Module-3

Convolutional Neural Networks : Convolution operation, Convolution layers in neural n/w, pooling, fully connected layers

Case Study :- Architecture of Lenet, Alexnet and VGG16.

* A Convolutional Neural network also known as CNN or ConvNet is a class of neural n/ws that specializes in processing data that has a grid-like topology such as an image.

* The name "convolutional neural n/w" indicates that the n/w employs a mathematical operation called Convolution.

  ↳ Convolution is a specialized kind of linear operation

* Convolutional n/ws are simply neural n/ws that use convolution in place of general matrix multiplication in at least one of their layers

URBAN EDGE

Convolution Operation :-

* Convolution is an operation on two functions of a real valued arguments.

eg:-

- Tracking the location of a spaceship with a laser sensor at discrete time Intervals

- Suppose our sensor is noisy
  ↳ To obtain a less noisy estimate, perform average on several measurements

  ↳ More recent measurements are more important, so perform several weighted average on several measurements.

$$s_t = \sum_{a=0}^{\infty} x_{t-a} w_{-a} = (x * w)t$$

Input Convolution Filter.

↳ The weight assay (w) is known as the filter

↳ Slide the filter over the input and compute the value of s_t based on a window around x_t.

URBAN EDGE

↳ Using 6 precedings:

$$S_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
0.01	0.01	0.02	0.02	0.04	0.4	0.5

X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70

1.80	

$$S = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

1.80

↳ Here the input and the kernel is one dimensional.

↳ If we are using 2D inputs, we need to use 2D filter (m x n)

↳ So the 2D formula becomes:

$$S_{ij} = (I * K)_{ij}$$

$$= \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a', j+b} k_{a,b}$$

URBAN EDGE ___/___/___

eg:-

Input

a	b	c	d
e	f	g	h
i	j	k	l

Kernel

w	x
y	z

o/p

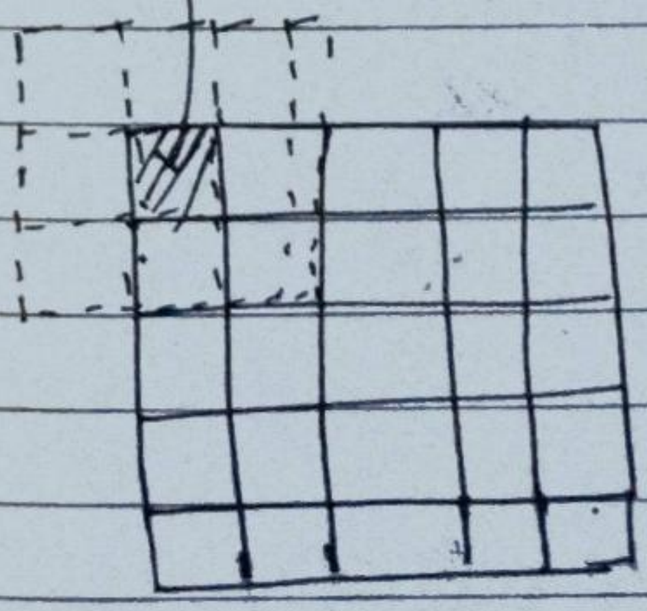
$aw+bx+ey+fz$	$bw+cx+fy+gz$	$cw+dx+gy+hz$
$ew+fx+iy+jz$	$fw+gx+jy+kz$	$gw+hx+ky+lz$

* If the pixel of interest is on center, then the kernel is centered on the pixel of interest

$$S_{ij} = (I * K)_{ij}$$

$$= \sum_{a=\lfloor -m/2 \rfloor}^{\lfloor m/2 \rfloor} \sum_{b=\lfloor -n/2 \rfloor}^{\lfloor n/2 \rfloor} I_{i-a,j-b} K_{m/2+a', n/2+b}$$

Pixel of Interest



* Each time we slide the kernel we get one value in the output
* The resulting output is called a feature map
* Use multiple features to get multiple feature maps.
* In the 1D case, slide a one dimensional filter over a one dimensional input
* In the 2D case, slide a two dimensional filter over a two dimensional input.
* In the 3D case, filter is a 3D, refer it as a volume. So slide the volume over the 3D input and compute the convolution operation. But the filter moves along the height and the width but not along the depth. As a result, the o/p will be 2D. Also can use multiple filters to get multiple feature map.

URBAN EDGE ___/___/___

Relationship b/w input size, output size and filter size.

* Define the following Quantities

*→ Width (W_1), Height (H_1) and depth (D_1) of the Original input

*→ Stride S

[Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on.

*→ Number of filters "k". If "k" filters, we will get "k" feature map

*→ The Spatial Extent (F) of each filter (like 3×5 etc)

[The depth of each filter is same as the depth of each input]

[Depth of the filter is same as the depth of each input]

* The o/p is again a volume $W_2 \times H_2 \times D_2$

Compute the dimension (W_2, H_2) of the o/p :-

↳ Slide the filter along the input to get the o/p (feature map)

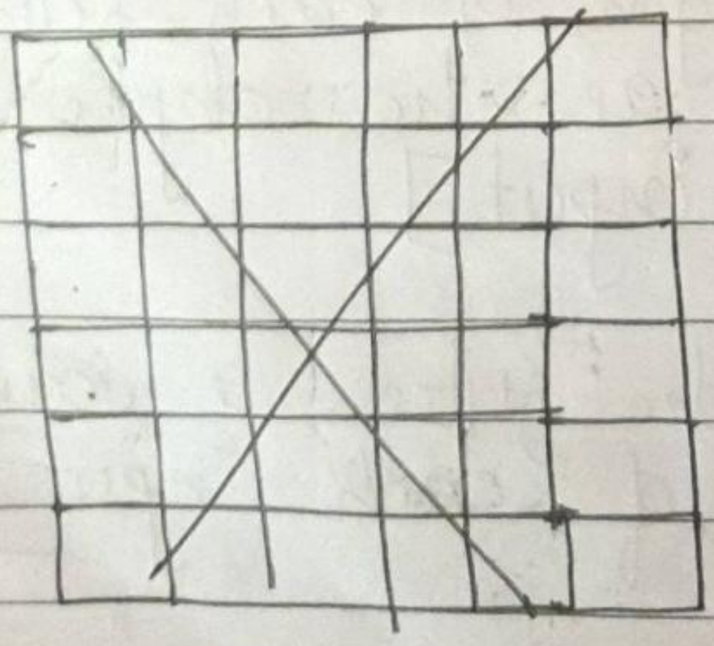
↳ But for some pixels, we cannot place the filter. because it will go out of the boundary

↳ For 3×3 filter, the o/p size of the feature map decreased by width 2 and height 2

∴ Results in an output which is of smaller dimensions than the input.

↳ As the size of the kernel increases, the above reduction in dimension is true for even more pixels

↳ Consider 5×5 kernel, even smaller o/p



3×3 filter → 2 columns subtracted

5×5 filter → 4 sub

7×7 filter → 6 sub.

URBAN EDGE

* In general $(W_1 - (F-1))$

$$W_2 = W_1 - F + 1$$

$$H_2 = H_1 - F + 1$$

* If we want the o/p to be of same size as the input, we can use padding

↳ Pad the inputs with appropriate no: of 0 inputs so that we can now apply the kernel at the corners

$$W_2 = W_1 - F + 2p + 1$$

$$H_2 = H_1 - F + 2p + 1$$

↳ If stride of 2

$$W_2 = \frac{W_1 - F + 2p}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2p}{S} + 1$$

↳ Depth of the o/p = K (no: of filters)

eg:- $227 \times 227 \times 3$ i/p

11×11 filters

96 = K

4 = S

} no padding

$O_2 = 96$ $W_2 = 55$ $H_2 = 55$

URBAN EDGE

Convolutional Layers in Neural Network:-

Convolution leverages three important ideas that motivated Computer Vision researchers.

(1) Sparse Interaction
(2) Parameter sharing
(3) Equivariant representation

* Traditional neural n/w layers use matrix multiplication by a matrix of parameters describing the interaction b/w the input and output unit. This means that every o/p unit interacts with every i/p unit.

* In Convolutional Neural networks, have sparse interaction. This is achieved by making kernel smaller than the input
  eg:- An image can have millions or thousands of pixels, but while processing it using kernel we can detect meaningful information that is of tens or hundreds of pixels.

  ↳ This means that we need to store fewer parameters that not only reduces the memory requirement of the model but also improves the statistical

efficiency of the model.

* Parameter sharing / weight sharing :-

If computing one feature at a spatial point (x_1, y_1) is useful then it should also be useful at some other spatial point say (x_2, y_2). In a traditional neural n/w, each element of the weight matrix is used once and then never revisited/ used, while convolution n/w has shared parameters

ie for getting o/p, weights applied to one i/p are the same as the weight applied elsewhere.

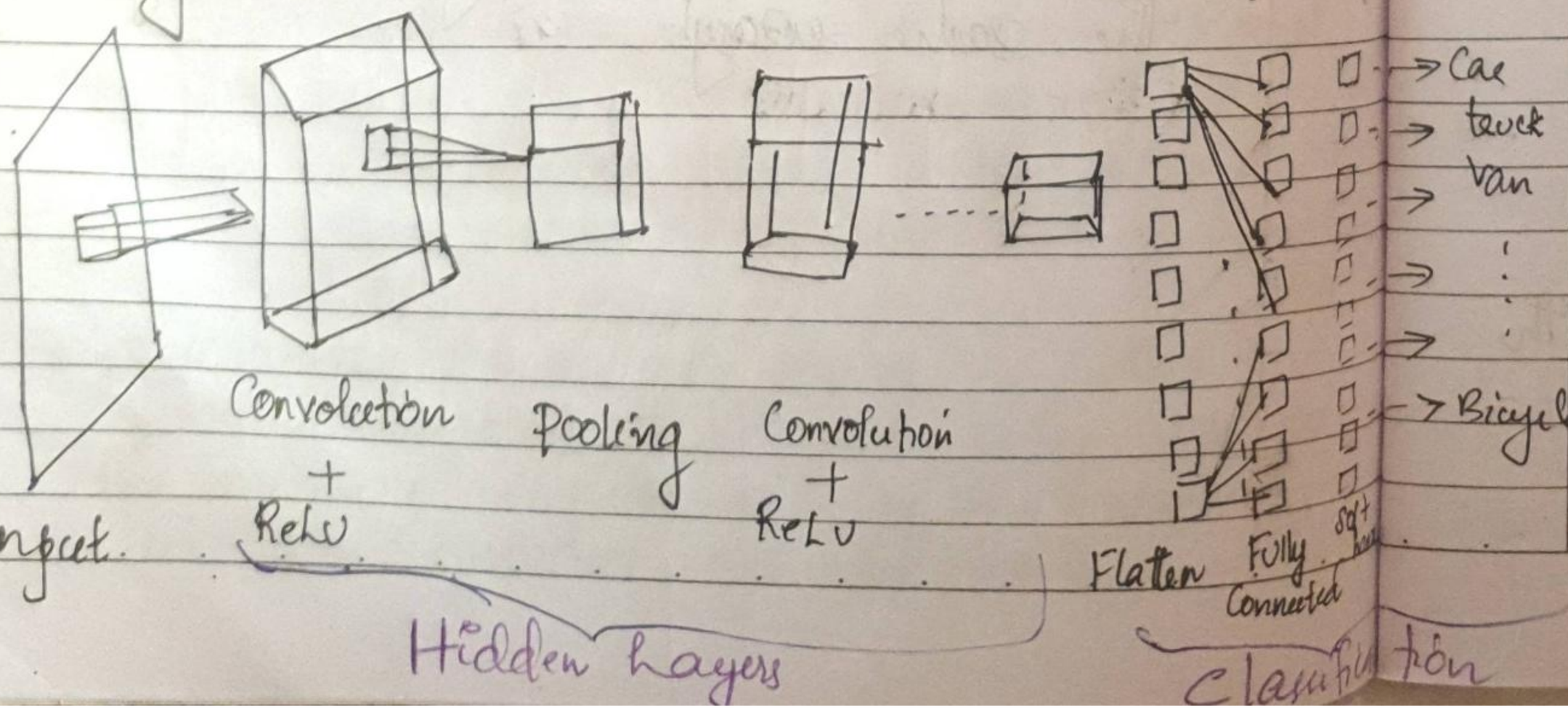
* Due to parameter sharing, the layers of convolutional of neural n/w will have a property of equivariance to translation.

ie If we changed the i/p in a way, the o/p will also get changed in the same way.

URBAN EDGE

Convolutional Layer:-

* The most important building block of a CNN is the convolutional layer
* Neurons in the first convolutional layer are not connected to every single pixels in their respective fields.
* Each neuron in the second convolutional layer is connected only to neurons located within a small rectangle (filter) in the first layer.
* This architecture allows the n/w to concentrate on small low-level features in the first hidden layer, then assemble them into larger higher-level features in the next hidden layer and so on. This is the reason why CNN's work so well for image recognition:



URBAN EDGE

Zero Padding :-

A neuron located in row i, column j of a given layer is connected to the outputs of the neurons in the previous layer located in rows i to $i+f_h-1$, columns j to $j+f_w-1$ [where f_h and f_w are the height and width of the respective field. In order for a layer to have the same height and width as the previous layer, it is common to add zeros around the inputs. This is called zero padding.

Filters :-

In Convolution neural network, the kernel is nothing but a filter that is used to extract the features from the images. The kernel is a matrix that moves over the input data, performs the dot product with the sub-region of input data, and gets the output as the matrix of dot products

URBAN EDGE

Stacking Multiple Feature Maps.

* Output of each convolutional layer as a thin 2D layer.

* But in reality a convolutional layer has multiple filters, and its output one feature map per filter, so it is more accurately represented in 3D.

* To do so, it has one neuron per pixel in each feature map, and all neurons within a given feature map share the same parameters
  - Neurons in different feature maps use different parameters.

* Convolutional layer simultaneously applies multiple trainable filters to its inputs, making it capable of detecting multiple features anywhere in its inputs.

Computation of the o/p of a given neuron in a convolutional layer:-

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{with}$$

$$\begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases}$$

URBAN EDGE ___/___/___

$Z_{i,j,k}^{l}$ = o/p of the neuron located in row i, column 'j' in feature map k of the convolutional layer (layer l)

S_h and S_w = Vertical and horizontal strides

f_h and f_w = height and weight of the receptive field.

$f_{n'}$ = no: of feature maps in the previous layer (layer l-1)

$x_{i',j',k'}^{l-1}$ = o/p of the neuron located in layer (l-1)

b_k = Bias term for feature map k

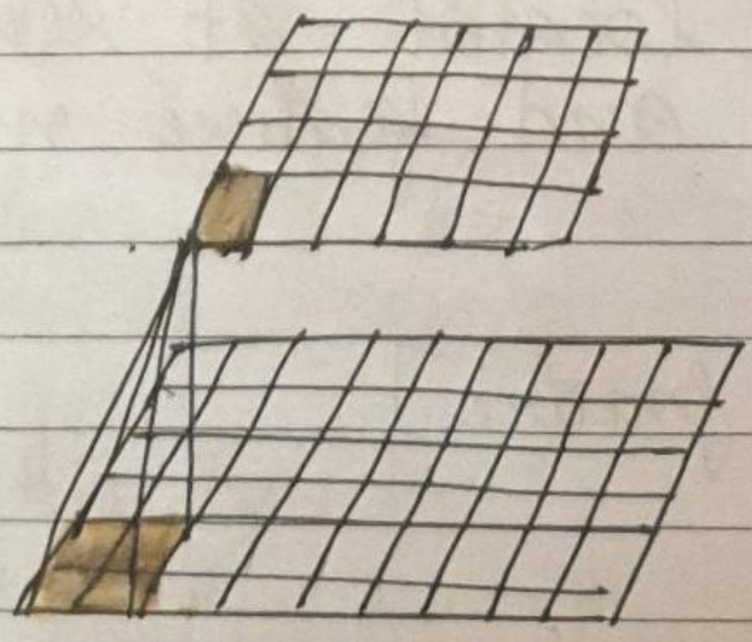
$w_{u,v,k',k}$ = Connection weight b/w any neuron in feature map 'k' of the layer 'l' and its input located at row u, column v and feature map k'.

Pooling layer:-

* Main goal of pooling layer is to subsample (shrink) the input image in order to reduce the computational load, the mly usage, and the no: of parameters (thereby limiting the risk of overfitting)

* Each neuron in a pooling layer is connected to the outputs of a limited no: of neurons in the previous layer located within a small rectangular receptive field.

* Pooling neuron has no weights, instead it aggregates the inputs using an aggregation function such as mean or max.



max.

1	(5)
3	2

URBAN EDGE ___/___/___

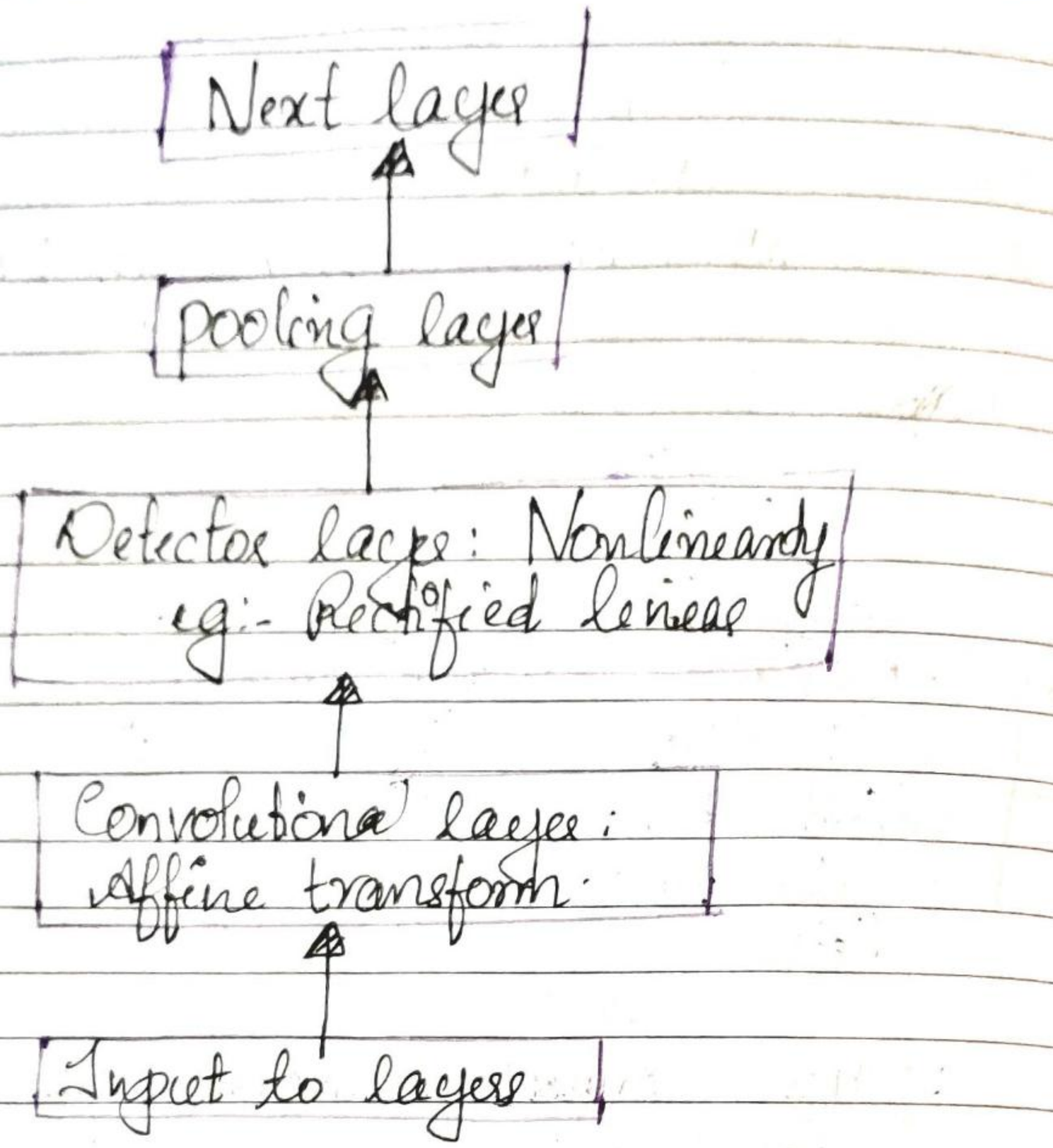
* Other than reducing computations, memory usage and the no: of parameters, a max pooling layer also introduces some level of invariance to small translations

* Max pooling also offers a small amount of rotational invariance and a slight scale invariance. Such invariance can be useful in cases where the prediction should not depend on these details, such as in classification tasks

* In Semantic segmentation [task of classifying each pixel in an image depending on the object that pixels belong to]
if the input image is translated by 1 pixel to the right, the op should also be translated by 1 pixel to the right. This is called Equivariance. [A small change to the inputs should lead to a corresponding small change in the output.

URBAN EDGE

Components of typical Convolutional Neural Network layers.



eg.