UNIT II

CHAPTER 2

Convolutional Neural Network

Syllabus

Introduction to CNN, Convolution Operation, Parameter Sharing, Equivariant Representation, Pooling, Variants of the Basic Convolution Function, The basic Architecture of CNN, Popular CNN Architecture – AlexNet.

2.1 CONVOLUTIONAL NETWORKS

GQ. Explain convolutional networks.

Convolutional network is a **special class of multilayer perceptrons,** designed to recognise two-dimensional shapes with a high degree of accuracy to skewing, scaling and to translation and other forms of distortion.

This task is achieved by a supervised manner which includes the following forms of constraints :

(1) Each neuron extracts local feature from the previous layer. Convolution layers apply a convolution operation to the input passing the result to the next layer. A convolution converts all the pixels in it's receptive field into a single value. The final output of the convolutional layer is a vector.

Convolutional layers are the major building blocks used in convolution neural networks.

In convolution layer, a neuron is only connected to a local area of input.

- A convolution layer contains units whose receptive fields cover a patch of receptive layer.
- The convolution layer is the core building block of a convolution network.

(2) Each computational layer is composed of feature maps in the form of a plane in which each individual neuron share the same synaptic weights. It has the following beneficial effects :

(i) Shift invariance through the use of **convolution** with a **kernel** of small size (and using sigmoid functions)

(ii) Reduction in the number of free parameters.

(3) **Subsampling** : Each convolutional layer performs local averaging and subsampling. It reduces the sensivity of the feature maps distortion.

In convolutional network, all weights in all layers are learned through training. The network learns to extract it's own features automatically.

In deep learning, a convolutional neural network (CNN) is an artificial neural network, applied to analyse visual imagery. They are also called as shift invariant or space invariant artificial neural networks (SIANN). Convolutional neural networks are equivariant and not invariant to translation.

- They have applications in image and video recognition, image segmentation, image classification, medical image analysis, image and video recognition, natural language processing brain-computer interfaces, and financial time series.

- CNNs are regularised versions of multilayer perceptronsy which are fully connected layers, i.e. each neuron in one layer is connected to all neurons in the next layer.

These days, deep learning is the threshold of many advanced technological applications, from self-driving cars to art and music. Deep learning enables the computer to build complex concepts from simpler concepts. Very soon we shall see that deep learning will be extended to applications that enable machines to thinks on their own.

2.2 CONVOLUTION OPERATION

Convolution is a **mathematical operation**. In mathematics, especially in Fourier, Laplace, Z-transforms, convolution operation paves the way to find the inverse transforms. The concept of convolution operates on two functions f and g that produces a third function (f * g), in the transformed domain.

The term convolution refers to both the result function and **the method of evaluating** it. It is defined as the integral of **product of two functions**, in which one function is shifted. The integral is evaluated for all values of shift, and gives the convolution function.

2.2.1 Cross-Correlation

Cross-correlation operation is similar to convolution operation.

Consider two functions f(x) and g(x). In cross-correlation, either f(x) or g(x) is reflected about Y-axis, i.e. it is a cross-correlation of f(x) and g(– x) or f(– x) and g(x).

In case of complex valued functions, the cross-correlation operator is the adjoint of the convolution operator.

2.2.2 Applications of Convolution

GQ. Mention applications of convolution.

(1) The applications of convolution include probability, statistics, acoustics, spectroscopy, signal processing and image processing, physics, engineering, computer vision, differential equations and various transforms. Also it has applications in the field of numerical analysis and numerical linear algebra, and in the design and implementation of finite impulse response filters in signal processing.

(2) The convolution is also defined for functions on Euclidean space and other groups. For example, periodic functions, such as the 'discrete-time Fourier transform' can be defined for a given interval and convolved by periodic convolution. Note that a 'discrete convolution' can be defined for functions on the set of integers. Finding the inverse of convolution operation is called as 'deconvolution'.

2.2.3 Convolution Formulae

(1) Let f(t) and g(t) be two continuous functions defined in $(-\infty, \infty)$, then

$$(f * g)(t) = f(t) * g(t)$$

$$= \int_{-\infty}^{\infty} f(u)\, g(t-u) \cdot du = \int_{-\infty}^{\infty} f(t-u)\, g(u) \cdot du$$

Remark

(i) The symbol 't' may not represent time domain

(ii) Convolution formula represents the area f(u) weighted by the function g(– u) shifted by the amount 't'. As 't' changes the weighting function g(t – u) gives different parts of the input function f(u).

(2) Let f(t) and g(t) be defined in $(0, \infty)$ and zero elsewhere, then convolution of f(t) and g(t) is defined as

$$(f * g)(t) = f(t) * g(t) = \int_0^t f(u)\, g(t-u) \cdot du$$

$$= \int_0^t f(t-u)\, g(u)\, du \qquad \left[\because \int_0^a f(x)\, dx = \int_0^a f(a-x)\, dx\right]$$

2.2.4 Notation

$$(f * g)(t) = \int_{-\infty}^{\infty} f(u)\, g(t-u)\, du = f(t) * g(t)$$

But $f(t) * g(t-t_0) = (f * g)(t-t_0)$

And $f(t-t_0) * g(t-t_0) = (f * g)(t-2t_0)$

(This result to be carefully noted)

2.2.5 Formation of Convolution

(i) Let f(t) and g(t) be defined in $(-\infty, \infty)$.

(ii) Replace each function in terms of a dummy variable u,

(iii) Replace f(u) or g(u) : $g(u) \rightarrow g(-u)$

(iv) Add a time-offset, t, which allows g(t – u) to slide along u-axis in $(-\infty, \infty)$

(v) The resulting waveform is the convolution of the function f and g.

2.2.6 Circular Convolution

GQ. What is circular and discrete convolution operation?

When a function g(t) is periodic with period T, then for functions, f, such that (f * g) exists, then the convolution is also periodic and identical to :

$$(f * g)(t) = \int_{t_0}^{t_0+T} \left[\sum_{K=-\infty}^{\infty} f(u + KT)\right] g(t+u)\, du,$$

where t_0 is an arbitrary choice. The summation is called a periodic summation of the function f.

And (f * g) is called as circular or cyclic convolution of f and g.

2.2.7 Discrete Convolution Operation

Discrete time convolution is an operation on two discrete time signals defined by :

$$(f * g)(n) = \sum_{K=-\infty}^{\infty} f(K)\, g(n-K)\,;$$

for all signal input functions f(n) and g(n) defined over set of integers Z.

(1) The operation is commutative,

(2) The operation of convolution is linear in each of the two function variables.

2.2.8 Properties

(1) $(K_1 f + K_2 g) * h = K_1 (f * h) + K_2 (g * h)$

(2) $f * (K_1 g + K_2 h) = K_1 (f * g) + K_2 (f * h)$

2.3 PARAMETER SHARING

Parameter sharing makes sets of parameters to be similar as we interpret various models or model components as sharing a unique set of parameters. We only store a subset of memory.

Let us consider two models A and B. They perform a classification task on similar input and output distributions. Here we can assume that the parameters for both models to be identical to each other.

If there is distance between the weights, we can impose norm-penalty. But instead we force the parameters to be similar. This

way we have only to store a subset of the parameters (e.g. storing only the parameters for model A instead of storing for both A and B). This leads to significant memory saving.

Example : The most extensive use of parameter sharing is in convolutional neural networks.

Natural images have specific statistical properties that are robust for translation.

For example, a photo of a cat remain a photo of a cat if it is translated one pixel to the right. Convolutional Neural Networks considers this property by sharing parameters across multiple image locations. It implies that we can find a cat with same cat detector in column (i) or (i + 1) in the image.

(1) Benefits of parameter-sharing
(2) Importance of parameter-sharing
(3) Weight sharing in CNN
(4) Parameters of CNN
(5) Parameters in neural network
(6) Calculation of CNN Parameters
(7) Parameters in deep learning
(8) Parameters in a model
(9) Difference between a variable and a parameter
(10) Parameter Sharing in Deep Learning
(11) Hard parameter sharing
(12) Soft parameter sharing

(1) Benefits of parameter-sharing

- Convolution Neural Networks have a couple of techniques known as parameter sharing and parameter tying. Parameter sharing is the method of sharing weights by all neurons in a particular feature map. It helps to reduce the number of parameters in the whole system, and it makes it computationally cheap.

- Parameter sharing is used in all convolution layers in the network. Parameter sharing reduces the training time and that directly reduces the number of weight updates during back propagation.
- Recurrence Neural Networks also use shared parameters. The shared weights perspective comes from thinking about RNNS as feed. Forward networks. If the weights were different at each moment, this would be a feed forward network.

(2) Importance of parameter sharing

- The main purpose of parameter sharing is a reduction of the parameters that the model has to learn.
- This is the purpose of using RNN. If one learns a different network for each time-step and feed the output of the first model to the second etc; one would end up with a regular feed forward network.

(3) Weight sharing in CNN

- A typical application of weight sharing is to share the same weights across all four filters.
- In this content weight sharing has the following effects. It reduces the number of weights that must be learned and that reduces model learning time and cost.

(4) Parameters of CNN

- In a CNN, Each layer has two kinds of parameters : weights and biases.
- The total number of parameters is just the sum of all weights and biases, = Number of weights of convolutional Layer = number of biases of the convolutional layer.

(5) Parameters in neural network

- Parameters are the coefficients of the model, and they are chosen by the model itself.
- It means that the algorithm, while learning, optimises these coefficients and returns an array of parameters which minimises the error.

(6) Calculation of CNN Parameters

Number of parameters = 0. CONN layer :

This is where CNN learns, hence we have weight matrices. To calculate the learnable parameters, all we have to do is, just multiply the shape of width m, height n, previous layer's filters d and account for all such filters k in the current layer.

(7) Parameters in deep learning

- Model parameters are properties of training data that will learn during the learning process.
- In case of deep learning, weight and bias are parameters. Parameter is used ϕ as a measure of how well a model is performing.

(8) Parameters in a model

- A model parameter is a configuration variable and it is internal to the model and its value can be estimated from data.
- They are required by the model when making predictions. Their values define the skill of the model on the problem. They are estimated from data.

(9) Difference between a variable and a parameter

- There is a clear difference between variables and parameters. A variable represents a model state, and may change during simulation.

- A parameter is commonly used to describe object statistically. A parameter is normally a constant in a single simulation, and is changed only when you need to adjust your model behaviour.

(10) Parameter Sharing in Deep Learning

- The two main approaches for performing MTL (Multi-task Learning) with neural networks are hard and soft parameters sharing, here we wish to learn **shared** or **"similar"** hidden representations for the different tasks.
- In order to impose these similarities between tasks, the model learns simultaneously for all tasks and with some constraints or regularisation on the relationship between related parameters.

(11) Hard parameter sharing

In **hard parameter sharing** we learn a common space representation for all tasks (i.e. completely share weights/ parameters between tasks). This shared feature space is used to model the different tasks, usually task specific layers. Hard parameters sharing acts as regularisation and reduces the risk of over fitting, as the model learns a representation that will generalise well for all tasks.

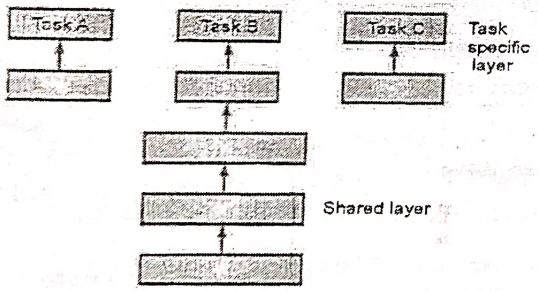


Fig. 2.3.1 : Hard parameter sharing architecture

Example of **hard parameter sharing architecture**

(12) Soft parameter sharing

- Instead of sharing exactly the **same** value of the parameters, in **soft parameter sharing**, we add a constraint to encourage similarities among related parameters.
- We learn a model for each task and penalise the distance between the different models parameters. This approach gives more flexibility for the tasks by only loosely coupling the shared space representations.
- We exhibit an example of soft parameter sharing architecture :

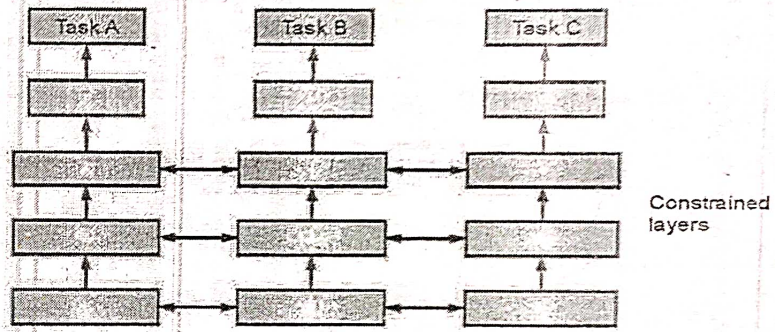


Fig. 2.3.2 : Architecture of soft parameter

Let us suppose that we are interested in learning 2 tasks, A and B, and denote the i^{th} layer parameters for task j by $W_i^{(j)}$, one possible approach to impose similarities between corresponding parameters is to augment our loss function with an additional loss term,

$$L = l + \sum_{S(L)} \lambda_i \| W_i^{(A)} - W_i^{(B)} \|_F^2$$

Where l is the original loss function for both tasks (e.g. sum of losses) and $\| \cdot \|_F^2$ is the squared Frobenius norm.

2.3.1 Pooling Layers

GQ. Explain Pooling layers and its different types.

(1) Pooling layers are used to reduce the dimensions of the feature maps. It reduces the number of parameters to learn and the amount of computation performed in the network.

(2) The pooling layer summarises the features present in a region of the feature map generated by a convolutional layer.

(3) A pooling layers is another building block of a CNN, when processing multichannel input-data, the pooling layer pools each input channel separately.

(4) Pooling layer reduce the dimensions of the data by combining the output of neuron-clusters. The pooling layer is used to reduce spatial dimensions, but not depth, on a convolutional neural network.

(5) Pooling layer operates on each feature map independently.

(6) Pooling is basically 'downscaling' the image obtained from the previous layers. It can be compared to shrinking an image to reduce the pixel density.

2.3.2 Average Pooling

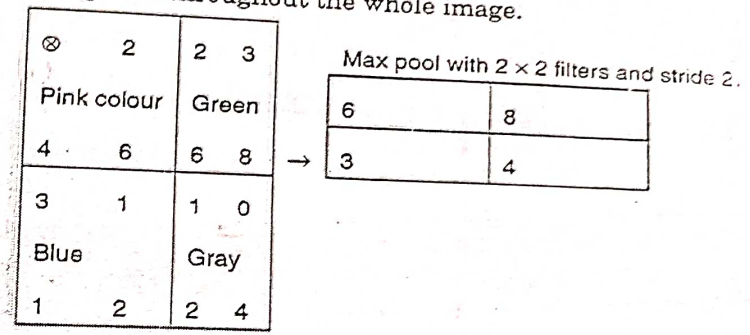
There are two types of widely used pooling in CNN layer.

1. Max pooling	2. Average Pooling

1. Max-pooling

- The popular kind of pooling is **Max-pooling**. Suppose we want to pool by a ratio of 2. It implies that the height and width of your image will be half of it's original value.
- So we have to compress every 4 pixels (a 2×2 grid) and map it to a new single pixel with **loosing** the **important** data from the ***missing*** pixels.

- Max pooling is done by taking the largest value of these 4 pixels. Thus one new pixel represents 4 old pixels by using the largest value of these 4 pixels. This is repeated for every group of 4 pixels throughout the whole image.



Here the largest pixels are 6,8,3 and 4.

- Here the quality of an image is reduced to avoid computational load on the system. By reducing the quality of the image, we can increase the 'depth' of the layer, to have more features in the reduced image.
- Reducing the image size helps the convolution layer after the pooling layer to look for 'higher level features'. It means that the convolution layer looks at the picture as a whole.

2. Average pooling

- **Average pooling** is different from **Max Pooling**. Average pooling retains 'less important' information about the elements of a block, or pool.
- But Max pooling chooses the maximum value and discards less important values (as shown in the Fig. 2.3.3).
- But 'less important' is also sometimes useful in a variety of situations.

4	3	1	5
1	3	4	8
4	5	4	3
6	5	9	4

Av [4, 3, 1, 3] = 2.75

4	3	1	5
1	3	4	8
4	5	4	3
6	5	9	4

→

2.8	4.5
5.3	5.0

Fig. 2.3.3

2.4 VARIANTS OF THE BASIC CONVOLUTION FUNCTION

First we define a few parameters that define a convolutional layer.

(1) **Kernel Size :** The kernel size defines the field of view of the convolution. A common choice for 2 D is 3 – that is 3 × 3 pixels.

(2) **Stride :** The stride defines the step size of the kernel when traversing the image. While its default is usually 1, we can use a stride of 2 for down sampling an image similar to Man Pooling.

(3) **Padding :** The padding defines how the border of a sample is handled. A (half) padded convolution will keep the spatial output dimensions equal to the input, but the unpadded convolution will crop away of the borders if the kernel is larger than 1.

(4) **Input and output channels :** A convolutional layer takes a certain number of input channels (I) and calculates a specific number of output channels (0). The needed parameters for such a layer can be calculated by I. O. K, where K is equal to number of values in the kernel.

2.4.1 Dilated Convolutions

- Dilated convolutions introduce another parameter to convolutions introduce another parameter to convolutional layers called the **dilation rate.** This gives a spacing between the values in a kernel.
- A 3 × 3 kernel with a dilation rate of 2 will have the same field of view as a 5 × 5 kernel, while using only 9 parameters.
- This delivers a wider field of view at the same computational cost.
- Dilated convolutions are particularly popular in the field of real-time segmentation.

2.4.2 Transposed convolutions

- We note that transposed convolution is **not a deconvolution.**
- Deconvolutions do exist but they are not common in the field of deep learning.
- A transposed convolution is somewhat similar because it produces the same spatial resolution a hypothetical deconvolutional layer will.
- But the actual mathematical operation that is performed on the values is different.
- A transposed convolutional layer carries out a regular convolution but reverts its spatial transformation.

We consider an **example :**

An image of 5 × 5 is fed into a convolutional layer.

The stride is set to 2, the padding is deactivated and the kernel is 3 × 3. This results in a 2 × 2 image.

- To reverse the process, a transposed convolution produces an output of 5 × 5 image. It performs a normal convolution operation.
- It reconstructs the spatial resolution from before and performs a convolution. (It is actually not a mathematical inverse, but for Encoder-decoder architecture, it is still very useful)
- This way we can combine the up scaling of an image with a convolution.

2.4.3 Separable convolutions

- In a separable convolution, we can split the kernel operation in to multiple steps.
- We express a convolution as y = convolutional (x, k) where y is the output image, x is the input image and k is the kernel.
- Let us assume that k can be calculated as $k = k_1 \cdot dot_1(k_2)$.
- This makes it a separable convolution because instead of doing a 2D convolution with k, we could get the same result by doing 2 1 D convolution with k_1 and k_2.

In image processing we use sobel kernel

SOBEL X and Y filters

−1	0	+1
−2	0	+2
−1	0	+1

X filter

+1	+2	+1
0	0	0
−1	−2	−1

Y filter

We can get the same kernel by multiplying the vector [1, 0, – 1] and [1, 2, 1] T.

This will require 6 instead of a parameters while doing the same operation.

2.5 LOCALLY CONNECTED LAYER

In some cases, we do not actually want to use convolution, but rather locally connected layers

(1) adjacency matrix in the graph of our MLP is the same, but every connection has its own weight, specified by a 6-D tensor **W**.

(2) The indices into **W** are respectively:

(3) *i*, the output channel,

(4) *j*, the output row,

(5) *k*, the output column,

(6) *l*, the input channel,

(7) *m*, the row offset within the input, and

(8) *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

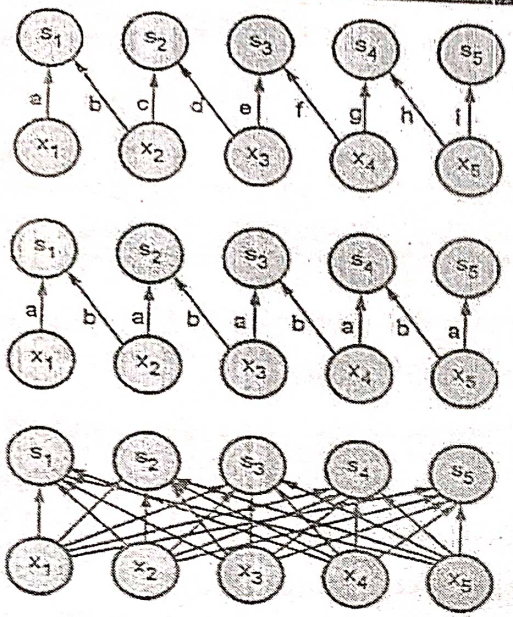


Fig. 2.5.1

☞ **Use of locally connected layers**

- Locally connected layers are useful when

we know that each feature should be a function of a small part of space, but there is no reason to think that the same feature should occur across all of space

Ex : if we want to tell if an image is a picture of a face, we only need to look for the mouth in the bottom half of the image

☞ **Constraining Outputs**

- Constrain each output channel i to be a function of only a subset of the input channels,
  - Make the first m output channels connect to only the first n input channels,
  - The second m output channels connect to only the second n input channels, etc

- Modeling interactions between few channels allows fewer parameters to:
  (1) Reduce memory, increase statistical efficiency, reduce computation for forward/back-propagation.
  (2) It accomplishes these goals without reducing no 15 of hidden units.

Network with further restricted connectivity

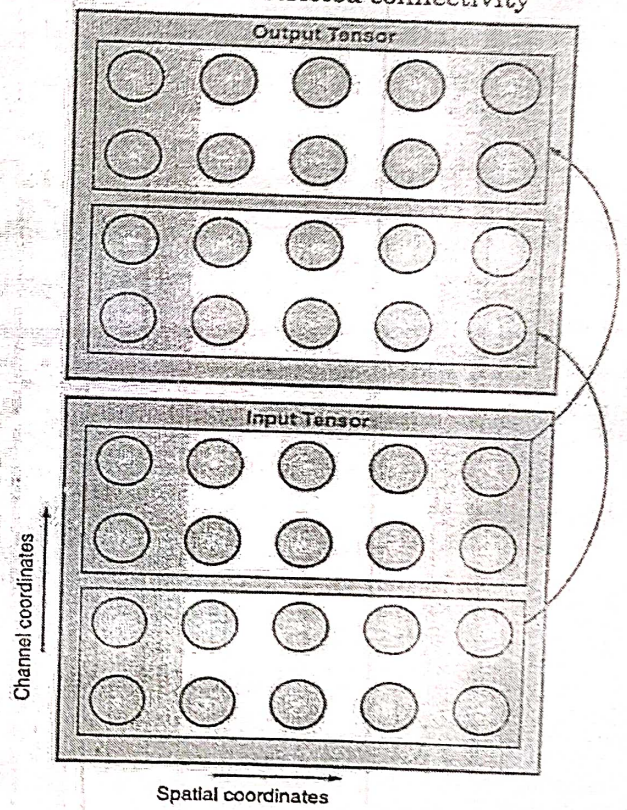


Fig. 2.5.2

☞ Tiled Convolution

- Compromise between a convolutional layer and a locally connected layer.
- Rather than learning a separate set of weights at *every* spatial location, we learn a set of kernels that we rotate through as we move through space.
- This means that immediately neighboring locations will have different filters, like in a locally connected layer,
- but the memory requirements for storing the parameters will increase only by a factor of the size of this set of kernels
- rather than the size of the entire output feature map.
- Comparison of locally connected layers, tiled

☞ Convolution and standard convolution

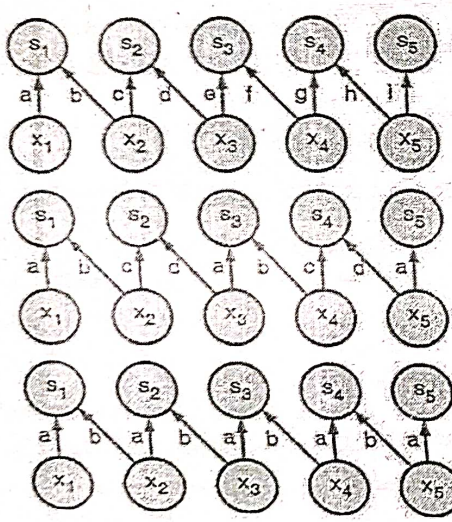


Fig. 2.5.3

☞ Defining Tiled Convolution Algebraically

- Let k be a 6-D tensor, where two of the dimensions correspond to different locations in the output map.
- Rather than having a separate index for each location in the output map, output locations cycle through a set of t different choices of kernel stack in each direction.
- If t is equal to the output width, this is the same as a locally connected layer

where % is the modulo operation, with t%t = 0, (t + 1) %t = 1, etc

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

Operations to implement convolutional nets

- Besides convolution, other operations are necessary to implement a convolutional network.
- To perform learning, need to compute gradient wrt the kernel, given the gradient with respect to the outputs.
- In some simple cases, this operation can be performed using the convolution operation, but when stride greater than 1, do not have this property.

☞ Implementation of Convolution

- Convolution is a linear operation and can thus be described as a matrix multiplication
- if we first reshape the input tensor into a flat vector
- Matrix involved is a function of the convolution kernel
- Matrix is sparse and each element of the kernel is copied to several elements of the matrix.
- This view helps us to derive some of the other operations needed to implement a convolutional network.

2.6 CONVOLUTION NEURAL NETWORK (CNN) ARCHITECTURE

GQ. Explain CNN architecture.

- A **Convolution Neural Network** (CNN) is a feed-forward neural network (FNN). So far CNNs have established extraordinary performance in image search services, voice recognition and natural language processing (NLP).
- We have already seen that a regular multilayer perceptron gives good result for small images. But it breaks down for larger images.
- For example, if the first layer has 1000 neurons, then there will be 10 million connections.
- CNNs solve this problem using partially connected layers. CNN has fewer parameters than a deep neural network (DNN), hence it requires less training data.
- In addition, CNN can detect any particular feature anywhere on the image, but a DNN can detect it only in that particular location.
- Since images have generally repetitive features, CNNs can generalise much better than DNNs for image processing tasks such as classification.
- A CNNs architecture has prior knowledge of how pixels are organised.
- Lower layers identify features in small areas of the images, while higher layers combine features of lower-layer into larger features. In case of DNNs, this doesn't work.
- Thus, in short, CNN is a class of neural networks that specialises in processing data that has a grid-like topology, such as an image.

- Each neuron works in it's own receptive field and is connected to other neurons in a way that they cover entire visual field.
- A CNN design begins with feature extraction and finishes with classification.

DNN Neural network

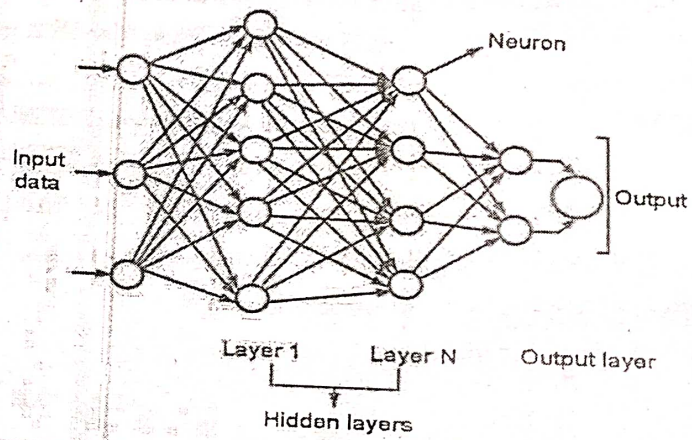


Fig. 2.6.1

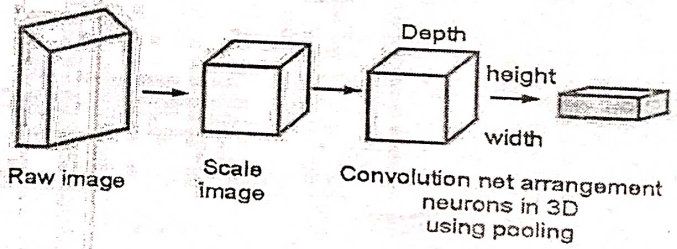


Fig. 2.6.2

In Fig. 2.6.1, there is DNN neural network. On the right, a convolution net arranges it's neurons in 3-dimensions, as seen in one of the layers.

2.6.1 Definition

Convolution is a mathematical operation. In 'convolution neural network' convolution is employed in the network.

2.6.2 Architecture

- A conventional neural network consists of an input layer, hidden layers and output layer. (as shown in the figure). The middle layers are called as hidden layers because their inputs and outputs are governed by activation function and convolution.
- In CNN, the input is a tensor with a shape : (Number of inputs) × (input height) × (input width) × (input channels).
- After passing through a convolutional layer, the image becomes a features map, also called an activation map, with shape :

(number of inputs) × (feature map height) × (feature map width) × (feature map channels).

- The input is convolved in convolutional layers and the result is forwarded to the next layer. Each convolutional neuron processes data only for it's receptive field.
- Fully connected feed forward neural networks architecture is impractical for larger inputs. It requires a very high number of neurons. For example, a fully connected layer of size 100 × 100 has 10,000 weights for each neuron in the second layer.
- Instead using convolution a 5 × 5 filing regions, only 25 learnable parameters are required. Also convolutional neural networks are ideal for data with a grid-like topology since separate features are taken into account during convolution.

2.6.3 Functions of Hidden Layers

GQ. Explain function of Hidden layers.

1. The first hidden layer performs convolution. Each feature map in the hidden layer consists of neurons and each neuron is assigned a receptive field.
2. The second hidden layer performs averaging and subsampling. Each layer consists of feature maps and the neurons of each feature map has a receptive field. It has a trainable bias, trainable coefficient and sigmoid function. They control the operating point of the neuron.
3. The next hidden layer performs a second convolution. Again each feature map in this hidden layer consists of neurons. And each neuron has connections with the previous hidden layer.
4. The next hidden layer performs averaging and subsampling.
5. The output layer performs final stage of convolution. Each neuron is assigned a receptive field and is assigned the possible characters.

The layers in the network alternate between convolution and subsampling, and we get a "bipyramidal" effect. Thus at each layer i.e. either subsampling or convolutional layer, the number of feature maps is increased while the space-resolution is reduced. The weight sharing reduces the number of free parameters in the notwork compared to the synaptic connections in multilayer perceptron. Also by the use of weight sharing, implementation of convolutional network in parallel form is possible.

2.6.4 Design of Multilayer Perceptron

GQ. Explain in detail Multilayer Perceptron.

- Using convolutional network, a multilayer perceptron of manageable size can learn a complex, high-dimensional, nonlinear mapping.
- Through the training set synaptic weights and bias levels can be learned by simple back propagation algorithm.
- Back propagation algorithm is the key-stone algorithm for the multilayer perceptrons. The partial derivatives of the cost function with respect to the free parameters of the network are determined by back-propagating of the error signals of the output neurons, hence the algorithm is called as Back-Propagation Algorithm.
- Updating the synaptic weights and biases of multilayer perceptron and deriving all partial derivatives of the cost function with respect to free parameters are the base factors for evaluating the power of the algorithm.
- Details involved in the **design of a multilayer perceptron** are as follows :

(1) All the neurons in the network are **nonlinear**. And nonlinearity is achieved by using a sigmoid function, and they are

(i) The nonsymmetric logistic function, and

(ii) The antisymmetric hyperbolic tangent function.

(2) Each neuron has it's own hyperplane in decision space, and the combination of hyperplanes formed by all the neurons in the network is interatively adjusted by a supervised learning process. And this patterns from different classes are separated with the few classification errors.

(3) For the pattern classification, the random back-propagation algorithm is used to perform the training.

(4) In nonlinear regression, the output range of the multilayer perceptron should be sufficiently larger.

2.6.5 Performance Measure

(i) Algorithm is based on **minimising the cost function**,

(ii) But minimising the cost function may lead to optimising the intermediate quantity, and this may not be the aim of the system.

Hence 'reward-to-volatility' ratio as a performace measure of risk-adjusted return is more appreciable than Eav.

2.6.6 Input Layer

The input layer passes the data directly to the first hidden layer. Here the data is multiplied by the first hidden layers weights. Then the input layer passes the data through the activation function then it passes on.

2.7 POPULAR CNN ARCHITECTURE – ALEX NET

Before we proceed with popular CNN Architecture, we define some terminology :

(i) A **wider** network means more feature maps (filters) in the convolutional layers.

(ii) A **deeper** network means more convolutional layers.

(iii) A network with **higher resolution** means that it processes input images with larger width and depth (spatial resolutions). That way the produced feature maps will have higher spatial dimensions.

Alex Net

- Alex net is made up of 5 convolutional layers starting from an (11×11) kernel.
- It was the first architecture that employed **max-pooling** layers, ReLu activation functions, and dropout for 3 enormous linear layers.
- The network was used for image classification with 1000 possible classes; Now we can implement it in 35 lines of pyTorch code.
- It was the first convolutional model that was successfully trained on **Imagenet** and at that time, it was more difficult to implement such a model in CUDA.
- To avoid over fitting, dropout is heavily used in the enormous linear transformations.

Inceptionnet / GoogleNet (2014)

- Increasing the depth (number of layers) is not the only way to make a model bigger.
- Increasing **both** the depth and width of the network while keeping computations to a constant level, is the main problem.
- To achieve this, information is processed at multiple scales and then aggregated locally.
- Now, aim is to achieve this without a memory explosion.
- The answer is with $|X|$convolutions. The main aim is dimension reduction, by reducing the output channels of each convolution block. Then we can process the input with different kernel sizes. As long as the output is padded, it is the same as in the input.

- To find the appropriate padding with single stride convs without dilation, padding P and kernel K are defined so that

  Out = in (input and output spatial dims) :

  Out = in + 2 · P − K + 1, which means that

  $$P = \left(\frac{k-1}{2}\right)$$

- Now we need the $|X|$ convolutional layer to 'project' the features to fewer channels in order to win computational power.

  And with these extra resources, we can add more layers.

- Actually, the 1×1 convs work similar to a low dimensional embedding.
- This increases not only depth, but also the width of the famous GoogleNet by using inception modules.
- The core building block, called the inception module is as follows :

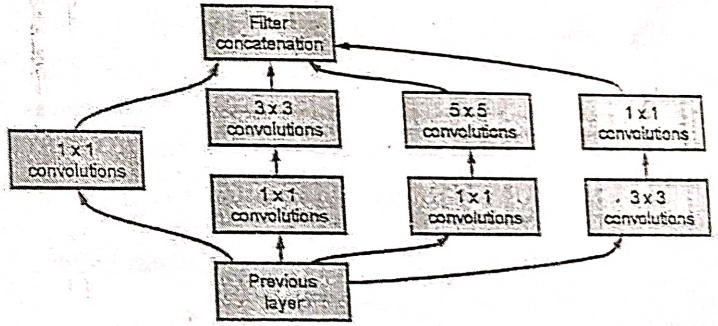


Fig. 2.7.1 : Inception module

- The whole architecture is called GoogleNet or InceptionNet. In essence, they try to approximate a sparse convent with normal dense layers.

- Note that only a small number of neurons are effective. They satisfy the Hebbian Principle
- "Neurons that fire together, wire together".
- In general, a larger kernel is preferred for information that resides globally, and a smaller kernel is preferred for information that is distributed locally.
- Besides, 1×1 convolution are used to compute reductions before the computationally expensive convolutions (3×3 and 5×5).
- The InceptionNet / GoogLeNet architecture consists of 9 inception modules stacked together, with max-pooling layers between (to halve the spatial dimensions).
- It consists of 22 layers (27 with the pooling layers). It uses global average pooling after the last inception module.

...Chapter End

❑❑❑