**Convolutional Neural Network(CNN)**

A convolutional neural network (CNN) is a type of artificial neural network used in image recognition and processing that is specifically designed to process pixel data.

The concepts of feature extraction, pooling layers, and using convolution in a neural network were introduced and finally, recognition or classification at the end was proposed in the Neocognitron.

This algorithm is inspired by the working of a part of the human brain which is the Visual Cortex. The visual Cortex is a part of the human brain which is responsible for processing visual information from the outside world. It has various layers and each layer has its own functioning i.e each layer extracts some information from the image or any visual and at last all the information received from each layer is combined and the image/visual is interpreted or classified.

**History of CNN:**

CNN’s were first developed and used around the 1980s. The most that a CNN could do at that time was recognize handwritten digits. It was mostly used in the postal sectors to read zip codes, pin codes, etc. The important thing to remember about any deep learning model is that it requires a large amount of data to train and also requires a lot of computing resources.

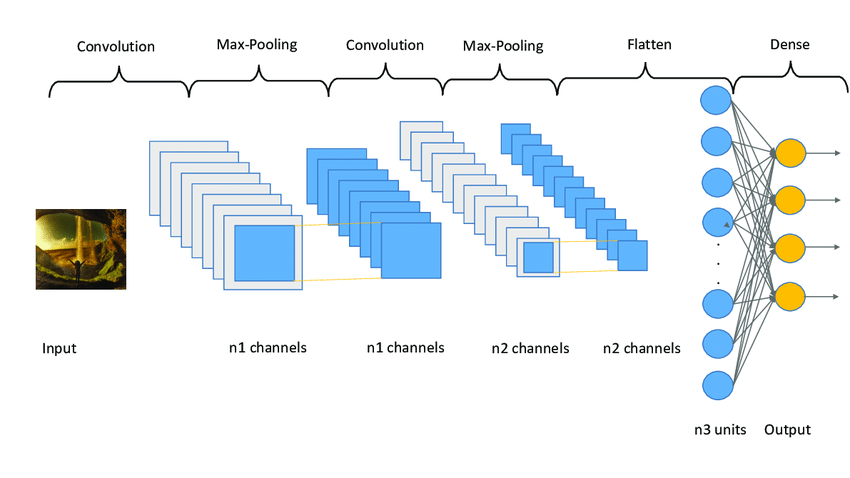
**CNN used for:**

CNNs are powerful image processing and artificial intelligence (AI) that use deep learning to perform both generative and descriptive tasks. CNN is used for Image & Video recognition, Image Analysis & Classification, Media Recreation, Recommendation Systems, Natural Language Processing, etc.

**How CNN works:**

A convolutional neural network (CNN or ConvNet), is a network architecture for deep learning which learns directly from data, eliminating the need for manual feature extraction. CNNs are powerful image processing, artificial intelligence (AI) that use deep learning to perform both generative and descriptive tasks, often using machine vision that includes image and video recognition, along with recommender systems and natural language processing (NLP). The layers of a CNN consist of an input layer, an output layer, and a hidden layer that includes multiple convolutional layers, pooling layers, fully connected layers, and normalization layers. CNN has its “neurons” arranged more like those of the frontal lobe, the area responsible for

processing visual stimuli in humans and other animals. CNNs are particularly useful for finding patterns in images to recognize objects, faces, and scenes. They can also be quite effective for classifying non-image data such as audio, time series, and signal data.



CNN work diagram

**Convolutional Neural Network (CNN) Architecture :**

Consider this – you are asked to identify objects in two given images. How would you go about doing that? Typically, you would observe the image, and try to identify different features, shapes, and edges from the image. Based on the information you gather, you would say that the object is a dog or a car and so on.

This is precisely what the hidden layers in a CNN do – find features in the image. The convolutional neural network can be broken down into two parts:

● The convolution layers: Extracts features from the input

● The fully connected (dense) layers: Uses data from the convolution layer to generate output



there are two important processes involved in the training of any neural network:

**1.** **Forward Propagation**: Receive input data, process the information, and generate output

**2.** **Backward Propagation**: Calculate error and update the parameters of the network

**How do CNNs work?**

* Technically, deep learning CNN models to train and test.
* Each input image will pass through a series of convolution layers with filters (Kernals),
* Pooling, fully connected layers (FC)
* And apply the Softmax function to classify an object with probabilistic values between 0 and 1
* They are composed of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.
* Convolutional neural networks provide a more scalable approach to image classification and object recognition tasks, leveraging principles from linear algebra, specifically matrix multiplication, to identify patterns within an image. That said, they can be computationally demanding, requiring graphical processing units (GPUs) to train models.

**CNN to process an input image and classifies the objects based on values.**

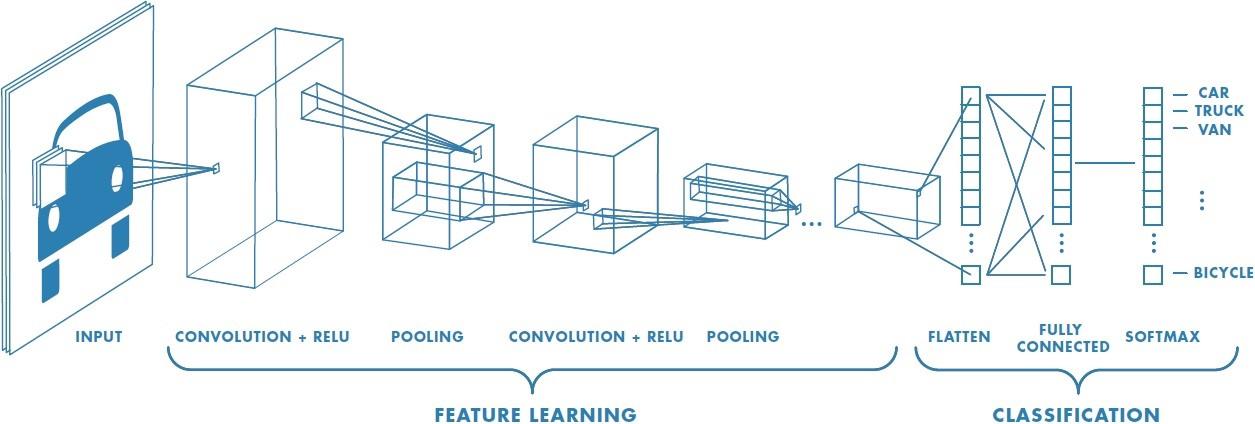


Fig: Neural network with many convolutional layers

**Working Simulation of A CNN:**

CNN image classifications take an input image, process it and classify it under certain categories

(Eg., Dog, Cat, Tiger, Lion). Computers see an input image as an array of pixels and it depends

on the image resolution. Based on the image resolution, it will see

h x w x d( h = Height, w = Width, d = Dimension ). Eg., An image of a 6 x 6 x 3 array of matrix of

RGB (3 refers to RGB values) and an image of 4 x 4 x 1 array of matrix of grayscale image.

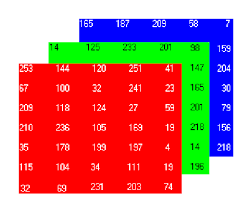
****

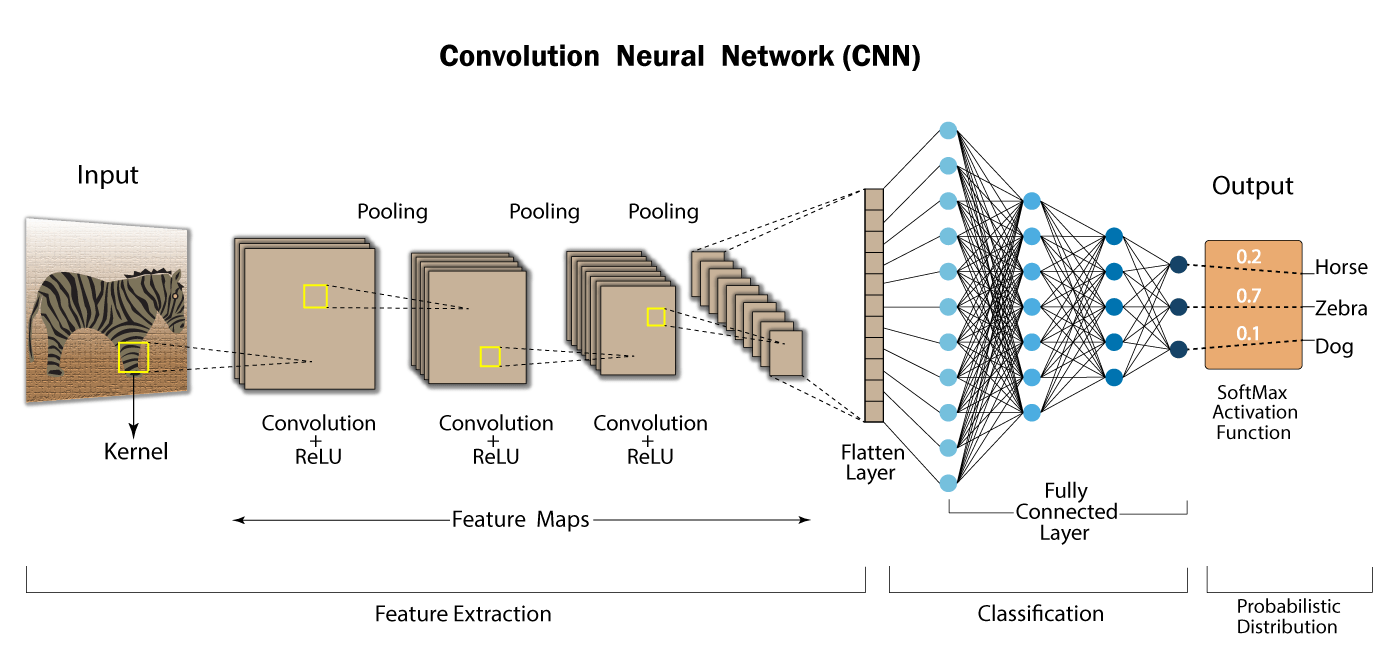
Figure: Array of RGB Matrix

Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernals), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.

**Architecture of CNN :**

A typical CNN has 4 layers

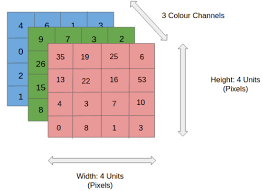
* Input layer
* Convolution layer
* Pooling layer
* Fully connected layer
* Output layer

****

**Input layer:**

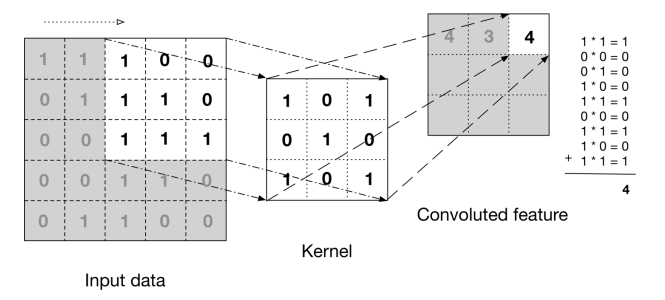
It's our input image and can be Grayscale or RGB. Every image is made up of pixels that range from 0 to 255. We need to normalize them i.e convert the range between 0 to 1 before passing it to the model. Below is the example of an input image of size 4\*4 and has 3 channels i.e RGB

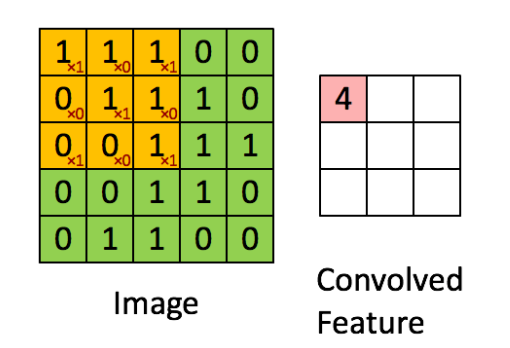
and pixel values

****

**Convolution Layer:**

Convolution is the first layer to extract features from an input image. Convolution Preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.

****

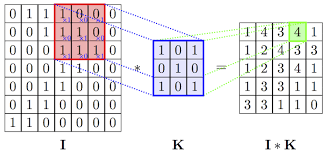
****

The above image shows what a convolution is. We take a filter/kernel(3×3 matrix) and apply it to the input image to get the convolved feature. This convolved feature is passed on to the next layer.

**Basic Convolution Operation:**

**Step 1:** Overlay the filter to the input, perform element wise multiplication, and add the result.

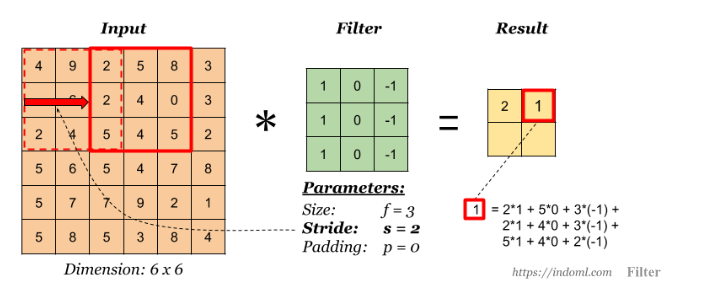
**Step 2**: move the overlay right one position (or according to the stride setting), and do the same calculation above to get the next result. And so on.



The total number of multiplications to calculate the result above is (4 x 4) x (3 x 3) = 144.

**Stride:**

Stride governs how many cells the filter is moved in the input to calculate the next cell in theresult.



with stride (s) = 2

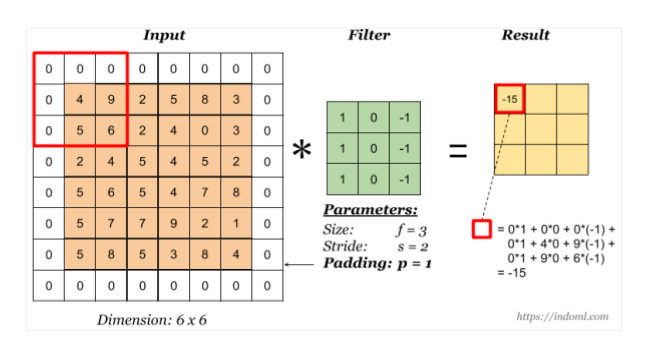
The total number of multiplications to calculate the result above is (2 x 2) x (3 x 3) = 36.

**Padding**:

Padding has the following benefits:

1. It allows us to use a CONV layer without necessarily shrinking the height and width of the volumes. This is important for building deeper networks, since otherwise the height/width would shrink as we go to deeper layers.

2. It helps us keep more of the information at the border of an image. Without padding, very few values at the next layer would be affected by pixels as the edges of an image.



Notice that the dimension of the result has changed due to padding. See the following section on how to calculate output dimension.

Some padding terminologies:

● “**valid**” padding: no padding

● “**same**” padding: padding so that the output dimension is the same as the input

Input: n X n

Padding: p

Filter size: f X f

Output: (n+2p-f+1) X (n+2p-f+1)

There are two common choices for padding:

**1.** Valid: It means no padding. If we are using valid padding, the output will be (n-f+1) X (n- f+1)

**2**. Same: Here, we apply padding so that the output size is the same as the input size, i.e.,

n+2p-f+1 = n

So, p = (f-1)/2

**Calculation Of CNN**

| | **1** | **7** | **9** | **4** | **2** | | --- | --- | --- | --- | --- | | **2** | **4** | **5** | **4** | **5** | | **5** | **6** | **2** | **4** | **0** | | **5** | **8** | **5** | **3** | **8** | | **3** | **3** | **2** | **1** | **4** |   **Image matrix** | **\*** | | **1** | **1** | **0** | | --- | --- | --- | | **1** | **0** | **1** | | **0** | **1** | **0** |   **Filter** | **=**  **Result** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Now we will calculate all value.

Here we have to do the matrix multiplication .

**Step 1 :**

| | **1** | **7** | **9** | **4** | **2** | | --- | --- | --- | --- | --- | | **2** | **4** | **5** | **4** | **5** | | **5** | **6** | **2** | **4** | **0** | | **5** | **8** | **5** | **3** | **8** | | **3** | **3** | **2** | **1** | **4** | | **\*** | | **1** | **1** | **0** | | --- | --- | --- | | **1** | **0** | **1** | | **0** | **1** | **0** | | **=** | | **21** |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

**matrix multiplication**

**Parameters:**

**Size, f = 3**

**Stride, s = 0**

**Padding , p =0**

**1\*1 + 7\*1 + 9\*0 + 2\*1 + 4\*0 + 5\*1 + 5\*0 + 6\*1 +2\*0**

**=1+7+2+5+6**

**= 21**

**Step 2 :**

| | **1** | **7** | **9** | **4** | **2** | | --- | --- | --- | --- | --- | | **2** | **4** | **5** | **4** | **5** | | **5** | **6** | **2** | **4** | **0** | | **5** | **8** | **5** | **3** | **8** | | **3** | **3** | **2** | **1** | **4** | | **\*** | | **1** | **1** | **0** | | --- | --- | --- | | **1** | **0** | **1** | | **0** | **1** | **0** | | **=** | | **21** | **26** |  | | --- | --- | --- | |  |  |  | |  |  |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

**matrix multiplication**

**Parameters:**

**Size, f = 3**

**Stride, s = 1**

**Padding , p =0**

**7\*1 + 9\*1 + 4\*0 + 4\*1 + 5\*0 + 4\*1 + 6\*0 + 2\*1 +4\*0**

**=7+9+4+4+2**

**=26**

**Step 3 :**

| | **1** | **7** | **9** | **4** | **2** | | --- | --- | --- | --- | --- | | **2** | **4** | **5** | **4** | **5** | | **5** | **6** | **2** | **4** | **0** | | **5** | **8** | **5** | **3** | **8** | | **3** | **3** | **2** | **1** | **4** | | **\*** | | **1** | **1** | **0** | | --- | --- | --- | | **1** | **0** | **1** | | **0** | **1** | **0** | | **=** | | **21** | **26** | **27** | | --- | --- | --- | |  |  |  | |  |  |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

**matrix multiplication**

**Parameters:**

**Size, f = 3**

**Stride, s = 2**

**Padding , p =0**

**9\*1 + 4\*1 + 2\*0 + 5\*1 + 4\*0 + 5\*1 + 2\*0 + 4\*1 +0\*0**

**=9+4+5+5+4**

**=27**

**Step 4 :**

| | **1** | **7** | **9** | **4** | **2** | | --- | --- | --- | --- | --- | | **2** | **4** | **5** | **4** | **5** | | **5** | **6** | **2** | **4** | **0** | | **5** | **8** | **5** | **3** | **8** | | **3** | **3** | **2** | **1** | **4** | | **\*** | | **1** | **1** | **0** | | --- | --- | --- | | **1** | **0** | **1** | | **0** | **1** | **0** | | **=** | | **21** | **26** | **27** | | --- | --- | --- | | **21** |  |  | |  |  |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

**matrix multiplication**

**Parameters:**

**Size, f = 3**

**Stride, s = 1**

**Padding , p =0**

**2\*1 + 4\*1 + 5\*0 + 5\*1 + 6\*0 + 2\*1 + 5\*0 + 8\*1 +5\*0**

**=2+4+5+2+8**

**=21**

**Step 5 :**

| | **1** | **7** | **9** | **4** | **2** | | --- | --- | --- | --- | --- | | **2** | **4** | **5** | **4** | **5** | | **5** | **6** | **2** | **4** | **0** | | **5** | **8** | **5** | **3** | **8** | | **3** | **3** | **2** | **1** | **4** | | **\*** | | **1** | **1** | **0** | | --- | --- | --- | | **1** | **0** | **1** | | **0** | **1** | **0** | | **=** | | **21** | **26** | **27** | | --- | --- | --- | | **21** | **24** |  | |  |  |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

**matrix multiplication**

**Parameters:**

**Size, f = 3**

**Stride, s = 2**

**Padding , p =0**

**4\*1 + 5\*1 + 4\*0 + 6\*1 +**

**2\*0 + 4\*1 + 8\*0 + 5\*1**

**+3\*0**

**=4+5+6+4+5**

**=24**

**Step 6 :**

| | **1** | **7** | **9** | **4** | **2** | | --- | --- | --- | --- | --- | | **2** | **4** | **5** | **4** | **5** | | **5** | **6** | **2** | **4** | **0** | | **5** | **8** | **5** | **3** | **8** | | **3** | **3** | **2** | **1** | **4** | | **\*** | | **1** | **1** | **0** | | --- | --- | --- | | **1** | **0** | **1** | | **0** | **1** | **0** | | **=** | | **21** | **26** | **27** | | --- | --- | --- | | **21** | **24** | **14** | |  |  |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

**matrix multiplication**

**Parameters:**

**Size, f = 3**

**Stride, s = 3**

**Padding , p =0**

**4\*1 + 5\*1 + 4\*0 + 2\*1 + 4\*0 + 0\*1 + 5\*0 + 3\*1 +8\*0**

**=4+5+2+3**

**= 14**

**Step 7 :**

| | **1** | **7** | **9** | **4** | **2** | | --- | --- | --- | --- | --- | | **2** | **4** | **5** | **4** | **5** | | **5** | **6** | **2** | **4** | **0** | | **5** | **8** | **5** | **3** | **8** | | **3** | **3** | **2** | **1** | **4** | | **\*** | | **1** | **1** | **0** | | --- | --- | --- | | **1** | **0** | **1** | | **0** | **1** | **0** | | **=** | | **21** | **26** | **27** | | --- | --- | --- | | **21** | **24** | **14** | | **24** |  |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

**matrix multiplication**

**Parameters:**

**Size, f = 3**

**Stride, s = 2**

**Padding , p =0**

**5\*1 + 6\*1 + 2\*0 + 5\*1 + 8\*0 + 5\*1 + 3\*0 + 3\*1 +2\*0**

**=5+6+5+5+3**

**=24**

**Step 8 :**

| | **1** | **7** | **9** | **4** | **2** | | --- | --- | --- | --- | --- | | **2** | **4** | **5** | **4** | **5** | | **5** | **6** | **2** | **4** | **0** | | **5** | **8** | **5** | **3** | **8** | | **3** | **3** | **2** | **1** | **4** | | **\*** | | **1** | **1** | **0** | | --- | --- | --- | | **1** | **0** | **1** | | **0** | **1** | **0** | | **=** | | **21** | **26** | **27** | | --- | --- | --- | | **21** | **24** | **14** | | **24** | **21** |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

**matrix multiplication**

**Parameters:**

**Size, f = 3**

**Stride, s = 3**

**Padding , p =0**

**6\*1 + 2\*1 + 4\*0 + 8\*1 + 5\*0 + 3\*1 + 3\*0 + 2\*1 +1\*0**

**=6+2+8+3+2**

**=21**

**Step 9 :**

| | **1** | **7** | **9** | **4** | **2** | | --- | --- | --- | --- | --- | | **2** | **4** | **5** | **4** | **5** | | **5** | **6** | **2** | **4** | **0** | | **5** | **8** | **5** | **3** | **8** | | **3** | **3** | **2** | **1** | **4** | | **\*** | | **1** | **1** | **0** | | --- | --- | --- | | **1** | **0** | **1** | | **0** | **1** | **0** | | **=** | | **21** | **26** | **27** | | --- | --- | --- | | **21** | **24** | **14** | | **24** | **21** | **20** | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

**matrix multiplication**

**Parameters:**

**Size, f = 3**

**Stride, s = 4**

**Padding , p =0**

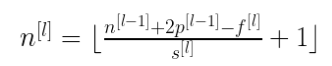
**2\*1 + 4\*1 + 0\*0 + 5\*1 + 3\*0 + 8\*1 + 2\*0 + 1\*1 +4\*0**

**=2+4+5+8+1**

**=20**

**Calculating the Output Dimension**

The output dimension is calculated with the following formula:



where the symbols denote math.floor() operation.

Generalized dimensions can be given as:

● Input: n X n X nc

● Filter: f X f X nc

● Padding: p

● Stride: s

● Output: [(n+2p-f)/s+1] X [(n+2p-f)/s+1] X nc’

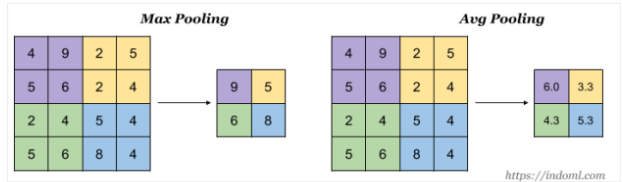
Here, nc is the number of channels in the input and filter, while nc’ is the number of filters.

**Non Linearity (ReLU)**

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is ƒ(x) = max(0,x). Why ReLU is important : ReLU’s purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values. There are other non linear functions such as tanh or sigmoid that can also be used instead of ReLU. Most of the data scientists use ReLU since performance wise ReLU is better than the other two.

**Pooling Layer**

Pooling layer is used to reduce the size of the representations and to speed up calculations, as well as to make some of the features it detects a bit more robust.Sample types of pooling are max pooling and avg pooling, but these days max pooling is more common.

****

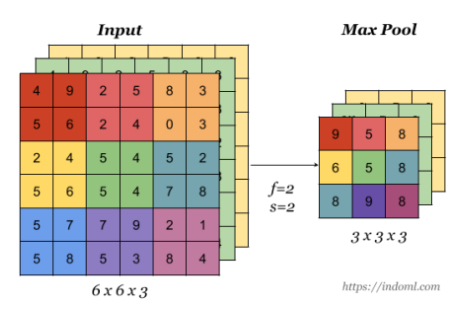
Interesting properties of pooling layer:

● it has hyper-parameters:

* ○ size (f)
* ○ stride (s)
* ○ type (max or avg)

● but it doesn’t have parameter; there’s nothing for gradient descent to learn

When done on input with multiple channels, pooling reduces the height and width (nW and nH)but keeps nC unchanged.



**Max Polling:**

Now we will do the max polling from the table. Here we take four value and choose the maximum value and repeatedly we will do it until reach the last row.

**Step 1 :**

| | **21** | **26** | **27** | | --- | --- | --- | | **21** | **24** | **14** | | **24** | **21** | **20** | | | **26** |  | | --- | --- | |  |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Here we choose the first four values from row and column, then select the max value.

Values : 21,26,21,24

Max : 26

**Step 2 :**

| | **21** | **26** | **27** | | --- | --- | --- | | **21** | **24** | **14** | | **24** | **21** | **20** | | | **26** | **27** | | --- | --- | |  |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Here we choose the second four values from row and column, then select the max value.

Values : 26,27,24,14

Max : 27

**Step 3 :**

| | **21** | **26** | **27** | | --- | --- | --- | | **21** | **24** | **14** | | **24** | **21** | **20** | | | **26** | **27** | | --- | --- | | **24** |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Values : 21,24,24,21

Max : 24

**Step 4 :**

| | **21** | **26** | **27** | | --- | --- | --- | | **21** | **24** | **14** | | **24** | **21** | **20** | | | **26** | **27** | | --- | --- | | **24** | **24** | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Values : 21,24,14,20

Max : 24

**Average Polling:**

Now we will do the average polling. To do that first you have to take first four value then calculate the average and do same task by step one.

**Step 1 :**

| | **21** | **26** | **27** | | --- | --- | --- | | **21** | **24** | **14** | | **24** | **21** | **20** | | | **23** |  | | --- | --- | |  |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Here values, 21,25,21,24

Average = (21+25+21+24) / 4 = 23

**Step 2 :**

| | **21** | **26** | **27** | | --- | --- | --- | | **21** | **24** | **14** | | **24** | **21** | **20** | | | **23** | **22.5** | | --- | --- | |  |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Here values, 26,27,24,14

Average = (26+27+24+14) / 4 = 22.5

**Step 3 :**

| | **21** | **26** | **27** | | --- | --- | --- | | **21** | **24** | **14** | | **24** | **21** | **20** | | | **23** | **22.5** | | --- | --- | | **22.5** |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Here values, 21,24,24,21

Average = ( 21+24+24+21) / 4 = 22.5

**Step 4 :**

| | **21** | **26** | **27** | | --- | --- | --- | | **21** | **24** | **14** | | **24** | **21** | **20** | | | **23** | **22.5** | | --- | --- | | **22.5** | **19.5** | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Here values, 24,14,21,20

Average = (24+14+21+20) / 4 = 19.5

**Summary**

POOL layers Accept an input volume of size Winput×Hinput×Dinput. They then require two parameters:

● The receptive field size F (also called the “pool size”).

● The stride S.

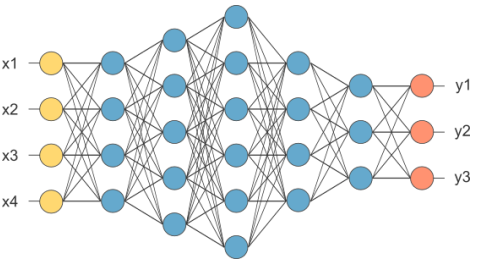
Applying the POOL operation yields an output volume of size Woutput×Houtput×Doutput, where:

* ● Woutput = ((Winput −F) / S) +1
* ● Houtput = ((Hinput −F) / S) +1
* ● Doutput = Dinput

**Fully connected Layers**

Till now we have performed the Feature Extraction steps, now comes the Classification part. The Fully connected layer (as we have in ANN) is used for classifying the input image into a label. This layer connects the information extracted from the previous steps (i.e Convolution layer and

Pooling layers) to the output layer and eventually classifies the input into the desired label. The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like a neural network.



**Figure : After pooling layer, flattened as FC layer**

In the above diagram, the feature map matrix will be converted as a vector (x1, x2, x3, ...). With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as softmax or sigmoid to classify the outputs as cat, dog, car, truck etc.

**Output Layer**

The last layer of CNN is output layer. It is responsible for producing the output probability of each given input class. A softmax unit is used to obtain the output probability. Softmax is commonly used because of it generate a well-performed probability distribution. The probability of eachoutput classes sum up to 1. The class containing the largest value will be the correct class.

**Why we use CNN instead of ANN:**

This is because there are some disadvantages with ANN:

* It is too much computation for an ANN model to train large-size images and different types of image channels.
* The next disadvantage is that it is unable to capture all the information from an image whereas a CNN model can capture the spatial dependencies of the image
* Another reason is that **ANN** is sensitive to the location of the object in the image i.e if the location or place of the same object changes, it will not be able to classify properly.

**CNN Real-life examples/Applications:**

**Face detection:** CNNs have been used to detect faces within images. The network takes an image as the input and produces a set of values that represent characteristics of faces or facial features at different parts of the image.

**Facial emotion recognition**: CNNs have been used to help distinguish between different facial expressions such as anger, sadness, or happiness.

**Object detection:** CNN has been applied to object recognition across images by classifying objects based on shapes and patterns found within an image. CNN models have been created that can detect a wide range of objects from everyday items such as food, celebrities, or animals to more unusual ones including dollar bills and guns.

**Self-driving or autonomous cars:** CNN has been used within the context of automated vehicles to enable them to detect obstacles or interpret street signs. CNN's have been used in conjunction with reinforcement learning, a branch of machine learning that focuses on positive and negative feedback from the environment to improve how CNN models respond when

encountering certain situations.

**Auto translation:** CNN is being used within the context of deep learning for automated translation between language pairs such as English and Bangla.

**Next word prediction in the sentence:** CNNs have been used to predict the next word of a sentence given some context. CNN models can process multiple sentences and learn which words typically follow others, such as “I am from Bangladesh” followed by “I speak Bengali.

**Handwritten character recognition:** CNNs can be used to recognize handwritten characters. CNNs take the image of a character as an input and break it down into smaller sections, identifying points that can connect or overlap with other points in order to determine the shape of the larger character.

**X-ray image analysis:** CNNs have been used for medical imaging to identify tumors or other abnormalities in X-ray images. CNN models can take an image of a human body part, such as the knee, and determine where within that image there might be a tumor based on previous similar images processed by CNN networks.

**Cancer detection:** CNNs have been used to detect cancer in medical images such as mammograms and CT scans. CNN models take the image of a patient and compare it against database images that contain similar characteristics, identifying when there are signs present within an image that indicate malignancy or damage to cells due both to genetics and environmental factors such as smoking habits. CNN models are able to produce highly accurate results and CNNs have even been able to identify cancerous cells with an accuracy of 95% compared to the 85-90% rate that pathologists were capable of identifying.

**Advantages of CNN:**

1. CNN is the most popular one as it attains the benefits of providing maximum performance and efficiency.

2. It can be used in various fields and perform major tasks like facial recognition, analyzing documents, understanding climate, image recognition, object identification, etc.

3. It can take a whole image as input data.

4. It is perfect when orthology matters or if you are working with characters such as emojis or bytes.

5. CNNs are very good feature extractors. This means that you can extract useful attributes from an already trained CNN with its trained weights by feeding your data on each level and tuning the CNN a bit for the specific task. Eg: Add a classifier after the last layer with labels specific to the task

6. Great for short texts(e.g. headlines)

7. It has the ability of features-learning, with which it can learn and detect related features of an object from different images.

8. It is more useful for image detection than all the other neural networks.

9 . Convolutional Neural networks (CNN) have great performance while classifying images that are very similar to the dataset.

**Disadvantages of CNN:**

1. If the images contain some degree of tilt or rotation then CNN's usually have difficulty in classifying the image.

2. If the CNN takes an image along with some noise it recognizes the image as a completely different image whereas the human visual system will identify it as the same image with the noise.

3. CNNs do not have coordinate frames which are a basic component of human vision.

4. CNN does not encode the position and orientation of the object.

5. Lack of ability to be spatially invariant to the input data.

6. A Convolutional neural network is significantly slower due to an operation such as a max pool.

7. If the CNN has several layers then the training process takes a lot of time if the computer doesn’t consist of a good GPU.

8. A ConvNet requires a large Dataset to process and train the neural network.

**Implement a CNN model using Python**

## **Small Image Classification Using Convolutional Neural Network (CNN)**

In this notebook, we will classify small images cifar10 dataset from tensorflow keras datasets. There are total 10 classes as shown below. We will use CNN for classification.

#importing the required libraries

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

import numpy as np

#load data and test different shape

(X\_train, y\_train), (X\_test,y\_test) = datasets.cifar10.load\_data()

# X\_train.shape

# X\_test.shape

# y\_train.shape

# y\_train[:5]

y\_train = y\_train.reshape(-1,)

# y\_train[:5]

y\_test = y\_test.reshape(-1,)

#define classess

classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]

# define a funciton to check image

def plot\_sample(X, y, index):

plt.figure(figsize = (15,2))

plt.imshow(X[index])

plt.xlabel(classes[y[index]])

plot\_sample(X\_train, y\_train, 0)

# output



#Normalizing the training data

X\_train = X\_train / 255.0

X\_test = X\_test / 255.0

#Build simple artificial neural network for image classification

ann = models.Sequential([

layers.Flatten(input\_shape=(32,32,3)),

layers.Dense(3000, activation='relu'),

layers.Dense(1000, activation='relu'),

layers.Dense(10, activation='softmax')

])

ann.compile(optimizer='SGD',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

ann.fit(X\_train, y\_train, epochs=5)

# output

Epoch 1/5

1563/1563 [==============================] - 123s 78ms/step - loss: 1.8148 - accuracy: 0.3522

Epoch 2/5

1563/1563 [==============================] - 147s 94ms/step - loss: 1.6277 - accuracy: 0.4265

Epoch 3/5

1563/1563 [==============================] - 127s 81ms/step - loss: 1.5431 - accuracy: 0.4552

Epoch 4/5

1563/1563 [==============================] - 124s 80ms/step - loss: 1.4846 - accuracy: 0.4771

Epoch 5/5

1563/1563 [==============================] - 114s 73ms/step - loss: 1.4344 - accuracy: 0.4933

<keras.callbacks.History at 0x7f4963caeb10>

from sklearn.metrics import confusion\_matrix , classification\_report

import numpy as np

y\_pred = ann.predict(X\_test)

y\_pred\_classes = [np.argmax(element) for element in y\_pred]

print("Classification Report: \n", classification\_report(y\_test, y\_pred\_classes))

# output

Classification Report:

precision recall f1-score support

0 0.47 0.57 0.52 1000

1 0.36 0.83 0.50 1000

2 0.40 0.27 0.32 1000

3 0.36 0.28 0.32 1000

4 0.45 0.36 0.40 1000

5 0.48 0.26 0.33 1000

6 0.55 0.49 0.52 1000

7 0.57 0.49 0.53 1000

8 0.55 0.65 0.59 1000

9 0.53 0.41 0.46 1000

accuracy 0.46 10000

macro avg 0.47 0.46 0.45 10000

weighted avg 0.47 0.46 0.45 10000

#Now let us build a convolutional neural network to train our images

cnn = models.Sequential([

layers.Conv2D(filters=32, kernel\_size=(3, 3), activation='relu', input\_shape=(32, 32, 3)),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(filters=64, kernel\_size=(3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

layers.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(10, activation='softmax')

])

cnn.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

cnn.fit(X\_train, y\_train, epochs=10)

# Output

Epoch 1/10

1563/1563 [==============================] - 67s 43ms/step - loss: 1.5303 - accuracy: 0.4517

Epoch 2/10

1563/1563 [==============================] - 68s 44ms/step - loss: 1.1667 - accuracy: 0.5909

Epoch 3/10

1563/1563 [==============================] - 72s 46ms/step - loss: 1.0326 - accuracy: 0.6398

Epoch 4/10

1563/1563 [==============================] - 75s 48ms/step - loss: 0.9589 - accuracy: 0.6674

Epoch 5/10

1563/1563 [==============================] - 73s 47ms/step - loss: 0.9016 - accuracy: 0.6880

Epoch 6/10

1563/1563 [==============================] - 73s 46ms/step - loss: 0.8535 - accuracy: 0.7044

Epoch 7/10

1563/1563 [==============================] - 72s 46ms/step - loss: 0.8162 - accuracy: 0.7152

Epoch 8/10

1563/1563 [==============================] - 67s 43ms/step - loss: 0.7738 - accuracy: 0.7319

Epoch 9/10

1563/1563 [==============================] - 67s 43ms/step - loss: 0.7385 - accuracy: 0.7445

Epoch 10/10

1563/1563 [==============================] - 67s 43ms/step - loss: 0.7106 - accuracy: 0.7525

<keras.callbacks.History at 0x7f4962397d10>

cnn.evaluate(X\_test,y\_test)

# Output

313/313 [==============================] - 4s 13ms/step - loss: 0.9854 - accuracy: 0.6750

[0.9853730201721191, 0.675000011920929]

y\_pred = cnn.predict(X\_test)

y\_pred[:5]

# output

array([[2.44965940e-03, 4.21860983e-04, 1.40938850e-03, 9.56237853e-01,

2.11278475e-05, 1.49789462e-02, 6.21186942e-03, 1.11014515e-06,

1.80649720e-02, 2.03139876e-04],

[1.39245570e-01, 1.65704787e-01, 4.66714955e-05, 1.23109442e-06,

6.90720408e-05, 1.84950029e-08, 2.36594868e-08, 1.65938175e-07,

6.92173839e-01, 2.75855977e-03],

[7.14910850e-02, 6.62503958e-01, 2.13063019e-03, 2.22452614e-03,

3.78195918e-03, 2.91555247e-04, 1.74341967e-05, 1.48912808e-02,

2.29083061e-01, 1.35844387e-02],

[9.78296995e-01, 1.74016086e-03, 1.68438274e-02, 9.24154083e-05,

1.39394880e-03, 4.46578127e-07, 1.75219739e-05, 3.61497951e-05,

1.48734450e-03, 9.11393072e-05],

[4.04586626e-06, 1.72887690e-06, 1.62631981e-02, 1.02724545e-02,

5.35741709e-02, 2.03645765e-03, 9.17699039e-01, 1.50371916e-05,

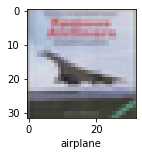
1.33445355e-04, 4.44185588e-07]], dtype=float32)

y\_classes = [np.argmax(element) for element in y\_pred]

y\_classes[:5]

Y\_test[:5]

plot\_sample(X\_test, y\_test,3)



classes[y\_classes[3]]

# output

Airplane

Colab link : <https://colab.research.google.com/drive/17i12TEkz-7UJsgw5oyUtUSBEO8-PBxqA?usp=sharing>

**Explanation of python code**

Well, in this section I am going to explain the python code on CNN. First I imported all necessary library files. Then import the dataset from keras and changed the shape of the train data. After then define a list of how many types of images are in this dataset.Then defined a function to visualize the image from dataset.Then I normalized the training data. After normalizing I defined an ANN model to compare between ANN and CNN models. After ANN I have defined a CNN model based on this dataset. Then test my model and predict new images using this CNN model.

**Resources for this paper**

**Github :** <https://github.com/codebasics/deep-learning-keras-tf-tutorial>

<https://www.analyticsvidhya.com/blog/2021/08/beginners-guide-to-convolutional-neural-network-with-implementation-in-python/?fbclid=IwAR3Mp8sye4fwV1SPl8vripo9RG9P37oO4LG_o56gY6oB_3mjlCPC8WtQDHE>