

DEEP LEARNING

(Identification of ships in satellite imagery)

*Summer Internship Report Submitted in partial fulfilment
Of the requirement for undergraduate degree of*

Bachelor of Technology

In

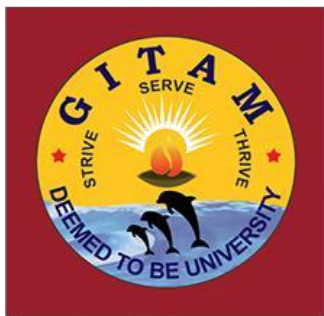
Computer Science and Engineering

By

A.N. Sathvik

221710308001

*Under the Guidance of
Assistant Professor*



GITAM
(DEEMED TO BE UNIVERSITY)
(Estd. u/s 3 of the UGC Act, 1956)

VISAKHAPATNAM ✶ HYDERABAD ✶ BENGALURU

Department of Computer Science and Engineering

GITAM School of Technology

GITAM (Deemed to be University)

Hyderabad-502329

July 2020

DECLARATION

I submit this deep learning project entitled “**Identification of ships in satellite imagery**” to GITAM (Deemed to Be University), Hyderabad in partial fulfilment of the requirements for the award of the degree of “**Bachelor of Technology**” in “**Computer Science and Engineering**”. I declare that it was carried out independently by me under the guidance of, Asst. Professor, GITAM (Deemed to Be University), Hyderabad, India.

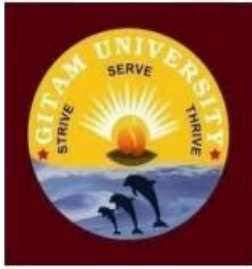
The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Place: -Hyderabad

A.N. Sathvik

Date: -

221710308001



GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated: **12.07.2020**

CERTIFICATE

This is to certify that the Deep Learning Report entitled “**Identification of ships in a satellite imagery**” is being submitted by A.N.Sathvik(221710308001) in partial fulfilment of the requirement for the award of Bachelor of Technology in **Computer Science and Engineering** at GITAM (Deemed To Be University), Hyderabad during the academic year 2020-21.

It is faithful record work carried out by him at the **Computer Science and Engineering Department**, GITAM University Hyderabad Campus under my guidance and supervision.

Assistant Professor
Department of CSE

Professor and HOD
Department of CSE

ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful completion of this internship.

I would like to thank respected **Dr. N. Siva Prasad**, Pro Vice Chancellor, GITAM Hyderabad and **Dr. CH. Sanjay**, Principal, GITAM Hyderabad.

I would like to thank respected, Head of the Department of Computer Science and Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present an internship report. It helped me a lot to realize of what we study for.

I would like to thank the respected faculties who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

A.N. Sathvik

221710308001

ABSTRACT

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human brain so deep learning is also a kind of mimic of human brain. In deep learning, we don't need to explicitly program everything. In this project with the use of deep learning concepts we will be identifying ships in a satellite image, we will be having a picture from a satellite and we have to identify whether it has a ship in it or not.

So, the approach I will be using for this project to work is deep learning and here with the help of that approach, we should mainly be able to differentiate between images having a ship and images not having it. I will be using CNN method, I know that it will be giving two results so, I used binary entropy method to differentiate both the outcomes.

TABLE OF CONTENTS

CHAPTER 1: MACHINE LEARNING

1.1 INTRODUCTION.....	1
1.2 IMPORTANCE OF MACHINE LEARNING.....	1
1.3 USES OF MACHINE LEARNING.....	2
1.4 TYPES OF LEARNING ALGORITHMS.....	3
1.4.1 Supervised Learning.....	3
1.4.2 Unsupervised Learning.....	4
1.4.3 Semi Supervised Learning.....	5

CHAPTER 2: PYTHON.....7

2.1 INTRODUCTION TO PYTHON.....	7
2.2 HISTORY OF PYTHON.....	7
2.3 FEATURES OF PYTHON.....	8
2.4 HOW TO SETUP PYTHON.....	9
2.4.1 Installation (using python IDLE)	9
2.4.2 Installation (using Anaconda)	10
2.5 PYTHON VARIABLE TYPES.....	12
2.5.1 Python Numbers.....	13
2.5.2 Python Strings.....	13
2.5.3 Python Lists.....	13
2.5.4 Python Tuples.....	13
2.5.5 Python Dictionary.....	14
2.6 PYTHON FUNCTION.....	14
2.6.1 Defining a Function.....	14

2.6.2 Calling a Function.....	15
2.7 PYTHON USING OOP's CONCEPTS.....	15
2.7.1 Class.....	15
2.7.2 __init__ method in class.....	16
CHAPTER 3: DEEP LEARNING.....	17
3.1 INTRODUCTION.....	17
3.2 IMPORTANCE.....	19
3.3 RELATION BETWEEN DATA MINING, MACHINE LEARNING AND DEEP LEARNING.....	20
3.4 USES OF DEEP LEARNING.....	20
3.5 TYPES OF DEEP LEARNING.....	26
CHAPTER 4: CASE STUDY.....	32
4.1 PROBLEM STATEMENT.....	32
4.2 DATA SET.....	32
4.3 OBJECTIVE OF THE CASE STUDY.....	32
4.4 Project Requirements.....	33
4.4.1 Packages used.....	33
4.4.2 Versions of package.....	33
4.4.3 Algorithms Used.....	33
CHAPTER 5: MODEL BUILDING USING CNN.....	34
5.1 PREPROCESSING OF THE DATA.....	34
5.1.1 Getting the Data Set.....	34
5.1.2 Importing Libraries.....	34
5.1.3 Importing ZIP File.....	34
5.1.4 Converting JSON File to array format.....	35

5.2 VISUALIZATION OF IMAGES.....	37
5.2.1 Visualization of shipsnet images.....	38
5.2.2 Visualization of scenes images	40
5.3 Preparing Input & Output data.....	41
5.3.1 Input Data.....	41
5.3.2 Output Data.....	42
5.3.3 Training Data and Scaling.....	43
5.4 Building the Model.....	43
5.4.1 Convolution Neural Network.....	43
5.4.2 Compile the Model.....	50
5.4.3 Training the Model.....	52
5.4.4 Accuracy, Loss.....	53
5.5 Prediction of ships in satellite imagery.....	54
 CONCLUSION.....	56
REFERENCES.....	56

LIST OF FIGURES:

Figure 1.1: Process flow	1
Figure 1.4.1.1: Supervised Learning	4
Figure 1.4.2.1: Unsupervised Learning	5
Figure 1.4.3.1: Semi-supervised Learning.....	6
Figure 2.3.1: Features of python.....	8
Figure 2.4.1.1: Python Download.....	10
Figure 2.4.2.1: Anaconda Download	11

Figure 2.4.2.2: Jupyter Notebook	12
Figure 2.7.1.1: Defining class.....	16
Figure 3.1.1: Idea of Deep Learning	18
Figure 3.4.1: Self-Driving	21
Figure 3.4.2: Healthcare	22
Figure 3.4.3: Voice Recognition.....	23
Figure 3.4.4: Image Recognition	24
Figure 3.4.5: Earthquakes.....	25
Figure 3.5.1: Classic Neural Networks.....	27
Figure 3.5.2: CNN.....	28
Figure 3.5.3: RNN.....	29
Figure 3.5.4: Boltzmann Machines	30
Figure 3.5.5: Autoencoders.....	31
Figure 5.1.2.1: Importing Libraries.....	34
Figure 5.1.3.1: Contents of ZIP File	35
Figure 5.1.4.1: JSON File.....	36
Figure 5.1.4.2: Input and Output.....	36
Figure 5.1.4.3: Folders of ZIP File.....	36
Figure 5.1.4.4: Location of ships & scenes	37
Figure 5.2.1: File name for ships	37
Figure 5.2.1.1: Visualization of ships	39
Figure 5.2.1.2: File name for scenes	39
Figure 5.2.2.1: Visualization of scenes.....	40
Figure 5.2.2.2: Shapes of input & output.....	40
Figure 5.3.1.1: Input Data	41

Figure 5.3.1.2: Conversion to keras model.....	42
Figure 5.3.2.1: Output Data.....	42
Figure 5.3.2.2: Traverse all images	42
Figure 5.3.3.1: Training and Scaling.....	43
Figure 5.4.1.1: Architecture of CNN	43
Figure 5.4.1.2: Convolution.....	44
Figure 5.4.1.3: Strides.....	45
Figure 5.4.1.4: Padding.....	46
Figure 5.4.1.5: Max Pooling.....	47
Figure 5.4.1.6: Importing required layers	48
Figure 5.4.1.7: Constructing the model.....	48
Figure 5.4.2.1: Compiling the Model.....	50
Figure 5.4.3.1: Training the Model	52
Figure 5.4.4.1: Accuracy & Loss variation.....	53
Figure 5.4.4.2: Graphs of train_acc, val_acc & train_loss, val_loss	54
Figure 5.5.1: Steps in Prediction of image.....	54
Figure 5.5.2: Procedure.....	55
Figure 5.5.3: Output after Predicting image.....	55

CHAPTER-1

MACHINE LEARNING

1.1 INTRODUCTION:

Artificial intelligence (AI) traditionally refers to an artificial creation of humanlike intelligence that can learn, reason, plan, perceive, or process natural language.

Artificial intelligence is further defined as “narrow AI” or “general AI”. Narrow AI, which we interact with today, is designed to perform specific tasks within a domain (e.g. language translation). General AI is hypothetical and not domain specific, but can learn and perform tasks anywhere. This is outside the scope of this paper. This paper focuses on advances in narrow AI, particularly on the development of new algorithms and models in a field of computer science referred to as machine learning.

1.2 IMPORTANCE OF MACHINE LEARNING:

Algorithms are a sequence of instructions used to solve a problem. Algorithms, developed by programmers to instruct computers in new tasks, are the building blocks of the advanced digital world we see today. Computer algorithms organize enormous amounts of data into information and services, based on certain instructions and rules. It's an important concept to understand, because in machine learning, learning algorithms – not computer programmers – create the rules.

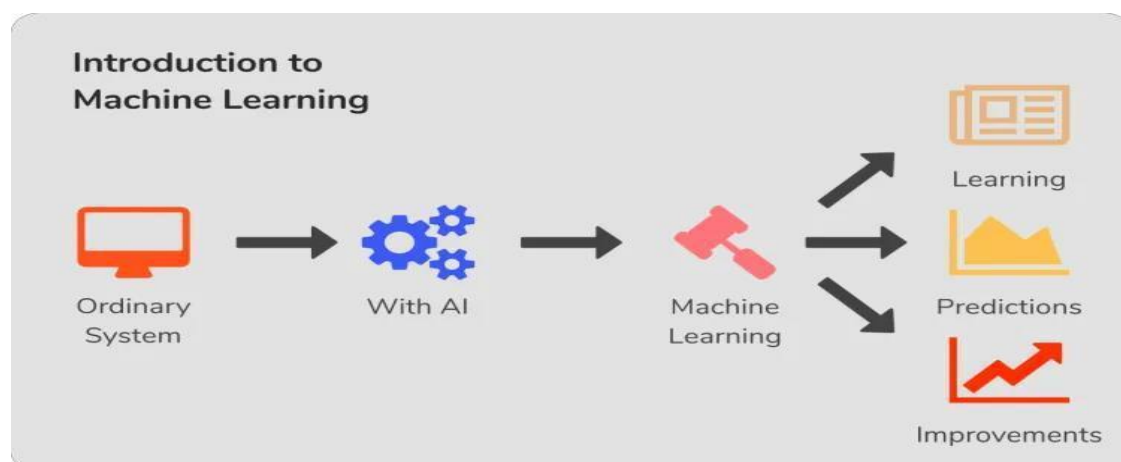
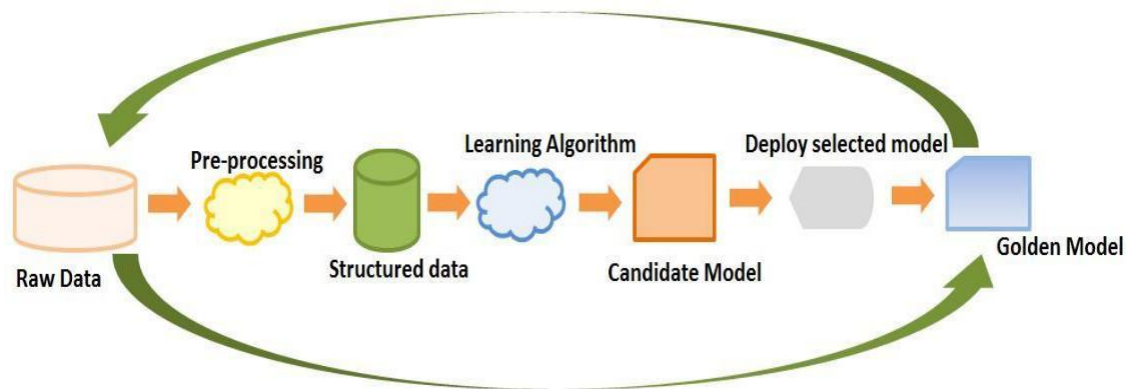


Figure 1.1: Process Flow

The process flow depicted here represents how machine learning works



1.3 USES OF MACHINE LEARNING: -

There are limitless applications of machine learning and there are a lot of machine learning algorithms available to learn. They are available in every form from simple to highly complex. Top 10 Uses of machine learning are as follows:

Image Recognition

The image recognition is one of the most common uses of machine learning applications. It can also be referred to as a digital image and for these images, the measurement describes the output of every pixel in an image. The face recognition is also one of the great features that have been developed by machine learning only. It helps to recognize the face and send the notifications related to that to people.

Voice Recognition

Machine learning (ML) also helps in developing the application for voice recognition. It also referred to as virtual personal assistants (VPA). It will help you to find the information when asked over the voice. After your question, that assistant will look out for the data or the information that has been asked by you and collect the required information to provide you

with the best answer. There are many devices available in today's world of Machine learning for voice recognition that is Amazon echo and googles home is the smart speakers. There is one mobile app called Google allo and smartphones are Samsung S8 and Bixby.

1.4 TYPES OF LEARNING ALGORITHMS:



As with any method, there are different ways to train machine learning algorithms, each with their own advantages and disadvantages. To understand the pros and cons of each type of machine learning, we must first look at what kind of data they ingest. In ML, there are two kinds of data — labelled data and unlabelled data.

There are also some types of machine learning algorithms that are used in very specific use cases, but three main methods are used today.

1.4.1 Supervised Learning:

Supervised learning is one of the most basic types of machine learning. In this type, the machine learning algorithm is trained on labelled data. Even though the data needs to be labelled accurately for this method to work, supervised learning is extremely powerful when used in the right circumstances.

Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.
 $Y = f(X)$

The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

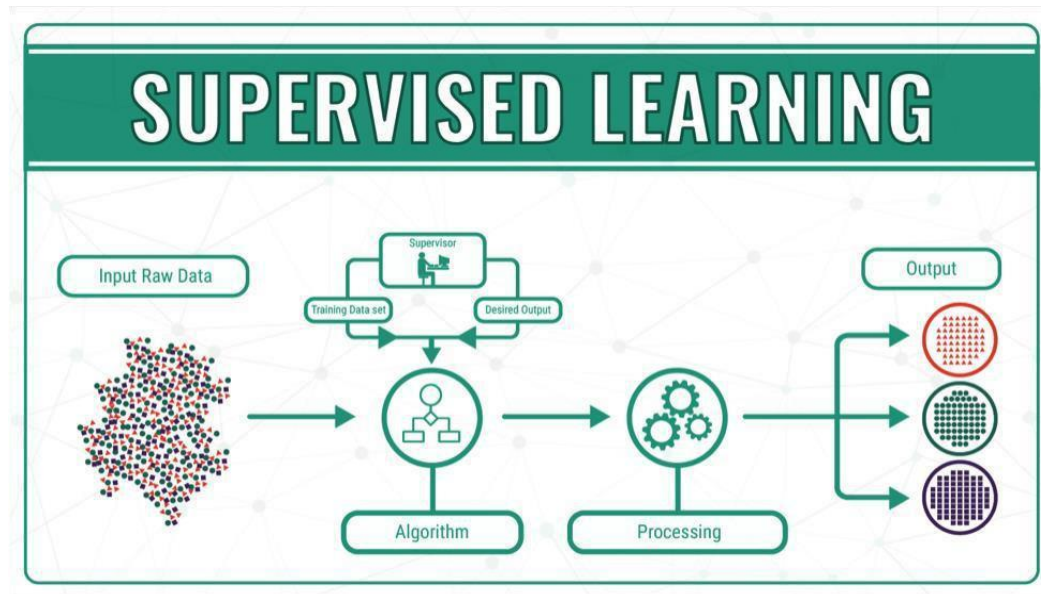


Figure 1.4.1.1: Supervised Learning

1.4.2 Unsupervised Learning:

Unsupervised machine learning holds the advantage of being able to work with unlabeled data. This means that human labor is not required to make the dataset machine-readable, allowing much larger datasets to be worked on by the program.

In supervised learning, the labels allow the algorithm to find the exact nature of the relationship between any two data points. However, unsupervised learning does not have labels to work off of, resulting in the creation of hidden structures. Relationships between data points are perceived by the algorithm in an abstract manner, with no input required from human beings.

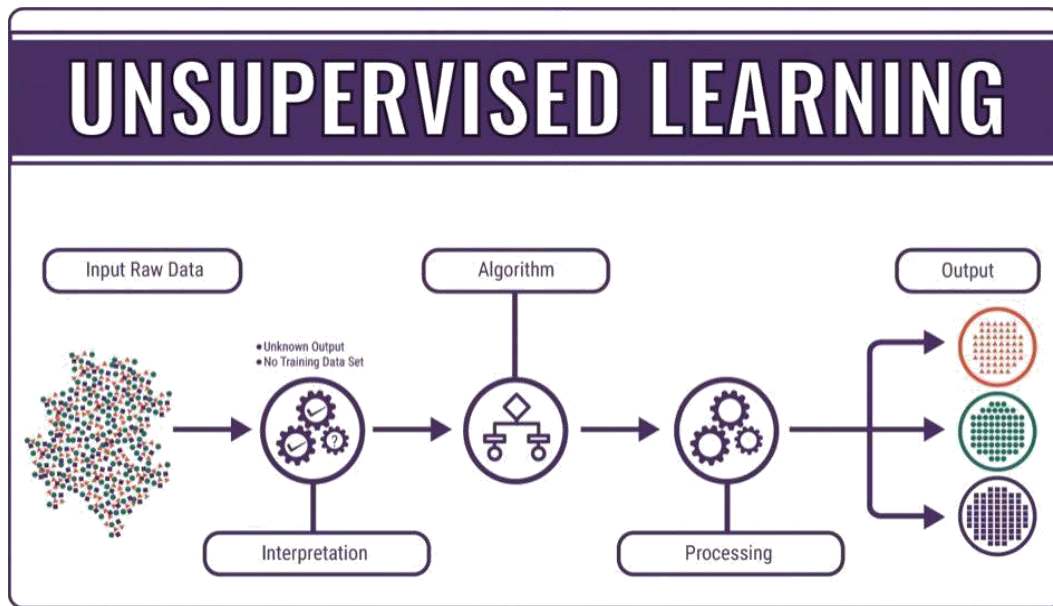


Figure 1.4.2.1: Unsupervised Learning

1.4.3 Semi Supervised Learning:

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labelled and unlabeled data for training. In a typical scenario, the algorithm would use a small amount of labelled data with a large amount of unlabeled data.

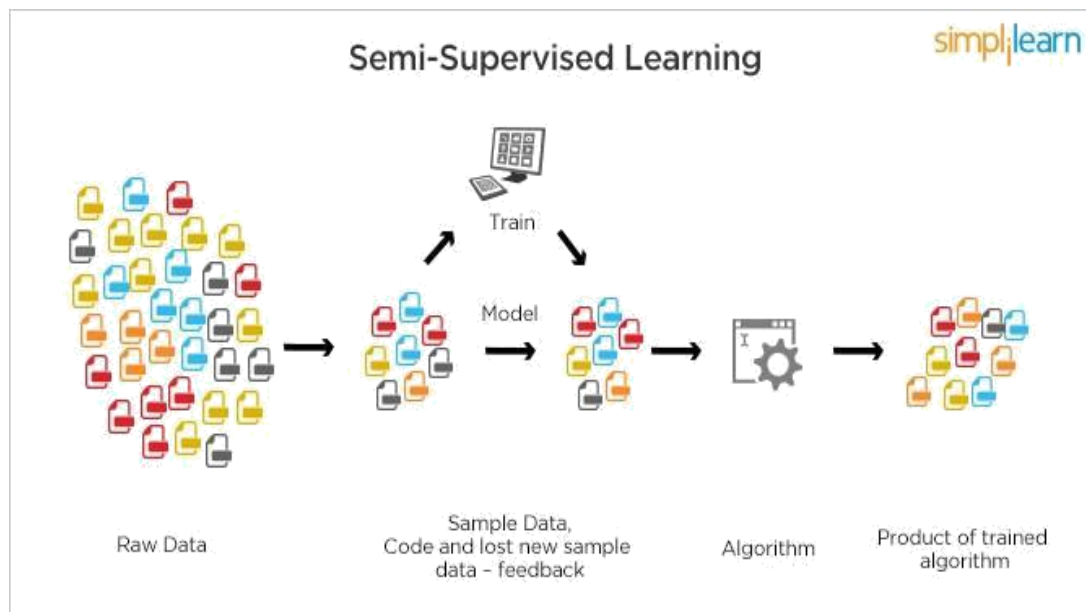


Figure 1.4.3.1: Semi Supervised Learning

CHAPTER 2

PYTHON

Basic programming language used for machine learning is: PYTHON

2.1 INTRODUCTION TO PYTHON:

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development, □ mathematics,
- system scripting.

2.2 HISTORY OF PYTHON:

- Python was developed by GUIDO VAN ROSSUM in early 1990's
- Its latest version is 3.7, it is generally called as python3

2.3 FEATURES OF PYTHON:

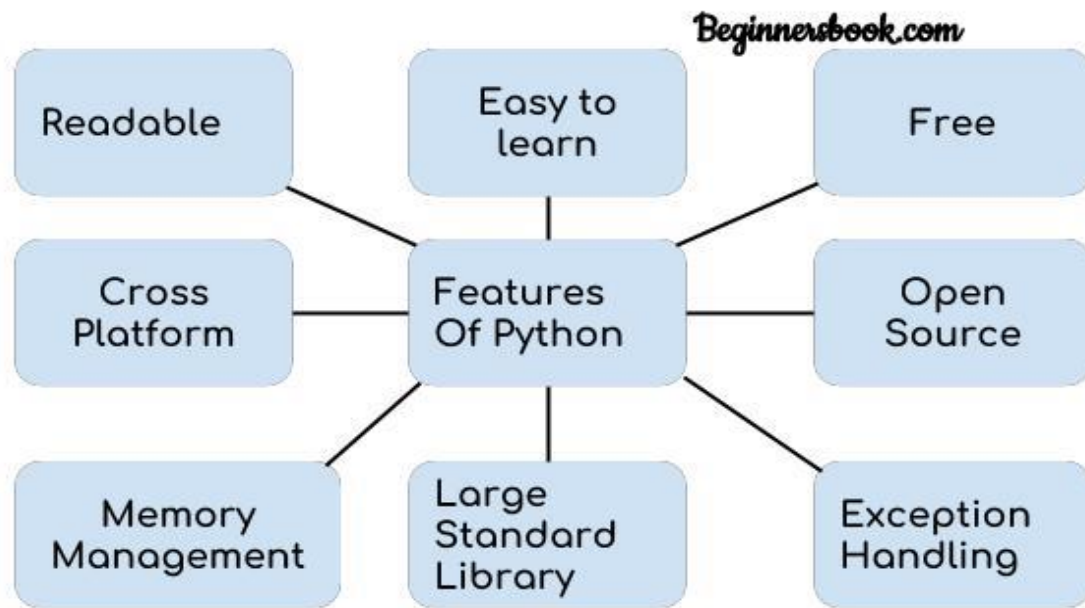


Figure 2.3.1: Features of python

- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax, this allows the student to pick up the language quickly.
- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintaining.
- **A broad standard library:** Python's bulk of the library is very portable and cross platform compatible on UNIX, Windows, and Macintosh.

- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

2.4 HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

2.4.1 Installation (using python IDLE):

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- Download python from www.python.org



Figure 2.4.1.1: Python download

- When the download is completed, double click the file and follow the instructions to install it.
- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.

2.4.2 Installation (using Anaconda):

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.
- Conda is a package manager quickly installs and manages packages.

In WINDOWS:

In windows

- Step 1: Open Anaconda.com/downloads in web browser.
- Step 2: Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer).
- Step 3: select installation type (all users).

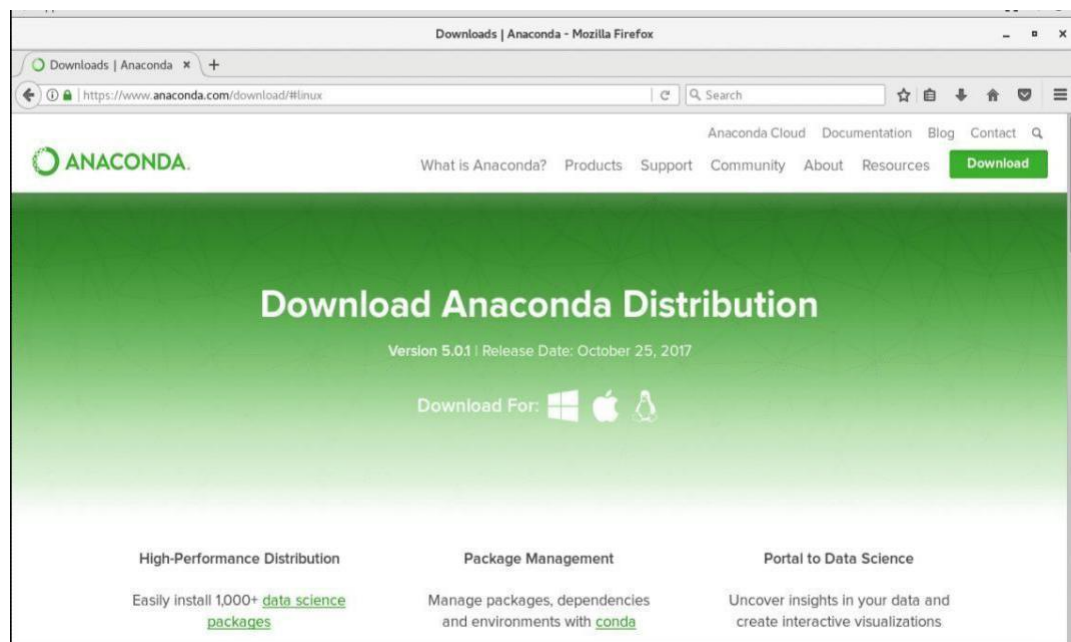


Figure 2.4.2.1: Anaconda download

- Step 4: Select path (i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish.
- Step 5: Open jupyter notebook (it opens in default browser).

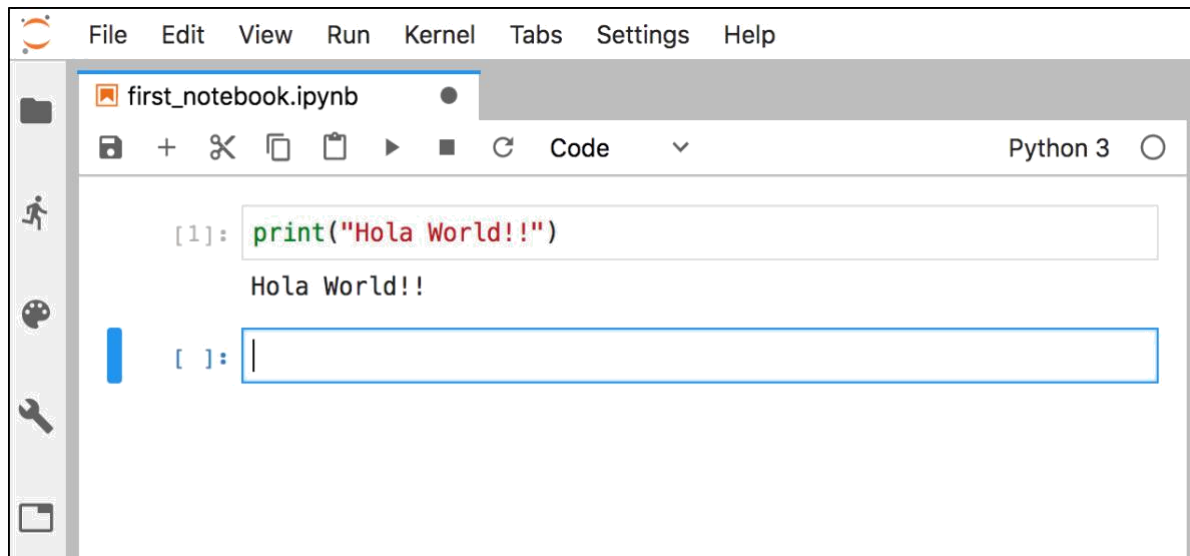


Figure 2.4.2.2: Jupyter notebook

2.5 PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types –
 1. Numbers
 2. Strings
 3. Lists
 4. Tuples
 5. Dictionary

2.5.1 Python Numbers:

- Number data types store numeric values. Number objects are created when you assign a value to them.
- Python supports four different numerical types – int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

2.5.2 Python Strings:

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

2.5.3 Python Lists:

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets ([]).
- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

2.5.4 Python Tuples:

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

- The main differences between lists and tuples are: Lists are enclosed in brackets ([]) and their elements and size can be changed, while tuples are enclosed in parentheses () and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example – Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

2.5.5 Python Dictionary:

- Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces ({}) and values can be assigned and accessed using square braces ([]).
- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.
- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

2.6 PYTHON FUNCTION:

2.6.1 Defining a Function:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword `def` followed by the function name and parentheses (i.e. `()`). Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses the code block within every function starts with a colon `:` and is indented. The statement returns

exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

2.6.2 Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

2.7 PYTHON USING OOP's CONCEPTS:

2.7.1 Class:

- **Class:** A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- **Class variable:** A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- **Data member:** A class variable or instance variable that holds data associated with a class and its objects.
- **Instance variable:** A variable that is defined inside a method and belongs only to the current instance of a class.
- **Defining a Class:** We define a class in a very similar way how we define a function. Just like a function; we use parentheses and a colon after the class name (I.e. () :) when we define a class. Similarly, the body of our class is 14 indented like a functions body

A simple class definition: *student*

```
class student:
    """A class representing a student."""
    def __init__(self, n, a):
        self.full_name = n
        self.age = a
    def get_age(self):
        return self.age
```

CSE 391 - Intro to AI

15



Figure 2.7.1.1: Defining a Class

2.7.2 `__init__` method in Class:

- The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.
- The init method has a special name that starts and ends with two underscores

CHAPTER 3

DEEP LEARNING

3.1 INTRODUCTION:

To understand what deep learning is, we first need to understand the relationship deep learning has with machine learning, neural networks, and artificial intelligence. The best way to think of this

relationship is to visualize them as concentric circles:

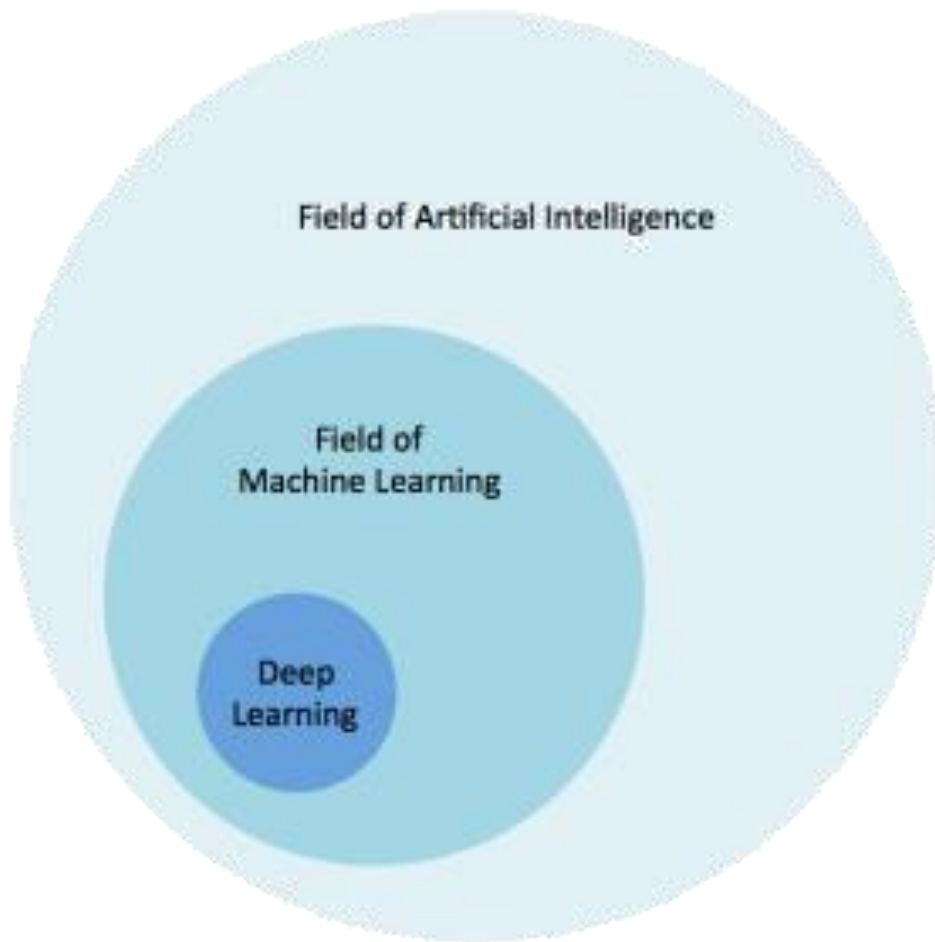


Figure 3.1.1: Idea of deep Learning

Deep learning is a specific subset of Machine Learning, which is a specific subset of Artificial Intelligence. For individual definitions:

- Artificial Intelligence is the broad mandate of creating machines that can think intelligently
- Machine Learning is one way of doing that, by using algorithms to glean insights from data
- Deep Learning is one way of doing that, using a specific algorithm called a Neural Network

3.2 IMPORTANCE

Artificial Intelligence as the name suggests is to make a machine artificially intelligent i.e. making the machine think or act like humans. It's not that the concept of AI is totally new instead it has been explored many decades ago but in recent years it has gained popularity and has led to various miraculous discoveries because of 2 major factors.

The 2 factors which have made all the world invest in this field are increase in computational speed and the amount of useful data available. In fact, the 90% of the total data available as of now has been gathered in the previous 3 to 4 years itself. If a robot is hard coded i.e. all the logic has been coded manually into the system then it's not AI so it can't be said that simply robots means AI.

To understand AI more clearly let's dive into its most popular and useful subset, Machine learning, it simply means making a machine learn from its experience and improving its performance with time just like the case of a human baby.

Concept of machine learning became feasible only when sufficient amount of data was made available for training machines. Machine learning helps in dealing with complex and robust systems. But most of the miraculous discoveries which we have come across in recent years have been made possible out of Deep Learning.

Now the question arises what is it in deep learning which has brought such a revolution in our lives. Basically, deep learning is itself a subset of machine learning but in this case the machine learns in a way in which humans are supposed to learn. The structure of deep learning model is highly similar to a human brain with large number of neurons and nodes like neurons in human brain thus resulting in artificial neural network.

In applying traditional machine learning algorithms, we have to manually select input features from complex data set and then train them which becomes a very tedious job for ML scientist but in neural networks we don't have to manually select useful input features, there are various layers of neural networks for handling complexity of the data set and algorithm as well.

In my recent project on human activity recognition, when we applied traditional machine learning algorithm like K-NN then we have to separately detect human and its

Journalism, Entertainment, Online Retail Store, Automobile, Banking and Finance, Healthcare, Manufacturing or even Digital Sector. Video recommendations, Mail Services, Self-Driving cars, Intelligent Chat bots, Voice Assistants are just trending achievements of Deep Learning.

Furthermore, Deep learning can most profoundly be considered as future of Artificial Intelligence due to constant rapid increase in amount of data as well as the gradual development in hardware field as well, resulting in better computational power.

3.3 RELATION BETWEEN DATA MINING, MACHINE LEARNING AND DEEP LEARNING:

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovers previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions. Deep learning, on the other hand, uses advanced computing power and special types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

3.4 USES OF DEEP LEARNING

It's predicted that many deep learning applications will affect your life in the near future. Actually, they are already making an impact. Within the next five to 10 years, deep learning development tools, libraries, and languages will become standard components of every software development toolkit.

So, here are the TOP 5 Deep Learning applications that will rule the world in 2018 and beyond

1. Self-driving cars

Companies building these types of driver-assistance services, as well as full-blown self-driving cars like Google's, need to teach a computer how to take over key parts (or all) of driving using digital sensor instead of a human's senses. To do that companies generally start out by training algorithms using a large amount of data.

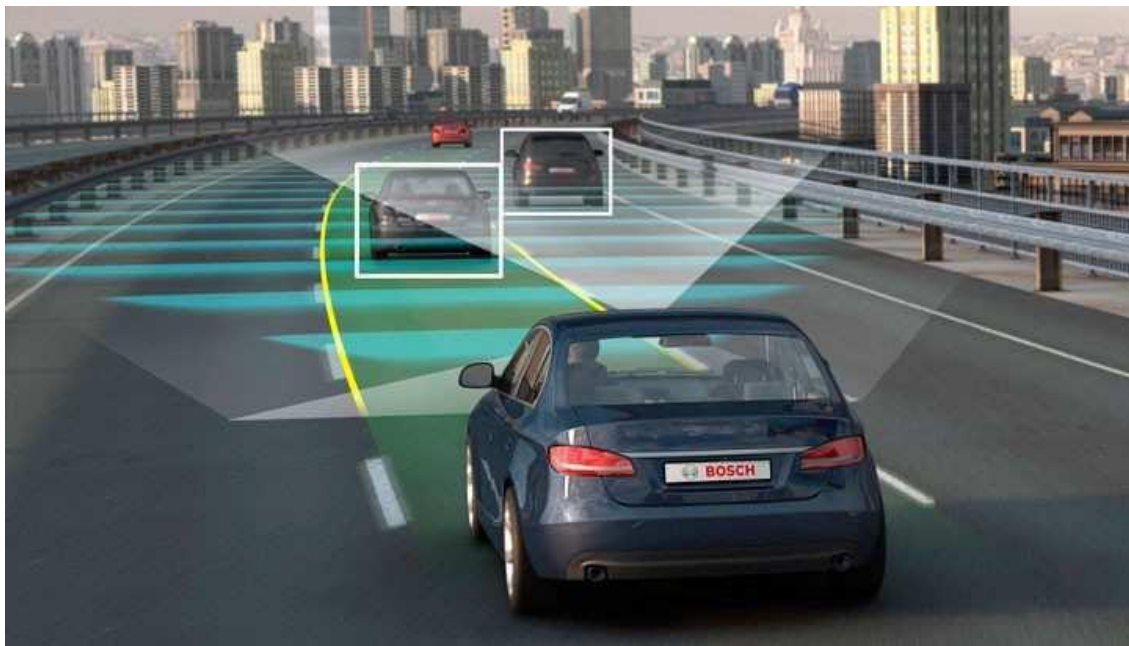


Fig 3.4.1: Self-driving cars

2. Deep Learning in Healthcare

Breast or Skin-Cancer diagnostics? Mobile and Monitoring Apps? or prediction and personalized medicine the basis of Biobank-data? AI is completely reshaping life sciences, medicine, and healthcare as an industry. Innovations in AI are advancing the future of precision medicine and population health management in unbelievable ways. Computer-aided detection, quantitative imaging, decision support tools and computer-aided diagnosis will play a big role in years to come.



Fig 3.4.2: Healthcare

3. Voice Search & Voice-Activated Assistants

One of the most popular usage areas of deep learning is voice search & voice-activated intelligent assistants. With the big tech giants have already made significant investments in this area, voice-activated assistants found on nearly every smartphone. Apple's Siri is on the market since October 2011. Google Now, the voice-activated assistant for Android, was launched less than a year after Siri. The newest of the voice-activated intelligent assistants is Microsoft Cortana.

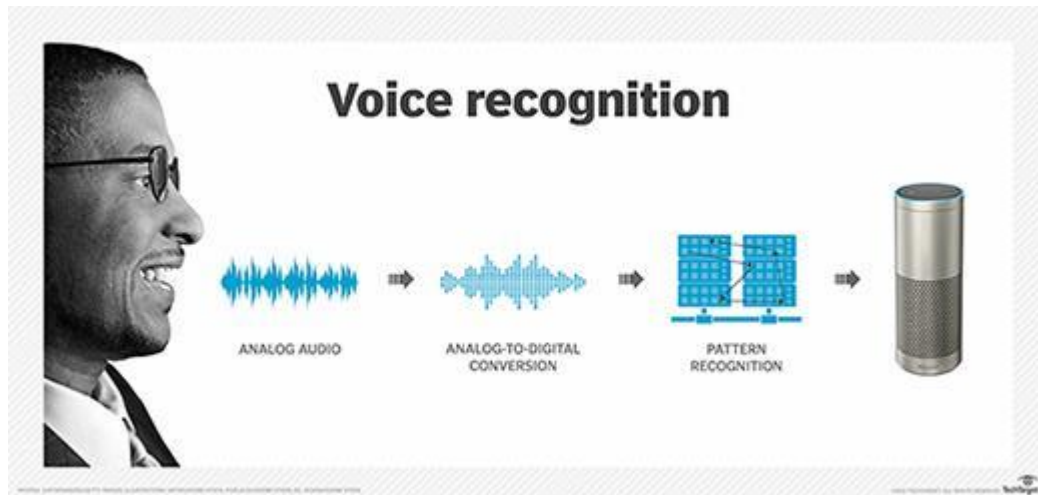


Fig 3.4.3: Voice Recognition

4. Image Recognition

Another popular area regarding deep learning is image recognition. It aims to recognize and identify people and objects in images as well as to understand the content and context. Image recognition is already being used in several sectors like gaming, social media, retail, tourism, etc.

This task requires the classification of objects within a photograph as one of a set of previously known objects. A more complex variation of this task called object detection involves specifically identifying one or more objects within the scene of the photograph and drawing a box around them.

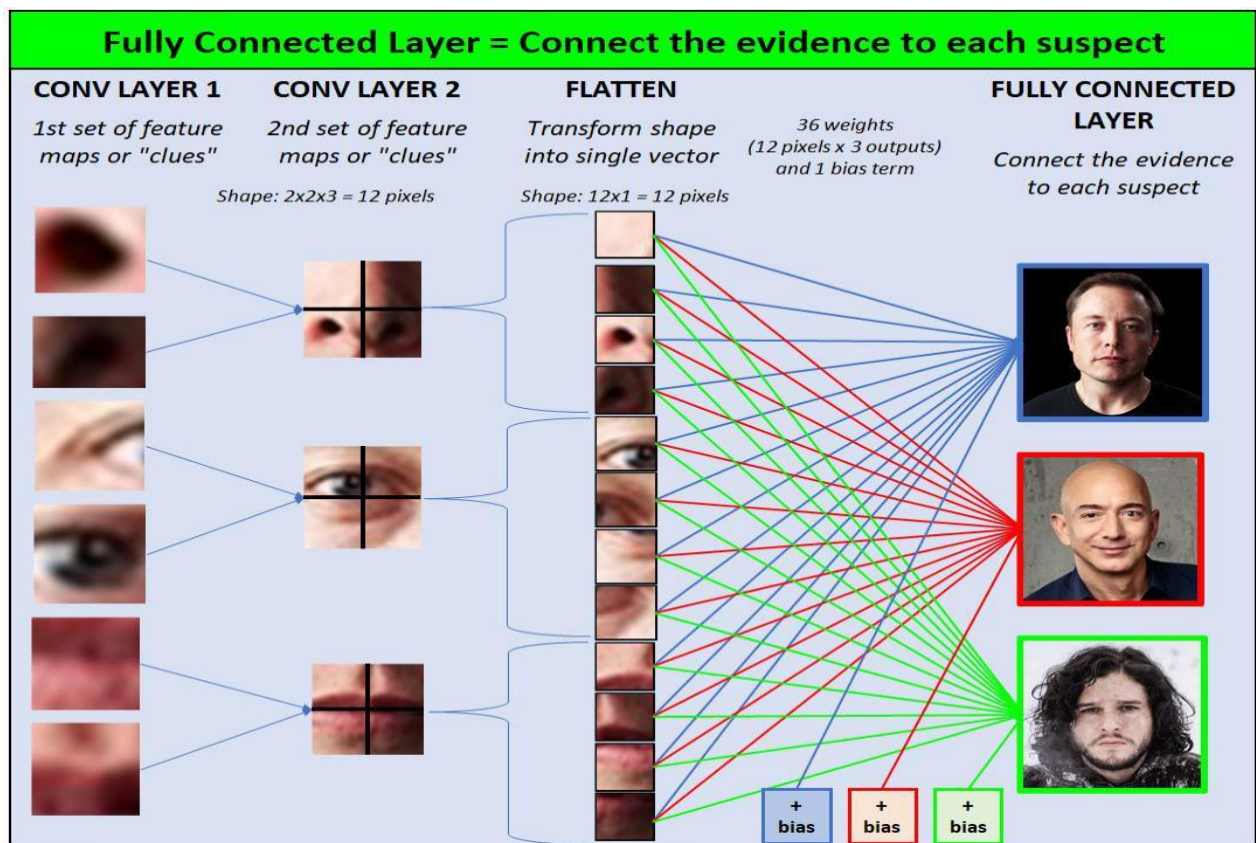


Fig 3.4.4: Image Recognition

5. Predicting Earthquakes

Harvard scientists used Deep Learning to teach a computer to perform viscoelastic computations, these are the computations used in predictions of earthquakes. Until their paper, such computations were very computer intensive, but this application of Deep Learning improved calculation time by 50,000%. When it comes to earthquake calculation, timing is important and this improvement can be vital in saving a life.

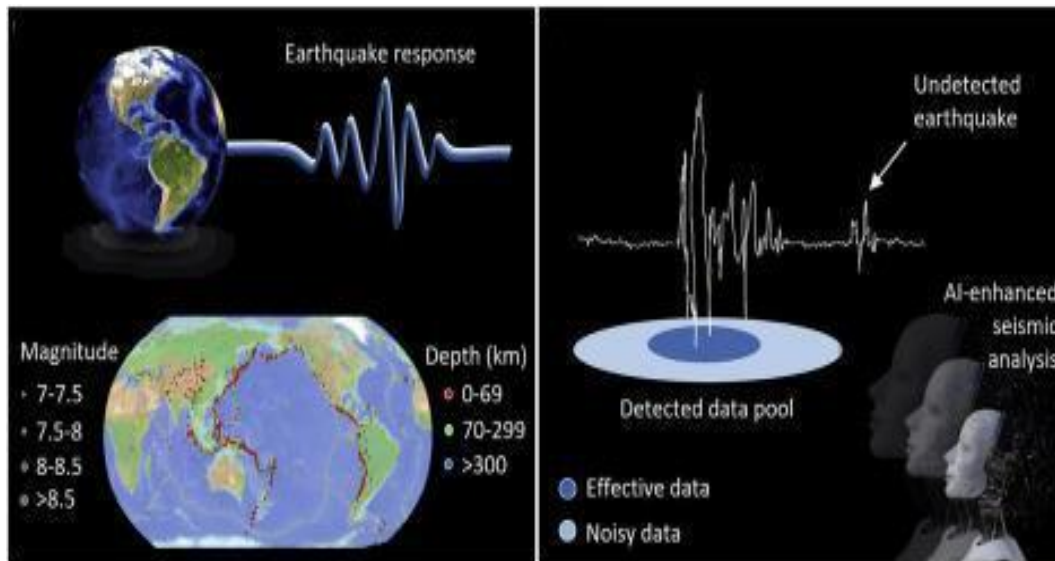


Fig 3.4.5: Earthquakes

3.5 TYPES OF DEEP LEARNING

Supervised Models

- Classic Neural Networks (Multilayer Perceptrons)
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)

Unsupervised Models

- Self-Organizing Maps (SOMs)
- Boltzmann Machines
- Autoencoders

Unsupervised



Supervised



Classic Neural Networks (Multilayer Perceptrons)

Classic Neural Networks can also be referred to as Multilayer perceptrons. The perceptron model was created in 1958 by American psychologist Frank Rosenblatt. Its singular nature allows it to adapt to basic binary patterns through a series of inputs, simulating the learning patterns of a human-brain. A Multilayer perceptron is the classic neural network model consisting of more than 2 layers.

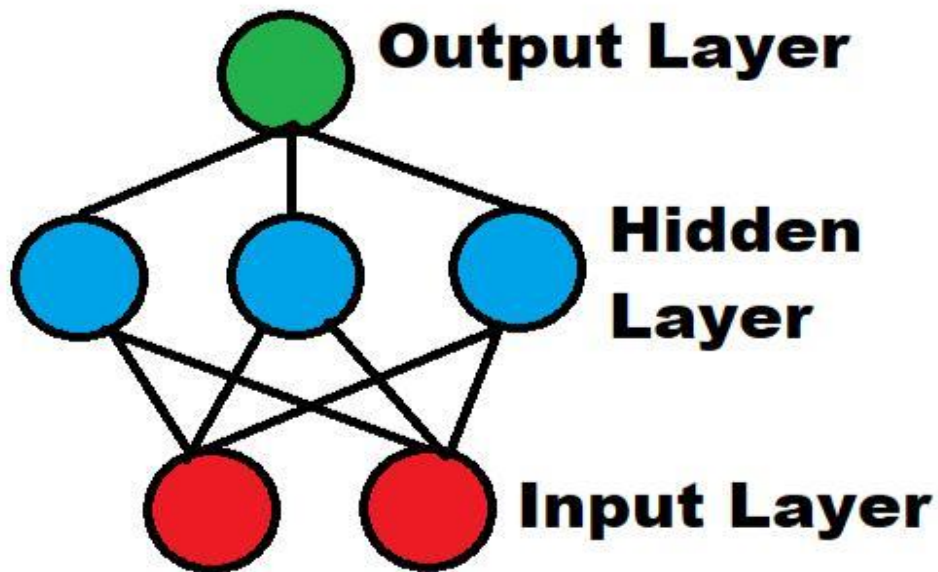


Fig 3.5.1: Classic Neural Network

Convolutional Neural Networks

A more capable and advanced variation of classic artificial neural networks, a Convolutional Neural Network (CNN) is built to handle a greater amount of complexity around pre-processing, and computation of data.

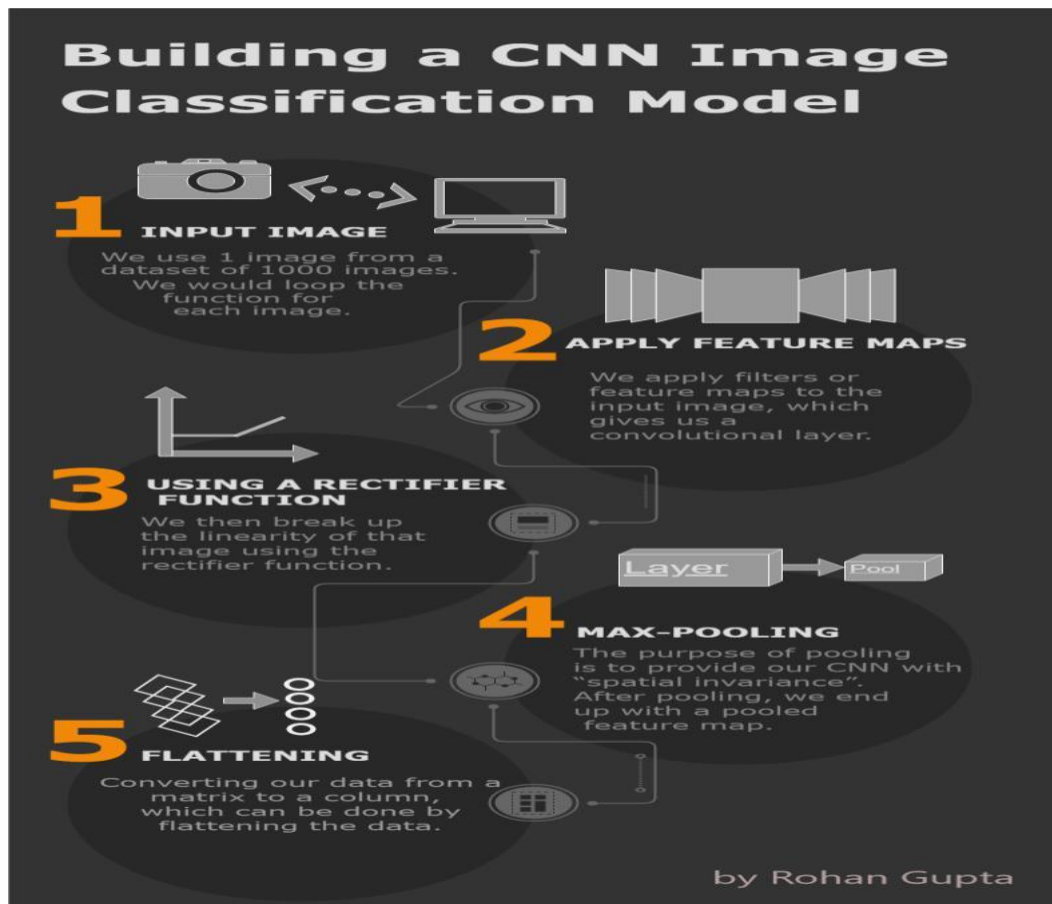


Fig 3.5.2: CNN

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) were invented to be used around predicting sequences.

LSTM (Long short-term memory) is a popular RNN algorithm with many possible use cases:

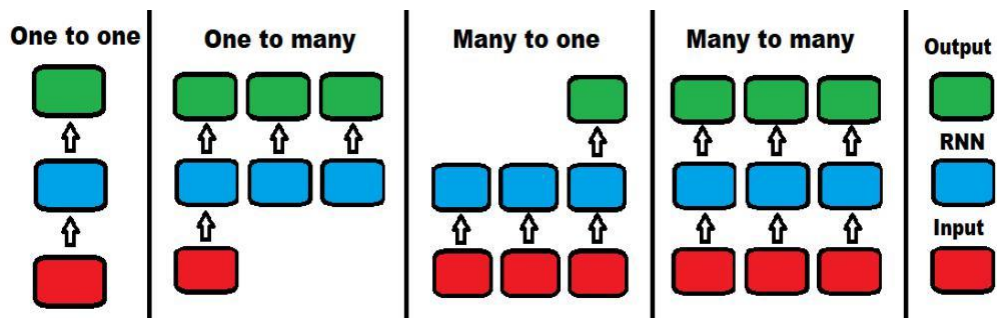


Fig 3.5.3: RNN

Self-Organizing Maps

Self-Organizing Maps or SOMs work with unsupervised data and usually help with dimensionality reduction (reducing how many random variables you have in your model). The output dimension is always 2-dimensional for a self-organizing map. So, if we have more than 2 input features, the output is reduced to 2 dimensions. Each synapse connecting out input and output nodes have a weight assigned to them. Then, each data point competes for representation in the model. The closest node is called the BMU (best matching unit), and the SOM updates its weights to move closer to the BMU. The neighbors of the BMU keep decreasing as the model progresses. The closer to the BMU a node is, the more its weights would change.

Boltzmann Machines

In the 4 models above, there's one thing in common. These models work in a certain direction. Even though SOMs are unsupervised, they still work in a particular direction as do supervised models. By direction, I mean:

Input \rightarrow Hidden Layer \rightarrow Output.

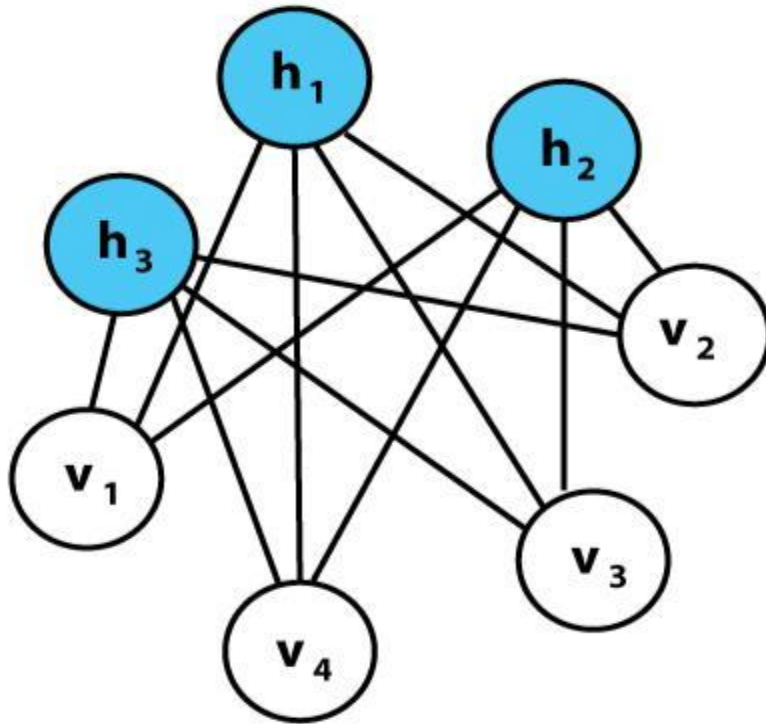


Fig 3.5.4: Boltzmann Machines

Autoencoders

Autoencoders work by automatically encoding data based on input values, then performing activation function, and finally decoding the data for output. A bottleneck of some sort imposed on the input features, compressing them into fewer categories. Thus, if some inherent structure exists within the data, the autoencoder model will identify and leverage it to get the output.

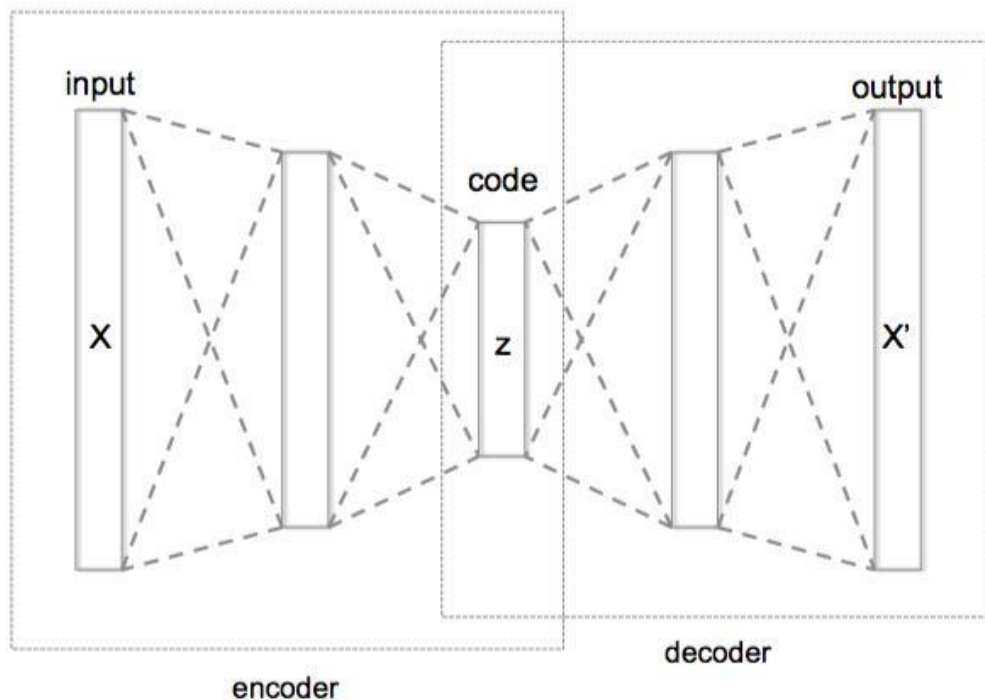


Fig 3.5.5: Autoencoders

CHAPTER 4

CASE STUDY

4.1 PROBLEM STATEMENT:

To identify whether a satellite image contains a ship in it or not using deep learning concepts.

4.2 DATA SET:

The dataset consists of image chips extracted from Planet satellite imagery collected over the San Francisco Bay and San Pedro Bay areas of California. It includes 4000 80x80 RGB images labelled with either a "ship" or "no-ship" classification. Image chips were derived from Planet Scope full-frame visual scene products, which are orthorectified to a 3-meter pixel size.

Provided is a zipped directory shipsnet.zip that contains the entire dataset as .png image chips. Each individual image filename follows a specific format: {label} __ {scene id} __ {longitude} _ {latitude}.png

label: Valued 1 or 0, representing the "ship" class and "no-ship" class, respectively.

scene id: The unique identifier of the Planet Scope visual scene the image chip was extracted from. The scene id can be used with the Planet API to discover and download the entire scene.

longitude latitude: The longitude and latitude coordinates of the image centre point, with values separated by a single underscore.

The dataset is also distributed as a JSON formatted text file shipsnet.json. The loaded object contains data, label, scene_ids, and location lists.

4.3 OBJECTIVE OF THE CASE STUDY:

The aim of this dataset is to help address the difficult task of identifying the presence of large ships in satellite images. Automating this process can be applied to many issues including monitoring port activity levels and supply chain analysis. We have to differentiate between images having ships and images not having them.

Link of Dataset: <https://www.kaggle.com/rhammell/ships-in-satellite-imagery/home>

4.4 PROJECT REQUIREMENTS:

4.4.1 Packages Used

- JSON
- NumPy
- Matplotlib
- TensorFlow
- OS
- Pandas
- Keras

4.4.2 Versions of package

Pandas Version- 1.0.5

TensorFlow Version- 2.2

Keras Version- 2.3.1

NumPy Version- 1.18.05

4.4.3 Algorithms Used

The Algorithm I used here is CNN i.e. Convolution Neural Networks.

CHAPTER 5

MODEL BUILDING USING CNN

5.1 PREPROCESSING OF THE DATA:

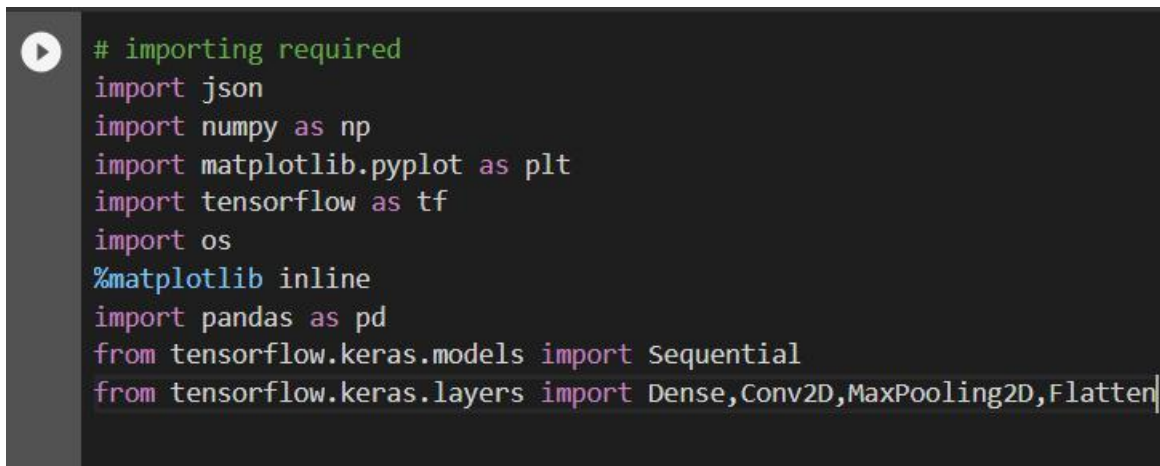
Pre-processing of the data actually involves the following steps:

5.1.1 GETTING THE DATASET:

We can get the data set from the database or we can get the data from client.

5.1.2 IMPORTING THE LIBRARIES:

We have to import the libraries as per the requirement of the algorithm.



```
# importing required
import json
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import os
%matplotlib inline
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten
```

Fig 5.1.2.1: Importing Libraries

5.1.3 IMPORTING THE ZIP-FILE:

There are a few ways we can use to unzip a file in collab, they are:

Method 1:

1. zip the file
2. Upload the zipped file, there is an Upload button under the Files Section.
3. Unzip it using the command on Collab! `unzip level_1_test.zip`

Method 2:

1. Zip the image folder
2. Upload the zip file to your Google drive
3. Turn to GC to authorize and mount your Google drive
4. Follow the link and paste the code to your GC notebook
5. Unzip the file from GC
6. Use the `os` library and list out directory
7. We can now access and see the content of the zip file.

```
# zip file and its content
os.listdir('/content/drive/My Drive/Project DSPS')

['shipsnet.json', 'shipsnet', 'scenes']
```

Fig 5.1.3.1: Content of ZIP File

Method 3:

If you have a download link then just this

`! wget <Link>`

Else upload then to your drive and then just use the following

5.1.4 Converting json file to array format

1. Open the JSON file
2. Load the JSON file.

```
# going through the json file and converting into array format, creating a dataset.
with open('/content/drive/My Drive/Project DSPS/shipsnet.json') as file_name:
    df=json.load(file_name)
    ships=pd.DataFrame(df)
print(ships)
```

	data	...	scene_ids
0	[82, 89, 91, 87, 89, 87, 86, 86, 86, 84, 8...	...	20180708_180909_0f47
1	[76, 75, 67, 62, 68, 72, 73, 73, 68, 69, 6...	...	20170705_180816_103e
2	[125, 127, 129, 130, 126, 125, 129, 133, 132,	20180712_211331_0f06
3	[102, 99, 113, 106, 96, 102, 105, 105, 103, 10...	...	20170609_180756_103a
4	[78, 76, 74, 78, 79, 79, 79, 82, 86, 85, 83, 8...	...	20170515_180653_1007
...
3995	[126, 122, 124, 138, 165, 186, 195, 199, 203,	20170815_180821_102d
3996	[130, 134, 139, 128, 117, 126, 141, 147, 142,	20170730_191230_0f21
3997	[171, 135, 118, 140, 145, 144, 154, 165, 139,	20161116_180804_0e14
3998	[85, 90, 94, 95, 94, 92, 93, 96, 93, 94, 94, 9...	...	20170211_181116_0e16
3999	[122, 122, 126, 126, 142, 153, 174, 190, 185,	20180206_184438_1043

[4000 rows x 4 columns]

Fig 5.1.4.1: JSON File

Initialization of input and output data

```
[107] # taking two parameters from dataset into
      # converting new variables x and y and of type int64 into array
x=np.array(df['data']).astype('int64')
y=np.array(df['labels']).astype('int64')
```

Fig 5.1.4.2: Input and Output

Content of folders inside zip file

There are 2 folders i.e. shipsnet, scenes and a json file in our zip file,

```
[ ] #Checking the contents of other two files i.e shipsnet and scenes
print(len(os.listdir("/content/drive/My Drive/Project DSPS/shipsnet/imgsh")))
print(len(os.listdir("/content/drive/My Drive/Project DSPS/scenes/imgsc")))
```

```
↳ 4000
8
```

Fig 5.1.4.3: Folders of ZIP File

Accessing images using os command

Images are present inside imgsh and imgsc folders inside shipsnet and scenes,

```
[ ] # finding images location of both ships and scenes using path join command
base_dir="/content/drive/My Drive/Project DSPS"
ships_dir=os.path.join(base_dir,'shipsnet')
scenes_dir=os.path.join(base_dir,'scenes')
shipsnet_images=os.path.join(ships_dir,'imgsh')
scenes_images=os.path.join(scenes_dir,'imgsc')
```

Fig 5.1.4.4: Location of ships and scenes

5.2 VISUALIZATION OF IMAGES:

Filename for shipsnet

Here, we are creating a file and storing 8 random images of shipsnet

```
[ ] # File name for ships
file_name_dir_ships=os.listdir(shipsnet_images)
file_name_dir_ships[:8]

[ ] ['1__20170901_181520_0e14__-122.35035768163974_37.7672039845457.png',
     '1__20170903_181304_1041__-122.3359293751746_37.7580383438423.png',
     '1__20170901_181520_0e14__-122.36740002724083_37.80182586116483.png',
     '1__20170909_181729_0e0f__-122.35067750648894_37.78126618441992.png',
     '1__20170905_181214_0f12__-122.32764083883828_37.736846543182324.png',
     '1__20170901_181520_0e14__-122.35121081280293_37.74752401629368.png',
     '1__20170830_181003_0f4e__-122.35157982161317_37.78327205920199.png',
     '1__20170903_181304_1041__-122.32785024812625_37.73627990920489.png']
```

Fig 5.2.1: File name for Ships

5.2.1 Visualization of shipsnet images

```
[ ] # visualising some images from shipsnet
plt.figure(figsize=(16,16))
j=1
for i in range(16):
    img=plt.imread(os.path.join(shipsnet_images,file_name_dir_ships[i]))
    plt.subplot(4,4,j)
    plt.imshow(img)
    plt.title(img.shape)
    plt.axis('off')

    j+=1
```

Now, I am visualizing 16 random images from shipsnet and we can observe that there will be images having ship in it and images with no ship in it, so we get a clear idea on what we are working on and what we are trying to differentiate.

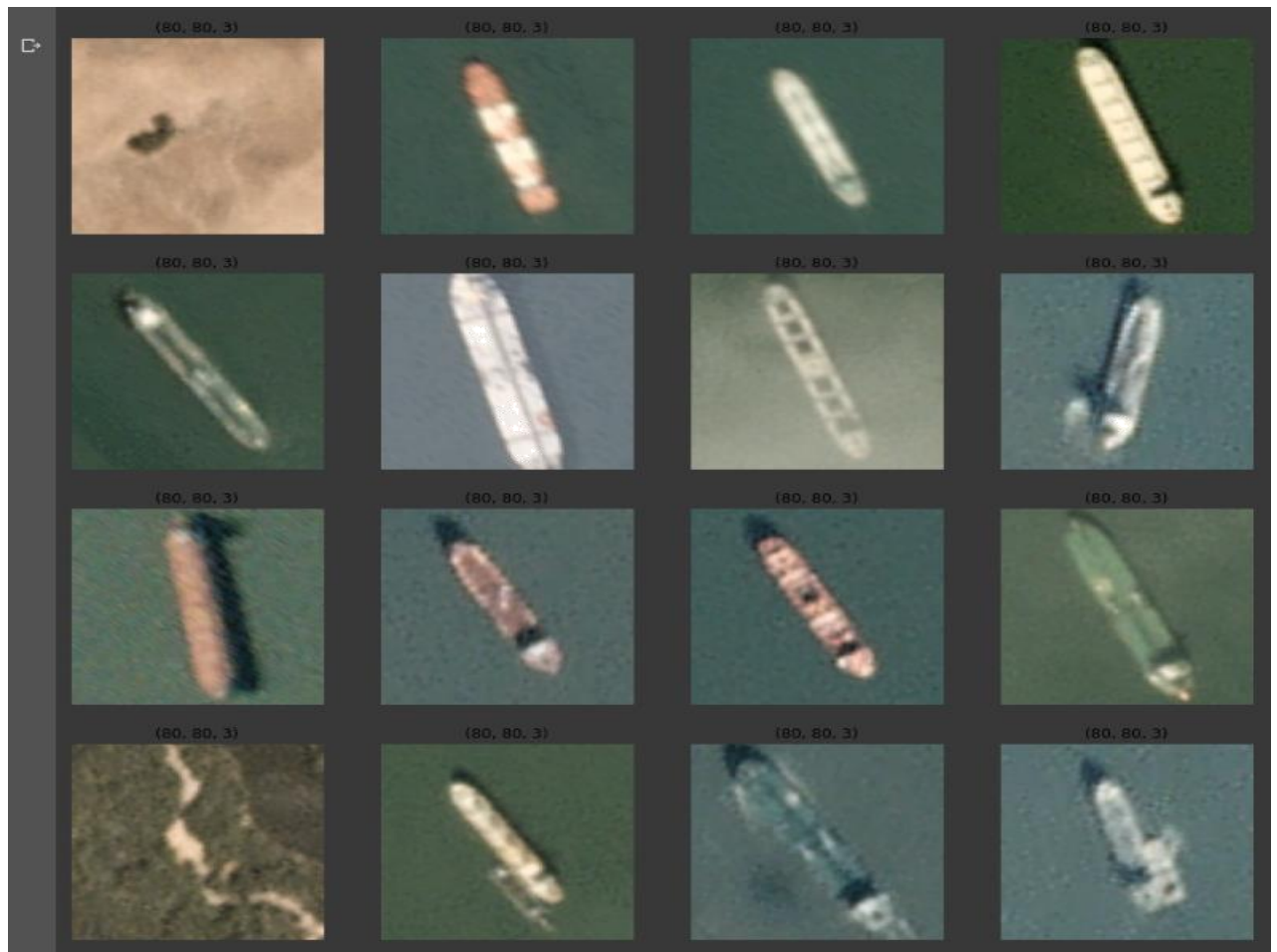


Fig 5.2.1.1: Visualization of ships

Filename for scenes

Here, we are creating a file and storing 8 random images of scenes

```
[ ] # File name for scenes
file_name_dir_scenes=os.listdir(scenes_images)
file_name_dir_scenes[:8]

[ ] ['lb_1.png',
     'lb_2.png',
     'lb_3.png',
     'lb_4.png',
     'sfbay_1.png',
     'sfbay_2.png',
     'sfbay_3.png',
     'sfbay_4.png']
```

Fig 5.2.1.2: File name for scenes

5.2.2 Visualization of scenes images

```
[ ] # visualising some images from scenes
plt.figure(figsize=(16,16))
s=1
for k in range(8):
    img=plt.imread(os.path.join(scenes_images,file_name_dir_scenes[k]))
    plt.subplot(4,4,s)
    plt.imshow(img)
    plt.title(img.shape)
    plt.axis('off')

    s+=1
```



Fig 5.2.2.1: Visualization of scenes

Shapes of Input and Output Data

```
[ ] # shapes of input and output data
print(x.shape)
print(y.shape)

[ ] (4000, 19200)
    (4000,)
```

Fig 5.2.2.2: Shape of input & output

5.3 Preparing Input and Output Data

5.3.1 Input Data

According to our approach we have x as input and we are preparing it for training. We can see that the shape of image is not in the required manner i.e. it is in the form of 3,80,80 but we want it to be 80,80,3 in-order to make this happen we use transpose command.

```
[ ] #Input data is having content in the form of array
x

[ ] array([[ 82,  89,  91, ...,  86,  88,  89],
          [ 76,  75,  67, ...,  54,  57,  58],
          [125, 127, 129, ..., 111, 109, 115],
          ...,
          [171, 135, 118, ...,  95,  95,  85],
          [ 85,  90,  94, ...,  96,  95,  89],
          [122, 122, 126, ...,  51,  46,  69]])

[ ] # shape of input data
x.shape

[ ] (4000, 19200)

[ ] # we can see that the format is not in the right manner so we need to reshape and transpose it
x = x.reshape([-1,3,80,80])
x[0].shape

[ ] (3, 80, 80)

[ ] #taking the input data and transposing its dimensions in the right format i.e first the width and height of image.
x = x.reshape([-1,3,80,80]).transpose([0,2,3,1])
```

Fig 5.3.1.1: Input Data



Fig 5.3.1.2: Conversion to keras model

5.3.2 Output Data

According to our approach we have y as output and we are preparing it for validation. The data here is of the form 0 and 1 only indicating images with no ship in it and images with ship in it.

```
# Output data
#we can see that y is the validation data we are gonna use and it has only 1 and 0 values in it
#and here 1 indicates images with ship on it and 0 indicates images with no ship on it.
y
array([1, 1, 1, ..., 0, 0, 0])
```

Fig 5.3.2.1: Output Data

Traversing through all Images.

We have images starting from 0 to 4000 so in-order to go through all of them,

```
[ ] # shuffling all indexes so that it can take a random value in between 0 - 4000
positions = np.arange(4000)
np.random.shuffle(positions)
```

Fig 5.3.2.2: Traverse all images

5.3.3 Training Data and Scaling

```
[ ] # taking the training datas and assigning them to random values.  
    x_train = X[positions]  
    y_train = y[positions]
```

Scaling and getting the train parameters ready

```
[ ] x_train = x_train / 255
```

Fig 5.3.3.1: Training and Scaling

5.4 Building the Model

5.4.1 Convolution Neural Network

CNN is now the go-to model on every image related problem. In terms of accuracy they blow competition out of the water. It is also successfully applied to recommender systems, natural language processing and more. The main advantage of CNN compared to its predecessors is that it automatically detects the important features without any human supervision. For example, given many pictures of cats and dogs it learns distinctive features for each class by itself.

CNN is also computationally efficient. It uses special convolution and pooling operations and performs parameter sharing. This enables CNN models to run on any device, making them universally attractive.

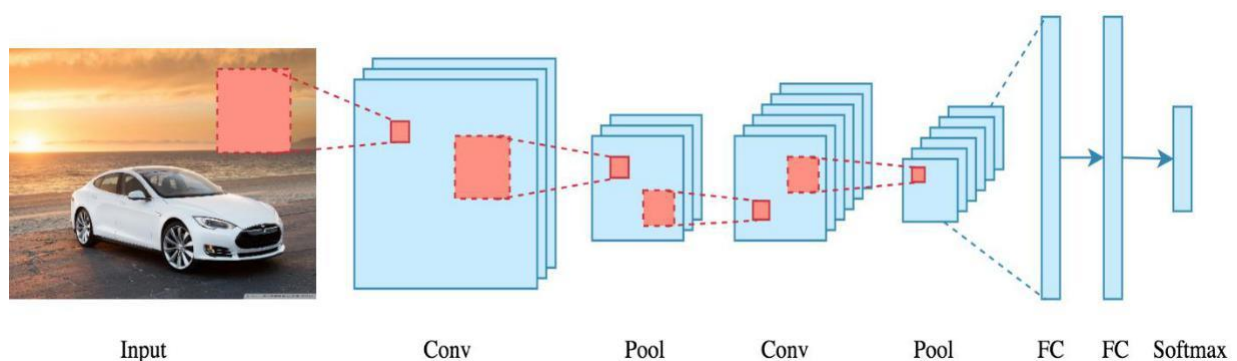


Fig 5.4.1.1: Architecture of CNN

Convolution

The main building block of CNN is the convolutional layer. Convolution is a mathematical operation to merge two sets of information. In our case the convolution is applied on the input

data using a convolution filter to produce a feature map. There are a lot of terms being used so let's visualize them one by one.

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

Input

1	0	1
0	1	0
1	0	1

Filter / Kernel

Fig 5.4.1.2: Convolution

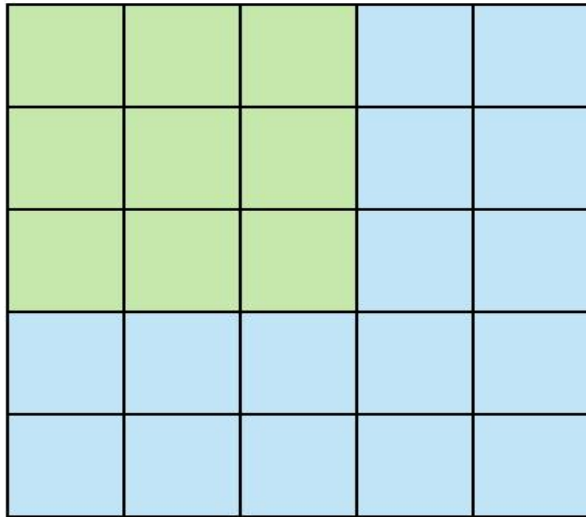
Non-Linearity

For any kind of neural network to be powerful, it needs to contain non-linearity. Both the ANN and autoencoder we saw before achieved this by passing the weighted sum of its inputs through an activation function, and CNN is no different. We again pass the result of the convolution operation through relu activation function. So, the values in the final feature maps are not actually the sums, but the relu function applied to them. We have omitted this in the figures above for simplicity. But keep in mind that any type of convolution involves a relu operation, without that the network won't achieve its true potential.

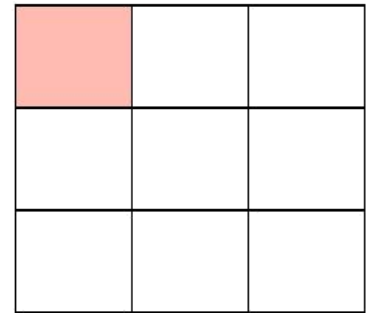
Since the data is ready, now we can build up the model. Here I am going to add 4 convolutional layers followed by 4 max-pooling layers. Then there is a Flatten layer and finally, there are 2 dense layers.

Strides

Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a stride of 2.



Stride 1



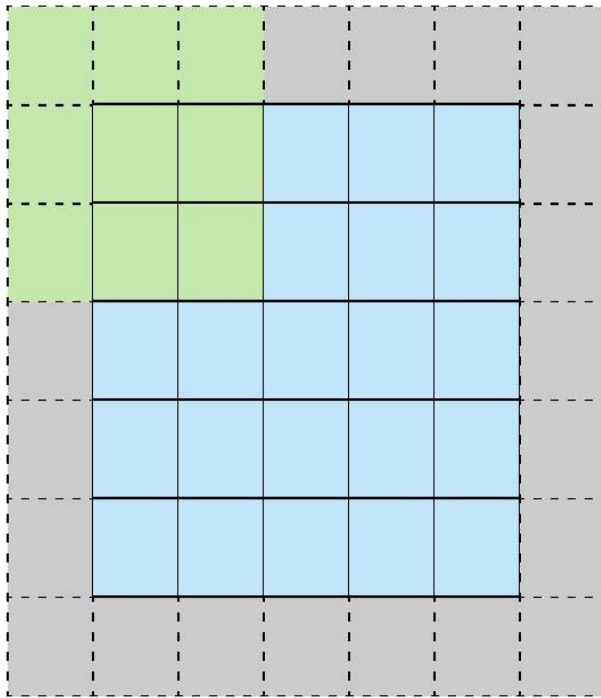
Feature Map

Fig 5.4.1.3: Strides

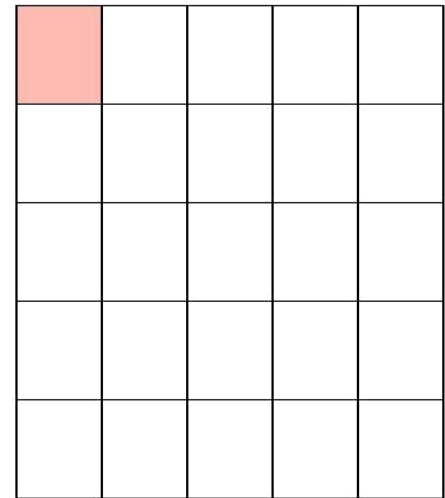
Padding

Sometimes filter does not fit perfectly fit the input image. We have two options:

- Pad the picture with zeros (zero-padding) so that it fits
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.



Stride 1 with Padding



Feature Map

Fig 5.4.1.4: Padding

Pooling

After a convolution operation we usually perform pooling to reduce the dimensionality. This enables us to reduce the number of parameters, which both shortens the training time and combats overfitting. Pooling layers down sample each feature map independently, reducing the height and width, keeping the depth intact.

The most common type of pooling is max pooling which just takes the max value in the pooling window. Contrary to the convolution operation, pooling has no parameters. It slides a window over

its input, and simply takes the max value in the window. Similar to a convolution, we specify the window size and stride.

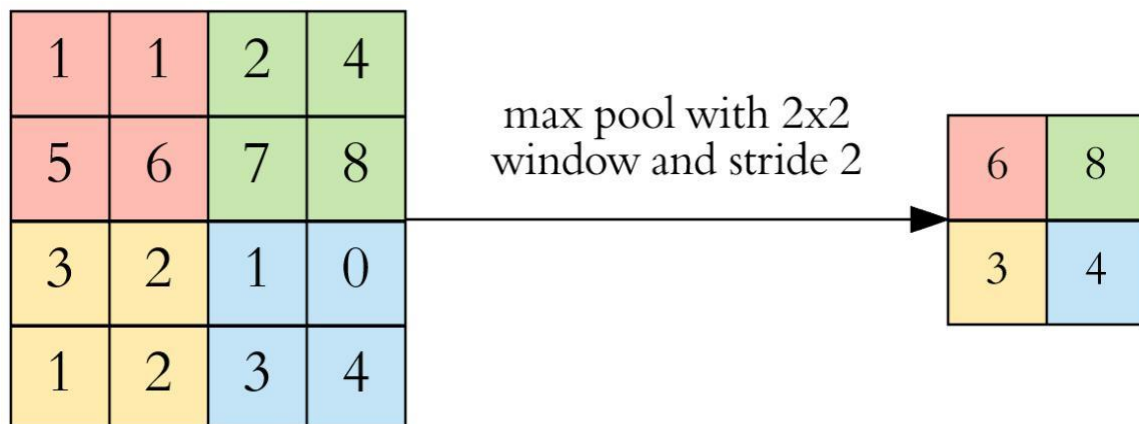


Fig 5.4.1.5: Max Pooling

Fully Connected

After the convolution + pooling layers we add a couple of fully connected layers to wrap up the CNN architecture.

Remember that the output of both convolution and pooling layers are 3D volumes, but a fully connected layer expects a 1D vector of numbers. So, we flatten the output of the final pooling layer to a vector and that becomes the input to the fully connected layer. Flattening is simply arranging the 3D volume of numbers into a 1D vector, nothing fancy happens here.

Construction

```
[ ] from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense,Conv2D,MaxPooling2D,Flatten #taking the required layers
```

Fig 5.4.1.6: Importing Required Layers

```
[125] model = Sequential()

model.add(Conv2D(16,3, padding='same', input_shape=(80,80,3), activation='relu')) # here we have 16 kernels having size 3
# For the first conv layer we have to mention the input shape i.e 80,80 and this will be taken as output and carried by other layers,activation technique is relu.
model.add(MaxPooling2D(pool_size=(2, 2))) #In max-pooling layer, I have added a 2x2 kernel so the max value will be taken when reducing the image size by 50%.

model.add(Conv2D(32,3,padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2))) #20x20

model.add(Conv2D(64,3, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2))) #10x10

model.add(Conv2D(32,3, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2))) #5x5

model.add(Flatten()) #The Flatten layer takes the output from previous max-pooling layer and converts into a 1D array so that it can be feed into the Dense layers.
model.add(Dense(512, activation='relu')) #actual learning process starts here, taking 512 neurons in the fully connected layer

model.add(Dense(1, activation='sigmoid')) #taking only 1 neuron in the final layer as we are having binary data
#and using activation as sigmoid as it doesn't have multiple classes.
model.summary() #total representation will be done
# the total parameters we get here are 453,186 and there are no non-trainable so we can proceed
```

Fig 5.4.1.7: Construction of Model

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 80, 80, 16)	448
max_pooling2d (MaxPooling2D)	(None, 40, 40, 16)	0
conv2d_1 (Conv2D)	(None, 40, 40, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 20, 20, 32)	0
conv2d_2 (Conv2D)	(None, 20, 20, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 10, 10, 64)	0
conv2d_3 (Conv2D)	(None, 10, 10, 32)	18464
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 32)	0
flatten (Flatten)	(None, 800)	0
dense (Dense)	(None, 512)	410112
dense_1 (Dense)	(None, 1)	513
Total params: 452,673		
Trainable params: 452,673		
Non-trainable params: 0		

In the first convolution layer, I have added 16 kernels which have the size of 3. Once the image is convoluted with kernel it will be passed through **relu** activation to obtain non-linearity. The input shape of this layer should be 80x80 since we resized images for that size. Since all the images are colored images, they have 3 channels for RGB.

In the max-pooling layer, I have added a 2x2 kernel such that the max value will be taken when reducing the image size by 50%.

The Flatten layer will take the output from the previous max-pooling layer and convert it to a 1D array such that it can be feed into the Dense layers. A dense layer is a regular layer of neurons in a neural network. This is where the actual learning process happens by adjusting the weights. Here we have 2 such dense layers and since this is a binary classification there is only 1 neuron in the output layer. The number of neurons in the other layer can be adjusted as a hyperparameter to obtain the best accuracy.

5.4.2 Compile the Model

```
[126] #compiling model
      model.compile(loss=tf.keras.losses.BinaryCrossentropy(),metrics=['accuracy'])
      # here it will take default optimizer i.e RMSprop and the loss method is binarycrossentropy as it has 0 and 1 and the metrics is accuracy.
```

Fig 5.4.2.1: Compiling Model

Here we need to define how to calculate the loss or error. Since we are using a binary classification, we can use **binary_crossentropy**. With the optimizer parameter, we pass how to adjust the weights in the network such that the loss gets reduced. There are many options that can be used and here I use the **RMSprop** method. Finally, the metrics parameter will be used to estimate how good our model is here we use the **accuracy**.

RMSprop Optimization

The RMSprop optimizer is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster. The difference between RMSprop and gradient descent is on how the gradients are calculated. The following equations show how the gradients are calculated for the RMSprop and gradient descent with momentum. The value of momentum is denoted by beta and is usually set to 0.9. If you are not interested in the math behind the optimizer, you can just skip the following equations.

Binary_crossentropy

Binary crossentropy is a loss function that is used in binary classification tasks. These are tasks that answer a question with only two choices (yes or no, A or B, 0 or 1, left or right).

Sigmoid is the only activation function compatible with the binary crossentropy loss function.

Accuracy

This metric creates two local variables, total and count that are used to compute the frequency with which `y_pred` matches `y_true`. This frequency is ultimately returned as binary accuracy: an idempotent operation that simply divides total by count.

If `sample_weight` is `None`, weights default to 1. Use `sample_weight` of 0 to mask values.

Arguments

- **name:** (Optional) string name of the metric instance.
- **dtype:** (Optional) data type of the metric result.

5.4.3 Training the Model

```
[127] # fit
history_model.fit(X_train,y_train,epochs=20,batch_size=32,validation_split=0.2)
# we have split the train and validation as 80 and 20 according to validation_split.

Epoch 1/20
100/100 [=====] - 1s 9ms/step - loss: 0.3074 - accuracy: 0.8703 - val_loss: 0.3341 - val_accuracy: 0.8650
Epoch 2/20
100/100 [=====] - 1s 7ms/step - loss: 0.1697 - accuracy: 0.9303 - val_loss: 0.0975 - val_accuracy: 0.9638
Epoch 3/20
100/100 [=====] - 1s 7ms/step - loss: 0.0970 - accuracy: 0.9634 - val_loss: 0.0553 - val_accuracy: 0.9800
Epoch 4/20
100/100 [=====] - 1s 7ms/step - loss: 0.0607 - accuracy: 0.9812 - val_loss: 0.0507 - val_accuracy: 0.9825
Epoch 5/20
100/100 [=====] - 1s 8ms/step - loss: 0.0457 - accuracy: 0.9853 - val_loss: 0.0479 - val_accuracy: 0.9850
Epoch 6/20
100/100 [=====] - 1s 7ms/step - loss: 0.0387 - accuracy: 0.9869 - val_loss: 0.0441 - val_accuracy: 0.9837
Epoch 7/20
100/100 [=====] - 1s 7ms/step - loss: 0.0327 - accuracy: 0.9906 - val_loss: 0.0490 - val_accuracy: 0.9837
Epoch 8/20
100/100 [=====] - 1s 7ms/step - loss: 0.0172 - accuracy: 0.9953 - val_loss: 0.0376 - val_accuracy: 0.9912
Epoch 9/20
100/100 [=====] - 1s 7ms/step - loss: 0.0257 - accuracy: 0.9937 - val_loss: 0.0742 - val_accuracy: 0.9725
Epoch 10/20
100/100 [=====] - 1s 8ms/step - loss: 0.0196 - accuracy: 0.9953 - val_loss: 0.0365 - val_accuracy: 0.9925
Epoch 11/20
100/100 [=====] - 1s 7ms/step - loss: 0.0136 - accuracy: 0.9947 - val_loss: 0.0351 - val_accuracy: 0.9925
Epoch 12/20
100/100 [=====] - 1s 7ms/step - loss: 0.0115 - accuracy: 0.9953 - val_loss: 0.0496 - val_accuracy: 0.9925
Epoch 13/20
100/100 [=====] - 1s 8ms/step - loss: 0.0106 - accuracy: 0.9969 - val_loss: 0.0397 - val_accuracy: 0.9912
Epoch 14/20
100/100 [=====] - 1s 8ms/step - loss: 0.0098 - accuracy: 0.9969 - val_loss: 0.0616 - val_accuracy: 0.9900
Epoch 15/20
100/100 [=====] - 1s 7ms/step - loss: 0.0091 - accuracy: 0.9972 - val_loss: 0.0335 - val_accuracy: 0.9925
Epoch 16/20
100/100 [=====] - 1s 8ms/step - loss: 0.0223 - accuracy: 0.9966 - val_loss: 0.0843 - val_accuracy: 0.9837
Epoch 17/20
100/100 [=====] - 1s 7ms/step - loss: 0.0051 - accuracy: 0.9981 - val_loss: 0.0578 - val_accuracy: 0.9925
Epoch 18/20
100/100 [=====] - 1s 8ms/step - loss: 0.0067 - accuracy: 0.9972 - val_loss: 0.0580 - val_accuracy: 0.9925
Epoch 19/20
100/100 [=====] - 1s 8ms/step - loss: 0.0314 - accuracy: 0.9978 - val_loss: 0.0519 - val_accuracy: 0.9937
Epoch 20/20
100/100 [=====] - 1s 7ms/step - loss: 0.0162 - accuracy: 0.9969 - val_loss: 0.0367 - val_accuracy: 0.9937
```

Fig 5.4.3.1: Training the Model

Here we are passing the train and validation data i.e. X_train, y_train we used to load our data, 32 batch size to cover all training and validation images and we are splitting the train and validation using validation_split i.e. 80-Training and 20-validation, with 20 epochs.

5.4.4 Accuracy and Loss variation

```
# checking the accuracy and loss variation in terms of training and validation data
# determining whether it is an overfitting or underfitting model.
train_accuracy=history.history['accuracy']
val_acc=history.history['val_accuracy']
epochees=list(range(1,21))
plt.plot(epochees,train_accuracy,label='train_acc')
plt.plot(epochees,val_acc,label='val_acc')
plt.legend()
plt.title("Accuracy of Model", fontsize=20)
plt.xlabel("epochees", fontsize=18)
plt.ylabel("train,val",fontsize=18)
```

```
[37] train_loss=history.history['loss']
val_loss=history.history['val_loss']
plt.plot(epochees,train_loss,label='train_loss')
plt.plot(epochees,val_loss,label='val_loss')
plt.title('loss')
plt.legend()
plt.title("Loss of Model", fontsize=20)
plt.xlabel("epochees", fontsize=18)
plt.ylabel("train,val",fontsize=18)
```

Fig 5.4.4.1: Accuracy & Loss Variation

Nature of Model

Based on the variation between accuracy and loss values we can judge the nature of model, if accuracy values of train is greater than and has a lot of variation than validation then, it is overfitting. If accuracy values of validation is more and has a lot of variation than train then, it is underfitting. If they overlap with each other then it is precise. Our model is precise.

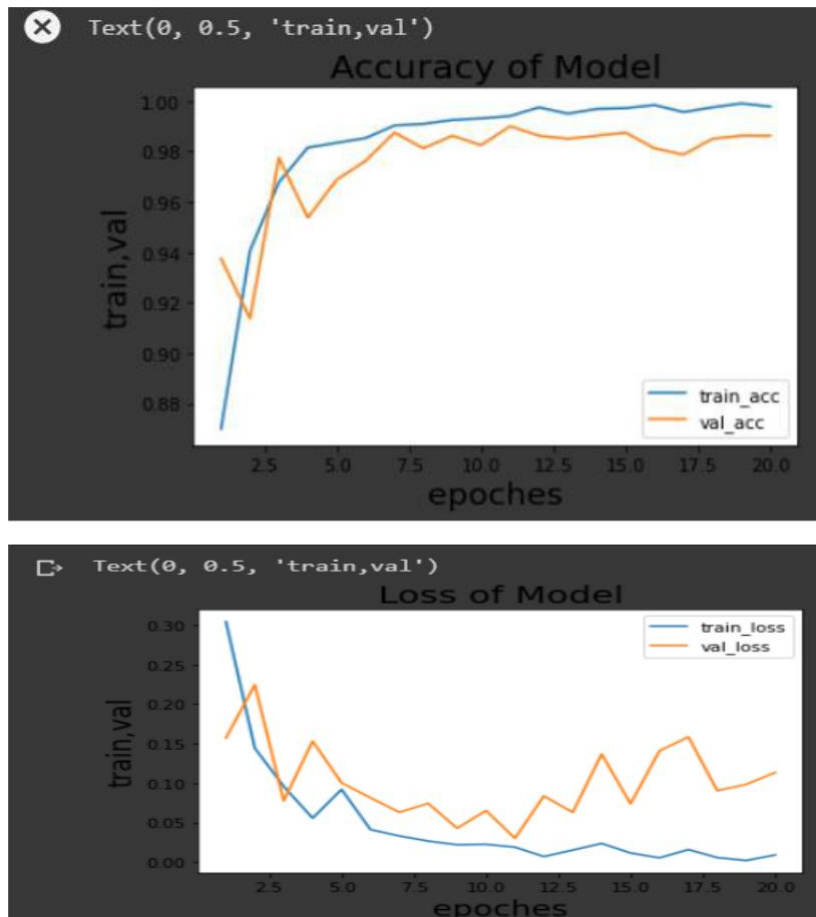


Fig 5.4.4.2: Graphs of train_acc, val_acc & train_loss, val_loss

5.5 Prediction of ships in satellite imagery

Prediction of image involves 4 steps, i.e.

Prediction of Image involves 4 steps

1. Read the image
2. Check the shape
3. Resize into required shape
4. Apply Scaling

Fig 5.5.1: Steps in Prediction of image


```
[ ] from tensorflow.keras.preprocessing import image # importing image to predict whether there is a ship or not
import numpy as np
img=image.load_img('/content/drive/My Drive/Project DSPS/shipsnet/imgsh/1_20170830_181003_0f4e_-122.35157982161317_37.78327205920199.png')
# loading an image and check whether there is a ship or not.
print(type(img)) # checking the type of array
img=tf.keras.preprocessing.image.img_to_array(img) #converting img to array format
print(img.shape)
print(type(img))
img=tf.image.resize(img,(80,80)) # resize into required shape (1,80,80,3)
img=img/255 # scaling img
print(img.shape)
img=np.expand_dims(img,axis=0) # to get (1,80,80,3) as image dimensions
print(img.shape)

[ ] <class 'PIL.PngImagePlugin.PngImageFile'>
(80, 80, 3)
<class 'numpy.ndarray'>
(80, 80, 3)
(1, 80, 80, 3)
```

Fig 5.5.2: Procedure

Firstly, we import image from TensorFlow, then we load the image and paste the URL of image we are predicting. After loading we convert the image to array format and then check the shape and type of image. Now we will be using this during resizing the image we are predicting on to convert it into required shape. Next, we will be applying scaling on the image and to get the dimensions right, expand dims enters the process, finally we check the shape and observe that we got the desired result.

```
[ ] model.predict(img) ## here we are predicting the image which has ship in it, should give result close to 1 or one.

[ ] array([[1.]], dtype=float32)
```

Fig 5.5.3: Output after Predicting image

Now, the main step of predicting the image whether it has ships or not, if it gives an array having value nearest to one then it has ship in it, else if the value close to 0 then it has no ship in it.

CONCLUSION: -

Now, we can conclude that using CNN algorithm, we have traversed through the shipsnet folder and have been able to differentiate images having ship and not having ship. Though, we have an imbalanced data i.e. images with ship on them are 3000 and images with ship are 1000 yet we have been able to be successful using metrics as accuracy, we have got validation accuracy equal to 99.37 and train accuracy as 99.69 and after plotting graph b/w accuracy we can observe that our model is precise. With the help of optimizer as RMSProp, and loss as BinaryCrossEntropy, we have compiled our model. By looking at the above value after predicting we can see that it gives an array having value 1., which is closest to 1 and according to our approach, if we get an array having value closest to 1 then it has ship in it. So, this satellite image has ship in it and we are successful in using the approach of CNN.

REFERENCES: -

<https://www.kaggle.com/rhammell/ships-in-satellite-imagery/home>

<https://medium.com/breathe-publication/top-15-deep-learning-applications-that-will-rule-the-world-in-2018-and-beyond-7c6130c43b01>

<https://github.com/SathvikAdiraju/Project-Ships-DSPS>