

**CHANGE DETECTION IN SATELLITE IMAGES  
USING CONVOLUTION NEURAL NETWORKS AND  
PCA K-MEANS CLUSTERING**

**A Major project**

Submitted in partial fulfilment of the requirements for the award of the  
degree

Of

**BACHELOR OF TECHNOLOGY**

**IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

**By**

**Rudra Pratap Dhara**

**16H61A0441**

**M . Manoj Kumar**

**16H61A0430**

**K . Pradeep Kumar**

**16H61A0426**

Under the guidance of

**Mrs. NEETU SRIVATSAVA**

Assistant Professor

Department of ECE



Department of Electronics and Communication Engineering

**ANURAG GROUP OF INSTITUTIONS**

**(Formerly CVSR College of Engineering)**

**(AUTONOMOUS)**

**(Affiliated to Jawaharlal Nehru Technological University, Hyderabad)**

**Venkatapur(V) ,Ghatkesar(M), Medchal Dist-500088**

**Year of Submission: 2016-2020**

**ANURAG GROUP OF INSTITUTIONS**  
**(Formerly CVSR COLLEGE OF ENGINEERING)**  
**(AUTONOMOUS)**

(Affiliated to Jawaharlal Nehru Technological University, Hyderabad

Venkatapur(V),Ghatkesar(M), Medchal Dist-500088

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION  
ENGINEERING**



**CERTIFICATE**

This is to certify that the project entitled “change detection in satellite images using convolution neural networks and PCA k-means clustering”

**being submitted by**

<b>Rudra Pratap Dhara</b>	<b>16H61A0441</b>
<b>M . Manoj Kumar</b>	<b>16H61A0430</b>
<b>K . Pradeep Kumar</b>	<b>16H61A0426</b>

in the partial fulfilment for the award of the degree of bachelor of technology in Electronics and Communication Engineering to the Jawaharlal Nehru Technological University, Hyderabad. This is a record of bonafide work carried out under my guidance and supervision. The results embodied in this project report have not been submitted to any other university or institute for the award of any degree or diploma.

**Mrs. NEETU SRIVATSAVA**

Internal Guide

Assistant Professor

**Dr. S.SATHEES KUMARAN**

Head of the Department

Department of ECE

External Examiner

## **ACKNOWLEDGEMENT**

This project is an acknowledgement to the inspiration, drive and technical assistance contributed by many individuals. This project would have never seen light of this day without the help and guidance we have received. We would like to express our gratitude to all the people behind the screen who helped us to transform an idea into a real application.

It's our privilege and pleasure to express our profound sense of gratitude to **Mrs. NEETU SRIVATSAVA, Assistant Professor**, Department of ECE for her guidance throughout this dissertation work.

We express our sincere gratitude to **Dr. S. SATHEES KUMARAN, Head of Department**, Electronics and Communication Engineering for his precious suggestions for the successful completion of this project. He is also a great source of inspiration to our work.

We would like to express our deep sense of gratitude to **Dr. K.S. RAO, Director**, Anurag Group of Institutions for his tremendous support, encouragement and inspiration.

Lastly, we thank almighty, our parents, friends for their constant encouragement without which this assignment would not be possible. We would like to thank all the other staff members, both teaching and non-teaching, which have extended their timely help and eased my work.

**By**

**Rudra Pratap Dhara**

**16H61A0441**

**M . Manoj Kumar**

**16H61A0430**

**K . Pradeep Kumar**

**16H61A0426**

## **DECLARATION**

We hereby declare that the result embodied in this project report entitled “**change detection in satellite images using convolution neural networks and PCA k-means clustering**” is carried out by us during the year 2016-2020 for the partial fulfilment of the award of **Bachelor of Technology in Electronics and Communication Engineering**, from **ANURAG GROUP OF INSTITUTION** (formerly CVSR college of Engineering). We have not submitted this project report to any other Universities / Institute for the award of any degree.

**By**

**Rudra Pratap Dhara**

**16H61A0441**

**M . Manoj Kumar**

**16H61A0430**

**K . Pradeep Kumar**

**16H61A0426**

## TABLE OF CONTENTS

Title	Page Number
Acknowledgement	03
Declaration	04
Abstract	10
CHAPTER 1: INTRODUCTION	11-15
1.1 What is Artificial Intelligence?	11
1.1.1 Subsets of Artificial Intelligence	11
1.2 What is Machine Learning?	12
1.2.1 Limitations of Machine Learning	12
1.3 What is Deep Learning?	13
1.3.1 How Deep Learning Works?	13-14
1.4 What are Auto-encoders?	14
1.5 What is Principal component analysis (PCA)	15
1.5.1 Step by step Computation of PCA	15
CHAPTER 2: LITERATURE SURVEY	16-28
CHAPTER 3: DEEP LEARNING IN CONSIZE	29-37
3.1 Deep Learning	29-30
3.2 Linear/Logistic Regression	30-31
3.2.1 Linear Regression	30-31
3.2.2 Logistic Regression	31
3.3 Activation Function	31-32
3.4 Weights	32
3.4.1 Use of Weights	32
3.5 Bias	32
3.6 Neural Network	32-34
3.7 Other important concepts of Neural Networks	34
3.7.1 Training	34
3.7.2 Error	35
3.7.3 Forward Propagation	35

Title	Page Number
3.7.4 Gradient Descent	35-36
3.7.5 Back Propagation	36-37
3.7.6 Regularization	37
3.7.7 Optimisation	37
CHAPTER 4: AUTO-ENCODER ALGORITHM	38-45
4.1 Introduction	38
4.2 Architecture	39
4.3 Implementation	40-42
4.3.1 Visualization	41-42
4.3.2 Advice	42
4.4 De-noising Auto-Encoders	42-43
4.4.1 Visualization	43
4.5 Sparse Auto-Encoders	44
4.6 Use cases	45
CHAPTER 5: PRINCIPAL COMPONENT ANALYSIS (PCA) ALGORITHM	46-53
5.1 What is PCA?	46-47
5.2 When to use PCA?	47
5.3 How does PCA work?	47-52
5.4 Why does PCA work?	52
5.5 Are there any extensions to PCA?	53
CHAPTER 6: TECHNOLOGIES USED IN THE PROJECT	54-66
6.1 Tensor Flow	54-60
6.1.1 What is Tensor Flow?	54
6.1.2 History of Tensor Flow	54
6.1.3 Tensor Flow Architecture	55
6.1.4 Where can Tensor Flow run?	55

Title	Page Number
6.1.5 Introduction to components of Tensor Flow	56
6.1.6 Why is Tensor Flow popular?	56
6.1.7 List of prominent algorithms supported by TensorFlow	57
6.1.8 Simple Tensor Flow example	57-58
6.1.9 Summary	59-60
6.2 OpenCV	60-62
6.2.1 Introduction	60-61
6.2.2 History	61-62
6.2.3 Applications of OpenCV	62
6.2.4 OpenCV Functionality	62
6.3 Django	63-66
6.3.1 Introduction	63
6.3.2 History	63-64
6.3.2.1 Django Version History	64
6.3.3 Popularity	65
6.3.4 Features of Django	65-66
6.3.5 Django installation	66
CHAPTER 7: PROCESS INVOLVED IN THE PROJECT	67-76
7.1 Block Diagram	67
7.2 Process Involved	67-68
7.3 Source Code	69-76
CHAPTER 8: OUTPUTS & RESULTS	77-80
CHAPTER 9: FUTURE SCOPE & APPLICATIONS	81
CHAPTER 10: CONCLUSION & REFERENCES	82-84

## LIST OF FIGURES

Figure Number	Title	Page Number
1.1	Artificial Intelligence	11
1.2	Curse of Dimensionality	12
1.3	Layers of Deep Neural Network	13
1.4	Working of Deep Learning	14
1.5	Process involved in Auto-encoders	14
2.1	ConvNets	27
3.1	Fields of Deep Learning	29
3.2	Layers in Deep Learning	29
3.3	Why Deep Learning?	30
3.4	Machine Learning Vs Deep Learning	30
3.5	Linear Regression	31
3.6	Logistic Regression	31
3.7	Neural Network	32
3.8	Graphical Representation 1	33
3.9	Graphical Representation 2	33
3.9.1	Functional Representation	34
3.9.2	Forward Propagation	35
4.1	Auto-encoder Process	38
4.2	Architecture	39
4.3	Implementation	40
4.4	MINST Dataset	40
4.5	Original & reconstructed images	41
4.6	De-noising auto-encoder	42
4.7	Process of De-noising	43
4.8	Process of auto-encoding	43
4.9	Regularized model	44
5.1	Original dataset	48
5.2	Output from PCA	48



Figure Number	Title	Page Number
5.3	Comparison of original dataset & output	50
5.4	PCA plot	51
6.1	Simple TensorFlow example	57
6.2	OpenCV Example 1	60
6.3	OpenCV Example 2	61

## ABSTRACT

With the popular use of high resolution remote sensing (HRRS) satellite images, a huge research efforts have been placed on change detection (CD) problem. An effective feature selection method can significantly boost the final result. While hand-designed features have proven difficulties to design features that effectively capture high and mid-level representations, the recent developments in machine learning (Deep Learning) omit this problem by learning hierarchical representation in an unsupervised manner directly from data without human intervention. In this letter, we propose approaching the change detection problem from a feature learning perspective. A novel deep Convolutional Neural Networks (CNN) features based HR satellite images change detection method is proposed. The main guideline is to produce a change detection map directly from two images using a pre-trained CNN. This method can omit the limited performance of hand-crafted features. Firstly, CNN features are extracted through different Convolutional layers. Then, a concatenation step is evaluated after a normalization step, resulting in a unique higher dimensional feature map. Finally, a change map was computed using pixel-wise Euclidean distance. Our method has been validated on real bi-temporal HRRS satellite images according to qualitative and quantitative analyses. The results obtained confirm the interest of the proposed method.

# CHANGE DETECTION IN SATELLITE IMAGES USING CONVOLUTIONAL NEURAL NETWORK AND PCA K- MEANS CLUSTERING

## CHAPTER 1

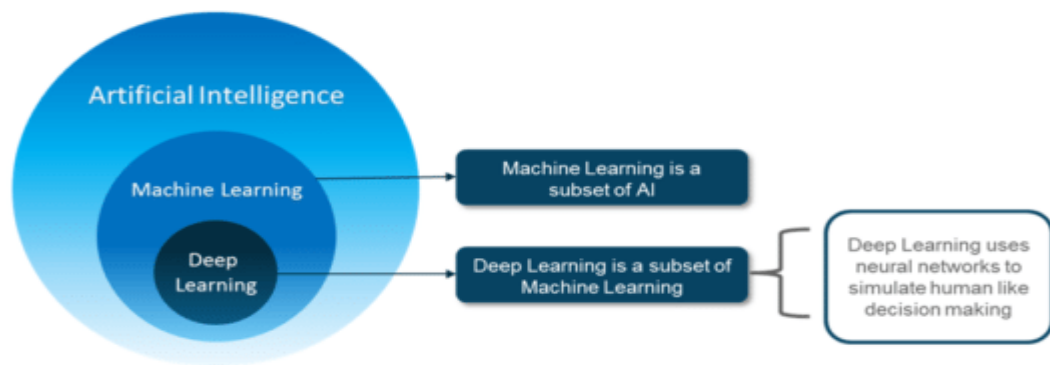
### INTRODUCTION

#### 1.1 WHAT IS ARTIFICIAL INTELLIGENCE?

Artificial Intelligence is nothing but the capability of a machine to imitate intelligent human behavior. AI is achieved by mimicking a human brain, by understanding how it thinks, how it learns, decides, and work while trying to solve a problem.

##### 1.1.1 SUBSETS OF ARTIFICIAL INTELLIGENCE

Till now, you would have heard a lot about Artificial Intelligence, Machine Learning and Deep Learning. However, do you know the relationship between all three of them? Basically, Deep learning is a sub-field of Machine Learning and Machine Learning is a sub-field of Artificial Intelligence as shown in the image below:



**Fig 1.1 Artificial Intelligence**

When we look at something like **AlphaGo**, it's often portrayed as a big success for deep learning, but it's actually a combination of ideas from several different fields of AI and machine learning. In fact, you would be surprised to hear that the idea behind deep neural networks is not new but dates back to 1950's. However, it became possible to practically implement it because of the high-end resource capability available nowadays.

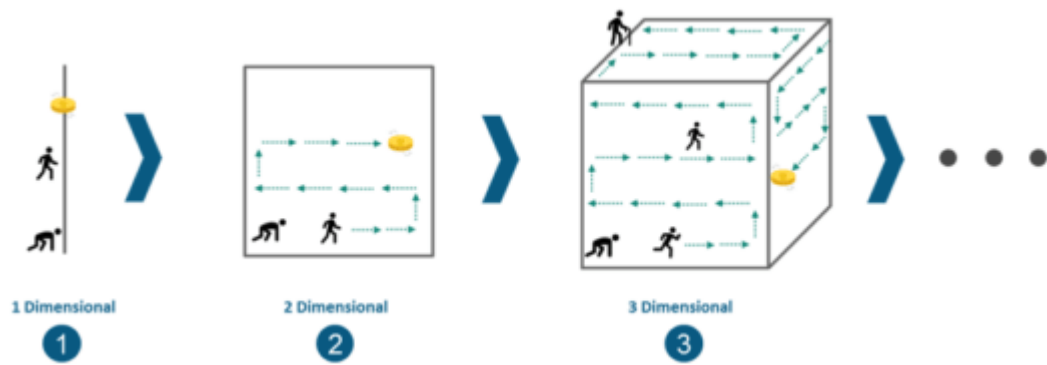
So, moving ahead in this deep learning tutorial blog, let's explore Machine Learning followed by its limitations.

## 1.2 WHAT IS MACHINE LEARNING?

Machine Learning is a subset of Artificial Intelligence which provide computers with the ability to learn without being explicitly programmed. In machine learning, we do not have to define explicitly all the steps or conditions like any other programming application. On the contrary, the machine gets trained on a training dataset, large enough to create a model, which helps machine to take decisions based on its learning.

### 1.2.1 LIMITATIONS OF MACHINE LEARNING

Machine Learning is not capable of handling high dimensional data that is where input & output is quite large. Handling and processing such type of data becomes very complex and resource exhaustive. This is termed as **Curse of Dimensionality**. To understand this in simpler terms, let's consider the following image:

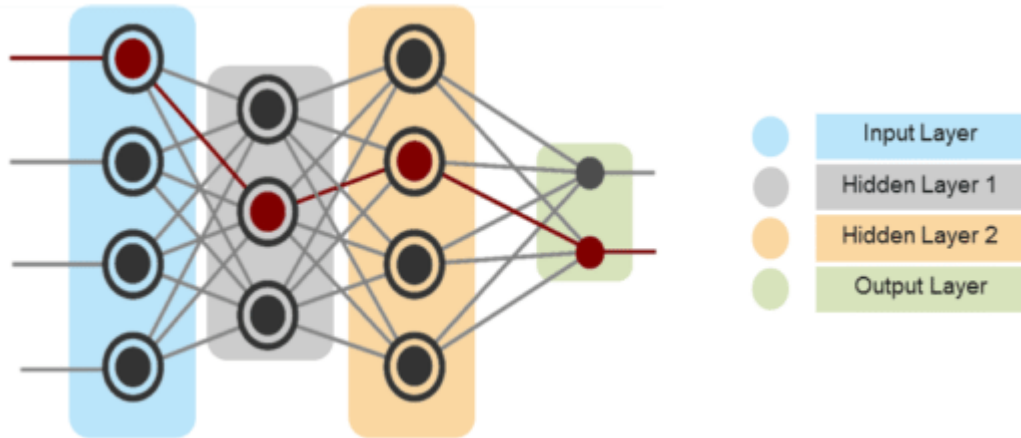


### 1.2 Curse of Dimensionality

Hence, you can observe the complexity is increasing as the dimensions are increasing. And in real-life, the high dimensional data that we were talking about has thousands of dimensions that makes it very complex to handle and process. The high dimensional data can easily be found in use-cases like Image processing, NLP, Image Translation etc.

Machine learning was not capable of solving these use-cases and hence, Deep learning came to the rescue. Deep learning is capable of handling the high dimensional data and is also efficient in focusing on the right features on its own. This process is called feature extraction. Now, let's move ahead in this Deep Learning Tutorial and understand how deep learning works.

## 1.3 WHAT IS DEEP LEARNING?



**Fig 1.3 Layers of Deep Neural Network**

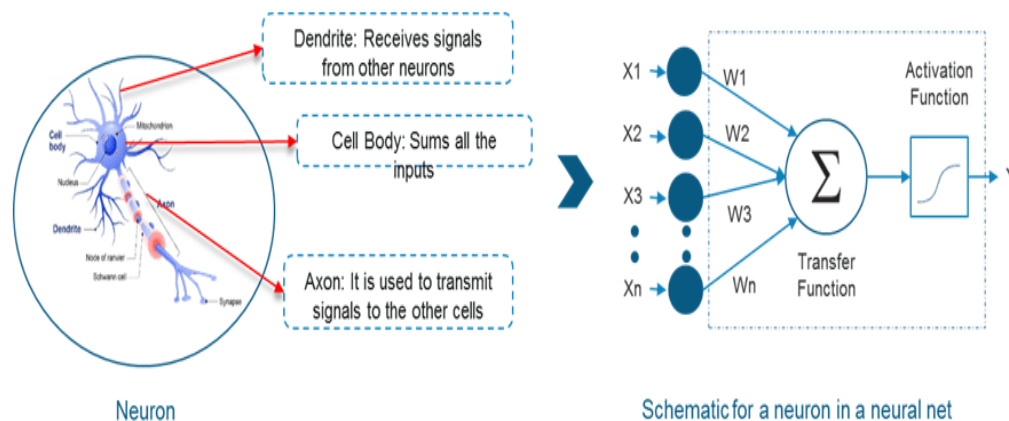
Any Deep neural network will consist of three types of layers:

- The Input Layer
- The Hidden Layer
- The Output Layer

- 1** In the above diagram, the first layer is the input layer which receives all the inputs and the last layer is the output layer which provides the desired output.
- 2** All the layers in between these layers are called hidden layers. There can be n number of hidden layers thanks to the high end resources available these days.
- 3** The number of hidden layers and the number of perceptrons in each layer will entirely depend on the use-case you are trying to solve.

### 1.3.1 HOW DEEP LEARNING WORKS?

In an attempt to re-engineer a human brain, Deep Learning studies the basic unit of a brain called a brain cell or a neuron. Inspired from a neuron an artificial neuron or a perceptron was developed. Now, let us understand the functionality of biological neurons and how we mimic this functionality in the perceptron or an artificial neuron:



**Fig 1.4 Working of Deep Learning**

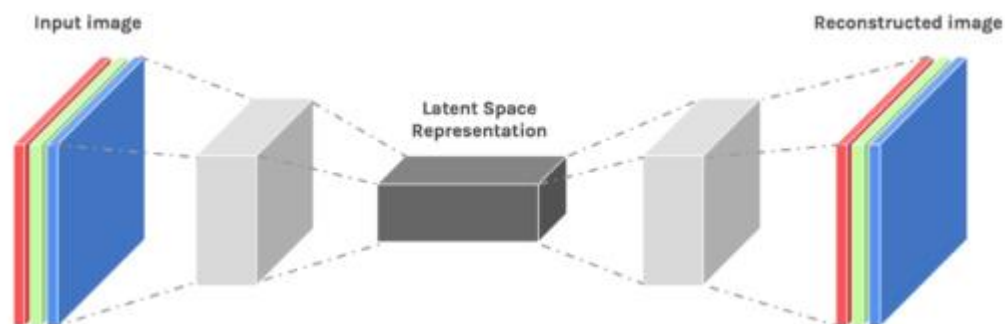
If we focus on the structure of a biological neuron, it has dendrites which is used to receive inputs. These inputs are summed in the cell body and using the Axon it is passed on to the next biological neuron as shown in the above image.

Similarly, a perceptron receives multiple inputs, applies various transformations and functions and provides an output.

As we know that our brain consists of multiple connected neurons called neural network, we can also have a network of artificial neurons called perceptrons to form a Deep neural network. So, let's move ahead in this Deep Learning Tutorial to understand how a Deep neural network looks like.

## 1.4 WHAT ARE AUTO-ENCODERS?

An auto-encoder **neural network** is an **Unsupervised Machine learning** algorithm that applies back propagation, setting the target values to be equal to the inputs. Auto-encoders are used to reduce the size of our inputs into a smaller representation. If anyone needs the original data, they can reconstruct it from the compressed data.



**Fig 1.5 Process involved in Auto-encoders**

We have a similar machine learning algorithm i.e; PCA which does the same task. So you might be thinking why do we need auto-encoders then? Let's continue this auto-encoders Tutorial and find out the reason behind using auto-encoders.

## **1.5 WHAT IS PRINCIPAL COMPONENT ANALYSIS (PCA)?**

Principal components analysis (PCA) is a dimensionality reduction technique that enables you to identify correlations and patterns in a data set so that it can be transformed into a data set of significantly lower dimension without loss of any important information.

The main idea behind PCA is to figure out patterns and correlations among various features in the data set. On finding a strong correlation between different variables, a final decision is made about reducing the dimensions of the data in such a way that the significant data is still retained.

Such a process is very essential in solving complex data-driven problems that involve the use of high-dimensional data sets. PCA can be achieved via a series of steps. Let's discuss the whole end-to-end process.

### **1.5.1 STEP BY STEP COMPUTATION OF PCA**

The below steps need to be followed to perform dimensionality reduction using PCA:

1. Standardization of the data
2. Computing the covariance matrix
3. Calculating the eigenvectors and eigenvalues
4. Computing the Principal Components
5. Reducing the dimensions of the data set

## CHAPTER 2

### LITURETURE SURVEY

**[1] Coppin, P.R., Bauer, M.E., 1996. Digital change detection in forest ecosystems with remote sensing imagery. Remote Sensing Reviews 13, 207–234.**

The world's forest ecosystems are in a state of permanent flux at a variety of spatial and temporal scales. Monitoring techniques based on multispectral satellite acquired data have demonstrated potential as a means to detect, identify, and map changes in forest cover. This paper, which reviews the methods and the results of digital change detection primarily in temperate forest ecosystems, has two major components. First, the different perspectives from which the variability in the change event has been approached are summarized, and the appropriate choice of digital imagery acquisition dates and interval length for change detection are discussed. In the second part, preprocessing routines to establish a more direct linkage between digital remote sensing data and biophysical phenomena, and the actual change detection methods themselves are reviewed and critically assessed. A case study in temperate forests (north-central U.S.A.) then serves as an illustration of how the different change detection phases discussed in this paper can be integrated into an efficient and successful monitoring technique. Lastly, new developments in digital change detection such as the use of radar imagery and knowledge-based expert systems are highlighted.

The data-gathering capabilities of space borne remote sensors have generated great enthusiasm over the prospect of establishing remote sensing-based systems for the continuous monitoring of forests and other renewable natural resources. Although Aldrich's prediction in 1975 of the accuracy of satellite remote sensing for monitoring forest change was not quickly or easily achieved, today it is well established that remote sensing imagery, particularly digital data, can be used to monitor and map changes in forests and other land cover types. Recent results (e.g., Collins and Woodcock, 1994; Coppin and Bauer, 1995) have demonstrated that it is feasible to develop automated forest cover monitoring methodologies. When the



inherent limitations of digital approaches are appropriately dealt with, preprocessing is adequately incorporated, and optimal change detection algorithms are selected, then Aldrich's prediction can be met. Although all of the possible change detection methods have not been applied to the same data for evaluation, consideration of more than 75 change detection studies leads to the following conclusions:

1. Vegetation indices are more strongly related to changes in the scene than the responses of single bands.
2. Accurate registration of multi-date imagery is a critical prerequisite of accurate change detection. However, residual miss-registration at the below-pixel level somewhat degrades areal assessment of change events at the change/no-change boundaries.
3. Some form of image matching or radiometric calibration is recommended to eliminate exogenous differences, for example due to differing atmospheric conditions, between image acquisitions. The goal, aptly stated by Hall et al. (1991), should be that following image rectification all images should appear as if they were acquired with the same sensor, while observing through the atmospheric and illumination conditions of the reference image.
4. Image differencing and linear transformations appear to perform generally better than the other methods of change detection. Differences among the different change maps and their accuracies are undoubtedly related to the complexity and variability in the spatial patterns and spectral-radiometric responses of forest ecosystems, as well to the specific attributes of the methods used.
5. The capability of using remote sensing imagery for change detection will be enhanced by improvements in satellite data that will become available over the next several years, and by the integration of remote sensing and GIS techniques, along with the use of supporting methods such as expert systems and ecosystem simulation models. Analysis of the literature provides ample evidence to support the conclusion that multi-date satellite imagery can be effectively used to detect and monitor changes in forests, especially forest disturbance. At the same time we agree

with the observation of Collins and Woodcock (1996) that one of the challenges confronting the remote sensing research community is to develop an improved understanding of the change detection process on which to build an understanding of how to match applications and change detection methods.

**[2] Celik, T. Unsupervised change detection in satellite images using principal component analysis and k-means clustering. IEEE Geosci. Remote Sens. Lett. 2009, 6, 772-776.**

In this letter, we propose a novel technique for unsupervised change detection in multi-temporal satellite images using principal component analysis (PCA) and k-means clustering. The difference image is partitioned into  $h \times h$  no overlapping blocks.  $S, S \leq h^2$ , orthonormal eigenvectors are extracted through PCA of  $h \times h$  no overlapping block set to create an eigenvector space. Each pixel in the difference image is represented with an  $S$ -dimensional feature vector which is the projection of  $h \times h$  difference image data onto the generated eigenvector space. The change detection is achieved by partitioning the feature vector space into two clusters using k-means clustering with  $k = 2$  and then assigning each pixel to the one of the two clusters by using the minimum Euclidean distance between the pixel's feature vector and mean feature vector of clusters. Experimental results confirm the effectiveness of the proposed approach.

An unsupervised change detection technique is developed by conducting k-means clustering on feature vectors which are extracted using  $h \times h$  local data projection onto eigenvector space. The eigenvector space is generated using PCA on  $h \times h$  no overlapping difference image blocks. The proposed method uses  $h \times h$  neighborhood to extract feature vector for each pixel so that it automatically considers the contextual information. The proposed algorithm is simple in computation yet effective in identifying meaningful changes which makes it suitable for real-time applications. It produces results comparable, even better, with the MRF-based approach [5], which requires computationally expensive data modeling and parameter estimation. Simulation results show that the proposed algorithm performs quite well on combating both the zero-mean Gaussian noise and

the speckle noise, which is quite attractive for change detection in optical and SAR images.

**[3] Johnson, R.D., Kasischke, E.S., 1998. Change vector analysis: a technique for the multispectral monitoring of land cover and condition. *International Journal of Remote Sensing* 19, 411–426.**

Change vector analysis (CVA) is a robust approach for detecting and characterizing radiometric change in multispectral remote sensing data sets. CVA is reviewed as a useful technique to: (1) process the full dimensionality of multispectral/multi-layer data so as to ensure detection of all change present in the data; (2) extract and exploit the 'components' of change in multispectral data; and (3) facilitate composition and analysis of change images. Examples drawn from various projects are included throughout this methodological discussion, in order to illustrate the CVA approach and suggest its potential utility.

Given that contemporary change detection applications often require or would benefit from the use of multispectral data, a full-dimensional data processing and analysis technique is frequently desirable or required to capture all changes. In a variety of change detection projects, experience has shown CVA to be an effective technique capable of full-filling this requirement both in stand-alone use, and when necessary, in conjunction with other procedures and/or algorithms. It is especially useful for projects in which:

- (1) The changes of interest and their spectral manifestation are not well known a priori
- (2) Changes of interest are known or thought to have high spectral variability and/or
- (3) Changes in both land cover type and condition may be of interest. Given the large archive of historical multispectral data, the emergence of hyper spectral sensors, and the continued expansion of the quantity and types of remote sensing data available, CVA may provide a capability of increasing importance in multivariate, full-dimensional change detection.

**[4] Bruzzone, Z.; Prieto, D.F. Automatic analysis of the difference image for unsupervised change detection. IEEE Trans. Geosci. Remote Sens. 2000, 38, 1171-1182.**

One of the main problems related to unsupervised change detection methods based on the “difference image” lies in the lack of efficient automatic techniques for discriminating between changed and unchanged pixels in the difference image. Such discrimination is usually performed by using empirical strategies or manual trial-and-error procedures, which affect both the accuracy and the reliability of the change-detection process. To overcome such drawbacks, in this paper, we propose two automatic techniques (based on the Bayes theory) for the analysis of the difference image. One allows an automatic selection of the decision threshold that minimizes the overall change detection error probability under the assumption that pixels in the difference image are independent of one another. The other analyzes the difference image by considering the spatial-contextual information included in the neighborhood of each pixel. In particular, an approach based on Markov Random Fields (MRF’s) that exploits intermixes class dependency contexts is presented. Both proposed techniques require the knowledge of the statistical distributions of the changed and unchanged pixels in the difference image. To perform an unsupervised estimation of the statistical terms that characterize these distributions, we propose an iterative method based on the Expectation-Maximization (EM) algorithm. Experimental results confirm the effectiveness of both proposed techniques.

In this paper, two techniques for the analysis of the difference image in unsupervised change-detection problems have been proposed. Such techniques, unlike classical ones, perform an automatic analysis of the difference image by exploiting theoretically well-founded methods. From a methodological viewpoint, the main innovation of this paper lies in the formulation of the unsupervised change-detection problem in terms of the Bayesian decision theory. In particular, we have proposed an iterative technique (based on the EM algorithm) that allows unsupervised estimations of the a priori probabilities and density functions associated with changed and unchanged pixels in the difference image. Such estimates make it possible to apply supervised methods in the context of

unsupervised change detection. Within this framework, two automatic techniques for the analysis of the difference image have been presented. The first technique is based on the assumption that the pixels in the difference image are independent of one another. Under this assumption, it allows the automatic selection of the decision-threshold value that minimizes the overall change-detection error probability.

It is worth noting that, thanks to the availability of the estimates provided by the EM algorithm, other decision strategies could also be adopted for the selection of the threshold (e.g., the Bayes rule for minimum cost). The second technique performs the analysis of the difference image by using an MRF approach that exploits the inter pixel class dependency context in order to improve the accuracy of the final change-detection map. For the sake of simplicity, a simple method for the MRF modeling has been used, even though more complex MRF models might be exploited (e.g., hierarchical MRF's and detail-preserving MRF's). In addition, more sophisticated MRF approaches might be adopted in order to further increase the accuracy and the degree of automation (e.g., automatic selection of the parameter) of the presented technique. Further research should be conducted to test the potential improvements associated with such approaches.

The experimental results reported in this paper confirm the effectiveness of both presented techniques. Such effectiveness depends mainly on the accuracy and the stability provided by the EM algorithm in estimating the statistical terms of the difference image. Thanks to this accuracy, the decision-threshold values provided by the proposed technique based on the pixel independence assumption turned out to be very close to the optimum ones for both considered data sets. This resulted in change-detection maps comparable to those provided by the corresponding minimum-error threshold. Further improvements in the change-detection accuracies were obtained by the proposed method, which exploits spatial-contextual information in the change-detection process. This method proved very effective even on images affected by high levels of noise. As a final remark, it is worth noting that we have formulated the EM algorithm under the assumption that the conditional density functions of classes can be modeled by Gaussian distributions. In several change-detection applications involving the use of images

acquired by passive sensors, this assumption seems to be a reasonable approximation. However, when the number of different typologies of land cover changes to be identified increases or when active sensors are used instead of passive ones, the Gaussian model might turn out to be inappropriate. In these cases, more general approaches to the mixture density estimation problem may represent powerful tools in obtaining accurate estimates of the density functions associated with changed and unchanged pixels.

**[5] Chen, C. Huo, Z. Zhou and H. Lu, "Unsupervised Change Detection in SAR Image using Graph Cuts," Geosciences and Remote Sensing Symposium, 2008. IGARSS 2008. IEEE International, Boston, MA, 2008, pp. III - 1162-III - 1165.**

In this paper, we present an unsupervised change detection approach in temporal sets of SAR images. The change detection is represented as a task of energy minimization and the energy function is minimized using graph cuts. Neighboring pixels are taken into account in a priority sequence according to their distance from the center pixel, and the energy function is formed based on Markov Random Field (MRF) model. Graph cuts algorithm is employed for computing maximum a-posteriori (MAP) estimates of the MRF. Experiments results obtained on a SAR data set confirm the effectiveness of the proposed approach. The comparisons between graph cuts algorithm and iterated conditional modes (ICM) algorithm about the quality of change map and running time of energy minimization illustrate that graph cuts algorithm is a huge improvement over ICM.

In this paper, we present an unsupervised change detection approach in temporal sets of SAR images using graph cuts. The change detection is represented in terms of energy minimization under MRFs framework, and the energy function is minimized using graph cuts. The proposed approach is capable of exploiting the spatial contextual information in images but also overcome the computational burden of traditional algorithm. Experiments results obtained on a SAR data set confirm the effectiveness of the proposed approach. It is also proven that graph cuts algorithm is a huge improvement over ICM and SA, and the minimum obtained by graph cuts is quite close to the global minimum. Future lines of research may be

related to: 1) more sophisticated model is adopted; 2) Other minimization approaches, loopy belief propagation (LBP), message passing, can be compared.

**[6] Bovolo, F. A multilevel parcel-based approach to change detection in very high resolution multitemporal images. IEEE Geosci. Remote Sens. Lett. 2009, 6, 33-37.**

This letter presents a novel parcel-based context sensitive technique for unsupervised change detection in very high geometrical resolution images. In order to improve pixel-based change-detection performance, we propose to exploit the spatial context information in the framework of a multilevel approach. The proposed technique models the scene (and hence changes) at different resolution levels defining multi-temporal and multilevel “parcels” (i.e., small homogeneous regions shared by both original images). Change detection is achieved by applying a multilevel change vector analysis to each pixel of the considered images. This technique properly analyzes the multilevel and multi-temporal parcel-based context information of the considered spatial position. The adaptive nature of multi-temporal parcels and their multilevel representation allow one a proper modeling of complex objects in the investigated scene as well as borders and details of the changed areas. Experimental results confirm the effectiveness of the proposed approach.

In this letter, a novel approach to change detection in VHR multi-temporal images has been presented. The approach is based on a representation of the spatiotemporal context of pixels and is ruled by precise hierarchical relationships between each pixel in the multi-temporal images and the regions that adaptively define the related context at different levels. Each pixel is characterized by a feature vector, which is obtained by computing the mean on the parcels at different scales for both images. The final change-detection map is computed by extending the well-known CVA procedure to the multilevel parcel-based feature vectors. Quantitative and qualitative results on the real VHR data set pointed out the effectiveness of the proposed technique. In particular, the proposed technique resulted in a sharply higher accuracy than the standard pixel-based CVA technique, which resulted in an unreliable change-detection map for the considered data set. In addition, once the

number  $L$  of considered segmentation levels was fixed, the proposed multilevel parcel-based method always obtained higher change-detection accuracy than the single-level parcel-based procedure.

The qualitative analysis of the change-detection maps pointed out that the proposed technique also resulted in a high fidelity in both homogeneous and border regions. In greater detail, the proposed multilevel approach resulted in change-detection maps that show both a more accurate modeling of geometrical details of changes than the single-level parcel-based method applied to low resolution level, and a better representation of homogeneous area than the single-level parcel-based methods applied at high resolution level. The most effective number and combination of resolution levels depend on the considered data set and the desired tradeoff between the accuracy in detail preservation and homogeneous area representation. It is worth noting that neglecting high resolution levels will result in a loss of sensibility to borders and geometrical details, whereas neglecting low resolution levels will result in a poor accuracy in homogeneous areas. The multilevel nature of the proposed technique allows one to also reduce false alarms that may be introduced from artifacts deriving from the pan-sharpening procedure. In this respect, after a given level, it is also possible to compute the contextual features by directly considering the low-resolution multispectral images, which are not affected from geometric and radiometric distortions. As future developments of this letter, we plan to do the following:

- 1) Extend the experimental analysis to VHR multi-temporal images acquired by other sensors
- 2) Introduce in the proposed approach a procedure for the adaptive selection of the number of scales and the kind of features to be considered for an optimal representation of the object in which the analyzed pixel is included; and
- 3) extend the use of the proposed method to the analysis of long series of multi-temporal images.



**[7] Kasetkasem and P. K. Varshney, "An image change detection algorithm based on Markov random field models," in IEEE Transactions on Geosciences and Remote Sensing, vol. 40, no. 8, pp. 1815-1823, Aug 2002.**

This paper addresses the problem of image change detection (ICD) based on Markov random field (MRF) models. MRF has long been recognized as an accurate model to describe a variety of image characteristics. Here, we use the MRF to model both noiseless images obtained from the actual scene and change images (CIs), the sites of which indicate changes between a pair of observed images. The optimum ICD algorithm under the maximum a posteriori (MAP) criterion is developed under this model. Examples are presented for illustration and performance evaluation.

**[8] A. Krizhevsky, I. Sutskever, and G. Hinton. Image Net classification with deep Convolutional neural networks. In NIPS, 2012.**

We trained a large, deep Convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five Convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce over fitting in the fully-connected layers we employed a recently-developed regularization method called “dropout” that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

**[9] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.**

In this work we investigate the effect of the Convolutional network depth on its accuracy in the large-scale image recognition setting. Our main contribution

is a thorough evaluation of networks of increasing depth using an architecture with very small ( $3 \times 3$ ) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers. These findings were the basis of our ImageNet Challenge 2014 submission, where our team secured the first and the second places in the localization and classification tracks respectively. We also show that our representations generalize well to other datasets, where they achieve state-of-the-art results. We have made our two best-performing ConvNet models publicly available to facilitate further research on the use of deep visual representations in computer vision

In this work we evaluated very deep Convolutional networks (up to 19 weight layers) for largescale image classification. It was demonstrated that the representation depth is beneficial for the classification accuracy, and that state-of-the-art performance on the ImageNet challenge dataset can be achieved using a conventional ConvNet architecture (LeCun et al., 1989; Krizhevsky et al., 2012) with substantially increased depth. In the appendix, we also show that our models generalize well to a wide range of tasks and datasets, matching or outperforming more complex recognition pipelines built around less deep image representations. Our results yet again confirm the importance of depth in visual representations.

**[10] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In ECCV, pages 818–833. Springer, 2014.**

Large Convolutional Network models have recently demonstrated impressive classification performance on the ImageNet benchmark Krizhevsky et al. [18]. However there is no clear understanding of why they perform so well, or how they might be improved. In this paper we explore both issues. We introduce a novel visualization technique that gives insight into the function of intermediate feature layers and the operation of the classifier. Used in a diagnostic role, these visualizations allow us to find model architectures that outperform Krizhevsky et al. on the ImageNet classification benchmark. We also perform an ablation study to discover the performance contribution from different model layers. We show our ImageNet model generalizes well to other datasets: when the softmax classifier is



**Figure 2.1:** Correlation analysis of ConvNets versus all the other descriptors in the Brazilian Coffee Scenes dataset. ConvNets are more correlated to BIC, ACC, and the other ConvNets, which perform better in this dataset.

Remote sensing image classification task. This evaluation adds two more scenarios in which pre-trained ConvNets generalize well. The results show that yes, the ConvNets generalize well even in domains considerably different from the ones they were trained for. We also compared ConvNets with a set of other visual descriptors. ConvNets achieved the highest accuracy rates for aerial images, while for coffee scenes they perform well but do not out-perform low-level color descriptors, such as BIC. A correlation analysis of ConvNets with other ConvNets as well as with other visual descriptors shows that they can potentially be combined to achieve even better results. Preliminary experiments fusing two ConvNets obtain state-of-the-art results for the well known UC Merced dataset. As future work, we intend to explore more opportunities for fusing ConvNet features and other descriptors based on our correlation analysis presented in this paper. We also would like to evaluate the impact of using features from lower layers of the ConvNets as well as to train ConvNets with specific remote sensing data. The evaluation of ConvNets in other remote sensing and aerial datasets is another interesting future work.

## CHAPTER 3

### DEEP LEARNING IN CONSIZE

#### 3.1 DEEP LEARNING:

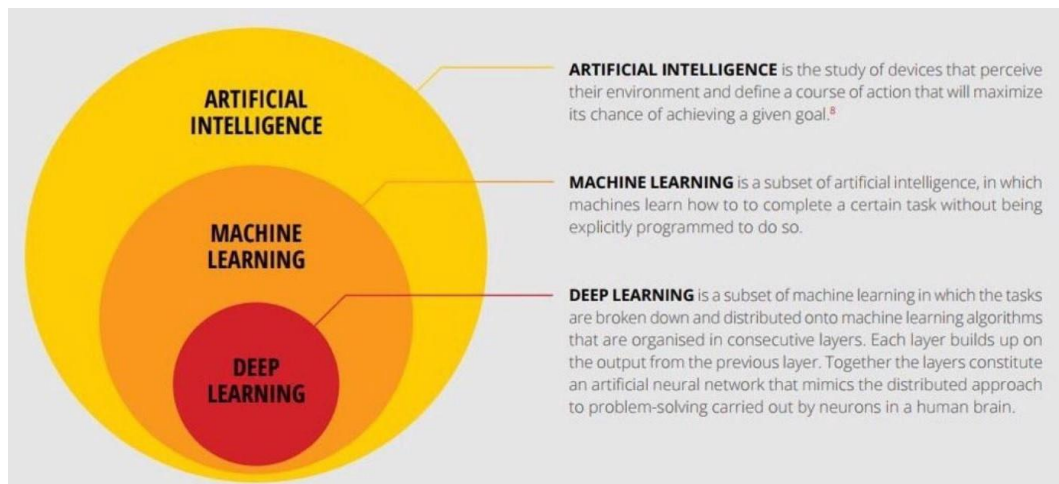


Fig : 3.1 Fields of Deep learning

Deep learning is a sub-field of machine learning dealing with algorithms inspired by the structure and function of the brain called artificial neural networks. In other words, it mirrors the functioning of our brains. Deep learning algorithms are similar to how nervous system structured where each neuron connected each other and passing information

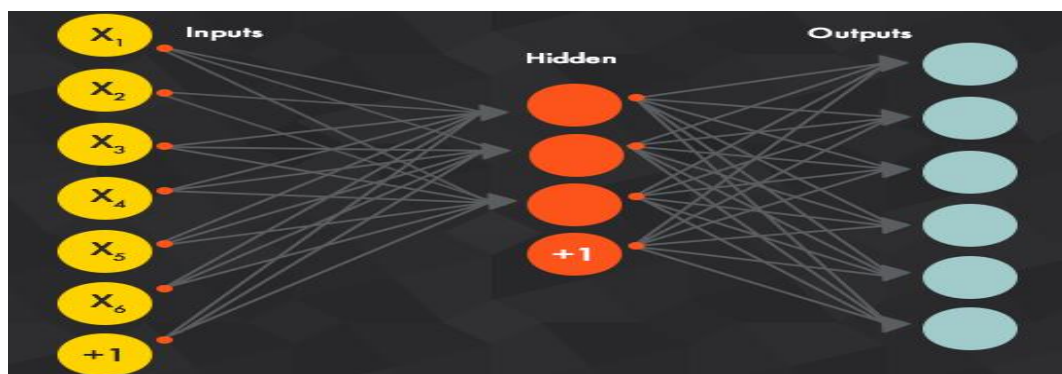
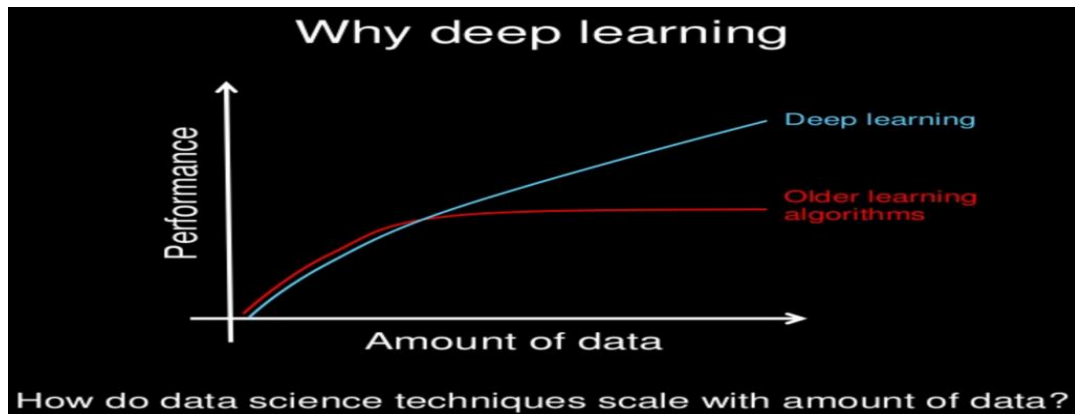


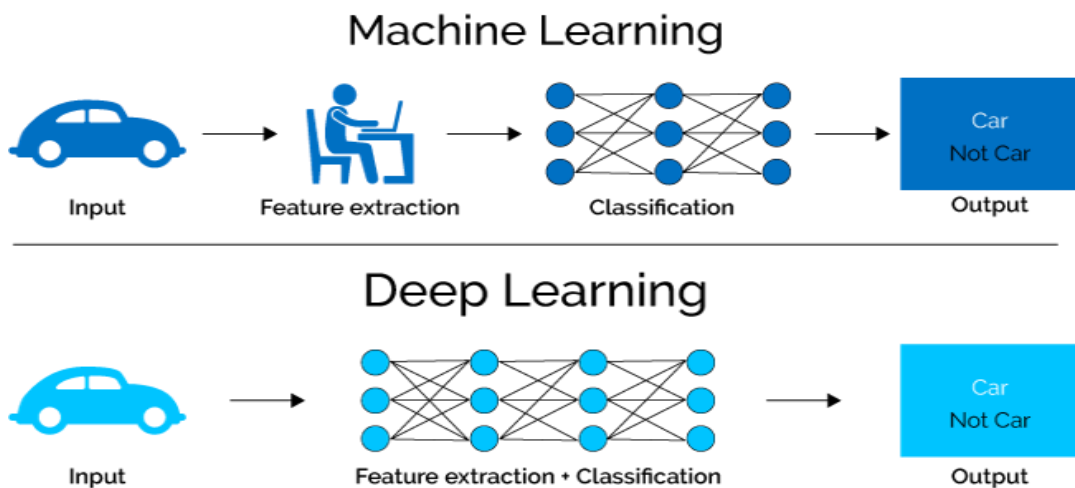
Fig : 3.2 Layers in Deep learning

Deep learning models work in layers and a typical model atleast have three layers. Each layer accepts the information from previous and pass it on to the next one.



**Fig : 3.3 why Deep learning ?**

Deep learning models tend to perform well with amount of data whereas old machine learning models stops improving after a saturation point.



**Fig : 3.4 Machine learning Vs Deep learning**

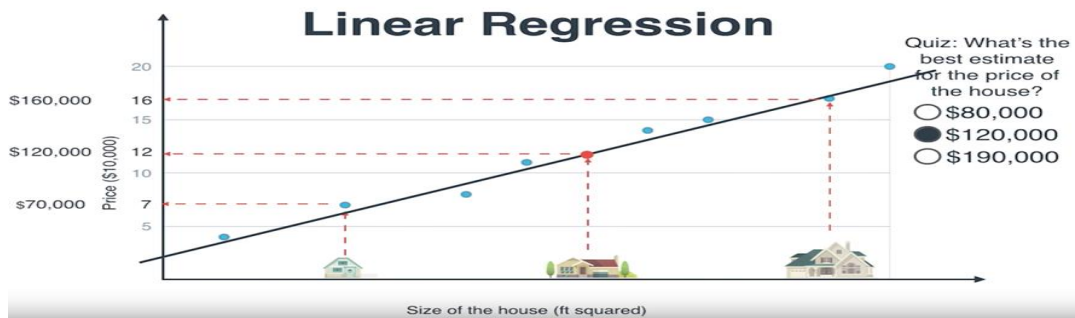
One of differences between machine learning and deep learning model is on the feature extraction area. Feature extraction is done by human in machine learning whereas deep learning model figure out by itself.

## **3.2 LINEAR/LOGISTIC REGRESSION**

We cannot start deep learning without explaining linear and logistics regression which is the basis of deep learning.

### **3.2.1 LINEAR REGRESSION**

It is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables.

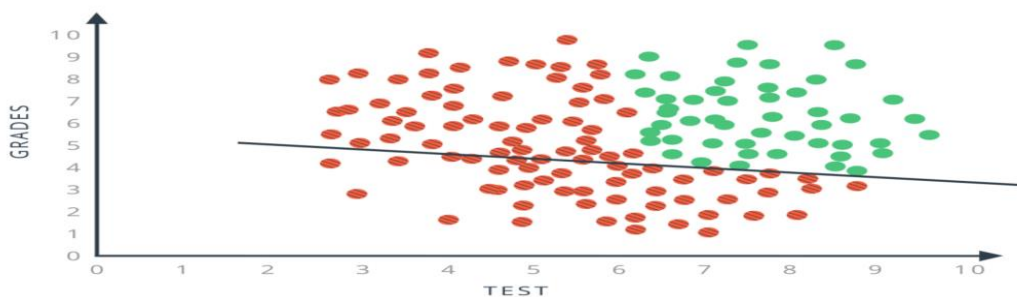


**Fig : 3.5 Linear Regression**

In this example, we have historical data based on the size of the house. We plot them into the graph as seen as dot points. Linear regression is the technique where finding a straight line between these points with less error (this will be explained later). Once we have a line with less error, we can predict the house price based on the size of the house.

### 3.2.2 LOGISTIC REGRESSION

It is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured in which there are only two possible outcomes: True or False.



**Fig : 3.6 Logistic Regression**

In this example, we have historical dataset of student which have passed and not passed based on the grades and test scores. If we need to know a student will pass or not based on the grade and test score, logistic regression can be used. In logistic regression, similar to linear regression, it will find best possible straight line that separate the two classification (passed and not passed).

### 3.3. ACTIVATION FUNCTION

Activation functions are functions that decide, given the inputs into the node, what should be the node's output? Because it's the activation function that decides the actual output, we often refer to the outputs of a layer as its "activations".

One of the simplest activation functions is the **Heaviside step function**. This function returns a **0** if the linear combination is less than 0. It returns a **1** if the linear combination is positive or equal to zero.

$$f(h) = \begin{cases} 0 & \text{if } h < 0 \\ 1 & \text{if } h \geq 0 \end{cases} \quad (1)$$

The output unit returns the result of  $f(h)$ , where  $h$  is the input to the output unit.

### 3.4. WEIGHTS

When input data comes into a neuron, it gets multiplied by a weight value that is assigned to this particular input. For example, the neuron above university example have two inputs, tests for test scores and grades, so it has two associated weights that can be adjusted individually.

#### 3.4.1 USE OF WEIGHTS

These weights start out as random values, and as the neural network learns more about what kind of input data leads to a student being accepted into a university, the network adjusts the weights based on any errors in categorization that the previous weights resulted in. This is called **training** the neural network.

Remember we can associate weight as  $m$  (slope) in the original linear equation.

$$y = mx + b \quad (2)$$

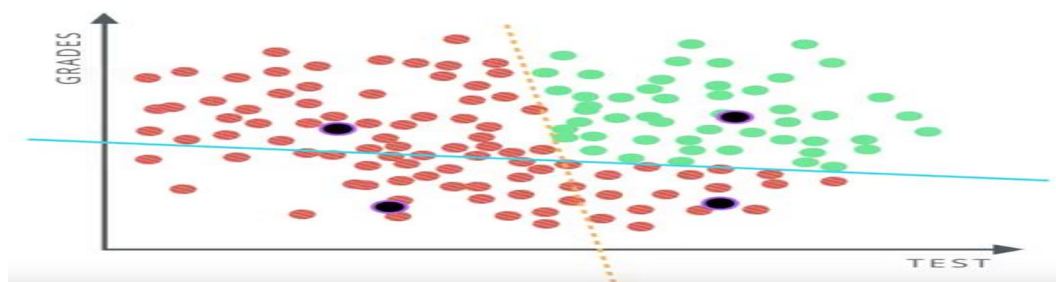
### 3.5. BIAS

Weights and biases are the learnable parameters of the deep learning models.

Bias represented as  $b$  in the above linear equation.

### 3.6. Neural Network

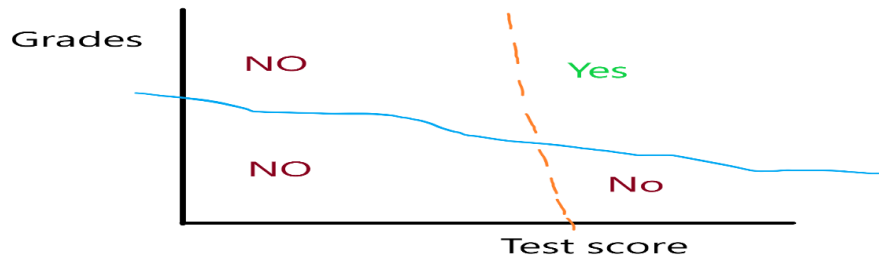
As explained above, deep learning is a sub-field of machine learning dealing with algorithms inspired by the structure and function of the brain called artificial neural networks. I will explain here how we can construct a simple neural network from the example. In the above example, Logistic regression is the technique to be used to separate data using single line. But most of the time we cannot classify the dataset using a single line with high accuracy.



**Fig : 3.7 Neural Network**



How about if we separate, data points with two lines.



**Fig : 3.8 Graphical Representation 1**

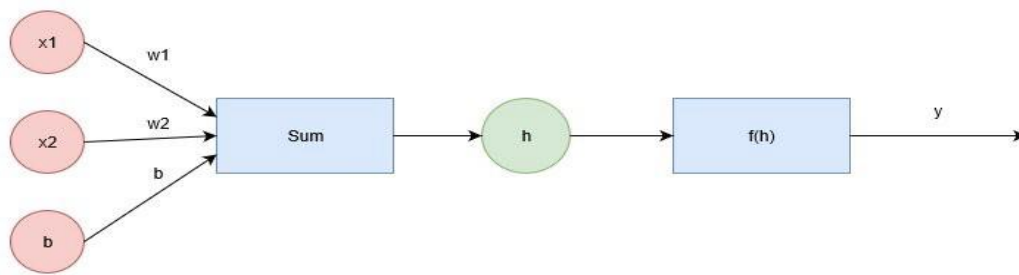
In this case, we say anything below blue line will be “No(not passed)” and above it will be “Yes(passed)”. Similarly, we say anything on the left side will be “No(not passed)” and on the right side “Yes(passed)”.



**Fig : 3.9 Graphical Representation 2**

As we have neurons in nervous system, we can define each line as one neuron and connected to next layer neurons along with neurons in the same layer. In this case we have two neurons that represents the two lines. The above picture is an example of simple neural network where two neurons accept that input data and compute yes or no based on their condition and pass it to the second layer neuron to concatenate the result from previous layer. For this specific example test score 1 and grade 8 input, the output will be “Not passed” which is accurate, but in logistic regression out we may get as “passed”. To summarize this, using multiple neurons in different layers, essentially we can increase the accuracy of the model. This is the basis of neural network.

The diagram below shows a simple network. The linear combination of the weights, inputs, and bias form the input  $h$ , which passes through the activation function  $f(h)$ , giving the final output, labeled  $y$ .



**Fig : 3.9.1 Functional Representation**

The good fact about this architecture, and what makes neural networks possible, is that the activation function,  $f(h)$  can be any function, not just the step function as shown.

For example, if you let  $f(h)=h$ , the output will be the same as the input. Now the output of the network is

$$y = \sum_i \omega_i \cdot x_i + b \dots\dots\dots (3)$$

This equation should be familiar to you, it's the same as the linear regression model! Other activation functions you'll see are the logistic (often called the sigmoid), tanh, and softmax functions.

$$\text{sigmoid}(x) = 1/(1+e^{-x}) \dots\dots\dots (4)$$

The sigmoid function is bounded between 0 and 1, and as an output can be interpreted as a probability for success. It turns out, again, using a sigmoid as the activation function results in the same formulation as logistic regression.

We can finally say output of the simple neural network based on sigmoid as below:

$$y = f(h) = \text{sigmoid}(\sum_i \omega_i \cdot x_i + b) \dots\dots\dots (5)$$

I will touch about learning process of neural networks briefly later, but in depth detail of learning of a particular model will be explained in coming articles in the series.

## **3.7. OTHER IMPORTANT CONCEPTS OF NEURAL NETWORKS**

### **3.7.1 TRAINING**

Weights start out as random values, and as the neural network learns more about what kind of input data leads to a student being accepted into a university(above example), the network adjusts the weights based on any errors in categorization that the previous weights resulted in. This is called **training** the neural network. Once we have the trained network, we can use it for predicting the output for the similar input.

### 3.7.2 ERROR

This very important concept to define how well a network performing during the training. In the training phase of the network, it make use of error value to adjust the weights so that it can get reduced error at each step. The goal of the training phase to minimize the error

**Mean Squared Error** is one of the popular error function. It is a modified version **Sum Squared Error**.

$$SSE = \sum_i (target^{(i)} - output^{(i)})^2$$

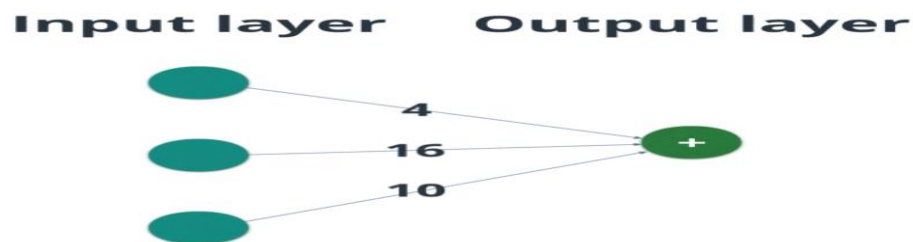
$$MSE = \frac{1}{n} \times SSE \dots\dots\dots (6)$$

Or we can write MSE as:

$$E = \frac{1}{2m} \sum_{\mu} (y^u - \hat{y}^u)^2 \dots\dots\dots (7)$$

### 3.7.3 FORWARD PROPOGATION

By propagating values from the first layer (the input layer) through all the mathematical functions represented by each node, the network outputs a value. This process is called a **forward pass**.



**Fig : 3.9.2 Forward Propagation**

### 3.7.4 GRADIENT DESCENT

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost). Gradient descent is best used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm. Gradient descent is used to find the minimum error by minimizing a “cost” function.

In the university example (explained it in the neural network section), the correct lines to divide the dataset is already defined. How does we find the correct line? As we know, weights are adjusted during the training process. Adjusting the weight will enable each neuron to correctly divide the dataset with given dataset.

To figure out how we’re going to find these weights, start by thinking about the goal. We want the network to make predictions as close as possible to the real

values. To measure this, we need a metric of how wrong the predictions are, the **error**. A common metric is the sum of the squared errors (SSE):

$$E = \frac{1}{2} \sum_{\mu} \sum_j \left( y_j^{\mu} - \hat{y}_j^{\mu} \right)^2 \dots\dots\dots (8)$$

Where  $\hat{y}$  is the prediction and  $y$  is the true value, and you take the sum overall output units  $j$  and another sum over all data points  $\mu$ .

The SSE is a good choice for a few reasons. The square ensures the error is always positive and larger errors are penalized more than smaller errors. Also, remember that the output of a neural network, the prediction, depends on the weights

$$\hat{y}_j^{\mu} = f \left( \sum w_{ij} x_i^{\mu} \right) \dots\dots\dots (9)$$

and accordingly the error depends on the weights

$$E = \frac{1}{2} \sum_{\mu} \sum_j \left( y_j^{\mu} - f \left( \sum_i w_{ij} x_i^{\mu} \right) \right)^2 \dots\dots\dots (10)$$

We want the network's prediction error to be as small as possible and the weights are the knobs we can use to make that happen. Our goal is to find weights  $w_{ij}$  that minimize the squared error  $E$ . To do this with a neural network, typically we use gradient descent.

With gradient descent, we take multiple small steps towards our goal. In this case, we want to change the weights in steps that reduce the error. Continuing the analogy, the error is our mountain and we want to get to the bottom. Since the fastest way down a mountain is in the steepest direction, the steps taken should be in the direction that minimizes the error the most.

### 3.7.5 BACK PROPOGATION

In neural networks, you forward propagate to get the output and compare it with the real value to get the error. Now, to minimize the error, you propagate backwards by finding the derivative of error with respect to each weight and then subtracting this value from the weight value. This is called back propagation.

Before, we saw how to update weights with gradient descent. The back propagation algorithm is just an extension of that, using the chain rule to find the error with the respect to the weights connecting the input layer to the hidden layer (for a two layer network).

Here is the back propagation algorithm,

Set the weight steps for each layer to zero

- The input to hidden weights  $\Delta w_{ij} = 0$  ..... (1)
- The output to hidden weights  $\Delta w_j = 0$  ..... (2)

For each record in training data:

- Make a forward pass through the network, calculating the output  $\hat{y}$
- Calculate the error gradient in the output unit,  $\delta^0 = (y - \hat{y}) f'(z)$  ..... (3)

$$\text{where , } z = \sum_j w_j a_j \text{ ..... (4)}$$

- Propagate the errors to the hidden layer  $\delta_j^h = \delta^0 w_j f'(h_j)$  ..... (5)
- Update the weight steps:

$$\Delta w_j = \Delta w_j + \delta^0 a_j \text{ ..... (6)}$$

$$\Delta w_{ij} = \omega_{ij} + \delta_j^h a_i \text{ ..... (7)}$$

- Update the weights where,  $\eta$  is the learning rate and m is the number of records

$$\Delta w_j = \omega_j + \eta \Delta w_j / m \text{ ..... (8)}$$

$$\Delta w_{ij} = \omega_{ij} + \eta \Delta w_{ij} / m \text{ ..... (9)}$$

### 3.7.6 REGULARISATION

Regularisation is the technique used to solve the over-fitting problem. Over-fitting happens when model is biased to one type of dataset. There are different types of regularisation techniques, I think the mostly used regularisation is dropout.

During the training, randomly selected neurons are not considered. We can set the number of neurons for the dropout. Their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass. I think best practice is to remove 20 % of neurons.

### 3.7.7 OPTIMISATION

Optimisation is technique used to minimize the loss function of the network. There are different type of optimisation algorithms. However, Gradient decent and it's variants are popular ones these days.

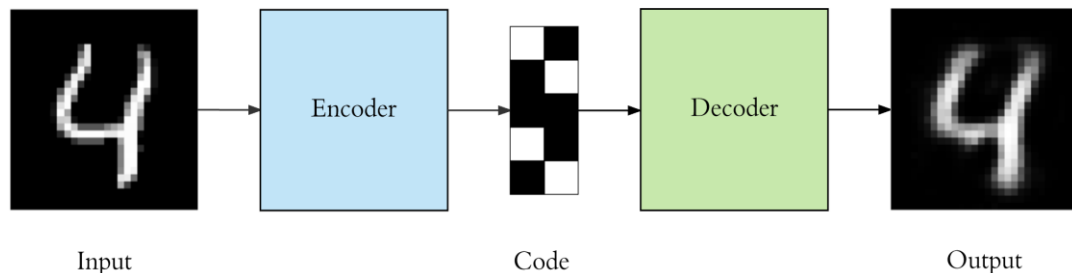
## CHAPTER 4

### AUTO-ENCODER ALGORITHM

#### 4.1 INTRODUCTION

Auto-encoders are a specific type of feed forward neural networks where the input is the same as the output. They compress the input into a lower-dimensional *code* and then reconstruct the output from this representation. The code is a compact “summary” or “compression” of the input, also called the latent-space representation.

An auto-encoder consists of 3 components: encoder, code and decoder. The encoder compresses the input and produces the code, the decoder then reconstructs the input only using this code.



**Fig : 4.1 Auto-encoder Process**

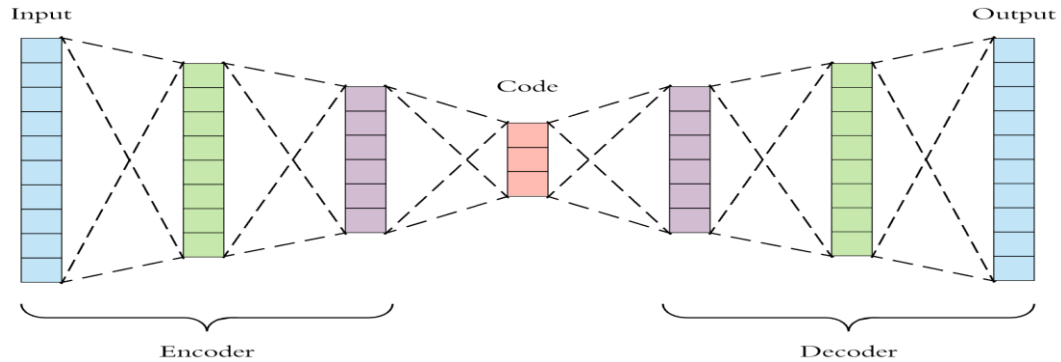
To build an auto-encoder we need 3 things: an encoding method, decoding method, and a loss function to compare the output with the target.

Auto-encoders are mainly a dimensionality reduction (or compression) algorithm with a couple of important properties:

- **Data-specific:** Auto-encoders are only able to meaningfully compress data similar to what they have been trained on. Since they learn features specific for the given training data, they are different than a standard data compression algorithm like gzip. So we can't expect an auto-encoder trained on handwritten digits to compress landscape photos.
- **Lossy:** The output of the auto-encoder will not be exactly the same as the input, it will be a close but degraded representation. If you want lossless compression they are not the way to go.
- **Unsupervised:** To train an auto-encoder we don't need to do anything fancy, just throw the raw input data at it. Auto-encoders are considered an unsupervised learning technique since they don't need explicit labels to train on. But to be more precise they are self-supervised because they generate their own labels from the training data.

## 4.2 ARCHITECTURE

Let's explore the details of the encoder, code and decoder. Both the encoder and decoder are fully-connected feed forward neural networks. Code is a single layer of an ANN with the dimensionality of our choice. The number of nodes in the code layer (code size) is a hyper-parameter that we set before training the auto-encoder.



**Fig : 4.2 Architecture**

This is a more detailed visualization of an auto-encoder. First the input passes through the encoder, which is a fully-connected ANN, to produce the code. The decoder, which has the similar ANN structure, then produces the output only using the code. The goal is to get an output identical with the input. Note that the decoder architecture is the mirror image of the encoder. This is not a requirement but it's typically the case. The only requirement is the dimensionality of the input and output needs to be the same. Anything in the middle can be played with.

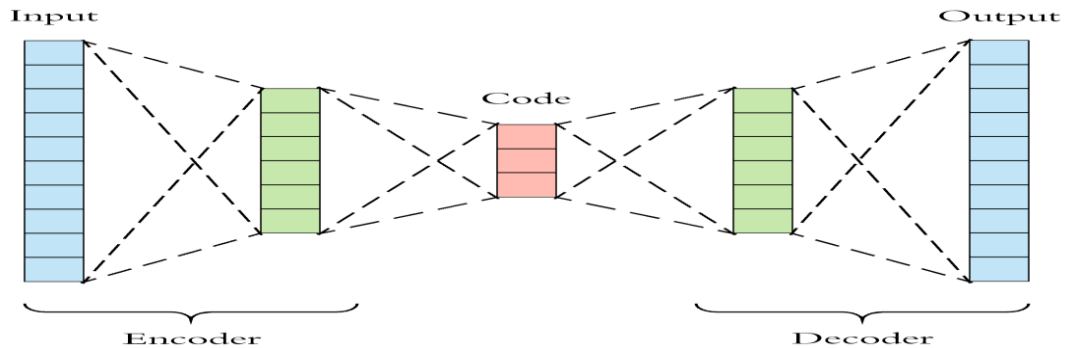
There are 4 hyper-parameters that we need to set before training an auto-encoder:

- **Code size:** number of nodes in the middle layer. Smaller size results in more compression.
- **Number of layers:** the auto-encoder can be as deep as we like. In the figure above we have 2 layers in both the encoder and decoder, without considering the input and output.
- **Number of nodes per layer:** the auto-encoder architecture we're working on is called a stacked auto-encoder since the layers are stacked one after another. Usually stacked auto-encoders look like a "sandwich". The number of nodes per layer decreases with each subsequent layer of the encoder, and increases back in the decoder. Also the decoder is symmetric to the encoder in terms of layer structure. As noted above this is not necessary and we have total control over these parameters.
- **Loss function:** we either use mean squared error (mse) or binary cross entropy. If the input values are in the range  $[0, 1]$  then we typically use cross entropy, otherwise we use the mean squared error.

Auto-encoders are trained the same way as ANNs via back propagation.

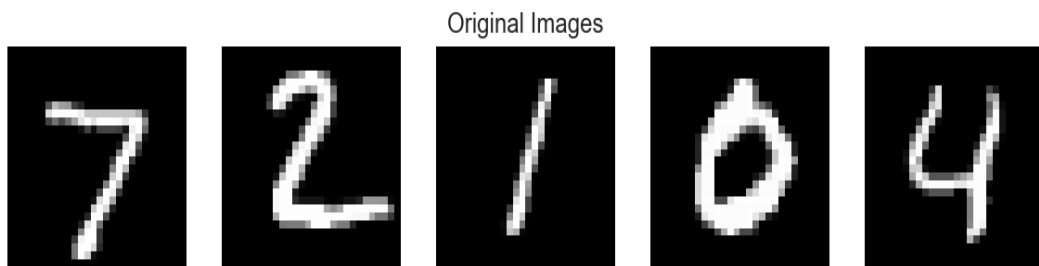
### 4.3 IMPLEMENTATION

Now let's implement an auto-encoder for the following architecture, 1 hidden layer in the encoder and decoder.



**Fig : 4.3 Implementation**

We will use the extremely popular MNIST dataset as input. It contains black-and-white images of handwritten digits.



**Fig : 4.4 MINST Dataset**

They're of size 28x28 and we use them as a vector of 784 numbers between [0, 1].

We will now implement the auto-encoder with Keras. The hyper-parameters are: 128 nodes in the hidden layer, code size is 32, and binary crossentropy is the loss function.

```
1 input_size = 784
1 hidden_size = 128
2 code_size = 32
3 input_img = Input(shape=(input_size,))
4 hidden_1 = Dense(hidden_size, activation='relu')(input_img)
5 code = Dense(code_size, activation='relu')(hidden_1)
6 hidden_2 = Dense(hidden_size, activation='relu')(code)
7 output_img = Dense(input_size, activation='sigmoid')(hidden_2)
8 autoencoder = Model(input_img, output_img)
9 autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
10 autoencoder.fit(x_train, x_train, epochs=5)
```



This is very similar to the ANNs we worked on, but now we're using the Keras functional API. Before we used to add layers using the sequential API as follows:

```
model.add(Dense(16, activation='relu'))  
model.add(Dense(8, activation='relu'))
```

With the functional API we do this:

```
layer_1 = Dense(16, activation='relu')(input)  
layer_2 = Dense(8, activation='relu')(layer_1)
```

It's more verbose but a more flexible way to define complex models. We can easily grab parts of our model, for example only the decoder, and work with that. The output of Dense method is a callable layer, using the functional API we provide it with the input and store the output. The output of a layer becomes the input of the next layer. With the sequential API the add method implicitly handled this for us.

Note that all the layers use the relu activation function, as it's the standard with deep neural networks. The last layer uses the sigmoid activation because we need the outputs to be between

[0, 1]. The input is also in the same range.

Also note the call to fit function, before with ANNs we used to do:

```
model.fit(x_train, y_train)
```

But now we do:

```
model.fit(x_train, x_train)
```

Remember that the targets of the auto-encoder are the same as the input. That's why we supply the training data as the target.

### 4.3.1 VISUALIZATION

Now let's visualize how well our auto-encoder reconstructs its input.

We run the auto-encoder on the test set simply by using the predict function of Keras. For every image in the test set, we get the output of the auto-encoder. We expect the output to be very similar to the input.

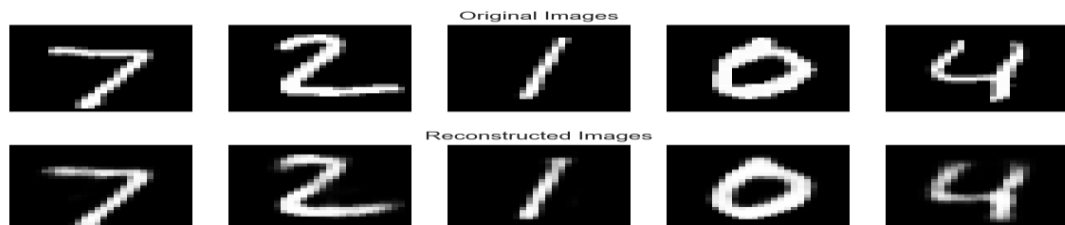


Fig : 4.5 Original & re-constructed images

They are indeed pretty similar, but not exactly the same. We can notice it more clearly in the last digit “4”. Since this was a simple task our auto-encoder performed pretty well.

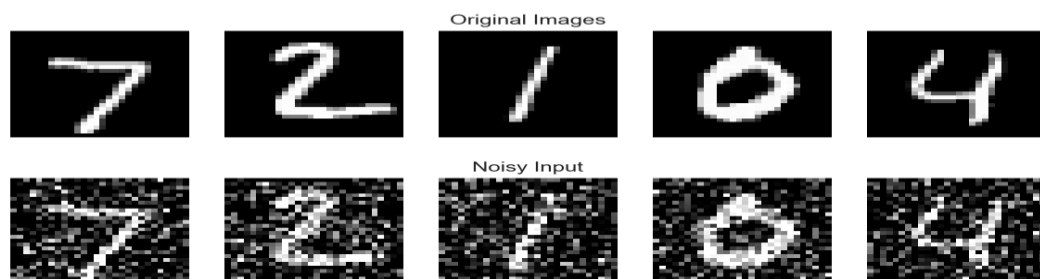
### 4.3.2 ADVICE

We have total control over the architecture of the auto-encoder. We can make it very powerful by increasing the number of layers, nodes per layer and most importantly the code size. Increasing these hyper-parameters will let the auto-encoder to learn more complex coding. But we should be careful to not make it too powerful. Otherwise the auto-encoder will simply learn to copy its inputs to the output, without learning any meaningful representation. It will just mimic the identity function. The auto-encoder will reconstruct the training data perfectly, but it will be over fitting without being able to generalize to new instances, which is not what we want.

This is why we prefer a “sandwich” architecture, and deliberately keep the code size small. Since the coding layer has a lower dimensionality than the input data, the auto-encoder is said to be under-complete. It won’t be able to directly copy its inputs to the output, and will be forced to learn intelligent features. If the input data has a pattern, for example the digit “1” usually contains a somewhat straight line and the digit “0” is circular, it will learn this fact and encode it in a more compact form. If the input data was completely random without any internal correlation or dependency, then an under-complete auto-encoder won’t be able to recover it perfectly. But luckily in the real-world there is a lot of dependency.

## 4.4. DE-NOISING AUTO-ENCODERS

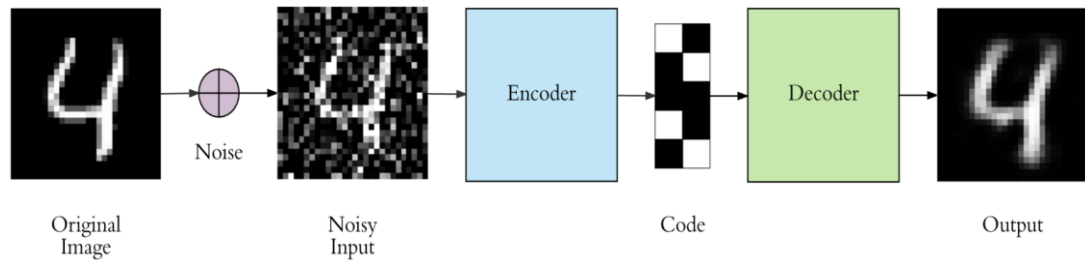
Keeping the code layer small forced our auto-encoder to learn an intelligent representation of the data. There is another way to force the auto-encoder to learn useful features, which is adding random noise to its inputs and making it recover the original noise-free data. This way the auto-encoder can’t simply copy the input to its output because the input also contains random noise. We are asking it to subtract the noise and produce the underlying meaningful data. This is called a de-noising auto-encoder.



**Fig : 4.6 De-Noising Auto-encoder**

The top row contains the original images. We add random Gaussian noise to them and the noisy data becomes the input to the auto-encoder. The auto-encoder doesn’t

see the original image at all. But then we expect the auto-encoder to regenerate the noise-free original image.



**Fig : 4.7 Process of De-noising**

There is only one small difference between the implementation of de-noising auto-encoder and the regular one. The architecture doesn't change at all, only the fit function.

We trained the regular auto-encoder as follows:

**autoencoder.fit(x\_train, x\_train)**

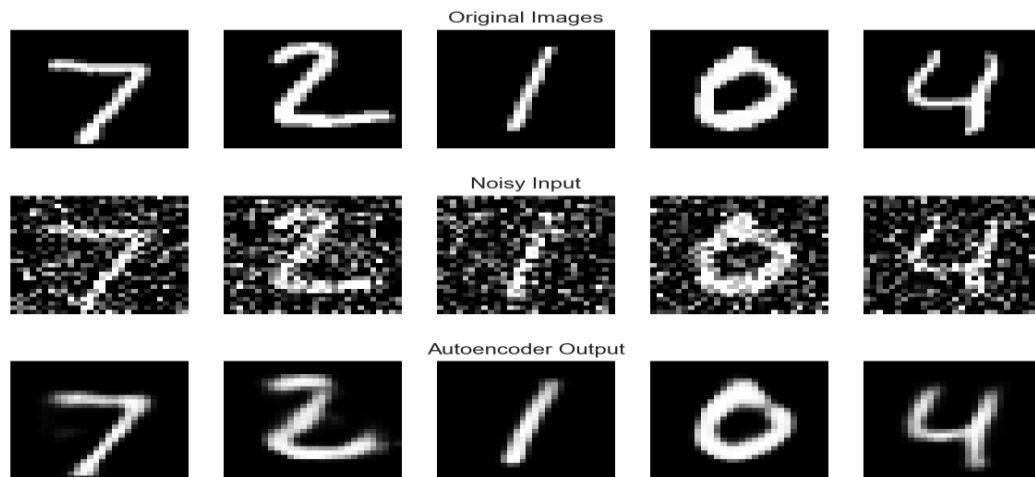
De-noising auto-encoder is trained as:

**autoencoder.fit(x\_train\_noisy, x\_train)**

Simple as that, everything else is exactly the same. The input to the auto-encoder is the noisy image, and the expected target is the original noise-free one.

#### 4.4.1 VISUALIZATION

Now let's visualize whether we are able to recover the noise-free images.



**Fig : 4.8 Process of Auto-encoding**

Looks pretty good. The bottom row is the auto-encoder output. We can do better by using more complex auto-encoder architecture, such as convolutional auto-encoders. We will cover convolutions in the upcoming article.

## 4.5 SPARSE AUTO-ENCODERS

We introduced two ways to force the auto-encoder to learn useful features:

- 1 Keeping the code size small and de-noising auto-encoders.
- 2 The next method is using regularization. We can regularize the auto-encoder by using a sparsity constraint such that only a fraction of the nodes would have non-zero values, called active nodes.

In particular, we add a penalty term to the loss function such that only a fraction of the nodes become active. This forces the auto-encoder to represent each input as a combination of small number of nodes, and demands it to discover interesting structure in the data. This method works even if the code size is large, since only a small subset of the nodes will be active at any time. It's pretty easy to do this in Keras with just one parameter. As a reminder, previously we created the code layer as follows:

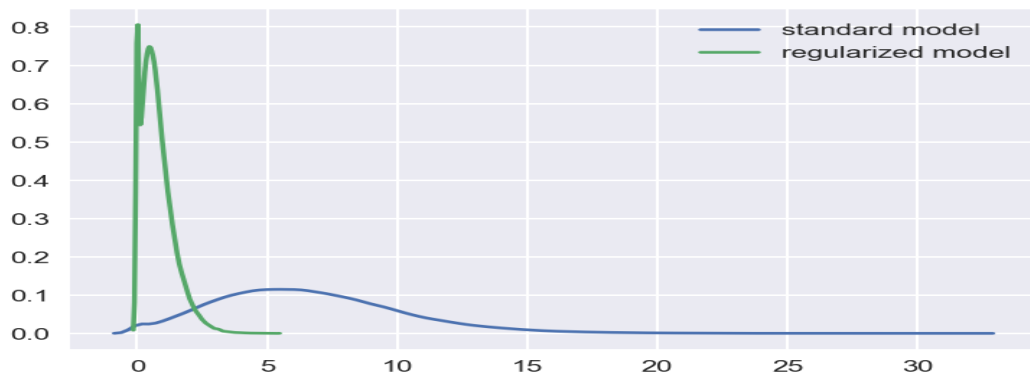
```
code = Dense(code_size, activation='relu')(input_img)
```

We now add another parameter called activity\_regularizer by specifying the regularization strength. This is typically a value in the range [0.001, 0.000001]. Here we chose 10e-6.

```
code = Dense(code_size, activation='relu', activity_regularizer=l1(10e-6))(input_img)
```

The final loss of the sparse model is 0.01 higher than the standard one, due to the added regularization term.

Let's demonstrate the encodings generated by the regularized model are indeed sparse. If we look at the histogram of code values for the images in the test set, the distribution is as follows:



**Fig : 4.9 Regularized model**

The mean for the standard model is 6.6 but for the regularized model it's 0.8, a pretty big reduction. We can see that a large chunk of code values in the regularized model are indeed 0, which is what we wanted. The variance of the regularized model is also fairly low.

## 4.6 USE CASES

Now we might ask the following questions. How good are auto-encoders at compressing the input? And are they a commonly used deep learning technique?

Unfortunately auto-encoders are not widely used in real-world applications. As a compression method, they don't perform better than its alternatives, for example jpeg does photo compression better than an auto-encoder. And the fact that auto-encoders are data-specific makes them impractical as a general technique. They have 3 common use cases though:

- 1 **Data de-noising:** we have seen an example of this on images.
- 2 **Dimensionality reduction:** visualizing high-dimensional data is challenging. t-SNE is the most commonly used method but struggles with large number of dimensions (typically above 32). So auto-encoders are used as a preprocessing step to reduce the dimensionality, and this compressed representation is used by t-SNE to visualize the data in 2D space.
- 3 **Variational Auto-encoders (VAE):** this is a more modern and complex use-case of auto-encoders and we will cover them in another article. But as a quick summary, VAE learns the parameters of the probability distribution modeling the input data, instead of learning an arbitrary function in the case of vanilla auto-encoders. By sampling points from this distribution we can also use the VAE as a generative model.

## CHAPTER 5

### PRINCIPAL COMPONENT ANALYSIS (PCA) ALGORITHM

#### 5.1 WHAT IS PCA?

Let's say that you want to predict what the gross domestic product (GDP) of the United States will be for 2017. You have lots of information available: the U.S. GDP for the first quarter of 2017, the U.S. GDP for the entirety of 2016, 2015, and so on. You have any publicly-available economic indicator, like the unemployment rate, inflation rate, and so on. You have U.S. Census data from 2010 estimating how many Americans work in each industry and American Community Survey data updating those estimates in between each census. You know how many members of the House and Senate belong to each political party. You could gather stock price data, the number of IPOs occurring in a year, and how many CEOs seem to be mounting a bid for public office. Despite being an overwhelming number of variables to consider, this just scratches the surface.

TL;DR — you have a lot of variables to consider.

If you've worked with a lot of variables before, you know this can present problems. Do you understand the relationships between each variable? Do you have so many variables that you are in danger of over-fitting your model to your data or that you might be violating assumptions of whichever modeling tactic you're using?

You might ask the question, "How do I take all of the variables I've collected and focus on only a few of them?" In technical terms, you want to "reduce the dimension of your feature space." By reducing the dimension of your feature space, you have fewer relationships between variables to consider and you are less likely to over-fit your model. (Note: This doesn't immediately mean that over-fitting, etc. are no longer concerns — but we're moving in the right direction!)

Somewhat unsurprisingly, reducing the dimension of the feature space is called "dimensionality reduction." There are many ways to achieve dimensionality reduction, but most of these techniques fall into one of two classes:

1. **Feature Elimination**
2. **Feature Extraction**

**Feature elimination:** Feature elimination is what it sounds like, we reduce the feature space by eliminating features. In the GDP example above, instead of considering every single variable, we might drop all variables except the three we think will best predict what the U.S.'s gross domestic product will look like. Advantages of feature elimination methods include simplicity and maintaining interpretability of your variables.

As a disadvantage, though, you gain no information from those variables you've dropped. If we only use last year's GDP, the proportion of the population in manufacturing jobs per the most recent American Community Survey numbers, and unemployment rate to predict this year's GDP, we're missing out on whatever the dropped variables could contribute to our model. By eliminating features, we've also entirely eliminated any benefits those dropped variables would bring.

**Feature extraction:** Feature extraction, however, doesn't run into this problem. Say we have ten independent variables. In feature extraction, we create ten "new" independent variables, where each "new" independent variable is a combination of each of the ten "old" independent variables. However, we create these new independent variables in a specific way and order these new variables by how well they predict our dependent variable.

You might say, "Where does the dimensionality reduction come into play?" Well, we keep as many of the new independent variables as we want, but we drop the "least important ones." Because we ordered the new variables by how well they predict our dependent variable, we know which variable is the most important and least important. But — and here's the kicker — because these new independent variables are combinations of our old ones, we're still keeping the most valuable parts of our old variables, even when we drop one or more of these "new" variables!

Principal component analysis is a technique for feature extraction — so it combines our input variables in a specific way, then we can drop the "least important" variables while still retaining the most valuable parts of all of the variables! As an added benefit, each of the "new" variables after PCA are all independent of one another. This is a benefit because the assumptions of a linear model require our independent variables to be independent of one another. If we decide to fit a linear regression model with these "new" variables (see "principal component regression" below), this assumption will necessarily be satisfied.

## 5.2 WHEN TO USE PCA?

- If we want to reduce the number of variables, but aren't able to identify variables to completely remove from consideration?
- If we want to ensure your variables are independent of one another?
- If we are comfortable making our independent variables less interpretable?

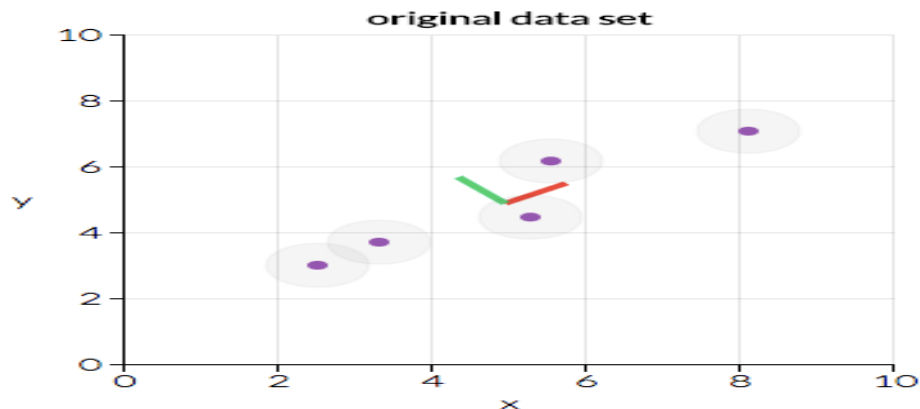
Then, PCA is a good method to use.

## 5.3 HOW DOES PCA WORK?

The section after this discusses how PCA works, but providing a brief summary before jumping into the algorithm may be helpful for context:

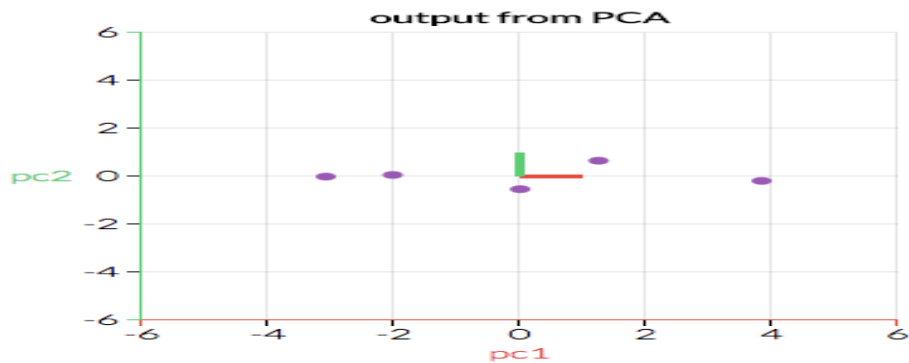
- We are going to calculate a matrix that summarizes how our variables all relate to one another.

- We'll then break this matrix down into two separate components: direction and magnitude. We can then understand the "directions" of our data and its "magnitude" displays the two main directions in this data: the "red direction" and the "green direction." In this case, the "red direction" is the more important one. We'll get into why this is the case later, but given how the dots are arranged, can you see why the "red direction" looks more important than the "green direction?" (Hint: What would fitting a line of best fit to this data look like?)



**Fig : 5.1 Original Dataset**

We will transform our original data to align with these important directions (which are combinations of our original variables). The screenshot below is the same exact data as above, but transformed so that the x-axes and y-axes are now the "red direction" and "green direction." What would the line of best fit look like here?



**Fig : 5.2 Output from PCA**

While the visual example here is two-dimensional (and thus we have two "directions"), think about a case where our data has more dimensions. By identifying which "directions" are most "important," we can compress or project our data into a smaller space by dropping the "directions" that are the "least important." **By projecting our data into a smaller space, we're reducing the dimensionality of our feature space... but because we've transformed our data**



**in these different “directions,” we’ve made sure to keep all original variables in our model!**

Here, I walk through an algorithm for conducting PCA. I try to avoid being too technical, but it’s impossible to ignore the details here, so my goal is to walk through things as explicitly as possible. A deeper intuition of *why* the algorithm works is presented in the next section.

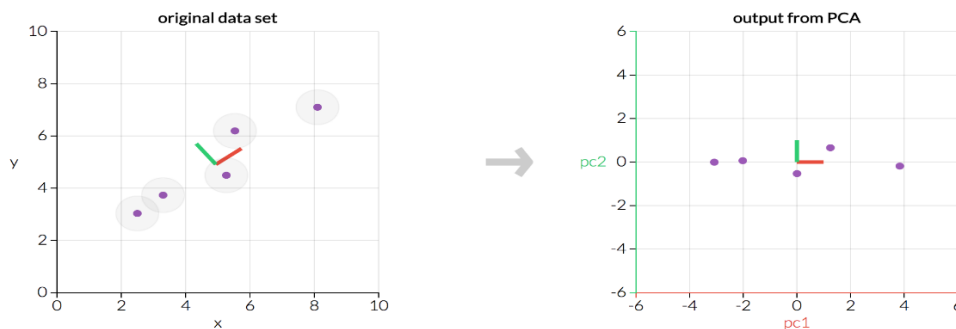
Before starting, you should have tabular data organized with  $n$  rows and likely  $p+1$  columns, where one column corresponds to your dependent variable (usually denoted  $Y$ ) and  $p$  columns where each corresponds to an independent variable (the matrix of which is usually denoted  $X$ ).

If a  $Y$  variable exists and is part of your data, then separate your data into  $Y$  and  $X$ , as defined above — we’ll mostly be working with  $X$ . (Note: if there exists no column for  $Y$ , that’s okay — skip to the next point!)

1. Take the matrix of independent variables  $X$  and, for each column, subtract the mean of that column from each entry. (This ensures that each column has a mean of zero.)
2. Decide whether or not to standardize. Given the columns of  $X$ , are features with higher variance more important than features with lower variance, or is the importance of features independent of the variance? (In this case, importance means how well that feature predicts  $Y$ .)
3. **If the importance of features is independent of the variance of the features, then divide each observation in a column by that column’s standard deviation.** (This, combined with step 2, standardizes each column of  $X$  to make sure each column has mean zero and standard deviation Call the centered (and possibly standardized) matrix  $Z$ .)
4. Take the matrix  $Z$ , [transpose it](#), and multiply the transposed matrix by  $Z$ . (Writing this out mathematically, we would write this as  $Z^T Z$ .) The resulting matrix is the [covariance matrix of  \$Z\$](#) , up to a constant.
5. Calculate the eigenvectors and their corresponding eigenvalues of  $Z^T Z$ . This is quite easily done in most computing packages— in fact, the [Eigen decomposition](#) of  $Z^T Z$  is where we decompose  $Z^T Z$  into  $PDP^{-1}$ , where  $P$  is the matrix of eigenvectors and  $D$  is the diagonal matrix with eigenvalues on the diagonal and values of zero everywhere else. The eigenvalues on the diagonal of  $D$  will be associated with the corresponding column in  $P$  — that is, the first element of  $D$  is  $\lambda_1$  and the corresponding eigenvector is the first column of  $P$ . This holds for all elements in  $D$  and their corresponding eigenvectors in  $P$ . We will always be able to calculate  $PDP^{-1}$  in this fashion. (Bonus: for those interested, we can always calculate  $PDP^{-1}$  in this fashion because  $Z^T Z$  is a [symmetric, positive semi-definite matrix](#).)
6. Take the eigenvalues  $\lambda_1, \lambda_2, \dots, \lambda_p$  and sort them from largest to smallest. In doing so, sort the eigenvectors in  $P$  accordingly. (For example, if  $\lambda_2$  is the largest eigenvalue, then take the second column of  $P$  and place it in the first column position.) Depending on the computing package, this may be done

automatically. Call this sorted matrix of eigenvectors  $P^*$ . (The columns of  $P^*$  should be the same as the columns of  $P$ , but perhaps in a different order.) **Note that these eigenvectors are independent of one another.**

7. Calculate  $Z^* = ZP^*$ . This new matrix,  $Z^*$ , is a centered/standardized version of  $X$  but now each observation is a combination of the original variables, where the weights are determined by the eigenvector. **As a bonus, because our eigenvectors in  $P^*$  are independent of one another, each column of  $Z^*$  is also independent of one another!**



**Fig : 5.3 Comparison of original dataset & output**

Note two things in this graphic:

The two charts show the exact same data, but the right graph reflects the original data transformed so that our axes are now the principal components.

In both graphs, the principal components are perpendicular to one another. **In fact, every principal component will ALWAYS be orthogonal** (a.k.a. official math term for perpendicular) **to every other principal component.**

**Because our principal components are orthogonal to one another, they are statistically linearly independent of one another... which is why our columns of  $Z^*$  are linearly independent of one another!**

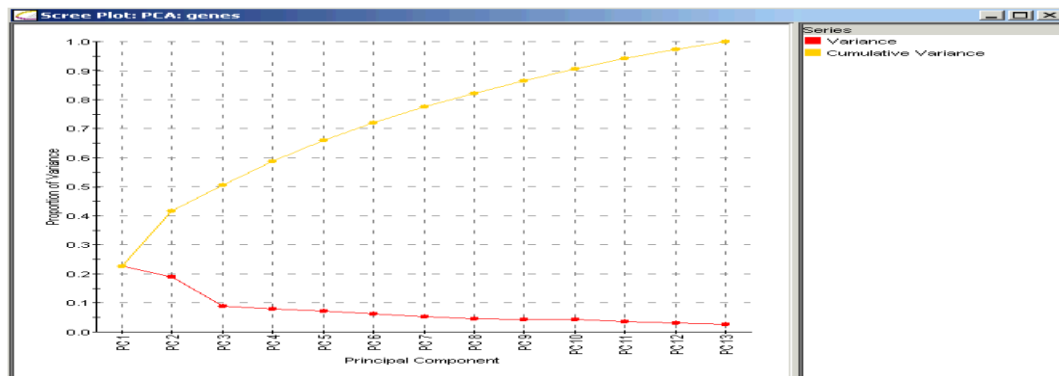
8. Finally, we need to determine how many features to keep versus how many to drop. There are three common methods to determine this, discussed below and followed by an explicit example:

**Method 1:** We arbitrarily select how many dimensions we want to keep. Perhaps I want to visually represent things in two dimensions, so I may only keep two features. This is use-case dependent and there isn't a hard-and-fast rule for how many features I should pick.

**Method 2:** Calculate the proportion of variance explained (briefly explained below) for each feature, pick a threshold, and add features until you hit that threshold. (For example, if you want to explain 80% of the total variability possibly explained by your model, add features with the largest explained proportion of variance until your proportion of variance explained hits or exceeds 80%.)

**Method 3:** This is closely related to Method 2. Calculate the proportion of variance explained for each feature, sort features by proportion of variance explained and plot the cumulative proportion of variance explained as you keep more features. (This plot is called a scree plot, shown below.) One can pick how many features to include by identifying the point where adding a new feature has a significant drop in variance explained relative to the previous feature, and choosing features up until that point.

Because each eigenvalue is roughly the importance of its corresponding eigenvector, the proportion of variance explained is the sum of the eigenvalues of the features you kept divided by the sum of the eigenvalues of all features.



**Fig : 5.4 PCA Plot**

Consider this scree plot for genetic data. The red line indicates the proportion of variance explained by each feature, which is calculated by taking that principal component's eigenvalue divided by the sum of all eigenvalues. The proportion of variance explained by including only principal component 1 is  $\lambda_1/(\lambda_1 + \lambda_2 + \dots + \lambda_p)$ , which is about 23%. The proportion of variance explained by including only principal component 2 is  $\lambda_2/(\lambda_1 + \lambda_2 + \dots + \lambda_p)$ , or about 19%.

The proportion of variance explained by including both principal components 1 and 2 is  $(\lambda_1 + \lambda_2)/(\lambda_1 + \lambda_2 + \dots + \lambda_p)$ , which is about 42%. This is where the yellow line comes in; the yellow line indicates the cumulative proportion of variance explained if you included all principal components up to that point. For example, the yellow dot above PC2 indicates that including principal components 1 and 2 will explain about 42% of the total variance in the model.

Now let's go through some examples:

**Method 1:** We arbitrarily select a number of principal components to include. Suppose I wanted to keep five principal components in my model. In the genetic data case above, these five principal components explains about 66% of the total variability that would be explained by including all 13 principal components.

**Method 2:** Suppose I wanted to include enough principal components to explain 90% of the total variability explained by all 13 principal components. In the genetic

data case above, I would include the first 10 principal components and drop the final three variables from  $\mathbf{Z}^*$ .

**Method 3:** Here, we want to “find the elbow.” In the scree plot above, we see there’s a big drop in proportion of variability explained between principal component 2 and principal component 3. In this case, we’d likely include the first two features and drop the remaining features. As you can see, this method is a bit subjective as “elbow” doesn’t have a mathematically precise definition and, in this case, we’d include a model that explains only about 42% of the total variability.

(Note: Some scree plots will have the size of eigenvectors on the Y axis rather than the proportion of variance. This leads to equivalent results, but requires the user to manually calculate the proportion of variance.)

**Once we’ve dropped the transformed variables we want to drop, we’re done! That’s PCA.**

## 5.4 WHY DOES PCA WORK?

While PCA is a very technical method relying on in-depth linear algebra algorithms, it’s a relatively intuitive method when you think about it.

- First, the covariance matrix  $\mathbf{Z}^T\mathbf{Z}$  is a matrix that contains estimates of how every variable in  $\mathbf{Z}$  relates to every other variable in  $\mathbf{Z}$ . Understanding how one variable is associated with another is quite powerful.
- Second, eigenvalues and eigenvectors are important. Eigenvectors represent directions. Think of plotting your data on a multidimensional scatter plot. Then one can think of an individual eigenvector as a particular “direction” in your scatterplot of data. Eigenvalues represent magnitude, or importance. Bigger eigenvalues correlate with more important directions.
- Finally, we make an assumption that more variability in a particular direction correlates with explaining the behavior of the dependent variable. Lots of variability usually indicates signal, whereas little variability usually indicates noise. Thus, the more variability there is in a particular direction is, theoretically, indicative of something important we want to detect.

Thus, PCA is a method that brings together:

1. A measure of how each variable is associated with one another. (Covariance matrix.)
2. The directions in which our data are dispersed. (Eigenvectors.)
3. The relative importance of these different directions. (Eigenvalues.)

PCA combines our predictors and allows us to drop the eigenvectors that are relatively unimportant.

## 5.5 ARE THERE EXTENSIONS TO PCA?

- Yes, more than I can address here in a reasonable amount of space. The one I've most frequently seen is principal component regression, where we take our untransformed  $Y$  and regress it on the subset of  $Z^*$  that we didn't drop. (This is where the independence of the columns of  $Z^*$  comes in; by regressing  $Y$  on  $Z^*$ , we know that the required independence of independent variables will necessarily be satisfied. However, we will need to still check our other assumptions.)
- The other commonly-seen variant I've seen is kernel PCA.

## CHAPTER 6

### TECHNOLOGIES USED IN THE PROJECT

#### 6.1 TENSORFLOW:

##### 6.1.1 WHAT IS TENSORFLOW?

Currently, the most famous deep learning library in the world is Google's TensorFlow. Google product uses machine learning in all of its products to improve the search engine, translation, image captioning or recommendations.

To give a concrete example, Google users can experience a faster and more refined the search with AI. If the user types a keyword on the search bar, Google provides a recommendation about what could be the next word.

Google wants to use machine learning to take advantage of their massive datasets to give users the best experience. Three different groups use machine learning:

- Researchers
- Data scientists
- Programmers.

They can all use the same toolset to collaborate with each other and improve their efficiency.

Google does not just have any data; they have the world's most massive computer, so Tensor Flow was built to scale. TensorFlow is a library developed by the Google Brain Team to accelerate machine learning and deep neural network research.

It was built to run on multiple CPUs or GPUs and even mobile operating systems, and it has several wrappers in several languages like Python, C++ or Java.

##### 6.1.2 HISTORY OF TENSORFLOW

A couple of years ago, deep learning started to outperform all other machine learning algorithms when giving a massive amount of data. Google saw it could use these deep neural networks to improve its services:

- Gmail
- Photo
- Google search engine

They build a framework called **Tensorflow** to let researchers and developers work together on an AI model. Once developed and scaled, it allows lots of people to use it.

### 6.1.3 TENSORFLOW ARCHITECTURE

Tensorflow architecture works in three parts:

- Preprocessing the data
- Build the model
- Train and estimate the model

It is called Tensorflow because it takes input as a multi-dimensional array, also known as **tensors**. You can construct a sort of **flowchart** of operations (called a Graph) that you want to perform on that input. The input goes in at one end, and then it flows through this system of multiple operations and comes out the other end as output.

This is why it is called TensorFlow because the tensor goes in it flows through a list of operations, and then it comes out the other side.

### 6.1.4 WHERE CAN TENSORFLOW RUN?

TensorFlow hardware, and software requirements can be classified into

**Development Phase:** This is when you train the mode. Training is usually done on your Desktop or laptop.

**Run Phase or Inference Phase:** Once training is done Tensorflow can be run on many different platforms. You can run it on:

1. Desktop running Windows, Mac-OS or Linux
2. Cloud as a web service
3. Mobile devices like IOS and Android

You can train it on multiple machines then you can run it on a different machine, once you have the trained model.

The model can be trained and used on GPUs as well as CPUs. GPUs were initially designed for video games. In late 2010, Stanford researchers found that GPU was also very good at matrix operations and algebra so that it makes them very fast for doing these kinds of calculations. Deep learning relies on a lot of matrix multiplication. TensorFlow is very fast at computing the matrix multiplication because it is written in C++. Although it is implemented in C++, TensorFlow can be accessed and controlled by other languages mainly, Python.

Finally, a significant feature of TensorFlow is the TensorBoard. The TensorBoard enables to monitor graphically and visually what TensorFlow is doing.

## 6.1.5 INTRODUCTION TO COMPONENTS OF TENSORFLOW

### 1. TENSOR

Tensorflow's name is directly derived from its core framework: **Tensor**. In Tensorflow, all the computations involve tensors. A tensor is a **vector** or **matrix** of n-dimensions that represents all types of data. All values in a tensor hold identical data type with a known (or partially known) **shape**. The shape of the data is the dimensionality of the matrix or array.

A tensor can be originated from the input data or the result of a computation. In TensorFlow, all the operations are conducted inside a **graph**. The graph is a set of computation that takes place successively. Each operation is called an **op node** and are connected to each other. The graph outlines the ops and connections between the nodes. However, it does not display the values.

### 2. GRAPHS

TensorFlow makes use of a graph framework. The graph gathers and describes all the series computations done during the training. The graph has lots of advantages:

- It was done to run on multiple CPUs or GPUs and even mobile operating system
- The portability of the graph allows to preserve the computations for immediate or later use. The graph can be saved to be executed in the future.
- All the computations in the graph are done by connecting tensors together
- A tensor has a node and an edge. The node carries the mathematical operation and produces an endpoints outputs. The edges the edges explain the input/output relationships between nodes.

## 6.1.6 WHY IS TENSORFLOW POPULAR?

- TensorFlow is the best library of all because it is built to be accessible for everyone.
- Tensorflow library incorporates different API to build at scale deep learning architecture like CNN or RNN.
- TensorFlow is based on graph computation; it allows the developer to visualize the construction of the neural network with Tensorboard.
- This tool is helpful to debug the program.
- Finally, Tensorflow is built to be deployed at scale.
- It runs on CPU and GPU.
- Tensorflow attracts the largest popularity on GitHub compare to the other deep learning framework.



## 6.1.7 LIST OF PROMINENT ALGORITHMS SUPPORTED BY TENSORFLOW

Currently, TensorFlow 1.10 has a built-in API for:

- Linear regression: `tf.estimator.LinearRegressor`
- Classification: `tf.estimator.LinearClassifier`
- Deep learning classification: `tf.estimator.DNNClassifier`
- Deep learning wide and deep: `tf.estimator.DNNLinearCombinedClassifier`
- Booster tree regression: `tf.estimator.BoostedTreesRegressor`
- Boosted tree classification: `tf.estimator.BoostedTreesClassifier`

## 6.1.8 SIMPLE TENSORFLOW EXAMPLE

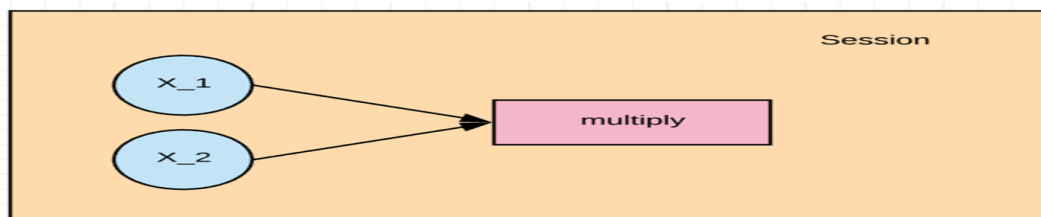
```
import numpy as np
```

```
import tensorflow as tf
```

In the first two line of code, we have imported tensorflow as `tf`. With Python, it is a common practice to use a short name for a library. The advantage is to avoid to type the full name of the library when we need to use it. For instance, we can import tensorflow as `tf`, and call `tf` when we want to use a tensorflow function

Let us practice the elementary workflow of Tensorflow with a simple example. Now let us create a computational graph that multiplies two numbers together.

During the example, we will multiply `X_1` and `X_2` together. Tensorflow will create a node to connect the operation. In our example, it is called `multiply`. When the graph is determined, Tensorflow computational engines will multiply together `X_1` and `X_2`.



**Fig : 6.1 Simple TensorFlow example**

Finally, we will run a TensorFlow session that will run the computational graph with the values of `X_1` and `X_2` and print the result of the multiplication.

Let's define the `X_1` and `X_2` input nodes. When we create a node in TensorFlow, we have to choose what kind of node to create. The `X1` and `X2` nodes will be a placeholder node. The placeholder assigns a new value each time we make a calculation. We will create them as a TF dot placeholder node.

### Step 1: Define the variable

```
X_1 = tf.placeholder(tf.float32, name = "X_1")
X_2 = tf.placeholder(tf.float32, name = "X_2")
```

When we create a placeholder node, we have to pass in the data type will be adding numbers here so we can use a floating-point data type, let's use `tf.float32`. We also need to give this node a name. This name will show up when we look at the graphical visualizations of our model. Let's name this node `X_1` by passing in a parameter called `name` with a value of `X_1` and now let's define `X_2` the same way. `X_2`.

### Step 2: Define the computation

```
multiply = tf.multiply(X_1, X_2, name = "multiply")
```

Now we can define the node that does the multiplication operation. In Tensorflow we can do that by creating a `tf.multiply` node.

We will pass in the `X_1` and `X_2` nodes to the multiplication node. It tells tensorflow to link those nodes in the computational graph, so we are asking it to pull the values from `x` and `y` and multiply the result. Let's also give the multiplication node the name `multiply`. It is the entire definition for our simple computational graph.

### Step 3: Execute the operation

To execute operations in the graph, we have to create a session. In Tensorflow, it is done by `tf.Session()`. Now that we have a session we can ask the session to run operations on our computational graph by calling `session.run()`. To run the computation, we need to use `run()`.

When the addition operation runs, it is going to see that it needs to grab the values of the `X_1` and `X_2` nodes, so we also need to feed in values for `X_1` and `X_2`. We can do that by supplying a parameter called `feed_dict`. We pass the value 1,2,3 for `X_1` and 4,5,6 for `X_2`.

We print the results with `print(result)`. We should see 4, 10 and 18 for 1x4, 2x5 and 3x6

```
X_1 = tf.placeholder(tf.float32, name = "X_1")
X_2 = tf.placeholder(tf.float32, name = "X_2")
multiply = tf.multiply(X_1, X_2, name = "multiply")
with tf.Session() as session:
    result = session.run(multiply, feed_dict={X_1:[1,2,3], X_2:[4,5,6]})
    print(result)
[ 4. 10. 18.]
```

## 6.1.9 SUMMARY

TensorFlow is the most famous deep learning library these recent years. A practitioner using TensorFlow can build any deep learning structure, like CNN, RNN or simple artificial neural network.

TensorFlow is mostly used by academics, startups, and large companies. Google uses TensorFlow in almost all Google daily products including Gmail, Photo and Google Search Engine.

Google Brain team's developed TensorFlow to fill the gap between researchers and products developers. In 2015, they made TensorFlow public; it is rapidly growing in popularity. Nowadays, TensorFlow is the deep learning library with the most repositories on GitHub.

Practitioners use Tensorflow because it is easy to deploy at scale. It is built to work in the cloud or on mobile devices like IOS and Android.

Tensorflow works in a session. Each session is defined by a graph with different computations. A simple example can be to multiply to number.

In Tensorflow, three steps are required:

### 1. Define the variable

```
X_1 = tf.placeholder(tf.float32, name = "X_1")
X_2 = tf.placeholder(tf.float32, name = "X_2")
```

### 2. Define the computation

```
multiply = tf.multiply(X_1, X_2, name = "multiply")
```

### 3. Execute the operation

```
with tf.Session() as session:
    result = session.run(multiply, feed_dict={X_1:[1,2,3], X_2:[4,5,6]})
    print(result)
```

One common practice in Tensorflow is to create a pipeline to load the data. If you follow these five steps, you'll be able to load data to TensorFlow

### 1. Create the data

```
import numpy as np
x_input = np.random.sample((1,2))
print(x_input)
```

### 2. Create the placeholder

```
x = tf.placeholder(tf.float32, shape=[1,2], name = 'X')
```

### 3. Define the dataset method

```
dataset = tf.data.Dataset.from_tensor_slices(x)
```

#### 4. Create the pipeline

```
iterator = dataset.make_initializable_iterator() get_next = iterator.get_next()
```

#### 5. Execute the program

```
with tf.Session() as sess:  
    sess.run(iterator.initializer, feed_dict={ x: x_input })  
    print(sess.run(get_next))
```

## 6.2 OPENCV

### 6.2.1 INTRODUCTION

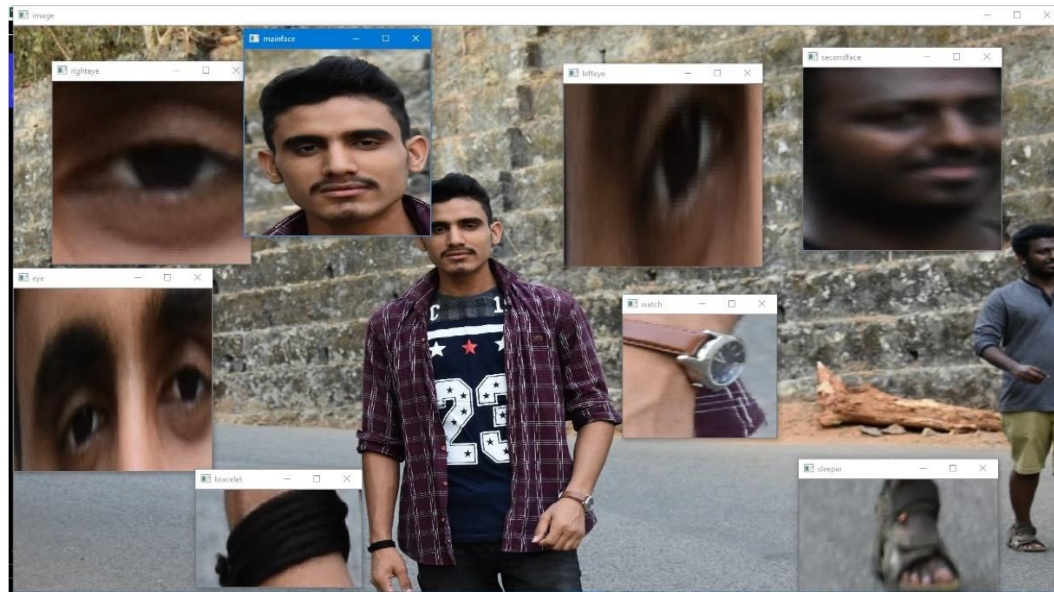
OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as Numpy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

The first OpenCV version was 1.0. OpenCV is released under a BSD license and hence it's free for both **academic** and **commercial** use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. When OpenCV was designed the main focus was real-time applications for computational efficiency. All things are written in optimized C/C++ to take advantage of multi-core processing.

Look at the following images



Fig : 6.2 OpenCV example 1



**Fig : 6.3 OpenCV example 2**

from the above original image, lots of pieces of information that are present in the original image can be obtained. Like in the above image there are two faces available and the person(I) in the images wearing a bracelet, watch and so on. So by the help of OpenCV we can get all these types of information from the original image.

It's the basic introduction to OpenCV we can continue the Applications and all the things in our upcoming articles.

## **6.2.2 HISTORY**

Officially launched in 1999 the OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications, part of a series of projects including real-time ray tracing and 3D display walls. The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.
- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
- Advance vision-based commercial applications by making portable, performance-optimized code available for free – with a license that did not require code to be open or free itself.

The first alpha version of OpenCV was released to the public at the IEEE Conference on Computer Vision and Pattern Recognition in 2000, and five betas

were released between 2001 and 2005. The first 1.0 version was released in 2006. A version 1.1 "pre-release" was released in October 2008.

The second major release of the OpenCV was in October 2009. OpenCV 2 includes major changes to the C++ interface, aiming at easier, more type-safe patterns, new functions, and better implementations for existing ones in terms of performance (especially on multi-core systems). Official releases now occur every six months and development is now done by an independent Russian team supported by commercial corporations.

In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site.

On May 2016, Intel signed an agreement to acquire Itseez, a leading developer of OpenCV.

**6.2.3 APPLICATIONS OF OPENCV:** There are lots of applications which are solved using OpenCV, some of them are listed below:

- face recognition
- Automated inspection and surveillance
- number of people – count (foot traffic in a mall, etc)
- Vehicle counting on highways along with their speeds
- Interactive art installations
- Anamoly (defect) detection in the manufacturing process (the odd defective products)
- Street view image stitching
- Video/image search and retrieval
- Robot and driver-less car navigation and control
- object recognition
- Medical image analysis
- Movies – 3D structure from motion
- TV Channels advertisement recognition

#### **6.2.4 OPENCV FUNCTIONALITY**

- Image/video I/O, processing, display (core, imgproc, highgui)
- Object/feature detection (objdetect, features2d, nonfree)
- Geometry-based monocular or stereo computer vision (calib3d, stitching, videostab)
- Computational photography (photo, video, superres)
- Machine learning & clustering (ml, flann)
- CUDA acceleration (gpu)

## 6.3 DJANGO

### 6.3.1 INTRODUCTION

Django is a web application framework written in Python programming language. It is based on MVT (Model View Template) design pattern. The Django is very demanding due to its rapid development feature. It takes less time to build application after collecting client requirement.

This framework uses a famous tag line - **The web framework for perfectionists with deadlines.**

By using Django, we can build web applications in very less time. Django is designed in such a manner that it handles much of configure things automatically, so we can focus on application development only.

### 6.3.2 HISTORY

Django was design and developed by Lawrence journal world in 2003 and publicly released under BSD license in July 2005. Currently, DSF (Django Software Foundation) maintains its development and release cycle.

Django was released on 21, July 2005. Its current stable version is 2.0.3 which was released on 6 March, 2018.

### 6.3.2.1 DJANGO VERSION HISTORY

Version	Date	Description
0.90	16 Nov 2005	
0.91	11 Jan 2006	magic removal
0.96	23 Mar 2007	newforms, testing tools
1.0	3 Sep 2008	API stability, decoupled admin, Unicode
1.1	29 Jul 2009	Aggregates, transaction based tests
1.2	17 May 2010	Multiple db connections, CSRF, model validation
1.3	23 Mar 2011	Timezones, in browser testing, app templates.
1.5	26 Feb 2013	Python 3 Support, configurable user model
1.6	6 Nov 2013	Dedicated to Malcolm Tredinnick, db transaction management, connection pooling.
1.7	2 Sep 2014	Migrations, application loading and configuration.
1.8 LTS	2 Sep 2014	Migrations, application loading and configuration.
1.8 LTS	1 Apr 2015	Native support for multiple template engines. <i>Supported until at least April 2018</i>
1.9	1 Dec 2015	Automatic password validation. New styling for admin interface.
1.10	1 Aug 2016	Full text search for PostgreSQL. New-style middleware.
1.11 LTS	1.11 LTS	Last version to support Python 2.7. <i>Supported until at least April 2020</i>
2.0	Dec 2017	First Python 3-only release, Simplified URL routing syntax, Mobile friendly admin.



### **6.3.3 POPULARITY**

Django is widely accepted and used by various well-known sites such as:

- Instagram
- Mozilla
- Disqus
- Pinterest
- Bitbucket
- The Washington Times

### **6.3.4 FEATURES OF DJANGO**

- Rapid Development
- Secure
- Scalable
- Fully loaded
- Versatile
- Open Source
- Vast and Supported Community

#### **Rapid Development**

Django was designed with the intention to make a framework which takes less time to build web application. The project implementation phase is a very time taken but Django creates it rapidly.

#### **Secure**

Django takes security seriously and helps developers to avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery etc. Its user authentication system provides a secure way to manage user accounts and passwords.

#### **Scalable**

Django is scalable in nature and has ability to quickly and flexibly switch from small to large scale application project.

#### **Fully loaded**

Django includes various helping task modules and libraries which can be used to handle common Web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds etc.

#### **Versatile**

Django is versatile in nature which allows it to build applications for different-different domains. Now a days, Companies are using Django to build various types

of applications like: content management systems, social networks sites or scientific computing platforms etc.

## Open Source

Django is an open source web application framework. It is publicly available without cost. It can be downloaded with source code from the public repository. Open source reduces the total cost of the application development.

## Vast and Supported Community

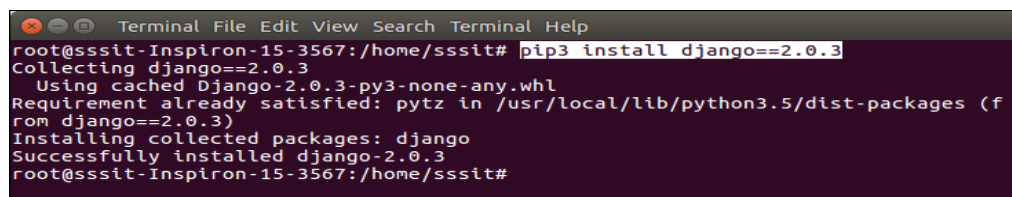
Django is an one of the most popular web framework. It has widely supportive community and channels to share and connect.

## 6.3.5 DJANGO INSTALLATION

- To install Django, first visit to **django official site** (<https://www.djangoproject.com>) and download django by clicking on the download section. Here, we will see various options to download The Django.
- Django requires **pip** to start installation. Pip is a package manager system which is used to install and manage packages written in python. For Python 3.4 and higher versions **pip3** is used to manage packages.
- we are installing Django in Ubuntu operating system.
- The complete installation process is described below. Before installing make sure **pip** is installed in local system.

Here, we are installing Django using pip3, the installation command is given below.

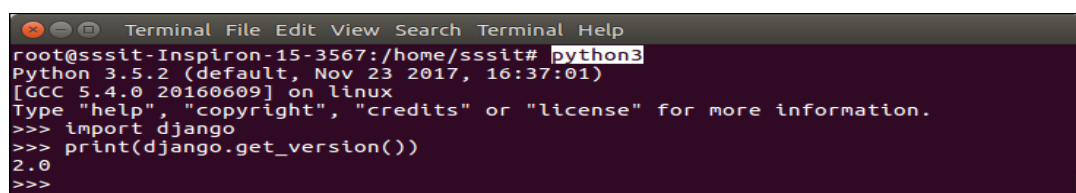
- **\$ pip3 install django==2.0.3**



```
Terminal File Edit View Search Terminal Help
root@sssit-Inspiron-15-3567:/home/sssit# pip3 install django==2.0.3
Collecting django==2.0.3
  Using cached Django-2.0.3-py3-none-any.whl
Requirement already satisfied: pytz in /usr/local/lib/python3.5/dist-packages (from django==2.0.3)
Installing collected packages: django
Successfully installed django-2.0.3
root@sssit-Inspiron-15-3567:/home/sssit#
```

- **Verify Django Installation**

After installing Django, we need to verify the installation. Open terminal and write **python3** and press enter. It will display python shell where we can verify django installation.

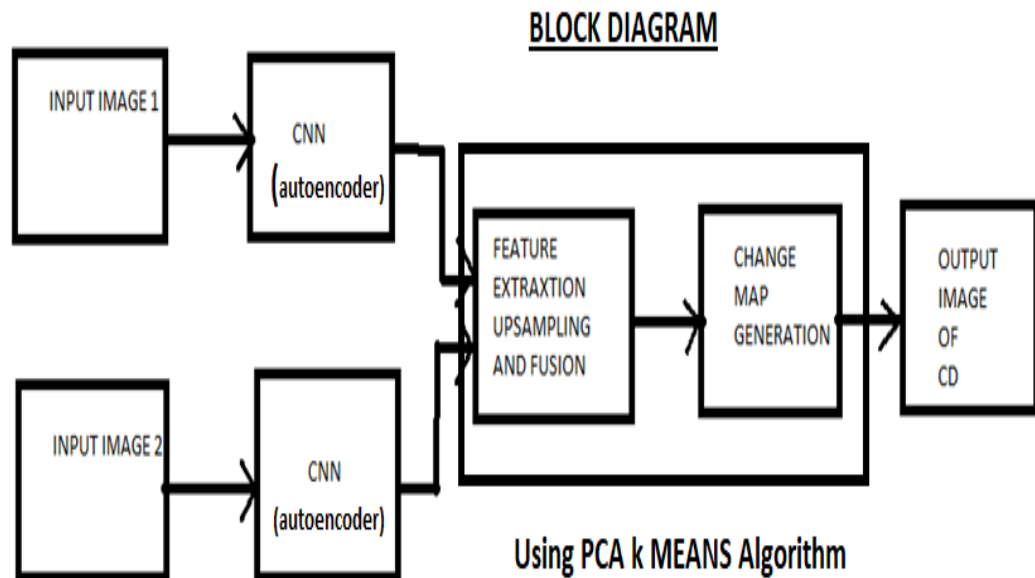


```
Terminal File Edit View Search Terminal Help
root@sssit-Inspiron-15-3567:/home/sssit# python3
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> print(django.get_version())
2.0
>>>
```

## CHAPTER 7

### PROCESS INVOLVED IN THE PROJECT

#### 7.1 BLOCK DIAGRAM



#### 7.2 PROCESS INVOLVED

The process involved is as follows:

##### INPUT IMAGES:

The input images basically provided as input are the satellite pictures of the same location

Co-ordinate taken at different timeframe ranging from months to years.

The input images are given to the auto-encoder for the images enhancement.

##### AUTOENCODERS:

Auto-encoders are a specific type of feed-forward neural networks where the input is the same as the output. They compress the input into a lower-dimensional code and then reconstruct the output from this representation. The code is a compact “summary” or “compression” of the input, also called the latent-space representation.

The process is basically carried out in order to the remove the noise that gets encrypted in the satellite images that can result in the false alarms.

The auto-encoder is a Convolutional Neural Network. The output of the auto-encoder is given to the PCA K-means algorithm.

## **FEATURE EXTRACTION,UP-SAMPLING, FUSION AND CHANGE MAP GENERATION USING PCA K MEANS ALGORITHM:**

Automatic change detection in images of a region acquired at different times is one of the most interesting topics of image processing. Such images are known as multi temporal images. Change detection involves the analysis of two multi temporal satellite images to find any changes that might have occurred between the two time stamps. It is one of the major utilization of remote sensing and finds application in a wide range of tasks like defence inspections, deforestation assessment, land use analysis, disaster assessment and monitoring many other environmental/man-made changes.

We will be outlining an unsupervised method for change detection in this blog post. It involves the automatic analysis of the change data, i.e. the difference image, constructed using the multi temporal images. A difference image is the pixel-by-pixel subtraction of the 2 images. Eigen vectors of pixel blocks from the difference image will then be extracted by Principal Component Analysis (PCA). Subsequently, a feature vector is constructed for each pixel in the difference image by projecting that pixel's neighbourhood onto the Eigen vectors. The feature vector space, which is the collection of the feature vectors for all the pixels, upon clustering by K-means algorithm gives us two clusters – one representing pixels belonging to the changed class, and other representing pixels belonging to the unchanged class. Each pixel will belong to either of the clusters and hence a change map can be generated. So, the steps towards implementing this application are:

- 1) Difference image generation and Eigen vector space (EVS)
- 2) Building the feature vector space (FVS)
- 3) Clustering of the feature vector space and change map

### **OUTPUT**

Output the segmented image with the white areas representing change areas and black region representing no change area. Thus the change detection of satellite images has been performed.

## 7.3 CODE

### #AUTOENCODER (CNN) CODE FOR IMAGE ENHANCEMENT

```
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D

from keras.models import Model

from keras.datasets import mnist

from keras import backend as K

import numpy as np

import os

import cv2

x_train = []

for i in os.listdir("train images"):

    img = cv2.imread("train images/"+i)

    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    img = cv2.resize(img, (500, 500))

    x_train.append(img)

x_train = np.array(x_train)

input_img = Input(shape=(500, 500, 3)) # adapt this if using `channels_first` image data
format

x = Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)

x = MaxPooling2D((2, 2), padding='same')(x)

x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)

x = MaxPooling2D((2, 2), padding='same')(x)

x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)

encoded = MaxPooling2D((2, 2), padding='same')(x)

y = Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)
```

```

y = UpSampling2D((2, 2))(y)

y = Conv2D(8, (3, 3), activation='relu', padding='same')(y)

y = UpSampling2D((2, 2))(y)

y = Conv2D(16, (3, 3), activation='relu')(y)

y = UpSampling2D((2, 2))(y)

decoded = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(y)


autoencoder = Model(input_img, decoded)

autoencoder.compile(optimizer='adadelta', loss='sparse_categorical')

encoder = Model(input_img, encoded)

autoencoder.fit(x_train, x_train, epochs=100)

model_data = autoencoder.to_json()

fl = open("Model.JSON", "w")

fl.write(model_data)

fl.close()

autoencoder.save("Model.h5")

```

### **#PCA K MEANS ALGORITHM FOR THE CHANGE DETECTION APPPROXIMATION FROM THE ENHANCED IMAGES**

```

from tensorflow.keras.models import model_from_json

import os

json_file = open('uploadimages\\Model.JSON', 'r')

loaded_model_json = json_file.read()

json_file.close()

model = model_from_json(loaded_model_json)

model.load_weights("uploadimages\\Model.h5")

import cv2

```

```

import numpy as np

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

from collections import Counter

from scipy.misc import imread, imresize, imsave


def find_vector_set(diff_image, new_size):

    i = 0

    j = 0

    vector_set = np.zeros((int(new_size[0] * new_size[1] / 25), 25))

    print('\nvector_set shape',vector_set.shape)

    while i < vector_set.shape[0]:

        while j < new_size[0]:

            k = 0

            while k < new_size[1]:

                block = diff_image[j:j+5, k:k+5]

                #print(i,j,k,block.shape)

                feature = block.ravel()

                vector_set[i, :] = feature

                k = k + 5

            j = j + 5

        i = i + 1

```

```

mean_vec = np.mean(vector_set, axis = 0)

vector_set = vector_set - mean_vec


return vector_set, mean_vec


def find_FVS(EVS, diff_image, mean_vec, new):

    i = 2

    feature_vector_set = []

    while i < new[0] - 2:

        j = 2

        while j < new[1] - 2:

            block = diff_image[i-2:i+3, j-2:j+3]

            feature = block.flatten()

            feature_vector_set.append(feature)

            j = j+1

        i = i+1


    FVS = np.dot(feature_vector_set, EVS)

    FVS = FVS - mean_vec

    print("\nfeature vector space size",FVS.shape)

    return FVS

```



```

def clustering(FVS, components, new):

    kmeans = KMeans(components, verbose = 0)

    kmeans.fit(FVS)

    output = kmeans.predict(FVS)

    count = Counter(output)

    least_index = min(count, key = count.get)

    print(new[0],new[1])

    change_map = np.reshape(output,(new[0] - 4, new[1] - 4))

    return least_index, change_map

```

```

def find_PCAKmeans(imagepath1, imagepath2):

    print('Operating')

    image1 = cv2.imread(imagepath1,0)

    image2 = cv2.imread(imagepath2,0)

    print(image1.shape,image2.shape)

    new_size = np.asarray(image1.shape) / 5

    new_size = new_size.astype(int) * 5

    image1 = imresize(image1, (new_size)).astype(np.int16)

    image2 = imresize(image2, (new_size)).astype(np.int16)

    diff_image = abs(image1 - image2)

```

```

imsave('diff.jpg', diff_image)

print('\nBoth images resized to ',new_size)

vector_set, mean_vec = find_vector_set(diff_image, new_size)

pca = PCA()
pca.fit(vector_set)
EVS = pca.components_

FVS = find_FVS(EVS, diff_image, mean_vec, new_size)

print('\ncomputing k means')

components = 3
least_index, change_map = clustering(FVS, components, new_size)

change_map[change_map == least_index] = 255
change_map[change_map != 255] = 0

change_map = change_map.astype(np.uint8)

kernel = np.asarray(((0,0,1,0,0),
                      (0,1,1,1,0),
                      (1,1,1,1,1),
                      (0,1,1,1,0),
                      (0,0,1,0,0)), dtype=np.uint8)

cleanChangeMap = cv2.erode(change_map,kernel)

```

```

imsave("cm.jpg", change_map)

imsave("ccm.jpg", cleanChangeMap)


if __name__ == "__main__":
    a = input("enter the first image: ")
    b = input("enter the second image: ")
    a = cv2.cvtColor(cv2.imread(a), cv2.COLOR_BGR2GRAY)
    b = cv2.cvtColor(cv2.imread(b), cv2.COLOR_BGR2GRAY)
    imsave("forest_1986.jpg",a)
    imsave("forest_1992.jpg",b)
    find_PCAKmeans(a,b)

```

### **#DJANGO CODE FOR WEB-DEVELOPMENT (OPTIONAL)**

```

from django.shortcuts import render, redirect
from .forms import UploadImageForm
from .fun_image import image_op
from .trainedmodel import difference
from PIL import Image
import cv2
import numpy as np
from django.core.files.storage import default_storage
from django.core.files.base import ContentFile

# Create your views here.


def homepage(request):

```

```

if request.method == 'POST':
    form = UploadImageForm(request.POST, files=request.FILES)

    if form.is_valid():
        print(type(difference))
        print(request.FILES['image1'])
        img1 = request.FILES['image1']
        img2 = request.FILES['image2']
        with open('img1.jpg', 'wb') as f:
            f.write(img1.read())
        with open('img2.jpg', 'wb') as f:
            f.write(img2.read())

        # path = default_storage.save('img1.jpg', ContentFile(img1.read()))
        a = difference()
        return redirect(f'link/{str(a)}')
    else:
        form = UploadImageForm()
        return render(request, 'upload.html', {'form':form})

def show_text(request, backend):

    return render(request, 'show_data.html', {"show": backend})

```

## CHAPTER 8

### OUTPUTS & RESULTS

#### RESULT 1:

Image 1



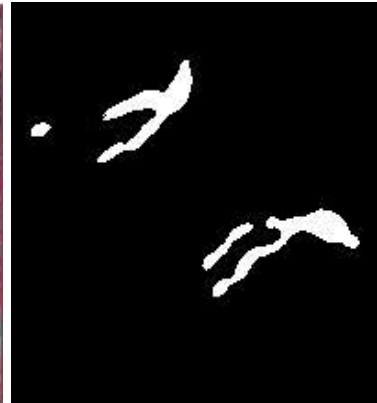
forest\_1986.png

image 2



forest\_1992.png

output



forest.jpg

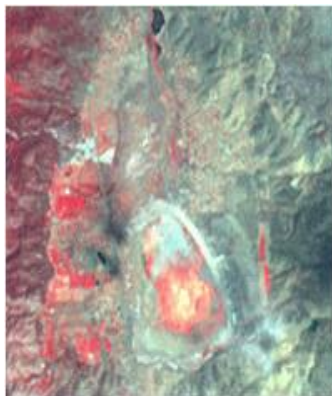
#### RESULT 2:

Image 1



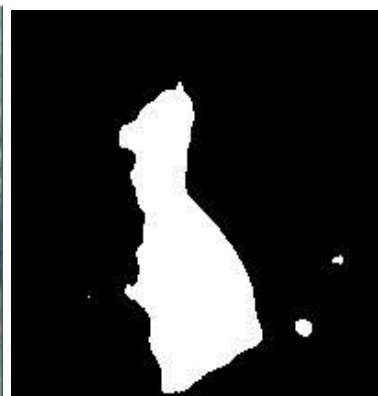
lake\_1986.png

image 2



lake\_1992.png

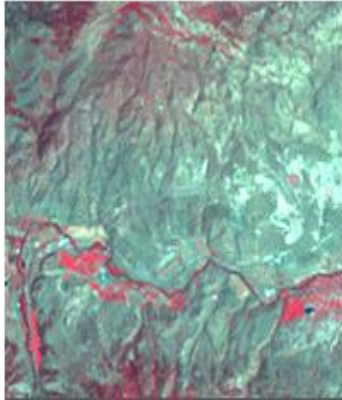
output



lake.jpg

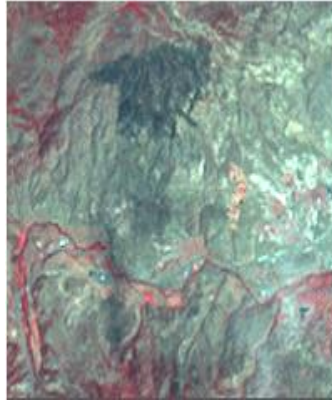
### RESULT 3:

**Image 1**



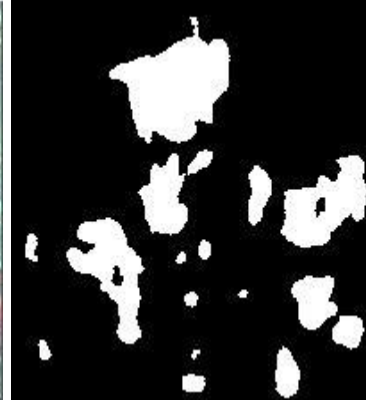
**burn\_1986.png**

**image 2**



**burn\_1992.png**

**output**



**burn.jpg**

### RESULT 4:

**Image 1**



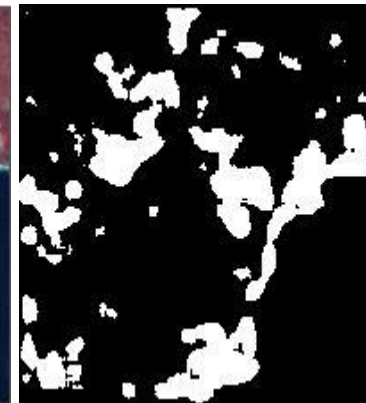
**conifer\_1986.png**

**image 2**



**conifer\_1992.png**

**output**



**conifer.jpg**

## RESULT 5:

**Image 1**



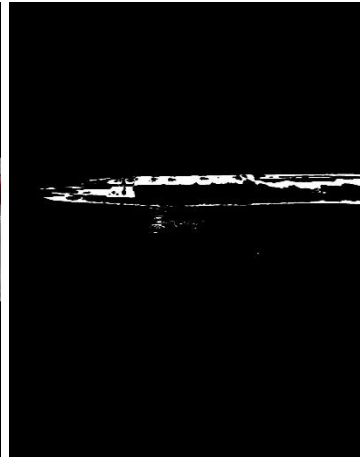
**pen\_bef.png**

**image 2**



**pen\_aft.png**

**output**



**pen.jpg**

## RESULT 6:

**Image 1**



**team\_bef.png**

**image 2**



**team\_aft.png**

**output**



**team.jpg**

## ACCURACY:

The accuracy for above images are as follows:

Image Name	Image Number	Accuracy (in %)
Forest	Result 1	82
Lake	Result 2	84
Burn	Result 3	85.36
Conifer	Result 4	87.74
Pen	Result 5	88.3
Team	Result 6	91.07



## **CHAPTER 9**

### **FUTURE SCOPE & APPLICATIONS**

In future studies, we plan to investigate more sophisticated strategies to high-level spatial information and shape based features encoding process to improve the invariance of representations. would also like to use state-of-the-art deeper CNNs like VGG-19 [9] and ResNet [16], and fine tune these networks on more specific dataset for HRRS images like UC Merced Land Use Dataset [17] and WHU-RS Dataset [18].

### **APPLICATIONS**

1. Medical image diagnosis
2. Military surveillance
3. Agriculture land monitoring
4. Astronomical phenomenon
5. Civilization density

## CHAPTER 10

### CONCLUSION& REFERENCES

In this paper, a novel CD is presented, based on CNN hyper features of two registered images, covering the same area took at different times, t1 and t2. Experiments has demonstrated the effectiveness of the CNN features on CD task. By fusing the information from several layers, the CD performances are greatly enhanced. The technique has a drawback of requirement of more computational time and the necessity of the registration step.

### REFERENCES

1. [1] Coppin, P.R., Bauer, M.E., 1996. Digital change detection in forest ecosystems with remote sensing imagery. *Remote Sensing Reviews* 13, 207–234.
2. [2] Celik, T. Unsupervised change detection in satellite images using principal component analysis and k-means clustering. *IEEE Geosci. Remote Sens. Lett.* 2009, 6, 772-776.
3. [3] Johnson, R.D., Kasischke, E.S., 1998. Change vector analysis: a technique for the multispectral monitoring of land cover and condition. *International Journal of Remote Sensing* 19, 411–426.
4. [4] Bruzzone, Z.; Prieto, D.F. Automatic analysis of the difference image for unsupervised change detection. *IEEE Trans. Geosci. Remote Sens.* 2000, 38, 1171-1182.
5. [5] Chen, C. Huo, Z. Zhou and H. Lu, "Unsupervised Change Detection in SAR Image using Graph Cuts," *Geoscience and Remote Sensing Symposium*, 2008. IGARSS 2008. IEEE International, Boston, MA, 2008, pp. III - 1162-III - 1165.
6. [6] Bovolo, F. A multilevel parcel-based approach to change detection in very high resolution multitemporal images. *IEEE Geosci. Remote Sens. Lett.* 2009, 6, 33-37.

7. [7] Kasetkasem and P. K. Varshney, "An image change detection algorithm based on Markov random field models," in *IEEE Transactions on Geoscience and Remote Sensing*, vol. 40, no. 8, pp. 1815-1823, Aug 2002.
8. [8] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.
9. [9] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
10. [10] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, pages 818–833. Springer, 2014.
11. [11] Penatti, O.A.; Nogueira, K.; dos Santos, J.A. Do Deep Features Generalize from Everyday Objects to Remote Sensing and Aerial Scenes Domains? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, Boston, MA, USA, 12 June 2015; pp. 44–51.
12. [12] Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; Darrell, T. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the ACM International Conference on Multimedia*, Orlando, FL, USA, 3–7 November 2014. 29.
13. [13] Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.* 2015, doi: 10.1007/s11263-015-0816-y.
14. [14] B. Hariharan, P. Arbelaez, R. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. In *CVPR*, 2015.
15. [15] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62-66, Jan. 1979.
16. [16] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385v1*.
17. [17] Yang, Y.; Newsam, S. Bag-of-visual-words and spatial extensions for land-use classification. In *Proceedings of the 18th SIGSPATIAL*

International Conference on Advances in Geographic Information Systems, San Jose, CA, USA, 2–5 November 2010; pp. 270–279.

18. [18] Xia, G.S.; Yang, W.; Delon, J.; Gousseau, Y.; Sun, H.; Maitre, H. Structural High-Resolution Satellite Image Indexing. In Processings of the ISPRS, TC VII Symposium Part A: 100 Years ISPRS—Advancing Remote Sensing Science, Vienna, Austria, 5–7 July 2010.

19. <https://opencv.org/>

20. <https://docs.python.org/3/tutorial/>