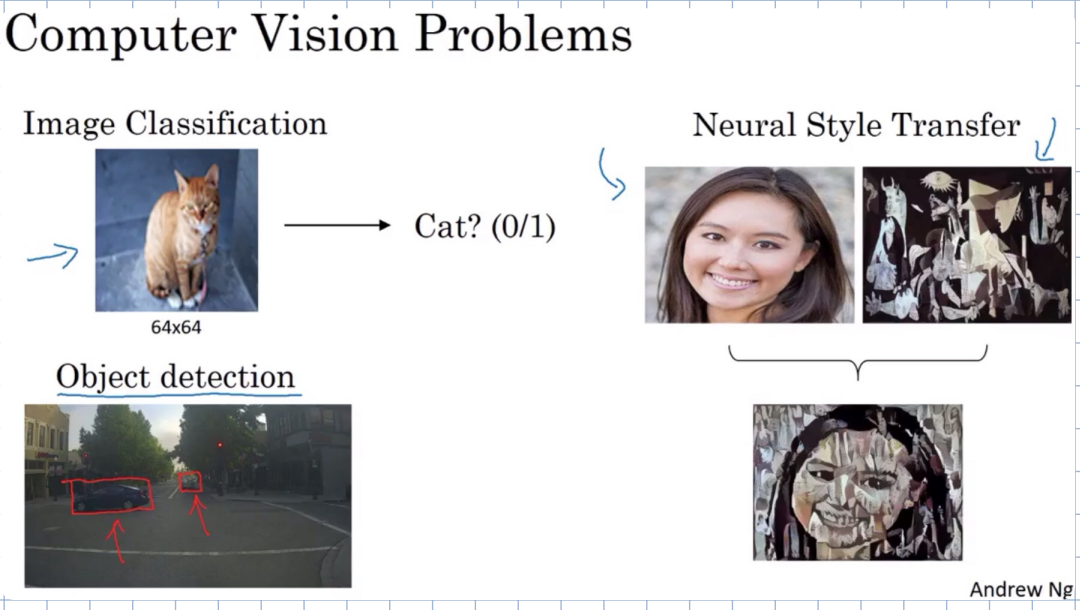
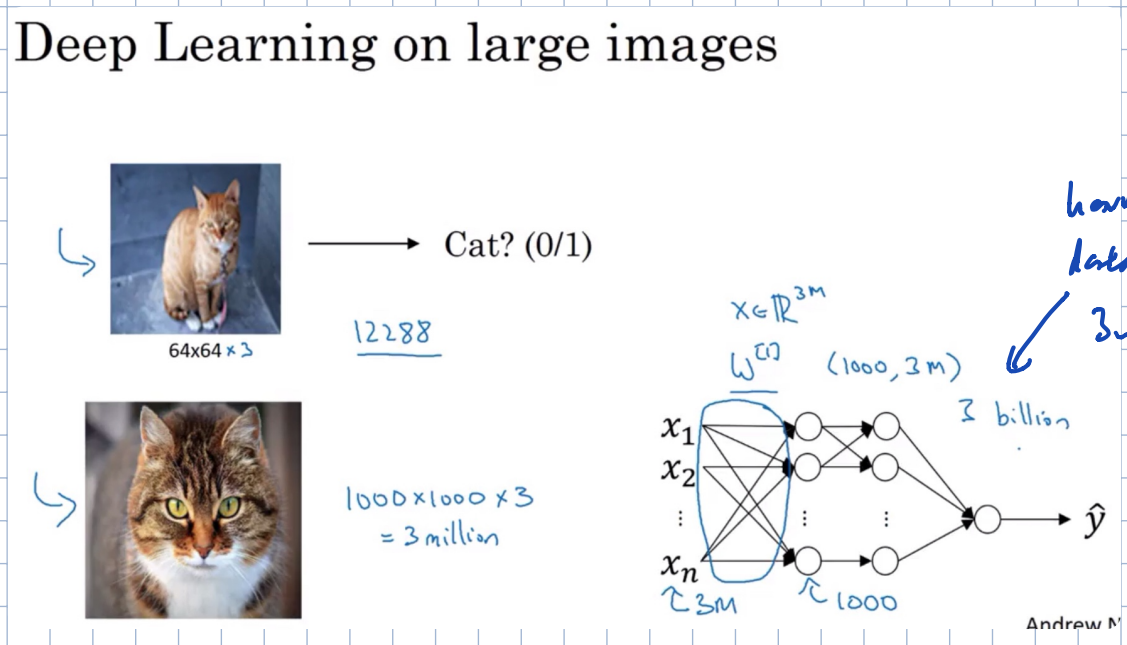
Course 4: Convolutional Neural Network

week 1: Foundations of CNN

Computer Vision





hard to find so many data not to overfit

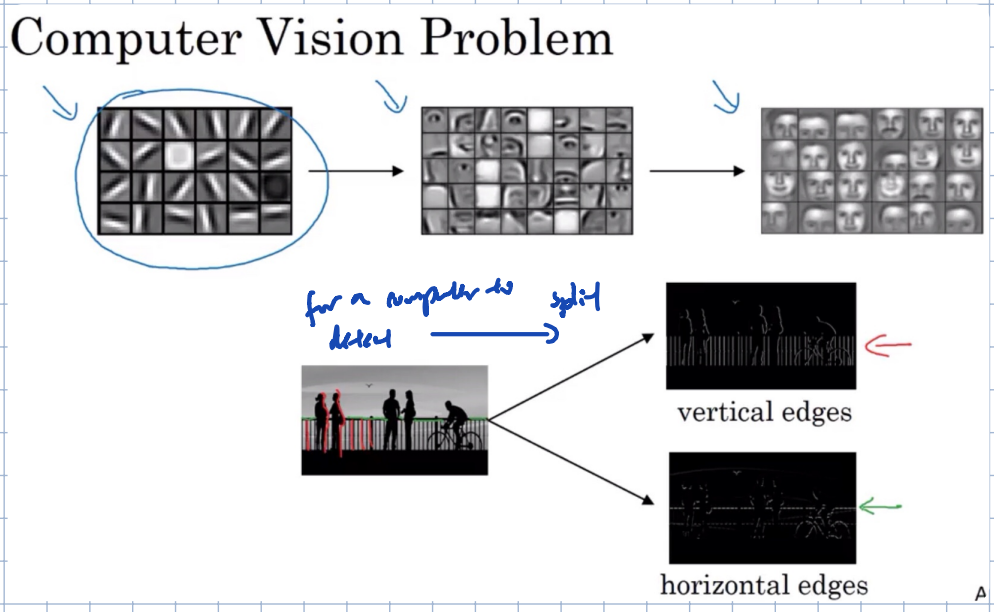
3 mil features

Hence, standard Fully Connected NN would work → too many features, and weights have huge number of elements.

↳ CNN

Edge detection example

The convolution operation is one of the fundamental building blocks of a CNN. → use edge detection as a motivating example here.



vertical edge detection

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6x6 grayscale

"convolution" *

1	0	-1
1	0	-1
1	0	-1

3x3 filter or kernel

=

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

4x4 output

↳ maybe another image

$3\times1 + 1\times1 + 2\times1 + 0\times0 + 0\times5 + 0\times7 + 1\times1 + 1\times8 + 1\times2 = -5$

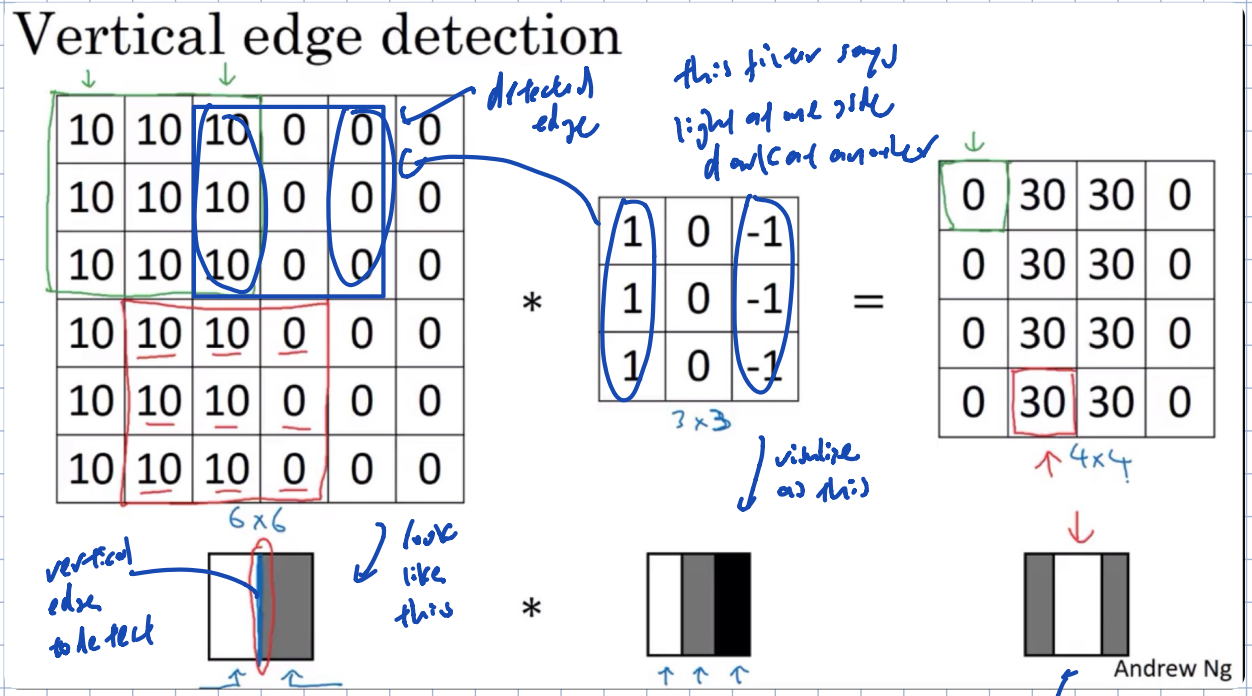
$0\times1 + 0\times5 + 1\times7 + 1\times0 + 5\times0 + 8\times0 + 2\times(-1) + 9\times(-1) + 5\times(-1) = -4$

python: conv_forward() function (deep learning)

tensorflow: tf.nn.conv2d

keras: Conv2D

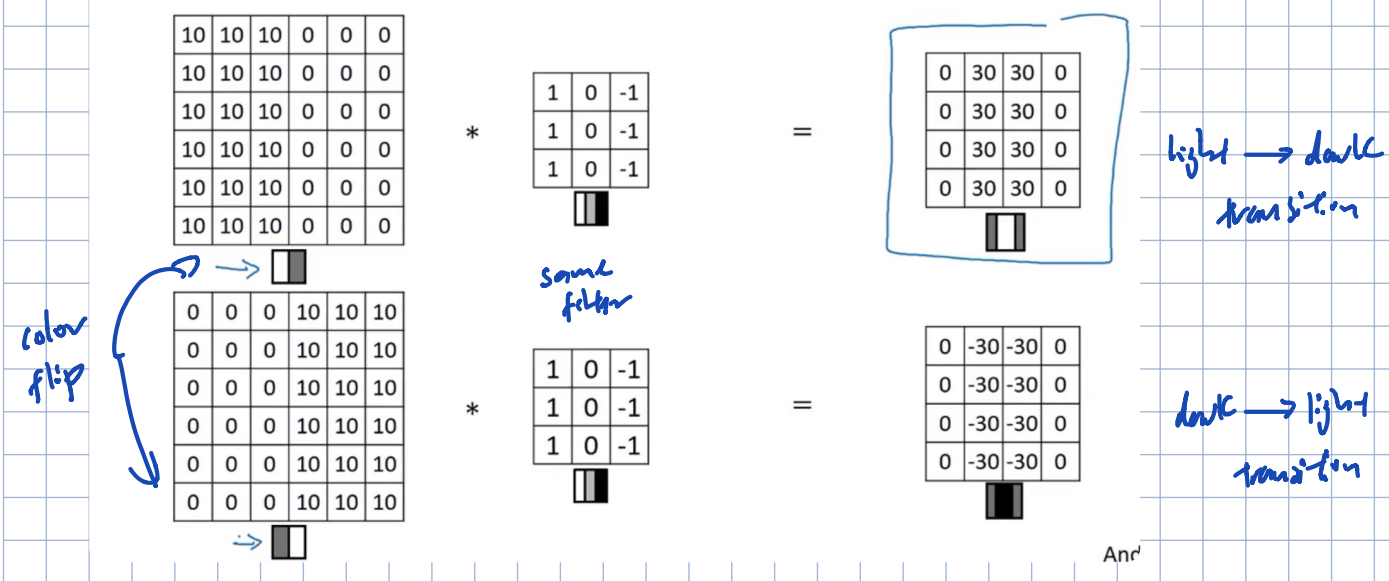
why this filter can act as edge detection ?



looks thick because we working with small image, if 1000x1000 image is used ↳ quite accurate

More Edge Detection

Previous example is about detecting vertical edge. Here we investigate positive and negative edges, i.e. difference between light to dark versus dark to light edge transitions.



Vertical and Horizontal Edge Detection

1	0	-1
1	0	-1
1	0	-1

Vertical

← vertical edge according to the filter is 3×3 region where pixel are bright on left, dark on the right part.

1	1	1
0	0	0
-1	-1	-1

Horizontal

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

6×6

*

1	1	1
0	0	0
-1	-1	-1

look like this

=

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

overfitted for small image
↳ capture pixel info on both intermediate value, transition will not be observed in larger pic

Learning to detect edges

1	0	-1
1	0	-1
1	0	-1

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

×

1	0	-1
2	0	-2
1	0	-1

Sobel filter

more weight on central row

3	0	-3
10	0	-10
3	0	-3

Scharr filter

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

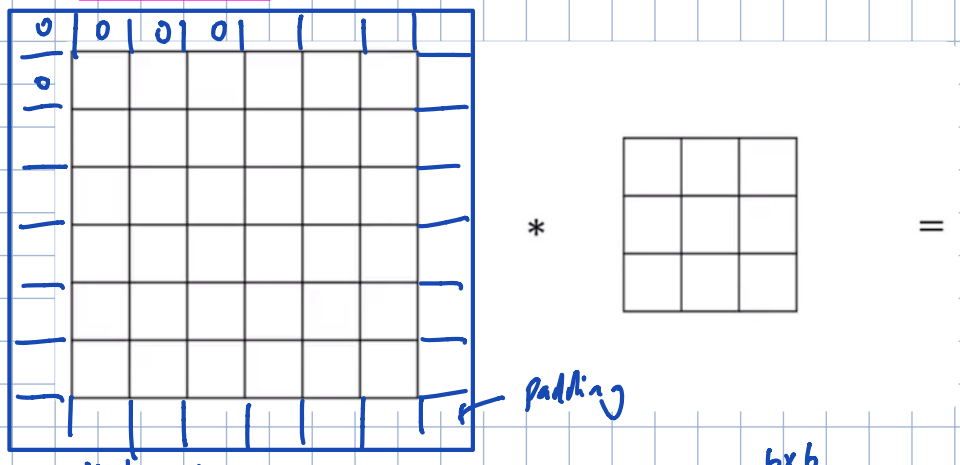
=

we can learn these instead of handpick them

→ gives good edge detector

↳ can be chosen to learn 45°, 75° edge

Padding



previously: $6\times6 * 3\times3 \rightarrow 4\times4$; now: 6×6 (padding) $\rightarrow 8\times8 * 3\times3 \rightarrow 6\times6$

↳ $n\times n * f\times f \rightarrow (n-f+1)\times(n-f+1)$

image · filter · assume stride = 1

Downside: ① every time we apply a convolutional operator, the image shrinks

↳ image will be really small after a few layers

② the image at the corners/sides are only being used once

↳ throwing away info at the pixel at the side

p = padding = 1 ⇒ padding all around with an extra border of 1 pixels.

↳ $(n+2p-f+1)\times(n+2p-f+1)$

$= (6+2-3+1)\times(6+2-3+1)$

$= 6\times6$

Valid and Same convolutions

In terms of how much to pad, there are 2 common choices:

① valid convolution

② same convolution

"Valid": no padding

$n \times n \quad * \quad f \times f \longrightarrow (n-f+1) \times (n-f+1)$

e.g. $6 \times 6 \quad * \quad 3 \times 3 \longrightarrow 4 \times 4$

"Same": Pad so that output size is the same as the input size

$(n \times n + p) * f \times f \longrightarrow (n+2p-f+1) \times (n+2p-f+1)$

e.g. if we want output size same as input size

$n+2p-f+1 = n$

$\hookrightarrow \quad p = \frac{f-1}{2}$

By convention in CV, f is always odd number

↳ if f is even, then need asymetric pading

↳ odd filter has a central position → distinguisher, nice to have

Strided Convolutions

2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

7x7

*

3	4	4
1	0	2
-1	0	3

3x3

stride = 2

=

91	100	83
69	91	127
44	72	74

3x3

In general:

$\lfloor z \rfloor = \text{floor}(z)$

$n \times n \quad * \quad f \times f$

padding p, stride s

$$\rightarrow \left\lfloor \left(\frac{n+2p-f}{s} + 1 \right) \right\rfloor \times \left\lfloor \left(\frac{n+2p-f}{s} + 1 \right) \right\rfloor$$

floor

↳ when it this fraction is not an integer? then we take the floor
↳ round the number down

i.e. the way this is implemented is that, the box multiplication is only performed when the box is fully contained within the image or image + padding

(in short, filter has to be completely within image or image + padding - then perform computation)

Technical note on cross-correlation vs. convolution

Convolution in math textbook:

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

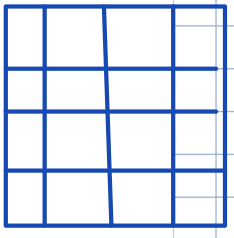
*

3	4	5
1	0	2
-1	9	7

flip first I and ↔ axis

7	9	-1
2	0	1
5	4	3

use this filter

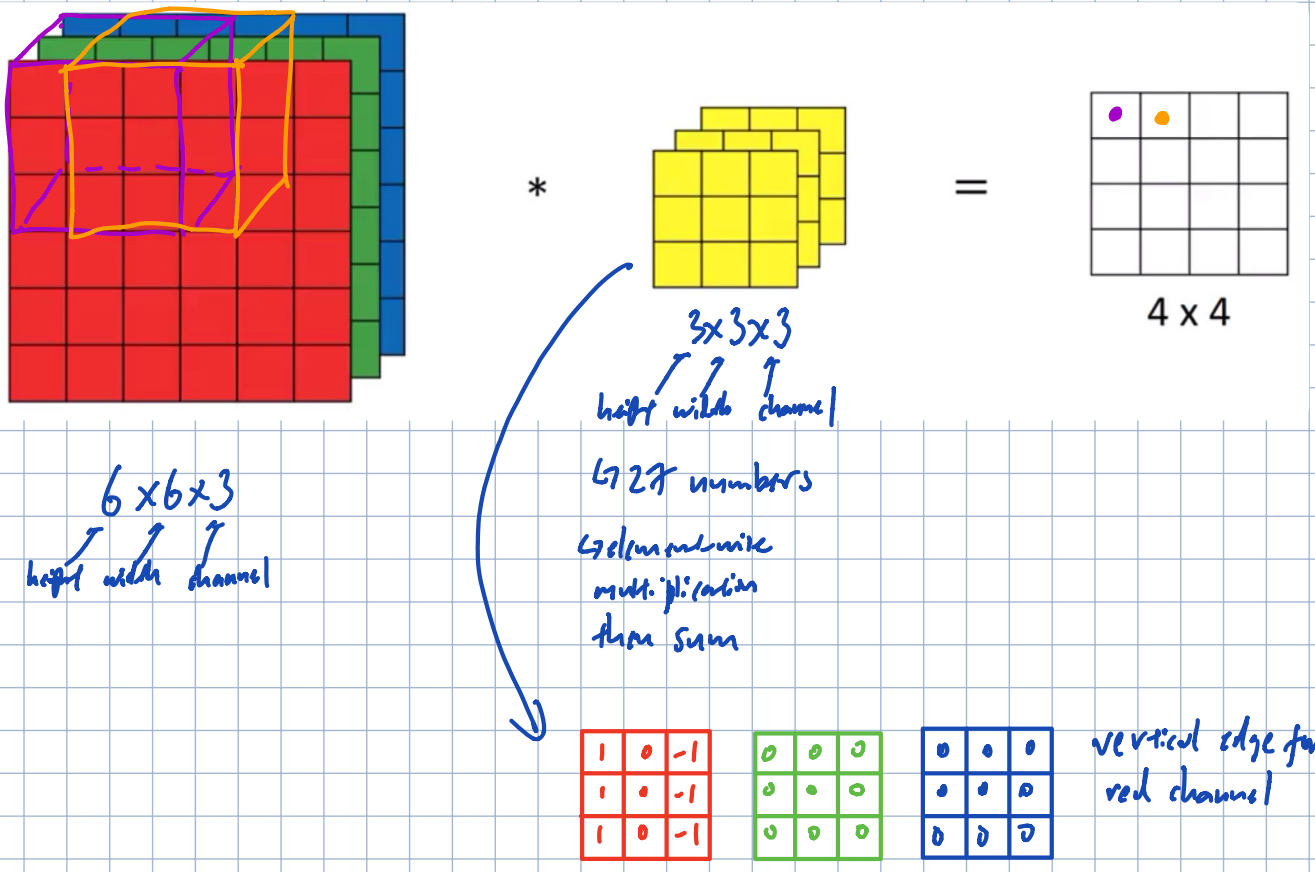


this is proper convolution from math ↑

we skip flipping in DL, so it should be called cross-correlation instead

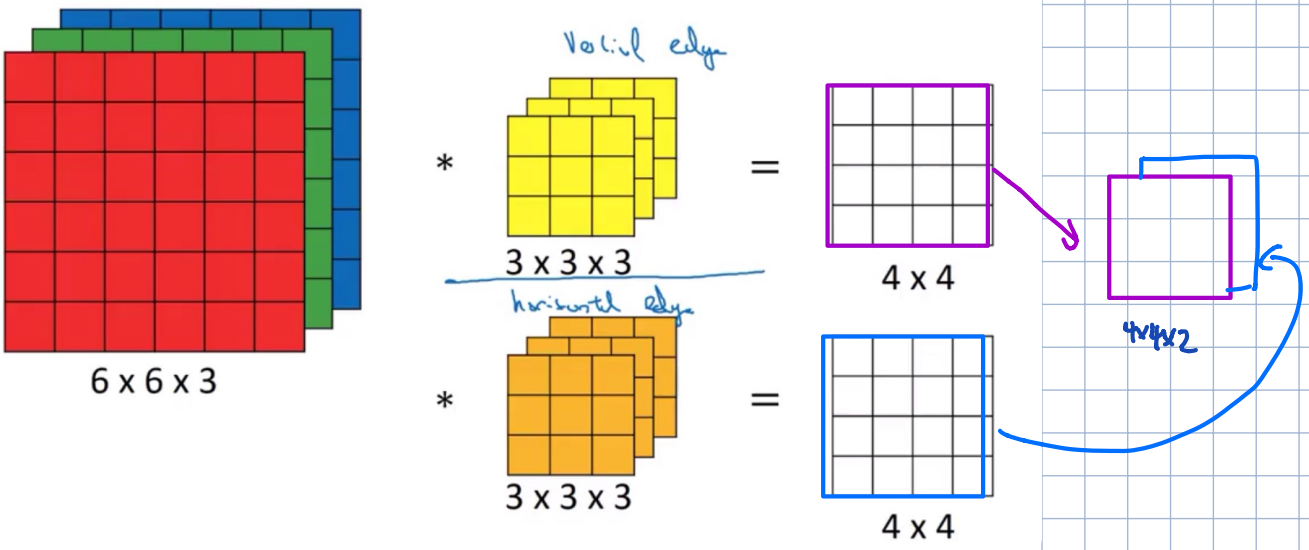
but we don't care, just call convolution anyway

Convolutions over volume



Multiple filters

detect vertical and horizontal edge at the same time



Summary: $n \times n \times n_c$ * $f \times f \times n_c$ $\longrightarrow$ $(n-f+1) \times (n-f+1) \times n_c'$

eg. $6 \times 6 \times 3$ * $3 \times 3 \times 3$ $\longrightarrow$ $4 \times 4 \times 2$

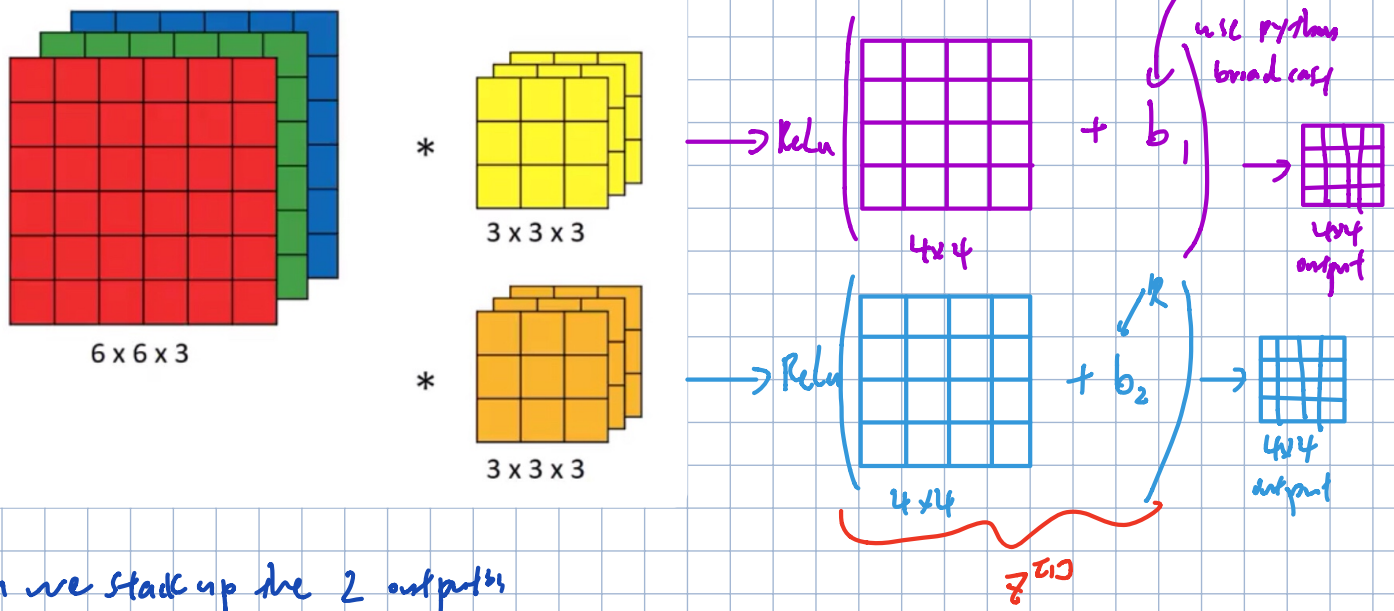
same

assume stride = 1

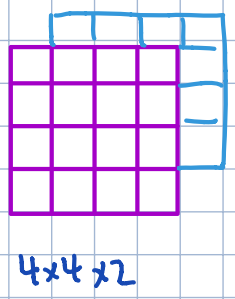
n_c' : no. of filters we use

One layer of a Convolutional Network

Example of a layer convolving with 2 filters



then we stack up the 2 outputs



From 6x6x3 → 4x4x2

1 layer of convolutional neural network

Recall that 1 step of forward prop is:

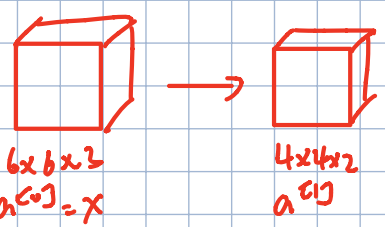
$$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$$

$$a^{[1]} = g(z^{[1]})$$

here, 6x6x3 image is $a^{[0]}$ or X

the filters 3x3x3 is similar to $W^{[1]}$

i.e. have 1 layer



In this example, there is 2 filters (or 2 features)

If 10 filters are used, then we will get 4x4x10 output volume

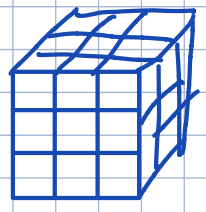
Number of parameters in one layer

If you have 10 filters that are 3 x 3 x 3 in one layer of a neural network, how many parameters does that layer have?

each filter is $3 \times 3 \times 3$

↳ 27 parameters

+ bias = 28 parameters



, so in total we have 280 parameters

no matter your input image size, the parameters # in this layer is still the same.

so these 10 filters can detect v_H, 45° etc for large images as well.

Summary of notation

If layer l is a convolution layer:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding ↗ valid: no padding ↘ same: input size = output size

$s^{[l]}$ = stride

Input to this layer: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

(n_H, n_W: height, width of activations input from previous layer; n_c: # of filters in previous layer)

output of this layer $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$ ($n_c^{[l]}$: # of filters in layer l)

↳ $n_{H/W}^{[l]} = \left\lfloor \frac{n_{H/W}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$

($n_{H/W}^{[l]}$: height & output volume)

⇒ Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$ (matches activations from previous layer)

⇒ activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$ (after bias & non-linearity ReLu)

↳ vectorization $A^{[l]} \rightarrow m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

⇒ weights:

$f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

($f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$: 1 filter; $n_c^{[l]}$: total # of filters)

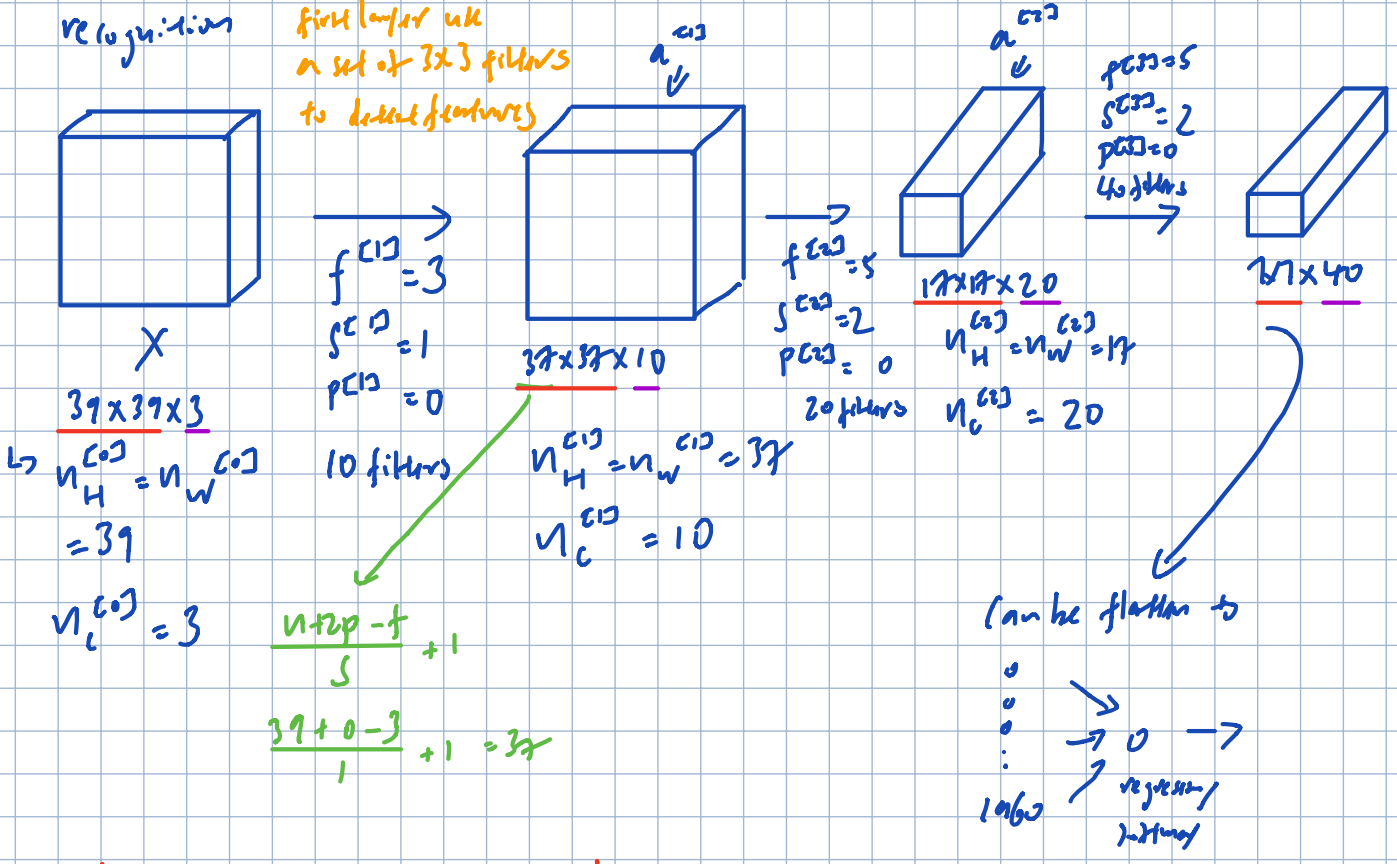
⇒ bias: 1 real number for each filter

$n_c^{[l]}$: a vector of this dimension

Lisa code: $(1, 1, 1, n_c^{[l]})$ for convenience

Simple Convolutional Network Example

Say we have an image, we want to do image classification or image recognition



$\frac{n+2p-f}{s}+1$

$\frac{39+0-3}{1}+1=37$

usually image size decreases with layers

while channel size increase with layers

Types of layer in a convolutional network

① convolution (Conv)

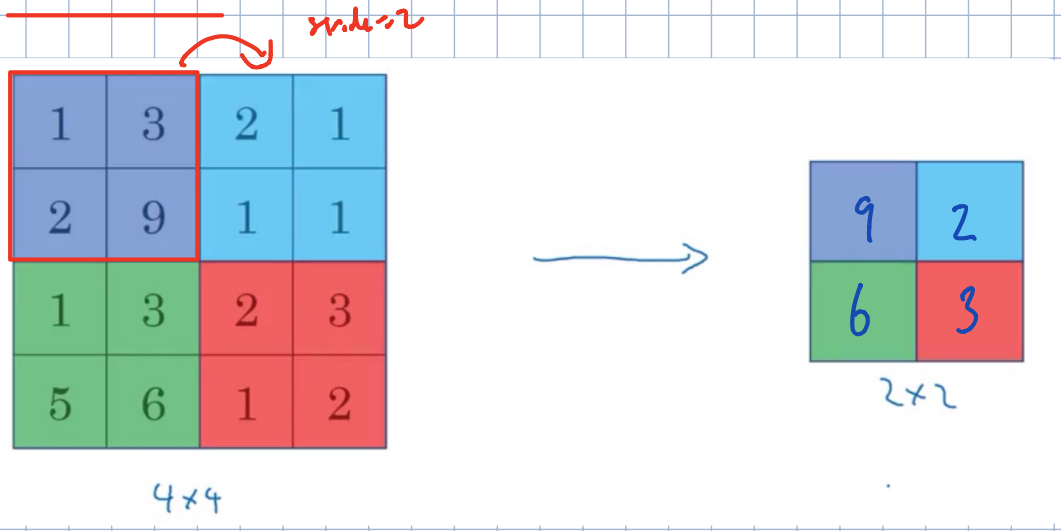
② pooling (Pool)

③ Fully connected (FC)

Pooling Layers

Pooling layers can be used to reduce the size of the representation, hence speed up the computation, also makes some of the features that detects a bit more robust.

Max Pooling



As it we apply a filter size of 2, stride of 2

↳ hyperparameters of max pooling. ← nothing for gradient descent to learn

Intuition: input are the activations of some layer of NN, max pooling detect a particular feature in the selected area

↳ If these features are detected anywhere in this filter, then keep a high number.

1	3	2	1	3
2	9	1	1	5
1	3	2	3	2
8	3	5	1	0
5	6	1	2	9

$5 \times 5 \times n_c$

→

9	9	5
9	9	5
8	6	9

$3 \times 3 \times n_c$

$f = 3$

$s = 1$

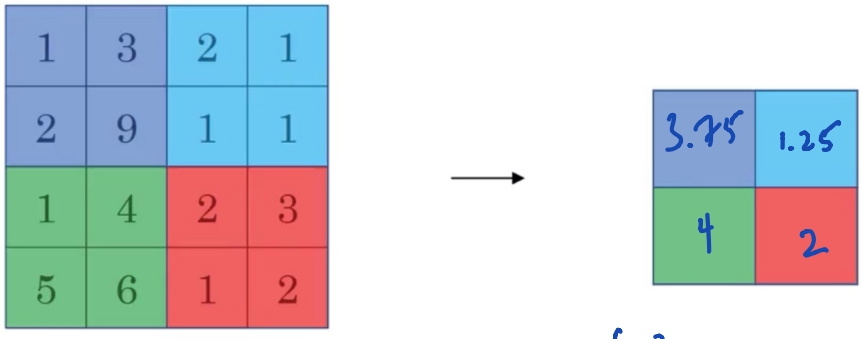
formula to determine output size of Conv

also works for max pooling

If there are multiple channel, do max pooling independently on each channel.

Pooling layer: Average pooling

used often



$f = 2$

$s = 2$

this is used when very deep in the NN, collapse $7 \times 7 \times 1000 \longrightarrow 1 \times 1 \times 1000$

Summary of pooling

Hyperparameters:

f : filter size $f=2, s=2$ $f=3, s=2$

s : stride

Max or average pooling

~~p: padding~~

No parameters to learn!

$n_H \times n_W \times n_c$

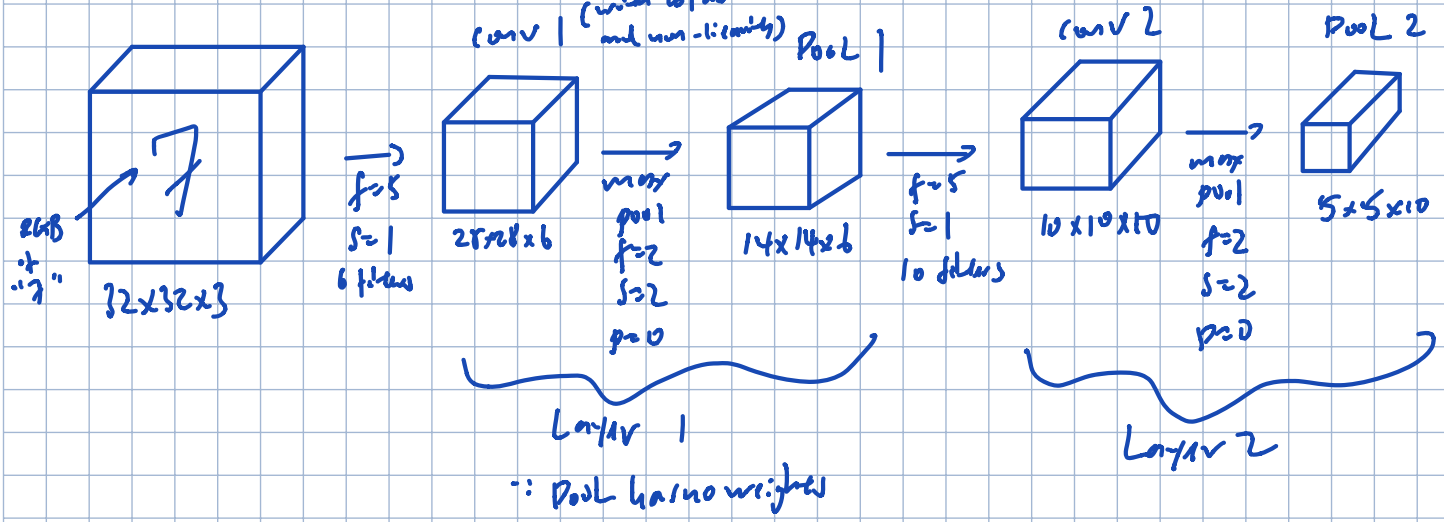
$\downarrow$

$\left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_W - f}{s} + 1 \right\rfloor$

$\times n_c$

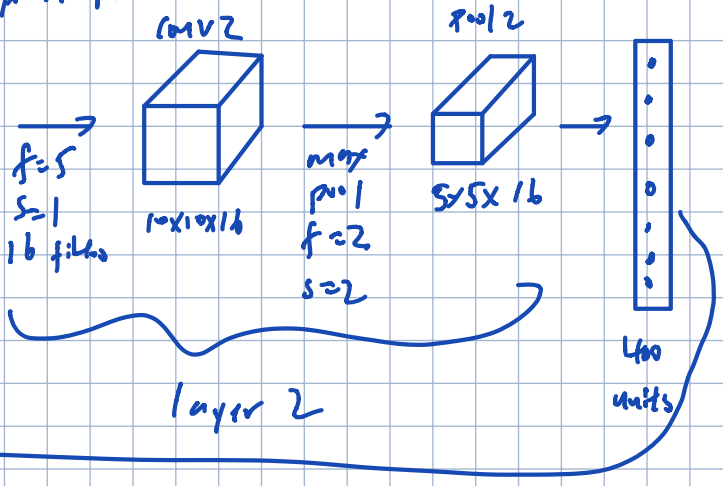
CNN example

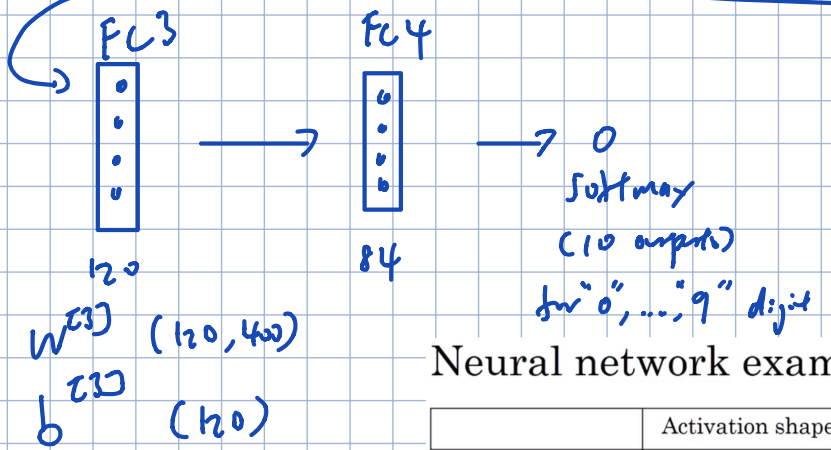
Say we input an image which are 32×32×3



∴ Pool has no weights

another layer example





$W^{[3]}$ (120, 400)

$b^{[3]}$ (120)

Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072 $a^{[0]}$	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208 → (5×5×3+1)×8 = 608
POOL1	(14,14,8)	1,568	0
CONV2 (f=5, s=1)	(10,10,16)	1,600	416 → (5×5×8+1)×16 = 3216
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,001 → 400×120+120 = 48120
FC4	(84,1)	84	10,081 → 120×84+84 = 10164
Softmax	(10,1)	10	841 → 84×10+40 = 880

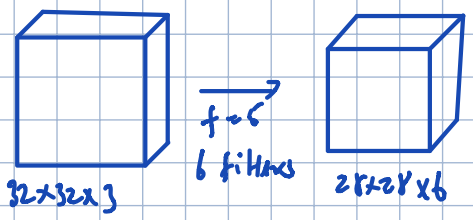
shrinking

why convolutions ?

Advantages of convolution over just dense layers:

① Parameter sharing

② sparsity of connections



3072

4704

(5×5+1)×6

= 456 parameters

$W \sim 3072 \times 4704$

$\sim$ 14mil parameters

Why convolutions

translation invariance — a cat shifted by a few pixel is still a cat

vertical edge detect everywhere in image

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

3×3

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

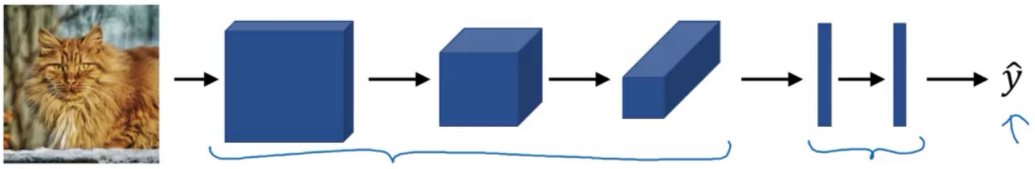
Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

→ **Sparsity of connections:** In each layer, each output value depends only on a small number of inputs.

Putting it together

Training set $(x^{(1)}, y^{(1)}) \ldots (x^{(m)}, y^{(m)})$.

w, b



Cost $J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

Use gradient descent to optimize parameters to reduce J

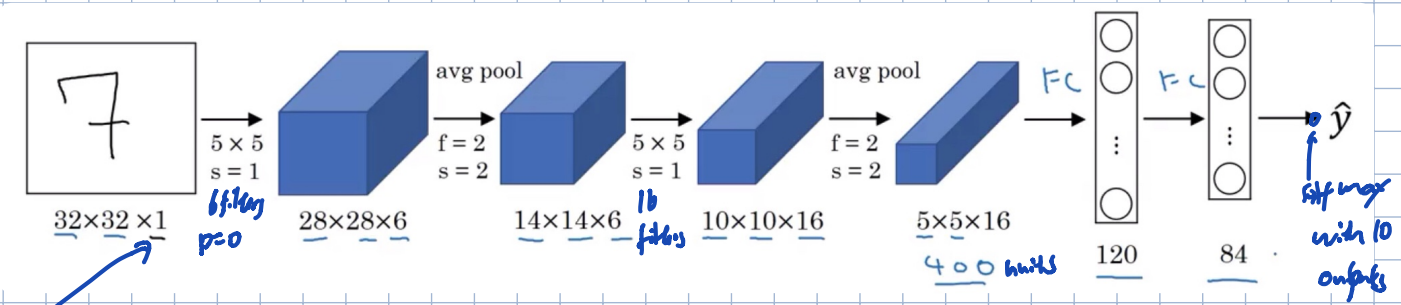
week 2: Deep Convolutional models: case studies

Some of the networks we will be looking at:

- LeNet-5
- Alex Net
- VGG
- Res Net
- Inception

Classic Networks

LeNet-5



LeNet-5 was trained on grayscale images

has about 60K parameters

As we progress deeper into NN,

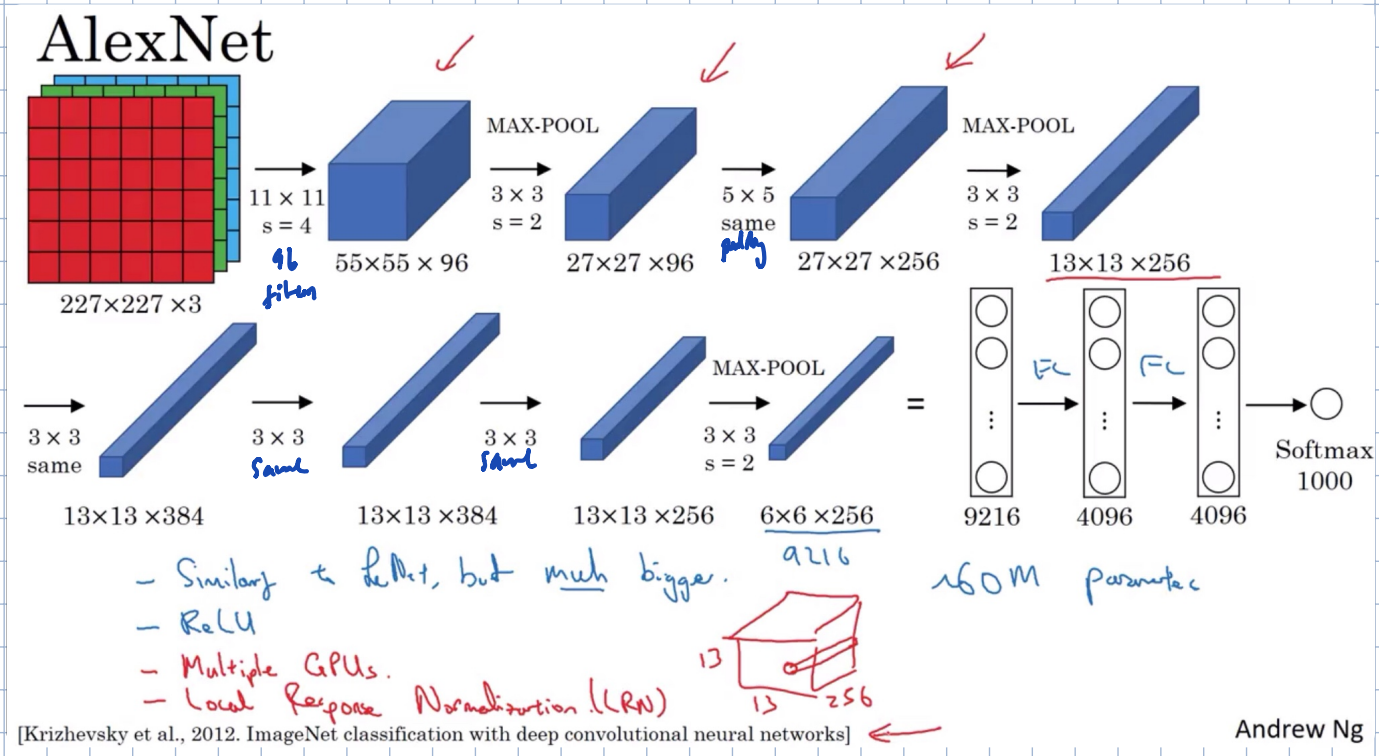
$n_H, n_W \downarrow \quad n_C \uparrow$

a common structure:

conv — pool — conv — pool — fc — fc — output

back then people use sigmoid / tanh and not ReLu.

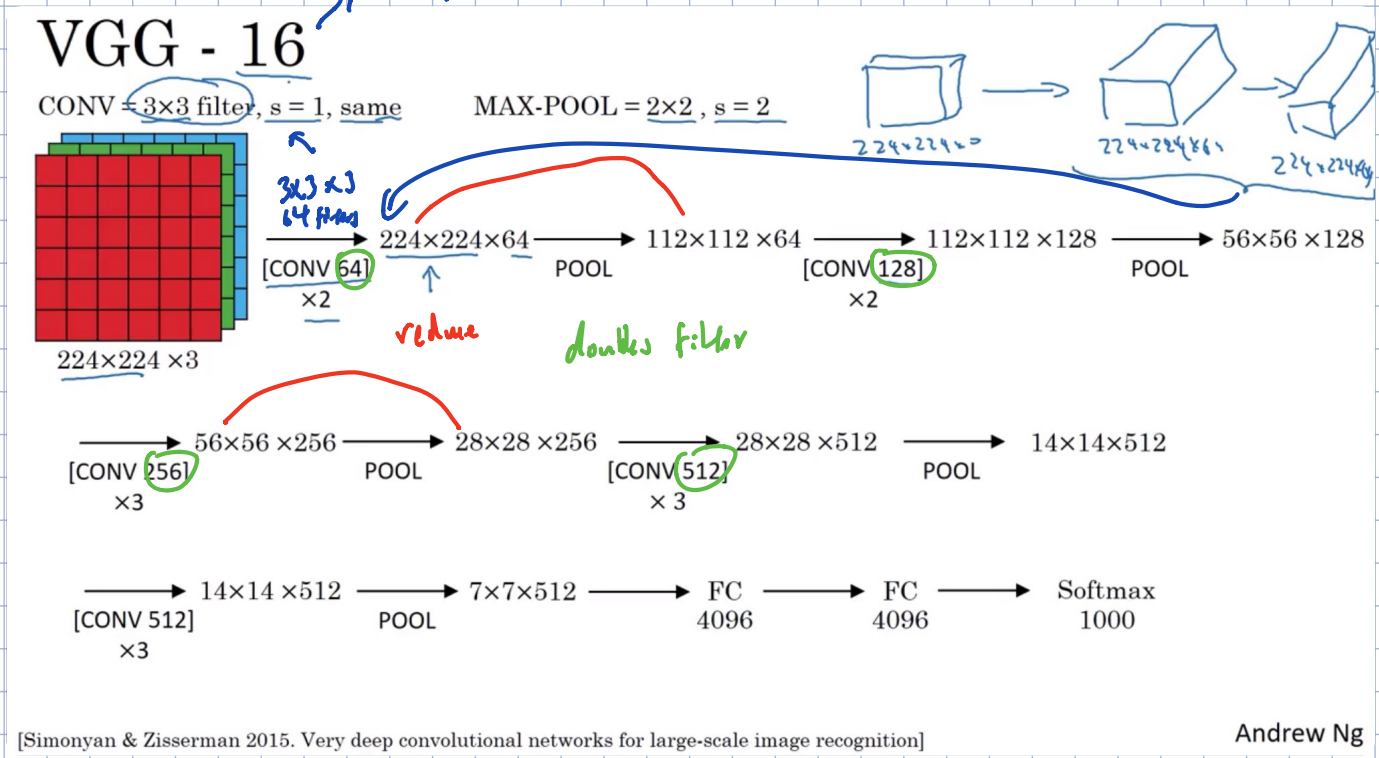
Alex Net



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

Andrew Ng

16 layers that have weights



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

Andrew Ng

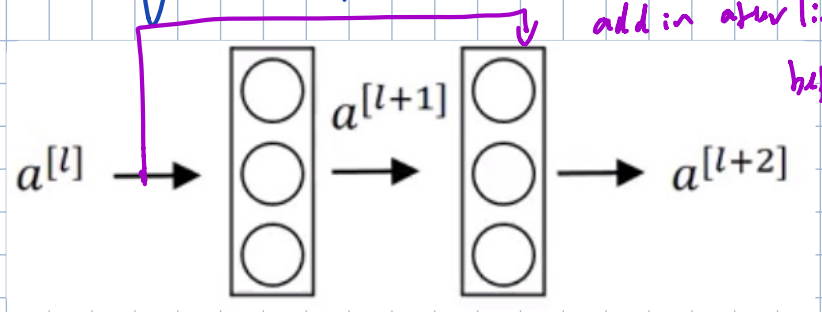
Residual Network (ResNets)

Very deep neural networks are difficult to train due to vanishing and exploding gradients. Here, we learn about skip connections which allows us to take the activations from one layer and feed them to another layer much deeper in NN.

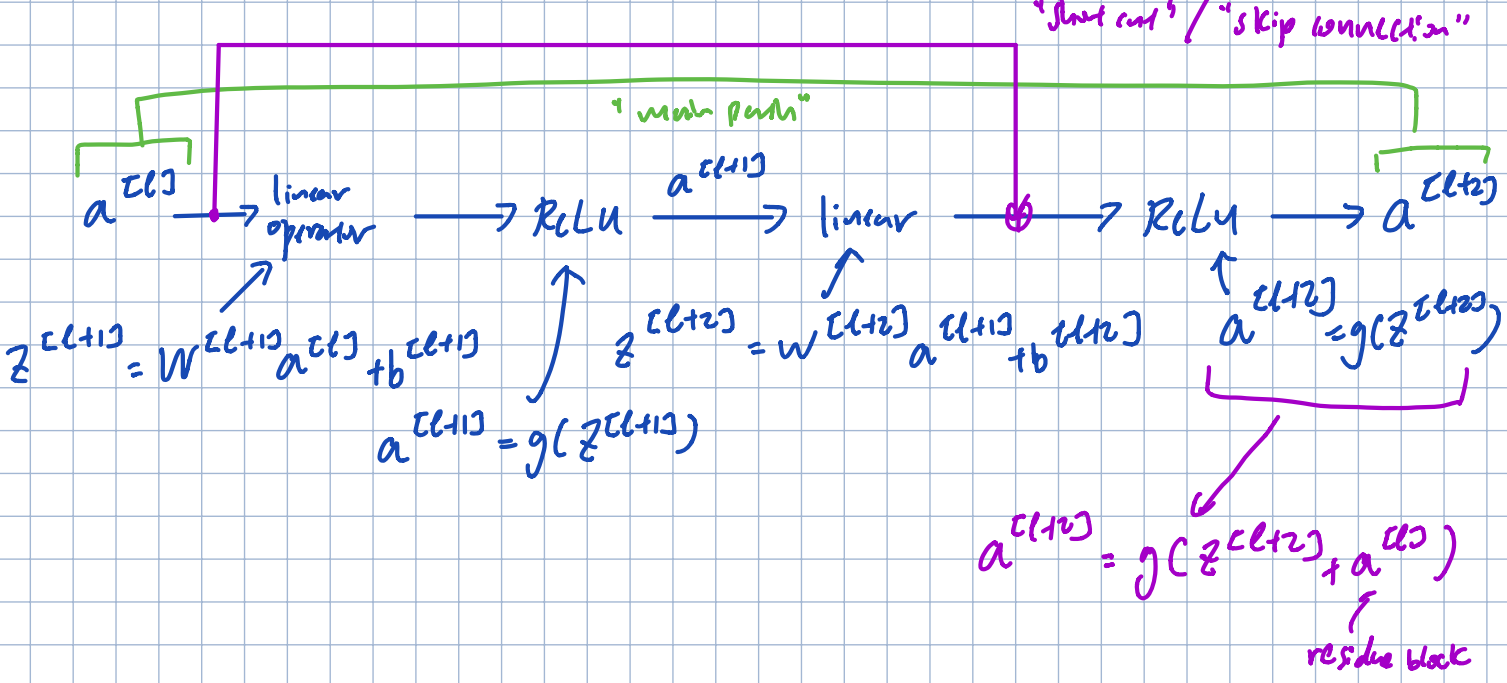
↳ we can build ResNet which we can train very deep networks.

Residual block

They are the building blocks of ResNet



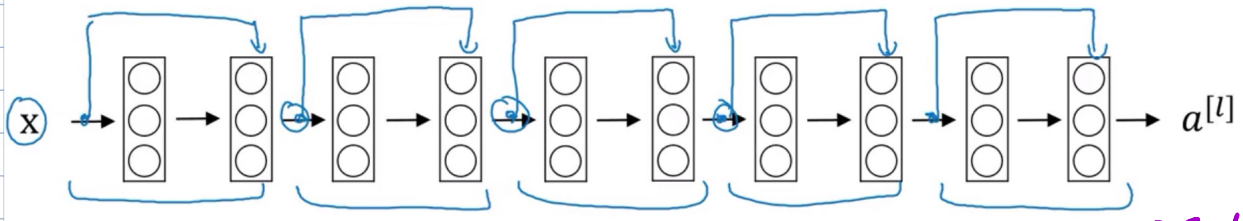
To go through steps above:



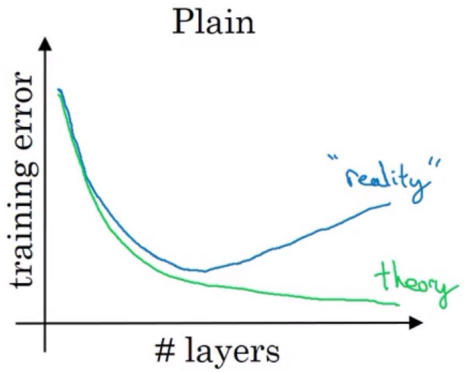
Residual Network

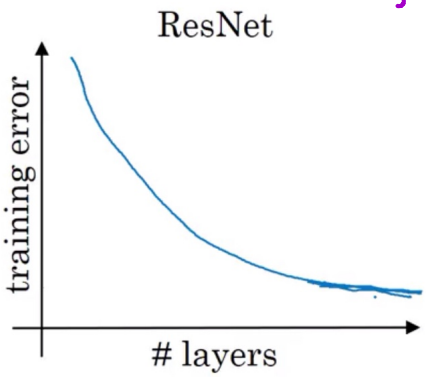
"Plain network"

training into a ResNet



5 Residual blocks





Why ResNets work?

e.g.

$X \rightarrow$ [Big NN] $\rightarrow a^{[l]}$

we want to go a bit deeper

$X \rightarrow$ [Big NN] $\rightarrow a^{[l]} \rightarrow$ [0 0 0] $\rightarrow$ [0 0 0] $\rightarrow a^{[l+2]}$

Res Block

use ReLu, i.e. $a \geq 0$

"same convolution"

can add up same dimension

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

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

$a^{[l+2]}$: if 256 dimension

$w^{[l+2]}$: regularization

if $w^{[l+2]} \rightarrow 0$, if $b^{[l+2]} \rightarrow 0$

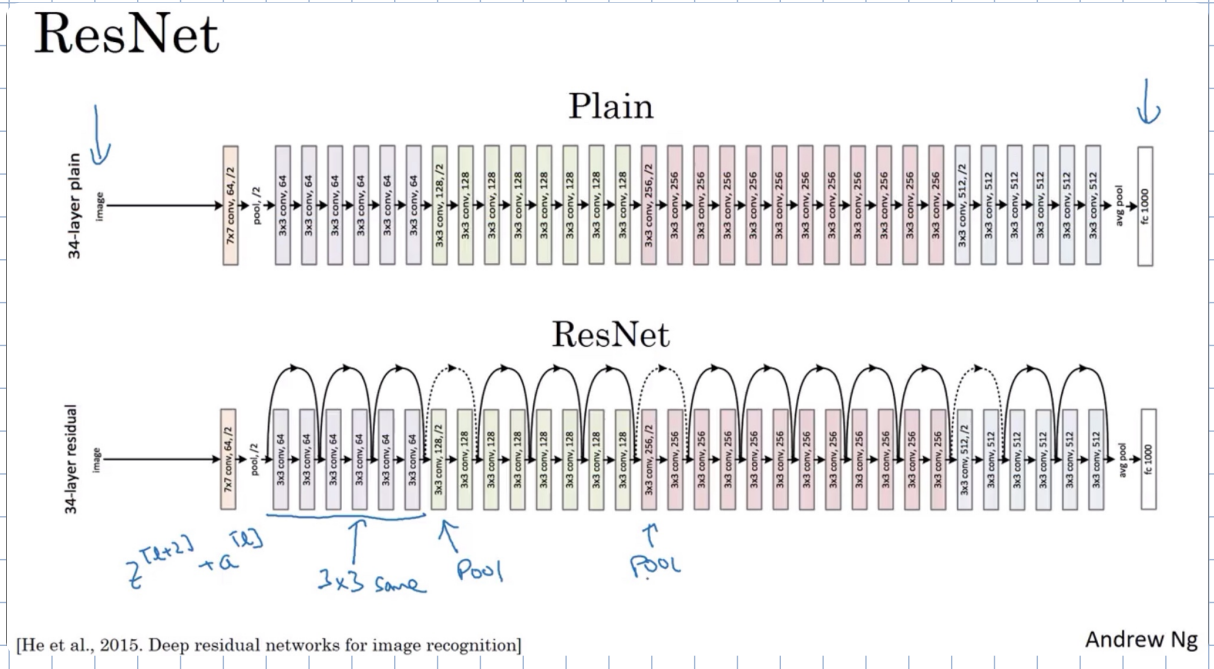
w_s: $\mathbb{R}^{256 \times 128}$; matrix of parameters; if 128 dimension

may add zero padding

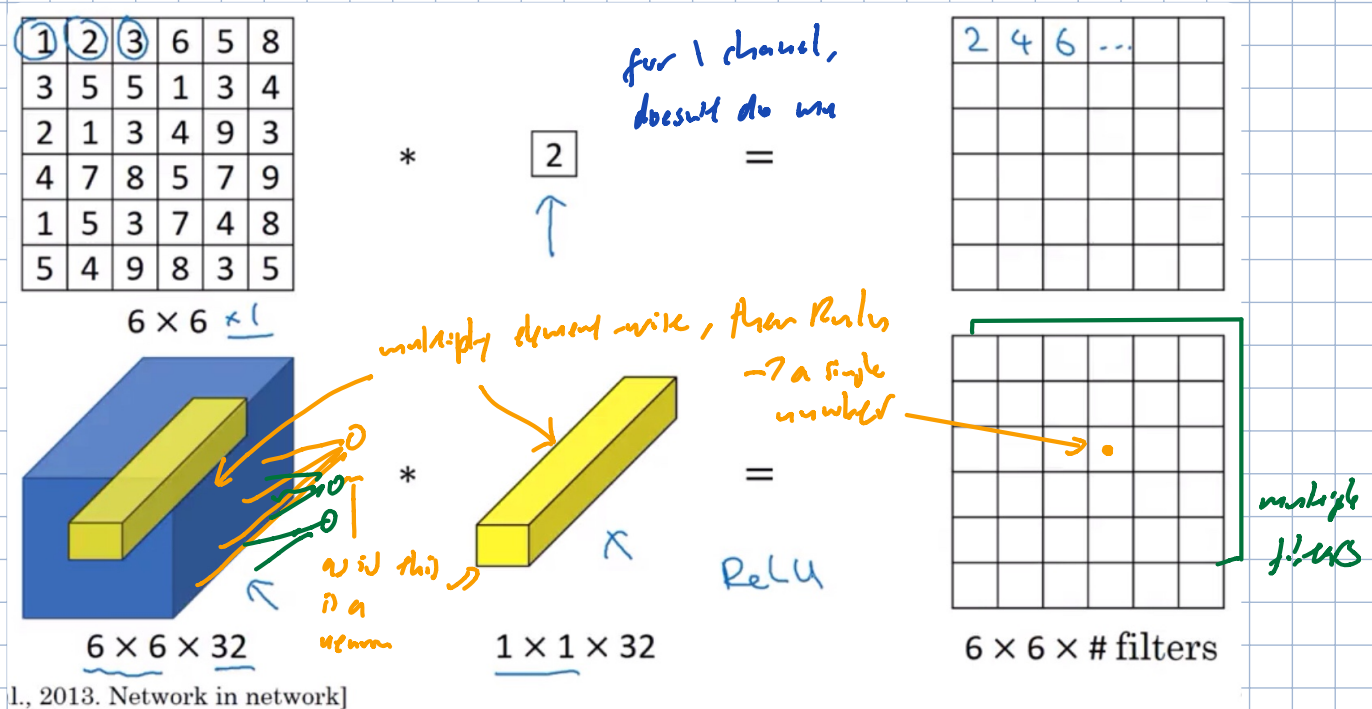
$= g(a^{[l]}) = a^{[l]}$

Identity function is easy for Residual block to learn

This means the extra 2 layers does not hurt the ability of a deeper NN.



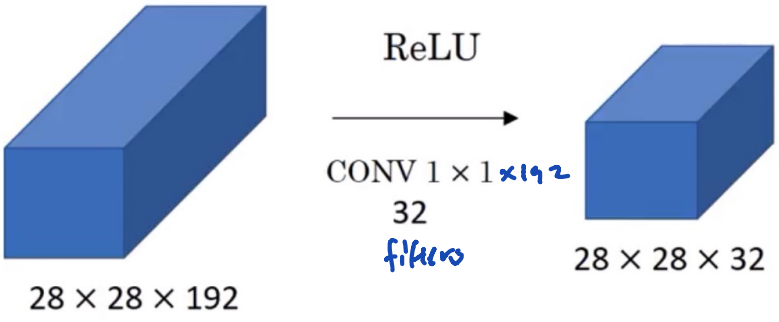
Networks in Networks and 1x1 Convolutions



called 1x1 convolution

or Network in Network

Using 1×1 convolutions



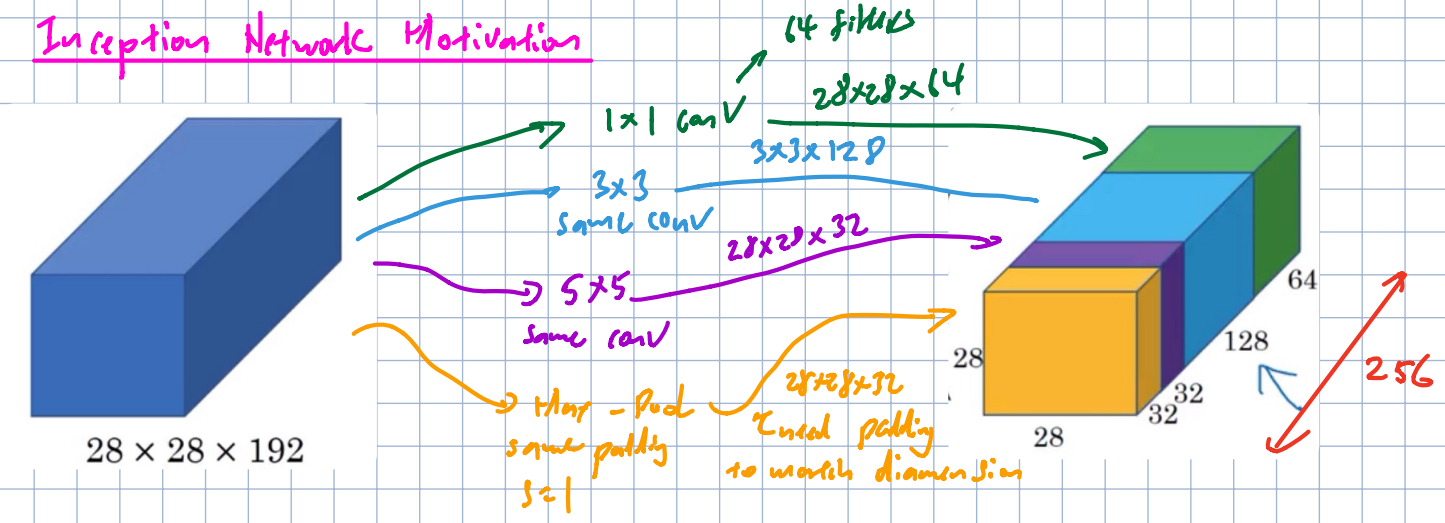
1×1 conv shrink n_c

pooling shrink n_H, n_W

say if we want to shrink height, width → use pooling layer

if we want to shrink # channels?

Inception Network Motivation



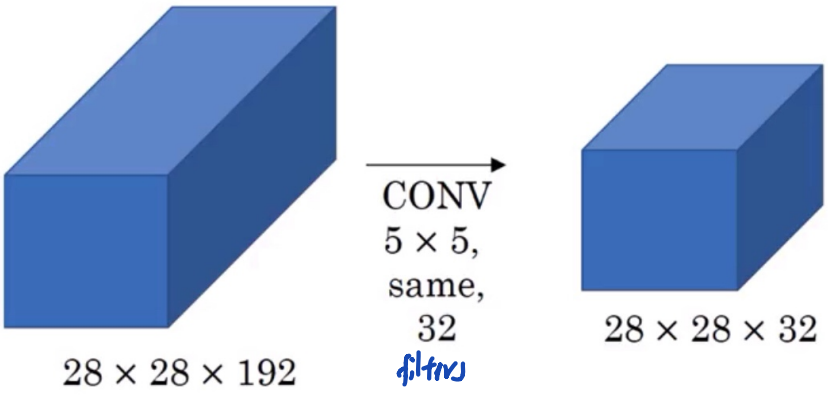
e.g. we input 28×28×192 volume, the inception layer do is instead of choosing what filter size we want in a conv layer, or even having a conv or pooling layer, we can do them all.

we have 1 inception module input 28×28×192, and output 28×28×256

↳ Basic idea: instead of pick out 1 filter size or pooling, we do them all and concatenate all the outputs. Let the network learn whatever parameters it wants to use, or whatever the combinations of these filter size it wants.

* But this description of inception has a problem: computational cost!

e.g. we focus on the 5×5 filter in the example



each filter is 5×5×192

↳ we need to compute

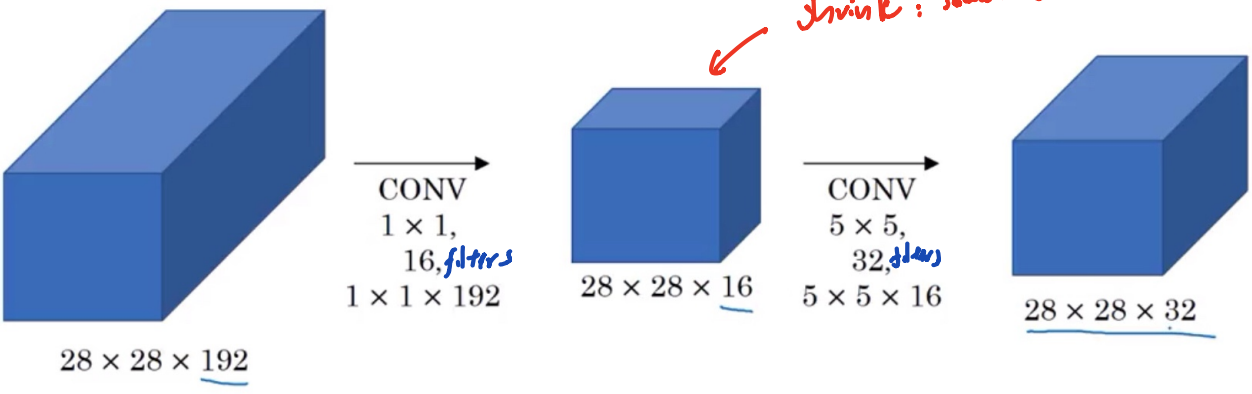
28×28×32 numbers

↳ for each number, need

5×5×192 multiplications

↳ total 120 million multiplications

Using 1×1 conv, we can reduce # of multiplications:



Intuition: sometimes called "Bottleneck layer"

Notice the output dimension are the same for the 2nd approach

To apply 1×1 conv, 16 filters, each is 1×1×192

28×28×16 × (192×1×1) = 2.4 mil ← first layer

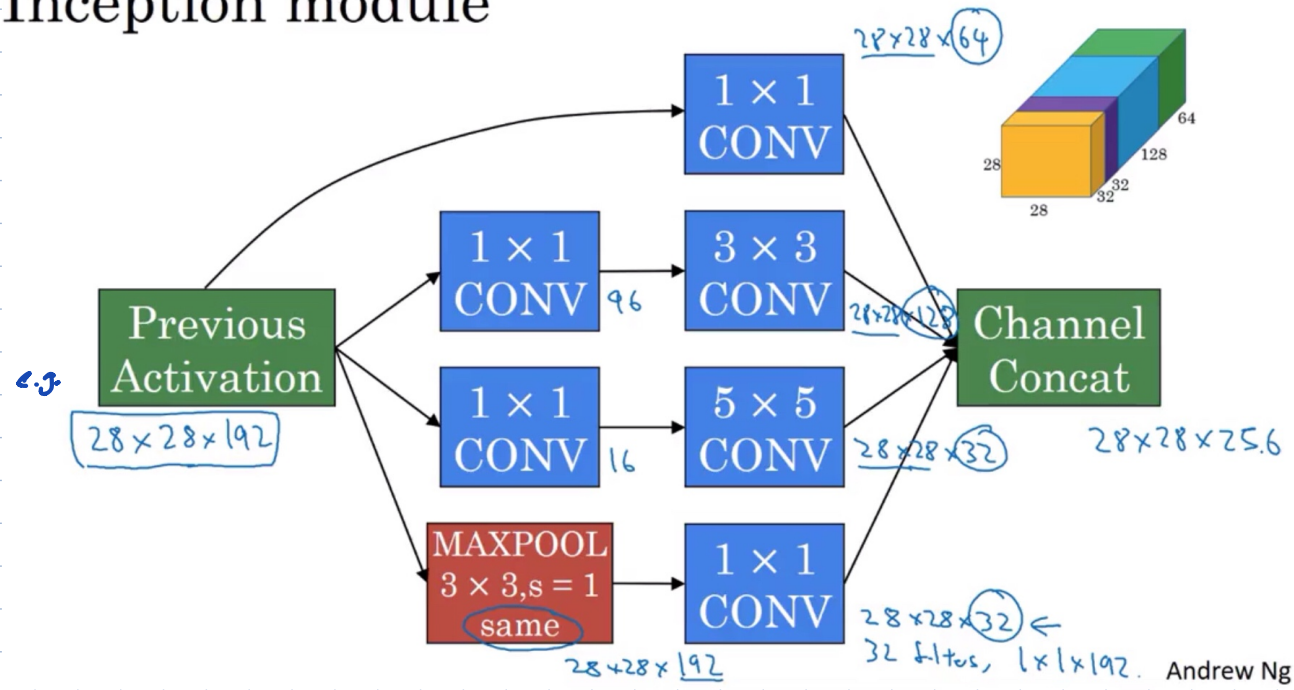
# of outputs

28×28×32 × (5×5×16) = 10.0 mil ← second layer

} 12.4 mil computations in total

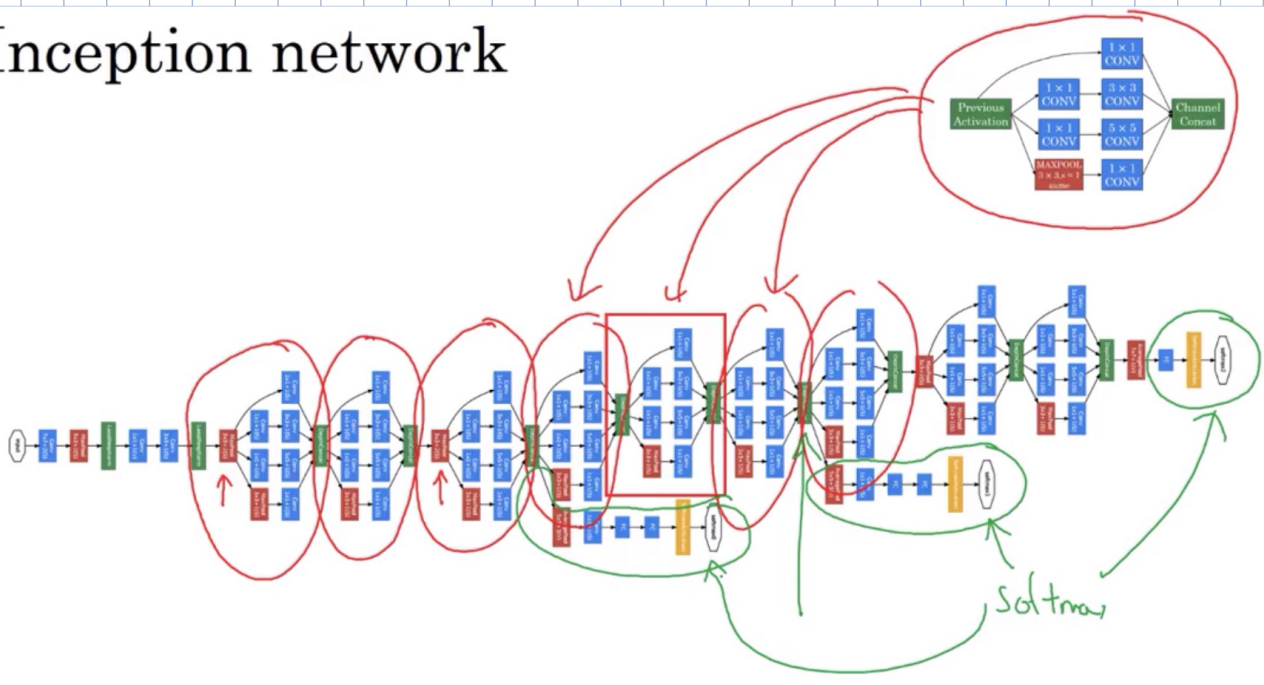
Inception Network

Inception module



Andrew Ng

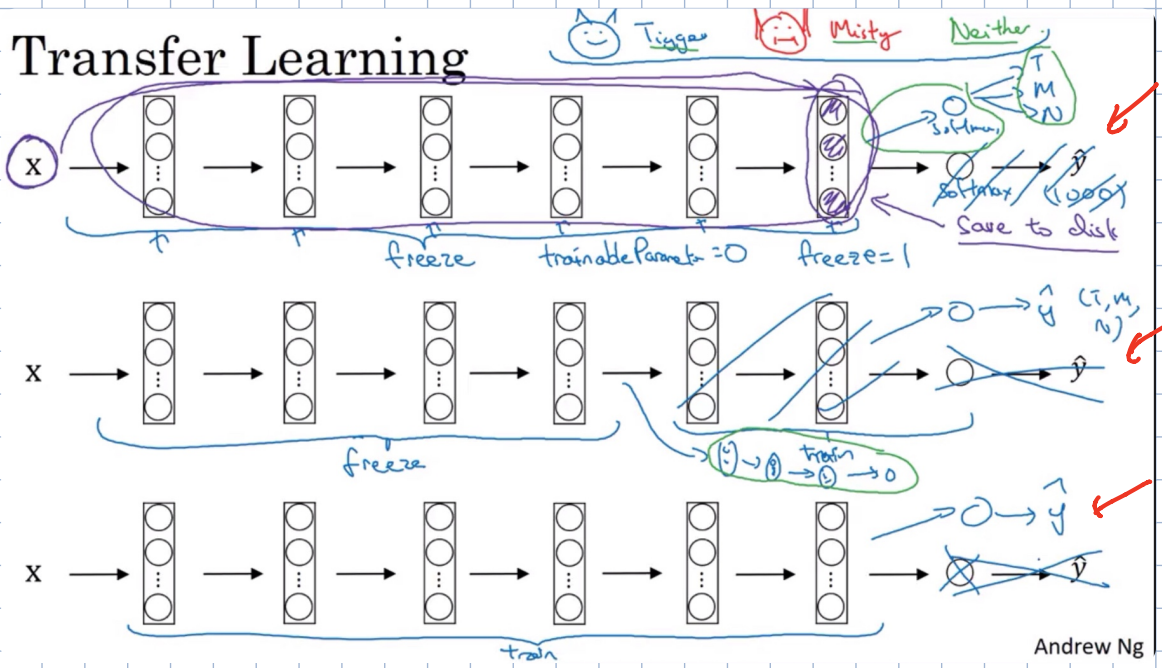
Inception network



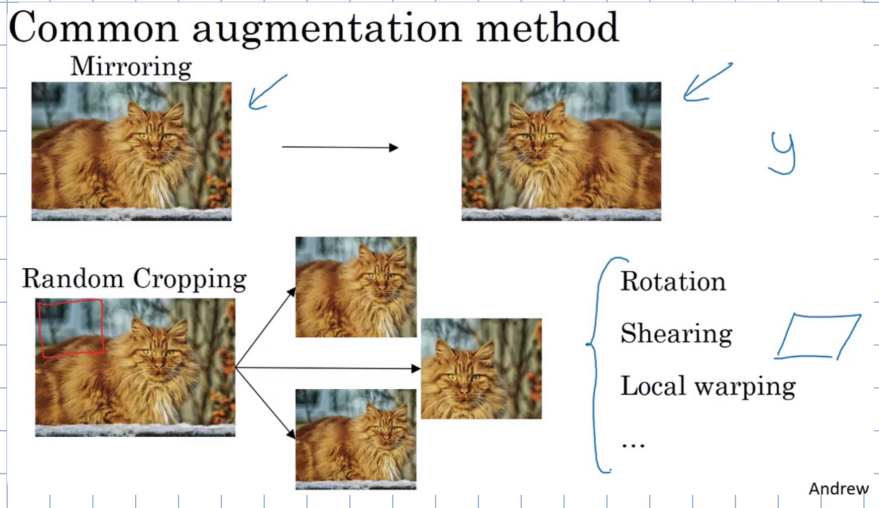
Szegedy et al., 2014, Going Deeper with Convolutions]

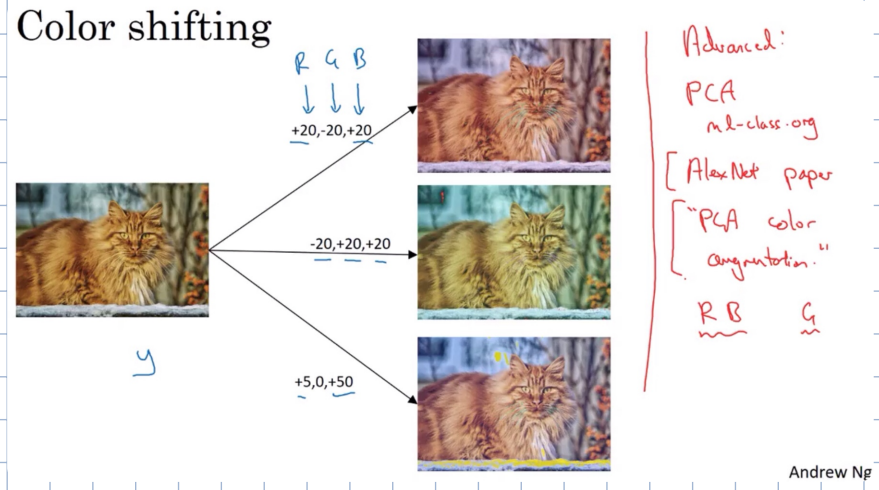
Andrew Ng

Transfer Learning

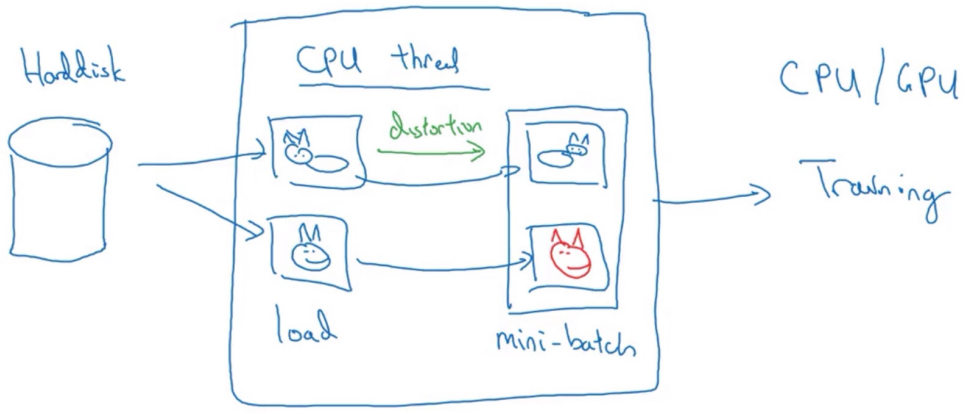


Data Augmentation



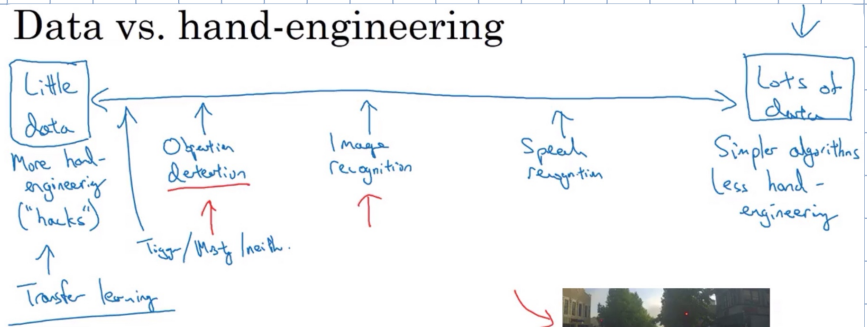


Implementing distortions during training



State of Computer Vision

Data vs. hand-engineering



Two sources of knowledge

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

Andrew Ng

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

Andrew Ng

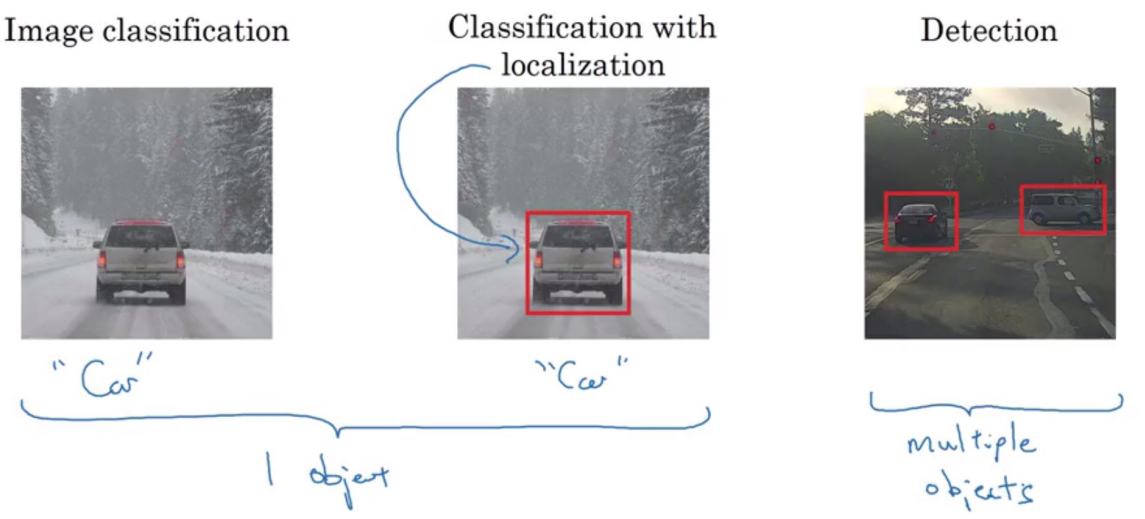
Use open source code

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

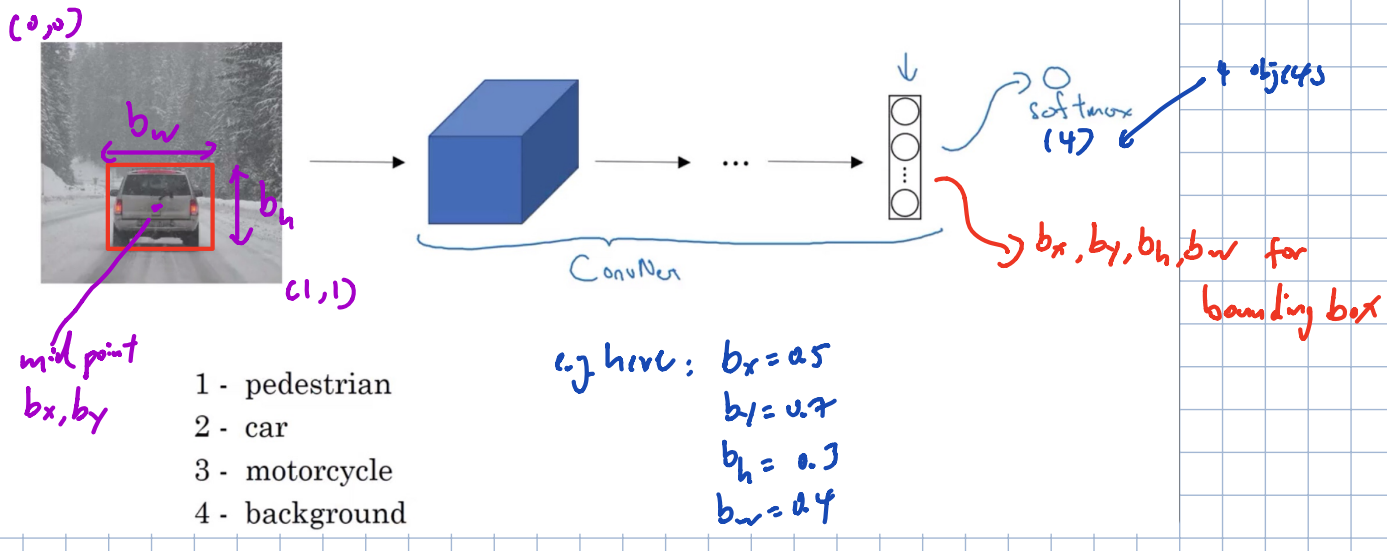
Week 3 : Object Detection

Object Localisation

What are localization and detection?



Classification with localization



1 - pedestrian
2 - car
3 - motorcycle
4 - background

e.g. here: $b_x = 0.5$
$b_y = 0.7$
$b_h = 0.3$
$b_w = 0.4$

Defining the target label y

1. pedestrain
2. car
3. motorcycle
4. background

Need to output b_x, b_y, b_h, b_w, class label (1-4)

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

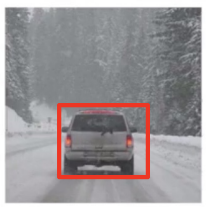
logistic regression

p_c: is there any object?
if object is class 1,2,3, then $p_c = 1$
if background or none, $p_c = 0$
Probability that there is an object (1 of the 3)

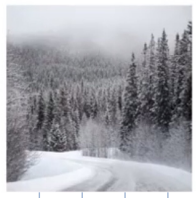
b_x, b_y, b_h, b_w → sq error

c_1, c_2, c_3 → * assume only 1 object → In practice, can use log-like loss

e.g.

$x =$ 

$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$x =$ 

$$y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$ don't care

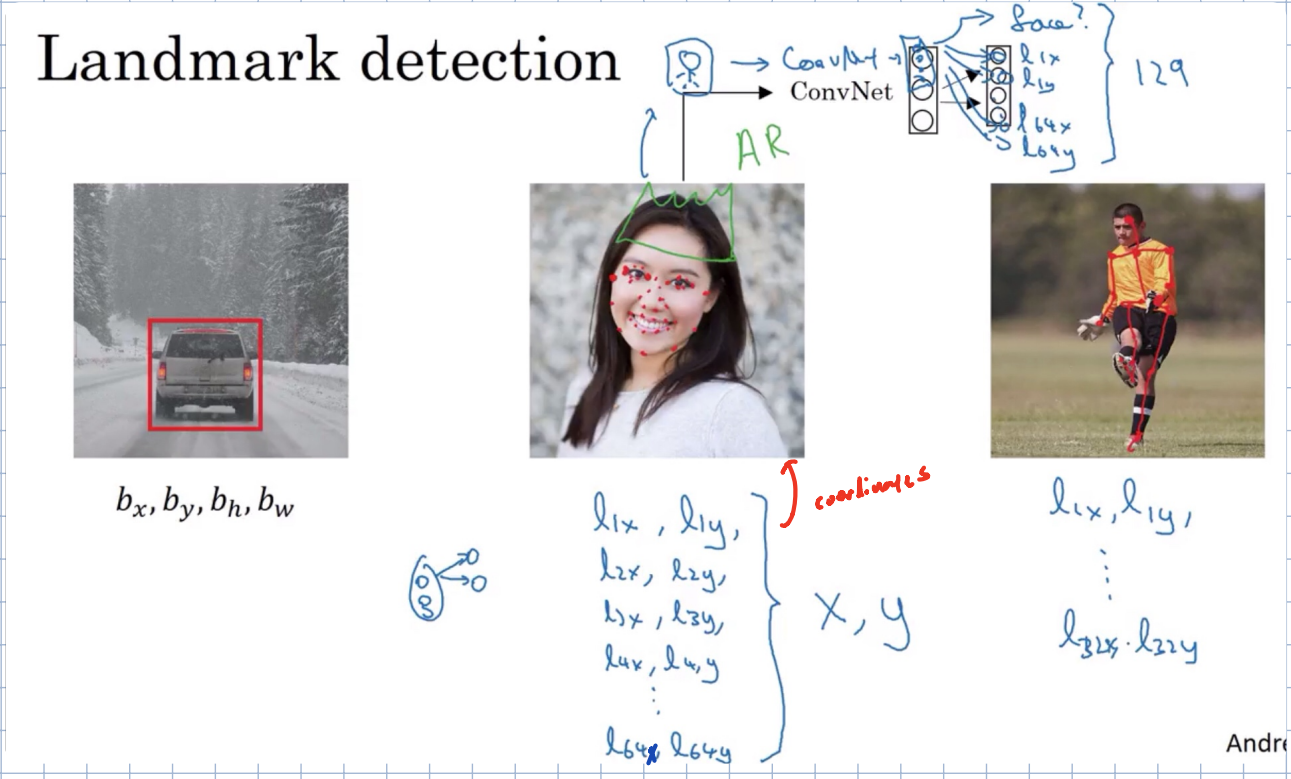
Loss Function

$\mathcal{L}(\hat{y}, y)$ (∵ y has 8 components here)

$$\begin{cases} = (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \cdots + (\hat{y}_8 - y_8)^2 & \text{if } y_1 = P_c = 1 \\ = (\hat{y}_1 - y_1)^2 & \text{if } y_1 = P_c = 0 \end{cases}$$

Landmark Detection

In general, we can have a NN just output x and y coordinates of important points and image (sometimes called landmark), that we want the NN to recognise.

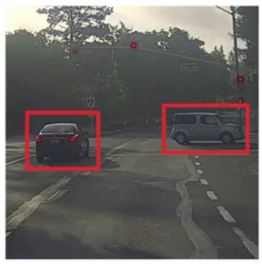


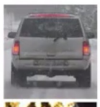
Landmarks has to be consistently defined across all images during labelling.

Object Detection

Car detection example

Training set:



x	y
	1
	1
	1
	0
	0

→ ConvNet → y

Once we trained up this ConvNet, we can use it in sliding window Detection

Sliding windows detection

→ ConvNet → 0

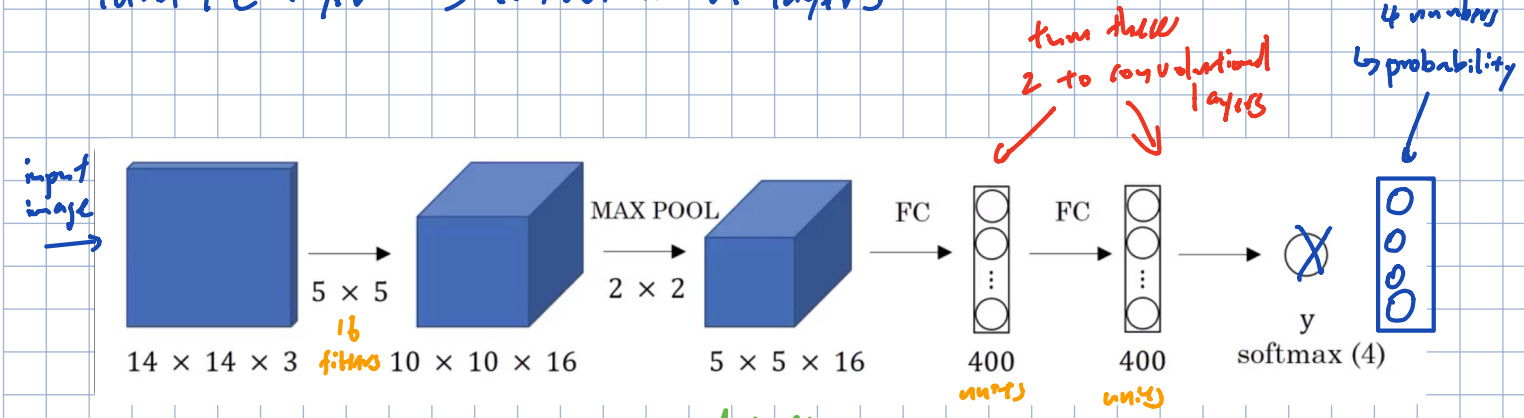
→ ConvNet

pick a window size

Disadvantage: computational cost

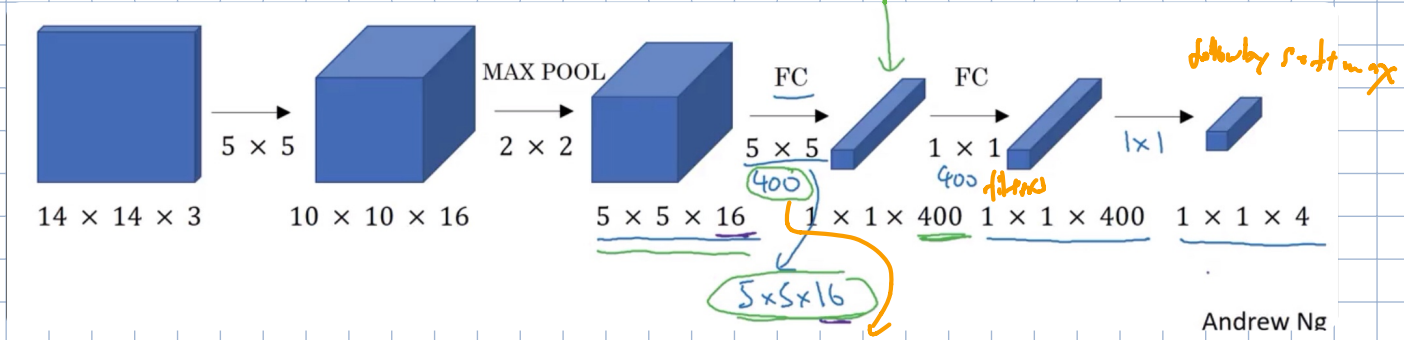
Convolutional Implementation of Sliding windows

Turn FC layer → Convolutional layers



mathematically, the same as some sliding window mapping

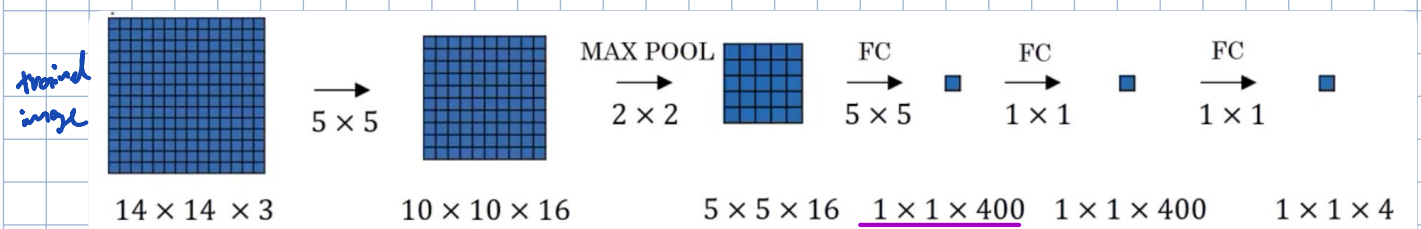
instead of flatten, view as 1×1×400 volume



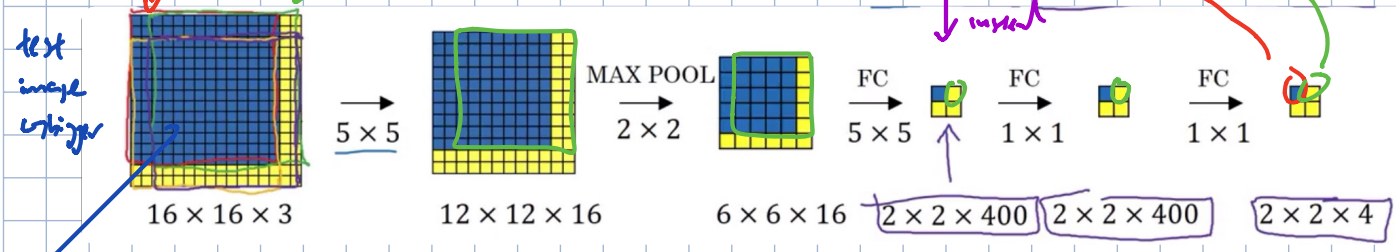
400 filters
5×5×16 ← look at all 16 channels

Convolution implementation of sliding windows

only front is drawn, they are volume



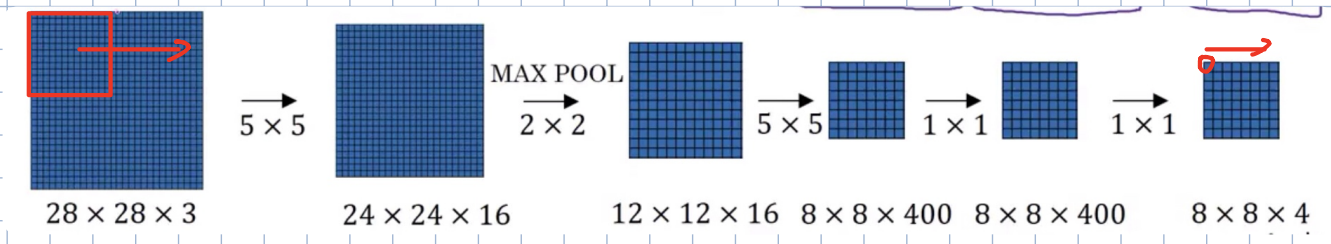
use previous instead

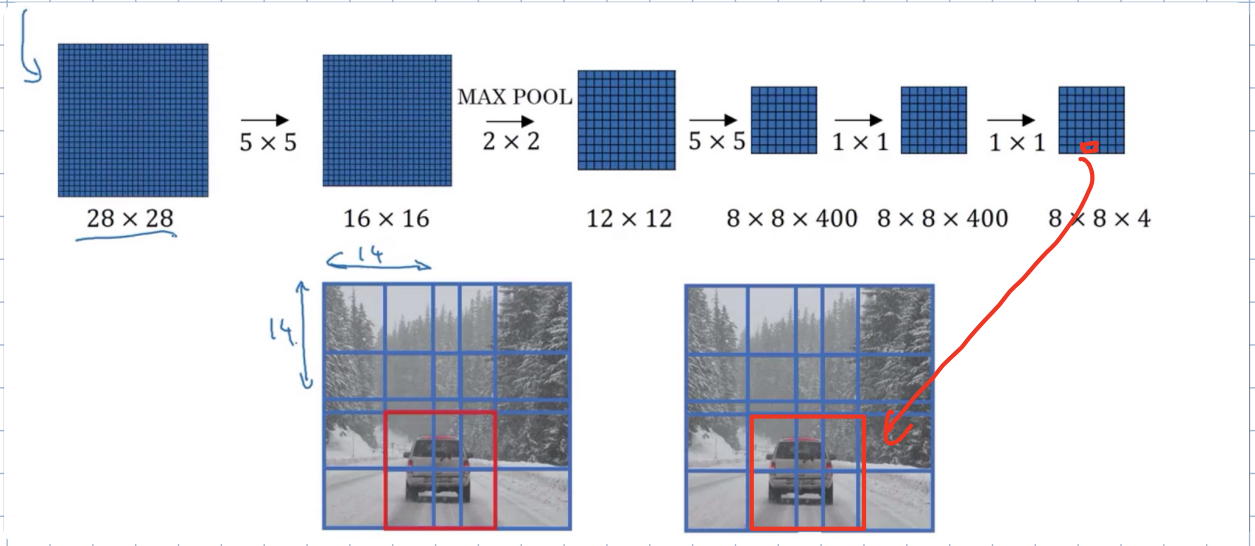


repeated computation in sliding windows → convolution allows sharing

instead of run 4 propagation
↳ run 4 at once

bigger test e.g.

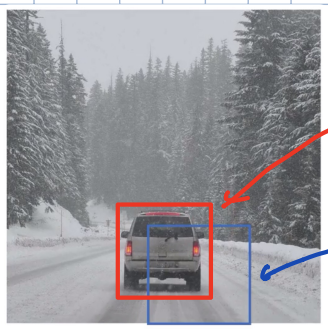




Instead of doing sequentially, to slide the window,
we can implement entire image, and make all predications at
the same time, i.e. 1 formal pass to predict (after training)
instead of each sliding window 1 formal pass to predict

However, the position of the bounding boxes are not too accurate

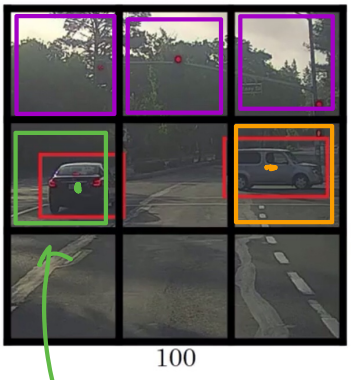
Bounding Box Prediction



ideally, should be this box, also may not be a square, best box might be a rectangle.

using sliding window, this might be the best box

YOLO algorithm



Basic idea: apply image classification and localisation algorithm, apply to each of the 9 grid cell.

How we define the labels for training?

For each grid cell, we specify label y:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

p_c: 0 or 1

c_1, c_2, c_3: 3 classes (not counting background)

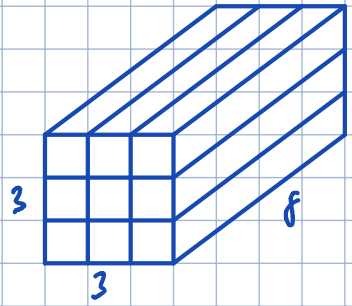
i.e. here we have a vector like this for each grid cell

$$\begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

pos of box

car

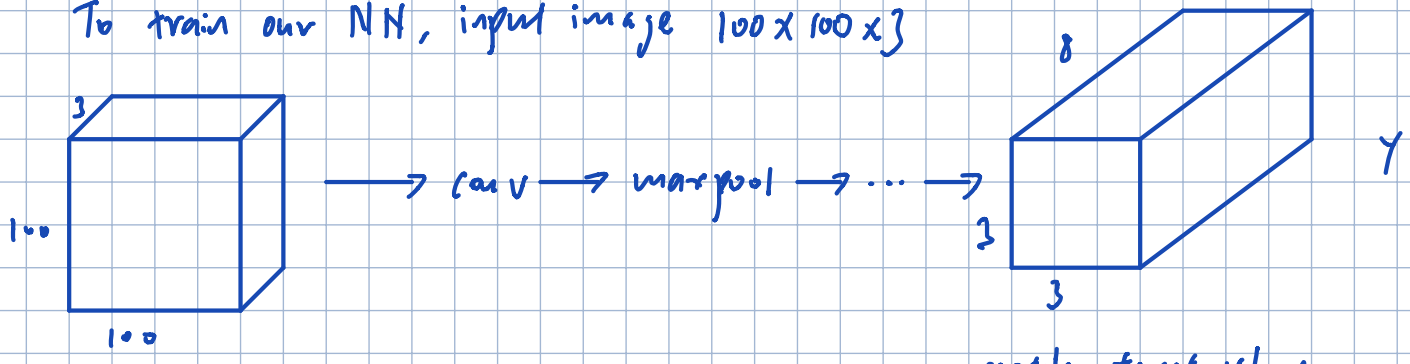
Total output volume: $3 \times 3 \times 8$ (target output)



each cell has a 8 dimensional vector

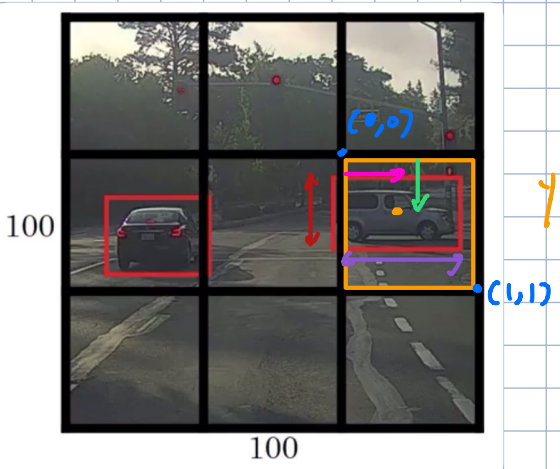
1 object is assigned to 1 grid cell (based on the location of the midpoint)

To train our NN, input image 100 x 100 x 3



match target volume

How do we encode bounding box (b_x, b_y, b_h, b_w)



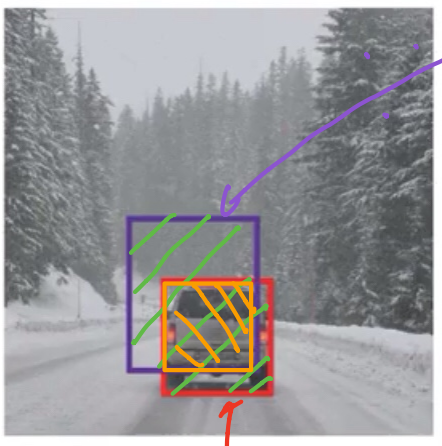
$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{matrix} 0.4 \\ 0.3 \\ 0.5 \\ 0.9 \end{matrix}$$

→ between 0 and 1

specified relative to the grid cell

↳ can be larger than 1

Intersection over Union



Intersection over Union (IoU)

$$= \frac{\text{size of } \square}{\text{size of } \square}$$

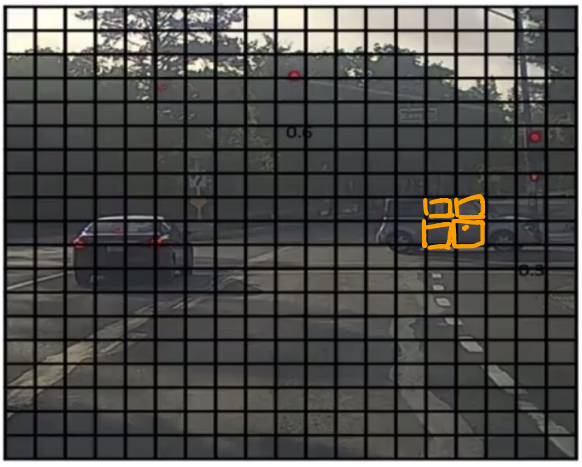
↳ "correct" if $IoU \geq 0.5$

↳ used to judge if the bounding box is correct or not

More generally, IoU is a measure of the overlap between 2 bounding boxes.

Non-max suppression

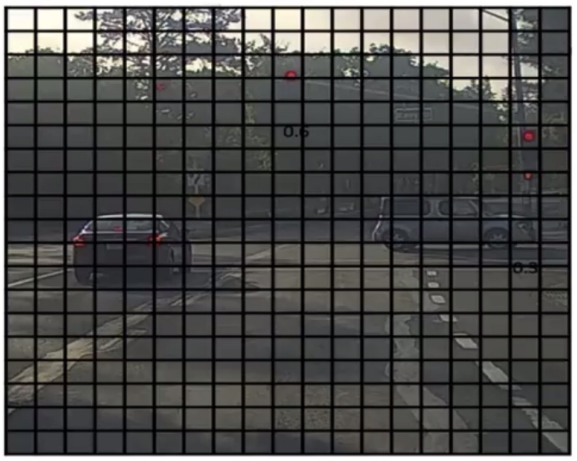
One of the problems of object detection is that algorithm may find multiple detections of the same objects.



multiple boxes think they got the center point.

multiple grid cells say my P_c is large
non-max suppression looks at the probability first
↳ take the largest one
↳ looks at remaining box and all those with high IoU
↳ suppress those box

Non-max suppression algorithm



19

19

e.g. 19x19x5 output volume

for simplification

each output prediction is:

$$\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$$

① Discard all boxes with $p_c \leq 0.6$, i.e. throw all low probability box

② while there are any remaining boxes:

↳ pick the box with largest p_c

↳ Output that as prediction

↳ discard any remaining box with IoU ≥ 0.5 with the box output (in the previous step)

→ boxes are either high p_c, or discarded

Anchor Boxes

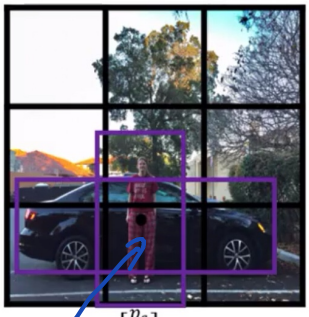
what if a grid cell want to detect multiple objects?

↳ use anchor boxes

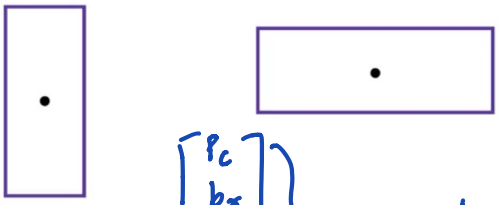
Overlapping objects:

associate 2 predications to 2 anchor boxes

predefined, ↳ box of prediction is more similar to which anchor box, then we use the parameter of that anchor box



Anchor box 1: Anchor box 2:



for this grid cell

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

3 classes

could output more than 1 class

instead →

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

anchor box 1 (predestrian)

anchor box 2 (car)

[Redmon et al., 2015, You Only Look Once: Unified real-time object detection] Andrew N

notice both midpoints of car and person are in the same grid cell.

Anchor box algorithm

Previously:

Each object in training image is assigned to grid cell that contains that object's midpoint.

Output y:

$3 \times 3 \times 8$

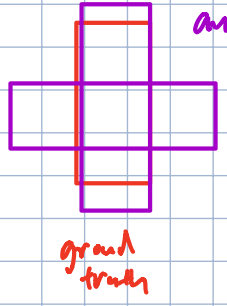
With two anchor boxes:

Each object in training image is assigned to grid cell that contains object's midpoint and anchor box for the grid cell with highest IoU.

output y:

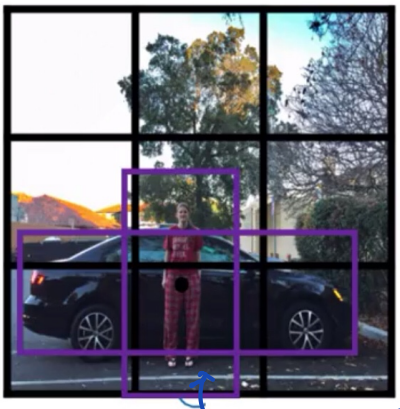
$3 \times 3 \times 16$ (2 boxes)

$3 \times 3 \times 2 \times 8$



→ object assigned to (grid cell, anchor box) pair

Anchor box example



Anchor box 1: Anchor box 2:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

pedestrain: $\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \end{bmatrix}$ car: $\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$

car only?: $\begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$ or $\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$

pedestrain more similar to anchor box 1

car more similar to anchor box 2

This algorithm don't handle well:

① 2 anchor box, 3 object in the same grid cell

② 2 object, same anchor box shape in same grid cell

} need tie breaker

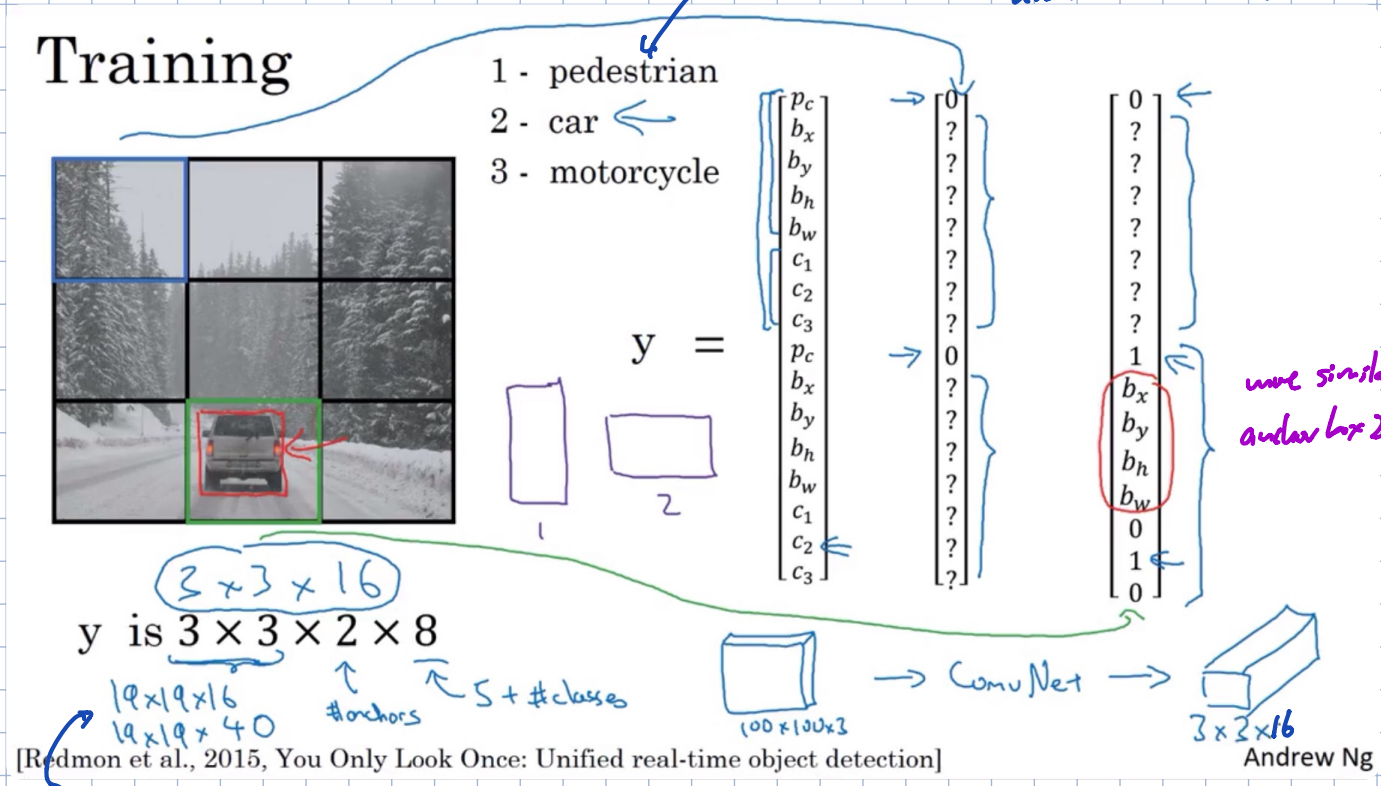
↳ quite rare if use finer grid cell.

↳ anchor box determination:

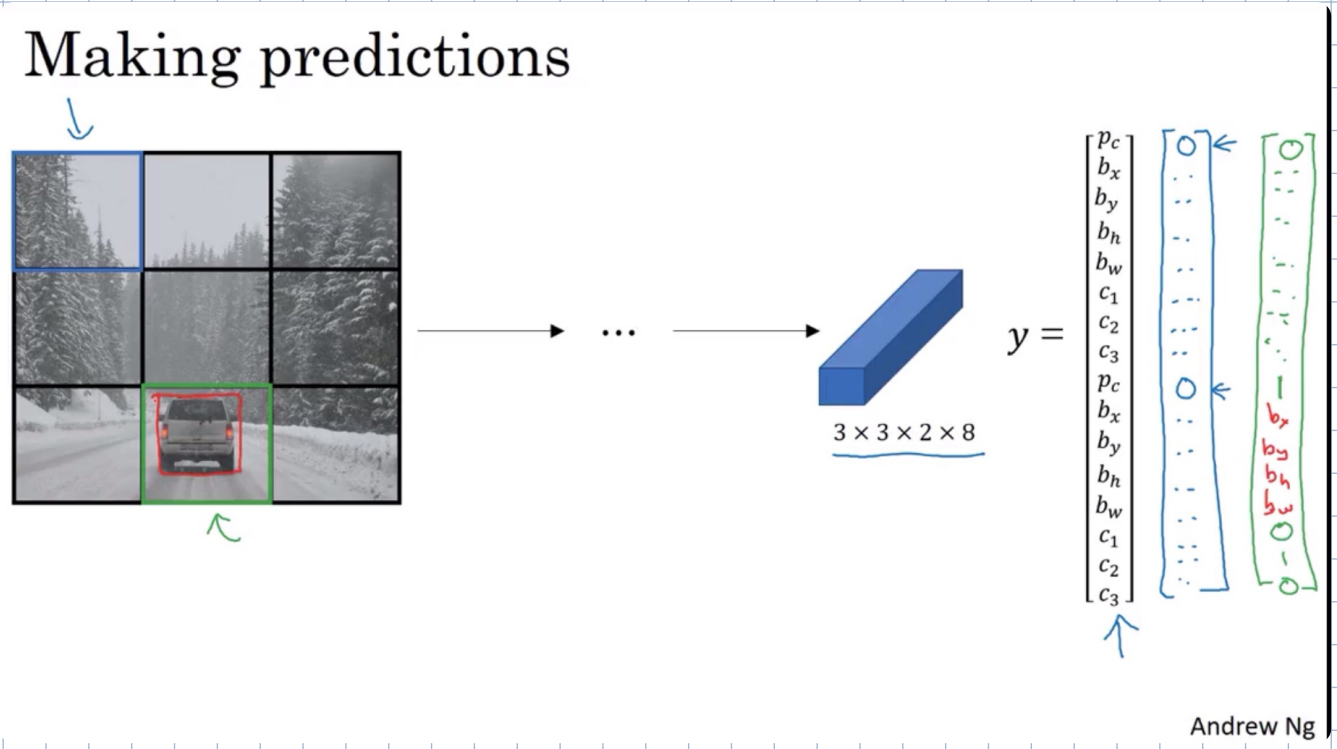
↳ used to be by hand

↳ now use k-means

YOLO Algorithm



[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

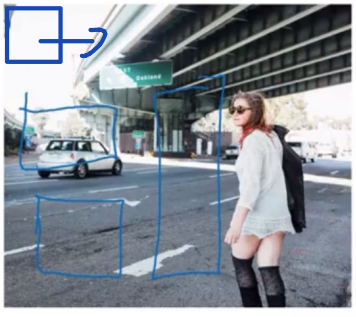


Outputting the non-max supressed outputs

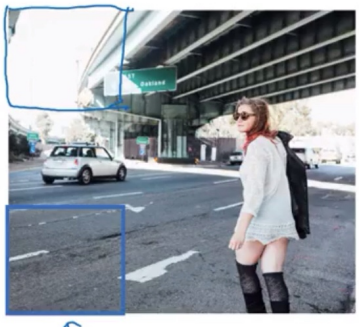
- For each grid ~~cell,~~ (cell) get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions.

Region Proposals

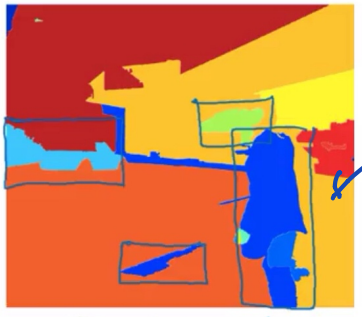
Region proposal: R-CNN (region)



sliding window



CNN
most boxes has no object



blob

Segmentation algorithm
~2,000 blobs
R-CNN

Girshik et. al, 2013, Rich feature hierarchies for accurate object detection and semantic segmentation] Andrew Ng

Faster algorithms

R-CNN: Propose regions. Classify proposed regions one at a time. Output label + bounding box.

Fast R-CNN: Propose regions. Use convolution implementation of sliding windows to classify all the proposed regions.

Faster R-CNN: Use convolutional network to propose regions.

[Girshik et. al, 2013. Rich feature hierarchies for accurate object detection and semantic segmentation]
[Girshik, 2015. Fast R-CNN]
[Ren et. al, 2016. Faster R-CNN: Towards real-time object detection with region proposal networks]

week 4: Face Recognition & Neural Style Transfer

what is face recognition?

Face verification vs. face recognition

→ Verification — 1:1 — 99% / 99.9

- Input image, name/ID
- Output whether the input image is that of the claimed person

→ Recognition — 1:K — K=100

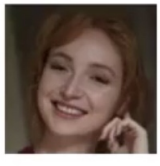
- Has a database of K persons
- Get an input image
- Output ID if the image is any of the K persons (or "not recognized")

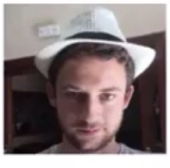
One shot Learning

For most face recognition applications, we need to recognise a person given just 1 single image → 1 shot learning

model only seen 1 image

same person?





say 4 employee in DB

people want to pass gate

Learning from one example to recognise the person again

we can try:

 → CNN → 0 softmax y (5 output)

training image size (only if too small)

Instead, we are going to let the model learn a "similarity" function

$d(\text{img1}, \text{img2})$ = degree of difference between images

small number: same person

big number: different person

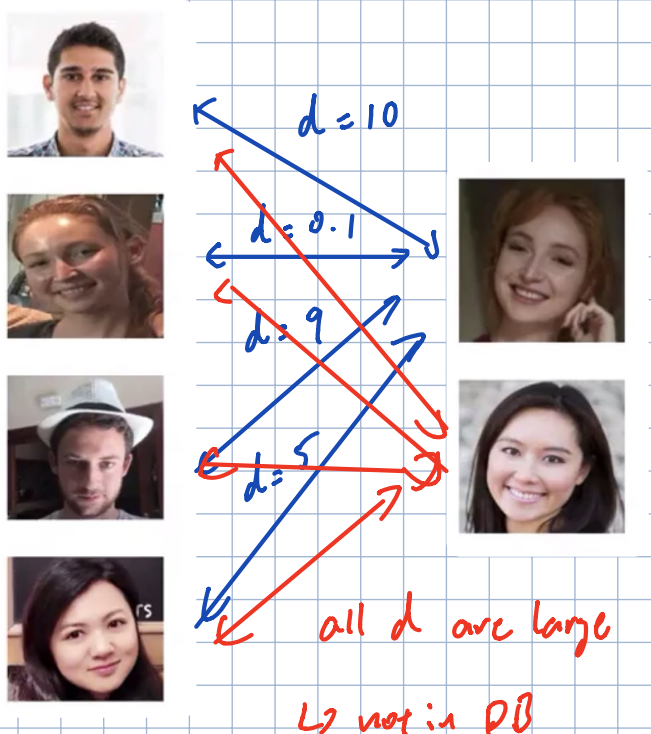
During recognition time, if

$d(\text{img1}, \text{img2}) \leq \tau$ → same person

$> \tau$ → different person

} face verification task

For face recognition task:

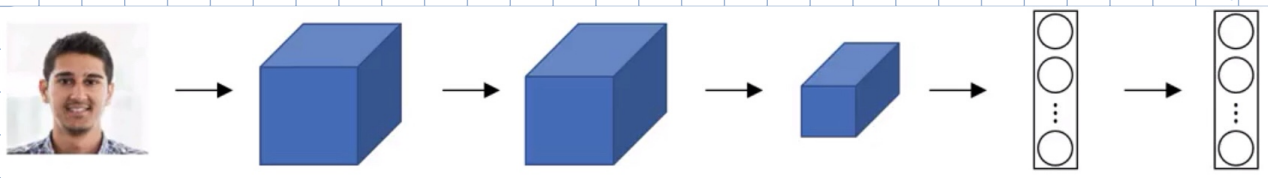


we have to learn

$d(\text{img1}, \text{img2})$

Siamese Network

e.g. 128 units

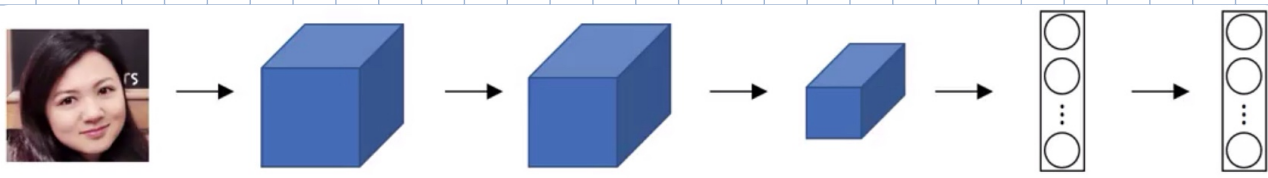


$x^{(1)}$

encoding of $x^{(1)}$: $f(x^{(1)})$

same NN with same parameters

128



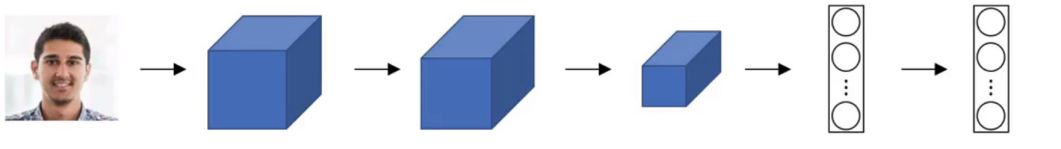
$x^{(2)}$

$f(x^{(2)})$

we have to make sure these 128 units are good representation of the pictures

then: $d(x^{(1)}, x^{(2)}) = \|f(x^{(1)}) - f(x^{(2)})\|_2^2$

Goal of learning



$f(x^{(1)})$

Parameters of NN define an encoding $f(x^{(i)})$ — 128

Learn parameters so that:

If $x^{(i)}, x^{(j)}$ are the same person, $\|f(x^{(i)}) - f(x^{(j)})\|^2$ is small.

If $x^{(i)}, x^{(j)}$ are different persons, $\|f(x^{(i)}) - f(x^{(j)})\|^2$ is large.

} use back prop, get parameters to make these are small

∴ different parameters → different encoding

Triplet Loss

To apply the triplet loss, we need to compare pairs of images




Anchor A Positive P Anchor A Negative N

we want encoding to be same

we want encoding to be different

we always look at 3 images at a time:

↳ Anchor, positive, negative images

Want: $\|f(A) - f(P)\|^2 \leq \|f(A) - f(N)\|^2$

$d(A, P)$ small value

$d(A, N)$ big value

↳ $\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 \leq 0 \Leftarrow -\alpha$

$f(P) = 0$, $f(N) = 0$

NN can actually make the 2 terms all 0 ⇒ trivially satisfy this equation

↳ $f(\text{img}) = \vec{0}$

↳ To prevent this from happening, we need to modify

Hence:

$$\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha \leq 0$$

α: margin

Triplet Loss function

This loss function is defined on triples of images:

Given 3 images: A, P, N

single triplet example

$$\mathcal{L}(A, P, N) = \max\left(\underbrace{\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2}_{\leq 0} + \alpha,\ 0\right)$$

so long we achieve the objective of making that ≤ 0, then loss $= 0$

if that thing > 0, we will have a positive loss

$$J = \sum_{i=1}^{m} \mathcal{L}(A^{(i)}, P^{(i)}, N^{(i)})$$

e.g. training set: 10K pics of 1K persons

take 10K pics to generate/select the triplets, then train NN

↳ we need multiple pic of the same person to train this model

∴ need (A, P) pair

How do we choose these triplets to form the training set?

During training, if A, P, N are chosen randomly,

$d(A, P) + \alpha \leq d(A, N)$ is easily satisfied.

$$\|f(A) - f(P)\|^2 + \alpha \leq \|f(A) - f(N)\|^2$$ ← chances are A, N are usually different

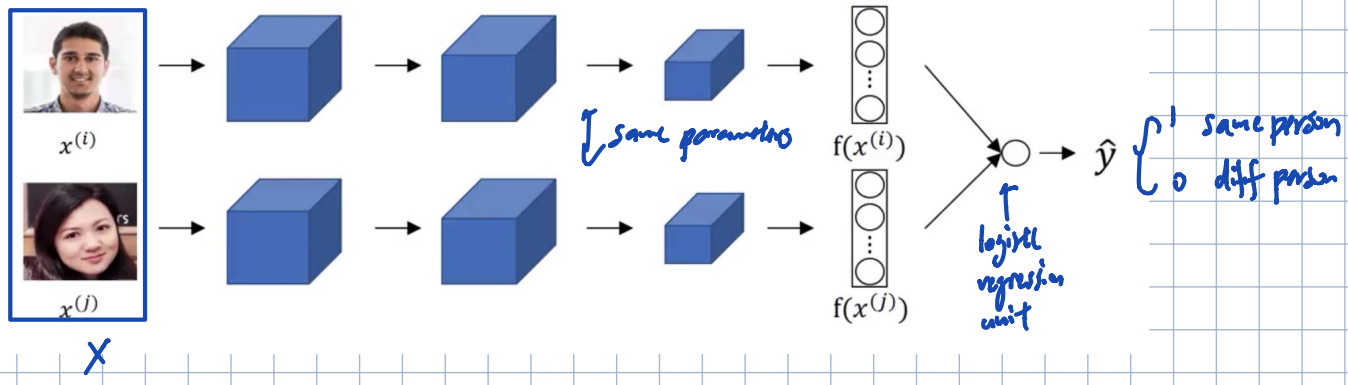
↳ choose triplets that are "hard" to train on.

$$d(A, P) + \alpha \leq d(A, N)$$

↳ $d(A, P) \approx d(A, N)$

Face verification and Binary classification

Learning the similarity function



$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_k \left| f(x^{(i)})_k - f(x^{(j)})_k \right| + b\right)$$

element-wise

or $\dfrac{(f(x^{(i)})_k - f(x^{(j)})_k)^2}{f(x^{(i)})_k + f(x^{(j)})_k}$ χ^2

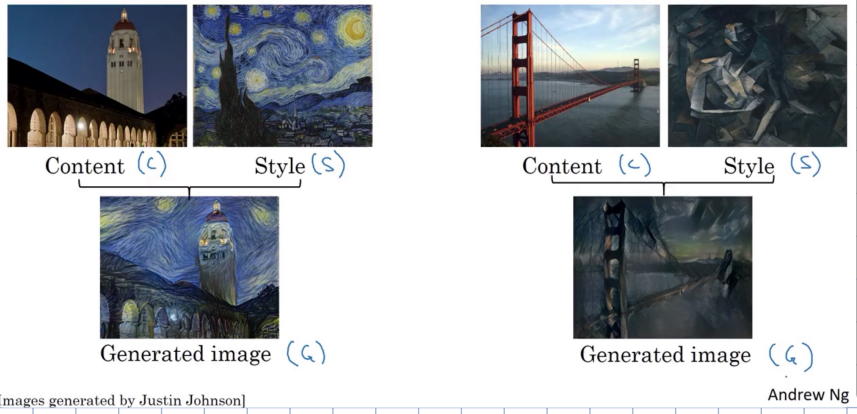
Face verification supervised learning

x	y	
	1	"Same"
	0	"Different"
	0	
	1	

[Taigman et. al., 2014. DeepFace closing the gap to human level performance]

Neural Style Transfer

Neural style transfer



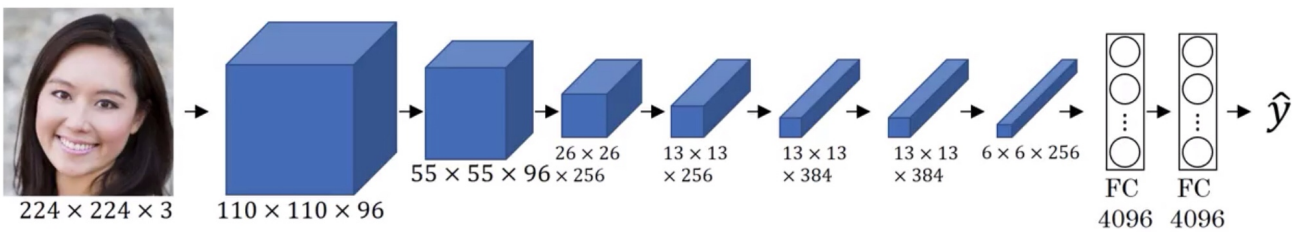
[Images generated by Justin Johnson]

Andrew Ng

we need to look at the features extracted by ConvNet at various layers.

What are deep ConvNets learning?

Visualizing what a deep network is learning



Pick a unit in layer 1. Find the nine image patches that maximize the unit's activation.

Repeat for other units.

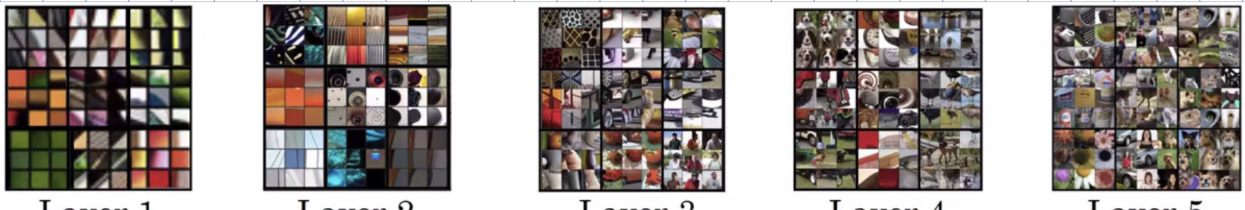
[Zeiler and Fergus., 2013, Visualizing and understanding convolutional networks]

Andrew Ng

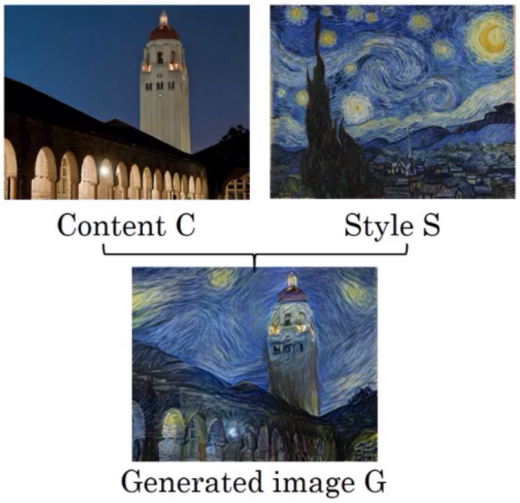
Later layers see larger patch

1st layer sees small patches

highly activated units



Neural Style Transfer: Cost function



$J(G)$ → a measure of how good is the image.

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

$J_{content}(C, G)$: how similar in content

$J_{style}(S, G)$: how similar in style

Find the generated image G

1. Initiate G randomly
   G: $100 \times 100 \times 3$ (RGB)
2. Use gradient descent to minimize $J(G)$

$$G := G - \frac{\partial}{\partial G} J(G)$$

initialise G randomly

atys et al., 2015. A neural algorithm of artistic style]

Andrew Ng

Content Cost Function

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

- Say you use hidden layer l to compute content cost.
- Use pre-trained ConvNet. (E.g., VGG network)
- Let $a^{[l](C)}$ and $a^{[l](G)}$ be the activation of layer l on the images
- If $a^{[l](C)}$ and $a^{[l](G)}$ are similar, both images have similar content

usually l is not too small and too large

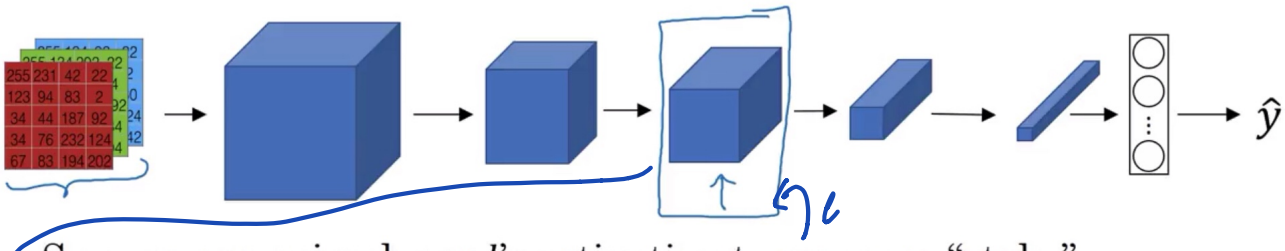
$$J_{content}(C, G) = \frac{1}{2} \| a^{[l](C)} - a^{[l](G)} \|^2$$

element-wise

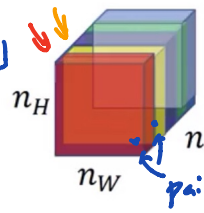
[assume unrolled into a vector]

Style Cost Function

Meaning of the "style" of an image



Say you are using layer l's activation to measure "style."
Define style as correlation between activations across channels.



How correlated are the activations across different channels?

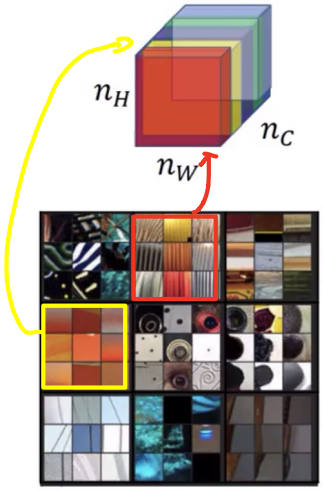
pair of numbers, → check correlation

[Gatys et al., 2015. A neural algorithm of artistic style]

Andrew Ng

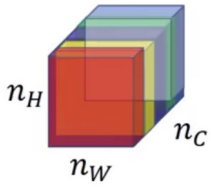
Intuition about style of an image

Style image



correlated: image wide vertical stripes usually occur together with orange tint in features

Generated Image



So correlation gives us info on how similar style between style image and generated image

Style matrix ← measures all the correlations

Let $a^{[l]}_{i,j,k}$ = activation at (i, j, k) (i: H, j: W, k: C)

$G^{[l]}$ is $n_c^{[l]} \times n_c^{[l]}$ ← style matrix

↳ $G^{[l]}_{kk'}$ measures the correlation between channel k and k' (where $k = 1, \dots, n_c^{[l]}$)

for 1 element in G

$$G^{[l](S)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](S)}_{ijk} a^{[l](S)}_{ijk'}$$

(S: style)

for generated image

$$G^{[l](G)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](G)}_{ijk} a^{[l](G)}_{ijk'}$$

"gram matrix" in linear algebra

$$J^{[l]}_{style}(S, G) = \left\| G^{[l](S)} - G^{[l](G)} \right\|^2_F$$

$$\text{normalize} = \frac{1}{(2 n_H^{[l]} n_W^{[l]} n_c^{[l]})^2} \sum_k \sum_{k'} \left(G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'} \right)^2$$

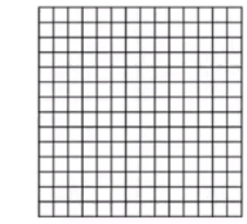
overall style cost function

$$J_{style}(S, G) = \sum_l \underbrace{\lambda^{[l]}}_{\text{weights: hyperparameters}} J^{[l]}_{style}(S, G)$$

recap:

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

1D and 3D Generalisations



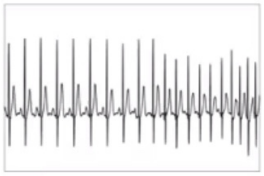
2D input image
14 × 14

*

2D filter
5 × 5

14x14x3 * 5x5x3

→ 10x10



*

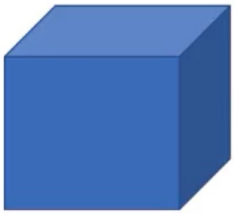
1	20	15	3	18	12	4	17

1	3	10	3	1

14 * 5

→ 10

3D convolution



*

3D volume

3D filter

input 14x14x14 x 1 ← channel

* 5x5x5 x 1 ← filter

→ 10x10x10 x 16 ← # of these are 16 filters