Why Case Studies? 見賢思齊

Classic Networks:

- LeNet-5
- AlexNet
- VGG-16

If I want to read the above classic networks, Andrew suggests the following order (from easy to hard)

AlexNet → VGG-16 → LeNet-5

ResNet

Inception Network

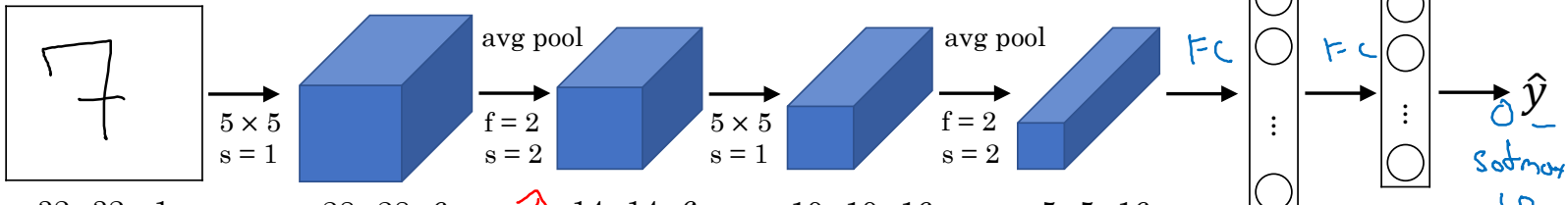
MobileNet

EfficientNet

These two networks help to build neural networks in embedded systems.

LeNet - 5

60k parameters



60K parameters.

$n_H, n_W \downarrow$ $n_C \uparrow$

non-linearity after pooling

$n_H \times n_W \times n_C$ $f \times f \times n_C$

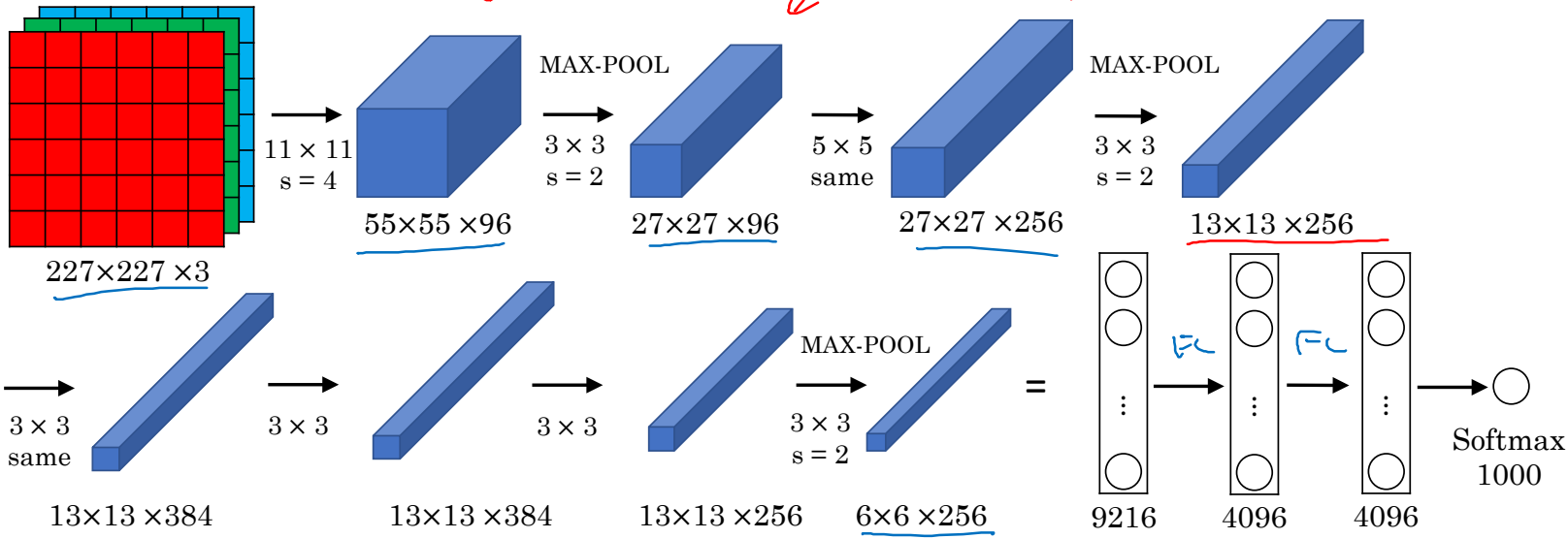
conv pool conv pool fc fc output

Activation: Sigmoid/tanh ReLU

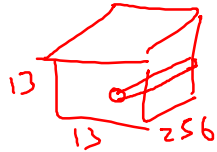
II III

[LeCun et al., 1998. Gradient-based learning applied to document recognition]

AlexNet 60m parameters



- Similar to LeNet, but much bigger. 60M parameters
- ReLU
- Multiple GPUs.
- Local Response Normalization (LRN)

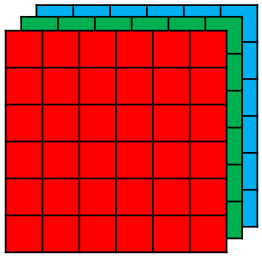


[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks]

VGG - 16

~~VGG-16~~ 138m parameters

CONV = 3×3 filter, s = 1, same MAX-POOL = 2×2 , s = 2



224×224 ×3

[CONV 64] ×2 → 224×224×64 → POOL → 112×112 ×64

[CONV 128] ×2 → 112×112 ×128 → POOL → 56×56 ×128

[CONV 256] ×3 → 56×56 ×256 → POOL → 28×28 ×256

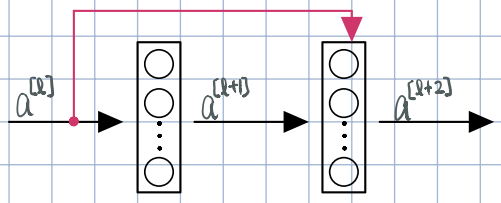
[CONV 512] ×3 → 28×28 ×512 → POOL → 14×14×512

[CONV 512] ×3 → 14×14 ×512 → POOL → 7×7×512 → FC 4096 → FC 4096 → Softmax 1000

$n_H, n_W \downarrow$ $n_c \uparrow$ ~138M

[Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition]

Residual Neural Network (ResNet)



Building Block

The path of neural network without shortcut:

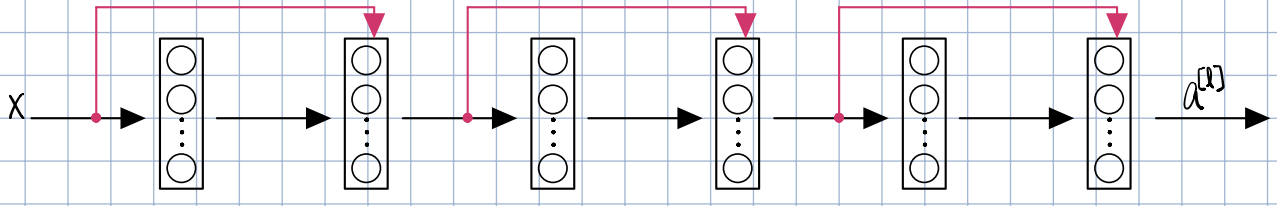
$a^{[l]} \rightarrow$ Linear $w^{[l+1]}a^{[l]} + b^{[l+1]}$ $\rightarrow z^{[l+1]} \rightarrow$ ReLU $g(z^{[l+1]}) \rightarrow a^{[l+1]} \rightarrow$ Linear $w^{[l+2]}a^{[l+1]} + b^{[l+2]}$ $\rightarrow z^{[l+2]} \rightarrow$ ReLU $g(z^{[l+2]}) \rightarrow a^{[l+2]}$

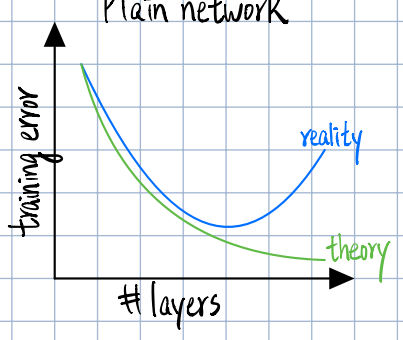
The path of neural network without shortcut:

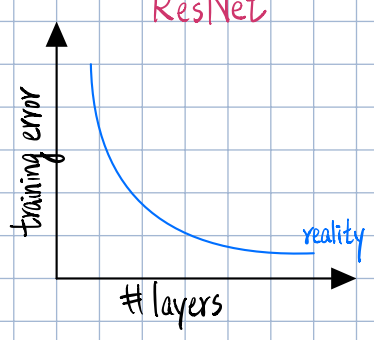
$a^{[l]} \rightarrow$ Linear $w^{[l+1]}a^{[l]} + b^{[l+1]}$ $\rightarrow z^{[l+1]} \rightarrow$ ReLU $g(z^{[l+1]}) \rightarrow a^{[l+1]} \rightarrow$ Linear $w^{[l+2]}a^{[l+1]} + b^{[l+2]}$ $\rightarrow z^{[l+2]} \rightarrow$ ReLU $g(z^{[l+2]} + a^{[l]}) \rightarrow a^{[l+2]}$

shortcut / skip connection

Turn a Plain network into a ResNet.



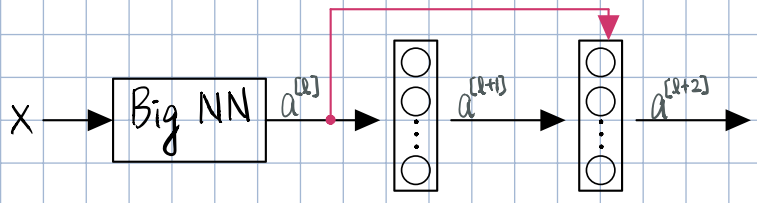




Why ResNet works? To gain intuition in ResNet.

Given a big neural network, $X \rightarrow$ Big NN $\rightarrow a^{[l]}$

I want to make it deeper,



$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$
$$= g(W^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]}) \quad \text{Assume } g \text{ is ReLU}$$

If $W^{[l+2]} = 0$ and $b^{[l+2]} = 0$, then $a^{[l+2]} = a^{[l]}$

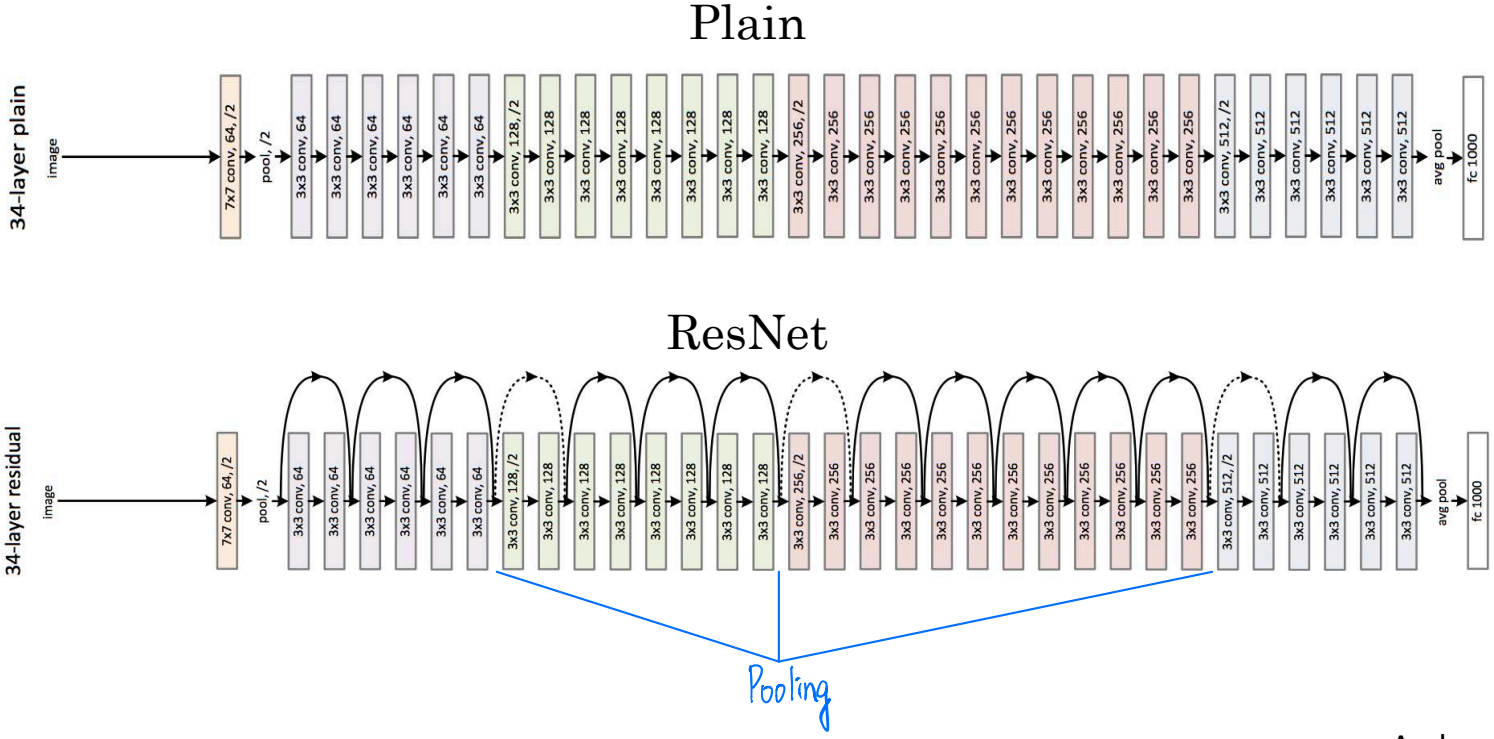
It shows that the identity function is easy for residual block to learn $a^{[l+2]} = a^{[l]}$. What it means is that adding the addition residual block in my neural network, it doesn't hurt my neural network's ability to do as well as the simpler network without these two extra layers.

Note: To make $g(z^{[l+2]} + a^{[l]})$ works, $z^{[l+2]}$ and $a^{[l]}$ need to have the same dimension.

In case of $z^{[l+2]}$ and $a^{[l]}$ having different dimensions, eg, $a^{[l]}$ is 128 and $a^{[l+2]}$ is 256, we can add an extra matrix before $a^{[l]}$. $\mathbb{R}^{256 \times 128}$ or just use zero-padding to $a^{[l]}$.

$$a^{[l+2]} = g(z^{[l+2]} + W_s a^{[l]})$$

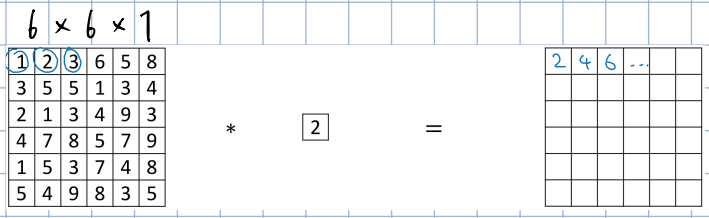
ResNet

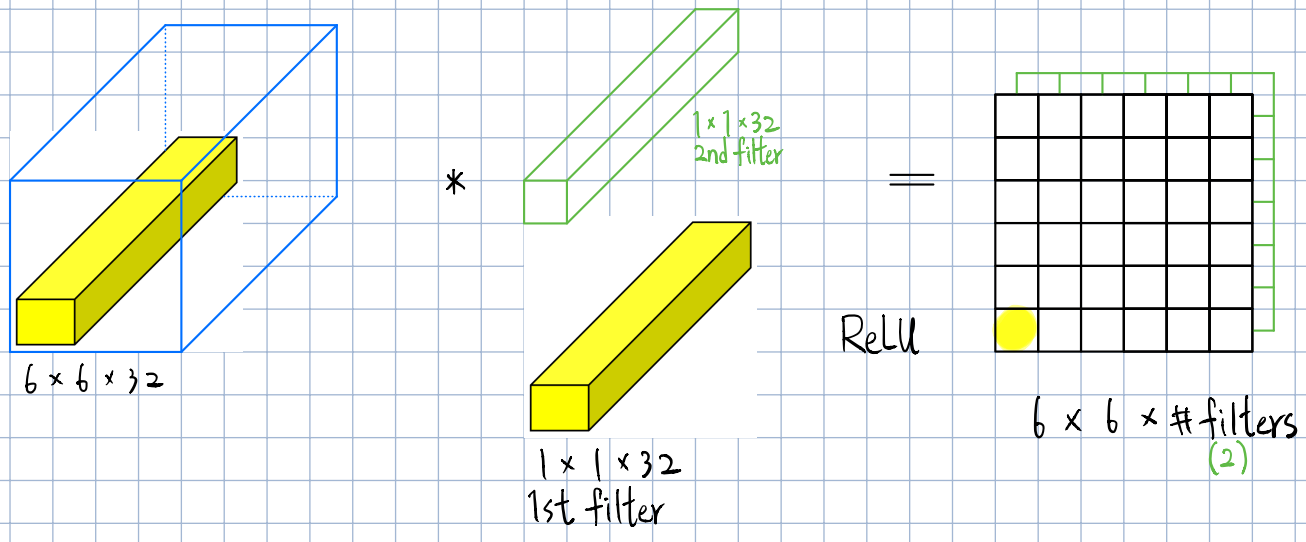


[He et al., 2015. Deep residual networks for image recognition]

Andrew Ng

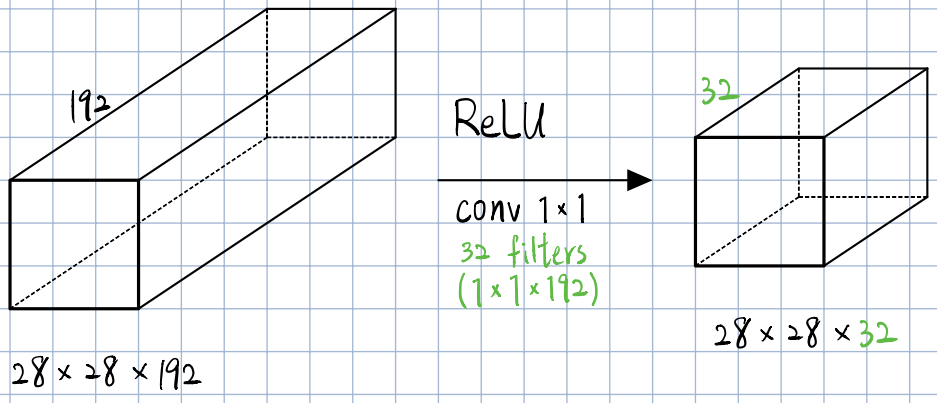
Networks in Networks and 1×1 Convolutions





Calculate the dot product of two 1×1×32 vectors, then apply ReLU on it.

Using 1×1 convolutions in an example.

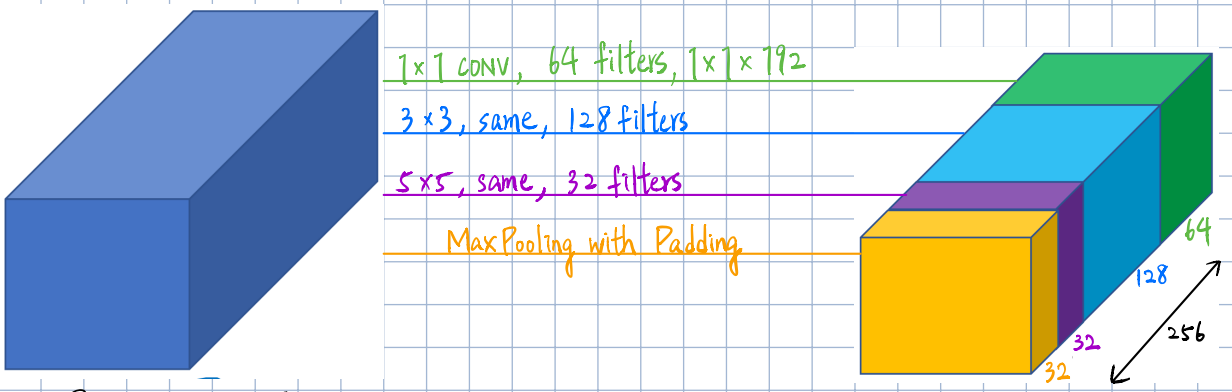


I know that the height and width can be shrinked by using pooling layer, what about # channels?

1×1 convolutions can shrink the number of channels.

Inception Network

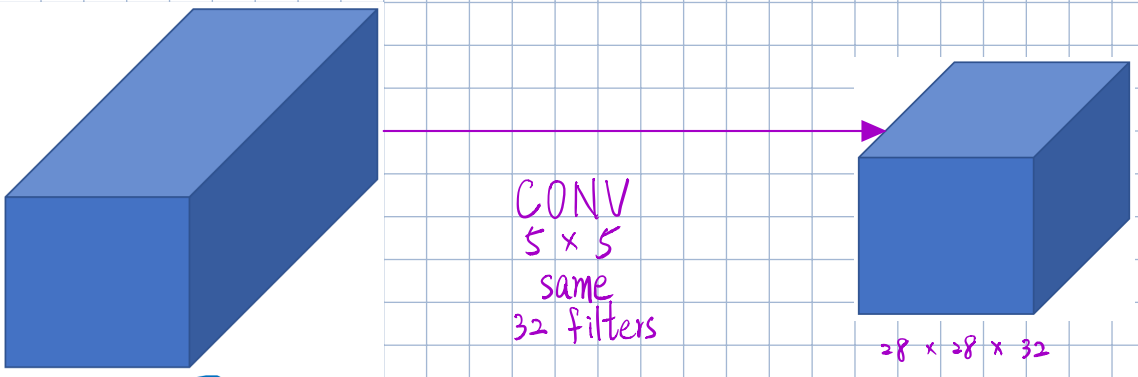
Motivation for Inception Network



28 × 28 × 192

28 × 28 × 256

The problem of computation cost



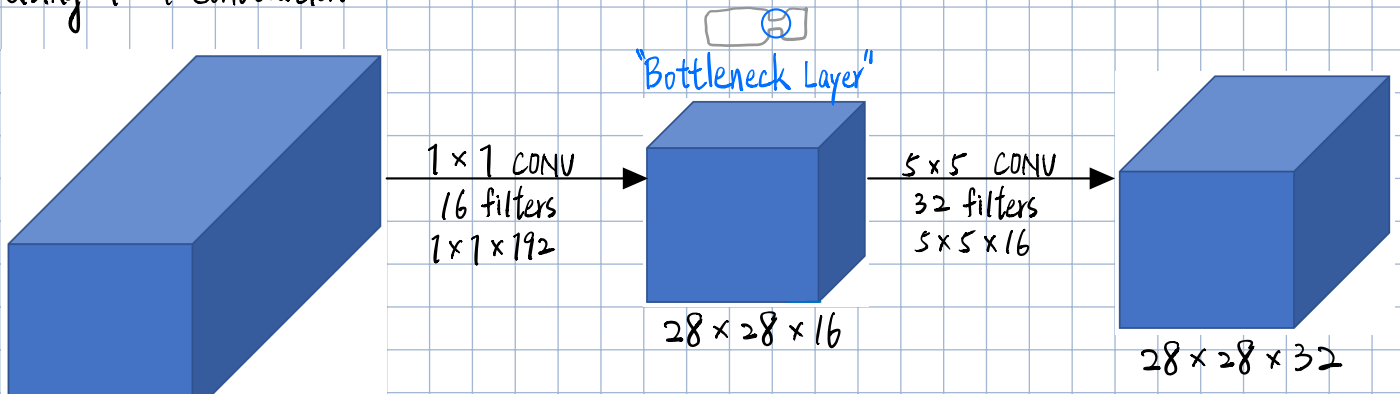
28 × 28 × 192

There are 32 filters. And each filter is $5 \times 5 \times 192$.

The computation cost is $(28 \times 28 \times 32) \cdot (5 \times 5 \times 192) = 120 \cdot 10^6$

$120 \cdot 10^6$ is too large. We need to reduce it by using 1×1 convolution below.

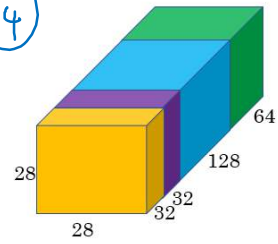
Using 1×1 convolution



28 × 28 × 192

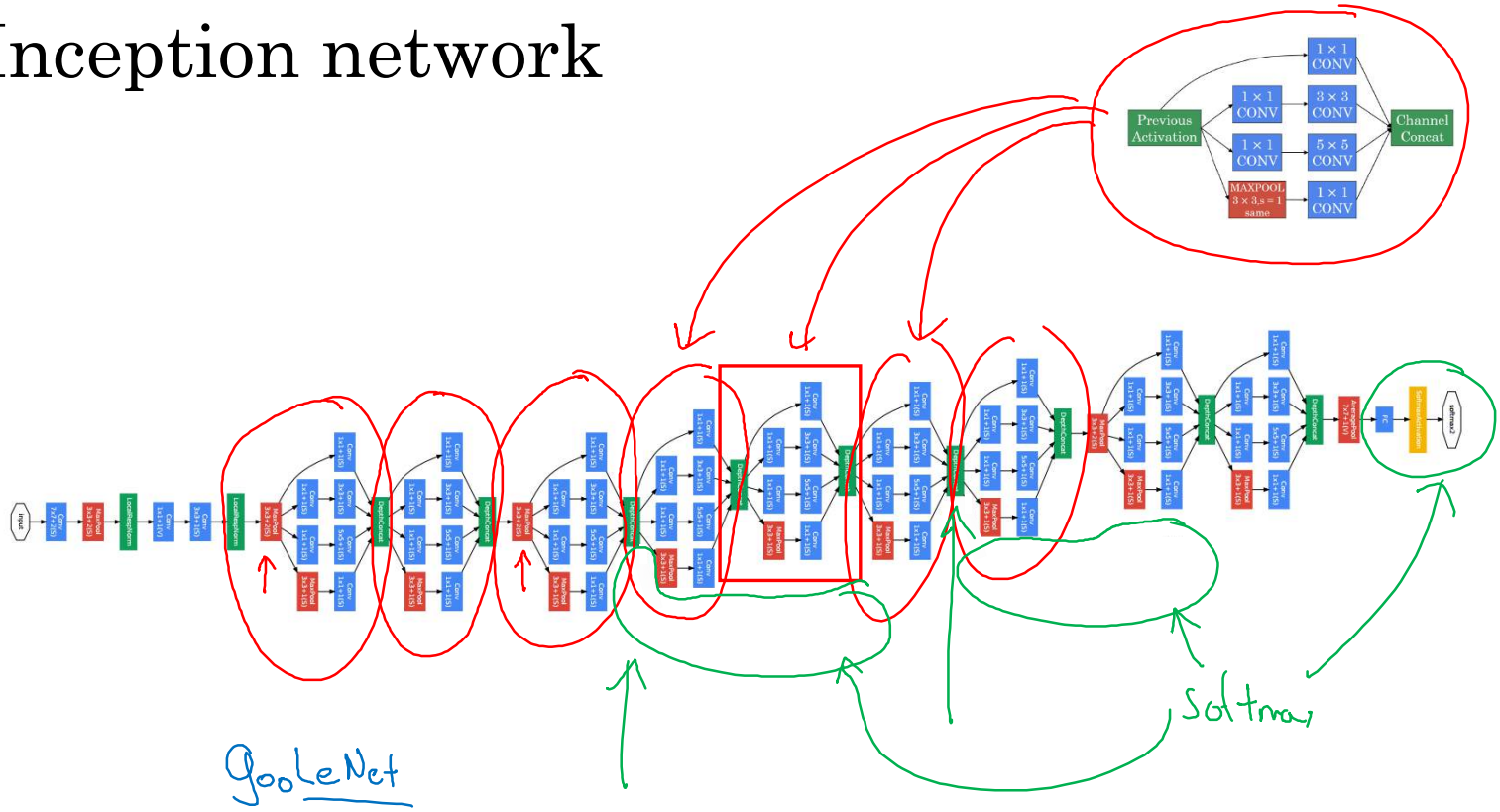
Computation Cost: $28 \times 28 \times 16 \times 192$
$+ 28 \times 28 \times 32 \cdot 5 \times 5 \times 16$
$= 2.4\,M + 10\,M = 12.4\,M \ll 120 \times 10^6$

Inception module



Andrew Ng

Inception network



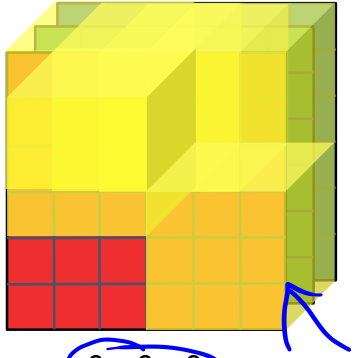
[Szegedy et al., 2014, Going Deeper with Convolutions]

Motivation for MobileNets

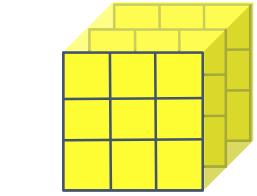
- Low computational cost at deployment
- Useful for mobile and embedded vision applications
- Key idea: Normal vs. depthwise-separable convolutions

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

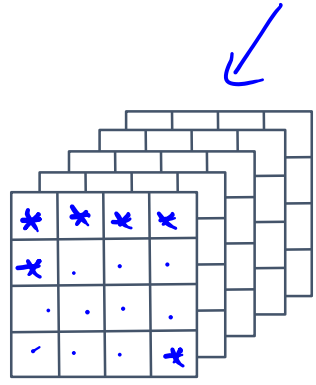
Normal Convolution



*



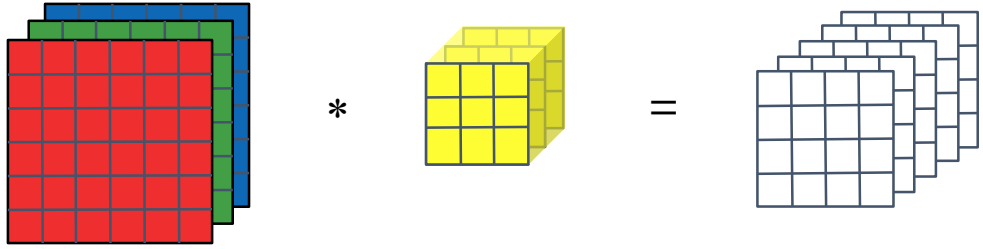
=



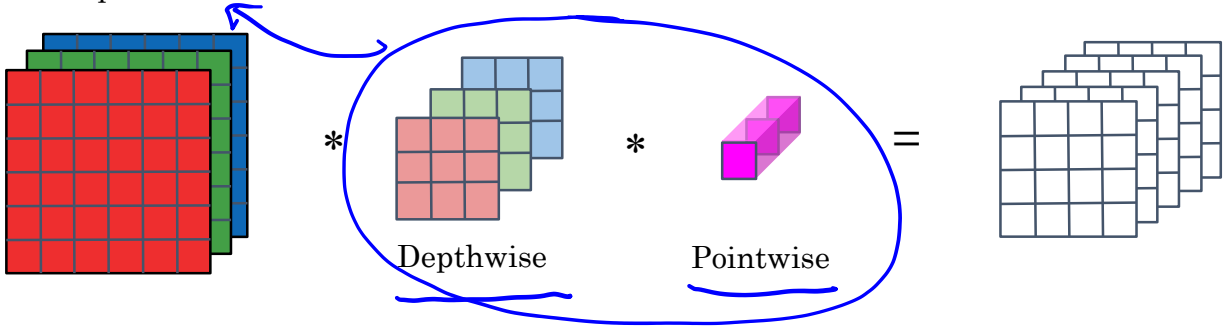
Computational cost	=	#filter params	x	# filter positions	x	# of filters (n_c')
2160	=	3x3x3	x	4x4	x	5

Depthwise Separable Convolution

Normal Convolution

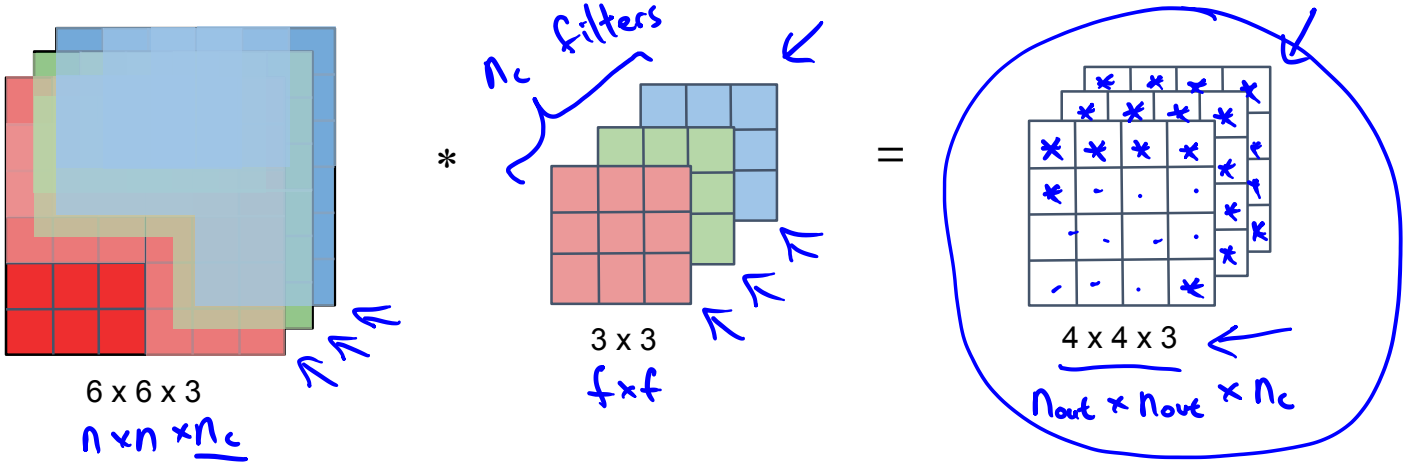


Depthwise Separable Convolution



Andrew Ng

Depthwise Convolution

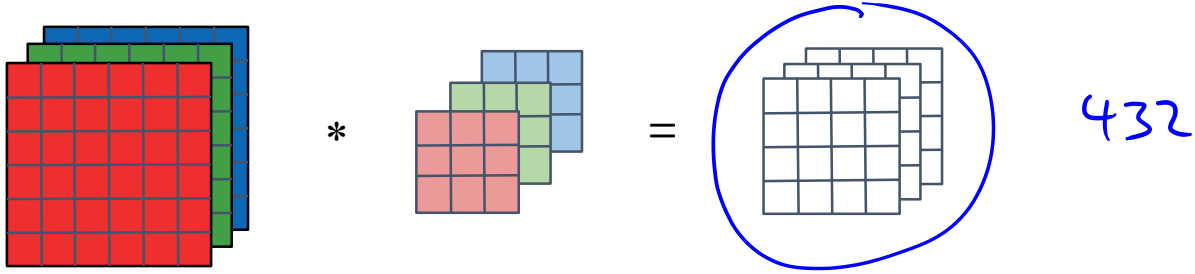


Computational cost	=	#filter params	x	# filter positions	x	# of filters
432	=	3×3	×	4×4	×	3

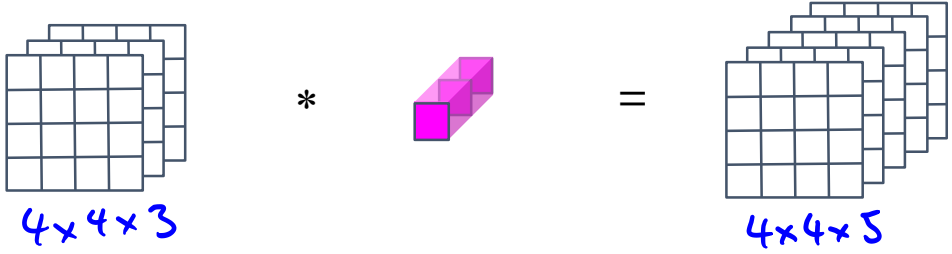
Andrew Ng

Depthwise Separable Convolution

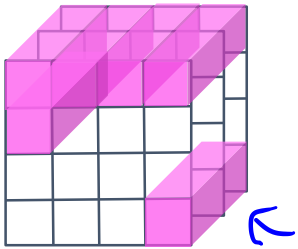
Depthwise Convolution



Pointwise Convolution



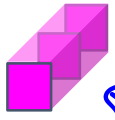
Pointwise Convolution



4 x 4 x 3

$n_{out} \times n_{out} \times n_c$

*



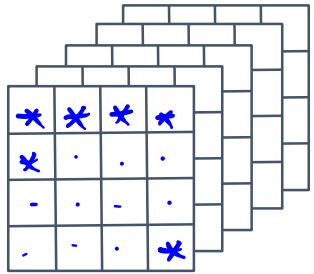
n_c' filters

5

1 x 1 x 3

$1 \times 1 \times n_c$

=



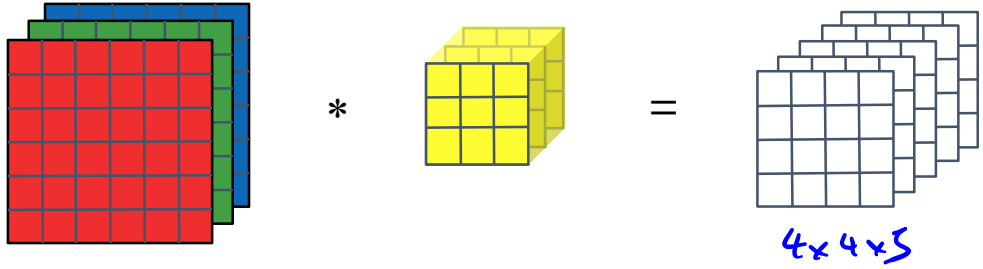
~~4 x 4 x 5~~ $4 \times 4 \times 5$

$n_{out} \times n_{out} \times n_c'$

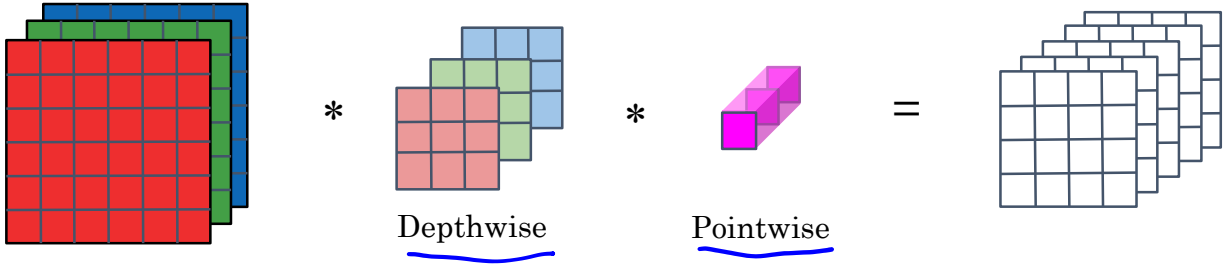
Computational cost	=	#filter params	x	# filter positions	x	# of filters
240	=	$1 \times 1 \times 3$	x	4×4	x	5

Depthwise Separable Convolution

Normal Convolution



Depthwise Separable Convolution



Andrew Ng

Cost Summary

Cost of normal convolution 2160

Cost of depthwise separable convolution

depthwise + pointwise

$432 + 240 = 672$

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

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

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

In our case, 31% cheaper

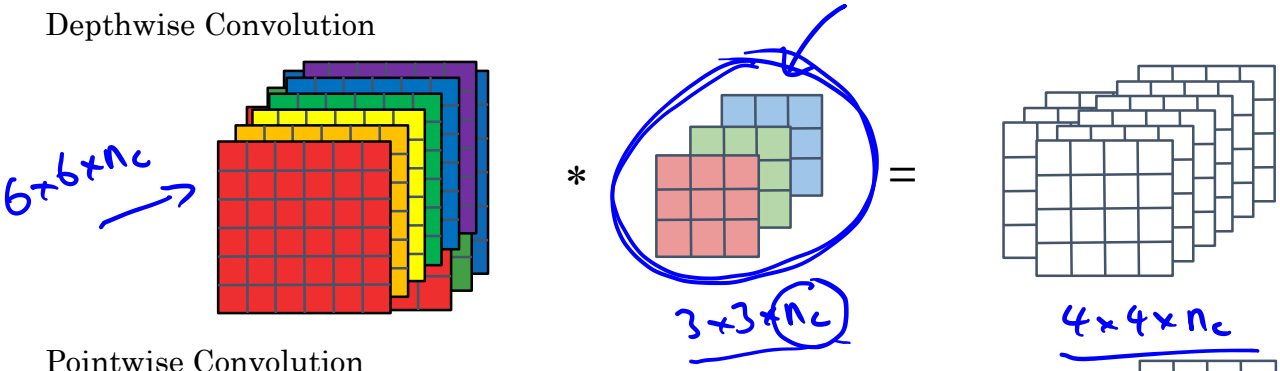
$$= \frac{1}{512} + \frac{1}{3^2}$$

$\sim$10 times cheaper

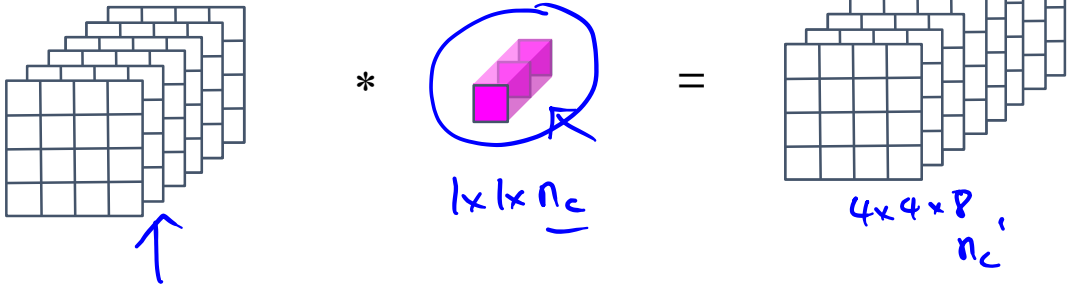
[Howard et al. 2017, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications]

Depthwise Separable Convolution

Depthwise Convolution



Pointwise Convolution



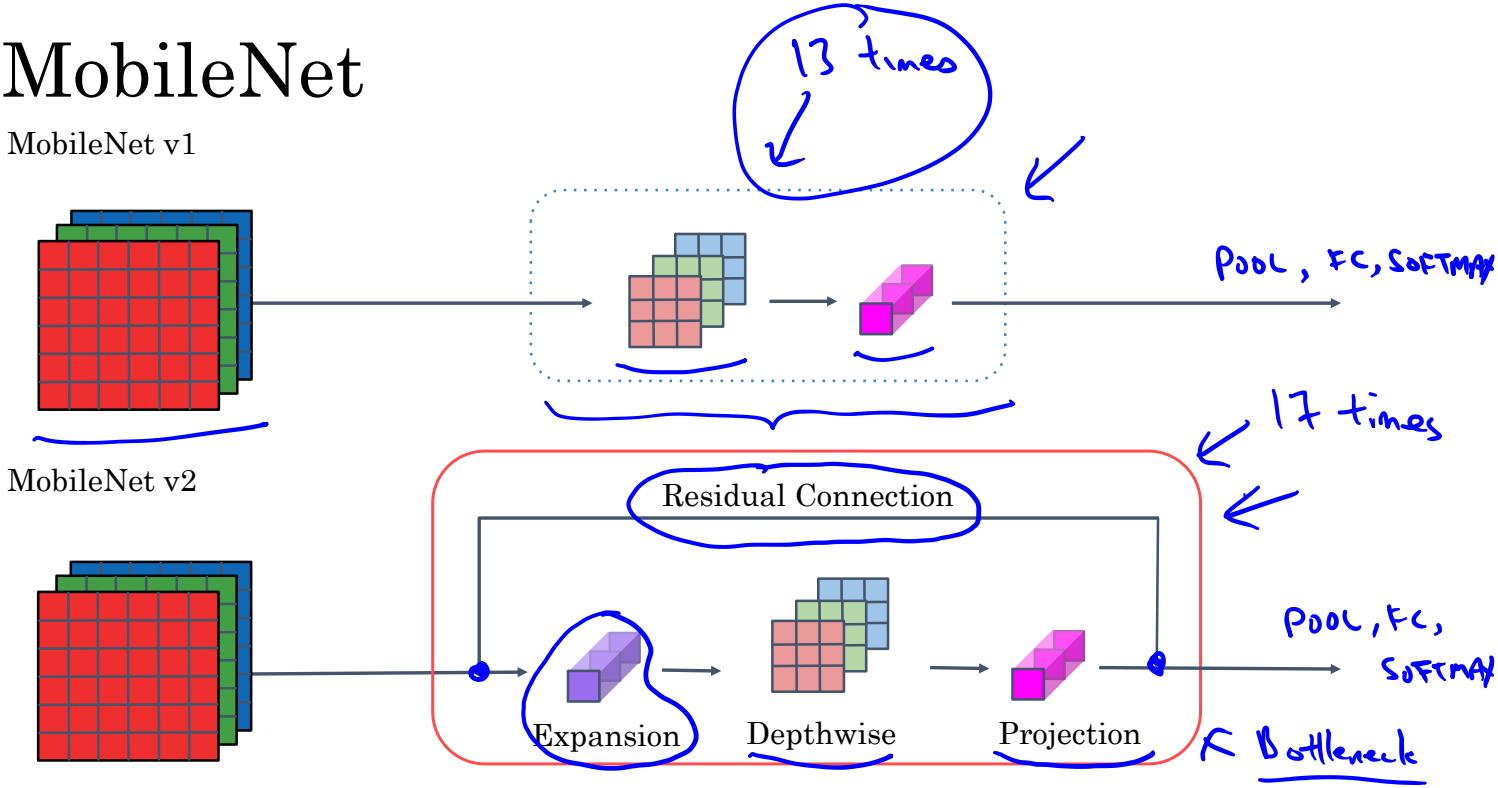
Andrew Ng

deeplearning.ai

Convolutional Neural Networks

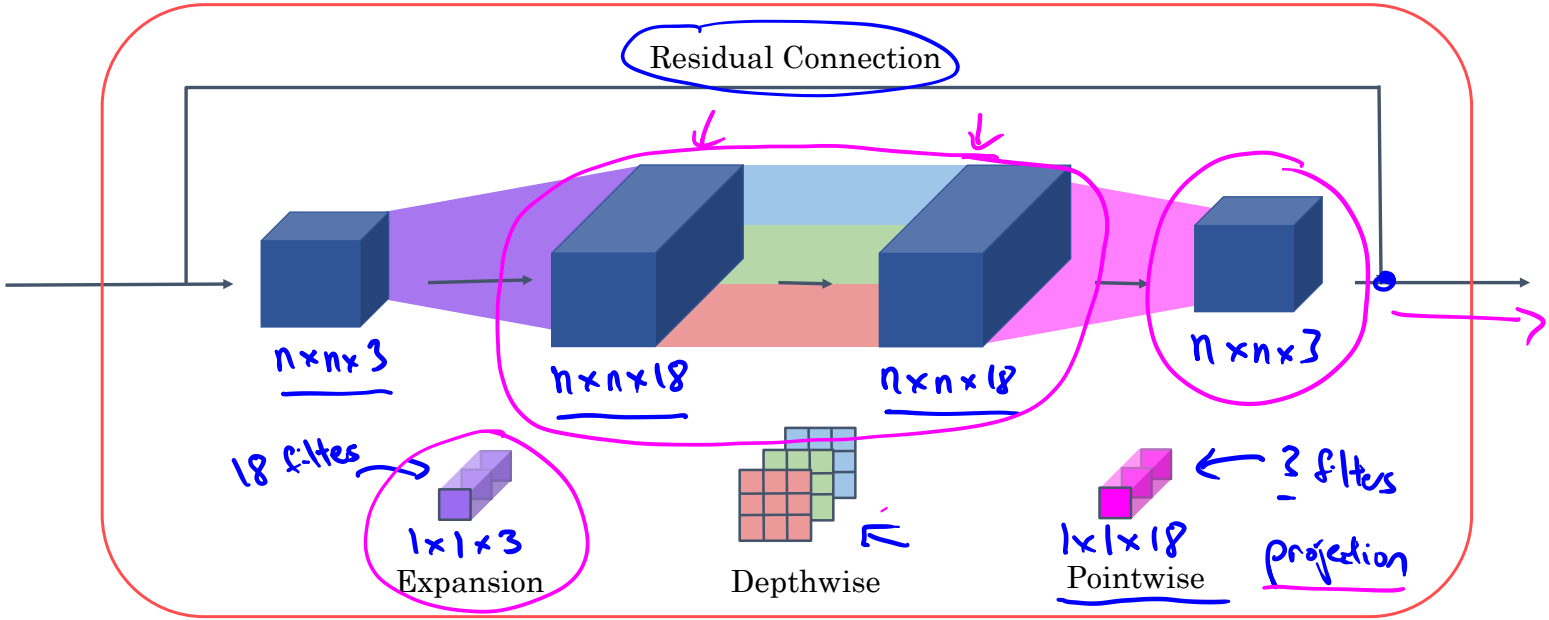
MobileNet Architecture

MobileNet



[Sandler et al. 2019, MobileNetV2: Inverted Residuals and Linear Bottlenecks]

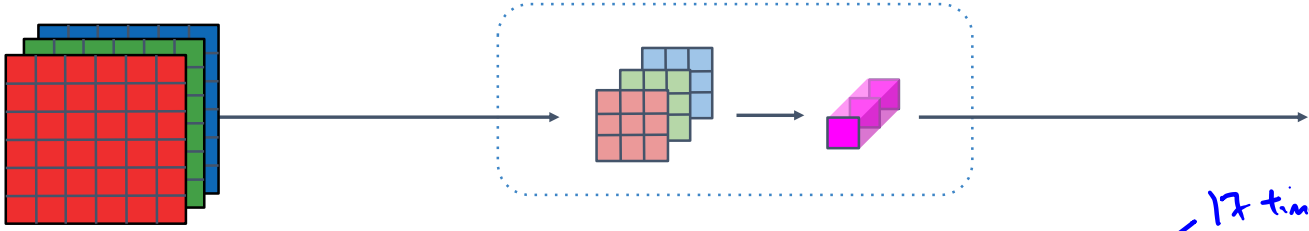
MobileNet v2 Bottleneck



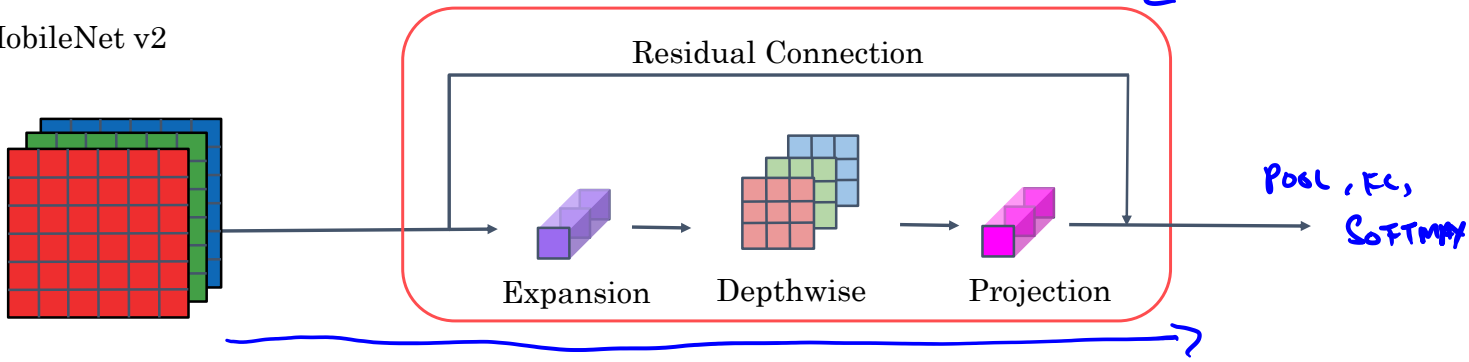
[Sandler et al. 2019, MobileNetV2: Inverted Residuals and Linear Bottlenecks]

MobileNet

MobileNet v1

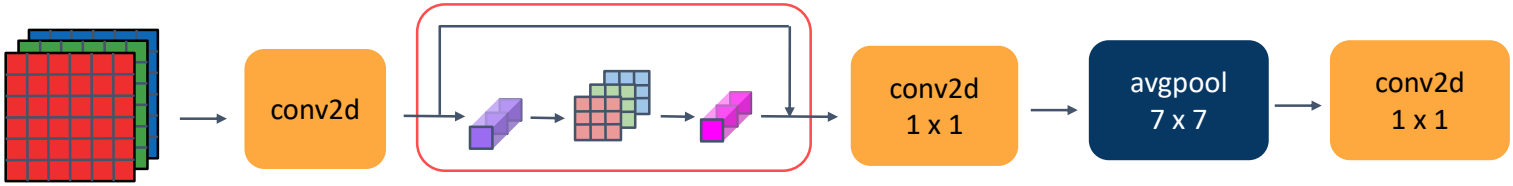


MobileNet v2



[Sandler et al. 2019, MobileNetV2: Inverted Residuals and Linear Bottlenecks]

MobileNet v2 Full Architecture



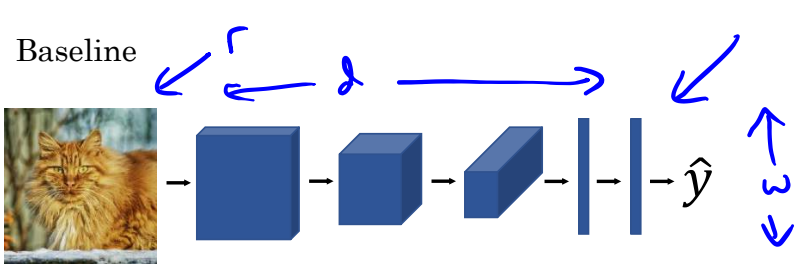
[Sandler et al. 2019, MobileNetV2: Inverted Residuals and Linear Bottlenecks]

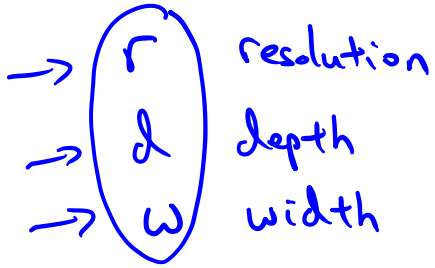
deeplearning.ai

Convolutional Neural Networks

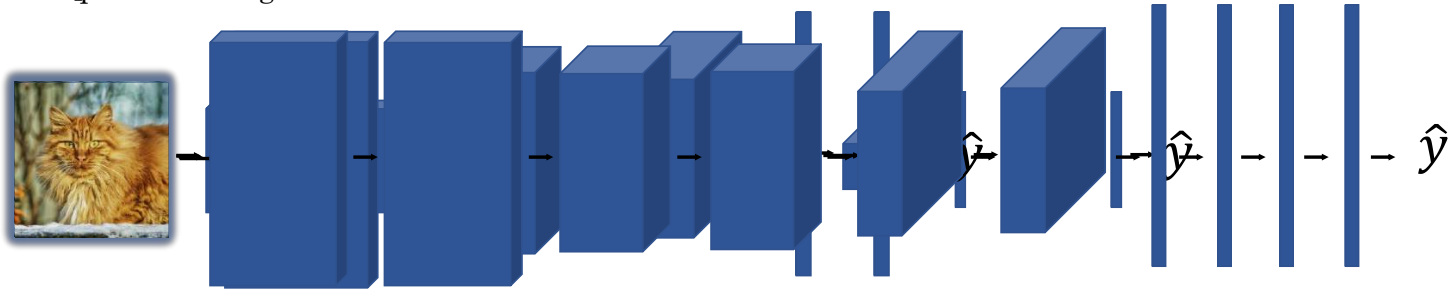
EfficientNet

EfficientNet





Bigger Resolution
Compound scaling



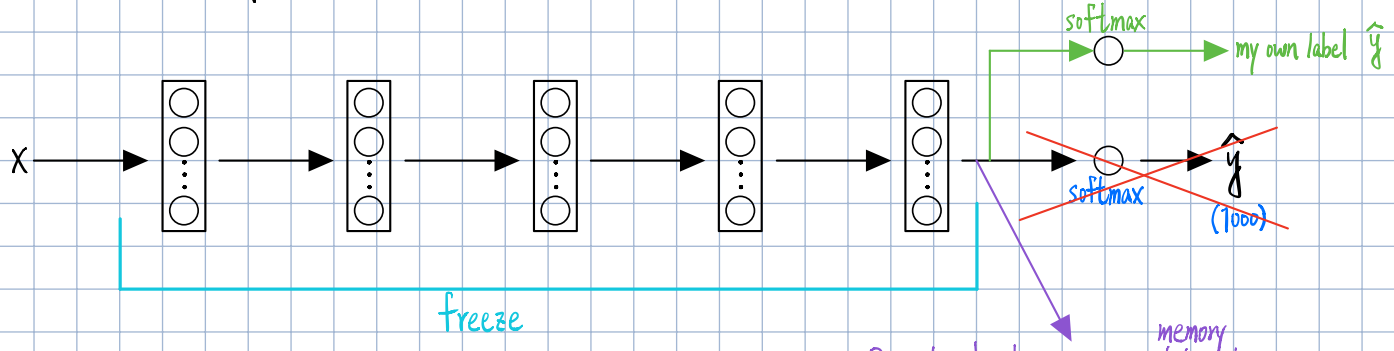
[Tan and Le, 2019, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks]

Transfer Learning

Andrew strongly suggests considering Transfer Learning for Computer Vision tasks.

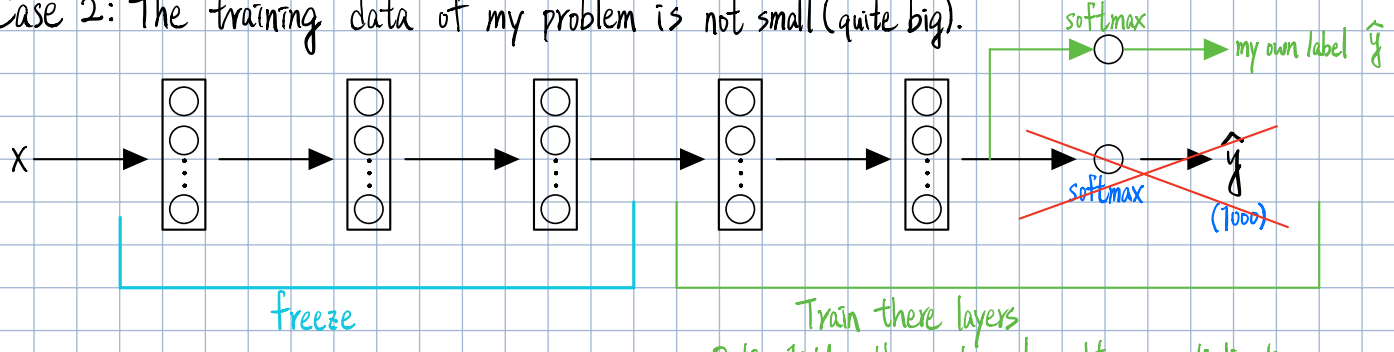
There are various ways to do transfer learning. Andrew categorizes different cases as follow:

Case 1: The training data of my problem is small. Freeze all front layers. Only train the softmax layer.



One trick to save computation memory time is to precompute the output values of the last frozen layer and save them to disk. During training, I just use the saved precomputed value as input to train the softmax layer.

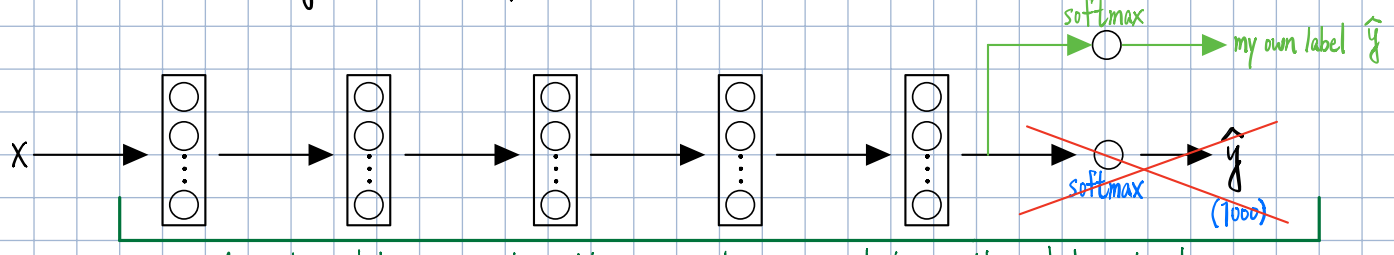
Case 2: The training data of my problem is not small (quite big).



Option 1: Use the pretrained weights as initialization

Option 2: Scratch the pretrained layers or weights. Start from scratch. Design new layers. Use random initialization.

Case 3: The training data of my problem is A LOT.



Use the whole pretrained weights as initialization and train the whole network.

deeplearning.ai

Practical advice for using ConvNets

Data augmentation

Common augmentation method

Remember to use multithreading to implement data augmentation during Training.

Mirroring

y

Random Cropping

- Rotation
- Shearing

- Local warping
- …

Color shifting

R G B

+20,-20,+20

-20,+20,+20

+5,0,+50

y

Advanced:

PCA

ml-class.org

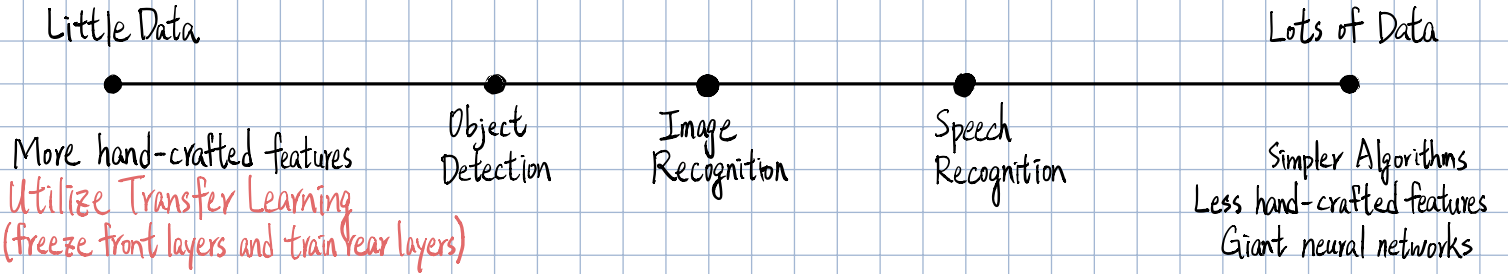
[AlexNet paper

"PCA color augmentation."

R B G

The State of Computer Vision

- Use open source implementations of networks' architectures published in the literature.
- Use pre-trained models and fine-tune on your dataset.



Machine Learning algorithms can be seen as having two sources of knowledge:

1. Labeled data (x, y) pair.
2. Hand engineered features/network architecture/other components

Andrew says hand engineering data is getting less important if there are lots of data.

I agree: Lots of data can improve accuracy a lot. It's definitely good.

I do NOT agree: Hand engineering insights can help to optimize a system's performance (memory usage / CPU usage / accuracy) with the same amount of data. It doesn't matter there is a lot or a few data. As long as hand engineering insights are used in a system, I am sure that my system has reached its theoretically best performance.

Eg, in object detection, an ML algorithm with dynamic system model (car) is definitely better than a "pure" ML algorithm without the knowledge of the dynamic system model.

Tips for doing well on benchmarks/winning competition. Andrew also says these two tips will probably not be helpful for deploying an actual application. So, these two tips are probably irrelevant to me. Another implication is that be suspicious about the results in papers. Papers will just use whatever means they can to show they are better when in actuality they are not. Thus, I must learn how to recognize such papers and throw them away. Anyway, here are the two tips:

- Ensembling: Train several networks independently and average their outputs $\hat{y}$.
- Multi-crop at test time (10-crop): Run classifier on multiple versions of test images and average results.