DL- Mod 4 compressed - deep learning

Deep Learning (University of Mumbai)

Scan to open on Studocu

Studocu is not sponsored or endorsed by any college or university
Downloaded by Arya Patil (arya04patil@gmail.com)

4 Convolutional Neural Networks (CNN) : Supervised Learning

Module 4

Syllabus

4.1 Convolution operation, Padding, Stride, Relation between input, output and filter size, CNN architecture: Convolution layer, Pooling Layer, Weight Sharing in CNN, Fully Connected NN vs CNN, Variants of basic Convolution function.

4.2 Modern Deep Learning Architectures : LeNET : Architecture, AlexNET : Architecture

4.1 Convolution Operation

- The convolution operation is the central operation used in a convolutional neural network.
- This operation makes use of a function to extract certain features from the image.
- This feature extraction is done by means of a filter (also known as kernel or feature detector), which is nothing but a small matrix typically which is applied repetitively over the input image.
- The process of passing filters over an image is called **convolution**.
- Assume that each pixel is represented by only 2 intensity values, 0 or 1.
- The binary image of size 7 × 7 is shown as the input image in Fig 4.1.1.
- Now, suppose we use our filter size as 3 × 3 (shown in Fig. 4.1.1), then, in the resultant image, the value for the first pixel is obtained by multiplying the filter intensity values with the input image intensity values and summing them up.
- To understand this, consider the first 3 × 3 square of the given image and multiply it by filter and sum all the values together to obtain first cell in the output feature map.
- Note that the sum is = 1*0 + 0*0 + 0*1 + 1*1 + 1*0 + 0*0 + 0*0 + 0*1 + 0*1 = 0. So, the first intensity value of the resultant image is 0 (See Fig 4.1.1).

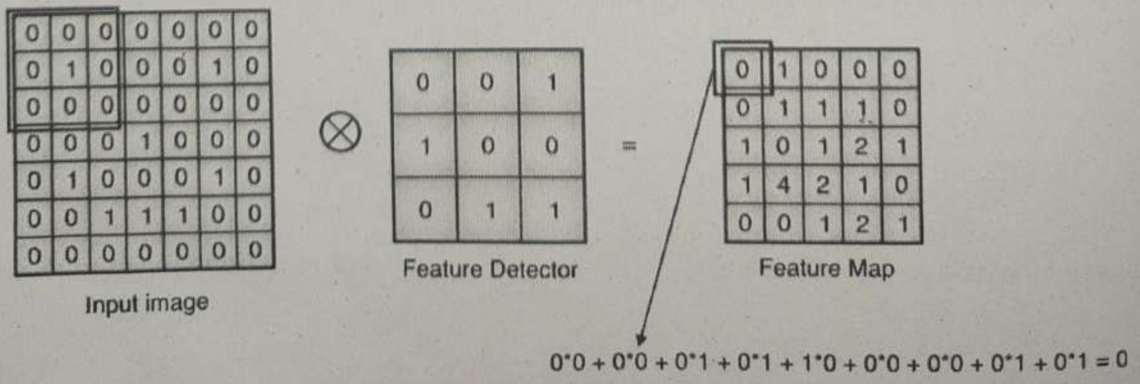

Fig. 4.1.1 : Continue...

This document is available free of charge on studocu

Downloaded by Arya Patil (arya04patil@gmail.com)

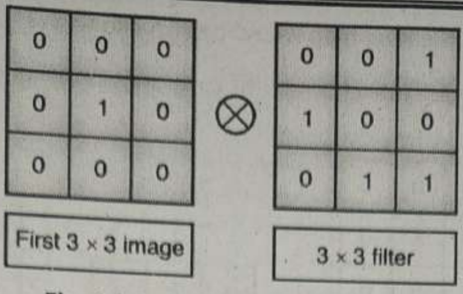

Fig. 4.1.1 : Convolution operation

- Next, we slide this filter to the right by one pixel (See Fig 4.1.2), and convolve the filter with the next part of the image.
- So, the resultant pixel intensity is: 0*0 + 0*0 + 0*1 + 1*1 + 0*0 + 0*0 + 0*0 + 0*1 + 0*1 = 1.

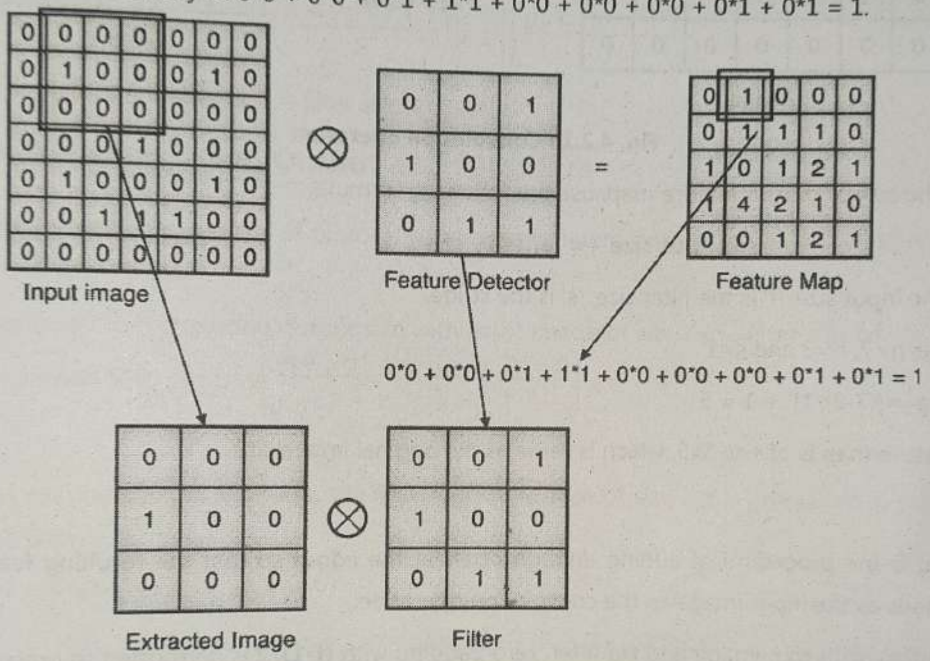

Fig. 4.1.2 : Covolution Opertaion

- Similarly, we slide by one pixel to the right each time and get all intensity values of the first row. After this, we slide the filter one row down and repeat this process to get the intensity output values for the 2nd row and so on, till we get the final feature map which has the reduced dimensions of 5 × 5. Note that the final feature map size is reduced.

4.2 Padding

- As we just discussed in Section 4.1, the convolutional operation reduces the size of the output. So, in cases where we want to retain the size of the output and save the information presented in the corners, we can use padding layers.
- Padding helps by adding extra rows and columns on the outer dimension of the images. So, the size of input data will remain similar to the output data.

Example

- For example, consider the input image with size **5× 5** and we apply **3 × 3** filter with stride 1.

Downloaded by Arya Patil (arya04patil@gmail.com)

- Without padding, the size of output feature map would be reduced to 3x3.
- Now suppose, we pad the image with 1 pixel border of 0s as shown in Fig. 4.2.1. Note that now after padding, the input image size has become 7x7.

$$\underset{\text{Image}}{\begin{array}{|c|c|c|c|c|c|c|}\hline 0&0&0&0&0&0&0\\\hline 0&2&4&9&1&4&0\\\hline 0&2&1&4&4&6&0\\\hline 0&1&1&2&9&2&0\\\hline 0&7&3&5&1&3&0\\\hline 0&2&3&4&8&5&0\\\hline 0&0&0&0&0&0&0\\\hline\end{array}} \times \underset{\text{Filter/ kernel}}{\begin{array}{|c|c|c|}\hline 1&2&3\\\hline -4&7&4\\\hline 2&-5&1\\\hline\end{array}} = \underset{\text{Feature}}{\begin{array}{|c|c|c|c|c|}\hline 21&59&37&-19&2\\\hline 30&51&66&20&43\\\hline -14&31&49&101&-19\\\hline 59&15&53&-2&21\\\hline 49&57&64&76&10\\\hline\end{array}}$$

Fig. 4.2.1 : Convolution operation

- To compute the size of output feature map, use the following formula.

Output size = [(n-f) / s] +1

- Where 'n' is the input size, 'f' is the filter size, 's' is the stride.

  So, in this case, n=7, f =3 and s=1

  So, output size =[(7-3)/1] + 1 = 5
- So, output feature map is of size 5x5, which is same as the original image size.

Same Padding

- Same padding is the procedure of adding enough pixels at the edges so that the resulting feature map has the same dimensions as the input image to the convolution operation.
- For same padding, with **nxn** image and **fxf** filter, zero padding with **(f-1)/2** is performed to preserve the size of the input image after convolution.
- For example, if input image size is 7x7 and the filter size is 3x3 then if we apply zero padding with padding size = (f-1)/2 = (3-1)/2 = 1. i.e. we zero pad with one pixel border, then the output size will be preserved.

Output size = [(n-f) /1] +1 = [(9-3)/1] +1 = 7

4.3 Stride

- In convolution operation, we systematically apply filters to an input image and creates output feature maps. The stride simply describes the step size when sliding the convolutional filter over the input image.
- When the number of strides is 1, we move the filters to 1 pixel at a time. Similarly, when the number of strides is 2, we carry the filters to 2 pixels, and so on.
- They are essential because they control the convolution of the filter against the input, i.e., Strides are responsible for regulating the features that could be missed while flattening the image. They denote the number of steps we are moving in each convolution.

This document is available free of charge on studocu

Downloaded by Arya Patil (arya04patil@gmail.com)

- The Fig. 4.3.1 shows how the convolution would work.

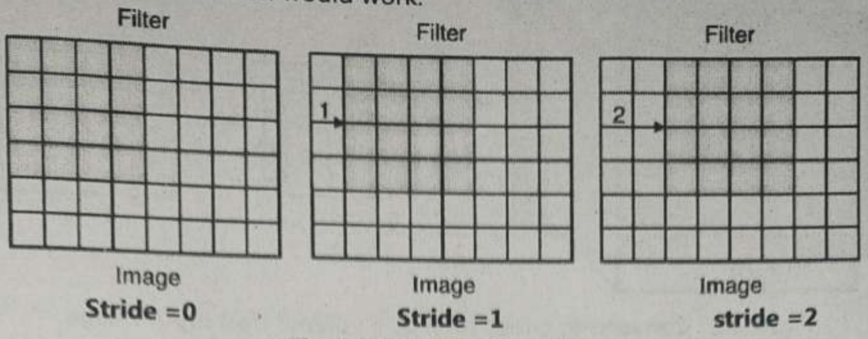

Fig. 4.3.1 : Stride Oprtaion

- In the first matrix, the stride = 0, In th [illegible] je, the s [illegible] the third image the stride=2. The size of the output image is calculated by : $\left[\left(\frac{n+2p-f+1}{S}\right)+1\right]\left[\frac{(n+2p-f+1)}{S}\right]$
- Where, n is input size, p = padding, f = filter size, s = stride.

4.4 Relation between Input, Output and Filter Size

- The choice of filter size, the number of filters and other parameters such as stride and padding together decides the size of output feature map.
- The general formula0 for computing output size with input image of size nxn , filter size fxf , 'k' number of filters, padding 'p' and stride 's' is given as $\left[\frac{(n+2p-f)}{s+1}\right]\times\left[\frac{(n+2p-f)}{s+1}\right]\times k$

Example

- Consider Input volume size: **32 x 32 x 3.** We apply 10 filters each of size 5 x 5 with stride 1, padding 2. Compute Output volume size.

  Here, n = 32 , f = 5, s =1, p=2, k =10.

  Output volume = (32+(2x2)-5 /1 +1) x (32+(2x2)-5 /1 +1) x 10 = 32 x 32 x 10

- Note that the last dimension of the output feature map will be always equal to the number of filters applied to the input image. In this case we apply 10 filters.

4.4.1 Convolution on RGB Image

- Since in RGB image, the number of channels (the depth) are 3 (Red, Green and blue), the input image is a volume of size 6x6x3 (shown in Fig. 4.4.1). So, we need to apply filter with the same number of channels as input. In Fig. 4.4.1, we apply 3x3x3 filter. Note that number of channels (or depth) of filter is also 3.

Note : The third dimension represents the number of channels or depth.

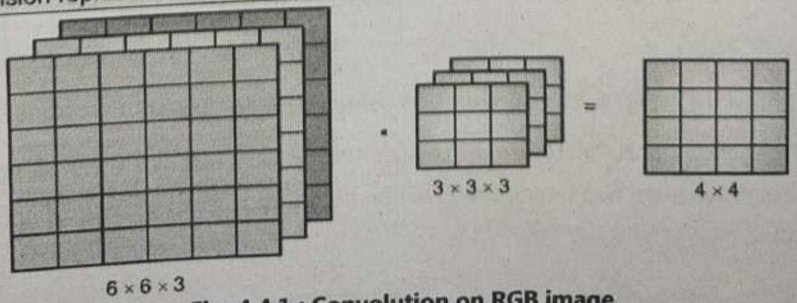

Fig. 4.4.1 : Convolution on RGB image

Downloaded by Arya Patil (arya04patil@gmail.com)

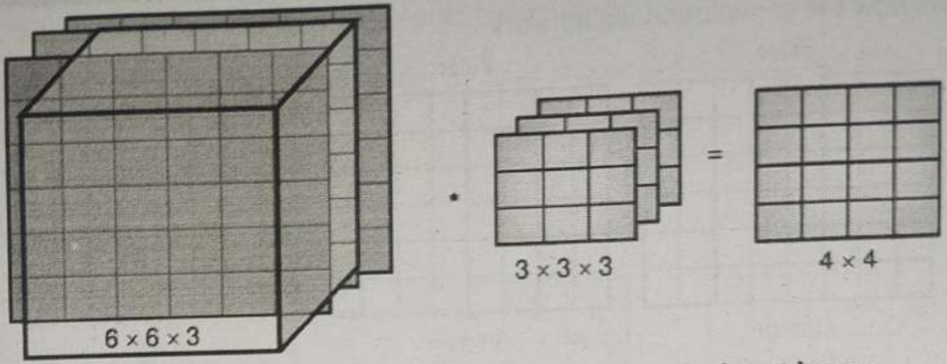

Fig. 4.4.2 : Convolving entire filter as a volume over input image

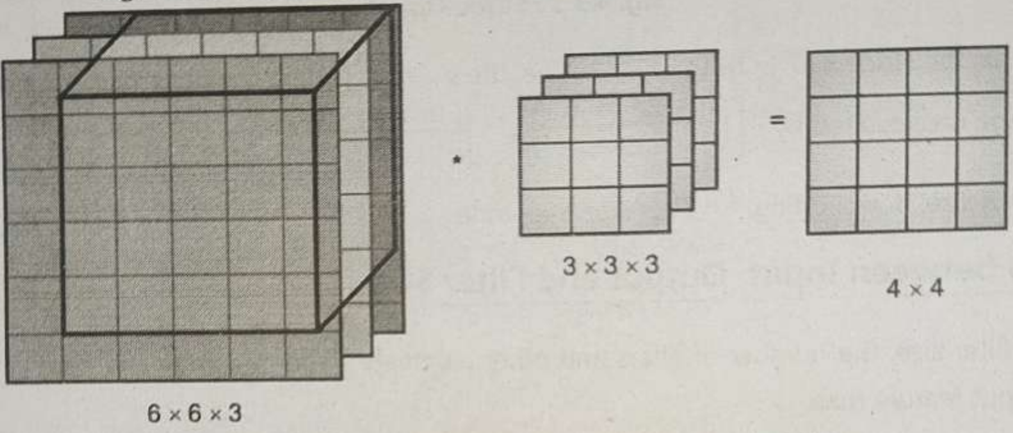

Fig. 4.4.3 : Convolving entire filter as a volume over input image

4.4.2 Applying Multiple Filters

- Till now we have applied only one filter over the input image. But in general, we can apply multiple filters while performing convolution. Each filter extracts different features from an image. In the Fig. 4.4.4, two filters each of dimension 3x3x3 have been applied.

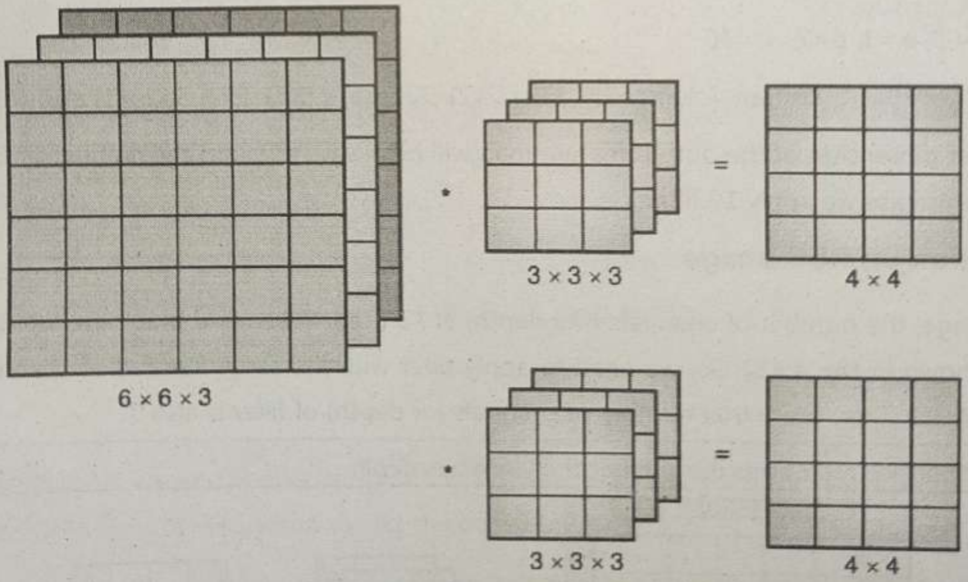

Fig. 4.4.4 : Convolution using multiple filters

- Note that each filter generates a 2D feature map of size 4x4. So, the number of filters defines the depth of the output feature map. So, if there are two filters, there will be two output feature maps generated. Hence, the output feature map size is 4x4x2.

This document is available free of charge on studocu
Downloaded by Arya Patil (arya04patil@gmail.com)

- Formula for finding output size of strides convolutions with padded input with multiple filters:
  - Input : n × n × c (c is the number of channels in the input image)
  - Filter : f × f × c (Number of channels in the filter must match ith the number of channels in input image)
  - Stride : s
  - Number of filters : k
  - pad length: p
  - Resultant output volume : **[(n+2p-f) / s + 1] x [(n + 2p – f)/s + 1] x k**

Ex. 4.4.1 : Given an input volume: 32x32x3 , if we apply 10 filters each of size 5x5x3, with stride 1 and padding 2. Find the output volume.

Soln. :

Here input size n = 32

Filter size f = 5

Number of filters k = 10

Stride s =1 and padding p = 2

Output volume size: ?

Output size Formula = (n+2p-f) / s + 1 x (n + 2p – f)/s + 1 x k

Compute (n+2p- f /s+1) = (32+2*2-5)/1+1 = 32 spatially, so

So, output volume = 32x32x10

4.4.3 One Convolution Layer

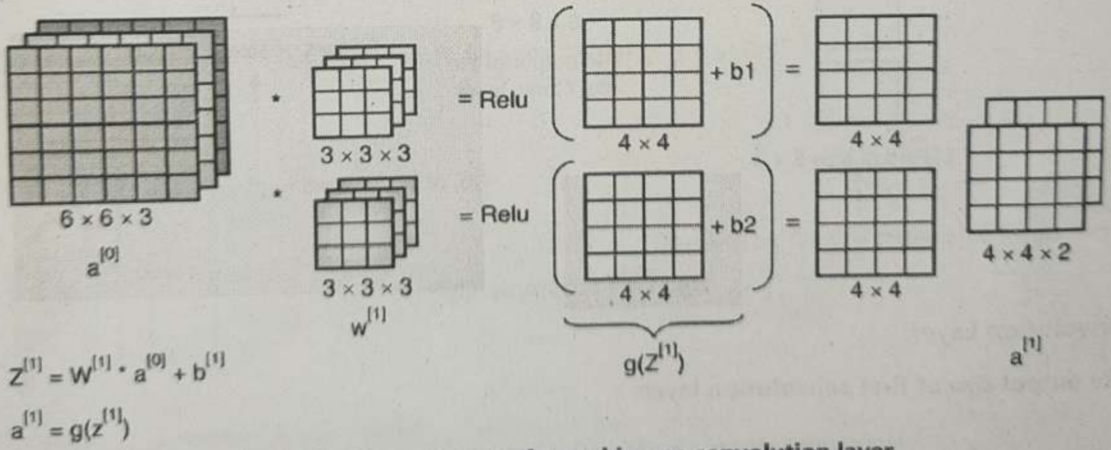

Fig. 4.4.5 : Operations performed in one convolution layer

- Fig. 4.4.5 depicts the operations performed in one convolution layer. The input image is an RGB image with dimension 6x6x3. Note that the number of channels (or depth) of input is 3.
- We apply 2 filters each of dimension 3x3x3. Note that the number of channels in each filter must be same as the no. of channels in input image.
- Convolution of input image with each filter generates an output feature map of size 4x4. So, there are two such 4x4 output feature maps generated.

Downloaded by Arya Patil (arya04patil@gmail.com)

- We apply ReLU activation to add non-linearity. Finally, combining these two 4x4 output feature maps give you 4x4x2 output map. It should be noted that the number of filters defines the number of channels (or depth) in the output feature map.

Calculating Number of trainable parameters in one convolution layer

- If you have 10 filters that are of size 3x3x3 in one layer of a Neural Network, how many parameters do that layer have?

Step 1 : Ccalculate the number of parameters in one filter

No. of parameters in one filter = 3x3x3 = 27 + b = 28 parameters

Note that, one bias has been added here.

Step 2 : Multiply this number with the total number of filters applied

We have 10 such filters

So, total number of parameters = 28 x 10 = 280

Example of calculating number of trainable parameters

Consider convolution neural network with two convolution layers shown in Fig. 4.4.6.

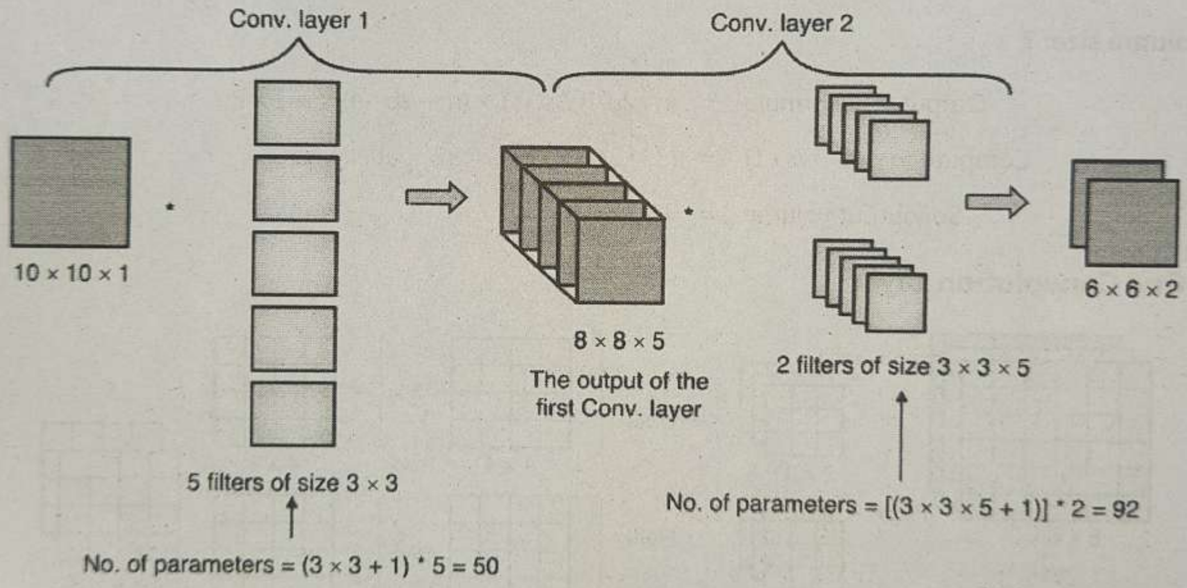
Conv. layer 1
Conv. layer 2
10 × 10 × 1
5 filters of size 3 × 3
No. of parameters = (3 × 3 + 1) * 5 = 50
8 × 8 × 5
The output of the first Conv. layer
2 filters of size 3 × 3 × 5
No. of parameters = [(3 × 3 × 5 + 1)] * 2 = 92
6 × 6 × 2

Fig. 4.4.6 : Example CNN

I. First convolution Layer

1. Calculate output size of first convolution layer

Here input size n = 10 x10x1

No. of filters K = 5

Filter size f = 3x3x1

By default stride s = 1, No padding p = 0

Output size = [(n+2p-f) / s + 1] x [(n + 2p – f)/s + 1] x k

= (10-3)/1 + 1 = 8

= 8 x 8 x 5

This document is available free of charge on studocu
Downloaded by Arya Patil (arya04patil@gmail.com)

2. Calculate number of trainable parameters for first convolution layer

No. of Parameters = No. of parameters in one filter x total number of filter applied

No. of parameters in one filter = 3 x 3 + 1 = 10 parameters

We have 5 such filters

So, total parameters = 10 x 5 = 50

II. Second convolution Layer

For second convolution layer, the output of the first convolution Layer acts as the input.

1. Calculate output size of the second convolution layer

Here input size n = 8 x8x5

Filter size = 3x3x 5

No. of filters K = 2

Stride s = 1, padding p = 0

Output size = [(n + 2p-f) / s + 1] x [(n + 2p – f)/s + 1] x k

= [(8 + 0 – 3)/1 +1] x [(8 +0-3)/1 +1] x 2

= 6 x 6 x 2

2. Calculate number of trainable parameters in second convolution layer

Number of parameters in one filter = (3 x 3 x 5) +1 = 46

We have such 2 filters

Total number of trainable parameters = 46 x 2 = 92

4.5 CNN Architecture : Convolution Layer, Pooling Layer

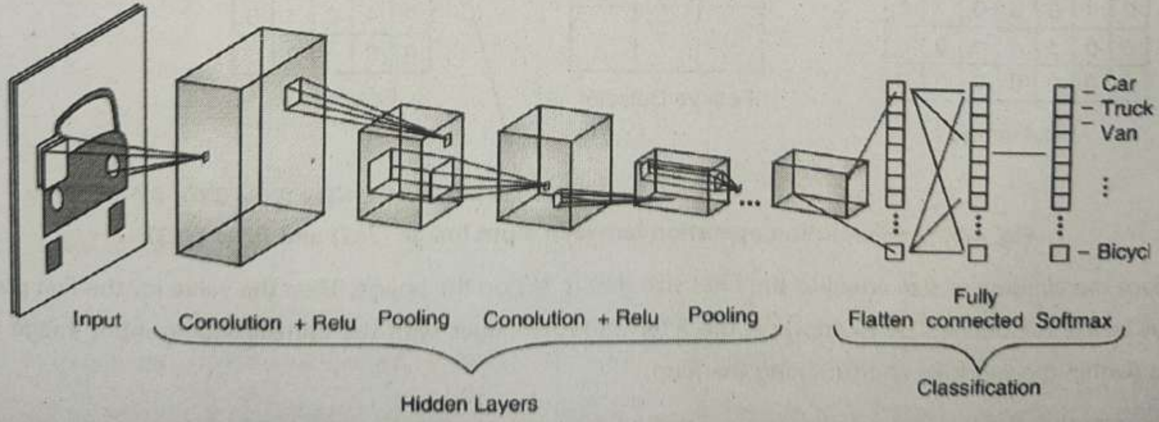

Fig. 4.5.1 : Basic structure of cconvolution neural network

- The convolution neural network contains many layers in its architecture: Most commonly used layers are: Input , convolution, pooling and fully connected layer.
- **Input Layer :** This layer accepts the input representation of the image sample.

Downloaded by Arya Patil (arya04patil@gmail.com)

- **Convolution Layer :** The convolution layer is the heart of the CNN architecture. This layer performs the convolution between the input image and the filters. It computes the output volume by computing the dot product between all filters and the input image. The convolution of an image with different filters allows to extract different features from the image. After applying the convolution operation, we apply an activation function element-wise. Some common activation functions are RELU : max (0, x), Sigmoid : 1/(1 + e – x), Tanh, Leaky RELU, etc. The volume remains unchanged after applying the activation.
- **Pooling Layer :** This layer is applied at regular intervals throughout a convolutional network to **reduce the volume of the output after convolution operation** in order to make the computation faster and to prevent overfitting. The most commonly used pooling operations are max pooling and average pooling.
- **Fully-Connected Layer :** The fully connected layer is applied in the end to combine the extracted features and finally predict the class to which the image belongs. There can be multiple fully connected layers that are connected in feed-forward fashion to obtain the final output.

4.5.1 Convolution Layer

- The convolution operation is the central operation used in a convolutional neural network. This operation makes use of a special function called filter (also known as kernel or feature detector) to extract certain features from the image.
- A filter is a small matrix typically which is applied repetitively over the input image.
- The process of passing filters over an image and computing a sum of dot product of corresponding intensity values of the image and filter is called **convolution.**
- The Fig. 4.5.2 explains the convolution operation between an input image of size 7 x 7 and a filter of size 3 x 3.

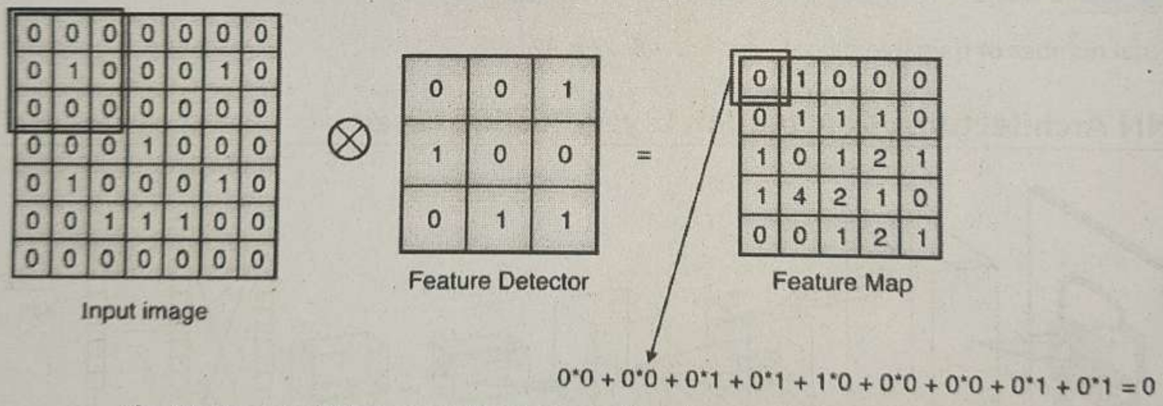

Fig. 4.5.2 : convolution operation between input image (7x7) and filter (3x3)

- We place the window of size equal to the filter size that is 3x3 on the image. Then the value for the first pixel in the output image is obtained by multiplying the filter intensity values with the corresponding input image intensity values (within the window) and summing them up.
- After computing the first output pixel value, we slide the window over the input image by one pixel (see Fig. 4.5.2) and again compute the sum of dot products of corresponding intensity values to obtain second pixel value in input feature map. This process is repeated until entire image is covered.
- Fig. 4.5.3 shows convolution operation between input image (7x7) and filter (3x3). Note the window over the input image is slid by one pixel.

This document is available free of charge on studocu

Downloaded by Arya Patil (arya04patil@gmail.com)

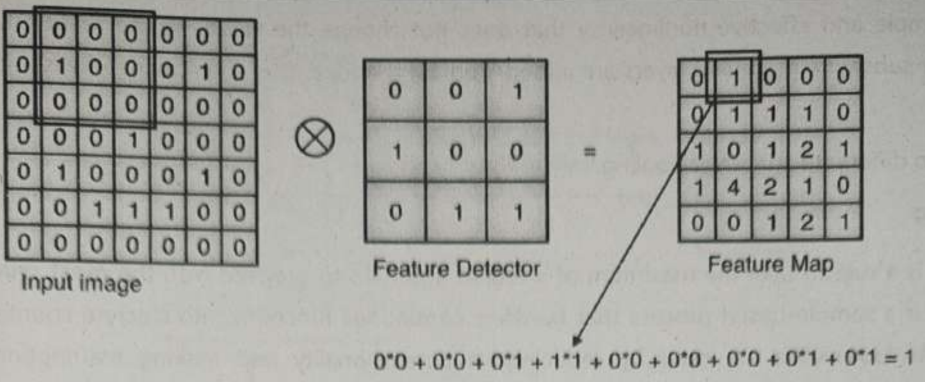

Fig. 4.5.3

Calculating size of the output feature map

Consider following parameters:

- The input image dimensions are **h x w x d**
- The dimensions of the filter are **f_h x f_w x d**
- Then the output is calculated as $(h- f_h +1) \times (w- f_w+1) \times 1$
- Note that initially depth of filter will be equal to the depth of input image. In this case d =1.

In the above example

- Input image dimensions are h x w x d = 7x7x1
- The dimension of filter $f_h \times f_w \times d = 3 \times 3 \times 1$
- Then the output is calculated as,

$$(h- f_h +1) \times (w- f_w+1) \times 1 = (7-3+1) \times (7-3+1) \times 1 = \mathbf{5 \times 5 \times 1}$$

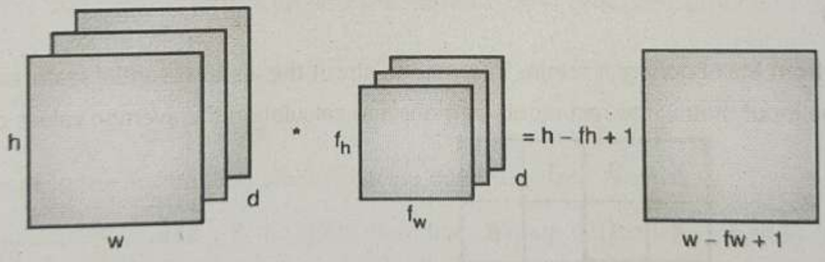

Fig. 4.5.4 : Convolution image with filter

4.5.2 Pooling Layer

- The pooling layer is another building block of a CNN and plays a vital role in pre-processing an image. The pooling layer is used to reduce the image size by reducing the number of parameters if the image is too large.
- Basically, its function is to progressively reduce the spatial size of the image to reduce the network complexity and computational cost.
- Spatial pooling is also known as down sampling or subsampling that reduces the dimensionality of each feature map but retains the essential features.
- A rectified linear activation function, or ReLU, is applied to each value in the feature map.

Downloaded by Arya Patil (arya04patil@gmail.com)

- ReLU is a simple and effective nonlinearity that does not change the values in the feature map but is present because later subsequent pooling layers are added. Pooling is added after the nonlinearity is applied to the feature maps.
- There are two different variants of pooling.

1. Max Pooling

Max pooling is a rule to take the maximum of a region and help to proceed with the **most crucial features** from the image. It is a sample-based process that transfers continuous functions into discrete counterparts. Its primary objective is to downscale an input by reducing its dimensionality and making assumptions about features contained in the sub-region that were rejected.

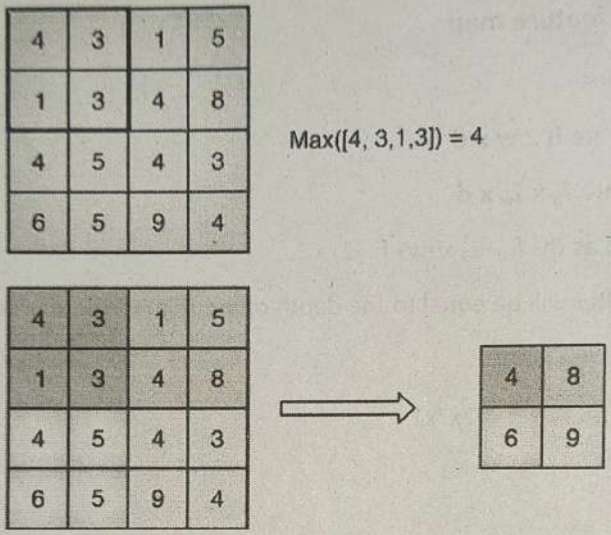

Fig. 4.5.5 : Example of max pooling

2. Average Pooling

- It is different from Max Pooling; it retains information about the lesser essential features. It simply downscales by dividing the input matrix into rectangular regions and calculating the average values of each area.

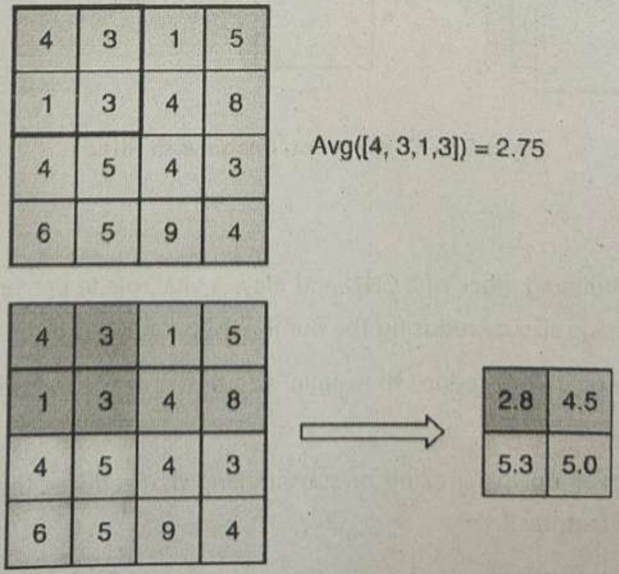

Fig. 4.5.6 : Example of average pooling

This document is available free of charge on studocu
Downloaded by Arya Patil (arya04patil@gmail.com)

Examples of pooling

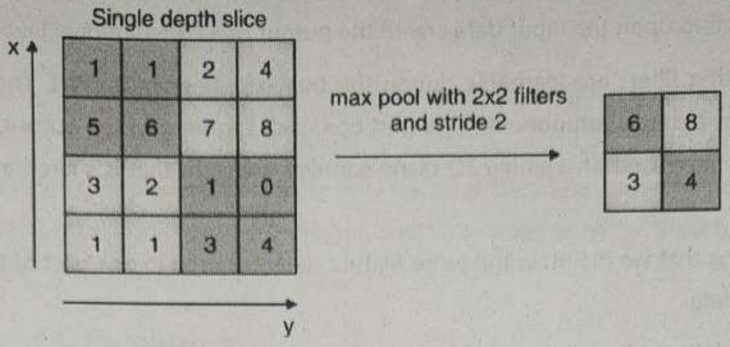

Fig. 4.5.7 : MAX POOLING

- For the given example, the filter size is 2×2 and is applied over the given image of 4×4. If a stride of 2 is used, that is moving 2 columns to the right and by 2 rows down, then the resultant image after pooling operation would be 2×2.
- At each instance, in the 2×2 neighborhood under consideration, the maximum intensity value is selected in the produced output.

So, we have max(1,1,5,6) = 6, the first intensity value

Max(2,4,7,8) = 8, the 2nd intensity value and so on.

More examples on pooling

- If the input size id 7×7 and 3× 3 pooling is applied with **stride 1**, then compute output size.

Use formula : $(n-f)/s + 1 \rightarrow [(7-3)/1] + 1 \rightarrow 5\times5$

So output size = 5 x 5

- If the input size is 7×7 and 3× 3 pooling is applied with **stride 2**, then compute output size.

Use formuls : $(n-f)/s + 1 \rightarrow \frac{(7-3)}{2} + 1 \rightarrow 3 \times 3$

So, ouitput size = 3x3

- In general, if the input to the maxpooling block is having the dimensions $w_1\times h_1\times d_1$
- And f is the pooling dimension, and s is the stride, then the resultant output is $w_2 \times h_2\times d_2$

Where $w_2 = (w1-f)/s + 1$

$h_2 = (h1-f)/s + 1$

$d_2 = d_1$

4.6 Weight Sharing in CNN

- A convolutional layer (conv layer) within a CNN contains a set of units, which can also be referred to as neurons. The conv layer also includes several filters within the layer, and this is a predefined hyperparameter.
- The number of filters within a layer indicates the depth dimension of the output volume of the activation/feature maps that are created by the conv layer as input to the next layer.

Downloaded by Arya Patil (arya04patil@gmail.com)

- Each of these filters has a set width and height, which corresponds to the local receptive field of a single unit within the layer. The filters acting upon the input data create the output of a convolutional layer, the feature map.
- The weight values within filters are learnable during the training phase of a CNN. The output dimension of the convolutional layer has a depth component, if we partition each segment of the output, we will obtain a 2D plane of a feature map. The filter used on a single 2D plane contains a weight that is shared across all filters used across the same plane.
- The advantage of this is that we maintain the same feature detector used in one part of the input data across other sections of the input data.
- The output of a convolutional layer is a set of feature maps, where each feature map is the result of a convolution operation between the fixed weight parameters within the unit and the input data.
- One of the essential characteristics of the convolutional neural network layer is its ability for the feature map to reflect any affine transformations that are made to the input image that is fed through the input layer.
- So any shift, tilt, or orientation made to the input data, the feature map will provide an output that is shifted, tilted, or orientated by the amount the input data was subjected to.

4.7 Fully Connected NN vs CNN

4.7.1 Fully Connected Neural Network

- In a fully connected neural network, the connections are dense means every neuron in one layer is connected to every neuron in the other layer.
- The major advantage of fully connected networks is that they are "structure agnostic" i.e. there are no special assumptions needed to be made about the input.
- While being structure agnostic makes fully connected networks very broadly applicable, such networks do tend to have weaker performance than special-purpose networks tuned to the structure of a problem space.

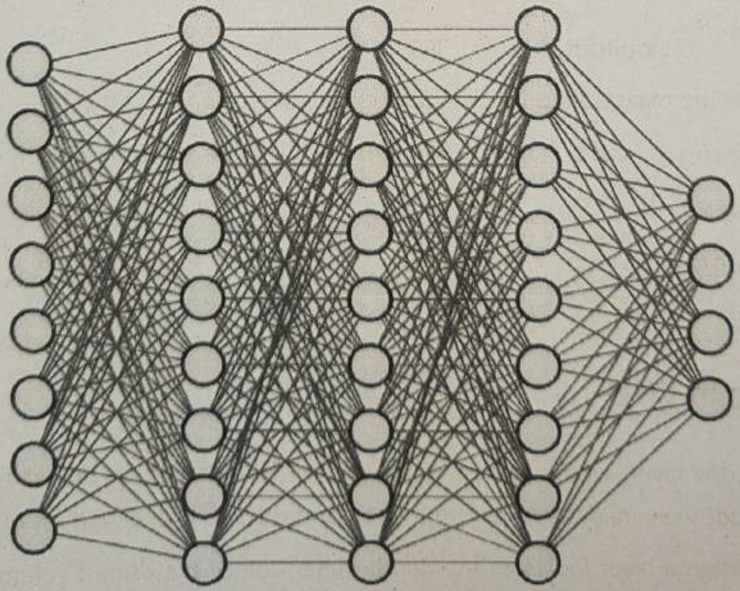

Fig. 4.7.1 : Fully connected Neural network

This document is available free of charge on studocu

Downloaded by Arya Patil (arya04patil@gmail.com)

4.7.2 Convolutional Neural Network

- CNN architectures make the explicit assumption that the **inputs are images**, which allows encoding certain properties into the model architecture.
- A simple CNN is a sequence of layers, and every layer of a CNN transforms one volume of activations to another through a differentiable function. Three main types of layers are used to build CNN architecture: Convolutional Layer, Pooling Layer, and Fully-Connected Layer.

4.8 Some Variants of Convolution

- Convolution in the context of neural networks does not refer exactly to the standard convolution operation in mathematics. The functions used here differ slightly. In this section, we describe the differences in detail and highlight their useful properties.
  1. Convolution operation in Neural Networks refers to an operation that consists of many applications of convolution in parallel. This is because convolution with a single kernel can only extract one kind of feature, albeit at many locations. Usually, we want to extract many kinds of features at many locations.
  2. Input is usually not a grid of real values, rather it is a vector of observations E.g., a colour image has R ,G, B values at each pixel. Input to the next layer is the output of the first layer which has many different convolutions at each position. When working with images, input, and output are 3-D tensors.
- Assume we have a 4-D kernel tensor K with element $K_{i,j,k,l}$ giving the connection strength between a unit in channel i of the output and a unit in channel j of the input, with an offset of k rows and l columns between output and input units.
- Assume our input consists of observed data 'V' with element $V_{i,j,k}$ giving the value of the input unit within channel i at row j and column k, Assume our output consists of Z with the same format as V.

4.8.1 Full Convolution

1. 0 Padding 1 stride

If Z is produced by convolving K across V without flipping K, then

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

(A) 0 padding s stride

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

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

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

Downloaded by Arya Patil (arya04patil@gmail.com)

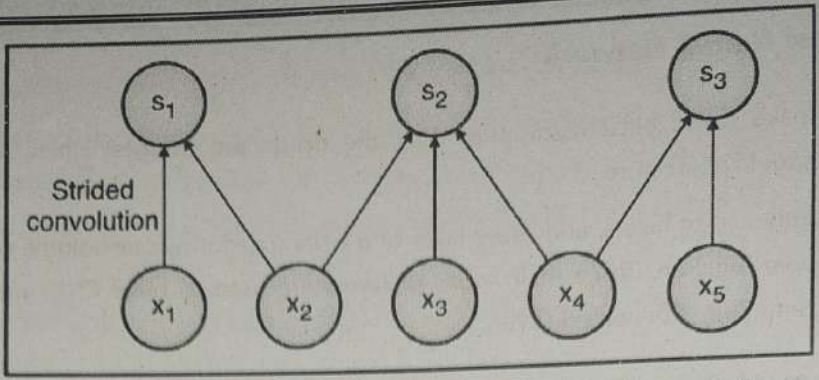

Fig. 4.8.1 : Convolution with a stride of length 2 implemented in a single operation

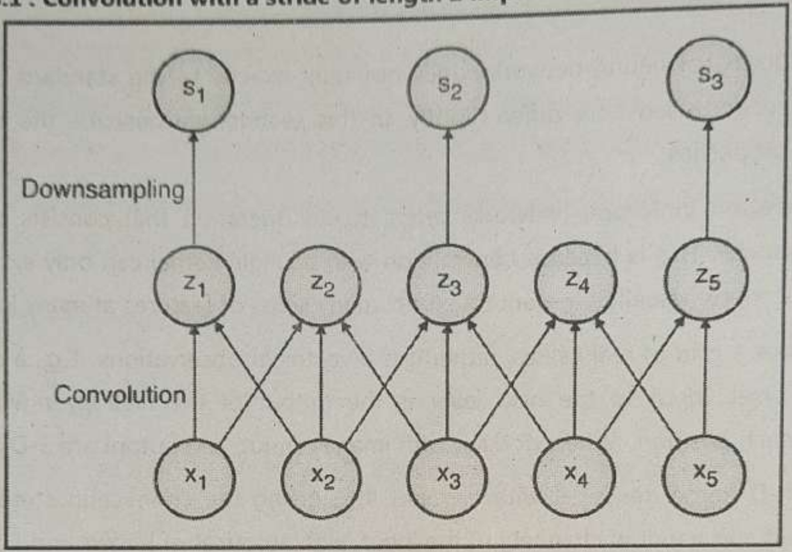

Fig. 4.8.2

- Convolution with a stride greater than one pixel is mathematically equivalent to convolution with a unit stride followed by down-sampling.
- Note that, two-step approach is computationally wasteful, because it discards many values that are discarded.

I. Effect of Zero-padding on network size

- Without 0 paddings, the width of the representation shrinks by one pixel less than the kernel width at each layer. We are forced to choose between shrinking the spatial extent of the network rapidly or using a small kernel. 0 padding allows us to control the kernel width and the size of the output independently.
- To understand this, consider a convolution network with a kernel of width six at every layer.
- Assume we do not use any pooling layer. So, only the convolution operation shrinks the network size. Assume that, we also do not use implicit zero padding. This causes the representation to shrink by 5 pixels at each layer.
- If we start the input layer of 16 pixels, we are only able to have 3 (i.e. 16/5 = 3) convolution layers and the last layer does not actually move the kernel. So, ideally only two of the layers are convolution layers.
- The rate of shrinking can be mitigated by using a smaller kernel, but smaller kernels are less expressive. By adding the implicit zeros at each layer we prevent the representation from shrinking with depth. This allows us to make an arbitrary deep neural networks.

This document is available free of charge on studocu
Downloaded by Arya Patil (arya04patil@gmail.com)

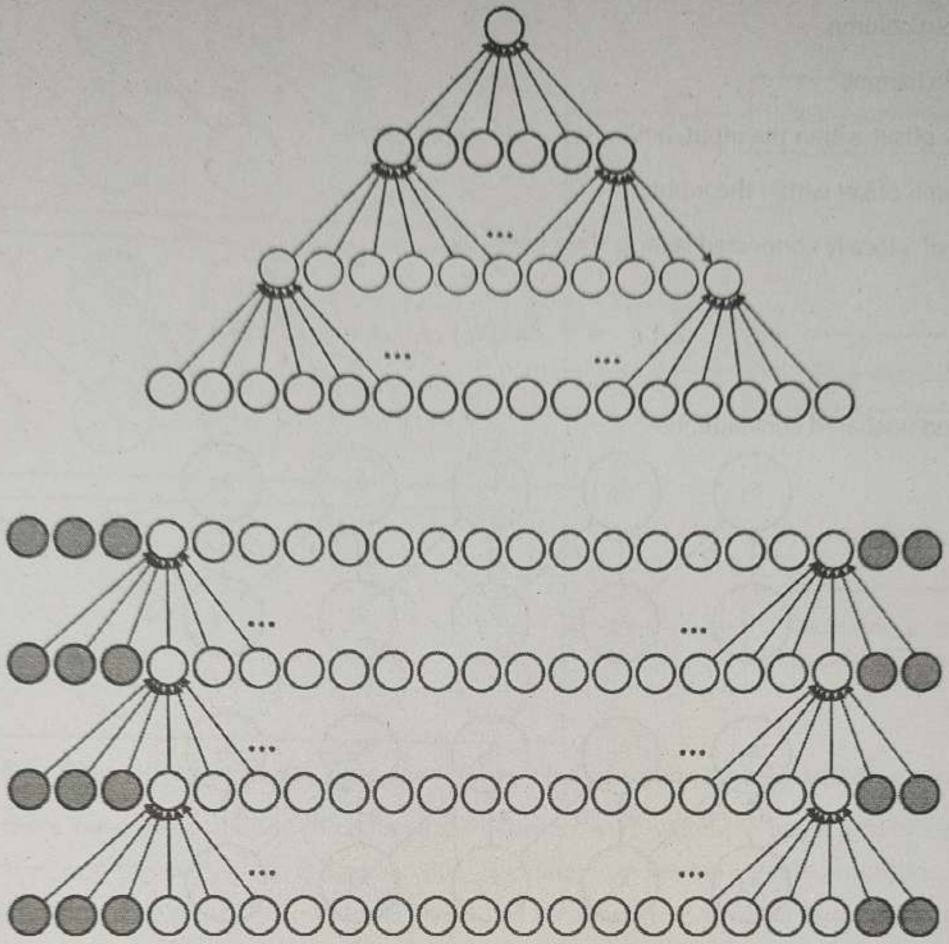

Fig. 4.8.3 : Effect of the zero padding on the network size

II. Special case of 0 padding

- There are three special cases of zero paddings namely: Valid padding, same padding and full padding.
- Valid padding : no 0 padding is used. Limited number of layers.
- Same padding: keep the size of the output to the size of the input. Unlimited number of layers. Pixels near the border influence fewer output pixels than the input pixels near the center.
- Full padding: Enough zeros are added for every pixels to be visited k (kernel width) times in each direction, resulting width m + k - 1. Difficult to learn a single kernel that performs well at all positions in the convolutional feature map.
- Usually, the optimal amount of 0 padding lies somewhere between 'Valid' or 'Same'

4.8.2 Unshared Convolution

- In some case when we do not want to use convolution but want to use locally connected layer, we use Unshared convolution. In this case 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. The indices into W are respectively:
  - i, the output channel,
  - j, the output row,

Downloaded by Arya Patil (arya04patil@gmail.com)

- o k, the output column,
- o l, the input channel,
- o m, the row offset within the input, and
- o n, the column offset within the input.

- The linear part of a locally connected layer is then given by,

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

- This is also called unshared convolution.

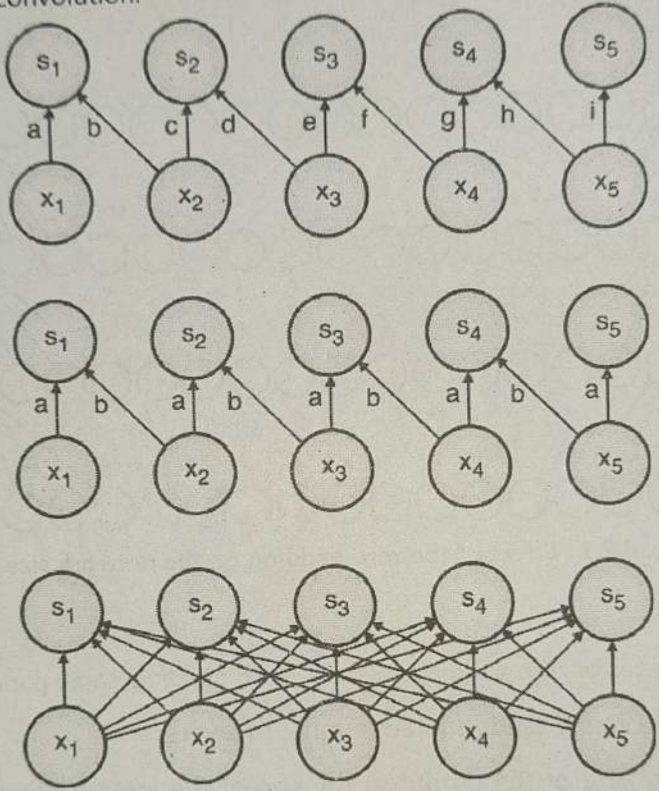

Fig. 4.8.4 : Comparison of local connection, convolution and full connection

- Note that in local connection, parameters are different whereas in convolution parameters have been shared.
- Unshared convolution is useful when we know that each feature should be a function of a small part of space, but no reason to think that the same feature should occur across all the space. e.g.: look for mouth only in the bottom half of the image.
- It can be also useful to make versions of convolution or local connected layers in which the connectivity is further restricted, e.g.: constrain each output channel i to be a function of only a subset of the input channel.

4.8.3 Tiled Convolution

- Learn a set of kernels that we rotate through as we move through space. Immediately neighbouring locations will have different filters, but the memory requirement for storing the parameters will increase by a factor of the size of this set of kernels. Comparison on locally connected layers, tiled convolution and standard convolution:

This document is available free of charge on studocu

Downloaded by Arya Patil (arya04patil@gmail.com)

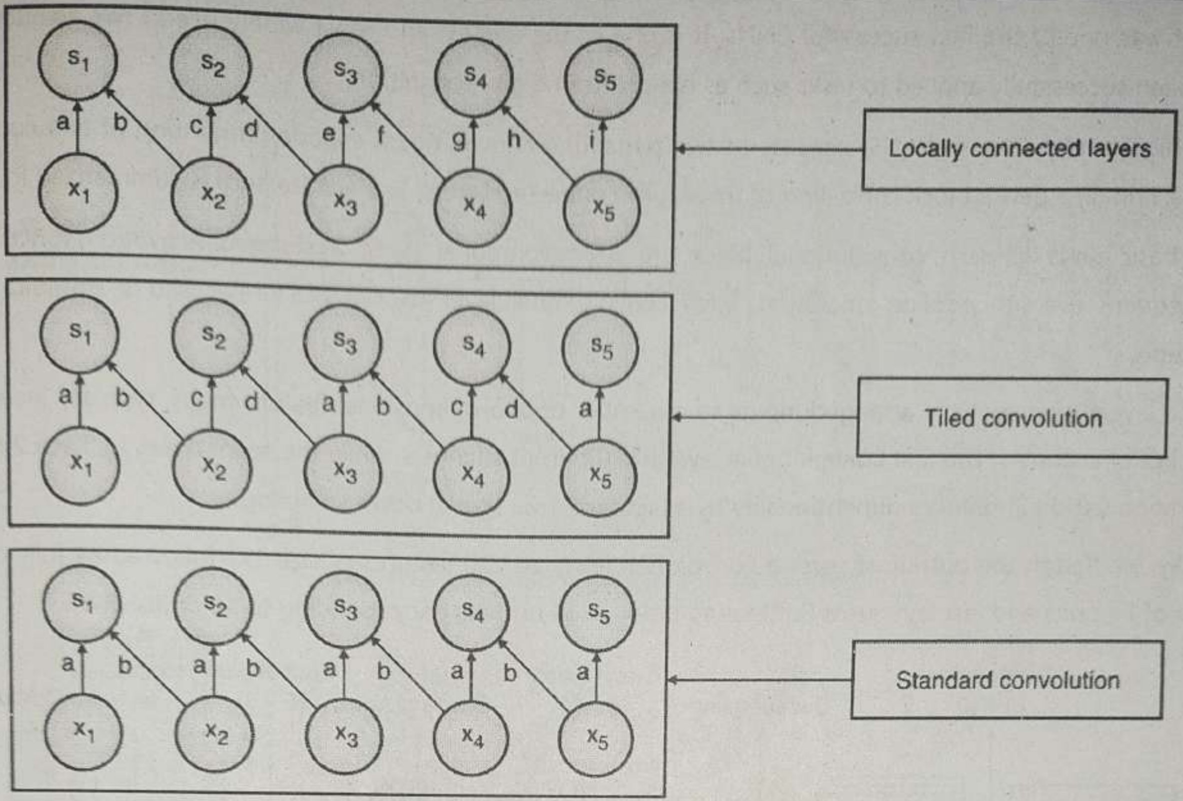

Fig. 4.8.5 : Comparison of locally connected layers, tiled convolution and standard convolution

- Note that all three have the same set of connections between units, when the same size of kernel is used. In the above diagram we use the kernel that is 2 pixels wide. The difference lies between the methods in which they share the parameters. Note that a locally connected layers has no sharing at all. Each connection has its own weight (shown using the unique labelling for each connection). Tiled convolution layer has a set of 't' different kernels. Here t=2 is taken. One of these kernels has edges labelled 'a' and 'b', while the other has edges labelled 'c' and 'd'. Each time we move one pixel to the right in the output, we move on to using a different kernel. This means that like the locally connected layers neighbouring units in the output have different parameters. Unlike the locally connected layer, after we have gone through all 't' kernels we cycle back to the first kernel. Traditional convolution is equivalent to tiled convolution with t=1. There is only one kernel, and it is applied everywhere.

4.9 Modern Deep Learning Architectures

- There are several popular state-of-the-art CNN architectures available. In general, most deep convolutional neural networks are made of a key set of basic layers, including the convolution layer, the sub-sampling/Pooling layer, dense layers, and the soft-max layer.
- Most CNN architectures typically consist of stacks of several convolutional layers and max-pooling layers followed by fully connected SoftMax layers at the end. The basic building components (convolution and pooling) are almost the same across all these architectures. However, some topological differences are observed in modern deep-learning architectures.

4.9.1 LeNet : Architecture

- LeNet is the first CNN architecture. It was developed in 1998 by Yann LeCun, Corinna Cortes, and Christopher Burges for handwritten digit recognition problems.

Downloaded by Arya Patil (arya04patil@gmail.com)

- LeNet was one of the first successful CNNs. It is one of the earliest and most widely-used CNN architectures and has been successfully applied to tasks such as handwritten digit recognition.
- At a high level, LeNet (LeNet-5) consists of two parts: (i) a convolutional encoder consisting of two convolutional layers; and (ii) a dense block consisting of three fully connected layers; The architecture is summarized in Fig. **.
- The basic units in each convolutional block are a convolutional layer, a sigmoid activation function, and a subsequent average pooling operation. Each convolutional layer uses a 5×5 kernel and a sigmoid activation function.
- These layers map spatially arranged inputs to a number of two-dimensional feature maps, typically increasing the number of channels. The first convolutional layer has 6 output channels, while the second has 16. Each 2×2 pooling operation (stride 2) reduces dimensionality by a factor of 4 via spatial down sampling.
- Finally, we flatten the output of second convolution layer to 120 features of size 1x1 followed by fully connected layer of 84 units and last layer uses SoftMax to produce 10 outputs corresponding to 10 digits (0-9).

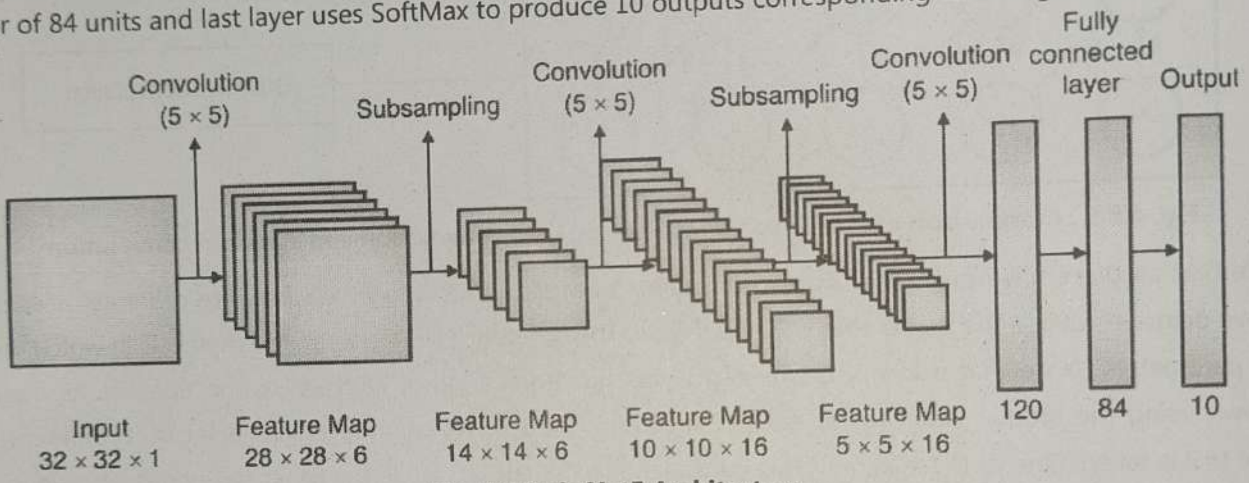

Fig. 4.9.1 : LeNet5 Architecture

4.9.2 AlexNet : Architecture

- AlexNet is the deep learning architecture that popularized CNN. It was developed by Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton.
- AlexNet network had a very similar architecture to LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other.
- AlexNet was the first large-scale CNN and was used to win the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012.
- The AlexNet architecture was designed to be used with large-scale image datasets and it achieved state-of-the-art results at the time of its publication.
- AlexNet is composed of 5 convolutional layers with a combination of max-pooling layers, 3 fully connected layers, and 2 dropout layers (See Fig. 4.9.2 (a)).
- The activation function used in all layers is ReLU. The activation function used in the output layer is SoftMax.
- The total number of parameters in this architecture is around 60 million.

This document is available free of charge on studocu

Downloaded by Arya Patil (arya04patil@gmail.com)

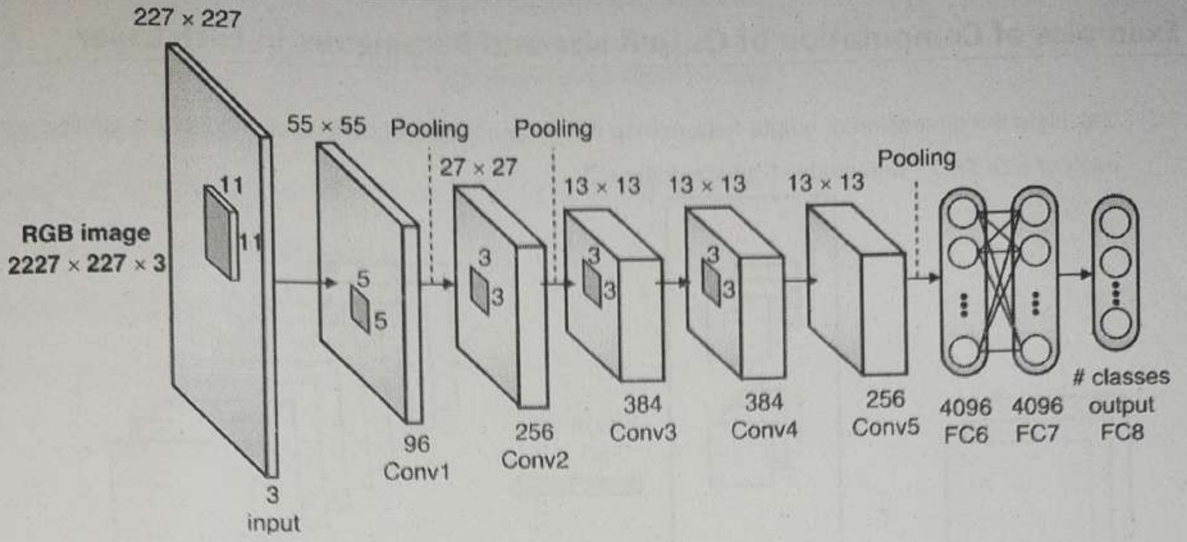

Fig. 4.9.2(a) : General architecture of AlexNet

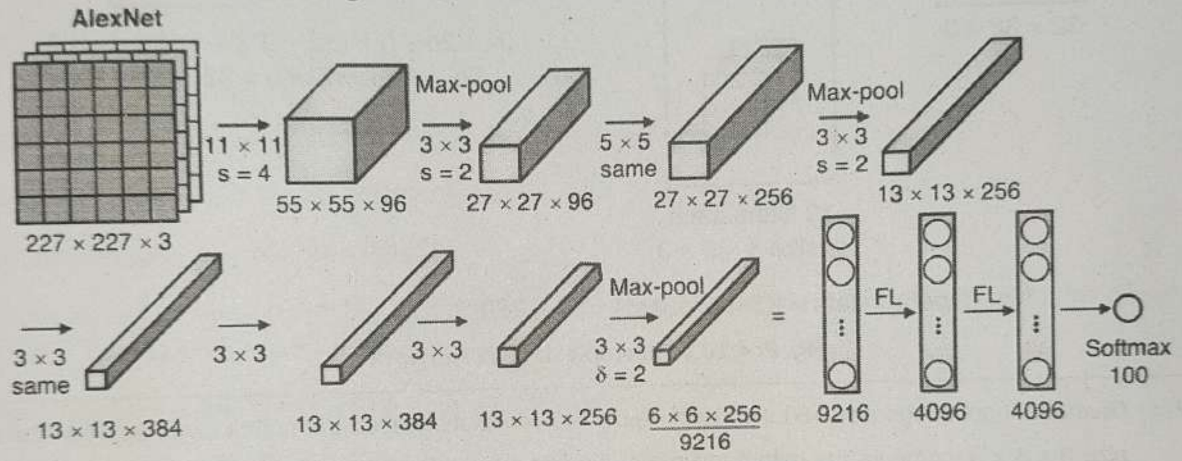

Fig. 4.9.2 (b) : Detailed architecture of AlexNet

Fig (b) shows details architecture of AlexNet. It starts with 227 x 227 x 3 images and the next convolution layer applies 96 of 11 x 11 filter with stride of 4. The output volume reduce its dimension by 55 x 55. Next layer is a pooling layer which applies max pool by 3 x 3 filter along with stride 2. It goes on and finally reaches FC layer with 9216 parameter and the next two FC layers with 4096 node each. At the end, it uses SoftMax function with 1000 output classes. It has **60 million** parameters.

Some of the highlights in AlexNet Architecture:

- It uses ReLU activation function instead Sigmoid or Tanh functions. It speed by more than 5 times faster with same accuracy
- It uses "Dropout" instead of regularisation to deal with overfitting. But the training time is doubled with dropout ratio of 0.5
- More data and bigger model with 7 hidden layers, 650K units and 60M parameters.

Note : ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual computer vision competition. Each year, teams compete on two tasks. The first is to detect objects within an image coming from 200 classes, which is called object localization. The second is to classify images, each labelled with one of 1000 categories, which is called image classification.

Downloaded by Arya Patil (arya04patil@gmail.com)

4.10 Examples of Computation of Output size and Parameters in Each Layer

Ex. 4.10.1 : Calculate the dimension of output feature map if the input image of dimension 32x32x3 is applied with 10 filters each of size 5x5x3 with stride=1 and padding = 2 .

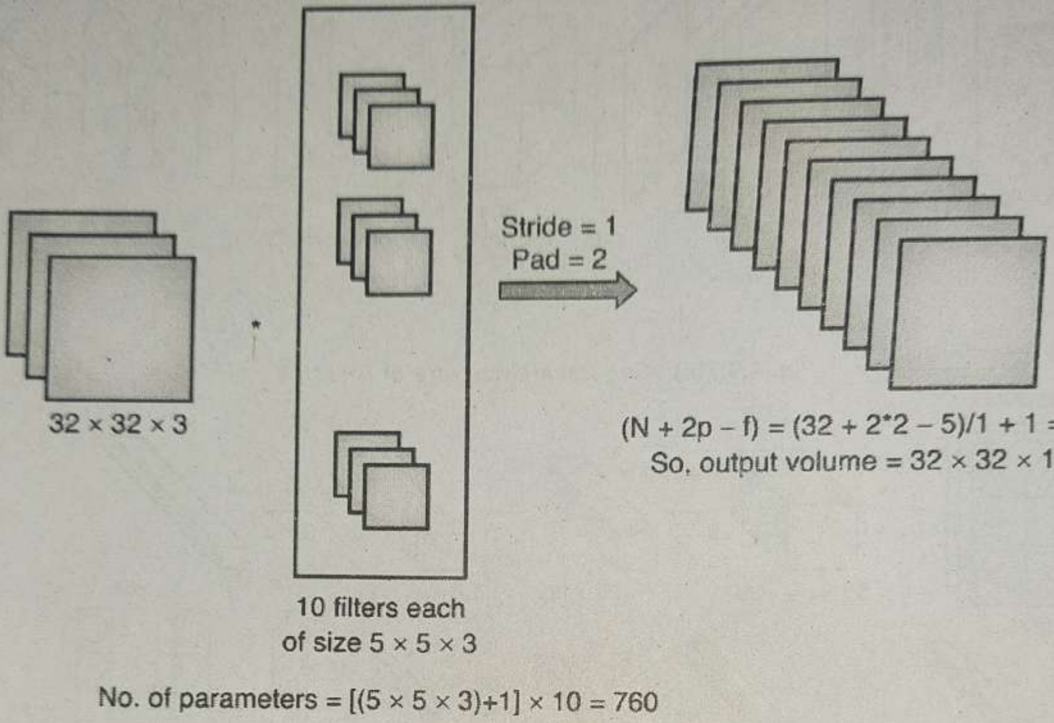
32 × 32 × 3
*
10 filters each of size 5 × 5 × 3
Stride = 1
Pad = 2
(N + 2p – f) = (32 + 2*2 – 5)/1 + 1 = 32
So, output volume = 32 × 32 × 10
No. of parameters = [(5 × 5 × 3)+1] × 10 = 760

Fig. P. 4.10.1 : Example 1: CNN Network

Ex. 4.10.2 : Given an input image size 100 x 100 x 3, Apply two convolution layer. In the first layer, apply 8 filters each of size 3 x 3 x 3, compute the output volume of the first convolution layer. In second convolution layer, apply 1 filter of size 3 x 3, calculate the output volume of second convolution layer. Also compute the number of parameters at each convolution layer. (Assume stride = 1 and padding = 0 when not given)

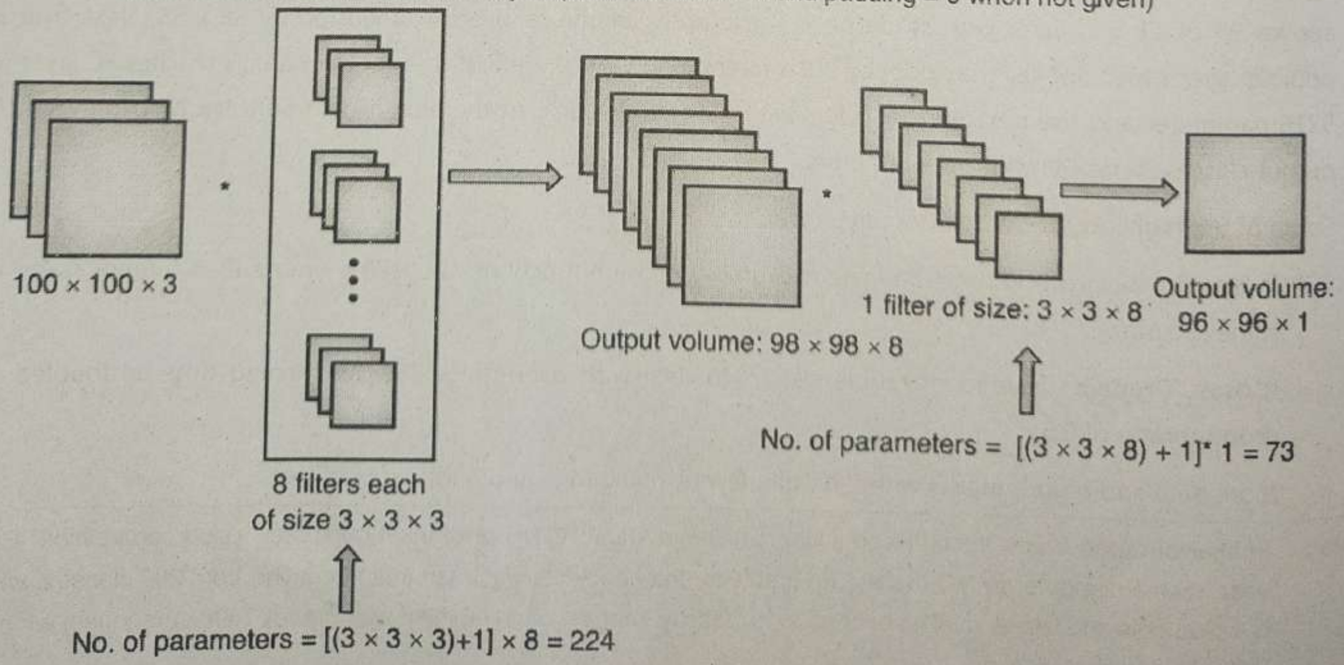
100 × 100 × 3
*
8 filters each of size 3 × 3 × 3
Output volume: 98 × 98 × 8
*
1 filter of size: 3 × 3 × 8
Output volume: 96 × 96 × 1
No. of parameters = [(3 × 3 × 8) + 1]* 1 = 73
No. of parameters = [(3 × 3 × 3)+1] × 8 = 224

Fig. P. 4.10.2 : Example 2 : CNN Network

This document is available free of charge on studocu

Downloaded by Arya Patil (arya04patil@gmail.com)

Review Questions

Q. 1 Explain convolution operation with a suitable example.

Q. 2 Explain following terms w.r.t CNN : Pooling, Stride , padding

Q. 3 Explain CNN architecture in detail.

Q. 4 Explain architecture of AlexNet.

Q. 5 Apply max pooling and average pooling on following image with stride = 2.

9	2	3	5
-1	-34	2	6
3	3	4	5
7	8	10	11
0	0	1	2

Q. 6 Apply padding on the above image and then apply max pooling with stride =2.

Q. 7 Differentiate between fully connected network and CNN.

Q. 8 Differentiate between LeNet and AlexNet.

Q. 9 Given an input image volume of dimension 28x28x3, apply 5 filters each of size 5x5x3. Consider no padding and stride 1, compute the output volume.

Q. 10 Also compute number of trainable parameters in this layer.

□□□

Downloaded by Arya Patil (arya04patil@gmail.com)