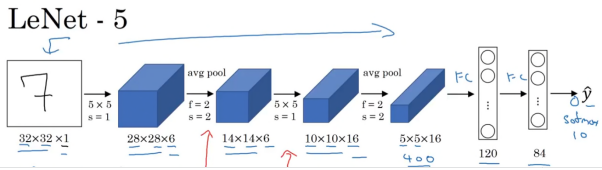
4 Convolutional deep network

Andrew Ng

Note by:
Hossein Mamaghanian
amirhm@gmail.com

Classic Networks.

lenet5 1998



Originally used for digit classification.

Network is build from a cascade of conv_Pool conv Poll_fc_fc → Softmax

the pattern as we go from in → output $n_H, n_W \downarrow$ $n_C \uparrow$

Non linearity is applied after pooling layer (sigmoid / Tanh)

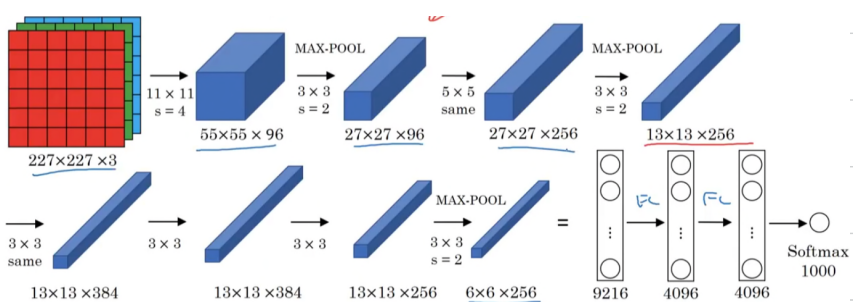
lecun etal 1998. Gradient based learning technique applied to doc. recognition

- original version lenet 5 has 60K parameters.

Alexnet (2012)

Has an architecture as below, and been trained on massive database of imageNet. Biggest advantage was using Relu activation.

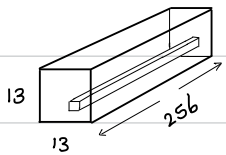
- Similar to leNet, But much bigger (60M Parameters)
- Using Relu
- Multiple Gpus
- local response normalization (LRN)
- lots of hyperparam ✗



Krizhevesky etal. 2019. ImageNet classification with deep con. NN.

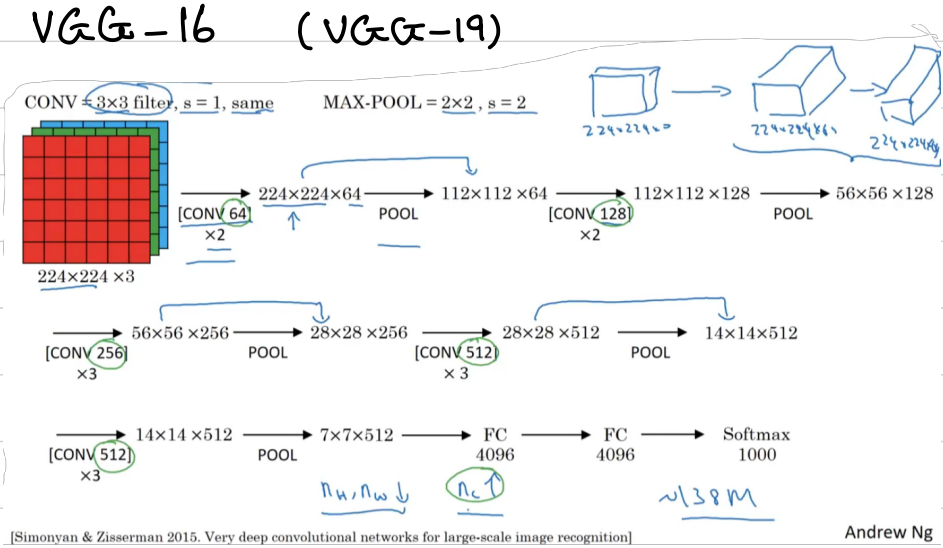
back in the time "local response normalization" is also used. The idea was to normalize across the filters.

(Not very useful, not used anymore)



Normalize.

VGG-16 (VGG-19)



→ 138M parameters!

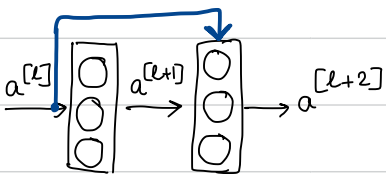
- "16" in the name, refers to number of layers, pretty large.
- simplified architecture
  - convolution layers all 3x3, s=1, 'same'
  - Max-pool 2x2, s=2
- The pattern is as we go deeper in Net $h\downarrow$ $w\downarrow$ $c\uparrow$ (~2 ~2 ~2)
- 138M parameters!!

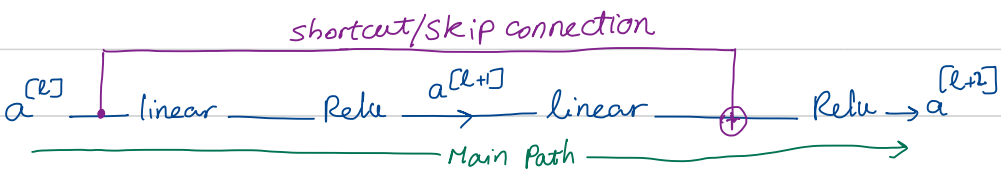
ResNet

Using Residual blocks to solve vanishing / exploding gradient problems
→ getting the activations from a layer and feed it to the deeper layers.

Residual block:

activation from layer [l] is added after linear operation, but before activation function. (Relu, ...)

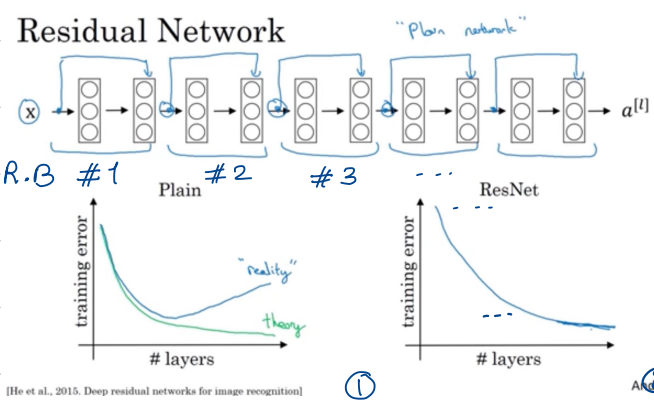




$$z^{[l+1]} = w^{[l+1]} a^{[l]} + b^{[l+1]}, \quad a^{[l+1]} = g(z^{[l+1]})$$

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

- the idea is to add a direct path (shortcut/skip) to the main path.



① ②

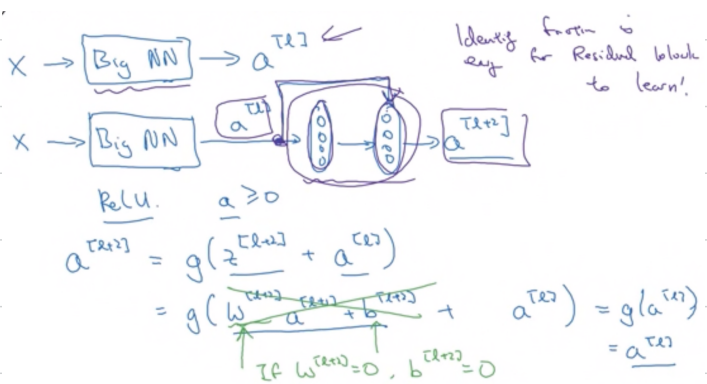
using Residual blocks helps to train deeper networks.

In theory as the network get deeper, training error should decrease, but wouldn't (due to problems in optimization techniques that we use) Fig1. Fig 2. By adding residual blocks, this get better.

Why ResNet works?

The main idea for using residual blocks, is that as layers number increase, we get worst results (due to problem of b.propagating the gradient). by adding residual blocks, we gurantee that results atleast does not get wors (and improves), since minimum, easily we learn to be identity at worst case.

- This helps to add layers and train deeper networks.



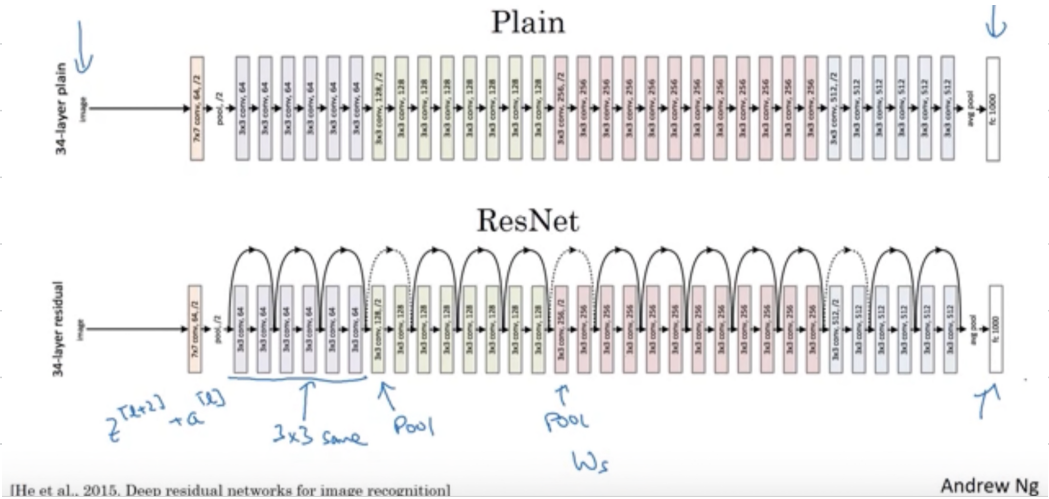
- if activations are not same cardinality, we could use matrix W_s, either we learn or just zero-pad.

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

$$= g(z^{[l+1]} + W_s a^{[l]}) \quad , a^{[l]} \in \mathbb{R}^{n}, a^{[l+1]} \in \mathbb{R}^{n} \Rightarrow W_s \in \mathbb{R}^{n \times m}$$

ResNet

Example in ImageNet network:

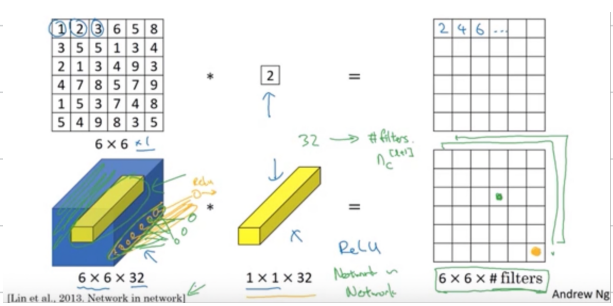


- Usually all conv layers are "Same" to keep cardinality.
- If there are Pooling or Pooling like operators, we use "W_s" to add residuals.

1x1 convolution

The main idea is to shrink #nc

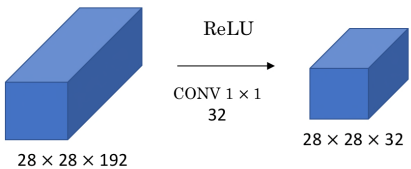
- Its like adding a fully connected layers across channels.



- Sometimes this is called "Network in network"

→ Example of using 1x1 conv on image to reduce #nc with less computation.

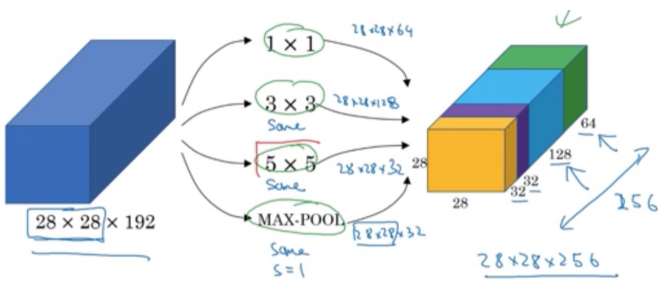
- also capable of learning more complex functions.



Inception modules

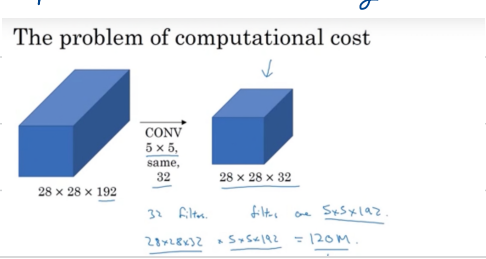
Motivation is that, we use all combination of filters with various hyper parameters and stack all of them, and let network choose what is useful to learn

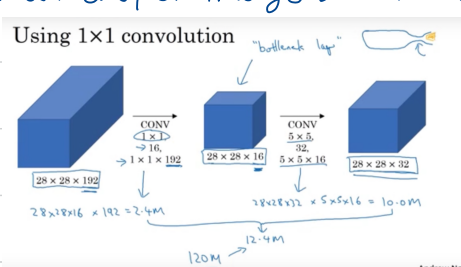
"Szegedy et al, 2014"
going deeper with convolution



- Using 1x1 conv for reducing the computational cost.

→ comparison between doing 5x5 filter directly on image or after 1x1 conv

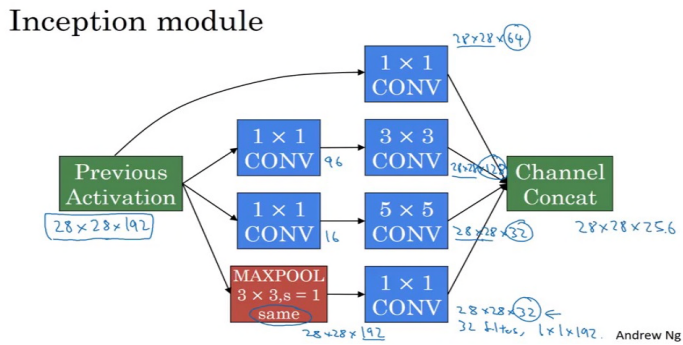




- This does not hurt performance and improves the computation.

Inception Networks

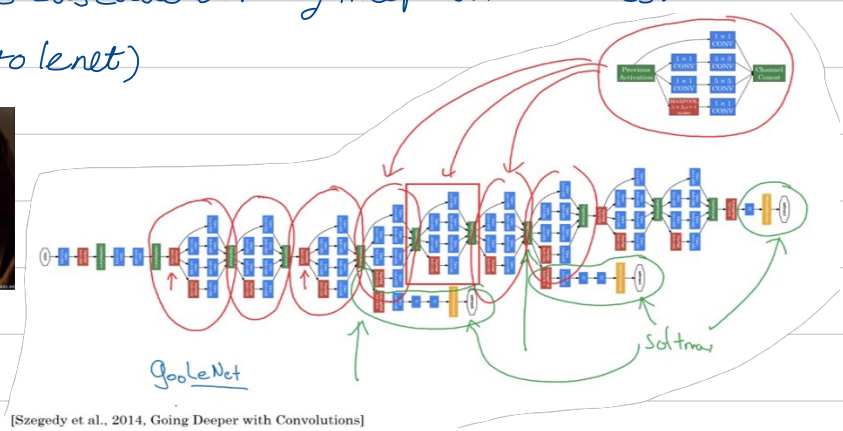
The idea is to build modules with multiple combination of the filters and let the network learn what is the best. (done using 1x1 conv)



Inception network is cascade of many inception modules.

Goolenet (homage to lenet)

↳ Authors has cited this meme in the paper.

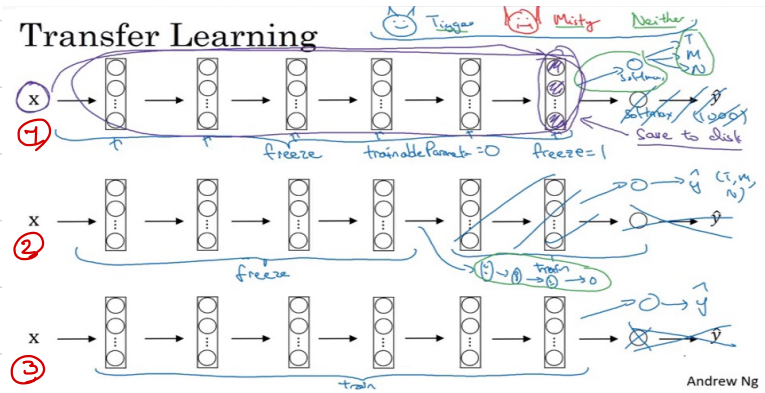


[Szegedy et al., 2014, Going Deeper with Convolutions]

green marks are FC + softmax, which are used to make predictions. This ensures that even the features computed in hidden intermediate layers are not bad. This has regularization effect and prevents over-fitting.

Transfer learning

main idea is that we take the already trained net and use it for other similar tasks.



Generally this could be done in 3 different ways:

1. Take an already trained network as-is. Freeze the layers except the output softmax layer (if number of classes are different change) and only train that layers.
   - → either by setting (trainable_weights=0 or freeze=1)
   - → or just take the output of last layer for all our training database. and use it as features that we want to train our network. this would be better since doesn't need to calculate in each epoch. the same outputs.
2. Take the trained network and freeze the first layers and only train the later layers, or maybe even change the later layers.
3. Take the trained network and start with it, (use it as initial weights) (useful only when we have enough training data and processing power.

Data augmentation.

when we want to increase the database size. CV needs a lot of training images to be trained.

First types:

① . Mirroring (easiest, almost always true.)

② . Random cropping (also frequently used)

③ Rotation

④ shearing 

⑤ local wrapping

Second types:

① . color shifting (R, G, B) ± a (Resembling different lighting)

② other techniques could be used for color augmentation like PCA (used in Alexnet paper)
used to find the color strength and overally keep the color tint.

Tips for doing well on benchmarks/winning competitions.

Ensembling:

- Train several networks independently and average their outputs.

3~15 Network

Multi-crop at test time.

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

"Reversed or flipped version"

10 crop: 

1 + 4 + 1 + 4

Use open source code:

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

Object localization.

→ 3 type of problems: mage classification

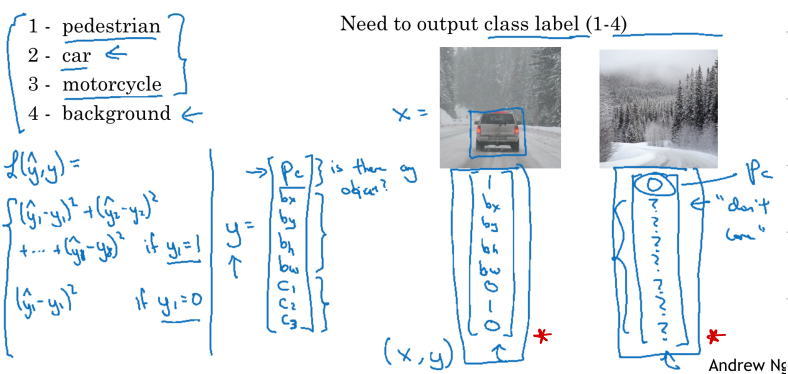
① Image classificatio

② classification & localization:

Not only an image classification is done, also a bounding box also provided.

③. Detection.

Defining the target label y



Outputs should look like these examples: *

P_c: Probability an object exist (Ped, car, ...)

$[b_x, b_y, b_h, b_w]$: bounding box Parameters.

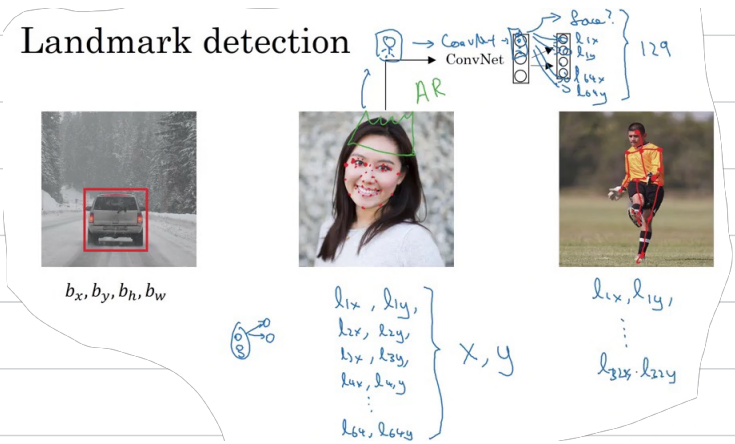
C_i: each class probability.

Main point is the definition of the loss function. which is different for P_c: 0 or 1. when $P_c = 0$, rest of outputs should not be considered (dont care)

landmark Detection

In landmark detection task, instead of just providing the bounding box, we learn landmark positions (like eye corners, ...)

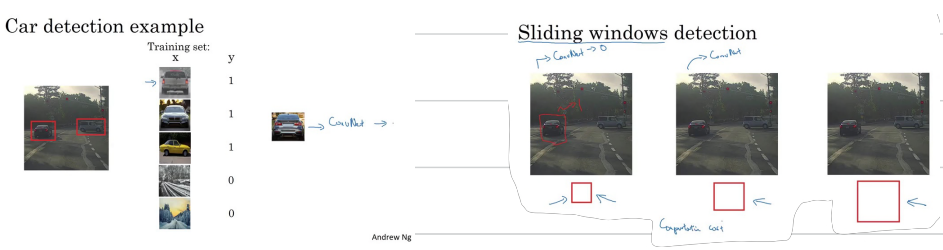
* identity of these landmark should remain same across all training



Detection example (using sliding window)

→ we train a net to classify tightly cropped images of (ex. car)

when trained for detection from original big images, sliding window is used to give cropped images to trained net.



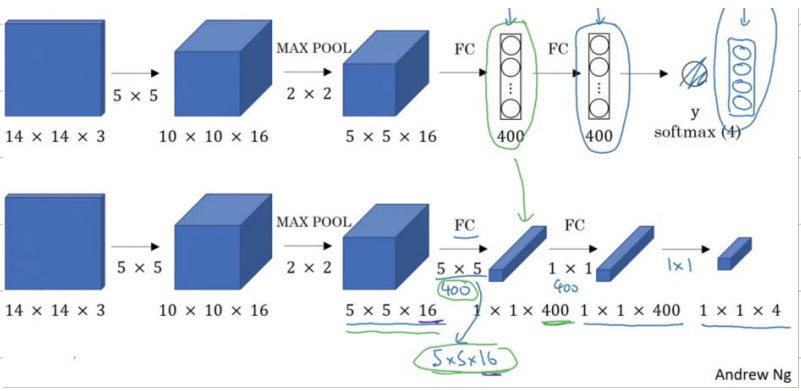
could be repeated with different window sizes and hope to detect objects.

Main disadvantages : ① computational cost : implementation with conv. layers.

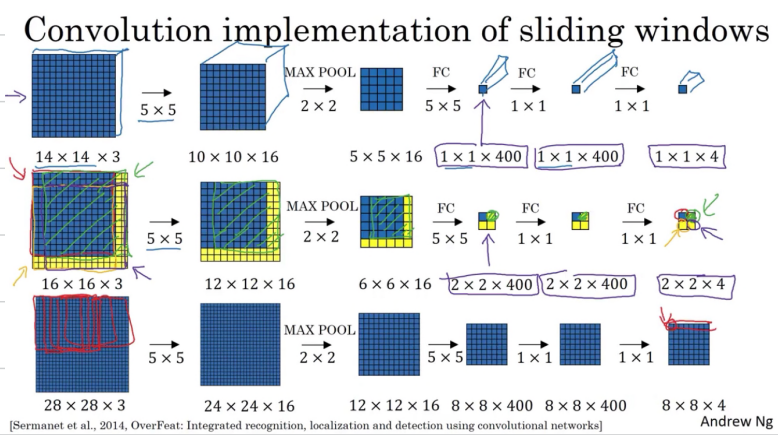
② Bounding boxes are not accurate.

Turning FC layer into convolutional layers.

Upper network could be represented in below Fully convolutional network.



- Now, we could use an structure like below to implement sliding window by convolution. Removing so much redundant computation.

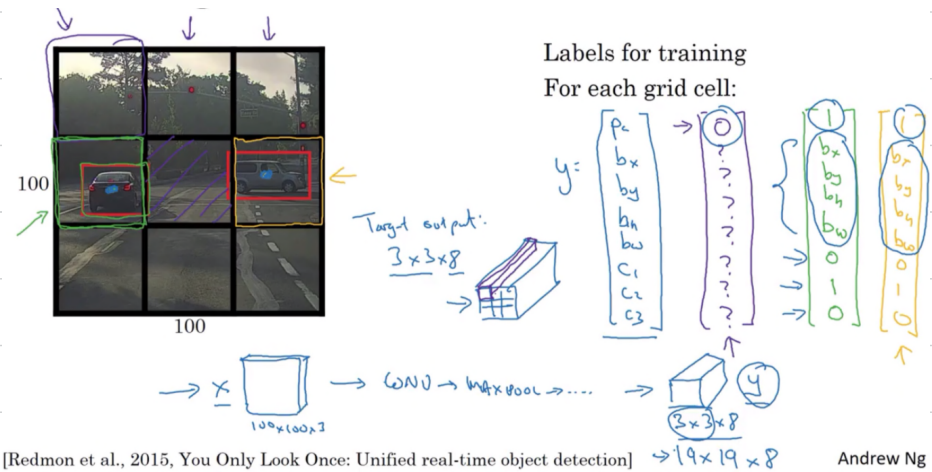


→ but still the position of bounding box is not accurate.

Bounding box prediction.

algorithms described above, does not necessarily predicts the accurate bounding box.

Yolo algorithm (You only look once).



The main ideas of using Yolo algorithm:

→ Image are divided into grids (3x3 : example in above)

→ Network is trained on each grid cell and the output is similar to what is seen before.

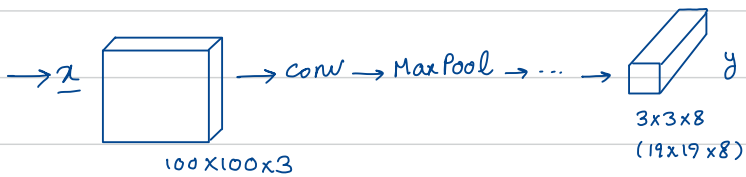
a) each object is just assigned to one grid (even if it has overlap).

b) b_h, b_w could be >1, means box could extend to other grids.

c) the total output for each image should be 3x3x8

d) In practice, we use much finer grids than 3x3 (like 19x19)

- Reducing chance of having multiple object in one grid.



Main advantages:

① Precise bounding boxes

② Coordinate of bounding box is not dictated by stride size of bounding box.

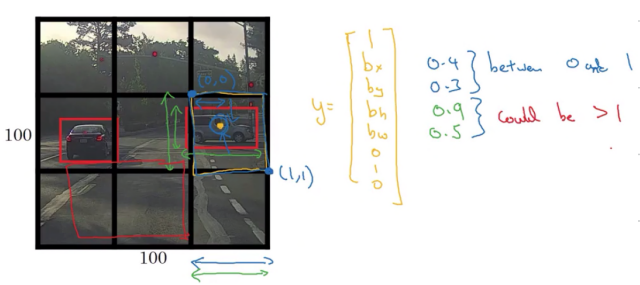
③ Convolutional implementation is used to save on computation.

- Net is not applied seperately 9 times (or 19x19) } Pretty efficient, very fast.

Specifying the bounding boxes:

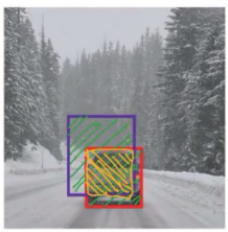
a) only the grid with center point has $p_c = 1$

b) b_x, b_y should be $0 \sim 1$, but b_h, b_w could extend to other grids.



Intersection over union

For evaluating the object detection algorithm.



$$= \frac{\text{size of } \square}{\text{size of } \square}$$, correct if IOU ≥ 0.5

normally ≥ 0.5 is used.

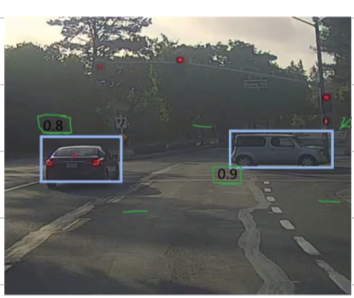
More generally, IOU is a measure of the overlap between two bounding box.

Non_max suppression

When we use grids, there is a good chance that many of the grids, which object overlap, detect same object (same obj detected in many)

→ Non_max cleans these multiple detections (one-box for one car)

- selecting bounding box with highest Probability (P_c)
- looks at IOU values between overlaping boxes and discards other boxes with high IOU with selected box.



Non max suppression algorithm

19× 19

- Each output prediction is: $\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$
- discard all boxes with $p_c \leq 0.6$
- while there are any remaining boxes:
  - Pick the box with largest p_c, → output as prediction
  - Discard any remaining box with IOU ≥ 0.5 with the box output in previous step.

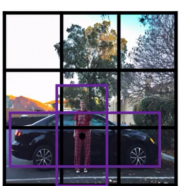
- if multiple classes exist, it's better to run the non-max suppression independently for each class.

Anchor box

So for we have looked at examples of one object in each grid.

Overlapping objects:

- Previously, each object in Training image is assigned to grid cell that contains the object middle point. each grid → $\begin{bmatrix} p_c \\ b_x \\ b_y \\ \vdots \\ c_3 \end{bmatrix}$, for 3×3 grid: y: 3×3×8
- Anchor box algorithms:
- with overlapping objects, we could use anchor boxes. each object in training to grid which has the middle point and anchor box which has highest IOU.



Anchor box 1 Anchor box 2

Anchor 1

$y = \left[\begin{array}{c} p_c \\ b_x \\ b_y \\ \vdots \\ c_3 \\ \hline p_c \\ b_x \\ \vdots \\ c_3 \end{array}\right]$

Anchor 2

for 3×3 grid with 2 anchor:

y: 3×3×16

→ example:

Anchor 1 : human

" 2 : car

$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$ anchor box 1, anchor box 2

first example for both human and car

second example only car exist in grid

The cases that this algorithm does not handle well:

1. More than two object in grid
2. 2 object with similar anchor box shape.

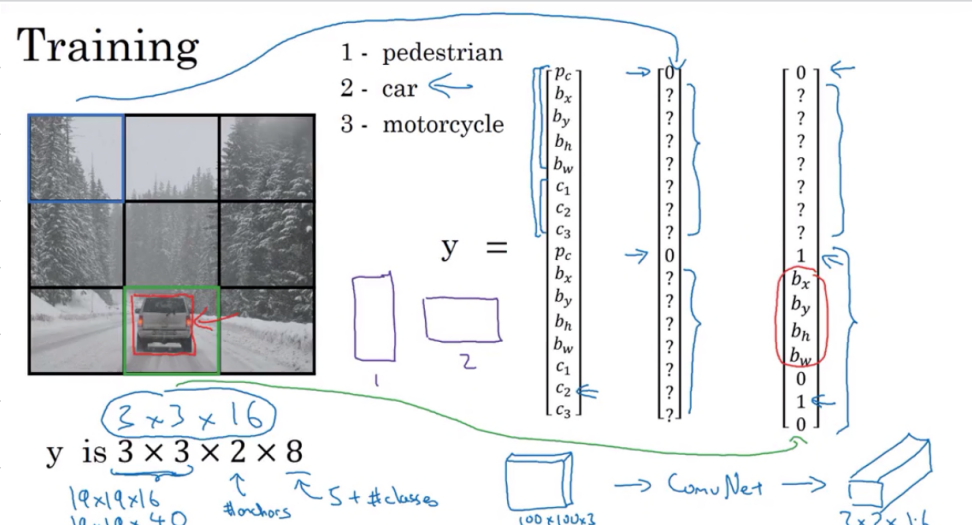
In practice, these scenarios should not happen often, but as solution:

- Much finer grids could be used to help to prevent above scenarios.
- Some default tie-breaker routines should be implemented.

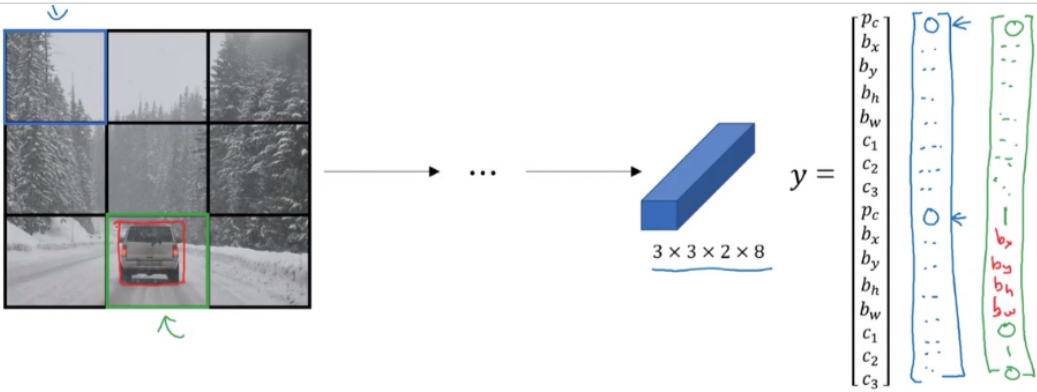
→ for selecting the shape of anchor-box for each class, we could use k-means over data of such class and find the most probable shape for given class

Yolo object detection (grid example of 3x3)

example of 3class, 2 anchor box



Making predictions:



→ Outputing the non-max suppressed outputs:

a. For each grid cell, get 2 predicted bounding boxes.

b. Get rid of low probability predictions,

c. For each class (pedestrain, car, motorcycle) use non-max suppression to generate final predictions.

a)

b)

c)

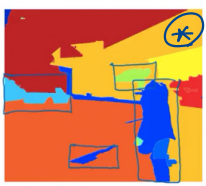
Region Proposal R-CNN

Idea is that we don't run CNN in all image (or even with sliding win)

- First we run a segmentation algorithm to find regions which is probable to have objects, then we ran the network only on those regions.

Region proposal: R-CNN





→ output of segmentation algorithm

→ probable regions given to CNN

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]

Andrew Ng

* even faster implementations are still slower than YOLO.

Face verification vs Face recognition

① Verification: 1:1

- Input image, name/ID → Output: weather image is for claimed name.

② Recognition: 1:K

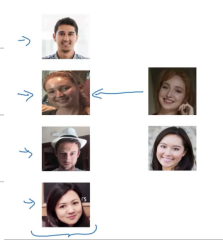
- has database of K person (K ~ 100)
- Input image ⟶ output ID if the image is any of K person (or not rec.)

verification is much simpler task.

One shot learning,

learning from one example to recognize the person again.

(We dont want to train a Network for only K person with lots of data from them)

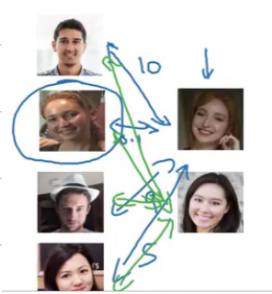


solution is to use similarity function:

Learning a similarity function

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

if d(img1, img2) < τ same } verification.
> τ different }

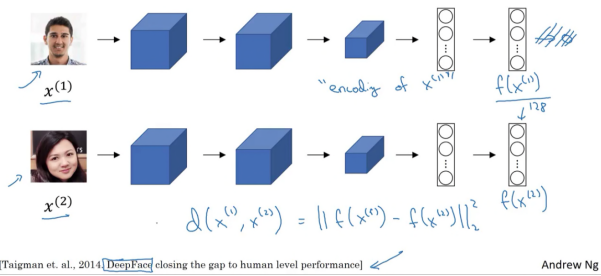


Siamese Network

To learn the disimilarity function

The idea is that, we train conv Net in a way that, if input image are from same person, to be close and v.s.

→ here, in fact outputs are the encoding of each photo to vector Y



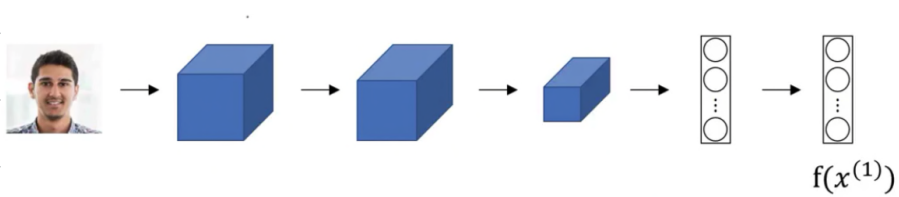
Goal of learning

given an input image $x^{(i)}$, we have an output encoding vector.

learn the parameter of NN, so that.

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

" " " different person " " is large.



→ the cost function which defined for this purpose is called triplt loss function.

Triplet_loss function

we defined the objective based on 3 input images in 2 pairs, in each pair one "Anchor" image is the same and the other is positive (another image of same person) and negative (not same person)

Anchor A | Positive P

Anchor A | Negative N

$$\|f(A)-f(P)\|^2 < \|f(A)-f(N)\|^2 - \alpha$$

$d(A,P)$ $\quad$ $d(A,N)$

α: similar to concept in SVM is margine, to prevent NN to learn trivial answer $f(\cdot)=\vec{0}$

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

So: given 3 image A, P, N:

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

then overall cost function for NN:

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

example: we have 10k picture from 1k person, we form in our database such Triplet examples. it is necessary to have more than 1 picture from any.

→ choosing A, P, N examples are important, if they selected randomly it is very easy to satisfy the loss function. $d(A,P)+\alpha \leq d(A,N)$

we should choose the triplets which are hard to train on!

→ computationally is also important, so much randomly selected one easy and g.d does not do to much to learn something.

correct triplet choice → speedup learning → better encoding.

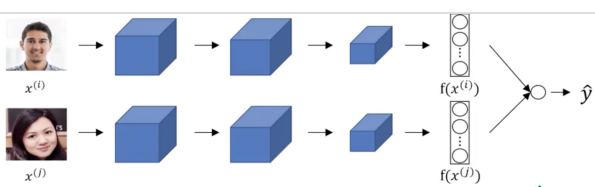
Paper: Schroff et al, 2015, faceNet: A unified embedding for face reco. & clustering

commercial systems : start from 1M images ⟶ 10M images. (Maybe 100M)

→ sine such db are not easily available, some has shared their pre-trained Network.

learning face verification as binary classification

→ another way of training Siamese network, could be like a binary classifi problem.



→ at end stage we add a logistic regression, $y = \begin{cases} 1 & \text{, inputs same persons.} \\ 0 & \text{, different persons.} \end{cases}$

↳ these two net, actually one same or parameters are tied together

cost function will be

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

↳ element wise difference

or othe measure also could be used

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

idea paper: Deep face:

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

→ This method as well could work pretty well too.

→ computational note: obviously at inference time, for example in case of face verification, all stored image could be just the encoding, not real image, to remove the need to calcute the same output over and over!

example training:

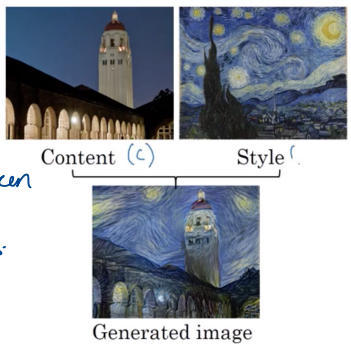
x	y
	1
	0
	0
	1

Neural style transfer.

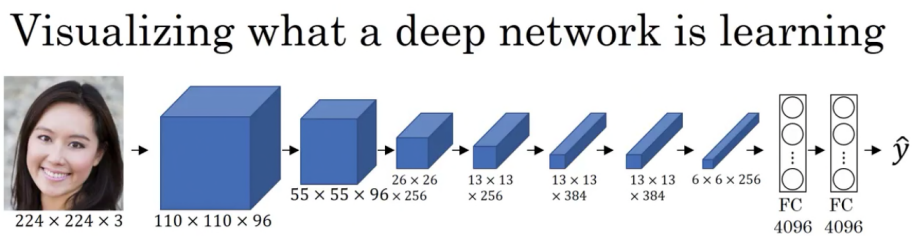
the goal is to re-create image in style of other image.

Two input: Content: the image that content is taken

Style: the style is copied from this.

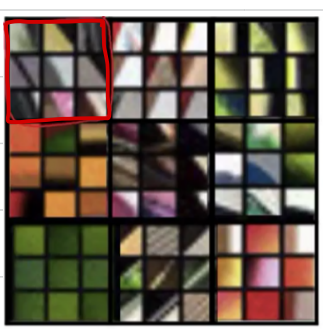


what a deep conv Net learns?

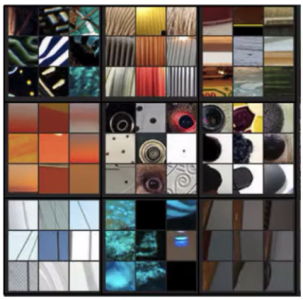


Zeiler and Fergus, 2013, Visualizing and understanding convolutional networks

- Pick a unit in layer n, and find all the patches that maximize the unit activation.
- example for a unit in layer 1 (red box) and for 9 unit (The whole Picture)
- it is cleer that each unit has learned a certain type of features (like edge, color, horizontal or vertical patterns).



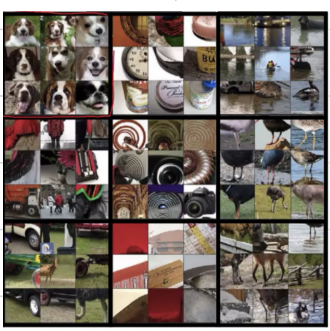
- Example for units in layer 2, as we go deeper the units sees bigger picture and bigger patterns.



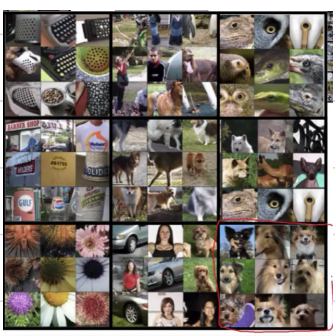
and similarly for other layers (3,4,5)

3

4



5



as we go deeper a bigger patterns and features are seen (like layer 4, unit 1 seems that responds to similar breeds of dogs. / 3 for body of water, ...)

Cost Function: Gatys et al, 2015 A neural algorithm of artistic style.

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

↓ How similar is content → How similar is style

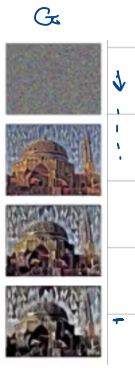
→ Having two h.p α, β are kind of redundant.

Finding the generated image G

1. Initialize G randomly
2. Use gradient decent to minimize $J(G)$

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

C S



---

defining Content cost function:

To define content cost function, it's important to correctly select the right hidden layer, not too deep, not too shallow.

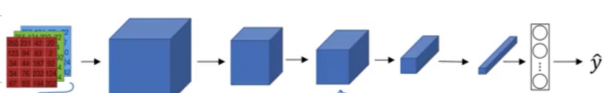
- select a pretrained ConvNet (e.g VGG network)
- the cost is defined as how close are the activation from layer l, for both C, G

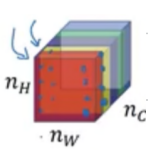
$$\rightarrow J_{content}(C,G) = \frac{1}{2} \| a^{[l]C} - a^{[l][G]} \|^2$$

defined as norm2 distance.

Style cost function:

For style, cost is defined as the correlation across the channels for the selected layer l:





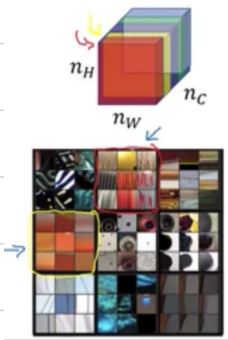
→ to define a measure for style

imagine this layer is selected.

we look the correlation for each activation across channels n_c.

Intuition:

let say first chanel red corresponds to red box shown activation response and 2nd yellow channel to yellow box.



→ if we measure the correlation, this gives us a measure for example how often the horizontal pattern occure with orang tint pattern (a sense of style in image).

→ Now if we measure degree of correlation for generated image across the channels, we could see how close the style is to that of original style reference image.

→ Style Matrix:

let $a^{[l]}_{i,j,k}$ = activation at (i,j,k) → then style matrix $G^{[l]}$ is $n_C^{[l]} \times n_C^{[l]}$ (H, W, n_C)

→ for one element in matrix $G^{[l]}$: $G^{[l]}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l]}_{ijk} \, a^{[l]}_{ijk'}$, $k, k' = 1, \dots, n_C^{[l]}$

(actually this is un-normalized cross covariance, not cross correlation)

→ this Matrix is calcuted for both style image and generated image.

→ In linear algebra, these are indeed "gram matrix"

Finally style cost function is defined as:

$$J^{[l]}_{style}(S,G) = \frac{1}{2n_W n_H n_C} \| G^{[l](S)} - G^{[l](G)} \|^2_F$$

↓ normalization ↙ Forbinius norm

normalization factor is not really important.

In practice, more overally visually pleasing results will taken if multiple layers are used as reference for the style. (considering both low/high level correlation between styles).

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

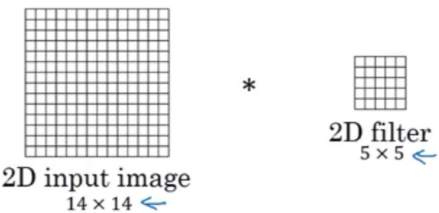
λ : new hyperparameter vector, weight for style of different layers.

and Finally overall cost function:

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

1D and 3D generalization

So far we saw the examples of convNets for 2D images, however similar concept could generalize for 1D and 3D data as well.



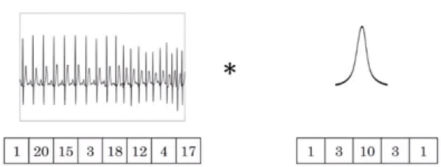
→ in 2D image examples, we saw previously how the filters captures information from different part of images. Image 14×14 * 5×5 → 10×10

2D image with n_c channel:

$$14 \times 14 \times n_c \quad * \quad \underbrace{5 \times 5 \times n_c}_{16 \text{ filters}} \quad : \quad 10 \times 10 \times 16$$

Similar for 1D signals:

imagine for example ECG signals of 14 Dimention (1D), then the filter is as well 1D (as shown in Figure).



$14 \quad * \quad 5 \quad = 10$

similar concept like having multiple channels for data and also number of filters are applicable here as well: ECG (14) x 3 chanel.

1D convolution: then filter: 5×3 (↕) → output 10

32 filter: $\underbrace{5 \times 3}_{32}$ → 10×32

→ generally for 1D data (sequence) we more use the RNNs, However using convNets one still possible and sometimes used.

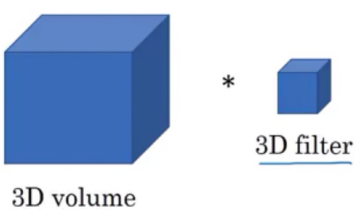
3D Data:

similar concepts for the 3D data,

example for 3D data are for example CAT scans, which multiple scans are capture in different slices.

→ multiple slices of CAT scan forms a 3D volume.
(They have spatial informations which useful to treat as 3D data, sometimes).

3D Convolution:



imagine 3D volume $14 \times 14 \times 14 \times n_c$

16 filter (3D) $5 \times 5 \times 5 \times n_c$

$= 10 \times 10 \times 10 \times 16$

other example for 3D dat, could be multiple frames from a movie clip, which we are interested as well in temporal information as well.