



deeplearning.ai

# Case Studies

---

Why look at  
case studies?

# Outline

Classic networks:

- LeNet-5 ←
- AlexNet ←
- VGG ←

ResNet (152)

Inception



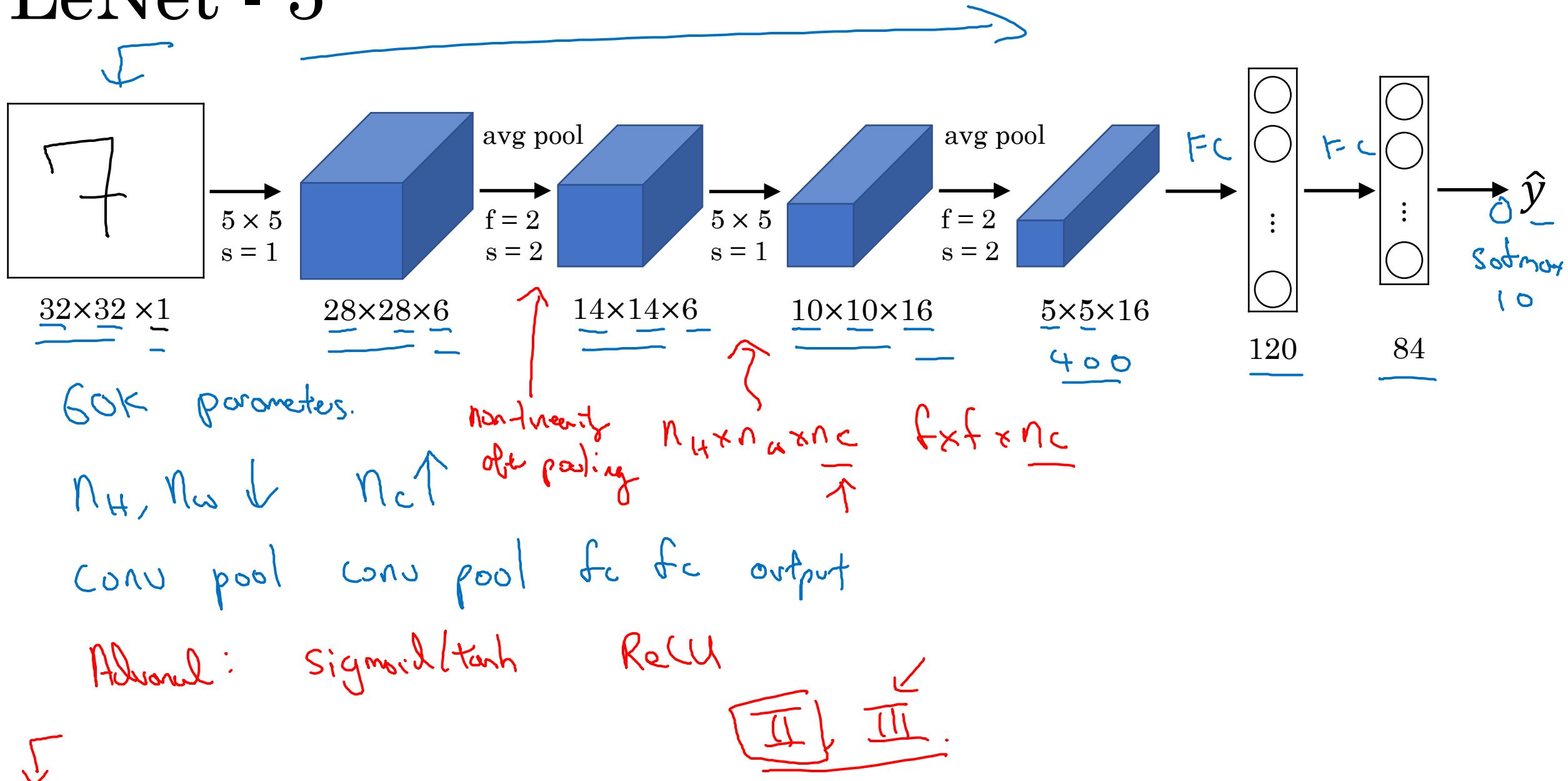
deeplearning.ai

# Case Studies

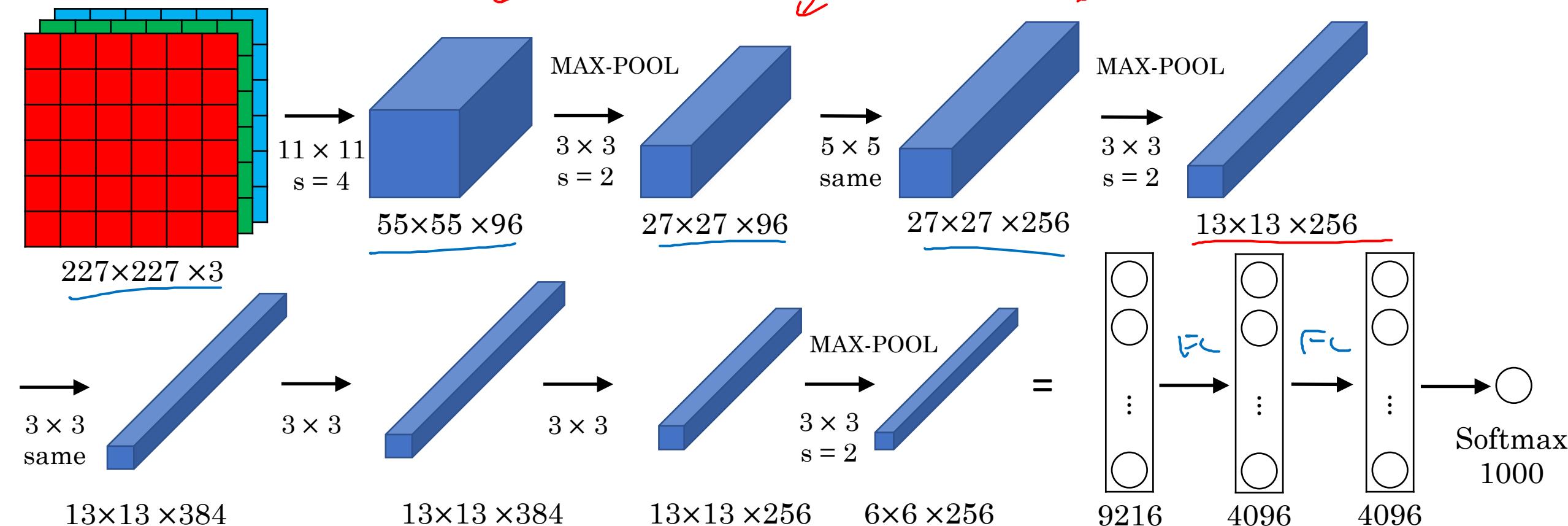
---

## Classic networks

# LeNet - 5



# AlexNet

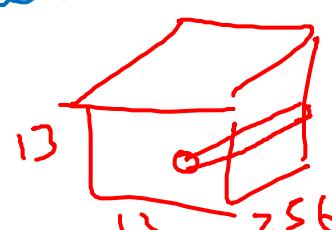


- Similar to LeNet, but much bigger.

- ReLU

- Multiple GPUs.

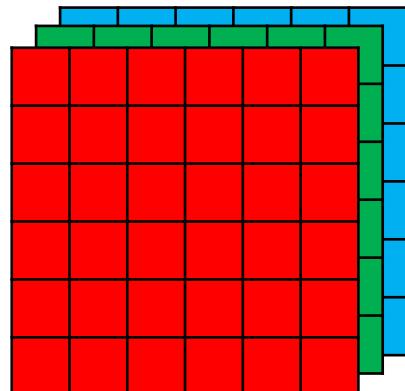
- Local Response Normalization (LRN)



$\sim 60M$  parameters

# VGG - 16

CONV =  $3 \times 3$  filter,  $s = 1$ , same



$224 \times 224$

[CONV 64]  
 $\times 2$

$224 \times 224 \times 64$  → POOL →  $112 \times 112 \times 64$

[CONV 128]  
 $\times 2$

$112 \times 112 \times 128$  → POOL →  $56 \times 56 \times 128$

$224 \times 224$

$56 \times 56 \times 256$  → POOL →  $28 \times 28 \times 256$  → [CONV 512]  
 $\times 3$  →  $28 \times 28 \times 512$  → POOL →  $14 \times 14 \times 512$

$14 \times 14 \times 512$  → POOL →  $7 \times 7 \times 512$  → FC 4096 → FC 4096 → Softmax 1000  
[CONV 512]  
 $\times 3$

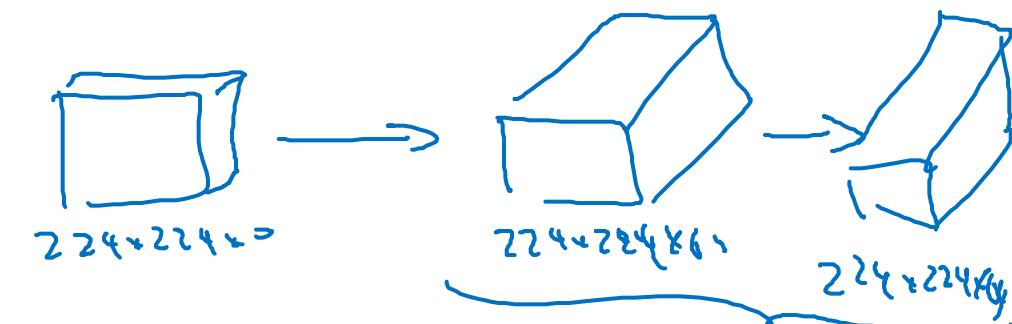
$n_H, n_W \downarrow$

$n_C \uparrow$

$\sim 38M$

# VGG-19

MAX-POOL =  $2 \times 2$ ,  $s = 2$



$112 \times 112 \times 128$  → POOL →  $56 \times 56 \times 128$



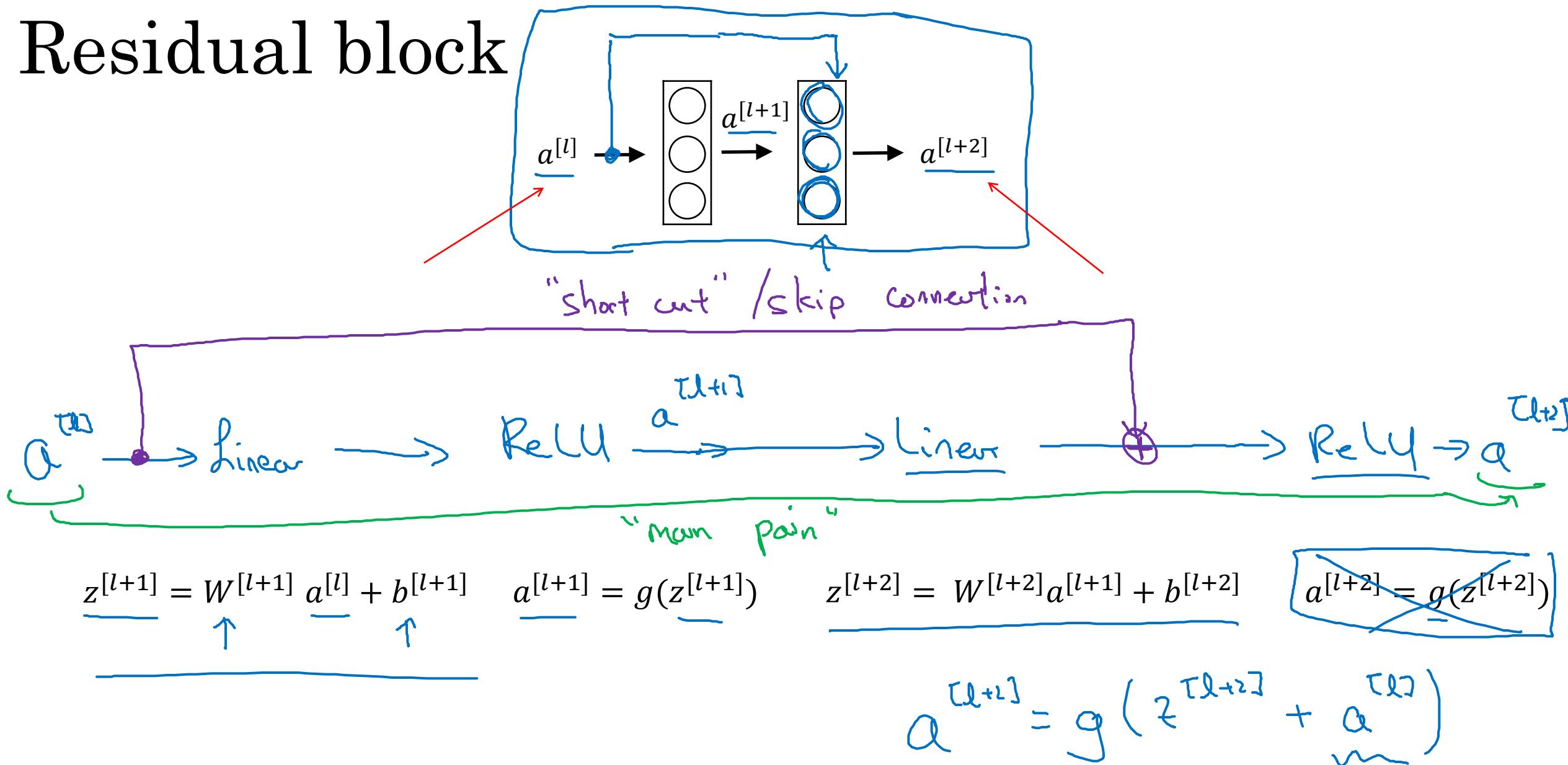
deeplearning.ai

## Case Studies

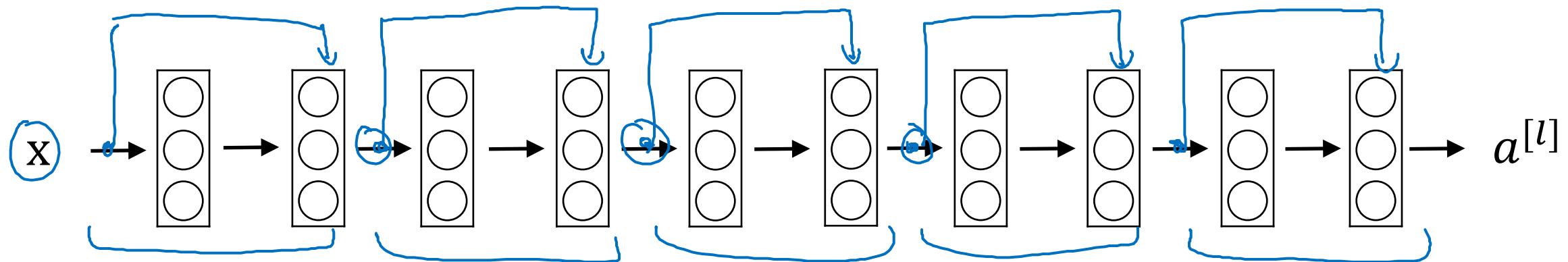
---

# Residual Networks (ResNets)

# Residual block

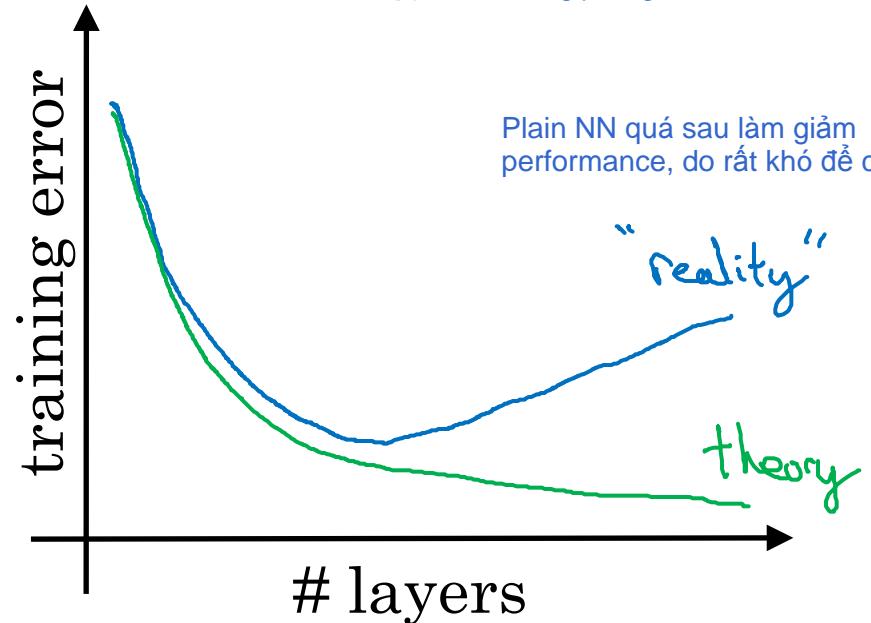


# Residual Network



Plain mạng phẳng

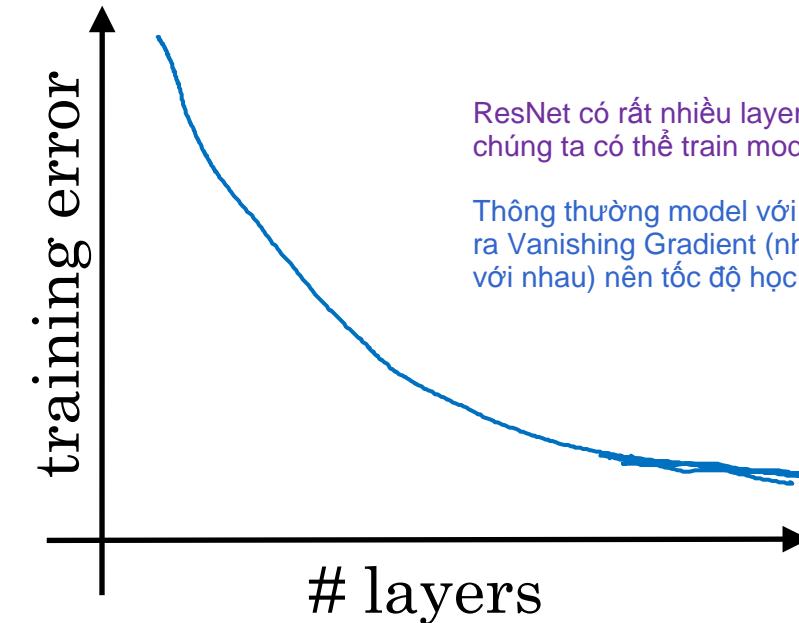
Plain NN quá sau làm giảm performance, do rất khó để chọn parameter



ResNet

ResNet có rất nhiều layers, thêm residual blocks chúng ta có thể train model tới hàng nghìn NN.

Thông thường model với rất nhiều layers hay xảy ra Vanishing Gradient (nhiều đạo hàm nhỏ nhân với nhau) nên tốc độ học sẽ rất chậm.





deeplearning.ai

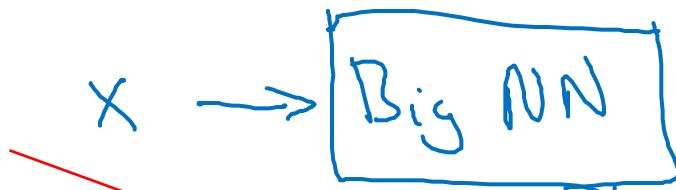
# Case Studies

---

## Why ResNets work

# Why do residual networks work?

Mạng này  
đã lớn rồi



Trường hợp  $z^{[l+2]}$  hay  $a^{[l+2]}$  khác dimension với  $a^{[l]}$   
ta có thể thêm matrix  $W_s$  có  
dimension như  
hình bên cạnh

ReLU.

$a^{[l+2]}$   
 $256$

$$a \geq 0$$

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

$$= g(W^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

$$= g(W^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

$$\text{If } W^{[l+2]} = 0, b^{[l+2]} = 0$$

Identify  
easy  
function is  
for Residual block  
to learn!

Chú ý:  $z^{[l+2]}$  và  $a^{[l]}$  phải có  
cùng dimensions, khi  
implement người ta hay dùng  
padding = 'same' để bảo toàn  
dimensions (residual block hay  
dùng cho Conv layers)

$$R^{256 \times 128} = g(a^{[l]}) = a^{[l]}$$

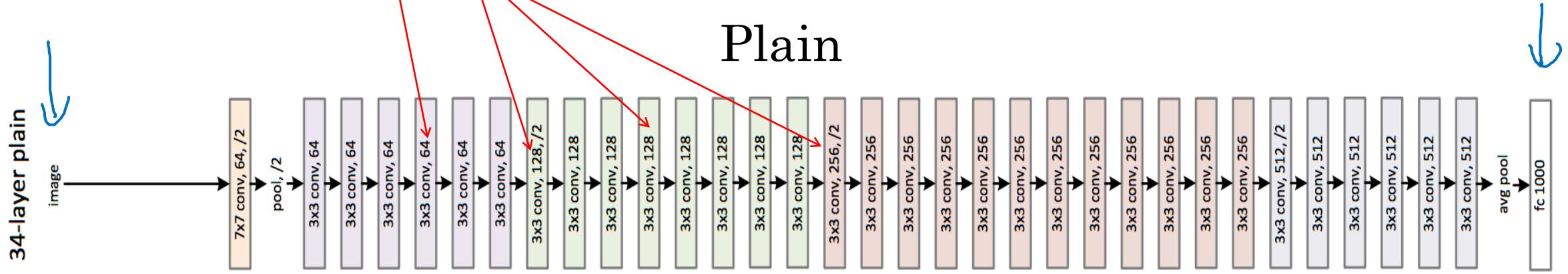
Giả sử  $W_{[l+1]}$  và  $b_{[l+1]}$  đều bằng 0 (rất dễ dàng để học)  
thì  $a_{[l+2]} = a_{[l]}$ . Do đó ResNet rất dễ học identity function, ở  
kia viết được 2 cái bằng nhau vì mình dùng ReLU (thêm vào thì output vẫn  
vậy khi các weights và bias = 0).

Residual blocks ko làm ảnh hưởng đến performance khi tăng  
layers (do residual block).

Không những vậy residual block còn giúp tăng performance nữa vì một số layers thêm vào học được các feature hữu ích.

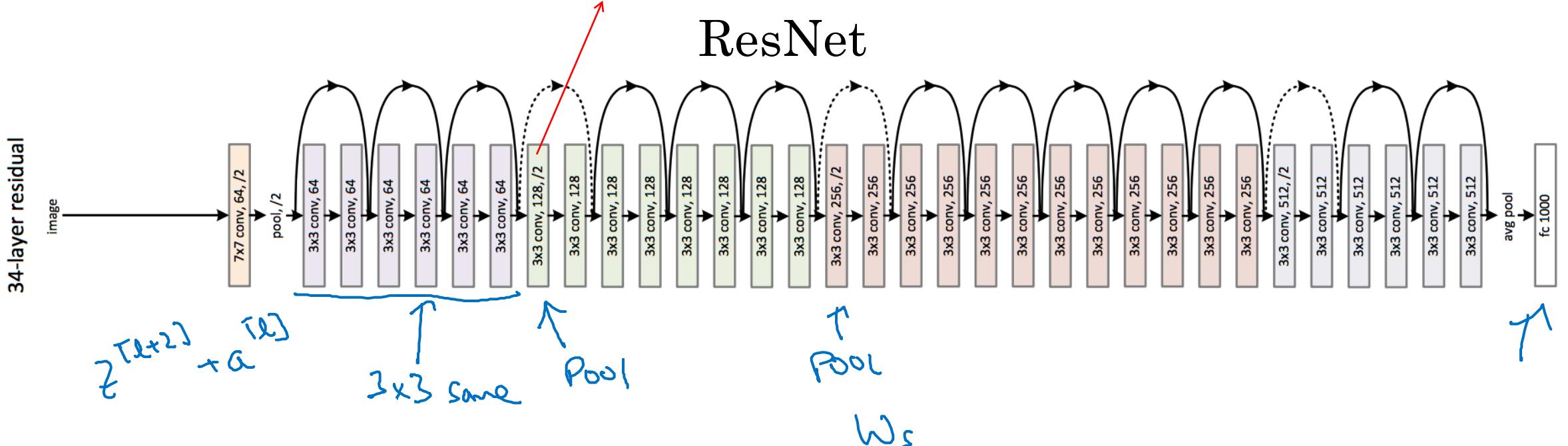
Andrew Ng

# ResNet



/2 chỉ dimension giảm đi một nửa, chỗ này là Pooling layer, số bên cạnh là channels

Nhận thấy dimension của  $z^{[l+2]}$  và  $a^{[l]}$  khác nhau nên cần thêm matrix  $W_s$  như trình bày ở trang trước.





deeplearning.ai

## Case Studies

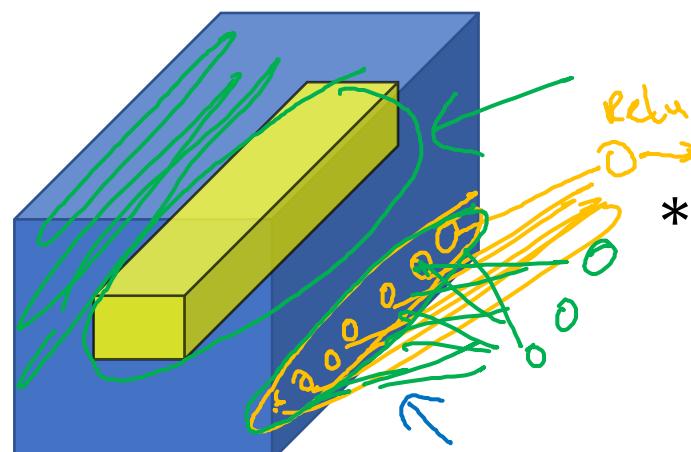
---

# 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

$6 \times 6$



$6 \times 6 \times 32$

[Lin et al., 2013. Network in network]

Một filter đặt vào tensor ban đầu rồi trượt (filter có số channels bằng số channel của tensor ban đầu, sau đó nhận element-wise rồi cộng lại, sau đó áp dụng Activation function ta nhận được 1 giá trị tương ứng 1 vị trí.

Làm tương tự cho các vị trí khác ta nhận được spatial dimension như hình bên

Chúng ta lại có # filters, lại làm tương tự cho từng filter, cuối cùng nhận được tensor có số channels = # filters

\*

2

=



32  $\rightarrow$  # filters.

$n_c^{[l+1]}$

=

ReLU

Network  $\hookrightarrow$  Network

$1 \times 1 \times 32$

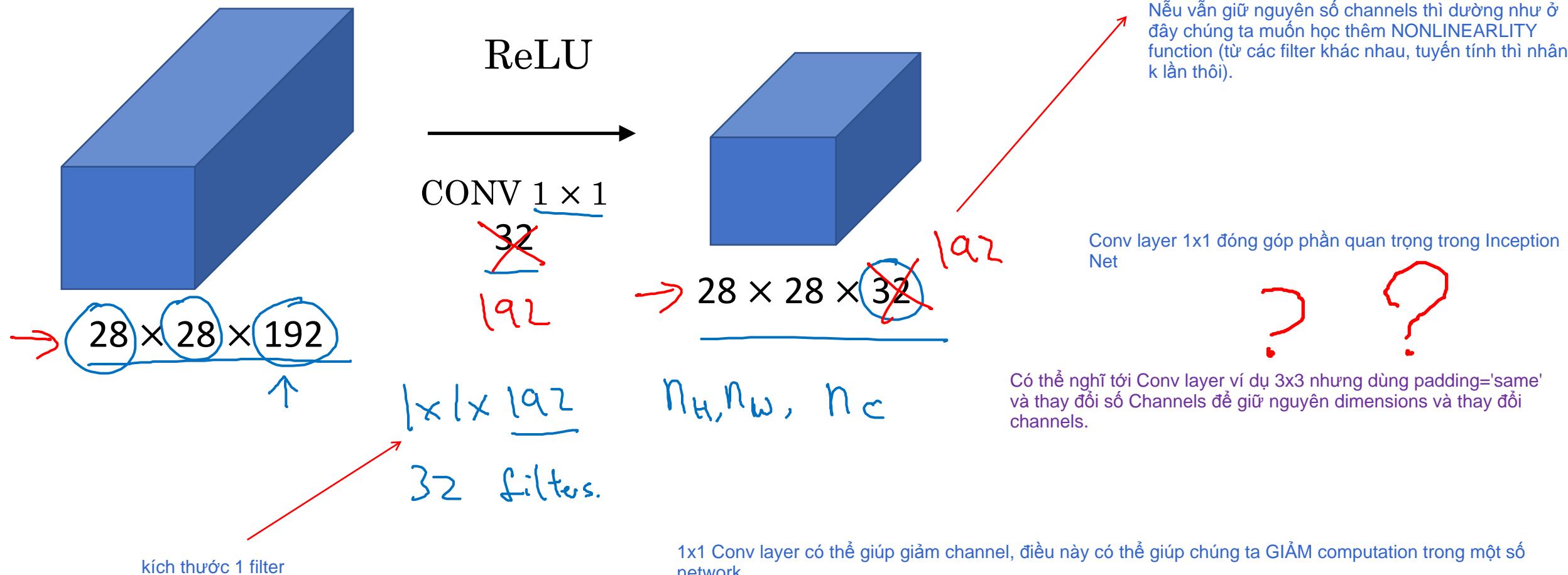
2	4	6	...	

Giữ nguyên spatial dimension nhưng thay đổi số CHANNELs tùy ý.


$6 \times 6 \times \# filters$

Andrew Ng

# Using $1 \times 1$ convolutions





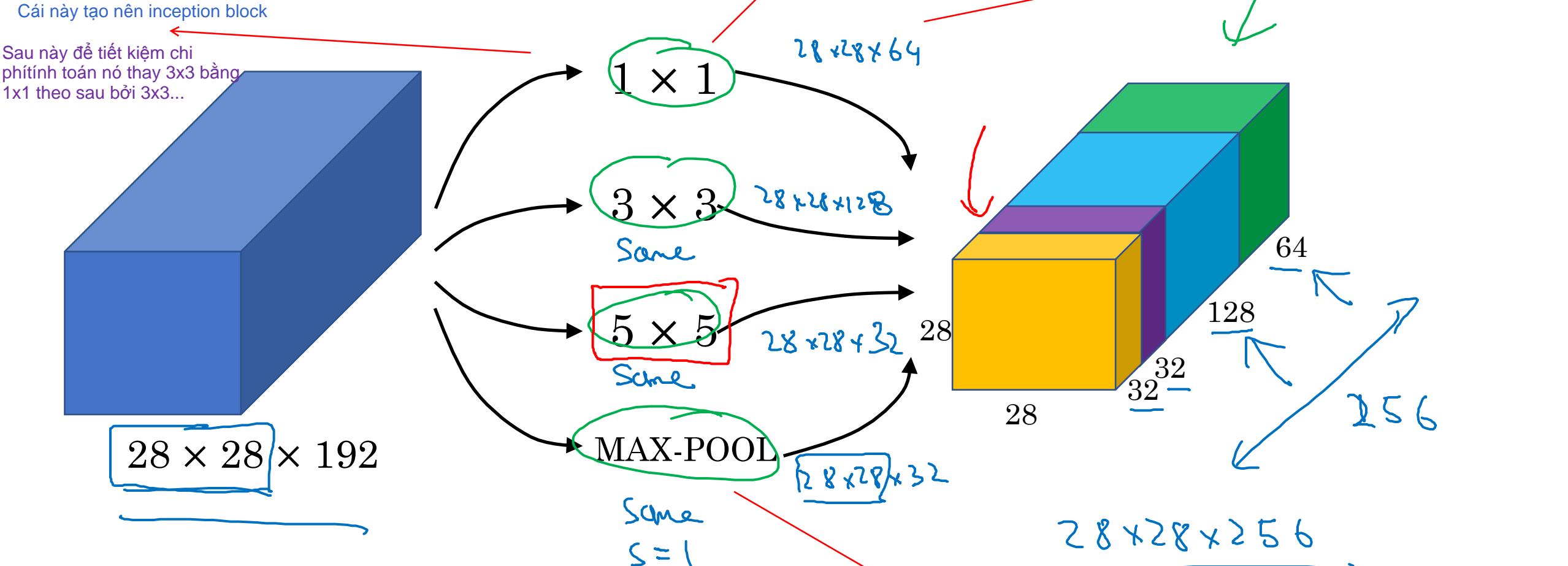
deeplearning.ai

## Case Studies

---

Inception network  
motivation

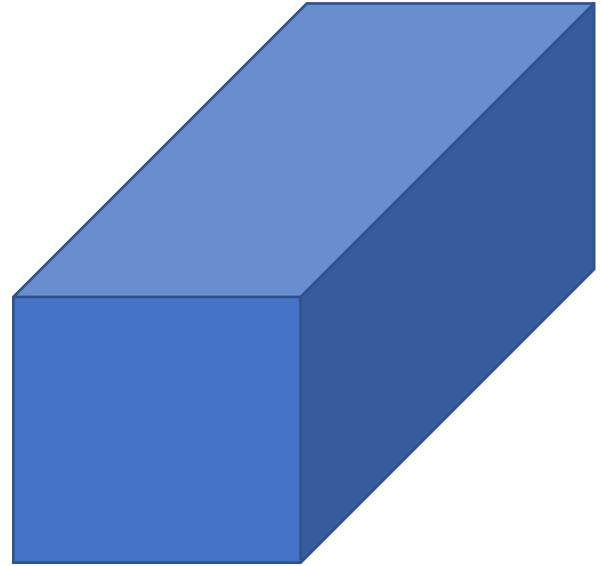
# Motivation for inception network



Ý tưởng: thay vì xem Conv layer có kích thước nào phù hợp thì sao ko thử kết hợp tất cả lại.

Chỗ này dùng cả Max-pooling, bình thường spatial dimension hay giảm đi 2 lần, tuy nhiên ở đây phải dùng padding để giữ nguyên spatial dimension

# The problem of computational cost



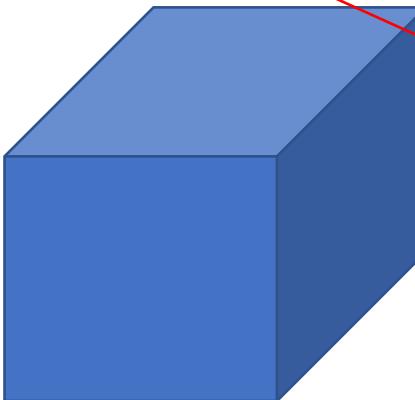
$28 \times 28 \times 192$

CONV  
 $5 \times 5$ ,  
same,  
32

32 Filters.

$28 \times 28 \times 32$

Cần phải thực hiện phép  
nhân element-wise như này  
phép cộng lại 1 lần ko tính



Cùng phân tích computation cost với Conv layer  $5 \times 5$

filter size  $5 \times 5 \times 192$  và có 32 filters. Đầu ra là  $28 \times 28 \times 32$

Xét cho 1 filter:

- 1 vị trí cần  $5 \times 5 \times 192$ . Có tổng cộng  $28 \times 28$  vị trí do đó là  $5 \times 5 \times 192 \times 28 \times 28$

Chúng ta có 32 filters nên tổng số phép tính là  $5 \times 5 \times 192 \times 28 \times 28 \times 32$

$$\frac{28 \times 28 \times 32 \times 5 \times 5 \times 192}{\text{filters one } \underline{\underline{5 \times 5 \times 192}}} = 120M.$$

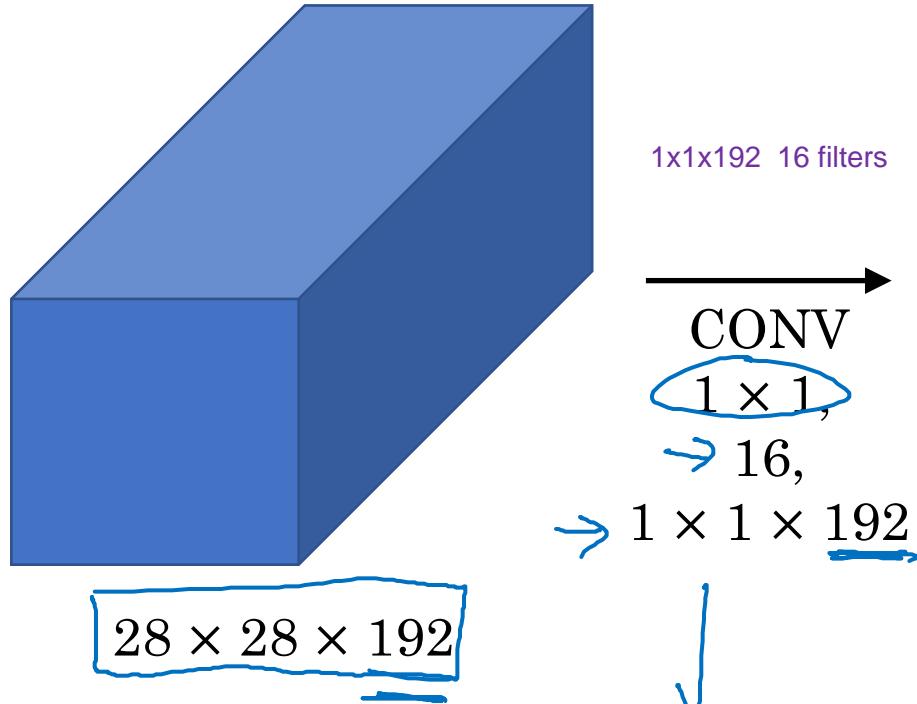
Dùng Conv layer  $5 \times 5$  trực tiếp sẽ có computation cost lớn hơn so với dùng Conv layer  $1 \times 1$  + Conv layer  $5 \times 5$

Tuy bottle layer giảm representation nhưng nó không ảnh hưởng quá nhiều đến performance mà tiết kiệm được nhiều chi phí tính toán

# Using $1 \times 1$ convolution

Nếu dùng trực tiếp Conv layer  $5 \times 5 \times 192$  với 32 filters ở trên mất khoảng 120 M phép tính

Nếu dùng Conv layer  $1 \times 1$  sau đó là Conv layer  $5 \times 5$  thì số phép tính sẽ giảm đi nhiều còn khoảng 12.4 M



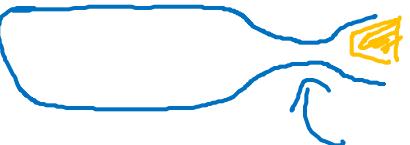
$$28 \times 28 \times 16 \times 192 = 2.4M$$

1 vị trí  $1 \times 1 \times 192$   
có  $28 \times 28$  vị trí trên output  $\Rightarrow 1 \times 1 \times 192 \times 28 \times 28$

Lại có 16 filters nên tổng cộng  
có  $1 \times 1 \times 192 \times 28 \times 28 \times 16 = 2.4M$

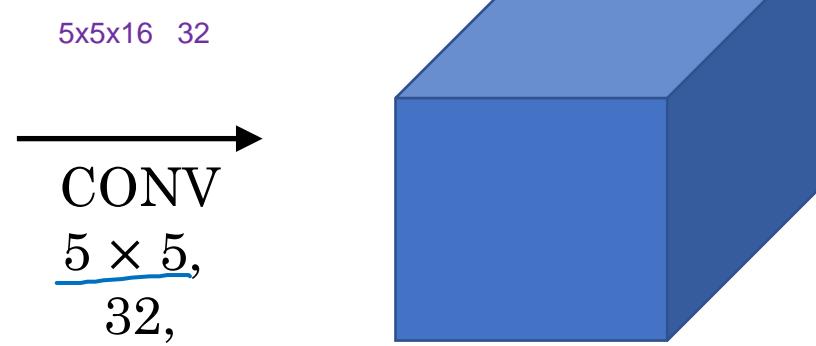
Nếu thay đổi số filters trung gian không phải là 16, ta sẽ có số phép tính là  $12.4 * \#filters / 16$

đôi khi gọi như vậy vì nó có kích thước nhỏ nhất, số channels của nó nhỏ nhất trong block



"bottlenecks layer"

Chỗ này có padding='same', cụ thể bao nhiêu thì phụ thuộc vào stride



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

1 vị trí  $5 \times 5 \times 16$   
có  $28 \times 28$  vị trí trên output  $\Rightarrow 5 \times 5 \times 16 \times 28 \times 28$

Lại có 32 filters nên tổng cộng  
có  $5 \times 5 \times 16 \times 28 \times 28 \times 32 = 10M$

120M



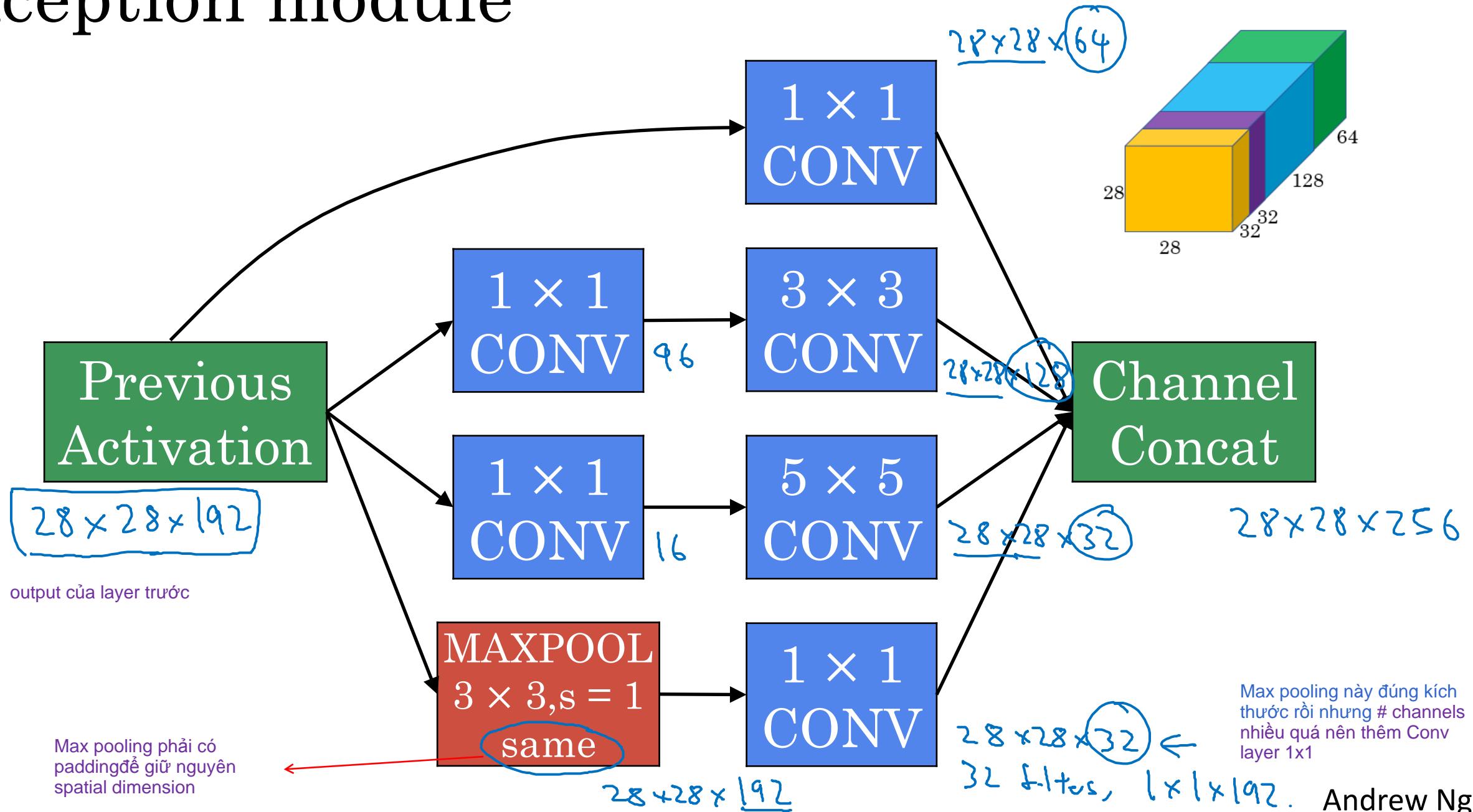
deeplearning.ai

## Case Studies

---

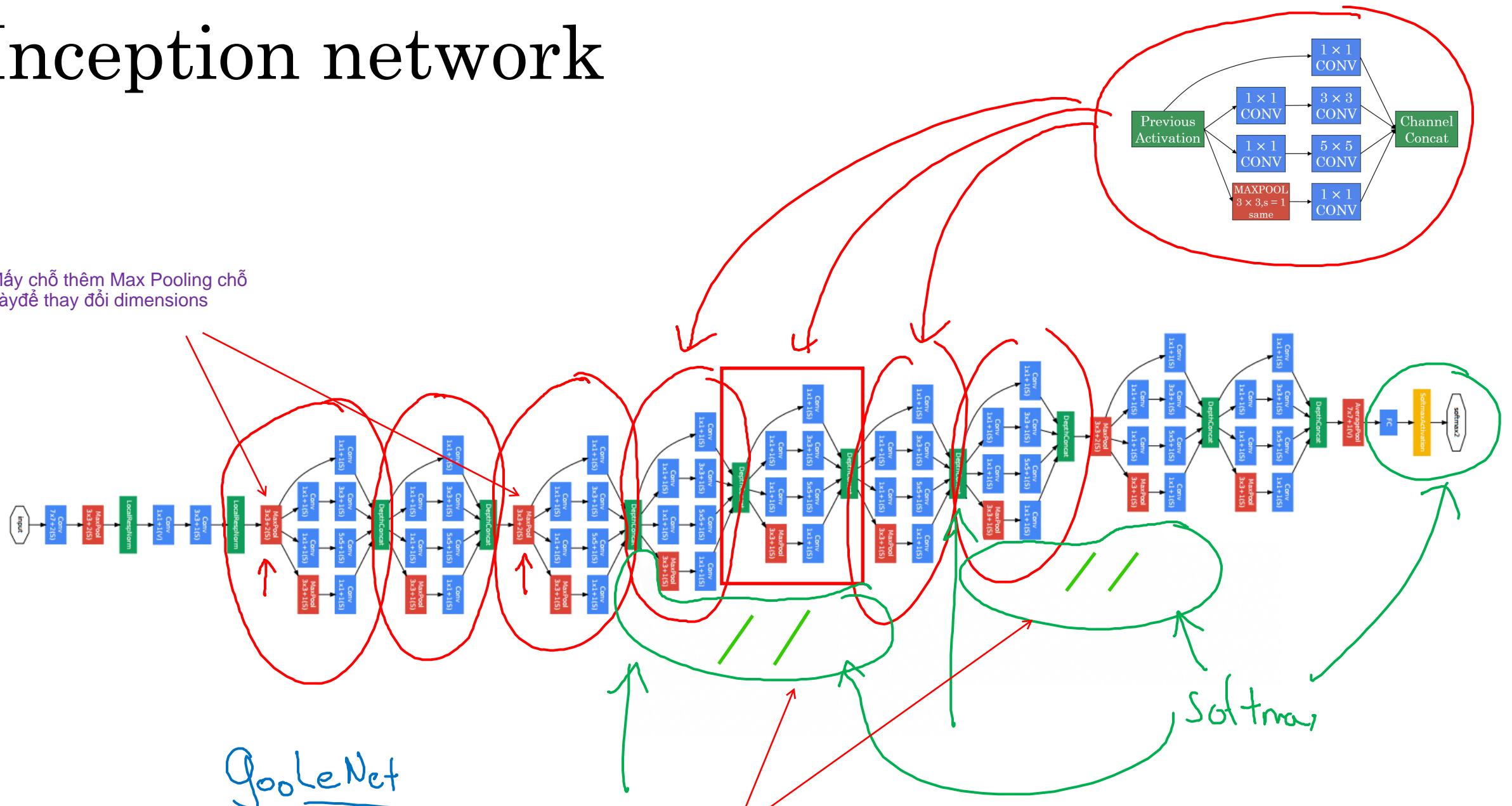
### Inception network

# Inception module



# Inception network

Máy chẽ thêm Max Pooling chẽ  
này để thay đổi dimensions



Ngoài output chính ra ở đây còn thêm 2 output nữa để dự đoán từ các intermediate layers.





deeplearning.ai

# Convolutional Neural Networks

---

## MobileNet

# Motivation for MobileNets

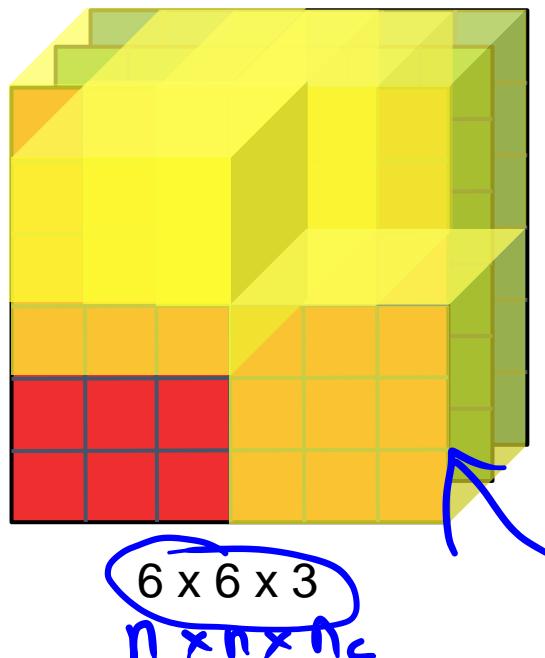
- Low computational cost at deployment
- Useful for mobile and embedded vision applications
- Key idea: Normal vs. depthwise-separable convolutions

Các thiết bị di động hay Raspberry cần mô hình có độ tính toán thấp

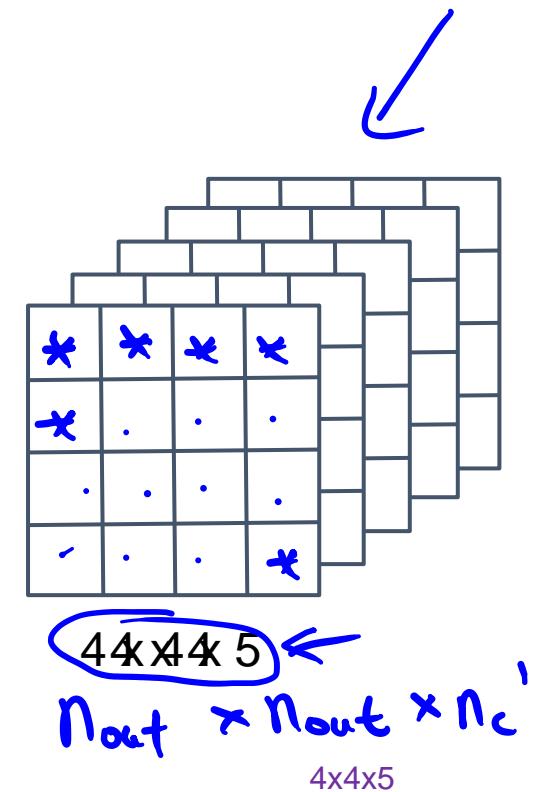
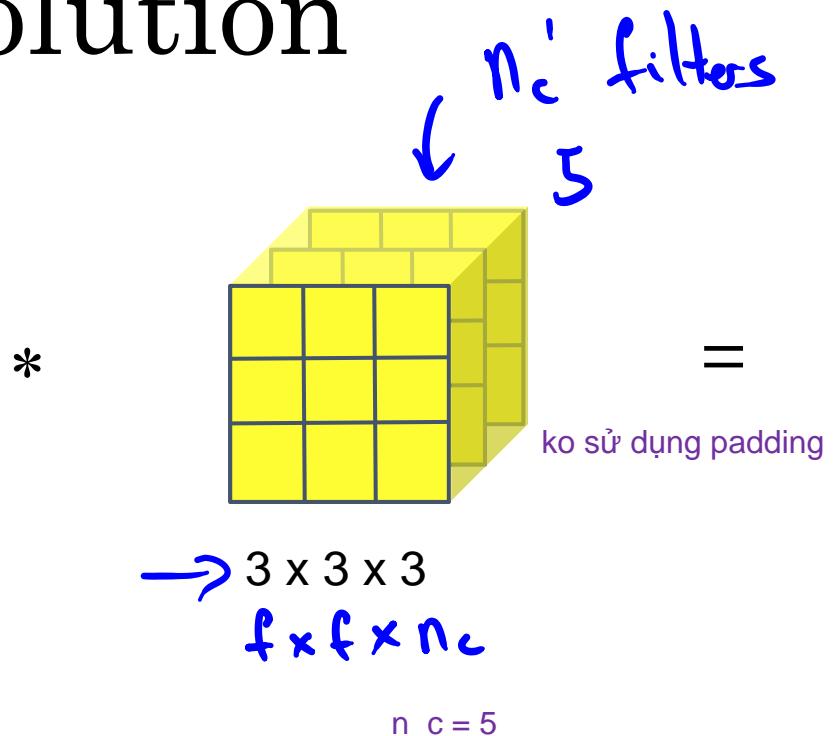


# Normal Convolution

Công thức tổng  
quát tính  
computation cost  
Tốt nhất tự tính  
cho hiểu



$$\text{Computational cost} = \underline{\underline{2160}}$$



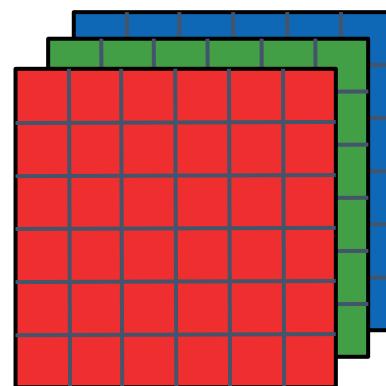
#filter params	x	# filter positions	x	# of filters
$3 \times 3 \times 3$	x	$4 \times 4$	x	5
				đây là số filters

tại một vị trí cân  
số này phép tính nhân  
element-wise

nhân với số vị trí

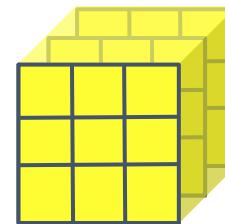
# Depthwise Separable Convolution

Normal Convolution

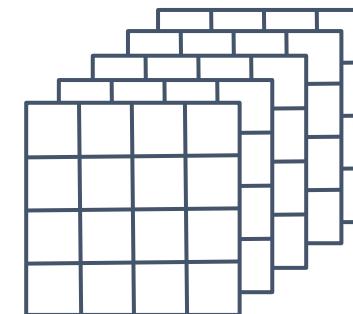


Convolution thông thường thì 1 filter phải có số channels = số channels của input đầu vào.

\*

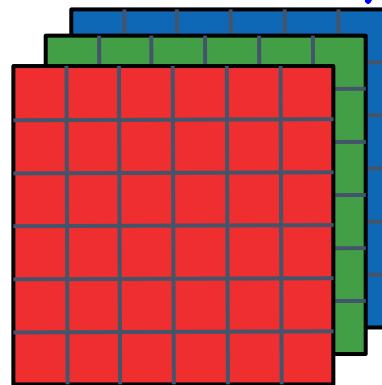


=

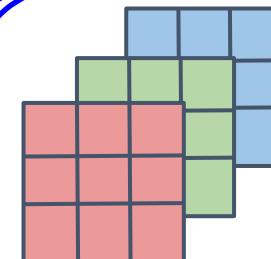


Depthwise Separable Convolution

có 2 bước là : depthwise và pointwise



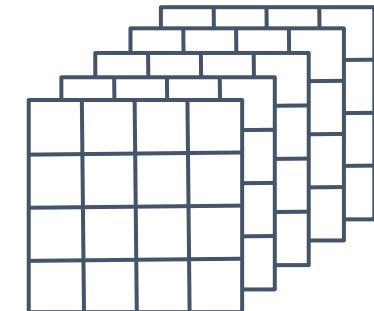
\*



\*



=

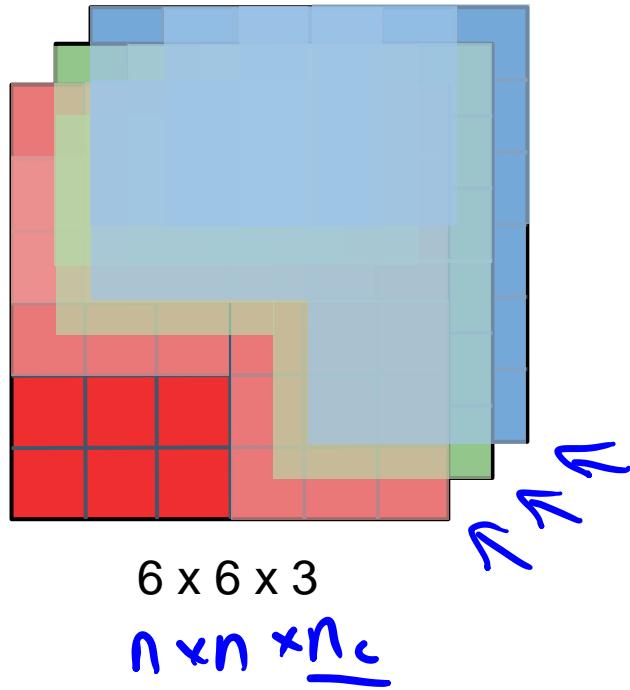


Depthwise

Pointwise

hay còn gọi là depth-wise spatial convolution (một filter chỉ có 1 channels và tác động lên duy nhất 1 channels của tensor đầu vào)

# Depthwise Convolution

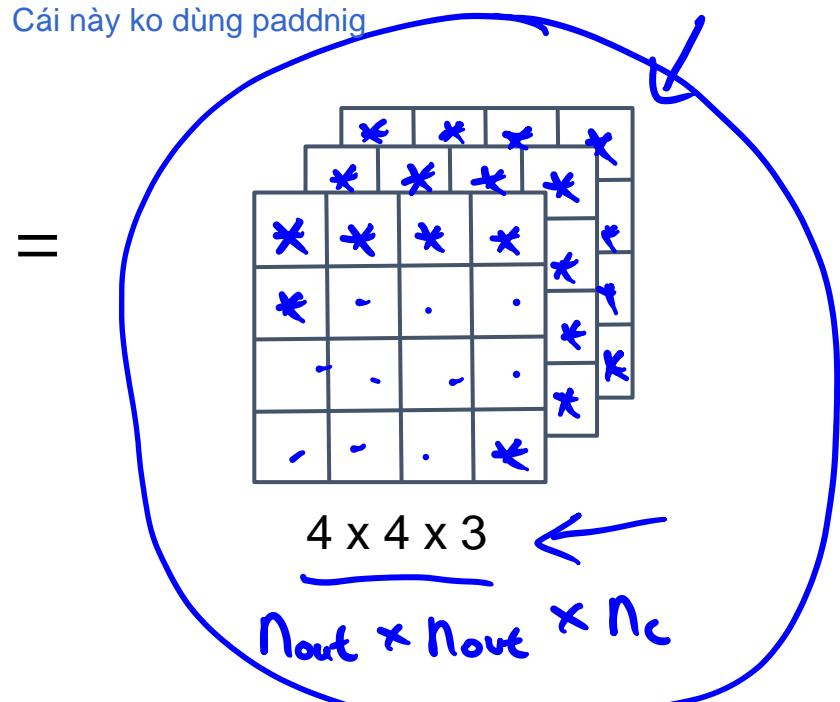
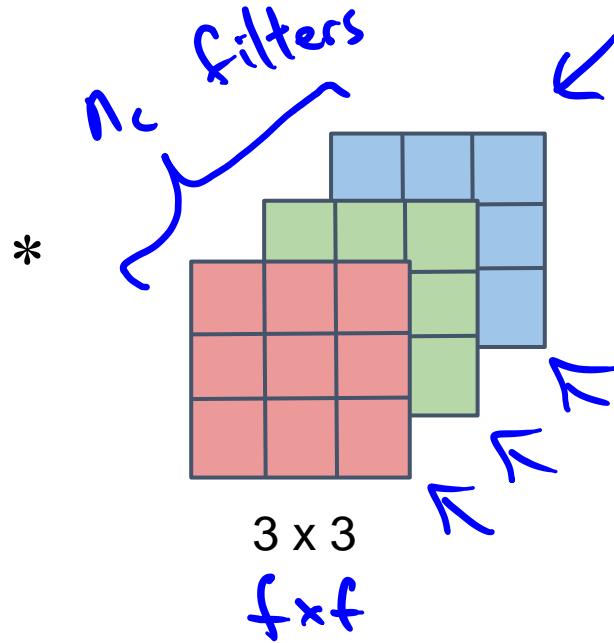


Computational cost

432

Mỗi filter sẽ tác động lên 1 channel trong đầu vào để được 1 lớp đầu ra.

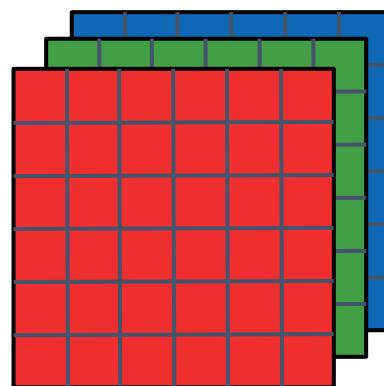
Nhiều filters sẽ có đầu ra nhiều channels.



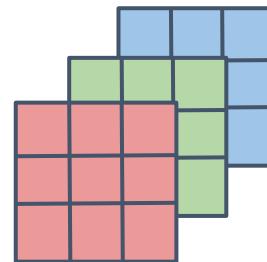
Computational cost	=	#filter params	x	# filter positions	x	# of filters
432	=	$3 \times 3$	x	$4 \times 4$	x	3
				Số phép tính tại một vị trí	Số vị trí $4 \times 4$ đầu ra	
				số filter đầu ra (cũng = đầu vào)		

# Depthwise Separable Convolution

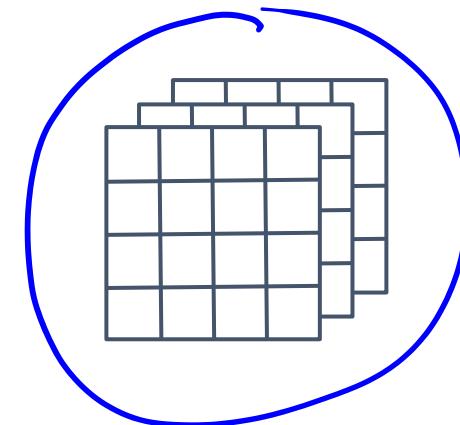
Depthwise Convolution



\*

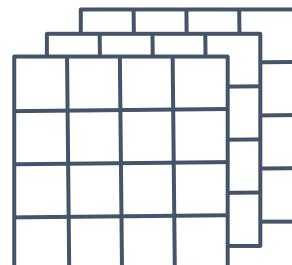


=



432

Pointwise Convolution



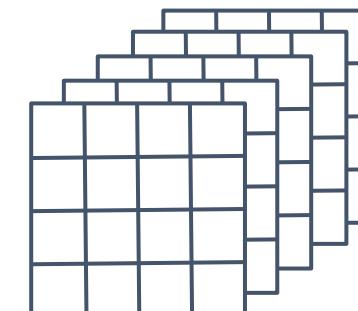
4x4x3

Đơn giản chỉ là Conv layer 1x1

\*

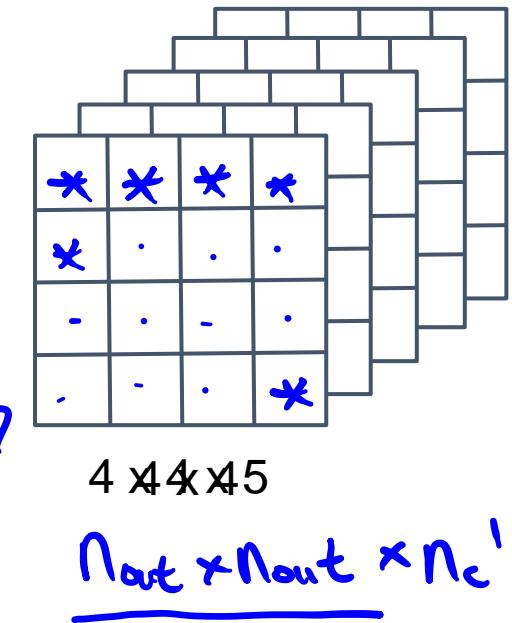
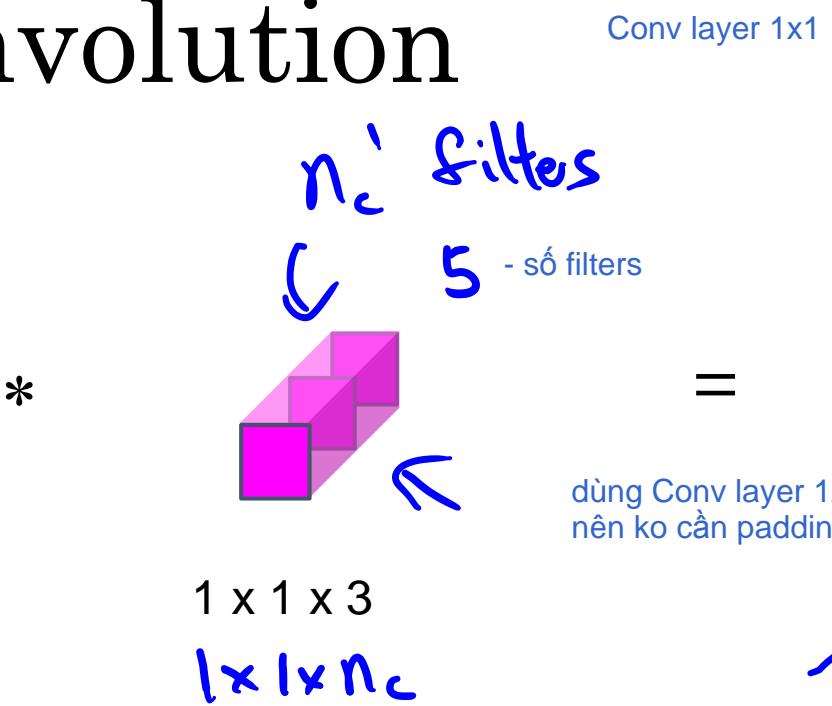
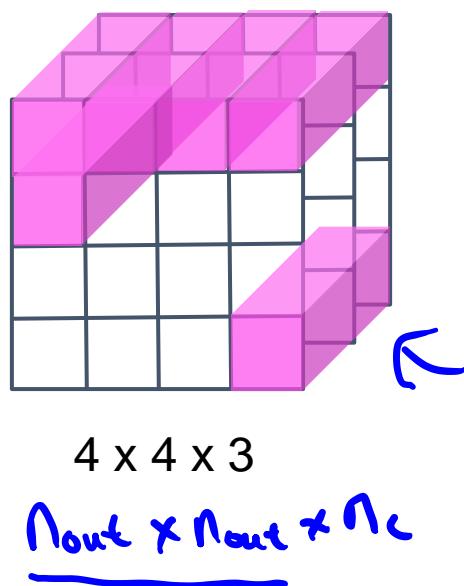


=



4x4x5

# Pointwise Convolution



Computational cost = #filter params  $\times$  # filter positions  $\times$  # of filters

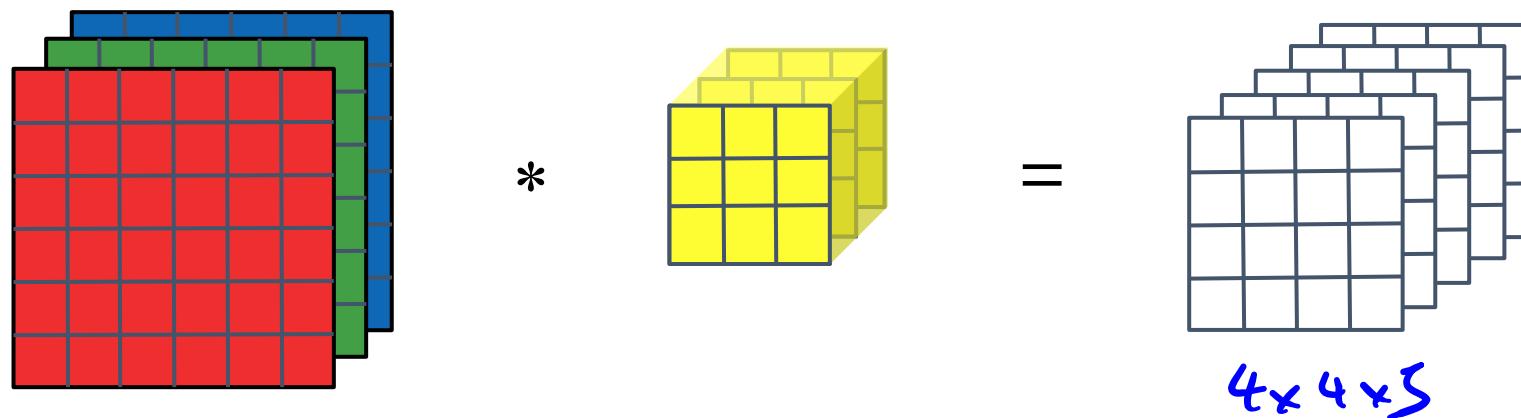
$$240 = 1 \times 1 \times 3 \times 4 \times 4 \times 5$$

số phép tính tại một vị trí

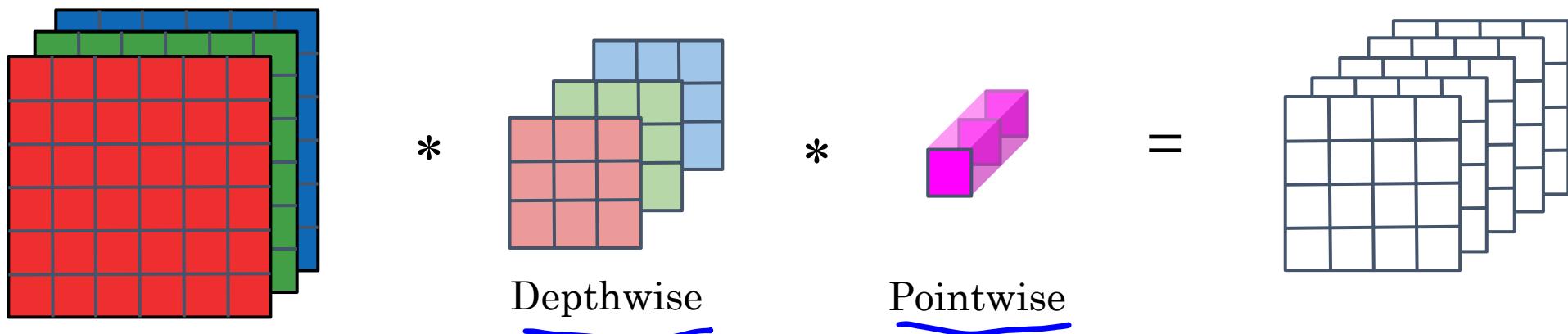
số lượng filters

# Depthwise Separable Convolution

Normal Convolution



Depthwise Separable Convolution



# Cost Summary

Cost of normal convolution

2160

Cost of depthwise separable convolution

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

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

Công thức này có thể dễ dàng chứng minh được  
khi mình viết tổng quát. Làm tương  
tự như trên.

f - kích thước filter

n\_c' - # output channels

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

kết quả bài  
đang làm đây

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

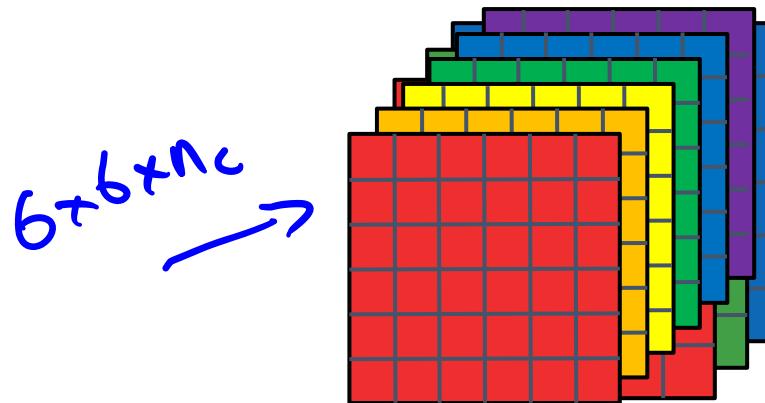
$$= \frac{1}{512} + \frac{1}{3^2}$$

n10 times cheaper

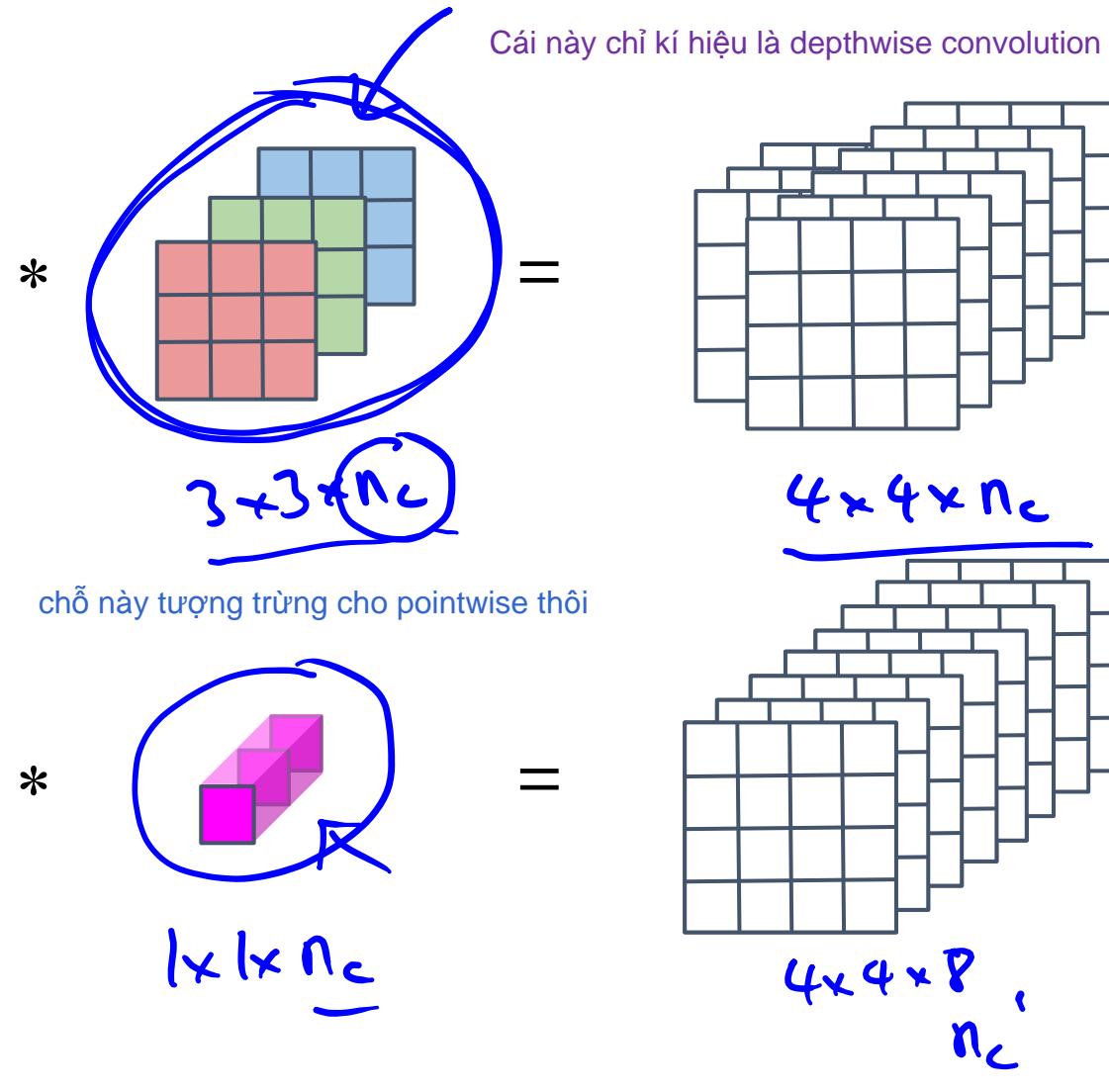
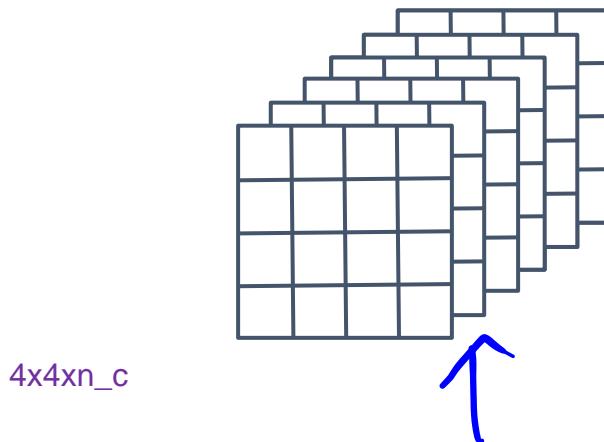
hay dùng trong thực tế

# Depthwise Separable Convolution

Depthwise Convolution



Pointwise Convolution





deeplearning.ai

# Convolutional Neural Networks

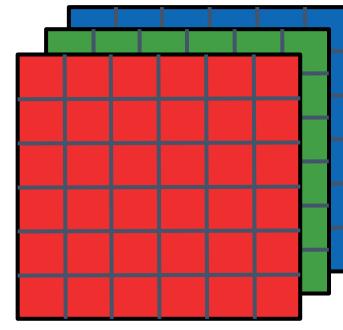
---

## MobileNet Architecture

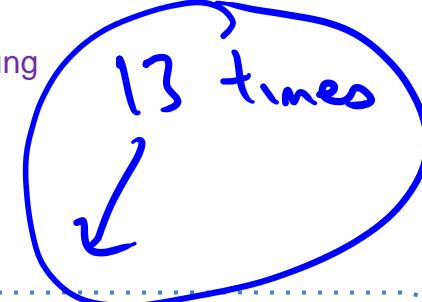
Ý tưởng của MobileNet là thay normal convolution bằng depth-wise seperable convolution

# MobileNet

MobileNet v1

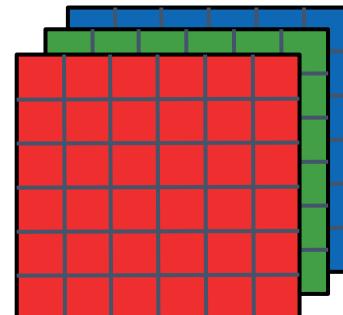


MobileNet v1 sử dụng  
13 lần block này



cấu trúc của MobileNet  
v1

MobileNet v2



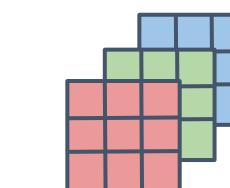
2 sự thay  
đổi chính  
của v2 so  
với v1

Residual Connection

có skip connection  
để học tốt hơn

expansion layer:  
layer mở rộng

Expansion



Depthwise

Projection cũng  
là pointwise convolution

Projection

POOL, FC,  
SOFTMAX

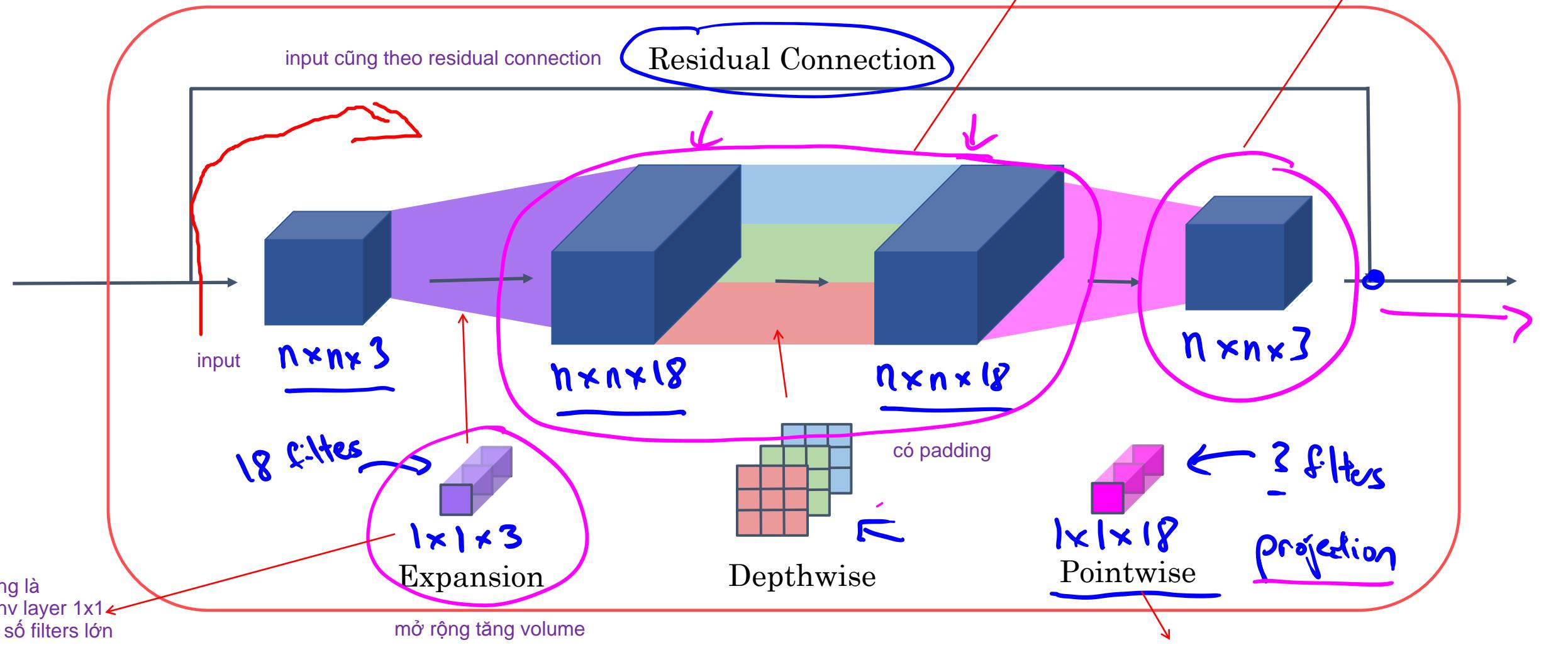
Bottleneck

Cũng là Conv layer 1x1 với  
số layers nhiều hơn

kiến trúc của version 2

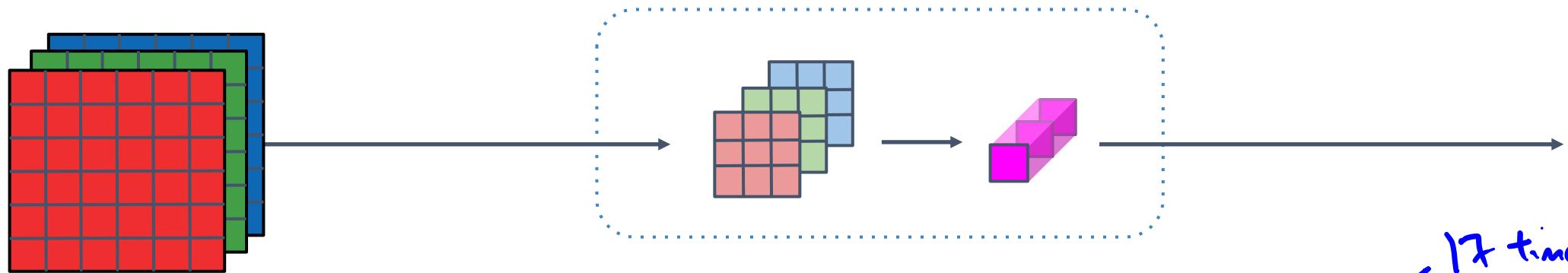
Phản dự đoán

# MobileNet v2 Bottleneck

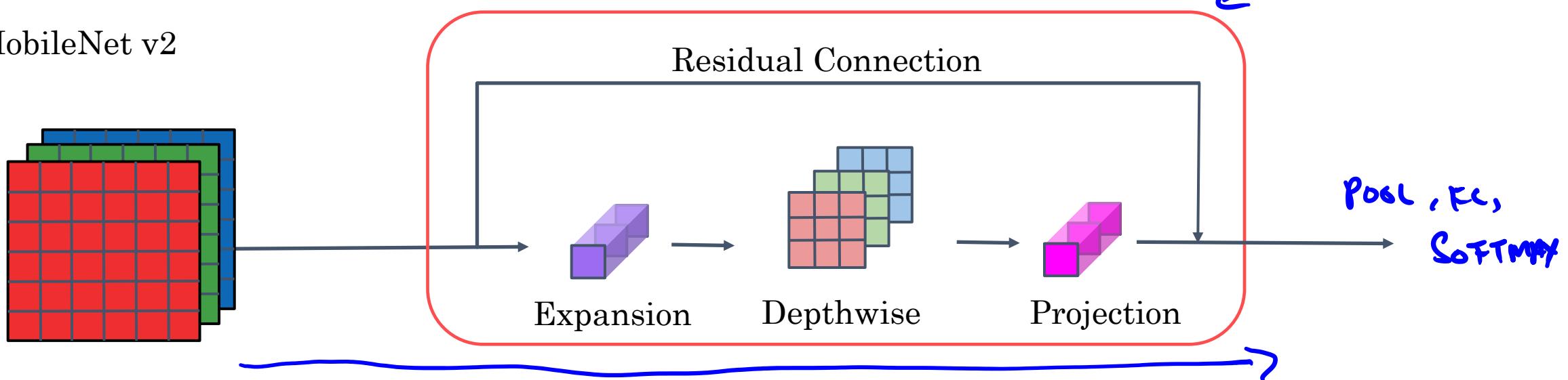


# MobileNet

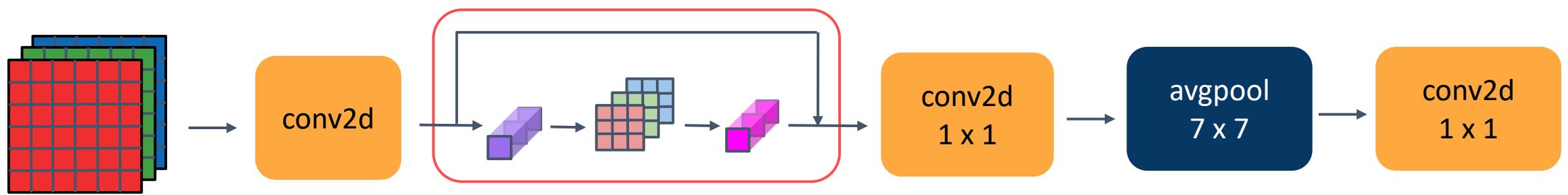
MobileNet v1



MobileNet v2



# MobileNet v2 Full Architecture





deeplearning.ai

# Convolutional Neural Networks

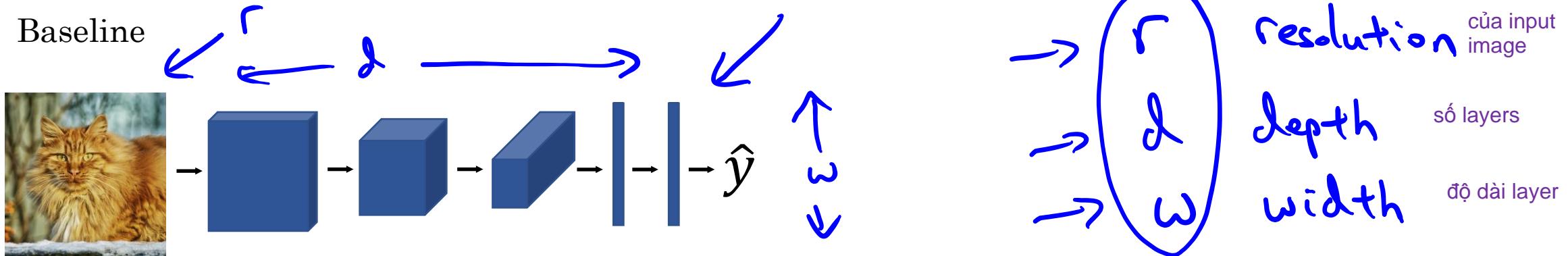
---

## EfficientNet

Tìm cách nào đó để tune các model cho phù hợp với từng device cụ thể?

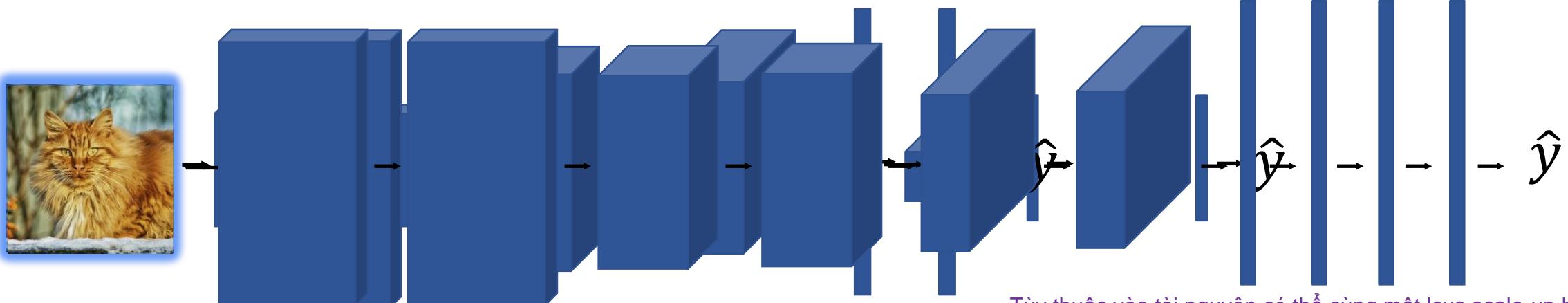
Ví dụ muốn giảm model chút để phù hợp cho thiết bị di động hay muốn tăng lên một chút để tăng chút accuracy.

# EfficientNet



Có 3 cách để scale up or down model: resolution, depth and width

## Widthwise Rescaling



Câu hỏi đặt ra là cho computational budget cách lựa chọn  $r$ ,  $d$ ,  $w$  như nào cho phù hợp nhất?

Tùy thuộc vào tài nguyên có thể cùng một lúc scale-up hay scale-down theo nhiều thông số



deeplearning.ai

Practical advice for  
using ConvNets

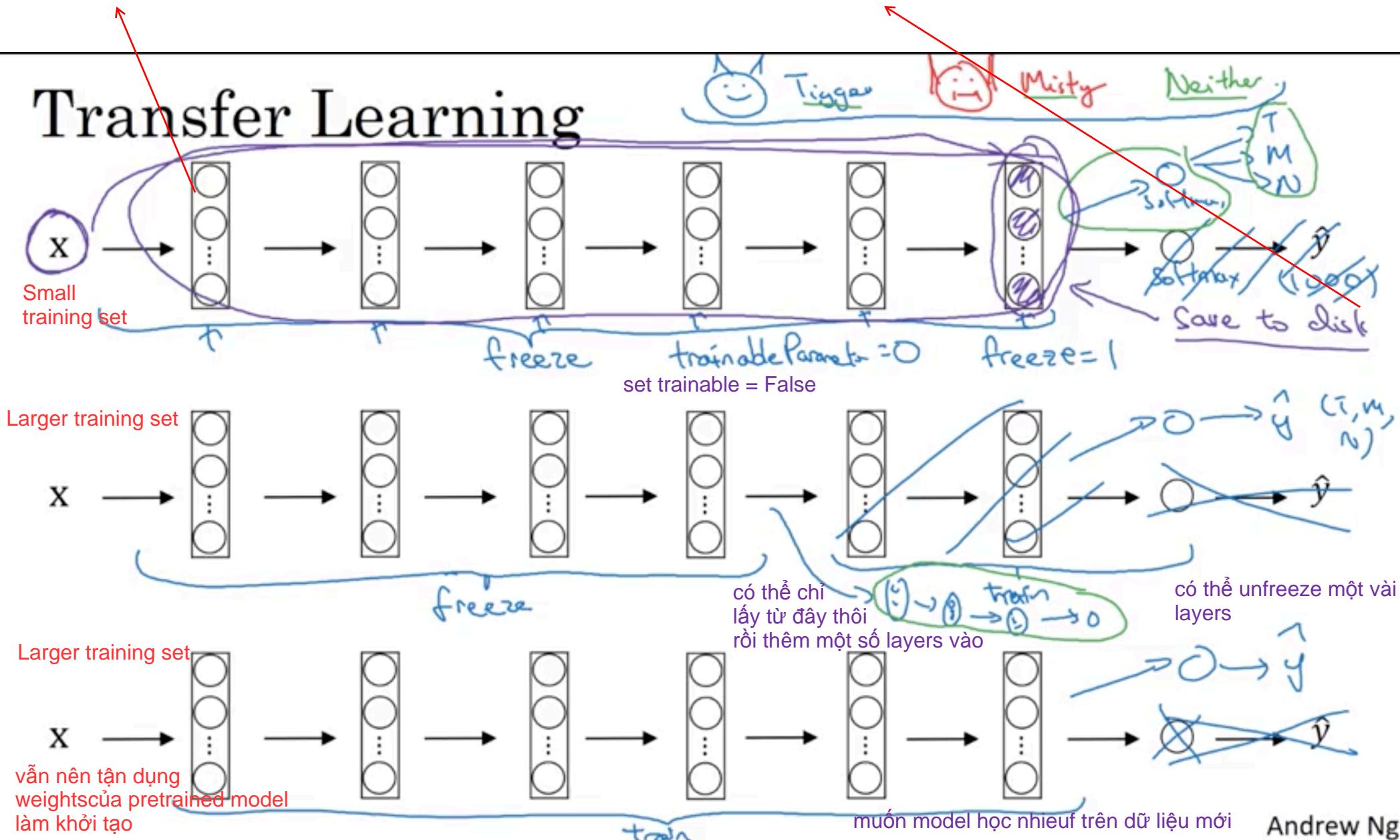
---

Transfer Learning

layer đầu này học được low-level feature nên nó gần như luôn hữu ích trong các TH

Cách 1 nếu chỉ train phần softmax layer hay FC thì có thể tính feature map trước của bộ dữ liệu rồi lưu vào máy tính, sau đó train shallow model có 1 vài layers cho tiết kiệm thời gian trong quá trình training. Lưu ý cách ghép lại weights.

# Transfer Learning



Khi train mà có dùng lại weights thì nên để learning rate nhỏ thôi để tránh phá vỡ weights (ko tận dụng được pre-trained model)



deeplearning.ai

Practical advice for  
using ConvNets

---

Data augmentation

# Common augmentation method

Mirroring



yc

Random Cropping

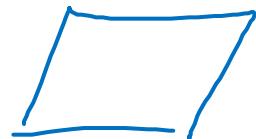


Rotation

Shearing

Local warping

...



# Color shifting



R G B  
↓ ↓ ↓  
+20,-20,+20



-20,+20,+20



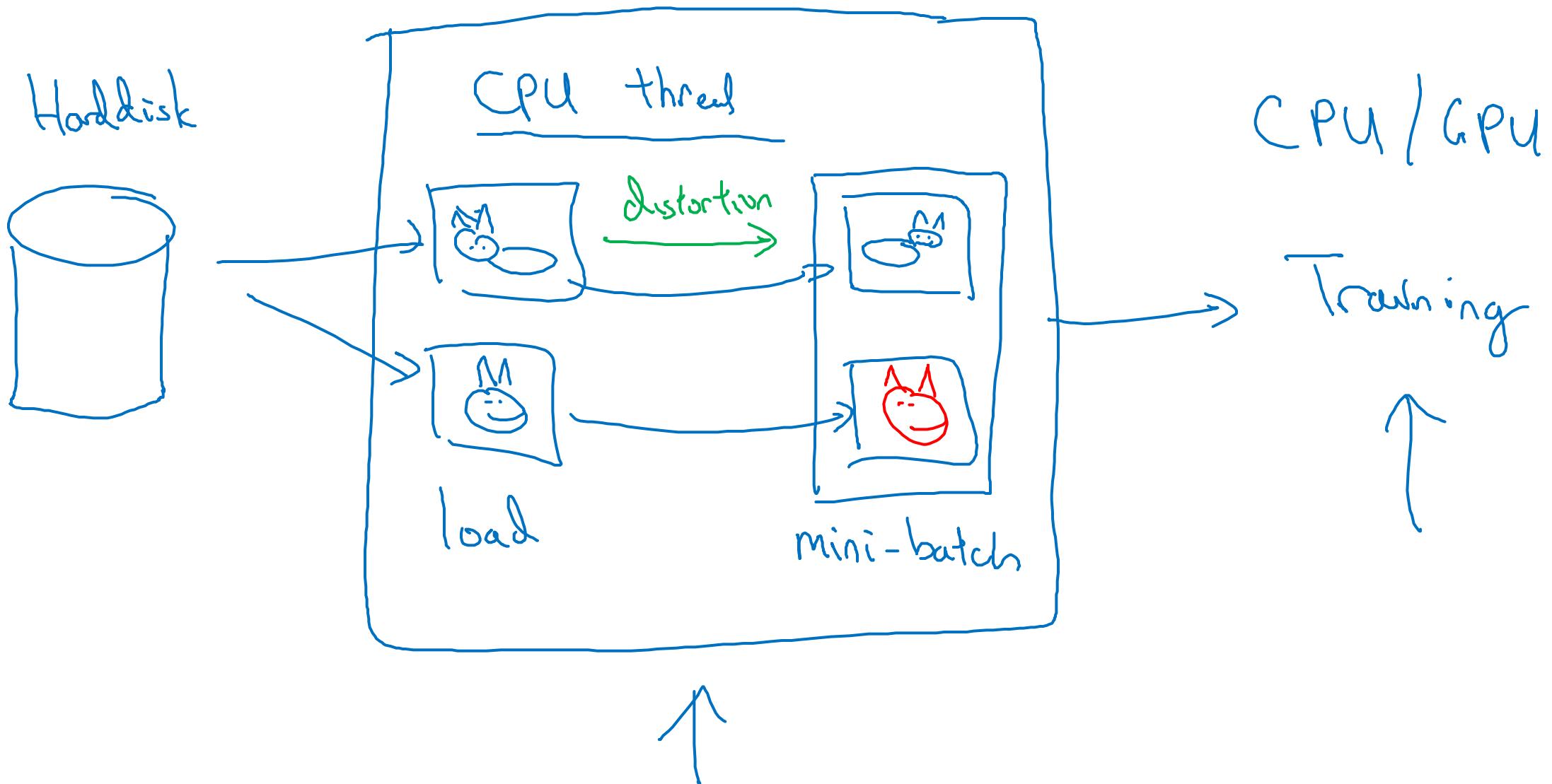
y

+5,0,+50



Advanced:  
PCA  
[ml-class.org](http://ml-class.org)  
[ AlexNet paper  
[ "PCA color augmentation."  
R B      G

# Implementing distortions during training





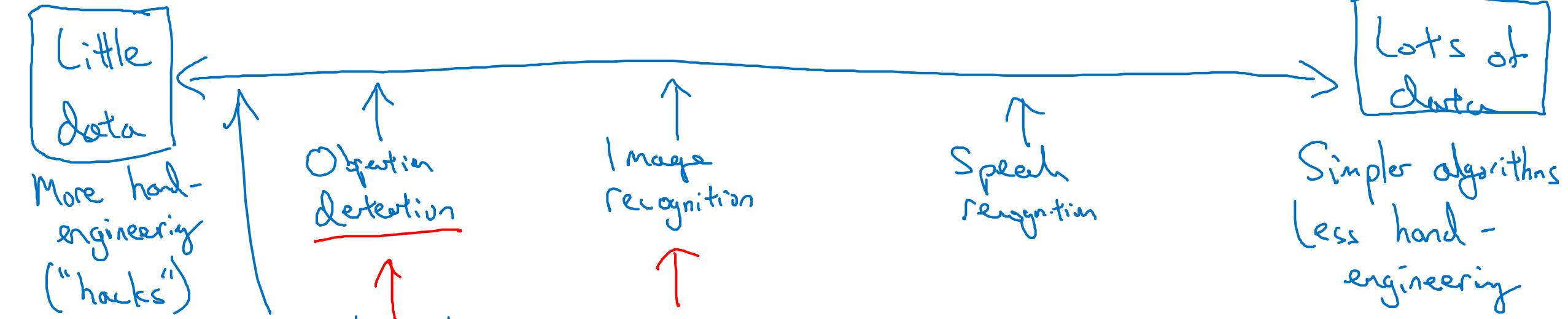
deeplearning.ai

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

$\rightarrow \hat{y}$

- 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