Introduction of AI:
Machine and Deep Learning

Presented By: Abdul Qayyum

Abdesslam BENZINOU, Kamal NASREDDINE, khawla bengaied

BLUE REVOLUTION

BIODIVERSITY UNDERESTIMATION IN OUR BLUE PLANET :
ARTIFICIAL INTELLIGENCE REVOLUTION
IN BENTHIC TAXONOMY

ÉCOLE NATIONALE D'INGÉNIEURS DE BREST

Lab-STICC

Ifremer

ISblue

Introduction of Machine Learning and Deep Learning

Outlines of Lecture:

In this lecture we will consider

- What is Artificial intelligence, Machine Learning ?
- Comparison of ML algorithms
- Feature Extraction and Scaling
- Applications of ML
- Introduction of Deep Learning
- Neural Network Basic
- Applications

Introduction

- AI and machine learning are often used interchangeably, especially in the realm of big data.

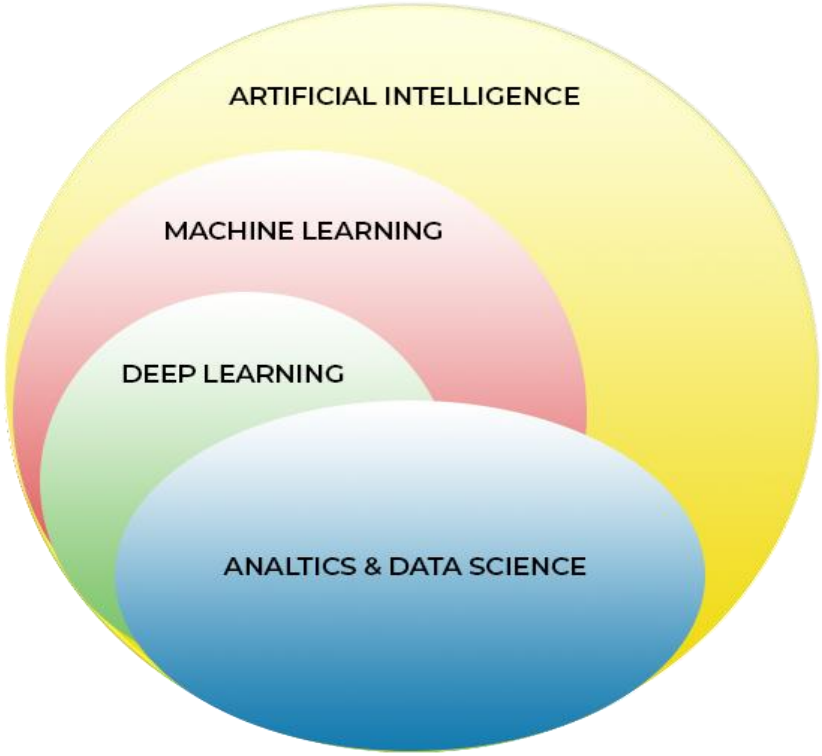
- **Artificial intelligence** is a broader concept than machine learning, which addresses the
use of computers to mimic the cognitive functions of humans.

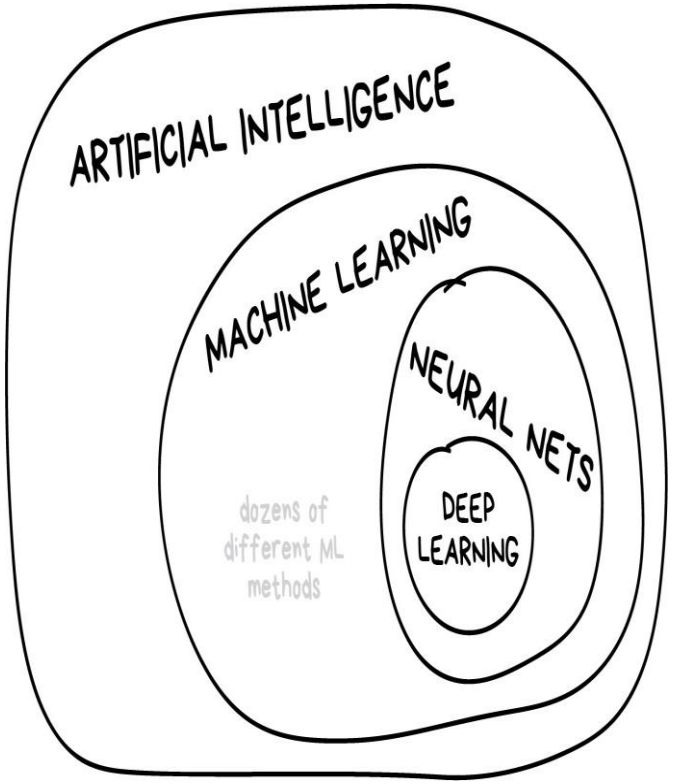
- When machines carry out tasks based on algorithms in an "intelligent" manner, that is AI.

- **Machine learning** is a subset of AI and focuses on the ability of machines to receive a set
of data and learn for themselves, changing algorithms as they learn more about the information
they are processing.

- **Deep learning** goes yet another level deeper and can be considered a subset of machine learning.

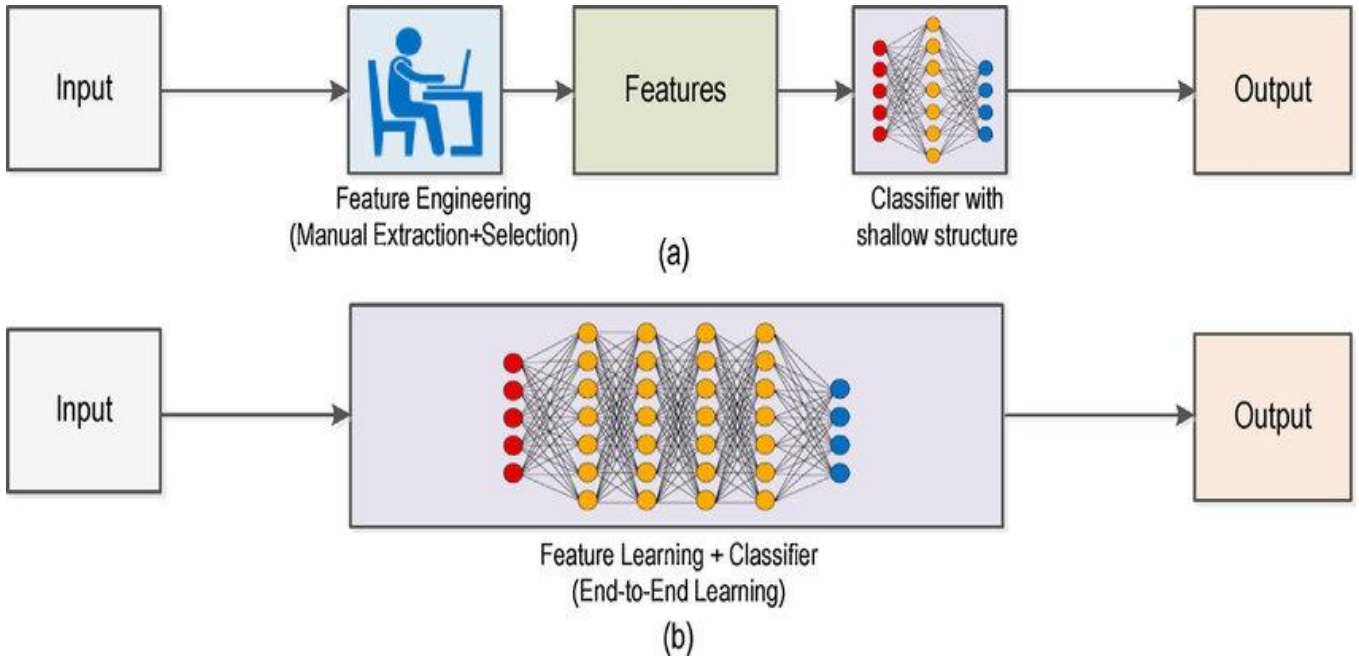
The concept of deep learning is sometimes just referred to as "deep neural networks,"
referring to the many layers involved. A neural network may only have a single layer of data,
while a deep neural network has two or more.





Source:
https://datascience.stackexchange.com/questions/16422/machine-learning-vs-deep-learning

Difference ML and DL

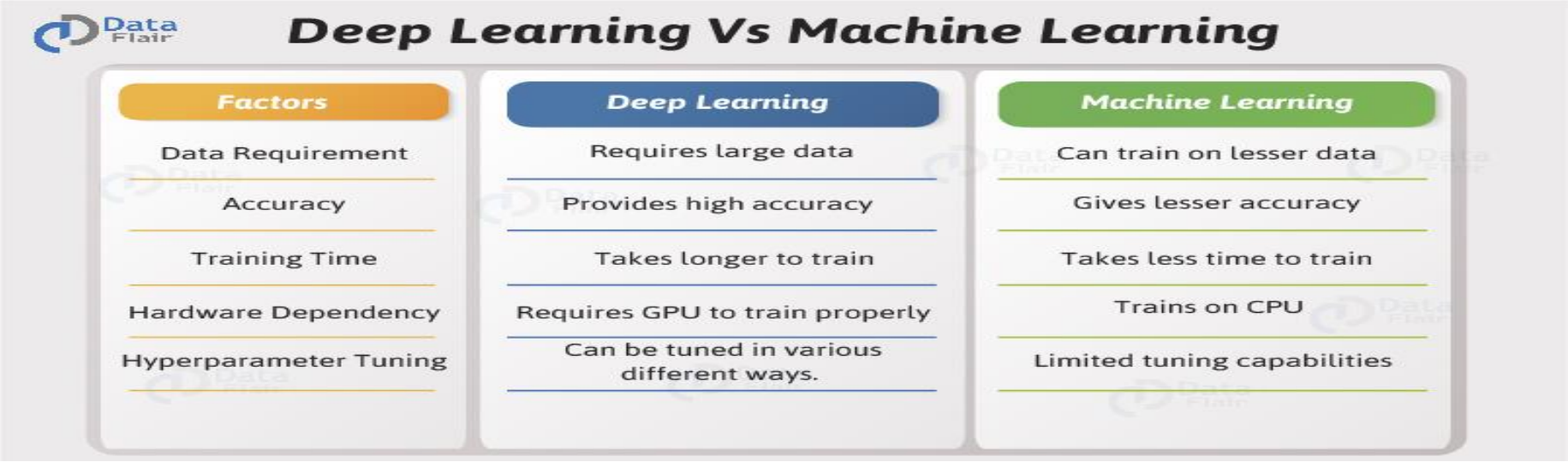


Machine Learning

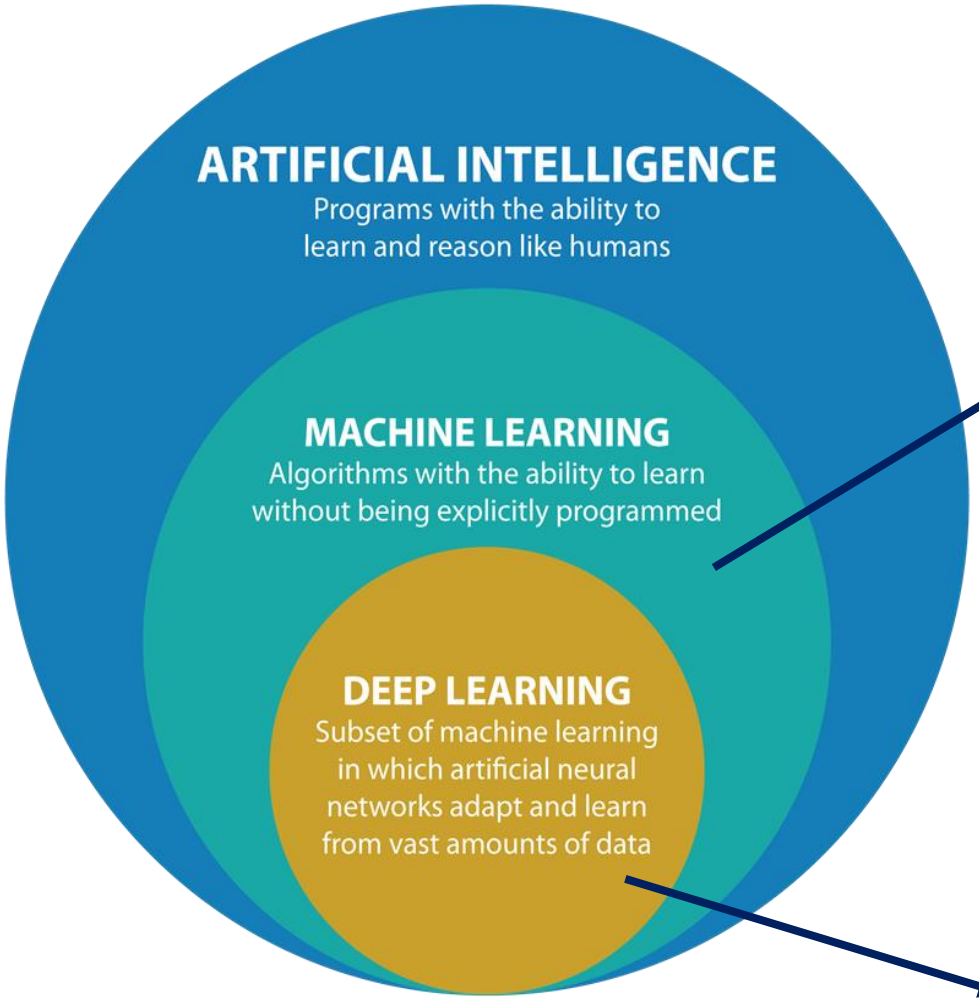
- Good results with small data sets
- Quick to train a model
- Need to try different features and classifiers to achieve best results
- Accuracy plateaus

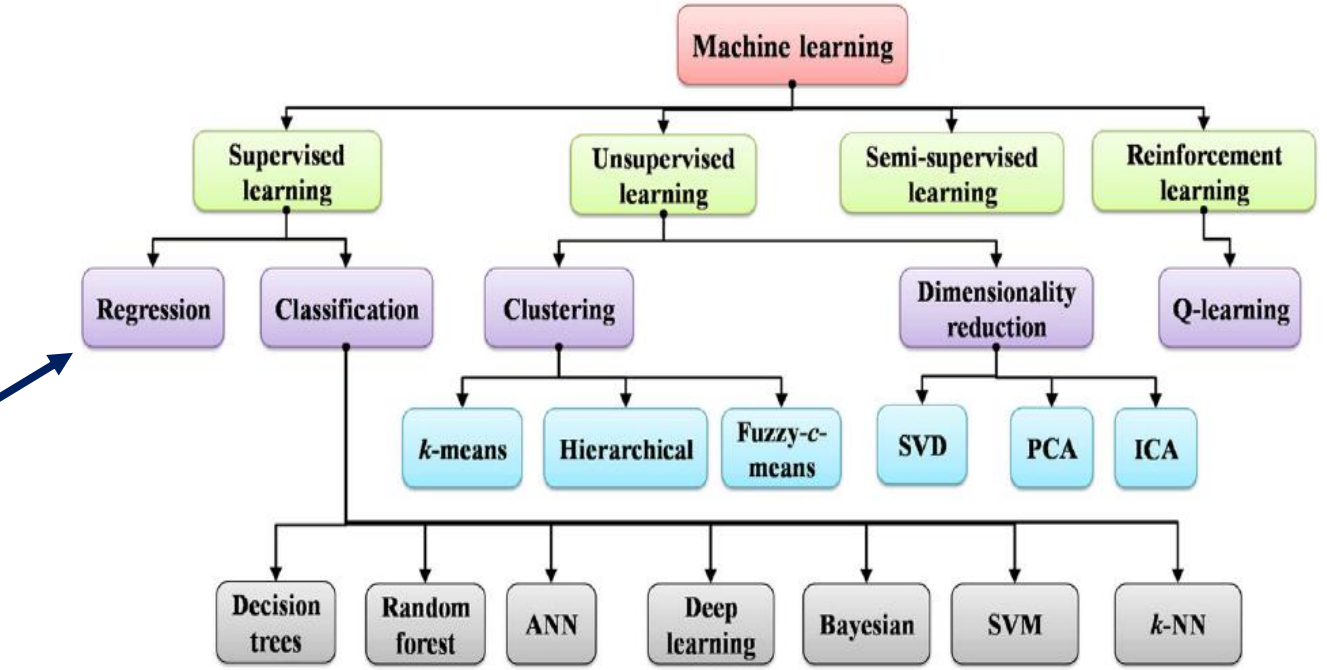
Deep Learning

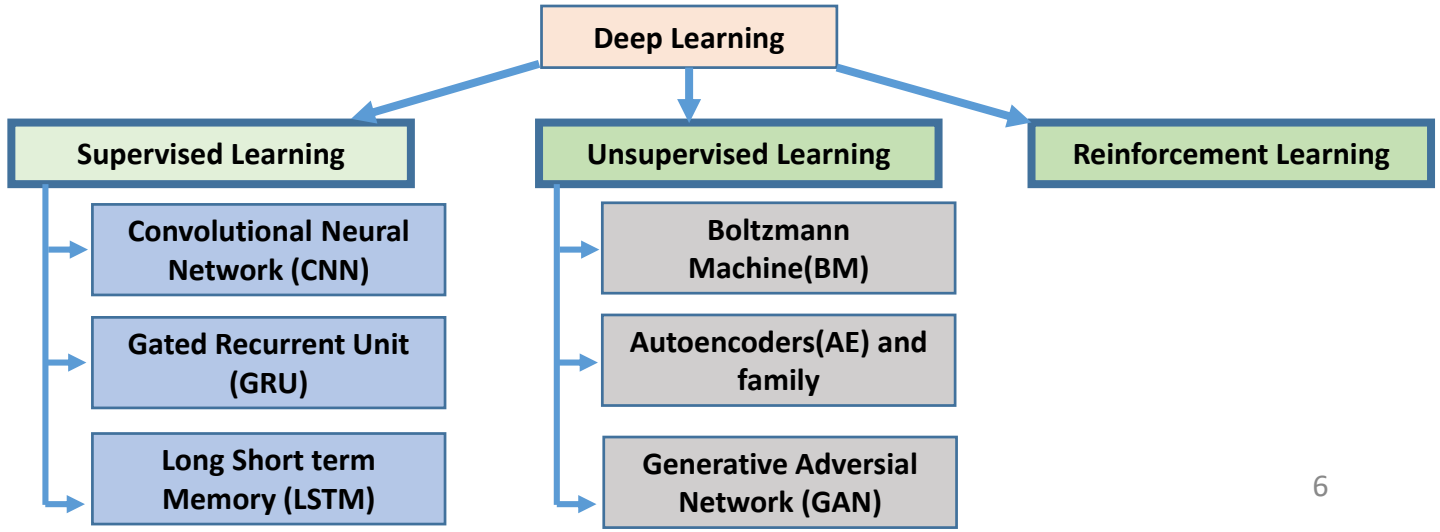
- Requires very large data sets
- Computationally intensive
- Learns features and classifiers automatically
- Accuracy is unlimited



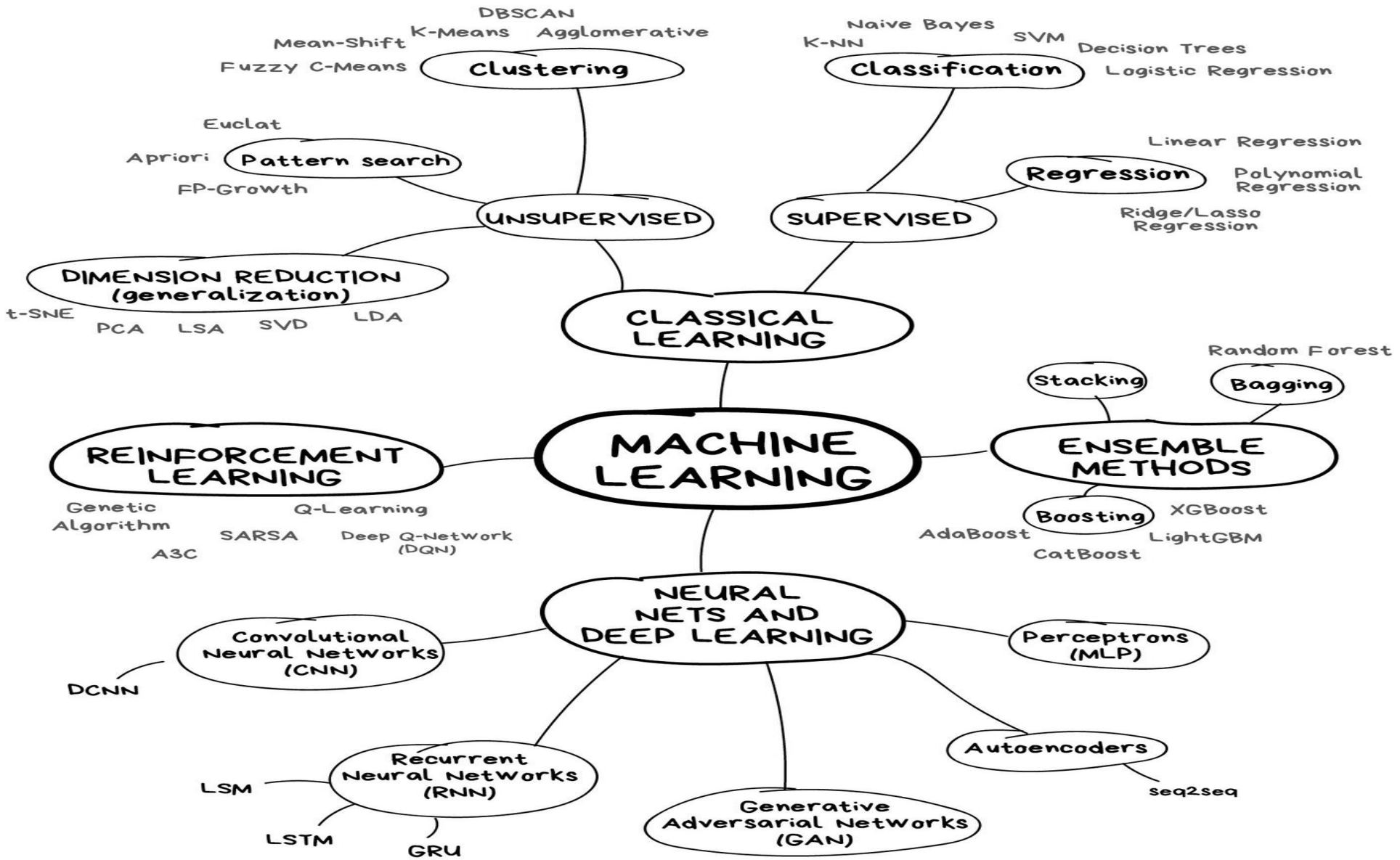
Algorithms of ML and DL



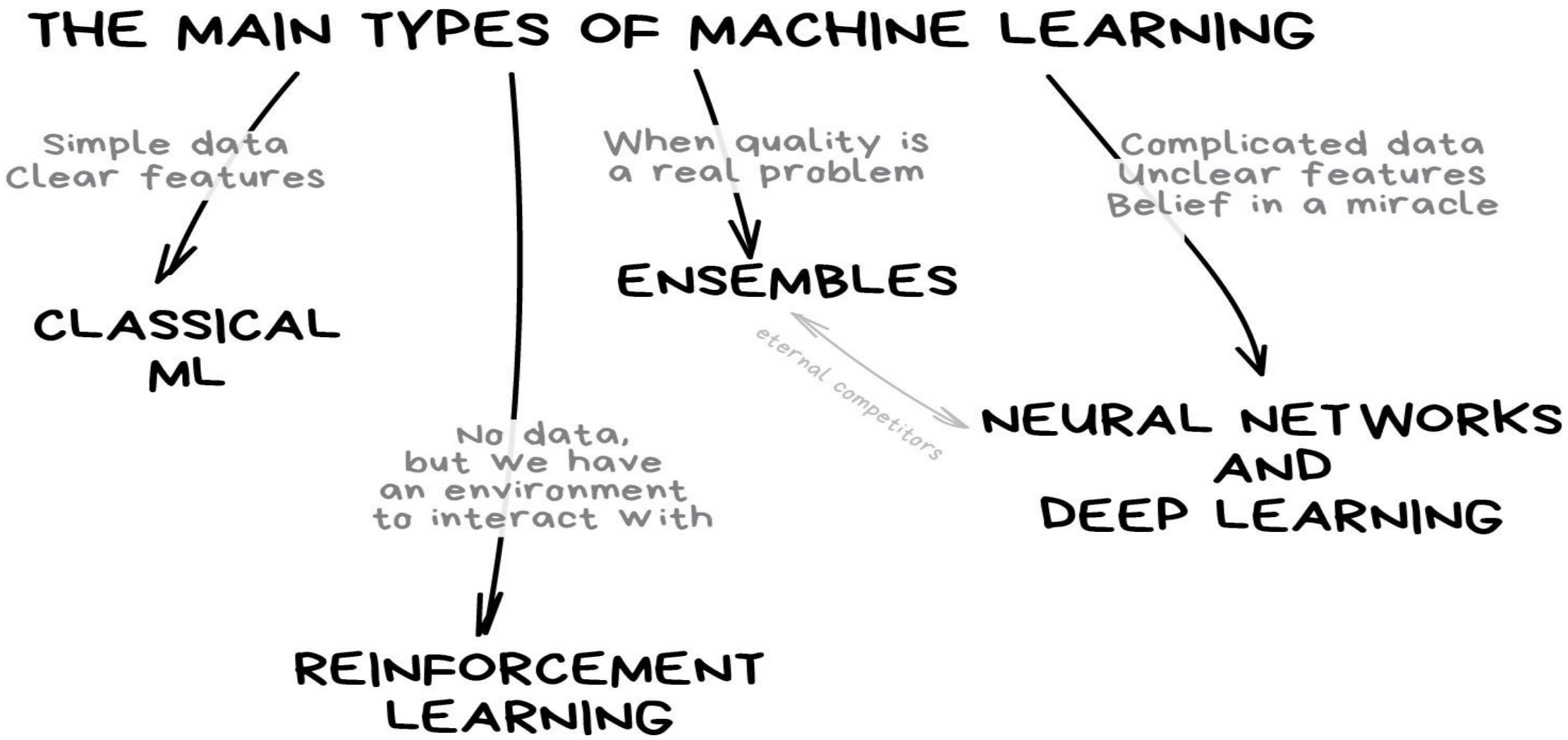




Machine Learning Tree

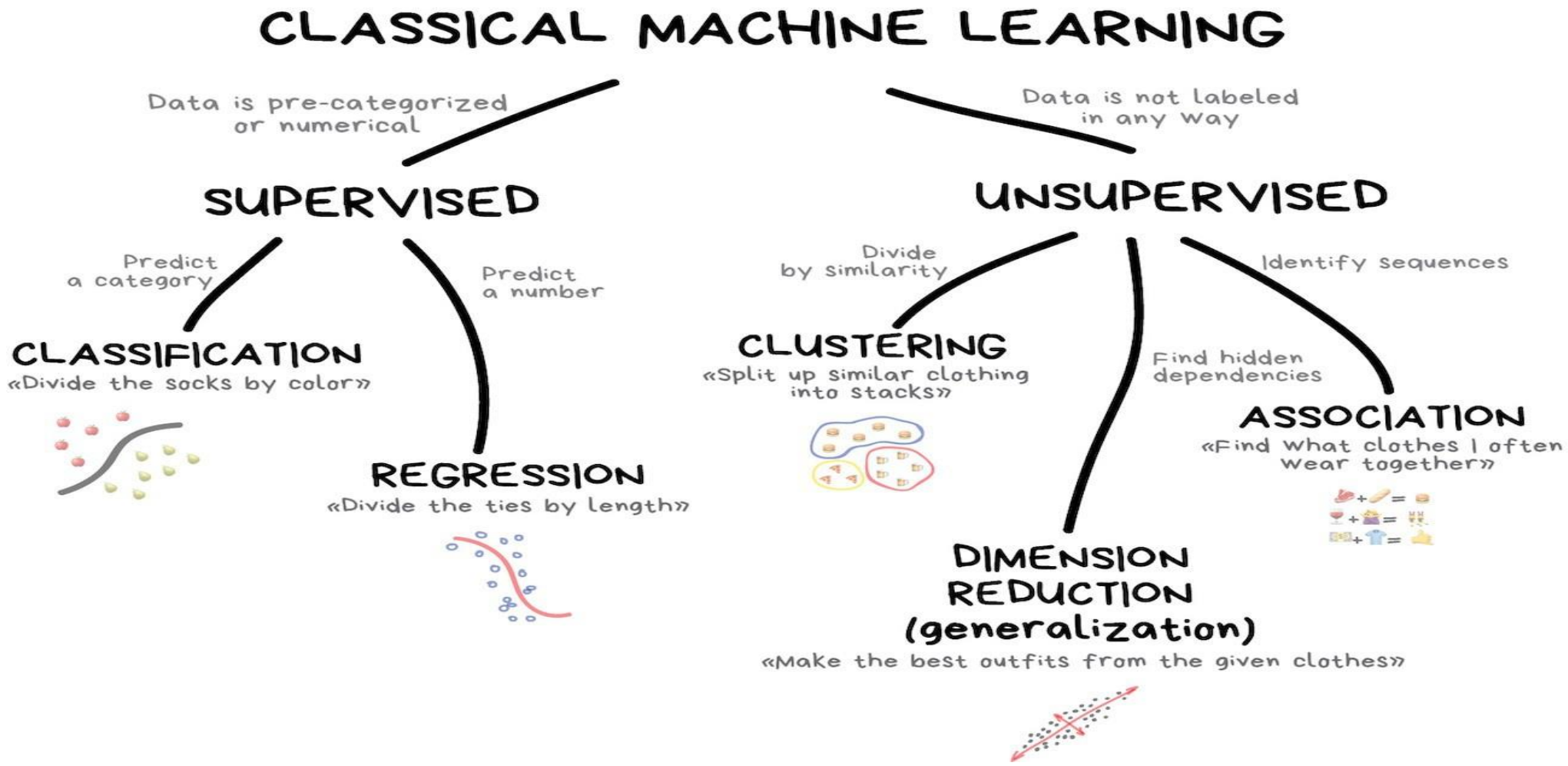


Types of Machine Learning



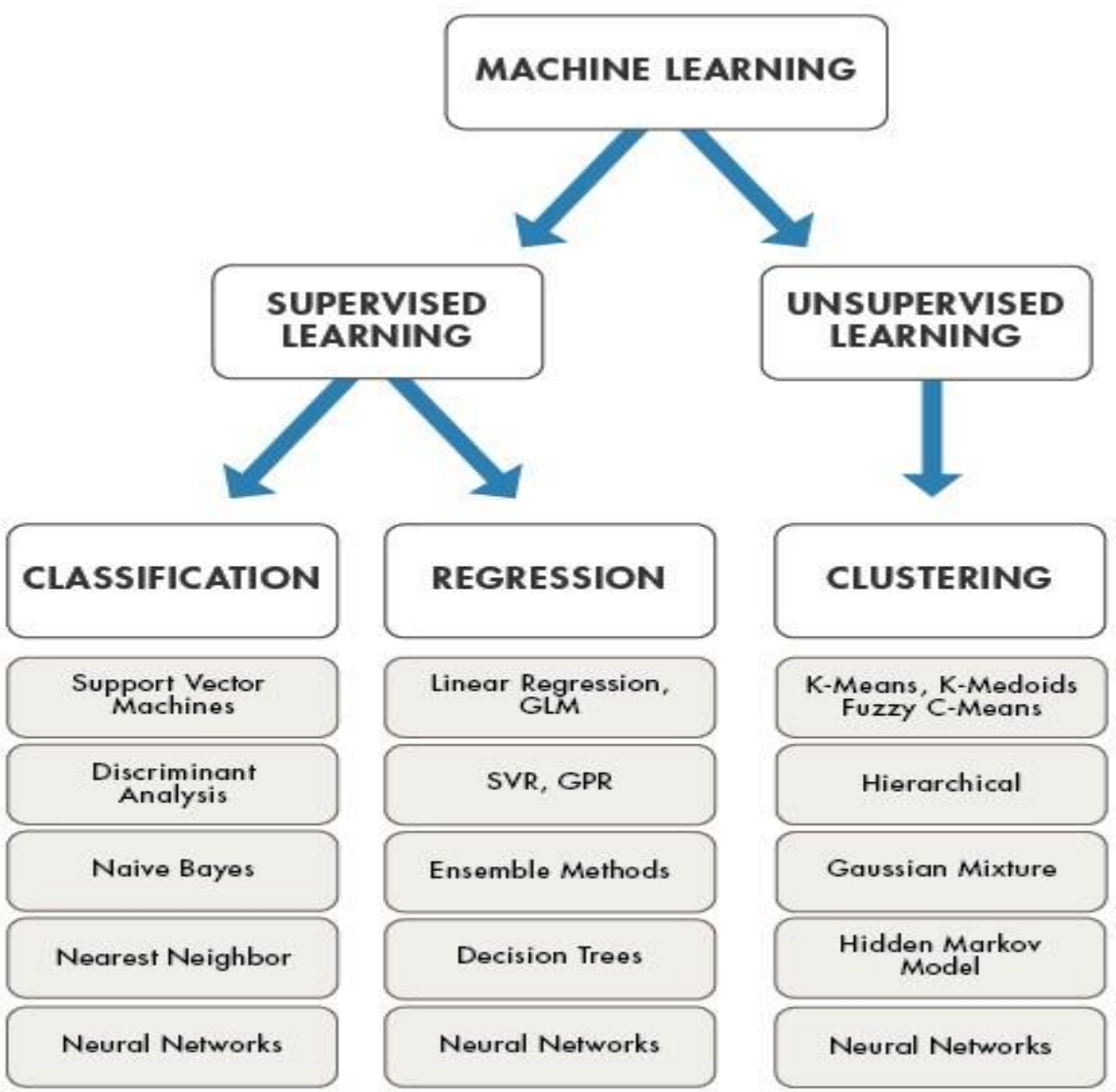
https://noeliagorod.com/2019/05/21/machine-learning-for-everyone-in-simple-words-with-real-world-examples-yes-again/

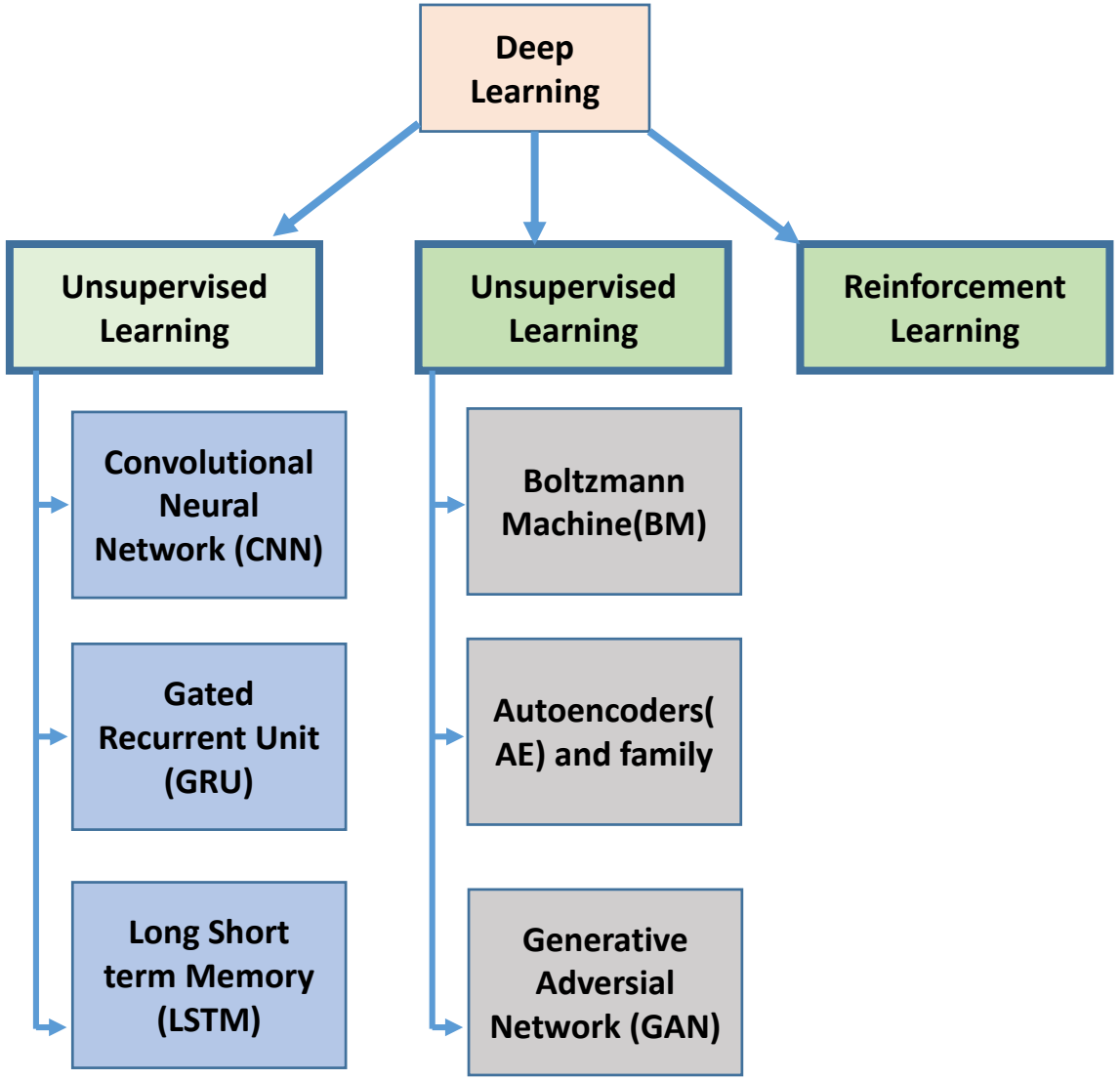
Classical Machine Learning



https://noeliagorod.com/2019/05/21/machine-learning-for-everyone-in-simple-words-with-real-world-examples-yes-again/

Supervised and Unsupervised Learning





Supervised Learning

Cat

Classification

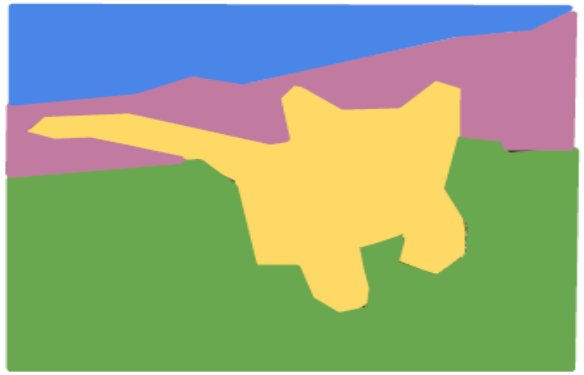
DOG, DOG, CAT

Object Detection

Data: (x,y)
x is data, y is label

Goal: Learn a function to map x→ y

Examples: Classification, Regression, Object detection, Semantic Segmentation, Image captioning



GRASS, CAT, TREE, SKY

Semantic Segmentation

A cat sitting on a suitcase on the floor

Image captioning

Source: http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture12.pdf

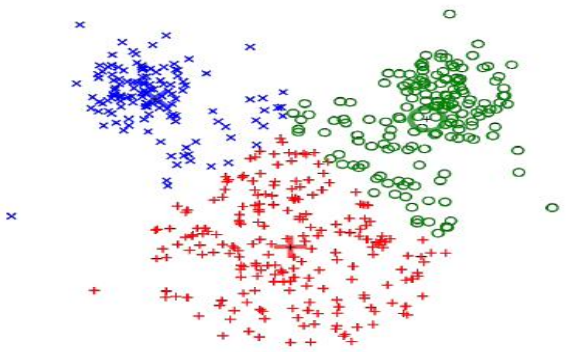
Unsupervised Learning

Data: x

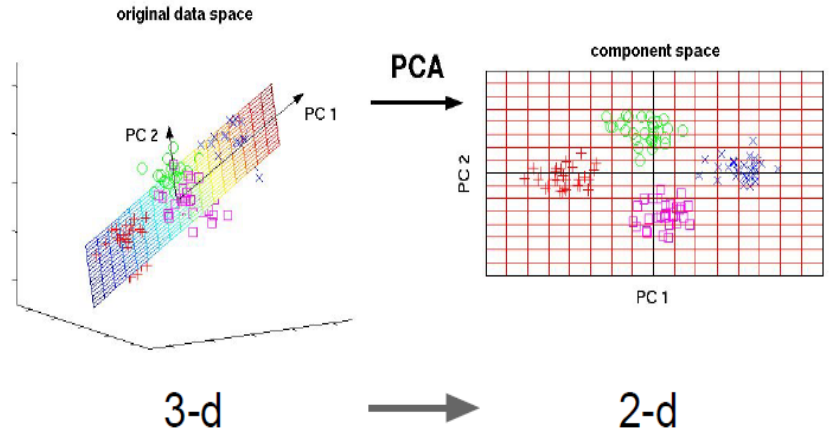
x is data, no label

Goal: Learn some underlying Hidden structure of the data

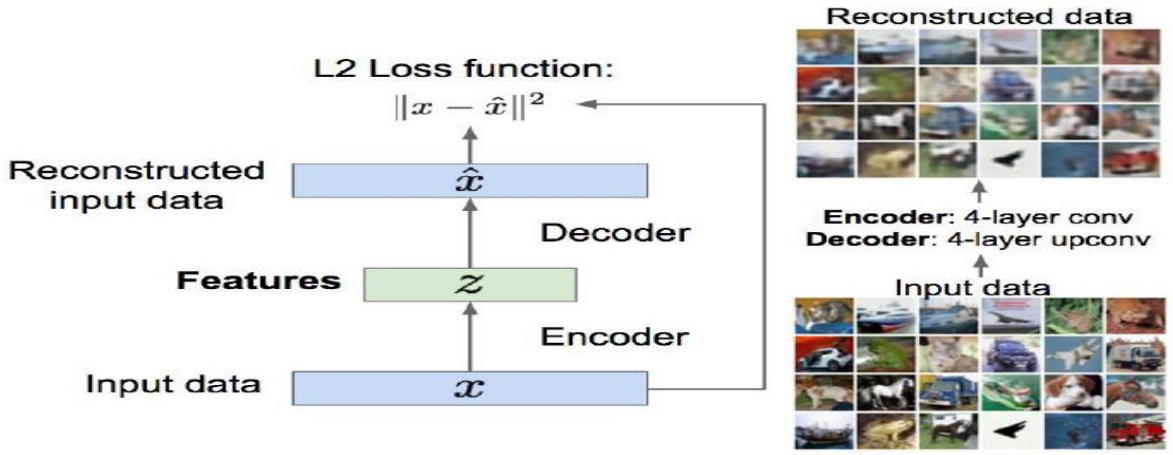
Examples: Clustering, Dimensionality reduction, Feature Learning, density estimation



K-means clustering



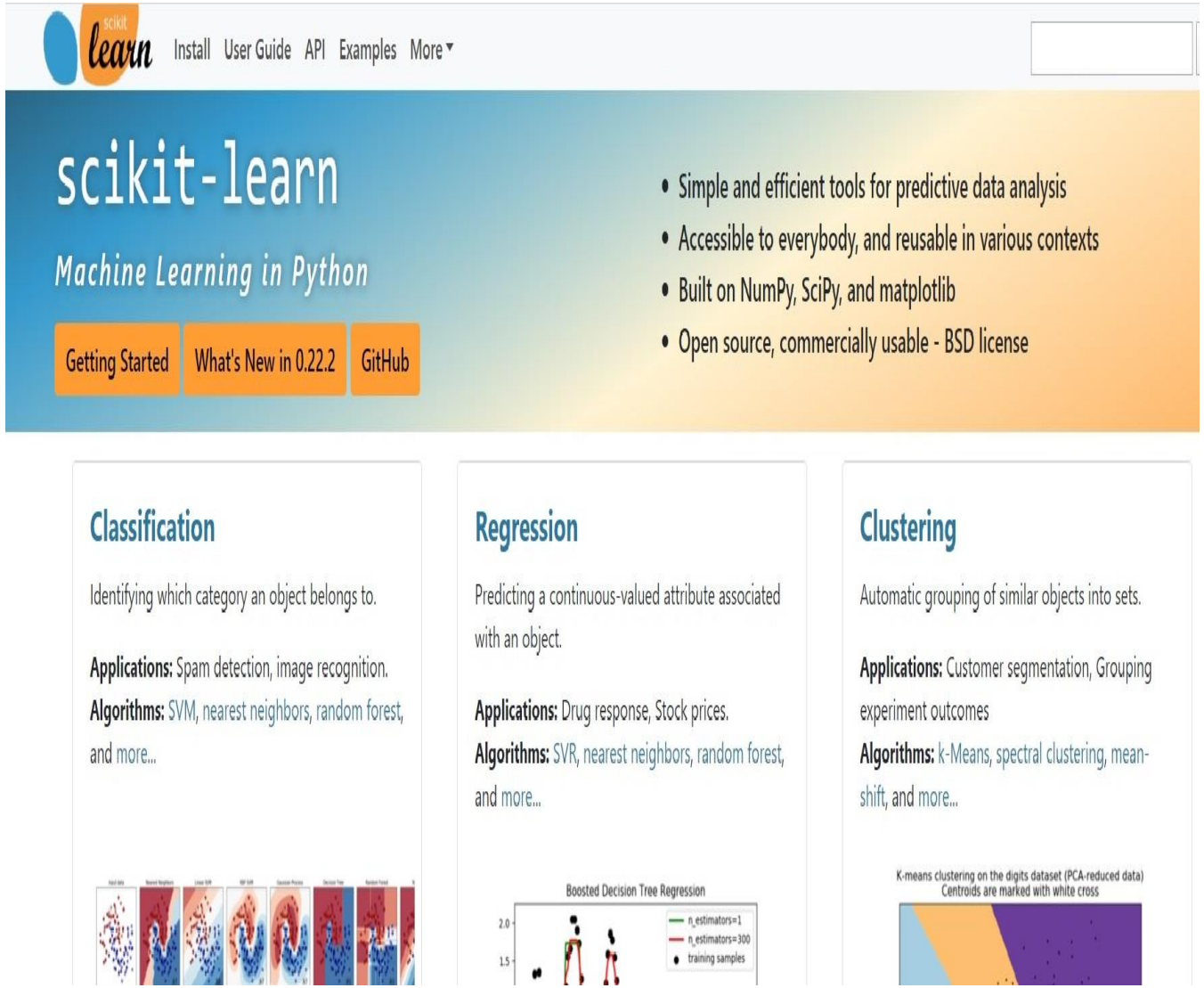
Principal Component Analysis
(Dimensionality reduction)

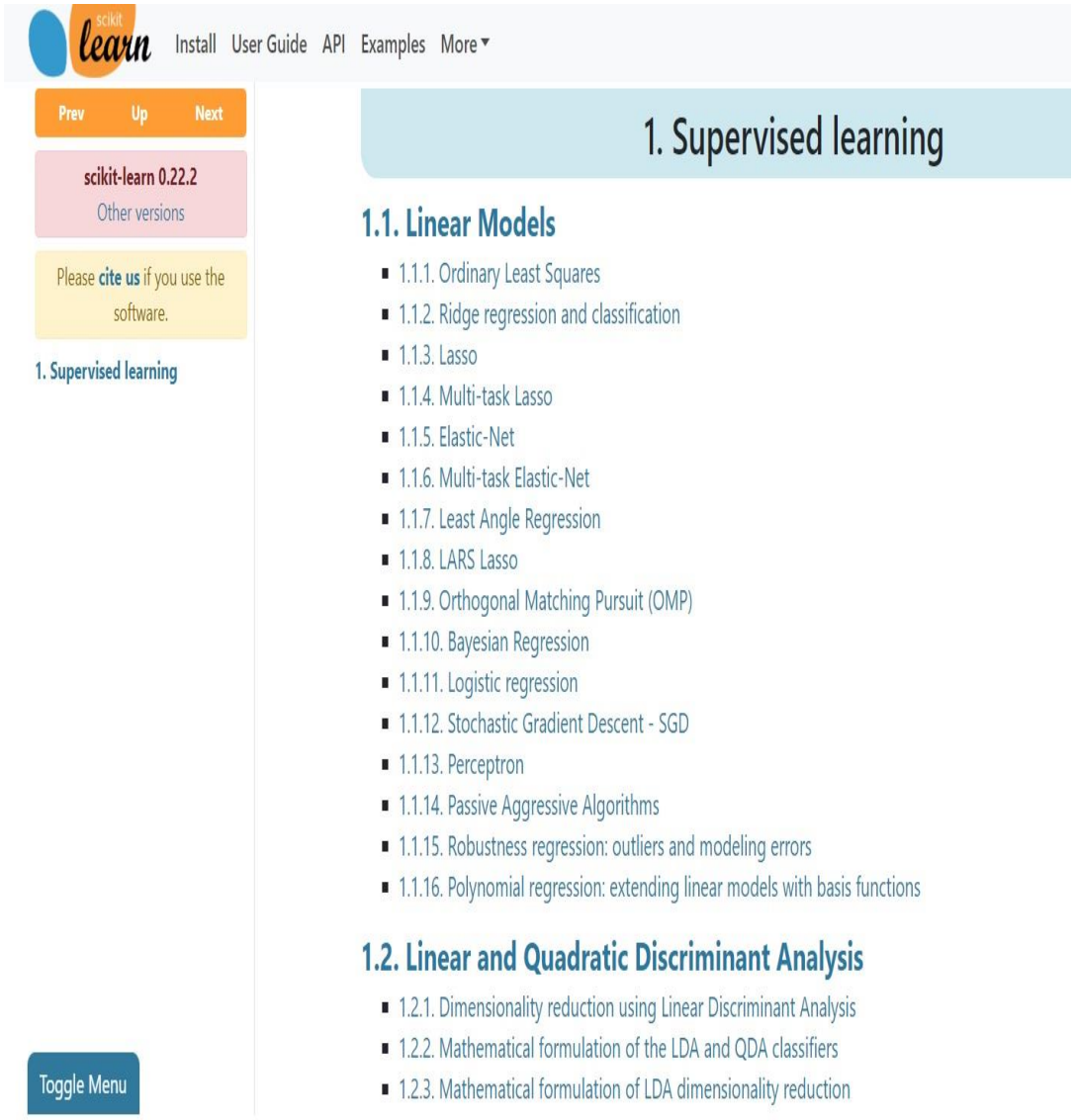


Autoencoders
(Feature learning)

Source: http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture12.pdf

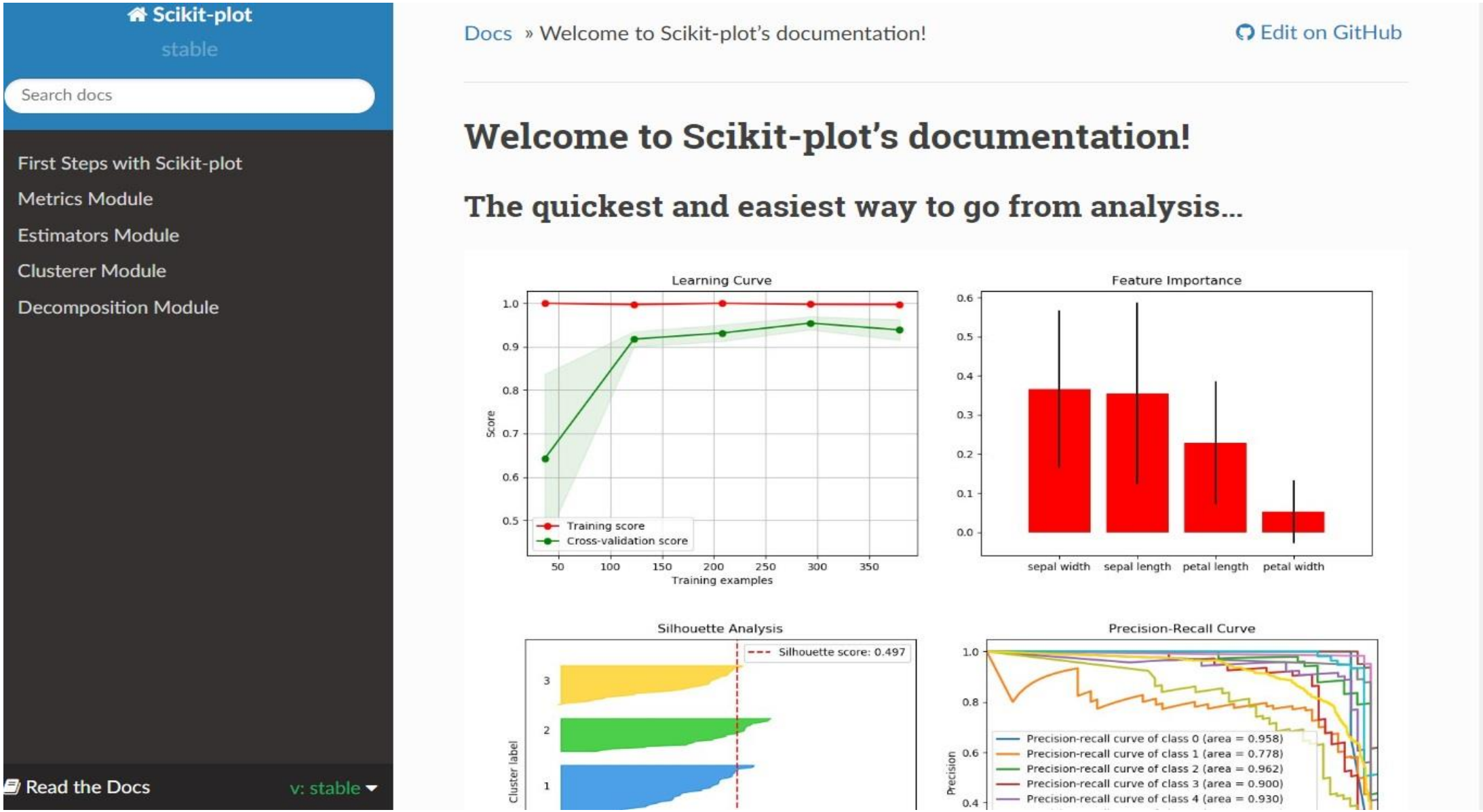
Scikit-learn





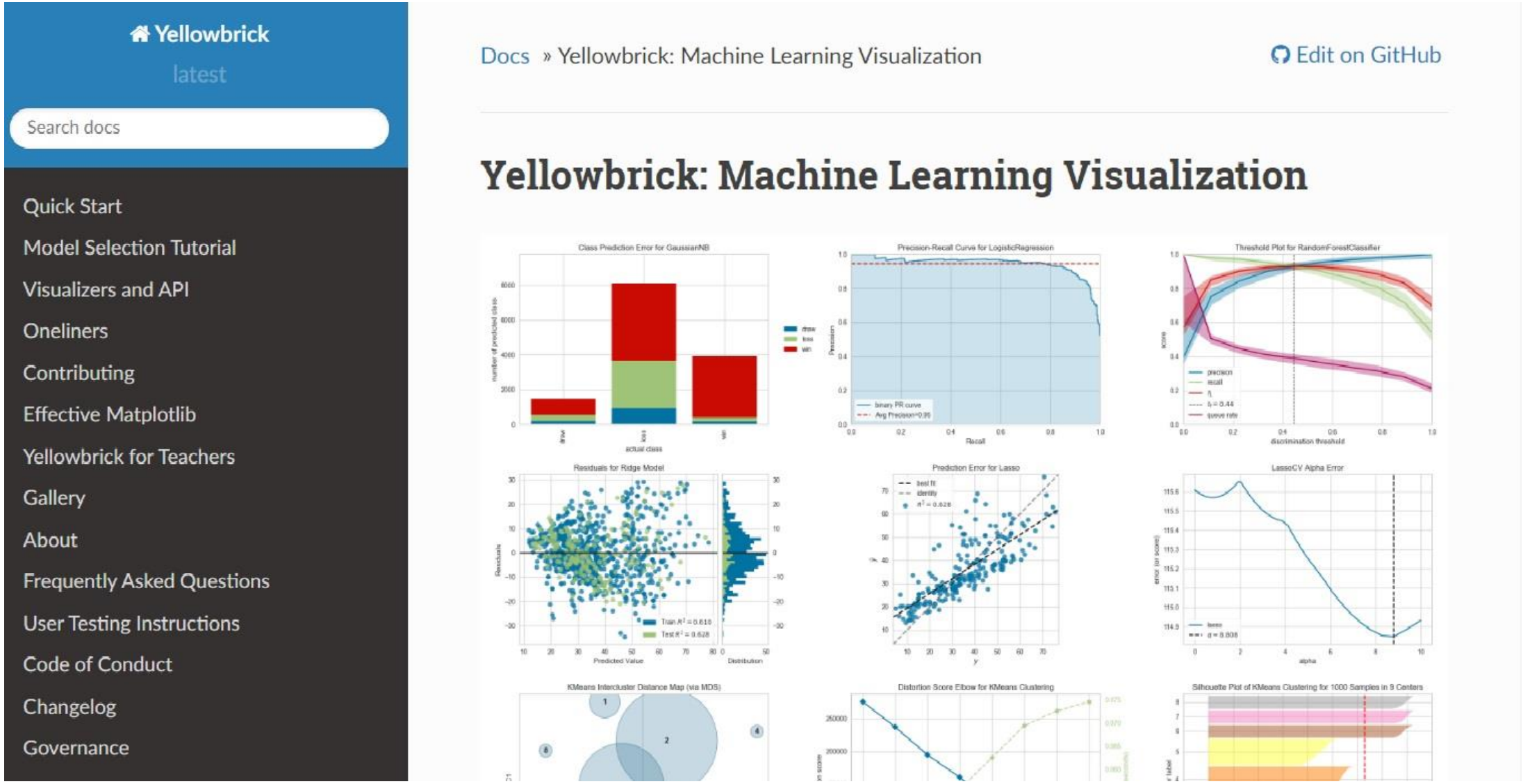
https://scikit-learn.org/stable/

Scikit-plot



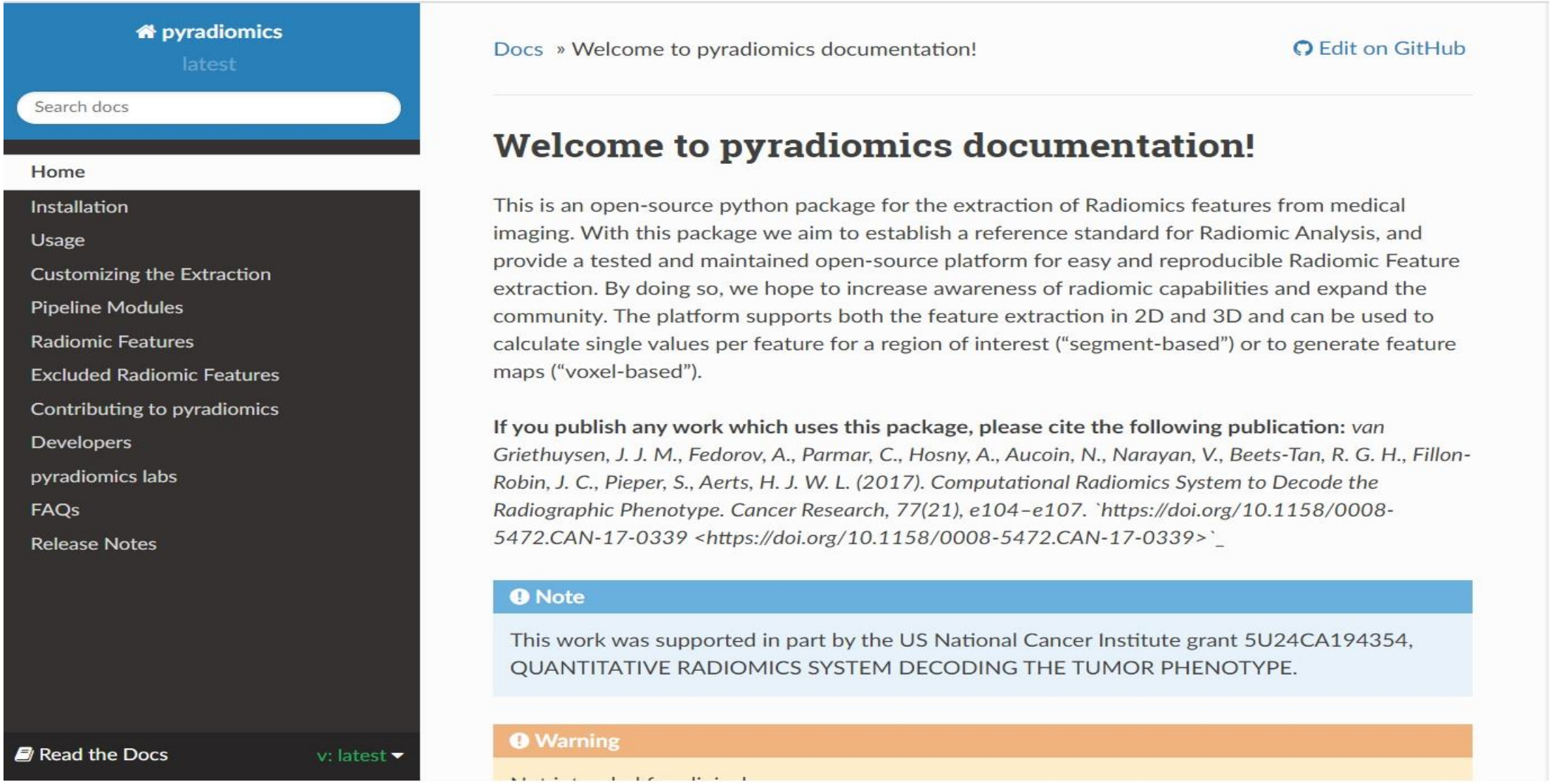
https://scikit-plot.readthedocs.io/en/stable/

Yellowbrick



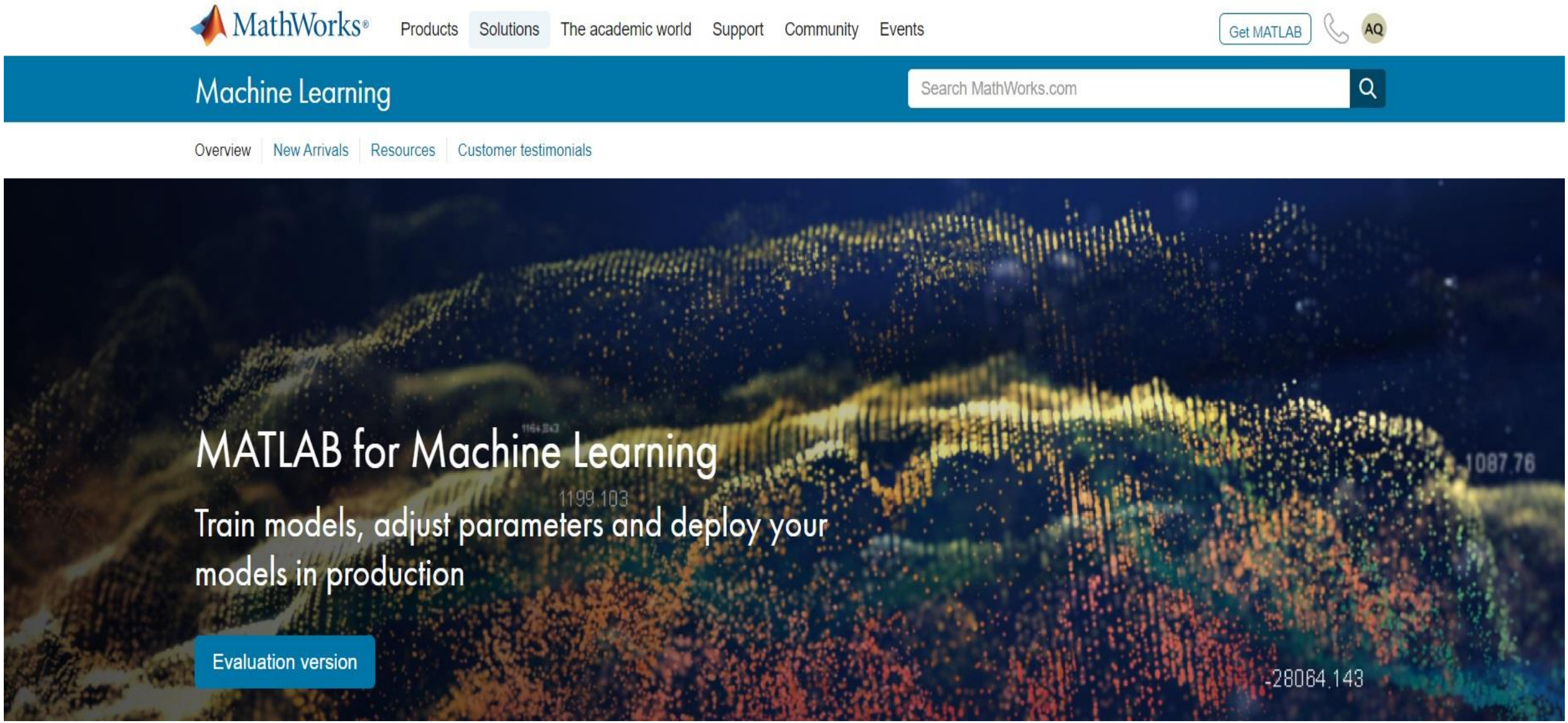
https://www.scikit-yb.org/en/latest/

Pyradiomics



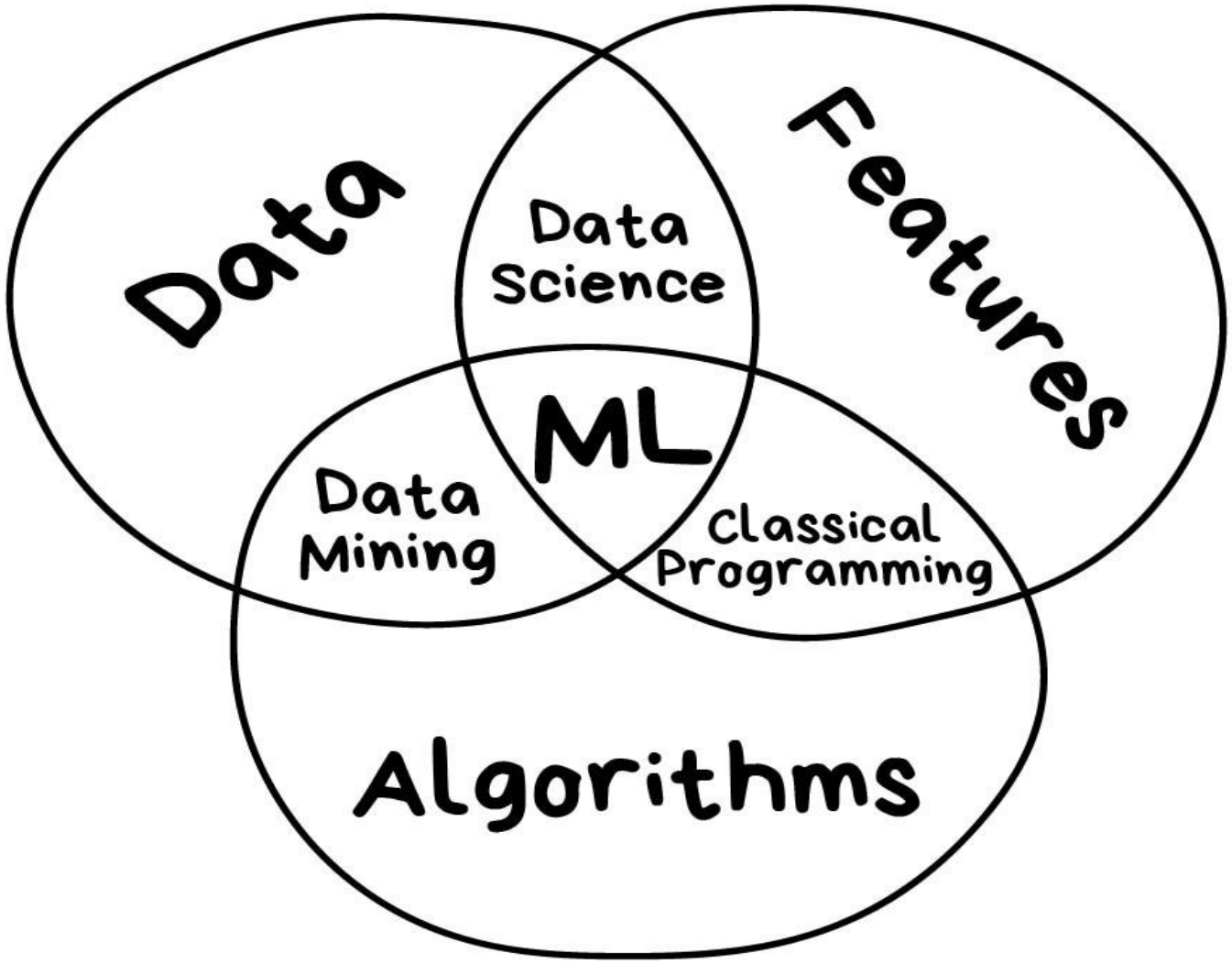
https://pyradiomics.readthedocs.io/en/latest/index.html

MATLAB



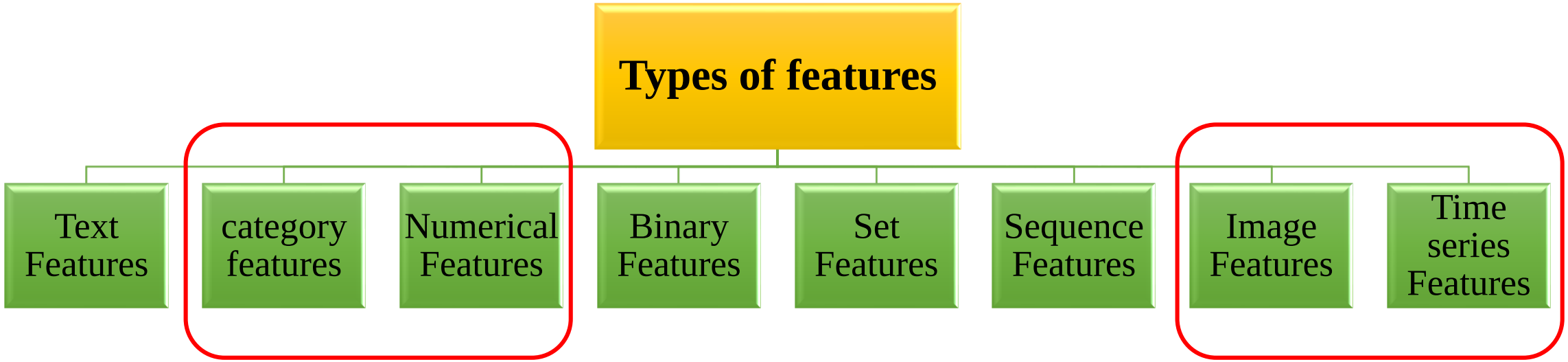
https://fr.mathworks.com/solutions/machine-learning.html

Components of ML



https://noeliagorod.com/2019/05/21/machine-learning-for-everyone-in-simple-words-with-real-world-examples-yes-again/

Types of Features



Feature Methods

Feature

- Characteristic, attribute, quality, aspect of something

Feature detection

- Selects regions of an image that have unique content, such as **corners or blobs**

Feature extraction

- Transformation of an input data into a set of features involves computing a descriptor, which is typically done on regions centered around detected features.

Feature selection

- Selection of a subset of features out of the extracted features

Feature Selection/Extraction

Feature Selection / Extraction

Curse of Dimensionality:

- samples data needed to train classifier
- function grows exponentially with dimensions
- Overfitting and Generalization performance

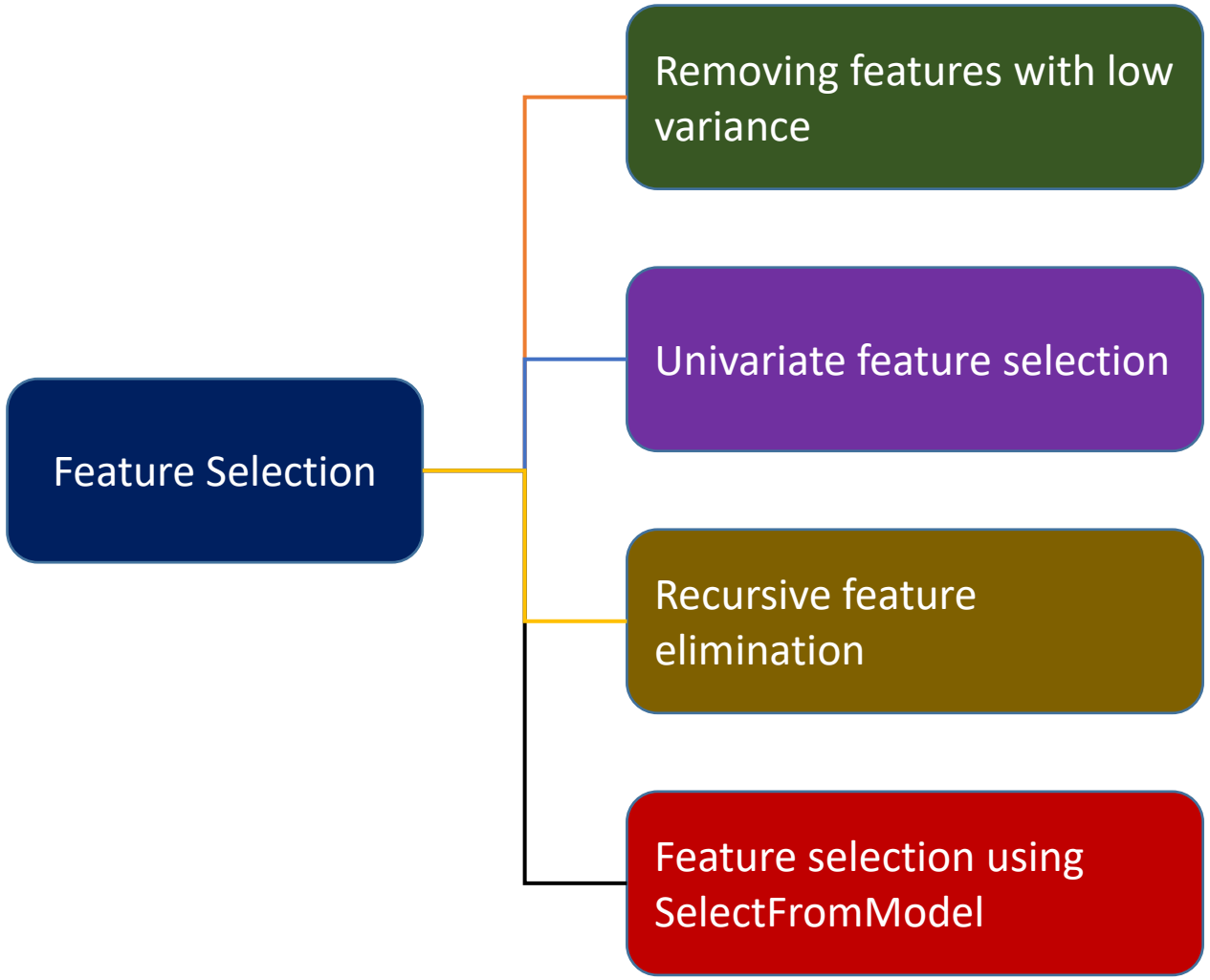
What features best characterize class?

- What words best characterize a document class
- Shape / texture / color

What features critical for performance?

- Subregions characterize protein function?

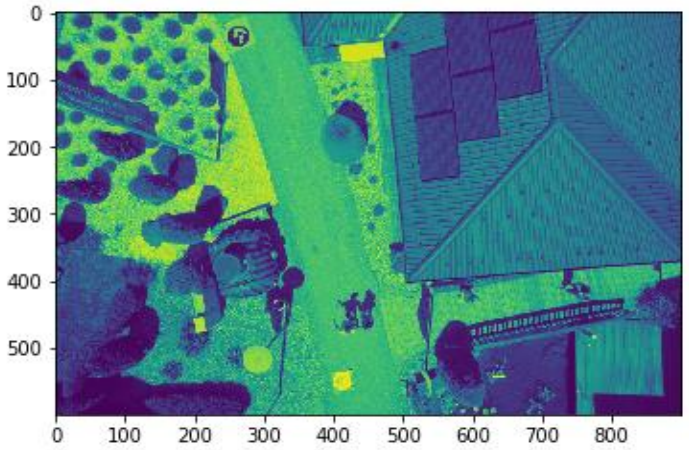
Feature Selection methods



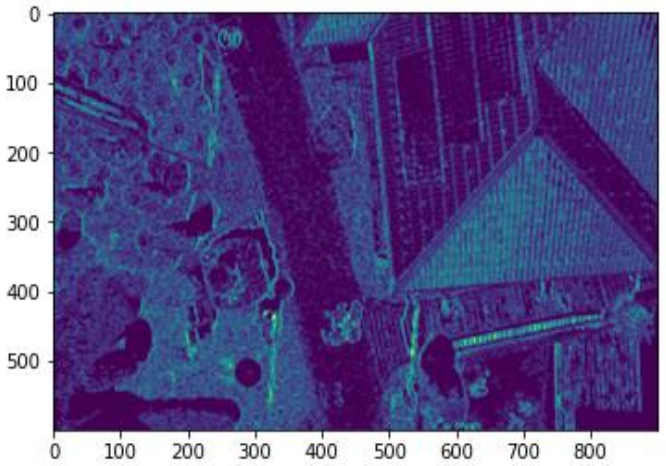
First Order Statistical Features

First Order Statistical Features	Mean(mean is the average value within the dataset)
	Variance(is a measure of the histogram width.it represents the deviation of gray levels from the means)
	Skewness(is a measure of the degree of histogram asymmetry around the mean)
	Kurtosis(is the measure of the histogram sharpness)

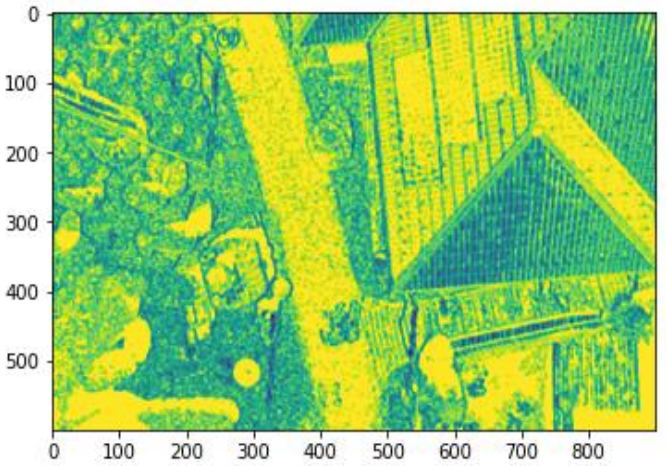
GLCM(grey level co-occurence matrix)



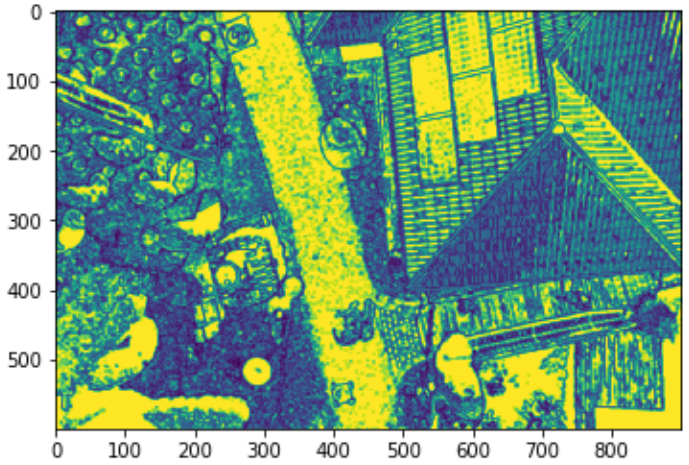
Main image



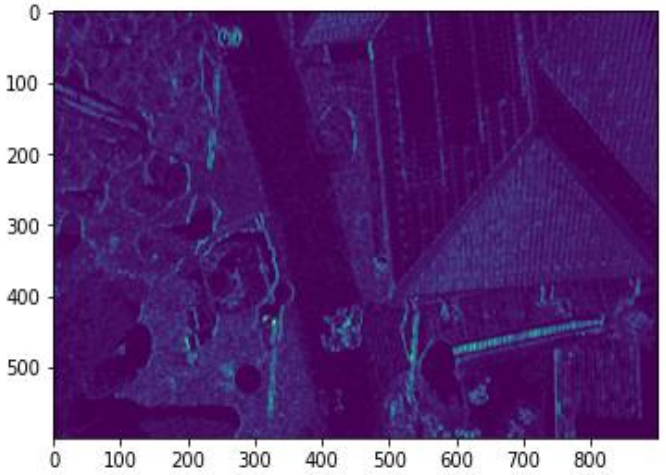
glcm_dissimilarity



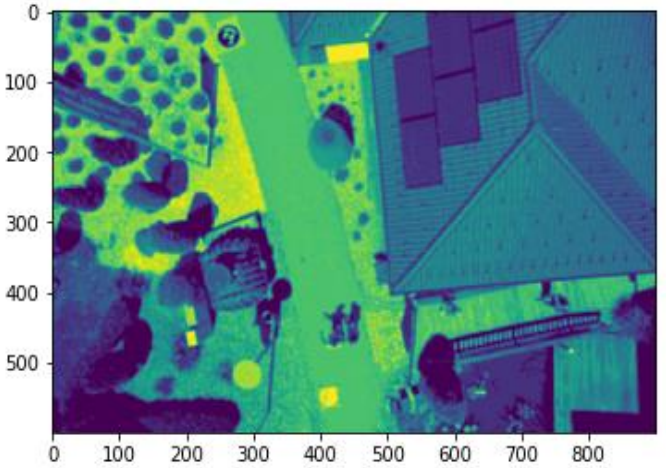
glcm_homogeneity



glcm_max



glcm_contrast

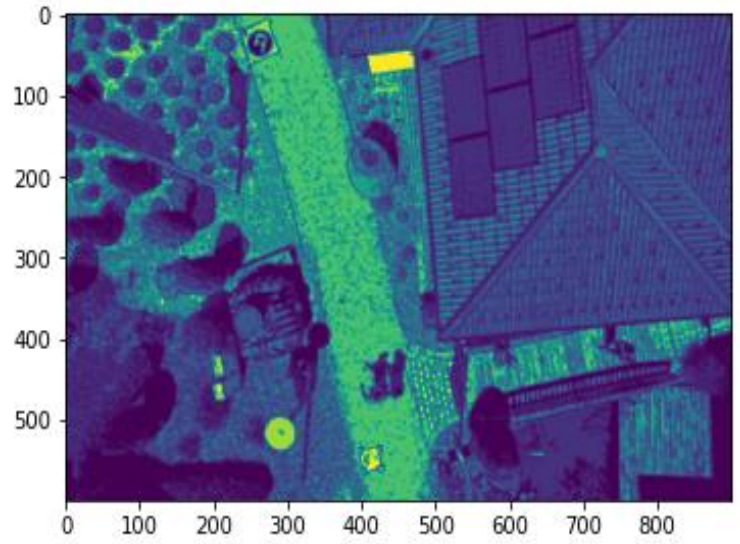


glcm_mean

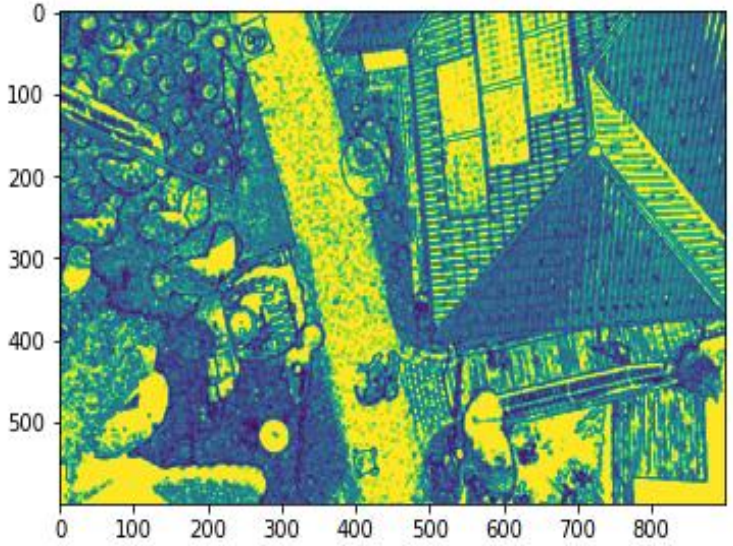
GLCM(grey level co-occurence matrix)

GLCM features

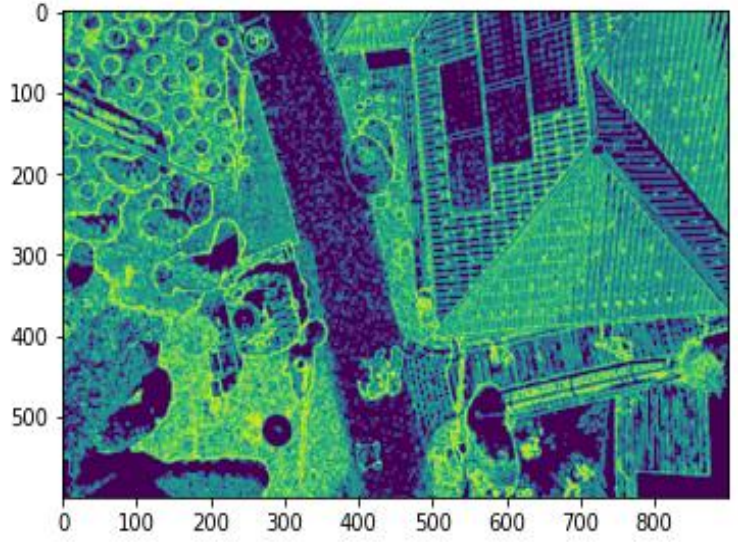
- Contrast
- Homogeneity
- Dissimilarity
- Std
- Mean
- Max
- ASM
- Entropy



glcm _std



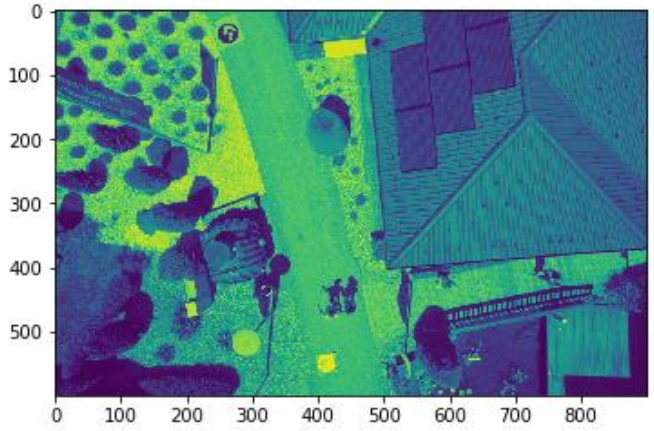
glcm_ASM



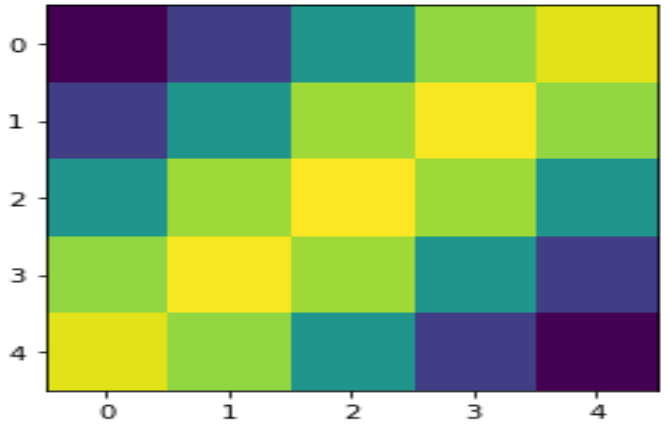
glcm _Entropy

Gabor Filter (Feature Extraction)

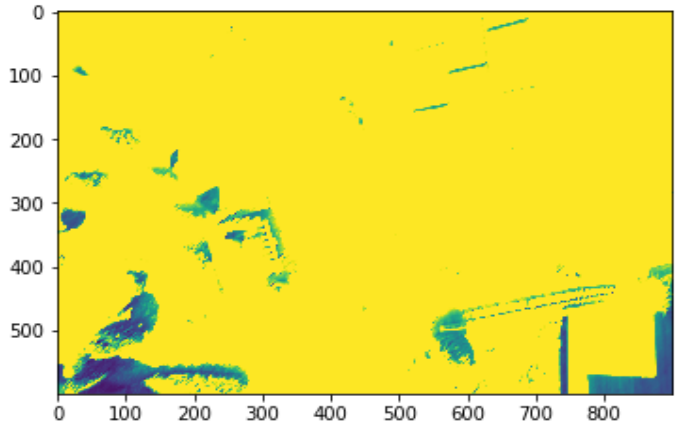
For image processing and computer vision, Gabor filters are generally used in texture analysis, edge detection, feature extraction, etc.
Gabor filters are special classes of bandpass filters, i.e., they allow a certain 'band' of frequencies and reject the others.



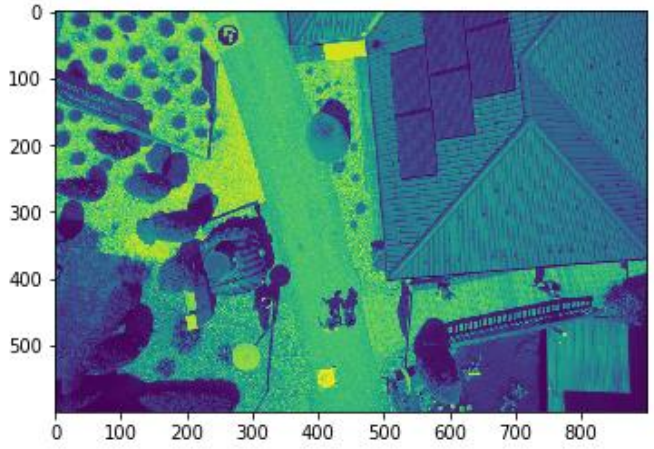
Main image



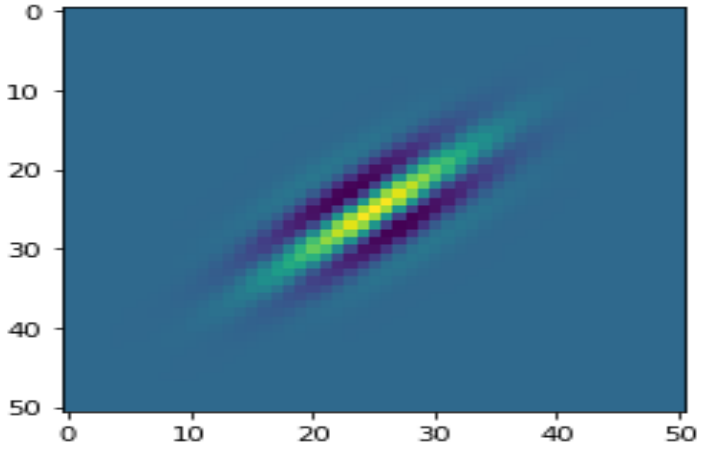
kernel function, sigma=3 k=5



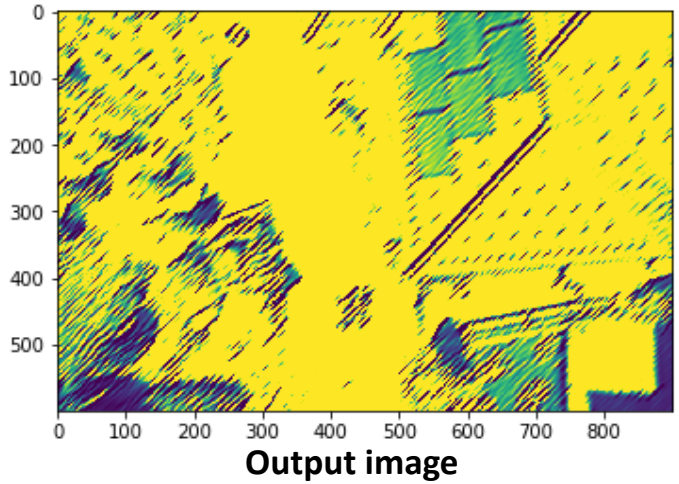
Output image



Main image

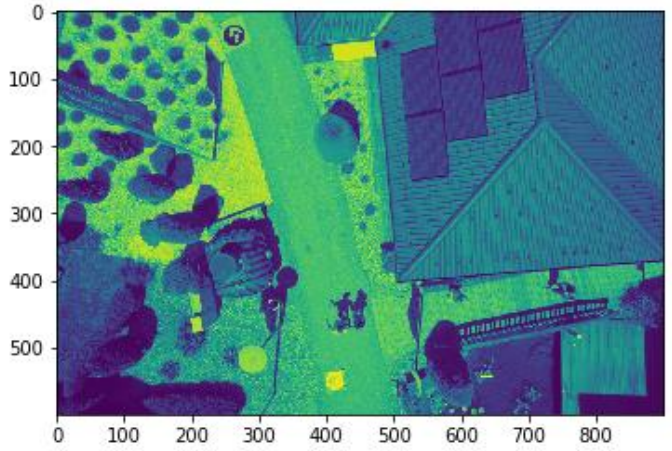


kernel function ,sigma=3 k=50

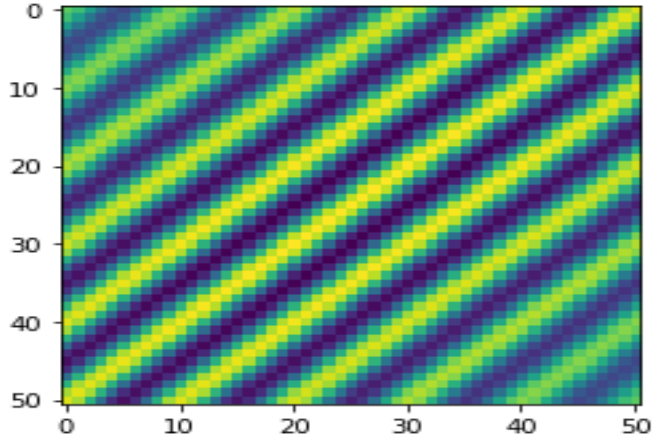


Output image

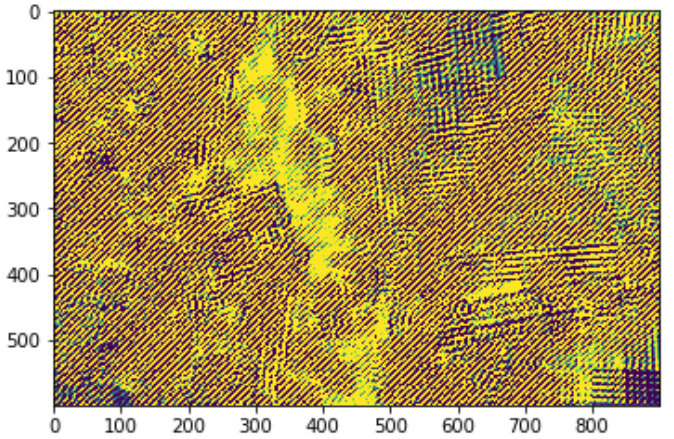
Gabor Filter (Feature Extraction)



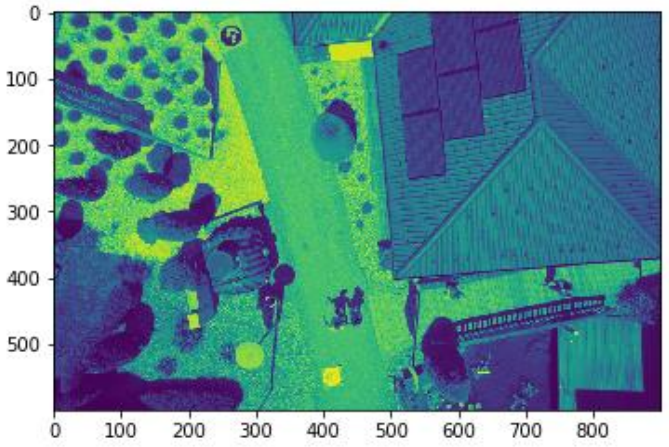
Main image



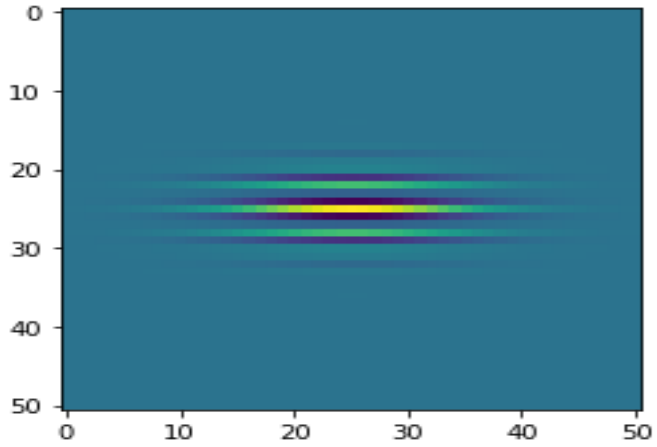
kernel function ,sigma=30,k=50



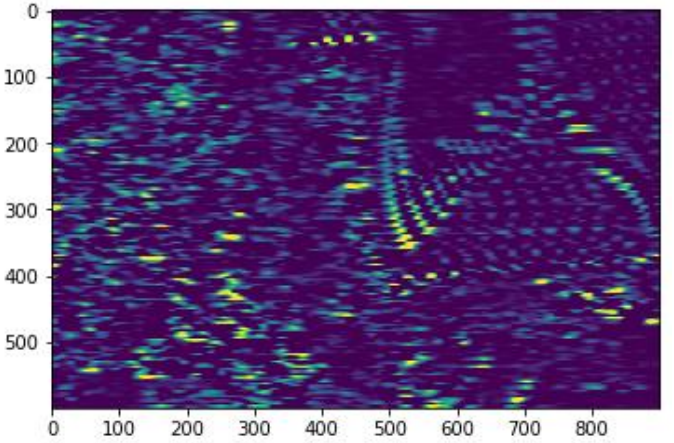
Output image



Main image



kernel function ,sigma=30,k=50



Output image

Detector and Descriptor (Feature Extraction)

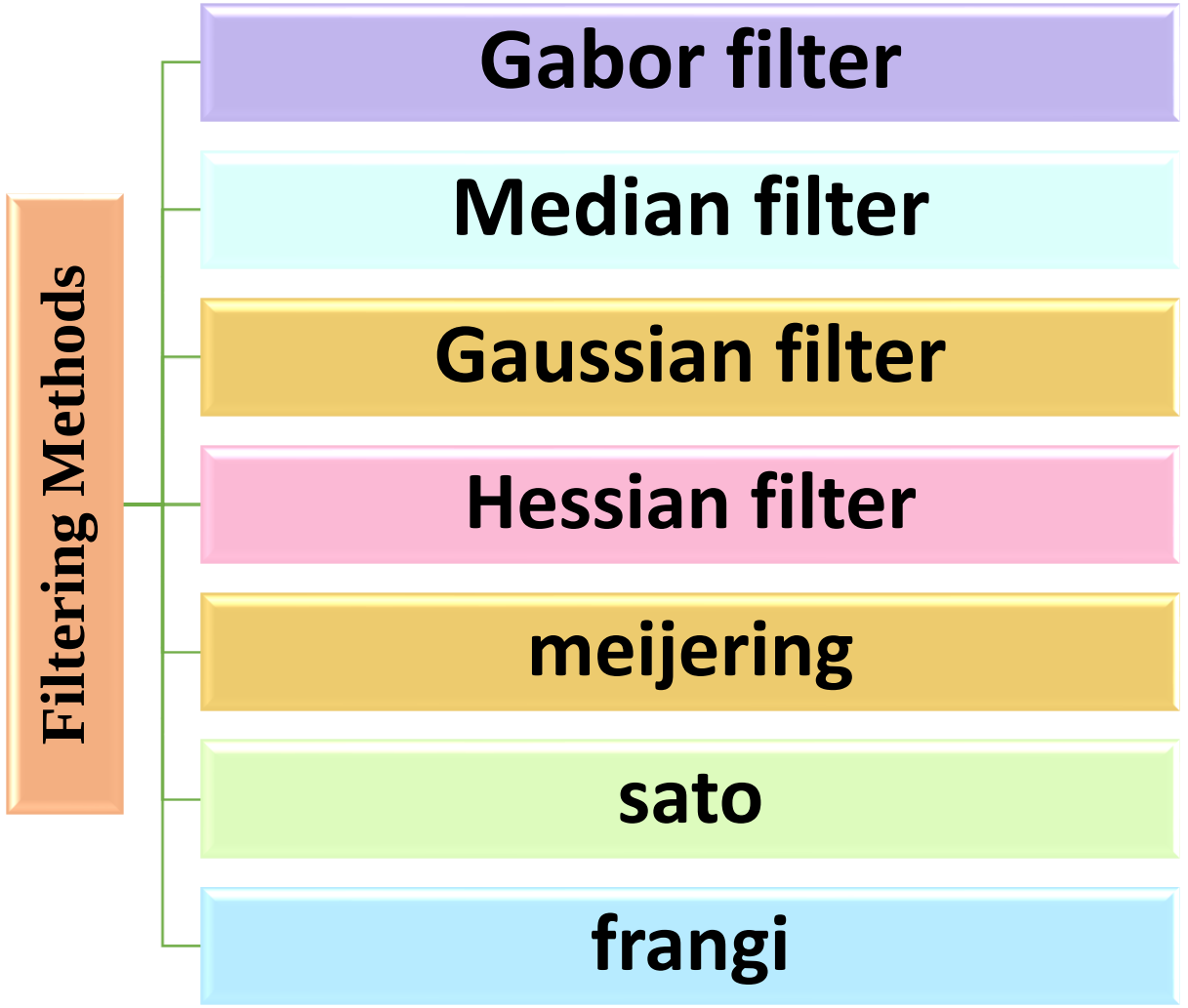
Detector

- FAST
- SURF
- BRISK

Descriptor

- HOG
- LBP
- SURF
- BRISK

Filtering Method (Feature Extraction)



Segmentation based on Image Features

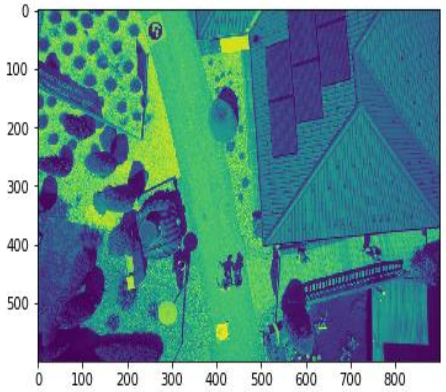
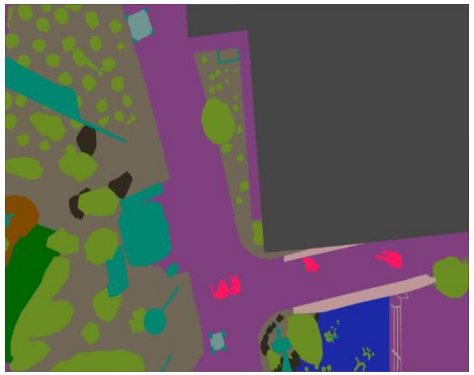


Image 300x300



mask 300x300

1x90000

1x90000

(pixels)	f1	f2								fn	Labels(depends on classes)
1											0
2											3
											0
											2
											0
											1
											1
											1
											1
90000											3

Feature Scaling or Normalization Methods

- **Feature scaling** is a method used to standardize the range of independent variables or **features** of data. In data processing, it is also known as data **normalization** and is generally performed during the data preprocessing step.

- **Why Scaling**
- Most of the times, your dataset will contain features highly varying in magnitudes, units, and range. But since most of the machine learning algorithms use Euclidian distance between two data points in their computations, this is a problem.
- If left alone, these algorithms only take in the magnitude of features neglecting the units. The results would vary greatly between different units, 5kg, and 5000gms. The features with high magnitudes will weigh in a lot more in the distance calculations than features with low magnitudes. To suppress this effect, we need to bring all features to the same level of magnitudes. This can be achieved by scaling.

Feature Scaling or Normalization Methods

- In this lecture we explore 3 methods of feature scaling that are implemented in scikit-learn:

1	2	3	4
StandardScaler	**MinMaxScaler**	**RobustScaler**	**Normalizer**

1. Standard Scaler

- The StandardScaler assumes your data is normally distributed within each feature and **will scale them such that the distribution is now centered around 0, with a standard deviation of 1.**
- The mean and standard deviation are calculated for the feature and then the feature is scaled based on:

$$z = \frac{x - \mu}{\sigma}$$

μ=Mean

$\sigma = standard\ deviation$

Feature Scaling or Normalization Methods

2. MinMaxScaler

- The MinMaxScaler is the probably the most famous scaling algorithm, and It essentially shrinks the range such that the range is now between **0 and 1 (or -1 to 1 if there are negative values).**
- This scaler works better for cases in which the standard scaler might not work so well. If the distribution is not Gaussian or the standard deviation is very small, the min-max scaler works better.
- However, it is sensitive to outliers, so if there are outliers in the data, you might want to consider the Robust Scaler follows the following formula for each feature:

$$X_{new} = \frac{X_i - \min(X)}{\max(X) - \min(X)}$$

Syntax :

```
from sklearn import preprocessing
scaler = preprocessing.MinMaxScaler()
scaled_df = scaler.fit_transform(df)
```

Feature Scaling or Normalization Methods

3. RobustScaler

- The RobustScaler uses a similar method to the Min-Max scaler but it instead uses the interquartile range, rather than the min-max, so that it is robust to outliers. Therefore it follows the formula:
- xi–Q1(x)/Q3(x)–Q1(x)
- For each feature. Of course, this means it is using the less of the data for scaling so it's more suitable for when there are outliers in the data.

Syntax:

```
from sklearn import preprocessing
scaler = preprocessing.RobustScaler()
robust_scaled_df = scaler.fit_transform(x)
```

Feature Scaling or Normalization Methods

4. Normalizer

- The normalizer scales each value by dividing each value by its magnitude in n-dimensional space for n number of features. Say your features were x, y, and z Cartesian co-ordinates your scaled value for x would be:
- Each point is now within 1 unit of the origin on this Cartesian coordinate system.
- X=xi/np.sqrt(xi**2+yi**2+zi**2

Syntax :

```
from sklearn import preprocessing
scaler = preprocessing.Normalizer()
scaled_df = scaler.fit_transform(df)
```

Feature Scaling or Normalization Methods

- **Where and when to Scale**
- **Rule of thumb I follow here is an algorithm that computes distance or assumes normality, scales your features!!!**

Some examples of algorithms where feature scaling matters are:

1. k-nearest neighbors with a Euclidean distance measure is sensitive to magnitudes and hence should be scaled for all features to weigh in equally.
2. Scaling is critical while performing Principal Component Analysis(PCA). PCA tries to get the features with maximum variance and the variance is high for high magnitude features. This skews the PCA towards high magnitude features.
3. **We can speed up gradient descent by scaling. This is because θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to the optimum when the variables are very uneven.**
4. **Tree-based models are not distance-based models and can handle varying ranges of features. Hence, Scaling is not required while modeling trees.**
5. **Algorithms like Linear Discriminant Analysis(LDA), Naive Bayes is by design equipped to handle this and gives weights to the features accordingly. Performing a feature scaling in these algorithms may not have much effect.**

Machine Learning Applications

Information Fusion 52 (2019) 1–12

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Full Length Article

Ensembles for feature selection: A review and future trends

Verónica Bolón-Canedo, Amparo Alonso-Betanzos*

CITIC Research Centre on Information and Communication Technology, University of A Coruña, Campus de Elviña s/n 15071, A Coruña, Spain

ARTICLE INFO

Keywords:
Ensemble learning
Feature selection

ABSTRACT

Ensemble learning is a prolific field in Machine Learning since it is based on the assumption that combining the output of multiple models is better than using a single model, and it usually provides good results. Normally, it has been commonly employed for classification, but it can be used to improve other disciplines such as feature selection. Feature selection consists of selecting the relevant features for a problem and discard those irrelevant or redundant, with the main goal of improving classification accuracy. In this work, we provide the reader with the basic concepts necessary to build an ensemble for feature selection, as well as reviewing the up-to-date advances and commenting on the future trends that are still to be faced.

Machine Learning Applications

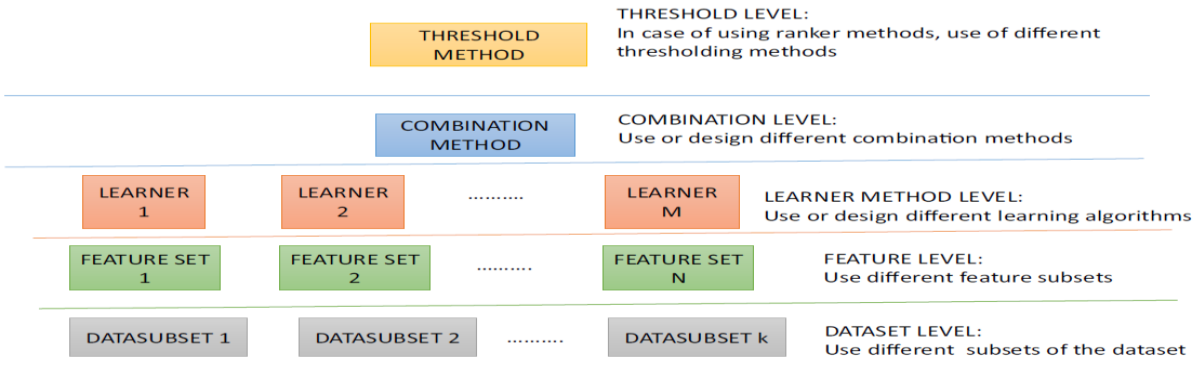


Fig. 1. The different levels than can be varied in an ensemble for feature selection.

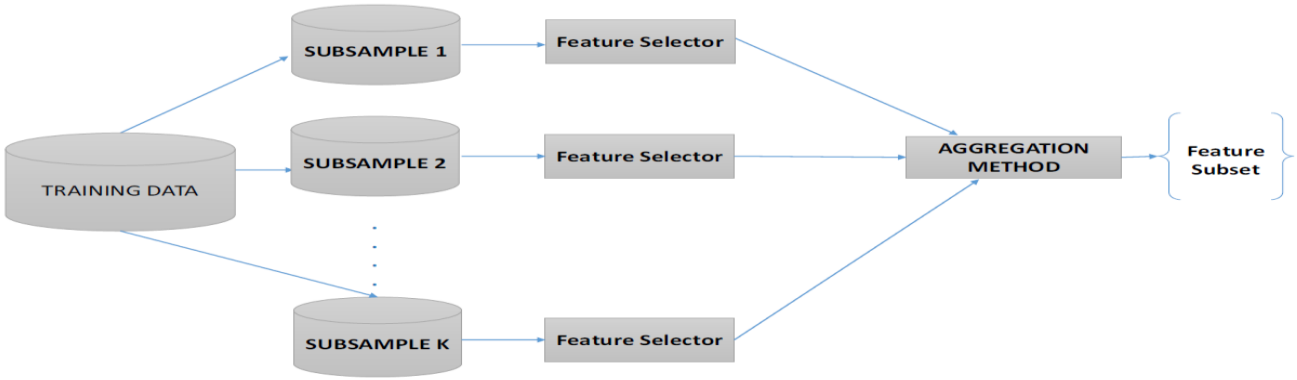


Fig. 2. A scheme of the Homogeneous approach.

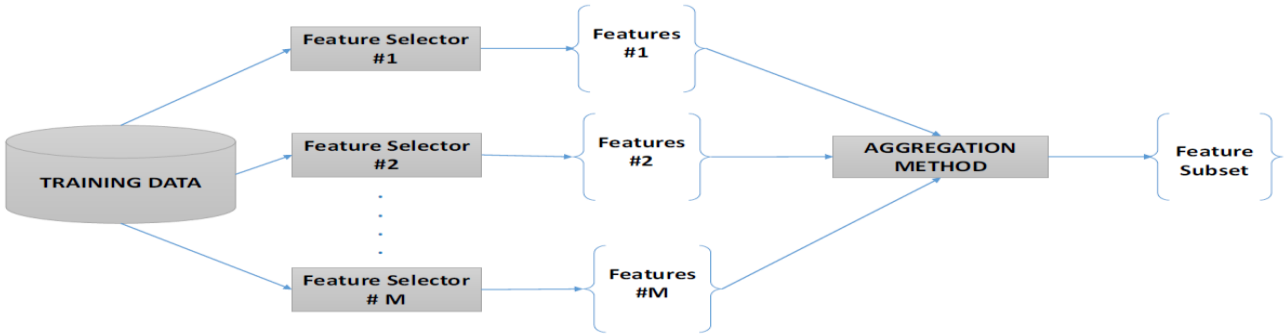


Fig. 3. A scheme of the Heterogeneous approach.

Information Fusion 52 (2019) 1–12

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Full Length Article

Ensembles for feature selection: A review and future trends

Verónica Bolón-Canedo, Amparo Alonso-Betanzos*

CITIC Research Centre on Information and Communication Technology, University of A Coruña, Campus de Elviña s/n 15071, A Coruña, Spain

ARTICLE INFO

Keywords:
Ensemble learning
Feature selection

ABSTRACT

Ensemble learning is a prolific field in Machine Learning since it is based on the assumption that combining the output of multiple models is better than using a single model, and it usually provides good results. Normally, it has been commonly employed for classification, but it can be used to improve other disciplines such as feature selection. Feature selection consists of selecting the relevant features for a problem and discard those irrelevant or redundant, with the main goal of improving classification accuracy. In this work, we provide the reader with the basic concepts necessary to build an ensemble for feature selection, as well as reviewing the up-to-date advances and commenting on the future trends that are still to be faced.

Machine Learning Applications

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Full Length Article

Machine learning for integrating data in biology and medicine: Principles, practice, and opportunities

Marinka Zitnik[a,*], Francis Nguyen[b,c], Bo Wang[d], Jure Leskovec[a,e,*], Anna Goldenberg[f,g,h,*], Michael M. Hoffman[b,c,g,h,*]

[a] Department of Computer Science, Stanford University, Stanford, CA, USA
[b] Department of Medical Biophysics, University of Toronto, Toronto, ON, Canada
[c] Princess Margaret Cancer Centre, Toronto, ON, Canada
[d] Hikvision Research Institute, Santa Clara, CA, USA
[e] Chan Zuckerberg Biohub, San Francisco, CA, USA
[f] Genetics & Genome Biology, SickKids Research Institute, Toronto, ON, Canada
[g] Department of Computer Science, University of Toronto, Toronto, ON, Canada
[h] Vector Institute, Toronto, ON, Canada

ARTICLE INFO

Keywords:
Computational biology
Personalized medicine
Systems biology
Heterogeneous data
Machine learning

ABSTRACT

New technologies have enabled the investigation of biology and human health at an unprecedented scale and in multiple dimensions. These dimensions include a myriad of properties describing genome, epigenome, transcriptome, microbiome, phenotype, and lifestyle. No single data type, however, can capture the complexity of all the factors relevant to understanding a phenomenon such as a disease. Integrative methods that combine data from multiple technologies have thus emerged as critical statistical and computational approaches. The key challenge in developing such approaches is the identification of effective models to provide a comprehensive and relevant systems view. An ideal method can answer a biological or medical question, identifying important features and predicting outcomes, by harnessing heterogeneous data across several dimensions of biological variation. In this Review, we describe the principles of data integration and discuss current methods and available implementations. We provide examples of successful data integration in biology and medicine. Finally, we discuss current challenges in biomedical integrative methods and our perspective on the future development of the field.

Machine Learning Applications

Contents lists available at ScienceDirect
Information Fusion
journal homepage: www.elsevier.com/locate/inffus

Full Length Article

Machine learning for integrating data in biology and medicine: Principles, practice, and opportunities

Marinka Zitnik[a,*], Francis Nguyen[b,c], Bo Wang[d], Jure Leskovec[a,e,*], Anna Goldenberg[f,g,h,*], Michael M. Hoffman[b,c,g,h,*]

[a] Department of Computer Science, Stanford University, Stanford, CA, USA
[b] Department of Medical Biophysics, University of Toronto, Toronto, ON, Canada
[c] Princess Margaret Cancer Centre, Toronto, ON, Canada
[d] Hikvision Research Institute, Santa Clara, CA, USA
[e] Chan Zuckerberg Biohub, San Francisco, CA, USA
[f] Genetics & Genome Biology, SickKids Research Institute, Toronto, ON, Canada
[g] Department of Computer Science, University of Toronto, Toronto, ON, Canada
[h] Vector Institute, Toronto, ON, Canada

ARTICLE INFO

Keywords:
Computational biology
Personalized medicine
Systems biology
Heterogeneous data
Machine learning

ABSTRACT

New technologies have enabled the investigation of biology and human health at an unprecedented scale and in multiple dimensions. These dimensions include a myriad of properties describing genome, epigenome, transcriptome, microbiome, phenotype, and lifestyle. No single data type, however, can capture the complexity of all the factors relevant to understanding a phenomenon such as a disease. Integrative methods that combine data from multiple technologies have thus emerged as critical statistical and computational approaches. The key challenge in developing such approaches is the identification of effective models to provide a comprehensive and relevant systems view. An ideal method can answer a biological or medical question, identifying important features and predicting outcomes, by harnessing heterogeneous data across several dimensions of biological variation. In this Review, we describe the principles of data integration and discuss current methods and available implementations. We provide examples of successful data integration in biology and medicine. Finally, we discuss current challenges in biomedical integrative methods and our perspective on the future development of the field.

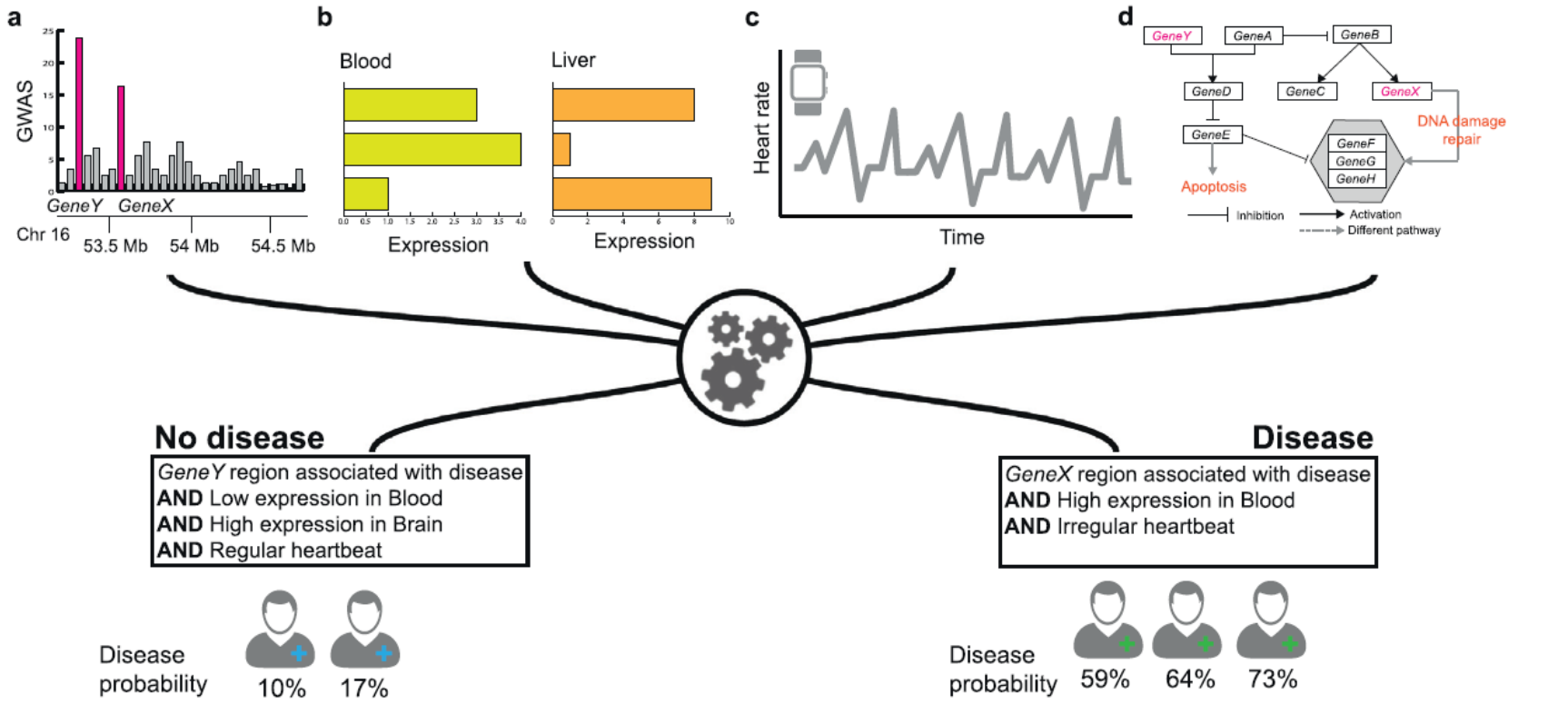


Fig. 1. The importance of data integration in biomedicine. Considering variation in only a single data type can miss many important patterns that can only be observed by considering multiple levels of biomedical data. Shown is a hypothetical example using disease diagnostics as a point of interest. When a new patient arrives to the clinic, (a) domain experts sequence the patient's genome and compare it with a database to identify mutations and disease-causing genes, (b) perform laboratory tests using tissue samples, and (c) process information about the patient's behavior and lifestyle. (d) The patient's genomic, transcriptomic, and lifestyle information is combined with curated databases of biomedical knowledge (*e.g.*, disease and metabolic pathways). Finally, a machine learning algorithm predicts probability that the patient will develop a particular disease in near future. To make accurate prediction, the machine learning model needs to use many different types of data. This example illustrates that accurate prediction can only be made by analyzing multiple types of patient's data.

Machine Learning Applications

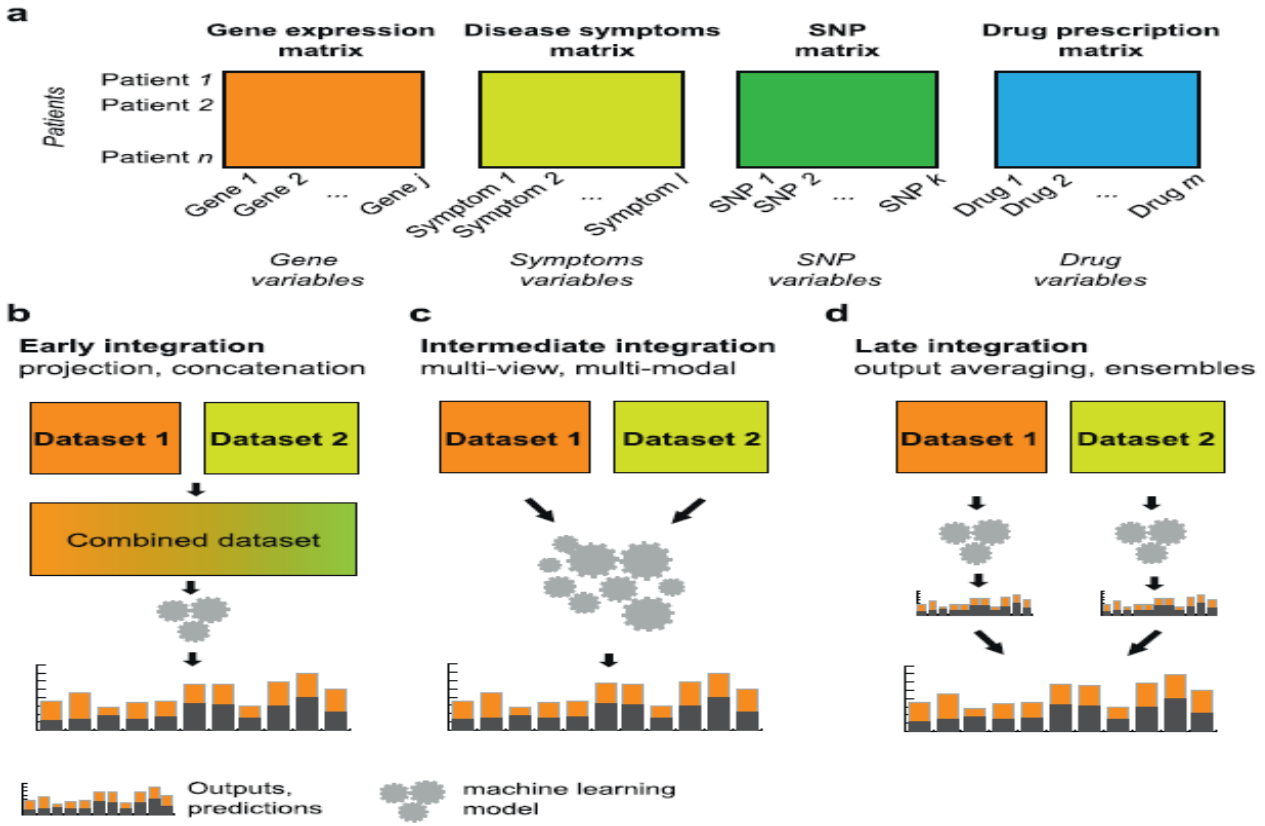


Fig. 2. Categorization of approaches for data integration. (a) Examples of multi-omics data about patients. (b–d) Data integration approaches can be divided into three categories. (b) *Early integration approaches* involve combining datasets from different data types at the raw or processed level before analysis and prediction. (c) *Intermediate integration approaches* transform or map the underlying datasets at the same time as they estimate model parameters. (d) *Late integration approaches* perform analysis on each dataset independently, which is followed by integration of the resulting models to generate predictions, *e.g.*, prognosis for a particular patient. SNP, single-nucleotide polymorphism.

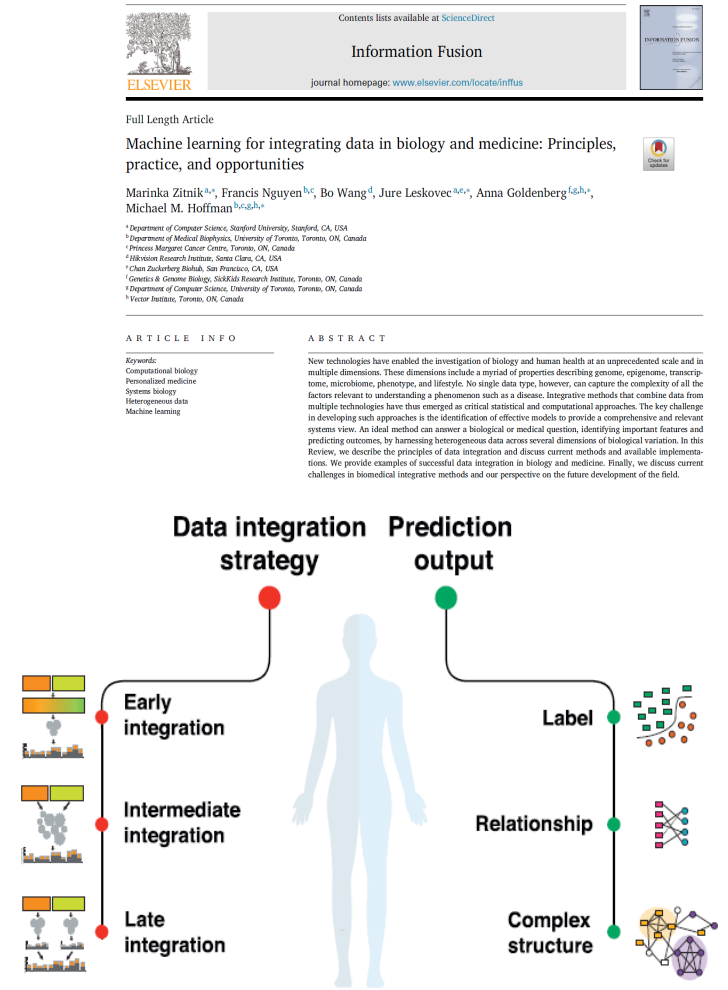


Fig. 3. Data integration. Data integration approaches combine multiples sources of information in a statistically meaningful way to provide a comprehensive analysis of biomedical dara. Broadly, existing approaches use three distinct strategies (*i.e.*, early, intermediate, and late integration; see also Fig. 2) and produce three types of prediction outputs (*i.e.*, a label representing probability of an entity belonging to a given class; a relationship representing probability of an association between two entities; and a complex structure, such as an inferred network or a partitioning of entities into groups).

Machine Learning Applications

Information Fusion 50 (2019) 92–111

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Full Length Article

Data fusion and machine learning for industrial prognosis: Trends and perspectives towards Industry 4.0

Alberto Diez-Olivan [a], Javier Del Ser [a,b,c,*], Diego Galar [a,d], Basilio Sierra [e]

[a] TECNALIA, Donostia-San Sebastián 20009, Spain
[b] Department of Communications Engineering, University of the Basque Country (UPV/EHU), Bilbao 48013, Spain
[c] Basque Center for Applied Mathematics (BCAM), Bilbao, Bizkaia 48009, Spain
[d] Department of Civil, Environmental and Natural Resources Engineering, Operation, Maintenance and Acoustics, Luleå University of Technology, Luleå, Sweden
[e] Department of Computer Sciences and Artificial Intelligence, University of the Basque Country (UPV/EHU), Donostia-San Sebastián 20018, Spain

ARTICLE INFO

Keywords:
Data-driven prognosis
Data fusion
Machine learning
Industry 4.0

ABSTRACT

The so-called "smartization" of manufacturing industries has been conceived as the fourth industrial revolution or Industry 4.0, a paradigm shift propelled by the upsurge and progressive maturity of new Information and Communication Technologies (ICT) applied to industrial processes and products. From a data science perspective, this paradigm shift allows extracting relevant knowledge from monitored assets through the adoption of intelligent monitoring and data fusion strategies, as well as by the application of machine learning and optimization methods. One of the main goals of data science in this context is to effectively predict abnormal behaviors in industrial machinery, tools and processes so as to anticipate critical events and damage, eventually causing important economical losses and safety issues. In this context, data-driven prognosis is gradually gaining attention in different industrial sectors. This paper provides a comprehensive survey of the recent developments in data fusion and machine learning for industrial prognosis, placing an emphasis on the identification of research trends, niches of opportunity and unexplored challenges. To this end, a principled categorization of the utilized feature extraction techniques and machine learning methods will be provided on the basis of its intended purpose: analyze what caused the failure (descriptive), determine when the monitored asset will fail (predictive) or decide what to do so as to minimize its impact on the industry at hand (prescriptive). This threefold analysis, along with a discussion on its hardware and software implications, intends to serve as a stepping stone for future researchers and practitioners to join the community investigating on this vibrant field.

Machine Learning Applications

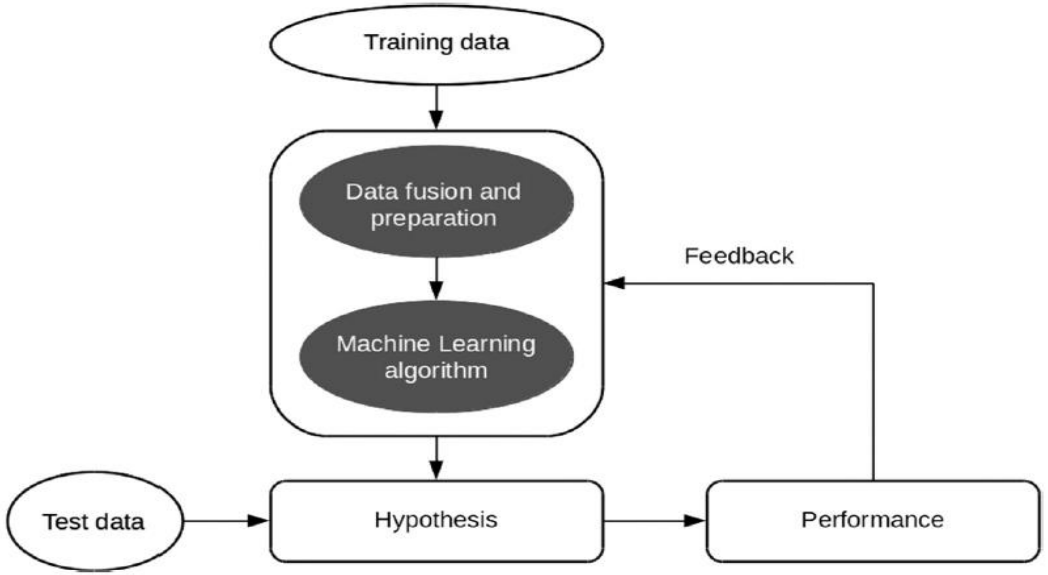


Fig. 1. Generic schematic diagram of a machine learning process.

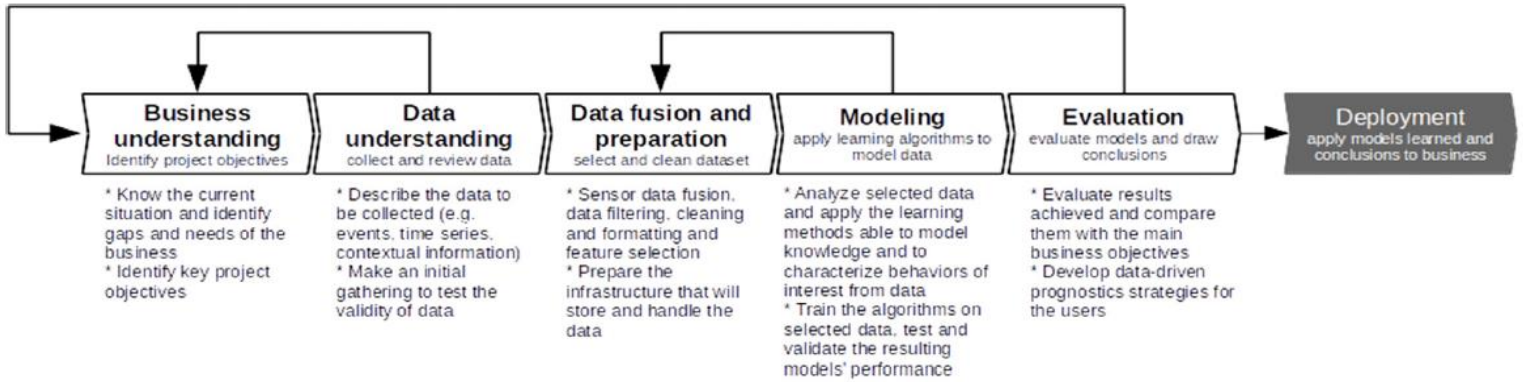


Fig. 2. The CRISP methodology for data-driven industrial prognosis.

Information Fusion 50 (2019) 92–111

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Full Length Article

Data fusion and machine learning for industrial prognosis: Trends and perspectives towards Industry 4.0

Alberto Diez-Olivan[a], Javier Del Ser[a,b,c,*], Diego Galar[a,d], Basilio Sierra[e]

[a] TECNALIA, Donostia-San Sebastián 20009, Spain
[b] Department of Communications Engineering, University of the Basque Country (UPV/EHU), Bilbao 48013, Spain
[c] Basque Center for Applied Mathematics (BCAM), Bilbao, Bizkaia 48009, Spain
[d] Department of Civil, Environmental and Natural Resources Engineering, Operation, Maintenance and Acoustics, Luled University of Technology, Luled, Sweden
[e] Department of Computer Sciences and Artificial Intelligence, University of the Basque Country (UPV/EHU), Donostia-San Sebastián 20018, Spain

ARTICLE INFO

Keywords:
Data-driven prognosis
Data fusion
Machine learning
Industry 4.0

ABSTRACT

The so-called "smartization" of manufacturing industries has been conceived as the fourth industrial revolution or Industry 4.0, a paradigm shift propelled by the upsurge and progressive maturity of new Information and Communication Technologies (ICT) applied to industrial processes and products. From a data science perspective, this paradigm shift allows extracting relevant knowledge from monitored assets through the adoption of intelligent monitoring and data fusion strategies, as well as by the application of machine learning and optimization methods. One of the main goals of data science in this context is to effectively predict abnormal behaviors in industrial machinery, tools and processes so as to anticipate critical events and damage, eventually causing important economical losses and safety issues. In this context, data-driven prognosis is gradually gaining attention in different industrial sectors. This paper provides a comprehensive survey of the recent developments in data fusion and machine learning for industrial prognosis, placing an emphasis on the identification of research trends, niches of opportunity and unexplored challenges. To this end, a principled categorization of the utilized feature extraction techniques and machine learning methods will be provided on the basis of its intended purpose: analyze what caused the failure (descriptive), determine when the monitored asset will fail (predictive) or decide what to do so as to minimize its impact on the industry at hand (prescriptive). This threefold analysis, along with a discussion on its hardware and software implications, intends to serve as a stepping stone for future researchers and practitioners to join the community investigating on this vibrant field.

Machine Learning Applications

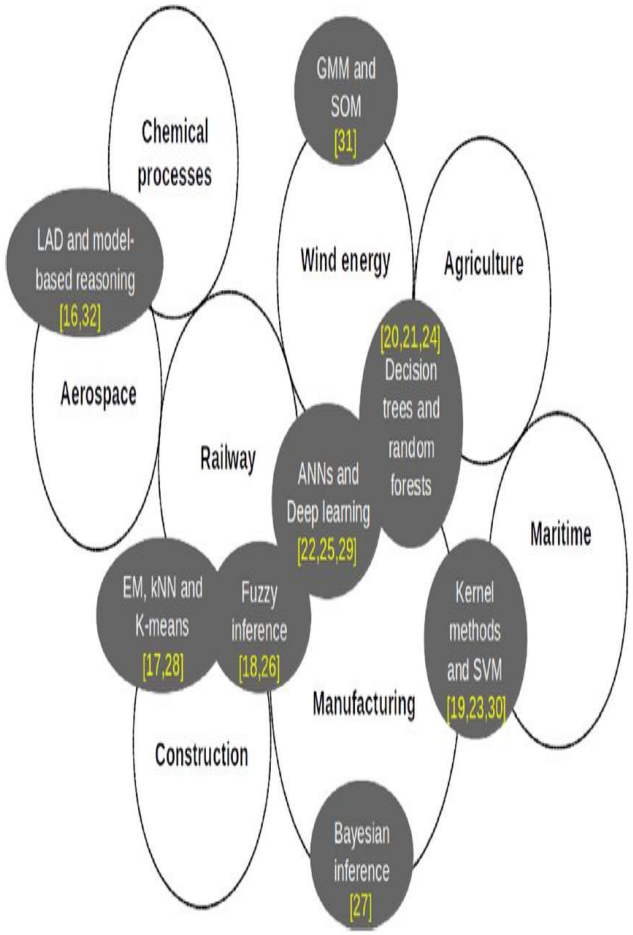


Fig. 4. Solutions and industrial sectors addressed for descriptive prognosis.

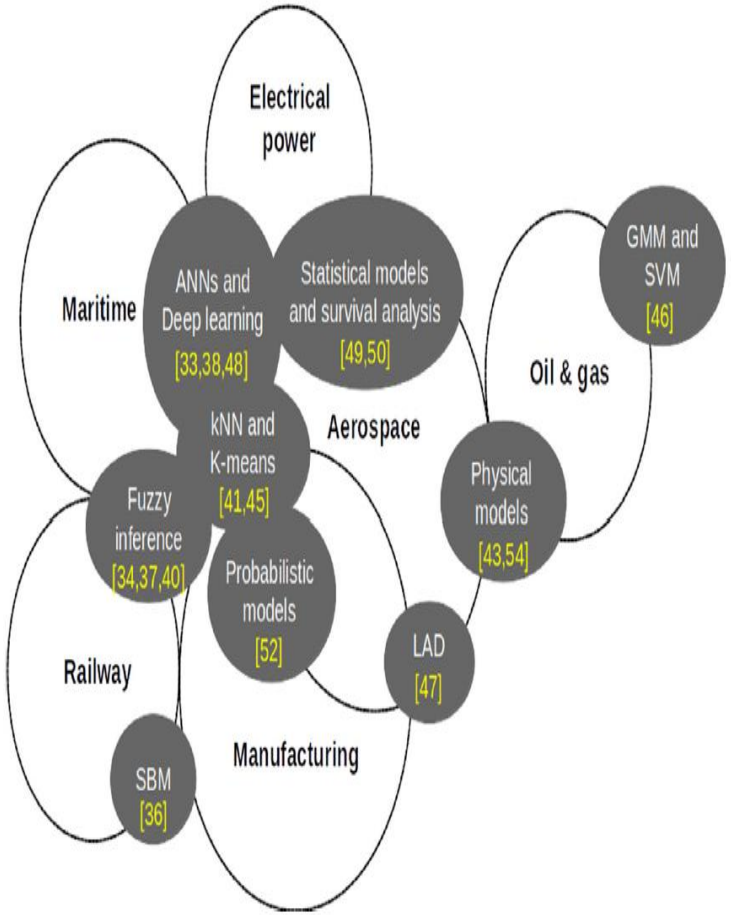


Fig. 5. Solutions and industrial sectors addressed for predictive prognostics.

Information Fusion 50 (2019) 92–111

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Full Length Article

Data fusion and machine learning for industrial prognosis: Trends and perspectives towards Industry 4.0

Alberto Diez-Olivan[a], Javier Del Ser[a,b,c,*], Diego Galar[a,d], Basilio Sierra[e]

[a] TECNALIA, Donostia-San Sebastián 20009, Spain
[b] Department of Communications Engineering, University of the Basque Country (UPV/EHU), Bilbao 48013, Spain
[c] Basque Center for Applied Mathematics (BCAM), Bilbao, Bizkaia 48009, Spain
[d] Department of Civil, Environmental and Natural Resources Engineering, Operation, Maintenance and Acoustics, Luled University of Technology, Luled, Sweden
[e] Department of Computer Sciences and Artificial Intelligence, University of the Basque Country (UPV/EHU), Donostia-San Sebastián 20018, Spain

ARTICLE INFO

Keywords:
Data-driven prognosis
Data fusion
Machine learning
Industry 4.0

ABSTRACT

The so-called "smartization" of manufacturing industries has been conceived as the fourth industrial revolution or Industry 4.0, a paradigm shift propelled by the upsurge and progressive maturity of new Information and Communication Technologies (ICT) applied to industrial processes and products. From a data science perspective, this paradigm shift allows extracting relevant knowledge from monitored assets through the adoption of intelligent monitoring and data fusion strategies, as well as by the application of machine learning and optimization methods. One of the main goals of data science in this context is to effectively predict abnormal behaviors in industrial machinery, tools and processes so as to anticipate critical events and damage, eventually causing important economical losses and safety issues. In this context, data-driven prognosis is gradually gaining attention in different industrial sectors. This paper provides a comprehensive survey of the recent developments in data fusion and machine learning for industrial prognosis, placing an emphasis on the identification of research trends, niches of opportunity and unexplored challenges. To this end, a principled categorization of the utilized feature extraction techniques and machine learning methods will be provided on the basis of its intended purpose: analyze what caused the failure (descriptive), determine when the monitored asset will fail (predictive) or decide what to do so as to minimize its impact on the industry at hand (prescriptive). This threefold analysis, along with a discussion on its hardware and software implications, intends to serve as a stepping stone for future researchers and practitioners to join the community investigating on this vibrant field.

Machine Learning Applications

Information Fusion 53 (2020) 222–239

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Online heart monitoring systems on the internet of health things environments: A survey, a reference model and an outlook

Marcus A.G. Santos[a], Roberto Munoz[b,c], Rodrigo Olivares[d], Pedro P. Rebouças Filho[e], Javier Del Ser[f], Victor Hugo C. de Albuquerque[a,*]

[a] University of Fortaleza, Fortaleza, CE, Brazil
[b] School of Informatics Engineering, Universidad de Valparaíso, Valparaíso, Chile
[c] Centro de Investigación y Desarrollo en Ingeniería en Salud, Universidad de Valparaíso, Valparaíso, Chile
[d] Pontificia Universidad Católica de Valparaíso, Valparaíso, 2362807, Chile
[e] Federal Institute of Education, Science and Technology of Ceará, Fortaleza, CE, Brazil
[f] TECNALIA, Donostia-San Sebastián, Spain. Department of Communications Engineering, University of the Basque Country, Bilbao, Spain. Basque Center for Applied Mathematics, Bilbao, Bizkaia, Spain

ARTICLE INFO

Keywords:
Internet of health things
Heart
Bio sensors
Online monitoring
Reference model

ABSTRACT

The Internet of Health Things promotes personalized and higher standards of care. Its application is diverse and attracts the attention of a substantial section of the scientific community. This approach has also been applied by people looking to enhance quality of life by using this technology. In this paper, we perform a survey that aims to present and analyze the advances of the latest studies based on medical care and assisted environment. We focus on articles for online monitoring, detection, and support of the diagnosis of cardiovascular diseases. Our research covers published manuscripts in scientific journals and recognized conferences since the year 2015. Also, we present a reference model based on the evaluation of the resources used from the selected studies. Finally, our proposal aims to help future enthusiasts to discover and enumerate the required factors for the development of a prototype for online heart monitoring purposes.

Machine Learning Applications

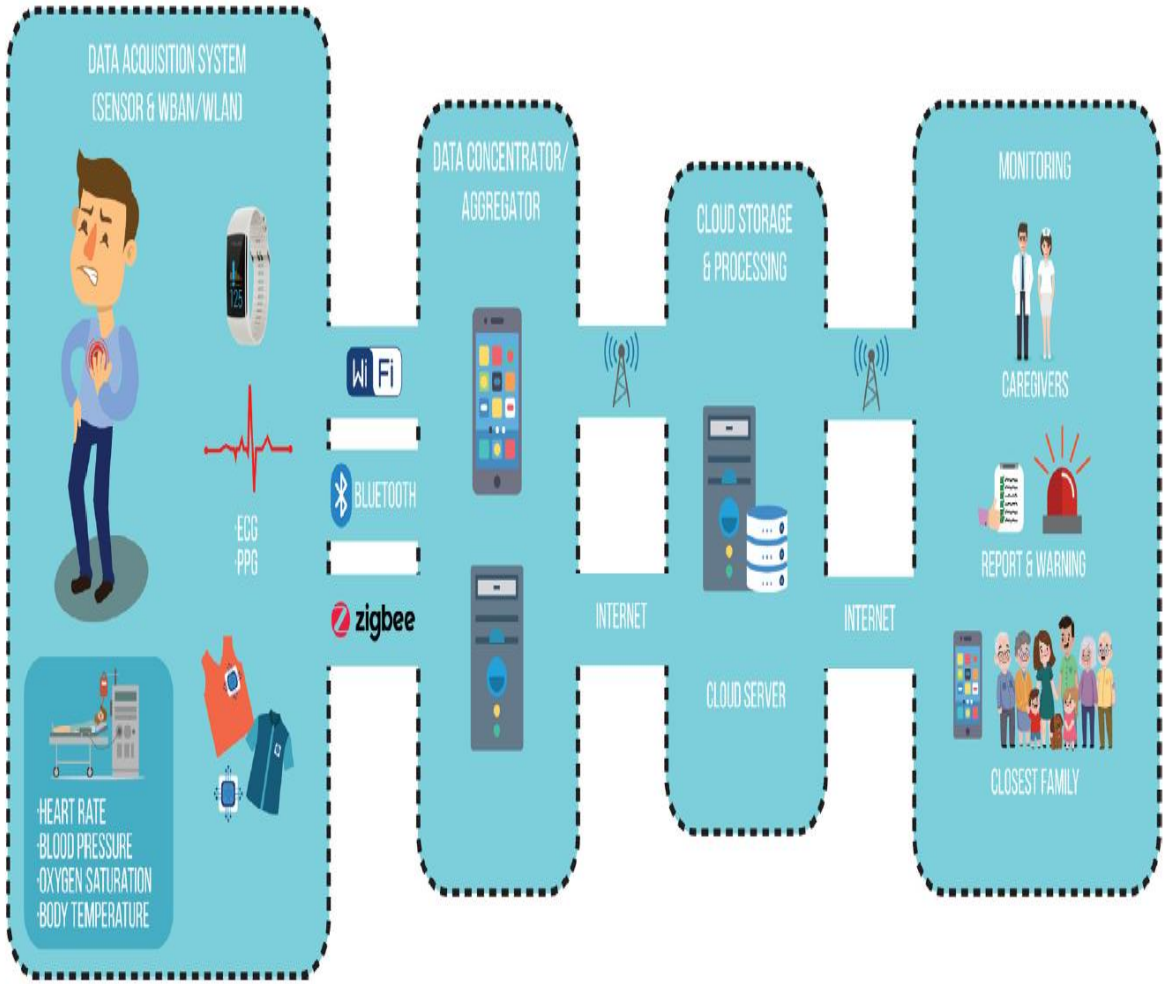


Fig. 1. IoT of the Heart.

Information Fusion 53 (2020) 222–239

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Online heart monitoring systems on the internet of health things environments: A survey, a reference model and an outlook

Marcus A.G. Santos[a], Roberto Munoz[b,c], Rodrigo Olivares[d], Pedro P. Rebouças Filho[e], Javier Del Ser[f], Victor Hugo C. de Albuquerque[a,*]

[a] University of Fortaleza, Fortaleza, CE, Brazil
[b] School of Informatics Engineering, Universidad de Valparaíso, Valparaíso, Chile
[c] Centro de Investigación y Desarrollo en Ingeniería en Salud, Universidad de Valparaíso, Valparaíso, Chile
[d] Pontificia Universidad Católica de Valparaíso, Valparaíso, 2362807, Chile
[e] Federal Institute of Education, Science and Technology of Ceará, Fortaleza, CE, Brazil
[f] TECNALIA, Donostia-San Sebastián, Spain. Department of Communications Engineering, University of the Basque Country, Bilbao, Spain. Basque Center for Applied Mathematics, Bilbao, Bizkaia, Spain

ARTICLE INFO

Keywords:
Internet of health things
Heart
Bio sensors
Online monitoring
Reference model

ABSTRACT

The Internet of Health Things promotes personalized and higher standards of care. Its application is diverse and attracts the attention of a substantial section of the scientific community. This approach has also been applied by people looking to enhance quality of life by using this technology. In this paper, we perform a survey that aims to present and analyze the advances of the latest studies based on medical care and assisted environment. We focus on articles for online monitoring, detection, and support of the diagnosis of cardiovascular diseases. Our research covers published manuscripts in scientific journals and recognized conferences since the year 2015. Also, we present a reference model based on the evaluation of the resources used from the selected studies. Finally, our proposal aims to help future enthusiasts to discover and enumerate the required factors for the development of a prototype for online heart monitoring purposes.

Machine Learning Applications

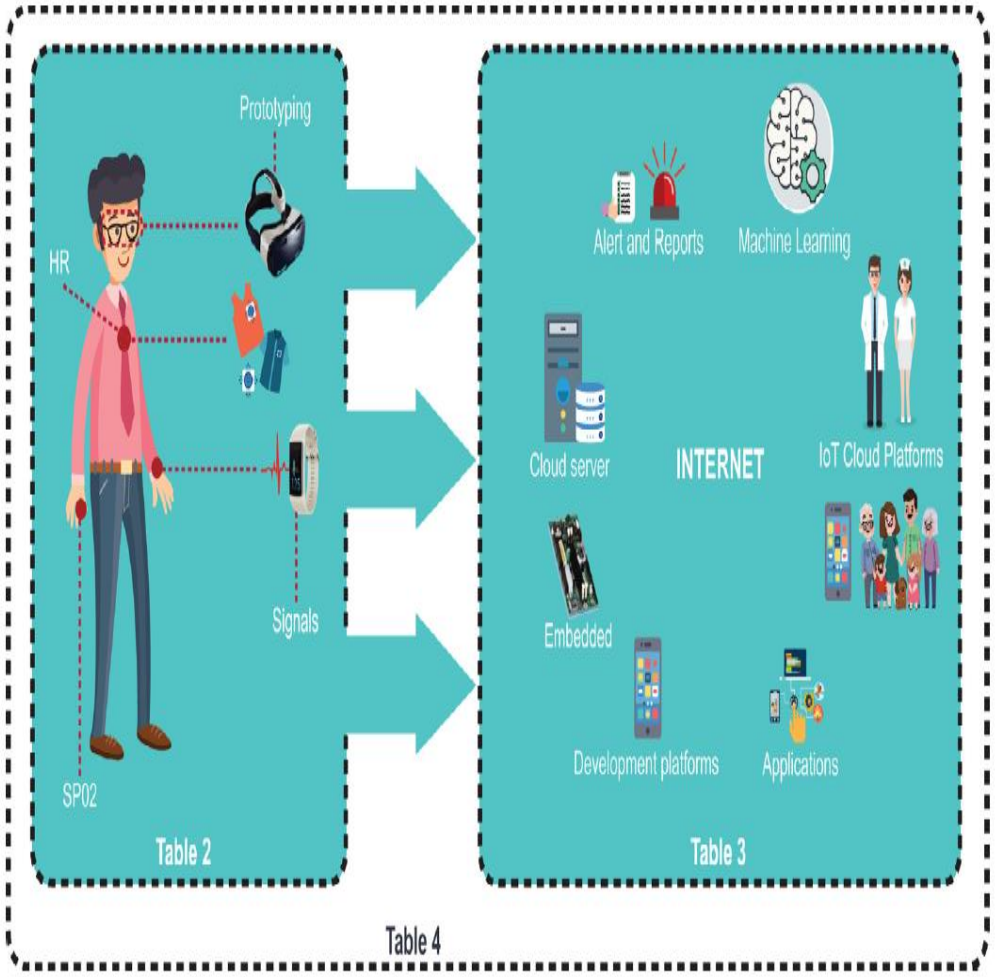


Fig. 2. Development framework.

Information Fusion 53 (2020) 222–239

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Online heart monitoring systems on the internet of health things environments: A survey, a reference model and an outlook

Marcus A.G. Santos[a], Roberto Munoz[b,c], Rodrigo Olivares[d], Pedro P. Rebouças Filho[e], Javier Del Ser[f], Victor Hugo C. de Albuquerque[a,*]

[a] University of Fortaleza, Fortaleza, CE, Brazil
[b] School of Informatics Engineering, Universidad de Valparaíso, Valparaíso, Chile
[c] Centro de Investigación y Desarrollo en Ingeniería en Salud, Universidad de Valparaíso, Valparaíso, Chile
[d] Pontificia Universidad Católica de Valparaíso, Valparaíso, 2362807, Chile
[e] Federal Institute of Education, Science and Technology of Ceará, Fortaleza, CE, Brazil
[f] TECNALIA, Donostia-San Sebastián, Spain. Department of Communications Engineering, University of the Basque Country, Bilbao, Spain. Basque Center for Applied Mathematics, Bilbao, Bizkaia, Spain

ARTICLE INFO

Keywords:
Internet of health things
Heart
Bio sensors
Online monitoring
Reference model

ABSTRACT

The Internet of Health Things promotes personalized and higher standards of care. Its application is diverse and attracts the attention of a substantial section of the scientific community. This approach has also been applied by people looking to enhance quality of life by using this technology. In this paper, we perform a survey that aims to present and analyze the advances of the latest studies based on medical care and assisted environment. We focus on articles for online monitoring, detection, and support of the diagnosis of cardiovascular diseases. Our research covers published manuscripts in scientific journals and recognized conferences since the year 2015. Also, we present a reference model based on the evaluation of the resources used from the selected studies. Finally, our proposal aims to help future enthusiasts to discover and enumerate the required factors for the development of a prototype for online heart monitoring purposes.

Introduction of Deep Learning

Unsupervised Learning Algorithms Or Generative Algorithms

- Autoencoders
  - Stack Autoencoders
  - Denoising Autoencoders
  - Sparse Autoencoders
  - Deep Autoencoders
- Boltzmann Machine (BM)
  - Restricted Boltzmann Machine (RBM)
  - Deep Belief Network (DBN)
- Variational Autoencoders (VEs)
- Generative adversarial networks(GAN)

Could be

- Recurrent Neural Network (RNN)
- Long Short-Term Memory Network (LSTM)

Supervised Learning Algorithms

- Artificial Neural Network (ANN)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Long Short-Term Memory Network (LSTM)
- Gated Recurrent Unit (GRU)

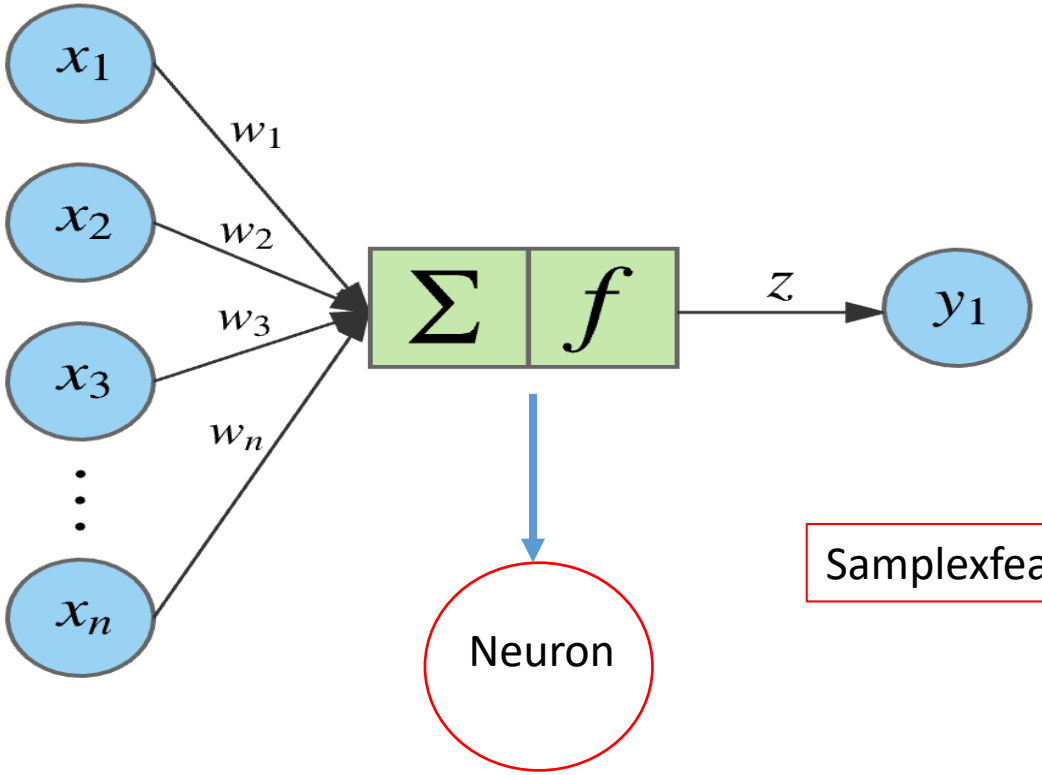
Could be

- Autoencoders and their variants

Artificial Neural Network (ANN)

How NN works?

Artificial Neural Networks (ANN) are multi-layer fully-connected neural nets that look like the figure below. They consist of an input layer, multiple hidden layers, and an output layer. Every node in one layer is connected to every other node in the next layer. We make the network deeper by increasing the number of hidden layers.



$$z = f(x \cdot w) = f\left(\sum_{i=1}^{n} x_i w_i\right)$$

$$x \in d_{1\times n},\ w \in d_{n\times 1},\ z \in d_{1\times 1}$$

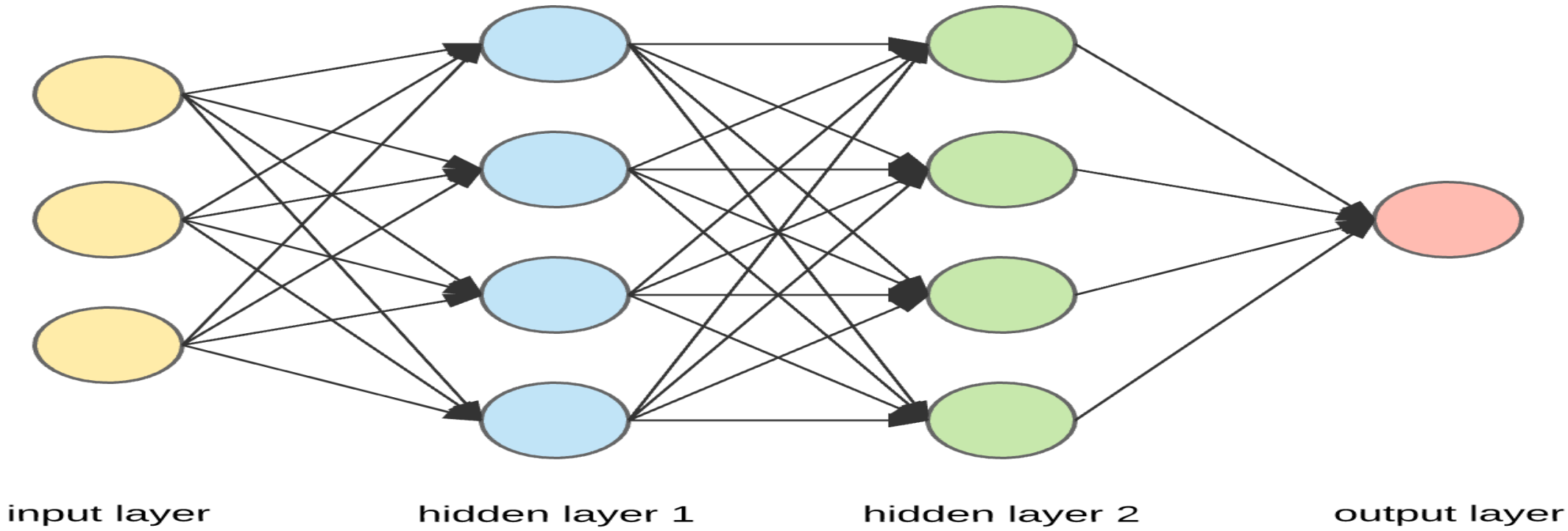
Samplexfeatures=1xn

$$z = f(b + x \cdot w) = f\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

$$x \in d_{1\times n},\ w \in d_{n\times 1},\ b \in d_{1\times 1},\ z \in d_{1\times 1}$$

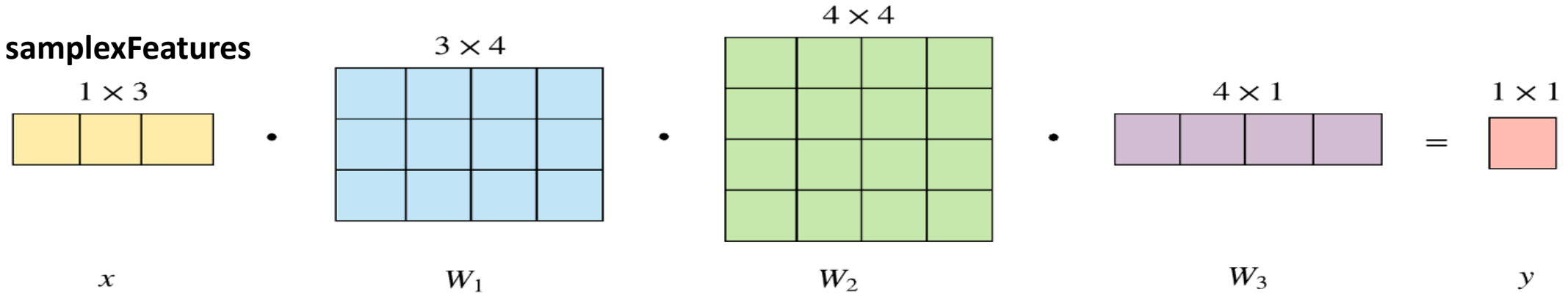
How NN works?

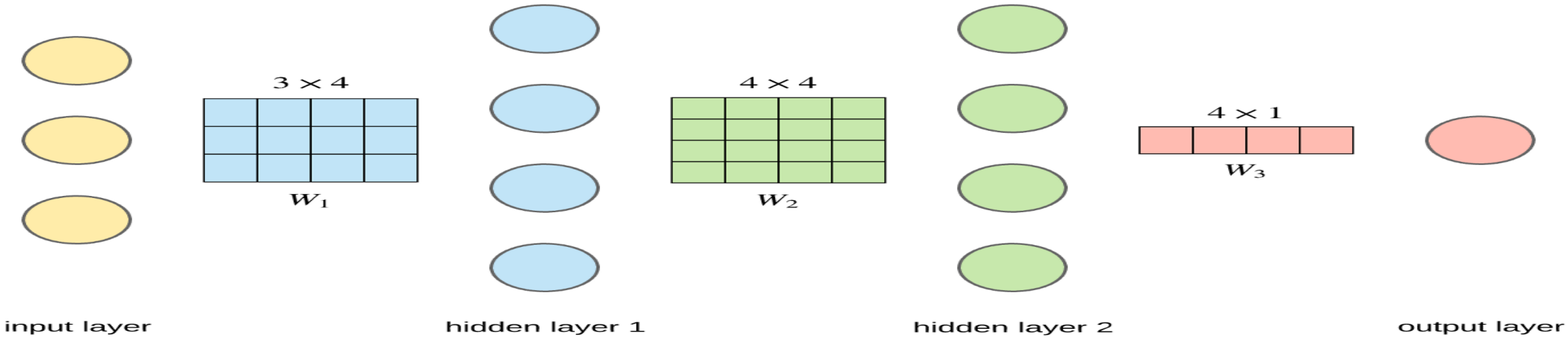
Artificial Neural Networks (ANN) are multi-layer fully-connected neural nets that look like the figure below. They consist of an input layer, multiple hidden layers, and an output layer. Every node in one layer is connected to every other node in the next layer. We make the network deeper by increasing the number of hidden layers.



How NN works?

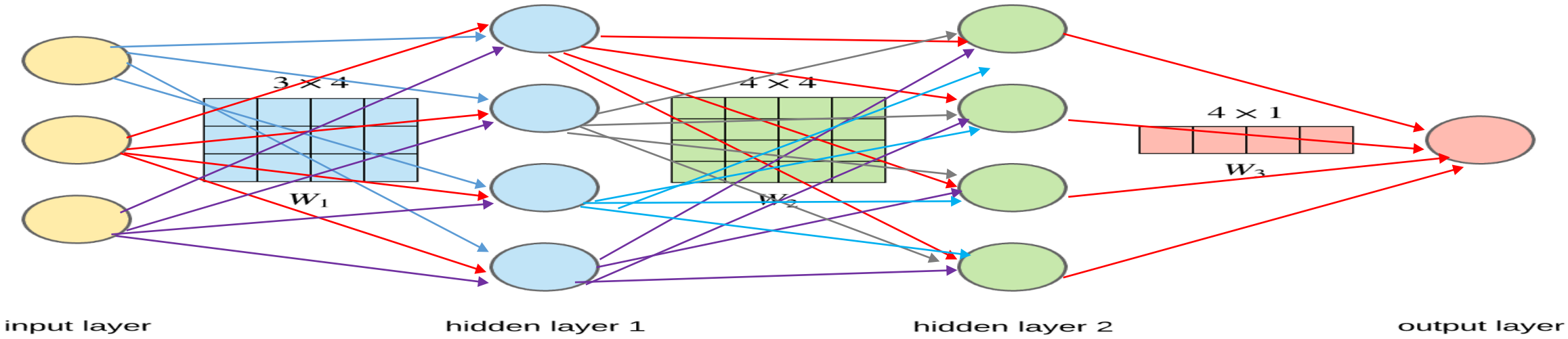
Training procedure works as follows:





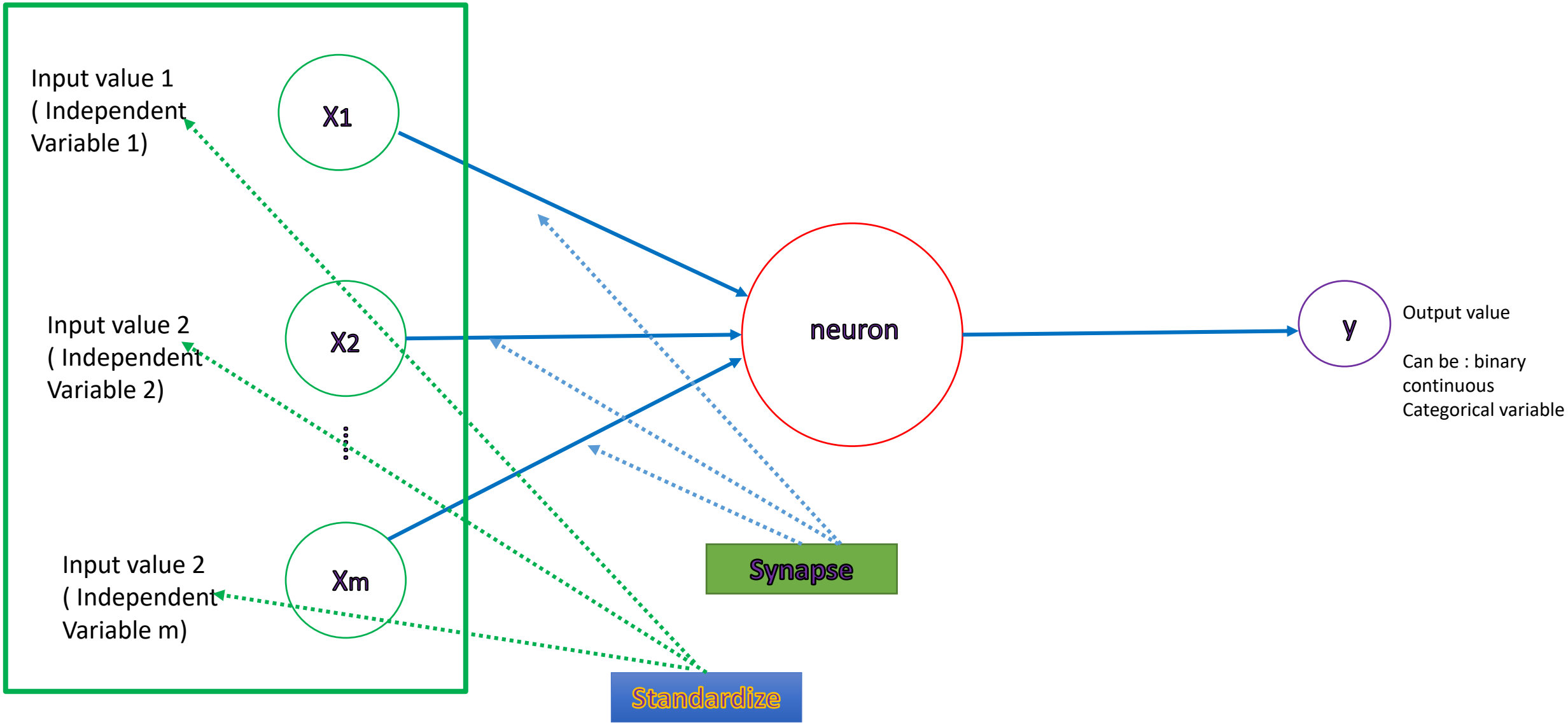
How NN works?

Training procedure works as follows:



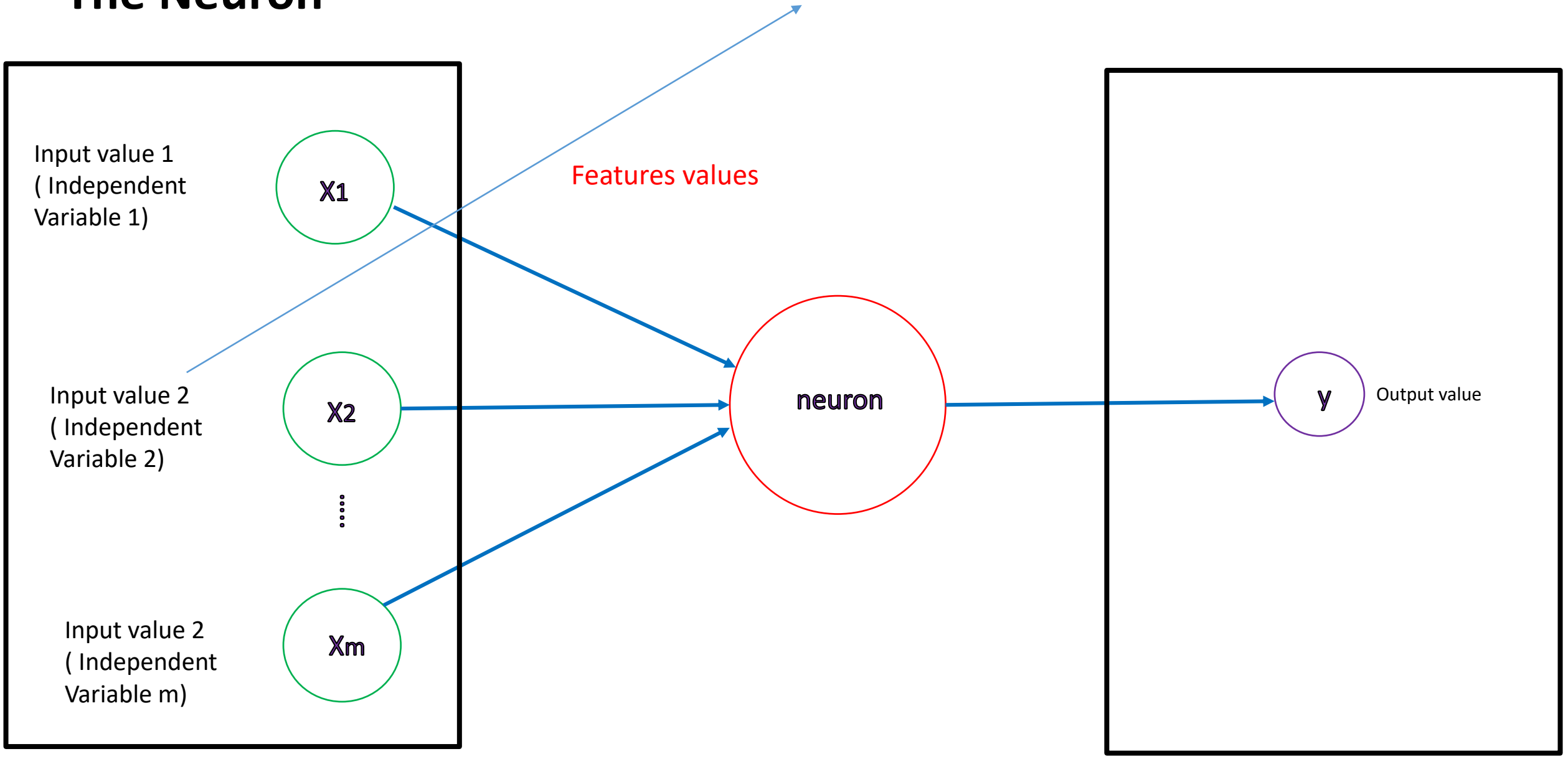
The Neuron

All independent variable is called one observation or one example or one sample



The Neuron

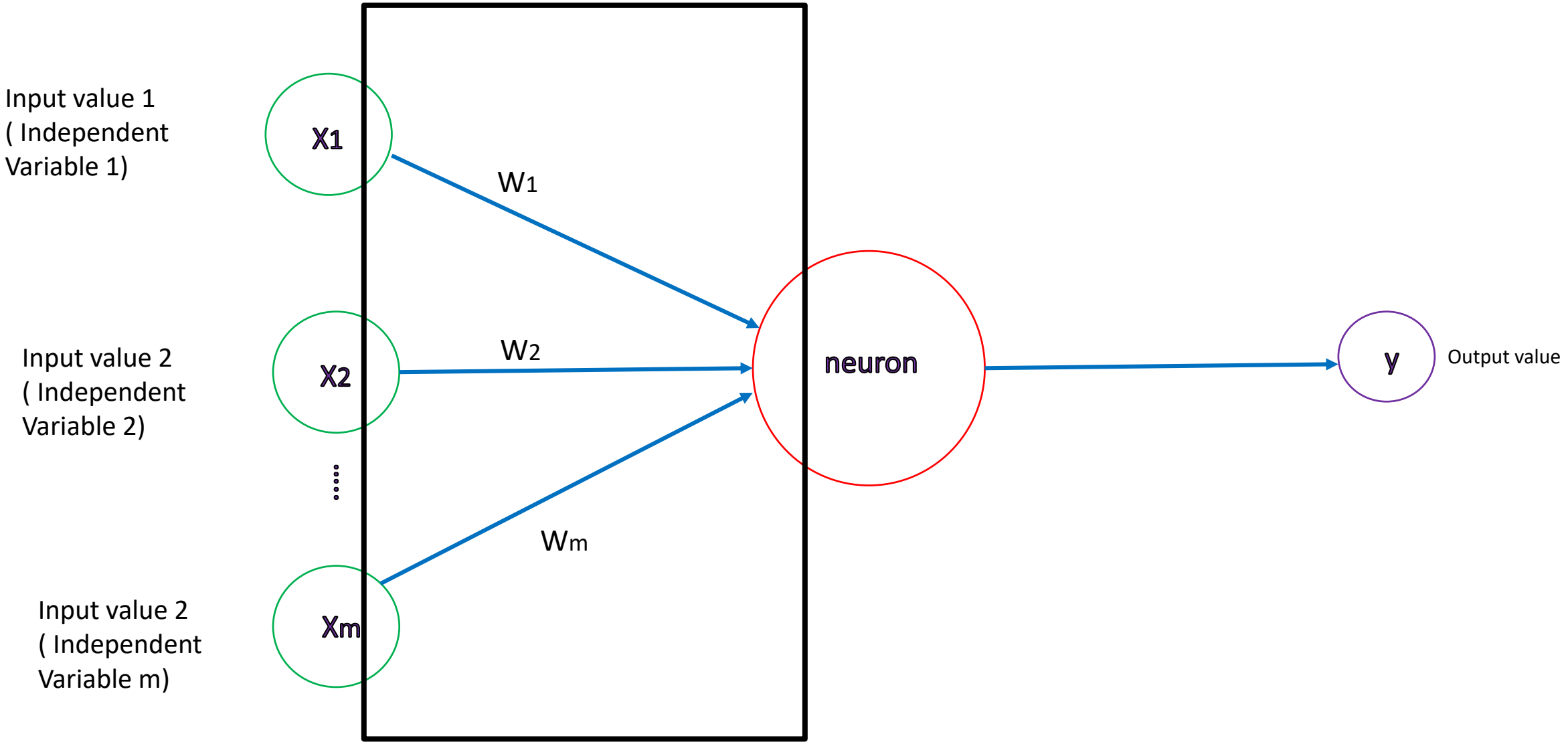
All independent variable is called one observation or one example or one sample



Same observation

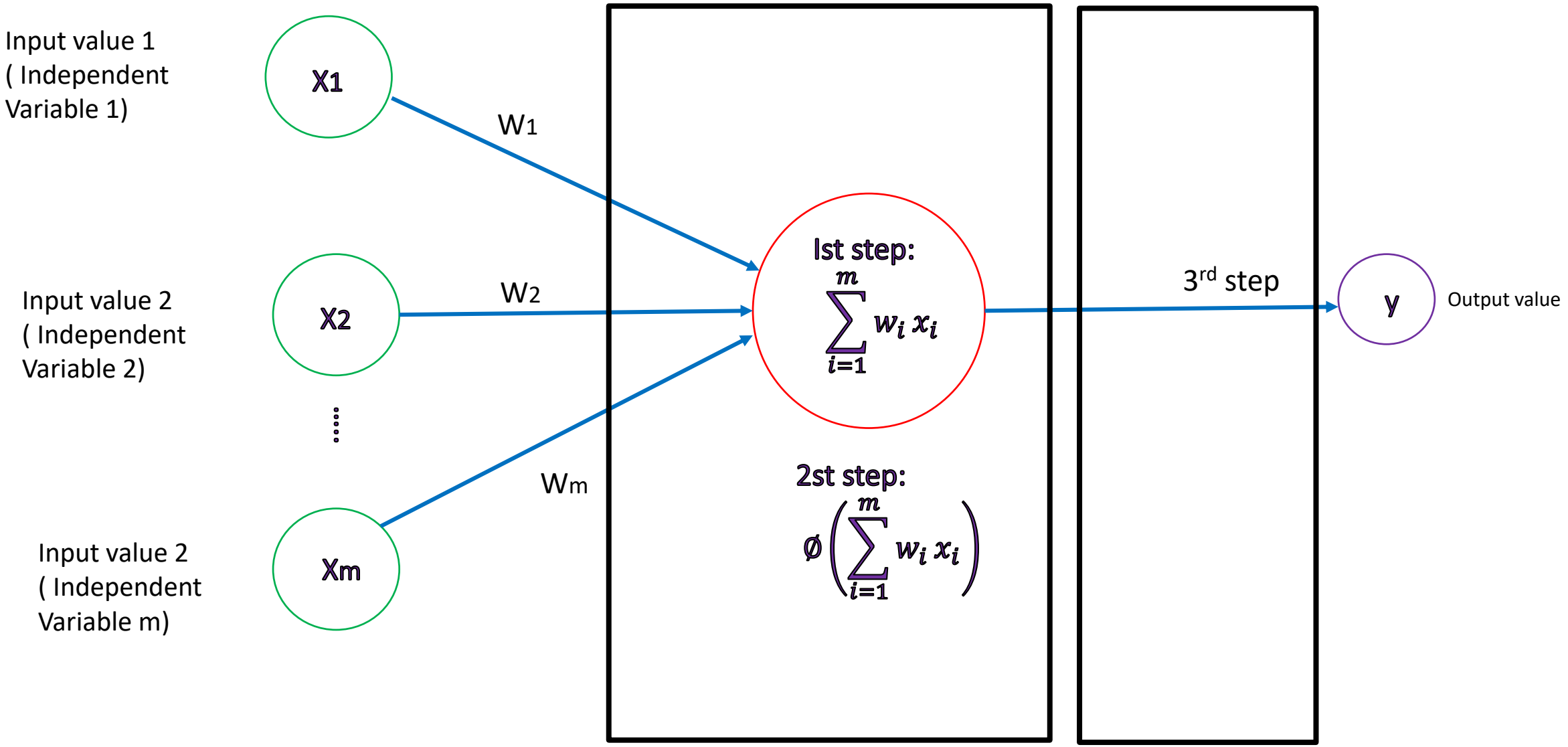
Single Observation ⟷ Single Observation

The Neuron

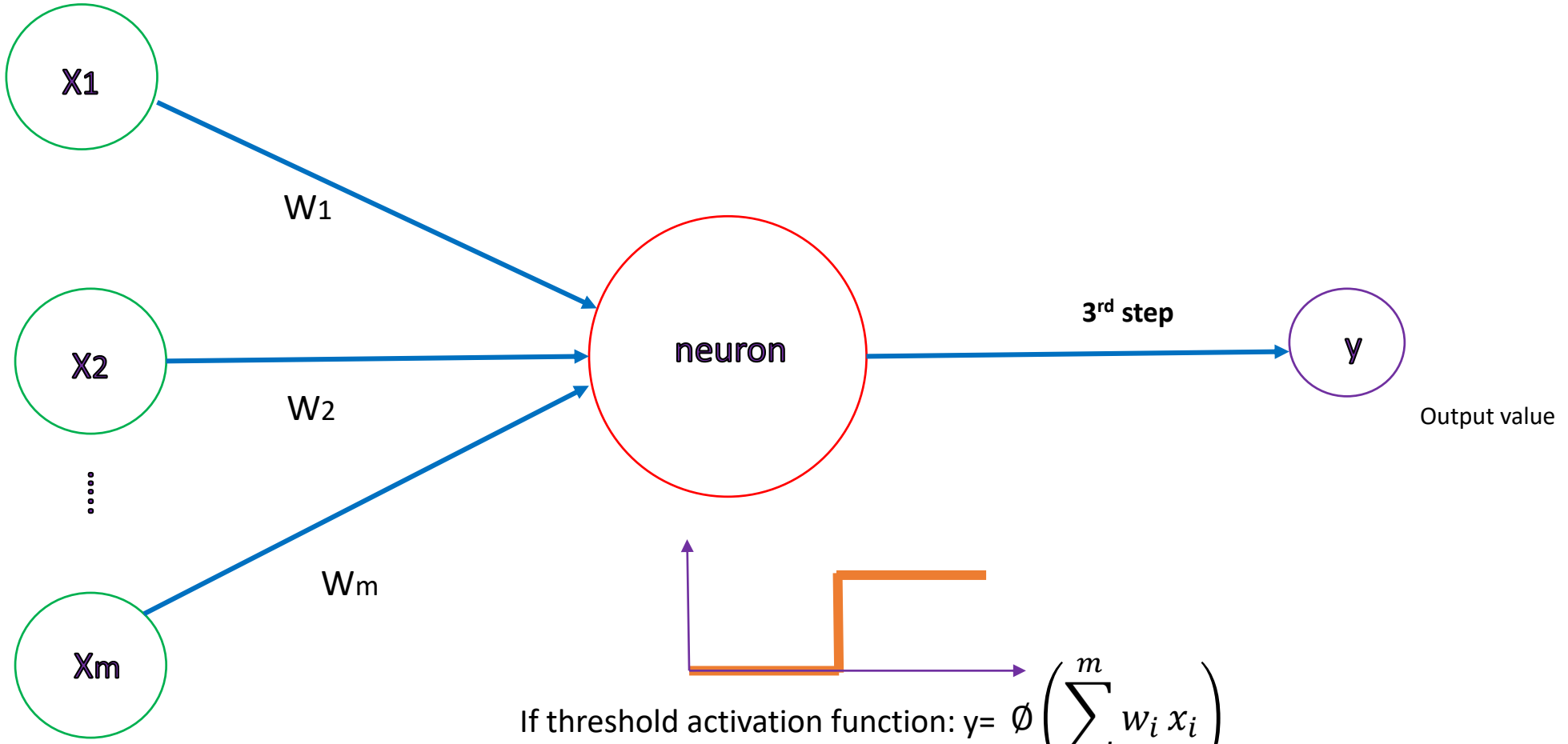

Input value 1
(Independent
Variable 1)
X1
W1
Input value 2
(Independent
Variable 2)
X2
W2
neuron
y
Output value
Wm
Input value 2
(Independent
Variable m)
Xm

The Neuron

The weighted sum of the input is passed to the activation



Neural Network



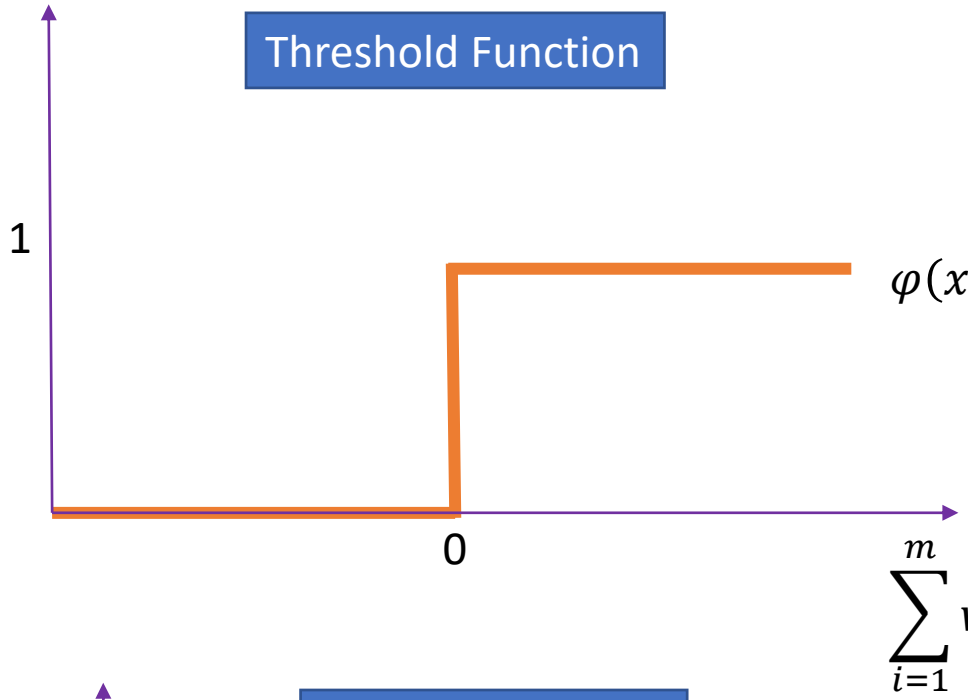
If threshold activation function: y= $\emptyset\left(\sum_{i=1}^{m} w_i\, x_i\right)$

Assuming the dependent variable is binary (y=0 or 1)

If sigmoid activation function: y= $P(y=1) = \emptyset\left(\sum_{i=1}^{m} w_i\, x_i\right)$

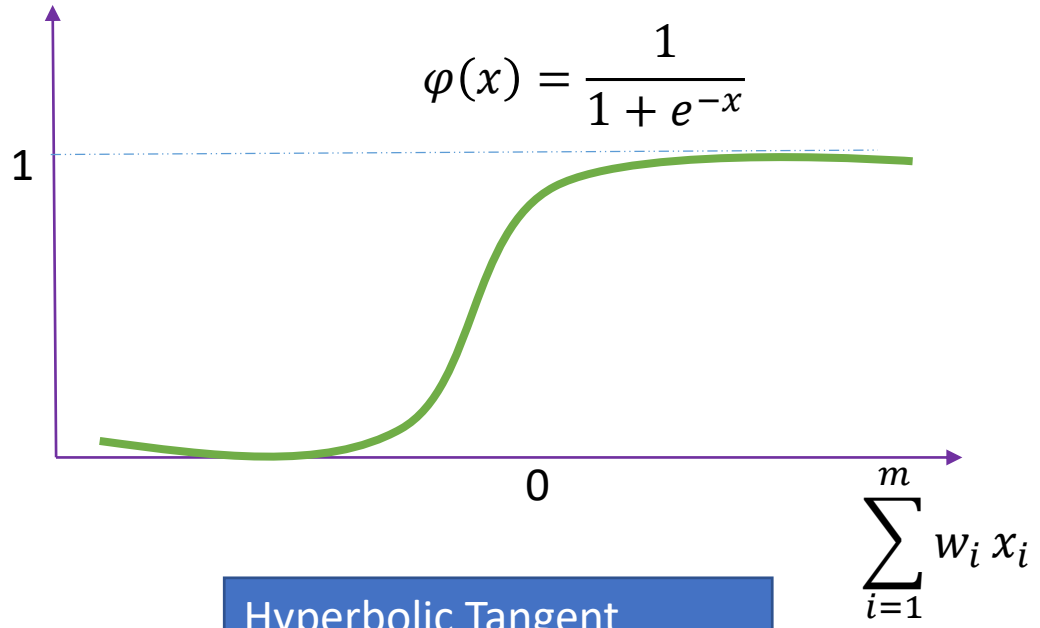
Activation Functions

Threshold Function



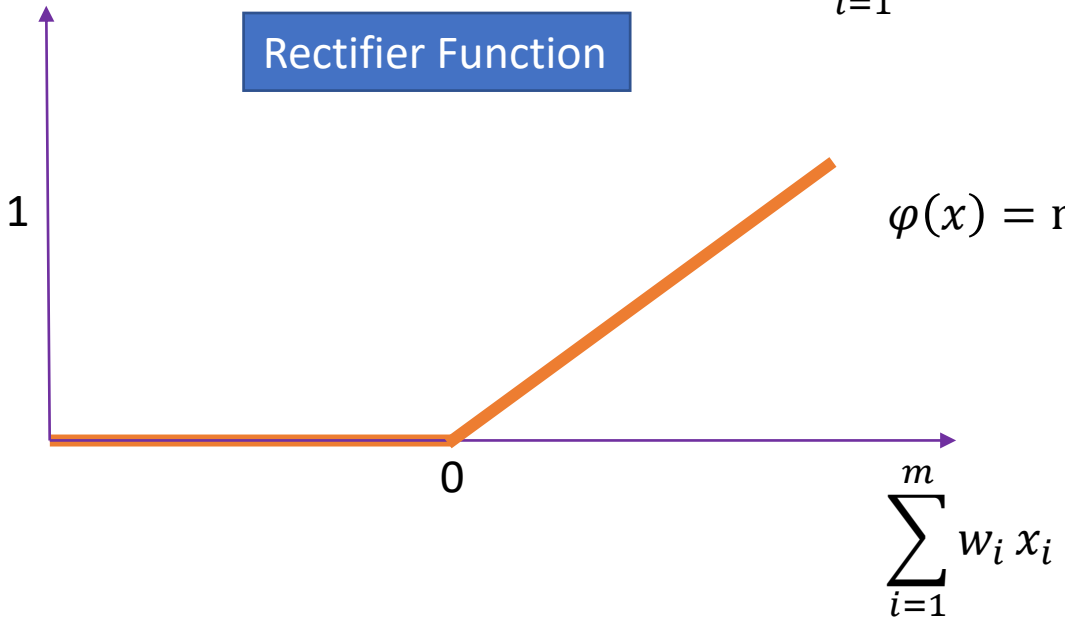
$$\varphi(x) = \begin{cases} 1 \; if \; x \geq 0 \\ 0 \; if \; x < 0 \end{cases}$$

Sigmoid Function



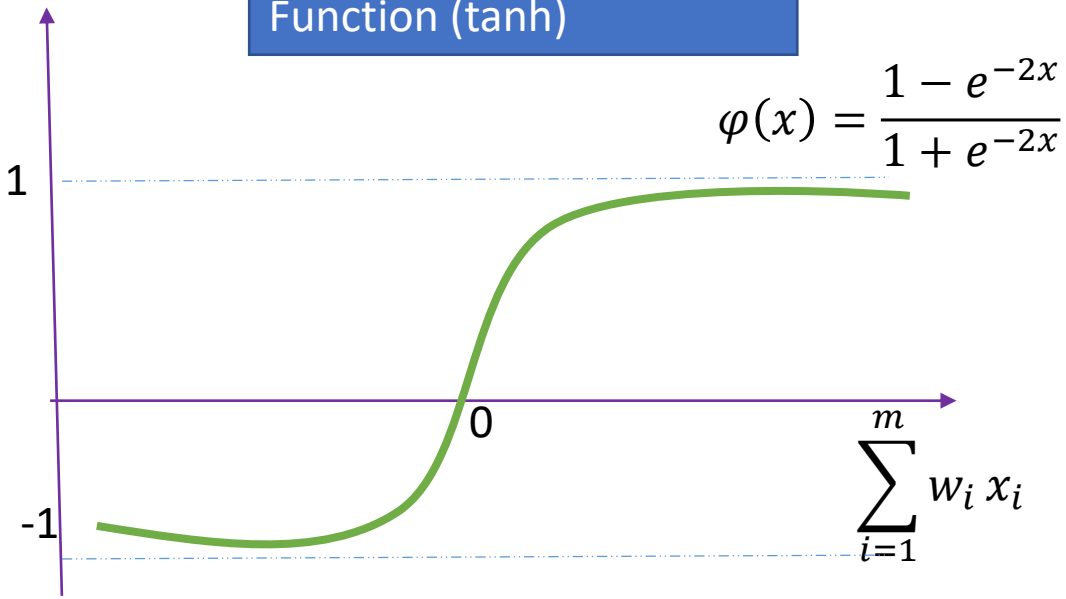
$$\varphi(x) = \frac{1}{1 + e^{-x}}$$

Rectifier Function



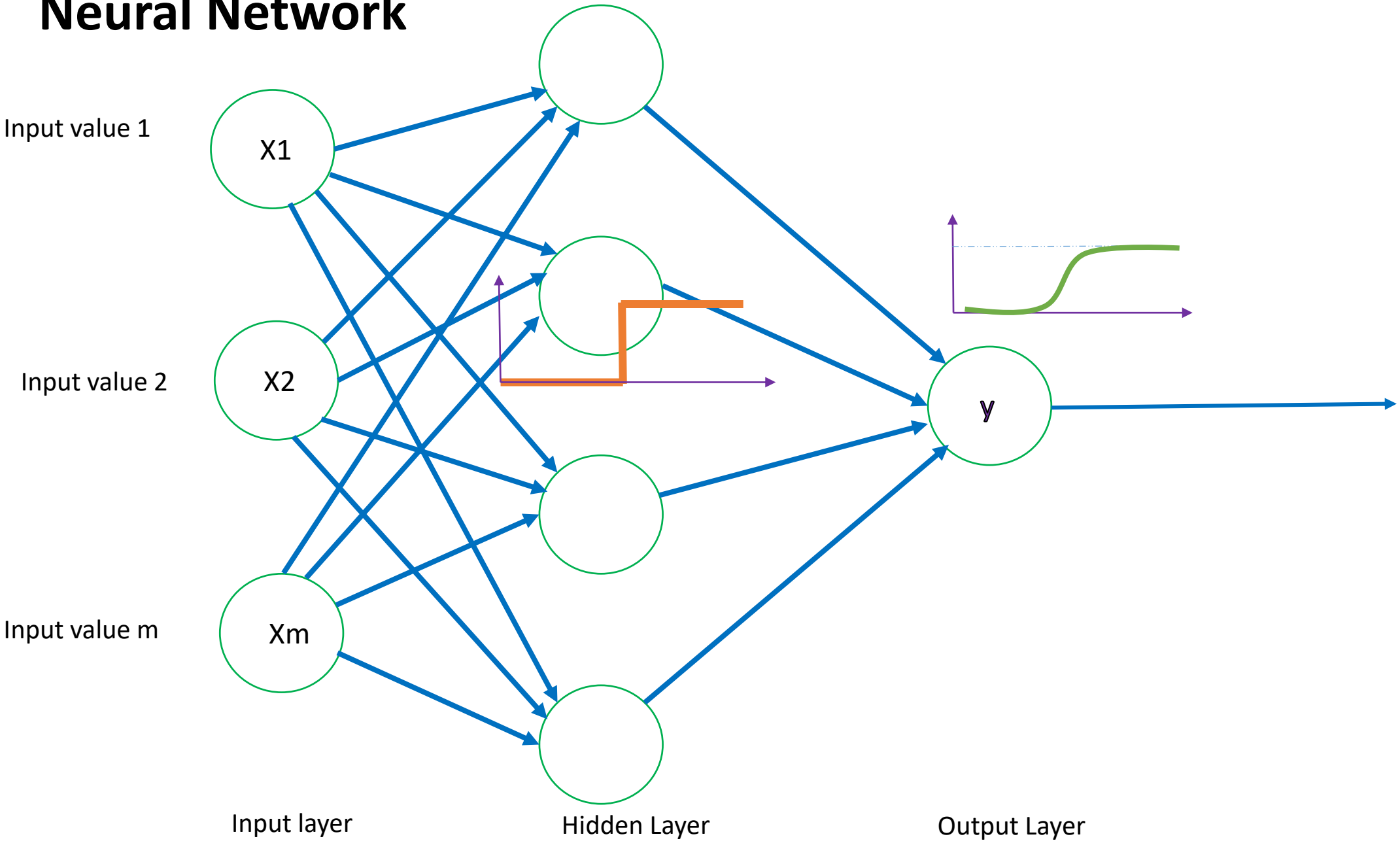
$$\varphi(x) = \max(x, 0)$$

Hyperbolic Tangent Function (tanh)

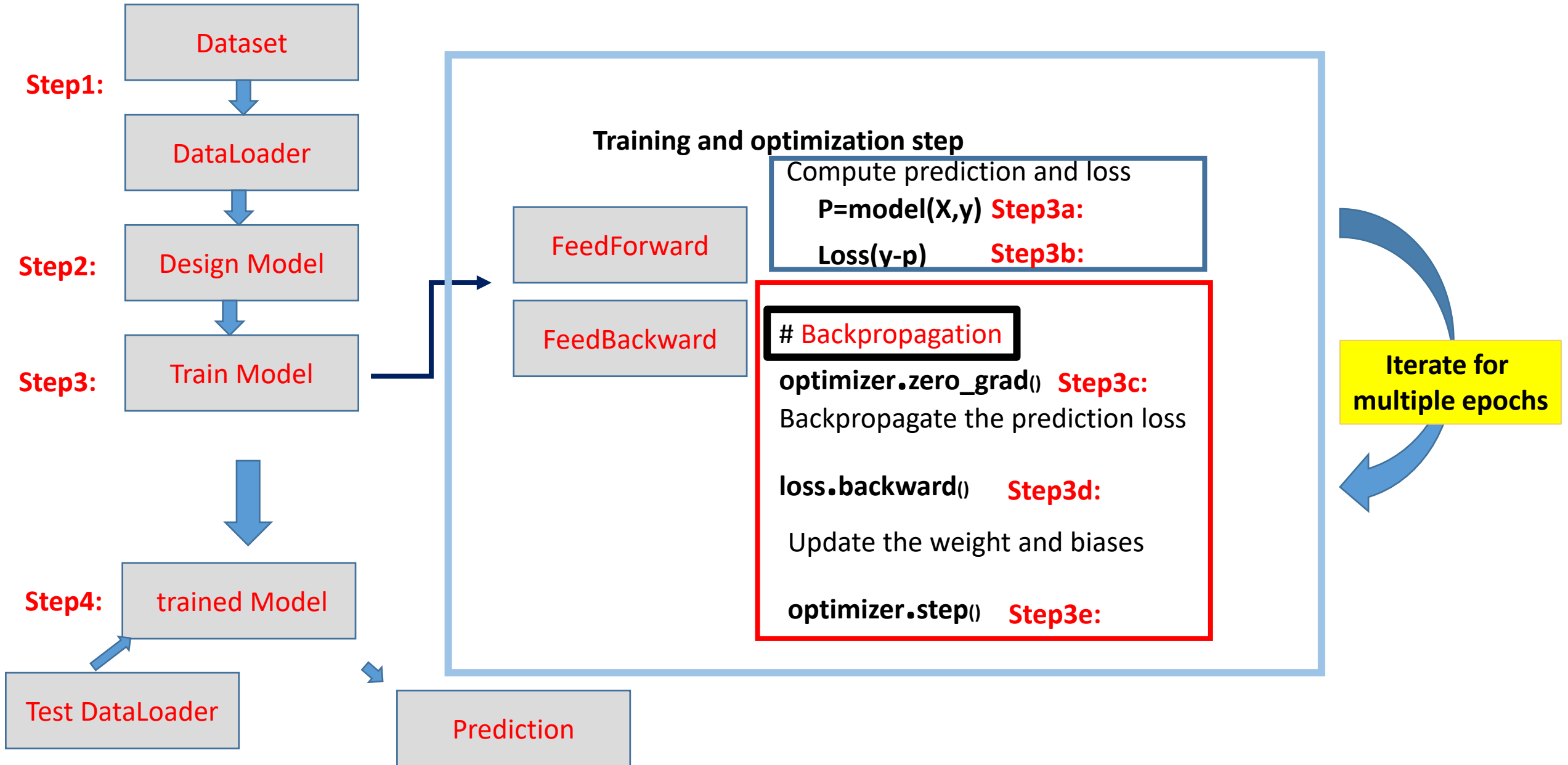


$$\varphi(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

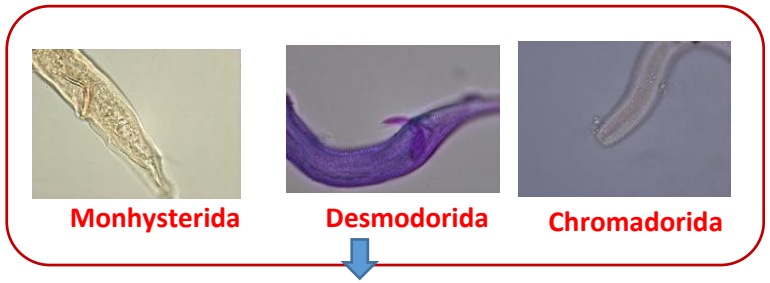
Neural Network



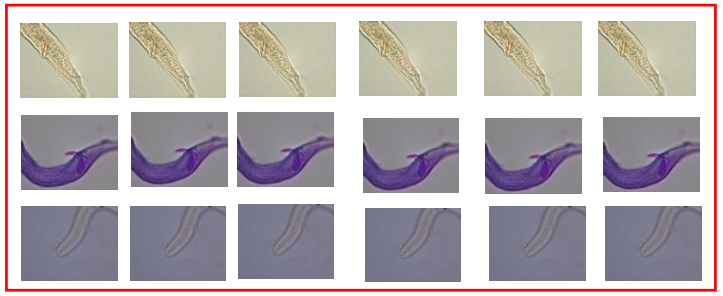
Training Deep Learning model



Training and Optimization



Convert into batches



18 images (18x3x224x224)

Feedforward

1
X
(18xfeatures)
18

ANN Model

P=18x3

	C1	c2	c3
1			
18			

How to compute loss function

Y=18x1,P=18x3

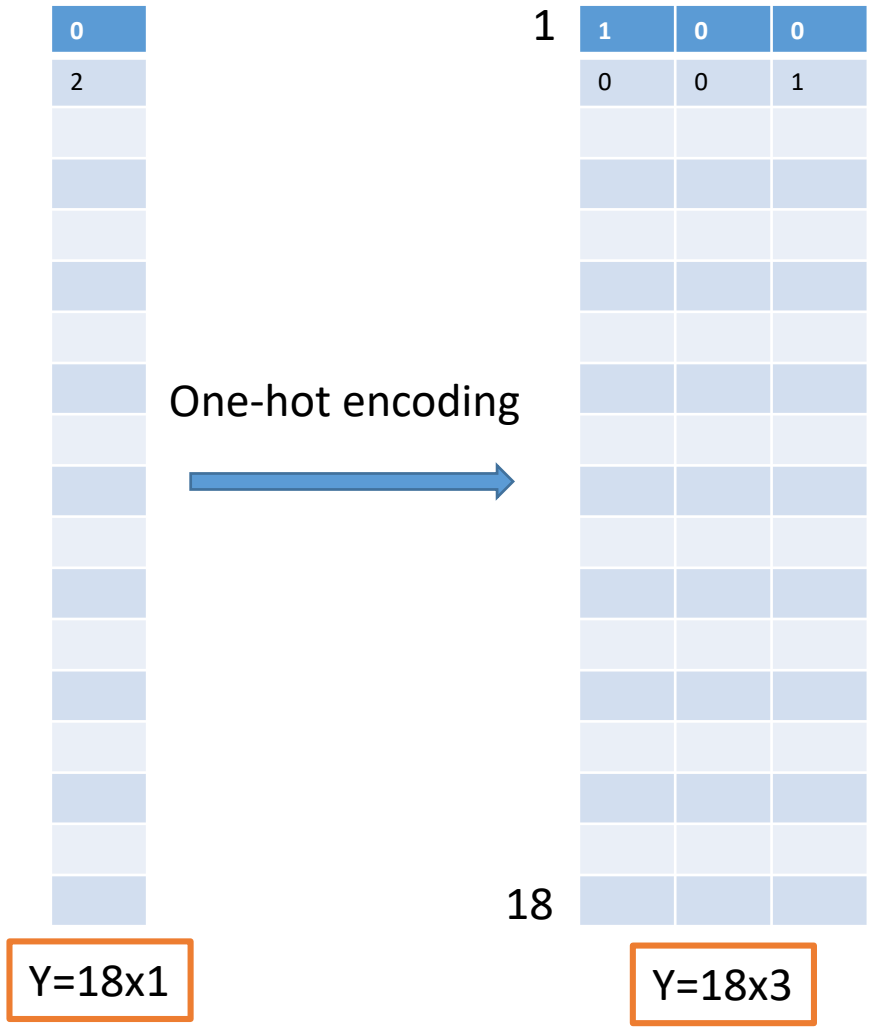
Y= 18

Feed backward and update the Gradient

Loss or error=(Y-P)

Perform a backwards pass from right to left and propagate the error to every individual node using backpropagation. Calculate each weight's contribution to the error, and adjust the weights accordingly using gradient descent. Propagate the error gradients back starting from the last layer.

Training and Optimization



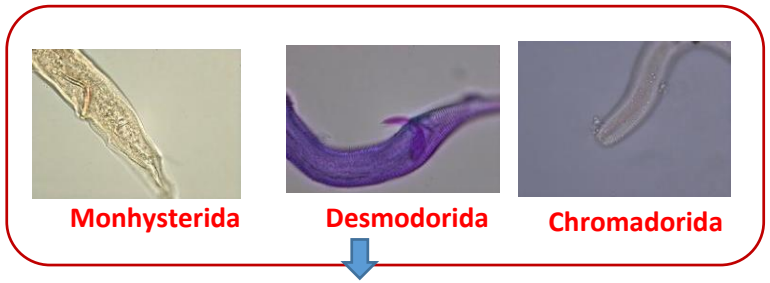
One-hot encoding vector for each label

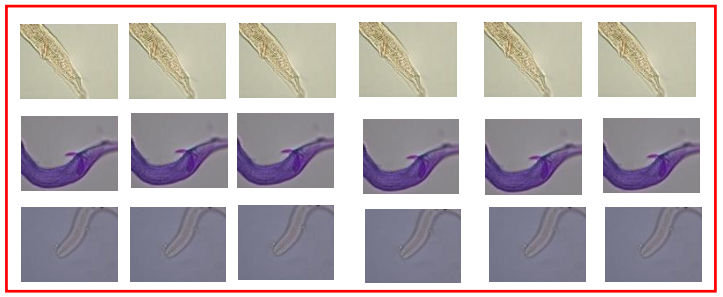
0	1, 0, 0
1	0, 1, 0
2	0, 0, 1

How to compute loss function

Y=18x1,P=18x3

Training and Optimization





18 images (18x3x224x224)

Feedforward

1

18

X
(18xfeatures)

ANN Model

P=18x3

Y=

18

P

1

18

C1 c2 c3

Loss or error=(Y-P)

Feedbackward and update the Grdaient

Y

On-hot encoding

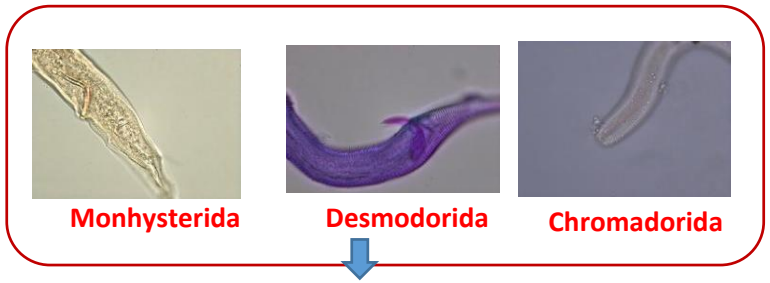
C1 c2 c3

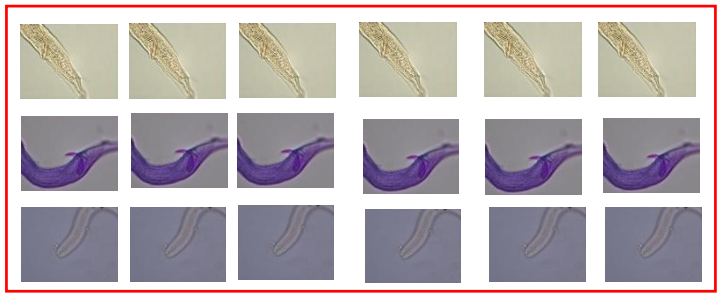
How to compute loss function

Y=18x3,P=18x3

Perform a backwards pass from right to left and propagate the error to every individual node using backpropagation. Calculate each weight's contribution to the error, and adjust the weights accordingly using gradient descent. Propagate the error gradients back starting from the last layer.

Testing the Trained Model





12 images (12x3x224x224)

1
12
Xtest

Feedforward

Trained ANN model

P=12x3

Probability

	C1	c2	c3
1	0.9	0.3	0.6
	0.4	0.5	0.8
	0.8	0.6	0.4
12	0.3	0.7	0.4

maximum

prediction

0
2
0
1

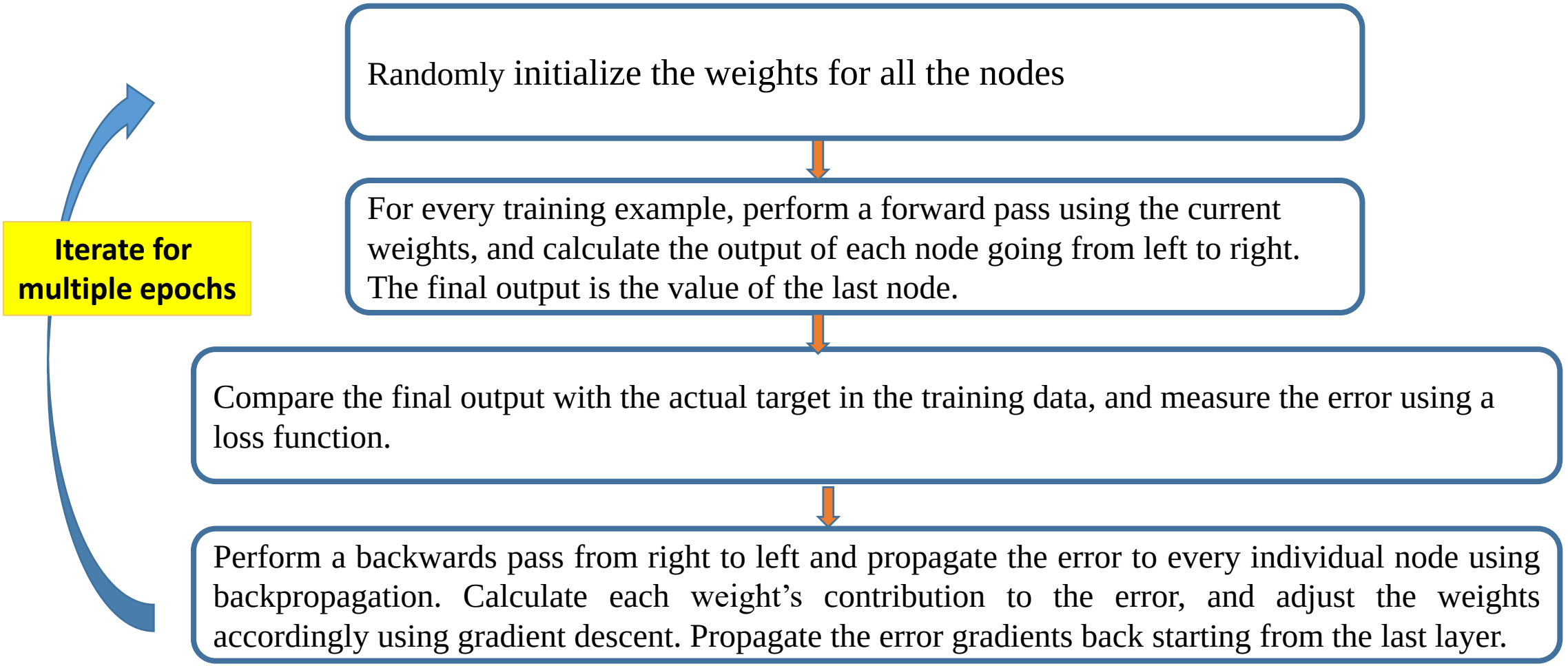
misclassify

1 Test sample ground truth labels 12

0	2	1	0														1

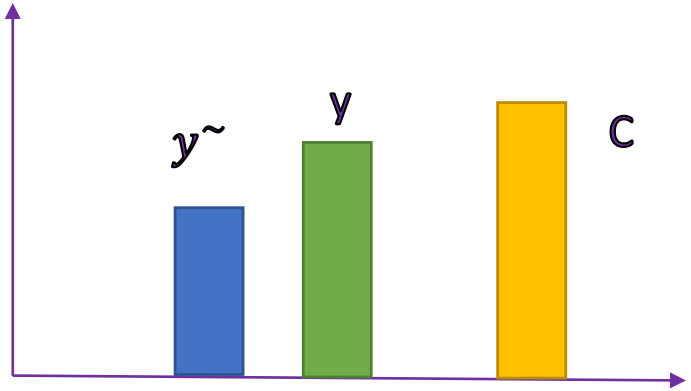
How NN works?

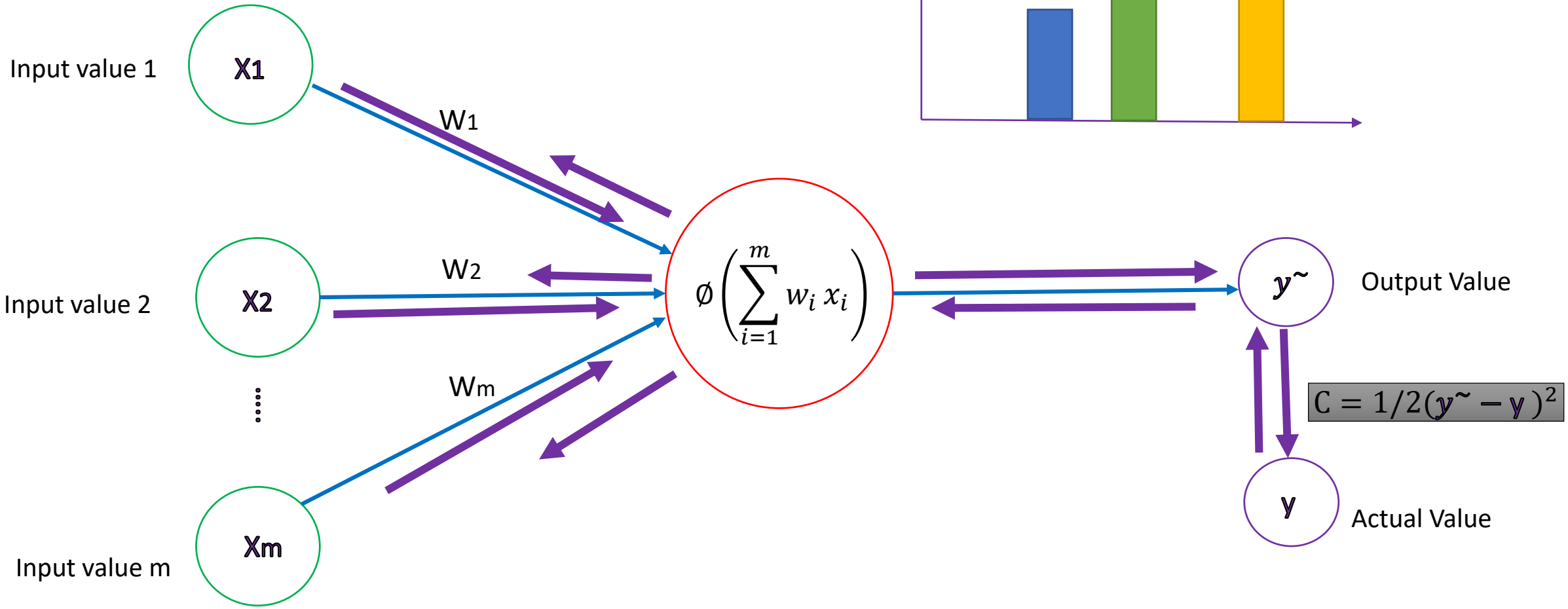
Training procedure works as follows:



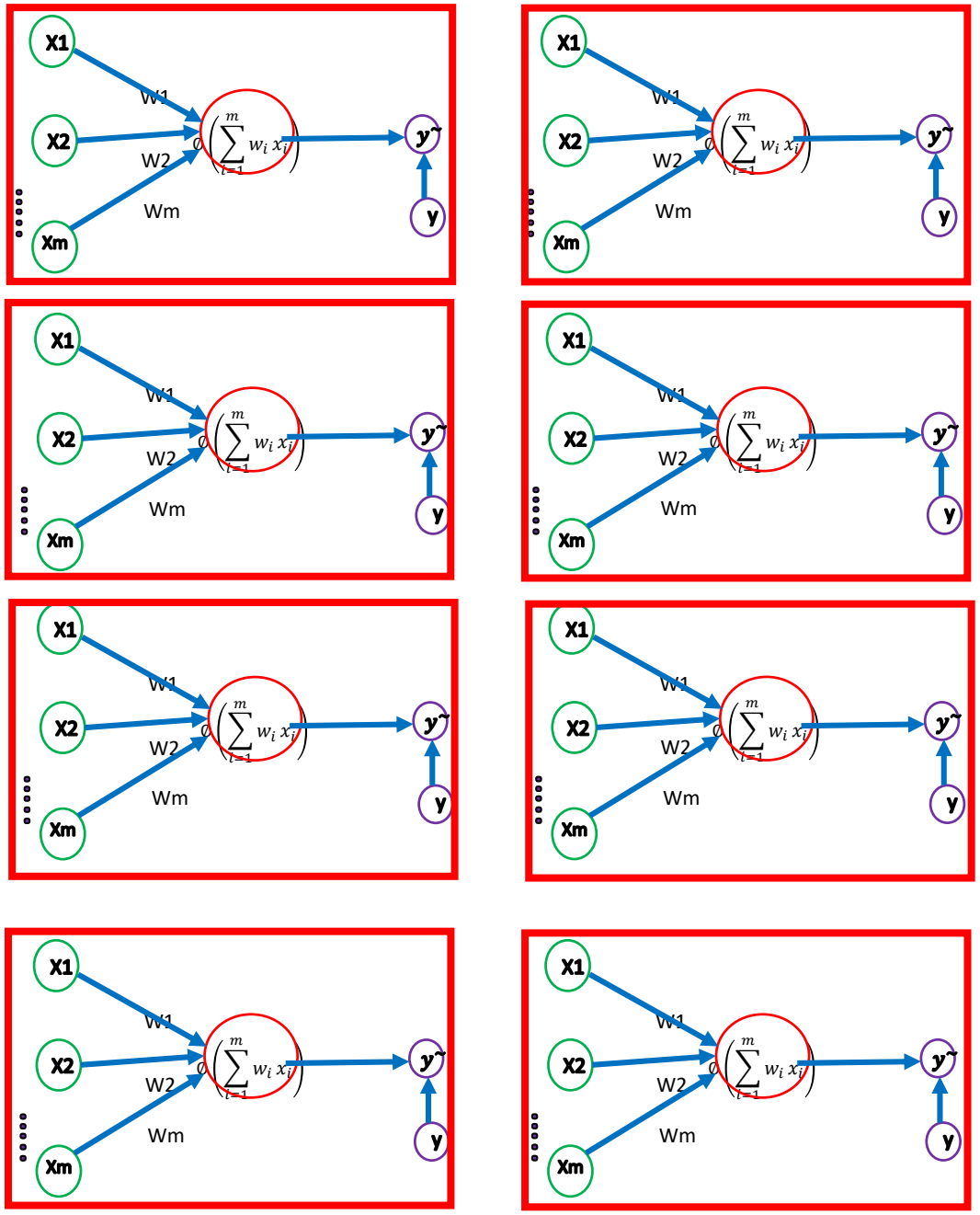
How NN works?

Row ID	Study HRS	Sleep HRS	Quiz	Exam
1	12	6	78%	93%



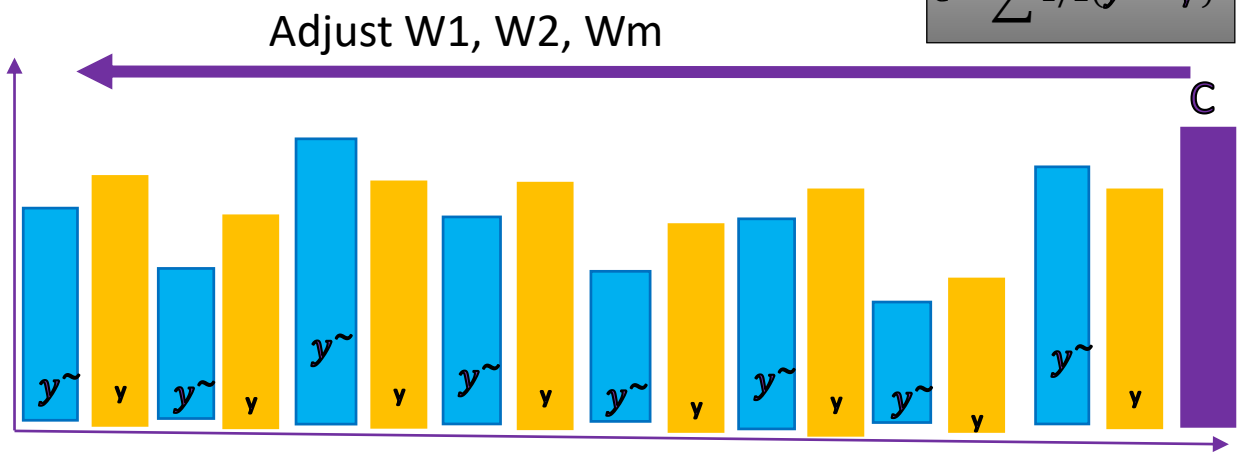


How NN works? An Example:



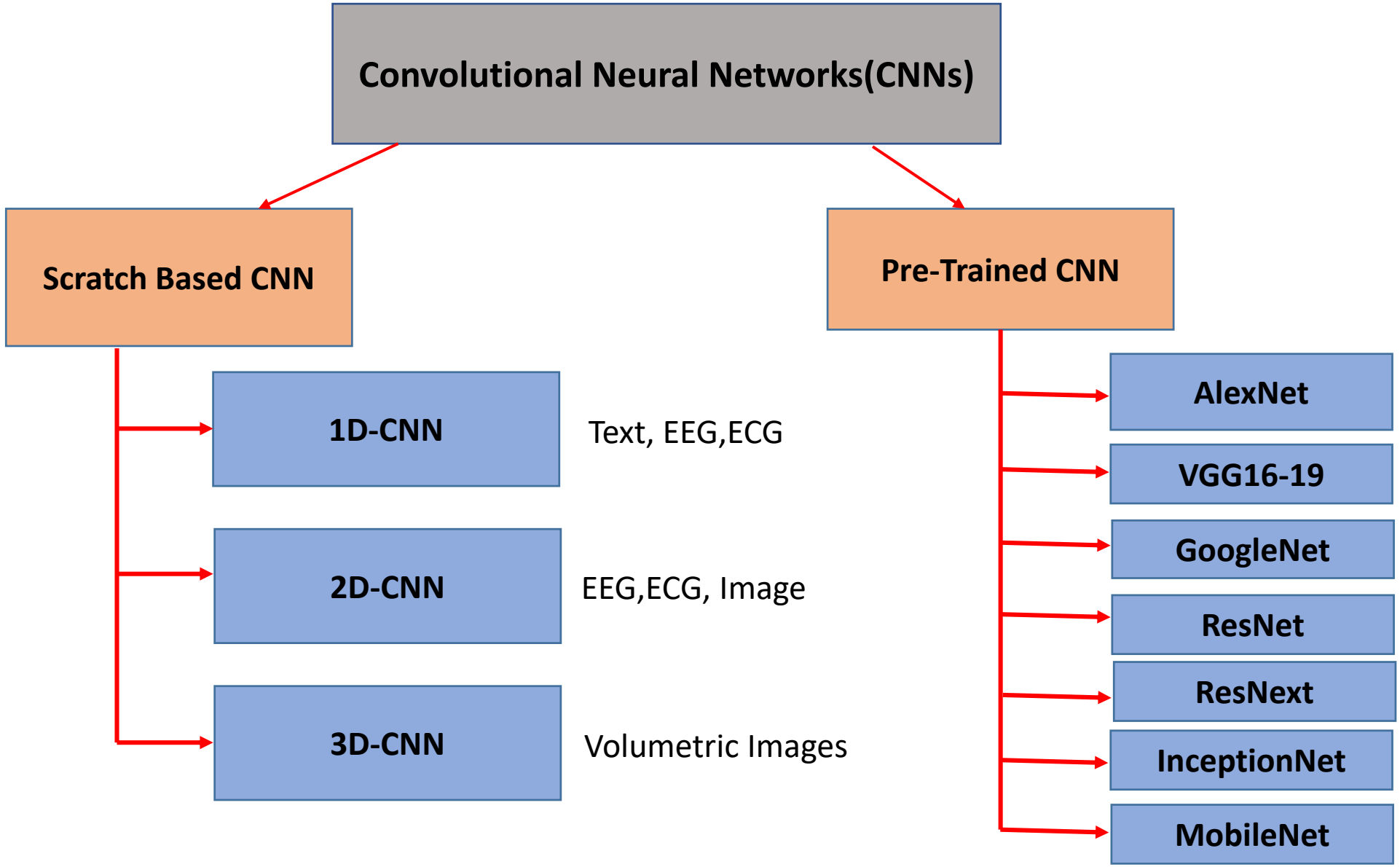
Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

$$C = \sum 1/2(\hat{y} - y)^2$$



Convolutional Neural Network (CNN)

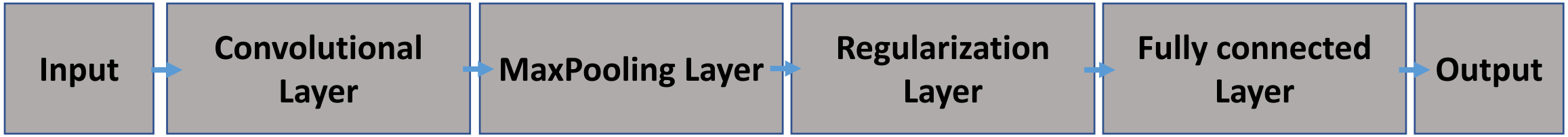
Convolutional Neural Networks



Convolutional Neural Networks

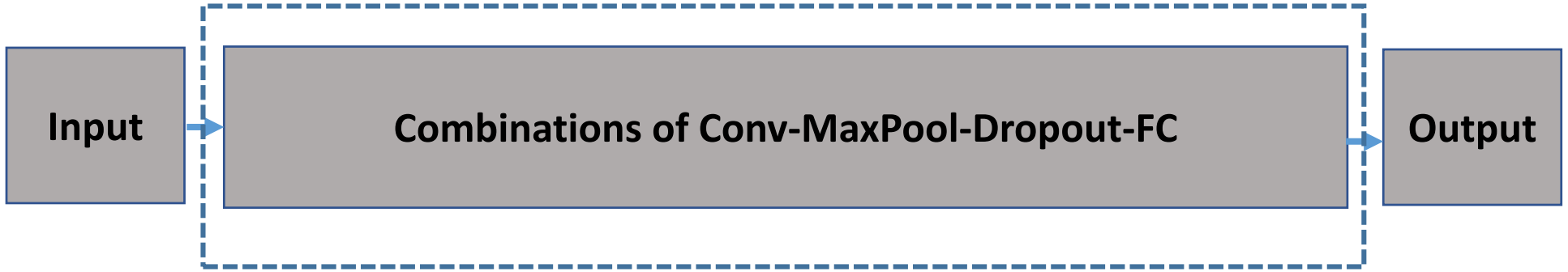
Scratch Based Network

Trained layers from scratch based on your own dataset



Pre-trained Networks

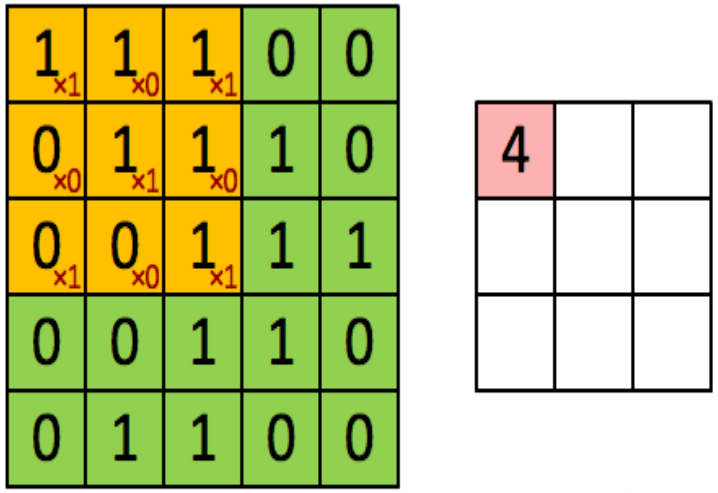
Trained layers on ImageNet dataset



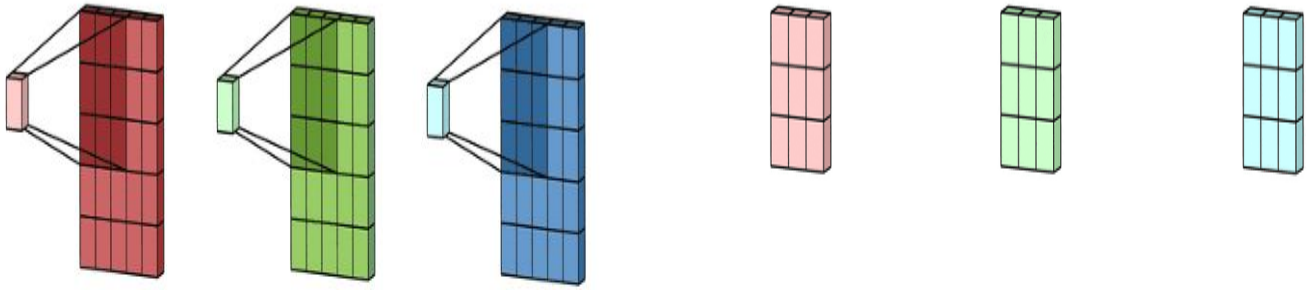
2D-CNN

Step 1- Convolution

- **One channel and 3 channel convolution**



Convolutional (1 channel)



Convolutional (3 channel)

https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9

Step 1- Convolution

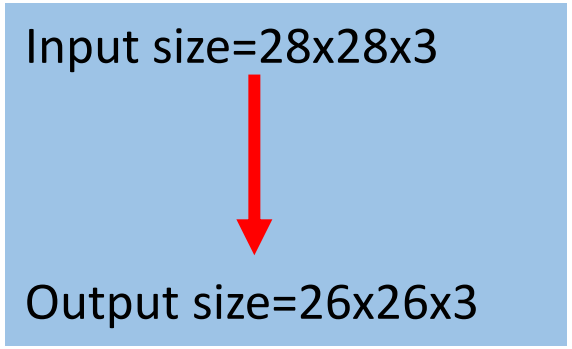
- **Convolution**

$$\text{Output\_Height}=\frac{Height\ of\ Image-Height\ of\ filter+2P}{S}+1$$

$$\text{Output\_width}=\frac{width\ of\ Image-width\ of\ filter+2P}{S}+1$$

Where P is **zero padding** and S is **stride**

W2=(W1−F+2P)/S+1=(28-3+2(0)/1+1)=26
H2=(H1−F+2P)/S+1 =(28-3+2(0)/1+1)=26

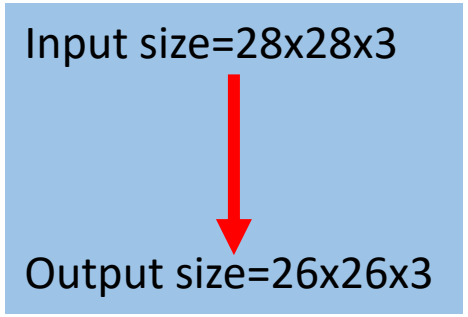


Step 1- Convolution

Conv Layer:
Accepts a volume of size =**W1×H1×D1**
Requires four hyperparameters:
Number of filters K, their spatial extent F or size of filter or kernel, the stride S, the amount of zero padding P.

Produces a volume of size W2×H2×D2 where:
W2=(W1−F+2P)/S+1
H2=(H1−F+2P)/S+1 (i.e. width and height are computed equally by symmetry)
D2=K (depth equal to length of number of filters)

Example:
Input_volume=Image or 2D matrix (height width x channels)=**W1×H1×D1** =28x28x3
2D Kernel or 2D filter= FxF(spatial filter or kernel)=**3x3xk(k is number of filters)**
In this example
k=3
Stride=1
P=0
W2=(W1−F+2P)/S+1=(28-3+2(0)/1+1)=26
H2=(H1−F+2P)/S+1 =(28-3+2(0)/1+1)=26
D2=3

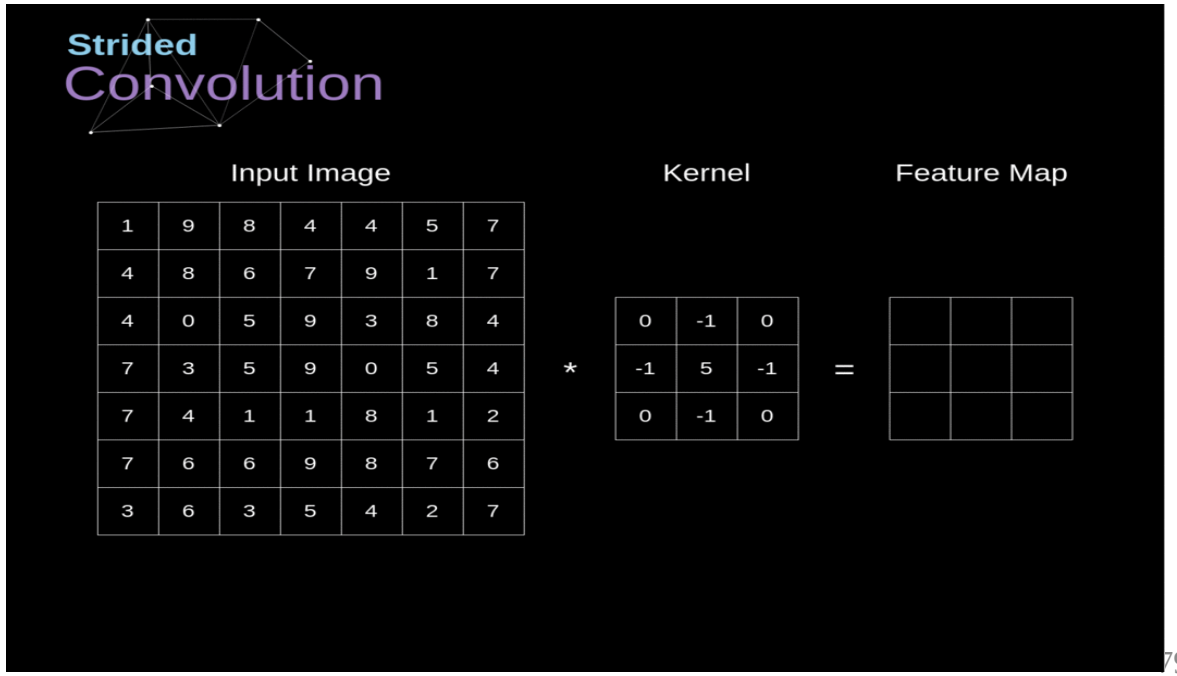


https://cs231n.github.io/convolutional-networks/

Step 1- Convolution

- **Convolution**
- we are dealing with two types of convolution — **Valid and Same.**
- Valid — means that we use the original image,
- Same — we use the border around it, so that the images at the input and output are the same size.
- In the second case, the padding width, should meet the following equation, where p is padding and f is the filter dimension (usually odd).

$$p = \frac{f-1}{2}$$

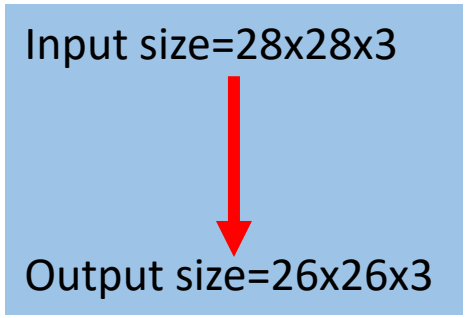


https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9

Step 1- Convolution

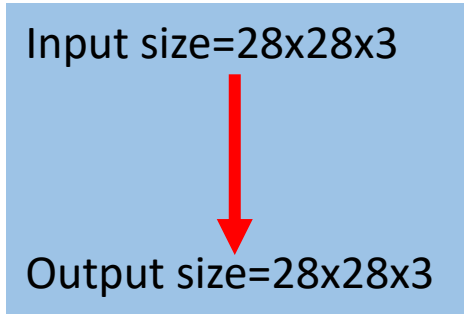
Valid Convolutional layer

Example:
Input_volume=Image or 2D matrix (height width x channels)=**W1×H1×D1** =28x28x3
2D Kernel or 2D filter= FxF(spatial filter or kernel)=**3x3xk(k is number of filters)**
In this example
k=3
Stride=1
P=0
W2=(W1−F+2P)/S+1=(28-3+2(0)/1+1)=26
H2=(H1−F+2P)/S+1 =(28-3+2(0)/1+1)=26
D2=3



Same Convolutional layer

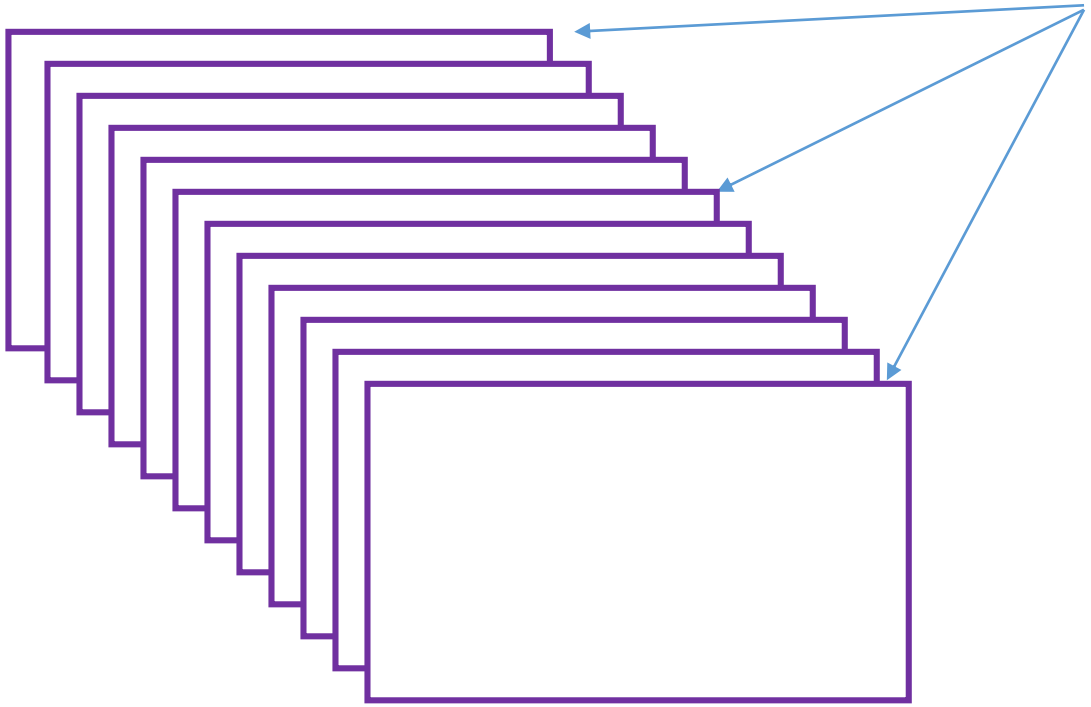
Example:
Input_volume=Image or 2D matrix (height width x channels)=**W1×H1×D1** =28x28x3
2D Kernel or 2D filter= FxF(spatial filter or kernel)=**3x3xk(k is number of filters)**
In this example
k=3
Stride=1
P=1 $p = \frac{f-1}{2}$
W2=(W1−F+2P)/S+1=(28-3+2(1)/1+1)=28
H2=(H1−F+2P)/S+1 =(28-3+2(1)/1+1)=28
D2=3



Step 1- Convolution

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

Input Image



Convolutional Layer

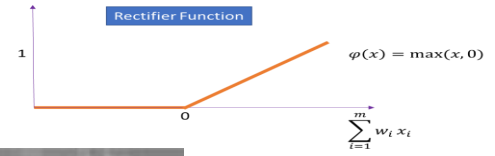
Step 1- Convolution

f1

f2

Step 1a- ReLU(Non-Linear)

Feature Map



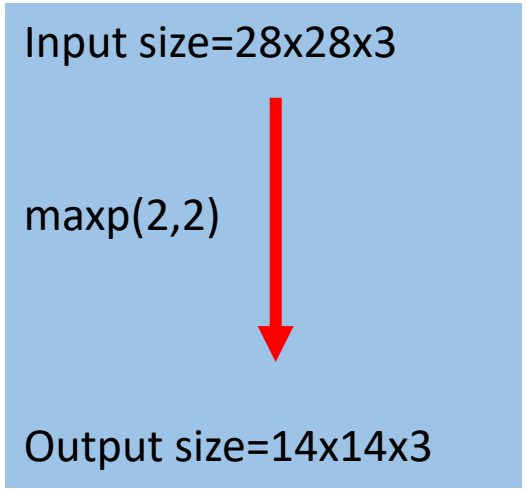
Step 2- Max Pooling

Accepts a volume of size **W1×H1×D1**
Requires two hyperparameters:

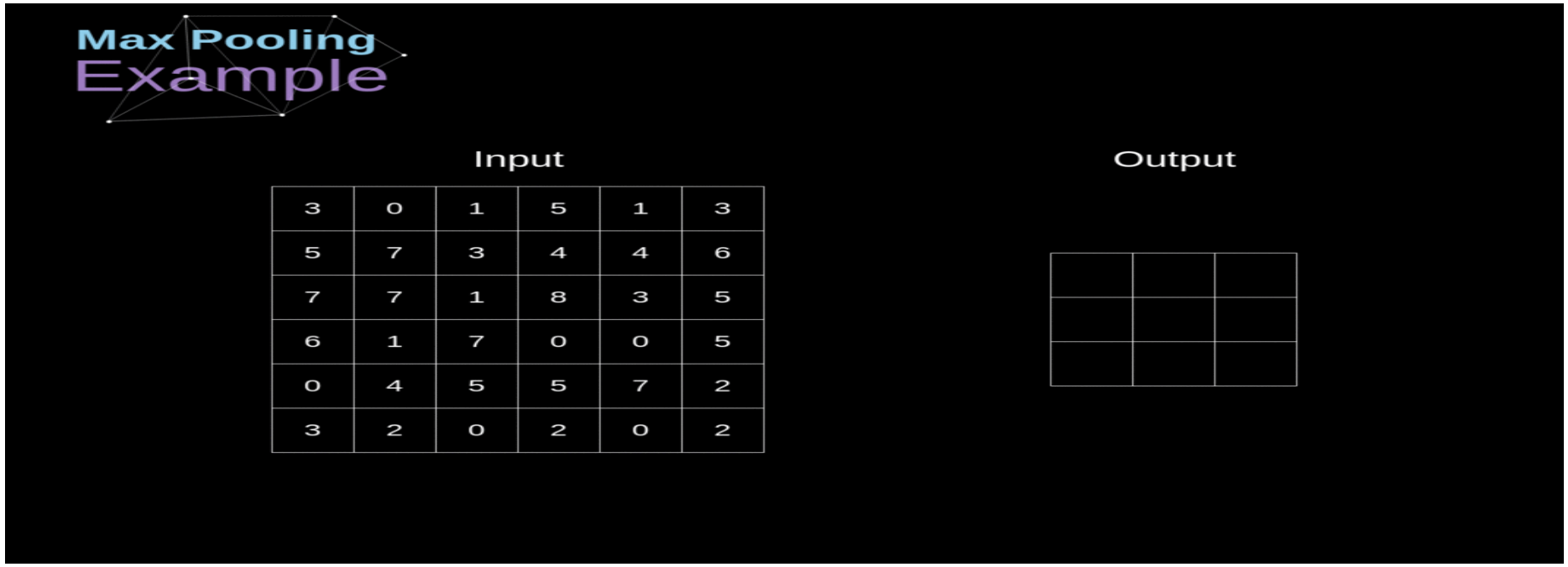
- **their spatial extent F,**
- **the stride S,**

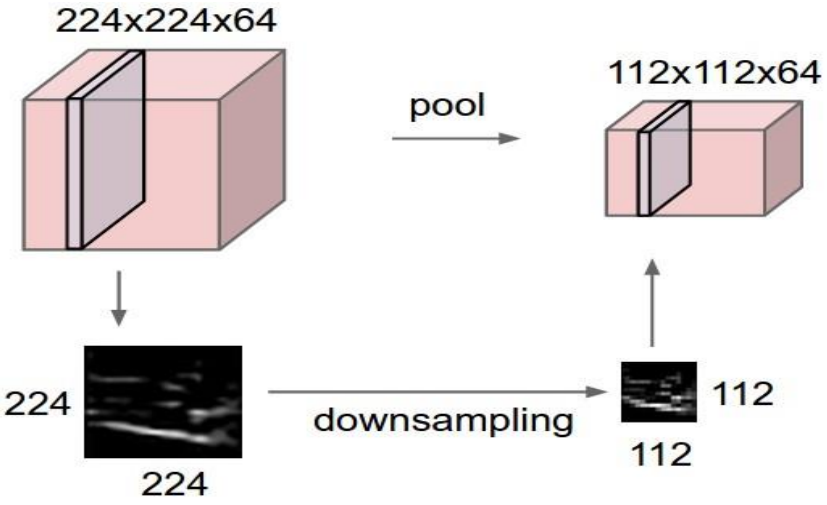
Produces a volume of size W2×H2×D2 where:
W2=(W1−F)/S+1
H2=(H1−F)/S+1
D2=D1
Introduces zero parameters since it computes a fixed function of the input.
For Pooling layers, it is not common to pad the input using zero-padding.

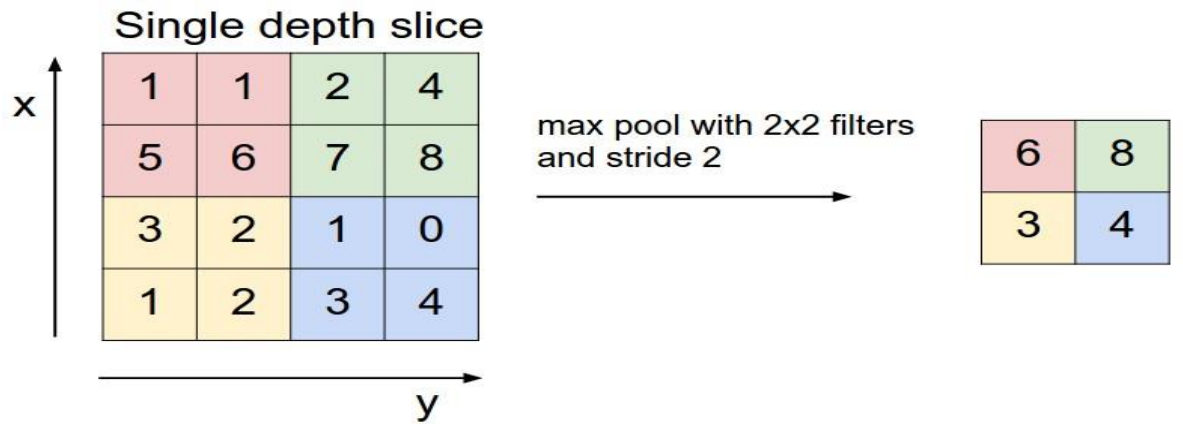
Example:
Input_volume=Image or 2D matrix (height width x channels)=**W1×H1×D1** =28x28x3
(pool_size)= FxF=**2x2**
In this example
Stride=2
P=0
W2=(W1−F+2P)/S+1=(28-3+2(0)/2+1)=14
H2=(H1−F+2P)/S+1 =(28-3+2(0)/2+1)=14
D2=3



Step 2- Max Pooling

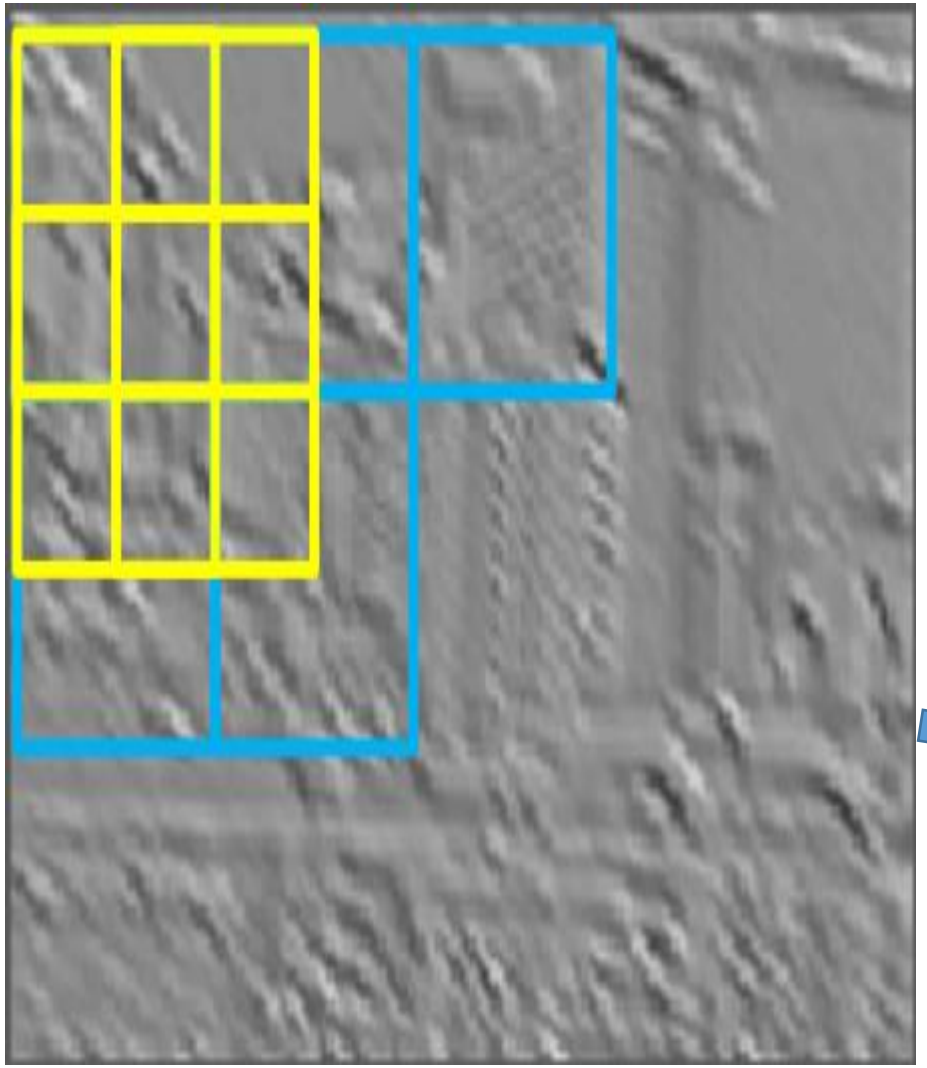




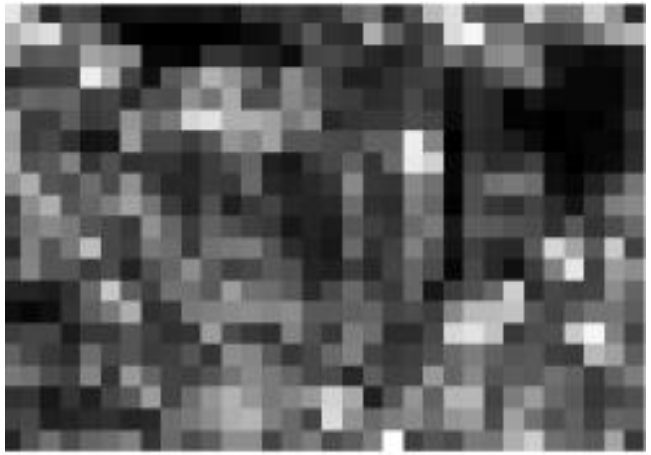


https://cs231n.github.io/convolutional-networks/

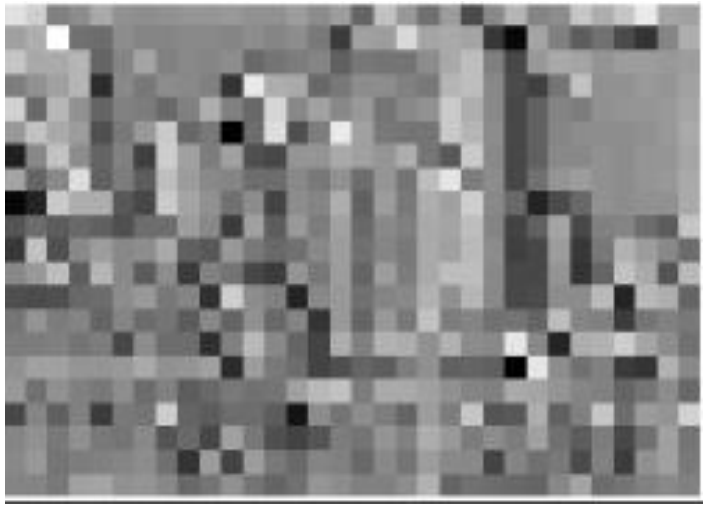
Step 2- Pooling Layers



Max



Sum



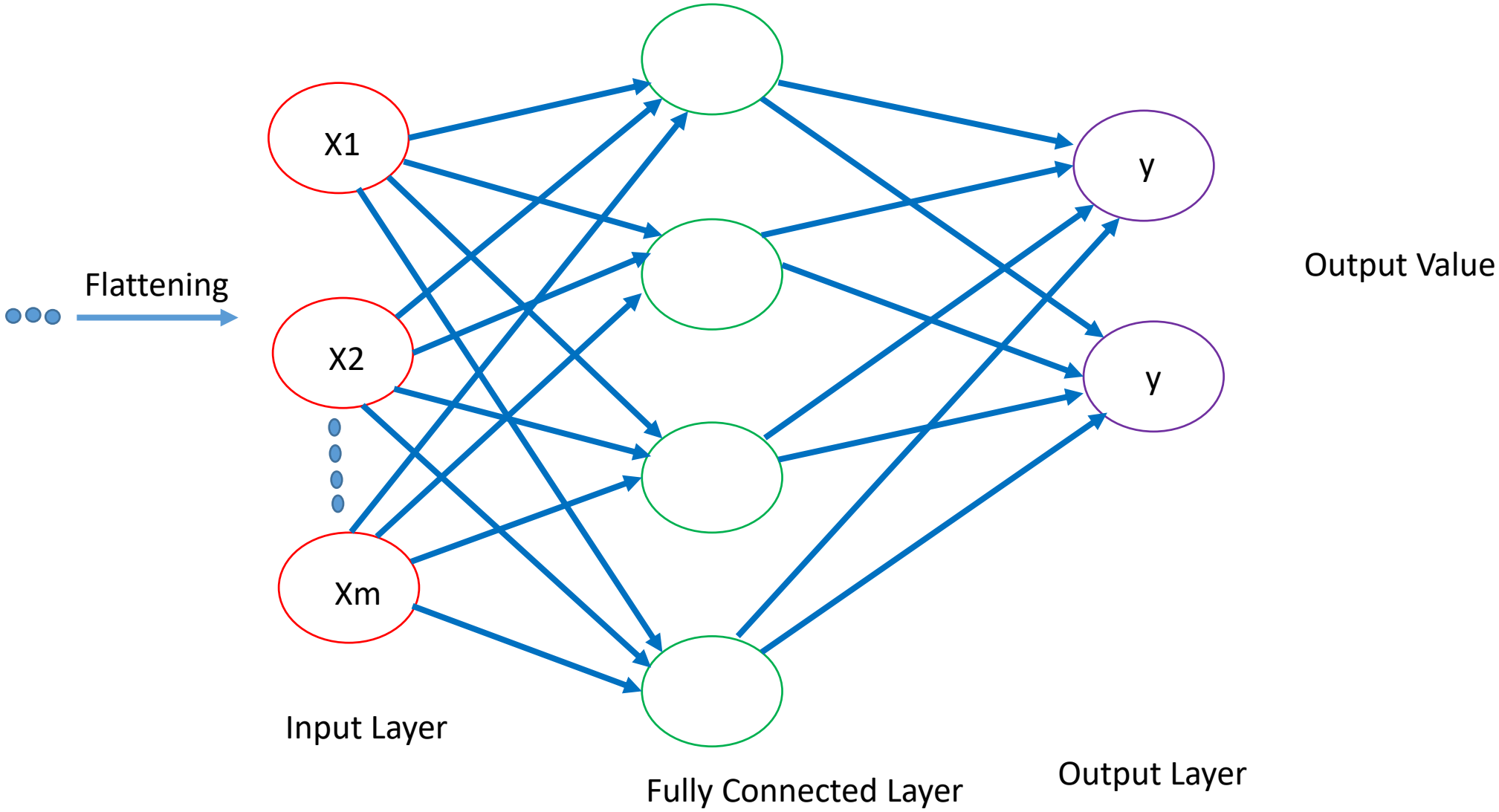
Step 3- Flattening

1	**1**	**0**
4	**2**	**1**
0	**2**	**1**

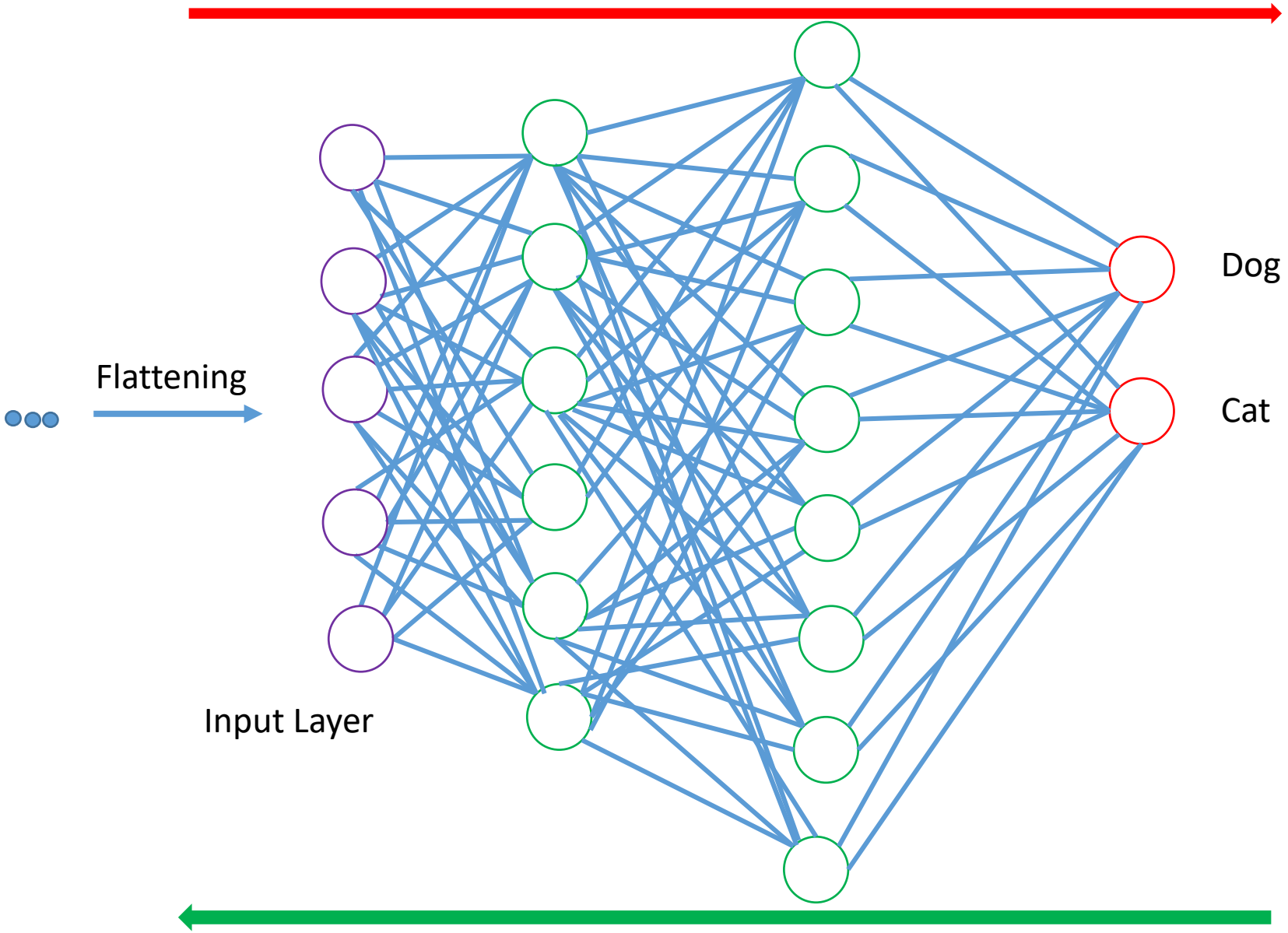
Pooled Feature Map

1
1
0
4
2
1
0
2
1

Step 4- Full Connection

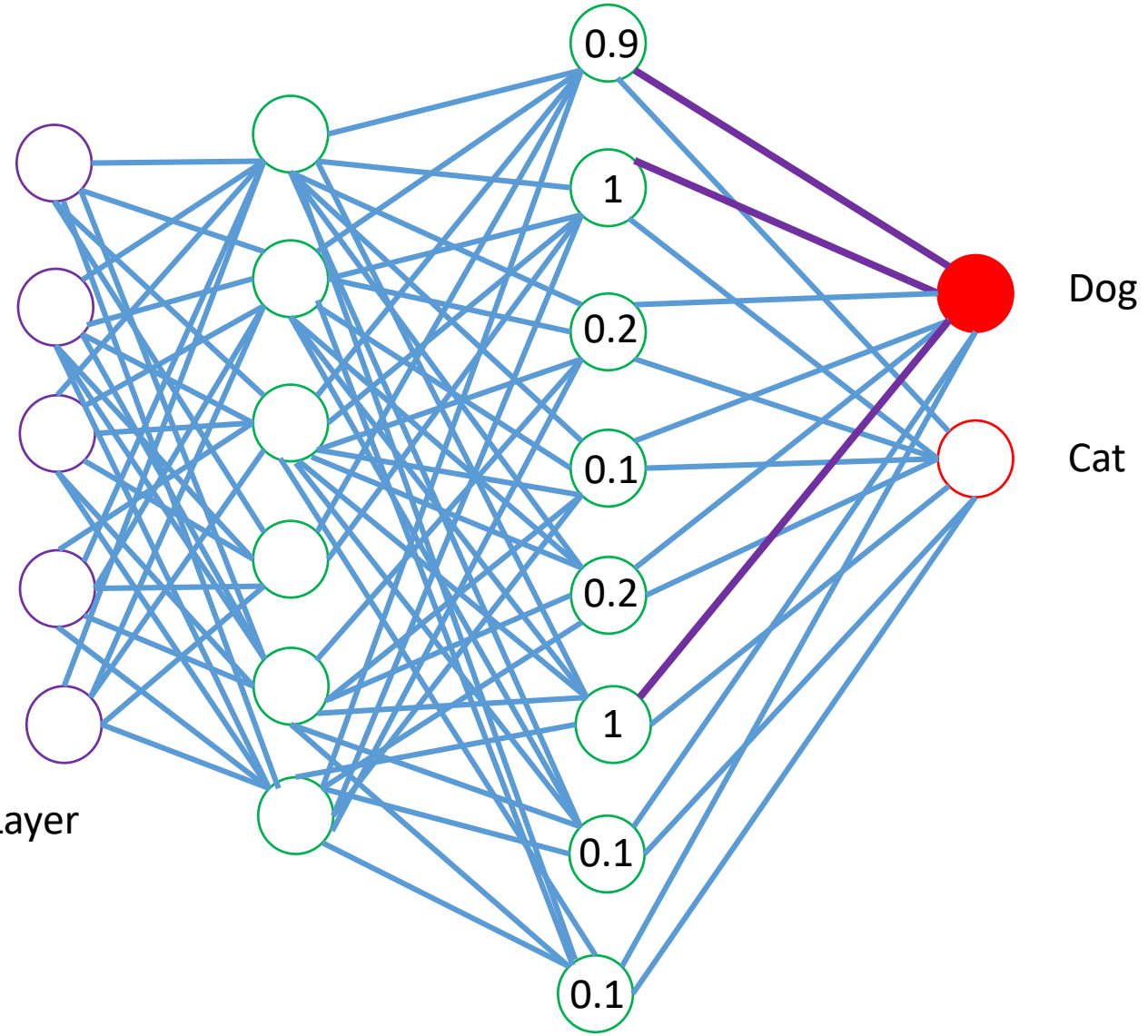


Step 4- Full Connection

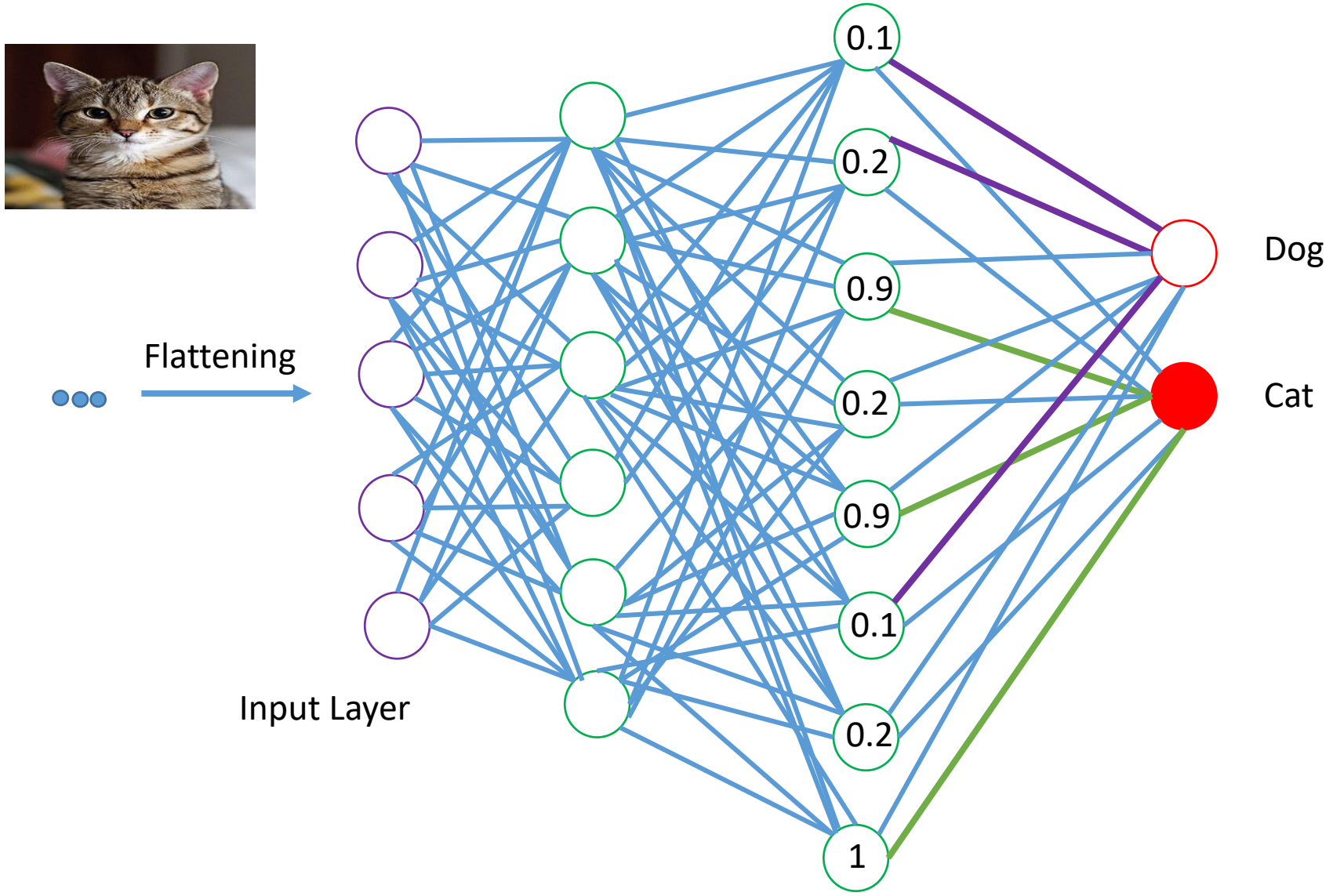


Step 4- Full Connection

Flattening



Step 4- Full Connection

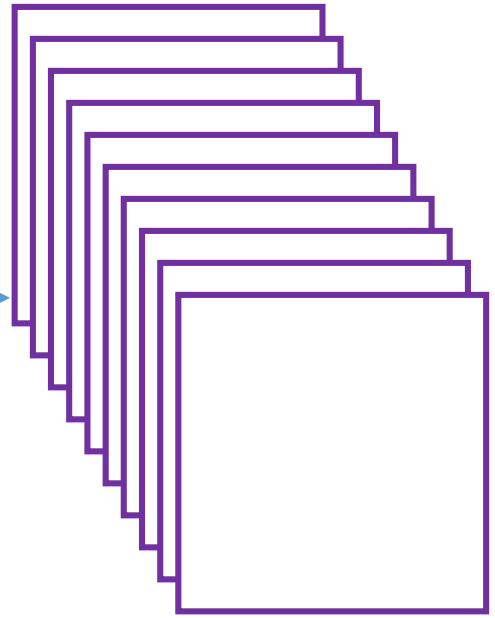


2D-Convolutional Neural Network

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

Input Image

convolution

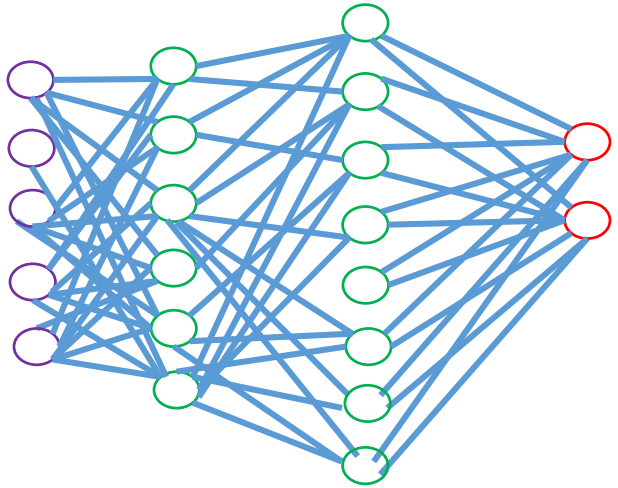


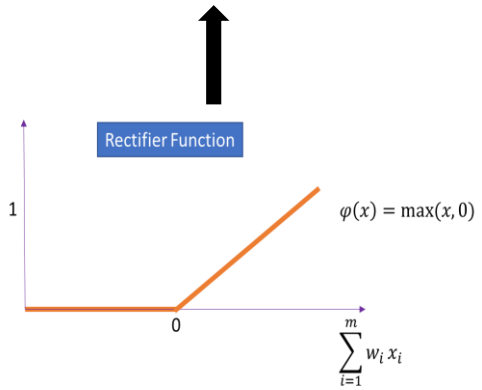
Convolutional Layer

Pooling

Pooling Layer

Flattening





Batch Normalization Layer

To increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.
Also, batch normalization allows each layer of a network to learn by itself a little bit more independently of other layers.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

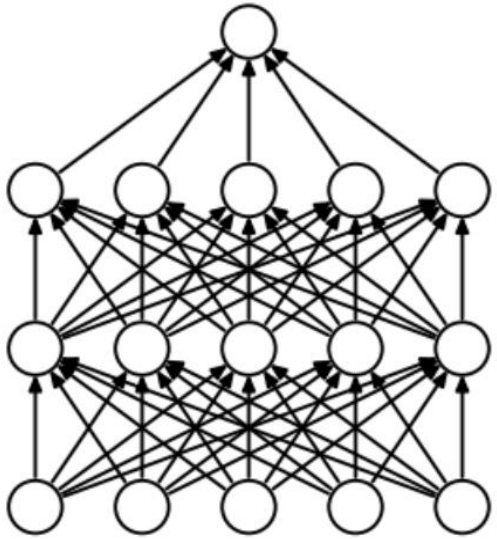
Dropout Layer

In machine learning, regularization is way to prevent over-fitting. Regularization reduces over-fitting by adding a penalty to the loss function. By adding this penalty, the model is trained such that it does not learn interdependent set of features weights. **Those of you who know Logistic Regression might be familiar with L1 (Laplacian) and L2 (Gaussian) penalties.**

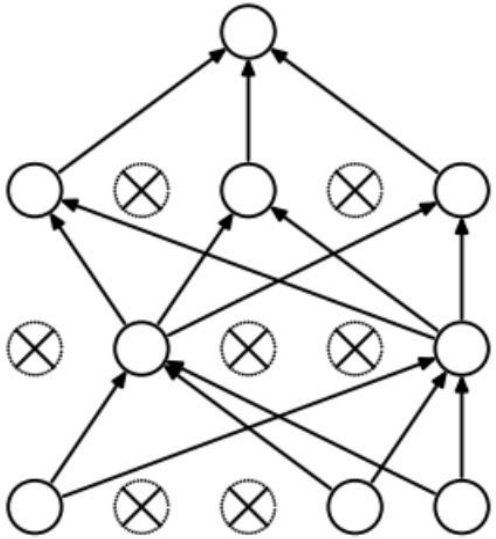
Dropout is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons.

Training Phase:

Training Phase: For each hidden layer, for each training sample, for each iteration, ignore (zero out) a random fraction, *p*, of nodes (and corresponding activations).



(a) Standard Neural Net



(b) After applying dropout.

Some Observations:

- Dropout forces a neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.
- Dropout roughly doubles the number of iterations required to converge. However, training time for each epoch is less.
- With H hidden units, each of which can be dropped, we have 2^H possible models. In testing phase, the entire network is considered and each activation is reduced by a factor p.

2D-Convolutional Neural Network

Number of Parameters of a Conv Layer

In a CNN, each layer has two kinds of parameters : weights and biases. The total number of parameters is just the sum of all weights and biases.

Parameters or trainable parameters are optimized during the training of deep learning layers such as convolutional layers and fully connected layers. There are no parameters associated with a MaxPool layer. The pool size, stride, and padding are hyperparameters.

No of parameters or trainable parameters are computed for Convolutional layer

C1 layer(parameters)=**(F1xF2xK+1)*M**

Where F1 and F2 are height and width of filter or kernel, K total number of filters or channels used in previous layers and M are the number of filters used in that layer

No of parameters or trainable parameters are computed for Fully connected layer 1

C1 layer(parameters)=**(K+1)*M**

K No of neurons for previous layers say convolutional or Maxpooling and M are the neurons or units used in that FC1 layer

2D-Convolutional Layer in Pytorch

```
# The CNN model has some combination of layers such as Convoltional layer, R
eLU Layer,
# BatchNormalization Layer,Maxpooling Layer,Fully connected layer
# we can define layer as
import torch
import torch.nn as nn
c1=nn.Conv2d(in_channels=3,out_channels=16,kernel_size=3,stride=1,padding=0)
inp=torch.rand(1,3,224,224) #batchxchannelxheightxwidth
out=c1(inp)
print(out.shape) # 224-3+1=222,224-3+1=222
```

2D-MaxPool Layer in Pytorch

```
# we can define layer as
import torch.nn as nn
mp=nn.MaxPool2d(2)
inp=torch.rand(1,3,224,224) #batchxchannelxheightxwidth
out=mp(inp)
print(out.shape) # 224/2,224/2
```

Faltten Layer in Pytorch

```
# Flatten Layer
import torch.nn as nn
flatten=nn.Flatten()
inp=torch.rand(1,3,224,224) #batchxchannelxheightxwidth
out=flatten(inp)
print(out.shape) # 3x224x224=150528
```

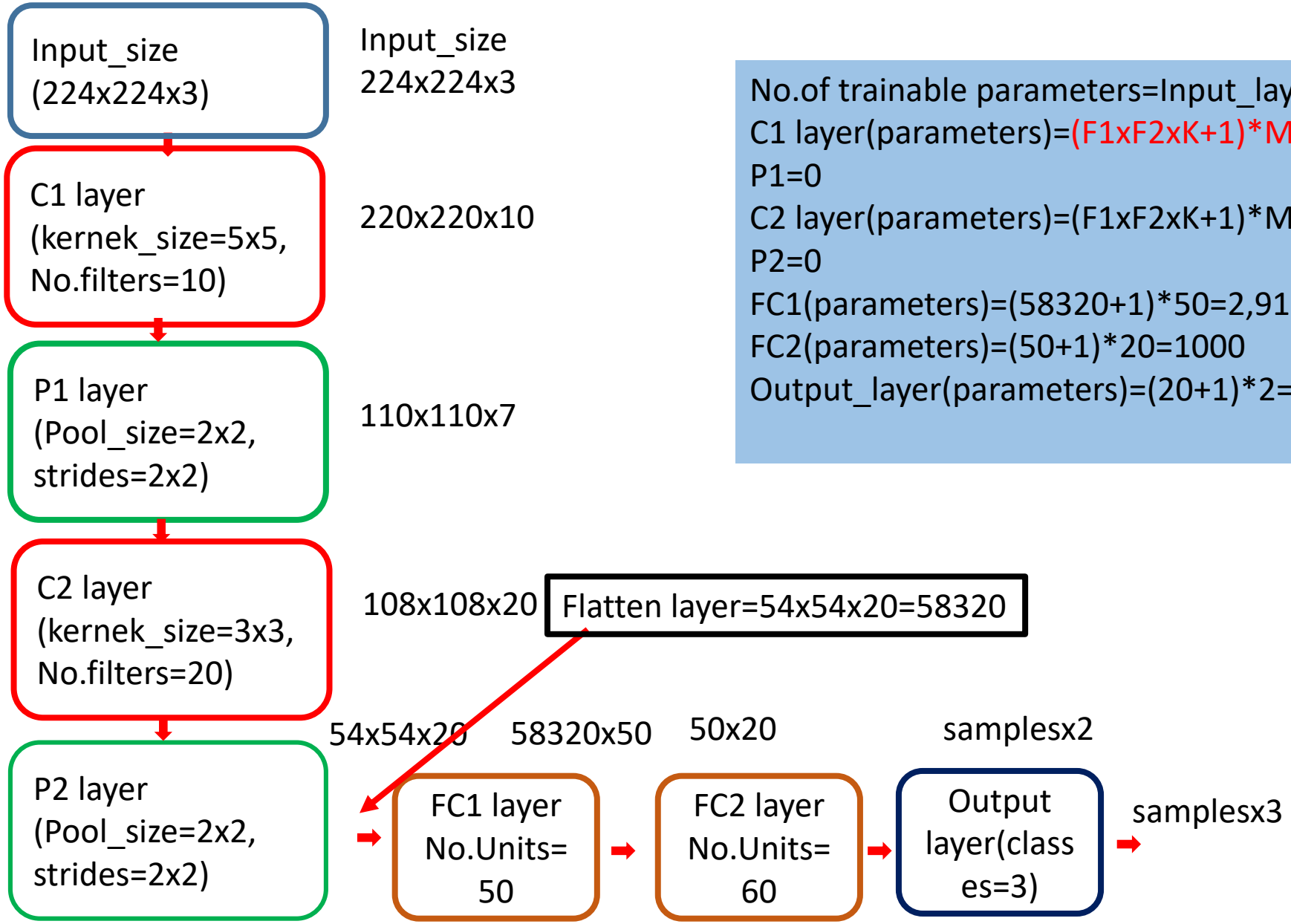
Fully connected Layer in Pytorch

```
# Fully connected layer
import torch.nn as nn
inp=torch.rand(1,3,224,224) #batchxchannelxheightxwidth
fm=nn.Flatten()
outf=fm(inp)
fc=nn.Linear(in_features=3*224*224,out_features=512) # recieve 3x224x224 fla
tten out andproduced 512
fully_connected=fc(outf)
print(fully_connected.shape) # 1x512
```

ReLU activation Layer in Pytorch

```
# reLU layer
print("before relu",fully_connected)
reluo=nn.ReLU()(fully_connected)
print("after relu",reluo)
```

2D-Convolutional Neural Network

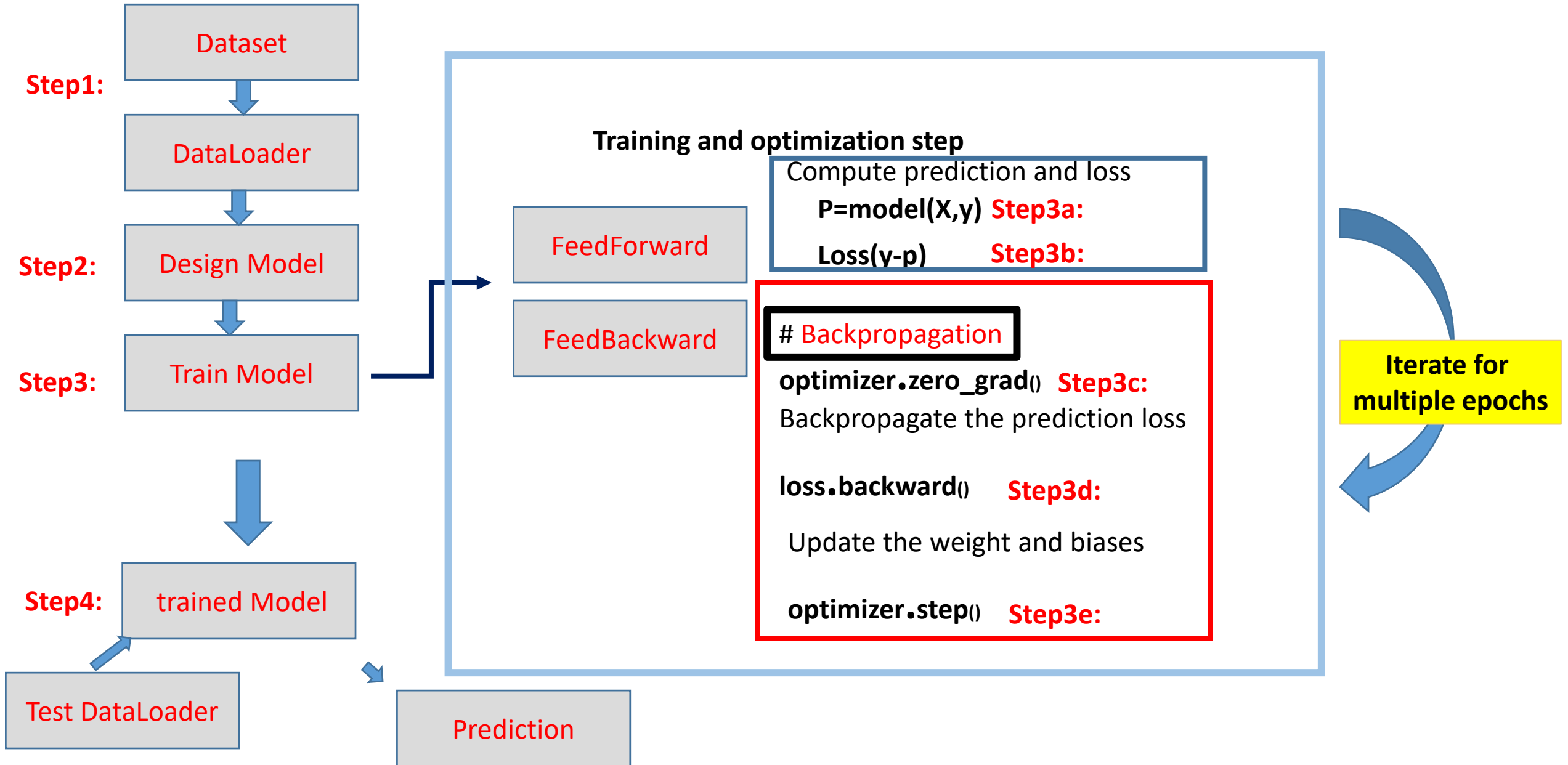


No.of trainable parameters=Input_layer=0
C1 layer(parameters)=(F1xF2xK+1)*M=(3x5x5+1)*10=750
P1=0
C2 layer(parameters)=(F1xF2xK+1)*M=(5x5x10+1)*20=5000
P2=0
FC1(parameters)=(58320+1)*50=2,916,000
FC2(parameters)=(50+1)*20=1000
Output_layer(parameters)=(20+1)*2=40

2D-Convolutional Neural Network

```
# scracth based model(define your owen layers)
import torch
import torch.nn as nn
######################################## scratch based models ##############################
class My_model(nn.Module):
    def __init__(self):
        super(My_model,self).__init__()
        # layer block 1
        self.conv1=nn.Conv2d(in_channels=3,out_channels=10,kernel_size=5) #224-5+1=220
        self.relu1=nn.ReLU()
        # layer block two
        self.batch1=nn.BatchNorm2d(10)
        self.maxpool1=nn.MaxPool2d(kernel_size=2) # 220/2=110
        self.conv2=nn.Conv2d(in_channels=10,out_channels=20,kernel_size=3) #111-3=108
        self.relu2=nn.ReLU()
        self.batch2=nn.BatchNorm2d(20)
        self.maxpool2=nn.MaxPool2d(kernel_size=2) # 222/2=54
        self.fc1=nn.Linear(54*54*20,20)
        self.fc2=nn.Linear(20,2)
    def forward(self,x):
        x=self.batch1(self.relu1(self.conv1(x)))
        x=self.maxpool1(x)
        print(x.shape)
        x=self.batch2(self.relu2(self.conv2(x)))
        x=self.maxpool2(x)
        print(x.shape)
        x=x.view(-1,54*54*20)
        print(x.shape)
        x=self.fc1(x)
        print(x.shape)
        x=self.fc2(x)
        #print(x.shape)
        return x
```

Training Deep Learning model

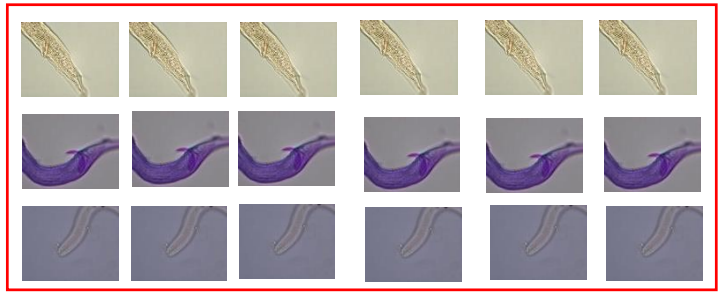


Training Deep Learning model

Step1:

Dataset

DataLoader



18x3x224x224

18 images in a batch

Step2:

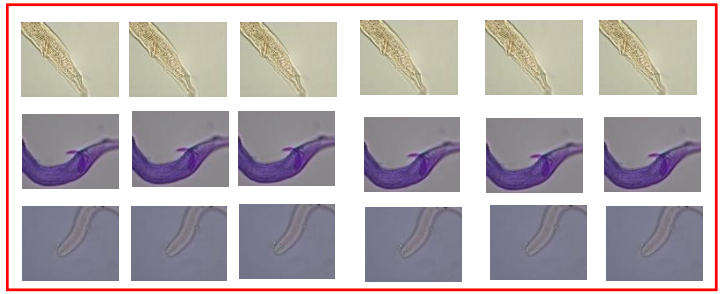
Design Model

Training Deep Learning model

Step1:

Dataset

DataLoader



18 images in a batch

18x3x224x224

Design Model

Step2:

```
# dataset path
pathtrain="/content/drive/MyDrive/ISblue_workshop/dataset/train"
pathval="/content/drive/MyDrive/ISblue_workshop/dataset/val"

from torchvision import datasets
from torch.utils.data import DataLoader
```

Load dataset

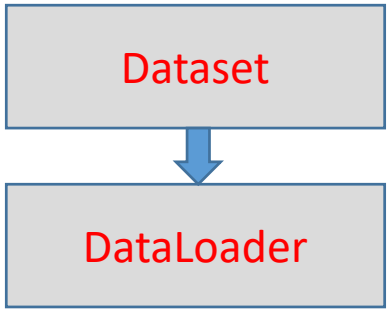
```
data_train=datasets.ImageFolder(pathtrain)
```

Convert data into batches

```
train_dataloader=DataLoader(data_train,batch_size=4,shuffle=True)
```

Training Deep Learning model

Step1:




```
data_train=datasets.ImageFolder(pathtrain)
```

```
train_dataloader=DataLoader(data_train,batch_size=4,shuffle=True)
```

same

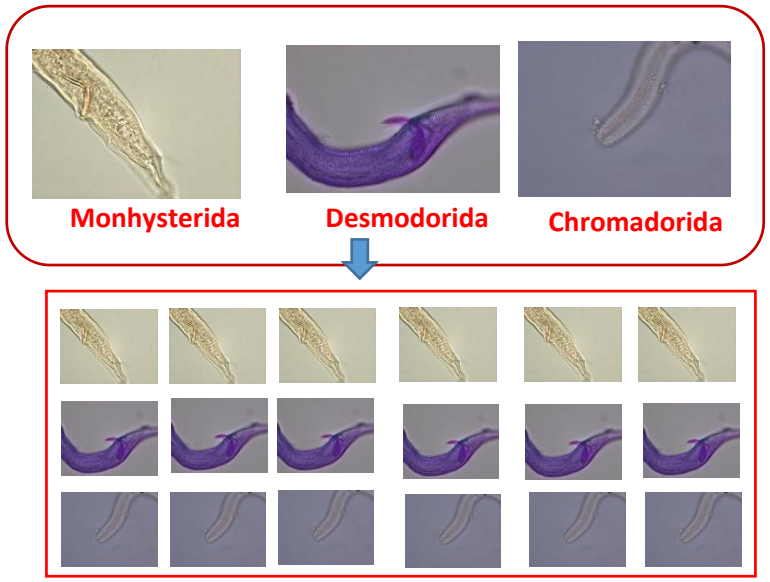
```
class Smallanimaldata(Dataset):
    def __init__(self,root,class_1,class_2,class_3,is_train):
        self.root=root
        self.class_1=class_1
        self.class_2=class_2
        self.class_3=class_3
        self.is_train = is_train
        self.totpath=self.clas1+self.clas2+self.clas3
    def __getitem__(self,idx):

            pathd,label=self.totpath[idx]
            #print(pathd)
            img=cv2.imread(pathd)
            img=self.transform(img)

            return {"im1":img,
                    "labl1":label}

     def __len__(self):
            return len(self.totpath)
```

Training Deep Learning model



18 images (18x3x224x224)

Step3:

Forward

P=model(X,y)

Loss(y-p)

Backward

optimizer.zero_grad()

loss.backward()

optimizer.step()

Feedforward

DL Model

P=18x3

Y= 18

P

C1	c2	c3

1

18

On-hot encoding

Y

C1	c2	c3

How to compute loss function

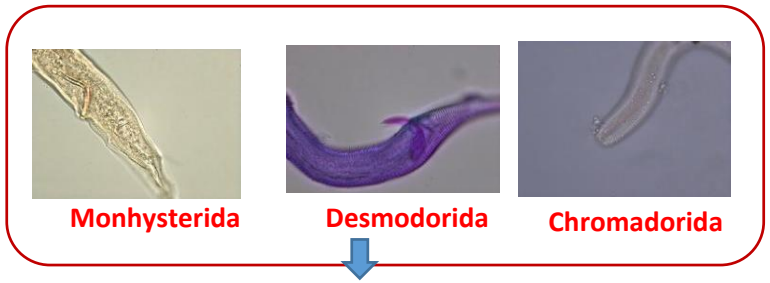
Y=18x3,P=18x3

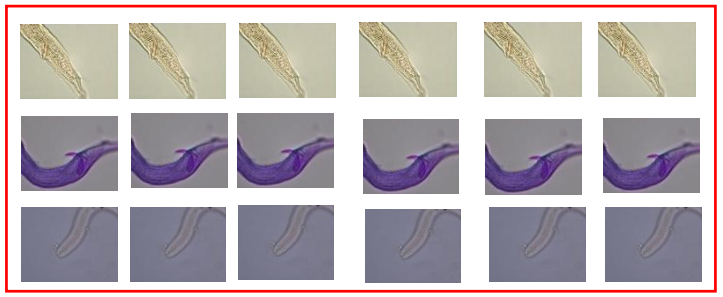
Feedbackward and update the Gradient

Loss or error=(Y-P)

Perform a backwards pass from right to left and propagate the error to every layers using backpropagation. Calculate each weight's contribution to the error, and adjust the weights accordingly using gradient descent. Propagate the error gradients back starting from the last layer.

Testing the Trained Model





12 images (12x3x224x224)

Forward

P=model(X,y)

Loss(y-p)

Feedforward

Trained DL model

P=12x3

Probability

	C1	c2	c3
1	0.9	0.3	0.6
	0.4	0.5	0.8
	0.8	0.6	0.4
12	0.3	0.7	0.4

maximum

prediction

0
2
0
1

1 Test sample ground truth labels 12

0	2	1	0														1

Training Deep Learning model

Define Loss function

```
loss_func=nn.CrossEntropyLoss()
```

Define optimizer

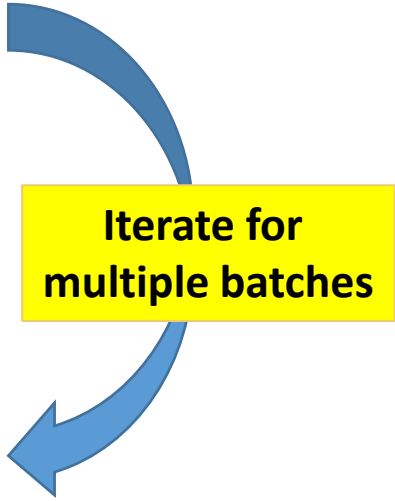
```
import torch.optim as optim
optimizer=optim.Adam(model.parameters(),lr=0.0001)
```

Define training loop for optimzation

```
for i, data in tqdm(enumerate(train_loader)
        counter+=1
        # extract dataset
        imge,label=data["im1"],data["labl1"]
        imge=imge.to(device)
        label=label.to(device)
        # zero_out the gradient
        optimizer.zero_grad()
        output=model(imge)
        loss=loss_func(output,label)
        # backpropagation
        loss.backward()
        optimizer.step()
```

P=model(X,y)

Loss(y-p)



Training Deep Learning model

Define Testing or validation loop

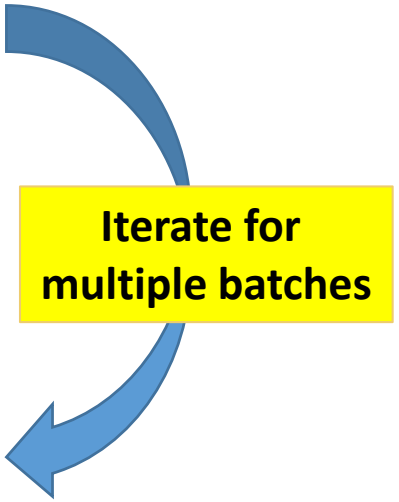
```
for i,data in tqdm(enumerate(valid_loader)):
            imge,label=data["im1"],data["labl1"]
            imge=imge.to(device)
            label=label.to(device)
            output=model(imge)
            loss=loss_func(output,label)
           _,pred=torch.max(output.data,1)
```

P=model(X,y)

Loss(y-p)

Feedforward

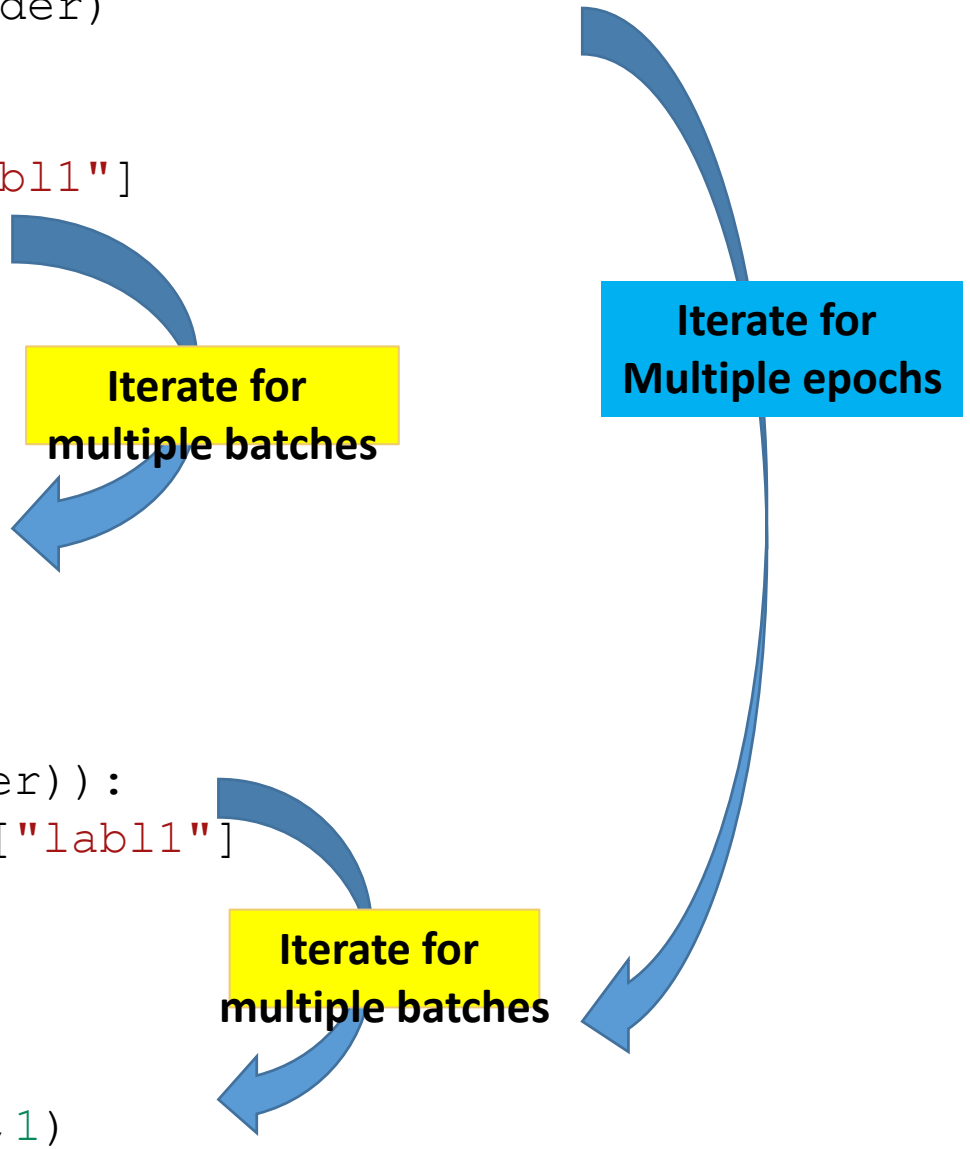
Get prediction



Training Deep Learning model

```
for epoch in range(epochs):
        for i, data in tqdm(enumerate(train_loader)
                counter+=1
                # extract dataset
                imge,label=data["im1"],data["labl1"]
                imge=imge.to(device)
                label=label.to(device)
                # zero_out the gradient
                optimizer.zero_grad()
                output=model(imge)
                loss=loss_func(output,label)
                # backpropagation
                loss.backward()
                optimizer.step()

        for i,data in tqdm(enumerate(valid_loader)):
                imge,label=data["im1"],data["labl1"]
                imge=imge.to(device)
                label=label.to(device)
                output=model(imge)
                loss=loss_func(output,label)
                _,pred=torch.max(output.data,1)
```



Training Deep Learning model

Loss Functions

`nn.L1Loss`	Creates a criterion that measures the mean absolute error (MAE) between each element in the input x and target y.
`nn.MSELoss`	Creates a criterion that measures the mean squared error (squared L2 norm) between each element in the input x and target y.
`nn.CrossEntropyLoss`	This criterion combines `LogSoftmax` and `NLLLoss` in one single class.
`nn.CTCLoss`	The Connectionist Temporal Classification loss.
`nn.NLLLoss`	The negative log likelihood loss.

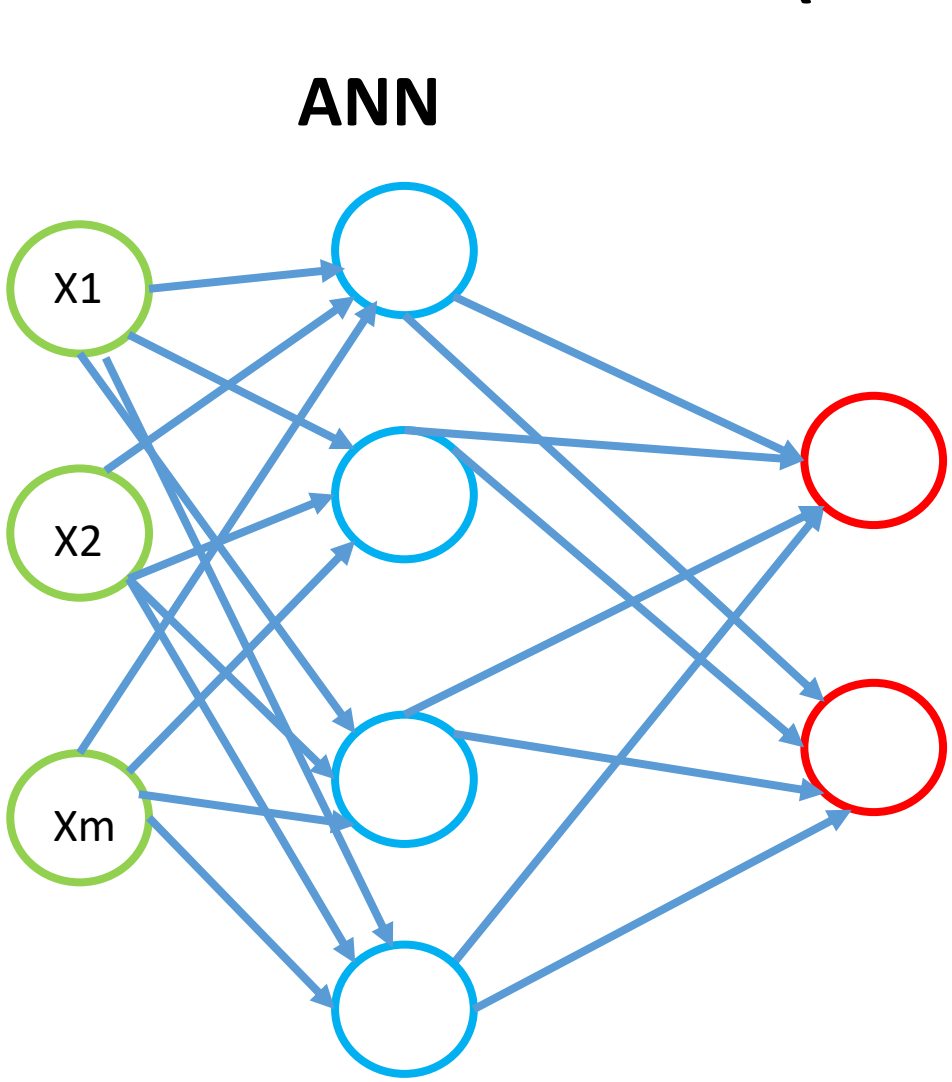
Training Deep Learning model

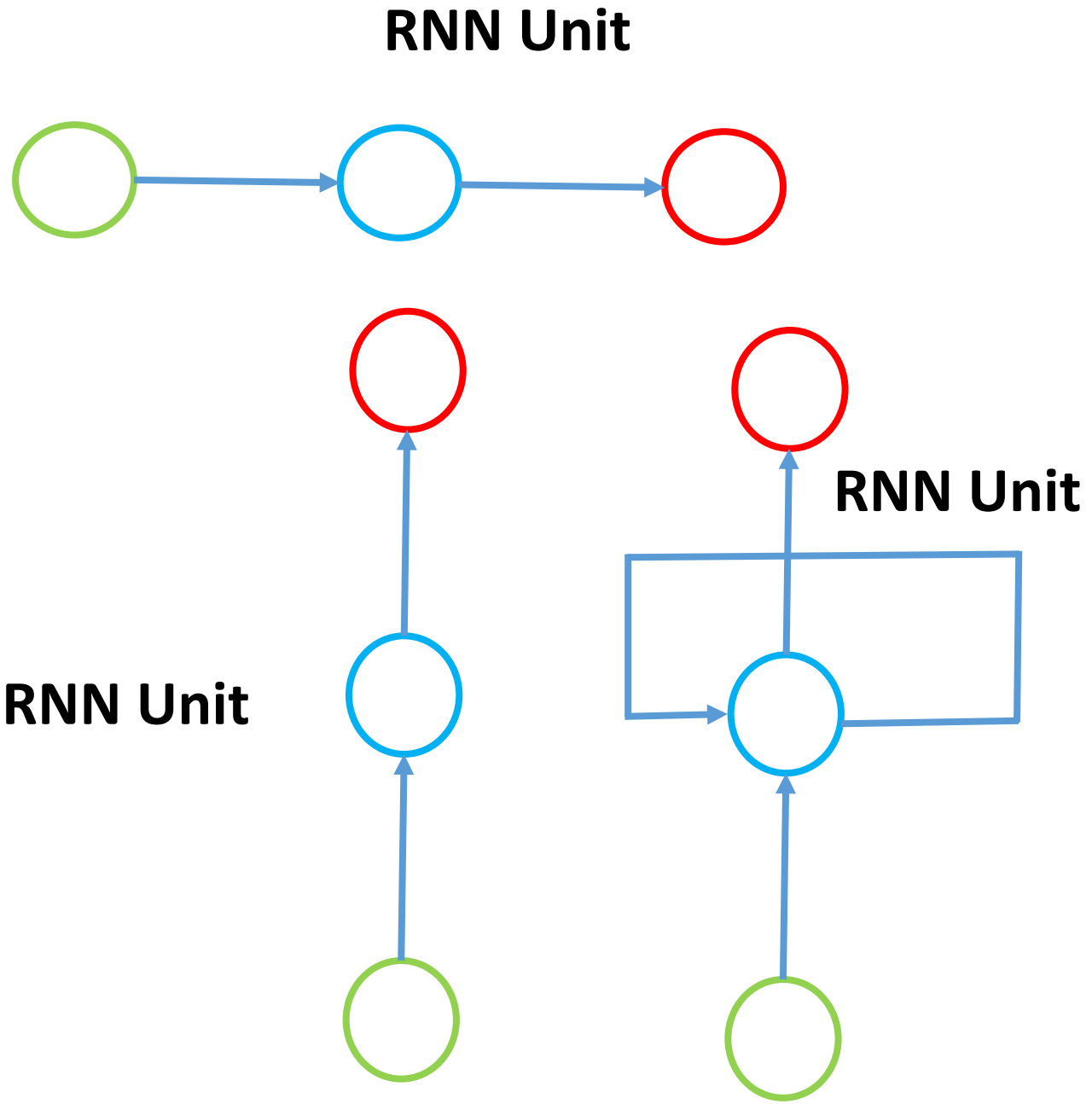
Algorithms

`Adadelta`	Implements Adadelta algorithm.
`Adagrad`	Implements Adagrad algorithm.
`Adam`	Implements Adam algorithm.
`AdamW`	Implements AdamW algorithm.
`SparseAdam`	Implements lazy version of Adam algorithm suitable for sparse tensors.

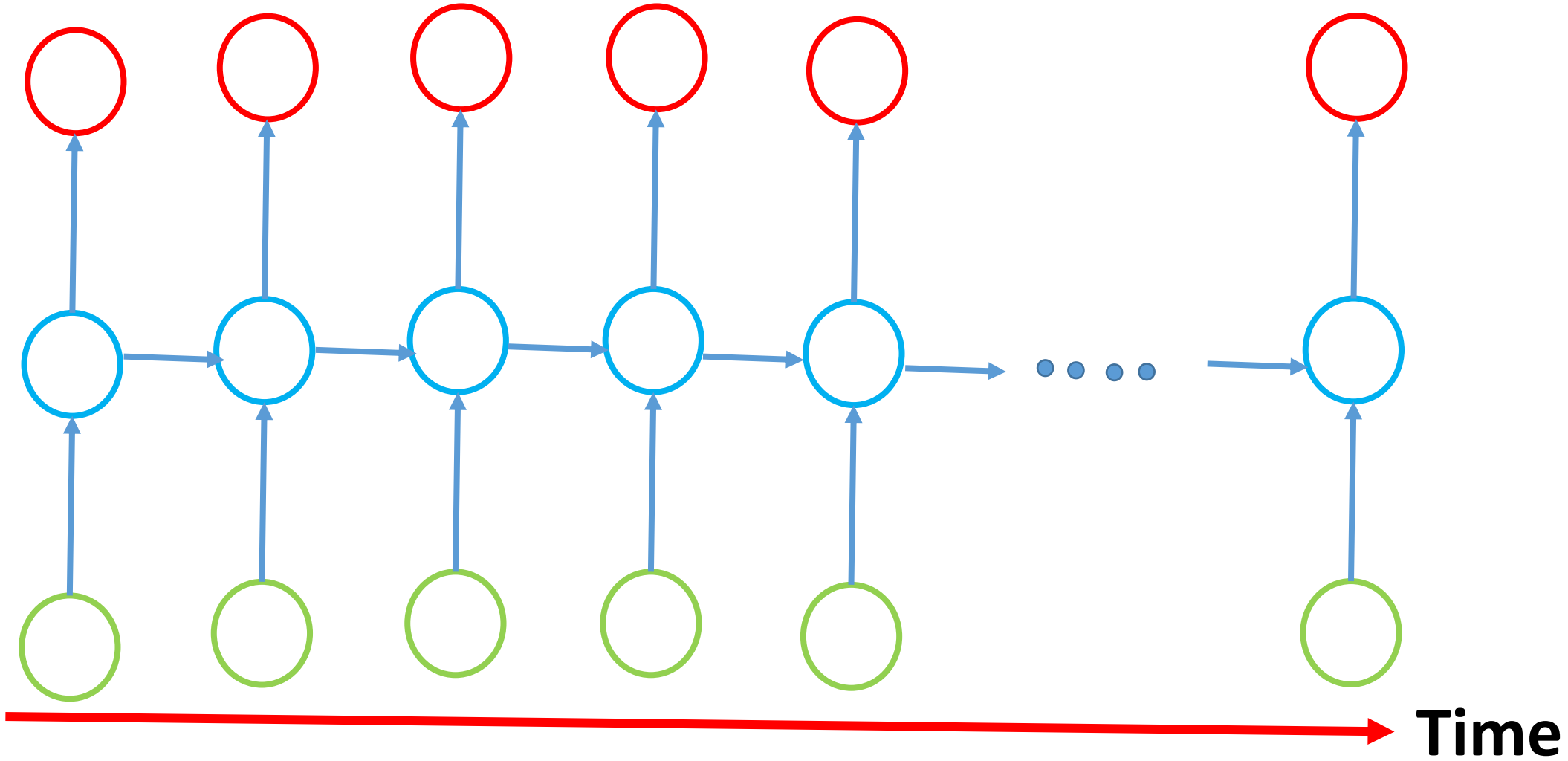
Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN)

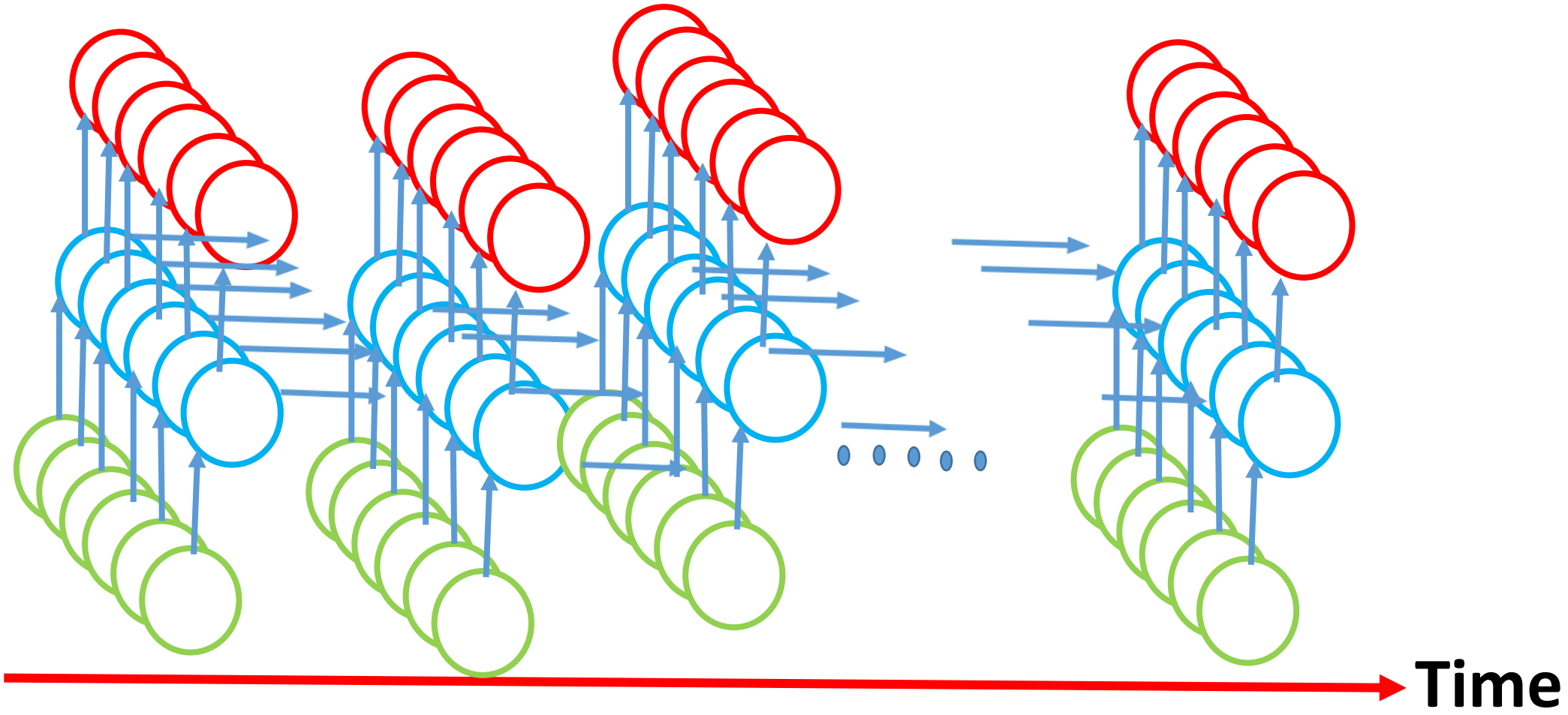




RNN

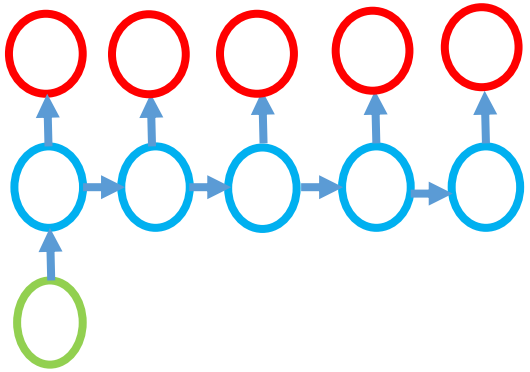


RNN



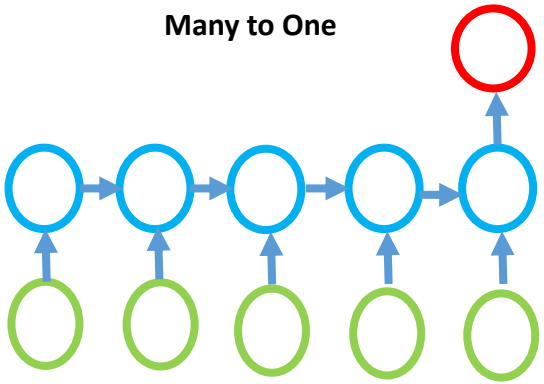
RNN Models

One to Many



Time

Many to One

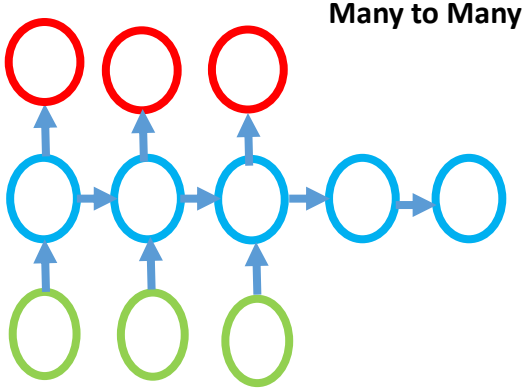


"Thanks for a great party at the weekend, we really enjoyed it!"

sentiment: positive
score: 86%

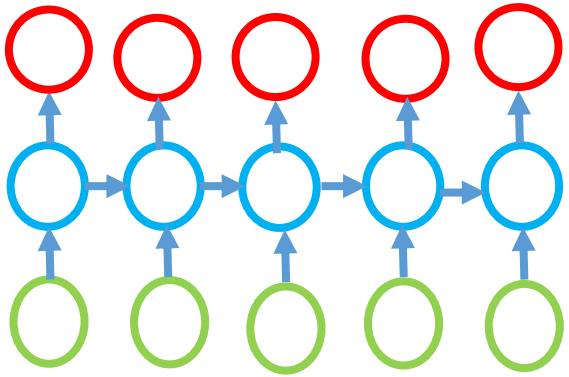
Time

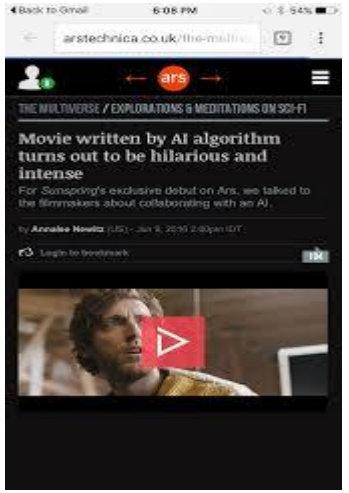
Many to Many



Time

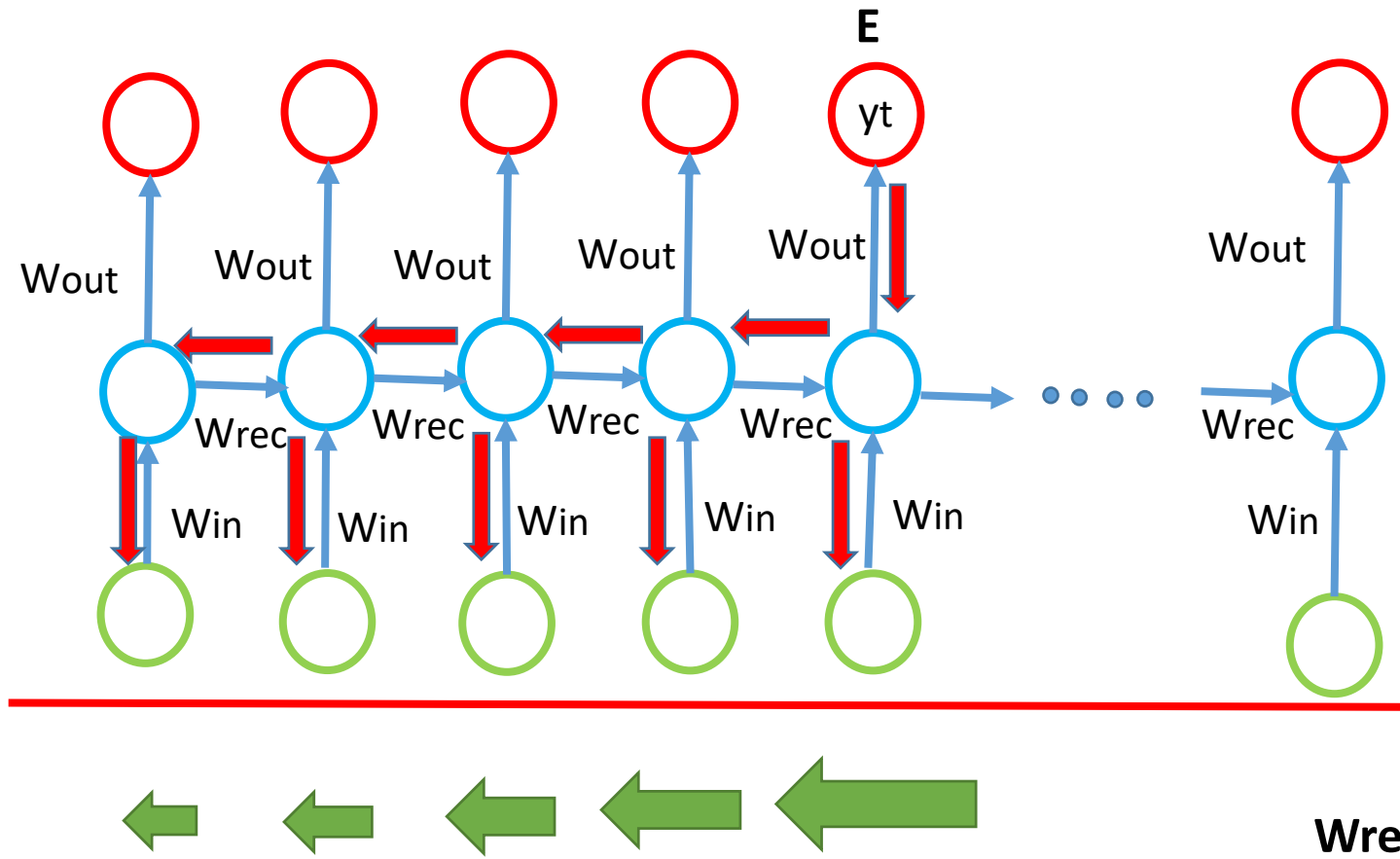
Many to Many





Time

The Vanishing Gradient Problem



The Vanishing Gradient Problem Solution:

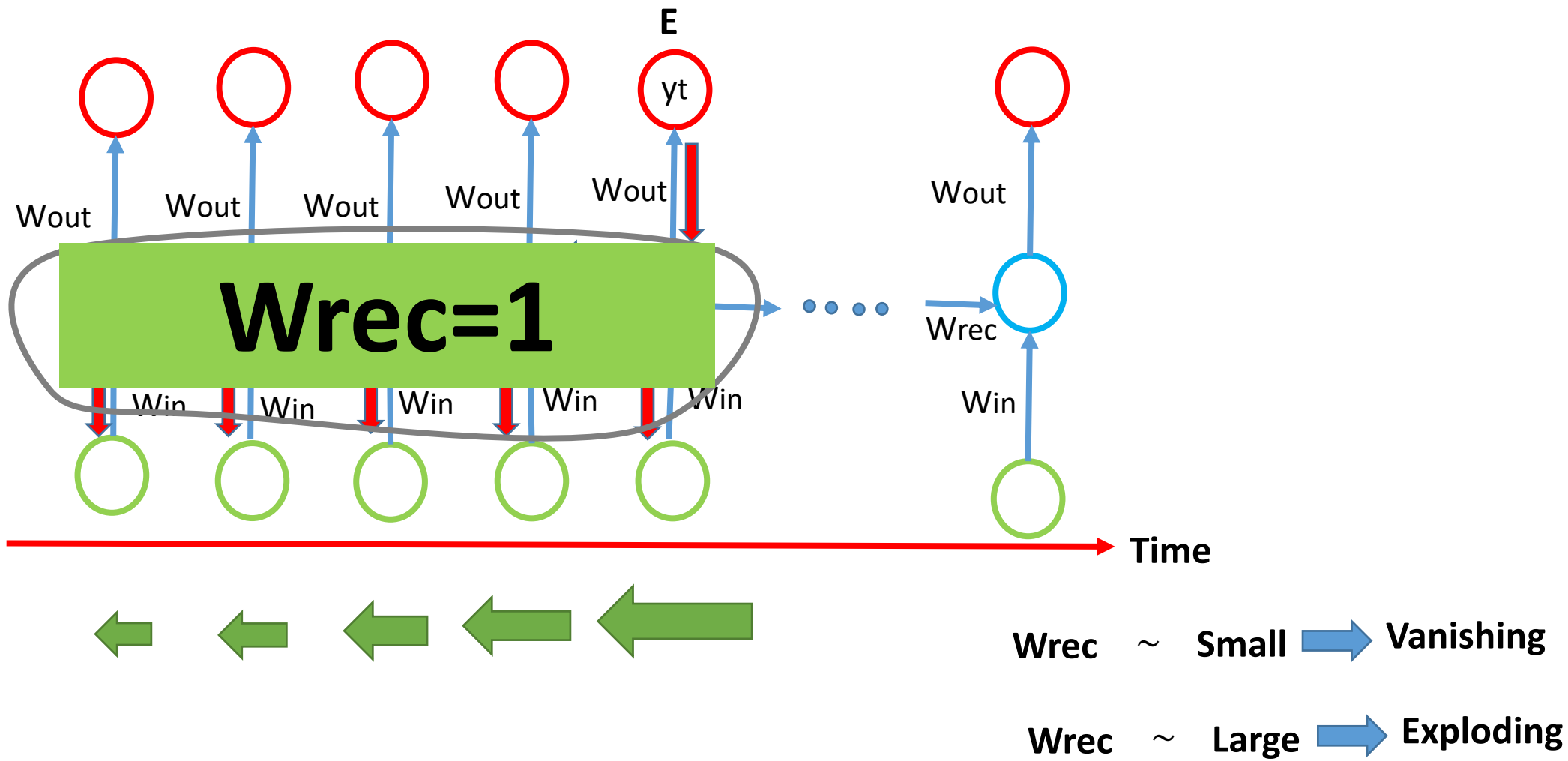
1. **Exploding**
   - **Truncated Backpropagation**
   - **Penalties**
   - **Grading Clipping**

2. **Vanishing Gradient**
   - **Weight initialization**
   - **Echo State Networks**
   - **Long Short-Term Memory Networks (LSTM)**

Wrec ~ Small ⇒ Vanishing

Wrec ~ Large ⇒ Exploding

LSTMs

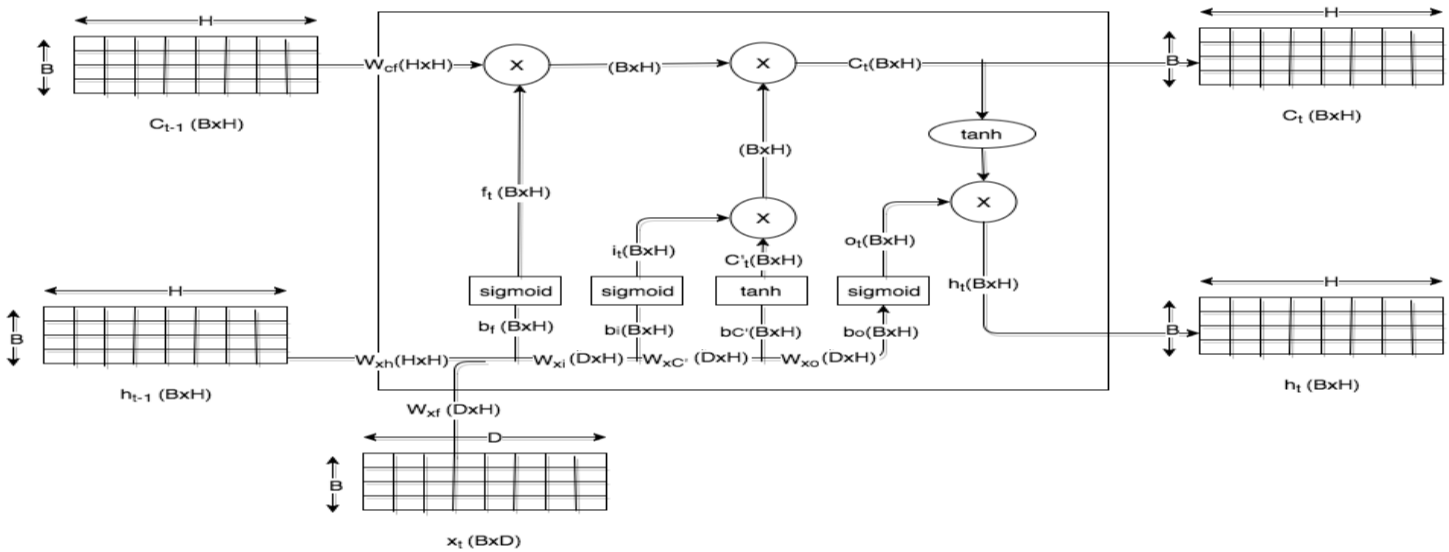


Wrec ~ Small ⇒ Vanishing

Wrec ~ Large ⇒ Exploding

LSTMs Dimension Computation

$$
\begin{aligned}
i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right) \\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \\
c_t &= f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \\
o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right) \\
h_t &= o_t \tanh(c_t)
\end{aligned}
$$



Input Dimensions

Input_size=xt=D=244x1

Hidden_size=ct-1=ht-1=H=1x128

Weights Dimensions

Wxi=DxH=244x128

Wxf=DxH=244x128

Wxc=DxH=244x128

Wxo=DxH=244x128

Weights Dimensions hidden states

Whi=HxH=128x128

Whf=HxH=128x128

Whc=HxH=128x128

Wco=HxH=128x128

Biases Dimensions

bi=H=1x128

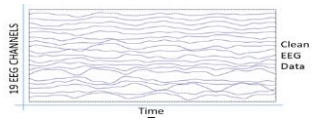
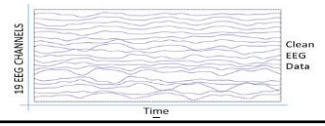
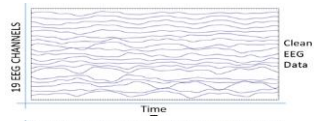
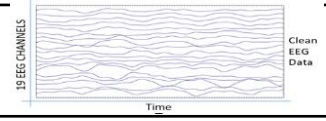
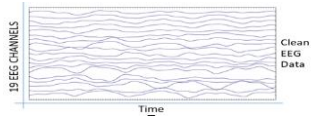
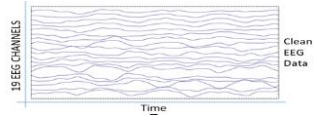
bf=H=1x128

bc=H=1x128

bo=H=1x128

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Basic dimensions in LSTM models for Training and Testing

No of samples	Classes	Dimension	Labels
1	Patient1	19x256 	0
2	Patient1	19x256 	0
	Patient1	----	0
	Patient1	---	0
50	Patient1	19x256 	0
51	Patient2	19x256 	1
	Patient2	19x256 	1
	Patient2	----	1
99	Patient2	----	1
100	Patient2	19x256 	1

Total samples x timestep x features

100 x 256 x 19

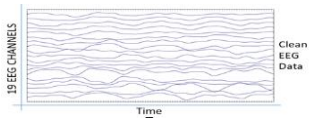
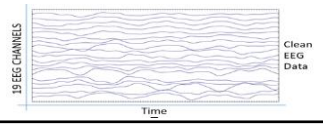
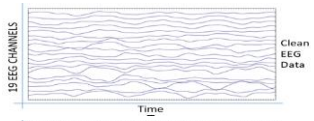
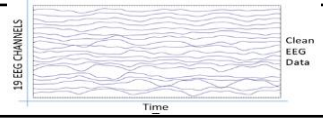
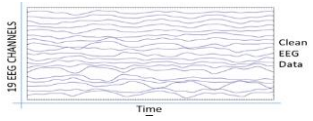
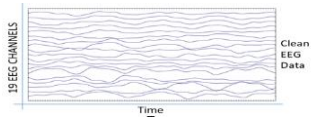
Training samples

80 x 256 x 19

Testing samples

20 x 256 x 19

Basic Dimensions in 1D-CNN for Training and Testing

No of samples	Classes	Dimension	Labels
1	Patient1	19x256 	0
2	Patient1	19x256 	0
	Patient1	----	0
	Patient1	---	0
50	Patient1	19x256 	0
51	Patient2	19x256 	1
	Patient2	19x256 	1
	Patient2	----	1
99	Patient2	----	1
100	Patient2	19x256 	1

Total samples x timestep x features

100 x 256 x 19

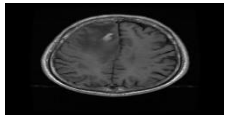
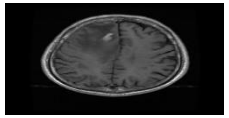
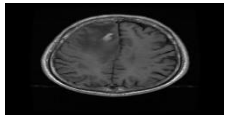
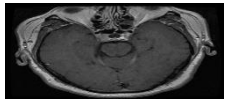
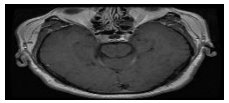
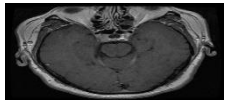
Training samples

80 x 256 x 19

Testing samples

20 x 256 x 19

Basic Dimensions in 2D-CNN for Training and Testing

No of samples	Classes	Dimension	Labels
1	Patient1	224x224x3 	0
2	Patient1	224x224x3 	0
	Patient1	----	0
	Patient1	---	0
50	Patient1	224x224x3 	0
51	Patient2	224x224x3 	1
	Patient2	224x224x3 	1
	Patient2	----	1
99	Patient2	----	1
100	Patient2	224x224x3 	1

Total samples x row x cols x channels

100 x 224x224x3

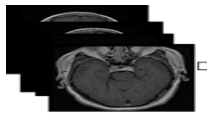
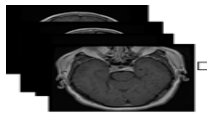
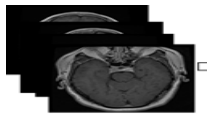
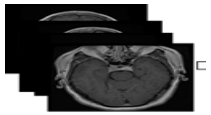
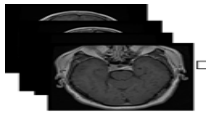
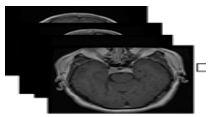
Training samples

80x224x224x3

Testing samples

20 x 224x224x3

Basic Dimensions in 3D-CNN for Training and Testing

No of samples	Classes	Dimension	Labels
1	Patient1	30x224x224x3 	0
2	Patient1	30x224x224x3 	0
	Patient1	----	0
	Patient1	---	0
50	Patient1	30x224x224x3 	0
51	Patient2	30x224x224x3 	1
	Patient2	30x224x224x3 	1
	Patient2	----	1
99	Patient2	----	1
100	Patient2	30x224x224x3 	1

Total samples x slices rows x cols x channels

100 x30x 224x224x3

Training samples

80x30x224x224x3

Testing samples

20 x30x 224x224x3

Data pre-processing and Acquisitions Libraries

Python
https://www.w3schools.com/python/python_intro.asp
NumPy
https://numpy.org/
Pandas
https://pandas.pydata.org/
Matplotlib
https://matplotlib.org/
ITK(segmentation and registration toolkit)
https://itk.org/
ITK-snap for 3d Modeling and visualization
http://www.itksnap.org/pmwiki/pmwiki.php
NiBabel processing nifty (Neuroimaging Informatics Technology Initiative) image
https://nipy.org/nibabel/gettingstarted.html

Libraries used for Deep Learning and Machine Learning Implementation

Machine Learning library

Scikit-Learn
https://scikit-learn.org/stable/
Scikit-learn plot
https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
Yellowbrick (visualization)
https://www.scikit-yb.org/en/latest/
MATLAB
https://fr.mathworks.com/products/matlab.html

Deep Learning libraries

Tensorflow
https://www.tensorflow.org/tutorials
Pytorch
https://pytorch.org/docs/stable/index.html
Keras
https://keras.io/
MATLAB
https://fr.mathworks.com/products/matlab.html


Q&A