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

Lecture – 39
Popular CNN Architecture: VGG16, Transfer Learning

Hello, welcome to the NPTEL online certification course on Deep Learning. We are discussing about various popular convolutional network, neural network models.

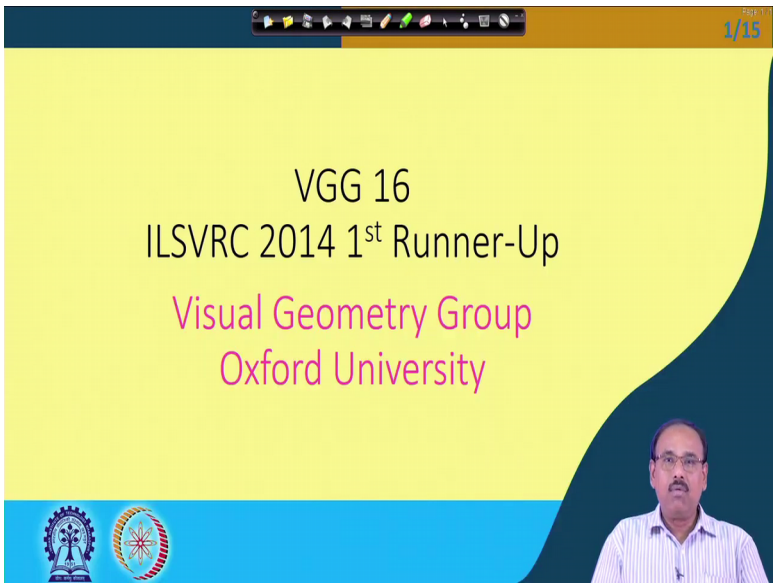
(Refer Slide Time: 00:43)

And in our previous class, we have talked about the AlexNet. And we have seen that the AlexNet consists of a number of convolutional layers, a number of max pool layers and a number of fully connected layers. So, we have seen that the number of convolution layers in case of AlexNet is 5, it has got 3 max pool layers and 3 fully connected layers. Of course, the output fully connected layer is having a softmax activation function, and the number of such softmax nodes is equal to 1000. So, one node corresponding to each of the category in imagenet database.

And we have also seen that the AlexNet architecture is designed in two different pipelines and it was trained on 2 GPU based machines. The other feature that we have seen in case of AlexNet is the size of the convolution kernels. AlexNet uses different kernel sizes. It uses 11 by 11 convolution kernel; it uses 5 by 5 convolution kernel and it

also uses 3 by 3 convolution kernel. In today's lecture, we are going to discuss about another popular CNN architecture which is VGG net.

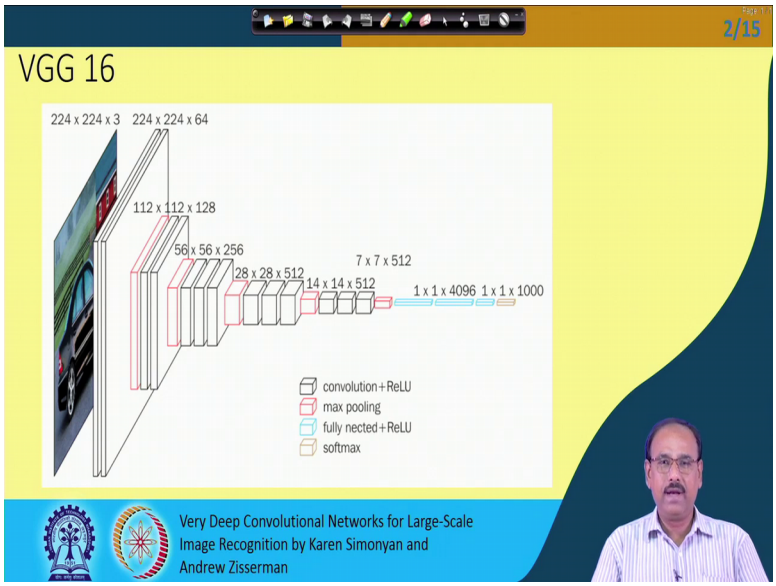
(Refer Slide Time: 02:11)



So, let us see that what is this architecture of the VGG net? So, VGG was is actually acronym for visual geometric group and VGG 16 which is actually a network having 16 different layers, I mean when I say that 16 different layers, these are the layers which have got tunable parameters there are other layers like max pool layer which does not contain any tunable parameter.

So, when you specify the number of layers or 16 this says that there are 16 layers containing tunable parameters. So, this particular architecture was suggested was proposed by visual geometric group from Oxford University. And this architecture was runners up in the visual recognition challenge the 1st runners up in the visual recognition challenge in 2014.

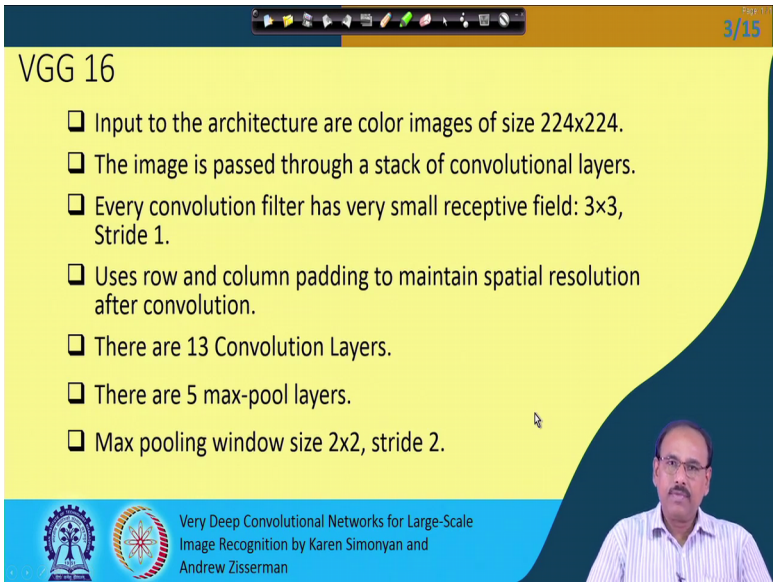
(Refer Slide Time: 03:11)



So, let us see that how this VGG 16 the architecture looks like. So, as you see over here, there are a number of convolutional layers. So, all the black rectangles, they represent the convolution layers, along with the non-linear activation function which is rectified linear unit. So, the convolution layer along with the rectified linear unit that makes convolution layer. So, you can see over here the number of convolution layers that in case of VGG net is 13.

So, it has 13 convolution layers. It has 5 max pool layers, all the red rectangles represent the max pool layers. It has got 3 fully connected unit fully connected network FC network. So, the total number of layers having tunable parameters is 13 for convolution layer and 4 for fully connected layers, so that makes it 16, so that is why it is VGG 16. Of course, at the output you have the softmax layer having 1000 outputs, again one output per image category in the imagenet database. So, overall this is the architecture of VGG 16.

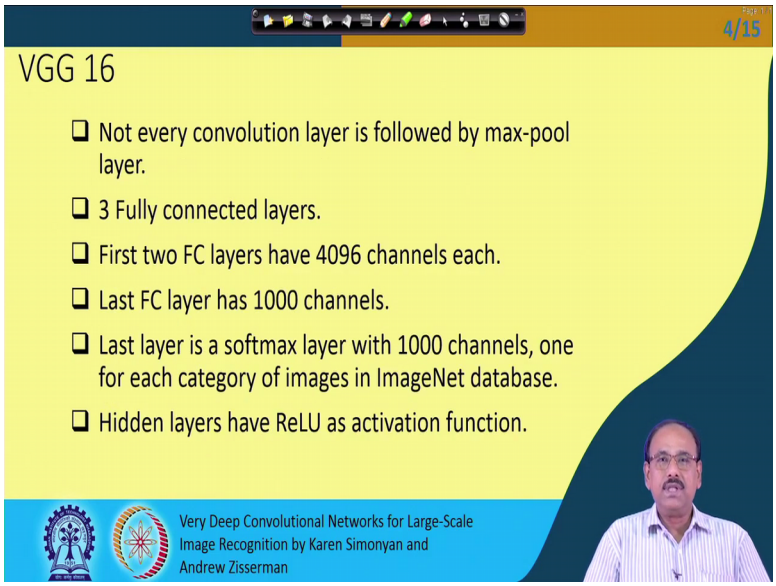
(Refer Slide Time: 04:43)



Now, if you look at the different features of VGG 16 CNN architecture, you find that the VGG 16, it accepts color images of size 224 by 224 pixels. So, color images. So, there are three different channels at the input, red, green and blue. And the image is passed through a number of convolutional layers as we have seen already in the architecture. In case of VGG 16, every convolution layer has a convolution kernel of size 3 by 3 and the convolution is carried out with a stride equal to 1, so that indicates that the receptive field of every convolution kernel is an image of size 3 by 3.

And every convolution kernel, every convolution layer uses row and column peeding padding, so that the size of the input feature map and the size of output feature map that remains the same or the resolution of the feature map after the convolution operation is performed, it remains the same. And as we have seen before there are 13 convolution layers, there are 5 max pool layers and the max pool operation is carried out over a window size of 2 by 2 and with stride equal to 2, which means that the max pool windows are non overlapping windows.

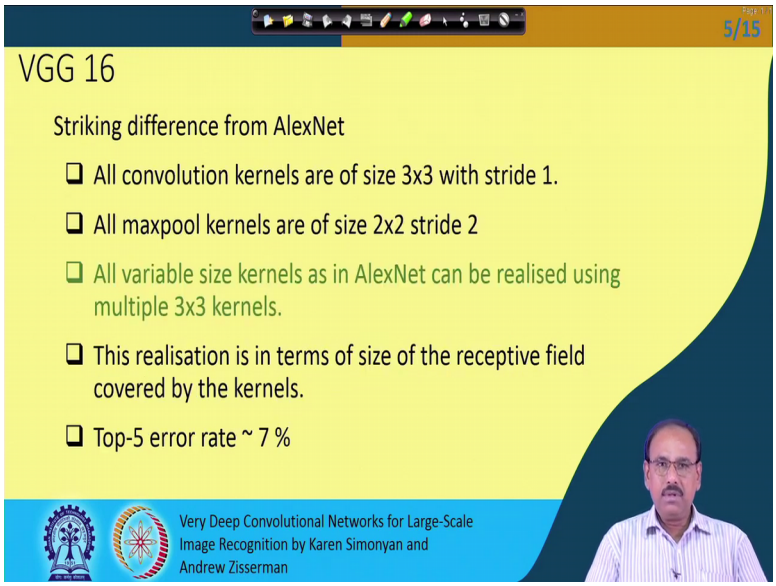
(Refer Slide Time: 06:27)



Among the other features it is obviously, because there are 13 convolution layer and 5 max pool layer, so not every convolution layer is followed by a max pool layer. There are number of convolution layers which appear one after another. There are fully connected layer which is having 3 fully connected layers, first two fully connected layers have 4096 channel each, and the last fully connected layer is having 1000 channels.

And the last layer as we have just seen it has got 1000 channels, one channel per category of image in the imagenet database. And every hidden layer or every convolution layer has activation function which is a non-linear activation function that is ReLu that is really a rectified linear unit.

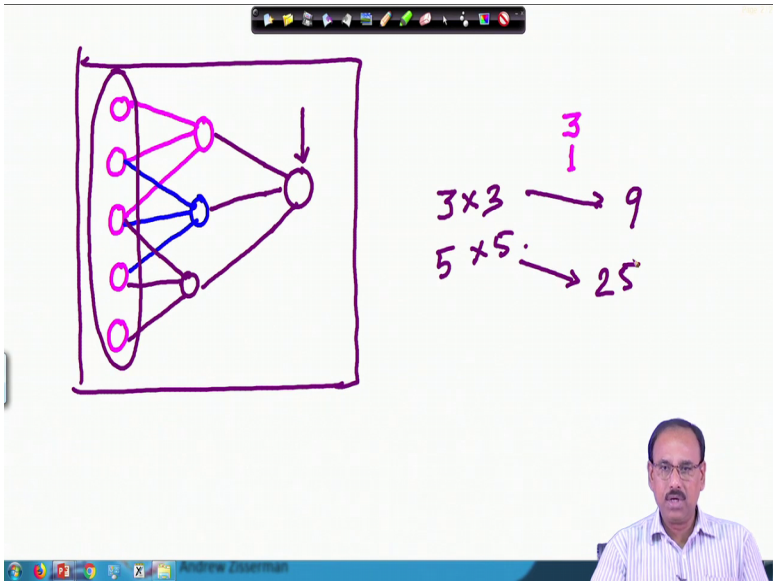
(Refer Slide Time: 07:29)



Now, if you try to see that what is the difference between this VGG 16 architecture and the AlexNet architecture that we have; that we have discussed earlier. In case of AlexNet as we have seen that AlexNet uses convolution kernels of different sizes There are convolution kernels of size 11 by 11, there are convolution kernels of size 5 by 5, there are convolution kernels of size 3 by 3. Whereas, in case of VGG 16, it uses convolution kernels of uniform size that is every convolution kernel is of size 3 by 3 and it uses it performs convolution operation with stride 1.

And as we have already seen that the max pool kernels are of size 2 by 2 and the max pool operation is performed with stride 2; that means, the max pool windows are non overlapping nature. However, though in case of AlexNet; AlexNet uses convolution kernels of different sizes, but every convolution kernel of different sizes can be realized using multiple 3 by 3 size kernels convolution kernels. Let us see how we can do it. So, let us assume that we have say and let us consider in one dimension for clarity of the explanation.

(Refer Slide Time: 09:11)



Let us take say 5 features. So, I have 5 different features of the input layer I can consider that I have 5 different pixels. And I want to perform a one-dimensional convolution with a convolution kernel size of say 3 and the stride that I use equal to 1. So, 3 is my kernel size which as I said that for clarity I will do it in 1 dimension, but it easily scales up to 2 dimension. So, my convolution kernel size is 3 and I have stride which is equal to 1.

So, as you perform the first layer of convolution, so the convolution operation will be performed over these 3 pixels to give you one feature in the next layer. Then as stride I am considering equal to 1, so this convolution kernel will be shifted by 1 and the next convolution operation will be over these 3 pixels, so that gives you the next feature in the next layer which is convolution of these 3 pixels.

And then this convolution kernel will again be shifted by one and the next operation will be over these 3 pixels giving me the next feature in the next layer. So, after one layer of convolution operations, I get these 3 features. And when you perform the next layer of convolution operation, then all these 3 pixels all these 3 features will take part in the convolution operation because I am having 3 by 3 kernel to give me a single feature.

So, now, if I consider this entire convolution operation which is done in a stacked manner or one convolution operation by a 3 by 3 kernel followed by another convolution kernel operation again by 3 the 3 by 3 kernel.

So, you find that this when I consider the convolution output is at this layer, it has performed a convolution operation over 5 different pixels over 5 consecutive pixels in previous to previous layer. So, this effectively gives the receptive field size, the size of the receptive field of this convolution kernel which is equal to 5. And you remember that this operation we have done by 2 successive convolution operations with convolution of kernel of size 3. So, this equally applies to 2 dimension.

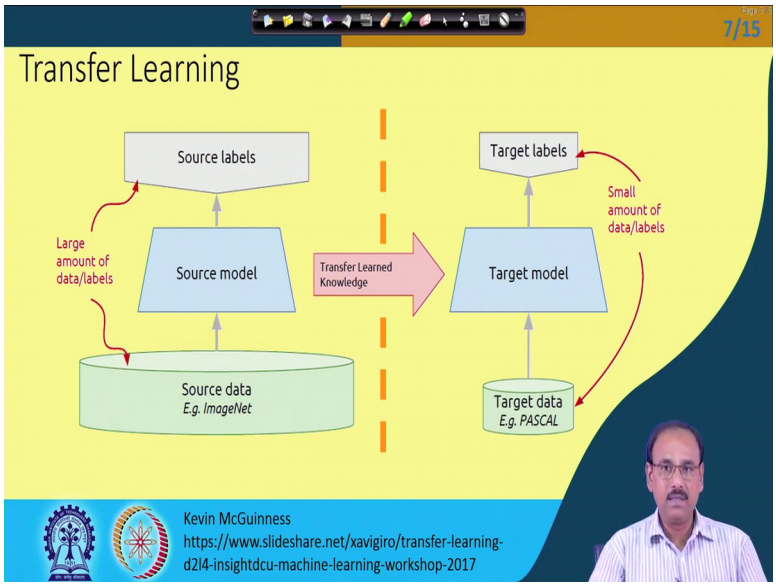
If I have a convolution kernel of size say 3 by 3 and perform 2 successive operations with this 3 by 3 kernel effectively, the receptive field size will be 5 by 5. Now, what is the advantage that you get? So, as I am using 2 stages of convolution operation with kernel size of 3 by 3, in the first stage I have 3 into 3 that is 9 parameters, in the second stage again I have 3 into 3 that is 9 parameters.

So, the total number of parameters that you have if I use 3 by 3 kernel size in 2 subsequent operations is equal to 9. Whereas, if I perform the same convolution of operation over a receptive field size of 5 by 5 in one stage, then my kernel size also has to be of size 5 by 5 as it is done in case of AlexNet and the number of parameters that you need in that case is 25.

So, we find that just by the stacked operation, the number of parameters have been reduced from 25 to 9. So, this gives a tremendous reduction in the number of parameters which needs to be trained in the back propagation learning operation, so that is clearly an advantage. So, that is why in case of VGG net, all the convolution kernels or are of size 3 by 3. And it is quite obvious that having multiple operation stage 8 operations with basic convolution kernel of size 3 by 3, I can have receptive field of size 5 by 5. If I have 3 subsequent operations, the receptive field will be 7 by 7; 4 sub subsequent operations, it will be 9 by 9, 5 subsequent operations, it will be 11 by 11.

So, variable size kernels, all variable size kernels which are used in AlexNet can be realized using multiple 3 by 3 kernels as just now we have demonstrated. And this realization is in terms of the size of the receptive field which is covered by the kernels. And this VGG 16 net gives a top error rate of around 7 percent. And if you compare this with the top 5 error rate that you get in case of AlexNet it is a significant reduction in the top 5 error rate.

(Refer Slide Time: 15:05)

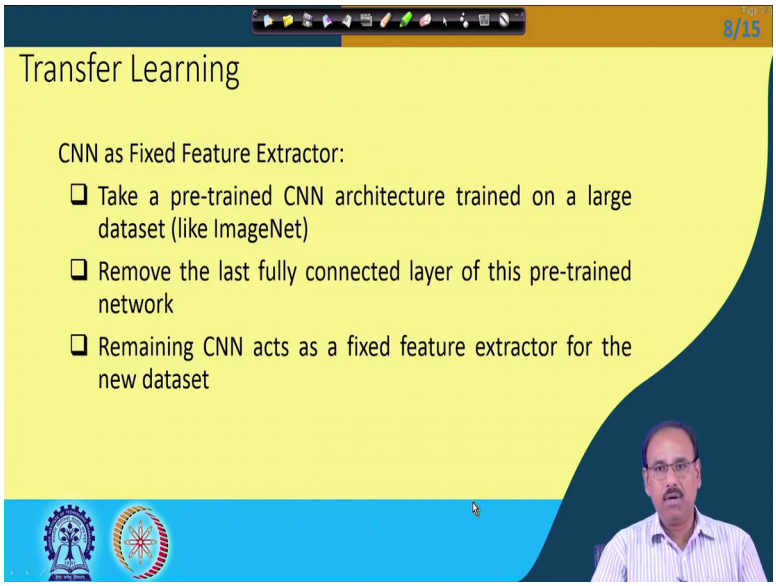


So, these are the features of VGG 16 network. And as we have just seen that with the depth of the neural network in case of deep neural network, the number of parameters which needs to be trained that increases tremendously. So, as a result, it is always advantageous that if we can take a pre trained network that is if we can borrow the parameters from a network which is already trained and then fine tune it for some other applications, then it will be more advantageous for training of the deep neural networks.

And transfer learning is an operation which helps you to have this parameters used for some application which was originally trained in some other application. So, effectively in this transfer learning, what you do is suppose I have a model which is trained on one set of data base. Say for example, I have a model which is trained on imagenet database, but I have an application which is on some other database, say for example, Pascal database.

So, it is possible that I can use the knowledge which has been acquired while training my model onto imagenet database. So, this knowledge can be transferred to another model which is to be used on Pascal database. And this is a concept which is known as transfer learning. And this transfer learning greatly enhances that greatly facilitates the training of deep neural networks. Now, why this transfer learning is possible?

(Refer Slide Time: 16:55)



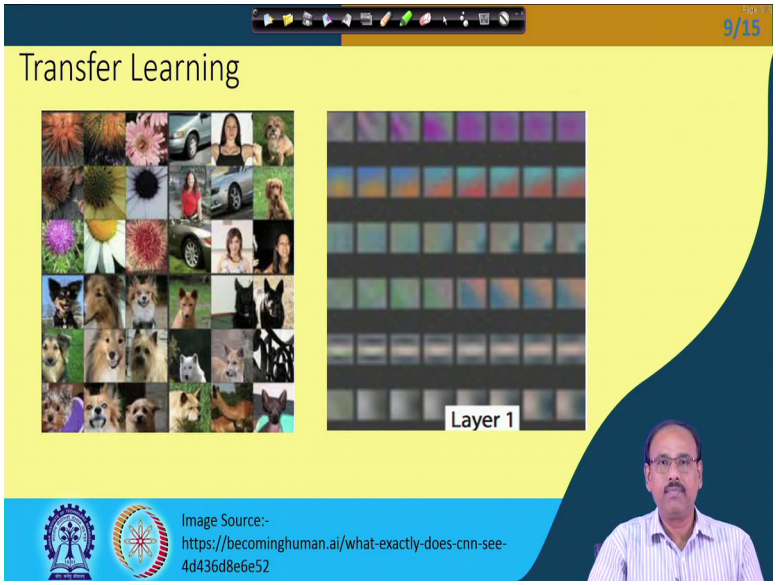
That is how is it possible that the knowledge that you gain in one domain is can be transferred to another domain. So, what you are basically doing is if you look at the features which are trained which are learned in the deep neural network that the features which are learned in the early layers of the network they are basically domain independent, they are more generic features. Whereas, features which are learned towards the later part of the deep learning of the deep neural network or towards the output side, they are mostly domain dependent.

So, as a result a CNN architecture which is pre trained on a large database like imagenet that can be used as an initial estimate of the parameters which can be used for other applications. And while doing so as we just said that the features learned in the early layers, they are mostly generic in nature. So, it is quite likely that the same set of features will also applicable in other applications. So, when you go for this pre training, what you can do is you can remove the fully connected layers because which are at the later part of your deep neural network or towards the output side of the deep neural network.

So, you can remove these fully connected neural networks from the pre training part ok. And if I remove that then the earlier parts of the convolution neural network, they simply act as feature or fixed feature extractors, because the same features may also be applicable for other application domain. Of course, at the finally what I need is a short of

fine-tuning; that means, end-to-end training, but while doing so, I already have some learned parameters which may be useful or which are quite useful.

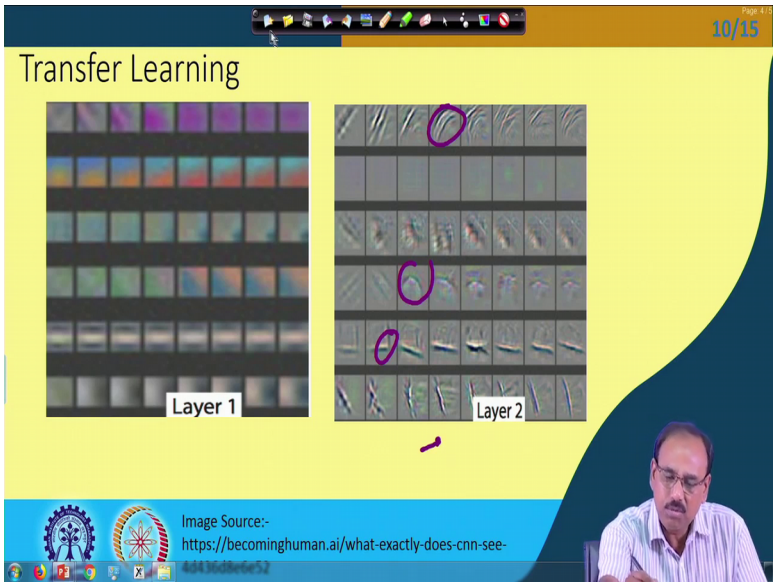
(Refer Slide Time: 18:57)



So, let us try to see that what are these different types of features which are learned in different layers. So, let us consider a scenario where we are training a deep neural network with a set of images as shown on the left hand side of this figure. Suppose, these are the images which are used for training of this deep neural network. And if you try to visualize the features which are learned in the first layer, in the first hidden layer or the first convolution layer, you will find that the features which are learned they are of this type of nature.

And it shows that the features which are learnt by the first layer in the convolutional neural network, they consist of say horizontal edges, vertical edges, diagonal edges and so on. So, these are mostly the features which are learned in the first layer.

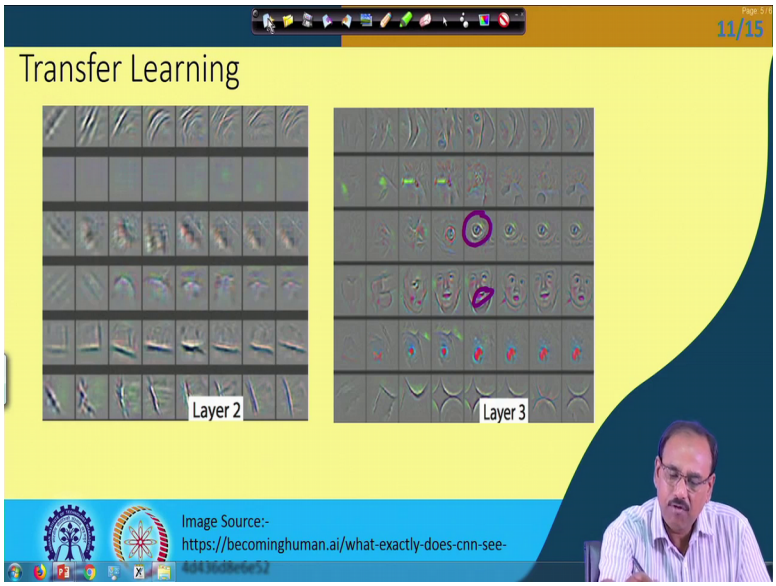
(Refer Slide Time: 20:11)



Now, if you move to the second layer, you find that the features which are learned in the first layer, they are learned further or you will try to learn the features in the second layer which are compositions of the features which are learned in the first layer. So, as shown in this figure, the features which are learned in the second layer you find over here they consists of say corners like this they consists of different curves here, here and so on.

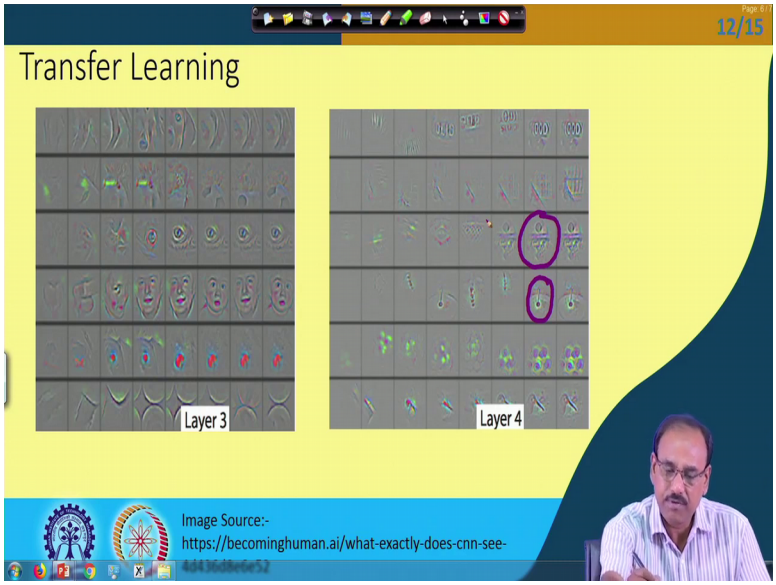
So, in the first convolution layer, the network learns mostly the edges, the vertical edges, horizontal edges, diagonal edges and so on. In the second layer, the network learns mostly the corners or compositions of this edges which are corners curves and so on.

(Refer Slide Time: 21:07)



As you move further or move deep inside the network you find that in the third layer, the features which are learned are basically compositions of the features which are learned in the second layer. So, here you find that you have features like lips; you have features like eyes, and so on, so which are features of the higher level of abstraction. And if you continue further, you will find that as I go to say fourth layer.

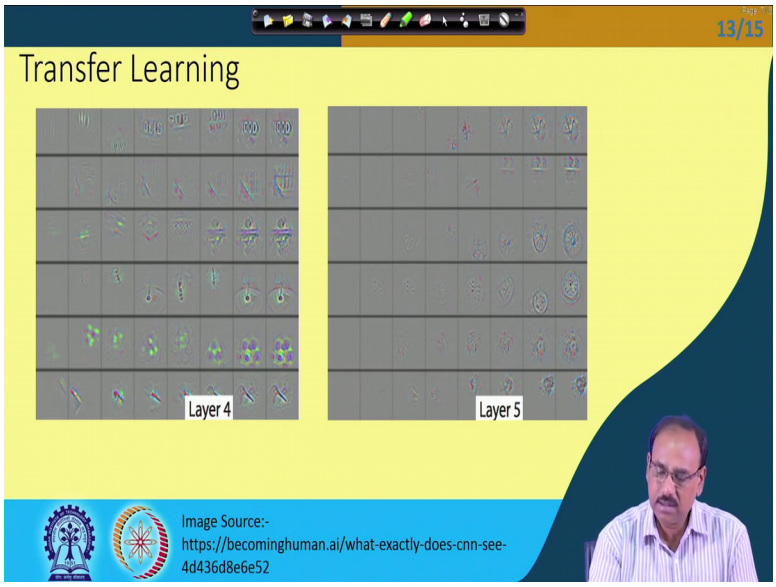
(Refer Slide Time: 21:35)



From here at the fourth layer you can see that the network learns say noses, it learns different types of say here maybe somewhere we have eyes, some contribution of face

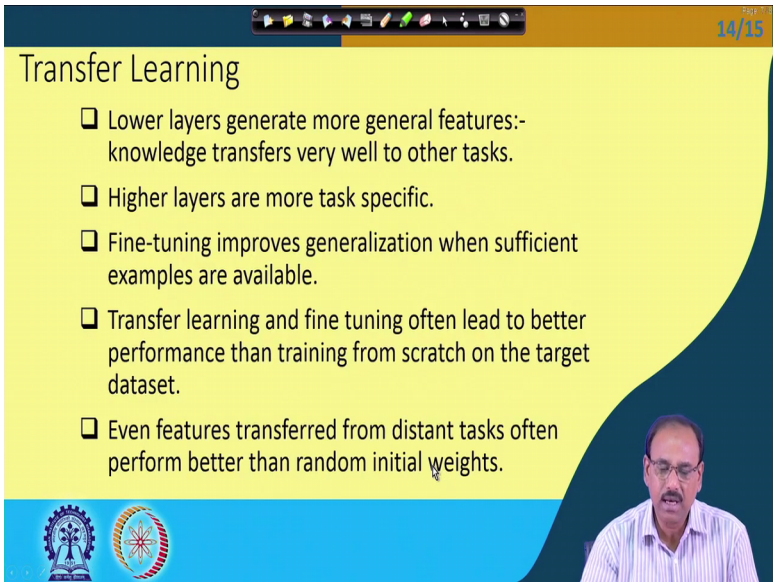
and so on. So, as you are moving in the deeper layers the features which are learned they become more and more domain specific, unlike in case of the features which are learned in the earlier layers which are more generic in nature.

(Refer Slide Time: 22:13)



In the same manner, if you move to the fifth layer, you find that the here the features learnt are say of cars of the wheels some faces and so on, so which are more specific to the application domain.

(Refer Slide Time: 22:27)

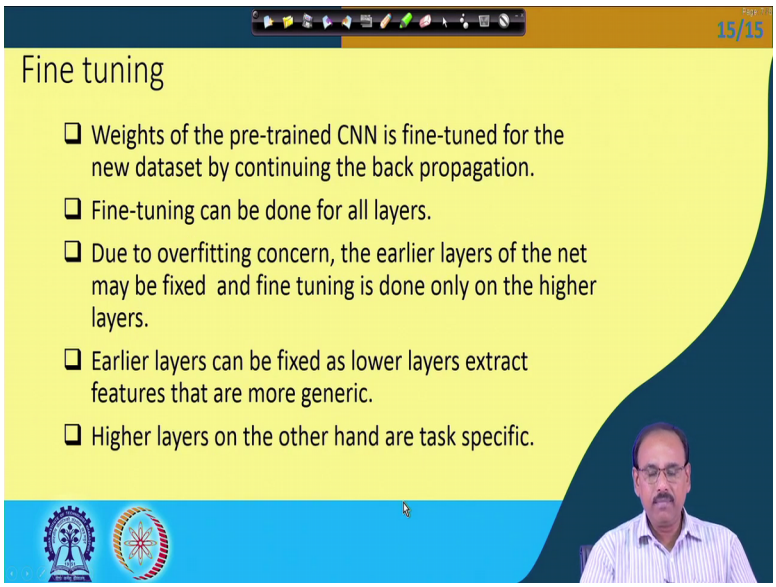


So, as a result of this as the features which are learned in the lower layers, they are more generic in nature. So, these features can be easily transferred or the knowledge gathered in the earlier layers can be easily transferred to some other application. And because the higher layers are more task specific or more domain specific, so using this higher layers or the features learned in the higher layers in other applications are difficult as a result what you need is some sort of fine-tuning, so that those domain specific features can be learned again.

So, in fact, it has been found that this fine-tuning that improves the generalization when sufficient examples of the new domain they are available. And the transfer learning and fine-tuning taken together, they often lead to better performance than training a neural network our deep neural network or from scratch on the new target data set. And even the features which are transferred from a domain which is not related to my target domain.

So, the features which are transferred from distant tasks they also very often perform better than random initialization of the weights, because whenever we train a neural network for initialization the weights of different layers are allotted or assigned at random. So, this is what is the concept of transfer learning.

(Refer Slide Time: 24:13)



And weights of the pre-trained convolution neural network is fine-tuned for the new data set. And when you are going for fine-tuning, obviously, the kind of algorithm that you

have to use is a back propagation algorithm. So, what you do is you take the parameters from a network which is already trained on some application. I have my new neural network which is set to be applied in some other application. So, to this new model, I feed my input data and I check what is the output error for this new input data.

And you try to reduce this output error by again gradient descent approach or back propagation learning approach. And while doing so you go for fine-tuning of the parameters which you have transferred from some other domain. When you do this fine-tuning this fine-tuning can be done for all the layers, but as we said that at the beginning or towards the early layers the feature learns features which are learned are mostly generic.

So, during fine-tuning we may not train the earlier layers, but what we will have to definitely train is the later layers particularly the fully connected layers, because the knowledge gained or the features learned in the fully connected layers or the later layers they are mostly domain specific whereas, the features learnt in the earlier layers they are mostly generic. So, during this back propagation learning, we can fix the parameters of the earlier layers and we can fine-tune the parameters of the later layers.

And as I said that the higher layers are task specific, so fine-tuning of the higher layer parameters is very very essential. So, with this I will conclude this part of the lecture. So, what here what we have learned is we have seen the architecture of the VGG 16 network, and we have also explained the concept of transfer learning where we can transfer or we can reuse the knowledge gained in one application in some other application.

Thank you.