Introduction to Machine Learning in Geosciences GEO371T/GEO398D.1

Convolutional Neural Networks

Mrinal K. Sen
Geosciences
UT Austin

November 14, 2023

Content

- Neural Network
- Deep Learning
- Convolution
- Tensors
- Convolutional Neural Networks

A Mathematcal Model of a Neuron

- Neural networks define functions of the inputs (hidden features), computed by neurons.
- Artificial neurons are called **units**.

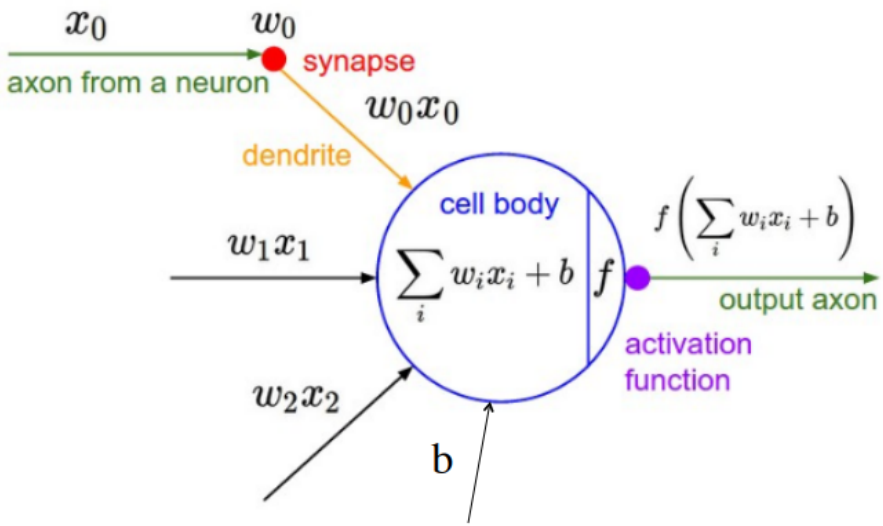


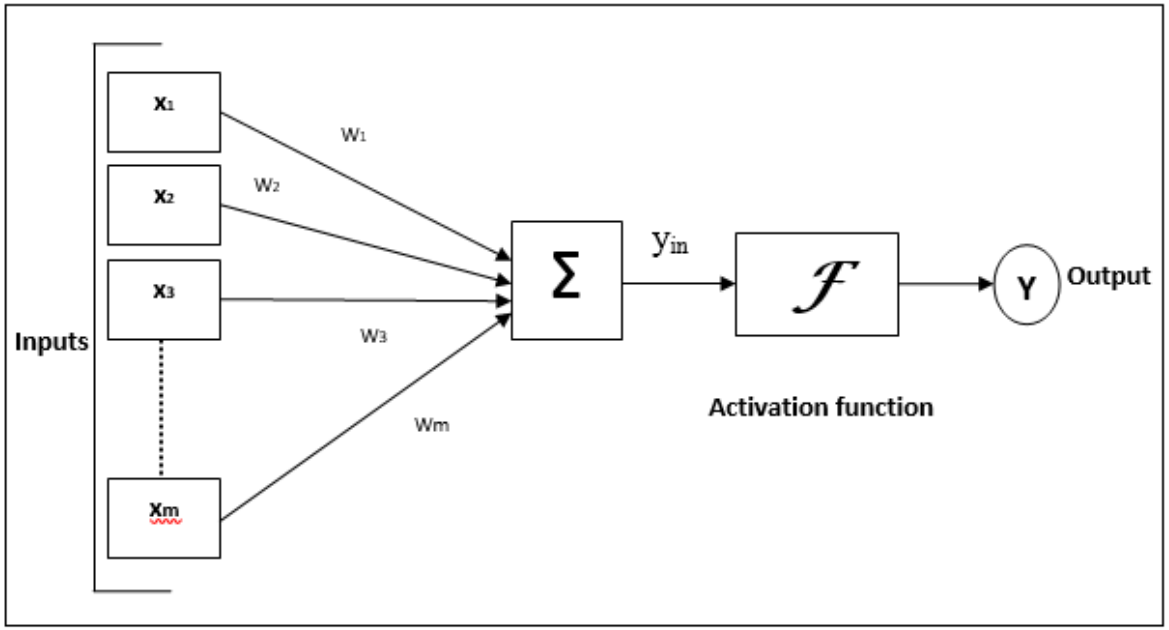
Figure: A mathematical model of the neuron in a neural network

[Pic credit: http://cs231n.github.io/neural-networks-1/]

A Mathematcal Model of a Neuron

Model of Artificial Neural Network

The following diagram represents the general model of ANN followed by its processing.



$$y_{in} = \sum_{i}^{m} x_i w_i.$$

$$Y = f(y_{in}).$$

Network Topology

Feedforward ANN

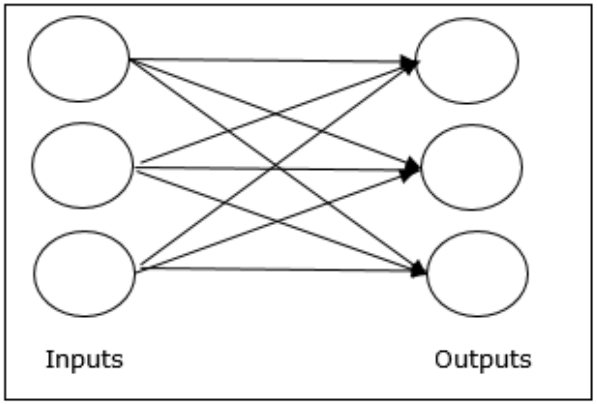


Figure: Single Layer

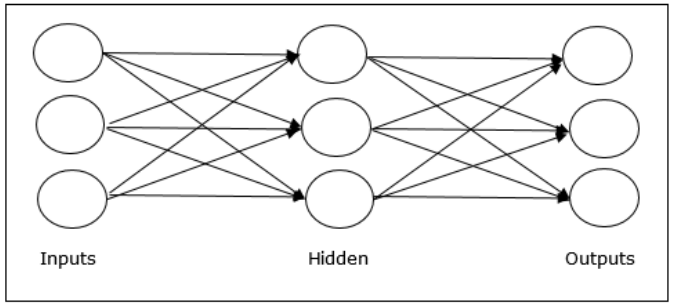
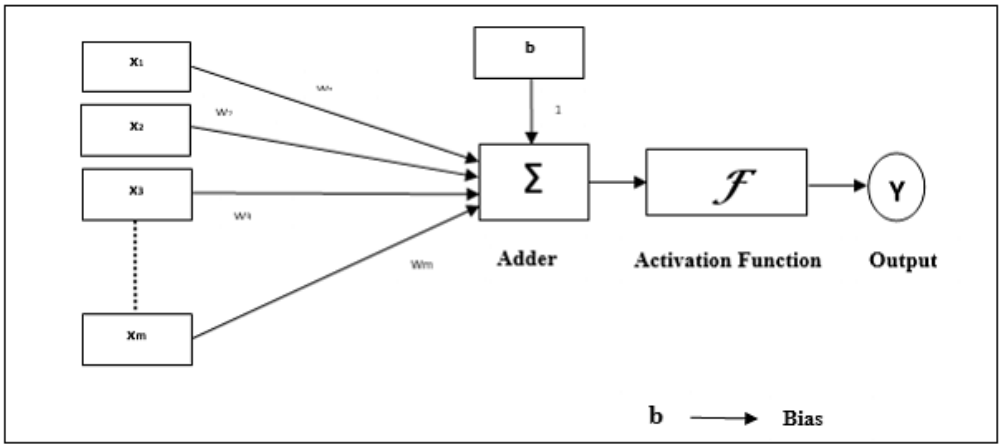


Figure: Multilayer

Perceptron

- Developed by Frank Rosenblatt by using McCulloch and Pitts model
- Basic operational unit of artificial neural networks.
- Employs supervised learning rule and is able to classify the data into two classes.

- https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_supervised_learning.htm



Perceptron thus has the following three basic elements −

- **Links** − It would have a set of connection links, which carries a weight including a bias always having weight 1.
- **Adder** − It adds the input after they are multiplied with their respective weights.
- **Activation function** − It limits the output of neuron. The most basic activation function is a Heaviside step function that has two possible outputs. This function returns 1, if the input is positive, and 0 for any negative input.

Perceptron LEARNING RULE

The Perceptron Learning Algorithm

Let

$$\mathcal{D} = (\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \langle \mathbf{x}^{[2]}, y^{[2]} \rangle, ..., \langle \mathbf{x}^{[n]}, y^{[n]} \rangle) \in (\mathbb{R}^m \times \{0, 1\})^n$$

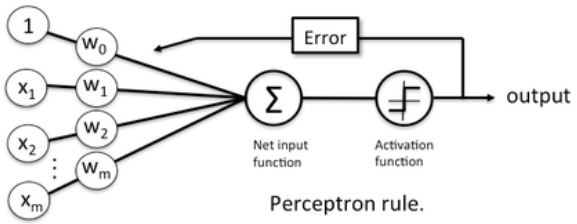
1. Initialize $\mathbf{w} := 0^m$ (assume notation where weight incl. bias)
2. For every training epoch:
   A. For every $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$:
      (a) $\hat{y}^{[i]} := \sigma\left(\mathbf{x}^{[i]T}\mathbf{w}\right)$
      (b) $\text{err} := (y^{[i]} - \hat{y}^{[i]})$
      (c) $\mathbf{w} := \mathbf{w} + err \times \mathbf{x}^{[i]}$

Sebastian Raschka STAT 479: Deep Learning SS 2019 28

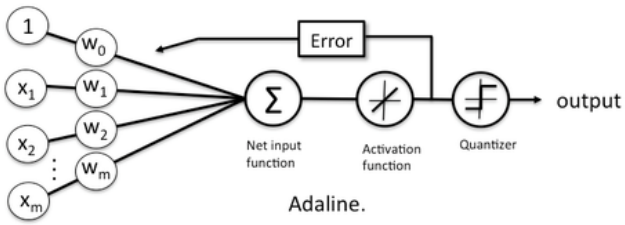
- Output value is the class level predicted by the unit step function!

ADALINE

- Particularly Interesting - illustrates the key concepts of defininining and minimizing continuous cost functions!
- Groundwork for more advanced ML methods such as *logistic regression, SVM, and regression models*.
- Perception uses a unit step function while Adaline uses a linear activaion function
- The Adaline (Adaptive Linear Element) and the Perceptron are both linear classifiers when considered as individual units. They both take an input, and based on a threshold, output e.g. either a 0 or a 1.
- The main difference between the two, is that a Perceptron takes that binary response (like a classification result) and computes an error used to update the weights, whereas an Adaline uses a continous response value to update the weights (so before the binarized output is produced).
- The fact that the Adaline does this, allows its updates to be more repesentative of the actual error, before it is thresholded, which in turn allows a model to converge more quickly.



Perceptron rule.



Adaline.

Minimizing Cost Function with Gradient Descent

- The Cost function: $J(\mathbf{W}) = \frac{1}{2}\sum_i (y^i - \phi(x^i))^2$.
- Using GD, we update weights by taking a step in the opposite direction of the gradient, $\nabla J(\mathbf{W})$:

$$\mathbf{W} \leftarrow \mathbf{W} + \Delta\mathbf{W}.$$

- The weight change $\nabla J(\mathbf{W})$ is defined as the negative of the gradient multiplied by the learning rate η

$$\Delta\mathbf{W} = -\eta\nabla J(\mathbf{W}).$$

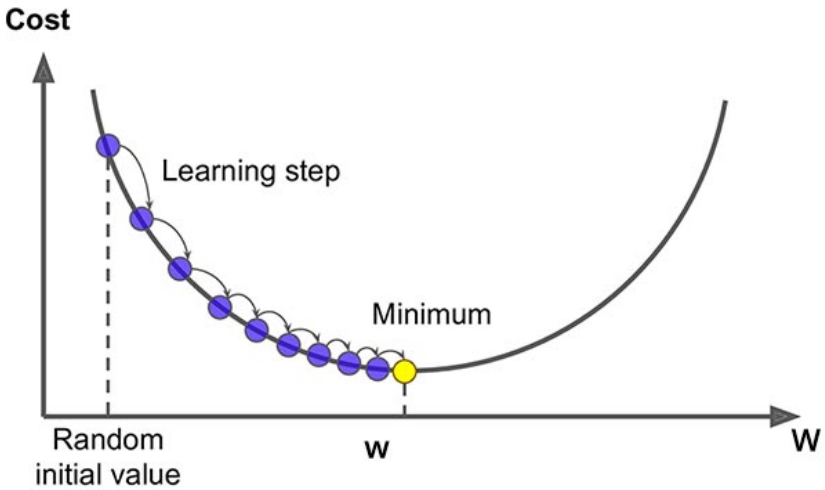
$$\frac{\partial J}{\partial w_i} = -\sum_i (y^i - \phi(x^i))x_j^i.$$

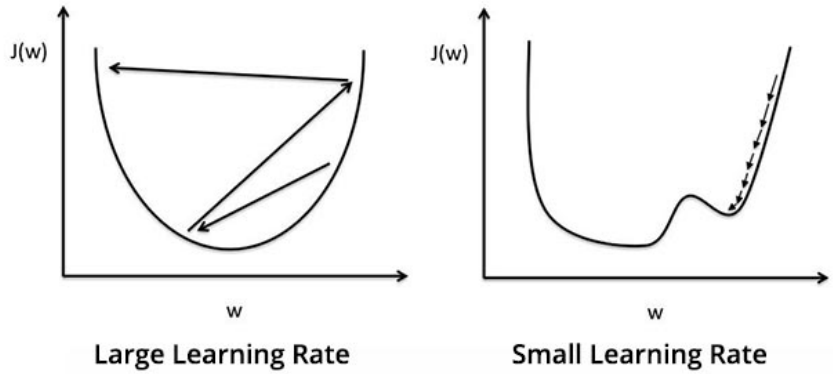
$$\Delta w_j = -\eta\frac{\partial J}{\partial w_i} = \eta\sum_i (y^i - \phi(x^i))x_j^i.$$

Minimizing Cost Function with Gradient Descent

Hyperparameters:

- The learning rate η
- The number of epochs $=$ *n_iters*

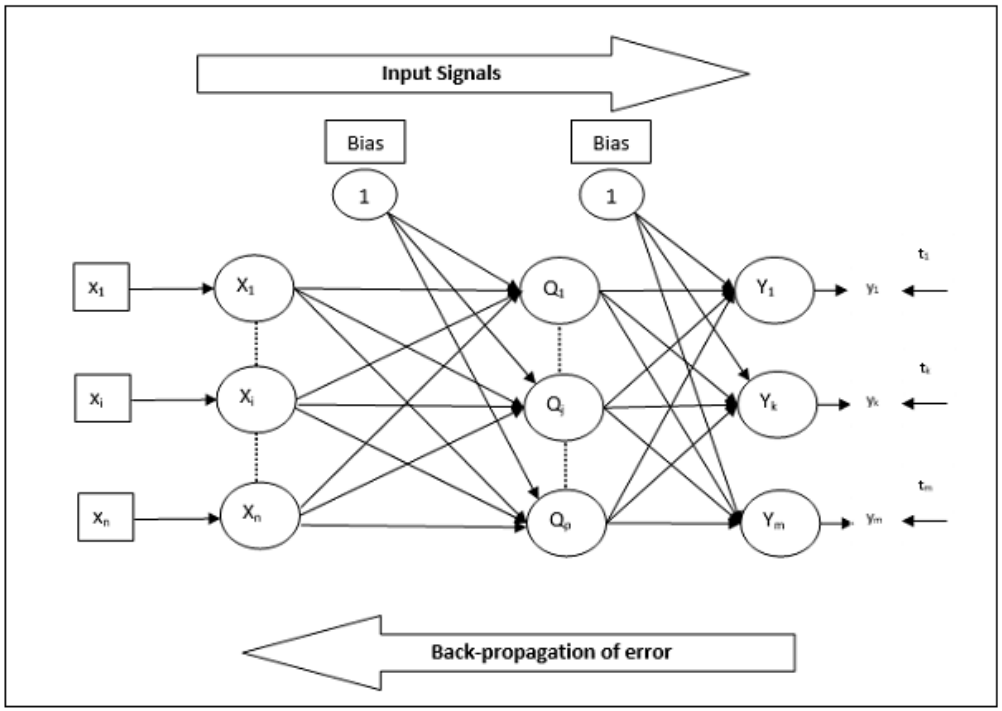




ANN/DNN

Goals:

- A learning rule that is more robust than the perception
- Combine multiple Neurons and layers of neurons (deep neural nets) to learn more complex decision boundaries (most real world problems are not linear)
- Handle multiple categories (not just binary) in classification.



Back Propagation Neural Network

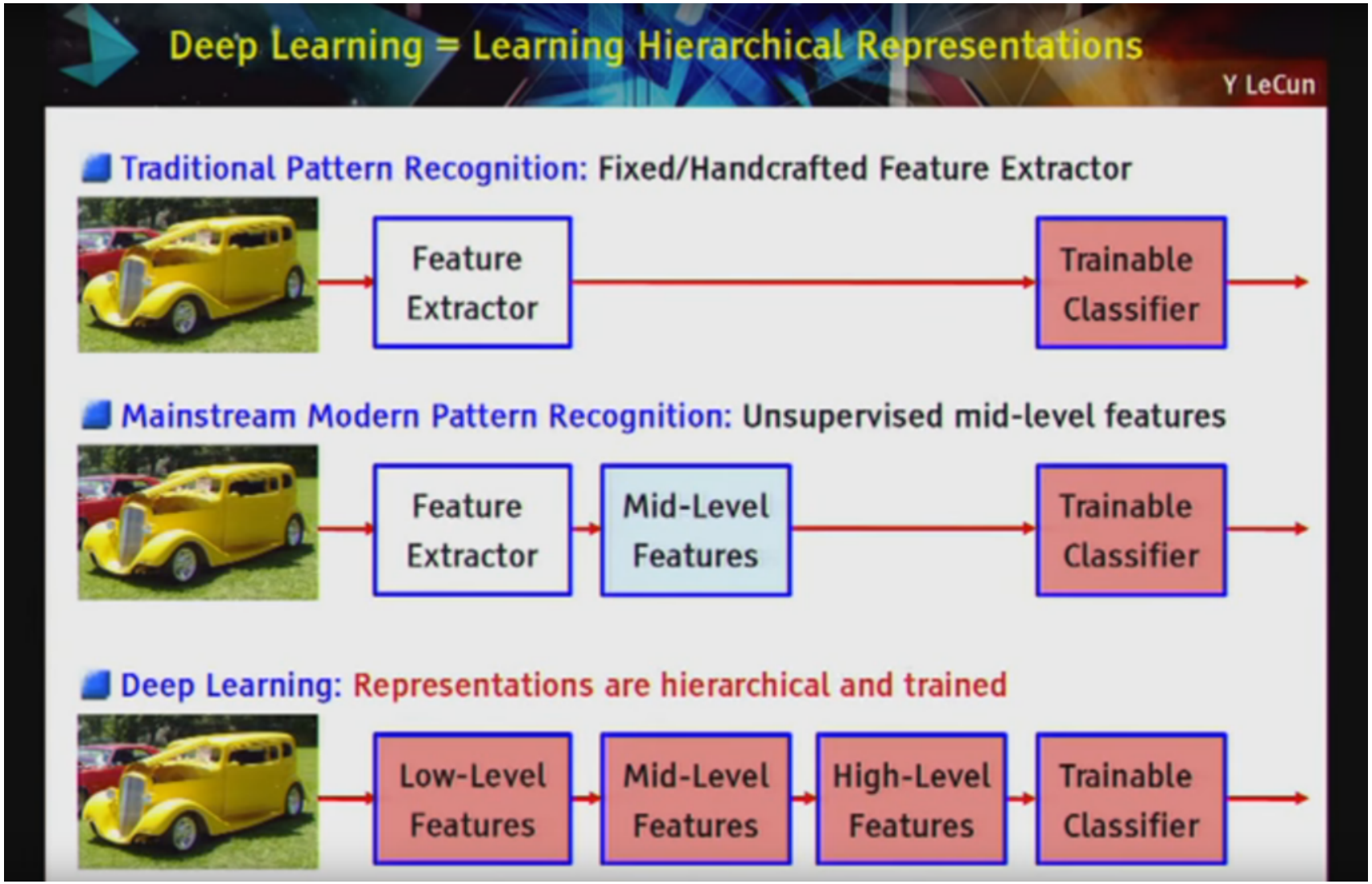
What was wrong with backpropagation in 1986?

- Very small labeled datasets
- Very slow computers
- Weight initialization
- Non-linearity

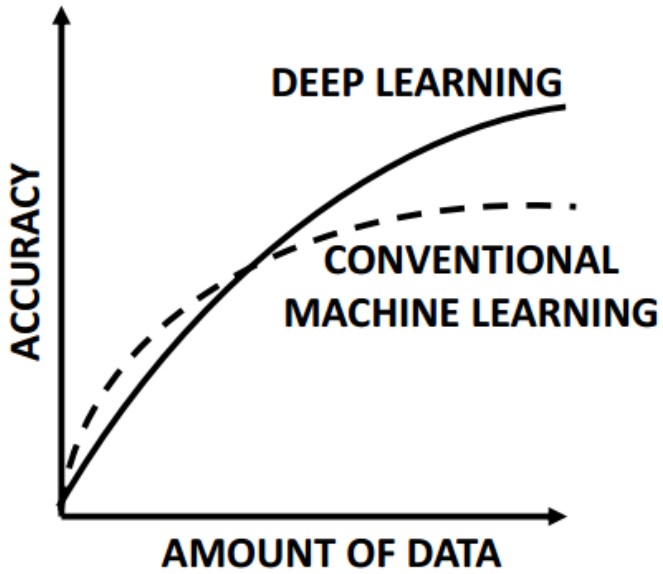
Note

- Better results with
  - more data
  - larger models
  - more computation
  - better algorithms
  - new insights
  - improved techniques
- What changed since 2000?
  - Lots of data and computational power!
- **There are indeed settings where you are better off using a conventional machine learning technique like a random forest.**

Deep Learning



Deep Learning



- For smaller data sets, traditional machine learning methods often provide slightly better performance.
- Traditional models often provide more choices, interpretable insights, and ways to handcraft features.
- For larger data sets, deep learning methods tend to dominate.

Introduction

- Convolutional neural networks (CNNs) were inspired by early findings in the study of biological vision.
- They have since become successful tools in computer vision and state-of-the-art models of both neural activity and behavior on visual tasks.
- The history of convolutional neural networks threads through both neuroscience and artificial intelligence. Like artificial neural networks in general, they are an example of brain-inspired ideas coming to fruition through an interaction with computer science and engineering.
-

https://arxiv.org/abs/2001.07092

Introduction

Wiesel : discovered two major cell types in the primary visual cortex (V1) of cats!

- The first type, the simple cells, respond to bars of light or dark when placed at specific spatial locations. Each cell has an orientation of the bar at which it fires most, with its response falling off as the angle of the bar changes from this preferred orientation (creating an orientation 'tuning curve').
- The second type, complex cells, have less strict response profiles; these cells still have preferred orientations but can respond just as strongly to a bar in several different nearby locations. Hubel and Wiesel concluded that these complex cells are likely receiving input from several simple cells, all with the same preferred orientation but with slightly different preferred locations.
- Fukushima transformed Hubel and Wiesel's findings into a functioning model of the visual system. This model, the Neocognitron, is the precursor to modern convolutional neural networks.

https://arxiv.org/abs/2001.07092

Introduction

- It contains two main cell types. The S-cells are named after simple cells and replicate their basic features: specifically, a 2-D grid of weights is applied at each location in the input image to create the S-cell responses.
- A "plane" of S-cells thus has a retinotopic layout with all cells sharing the same preferred visual features and multiple planes existing at a layer. The response of the C-cells (named after complex cells) is a nonlinear function of several S-cells coming from the same plane but at different locations.
- After a layer of simple and complex cells representing the basic computations of V1, the Neocognitron simply repeats the process again. That is, the output of the first layer of complex cells serves as the input to the second simple cell layer, and so on. With several repeats, this creates a hierarchical model that mimics not just the operations of V1 but the ventral visual pathway as a whole. The network is "self-organized", meaning weights change with repeated exposure to unlabeled images.

Introduction

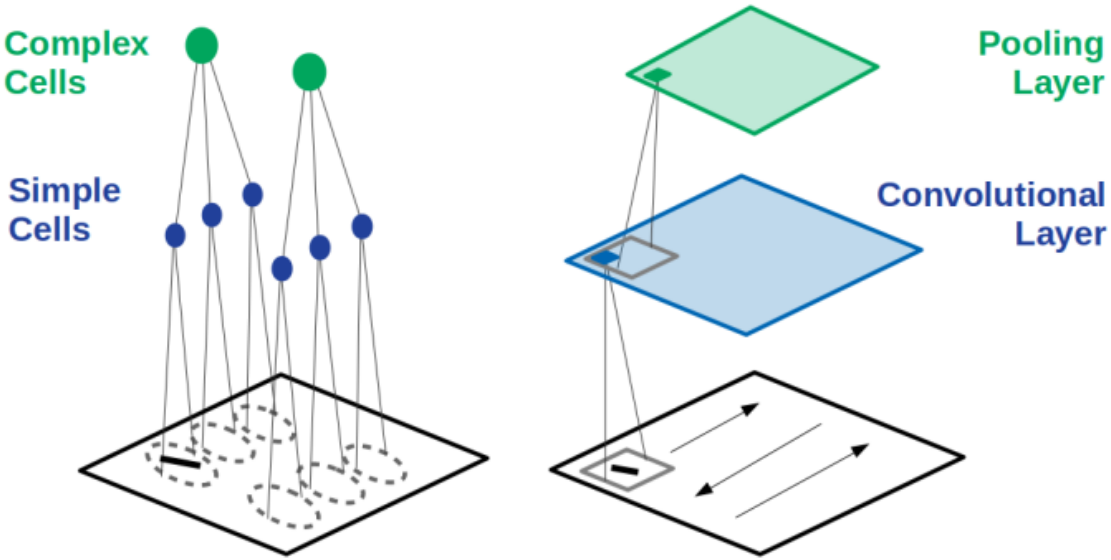
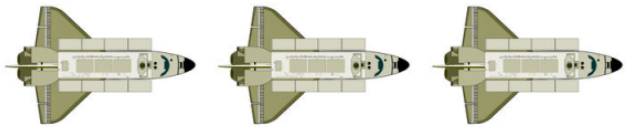


Figure 1. The relationship between components of the visual system and the base operations of a convolutional neural network. Hubel and Wiesel discovered that simple cells (left, blue) have preferred locations in the image (dashed ovals) wherein they respond most strongly to bars of particular orientation. Complex cells (green) receive input from many simple cells and thus have more spatially invariant responses. These operations are replicated in a convolutional neural network (right). The first convolutional layer (blue) is produced by applying a convolution operation to the image. Specifically, the application of a small filter (gray box) to every location in the image creates a feature map. A convolutional layer has as many feature

https://arxiv.org/abs/2001.07092

Convolution



x_0 x_1 x_2

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

input
filter
convolution

- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals
- Now suppose our sensor is noisy
- To obtain a less noisy estimate we would like to average several measurements
- More recent measurements are more important so we would like to take a weighted average

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution

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

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W	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

S					1.80		

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t
- Here the input (and the kernel) is one dimensional
- Can we use a convolutional operation on a 2D input also?

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution

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

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W	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

S	1.80	1.96	2.11	2.16	2.28	2.42

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t
- Here the input (and the kernel) is one dimensional
- Can we use a convolutional operation on a 2D input also?

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i-a,j-b} K_{a,b} I_{i+a,j+b} K_{a,b}$$

- We can think of images as 2D inputs
- We would now like to use a 2D filter $(m \times n)$
- First let us see what the 2D formula looks like
- This formula looks at all the preceding neighbours $(i-a, j-b)$
- In practice, we use the following formula which looks at the succeeding neighbours

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution

Input

a	b	c	d
e	f	g	h
i	j	k	ℓ

Kernel

w	x
y	z

Output

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

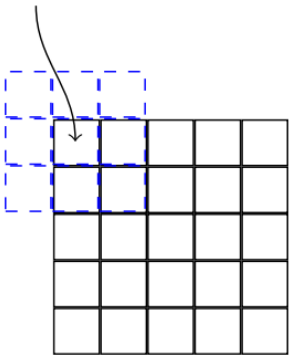
- Let us apply this idea to a toy example and see the results

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution

$$S_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a,j-b} K_{\frac{m}{2}+a,\frac{n}{2}+b}$$

pixel of interest



- For the rest of the discussion we will use the following formula for convolution
- In other words we will assume that the kernel is centered on the pixel of interest
- So we will be looking at both preceeding and succeeding neighbors

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution

$$* \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} =$$

blurs the image

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution

$$* \begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix} =$$

sharpens the image

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

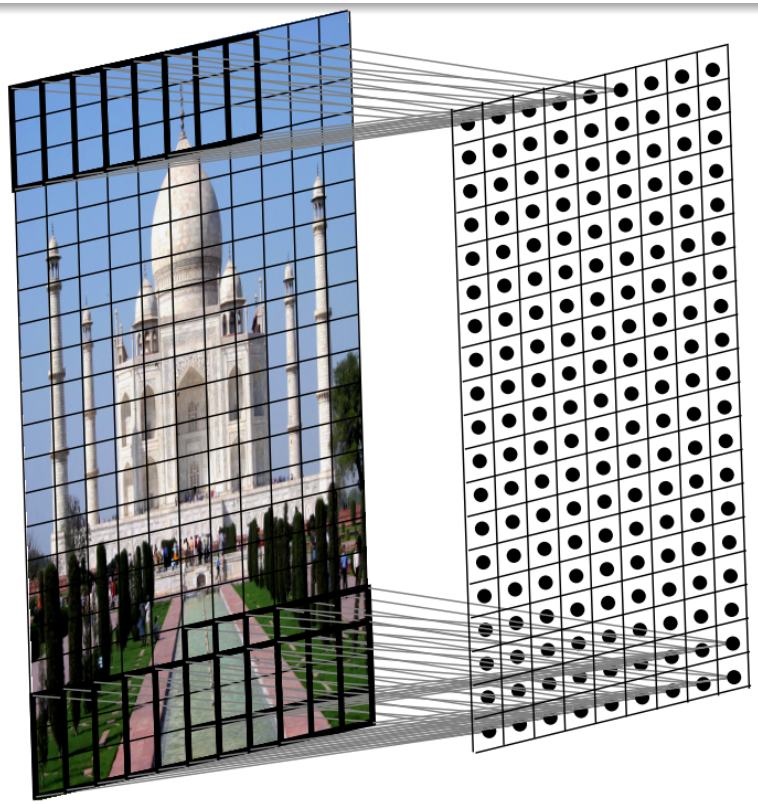
Convolution

$$* \begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix} =$$

detects the edges

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution



- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output
- The resulting output is called a feature map.
- We can use multiple filters to get multiple feature maps.

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input
- In the 2D case, we slide a two dimenstional filter over a two dimensional output
- What would happen in the 3D case?

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

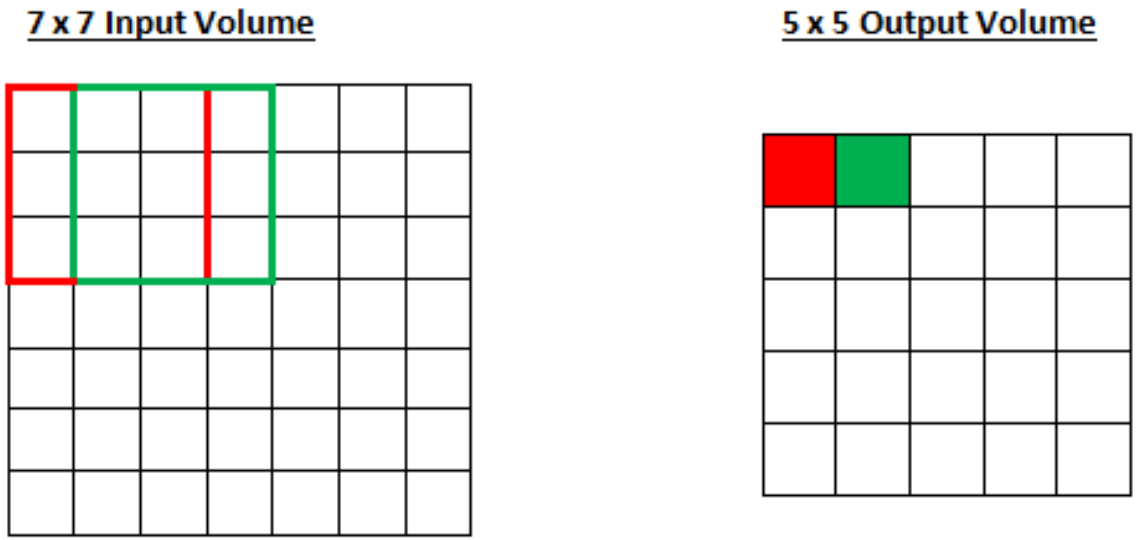
A	B	C	B	A	B	C

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

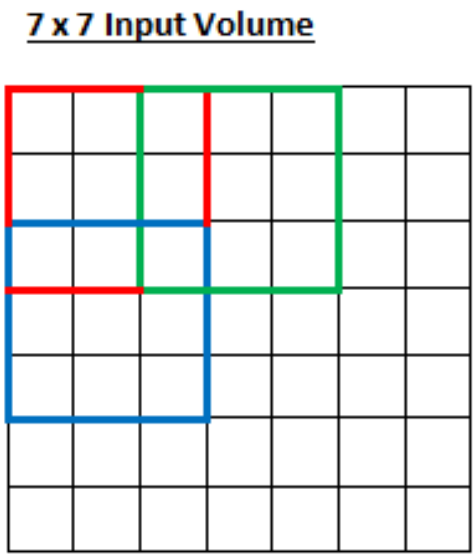
Convolution

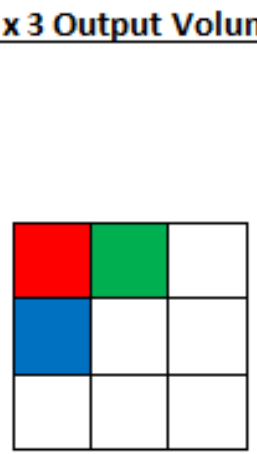
- **Kernel Size K**: The size of the sliding kernel or filter.
- **Stride Length S**: Defines how much is the kernel slid before the dot product is carried out to generate the output pixel
- **Padding P**: The frame size of zeros inserted around the input feature map.

Convolution

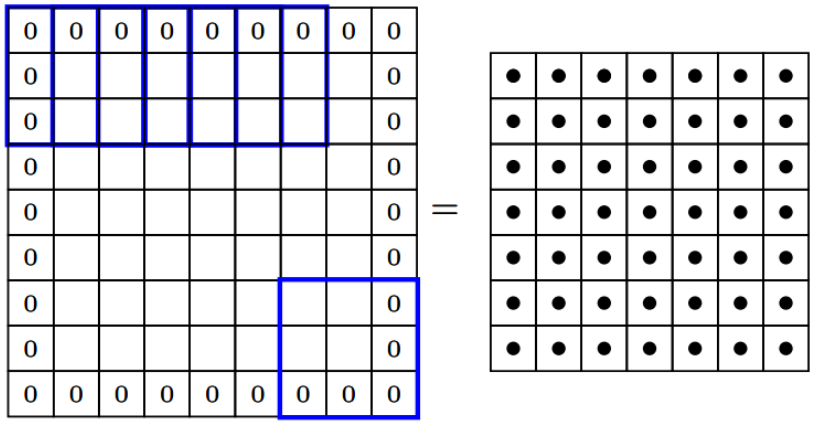


Convolution





Convolution



- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad P = 1 with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right

We now have,

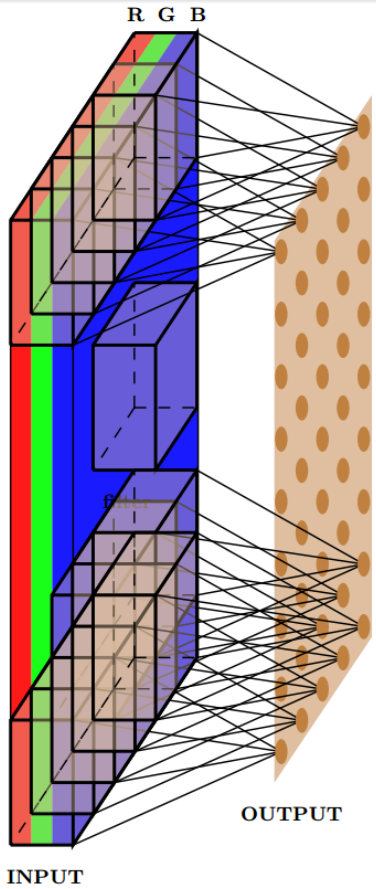
$$W_2 = W_1 - F + 2P + 1$$

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

We will refine this formula further

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution



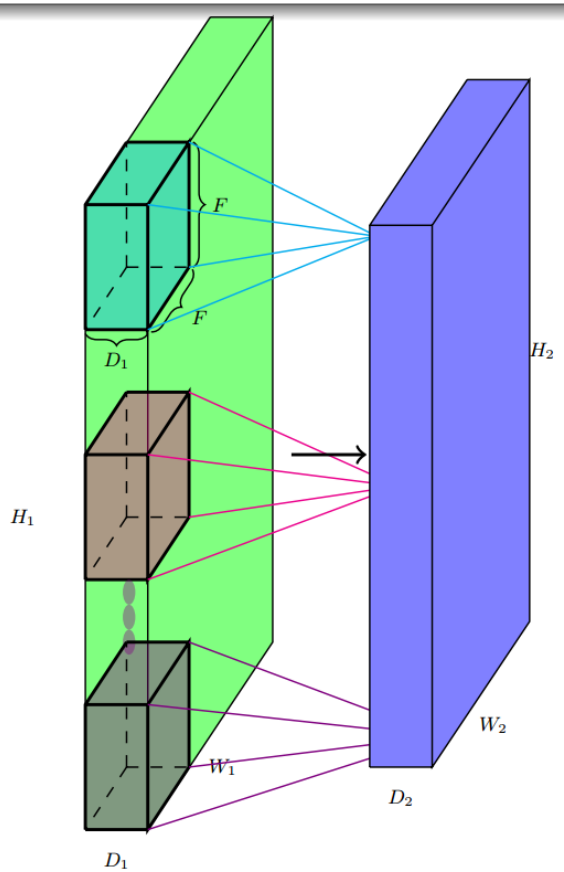
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation
- Note that in this lecture we will assume that the filter always extends to the depth of the image
- In effect, we are doing a 2D convolution operation on a 3D input (because the filter moves along the height and the width but not along the depth)
- As a result the output will be 2D (only width and height, no depth)
- Once again we can apply multiple filters to get multiple feature maps

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Tensors

- $\mathbf{x} \in \mathbb{R}^D$: a column vector with D elements.
- $\mathbf{X} \in \mathbb{R}^{HXW}$: a matrix with H rows and W columns!
- $\mathbf{x} \in \mathbb{R}^{HXWXD}$: a third order tensor. $D = 1$ reduces to a matrix!
- A color image is in fact an order 3 tensor. An image with H rows and W columns is a tensor with size $HW3$: if a color image is stored in the RGB format, it has 3 channels (for R, G and B, respectively), and each channel is a HW matrix (second order tensor) that contains the R (or G, or B) values of all pixels.
- Tensors are essential in CNN. The input, intermediate representation, and parameters in a CNN are all tensors. Tensors with order higher than 3 are also widely used in CNNs.

Convolution



- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)
- The number of filters K
- The spatial extent (F) of each filter (the depth of each filter is same as the depth of each input)
- The output is $W_2 \times H_2 \times D_2$ (we will soon see a formula for computing W_2, H_2 and D_2)

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

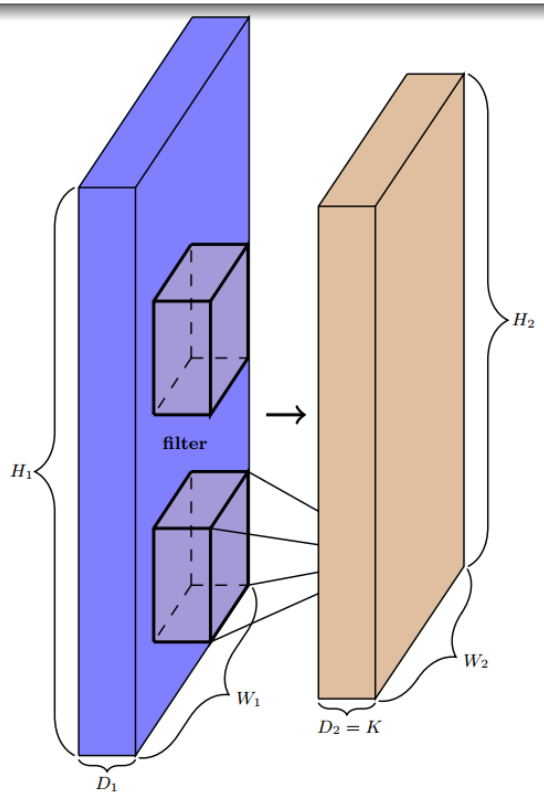
Convolution

$$\textit{In general, } W_2 = W_1 - F + 1$$
$$H_2 = H_1 - F + 1$$
$$\textit{We will refine this formula further}$$

- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller output now

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution



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

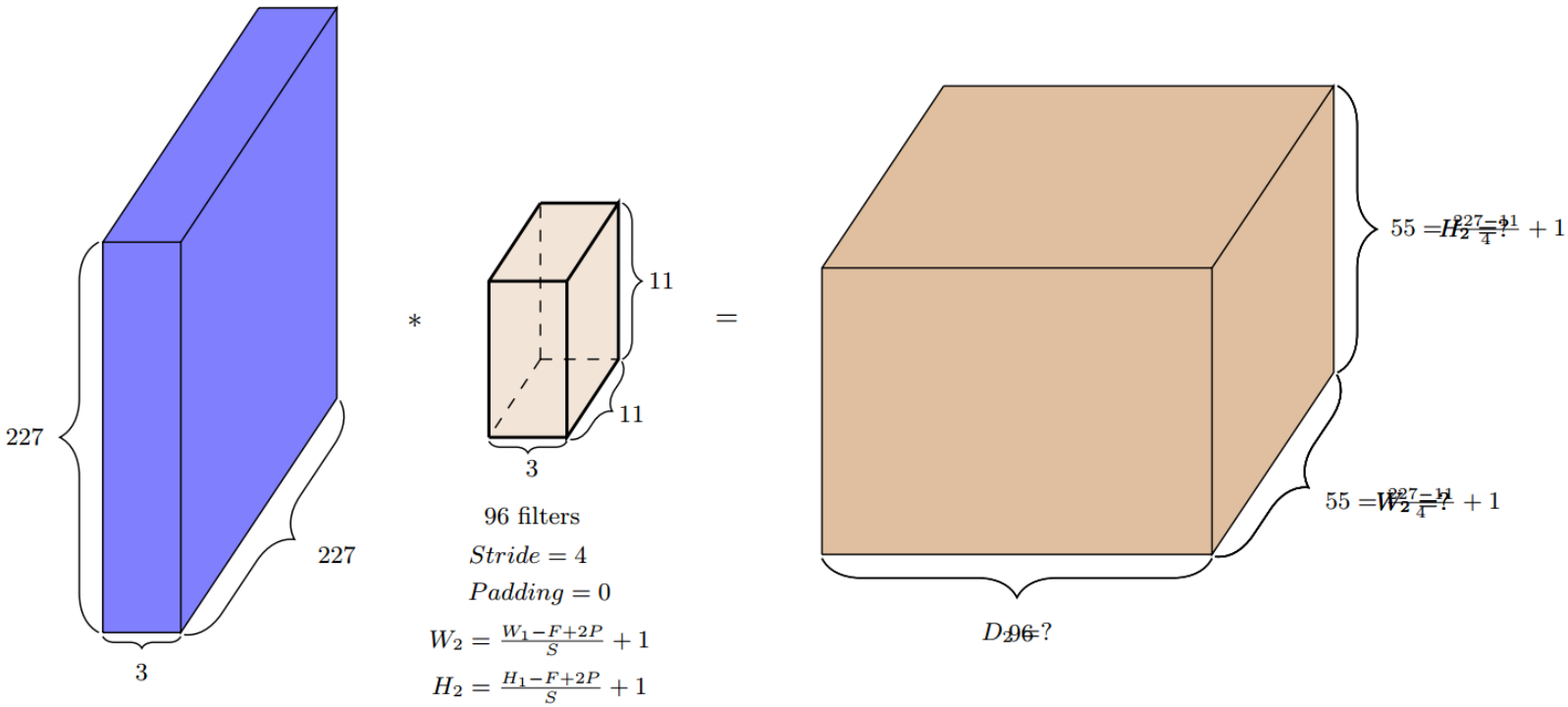
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

$$D_2 = K$$

- Finally, coming to the depth of the output.
- Each filter gives us one 2D output.
- K filters will give us K such 2D outputs
- We can think of the resulting output as $K \times W_2 \times H_2$ volume
- Thus $D_2 = K$

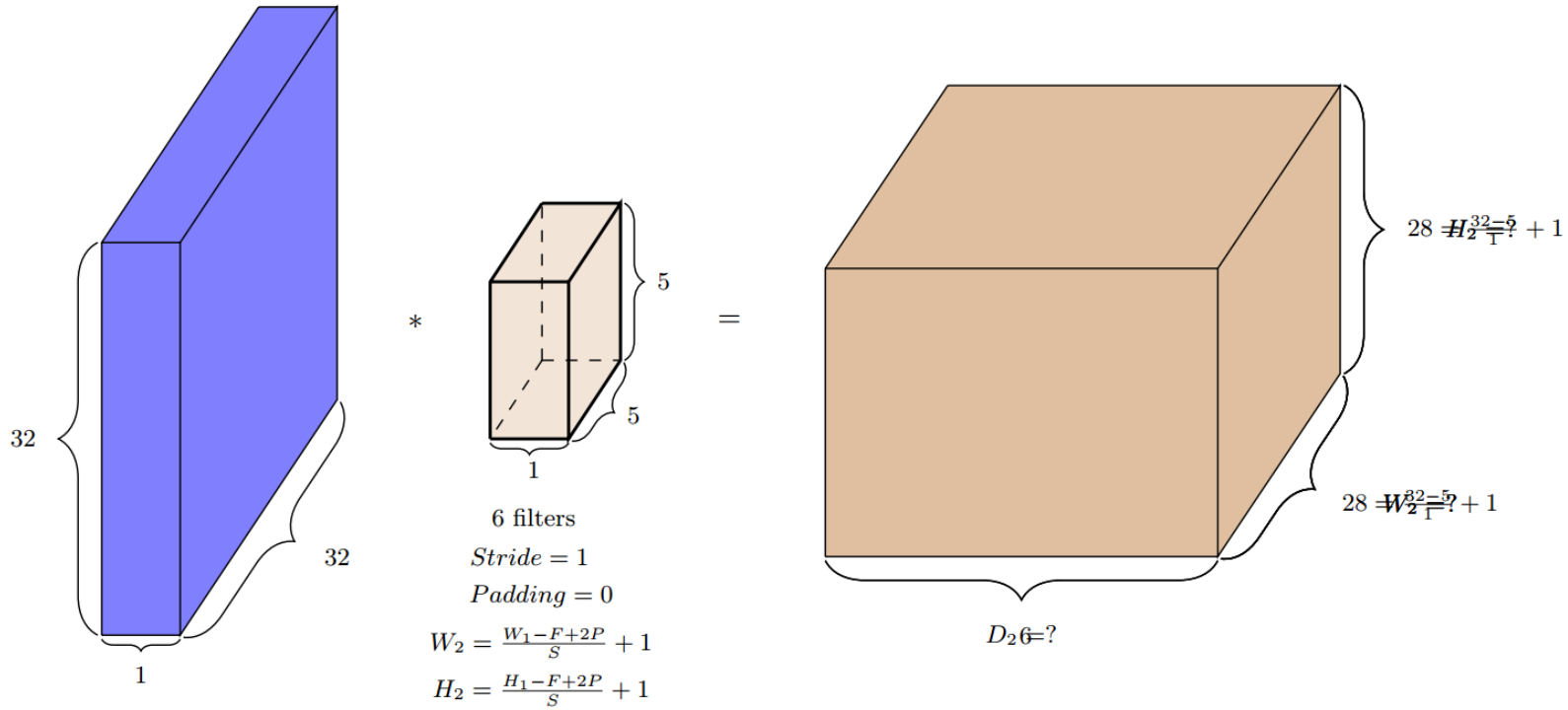
https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution



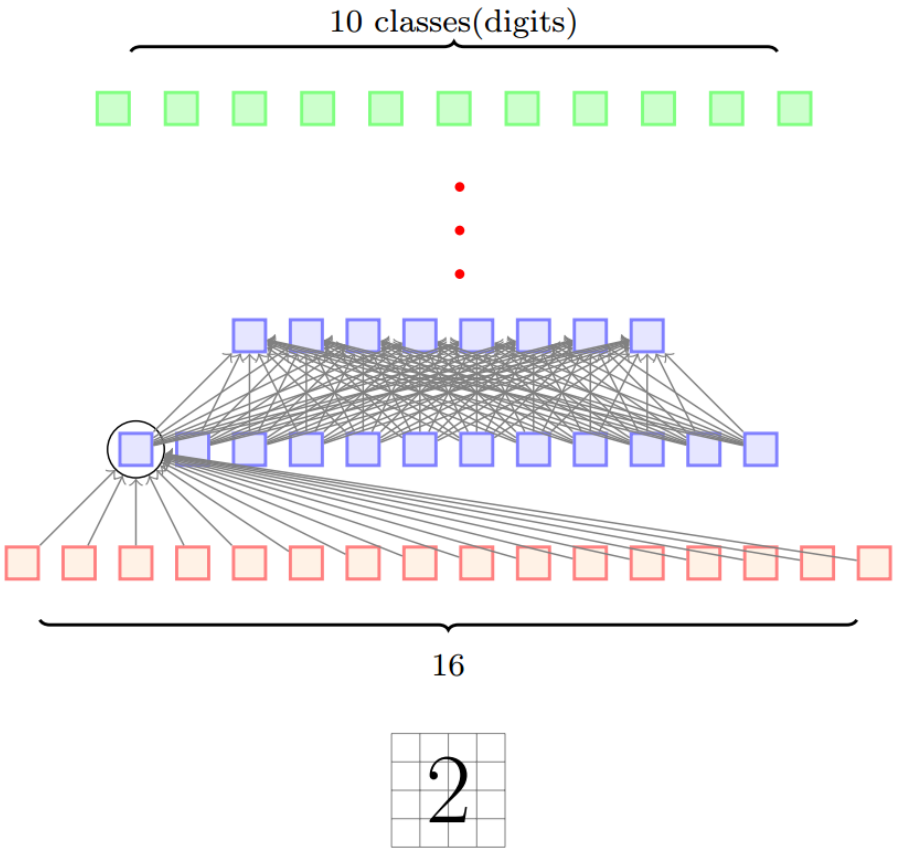
https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution



https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

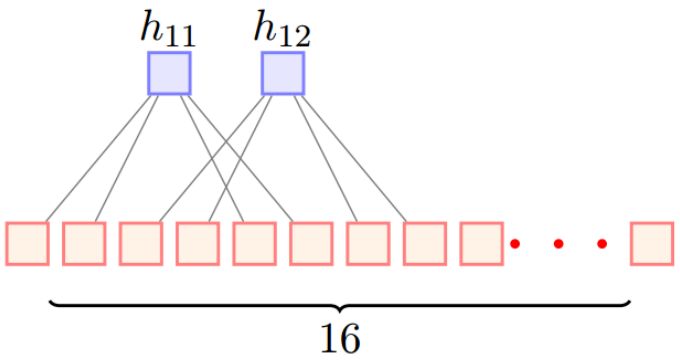
Convolution

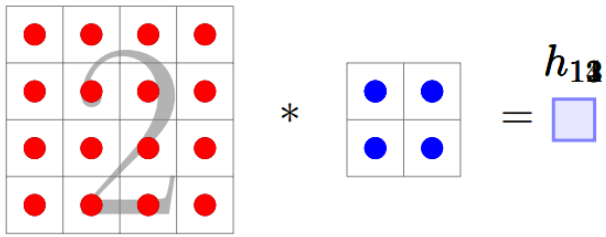


- This is what a regular feed-forward neural network will look like
- There are many dense connections here
- For example all the 16 input neurons are contributing to the computation of h_{11}
- Contrast this to what happens in the case of convolution

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution

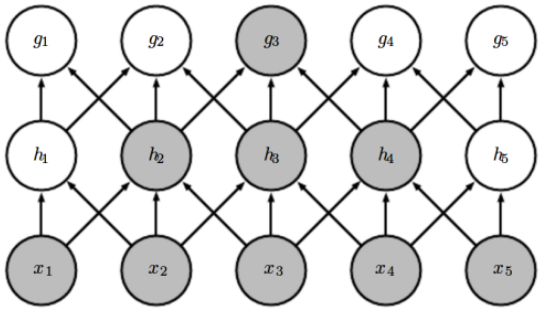




- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}
- The connections are much sparser
- We are taking advantage of the structure of the image(interactions between neighboring pixels are more interesting)
- This **sparse connectivity** reduces the number of parameters in the model

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution

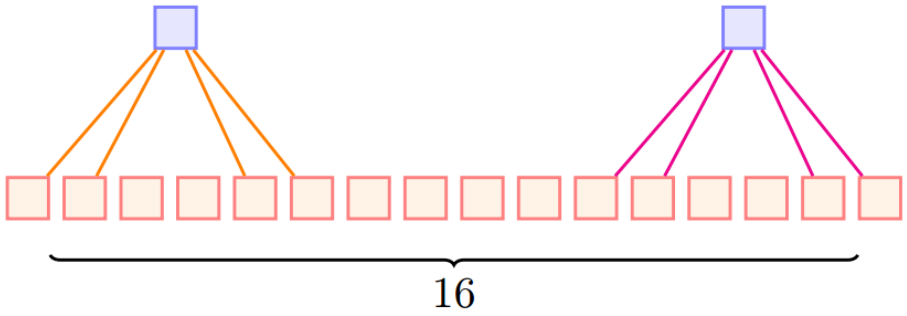


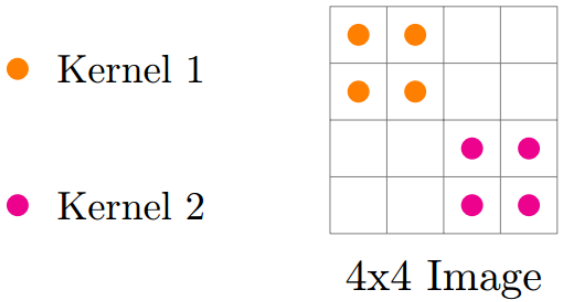
- But is sparse connectivity really good thing ?
- Aren't we losing information (by losing interactions between some input pixels)
- Well, not really
- The two highlighted neurons (x_1 & x_5)* do not interact in *layer* 1
- But they indirectly contribute to the computation of g_3 and hence interact indirectly

* Goodfellow-et-al-2016

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution

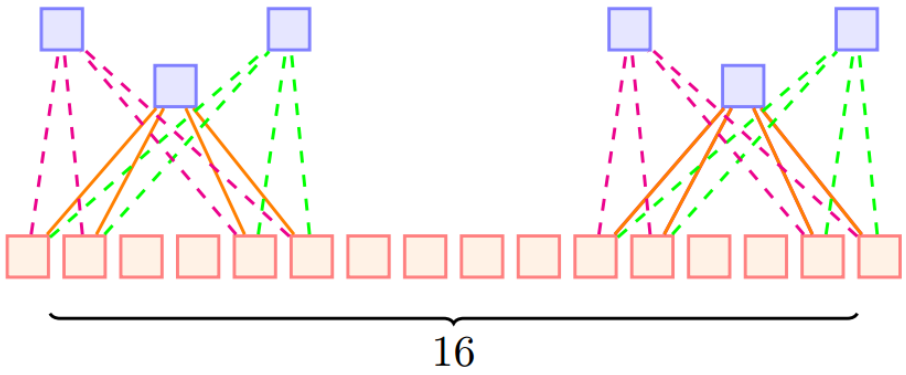




- Another characteristic of CNNs is **weight sharing**
- Consider the following network
- Do we want the kernel weights to be different for different portions of the image?
- Imagine that we are trying to learn a kernel that detects edges
- Shouldn't we be applying the same kernel at all the portions of the image?

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

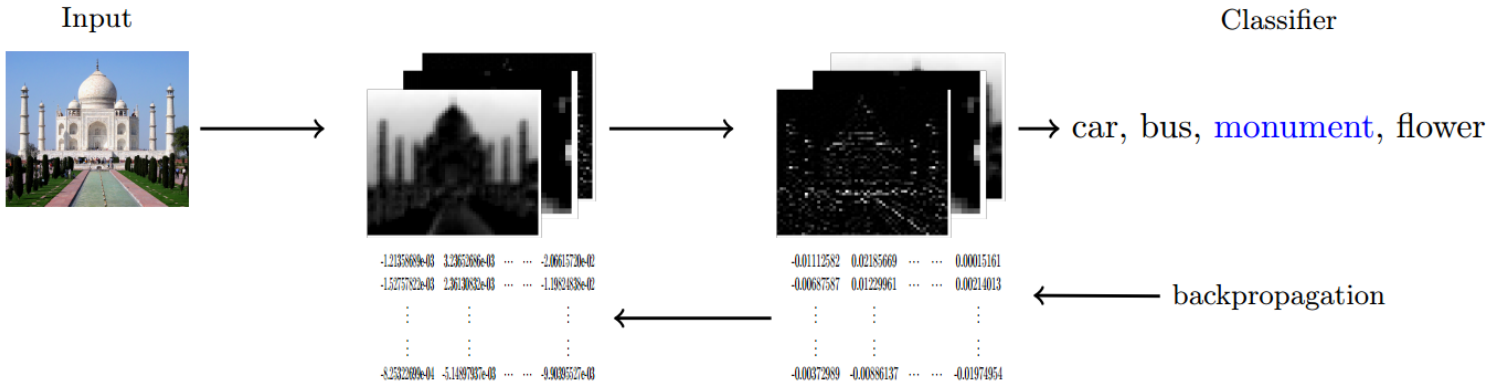
Convolution



- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?
- No, we can have many such kernels but the kernels will be shared by all locations in the image
- This is called "weight sharing"

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution



- **Can we learn multiple layers of meaningful kernels/filters in addition to learning the weights of the classifier?**
- Yes, we can !
- Simply by treating these kernels as parameters and learning them in addition to the weights of the classifier (using back propagation)
- Such a network is called a Convolutional Neural Network.

https://www.cse.iitm.ac.in/ miteshk/CS7015/Slides/Handout/Lecture11.pdf

Convolution

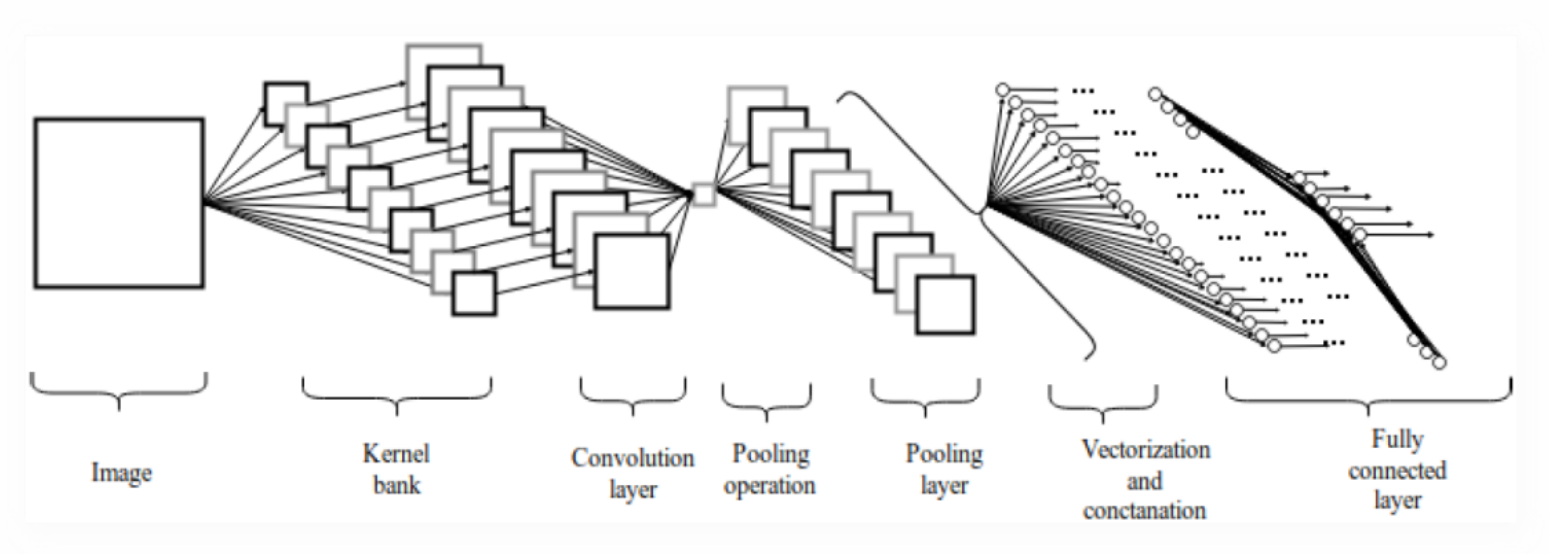
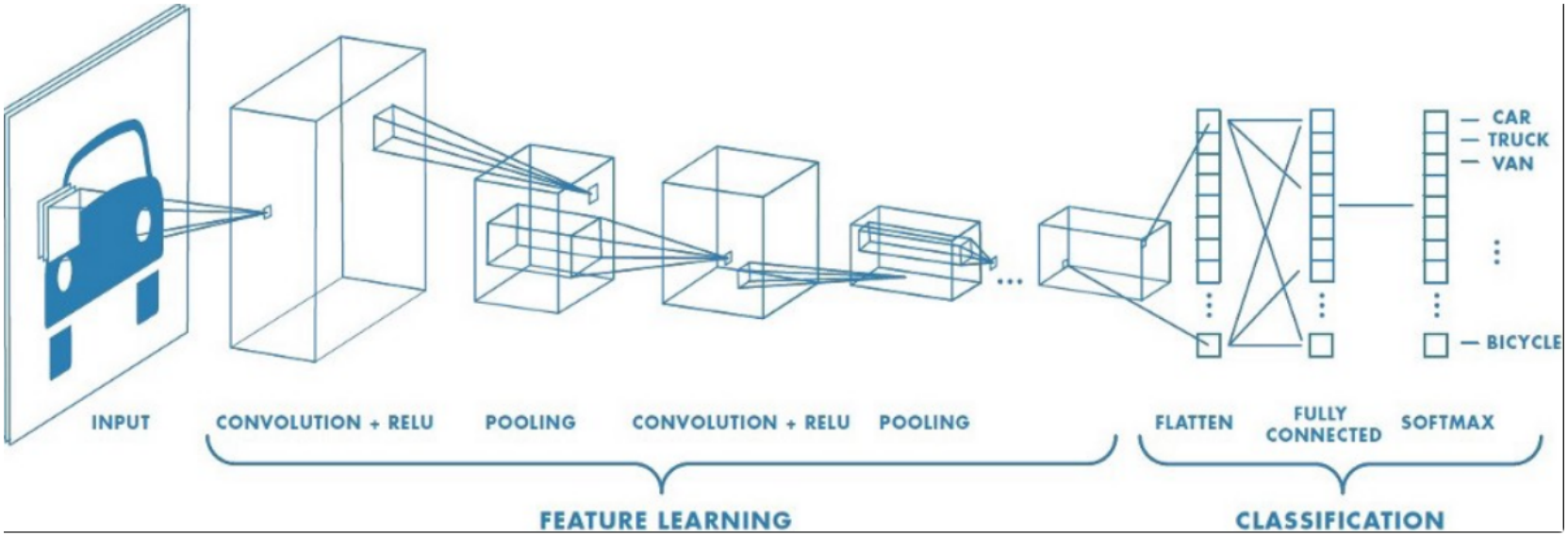
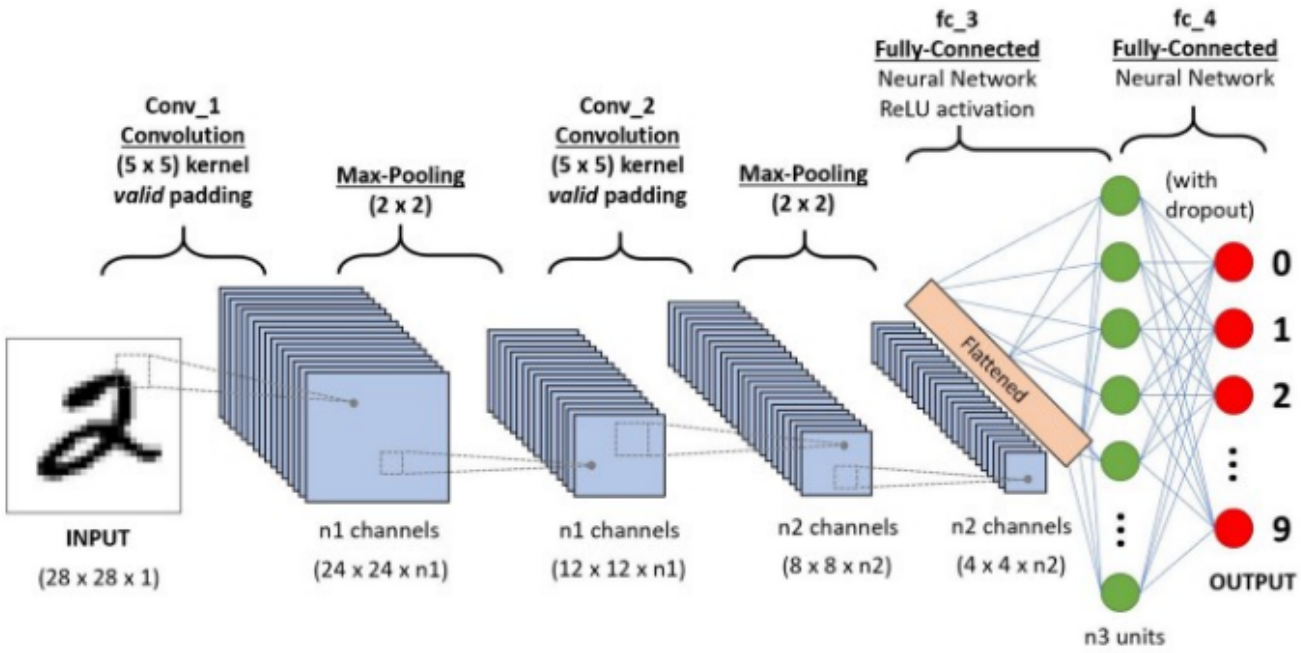


Figure 1: Architecture of CNN ([GBC, YB]

Convolution



Convolution



A CNN sequence to classify handwritten digits

CNN: Hidden Layers

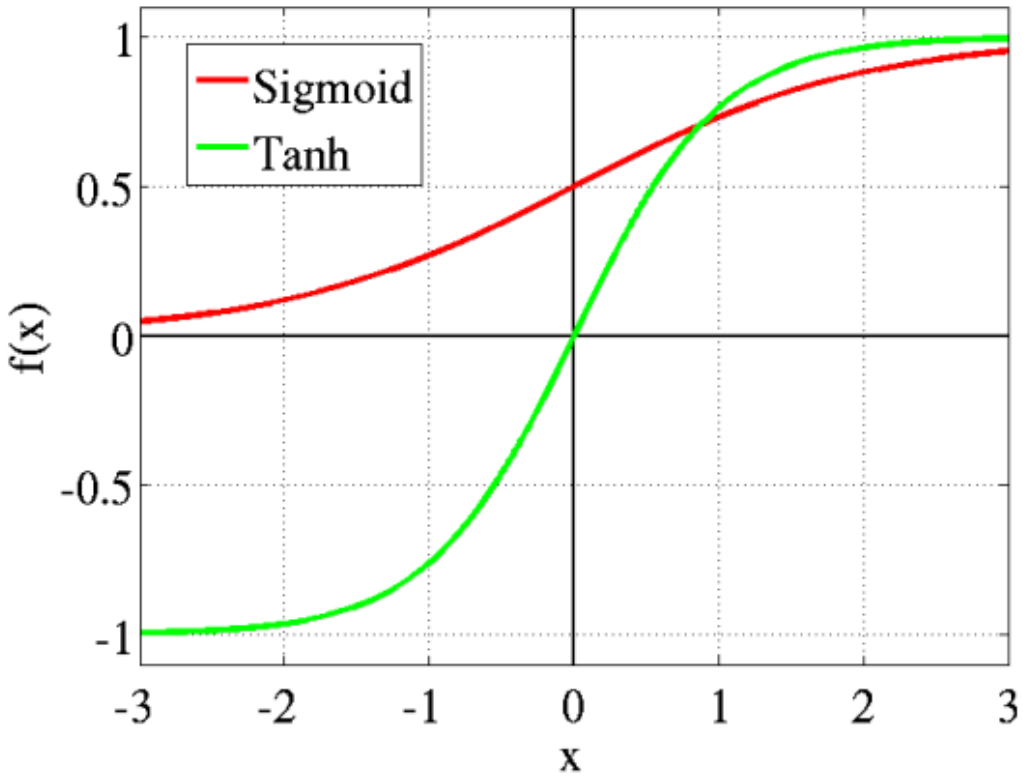
- **Layer function**: Basic transforming function such as convolutional or fully connected layer.
- **Pooling**: Used to change the spatial size of the feature map either increasing (up-sampling) or decreasing (most common) it. For example maxpooling, average pooling, and unpooling.
- **Normalization**: This subfunction normalizes the data to have zero mean and unit variance. This helps in coping up with problems such as vanishing gradient, internal covariate shift, etc. (more information). The two most common normalization techniques used are local response normalization and batch normalization.
- **Activation**: Applies non-linearity and bounds the output from getting too high or too low.

https://towardsdatascience.com/a-visualization-of-the-basic-elements-of-a-convolutional-neural-network-75fea30cd78d

- **Sigmoid**: $f(x) = \frac{1}{1+e^{-x}}$; $f'(x) = f(x)(1 - f(x))$.
- **tanh**: $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$; $f'(x) = 1 - f(x)^2$.
- **ReLU**: $f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$ $f'(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$
- **Leaky ReLU**: $f(x) = \begin{cases} 0.01x & x < 0 \\ x & x \geq 0 \end{cases}$ $f'(x) = \begin{cases} 0,01 & x < 0 \\ 1 & x \geq 0 \end{cases}$
- **Softmax**: $f(x_j) = \frac{e^{x_j}}{\sum_{k=1}^{d} e^{x_l}}$

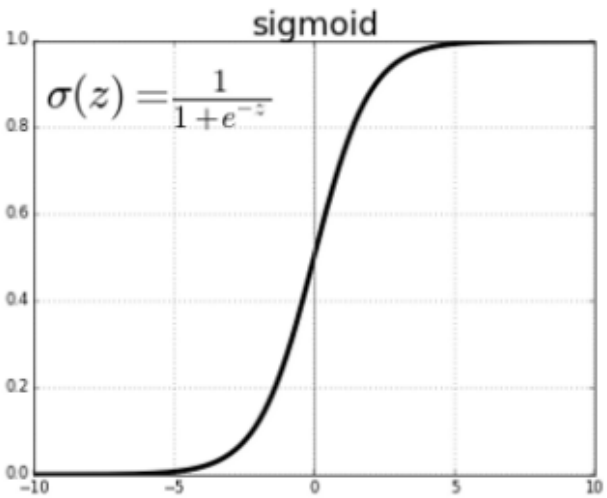
CNN

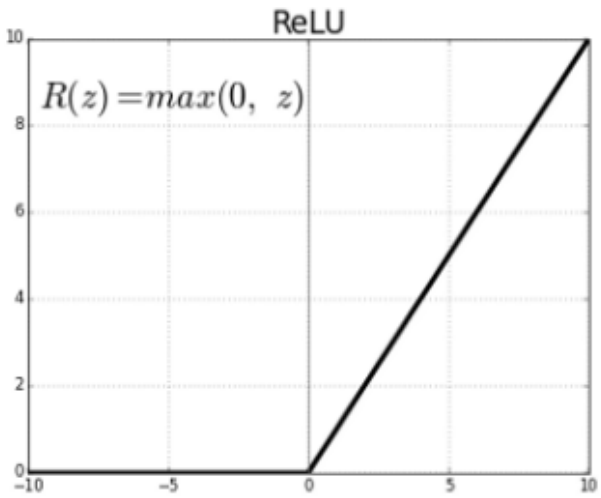
Activation Functions



CNN

Activation Functions





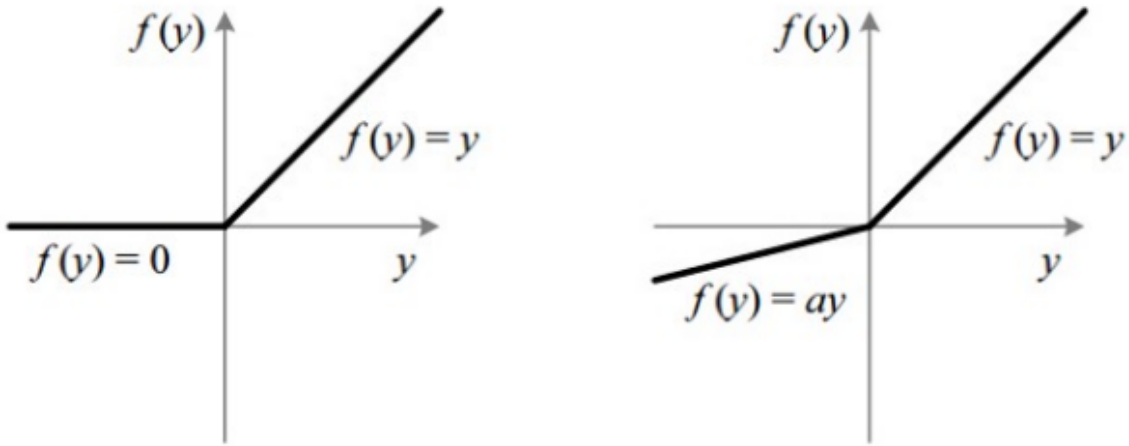
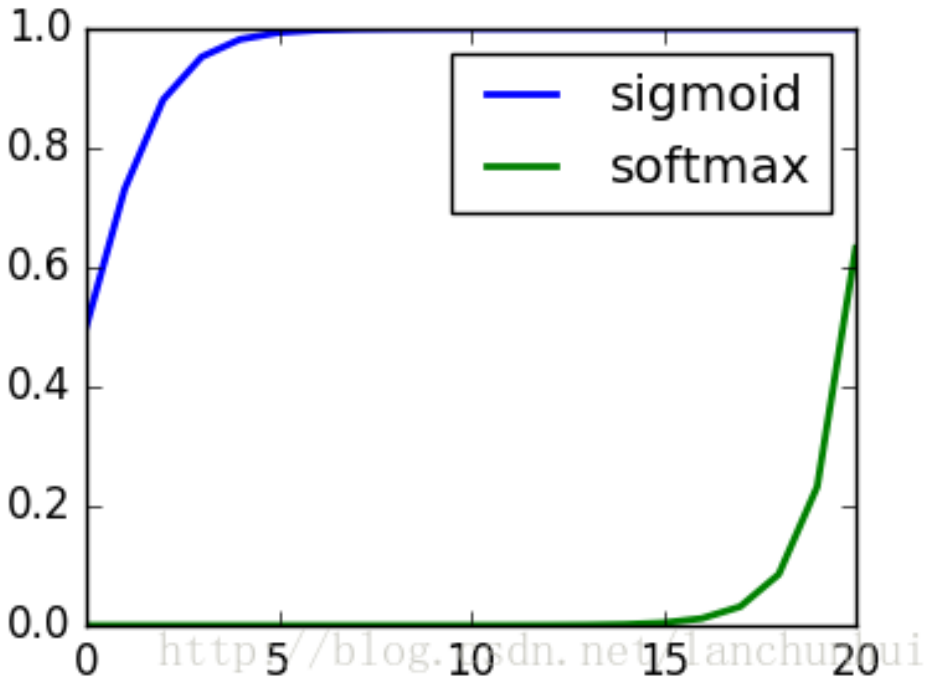


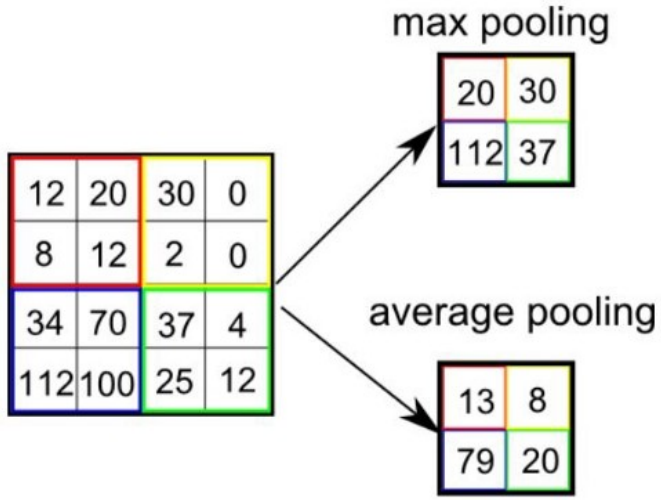
Fig : ReLU v/s Leaky ReLU

CNN

Activation Functions

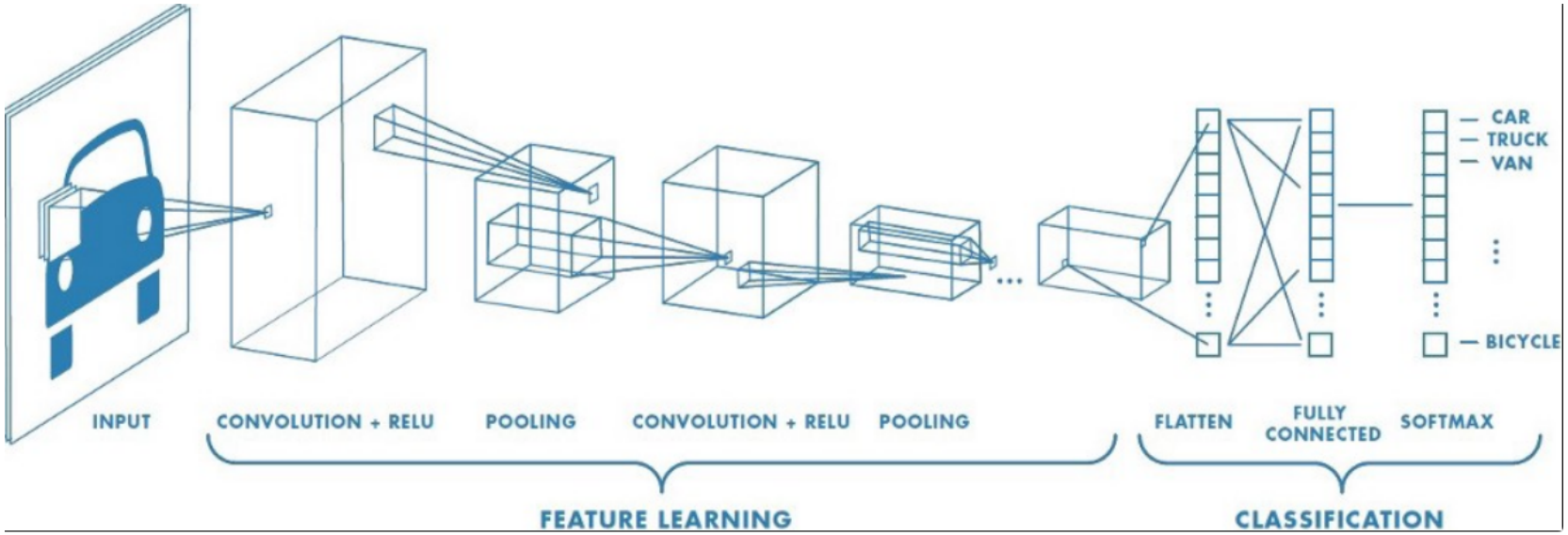


- **Max Pooling** returns the maximum value from the portion of the image covered by the Kernel.
- **Average Pooling** returns the average of all the values from the portion of the image covered by the Kernel.
- Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism.



https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Convolution



- After the pooling layers, pixels of pooling layers is stretched to single column vector.
- These vectorized and concatenated data points are fed into dense layers ,known as fully connected layers for the classification

Feed-Forward Run (or **Propagation**) can be explained as multiplying the input value by randomly initiated bias values of each connection of every neurons followed by summation of all the products of all the neurons. Then passing the net input value through non-linear activation functions.
In a discrete color space, image and kernel can be represented as a 3D tensor with the dimension of (H, W, C) and (k_1, k_2, C) where H, W, and C represent the H^{th}, and W^{th} pixel in C^{th} channel. First two indices indicate the spatial coordinates and last index is indicating the color channel.
Mathematical form of convolution operation over multi dimensional tensor can be written as:

$$(I \otimes K)_{ij} = \sum_{m=1}^{k_1} \sum_{n=1}^{k_2} \sum_{c=1}^{C} K_{m,n,c} I_{i+m,j+n,c}$$

[1] Chandra Bajaj CNN lecture Notes

If a kernel bank $K_{u,v}^{p,q}$ convolved with image $I_{m,n}$ with stride value of 1 and zero padding value of 0, then feature maps of the convolution layer $C_{m,n}^{p,q}$ can be computed by,

$$C_{m,n}^{p,q} = \sum_{m=1}^{k_1} \sum_{n=1}^{k_2} I_{m-u,n-v} \cdot K_{u,v}^{p,q} + b^{p,q}$$

where $b^{p,q}$ are bias term. These feature maps are passed through a non-linear activation function σ

$$C_{m,n}^{p,q} = \sigma \left(\sum_{m=1}^{k_1} \sum_{n=1}^{k_2} I_{m-u,n-v} \cdot K_{u,v}^{p,q} + b^{p,q} \right)$$

where σ is a ReLU activation function.

Max Pooling layer can be calculated as

$$P_{m,n}^{p,q} = \max(C_{m,n}^{p,q})$$

This Pooling layer $P^{p,q}$ is concatenated to form a long vector with the length p x q and is fed into fully connected dense layers for the classification, then the vectorized data points a_i^{l-1} in $l-1$ layer is given by

$$a_i^{l-1} = f(P^{p,q})$$

This long vector is fed to a fully connected dense layers from lth layer to $(L+1)$th layer. For example, if the fully connected dense layers is developed with L number of layers and n number of neurons, then l is the first layer.

3.1 Parameter updates

To minimize loss function, we need to update the learning parameters via gradient descent scheme. With the idea of back propagation, we can estimate the partial derivative so the following update scheme is computable:

$$W^l = W^l - \alpha \frac{\partial L(\hat{\mathbf{y}}^{L+1}, \mathbf{y})}{\partial W^l}$$

$$W^l = W^l - \alpha \frac{\partial L(\hat{\mathbf{y}}^{L+1}, \mathbf{y})}{\partial W^l}$$

$$K^{p,q} = K^{p,q} - \alpha \frac{\partial L(\hat{\mathbf{y}}^{L+1}, \mathbf{y})}{\partial K^{p,q}_{u,v}}$$

$$b^{p,q} = b^{p,q} - \alpha \frac{\partial L(\hat{\mathbf{y}}^{L+1}, \mathbf{y})}{\partial b^{p,q}}$$

where α is the learning rate.

Convolution