Linear Special Filtering

(**) Formula:

- Kernel shape : m x n
- Number of pixels padded : p
- Stride : s
- Input image shape : M x N
- Output image shape : H x W

$$H = \text{floor}\left(\frac{M + 2p - m}{s} + 1\right)$$

$$W = \text{floor}\left(\frac{N + 2p - n}{s} + 1\right)$$

if we want input and output to be same shape:

No. of padding pixels, $p = \frac{m-1}{2}$

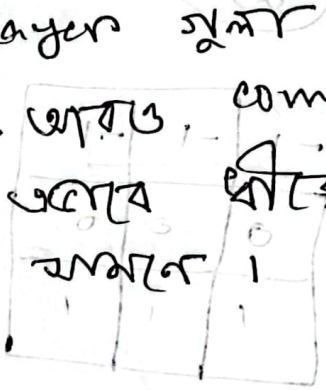
$$= \frac{n-1}{2}$$

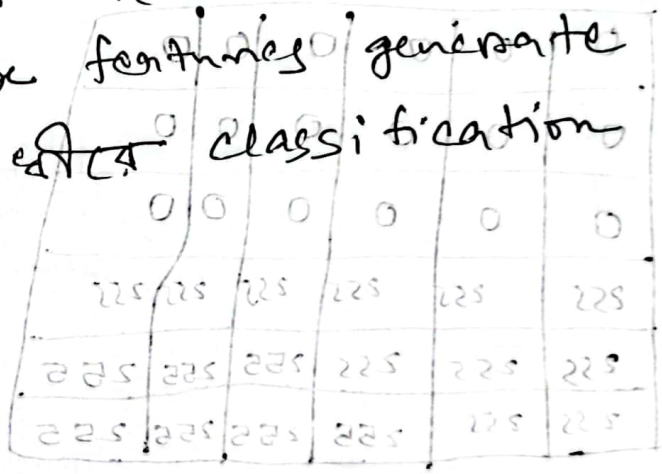
CNN

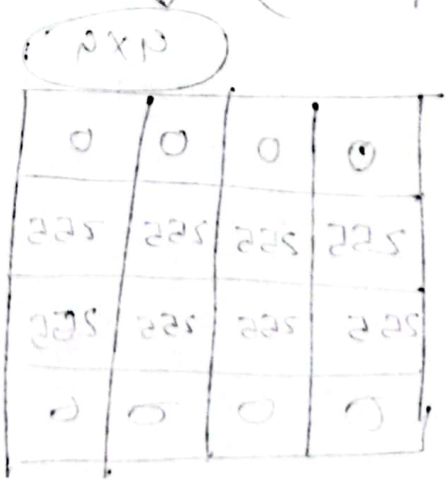
** CNN → Spetial NN
- Convolution layer (filtering)
- Pooling layer (Down sampling)
- FC layer

Conv.

- প্রথম দিকের layer Edge detect করে
- যত deep এ যাওয়া যায়, Conv. layer গুলা ও আরো complex features generate করে, এভাবে ধীরে ধীরে classification করে আমরা।







Convolution layer

** এর কাজ features বের করা একটি image থেকে।

- আমরা দিবো layer edge detect করবে।

** example of a conv. operation:

(inp) 6×6

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

3×3

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

↑ filter / kernel

↓

2 kernel Horizontal edge detection করব

(item wise multiplication)

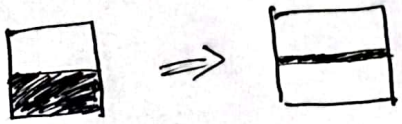
4×4

0	0	0	0
255	255	255	255
255	255	255	255
0	0	0	0

← (output)

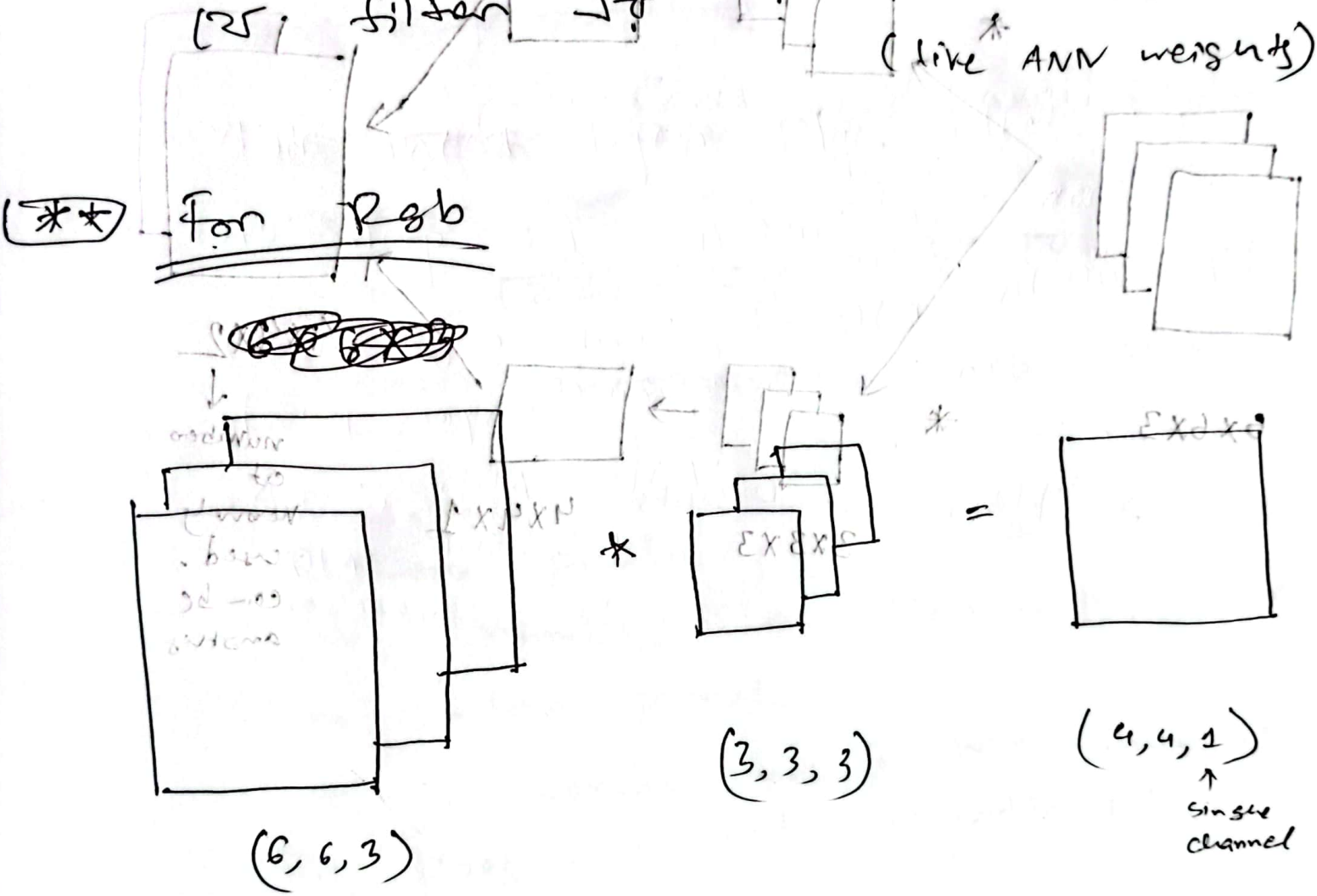
← (feature map)

basically:



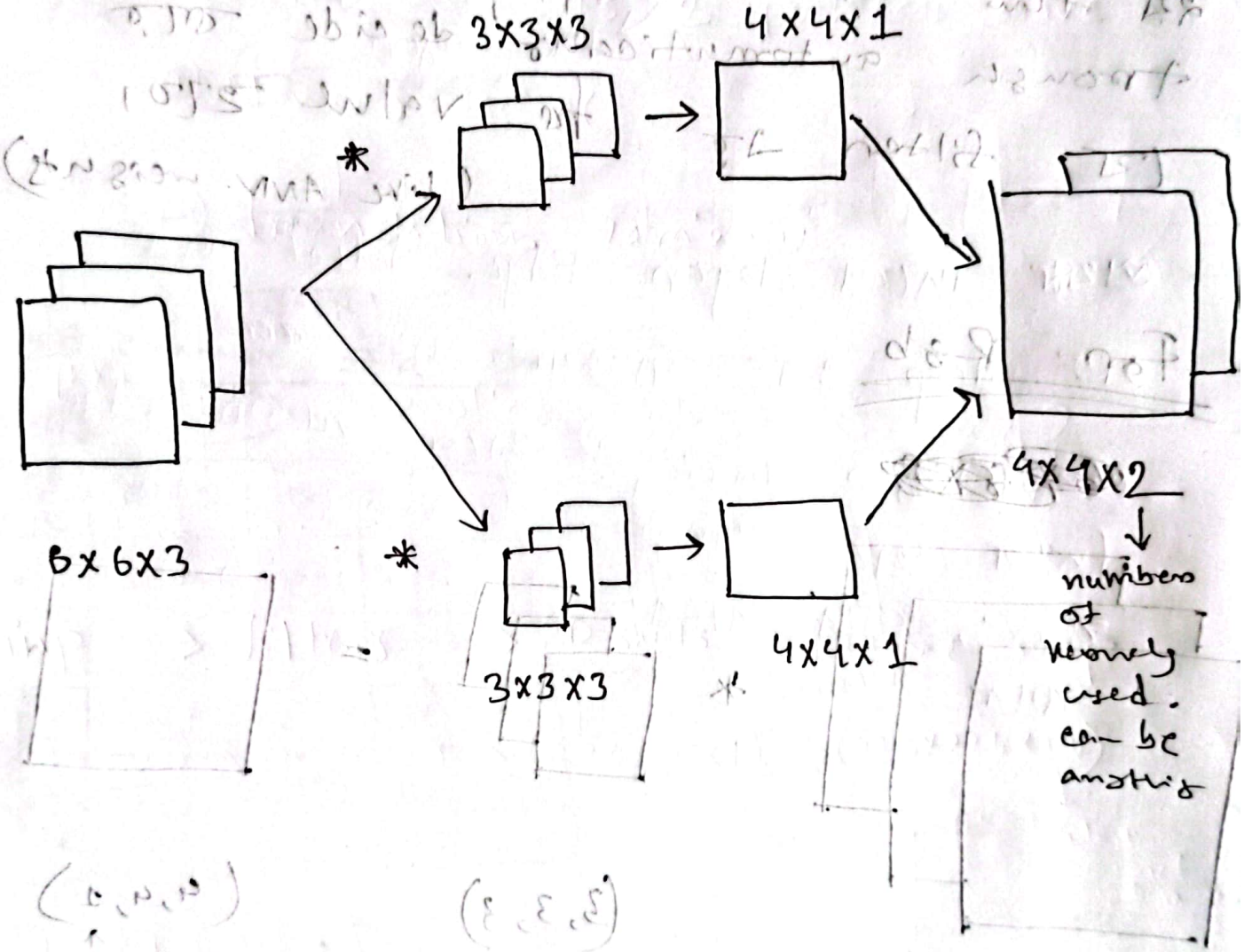
Horizontal edge detection

- filter randomly initialize
  CNN back propagation
  automatically decide
  value
  (like ANN weights)



$$(M \times M \times C) * (m \times m \times c) = (x \times x \times 1)$$

** what if multiple kernel used ?



Padding & Stridge

** Why padding?

~~- So that, the info in the edge does not get lost.~~

- যত বেশি conv. layer use/apply করবো ing প্রতি layer এর output ছোট হতে থাকবে।
  in short information lose হয়।

- Borders pixels are part of less number of conv. in comparison to center pixels.
  so, if Borders pixels are imported. it does not get the attention it deserves.

** Reasons to stride:

(1) When we want high level features, don't want low level features. (higher stride)

(2) Computing power →

Pooling layers

(**) After Conv. ops 2 problems arise →

① Memory issues

② Translation variance

(**) How pooling solves these issues?

Problem ①:

inp → filters → 2D sheets → feature map

(228×228×3) → (100×100×3) → (226×226×1) → (226×226×100)

(226×226×100) → filters/kernel numbers

Huge Numbers (Size)

need to reduce the size of the feature map

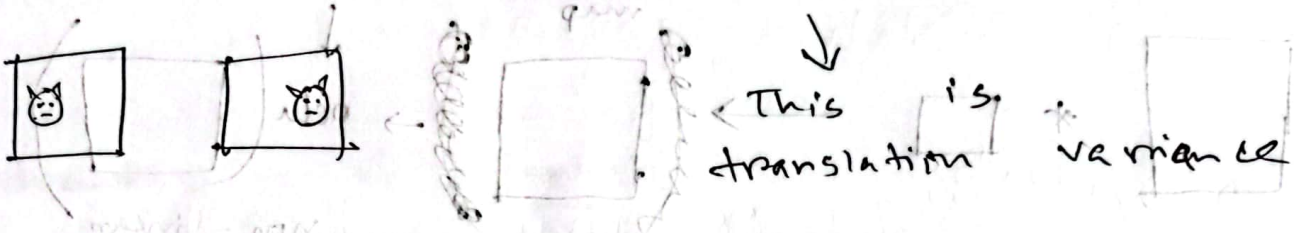
=> stride বাড়ালে যত reduce হয় কিছু

আরও reduce করার দরকার, ঐ

that is done by pooling.

Problem (11):

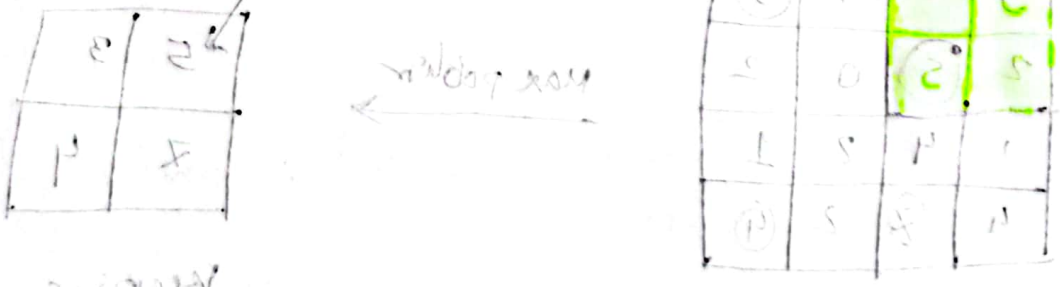
⇒ features become location dependent in CNN



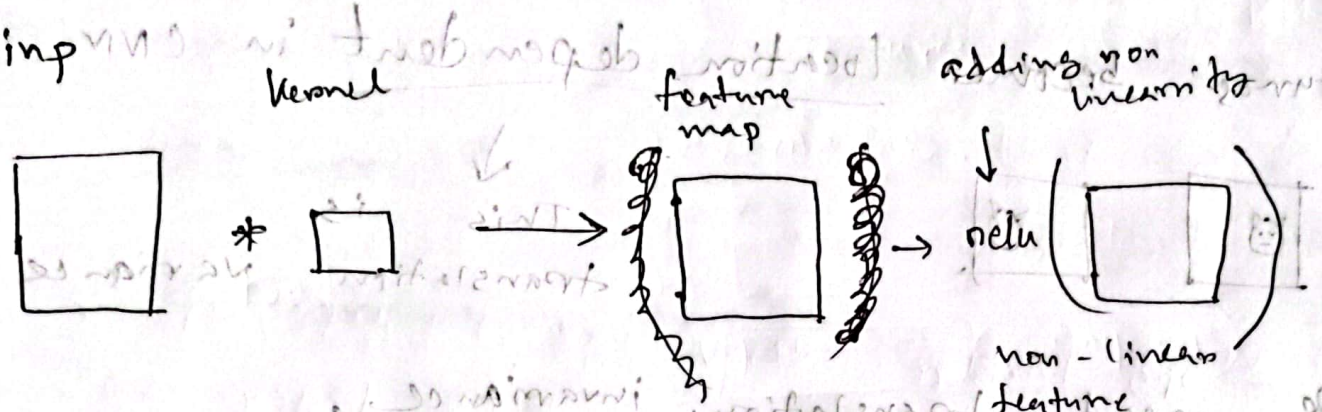
⇒ We need translation invariance.

"feature ~~[illegible]~~ location ~~[illegible]~~ same ~~[illegible]~~ treat ~~[illegible]~~".

(12) Pooling is a way to downsample your ~~[illegible]~~ feature map



**



Then pooling

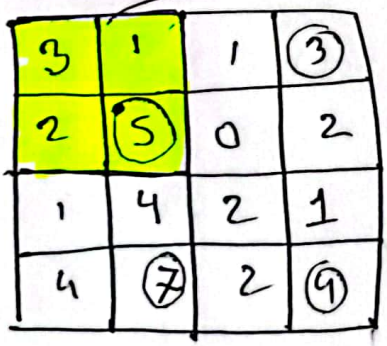
(i) Max pooling

(ii) Min pooling

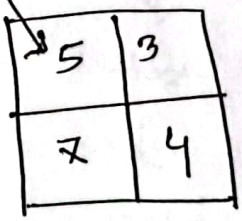
(iii) Avg. pooling

Example:

Non-linear feature map



Max pooling



Given:

Size of window: (2X2)

Stride: 2

type: Maxpooling

Keeping the dominant features in a 2X2 window

↓

- ignoring the low level details

↓

Keeping Higher level details.

Pooling in volumes

individual to pooling in each feature maps

Advantage of feature maps:

(I) Reduce size:

$(228 \times 228 \times 3) * (3 \times 3 \times 3) \rightarrow (226 \times 226 \times 3) \xrightarrow{\text{pooling}} (113 \times 113 \times 100)$

(II) Invariance:

Gives importance of features irrespective of locations.

(III) Enhance feature. (only in the case of Maxpooling)

(IV) No need of traing.

↳ In conv. layers we need training for the kernel but pooling is just a taking the large value.

dis advantage:

→ pooling gives feature of translation invariance.

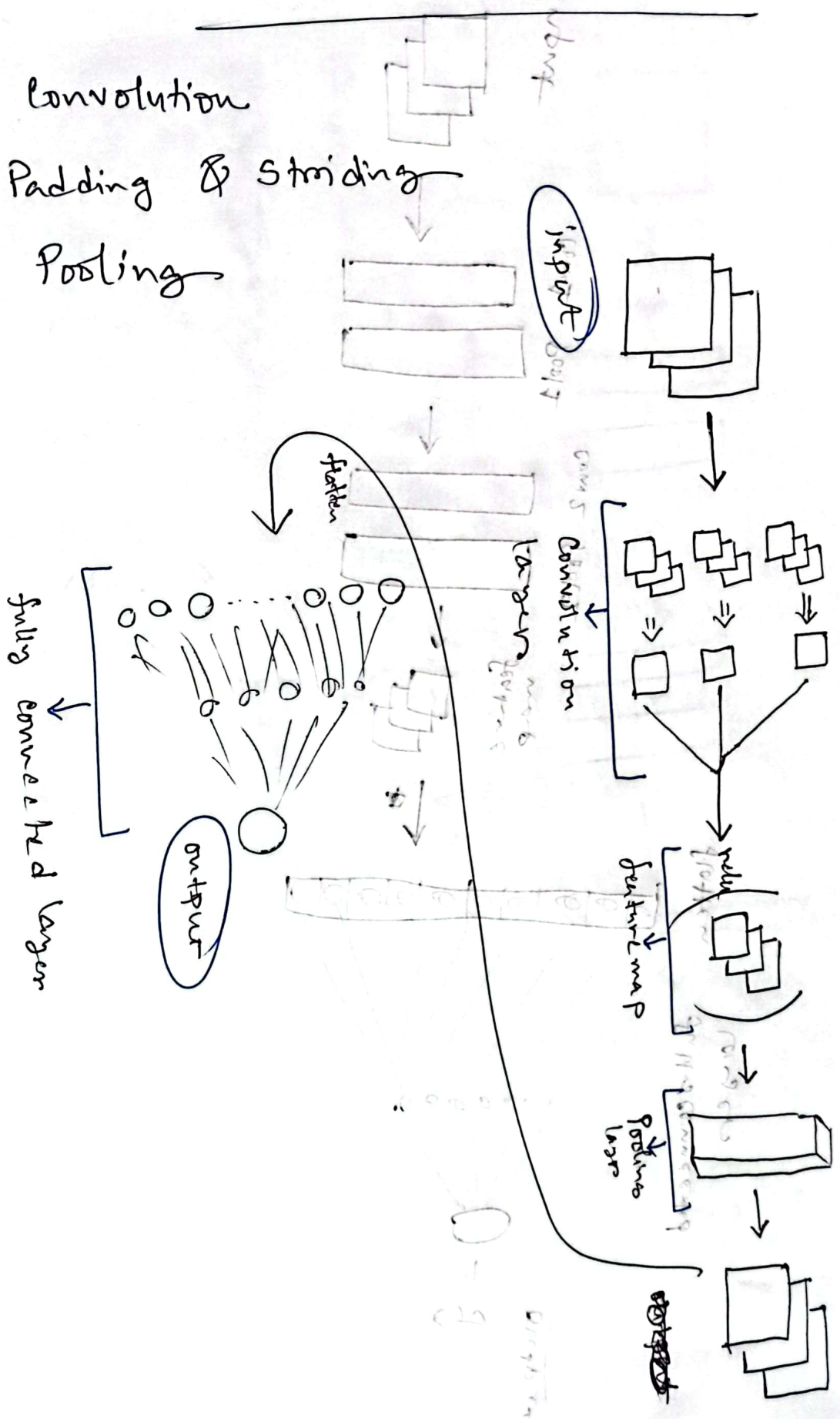
↓ imp in img classification,
(cat img, where img)

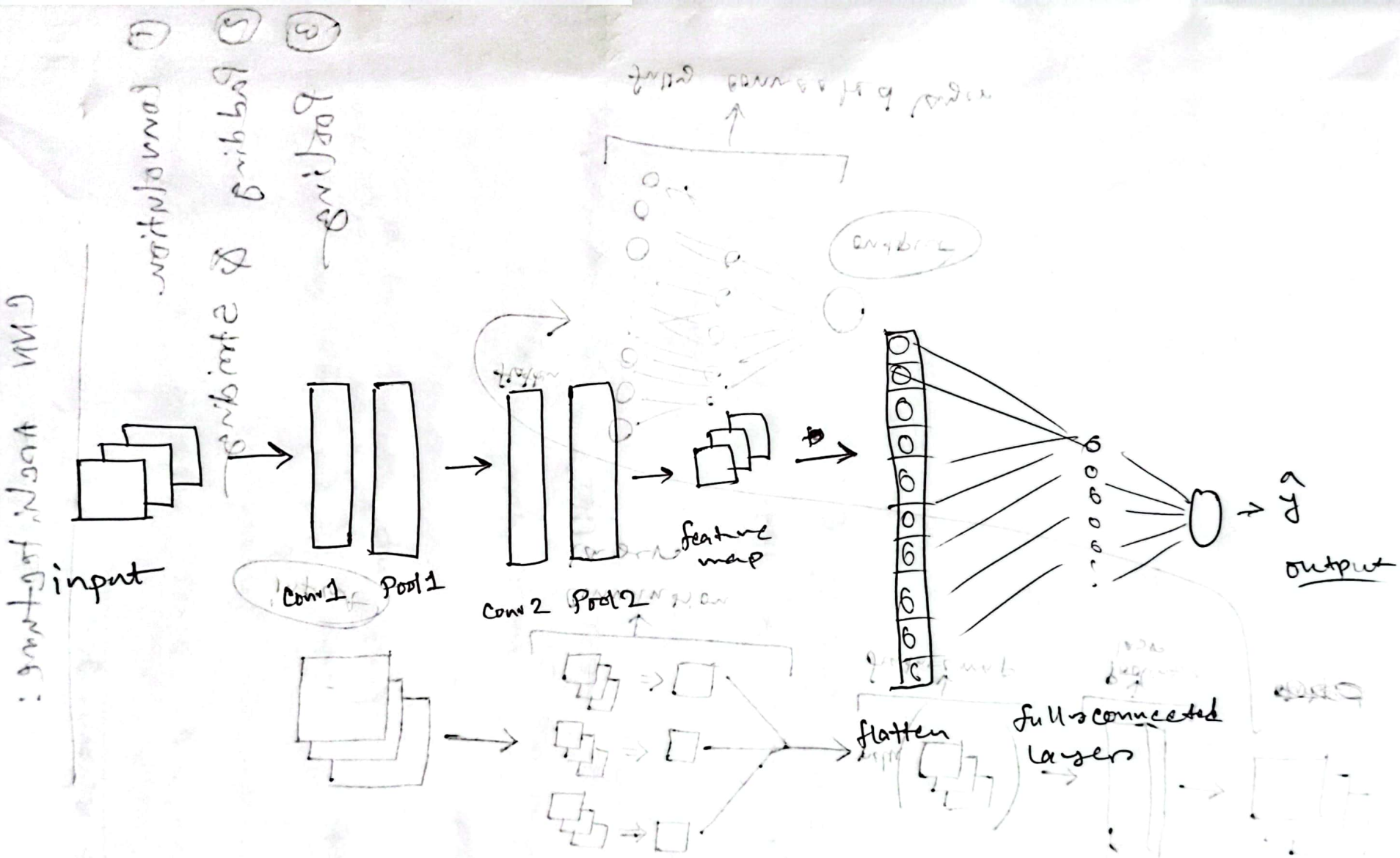
But we need the location in some cases like image segmentation

→ lose a lot of information while reducing.

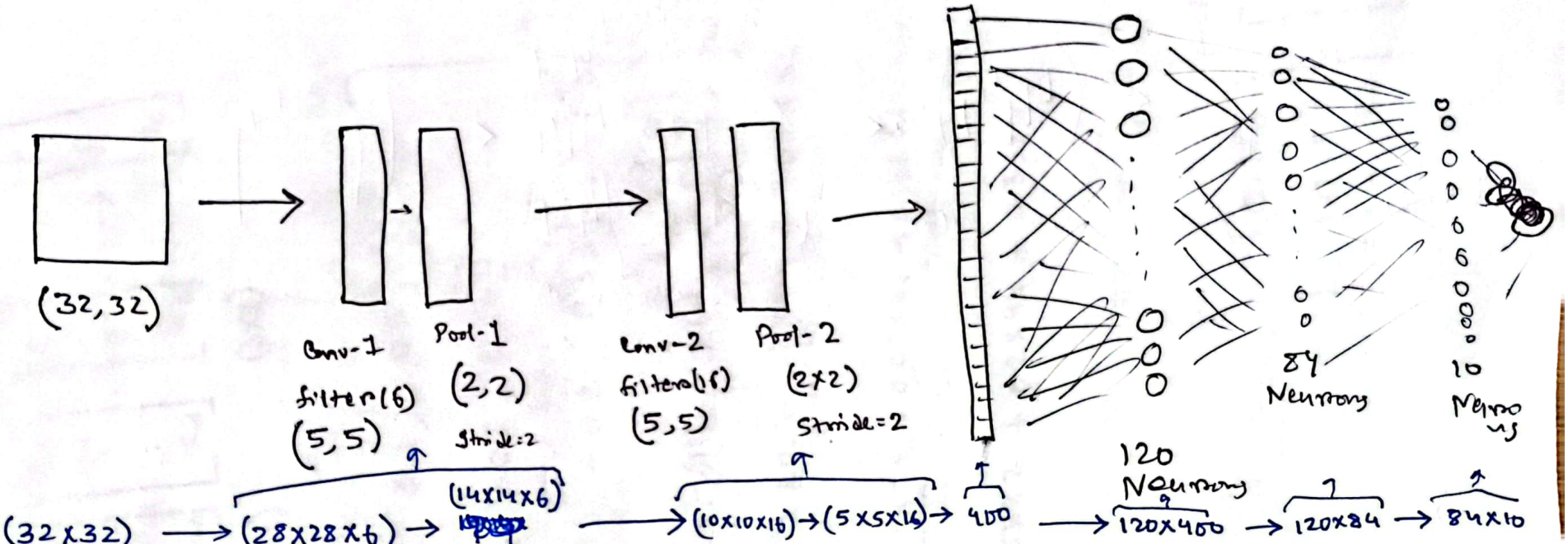
CNN Architecture:

1. Convolution
2. Padding & Striding
3. Pooling




input
Conv 1
Pool 1
Conv 2
Pool 2
feature map
flatten
fully connected layers
output

LeNet-5


(32,32)
Conv-1
filter(6)
(5,5)
Pool-1
(2,2)
stride=2
Conv-2
filter(16)
(5,5)
Pool-2
(2x2)
stride=2
84
Neurons
10
Neurons
(32x32)
(28x28x6)
(14x14x6)
(10x10x16)
(5x5x16)
400
120
Neurons
120x400
120x84
84x10