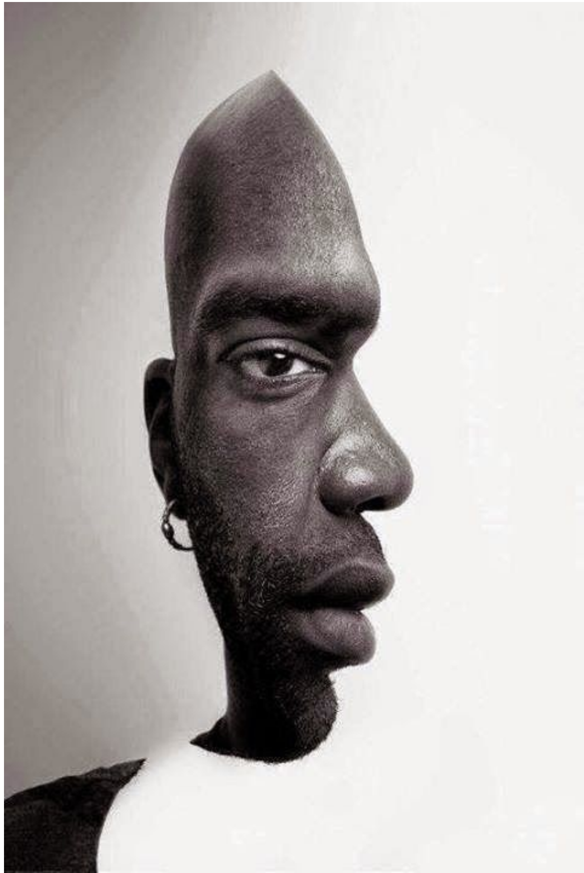
Convolutional Networks

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

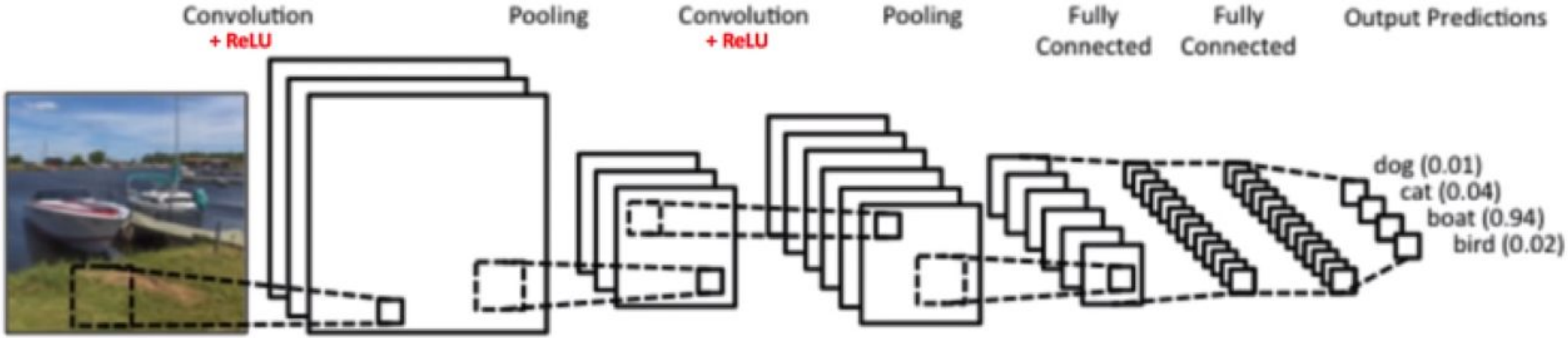
MLP: Problems

- The amount of weights rapidly becomes unmanageable for large input images
  - E.g. 224 x 224 x 3 image: >150,000 weights!
- Not translation invariant
  - Reacts differently to an input's shifted version

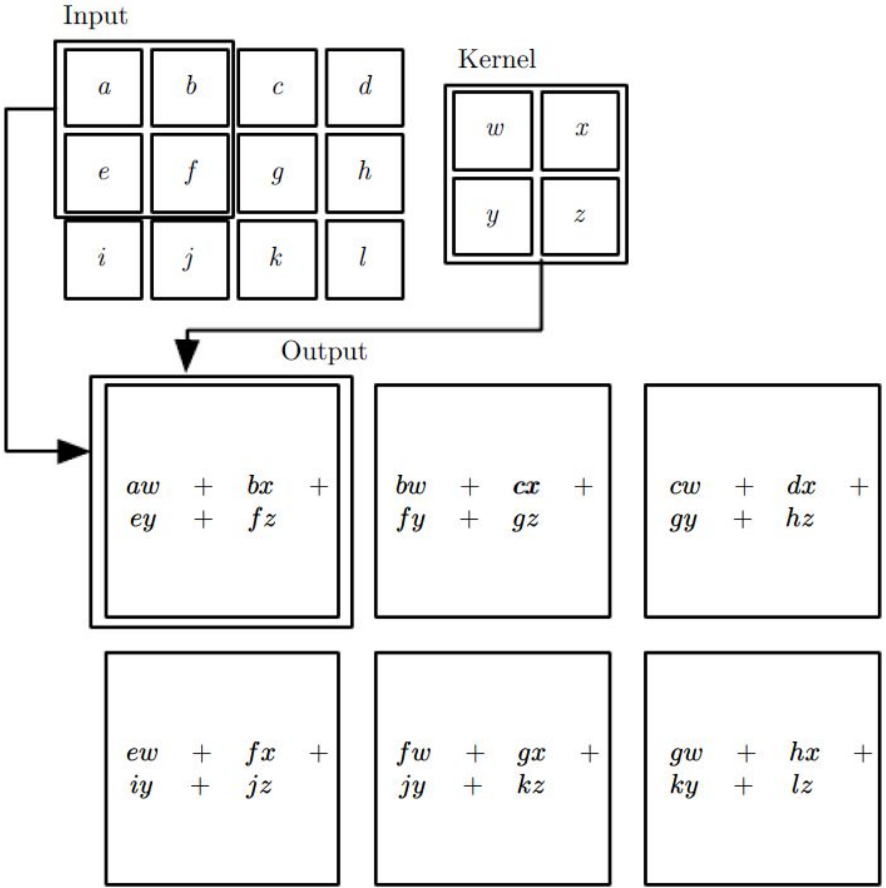
What do you see? Variance in Input Image



CNN: Architecture



Convolution Operation



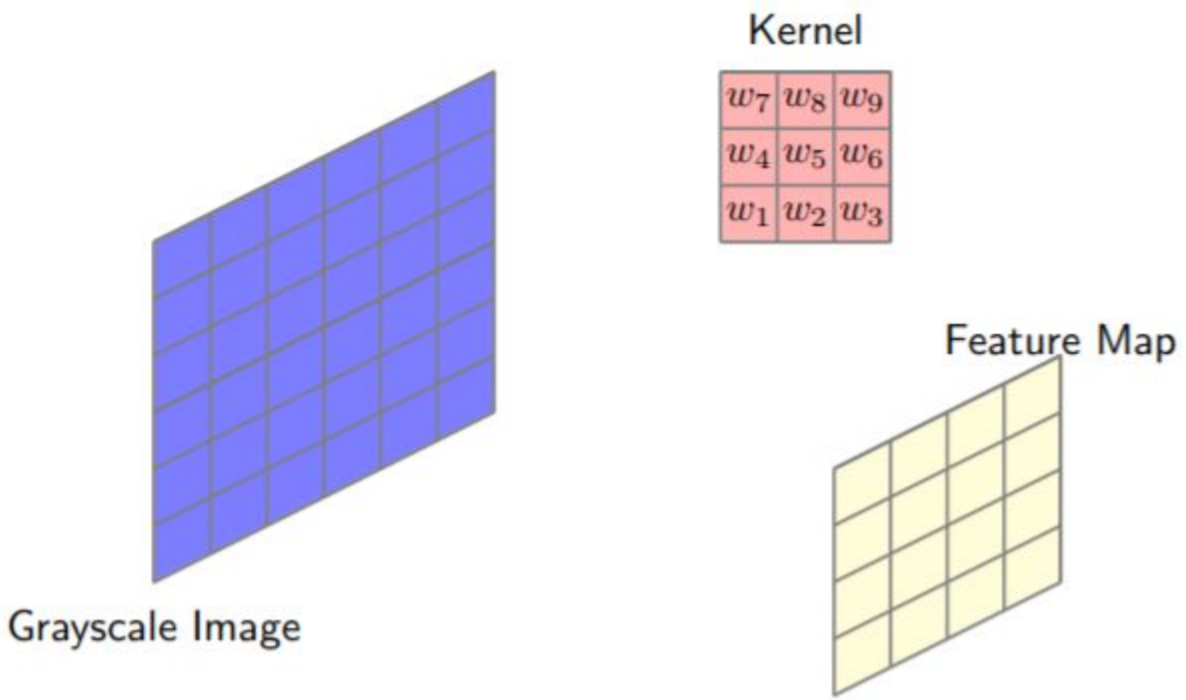
Convolution Operation

- $s(t) = (x * w)(t) = \sum_{a = -\infty, +\infty} x(a)w(t - a)$
  - x: Input, w: Kernel, s: Feature map

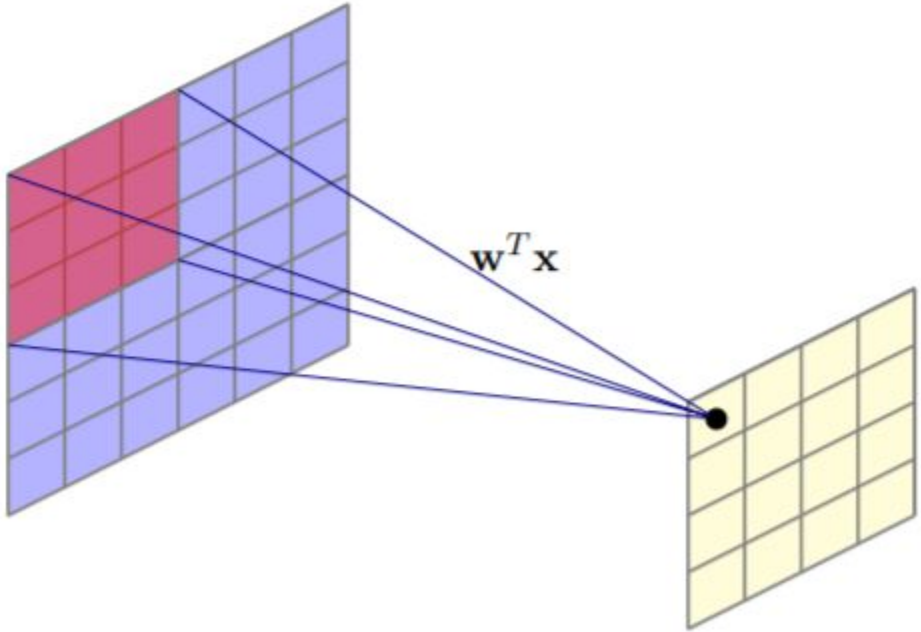
- Two-dimensional
  - $S(i, j) = (I*K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n)$

- Convolution is commutative
  - $S(i, j) = (K*I)(i, j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$

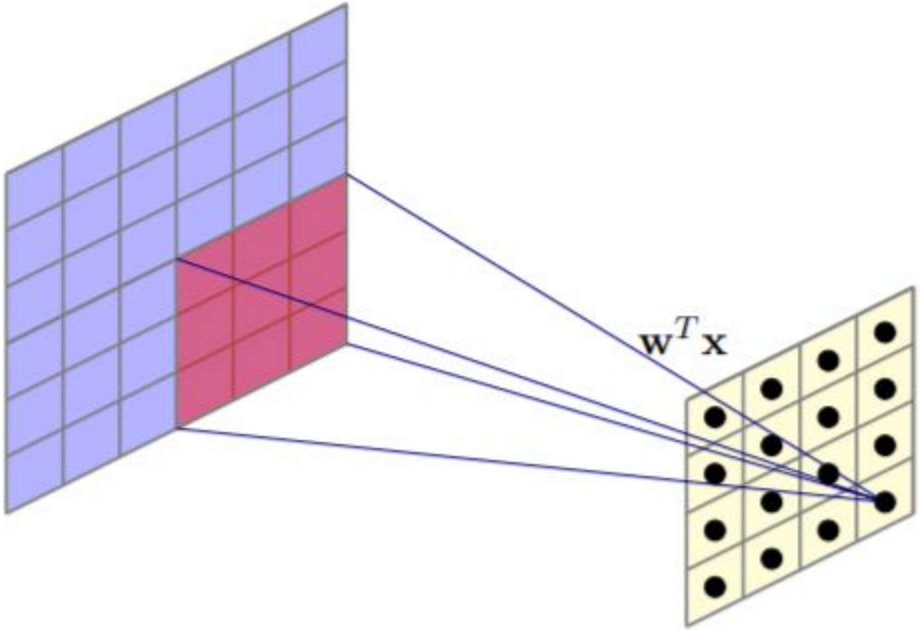
Visual example: Convolution Operation



Visual example example: Convolution Operation

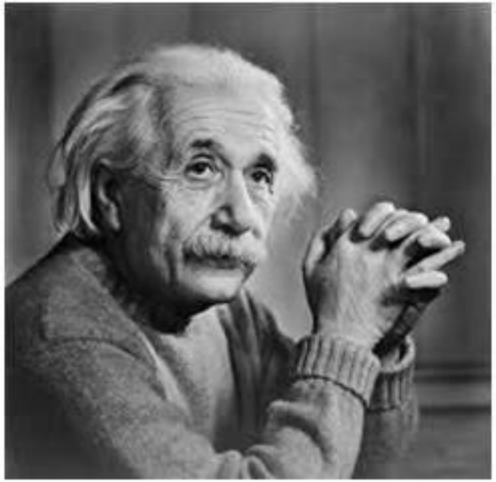


An example: Convolution Operation



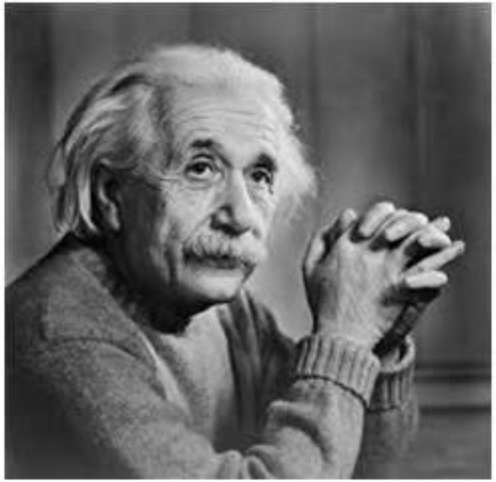
Convolution: Vertical

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



Convolution: Horizontal

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



Convolution Operation: Output Size

- For convolutional layer:
  - Suppose input is of size $W_1 \times H_1 \times D_1$
  - Filter size is K and stride S
  - We obtain another volume of dimensions $W_2 \times H_2 \times D_2$
- As before:
  - $W_2 = (W_1 - K)/S + 1$ and $H_2 = (H_1 - K)/S + 1$
- Depths will be equal

Convolution Operation: Output Size

- Output size: (N – K)/S + 1
  - N: input dimension
  - K: Kernel size
  - S: Stride
- In previous example:
  - N = 6, K = 3, S = 1,
  - Output size =
    - 4

Convolution: Motivation

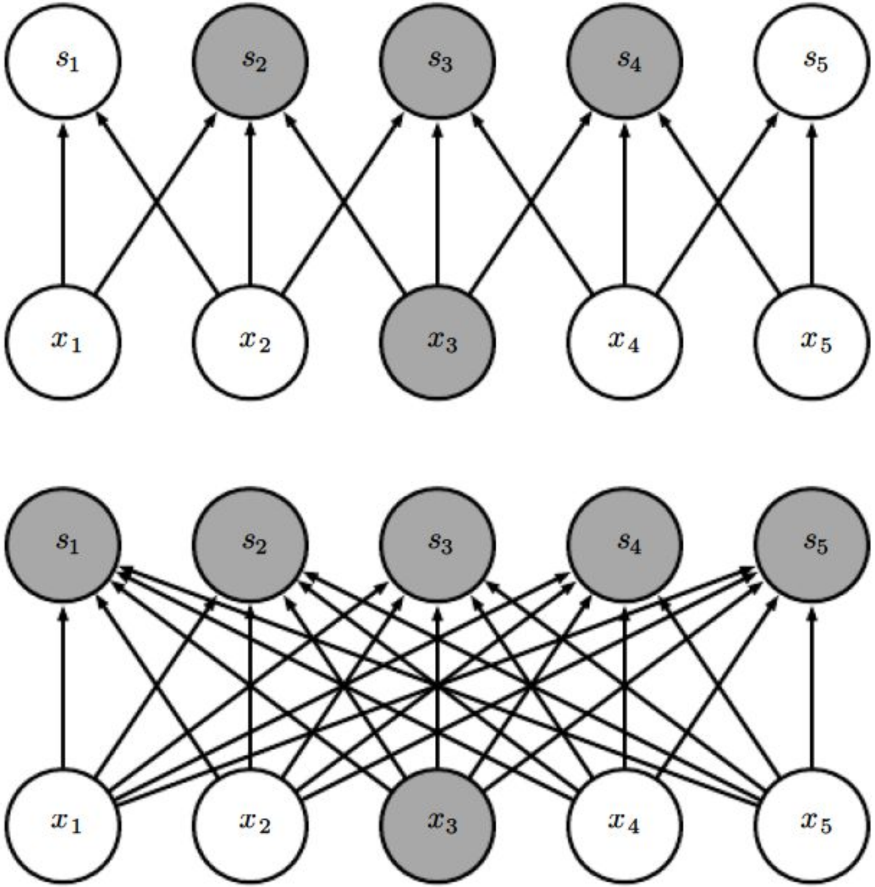
- Convolution leverages four ideas that can help ML systems:
  - Sparse interactions
  - Parameter sharing
  - Equivariant representations
  - Ability to work with inputs of variable size

Convolution Operation: Sparse Interactions

- Sparse interactions
  - Convolutional networks typically have sparse interactions (also referred to as sparse connectivity or sparse weights)

- This is accomplished by making the kernel smaller than the input.
  - need to store fewer parameters, computing output needs fewer operations ($O(m \times n)$ versus $O(k \times n)$)

Convolution Operation: Sparse Interactions

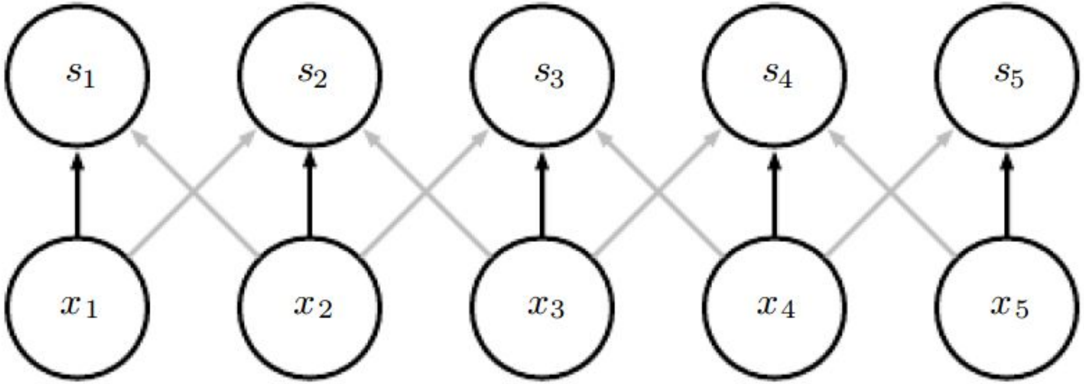


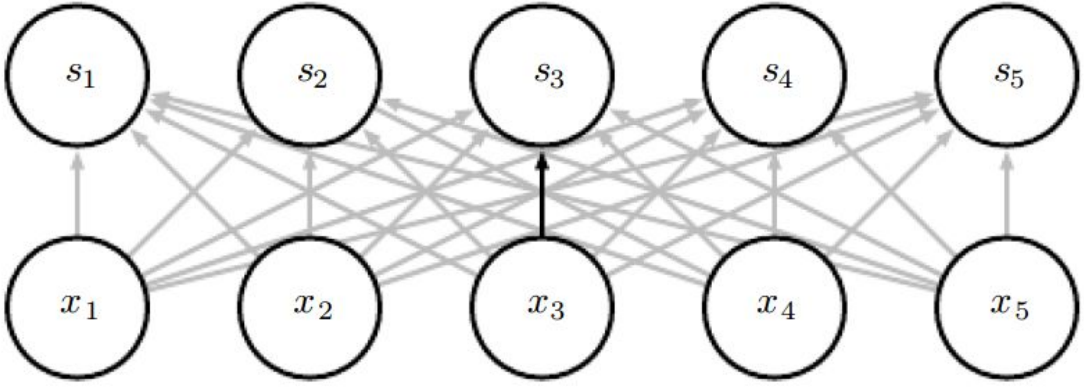
Convolution Operation: Parameter Sharing

- Parameter sharing refers to using the same parameter for more than one function in a model
  - Each member of the kernel is used at every position of the input

- Rather than learning a separate set of parameters for every location, we learn only one set.
  - This does not affect the runtime of forward propagation; it is still $O(k \times n)$, but it does further reduce the storage requirements of the model to k parameters.
  - Storage improves dramatically as $k \ll m, n$

Convolution Operation: Parameter Sharing





Convolution Operation:Equivariant representations

- To say a function is equivariant means that if the input changes, the output changes in the same way.
  - Function f (x) is equivariant to a function g if
    - $f(g(x)) = g(f(x))$

- The form of parameter sharing used by CNNs causes each layer to be equivariant to translation
  - That is, if g is any function that translates the input, the convolution function is equivariant to g

Convolution Operation:Equivariant representations

- Implication: While processing time series data, convolution produces a timeline that shows when different features appeared (if an event is shifted in time in the input, the same representation will appear in the output)

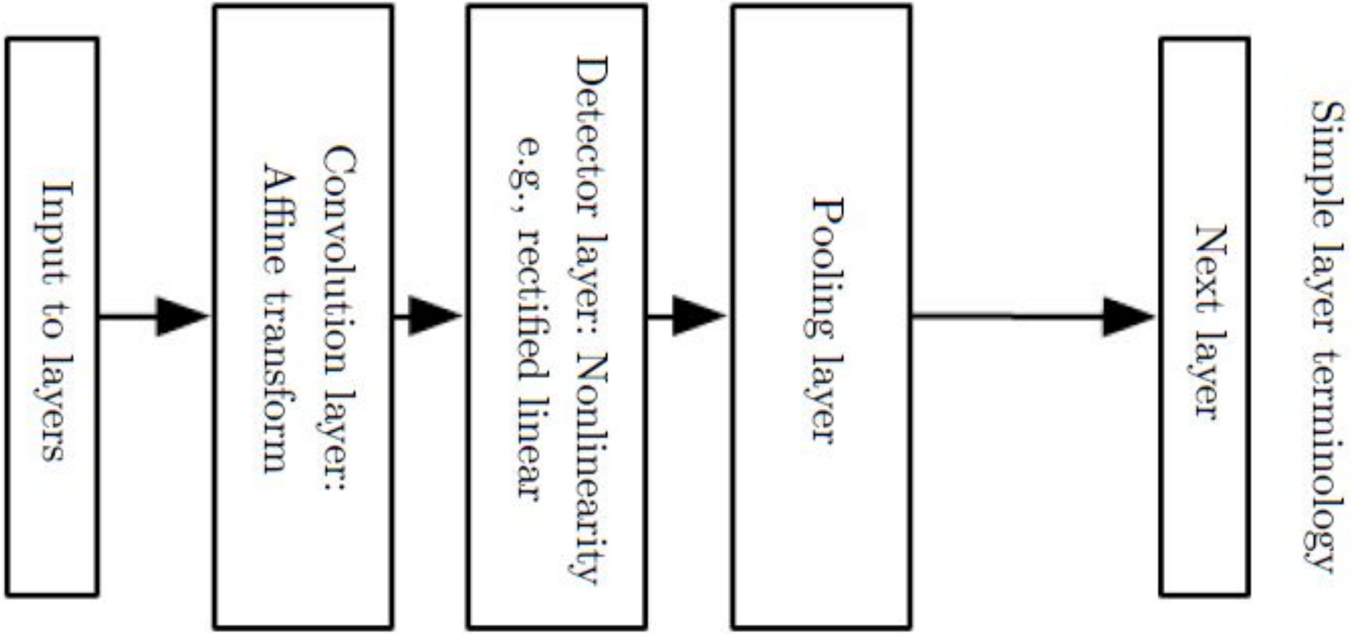
- Images: If we move an object in the image, its representation will move the same amount in the output

- This property is useful when we know some local function is useful everywhere (e.g. edge detectors)

- Convolution is not equivariant to other operations such as change in scale or rotation

Pooling

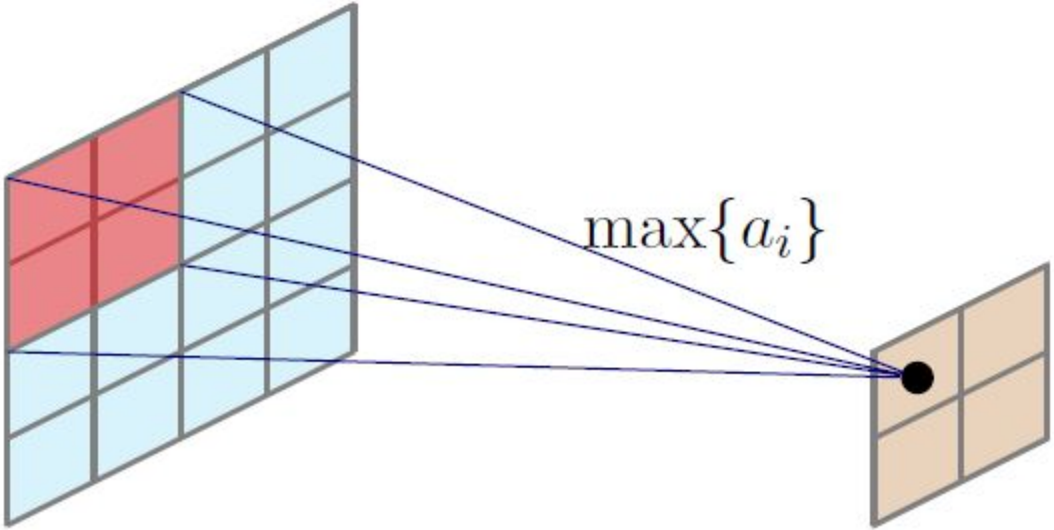
- Linear Activations [Convolution] -> Non-linear Activations [Detector] -> Pooling



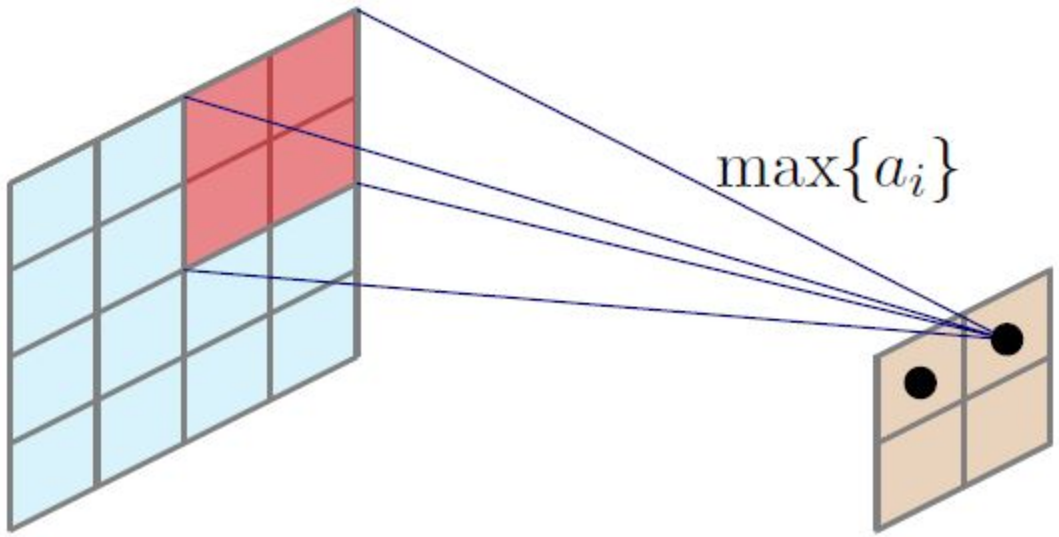
Pooling

- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.
  - Max pooling operation reports the maximum output within a rectangular neighborhood.
- Other popular pooling functions include
  - the average of a rectangular neighborhood,
  - the L2 norm of a rectangular neighborhood, or
  - a weighted average based on the distance from the central pixel
- In all cases, pooling helps to make the representation become approximately invariant to small translations of the input.

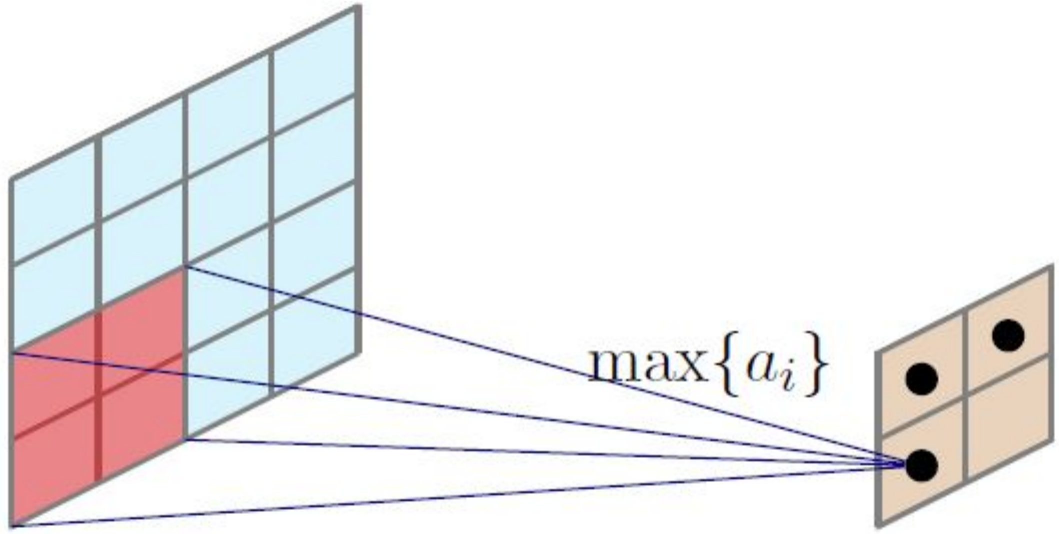
Pooling: Max Pooling



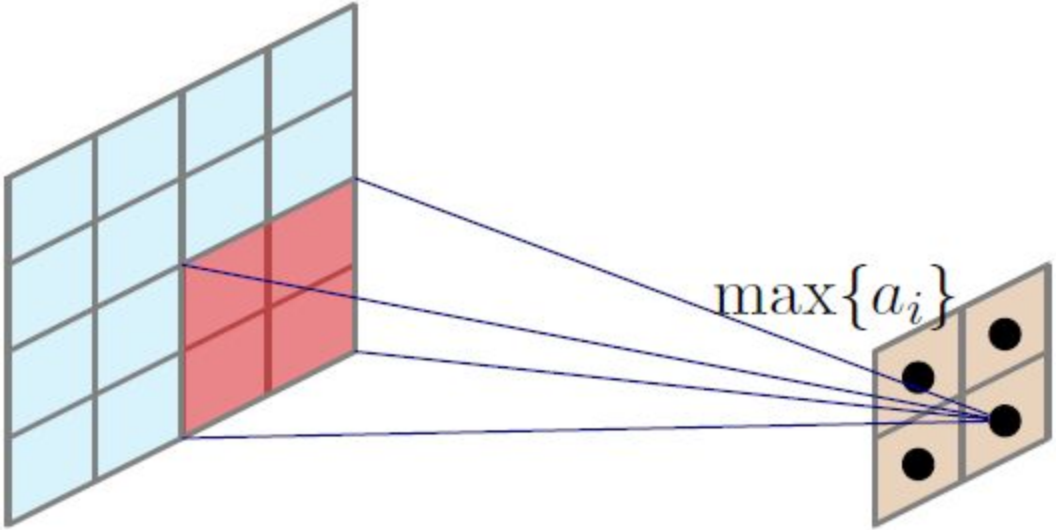
Pooling: Max Pooling



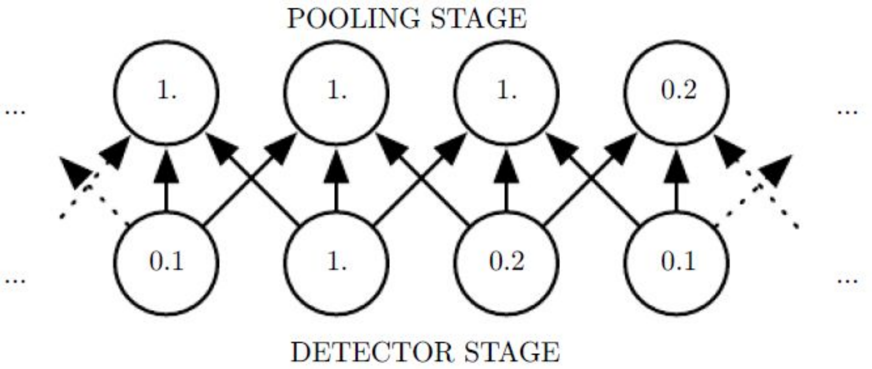
Pooling: Max Pooling

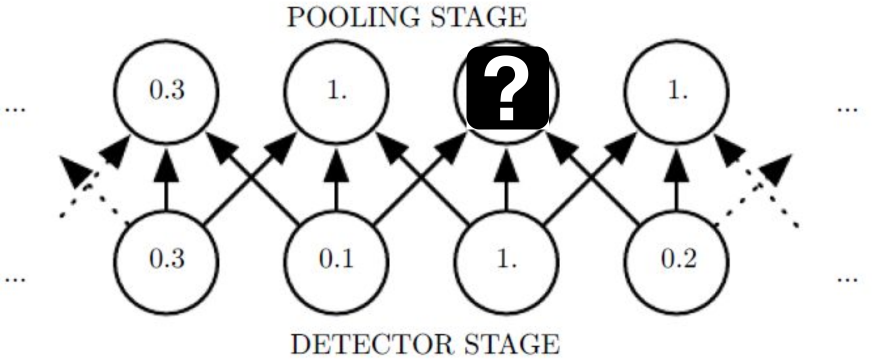


Pooling: Max Pooling



Pooling

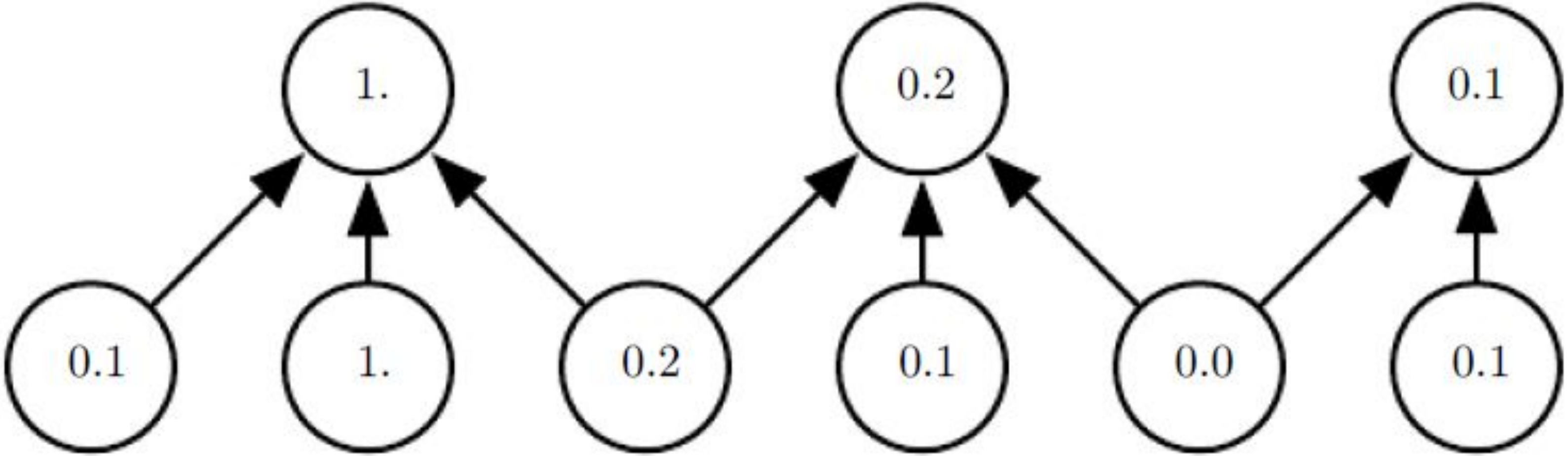

POOLING STAGE
1.
1.
1.
0.2
0.1
1.
0.2
0.1
DETECTOR STAGE


POOLING STAGE
0.3
1.
?
1.
0.3
0.1
1.
0.2
DETECTOR STAGE

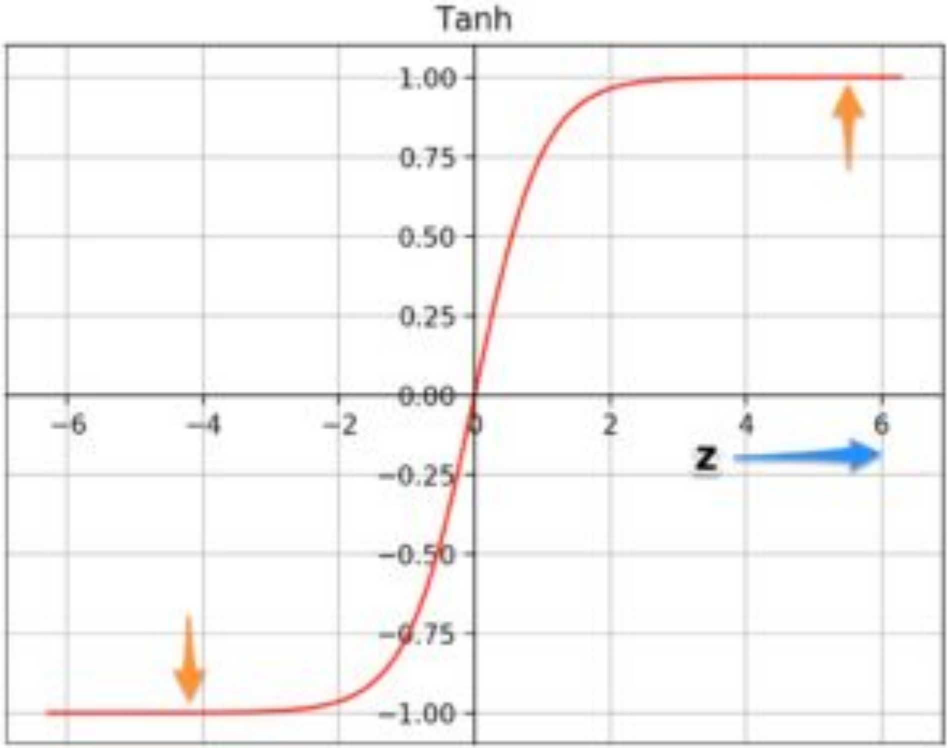
Pooling

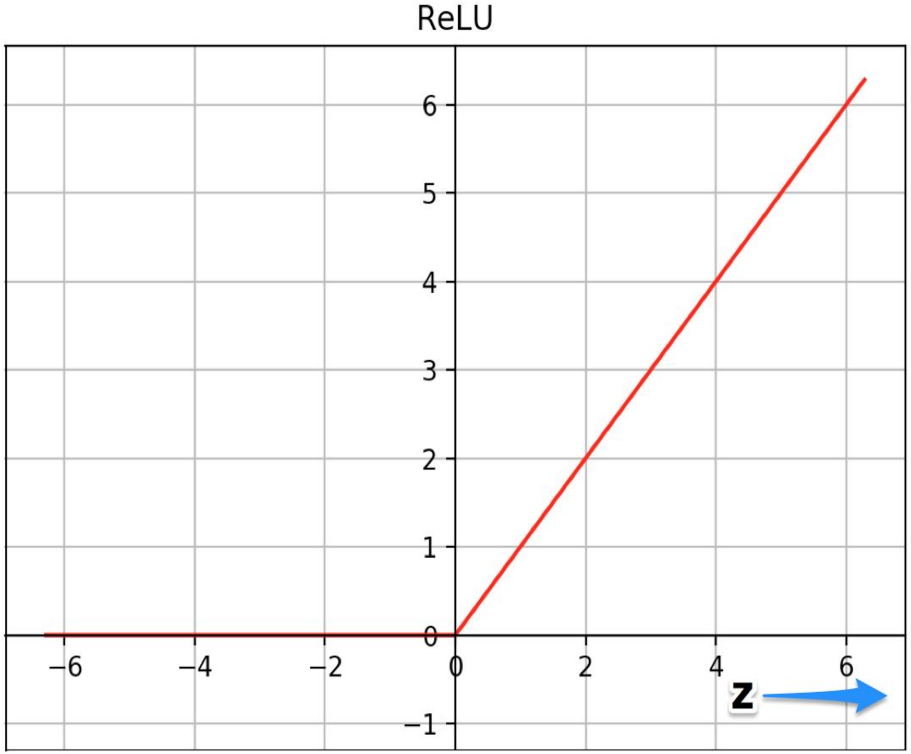
- Invariance to local translation can be a very useful property if we care more about *whether some feature is present* than *exactly where it is*
  - For example, when determining whether an image contains a face, we need not know the location of the eyes with pixel-perfect accuracy.
- In other contexts, it is more important to preserve the location of a feature.
  - For example, if we want to find a corner defined by two edges meeting at a specific orientation
- Since pooling is used for downsampling, it can be used to handle *inputs of varying sizes*

Pooling: Downsampling

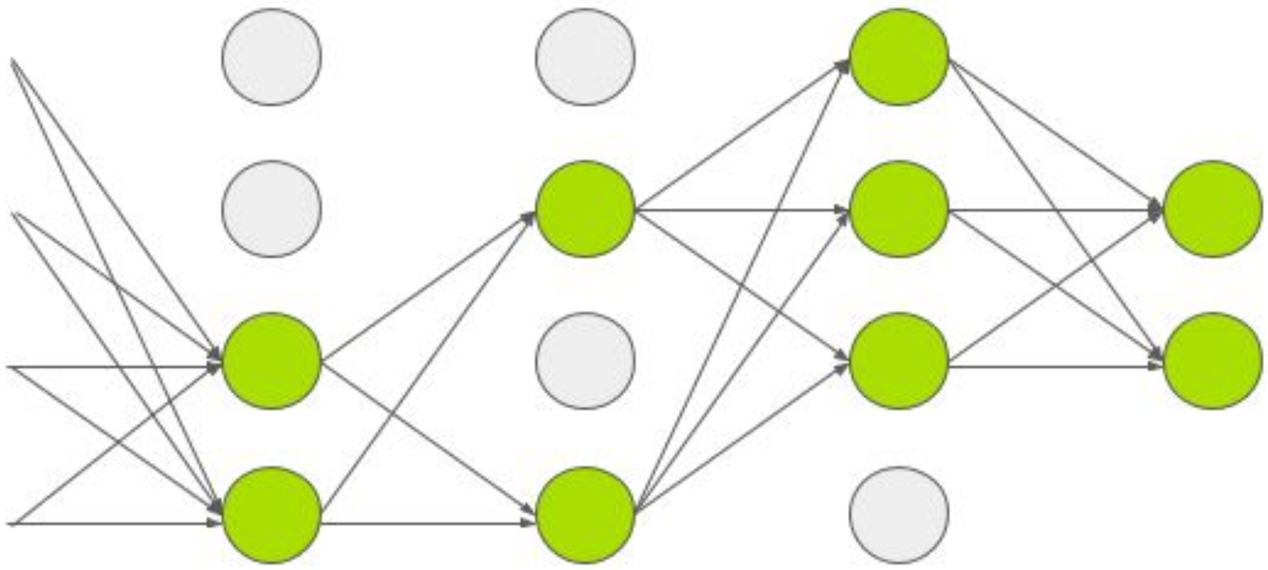


ReLu: Rectified Linear Unit

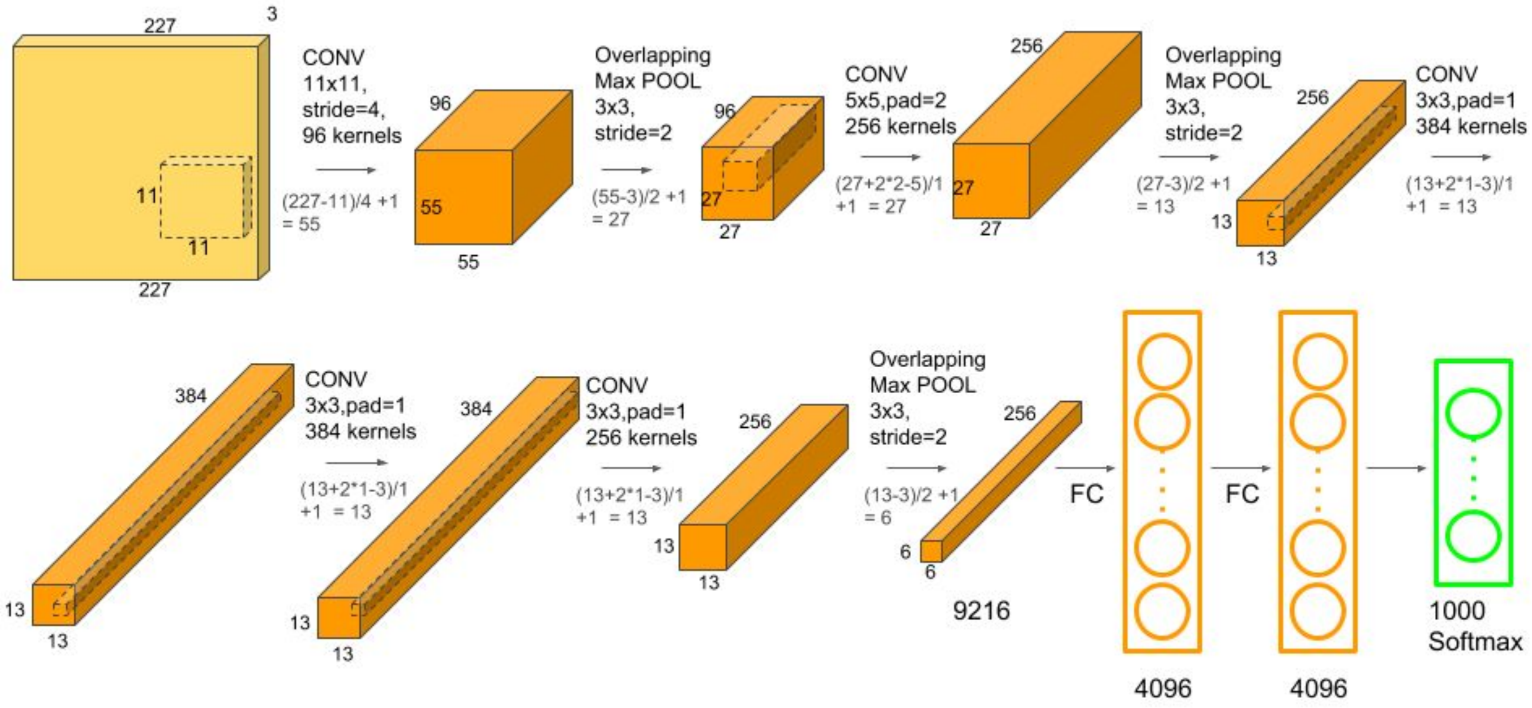




Dropout

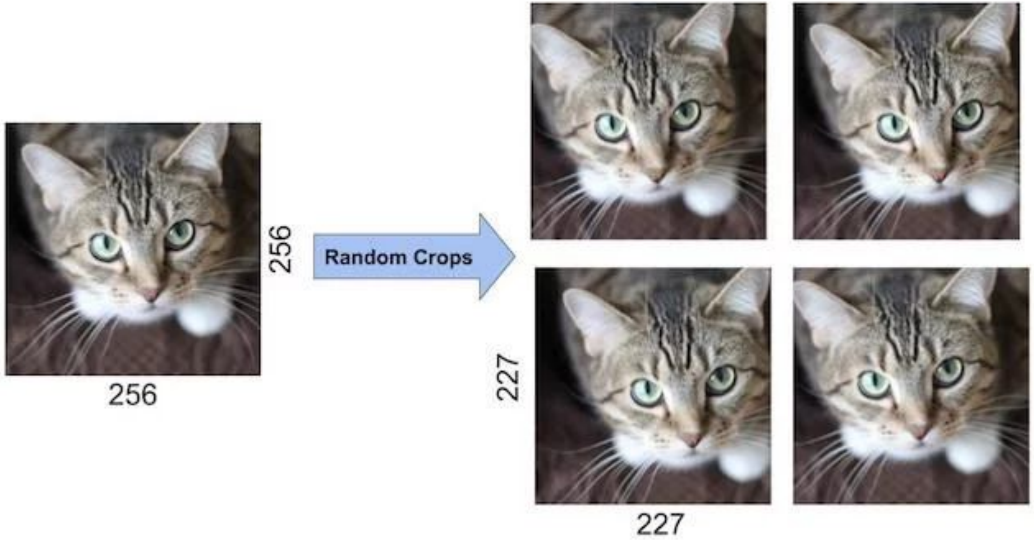


Walk Through: AlexNet



AlexNet

- The input to AlexNet is an RGB image of size 256×256
- Random crops of size 227×227 were generated from inside the 256×256 images to feed the first layer of AlexNet.
- Input: 227×227 x 3



AlexNet

- 5 Convolutional Layers and 3 Fully Connected Layers
- It has 60 million parameters and 650,000 neurons and took five to six days to train on two GTX 580 3GB GPUs!
- Input: 227×227 x 3
- The first Conv Layer of AlexNet contains 96 kernels of size 11x11x3.
  - Width and height of output: (227-11)/4 + 1 = 55
- Number of parameters in first layer?
  - (11x11x3 +1)*96 = 34,9444
- Number of Computations: 34,9444 x 55 x 55 = 10,57,05,600
- https://airtable.com/shrArXKRCau4KhAwZ/tbloN5WYjFYpKGUEt?blocks=hide

References

1. Chapter 9, Deep Learning: Ian Goodfellow, Yoshua Bengio, Aaron Courville (http://www.deeplearningbook.org/)
2. https://neurdiness.wordpress.com/2018/05/17/deep-convolutional-neural-networks-as-models-of-the-visual-system-qa/
3. https://www.rctn.org/bruno/public/papers/Fukushima1980.pdf
4. https://www.youtube.com/watch?v=2-Ol7ZB0MmU
5. https://www.youtube.com/watch?v=ZOXOwYUVCqw