

# Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>



deeplearning.ai

# Case Studies

---

Why look at  
case studies?

# Outline

## Classic networks:

- LeNet-5 ←
- AlexNet ←
- VGG ←

ResNet (152)

Inception



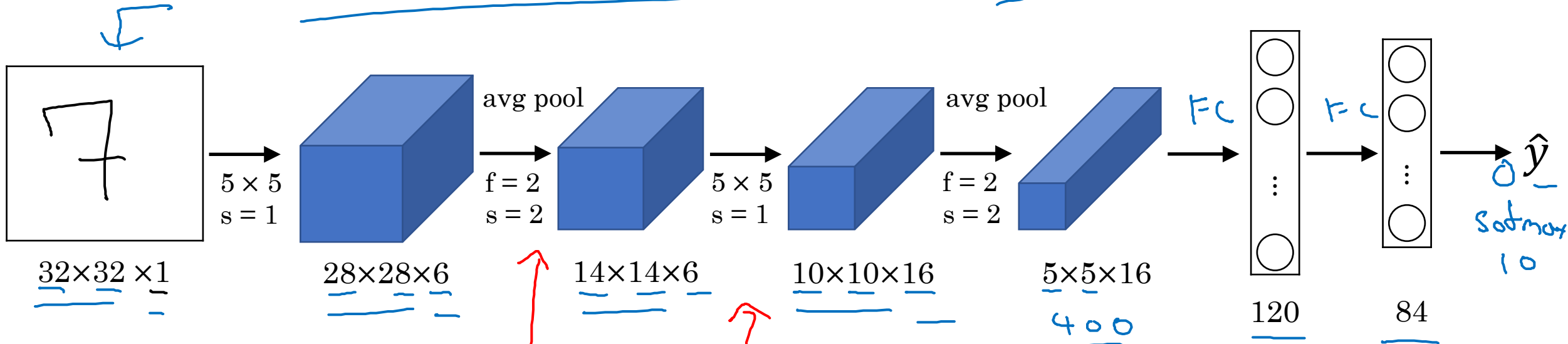
deeplearning.ai

# Case Studies

---

## Classic networks

# LeNet - 5



60K parameters.

$n_H, n_w \downarrow$   $n_c \uparrow$

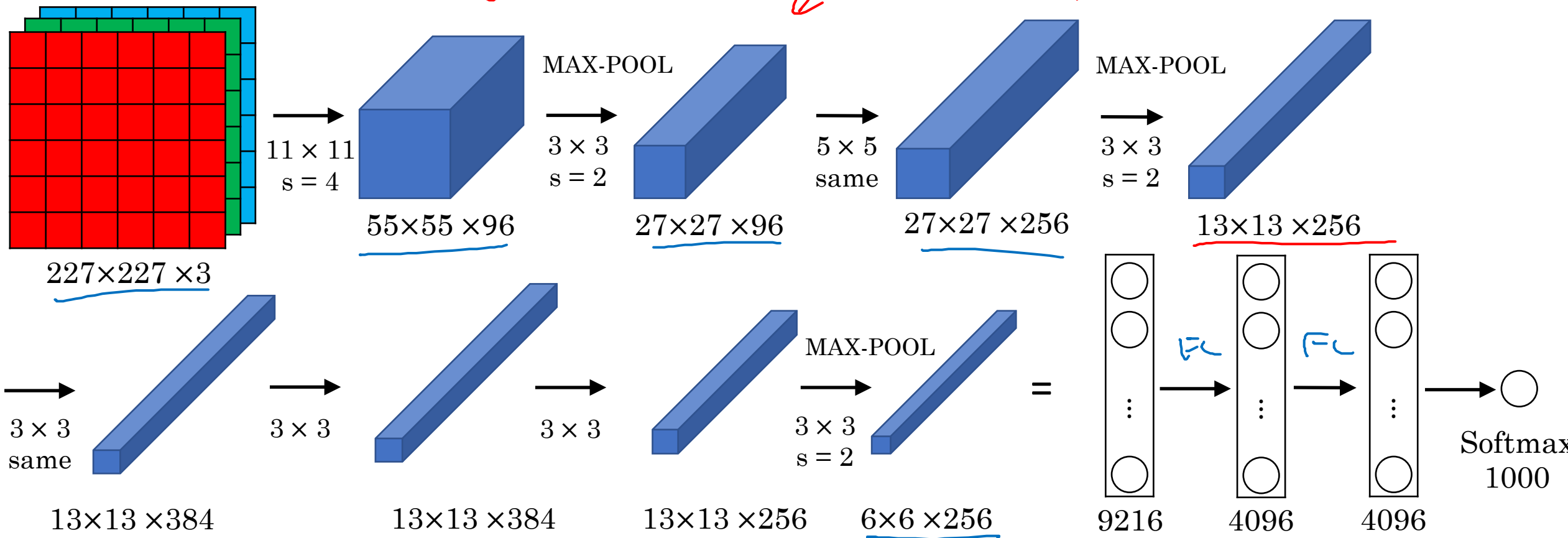
conv pool conv pool fc fc output

non-linearity after pooling  $n_H \times n_w \times n_c$   $f \times f \times n_c$

Advanced: sigmoid/tanh ReLU

II III

# AlexNet

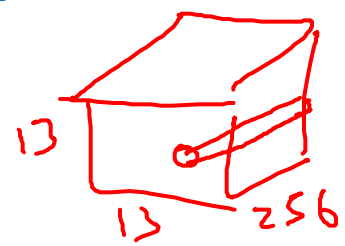


- Similar to LeNet, but much bigger. 9216 160M parameters

- ReLU

- Multiple GPUs.

- Local Response Normalization (LRN)

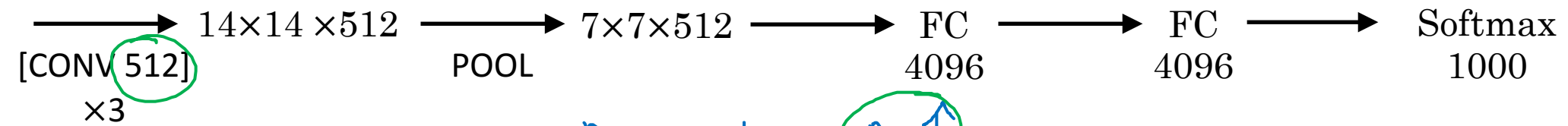
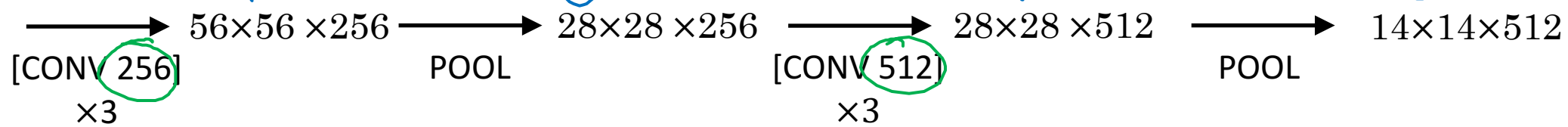
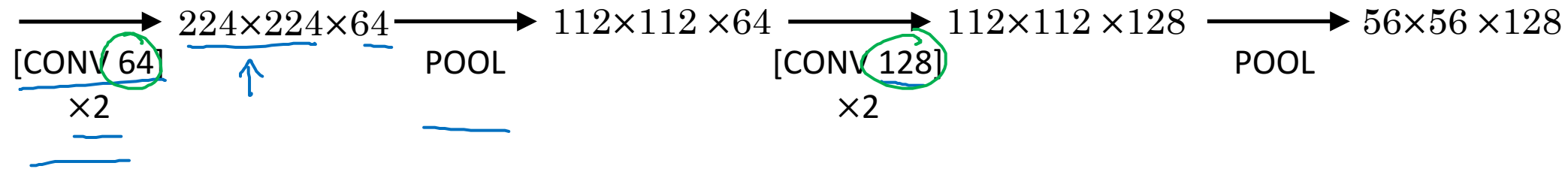
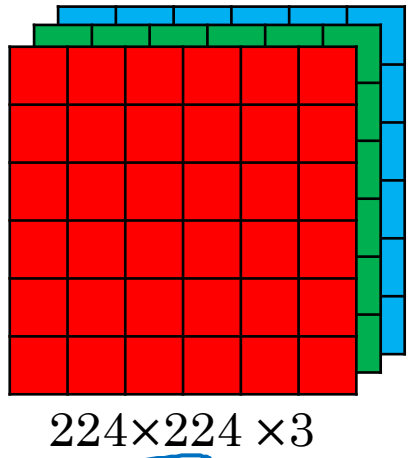
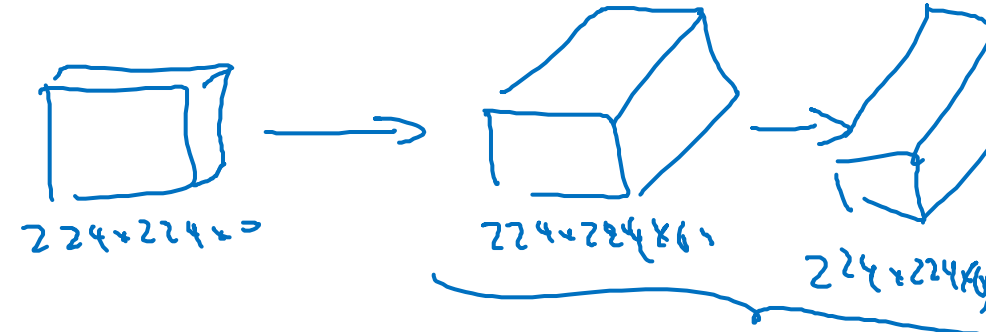


# VGG - 16

VGG-19

CONV = 3x3 filter, s = 1, same

MAX-POOL = 2x2, s = 2



$n_H, n_W \downarrow$

$n_C \uparrow$

~138M



deeplearning.ai

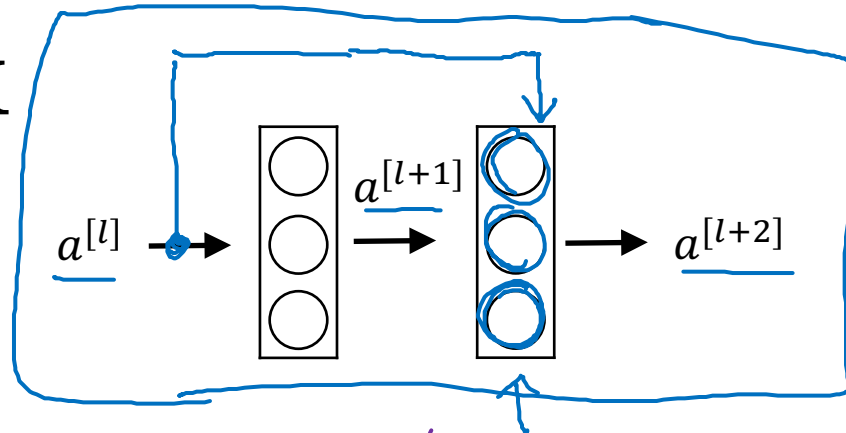
# Case Studies

---

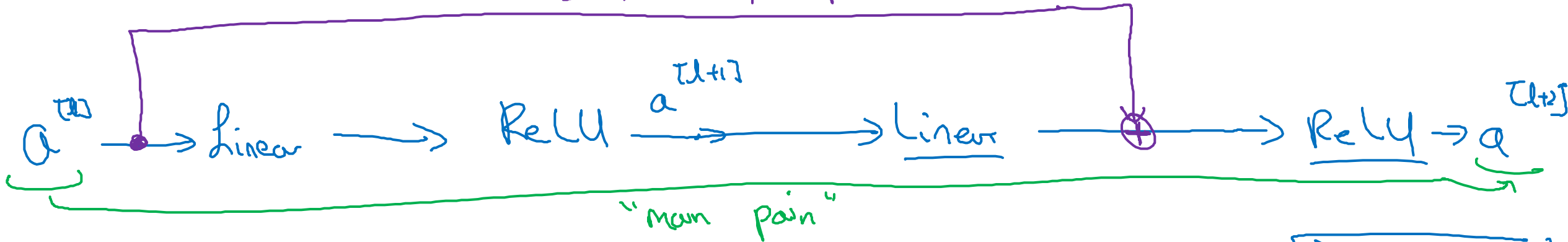
## Residual Networks (ResNets)



# Residual block



"short cut" / skip connection



$$\underline{z^{[l+1]}} = W^{[l+1]} \underline{a^{[l]}} + b^{[l+1]}$$

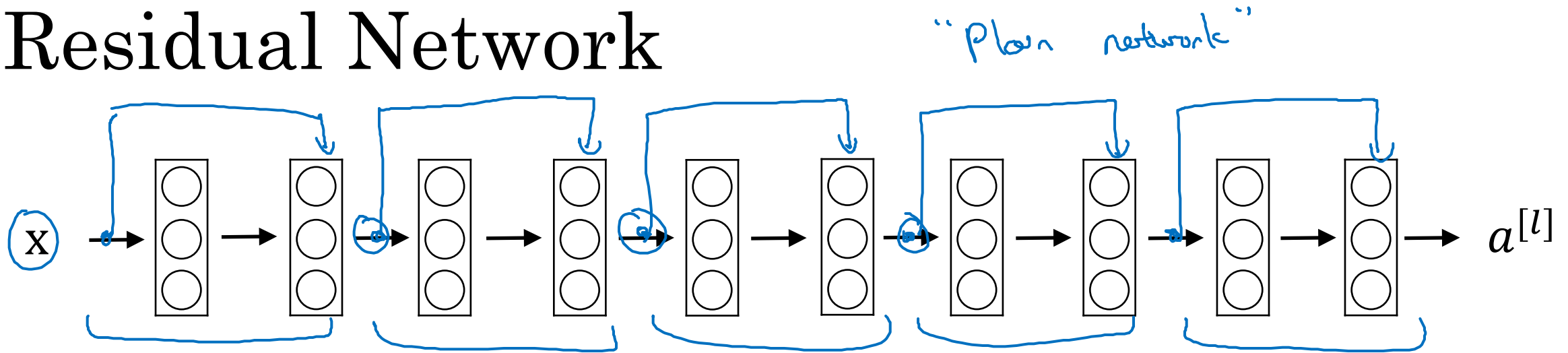
$$\underline{a^{[l+1]}} = g(\underline{z^{[l+1]}})$$

$$\underline{z^{[l+2]}} = W^{[l+2]} \underline{a^{[l+1]}} + b^{[l+2]}$$

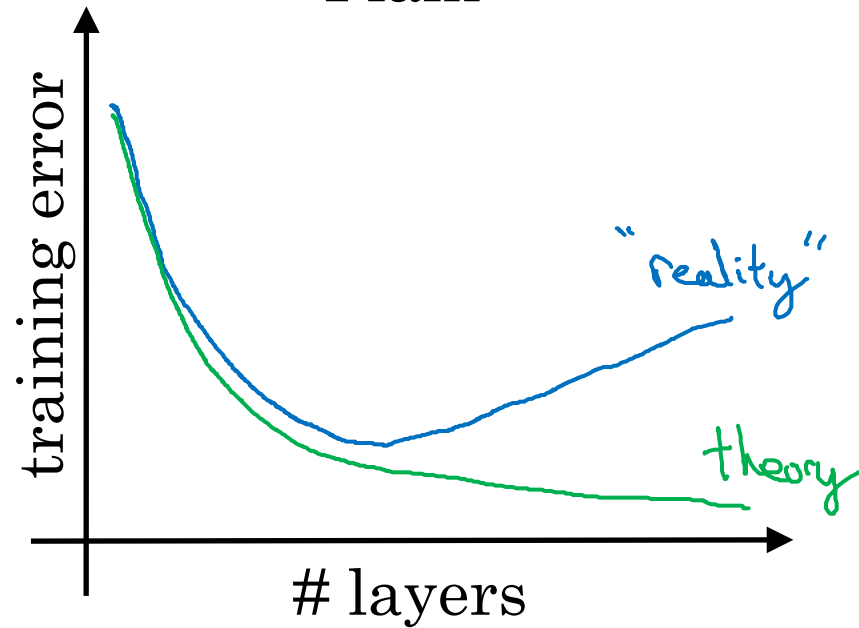
~~$$\underline{a^{[l+2]}} = g(\underline{z^{[l+2]}})$$~~

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

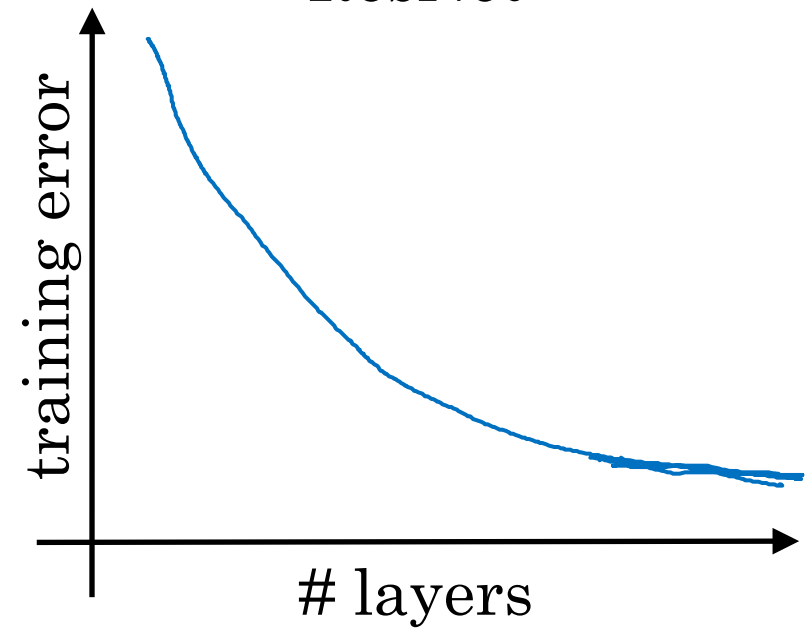
# Residual Network



Plain



ResNet





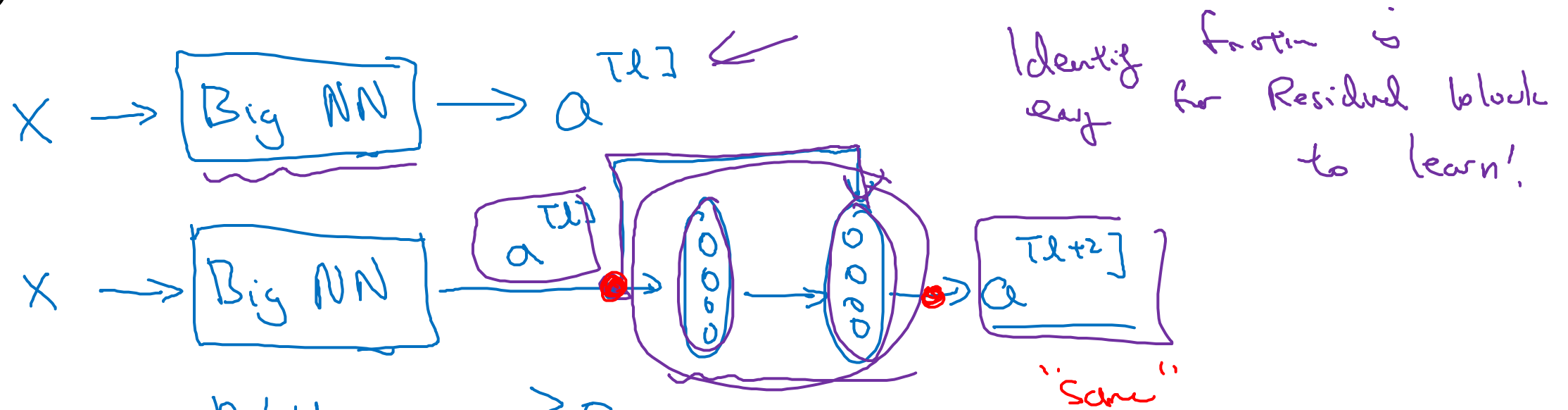
deeplearning.ai

# Case Studies

---

# Why ResNets work

# Why do residual networks work?



ReLU.

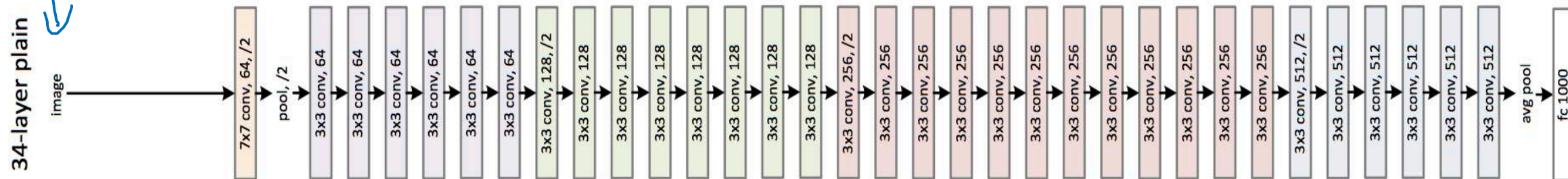
$$a \geq 0$$

$$\begin{aligned}
 a^{[l+2]} &= g(z^{[l+2]} + a^{[l]}) \\
 &= g(\cancel{w^{[l+2]} a^{[l+1]} + b^{[l+2]}} + \underbrace{w_s}_{R^{256 \times 128}} \underbrace{a^{[l]}}_{128}) = g(a^{[l]}) \\
 &= \underline{a^{[l]}}
 \end{aligned}$$

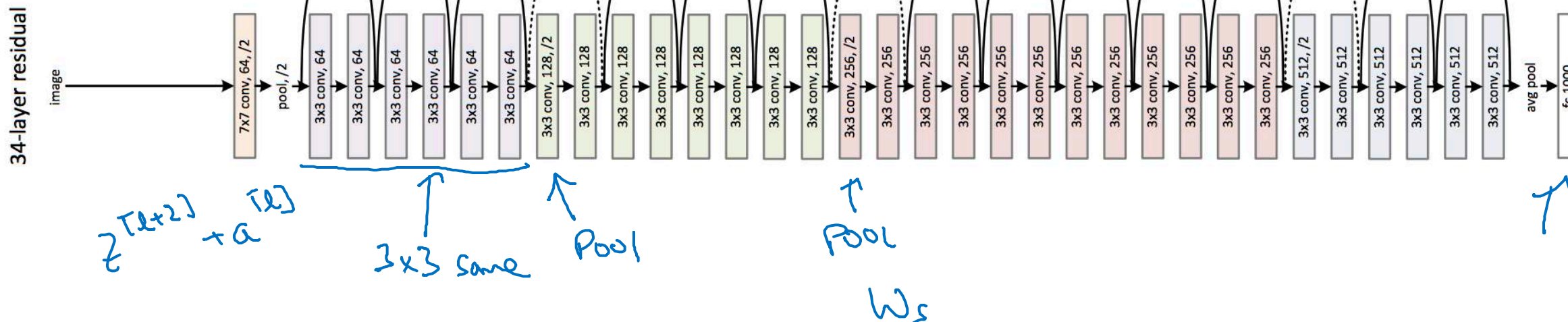
If  $w^{[l+2]} = 0, b^{[l+2]} = 0$

# ResNet

## Plain



## ResNet





deeplearning.ai

# Case Studies

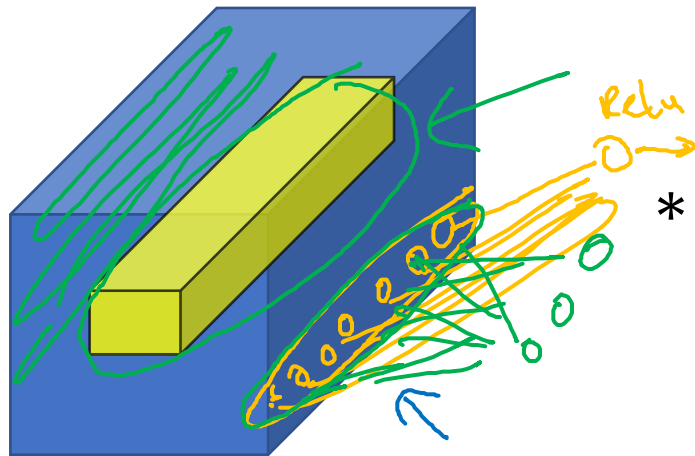
---

**Network in Network  
and  $1 \times 1$  convolutions**

# Why does a $1 \times 1$ convolution do?

1	2	3	6	5	8
3	5	5	1	3	4
2	1	3	4	9	3
4	7	8	5	7	9
1	5	3	7	4	8
5	4	9	8	3	5

$6 \times 6 \times 1$



$6 \times 6 \times 32$

\*

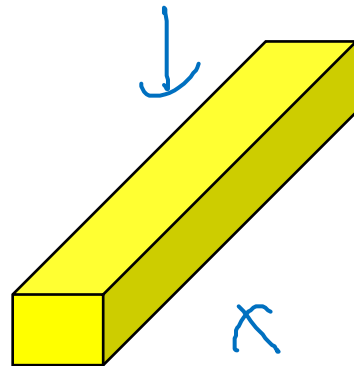
2

=



32

→ # filters.  
 $n_c^{[L+1]}$



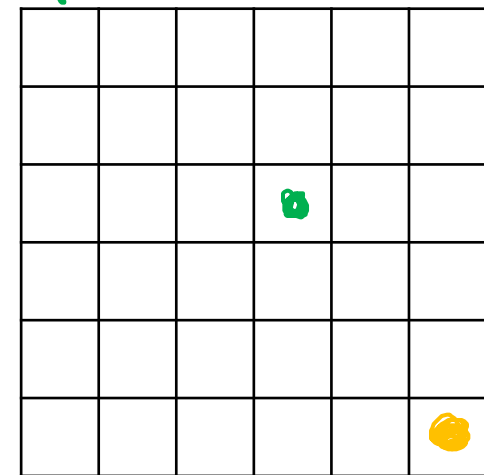
$1 \times 1 \times 32$

=

ReLU

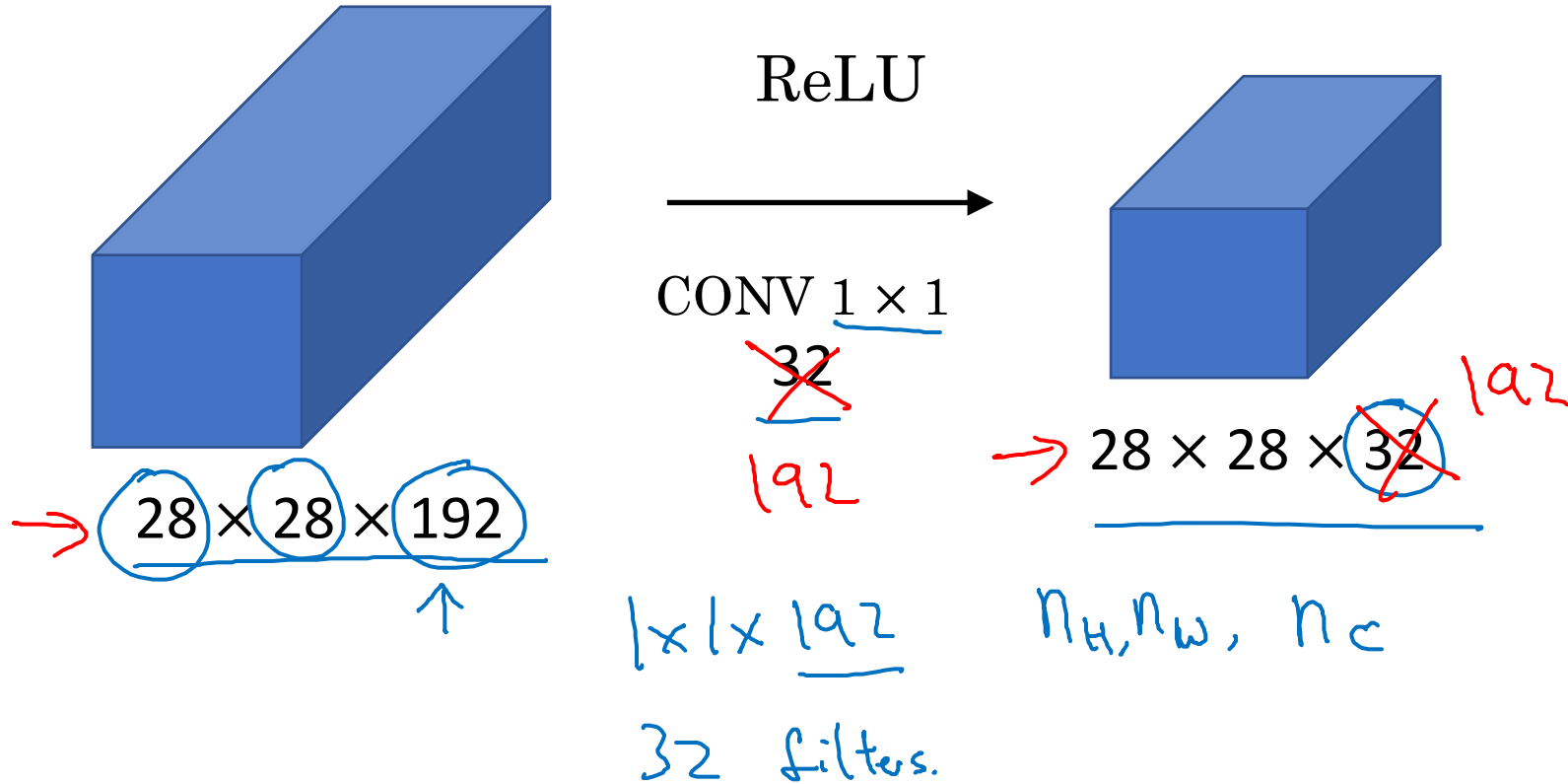
Network in Network

2	4	6	...		



$6 \times 6 \times \# \text{ filters}$

# Using $1 \times 1$ convolutions







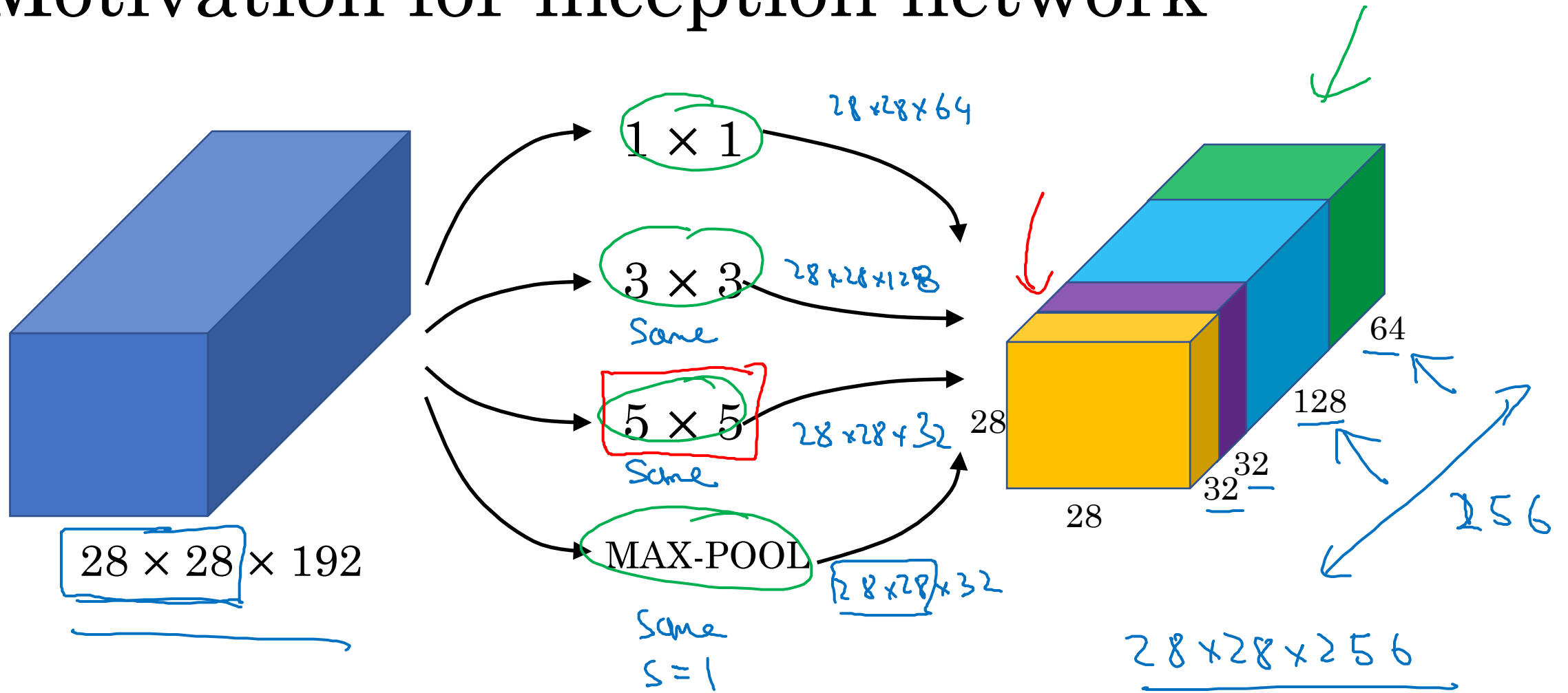
deeplearning.ai

# Case Studies

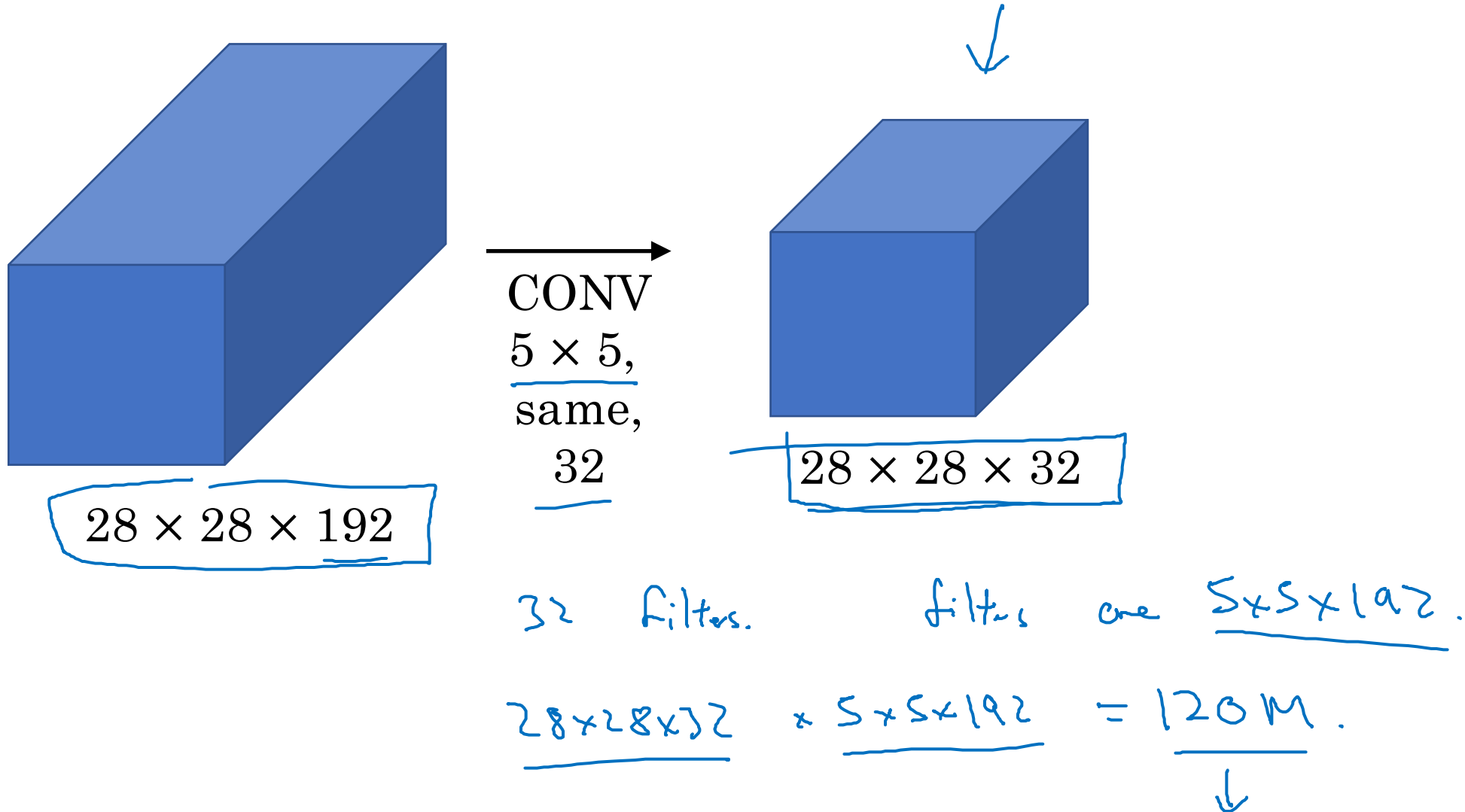
---

## Inception network motivation

# Motivation for inception network

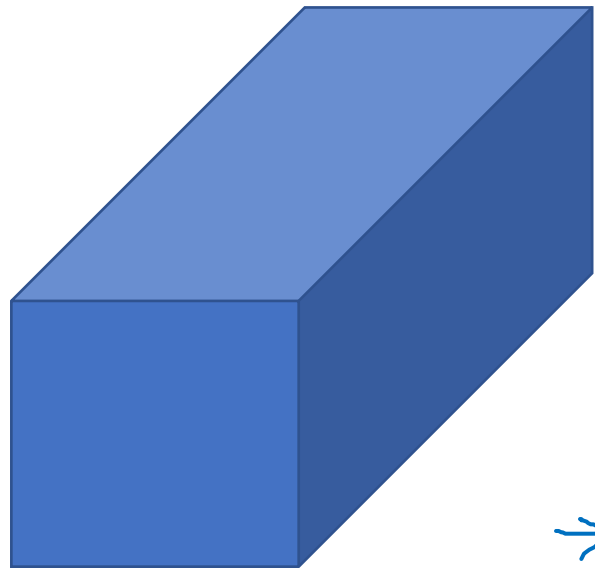
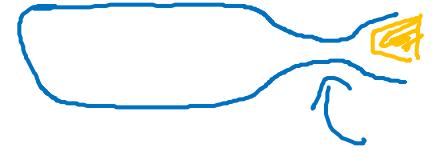


# The problem of computational cost



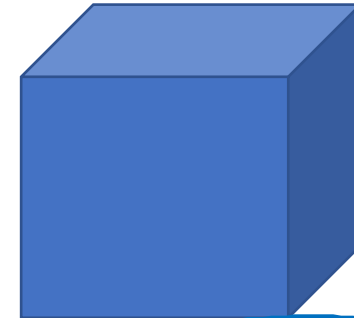
# Using $1 \times 1$ convolution

"bottleneck layer"



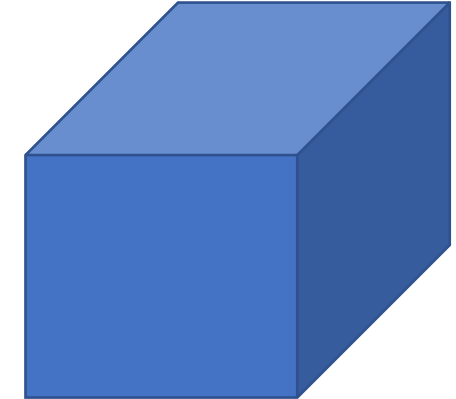
$28 \times 28 \times 192$

CONV  
 $1 \times 1$   
 $\rightarrow 16$   
 $\rightarrow 1 \times 1 \times 192$



$28 \times 28 \times 16$

CONV  
 $5 \times 5$   
32,  
 $5 \times 5 \times 16$



$28 \times 28 \times 32$

$$28 \times 28 \times 16 \times 192 = 2.4M$$

$$28 \times 28 \times 32 \times 5 \times 5 \times 16 = 10.0M$$

12.4M

120M



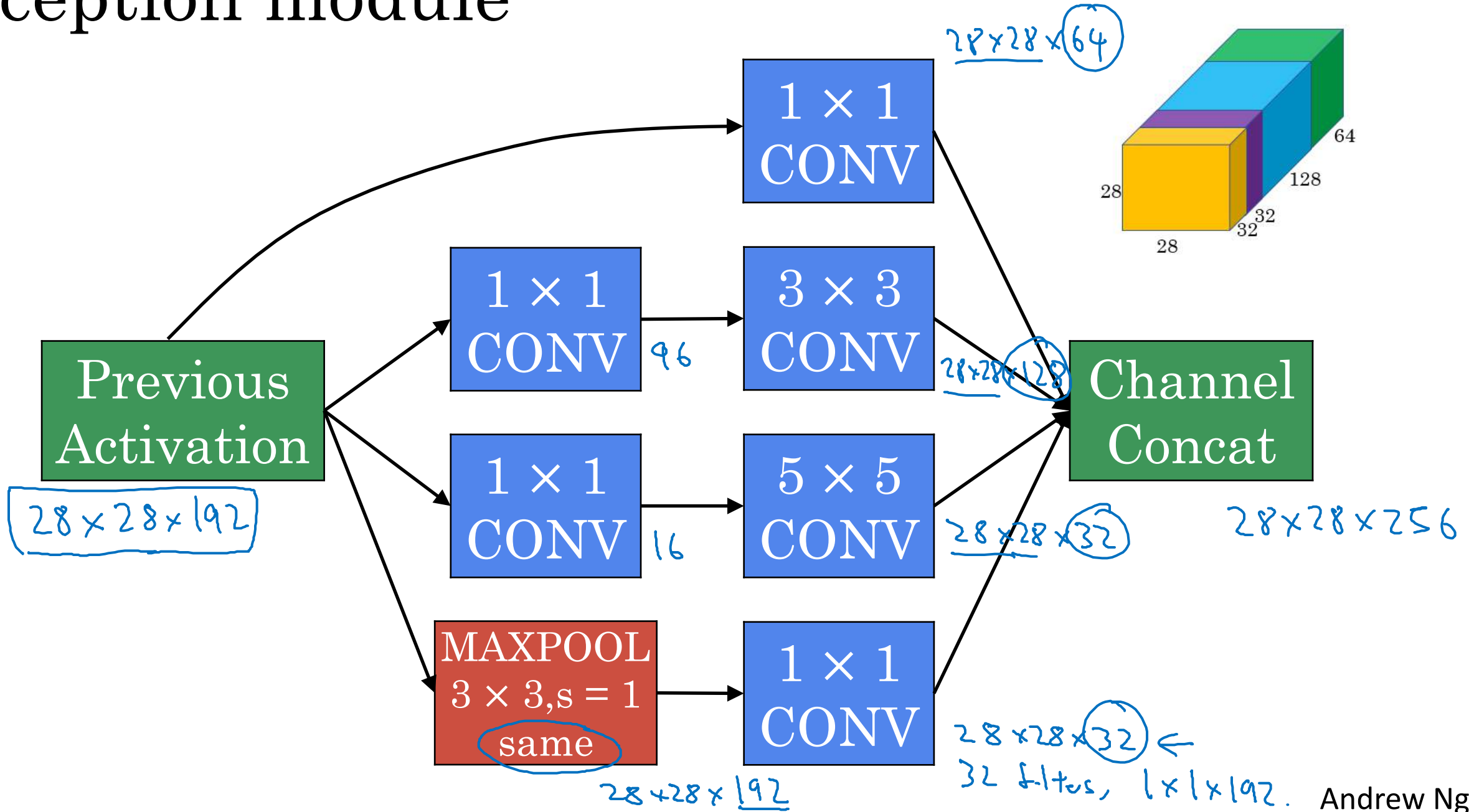
deeplearning.ai

# Case Studies

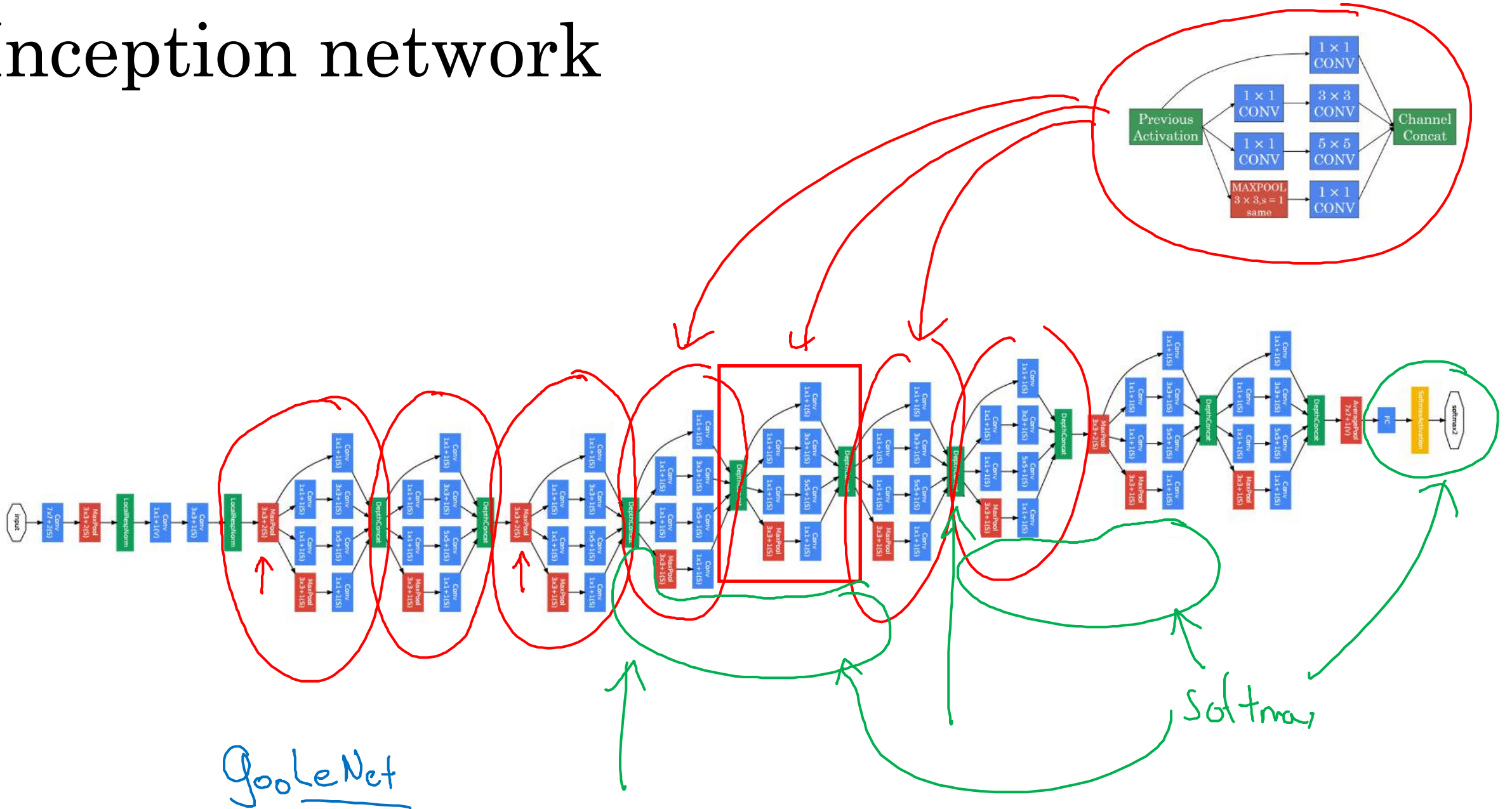
---

## Inception network

# Inception module



# Inception network









deeplearning.ai

# Convolutional Neural Networks

---

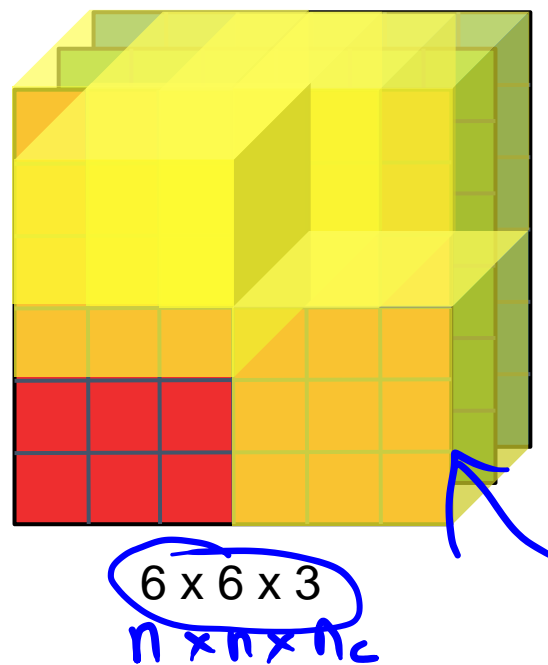
## MobileNet

# Motivation for MobileNets

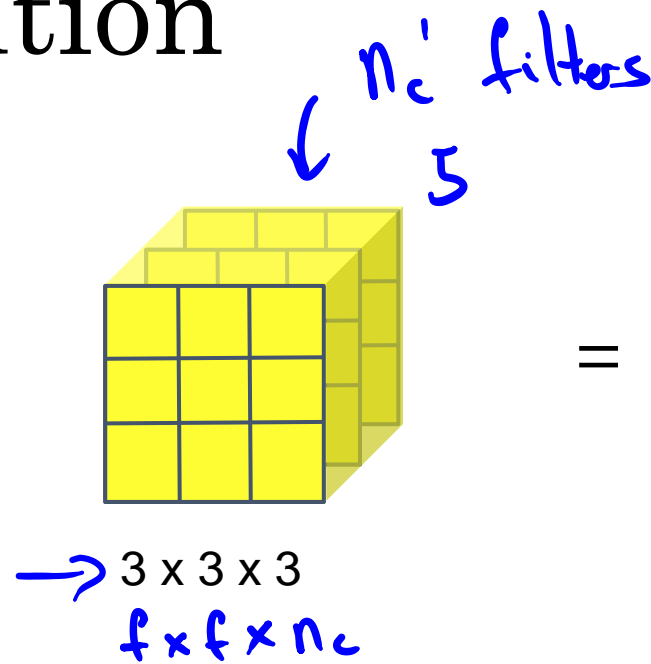
- Low computational cost at deployment
- Useful for mobile and embedded vision applications
- Key idea: Normal vs. depthwise-separable convolutions



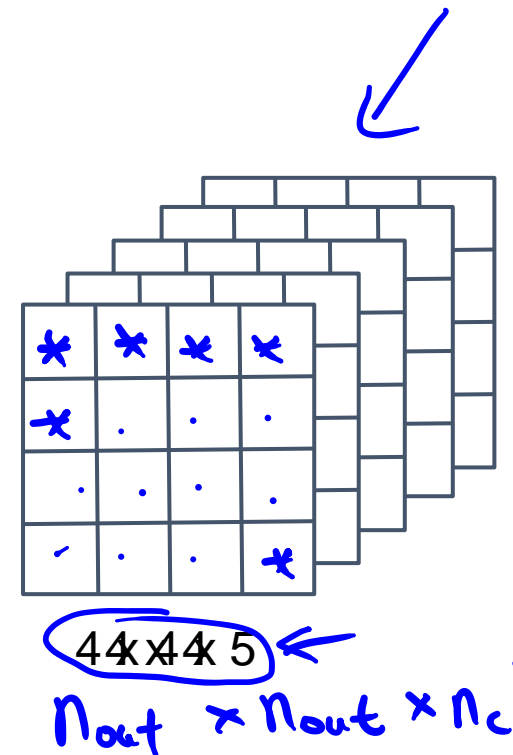
# Normal Convolution



\*



=

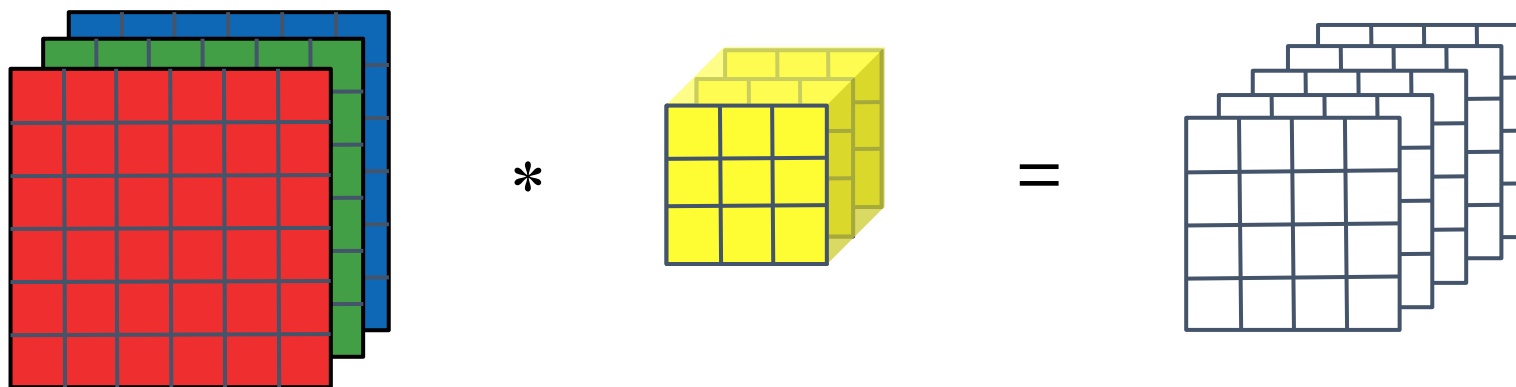


Computational cost =  $\frac{\text{\#filter params}}{3 \times 3 \times 3} \times \frac{\text{\# filter positions}}{4 \times 4} \times \text{\# of filters}$

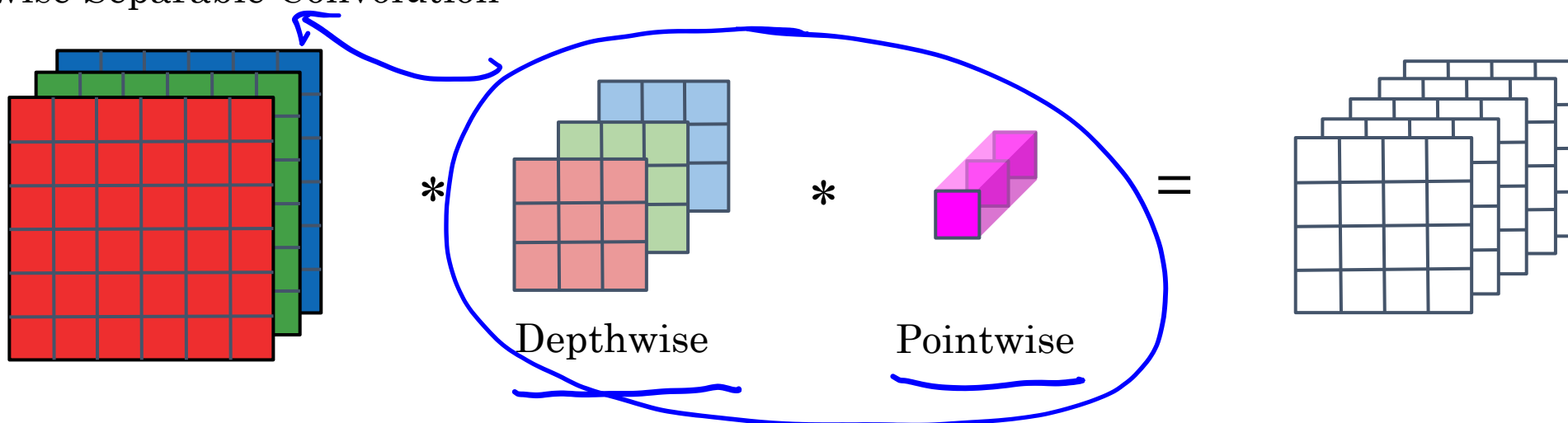
$\rightarrow \underline{2160} = \frac{3 \times 3 \times 3}{3 \times 3 \times 3} \times \frac{4 \times 4}{4 \times 4} \times 5$

# Depthwise Separable Convolution

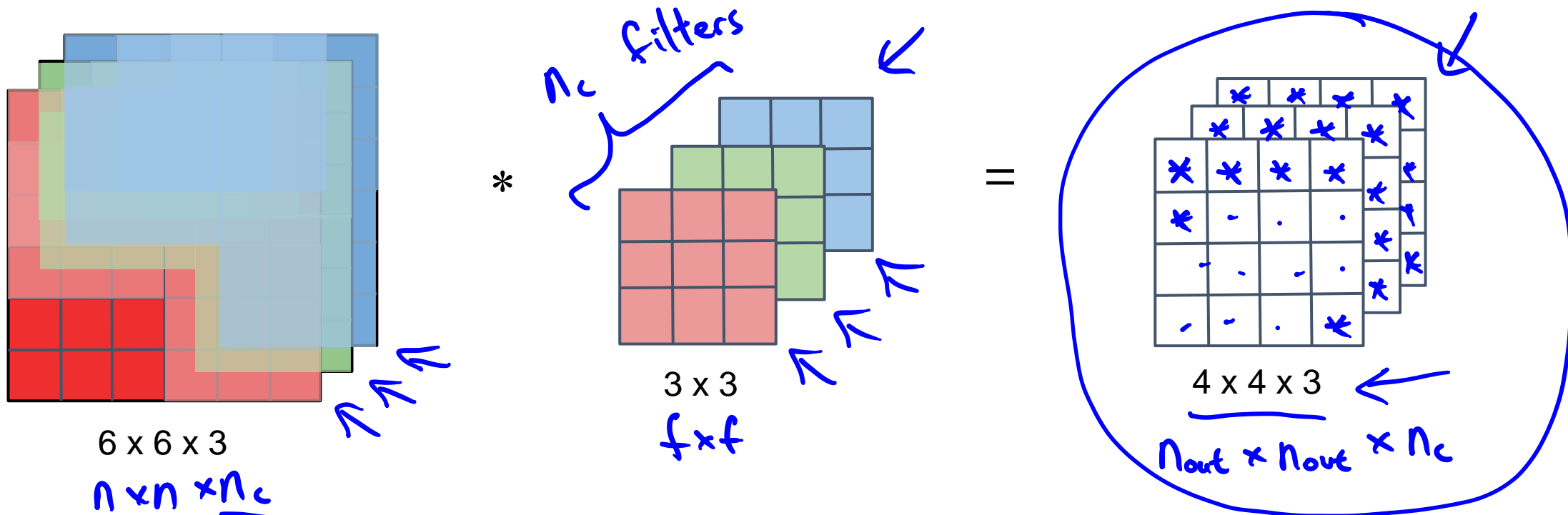
Normal Convolution



Depthwise Separable Convolution



# Depthwise Convolution

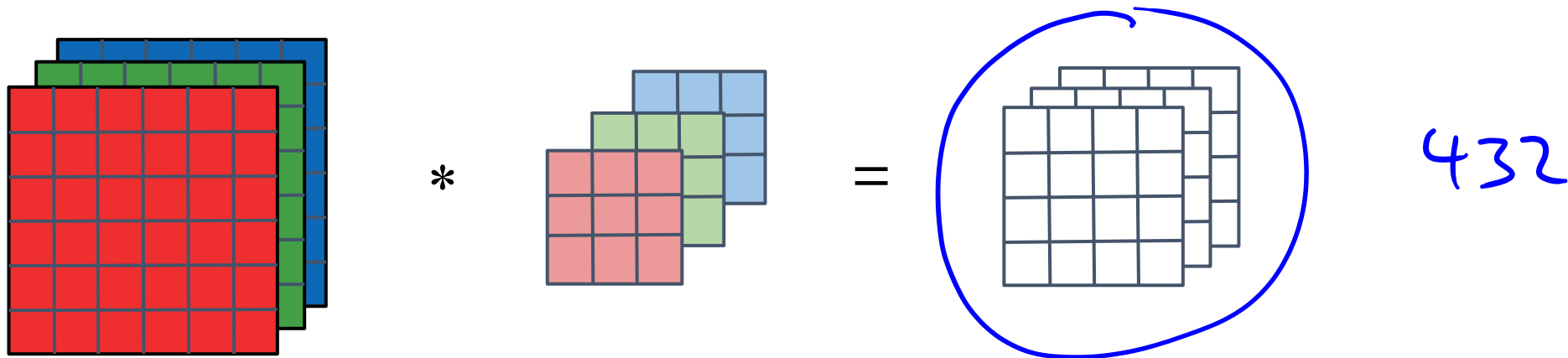


Computational cost = #filter params x # filter positions x # of filters

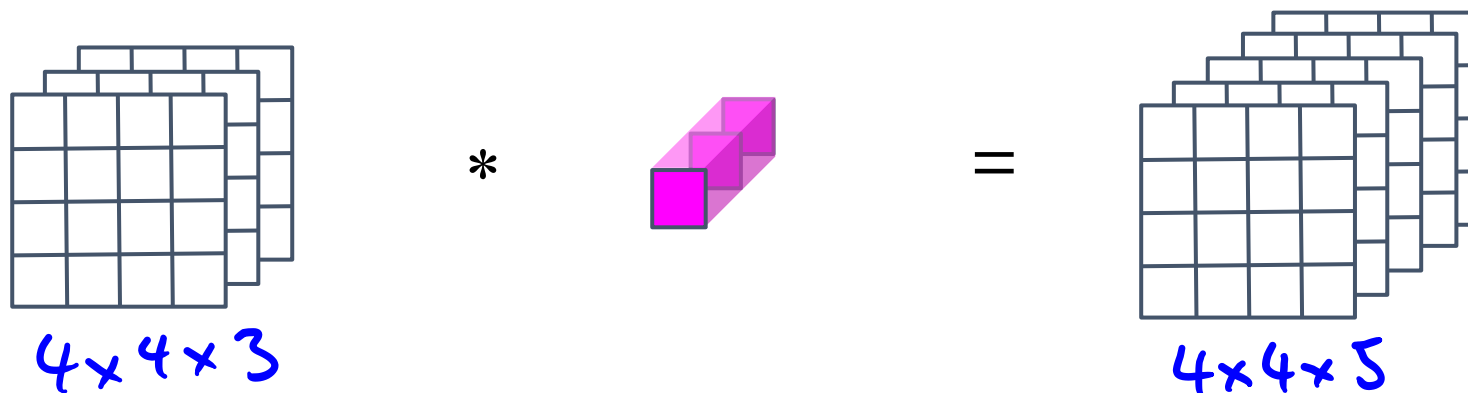
$432 = \underbrace{3 \times 3} \times \underbrace{4 \times 4 \times 3}$

# Depthwise Separable Convolution

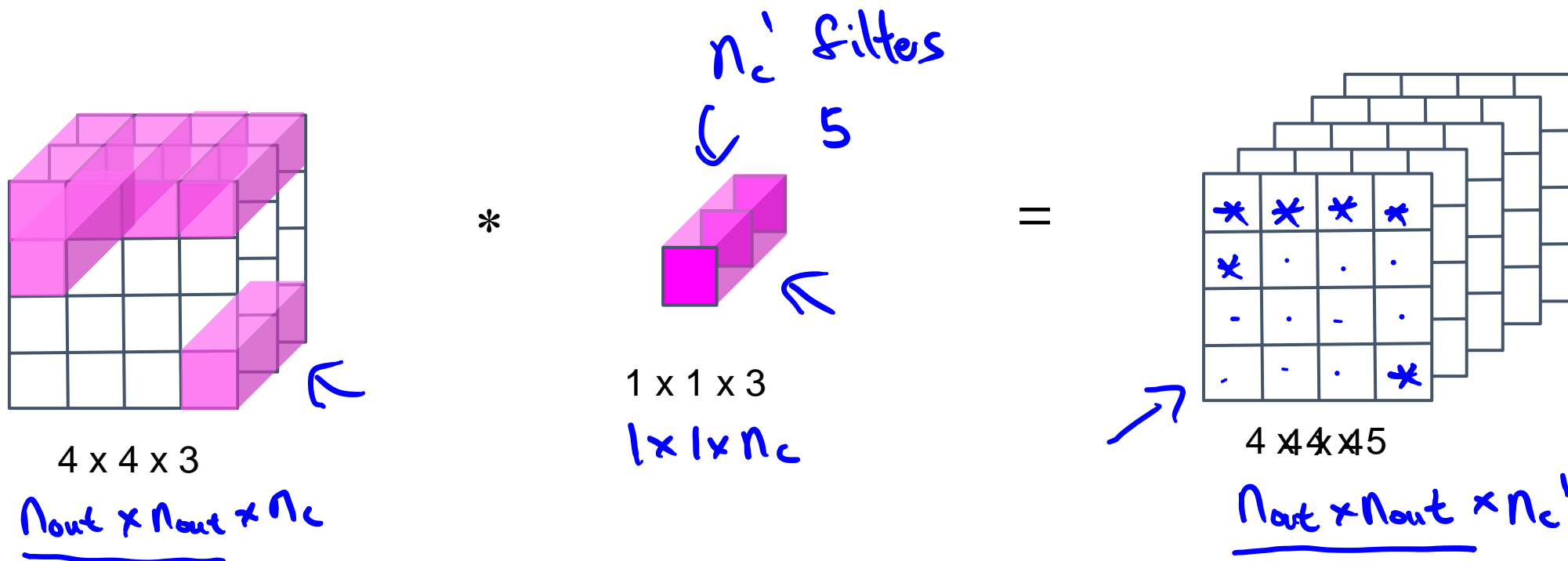
Depthwise Convolution



Pointwise Convolution



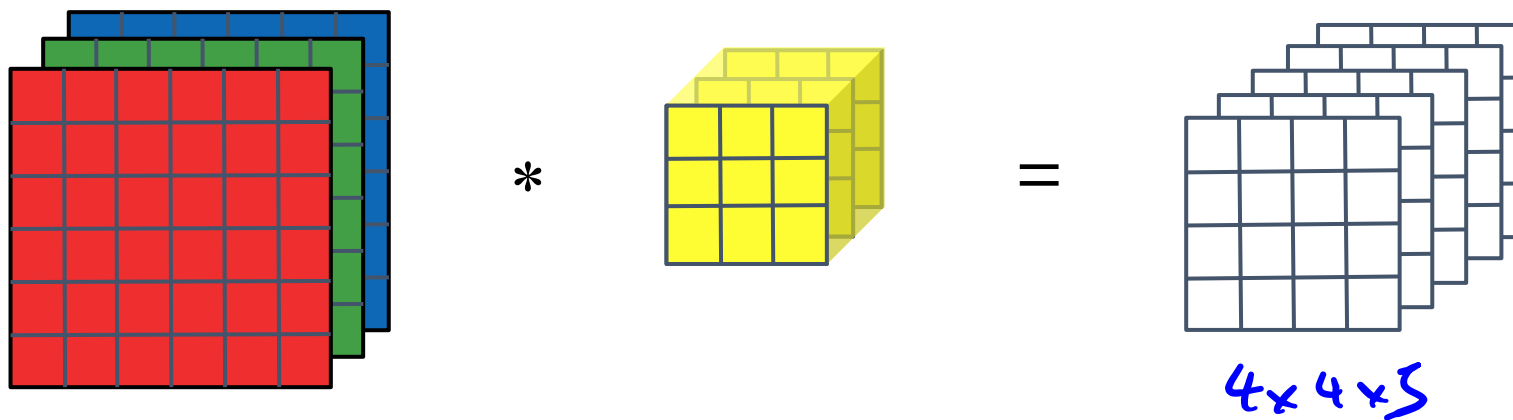
# Pointwise Convolution



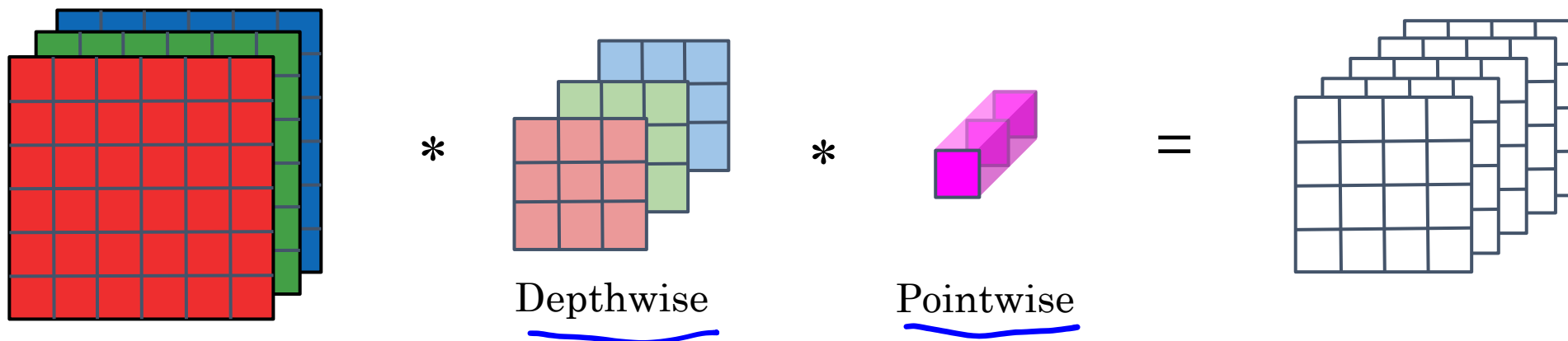
Computational cost = #filter params  $\times$  # filter positions  $\times$  # of filters  
 $240 = 1 \times 1 \times 3 \times 4 \times 4 \times 5$

# Depthwise Separable Convolution

Normal Convolution



Depthwise Separable Convolution





# Cost Summary

Cost of normal convolution 2160

Cost of depthwise separable convolution

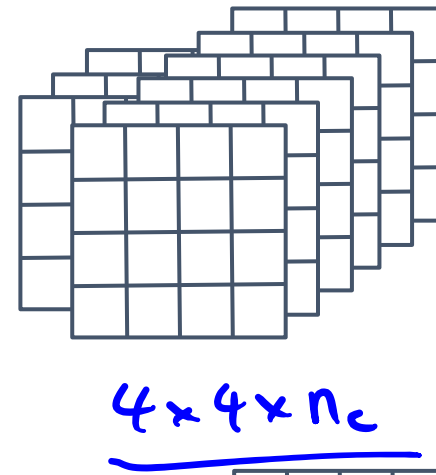
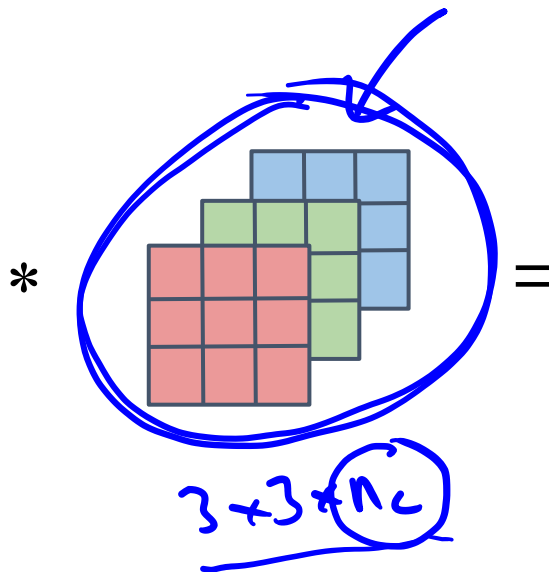
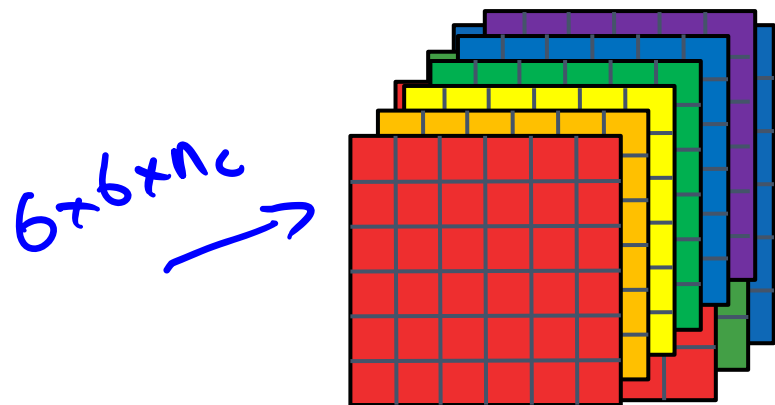
$$\begin{array}{l} \text{depthwise} + \text{pointwise} \\ 432 + 240 = 672 \end{array}$$

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

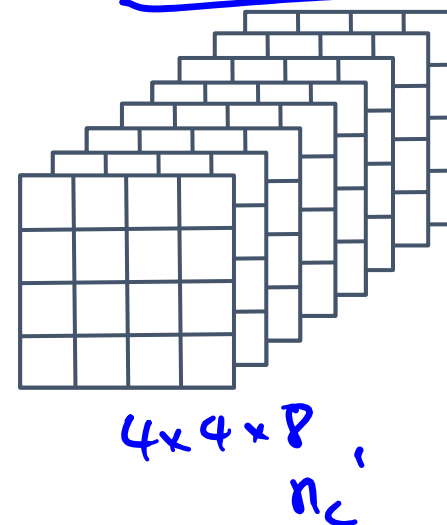
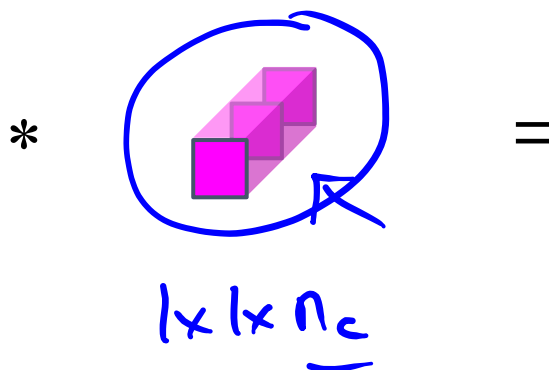
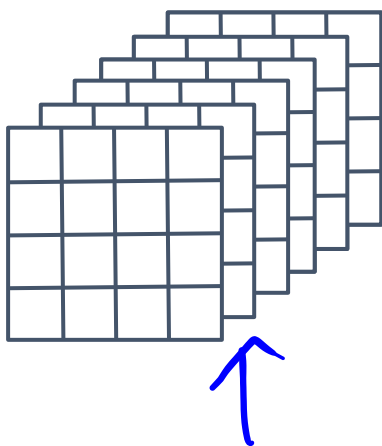
$$\begin{aligned} &= \frac{1}{n_c} + \frac{1}{f^2} \\ &= \frac{1}{5} + \frac{1}{9} \\ &= \frac{1}{512} + \frac{1}{3^2} \\ &\sim 10 \text{ times cheaper} \end{aligned}$$

# Depthwise Separable Convolution

Depthwise Convolution



Pointwise Convolution





deeplearning.ai

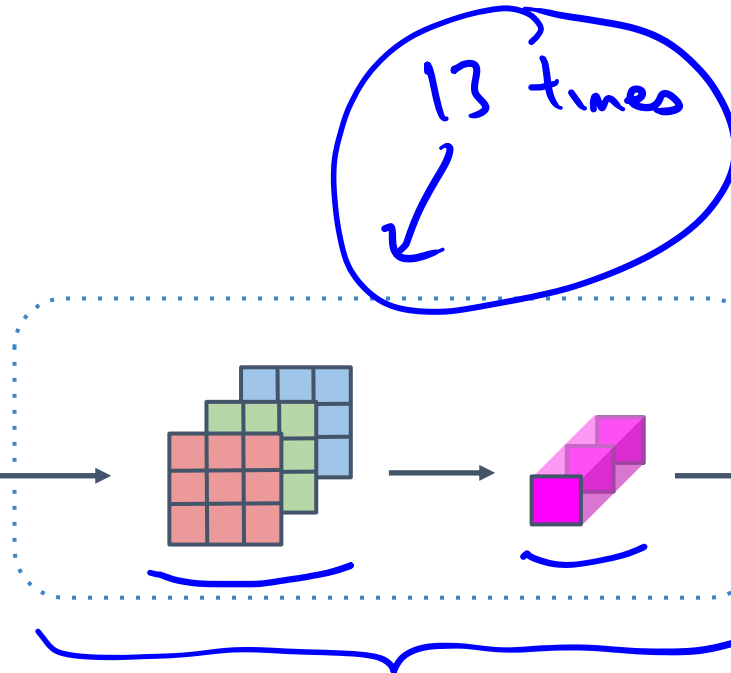
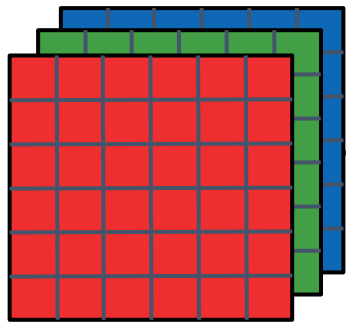
# Convolutional Neural Networks

---

## MobileNet Architecture

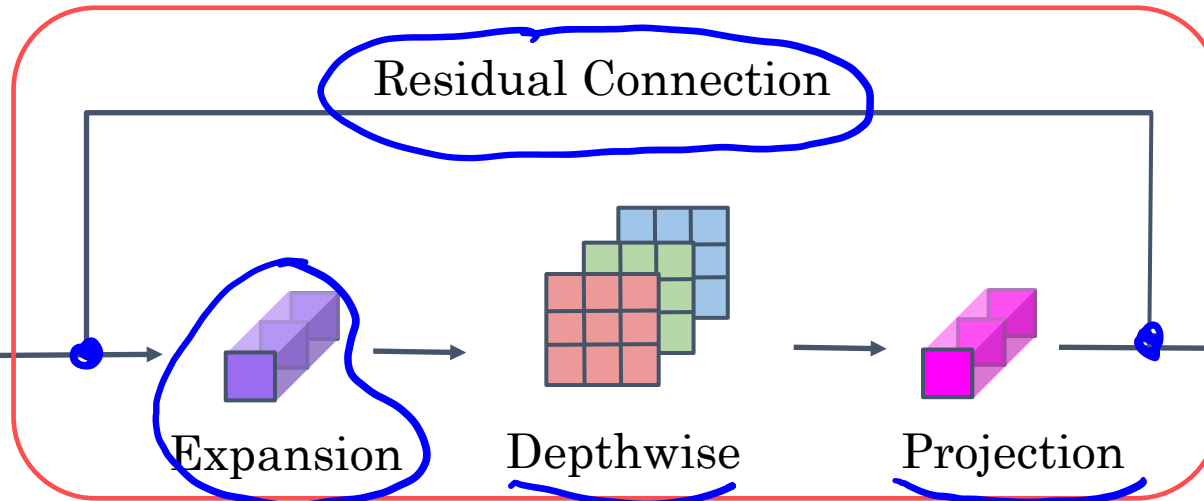
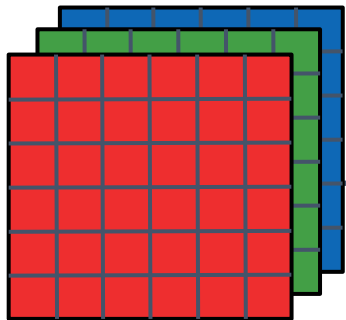
# MobileNet

MobileNet v1



Pool, FC, SOFTMAX

MobileNet v2

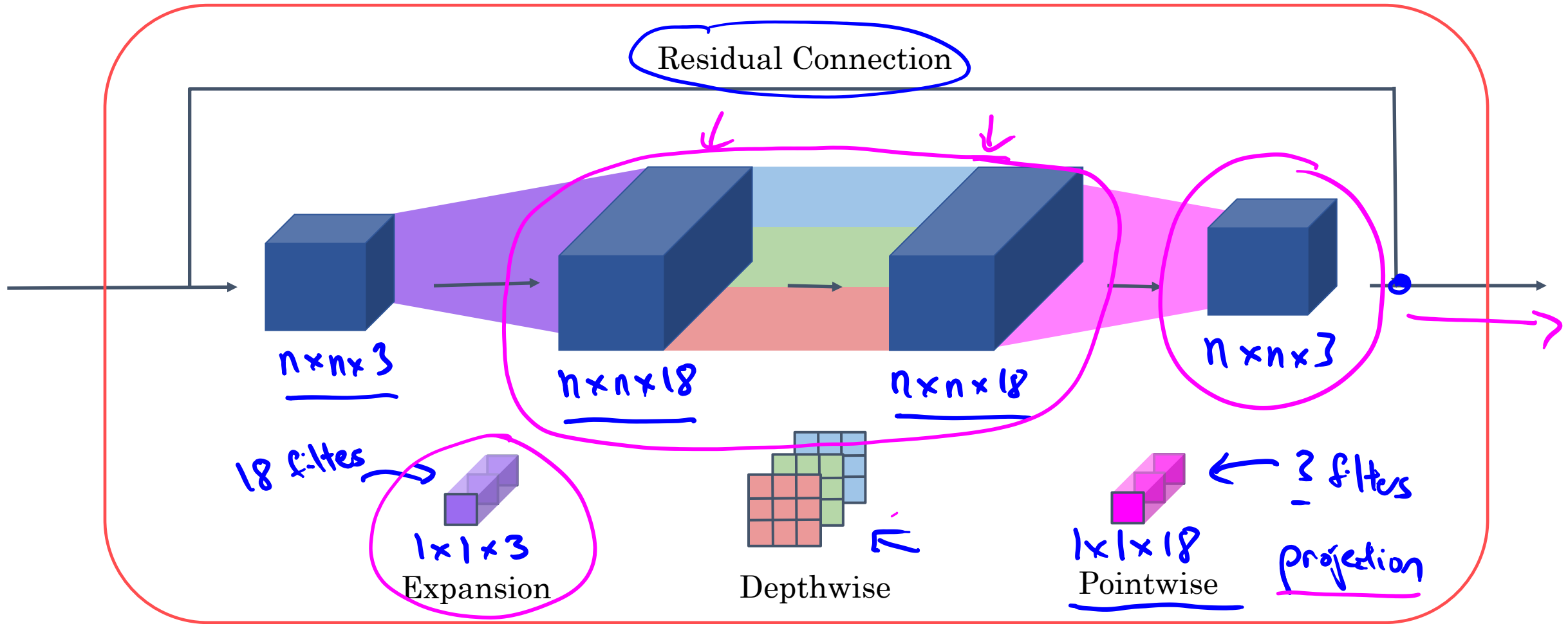


17 times

Pool, FC, SOFTMAX

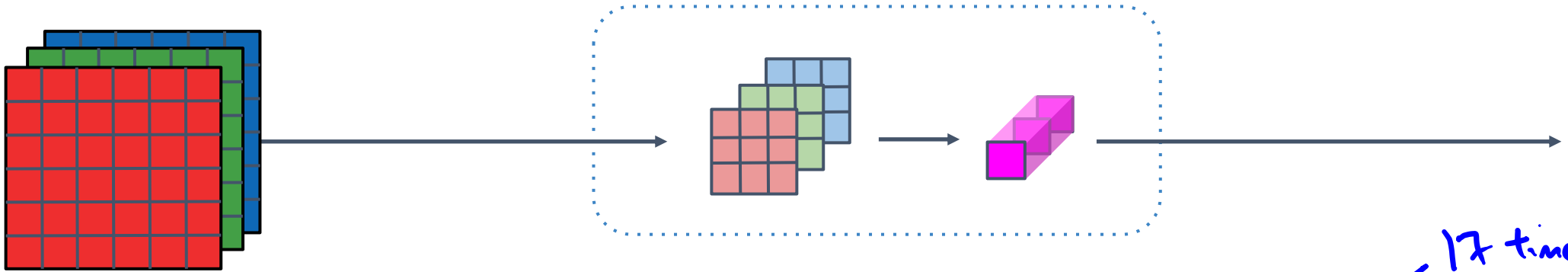
Bottleneck

# MobileNet v2 Bottleneck

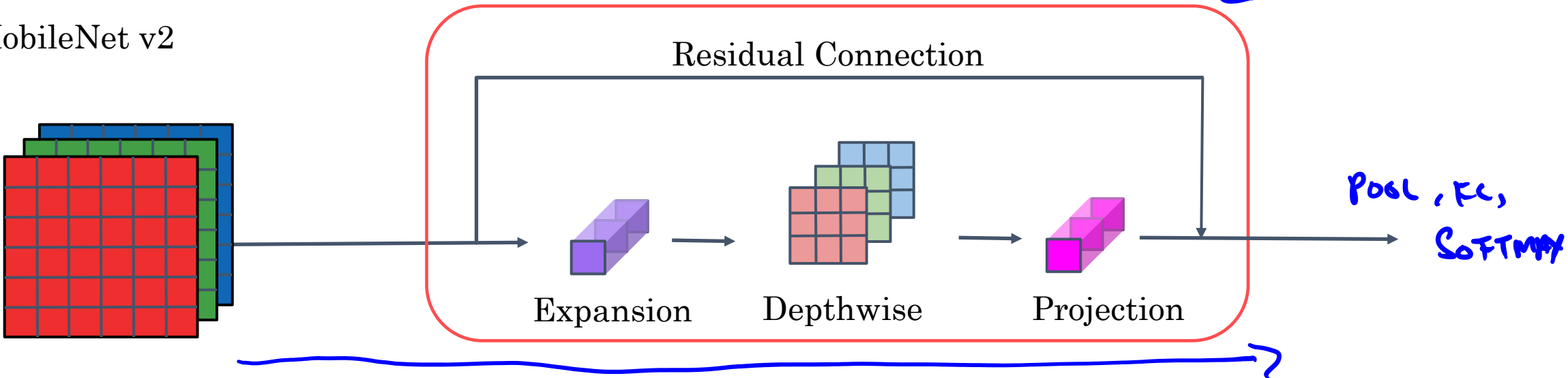


# MobileNet

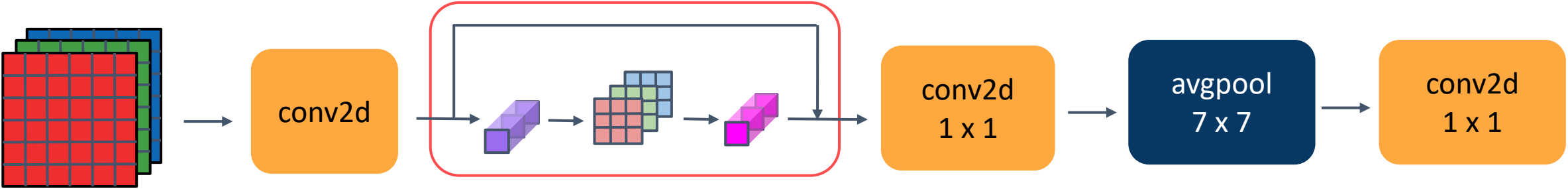
MobileNet v1



MobileNet v2



# MobileNet v2 Full Architecture





deeplearning.ai

# Convolutional Neural Networks

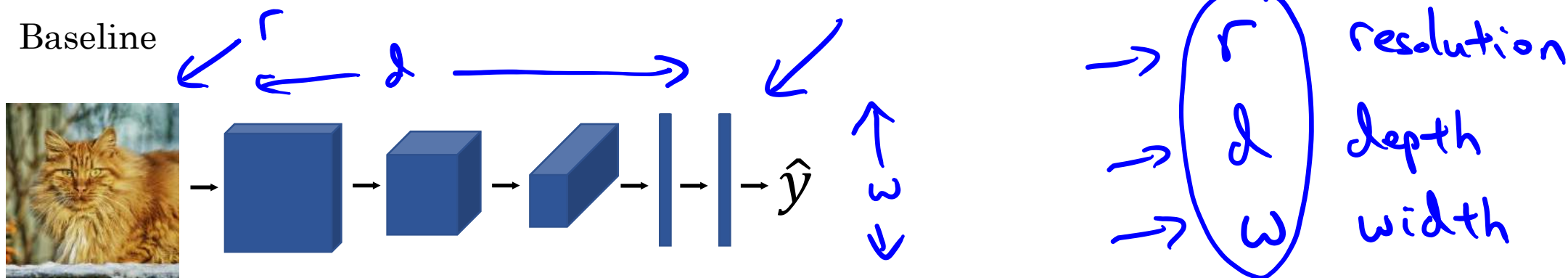
---

## EfficientNet

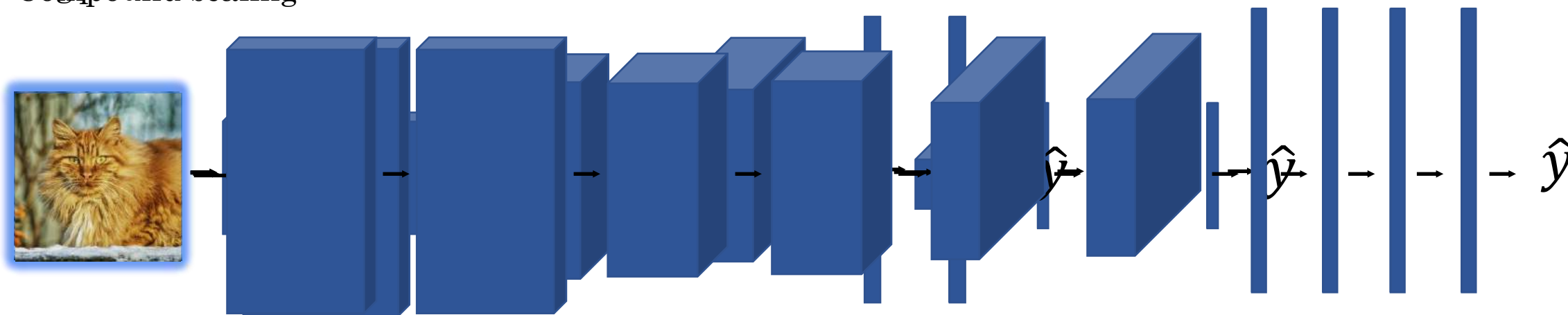


# EfficientNet

Baseline



Higher Resolution  
Computing





deeplearning.ai

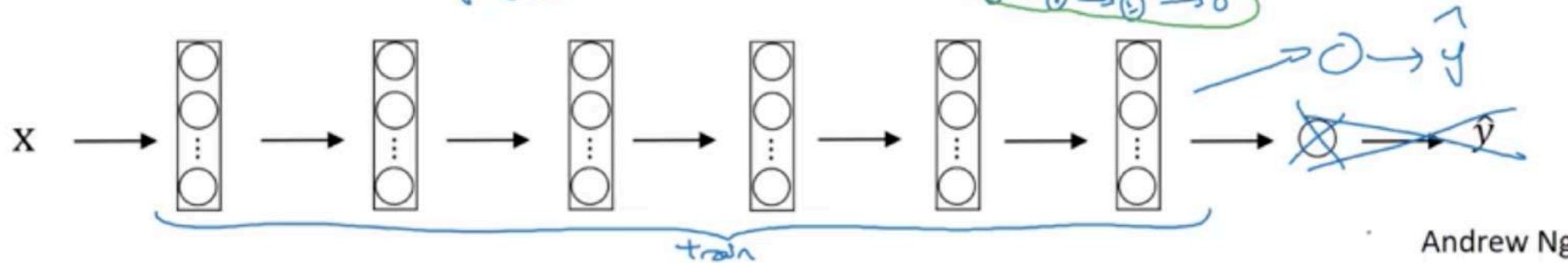
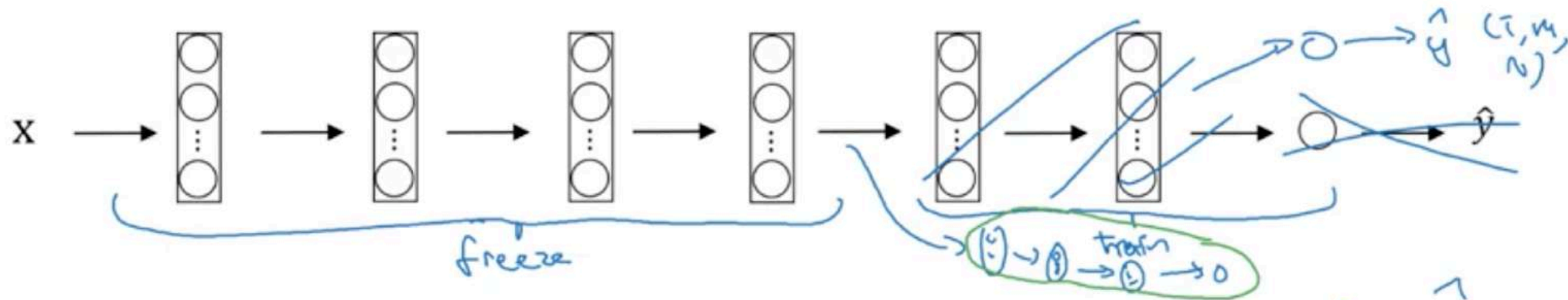
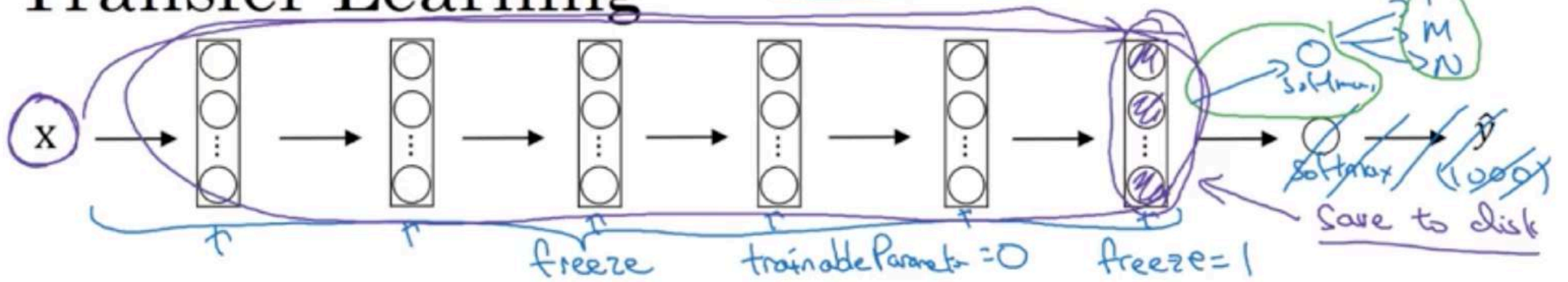
Practical advice for  
using ConvNets

---

**Transfer Learning**

# Transfer Learning

😊 Tigger     😞 Misty     Neither





deeplearning.ai

Practical advice for  
using ConvNets

---

**Data augmentation**

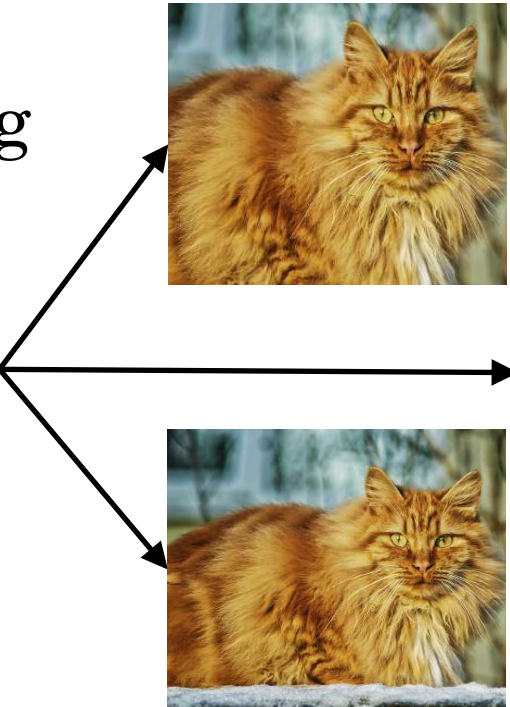
# Common augmentation method

Mirroring



y

Random Cropping



- Rotation
  - Shearing
  - Local warping
  - ...
- 
- A hand-drawn blue parallelogram, representing a shearing transformation.

# Color shifting



↳

R G B  
↓ ↓ ↓  
+20, -20, +20



-20, +20, +20



+5, 0, +50



Advanced:

PCA

[ml-class.org](http://ml-class.org)

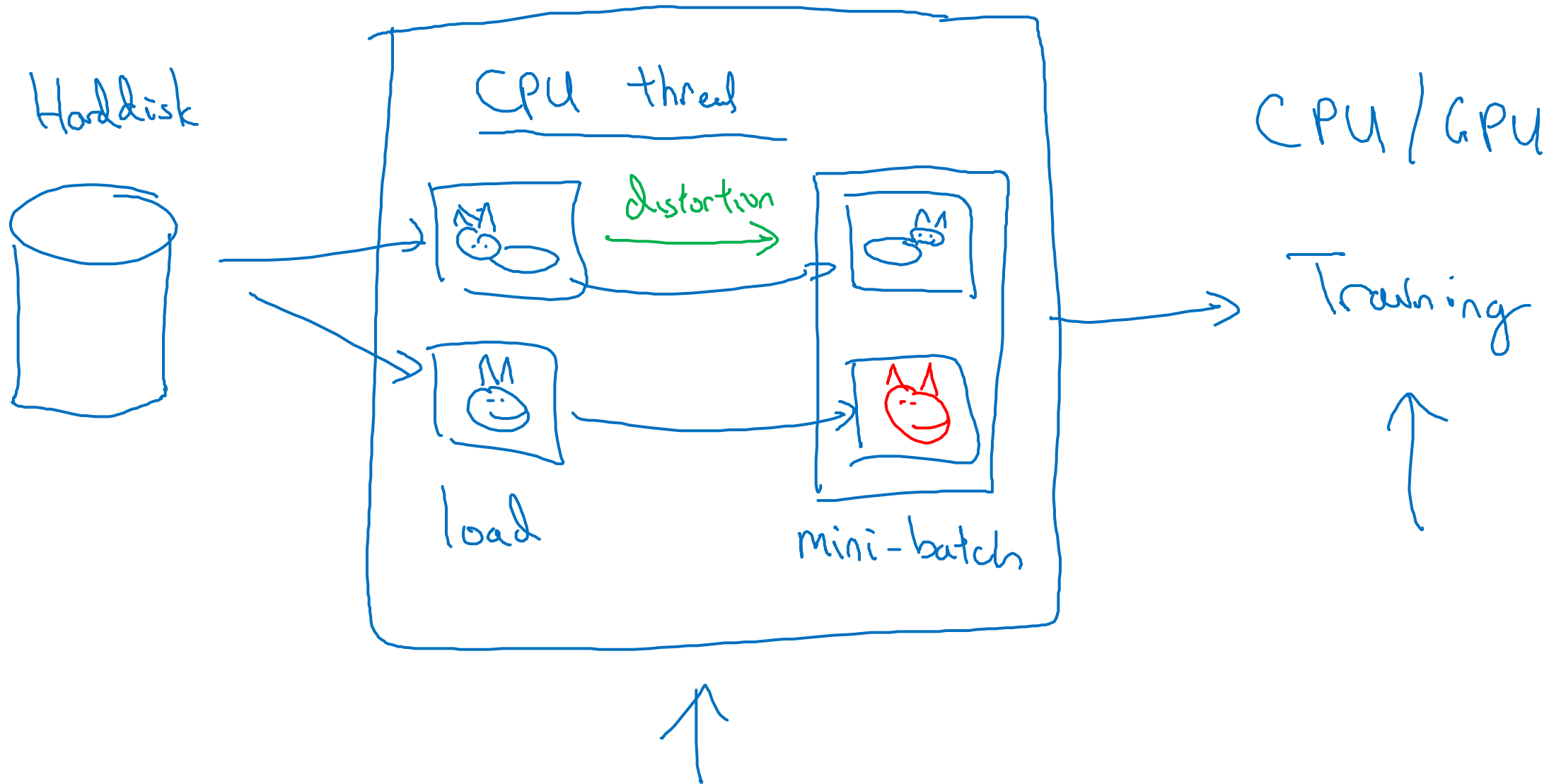
[ AlexNet paper

["PCA color augmentation."]

R B

G

# Implementing distortions during training





deeplearning.ai

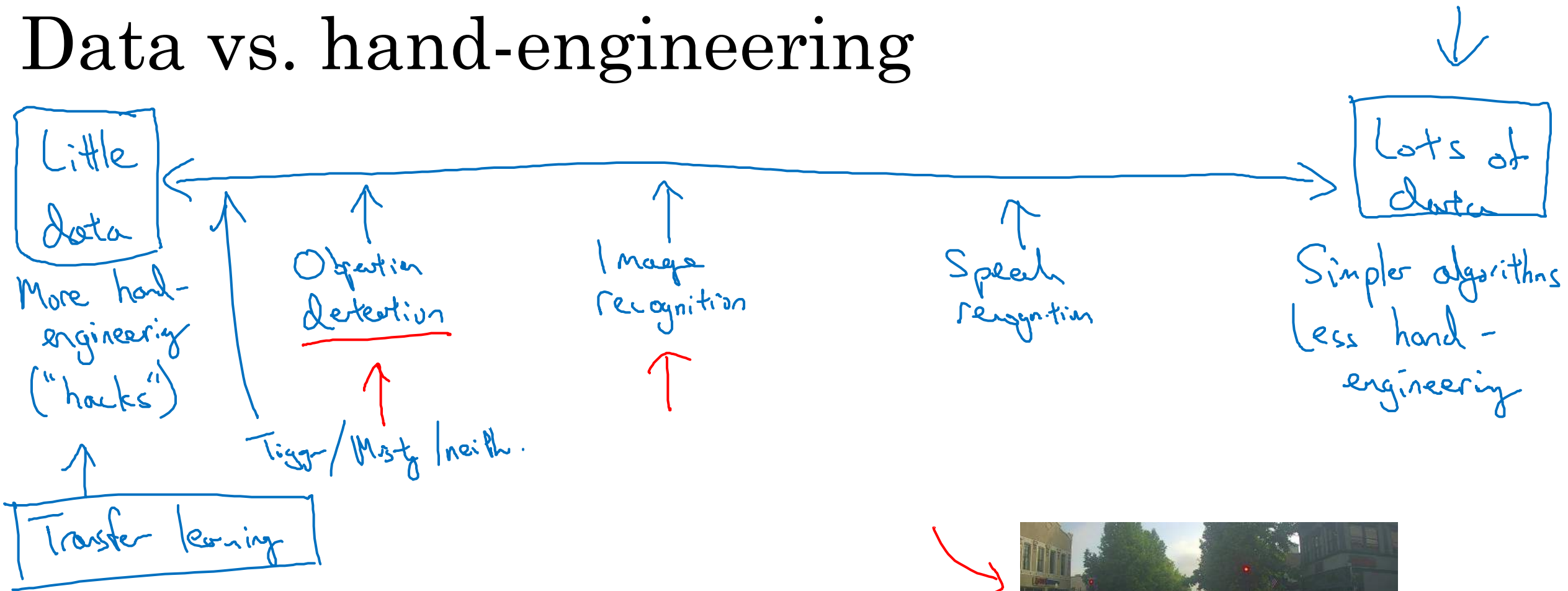
Practical advice for  
using ConvNets

---

The state of  
computer vision



# Data vs. hand-engineering



Two sources of knowledge

- • Labeled data  $(x, y)$
- • Hand engineered features/network architecture/other components



# Tips for doing well on benchmarks/winning competitions

## Ensembling

3-15 networks

→  $\hat{y}$

- Train several networks independently and average their outputs

## Multi-crop at test time

- Run classifier on multiple versions of test images and average results

10-crop



1

+



4

+



1

+



4

# Use open source code

- Use architectures of networks published in the literature
- Use open source implementations if possible
- Use pretrained models and fine-tune on your dataset