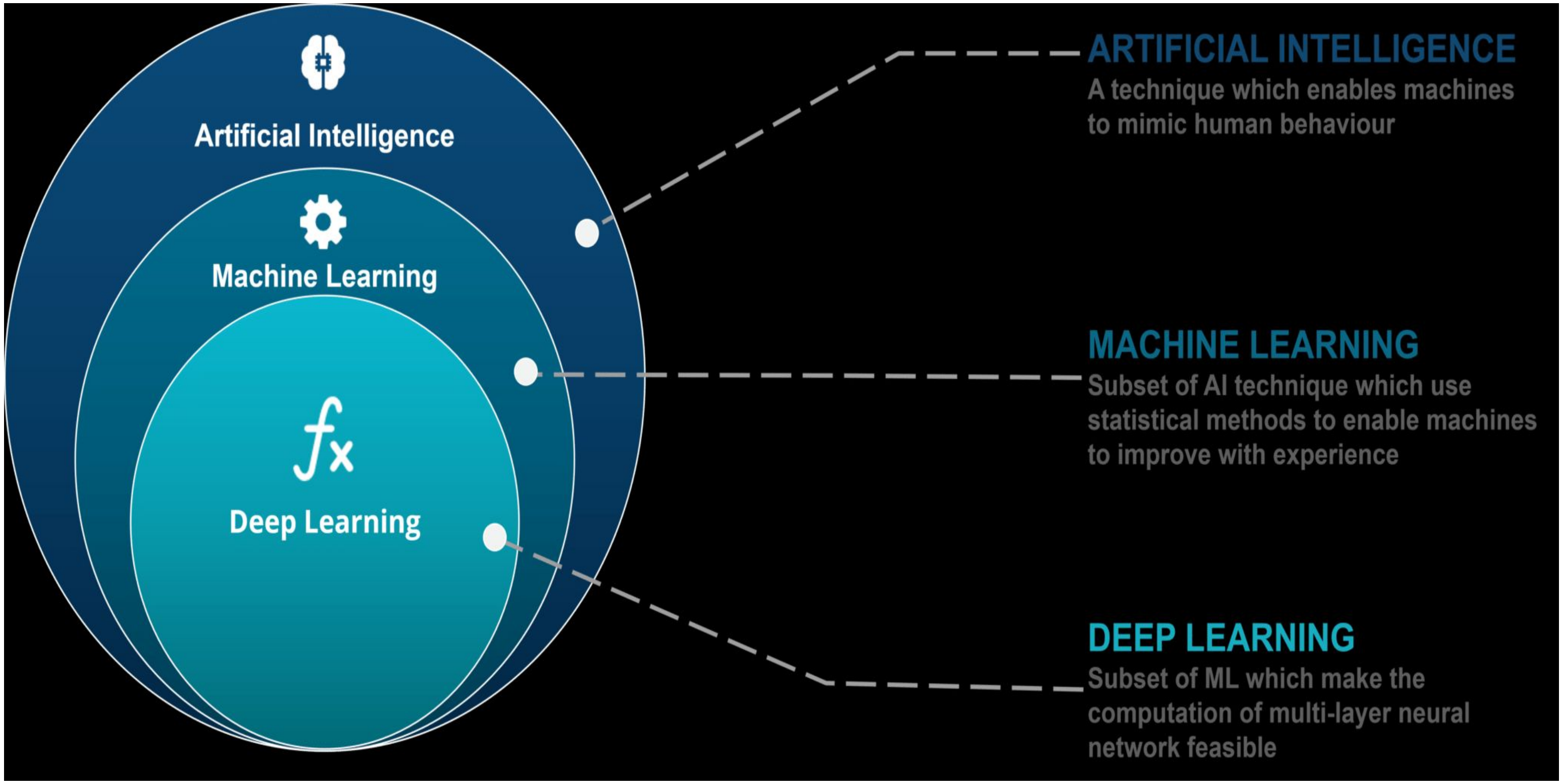
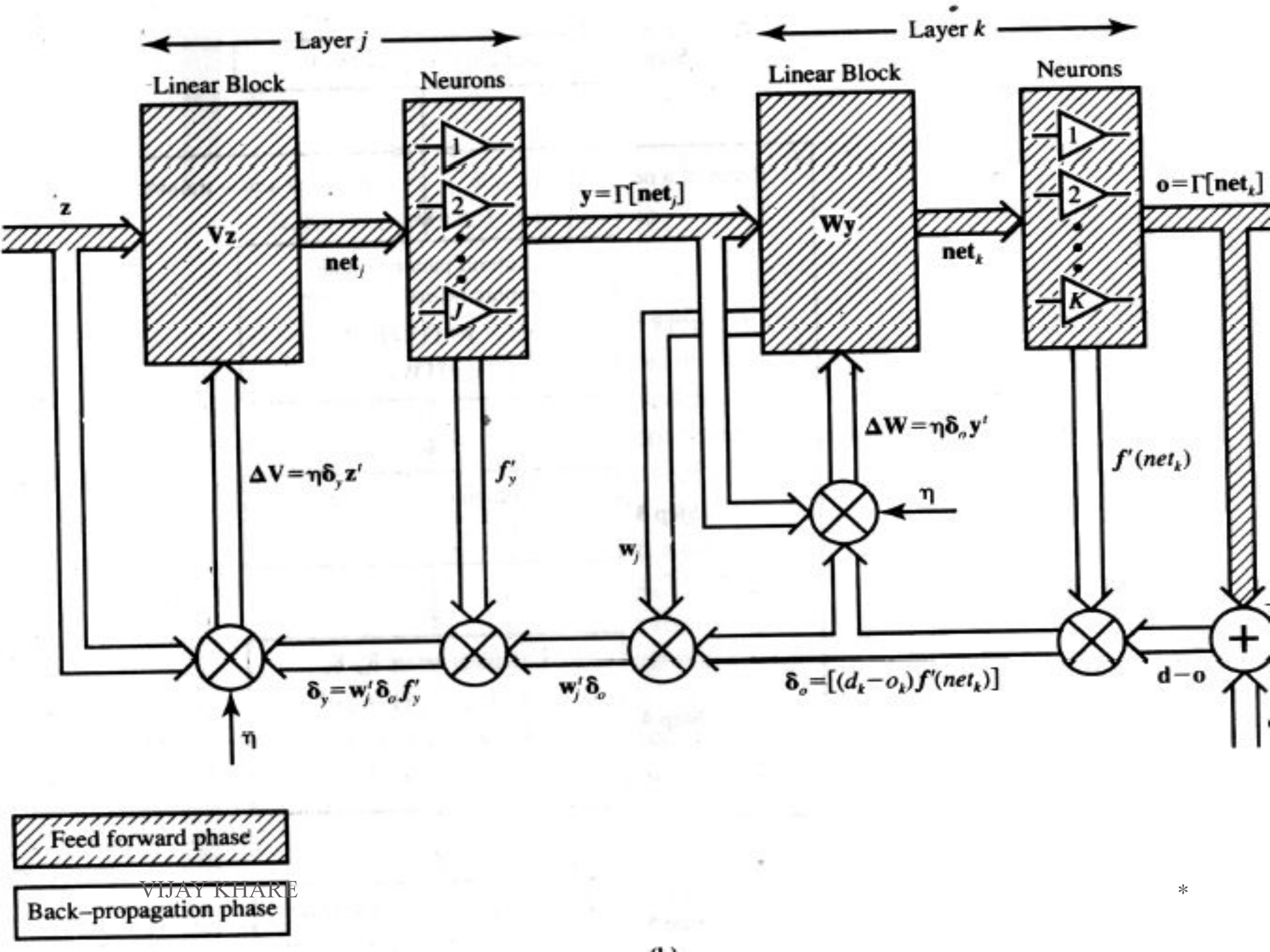
Deep learning for biomedical Application

Vijay Khare

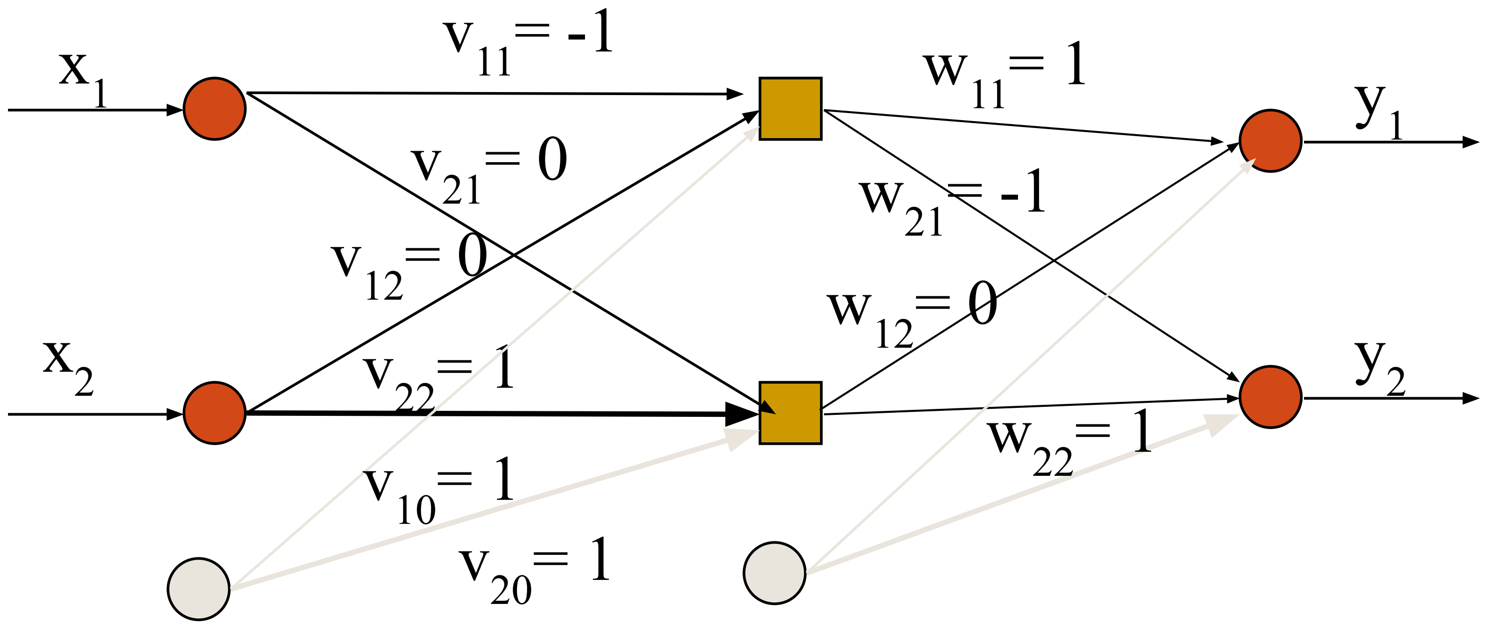
Artificial Intelligence is the broader umbrella under which Machine Learning and Deep Learning come. And you can also see in the diagram that even deep learning is a subset of Machine Learning. So all three of them AI, machine learning and deep learning are just the subsets of each other. So let us move on and understand how exactly they are different from each other


Artificial Intelligence
Machine Learning
Deep Learning
ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour
MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience
DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

This is an era of Artificial intelligence. Neural networks form the base of deep learning, which is a subfield of machine learning, where the structure of the human brain inspires the algorithms. Neural networks take input data, train themselves to recognize patterns found in the data, and then predict the output for a new set of similar data. Therefore, a neural network can be thought of as the functional unit of deep learning, which mimics the behavior of the human brain to solve complex data-driven problems.


Layer j
Layer k
Linear Block
Neurons
Linear Block
Neurons
$\mathbf{z}$
$\mathbf{Vz}$
$\mathbf{net}_j$
1
2
J
$\mathbf{y} = \Gamma[\mathbf{net}_j]$
$\mathbf{Wy}$
$\mathbf{net}_k$
1
2
K
$\mathbf{o} = \Gamma[\mathbf{net}_k]$
$\Delta \mathbf{V} = \eta \boldsymbol{\delta}_y \mathbf{z}^t$
f'_y
$\mathbf{w}_j$
$\Delta \mathbf{W} = \eta \boldsymbol{\delta}_o \mathbf{y}^t$
$f'(net_k)$
η
η
$\boldsymbol{\delta}_y = \mathbf{w}_j^t \boldsymbol{\delta}_o f'_y$
$\mathbf{w}_j^t \boldsymbol{\delta}_o$
$\boldsymbol{\delta}_o = [(d_k - o_k) f'(net_k)]$
$\mathbf{d} - \mathbf{o}$
+
▨ Feed forward phase
Back–propagation phase

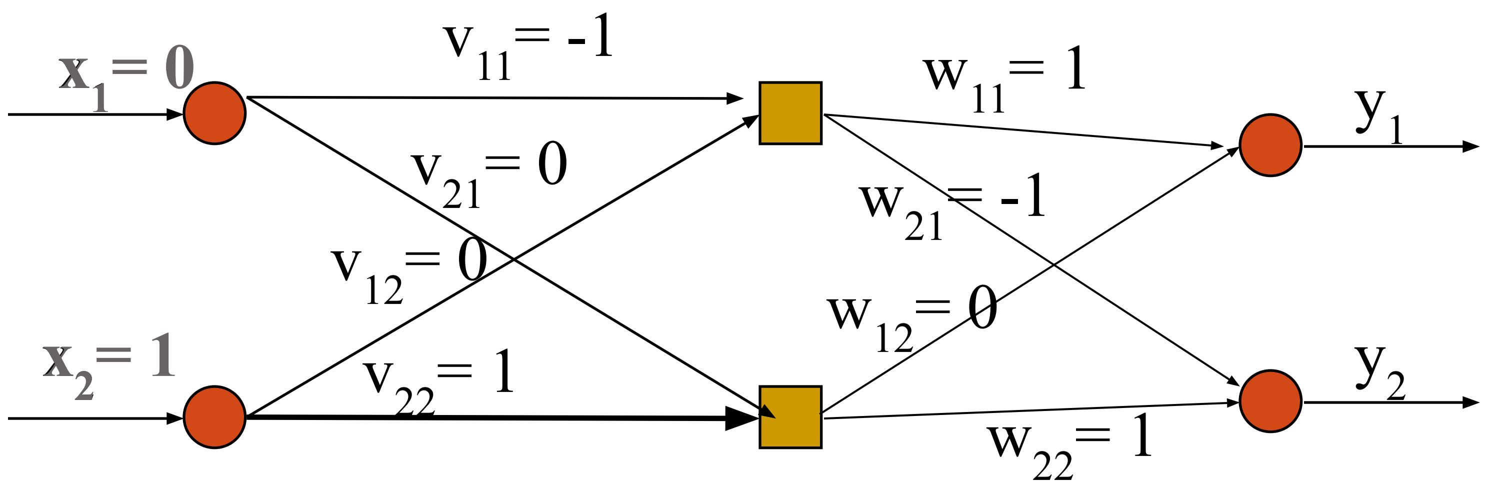
Once weight changes are computed for all units, weights are updated at the same time (bias included as weights here). An example:



Use identity activation function (ie g(a) = a)

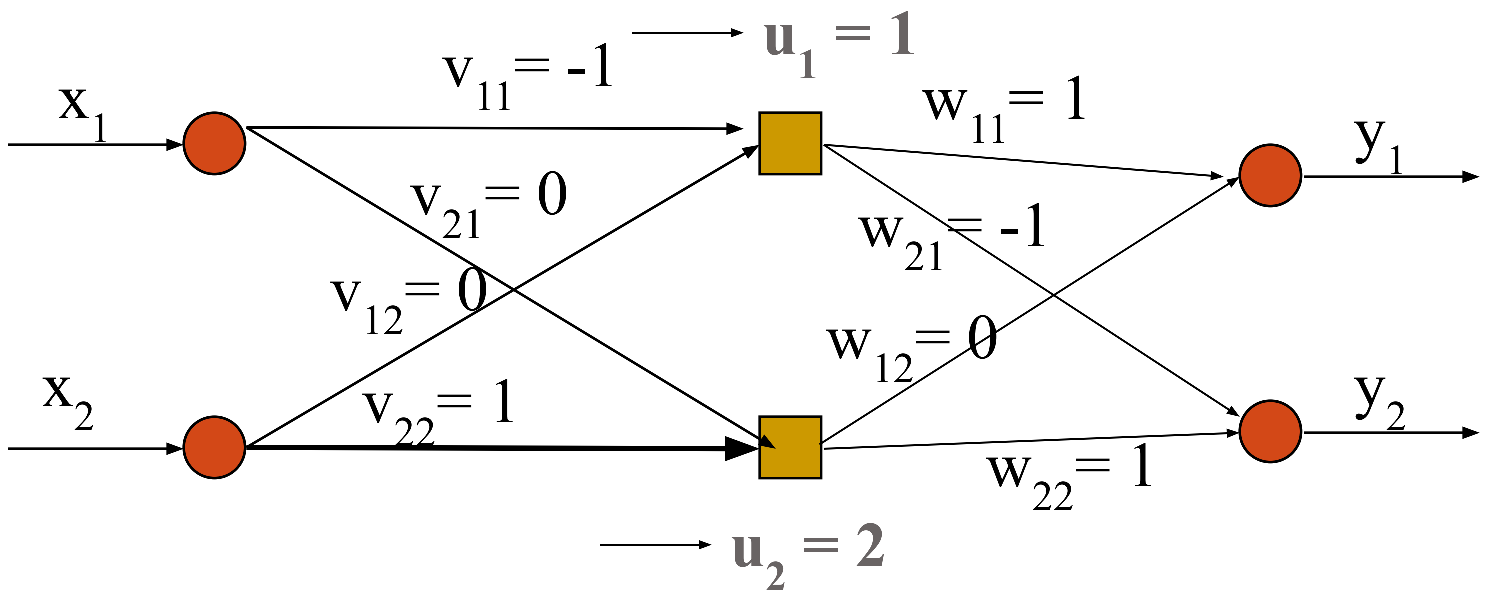
All biases set to 1. Will not draw them for clarity.

Learning rate $\eta = 0.1$



Have input [0 1] with target [1 0].

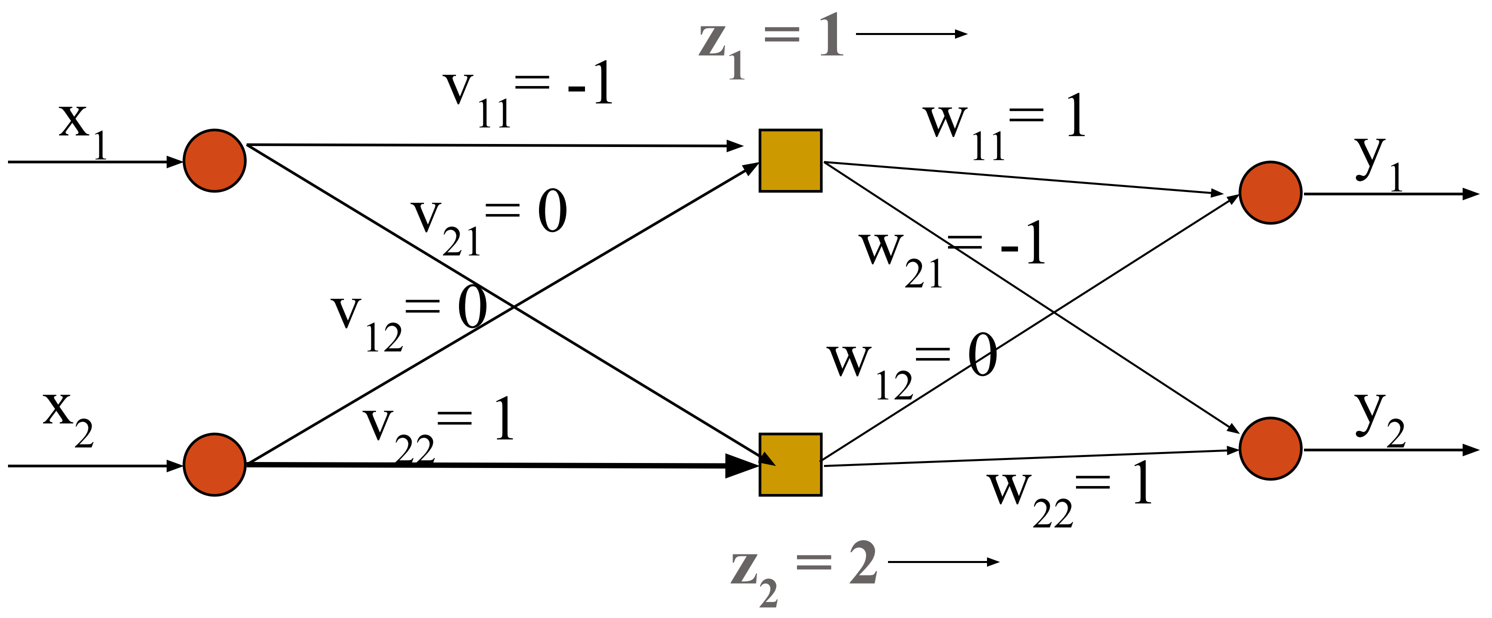
Forward pass. Calculate 1st layer activations:



$u_1 = -1x0 + 0x1 +1 = 1$

$u_2 = 0x0 + 1x1 +1 = 2$

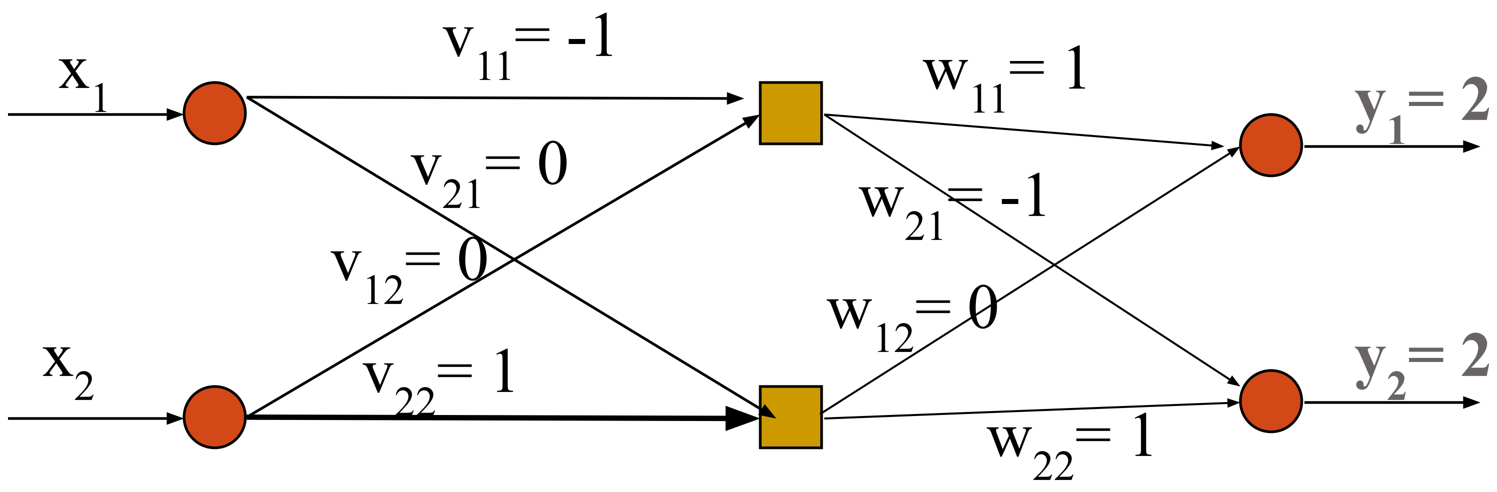
Calculate first layer outputs by passing activations thru activation functions



$z_1 = g(u_1) = 1$

$z_2 = g(u_2) = 2$

Calculate 2nd layer outputs (weighted sum thru activation functions):

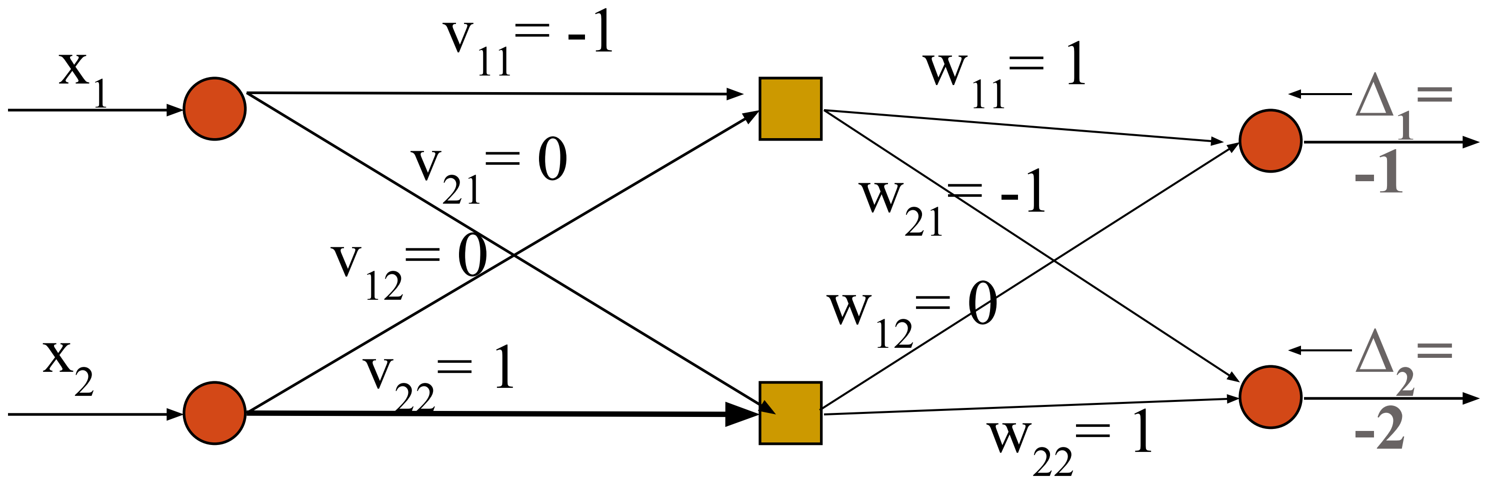


$y_1 = a_1 = 1x1 + 0x2 +1 = 2$

$y_2 = a_2 = -1x1 + 1x2 +1 = 2$

Backward pass:

$$w_{ij}(t+1) - w_{ij}(t) = \eta\Delta_i(t)z_j(t)$$
$$= \eta(d_i(t) - y_i(t))g'(a_i(t))z_j(t)$$



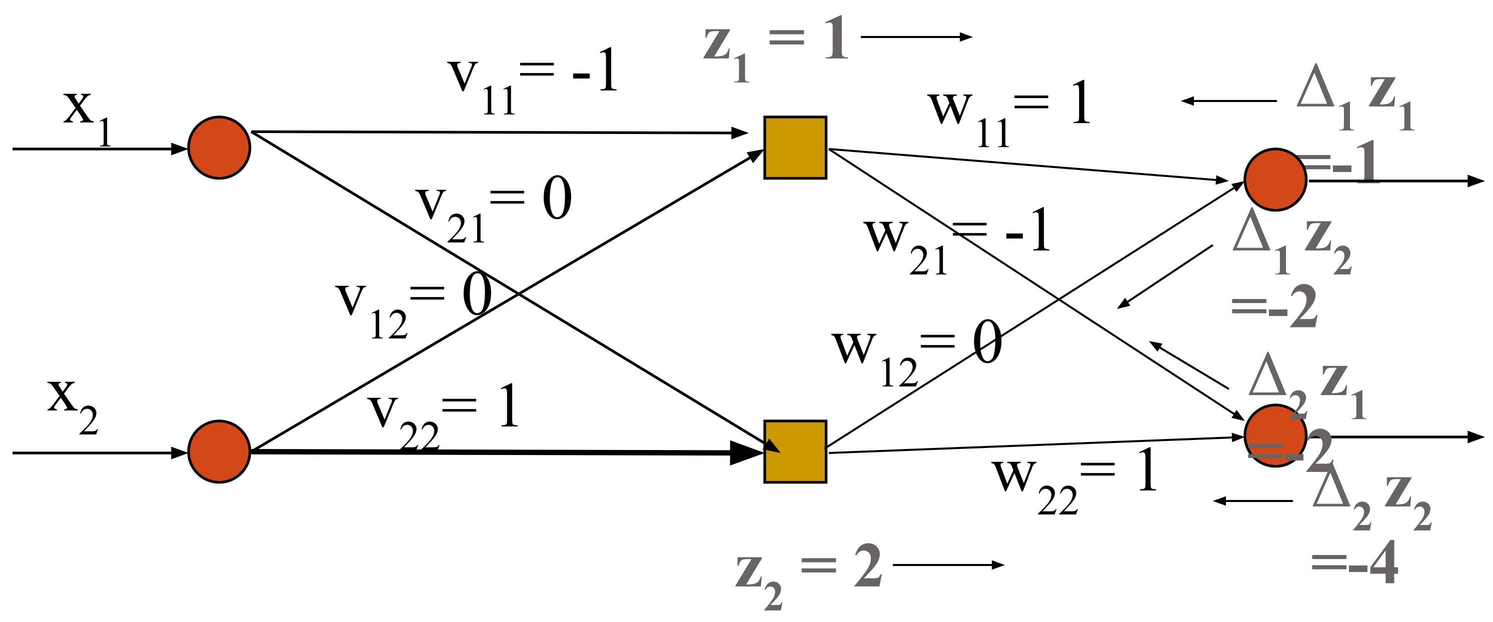
Target =[1, 0] so $d_1 = 1$ and $d_2 = 0$

So:

$\Delta_1 = (d_1 - y_1) = 1 – 2 = -1$

$\Delta_2 = (d_2 - y_2) = 0 – 2 = -2$

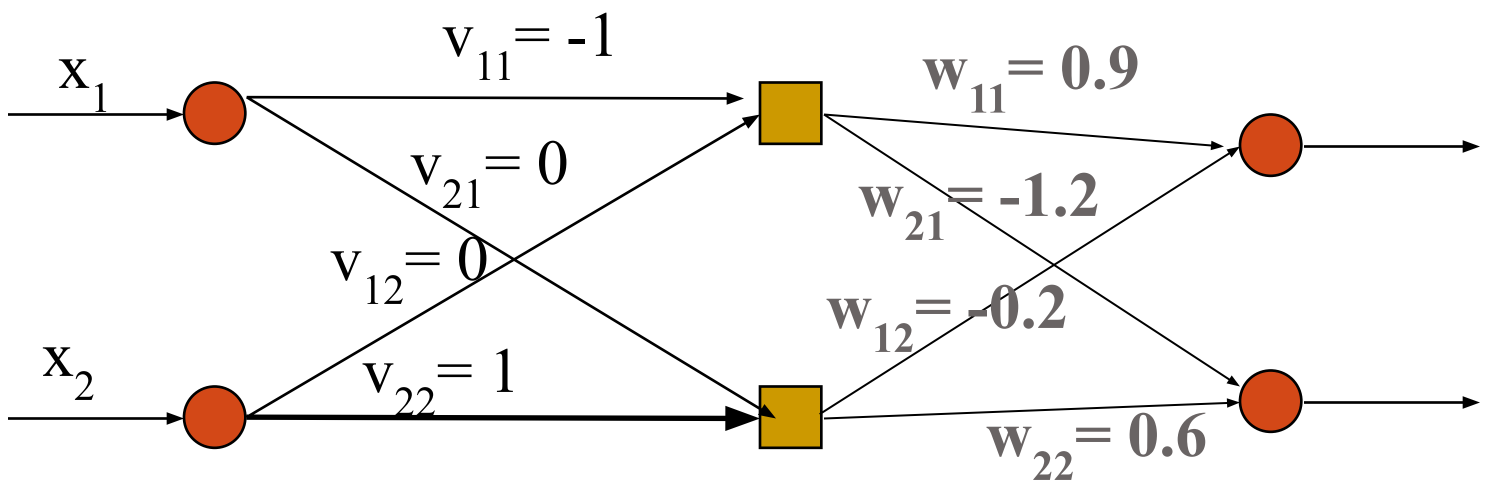
Calculate weight changes for 1st layer (cf perceptron learning):



$$w_{ij}(t+1) - w_{ij}(t) = \eta \Delta_i(t) z_j(t)$$

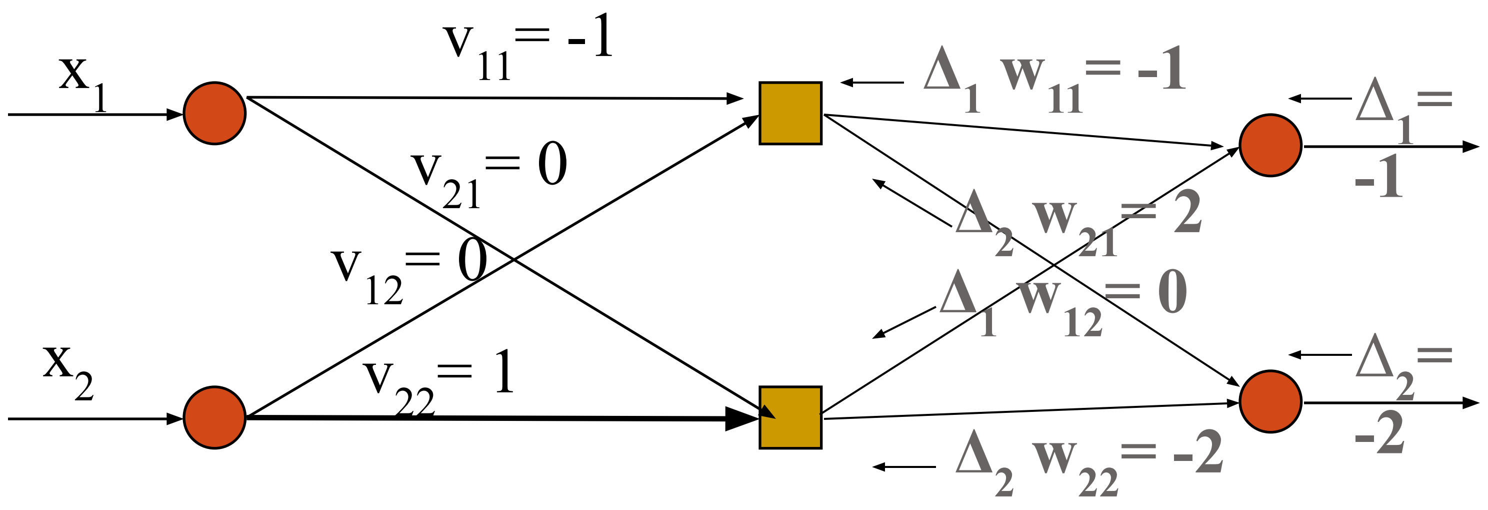
Weight changes will be:

$$w_{ij}(t+1) - w_{ij}(t) = \eta \Delta_i(t) z_j(t)$$

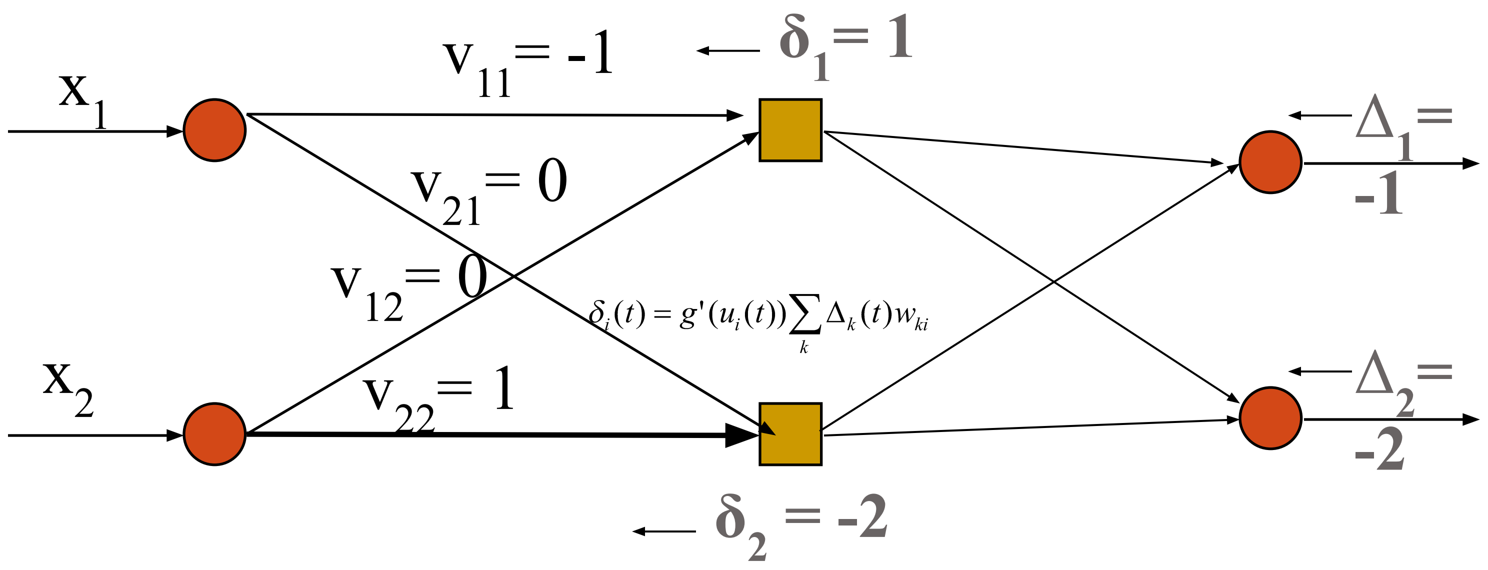


But first must calculate δ’s:

$$\delta_i(t) = g'(u_i(t))\sum_k \Delta_k(t) w_{ki}$$



Δ's propagate back

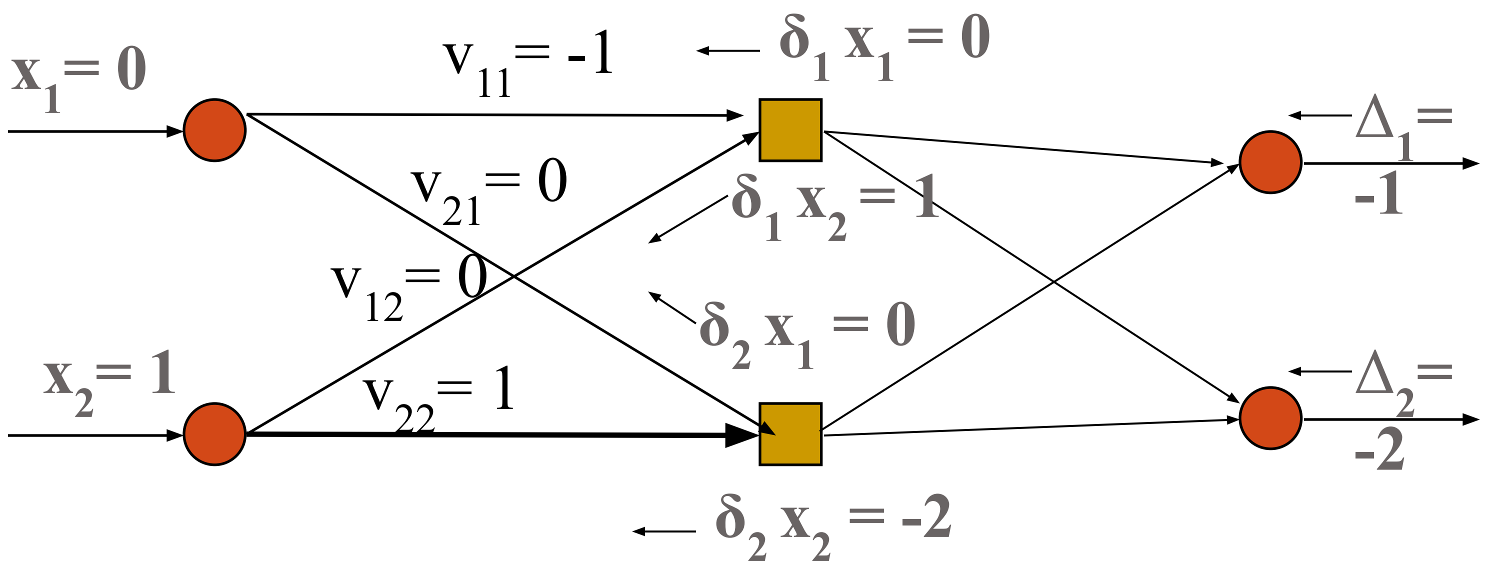


$\delta_1 = -1 + 2 = 1$

$\delta_2 = 0 – 2 = -2$

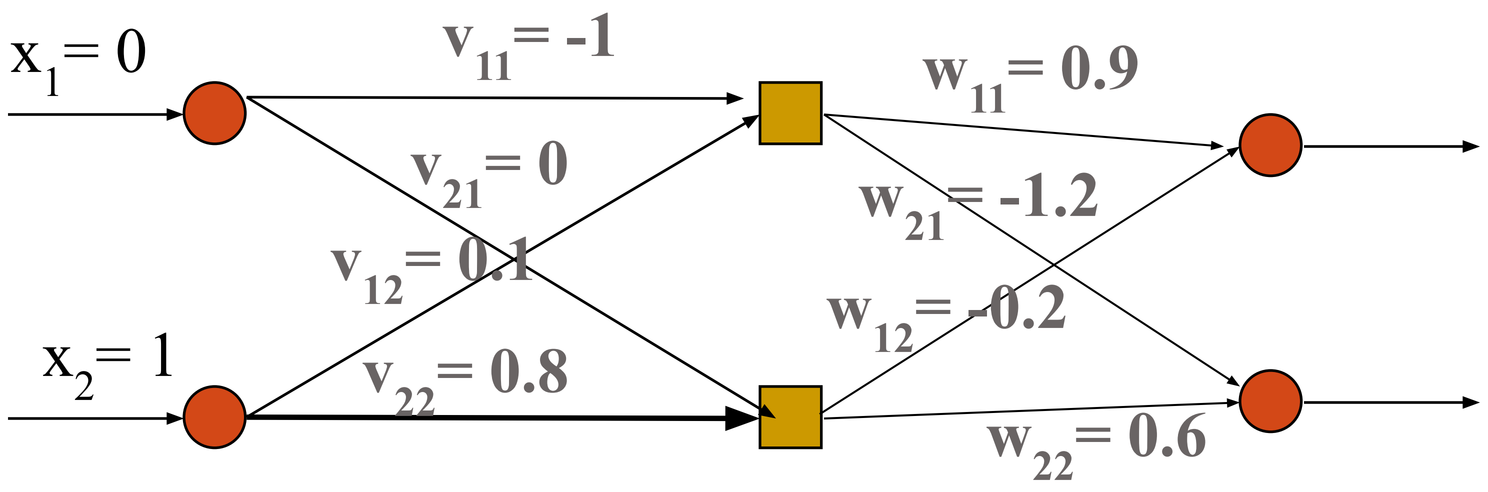
And are multiplied by inputs

$$v_{ij}(t+1) - v_{ij}(t) = \eta \delta_i(t) x_j(t)$$



Finally change weights:

$$v_{ij}(t+1) - v_{ij}(t) = \eta \delta_i(t) x_j(t)$$

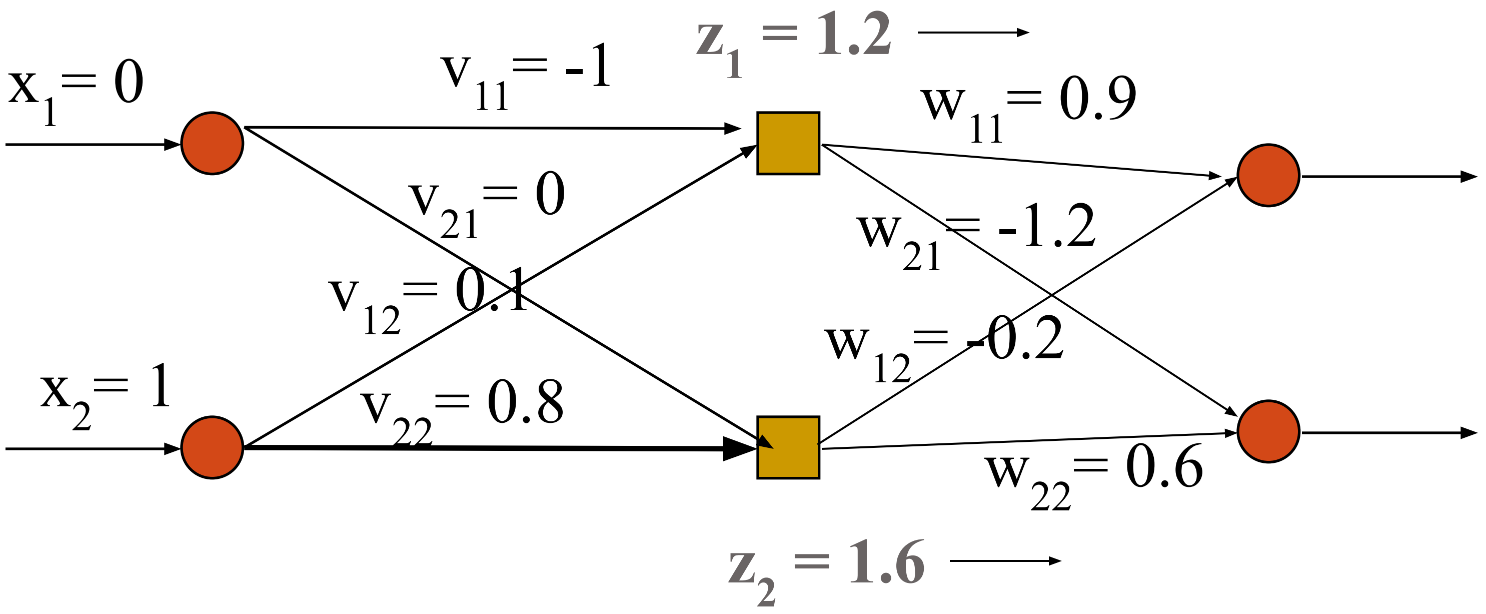


Note that the weights multiplied by the zero input are unchanged as they do not contribute to the error

We have also changed biases (not shown)

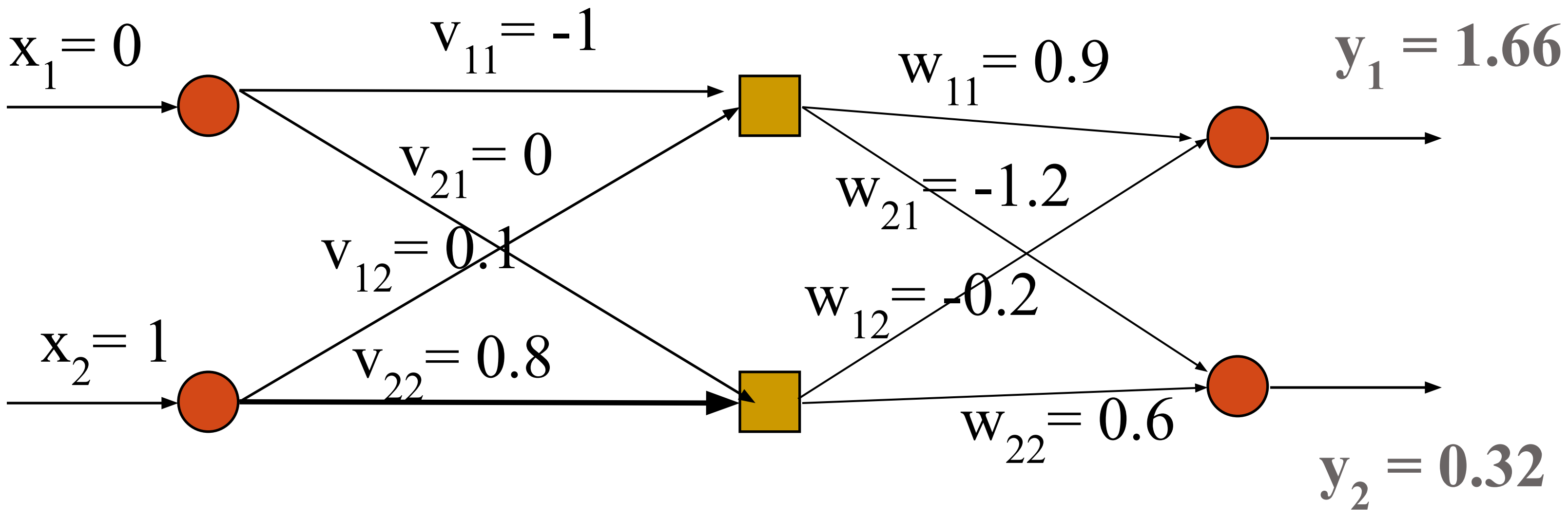
Now go forward again (would normally use a new input vector):

$$v_{ij}(t+1) - v_{ij}(t) = \eta \delta_i(t) x_j(t)$$



Now go forward again (would normally use a new input vector):

$$v_{ij}(t+1) - v_{ij}(t) = \eta \delta_i(t) x_j(t)$$



Outputs now closer to target value [1, 0]

Advantages

- Massive parallel computing system.
- Ability learning : so it is generalized mean data which is not encounter during training for that it give reasonable output.
- Nonlinearity : special properties while solving the signal generated with physical mechanism.
- input- out put mapping.
- Fault-tolerant/graceful- When Implemented in hardware degrading in adverse operating condition.
- Adaptively : it adapt the change or easily train with minor change.

Applications

- **a. Classification of data:**
- Based on a set of data, our trained neural network predicts whether it is a dog or a cat?
- **b. Anomaly detection:**
- Given the details about transactions of a person, it can say that whether the transaction is fraud or not.
- **c. Speech recognition:**
- We can train our neural network to recognize speech patterns. Example: Siri, Alexa, Google assistant.

d. Audio generation:

Given the inputs as audio files, it can generate new music based on various factors like genre, singer, and others.

- **e. Time series analysis:**

  A well trained neural network can predict the stock price.

- **f. Spell checking:**

  We can train a neural network that detects misspelled spellings and can also suggest a similar meaning for words. Example: Grammarly

- **g. Character recognition:**

  A well trained neural network can detect handwritten characters.

- **h. Machine translation:**

  We can develop a neural network that translates one language into another language.
- **i. Image processing:**

  We can train a neural network to process an image and extract pieces of information from it.

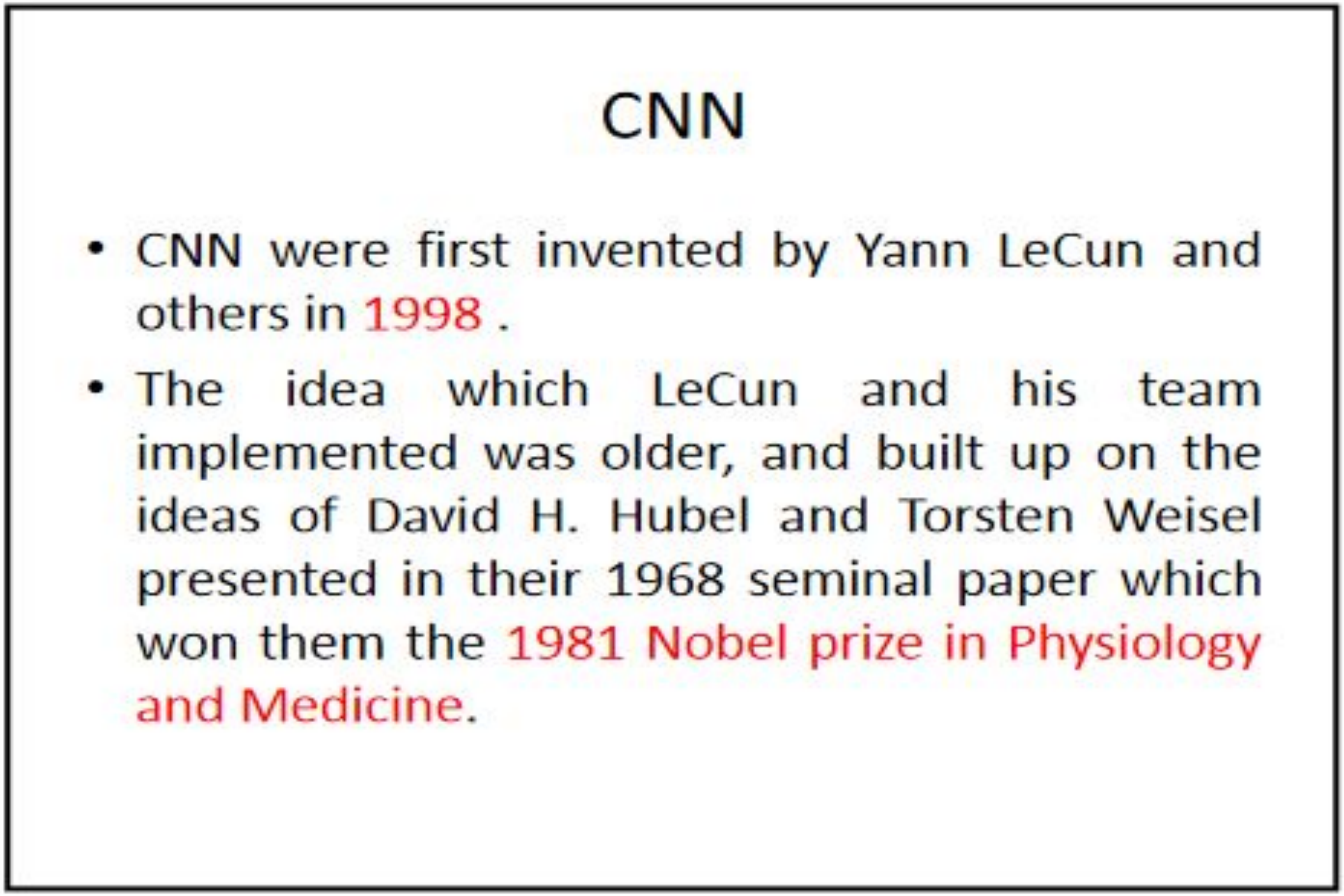
- **h. Machine translation:**

  We can develop a neural network that translates one language into another language.
- **i. Image processing:**

  We can train a neural network to process an image and extract pieces of information from it.

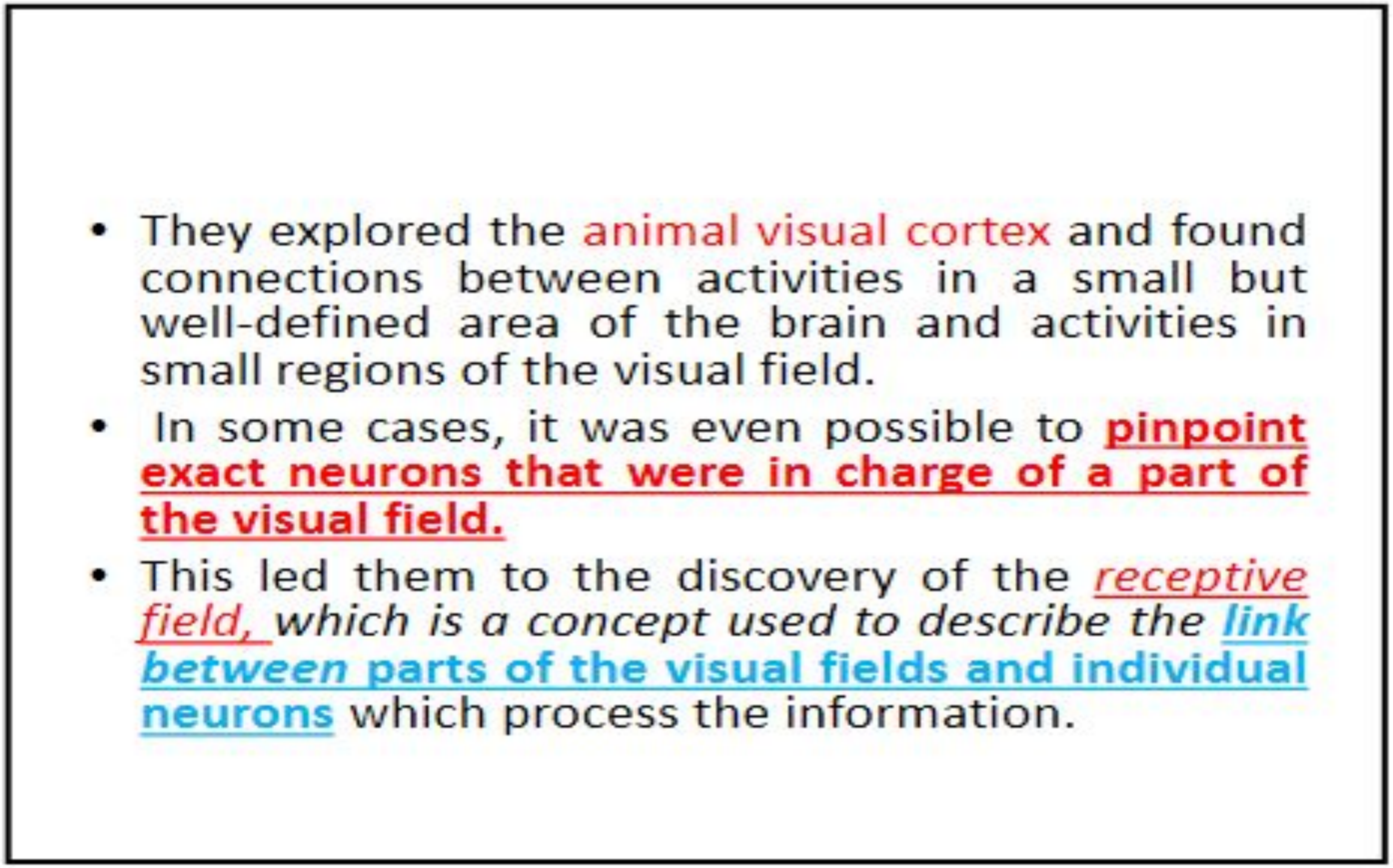
Convolution Neural Network

- Neural networks on very large images increases the computation and memory cost. To combat this obstacle, convolutional neural networks help us to bring down these factors and generate better results.



CNN

- CNN were first invented by Yann LeCun and others in 1998 .
- The idea which LeCun and his team implemented was older, and built up on the ideas of David H. Hubel and Torsten Weisel presented in their 1968 seminal paper which won them the 1981 Nobel prize in Physiology and Medicine.



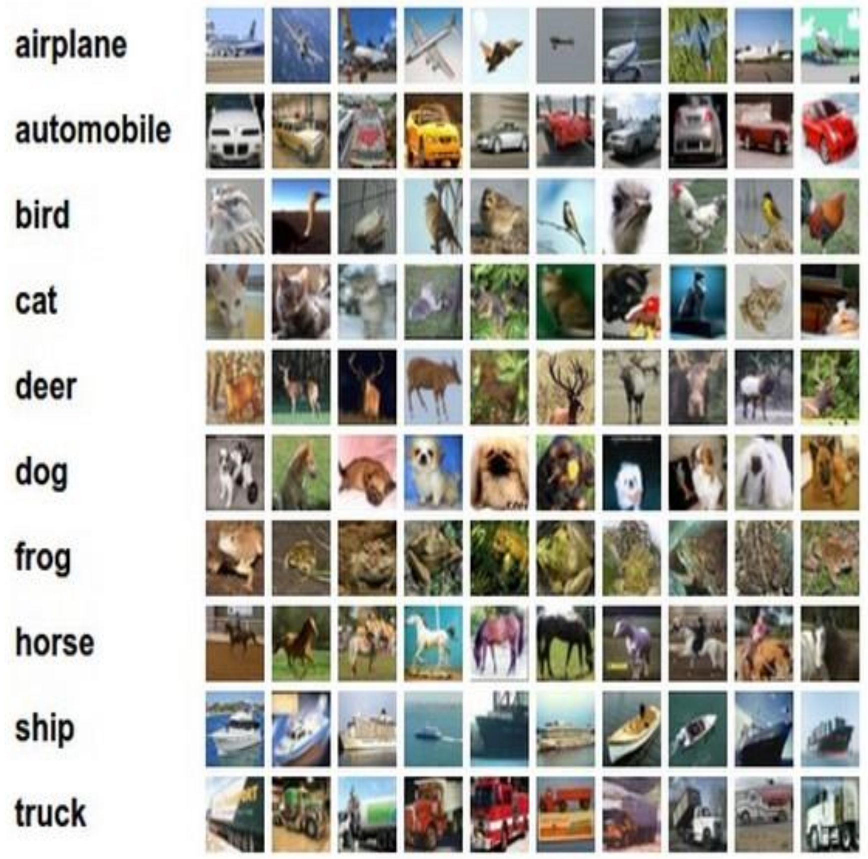
- They explored the animal visual cortex and found connections between activities in a small but well-defined area of the brain and activities in small regions of the visual field.
- In some cases, it was even possible to **pinpoint exact neurons that were in charge of a part of the visual field.**
- This led them to the discovery of the *receptive field, which is a concept used to describe the* ***link between*** **parts of the visual fields and individual neurons** which process the information.

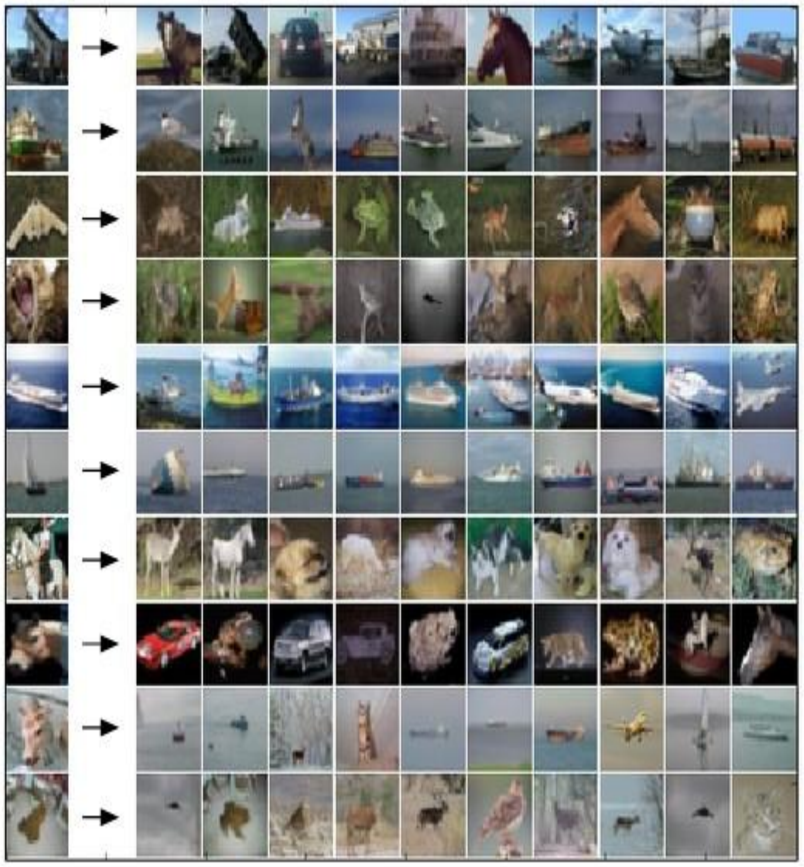
CNN

- We saw how using deep neural networks on very large images increases the computation and memory cost. To combat this obstacle, we will see how convolutions and convolutional neural networks help us to bring down these factors and generate better results .Despite its simplicity, has a large variety of practical applications. Some of the others computer vision problems
- Image classification
- Object detection
- Neural style transfer

CNN

- **Image classification dataset: CIFAR-10.** One popular toy image classification dataset is the <u>CIFAR-10 dataset</u> One popular toy image classification dataset is the CIFAR-10 dataset(<u>Canadian Institute For Advanced Research</u>)).
- This dataset consists of 60,000 tiny images that are 32 pixels high and 32 wide. Each image is labeled with one of 10 classes (for example *"airplane, automobile, bird, etc"*).
- These 60,000 images are partitioned into a training set of 50,000 images and a test set of 10,000 images. In the image below you can see 10 random example images from each one of


airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck



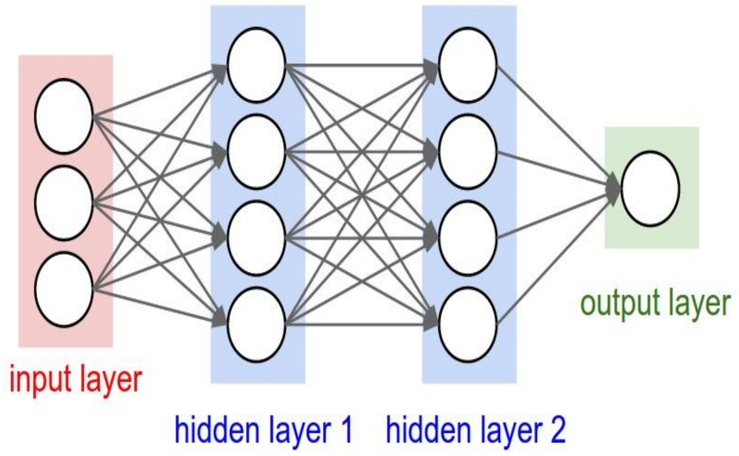
Regular Neural Nets don't scale well to full images.

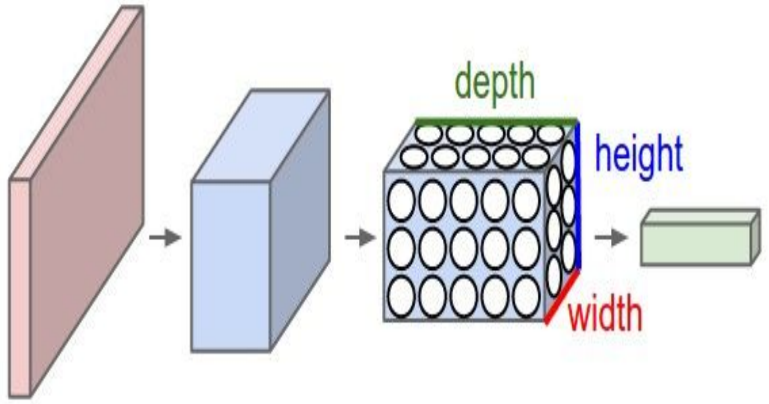
- In CIFAR-10, images are only of size 32x32x3 (32 wide, 32 high, 3 color channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 32*32*3 = 3072 weights.
- This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images. For example, an image of more respectable size, e.g. 200x200x3, would lead to neurons that have 200*200*3 = 120,000 weights.
- Moreover, we would almost certainly want to have several such neurons, so the parameters would add up quickly! Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to over fitting.

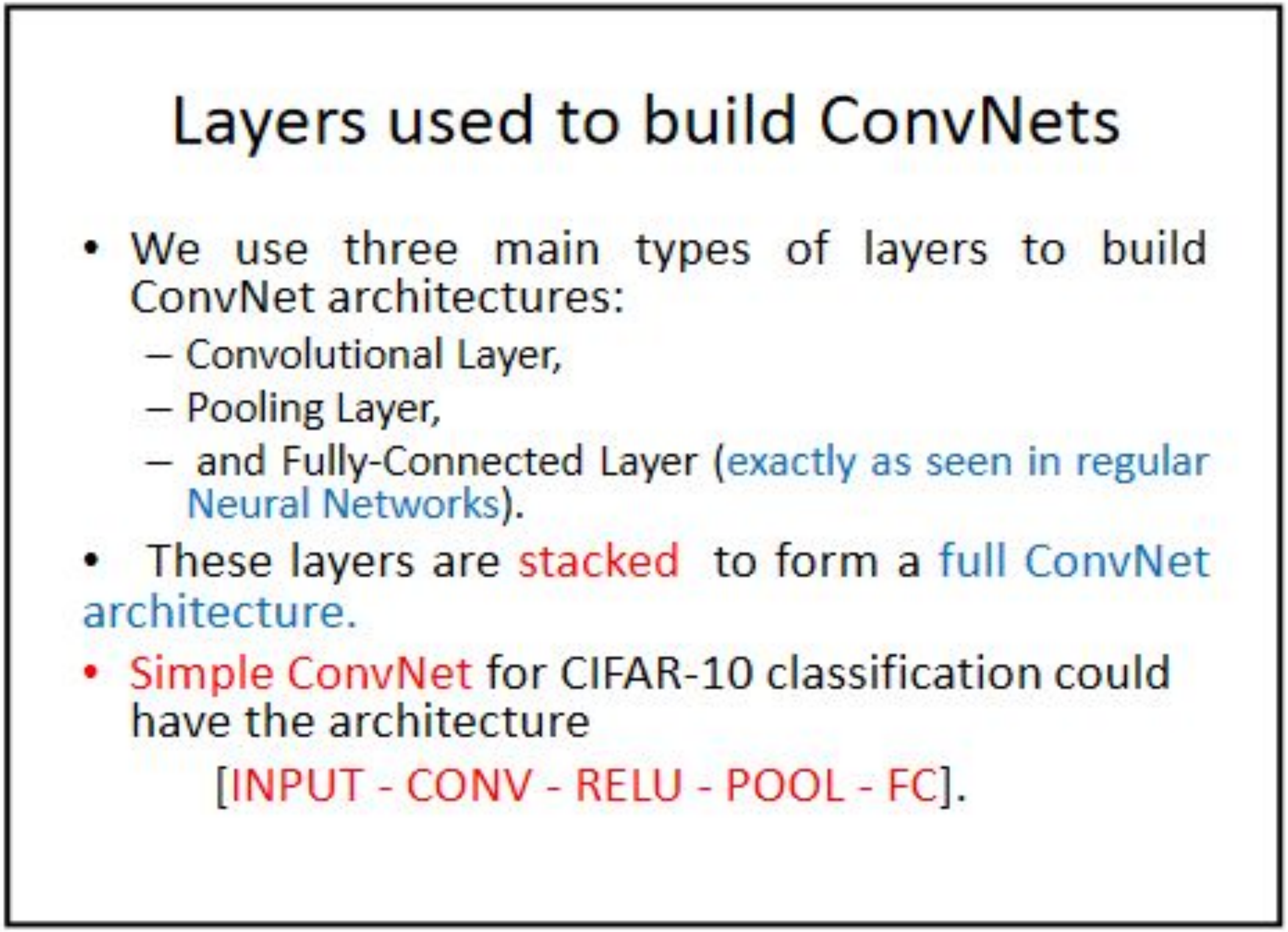
CNN

- Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network,

A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).





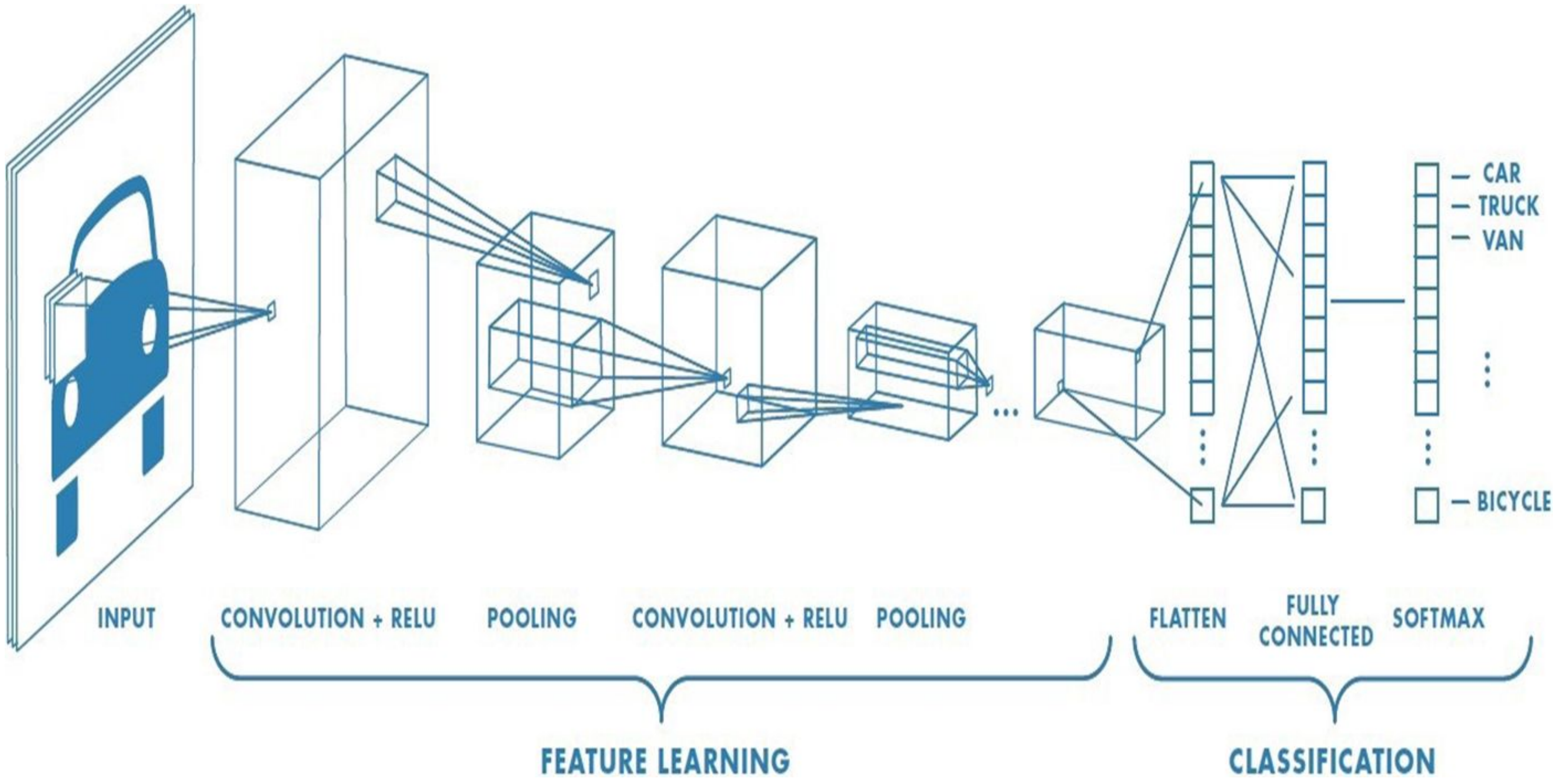


Layers used to build ConvNets

- We use three main types of layers to build ConvNet architectures:
  - Convolutional Layer,
  - Pooling Layer,
  - and Fully-Connected Layer (exactly as seen in regular Neural Networks).
- These layers are stacked to form a full ConvNet architecture.
- Simple ConvNet for CIFAR-10 classification could have the architecture

  [INPUT - CONV - RELU - POOL - FC].

CNN

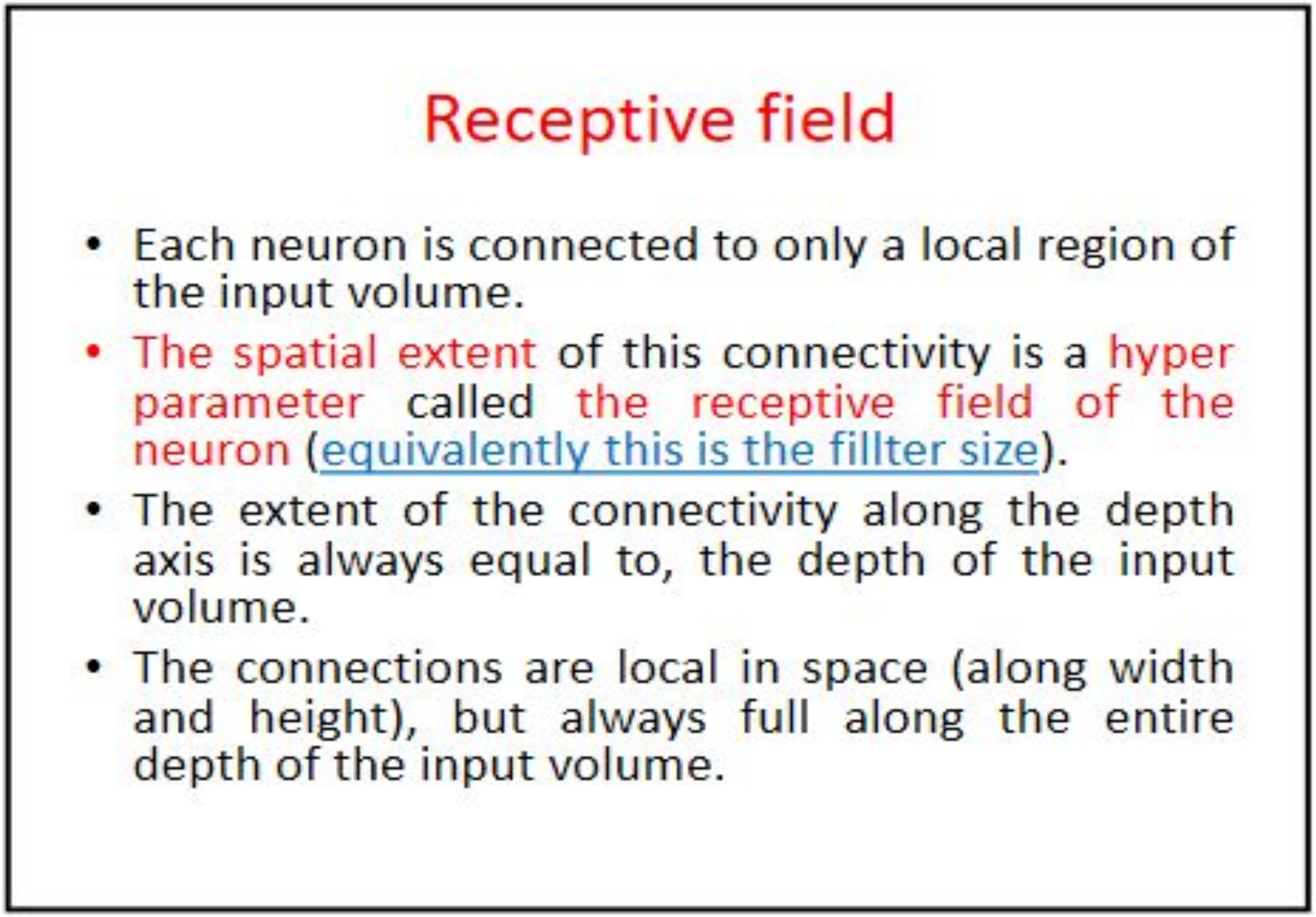


CAR
TRUCK
VAN
BICYCLE
...
INPUT
CONVOLUTION + RELU
POOLING
CONVOLUTION + RELU
POOLING
FLATTEN
FULLY CONNECTED
SOFTMAX
FEATURE LEARNING
CLASSIFICATION

- NPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. if we decided to use 12 filters.
- RELU layer will apply an element wise activation function, such as the max(0,x) thresholding at zero. This leaves the size of the volume unchanged
- POOL layer will perform a down sampling operation along the spatial dimensions (width, height), resulting in volume will reduced.
- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

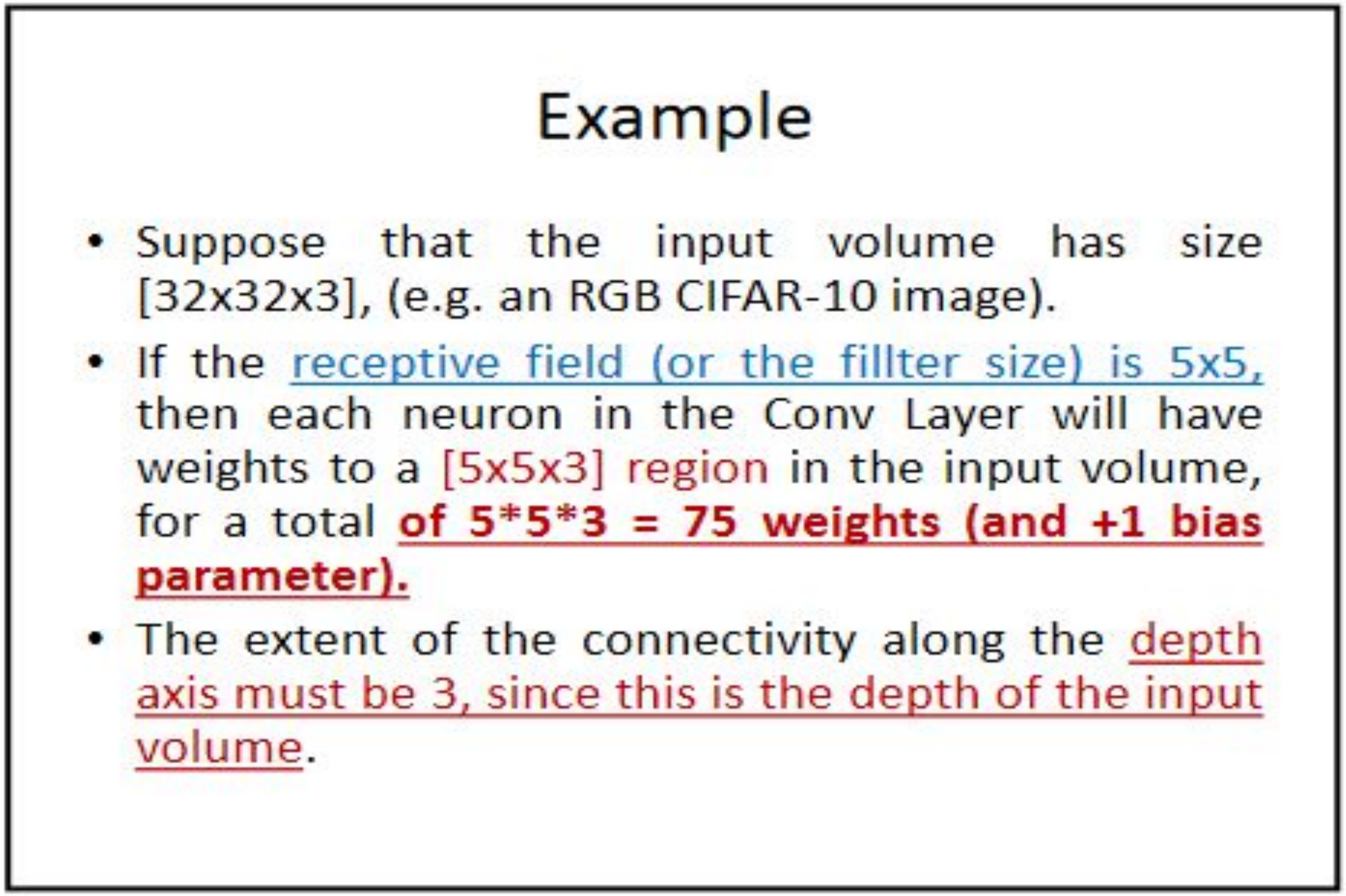
- In this way, ConvNets transform the original image layer by layer from the original pixel values to the final class scores.
- Note that some layers contain parameters and other don't. In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons).
- On the other hand, the RELU/POOL layers will implement a fixed function. The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in the training set for each image.

- A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume (e.g. holding the class scores)
- There are a few distinct types of Layers (e.g. CONV/FC/RELU/POOL are by far the most popular)
- Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function
- Each Layer may or may not have parameters (e.g. CONV/FC do, RELU/POOL don't)
- Each Layer may or may not have additional hyperparameters (e.g. CONV/FC/POOL do, RELU doesn't)



Receptive field

- Each neuron is connected to only a local region of the input volume.
- The spatial extent of this connectivity is a hyper parameter called the receptive field of the neuron (equivalently this is the fillter size).
- The extent of the connectivity along the depth axis is always equal to, the depth of the input volume.
- The connections are local in space (along width and height), but always full along the entire depth of the input volume.

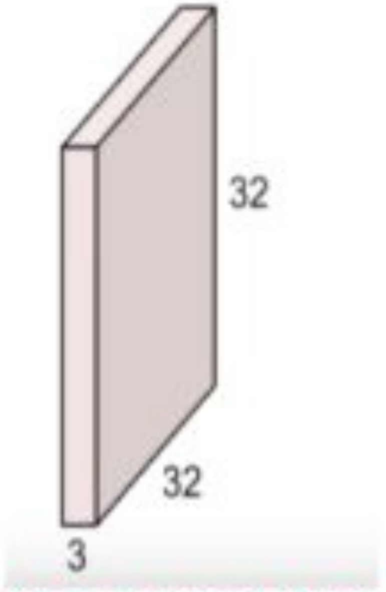


Example

- Suppose that the input volume has size [32x32x3], (e.g. an RGB CIFAR-10 image).
- If the receptive field (or the fillter size) is 5x5, then each neuron in the Conv Layer will have weights to a [5x5x3] region in the input volume, for a total **of 5*5*3 = 75 weights (and +1 bias parameter).**
- The extent of the connectivity along the depth axis must be 3, since this is the depth of the input volume.

Convolution Layer

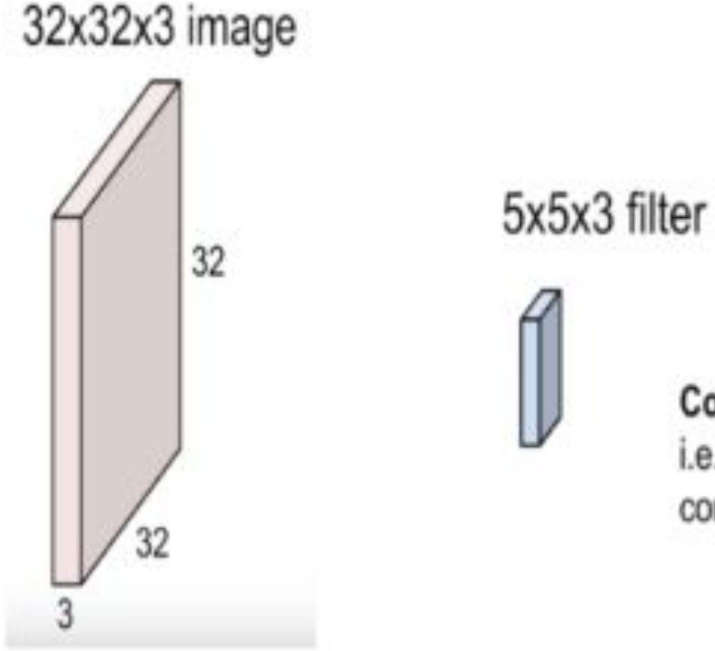
32x32x3 image -> preserve spatial structure



Instead of stretching the image into one long vector we are now going to keep the structure of the three dimensional input.

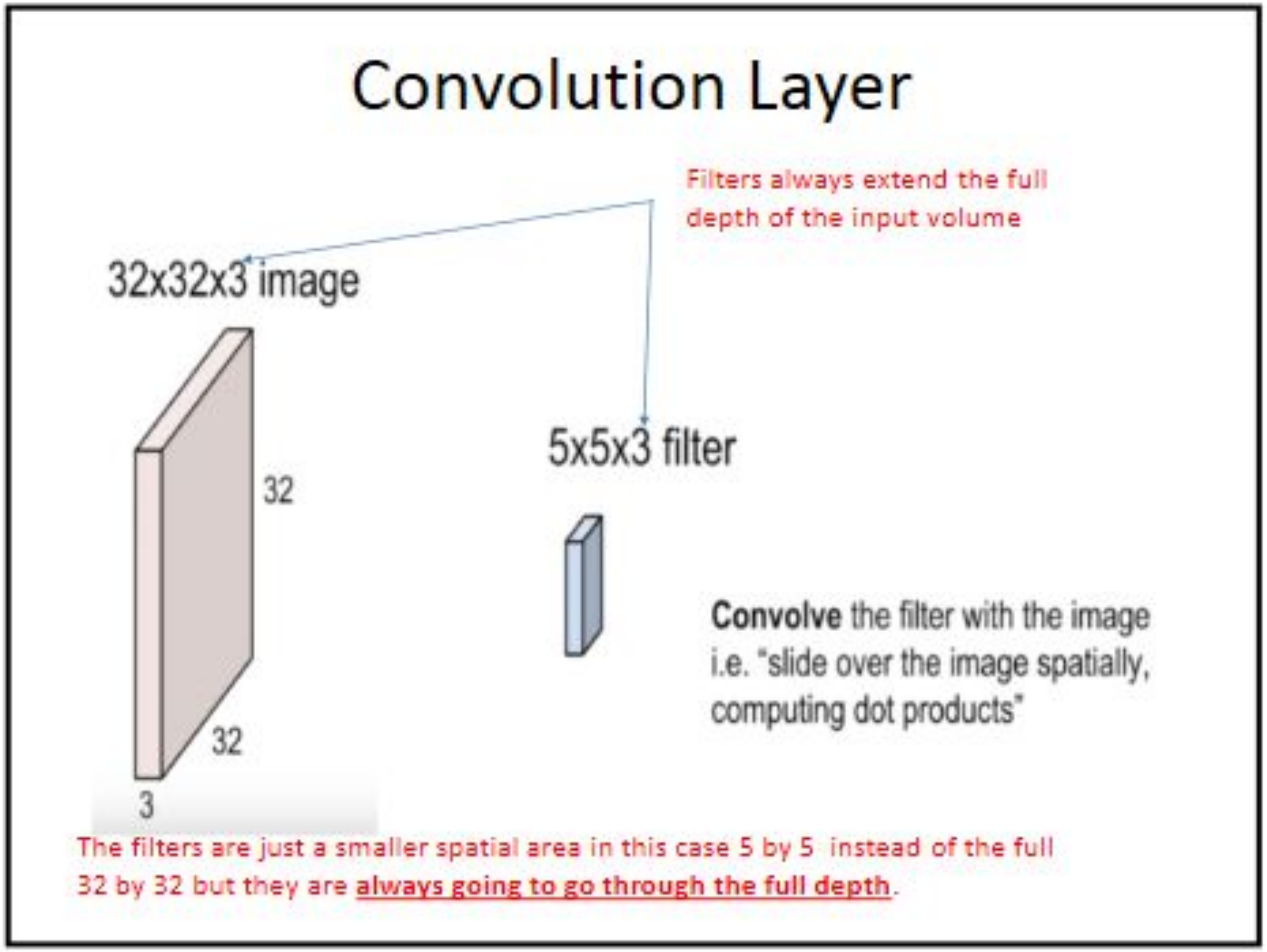
Main difference between convolution layer and fully connected layer is that we are trying to preserve spatial structure

Convolution Layer



Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

Filter = kernel =receptive field



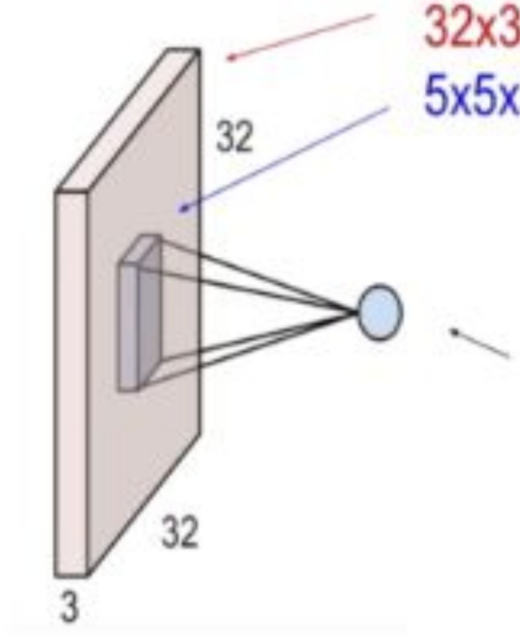
Convolution Layer

Filters always extend the full depth of the input volume

Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

The filters are just a smaller spatial area in this case 5 by 5 instead of the full 32 by 32 but they are **always going to go through the full depth**.

Convolution Layer

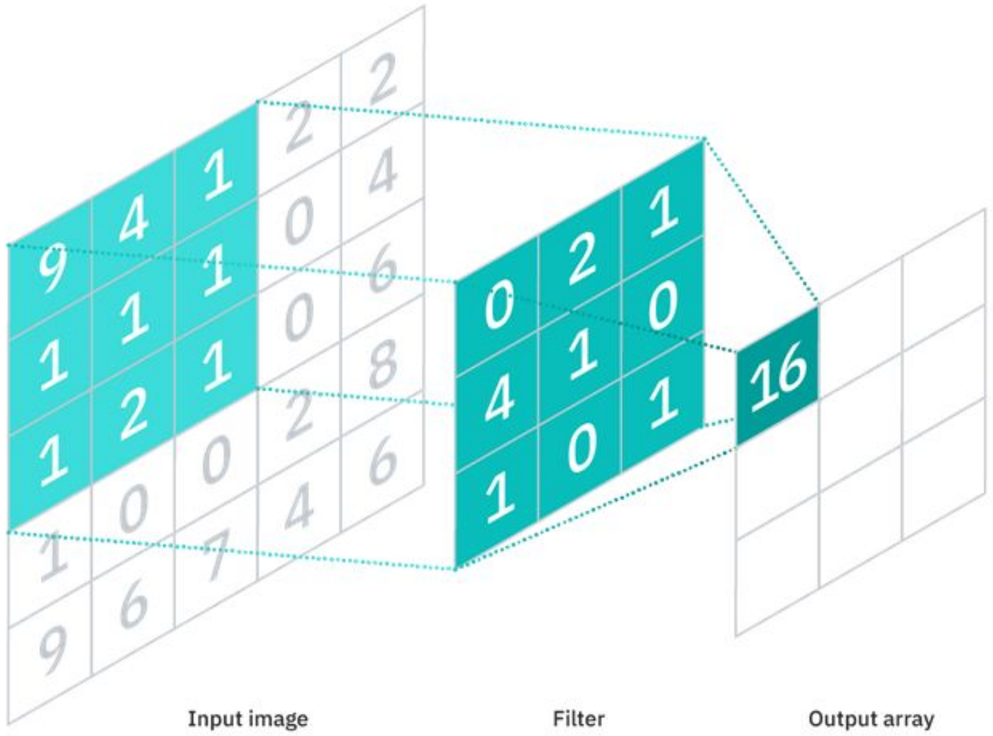


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

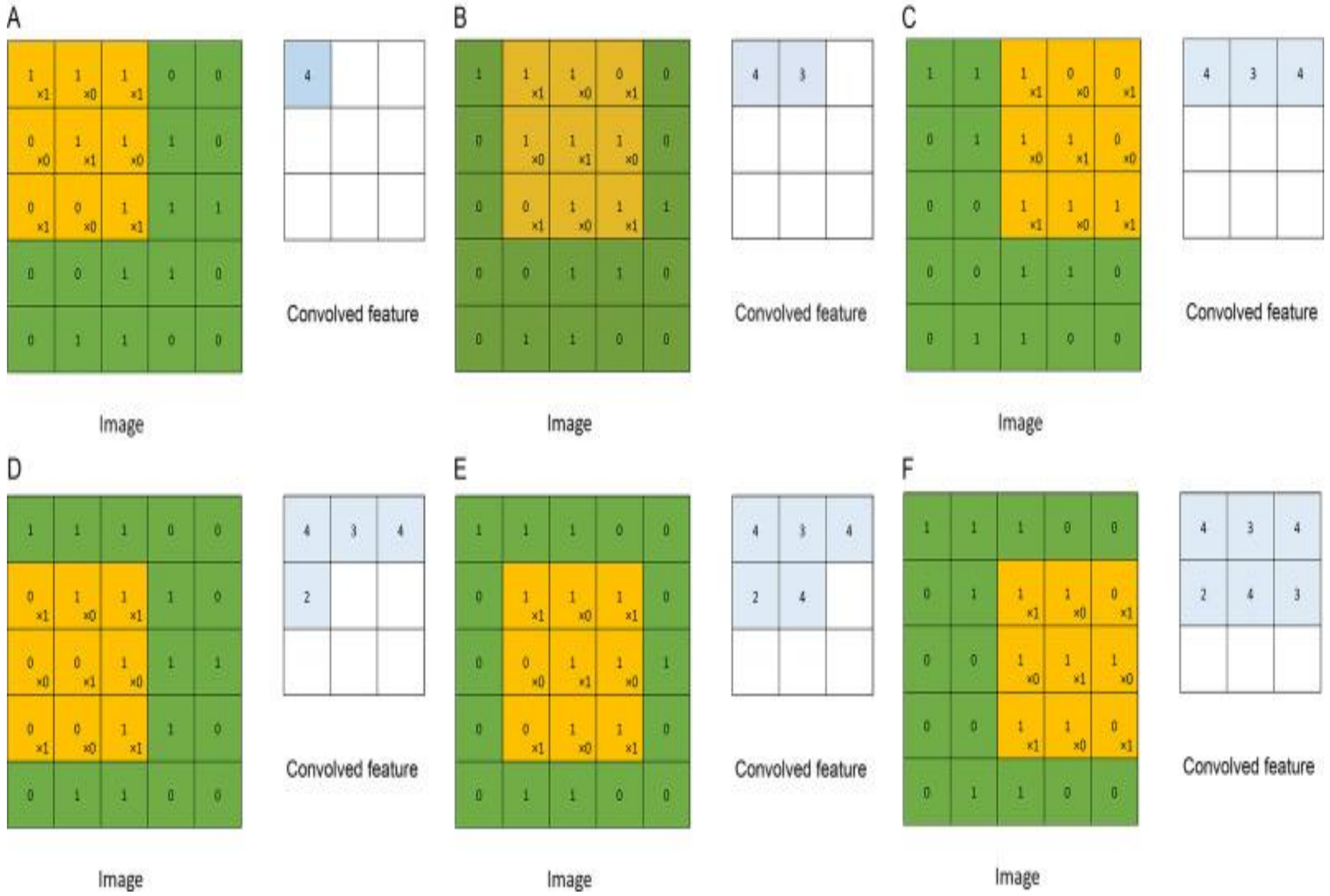
$$w^T x + b$$

We would stretch out 5 by 5 by 3 a get (1 x 75) and multiply that with image of (75 by 1)

Out put=9*0+4*2+1*1+1*4+1*1+1*0+1*1+2*0+1*1=16

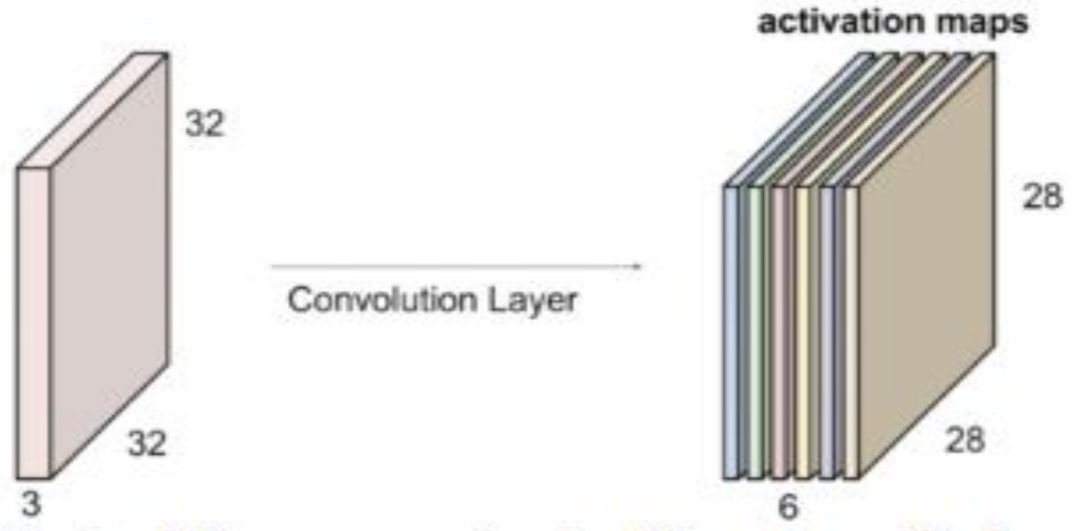


VIJAY KHARE


A
Image
Convolved feature
B
Image
Convolved feature
C
Image
Convolved feature
D
Image
Convolved feature
E
Image
Convolved feature
F
Image
Convolved feature

- As you can see in the image above, each output value in the feature map does not have to connect to each pixel value in the input image. It only needs to connect to the receptive field, where the filter is being applied.

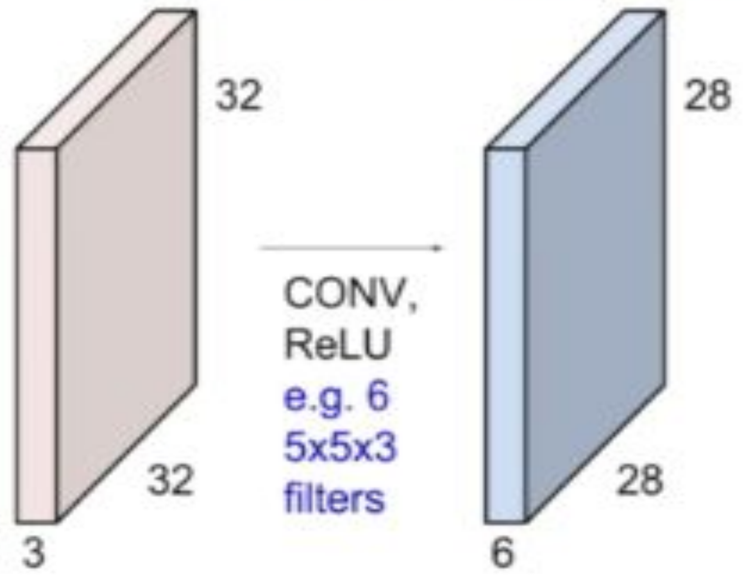
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



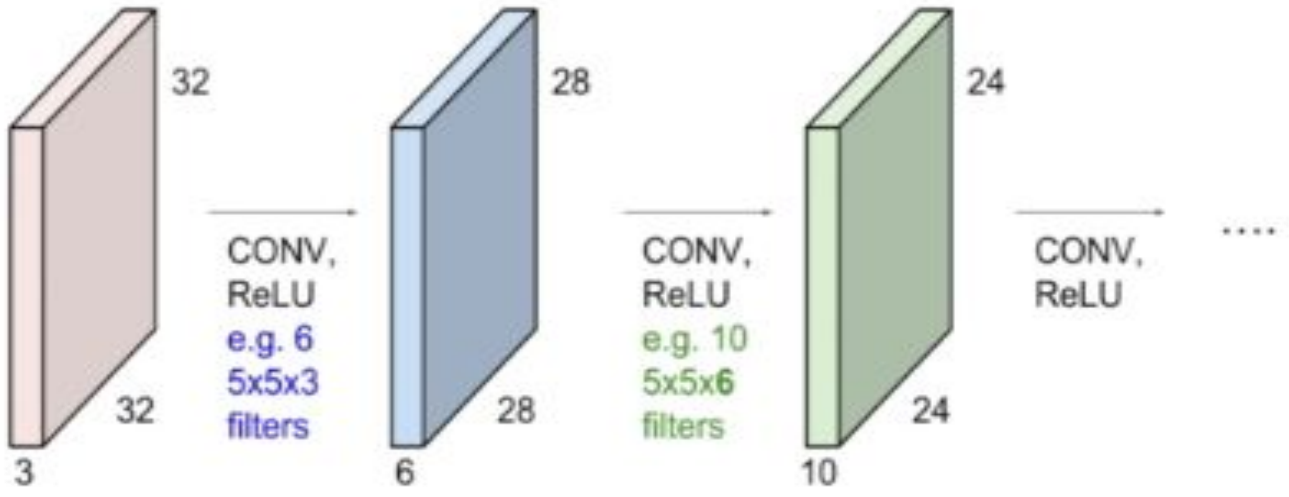
We stack them up and get a "New Image" of size 28 x 28 x 6

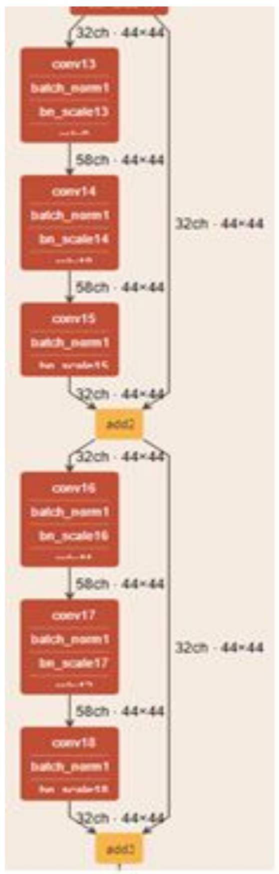
If we have 6 of these 5 x5x3 filters = total six activation maps out.
our "new image" is going to be 6 by 28 by 28

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.

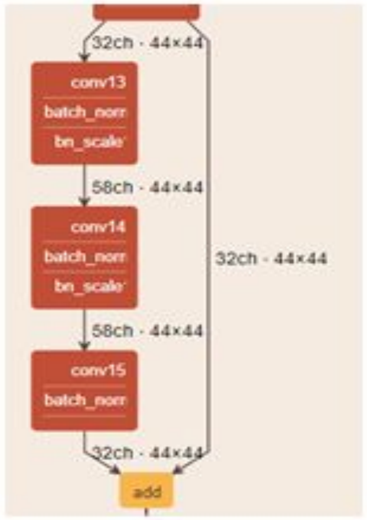


Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.





original architecture



modified architecture with some convolution layers removed

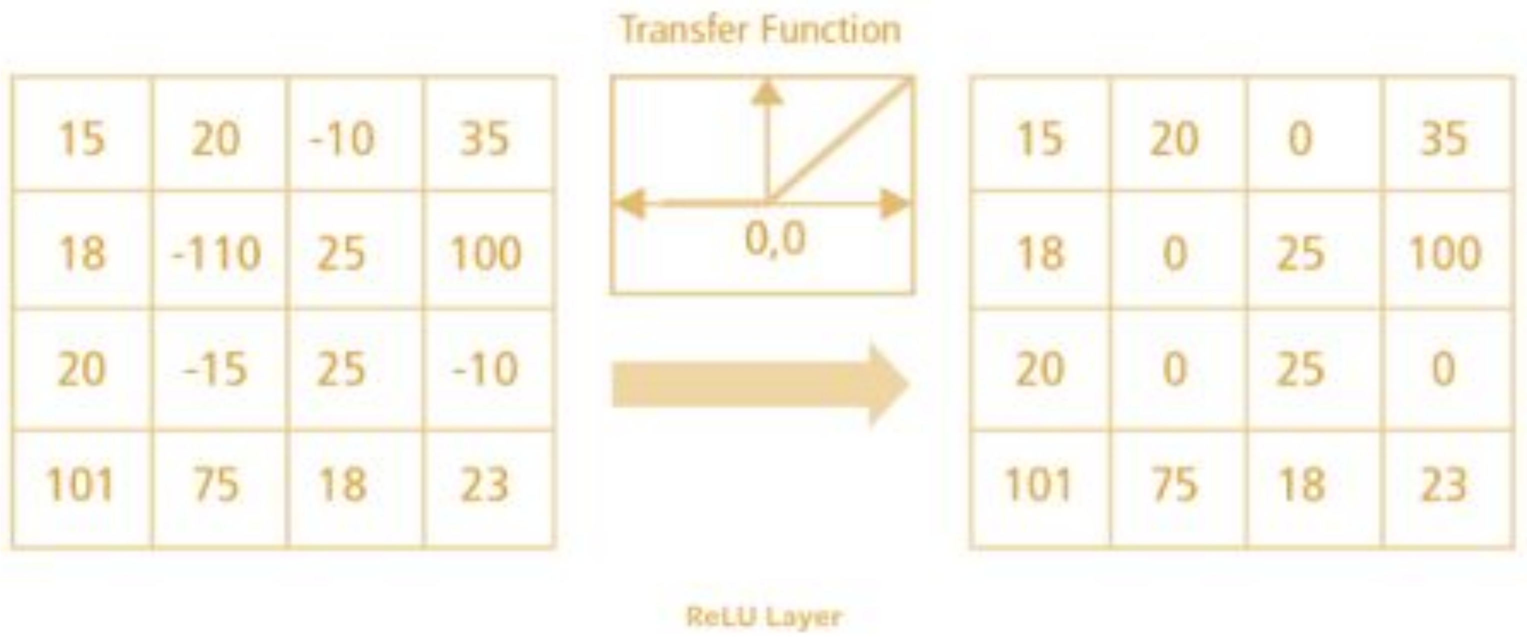
- Let us first start with the more straightforward part. Knowing the number of input and output layers and the number of their neurons is the easiest part. Every network has a single input layer and a single output layer. The number of neurons in the input layer equals the number of input variables in the data being processed. The number of neurons in the output layer equals the number of outputs associated with each input. But the challenge is knowing the number of

- The answer is you cannot analytically calculate the number of layers or the number of nodes to use per layer in an artificial neural network to address a specific real-world predictive modeling problem. The number of layers and the number of nodes in each layer are model hyperparameters that you must specify and learn. You must discover the answer using a robust test harness and controlled experiments. Regardless of the heuristics, you might encounter, all answers will come back to the need for careful experimentation to see what works best for

- For example, the filter size is one such hyperparameter you should specify before training your network.
  For an image recognition problem, if you think that a big amount of pixels are necessary for the network to recognize the object you will use large filters (as 11x11 or 9x9). If you think what differentiates objects are some small and local features you should use small filters (3x3 or 5x5).
  These are some tips but do not exist any rules.

Activation function

- ReLU stands for Rectified Linear Unit for a non-linear operation. The output is *f* (x) = max(0,x).
- Why ReLU is important : ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be



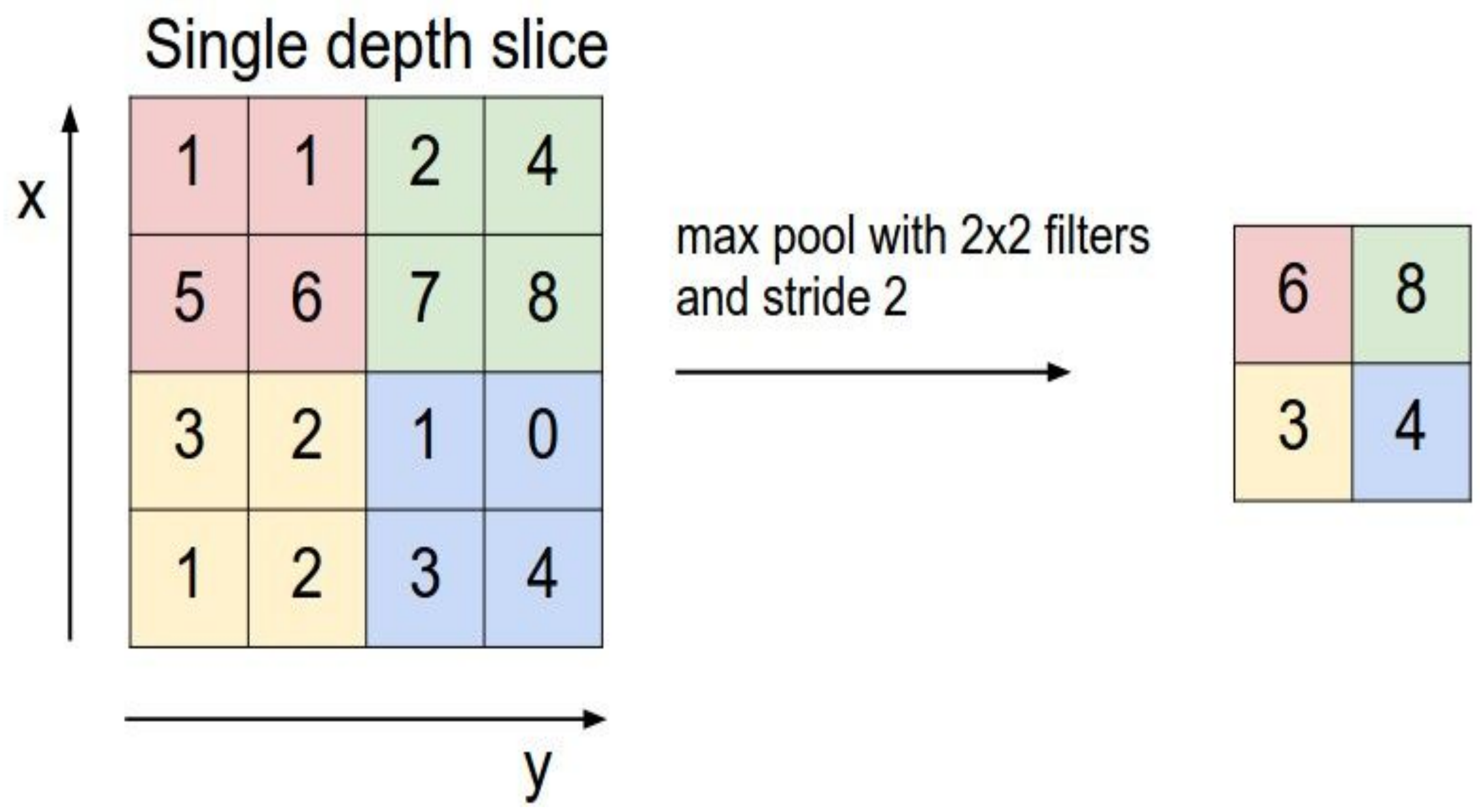
15	20	-10	35
18	-110	25	100
20	-15	25	-10
101	75	18	23

15	20	0	35
18	0	25	100
20	0	25	0
101	75	18	23

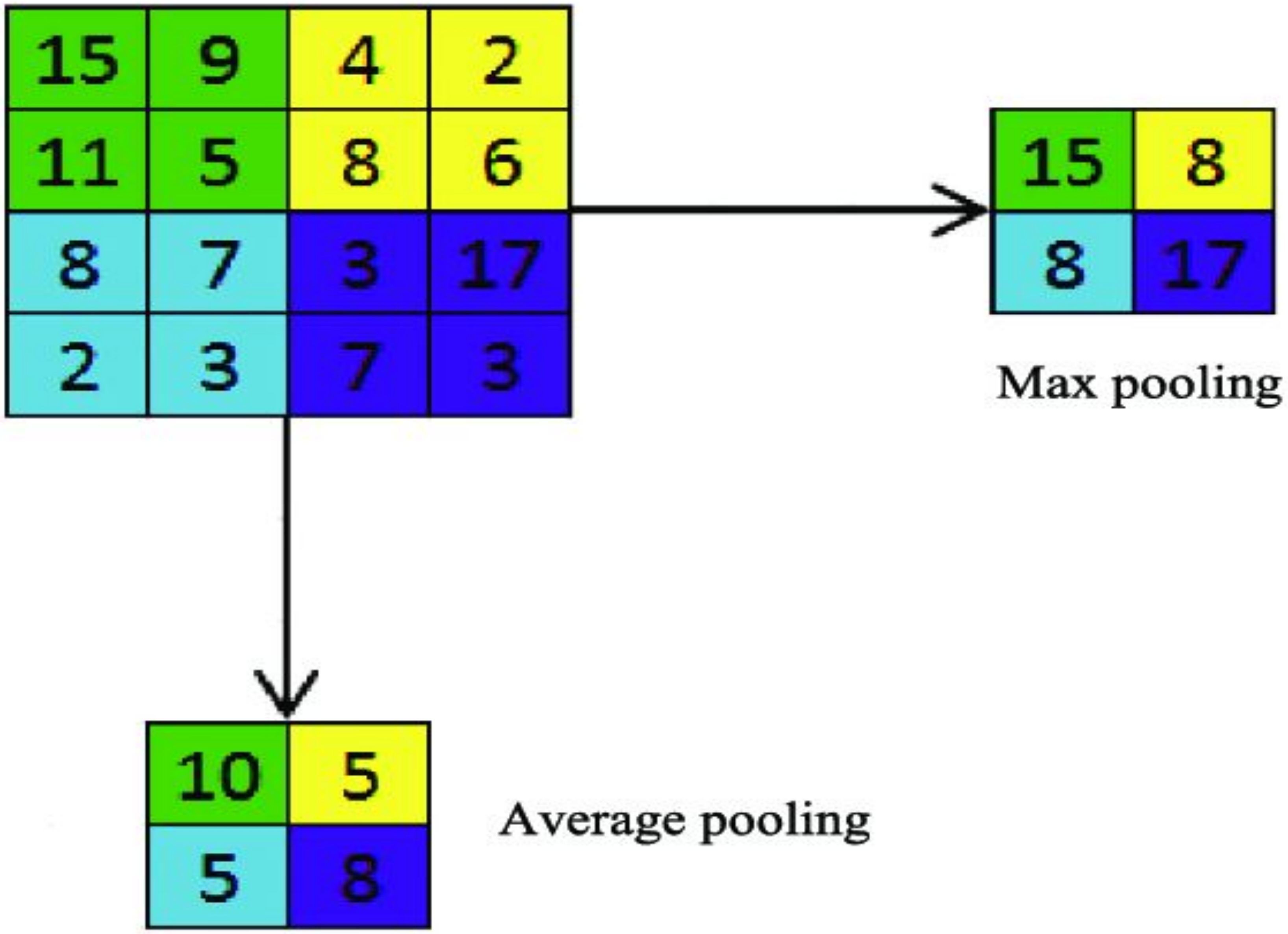
- There are other non linear functions such as tanh or sigmoid that can also be used instead of ReLU. Most of the data scientists use ReLU since performance wise ReLU is better than the other two.

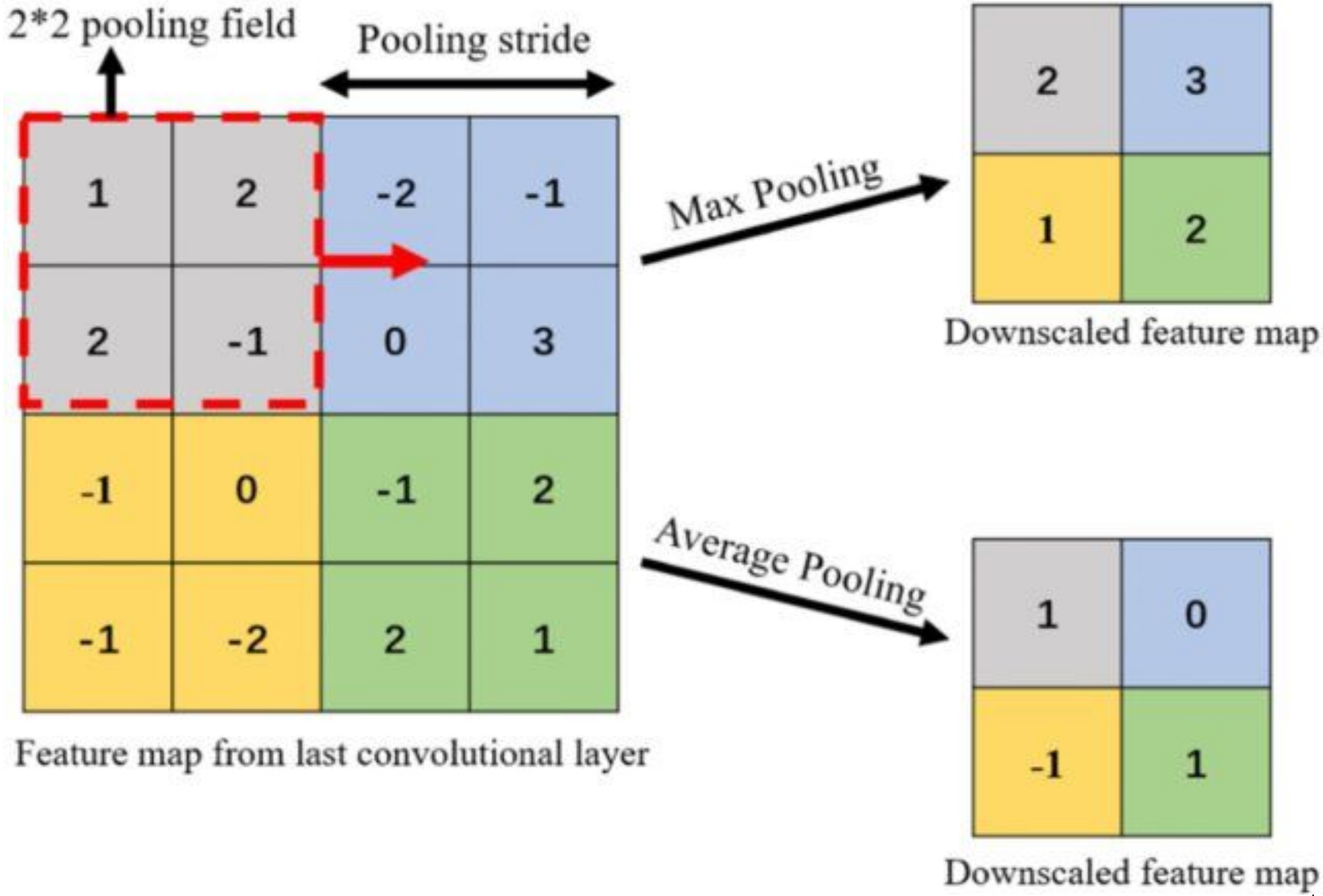
Pooling Layer

- It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture.
- Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.
- The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation.
- The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations.
- Every MAX operation would in this case be taking a max over 4 numbers (little 2x2 region in some depth slice). The depth dimension remains unchanged. More


Single depth slice
x
y
1
1
2
4
5
6
7
8
3
2
1
0
1
2
3
4
max pool with 2x2 filters
and stride 2
6
8
3
4

- **Max pooling:** As the filter moves across the input, it selects the pixel with the maximum value to send to the output array. As an aside, this approach tends to be used more often compared to average pooling.
- **Average pooling:** As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.
- While a lot of information is lost in the pooling layer, it also has a number of benefits to the CNN. They help to reduce complexity, improve efficiency, and limit risk of over fitting.


15
9
4
2
11
5
8
6
8
7
3
17
2
3
7
3
15
8
8
17
Max pooling
10
5
5
8
Average pooling


2*2 pooling field
Pooling stride
1
2
-2
-1
2
-1
0
3
-1
0
-1
2
-1
-2
2
1
Feature map from last convolutional layer
Max Pooling
2
3
1
2
Downscaled feature map
Average Pooling
1
0
-1
1
Downscaled feature map

- In CNNs, max pooling is generally preferred over average pooling because it is better at capturing local feature information and preserving the most prominent features in an image, like edges and textures. Max pooling selects the maximum value within a window, resulting in a sharper representation of the feature map

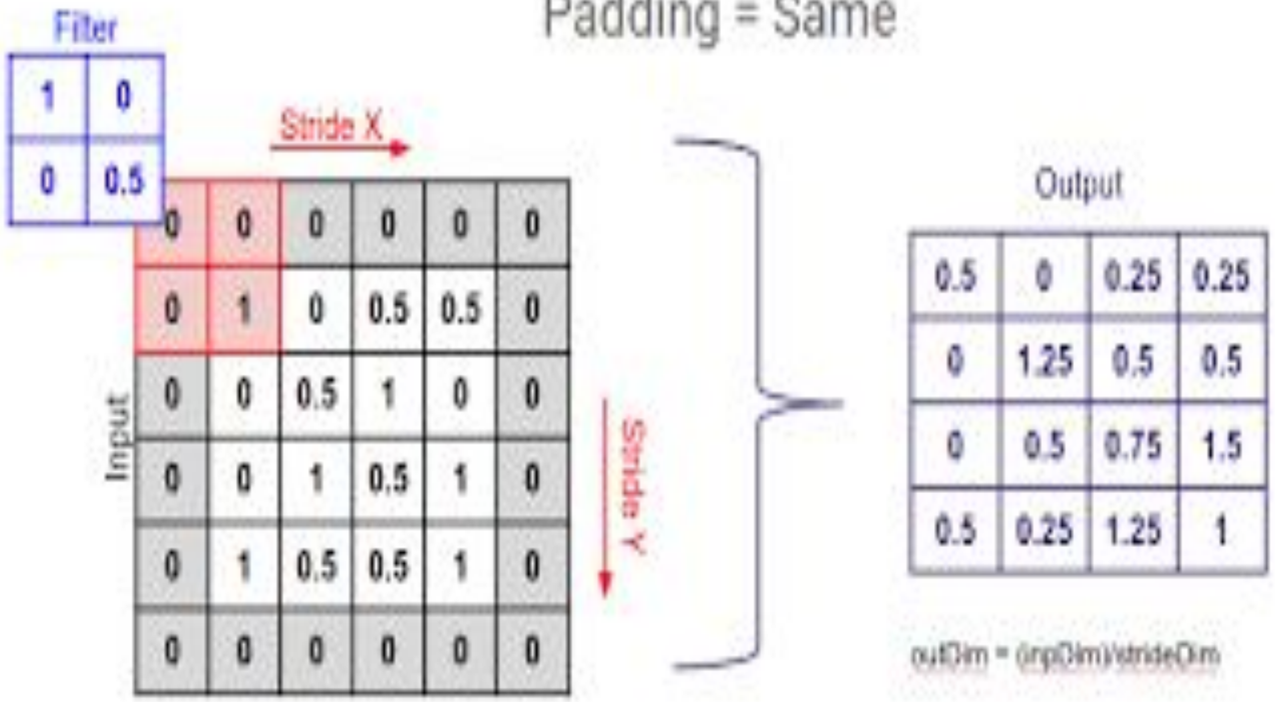
- **Better Feature Preservation:**
- Max pooling selects the most salient features, while average pooling provides a more balanced representation, which may not be as effective in highlighting important features.
- **Clearer Feature Maps:**
- Max pooling produces a sharper representation of the feature map, which can be beneficial in computer vision tasks where highlighting edges and textures is important
- **Faster Computation:**

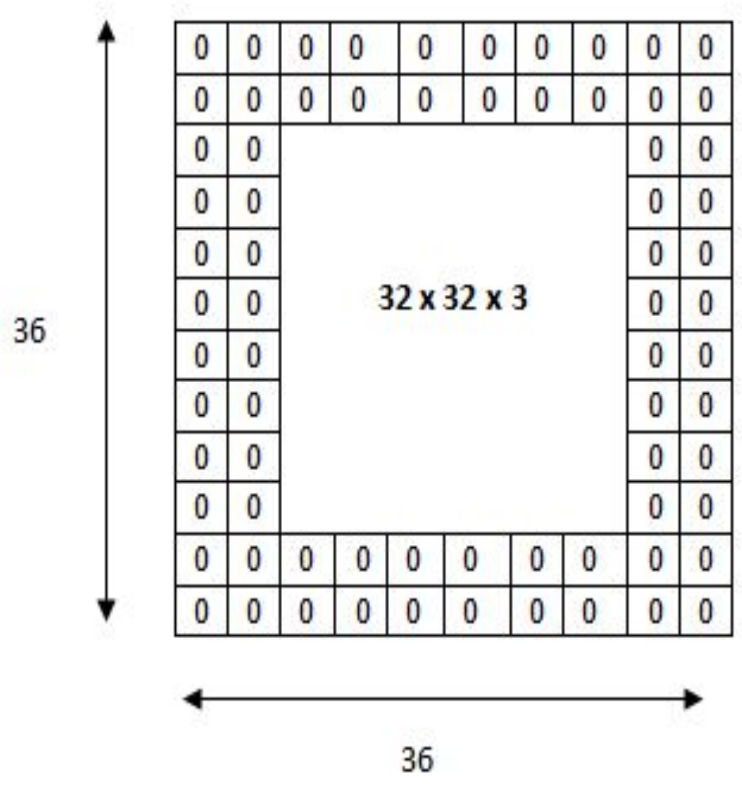
- Note that the weights in the feature detector remain fixed as it moves across the image, which is also known as parameter sharing. Some parameters, like the weight values, adjust during training through the process of backpropagation and gradient descent. However, there are three hyperparameters which affect the volume size of the output that need to be set before the training of the neural network begins. These include:

- 1. The **number of filters** affects the depth of the output. For example, three distinct filters would yield three different feature maps, creating a depth of three.
- 2. **Stride** is the distance, or number of pixels, that the kernel moves over the input matrix. While stride values of two or greater is rare, a larger stride yields a smaller output.
- 3. **Zero-padding** is usually used when the filters do not fit the input image. This sets all elements that fall outside of the input matrix to zero, producing a larger or

- **Valid padding:** This is also known as no padding. In this case, the last convolution is dropped if dimensions do not align.
- **Same padding:** This padding ensures that the output layer has the same size as the input layer
- **Full padding:** This type of padding increases the size of the output by adding zeros to the border of the input.

Padding = Same

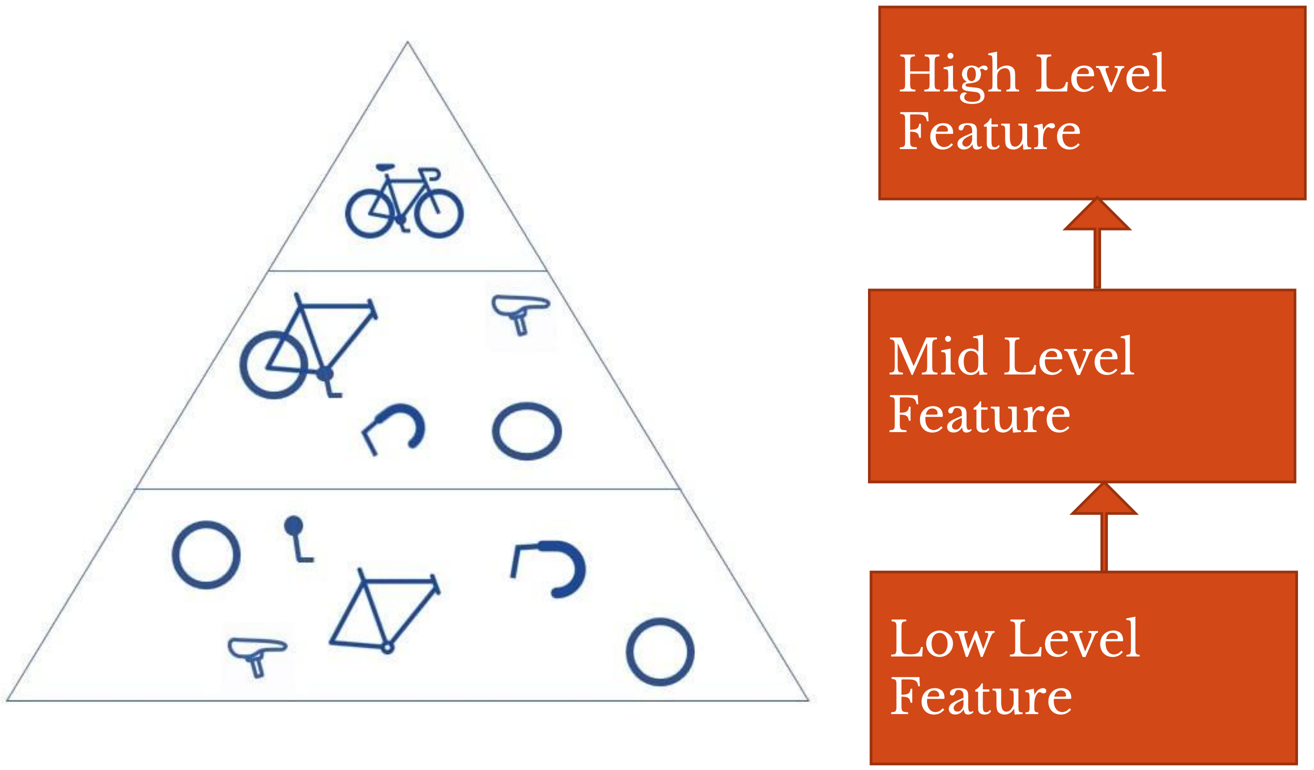




The input volume is 32 x 32 x 3. If we imagine two borders of zeros around the volume, this gives us a 36 x 36 x 3 volume. Then, when we apply our conv layer with our three 5 x 5 x 3 filters and a stride of 1, then we will also get a 32 x 32 x3 output volume.

- As we mentioned earlier, another convolution layer can follow the initial convolution layer. When this happens, the structure of the CNN can become hierarchical as the later layers can see the pixels within the receptive fields of prior layers. As an example, let’s assume that we’re trying to determine if an image contains a bicycle.

- You can think of the bicycle as a sum of parts. It is comprised of a frame, handlebars, wheels, pedals, et cetera. Each individual part of the bicycle makes up a lower-level pattern in the neural net, and the combination of its parts represents a higher-level pattern, creating a feature hierarchy within the CNN.


High Level Feature
Mid Level Feature
Low Level Feature

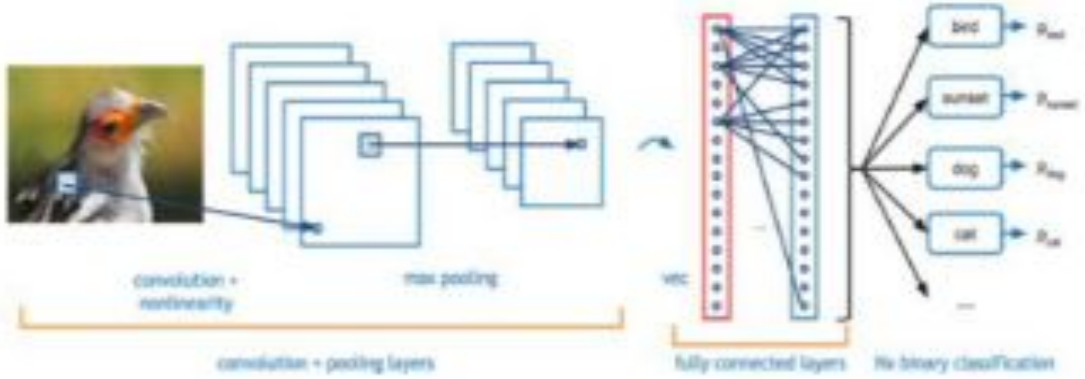
- Accepts a volume of size $W1 \times H1 \times D1$
- Requires four hyperparameters:
- Number of filters K,
- their spatial extent F,
- the stride S,
- the amount of zero padding P.
- Produces a volume of size $W2 \times H2 \times D2$ where:
- $W2=(W1-F+2P)/S+1$
- $H2=(H1-F+2P)/S+1$ (i.e. width and height are computed equally by symmetry)
- $D2=K$

- With parameter sharing, it introduces $F \cdot F \cdot D1$ weights per filter, for a total of $(F \cdot F \cdot D1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W2 \times H2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.
- A common setting of the hyperparameters is $F=3, S=1, P=1$. However, there are common conventions and rules of thumb that motivate these hyperparameters.

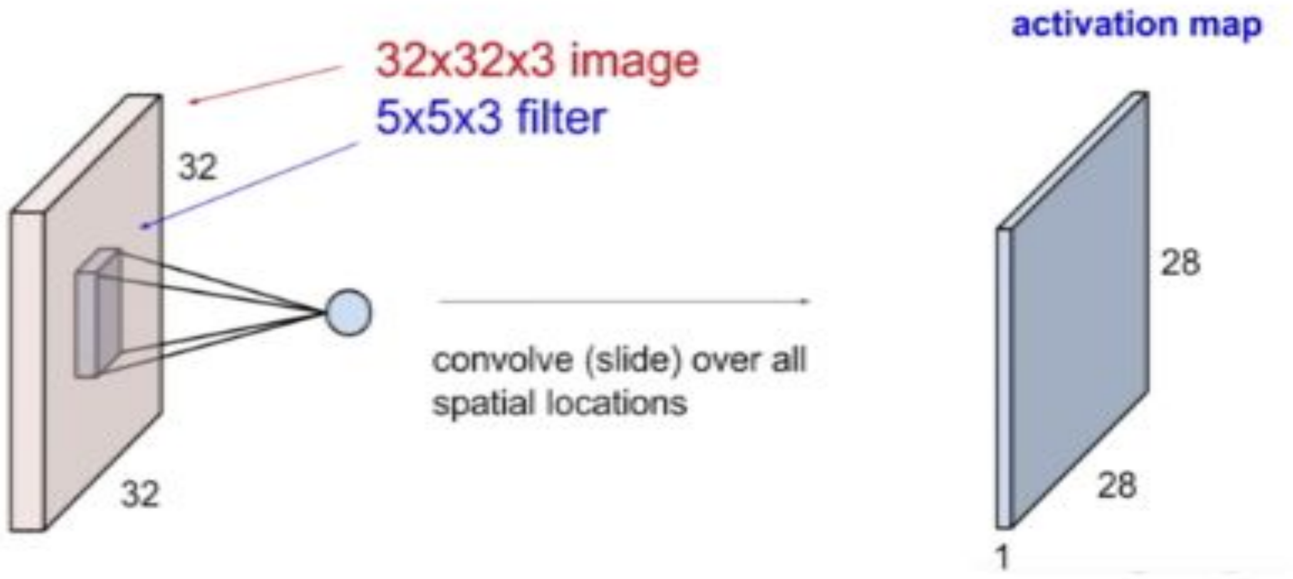
Convolution Neural Networks

So there are four main operations in a basic CNN:

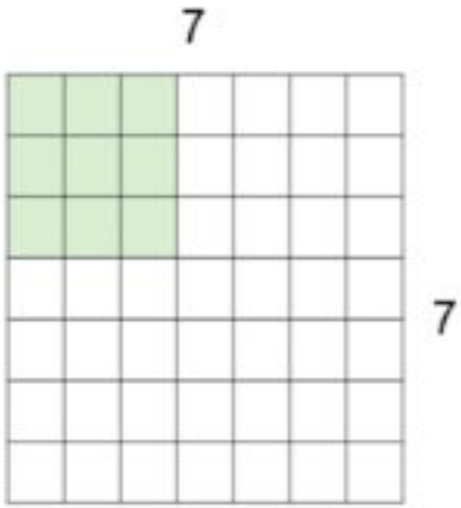
1. Convolution
2. Non Linearity (ReLU, tanh, sigmoid)
3. Pooling or Down-Sampling of features
4. Prediction



Spatial Dimensions:

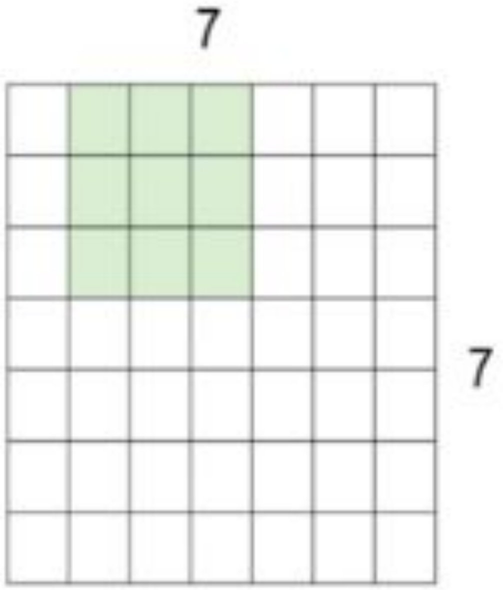


Spatial Dimensions:



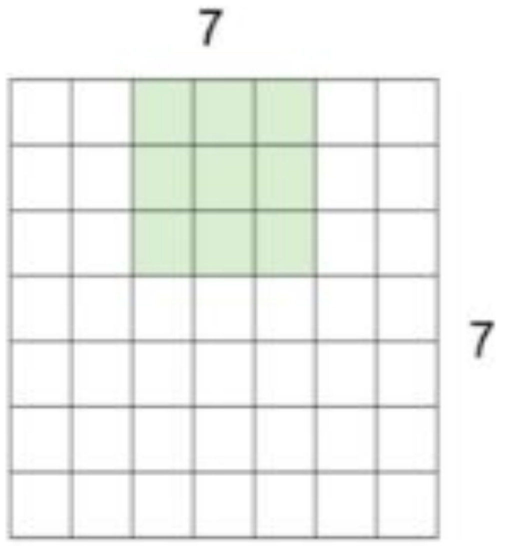
7x7 input (spatially)
assume 3x3 filter

Spatial Dimensions:



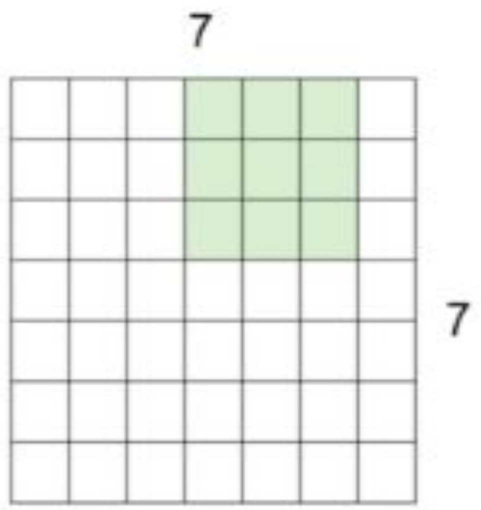
7x7 input (spatially)
assume 3x3 filter

Spatial Dimensions



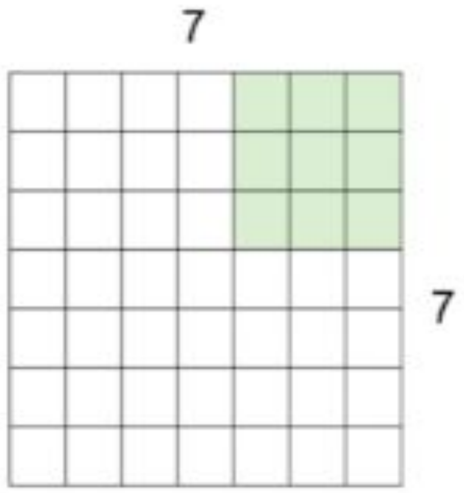
7x7 input (spatially)
assume 3x3 filter

Spatial Dimensions



7x7 input (spatially)
assume 3x3 filter

Spatial Dimensions

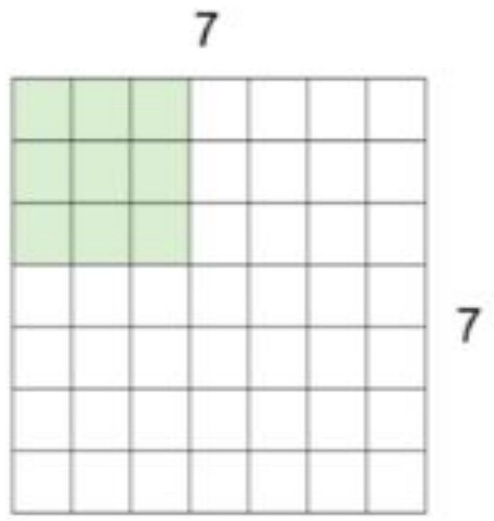


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

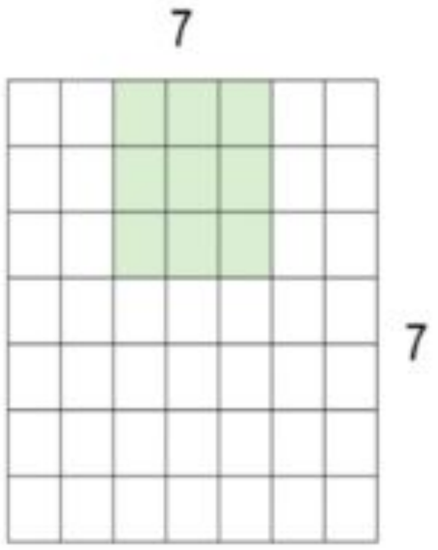
5x5 output :by sliding 3x3 filter over 5 spatial locations horizontally and five spatial location vertically.

Stride 2:



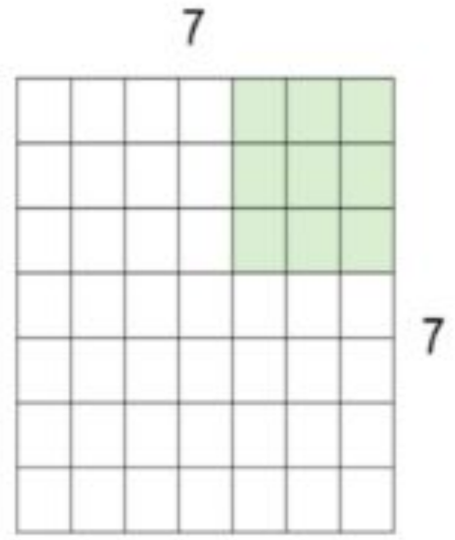
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Stride 2:



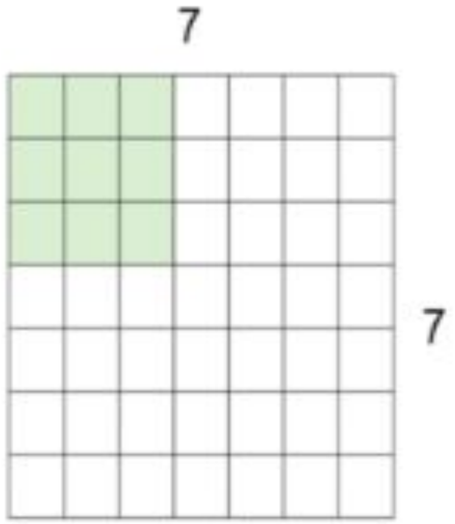
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Stride 2:

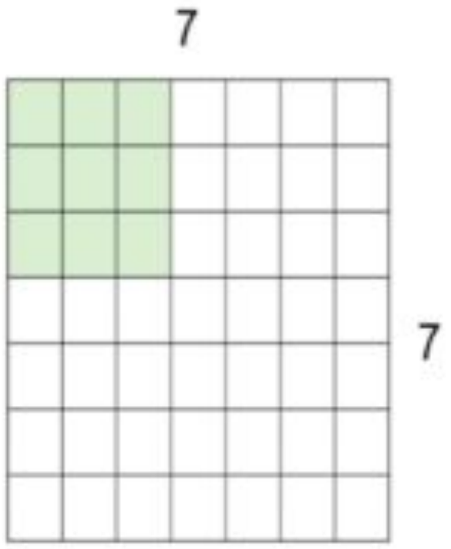


7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Stride 3:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

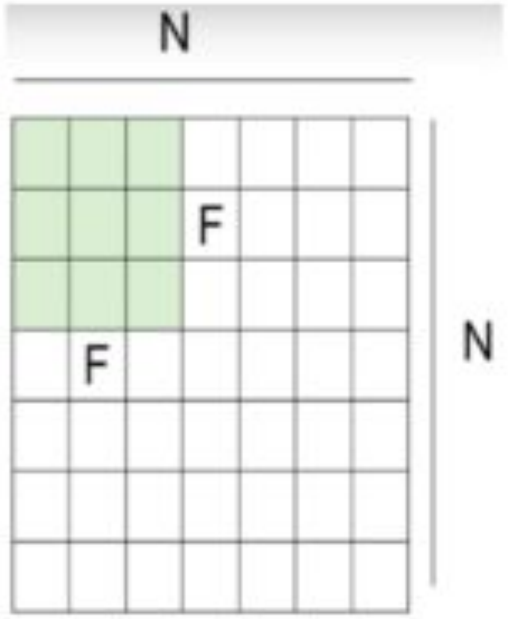


asymmetric output!

7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

Calculating output size:



Output size:

(N - F) / stride + 1

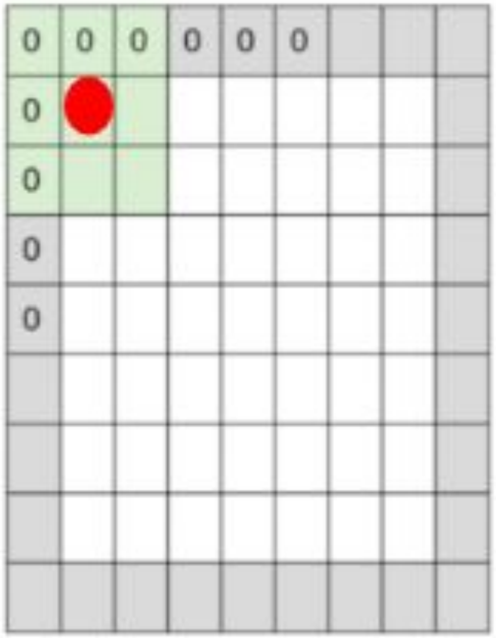
e.g. N = 7, F = 3:

stride 1 => (7 - 3)/1 + 1 = 5

stride 2 => (7 - 3)/2 + 1 = 3

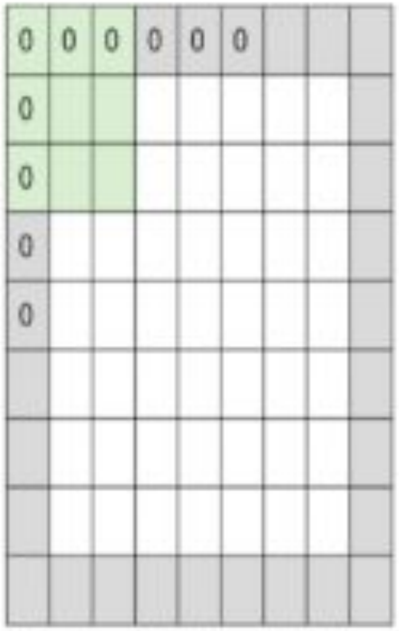
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

In practice: Common to zero pad the border



- In practice it's common to zero pad the borders in order to maintain the same input/output size.
- Pad input image with zeros: now place a filter centered at the upper right hand pixel location of actual input image.
- Zero padding helps us process the edges/corners of the image.

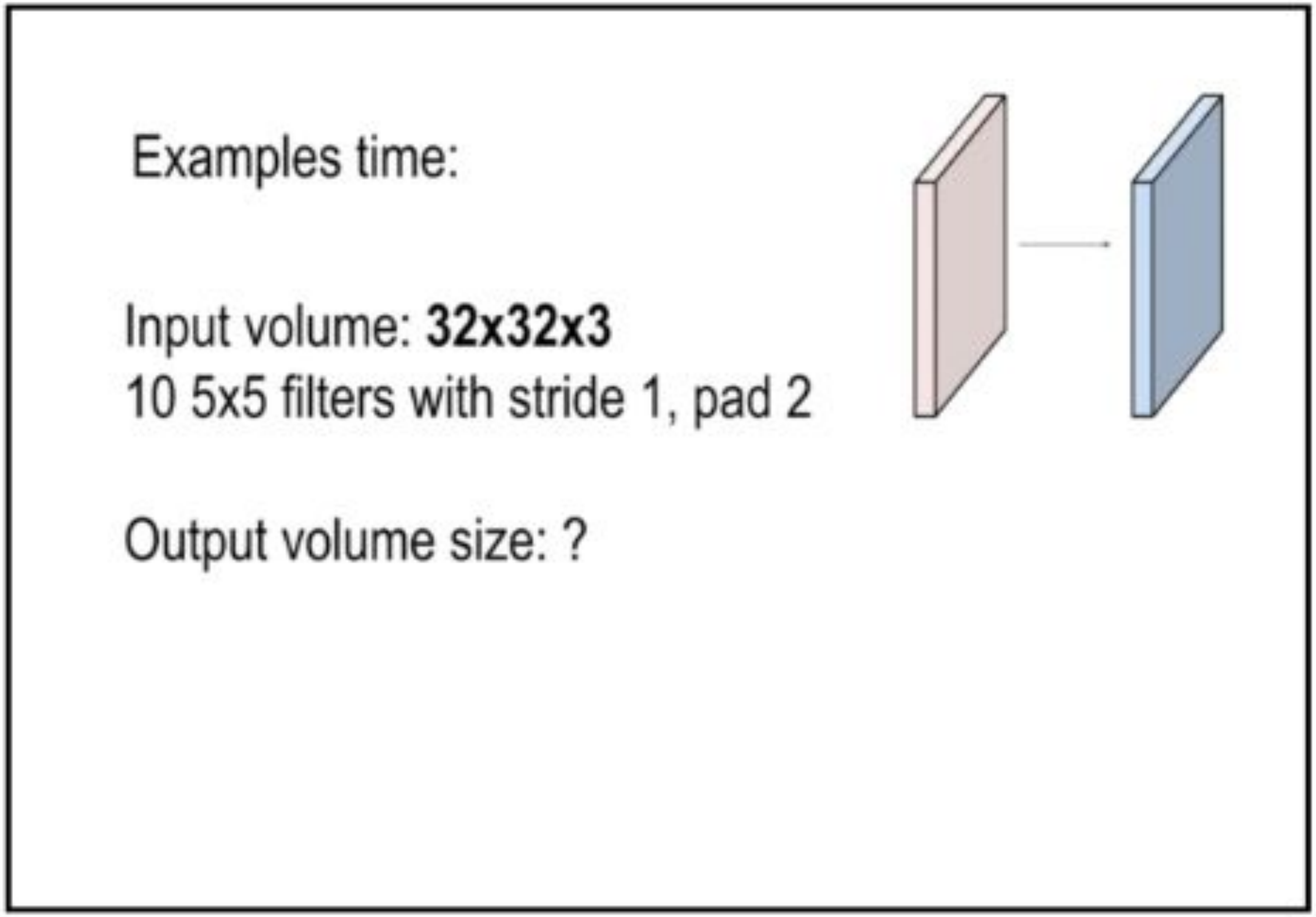
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

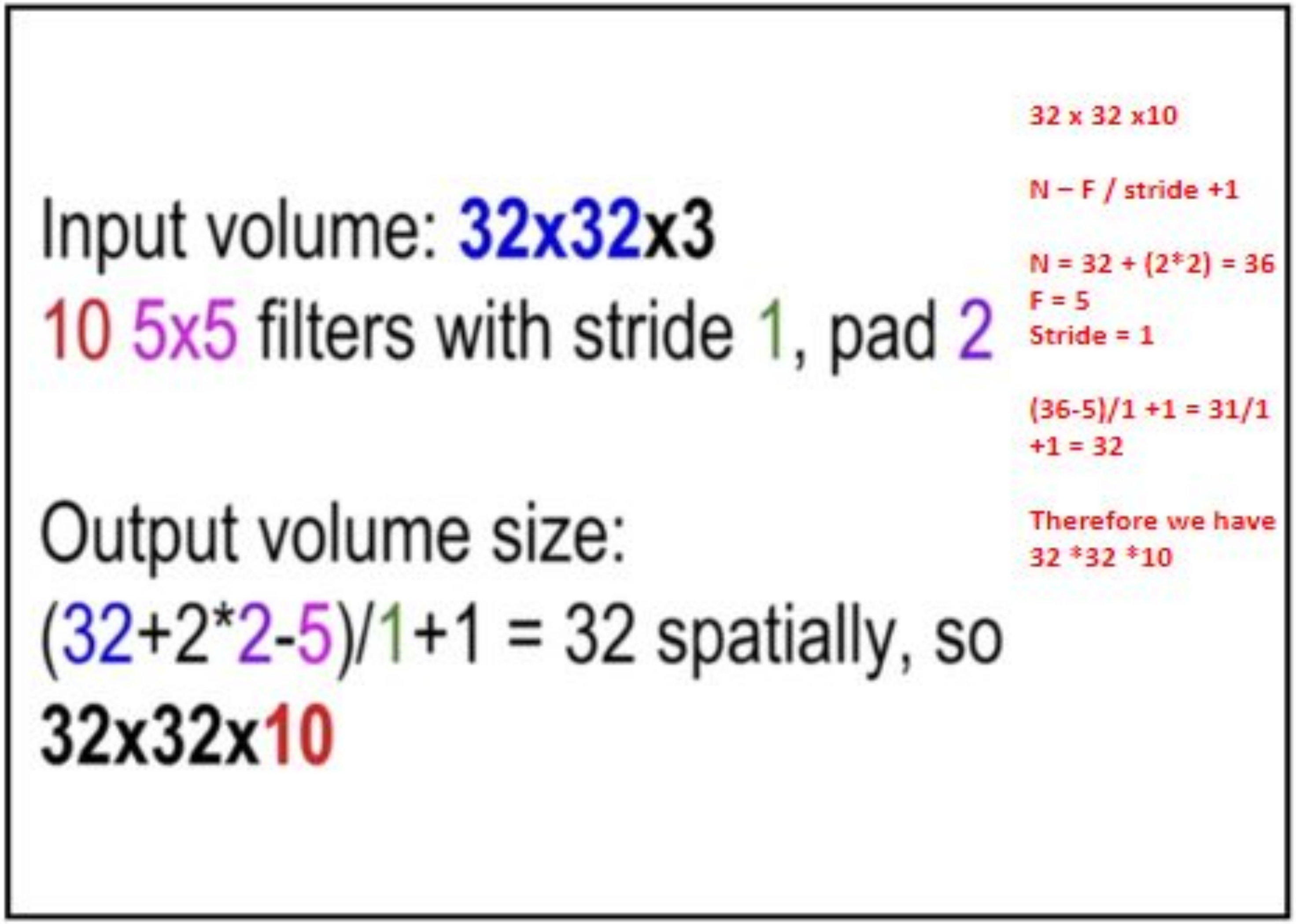
Now N = 9, F, s still equal to 3 and stride is still 1.
Total output size of that layer would be 7 x 7 x (# of filters that you have)



Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?



Input volume: 32x32x3

10 5x5 filters with stride 1, pad 2

Output volume size:

(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

32 x 32 x10

N – F / stride +1

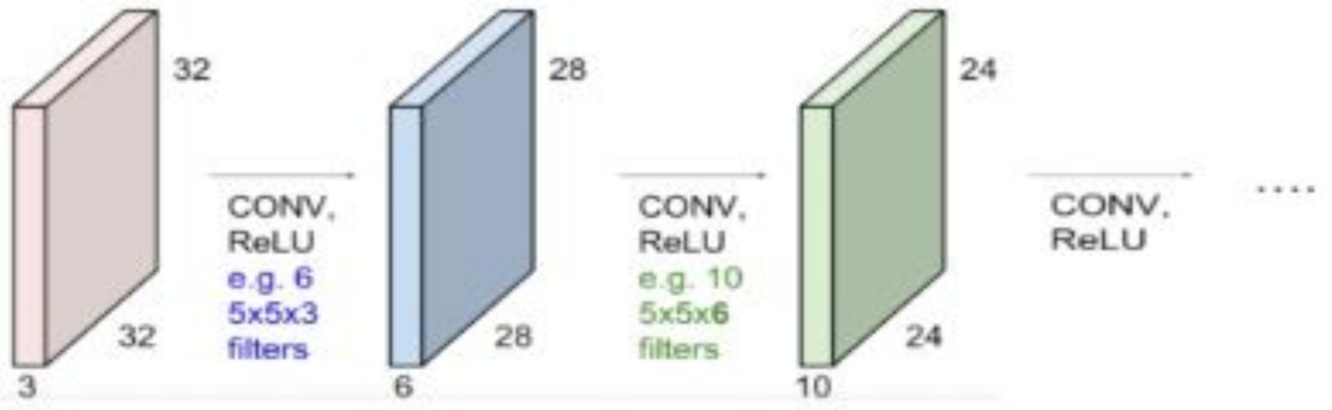
N = 32 + (2*2) = 36
F = 5
Stride = 1

(36-5)/1 +1 = 31/1 +1 = 32

Therefore we have 32 *32 *10

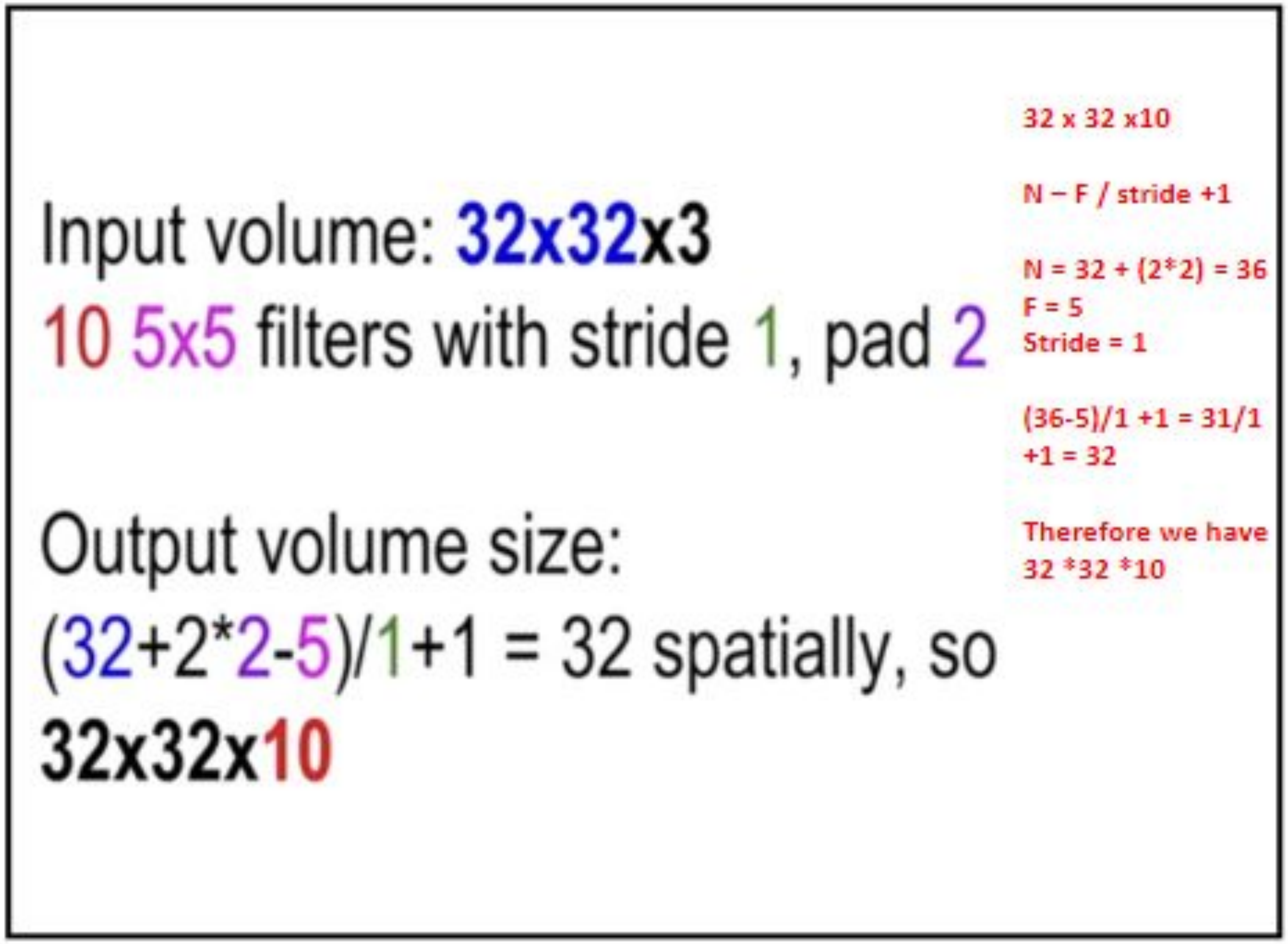
Remember back to...

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



Drawback:

- ❑ **It is similar to throwing away data points.**
- ❑For a deep network, very quickly the size of **activation maps is going to shrink to very small**.
- ❑Not desirable:
  - ❑losing information or smaller number of values in order to represent the original image.



Input volume: 32x32x3

10 5x5 filters with stride 1, pad 2

Output volume size:

(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

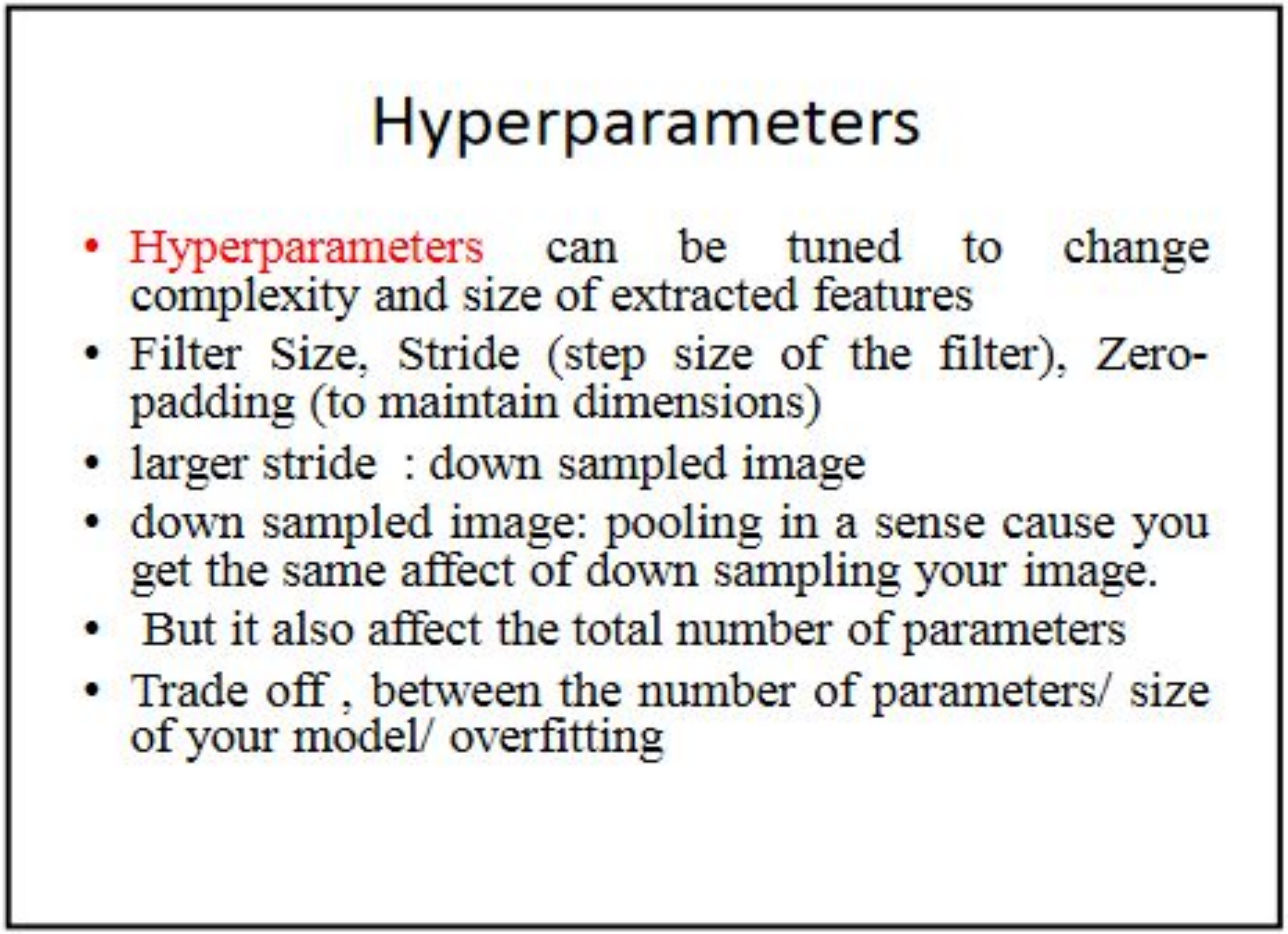
32 x 32 x10

N – F / stride +1

N = 32 + (2*2) = 36
F = 5
Stride = 1

(36-5)/1 +1 = 31/1 +1 = 32

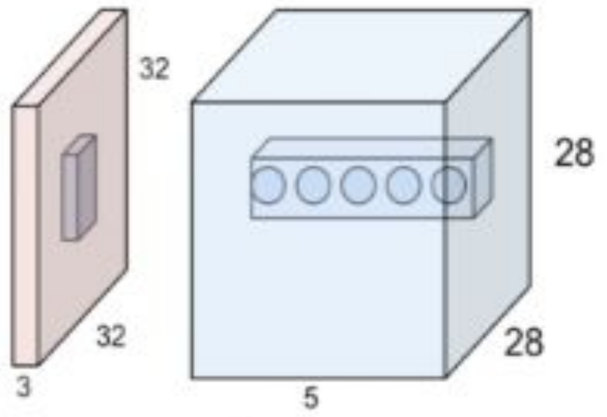
Therefore we have 32 *32 *10



Hyperparameters

- Hyperparameters can be tuned to change complexity and size of extracted features
- Filter Size, Stride (step size of the filter), Zero-padding (to maintain dimensions)
- larger stride : down sampled image
- down sampled image: pooling in a sense cause you get the same affect of down sampling your image.
- But it also affect the total number of parameters
- Trade off , between the number of parameters/ size of your model/ overfitting

Neuron View of the CONV Layer



E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
(28x28x5)

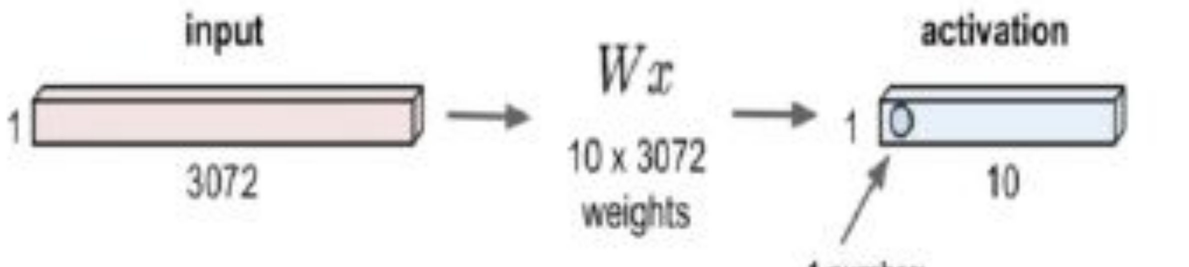
- Look at the filters across in on spatial location of the activation grid, going through the depth of these 5 neurons,
- All of these neurons are looking at the same region in the input image
- But they are just looking for different aspects
- Because each neuron comes from a different filter applied to the same spatial location in input image.

Fully-Connected Layer

- The name of the full-connected layer aptly describes itself. As mentioned earlier, the pixel values of the input image are not directly connected to the output layer in partially connected layers. However, in the fully-connected layer, each node in the output layer connects directly to a node in the previous layer.
- This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While convolutional and pooling layers tend to use ReLu functions, FC layers usually leverage a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1.

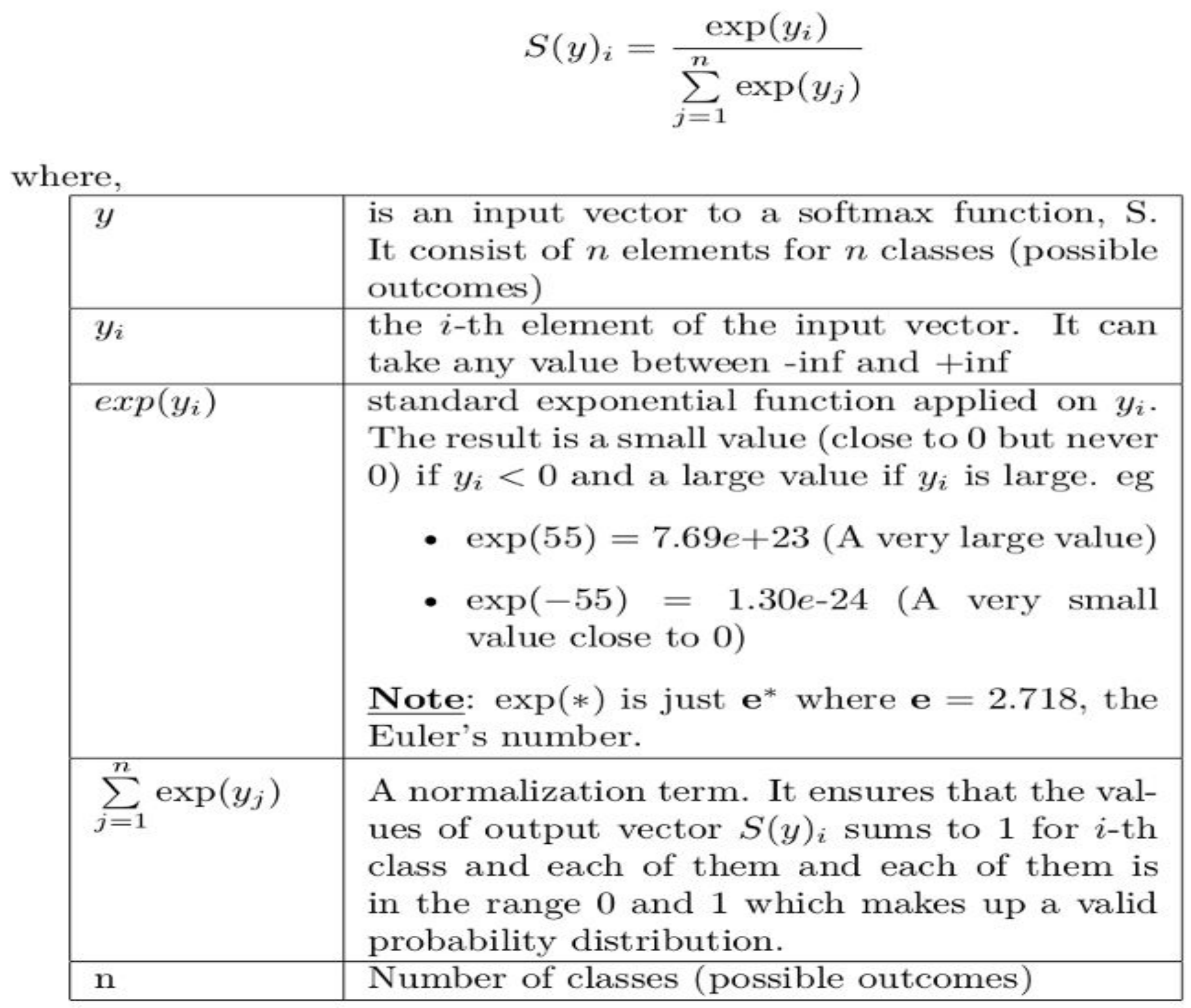
Fully Connected Layer

- 32 x 32 x 3 image => stretch 3072 x 1



1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

Softmax is an activation function that scales numbers/logits into probabilities. The output of a Softmax is a vector (say v) with probabilities of each possible outcome. The probabilities in vector v sums to one for all possible outcomes or classes.
Mathematically, Softmax is defined as,

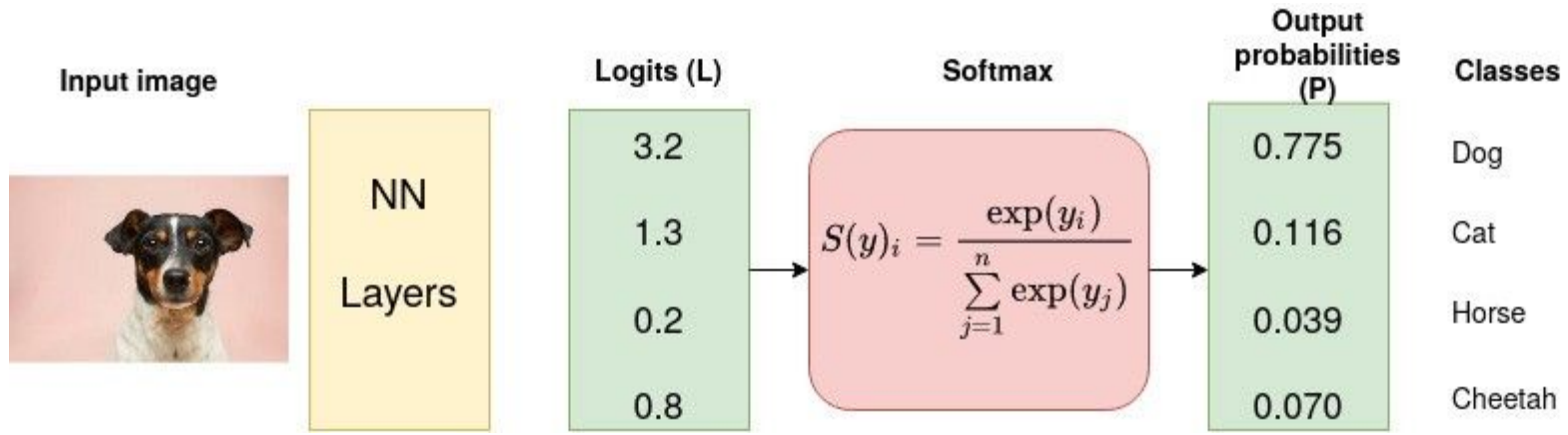


$$S(y)_i = \frac{\exp(y_i)}{\sum\limits_{j=1}^{n} \exp(y_j)}$$

where,

y	is an input vector to a softmax function, S. It consist of n elements for n classes (possible outcomes)
y_i	the i-th element of the input vector. It can take any value between -inf and +inf
$exp(y_i)$	standard exponential function applied on y_i. The result is a small value (close to 0 but never 0) if $y_i < 0$ and a large value if y_i is large. eg • $\exp(55) = 7.69e{+}23$ (A very large value) • $\exp(-55) = 1.30e\text{-}24$ (A very small value close to 0) **<u>Note</u>**: $\exp(*)$ is just $\mathbf{e}^*$ where $\mathbf{e} = 2.718$, the Euler's number.
$\sum\limits_{j=1}^{n} \exp(y_j)$	A normalization term. It ensures that the values of output vector $S(y)_i$ sums to 1 for i-th class and each of them and each of them is in the range 0 and 1 which makes up a valid probability distribution.
n	Number of classes (possible outcomes)

*

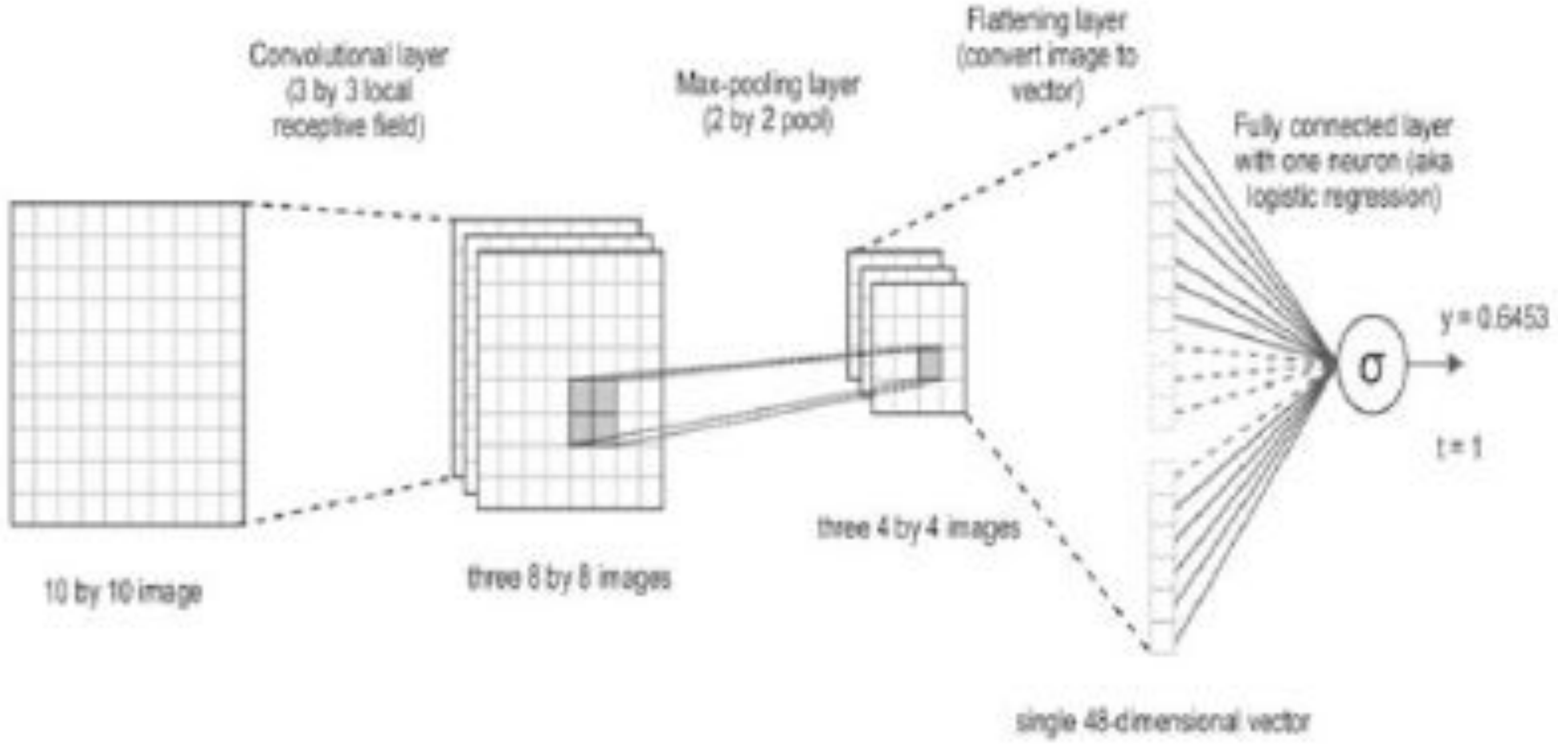

Input image
NN
Layers
Logits (L)
3.2
1.3
0.2
0.8
Softmax
$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^{n} \exp(y_j)}$
Output probabilities (P)
0.775
0.116
0.039
0.070
Classes
Dog
Cat
Horse
Cheetah

$$\begin{aligned}
\exp(3.2) &= 24.5325 \\
\exp(1.3) &= 3.6693 \\
\exp(0.2) &= 1.2214 \\
\exp(0.8) &= 2.2255
\end{aligned}$$

and therefore,

$$\begin{aligned}
S(3.2) &= \frac{\exp(3.2)}{\exp(3.2) + \exp(1.3) + \exp(0.2) + \exp(0.8)} \\
&= \frac{24.5325}{24.5325 + 3.6693 + 1.2214 + 2.2255} \\
&= 0.775
\end{aligned}$$

A convolutional neural network with a convolutional layer, a max-pooling layer, a flattening layer and a fully connected layer with one neuron



Convolutional layer (3 by 3 local receptive field)
Max-pooling layer (2 by 2 pool)
Flattening layer (convert image to vector)
Fully connected layer with one neuron (aka logistic regression)
σ
y = 0.6453
t = 1
10 by 10 image
three 8 by 8 images
three 4 by 4 images
single 48-dimensional vector

Loss Function

- Machines learn by means of a loss function.
- It’s a method of evaluating how well specific algorithm models the given data. If predictions deviates too much from actual results, loss function would cough up a very large number.
- Gradually, with the help of some optimization function, loss function learns to reduce the error in prediction. In this article we will go through several loss functions and their applications in the domain of machine/deep learning.

- There's no one-size-fits-all loss function to algorithms in machine learning. There are various factors involved in choosing a loss function for specific problem such as type of machine learning algorithm chosen, ease of calculating the derivatives and to some degree the percentage of outliers in the data set.

- Broadly, loss functions can be classified into two major categories depending upon the type of learning task we are dealing with — **Regression losses** and **Classification losses**. In classification, we are trying to predict output from set of finite categorical values i.e Given large data set of images of hand written digits, categorizing them into one of 0–9 digits. Regression, on the other hand, deals with predicting a continuous value for example given floor area, number of rooms, size of rooms, predict the price of room

Regression Losses

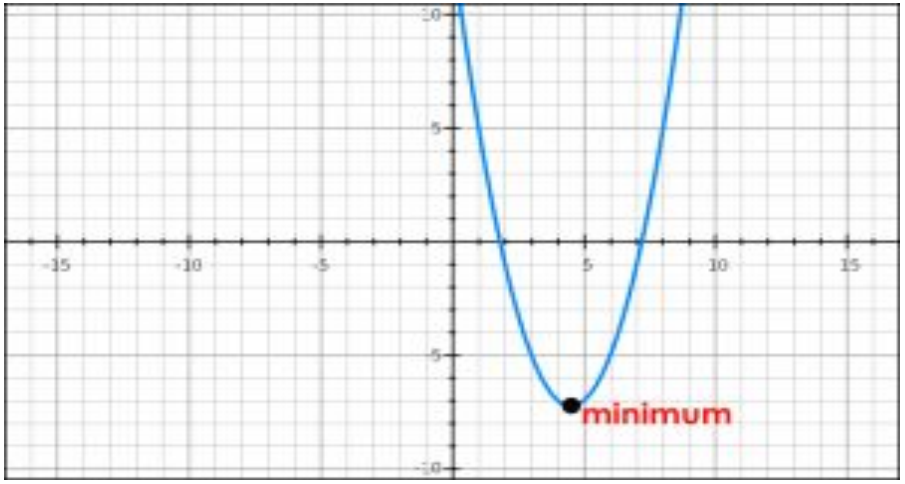
Mean Square Error/Quadratic Loss/L2 Loss

Mathematical formulation :-

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

As the name suggests, *Mean square error* is measured as the average of squared difference between predictions and actual observations. It’s only concerned with the average magnitude of error irrespective of their direction.

Let's talk a bit more about the MSE loss function. It is a positive quadratic function (of the form ax^2 + bx + c where a > 0). Remember how it looks graphically



A quadratic function only has a global minimum. Since there are no local minima, we will never get stuck in one. Hence, it is always guaranteed that Gradient Descent will converge (*if it converges at all*) to the global minimum.

- The MSE loss function penalizes the model for making large errors by squaring them. Squaring a large quantity makes it even larger, right? But there’s a caveat. This property makes the MSE cost function less robust to outliers. Therefore, ***it should not be used if our data is prone to many outliers.***

Binary Cross Entropy Loss

- 1**Let us start by understanding the term ‘entropy’**. Generally, we use entropy to indicate disorder or uncertainty. It is measured for a random variable X with probability distribution p(X):

$$S = \begin{cases} -\int p(x).\log p(x).dx, & \text{if } x \text{ is continuous} \\ -\sum_x p(x).\log p(x), & \text{if } x \text{ is discrete} \end{cases}$$

- The negative sign is used to make the overall quantity positive.

- A greater value of entropy for a probability distribution indicates a greater uncertainty in the distribution. Likewise, a smaller value indicates a more certain distribution.

- This makes binary cross-entropy suitable as a loss function – you want to minimize its value. We use binary cross-entropy loss for classification models which output a probability p.

- Probability that the element belongs to class 1 (or positive class) = p
- Then, the probability that the element belongs to class 0 (or negative class) = 1 – p
- Then, the cross-entropy loss for output label y (can take values 0 and 1) and predicted probability p is defined as:

$$L = -y * \log(p) - (1-y) * \log(1-p) = \begin{cases} -\log(1-p), & if\ y = 0 \\ -\log(p), & if\ y = 1 \end{cases}$$

- Then, the cross-entropy loss for output label y (can take values 0 and 1) and predicted probability p is defined as:
- This is also called Log-Loss. To calculate the probability p, we can use the sigmoid function. Here, z is a function of our input features:

$$S(z) = \frac{1}{1 + e^{-z}}$$

- loss function

The range of the sigmoid function is [0, 1] which makes it suitable for calculating probability.

Mean Absolute Error/L1 Loss
Mathematical formulation :-

- *Mean absolute error*, on the other hand, is measured as the average of sum of absolute differences between predictions and actual observations. Like MSE, this as well measures the magnitude of error without considering their direction. Unlike MSE, MAE needs more complicated tools such as linear programming to compute the gradients. Plus MAE is more robu make use of squa

$$MAE = \frac{\sum_{i=1}^{n} | y_i - \hat{y}_i |}{n}$$

- ***he MAE cost is more robust to outliers as compared to MSE***. However, handling the absolute or modulus operator in mathematical equations is not easy. I’m sure a lot of you must agree with this! We can consider this as a disadvantage of MAE.

- **Mean Bias Error**
- This is much less common in machine learning domain as compared to it's counterpart. This is same as MSE with the only difference that we don't take absolute values. Clearly there's a need for caution as positive and negative errors could cancel each other out. Although less accurate in practice, it could determine if the model has positive bias or negative bias
- ***Mathem***

$$MBE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{n}$$

Classification Losses

- **Hinge Loss/Multi class SVM Loss**
- In simple terms, the score of correct category should be greater than sum of scores of all incorrect categories by some safety margin (usually one). And hence hinge loss is used for maximum-margin classification, most notably for support vector machines classification, most notably for supp ugh not differentiable, it's a convex function which makes it easy to work with usual convex optimizers used in machine

$$SVMLoss = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$$

- Consider an example where we have three training examples and three classes to predict — Dog, cat and horse. Below the values predicted by our algorithm for each

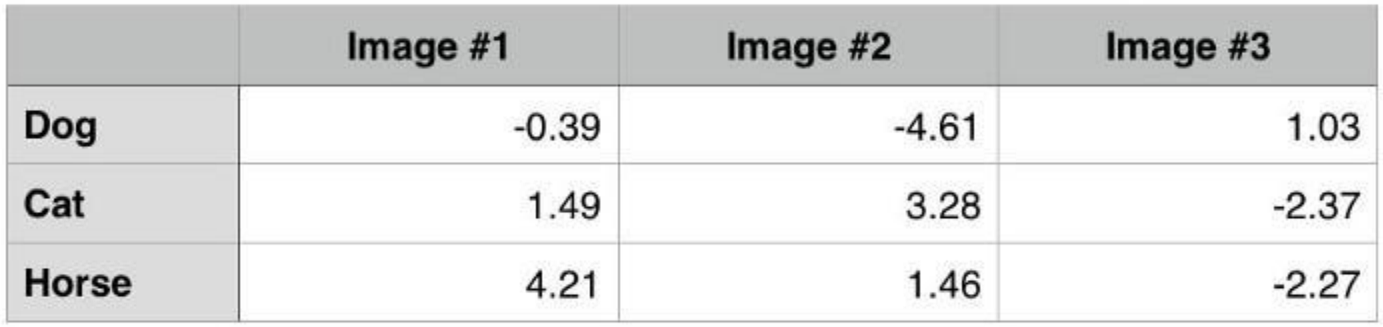


	Image #1	Image #2	Image #3
Dog	-0.39	-4.61	1.03
Cat	1.49	3.28	-2.37
Horse	4.21	1.46	-2.27

Computing hinge losses for all 3 training examples :-

## 1st training example

max(0, (1.49) - (-0.39) + 1) + max(0, (4.21) - (-0.39) + 1)

max(0, 2.88) + max(0, 5.6)

2.88 + 5.6

8.48 (High loss as very wrong prediction)

## 2nd training example

max(0, (-4.61) - (3.28)+ 1) + max(0, (1.46) - (3.28)+ 1)

max(0, -6.89) + max(0, -0.82)

0 + 0

*0 (Zero loss as correct prediction)***## 3rd training example**

max(0, (1.03) - (-2.27)+ 1) + max(0, (-2.37) - (-2.27)+ 1)

max(0, 4.3) + max(0, 0.9)

4.3 + 0.9

5.2 (High loss as very wrong prediction)

- **Cross Entropy Loss/Negative Log Likelihood**
- This is the most common setting for classification problems. Cross-entropy loss increases as the predicted probability diverges from the actual label.
- ***Mathematical formulation*** :-

$$CrossEntropyLoss = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

- Notice that when actual label is 1 (y(i) = 1), second half of function disappears whereas in case actual label is 0 (y(i) = 0) first half is dropped off. In short, we are just multiplying the log of the actual predicted probability for the ground truth class. An important aspect of this is that cross entropy loss penalizes heavily the predictions that are *confident but wrong*.

- Multi-Class Cross Entropy Loss
- The multi-class cross-entropy loss is a generalization of the Binary Cross Entropy loss. The loss for input vector X_i and the corresponding one-hot encoded target

$$L(X_i, Y_i) = -\sum_{j=1}^{c} y_{ij} * \log(p_{ij})$$

$where\ Y_i\ is\ one-hot\ encoded\ target\ vector\ (y_{i1}, y_{i2}, \ldots, y_{ic}),$

$$y_{ij} = \begin{cases} 1, & if\ i_{th}\ element\ is\ in\ class\ j \\ 0, & otherwise \end{cases}$$

$$p_{ij} = f(X_i) = Probability\ that\ i_{th}\ element\ is\ in\ class\ j$$