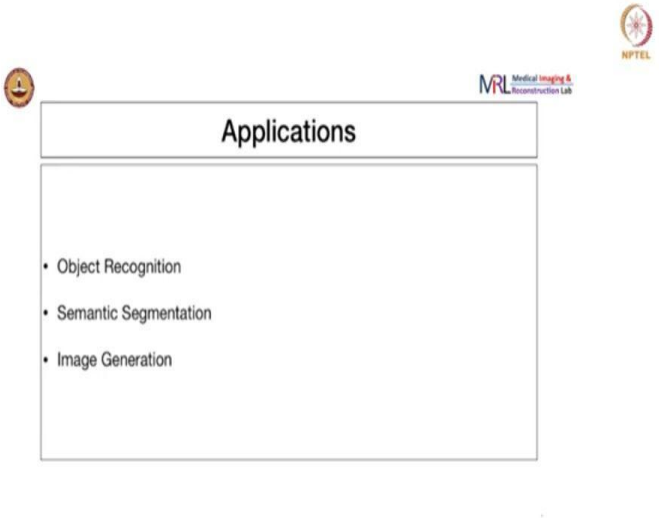
Medical Image Analysis
Professor. Ganapathy Krishnamurthy
Department of Engineering Design
Indian Institute of Technology, Madras
Lecture 45

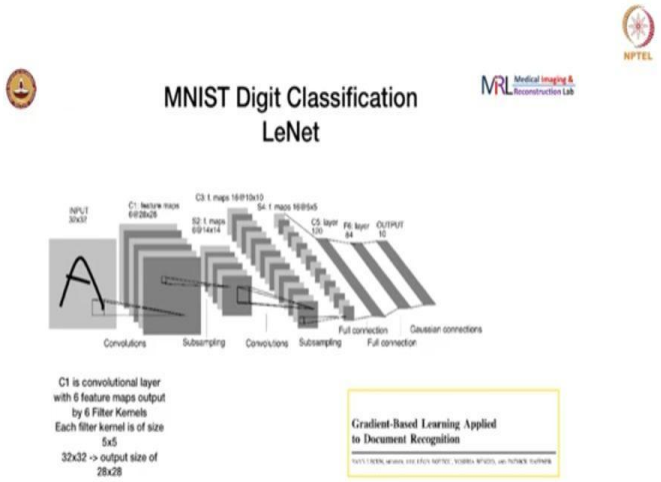
Applications of CNNs

(Refer Slide Time: 00:16)



Hello and welcome back. So, in this video we will look at some of the more commonly used CNN architectures specifically for object recognition. There are of course other applications like semantic segmentation and as well as image generation. We will look at these applications in a different video this week or maybe even the next.

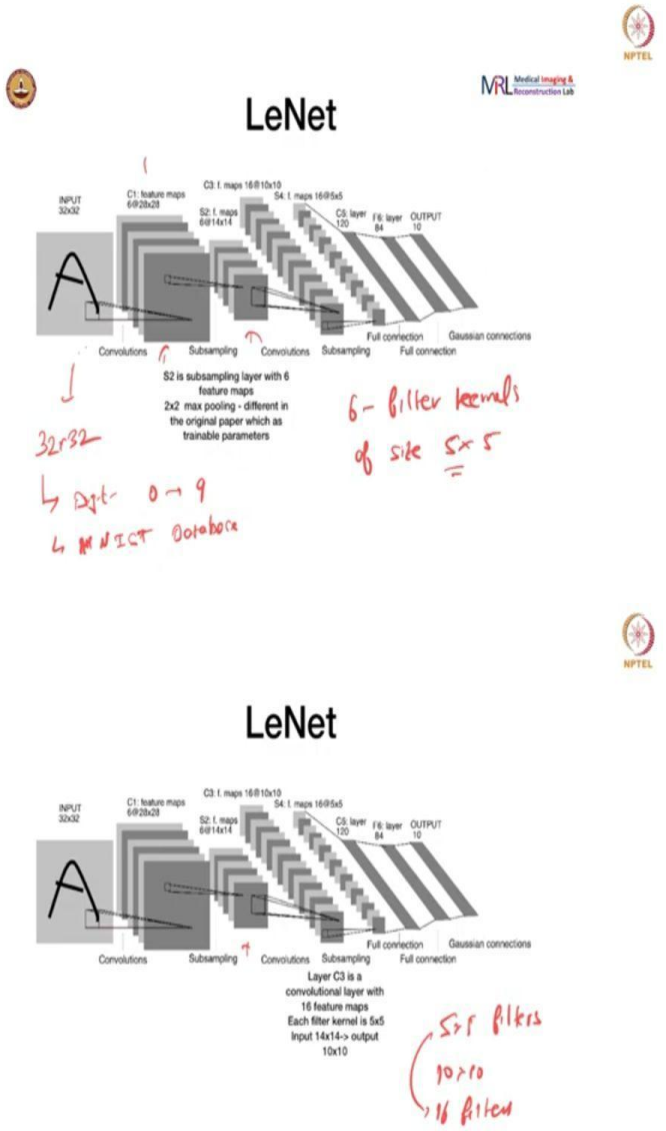
(Refer Slide Time: 00:43)



So, so now the earliest and of course a well studied, and often studied, often give us an example of CNN is the paper by Yann Lecun. This was 1998 I think, it is referred to on the right, so please look it up. So in this paper, the author describes a digit classification network. So, basically a database of handwritten digits, which has been scanned and digitized into images is given as input to this neural network. And the output is basically the labeling for digits 0 to 9. So, it is a classifier, it is used for recognizing images of digits, handwritten digits, which have been scanned in.

So, it takes us input a picture of size 32×32; and the output is a vector of size 10. So, because it is a 10 way classification, digit 0 to 9; so vector size 10 is the output. It is it is actually one of the earliest and more well reported and more successful neural networks; so we will start off with that briefly.

(Refer Slide Time: 01:55)



So, the input to this network is an image 32×32 image; or this is supposed to be of a digit, but an illustration it is usually an alphabet. This image is always like that; images taken from the paper. So, it is 32×32 image of a digit, digit 0 to 9. There is actually this database which contains this digit is known as the MNIST database; you can Google it or search for it online and download those, all the 1000's of datasets, 1000's of images of digits, handwritten digits that you can use for training. So, if you have never done some programming before for deep neural networks, you know this is a good place to start; so MNIST digital classification.

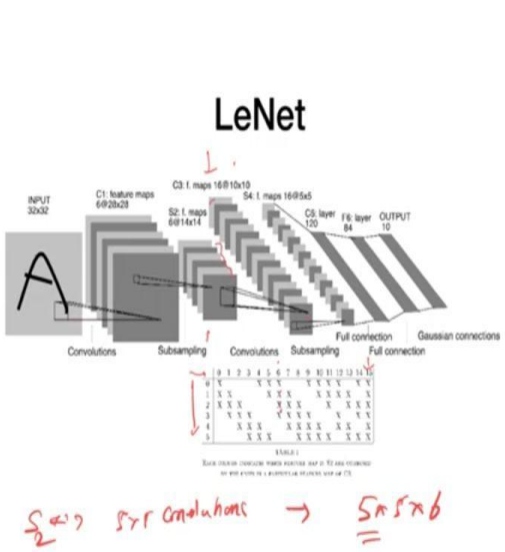
So, the input is a 32×32 image. And as we look at the first layer, so this is the input layer for this input image; that is a, it is a feature, you can think of the input itself as a feature map of size 32×32. The first layer has 6 feature maps, so it is a convolutional layer; the first layer is a convolutional layer. It has a 6 feature maps, each of size 28×28. So, how do we get to this 6 feature maps? Because you have 6 filter kernels of size 5×5. So, if you and there is no zero padding; so 32×32 convolved with 5 filters will give image outputs of size 28×28.

So, you define 6 filters, you will get 6 feature maps. So, then you have this sub-sampling layer; sub-sampling is done with a 2×2. 2×2 max pooling. So, we will just assume that it is slightly different in the paper I think; but anyway, for our purposes of understanding this. So, we have a 2×2 max pooling layer here; the second layer is a max there is two, it is a max pooling layer. So, you have a 28×28 input. So, if you do 2×2 max pooling, so you use a factor of 2 on either dimension; you have 14×14 as output right there, it is your output layer.

So, you start off with the input, do a convolutional layer with 5×5 filter size; you get 6 filters. So, your output is 28×28, you get 6 of these feature maps. And for each on each one of them, you do max pooling; and that will reduce its size to 14×14. And once again, on top of these 14×14, you do convolutions, once again with 5×5 filters filter kernels. And this time your output is of size 10×10.

That is again, again without zero padding, your output is size is 14-5+1; that gives you 10×10. But, you have 16 filters, 16 filters of size 5×5; so you will get this output, which is again a 16 feature maps as output. So, so the layer C3 is basically a feature map; it is it is basically a convolutional layer with 16 feature maps. And each kernel filter is 5×5, the input is 14×14, and output is 10.

(Refer Slide Time: 05:36)



So, but the, the way he did that, so there are 16 outputs; you can see look at the 16 outputs, and he has 6 inputs. So, the each column indicates which feature map in S2 are combined. So basically, he did the way we had talked about convolutions when we did the, the conventional convolutions is that when you say for instance for S2. In S2, when you say I am doing in this layer, you are going to do on S2, you are going to apply 5×5 convolutions.

So, which means that your filter size is actually 5×5×6; so, you are going to do the mixing or nonlinear combination across the channels as well.

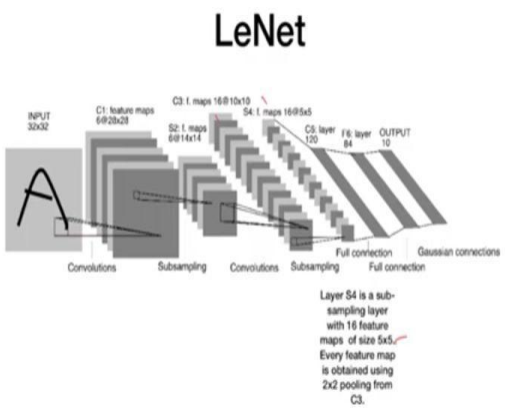
So, then your filter kernel size that actually 5×5×6; that is what that is it that will be the usual interpretation. However, in this paper, what is what the, what it describes is that for every. So, if you look at every feature map here, if you label them from 0 to 15; each of the feature maps only takes into, takes the combinations of specific sets of feature maps in this layer. So, for instance, if you look at layer 6, the output each of the output units in 6 takes inputs only from feature maps, 0, 1, 2 and 3. So, the columns wherever it is X mark, those are the marks gives you which feature maps were used to produce the output.

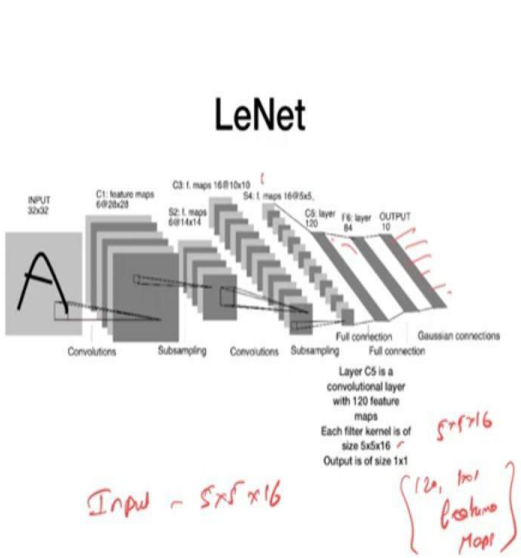
So, instead of your filter kernel extending across all of the feature maps, you only extend across specific feature maps; and that is how we got this, you got this you get this output of 16 feature maps; each of size 10×10. But, each of these feature maps takes only inputs from a specific subset of feature map. So, except for 15, 15 seems to; feature map 15 will, it takes in input from

all of the feature maps in layer in S2. So, this is a is one of the different unique things in this paper. Of course, you can implement networks like that now too and there are some papers which talk about that, where they do group based convolutions.

So, this is this is one of the novelties in this paper, and how we combine the feature maps with not including all of them to produce a convolutional output. So, once again, this is for the layer C3 is a convolutional layer, or C3 is the output of a given S2 as input; so, that is how we look at it.

(Refer Slide Time: 08:26)

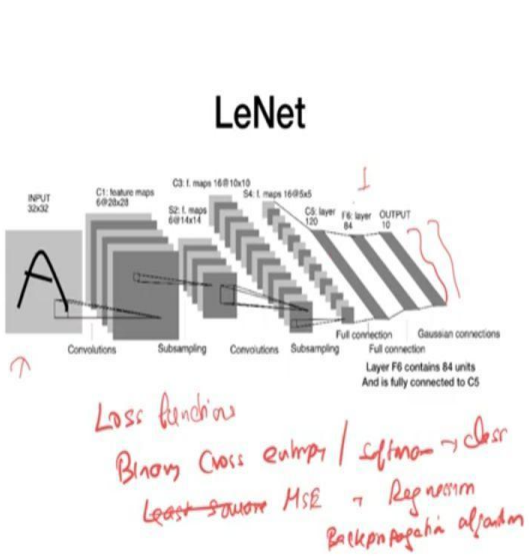




Similarly, if you look at layer S4, it goes from C3 to S4. Again, it is a sub-sampling layer with 16 feature maps of size 5×5; because 10×10, if you do a 2×2 sampling, 2×2 max pooling, you get 5×5; that is least layer. That is S4 is the sub-sample layer from C3; now, layer C5. Here this is a once again is the convolutional layer with 120 feature maps. So, you can think of each filter kernel of size 5×5×16. So, your input feature map in S4 is of size 5×5; so the input is 5×5. Of course, if you take into account the channels, it is 5×5×16.

So, now we have the filter kernel; the filter kernel can also be of size 5×5×16. And if you define 120 of them, you get 120 , 1×1 feature maps as output; but that is one way of doing it, and from there on, it is the fully connected layers to 10 outputs. Here, you can do softmax over 10; even though the paper, it is a different process that is described; but we can do softmax when you code, you can code it a softmax over 10. And of course these from 120 to 84, it is a fully connected layer, and from 84 to 10 outputs is also a fully connected layer. That is what we see here.

(Refer Slide Time: 09:57)



So, layer F6 and the 84 units fully connected to C5; and if you go to the output layer again from 84, you get softmax 10 outputs. So, this is a very nice network to studies because you can actually understand each one of these, what is what is happening on each one of them one of these layers.

And you can look at by hand go to the calculation, see, what kind of sizes are there in terms of the output? What the size of the input when you say, max pooling different type; try out different types of max pooling, different feature kernel size etcetera. And of course, also use this illustrate one more point that in the at the finally, when you when you go towards the output side of the neural network, typically make them into fully connected layers.

So, even in some of the more recent networks you would have to for classification that is you still have to convert these few last few layers into fully connected layers. And that is typically where a lot of the parameters come into play because they are they are fully connected. And also another thing is that you can always output, whatever that is that you want to estimate. So, in this case, you are outputting 10 class probabilities. So, again, you can always interpret these outputs as class probabilities; you can also do regression. So, again depending on your task, this output can be of course properly designed.

Now, one of the things that I have not gone into great detail in all of this, I think it is a good time to bring this up is; we have not looked at particular loss functions. But, I had mentioned in an earlier lecture, you typically use a binary cross entropy or softmax for classification. And you use least squares, the squares or I would say mean square error; I would say the least squares, say mean square for a regression.

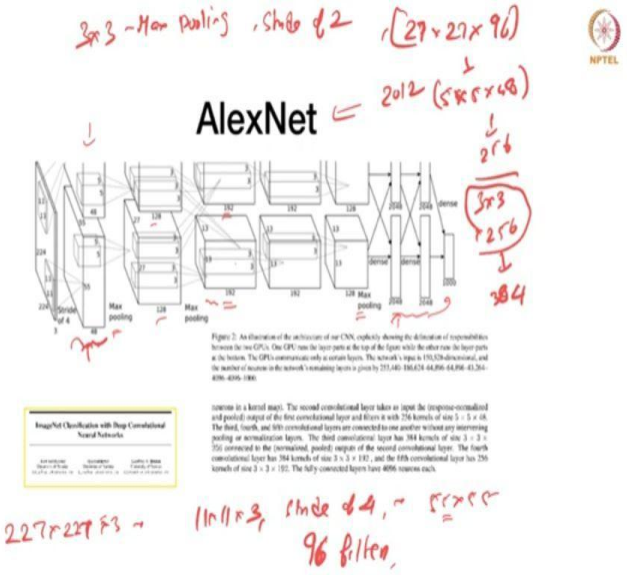
Also, I have not talked about regularization, wherein you can have regularization over the weights L1 and L2, regularization over the weights. But, that is not the topic that I delve into. But, this is a typical loss function, the weights of the network that is the weights of the network, nothing but the filters.

You estimate the filters using a backpropagation of course, you learn the weights with gradient descent about the weight updates. In order to calculate the weight updates, you would use the back propagation algorithm. The algorithm of course remains the same; details might be slightly different, but it remains the same for feed forward and also for convolutional neural networks. So, what I have described on all of these will whatever I want to describe just the forward pass. You do not want to talk about the actual, the backdrop algorithm what happens in how will you train.

So the, the idea in all of these networks is that right now, I only describe the forward pass; you have an image that comes in, goes to all these layers, and then it produces 10 outputs, which you interpret as probabilities. And of course, you have the appropriate loss function, in this case softmax, calculate the error, and then you back propagate the error through all the weights in the network; or that is a filters.

And then once again, you do it for multiple batches. Of course, the weight, which is all based on the gradient descent algorithm only; there are variations of recent also available nowadays. And they are all they come in, packaged in a lot of these commonly used platforms like Pytorch and the Tensorflow.

(Refer Slide Time: 13:39)



So, the first network we will look at is the AlexNet. This is from 2012 paper, the submission to the ImageNet challenge which it won; it was a winning entry and as of CNN, using a CNN, and this is the architecture there. So, it it it was ahead of the nearest competitors by quite a margin and sparked a huge interest in the computer vision, and medical image analysis community.

So, we look at the architecture here. The network was trained across two GPUs; that is what is shown in this image in this architecture map. In some layers, the computations reside within the same GPU, that being said, the for the filtering for applying the convolutions you will use the feature maps. The input feature maps are those that are more present only in that GPU; so there

is no transfer of data there. But in one layer, which is shown by these dotted lines here, you would use a feature maps in both the GPUs.

So, we will not go into much detail, but you can do it, and be sure that it is not too hard to understand. But, just to illustrate the first couple of layers, we will walk through it. So, the input to this network is a 227×227×3 image; so three channels RGB image. So, the first convolution layer uses 11×11 filters. Of course, technically, it is 11×11×3 filters with a stride of 4. So, that will give and that will give rise to feature maps with size 55×55, which you can verify.

And once if this of course we define 96 such filters, so which leads to 48 filters in one GPU and 48 filters in the other; that is the output of the first convolutional layer. Or, the second convolutional layer takes as input the, though the max pooled version of this. So, which means that you have to max pool 3, the max pooling done is a 3×3 max pooling with a stride of 2. So, the 55×55 feature maps with a video 3×3 of a max pooling with the stride of 2, you will get 27× 27.

And then followed by, so now we have 27×27×96 feature maps; of course, 48 in each GPU, on which you will apply 5×5 across 48 filled convolutional filter kernel. So, you will define 256 of them. And that will give rise to, if you do if you do this with appropriate padding, you will get 27 across 27 feature maps once again.

But, 256 of them because that is that is the number of feature; that is number of filter kernels you have defined. So, you have from going from here. So from here, we have 96 feature maps in this layer, 48 in each GPU, each of size 55×55; and you do max pooling followed by convolution. So, that gives the output is 256 feature maps, 128 in each; each of size 27×27.

And once again, here you can do another max pooling, followed by padded convolution. So, if you do max pooling, followed by convolutions, in this case, 3×3 convolutions. So, if you look at this particular layer, so you have your input is 27×27×256. So, your filter 3×3 filters are 3×3 × 256, and you are defining a 384 of them.

So, that is what gives rise to 13×13, 384 output; 13×13 feature maps, 192 in one GPU, 192 in the other. So here, because this layer, you are you are getting inputs from every neuron in these

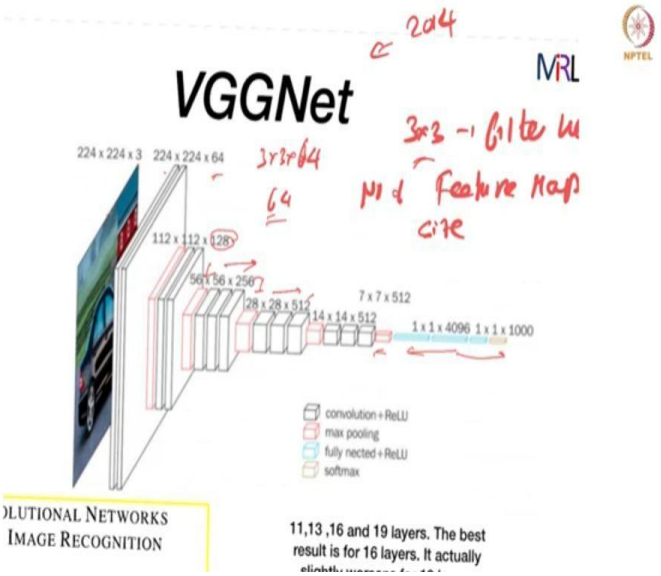
feature maps is getting input from all the feature maps in the previous layer, even if they are in two different GPUs.

So, these are the three layers that I would like to go, that I wanted to present; you can meanwhile, at your leisure, figure out what the conditions are processes as you go towards the end of the output side of the network. So, at some point you, you will have dense connections going from these from the after the max pooling, the last convolution layer after the max pooling, you will do dense connections to get 4096. Once again dense connections 4096 followed by thousand way softmax classification. So, this I just got walked through to make you understand the sizes, how you calculate the output sizes, so that it is consistent when you make your own networks.

Again, it is not too difficult; you can do it by yourself; watch the video again and do it for yourself. The idea is to, so whenever there is max pooling, do the max pooling; and so read the text because the text very clearly says after max pooling, you do the convolutions. So, from the first convolution layer, do max pooling, and then a convolution.

And the second max pooling layer, once again second convolution layer, before you do the convolution to the max pooling and so on and so forth. And finally, you will end up with dense connections and 1000 way output. I just had excellent results in the ImageNet challenge. I will not show exactly the classification error etcetera; but it was very low. So, it was I think, 15% or something that was the error; so, will look at the other architectures in the next few slides.

(Refer Slide Time: 20:00)



So, we will look at the VGGNet architecture; this is a 2014 entry to that ImageNet challenge 2014 paper. Here they are looking at using small filter guns; so they used in throughout all the layer 3×3 filter kernels. So, if you remember from the previous slides in the AlexNet used 11× 11, 5×5 and 3×3.

Here, they show that if you do a sequence of 3×3 convolution in succession, you get the same receptive field ; maybe a 7×7 filter kernel, but with the lesser number of parameters. So, that with that in mind, so they have defined; throughout the network they use only 3×3 filter kernels.

The two schemes two things to keep in mind. One is that as the as we go deeper into the network as the number, as the number of layers increase, the number of feature maps increases. The number of feature maps increases per layer, and the size of each feature map reduces. So, let us walk through this architecture and just to confirm, that is the case.

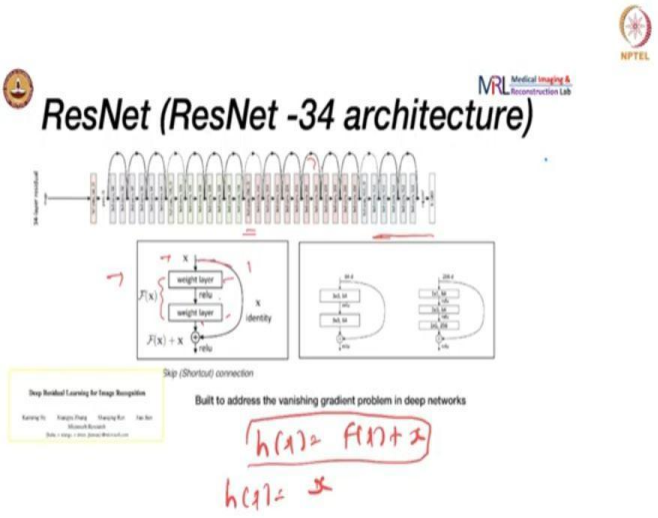
So for instance, if you look at this, the input is a 224×224×3; and then you have two, I think a sequence of two layers with a 3×3 convolution, 64 channels. In a sense that, from the from here, the first convolution layer defines 3×3×3 kernels 64 of them. The second convolutional layer also would have 64 filter maps, but of size 3×3×64. Of course, padded convolutions are used or same convolutions as they called to keep the size fixed.

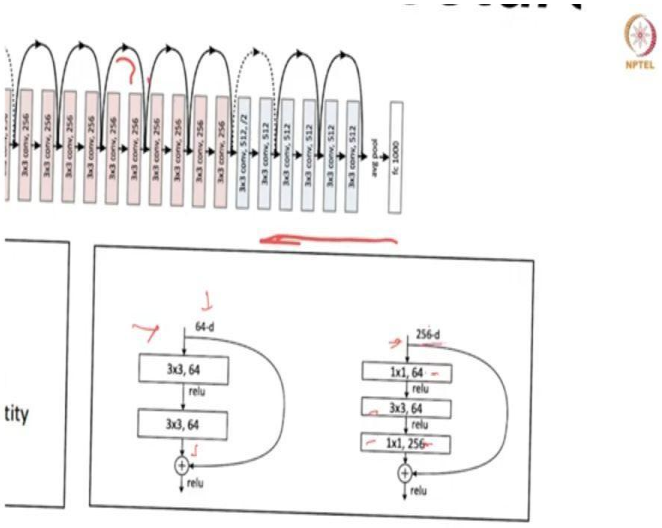
Now, following this there is the down sampling is achieved using 2×2 max pooling with a stride of 2. So, following max pooling once again, you do 3×3 convolutions across the feature maps again 64 of them. You want to get one then because you define 128 filter columns, so that the output is 128 feature maps. Similarly, there is down sampling followed by sequence of convolutions with the same convolutions maintaining the size, down sampling sequence of convolutions. And the last number tells you the 512, tells you the number of filter kernels defined in those layers.

So, the last convolution layer output is 7×7×512, which you which is kind of stretched out to fully connect to a 4096 fully connected layer, which is then fully connected to a thousand way softmax. So, this actually achieved a excellent; and this network architecture is often used as a as a feature extractor. So, you would train, you take a pre-trained version of this; so this, this has been trained for ImageNet, which cell used to extract features for other tasks. So, because of the filter kernels etcetera, it was, it is considered a good feature extractor in general.

There are variations of this with multi different types of layers; and the best results was obtained for a 16 layer VGGNet.

(Refer Slide Time: 23:30)





So, in this we will look at the ResNet architecture of the residual net. So, this was this architecture was done to prevent the vanishing gradient problem when the networks become very deep. So, it was observed that empirically many groups have observed that as they increase the number of layers in the network; so for instance, in the case of CNN, so you have keep adding more and more layers for a particular task, let us say classification. After a certain point the the the results seem to degrade. So, ideally the expectation was that if you have deeper networks, you will have better results; but that did not seem to happen.

And the explanation is that when you have deeper networks and you do the back propagation algorithm, which does the weight updates; then as the network goes deeper than the gradients that are used to make these weight updates grow vanishingly small. So, this in order to help this to solve this problem, this residual connection is proposed; so here is what how it works.

And just look at this in two different ways. So, one is that the output, we have write it now in this fashion, and then explain what this is. So, if you have this intermediate layer convolution layer in the middle of a network, the x is the incoming bunch of feature maps; and you have two convolutional layers with a non-linearity in between.

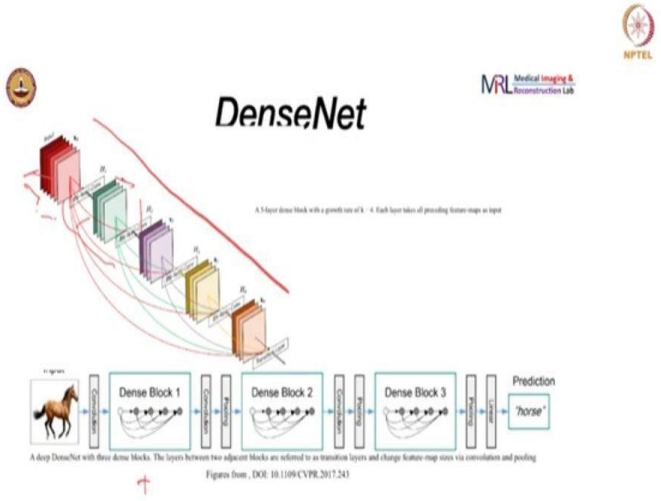
So, what you do is you add or you add the input to the earlier layers, two layers before to the output of this layer, this second big layer; so and then so the entire output is this. And the worst case scenario, you get h(x) = x; because even it is easy for the network to learn zero. So, basically the identity transform. So, f(x) is nothing it does nothing, other than this this entire two

networks put two layers put together. If you call that as as a layer or a module, and all it is just doing is transmuting the input to the output; that is that is very very easily done and can be very easily learned.

So, the other way of looking at it is that the, the vanishing gradient problem happens; because when the network is very deep and your gradients actually flow from the output towards input layer. So, as a and this this process involves a lot of matrix multiplication. So, at some point, the gradients can become very small very small. But, if you have the skip connection, then this does not involve any matrix multiplication rather than identity; then you have a better flow of gradients, you can get better weight updates; so, that is the idea behind it.

And how do we accomplish this residual connection? There are two ways of doing this. One way is to directly add, like shown here. Here we have to make sure that the input dimension the number of channels, and the output from the last second layer; a number of channels that is matched, so that you can add channels to channel. If there is a mismatch, one way of doing it is you if there is a possibility of mismatch, one is that you; when you you reduce you down sample or project it using 1×1 convolution. Let us say in this case 256 becomes 64 channels; and of course you do the usual 3×3 convolution; and then you project it up to 256 channels where then you can add. So, these are two techniques typically used for.

(Refer Slide Time: 27:02)



So, we look at the DenseNet architecture, which is basically the ResNet kind of architecture taken to an extreme. Here, the idea is that each layer takes all preceding feature maps; the layer sends all the feature maps from the preceding layers as input. So, but then in order to prevent the so called feature map explosion, because you can not keep appending; here they just append. You, you you make sure that; the output number of output feature maps from every layer is limited. They call as the growth rate, and it is usually set between 4 to 8; so 4 is the one that we will typically use.

And the the the network itself is made up of the so called dense blocks. So, each dense block as you see in this picture, is made up of a bunch of convolution layers. So if you look at this, this is one dense block, so there is an input. And then you have, you will do this so called batch now on. I will talk about batch norm in later later video, which is just, it is just one module that you can drop in into any neural network; batch norm normalization actually, you can think of it that way, followed by in the sense value and convolutions. So, these three sequence of operations and this output is fixed.

So, then you will get only four feature maps as output from this convolutional layer. But, you see that in the next for the next convolution layer, this is actually appended. So, you will append for the next convolutional layer as input, you will append both the input here and these outputs. So, you keep doing that for all the layers; you see that these layers are used as input for all the subsequent layers. So, a sequence of such 4 or 5 convolutions will be one dense block. So, after that, the output of the dense block is taken and then you do another convolution pooling and resizing etcetera.

So, you can have a bunch of dense blocks. So, these together they would can translate into maybe hundreds of layers, convolution layers. And this again, had very good results on the image recognition. And it is been adopted, we used in a lot of other architectures as well, especially in segmentation, which we will talk about in a later video. So, I am I am going to stop here with an explanation of architectures; there are quite a few that have come after this also. But, once again at this point, once you know how to read through these, I urge you to read some of these papers; then you can easily understand what the Uber is lying upon.

So, typical architecture consists of these days, most of these architectures will have a residual connection. There is the batch norm layer, which I said this is we will talk we will cover later in

a later video, where we will talk about how what are the different operations, other operations other than max pooling and convolutions; in a for training a neural network, where we will talk about that next week in the next week's videos. When we when before we come to semantic segmentation and even generational models.

So, the common modules are basically batched on a non-linearity and convolution, that typically in the process of medical image analysis; these are the three commonly used computations. And then it is also common to bunch to a bunch convolutions together; so 2 or 3 convolutional layers put together. You also have same convolutions, so that the size of the feature map does not change. 1×1 convolutions to a project down, project up the number of channels that you want.

So with this, we will end this week's set of lectures. We will talk about some of the other computations and involved in CNNs in later weeks, next week; and we will talk about semantic segmentation as well. Thank you.