DEEP LEARNING SPECIALIZATION
COURSE 4/5

D-4-1

CONVOLUTIONAL NEURAL NETWORKS

→ SOME COMPUTER VISION PROBLEMS

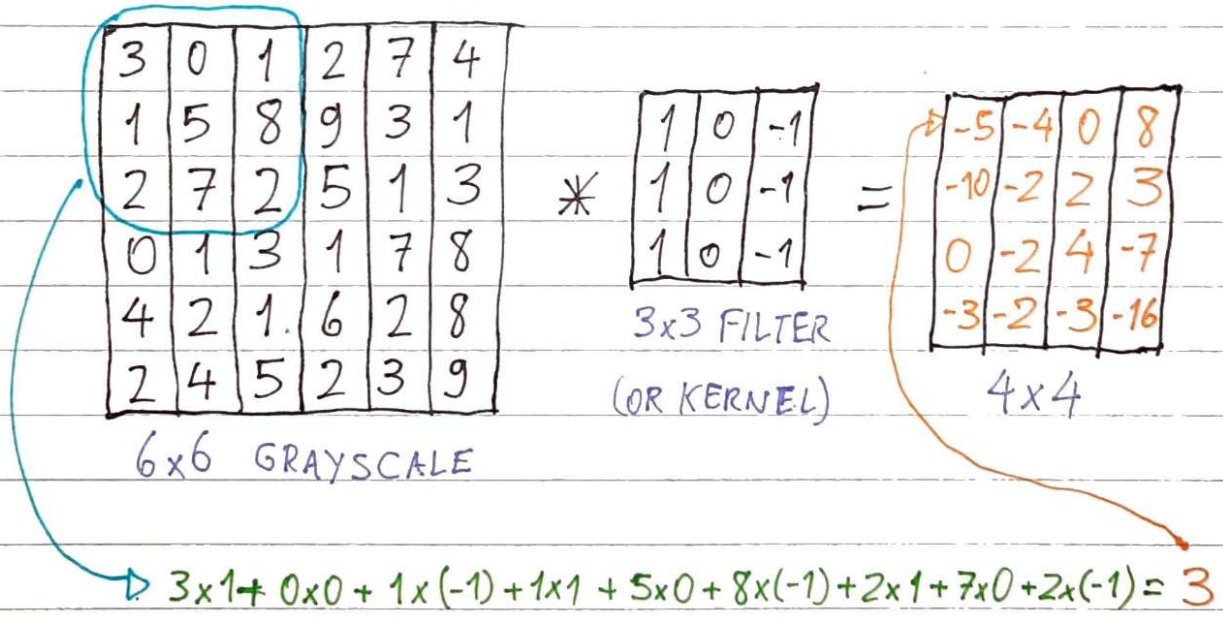
* IMAGE CLASSIFICATION (CAT/NOT CAT)
* OBJECT DETECTION (EXAMPLE: DETECT POSITION OF CARS
* NEURAL STYLE TRANSFER (IMAGE STYLIZATION)

→ DEEP LEARNING ON LARGE IMAGES

* FOR AN 64x64 RGB IMAGE: 12288 INPUTS
* FOR AN 1000x1000 RGB: 3 MILLION INPUTS
  - TOO COMPLEX

→ EDGE DETECTION

* VERTICAL EDGE DETECTION



- FUNCTION IN PYTHON: conv_forward
- TENSORFLOW: tf.nn.conv2d
- KERAS: Conv2D

* ANOTHER EXAMPLE OF VERTICAL EDGE DETECTION

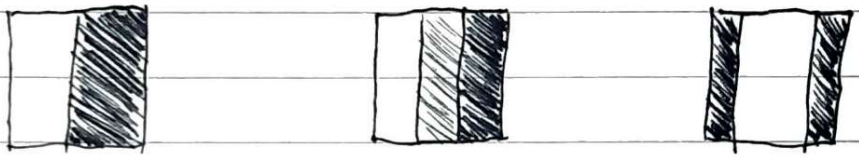
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

$*$

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

$=$

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



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

$*$

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

$=$

0	-30	-30	0
0	-30	-30	0
0	-30	-30	0
0	-30	-30	0

INVERTED.:

* VERTICAL AND HORIZONTAL EDGE DETECTIONS

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

VERTICAL FILTER

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

HORIZONTAL FILTER

→ MORE EDGE DETECTION

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

SOBEL FILTER: MORE WEIGHT TO CENTRAL ROW, MORE ROBUST

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

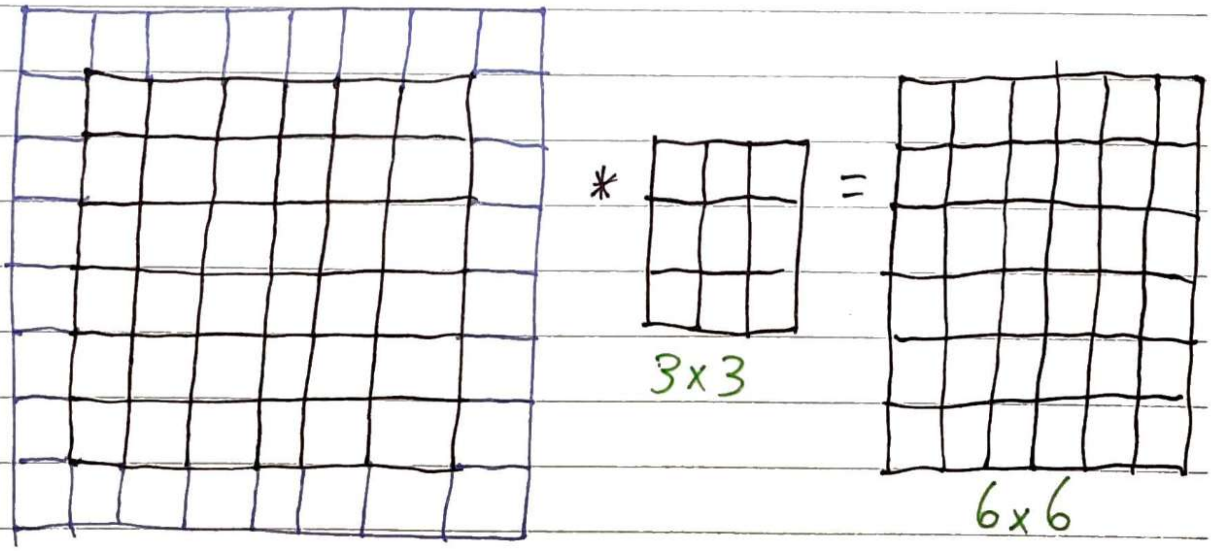
SCHARR FILTER

W_1	W_2	W_3
W_4	W_5	W_6
W_7	W_8	W_9

→ FILTER WITH LEARNABLE PARAMETERS (USING BACKPROP)

→ PADDING

* CONSIDERING AN IMAGE WITH DIMENSION $n \times n$ AND AN FILTER WITH DIMENSION $f \times f$:
  - RESULT OF CONVOLUTION WILL HAVE DIMENSION $(n-f+1) \times (n-f+1)$
  - SHRINKING OUTPUT
  - THROWING AWAY INFORMATION FROM EDGES
* PADDING: ADDING EDGES WITH ZEROS, SO THE CONVOLUTION RESULT WILL HAVE RESOLUTION $(n+2p-f+1) \times (n+2p-f+1)$



6x6 → PADDING 1: 8x8

→ VALID AND SAME CONVOLUTIONS

* VALID: NO PADDING

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

* SAME: PAD, SO THE OUTPUT SIZE IS THE SAME AS THE INPUT SIZE

$$p = \frac{f-1}{2} \longrightarrow f \text{ IS ODD}$$

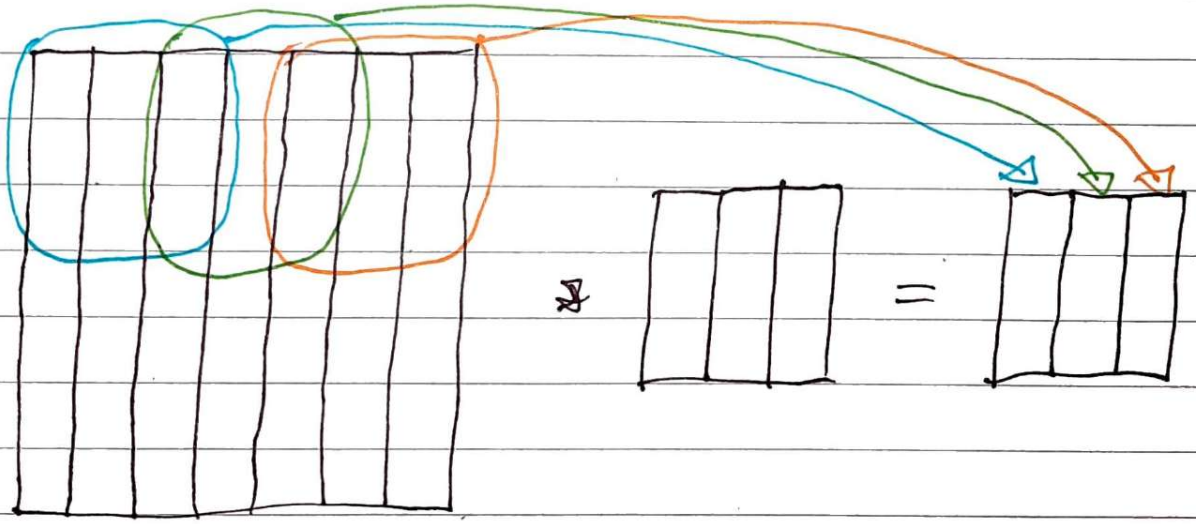
$$(n+2p) \times (n+2p) \quad * \quad f \times f \quad = (n+2p-f+1) \times (n+2p-f+1)$$

$$n-2p-f+1 \quad = \quad n+(\cancel{f-1})-\cancel{f+1} \quad = \quad n$$

→ STRIDED CONVOLUTIONS

* CONVOLUTIONS ARE MADE TAKING "STEPS"

* EXAMPLE: 7x7 * 3x3 (STRIDE = 2)



* IN A CONVOLUTION BETWEEN nxn AND fxf WITH PADDING p AND STRIDE s:

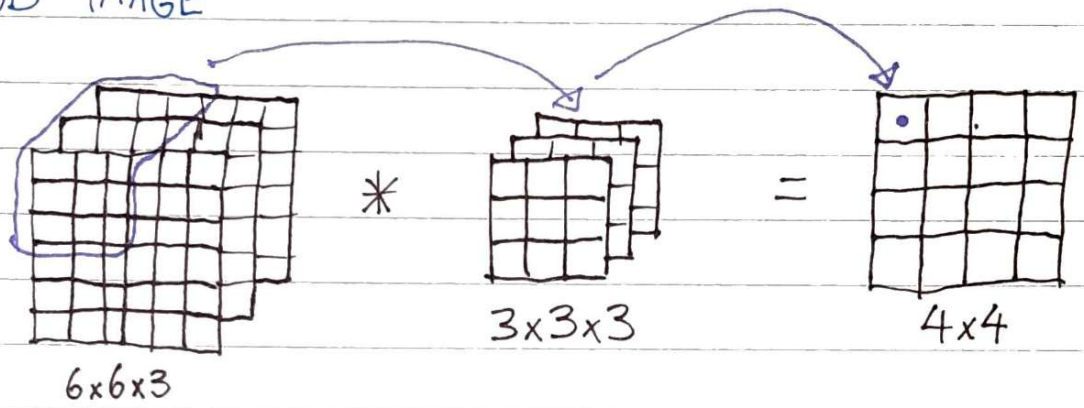
- CONVOLUTION WITH DIMENSIONS $\left\lfloor \frac{n+2p-f+1}{s} \right\rfloor \times \left\lfloor \frac{n+2p-f+1}{s} \right\rfloor$

TECHNICAL NOTE ON CROSS-CORRELATION VS. CONVOLUTION

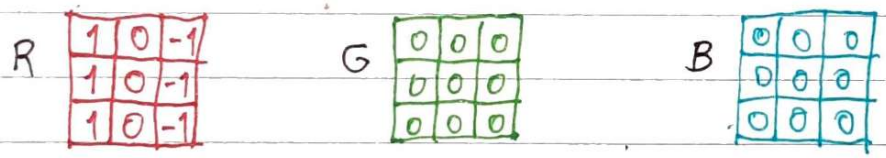
* CONVOLUTION IN MATH TEXT BOOK: INVERSION OF THE FILTER MATRIX IN HORIZONTAL AND VERTICAL
* "CONVOLUTION" USED IN MACHINE LEARNING IS ACTUALLY CROSS-CORRELATION: FILTER IS NOT FLIPPED

CONVOLUTIONS ON VOLUME

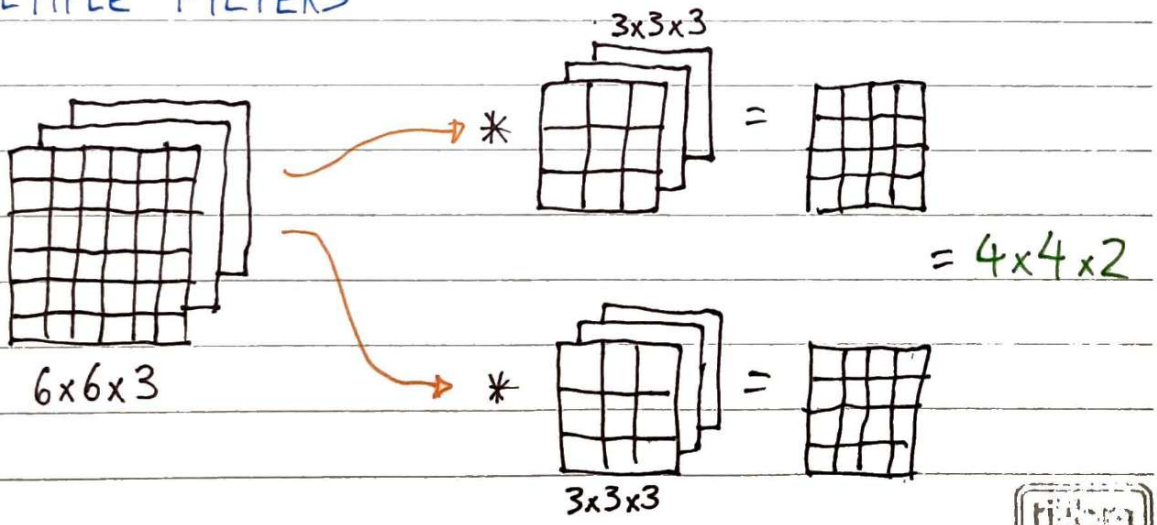
* RGB IMAGE



* EDGE DETECTION IN R CHANNEL



* MULTIPLE FILTERS



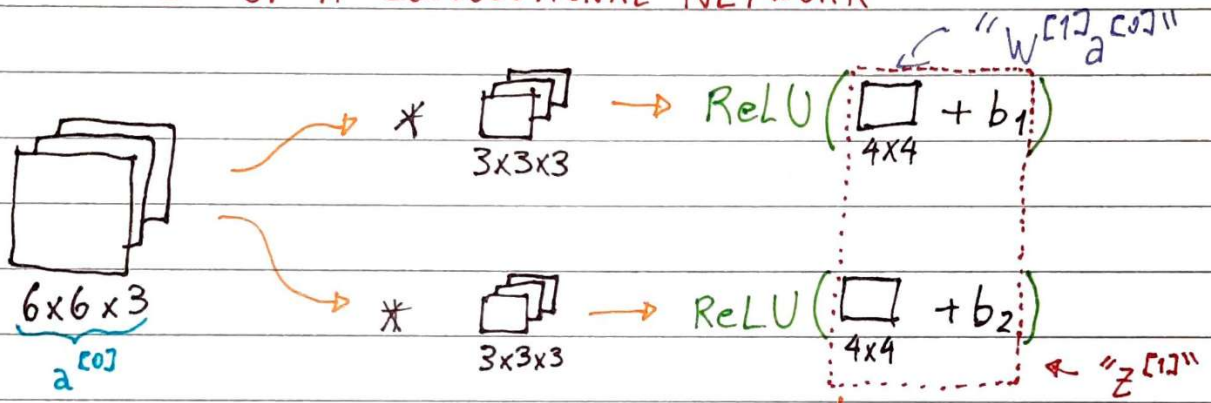
* SUMMARY

$n \times n \times n_c \;\; * \;\; f \times f \times n_c = (n-f+1) \times (n-f+1) \times n_c'$

$6 \times 6 \times 3 \;\; * \;\; 3 \times 3 \times 3 = 4 \times 4 \times 2$

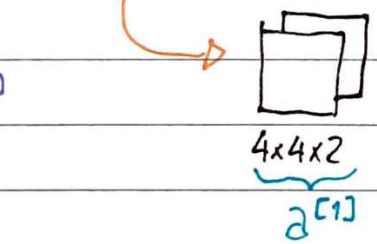
↑ NUMBER OF CHANNELS (OR DEPTH)

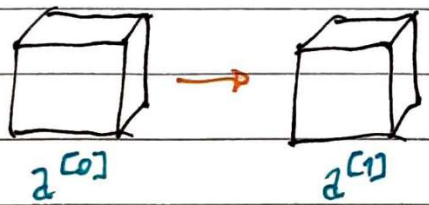
→ ONE LAYER OF A CONVOLUTIONAL NETWORK



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

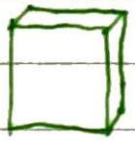
$a^{[1]} = g(z^{[1]})$





* NUMBER OF PARAMETERS IN A CONVOLUTIONAL LAYER

- IF YOU HAVE 10 FILTERS THAT ARE 3X3X3 IN ONE LAYER OF A NEURAL NETWORK, HOW MANY PARAMETERS DOES THAT LAYER HAVE?



$3 \times 3 \times 3 = 27 \cdot W + 1 \cdot b = 28$ PARAMETERS

28×10 FILTERS $= 280$ PARAMETERS

* NOTATION

- IF LAYER l IS A CONVOLUTION LAYER:

$f^{[l]}$ = FILTER SIZE
$p^{[l]}$ = PADDING
$s^{[l]}$ = STRIDE
$n_c^{[l]}$ = NO. OF FILTERS

INPUT: $n_h^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$

OUTPUT: $n_h^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

$$n_{h,w}^{[l]} = \left[\frac{n_{h,w}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right]$$

EACH FILTER IS: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

ACTIVATIONS:

$a^{[l]} \rightarrow n_h^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

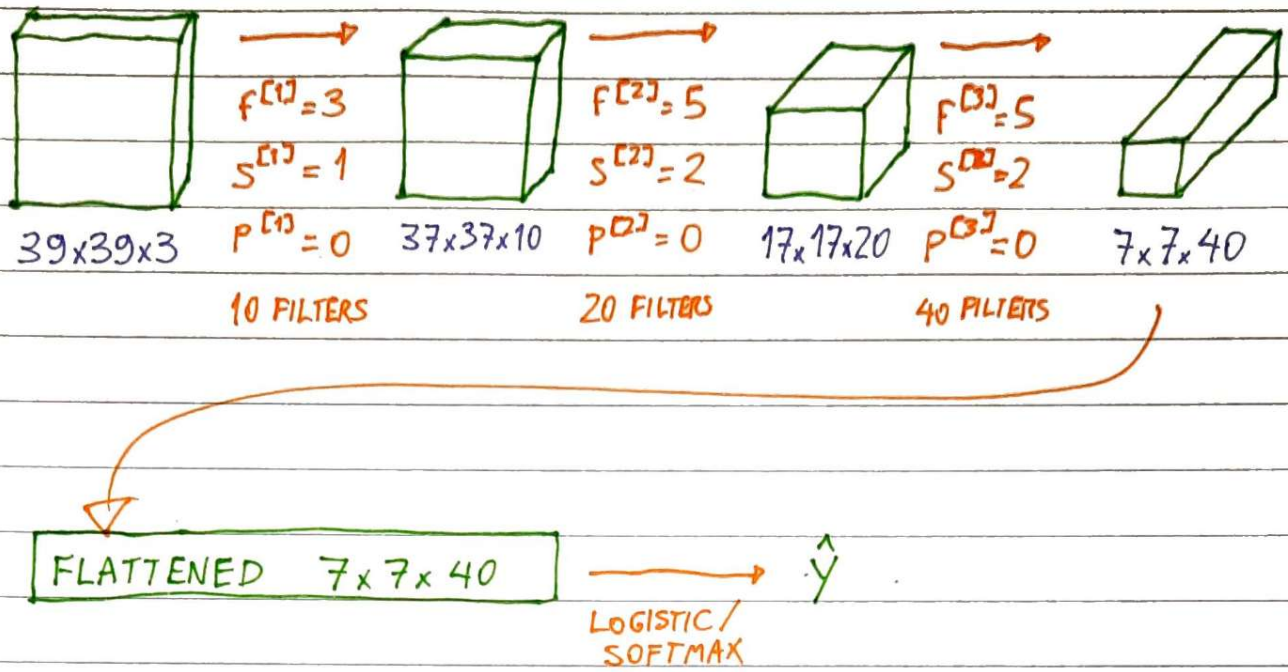
$A^{[l]} \rightarrow m \times n_h^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

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

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

- THIS NOTATION ISN'T A STANDARD PATTERN, SO IT MAY BE DIFFERENT IN SOME FRAMEWORKS

→ EXAMPLE OF A SIMPLE CONVOLUTIONAL NETWORK

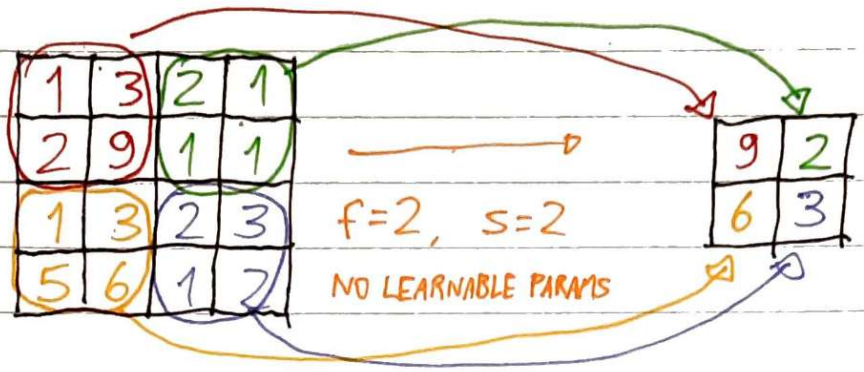


→ TYPES OF LAYER IN A CONVOLUTIONAL NETWORK

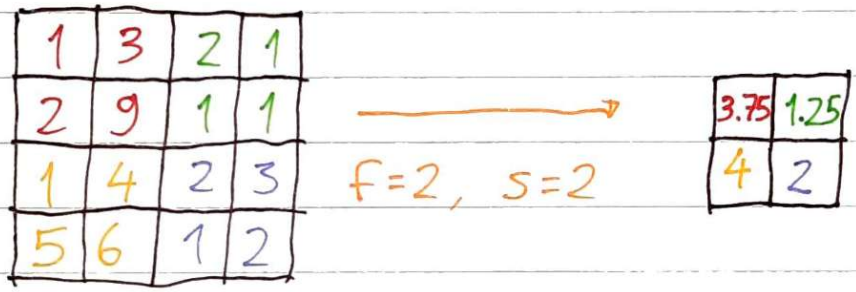
* CONVOLUTION (CONV)
* POOLING (POOL)
* FULLY CONNECTED (FC)

POOLING LAYERS

* LAYERS WITH FIXED COMPUTATION AND NO LEARNABLE PARAMETERS
* MAX POOLING: SELECT MAXIMUM VALUE OF EACH REGION



* AVERAGE POOLING: COMPUTES AVERAGE OF EACH REGION



* SUMMARY OF POOLING

- HYPERPARAMETERS:

f - FILTER SIZE

S - STRIDE

MAX OR AVERAGE POOLING

P - PADDING (IN MOST CASES ISN'T USED IN POOLING)

NO LEARNABLE PARAMETERS

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

FULLY CONNECTED (FC) LAYERS

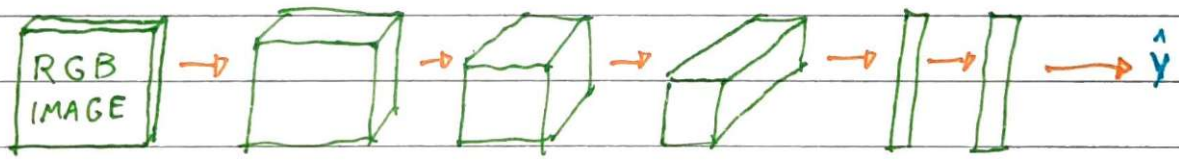
* TYPICAL LAYERS AS FOUND IN STANDARD NEURAL NETWORKS
* IN MOST OF THE CASES, FC LAYERS ARE THE LATER LAYERS OF CONVOLUTIONAL NETWORKS

WHY CONVOLUTION?

* PARAMETER SHARING
  - A FEATURE DETECTOR (SUCH AS A VERTICAL EDGE DETECTOR) THAT'S USEFUL IN ONE PART OF THE IMAGE IS ALSO USEFUL IN ANOTHER PART OF THE IMAGE
* SPARSITY OF CONNECTIONS
  - IN EACH LAYER, EACH OUTPUT VALUE DEPENDS ONLY ON A SMALL NUMBER OF INPUTS

TRAINING A C.N.N.

TRAINING SET $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$

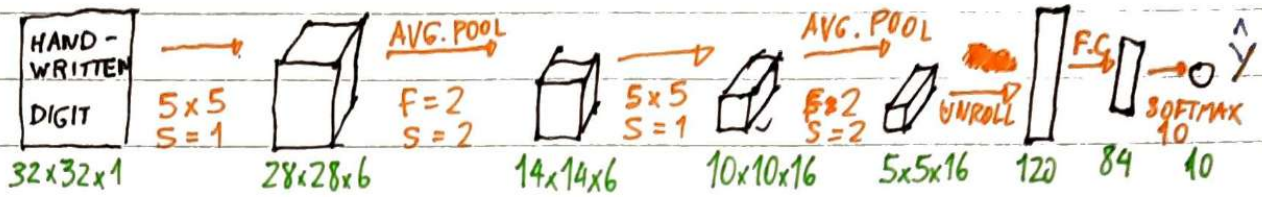


- COST $J = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$

- USE GRADIENT DESCENT OR OTHER OPTIMIZATION ALGORITHM TO OPTIMIZE PARAMETERS TO REDUCE J

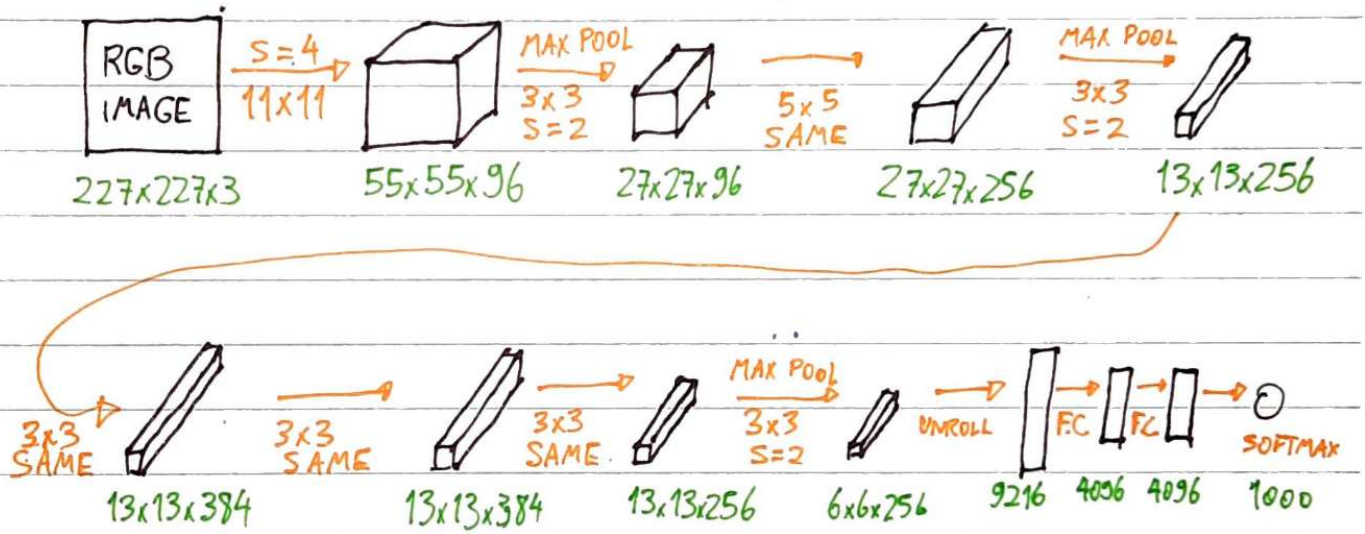
→ CLASSIC CONVOLUTIONAL NETWORKS

* LENET-5 (APPROX. 60K PARAMETERS)



- FROM EARLY TO LATER LAYERS: $n_H, n_W \downarrow$ $n_C \uparrow$
- LAYER SEQUENCE: CONV → POOL → CONV → POOL → FC → FC → OUTPUT

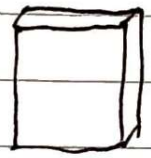
* ALEXNET (APROX. 60M PARAMETERS)



- SIMILARITY TO LENET, BUT MUCH BIGGER
- ReLU
- AT PAPER: GPU'S WERE SLOWER, THEY USED MULTIPLE GPU'S
- AT PAPER: THEY USED LOCAL RESPONSE NORMALIZATION (TODAY WE KNOW LRU DOESN'T HELP TOO MUCH)

* VGG-16 (138 MILLION PARAMETERS)

- ALL CONV LAYERS WITH 3x3 FILTERS, S=1, SAME CONVOLUTION
- ALL POOLING LAYERS ARE MAX POOLING 2x2, S=2



224x224x3 → (CONV 64 x2) → 224x224x64 → POOL → 112x112x64 → (CONV 128 x2) → 112x112x128 → POOL → 56x56x128 → (CONV 256 x3) → 56x56x256 → POOL → 28x28x256 → (CONV 512 x3) → 28x28x512 → POOL → 14x14x512 → (CONV 512 x3) → 14x14x512 → POOL → 7x7x512 → FC 4096 → FC 4096 → SOFTMAX 1000

- SIMPLE AND SYSTEMATIC ARCHITECTURE
- $n_H, n_W \downarrow$ $n_C \uparrow$

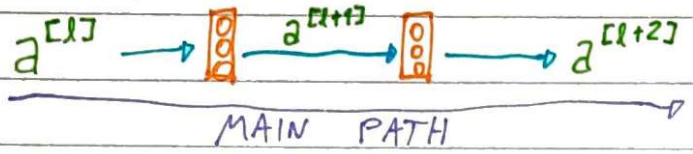
D-4-13

→ RESIDUAL NETWORKS (RESNETS)

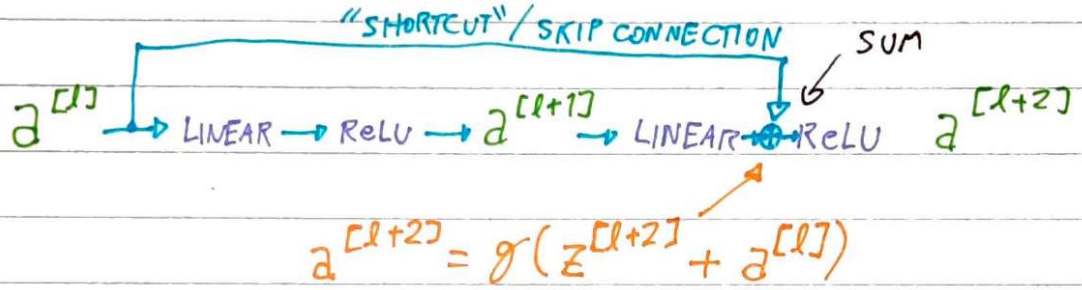
* DEEP NEURAL NETWORKS ARE HARD TO TRAIN

- VANISHING AND EXPLODING GRADIENTS

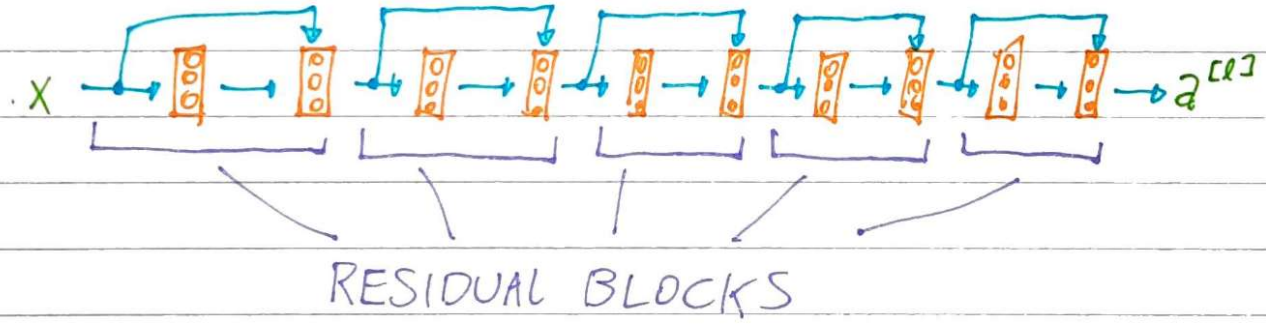
* RESIDUAL BLOCK



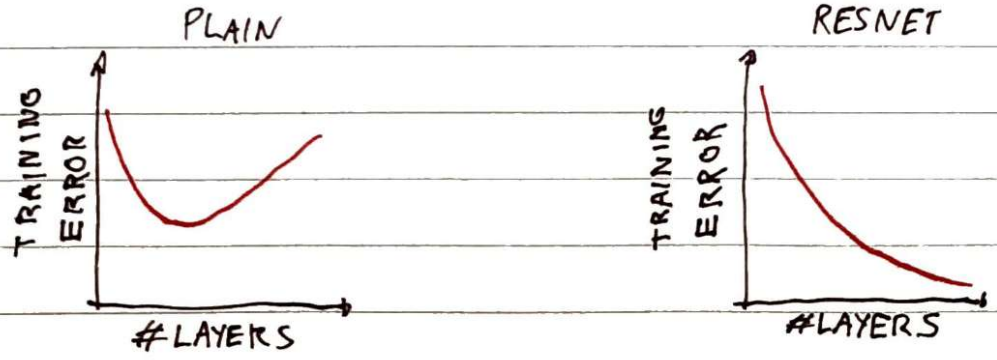
- HOW RESIDUAL BLOCKS [redacted] WORK?



* RESIDUAL NETWORK EXAMPLE

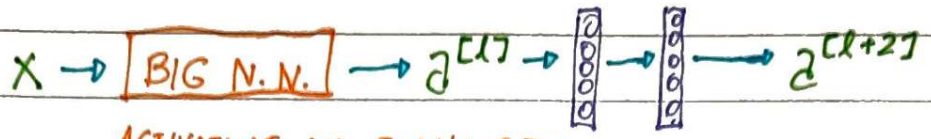


* TRAINING ERROR VS. # LAYERS AT PLAIN AND RESNET



→ WHY DO RESIDUAL NETWORKS WORK?

* EXAMPLE



ACTIVATIONS IN ReLU: $a \geq 0$

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]}) \quad \leftarrow \text{ASSUMING } z^{[l+2]} \text{ AND } a^{[l]} \text{ HAVE THE SAME DIMENSIONS}$$
$$= g(w^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

IF $w^{[l+2]} = 0$ AND $b^{[l+2]} = 0$:

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

IDENTITY FUNCTION IS EASY FOR RESIDUAL BLOCK TO LEARN

• IF $z^{[l+2]}$ AND $a^{[l]}$ DOESN'T HAVE SAME DIMENSIONS:

EXAMPLE: $a^{[l]} \rightarrow 128$ / $a^{[l+2]} \rightarrow 256$:

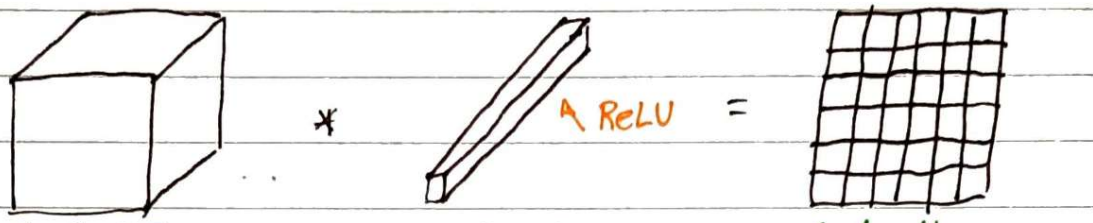
CALCULATE $g(z^{[l+2]} + W_s \cdot a^{[l]})$

$W_s \in \mathbb{R}^{256 \times 128}$

W_s CAN BE A MATRIX OF PARAMETERS TO BE LEARNED OR A FIXED MATRIX TO IMPLEMENT ZERO PADDING

→ 1×1 CONVOLUTION

* IN A SINGLE CHANNEL LAYER: ONLY MULTIPLYING NUMBER
* IN A MULTIPLE CHANNEL LAYER: ELEMENT-WISE PRODUCT BETWEEN NUMBERS OF THE CHANNEL AND NUMBERS ON THE FILTER



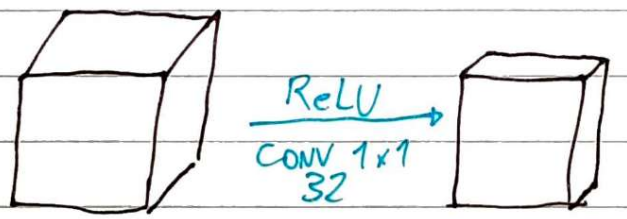
6×6×32 | 1×1×32 | 6×6×# FILTERS

- BASICALLY HAVING A F.C. NET THAT APPLIES TO EACH OF THE DIFFERENT POSITIONS

32 → # FILTERS ($n_c^{[l+1]}$)

- 1×1 CONVOLUTION ALSO CALLED AS NETWORK IN NETWORK

* USING 1×1 CONVOLUTIONS



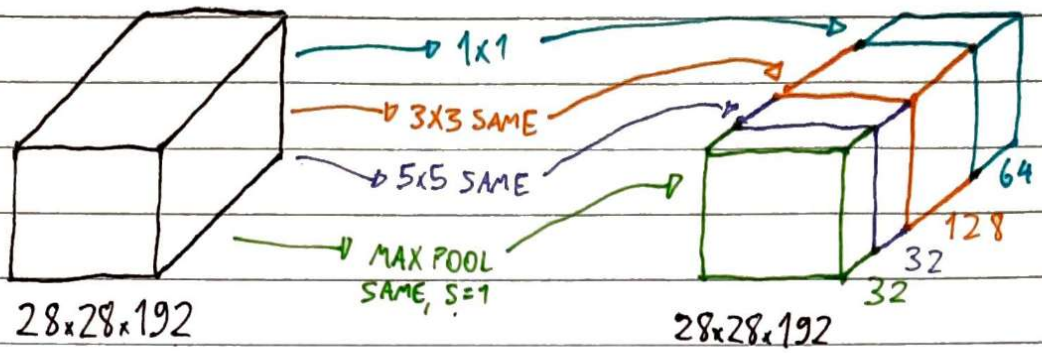
28×28×192 | 1×1×192 (32 FILTERS) | 28×28×32

IF 192 FILTERS: INTRODUCE NON-LINEARITY ON THE SISTEM

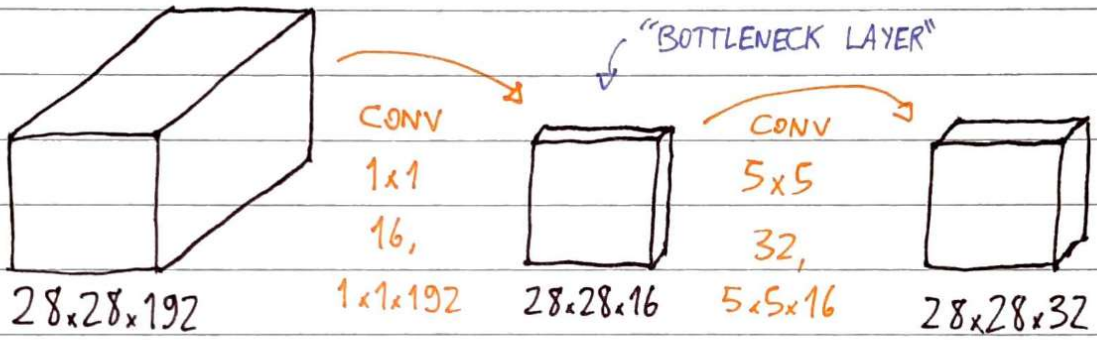
D-4-16

→ MOTIVATION FOR INCEPTION NETWORK

* MANY DIFFERENT CONVOLUTIONS STACKED ON A SINGLE OUTPUT



* PROBLEM: COMPUTATIONAL COST
* TO SOLVE COST PROBLEM: SHRINKING WITH 1x1 CONVOLUTION



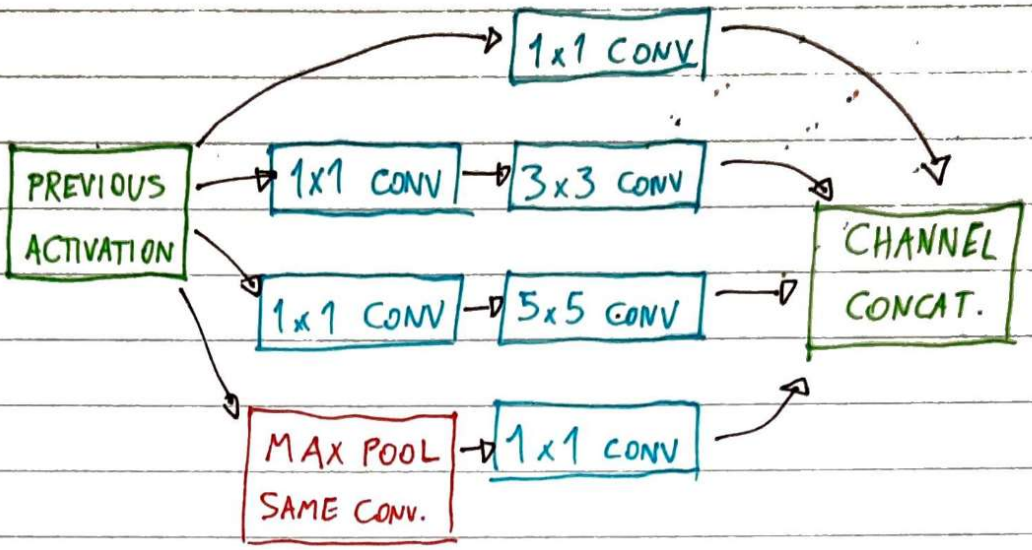
- WITHOUT BOTTLENECK LAYER: 120 MILLION MULTIPLICATIONS
- WITH BOTTLENECK LAYER: 12.4 MILLION MULTIPLICATIONS

- BOTTLENECK LAYER DOESN'T AFFECT NEGATIVELY THE NETWORK PERFORMANCE

D-4-17

→ INCEPTION NETWORK

* INCEPTION BLOCK (EXAMPLE)



* INCEPTION NETWORK

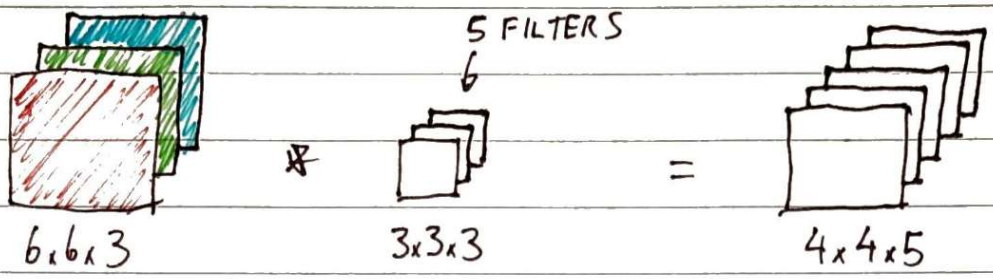
- INCEPTION BLOCKS ON CASCADE
- INTERMEDIATE OUTPUTS HELPS TO EVALUATE PERFORMANCE OF THE NETWORK

→ MOTIVATION FOR MOBILENETS

* LOW COMPUTATIONAL COST AT DEPLOYMENT
* USEFUL FOR MOBILE AND EMBEDDED VISION APPLICATIONS
* KEY IDEA: NORMAL VS. DEPTHWISE-SEPARABLE CONVOLUTIONS

→ DIFFERENCE BETWEEN NORMAL AND DEPTHWISE-SEPARABLE CONVOLUTION

* NORMAL CONVOLUTION



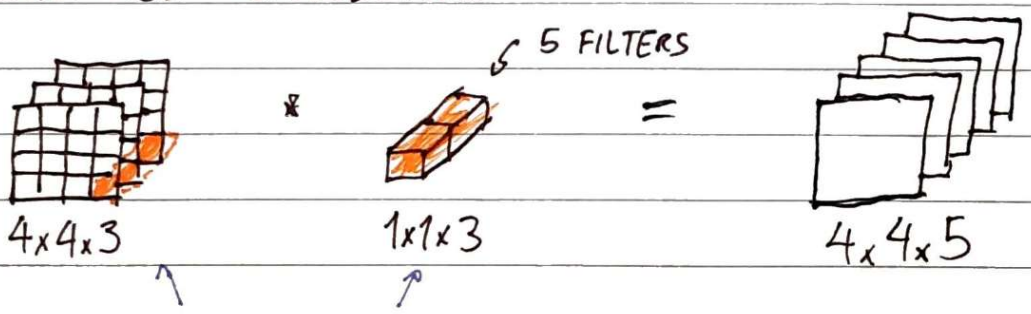
- COMPUTATIONAL COST: # FILTER PARAMS * # FILTER POSITIONS * # OF FILTERS

2160 = 3x3x3 x 4x4 x 5

* DEPTHWISE-SEPARABLE CONVOLUTION

- SEPARATE CONVOLUTION IN TWO STEPS TO REDUCE COMPUTATIONAL COST
- DEPTHWISE CONVOLUTION

- POINTWISE CONVOLUTION



* COMPUTATIONAL COST FOR DEPTHWISE-SEPARABLE CONVOLUTION
  - DEPTHWISE: $(3 \times 3) \times (4 \times 4) \times (3) = 432$
  - POINTWISE: $(1 \times 1 \times 3) \times (4 \times 4) \times (5) = 240$
  - TOTAL: $432 + 240 = 672$

$672 < 2160$ → MORE EFFICIENT AT 672

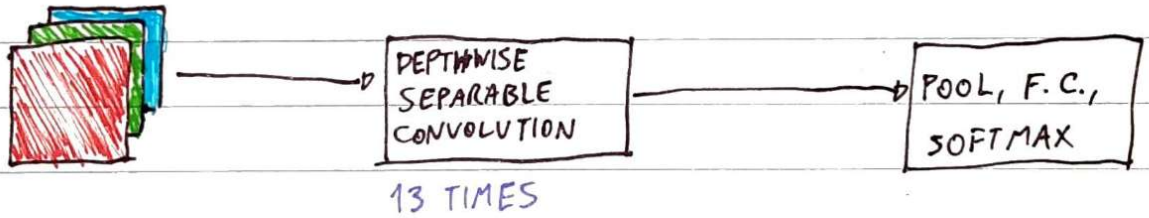
- PERCENTAGE OF COST BETWEEN NORMAL AND DEPTHWISE-SEPARABLE

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

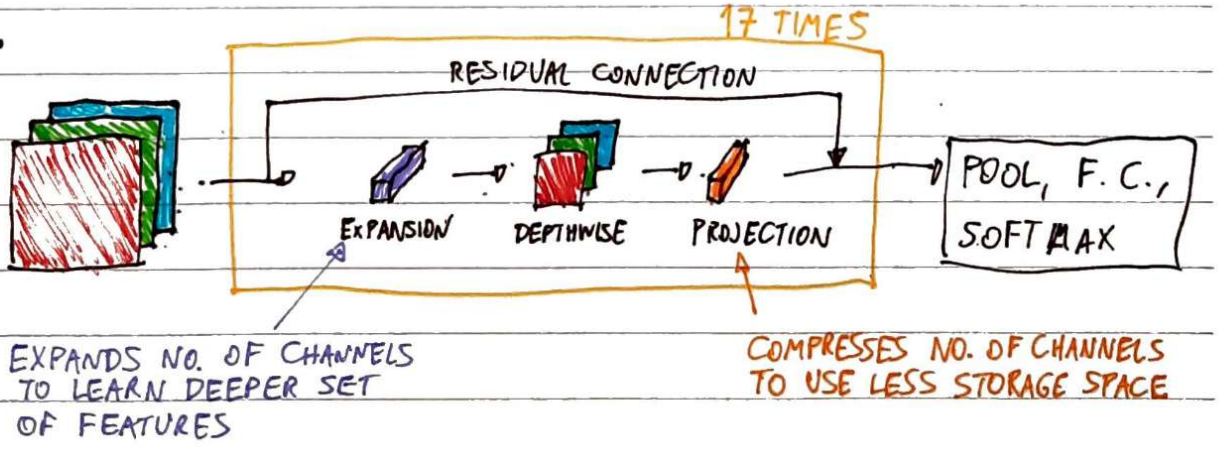
- DEPTHWISE-SEPARABLE CONVOLUTION WORKS FOR ANY NUMBER OF CHANNELS

→ MOBILENET ARCHITECTURE

* MOBILENET V1



* MOBILENET V2



D-4-20

EFFICIENTNET

* HOW TO TUNE N.N. TO ~~AN~~ SPECIFIC PLATFORM?
  - TUNE IMAGE RESOLUTION, N.N. DEPTH AND LAYER WIDTHS
  - OPEN SOURCE IMPLEMENTATION OF EFFICIENTNET

USING OPEN-SOURCE IMPLEMENTATIONS

* LOOK FOR OPEN-SOURCE IMPLEMENTATIONS ON GITHUB
* YOU CAN ADAPT THE N.N. OR USE TRANSFER LEARNING
* DOWNLOAD WEIGHTS OF PRE-TRAINED NEURAL NETWORKS AND USE THEM AS INITIALIZATION AND USE TRANSFER LEARNING

TRANSFER LEARNING

* EXAMPLE: CAT RECOGNIZER. YOU DOWNLOAD A NETWORK AND IT'S WEIGHTS AND WANNA CHANGE THE NETWORK, SO IT CAN RECOGNIZE YOUR CATS
  - IF YOU HAVE FEW DATA: FREEZE ALL CONVOLUTIONAL LAYERS AND ONLY CHANGE AND TRAIN THE SOFTMAX LAYER
  - IF YOU HAVE A SIGNIFICANT AMOUNT OF DATA: FREEZE SOME OF THE EARLY LAYERS, CHANGE SOFTMAX LAYER AND TRAIN LATER LAYERS AND SOFTMAX LAYER
  - IF YOU HAVE A LOT OF DATA: USE DOWNLOADED WEIGHTS AS INITIALIZATION AND TRAIN THE ENTIRE NETWORK

D-4-21

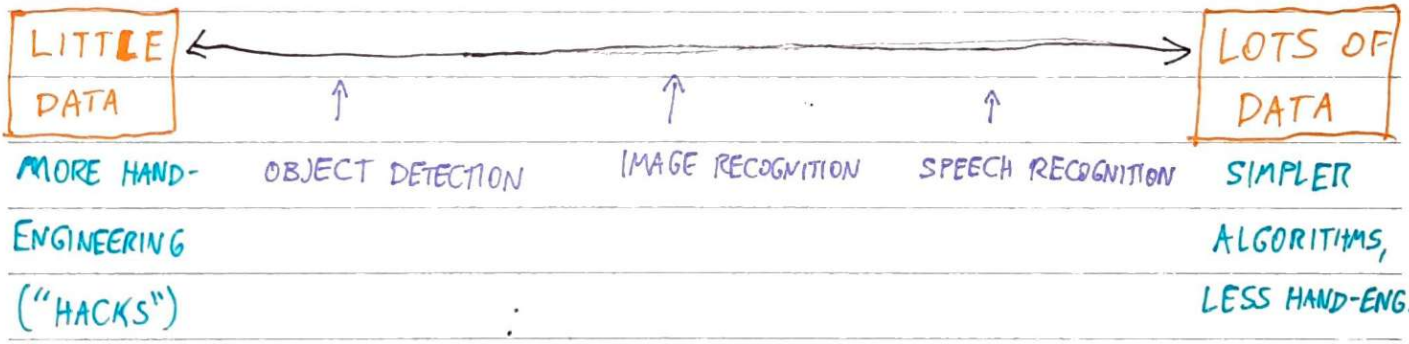
→ DATA AUGMENTATION (IMAGES)

* COMMON AUGMENTATION METHOD
  - MIRRORING, RANDOM CROPPING, ROTATION, SHEARING, LOCAL WARPING...

* COLOR SHIFTING
  - MANIPULATE R, G AND B FOR COLOR SHIFTING
  - PCA COLOR AUGMENTATION
  - ALEXNET PAPER

* IMPLEMENTING DISTORTIONS DURING TRAINING
  - USE A CPU THREAD TO LOAD DATA AND APPLY DISTORTIONS (OR A MULTIPLE-THREAD)

→ DATA VS. HAND-ENGINEERING



* TWO SOURCES OF KNOWLEDGE
  - LABELED DATA
  - HAND ENGINEERED FEATURES/NETWORK ARCHITECTURE/OTHER COMPONENTS

→ TIPS FOR DOING WELL ON BENCHMARKS/WINNING COMPETITIONS

* ENSEMBLING
  - TRAIN SEVERAL NETS INDEPENDENTLY AND AVERAGE THEIR OUTPUTS

* MULTI-CROP AT TEST TIME
  - RUN CLASSIFIER ON MULTIPLE VERSIONS OF TEST IMAGES AND AVERAGE RESULTS

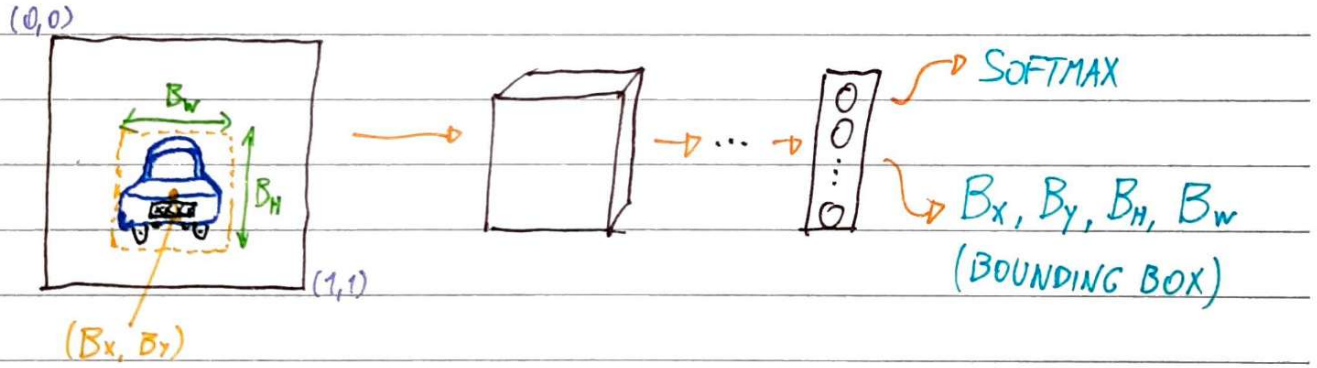
→ USE OPEN-SOURCE CODE

* USE ARCHITECTURES OF NETWORKS PUBLISHED IN THE LITERATURE
* USE OPEN-SOURCE IMPLEMENTATIONS IF POSSIBLE
* USE PRE-TRAINED MODELS AND FINE-TUNE ON YOUR DATASET

→ OBJECT LOCALIZATION

* IMAGE CLASSIFICATION
  - LABEL AN IMAGE
* CLASSIFICATION WITH LOCALIZATION
  - LABEL AND DRAW BOUNDING BOX AROUND THE OBJECT
* OBJECT DETECTION
  - MULTIPLE OBJECTS FROM DIFFERENT CATEGORIES IN THE SAME IMAGE

→ CLASSIFICATION WITH LOCALIZATION



TARGET LABEL Y:

1- PEDESTRIAN
2- CAR
3- MOTORCYCLE
4- BACKGROUND

$$Y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}$$

P_c - PROBABILITY OF OBJECT

b_x, b_y, b_h, b_w - BOUNDING BOX

C_1, C_2, C_3 - CLASSES

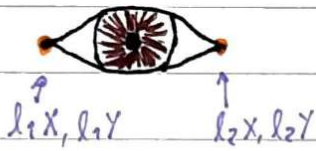
D-4-23

→ LOSS FUNCTION FOR CLASSIFICATION WITH LOCALIZATION

$$L(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + \cdots + (\hat{y}_8 - y_8)^2, & \text{IF } y_1 = 1 \rightarrow P_c \\ (\hat{y}_1 - y_1)^2, & \text{IF } y_1 = 0 \end{cases}$$

→ LANDMARK DETECTION

* INSTEAD OF RETRIEVE BOUNDING BOX, THE NETWORK RETRIEVES KEY POINTS ON THE IMAGE
  - EXAMPLE: CORNERS OF THE EYE, BODY POSE



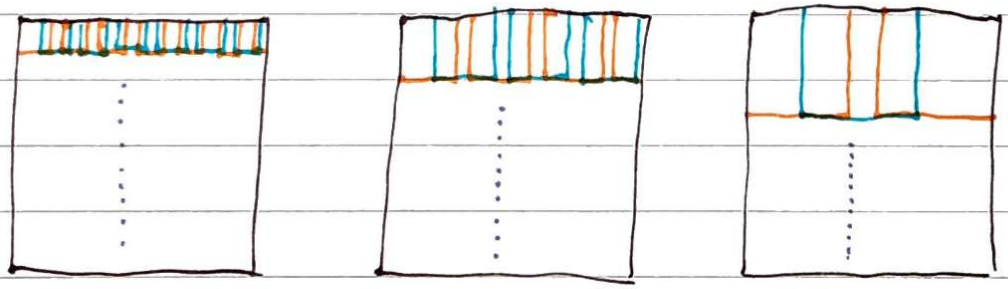
→ $[l_1X, l_1Y, \ldots \ldots \ldots l_nX, l_nY]$

→ OBJECT DETECTION (SINGLE OBJECT)

* TRAINING CONVNET
  - USE IMAGES CONTAINING A SINGLE OBJECT WITH BINARY CLASSIFICATION

~~[illegible]~~

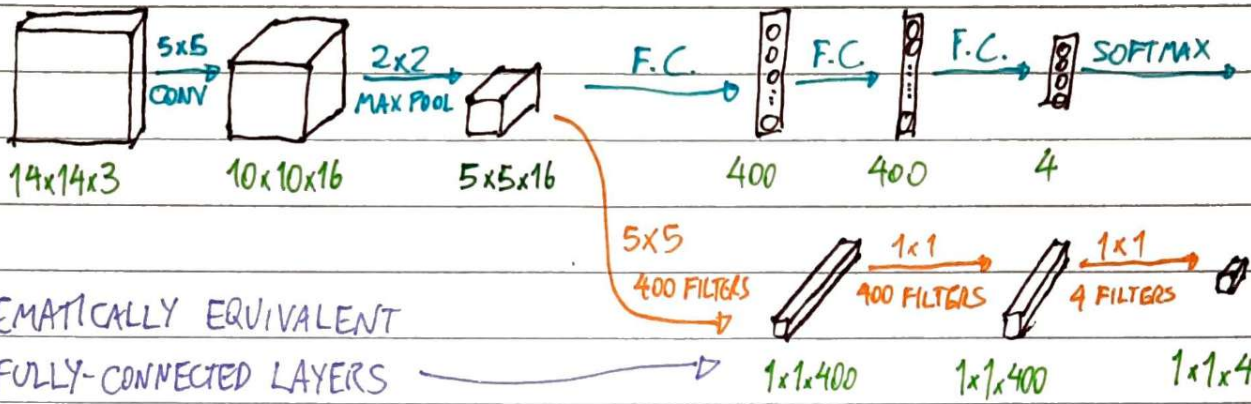
* SLIDING WINDOWS DETECTION
  - SELECT SLICES OF THE IMAGE, AND THEN PREDICT RESULTS
  - SLICES SHOULD "SLIDE" THROUGH THE IMAGE AND SCALE UP



- UNFORTUNATELY, THIS ~~REQUIRES~~ RESULTS IN A HIGH COMPUTATIONAL COST
- FORTUNATELY, THERE'S AN EFFICIENT WAY TO IMPLEMENT SLIDING WINDOWS DETECTION

→ CONVOLUTIONAL IMPLEMENTATION OF SLIDING WINDOWS

* TURNING F.C. LAYER INTO CONVOLUTIONAL LAYERS (EXAMPLE)



MATHEMATICALLY EQUIVALENT TO FULLY-CONNECTED LAYERS →

* CONVOLUTION IMPLEMENTATION OF SLIDING WINDOWS

- BASED ON OVERFEAT PAPER

NETWORK:

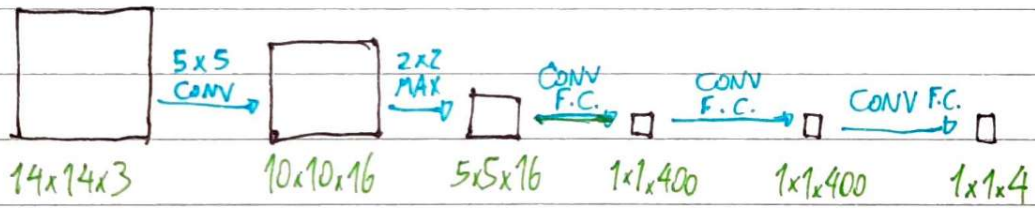
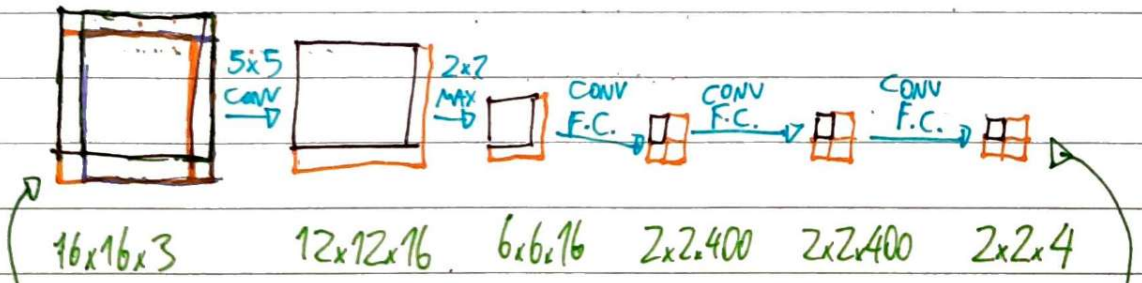


IMAGE FOR DETECTION:



4 SLIDING WINDOWS

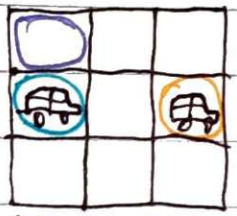
EACH POINTWISE PART OF THIS LAYER CORRESPONDS TO A SLIDING WINDOW

D-4-25

* ADVANTAGES OF CONVOLUTIONAL SLIDING WINDOWS
  - COMPUTES ALL OF THE SLIDING WINDOWS AT SAME CONVOLUTION
  - LESS COMPUTATIONAL COST
* DISADVANTAGES OF CONVOLUTIONAL SLIDING WINDOWS
  - BOUNDING BOXES ARE NOT PRECISE

→ YOLO ALGORITHM (YOU ONLY LOOK ONCE)

* EXAMPLE



100x100 RGB IMAGE

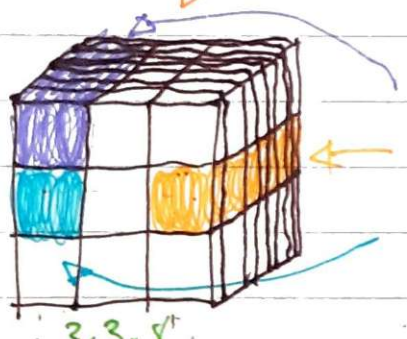
DIVIDED IN A 3x3 GRID (REAL IMPL. 19x19)

APPLY CLASSIFICATION TO EACH ONE OF THE GRID DIVISIONS

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

8-LENGHT VECTOR FOR EACH GRID DIVISION

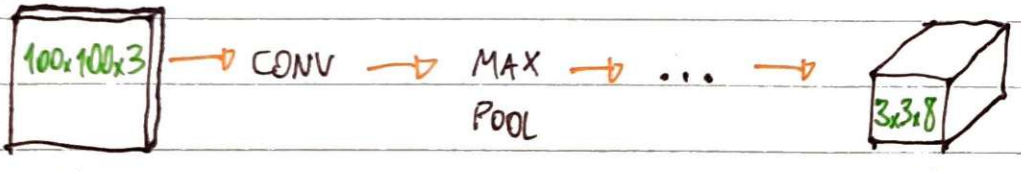
TARGET OUTPUT: 3x3x8



3x3x8

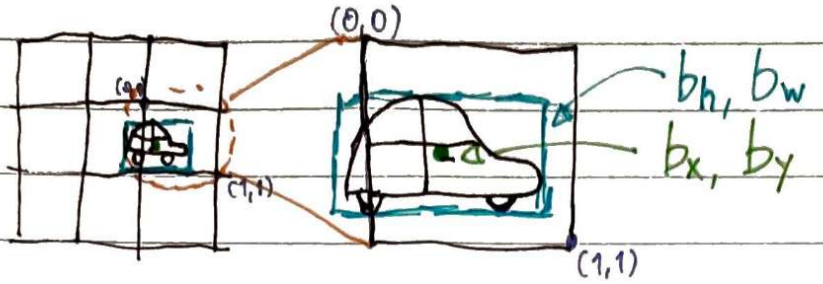
EACH PART IS A 8-LENGTH VECTOR OF A GRID DIVISION. THIS IS A CONVOLUTIONAL COMPUTATION, SO IT SHARES COMPUTATION FOR THE DIVISIONS: THE ALGORITHM RUNS FASTER

NETWORK:



X Y

→ SPECIFYING THE BOUNDING BOX

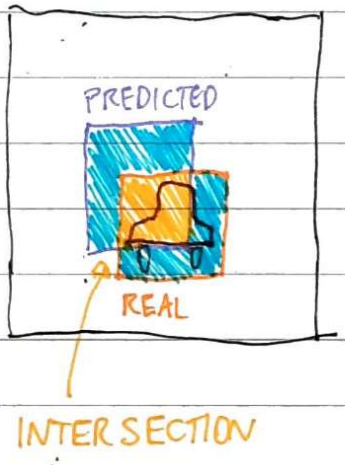


$$Y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

b_x, b_y: INSIDE THE GRID SUBDIVISION. VALUES BETWEEN 0 AND 1.

b_h, b_w: IT CAN EXTENDS BEYOND THE SUBDIVISION. CAN BE GREATER THAN 1.

→ INTERSECT OVER UNION (IOU)



$$IOU = \frac{\text{SIZE OF } \blacksquare}{\text{SIZE OF } \blacksquare + \text{SIZE OF } \blacksquare}$$

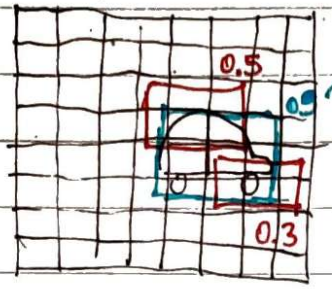
SIZE OF UNION

"CORRECT" IF $IOU \geq 0{,}5$

MORE GENERALLY, INTERSECTION OVER UNION IS A MEASURE OF THE OVERLAP BETWEEN TWO BOUNDING BOXES.

→ NON-MAX SUPPRESSION

* AVOIDS A SINGLE OBJECT FOR BEING DETECTED MORE THAN ONCE



→ CALCULATE IOU FOR EACH SUBDIVISION, CONSIDER ONLY THE HIGHEST IOU AND SUPPRESS THE OTHER DETECTIONS

* ALGORITHM

- CONSIDERING DETECTION OF ONLY ONE CLASS TO SIMPLIFY THE EXAMPLE

→ EACH OUTPUT PREDICTION IS:

P_c — PROBABILITY OF CAR DETECTION

b_x

b_y

b_h

b_w

→ DISCARD ALL BOXES WITH $P_c \leq 0.6$

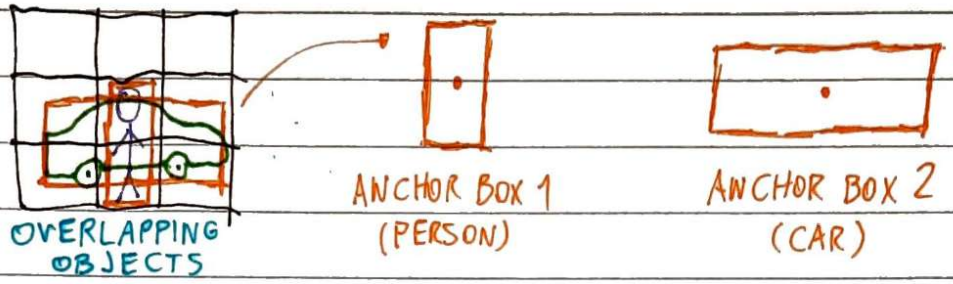
→ WHILE THERE ARE ANY REMAINING BOXES:

→ 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

* DETECT MULTIPLE OBJECTS ON THE SAME GRID SUBDIVISION



$$y = \begin{bmatrix} p_c \\ \vdots \\ c_3 \\ p_c \\ \vdots \\ c_3 \end{bmatrix} \begin{matrix} \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} p_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3 \text{ FOR ANCHOR BOX 1} \\ \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} p_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3 \text{ FOR ANCHOR BOX 2} \end{matrix}$$

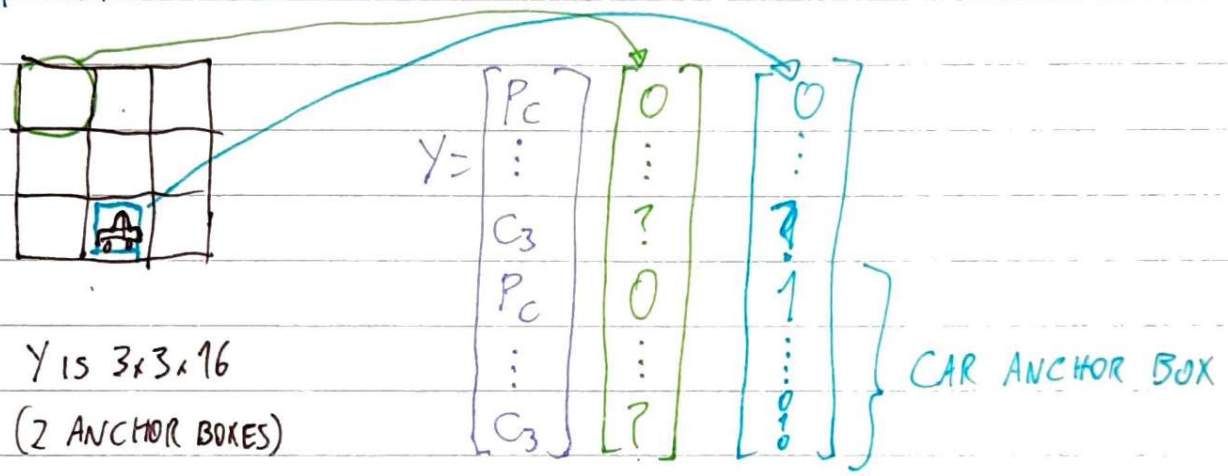
* ALGORITHM

PREVIOUSLY:	WITH TWO ANCHOR BOXES:
EACH OBJECT IN TRAINING IMAGE IS ASSIGNED TO GRID CELL THAT CONTAINS THAT OBJECT'S MIDPOINT. (GRID CELL)	EACH OBJECT IN TRAINING IMAGE IS ASSIGNED TO GRID CELL THAT CONTAINS OBJECT'S MIDPOINT AND ANCHOR BOX FOR THE GRID CELL WITH HIGHEST IOU. (GRID CELL, ANCHOR BOX)
OUTPUT Y: $3 \times 3 \times 8$	OUTPUT Y: $3 \times 3 \times 16$ (FOR n ANCHOR BOXES: $3 \times 3 \times (n \times 8)$)

* CASES WHEN ANCHOR BOXES MAY NOT WORK WELL
  - TWO OBJECTS WITH SIMILAR BOUNDING BOXES AT SAME GRID CELL
  - # OF ANCHOR BOXES < # OF OBJECTS IN GRID CELL
  - THESE CASES ~~DOESN'T~~ DON'T OCCUR OFTEN WITH A LARGE GRID, LIKE 19x19

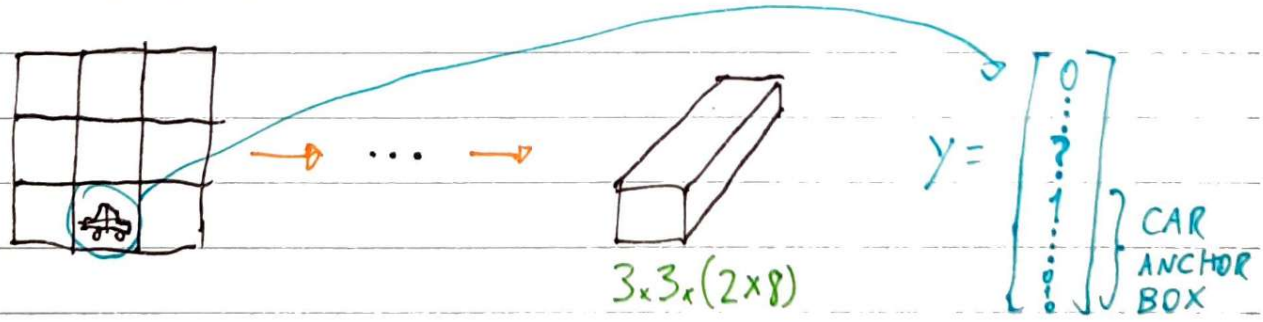
→ YOLO ALGORITHM ON PRACTICE

* TRAINING SET:



Y is 3x3x16
(2 ANCHOR BOXES)

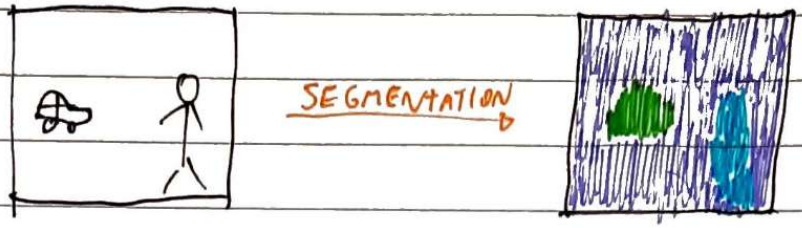
* PREDICTIONS:



* OUTPUTTING THE NON-MAX SUPPRESSED OUTPUTS:
  - FOR EACH GRID CELL, GET THE 2 PREDICTED BOUNDING BOXES
  - GET RID OF LOW PROBABILITY PREDICTIONS
  - FOR EACH CLASS (E.G. PEDESTRIAN, CAR, MOTORCYCLE) USE NON-MAX SUPPRESSION TO GENERATE FINAL PREDICTIONS

REGION PROPOSAL: R-CNN

* RUNS SEGMENTATION ALGORITHM TO DETECT WHETHER REGIONS TO RUN ONLY A FEW WINDOWS RATHER THAN SLIDING WINDOWS ON THE ENTIRE IMAGE



* CON: R-CNN IS A LOT SLOWER

* ENHANCEMENTS

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 NEURAL NETWORK TO PROPOSE REGIONS.

* FASTER R-CNN IS STILL A LITTLE BIT SLOWER THAN YOLO ALGORITHM

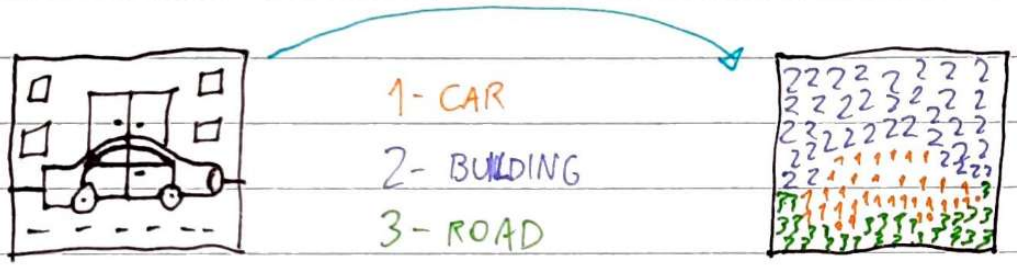
OBJECT DETECTION VS. SEMANTIC SEGMENTATION

* OBJECT DETECTION: DRAW BOUNDING BOXES AROUND THE DETECTED OBJECT
* SEMANTIC SEGMENTATION: SEGMENT OBJECTS AT PIXEL LEVEL, DISCOVERING WHAT EXACT PIXELS BELONGS TO AN OBJECT

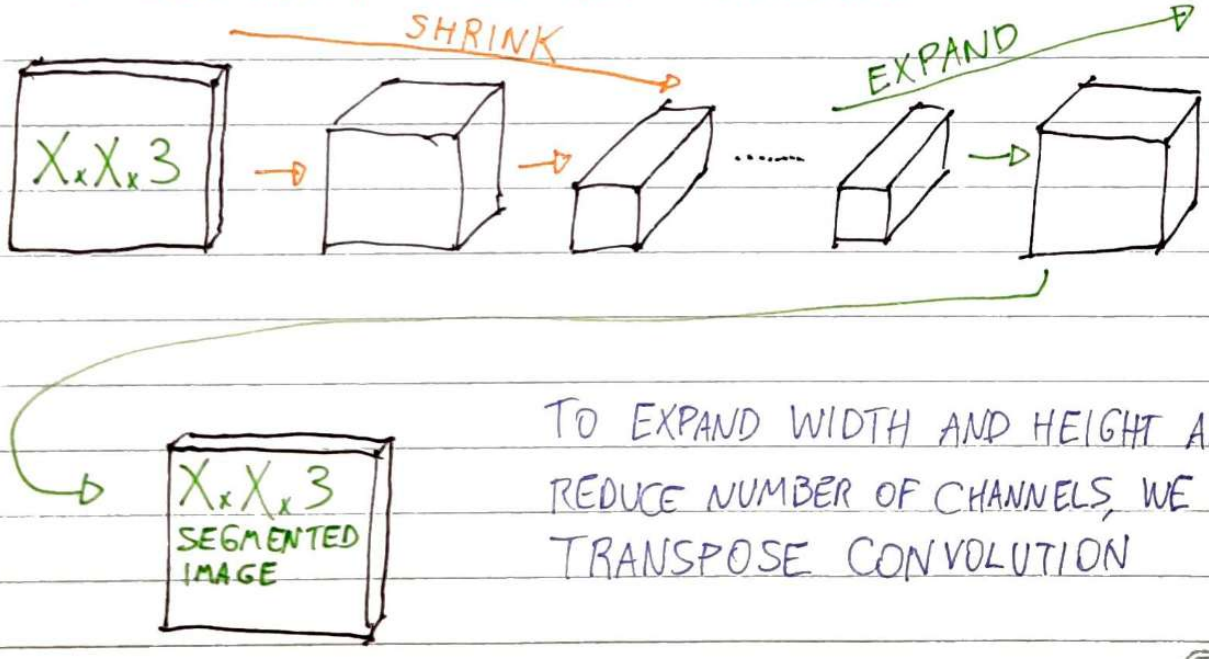
WHY SEMANTIC SEGMENTATION?

* IMPROVINGS ON DISEASE DETECTION BY X-RAY SEGMENTATION AND ANALYSIS, FOR EXAMPLE
* SURGICAL PLANNING

PER-PIXEL CLASS LABELS

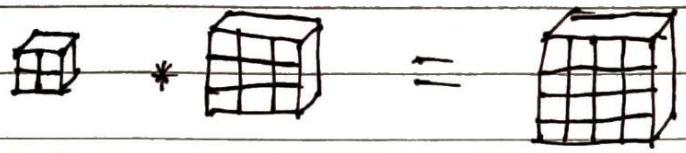


DEEP LEARNING FOR SEMANTIC SEGMENTATION



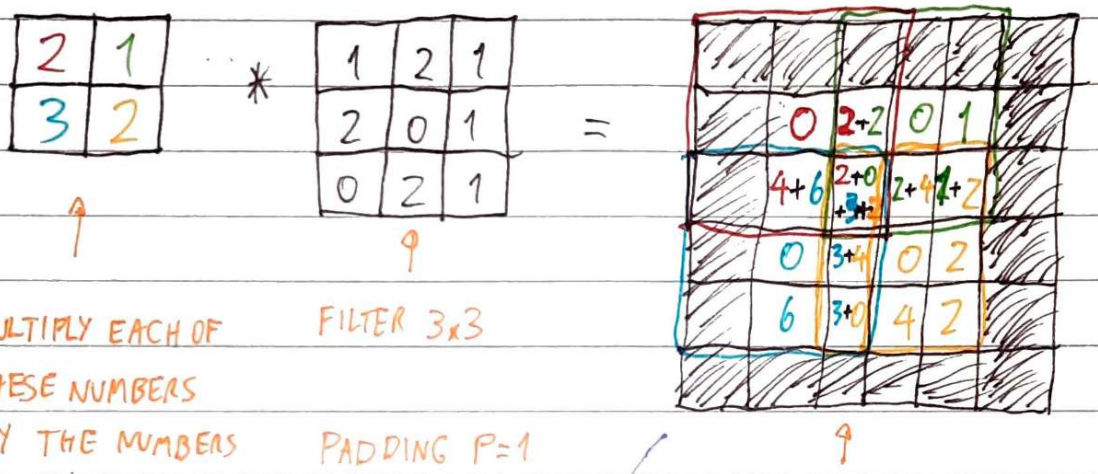
TO EXPAND WIDTH AND HEIGHT AND REDUCE NUMBER OF CHANNELS, WE USE TRANSPOSE CONVOLUTION

→ TRANSPOSE CONVOLUTION



2x2 * 3x3 = 4x4

* EXAMPLE



MULTIPLY EACH OF THESE NUMBERS BY THE NUMBERS OF THE FILTER.

FILTER 3x3

PADDING P=1

STRIDE S=2

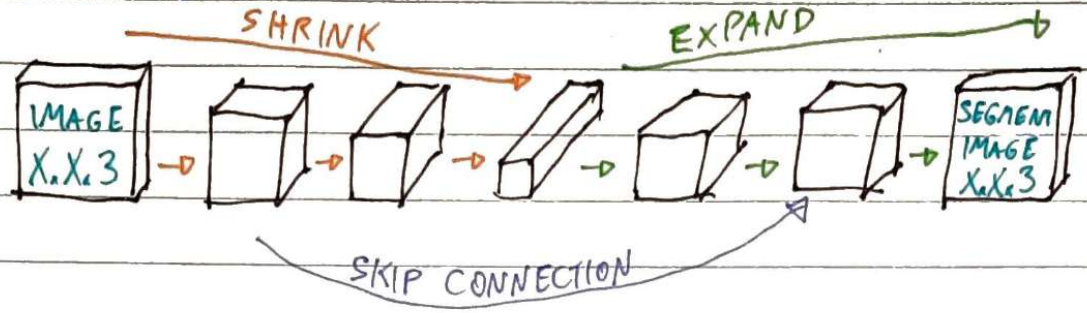
IGNORE PADDING AREA

0	4	0	1
10	7	6	3
0	7	0	2
6	3	4	2

RESULT

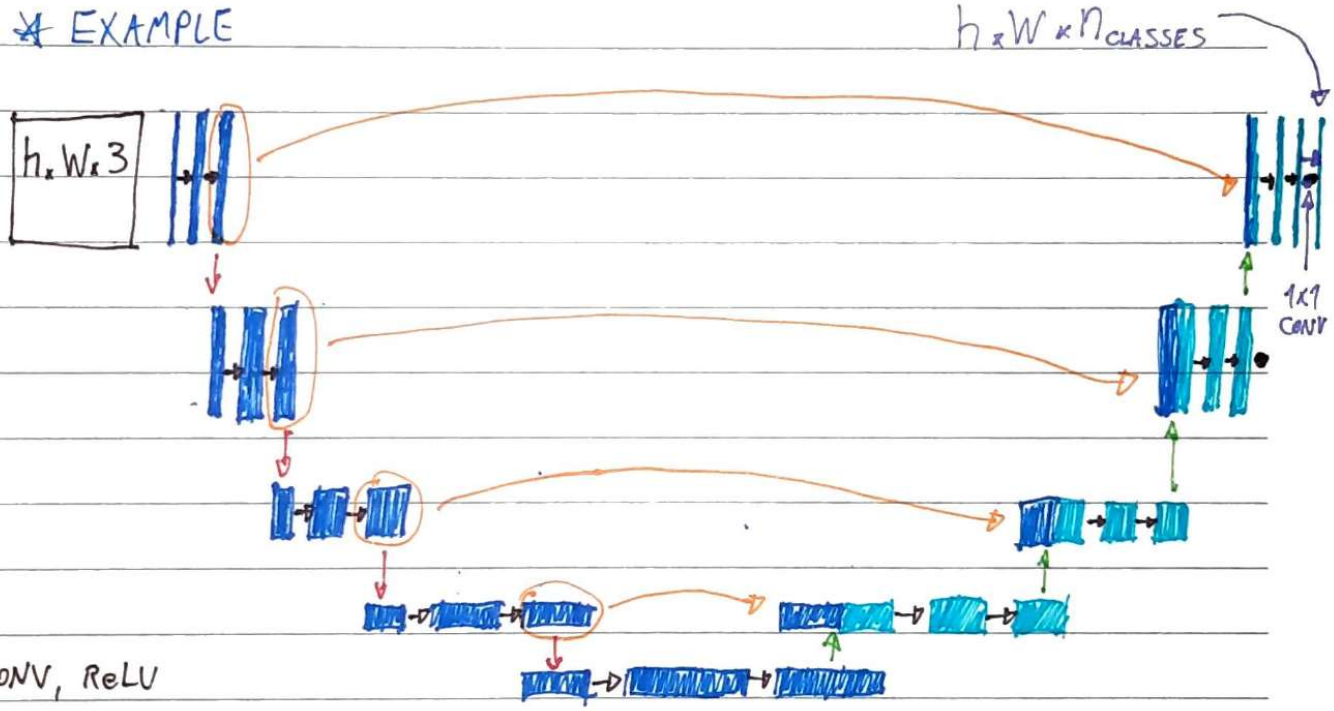
U-NET ARCHITECTURE

* INTUITION



- HIGH LEVEL LATER LAYER INFORMATION IS VALUABLE, BUT NOT PIXEL-LEVEL PRECISE
- LOW LEVEL EARLY LAYER INFORMATION CAN HELP IMPROVE PRECISION

* EXAMPLE



→: CONV, ReLU

↓: MAX POOLING

↑: TRANSPOSE CONV.

⇢: SKIP CONNECTION

→: CONV (1x1)

FACE VERIFICATION VS. FACE RECOGNITION

* VERIFICATION (1:1 PROBLEM)
  - INPUT IMAGE, NAME/ID
  - OUTPUT WHETHER THE INPUT IMAGE IS THAT OF THE CLAIMED PERSON
* RECOGNITION (1:K PROBLEM)
  - HAS A DATABASE OF K PERSONS
  - GET AN INPUT IMAGE
  - OUTPUT ID OF THE IMAGE IF IT'S ANY OF THE K PERSONS (OR "NOT RECOGNIZED")

ONE-SHOT LEARNING

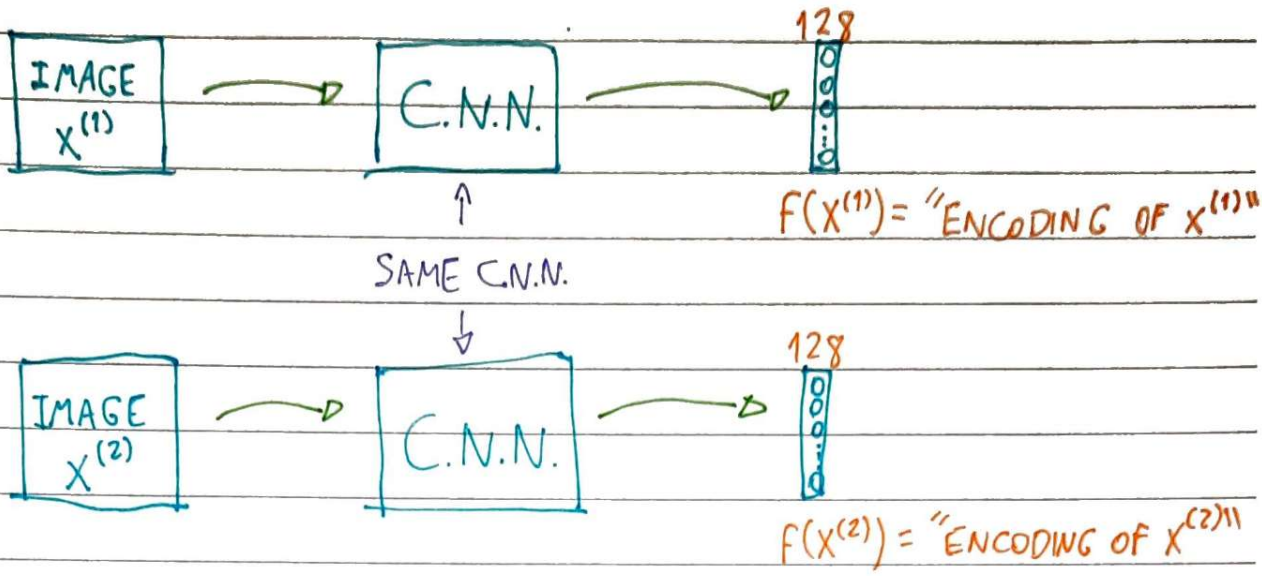
* LEARNING FROM ONE EXAMPLE TO RECOGNIZE THE PERSON AGAIN
* COMMON CNN TRAINING DOESN'T WORK WELL: SMALL TRAINING SET, NEEDS RETRAINING/RESHAPING IF A NEW PERSON IS REGISTERED
* APPROACH: LEARNING A "SIMILARITY" FUNCTION

$d(img1, img2)$ = DEGREE OF DIFFERENCE BETWEEN IMAGES

IF $d(img1, img2) \leq \tau$ → "SAME"
IF $d(img1, img2) > \tau$ → "DIFFERENT"
} → VERIFICATION

τ ← THRESHOLD HYPERPARAMETER

→ SIAMESE NETWORK



* KEY IDEA: RUN IDENTICAL C.N.N ON TWO DIFFERENT INPUTS AND COMPARE THEM

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

* PARAMETERS OF NN DEFINE AN ENCODING $f(X^{(i)})$

* LEARN PARAMETERS SO THAT:
  - IF $X^{(i)}$, $X^{(j)}$ ARE THE SAME PERSON, $d(X^{(i)}, X^{(j)})$ IS SMALL
  - IF $X^{(i)}$, $X^{(j)}$ ARE DIFFERENT PERSONS, $d(X^{(i)}, X^{(j)})$ IS LARGE

→ TRIPLET LOSS

* LEARNING OBJECTIVE

- ANCHOR IMAGE: REFERENCE IMAGE WITH A FACE (A)
- POSITIVE IMAGE: IMAGE WITH A FACE OF THE SAME PERSON IN THE ANCHOR IMAGE (P)
- NEGATIVE IMAGE: IMAGE WITH A FACE OF A DIFFERENT PERSON (N)

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

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

α: MARGIN

* LOSS FUNCTION

- GIVEN 3 IMAGES A, P, N:

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

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

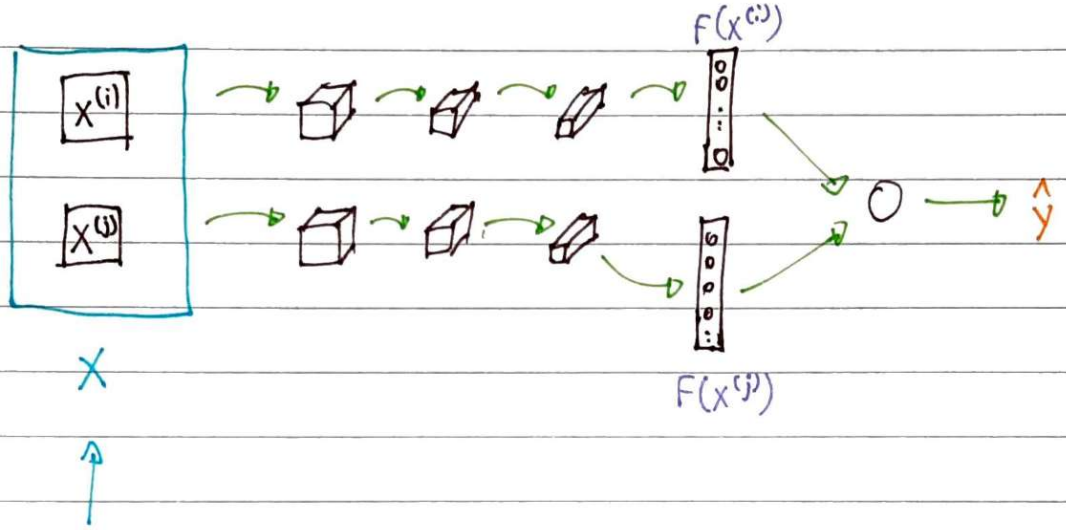
* WHY IS THE MARGIN (α) IMPORTANT?

- AVOID OBVIOUS SOLUTION ($f(A) - f(P) = 0$, $f(A) - f(N) = 0$)
- PUSH $d(A, P)$ AND $d(A, N)$ DISTANT TO EACH OTHER

* 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, SO TRAINING ISN'T SO EFFECTIVE
- CHOOSE TRIPLETS THAT'RE "HARD" TO TRAIN ON
  $d(A,P) \approx d(A,N)$ IS "HARD"

→ FACE VERIFICATION AND BINARY CLASSIFICATION



$X^{(i)}$: NEW IMAGE / $X^{(j)}$: DATABASE IMAGE

* OUTPUT NEURON: ACTIVATION FUNCTION (LIKE SIGMOID)

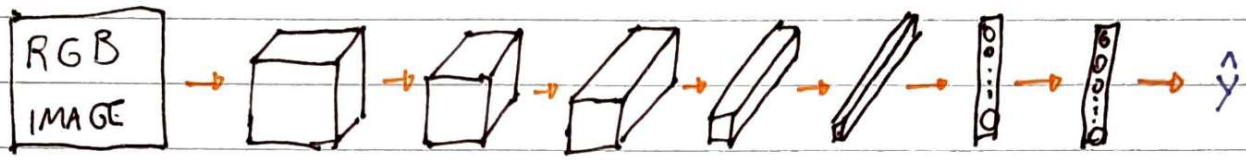
$$\hat{y} = \sigma\left(\sum_{k=1}^{127} w_k \left| F(x^{(i)})_k - F(x^{(j)})_k \right| + b\right)$$

* FOR DATABASE IMAGES: $f(X^{(j)})$ CAN BE PRE-COMPUTED

NEURAL STYLE TRANSFER

* A CONTENT IMAGE IS STYLIZED BASED ON A STYLE IMAGE, CREATING A GENERATED IMAGE

WHAT ARE DEEP CONVNETS REALLY LEARNING?



IF WE LOOK FOR IMAGE PATCHES THAT MAXIMIZE UNITS ACTIVATION IN THE LAYERS:

LAYER 1: EDGES AND CORNERS
LAYER 2: A LITTLE MORE COMPLEX SHAPES
LAYER 3: MORE COMPLEX SHAPES, PARTS OF OBJECTS
LAYER 4: EVEN MORE COMPLEX: OBJECTS, ANIMALS, ETC.
LAYER 5: EVEN MORE SOPHISTICATED SHAPES

NEURAL STYLE TRANSFER COST FUNCTION

C - CONTENT IMAGE / S - STYLE IMAGE / G - GENERATED IMAGE

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

FINDING THE GENERATED IMAGE G

1 - INITIALIZE G RANDOMLY (G: $100 \times 100 \times 3$) RGB

2 - USE GRADIENT DESCENT TO MINIMIZE J(G)

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

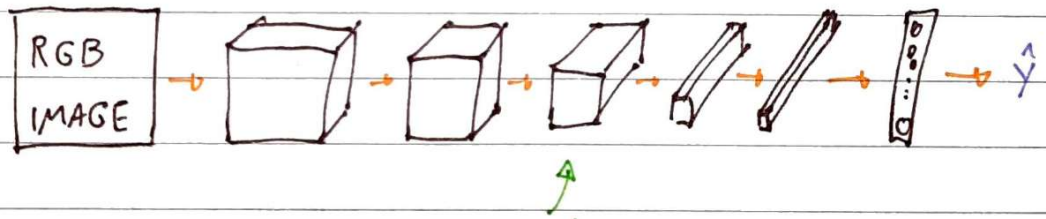
→ CONTENT COST FUNCTION

* SAY YOU USE HIDDEN LAYER l TO COMPUTE CONTENT COST
  - l IS NEITHER TOO SHALLOW NOR TOO DEEP LAYER
* USE PRE-TRAINED CONVNET (E.G., VGG NETWORK)
* LET $a^{[l](C)}$ AND $a^{[l](G)}$ BE THE ACTIVATION OF LAYER l ON THE IMAGES
* IF $a^{[l](C)}$ AND $a^{[l](G)}$ ARE SIMILAR, BOTH IMAGES HAVE SIMILAR CONTENT

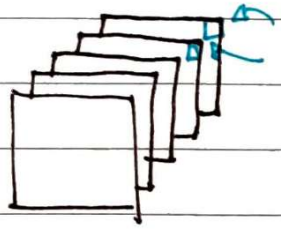
$$J_{CONTENT}(C, G) = \frac{1}{2} \left\| a^{[l](C)} - a^{[l](G)} \right\|^2$$

ELEMENT-WISE SUM OF SQUARE DIFFERENCES OF ELEMENTS IN LAYER l

→ MEANING OF THE "STYLE" OF AN IMAGE



SAY YOU ARE USING LAYER l'S ACTIVATIONS TO MEASURE "STYLE".
DEFINE STYLE AS CORRELATION BETWEEN ACTIVATIONS ACROSS CHANNELS.



HOW CORRELATED ARE THE ACTIVATIONS ACROSS DIFFERENT CHANNELS?

→ INTUITION ABOUT STYLE OF AN IMAGE

* CORRELATION SHOWS WHICH TEXTURES OCCURS TOGETHER TO SOME SPECIFIC COLORS, OR HOW OFTEN THEY OCCUR, FOR EXAMPLE

→ STYLE MATRIX

* LET $a^{[l]}_{i,j,K}$ = ACTIVATION AT (i, j, K) (H, W, C)

* $G^{[l]} = n_c^{[l]} \times n_c^{[l]}$

* $G^{[l]}_{KK'}$ → CORRELATIONS OF CHANNELS K AND K'

$$G^{[l]}_{KK'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l]}_{ijK} a^{[l]}_{ijK'}$$

(UNNORMALIZED CROSS-COVARIANCE)

* COMPUTE $G^{[l](S)}_{KK'}$ AND $G^{[l](G)}_{KK'}$

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

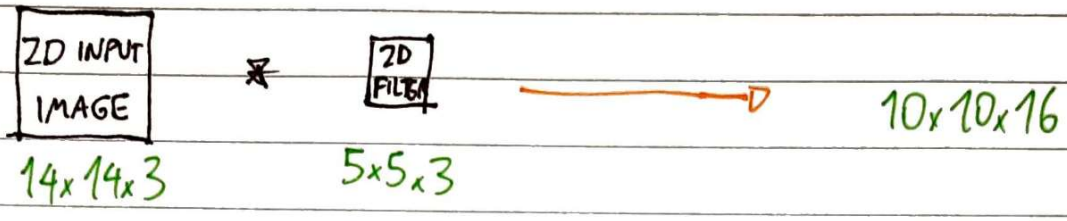
$$J^{[l]}_{STYLE}(S, G) = \frac{1}{\left(2 n_H^{[l]} n_W^{[l]} n_c^{[l]}\right)^2} \sum_K \sum_{K'} \left(G^{[l](S)}_{KK'} - G^{[l](G)}_{KK'}\right)^2$$

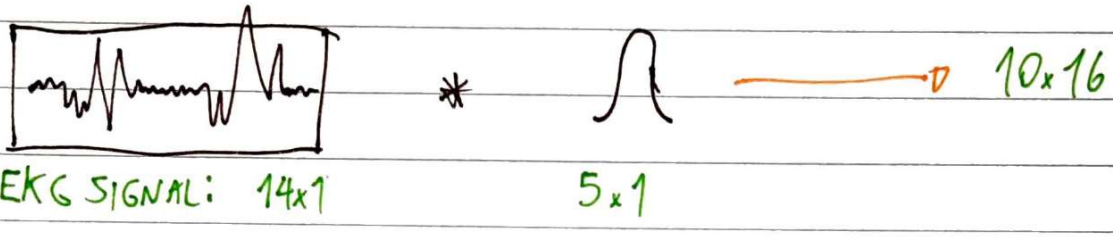
SUM $J^{[l]}$ OF DIFFERENT LAYERS

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

HYPERPARAMETER

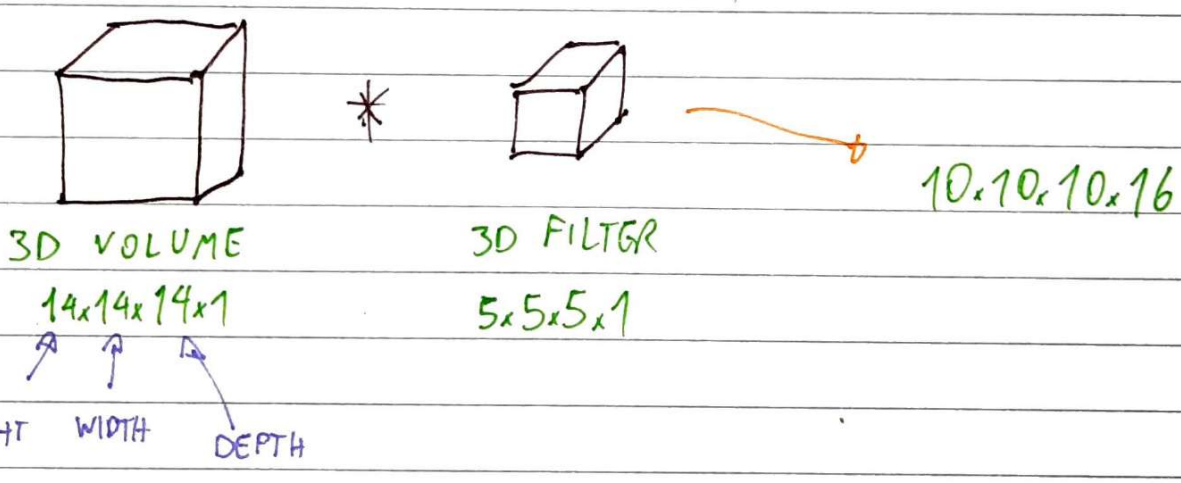
CONVOLUTIONS IN 2D AND 1D





3D DATA

* EXAMPLE: CT SCAN (3D XRAY SCAN OF YOUR BODY)



* OTHER EXAMPLE: MOVIE DATA