

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

Lecture – 41
GoogleNet

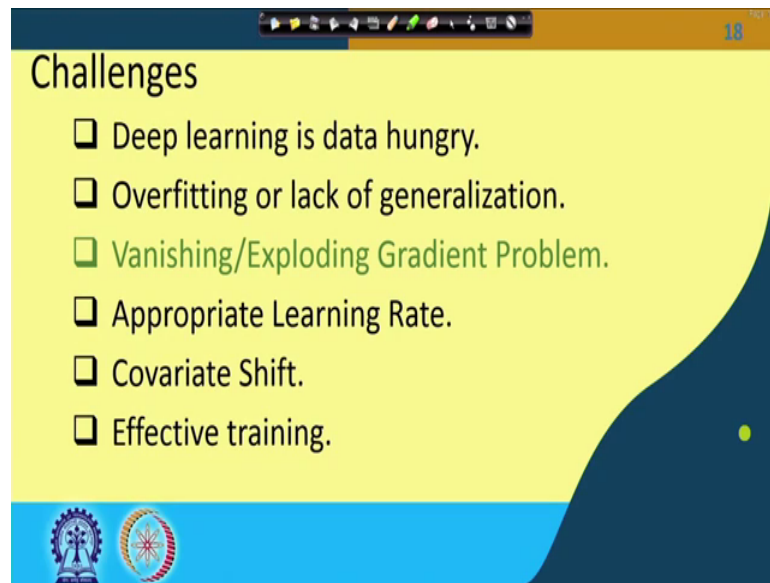
Hello, welcome to the NPTEL online certification course on Deep Learning. For last few lectures, we are discussing about the different popular convolutional neural network architectures. And while doing so, in our previous lecture we have tried to see that what are the challenges of a deep neural network and we were also discussing that how those challenges are addressed or can be addressed in different networks.

(Refer Slide Time: 01:05)



So, we have talked about the challenges in deep learning and let us just try to recapitulate that what are those different challenges that we have seen.

(Refer Slide Time: 01:17)



So, the first challenge that we have said is a deep learning is data hungry. In the sense, for training of the deep neural network you need a large volume of data maybe thousands and thousands or even lakhs of data for the training purpose and all those data have to be annotated data. However, practically it may not be possible to generate sufficient number of data which will be used for the training purpose.

So, in that case what you have to go for? We have to go for data augmentation techniques, so that even if you have the limited number of data by processing those data we can multiply the data volume. And the kind of processing can be that you can take different crops or different portions of the data which is taken from the data which is already available for the training purpose.

Or you can employ different types of distortions on the given training data, but keeping its custom maybe you can take the mirror imaging of it or you can modify the intensity you can modify the color and so on. So, that is how you can address the problem of generating voluminous data for training of the deep neural network.

The other kind of challenge or the problem which is faced in deep learning is over fitting of the model or lack of generalization of the model. As the number of parameters in the deep neural network are very high, so it is quite possible that while training the neural network will try to memorize the data which has been fed for the training and in the

process it does not really understand or really encode the structure or the features which are present in the data.

So, to avoid this over fitting problem there are different techniques which have been employed and we have discussed many of them in our previous lectures, one of them we have said is that when you try to train an auto encoder network one of the layers in the auto encoder network is a bottleneck layer and the purpose of using this bottleneck layer is that whenever you are training data or the information process through the network it passes to a constant region.

So, as a result the network does not or cannot simply memorize the training data, but it has to understand the structure or the features present in the training data. And that features can be used later on for recognition purpose or understanding purpose. Similarly, other kind of approaches to tackle this over fitting is having noisy auto encoder for the training data, you add noise to the training data, so that the auto encoder try to tries to understand what is the inherent structure or salient features which are present in the training data and using that it tries to recognize the image or it tries to understand the image.

So, there are different such approaches by which the over fitting or lack of generalization that can be avoided. One of them is by having the generalization loss, incorporating a generalization loss in the loss function where the generalization loss can be L 2 norm of the weight vectors which are there in your neural network. So, there are various approaches in which the over fitting can be addressed and we have discussed about those addresses in our previous lectures.

The other problem that we have said is the vanishing or exploding gradient problem. And we have also said that this problem becomes very severe as the depth of the neural network becomes very large. The reason for this is that when you train a neural network or when you tune or update the parameters of the neural network, it is the gradient of the output error which is considered for updation of the parameters or the weights in the neural network.

And why you take the gradient is using the gradient descent approach you try to minimize the output error with respect to the different parameters and while doing so you update the parameters in such a way that the output error is reduced.

(Refer Slide Time: 06:07)

Vanishing Gradient Problem

$X \xrightarrow{W_1} f_1 \xrightarrow{W_2} f_2 \xrightarrow{W_3} f_3 \xrightarrow{W_4} f_4 \rightarrow O$

$$\frac{\partial O}{\partial W_1} = X \cdot f_1' \cdot W_2 \cdot f_2' \cdot W_3 \cdot f_3' \cdot W_4 \cdot f_4'$$

Handwritten notes: $0 \rightarrow \sigma^1$, $1 \rightarrow \sigma^2$, $2 \rightarrow \sigma^3$, $3 \rightarrow \sigma^4$

So, if I look at this next slide which of course, we have discussed in our previous lecture that I have taken a very simple multilayer neural network, where every layer consists of one node and input is also a scalar, say input is a scalar X and because every layer contains only one node, so the activation that you get from a every layer is also a scalar.

So, going by this your final output O over here, this output O will be basically the function f_4 of W_4 multiplied by f_3 , f_3 of W_3 multiplied by output of f_2 , where f_2 is W_2 multiplied by the output of f_1 , where f_1 is the function of W_1 times X and each of these functions f_1, f_2, f_3, f_4 , they are basically non-linear activation functions.

So, if I wanted to update this input vector this weight vector W_1 , so that the error at the output is reduced or minimized then I have to make use of the gradient descent approach or I have to find out the gradient of the output error with respect to W_1 . Similarly, for updation of W_2 , I have to find out the gradient of the output layer with respect to W_2 and so on and when you try to take the gradient of the output layer our term comes which is gradient of output with respect to the corresponding parameter.

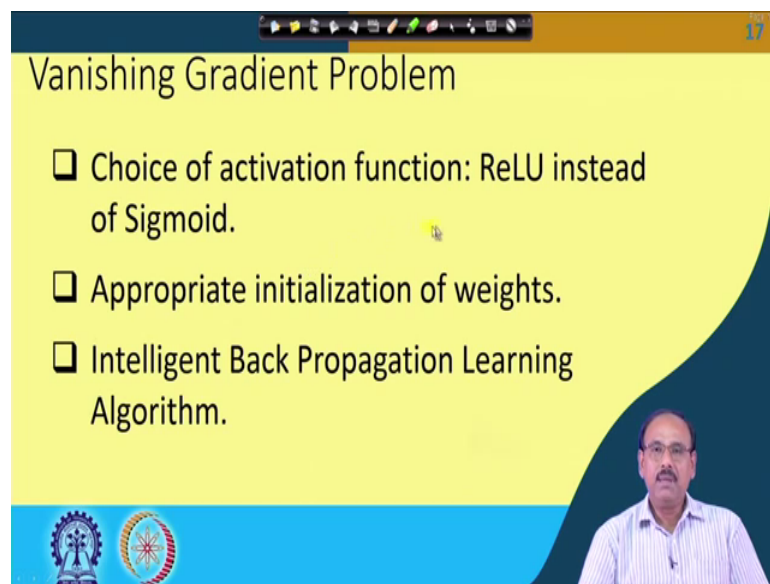
So, suppose in this case we are trying to update the value of the input weight of the parameter W_1 . So, I have to compute what is $\frac{\partial O}{\partial W_1}$. And if you compute in this following this chain relation in this function you find that $\frac{\partial O}{\partial W_1}$ will be $X \cdot f_1' \cdot W_2 \cdot f_2' \cdot W_3 \cdot f_3' \cdot W_4 \cdot f_4'$. And the number of such terms in this product will go on increasing with the depth of the network.

If it is 5, if the depth of the network is 5 then two more terms will appear in there will appear in this expression that is W_5 and f'_5 and so on.

So, if you study this product which is nothing, but the gradient of the partial derivative of the output with respect to your input parameter with respect to the parameter W_1 , you find that there are various terms which effects this gradient and one of the terms are f'_2 , f'_3 , f'_4 and the derivatives of that. That means, the nature of the non-linear activation function used in these neural networks they have a say on the value of this derivative. And the other factors are the weights W_2 , W_3 , W_4 and so on. These weights also decide what will be the value of $\frac{\partial O}{\partial W_1}$.

So, naturally, the nature of this non-linear activation functions that decides what is what will be the value of this product.

(Refer Slide Time: 09:46)



The slide is titled "Vanishing Gradient Problem" and lists three solutions:

- Choice of activation function: ReLU instead of Sigmoid.
- Appropriate initialization of weights.
- Intelligent Back Propagation Learning Algorithm.

The slide also features a video inset of a man speaking in the bottom right corner and logos of institutions in the bottom left corner.

That is the reason that in case of deep neural network instead of using sigmoidal function as a non-linear activation the non-linear activation function which is mostly used in deep neural network is rectified linear unit, because if I take sigmoidal function then the maximum derivative of the sigmoidal function as we have told in our previous class is given by $\frac{1}{4}$, which is already less than 1.

And in this product term you find that if the all the terms are less than 1, then the final product will be very very less than 1, which will be almost 0, negligible. And if it

becomes almost 0 then when you try to update W_1 as $W_1 - \Delta W_1$ that ΔW_1 terms will almost vanish and as a result the W_1 will not be updated at all.

So, the nature of this non-linear function f that is very very important and why you use ReLU is the derivative of ReLU is equal to 1, when as long as its argument are greater than 0, right. So, that is why ReLU is preferred as the non-linear activation in case of deep neural network. The other terms are the weights W_1, W_2, W_3 . So, naturally you find that if all of them are ReLU non-linear activation functions that ReLU do in that case this product becomes $W_2 \times W_3 \times W_4$ because for ReLU the derivative of non-linear function is 1.

And here if W_2, W_3, W_4 all of them are less than 1, so let us assume they are 0.5. So, it will be 0.5 into 0.5 into 0.5. So, the product will be very very less than 1. And that is what is vanishing gradient problem because as you are moving towards the earlier layers, the value of the gradient goes on reducing exponentially. On the other hand, if the weights are greater than 1 then all of them multiplied together increases the gradient exponentially and that is what is gradient explosion problem.

So, in one case it is vanishing gradient problem, in other case it is exploding gradient problem and both of them are bad for training of the deep neural network and that is the reason it has been found that in many cases the deep neural network performance is worse than its shallow counterpart.

So, to address this problem either we have to choose the weights appropriately, so there we have said that when you choose the weights at a random initially for if a node receives the input from say N number of nodes. So, if I have a situation something like this. So, I have a node over here in a certain layer and it receives input from N number of nodes then these weights are to be decided at random where the mean of the weights will be 0 and the standard deviation of the weights or the variance of the weights have to be $\frac{1}{N}$.

So, this is the mean μ and this is the variance σ^2 of the weight components that you want to assign at this layer. And in some cases instead of one the variance $\frac{1}{N}$, the variance is also taken to be $\frac{1}{N}$. And that is rule of thumb how you decide or how you select the weights at random to the connection weights at any of the layers.

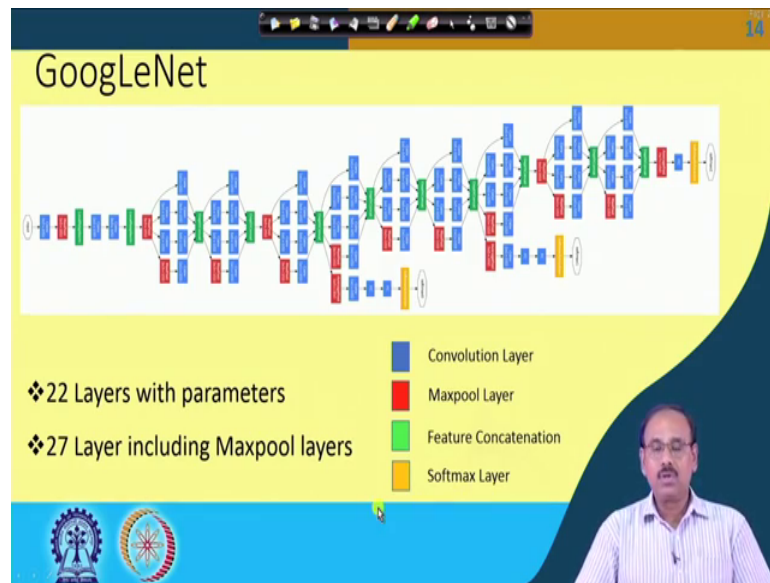
And the other possibility, the other approach to tackle this vanishing gradient problem as we said earlier is by taking intelligent back propagation learning algorithm; that means, by having a proper architecture or the algorithm to be used for that professional learning. So, this is how what we are going to discuss today. That will be discussing about two architectures, one is GoogLeNet and the other one is ResNet or residual network and we will try to see that how this vanishing gradient problem are addressed in GoogLeNet as well as ResNet.

(Refer Slide Time: 14:09)



So, firstly, let us talk about GoogLeNet. In fact, in our previous one of our previous lectures we have talked about VGGNet that was particularly VGG 16 and we have said that VGG 16 was the first runnersup in the visual recognition challenge 2014 and GoogLeNet was actually the winner of ILSVRC that is image net large-scale visual recognition challenge 2014.

(Refer Slide Time: 14:47)



So, let us see what this GoogLeNet is. So, here this diagram shows the architecture of the GoogLeNet where every blue box represents a convolution layer, every red box represents maxpool layer, the green box is represent the feature concatenation all the features that you compute from the convolution layers or after the maxpool layers they are concatenated together to give a feature map.

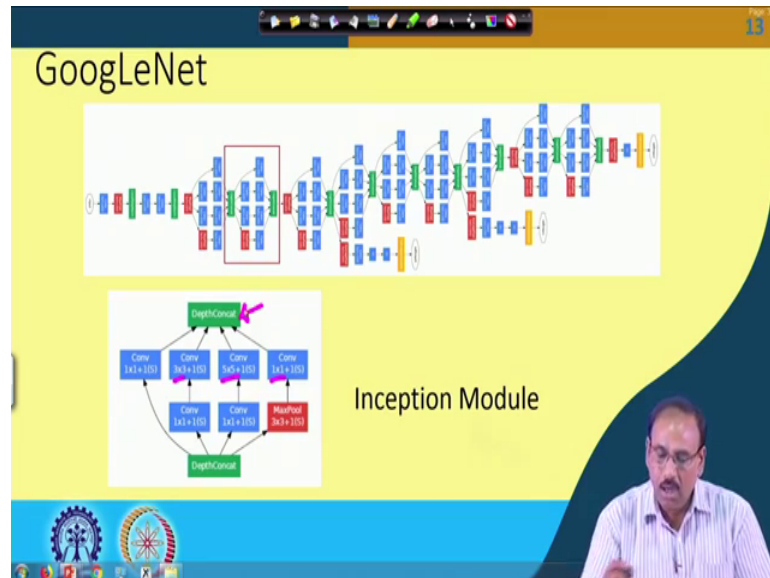
And at the output of the classifier at the final after the fully connected layer fc layer you have the softmax layer which is used for classification. And as before the number of nodes in the soft max layer is 1000 one bar image category in the image net database. And if you look at this GoogLeNet architecture you find that there are 22 layers with parameters.

So, these layers are actually convolution layers and fully connected layers. And if you also consider the Maxpool layers; obviously, the Maxpool layers does not have any tunable parameters then the total number of layers that you have in the GoogLeNet is 27 and you can compare the architecture of GoogLeNet with the VGGNet, right. So, this is what is GoogLeNet architecture.

And if you look inside GoogLeNet architecture you find that there are a number of units which are almost repetitive. Say for example, this is one unit, this is another unit, this is another unit and so on. So, there are 9; 1 2 3 4 5 6 and the yeah there are 9 such units in

the Google Network and these units are almost identical. So, these units are actually inception units they are called as inception units.

(Refer Slide Time: 16:49)

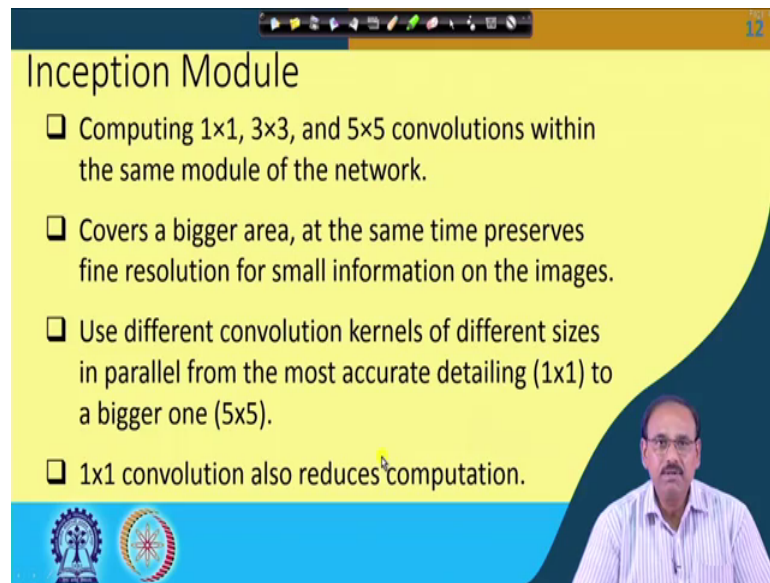


So, let us proceed further. So, what are these inception units or inception modules? You find that this inception units or inception modules, they actually compute under the features which are used in the Google Network. And if you look at the way they work or the number of different types of convolution units they have convolution kernels of size 3 by 3, convolution kernels of size 5 by 5 and also convolution kernels of size 1 by 1.

So, GoogLeNet actually introduces a 1 by 1 convolution kernel and we will see that what is the use or what is the advantage that you gain using this 1 by 1 convolution kernel. So, this is what is the inception unit. You have the different completion layers, you have the Maxpool layers and finally, outputs of all these different convolution channels or feature, convolutions are actually the operations to extract the feature state that is what we have seen earlier. So, finally, in this inception module the different features which are computed by different convolution kernels they are stacked together to give you a feature map.

So, here what we have is depth concatenation, all the feature maps given by different convolution channels or different feature channels they are stacked one after another to have the final feature map. So, let us proceed further to see that what are the things that we have in this inception module.

(Refer Slide Time: 18:25)



The slide is titled "Inception Module" and contains the following text:

- ❑ Computing 1x1, 3x3, and 5x5 convolutions within the same module of the network.
- ❑ Covers a bigger area, at the same time preserves fine resolution for small information on the images.
- ❑ Use different convolution kernels of different sizes in parallel from the most accurate detailing (1x1) to a bigger one (5x5).
- ❑ 1x1 convolution also reduces computation.

The slide also features a small video inset of a man speaking in the bottom right corner, and logos of institutions in the bottom left corner.

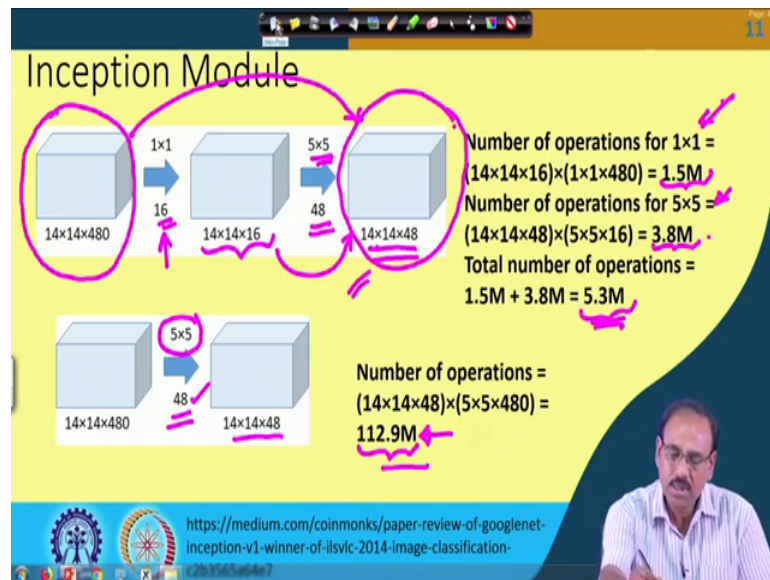
So, this inception module has 1 by 1 convolution kernel, it has 3 by 3 convolution Kernel and it has 5 by 5 convolution kernel and all these convolution kernels are within the same inception module of the network. And this is a concept which is called network inside network. So, I have inception networks or inspection modules which are networks by themselves, which are within the inter entire GoogLeNet.

Now, when I have larger convolution kernels, larger convolution kernels cover a larger area to compute the features. So, when it is 5 by 5 convolution kernel, it covers a receptive field of 5 by 5 pixels to give you the features within those 5 by 5 pixels. Similarly, if you use 1 by 1 conversion kernel, then 1 by 1 convolution kernel actually gives you the further details which are present in the image, ok.

So, this 1 by 1 convolution kernels that gives you the most accurate features, whereas if you use the convolution kernel size you get a feature which is over a trend or over a larger area of the image. So, as you vary the convolution kernel size say 1 by 1, 3 by 3 to 5 by 5 you are actually extracting the features at different scales starting from the finest features in 3 by 3 to the cores features given finest features given by 1 by 1 convolution kernel to the course features given by 3 by 3 and 5 by 5 convolution kernel.

When you use 1 by 1 convolution kernel that also reduces the amount of computation that you have to perform. Let us see how this 1 by 1 convolution kernel reduces the amount of computation that you have to perform

(Refer Slide Time: 20:23)



So, let us try to say this, suppose I have an input feature map which is of size 14 by 14 by 480, so there are 480 different feature maps or the channels which are concatenated together and size of every feature map is of size 14 by 14. And from there I wanted to extract features which are again 14 by 14 by 48; that means, size of every feature map remains the same which is 14 by 14, but the number of channels are reduced to 48. So, this can be done in two ways, either use it directly by using 5 by 5 kernels and 48 number of such kernels of course, with stride.

So, each of this 5 by 5 kernels gives you with stride and appropriate padding gives you a feature map of size 14 by 14 by 1. Each of these 48 channels each of these 48 kernels will give you a feature map of size 14 by 14 by 1, as I said with appropriate stride and the corresponding padding operations. And each of them gives you one feature map. So, when I have 48 number of such kernels I get 48 feature maps you stack them together you get 14 by 14 by 480 number of kernels.

The other way is that suppose this is just one of the ways this can be done, there are numerous possibilities in which the same can be implemented. So, one of the approaches first let us assume that we will have 1 by 1 kernel and each of this 1 by 1 kernel will give you 14 by 14 feature maps, 14 by 14 by 1 feature maps and if I use 16 such kernel I will get a feature map of 14 by 14 by 16, by stacking the outputs of outputs which are maps computed by each of the kernels.

And then from here at the second level I can have 5 by 5 kernels, so each of the 5 by 5 kernels will map this feature map 14 by 14 by 16 to 14 by 14 by 1 of course, with proper stride and proper padding that we have already discussed before. So, each of this 5 by 5 kernel is giving me a feature map of size 14 by 14 by 1. If I use 48 number of such kernels then I have 48 such feature maps each of size 14 by 14 by 1, you stack 48 such feature maps together you get a feature map of 14 by 14 by 48. So, you find that here also I get 14 by 14 by 48, here also I get 14 by 14 by 48.

Now, if you compute the number of operations that you have to perform in a both of these options you will find that, when I use this option that is directly from 14 by 14 by 48 to 14 by 14 by 48 using 48 kernels each of size 5 by 5 the total number of operations that I have to perform is 14 into 14 into 48 into 5 into 5 into 480 that gives you around 112.9 million operations.


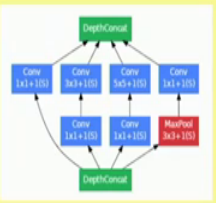
Whereas, if I first use this 1 by 1 kernels followed by 5 by 5 kernels for the number of operations that you perform with 1 by 1 kernels is around 1.5 million, you can compute this to verify and subsequently with 5 by 5 kernels the number of operations is around 3.8 million. So, the total number of operations in this option becomes this plus this which gives you total 5.3 million operations.

So, you compare this with this for direct conversion or direct mapping, I need 112.9 million operations whereas, using initially this 1 by 1 convolution layers convolution kernels the total number of operations comes out to 5.3 and which is obviously, much less compared to one order less it is just 5.3 against 112.9. So, your computation has been reduced just again. So, this is one of the advantage that you get using 1 by 1 convolution layers.

(Refer Slide Time: 25:32)

Inception Module

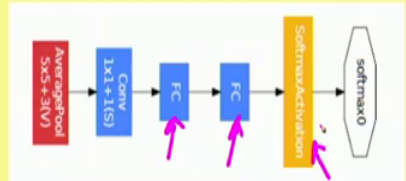
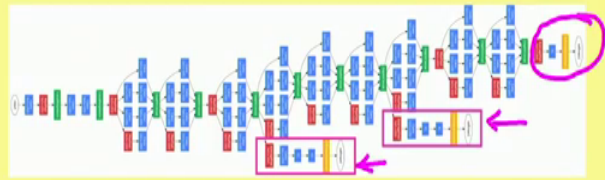
- ❑ Outputs of these filters are then stacked along the channel dimension.
- ❑ Multi-level feature extractor.
- ❑ There are 9 such inception modules.
- ❑ Top-5 error rate of less than 7 %.




So, once you do that then finally, what you have is the output of these filters each of these channels are then stacked together to give you the final feature map. So, every inception module is acting as a multilevel feature extractor and you see there are 9 such inception modules and this GoogLeNet gave a top 5 error rate of less than 7 percent which is of course, marginally better than what you have got in case of VGGNet. So, this is what the Google Network.

(Refer Slide Time: 26:11)

GoogLeNet



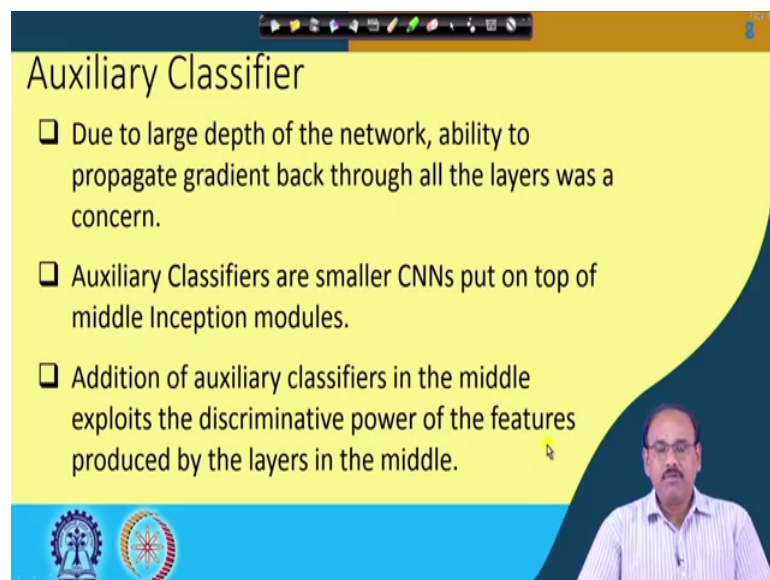
Auxiliary Classifier



Now, let us just see that how this GoogLeNet tries to tackle the vanishing gradient problem. So, as we said that the fun final layers in the GoogLeNet which is over here they are actually classification layers and you look at that this GoogLeNet gives you two more classifiers which are known as auxiliary classifiers which are added in the middle layers and these are the auxiliary classifiers.

So, this auxiliary classifiers have this fully connection, fully connected layers, it has the softmax activation layer then output of the softmax activation layer is actually a classifying output. So, what is the purpose of having this auxiliary classifiers?

(Refer Slide Time: 27:00)



The slide is titled "Auxiliary Classifier" and contains the following text:

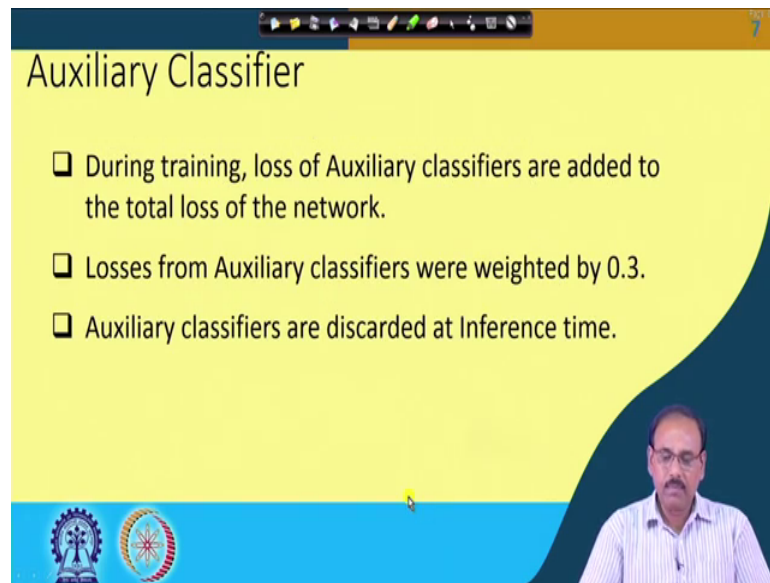
- ❑ Due to large depth of the network, ability to propagate gradient back through all the layers was a concern.
- ❑ Auxiliary Classifiers are smaller CNNs put on top of middle Inception modules.
- ❑ Addition of auxiliary classifiers in the middle exploits the discriminative power of the features produced by the layers in the middle.

The slide also features a video inset of a man speaking in the bottom right corner and two logos in the bottom left corner.

You will find that as we have told before that you have the vanishing gradient problem because of the depth of the network while training as we try to propagate the error or the gradient towards the early layer of the network the gradient almost vanishes. And this auxiliary classifiers are basically smaller convolution networks CNNs which are put on top of middle inception modules.

So, this addition of auxiliary classifiers in the middle layers in the on top of the middle of inception modules actually tries to exploit the discriminative power of the features produced by the layers in the middle. So, as they are classifiers they will also give a classification error and these classification errors are computed from the middle level inception modules.

(Refer Slide Time: 27:57)



The slide is titled "Auxiliary Classifier" and contains the following text:

- ❑ During training, loss of Auxiliary classifiers are added to the total loss of the network.
- ❑ Losses from Auxiliary classifiers were weighted by 0.3.
- ❑ Auxiliary classifiers are discarded at Inference time.

The slide also features a video inset of a man speaking in the bottom right corner and logos of institutions in the bottom left corner.

So, while back propagation when you are performing the back propagation learning, the loss computed from this auxiliary classifiers are added to the total loss of the network and of course, these auxiliary layer outputs the lost on the auxiliary layer outputs are scaled by a factor of 3 and then added to the final loss functions. And because of this, because you are computing the loss from the middle layers and adding to the final loss function and that is being used during the back propagation operation.

So, this is expected that the vanishing gradient problem will be solved to some extent. Of course, this auxiliary classifiers are used only during the training of the network. Auxiliary classifiers are not used at the test time or at the inference time. So, we have seen that in case of GoogleNet what is the architecture of the GoogleNet and using auxiliary classifiers how the GoogleNet tries to tackle the problem of vanishing gradient.

So, in our next class we will talk about the race net or residual network and we will see that how this residual network also tries to address the vanishing gradient problem. So, we will stop here today.

Thank you.