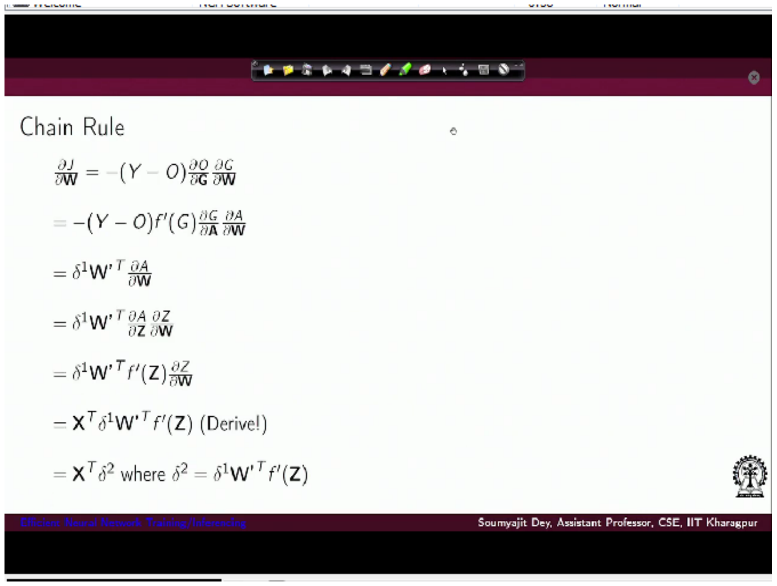
GPU Architectures and Programming

Prof. Soumyajit Dey
Department of Computer Science and Engineering
Indian Institute of Science Education and Research-Kharagpur

Lecture-57
Efficient Neural Network Training/Inferencing (Contd.)

Hi, welcome back to the lecture series on GPU architectures and programming.

(Refer Slide Time: 00:29)



Chain Rule

$$\frac{\partial J}{\partial \mathbf{W}} = -(Y - O)\frac{\partial O}{\partial G}\frac{\partial G}{\partial \mathbf{W}}$$

$$= -(Y - O)f'(G)\frac{\partial G}{\partial \mathbf{A}}\frac{\partial A}{\partial \mathbf{W}}$$

$$= \delta^1 \mathbf{W}'^T \frac{\partial A}{\partial \mathbf{W}}$$

$$= \delta^1 \mathbf{W}'^T \frac{\partial A}{\partial \mathbf{Z}}\frac{\partial Z}{\partial \mathbf{W}}$$

$$= \delta^1 \mathbf{W}'^T f'(\mathbf{Z})\frac{\partial Z}{\partial \mathbf{W}}$$

$$= \mathbf{X}^T \delta^1 \mathbf{W}'^T f'(\mathbf{Z}) \text{ (Derive!)}$$

$$= \mathbf{X}^T \delta^2 \text{ where } \delta^2 = \delta^1 \mathbf{W}'^T f'(\mathbf{Z})$$

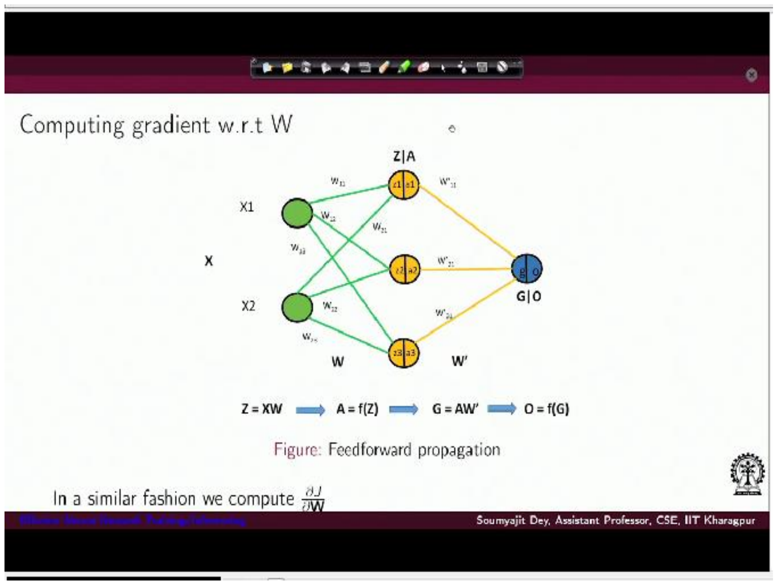
Efficient Neural Network Training/Inferencing

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So we will just start with a small (()) (00:30) of our coverage in the last lecture. So, one was regarding this chain rule. So, if you just check, we have been earlier talking about this y's in vector notation, but since we are having a summation here that would mean we are doing a summation over the components errors. So, that should essentially be small y i's and also continuing further.

One other issue would be like here in the chain rules derivative since we are considering capital Y's which are the vectors. So the summation would really not be there, right.

(Refer Slide Time: 01:06)



Computing gradient w.r.t W

$$Z = XW \rightarrow A = f(Z) \rightarrow G = AW' \rightarrow O = f(G)$$

Figure: Feedforward propagation

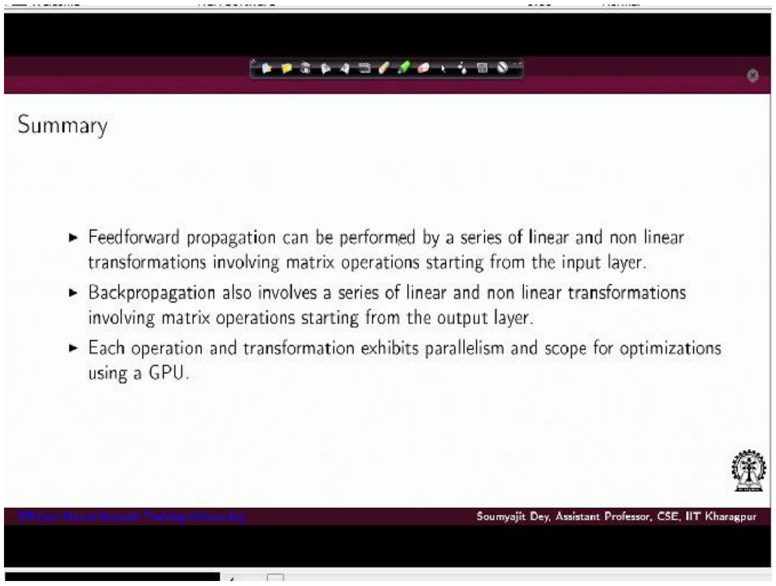
In a similar fashion we compute $\frac{\partial J}{\partial W}$

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

Why so if we remember that we have already discussed how we do the forward pass, and how are the derivatives related, how to compute the derivative of the loss function with respect to the weight matrices w and w prime, and how those are really used to figure out this value of the derivative that is in the second stage, it is this X transpose delta 2 where I can also have the delta 2 expressed as the value of delta 1.

And W prime transpose and then the activation functions gradient at those values, right. So, with this, I believe we had already discussed what was a backpropagation rule that in the backpropagation pass you re-compute W primes, so you re-evaluate using your computation of delta 1 and delta 2. You are essentially re-computing W prime to be W prime minus A transpose delta 1 and then you are also using delta 2 to figure out what will be the updated value of W. So, that would be W minus X transpose over delta 2 like that, right.

(Refer Slide Time: 02:10)

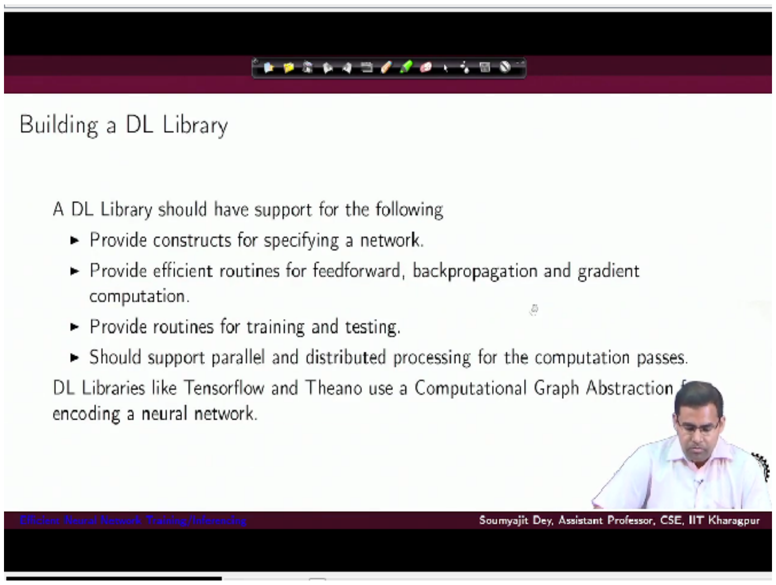

Summary

- Feedforward propagation can be performed by a series of linear and non linear transformations involving matrix operations starting from the input layer.
- Backpropagation also involves a series of linear and non linear transformations involving matrix operations starting from the output layer.
- Each operation and transformation exhibits parallelism and scope for optimizations using a GPU.

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, like this was our summary, that whenever you are doing the feedforward propagation, it is a series of linear and nonlinear transformations, essentially all in matrix operations. And during the backpropagation, you also have a series of such transformations. Again, that those can be I mean, but they start from the output layer towards the frontal layers. So, that is why it is backpropagation. And we want to do all those things using the parallelism that is available in the GPUs, right.

(Refer Slide Time: 02:37)


Building a DL Library

A DL Library should have support for the following

- Provide constructs for specifying a network.
- Provide efficient routines for feedforward, backpropagation and gradient computation.
- Provide routines for training and testing.
- Should support parallel and distributed processing for the computation passes.

DL Libraries like Tensorflow and Theano use a Computational Graph Abstraction for encoding a neural network.

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, with this background, in case you are interested in building a deep learning library, so that would mean your library should have to support the following things that means it should provide constructs like how somebody can easily specify the neural network architecture, what kind of efficient routines, I can implement as part of the library, so that I have feedforward back propagation and gradient computation these features available, right.

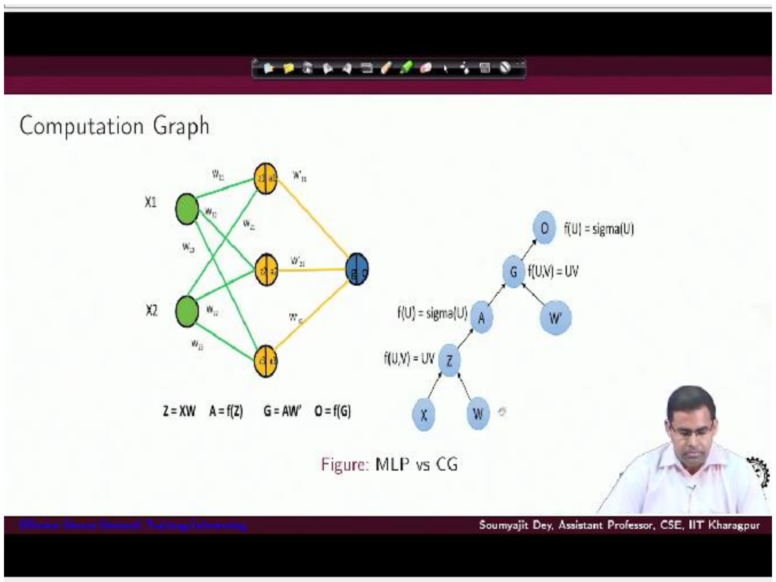
So they should just be available as function calls when I am building a complex neural network architecture and going to implement it for doing all these computations, right. So, of course, and using these routines, I will like to build high level routines for doing that neural network training as well as testing process right. Moreover, as we can understand that the library we like to exploit HPC resources.

So, all the operations have to be implemented, these operations of feedforward back propagation gradient computations, they have to be implemented using suitable data parallel languages like CUDA or open CL or something else. And it should really be configurable. That means once I tell that this library should be running in this kind of an architecture, it should be explored.

To figure out well, let us say I am trying to say that I want this library to run in a CPU GPU system, where the GPU has this nice is a sense this many espies and all that, then I should follow the standards we discussed earlier that how to really define the architecture, how to write your program in such a way that all the variables get configured based on the architecture or resources that are available right.

Something like this is already available for us. Many of us have already heard about this popular deep learning libraries like Tensorflow and theano. And they essentially follow all this I mean, standards and discussions we have just done. But at the high level, they provide an abstract view of the neural network, they provide what we call as a computational graph abstraction, which helps you to encode the different parameters and the connections that should be there in a very complex neural network architecture.

(Refer Slide Time: 04:56)



So we are just trying to provide an abstract view on the left hand side, suppose you have this multi layer perceptron like we have discussed earlier. So, you have the input that input X vector is getting transformed with W right and then you get something called this intermediate value that is a Z which gets transformed with the activation function, you get something called A, these are all intermediate values that you are computing.

Then again you have to do further transformation on A using these weights of W prime that gives you G and on this you apply the final activation function f again to get the output, right, like I mean, we have to understand that this is the neural network computation which this architecture will do. And we like to encode it in the form of a data flow graph. So, that is the usual way in which a computer scientist likes to encode his computation.

So, it is basically like you have notes which kind of represent the compute I mean input data or some transformation that has to be done right. And then what the output of the transformation is provided to another node, where some further transformation will be done, right. So as you can see, on the right hand side, we have an example computation graph where we have 2 nodes X and W which provide the input data.

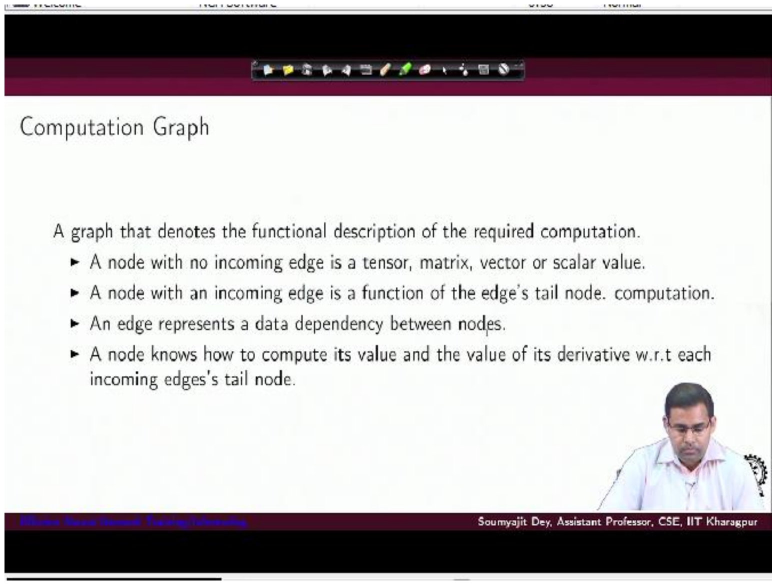
And these data flows following these edges to this node, where I have computed the value Z and this node is level like this, f of U, V is UV. So, essentially I am trying to say that this node performs the following transformation. That is, it takes the left and right operands and does a matrix multiplication, right, then the output will flow to another node I call that output as A right. So, this is dependencies further captured by this arrow from node Z to node A.

And here how do I really get A. So the computation that will provide me A is again, denoted here using this function, f of U. So I am just saying that the function takes whatever argument, let us call it U, the output is sigma of U, sigma is a activation function here, right. And then again, these output A and some other input node W prime, these 2 inputs flow to give me another output G.

And this output is derived by considering the corresponding function levelling, where I am saying that this function is f which takes again 2 parameters and does their multiplication. So again, A and W prime gets multiplied to give me G. And then again, I have the another activation function giving me the final output which is O. So the convention we are following here is every node has the data representation.

And following the dependency as data flows to an output node and the way I get output node is by a suitable function transformation and the function is provided as a level right. So, this is how we are trying to signify how the computation, how the data flows through this transformation network in the form of a computation graph. So, better to say is just another way to represent it, where we are making the dependencies explicit.

(Refer Slide Time: 08:27)



Computation Graph

A graph that denotes the functional description of the required computation.

- A node with no incoming edge is a tensor, matrix, vector or scalar value.
- A node with an incoming edge is a function of the edge's tail node. computation.
- An edge represents a data dependency between nodes.
- A node knows how to compute its value and the value of its derivative w.r.t each incoming edges's tail node.

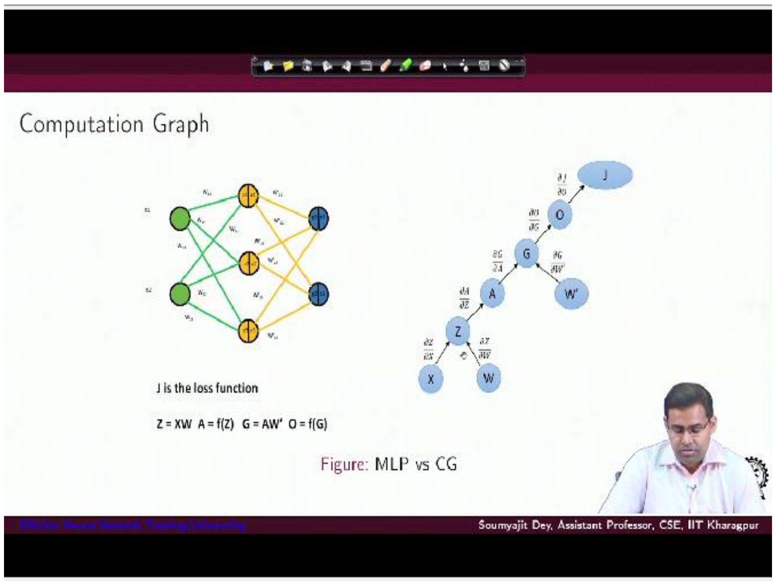
Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, just to summarize this is a graph that will denote the functional description that I required for the computation. And node which has got no incoming edge is a tensor or it can be a matrix or a vector or a scalar value. A node which has an incoming edge is a function of the edges tail node computation. The edge represents a data dependency between the nodes like we saw earlier, the edge just tells me that the data will flow in the direction of the edge.

And as long as the data is not available in the input nodes, the edges leading to some output node there, the computation is not going to happen, right. And node knows how to compute its value and the value of its derivative with respect to each incoming edges tail node. Now that is important. So every node knows how to compute this value. And that is known by the associated function.

Here this node knows that it has to apply this multiplication function, right. And also, it knows how to compute the value of its derivative with respect to each incoming edges tail node.

(Refer Slide Time: 09:36)



Computation Graph

J is the loss function

Z = XW A = f(Z) G = AW' O = f(G)

Figure: MLP vs CG

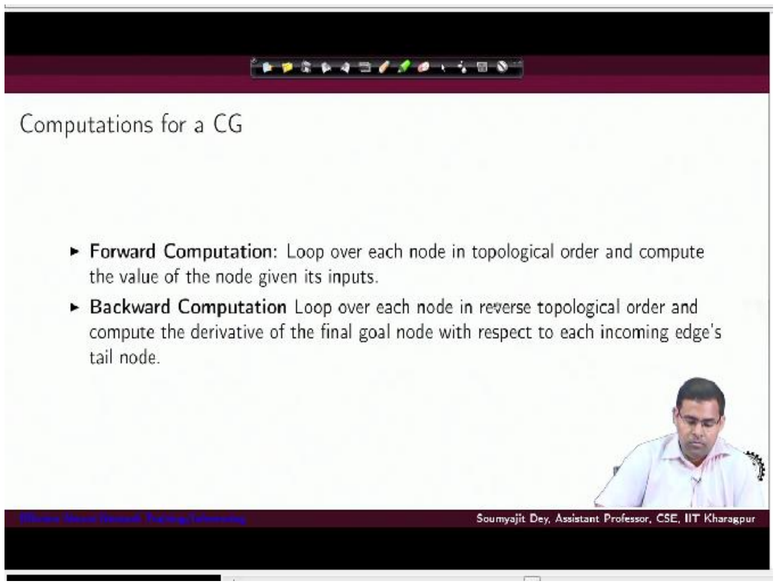
Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So if asked, this node can compute the derivative of Z with respect to W okay. Now, this is the next thing that we will use. So considering J as the loss function, again, let us recall that we have this computation going on. So X multiplied by W is going to give me Z. We then compute f of Z, which gives me A. We use A and W prime to generate G, then we compute I mean, apply f on G to get O.

And finally we using O, I compute the loss function, right. And as we have been discussing that every node knows, since the dependency of the node is always I can figure out the dependency by looking into the incoming edges. So for every node, I can compute its derivative with respect to the variables in these incoming edges, corresponding nodes, right. So, given Z, I can figure out this at dW are tracking backwards.

Let us say I want to compute some overall derivative, I can follow the derivatives chain rule as we know, and using the chain rule, I can compute each of the individual derivatives and compose them. So we will see an example. But before that, all I am trying to say here since I have this kind of a dependency, for every node, I can figure out let us say from J to O there is a dependency I can compute del J del O. Similarly, I can compute del O del G, del G del W prime, del G del A. del A del Z so on so forth.

(Refer Slide Time: 11:12)


Computations for a CG

- **Forward Computation:** Loop over each node in topological order and compute the value of the node given its inputs.
- **Backward Computation** Loop over each node in reverse topological order and compute the derivative of the final goal node with respect to each incoming edge's tail node.

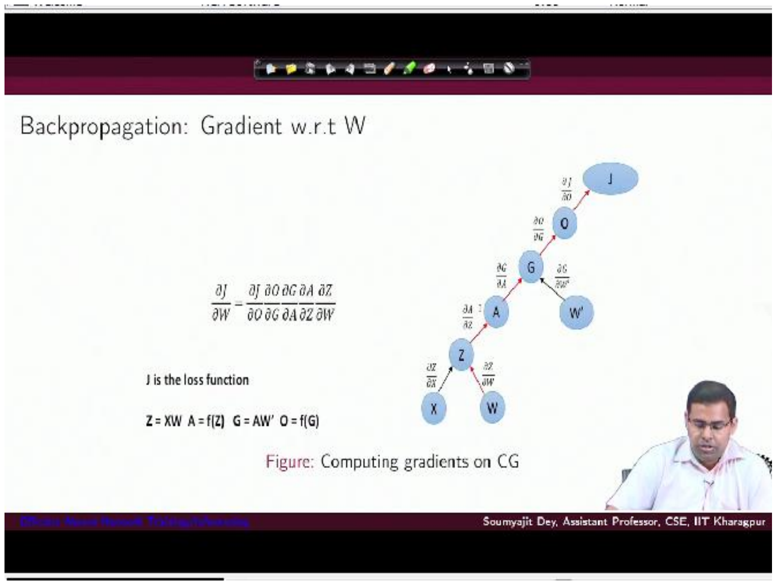
Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, whatever is in the forward computation for the computational graph, so, it simply loops over each node in the topological order and computes the value of the node given its inputs, so, as we know that the topological order is nothing but a linear representation of the node ordering of a directed acyclic graph right. So, essentially I can compute a topological order for any such graph where the node which is dependent on some previous node.

That dependent node will come later on in the order right. So, following the topological order, I can simply loop over the graph and compute the corresponding outputs. And then for backward computation, what am I supposed to do if you remember what we did in the neural network, you just went back from the output layer to the input layers right, essentially the same thing. So, you loop over the nodes in the reverse topological order.

And you want to computing the derivative the final node with respect to each incoming edges tail node. That means you do this computation, you compute del J del O, del O del G and like that. So, this is how I can do the forward and backward computation.

(Refer Slide Time: 12:27)



Backpropagation: Gradient w.r.t W

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial O}\frac{\partial O}{\partial G}\frac{\partial G}{\partial A}\frac{\partial A}{\partial Z}\frac{\partial Z}{\partial W}$$

J is the loss function

Z = XW A = f(Z) G = AW' O = f(G)

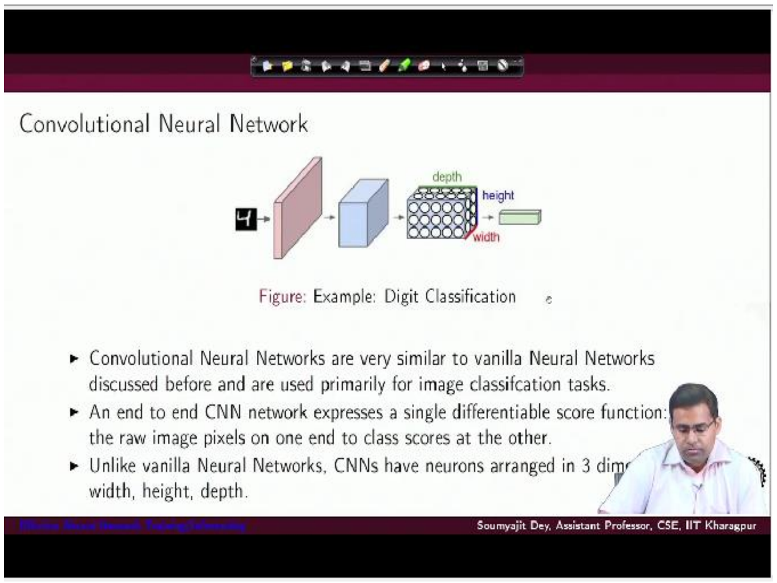
Figure: Computing gradients on CG

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

But how does that really help, our overall objective is to compute this right del J del W. So, as you can see, we have been discussing earlier that I can follow derivatives chain rule. And since each of these expressions would be available to me, because I have these dependencies for each node, I know their values and all that. So, I can just compute each of these individual derivatives and compose them to get what is the overall derivative of the loss function.

Similarly, I can also compute the derivative of the loss function with respect to W prime and that follows these red mark arrows right.

(Refer Slide Time: 13:05)



Convolutional Neural Network

Figure: Example: Digit Classification

- Convolutional Neural Networks are very similar to vanilla Neural Networks discussed before and are used primarily for image classifcation tasks.
- An end to end CNN network expresses a single differentiable score function: the raw image pixels on one end to class scores at the other.
- Unlike vanilla Neural Networks, CNNs have neurons arranged in 3 dim width, height, depth.

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, essentially this is how the computational graph abstraction is used to specify a neural network in for a corresponding deep learning library. And then given suitable implementation of the different propagation passes that will be available in the library, the computation really goes on, right. So, from this point, let us switch to another alternative neural network architecture, which is better known as convolution neural network.

Well, this is very similar to normal neural networks that we have already discussed, but they are kind of also specialized that their architecture is tuned towards image classification tasks. So, what is special about image classification tasks. Let us understand that an input image when we know that is an image, it always is going to I mean for a very regular structure, it is going to be a multidimensional matrix, where each of the pixel values are encoded for corresponding to different channel of colors right.

And given that an image has got certain of intrinsic properties, the convolution neural network makes use of those properties to decide these architectural primitives and end to end convolutional neural network will express a simple differentiable score function, it will essentially map the raw image pixels from the input to class course at the other. So the class code can be many things right I mean, so you are essentially let us say you are trying to I mean recognize some numerals right.

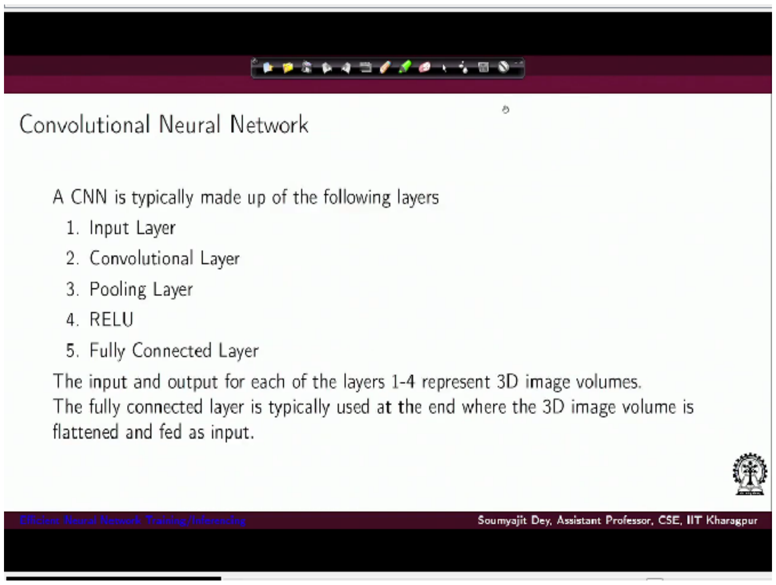
So essentially, you want this number 4 to be recognized, and you have many output possibilities. 10 digit possibilities. So the class for the digit 4 should get the highest value, right. And that is how you want this neural network architecture to work and assign the class code right. But the way it differs is the CNNs since they are going to be tuned towards processing images only, they are arranged in some specific ways.

For example, when you are capturing an image, typically the arrangement even for a 2D image, your data structure shall be a 3 dimensional thing. Essentially, you should have the pixel values for corresponding to height and width coordinates and also for each coordinate pair of height and weight, you are going to have 3 channels and you have to have memberships for all those color channels right.

So, you have the input standard in a available in a three dimensional standard. And corresponding to that, the CNN will have neurons arranged in three dimensions as well. And also there is something important that, in general, when people dig up CNNs, they also make use of facts like well, the neural network architecture did not have full connections at every point like each intermediate node can represent a filter, which is tuned towards gathering a specific kind of information and it may not be gathering it from all parts of the image versus specific parts of the image and all that.

All you are trying to say is that once we decide that the input is an image, and these are the features of the image I am looking for, accordingly, I can decide weights and accordingly I can also decide the connectivity of the intermediate nodes.

(Refer Slide Time: 16:43)



Convolutional Neural Network

A CNN is typically made up of the following layers

1. Input Layer
2. Convolutional Layer
3. Pooling Layer
4. RELU
5. Fully Connected Layer

The input and output for each of the layers 1-4 represent 3D image volumes.
The fully connected layer is typically used at the end where the 3D image volume is flattened and fed as input.

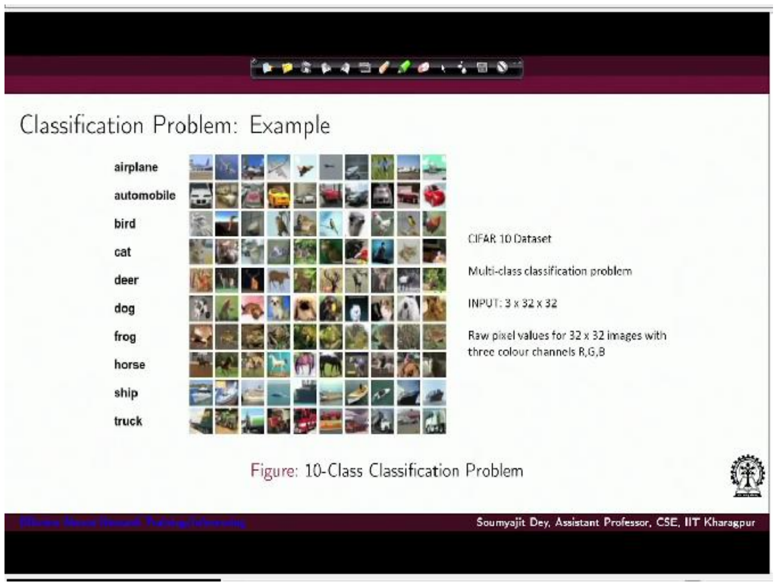
Efficient Neural Network Training/Inferencing   Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, holistically speaking, the CNN is typically made of the following layers, you have the input layer, after the input layer, you have these what we call as convolutional layers. For the convolution layer represent different image filters available as 3D masks which will be applied for mining information from the image. And then you have something called the pooling layer. Now this is followed by an activation layer typically that is taken as RELU.

And at the end you have what we call as a fully connected layer, which is just like the normal neural networks. The input and output for each of these layers to 1 to 4, they represent 3D image volumes.

So, up to the RELU layer, the input is a 3D image. After the convolution operation, I get 3D volumes. After the pooling, I will again get a 3D volume with some reduction in the length, the height widths etc. Even after RELU I have a 3D volume, but the fully connected layer is used at the end and the 3D volume is flattened, and it is provided as a linearized score, right.

(Refer Slide Time: 17:53)



Classification Problem: Example

CIFAR 10 Dataset

Multi-class classification problem

INPUT: 3 x 32 x 32

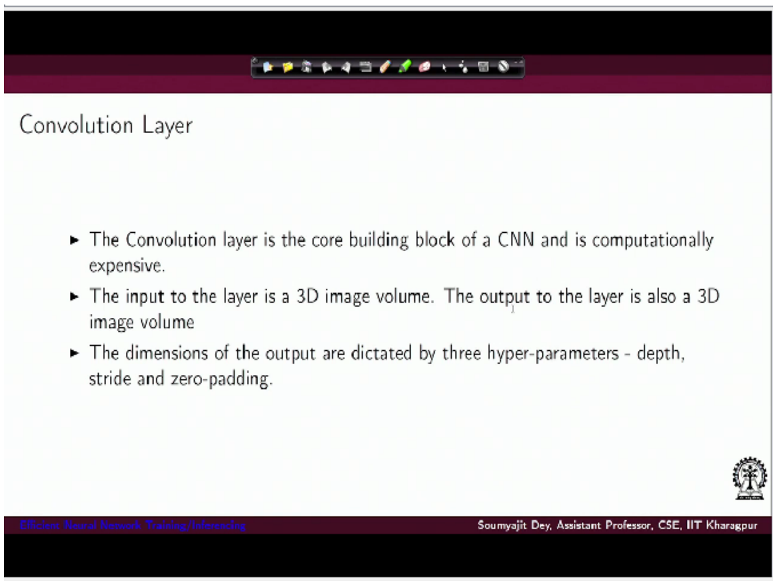
Raw pixel values for 32 x 32 images with three colour channels R,G,B

Figure: 10-Class Classification Problem

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, for example, we will just show some examples here like which are popularly use data sets, for example, this CIFAR 10 data set. For that, I can have a multi class classification problem, which is pretty well known. I have input images of size 32 cross 32. Let us say I am trying to classify these images into these categories provided in the left hand side. And of course, I have 3 channels for red, green and blue, right. So, overall the in each of the input images is of the dimension 3 cross 32 plus 32.

(Refer Slide Time: 18:29)



Convolution Layer

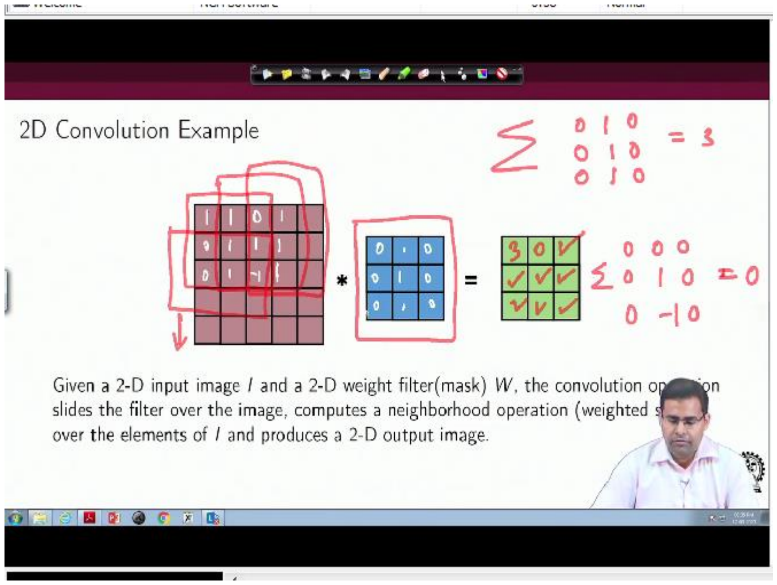
- The Convolution layer is the core building block of a CNN and is computationally expensive.
- The input to the layer is a 3D image volume. The output to the layer is also a 3D image volume
- The dimensions of the output are dictated by three hyper-parameters - depth, stride and zero-padding.

Efficient Neural Network Training/Inferencing

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, the convolution layer, this is essentially the core building block of a CNN and this is a computationally expensive layer. And it works on the input images, which are 3D image volumes. And essentially, there are some convolution masks or filters which operate on these input volumes and provide an as output this layer is going to again produce a 3D volume. Now, the dimension of this output, they are dictated by 3 parameters. That what is the depth of the layer at what stride it is going to work and the amount of zero paddings.

(Refer Slide Time: 19:09)



2D Convolution Example

Given a 2-D input image I and a 2-D weight filter(mask) W, the convolution op[illegible]ion slides the filter over the image, computes a neighborhood operation (weighted s [illegible] over the elements of I and produces a 2-D output image.

Will understand what it means. So let us take an example of a 3D convolution. But before that, let us just understand that I mean how this is again, I mean, conforming to the basic neural network architecture. So instead of having inputs like X 1, X 2, etc., which we're discussing in our earlier things, essentially, we were talking about test data or training or sorry, the training data or the test data depending on what kind of what you are doing, which will like vectors or matrices or tensors, right.

And essentially, the image is also following such a representation. For example, on the left hand side, we are considering here, a 2D image. But the thing is we are operating it with a weight matrix. And it happens to be that in this case, the weight matrix is basically an image filter or a mask right and this and the way you choose the mask depends on what kind of operations you want to do on the image, what kind of information you want to take out of the image, right.

So here you have an image, here we have a 2D weight filter mask, let us call it W, because this is essentially like a matrix vector multiplication, you are doing matrix multiplication you are doing the image is getting transformed with this mask. But the operation you are doing sorry, is a bit different from standard matrix, matrix multiplication. What you are really doing here is convolution operation, because that is how you operate filters on images.

I mean, of course, what is convolution, how exactly we do convolution. I mean, what is the motivation behind doing that. I mean, as we know that these are standard image processing algorithms, and they are a standard transformations coming from signal systems theory, who is defined that will, by performing convolution with some specific filter, what information am I really going to get.

So we are really not looking into that whether but rather restricting ourselves to understanding that well how to implement a convolution. And the important point for us at this point is to understand that here, the weights do not do normal matrix multiplication, but rather that gets replaced with the convolution operation, which is the specific feature of convolution neural networks, right.

So, how does it really work. For example, if we take this 5 cross 5 image, getting convolve with a 3 cross 3 mask, essentially, this mask is going to slide over the possible locations on this image, and compute and output 2D image like this, okay. **(Video Starts: 21:44)** So, let us try and understand how this is going to work. So maybe we will push in some values here. **(Video Ends: 22:01)**

So let us take it very simply, let us say it is all 1's. I am just writing some example values here. Now, when I do the convolution, what is really going to happen is in your output, the value at this location you are getting by applying this input mask on this possible location, so just overlay this mask over this part of the input image, right and then you do element wise multiplication. So what do you really get.

So essentially you are multiplying element wise. So the values that you are going to get 0, then 1, then 0 then again 0 1 0 then again 0 1 0 right, now just sum them up. So, you get a 3, right so, that is what you right here right. So, is that simple. Now, this gives you the value the 0 0th location of the output image. What about the next location. How do you get it. So, again just take the mask and overlay it, but by sliding it one part one position, right.

So essentially, you will be applying the mask on these locations. So, let us say these are my values. So again you can do the computation now. So all you get is right. So, sum it up and that is going to give you a 0. So that is what you get. Similarly, you overlay the mask again here, right, fill up this entry. And then you downshift the mask by one position, fill up this entry, downshift, another position will have this entry, right shift one position, another position and downshift like that, right.

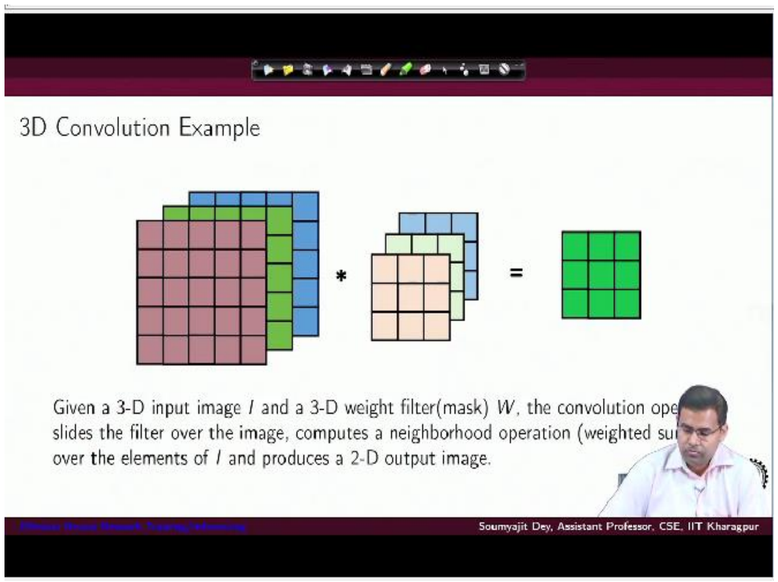
So in that way, you just compute all the values, given this 2D weight filter or the mask, right. So we can continue like this. **(Video Starts: 25:44)** So that is how you can build on this example. And here in this animation, we are just trying to show that for the output location, how the change of the **over** overlay plays a role when you compute the output values of the image. So as you can see, you have this 2D input image, you apply the 2D weight filter mask and you are getting a 2D output image.

So there is no change in the dimension. And of course, the dimension of this output image depends on 2 things, the dimension of the input image as well as the dimension of the mask. And in what is try I am applying the operation. We will see what that means. So then, as you can see that for the first location, you are putting the mask in the left, top left position that is possible.

That is the overlay region, you shifted by one position and you keep on shifting like this in the output image, right. Move to the next row, again keep on shifting in that row, and so on so forth right. This is how you do the 2D convolution. A standard image processing transform. All that is happening is we are now claiming this is getting done in the first layer or any layer which we are defining as a convolution layer of a neural network pipeline.

That makes it is a specific pipeline and we call it a convolution neural network. All we are saying in some of the layers, we are not doing a generalized matrix multiplication, we are replacing it by this kind of image filter mask convolution operations to get a different to get and transform image vector fine. **(Video Ends: 27:33)**

(Refer Slide Time: 27:36)



3D Convolution Example

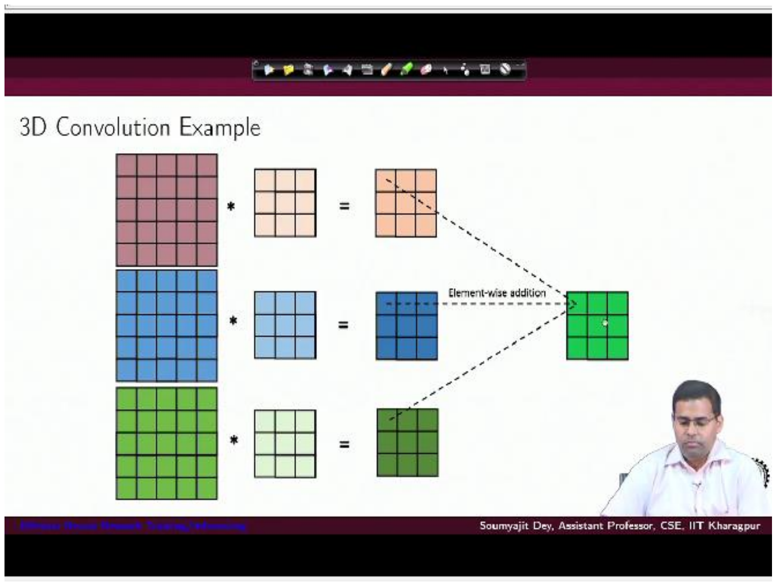
Given a 3-D input image I and a 3-D weight filter(mask) W, the convolution ope slides the filter over the image, computes a neighborhood operation (weighted su over the elements of I and produces a 2-D output image.

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

Similarly, this idea can hold for even 3D convolutions. So, just like we are considering 2D images and 2D masks, simply considered 3D image and convolving the 3D image so, there is a convolution we are denoting it here via 3D weight filter or mask W. So, what will happen here, the operation will slide over the image and compute the neighbourhood operation, the weighted sum. So essentially, the weights are getting multiplied with these values in the image that we saw.

And then finally, we are doing the summation and then you do it over the elements of i and produce the output image right. So, essentially, you have a 3D input image, a 3D weight filter mask, and the convolution will operation will slide the individual filters over the images and it will do the neighborhood computation, like we discussed earlier already over the elements and produce the output image right.

(Refer Slide Time: 28:43)



So here we are describing it further, what really is going on. So as you can see, these are the on the left hand side we have the image and we are showing the different channels of the image, right. And then I have the 3D mask. So there are 3 2D masks essentially, right. So, that is what we have here, right these are the convolution operation, and here we are explaining it with a broader diagram.

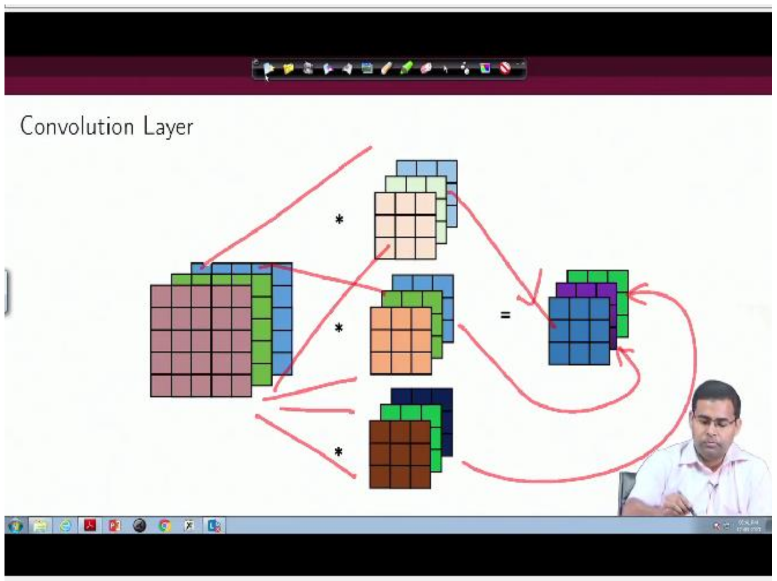
So, essentially you do pair wise convolution for each of the layers right. So, you have the mask arranged with specific layers for specific image channels and you just do component wise convolution operation computation. So, this is for the first channel, this is for the second image channel, this is for a third image channel, you are just doing convolution for the corresponding mask layers.

And doing the operation like we have discussed earlier, you are taking this mask overlaying it over the image and then sliding it on both ways and computing each of the entries of the output image transformation. And how do you really compute after overlaying the mask you are doing element wise multiplication, essentially you are multiplying these image pixel information with the mask weights.

And finally doing a weighted sum of this region, again the weighted sum of the next region and so on so forth, right. And finally, you do element wise addition to just flatten this 3D outputs to a 2D image transformed output, right. So, here you are again doing element wise addition right. So, first, you already have already done an element wise addition, which was also weighted by the mask for computing each of these entries.

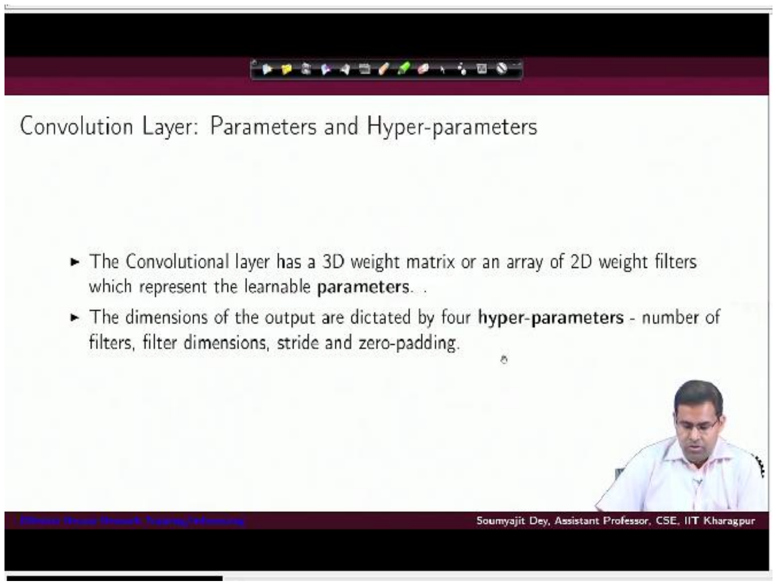
And then for this 3 sets of 2D masks, you do individual element wise addition to get the values here, here, here, so on so forth right.

(Refer Slide Time: 30:36)



Now, let us move forward.

(Refer Slide Time: 30:42)



Convolution Layer: Parameters and Hyper-parameters

- The Convolutional layer has a 3D weight matrix or an array of 2D weight filters which represent the learnable **parameters**. .
- The dimensions of the output are dictated by four **hyper-parameters** - number of filters, filter dimensions, stride and zero-padding.

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, if we keep on doing it for multiple masks, because here we just considered a 3D image with one 3D mask, but in general, in a neural network architecture, I can have multiple mask right. So, these are 3D image, there is one convolution to be done with one 3D mask, another convolution to be done with another 3D mask, another convolution to be done with another 3D mask. So, as we have seen just in the previous picture.

So, this convolution is going to provide me with one 2D output, this convolution is going to provide me with another 2D output. And again this convolution is going to provide me with another 2D output, so on and so forth, right. Now, we have already discussed the notion of hyper parameters. For example for our earlier normal vanilla neural network example. The hyper parameters were the different layers.

And the different set of the neural network that is essentially the architecture of the neural network, the weight matrices and all that. So it also decided what it should be in this case for convolution layer, right. So, with this basic introduction to CNNs, we will end this lecture. Thank you for your attention.