Course 4

* Kernel for vertical edge : $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$

Kernel for horizontal edge: $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Sobel filter: $\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$

Schass filter: $\begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix}$

Eg: 6×6 image $*$ 3×3 filter $=$ 4×4 image

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

Image shrinks on applying convolution filters.
Also, edge pixels are not used as much as the non-edge pixels

To counter this, 1 pixel padding is applied to the original image

This increase weight or usage of edge pixels and retains the same size output as original image.

Valid convolution : $n \times n * f \times f = n-f+1 \times n-f+1$

Same convolution : $n+2p \times n+2p * f \times f = n \times n$

output size / input size

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

$\therefore p = \frac{f-1}{2}$ $\quad p \rightarrow$ padding

filters are generally odd size square matrix.

* Strided Convolution (s): jumps 's' pixels when the filter is moved

Output size : $\left[\frac{n+2p-f}{s}+1\right] \times \left[\frac{n+2p-f}{s}+1\right]$

$\lfloor x \rfloor$ = floor function

★ Convolution over Volume.

$6 \times 6 \times 3 \;*\; 3 \times 3 \times 3 = 4 \times 4 \times 1$

$6 \times 6 \times 3$: h, w, channels — $3 \times 3 \times 3$: h, w, channels — $4 \times 4 \times 1$: 2D image

★ Notation

$f^{[l]}$ → filter layer

$p^{[l]}$ → 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: $n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

$$n_H^{[l]} = \left[\frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right]$$

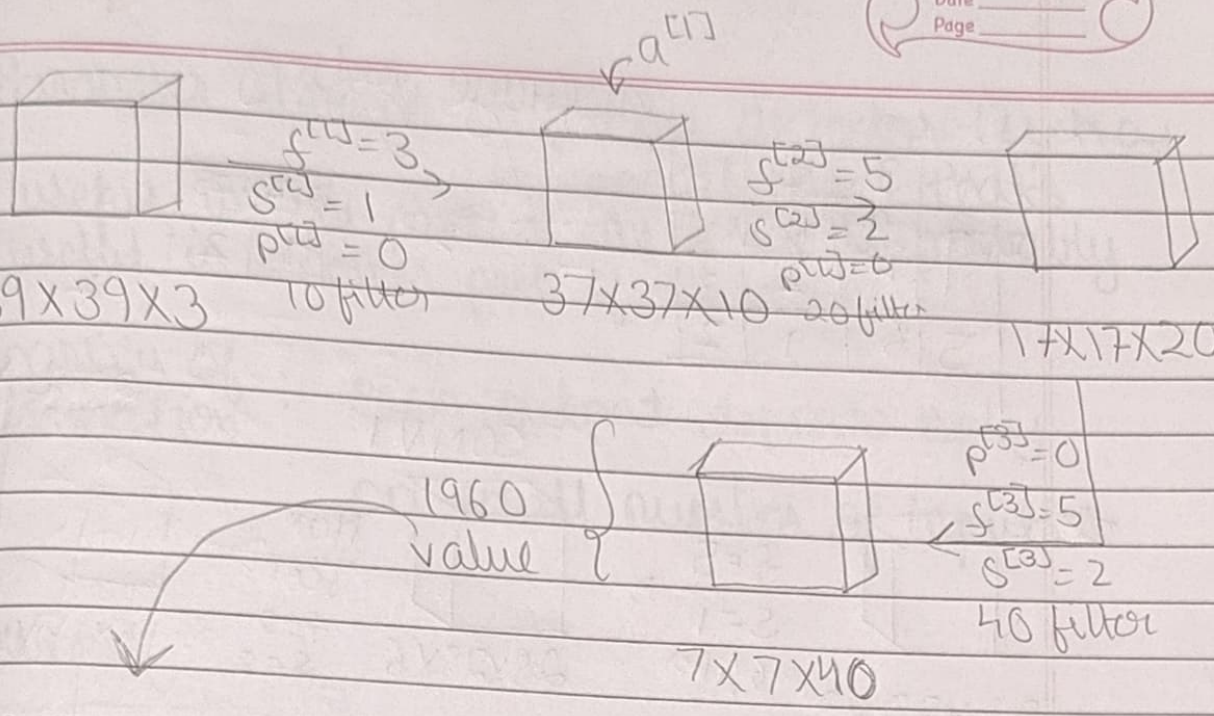
$$n_w^{[l]} = \left[\frac{n_w^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right]$$

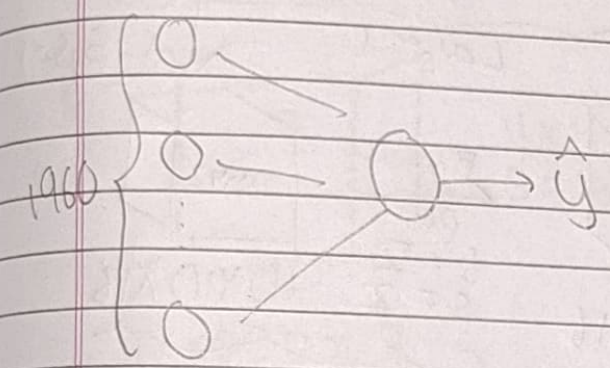
each filter → $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

$a^{[l]} = n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

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

bias: $n_c^{[l]}$





generally,

$n_H, n_W \downarrow$ } as we go deeper in the CNN
$n_C \uparrow$ }

★ Pooling Layer: Max pooling

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

4×4

→

9	2
6	3

2×2

$f^{[1]}=2$
$s^{[1]}=2$

↓

hyperparameters fixed

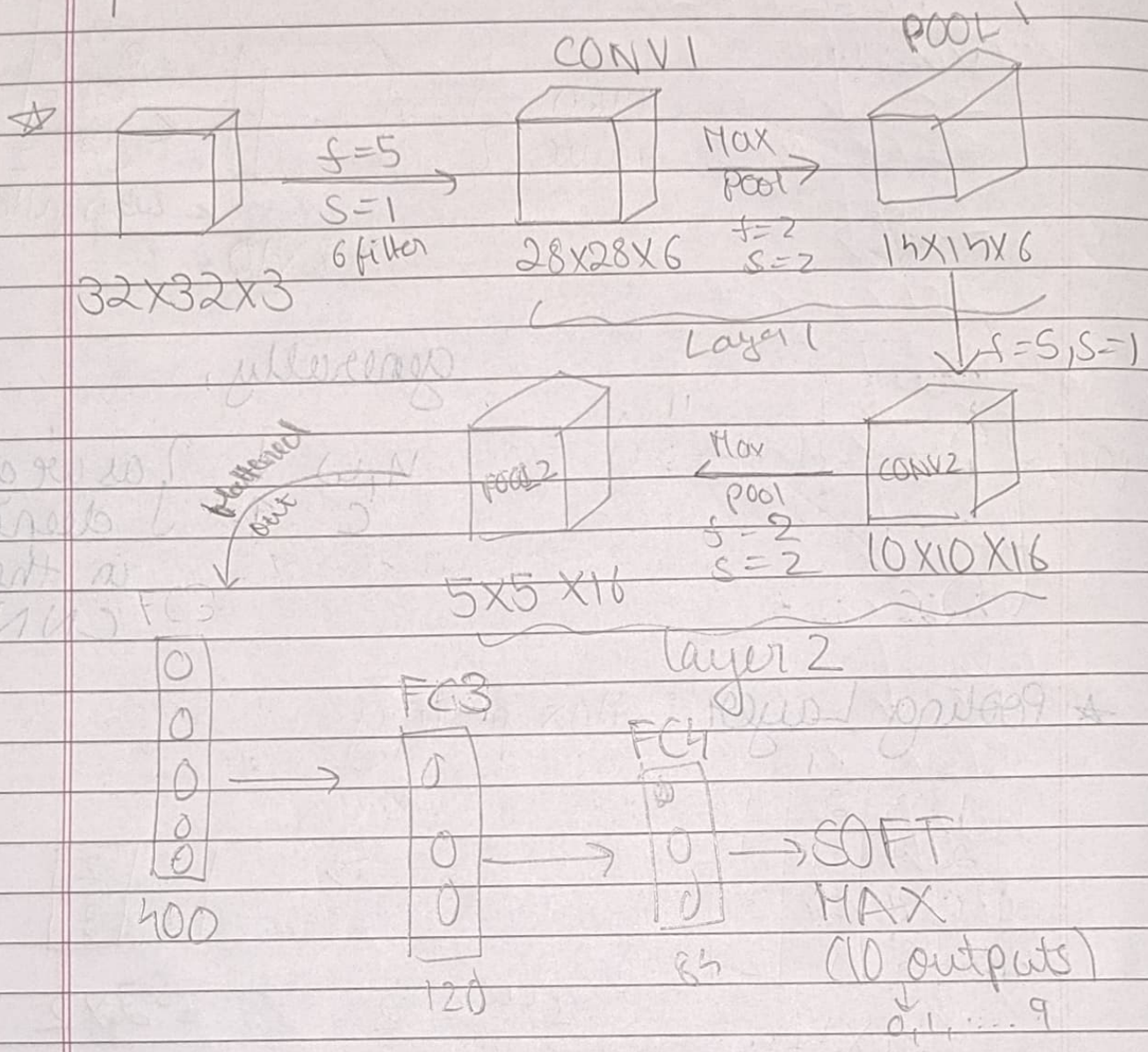
no learning

average pooling

1	3	2	1
2	9	1	1
1	4	2	3
5	6	1	2

→

3.75	1.25
4	2

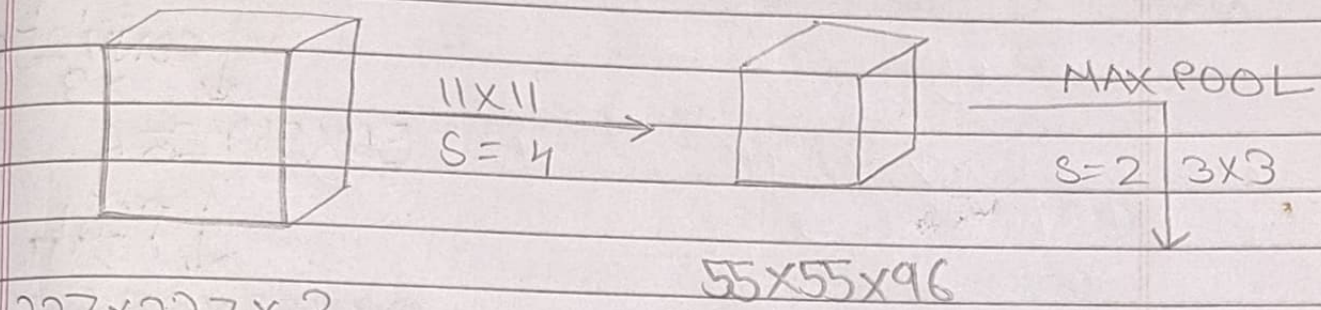


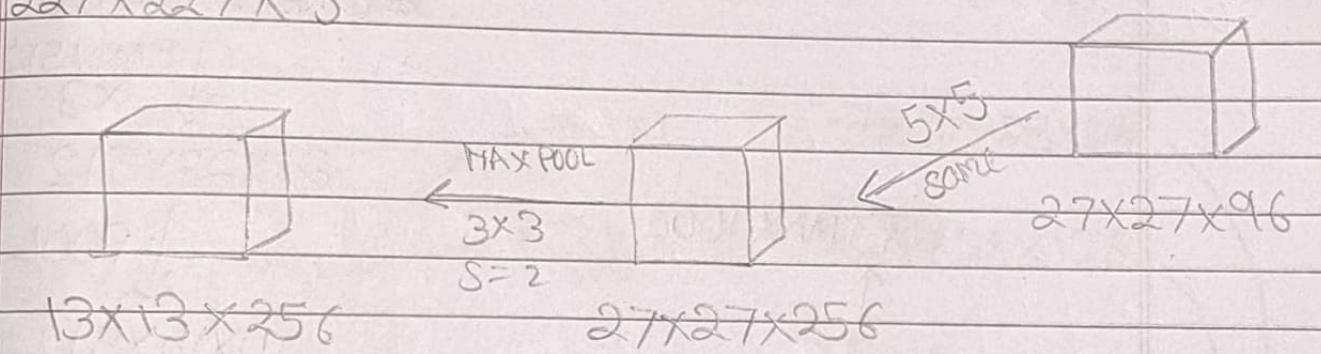
$n_H, n_W \downarrow$ $n_C \uparrow$

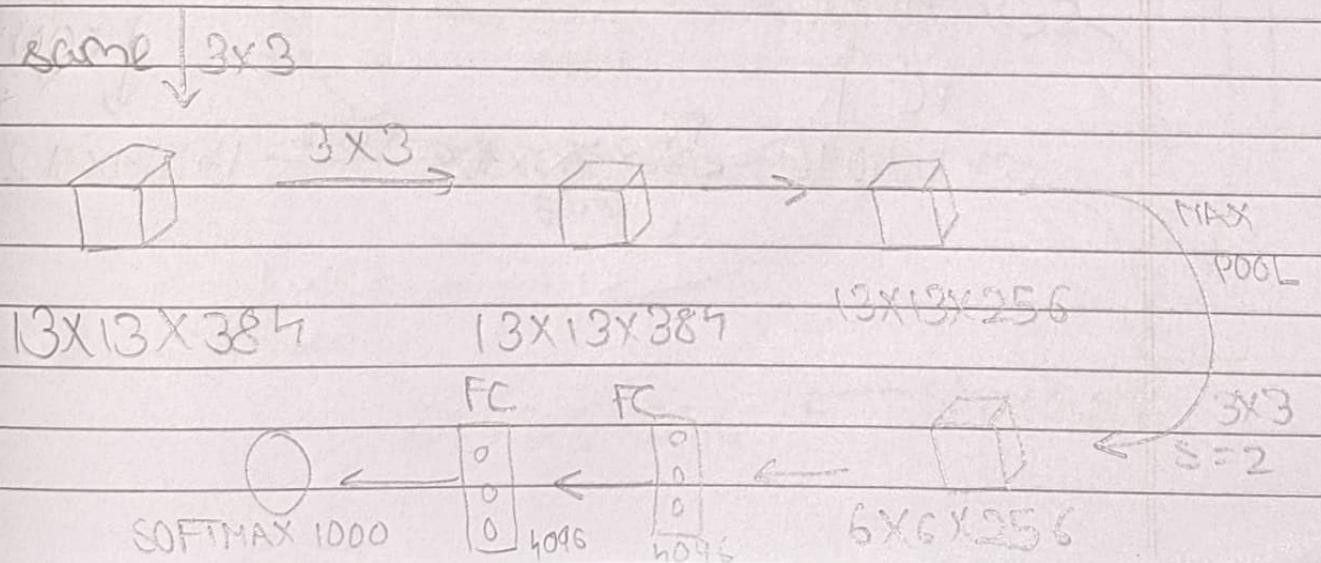
Parameter sharing: A feature detector (such as a vertical edge) that's useful in one part of the image is probably useful in another part of the image.

Sparsity of connection: each output depends only on small number of inputs

* Alexnet







VGG-16:

CONV = 3×3, S=1, same
POOL = 2×2, S=2

224×224×3 —CONV 64 ×2→ 224×224×64

↓ POOL

112×112×64

↓ CONV 128 ×2

112×112×128

↓ POOL

56×56×128

↓ CONV 256 ×3

56×56×256

↓ POOL

28×28×256

↓ CONV 512 ×3

28×28×512

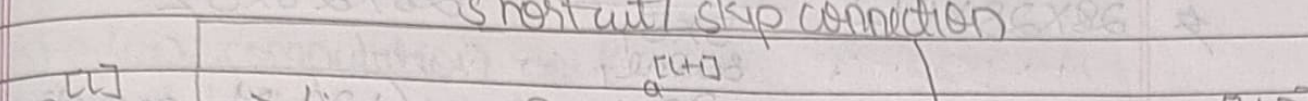
↓ POOL

14×14×512

↓ CONV 512 ×3

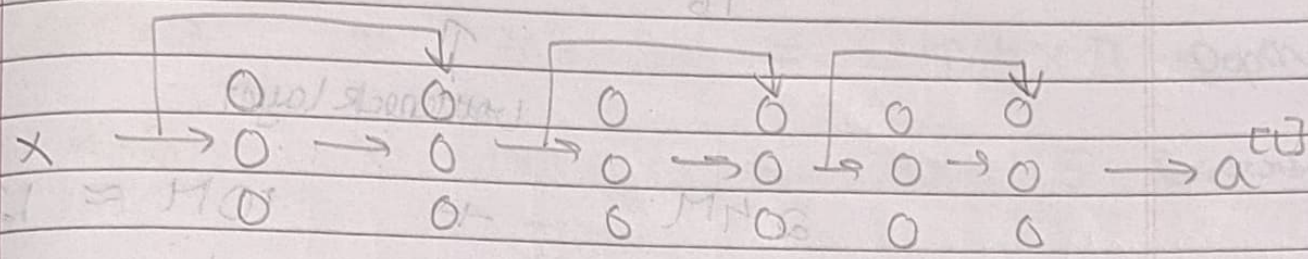
14×14×512 —POOL→ 7×7×512 —FC→ 4096 —FC→ 4096 —FC→ SOFTMAX 1000

★ Residual Block



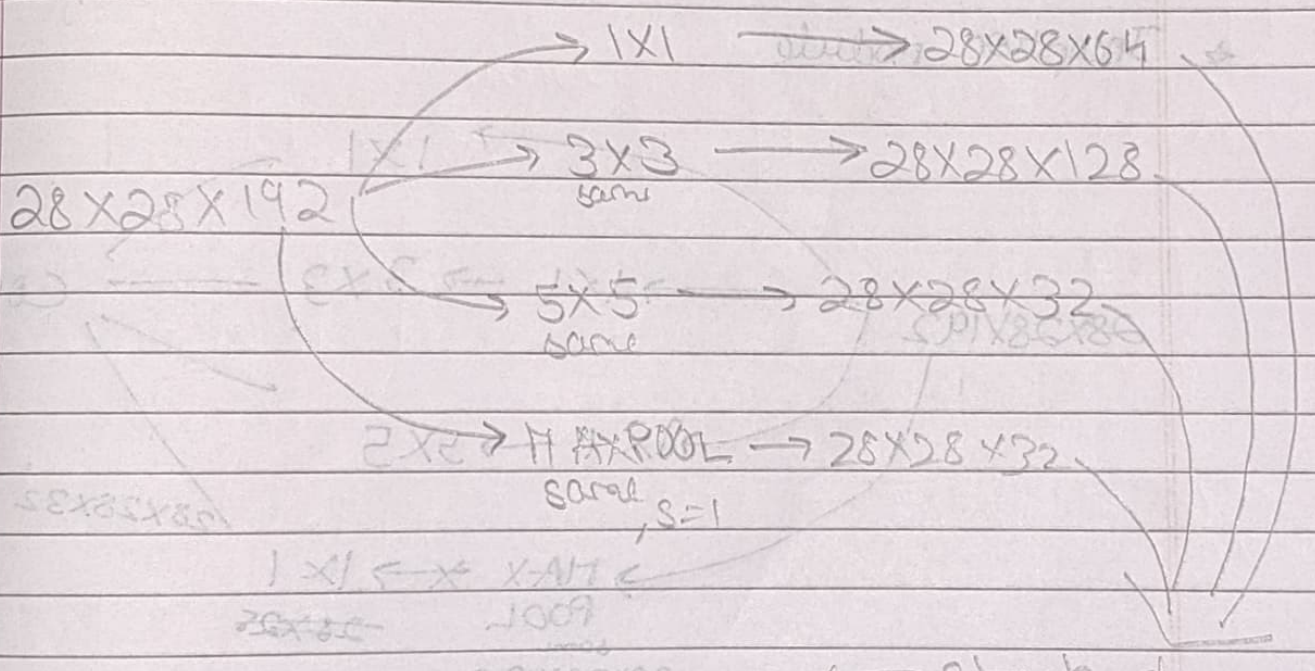
$$\therefore a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

★ Resnet



ResNet Example

★ Inception network



★ $28 \times 28 \times 192 \xrightarrow[5\times5,\ \text{same},\ 32]{\text{CONV}} 28 \times 28 \times 32$

120 M operations needed

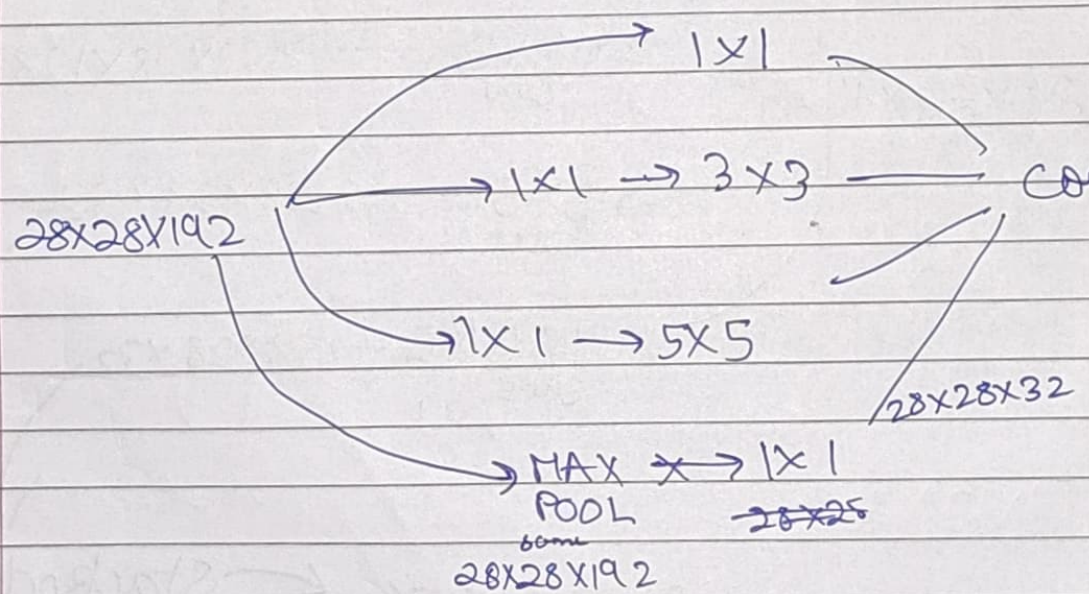
★ $28 \times 28 \times 192 \xrightarrow[1\times1,\ 16]{\text{CONV}} 28 \times 28 \times 16 \xrightarrow[5\times5,\ 32]{\text{CONV}} 28 \times 28 \times 32$

↑ bottleneck layer

2.4M + 10M ≈ 12.4M

Output is of same dimension but the computation is reduced by a factor of 10

★ Inception module



* Mobile Nets

↳ low computation cost
useful for mobile & embedded vision applications

Depthwise Separable Convolution.

St°. $6 \times 6 \times 3 \quad * \quad 3 \times 3 \times 3 = 4 \times 4 \times 3$ Depthwise

St°. $4 \times 4 \times 3 \quad * \quad 1 \times 1 \times 3 = 4 \times 4 \times 5$ Pointwise

5 filters

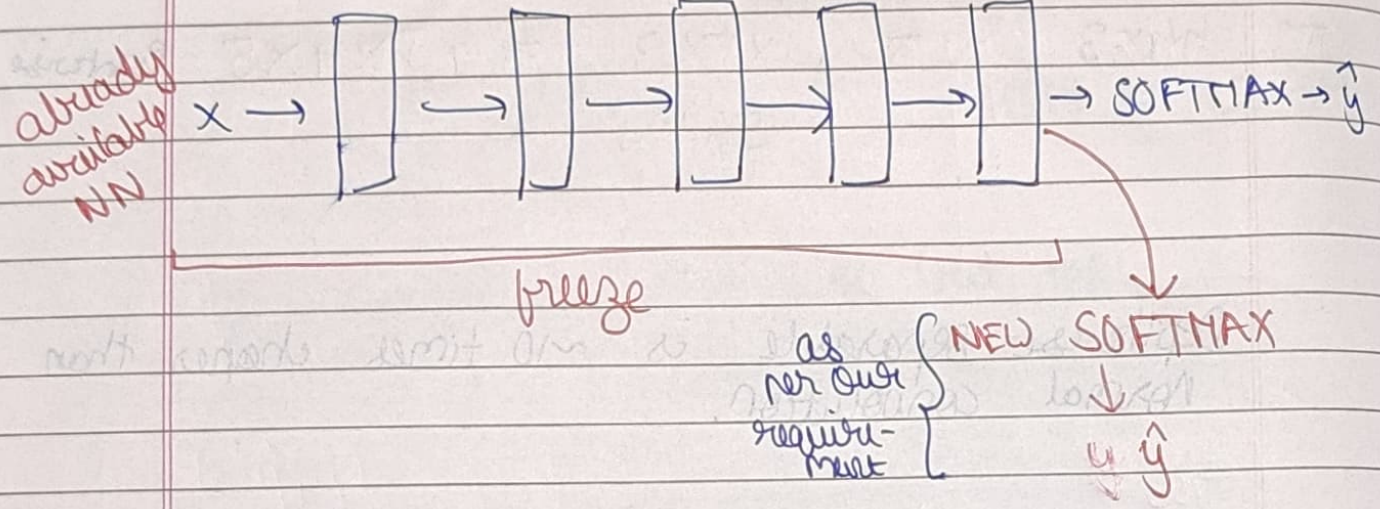
Depthwise separable is ~10 times cheaper than normal convolution.

★ Efficient NN

↳ best NN with limited computational power

Can be done using → r resolution of image
→ d depth of NN
→ w width of NN (No of neurons)

★ Transfer Learning



More the data → less layer freeze and train on the remaining layers

Output of the frozen layers can be computed and saved to disk to enhance the speed of training.

If lots of data then use the weight only as initialization.

PCA Alexnet Paper

classmate
Date
Page

Data Augmentation

- Mirroring image } popular
- Random cropping }
- rotation } not used as much
- shearing }
- local warping }

→ Color shifting → can RGB value of all pixel

ensembling → average outputs of multiple NN

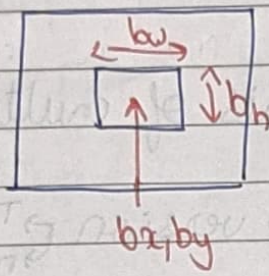
Multicrop → 1 image → 10 version → TL, TR, BL, BR
↳ n → TL, TR, BL, BR

* Object Detection

Classification with localisation

Convnet → SOFTMAX → $\hat{y}$

Convnet → b_x, b_y, b_w, b_h → % of image length



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

P_c → Obj hai ki nhi ?

b_x, b_y, b_w, b_h → bounding box

C_1, C_2, C_3 → classes

if $y_1 = 0$ → no obj → other parameters dont matter

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

Image → convnet → Face?, l_{1x}, l_{1y} ... l_{64x}, l_{64y} → 129

↳ landmark (point)

detection

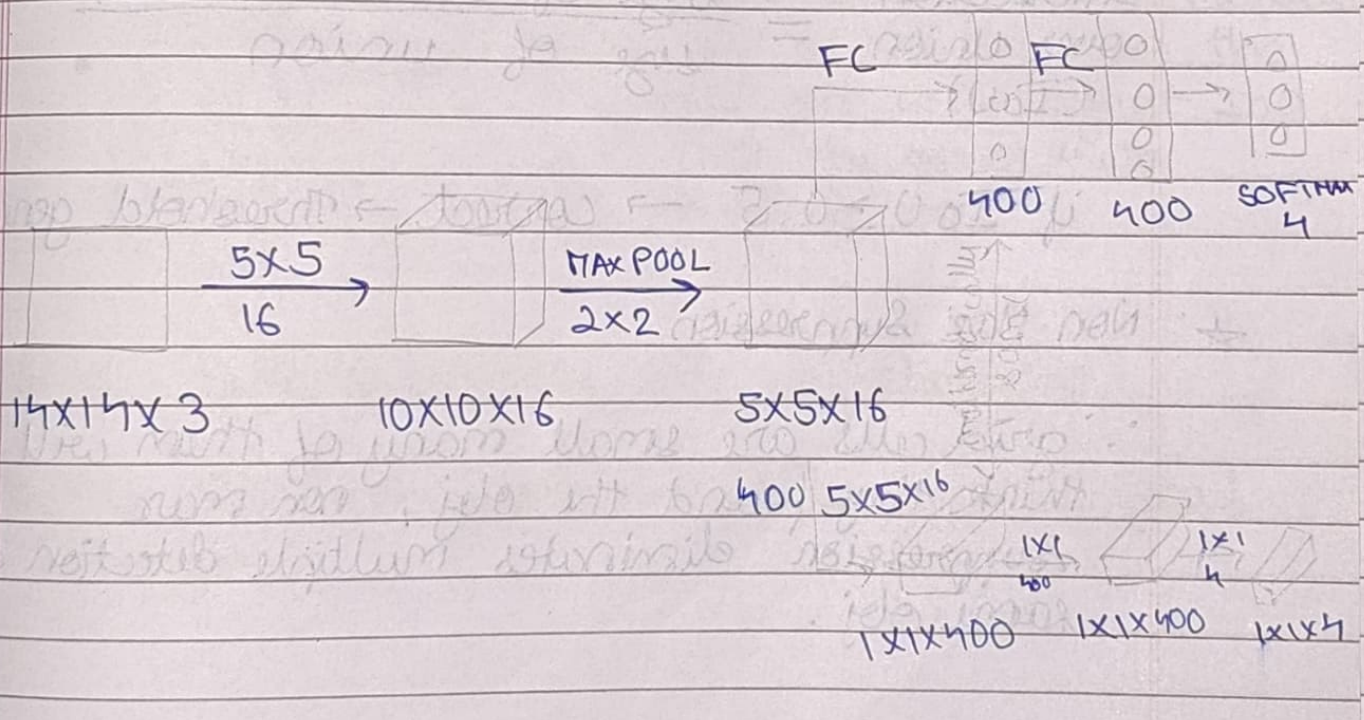
★ Sliding Window Algorithm

SI: Train a convnet using closely cropped car images. i.e image mainly is car

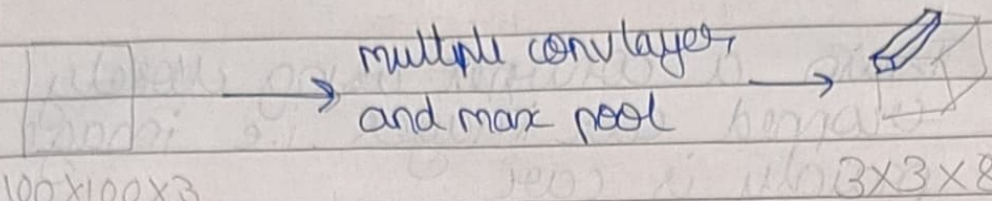
SII: Just like filters in convolution, take parts of images and feed to the CNN.
The window/part of image is made bigger and bigger and resized to the size of input CNN requires.

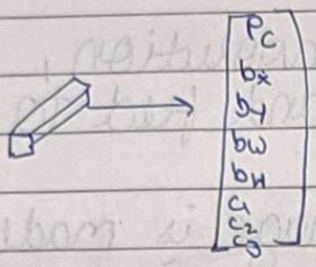
It is very expensive to run.

★ Turning FC layer into convolution layer.



★ YOLO Algorithm





Object is assigned to only one of the grid cells even if it spans multiple grid cells.

b_x, b_y ($0 \le b_x, b_y < 1$), b_w, b_h (could be > 1) are specified relative to grid cell

★ Evaluating object localization

$$\text{Intersection over Union (IoU)} = \frac{\text{size of intersection}}{\text{size of union}}$$

if $\text{IoU} \geq 0.5$ → correct → Threshold generally

★ Non Max Suppression

∵ grid cells are small many of them will think they found the obj, non max suppression eliminates multiple detection of same obj.

It takes the most confident prediction and eliminate the other prediction which have high IoU with the most confident prediction.

Also, $P_c \leq 0.6$ are discarded anyway

remaining boxes having $IoU \geq 0.5$ with most confident prediction are eliminated.

* Anchor box

define many anchor boxes and make y output $P_c, b_x, b_y, b_w, b_h, c_1, c_2, c_3$ corresponding to each anchor box.

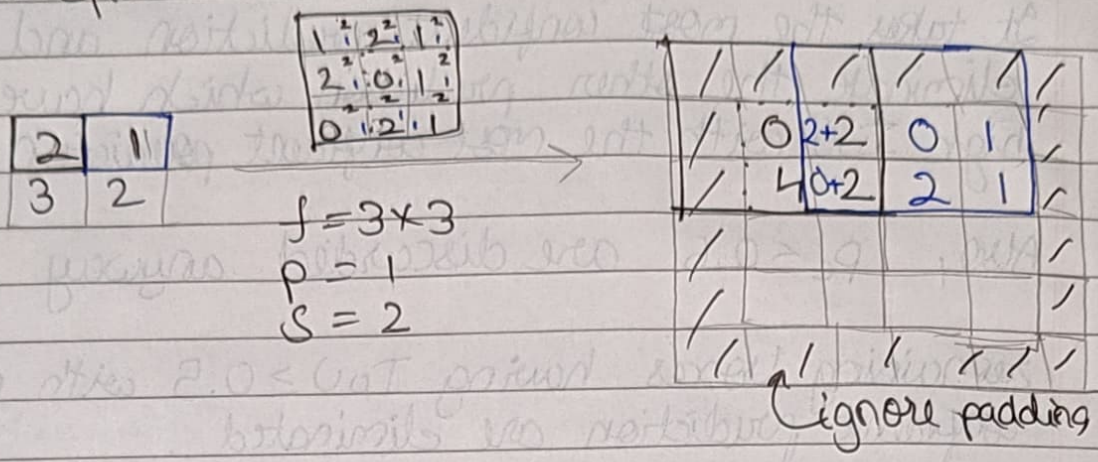
$y = \begin{bmatrix} P_c \\ b_x \\ \vdots \\ c_3 \\ P_c \\ b_x \\ \vdots \\ c_3 \end{bmatrix}$ anchor 1, anchor 2

each obj in training image is assigned to grid cell that contains objects midpoint and anchor box for the grid cell with highest IoU with the ground truth bounding box.

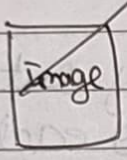
low prob bounding box are eliminated

non max suppression used to generate final predictions.

☆ Transpose Convolution.



☆ U Net Architecture



★ Face verification vs Face Recognition.

Verify: 1:1 — Output is whether input is that of the claimed person.

Recognition: 1:K — To check against database of K persons. output is ID of the person.

★ Oneshot learning

↳ learn from a single picture

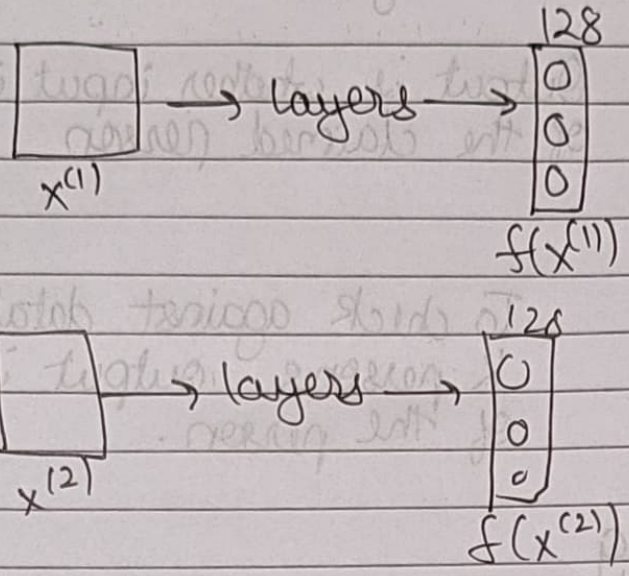
CNN dont work well bcz of small train set

SO, a similarity function is used.

d(1,2) = degree of diff between 1 & 2

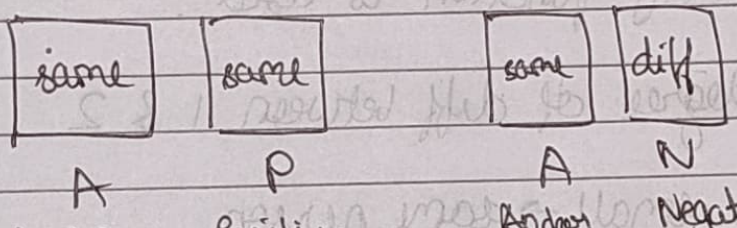
if d(1,2) is small → same person
if d(1,2) is large → diff person

★ Siamese network



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

★ Triplet Loss.



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

$$d(A,P) - d(A,N) \leq 0$$

$$d(A,P) - d(A,N) + \alpha \leq 0$$

↓

margin

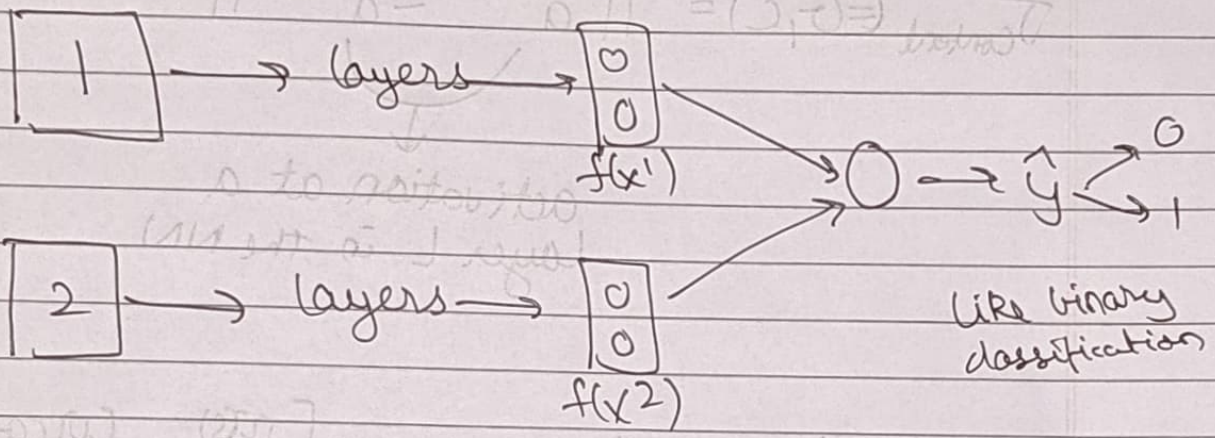
to prevent NN to give same output to all inputs.

α generally 0.2

Loss func: $L(A,P,N) = \max(d(A,P) - d(A,N) + \alpha, 0)$

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

Choose triplets that are hard to train bez if random the equⁿ is eazily satisfied



$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_i \left| f(x^1)_k - f(x^2)_k \right| + b\right)$$

* Neural style transfer

Content + Style = generated image

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

SI: Randomly initialize a image of req dimension

SII: Use gradient descent to minimize $J(G)$.

Content cost function:

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

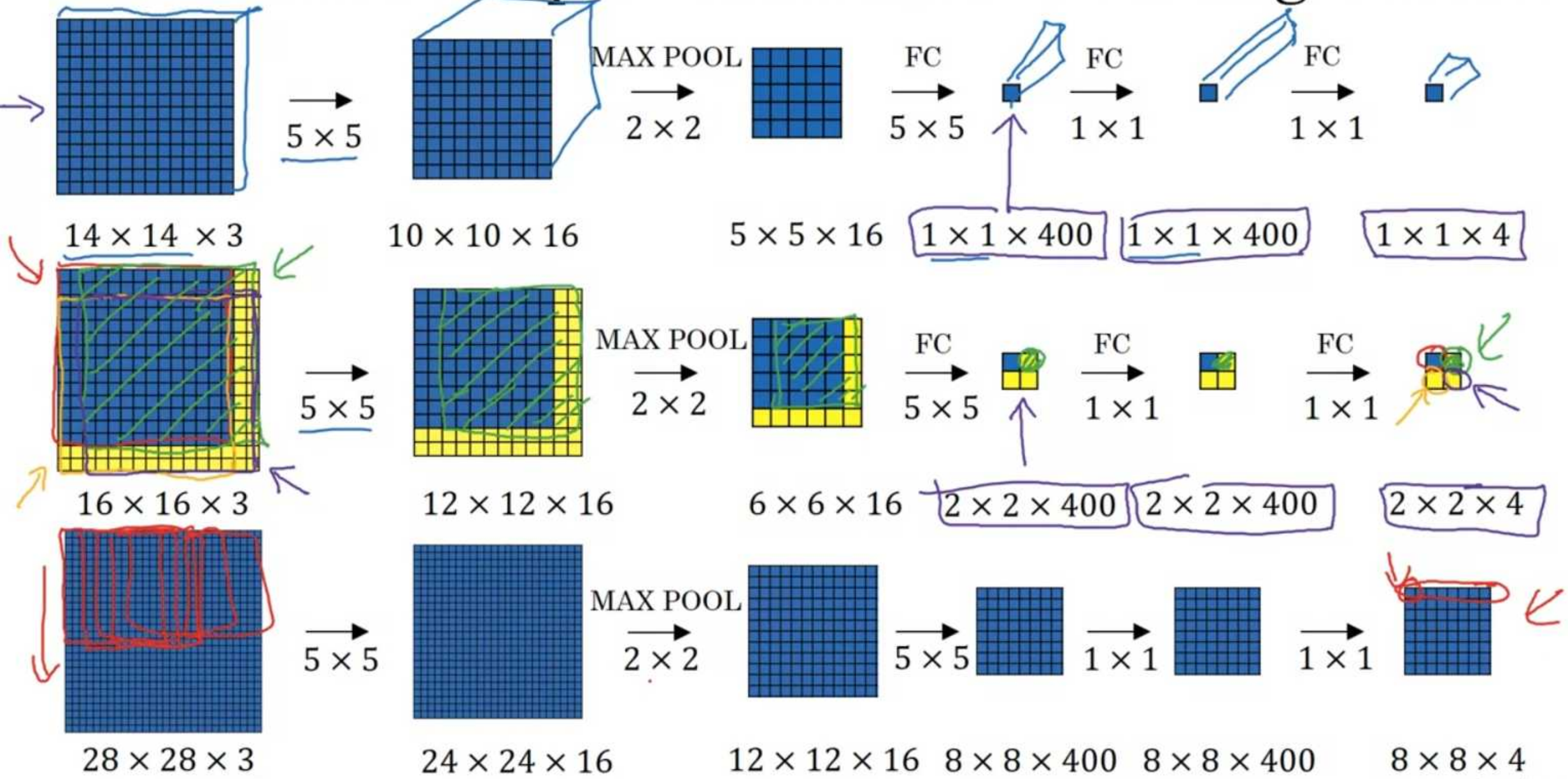
$a^{[l](C)}$: activation at a layer l in the NN

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

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

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

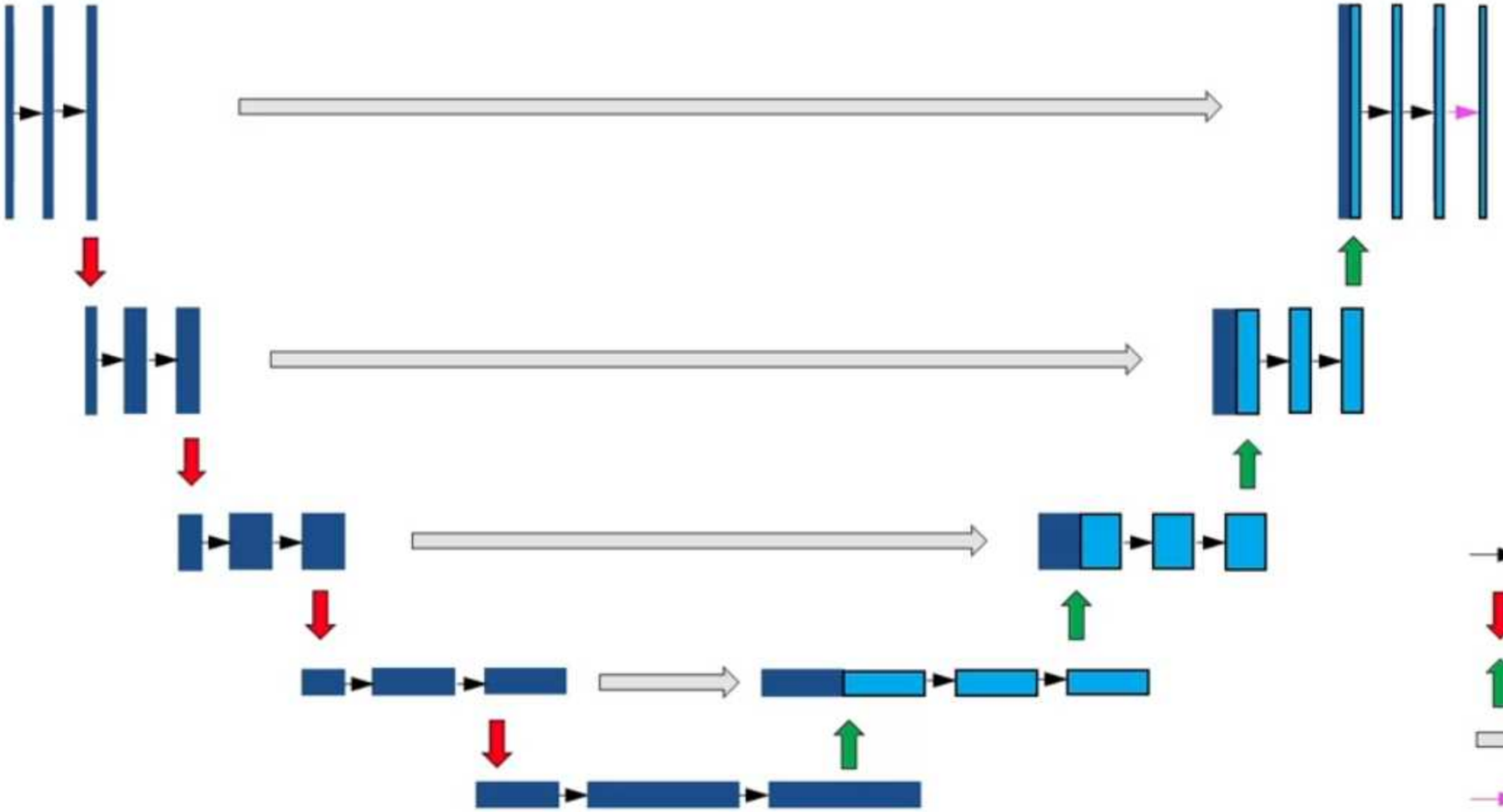
Convolution implementation of sliding windows



[Sermanet et al., 2014, OverFeat: Integrated recognition, localization and detection using convolutional networks]

Andrew Ng

U-Net



[Ronneberger et al., 2015, U-Net: Convolutional Networks for Biomedical Image Segmentation]