Neural Networks for Signal Processing-I

Prof. Shayan Srinivasa Garani

Department of Electronic System Engineering

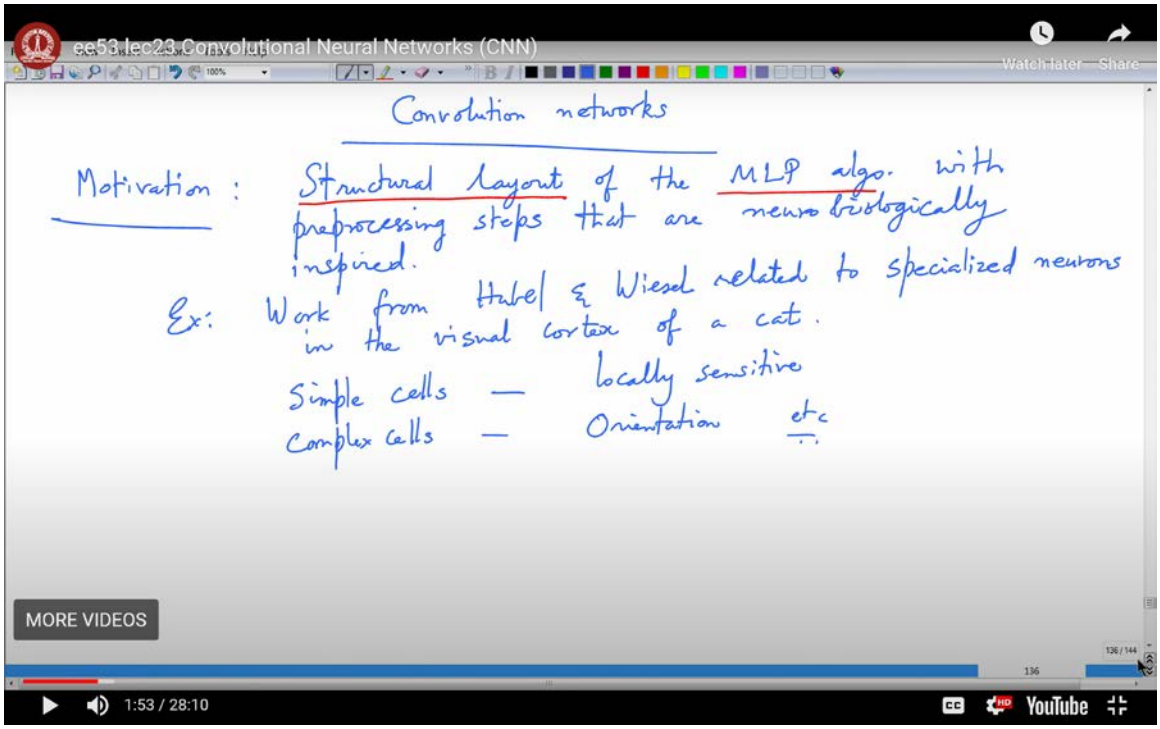
Indian Institute of Science, Bengaluru

Lecture – 23

Convolutional Neural Networks (CNN)

Having explored the multilayer perceptron and delved into the nuances of the backpropagation algorithm, including the intricacies of Jacobians and Hessians, let us now turn our attention to a pivotal architecture in deep learning: the convolutional neural network (CNN). This architecture is fundamental to many deep learning algorithms.

(Refer Slide Time: 01:53)



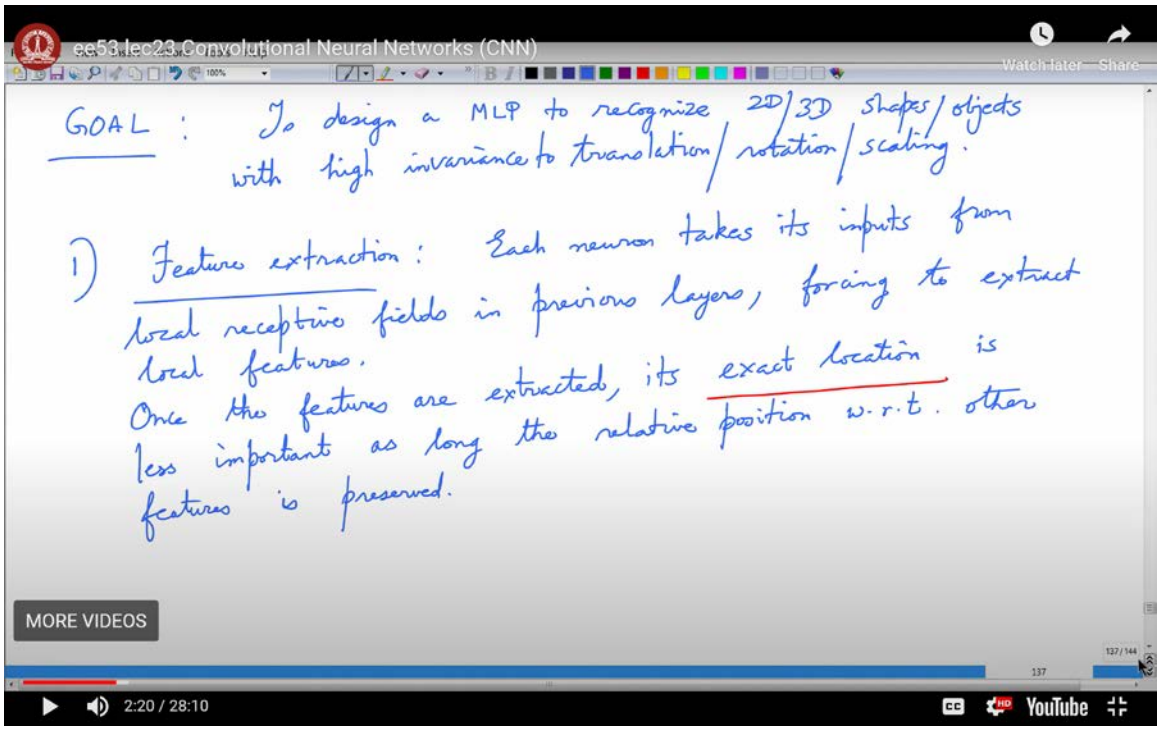
So, what drives the development of convolutional neural networks? The motivation behind CNNs is rooted in neurobiology. Specifically, the structural design of CNNs is inspired by preprocessing mechanisms observed in biological systems. We draw on the pioneering

work of Hubel and Wiesel, who studied the visual cortex of cats. Their research identified specialized neurons known as simple cells and complex cells.

Simple cells are locally sensitive, responding to small, specific features in their receptive fields. In contrast, complex cells are capable of detecting more advanced features, such as orientation and motion, due to their more sophisticated processing capabilities. The question then arises: can we construct a neural network that emulates this neurobiological functionality?

With this objective in mind, we aim to design a multilayer perceptron that excels in recognizing 2D and 3D shapes and objects, while maintaining high invariance to transformations such as translation, rotation, scaling, and other affine variations. This approach aligns with our goal of building a network that mirrors the advanced features of biological vision systems.

(Refer Slide Time: 02:20)

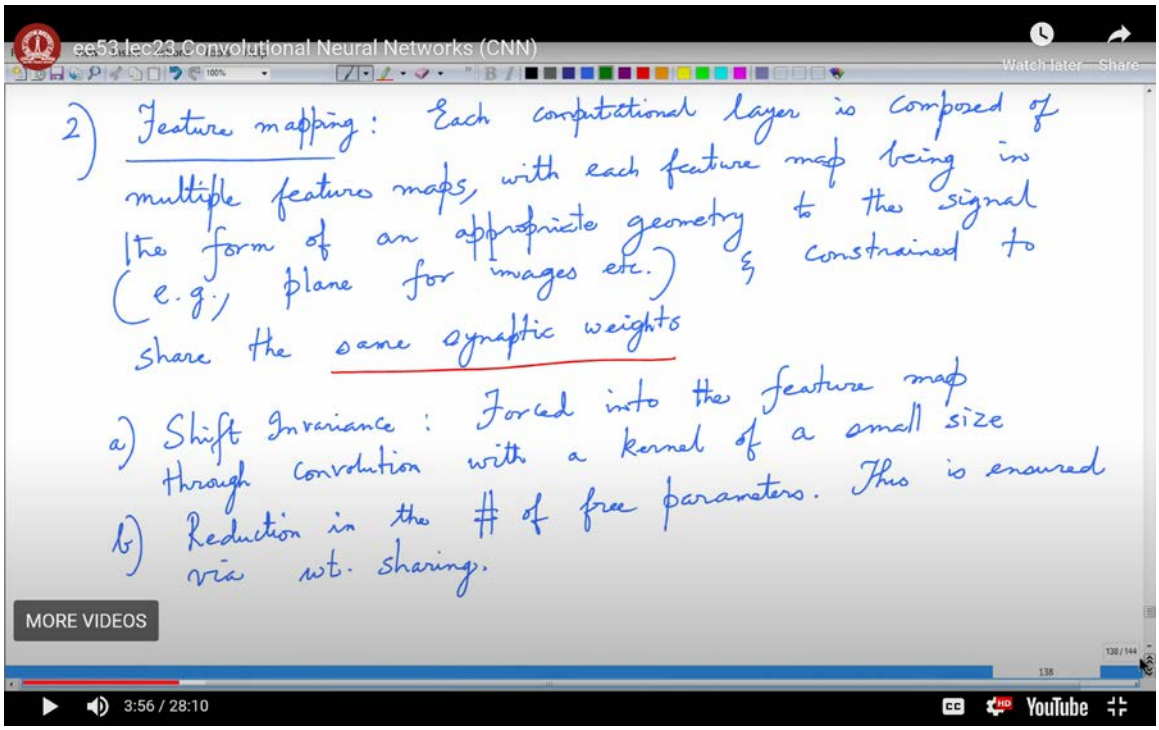


The idea behind convolutional neural networks (CNNs) was remarkably advanced by LeCun and his colleagues, who pioneered this approach. We will explore this concept in

greater detail in the upcoming slides. The first crucial step in CNNs is feature extraction. Each neuron in a convolutional layer takes inputs from local receptive fields in the previous layers, which forces it to extract local features from the input image or 1D signal. For instance, this involves applying a mask, such as a filter or kernel, across the image or signal to extract these local features.

Once these features are extracted, their exact location becomes less important than their relative position to other features. This means that while the precise location of a feature within the image may vary, the network focuses on maintaining the relative positions of features to each other, which is crucial for effective pattern recognition.

(Refer Slide Time: 03:56)

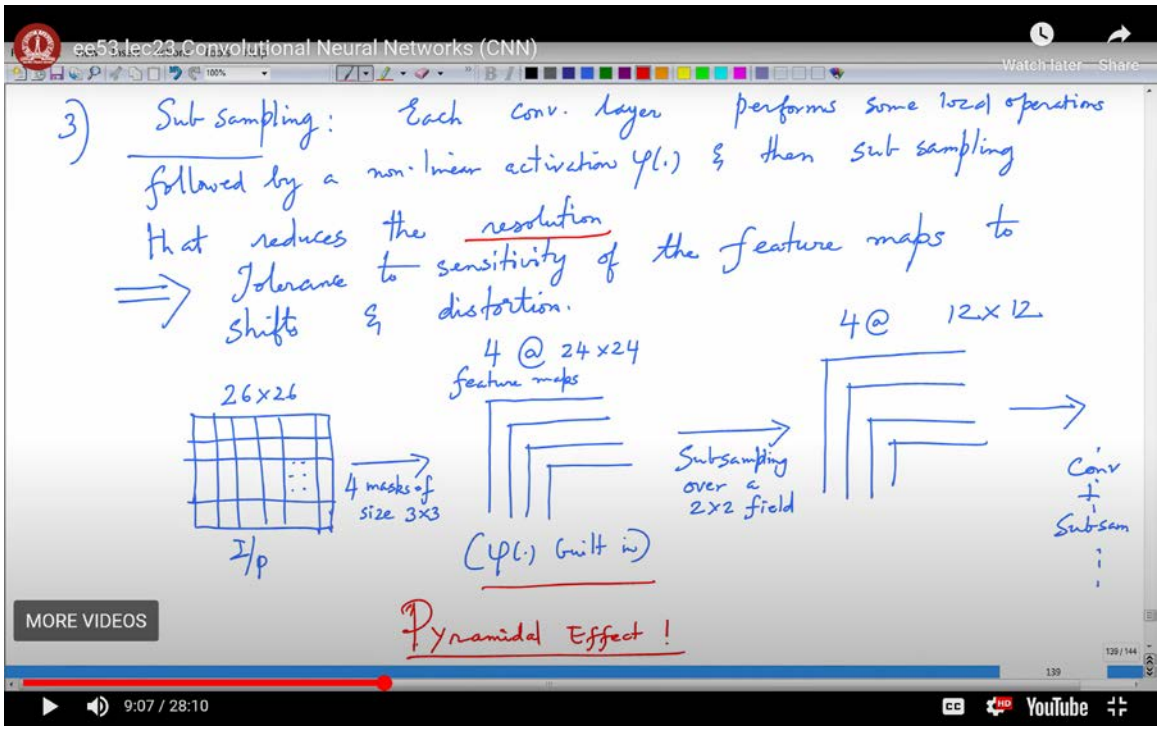


Following feature extraction, we proceed to feature mapping. Each computational layer in a CNN consists of multiple feature maps, each corresponding to a particular geometry suitable for the signal being processed. For example, feature maps for images might be two-dimensional planes, while those for 1D signals could be one-dimensional arrays. These feature maps share the same synaptic weights, which leads to two key benefits:

1. Shift Invariance: The convolution operation inherently incorporates shift invariance. By using a kernel of smaller size and applying it across the input, the CNN achieves shift invariance naturally. This means the network can recognize features regardless of their position in the input.

2. Reduction in Free Parameters: Weight sharing across connections in feature maps reduces the number of free parameters significantly. Unlike fully connected networks where weights are not shared and thus more numerous, CNNs use shared weights, which leads to a more efficient model with fewer parameters.

(Refer Slide Time: 09:07)

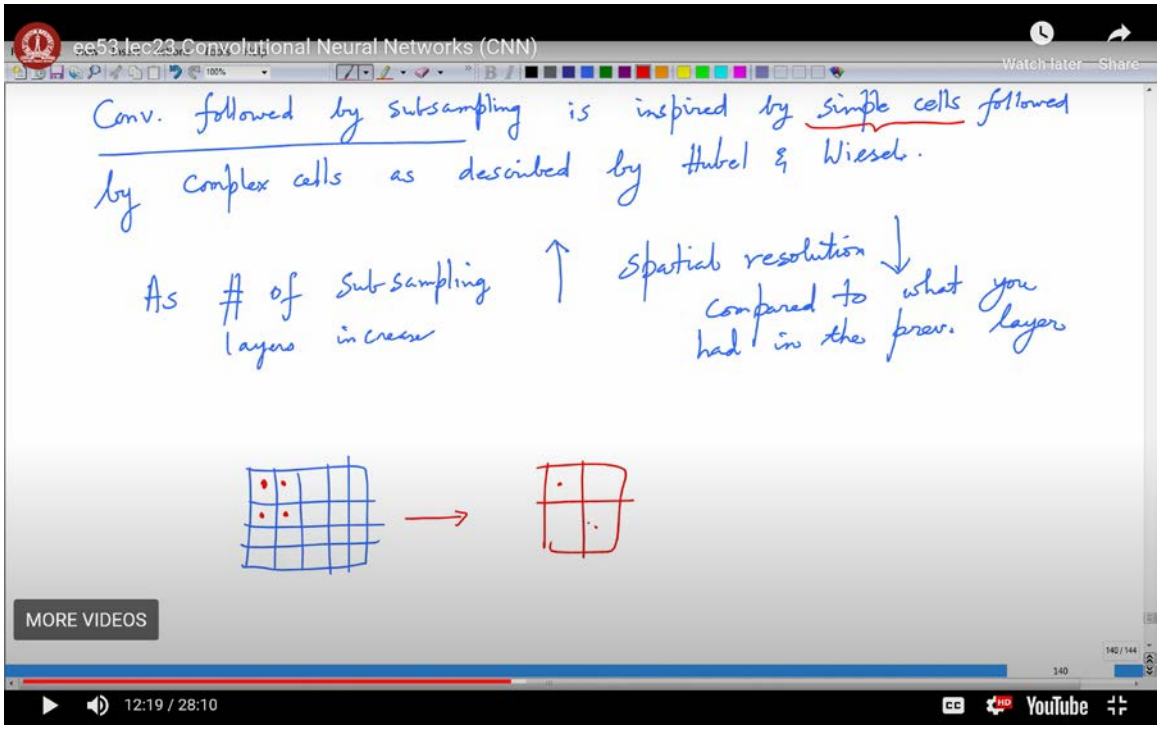


The next step in the CNN process is subsampling. After the convolutional layers have performed their local filtering operations, the output is subjected to a non-linear activation function, denoted as φ. This is followed by a subsampling process, such as max pooling, which further reduces the dimensionality of the feature maps while retaining the essential information.

Subsampling leads to a reduction in the resolution of the signal, which becomes apparent when we examine examples of how this resolution diminishes. These operations confer a degree of tolerance to shifts and distortions in the feature maps, enhancing the network's robustness. Let's explore how this architecture functions.

Imagine we start with a 26 by 26 input image or array. Suppose we use 4 masks, each of size 3 by 3. By applying these masks, we can extract features, resulting in 4 feature maps, each of size 24 by 24. This follows from the convolution process, where the output size is calculated as (26 - 3 + 1) = 24. After the convolution, a non-linear activation function, denoted as φ, is applied to each element of the feature maps. This non-linear function modifies the values within the feature map, providing a crucial aspect of the processing.

(Refer Slide Time: 12:19)

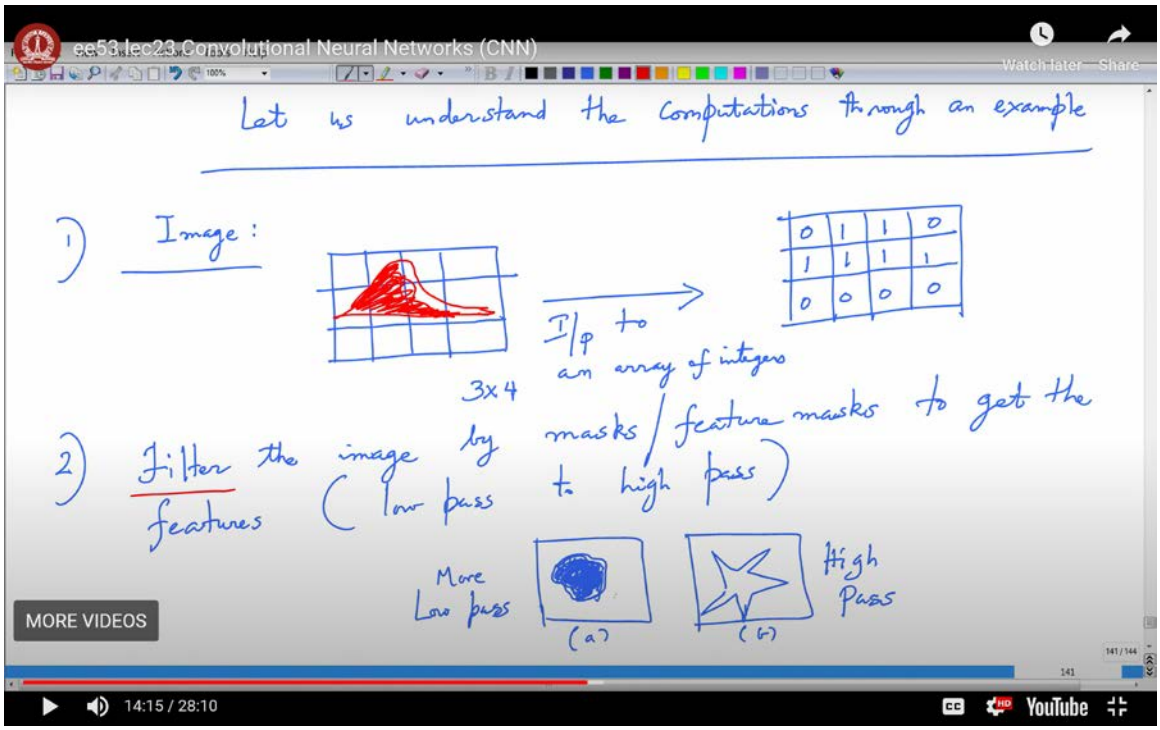


Following the convolution and non-linear activation, subsampling is performed to manage the large number of feature maps and their substantial sizes. For instance, if we subsample using a 2 by 2 field, and assuming we started with 4 feature maps, we end up with 4 new

feature maps, each of size 12 by 12. This process continues in a recursive, pyramidal fashion: convolution followed by subsampling, and so forth.

At this point, you might wonder about the choice of the masks: Should they be fixed, or should they adapt during training? This is a key consideration. In Hakin's book, feature maps are discussed as filters, and we can explore both fixed and adaptive approaches. Understanding these concepts will provide insight into the different architectural choices and their impacts.

(Refer Slide Time: 14:15)



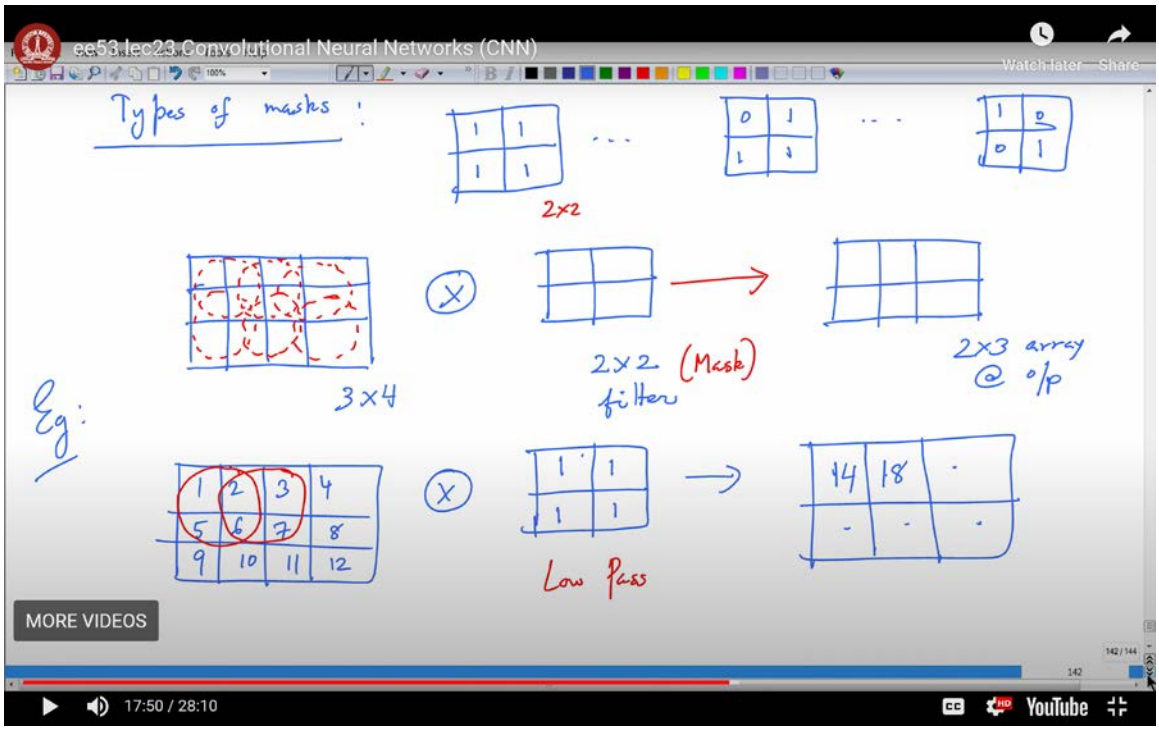
The idea of alternating between convolution and subsampling is inspired by neurobiological principles, reflecting how such processing strategies are modeled after the structure and function of the visual cortex.

As I mentioned at the beginning of this lecture, our goal is to develop a neural network architecture inspired by the pioneering work of Hubel and Wiesel, who identified simple cells and complex cells in the visual cortex. Simple cells are adept at detecting local

features, while complex cells are capable of recognizing more advanced features such as orientation. This architectural inspiration guides our design.

As we increase the number of subsampling layers, effectively descending through the hierarchy of convolution followed by subsampling, the spatial resolution of the feature maps decreases. In other words, the deeper we go into the network, the lower the spatial resolution compared to previous layers. This makes sense because, starting with an initial image, we first extract local features, represented by red dots in our example. As we apply subsampling, we reduce the number of sampled points, leading to a more abstract representation of the data.

(Refer Slide Time: 17:50)



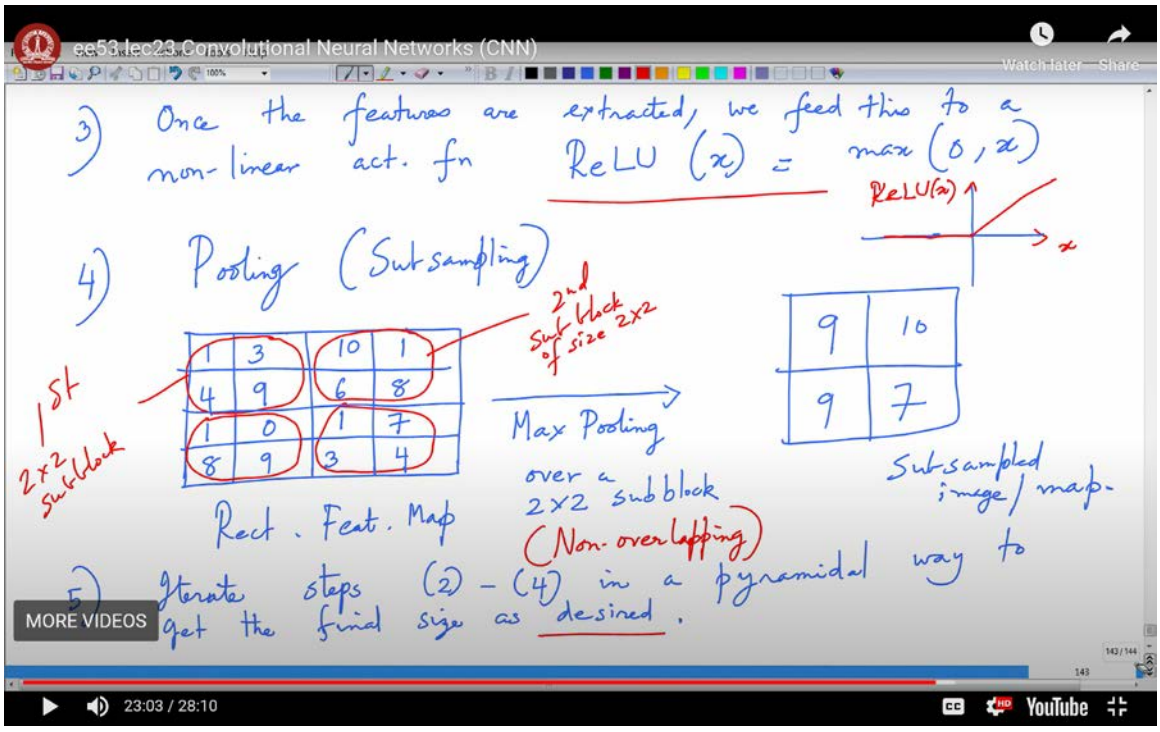
The reduction in spatial resolution due to subsampling helps manage the complexity of the feature set, making the network more efficient. Let’s explore this concept through computations. Consider an image represented as a 3 by 4 array of integers, where each integer corresponds to a pixel's value based on its bit precision (8-bit, 16-bit, etc.). This

image data, captured by a camera, is converted into integers according to the desired bit depth.

We then apply a filter to this image, which could be either adaptive or non-adaptive. For simplicity, let's start with a non-adaptive filter. A non-adaptive filter uses a predetermined mask to extract features from the image. The role of this mask is to detect specific types of information: low-pass filters capture content in the low-frequency range of the signal, while high-pass filters detect edges and other high-frequency features.

To visualize this, consider two images: Image A and Image B. Image A resembles a blue-shaded circle, which might be detected by a low-pass filter, while Image B might reveal edges and high-frequency details, identifiable by a high-pass filter. This example illustrates how different masks can be used to extract varying types of features from an image.

(Refer Slide Time: 23:03)



This content is predominantly low-pass, while the star shape with its rough edges represents high-pass information due to its pronounced edges. When we think about masks, consider a 2 by 2 mask. This type of mask typically performs a low-pass operation by

averaging values. For instance, when you slide this mask over an image, you're effectively applying a low-pass filter. Similarly, you can create band-pass or high-pass filters by adjusting which values in the mask are set to zero.

To illustrate, take a 2 by 2 mask and apply it to a 3 by 4 array representing an image. As you slide the mask across the image with overlapping blocks of pixels, you produce a 2 by 3 output array. For example, if the mask has values of 1 across all positions, applying it to a block of pixels such as [1, 2, 5, 6] involves performing a dot product with the mask. The result is 14. If the next block is [2, 3, 6, 7], the dot product yields 18. These operations are based on fixed coefficients in this example, but in practice, convolutional networks learn these coefficients through training. The actual values of the mask, denoted as w_i, are learned rather than fixed.

Once the features are extracted through convolution, they are fed into a non-linear activation function, which operates on each point in the array. One common activation function is the Rectified Linear Unit (ReLU). In our previous discussions on backpropagation, we covered the sigmoid and hyperbolic tangent activation functions. However, these functions can suffer from the vanishing gradient problem, which is why newer methods like ReLU are often preferred.

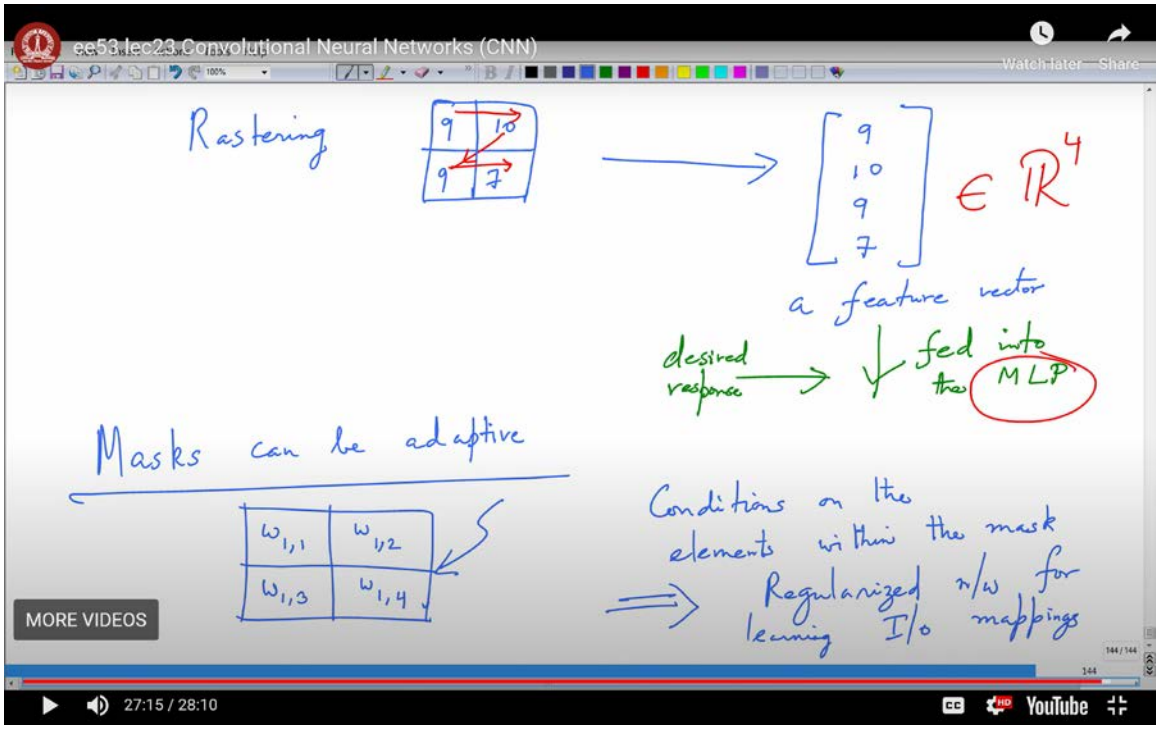
What exactly is the vanishing gradient problem? To understand this, consider the process of computing local gradients, which involves calculating partial derivatives. Suppose you have four partial derivatives, each of which is 0.5. If you multiply these derivatives together, $0.5 \times 0.5 \times 0.5 \times 0.5$, you get a very small value. As you add more layers to your network, the number of such small values compounds, and the gradients can become exceedingly small. This is the vanishing gradient problem: the gradients diminish to the point where they no longer contribute to learning, which hinders the effectiveness of the learning algorithm.

To address this issue, we use the Rectified Linear Unit (ReLU). The ReLU activation function operates as follows: it outputs the maximum of 0 and the input value x. In other words, for any input x, ReLU returns $\max(0, x)$. This means that all negative values are set to 0, while positive values remain unchanged. The advantage of this is that when you

take the derivative of the ReLU function, it is either 0 or 1. At 0, while the derivative is not strictly defined, it is typically treated as 0 or simply ignored. For positive values, the derivative is 1. This prevents the problem of gradients shrinking to zero, which is why ReLU is widely used.

Now, let's discuss pooling or subsampling. After applying the ReLU activation function, we might end up with a feature map like this: the first row could be [1, 3, 10, 1] and the second row could be [4, 9, 6, 8]. Max pooling is typically done over non-overlapping 2 by 2 blocks. You might wonder why the pooling should be non-overlapping, especially since the convolution operation involves overlapping masks. Non-overlapping pooling simplifies the process and reduces computational redundancy by discarding certain samples, which are unnecessary.

(Refer Slide Time: 27:15)



Let’s illustrate this with an example. Consider a 4 by 4 array and apply max pooling with a 2 by 2 sub-block. For the first 2 by 2 block [1, 3, 4, 9], the maximum value is 9, which is recorded. For the next 2 by 2 block [10, 1, 8, 6], the maximum value is 10. This process is

repeated for all sub-blocks, resulting in a subsampled image. This pooling operation can be repeated in a pyramidal fashion, convolution followed by subsampling, to achieve the desired final size of the feature map.

The resolution ultimately depends on the accuracy you require and the computational resources you can afford. This is a key consideration. Once you have your final 2 by 2 array, you rasterize it to obtain a feature vector, which, in this example, is a vector in R^4.

This feature vector can then be fed into a multilayer perceptron (MLP) or other types of networks. There are two main approaches to consider:

1. Fixed Masks: You can use a set of fixed masks to extract feature maps. After performing convolution and subsampling, you end up with a feature vector that can be input into the MLP. This approach involves a predetermined set of masks for feature extraction, and the MLP will learn the input-output mappings based on these fixed masks.

2. Adaptive Masks: Alternatively, you can use adaptive masks, where the masks are parameterized by weights such as $W_{1,1}, W_{1,2}, W_{1,3}, W_{1,4}$. These weights are not fixed; instead, they are learned adaptively using the backpropagation algorithm within the convolutional network framework. This approach allows the feature maps to be adjusted through learning, providing more flexibility and potentially better performance.

In addition, you might question whether it's possible to impose regularity conditions on these adaptive masks. For instance, you could have multiple masks to control the features you want to extract more precisely. Regularization techniques can be applied to the coefficients of these weight vectors to ensure a more controlled and effective learning process.

In summary, you have two main options: using fixed masks or adaptive masks. The choice depends on your specific needs and constraints. Both approaches involve convolution followed by subsampling and rectification, and learning the weights can be handled similarly to how we derived the backpropagation algorithm. This concludes our lecture on convolutional neural networks, and in the upcoming sessions, we will demonstrate these concepts in practice.