Computer Vision

Convolution Neural Networks

Course Instructor:
Dr. Suman Kumar Maji

Introduction

- Convolution neural networks (CNNs) are similar to feedforward networks, but they're usually utilized for image recognition, pattern recognition, and / or computer vision.
- These networks harness principles from linear algebra, particularly matrix multiplication, to identify patterns within an image.

Layers of a CNN

- Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs.
- They have three main types of layers, which are:
  1. Convolutional layer
  2. Pooling layer
  3. Fully-connected (FC) layer
- The convolutional layer is the first layer of a convolutional network.
- While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layer is the final layer.
- With each layer, the CNN increases in its complexity, identifying greater portions of the image.
- Earlier layers focus on simple features, such as colors and edges.
- As the image data progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object.

Padding

- Convolving an input of 6 X 6 dimension with a 3 X 3 filter results in 4 X 4 output.
- Similarly, if the input is n X n and the filter size is f X f, then the output size will be (n-f+1) X (n-f+1):
  - **Input:** n X n
  - **Filter size:** f X f
  - **Output:** (n-f+1) X (n-f+1)
- There are primarily two disadvantages here:
  1. Every time we apply a convolutional operation, the size of the image shrinks
  2. Pixels present in the corner of the image are used only a few number of times during convolution as compared to the central pixels. Hence, we do not focus too much on the corners since that can lead to information loss

cont...

- To overcome these issues, we can pad the image with an additional border, i.e., we add one pixel all around the edges.
- This means that the input will be an 8 X 8 matrix (instead of a 6 X 6 matrix).
- Applying convolution of 3 X 3 on it will result in a 6 X 6 matrix which is the original shape of the image.
- This is where padding comes to the fore:
  - Input: n X n
  - Padding: p
  - Filter size: f X f
  - Output: (n+2p-f+1) X (n+2p-f+1)

Types of Padding

- There are two common choices for padding:
- **Valid:** It means no padding. If we are using valid padding, the output will be (n-f+1) X (n-f+1)
- **Same:** Here, we apply padding so that the output size is the same as the input size, i.e.,
  $n+2p-f+1 = n$
  So, $p = (f-1)/2$
- This way we don't lose a lot of information and the image does not shrink either.

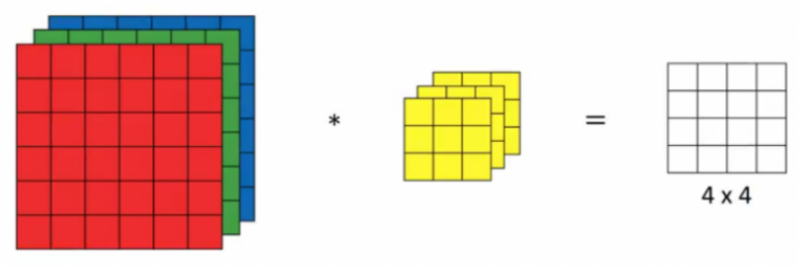
Strided Convolutions

- Stride helps to reduce the size of the image.
- Suppose we choose a stride of 2. So, while convoluting through the image, we will take two steps – both in the horizontal and vertical directions separately. The dimensions for stride s will be:
  - **Input:** n X n
  - **Padding:** p
  - **Stride:** s
  - **Filter size:** f X f
  - **Output:** [(n+2p-f)/s+1] X [(n+2p-f)/s+1]

Convolutions Over Volume

- Suppose, instead of a 2-D image, we have a 3-D input image of shape 6 X 6 X 3.
- We will use a 3 X 3 X 3 filter instead of a 3 X 3 filter. Let's look at an example:
  Input: 6 X 6 X 3
  Filter: 3 X 3 X 3
- The dimensions above represent the height, width and channels in the input and filter.
- The number of channels in the input and filter should be same.
- This will result in an output of 4 X 4.

cont...

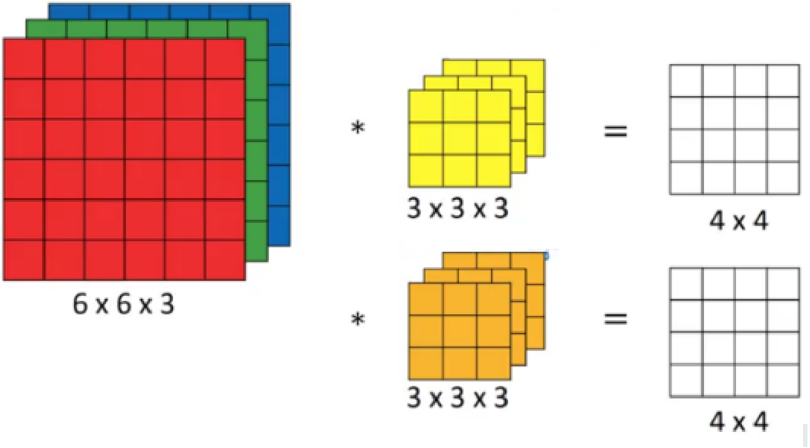


- Since there are three channels in the input, the filter will consequently also have three channels.
- After convolution, the output shape is a 4 X 4 matrix.
- So, the first element of the output is the sum of the element-wise product of the first 27 values from the input (9 values from each channel) and the 27 values from the filter.
- After that we convolve over the entire image.

Multiple Filters Edges

- Instead of using just a single filter, we can use multiple filters as well.
- Let's say the first filter will detect vertical edges and the second filter will detect horizontal edges from the image.
- If we use multiple filters, the output dimension will change.
- So, instead of having a 4 X 4 output as in the above example, we would have a 4 X 4 X 2 output (if we have used 2 filters):

cont...



cont...

- Generalized dimensions can be given as:
  - Input: n X n X nc
  - Filter: f X f X nc
  - Padding: p
  - Stride: s
  - Output: [(n+2p-f)/s+1] X [(n+2p-f)/s+1] X nc'
- nc is the number of channels in the input and filter, while nc' is the number of filters.

One Layer of a Convolutional Network

- Once we get an output after convolving over the entire image using a filter, we add a bias term to those outputs and finally apply an activation function to generate activations.
- This is one layer of a convolutional network.
- Recall that the equation for one forward pass is given by:

$$z^{[1]} = w^{[1]} * a^{[0]} + b^{[1]}$$

$$a^{[1]} = g(z^{[1]})$$

cont...

- The input $a^{[0]}$ has dimensions $6 \times 6 \times 3$ and the filters $w^{[1]}$ have dimensions $3 \times 3 \times 3$.
- These activations from layer 1 act as the input for layer 2, and so on.
- The number of parameters in convolutional neural networks is independent of the size of the image.
- It essentially depends on the filter size.
- Suppose we have 10 filters, each of shape $3 \times 3 \times 3$. What will be the number of parameters in that layer? Let's try to solve this:

$$\text{Number of parameters for each filter} = 3 \times 3 \times 3 = 27$$

- There will be a bias term for each filter, so the total parameters per filter:

$$27 + 1 = 28$$

cont...

- As there are 10 filters, the total parameters for that layer:

$$28 \times 10 = 280$$

- No matter how big the image is, the parameters only depend on the filter size.
- Let's have a look at the summary of notations for a convolution layer:

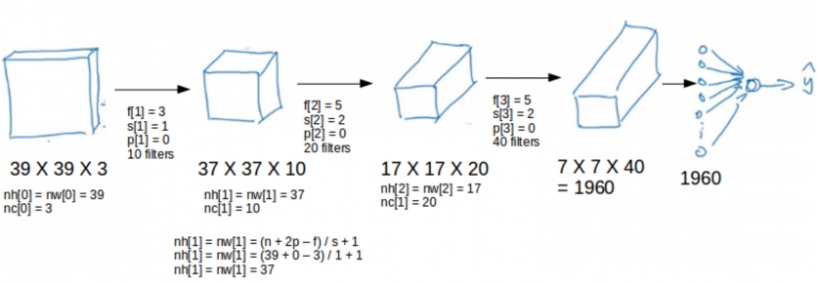
$$f^{[l]} = \text{filter size}$$

$$p^{[l]} = \text{padding}$$

$$s^{[l]} = \text{stride}$$

$$n_c^{[l]} = \text{number of filters}$$

Simple Convolutional Neural Networks Example



nh[1] = nw[1] = (n + 2p – f) / s + 1
nh[1] = nw[1] = (39 + 0 – 3) / 1 + 1
nh[1] = nw[1] = 37

cont...

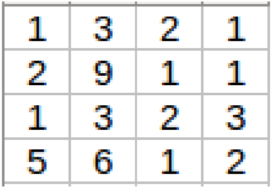
- We take an input image (size = 39 X 39 X 3 in our case), convolve it with 10 filters of size 3 X 3, and take the stride as 1 and no padding.
- This will give us an output of 37 X 37 X 10.
- We convolve this output further and get an output of 7 X 7 X 40 as shown above.
- Finally, we take all these numbers (7 X 7 X 40 = 1960), unroll them into a large vector, and pass them to a classifier that will make predictions.
- This is a microcosm of how a convolutional network works.

cont...

- There are a number of hyperparameters that we can tweak while building a convolutional Neural network.
- These include the number of filters, size of filters, stride to be used, padding, etc.
- We will look at each of these in detail later in this article.
- Just keep in mind that as we go deeper into the network, the size of the image shrinks whereas the number of channels usually increases.
- In a convolutional network (ConvNet), there are basically three types of layers:
  1. Convolution layer
  2. Pooling layer
  3. Fully connected layer

Pooling Layers

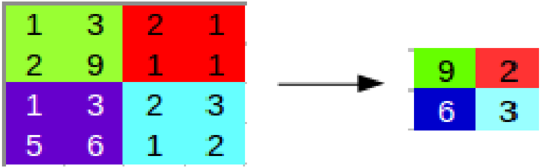
- Pooling layers are generally used to reduce the size of the inputs and hence speed up the computation.
- Consider a 4 X 4 matrix as shown below:



1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

cont...

- Applying max pooling on this matrix will result in a 2 X 2 output:



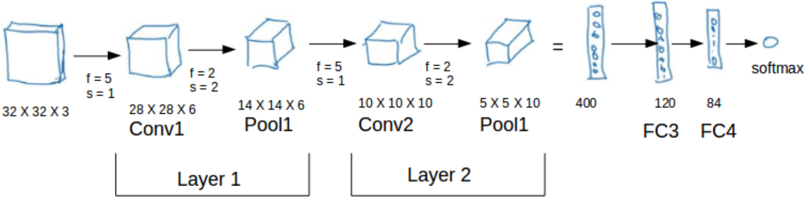
$$\begin{bmatrix} 1 & 3 & 2 & 1 \\ 2 & 9 & 1 & 1 \\ 1 & 3 & 2 & 3 \\ 5 & 6 & 1 & 2 \end{bmatrix} \longrightarrow \begin{bmatrix} 9 & 2 \\ 6 & 3 \end{bmatrix}$$

- For every consecutive 2 X 2 block, we take the max number.
- Here, we have applied a filter of size 2 and a stride of 2.
- These are the hyperparameters for the pooling layer.

cont...

- Apart from max pooling, we can also apply average pooling where, instead of taking the max of the numbers, we take their average.
- In summary, the hyperparameters for a pooling layer are:
  1. Filter size
  2. Stride
  3. Max or average pooling
- If the input of the pooling layer is nh X nw X nc, then the output will be $[\{(nh – f) / s + 1\}$ X $\{(nw – f) / s + 1\}$ X nc].

CNN Example

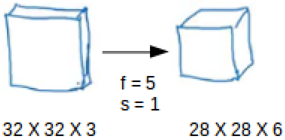

32 X 32 X 3
f = 5
s = 1
28 X 28 X 6
Conv1
f = 2
s = 2
14 X 14 X 6
Pool1
f = 5
s = 1
10 X 10 X 10
Conv2
f = 2
s = 2
5 X 5 X 10
Pool1
=
400
120
FC3
84
FC4
softmax
Layer 1
Layer 2

cont...

- There are a combination of convolution and pooling layers at the beginning, a few fully connected layers at the end and finally a softmax classifier to classify the input into various categories.
- There are a lot of hyperparameters in this network which we have to specify as well.
- Generally, we take the set of hyperparameters which have been used in proven research and they end up doing well.
- As seen in the above example, the height and width of the input shrinks as we go deeper into the network (from 32 X 32 to 5 X 5) and the number of channels increases (from 3 to 10).

Need of Convolutions

- There are primarily two major advantages of using convolutional layers over using just fully connected layers:
  1. Parameter sharing
  2. Sparsity of connections
- Consider the below example:



Parameter Sharing

- If we would have used just the fully connected layer, the number of parameters would be = 32*32*3*28*28*6, which is nearly equal to 14 million.
- If we see the number of parameters in case of a convolutional layer, it will be = (5*5 + 1) * 6 (if there are 6 filters), which is equal to 156.
- Convolutional layers reduce the number of parameters and speed up the training of the model significantly.
- In convolutions, we share the parameters while convolving through the input.
- The intuition behind this is that a feature detector, which is helpful in one part of the image, is probably also useful in another part of the image. So a single filter is convolved over the entire input and hence the parameters are shared.

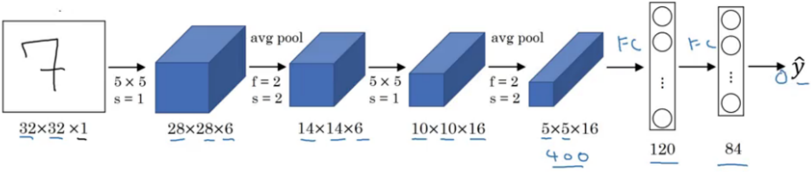
Sparsity of connections

- The second advantage of convolution is the sparsity of connections.
- For each layer, each output value depends on a small number of inputs, instead of taking into account all the inputs.

Deep Convolutional Models: Some classic networks

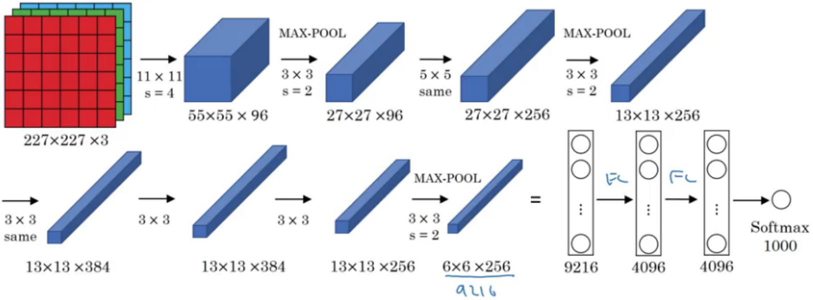
- LeNet-5
- AlexNet

LeNet-5



- It takes a grayscale image as input. Once we pass it through a combination of convolution and pooling layers, the output will be passed through fully connected layers and classified into corresponding classes.
- The total number of parameters in LeNet-5 are:
  - Parameters: 60k
  - Layers flow: Conv → Pool → Conv → Pool → FC → FC → Output
  - Activation functions: Sigmoid/tanh and ReLU

AlexNet


227×227 ×3
11 × 11
s = 4
55×55 × 96
MAX-POOL
3 × 3
s = 2
27×27 ×96
5 × 5
same
27×27 ×256
MAX-POOL
3 × 3
s = 2
13×13 ×256
3 × 3
same
13×13 ×384
3 × 3
13×13 ×384
3 × 3
13×13 ×256
MAX-POOL
3 × 3
s = 2
6x6 ×256
9216
=
9216
FC
4096
FC
4096
Softmax
1000

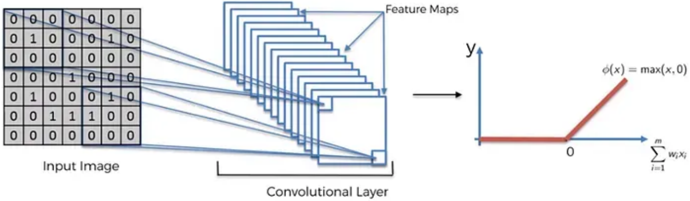
cont...

- This network is similar to LeNet-5 with just more convolution and pooling layers:
  - Parameters: 60 million
  - Activation function: ReLu

Layers of Convolutional Neural Networks

- Activation Layer
- Normalization Layer
- Dropout Layer
- Fully Connected layer

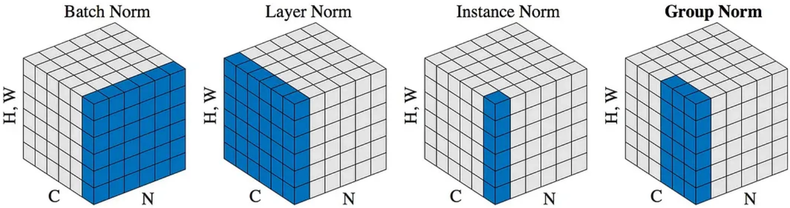
Activation Layer



0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

- The activation layer applies a non-linear activation function, such as the ReLU function, to the output of the pooling layer.
- This function helps to introduce non-linearity into the model, allowing it to learn more complex representations of the input data.

Normalization Layer

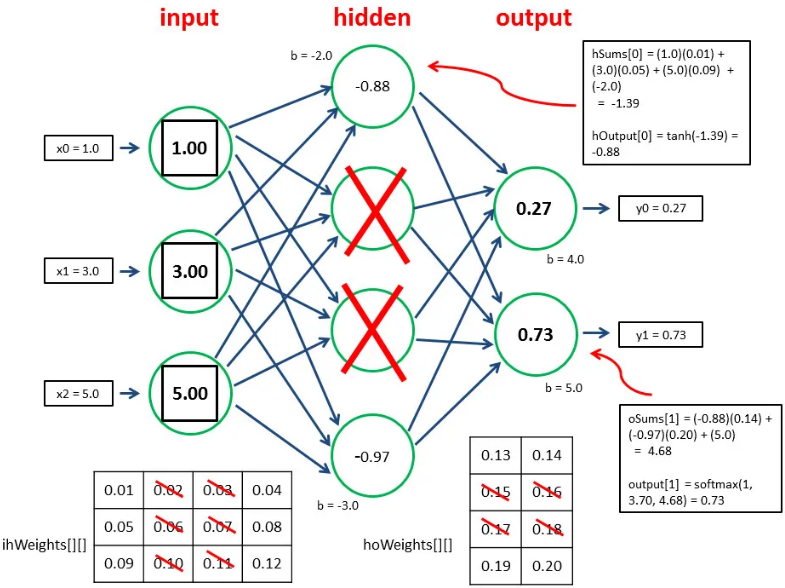


- The normalization layer performs normalization operations, such as batch normalization or layer normalization, to ensure that the activations of each layer are well-conditioned and prevent overfitting.

Dropout layer

- The dropout layer is used to prevent overfitting by randomly dropping out neurons during training.
- This helps to ensure that the model does not memorize the training data but instead generalizes to new, unseen data.

Dropout layer working



hSums[0] = (1.0)(0.01) + (3.0)(0.05) + (5.0)(0.09) + (-2.0)
 = -1.39

hOutput[0] = tanh(-1.39) = -0.88

oSums[1] = (-0.88)(0.14) + (-0.97)(0.20) + (5.0)
 = 4.68

output[1] = softmax(1, 3.70, 4.68) = 0.73

ihWeights[][]

0.01	~~0.02~~	~~0.03~~	0.04
0.05	~~0.06~~	~~0.07~~	0.08
0.09	~~0.10~~	~~0.11~~	0.12

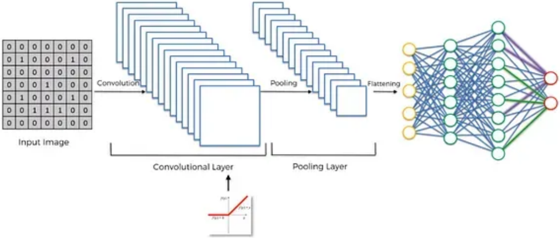
hoWeights[][]

0.13	0.14
~~0.15~~	~~0.16~~
~~0.17~~	~~0.18~~
0.19	0.20

Fully connected layer

- After the convolutional and pooling layers have extracted features from the input image, the fully connected layer can then be used to combine those features and make a final prediction.
- In a CNN, the fully connected layer is usually the final layer and is used to produce the output predictions.
- The activations from the previous layers are flattened and passed as inputs to the fully connected layer, which performs a weighted sum of the inputs and applies an activation function to produce the final output.

Fully connected layer



0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0