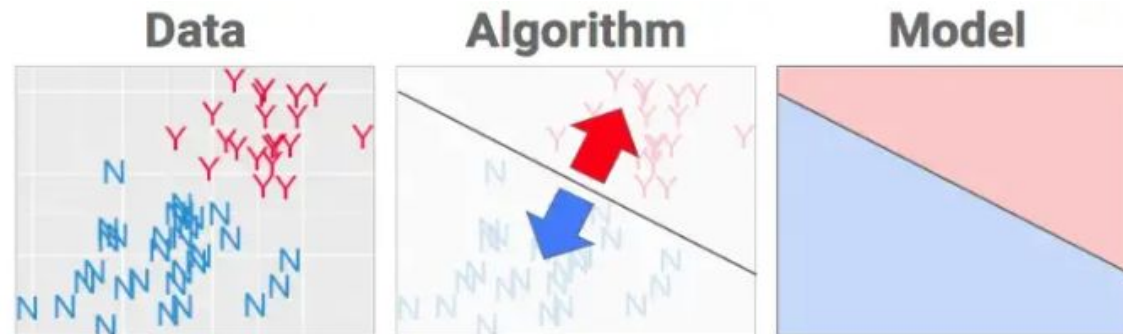
Linear Regression, Regression Trees, Non-Linear Regression, Bayesian Linear Regression, Polynomial Regression

Classification (entail the prediction of a numerical value)

Classification algorithms are used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Female, True-false.

Few examples of Classification algorithms under supervised learning:

Random Forest, Decision Trees, Logistic Regression, Support vector Machines.



Classification of deep learning:

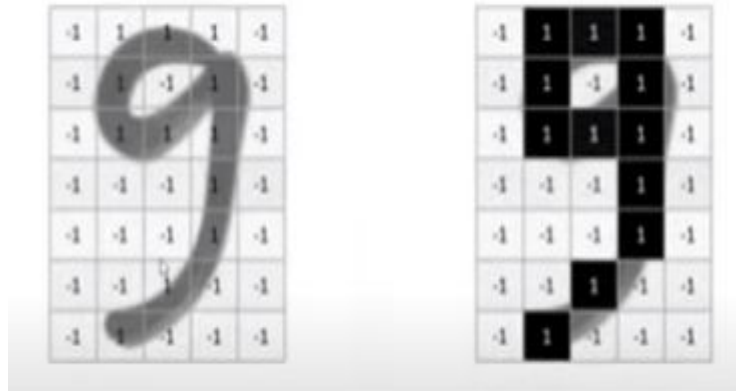
- Supervised Learning
 - ☐ Artificial Neural Network
 - ☐ Recurrent Neural Network
 - ☐ Convolutional Neural Network
- Unsupervised learning
 - ☐ Self-organizing maps
 - ☐ Boltzmann Machine
 - ☐ Autoencoders

Introduction to CNN

- Convolutional Networks (LeCun, 1989), are also known as Convolutional Neural Networks or CNNs sometimes also called ConvNets.
- It is a feed-forward neural network as the information moves from one layer to the next. It consists of hidden layers having convolution and pooling functions in addition to the activation function for introducing non-linearity.
- Used for image recognition and processing.
- Detect features in an image, such as edges, corners, and textures.

- Computers see an input image as an array of pixels. The filters are applied to small regions of the image, and as the filters move across the image, they create a set of feature maps that capture different aspects of the image.
- These feature maps are then fed into a series of convolutional layers, which combine the feature maps to create more complex representations of the image.
- Finally, the output of the convolutional layers is fed into one or more fully connected layers, which perform classification or regression tasks based on the features learned from the image.

- Computer vision:
- In computer vision, images are typically represented as a grid of pixels. Each pixel represents a small part of the image and is assigned a value that corresponds to its color or grayscale intensity.
- The pixels in an image are arranged in rows and columns to form a grid. The number of rows and columns in the grid depend on the size and resolution of the image. For example, an image with a resolution of 1280 x 720 has 1280 columns and 720 rows of pixels.
- When a computer processes an image, it reads the pixel values in the grid and uses them to analyze and manipulate the image. In a CNN, the filters are applied to small regions of the image, which correspond to a small set of adjacent pixels. By analyzing the values of these pixels, the filters can detect features such as edges, corners, and textures.



Convolution Network ex:

- CNN first learns to recognize the components of an image and then combine these components (pooling) to recognize the larger structure of the whole image.

One challenge is to detect the edges, horizontal and vertical edges. We create this notion of a filter (aka, a kernel). Consider an image as the following

$$\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix}$$

and we take a filter (aka, a kernel, in mathematics) which takes the following form,

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

and we apply "convolution" operation, which is the following

$$\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

and the first element would be computed as

$$3 \times 1 + 1 \times 1 + 2 \times 1 + 0 + 0 + 0 + 1 \times -1 + 8 \times -1 + 2 \times -1 = -5$$

Vertical and Horizontal filters

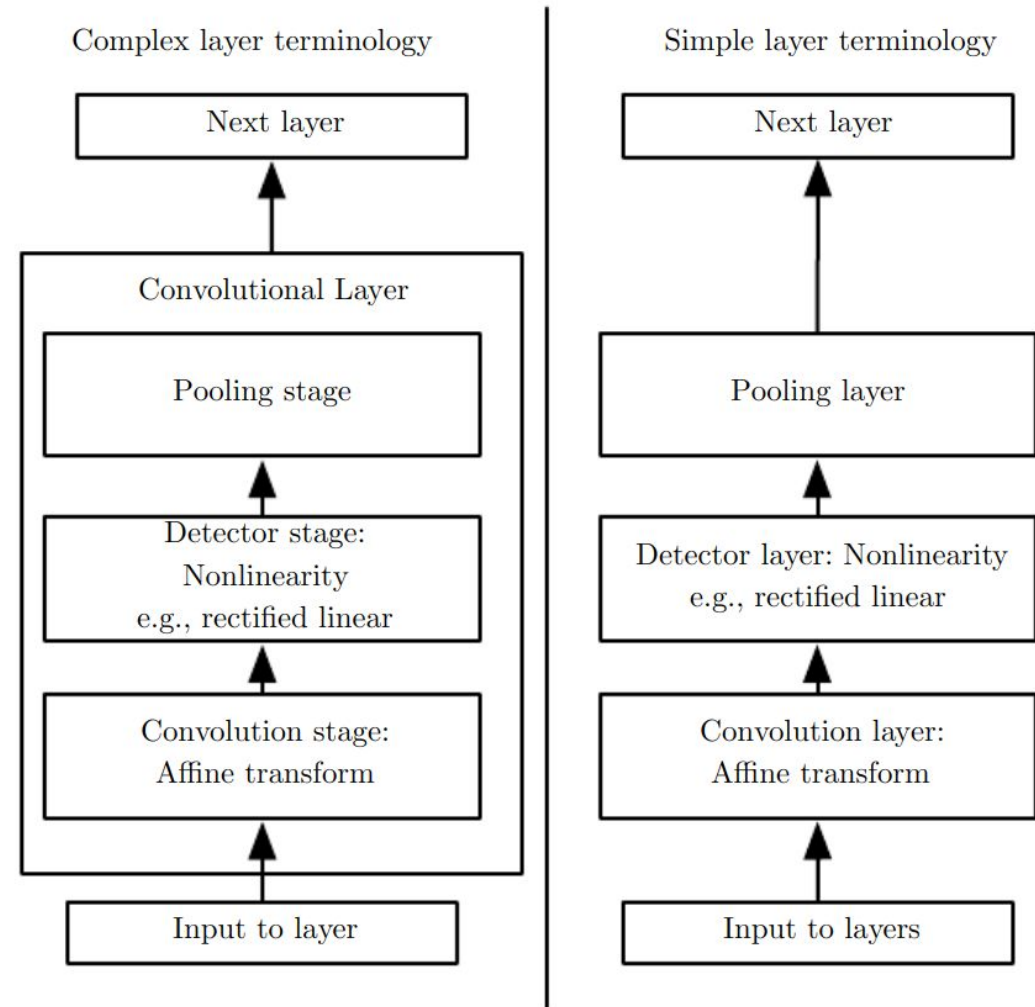
Shifting the 3 by 3 matrix, which is the filter (aka, kernel), one column to the right, and we can apply the same operation. For a 6 by 6 matrix, we can shift right 4 times and shift down 4 times. In the end, we get

$$\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -5 & -4 & 0 & 8 \\ -10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}, \text{ and } \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix},$$

The components of Typical CNN Layer

- There are two commonly used sets of terminology for describing these layers.
- Complex layer: In this terminology, the convolutional net is viewed as a small number of relatively complex layers, with each layer having many “stages.” In this terminology, there is a one-to-one mapping between kernel tensors and network layers.
- Simple layer: In this terminology, the convolutional net is viewed as a larger number of simple layers; every step of processing is regarded as a layer in its own.

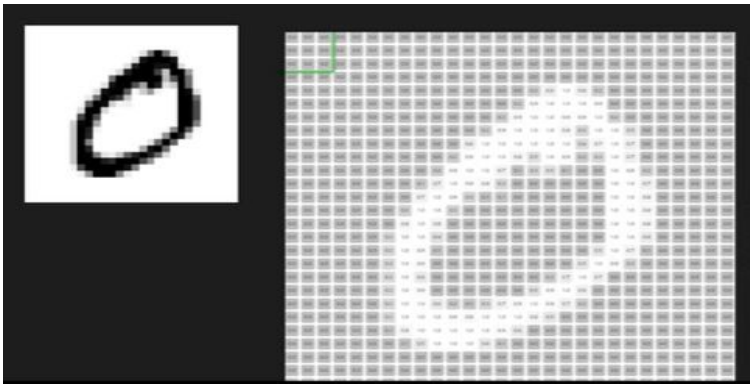


- Input Layer: This is where the raw image data is fed into the network. Each pixel in the image corresponds to an input node in this layer.
- Convolutional Layer: This layer performs convolutions on the input image. Convolutions involve sliding a small filter (also called a kernel) over the image and performing element-wise multiplication and summing to produce a feature map. This helps the network learn features like edges, textures, and patterns.
- Pooling Layer: This layer reduces the spatial dimensions (width and height) of the feature maps while retaining important information. Max pooling, for instance, selects the maximum value in a certain window and discards the rest, thus reducing computation and controlling overfitting.
- There are two commonly used sets of terminology for describing these layers.
- Complex layer: In this terminology, the convolutional net is viewed as a small number of relatively complex layers, with each layer having many “stages”. In this terminology, there is a one-to-one mapping between kernel tensors and network layers.
- Simple layer: In this terminology, the convolutional net is viewed as a larger number of simple layers; every step of processing is regarded as a layer in its own. This means that not every “layer” has parameters.

The convolution operation

- Convolution is an operation on two functions of a real-valued argument. (main task= extraction of features)
- Suppose a location of a spaceship is tracked with a laser sensor. The laser sensor provides a single output $x(t)$, indicating the position of the spaceship at time t . Both x and t are real-valued, such that we can get a different reading from the laser sensor at any instant in time. Suppose that laser sensor is somewhat noisy.
- To obtain a less noisy estimate \square several measurements will need to be averaged \square more recent measurements \square more relevant.

- Example: Gray Scale Image. Imagine this as 28*28 pixel image with values 0 to 255.



- Initialize CNN filter with random values and with back-propagation it will decide the rest of the values of the filter. These values will be decided during the training.

- <https://deeplizard.com/convolution>

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

 $*$

-1	-1	-1
0	0	0
1	1	1

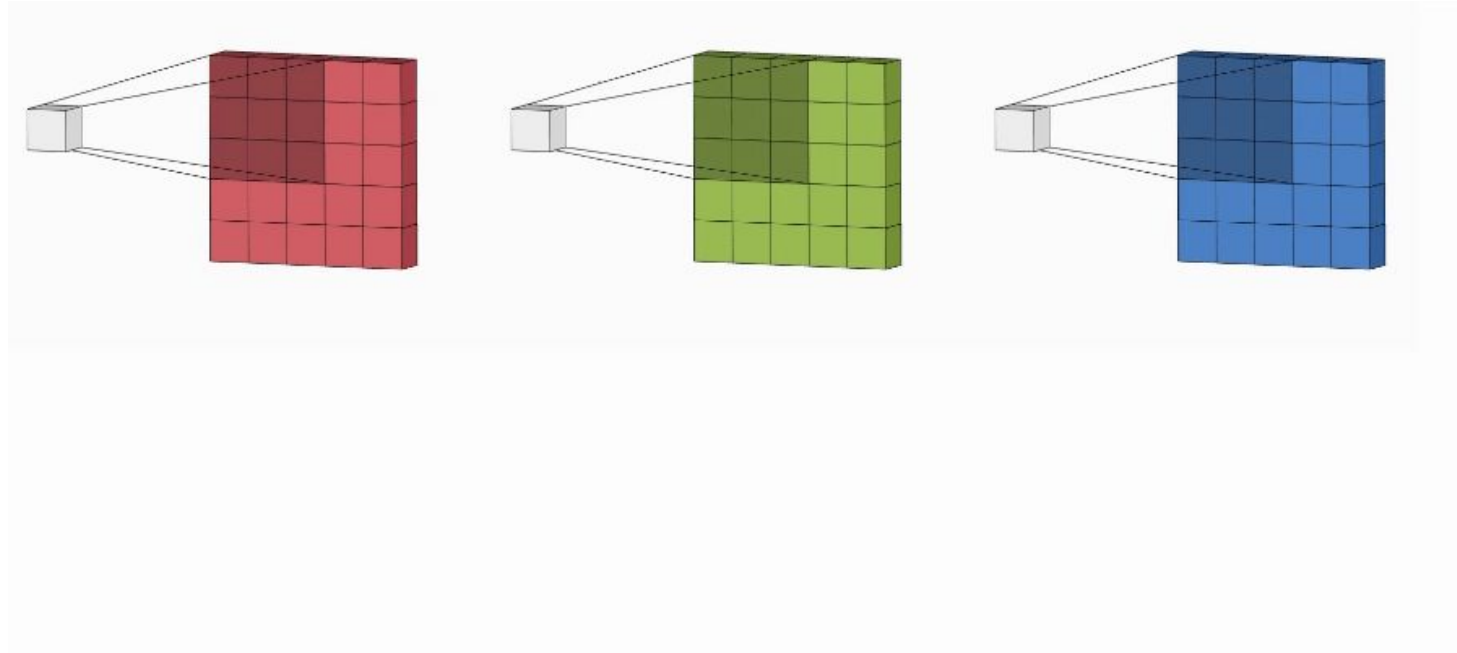
 $=$

2

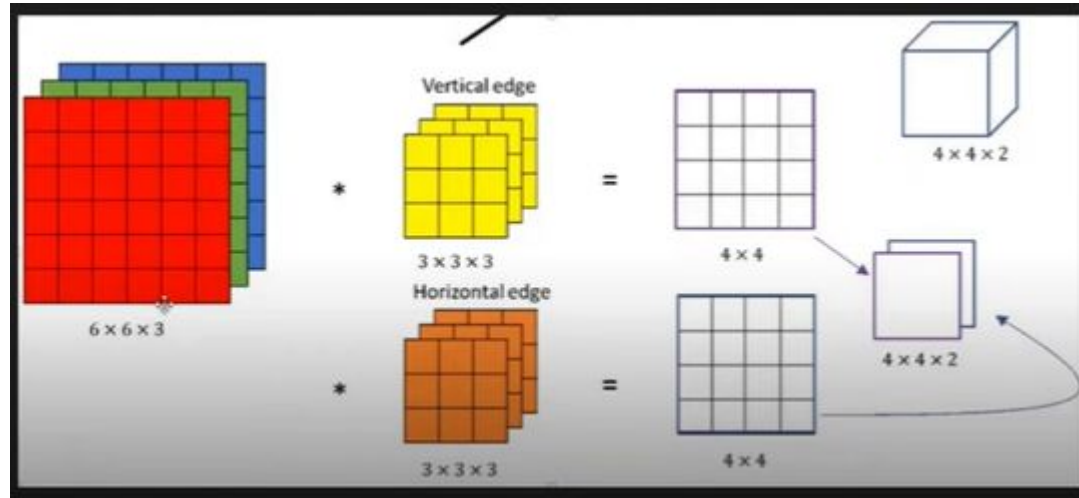
5

5

- RGB



- For Multiple features



- This can be done with a weighting function $w(a)$, where a is the age of a measurement. If such a weighted average operation is applied at every moment, a new function (s) is obtained providing a smoothed estimate of the position of the spaceship.
- $s(t) = \int x(a) w(t-a) da$
- This operation is called as convolution. The convolution operation is typically denoted with an asterisk: $s(t) = (x * w)(t)$
- Here, w needs to be a valid probability density function, or the output is not a weighted average. Also, w needs to be 0 for all negative arguments, or it will look into the future, which is presumably beyond our capabilities.

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a) w(t-a)$$

- In a convolutional neural network, we often talk about two things: "input" and "kernel." The input is the data we want to process (like an image), and the kernel is like a filter or a small grid of numbers we use to extract features from the input.

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n) K(i-m, j-n)$$

- Imagine you have data that's collected over time, like measurements from a sensor taken every second. We can think of this data as a sequence of numbers, and we want to perform a special kind of operation on it called "convolution."

- Convolution is like sliding the kernel over the input data and multiplying and adding numbers at each step to get a new set of numbers, which we call the "feature map." It helps us find important patterns or features in the data.

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n) K(m, n)$$

- Now, sometimes we're not just dealing with one-dimensional data like a sequence. For example, when we work with images, we have a grid of numbers. So, we use a two-dimensional kernel and perform a similar operation to find features in the image.

- In math, you can write the convolution operation in two ways: one where you flip the kernel, and another where you don't. But in machine learning, we usually use the version where we don't flip the kernel. It's a bit easier to work with in software.

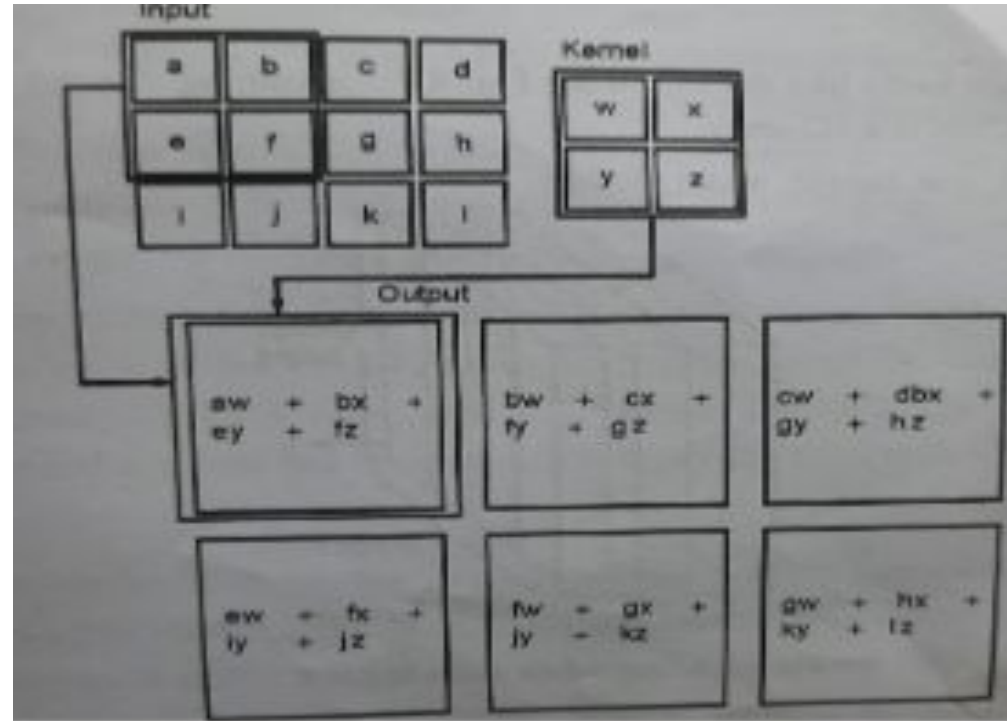
$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n) K(m, n)$$

- So, in simple terms, convolution is a way to process data, like images or sequences, to find important features. It involves sliding a small grid (kernel) over the data and doing some calculations to get a new set of numbers (feature map). While there are different math notations for it, in machine learning, we often use the version where we don't flip the kernel.

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n) K(m, n)$$

- Mathematically, the convolution and cross-correlation operations can be defined as:
- Convolution: $(f * g)[n] = \sum_k f[k] g[n-k]$
- Cross-correlation: $(f \otimes g)[n] = \sum_k f[k] g[n+k]$
- Many machine learning libraries implement cross-correlation but call it convolution. In the context of machine learning, the learning algorithm will learn the appropriate values of the kernel in the appropriate place, so an algorithm based on convolution with kernel flipping will learn a kernel that is flipped relative to the kernel learned by an algorithm without the flipping.

- Convolution without kernel flipping.



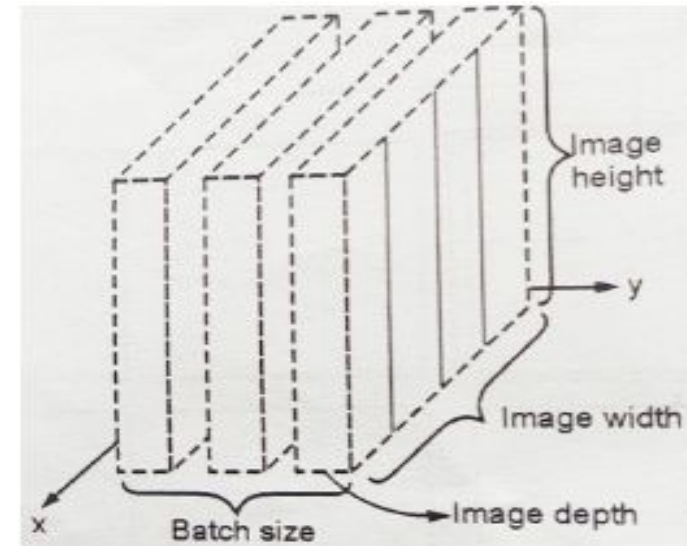
Motivation behind Convolution

- In mathematics and signal processing, convolution is a mathematical operation on two functions that produces a third function, which expresses how the shape of one function is modified by the other. It is a specialized type of linear operation.
- Properties:
 - 1. Sparse interaction
 - 2. Parameter sharing
 - 3. Equivariant representation
- Parameter sharing is a feature detector (such as a vertical edge detector) that is useful in one part of the image is probability useful in another part of the image. Parameter sharing implies that the same set of weights or filters is used across the entire input. This makes the data computationally efficient and capable of learning meaningful patterns.
- Sparse interaction means that in convolution, each element in the output (resulting) function depends only on a limited portion of the input function. In other words, not every part of the input affects every part of the output, which is especially useful when dealing with large datasets.
- Equivariant representation means that when the input data is transformed, the output is also transformed in a corresponding manner. In other words, if you change something in the input, you'll see a related change in the output. This property is valuable for capturing meaningful patterns in data.

Some Important Terminologies

1. Input shape

- The input data to CNN looks like the following assuming the data is a collection of images. Input to CNN is a 4D array.
- Input shape = (batch size, height, width, depth)
- Batch size is the number of training examples in one forward/backward pass and depth of the image, is the number of color channels.
- RGB image has a depth of 3, and the greyscale image has depth of 1.



2. Output shape

- The output of the CNN is also a 4D array.
- Output shape (batch_size, height, width, depth)
- Where batch size would be the same as input batch size but the other 3 dimensions of the image might change depending upon the values of filter, kernel size, and padding.

3. Filter

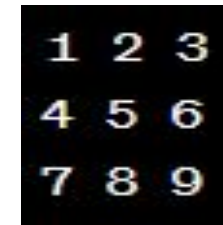
- In a convolution neural network, input data is convolved over with a filter which is used to extract features. Filter or a kernel is a matrix that moves over the image pixel data (input). It performs a dot product with that particular region of the input data and outputs the matrix of the dot product.



4. Padding

- Padding works by extending the area of an image processed by a convolutional neural network. The kernel is the neural network filter which moves across the image, scanning each pixel and converting data into a smaller, or sometimes larger, format. In order to assist the kernel with processing the image, padding is added to the frame of the image to allow for more space for the kernel to cover the image. Adding padding to an image processed by a CNN allows for more accurate analysis of images.
- In simple words, Padding in convolution refers to the process of adding extra pixels or values around the edges of an input image before applying a convolutional operation. The main purpose of padding is to preserve the spatial dimensions of the input image and to avoid reducing the size of the output feature map.

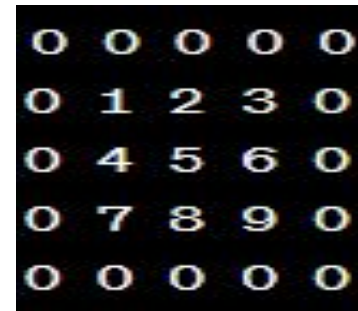
- Let's use an example to make it clear:
- Imagine you have a 3x3 grid of numbers representing an image:



1	2	3
4	5	6
7	8	9

- Now, if you want to apply a 2x2 convolution operation to this image, the kernel (a small grid used for calculations) will slide over it. Without padding, when the kernel reaches the edges of the image, it won't fully cover all the original image pixels. You'll get a smaller output, losing some information from the edges.
- But if you add padding, it's like adding an extra row and column of zeros around the image:

- Now, when you slide the kernel, it can fully cover all the original image pixels, and you get an output that's the same size as the input. Padding helps preserve the spatial information at the edges.



0	0	0	0	0
0	1	2	3	0
0	4	5	6	0
0	7	8	9	0
0	0	0	0	0

- So, in simple terms, padding in deep learning is about adding extra space or values around your data to make sure you don't lose important information when applying operations like convolution. It helps maintain the input size and keeps the edges of your data intact during processing.

5. Stride

- The number of rows and columns traversed per slide are referred as stride. It can be thought as by how many pixels we want our filter to move as it slides across the image. Strides of 1, both for height and width can be used or sometimes, larger strides are used. When the stride is one the filter is moved to one pixel at a time and when the stride is two the filter is moved to two pixels at a time. (how fast the filter moves across a picture)

6. Tensors

- In machine learning applications, the input is usually a multi-dimensional array of data and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm. These multidimensional arrays are called as tensors. A tensor can be a generic structure that can be used for storing, representing, and changing data.

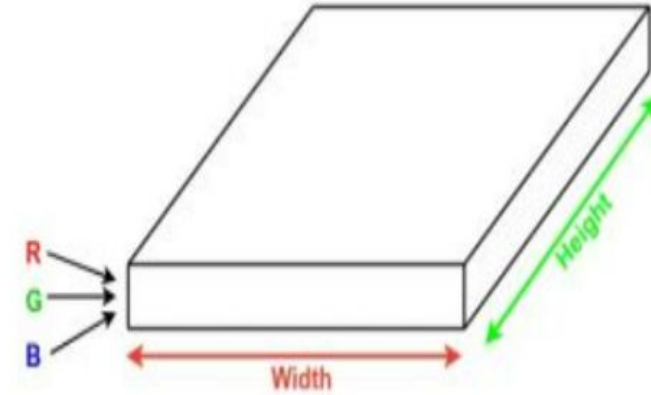
Relation between input, output and filter size

In Convolutional Neural Networks (CNNs), the filter size, input size, and output size are interrelated.

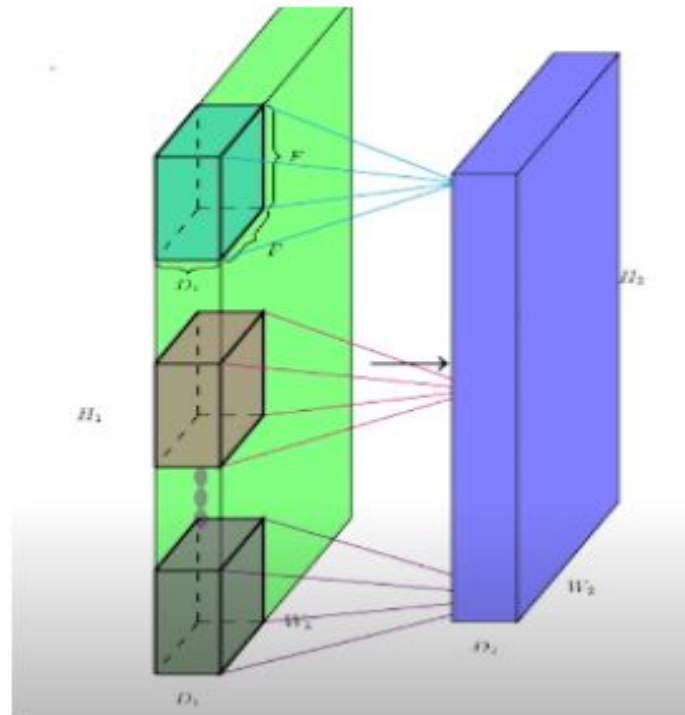
The input to a CNN is typically an image or a feature map.

The size of the input is represented by its height, width, and depth (number of channels). The output of a CNN is also an image or a feature map, with a height, width, and depth that depends on the architecture of the network and the configuration of its layers.

In most cases, depth is 3(RGB) or 1(Grayscale) images.



- Width (W_1), Height (H_1) and Depth (D_1) of the original input.
- Stride S .
- Number of filters: K .



The spatial extent (F) of each filter (the depth of each filter is same as the depth of each input)

The output is $W_2 \times H_2 \times D_2$ (we will soon see a formula for computing W_2 , H_2 and D_2)

So what should our final formula look like,

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

- Example: The input given is \square filter = 1 \square kernel size=(3,3) \square input shape=(10,10,1)

1. Number of parameters in the convolution layer:

- Weights in one filter of size $(3, 3) = 3*3=9$
- Bias: 1[One bias will be added to each filter. Since only one filter kernel is used, bias=1]
- In CNNs, biases are additional parameters that are learned alongside the weights. They allow the network to shift the activation function and therefore the output of the layer.
- Total parameters for one filter kernel of size $(3, 3) = 9+1 = 10$

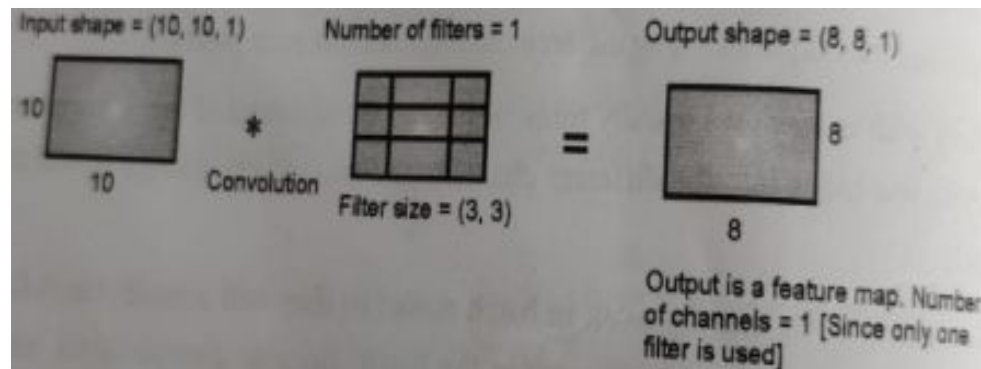
2. Calculating the output shape:

- Output shape:

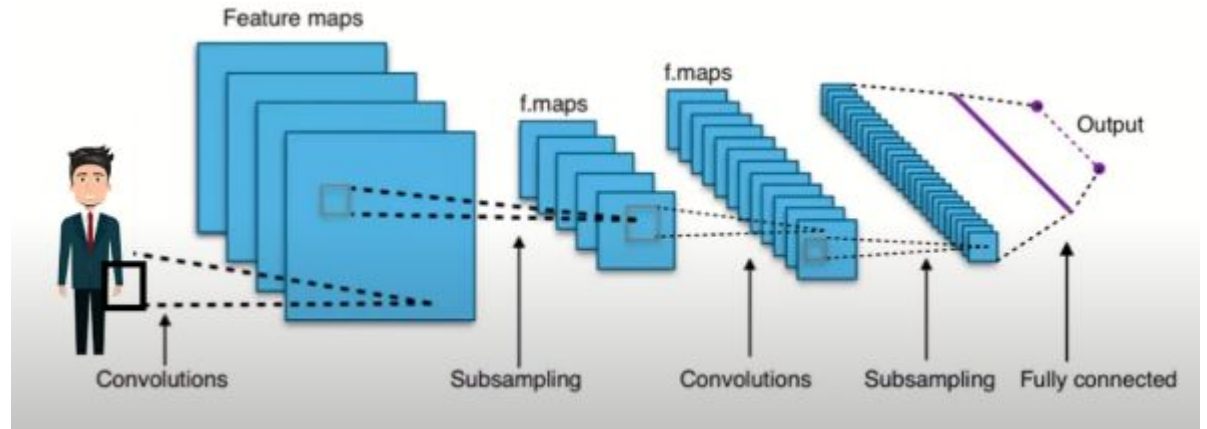
$$\frac{n + 2p - f}{s} + 1$$

- Where $s \rightarrow$ stride, $p \rightarrow$ padding, $n \rightarrow$ input size, $f \rightarrow$ filter size.
- Stride by default = 1, padding is not mentioned (so, $p = 0$)
- Hence, output shape = $n - f + 1 = 10 - 3 + 1 = 8$

- After applying convolution on the input image using a convolution filter, the output will be a feature map. The number of channels in the feature map depends on the number of filters used. Here, in this example, only one filter is used. So, the number of channels in the feature map is 1.
- So, Output shape of feature map = (8, 8, 1)



CNN Architecture



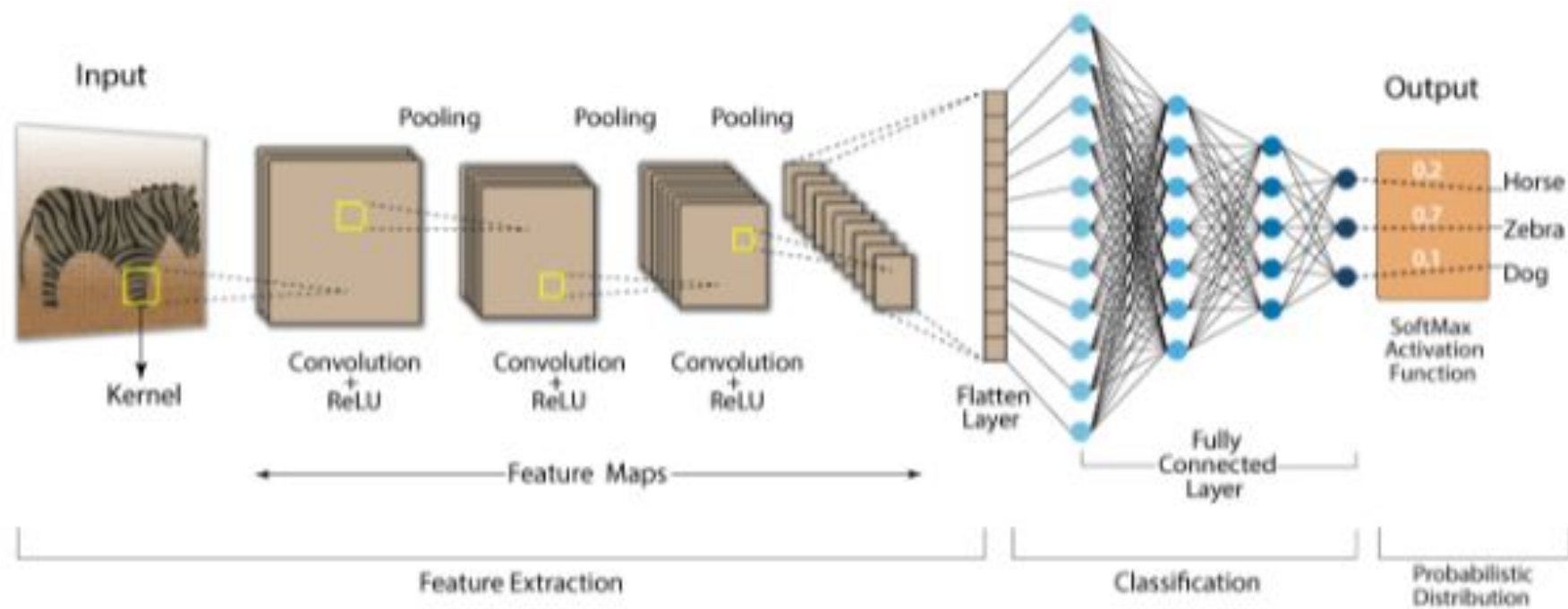
- Convolution layer:
- A filter passes over the input image, scanning the pixels and creating a feature map.
- Pooling layer:
- Down sampling or subsampling is a method of reducing number of pixels without losing the important information.
- With max pooling, we could retain the strong pixels while ignoring the weaker pixels.

- **Flattening:** Output of previous layer is flattened to a single vector and provided as an input to next layer.

The first fully connected layer — takes the inputs from the feature analysis and applies weights to predict the correct label.

Fully connected output layer — This gets us the final probabilities for each label.

- **Softmax** activation is a type of activation function used in convolutional neural networks (CNNs). It is used to normalize the output of a neural network into a probability distribution, which can then be used to make predictions.



- **Convolution Layer**

- The convolution layer is a core building block of CNN. It plays an important role in handling the main portion of the network's computational load. At this layer a dot product is performed between two matrices, where one matrix is a set of learnable parameters also known as a kernel, and the other matrix represents the restricted portion of the receptive field.
- During the forward pass, the kernel slides across the height and width of the image and generates a two-dimensional image representation of that receptive region. This is known as an activation map that gives the response of the kernel at each spatial position of the image. The sliding size of the kernel is called a stride.
- Consider the input image of size $W \times W \times (d)$. D_{out} is the number of kernels having a spatial size F . stride S and amount of padding is P . The size of output volume is given by the following formula:
- $W_{out} = \text{Output volume: } W_{out} \times W_{out} \times D_{out}$

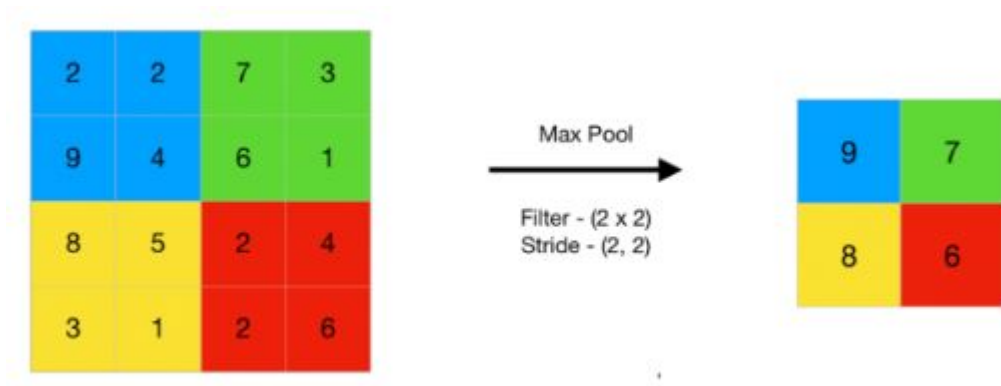
- **Pooling Layer**

- A typical layer of a convolutional network consists of three stages:

1. In the first stage, the layer performs several convolutions in parallel to produce a set of linear activations.
2. In the second stage, each linear activation is run through a nonlinear activation function, such as the rectified linear activation function. This stage is sometimes called the detector stage.
3. In the third stage, we use a pooling function to modify the output of the layer further. A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. For example, the max pooling operation reports the maximum output within a rectangular neighborhood.

- **Max pooling:**

Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.





Input

*



1 filter

=

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

maxpool
2x2 filters (stride 2)

8	4
7	5

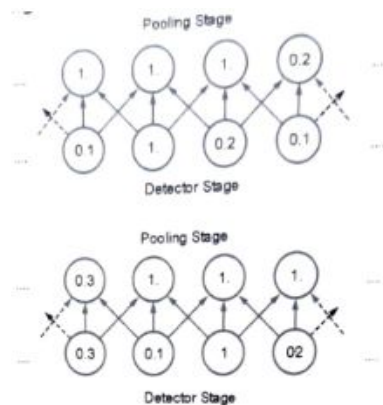
1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

maxpool
2x2 filters (stride 1)

8	8	4
8	8	5
7	6	5

Max pooling introducing invariance:

Invariance refers to the property of an object, system, or phenomenon that remains unchanged under certain transformations or operations. In other words, an object or system is said to be invariant under a particular transformation if it appears the same before and after the transformation.



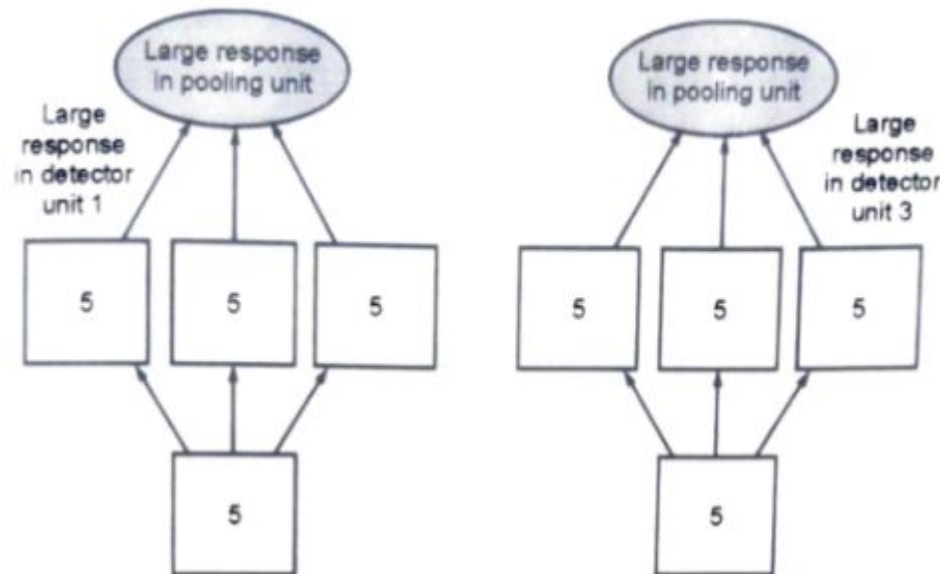
- Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is. For example, when determining whether an image contains a face, we need not know the location of the eyes with pixel-perfect accuracy, we just need to know that there is an eye on the left side of the face and an eye on the right side of the face. In other contexts, it is more important to preserve the location of a feature.

- (Top) A view of the middle of the output of a convolutional layer. The bottom row shows outputs of the nonlinearity. The top row shows the outputs of max pooling, with a stride of one pixel between pooling regions and a pooling region width of three pixels.
- (Bottom) A view of the same network, after the input has been shifted to the right by one pixel. Every value in the bottom row has changed, but only half of the values in the top row have changed, because the max pooling units are only sensitive to the maximum value in the neighborhood, not its exact location.

- Example of learned invariances:

Pooling over spatial regions produces invariance to translation, but if we pool over the outputs of separately parametrized convolutions, the features can learn which transformations to become invariant to.

A pooling unit that pools over multiple features that are learned with separate parameters can learn to be invariant to transformations of the input.



- Here we show how a set of three learned filters and a max pooling unit can learn to become invariant to rotation. All three filters are intended to detect a hand-written 5.
- Each filter attempts to match a slightly different orientation of the 5.
- When a 5 appears in the input, the corresponding filter will match it and cause a large activation in a detector unit.
- The max pooling unit then has a large activation regardless of which detector unit was activated. We show here how the network processes two different inputs, resulting in two different detector units being activated.

- **Pooling with down sampling:**

- Because pooling summarizes the responses over a whole neighborhood, it is possible to use fewer pooling units than detector units.

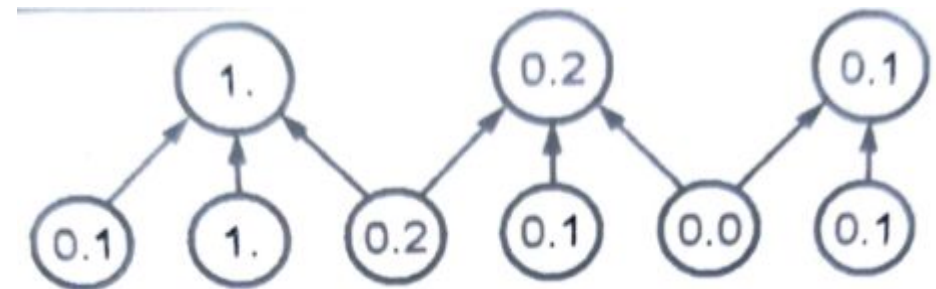
Here we use max-pooling with a pool width of three and a stride between pools of two. This reduces

the representation size by a factor of two, which reduces the computational and statistical burden

on the next layer.

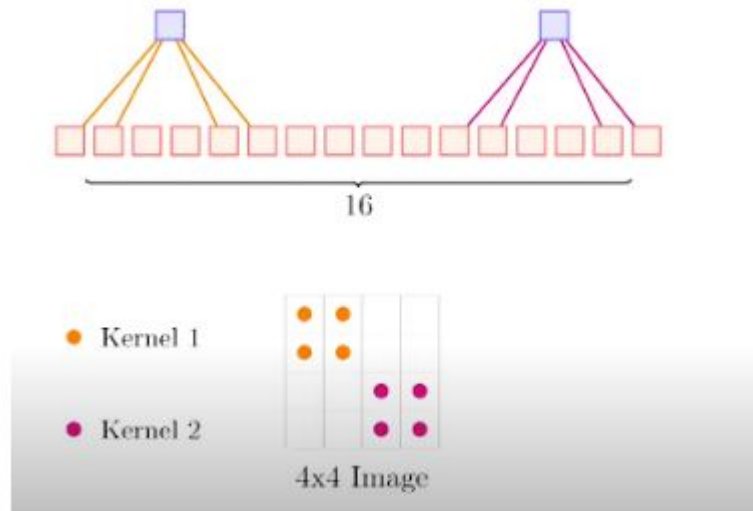
Note that the rightmost pooling region has a smaller size, but must be included if we do not want to

ignore some of the detector units.



- **This improves the computational efficiency of the network because the next layer has roughly k times fewer inputs to process. When the number of parameters in the next layer is a function of its input size (such as when the next layer is fully connected and based on matrix multiplication) this reduction in the input size can also result in improved statistical efficiency and reduced memory requirements for storing the parameters.**
- For many tasks, pooling is essential for handling inputs of varying size.
- **WEIGHT SHARING IN CNN**
- Weight sharing refers to the practice of using the same filter at multiple locations in the image.
- It allows the same set of weights to be used across multiple locations in an image.
- A typical application of weight sharing is in convolutional neural networks: CNNs work by passing a filter over the image input.
- Example, a 4×4 image and a 2×2 filter with a stride size of 2, this would mean that the filter (which has four weights, one per pixel) is applied four times, making for 16 weights total. A typical application of weight sharing is to share the same weights across all four filters.

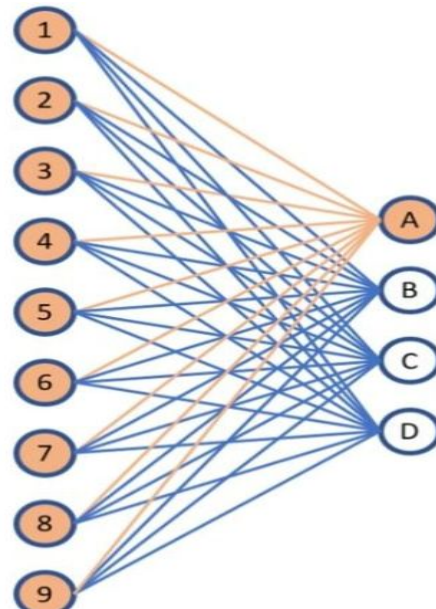
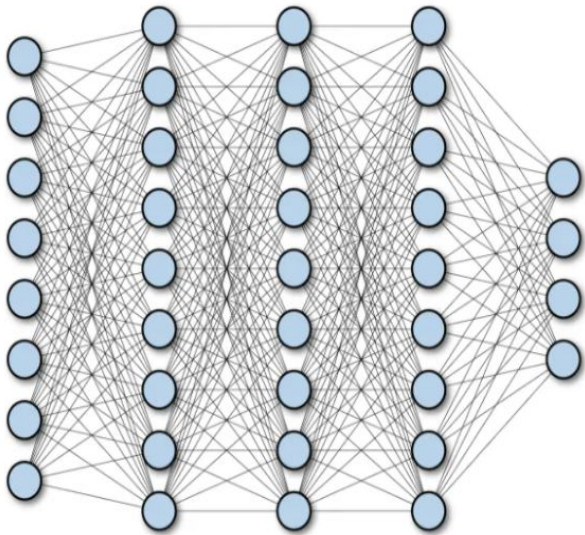
- In this context weight sharing has the following effects:
 - It reduces the number of weights that must be learned (from 16 to 4, in this case), which reduces model training time and cost.
 - It makes feature search insensitive to feature location in the image.
 - Weight sharing is for all intents and purposes a form of regularization.
- And as with other forms of regularization, it can actually increase the performance of the model.



CNN and FULLY CONNECTED NEURAL NETWORK:

A fully connected neural network

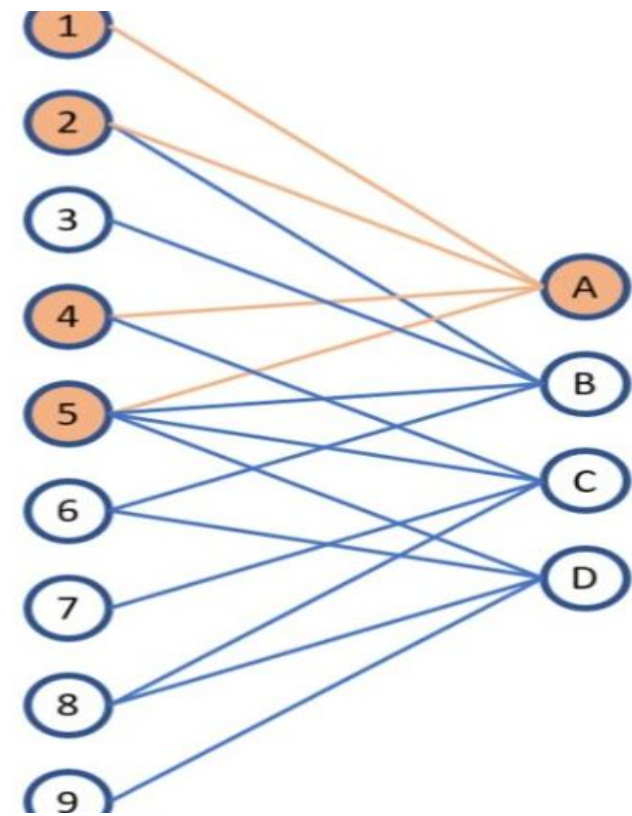
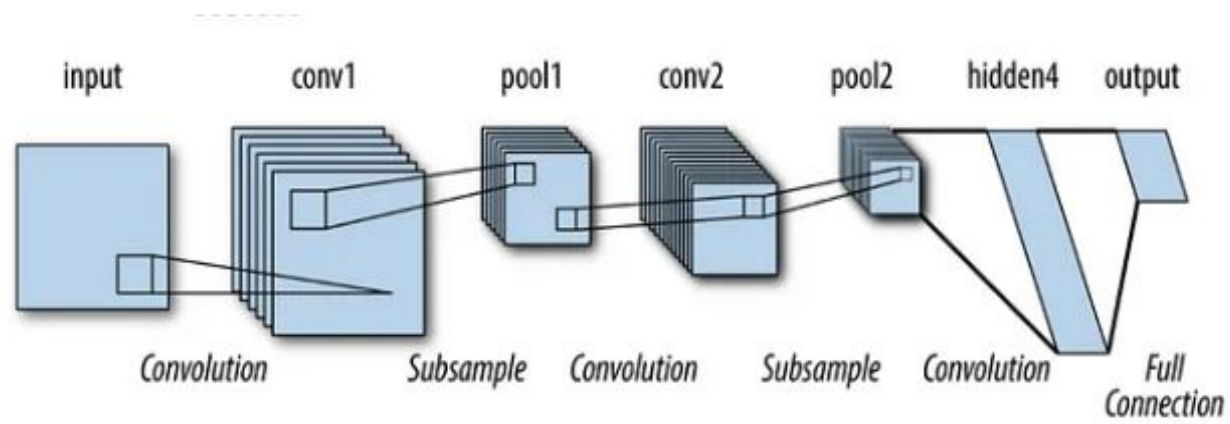
- It consists of a series of fully connected layers that connect every neuron in one layer to every neuron in the other layer.
- The major advantage of fully connected networks is that they are “structure agnostic” i.e. there are no special assumptions needed to be made about the input.
- While being structure agnostic makes fully connected networks very broadly applicable, such networks do tend to have weaker performance than special-purpose networks tuned to the structure of a problem space.



The orange lines represent the first neuron (or perceptron) of the layer. The weights of this neuron only affect output A, and do not have an effect on outputs B, C or D.

- **Convolutional Neural Network:**

- A simple CNN is a sequence of layers, and every layer of a CNN transforms one volume of activations to another through a differentiable function.
- Three main types of layers are used to build CNN architecture: Convolutional Layer, Pooling Layer and Fully-Connected Layer.
- Convolutional networks are among the first working deep networks those are trained with the technique of back-propagation. Convolutional networks became successful as they were more computationally efficient than fully connected networks. Hence, it was easy to run multiple experiments with them and tune the hyper parameters.
- Larger networks also seem to be easier to train. With modern hardware, large fully connected networks appear to perform reasonably on many tasks, even when using datasets that were available and activation functions that were popular during the times when fully connected networks were believed not to work well. It may be that the primary barriers to the success of neural networks were psychological (practitioners did not expect neural networks to work, so they did not make a serious effort to use neural networks).
- Convolutional networks provide a way to specialize neural networks to work with data that has a clear grid-structured topology and to scale such models to very large size. This approach has been the most successful on a two-dimensional, image topology.



Variants of basic convolution function:

Definition of 4-D kernel tensor

- Assume we have a 4-D kernel tensor \mathbf{K} with element $K_{i,j,k,l}$ giving the connection strength between
 - a unit in channel i of the output and
 - a unit in channel j of the input,
 - with an offset of k rows and l columns between output and input units
- Assume our input consists of observed data \mathbf{V} with element $V_{i,j,k}$ giving the value of the input unit
 - within channel i at row j and column k .
- Assume our output consists of \mathbf{Z} with the same format as \mathbf{V} .
- If \mathbf{Z} is produced by convolving \mathbf{K} across \mathbf{V} without flipping \mathbf{K} , then

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

- **A. Full convolution:**

- 0 padding 1 stride.

- If \mathbf{Z} is produced by convolving \mathbf{K} across \mathbf{V} without flipping \mathbf{K} , then

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

8

- 0 padding s stride

Convolution with a stride: Definition

- We may want to skip over some positions in the kernel to reduce computational cost
 - At the cost of not extracting fine features
- We can think of this as down-sampling the output of the full convolution function
- If we want to sample only every s pixels in each direction of output, then we can define a down-sampled convolution function c such that

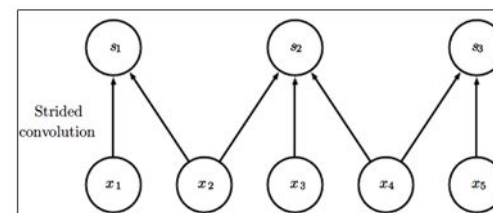
$$Z_{i,j,k} = c(\mathbf{K}, \mathbf{V}, s)_{i,j,k} = \sum_{l,m,n} [V_{l,(j-1) \times s + m, (k-1) \times s + n} K_{i,l,m,n}]$$

- We refer to s as the stride. It is possible to define a different stride for each direction

9

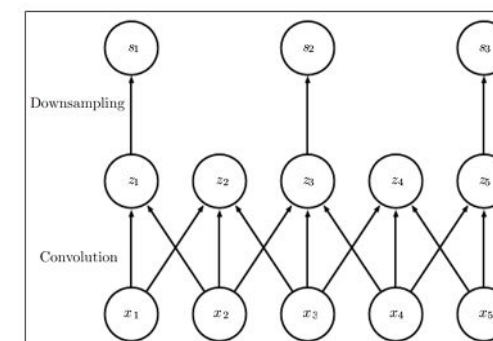
Some 0 paddings and 1 stride

- One essential feature of any convolutional network implementation is the ability to implicitly zero-pad the input V in order to make it wider.
- Without this feature, the width of the representation shrinks by one pixel less than the kernel width at each layer.
- Zero padding the input allows us to control the kernel width and the size of the output independently. Without zero padding, we are forced to choose between shrinking the spatial extent of the network rapidly and using small kernels both scenarios that significantly limit the expressive power of the network.



Here we use a stride of 2

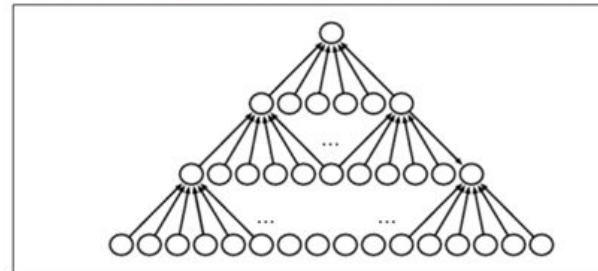
Convolution with a stride of length two implemented in a single operation



Convolution with a stride greater than one pixel is mathematically equivalent to convolution with a unit stride followed by down-sampling.

Two-step approach is computationally wasteful, because it discards many values that are discarded

Convolutional net with a kernel of width 6 at every layer
No pooling, so only convolution shrinks network size



We do not use any implicit zero padding
Causes representation to shrink by five pixels at each layer
Starting from an input of 16 pixels we are only able to have 3 convolutional layers and the last layer does not even move the kernel, so only two layers are convolutional



By adding 5 implicit zeroes to Each layer, we prevent the Representation from shrinking with depth
This allows us to make an arbitrarily deep convolutional network

Three special cases of zero padding

- Valid: no 0 padding is used.
- One is the extreme case in which *no zero-padding is used* whatsoever, and the convolution kernel is only allowed to visit positions where the entire kernel is contained entirely within the image. This is also called as valid convolution. In this case, all *pixels in the output are a function of the same number of pixels in the input*, so the behavior of an output pixel is somewhat more regular. However, the size of the output shrinks at each layer.
- Same: keep the size of the output to the size of input.
- If the input image has width m and the kernel has width k , the output will be of width $m-k+1$. The rate of this shrinkage can be dramatic if the kernels used are large. Since the shrinkage is greater than 0, it limits the number of convolutional layers that can be included in the network. As layers are added, the spatial dimension of the network will eventually drop to 1×1 , at which point additional layers cannot meaningfully be considered convolutional.

- Full: Enough zeros are added for every pixels to be visited k (kernel width) times in each direction, resulting width $m + k - 1$.
- Another special case of the zero-padding setting is when just enough zero-padding is added to keep the size of the output equal to the size of the input. This is called as same convolution. In this case, the network can contain as many convolutional layers as the available hardware can support, since the operation of convolution does not modify the architectural possibilities available to the next layer. However, the input pixels near the border influence fewer output pixels than the input pixels near the center. This can make the border pixels somewhat underrepresented in the model.
- Difficult to learn a single kernel that performs well at all positions in the convolutional feature map. Usually the optimal amount of zero padding lies between same and valid convolution.

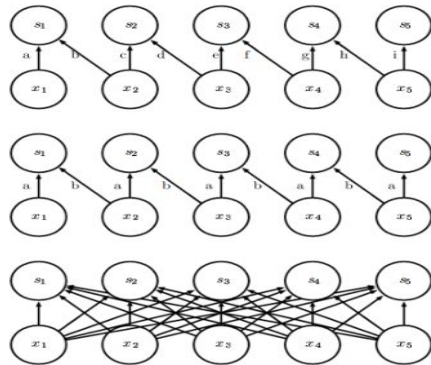
B. Unshared convolution:

- In some cases, we do not actually want to use convolution, but rather locally connected layers
 - adjacency matrix in the graph of our MLP is the same, but every connection has its own weight, specified by a 6-D tensor \mathbf{W} .
 - The indices into \mathbf{W} are respectively:
 - i , the output channel,
 - j , the output row,
 - k , the output column,
 - l , the input channel,
 - m , the row offset within the input, and
 - n , the column offset within the input.
- The linear part of a locally connected layer is then given by

$$Z_{i,j,k} = \sum_{l,m,n} [V_{l,j+m-1,k+n-1} w_{i,j,k,l,m,n}]$$

- Also called unshared convolution

Local connections, convolution, full connections



13

(Top) A locally connected layer with a patch size of two pixels. Each edge is labeled with a unique letter to show that each edge is associated with its own weight parameter.

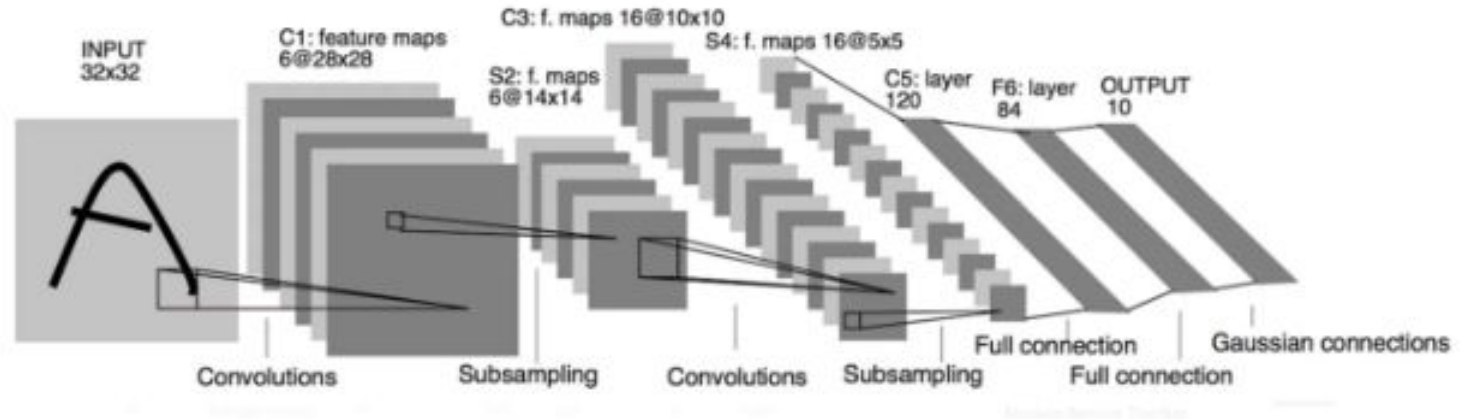
(Center) A convolutional layer with a kernel width of two pixels. This model has exactly the same connectivity as the locally connected layer. The difference lies not in which units interact with each other, but in how the parameters are shared. The locally connected layer has no parameter sharing. The convolutional layer uses the same two weights repeatedly across the entire input, as indicated by the repetition of the letters labeling each edge.

(Bottom) A fully connected layer resembles a locally connected layer in the sense that each edge has its own parameter (there are too many to label explicitly with letters in this diagram). It does not, however, have the restricted connectivity of the locally connected layer.

It can be also useful to make versions of convolution or local connected layers in which the connectivity is further restricted, eg: constrain each output channel i to be a function of only a subset of the input channel.

Modern deep learning Architectures

- **1. LeNet-5 Architecture**



- In 1989, Yann LeCun presented a convolutional neural network named LeNet. In general, LeNet refers to LeNet-5 and is a straightforward convolutional neural network.
- LeNet-5 CNN architecture is made up of 7 layers. The layer composition consists of 3 convolutional layers, 2 subsampling layers and 2 fully connected layers.

Features of LeNet-5

- Every convolutional layer includes three parts: convolution, pooling, and nonlinear activation functions
- Using convolution to extract spatial features (Convolution was called receptive fields originally)
- Subsampling average pooling layer
- tanh activation function
- Using MLP as the last classifier
- Sparse connection between layers to reduce the complexity of computation

- **Layer 1-** The first layer is the input layer; It is generally not considered a layer of the network as nothing is learned on that layer. The input layer supports 32x32, and these are the dimensions of the images that will be passed to the next layer.
- Those familiar with the MNIST dataset will know that the images in the MNIST dataset are 28 x 28 in dimensions. In order for the dimension of MNIST images to meet the requirements of the input layer, the 28x28.
- **Layer 2-** Layer C1 is a convolution layer with six 5×5 convolution kernels, and the feature allocation size is 28×28 , whereby input image information can be avoided.
- **Layer 3-** Layer S2 is the under sampling / grouping layer which generates 6 function graphs of length 14x14. Each cell in every function map is attached to 2x2 neighborhoods at the corresponding function map in C1.
- **Layer 4-** C3 convolution layer encompass sixteen 5x5 convolution kernels The input of the primary six function maps C3 is every continuous subset of the 3 function maps in S2, the access of the following six function maps comes from the access of the 4 continuous subsets and the input for the following 3 function maps is crafted from the 4 discontinuous subsets. Finally, the input for the very last function diagram comes from all the S2 function diagrams.
- **Layer 5-** Layer S4 is just like S2 with a length of 2x2 and an output of sixteen 5x5 function graphics.

- **Layer 6-** Layer C5 is a convolution layer with one hundred twenty convolution cores of length 5×5 . Each cell is attached to the 5×5 neighborhoods along sixteen S4 function charts. Since the function chart length of S4 is likewise 5×5 , the output length of C5 is $1 * 1$, so S4 and C5 are absolutely linked.
- It is referred to as a convolutional layer in preference to a completely linked layer due to the fact if the input of LeNet-5 becomes large and its shape stays unchanged, then its output length is bigger than 1×1 , i.e. now no longer a completely linked layer.
- **Layer 7-** The F6 layer is connected to C5 and 84 feature charts are generated. In the grayscale images used in the research, the pixel values from 0 to 255 were normalized to values between -0.1 and 1,175. The reason for normalization is to make sure the image stack has a mean of 0 and a standard deviation of 1.

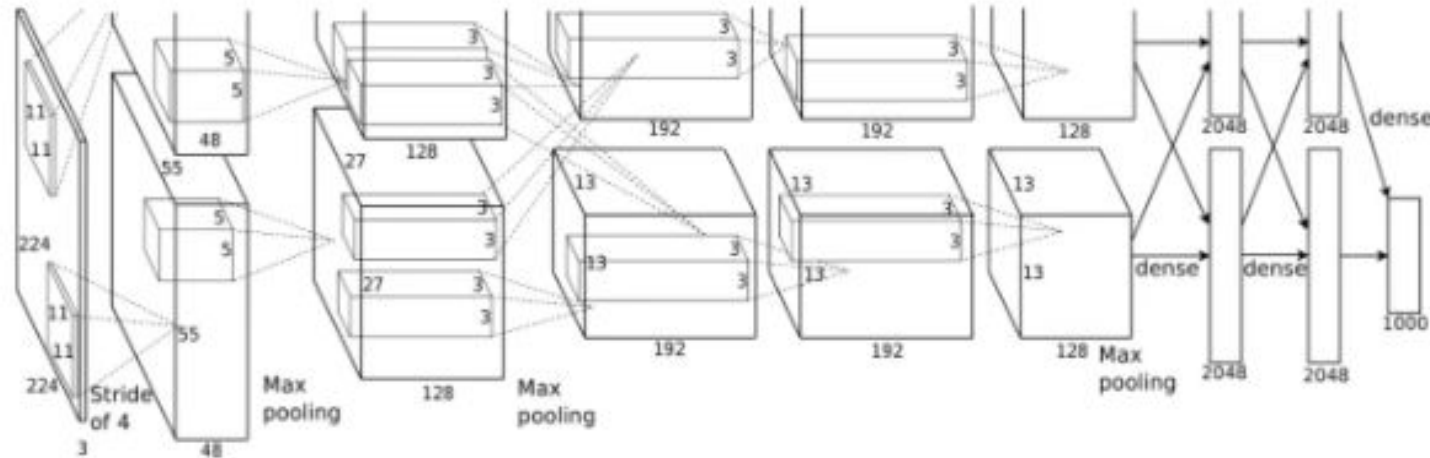
- The advantages of this are in the reduction of the training time. In the following example we will normalize the pixel values of the images to take values between 0 and 1.

-

Layer	# filters / neurons	Filter size	Stride	Size of feature map	Activation function
Input	-	-	-	32 X 32 X 1	
Conv 1	6	5 * 5	1	28 X 28 X 6	tanh
Avg. pooling 1		2 * 2	2	14 X 14 X 6	
Conv 2	16	5 * 5	1	10 X 10 X 16	tanh
Avg. pooling 2		2 * 2	2	5 X 5 X 16	
Conv 3	120	5 * 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully Connected 2	-	-	-	10	Softmax

1. The first layer is the input layer with feature map size $32 \times 32 \times 1$.
2. Then we have the first convolution layer with 6 filters of size 5×5 and stride is 1. The activation function used at this layer is tanh. The output feature map is $28 \times 28 \times 6$.
3. Next, we have an average pooling layer with filter size 2×2 and stride 1. The resulting feature map is $14 \times 14 \times 6$. Since the pooling layer doesn't affect the number of channels.
4. After this comes the second convolution layer with 16 filters of 5×5 and stride 1. Also, the activation function is tanh. Now the output size is $10 \times 10 \times 16$.
5. Again comes the other average pooling layer of 2×2 with stride 2. As a result, the size of the feature map reduced to $5 \times 5 \times 16$.
6. The final pooling layer has 120 filters of 5×5 with stride 1 and activation function tanh. Now the output size is 120.
7. The next is a fully connected layer with 84 neurons that result in the output to 84 values and the activation function used here is again tanh.
8. The last layer is the output layer with 10 neurons and Softmax function. The Softmax gives the probability that a data point belongs to a particular class. The highest value is then predicted.
9. This is the entire architecture of the Lenet-5 model. The number of trainable parameters of this architecture is around sixty thousand.

2. AlexNet:



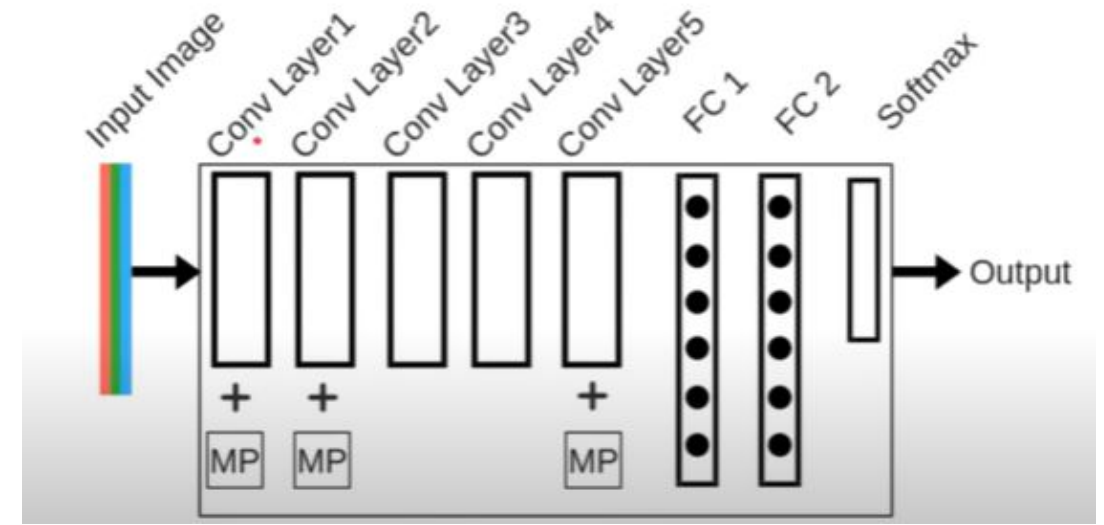
- The architecture consists of eight layers: five convolutional layers and three fully-connected layers.
- But this isn't what makes AlexNet special; these are some of the features used that are new approaches to convolutional neural networks:

- **ReLU Nonlinearity:** AlexNet uses Rectified Linear Units (ReLU) instead of the tanh function, which was standard at the time. ReLU's advantage is in training time; a CNN using ReLU was able to reach a 25% error on the CIFAR-10 dataset six times faster than a CNN using tanh.
- **Multiple GPUs:** Back in the day, GPUs were still rolling around with 3 gigabytes of memory (nowadays those kinds of memory would be rookie numbers). This was especially bad because the training set had 1.2 million images. AlexNet allows for multi-GPU training by putting half of the model's neurons on one GPU and the other half on another GPU. Not only does this mean that a bigger model can be trained, but it also cuts down on the training time.

- **Overlapping Pooling:** CNNs traditionally “pool” outputs of neighboring groups of neurons with no overlapping. However, when the authors introduced overlap, they saw a reduction in error by about 0.5% and found that models with overlapping pooling generally find it harder to overfit.
- **The Overfitting Problem:** AlexNet had 60 million parameters, a major issue in terms of overfitting. Two methods were employed to reduce overfitting.
- **Data Augmentation:** The authors used label-preserving transformation to make their data more varied. Specifically, they generated image translations and horizontal reflections, which increased the training set by a factor of 2048. They also performed Principle Component Analysis (PCA) on the RGB pixel values to change the intensities of RGB channels, which reduced the top-1 error rate by more than 1%.

- **Dropout**: This technique consists of “turning off” neurons with a predetermined probability (e.g. 50%). This means that every iteration uses a different sample of the model’s parameters, which forces each neuron to have more robust features that can be used with other random neurons. However, dropout also increases the training time needed for the model’s convergence.

- Conv= convolutional layer
- MP= max pooling(reduces dimension)
- FC= Fully connected



- Refer: *ResNet*, *ZF-Net*, *VGGNet*, *GoogLeNet* also.