Computer Vision

COURSE 4
WEEK 1

Image Classification
Object detection
Neural Style Transfer
} Computer vision problems

filter = kernel

Edge Detection

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

Vertical filter

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

horizontal filter

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

Sobel filter

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

Scharr filter

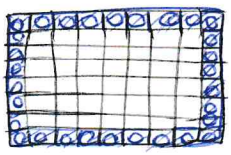
Convolutional neural network frameworks

python: conv_forward
tensorflow: tf.nn.conv2d
keras: conv2D

$n \times n \quad * \quad f \times f \quad = \quad n-f+1 \times n-f+1$

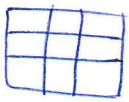
Padding

Problems was: without padding
- shrinking output (dimension)
- throwing away a lot of info from the edges of image



$n \times n$
6×6

* asterisk



3×3
$f \times f$

$= \quad n+2p-f+1 \times n+2p-f+1$

6×6

p = padding

Valid and Same Convolutions:

Valid: no padding $\quad n \times n \ * \ f \times f \longrightarrow n-f+1 \times n-f+1$

Same: pad so that output size is the same as the input size.

$$n+2p-f+1 \times n+2p-f+1$$

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

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

f is usually odd.

3×3 very common $\quad 5 \times 5 \quad 7 \times 7$

Strided Convolutions

$$n \times n \quad * \quad f \times f \longrightarrow \frac{n+2p-f}{s}+1 \quad \times \quad \frac{n+2p-f}{s}+1$$

Technical note on cross-correlation vs convolution

3	4	5
1	0	2
-1	9	7

we've skipped this narrowing operation

7	2	5
9	0	4
-1	1	3

In deep learning literature it is called as a convolutional operation.

Technically this operation is maybe better called cross-validation.

$(A * B) * C = A * (B * C)$ → "associativity" in maths

Convolutions Over Volume

Convolutions on RGB images :

$6 \times 6 \times 3 \quad * \quad 3 \times 3 \times 3 \longrightarrow n-f+1 \times n-f+1 \times n_c'$

$4 \times 4 \times 2$ ← # filters

height, weight, #channels (depth)

IMAGE FILTER

Multiple Filters :

$6 \times 6 \times 3 \quad * \quad$ vertical edge filter $3 \times 3 \times 3 \quad =$ 2D 4×4

$* \quad$ horizontal edge filter $3 \times 3 \times 3 \quad =$ 4×4

$4 \times 4 \times 2$

One Layer Of a Convolutional Network Example

$6 \times 6 \times 3 \quad * \quad 3 \times 3 \times 3 \longrightarrow \text{ReLU}((4 \times 4) + b_1) \longrightarrow 4 \times 4$

$a^{[0]} \quad * \quad 3 \times 3 \times 3 \longrightarrow \text{ReLU}((4 \times 4) + b_2) \longrightarrow 4 \times 4$

"$W^{[1]}$" "$W^{[1]} a^{[0]}$"

$4 \times 4 \times 2$

filters

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

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

Number of parameters in one layer :

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

$3 \times 3 \times 3 = 27 + 1_{\text{bias}} = 28 \qquad 28 \times 10_{\text{filters}} = 280$ parameters

Summary of Notation

If layer l is a convolution layer:

$f^{[l]}$ = filter size
$p^{[l]}$ = padding
$s^{[l]}$ = stride
$n_c^{[l]}$ = number of filters

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

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

$$n_{H,W}^{[l]} = \frac{n_{H,W}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1$$

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

Activations : $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

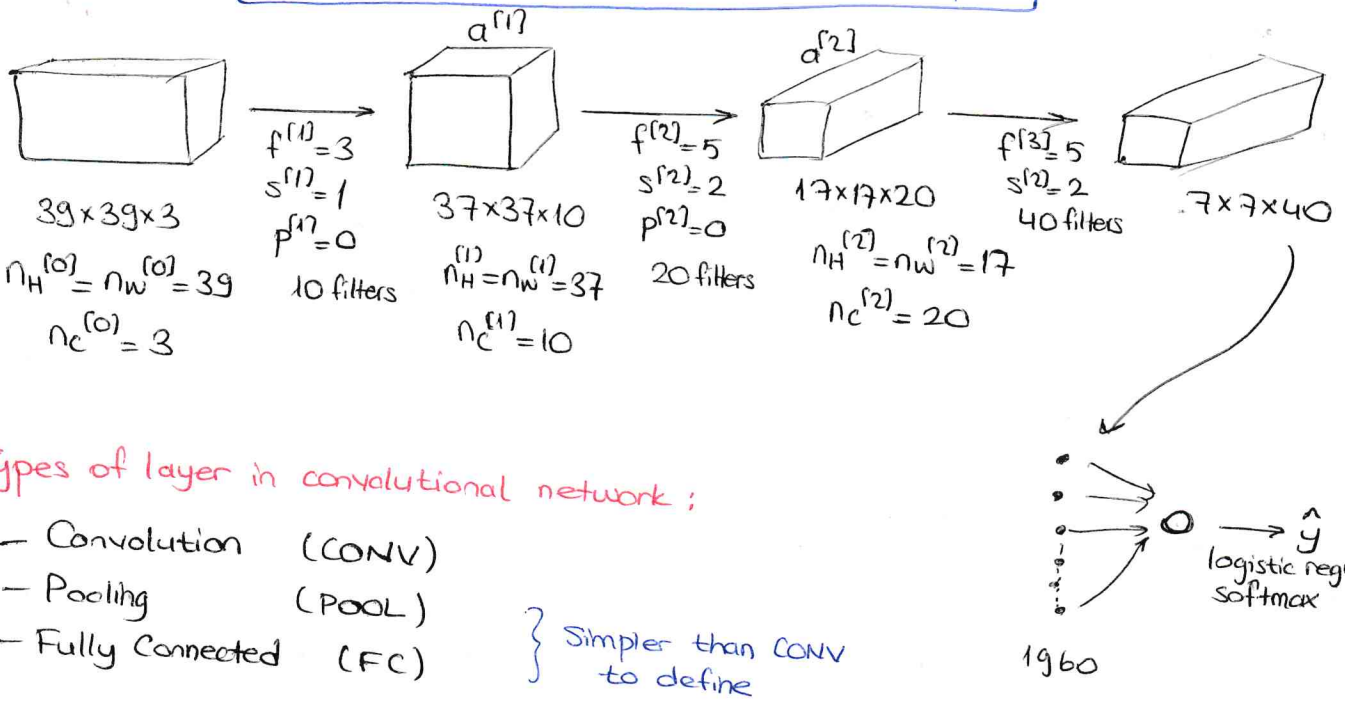
$A^{[l]} = m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

Weights : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$ ← # filters in layer l

bias : $n_c^{[l]} \rightarrow (1, 1, 1, n_c^{[l]})$

$n_c \times n_H \times n_W$

Simple Convolutional Network Example



Types of layer in convolutional network :

- Convolution (CONV)
- Pooling (POOL)
- Fully Connected (FC)

} Simpler than CONV to define

Pooling Layers

Pooling layer : Max pooling

$5 \times 5 \times 2 \longrightarrow 3 \times 3 \times 2$ $\left(\frac{n+2p-f}{s} + 1\right)$

↓ #channel $(n \times n \times n_c)$

↓ #channel $(3 \times 3 \times n_c)$

One interesting property of max pooling is that it has a set of parameters to learn. There is actually nothing for gradient descent to learn.

Average pooling

In very deep ~~teamba~~ neural network, you might use average pooling to collapse your representation.

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

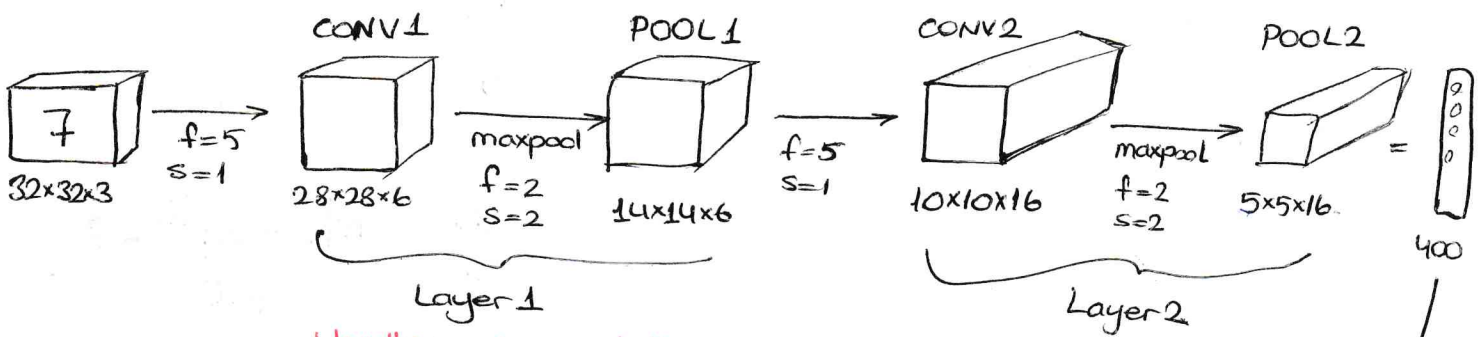
※ max pooling is more common than average pooling.

Hyperparameters :

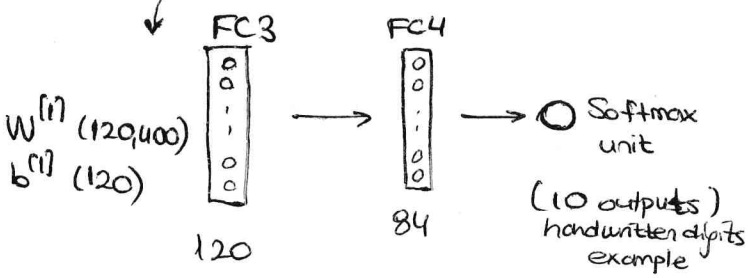
f : filter size — $f=2, s=2$ / $f=3, s=2$
s : stride
Max or average pooling
p : padding (rarely)

CNN Example

(LeNet-5) → Inspired by this net.



Usually people record the number of layers that have weight, that have parameters.
The pooling layer has no parameters (weights)
CONV1 and POOL1 come together (are grouped) as layer 1



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

Very common pattern in neural network

If activation size drops too quickly, that is usually not great for performance

Why Convolutions?

! Number of parameters in the conv layer remains quite small.

Reasons ~~of~~ how small ~~amount~~ number of parameters run well:

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

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

Why look at case studies?

Good way to get intuition on how to build conv nets is to read or to see other examples of effective conv nets.

Outline:

Classic networks:

- LeNet-5
- AlexNet
- VGG -16

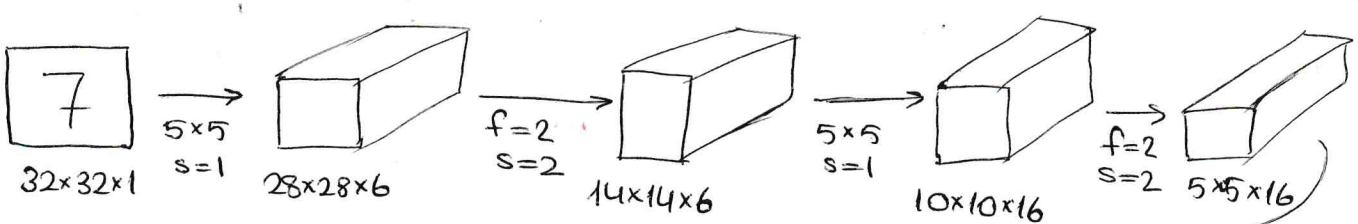
Resnet (152)

Inception

} these ideas are important to get some intuition and very interesting and very helpful for your work.

Classic Networks

LeNet-5



60K parameters:

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

Conv (1 or more) pool conv (1 or more) pool fc fc output

Advanced comments:

In the paper people used sigmoid and tanh nonlinearities.
not ReLU

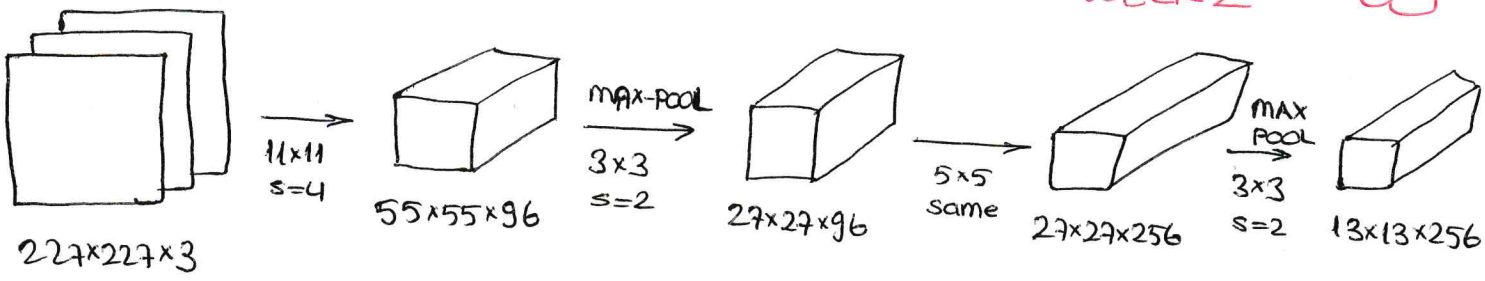
Computers were slow

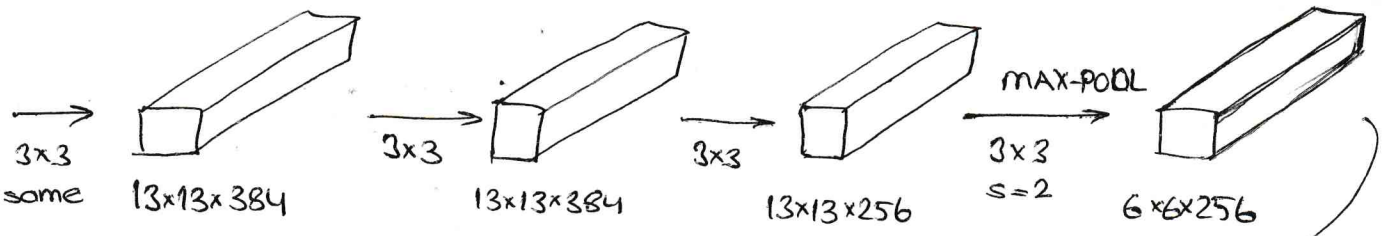
non-linearity after pooling layer

these ideas are from section 2 and 3 of the paper.
- section 2: about architecture
- section 3: has bunch of experiments and results

COURSE 4
WEEK 2

AlexNet :

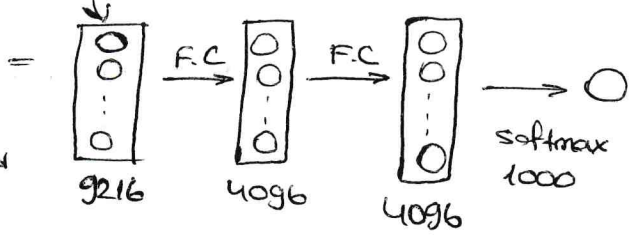




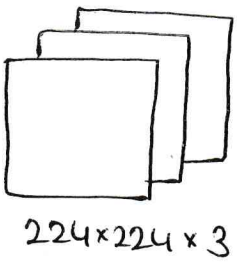
- similarity to LeNet but much bigger

  ~ 60M parameters

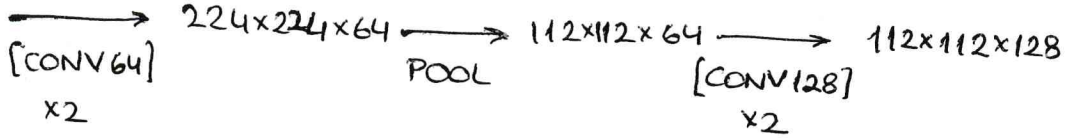
- ReLU
- GPU's were a little bit slower, had a compliated way of training on 2 GPU's (Advanced)
- Local Response normalization (this type of layer isn't used much) (Advanced)

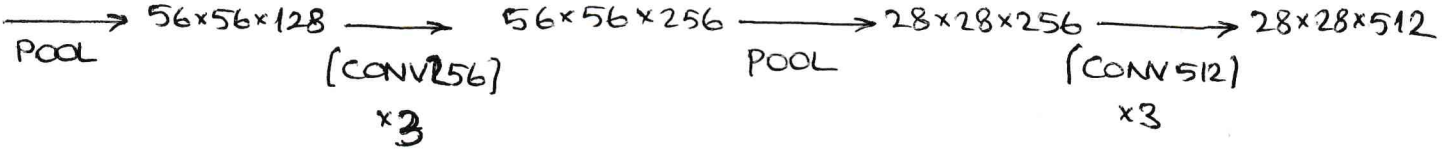


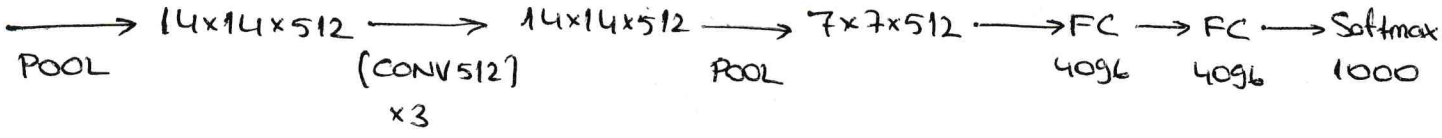
VGG-16 :



CONV = 3x3 filter, s=1, same MAX-POOL = 2x2, s=2







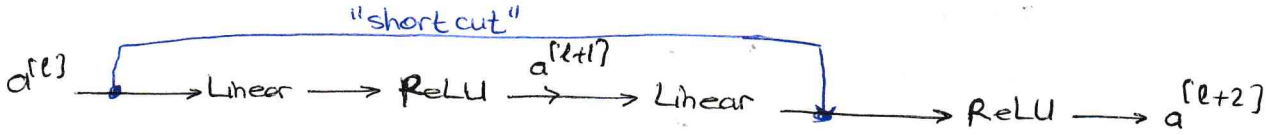
~ 138M parameters

totally 16 layer.

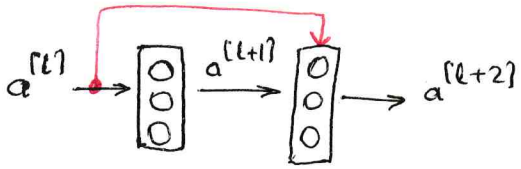
ResNets

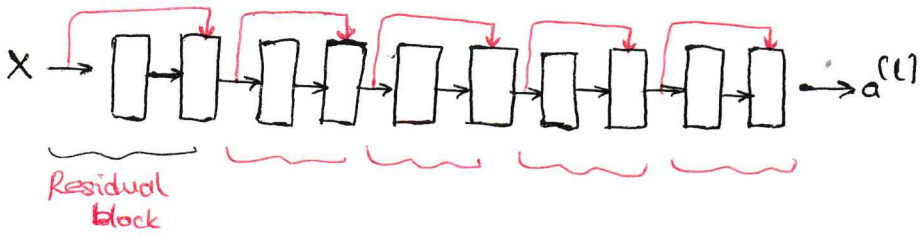
COURSE 4
WEEK 2
3

Residual Block:



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

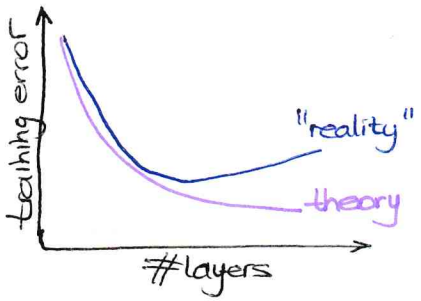




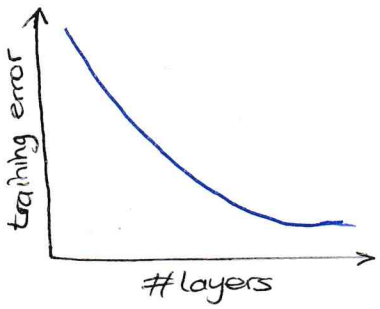
"Plain network"

→ 5 residual blocks

Plain



ResNet



By taking these activations and allowing it to go much deep in neural network, this really helps with the vanishing and exploding gradient problems and allows to train much deeper networks.

Why Resnets Work

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

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

$$= g(a^{[l]}) \quad \text{(relu)}$$
$$= a^{[l]}$$

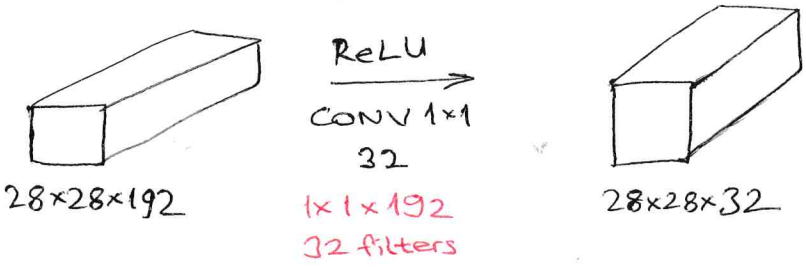
Identity function is easy for residual block to learn! (And it's easy to get $a^{[l+2]} = a^{[l]}$ because of skip connection)

We're assuming that $z^{[l+2]}$ and $a^{[l]}$ have the same dimension.

Networks in Networks and 1×1 Convolutions

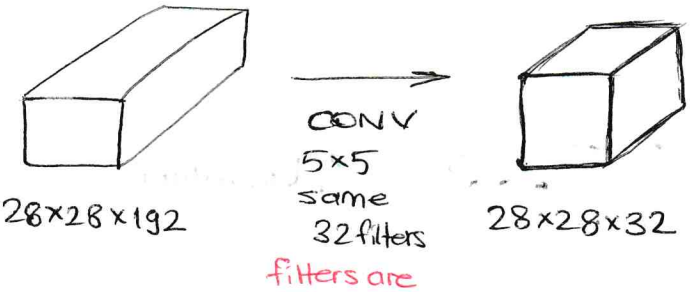
COURSE 4
WEEK 2
4

It is sometimes called Network in Network.



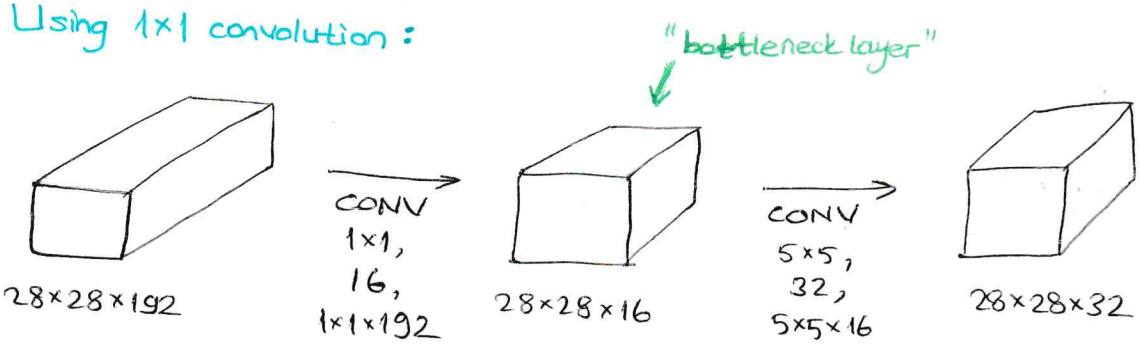
Inception Network Motivation

Computational cost:



$28 \times 28 \times 32 \times 5 \times 5 \times 192 \cong 120\,M$

Using 1×1 convolution:

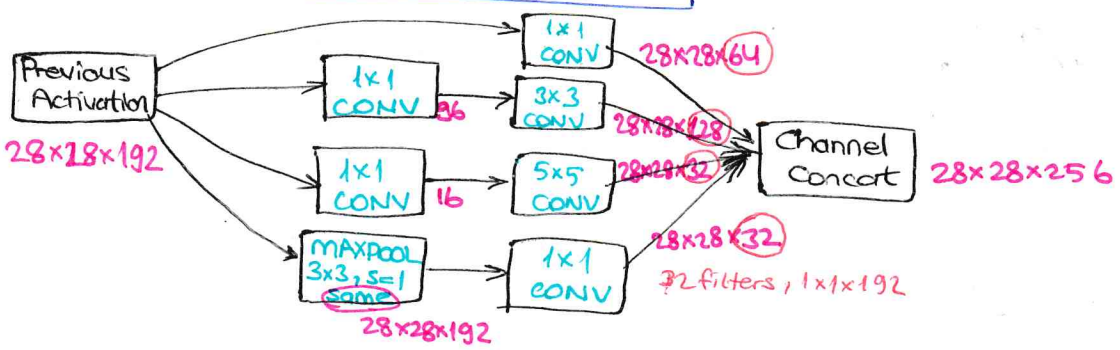


It doesn't seem to hurt the performance but saves you a lot of computation.

$28 \times 28 \times 16 \times 192 \times 1 \times 1 = 2.4\,M$ $\quad 28 \times 28 \times 32 \times 5 \times 5 \times 16 = 10.0\,M$

$\rightarrow 12.4\,M$

Inception Network



⇒ Inception Block

There can be some side branches.
FC and softmax
This appears to have regularizing effect on inception network and help prevent this network from overfitting.

Practical Advices for using ConvNets

COURSE 4
WEEK 2
5

Transfer Learning

→ If you download ways that someone else has already trained on the network architecture and use that as pre-training and transfer that you might be interested in.

Datasets

ImageNet
MS COCO
Pascal Types of Datasets

#Download some open source implementation of a neural network and download not just the code but also the weights

If you have less data, use pretrained network and with those weights just train your softmax layer.
But, if you have more data, the number of layers you freeze could be smaller and then the number of layers you train on top could be greater.

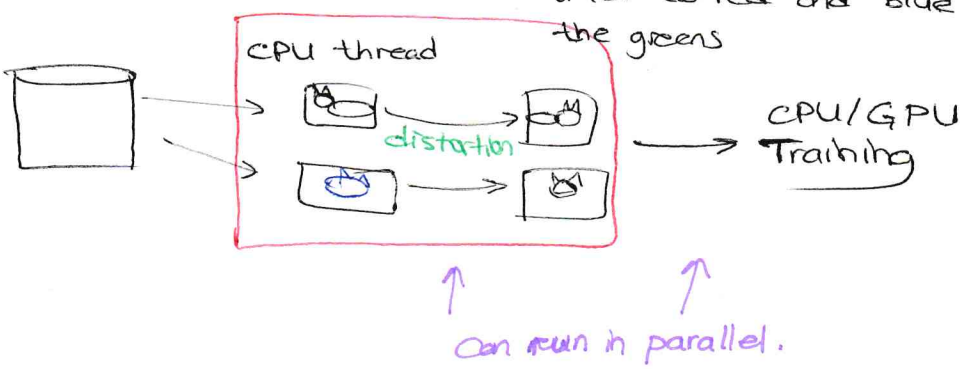
Data Augmentation

Common Augmentation method:

* - Mirroring
* - Random cropping
- Rotation
- Shearing
- Local warping
- Color shifting

Advanced !

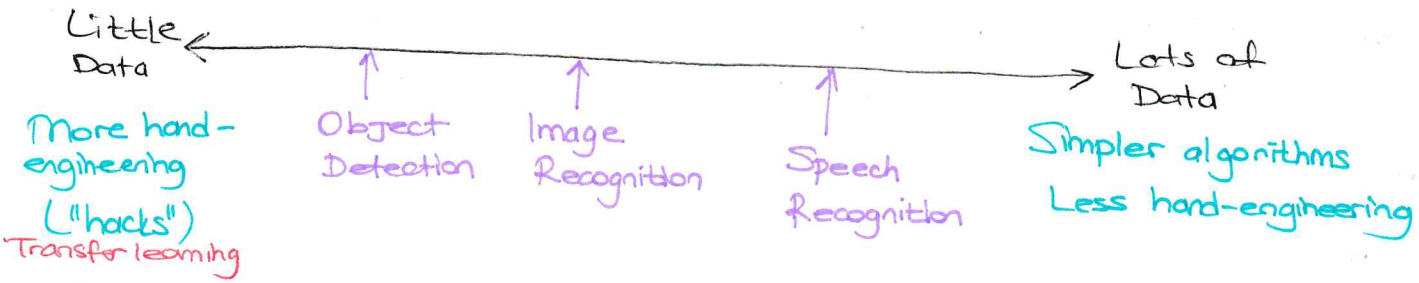
"PCA color augmentation" (in Alexnet paper)
for example, if you have image mainly purple, if it mainly has red and blue tints and very little green, then PCA color augmentation, will add and subtract a lot to red and blue, where it balances all the greens relatively to



State of Computer Vision

COURSE 4
WEEK 2

Data vs. hand engineering



Two sources of knowledge
- Labeled data
- Hand engineered features / network architecture / other components

∝ When you don't have enough data engineering is very difficult, very skillful task that requires a lot of insight.

Tips for doing well on benchmarks / winning competitions :

Ensembling

- Train several networks independently and average their outputs. ($\hat{y}$)

(This will cause maybe 0.1 or 0.2 better performance)

(never use in production) unless you have huge computational budget

Multi-crop at test time (a form of applying data augmentation to your test image)

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

10-crop (if you have the computational budget, you could use in production)

Use open source code

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

FACE RECOGNITION

One Shot Learning

COURSE 4
WEEK 4

When you have few images. (Such as one)

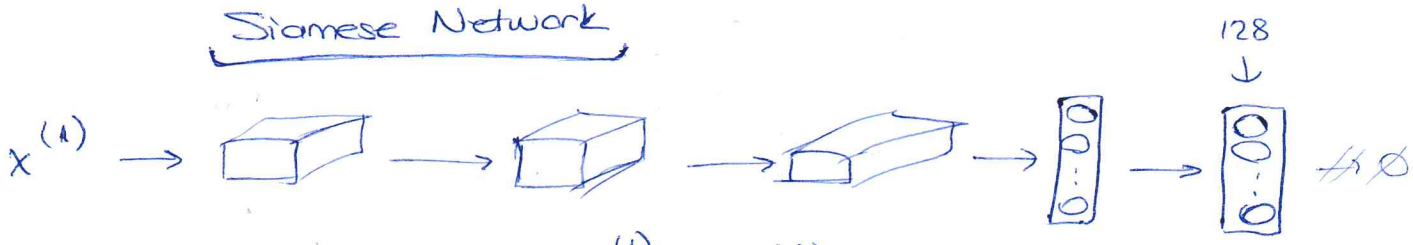
Learning a "similarity" function

d(img1, img2) = degree of difference between images

If $d(img1, img2) \leq \tau$ "same"
$> \tau$ "different"

} This is how you address the face verification problem

Siamese Network



encoding of the $x^{(i)}$ → $f(x^{(i)})$

feature vector

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

Article: DeepFace closing the gap to human level performance (Taigman)

Sometimes this is fed to a softmax unit to make a classification.

Summary:
Parameters of NN define an encoding $f(x^{(i)})$

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

Triplet Loss

Anchor A, Positive P, Negatif N

$$\underbrace{\| f(A) - f(P) \|^2}_{d(A,P)} + \alpha \leq \underbrace{\| f(A) - f(N) \|^2}_{d(A,N)}$$

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

α ↳ margin

Loss function:

Given 3 images A, P, N

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

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

Training set: 10k pictures of 1k persons

Choosing the Triplets A, P, N:

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

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

→ training would be easy and so gradient descent won't do anything because your NN will just get them right

Choose triplets that're hard to train on.

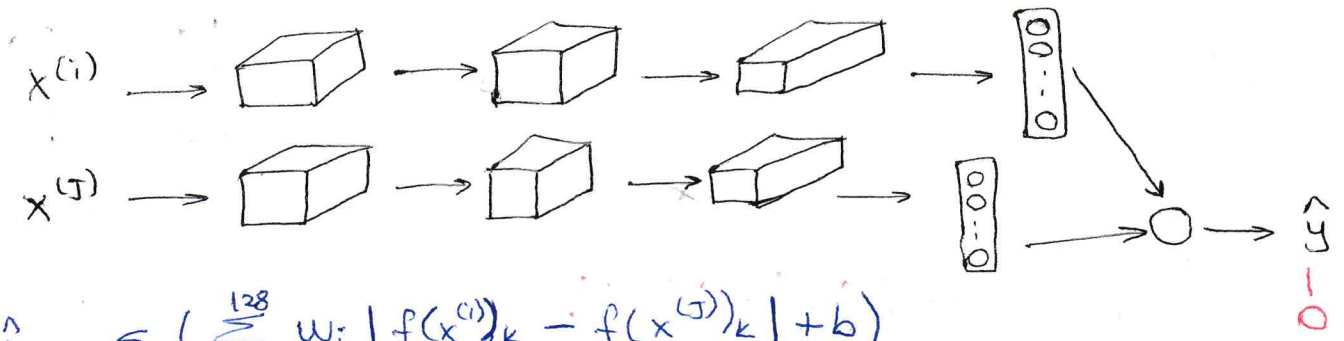
$d(A, P) + \alpha \leq d(A, N)$

Article: FaceNet: A unified embedding for face recognition and clustering

Face Net
Deep Face

} Popular way of naming algorithms in deep learning world.

Face Verification and Binary Classification



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

w_i and b from logistic regression unit.

$$\frac{\left(f(x^{(i)})_k - f(x^{(j)})_k\right)^2}{f(x^{(i)})_k + f(x^{(j)})_k}$$

→ Chi Square

* Images from database are precomputed ones and when new image come it is compared with precomputed feature vectors of database images.

Neural Style Transfer

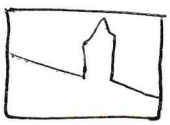
What are deep ConvNets Learning

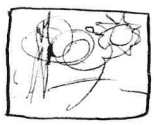
Paper by Mathew Zeiler and Rob Fergus
"Visualizing and Understanding Convolutional Networks"

Pick a unit in layer 1. Find the nine image patches that maximizes the unit's activation.
Repeat for other units.

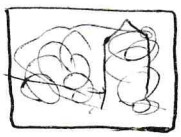
From simple things to complicated things in layers.
Such as edges in layer 1, to textures in layer 2, up to detecting very complex objects in the deeper layers.

Cost Function


Content C


Style S

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



Paper by Gatys et al., 2015. A neural algorithm of artistic style. Images on slide generated by Justin Johnson

Find the Generated Image G

1. Initiate G randomly.
   G: 100 × 100 × 3 ← RGB
2. Use gradient descent to minimize J(G)
   $G := G - \frac{\partial}{\partial G} J$

Content Cost Function

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

- 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 ~~content~~ activation of layer l on the images.
- If $a^{[l](C)}$ and $a^{[l](G)}$ are similar, both images have similar content.

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

→ element-wise sum of squared differences between two activations

(½ not important (just to normalize but can be adjusted by α))

Style Cost Function

Style Matrix

Let $a_{ijk}^{[l]}$ = activation at (i, j, k). $G^{[l]}$ is $n_c^{[l]} \times n_c^{[l]}$

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

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

in linear algebra this is also called "(Gram matrix)", but in this video, it is used "style matrix"

$$J_{style}^{[l]}(S, G) = \frac{1}{(\dots)} \| G^{[l](S)} - G^{[l](G)} \|_F^2$$

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

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

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

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

We would like the "generated" image G to have similar content as the input image C.

Suppose you have chosen some layer's activations to represent the content of an image.

In practise, you'll get the most visually pleasing results if you choose a layer in the middle of network.

$a^{[l](C)}$ simply → $a^{(C)}$ shape: $n_H \times n_W \times n_C$

$$J_{content}(C, G) = \frac{1}{4 \times n_H \times n_W \times n_C} \cdot \sum_{\text{all entries}} (a^{(C)} - a^{(G)})^2$$

? Each color represents the activations of a filter as it was convolved around an image.

Style Matrix:

Gram matrix → $(v_1, \ldots, v_n)$

$G_{ij} = v_i^T v_j = np.dot(v_i, v_j)$

↳ compares how similar v_i is to v_j

default image → $(n_H \times n_W, n_C) \rightarrow G = (n_C, n_H \times n_W) \cdot (n_H \times n_W, n_C)$

result = (n_C, n_C)

G_{ij} measures how similar the activations of filter i are to the activations of filter j.

One important part of the gram matrix is that the diagonal elements such as G_{ii} also measures how active filter i is. For example, suppose filter i is detecting vertical textures in the image. Then G_{ii} measures how common vertical textures are in the image as a whole: If G_{ii} is large, this means that the image has a lot of vertical texture.

STYLE WEIGHTS

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