Deep Learning

Prof. Prabir Kumar Biswas
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 52
Deconvolution Layer

Hello, welcome to the NPTEL online certification course on Deep Learning. In our previous class, we have talked about one of the applications of the deep learning techniques which was the face net and the purpose was to recognize or to verify human faces, and the applications that we have seen that it has many applications starting from biometric authentication to security to surveillance, health care and many more applications.

And you recollect that the kind of network that we have considered so far that is the deep convolutional neural network, this sort of network that we are considered they are actually discriminative networks. In the sense, that the network learns various classes or various categories of the objects or the images. And, once the network is trained properly then given an unknown image or an unknown object the network tries to associate or tries to classify that unknown image to one of the classes or one of the categories which the network has already learned.

That means it basically discriminates one object from other objects. And the face recognition or face authentication, the image recognition, the character recognition all these belong to this particular kind of category. Now, there might be other applications say for example, I may like to apply the deep learning techniques for denoising or noise removal from an image or maybe I want to apply the same machine learning technique or deep learning technique for semantic segmentation of an image. The semantic segmentation is nothing, but I identifying the different pixels within the image to which class that particular pixel belongs.

So, there again it is a short of discrimination, but it is not the discrimination at the image level, but it is the discrimination at the pixel level. That is every pixel will be classified to one of the classes. If the classes are just two classes the background and foreground then we have to decide whether a pixel belongs to foreground or the pixel belongs to background. Similarly, if an image contains say car, human being, dog and many such different objects then we may have to identify that whether a pixel belongs to a car or a pixel belongs to a human body or a pixel belongs to dog region and the collection of all those pixels belonging to a particular region or a particular category that forms a segment. So, semantic segmentation is one of the very very important application in machine vision techniques.

Now, you find that when you go for this kind of semantic segmentation or pixel wise decision making then simple convolutional neural network that we have discussed so far that really does not work. So, in this case what you may have to do is something which is the reverse of what the convolutional network has done; that means, the kind of operation that is that what we need is what is a deconvolution sort of operation, that is the reverse of the convolutional operation.

(Refer Slide Time: 04:19)

CONCEPTS COVERED

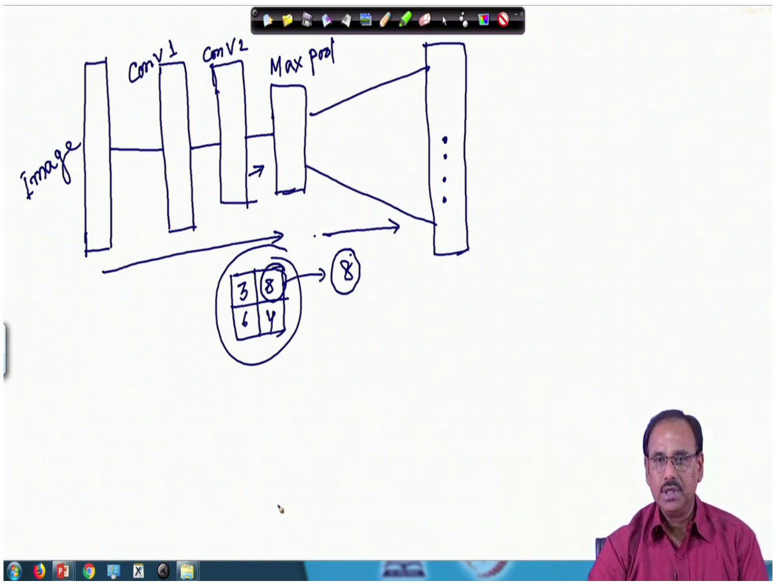
Concepts Covered:

- FaceNet
- Filtering/Semantic Segmentation
- Deconvolution
- Upsampling

So, in today's lecture what we are going to discuss is some applications of filtering or semantic segmentation. But, as I said that filtering or semantic segmentation these requires taking a decision at the pixel level not at the image level and for which we need an operation, which is some sort of inverse of the convolutional operation. So, what we will discuss today before we go for filtering and semantic segmentation kind of applications is what is deconvolution and what is upsampling. Because, when I talk out about the convolutional neural network after say one or more convolutional layers we had a upsampling there or which we called as a max pool layer.

So, we will discuss about what is deconvolution and how the upsampling can be done in order to move to your actual image domain because there we have to take the decision at the pixel level in the image domain. So, let us see that what we have done in case of convolutional neural network so far.

(Refer Slide Time: 05:47)



So, what we have seen that in case of convolutional neural network; obviously, you had an input layer where the input was an image. So, here the input was the image. Then this image is passed to next layer which is as a convolution layer, it may passed through a number of convolution layers. So, we can call it convolution 1, convolution 2, then it passes to say a max pool layer. So, what we have seen that what the max pool layer does is within a neighborhood it extracts what is the maximum activation function or where the feature value is maximum.

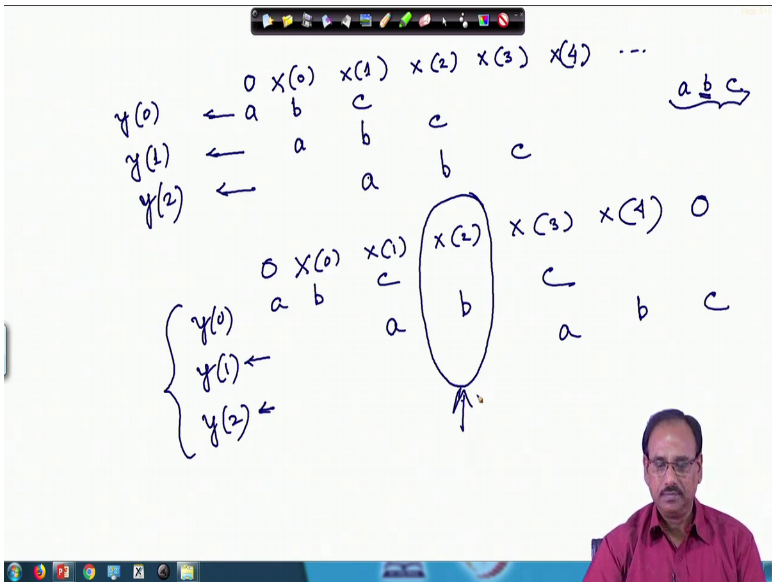
So, if I have if I take perform this max pool operation in a neighborhood of say 2 by 2 and suppose the activation values which is passed from this convolution layer 2 to this max pool layer within this 2 by 2 neighborhood it had values something like 3, 8, 6, 4 something like this. So, out of these 4 activations you find the maximum value is actually 8. So, what the max pool layer will do is it will simply extract 8. So, all these 4 pixels or all these 4 activations will be replaced by one activation which is the activation value 8.

And as a result what the max pool layer does is it reduces the dimensionality of the feature map which is created by the convolutional layer. So, but as we said that when I need to take an action at the pixel level then what I have to do is I have to explode this and come to a pixel domain, where the size is same as your input size and the network has to take decisions on every individual pixel within this image domain. So, as in the forward direction we have the convolutional operations as well as upsampling operations like max pooling.

In the reverse operation what you have to do is we have to have deconvolution operations along with up pulling, that is here what we have done is we have done upsampling you have reduced to the dimension, in the reverse operation we also have to increase the dimension so that finally, we can go to the size of the output which is same as the size of the image.

So, now, let us see what how this can be done. So, before that I will explain with one-dimension the same is applicable in two-dimensional cases. So, what we have done in case of convolution?

(Refer Slide Time: 08:47)



Suppose, let us assume that we had a set of input samples x 0, x 1, x 2, x 3, x 4 and so on. And I have a convolution Kernel, say a convolution Kernel of size 3 or the Kernel coefficients are given by a, b and c. So, during convolution operation what we have done is we have placed at this Kernel, I mean the way the convolution is computed is in such a way that say a is placed over here, the Kernel center if I assume the Kernel center is b, b is placed over here and c is placed over here and this gives you the output sample y 0, y 0 is nothing but here we assume that we had padded it with 0, right. So, y 0 is nothing, but 0 times a plus x 0 times b plus x 1 times c. So, y 0 simply becomes x 0 times b plus x 1 times c.

Then if you shift the Kernel, you make it a over here, b over here c over here and you get an output y 1, where y 1 is given by a times x naught plus a times x 0 plus b times x 1 plus c times x 2 and so on. Then to compute y 2 we have shifted the Kernel once more. So, a came here, b came here, c came here and the corresponding point by point multiplication followed by the addition gives you y 2. And this is the kind of convolution where we have said that this convolution is with stride equal to 1, and when I take a convolution with stride equal to 1 your number of samples of the input and the number of samples at the output that is the convolution output it remains the same.

Then we have also said that instead of computing convolution with stride 1 we can also compute convolution with stride 2. So, if you compute convolution with stride 2, then number of samples at the output and number of samples at the input they will be different. In fact, the number of samples at the output will be less than the number of samples at the input. So, convolution with stride greater than 1 reduces the size of the feature map.

The same reduction in size of the feature map you get if you perform convolution with stride 1 followed by a max pouring operation that is what is usually done in case of convolutional neural network. But what is the difference between these two? When you perform convolution with stride 2, let us assume say I have same one x 0, x 1, x 2, x 3 and x 4 which are my input samples and I want to perform the convolution with the same convolution Kernel, but now in this case with stride equal to 2.

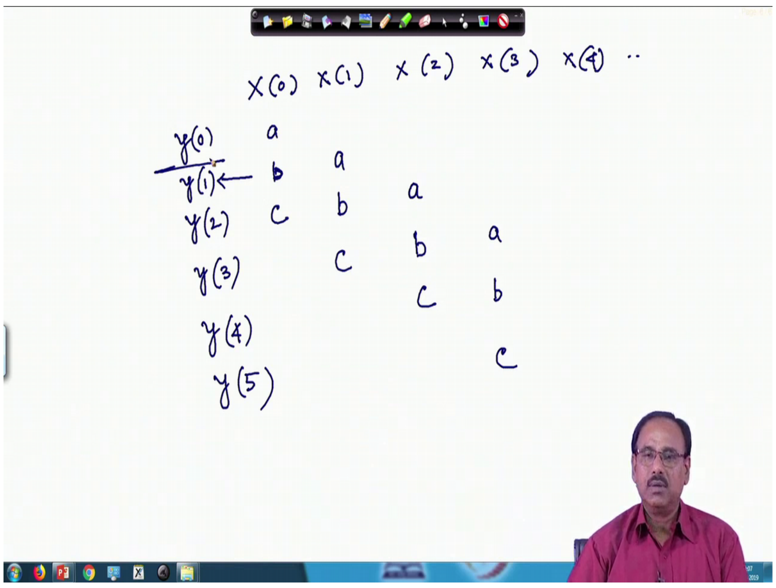
So, my y is 0, now becomes a times 0 plus b times x naught plus c times x 1. Now, stride is 2, so a is shifted here, b goes here, c goes here, so I get my y 1 which is a times x 1 plus b times x 2 plus c times x 3. So, this is what is my y 1. Similarly, y 2 again stride its 2. So, a goes here, b goes here, c goes outside, so here what I had is up adding with 0, so my y 2 now becomes a times x 3 plus b times x 4.

So, you find that once this convolution is done your number of samples at the output of the convolved output is 3 against when the number of samples at the input was 5. But effectively what you have done is you find that I have not computed what would have been the filter response if the convolution was centered at x 2. I have simply omitted that particular computation. And it may be possible that at this location the Kernel response of the activation would have been maximum.

So, if I perform convolution with stride more than 1, I may miss some activations which actually would have been maximum. So, it is always better that you perform convolution with stride 1 followed by max pooling because max pooling had this been maximum the max pooling operation would have captured this after performing convolution with stride 1.

Now, so, what we are talking about the reverse operation? This is what is the convolution operation. What we are trying to find out is what is deconvolution. So, again to illustrate deconvolution in one-dimensional case, you find that in case of convolution what we have done is we have found out the response of a filter placing the filter over our receptive field. The deconvolution can be thought of just to the reverse process, that is if I have a particular activation of the response of a filter where the filter has certain Kernel can I throw the information in the reverse direction; that is can I create the receptive field, ok. So, it is something like this.

(Refer Slide Time: 14:45)



So, suppose I have again my input samples which are x 0, x 1, x 2, x 3, x 4 and so on. So, you find that when we have talked about the convolution our Kernels was placed like this a, b, c this was the way we placed the Kernels and I computed the output sample as weighted sum of the input samples where the samples are weighted by the corresponding Kernel coefficient.

In deconvolution I just want to perform that reverse operation. How do I do it? So, now what I do is I put my Kernel coefficients in this forms say a, b and c, these are my Kernel coefficients. And what I get is this y is 0 is now becoming a times x naught, y 1 becomes b times x naught and y 2 becomes c times x naught. So, this x naught which you can say it is the filter response now flows to the other direction or it is distributed within the neighborhood which we can say that a short of receptive field. So, a filter response is now distributed to its neighborhood.

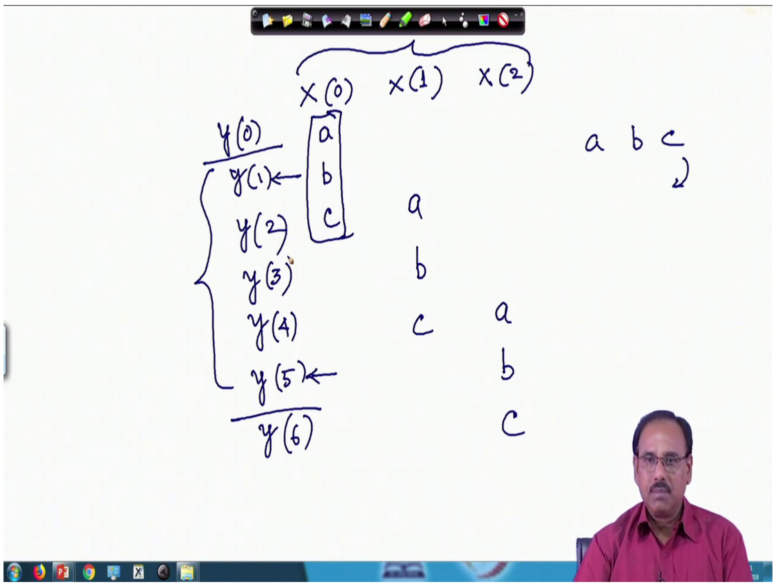
And again I can have this strider operation. So, in the first case if I take the operations with stride equal to 1, so this filter will be strided by 1. So, this is what I get. There is my y 3. So, up to this my y 0 will be a times x naught, y 1 will be b times x naught plus a times x 1, y 2 will be c times x naught plus b times x 1 and y 3 will be c times x 1. I can go on further I again stride the filter coefficients like this. So, it comes a, b, c and I compute y 4.

So, now, you find that your y 2 becomes c times x naught plus b times x 1 plus a times x 2; y 3 becomes c times x naught plus b times x 2; y 4 becomes c times x 4 x 2. You can go on. So, it is a, b, c now I can compute y 5, where y 5 will be c times x t; y 4 is now c times x 2 plus b times x 3 and y 3 is c times x 1 plus b times x 2 plus a times x 3 and it can go on like this. And once you do this you find that this center of the Kernel is actually associated with y 1.

So, as we have done in case of convolution that in order to have your input array and the output array size to be same what we had done is we had to added extra 0s at the beginning, and at the end of the sequence in this case to have your input array size and the output array size to be the same what you have to do is we have to crop these extra elements that has been computed by this deconvolution Kernel. So, that your input size and the output size remains the same. The same operation can also be done with stride more than one.

So, you found that what we have seen in case of convolution that if you perform convolution with stride more than one then the number of samples at the output becomes less than number of samples at the input. Similarly, if I perform convolution with stried more than one then number of samples that the output has to be more than number of samples at the input. Let us see how does it actually happen.

(Refer Slide Time: 19:11)



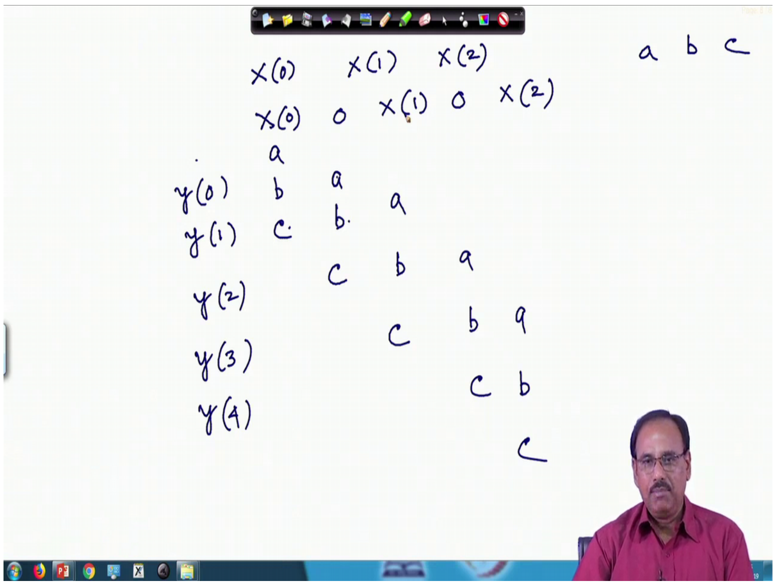
So, let me assume that I have just 3 input samples x 0, x 1 and x 2. And my filter coefficients again is a, b and c, ok. So, first what will compute is I have again I compute what is my a 0, what is my what is y 0, what is y 1, what is y 2 and all that. So, right now I compute say y 0 over here, this is y 1, y 2, so y 0 will be a times x 0 till now, y 1 will be b times x 0 and y 2 will be c times x 0. Then you shift the Kernel and as we said that we are using stride 2, so this Kernel will be shifted by 2. So, now it becomes a, b and c and I get y 2, y 3 and y 4.

So, what is y 2 now? y 2 now becomes c times x 0 plus a times x 1, y 3 becomes b times x 1 and y 4 becomes c times x 1. Again you shift it with stride 2, so I have a, b and c this is what is my Kernel and I compute y 5 and y 6. But as we said before that this x y 0 and y 6, these are the two deconvolution outputs which will crop because the center of the Kernel coincides with y 1 here and the center of the Kernel coincides with y 5 here. So, the number of samples that you are generating after deconvolution where the deconvolution stride is equal to 2 is equal to 5. So, I have created 5 samples against 3 samples at the input.

So, you find that if you perform deconvolution operation with deconvolutional Kernel width stride more than 1, in that case you are expanding your output or in other words we can say that we are gradually moving towards the original size of the image. And if you look at this if you compare what was your convolution, what is the deconvolution, in case of convolution we had put the Kernels as in this form a, b and c where x 0 was multiplied with a, x 1 was multiplied with b, x 3 was multiplied x 2 was multiplied with c to give you one sample of the output. In case of deconvolution, what you are doing is you are taking this simply the transpose of this. So, now, the a, b and c the Kernel coefficients that you get are nothing, but that transpose of the Kernel coefficients that you have used for convolution purpose.

So, the deconvolution operation is also sometimes called as transposed convolution. Operation system whether I call it deconvolution or whether I call it transpose convolution the operation is same. There is another name which is given to the same operation which is known as sub pixel convolution. So, what is sub pixel convolution? In case of sub pixel convolution, again let me assume that we have 3 input samples x 0, x 1 and x 2.

(Refer Slide Time: 22:51)



What you do is you up sample these input samples. So, what we do is I make it x 0, 0, x 1, 0, x 2. So, from 3 samples I am upsampling it to 5 samples interlaced with 0s. Now, if I perform the same deconvolution operation with the Kernels that we have used a, b, and c, my operation will be something like this. So, if I put a, b, c over here I get my y 0, I get my y 1, I get my y 2, I get my y 3, I get my y 4 and so on.

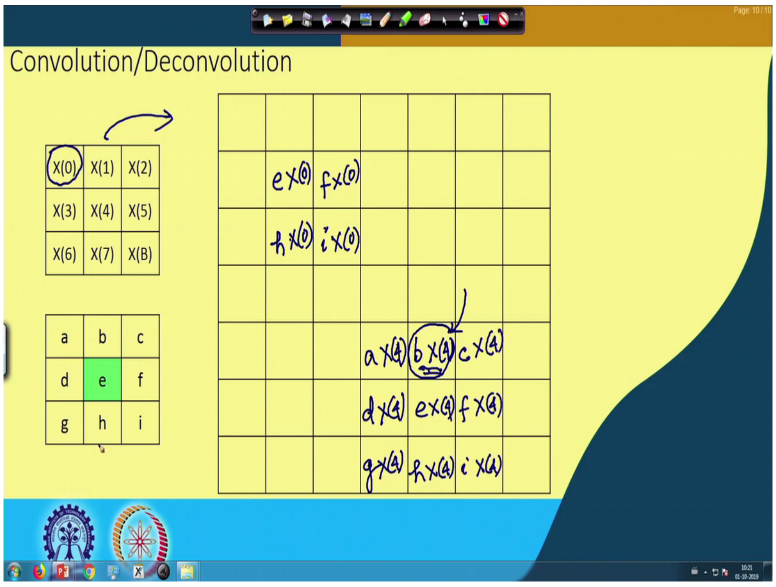
But in this case first what I am getting is a, x 0 times a that comes over here x 0 times b and that gives me y 0, x 0 times c that gives me y 1 then you shift the Kernel with stride 1. So, it becomes a, b and c over here, go on doing it, a b c over here, a b c over here, a b c over here. So, you find that your y 0, now becomes b times x naught plus a times 0. So, this is actually b times x naught; y 1 becomes c times x naught plus b times 0 plus a times x 1, so it is c times x 0 plus a times x 1; y 1 becomes 3 times 0 plus b times x 1 plus a times 0, so it is simply b times x 1; y 3 becomes c times x 1 plus b times 0 plus a times x 2, so it is c times x 1 plus a times x 2. Similarly, y 4 simply becomes b times x 2.

So, if you compare this computation with the previous computation that we have done on our original samples x 0, x 1, x 2 with stride equal to 2, where the number of output samples was increased to 5. And I am getting the same thing the same output if I simply upsample the input samples interlaced with 0s and then perform the convolution of the deconvolution with stride equal to 1 and obviously, you crop the boundary pixels or the boundary sample values output values that you get to get your acceptable sample values.

So, both these operations, the deconvolution operation with stride 2 and the deconvolution operation with stride 1 taking the upsampled version of the input samples both of them give you the same output. And this is called sub pixel deconvolution, sub pixel convolution. The reason being I actually have samples x 0 and x 1 at index location 0 and 1, but I am assuming that I have a virtual sample in between x 0 and x 1 which is my sub sample or sub pixel and that is at location in between index 0 and 1.

So, this is also known as sub pixel convolution. So, whether I call it deconvolution operation or transpose convolution operation or sub pixel operation all of them are actually same. And the same is equally applicable in case of two-dimension. So, how do you apply it in case of two-dimension?

(Refer Slide Time: 27:13)



Let us take this particular array. Say, I have input set of samples, I am taking a 3 by 3 array. My input set of samples are x 0, x 1, x 2, x 3, x 4, x 5, x 6, x 7 and x 8. So, this is my 3 by 3 array of the pixel values or the 3 by 3 array of the feature map. And I assume that I also have a 3 by 3 Kernel convolution or deconvolution Kernel whatever you call it which is a, b, c, d, e, f and g, h, i, where e is the center coefficient of the convolution Kernel.

Now, how the convolution will be done or the deconvolution will be done? If I put this Kernel centered at location x naught and whatever the deconvolution value, that I get let us put it over here. So, I get e times x naught over here because e is the center coefficient, over here I get a times x 0, over here I get b times x, b times x 1, here I get c times x 2, here I get d times x 3, here I get sorry the operation is actually like this. So, I am putting my Kernel centered at say x 0. And whatever the deconvolution values that you get will be something like this. So, I put over here. So, here I get e times x 0. Here, because this one has to be expanded to all its neighbors after being weighted by the corresponding Kernel coefficients.

So, at this location what I will get is h times h 0, at this location I will get i times x 0, at this location I will get f times x 0. Similarly, if I put the Kernel centered at location x 4, let me put the result over here. So, I am putting the Kernel centered at location x 4. So, over here I will get e times x 4, over here I will get b times x 4, here I will get a times x

4, here I get c times x 4, here I get f times x 4, here you get I times x 4, here it is h times x 4, g times x 4 and here what you get is d times x 4.

And you find that when I am spreading this information of the feature value export to its neighborhood in various locations, say for example, in this location I will get also information coming from other neighboring pixels. So, all those partial informations you have to add together to find out what will be the final deconvolved value at this particular location. So, here I will get information from x 1, I will get information from x 2, I will get information from x 3 and so on, but I bring weighted by the corresponding Kernel coefficient and all these partial products are to be added together to find out what is the final deconvolved value at this particular location.

The same operation as we have seen that sub pixel convolution can be done, if I simply go for upsampling of this interlaced with 0 and then perform the convolution operation with the same Kernel with stride equal to 1. That gives you expansion or increase in size of the deconvolved output. So, when we have talked about the convolutional neural network, we had the convolution operation followed by the max pooling operation, where you have said that a max pooling is some short of down sampling of the features that you get, so that your size gets reduced.

So, in the reverse operation what we have to do is first we have to go for upsampling, that is whatever max pooling operation has to be done that has to be unpooled and after unpooling I had to convolve with the convolution Kernel or I have to deconvolve with the d convolution Kernel to get your final feature maps. And that has to be done stage by stage. It depends upon how many convolution layers and how many max pooling layers you had on the convolution side. On the reverse side I have to have equal number of max pooling operation and the deconvolution operation before I finally, get my output.

So, today what we have discussed is that to take some decision at the pixel level or the image level, I need to perform some operation which is just opposite of the convolution and max pooling operations that we have done in the convolutional neural network. And this inverse operation is what is the deconvolution operation along with the up pooling operation. And here our next class or next few classes, we will discuss about that how these deconvolution operations are useful for image domain operations like noise, filtering, semantic segmentation and different kinds of applications.

Thank you.