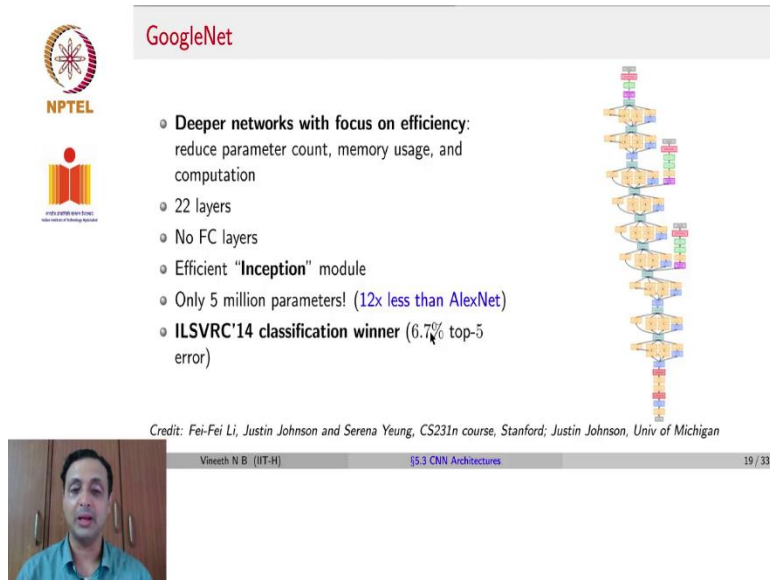


Deep Learning for Computer Vision
Professor. Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture No. 35

Evolution of CNN Architectures for Image Classification - Part 02

(Refer Slide Time: 0:16)



GoogleNet

- Deeper networks with focus on efficiency: reduce parameter count, memory usage, and computation
- 22 layers
- No FC layers
- Efficient "Inception" module
- Only 5 million parameters! (12x less than AlexNet)
- ILSVRC'14 classification winner (6.7% top-5 error)

Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford; Justin Johnson, Univ of Michigan

Vineeth N B (IT-H) 35.3 CNN Architectures 19 / 33

Along with VGG in 2014 was also another network called the GoogleNet. GoogleNet came from Google focused on deeper networks. But, with an objective, for efficiency to reduce the parameter count, to reduce memory usage, and to reduce computation. So, how did they go about achieving all of this? They had up to 22 layers in the network; remember VGG went up to 19, so the GoogleNet went to 22 layers. So, the networks were getting deeper each year, they did not have any FC layers.

How did that help? Recall that FC layers had the maximum number of parameters for both AlexNet and VGGNet. So, by getting rid of FC layers, they reduced the number of parameters. And what else did they introduce? They introduced a module known as the inception module; which will describe in detail over the next few slides. Please do not worry if you are not able to see this figure carefully; will describe it in more detail over the next few slides. It had a total of 5 million parameters, which was 12x less than AlexNet and significantly lesser than VGG.

And it was the winner in the 2014 ImageNet challenge ahead of VGG. VGG was a run-up runner-up this year that year and GoogleNet won the challenge with a 6.7 percent top-5 error. So, as you can see every year as the networks got deeper, the error started dropping significantly.

(Refer Slide Time: 2:04)

GoogleNet

- NPTEL
- IIT Madras

- **Inception module:** Local unit with parallel branches
- Local structure repeated many times throughout the network

The diagram illustrates the Inception module's internal structure. It starts with a 'Previous Layer' that feeds into four parallel paths:

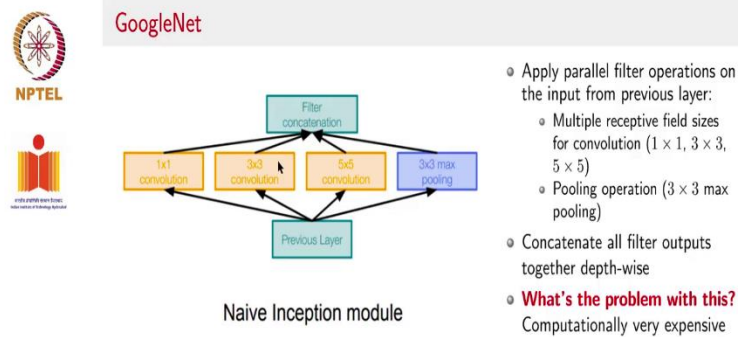
- A path with a single '1x1 convolution' layer.
- A path with a '1x1 convolution' layer followed by a '3x3 convolution' layer.
- A path with a '1x1 convolution' layer followed by a '5x5 convolution' layer.
- A path with a '3x3 max pooling' layer followed by a '1x1 convolution' layer.

 All four paths converge into a 'Filter concatenation' layer. To the right, a vertical stack of these modules is shown, with a red box highlighting one of them. At the bottom left, a video inset shows Justin Johnson, with the text 'Justin Johnson, Univ of Michigan' and 'Vineeth N B (IIT-H)' below it. The slide footer includes '§5.3 CNN Architectures' and '20 / 33'.

Let us now talk about the inception module, which was a key innovation in their approach. The inception module looks somewhat like this, which was one of those units that keep repeating in their architecture. So, the output of the previous layer was branched out into one layer which did a 1x1 convolution. In parallel another couple of layers, which did 1x1, 3x3, another couple of layers which did 1x1 and 5x5. And another one we did Max-pooling and 1x1 convolution.

So, this was a local unit with parallel branches, and this local structure was repeated many times in the neural network as you can see. This inception module kept was tagged on top of each other to complete the architecture of the GoogleNet.

(Refer Slide Time: 3:03)



Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019



Vineeth N B (IIT-H)

§5.3 CNN Architectures

21 / 33

So, the Naive inception module was the final inception module that was in the architecture; but let us develop this slowly. So, the Naïve inception module takes a previous layer, instead of deciding a particular feature size, filter size, or VGG which always relies only on 3×3 . What the GoogleNet designers decided to do was they decided to do multiple such filter sizes in parallel, and concatenate all of them. So, you have a 1×1 , a 3×3 , a 5×5 , even a 3×3 Max-pool and they concatenated all of those. So, the depth that you get in this layer after the concatenation would be the concatenation of all of those depths; that you get as the output of each of these convolutions or Max-pooling.

So, they had multiple receptive field sizes and they had a pooling operation, which they concatenated depth-wise. But, there is a problem here, do you see the problem? The problem is this is computationally very expensive. Remember we said for AlexNet that most of its computations are in its convolutional layer; while the parameters may not be high. Remember you have to take the filter and convolve on every part of the image, and that can be pretty expensive. And now when this inception module does this across 1×1 convolution, 3×3 convolutions, and 5×5 convolutions simultaneously. And then concatenates things become even more computationally expensive. So, what do we do?

(Refer Slide Time: 4:50)

GoogleNet

Solution: Use 1×1 "Bottleneck" layers to reduce channel dimension before expensive conv layers

1x1 CONV with 32 filters
(each filter has size $1 \times 1 \times 64$, and performs a 64-dimensional dot product)

- Preserves spatial dimensions, reduces depth!
- Projects depth to lower dimension (combination of feature maps)

Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

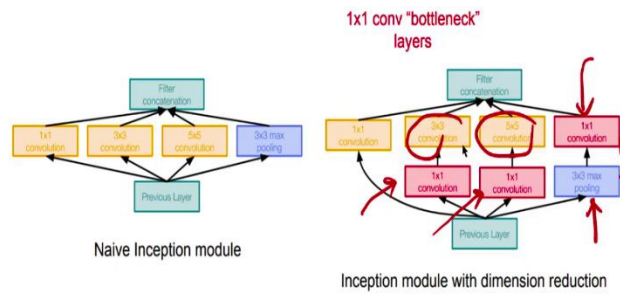
Vineeth N B. (IIT-H) 5.3 CNN Architectures 22 / 33

They introduced something called 1×1 Bottleneck layer to reduce the channel dimension before expensive convolution layers. Now, let us look at what this means, so if you have a $56 \times 56 \times 64$ volume, you can do a 1×1 convolution across the depth. So, that filter would be $1 \times 1 \times 64$, and if you do that, the entire $56 \times 56 \times 64$ will collapse to 56×56 . Because for each of those special locations, a $1 \times 1 \times 64$ convolutions will give you one scalar as an output, which means you will be left with one pixel here, one pixel here, one pixel here so on, and so forth. In the next layer which would lead to a 56×56 .

So, remember the filter size for this 1×1 convolution would have been $1 \times 1 \times 64$; that is the 1×1 convolution of the Bottleneck, convolution of Bottleneck layer we are talking about. But they had 32 such filters, which means 32 such $1 \times 1 \times 64$ filters; which results in the output now becoming $56 \times 56 \times 32$ because of 32 such filters. So, by doing the 1×1 convolution, a $56 \times 56 \times 64$ volume became a $56 \times 56 \times 32$ volume.

And you can imagine now that you could have instead 32 filters, you could have just chosen 3 filters, 4 filters so on and so forth. So, the depth could have been arbitrarily reduced by using this trick of doing 1×1 convolution; and that is what the GoogleNet designers did in the inception module. What does this help with? This preserves spatial dimensions and reduces depth, and it projects the depth to lower dimensions.

(Refer Slide Time: 7:02)



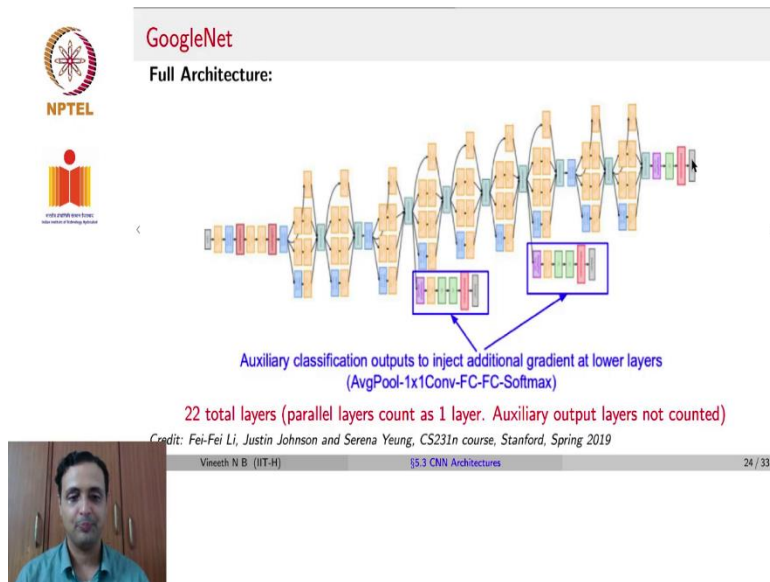
Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019



So, now the final inception module was the Naïve inception module we saw a couple of slides back. The final inception module does the same convolutions, but now before doing those convolutions applies 1x1 convolution to reduce the depth. Which, means the number of convolutions or operations that you have to do will reduce because of that and then does 3x3 of 5x5 convolutions. This trick helps them become far more efficient in the number of computations. Also, you see here that there is a 1x1 convolution after Max-pool, instead of before Max-pool when compared to the other branches.

The reason for that is Max-pool maintains depth, it maintains the depth at the same thing as what was given. So, when you do Max-pooling, the Max-pooling is done in a 2x2 neighborhood on each channel in the volume. So, for every channel the number of channels remains the same; so they first do the Max-pooling and then do 1x1 convolution to reduce the number of channels. This inception module brings down or increases the efficiency, brings down the computations; but increases the efficiency of the GoogleNet.

(Refer Slide Time: 8:25)



So, here is the complete architecture, so is the this is where the input comes in; and then you have a small stem network which is a Conv-Pool. And two times a Conv-pool, so you repeat that; then comes all these inception modules, stacked one after the other. And at the end, you have classifier output with no fully connected layers, and in between, you see here that there are auxiliary classifier outputs. This is because it is possible that a gradient in the last layer may not reach earlier layers when you have many stacked layers across the neural network.

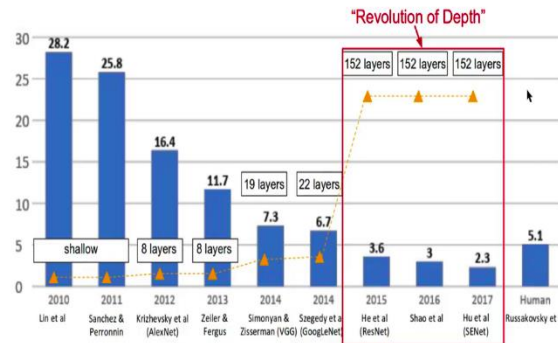
So, they also had intermediate classification outputs, where they took the output of a particular layer had a bunch of classification layers that had a loss. And that loss is gradient was also backpropagated at these locations to improve the strength of the gradient to earlier layers for weight update. As you can see the classification layer both here and here; had a structure of average pool 1x1 convolution FC-FC Softmax. There were 22 layers in total, the parallel layers count as one layer, so these different layers that you see inside the inception module.

The branches we just counted as one layer, given that it has a total of 22 layers; and the auxiliary output layers are not counted. Just from this end to the other end, together it is 22 layers in GoogleNet.

(Refer Slide Time: 10:08)



Deeper the Merrier



Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

Vineeth N B. (IIT-H)

§5.3 CNN Architectures

25 / 33



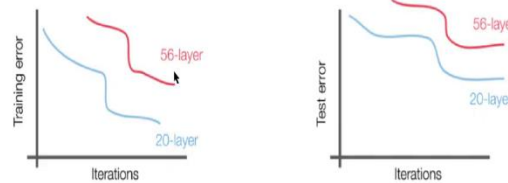
So, as you can see each year since 2012, the networks got deeper and deeper and deeper. So, the obvious question the researchers in the next year had was how far can we take it? And they took it to about 150 layers which we will see in a moment.

(Refer Slide Time: 10:28)



How deep can we go?

What happens when we continue stacking deeper layers on a "plain" convolutional neural network?



Deeper model does worse than shallow model! **Why?**

The deep model is actually **underfitting** since it also performs worse than the shallow model on the training set

Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford; Justin Johnson, Univ of Michigan

Vineeth N B. (IIT-H)

§5.3 CNN Architectures

26 / 33





So, while asking this question, they try to understand that how deep can you go with CNN. And when they studied stacking deeper and deeper layers on a plain CNN, they found that 20 layer CNN had a lower training error over the iterations of training than a 56 layer CNN. This was a

little non-intuitive and the same observation was also made on the test error in the same setting. So, the 20 layer CNN had a lower test error over iterations when compared to a 56 layer CNN.

Why would this be happening? A deeper model performs worse than a shallow model. This was a little counterintuitive to how the research was progressing with CNNs at that time. One possible explanation you could give for why this could be happening is the initial guess could be that the deep model is perhaps overfitting, and that is why a test error increases for 56 layers neural network. You have more parameters and it perhaps is becoming complex and hence is overfitting. But, if that was true the training error would have been stronger for a 56 layer network over a 20 layer network.

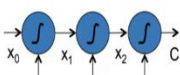
Unfortunately, even there the 20 layer network had a lower error, which means the 56 layer network was also underfitting. What would have been the reason for this to happen as the depth of the neural network increases?

(Refer Slide Time: 12:18)

How deep can we go? Vanishing/Exploding Gradient

Consider a simple network:

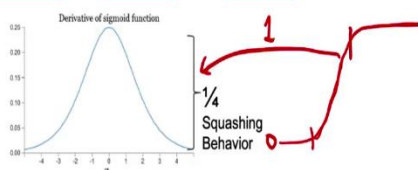



$$\frac{\partial L}{\partial x_0} = \sigma'(w_3^T x_2) \times w_3 \times \sigma'(w_2^T x_1) \times w_2 \times \sigma'(w_1^T x_0) \times w_1$$

$\begin{matrix} < 1/4 \\ \downarrow \\ \sigma'(w_3^T x_2) \end{matrix}$

$\begin{matrix} < 1/4 \\ \downarrow \\ \sigma'(w_2^T x_1) \end{matrix}$

$\begin{matrix} < 1/4 \\ \downarrow \\ \sigma'(w_1^T x_0) \end{matrix}$





Vineeth N B. (IT-H)
§5.3 CNN Architectures
27 / 33


The answer for this lies in what is known as the vanishing gradient problem. The problem complementary to vanishing gradient is known as exploding gradient, but let us talk about each of them in one after the other. So, if you took a simple feed-forward neural network, let us assume with sigmoid activations in each layer. So, here is a very simplistic visual for that. You have x_0 , there is a weight one that comes in the first layer, there is a sigmoid; then x_1 , w_2 is a weight in the second layer, a sigmoid, and so on until the last layer.

If you now took a loss L and wanted to find $\frac{\partial L}{\partial x_0}$; this is one of the initial layer's inputs. You have $\frac{\partial L}{\partial x_0} = \sigma'(w_3^T x_2)w_3\sigma'(w_2^T x_1)w_2\sigma'(w_1^T x_0)w_1$. σ' is the gradient of the sigmoid function. This comes from applying the chain rule. You work out the gradient in each step, considering a sigmoid activation function in each layer; you would get this.

But, the key point I would like to draw you towards here is that the gradient of the sigmoid is less than 1 by 4. Why is that the case? Because this is how the gradient of the sigmoid function looks. It starts at a 0 close to 0, it peaks at 0.25 and then falls again. This comes from the shape of the sigmoid function, so the gradient is deep in this region. It increases and starts falling and flattens out again, so the gradient of such a sigmoid function turns out to be something like this; and the range is between 0 and 1, which means the gradient turns out as capping at 0.25. This means $\frac{\partial L}{\partial x_0}$ has a bunch of values, which are all less than one-fourth.

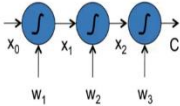
What you expect will happen to $\frac{\partial L}{\partial x_0}$; the product of very small values will quickly go to 0.

(Refer Slide Time: 14:52)




How deep can we go? Vanishing/Exploding Gradient

Consider a simple network:



$$\frac{\partial L}{\partial x_0} = \overset{< 1/4}{\sigma'(w_3^T x_2)} \times w_3 \times \overset{< 1/4}{\sigma'(w_2^T x_1)} \times w_2 \times \overset{< 1/4}{\sigma'(w_1^T x_0)} \times w_1$$

- **Vanishing gradients:** Deeper the network, gradients vanish quickly, thereby slowing the rate of change in initial layers
- **Exploding gradients:** Happen when the individual layer gradients are much higher than 1, for instance - can be overcome by **gradient clipping**



Vineeth N. B. (IIT-H)
§5.3 CNN Architectures
27 / 33

And you have the problem of what is known as vanishing gradients. As you go deeper and deeper into a neural network, the gradient especially if you have such squashing activation functions, the gradients are less than one and as they get multiplied over the several layers during backpropagation; the gradient becomes feeble and feeble as it proceeds towards the initial layers.


In other words, as the gradients never reach the initial layers or the initial layers, may get stuck with the random initialization that we came up with, and may never go further.

So, you could argue that instead of randomly initializing the neural network with some random weights. And then not updating them at all convincingly, why have those layers in the first place? Let me not take such a deep network; that could be one way of arguing against solving the vanishing gradient problem. But, will talk about a different approach in a slide from now. The complementary problem known as the exploding gradients can happen, when maybe there is another activation function, such as ReLU; where the gradient could be 1 or greater than 1 or depends on what the activations are.


Or, as I said any other function where these terms end up becoming greater than 1, the product of those values as you go deeper through many layers can explode to a high value and that also can cause problems. However, exploding gradients are often easily handled by a trick known as gradient clipping; this is very popularly used in practice. People just say that when the gradient goes beyond 10 for instance, you assume the gradient to be 10.

So, anything greater than 10 will just be 10 you just clip it, and at that value and that will ensure that the gradient does not explode.

(Refer Slide Time: 17:02)



NPTEL

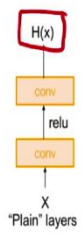


NPTEL

ResNet

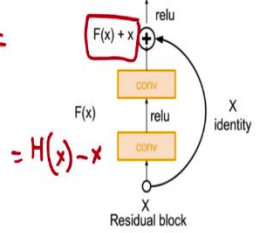
The deeper model should be able to perform at least as well as the shallower model; *how?*

Solution: Change the network with identity connections between layers:



"Plain" layers


=



Residual block

Use network layers to fit residual $F(x) = H(x) - x$ instead of $H(x)$ directly

Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019





Coming back to the vanishing gradients problem, this problem was addressed in CNN architecture in 2015 called ResNet or residual nets. They wanted to ensure that a deep model should at least perform as well as a shallow model. How is this possible? The idea they came up with was to introduce what are known as residual blocks, which are connected through identity connections or identity matrices. What does that mean? In the original CNN, you have an x ; it goes through a few layers such as Conv layer, then a ReLU, then a Conv layer. And it gets transformed over those layers, and becomes a certain $H(x)$; this could be one portion of a neural network for instance.

But, in a residual block what these designers came up with was to say that x goes through a Conv layer. A ReLU, a Conv just as Vanilla CNN; but they also have a side channel to take the input x as it is. And added up here two layers later, which means what was originally the function $H(x)$; now becomes $F(x)$ which is a new transformation that happens through these layers. Plus an x that comes directly here and gets added at this layer after the Conv. Each such block where the identity connections come back and add the original input is known as a residual block. Why is it a residual block?

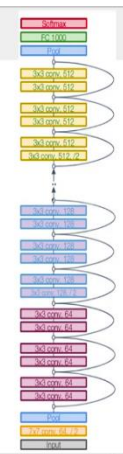
Because originally these layers captured a function $H(x)$; but now these layers capture the residual $H(x) - x$. Remember, now we are saying here that this together here must be equal to $H(x)$; or rather $F(x)$ will be $H(x) - x$, which is now residual rather than $H(x)$ itself.


(Refer Slide Time: 19:26)

ResNet

- A residual network is a stack of many residual blocks
- Each residual block has two 3×3 conv layers
- Periodically, double number of filters and downsample spatially using stride 2 ($/2$ in each dimension)
- Use **global average pooling** and a single linear layer at the end (FC 1000 to output classes)
- Total depths of 34, 50, 101, or 152 layers for ImageNet dataset





Vineeth N B (IIT-H)
35.3 CNN Architectures
29 / 33

So, the way the residual network was designed is to have a stack of many residual blocks. Each residual block had two 3 by 3 Conv layers as you can see, 3 by 3 Conv layers; and then the x that came out of this. Or, whatever representation came out from this particular joint here, was directly passed on again at the to the output of those two Conv layers. And this was repeated again and again over the blocks in the neural network.

Periodically, the number of filters was doubled, so you can see here that 64 which was the number of filters as in the initial residual blocks became 128 after the set of residual blocks. Which after a few more residual blocks became 512 and so on and it was also down-sampled spatially using stride 2 that is divided by 2 in each dimension to bring the size of the feature maps lower as you went deeper. Remember that helps you control the number of neurons in later layers by downsampling, the feature maps. They also used a technique called global average pooling.

Global average pooling is like Max-pooling average pooling, but global average pooling is the entire average of a feature map. So, if you have a feature map, you take the average of all the pixels in that feature map and it gives you one scalar. So, if you had k channels in a particular layer, by doing like global average pooling; you will get k scalars, 1 scalar per channel which will be the global average of that channel. You can imagine that as giving a feature map taking the average intensity of that feature map; that is global average pooling. And then for each channel, you would get one global average pool value.

So, if you add k feature maps, you would get a k dimensional vector as the output of global average pooling. And they had a single linear layer at the end you can see, that there are no fully connected layers here; but for the FC 1000. FC 1000 is done because ImageNet has 1000 classes, so the FC 1000 is required to give 1000 outputs in that last layer, and finally a Softmax to convert those logits into probabilities. They had total depths of 34, 50, 101, and 152 layers for ImageNet. So, it is called ResNet 34, ResNet 50, ResNet 101, ResNet 152 so on and so forth.

(Refer Slide Time: 22:17)

The slide features the NPTEL logo and the text: "For deeper networks (ResNet-50+), use 'bottleneck' layer to improve efficiency (similar to GoogLeNet)".

The diagram illustrates a bottleneck layer. It starts with a $28 \times 28 \times 256$ input. The first step is a 1×1 convolution with 64 filters, projecting to $28 \times 28 \times 64$. The second step is a 3×3 convolution operating over only 64 feature maps. The final step is another 1×1 convolution with 256 filters, projecting back to $28 \times 28 \times 256$. A residual connection bypasses these operations and is added to the output of the final 1×1 convolution.



Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

Vineeth N B. (IIT-H) 5.3 CNN Architectures 30 / 33

For very deep networks, which went beyond 50 layers in a ResNet; they also used a Bottleneck layer. You can see the 1×1 convolutions to improve the efficiency similar to GoogLeNet. Remember the 1×1 convolutions help you reduce the depth from a certain depth to a smaller depth and then you perform your operations at that smaller depth. And you can again now do 1×1 convolution to go back to an initial depth if you like which is something that they did do to keep the depth in the same range. But, reduce the computations by a couple of 1×1 or bottleneck layers, a couple of 1×1 convolutions, or Bottleneck layers periodically.

So, this is the example they start with they had a $28 \times 28 \times 256$ input; the bottleneck did it to $28 \times 28 \times 64$ by doing 1×1 convolution. Then do the 3×3 convolutions only on those 64 feature maps and then once again did 1×1 convolution, and took it back to $28 \times 28 \times 256$. This keeps the dimension the same as the input, but these 1×1 convolutions make the number of computations far lesser, and hence the training more efficient.

(Refer Slide Time: 23:43)



ResNet


- Able to train very deep networks
- Deeper networks perform better than shallow networks now (as expected); residual blocks help avoid vanishing gradient
- 1st place in all ILSVRC and COCO 2015 competitions
- We will discuss detection, localization, segmentation and the COCO dataset a bit later

ResNet @ ILSVRC & COCO 2015 Competitions

1st place in all five major challenges

- ImageNet Classification: "Ultra-deep" 152-layer nets
- ImageNet Detection: 16% better than the 2nd best
- ImageNet Localization: 27% better than the 2nd best
- COCO Detection: 11% better than the 2nd best
- COCO Segmentation: 12% better than the 2nd best

Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019



Vineeth N B. (IIT-H) | 5.3 CNN Architectures | 31 / 33

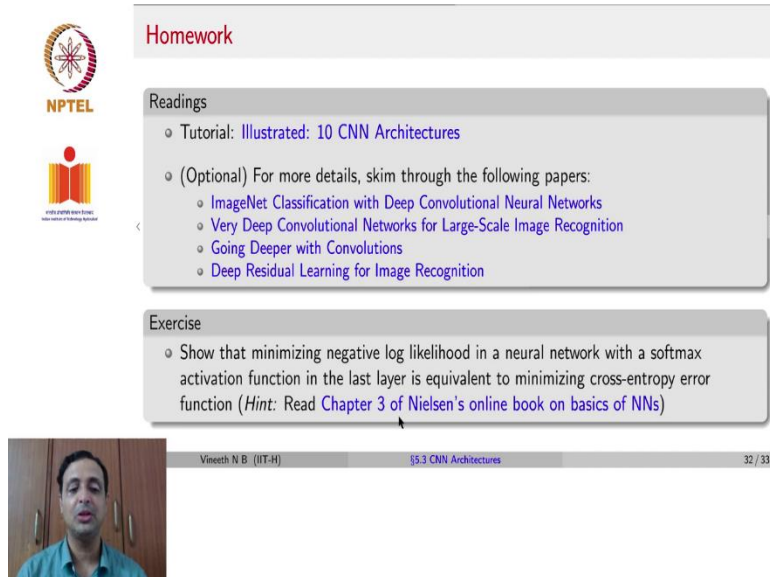
With this ResNet, they were able to train very deep networks. As we mentioned they went up to 152 layers to train on ImageNet, and they now found that deeper networks perform better than shallow networks. Why so? Why does this identity map or this residual block help train deeper networks? When we backpropagate the gradient that comes to a particular point in the residual network; while, there is a path of the gradient through these layers, which may result in an attenuation of the gradient. There is another side path for the gradient to go as it is to a previous layer, without any diminishing in its value.

And these identity connections that connect residual blocks are like serve like highways, for the gradient to just flow through; they are like short circuits. The gradient just flows through them, while there is a part of the gradient that still keeps coming through the layers. And that allows even the earlier layers to get a strong gradient signal, and now deeper networks perform better than shallow networks. And it won first place in all ImageNet competitions and also a competition called COCO that was introduced in 2015.

So, they won the char first place in all the major challenges in 2015. We mentioned classification ImageNet detection, where they were 16 percent better than the second-best ImageNet localization; where they were 27 percent better than the second-best COCO detection, where they were 11 percent better than the second-best. COCO's segmentation where they were 12 percent better than the second-best. We will discuss detection, localization, segmentation, and the COCO dataset a bit

later. But, just to let you know that this model swept through all challenges in 2015. And has since then been an important architecture that people use for various applications in computer vision.

(Refer Slide Time: 25:56)



The screenshot shows a presentation slide with the following content:

- NPTEL** logo (National Programme on Technology Enhanced Learning)
- Homework** section header
- Readings** section:
 - Tutorial: [Illustrated: 10 CNN Architectures](#)
 - (Optional) For more details, skim through the following papers:
 - [ImageNet Classification with Deep Convolutional Neural Networks](#)
 - [Very Deep Convolutional Networks for Large-Scale Image Recognition](#)
 - [Going Deeper with Convolutions](#)
 - [Deep Residual Learning for Image Recognition](#)
- Exercise** section:
 - Show that minimizing negative log likelihood in a neural network with a softmax activation function in the last layer is equivalent to minimizing cross-entropy error function (*Hint: Read Chapter 3 of Nielsen's online book on basics of NNs*)

At the bottom of the slide, there is a video feed of a man speaking, and a footer with the text: "Vineeth N. B. (IIT-H) 35.3 CNN Architectures 32 / 33".

So, the homework for this lecture is to go through this illustration of 10 CNN architectures, which is very well done which would perhaps help you understand these architectures better. If you are further interested, you can go through these respective papers, which are for ImageNet, ResNets, VGGs so on which could probably help you understand some of the decisions; which were made in the design of these architectures.

The exercise for this lecture is going to show that minimizing negative log-likelihood in a neural network with a Softmax activation function in the last layer, is equivalent to minimizing a cross-entropy loss function. We are not going to work this out in the next lecture, please read this chapter 3 of Nielsen's online book, if you would like to understand this further.

(Refer Slide Time: 26:57)



References

- Yann LeCun et al. "Gradient-based learning applied to document recognition". In: 1998.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *NIPS*. 2012.
- Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2015).
- Christian Szegedy et al. "Going deeper with convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 1–9.
- Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.
- Johnson, Justin, *EECS 498-007 / 598-005 - Deep Learning for Computer Vision (Fall 2019)*. URL: <https://web.eecs.umich.edu/~justincj/teaching/eecs498/> (visited on 06/29/2020).
- Li, Fei-Fei; Johnson, Justin; Serena, Yeung; *CS 231n - Convolutional Neural Networks for Visual Recognition (Spring 2019)*. URL: <http://cs231n.stanford.edu/2019/> (visited on 06/29/2020).



Vineeth N B. (IIT-H)

§5.3 CNN Architectures

33 / 33

With that here are your references.