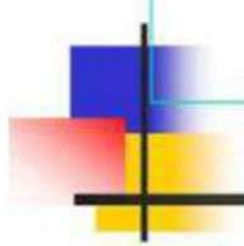
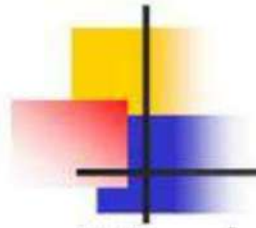


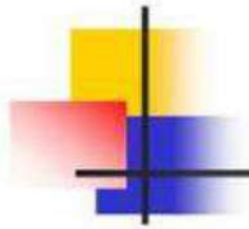
Goodfellow: Chap 5 Machine Learning Basics

Dr. Charles Tappert



Introduction

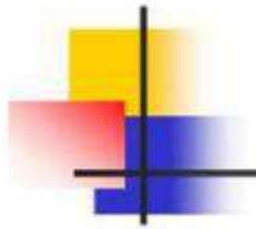
- Machine Learning
  - Form of applied statistics with extensive use of computers to estimate complicated functions
- Two central approaches
  - Frequentist estimators and Bayesian inference
- Two categories of machine learning
  - Supervised learning and unsupervised learning
- Most machine learning algorithms based on stochastic gradient descent optimization
- Focus on building machine learning algorithms



1 Learning Algorithms

- A machine learning algorithm learns from data
- A computer program learns from
  - experience *E*
  - with respect to some class of tasks *T*
  - and performance measure *P*
  - if it's performance *P* on tasks *T* improves with experience *E*

A machine learning algorithm is an algorithm that is able to learn from data. But what do we mean by learning? Mitchell (1997) provides the definition "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." One can imagine a very wide variety of experiences E, tasks T, and performance measures P, and we do not make any attempt in this book to provide a formal definition of what may be used for each of these entities. Instead, the following sections provide intuitive descriptions and examples of the different kinds of tasks, performance measures and experiences that can be used to construct machine learning algorithms.



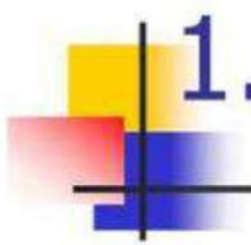
1.1 The Task, *T*

- Machine learning tasks are too difficult to solve with fixed programs designed by humans
- Machine learning tasks are usually described in terms of how the system should process an example where
  - An example is a collection of features measured from an object or event
- Many kinds of tasks
  - Classification, regression, transcription, anomaly detection, imputation of missing values, etc.

Task

- Machine learning tasks are usually described in terms of how the machine learning system should process an **example.**
- An example is a collection of **features** that have been quantitatively measured from some object or event that we want the machine learning system to process.
- Eg: program the robot to learn to walk, driverless car

- Many kinds of tasks can be solved with machine learning. Some of the most common machine learning tasks include the following:
- Classification
- Regression
- Machine Translation
- Anomaly detection

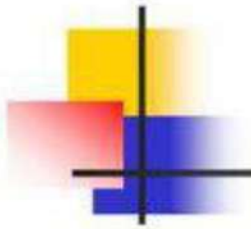


1.2 The Performance Measure, *P*

- Performance *P* is usually specific to the Task *T*
- For classification tasks, *P* is usually accuracy
  - The proportion of examples correctly classified
  - Here we are interested in the accuracy on data not seen before – a test set

Performance

- In order to evaluate the abilities of a machine learning algorithm, we must design a quantitative measure of its performance. Usually this performance measure *P* is specific to the task *T* being carried out by the system.
- Accuracy- proportion of examples for which the model produces the correct output
- Error rate- the proportion of examples for which the model produces an incorrect output
- Task classification- measures accuracy



1.3 The Experience, *E*

- Most learning algorithms in this book are allowed to experience an entire dataset
- Unsupervised learning algorithms
  - Learn useful structural properties of the dataset
- Supervised learning algorithms
  - Experience a dataset containing features with each example associated with a label or target
    - For example, target provided by a teacher
- Blur between supervised and unsupervised

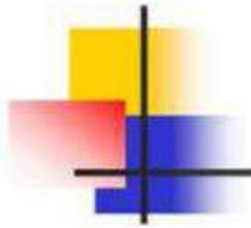
- Supervised algorithm- experience a dataset containing features, but each example is also associated with a **label** or **target**
- **Unsupervised algorithm-** experience a dataset containing many features, then learn useful properties of the structure of this dataset.
- Reinforcement learning- interact with an environment, so there is a feedback loop between the learning system and its experiences

1.4 Example: Linear Regression

- Linear regression takes a vector x as input and predicts the value of a scalar y as it's output
  - Task: predict scalar y from vector x $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$
  - Experience: set of vectors X and vector of targets y
  - Performance: mean squared error => normal eqs
  - Solution of the normal equations is

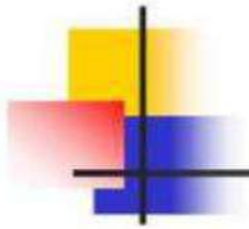
$$\boldsymbol{w} = \left(\boldsymbol{X}^{(\text{train})\top} \boldsymbol{X}^{(\text{train})} \right)^{-1} \boldsymbol{X}^{(\text{train})\top} \boldsymbol{y}^{(\text{train})}$$

2 Capacity, Overfitting and Underfitting



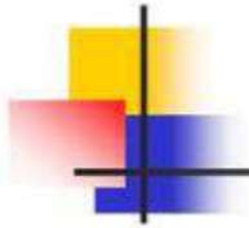
- The central challenge in machine learning is to perform well on new, previously unseen inputs, and not just on the training inputs
- We want the generalization error (test error) to be low as well as the training error
- This is what separates machine learning from optimization

2 Capacity, Overfitting and Underfitting



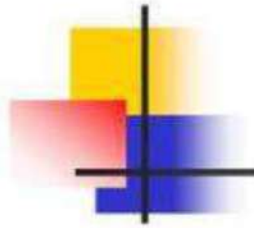
- How can we affect performance on the test set when we get to observe only the training set?
- Statistical learning theory provides answers
- We assume train and test sets are generated independent and identically distributed – i.i.d.
- Thus, expected train and test errors are equal

2 Capacity, Overfitting and Underfitting



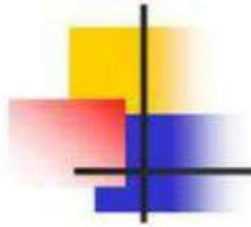
- Because the model is developed from the training set the expected test error is usually greater than the expected training error
- Factors determining machine performance are
  - Make the training error small
  - Make the gap between train and test error small
- These two factors correspond to the two challenges in machine learning
  - Underfitting and overfitting

2 Capacity, Overfitting and Underfitting



- Underfitting and overfitting can be controlled somewhat by altering the model's capacity
  - Capacity = model's ability to fit variety of functions
    - Low capacity models struggle to fit the training data
    - High capacity models can overfit the training data
- One way to control the capacity of a model is by choosing its hypothesis space
  - For example, simple linear vs quadratic regression

2 Capacity, Overfitting and Underfitting



- While simpler functions are more likely to generalize (have a small gap between training and test error) we still want low training error
- Training error usually decreases until it asymptotes to the minimum possible error
- Generalization error usually has a U-shaped curve as a function of model capacity

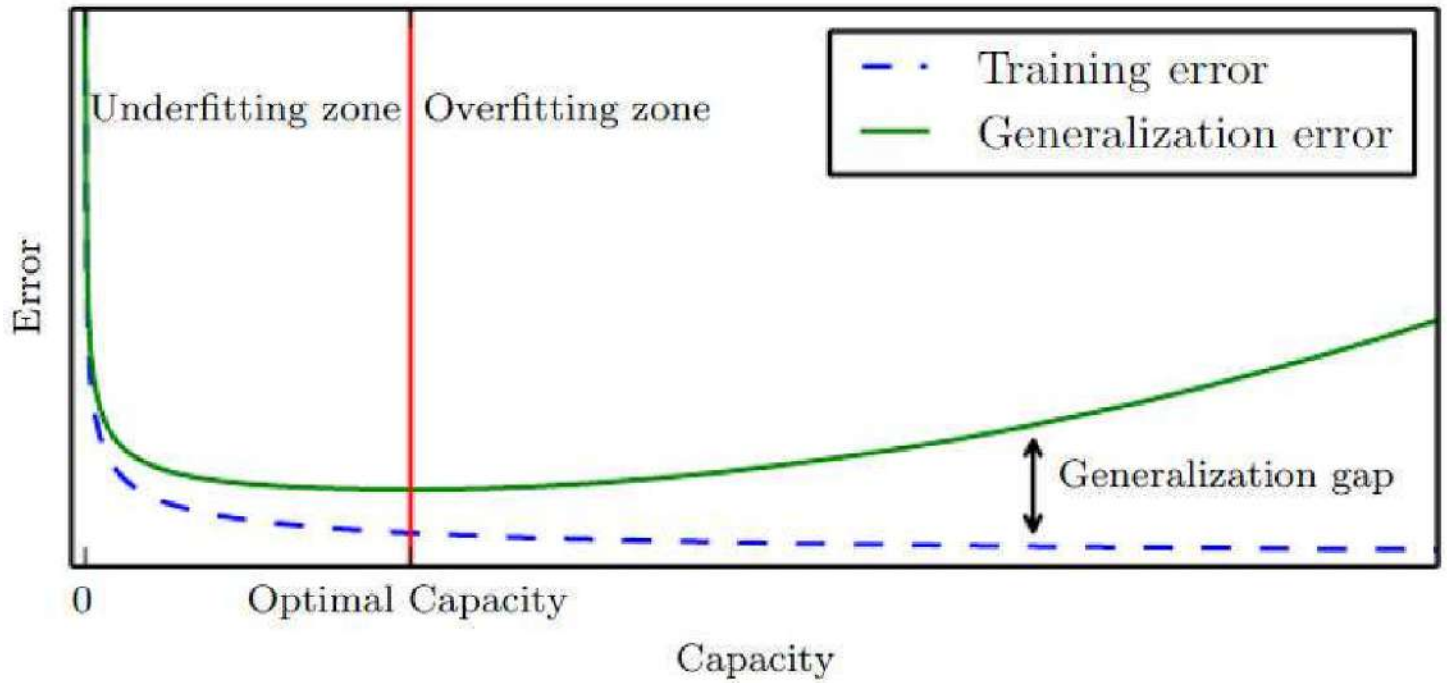
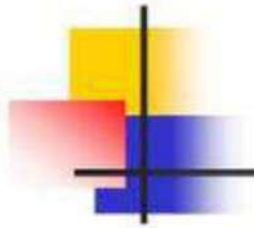


Figure 5.3: Typical relationship between capacity and error. Training and test error behave differently. At the left end of the graph, training error and generalization error are both high. This is the **underfitting regime**. As we increase capacity, training error decreases, but the gap between training and generalization error increases. Eventually, the size of this gap outweighs the decrease in training error, and we enter the **overfitting regime**, where capacity is too large, above the **optimal capacity**.

2 Capacity, Overfitting and Underfitting



- Non-parametric models can reach arbitrarily high capacities
- For example, the complexity of the nearest neighbor algorithm is a function of the training set size
- Training and generalization error vary as the size of the training set varies
  - Test error decreases with increased training set size

Parameters in ML

- Parameter is a value that a model learns from the training data.
- It is used to control the behavior of the algorithm and to make predictions on new data.
- Eg:, in linear regression, the parameters are the coefficients that represent the slope and intercept of the regression line.

Functions

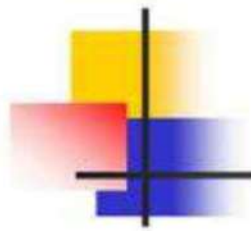
- A function, on the other hand, is a mathematical relationship between the input data and the output data.
- In machine learning, functions can be used to make predictions based on the input data.
- For example, in a decision tree, a function is used to determine the path to follow through the tree to reach a predicted outcome.
- In summary, parameters are the values learned by a machine learning algorithm to make predictions, while functions are the mathematical relationships used by the algorithm to map input data to output data.

Function Estimation As we mentioned above, sometimes we are interested in performing function estimation (or function approximation). Here we are trying to predict a variable $\boldsymbol{y}$ given an input vector $\boldsymbol{x}$. We assume that there is a function $f(\boldsymbol{x})$ that describes the approximate relationship between $\boldsymbol{y}$ and $\boldsymbol{x}$. For example, we may assume that $\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ stands for the part of $\boldsymbol{y}$ that is not predictable from $\boldsymbol{x}$. In function estimation, we are interested in approximating f with a model or estimate $\hat{f}$. Function estimation is really just the same as estimating a parameter $\boldsymbol{\theta}$; the function estimator $\hat{f}$ is simply a point estimator in function space. The linear regression example (discussed above in section 5.1.4) and the polynomial regression example (discussed in section 5.2) are both examples of scenarios that may be interpreted either as estimating a parameter $\boldsymbol{w}$ or estimating a function $\hat{f}$ mapping from $\boldsymbol{x}$ to y.

- Parametric machine learning models make assumptions about the underlying distribution of the data and have a fixed number of parameters, which are estimated from the training data. Examples of parametric models include linear regression, logistic regression, and Naive Bayes.
- Non-parametric machine learning models, on the other hand, do not make assumptions about the underlying distribution of the data and can have a variable number of parameters, which are not fixed prior to training. Eg: nearest neighbor

2.1 No Free Lunch Theorem

- Averaged over all possible data generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points
- In other words, no machine learning algorithm is universally any better than any other
- However, by making good assumptions about the encountered probability distributions, we can design high-performing learning algorithms
  - This is the goal of machine learning research



3 Hyperparameters and Validation Sets

- Most machine learning algorithms have several settings, called hyperparameters, to control the behavior of the algorithm

3 Hyperparameters and Validation Sets

- To avoid overfitting during training, the training set is often split into two sets
  - Called the training set and the validation set
  - Typically, 80% of the training data is used for training and 20% for validation
- Cross-Validation
  - For small datasets, *k*-fold cross validation partitions the data into *k* non-overlapping subsets
    - *k* trials are run, train on *(k-1)/k* of the data, test on *1/k*
    - Test error is estimated by averaging error on each run

Hyperparameters

- Hyperparameters are the settings or configurations of a machine learning algorithm that cannot be learned from the data during training.
- Instead, they need to be set by the user prior to training.
- The choice of hyperparameters can significantly affect the performance of the model, and tuning them is an important part of the machine learning workflow.
- Eg: Neural Networks: number of layers, number of neurons per layer, activation function, learning rate, regularization strength
- Decision Trees: maximum depth of the tree, minimum number of samples required to split a node, minimum number of samples required to be at a leaf node

Cross validation

- *k*-fold cross-validation
- partition of the dataset by splitting it into *k* non-overlapping subsets.
- The test error may then be estimated by taking the average test error across *k* trials.
- On trial *i*, the *i* -th subset of the data is used as the test set and the rest of the data is used as the training set

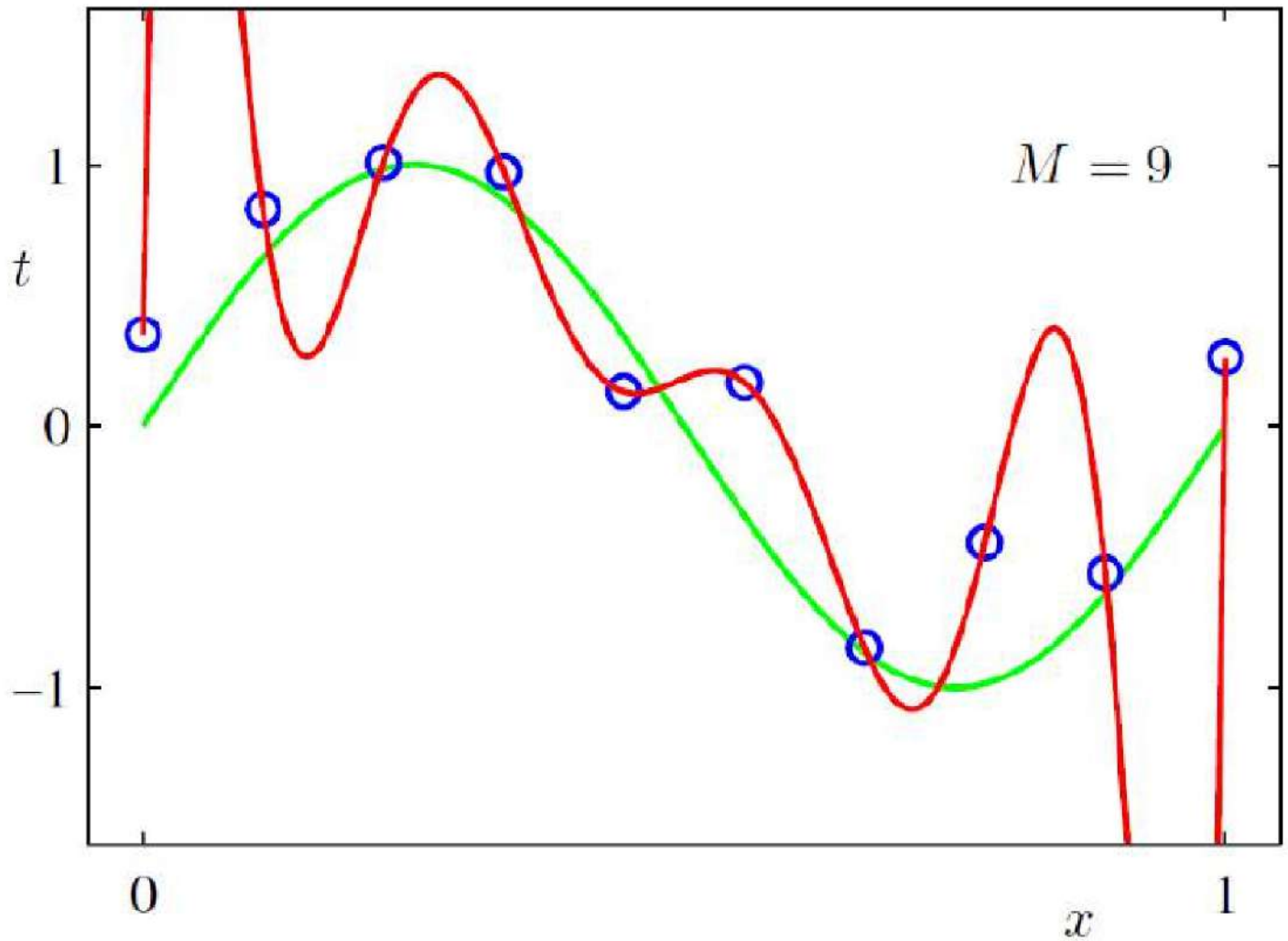
Regularization

- When building models, data scientists and statisticians often talk about penalty, regularization and shrinkage. What do these terms mean and why are they important?

- **The goal of machine learning is to model the pattern and ignore the noise. Anytime an algorithm is trying to fit the noise in addition to the pattern, it is overfitting.**

Overfitting example

$$t = \sin(2\pi x) + \epsilon$$



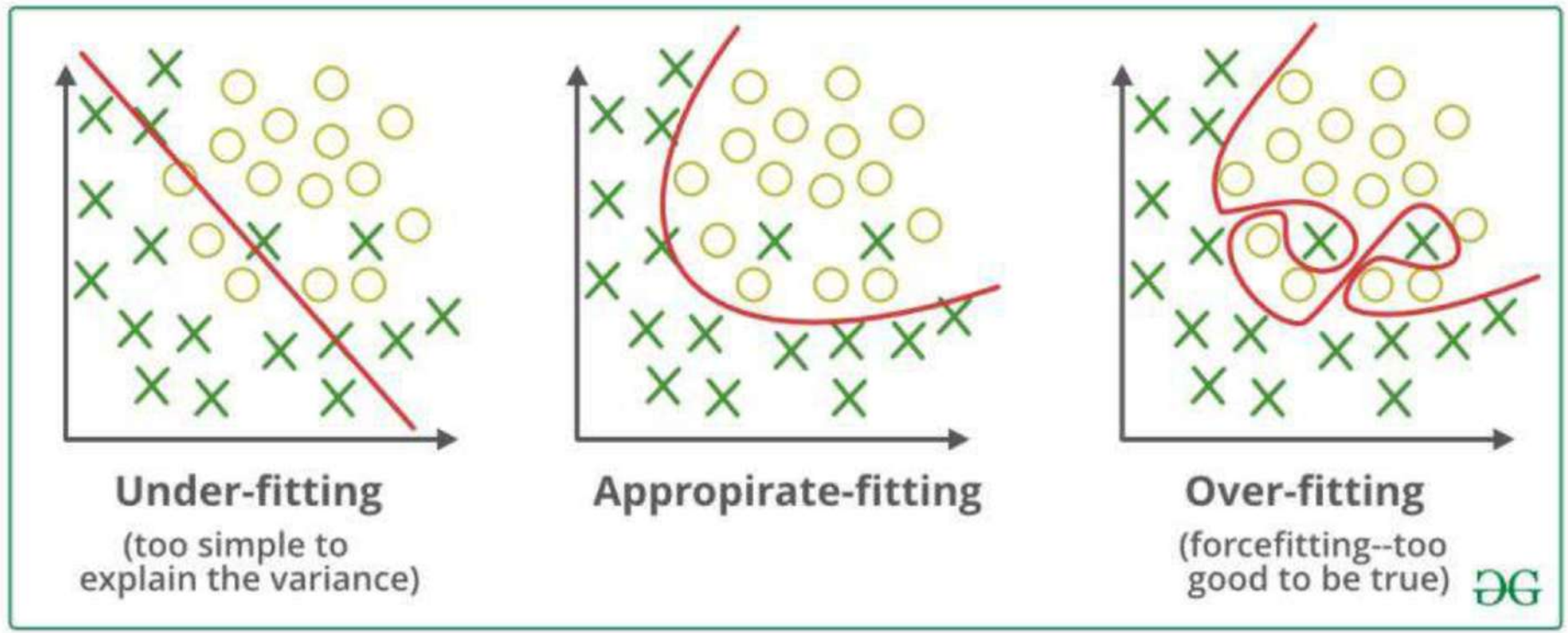
- **Regularization "refers to a process of introducing additional information in order to solve an ill-posed problem or to prevent over fitting.**

- This information usually comes in the form of a penalty for complexity, such as restrictions for smoothness or bounds on the vector space norm."

  - In general: any method to prevent overfitting or help the optimization

  - Specifically: additional terms in the training optimization objective to prevent overfitting or help the optimization

Overfitting is a phenomenon that occurs when a Machine Learning model is constraint to training set and not able to perform well on unseen data.



Regularization

- The main idea behind the regularization technique is **to penalize complex models, i.e.- to define a penalty function to quantify complexity of the model.** (the more complex models will have a greater penalty associated with them). Since most of the training algorithms are considered an optimization problem where the loss is minimized we add the penalty term and minimize the whole expression together.

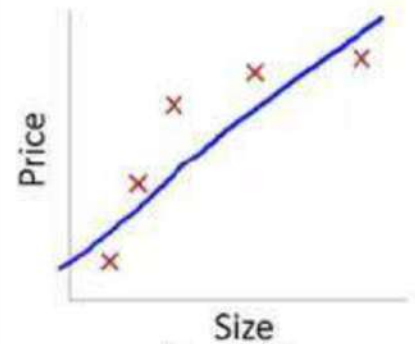
Bias-Variance

Bias-Variance Tradeoff

Bias is the amount of error introduced by approximating real-world phenomena with a simplified model.

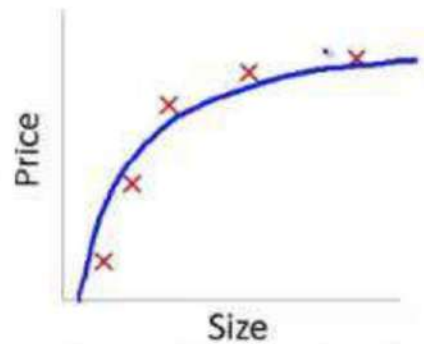
Variance is how much your model's test error changes based on variation in the training data. It reflects the model's sensitivity to the idiosyncrasies of the data set it was trained on.

As a model increases in complexity and it becomes more wiggly (**flexible**), its bias decreases (it does a good job of explaining the training data), but variance increases (it doesn't generalize as well). **Ultimately, in order to have a good model, you need one with low bias and low variance.**



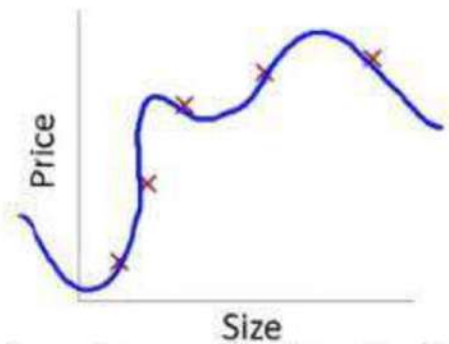
$\theta_0 + \theta_1 x$

High bias
(underfit)



$\theta_0 + \theta_1 x + \theta_2 x^2$

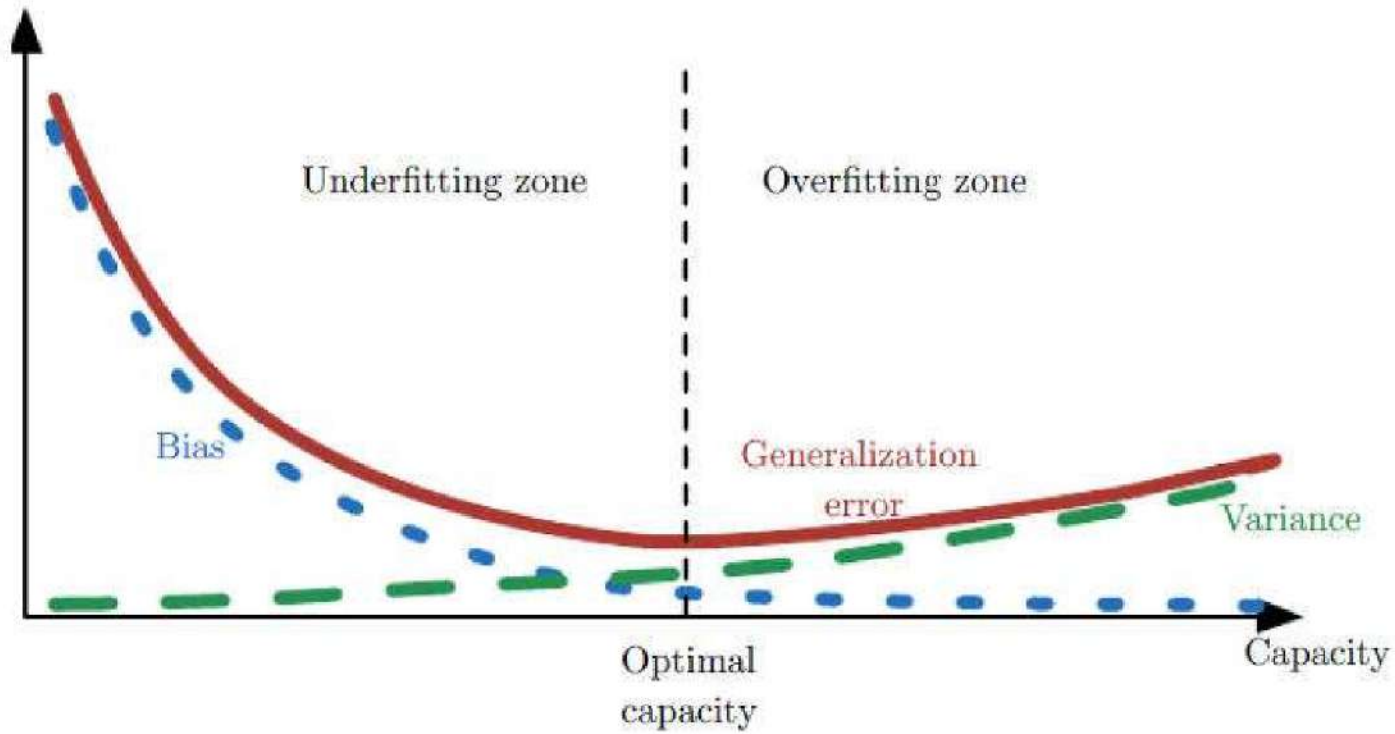
"Just right"



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

High variance
(overfit)

- Bias-variance tradeoff is a fundamental concept in machine learning that refers to the tradeoff between the model's ability to fit the training data (bias) and its ability to generalize to new, unseen data (variance)
- A model with high bias tends to underfit the data, while a model with high variance tends to overfit the data.
- The goal is to find the optimal balance between bias and variance to achieve the best possible predictive performance on new data.
- Techniques such as regularization, cross-validation, can help to mitigate the bias-variance tradeoff.


Underfitting zone
Overfitting zone
Bias
Generalization
error
Variance
Optimal
capacity
Capacity

6.5.2 Chain Rule of Calculus

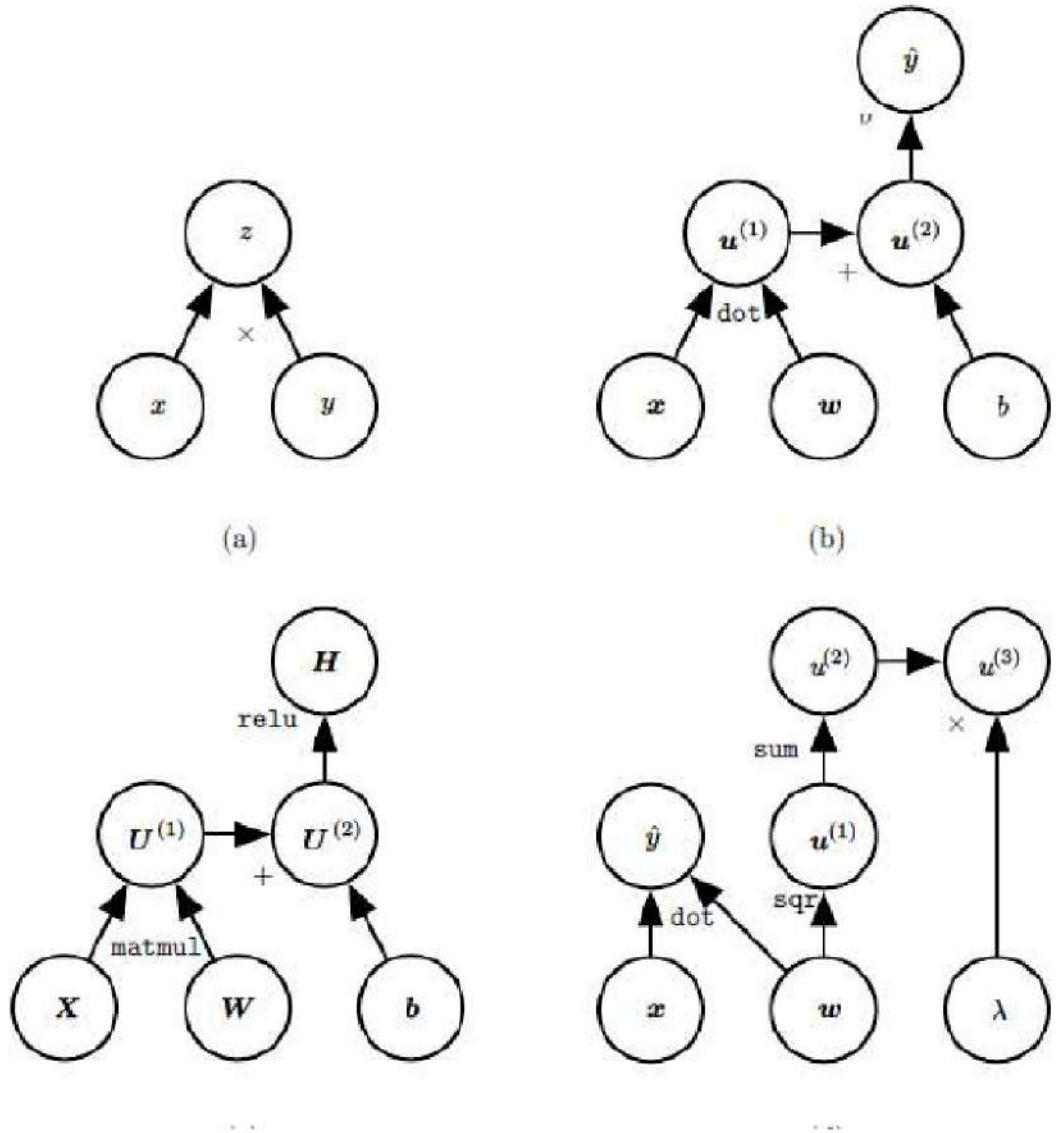
The chain rule of calculus (not to be confused with the chain rule of probability) is used to compute the derivatives of functions formed by composing other functions whose derivatives are known. Back-propagation is an algorithm that computes the chain rule, with a specific order of operations that is highly efficient.

Let x be a real number, and let f and g both be functions mapping from a real number to a real number. Suppose that $y = g(x)$ and $z = f(g(x)) = f(y)$. Then the chain rule states that

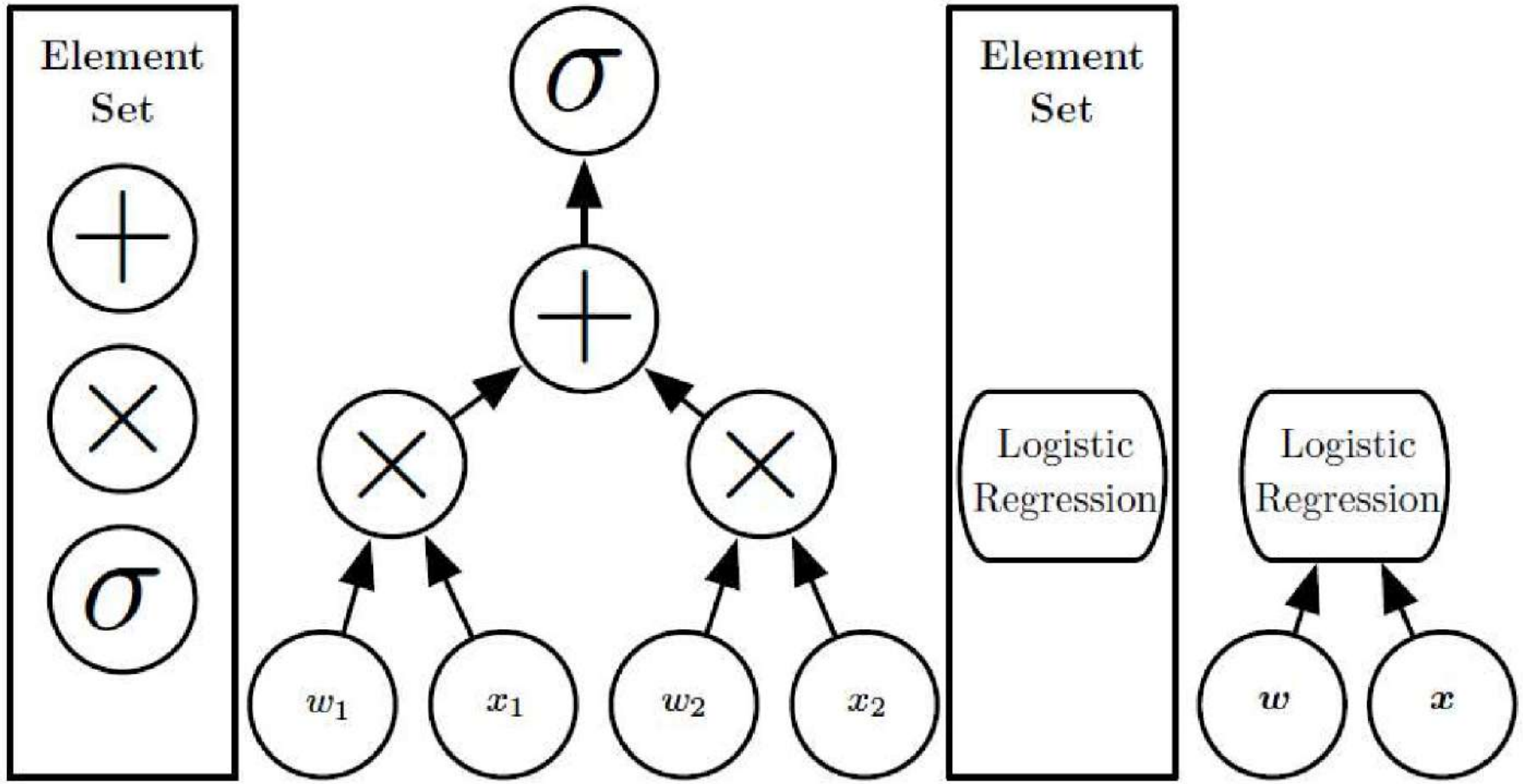
$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}. \tag{6.44}$$

We can generalize this beyond the scalar case. Suppose that $\boldsymbol{x} \in \mathbb{R}^m$, $\boldsymbol{y} \in \mathbb{R}^n$,

Computational graphs



Computational Graphs



- Directed graphs, also known as directed acyclic graphs (DAGs), are a type of graph in which edges have a direction associated with them. In other words, they are graphs in which the connections between nodes are one-way.
- A directed computational graph, also known as a neural network graph, is a type of directed graph that represents the computation flow of a neural network. Each node in the graph represents a neuron or a computational step, and the edges represent the connections between them.

Introduction to Convolution

Contents

- Image
- Filter/mask/kernel/template/window
- Convolution
- Padding
- Stride
- Types of filters

What's an image?

"A lady is crossing the road "

6 words, 3 stop words, 13 char sequence

"An image is worth 1000 words"

"An image is worth 1,00,000 words"

600X300 image,3 channels, 5,40,000 pixel intensity values

Image

255	144	…	120
57	100	…	32
…	…	…	…
251	41	…	23
57	209	…	118
27	59	…	19
…	…	…	…
4	203	…	69
35	170	…	199
17	11	…	97
…	…	…	…
145	165	…	210

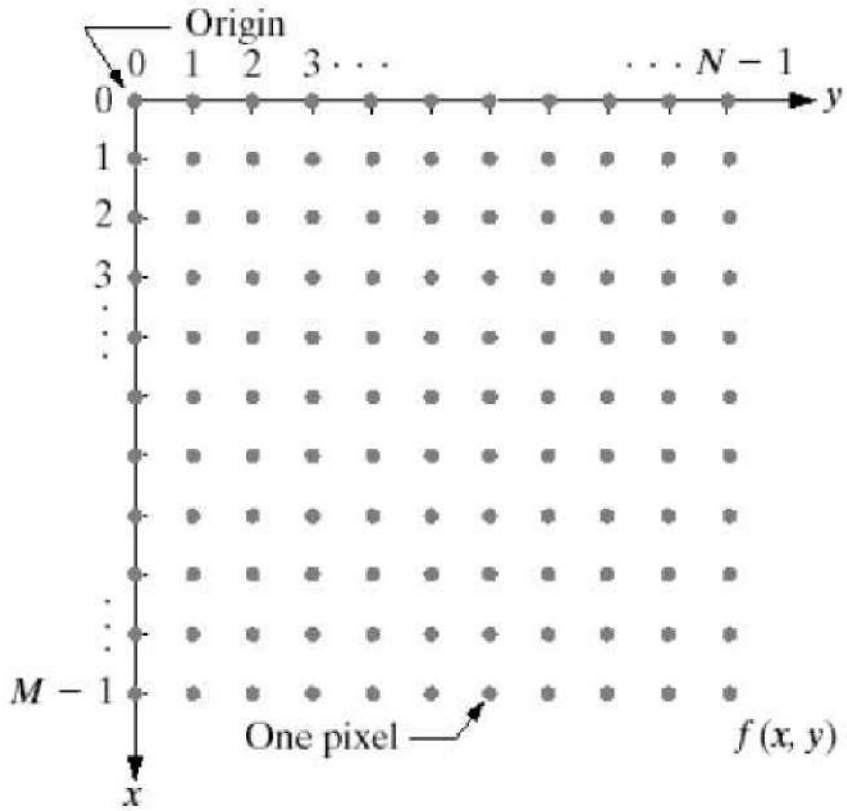
What is an image

- An image is nothing more than a two dimensional signal.
- It is defined by the mathematical function f(x,y) where x and y are the two co-ordinates horizontally and vertically.
- The value of f(x,y) at any point gives the pixel value at that point of an image.

Fundamentals of Digital Images

- **An image**: **a multidimensional function of spatial coordinates**.
- **Spatial coordinate**: (x,y) for 2D case such as photograph,
  (x,y,z) for 3D case such as CT scan images
  (x,y,t) for movies
- The function f may represent intensity (for monochrome images) or color (for color images) or other associated values.

Conventional Coordinate for Image Representation



- This image is a two dimensional array of numbers ranging between 0 and 255.

128	30	123
232	123	321
123	77	89
80	255	255

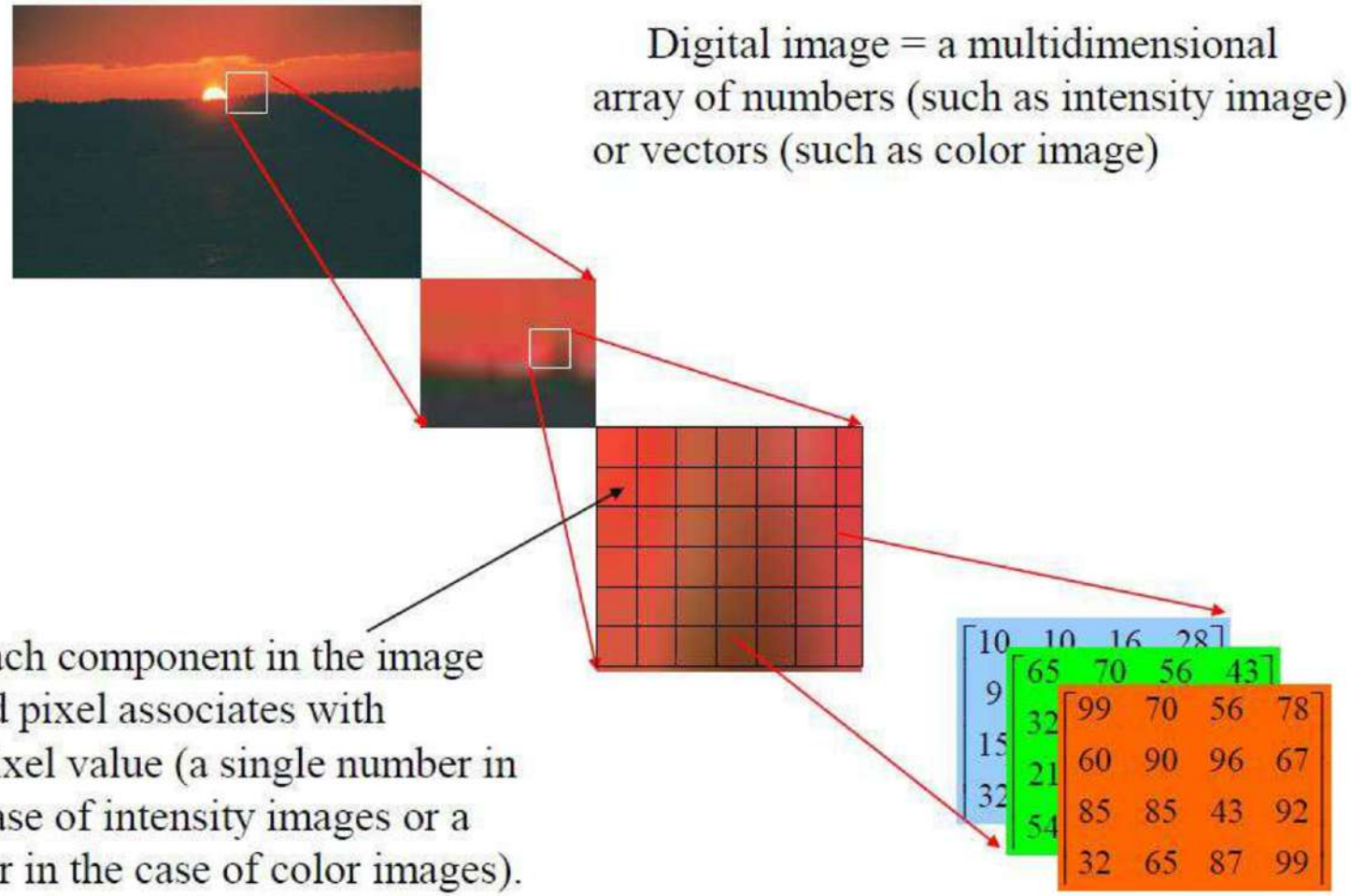
- Each number represents the value of the function f(x,y) at any point. In this case the value 128 , 230 ,123 each represents an individual pixel value.
- The dimensions of the picture is actually the dimensions of this two dimensional array.

What We See

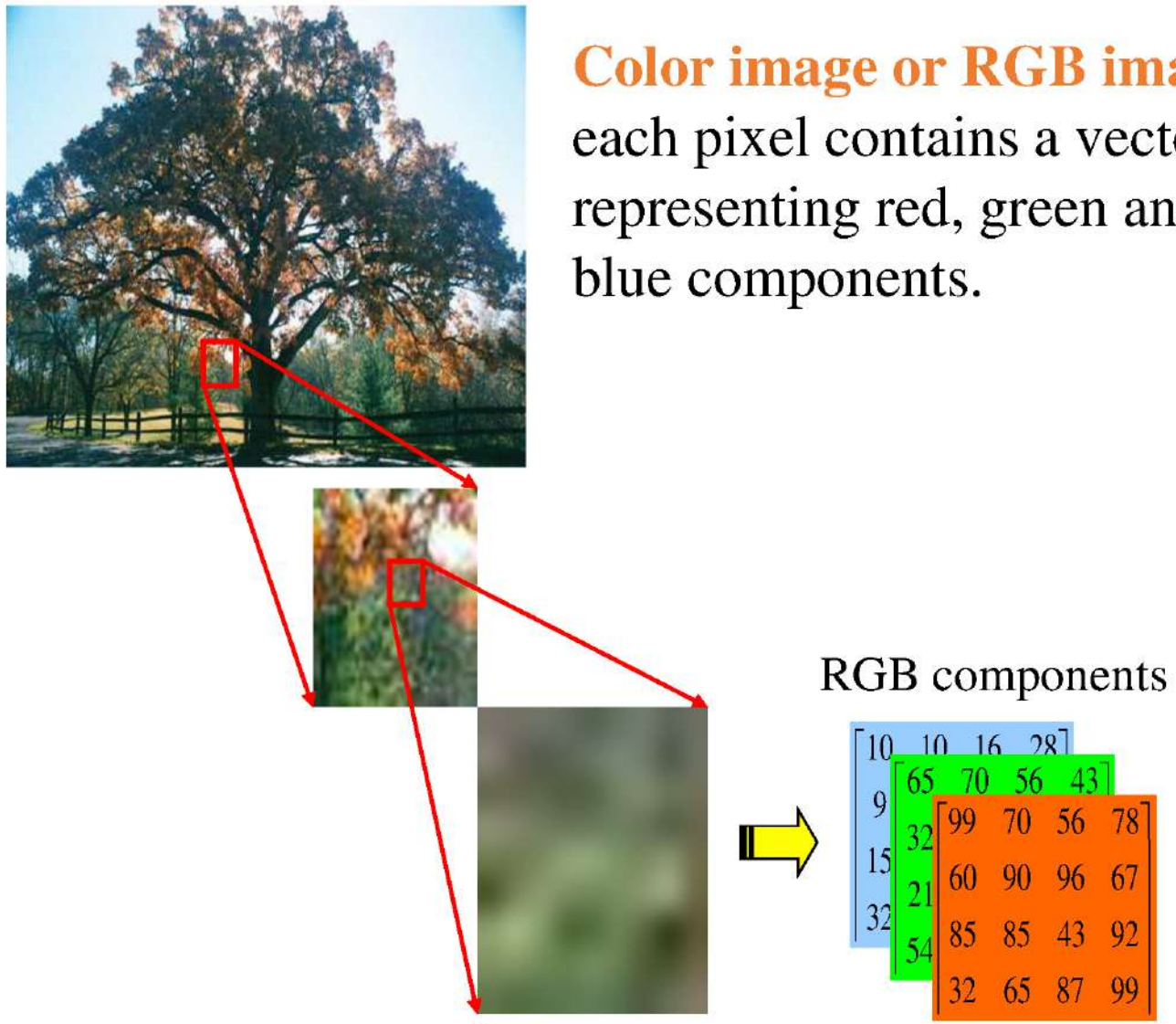
```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
```

What Computers See

Digital Image



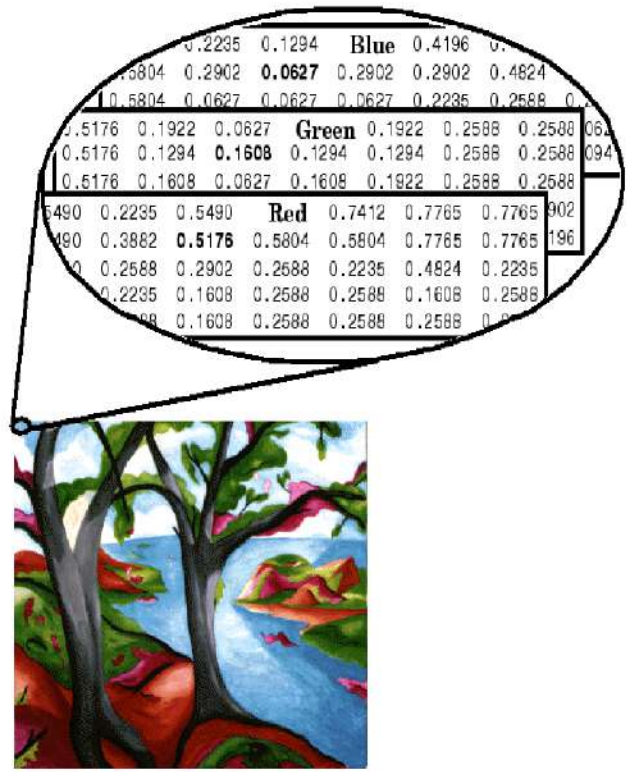
Digital Image Types : RGB Image

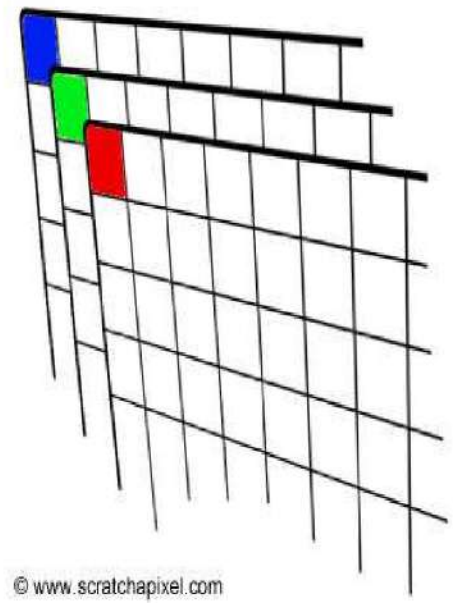


Color image or RGB image: each pixel contains a vector representing red, green and blue components.

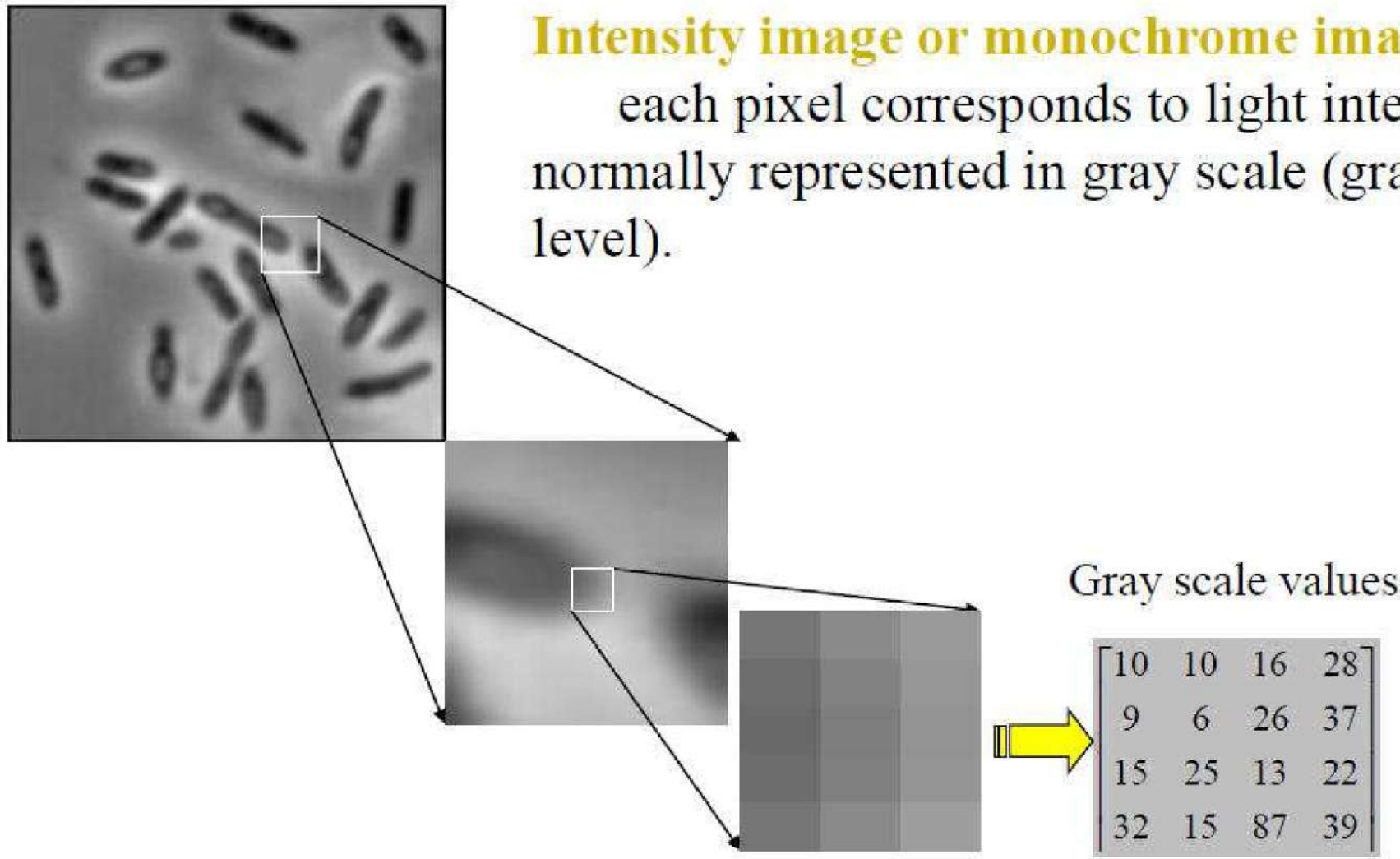
RGB image

- Image – 32 X 32 X 3





Digital Image Types : Intensity Image



Intensity image or monochrome image

each pixel corresponds to light intensity normally represented in gray scale (gray level).

Gray scale values

$$\begin{bmatrix} 10 & 10 & 16 & 28 \\ 9 & 6 & 26 & 37 \\ 15 & 25 & 13 & 22 \\ 32 & 15 & 87 & 39 \end{bmatrix}$$

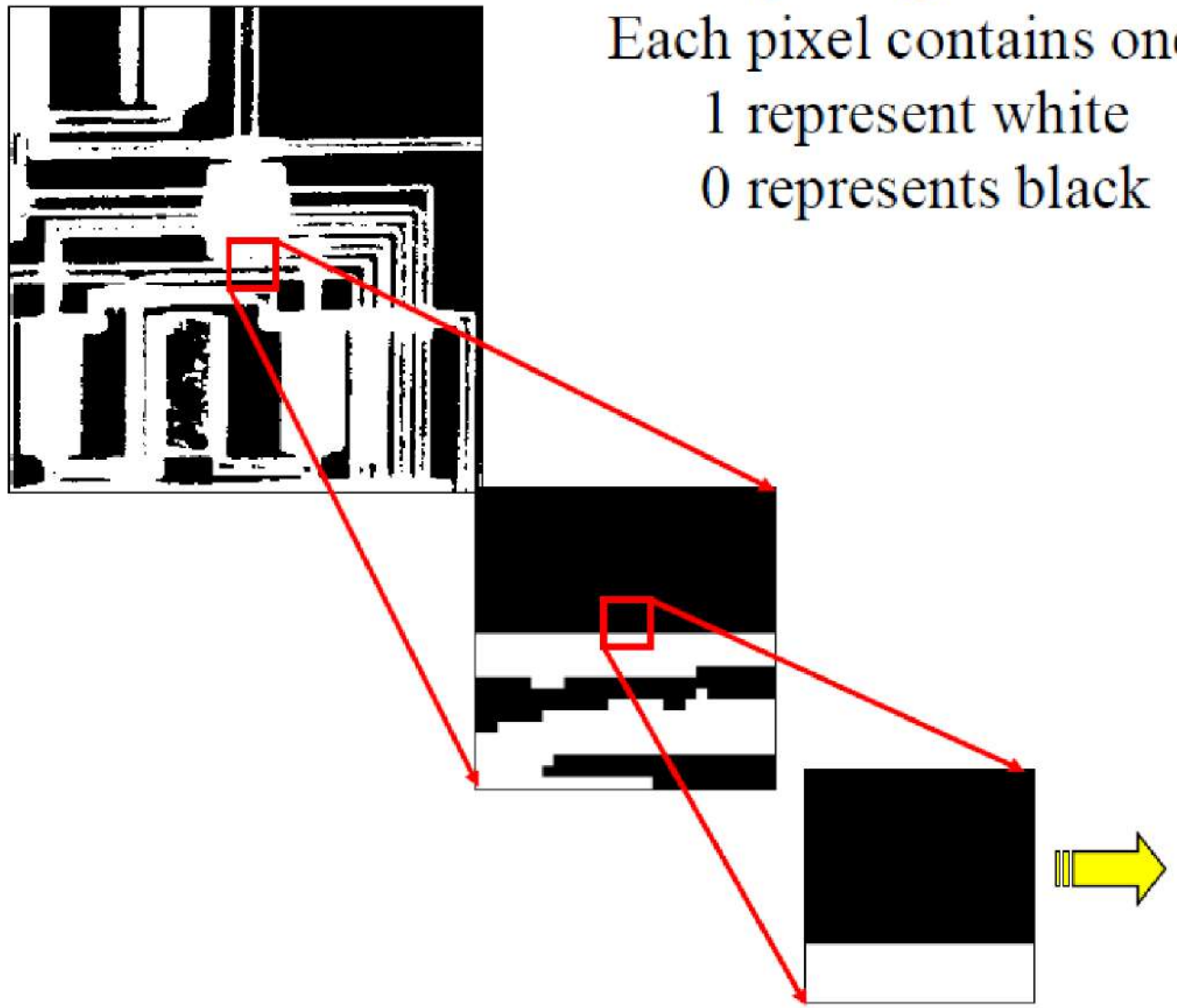
Image Types : Binary Image

Binary image or black and white image

Each pixel contains one bit :

1 represent white

0 represents black



Binary data

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

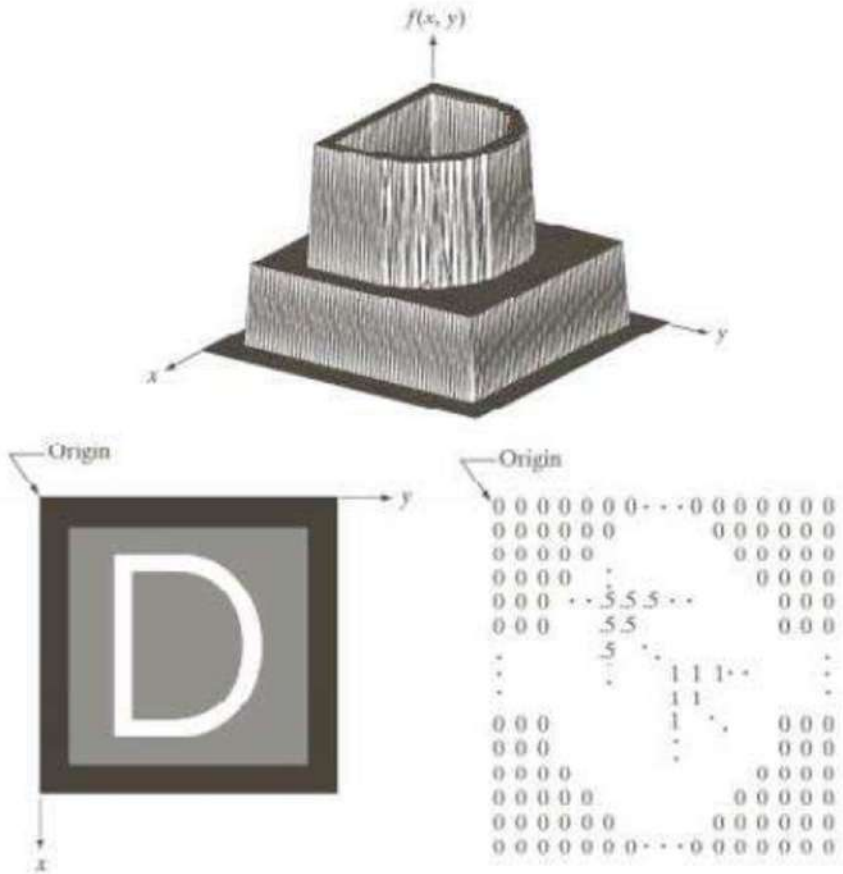
A simple image formation model

- Images denoted by two-dimensional functions $f(x,y)$
- Value of amplitude of f *at spatial coordinate* (x,y) *is a positive scalar quantity and finite.*
- Image generated by physical process, its intensity values are proportional to the energy radiated by a physical source =>
- $0 < f(x,y) < \infty$
- $f(x,y)$ *may be characterized by 2 components:*
  - **The amount of source illumination *incident on the scene:*** *illumination* $i(x,y)$
  - **The amount of illumination *reflected by the objects of the scene:*** *reflectance* $r(x,y)$
- $f(x,y) = i(x,y)\, r(x,y)$, *where* $0 < i(x,y) < \infty$ *and* $0 < r(x,y) < 1$

Image sampling and quantization

- To create a digital image, we need to convert the continuous sensed data **into digital form**
- This involves 2 processes: **Sampling and Quantization**
- An image is continuous in its **coordinates** and also in its **amplitude**(intensity values)
- Digitizing coordinate values- **sampling**
- Digitizing amplitude values- **quantization**
- for sensing strip- sampling **in one direction**
- For a sensing array- sensing in **both the directions**
- The real plane spanned by coordinates of an image is called **spatial domain**.
- X and Y being referred to as **spatial variables** or **spatial coordinates**

Representing digital image



Representing digital image

- Image is a 2 D array f(x,y)
- M rows and N Columns
- *(x,y) = discrete coordinates, x = 0, 1, 2,…, M-1 and y = 0, 1, 2…, N-1*
- Value of the image at any spatial coordinate x,y is f(x,y)
- L, **intensity levels**= 2^k
- K is the number of bits to represent a pixel
- Range of intensity values is **0 to L-1**
- **Dynamic range of an imaging system** – Ratio of the maximum measurable intensity to the minimum detectable intensity level in the system
- Upper limit is determined by **saturation**
- Lower limit is determined by **noise**
- **Contrast-** The difference in intensity between the highest and lowest intensity levels in an image
- Image which has high dynamic range has **high contrast**
- Image which has low dynamic range has dull, washed out gray look

- The number of bits required to store a digitized image is
  - $b = M \times N \times k$
- when $M=N$, it becomes $b = N^2 k$
- Image with 2^k intensity levels => *"k-bit image" (ex: 256 → 8-bit image)*

TABLE 2.1
Number of storage bits for various values of N and k.

N/k	1 ($L = 2$)	2 ($L = 4$)	3 ($L = 8$)	4 ($L = 16$)	5 ($L = 32$)	6 ($L = 64$)	7 ($L = 128$)	8 ($L = 256$)
32	1,024	2,048	3,072	4,096	5,120	6,144	7,168	8,192
64	4,096	8,192	12,288	16,384	20,480	24,576	28,672	32,768
128	16,384	32,768	49,152	65,536	81,920	98,304	114,688	131,072
256	65,536	131,072	196,608	262,144	327,680	393,216	458,752	524,288
512	262,144	524,288	786,432	1,048,576	1,310,720	1,572,864	1,835,008	2,097,152
1024	1,048,576	2,097,152	3,145,728	4,194,304	5,242,880	6,291,456	7,340,032	8,388,608
2048	4,194,304	8,388,608	12,582,912	16,777,216	20,971,520	25,165,824	29,369,128	33,554,432
4096	16,777,216	33,554,432	50,331,648	67,108,864	83,886,080	100,663,296	117,440,512	134,217,728
8192	67,108,864	134,217,728	201,326,592	268,435,456	335,544,320	402,653,184	469,762,048	536,870,912

2.5 Some Basic Relationships between Pixels

Given an image *f(x,y) and pixels p or q*

2.5.1 Neighbours of a pixel

• A pixel p at *(x,y) has 4 horizontal and vertical neighbours, whose coordinates are:*

(x+1,y), (x-1,y), (x,y+1), (x,y-1) → set N4(p) (4-neighbours of p)

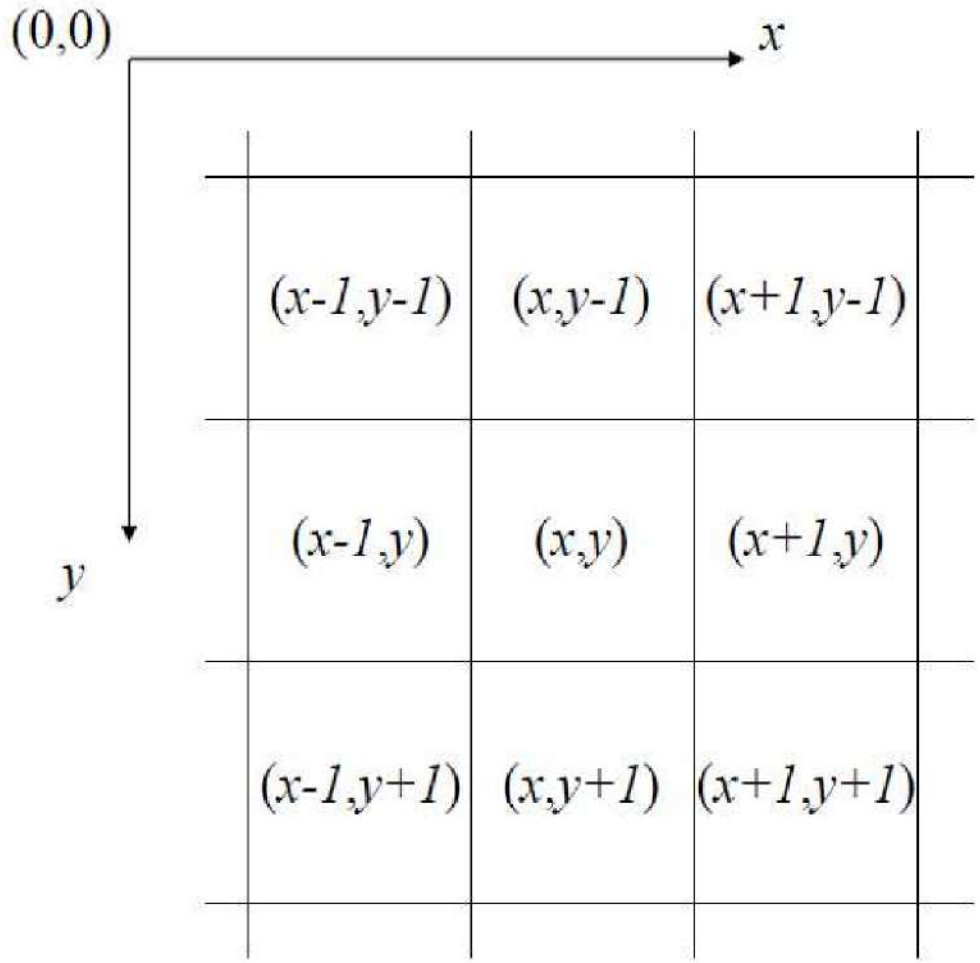
NB: each is a unit distance from p, and some of these locations lie outside the image (borders)

• The 4 *diagonal neighbours of p have coordinates:*

(x+1,y+1), (x+1,y-1), (x-1,y+1), (x-1,y-1) → set ND(p)

• N4(p) ∪ ND(p) = N8(p) : the set *of 8-neighbours of p*

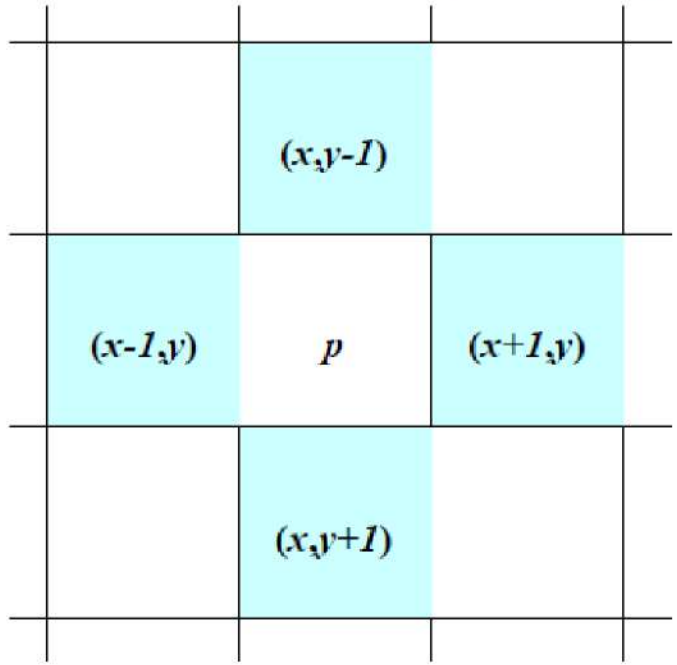
Basic Relationship of Pixels



Conventional indexing method

Neighbors of a Pixel

Neighborhood relation is used to tell adjacent pixels. It is useful for analyzing regions.



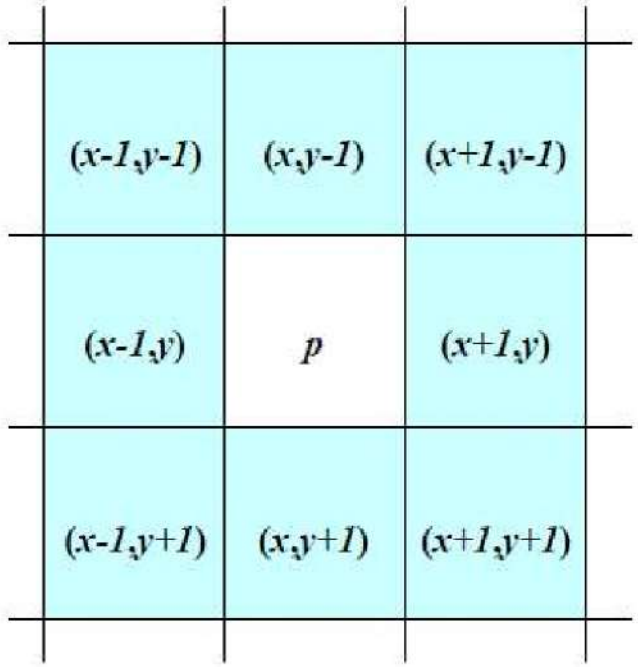
4-neighbors of *p*:

$$N_4(p) = \{ (x-1,y), (x+1,y), (x,y-1), (x,y+1) \}$$

4-neighborhood relation considers only vertical and horizontal neighbors.

Note: $q \in N_4(p)$ implies $p \in N_4(q)$

Neighbors of a Pixel (cont.)

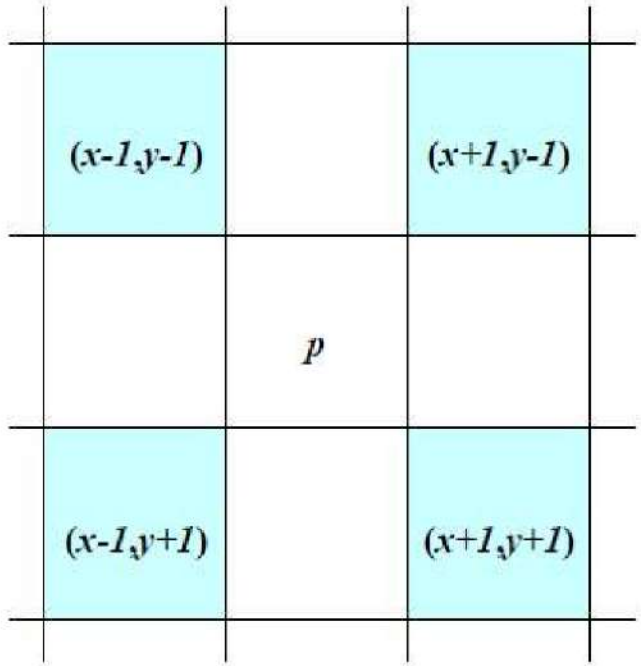


8-neighbors of *p*:

$$N_8(p) = \begin{Bmatrix} (x-1,y-1) \\ (x,y-1) \\ (x+1,y-1) \\ (x-1,y) \\ (x+1,y) \\ (x-1,y+1) \\ (x,y+1) \\ (x+1,y+1) \end{Bmatrix}$$

8-neighborhood relation considers all neighbor pixels.

Neighbors of a Pixel (cont.)



Diagonal neighbors of *p*:

$$N_D(p) = \{(x-1,y-1),(x+1,y-1),(x-1,y+1),(x+1,y+1)\}$$

Diagonal -neighborhood relation considers only diagonal neighbor pixels.

Convolution

- A convolution is essentially sliding a filter over the input
- Usually applied on image to extract features from that
- “A convolution can be thought as “looking at a function’s surroundings to make better/accurate predictions of its outcome.”
- Rather than looking at an entire image at once to find certain features it can be more effective to look at smaller portions of the image.

Filter/mask/kernel/window

- In image processing, a **kernel**, **convolution matrix**, or **mask** is a small matrix.
- It is used for blurring, sharpening, edge detection, edge enhancement, and more. This is accomplished by doing a convolution between a kernel and an image.

2	3	1	4	5
7	3	2	0	9
4	3	2	1	7
6	4	9	0	2
1	3	4	9	8

Image

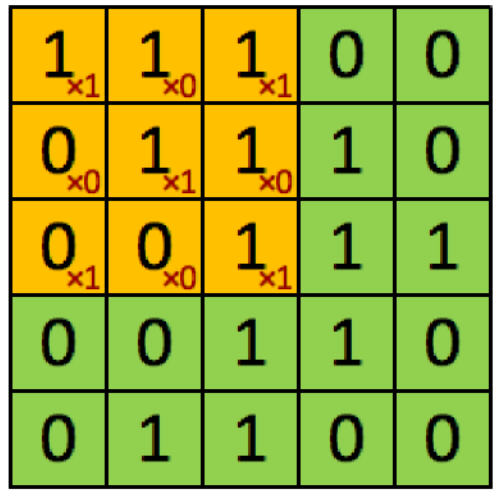
5 X 5

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

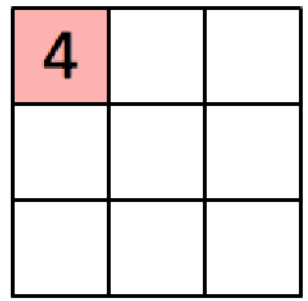
3 x 3

filter

- **Padding-** To handle the edge pixels there are several approaches:
- Padding with zero value pixels
- Reflection padding
- Losing the edge pixels



Image



Convolved Feature

- The array you end up with is called a **feature map** or an **activation map**!
- It's smaller than the original input image if no padding is applied
- Stride- By how many pixels the filter is sliding through the image

- In convolutional network terminology, the first argument to the convolution is often referred to as the **input**
- Second argument as the **kernel**.
- The output is sometimes referred to as the **feature map**.

Depth of the filter

- Depth of the filter depends on (and is equal to) the number of channels of the input image.
- So, we apply a **3X3X1** convolution filter on gray-scale images (the number of channels = 1)
- we apply a **3X3X3** convolution filter on a colored (RGB) image (the number of channels = 3).

Fundamentals of Spatial Filtering

- Spatial filters (also called spatial *masks, kernels, templates or windows)*
- *Filtering refers to accepting(passing) or rejecting certain frequency components.*
- *Filter that passes low frequency is called lowpass filter*

Example : linear spatial filter using a 3x3 neighbourhood:

At any point *(x,y)* in the original image, response *g(x,y)* of the filter:

$$g(x,y) = w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y) + \ldots$$
$$+w(0,0)f(x,y) + \cdots + w(1,1)f(x+1,y+1)$$

Centre coefficient of filter

3.4.1 The Mechanics of Spatial Filtering

For a mask of size mxn, we assume:

$m = 2a + 1$ and $n = 2b + 1$

where a and b are positive integers (*odd size*)

General expression of a linear spatial filtering of an image of size M x N with a filter of size m x n :

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

(each pixel in w visiting every pixel in f)

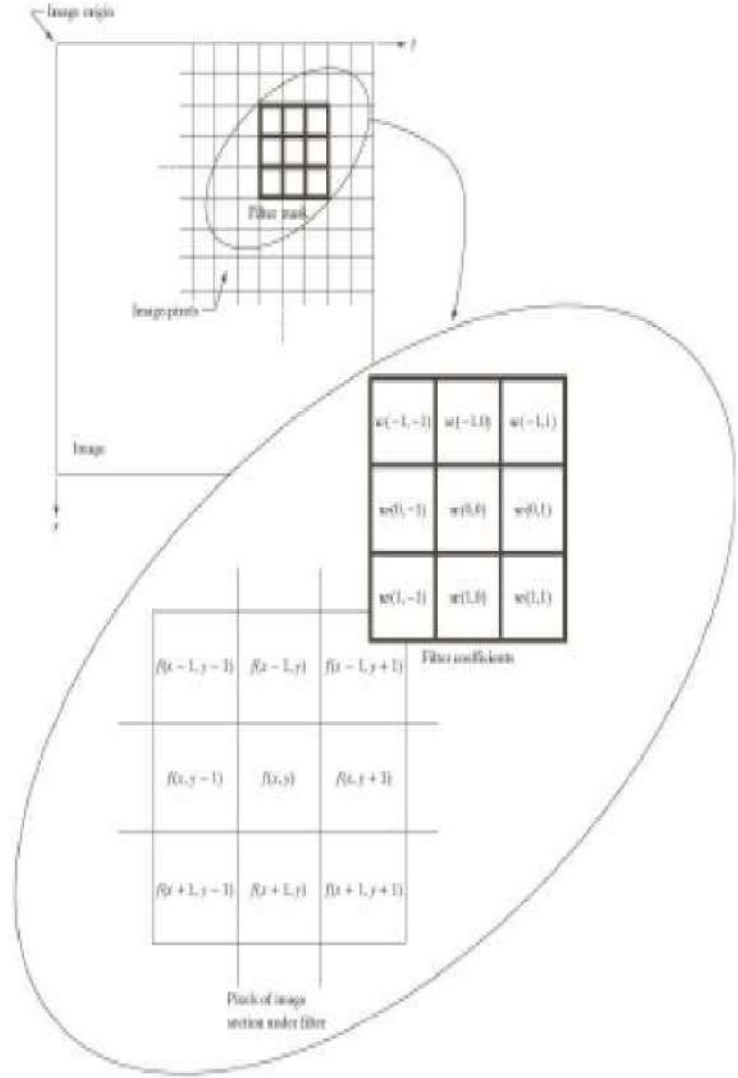
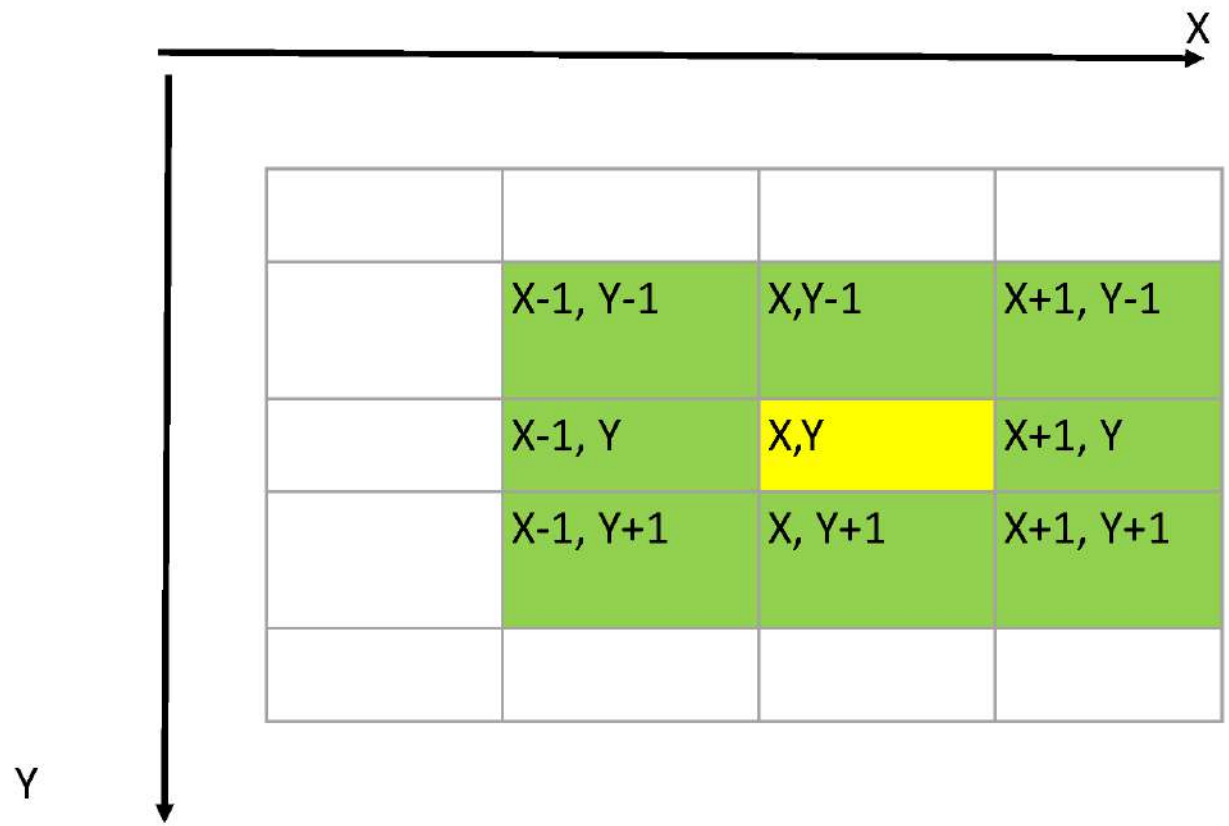
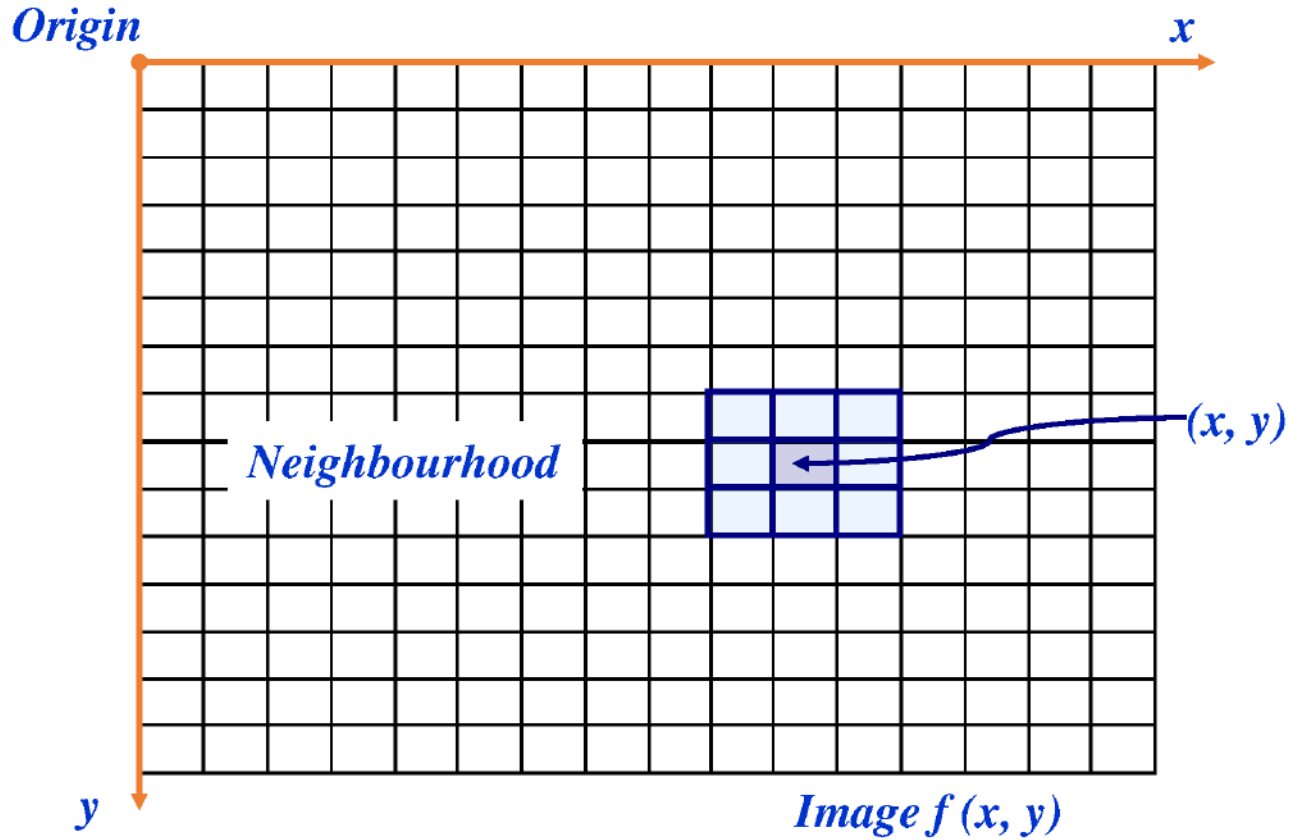


FIGURE 3.28 The mechanics of linear spatial filtering using a 3 × 3 filter mask. The form chosen to denote the coordinates of the filter mask coefficients simplifies writing expressions for linear filtering.

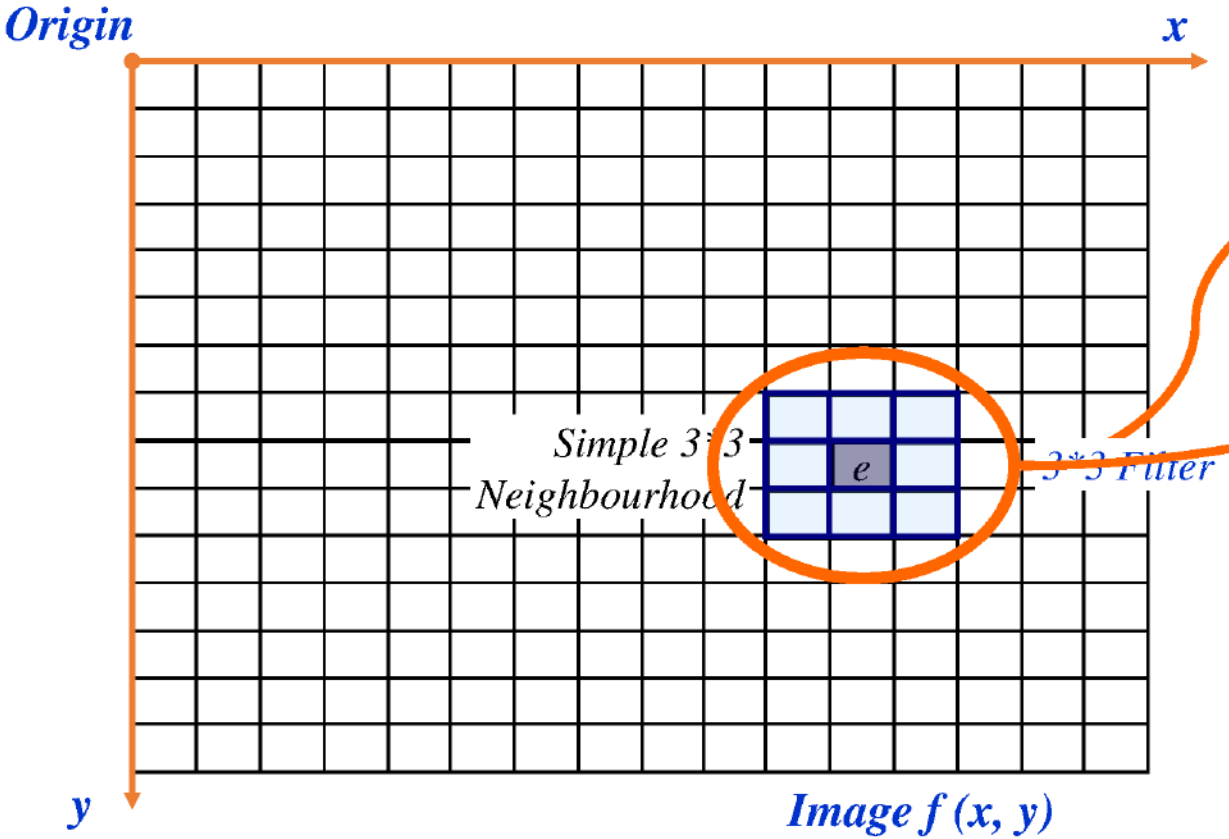

X
Y
X-1, Y-1
X,Y-1
X+1, Y-1
X-1, Y
X,Y
X+1, Y
X-1, Y+1
X, Y+1
X+1, Y+1

Neighbourhood Operations

- Neighbourhood operations simply operate on a larger neighbourhood of pixels than point operations
- Neighbourhoods are mostly a rectangle around a central pixel
- Any size rectangle and any shape filter are possible



The Spatial Filtering Process



a	*b*	*c*
d	*e*	*f*
g	*h*	*i*

Original Image Pixels

*

r	*s*	*t*
u	*v*	*w*
x	*y*	*z*

Filter

$$e_{processed} = v*e + r*a + s*b + t*c + u*d + w*f + x*g + y*h + z*i$$

The above is repeated for every pixel in the original image to generate the filtered image

- How to relate the convolution with the neural network?
- image intensity values – inputs
- Filter values- weights

Different types of filters

- Low pass- to extract high frequency component – used to blur the image
- High pass – to extract high frequency component like edges, lines etc.

−1	−1	−1
2	2	2
−1	−1	−1

Horizontal

−1	−1	2
−1	2	−1
2	−1	−1

+45°

−1	2	−1
−1	2	−1
−1	2	−1

Vertical

2	−1	−1
−1	2	−1
−1	−1	2

−45°

0	0	2	0	0
0	0	2	0	0
0	0	2	0	0
0	0	2	0	0
0	0	2	0	0

Image

5 X 5

0	**-4**	**8**	**-4**	**0**
0	-6	12	-6	0
0	-6	12	-6	0
0	-6	12	-6	0
0	-4	8	-4	0

Convolutions have been used for a while

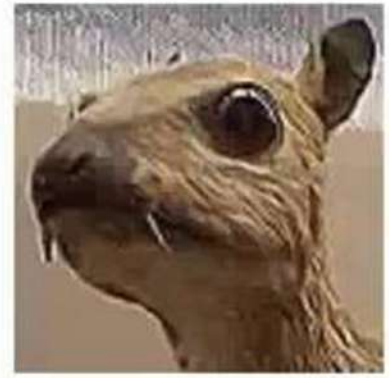
Kernel

Original image

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

Edge detection

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



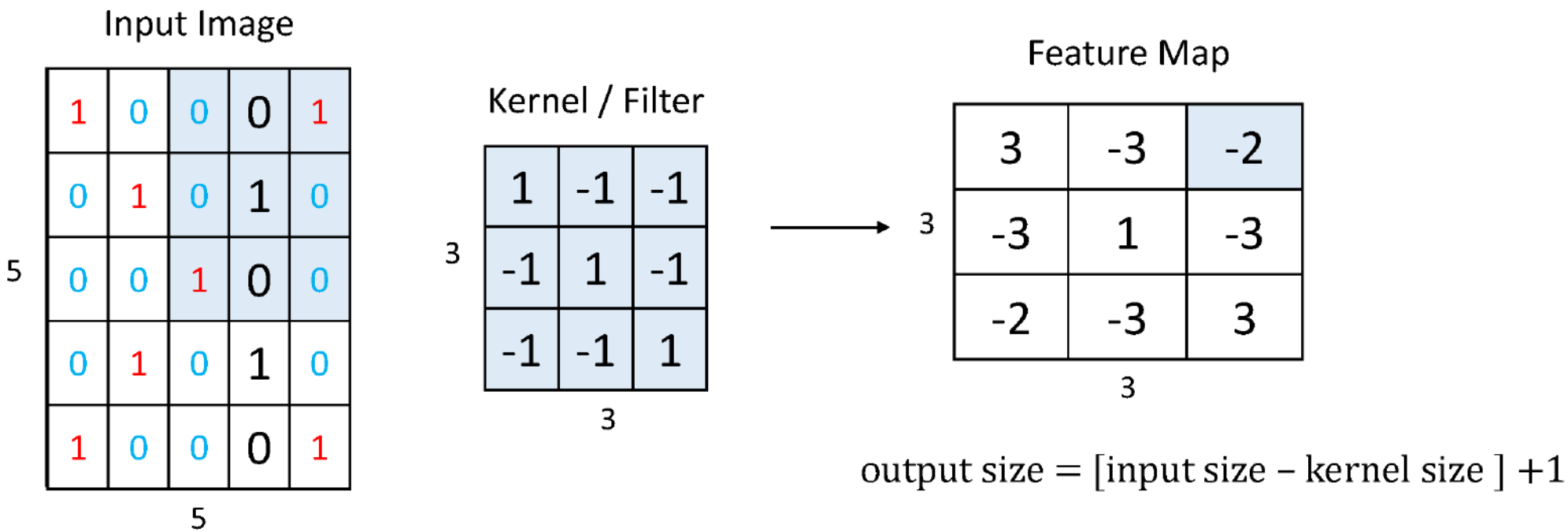
Sharpening

$$*\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$

Blurring

Step 2: Convolution

- Convolution is an element-wise multiplication of two matrices (sub-matrix) followed by a sum
- Eg: without padding



$$\text{output size} = [\text{input size} - \text{kernel size}\ ] + 1$$

$$= [5 - 3] + 1$$

1	0	0	0	1
0	1	0	1	0
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1

1	-1	-1
-1	1	-1
-1	-1	1

Stride = 1

3	-3	-2
-3	1	-3
-2	-3	3

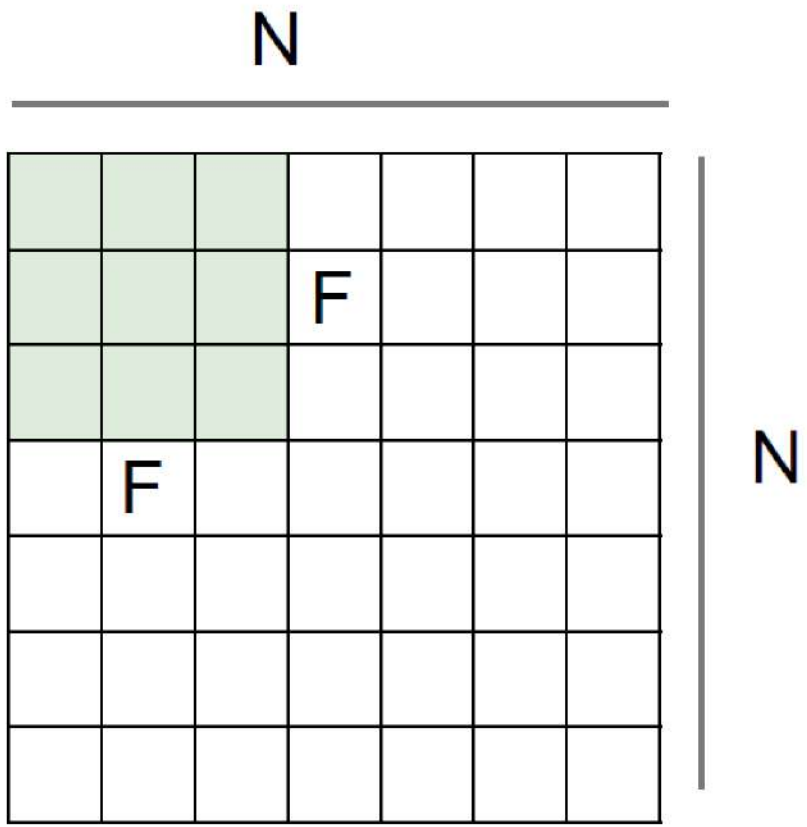
1	0	0	0	1
0	1	0	1	0
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1

1	-1	-1
-1	1	-1
-1	-1	1

Stride = 2

3	-2
-2	3

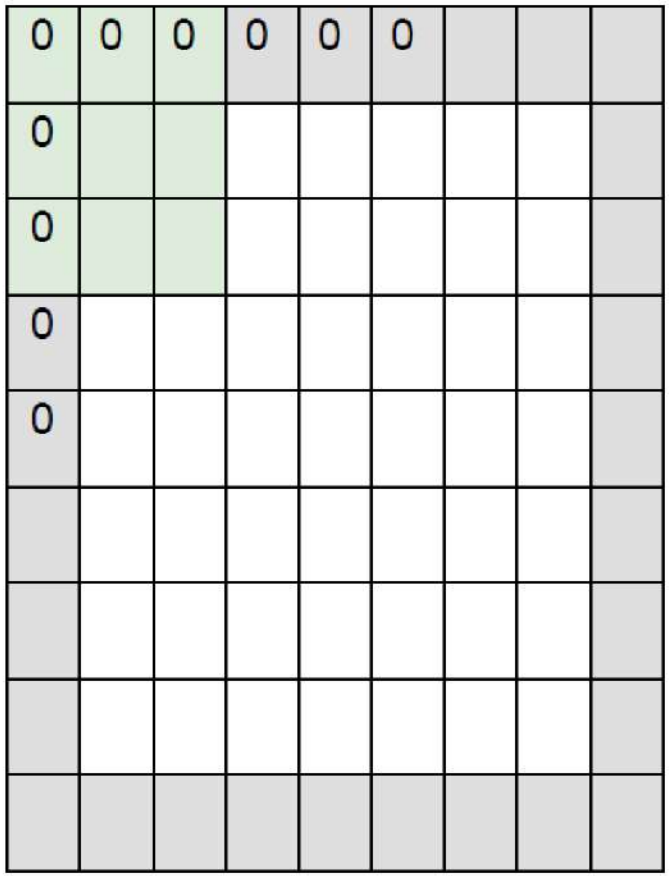
$$\text{Output size(without padding)} = \frac{[\text{input size} - \text{kernel size}]}{\text{stride}} + 1$$



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 :\

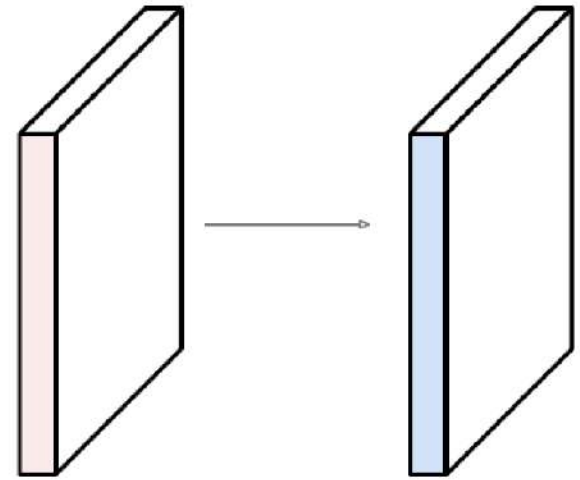
Zero-Padding: common to the border



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

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Examples time:

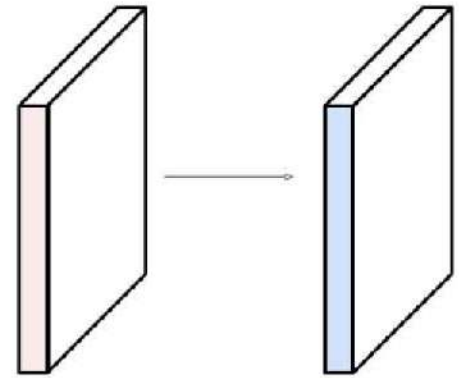


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

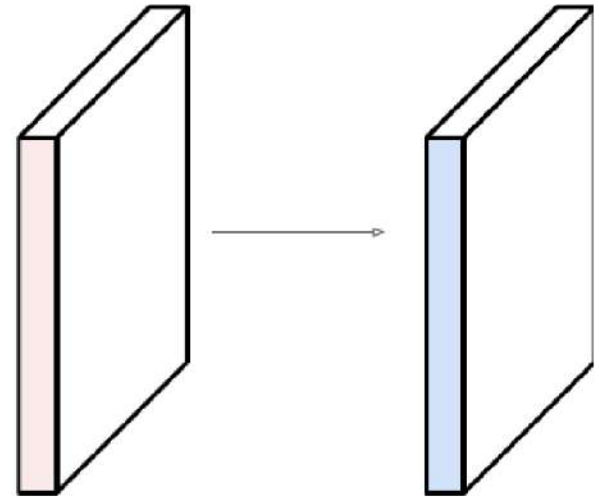
Examples time:



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

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Examples time:



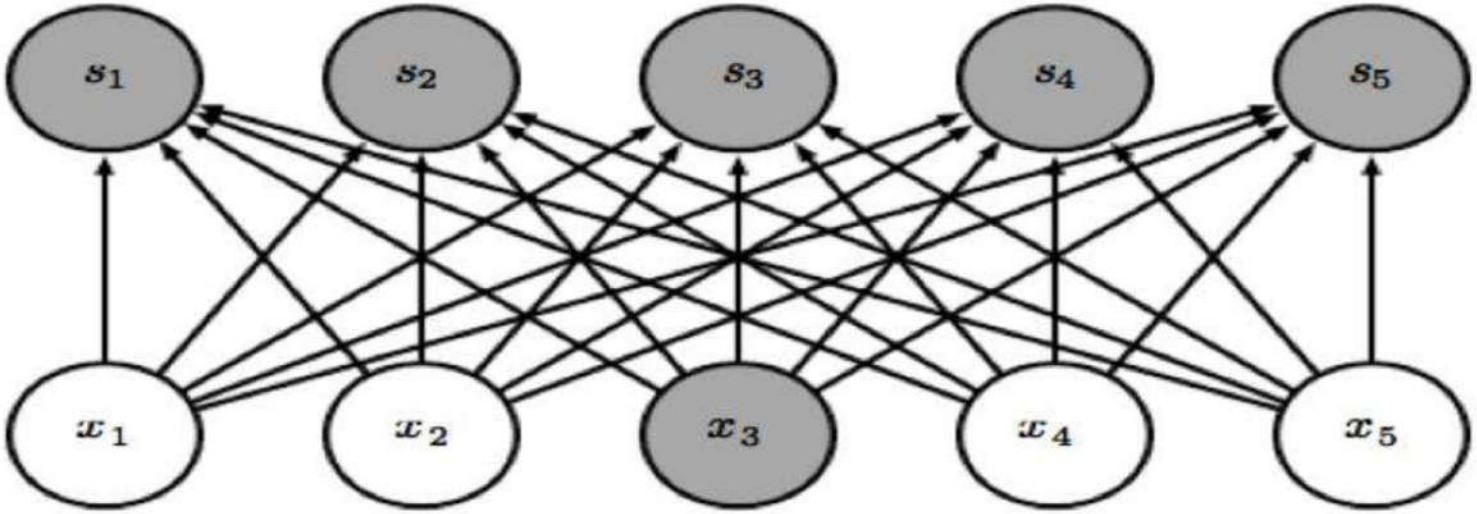
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

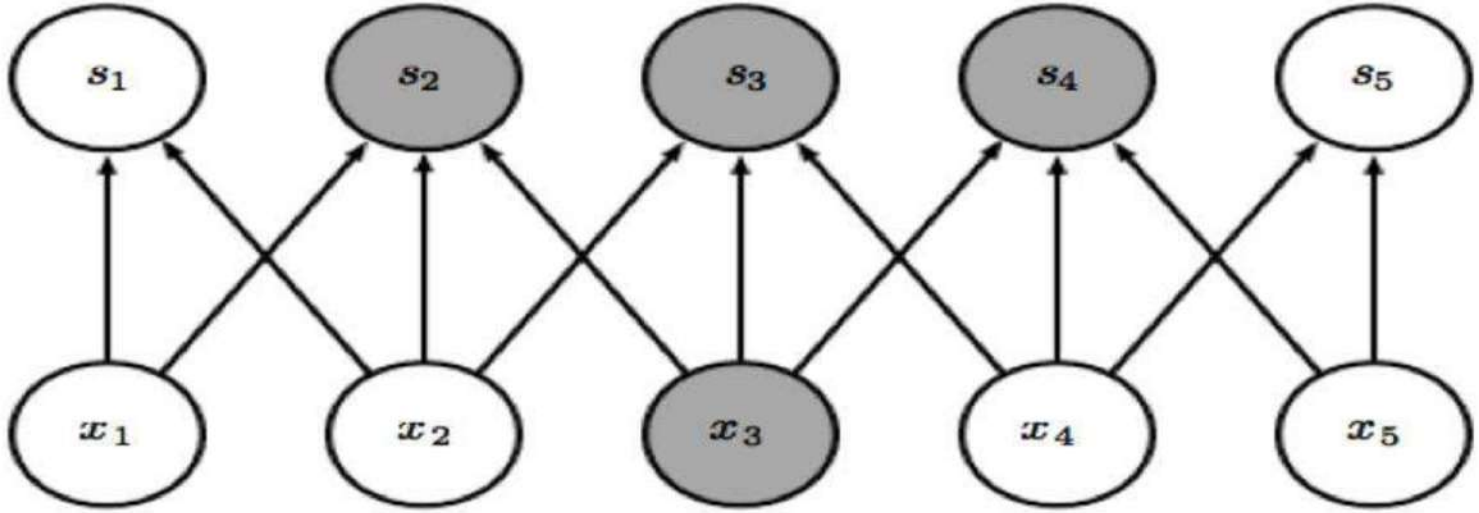
Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

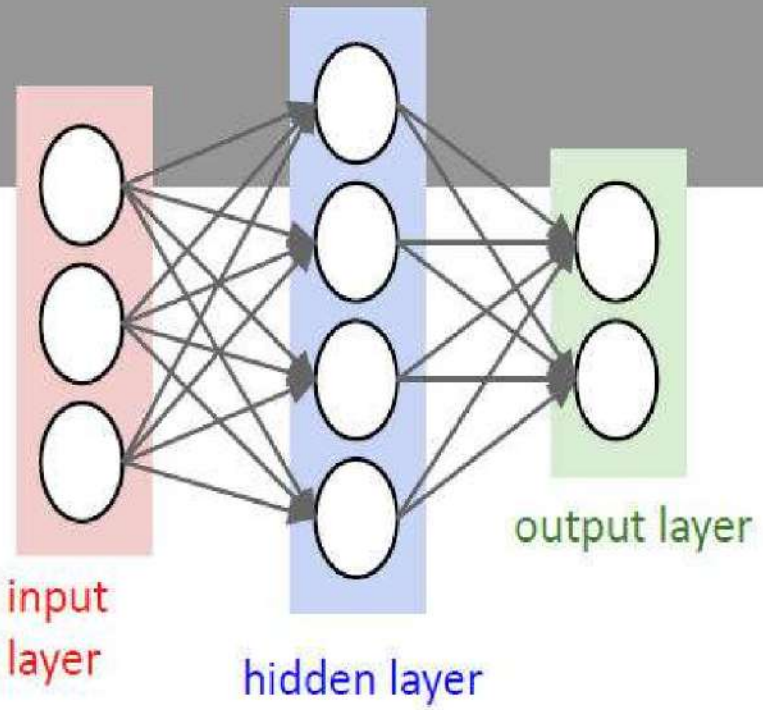
3 reasons why convolution is cool

Reason 1 : Sparse Connectivity

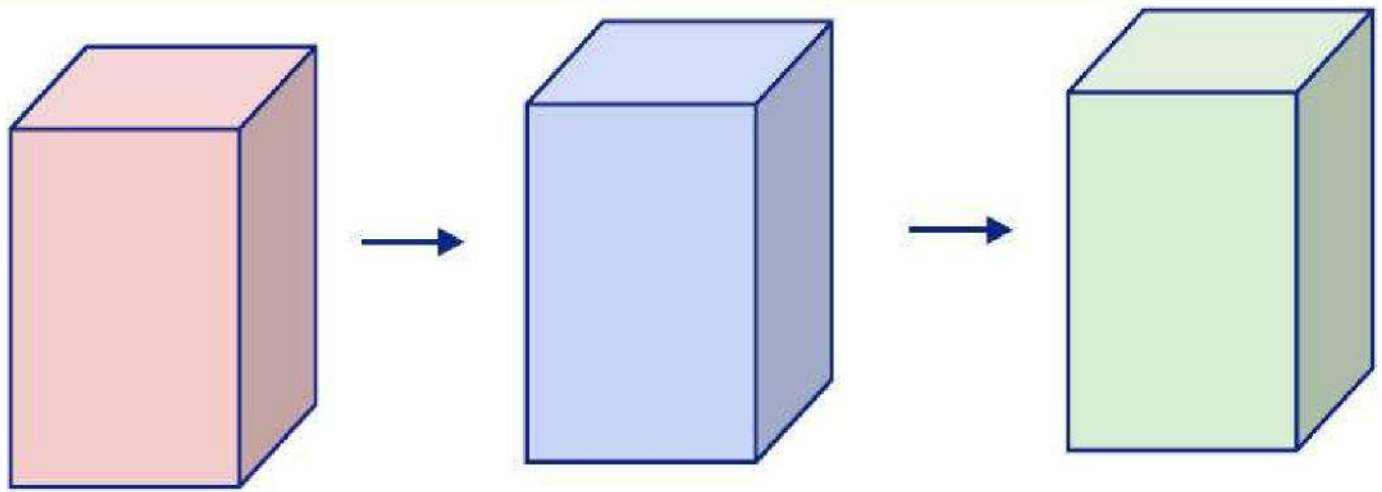




Fully connected

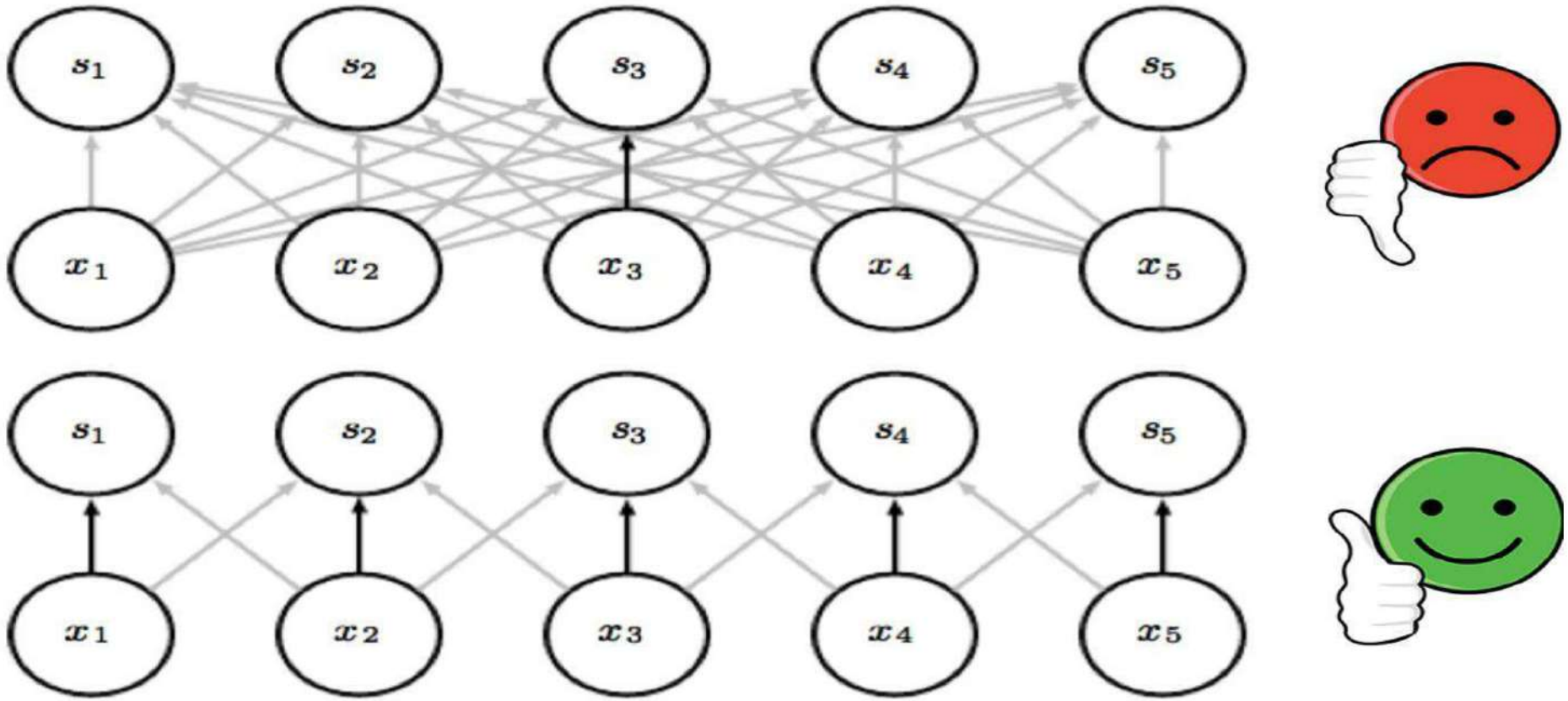


Convolutional
:



- Traditional neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit.
- This means **every output unit interacts** with every input unit.
- CNN has **sparse interactions** (also referred to as sparse connectivity or sparse weights).
- This is accomplished by making the **kernel smaller than the input**.
- When processing an image, the input image might have thousands or millions of pixels, but **we can detect small, meaningful features** such as edges with kernels that occupy only tens or hundreds of pixels.
- This means that we need to store fewer parameters, which both **reduces the memory requirements** of the model and improves its statistical efficiency.

Reason 2 : Parameter sharing



- Parameter sharing refers to using the **same parameter for more than one function in a model**
- In a traditional neural net, **each element of the weight matrix is used exactly once** when computing the output of a layer.
- It is multiplied by one element of the input and then never revisited. a network has **tied weights**, because the value of the weight applied to one input is tied to the value of a weight applied elsewhere.
- In CNN, each member of the kernel is used at every position
- of the input

Reason 3 : Equivariant Representations

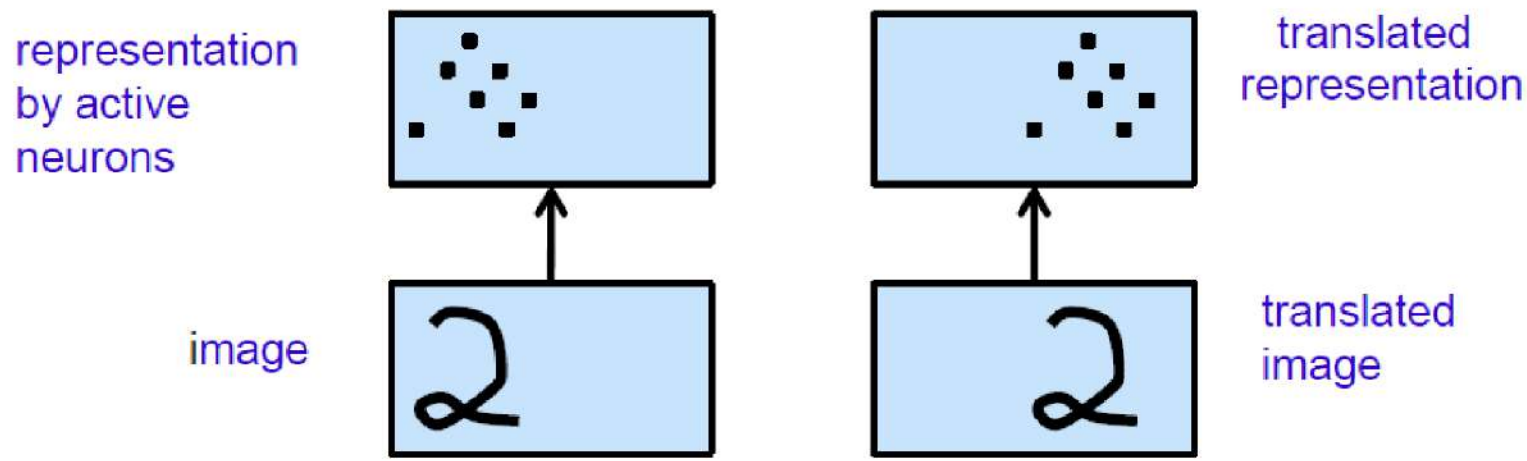
When the input changes -> output changes in the same way

Eg. Let I be a function giving images brightness at integer coordinates
Let g be a function mapping one image function to another image function,
such that $I' = g(I)$ is the image function with $I'(x,y) = I(x - 1,y)$.
This shifts every pixel of I one unit to the right.
If we apply this transformation to I, then apply convolution,
the result will be the same as if we applied convolution to I',
then applied the transformation g to the output.

Reason 3 : Equivariant Representations

When the input changes -> output changes in the same way

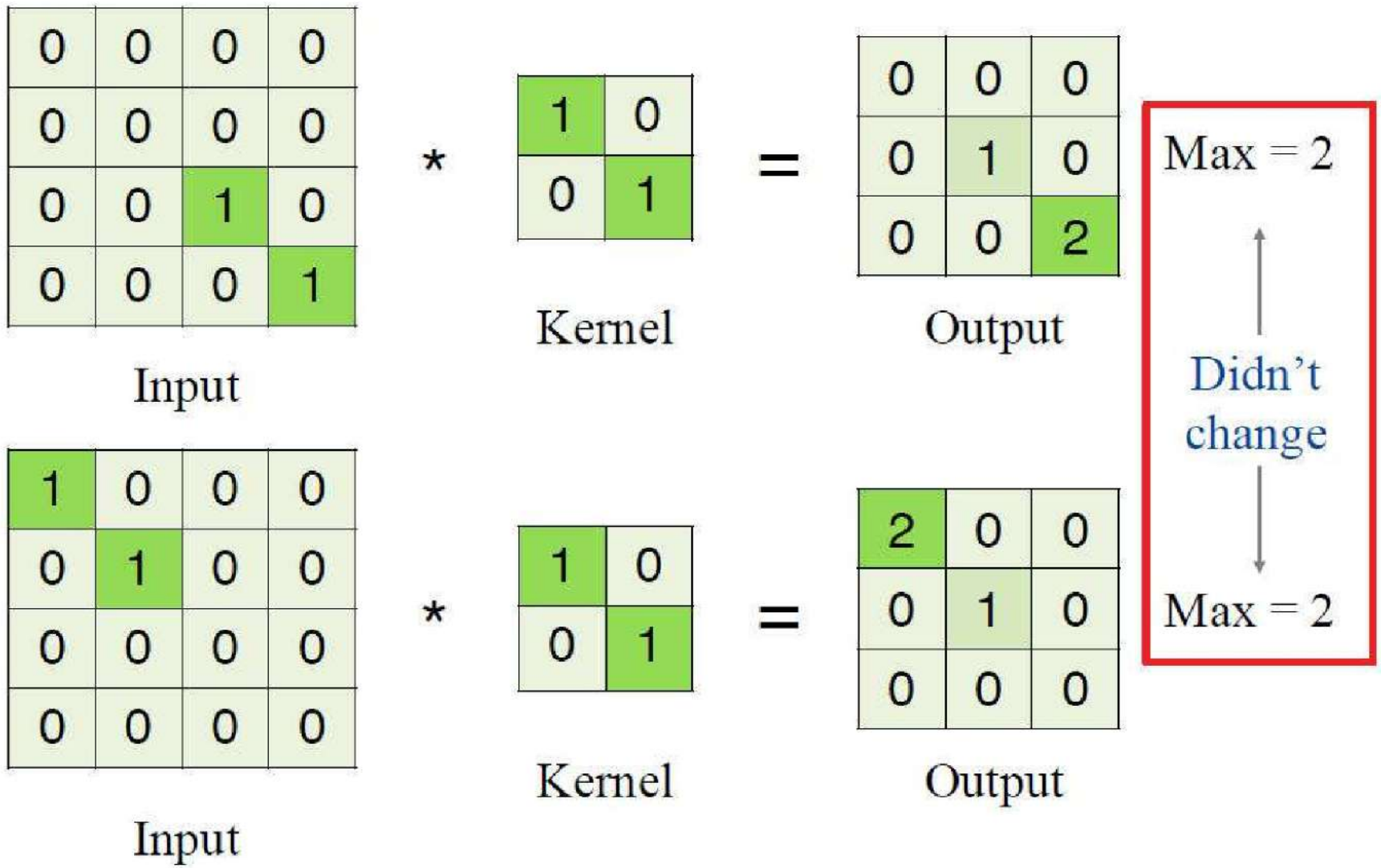
Eg. Let I be a function giving images brightness at integer coordinates
Let g be a function mapping one image function to another image function,
such that $I' = g(I)$ is the image function with $I'(x,y) = I(x - 1,y)$.
This shifts every pixel of I one unit to the right.
If we apply this transformation to I, then apply convolution,
the result will be the same as if we applied convolution to I',
then applied the transformation g to the output.



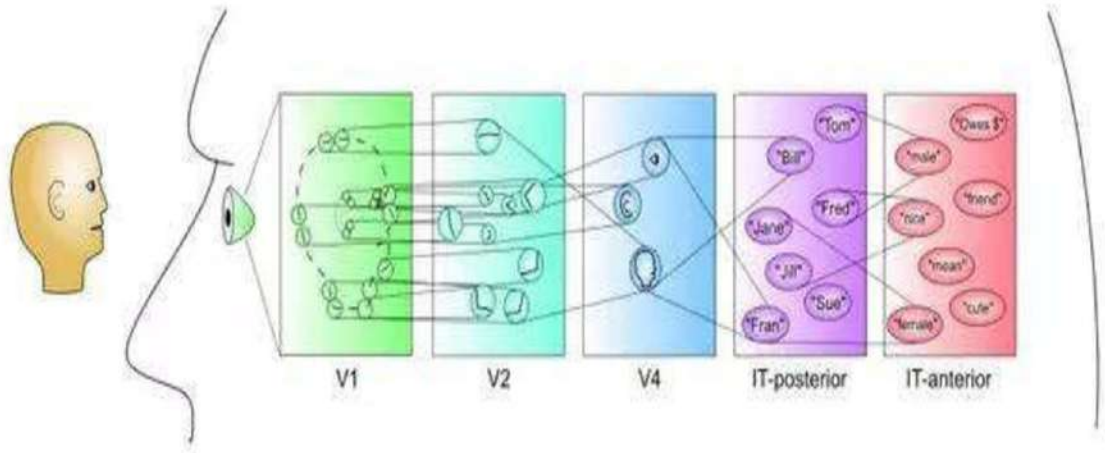
- Invariant knowledge: If a feature is useful in some locations during training, detectors for that feature will be available in all locations during testing.

Reason 3 contd.

How do we maintain translation invariance?



What inspired convolution neural networks?



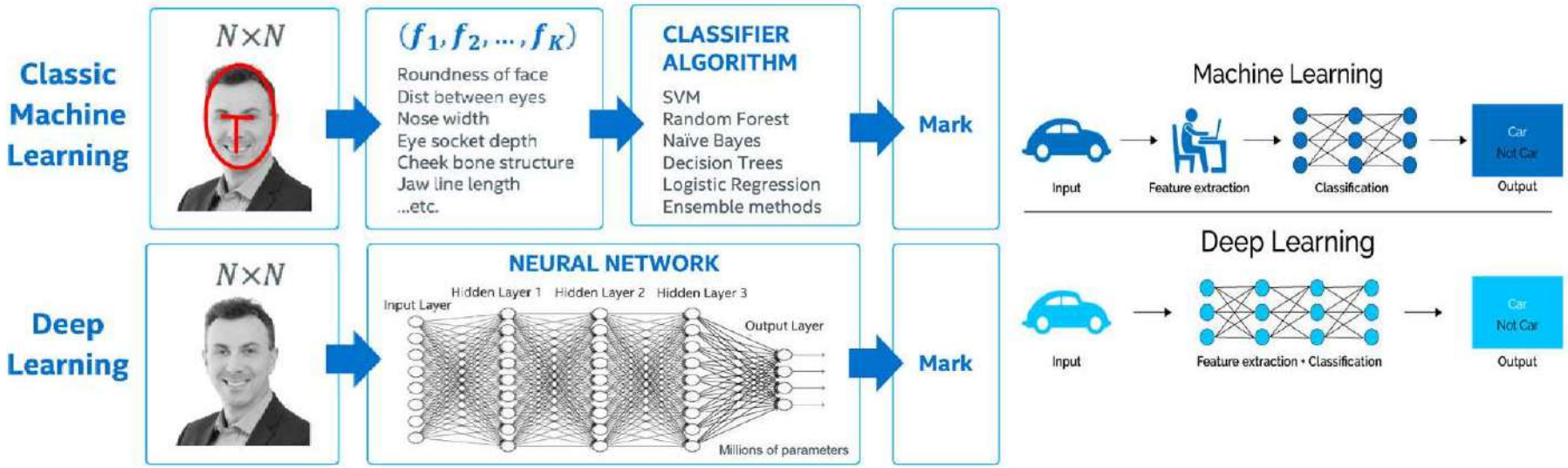
The architecture of deep convolutional neural networks was inspired by the ideas mentioned above

- local connections

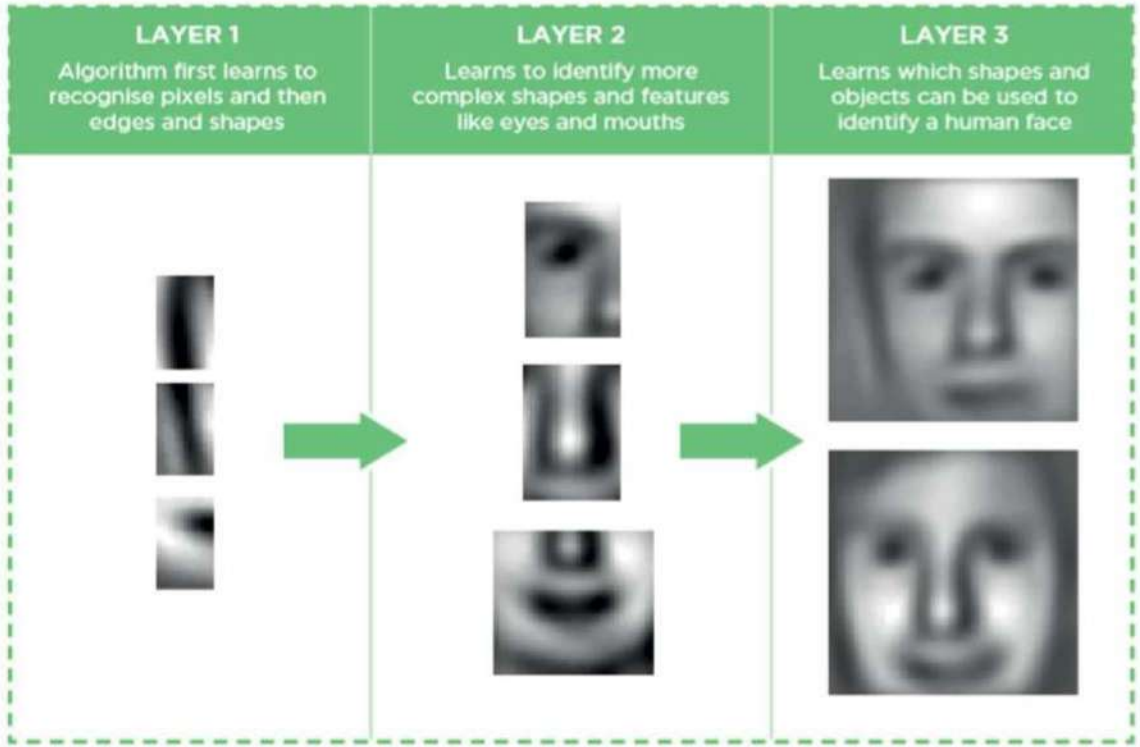
- layering

- spatial invariance (shifting the input signal results in an equally shifted output signal. , most of us are able to recognize specific faces under a variety of conditions because we learn abstraction These abstractions are thus invariant to size, contrast, rotation, orientation

Neural feature extraction vs classical ML

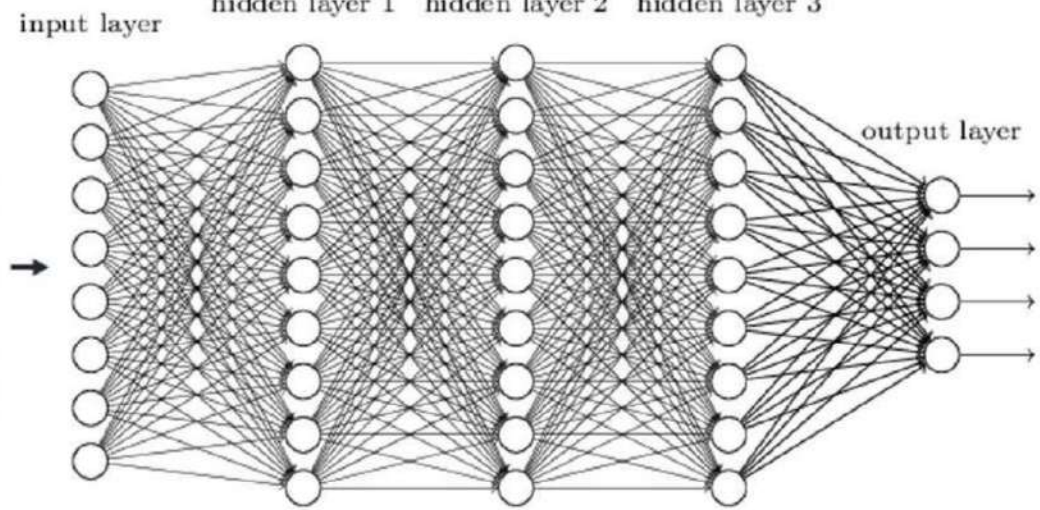


What are layers and what happens inside?



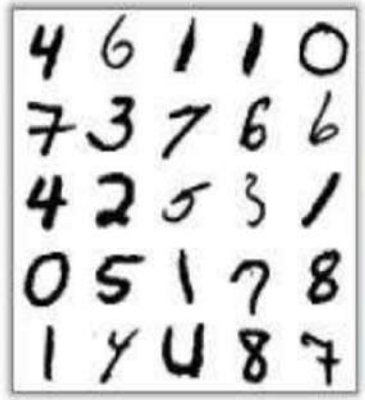
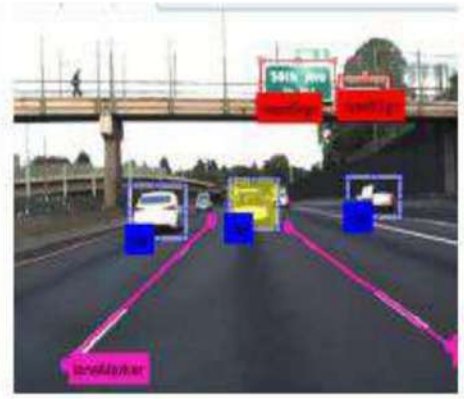
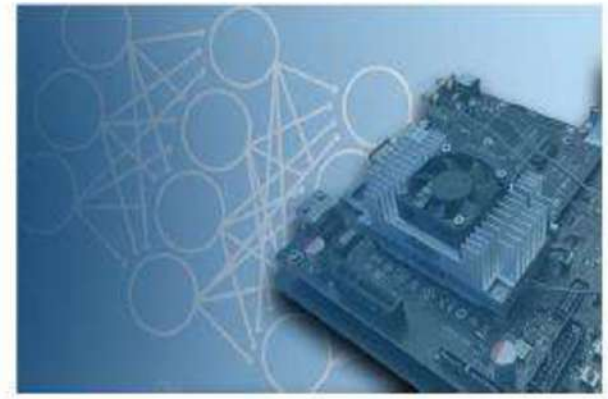
What are layers and what happens inside?

Deep neural networks learn hierarchical feature representations



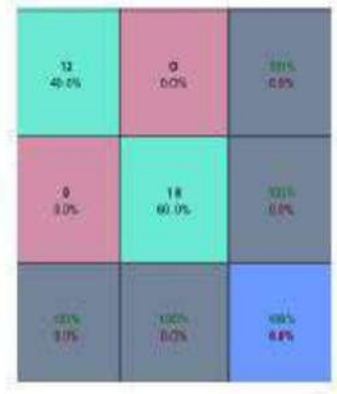
Disadvantages of using ANN for image classification

1. Too much computation
2. Treats local pixels same as pixels far apart
3. Sensitive to location of an object in an image

What is Deep Learning?



Deep learning is a type of machine learning in which a model learns to perform tasks directly from images, text, or sound.

Deep learning is usually implemented using a **neural network.**

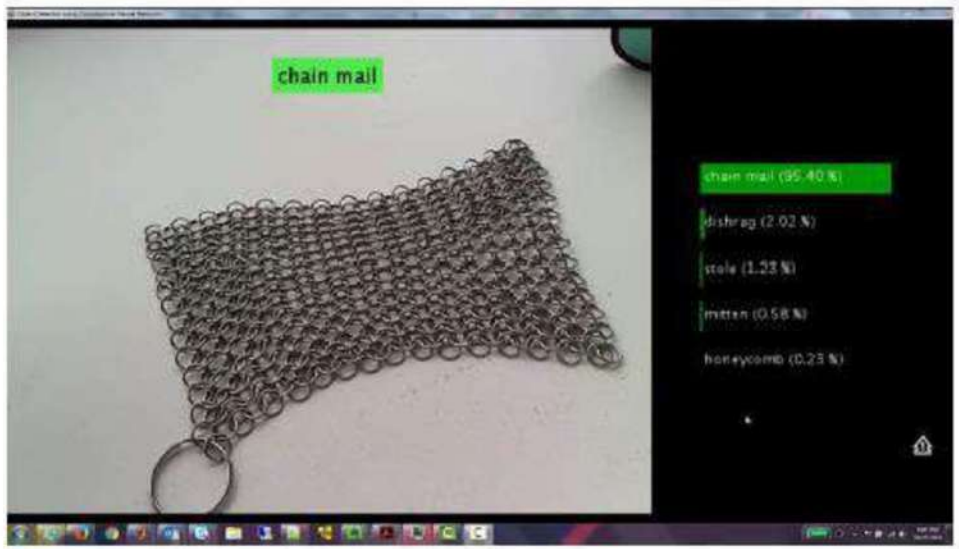
The term “deep” refers to the **number of layers** in the network—the more layers, the deeper the network.

How Companies Use CNNs

Data, data, data.

- The companies that have lots of this magic 4 letter word are the ones that have an inherent advantage over the rest of the competition.
- The more training data that you can give to a network, the more training iterations you can make, the more weight updates you can make, and the better tuned to the network is when it goes to production.
- Facebook uses neural nets for their automatic tagging algorithms, Google for their photo search, Amazon for their product recommendations, Pinterest for their home feed personalization, and Instagram for their search infrastructure.

Object recognition using deep learning



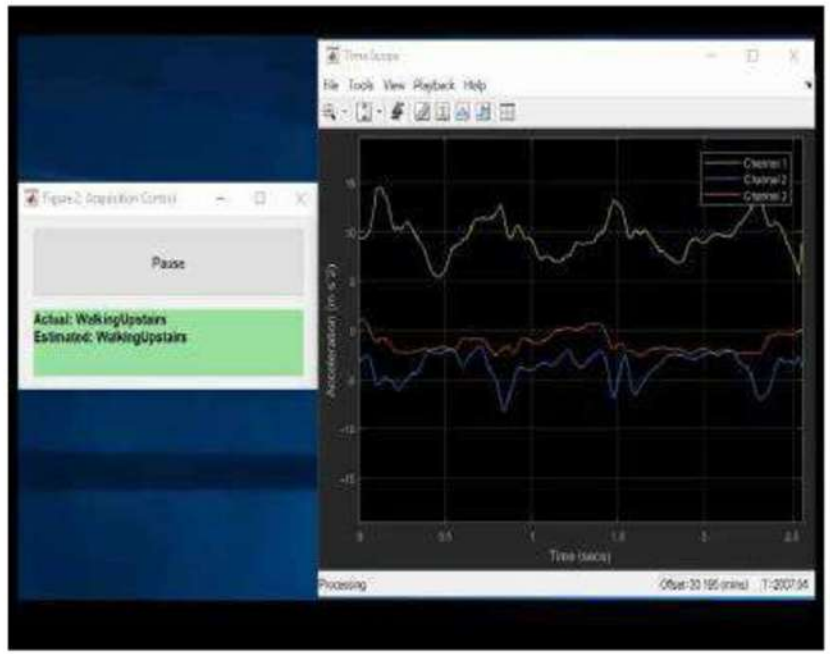
Training (GPU)	Millions of images from 1000 different categories
Prediction	Real-time object recognition using a webcam connected to a laptop

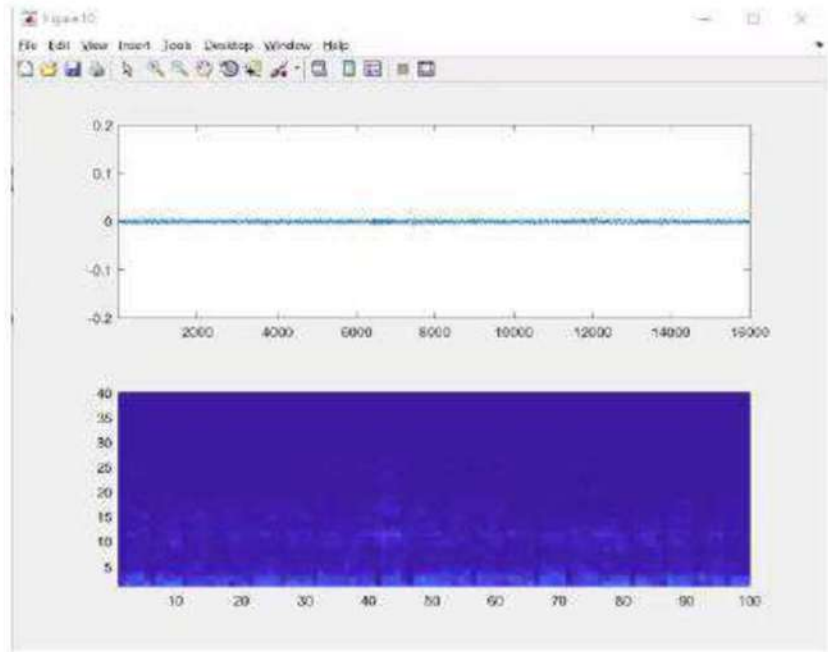
Detection and localization using deep learning

Regions with Convolutional Neural Network Features (R-CNN)

Semantic Segmentation using SegNet

Analyzing signal data using deep learning





Signal Classification using LSTMs

Speech Recognition using CNNs

Example: Food classifier using deep transfer learning

Hot dog →

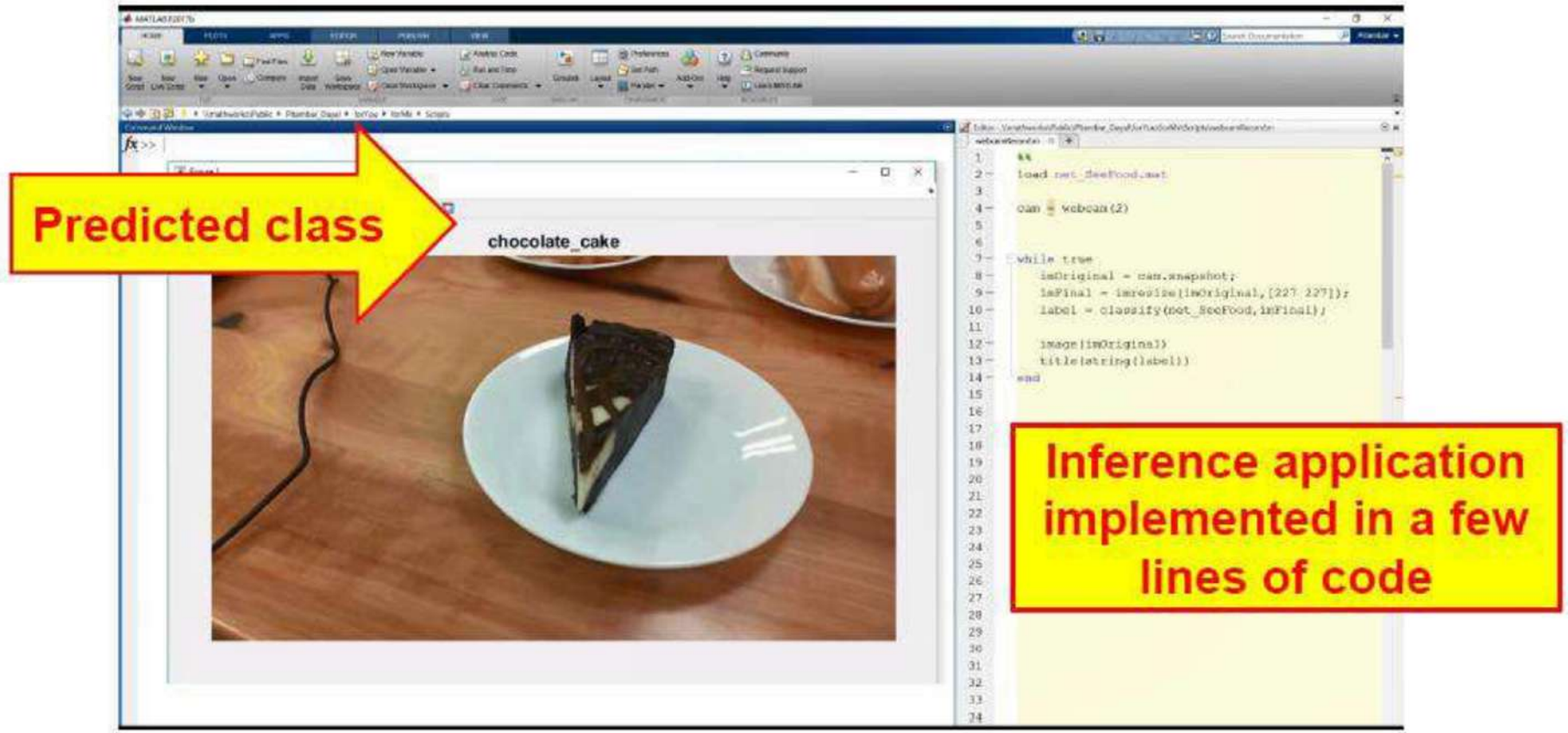
French fries →

Chocolate cake →

Pizza →

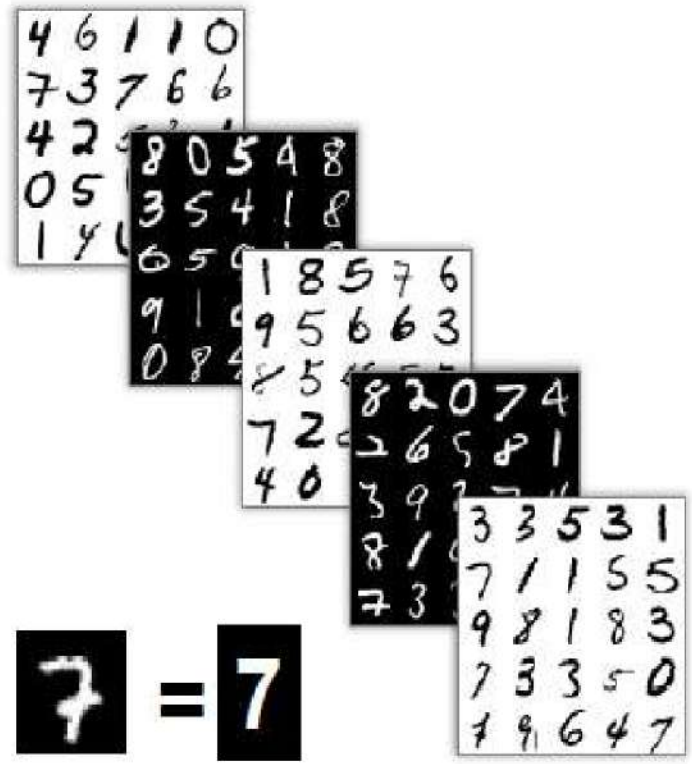
Ice cream →

5 Category Classifier


Predicted class
chocolate_cake
Inference application implemented in a few lines of code

MNIST: The “Hello, World!” of computer vision

What?	A set of handwritten digits from 0-9
Why?	An easy task for machine learning beginners
How many?	60,000 training images 10,000 test images
Best results?	99.79% accuracy



Sources: http://yann.lecun.com/exdb/mnist/
https://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results

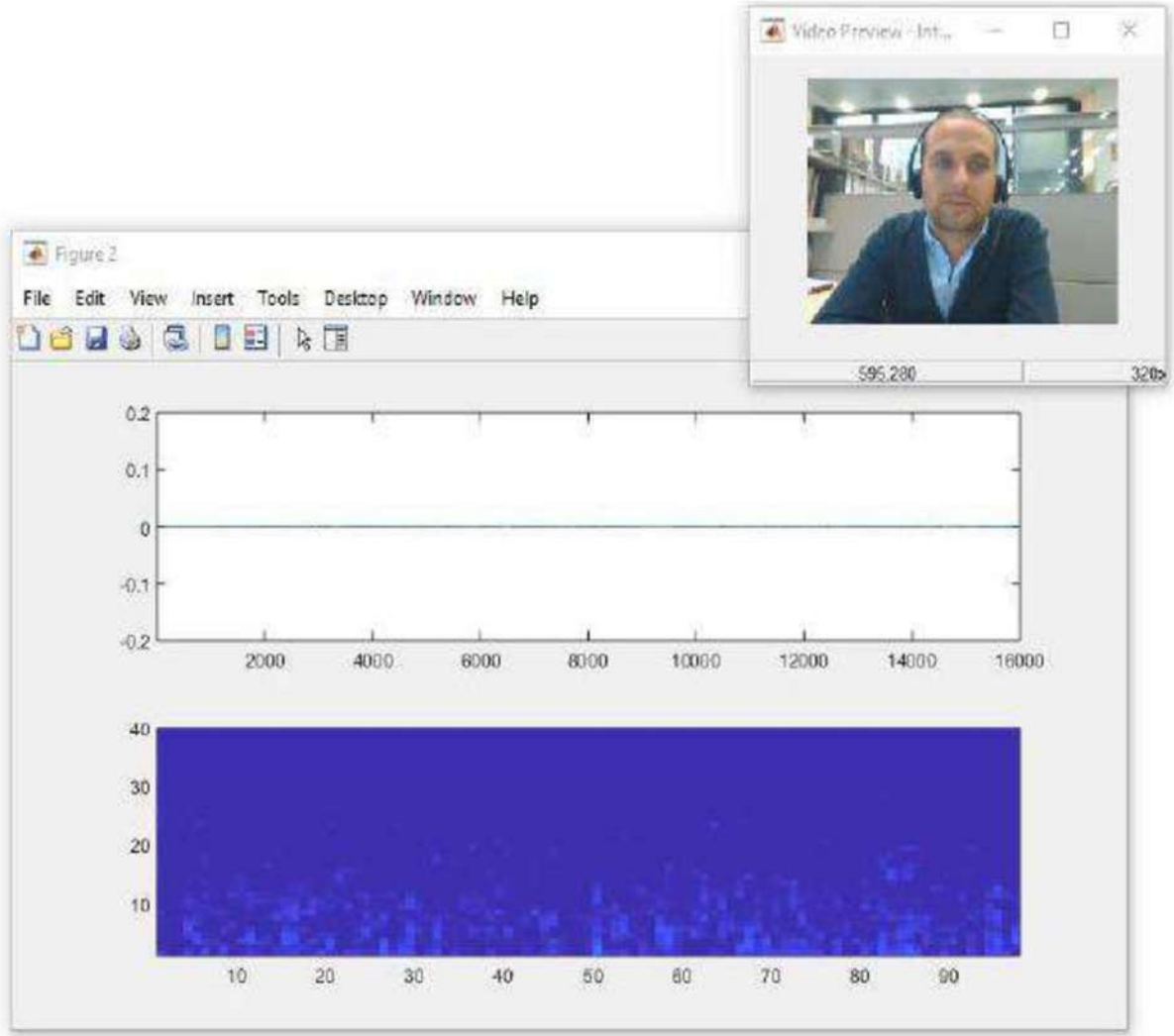
Demo: **Speech Command Recognition Using Deep Learning**

Commands

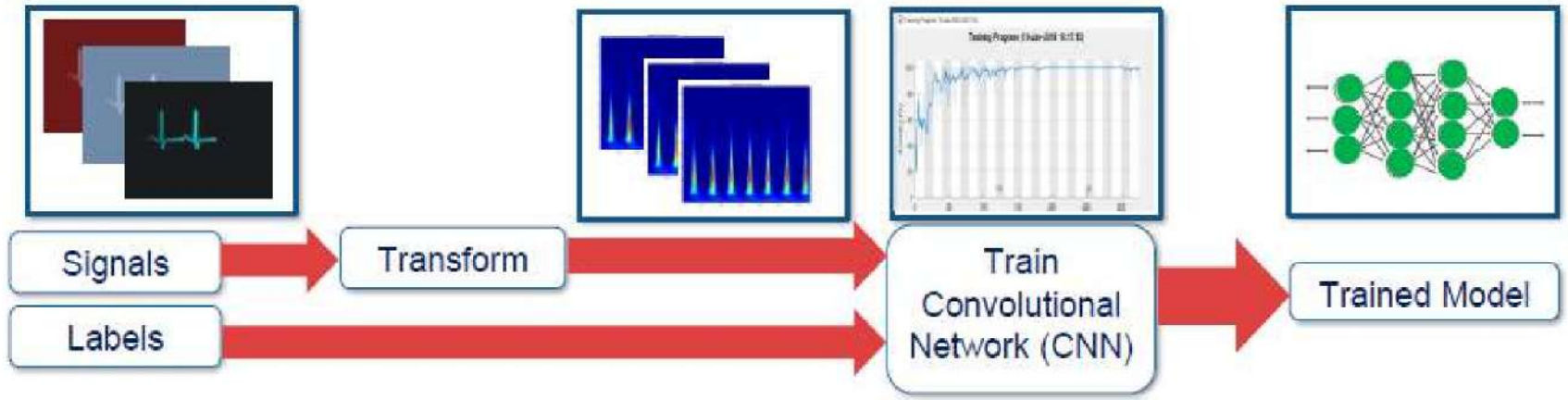
- Yes
- No
- Up
- Down
- Left
- Right
- On
- Off
- Stop
- Go

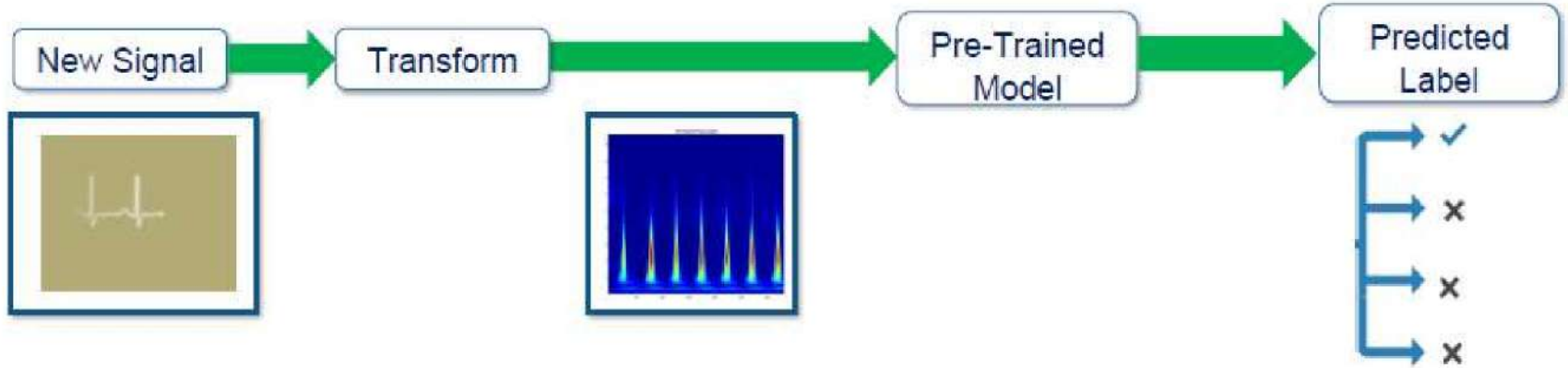
Non-Commands
(= Unknown)

- Bed
- Bird
- Cat
- Dog
- Happy
- House
- Marvin
- Sheila
- Tree
- Wow
- Zero
- One
- Two
- Three
- Four
- Five
- Six
- Seven
- Eight
- Nine

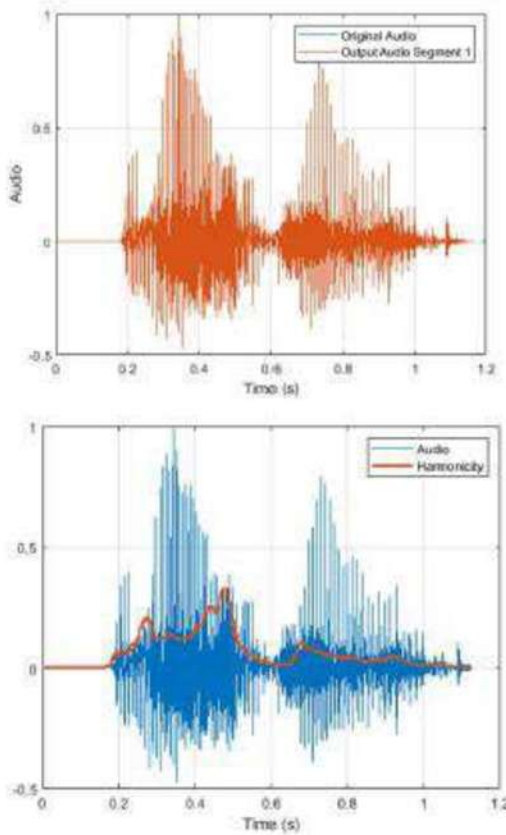


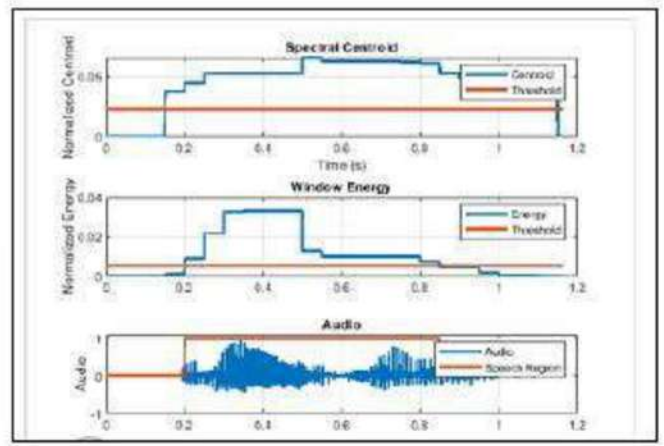
How can things work with Audio?



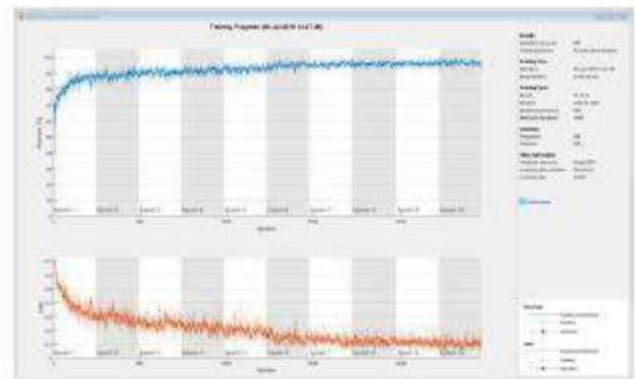


Classify Speaker Using LSTMs






```
layers = [ ...
    sequenceInputLayer(NumFeatures)
    bilstmLayer(100,"OutputMode","sequence")
    reluLayer
    bilstmLayer(100,"OutputMode","last")
    fullyConnectedLayer(2)
    softmaxLayer
    classificationLayer];
```



Deep Learning for Text Generation

More from Chatbots Life

5 Incredible Ways in Which Chatbots Can Enhance Custome...

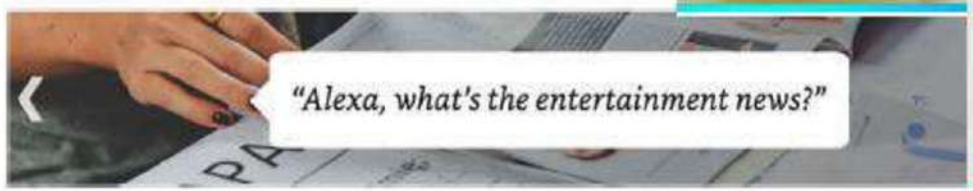
THE ALGORITHMS BEHIND THE HEADLINES

How machine-written news redefines the core skills of human journalists

Arjen van Dalen

Pages 648-658 | Published online: 30 Mar 2012

Download citation https://doi.org/10.1080/17512786.2012.667268



Typically, integrations have focused on what a user can say to Alexa, not what she can say back. With Wordsmith, we get human-sounding responses that naturally use different language each time.

Narrative Science Brings Natural Language to Qlik Visualization Software

Word-By-Word Text Generation Using Deep Learning

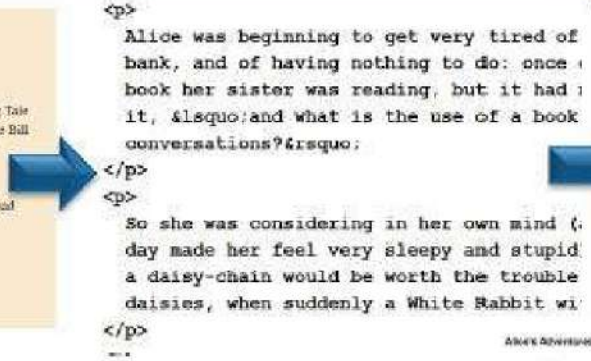
Access and Explore Data

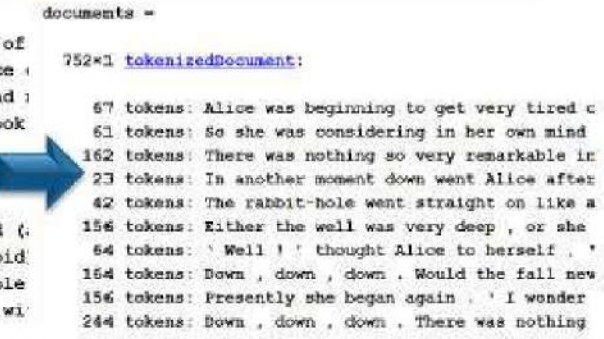
Preprocess Data

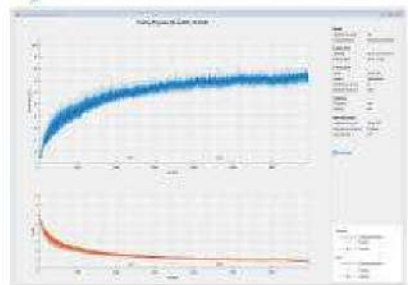
Clean-up Text

Convert to Numeric

Develop Predictive Models







- Word Docs
- PDF's
- Text Files

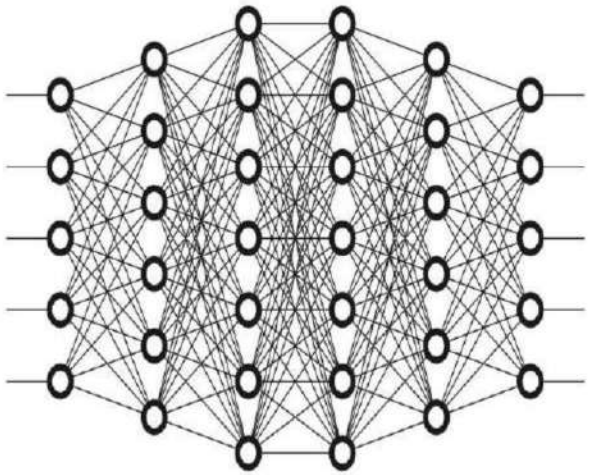
- Stop Words
- Stemming
- Tokenization

- Bag of Words
- TF-IDF
- Word Embeddings

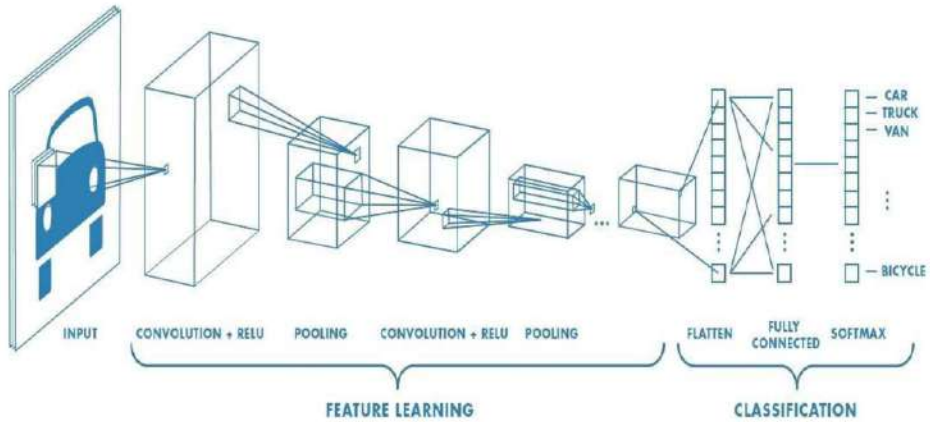
Deep Learning Architecture

https://towardsdatascience.com/convolution-neural-network-for-image-processing-using-keras-dc3429056306

Convolution neural network



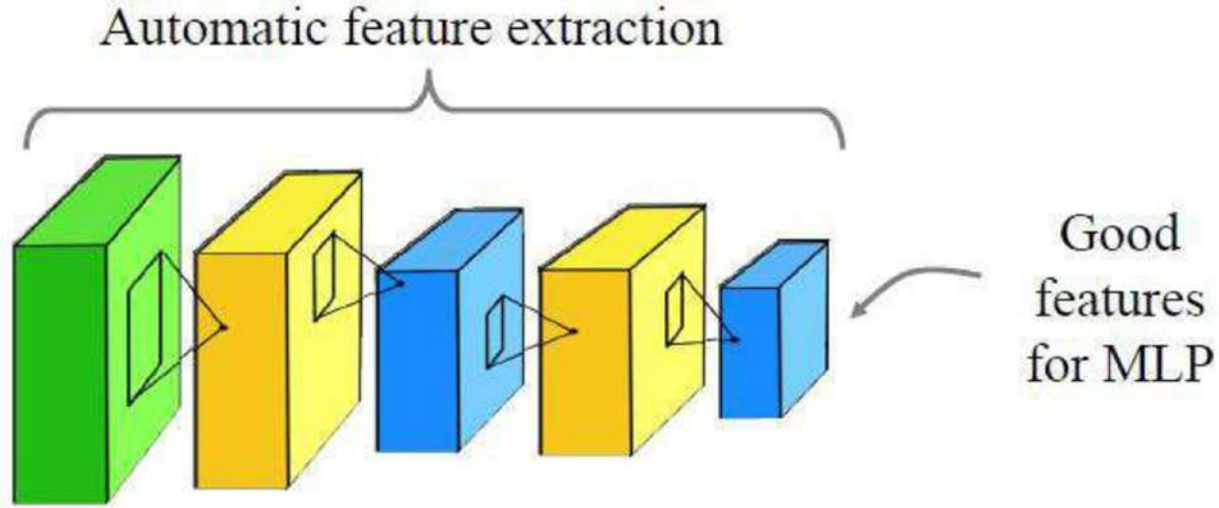
VS



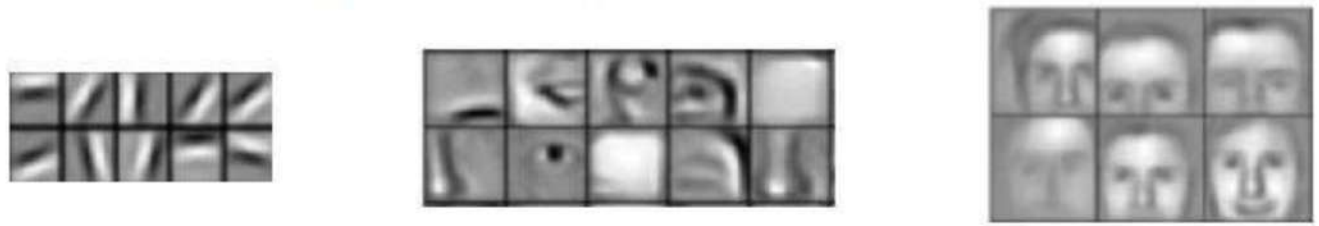
- Why do I see boxes every time , where are neurons?
- Why is that I see only a part of image being taken ?
- Why Convolution, pooling , flattening, dense layer?

Learning deep representations

Neurons of deep convolutional layers learn complex representations that can be used as features for classification with MLP.

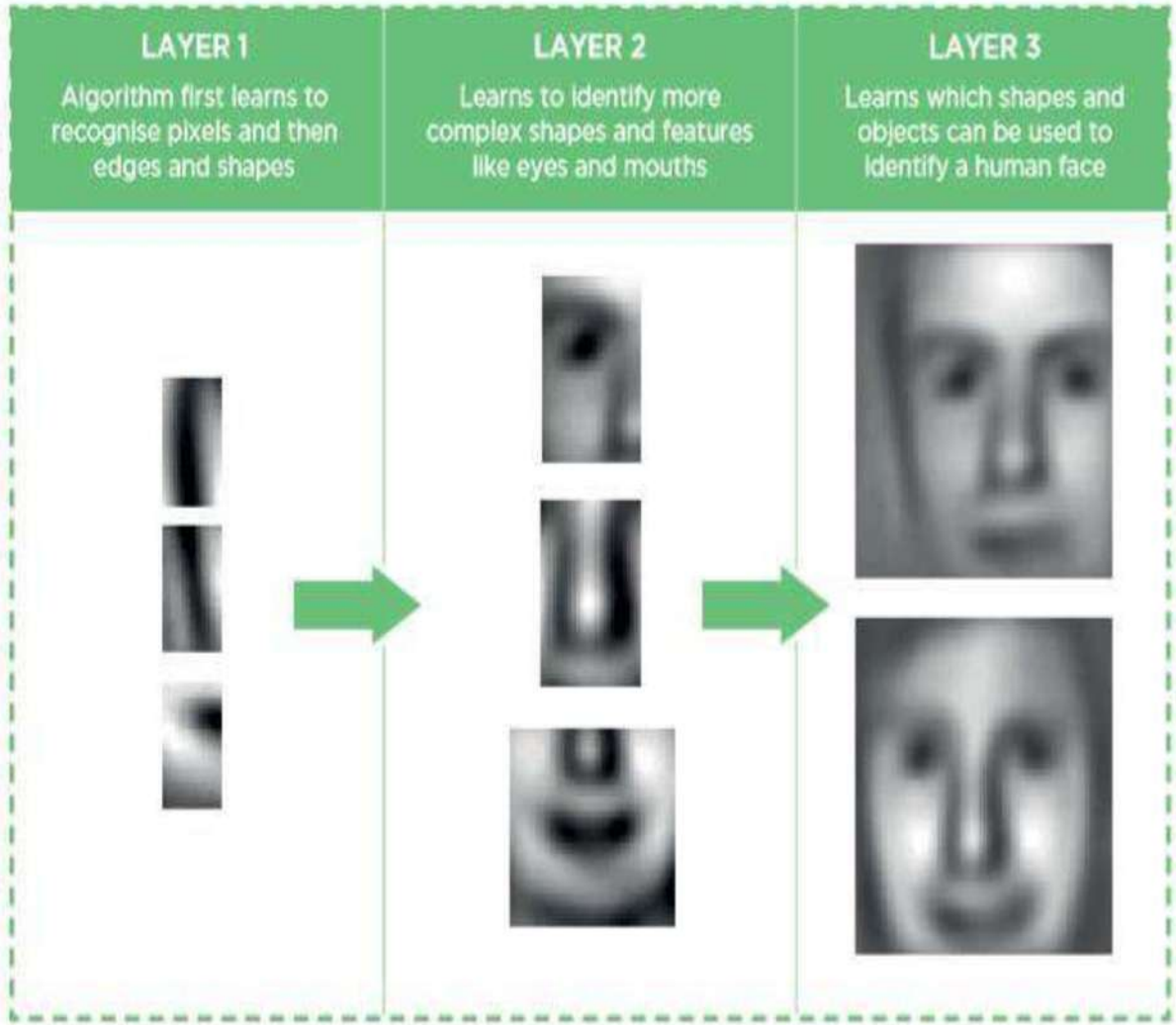


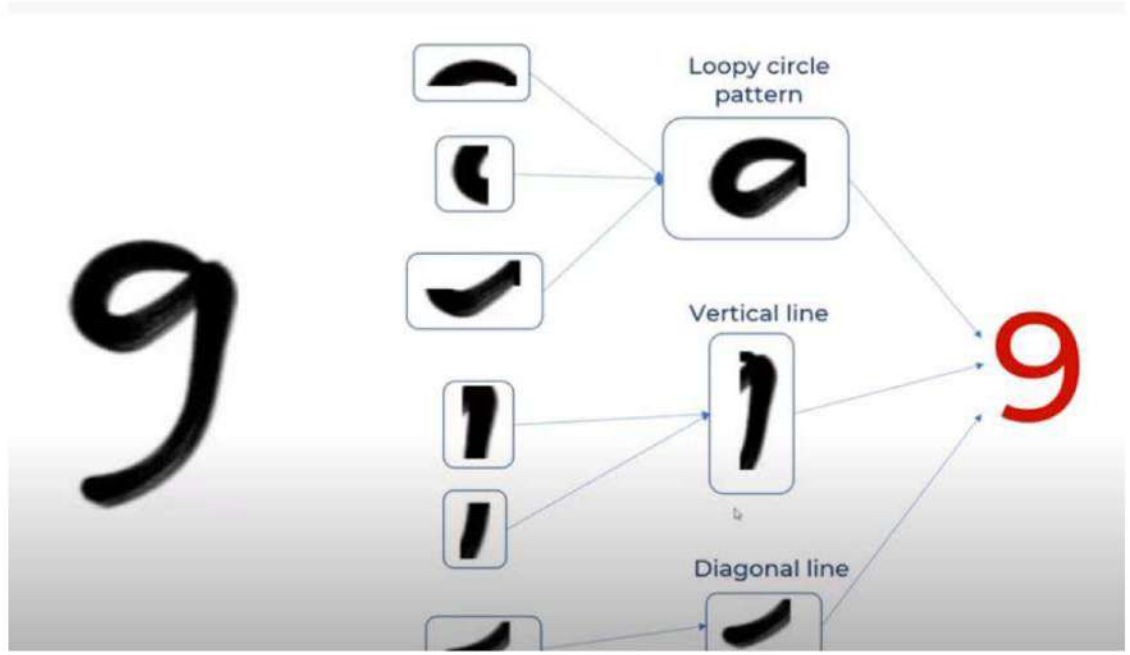
Inputs that provide highest activations:



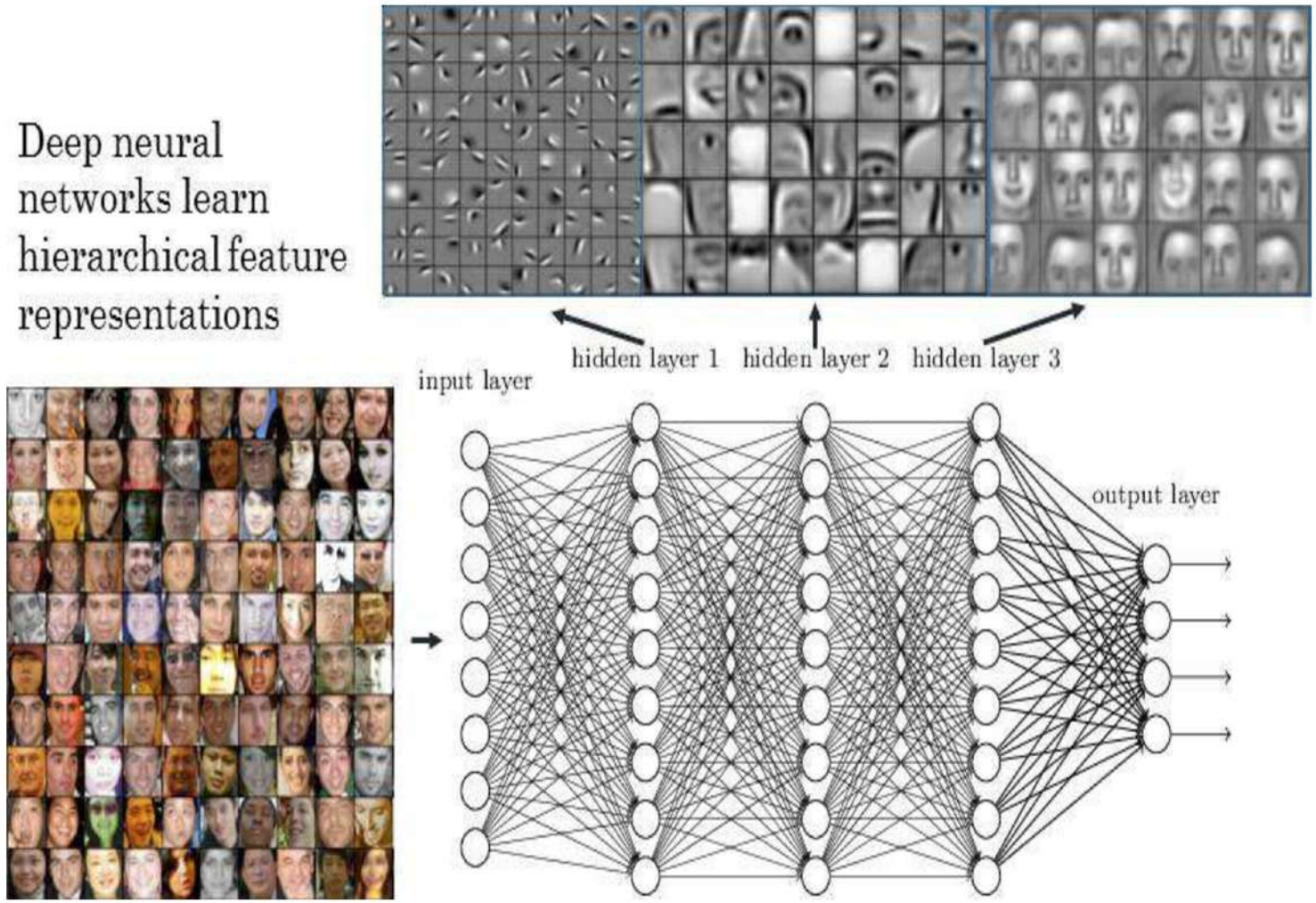
http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf

What are layers and what happens inside?

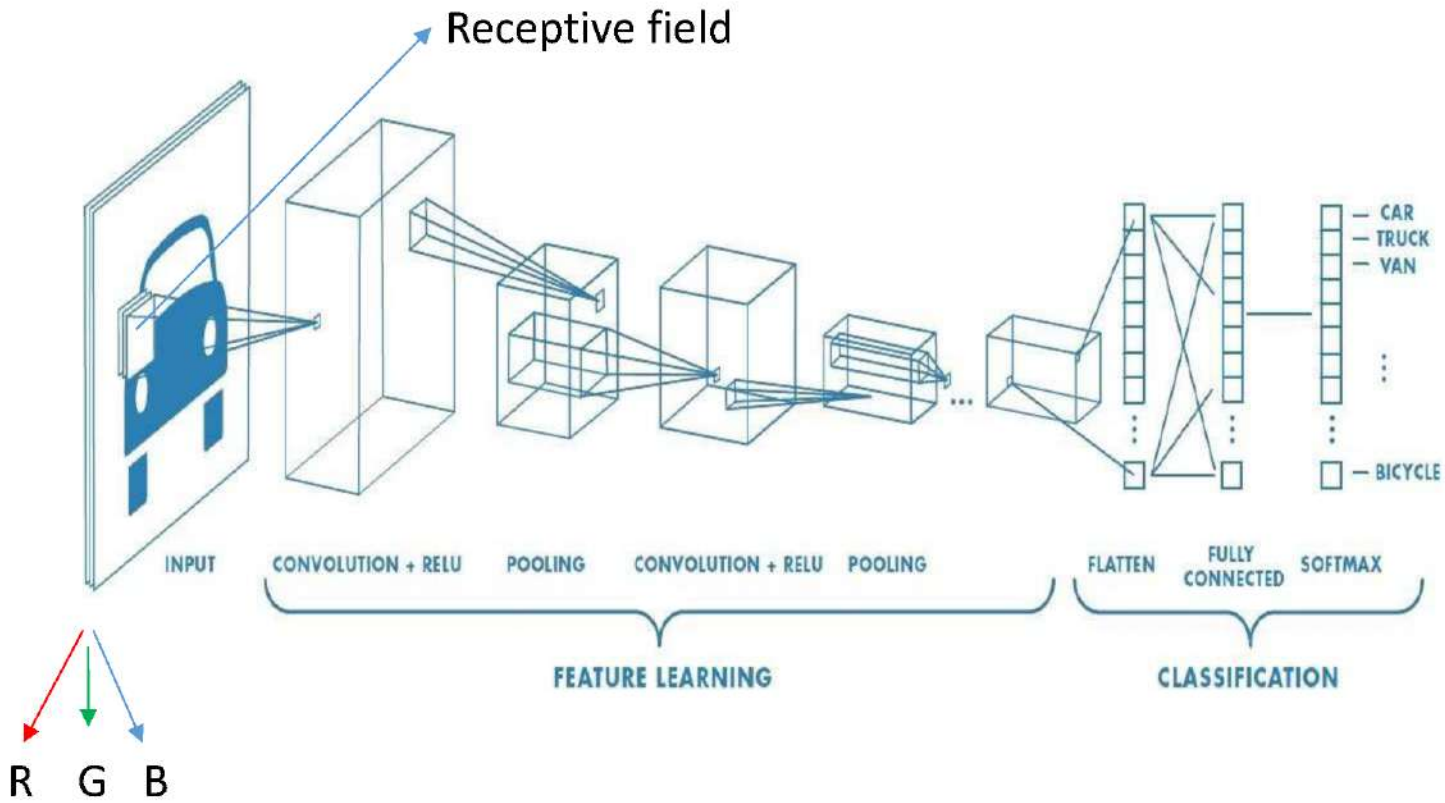



Loopy circle pattern
Vertical line
Diagonal line
9

What are layers and what happens inside?

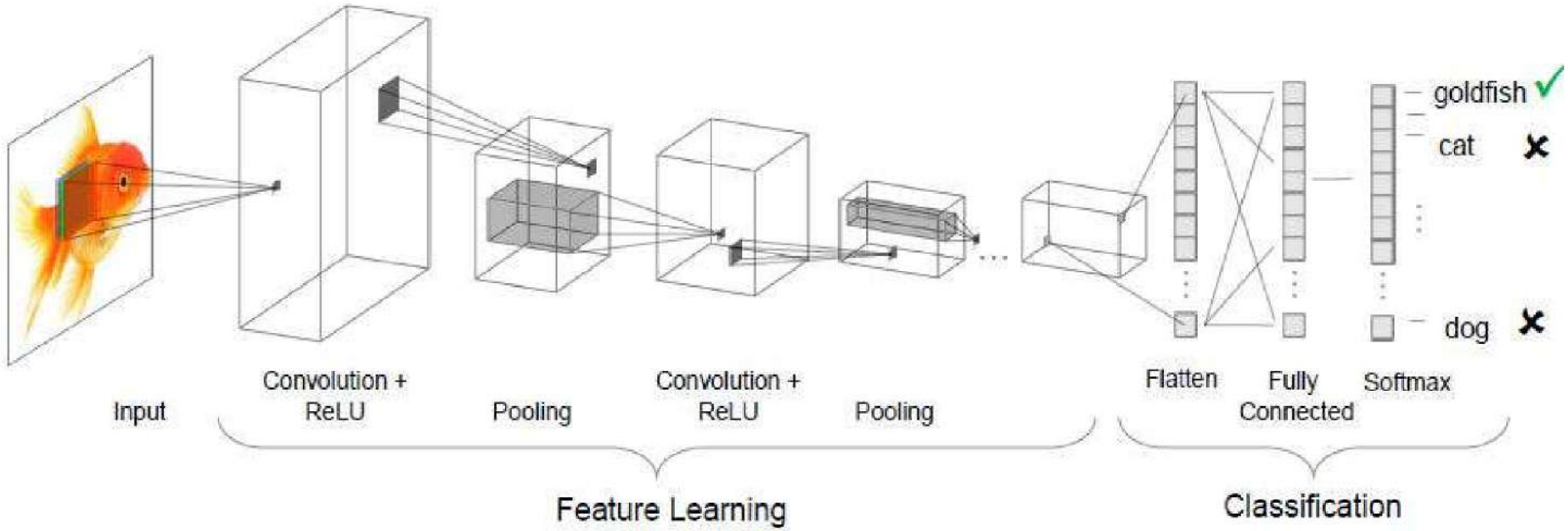


CNN –simplified



- Why do I see boxes every time , where are neurons?
- Why is that I see only a part of image being taken
- Why Convolution, pooling , flattening, dense layer

Convolutional Neural Network (CNN)/ Deep Neural Network (DNN)



- The process of building a Convolutional Neural Network always involves four major steps.
- **Step - 1 : Convolution**
- **Step - 2 : Pooling**
- **Step - 3 : Flattening**
- **Step - 4 : Full connection**

Step1: Image data

- Great training datasets are CIFAR and CoCo. We'll use CIFAR.

255	144	...	120
57	100	...	32
...	...	...	...
251	41	...	23
57	209	...	118
27	59	...	19
...	...	...	...
4	203	...	69
35	170	...	199
17	11	...	97
...	...	...	...
145	165	...	210

Step 2: Convolution

- Convolution is an element-wise multiplication of two matrices (sub-matrix) followed by a sum

Input Image

1	0	0	0	1
0	1	0	1	0
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1

5 (rows) × 5 (columns)

Kernel / Filter

1	-1	-1
-1	1	-1
-1	-1	1

3 × 3

Feature Map

3	-3	-2
-3	1	-3
-2	-3	3

3 × 3

$$\text{output size} = [\text{input size} - \text{kernel size}\ ] + 1$$

$$= [5 - 3] + 1$$

1	0	0	0	1
0	1	0	1	0
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1

1	-1	-1
-1	1	-1
-1	-1	1

Stride = 1

3	-3	-2
-3	1	-3
-2	-3	3

1	0	0	0	1
0	1	0	1	0
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1

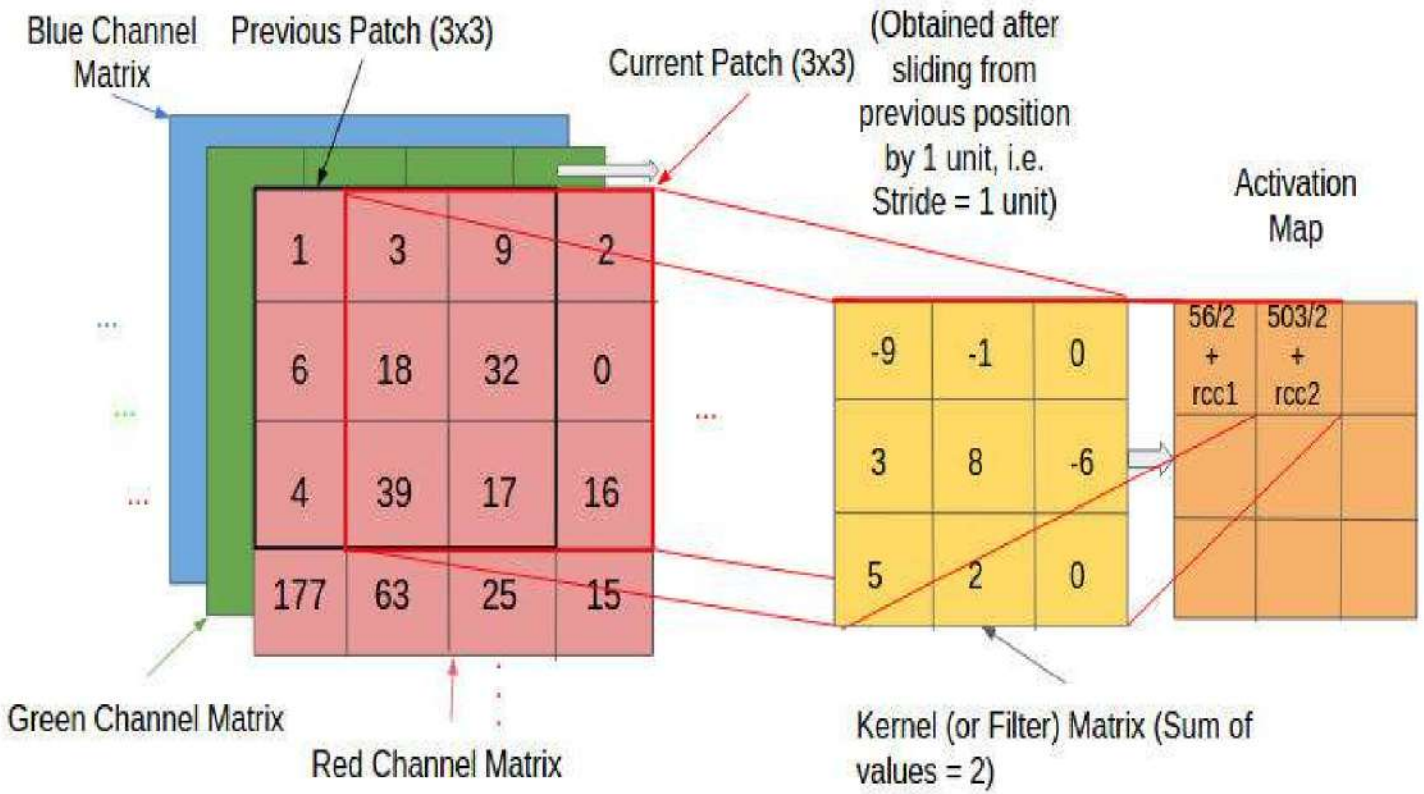
1	-1	-1
-1	1	-1
-1	-1	1

Stride = 2

3	-2
-2	3

$$\text{Output size(without padding)} = \frac{[\text{input size} - \text{kernel size}]}{\text{stride}} + 1$$

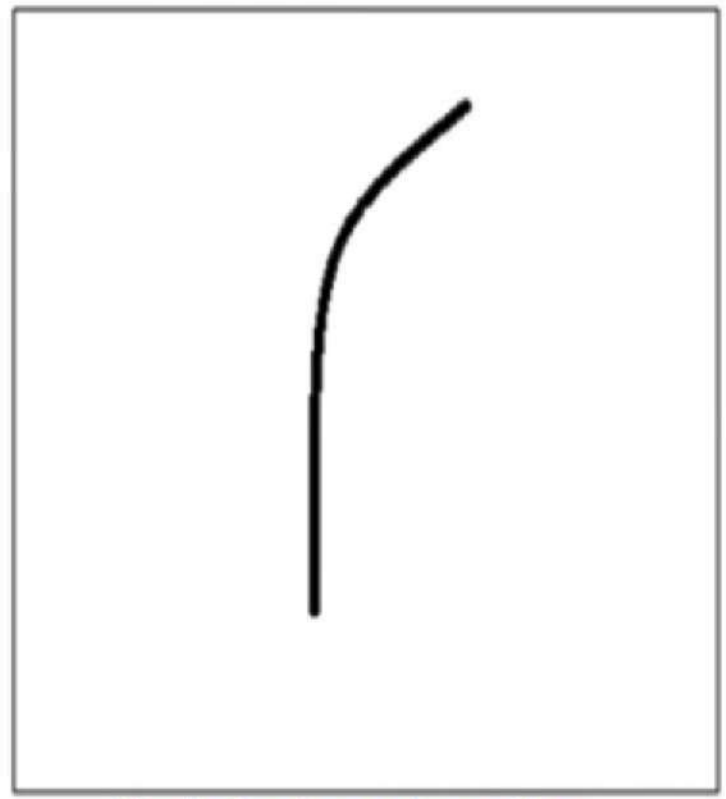
Step 2 : Convolution



Step 2 : Convolution

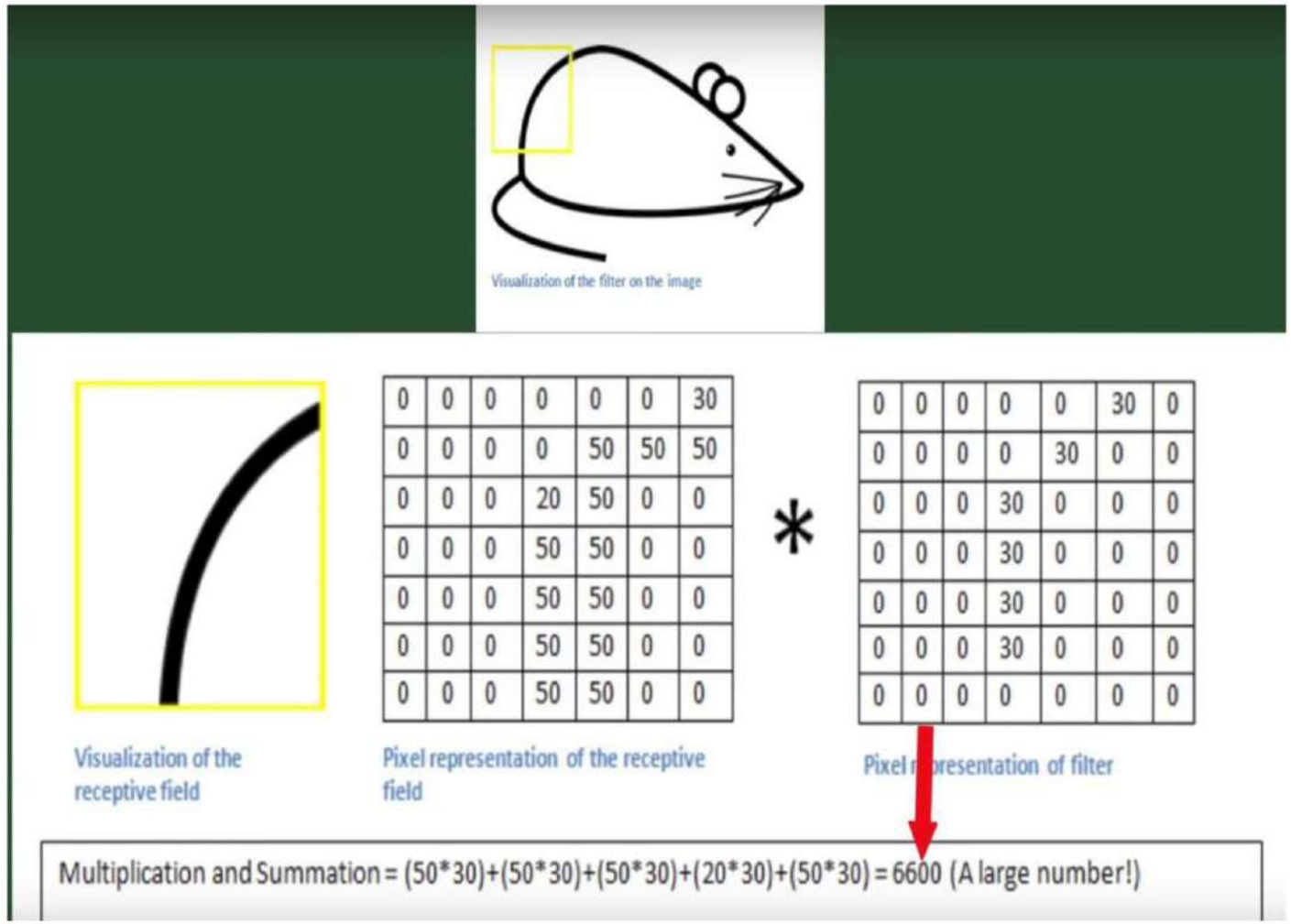
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

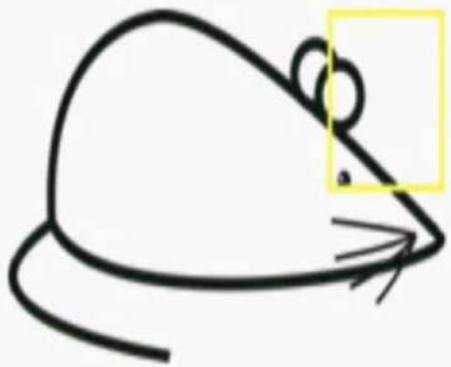


Visualization of a curve detector filter

Step 2 : Convolution



Step 2 : Convolution



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

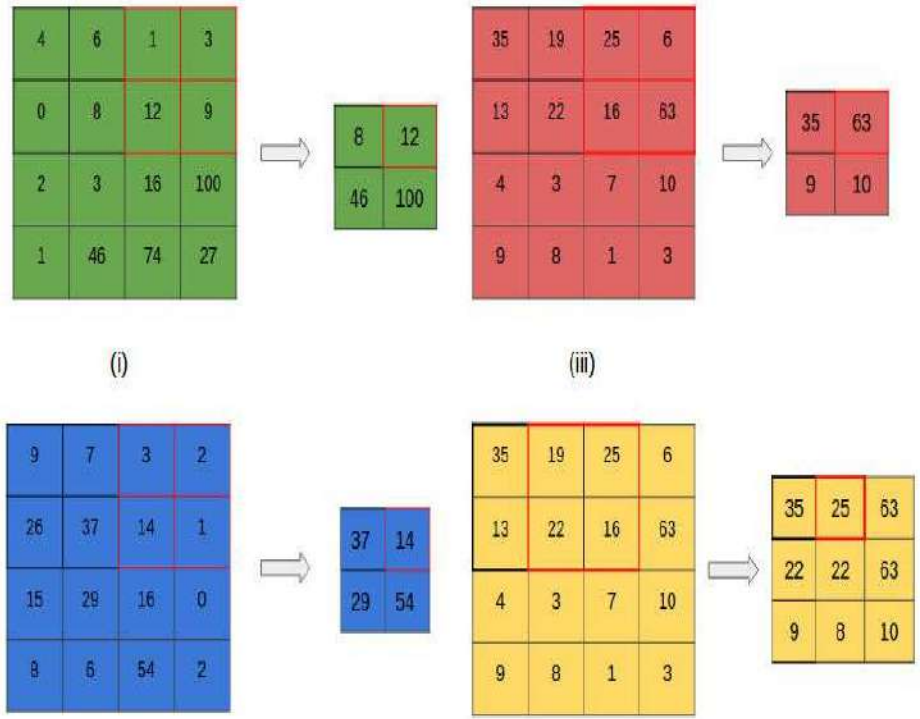
*

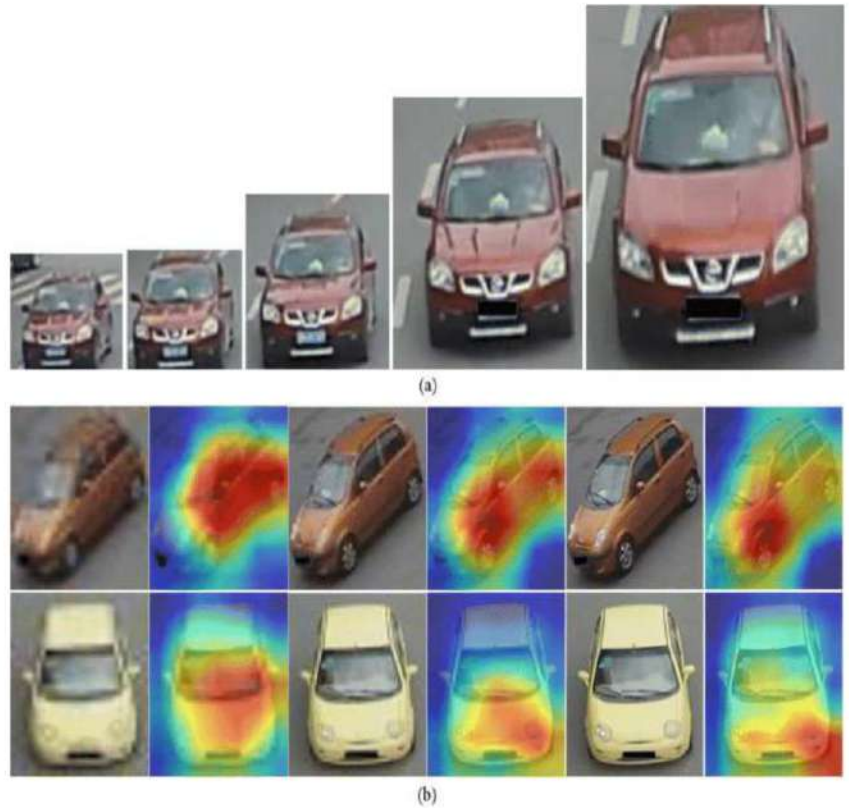
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

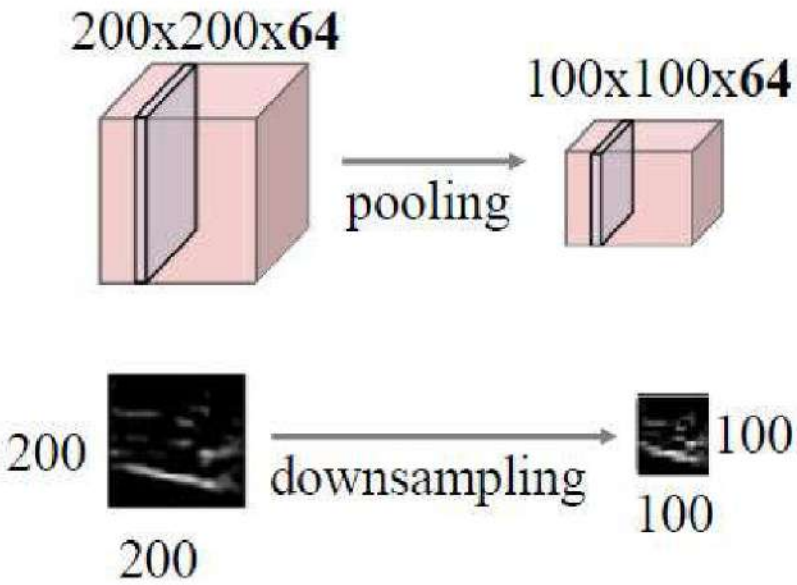
Step3 : Pooling

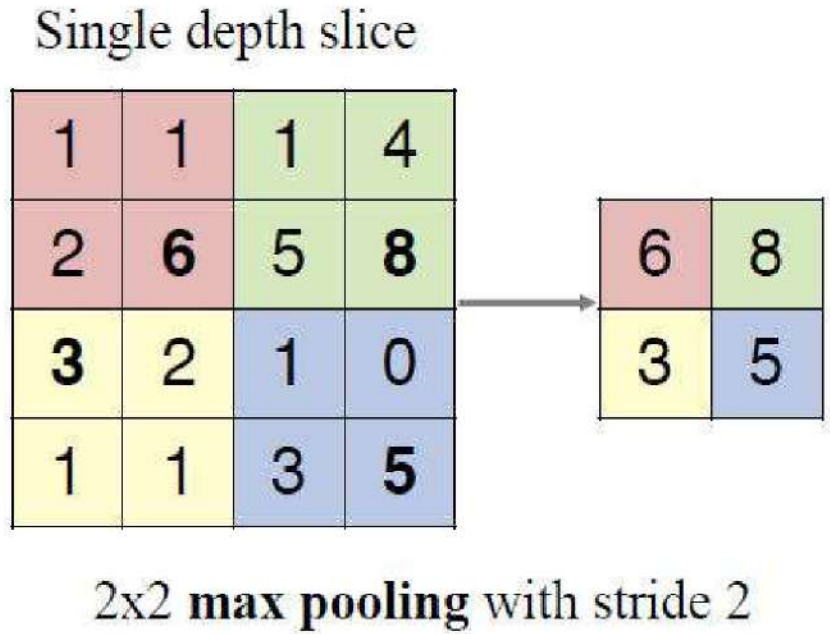




Pooling layer will help!

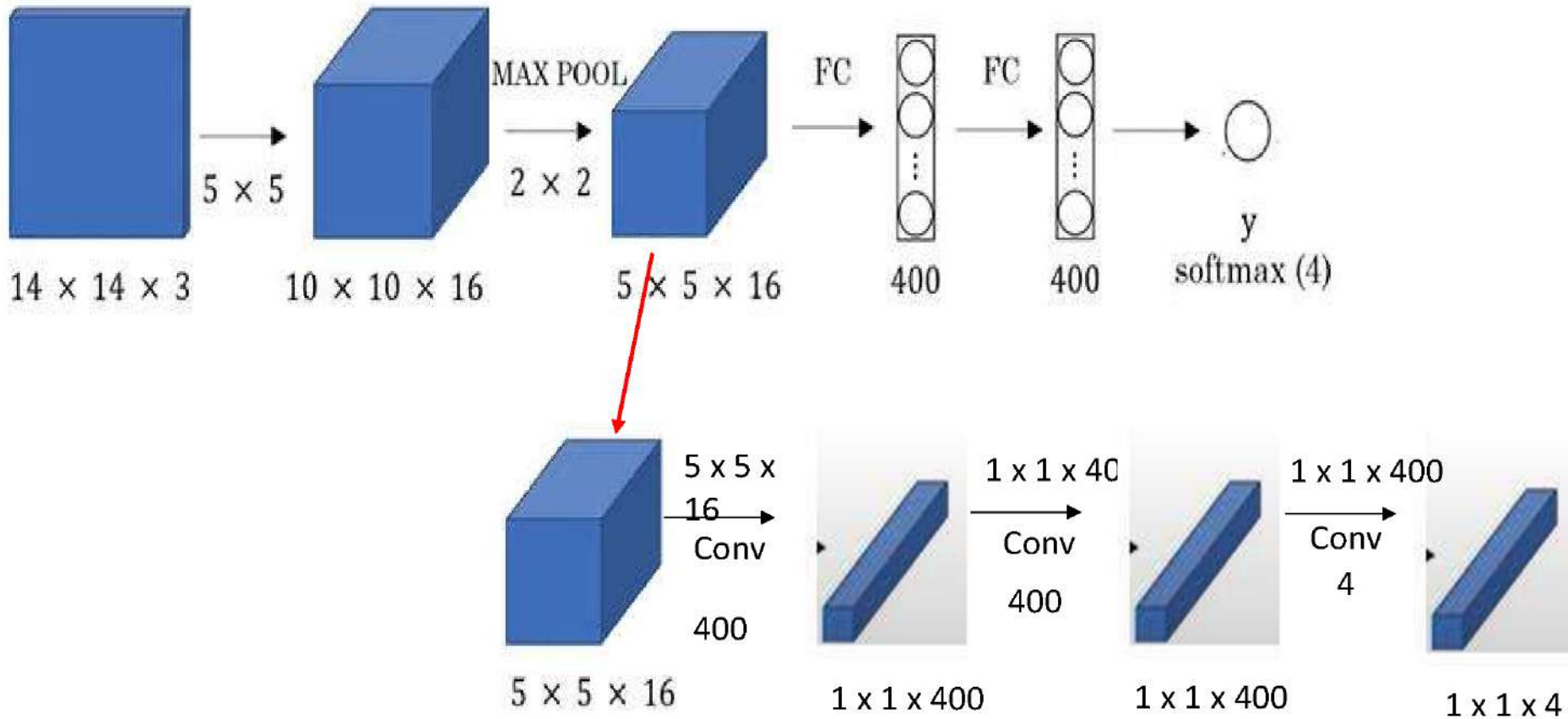
This layer works like a convolutional layer but doesn't have kernel, instead it calculates **maximum** or **average** of input patch values.





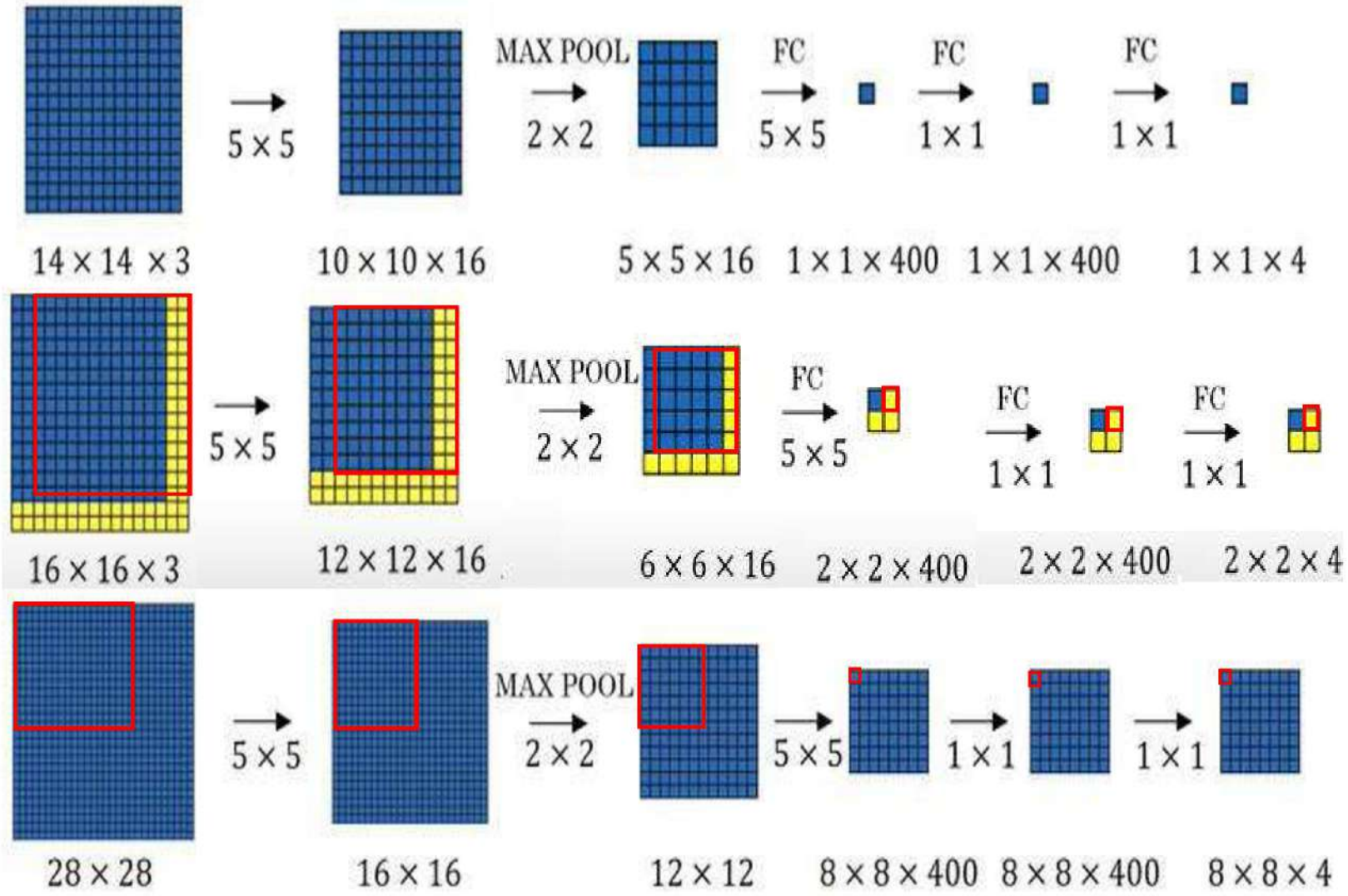
2x2 **max pooling** with stride 2

Turning FC Layer into Convolutional Layer



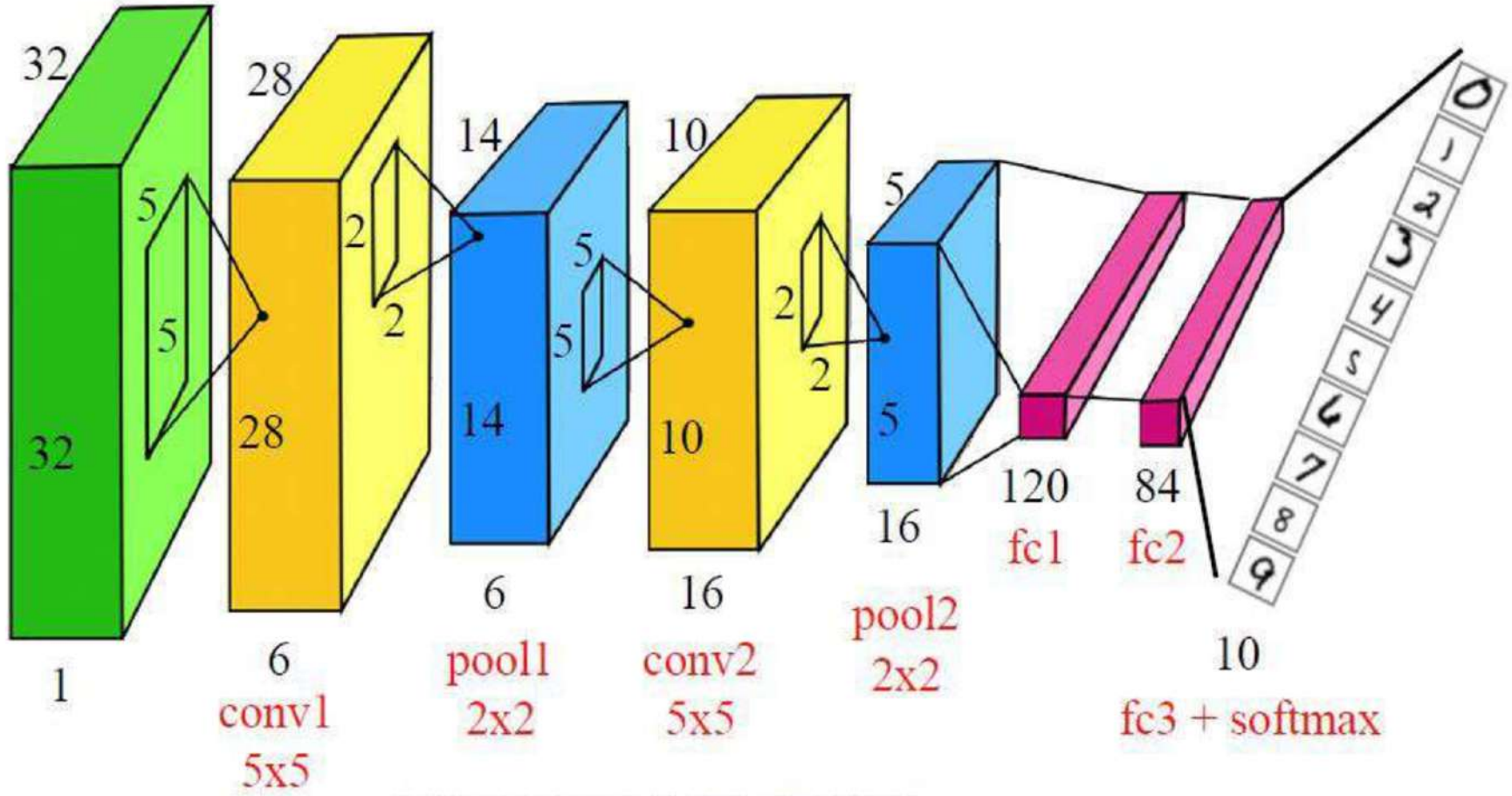
https://towardsdatascience.com/understanding-and-calculating-the-number-of-parameters-in-convolution-neural-networks-cnns-fc88790d530d

Convolution Implementation of Sliding Windows



Putting it all together into a simple CNN

LeNet-5 architecture (1998) for handwritten digits recognition on MNIST dataset:



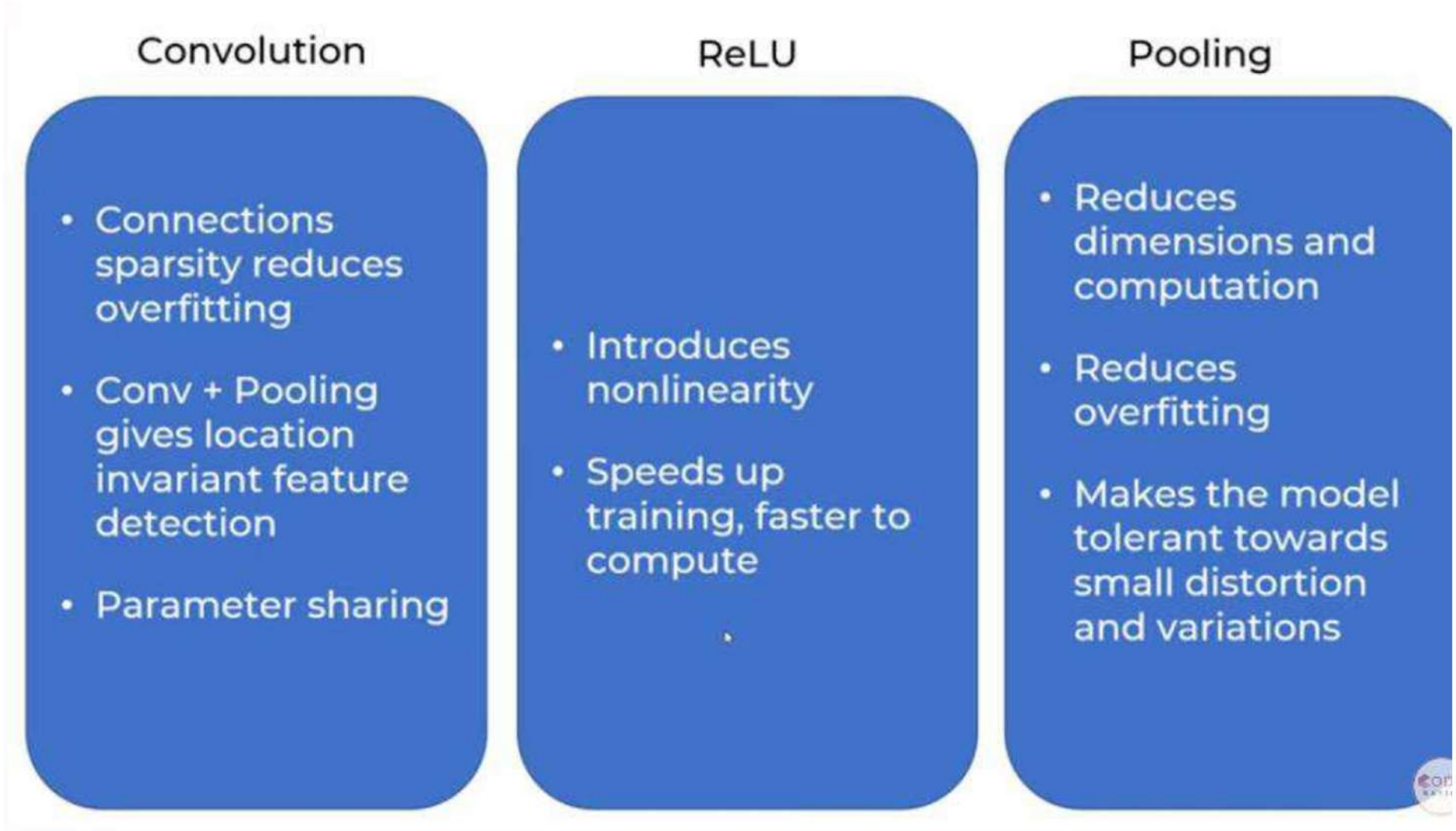
http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

Benefits of pooling

Reduces dimensions & computation

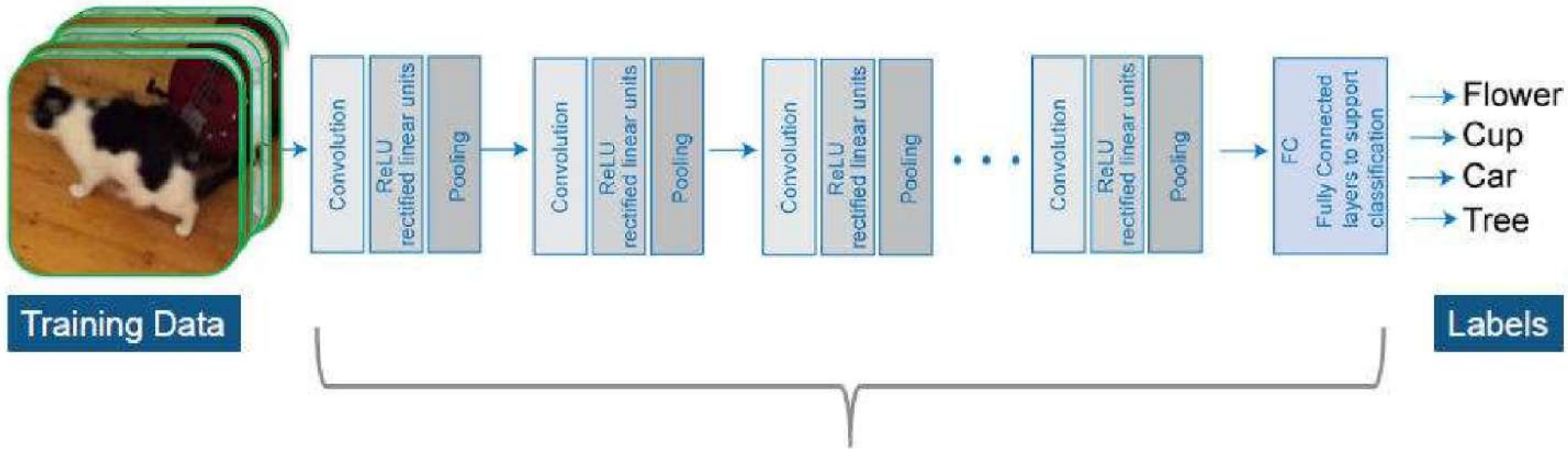
Reduce overfitting as there are less parameters

Model is tolerant towards variations, distortions


Convolution
• Connections sparsity reduces overfitting
• Conv + Pooling gives location invariant feature detection
• Parameter sharing
ReLU
• Introduces nonlinearity
• Speeds up training, faster to compute
Pooling
• Reduces dimensions and computation
• Reduces overfitting
• Makes the model tolerant towards small distortion and variations

What Happens During Training?

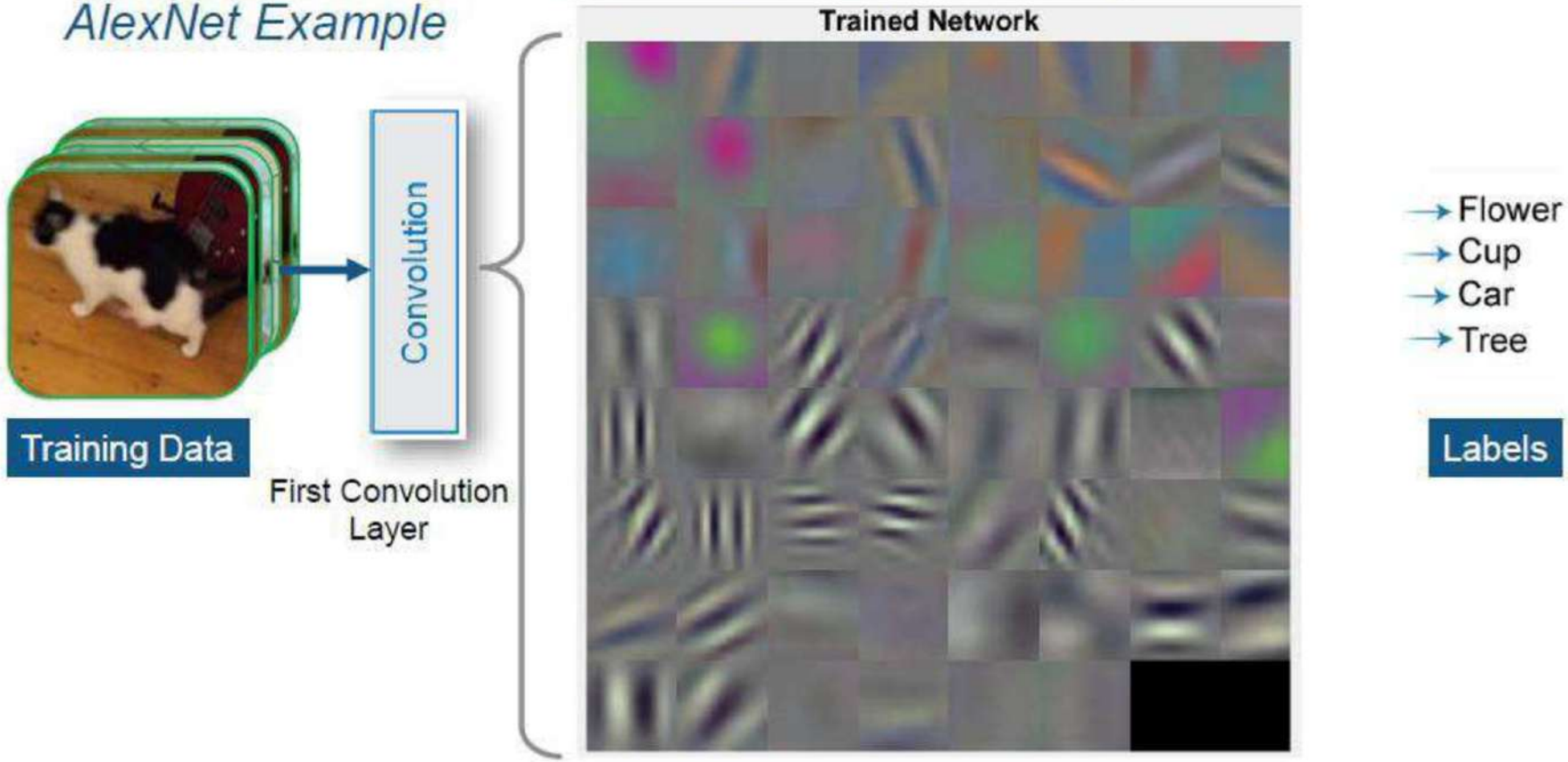
AlexNet Example



Layer weights are learned during training

What Network Learns During Training

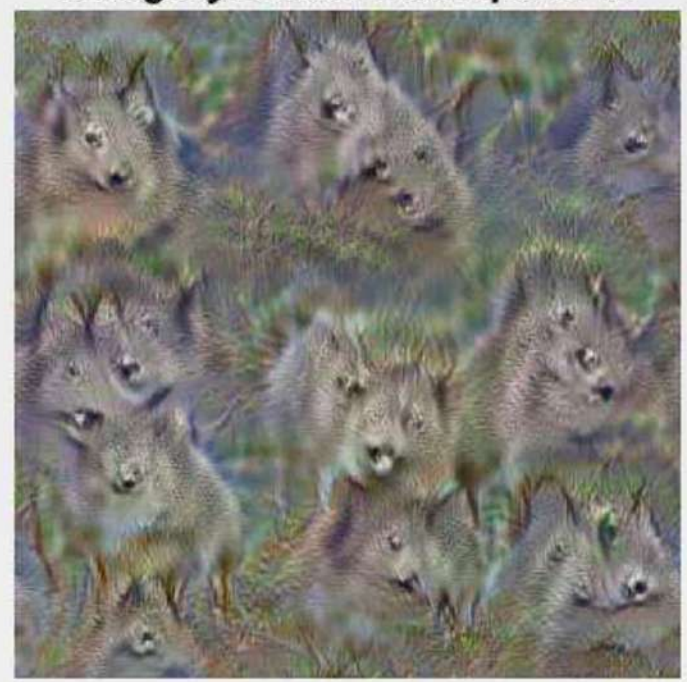
AlexNet Example



Visualize Features Learned During Training

AlexNet Example

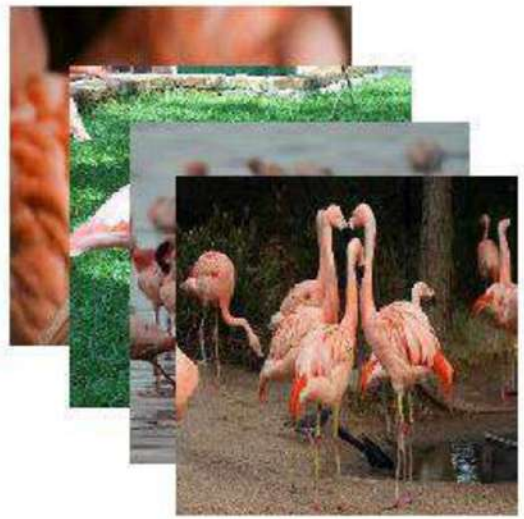
Sample Training Data



Features Learned by Network

Visualize Features Learned During Training

AlexNet Example



Sample Training Data

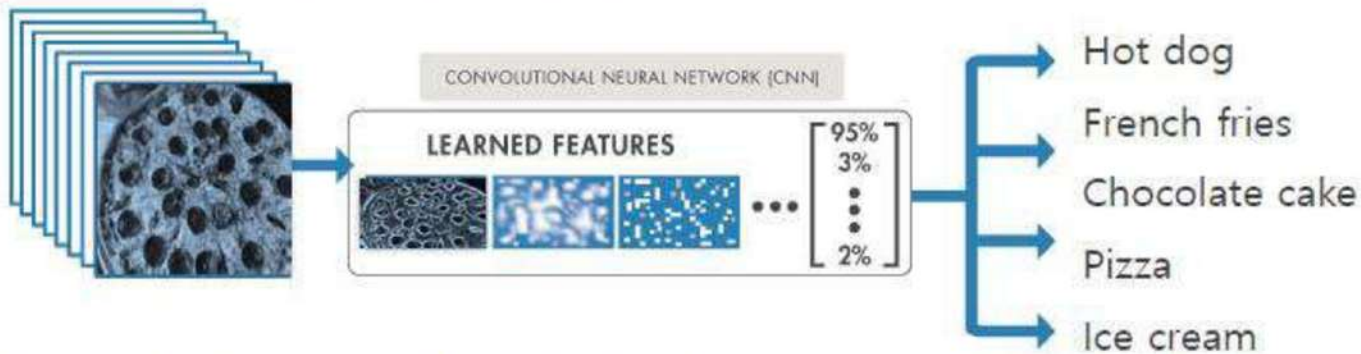
Features Learned by Network

- https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/

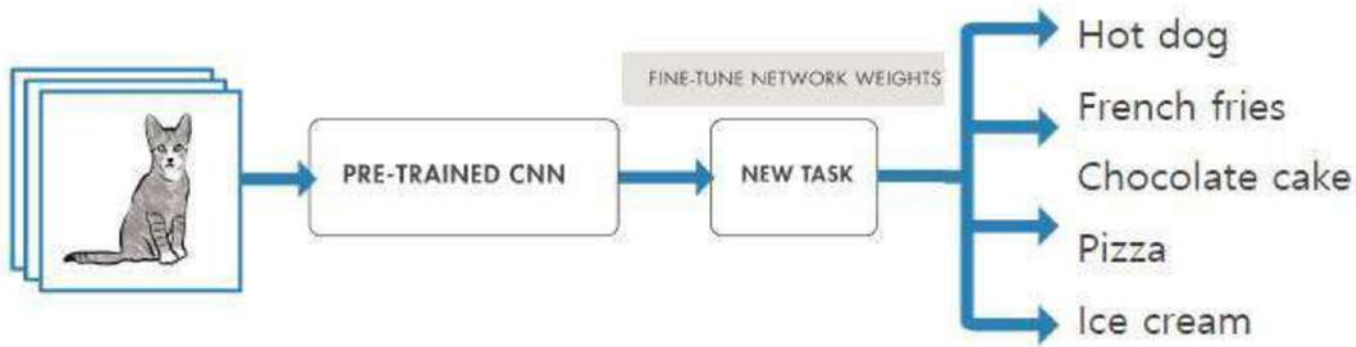
- https://towardsdatascience.com/convolution-neural-network-for-image-processing-using-keras-dc3429056306

Two Approaches for Deep Learning

1. Train a Deep Neural Network from Scratch



2. Fine-tune a pre-trained model (transfer learning)

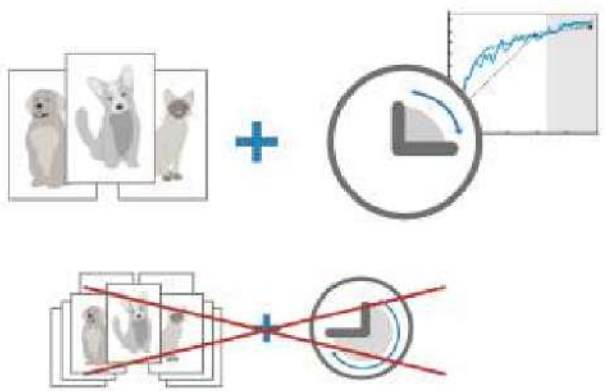


Transfer Learning

- **Transfer learning** is the process of taking a pre-trained model (the weights and parameters of a network that has been trained on a large dataset by somebody else) and “fine-tuning” the model with your own dataset.
- The idea is that this pre-trained model will act as a feature extractor.
- You will remove the last layer of the network and replace it with your own classifier (depending on what your problem space is).
- You then freeze the weights of all the other layers and train the network normally (Freezing the layers means not changing the weights during gradient descent/optimization).

Why Perform Transfer Learning ?

- Leverage best network types from top researchers
- Reference models (such as AlexNet, VGG-16, VGG-19) are great feature representations
- Requires less data and training time

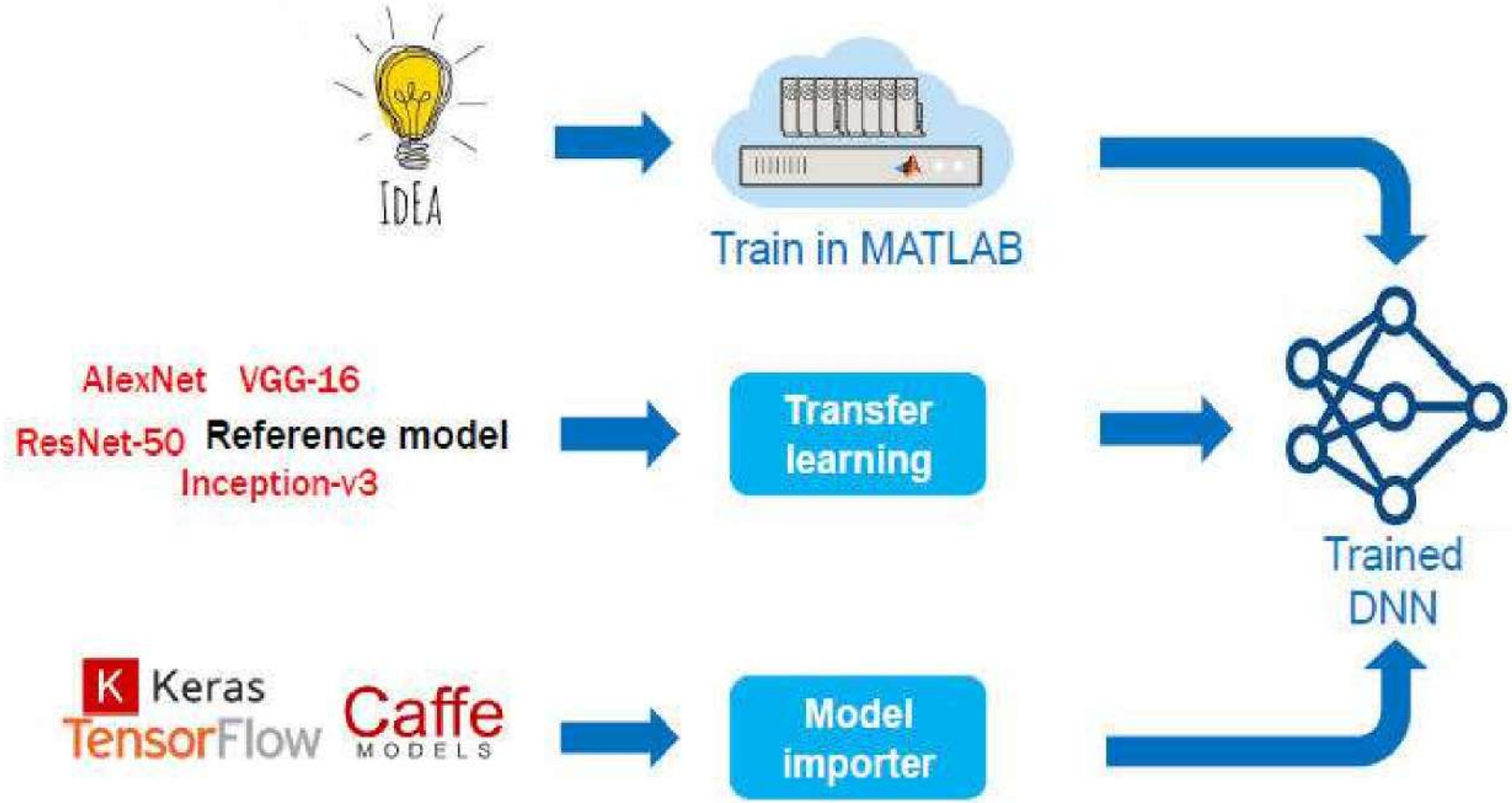


Pretrained Models in MATLAB

- AlexNet `net = alexnet;`
- VGG-16 `net = vgg16;`
- VGG-19 `net = vgg19;`
- GoogLeNet `net = googlenet;`
- Inceptionv3 `net = inceptionv3;`
- Resnet50 `net = resnet50;`
- Resnet101 `net = resnet101;`
- InceptionResnetv2 `net = inceptionresnetv2`
- Squeezenet `net = squeezenet;`

Download from within MATLAB

How do I obtain trained DNN



Import the Latest Models

* single line of code to access model

Pretrained Models*

- `alexnet`
- `vgg16`
- `vgg19`
- `googlenet`
- `inception-v3`
- `resnet-18`
- `resnet-50`
- `resnet-101`
- `inception-resnet-v2`
- `squeezenet`
- `densenet-201`

```
net = alexnet;
net = vgg16;
net = vgg19;
net = googlenet;
net = inceptionv3;
net = resnet50;
net = resnet101;
net = inceptionresnetv2;
net = squeezenet;
net = densenet201;
```

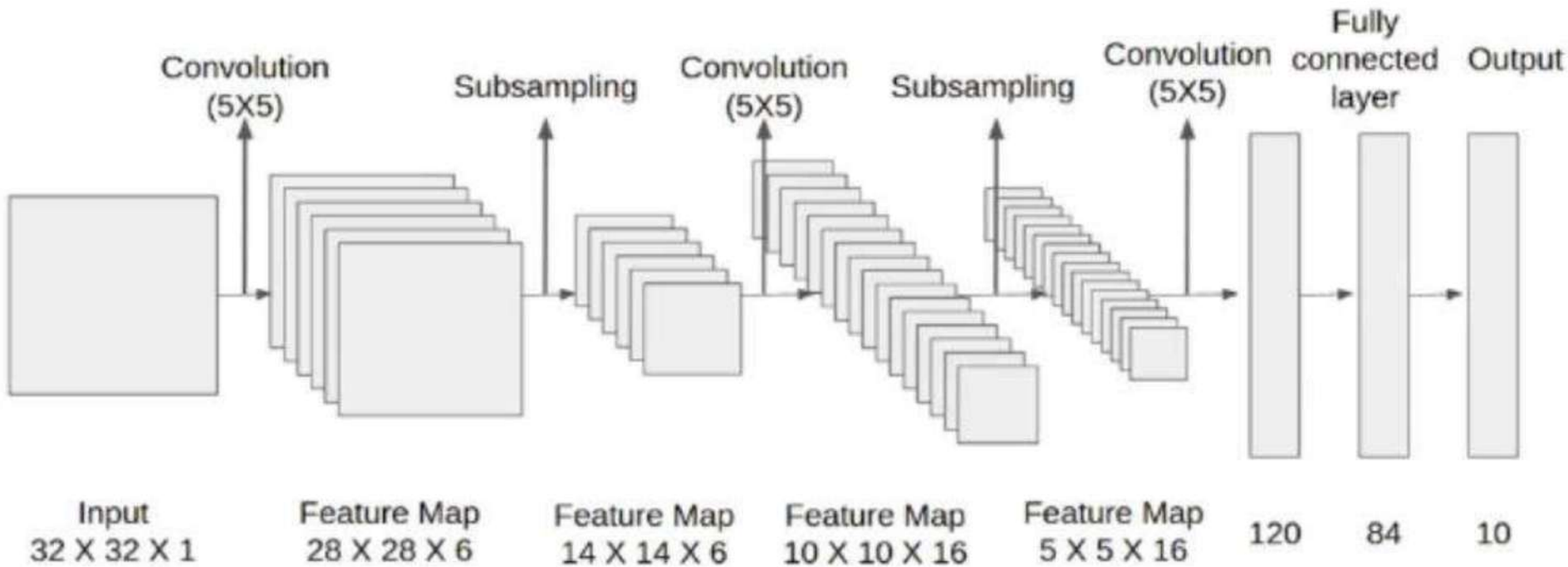
Lenet-5

- Lenet-5 is one of the earliest pre-trained models proposed by Yann LeCun and others in the year 1998
- used this architecture for recognizing the handwritten and machine-printed characters.
- The main reason behind the popularity of this model was its simple and straightforward architecture.
- It is a multi-layer convolution neural network for image classification.

Lenet-5

- The network has 5 layers with learnable parameters
- It has three sets of convolution layers with a combination of average pooling. Tanh is the activation function used in convolution layers.
- After the convolution and average pooling layers, we have two fully connected layers.
- At last, a Softmax classifier which classifies the images into respective class.

Here is the final architecture of the Lenet-5 model.



Layer	# filters / neurons	Filter size	Stride	Size of feature map	Activation function
Input	-	-	-	32 X 32 X 1	
Conv 1	6	5 * 5	1	28 X 28 X 6	tanh
Avg. pooling 1		2 * 2	2	14 X 14 X 6	
Conv 2	16	5 * 5	1	10 X 10 X 16	tanh
Avg. pooling 2		2 * 2	2	5 X 5 X 16	
Conv 3	120	5 * 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully Connected 2	-	-	-	10	Softmax

Number of parameters

Alexnet

- Watch the video available in the below link
- https://www.analyticsvidhya.com/blog/2021/03/introduction-to-the-architecture-of-alexnet/

Alexnet

- proposed in 2012 by Alex Krizhevsky and his colleagues.
- won the Imagenet large-scale visual recognition challenge in 2012
- 8 layers
- five layers with a combination of max pooling followed by 3 fully connected layers
- Relu activation in each of these layers and softmax in the output layer.
- using the relu as an activation function accelerated the speed of the training process by almost six times.

Alexnet

- used the dropout layers, that prevented their model from overfitting.
- model is trained on the Imagenet dataset
- The Imagenet dataset has almost 14 million images across a thousand classes.
- The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.
- introduced padding to prevent the size of the feature maps from reducing drastically.
- The input to this model is the images of size 227X227X3(original 224X224X3).

What is the use of dropout layer?

- One of the method to avoid overfitting.
- One typical characteristic of CNNs is a Dropout layer.
- The Dropout layer is a mask that **nullifies the contribution of some neurons towards the next layer and leaves unmodified all others**.

Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
Input	-	-	-	-	227 x 227 x 3	-
Conv 1	96	11 x 11	4	-	55 x 55 x 96	ReLU
Max Pool 1	-	3 x 3	2	-	27 x 27 x 96	-
Conv 2	256	5 x 5	1	2	27 x 27 x 256	ReLU
Max Pool 2	-	3 x 3	2	-	13 x 13 x 256	-
Conv 3	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 4	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 5	256	3 x 3	1	1	13 x 13 x 256	ReLU
Max Pool 3	-	3 x 3	2	-	6 x 6 x 256	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-

Fully Connected 1	-	-	-	-	4096	ReLU
Dropout 2	rate = 0.5	-	-	-	4096	-
Fully Connected 2	-	-	-	-	4096	ReLU
Fully Connected 3	-	-	-	-	1000	Softmax

VGG16

- VGG- Network is a convolutional neural network model proposed by K. Simonyan and A. Zisserman in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition” .
- This architecture achieved top-5 test accuracy of 92.7% in ImageNet, which has over 14 million images belonging to 1000 classes.
- It is one of the famous architectures in the deep learning field.
- Replacing large kernel-sized filters with 11 and 5 in the first and second layer respectively showed the improvement over AlexNet architecture, with multiple 3×3 kernel-sized filters one after another.
- It was trained for weeks and was using NVIDIA Titan Black GPU’s.


Input
Conv 1-1
Conv 1-2
Pooing
Conv 2-1
Conv 2-2
Pooing
Conv 3-1
Conv 3-2
Conv 3-3
Pooing
Conv 4-1
Conv 4-2
Conv 4-3
Pooing
Conv 5-1
Conv 5-2
Conv 5-3
Pooing
Dense
Dense
Dense
Output

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

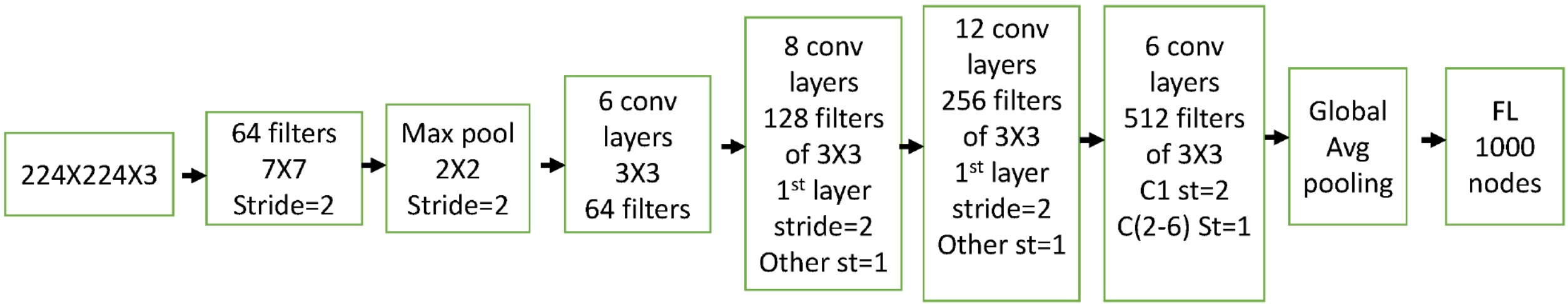
```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
```

```
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
```

Resnet 34

- Resnet34 is a state-of-the-art image classification model, structured as a 34 layer convolutional neural network and defined in "**Deep Residual Learning for Image Recognition**".
- Restnet34 is pre-trained on the ImageNet dataset which contains 100,000+ images across 200 different classes.

Resnet 34

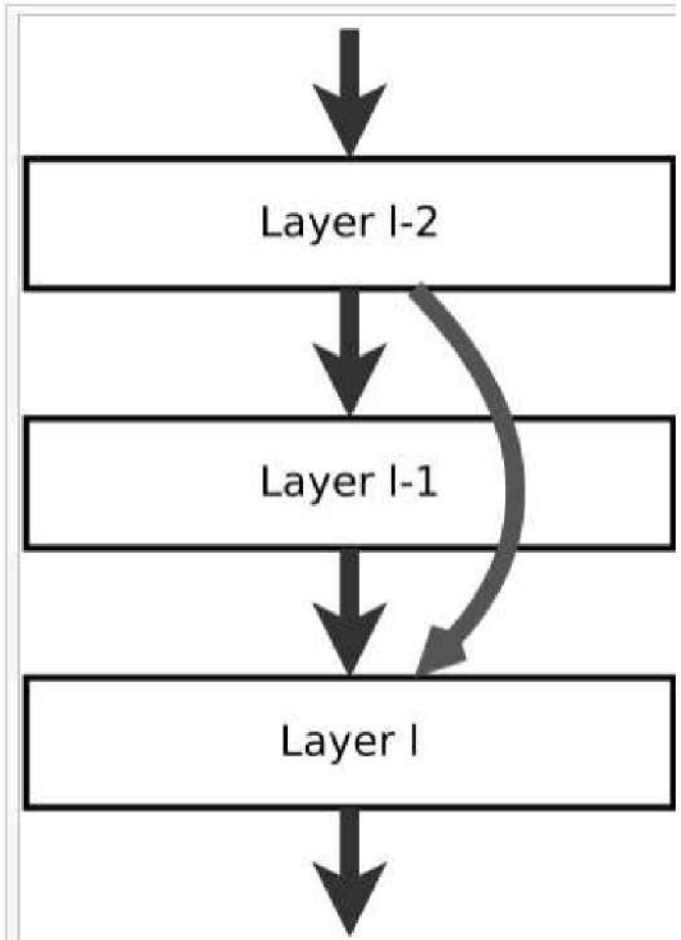


Approximately 21.3 parameters

- Previous CNN architectures were not able to scale to a large number of layers, which resulted in limited performance.
- when adding more layers, researchers faced the “vanishing gradient” problem.
- Neural networks are trained through a backpropagation process that relies on gradient descent, shifting down the loss function and finding the weights that minimize it.
- If there are too many layers, repeated multiplications will eventually reduce the gradient until it “disappears”, and performance saturates or deteriorates with each layer added.

- ResNet provides an innovative solution to the vanishing gradient problem, known as “skip connections”.
- ResNet stacks multiple identity mappings, skips those layers, and reuses the activations of the previous layer.
- Skipping speeds up initial training by compressing the network into fewer layers.
- Then, when the network is retrained, all layers are expanded and the remaining parts of the network—known as the residual parts—are allowed to explore more of the feature space of the input image.

- Most ResNet models skip two or three layers at a time
- More advanced ResNet architectures, known as HighwayNets, can learn “skip weights”, which dynamically determine the number of layers to skip.
- *Skip connections* or *shortcuts* are used to jump over some layers
- Typical *ResNet* models are implemented with double- or triple- layer skips that contain nonlinearities (ReLU) and batch normalization in between.



two main reasons to add skip connections

- to avoid the problem of vanishing gradients,[5] thus leading to easier optimization of neural networks
- to mitigate the Degradation (accuracy saturation) problem, i.e. adding more layers to a suitably deep model leads to higher training error

Regularization

- Read text book (Ian GoodFellow) chapter7

- A central problem in machine learning is how to make an algorithm that will perform well not just on the training data, but also on new inputs.
- Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.”

- The main idea behind the regularization technique is **to penalize complex models, i.e.- to define a penalty function to quantify complexity of the model.** (the more complex models will have a greater penalty associated with them). Since most of the training algorithms are considered an optimization problem where the loss is minimized we add the penalty term and minimize the whole expression together.

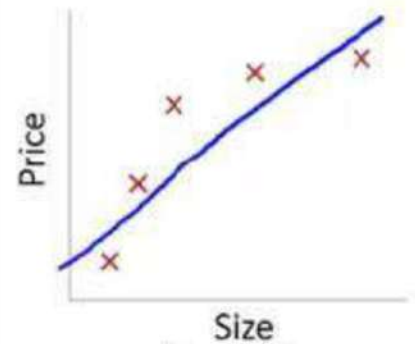
Bias-Variance

Bias-Variance Tradeoff

Bias is the amount of error introduced by approximating real-world phenomena with a simplified model.

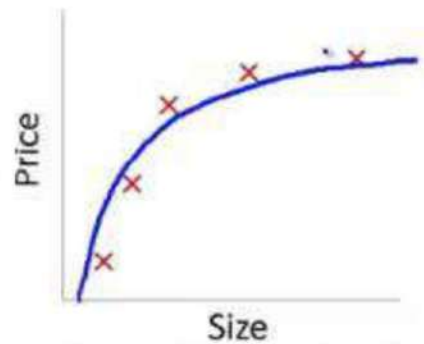
Variance is how much your model's test error changes based on variation in the training data. It reflects the model's sensitivity to the idiosyncrasies of the data set it was trained on.

As a model increases in complexity and it becomes more wiggly (**flexible**), its bias decreases (it does a good job of explaining the training data), but variance increases (it doesn't generalize as well). **Ultimately, in order to have a good model, you need one with low bias and low variance.**



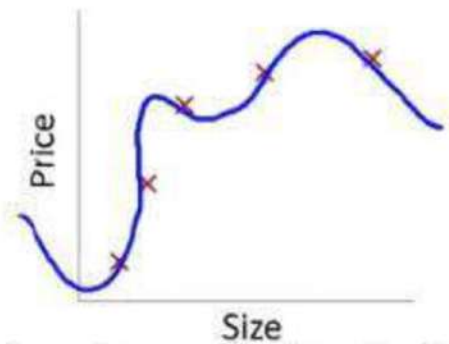
$\theta_0 + \theta_1 x$

High bias
(underfit)



$\theta_0 + \theta_1 x + \theta_2 x^2$

"Just right"



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

High variance
(overfit)

- Bias-variance tradeoff is a fundamental concept in machine learning that refers to the tradeoff between the model's ability to fit the training data (bias) and its ability to generalize to new, unseen data (variance)
- A model with high bias tends to underfit the data, while a model with high variance tends to overfit the data.
- The goal is to find the optimal balance between bias and variance to achieve the best possible predictive performance on new data.
- Techniques such as regularization, cross-validation, can help to mitigate the bias-variance tradeoff.

L2 regularization(ridge regression)

- Add a term in cost function
- Coefficient regularization

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left(h_{\theta} \left(x^{(i)} \right) - y^{(i)} \right)^2$$

- it modifies the over-fitted or under fitted models by adding the penalty equivalent to the sum of the squares of the magnitude of coefficients.

Cost function = Loss + λ x $\sum \|w\|^2$

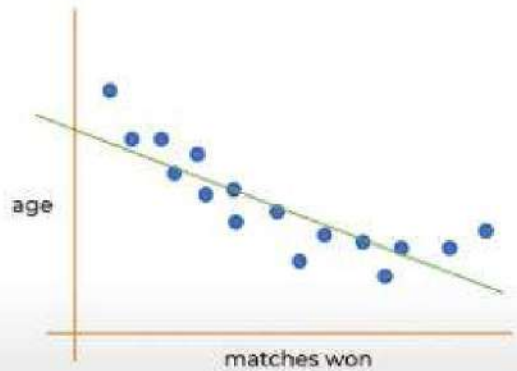
Here,

Loss = Sum of the squared residuals

λ = Penalty for the errors

W = slope of the curve/ line

underfit

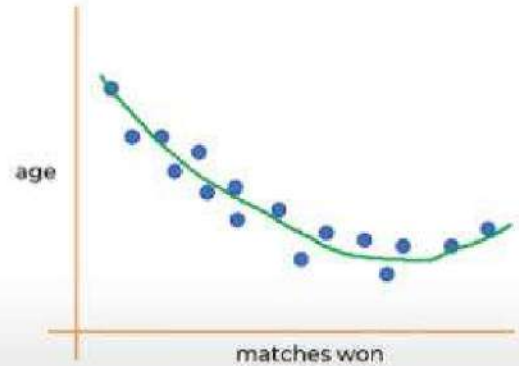


$$\text{match won} = \theta_0 + \theta_1 * age$$

overfit

$$\text{match won} = \theta_0 + \theta_1 * age \quad +\theta_2 * age^2 \quad +\theta_3 * age^3 \quad +\theta_4 * age^4$$

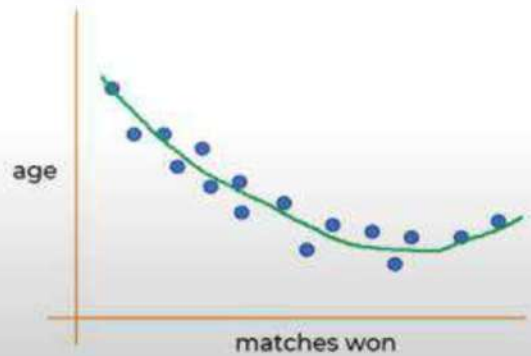
balanced fit



$$\text{match won} = \theta_0 + \theta_1 * age \quad +\theta_2 * age^2$$

$$\text{match won} = \theta_0 + \theta_1 * age + \theta_2 * age^2 + \theta_3 * age^3 + \theta_4 * age^4$$

Try to make θ_3 and θ_4 almost close to zero



$$\text{match won} = \theta_0 + \theta_1 * age + \theta_2 * age^2$$

- MSE should be minimum in each iteration

Mean Squared Error

$$ms\varepsilon = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - h_\theta(x_i)\right)^2$$

$$h_\theta(x_i) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3$$

L2 Regularization

$$mse = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - h_\theta(x_i)\right)^2 + \lambda \sum_{i=1}^{n} \theta_i^2$$

Adding penalty term

$$h_\theta(x_i) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3$$

L1 Regularization

$$mse = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - h_\theta(x_i)\right)^2 + \lambda \sum_{i=1}^{n} |\theta_i|$$

Adding penalty term

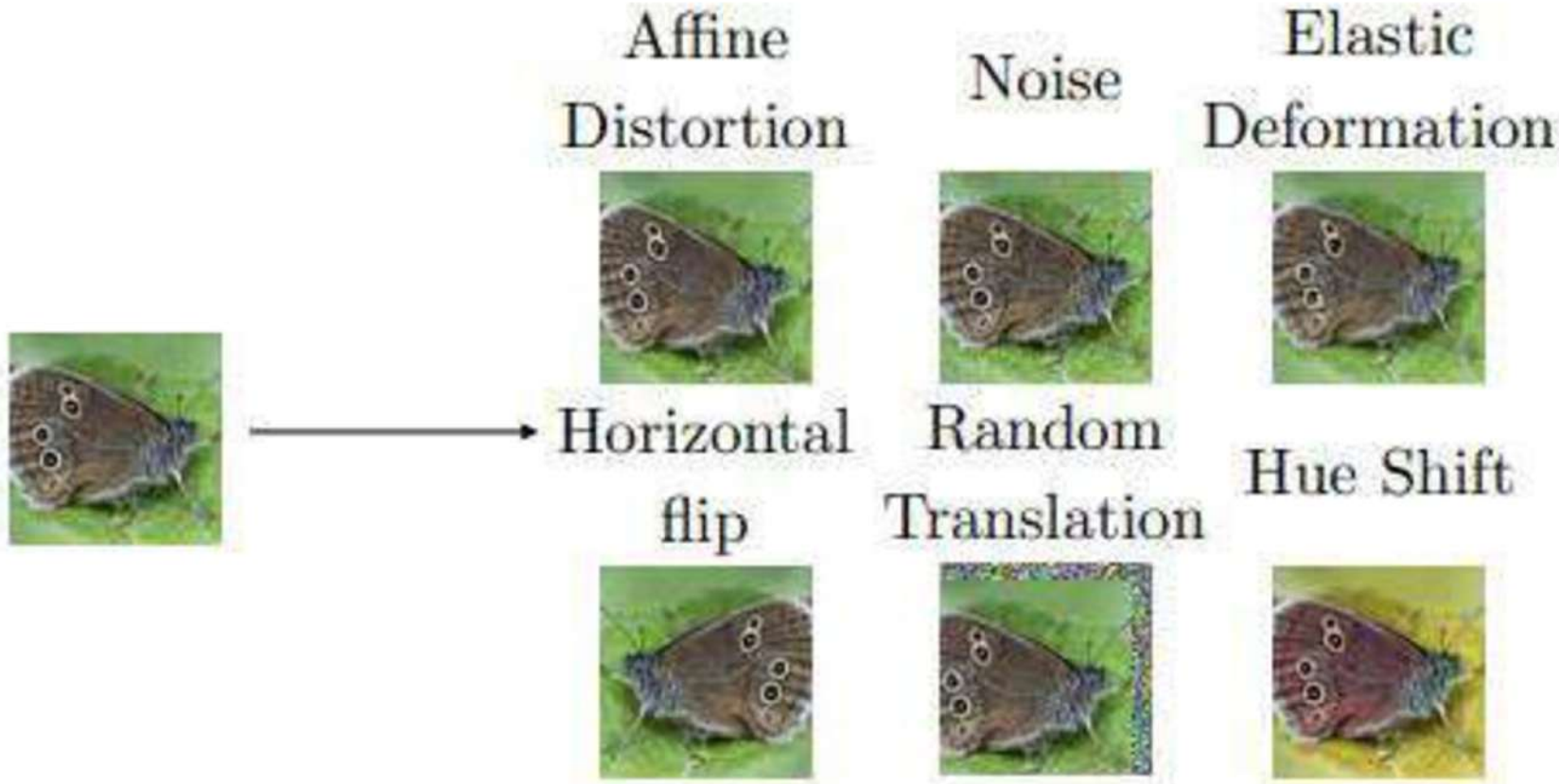
L1 regularization (Lasso regression)

- A regularization technique that requires us to minimize the sum of absolute values between features and target variable.

Dataset augmentation

- Translation
- Flipping
- Injecting noise

Dataset Augmentation



Bagging and boosting

- Ensemble learning helps improve machine learning results by combining several models.
- This approach allows the production of better predictive performance compared to a single model.
- Basic idea is to learn a set of classifiers (experts) and to allow them to vote.
- **Bagging** and **Boosting** are two types of **Ensemble Learning**.
- These two decrease the variance of a single estimate as they combine several estimates from different models. So the result may be a model with higher stability.
- **Bagging**: It is a homogeneous weak learners' model that learns from each other independently in parallel and combines them for determining the model average.
- **Boosting**: It is also a homogeneous weak learners' model but works differently from Bagging. In this model, learners learn sequentially and adaptively to improve model predictions of a learning algorithm.

Bagging and boosting

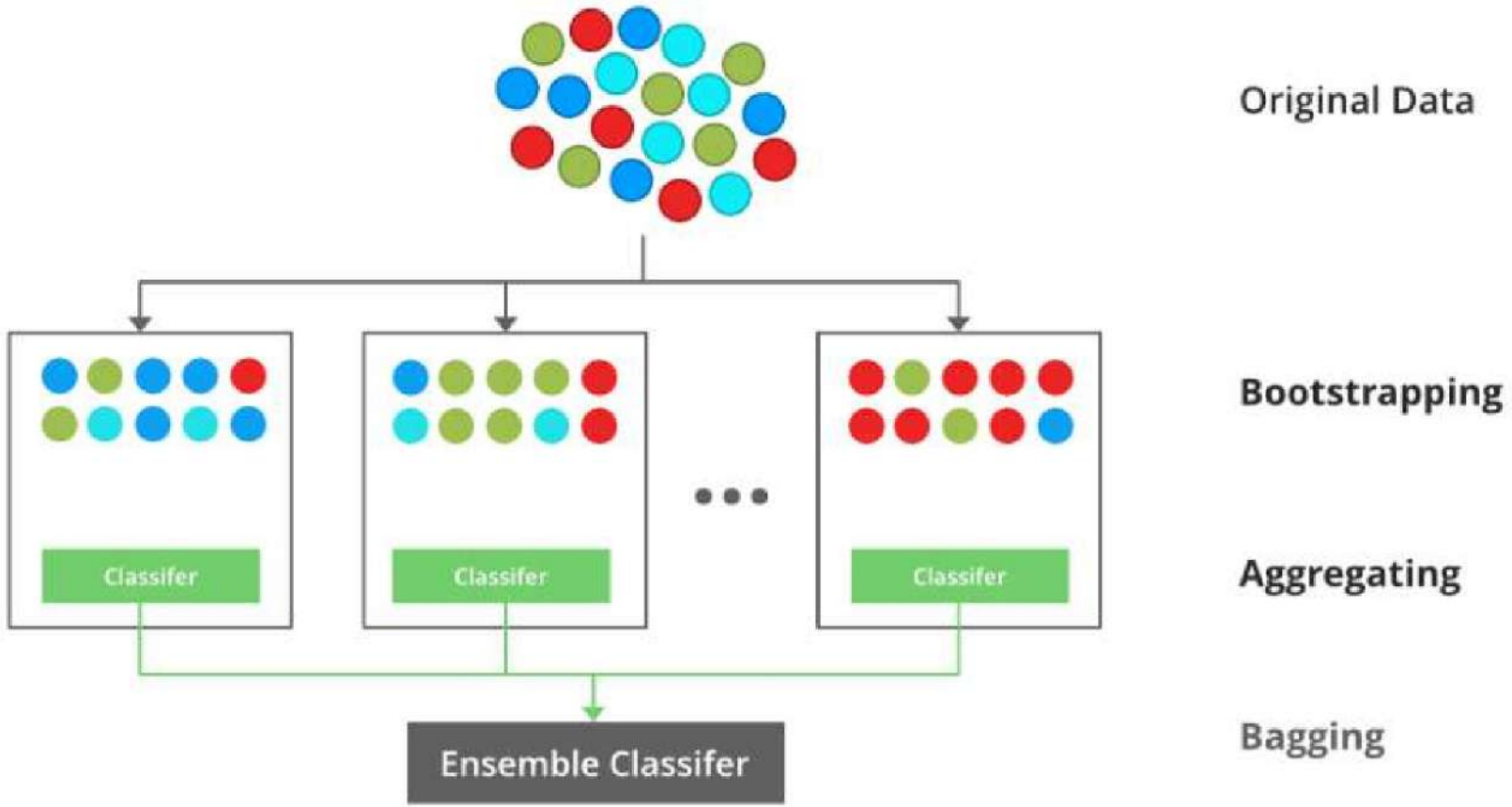
- Adv – better accuracy, lower overfitting
- Drawback – high computation, slower

- helps improve machine learning results by combining several models.
- This approach allows the production of better predictive performance compared to a single model.
- Basic idea is to learn a set of classifiers (experts) and to allow them to vote.
- **Bagging** and **Boosting** are two types of **Ensemble Learning**.
- Ensemble learning helps improve machine learning results by combining several models.
- These two decrease the variance of a single estimate as they combine several estimates from different models. So the result may be a model with higher stability.

- **Bagging**: It is a homogeneous weak learners' model that learns from each other independently in parallel and combines them for determining the model average.
- **Boosting**: It is also a homogeneous weak learners' model but works differently from Bagging. In this model, learners learn sequentially and adaptively to improve model predictions of a learning algorithm.

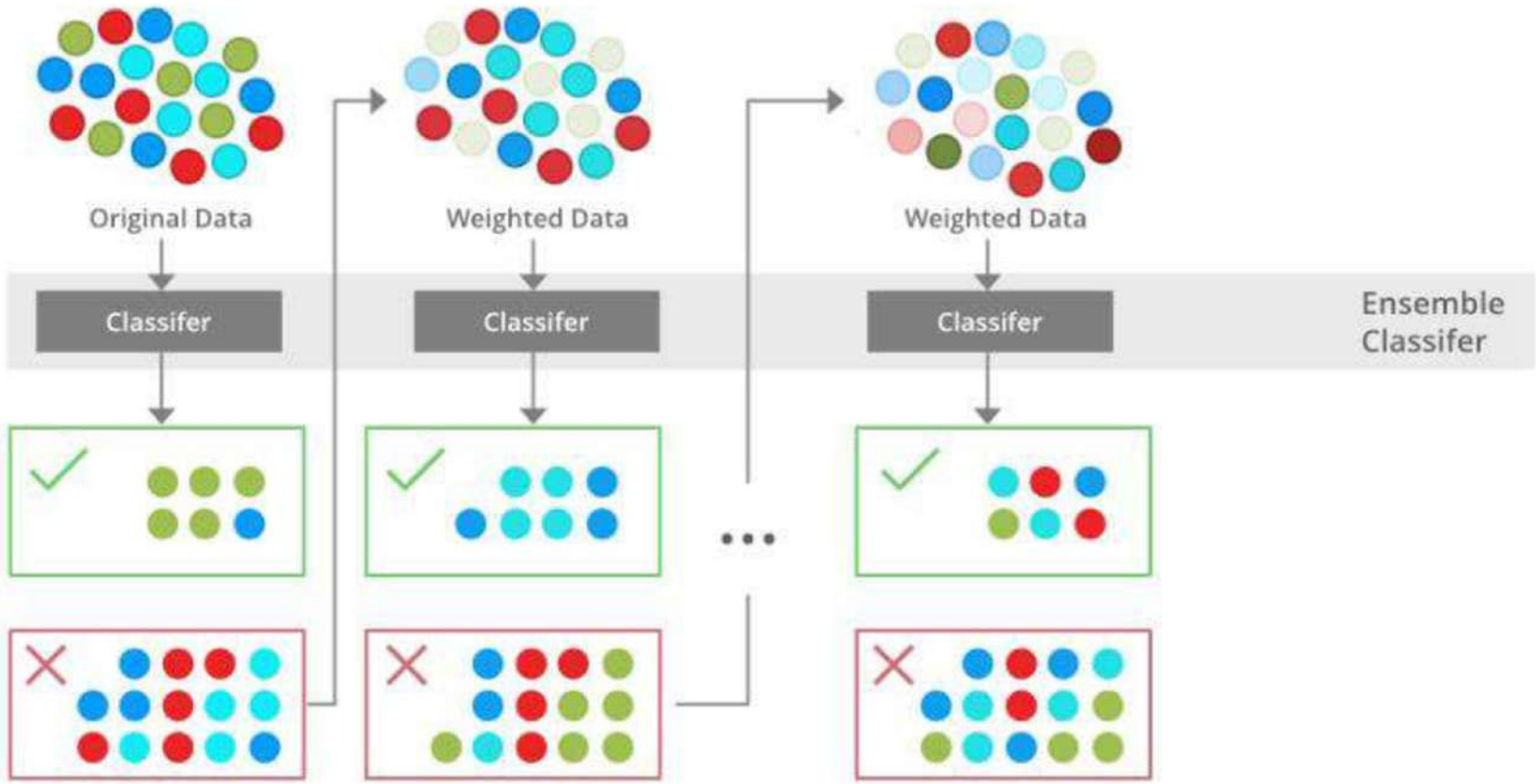
- **Bagging**
- **B**ootstrap **A**ggregating, also known as bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It decreases the variance and helps to avoid overfitting. It is usually applied to decision tree methods. Bagging is a special case of the model averaging approach.

- **Implementation Steps of Bagging**
- **Step 1:** Multiple subsets are created from the original data set with equal tuples, selecting observations with replacement.
- **Step 2:** A base model is created on each of these subsets.
- **Step 3:** Each model is learned in parallel with each training set and independent of each other.
- **Step 4:** The final predictions are determined by combining the predictions from all the models.
-


Original Data
Classifer
Classifer
Classifer
Bootstrapping
Aggregating
Ensemble Classifer
Bagging

- **Boosting**
- Boosting is an ensemble modeling technique that attempts to build a strong classifier from the number of weak classifiers. It is done by building a model by using weak models in series. Firstly, a model is built from the training data**. Then the second model is built which tries to correct the errors present in the first model**. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models is added.

- *Initialise the dataset and assign equal weight to each of the data point.*
- *Provide this as input to the model and identify the wrongly classified data points.*
- *Increase the weight of the wrongly classified data points and decrease the weights of correctly classified data points. And then normalize the weights of all data points.*
- *if (got required results)*
  Goto step 5
  else
  Goto step 2
- *End*



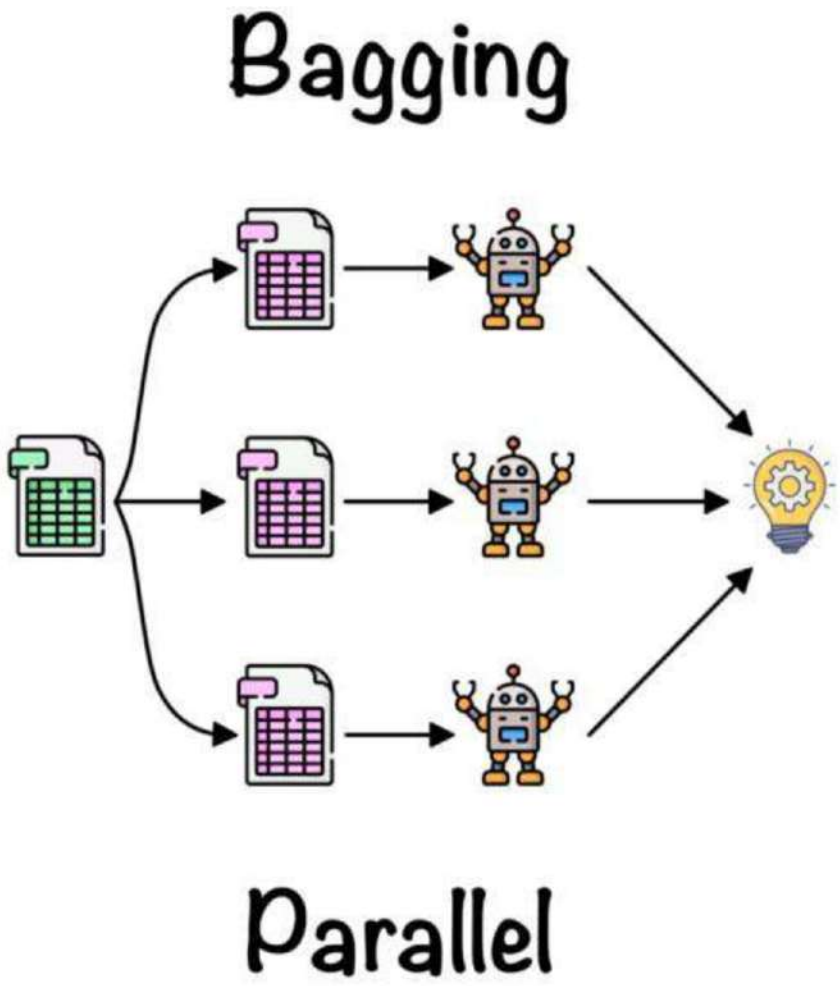
An illustration presenting the intuition behind the boosting algorithm, consisting of the parallel learners and weighted dataset.

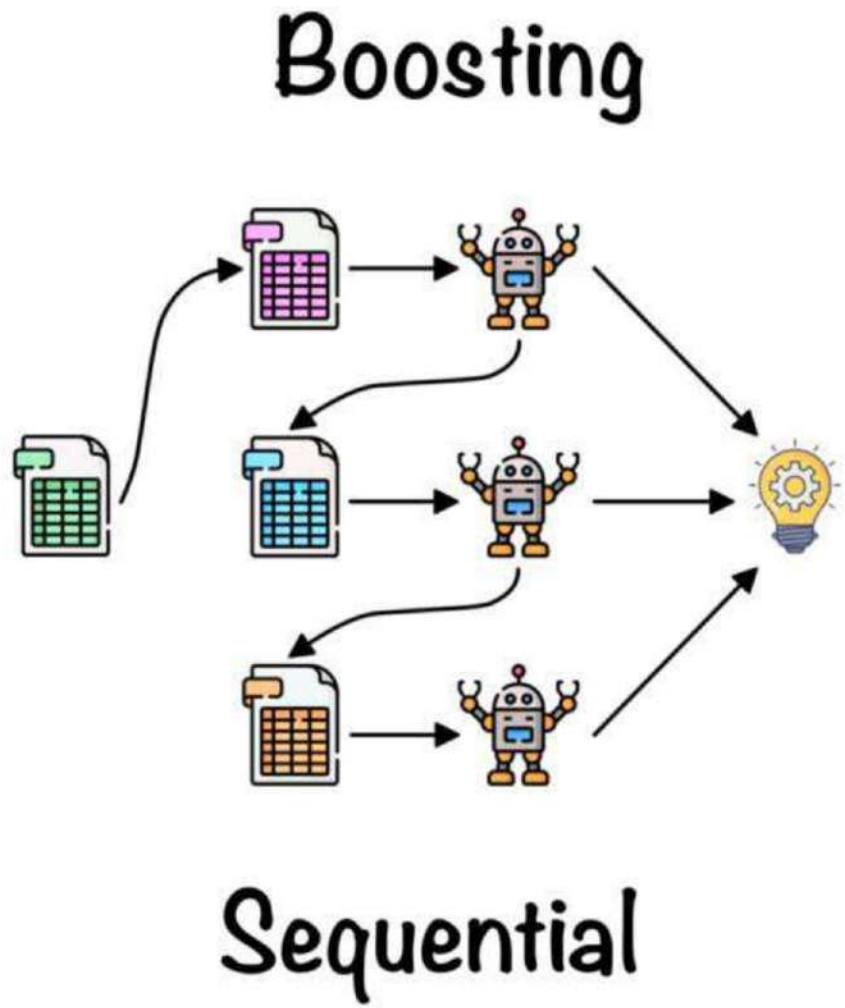
Similarities Between Bagging and Boosting

Bagging and Boosting, both being the commonly used methods, have a universal similarity of being classified as ensemble methods. Here we will explain the similarities between them.

1. Both are ensemble methods to get N learners from 1 learner.
2. Both generate several training data sets by random sampling.
3. Both make the final decision by averaging the N learners (or taking the majority of them i.e Majority Voting).
4. Both are good at reducing variance and provide higher stability.

Random Forest: Bagging vs Boosting



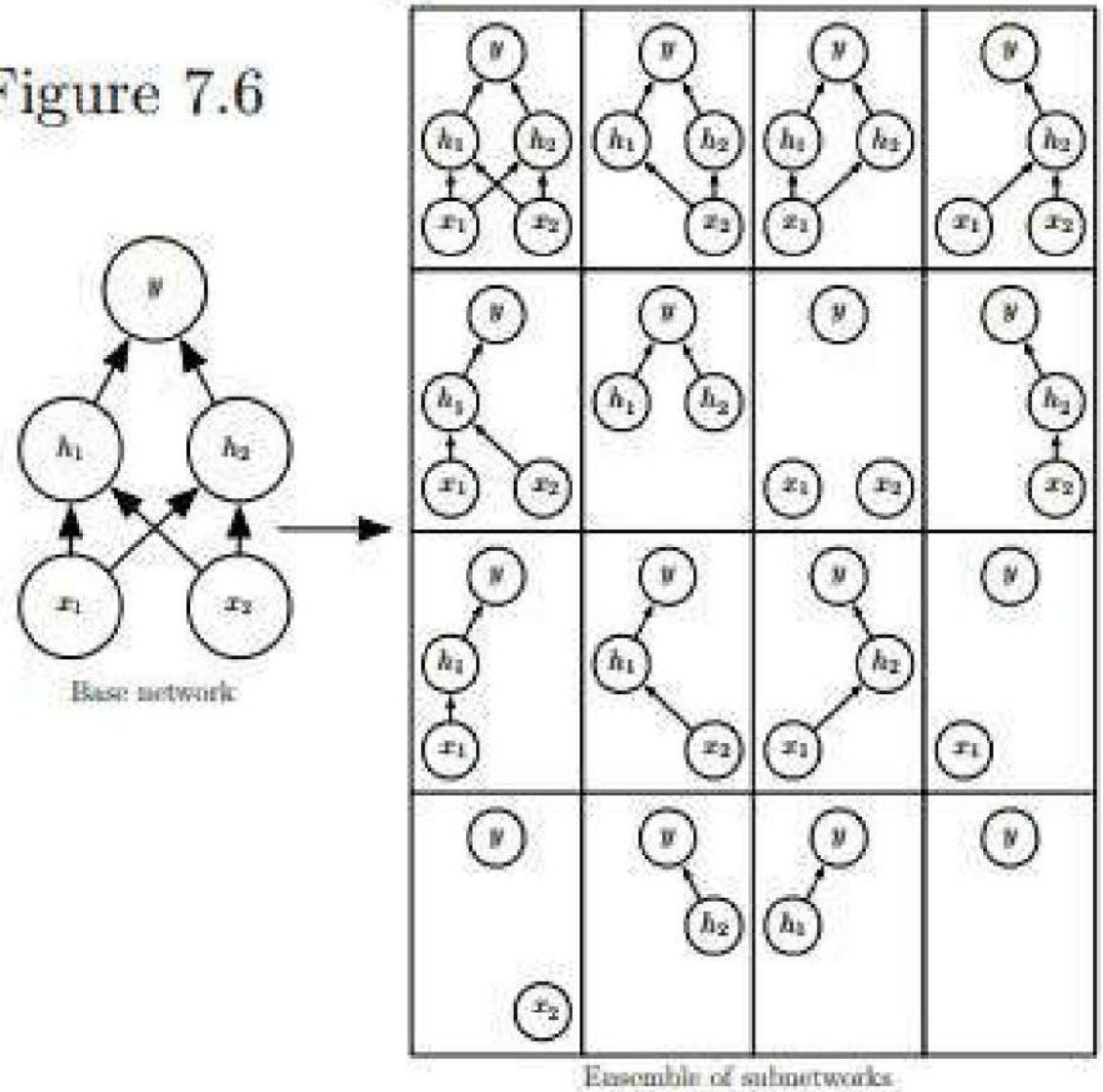


towardsdatascience.com/

Dropout

computationally inexpensive but powerful method of regularization

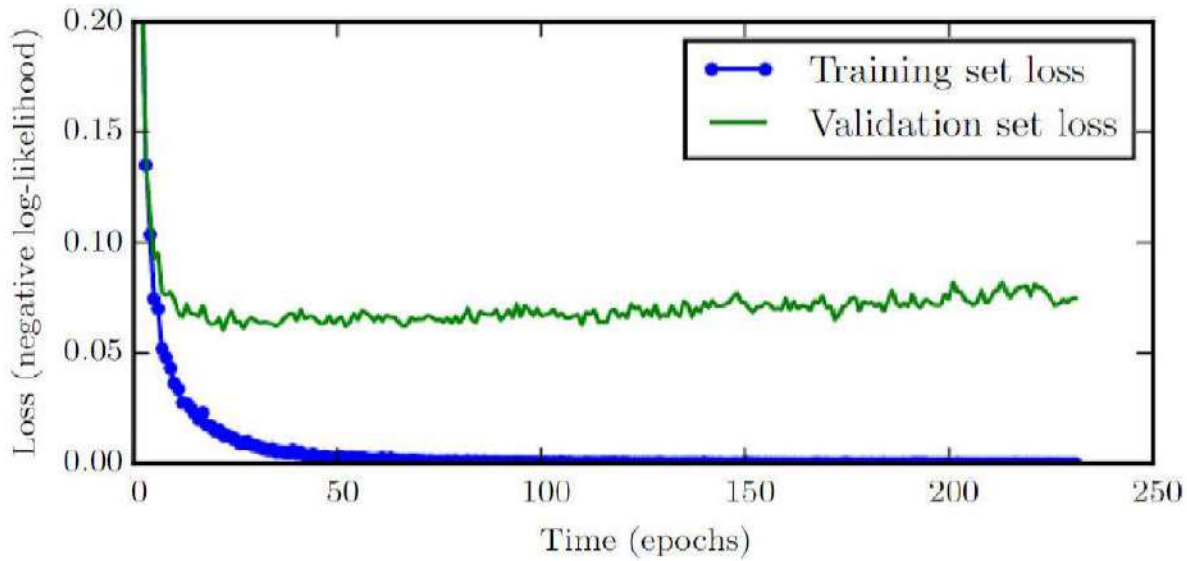
Figure 7.6



- Bagging involves training multiple models, and evaluating multiple models on each test example
- This seems impractical when each model is a large neural network, since training and evaluating such networks is costly in terms of runtime and memory.
- In most modern neural networks, based on a series of affine transformations and nonlinearities, we can effectively remove a unit from a network by multiplying its output value by zero

Early stopping

- When training large models with sufficient representational capacity to overfit the task, we often observe that training error decreases steadily over time, but validation set error begins to rise again



- Obtain a model with better validation set error by returning to the parameter setting at the point in time with the lowest validation set error.
- **Every time the error on the validation set improves, we store a copy of the model parameters.**
- When the training algorithm terminates, we return these parameters, rather than the latest parameters

Parameter Tying and Parameter Sharing

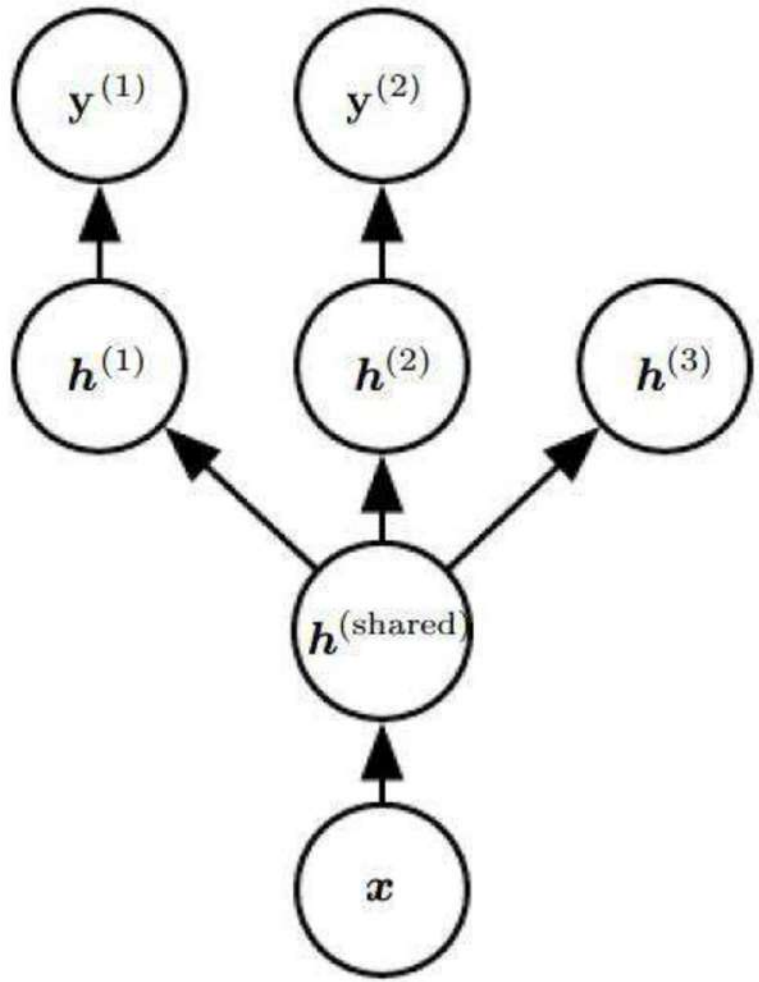
- Refer text book

Multi-task Learning

- optimize a single model for multiple tasks
- The sharing of representations between related tasks, a model will learn better decision boundaries on the original task.
- Such an approach is called “Multi-Task Learning.”

- Multi-Task Learning refers to a **single shared machine learning model that can perform multiple different** (albeit related) tasks.
- Multi-Task Learning offers advantages like improved data efficiency, faster model convergence, and reduced model overfitting due to shared representations.
- For example, learning to ride a bicycle makes it easy for someone to learn to ride a motorbike later on, which builds upon similar concepts of body balance. This is referred to as the **inductive transfer of knowledge.**

- This mechanism of knowledge transfer is what allows humans to learn new concepts with only a few examples or no examples at all (which in Machine Learning is called "Few-Shot Learning" and "Zero-Shot Learning," respectively).
- For example, a model trained for image classification learns to classify samples by localizing specific objects in the images.
- Such a model, when used in conjunction with solving an object detection problem, already has the underlying feature localization capability, leading to faster model convergence. Thus, the same model is used to generalize over different tasks.


$\mathbf{y}^{(1)}$
$\mathbf{y}^{(2)}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{h}^{(3)}$
$\boldsymbol{h}^{(\text{shared})}$
$\boldsymbol{x}$

Optimization

The goal of Optimization in Deep learning-

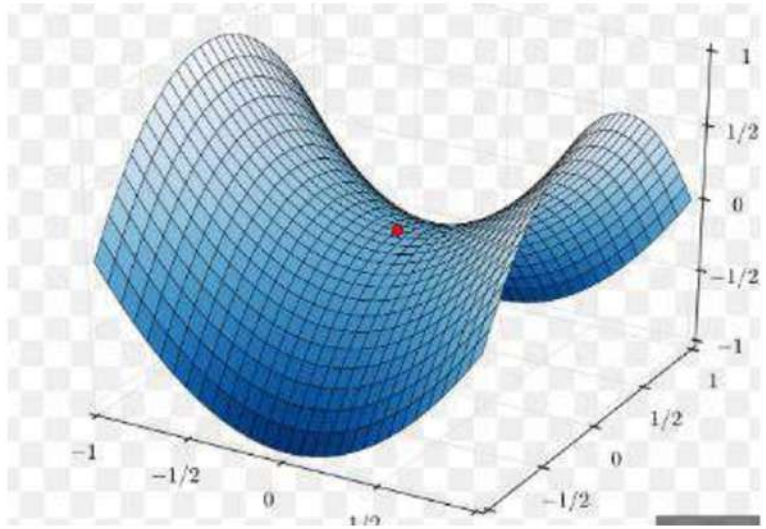
- Deep learning **aims to decrease generalization error**.
- In order to achieve this, we must be aware of overfitting as well as use the optimization procedure to lower the training error.

Why is optimization important in deep learning?

- The process of optimization **aims to lower the risk of errors or loss from these predictions, and improve the accuracy of the model**.
- Machine learning models are often trained on local or offline datasets which are usually static.
- Optimization improves the accuracy of predictions and classifications, and minimizes error.

Saddle point

- a saddle point or minimax point is a point on the surface of the graph of a function where the slopes (derivatives) in orthogonal directions are all zero (a critical point), but which is not a local extremum of the function.
-



Learning and Optimization

- Learning and pure optimization are two different concepts in the field of artificial intelligence and machine learning.
- Pure optimization refers to the process of finding the **optimal values for a set of parameters that maximize or minimize a given objective function**. This process typically involves using mathematical techniques such as calculus or linear algebra to iteratively update the parameter values until the objective function is optimized.
- learning refers to the process of acquiring knowledge or skills through experience, study, or instruction. In the context of machine learning, learning involves using data to automatically improve the performance of a model over time. This is achieved through algorithms that adjust the model's parameters based on the data it receives, allowing it to make better predictions or decisions.

- learning is a more general process that involves adapting to new situations and acquiring knowledge, while optimization is a more specific process that involves finding the best values for a given set of parameters.

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α.

Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.

Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$

Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$

Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

end while

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ

Require: Initial parameter $\boldsymbol{\theta}$

Require: Small constant δ, perhaps 10^{-7}, for numerical stability

Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ.

Require: Initial parameter $\boldsymbol{\theta}$

Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers.

Initialize accumulation variables $\boldsymbol{r} = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$

Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$

end while

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})

Require: Initial parameters $\boldsymbol{\theta}$

Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$

Initialize time step $t = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

$t \leftarrow t + 1$

Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$

Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$

Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$

Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$

Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

Recurrent neural network-RNN

- Both ANN and CNN generally have
  - Fixed size inputs
  - The whole input available simultaneously

RNN

- Sequential processing
  - Time series data
- Variable sized input
- Eg: language translation, speech processing,
- Video analysis, task handled by Alexa

Applications

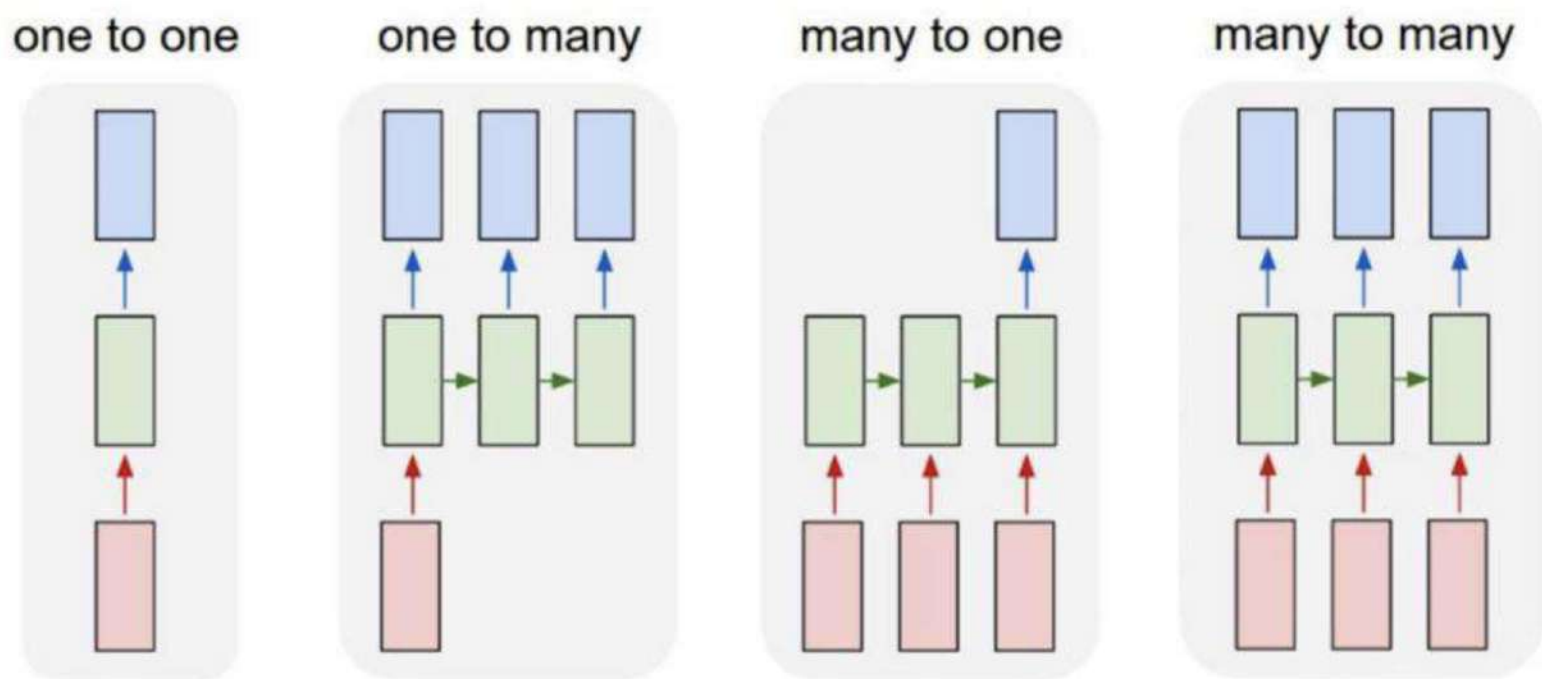
- Text generation
- Stock price prediction
- Image Captioning
- Time series analysis
- Translation
- Visual sequence tasks
- Cursive handwriting recognition

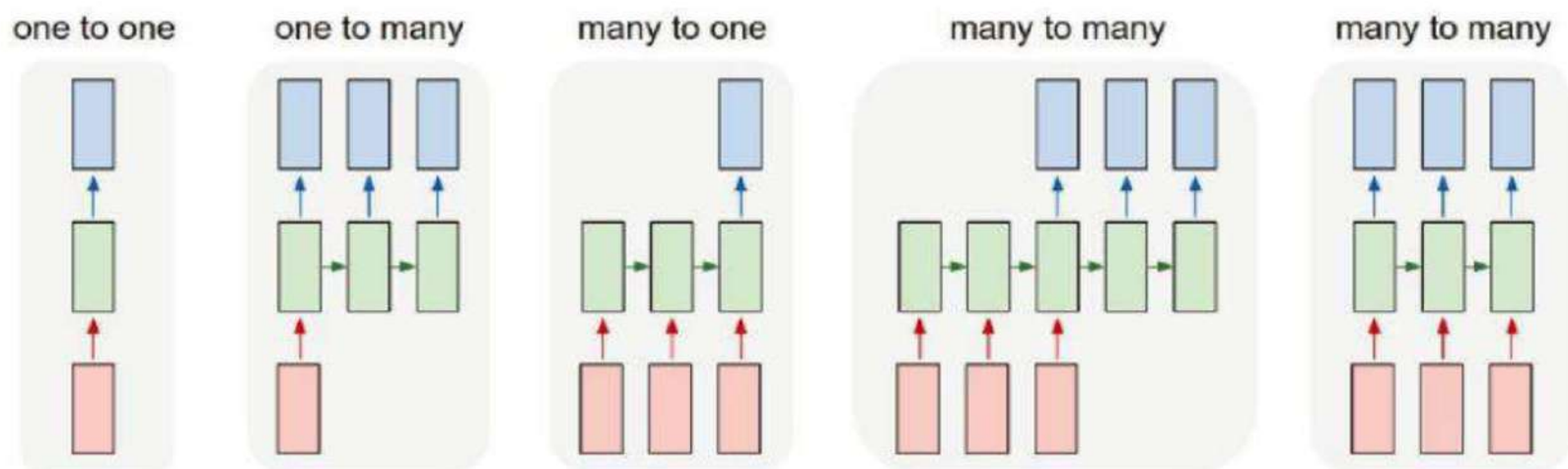
When do you need to use a Recurrent Neural Network ?

- **“Whenever there is a sequence of data and that temporal dynamics that connects the data is more important than the spatial content of each individual frame.”**
- Examples of sequential data are financial data or the DNA sequence.
- The most popular type of sequential data is perhaps Time series data, which is just a series of data points that are listed in time order.

Examples of RNN Application

- While Feed-Forward Neural Networks map one input to one output, RNN's can map one to many, many to many and many to one.

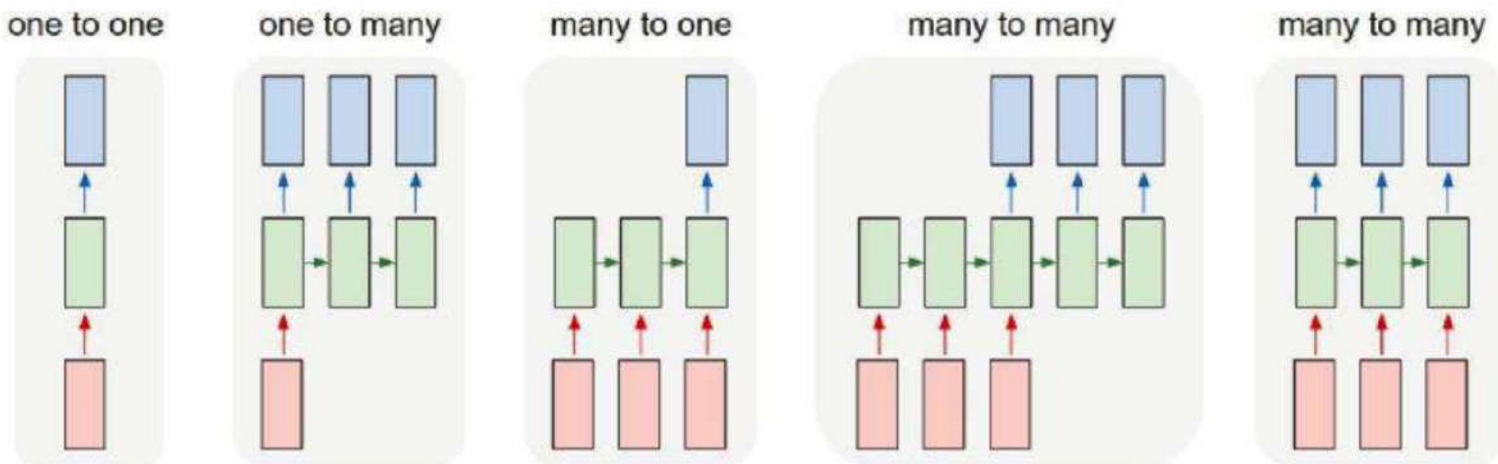




Vanilla Neural Network

fixed-sized input -> fixed-size output

e.g. image classification

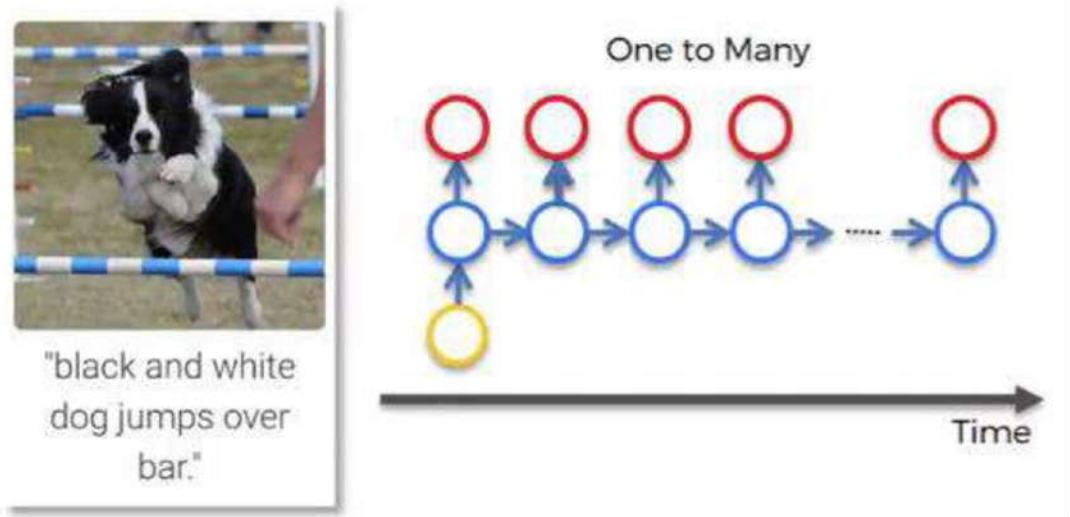


sequence output

e.g. image captioning

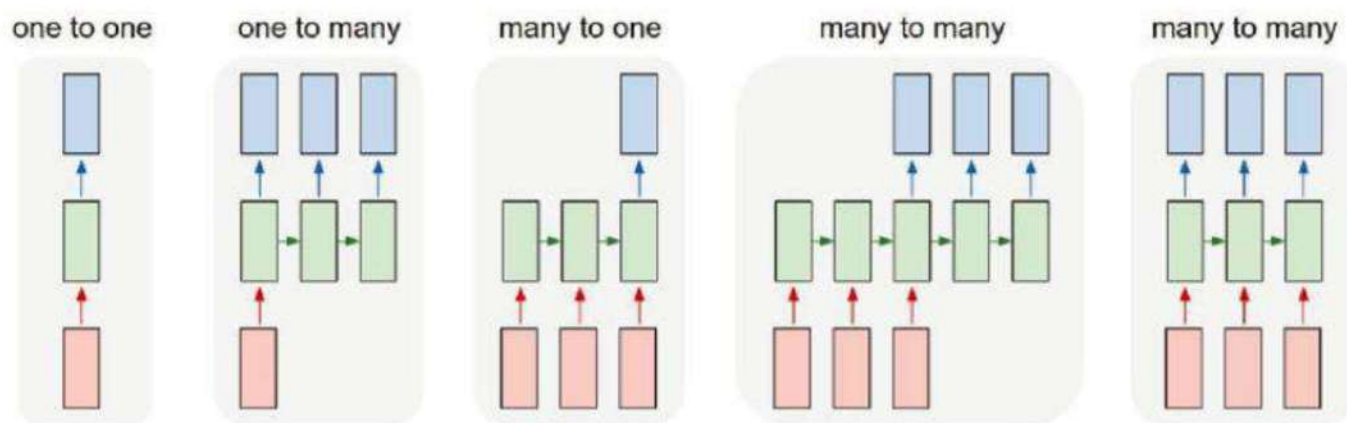
image -> sequence of words

- **One to many** This is a network with one input and multiple outputs.
- For instance, it could be an image (input), which is described by a computer with words (outputs).



•This picture of the dog first went through CNN and then was fed into RNN. The network describes the given picture as “black and white dog jumps over bar”.

•While CNN is responsible here for image processing and feature recognition, our RNN allows the computer to make sense out of the sentence. As you can see, the sentence actually flows quite well.

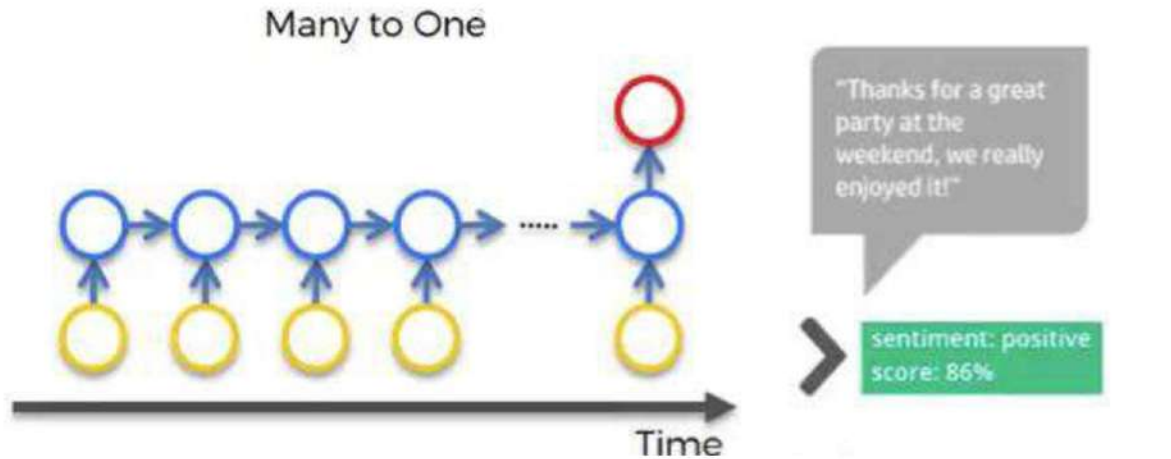


sequence input

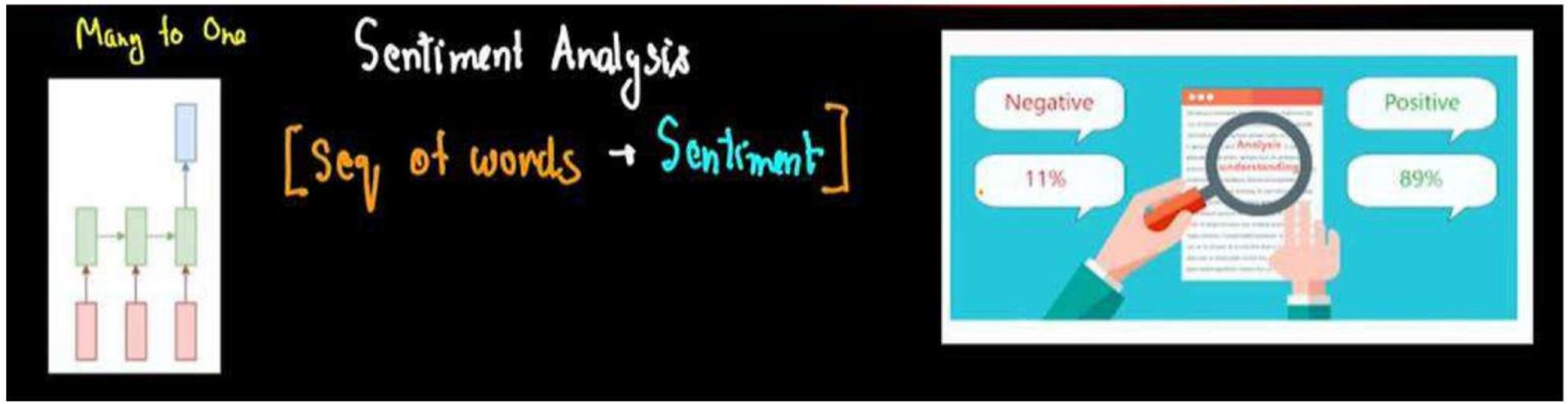
e.g. sentiment classification

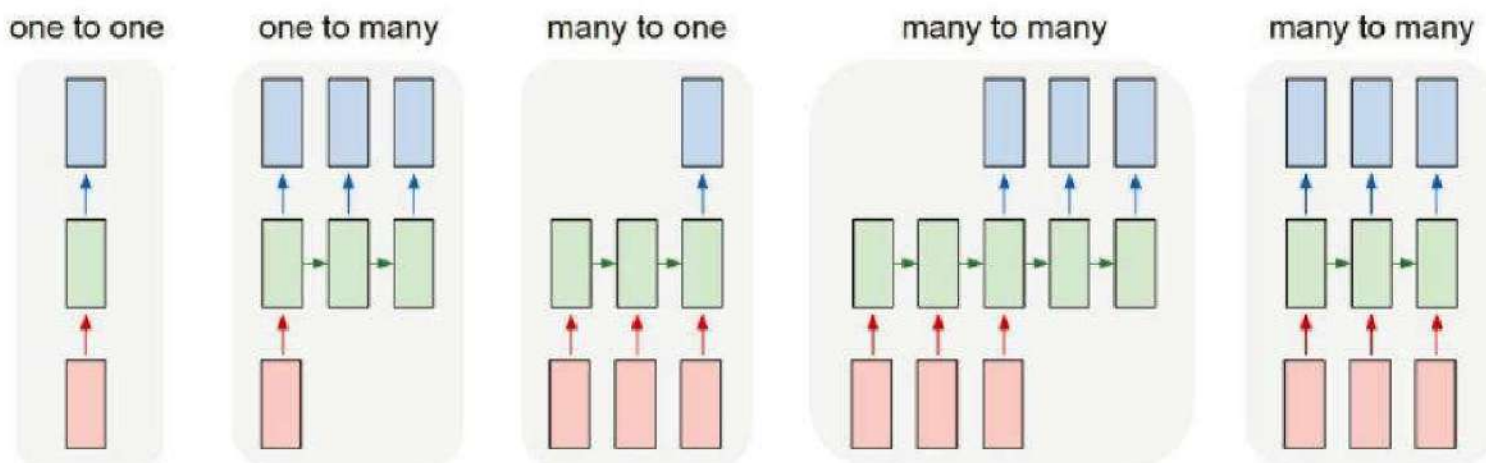
sequence of words -> sentiment

- **Many to one**
- An example of this relationship would be sentiment analysis, when you have lots of text, such as a customer's comment, for example, and you need to gauge what's the chance that this comment is positive, or how positive this comment actually is, or how negative it is.



Many to one


Many to One
Sentiment Analysis
[Seq of words → Sentiment]
Negative
11%
Positive
89%

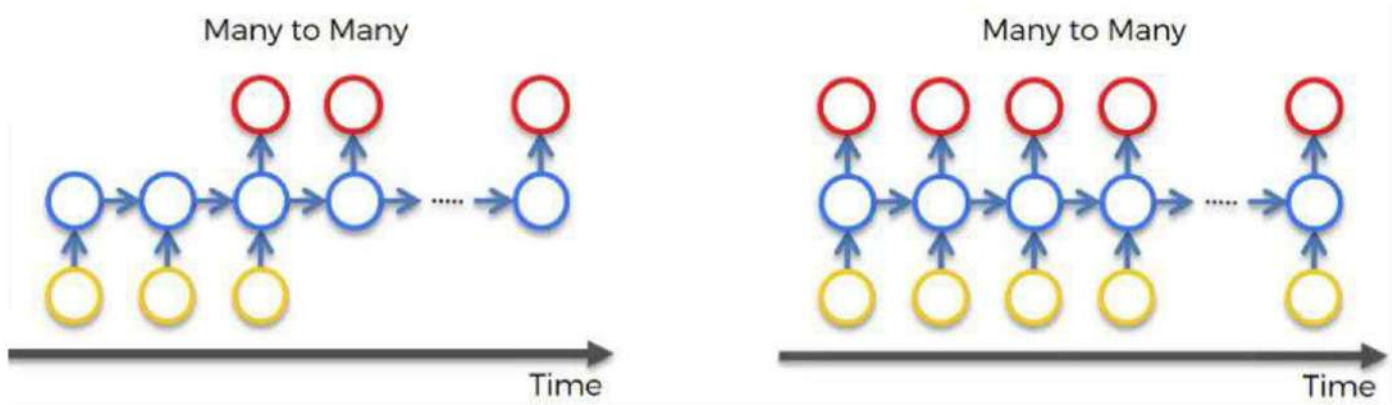


sequence input and sequence output

e.g. machine translation

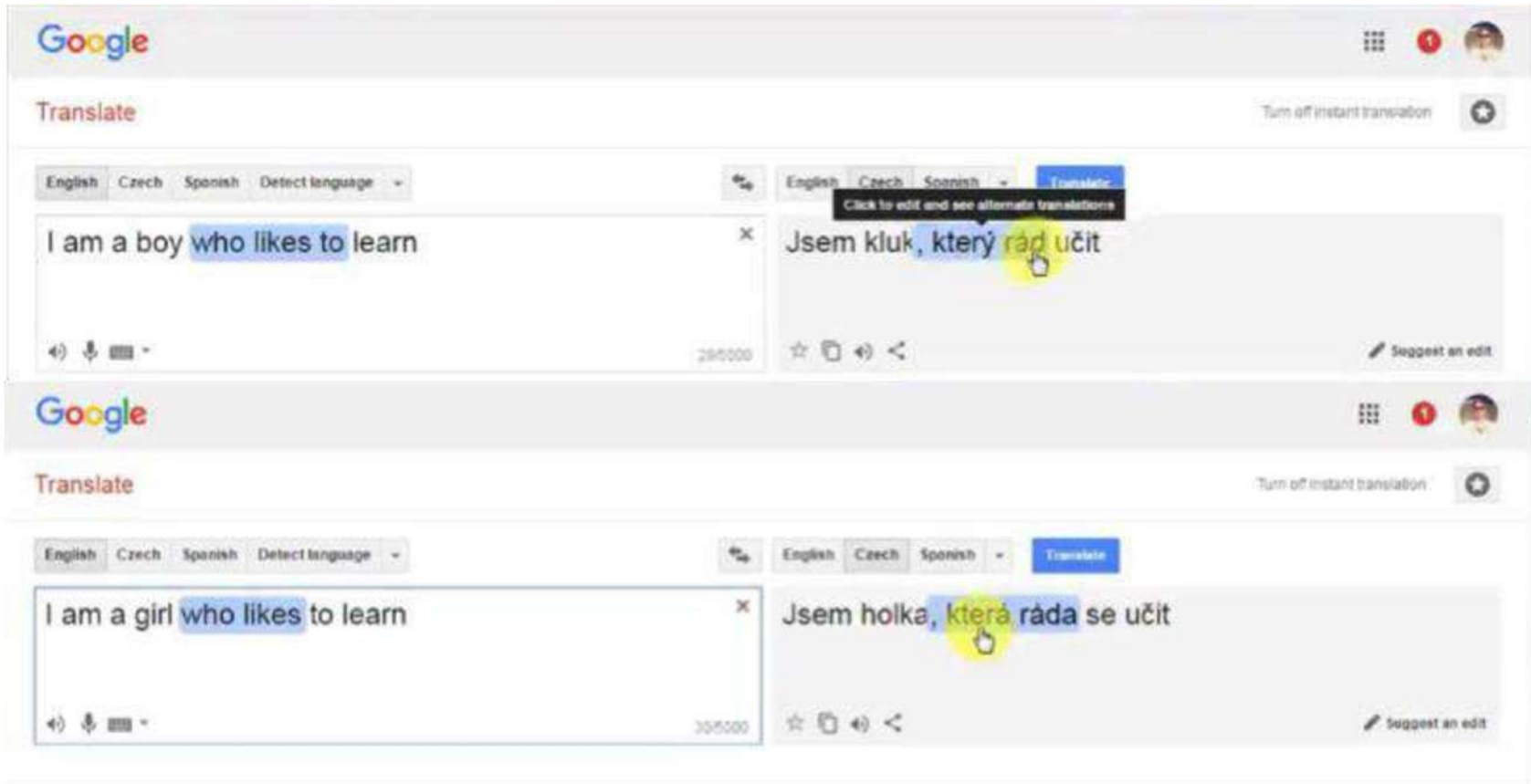
seq of words -> seq of words

- **Many to many**



•Translations can be a good example of many to many type of network.
•We don’t know if Google Translator uses RNNs or not, but the concept remains the same.
•Speech recognition

- As you can see in the picture below, we're translating one sentence from English to Czech. In some other languages, including Czech, it is important for the verb phrase, what gender your person is.



So, when we have "a boy" in the input sentence, the translation of the "who likes" part looks like "který rád". But as we change a person to "a girl", this part changes to "která ráda", reflecting the change of the subject.

Many to many

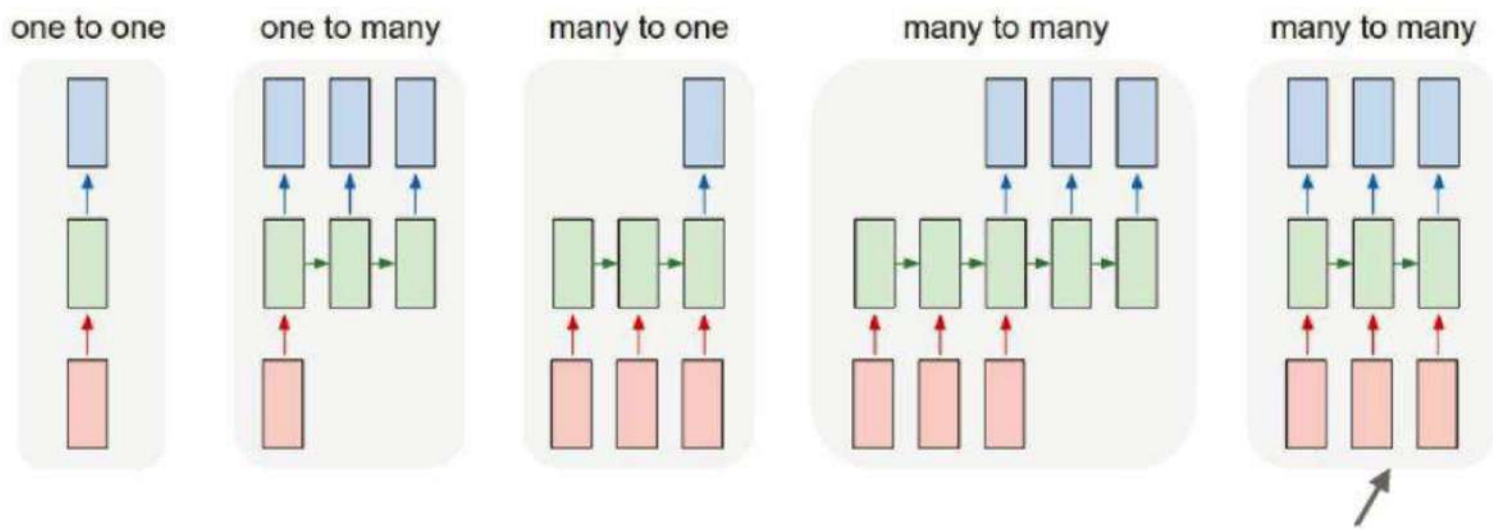

Language Translation
[Seq of words → Seq of words]
Speech recognition
[Seq of audio signal → Seq of words]
Output size independent
Alexa
Alexa
Trigger Word Recognition
Barge-In / ASR Assist
Beam Forming
Google Translate
Japanese
Tap to enter text

- The concept is the following: you need the short-term information about the previous word to translate the next word.

- You can’t just translate word by word. And that’s where RNNs have power.

- Of course, not every example has to be related to text or images. There can be lots and lots of different applications of RNN.

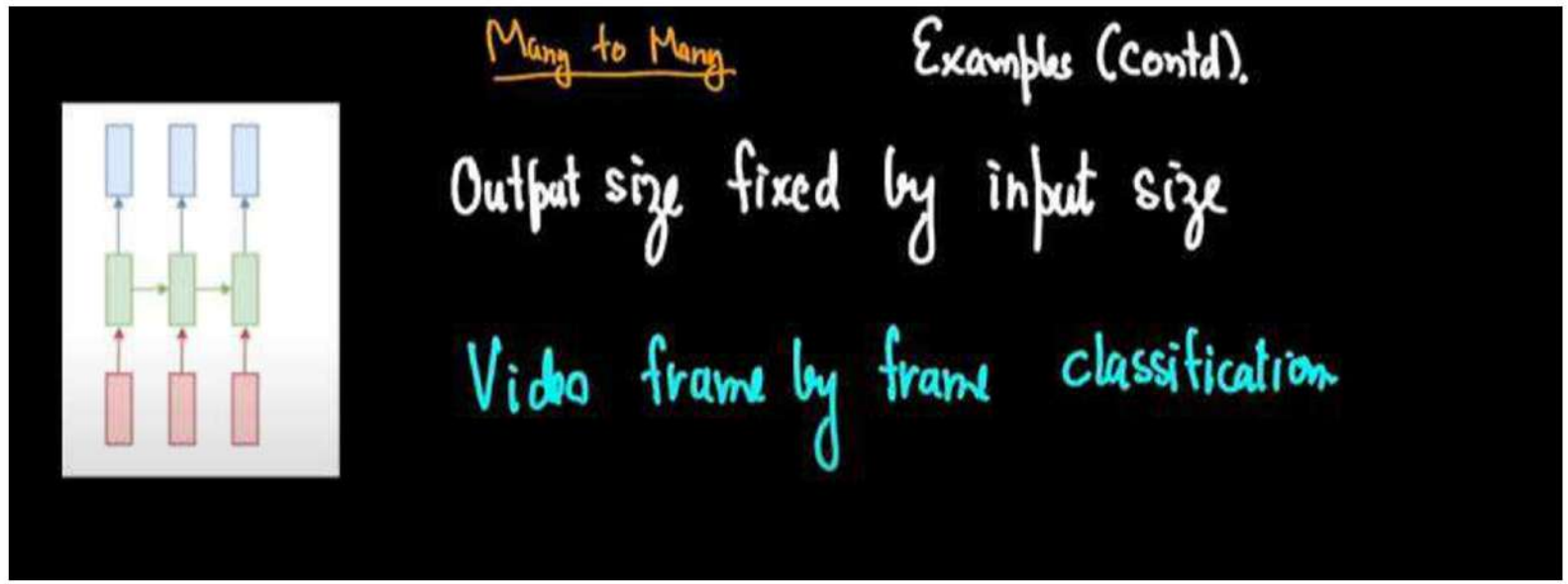
- For instance, many to many relationship is reflected in the network used to generate subtitles for movies. That’s something you can’t do with CNN because you need context about what happened previously to understand what’s happening now, and you need this short-term memory embedded in RNNs.



synced sequence input and output

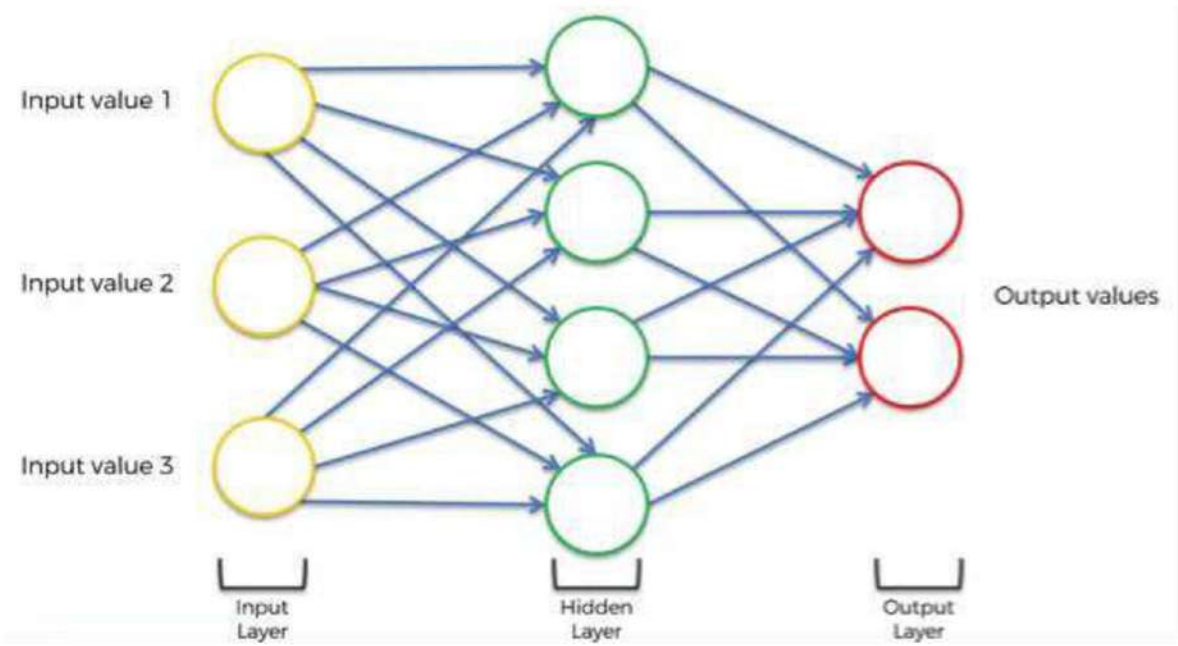
e.g. video classification on frame level

Many to many

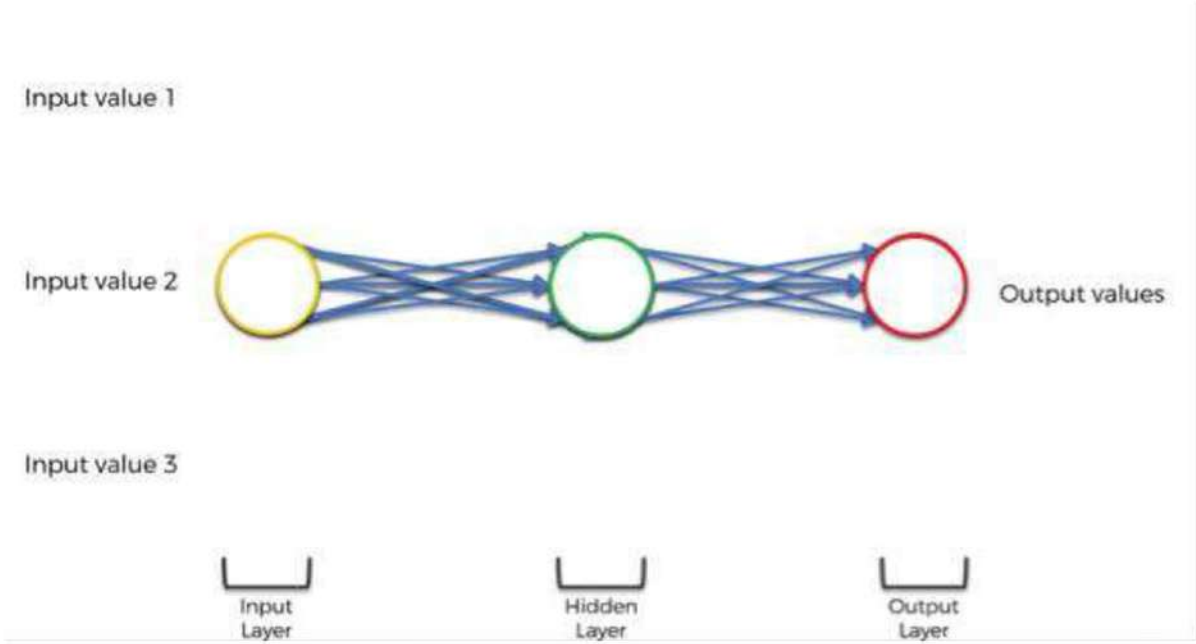

Many to Many
Examples (contd).
Output size fixed by input size
Video frame by frame classification

Representation of the Recurrent Neural Networks

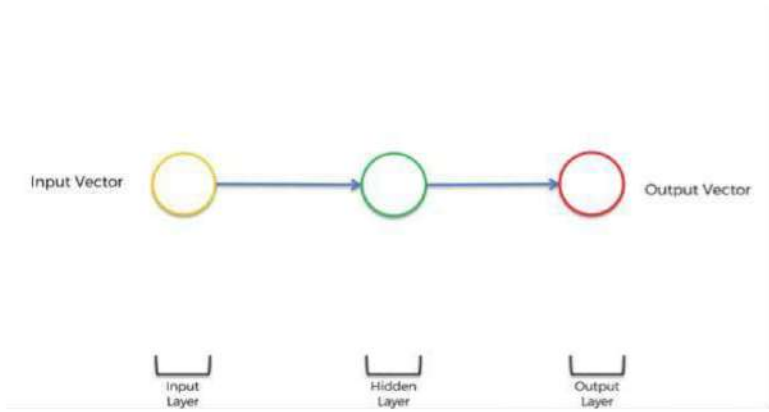
- We begin with a transformation of a simple ANN shown below into RNN.



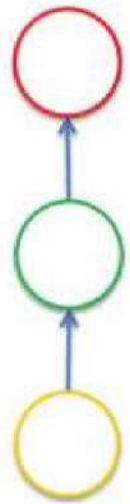
- Squashing the network. The layers are still there but think of it as if we're looking from underneath this neural network.



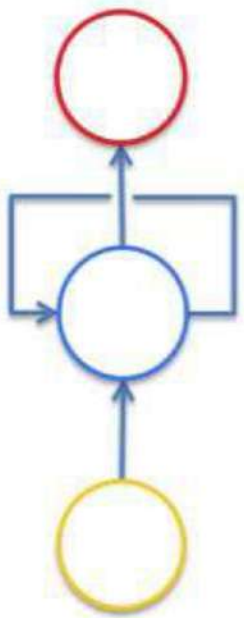
•Changing the multiple arrows into two.



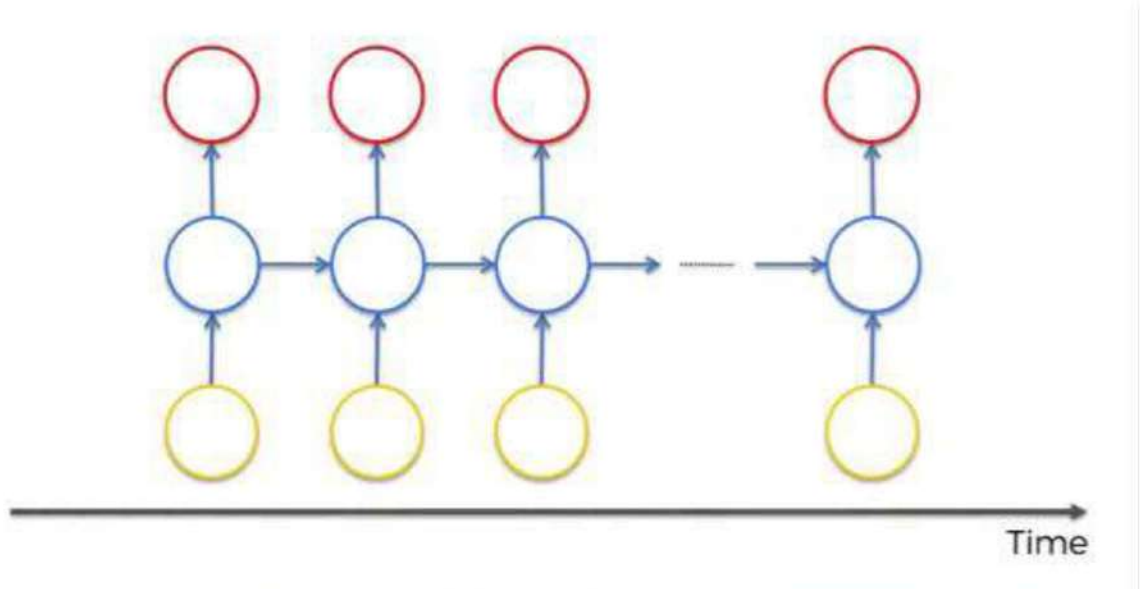
- Twisting it to make it vertical because that's the standard representation.



•Adding a line, which represents a temporal loop. This is an old representation of RNNs and basically means that this hidden layer not only gives an output but also feeds back into itself.



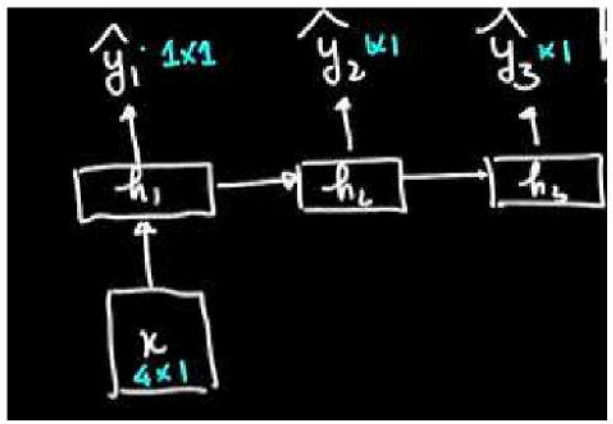
- Unrolling the temporal loop and representing RNNs in a new way. Now each circle represents not only one neuron, but a whole layer of neurons.

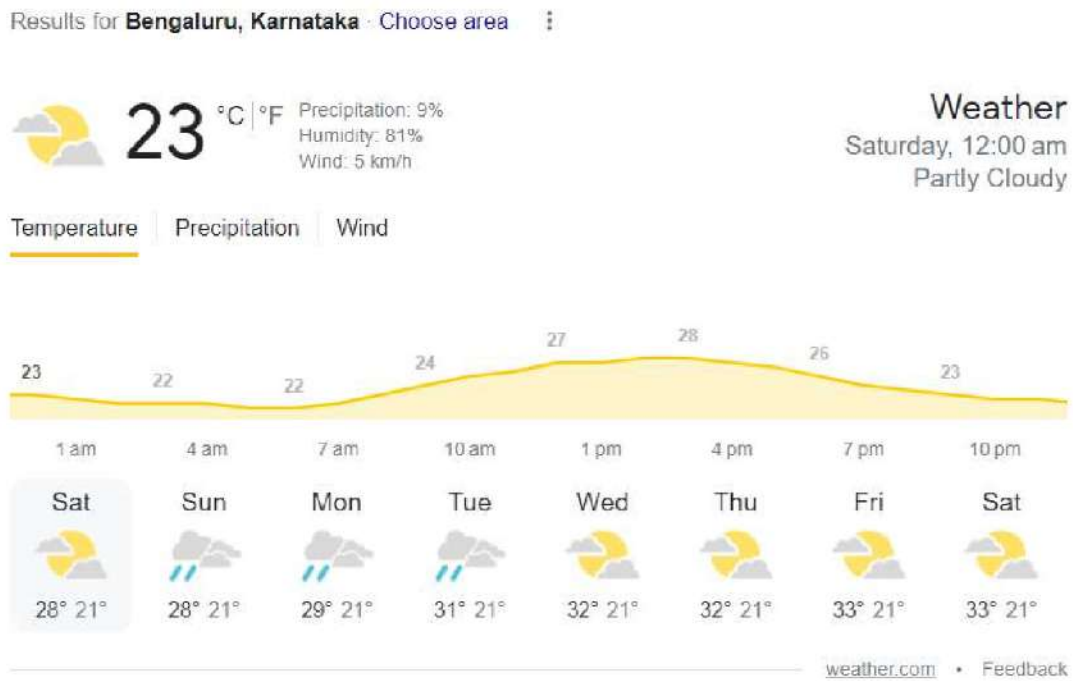


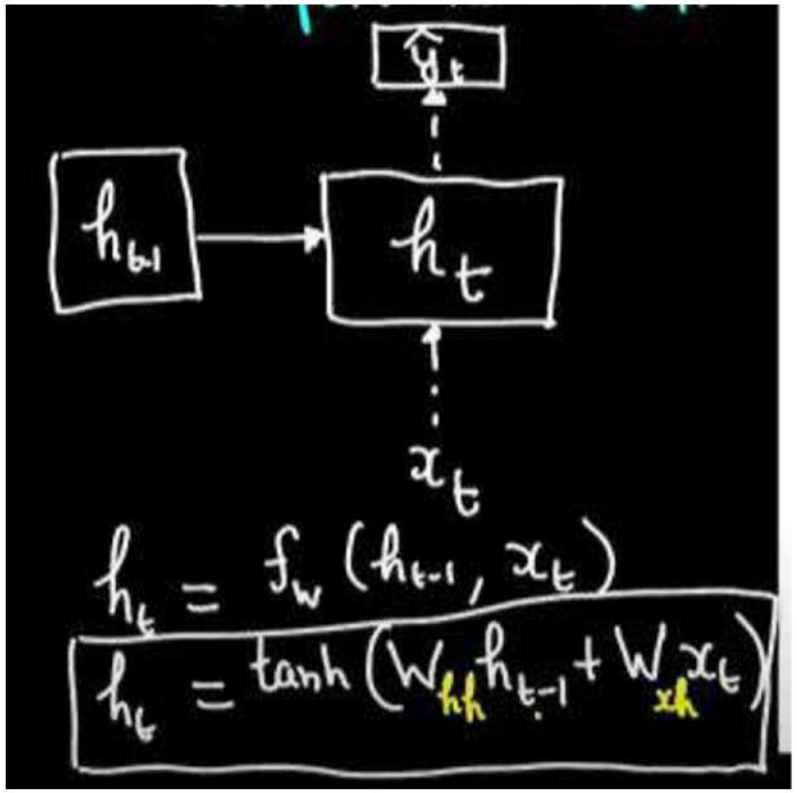
•The idea behind RNNs is that the neurons have some sort of short-term memory providing them with the possibility to remember, what was in this neuron just previously.

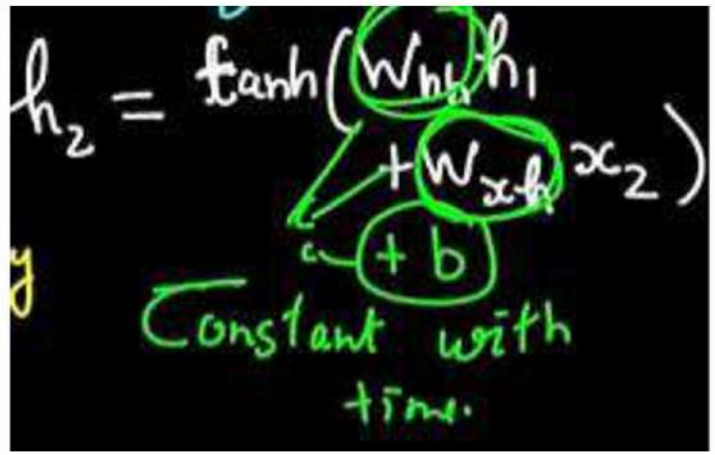
•Thus, the neurons can pass information on to themselves in the future and analyze things.

- RNN will try to incorporate the idea of equally spaced , repetitive temporal relationships.

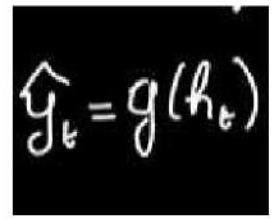








Whh and wxh remains constant throughout



- G can be a linear or non linear function and it depends on classification or regression problem
- If binary classification- g can be sigmoid
- If multiple classification- g can be softmax

How does RNN work ?

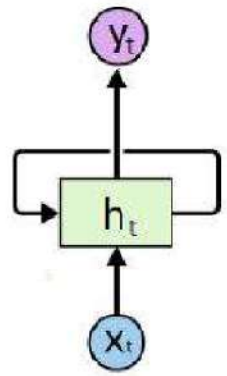
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step



- Have **memory** that keeps track of information observed so far
- Maps from the entire history of previous inputs to each output
- Handle sequential data

How does RNN work ?

- Imagine you have a normal feed-forward neural network and give it the word “neuron“ as an input and it processes the word character by character.
- At the time it reaches the character “r“, it has already forgotten about “n“, “e“ and “u“, which makes it almost impossible for this type of neural network to predict what character would come next.
- A Recurrent Neural Network is able to remember exactly that, because of it’s internal memory. It produces output, copies that output and loops it back into the network.

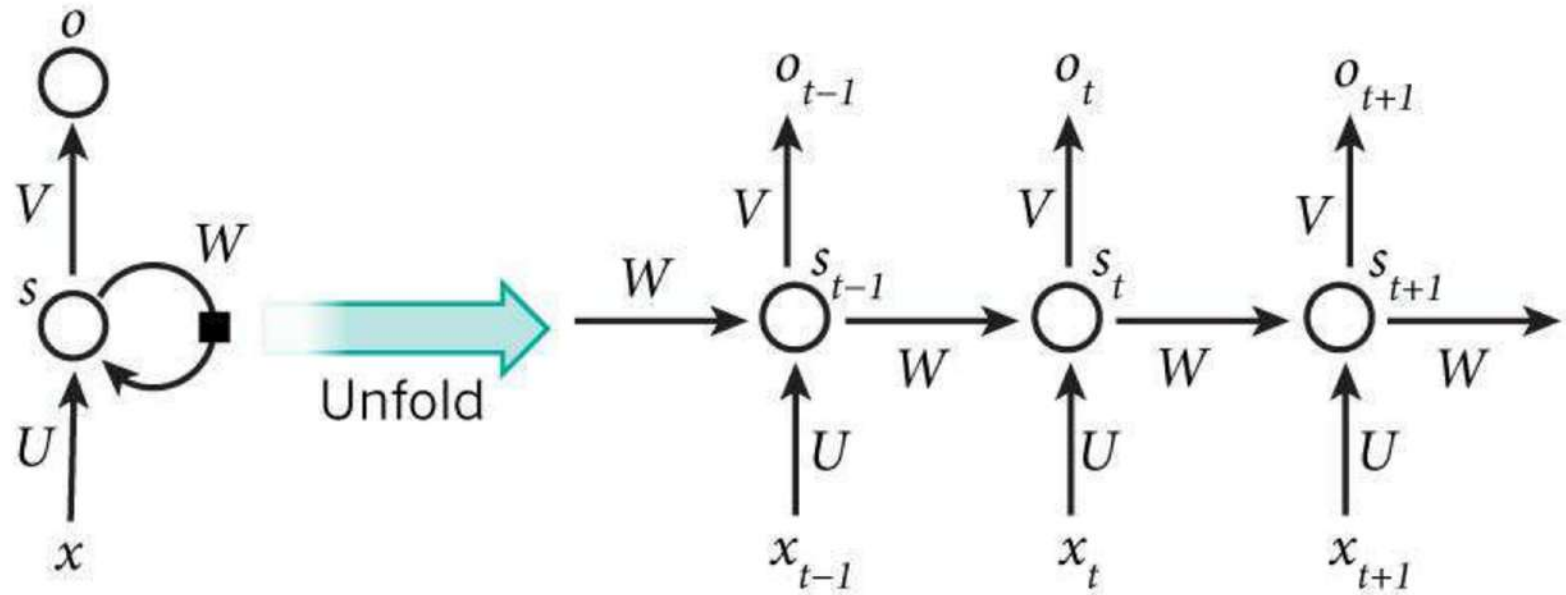
How does RNN work ?

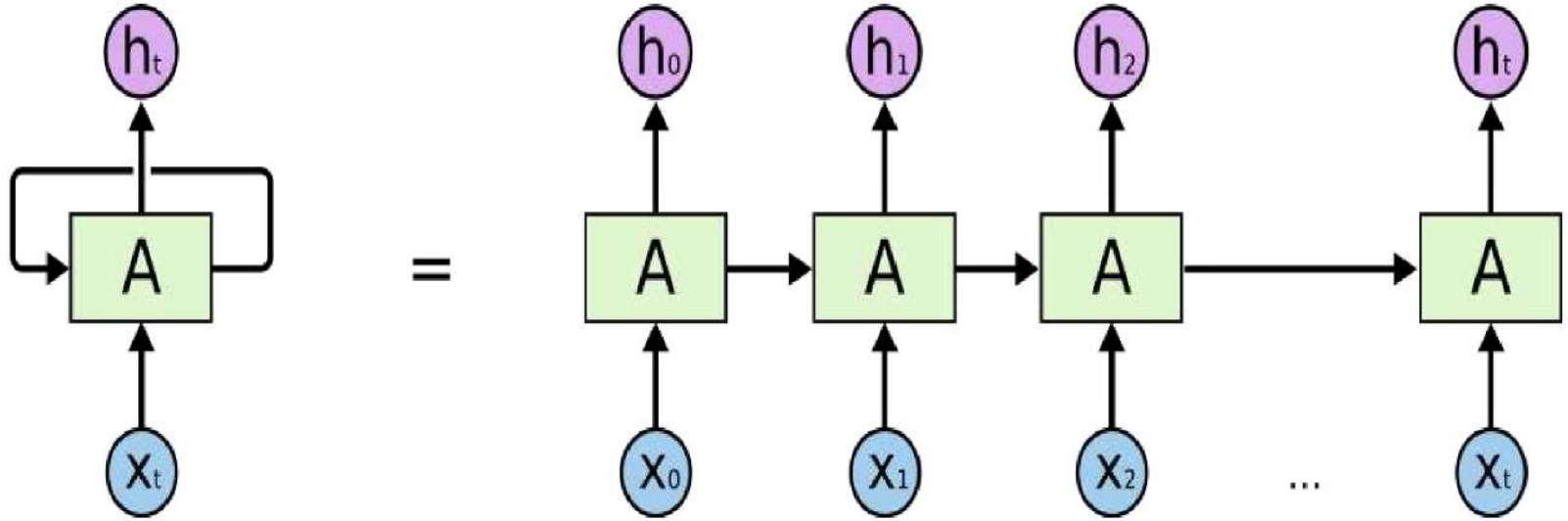
- Therefore a Recurrent Neural Network has two inputs, the present and the recent past.
- This is important because the sequence of data contains crucial information about what is coming next, which is why a RNN can do things other algorithms can’t.
- A Feed-Forward Neural Network assigns, like all other Deep Learning algorithms, a weight matrix to its inputs and then produces the output.
- Note that RNN’s apply weights to the current and also to the previous input.
- Furthermore they also tweak their weights for both through gradient descent and Backpropagation Through Time

- Recurrent Neural Networks suffer from short-term memory.
- If a sequence is long enough, they'll have a hard time carrying information from earlier time steps to later ones.
- So if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning.

- During back propagation, recurrent neural networks suffer from the vanishing gradient problem.
- Gradients are values used to update a neural networks weights.
- The vanishing gradient problem is when the gradient shrinks as it back propagates through time.
- If a gradient value becomes extremely small, it doesn’t contribute too much learning.

- https://www.youtube.com/watch?v=S0XFd0VMFss


o
V
W
s
U
x
Unfold
o_{t-1}
o_t
o_{t+1}
s_{t-1}
s_t
s_{t+1}
x_{t-1}
x_t
x_{t+1}


h_t
A
x_t
=
h_0
h_1
h_2
h_t
A
A
A
A
x_0
x_1
x_2
...
x_t

BPTT(Backpropagation Through Time)

- In Backpropagation Through Time it is required to do the conceptualization of unrolling, since the error of a given timestep depends on the previous timestep.
- Within BPTT the error is back-propagated from the last to the first timestep, while unrolling all the timesteps.
- This allows calculating the error for each timestep, which allows updating the weights.
- Note that BPTT can be computationally expensive when you have a high number of timesteps.

Two issues of standard RNN's

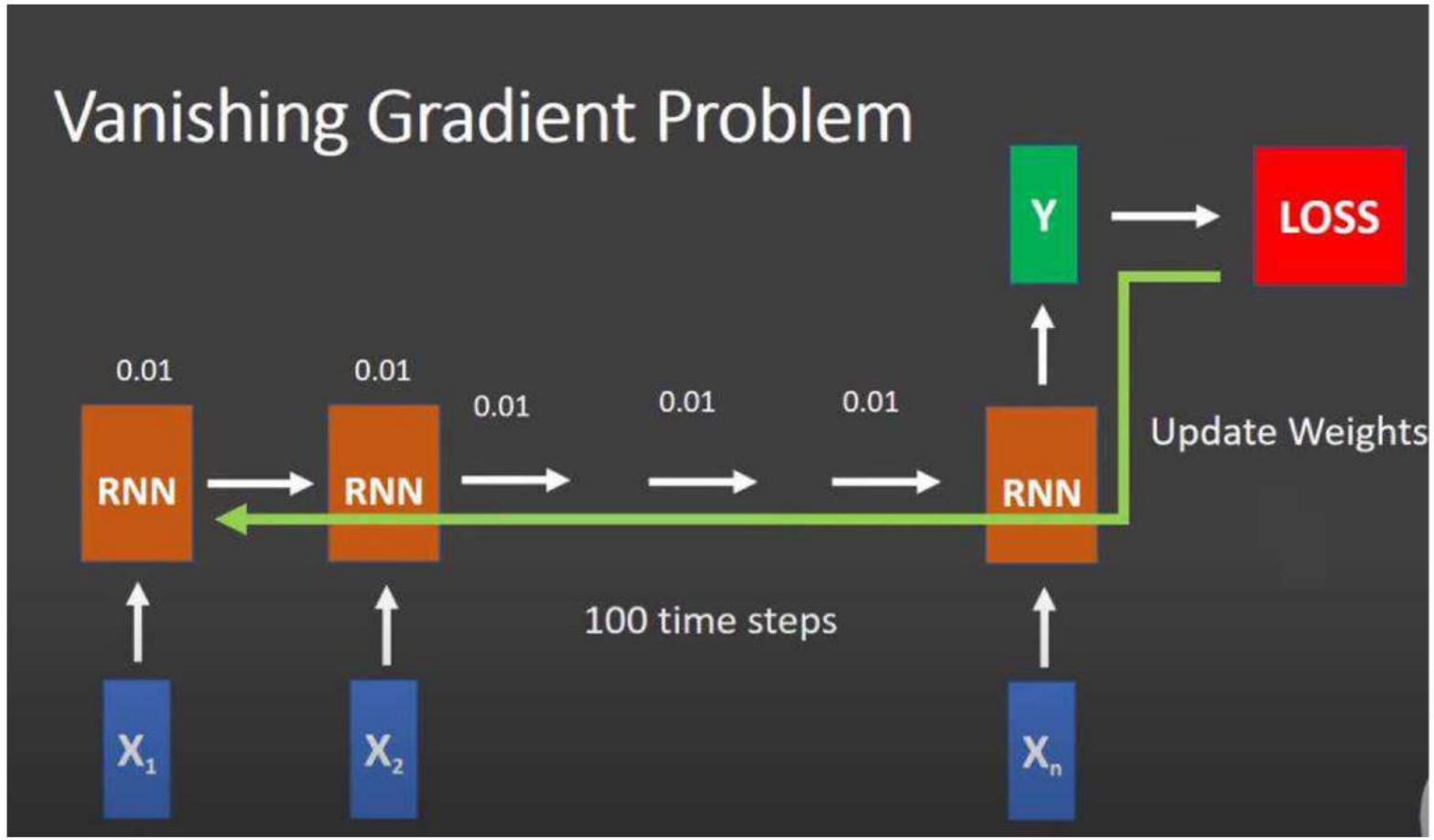
- **Exploding Gradients**

  We speak of "Exploding Gradients" when the algorithm assigns simply high importance to the weights, without much reason. But fortunately, this problem can be easily solved if you truncate or squash the gradients. Clipping the gradient is the solution.

- **Vanishing Gradients**

  During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weights. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn't contribute too much learning.

  This was a major problem in the 1990s and much harder to solve than the exploding gradients. Fortunately, it was solved through the concept of LSTM by Sepp Hochreiter and Juergen Schmidhuber.


Vanishing Gradient Problem
Y
LOSS
0.01
0.01
0.01
0.01
0.01
RNN
RNN
RNN
Update Weights
100 time steps
X_1
X_2
X_n

- **In case of exploding gradient**, you can:

- stop backpropagating after a certain point, which is usually not optimal because not all of the weights get updated

- penalize or artificially reduce gradient

- put a maximum limit on a gradient(clip)

- **In case of vanishing gradient**, you can:

- initialize weights so that the potential for vanishing gradient is minimized

- have Long Short-Term Memory Networks (LSTMs)

- LSTMs are considered to be the go-to network for implementing RNNs
- Skip connection

LSTM (Long Short Term Memory)

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Drawback of RNN

- Recurrent Neural Networks suffer from short-term memory.
- If a sequence is long enough, they'll have a hard time carrying information from earlier time steps to later ones.
- So if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning.
- LSTM's and GRU's were created as the solution to short-term memory. They have internal mechanisms called gates that can regulate the flow of information.

LSTM

- These gates can learn which data in a sequence is important to keep or throw away.
- By doing that, it can pass relevant information down the long chain of sequences to make predictions.
- Almost all state of the art results based on recurrent neural networks are achieved with these two networks.
- LSTM’s and GRU’s can be found in speech recognition, speech synthesis, and text generation. You can even use them to generate captions for videos.

Example

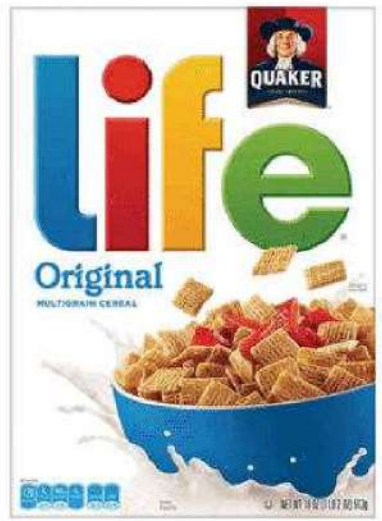
Customers Review 2,491

Thanos

September 2018

Verified Purchase

Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!



A Box of Cereal

$3.99

Customers Review 2,491

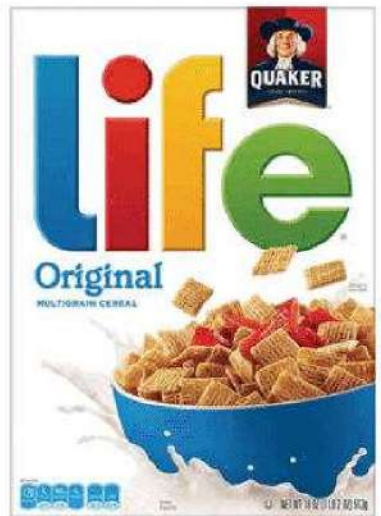
Thanos

September 2018

Verified Purchase

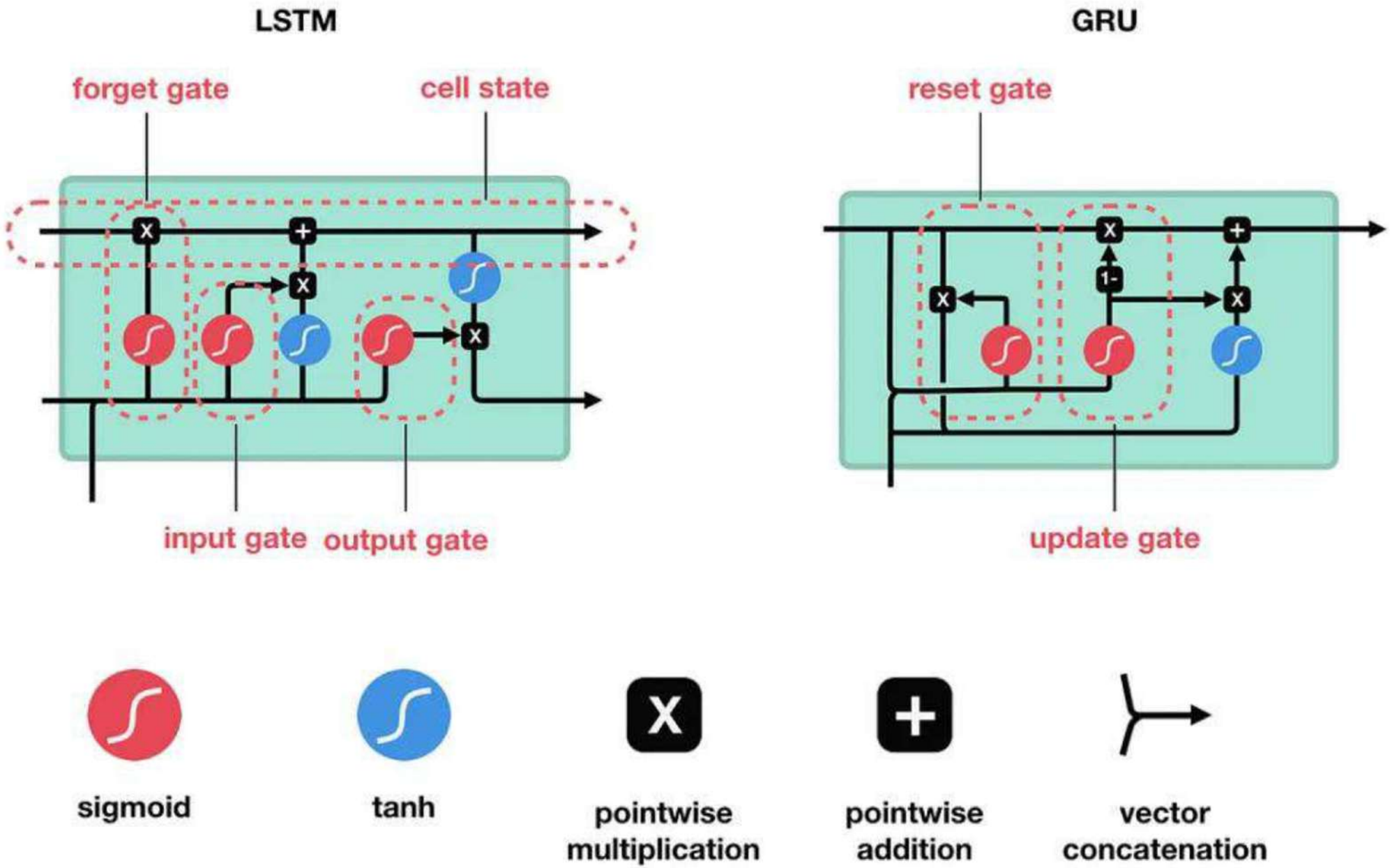
Amazing!
perfectly balanced breakfast.

will definitely be buying again!



A Box of Cereal
$3.99

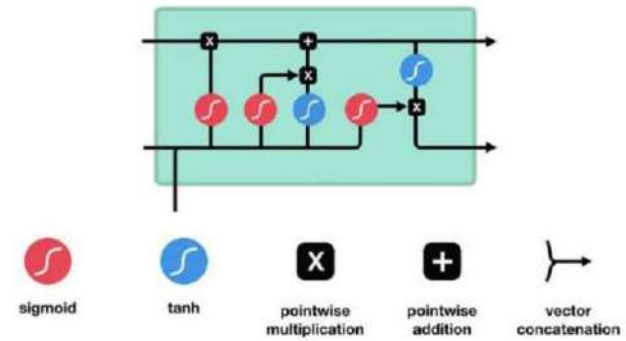
And that is essentially what an LSTM or GRU does. It can learn to keep only relevant information to make predictions, and forget non relevant data. In this case, the words you remembered made you judge that it was good.


LSTM
forget gate
cell state
input gate
output gate
GRU
reset gate
update gate
1-
sigmoid
tanh
pointwise multiplication
pointwise addition
vector concatenation

LSTM

Forget gate

- This gate decides what information should be thrown away or kept.
- Information from the previous hidden state and information from the current input is passed through the sigmoid function.
- Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.



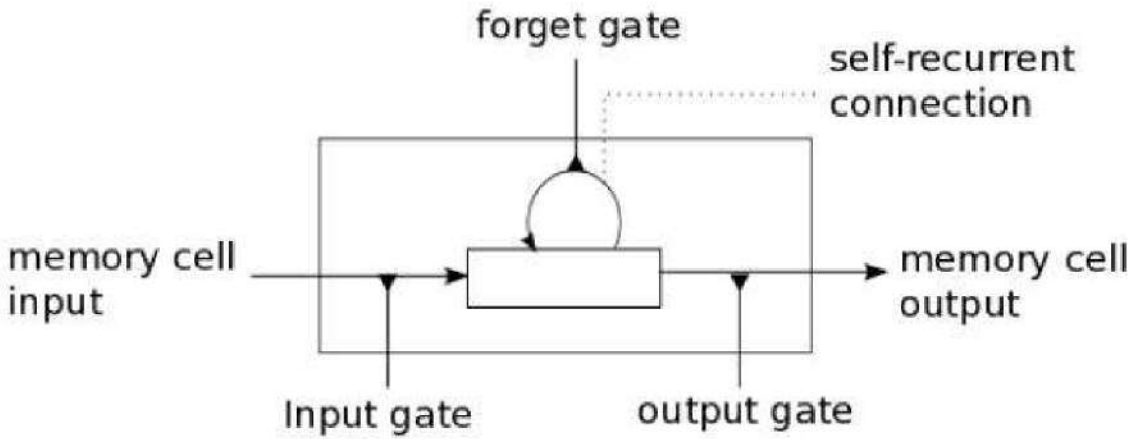
Input gate

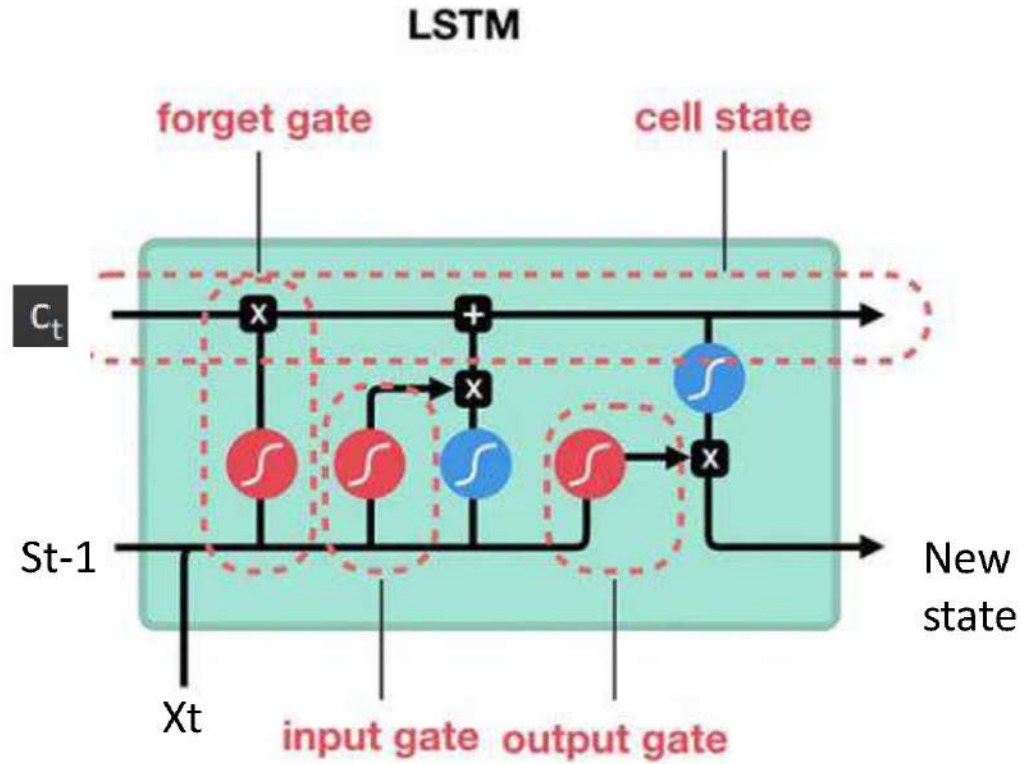
- To update the cell state, we have the input gate.
- First, we pass the previous hidden state and current input into a sigmoid function.
- That decides which values will be updated by transforming the values to be between 0 and 1.
- 0 means not important, and 1 means important.
- You also pass the hidden state and current input into the tanh function to squish values between -1 and 1 to help regulate the network.
- Then you multiply the tanh output with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output.

Output gate

- The output gate decides what the next hidden state should be.
- hidden state contains information on previous inputs.
- The hidden state is also used for predictions. First, we pass the previous hidden state and the current input into a sigmoid function.
- Then we pass the newly modified cell state to the tanh function.
- We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry.
- The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

Basic LSTM





$f_t = \sigma(W_f S_{t-1} + W_f X_t)$ - Forget Gate

$i_t = \sigma(W_i S_{t-1} + W_i X_t)$ - Input Gate

$o_t = \sigma(W_o S_{t-1} + W_o X_t)$ - Output Gate

$\tilde{C}_t = tanh(W_c S_{t-1} + W_c X_t)$

$c_t = (I_t * \tilde{C}_t) + (f_t * c_{t-1})$ - Cell State

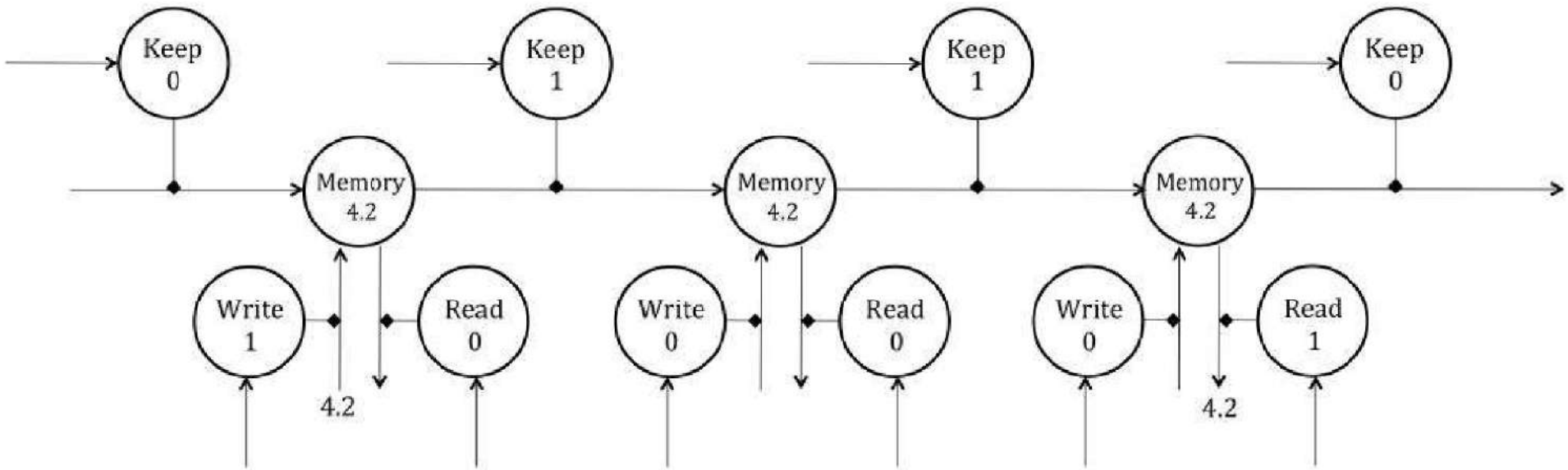
$h_t = o_t * tanh(c_t)$ – New State

Core Concept

- The core concept of LSTM's are the cell state, and it's various gates.
- The cell state act as a transport highway that transfers relative information all the way down the sequence chain. You can think of it as the "memory" of the network.
- The cell state, in theory, can carry relevant information throughout the processing of the sequence.
- So even information from the earlier time steps can make it's way to later time steps, reducing the effects of short-term memory. As the cell state goes on its journey, information get's added or removed to the cell state via gates.
- The gates are different neural networks that decide which information is allowed on the cell state. The gates can learn what information is relevant to keep or forget during training.

Unrolling LSTM through time

If the output of the **forget gate (keep)** is 1, the information is kept in the cell state. The new cell state and the new hidden is then carried over to the next time step. To review, the Forget **gate decides** what is **relevant** to **keep** from prior steps. The **input gate decides** what information is **relevant** to add from the **current** step. The output **gate** determines what the next hidden state should be.



It is these gates and the consistent dataflow called the Constant Error Carrousel (CEC) that keep each cell stable (neither exploding nor vanishing)

- when we move from **RNN** to **LSTM** (Long Short-Term Memory), we are introducing more & more controlling knobs, which control the flow and mixing of Inputs as per trained Weights.
- So, **LSTM** gives us the most Control-ability and thus, **Better** Results.
- But also comes with more Complexity and Operating Cost

- https://www.youtube.com/watch?v=UNmqTiOnRfg
- https://www.youtube.com/watch?v=WCUNPb-5EYI
- https://www.youtube.com/watch?v=S0XFd0VMFss
- https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5
- https://www.superdatascience.com/blogs/the-ultimate-guide-to-recurrent-neural-networks-rnn
- http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Machine Learning (19CSE305)

Markov Models, HMM

Dr. Peeta Basa Pati
Ms. Priyanka V
Department of Computer Science & Engineering,
Amrita School of Engineering, Bengaluru

Topics

- States and transition probabilities
- Markov Models
- HMM's

A very good reference book on this module:

Pattern Recognition – An Introduction by **Susheela Devi & Narasimha Murthy**

Images used in these slides are taken from multiple sources. All authors are acknowledged.

Intro

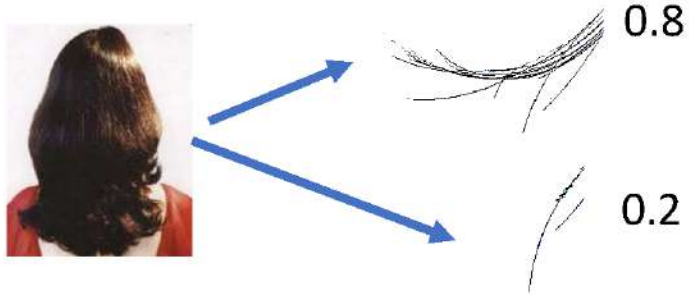
- Classifiers so far – feature vectors are unrelated
- What if they are?
  - If it has rained yesterday, its likely to rain today
  - In IPL if a team has been winning, like to win in the next match
  - If a student has been scoring good grades, likely to score good grade this semester
  - Covid cases are decreasing for last 1 week, likely to decrease today
  - A restaurant has been serving good food in your last 3 visits, likely to serve good food now
  - A person has allergy during Diwali time last 3 years, likely to have allergy this time
- Past events determine the future in many scenarios
- We could leverage the underlying pattern to decide better

Observable & Hidden states

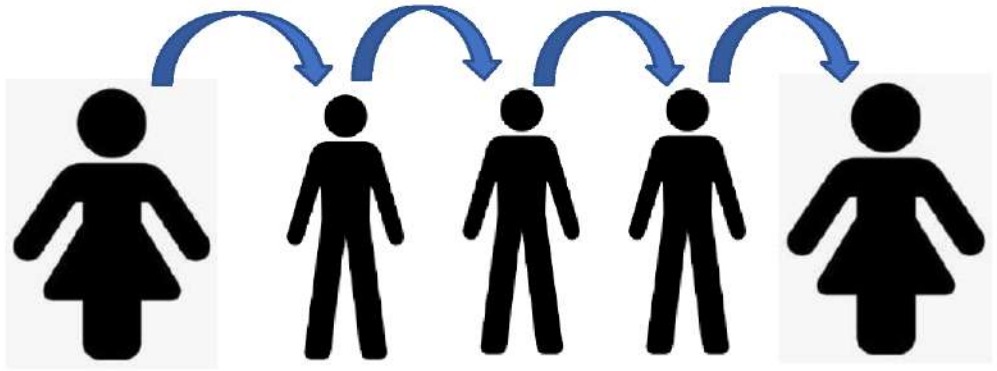
Assume that we measure a person's hair length and try to predict the gender.

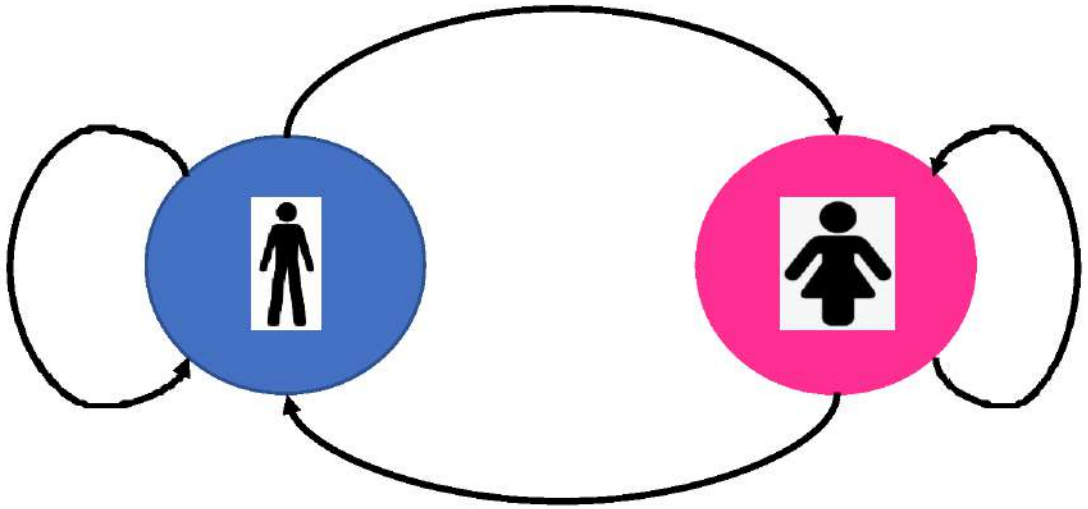
Prior probabilities are probabilities of states. Ex: probability of a person being male or female. Male = 0.55; Female = 0.45

Class conditional densities are observations associated with a class. Ex: hair being short given the person is a female. Male = {0.95, 0.05}; Female = {0.2, 0.8}



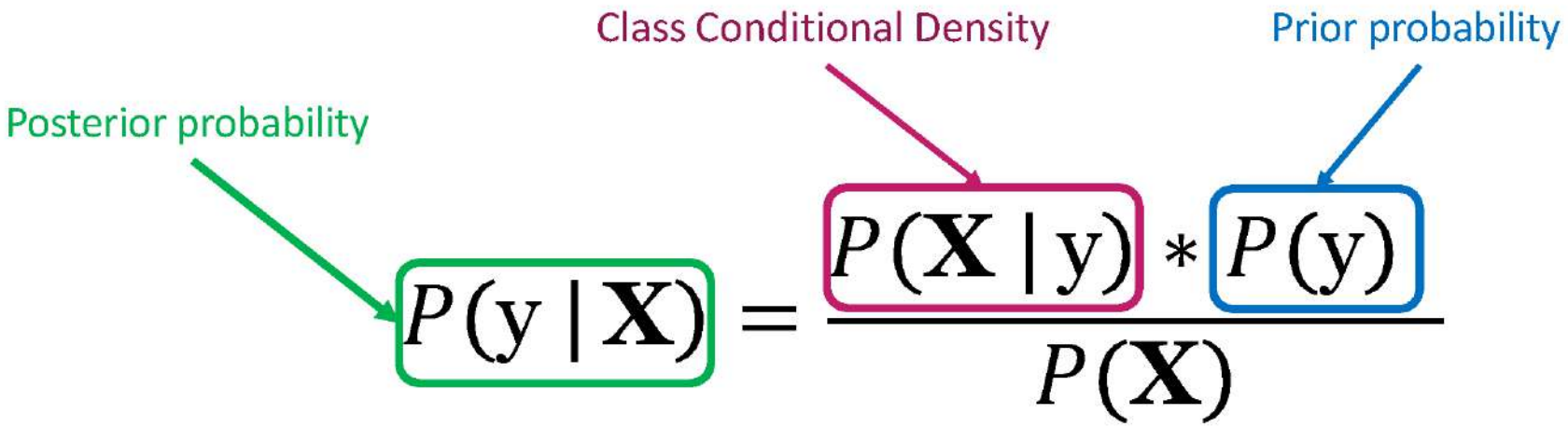
State Transition Probabilities





	Female	Male
Female	p_{ff}	p_{fm}
Male	p_{mf}	p_{mm}

Bayes Classifier



$$y_{MAP} = \underset{y_j}{argmax} \{P(y_j \mid \mathbf{X})\}$$

Maximum a posteriori probability

y_j are different classes

Markov Process

$\{s_1, s_2, \ldots, s_N\}$ Set of possible states: Male/ Female, Rain / Dry

$s_{i1}, s_{i2}, \ldots, s_{ik}, \ldots$ Sequence of states for i^{th} observation

$$P(s_{ik} \mid s_{i1}, s_{i2}, \ldots, s_{ik-1}) = P(s_{ik} \mid s_{ik-1})$$

Markov process (first order)

probability of a state depends only on the previous state.

$\pi_i = P(s_i)$ Initial / Prior Probability

$a_{ij} = P(s_i \mid s_j)$ State Transition Probability

Significance of i variation

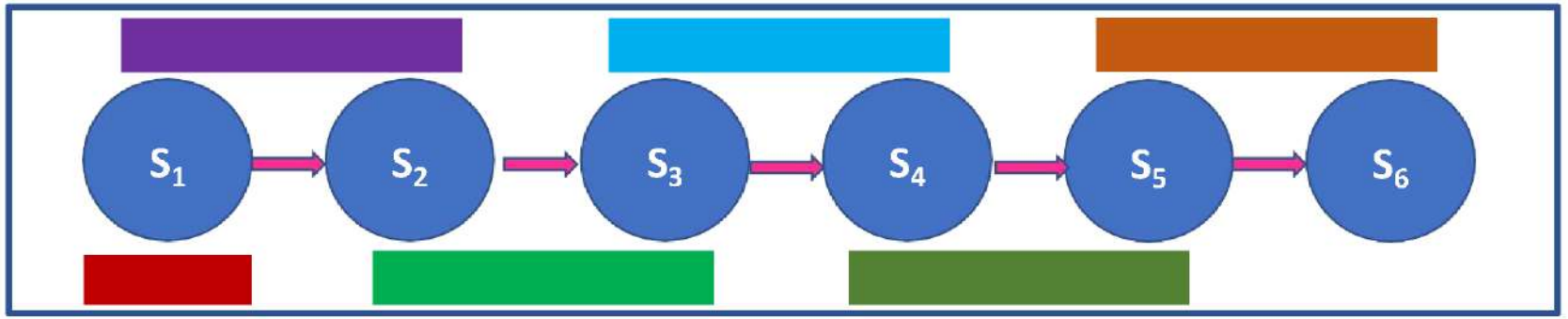
Probabilities may change based on time or place

Sequence Probability

$\{s_1, s \ , \ldots$ Set of possible states: Male/ Female, Rain / Dry

$s_{i1}, s_{i2}, \ldots, s_{ik}$ Sequence of states for ith observation

$$P(s_{ik} \mid s_{i1}, s_{i2}, \ldots, s_{ik-1}) = P(s_{ik} \mid s_{ik-1})$$ Markov process (first order)



$$P(s_{i1}, s_{i2}, \ldots, s_{ik}) = P(s_{ik} \mid s_{i1}, s_{i2}, \ldots, s_{ik-1}) P(s_{i1}, s_{i2}, \ldots, s_{ik-1})$$

$$= P(s_{ik} \mid s_{ik-1}) P(s_{i1}, s_{i2}, \ldots, s_{ik-1}) = \ldots$$

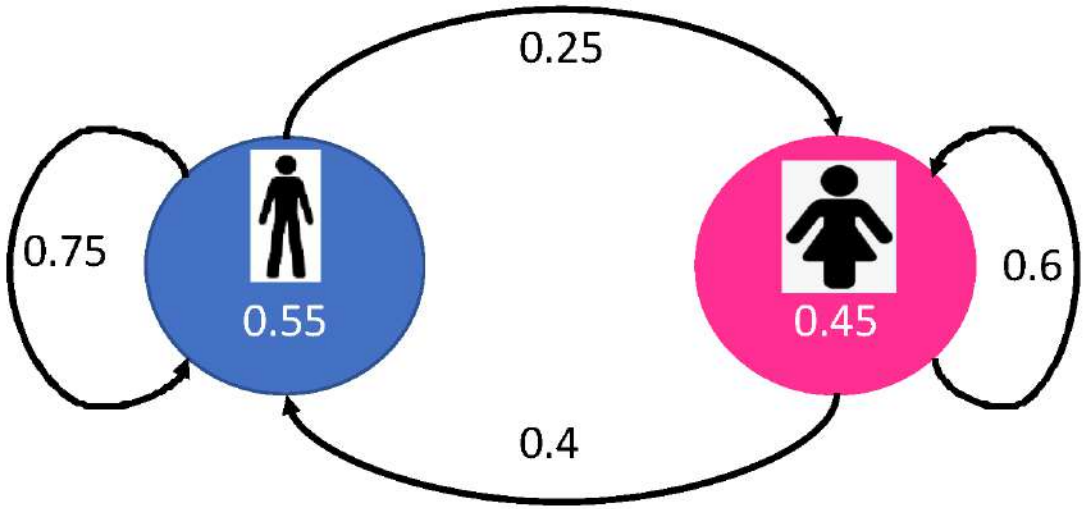
$$= P(s_{ik} \mid s_{ik-1}) P(s_{ik-1} \mid s_{ik-2}) \ldots P(s_{i2} \mid s_{i1}) P(s_{i1})$$

Example

What is the Probability for {Girl, Boy}?

P({Girl, Boy}) = P(Boy | Girl)* P(Girl)

= 0.4 * 0.45 = 0.18



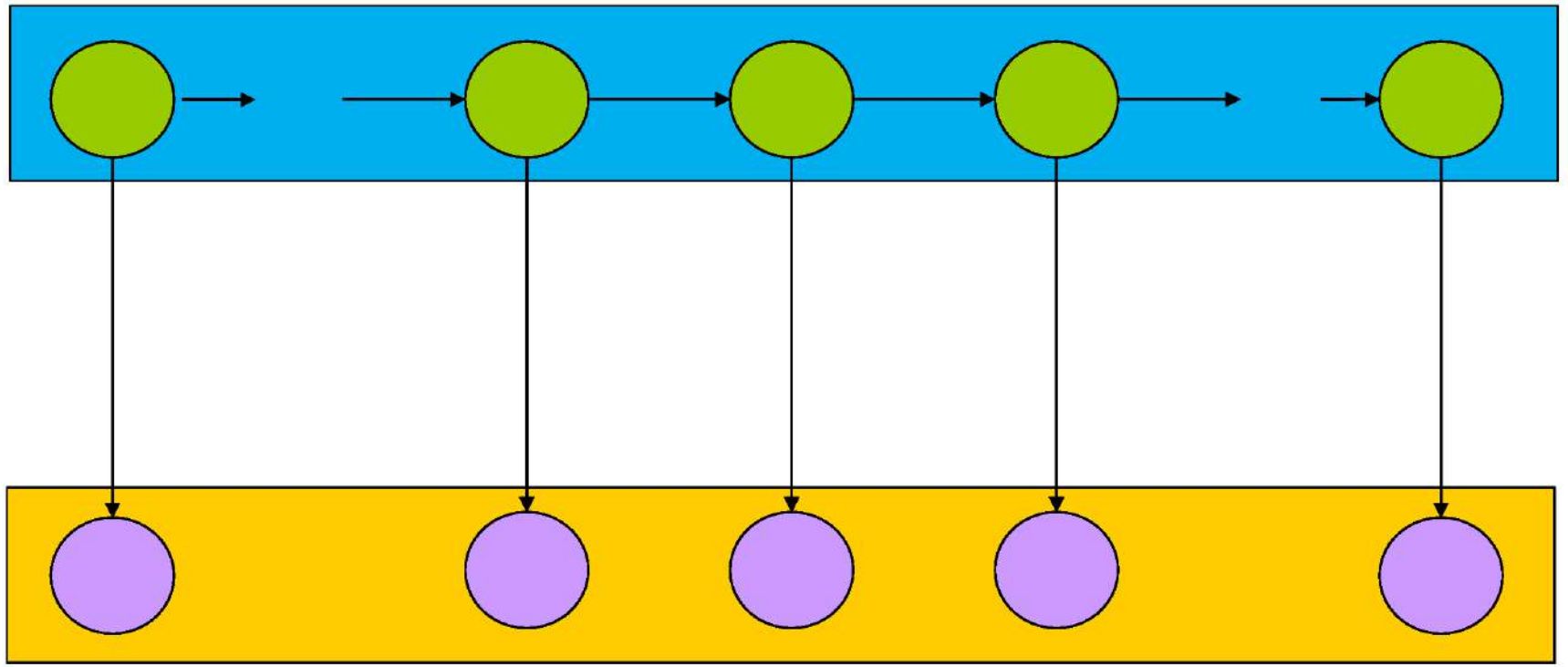
P({Boy, Girl, Girl, Boy})??

P({Boy, Girl, Girl, Boy}) = P(Boy | {Girl, Girl, Boy}) * P(Girl | {Girl, Boy}) * P(Girl | Boy) * P(Boy)

= P(Boy | Girl) * P(Girl | Girl) * P(Girl | Boy) * P(Boy)

= 0.4 * 0.6 *0.25 * 0.55 = 0.033

Hidden Markov Model (HMM)



Green Circles are Hidden states

- Dependent only on the previous state
- "The past is independent of the future given the present."

Purple Circles are Observed states

- Dependent only on their corresponding hidden state

HMM definition

$\{s_1, s \ , \ldots$ Set of possible states: Male/ Female, Rain / Dry

$S_i = s_{i1}, s_{i2}, \ldots, s_i$ Sequence of states (i^{th}) for observations

$P(s_{ik} \mid s_{i1}, s_{i2}, \ldots, s_{ik-1}) = P(s_{ik} \mid s_{ik-1})$ Markov process (first order)

$\{v_1, v_2, \ldots, v_M\}$ Possible Observation variations (long / short hair)

Actual states are hidden and not observable

$O=o_1 \ldots o_K$ Observation Sequence

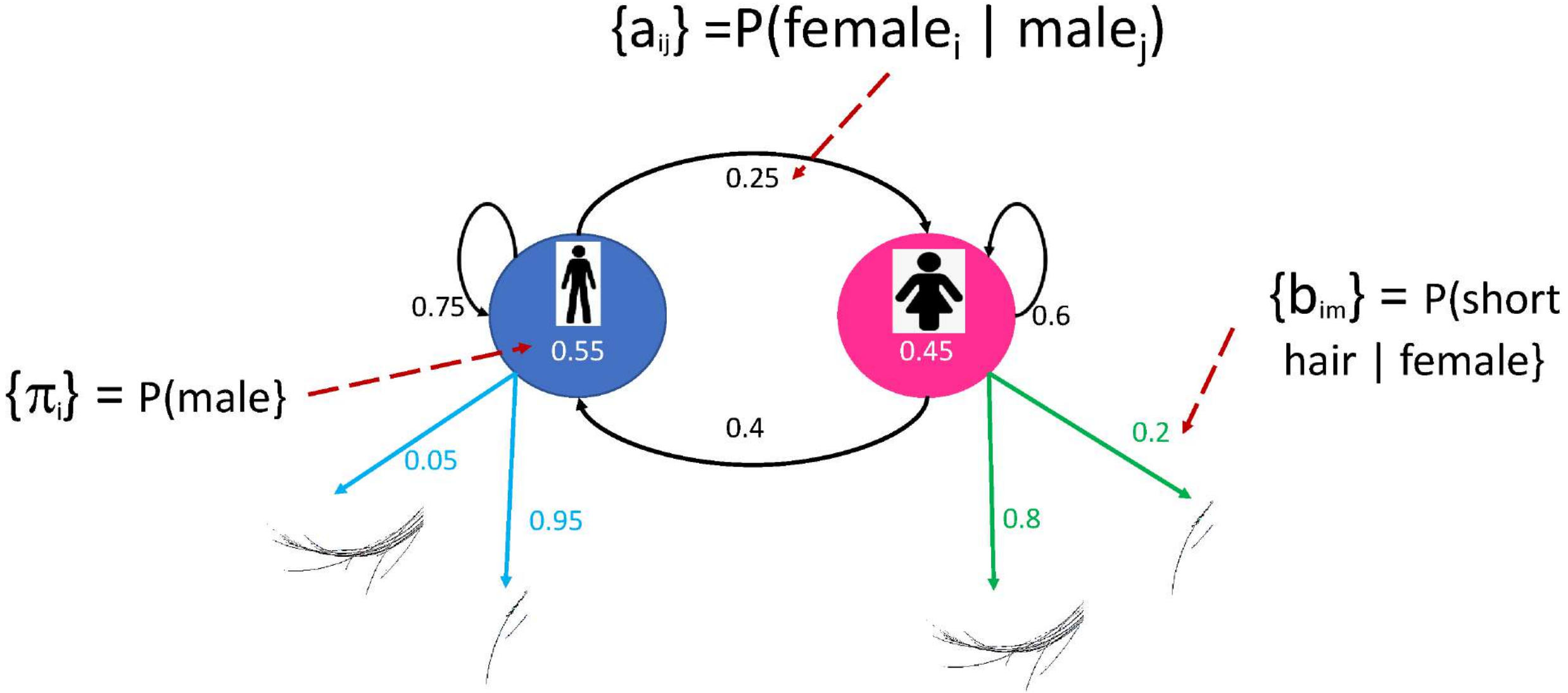
$A = \{a_{ij}\} \rightarrow$ state transition probabilities

$B = \{b_{im}\}$, $b_{im} = P(v_m \mid s_i) \rightarrow$ observation state probabilities

$\pi=\{\pi_i\}$, $\pi_i = P(s_i) \rightarrow$ initial state probabilities

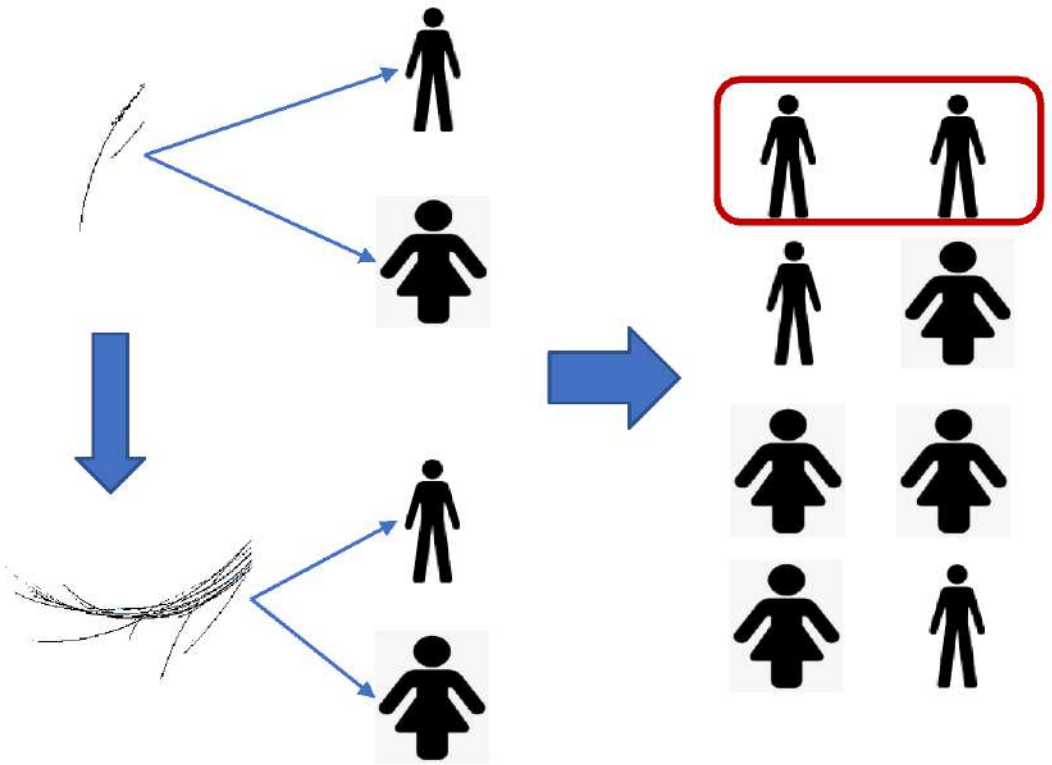
$M=(A, B, \pi) \rightarrow$ Model

HMM Probabilities



Example

We observed a short hair (SH) followed by a long hair (LH), what is the inference on the gender of people?

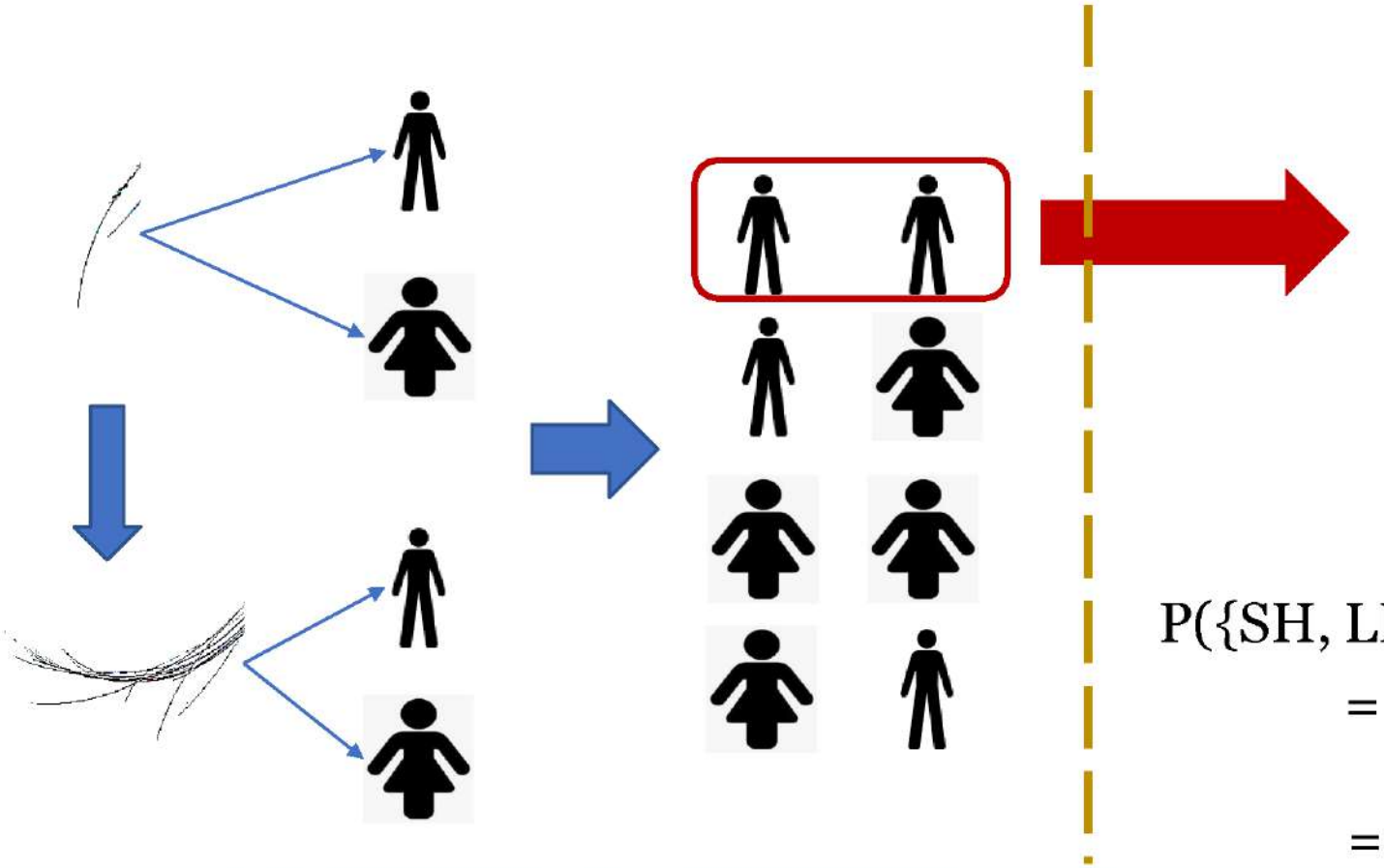


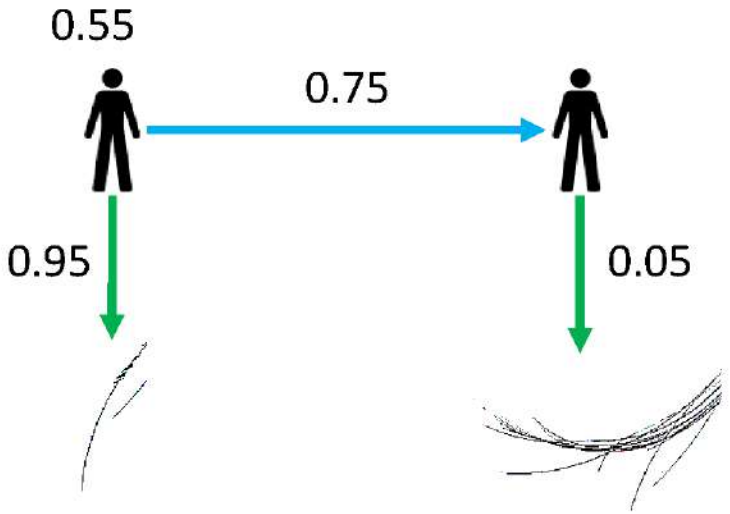
- Hidden states lead to observations – sex leads to observations on hair length
- Calculate the probability for observations from various hidden state combinations; identify the max value
  - The hidden states with max probability, are decided to have generated the observations
- Computer $P(O \mid Si, M)$

Decide $S_i = \underset{S_i}{\text{ARGMAX}}\ P(O \mid Si, M)$

Example

We observed a short hair (SH) followed by a long hair (LH), what is the inference on the gender of people?





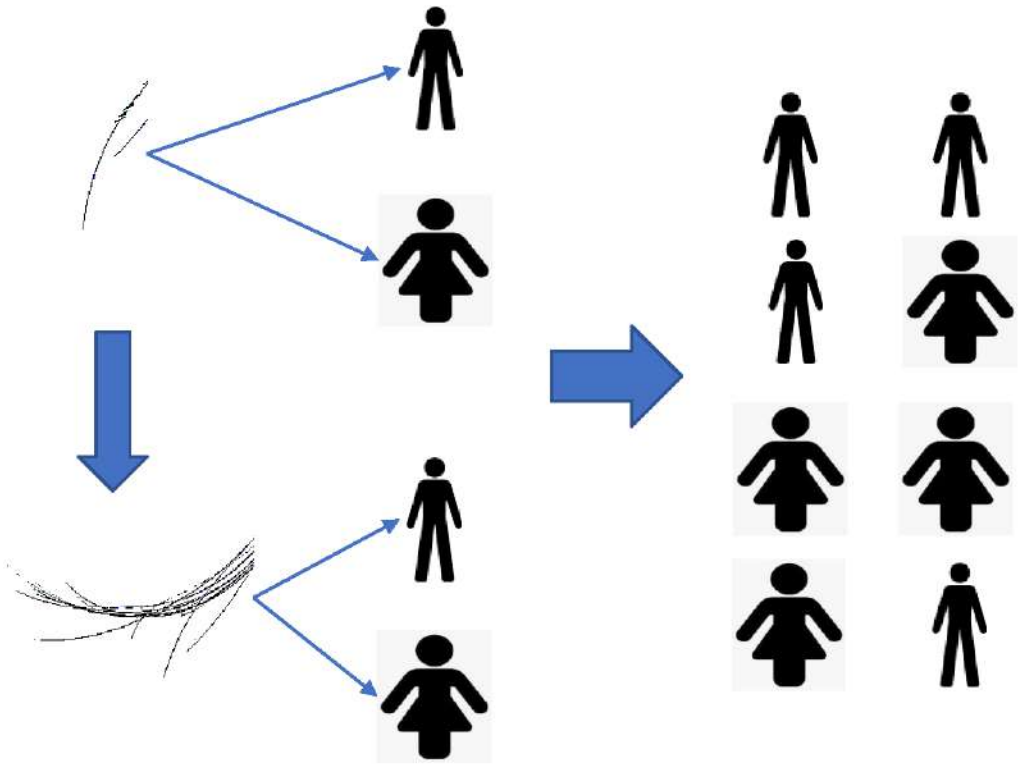
P({SH, LH},{Boy, Boy})

= P(SH | Boy) * P(Boy) * P (Boy | Boy) * P(LH | Boy)

=0.95 * 0.55 * 0.75 * 0.05 = 0.0196

Example

We observed a short hair followed by a long hair, what is the inference on the gender of people?



P({Boy, Boy}) = P(short hair | Boy) * P(Boy) * P (Boy | Boy) * P(long hair | Boy)

=0.95 * 0.55 * 0.75 * 0.05 = 0.0196

P({Boy, Girl}) = P(short hair | Boy) * P(Boy) * P (Girl | Boy) * P(long hair | Girl)

= 0.95 * 0.55 * 0.25 * 0.8 = 0.1045

P({Girl, Girl}) = P(short hair | Girl) * P(Girl) * P (Girl | Girl) * P(long hair | Girl)

= 0.2 * 0.45 * 0.6 * 0.8 = 0.0432

P({Girl, Boy}) = P(short hair | Girl) * P(Girl) * P (Boy | Girl) * P(long hair | Boy)

= 0.2 * 0.45 * 0.4 * 0.05 = 0.0018

References

- https://www.youtube.com/watch?v=kqSzLo9fenk
- https://cedar.buffalo.edu/~govind/CS661/Lec12.ppt
- https://nlp.stanford.edu/fsnlp/hmm-chap/blei-hmm-ch9.ppt
- Pattern Recognition – An Introduction by Susheela Devi & MNM
- Pattern Recognition – Duda, Hart & Stork

Thank you !!!!!