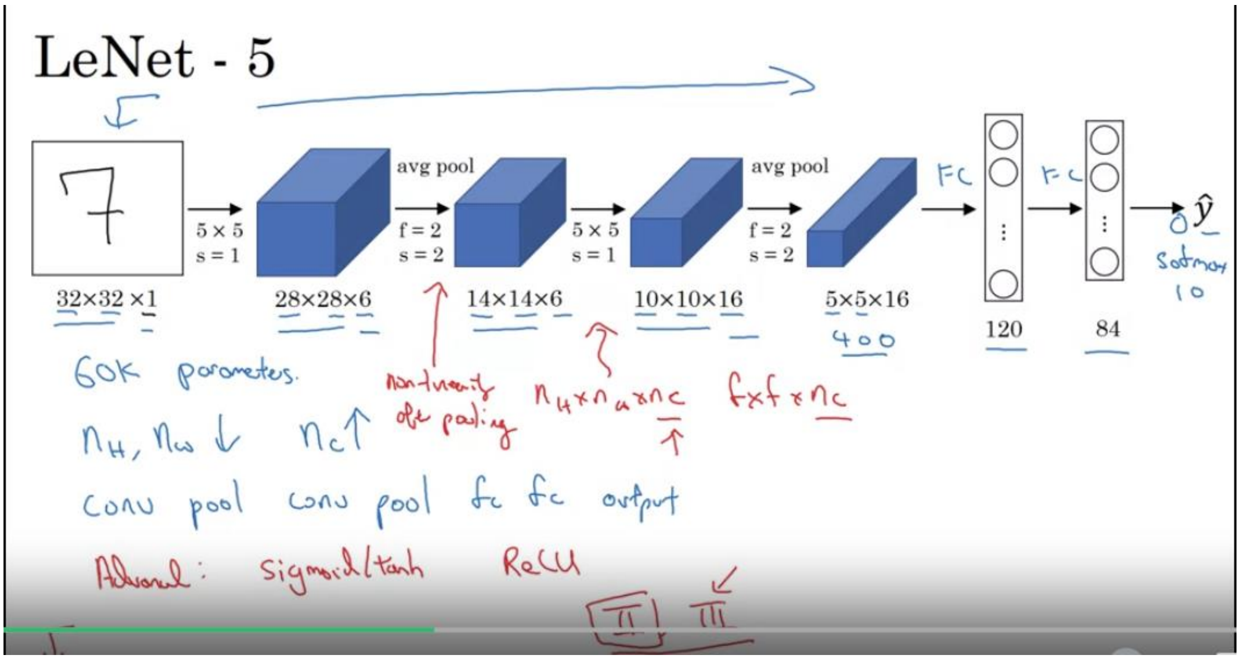
Deep Convolutional Models

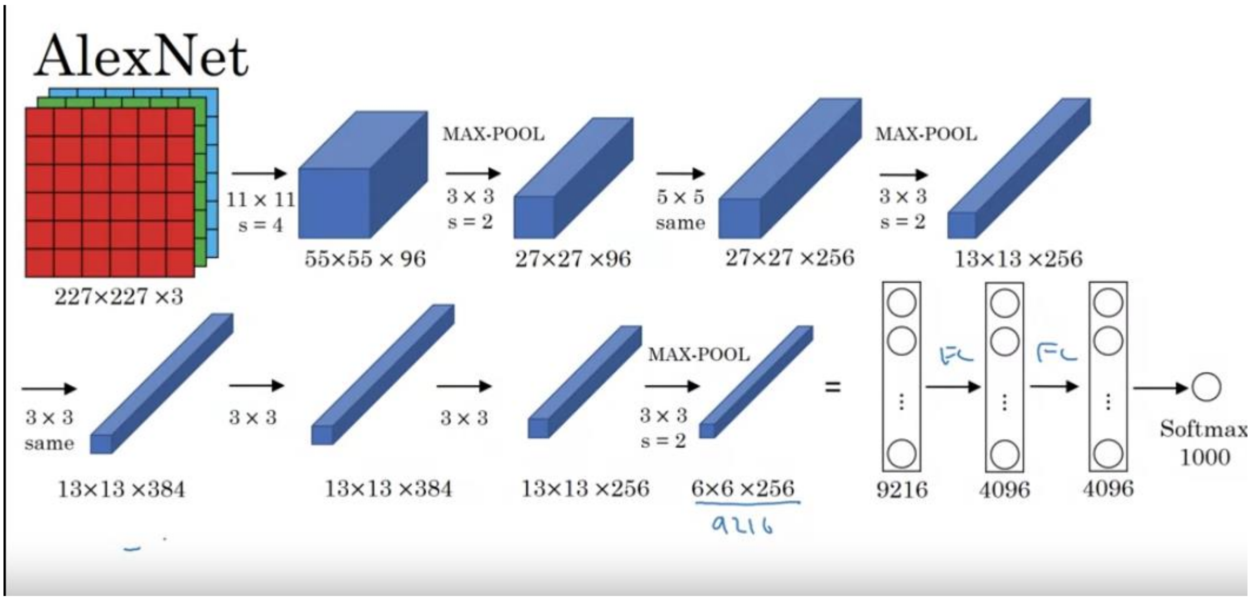
Classic Neural Networks:

1. LeNet – 5 Architecture:



LeNet-5 was trained on grayscale images, consists of convolutional layers followed by pooling layers and fully connected layers.

2. AlexNet:

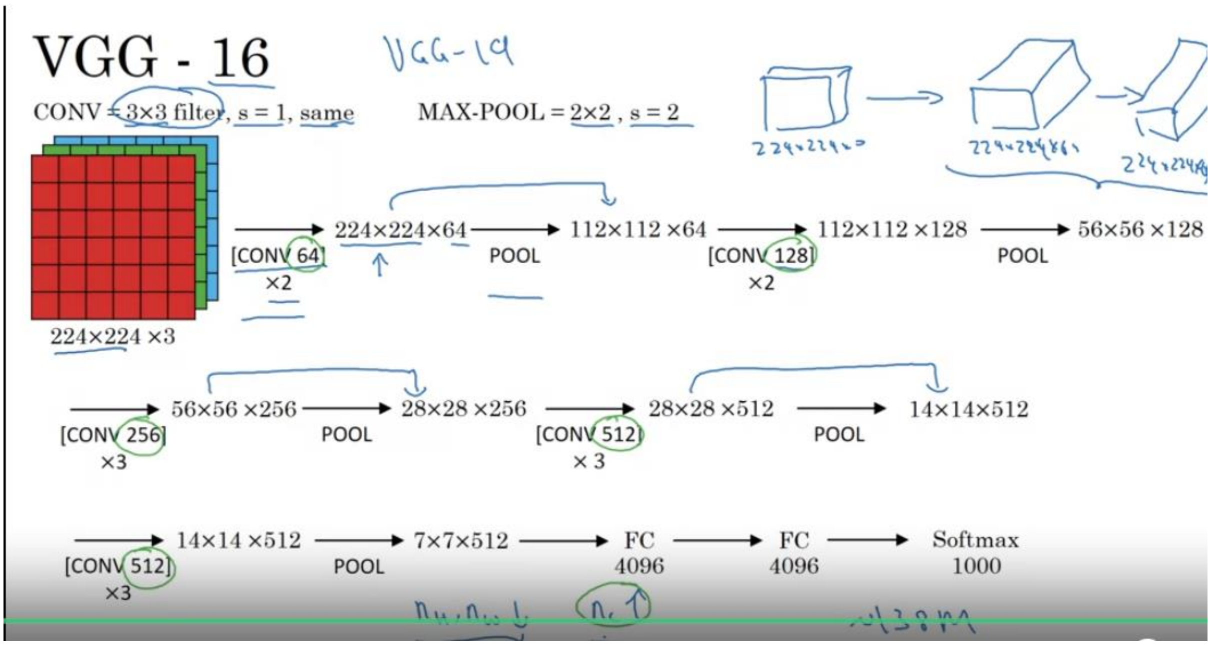


AlexNet, a larger network, also follows a similar pattern as of LeNet but with more layers and parameters.

LeNet have 60K parameters while AlexNet have about ~60M parameters.

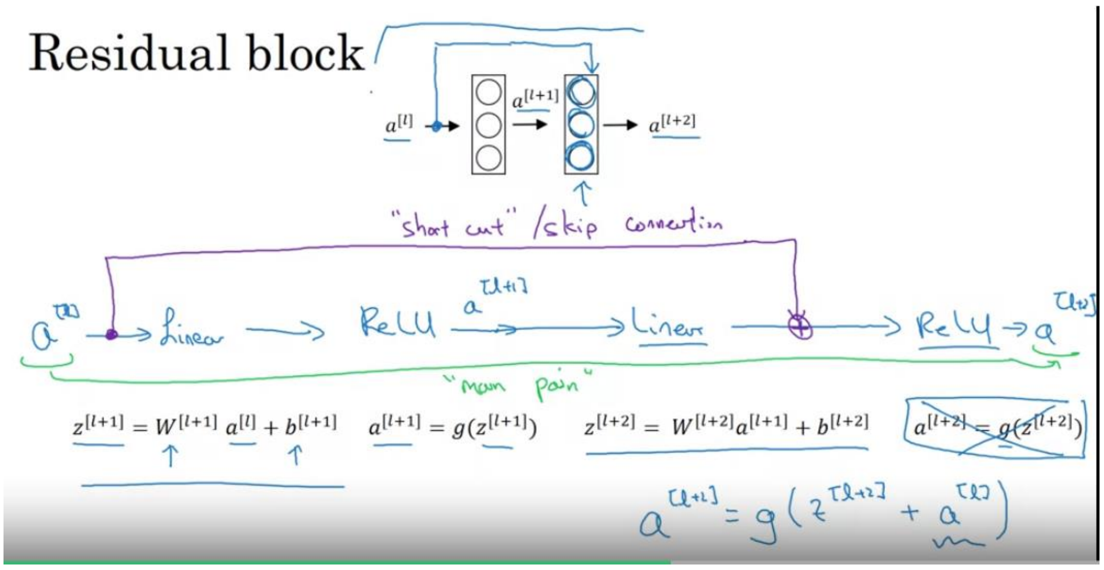
3. VGG – 16:

It is pretty large neural network as it contains ~138M parameters.



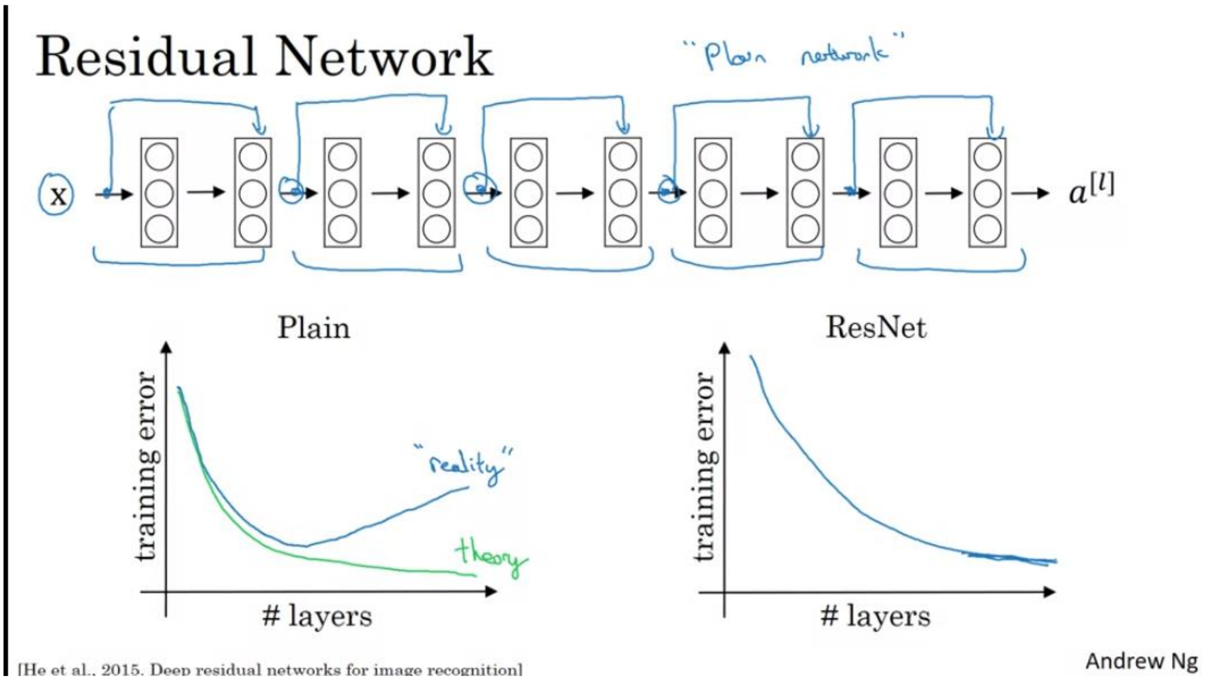
VGGNet simplifies the architecture by using only 3x3 filters and 2x2 pooling layers.

Residual Network:



Residual block
$a^{[l]}$
$a^{[l+1]}$
$a^{[l+2]}$
"short cut" / skip connection
$a^{[l]}$ → Linear → ReLU $a^{[l+1]}$ → Linear → ⊕ → ReLU → $a^{[l+2]}$
"main path"
$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]}$
$a^{[l+1]} = g(z^{[l+1]})$
$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$
$\cancel{a^{[l+2]} = g(z^{[l+2]})}$
$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$

- **Skip Connections:** Shortcuts that let information jump over some layers in the network, making it easier for the network to learn.
- **Residual Block:** A basic building unit in ResNet that includes a shortcut to help the network learn better by skipping certain layers.
- **Stacking Residual Blocks**: By putting many of these building units together, ResNet can be very deep—over 100 layers—without being too hard to train.



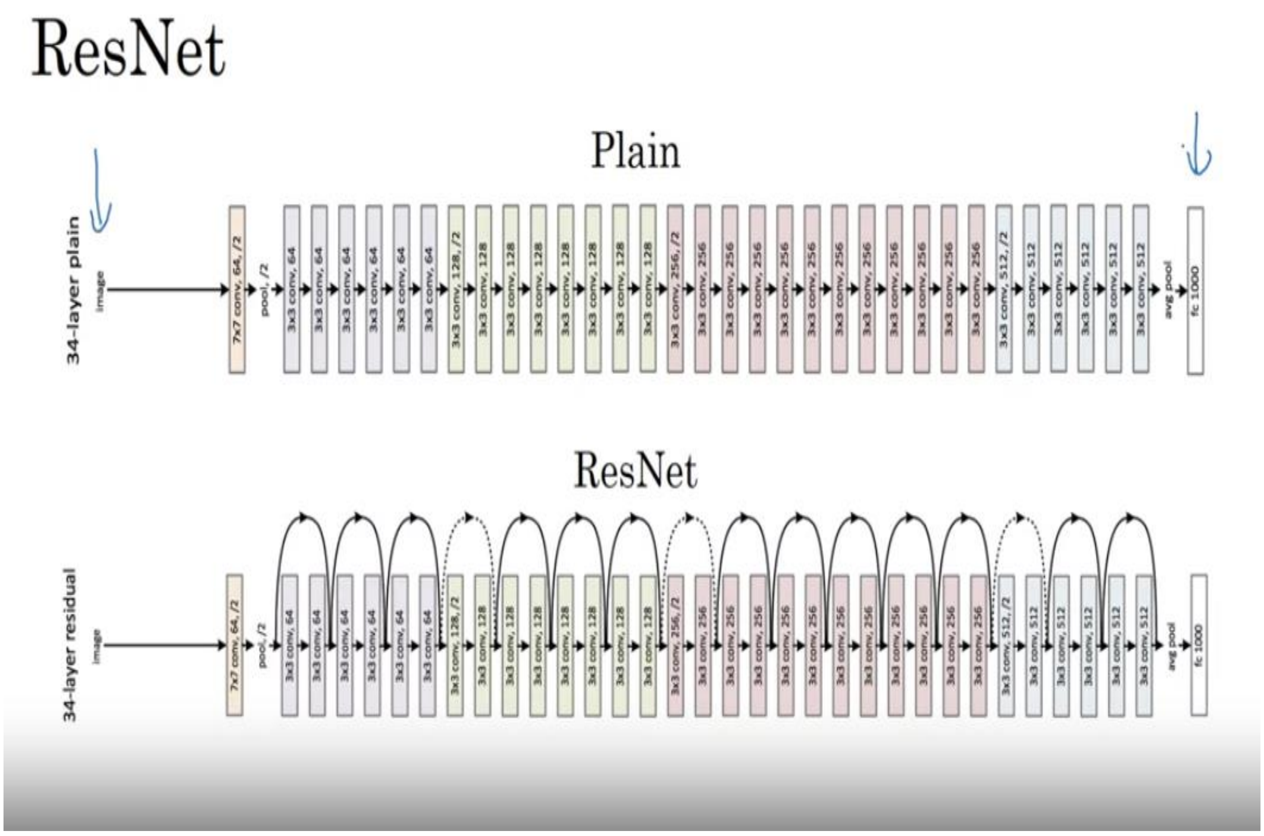
Why ResNet work?

The ResNet works well because they can be made deeper without negatively impacting their performance on the training set.

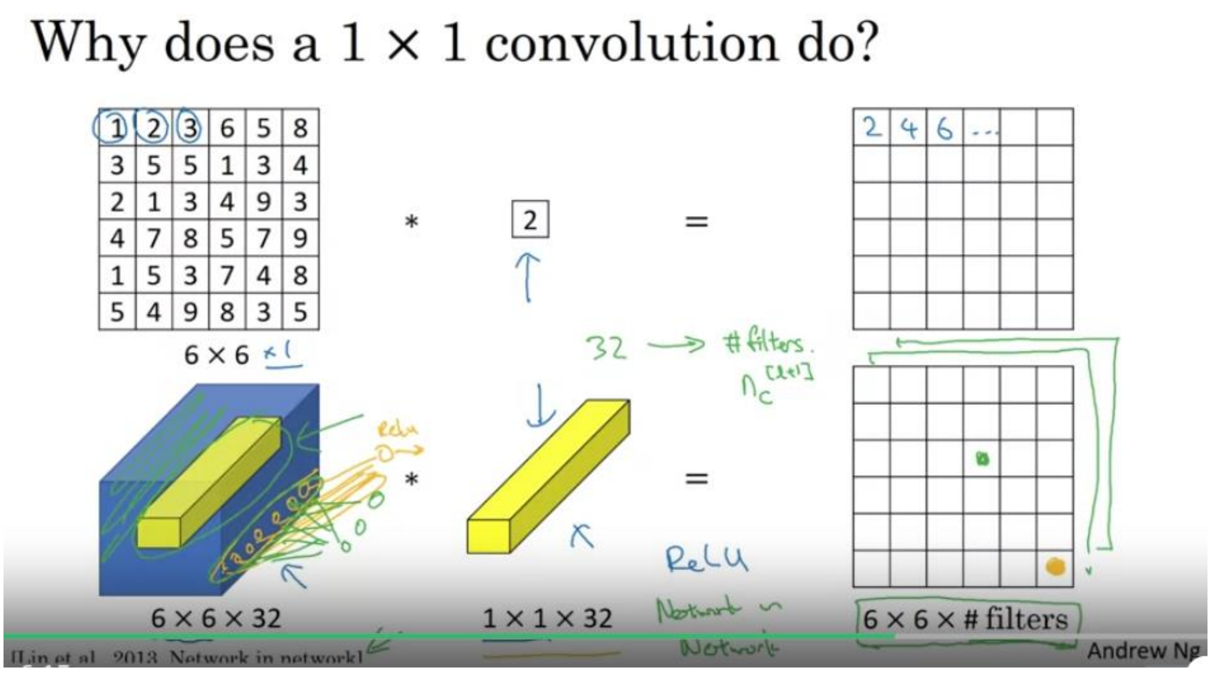
ResNet, learn the identity function and easily copy the activation from one layer to another.

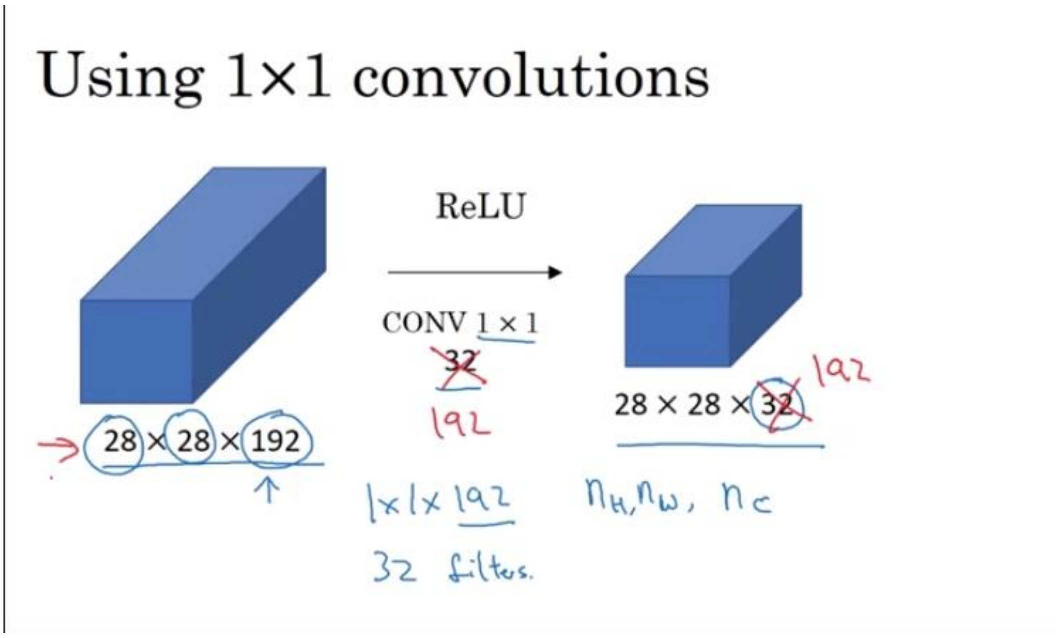
By adding skip connections, ResNets can preserve the identity function and make it easier to train deeper networks.

The use of same convolutions ensures that the dimensions of the input and output remain the same, making it possible to carry out the skip connections.



Networks in Networks and 1x1 Convolutions:





Lecture summary:

Definition of One-by-One Convolution:

Involves applying a one-by-one filter to an image or volume.

Initially appears as simple multiplication but has more complex applications with multiple channels.

Single-Channel vs. Multiple Channels:

With a single-channel image, a one-by-one convolution multiplies each element by the filter value.

With multiple channels, the one-by-one convolution performs element-wise multiplication for each channel and then applies a ReLU nonlinearity to output a single real number.

Use of Multiple Filters:

Multiple one-by-one filters can be used to create a more complex computation by combining inputs from different channels.

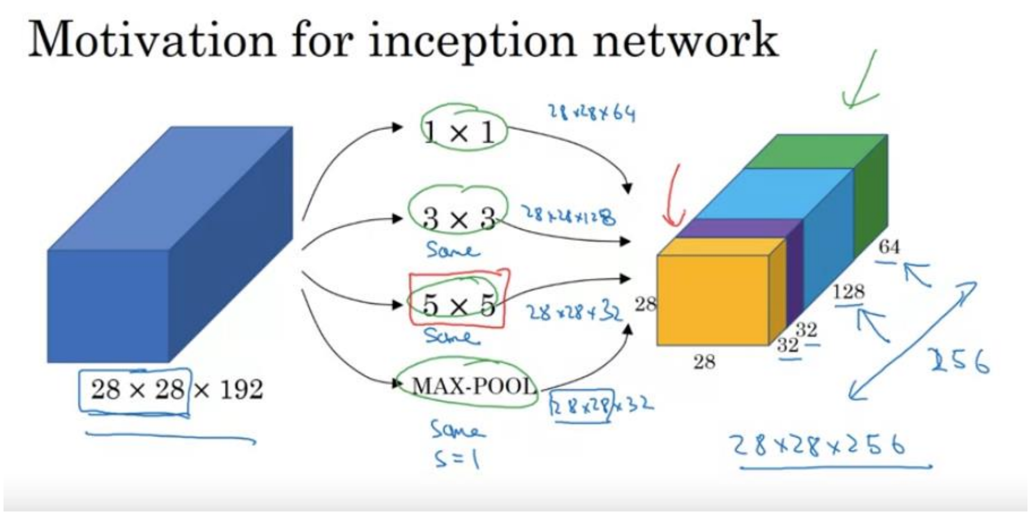
Channel Reduction:

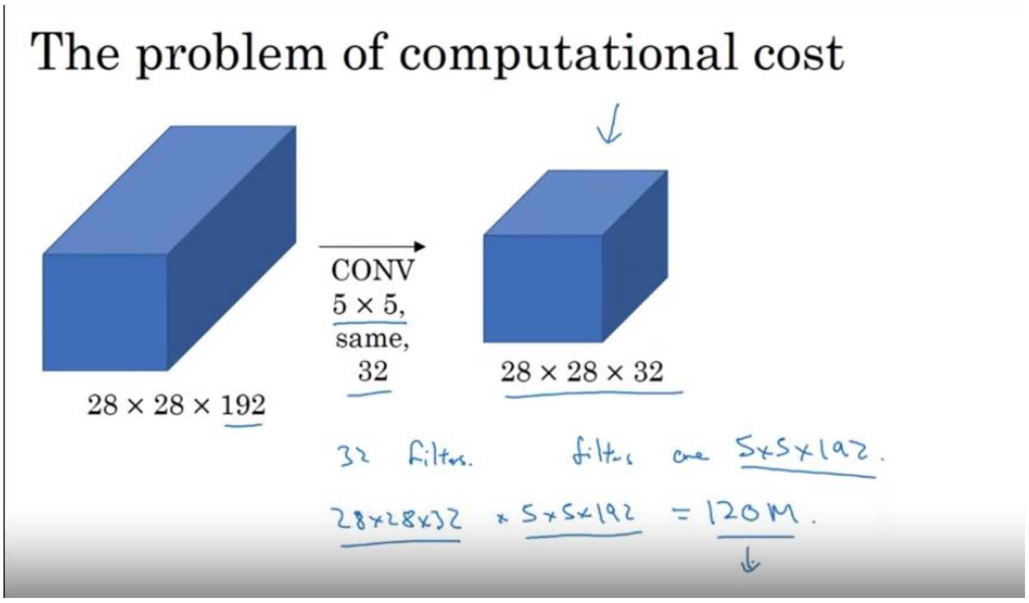
One-by-one convolutions can reduce the number of channels in a volume while keeping the height and width the same.

Inception Network:

Instead of choosing one filter size or pooling layer, the inception module uses multiple filter sizes and concatenates the outputs.

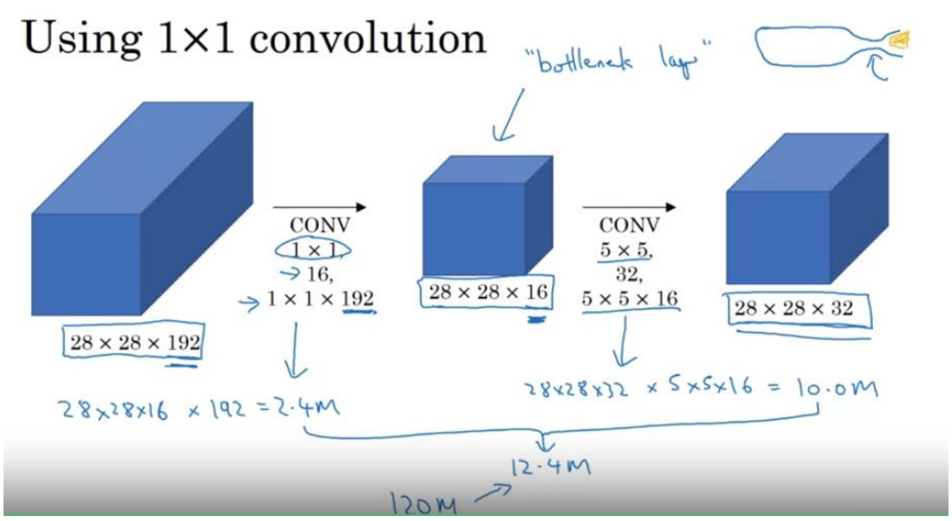
Andrew Ng: “ The basic idea is that instead of choosing one of these filter sizes or pooling you want and committing to that, you can do them all and just concatenate all the outputs, and let the network learn whatever parameters it wants to use, whatever the combinations of these filter sizes it wants.”



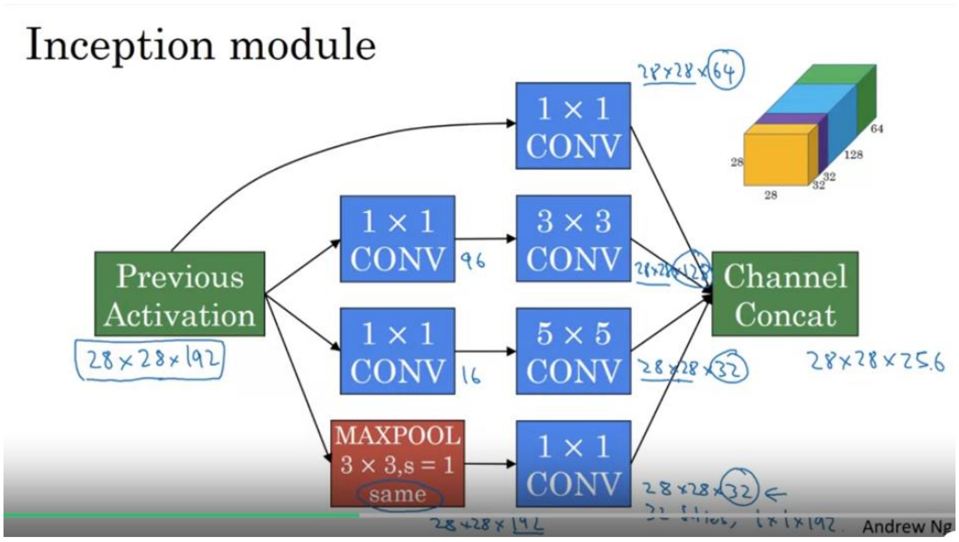


120 million multiplies on the modern computer is still a pretty expensive operation. By using 1x1 convolutions, the computational cost can be reduced to about a factor of 10. This bottleneck layer shrinks the representation size before increasing it again.

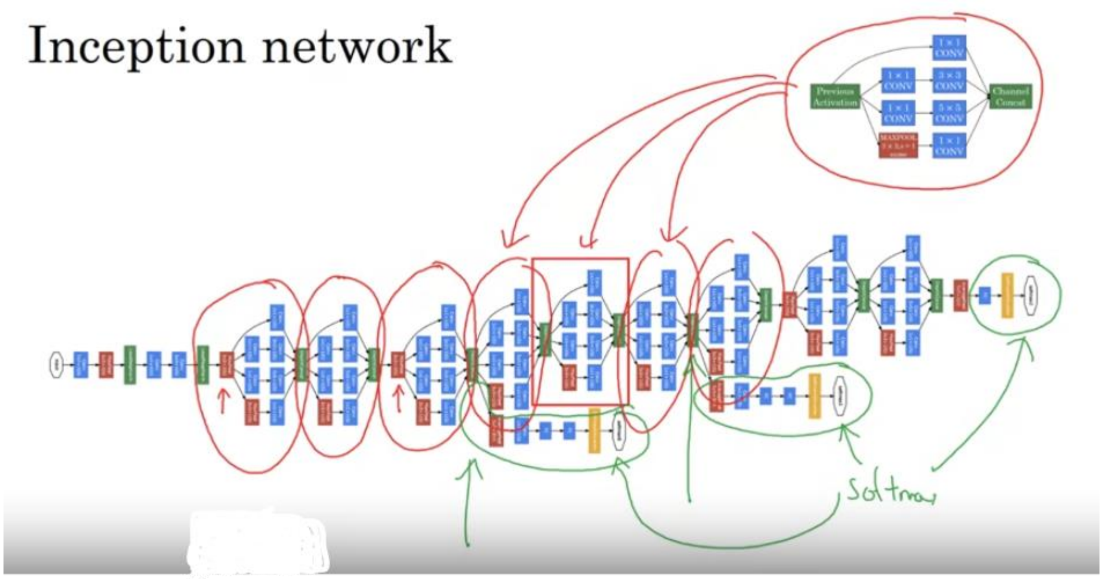
Using 1×1 convolution



Inception module



Inception Network:



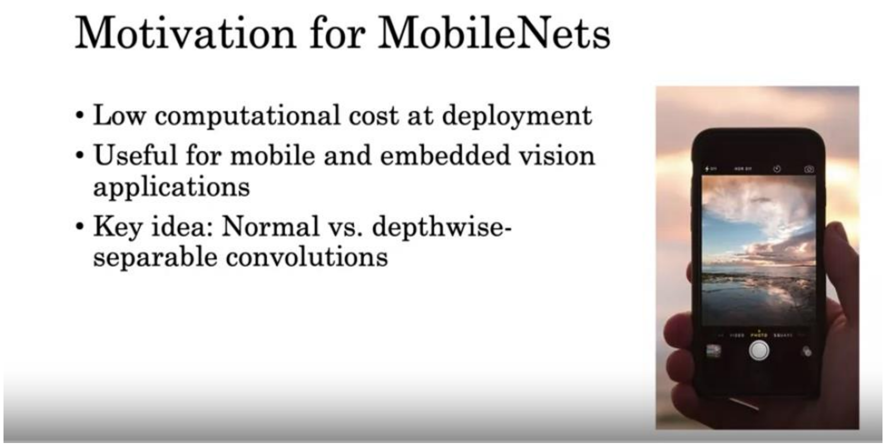
The Inception network was developed by authors at Google. Who called it GoogleNet.

The Inception module processes an input activation map (e.g., 28x28x192) using various convolutions: a 1x1 convolution followed by a 5x5 convolution, a 3x3 convolution, and additional 1x1 convolutions.

To reduce the number of channels from 192 to 32, a final 1x1 convolution with 32 filters is applied, which compresses the channel dimension while preserving spatial dimensions.

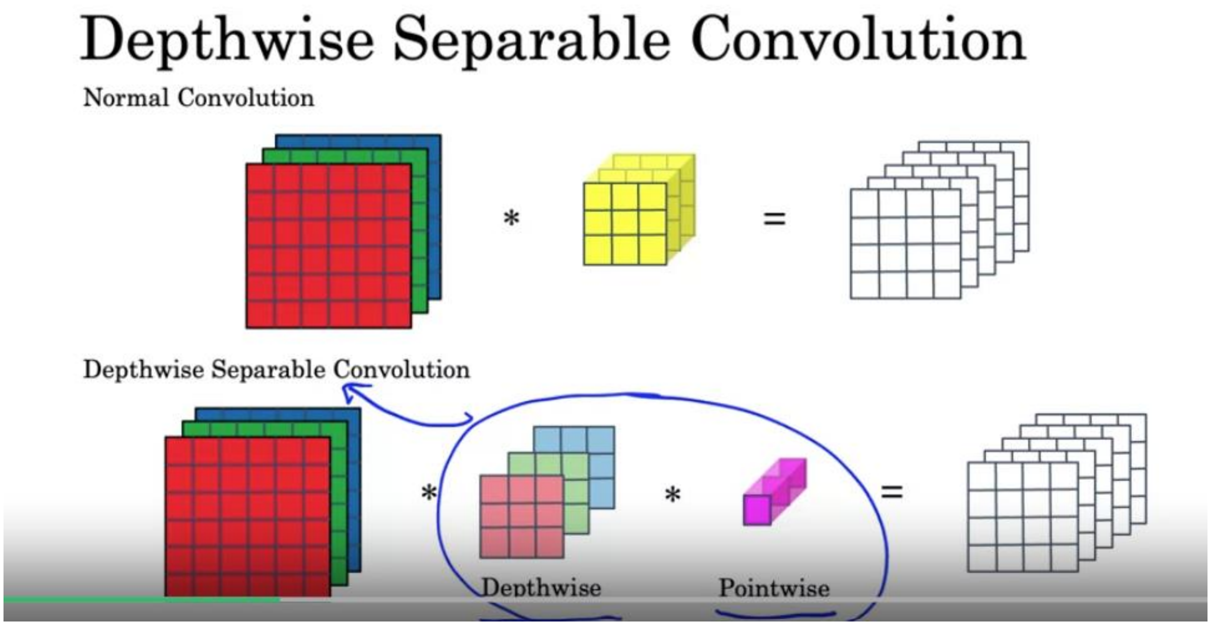
This step helps manage computational complexity and prevents excessive channel depth.

MobileNet Architecture:



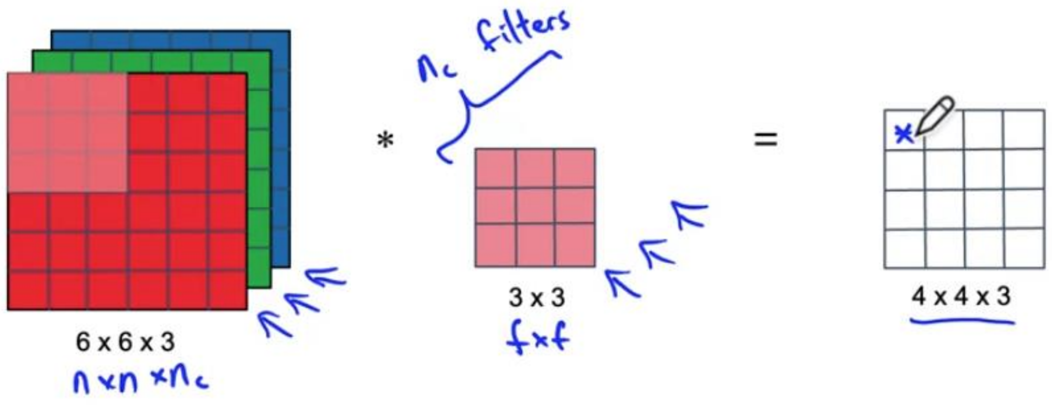
Motivation for MobileNets

- Low computational cost at deployment
- Useful for mobile and embedded vision applications
- Key idea: Normal vs. depthwise-separable convolutions

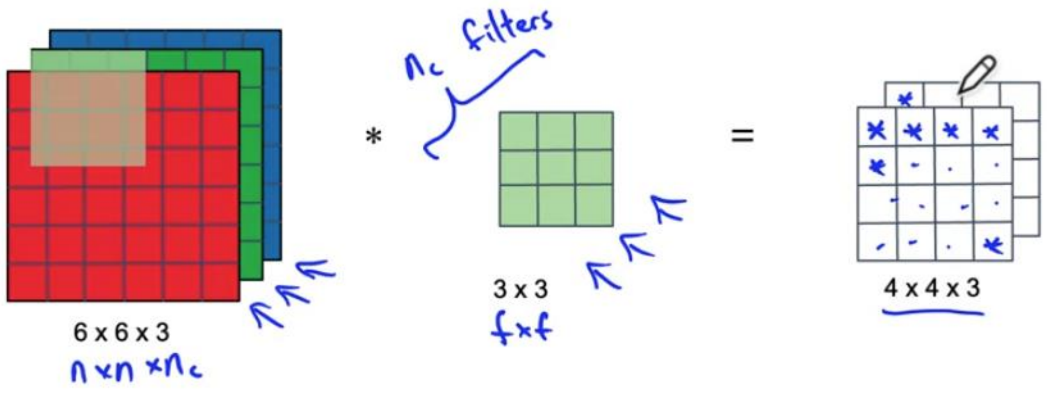


Depthwise Separable Convolution

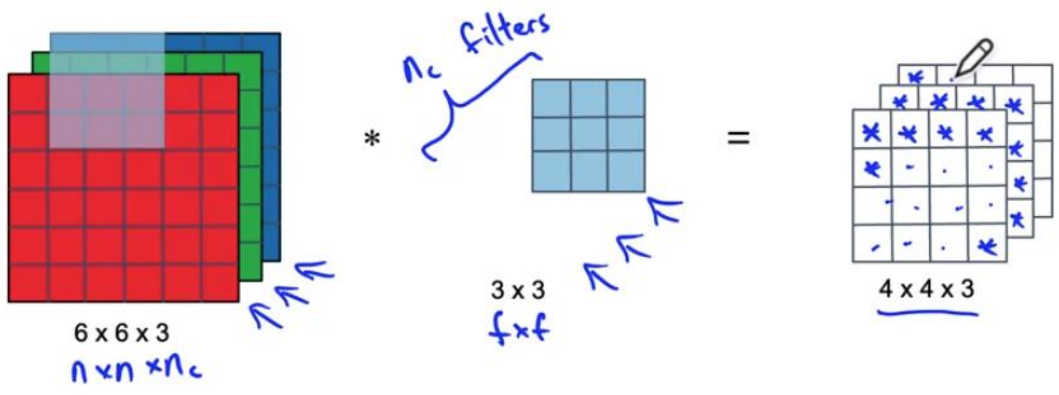
Depthwise Convolution

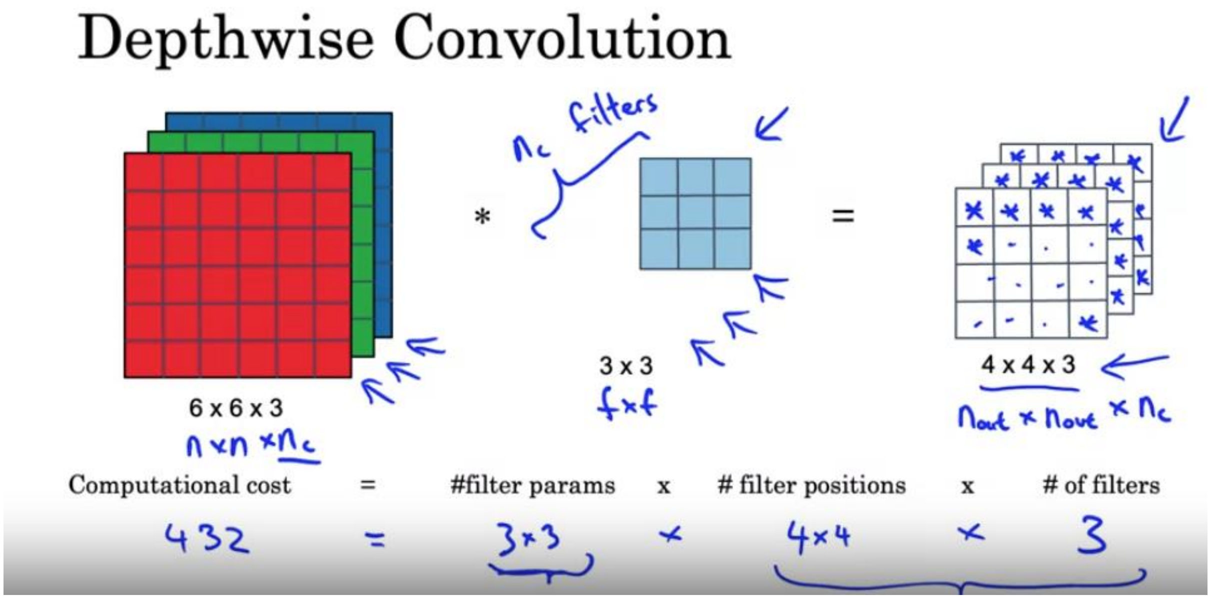


Depthwise Convolution



Depthwise Convolution

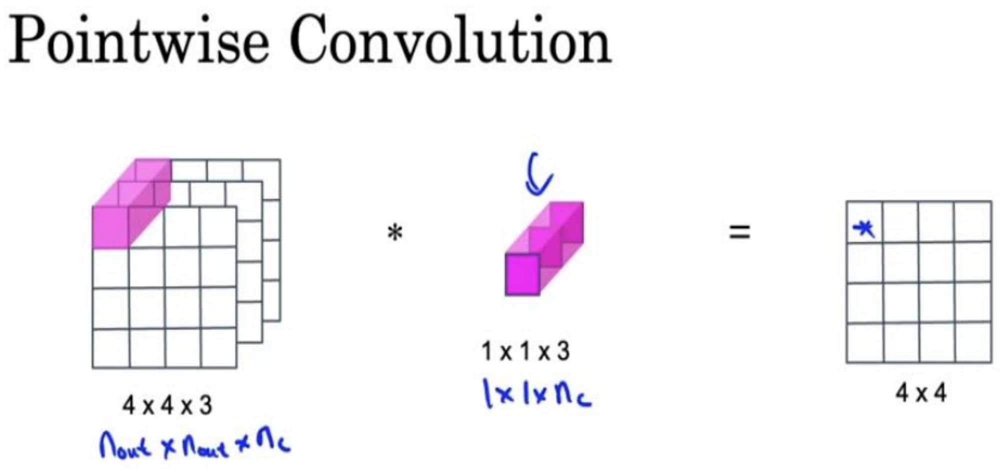




To generate each of the 4x4x3 output values, 9 multiplications (it is 3 x 3) are needed per value. The total computational cost is 3×3 (filter size) multiplied by the number of filter positions (4x4) and then by the number of filters (3) that is equal to 432.

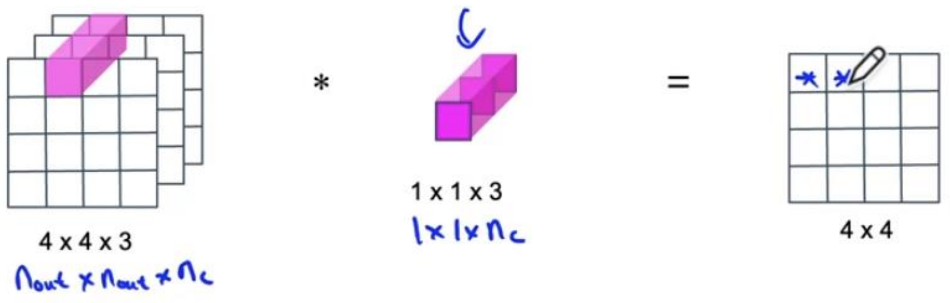
The final step is to apply a Pointwise Convolution to the 4x4x3 output, resulting in a 4x4x5 output.

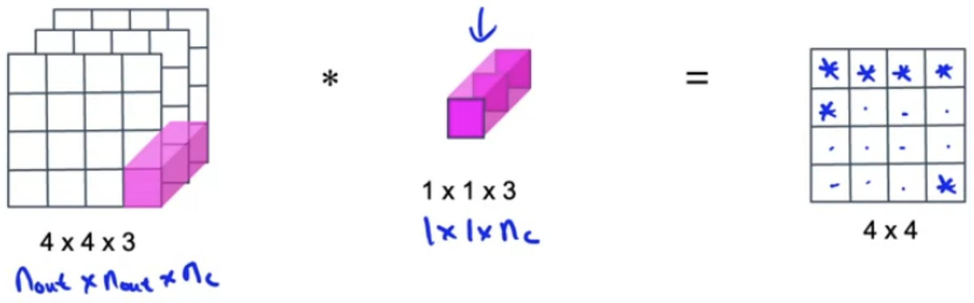
To take the intermediate set of values (n_out x n_out x n_c), we convolve it with a filter 1 x 1 x nc, which is 1 x 1 x 3 in this case. Here n_c stands for number of channels.



Shifted it by one each time:

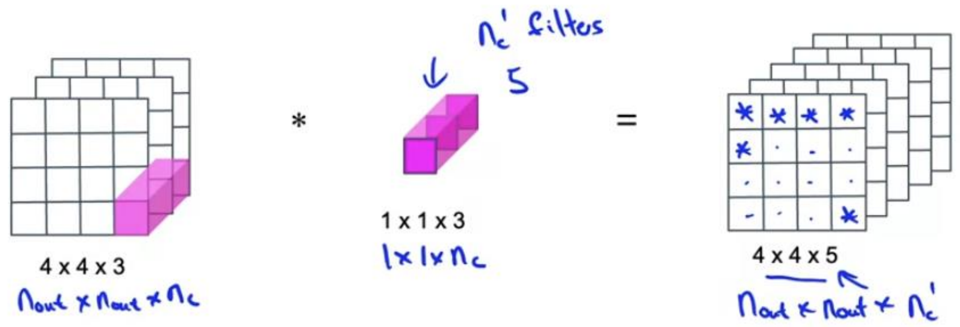
Pointwise Convolution





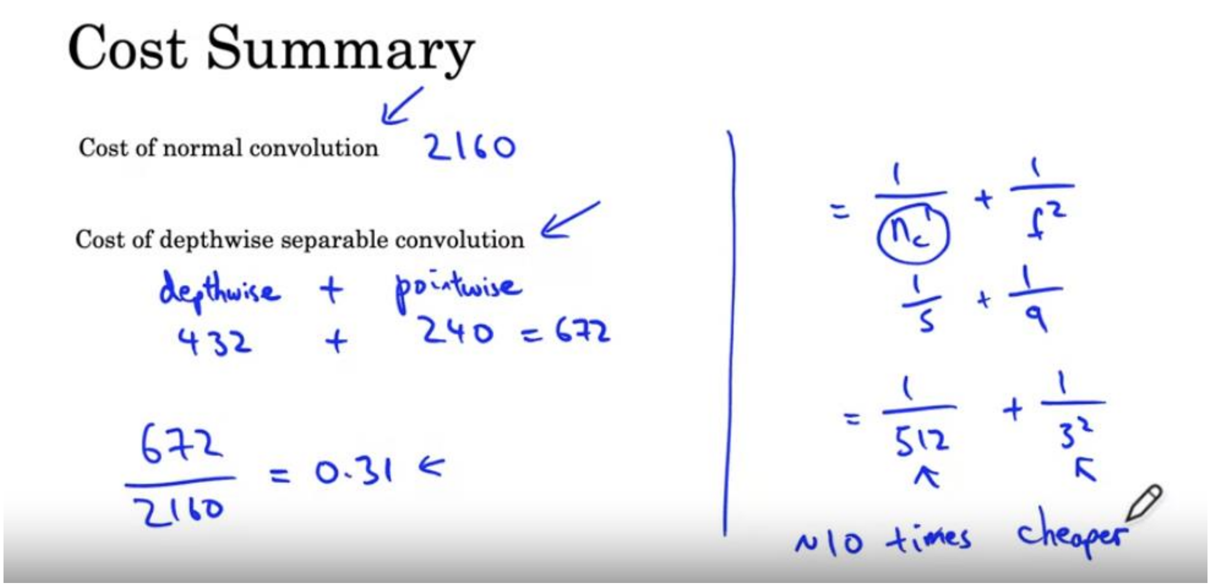
Here the number of filters we applied is 5. It can vary based on task complexity and output dimensions. The choice of nc' is determined through experimentation and optimization for performance and accuracy.

Pointwise Convolution


n_c' filters
5
*
=
4 x 4 x 3
$n_{out} \times n_{out} \times n_c$
1 x 1 x 3
$1 \times 1 \times n_c$
4 x 4 x 5
$n_{out} \times n_{out} \times n_c'$

Computational cost = #filter params x # filter positions x # of filters

$240 = 1 \times 1 \times 3 \times 4 \times 4 \times 5$



Cost Summary

Cost of normal convolution 2160

Cost of depthwise separable convolution

depthwise + pointwise

432 + 240 = 672

$$\frac{672}{2160} = 0.31$$

$$= \frac{1}{\boxed{n_c'}} + \frac{1}{f^2}$$

$$\frac{1}{5} + \frac{1}{9}$$

$$= \frac{1}{512} + \frac{1}{3^2}$$

~10 times cheaper

Description:

Computational Comparison:

Normal Convolution: 2160 multiplications.

Depthwise Separable Convolution:

- Depthwise step: 432 multiplications.
- Pointwise step: 240 multiplications.
- Total: 672 multiplications.

Ratio: 672 / 2160 ≈ 0.31.

NOTE: Depthwise separable convolution is about 31% as computationally expensive as normal convolution.

Cost Ratio of Depthwise Separable Convolution:

General Ratio: $1/nc' + 1/f^2$

In our example Calculation: nc' = 5 and f = 3

So the Ratio = $1 / 5 + 1 / 3^2 = 0.31$~

In a more typical neural network example, n_c prime will be much bigger.

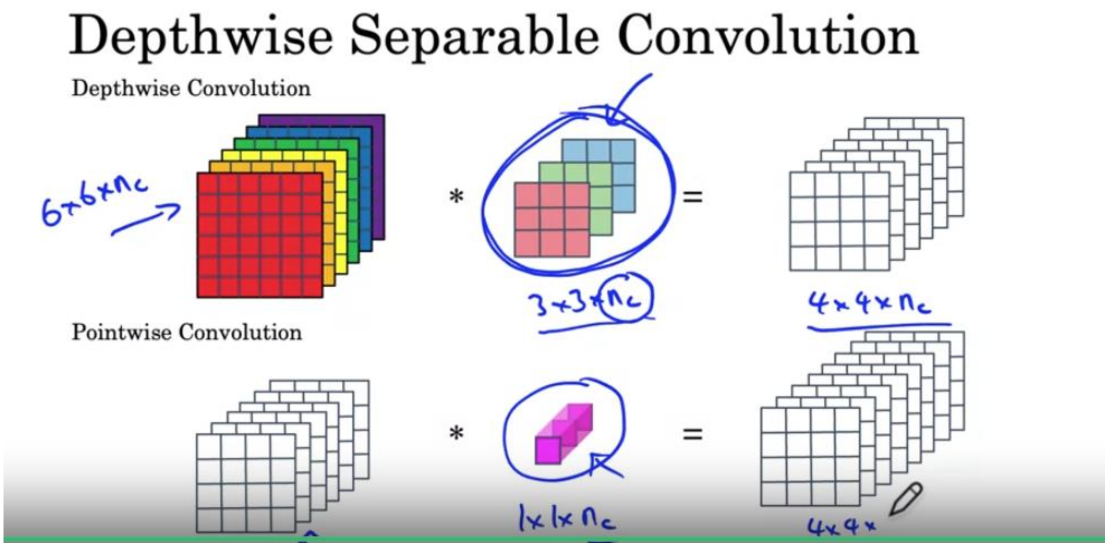
It may be, say, one over 512. If you have 512 channels in your output plus $1/ 3^2$ this would be a fairly typical parameters or new network.

Roughly, the depthwise separable convolution may be about one-ninth, rounding up roughly 10 times cheaper in computational costs.

That's why the depthwise separable convolution as a building block of a convnet, allows you to carry out inference much more efficiently than using a normal convolution.

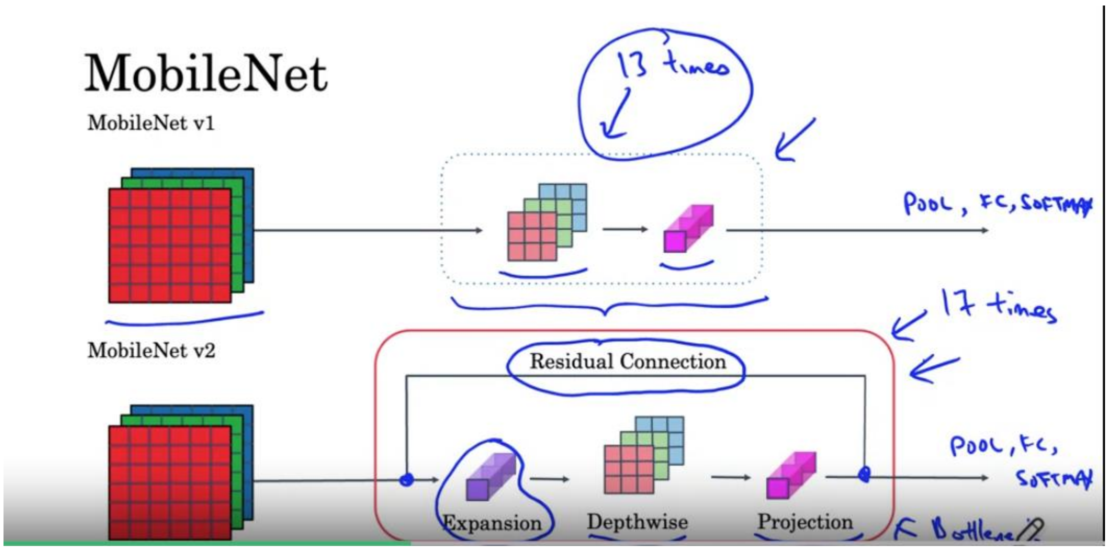
The depthwise separable convolution works for any number of input channels.

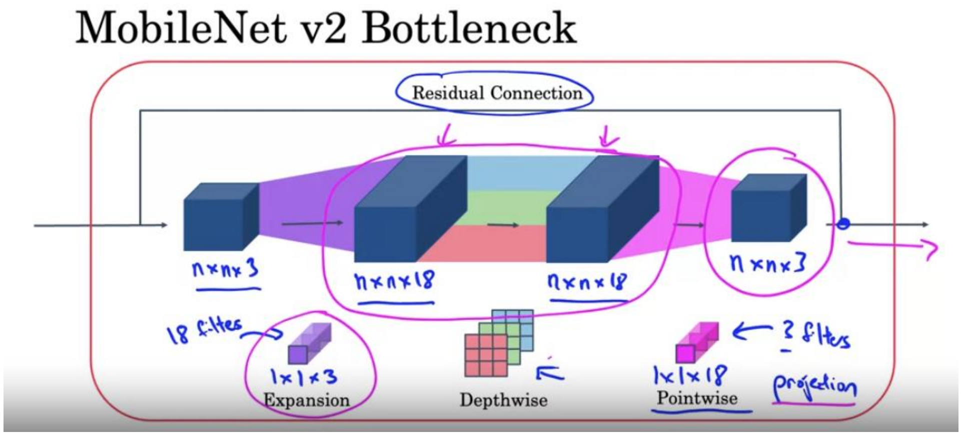
In the given example If there are 6 input channels,nc = 6, and use filter
3 x 3 x 6, Then the intermediate output will be 4x4x6 (height x width x channels).



After implementing 1 x 1 x n_c filter, we will get 4x4x8 as the final output.

MobileNet Architecture:





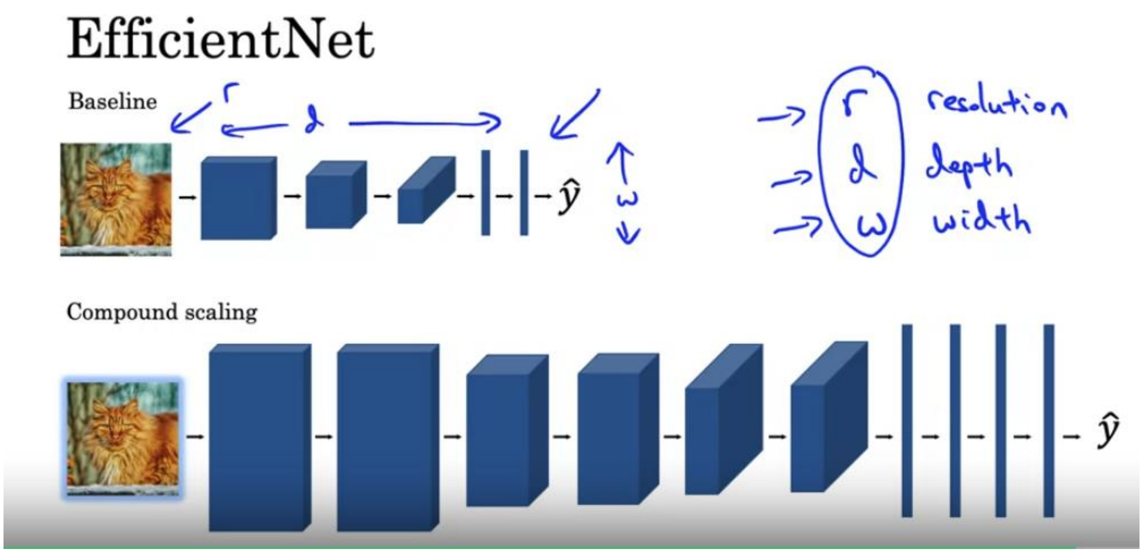
The MobileNet v2 architecture is an improvement on the MobileNet v1 architecture.

It has two main changes: the addition of a residual connection and an expansion layer before the depthwise convolution.

The residual connection allows backward propagation more efficiently, while the expansion layer increases the size of the representation within the bottleneck block.

The bottleneck block consists of an expansion operation, a depthwise separable convolution, and a pointwise convolution. The expansion operation increases the dimension of the input by a factor of six, and the depthwise separable convolution reduces the dimension back to the original size. The pointwise convolution then projects the dimension down to the desired output size.

EfficientNet Architecture:



Lecture Summery:

EfficientNet neural network allows to scale up or down the size of the network based on the computational resources of a device.

It provides a way to optimize the performance of the network within a given computational budget.

The three main factors that can be adjusted in EfficientNet to scale up or down the network are:

- **Image resolution:** You can choose a higher or lower resolution for the input image.
- **Depth of the network:** You can increase or decrease the number of layers in the neural network.
- **Width of the layers:** You can vary the width of the layers in the network.

By choosing the right combination of these factors, we can achieve the best possible performance for a specific device.

EfficientNet also offers compound scaling, where we can simultaneously scale up or down the resolution, depth, and width of the network.

This flexibility makes EfficientNet a valuable tool for building neural networks for mobile and embedded devices with limited computation and memory resources.