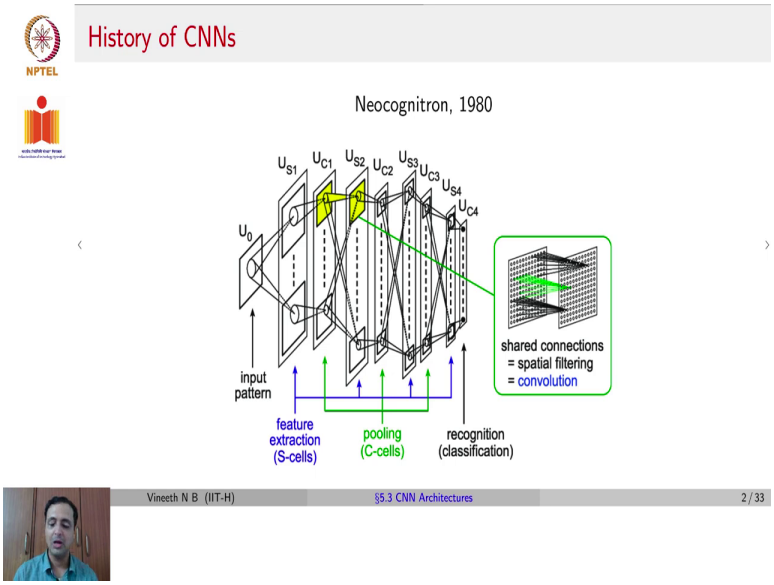
Deep Learning for Computer Vision
Professor. Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture No. 34
Evolution of CNN Architectures for Image Classification - Part 01

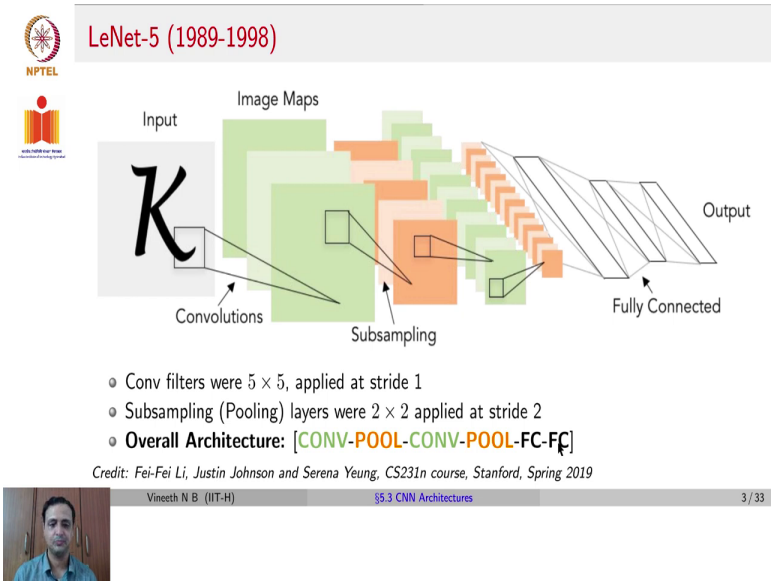
Having seen an introduction to convolutional neural networks, and how back propagation works in convolutional and pooling layers, let us now move forward and see how CNN architectures have evolved over the years. Today depending on what task you have at hand, and what constraints you have for a particular task, there are a wide variety of architectures that are available; we will try to cover a significant number of them over the next couple of lectures. Let us start with the earliest like earliest architectures.

(Refer Slide Time: 0:53)



The history of CNNs dates back to the Neocognitron in 1980, which was perhaps the earliest precursor of CNNs. The concept of feature extraction, the concept of pooling layers as well as using convolution in a neural network; and finally having recognition or classification at the end was proposed in the Neocognitron.

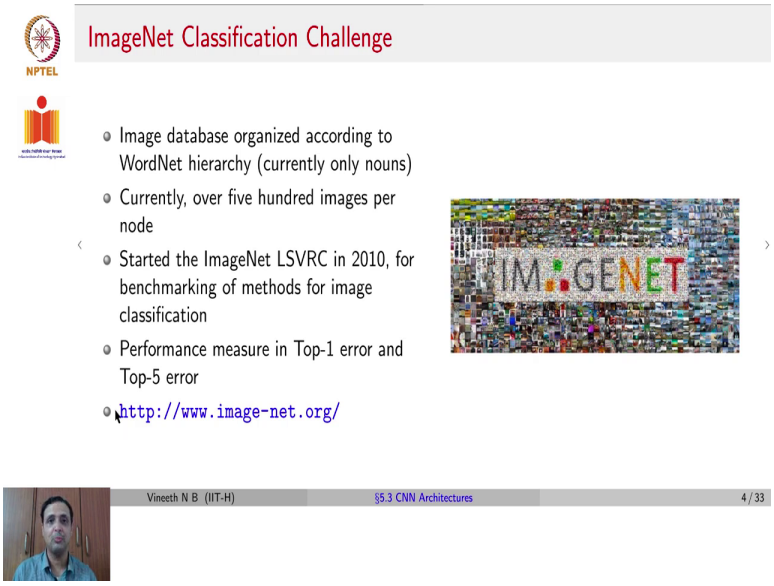
(Refer Slide Time: 1:20)



But, the name convolutional neural networks came with the design of the LeNet by Andre Kuhn and team. Which was largely developed between 1989 and 1998 for the handwritten digit recognition task. The LeNet convolutional neural network architecture looks like what you see here. It starts with a convolutional layer, then goes for a pooling layer, then again a convolutional layer, then a pooling layer; then to fully connected layers and then finally the output layer.

The convolutional filters were of the size 5x5 , applied at stride 1. The pooling layers were 2 cross 2, applied at stride 2. You can see the overall architecture written this way. This is a common way of writing about the architectures of neural networks; where you say CONV-POOL-CONV-POOL-FC-FC. It denotes a convolutional layer followed by a pooling layer, followed by a convolutional layer, followed by a pooling layer, FC layer and FC layer. Even architectures until today follow similar principles in their design of CNN architectures.

(Refer Slide Time: 2:40)

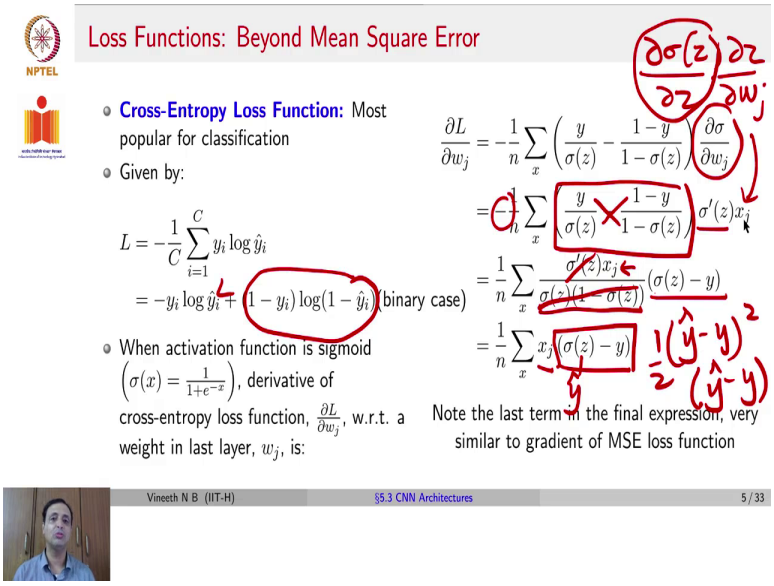


A major effort in development of newer architectures for CNNs and newer designs of architectures was thanks to the ImageNet classification challenge; which started in 2010. ImageNet is a dataset and they organized the challenge called the large scale visual recognition challenge in 2010. Image database in ImageNet is organized according to the WordNet hierarchy. It currently has images corresponding to various norms in the WordNet hierarchy. It has millions of images at this time and currently about 500 images per node in the WordNet hierarchy.

This led to a significant effort across researchers to benchmark their machine learning models, computer vision models in particular for image classification on a common dataset in which everybody could check how their models performed. The performance measure on the dataset was known as the top-1 error; which is the error, when you look at the highest the best prediction of the neural network against the ground truth; which is known as the top-1 error.

And the top-5 error, where you try to check if the ground truth was in the top-5 of the labels in the output layer of the neural network; which is still a good measure. Because the ImageNet dataset has about 1000 objects, which means your output layer in the neural network has 1000 neurons. As long as your ground truth label is in the top-5 of the values, the scores that you get in the output layer, you consider this a correct match and not doing so would give a top-5error. For more details on the ImageNet dataset which is very popularly followed by computer vision researchers today, please see this link.

(Refer Slide Time: 4:46)



Before we move forward and discuss CNN architectures, let us take a step back and see how neural networks in general can be used for the classification problem. When we discussed feed forward neural networks, we took the loss function as the mean square error; which was fine for that time. But, when you are working on a classification problem, mean square error may not be the right choice of a loss function. Mean square error works better for regression problems.

In classification problems, the most common loss function is known as the cross-entropy loss function which measures the entropy between the probability distribution on the output labels in the ground truth, versus the probability distribution on the output labels as output by the neural network. This gives you a sense of error committed by the neural network; let us see the formal definition. So, the cross-entropy loss function is given by $-\frac{1}{C}\sum_{i=1}^{C} y_i log\hat{y}_i$, where C is the total number of classes, $\hat{y}_i$ hat is the prediction for a particular class I of a neural network or the score of the last layer of the neural network for the ith class, and y_i is the correct label for that particular class.

Rather, you can say that y_i is if you had a certain number of classes, let us say 1 to C; let us assume the second class was the correct class. Then you would have your y vector expected y vector to be [0, 1, 0, …, 0, 0, 0] until this C^{th} class. This would be the probability distributions

on the label space in the ground truth; because this is the correct class. So, this would be y_2, this would be y_1 and so on and so forth. That is the $\hat{y}_i$ we are talking about here, and $\hat{y}_i$t is the score of the output layer of the neural network. In the binary case this simplifies to $-y_i log\,\hat{y}_i + (1 - y_i) log(1 - \hat{y}_i)$.

For example, when you use a sigmoid, you will have only one score that comes out of a neural network. And $1 - y_i$ will be the score for the negative class. So, this is how cross-entropy will simplify for a binary setting binary classification setting. Now, if we introduce a new loss function, the first thing to check is its differentiable; by looking at it one can say that cross-entropy is definitely differentiable. And then what are its gradients with respect to a say weight in the last layer. Because once we get that we can propagate it further through earlier layers, through whatever we have discussed so far.

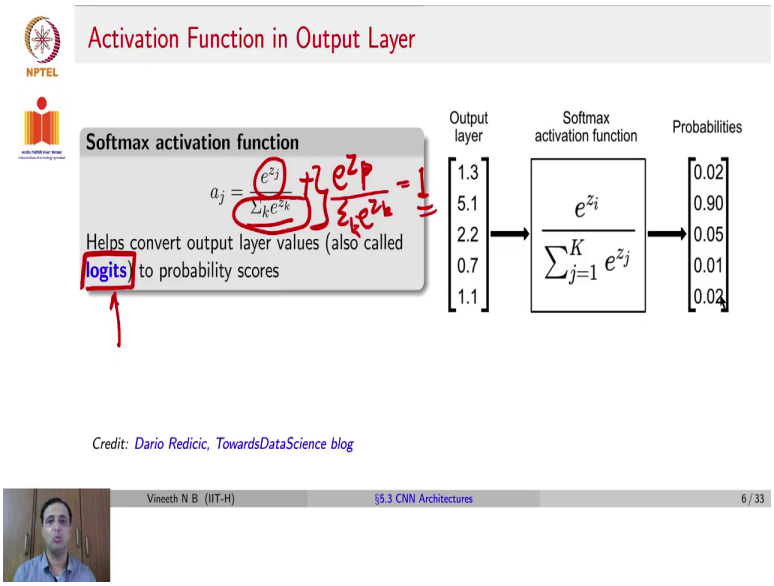
So, if we consider the activation function to be sigmoid, which is let us assume a binary classification problem. The derivative of the cross-entropy loss function with respect to a weight in the last layer w_j is given by

$$\frac{\partial L}{\partial w_j} = -\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{(1-\sigma(z))}\right)\frac{\partial \sigma}{\partial w_j} = -\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{(1-\sigma(z))}\right)\sigma'(z)x_j = \frac{1}{n}\sum_x (\sigma(z) - y)$$

remember $\sigma(z) = \hat{y}$.

So, even in mean square error when you have $(\hat{y} - y)^2/2$, if you differentiate, you would get $(\hat{y} - y)$ in the loss. So, the gradient of the cross-entropy looks very similar to how the gradient of the mean square error looks.

(Refer Slide Time: 11:16)



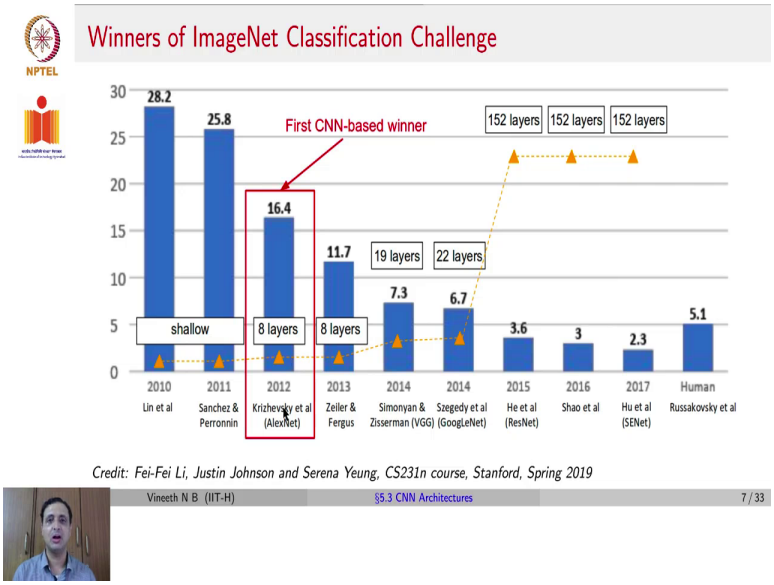
One more point before we move to CNN architectures is when you are doing classification. While, we talked about different kinds of activation functions; such as sigmoid, Tanh, ReLU, leaky ReLU so on and so forth. We typically use what is known as the softmax activation function in the output layer alone. What is the softmax activation function? Given $z_1 to\ z_k$ in the last layer, let us say they are all the outputs of the c neurons in the last layer. The final output of the network is given by $\frac{e^{z_j}}{\sum_k z_k}$. What does this do?

Remember in this case, the summation of all the e^{z_k} in the last layer will always be greater than e^{z_k} which means these values are definitely going to lie between 0 and 1; also because the denominator is the summation of all those exponential terms, exponential values of those neurons in the last layer. Doing this for every single neuron will also add up to 1.

In a way now we have converted whatever scores, the neural network outputs into a vector of probability values; where each is lying between 0 and 1, and they all add up to 1. This allows us to interpret the output of a neural network as a probability distribution over your output layer space. So, softmax is a very is a default activation function; that is used in the output layer of neural networks used for classification. A small notation or a nomenclature here the value is before the probability score are typically called the logits of a neural network.

So, the neural network outputs logits, you apply a softmax activation function; and convert it into probability scores. Here is an example on the right; you see the output layer gave a set of real values as output. By doing the softmax activation function, this became [0.02, 0.9, 0.05, 0.01 and 0.02] This is because of the exponential scaling of the values, and these add up to 1; allowing us to interpret that vector as a distribution over your output labels.

(Refer Slide Time: 14:18)



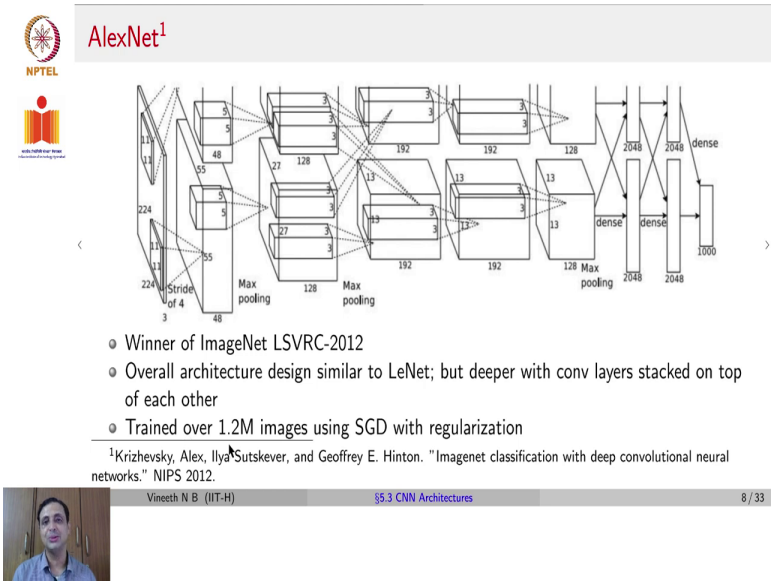
Let us now come back to the ImageNet classification challenge. So, all these CNNs that will talk about now all of them use cross-entropy loss as the loss, and more or less use the softmax activation function in the last layer as a common choice of activation function and loss across these CNN architectures. As we said a few minutes ago, the ImageNet dataset was created and launched in 2010, and researchers around the world participated in the ILSRVC challenge. And in 2010, the winning error rate was 28.2; this was done without neural networks. And in 2011 researchers improved the score from 28.2 to 25.8 error rate.

And in 2012, Alex Krizevsky and Geoffrey Hinton came up with CNN architecture popular to this day as AlexNet, which reduced the error from 25.8 to 16.4 which was a significant improvement at that time. So, this was the first winner of the ImageNet challenge which was based on a CNN, and since 2012, every year's challenge has been won by a CNN; significantly outperforming other deep other shallow machine learning methods. So, the traditional machine

learning methods are often called shallow learning; because you do not have deep learning over many many layers.

But this will become clearer to us in the next week's lectures, when we try to understand what value the depth of the neural network brings to the architecture. Let us now start with the AlexNet's architecture and try to understand it.

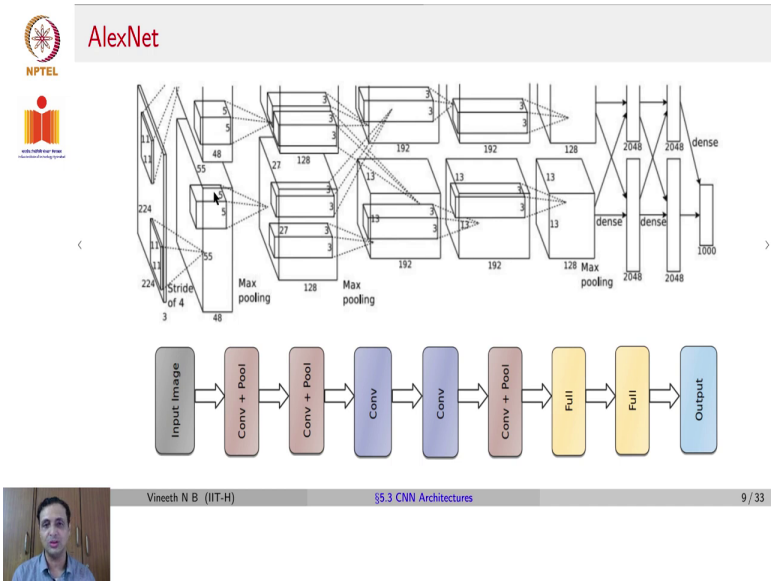
(Refer Slide Time: 16:22)



Here was the architecture of AlexNet; we actually looked at the first layer of AlexNet as an example when we worked out sizes in the first lecture on CNNs. The input is a volume which is given by 224x224x3, which is the RGB channel. And the filter size was 11x11, of course a depth of 3 which follows in all of these scenarios. And with a stride of 4, this led to a 55x55x96 of which 48 filters went to one of the processing units. And the remaining 55x55x48 went to another processing unit. We will try to describe this in more detail over the next few slides.

But, this was the overall architecture that won the LSVRC in 2012. It had designed principles similar LeNet. But, it also had many features which were different and it was also deeper than LeNet and there were also a few other important improvements which we will talk about. It trained over 1.2 million images using SGD with regularization.

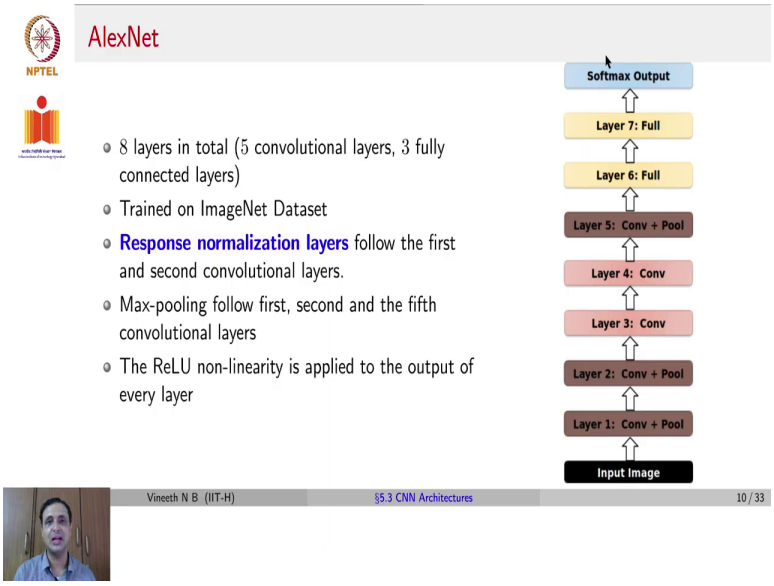
(Refer Slide Time: 17:54)



Let us try to look at this architecture more carefully. So, it has the input image which was 224x224x3, which is the size of images in the ImageNet dataset. The first layer had Conv plus Pool, which you see the output together here. So, this output of 55x55x48 was after doing the convolution part and then you do a max pooling, which reduces the size further. Then once again they had a Conv plus Pool, then only a Conv, then only a Conv, then again a Conv plus pool; then a fully connected layer, then a fully connected layer and then finally the output layer.

Let us look at at least the first layer's numbers once more, and you can perhaps work out the remaining layers based on the principles that we talked about in the first lecture this week. So, 224x224x3, 11x11 was the size of the filter; and then you have the stride of 4. So, you would get your output to be, they also had a certain level of padding; so you would have 227x227 as the final input given to the network. So, to be (227-11)/4 +1 which gives you 55. That is what we talked about even earlier. Similarly, you can work out the parameters in each of these layers, by knowing by knowing the number of filters; and the kernel size that you see in each layer, which is shown on the figure.

(Refer Slide Time: 19:39)

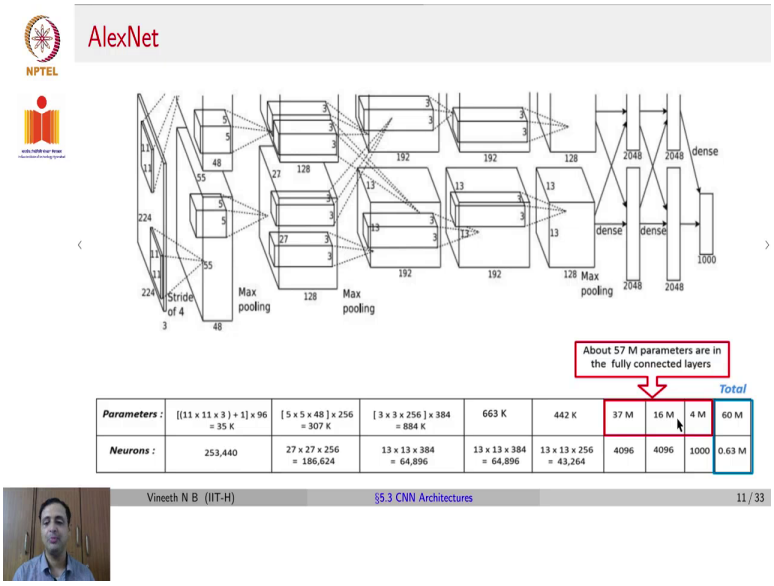


It had 8 layers in total, 5 convolutional layers and 3 fully connected layers here and here; the last 3 layers were fully connected. It was trained on the ImageNet dataset as we mentioned. The AlexNet designers also introduced a normalization layer called the response normalization layer. So, we so far saw a batch normalization layer, similarly the response normalization layer; normalized all the values in a particular location across the channels in a given layer.

For example, if you had a certain depth, you would then take a particular pixel location and go along the depth and normalize all the values along the depth at that particular special position. This was known as response normalization, and the authors argued that this would give an effect known as lateral inhibition. So, it would allow one of those to win over the other by getting a higher value in terms of the output after normalization and so on. In addition to this, this particular architecture also introduced the rectified linear unit or ReLU as an activation function.

That was one of the contributions that made this architecture a success. Of course, ReLU is now a default component or a default choice for an activation function, in many other CNNs. As you can see in the architecture Max-pooling follows the first, second and the fifth convolutional layer; the third and fourth did not have a pooling. And the ReLU non-linearity was applied to the output of every layer here in the entire architecture. Why was such a design chosen? At that time perhaps that was the best that was the architecture; that led to the best empirical performance.

(Refer Slide Time: 21:47)



Let us try to analyze the complexity of this architecture in terms of parameters and neurons. You see here the parameters in the first layer would be given by, there are 11x11x3 is the size of the filter. Plus 1 for a bias and 96 are the total number of filters chosen for the first layer; 48 which went to one processing unit and 48 which went to another processing unit. So, the designers of this architecture tried to take advantage of two processes that they had, in particular two GPUs that they had into which they simultaneously forked out these volumes into two different GPUs.

And then combine them in later layers, where you see these cross inputs as dotted lines, is when the output of one GPU was also fed to another GPU and so on and so forth. These cross links in the architecture should help you point out in which layers that was done. Coming back to the parameters. So, the total number of parameters in the first layer was 11x11x3 size of the filter plus 1 bias into 96, the total number of filters which roughly comes to about 35000. On the other hand, the numbers of neurons are given by the size of the image itself, which is 55x55x96. And that turns out to be something like 253440.

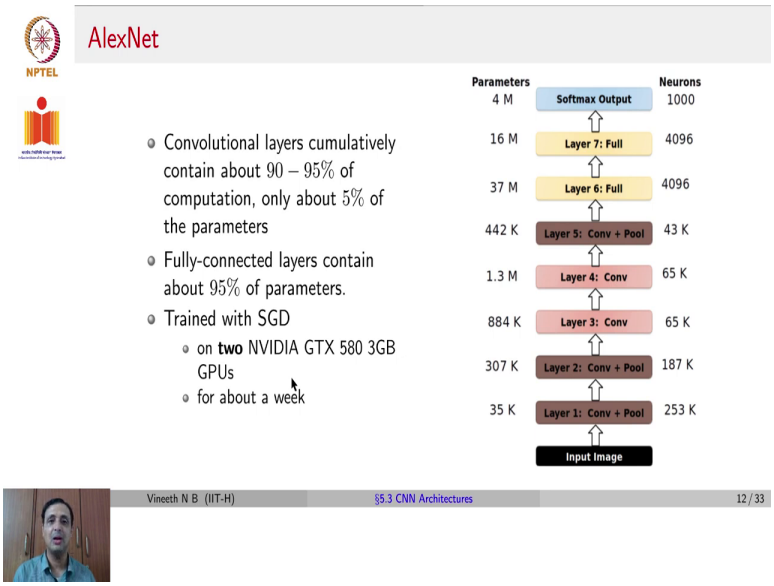
Similarly, you can compute the numbers for the other layers. The second layer the parameters turn out to be about 300k, and the neurons turn out to be about 186000 odd. Third layer, new parameters are 884k, number of neurons are 64000; so you can say see that the number of neurons is reducing, because the size is reducing after applying pooling. By a neuron we mean

each pixel in a feature map in the next layer; that is what constitutes a neuron here. And because we do pooling, the size of the image becomes from 224x224 to 55x55 to 27x27 to 13x13.

And because of the reducing size, the neurons reduce; but the number of parameters keep increasing because the number of filters taken are significant. Initially it was 96, the next layer had 128 + 128 you know filters or totally 256 filters, then 192 + 192 that is about 384 filters and so on and so forth. You can see that as you go deeper, the number of parameters keeps increasing, the number of neurons decreases until you go to the fully connected layers, where the number of parameters simply takes off and goes to the order of millions. So, about of the total 60 million parameters about 57 million parameters are located only in the fully connected.

And that should again go back and explain to you, how efficient the convolutional and pooling layers are in terms of number of parameters. So, and the number of neurons comes to totally about 0.63 million.

(Refer Slide Time: 25:17)



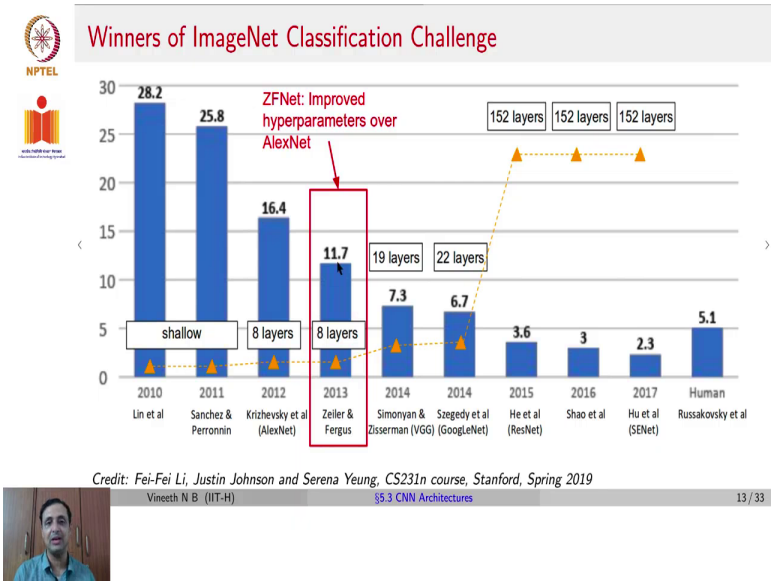
Another way of looking at it is just visualizing this from a different perspective. You can see here that go the parameters, go from 35k to millions in the last three layers; while the neurons start 253k and reduce towards the last layers. One interesting observation of this architecture is the convolutional layers may contain 5 percent of the parameters. Remember they contain only 3

million of the total 60 million parameters that AlexNet has. But, they account for 95 percent of the computation, why is that so?

Although they have only 5 percent of the parameters. Remember in convolution, you have to take the same weight and keep placing it on every location in the original image and get the output. Although the weights of few, the number of computations is high in a convolutional layer. So, 90 to 95 percent of computation sits in the convolutional layers; whereas 90 to 95 percent of the parameters lies in the fully connected layers.

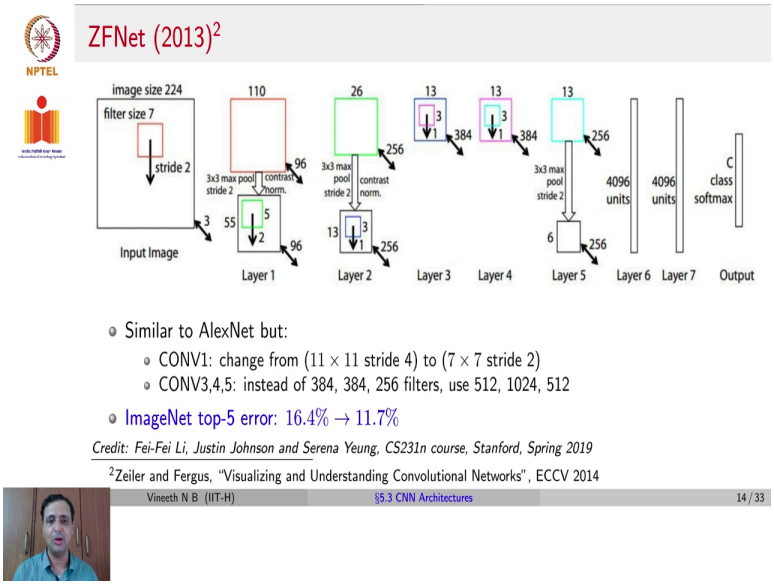
The AlexNet was trained with SGD, as I mentioned on two NVIDIA GTX 580 3GB GPUs and that is what allowed them to fork the feature maps into two different GPUs. And it took them about a week to train this architecture; today it can be done in much lesser time. Thanks to improvement of GPUs, but at that time it took them a week to train this architecture.

(Refer Slide Time: 26:55)



In the following year 2013, the winner of the ImageNet LSRBC was a network known as the ZFNet. It came from the names of Zeiler and Fergus, who came up with this design of CNN.

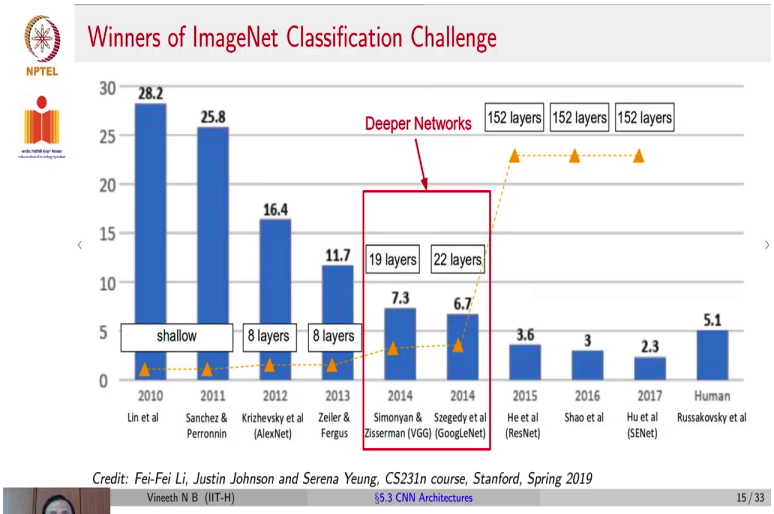
(Refer Slide Time: 27:15)



The main contribution of the ZFNet in 2013 over AlexNet was the architecture was fairly the same. The overall Conv Pool, Conv Pool, Conv, Conv, Conv Pool fully connected, fully connected. But, there were a few changes in hyper parameters, what were the changes in hyper parameters? In Conv1, the filter size was changed from 11x11 stride 4 to 7x7 stride 2. In Conv 3, 4 and 5 the number of filters was increased from 384, 384 to 256 to 512, 1024 and 512. And this careful choosing of hyper parameters led to a significant decrease in the top-5 errors from 16.4 percent to 11.7 percent.

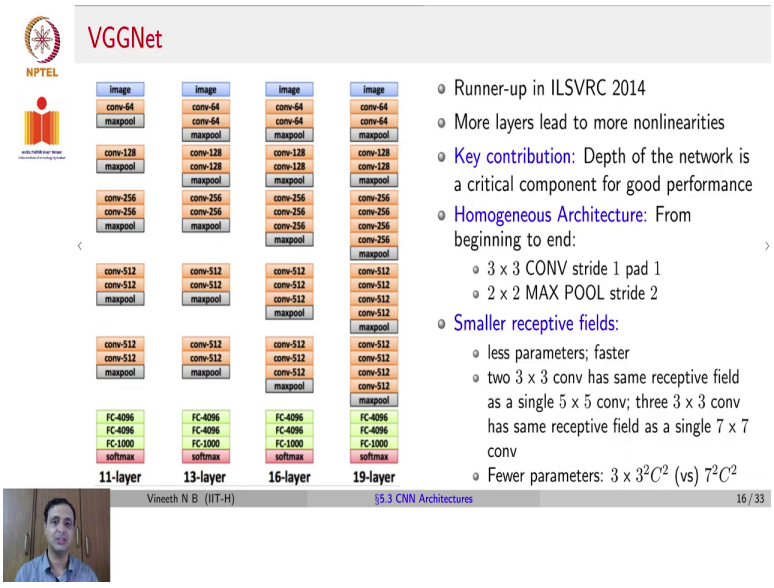
One term I should clarify at this stage is the notion of hyper parameters, why do we call these values 11x11, 7x7, 384 as hyper parameters? Because the parameters that we learn are the weights of the neural network; all the other values are user defined, they are not learnt. So, we call them hyper parameters, the parameters are the weights which is what we actually learn; but, this was the main contribution of ZFNet in 2013.

(Refer Slide Time: 28:39)



Moving on to 2014, one of the major contributions 2014 was a new architecture known as the VGGNet.

(Refer Slide Time: 28:50)



The VGGNet, stands for an (arcade) architecture, developed by the VGG group at stanf… at Oxford, which is also stands for visual geometry group. And they came up with an architecture with a certain philosophy in their design, they came up with multiple depths in their architecture. They argued that while AlexNet took LeNet from a certain depth to the next level, and the ZFNet

maintained the same depth as AlexNet. They argued that by making a CNN deeper, you could solve problems better, and you could get a lower error rate on the ImageNet classification challenge.

So, the way they went about doing it is to take multiple architectures of different depths. So, you can see here that the input is on top, the output is at the bottom in all of these architectures. You can see Conv layers, Max-pool layers, Conv layers, Max-pool layers so on and so forth and these 64, 128 denote the sizes that you have in these architectures. The interesting design philosophy that they had, so this architecture won the runner-up in the ImageNet challenge in 2014. And their philosophy was that by increasing the depth, you can model more non-linearities in your function; and the key contribution was that depth is a critical component.

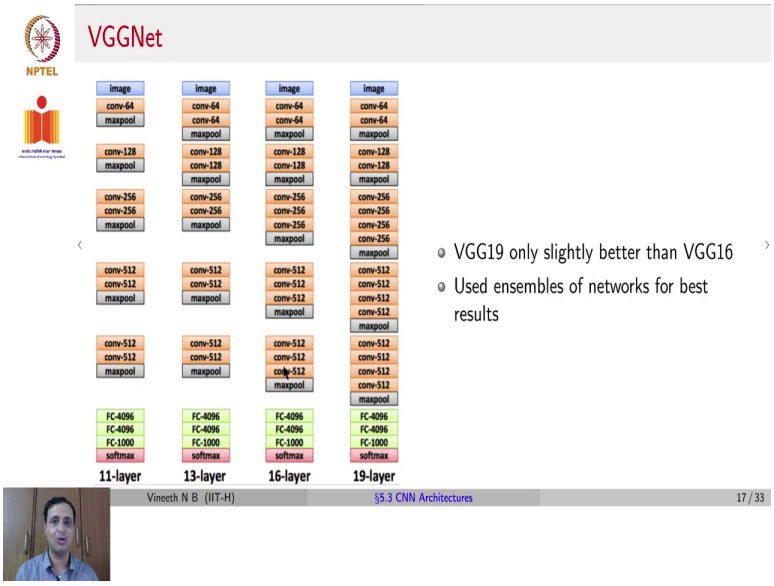
They maintained a homogeneous architecture from beginning to end, what does that mean? They used only a 3x3 convolution across all of their layers. We saw with AlexNet that it had 11x11 filter the first layer and ZFNet changed that to 7x7 filter in the next year. But, VGG argued that they would just maintain instead of struggling to find the filter size, while designing the architecture. They decided to fix it at 3x3, how does that help will come to that later moment. And they had a 2x2 Max-pool across all of their pooling layers in the network.

Their main idea was that a smaller receptive field means lesser parameters, which means if you have a 3x3 filter; there are lesser parameters to learn. More importantly, two 3x3 convolutions done in sequence have the same effect as one 5x5 convolution. When you do one 3x3 convolution, you get an output; if you do a 3x3 convolution in the output of that first one. You would have the effect of the neighborhood of 5x5 pixels in the original image having an effect on the latter later output. Rather, two 3x3 convolutions have the same receptive field as a single 5x5 . Similarly, three 3x3 convolutions have the same receptive field as a 7x7 convolution and so on and so forth.

Rather, instead of trying to keep filter sizes like 11x11 or 7x7; they argued that they could always keep the filter size as 3x3. Importantly, as we said it would have fewer parameters, if you took a single 7x7 convolution; you would have 7*7 C*C square parameters, 49 C*C. Whereas, if you had three 3x3 convolutions arranged in sequence; you would have 3*3*C*C parameters, which

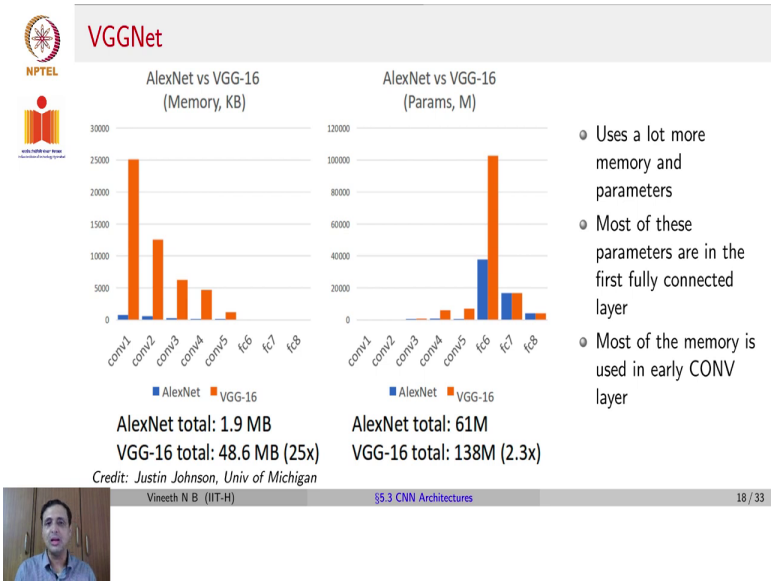
would be 27 versus 49. So, this would have a fewer parameters than doing a single 7x7 convolution.

(Refer Slide Time: 32:44)



They found in this architecture that VGG19 was only marginally better than VGG16. So, until VGG16 the performance kept improving with depth, but it seemed to saturate after a certain depth. And this issue was addressed later as we will see in the next few slides. And they also tried using ensembles of different networks to get improved performance on the same task.

(Refer Slide Time: 33:13)



In terms of parameters, VGG however had a significant memory overhead and a parameter overhead. So, you can see here a comparison of AlexNet versus VGG16 in terms of memory. Memory would simply be the number of neurons across all of your layers in the VGG versus number of neurons across all of the layers in your AlexNet. Remember again when we say number of neurons, we mean the size of the feature maps; because every pixel in every feature map is a neuron for us in a CNN.

So, you can work this out carefully and find that AlexNet has a total of 1.9 MB versus VGG16, which has a 48.6 MB memory in terms of parameters. The first few convolution layers are reasonably small for AlexNet and VGG in the fully connected layers. VGG has a significant increase in parameters and while AlexNet had a total of about 60 to 61 million parameters; VGG16 ended up having about 138 million parameters. So, while VGG works well in many settings, that is a limitation of VGG, because of its increased parameter count and memory.

While, each choice of 3x3 by itself reduces the parameters; the parameters added up for them because of later layers, the depth, the fully connected layers so on and so forth. It uses a lot of lot more memory and parameters; and as we saw most of these parameters are in the first fully connected layer. Most of the memory as we saw earlier is in the early convolution layer because that is when you need a maximum number of neurons to contain all of those feature maps.