Deep Learning For Computer Vision

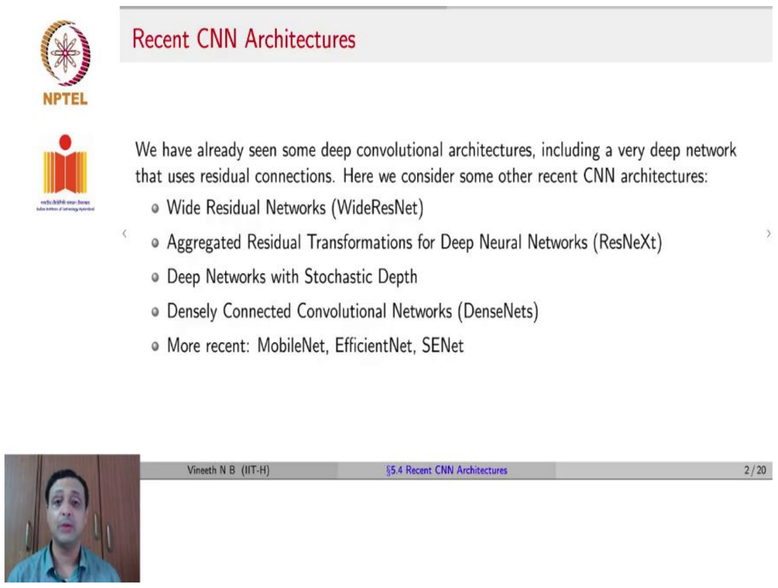
Professor Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Lecture 36
Recent CNN Architectures

In the architectures that we have discussed so far, you will notice that whatever components were added be the inception module the bottleneck layer, or the residual block none of them affect backpropagation in any way. They are all different ways of connecting inputs out to outputs and as long as one computes the chain rule carefully and accounts for the gradient through all possible paths that reach a particular node backpropagation can be implemented like this.

In this lecture, we will move on to talk about a few more recent CNN architectures that have been developed since residual nets. ResNet won the ImageNet challenge in 2015 and has continued to be a popular choice for several applications. We will talk about ways in which they have been extended in this lecture.

(Refer Slide Time: 1:22)



Recent CNN Architectures

We have already seen some deep convolutional architectures, including a very deep network that uses residual connections. Here we consider some other recent CNN architectures:

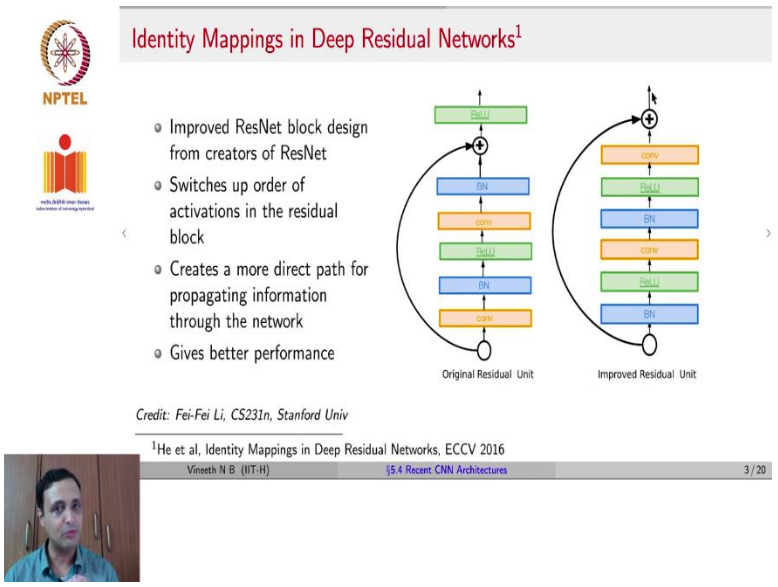
- Wide Residual Networks (WideResNet)
- Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)
- Deep Networks with Stochastic Depth
- Densely Connected Convolutional Networks (DenseNets)
- More recent: MobileNet, EfficientNet, SENet

Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 2 / 20

In particular, we will talk about WideResNets, ResNeXt which is an extension of residual nets.

Deep networks with stochastic depth DenseNets, MobileNet, EfficientNet and squeeze excitation net so on and so forth. Hopefully, this will give you a flavor of how you can improve CNN architectures.

(Refer Slide Time: 1:51)



Identity Mappings in Deep Residual Networks[1]

- Improved ResNet block design from creators of ResNet
- Switches up order of activations in the residual block
- Creates a more direct path for propagating information through the network
- Gives better performance

ReLU
BN
conv
ReLU
BN
conv
Original Residual Unit
conv
ReLU
BN
conv
ReLU
BN
Improved Residual Unit

Credit: Fei-Fei Li, CS231n, Stanford Univ

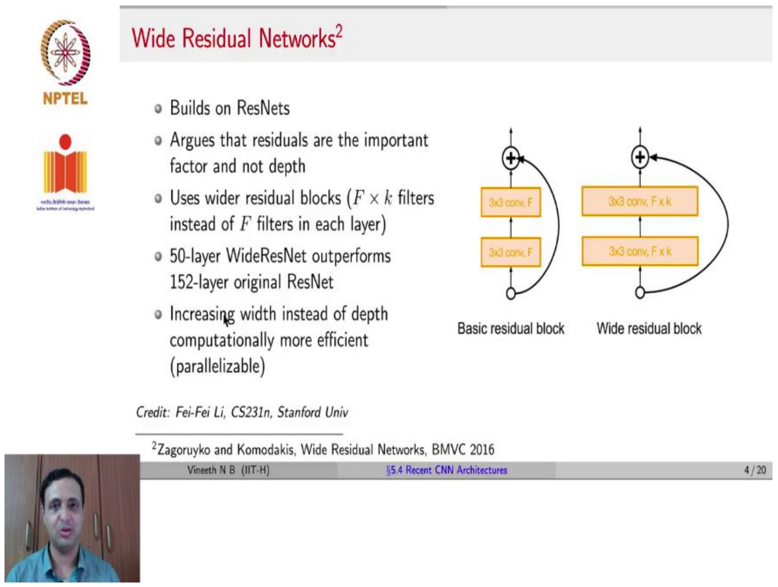
[1]He et al, Identity Mappings in Deep Residual Networks, ECCV 2016

Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 3 / 20

One of the earliest works that followed ResNets was a work that tried to understand the importance of identity mapping in residual networks. As we discussed last class the identity mapping between residual blocks allows the gradient to flow through a highway during backpropagation. The design the creators of ResNet improved their design here with this effort where if you notice the original residual unit here at the end of the residual block there was a ReLU that was applied on the output of the residual block.

The creators of ResNet noticed that could affect the flow of information through that direct path and hence change the design to bring ReLU into the residual block. And this they found created a more direct path for propagating information because now this identity map connection does is not succeeded by ReLU and the information passes as is. And it was also found to give better performance in their empirical studies.

(Refer Slide Time: 3:16)



Wide Residual Networks[2]

- Builds on ResNets
- Argues that residuals are the important factor and not depth
- Uses wider residual blocks ($F \times k$ filters instead of F filters in each layer)
- 50-layer WideResNet outperforms 152-layer original ResNet
- Increasing width instead of depth computationally more efficient (parallelizable)

Credit: Fei-Fei Li, CS231n, Stanford Univ

[2]Zagoruyko and Komodakis, Wide Residual Networks, BMVC 2016

Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 4 / 20

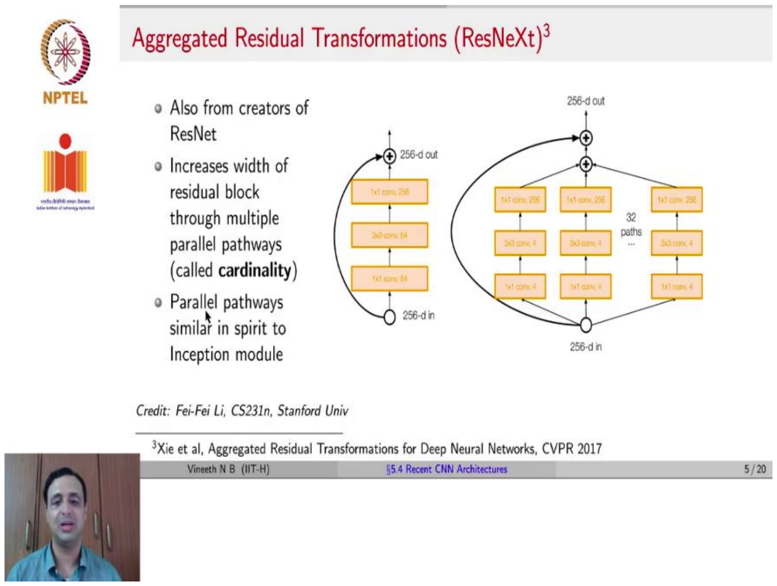
A popular variant of ResNets which is used a lot today is known as WideResNets. As the name suggests, it was an improvement of residual nets but the objective in this design was a hypothesis that residual networks work because of the residual block and not because of increasing the depth. In residual networks, remember on one hand the residual blocks were introduced to mitigate the vanishing gradient problem.

On the other hand, the authors argued that allowed us to create deeper and deeper networks which gave better performance. But in WideResNets, the authors argued that you do not need depth as long as you have strong residual blocks that can give you good performance. So to show that and to study that hypothesis they increased the width of each residual block by multiplying the number of filters k fold.

In the basic residual block, if you had two three to three convolutions with f filters each or f feature maps each that is again if you recall the depth of the output of each layer. They increased each of them k fold and made it fxk and they showed that with this increase of width in each residual block, a 50 layer WideResNet could outperform a 152 layer original ResNet. This increases the width instead of depth can also make computation efficient through techniques like parallelization.

If things are done one after the other which is what happens when you increase the depth it is not possible to parallelize. But when you increase the width you could potentially send each filter to a different computing unit and then retrieve the results and take it forward. So, increasing the width instead of depth has a computational benefit too.

(Refer Slide Time: 5:38)



Aggregated Residual Transformations (ResNeXt)[3]

- Also from creators of ResNet
- Increases width of residual block through multiple parallel pathways (called **cardinality**)
- Parallel pathways similar in spirit to Inception module

Credit: Fei-Fei Li, CS231n, Stanford Univ

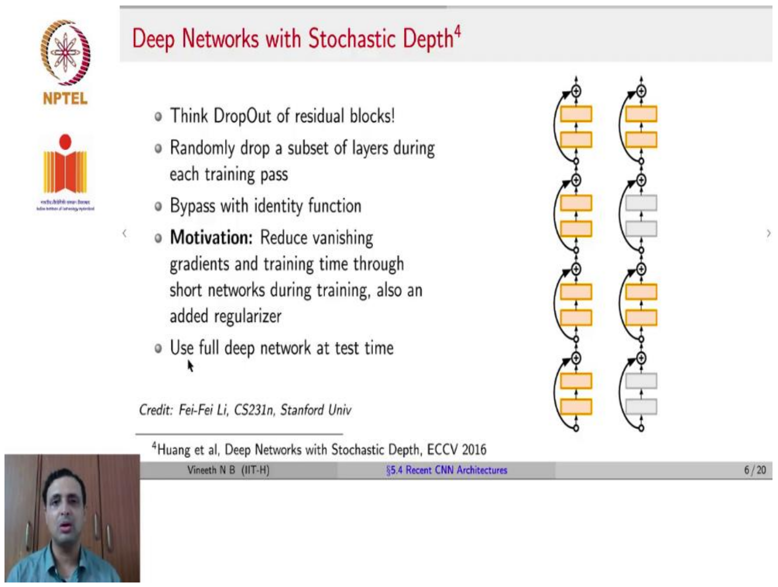
[3]Xie et al, Aggregated Residual Transformations for Deep Neural Networks, CVPR 2017

Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 5 / 20

Another work that was again by the creators of ResNet was known as aggregated residual transformations, which was also called ResNeXt. As the visual shows, if the left block is the original residual block, the right block which was proposed in ResNeXt was multiple branches, multiple branches in each residual block. So you can see here that you have a one-by-one Conv with four filters and a three-by-three Conv with four filters, then a one-by-one Conv with 256 filters. So this reduces the depth to four and then increases it to 256, keeping in mind the same idea of the bottleneck layer.

So they did this along 32 different paths and showed that this could improve performance. In a sense, this tried to bring the idea from the inception module into residual networks.

(Refer Slide Time: 6:42)


Deep Networks with Stochastic Depth[4]

- Think DropOut of residual blocks!
- Randomly drop a subset of layers during each training pass
- Bypass with identity function
- **Motivation:** Reduce vanishing gradients and training time through short networks during training, also an added regularizer
- Use full deep network at test time

Credit: Fei-Fei Li, CS231n, Stanford Univ

[4]Huang et al, Deep Networks with Stochastic Depth, ECCV 2016

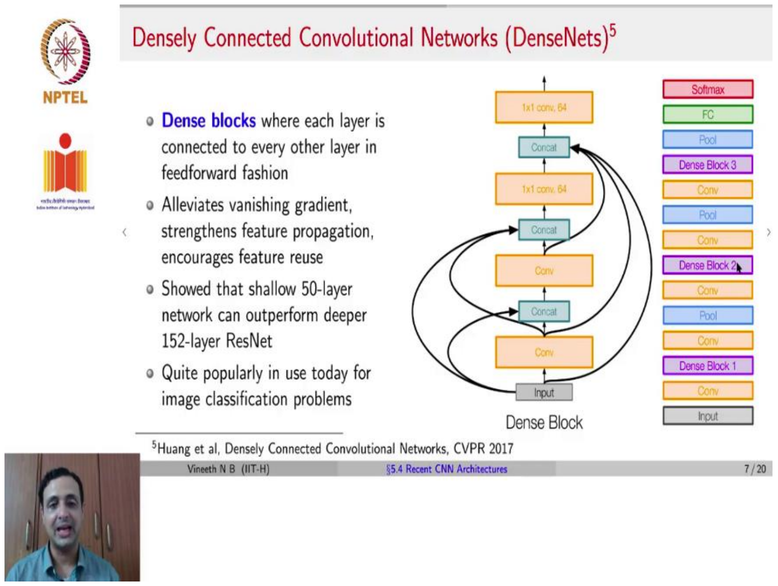
Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 6 / 20

Another interesting extension of residual networks was the idea of deep networks with stochastic depth. The way to think of this approach is to think of dropout in terms of residual blocks because now residual blocks are connected through identity connections. This now gives you an interesting direction for dropping layers. You could not have done this with simple feed-forward networks or CNN speed forward CNNs. Because if you remove the layer the network would get disconnected but now because of the presence of the identity connections through residual blocks you could randomly drop a few of the residual blocks in every mini-batch iteration.

And that could be done in a stochastic way by randomly sampling from a uniform distribution or any other probability distribution and accordingly deciding which blocks should be dropped. So the motivation is this reduces vanishing gradients further and it can also reduce training time because the forward propagation is perhaps shorter now and it can also act as an added regularizer, some more noise just like how dropout serves as a regularizer.

And at test time, very similar to dropout again, you use the full network as it is very similar to how it is done in dropout.

(Refer Slide Time: 8:24)



Densely Connected Convolutional Networks (DenseNets)[5]

- **Dense blocks** where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse
- Showed that shallow 50-layer network can outperform deeper 152-layer ResNet
- Quite popularly in use today for image classification problems

[5]Huang et al, Densely Connected Convolutional Networks, CVPR 2017

Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 7 / 20

Moving on from these methods came a method known as DenseNets which is perhaps after ResNets and a version of ResNets known as inception ResNets. DenseNets are perhaps the most popular choice for computer vision tasks. As the name suggests, DenseNets stands for densely connected convolutional networks.

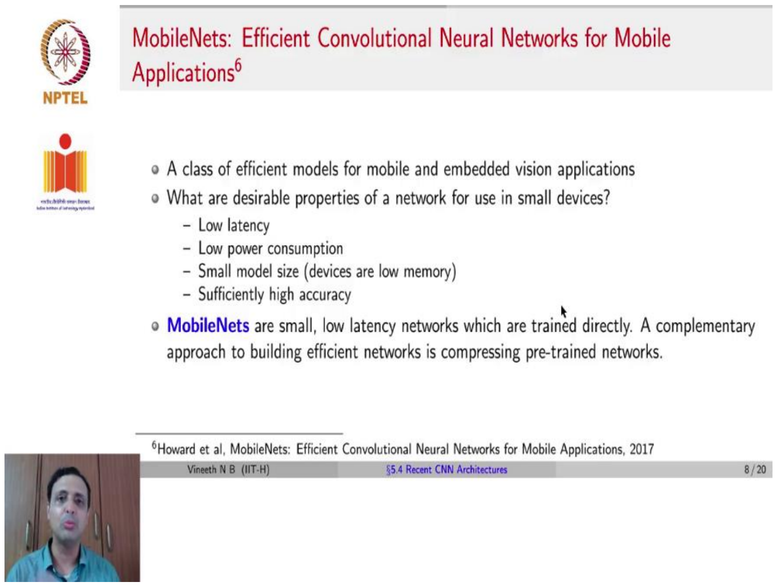
In this case, you have a dense block similar to a residual block but a dense block where each layer is connected to every other layer. Remember in ResNets you had identity connections only periodically after every residual block. But in a DenseNet within a dense block, you have identity connections from every layer to every other layer. You also call them to skip connections.

These connections that take you from one layer to another layer are called identity connections because the weight on that connection is identity. They are also called skip connections because you are skipping a few layers in between and going straight away to a later layer. Similar to ResNets this also alleviates the vanishing gradient because of those identity connections that give keep giving a stronger gradient through these DenseNet blocks.

And they showed that using this approach a shallow 50 layer network can outperform a deeper 152 layer ResNet. Of course, you have to keep in mind that the word shallow here is a relative term with respect to 152 layers. This is popularly in use today for image classification. You can see the overall architecture of this network on the far right here where you see that you have an input, you have a conv layer, and then comes one entire dense block of all of these dense connections.

So you can see again that within each dense block, you have a conv layer, then you concatenate using the input, then you have a conv layer you again concatenate using the input from the previous Conv layer as well as the original input. Then you again have a conv layer and then concatenate and then you can have a conv layer before you come out of that dense block. After the dense block, you have a conv pool conv once again dense block Conv pool conv a dense block pool fully connected and then finally a Softmax layer for classification.

(Refer Slide Time: 11:12)



MobileNets: Efficient Convolutional Neural Networks for Mobile Applications[6]

- A class of efficient models for mobile and embedded vision applications
- What are desirable properties of a network for use in small devices?
  - Low latency
  - Low power consumption
  - Small model size (devices are low memory)
  - Sufficiently high accuracy
- **MobileNets** are small, low latency networks which are trained directly. A complementary approach to building efficient networks is compressing pre-trained networks.

[6]Howard et al, MobileNets: Efficient Convolutional Neural Networks for Mobile Applications, 2017

Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 8 / 20

As architectures started evolving through the interesting use of skip connections all the variants of ResNet and DenseNets were different ways of using skip connections to avoid the vanishing gradient problem as well as make computations more efficient. There was another need emanating as CNNs became popular for vision applications which were can we use vision applications on low power devices, embedded devices, or what are known as edge devices, devices that sit at the edge at the cusp of interaction with users or an environment for that matter.

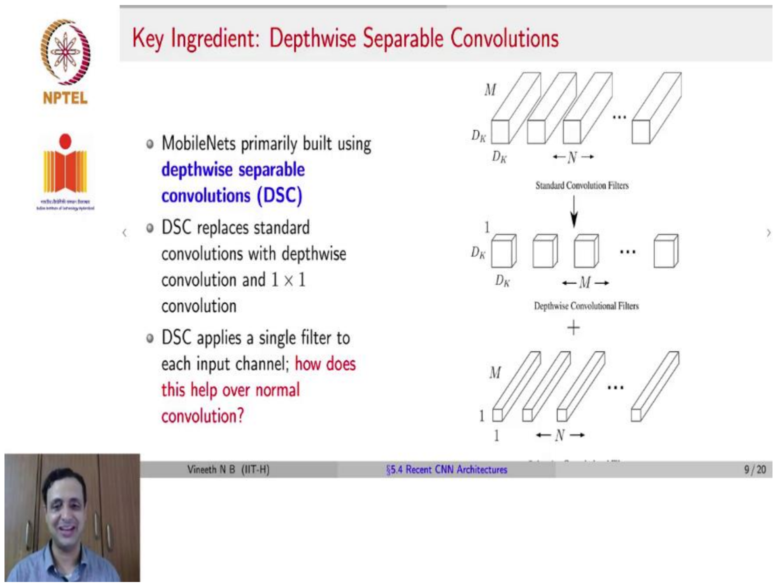
MobileNets was one such effort that refers to a class of efficient models for mobile and embedded vision applications it was developed in 2017 for the first time. And before we describe MobileNets, what are the desirable properties of a CNN for use in small devices? You need low latency which means the forward prop of a neural network should be real-time, should be should not have latency, low power consumption, running a neural network should not consume too much power.

Often in these edge devices, there is very little there the power sources are very little and they may have even other components to run. So you do not want too much power to go into the running of a neural network. You want the model size to be small we saw that from AlexNet to VGG there was a model size increase. But then with ResNets, the model's ties started decreasing while being able to achieve similar performance.

And you still want at least a sufficiently high accuracy, you may be found to drop one or two percentage points in accuracy but at significantly smaller model size and a low power footprint. MobileNets are aimed in that direction they are small, low latency networks which are trained directly. A complementary approach to this problem which has also been popular over the last few years is the idea of compressing neural networks.

Compressing while in compression of neural networks you try to prune out weights or there are other methods in which you can try to remove the weights that you think do not matter for the outcome. But MobileNets try to do this by design.

(Refer Slide Time: 14:03)



Key Ingredient: Depthwise Separable Convolutions

- MobileNets primarily built using **depthwise separable convolutions (DSC)**
- DSC replaces standard convolutions with depthwise convolution and 1×1 convolution
- DSC applies a single filter to each input channel; how does this help over normal convolution?

M
D_K
D_K
N
Standard Convolution Filters
1
D_K
D_K
M
Depthwise Convolutional Filters
M
1
1
N

Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 9 / 20

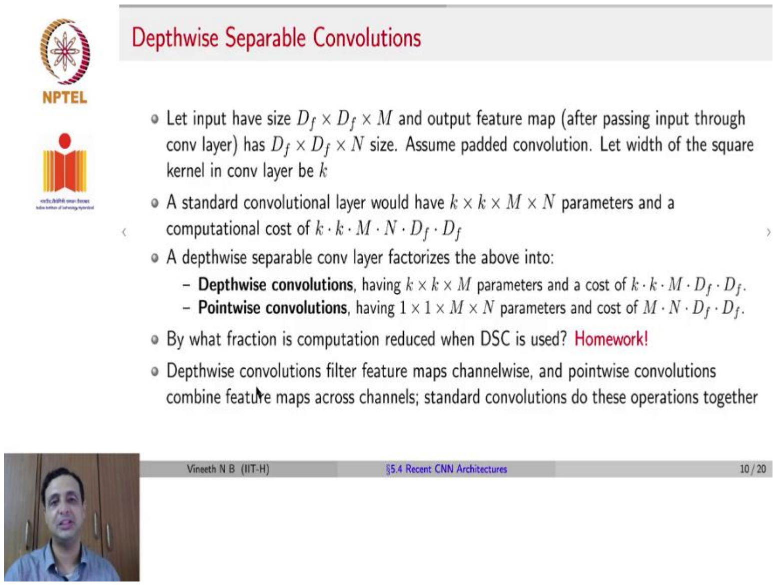
The key ingredient of MobileNets is one of the variants of convolution that we talked about in the first lecture this week which is depth-wise separable convolutions.

In your standard way of convolving it would be to do a D_kxDKxM convolution and N such filters. But now in depth wise convolutional separable convolutions what you do is you first convolve only on each channel. So which means this is D_kxD_kx1, M different times. So if there are M channels you do D_kxD_kx1, D_kxD_kx1on the second channel, D_kxD_kx1 on the third channel and so on until M channels.

This is going to give you a certain set of outputs. Now, on these outputs, you concatenate and then run a 1x1xM convolution along with the depth. Remember each D_kxD_kx1 will give you an output that is the same size as the input. For instance, if you pad and you would have M such outputs, now you do one-on-one M convolution across all of those channels you will have N such 1x1xM filters to finally give you the same output that you would have got with standard convolution.

Let us see this in some more detail. So DSC which is depth-wise separable convolutions replaces standard convolutions with depth-wise convolution followed by 1x1 convolution. As you can see, 1x1 convolution is a very popular choice to reduce computations in CNNs. And DSC applies a single filter to each input channel. So how do you think this helps over standard convolution?

(Refer Slide Time: 16:13)



Depthwise Separable Convolutions

- Let input have size $D_f \times D_f \times M$ and output feature map (after passing input through conv layer) has $D_f \times D_f \times N$ size. Assume padded convolution. Let width of the square kernel in conv layer be k
- A standard convolutional layer would have $k \times k \times M \times N$ parameters and a computational cost of $k \cdot k \cdot M \cdot N \cdot D_f \cdot D_f$
- A depthwise separable conv layer factorizes the above into:
  - **Depthwise convolutions**, having $k \times k \times M$ parameters and a cost of $k \cdot k \cdot M \cdot D_f \cdot D_f$.
  - **Pointwise convolutions**, having $1 \times 1 \times M \times N$ parameters and cost of $M \cdot N \cdot D_f \cdot D_f$.
- By what fraction is computation reduced when DSC is used? Homework!
- Depthwise convolutions filter feature maps channelwise, and pointwise convolutions combine feature maps across channels; standard convolutions do these operations together

Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 10 / 20

Let us try to take a few dimensions and analyze this carefully. Assume that input is D_fxD_fxM and output feature map after a conv layer is D_fxD_fxM, assuming that the size did not reduce. Now if you assume padded convolution to ensure the size does not reduce. Now, let the square convolution kernel be K, rather than KxK. Now in a standard convolutional layer, this would have KxKxMxN parameters.

Because you would have KxKxM, KxK for the filter size and M for the depth and N such filters so that would lead to KxKxMxN. And a computational cost would be $K * K * M * N * D_f * D_f$ because that is the width and height of the input. Let us see what happens if you use a depth-wise separable convolution.

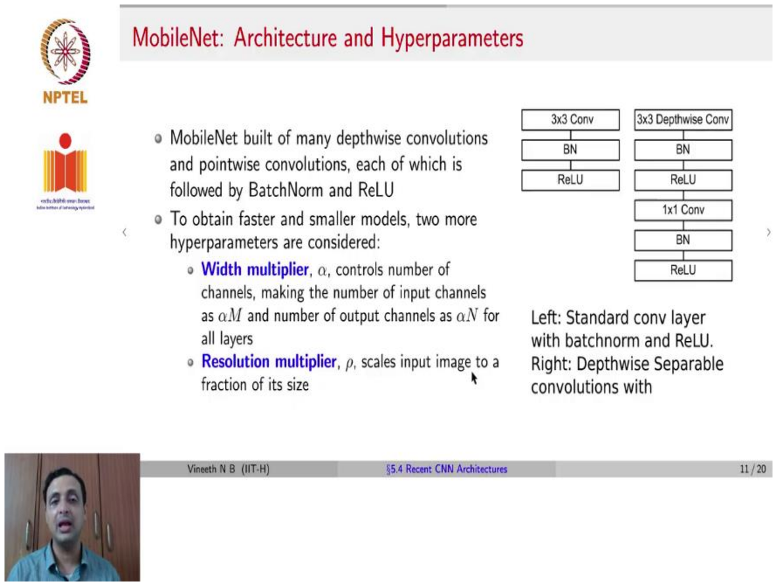
If you do depth-wise separable convolution this factorizes the above computations into two parts first depthwise convolutions, then point-wise convolutions right which is the 1x1xM part. The depth-wise convolutions which operate channel-wise have KxKxM parameters, each filter being KxK and you have M such for each channel. The total cost there would be $K * K * M * D_f * D_f$.

And the point-wise convolutions become 1x1xM which is the size of each filter and N such filters which means 1x1xMxN parameters. The total cost will be $M * N * D_f * D_f$. So if you now see what was multiplicative in terms of these parameters now becomes an addition of $K * K * M * D_f * D_f + M * N * D_f * D_f$. By what fraction is computation reduced?

You can try computing it yourself but clearly, you can see that this would become additive this would become multiplicative and that should tell you by what fraction the computations reduce. This is very similar in principle to separable convolutions that we saw way back in the first week. Remember, even inseparable convolutions instead of using a 3x3 filter which may have nine computations if you do a 3x1 filter followed by a 3x1 filter along the other dimension you are going to have three plus three which is 6 computations.

It is a very simple similar principle here and you can work this out as to what fraction of computation is reduced. To summarize, depth-wise convolutions filter feature maps channel-wise and point-wise convolutions combine the feature maps across channels. Standard convolution tries to do both at the same in the same step.

(Refer Slide Time: 19:35)


MobileNet: Architecture and Hyperparameters

- MobileNet built of many depthwise convolutions and pointwise convolutions, each of which is followed by BatchNorm and ReLU
- To obtain faster and smaller models, two more hyperparameters are considered:
  - **Width multiplier**, α, controls number of channels, making the number of input channels as αM and number of output channels as αN for all layers
  - **Resolution multiplier**, ρ, scales input image to a fraction of its size

Left: Standard conv layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with

Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 11 / 20

The MobileNet architecture also had a couple of other components to help improve the performance. So had after the depthwise convolution, they had a batch norm and a ReLU layer and after the pointwise convolution or rather a 1x1 convolution, they also had a batch norm and a ReLU layer that is what they found to give a good empirical performance. And to also obtain faster and smaller models they had two hyperparameters, a width multiplier that controls the number of channels based on a multiplier.

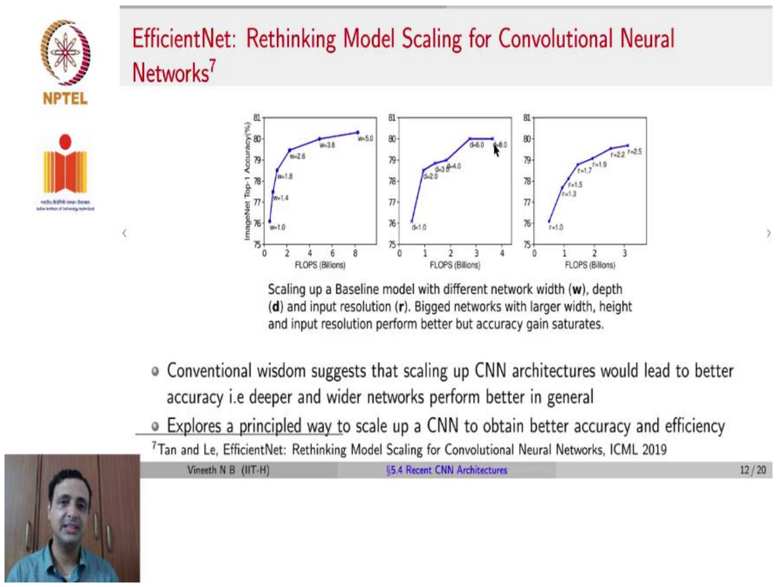
So if you had an M and an N for the input number of channels and the next layers output number of channels, α is a constant that scales M and N based on the constraints under which you are operating these models. MobileNets was intended to be a family of models which can be designed for certain constraints rather than a single model.

Until now, whatever model we saw ResNet, VGG, GoogleNet, ResNeXt any of those variants were one specific model with a specific architecture but MobileNets is about developing a family of architectures where the architecture can be fine-tuned based on certain constraints and these multipliers a width multiplier α and a resolution multiplier rho try to cater to that need of changing constraints.

It could be hardware constraints it could be power constraints. So based on a particular constraint you could use the width multiplier to automatically adjust the architecture the M and the N in the architecture would get scaled down or scaled up based on some constraints or what is what additional resources you may have in that case it could get scaled up. Similarly, the resolution multiplier scales the input image to a fraction of its size based on constraints that you may have in a particular setting.

So that allows you to use the same idea to develop a family of models that cater to different constraints in the setting of where this model is used.

(Refer Slide Time: 21:58)



EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks[7]

Scaling up a Baseline model with different network width (**w**), depth (**d**) and input resolution (**r**). Bigged networks with larger width, height and input resolution perform better but accuracy gain saturates.

- Conventional wisdom suggests that scaling up CNN architectures would lead to better accuracy i.e deeper and wider networks perform better in general
- Explores a principled way to scale up a CNN to obtain better accuracy and efficiency

[7]Tan and Le, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, ICML 2019

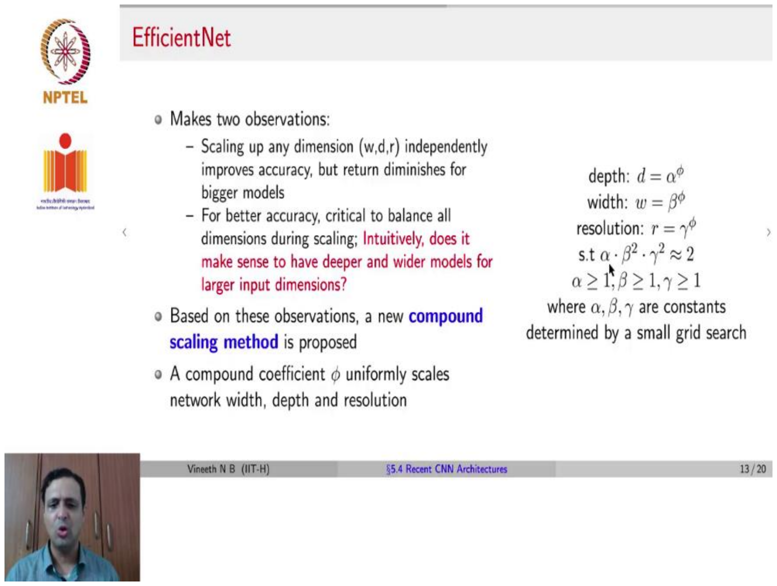
Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 12 / 20

A similar approach was proposed last year in 2019 known as EfficientNet this had a defined different premise to the way it went about the same idea but the objective was similar which is to develop a family of models that cater to some constraints that you may have in terms of computational resources.

And the way they approach this problem is to say that we generally understand through conventional wisdom that if you scale up CNN architectures in terms of width, in terms of depth, in terms of input resolution the model should work better. But when they made when they ran studies to observe the trend of increasing depth, increasing width, and increasing resolution they found that it did increase but plateaued after a certain time.

And they tried this method to explore a principled way in which a CNN can be scaled based on resource constraints.

(Refer Slide Time: 23:05)

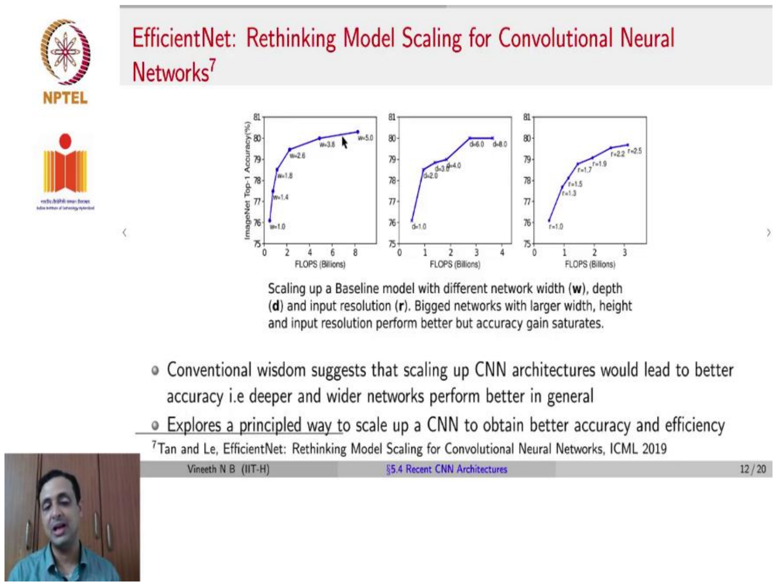


EfficientNet

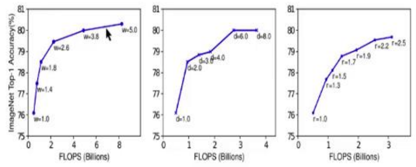
- Makes two observations:
  - Scaling up any dimension (w,d,r) independently improves accuracy, but return diminishes for bigger models
  - For better accuracy, critical to balance all dimensions during scaling; Intuitively, does it make sense to have deeper and wider models for larger input dimensions?
- Based on these observations, a new **compound scaling method** is proposed
- A compound coefficient ϕ uniformly scales network width, depth and resolution

depth: $d = \alpha^{\phi}$
width: $w = \beta^{\phi}$
resolution: $r = \gamma^{\phi}$
s.t $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$
$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$
where α, β, γ are constants determined by a small grid search

Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 13 / 20



EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks[7]



Scaling up a Baseline model with different network width (**w**), depth (**d**) and input resolution (**r**). Bigged networks with larger width, height and input resolution perform better but accuracy gain saturates.

- Conventional wisdom suggests that scaling up CNN architectures would lead to better accuracy i.e deeper and wider networks perform better in general
- Explores a principled way to scale up a CNN to obtain better accuracy and efficiency

[7]Tan and Le, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, ICML 2019

Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 12 / 20

As we just said, scaling up any dimension independently improves accuracy but return diminishes for bigger models which is what you see in these graphs. The first graph points out flops on the x-axis and the ImageNet accuracy on the y axis. You can see that as w the width of the model keeps increasing flops keep increasing but the accuracy also keeps increasing but at a certain point, the accuracy saturates even as you keep increasing the width of the model. You see a similar behavior for depth. You see a similar behavior even for the resolution of the input image. So the inference that they made was it may be critical to balance all of these dimensions while scaling up or scaling down a model.

It is not about only increasing width like WideResNets, it is not about only increasing depth like in ResNet itself but one should consider changing the width, changing the depth, and changing the input resolution carefully in a balanced manner to get maximum throughput with your resource constraints. So how do you go about doing this? They proposed a new scaling method known as a compound scaling method.

The idea was you could scale your architecture given a particular baseline architecture. You could scale the depth, width, and input resolution of the image through your current model design using a single scaling coefficient called φ and that is why it is known as compound scaling. It is based on one single coefficient but all your dimensions, your depth, your width, and your resolution all get changed based on that single coefficient.

Through a lot of empirical studies, they came up with this specific formulation that you see on the right which says that $depth: d = \alpha^{\varphi}$; $width: w = \beta^{\varphi}$; $resolution: r = \gamma^{\varphi}\ s.t.\ \alpha * \beta^2 * \gamma^2$ should be about 2 and each of α, β, γ must be greater than or equal to 1. And they are all constants.

You may wonder how this came. Firstly let us try to understand each of these why α into β square into γ square and not α square into β square and γ square for instance. The reason for this is that convolutions are the costliest operation in a CNN and when you increase depth convolution or the number of operations increases linearly but when you increase the width of the resolution, width is the number of filters or the resolution.

The computations increase in a squared manner and that is the reason this relationship is in place and this was empirically found by them. Now, why should it be equal to two and one can explain this through an example. Let us assume φ was equal to one and in their context, this means that you are now provided with double your current resources. You had a certain computational resource so far, for example, you had a certain GPU and now today you bought a new one and your resources doubled.

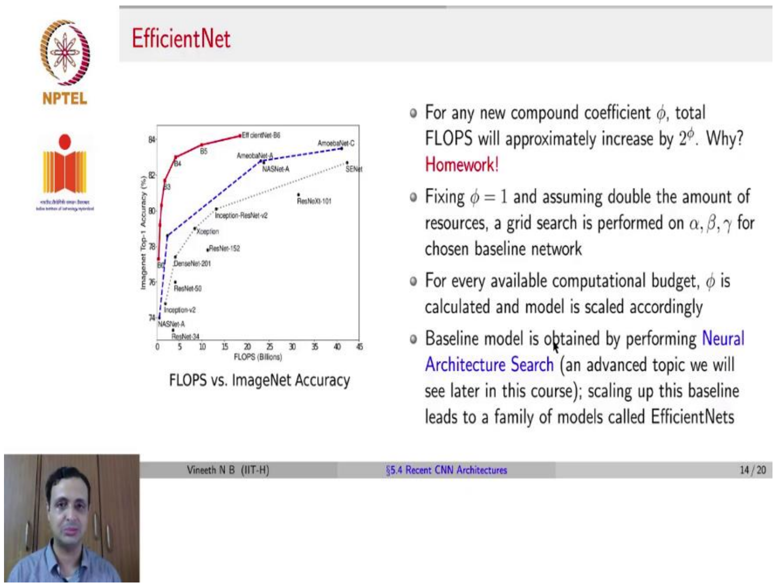
In that scenario, φ would be set to one in their framework and if φ is set to 1 you can make out that you would solve for $\alpha * \beta^2 * \gamma^2 = 2$ with α, β, γ greater than equal to 1. They solve this using a grid search and you would get values such as α to be 1.1, β to be 1.15, γ to be 1.3, or one of those variants that would give you the solution.

Now, when you do that you will notice that D would become 1.1 times the old D, w will become 1.2 times the old w and r would become say 1.3 times the old γ. So now when you multiply all these d w r, you would have d w r will be α into d into β into old w into γ old r.

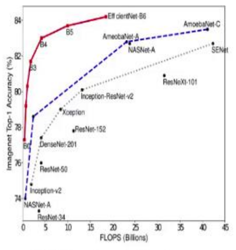
Now, because d w and r were old ones, the new factor which gets added will be $\alpha * \beta^2 * \gamma^2$ because that is how things will increase now which will now be two. So when $\varphi = 1$ and you have doubled your resource availability, solving this problem, solving this equation here α into β square into γ square is equal to 2 allows you to scale your depth, width, and resolution in a manner in which you will maximum go up to two times your current number of computations.

That is the way you go about doing it and once you decide your α, β, γ for a given problem you then change φ and you can get a family of models.

(Refer Slide Time: 28:36)



EfficientNet



FLOPS vs. ImageNet Accuracy

- For any new compound coefficient ϕ, total FLOPS will approximately increase by 2^ϕ. Why? Homework!
- Fixing $\phi = 1$ and assuming double the amount of resources, a grid search is performed on α, β, γ for chosen baseline network
- For every available computational budget, ϕ is calculated and model is scaled accordingly
- Baseline model is obtained by performing Neural Architecture Search (an advanced topic we will see later in this course); scaling up this baseline leads to a family of models called EfficientNets

Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 14 / 20

One question here, for any compound efficient φ the total flops will approximately increase by 2^{φ}. Why is that so? I think I partially gave you the answer already if you go back and look at what I just said remember if $\alpha = 1$, we saw that D will become α times the old depth.

And remember these are multipliers, right? These are scaling coefficients. So when we say D is equal to α of α we mean if α is 1.2 then D will become 1.2 times the old depth, it is a scaling coefficient. So if the total number of computations is d into w into r your new number of computations after doing the scaling would be α into d if you get α=1.2, β=1.1, and γ=1.3.

Then d would be α times the old d, w will be β times β square times the old w, r will be γ square times the old r. Remember, for width and resolution things get increased in a squared manner. So now you know that it will be the new depth would be α into old d into the new total number of computations would be α into old d into β square into old w into γ square into old r which would effectively become two times your earlier computations.

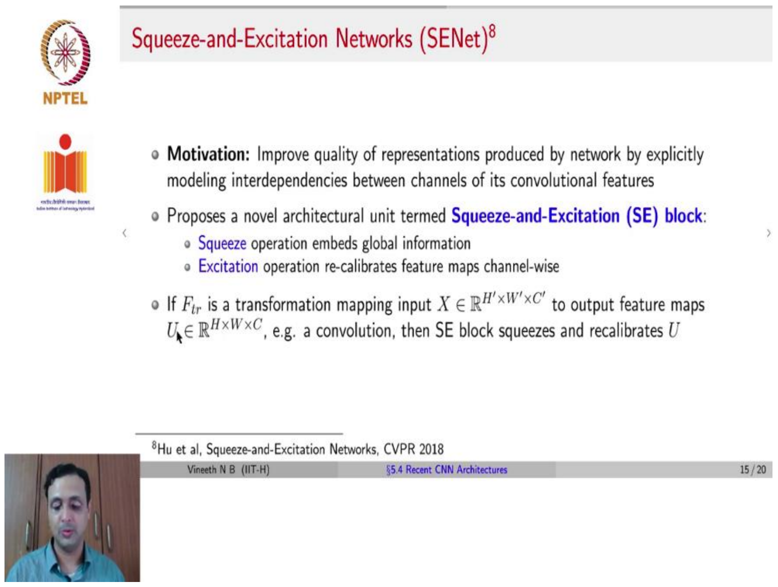
This was for a choice of φ to be one. If you chose φ to be two you can work this out in a similar manner and show that the number of computations will increase by two power φ. Using this approach you can fix first fix φ is equal to 1, you do a grid search on α, β, γ, solve this equation that you see here α into β square into γ square is equal to 2 and that would give you a certain value of α, β, γ.

Now for that choice of α, β, γ vary your φ based on whatever your computational budget is. As I said if your computational budget double set φ is equal to one and now if your computation budget is reduced accordingly change your φ and using this you will get a family of models where each model has scaled depth, scaled width, and scaled input resolution and with this they show that they can outperform MobileNets and achieve significantly high accuracy even under resource constraints.

The baseline model in this approach is obtained through a method known as neural architecture search. So the baseline model was not a ResNet it is not like they took a ResNet and then scaled the width, resolution, and depth. It is not like they took an AlexNet. The baseline model which is then scaled based on different constraints is found using a method called neural architecture search.

This is a topic that we will cover much later in this course since it is an advanced topic and needs a few more concepts before talking about it.

(Refer Slide Time: 31:57)



Squeeze-and-Excitation Networks (SENet)[8]

- **Motivation:** Improve quality of representations produced by network by explicitly modeling interdependencies between channels of its convolutional features
- Proposes a novel architectural unit termed **Squeeze-and-Excitation (SE) block**:
  - Squeeze operation embeds global information
  - Excitation operation re-calibrates feature maps channel-wise
- If F_{tr} is a transformation mapping input $X \in \mathbb{R}^{H' \times W' \times C'}$ to output feature maps $U \in \mathbb{R}^{H \times W \times C}$, e.g. a convolution, then SE block squeezes and recalibrates U

[8]Hu et al, Squeeze-and-Excitation Networks, CVPR 2018

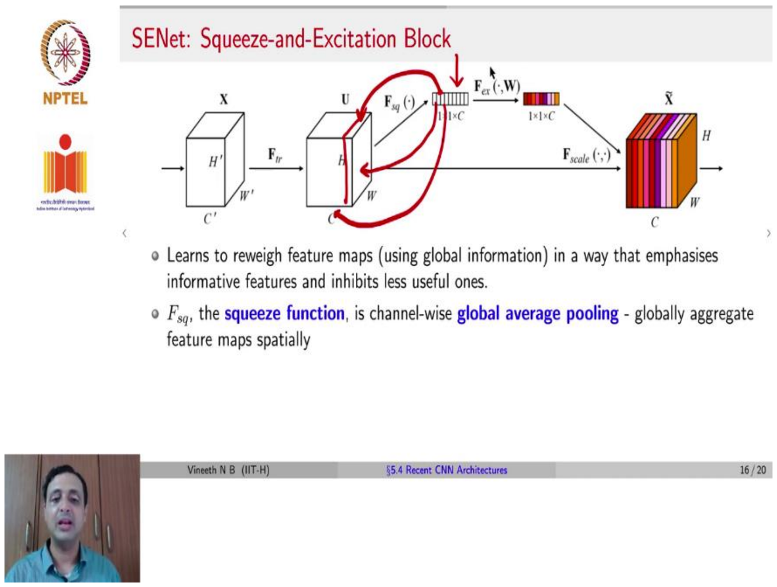
Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 15 / 20

The last architecture that we will talk about in this lecture is again a fairly recent one that was introduced in 2018 and this is known as the squeeze and excitation network. It was once again a network that helped improve the efficiency of the model by reducing computations in a certain way. Let us see how they went about it. The main motivation of this approach was to improve the quality of the representations which means representations of a CNN are the outputs or the certain feature maps of certain layers in the CNN. They wanted to improve those feature maps by modeling interdependencies between channels in each convolutional layer.

This was achieved by the introduction of a new architectural unit known as a squeeze and excitation block or an SE block which consisted of two operations, a squeeze operation that embeds global information and an excitation operation that recalibrates feature maps channel-wise. Let us see how this happens.

To understand it, let us consider an input x which has H'xW'xC' dimensions, an F_{tr} is a transformation that takes you from x to u which has dimensions HxWxC.

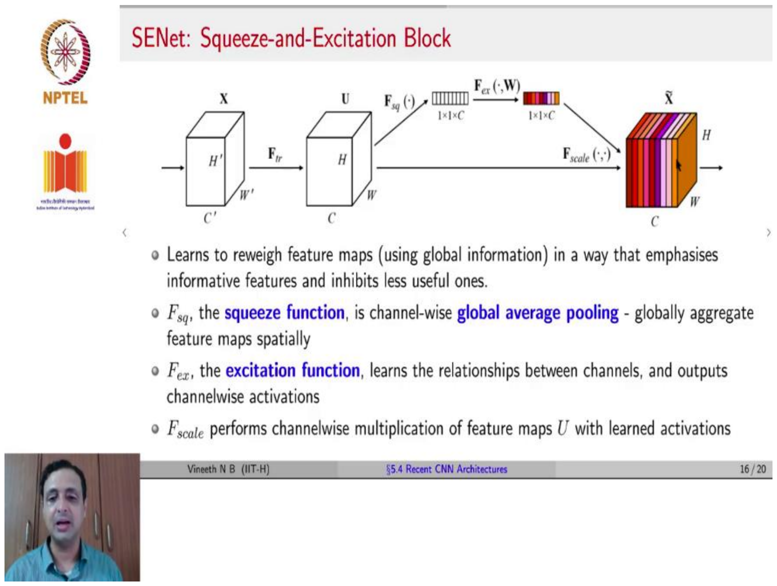
(Refer Slide Time: 33:35)



SENet: Squeeze-and-Excitation Block

- Learns to reweigh feature maps (using global information) in a way that emphasises informative features and inhibits less useful ones.
- F_{sq}, the **squeeze function**, is channel-wise **global average pooling** - globally aggregate feature maps spatially

Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 16 / 20



SENet: Squeeze-and-Excitation Block

- Learns to reweigh feature maps (using global information) in a way that emphasises informative features and inhibits less useful ones.
- F_{sq}, the **squeeze function**, is channel-wise **global average pooling** - globally aggregate feature maps spatially
- F_{ex}, the **excitation function**, learns the relationships between channels, and outputs channelwise activations
- F_{scale} performs channelwise multiplication of feature maps U with learned activations

Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 16 / 20

The SE block works on U and the way it works is what is shown in this visual here. So you have an x block it could be an input to a convolutional layer F transform could be F_{tr} which is denoted by F_{tr} is a convolutional layer that probably took x to u with the dimensions HxWxC.

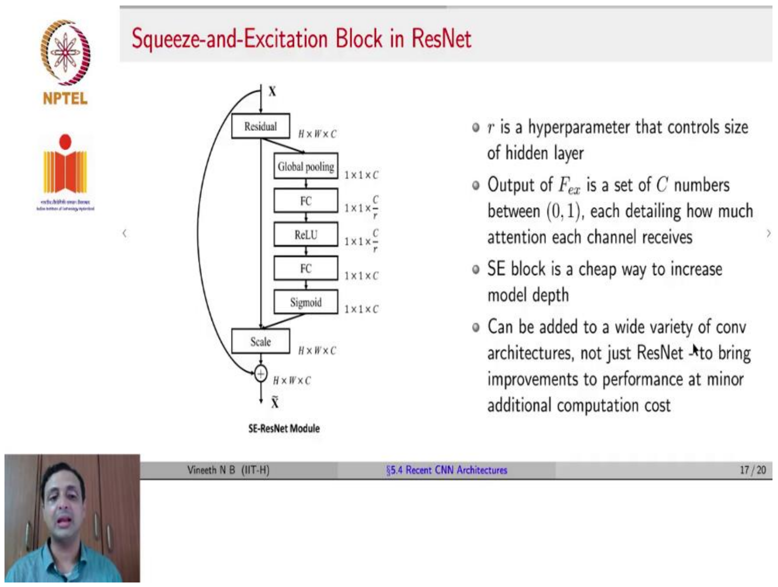
So the output of u is where the squeeze excitation block is squeezed in and what it does is to do two things as we just said. Firstly, you have a squeeze function which is denoted by F_{sq} in this visual here where you take each of these c channels in u and do global average pooling. Once again as we talked about this earlier in global average pooling you take the average of all the pixels in each of these channels.

So you have c channels here in u, for each channel you take the average that gives you by doing global average pooling you would get one scalar per channel. So this channel would give you one scalar, the next channel here let us assume that the next channel was something like this averaging all of those values will give you one scalar. Then similarly and for each of those channels here and that would form this 1x1x1 c vector, that is the squeezing part.

Then they have an excitation function which is denoted by F_x here which learns a set of weights, squeezing is only global average pooling. Remember pooling layers are parameter-free, there are no weights. They only squeeze they only sub-sample but then they have an excitation function here which is parameterized by some weights w which re-weights each of these scalars here by certain numbers based on weights.

What are we trying to do? We are trying to learn the relationships between the channels and the weight of each of these channels' outputs accordingly. These learned outputs of the excitation function are then used to multiply each of the channels in u by these values to get the new output $\tilde{X}$. You can see that $\tilde{X}$ has the same dimensions as HxWxC but now the values in each of these channels are scaled by the values that were learned through the excitation function.

(Refer Slide Time: 36:16)



Squeeze-and-Excitation Block in ResNet

- r is a hyperparameter that controls size of hidden layer
- Output of F_{ex} is a set of C numbers between $(0, 1)$, each detailing how much attention each channel receives
- SE block is a cheap way to increase model depth
- Can be added to a wide variety of conv architectures, not just ResNet to bring improvements to performance at minor additional computation cost

Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 17 / 20

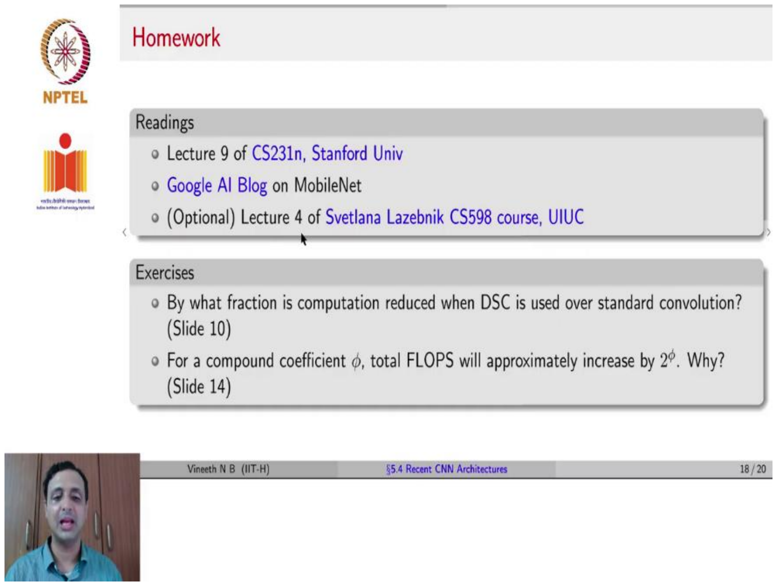
The SE block can also be placed inside a ResNet, the way you see in this visual. So you have a residual block and then you take it through a new SE block that is embedded inside the ResNet. In this case, the first step the output of a residual block is HxWxC. The first step is global average pooling and the size becomes 1x1xC. Now using a fully connected layer you reduce the size to 1x1xC by r.

Remember bottleneck layer you try to reduce the number of computations it is the same idea here where r is a hyperparameter that controls the size of that hidden layer. Then you perform a ReLU then you again increase the size back to c and then do a sigmoid and then send it back to scale and improve the original volume that you got in the residual block. So this is known as the SE ResNet where the squeeze and excitation block is embedded into the ResNet module.

Just to explain this the output of F_{ex} is the set of c numbers between 0 1, each detailing how much attention each channel receives. We talked about it as the scaling for each channel and this becomes a simple way to increase model depth artificially, to scale up each channel is one way to increase the effect of model depth and this could be added to a wide variety of convolutional architectures, not just ResNets.

But it could be added to many architectures and can help improve performance with very little added cost.

(Refer Slide Time: 38:06)



Homework

Readings

- Lecture 9 of CS231n, Stanford Univ
- Google AI Blog on MobileNet
- (Optional) Lecture 4 of Svetlana Lazebnik CS598 course, UIUC

Exercises

- By what fraction is computation reduced when DSC is used over standard convolution? (Slide 10)
- For a compound coefficient ϕ, total FLOPS will approximately increase by 2^{ϕ}. Why? (Slide 14)

Vineeth N B (IIT-H) §5.4 Recent CNN Architectures 18 / 20

To conclude, please follow the readings of lecture nine of CS231n. There is a very nice Google AI blog on the MobileNet worth reading and an optional lecture on Svetlana Lazebnik. Your exercises were by what fraction is computation reduced when DSC is used over standard convolution, depth-wise separable convolution which was on slide ten and for compound if coefficient φ.

In EfficientNets total flops, we said will approximately increase by two power φ y. We partially answered the questions in this lecture so we would not answer this the next time but these are a couple of things for you to think through at the end of this lecture so that you know you have understood what we discussed.