1. **Explain convolutional neural network, and how does it work?**

A **Convolutional Neural Network (CNN)** is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data.

When it comes to Machine Learning, [Artificial Neural Networks](https://www.geeksforgeeks.org/implementing-ann-training-process-in-python/) perform really well. Neural Networks are used in various datasets like images, audio, and text. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use [**Recurrent Neural Networks**](https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/) more precisely an [LSTM](https://www.geeksforgeeks.org/understanding-of-lstm-networks/), similarly for image classification we use Convolution Neural networks. In this blog, we are going to build a basic building block for CNN.

In a regular Neural Network there are three types of layers:

1. **Input Layers:** It’s the layer in which we give input to our model. The number of neurons in this layer is equal to the total number of features in our data (number of pixels in the case of an image).
2. **Hidden Layer:** The input from the Input layer is then fed into the hidden layer. There can be many hidden layers depending on our model and data size. Each hidden layer can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of the output of the previous layer with learnable weights of that layer and then by the addition of learnable biases followed by activation function which makes the network nonlinear.
3. **Output Layer:** The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into the probability score of each class.

The data is fed into the model and output from each layer is obtained from the above step is called **[feedforward](https://www.geeksforgeeks.org/understanding-multi-layer-feed-forward-networks/)**, we then calculate the error using an error function, some common error functions are cross-entropy, square loss error, etc. The error function measures how well the network is performing. After that, we backpropagate into the model by calculating the derivatives. This step is called **[Backpropagation](https://www.geeksforgeeks.org/backpropagation-in-data-mining/)** which basically is used to minimize the loss.

**Convolution Neural Network**

Convolutional Neural Network (CNN) is the extended version of [artificial neural networks (ANN)](https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/) which is predominantly used to extract the feature from the grid-like matrix dataset. For example visual datasets like images or videos where data patterns play an extensive role.

**CNN architecture**

Convolutional Neural Network consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers.

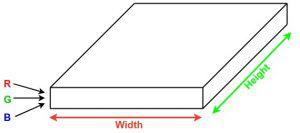


*Simple CNN architecture*

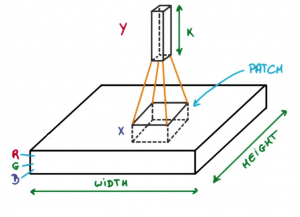
The Convolutional layer applies filters to the input image to extract features, the Pooling layer downsamples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through backpropagation and gradient descent.

**How Convolutional Layers works**

Convolution Neural Networks or covnets are neural networks that share their parameters. Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image), and height (i.e the channel as images generally have red, green, and blue channels).



Now imagine taking a small patch of this image and running a small neural network, called a filter or kernel on it, with say, K outputs and representing them vertically. Now slide that neural network across the whole image, as a result, we will get another image with different widths, heights, and depths. Instead of just R, G, and B channels now we have more channels but lesser width and height. This operation is called **Convolution**. If the patch size is the same as that of the image it will be a regular neural network. Because of this small patch, we have fewer weights.



*Image source: Deep Learning Udacity*

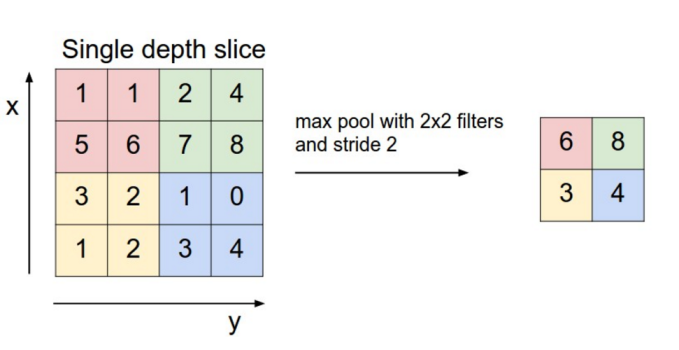
Now let’s talk about a bit of mathematics that is involved in the whole convolution process.

* Convolution layers consist of a set of learnable filters (or kernels) having small widths and heights and the same depth as that of input volume (3 if the input layer is image input).
* For example, if we have to run convolution on an image with dimensions 34x34x3. The possible size of filters can be axax3, where ‘a’ can be anything like 3, 5, or 7 but smaller as compared to the image dimension.
* During the forward pass, we slide each filter across the whole input volume step by step where each step is called [**stride**](https://www.geeksforgeeks.org/ml-introduction-to-strided-convolutions/) (which can have a value of 2, 3, or even 4 for high-dimensional images) and compute the dot product between the kernel weights and patch from input volume.
* As we slide our filters we’ll get a 2-D output for each filter and we’ll stack them together as a result, we’ll get output volume having a depth equal to the number of filters. The network will learn all the filters.

**Layers used to build ConvNets**

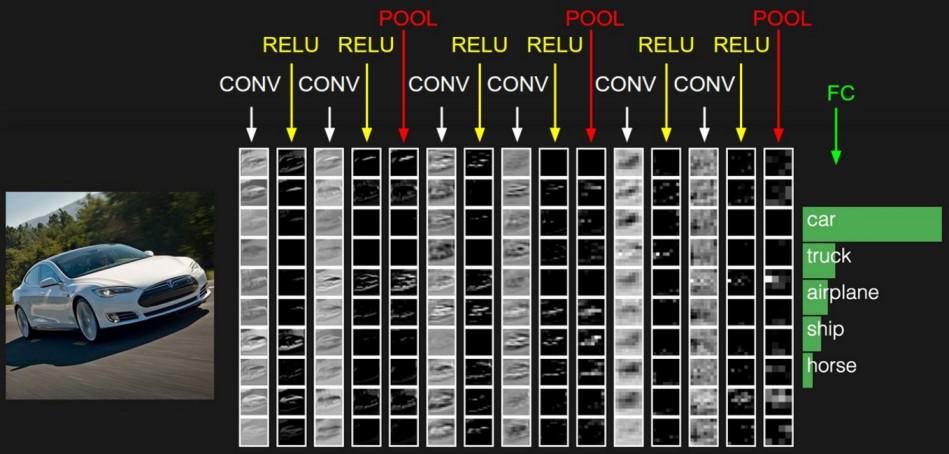
A complete Convolution Neural Networks architecture is also known as covnets. A covnets is a sequence of layers, and every layer transforms one volume to another through a differentiable function.   
**Types of layers:** datasets  
Let’s take an example by running a covnets on of image of dimension 32 x 32 x 3.

* **Input Layers:** It’s the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3.
* **Convolutional Layers:**This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The filters/kernels are smaller matrices usually 2×2, 3×3, or 5×5 shape. it slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred as feature maps. Suppose we use a total of 12 filters for this layer we’ll get an output volume of dimension 32 x 32 x 12.
* [**Activation Layer:**](https://www.geeksforgeeks.org/activation-functions-neural-networks/)By adding an activation function to the output of the preceding layer, activation layers add nonlinearity to the network. it will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are **RELU**: max(0, x),  **Tanh**, **Leaky RELU**, etc. The volume remains unchanged hence output volume will have dimensions 32 x 32 x 12.
* [**Pooling layer:**](https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/) This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting. Two common types of pooling layers are **max pooling** and **average pooling**. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12.



*Image source: cs231n.stanford.edu*

* **Flattening:**The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer for categorization or regression.
* **Fully Connected Layers:**It takes the input from the previous layer and computes the final classification or regression task.



*Image source: cs231n.stanford.edu*

* **Output Layer:** The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or softmax which converts the output of each class into the probability score of each class.

**Example:**

Let’s consider an image and apply the convolution layer, activation layer, and pooling layer operation to extract the inside feature.

**Input image:**



*Input image*

**Step:**

* import the necessary libraries
* set the parameter
* define the kernel
* Load the image and plot it.
* Reformat the image
* Apply convolution layer operation and plot the output image.
* Apply activation layer operation and plot the output image.
* Apply pooling layer operation and plot the output image.

1. **How does refactoring parts of your neural network definition favor you?**

**Refactoring is the art of improving the structural design of a software system without altering its external behavior. Today, refactoring has become a well-established and disciplined software engineering practice that has attracted a significant amount of research presuming that refactoring is primarily motivated by the need to improve system structures. However, recent studies have shown that developers may incorporate refactoring strategies in other development-related activities that go beyond improving the design especially with the emerging challenges in contemporary software engineering. Unfortunately, these studies are limited to developer interviews and a reduced set of projects. To cope with the above-mentioned limitations, we aim to better understand what motivates developers to apply a refactoring by mining and automatically classifying a large set of 111,884 commits containing refactoring activities, extracted from 800 open source Java projects. We trained a multi-class classifier to categorize these commits into three categories, namely, Internal Quality Attribute, External Quality Attribute, and Code Smell Resolution, along with the traditional Bug Fix and Functional categories. This classification challenges the original definition of refactoring, being exclusive to improving software design and fixing code smells. Furthermore, to better understand our classification results, we qualitatively analyzed commit messages to extract textual patterns that developers regularly use to describe their refactoring activities. The results of our empirical investigation show that (1) fixing code smells is not the main driver for developers to refactoring their code bases. Refactoring is solicited for a wide variety of reasons, going beyond its traditional definition; (2) the distribution of refactoring operations differs between production and test files; (3) developers use a variety of patterns to purposefully target refactoring-related activities; (4) the textual patterns, extracted from commit messages, provide better coverage for how developers document their refactorings.**

**3. What does it mean to flatten? Is it necessary to include it in the MNIST CNN? What is the reason for this?**

**In neural networks, especially in the context of processing images, flattening the data is a common preprocessing step that is done to convert a 2D or 3D array (such as an image) into a 1D array. This flattening operation is typically performed before feeding the data into a fully connected neural network layer.**

**Here are some reasons why we flatten the data in neural networks while processing images:**

1. **Compatibility with Fully Connected Layers: Fully connected layers in neural networks expect 1D input data. By flattening the 2D or 3D image data into a 1D array, we can easily connect it to the input layer of a neural network composed of fully connected layers.**
2. **Simplicity and Efficiency: Flattening the data simplifies the input representation. Instead of dealing with complex 2D or 3D structures, we convert the data into a simple vector format, which is easier to process and manipulate.**
3. **Reduction of Dimensionality: Flattening reduces the dimensionality of the data, which can help in reducing the computational complexity of the network. It also helps in reducing the number of parameters in the subsequent layers, which can prevent overfitting in some cases.**
4. **Uniform Input Size: Flattening ensures that each input sample has a consistent size, which is important for feeding the data into the neural network. This uniformity simplifies the implementation and helps in handling the data efficiently.**
5. **Historical Reasons: Initially, fully connected neural networks were the predominant architecture used for image processing tasks. Flattening the data was a natural step in this context. Even though convolutional neural networks (CNNs) are now more commonly used for image processing tasks, the practice of flattening data is still prevalent in certain architectures or when using fully connected layers in combination with CNNs.**

**While flattening the data is a common practice, it is important to note that with the rise of convolutional neural networks (CNNs), which are specifically designed to work with image data without the need for flattening, this step may not always be necessary. CNNs preserve the spatial structure of the data through convolutional and pooling layers, allowing them to learn features directly from the 2D or 3D input data.**

**4. What exactly does NCHW stand for?**

**NCHW is an acronym describing the order of the axes in a tensor containing image data samples.**

* **N: Number of data samples.**
* **C: Image channels. A red-green-blue (RGB) image will have 3 channels.**
* **H: Image height.**
* **W: Image width.**

**NCHW is sometimes referred to as a channels-first layout.**

**5. Why are there 7\*7\*(1168-16) multiplications in the MNIST CNN's third layer?**

**CNN is basically a model known to be Convolutional Neural Network and in recent times it has gained a lot of popularity because of its usefulness. CNN uses multilayer perceptrons to do computational works. CNN uses relatively little pre-processing compared to other image classification algorithms. This means the network learns through filters that in traditional algorithms were hand-engineered. So, for the image processing tasks CNNs are the best-suited option.**

**Applying a Convolutional Neural Network (CNN) on the MNIST dataset is a popular way to learn about and demonstrate the capabilities of CNNs for image classification tasks. The MNIST dataset consists of 28×28 grayscale images of hand-written digits (0-9), with a training set of 60,000 examples and a test set of 10,000 examples.**

**Here is a basic approach to applying a CNN on the MNIST dataset using the Python programming language and the Keras library:**

1. **Load and preprocess the data: The MNIST dataset can be loaded using the Keras library, and the images can be normalized to have pixel values between 0 and 1.**
2. **Define the model architecture: The CNN can be constructed using the Keras Sequential API, which allows for easy building of sequential models layer-by-layer. The architecture should typically include convolutional layers, pooling layers, and fully-connected layers.**
3. **Compile the model: The model needs to be compiled with a loss function, an optimizer, and a metric for evaluation.**
4. **Train the model: The model can be trained on the training set using the Keras fit() function. It is important to monitor the training accuracy and loss to ensure the model is converging properly.**
5. **Evaluate the model: The trained model can be evaluated on the test set using the Keras evaluate() function. The evaluation metric typically used for classification tasks is accuracy.**

**Here are some tips and best practices to keep in mind when applying a CNN on the MNIST dataset:**

1. **Start with a simple architecture and gradually increase complexity if necessary.**
2. **Experiment with different activation functions, optimizers, learning rates, and batch sizes to find the optimal combination for your specific task.**
3. **Use regularization techniques such as dropout or weight decay to prevent overfitting.**
4. **Visualize the filters and feature maps learned by the model to gain insights into its inner workings.**
5. **Compare the performance of the CNN to other machine learning algorithms such as Support Vector Machines or Random Forests to get a sense of its relative performance.**

**We can get 99.06% accuracy by using CNN(Convolutional Neural Network) with a functional model. The reason for using a functional model is to maintain easiness while connecting the layers.**

**6.Explain definition of receptive field?**

**In the context of Convolutional Neural Networks (CNNs), a receptive field refers to the region of the input space that a particular convolutional neuron is looking at. In other words, it is the area of the input image that influences a particular feature map in the network.**

**Receptive fields in CNNs can be of two types: local receptive fields and global receptive fields.**

1. **Local Receptive Field: This refers to the spatial extent of the input that a single neuron in a convolutional layer is connected to. In a convolutional layer, each neuron is connected to a small region of the input image known as the local receptive field. This local receptive field is determined by the size of the convolutional kernel (also called filter) applied to the input image. The weights in the kernel are learned during the training process.**
2. **Global Receptive Field: The global receptive field of a neuron in a CNN refers to the entire input region that influences the neuron's output. As we move deeper into the network through multiple layers of convolutions and pooling operations, the receptive field of neurons increases. Neurons in deeper layers have larger receptive fields because they receive input from a larger area of the input image due to the pooling and convolution operations in the network.**

**Understanding receptive fields is important in CNNs because it helps in capturing features at different scales in the input data. Neurons with small receptive fields capture fine details, while neurons with larger receptive fields capture more global features.**

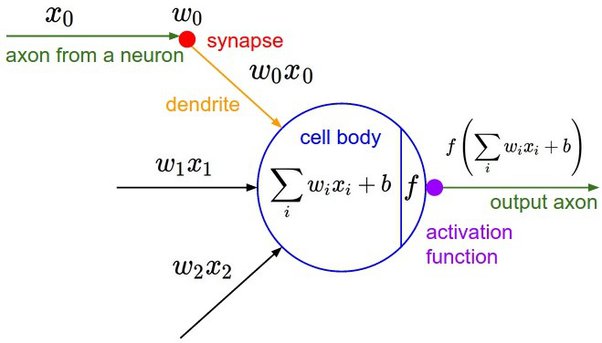
**By stacking multiple layers in a CNN, each layer can learn increasingly complex features by considering larger receptive fields, allowing the network to extract hierarchical representations of the input data.**

1. **What is the scale of an activation's receptive field after two stride-2 convolutions? What is the reason for this?**

When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume. Instead, we connect each neuron to only a local region of the input volume. The spatial extent of this connectivity is a hyperparameter called the **receptive field** of the neuron (equivalently this is the filter size). The extent of the connectivity along the depth axis is always equal to the depth of the input volume. It is important to emphasize again this asymmetry in how we treat the spatial dimensions (width and height) and the depth dimension: The connections are local in space (along width and height), but always full along the entire depth of the input volume.

*Example 1*. For example, suppose that the input volume has size [32x32x3], (e.g. an RGB CIFAR-10 image). If the receptive field (or the filter size) is 5x5, then each neuron in the Conv Layer will have weights to a [5x5x3] region in the input volume, for a total of 5\*5\*3 = 75 weights (and +1 bias parameter). Notice that the extent of the connectivity along the depth axis must be 3, since this is the depth of the input volume.

*Example 2*. Suppose an input volume had size [16x16x20]. Then using an example receptive field size of 3x3, every neuron in the Conv Layer would now have a total of 3\*3\*20 = 180 connections to the input volume. Notice that, again, the connectivity is local in space (e.g. 3x3), but full along the input depth (20).



**Left:** An example input volume in red (e.g. a 32x32x3 CIFAR-10 image), and an example volume of neurons in the first Convolutional layer. Each neuron in the convolutional layer is connected only to a local region in the input volume spatially, but to the full depth (i.e. all color channels). Note, there are multiple neurons (5 in this example) along the depth, all looking at the same region in the input - see discussion of depth columns in text below. **Right:** The neurons from the Neural Network chapter remain unchanged: They still compute a dot product of their weights with the input followed by a non-linearity, but their connectivity is now restricted to be local spatially.

**7. What is the tensor representation of a color image?**

General tensors can represent colour images more naturally than conventional features; however the general tensorsý stability properties are not reported and remain to be a key problem. In this paper, we use the tensor minimax probability (TMPM) to prove that the tensor representation is stable. The proof is based on the random subspace method through a large number of experiments.

1. **How does a color input interact with a convolution?**

**The invariance of the CNN to an artifact is derived from the data.**

**The CNN only has the data to learn if color is a decisive factor for recognizing an object or not. If you only present it with red 'A's, it will learn that red is a decisive factor for recognizing the 'A'. By presenting it with a large number of different 'A's that are colored differently. The CNN will learn that color has little influence in recognizing an 'A'. The weight of the red channels or red features will not be dominant. You might even find that the CNN will learn grayscale filters instead of color sensitive filters.**

**The color distribution for some objects, especially natural objects like those in ImageNet is not uniform. This results in the CNN learning color sensitive filters. After training the filters will be weighted according to the distribution with which it can recognize the object with least amount of error.**

**For instances of objects that may be appear in different colors, where these colors are arbitrary (e.g. letters or digits on a sign/poster) we need to present sufficient examples for the CNN to untangle color information from recognizing those letters and digits. If it happens to only recognize red 'A's, it's because we never showed it otherwise.**