Neural Networks for Signal Processing-I

Prof. Shayan Srinivasa Garani

Department of Electronic System Engineering

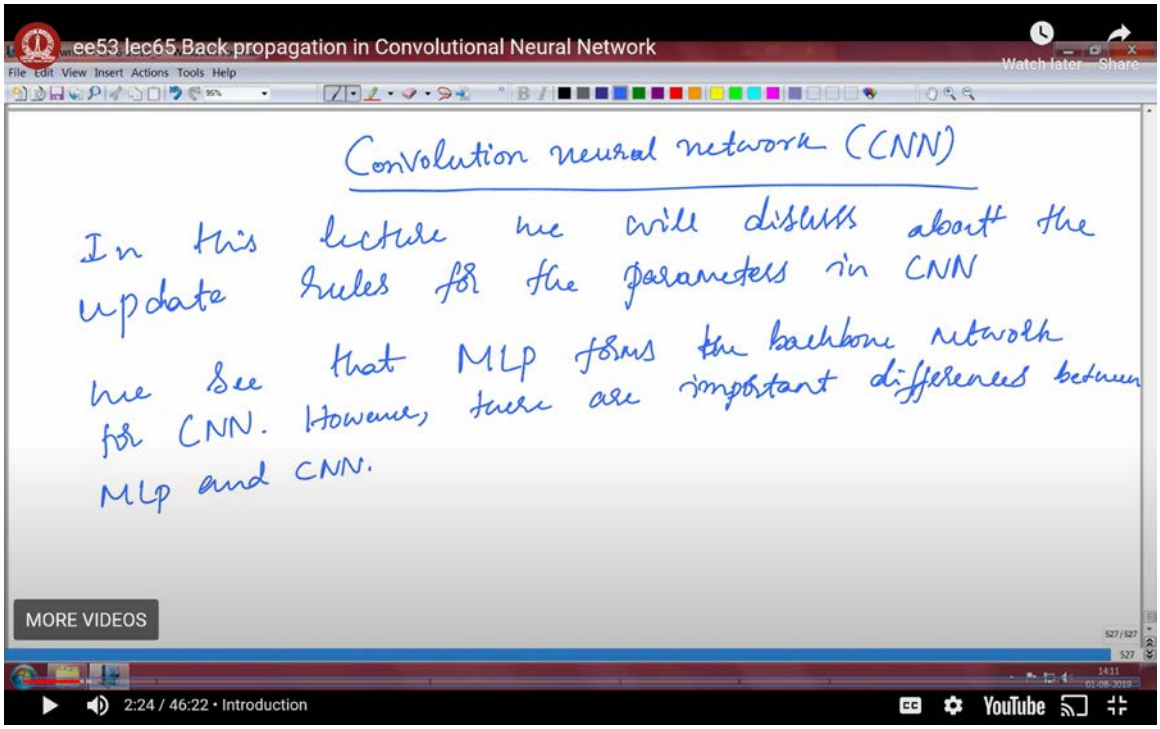
Indian Institute of Science, Bengaluru

Lecture – 65

Back Propagation in Convolutional Neural Network

In this lecture, we will delve into the update rules for parameters in Convolutional Neural Networks, often abbreviated as CNNs. We have previously explored the concepts of Multilayer Perceptrons (MLPs) in detail, including their foundational principles and the derivation of update rules for their parameters. It is important to note that while MLPs serve as the foundational architecture for CNNs, there are key distinctions between the two.

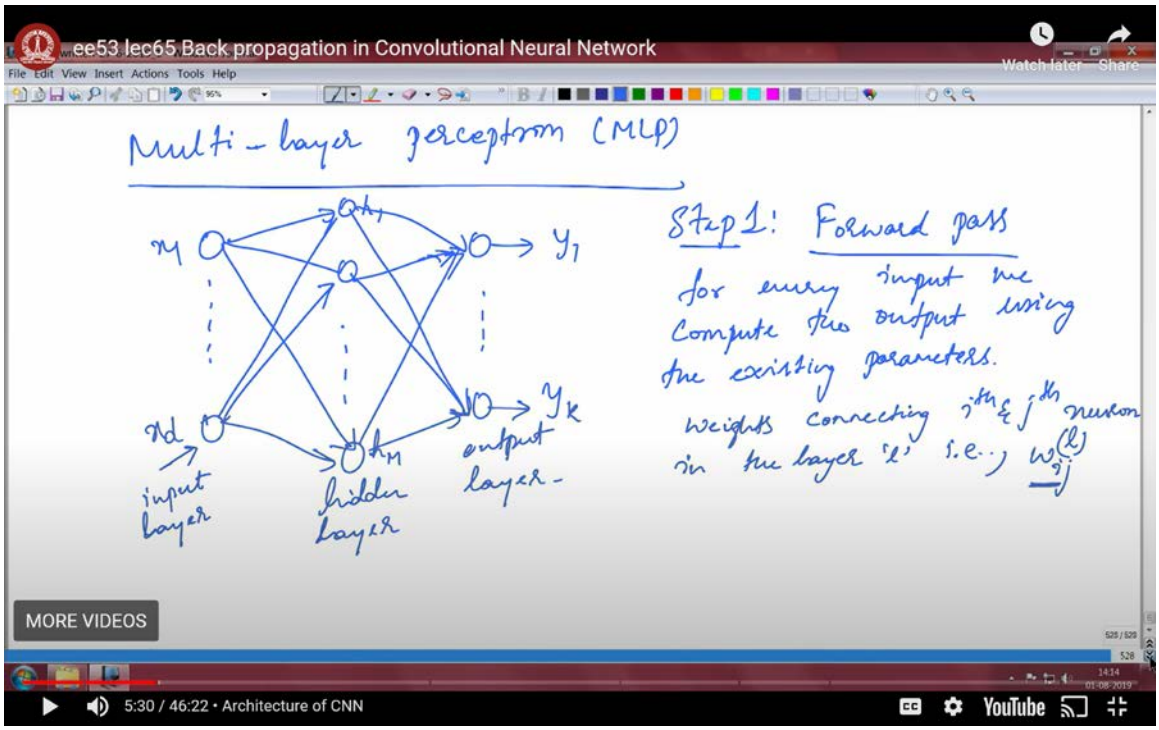
(Refer Slide Time: 02:24)



To set the stage, we will first revisit the MLP briefly, and then we will proceed to the specifics of deriving update rules for CNN parameters.

In a Multilayer Perceptron, the network consists of an input layer, one or more hidden layers, and an output layer. Suppose our input comes from a d-dimensional space, and we have M hidden neurons and k output neurons. An MLP is characterized as a fully connected network, meaning every neuron in one layer is connected to every neuron in the subsequent layer.

(Refer Slide Time: 05:30)



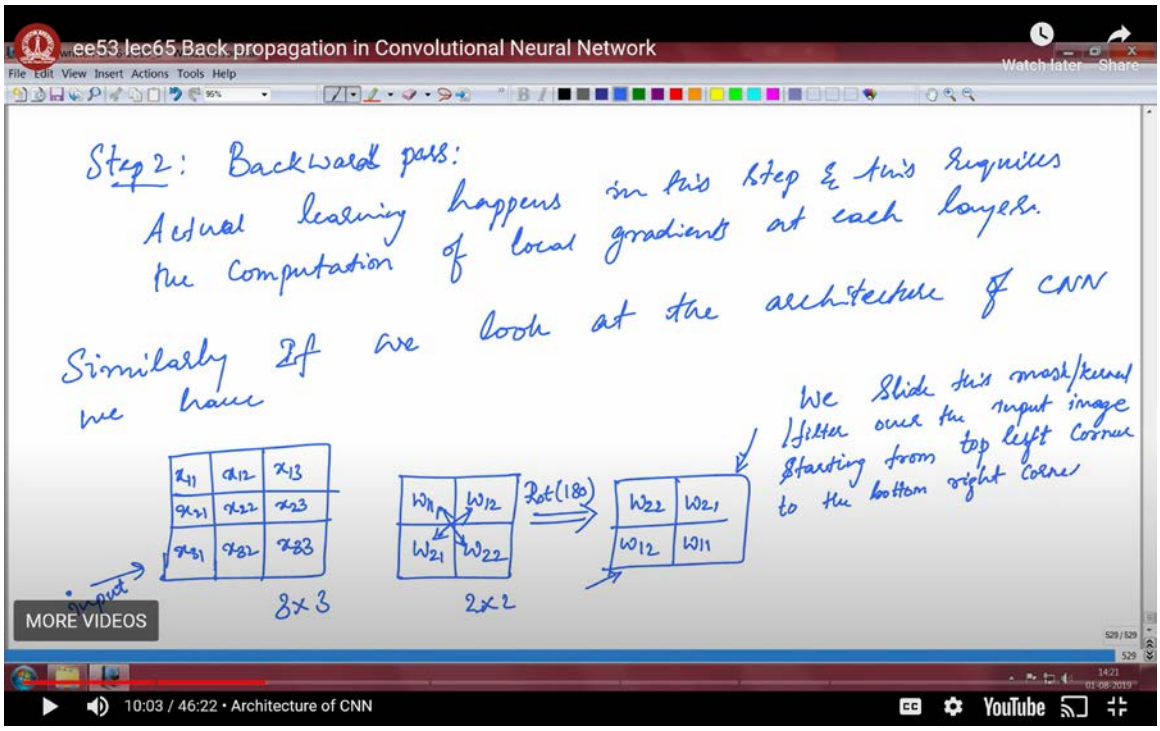
The MLP algorithm operates in two main steps. The first step is the forward pass. During this step, we compute the output for each input using the current set of parameters, which are the weights connecting the i-th neuron to the j-th neuron in layer L, denoted as W_{ij}^{L}. These weights are used to compute the output of the network.

The actual learning occurs in the second step, known as the backward pass. In this phase, we compute the local gradients at each layer, which is crucial for updating the network's parameters based on the error.

Since MLPs form the backbone of CNNs, it’s useful to highlight the similarities and differences between the two. Let’s now examine the architecture of CNNs.

In a Convolutional Neural Network, consider an example where the input is an image of size 3 × 3, though in practice, it could be an image of any size n × n. For this example, let's denote the pixel values of this 3 × 3 image as x_{11}, x_{12}, x_{13}, and so forth. This image represents our input to the CNN.

(Refer Slide Time: 10:03)



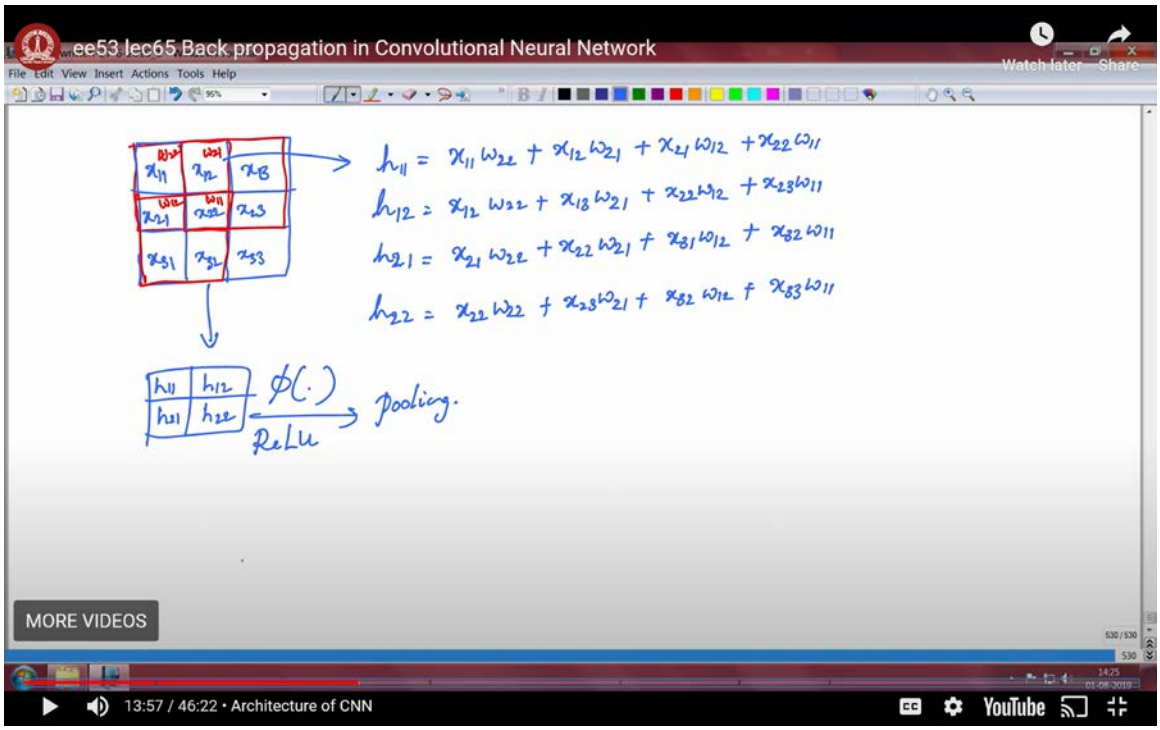
Let's consider a filter of size 2 × 2, though in general, it can be of size k × k where k < n. For this example, the parameters of the filter are W_{11}, W_{12}, W_{21}, and W_{22}. In a Convolutional Neural Network (CNN), this filter undergoes a rotation by 180 degrees. This means we exchange the positions of these elements to obtain W_{22}, W_{21}, W_{12}, and W_{11}.

After rotating the filter, we slide it over the image starting from the top-left corner to the bottom-right corner. For clarity, let's assume our input image has pixel values denoted as x_{11}, x_{12}, x_{21}, x_{22}, and so forth. We start by placing the rotated filter at the top-left corner of the image.

At this position, we calculate a value H_{11} using the following formula:

$$H_{11} = x_{11} \cdot W_{22} + x_{12} \cdot W_{21} + x_{21} \cdot W_{12} + x_{22} \cdot W_{11}.$$

(Refer Slide Time: 13:57)



We then slide the filter to the next position and compute a new value H_{12}:

$$H_{12} = x_{12} \cdot W_{22} + x_{13} \cdot W_{21} + x_{22} \cdot W_{12} + x_{23} \cdot W_{11}.$$

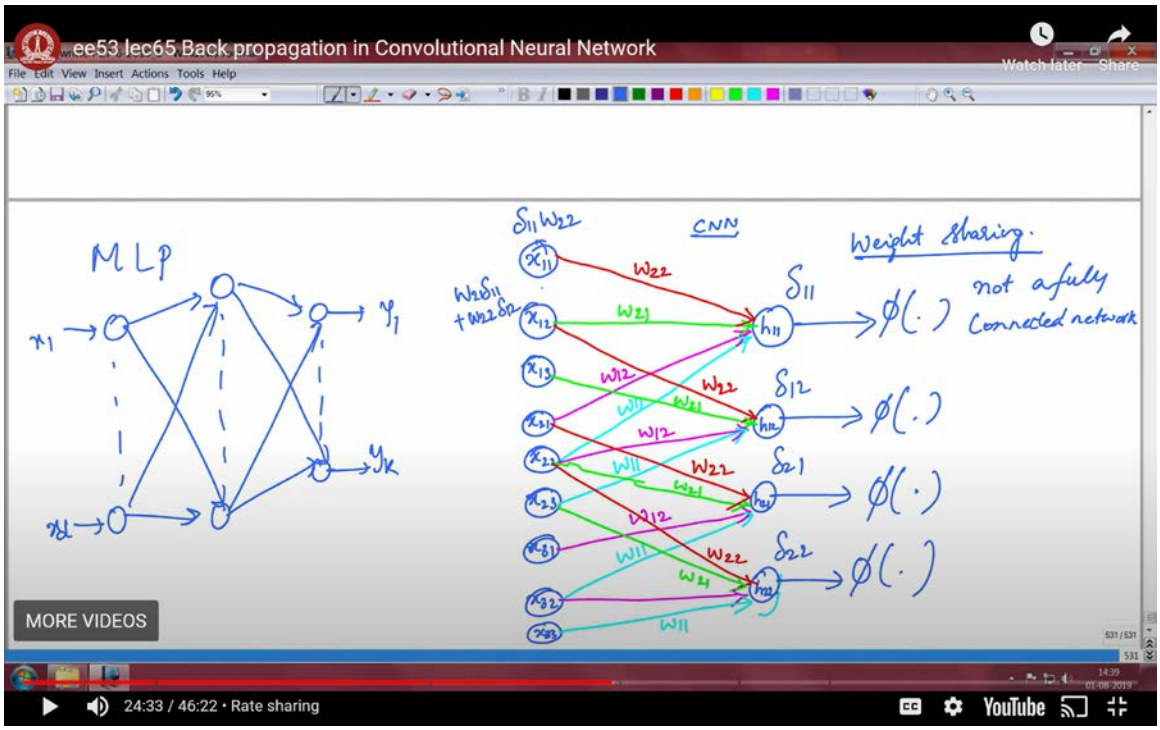
Continuing this process, we get:

$$H_{21} = x_{21} \cdot W_{22} + x_{22} \cdot W_{21} + x_{31} \cdot W_{12} + x_{32} \cdot W_{11},$$

$$H_{22} = x_{22} \cdot W_{22} + x_{23} \cdot W_{21} + x_{32} \cdot W_{12} + x_{33} \cdot W_{11}.$$

These computations result in four different values: H_{11}, H_{12}, H_{21}, and H_{22}. The next step in the CNN architecture is to apply an activation function, typically the ReLU function, to these values. After activation, pooling operations are usually performed. This process constitutes one block of the CNN, and this block is repeated multiple times depending on the specific application or computational resources available.

(Refer Slide Time: 24:33)

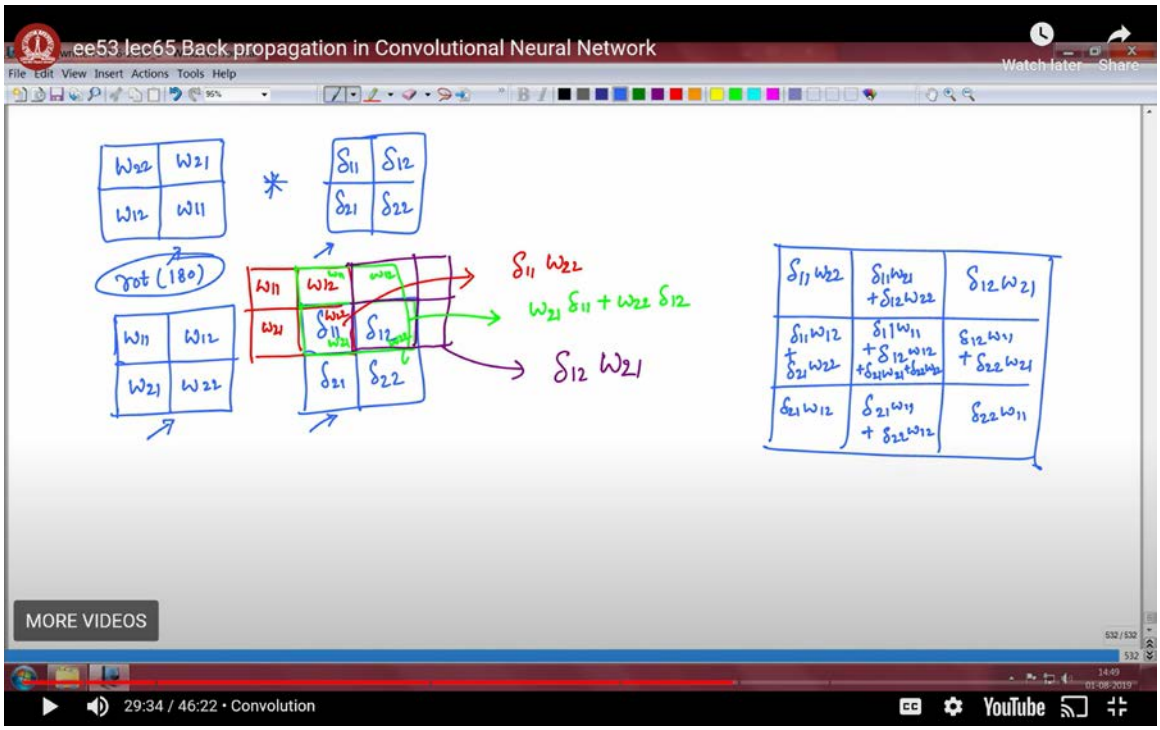


Now, let’s compare the CNN architecture with that of a Multilayer Perceptron (MLP). In an MLP, we have an input layer, hidden layers, and an output layer, with every neuron fully connected to every neuron in adjacent layers. In contrast, a CNN is partially connected, and a key feature is weight sharing, where the same filter weights are applied across different regions of the image.

To illustrate these differences, let’s examine the set of equations provided. We have four terms: H_{11}, H_{12}, H_{21}, and H_{22}. By analyzing these terms and their associated filter connections, we can understand how CNNs leverage parameter sharing and local connectivity differently from MLPs. For instance, to compute H_{11}, we use x_{11} weighted by W_{22}, demonstrating the localized application of weights in the CNN architecture.

To compute H_{11}, we need to consider the following contributions: x_{12} weighted by W_{21}, x_{21} weighted by W_{12}, and x_{22} weighted by W_{11}. These connections illustrate how inputs are combined with different parameters to generate the values for the next layer.

(Refer Slide Time: 29:34)



Now, let's examine how to obtain H_{12}. Referring to the equation for H_{12}, we need the inputs x_{12}, x_{13}, x_{22}, and x_{23}. Specifically:

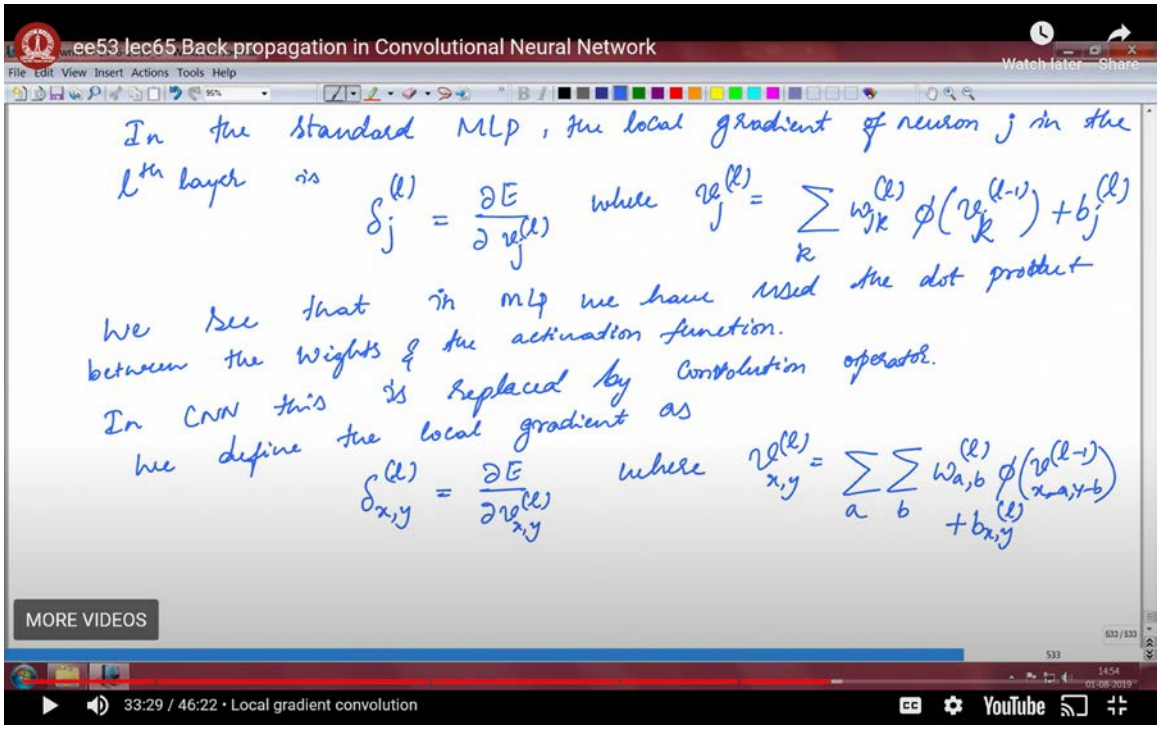
- x_{12} is weighted by W_{22},
- x_{13} is weighted by W_{21},
- x_{22} is weighted by W_{12},
- x_{23} is weighted by W_{11}.

If we compare the connections for x_{11} to H_{11} and x_{12} to H_{12}, we notice that the same parameters are used, which illustrates the concept of weight sharing. Similarly, we continue this process for other connections:

- For H_{21}, we use x_{21} weighted by W_{22}, x_{22} weighted by W_{21}, x_{31} weighted by W_{12}, and x_{32} weighted by W_{11}.
- For H_{22}, we use x_{22} weighted by W_{22}, x_{23} weighted by W_{21}, x_{32} weighted by W_{12}, and x_{33} weighted by W_{11}.

These computations complete the connection between the input and the next layer in the CNN. After obtaining these values, activation functions are applied to introduce non-linearity.

(Refer Slide Time: 33:29)



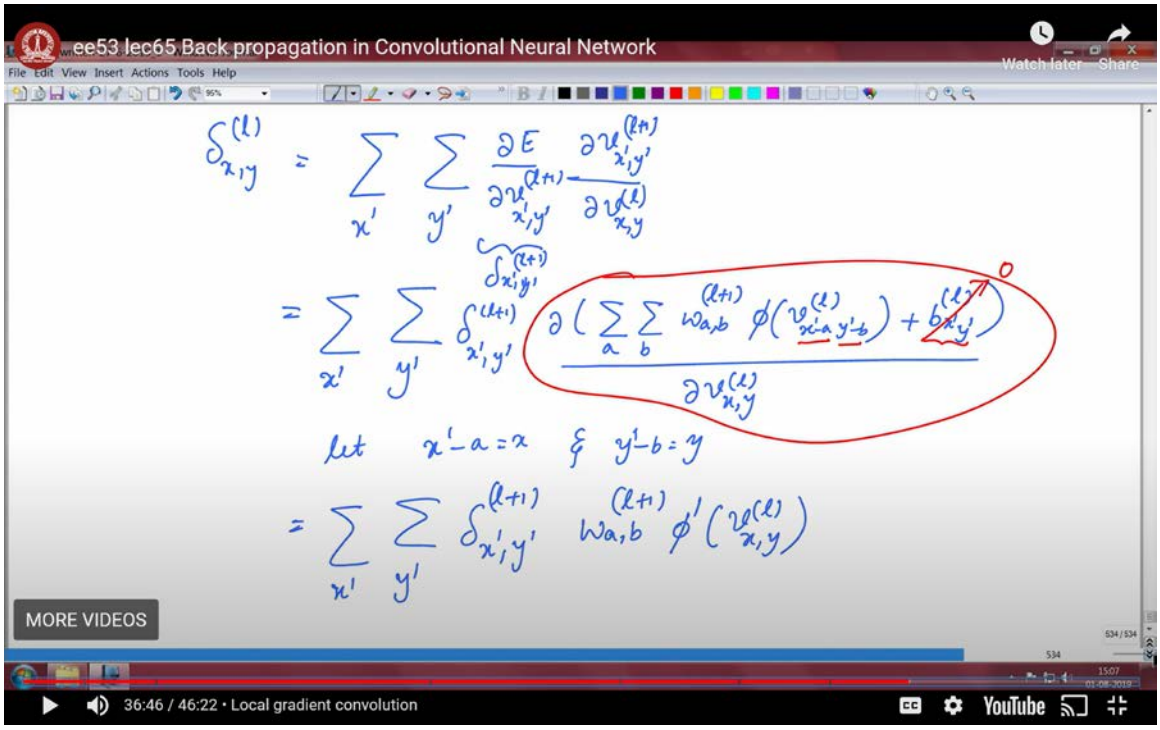
One crucial observation is that a CNN is not a fully connected network. In the forward pass, we compute the H_{ij} values, apply activation functions, and then perform pooling, iterating as needed.

Moving on to the backward pass, we need to update the weight parameters based on local gradients. Similar to what we did in a Multilayer Perceptron (MLP), we compute local gradients at each node. Suppose we have local gradients δ_{11}, δ_{12}, δ_{21}, and δ_{22} at the nodes.

To compute the local gradient at a node, such as x_{11}, we use the local gradient from the previous layer weighted by the parameter connecting x_{11} to H_{11}, which is $\delta_{11} \cdot W_{22}$. For the node x_{12}, the local gradient would be calculated as the sum of contributions from connections H_{11} and H_{12}, giving us $\delta_{11} \cdot W_{21} + \delta_{12} \cdot W_{22}$. Similarly, for x_{13}, the local gradient is $\delta_{12} \cdot W_{21}$.

This method of calculating local gradients can be generalized for other nodes and layers. By applying these gradients to the rotated filter, we update the filter parameters accordingly.

(Refer Slide Time: 36:46)



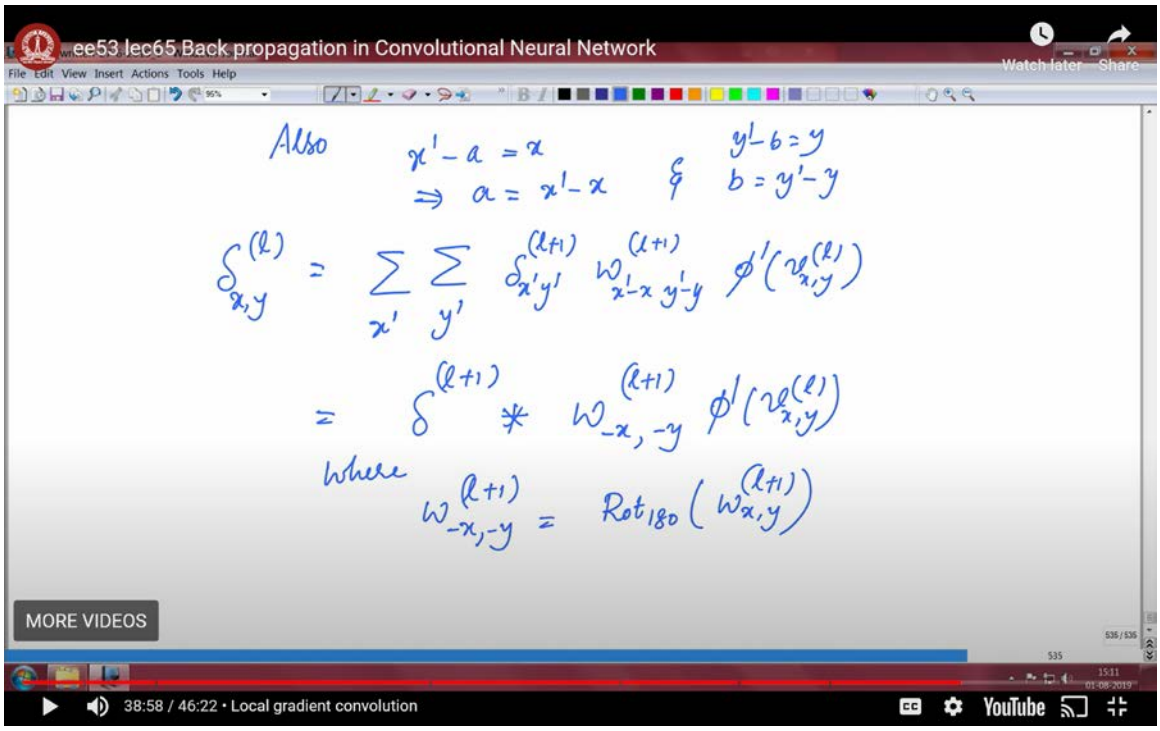
In convolutional neural networks, to compute the gradients effectively, we first perform a convolution operation. Here's a detailed explanation of the process:

During convolution, the mask (or filter) is rotated by 180 degrees, transforming it into W_{11}, W_{22}, W_{12}, and W_{21}. This rotated mask is then moved over the set of local gradients, starting from the top-left corner and progressing to the bottom-right. Let’s break this down step by step:

Initially, the filter is positioned at the top-left corner of the input. At this position, the output gradient at δ_{11} is computed as W_{22}. Next, the filter is shifted to the right. At this new position, the output gradient becomes $W_{21}\delta_{11} + W_{22}\delta_{12}$. Moving the filter further, we get $W_{12}\delta_{12}$, and similarly, the local gradients are calculated for each position, eventually

covering all values at each node. This method, involving the rotation of the filter, is a straightforward way to compute all local gradients at the input layer.

(Refer Slide Time: 38:58)



Now, let's delve into the mathematical framework behind this process. In a standard Multilayer Perceptron (MLP), the local gradient of neuron J in layer L is defined as:

$$\delta_J^{(L)} = \frac{\partial E}{\partial V_J^{(L)}}$$

where E represents the error, and $V_J^{(L)}$ is given by:

$$V_J^{(L)} = \sum_k W_{jk}^{(L)} \phi\left(V_k^{(L-1)}\right) + b_J^{(L)}$$

Here, $W_{jk}^{(L)}$ are the weights, φ is the activation function, and $b_J^{(L)}$ is the bias at layer L.

In contrast, in Convolutional Neural Networks (CNNs), this dot product operation is replaced by a convolution operator. The local gradient at a specific location (x, y) in layer L is defined as:

$$\delta_{(x,y)}^{(L)} = \frac{\partial E}{\partial V_{(x,y)}^{(L)}}$$

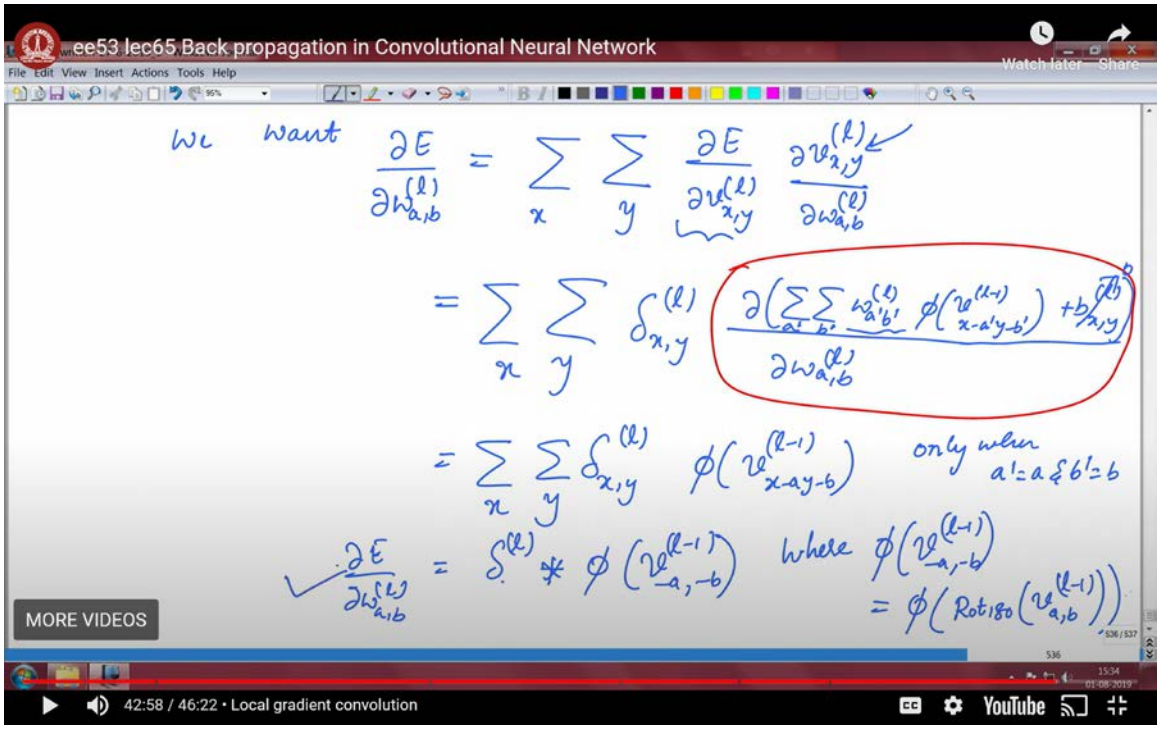
where $V_{(x,y)}^{(L)}$ is given by:

$$V_{(x,y)}^{(L)} = \sum_{a,b} W_{(a,b)}^{(L)} \phi\left(E_{x-a,y-b}^{(L-1)}\right) + b_{(x,y)}^{(L)}$$

Further simplification leads to:

$$\delta_{(x,y)}^{(L)} = \sum_{x',y'} \frac{\partial E}{\partial V_{(x',y')}^{(L+1)}}$$

(Refer Slide Time: 42:58)



This equation shows that we need to consider all local gradients from the previous layer that are connected to the node (x, y). This is why we use the chain rule and sum over all possible x' and y' from the previous layer. To simplify further, substitute x' - a with x and y' - b with y.

Thus, this convolutional approach leverages the rotation and shifting of the filter to calculate local gradients, which is crucial for updating the weights in CNNs.

Let's break down the process of updating parameters in Convolutional Neural Networks (CNNs) with a detailed explanation:

First, consider the derivative term of the activation function φ with respect to v at layer L. This derivative is non-zero only when x' - a = x and y' - b = y. In all other cases, the derivative is zero, making those terms constants. Additionally, the bias term is independent of x and y, so it contributes zero to the derivative.

Given these conditions, we can express the local gradient at layer L for location (x', y') as:

$$\delta^{(L)}_{(x',y')} = W^{(L+1)}_{(a,b)} \cdot \Phi'\left(V^{(L)}_{(x,y)}\right)$$

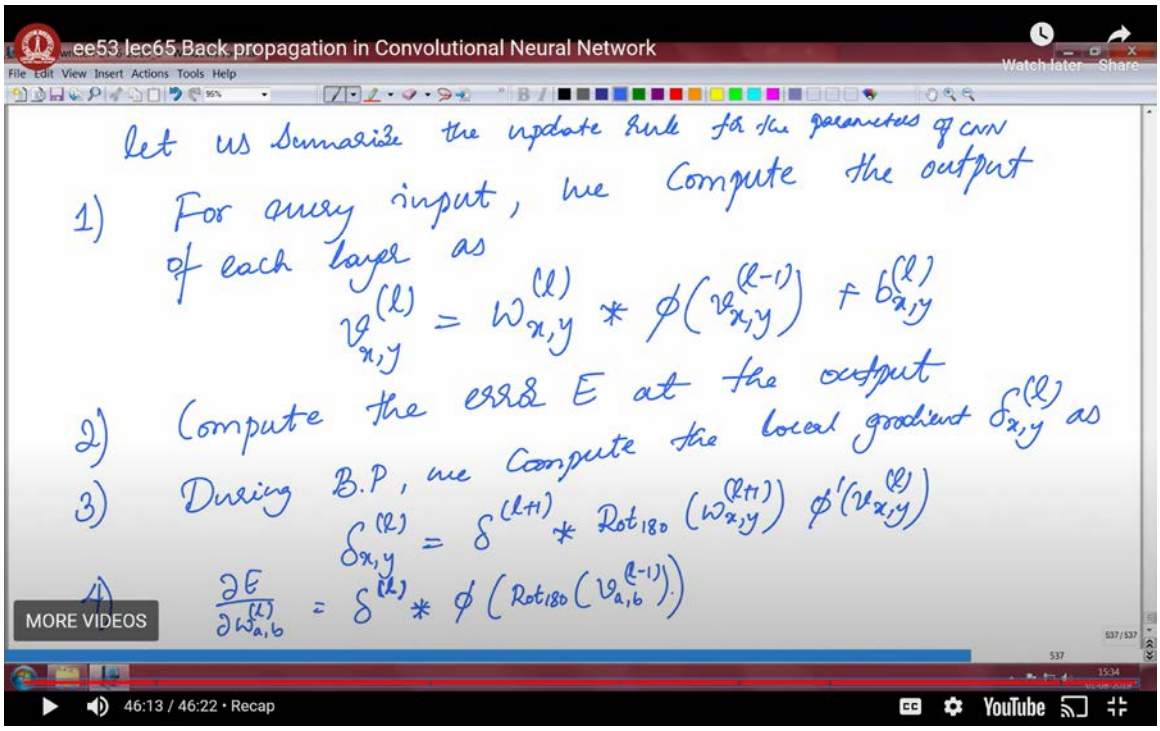
where a = x' - x and b = y' - y. Here, $W^{(L+1)}_{(a,b)}$ represents the filter weights after applying a 180-degree rotation. This simplifies to a convolution operation between the local gradient at layer L+1 and the rotated weights $W^{(L+1)}_{(-x,-y)}$.

Next, we need the gradient with respect to the weight parameters. The gradient $\frac{\partial E}{\partial W^{(L)}_{(a,b)}}$ is computed as:

$$\frac{\partial E}{\partial W^{(L)}_{(a,b)}} = \sum_{x,y} \delta^{(L)}_{(x,y)} \cdot \varphi\left(V^{(L-1)}_{(x-a,y-b)}\right)$$

Here, $\delta^{(L)}_{(x,y)}$ is the local gradient at layer L, and $\varphi\left(V^{(L-1)}_{(x-a,y-b)}\right)$ is the activation function from the previous layer. The bias term, being independent of a and b, does not contribute to this calculation.

(Refer Slide Time: 46:13)



To summarize the update procedure for CNN parameters:

1. Forward Pass: For every input, compute the output of each layer. This involves convolving the weight parameters with the activation functions from the previous layer and adding the bias term.

2. Error Computation: Calculate the error E at the output layer.

3. Backward Pass: Compute the local gradients $\delta_{(x,y)}^{(L)}$ at each layer. This is achieved by convolving the local gradient from layer L+1 with the 180-degree rotated weights and multiplying by the derivative of the activation function at layer L.

4. Parameter Update: Use the computed gradients to update the weight parameters. This involves convolution between the local gradients and the rotated activation function from the previous layer.

These four steps encapsulate the process needed to update the weight parameters in a convolutional neural network. Thank you.