Convolutional Neural Networks

In computer vision applications, Convolutional Neural Networks (ConvNet or CCN) which is a class of Artificial Neural Networks considered a state of the art method for solving vision tasks. When solving the vision task, ANN with feed-forward architecture is not feasible because of these two problems: As each layer in ANN is connected to its corresponding layers which results in too many parameters in the network [4], and it requires a large amount of memory and large computation power. For instance, considering a CIFAR-10 dataset image size of 32 x 32 x 3 pixels that requires a total number of 3072 weights and this number is still manageable with ANN architecture but these fully-connected architectures would not scale to larger images. Considering an image size of 224 x 224 x 3 that sums up to 150,258 for a single fully-connected neuron for the first layer. This number of parameters would increase exponentially in the following layers of the network [6]. This large number of parameters would lead the network to overfitting. Another deficiency of the fully connected architecture, it completely ignores the spatial information of the image and results in it can't receive the information of correlation between local features of the image [6].

Convolutional Neural Network architecture is the solution for these problems which we faced in fully-connected Networks. The architecture of the CNN is identical to the human brain. The connectivity pattern of the neurons are identical to the human brain and the whole organization is inspired by the visual cortex [8]. CNN are developed to automatically and adaptively learn spatial hierarchical features, from low to high features [1]. CNN architecture is composed of different types of layers known as convolution layer, pooling layer, activation functions and the fully connected layer. In the following sections, these blocks of CNN will be briefly discussed.

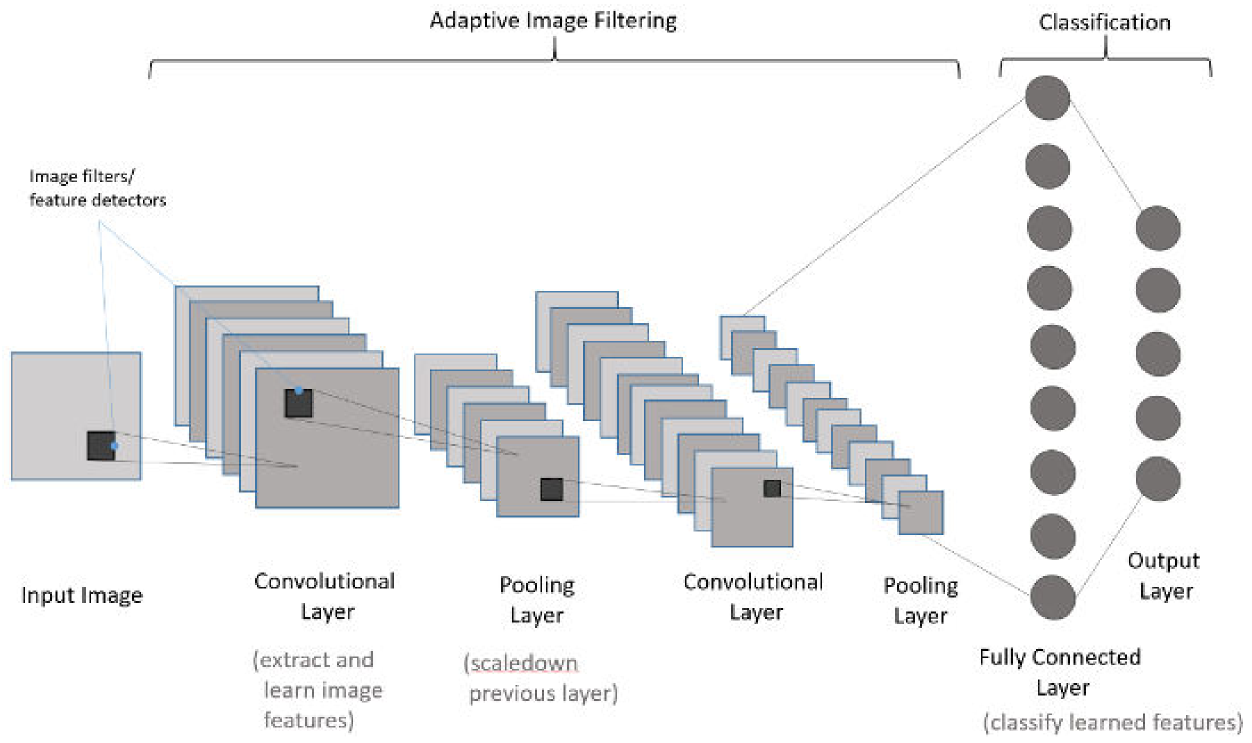

Figure 1: A basic ConvNet with two convolutional layers followed by pooling layer for Feature extraction and two FC layers for classification.

Convolutional Layer

Convolutional layer or Conv layer usually is the first and core building block of ConvNets that extract feature from an image and outputs feature maps by convolution operation. Conv layer consists a set of filters/kernels/weights and each filter convolves over the entire input image and computes dot products between the filter and input of the image at that position. Figure 2.4 shows an example of convolution operation. Typically, the kernel sizes are of (3 × 3),(5 × 5),(7 × 7) and number of kernels can be any number which refers to the depth of feature map. To preserve the input image size padding (usually 0) is used which known as *same* mode. Number of padding to be used depends on the kernel size. If no padding is used the output reduces the resolution known as *valid* mode.

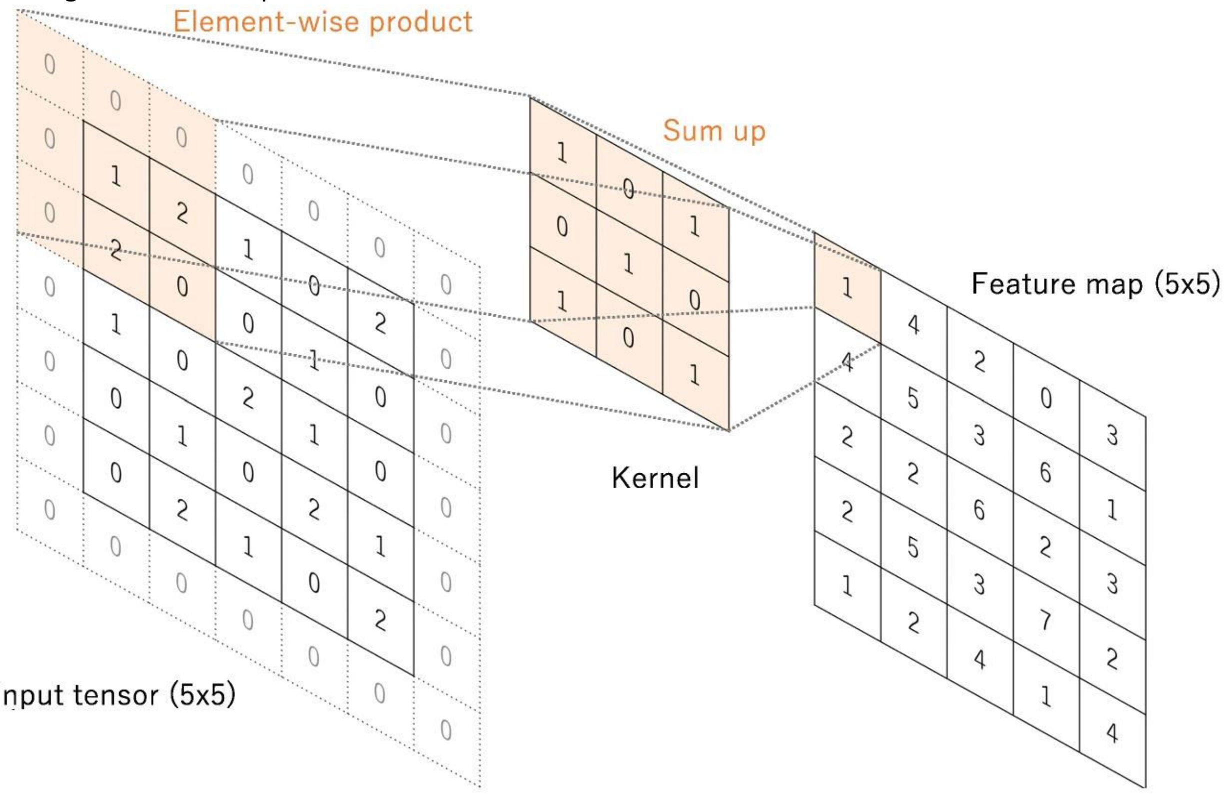

Figure 2: Convolution operation in the image uses (3 x 3) kernel size with stride=1, padding =1, obtains the same feature map size as of the input image (5 x 5). The above image feature maps results, 1 = (0 x 1) + (0 x 0) + (0 x 1) + (0 x 0) + (1 x 1) + (2 x 0) + (0 x 1) + (2 x 0) + (0 x 1) [1].

The distance between consecutive kernel positions is known as stride. The kernel slides one pixel after each operation by default (stride = 1), but this can also be controlled by changing the stride. Stride 2 means kernel would shift 2 pixels after each operation. The convolutional output size of an ($n \times n$) image with filter size ($f \times f$), padding p and strides can be calculated as ((n+2p−fs +1)×(n+2p−fs +1)). In case of color images, the input size becomes ($n \times n \times nc$) and filter size ($f \times f \times nc$), where nc is the number of channels. The process described above known as 2D convolution. The depth of the feature map is equal to the number of filters used. In 3D convolution feature map produced by a single filter is also 3D. The output after convolution operation are passed through a non-linear activation function (more on section 2.2.3) which gives a nonlinear representation to learn complex functions that maps the input data to the classification output. The kernels/filters/weights associated with a ConvNet are adjusted while training to extract features from the input image.

Pooling Layer:

After a Conv layer a pooling layer is applied that provides typical down sampling to reduce the dimensionality of the feature maps and while keeping the spatial information. Because of the down sampling the number learnable parameters also decrease for the subsequent Conv layers and introduces a translation invariance to small shifts and distortions. As this is only a mathematical operational layer there is no learnable parameters involved, although filter size, padding and strides works in a similar way that of a Conv layer. Max pooling is the most used pooling operation which extract the maximum value out of each patch from the input feature map. Figure 3 shows a max pooling operation with filter size (2×2) with stride 2 so that patches does not overlap. Other types of pooling include average pooling, min pooling, etc. which has their specific application.

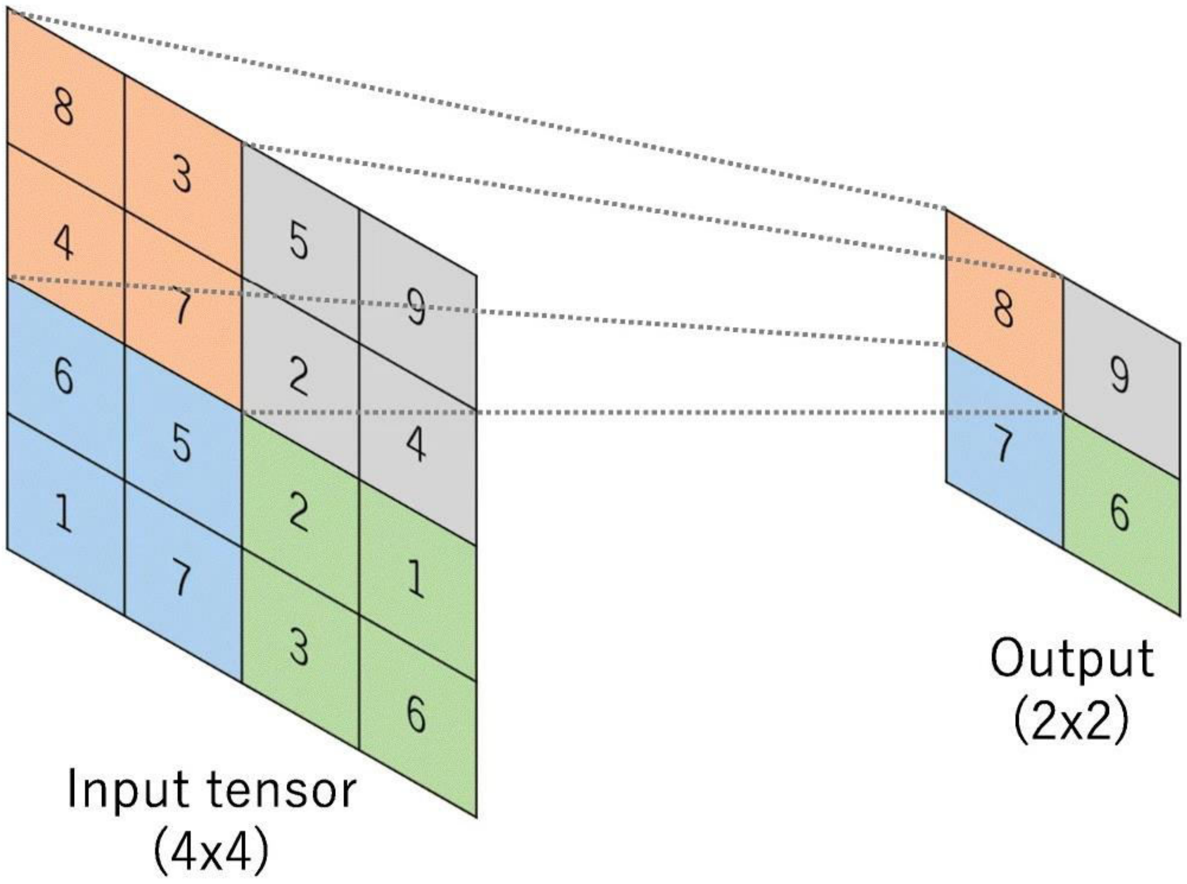

Figure: 3 Max pooling operation on (4 x 4) image produces the feature map size of (2 x 2) with stride=2, and padding = 0 [1].

Activation Functions:

As mentioned before, the output of each convolutional operation passes through an activation function. This is true for fully-connected layers as well. Activation function maps a linear input to a non-linear output which helps the network to learn more complex data classification. Choosing the right activation function is very crucial for a network to learn during backpropagation which tries to minimize the loss function by adjusting learnable parameters using gradient descent. There are several activation functions that can be used such as sigmoid, tanh, ReLU, Leaky ReLU, etc.

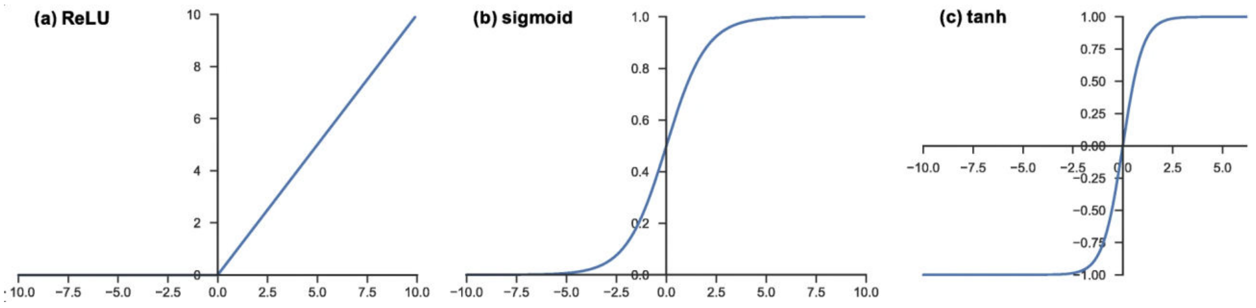

Figure 4: Commonly applied activation functions to neural networks. (a) ReLU, (b) Sigmoid and (c) tanh [1].

Sigmoid activation function maps all input between 0 and 1 that means data is squeezed and gradient for very large positive or negative value becomes close to 0. Once the gradient becomes zero the neuron becomes useless as there is no learning. This problem is known as *vanishing gradient problem*. While tanh activation function, increase the output range between -1 to 1, it does not solve the problem. To solve the issue activation function named Rectified Linear Unit or ReLU is used which does not squeeze the input. ReLU passes all positive value of the input as it is and all negative value is mapped to 0. ReLU is proven to be the most used and best activation function in the inner layers of the network [2] [3]. Sometimes during training, bias of some neurons can become very negative and since ReLU blocks negative value these neurons are off even worse if the training input is not strong enough to overcome this negative bias then the neuron becomes dead. This is known as dying ReLU problem which can be solve using Leaky ReLU, similar to ReLU but negative numbers are not completely blocked instead mapped to a small negative value. However, this is very rare and for most of the cases ReLU would work just fine. The last layer of neural network usually consists an activation function called Softmax which is similar to sigmoid function but gives a probabilistic output for each class of the classification. Each element in the output depends on the entire set of elements of the input. Softmax activation function normalizes the outputs from the last fully-connected layer to target class between 0 and 1 and summation of all values equal to 1.

Normalization Layer:

Normalization layers are proposed to use in ConvNets to force the output of each layer into a certain range which empirically shows faster convergence. However, this is only observable in a deep network and with size of the normalization is large enough. There are many normalization techniques introduced such as batch normalization, group normalization, weight normalization, etc. to normalize the values in a different way. As the contribution has been shown to be very minimal, these normalization layers have fallen out of interest in practice.

Fully-Connected Layer:

The last layer of ConvNets is a fully-connected layer where all the neurons of the layer have connection from the previous activations. This is same as the hidden layers of Neural Networks except it has different activation function (Softmax) than the hidden layers which gives probabilistic out of all the output neurons. Before connecting to the fully-connected layers the output of last Conv layer is flattened to make it a 1D array. Sometimes ConvNets also have couple of fully-connected layers (with ReLU) similar to hidden layers of Neural Networks before the last fully-connected classification layer.

Training of the CNN:

Training a CNN means adjusting the kernels/filters of Conv layers and weights and biases of fully-connected layers to minimize the difference between predicted output and ground truth labels on the training dataset. The difference between ground truth and predicted output is calculated using a loss function. First, the network is initialized with random kernels, weights and biases of a certain range and during forward pass model performance is calculated for particular kernels and weights by a loss function on a training dataset. The learnable parameters are then updated to minimize the loss calculated during forward pass by an optimization algorithm and the process is known as backpropagation.

Loss Function:

While forward propagation the difference between the ground truth or the labels of the training dataset and the predicted output is calculated using a loss function which is also known as cost function. Main goal of training a network is to minimize these costs by updating the learnable parameters in each iteration of training (epoch). Loss function needs to be determined according to the specific classification task such as Cross Entropy is used for multiclass classification, Mean Squared Error (MSE) is used for regression to continuous values. The loss function is one of the hyperparameters that needs to be determined while optimizing the model for a given task.

References:

1. Yamashita, Rikiya ; Nishio, Mizuho ; Do, Richard Kinh G. ; Togashi, Kaori: *Convolutional neural networks: an overview and application in radiology*
2. Ramachandra, Prajit ; Zoph, Barret ; V.Le, Quoc: Searching for activation functions. In: *6th International Conference on Learning Representations, ICLR 2018 -Workshop Track Proceedings*, 2018
3. Nair, Vinod; Hinton, Geoffrey E.: Rectified linear units improve Restricted Boltzmann machines. In: *ICML 2010 - Proceedings, 27th International Conference on Machine Learning*, 2010. – ISBN 9781605589077
4. Cs231n.github.io. 2022. *CS231n Convolutional Neural Networks for Visual Recognition*. [online] Available at: <https://cs231n.github.io/convolutional-networks/> [Accessed 17 April 2022].

5. *Lecun Y, Haffner P, Bottou L, Bengio Y*. Object Recognition with Gradient-Based Learning. In: Forsyth DA (ed). Shape, contour and grouping in computer vision. Berlin [etc.]: Springer, 1999: 319 – 345

6. *C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich*. Going deeper with Convolutions. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015: 1 – 9