Computer Vision.

e.g. Image classification.

Object detection.

Neural Style Transfer.

Deep learning on large images:

$64 \times 64 \times 3 = 12288$. it's ok.

↑ 3 RGB channel.

$1000 \times 1000 \times 3 = 3$ million. which mean the input has 3m x_i.

Suppose next layer has 1000 units.

Then the first weight matrix W will be $1000 \times 3m$ matrix. (HUGE).

So we need CONVOLUTION technique.

Edge detection example.

Given a picture.

① Detect vertical edges.

② Detect horizontal edges.

Vertical edge detection.

element-wise product.

$3\times1+1\times1+2\times1+0\times0+0\times0+0\times1+1\times(-1)+8\times(-1)+2\times(-1)=-5$

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

$*$

Kernel matrix.

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

3×3 filter.

↑ Convolution operation

$=$

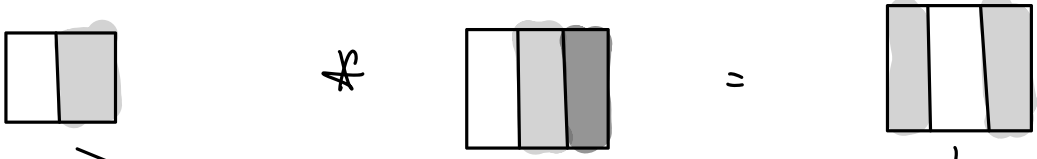
-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

4×4 result.

6×6 gray scale image.

Code e.g. Python: conv-forward. TF: tf.nn.conv2d. keras: Conv2D.

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

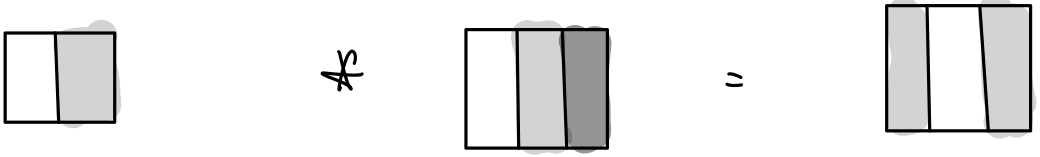


Result as the vertical edge in the middle.

More edge detection.

Previous example.

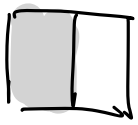
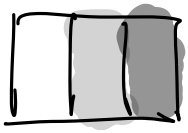
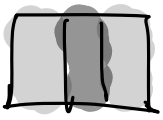
$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$



What about we reverse the pos. of 10 and 0.

$$\begin{bmatrix} 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \end{bmatrix}$$

It looks darker.

Means from dark to light.

Vertical and Horizontal Edge detection.

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

lighter top.

darker bottom.

Vertical. Horizontal.

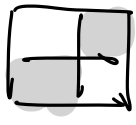
$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 30 & 10 & -10 & -30 \\ 30 & 10 & -10 & -30 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

positive edge

neg. edge

blending edge.

$$\begin{bmatrix} 10 & 10 & 0 \\ 10 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix}$$




Learning to detect edges.

$$\begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 3 & 0 & -3 \end{bmatrix}$$

2	0	-2	10	0	-10
1	0	-1	3	0	-3

Sobel filter. Scharr filter.

More weight on middle.

Actual, you can learn all these 9 numbers. Treating them as params

Padding.

We see: $(6\times6) * (3\times3) = (4\times4)$.

Math: $(n\times n) * (f\times f) = [(n-f+1)\times(n-f+1)]$

When looking the corner, these pixels only being used once.

Edge pixels received less attention. (Throw informations).

Solution: pad the image with additional border.

6×6 image into 8×8 image.

And $(8\times8) * (3\times3) = (6\times6)$. Preserved the original image size.

Valid and Same Convolution.

"Valid". $n\times n * f\times f \rightarrow n-f+1 \times n-f+1$.

no padding.

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

P: padding size. Say the kernel is f×f.

So if $p = \frac{f-1}{2}$. Then the image will preserve to be the same size.

e.g. $f=3$. $p=\frac{3-1}{2}=1$. $f=5$. $p=\frac{5-1}{2}=2$.

By convention, the width of fitter is usually odd.

(So that you can have a central pixel. to talk about position)

Strided Convolution.

We take element-wise product as usual. But we take a bigger step.

$(n \times n) * (f \times f)$.

padding p. Stride s.

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

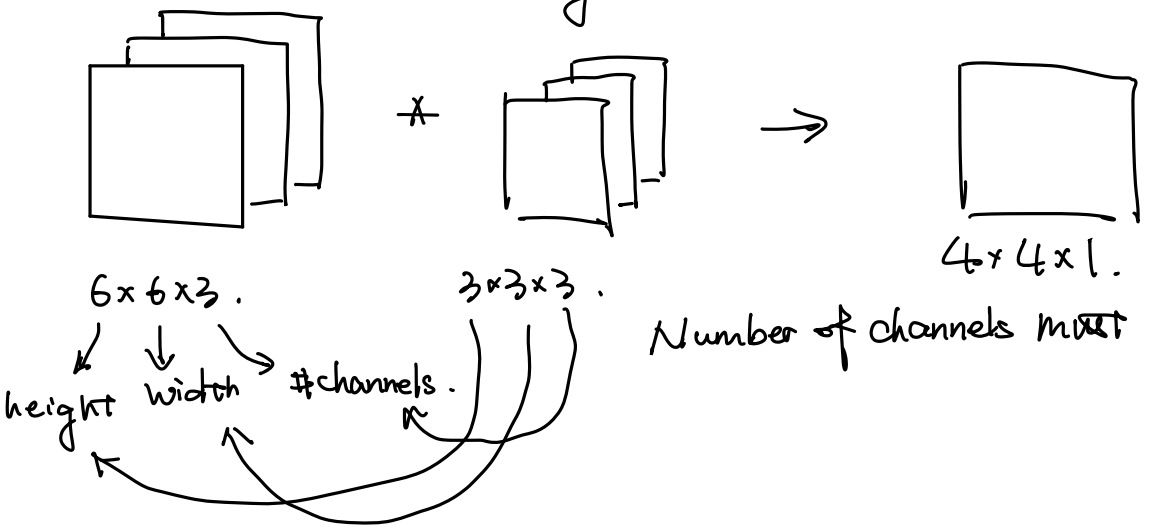
e.g. $(7 \times 7) * (3 \times 3)$: $\frac{7+0-3}{2} + 1 = 3$. final 3×3 result.

padding 0. Stride 2.

We can add floor operation $\lfloor \frac{n+2p-f}{s} \rfloor$. By convention, if the image was not fully land (include padding) inside the filter. we just don't do operation.

Convolutions over volumes.

Convolution on RGB images.



Number of channels must must match.

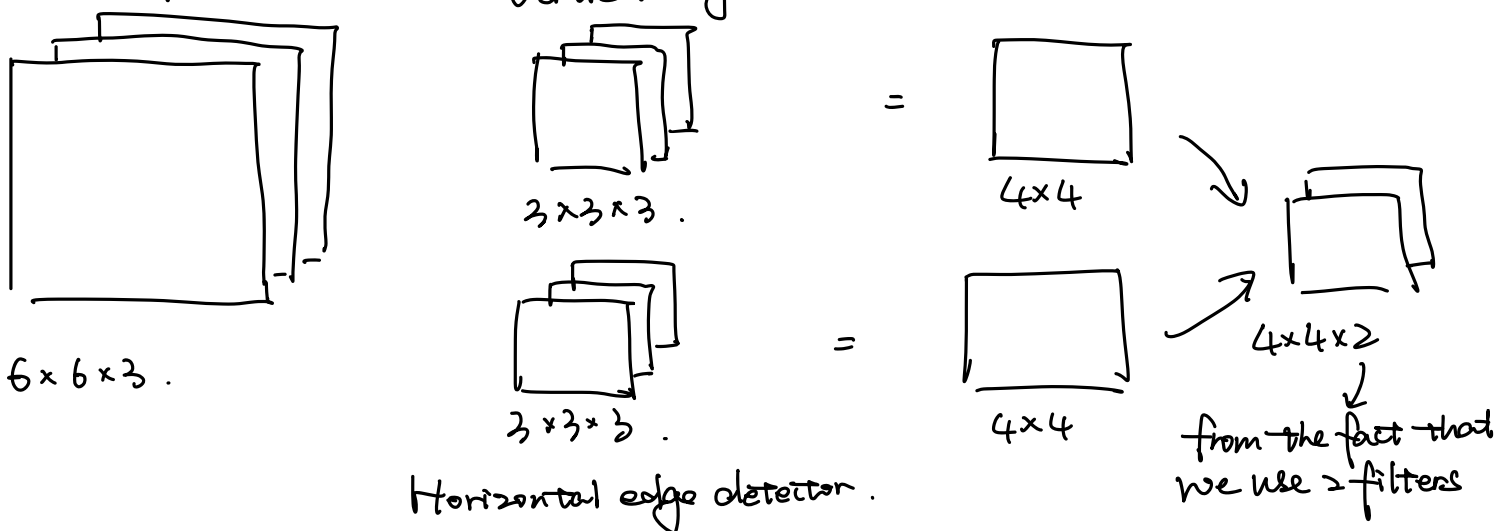
Consider $3 \times 3 \times 3 = 27$ numbers in the filter. Put this "Cube" in the 3 channel image. Element-wise product. Summation. $\Rightarrow$ Result.

e.g. You might want to detect red edges.

R: $\begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}$ G: $\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$ B: $\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$.

Edge detector only in red.

Multiple filters.



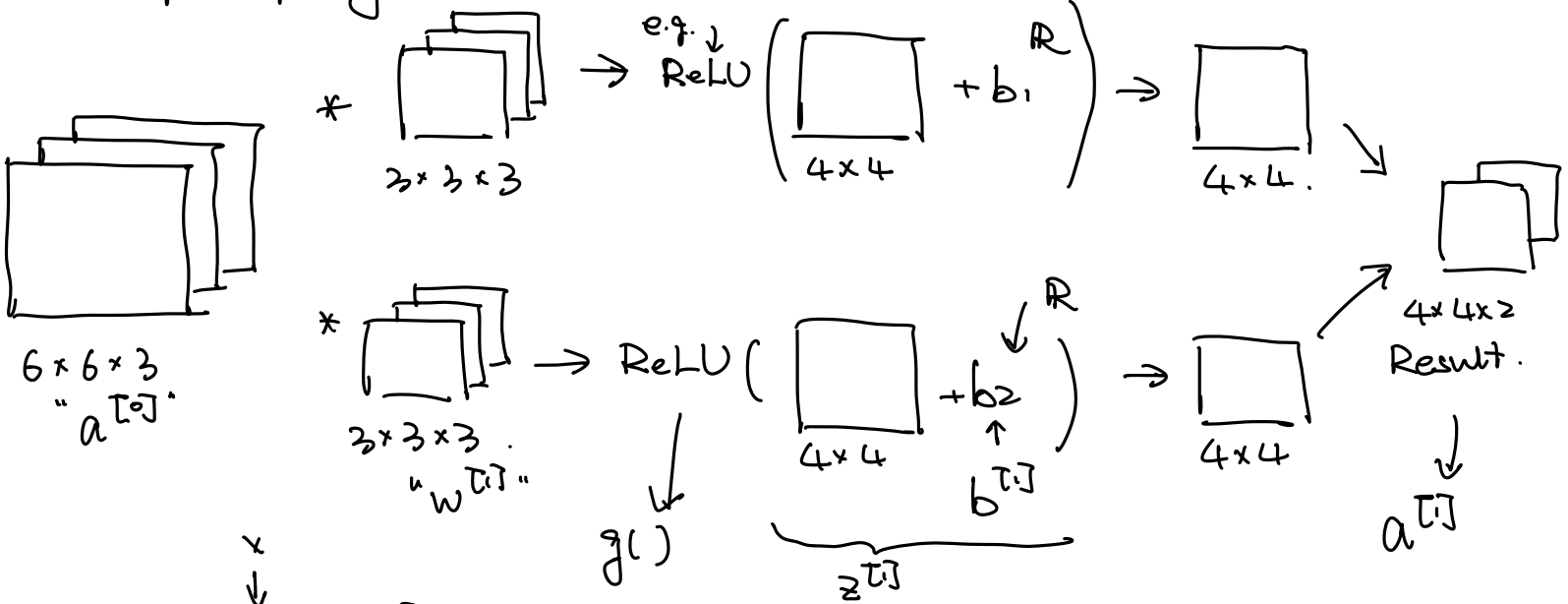
Summary: $n \times n \times n_c * f \times f \times n_c \rightarrow n-f+1 \times n-f+1 \times n_c'$

(n_c: Number of channels; n_c': Number of filters.)

$6 \times 6 \times 3 * 3 \times 3 \times 3$

One layer of a convolutional network.

example of layer.



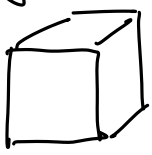
$$z^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$$

($a^{[0]}$: x)

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

Number of params. in one layer.

If you have 10 filters that are 3×3×3 in one layer of a NN.



$3\times3\times3$ 27 params.
+ bias : 28 param

in total 280 params.

No matter how large the input is. you can always use these filters to detect features.

Summary of Notation.

If layer l is a conv. layer.

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

$p^{[l]}$ = padding size.

$s^{[l]}$ = striding size.

$n_c^{[l]}$ = number of filters.

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

Height. Width. # Channel.

Output. $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$.

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

Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

Activation: $a^{[l]} \to n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$ $A^{[l]} \to m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

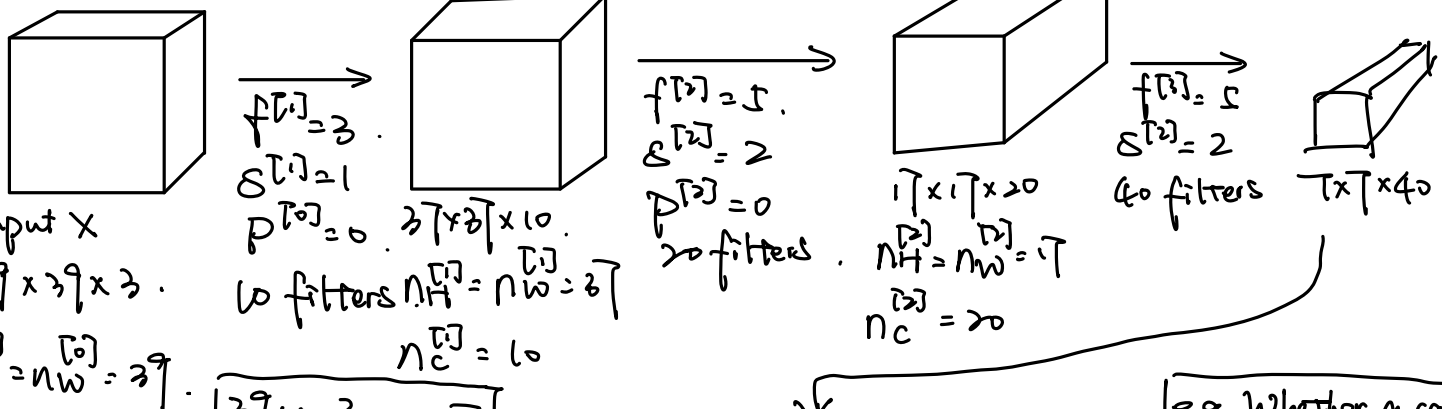
where m is the batch size.

Weights: $\underbrace{f^{[l]} \times f^{[l]} \times n_c^{[l-1]}}_{\text{One filter size.}} \times \underbrace{n_c^{[l]}}_{\text{Number of filters in layer } l.}$

Bias: $n_c^{[l]} \to \underbrace{(1,1,1.n_c^{[l]})}_{\text{later for convenience.}}$

A Simple Conv. NN example.

Example. ConvNet.



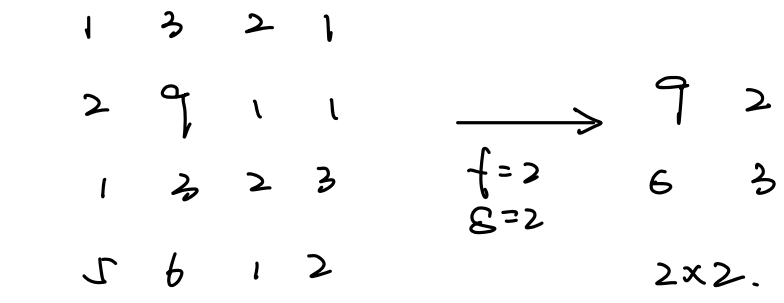
logistic
Softmax

Types of layer in a Conv. net.

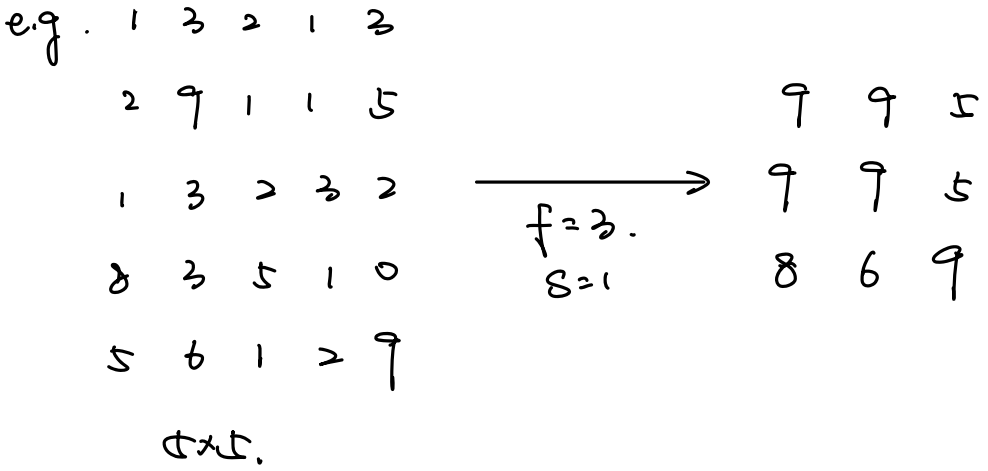
- Convolution. (Conv.).
- Pooling. (POOL).
- Fully connected. (FC).

Pooling layers

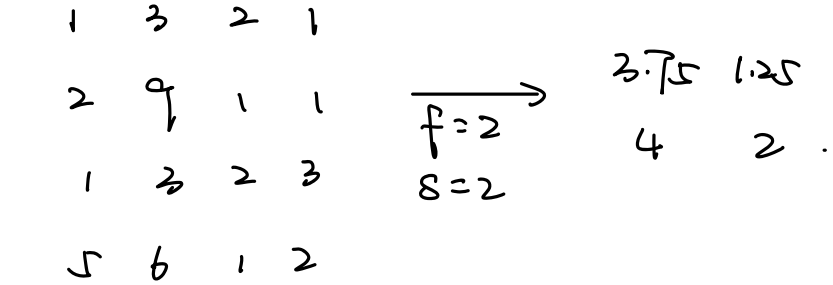
Max pooling.



Notice: No params to learn. A fixed layer.



Average Pooling.



Max pooling use much more often than Average pooling.

Summary of Pooling:

Hyperparams:

f: filter size.

s: Stride size.

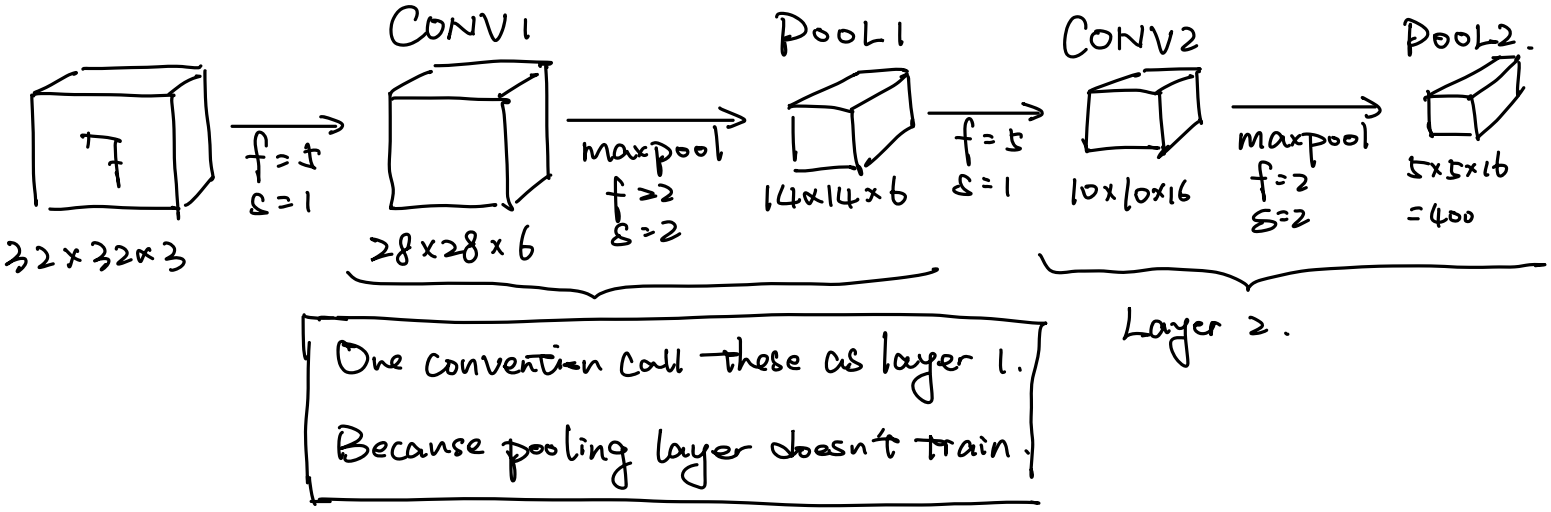
Max or average pooling.

Common value: $f=2, s=2$; $f=3, s=2$.

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

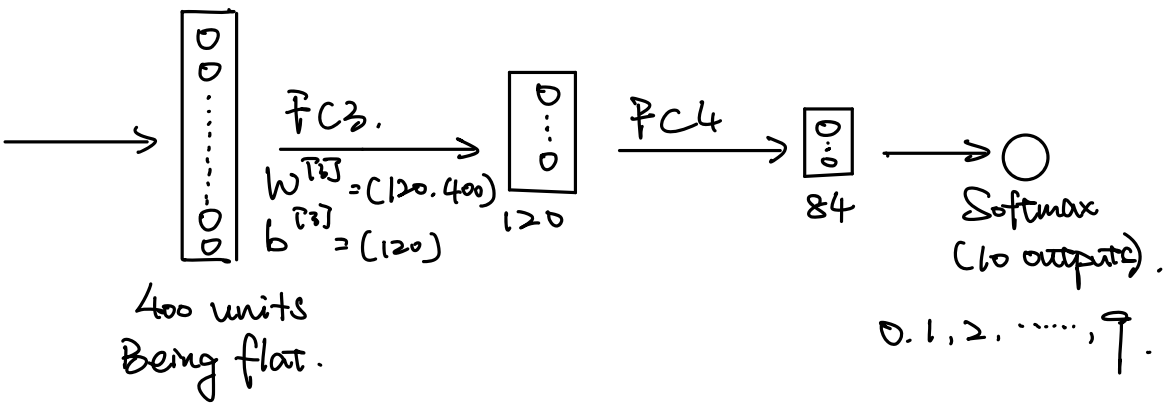
CNN example. (LeNet-5).

Digit recognition.



One convention call these as layer 1.

Because pooling layer doesn't train.



Common pattern. n_H, n_W decrease, n_c increase.

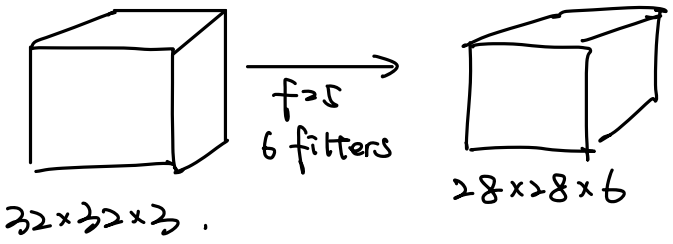
CONV - POOL - CONV - POOL - FC - FC - Softmax.

	Activation Shape	Activation Size	# Params
Input	(32, 32, 3)	3072	0
CONV1	(28, 28, 8)	6272	208

Pool1	(14.14.8)	1568	0
CONV2	(10.10.16)	1600	416.
Pool2	(5,5.16)	400	0
FC3	(120.1)	120	48001.
FC4.	(84.1)	84	10081.
Softmax.	(10.1).	10	841.

Why Convolutions.

Say:



3072 4704

If you create a fully connected network between the two.

You need 14m weights.

But by using conv layer. it has $6\times(5\times5+1)=156$ parameters.

Reason①: Param sharing: A feature detector (such as vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

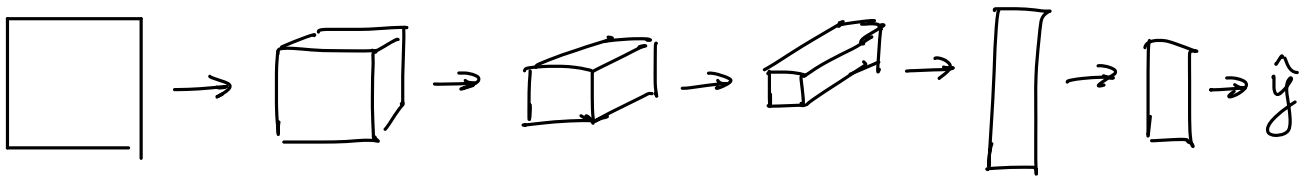
E.G: You don't need to learn multiple edge detectors. one for upper corner and another for bottom corner.

Reason②. Sparsity of connections: In each layer. each output value. depends only on a small number of inputs.

e.g Using 3×3 filter. each output only have relation with 9 inputs

Putting it together.

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



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

Use gradient descent to optimize parameters to reduce J.

Classic Networks.

→ LeNet-5.

→ AlexNet

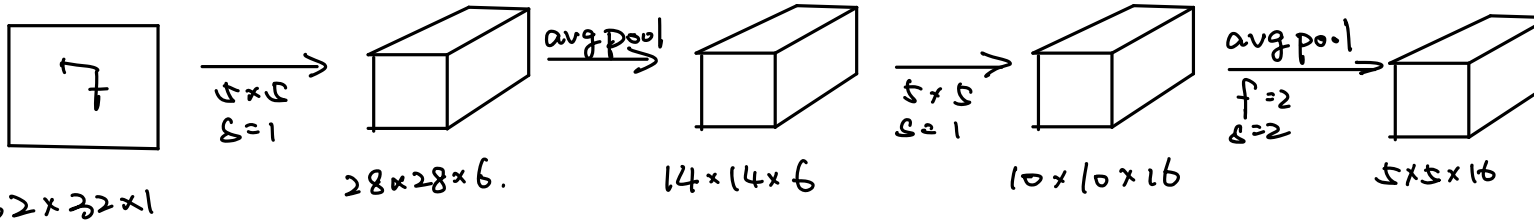
→ VGG.

ResNet. (152). deep

Inception.

Classic Networks.

LeNet-5.



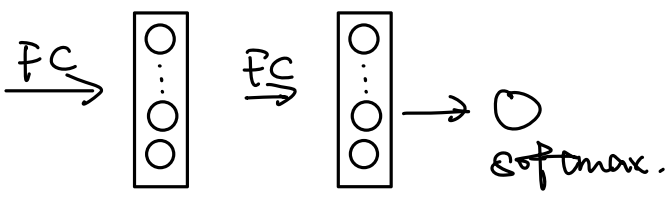
32×32×1 Gray scale

28×28×6.

14×14×6

10×10×16

5×5×16



120

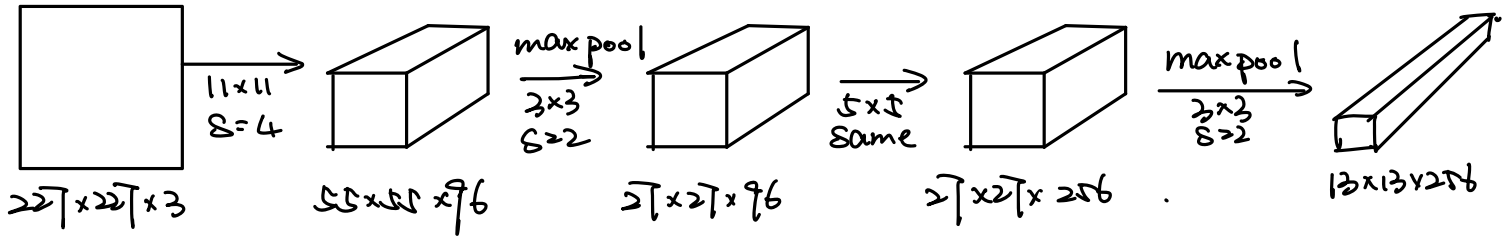
84.

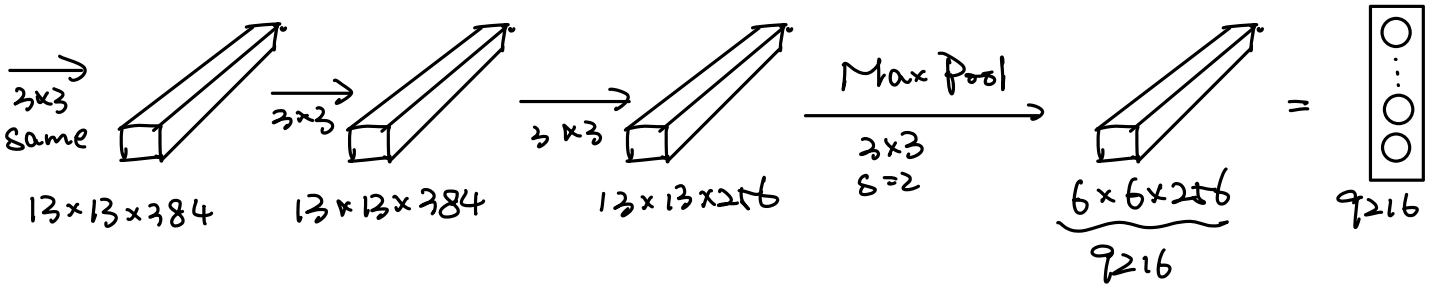
This network is actually small.

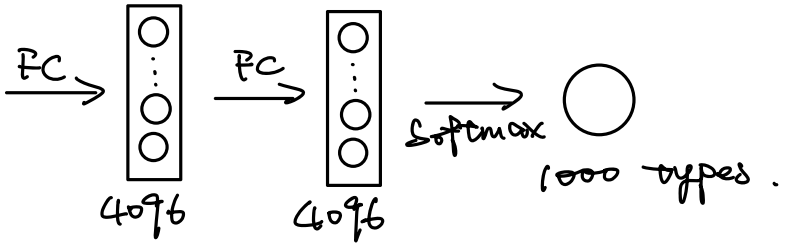
It only takes 60k params.

Trend: $n_H, n_W \downarrow$. $n_C \uparrow$

AlexNet.







→ Similar to LeNet. But much bigger. ~ 60m params.

→ Using ReLU activation function.

→ Multiple GPUs.

→ Local response normalization (LRN). (Doesn't help much.)

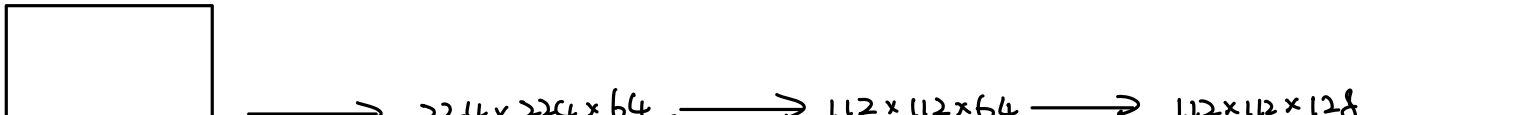
e.g. look one of the block. Say 13×13×256 block. Look at one pixel. and normalize all 256 of them at that point.

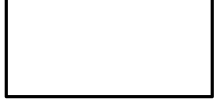
Intuition: For each position. you don't want too large/small activation

VGG-16.

CONV = 3×3 filters. S=1. Same. MAX-POOL = 2×2. S=2.

It simplifies the CNN.





[CONV 64] ×2 → Pool → CONV 128 ×2

224×224×3. We did two Convs here with 64 filters.

→ 56×56×128 —CONV 256 ×3→ 56×56×256 → Pool → 28×28×256 —CONV 512 ×3→ 28×28×512

→ Pool → 14×14×512 —CONV 512 ×3→ 14×14×512 → Pool → 7×7×512 → FC 4096 → FC 4096 → Softmax 1000.

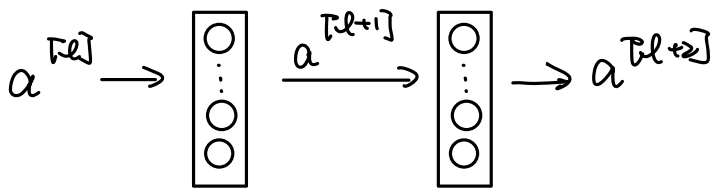
~138 m params. Pretty large.

The "16" in the name mean there are 16 layers in the network that has weights. Other e.g. VGG-19.

Notice. # filters keep doubling when the network going deeper.

Residual Network. (ResNets).

Residual Block.



Steps of computation:

$a^{[l]} \rightarrow$ linear. $\rightarrow$ ReLU $\rightarrow a^{[l+1]} \rightarrow$ linear $\rightarrow$ ReLU $\rightarrow a^{[l+2]}$

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

From $a^{[l]}$ to $a^{[l+2]}$. it will take all the way.

Call this "main path".

What residual block did is.

$a^{[l]} \rightarrow$ linear. $\rightarrow$ ReLU $\rightarrow a^{[l+1]} \rightarrow$ linear $\rightarrow$ (+) $\rightarrow$ ReLU $\rightarrow a^{[l+2]}$

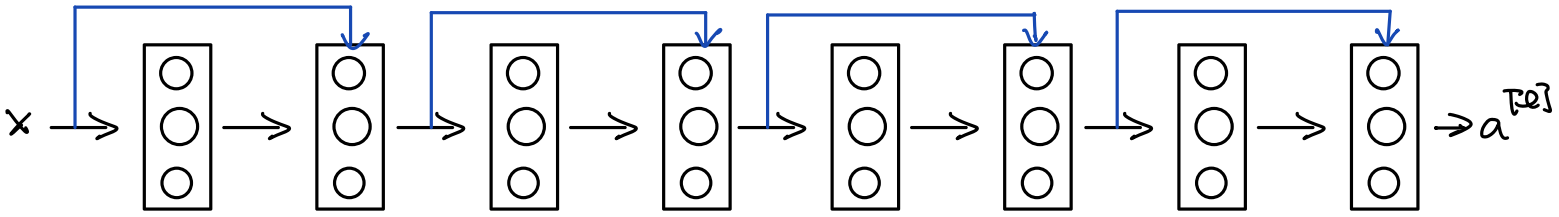
"Short Cut" / Skip connection.

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

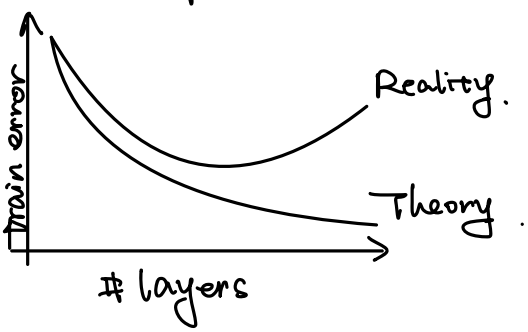
$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$

Residual Network.

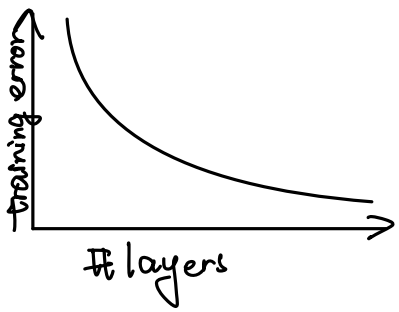
"Plain network"



Plain



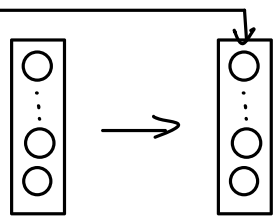
ResNet.



Allow you to train much deeper network

Why ResNet work

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

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

ReLU as activation $a \geq 0$

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

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

↑ if $w^{[l+2]} = 0$, $b^{[l+2]} = 0$. ↓ $g(a^{[l]}) = a^{[l]}$. Since $a^{[l]} \geq 0$.

Easier for $a^{[l+2]} = a^{[l]}$. So adding this doesn't hurt.

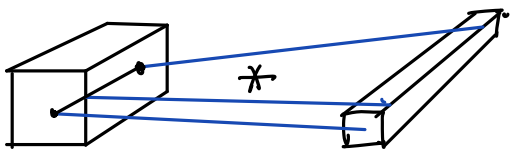
Notice. this $g(z^{[l+2]} + a^{[l]})$ need $z^{[l+2]}$ and $a^{[l]}$ have same dim.

That's why in ResNet we see many conv. layer preserve dim.

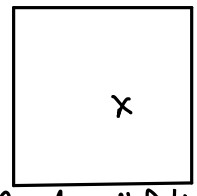
Otherwise. we may write $g(z^{[l+2]} + w_s \cdot a^{[l]})$. where w_s makes sure the dimensions are the same.

Network in Network and 1×1 Conv.

$$\begin{matrix}1&2&3\\3&5&5\\2&1&3\end{matrix} * \begin{matrix}2\\6\\4\end{matrix} = \begin{matrix}2&4&6\\6&10&10\\4&2&6\end{matrix}.$$



$=$ ReLU



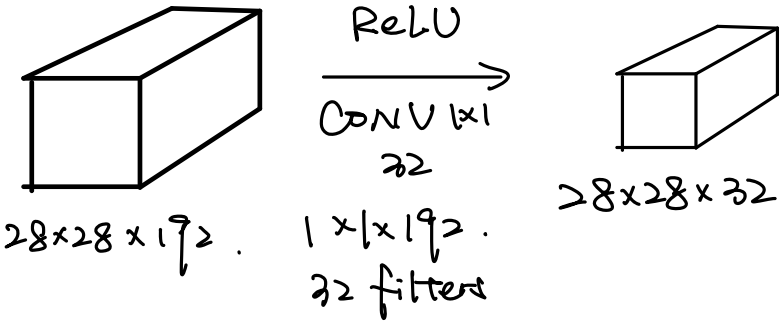
You take one "slice" of numbers, end up with single real number. with ReLU non-linearality.

It's like input 32 numbers with a network. and output to #filters dimension.

slice: | elementwise product sum | → | #filters output.

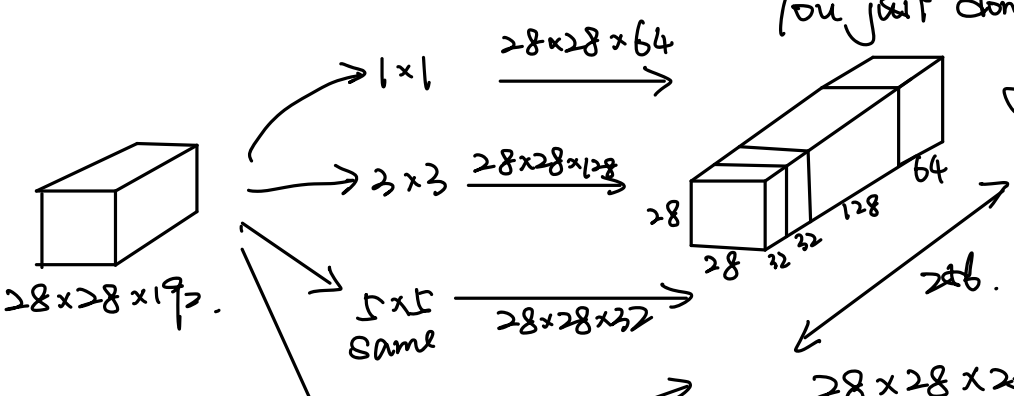
32 numbers — filter

Using 1×1 convolutions.



You can shrink the network by doing the 1×1 conv.

Inception Network Motivation.



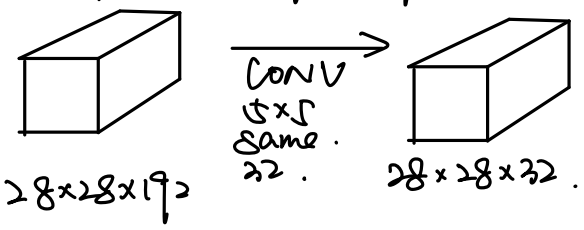
You just don't pick what filter/pooling you want. you just do them all. and stack all of them.

(Only problem: computation)

MaxPool → 28x28x32

Need padding

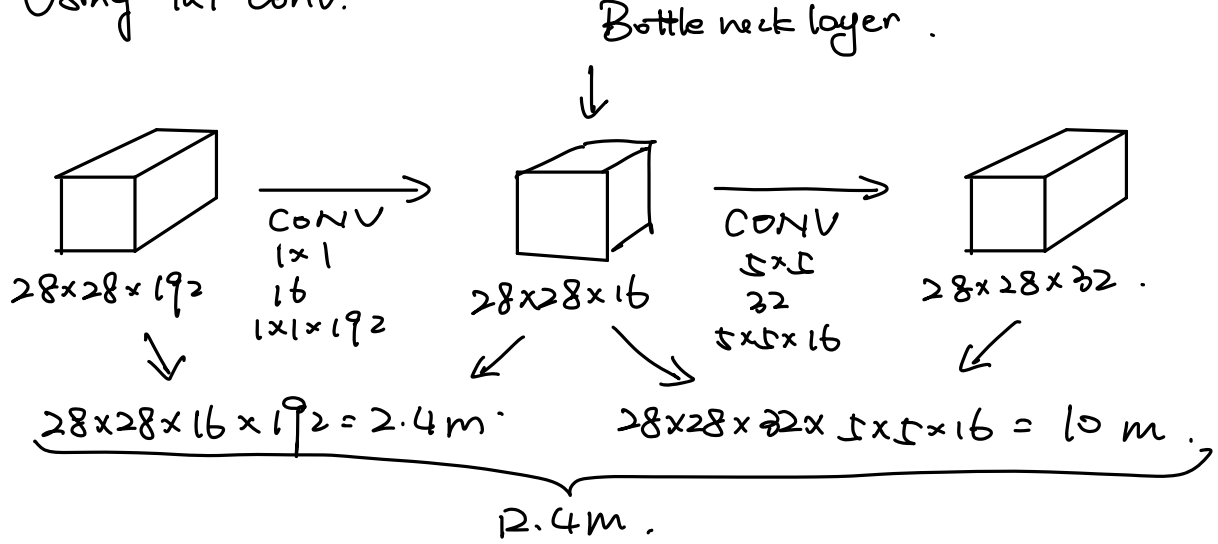
The problem of computation cost.



32 filters. filter are 5x5x192.

28x28x32x5x5x192 = 120M of computations. (Costly!).

Using 1x1 conv.

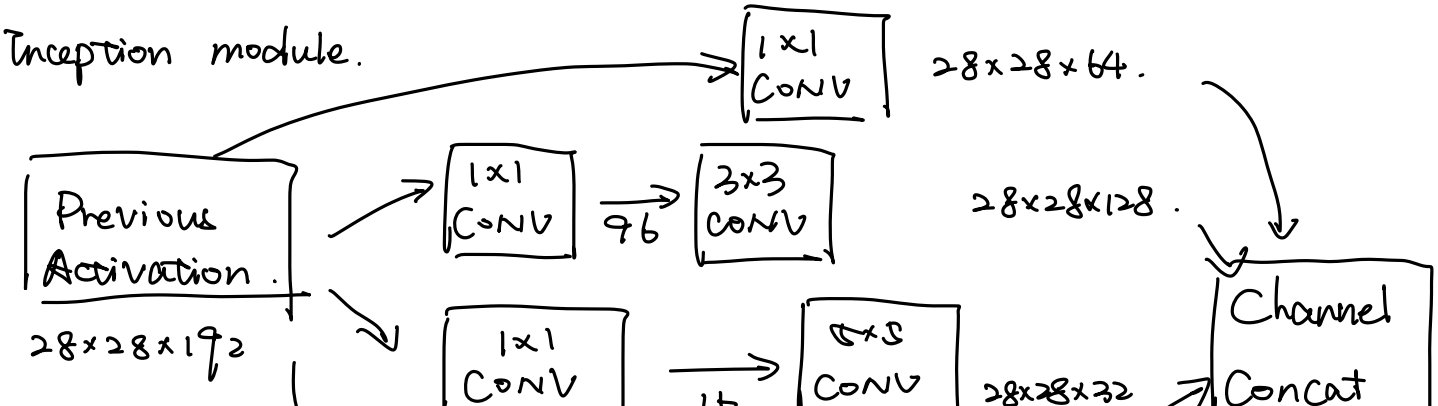


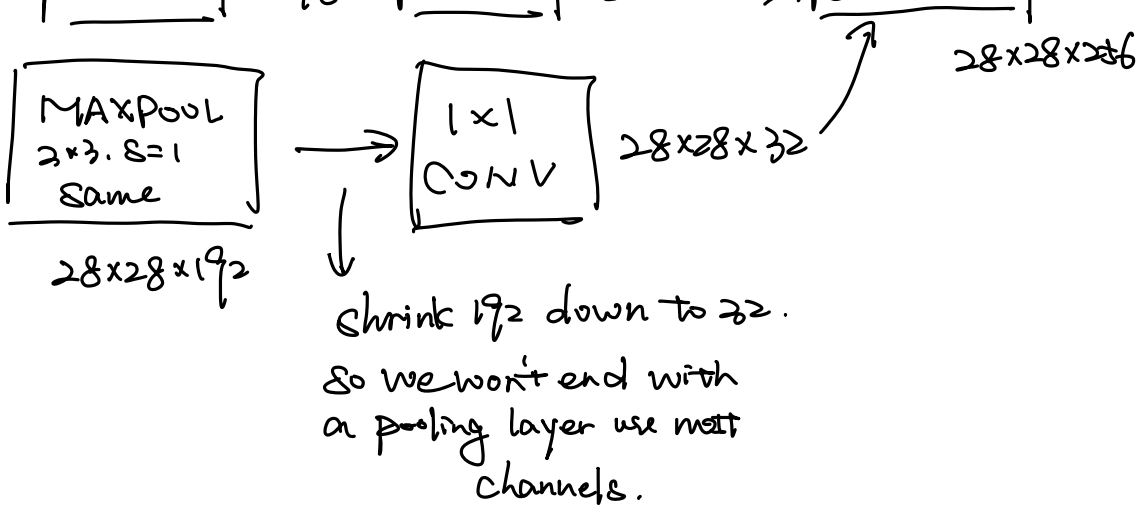
Reduce from 120 m to 12.4 m. Approx. $\frac{1}{10}$.

Question: Will bottleneck hurt performance? Usually no.

Inception Network.

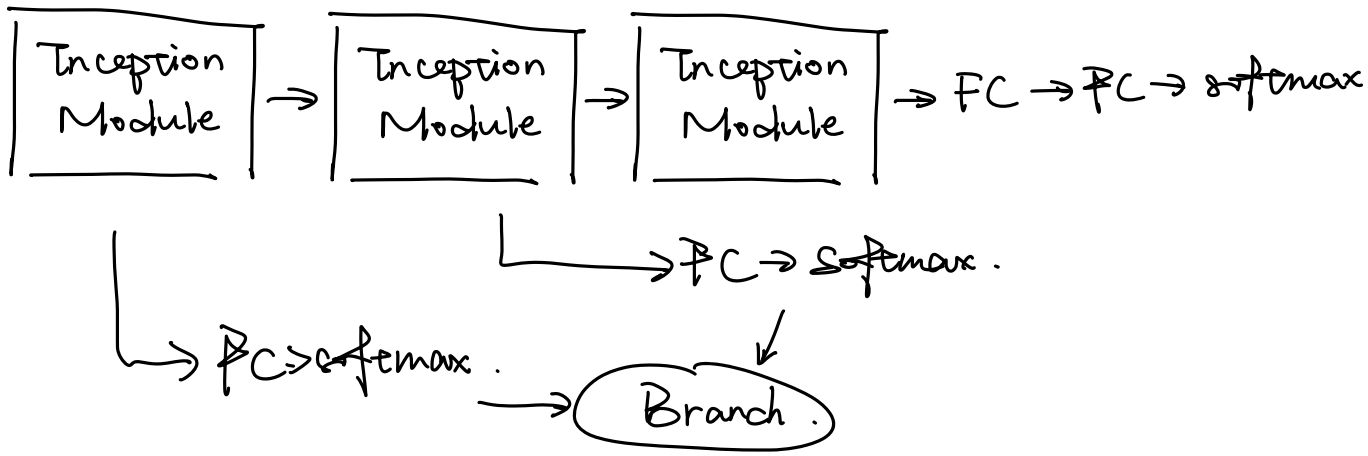
Inception module.





Basically. an inception network is a network with a lot of inception module / block.

Also. there are some hidden layers with branch. connect to FC and softmax to predict result.



Tips for doing well on benchmarks / winning competition.

① Ensembling

- Train several networks independently and average their outputs.
- * But it needs you to run on multiple networks. barely used for production. deployed for actual user. (Also waste memory).

② Multi-crop at test time.

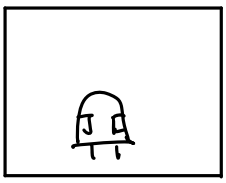
- Run classifier on multiple versions of test images and average results

Use open source code.

- Use architectures of networks published in the literature.
- Use open source code if possible.
- Use pre-trained model and fine-tune on your data.

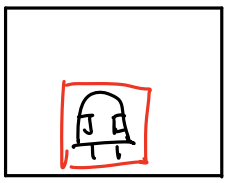
Object localization.

Image Classification.

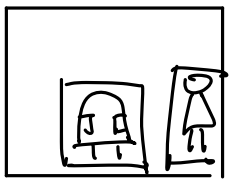


"Car"

Classification with localization



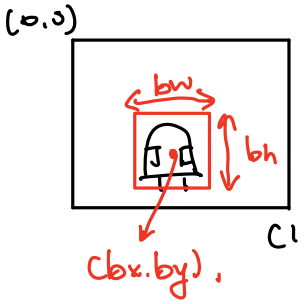
Detection.



1 obj.

Multiple Objs.

Classification with localization.



→ ConvNet. → ○ → Softmax

1. Pedestrian. 2. Car. 3. motorcycle. 4. background.

* Classification part.

↓ New 4 outputs. bx. by. bh. bw. for bounding box.

e.g. bx=0.5. by=0.7

bh=0.3. bw=0.4.

Defining the target 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: is there an object. 0 no object. 1. has object.

b_x, b_y, b_h, b_w: Bounding box.

C_1, C_2, C_3: Which object it is.

If $P_c = 1$. Other value we just don't care.

The loss function.

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

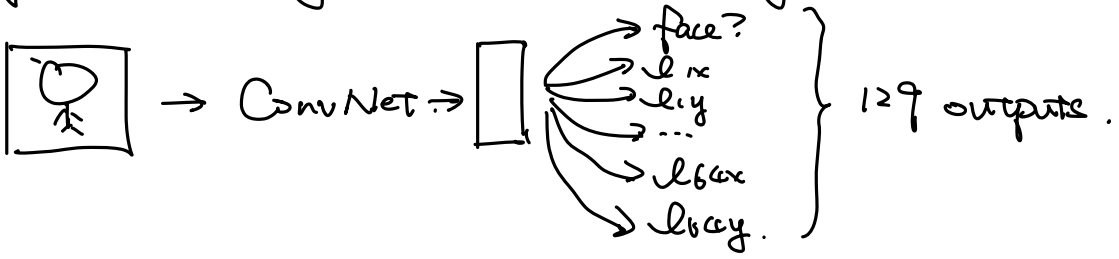
Others don't care.

Of course you may use multiple loss functions for diff. parts.

Landmark detection.

e.g. Given an image of face. output the coord. for the 4 corner of eyes.

e.g. Points help you to land mouth. eyes. ...



Being able to detect these features. e.g. Social media apps can put "hats" "crown" on people head.

Object Detection.

Car detection example.

Training set

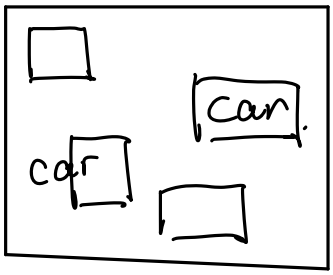
x	y
car	1
⋮	1
	⋮

car → ConvNet → y.

Center croped images.

"The object fill the image".

Then we use Sliding Window detection.



Small square.

We just feed in the little square region into the ConvNet we trained. to classify whether 0 or 1.

Repeat with resized window (maybe larger).

Do this multiple times.

As long as the car is somewhere in the image.

You can show a square with a car on the image.

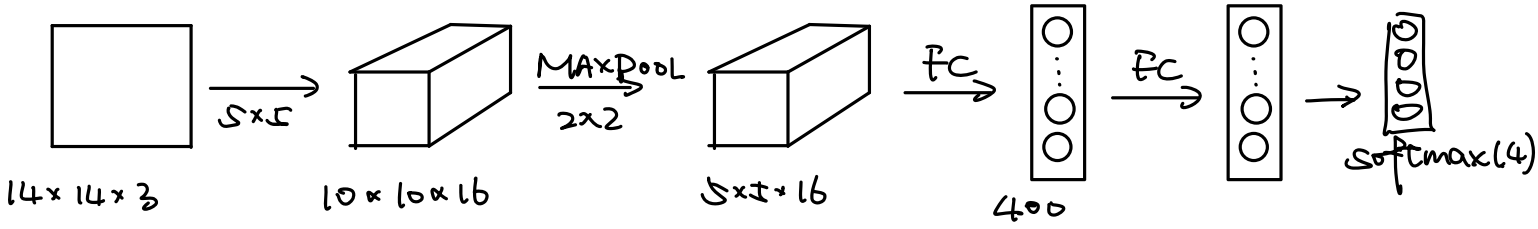
* Computational Cost.

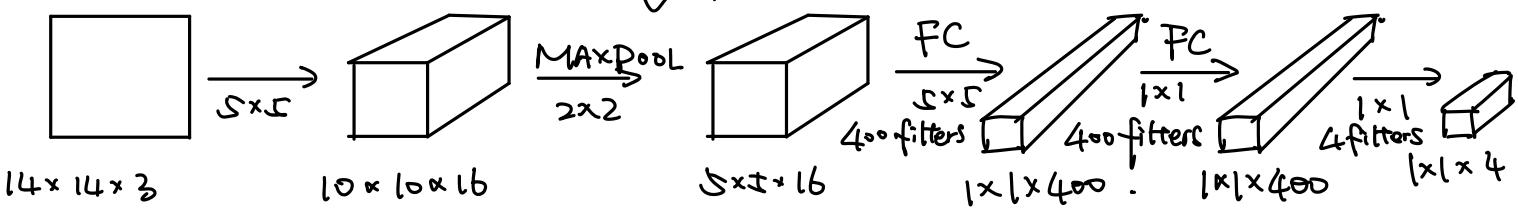
Previously. with simple linear classifier. OK.

Currently. DN. cost a lot. so

Convolutional implementation of sliding window.

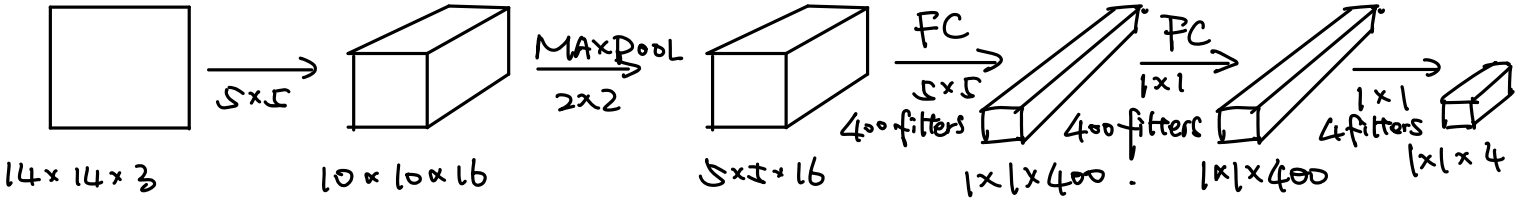
Turning FC layers into convolutional layers.



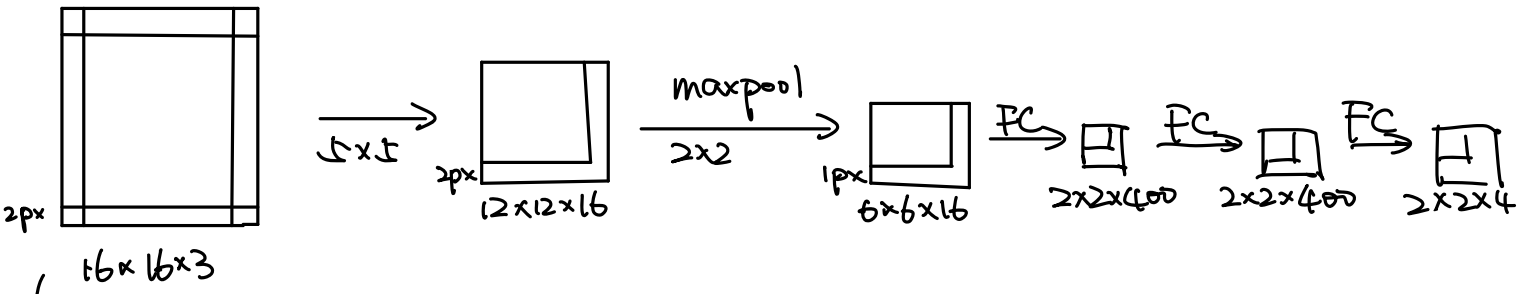


Mathematically. it is the same.

Convolution implementation of sliding window.



Testing image:



it seems like we need instead. we directly put

to do 4 times sliding ⇒ the image into the network.

window here.

Now consider the last output.

The upper left gives you the result of upper left 14×14×3

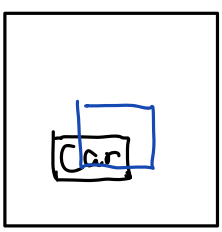
2×2×4. image result. Upper right corner, result of upper right 14×14×3.

The four sliding windows share a lot of computations. We don't need to compute them for multiple times.

Basically. instead of doing the sliding windows sequentially.

we just put the entire image into the CNN. and it will give all the results for all the windows.

Intersection Over Union. IoU.



: Predict box

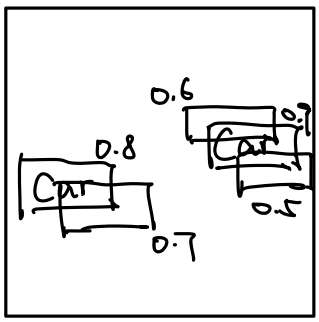
: Ground truth box.

$$\text{Intersection over union} = \frac{\text{size of intersection}}{\text{size of union}}.$$

"correct" if IoU ≥ 0.5. You can use this to judge whether

"May use 0.5. 0.6. 0.7. the localization is correct.

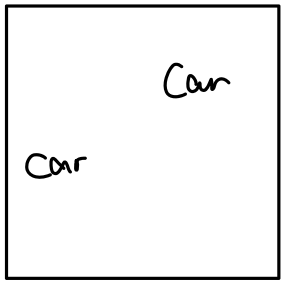
Non-max Surpression.



You want each obj only being detect once in your algorithm. But by setting grid cell. Many cells may say they have large probability to contain the center of the object.

Pick the highest probability box highlight

Probability may comes from IoU.



Each output prediction is: $\begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$

Discard all boxes with $P_c \leq 0.6$.

19x19.

While there are any remaining boxes.

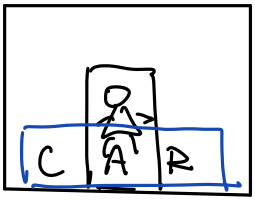
→ Pick the box with the largest P_c output

that as a prediction.

→ Discard any remaining box with IoU ≥ 0.5 with the box output in the previous step.

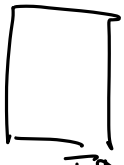
Anchor Boxes.

Overlapping Objects.



$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ C_1 \end{bmatrix}$

Anchor box 1. Anchor box 2



$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ \vdots \\ C_2 \\ C_3 \\ P_c \end{bmatrix}$ Anchor box 1.

c_2
c_3

p_c
b_x
b_y
$\vdots$
c_3

} Anchor box 2.

Previously:

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

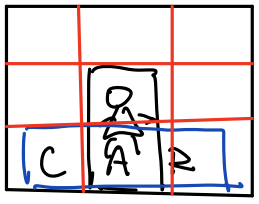
output: $3 \times 3 \times 8$.

With two anchor boxes.

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

output: $3 \times 3 \times 16$ / $3 \times 3 \times 2 \times 8$.

example.



$y =$

p_c	1
b_x	b_x
b_y	b_y
b_h	b_h
b_w	b_w
c_1	1
c_2	0
c_3	0
p_c	1
b_x	b_x
b_y	b_y
b_h	b_h
b_w	b_w
c_1	0
c_2	1
c_3	0

Car only.

0	Anchor box 1
?	
?	
?	
?	
?	
?	
?	
1	Anchor box 2.
b_x	
b_y	
b_h	
b_w	
0	
1	
0.	

Put it together: YOLO Alg.

Region proposal: R-CNN.

Rather than running sliding window on every window. it use some image segmentation algorithm. And only run window on some part of your image.

Face recognition.

Face verification vs. face recog.

Verification:

→ Input image, name, ID.

→ Output whether the input image is that of the claimed person.

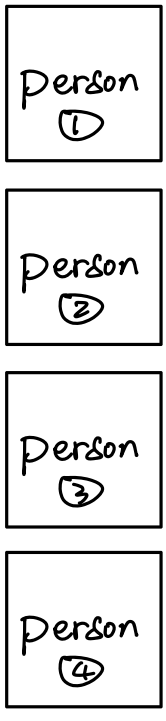
Recognition:

→ 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.



We need to learning from one example to recognize the person again.

Learning a "similarity" function.

d(img1, img2) = degree of diff. between images.

If d(img1, img2) ≤ τ. (some threshold). Predict: same } Verification

for recognition problem.

Just do pairwise comparison.

Pick the highest one larger than the threshold.

Triplet Loss

Learning Objective.

Anchor A | Positive. P | Anchor A | Negative. N.

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

$d(A,P)$ $d(A,N)$.

$\Rightarrow \|f(A)-f(P)\|^2 - \|f(A)-f(N)\|^2 \leq 0$.

But: $f(img)=\vec{0}$ if for all images. it also satisfy the equation.

So we need to modify: → margin. (Like margin in SVM).

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

Loss function.

Given three images A.P.N. (Triplet loss).

$L(A,P,N) = \max(\|f(A)-f(P)\|^2 - \|f(A)-f(N)\|^2 + \alpha, 0)$

① As long it is smaller than 0. we set loss 0.

② As long it greater than 0. loss is pos.

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

Training set: 10k pictures of 1k persons. (Avg. 10 for each).

If only 1 for each maybe its hard to train.

But after training. for new adaled person. may only have one pi

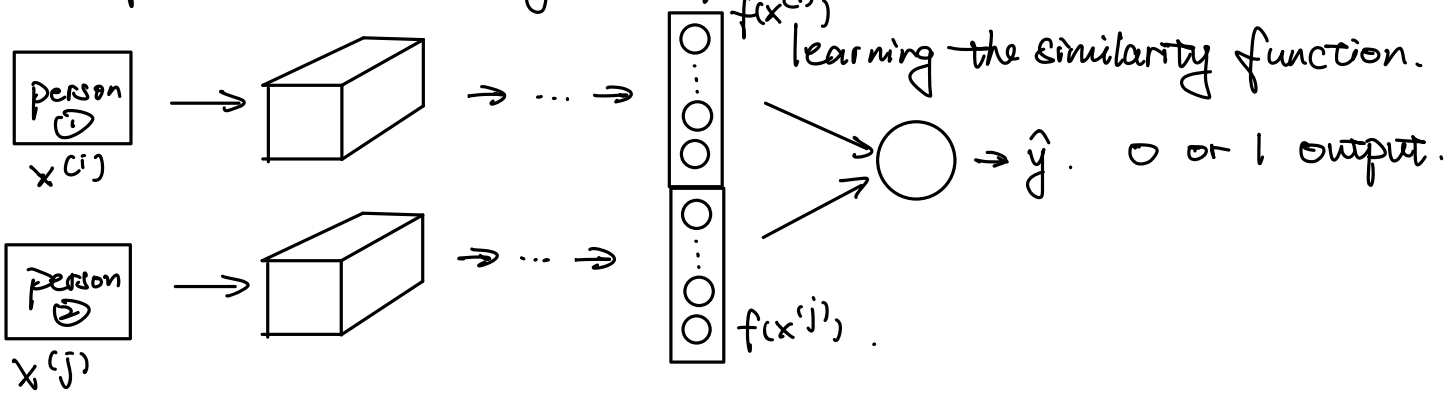
Choosing the triplet A.P.N.

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

So we need to choose triplets that are hard to train on.

e.g. $d(A,P) \approx d(A,N)$.

Face verification and binary classification.



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

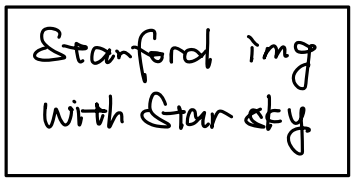
Take the elementwise diff between two encodings.

Or. χ^2 diff: $\dfrac{(f(x^{(i)})_k - f(x^{(j)})_k)^2}{f(x^{(i)})_k + f(x^{(j)})_k}$.

Neural Style Transfer.

Stanford University img	Star Sky

Content Style.



Generated image. G.

Cost function.

Neural style transfer cost func.

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

where C is the content img. S is the style img. G is the generated img.

Find the generated img G.

① Initiate G randomly. (White noise img).

$G = 100 \times 100 \times 3$.

② Use GD to minimize $J(G)$.

$G := G - \frac{\partial}{\partial G} J(G)$ More and more like both imgs.

Content cost func.

Overall 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).

→ Let $a^{[l](C)}$ and $a^{[l](G)}$ be the activation of layer l on the imgs.

→ If $a^{[l](C)}$ and $a^{[l](G)}$ are similar, both images have similar content.

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

Style Cost Func.

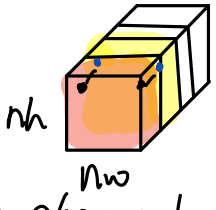
Meaning of "style" of an image.

Saying you use layer l to measure "style".

Define style as correlation between activations across channels.

"How correlated are the activations across different channels?".

e.g. Suppose we have 3 channels.



Blue on yellow channel

Black on red channel.

We take each pair dots on different channel.

Look across all the pairs on n_h and n_w. look how correlated they are.

The correlation gives you one way to measure how often diff features occur or doesn't occur together.

Style Matrix:

let $a^{[l]}_{i,j,k}$ = activation at (i,j,k). $G^{[l](S)}$ is $n_c^{[l]} \times n_c^{[l]}$.

high weight channel

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

Two different activation on different channels but in same pos.

e.g. if we have 3 channels. it will be a 3×3 symmetric matrix. (Gram matrix).

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

$$J^{[l]}_{style}(S,G) = \| G^{[l](S)} - G^{[l](G)} \|_F^2 = \sum_k \sum_{k'} \left(G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'} \right)^2.$$

Also can add norm term $\frac{1}{(2 n_H^{[l]} n_w^{[l]} n_c^{[l]})^2}$

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

$\lambda^{[2]}$ to allow you to pick which layer you want to use. (and how to use it.

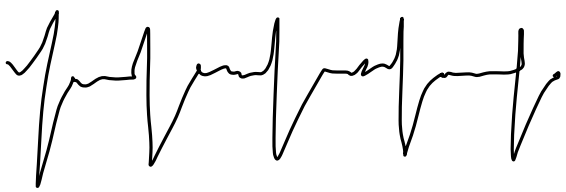
To sum up.

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

1D and 3D generalizations of model.

Now we learned how to process 2D images. (May have multi channels)

e.g. 1D signal.



$14 \times 1 * 5 \times 1 \Rightarrow 10 \times 1$

1 2 3 4 5 6 7 8 9 10 11 12 13 14

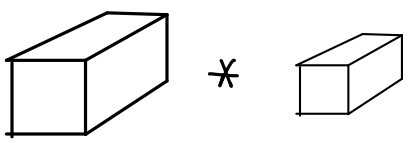
e.g 14 dim

1, 3, 10, 3, 1

5 dim.

Also if you have multiple channel.

3D data. like CT of brain. (Multiple slices).



3D volume $14 \times 14 \times 14$

3D filter. $5 \times 5 \times 5$.

$(14 \times 14 \times 14) * (5 \times 5 \times 5) \Rightarrow 10 \times 10 \times 10$.

This is one channel. May multi channel.

like. if we have 16 filter. : $10 \times 10 \times 10 \times 16$.

$* 5 \times 5 \times 5 \times 16$. 32 filters.

$\Rightarrow 6 \times 6 \times 6 \times 32$.

e.g. 3D volume: Movie data. Medical data.