DEEP LEARNING

(Prediction of Ships from Satellite Images)

Summer Internship Report Submitted in partial fulfillment

of the requirement for undergraduate degree of

Bachelor of Technology

In

COMPUTER SCIENCE AND ENGINEERING

By

Dheeraj Velury

221710308061

Under the Guidance of

Assistant Professor

GITAM

(DEEMED TO BE UNIVERSITY)

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 industrial training work entitled **“Prediction of Ships from Satellite Images**” to GITAM (Deemed To Be University), Hyderabad in partial fulfillment 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 **Mr.**, 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   Dheeraj Velury

Date: 221710308061

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled **"Prediction of Ships from Satellite Images"** is being submitted by Dheeraj Velury (221710308061) in partial fulfillment 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 2019-20

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.

Dr.S.Phani Kumar

Assistant Professor	Professor and HOD
Department of CSE	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 competition of this internship.

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

I would like to thank Dr.S.Phani Kumar, 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 **Mr.** 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.

Dheeraj Velury

221710308061

ABSTRACT

Machine learning algorithms are used to predict the images from the data set by splitting the data set into train, test and build. Machine learning algorithms models of higher accuracy to predict the images is the primary task to be performed on Shipsnet data set. My perception of understanding the given data set has been in the view of undertaking a client's requirement of predicting if the image is a ship or not.

To get a better understanding and work on a strategical approach for solution of the client, I have adapted the view point of looking at various satellite imagery and for further deep understanding of the problem, I have taken the stance of a port authority officer and reasoned out the various factors of choice of the images, and my primary objective of this case study was to monitor the ships traffic in San Francisco port and predict if it is a ship or not to determine the port activity levels and supply chain analysis.

Table of Contents:

LIST OF FIGURES:

1. Machine Learning

1.1. Introduction:

Machine Learning (ML) is coming into its own, with a growing recognition that ML can play a key role in a wide range of critical applications, such as data mining, natural language processing, image recognition, and expert systems. ML provides potential solutions in all these domains and more and is set to be a pillar of our future civilization.

Over the last few decades ML has become one of the mainstays of information technology and with that, a rather central, albeit usually hidden, part of our life. With the ever analysis it will become even more pervasive as necessary ingredient for technological progress.

Although machine learning is a field within computer science, it differs from traditional computational approaches. In traditional computing, algorithms are sets of explicitly programmed instructions used by computers to calculate or problem solving. Machine learning algorithms instead allow for computers to train on data inputs and use statistical analysis in order to output values that fall within a specific range. Because of this, machine learning facilitates computers in building models from sample data in order to automate decision-making processes based on data inputs.

Any technology user today has benefitted from machine learning. Facial recognition technology allows social media platforms to help users tag and share photos of friends. Optical character recognition (OCR) technology converts images of text into movable type. Recommendation engines, powered by machine learning, suggest what movies or television shows to watch next based on user preferences. Self-driving cars that rely on machine learning to navigate may soon be available to consumers.

Machine learning is a continuously developing field. Because of this, there are some considerations to keep in mind as you work with machine learning methodologies or analyze the impact of machine learning processes.

1.2. Importance of Machine Learning:

Gone are the days when programmers would tell a machine how to solve a problem at hand. We are in the era of machine learning where machines are left to solve problems, on their own, by identifying the patterns in each data set. Analyzing hidden trends and patterns makes it easy to predict future problems and prevent them from occurring.

A machine learning algorithm usually follows a certain type of data and then uses the patterns hidden in that data to answer more questions. For example, showing a computer a series of photographs, some of which say that "this is a horse" and some of which say, "this is not a horse." After this exercise, if you show some more photographs to the same computer, it will be on a mission to identify which of those photographs are of a horse and which of those are not that of a horse. Every correct and incorrect guess of the computer is added to its memory, which makes it smarter in the longer run and enriches its learning over a period.

Because of machine learnings capabilities to learn the data and produce accurate results has made it one of the most important skills men can learn. And because of these capabilities ML is used in various fields.

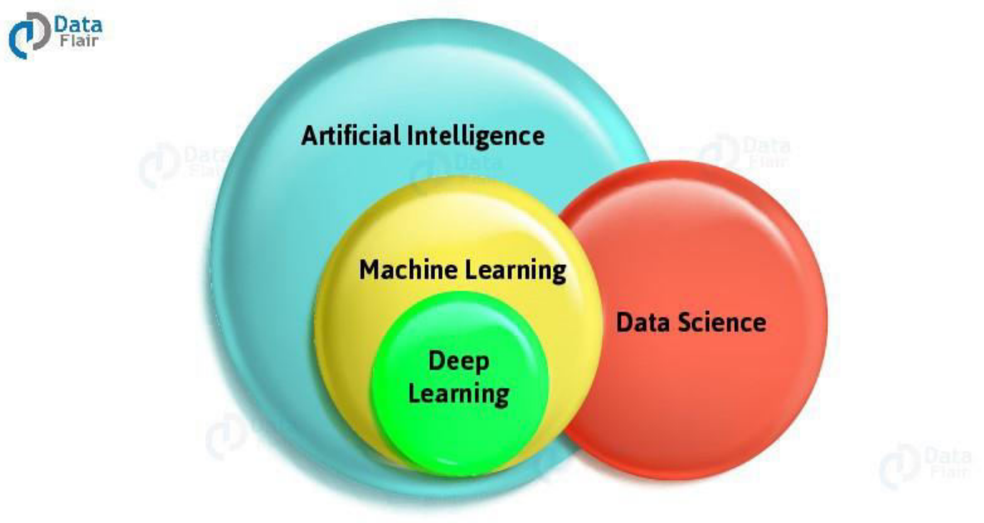


Fig.1.2.1 ML usage in various fields.

The idea is quite simple. We are not targeting to understand the underlying processes that help us learn. We write computer programs that will make machines learn and enable them to perform tasks, such as pattern recognition, prediction. The goal of learning is to construct a model that takes the input and produces desired results. Sometimes, we can understand the model and other times it is like a black box for us, the working of which cannot be explained. The model can be considered as an approximation of the process we want machines to mimic. In a situation, it is possible that we obtain errors for some input, but most of the time the model provides correct answers. Hence, another measure if performance of a machine learning algorithm will be based on accuracy of the result.

1.3. Uses of Machine Learning

- **Virtual Personal Assistants**

Siri, Alexa, Google now are some of the popular examples of virtual personal assistants. As the name suggests, they assist in finding information, when asked over voice. All you need to do is activate them and ask, “What is my schedule for today?”, “What are the flights from Germany to London”, or similar questions. For answering, your personal assistant looks out for the information, recalls your related queries, or send a command to other resources (like phone apps) to collect info. You can even instruct assistants for certain tasks like “Set an alarm for 6 AM next morning”, “Remind me to visit Visa Office day after tomorrow”.

Machine learning is an important part of these personal assistants as they collect and refine the information on the basis of your previous involvement with them. Later, this set of data is utilized to render results that are tailored to your preferences.

Virtual Assistants are integrated to a variety of platforms. For example:

- Smart Speakers: Amazon Echo and Google Home

Predictions while Commuting

- **Traffic Predictions:**

We all have been using GPS navigation services. While we do that, our current locations and velocities are being saved at a central server for managing traffic. This data is then used to build a map of current traffic. While this helps in preventing the traffic and does congestion analysis, the underlying problem is that there are a smaller number of cars that are equipped with GPS. Machine learning in such scenarios helps to estimate the regions where congestion can be found on the basis of daily experiences.

- **Online Transportation Networks:**

When booking a cab, the app estimates the price of the ride. When sharing these services, how do they minimize the detours? The answer is machine learning. Jeff Schneider, the engineering lead at Uber ATC reveals in an interview that they use ML to define price surge hours by predicting the rider demand. In the entire cycle of the services, ML is playing a major role.

- **Video Surveillance:**

Imagine a single person monitoring multiple video cameras! Certainly, a difficult job to do and boring as well. This is why the idea of training computers to do this job makes sense.

The video surveillance system nowadays is powered by AI that makes it possible to detect crime before they happen. They track unusual behavior of people like standing motionless for a long time, stumbling, or napping on benches etc. The system can thus give an alert to human attendants, which can ultimately help to avoid mishaps. And when such activities are reported and counted to be true, they help to improve the surveillance services. This happens with machine learning doing its job at the backend.

- **Social Media Services:**

From personalizing your news feed to better ads targeting, social media platforms are utilizing machine learning for their own and user benefits. Here are a few examples that you must be noticing, using, and loving in your social media accounts, without realizing that these wonderful features are nothing but the applications of ML.

- **People You May Know:**

Machine learning works on a simple concept: understanding with experiences. Facebook continuously notices the friends that you connect with, the profiles that you visit very often, your interests, workplace, or a group that you share with someone etc. On the basis of continuous learning, a list of Facebook users are suggested that you can become friends with.

- **Face Recognition:**

You upload a picture of you with a friend and Facebook instantly recognizes that friend. Facebook checks the poses and projections in the picture, notice the unique features, and then match them with the people in your friend list. The entire process at the backend is complicated and takes care of the precision factor but seems to be a simple application of ML at the front end.

- **Similar Pins:**

Machine learning is the core element of Computer Vision, which is a technique to extract useful information from images and videos. Pinterest uses computer vision to identify the objects (or pins) in the images and recommend similar pins accordingly.

- **Email Spam and Malware Filtering:**

There are a number of spam filtering approaches that email clients use. To ascertain that these spam filters are continuously updated, they are powered by machine learning. When rule-based spam filtering is done, it fails to track the latest tricks adopted by spammers. Multi-Layer Perceptron, C 4.5 Decision Tree Induction are some of the spam filtering techniques that are powered by ML.

Over 325,000 malwares are detected every day and each piece of code is 90–98% similar to its previous versions. The system security programs that are powered by machine learning understand the coding pattern. Therefore, they detect new malware with 2–10% variation easily and offer protection against them.

- **Online Customer Support:**

A number of websites nowadays offer the option to chat with customer support representative while they are navigating within the site. However, not every website has a live executive to answer your queries. In most of the cases, you talk to a chatbot. These bots tend to extract information from the website and present it to the customers. Meanwhile, the chatbots advance with time. They tend to understand the user queries better and serve them with better answers, which is possible due to its machine learning algorithms.

- **Search Engine Result Refining:**

Google and other search engines use machine learning to improve the search results for you. Every time you execute a search, the algorithms at the backend keep a watch at how you respond to the results. If you open the top results and stay on the web page for long, the search engine assumes that the results it displayed were in accordance to the query. Similarly, if you reach the second or third page of the search results but do not open any of the results, the search engine estimates that the results served did not match requirement. This way, the algorithms working at the backend improve the search results.

- **Product Recommendations:**

You shopped for a product online few days back and then you keep receiving emails for shopping suggestions. If not this, then you might have noticed that the shopping website or the app recommends you some items that somehow matches with your taste. Certainly, this refines the shopping experience, but did you know that its machine learning doing the magic for you? On the basis of your behavior with the website/app, past purchases, items liked or added to cart, brand preferences etc., the product recommendations are made.

- **Online Fraud Detection:**

Machine learning is proving its potential to make cyberspace a secure place and tracking monetary frauds online is one of its examples. For example: PayPal is using ML for protection against money laundering. The company uses a set of tools that help them to compare millions of transactions taking place and distinguish between legitimate or illegitimate transactions taking place between the buyers and sellers.

1.4. Types of Machine Learning:

There are 3 major types in machine learning which are widely used in today's world.

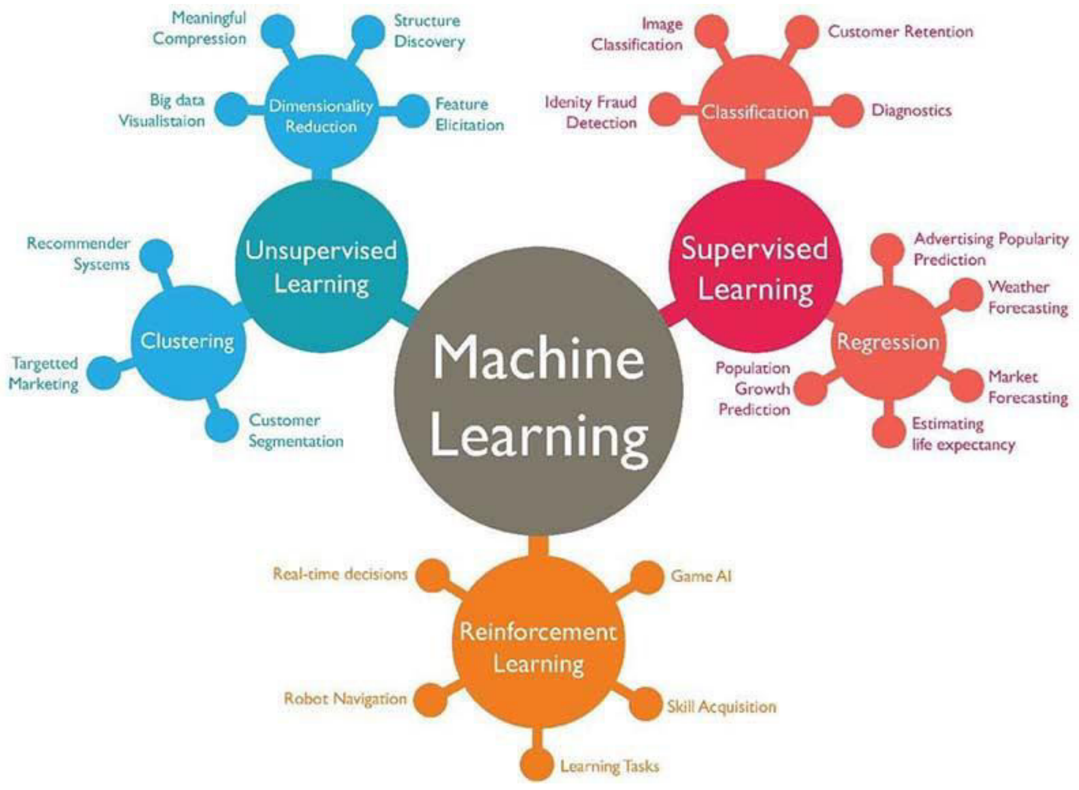


Fig 1.4.0 Various types of ML

1.4.1. Supervised Learning:

Supervised learning is one of the most basic types of machine learning. In this type machine learning algorithm is trained on labeled data, even though data needs to be labeled accurately for this method to work, supervised learning is extremely powerful. Here data is labeled which means that each data is tagged with correct label.

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

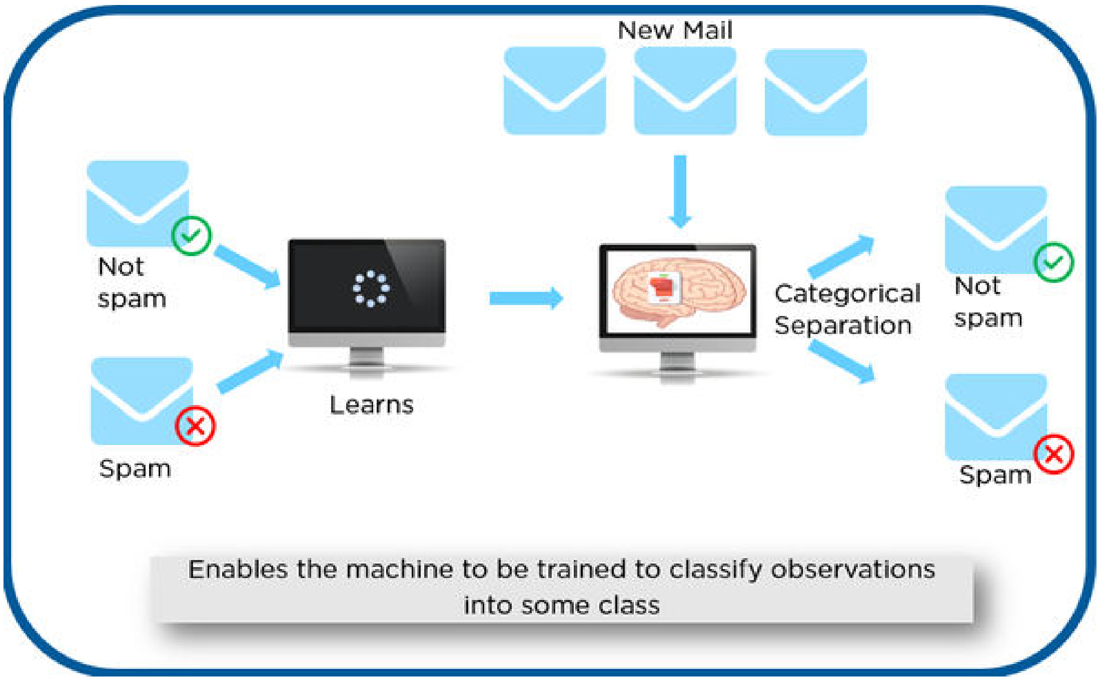

New Mail
Not spam
Spam
Learns
Categorical Separation
Not spam
Spam
Enables the machine to be trained to classify observations into some class

Fig 1.4.1 Supervised learning

As shown in the above example, we have initially taken some data and marked them as ‘Spam’ or ‘Not Spam’. This labeled data is used by the training supervised model.

Once it is trained, we can test our model by testing it with some new mails and checking if the model is able to predict the right output.

Types of Supervised Learning:

- **Classification**: A classification problem is when the output variable is a category, such as red or blue or disease and no disease.

- **Regression**: A regression problem is when the output variable is a real value, such as weight or rupees.

1.4.2. Unsupervised Learning:

In unsupervised learning, data is unlabeled, so the learning algorithm is left to find commonalities among its input data. As unlabeled data are more abundant than labeled data, machine learning methods that facilitate unsupervised learning are particularly valuable. The goal of unsupervised learning may be as straightforward as discovering hidden patterns within a dataset, but it may also have a goal of feature learning, which allows the computational machine to automatically discover the representations that are needed to classify raw data.

Unsupervised learning is commonly used for transactional data. You may have a large dataset of customers and their purchases, but as a human you will likely not be able to make sense of what similar attributes can be drawn from customer profiles and their types of purchases. With this data fed into an unsupervised learning algorithm, it may be determined that women of a certain age range who buy unscented soaps are likely to be pregnant, and therefore a marketing campaign related to pregnancy and baby products can be targeted to this audience in order to increase their number of purchases.

Without being told a “correct” answer, unsupervised learning methods can look at complex data that is more expansive and seemingly unrelated in order to organize it in potentially meaningful ways. Unsupervised learning is often used for anomaly detection including fraudulent credit card purchases, and recommender systems that suggest what products to buy next. In unsupervised learning, untagged photos of dogs can be used as input data for the algorithm to find likenesses and classify dog photos together.

Types of Unsupervised Learning:

- **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- **Association**: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

1.4.3. Reinforcement Learning:

It directly takes inspiration from how human beings learn from data in their lives. It features an algorithm that improves upon itself and learns from new situations using a trial-and-error method. Favorable outputs are encouraged, or ‘reinforced’, and non-favorable outputs are discouraged or ‘punished’. Based on the psychological concept of conditioning, reinforcement learning works by putting the algorithm in a work environment with an interpreter and a reward system. In every iteration of the algorithm, the output result is given to the interpreter, which decides whether the outcome is favorable or not. In case of the program finding the correct solution, the interpreter reinforces the solution by providing a reward to the algorithm. If the outcome is not favorable, the algorithm is forced to reiterate until it finds a better result. In most cases, the reward system is directly tied to the effectiveness of the result. In typical reinforcement learning use-cases, such as finding the shortest route between two points on a map, the solution is not an absolute value. Instead, it takes on a score of effectiveness, expressed in a percentage value. The higher this percentage value is, the more reward is given to the algorithm. Thus, the program is trained to give best possible solution for the best possible reward.

2. Deep Learning

2.1. Deep Learning and its importance:

Deep learning algorithms run data through several “layers” of neural network algorithms, each of which passes a simplified representation of the data to the next layer. Most machine learning algorithms work well on datasets that have up to a few hundred features, or columns. 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 other projects such as human activity recognition , when we apply traditional machine learning algorithm like K-NN then we have to separately detect human and its activity also have to select impactful input parameters manually which becomes a very tedious task as data set is way too complex but the complexity dramatically reduces on applying artificial neural network, such is the power of deep learning. Yes, it is correct that deep learning algorithms take lot of time for training sometimes even weeks, but its execution on new data is so fast that it is not even comparable with traditional ML algorithms. Deep learning has enabled Industrial Experts to overcome challenges which were impossible, a decade ago like Speech and Image recognition and Natural Language Processing. Majority of the Industries are currently depending on it, be it 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.

2.2. Uses of Deep Learning:

i. **Translations**: Although automatic machine translation is not new, deep learning is helping enhance automatic translation of text by using stacked networks of neural networks and allowing translations from images.

ii**. Adding color to black-and-white images and videos**: It used to be a time-consuming process where humans had to add color to black-and-white images and videos by hand can now be automatically done with deep-learning models.

iii. **Language recognition**: Deep learning machines are beginning to differentiate dialects of a language. A machine recoginses that someone is speaking English and then engages an AI that is learning to tell the differences between dialects. Once the dialect is determined, another AI will step in that specializes in that particular dialect. All of this happens without involvement from a human.

iv. **Autonomous vehicles**: There's not just one AI model at work as an autonomous vehicle drives down the street. Some deep-learning models specialize in streets signs while others are trained to recognize pedestrians. As a car navigates down the road, it can be informed by up to millions of individual AI models that allow the car to act.

v. **Computer vision**: Deep learning has delivered super-human accuracy for image classification, object detection, image restoration and image segmentation—even handwritten digits can be recognized. Deep learning using enormous neural networks is teaching machines to automate the tasks performed by human visual systems.

vi. **Text generation**: The machines learn the punctuation, grammar and style of a piece of text and can use the model, it is developed to automatically create an entirely new text with the proper spellings, grammar and style of the example text. Everything from Shakespeare to Wikipedia entries have been created.

vii. **Deep-learning robots**: Deep-learning applications for robots are plentiful and powerful from an impressive deep-learning system that can teach a robot just by observing the actions of a human completing a task to a housekeeping robot that's provided with input from several other AIs in order to take action. Just like how a human brain processes input from past experiences, current input from senses and any additional data that is provided, deep-learning models will help robots execute tasks based on the input of many different AI opinions.

2.3. Relation between Data Mining, Machine Learning and Deep Learning:

The deep learning, data mining and machine learning share a foundation in data science, and there certainly is overlap between the two. Data mining can use machine learning algorithms to improve the accuracy and depth of analysis, and vice-versa; machine learning can use mined data as its foundation, refining the dataset to achieve better results. You could also argue that data mining and machine learning are similar in that they both seek to address the question of how we can learn from data. However, the way in which they achieve this end, and their applications, form the basis of some significant differences

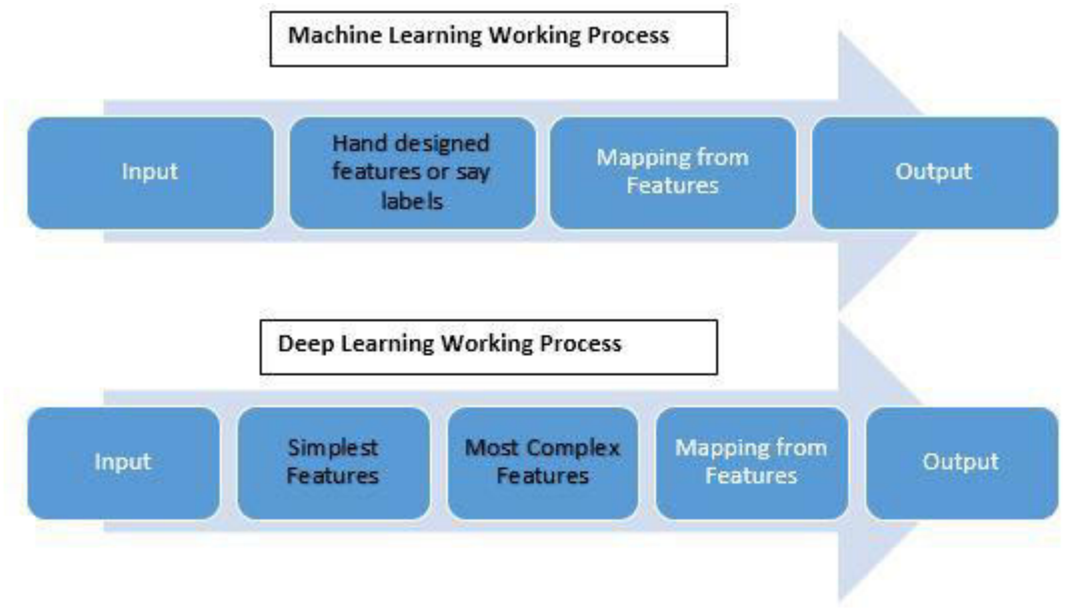


Fig 2.3.1 Deep learning machine learning working process.

Machine Learning comprises of the ability of the machine to learn from trained data set and predict the outcome automatically. It is a subset of artificial intelligence. Deep

Learning is a subset of machine learning. It works in the same way on the machine just like how the human brain processes information. Like a brain can identify the patterns by comparing it with previously memorized patterns, deep learning also uses this concept. Deep learning can automatically find out the attributes from raw data while machine learning selects these features manually which further needs processing. It also employs artificial neural networks with many hidden layers, big data, and high computer resources. Data Mining is a process of discovering hidden patterns and rules from the existing data. It uses relatively simple rules such as association, correlation rules for the decision-making process, etc. Deep Learning is used for complex problem processing such as voice recognition etc. It uses Artificial Neural Networks with many hidden layers for processing. At times data mining also uses deep learning algorithms for processing the data.

3. PYTHON

3.1. Introduction:

Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code. Python is a programming language that lets you work quickly and integrate systems more efficiently. Python is dynamically typed, and garbage collected. It supports multiple programming paradigms, including structured, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuseability. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed.

3.2. Setup of Python:

- Python distribution is available for a wide variety of platforms. You need to download only the binary code applicable for your platform and install Python.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python https://www.python.org/

3.2.1. Installation:

To start, go to python.org/downloads and then click on the button to download the latest version of Python

● We can download python IDLE in windows, mac and Linux operating systems.

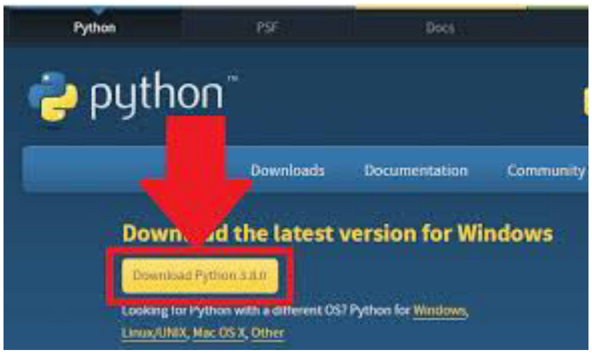
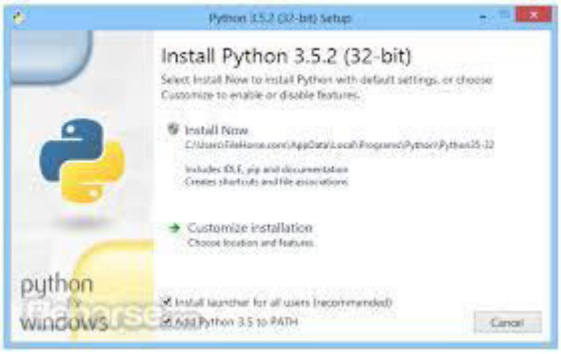

Python
PSF
Docs
python
Downloads
Documentation
Community
Download the latest version for Windows
Download Python 3.8.0
Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Fig 3.2.1 Python download

● Run the .exe file that you just downloaded and start the installation of Python by clicking on Install Now

● We can give environmental variable i.e. path after completion of downloading Fig 3.2.1.1 python installation


Python 3.5.2 (32-bit) Setup

Install Python 3.5.2 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\FileHorse.com\AppData\Local\Programs\Python\Python35-32

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.5 to PATH

Fig 3.2.1.1 Python installation

● When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.

3.2.2. Python Installing 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. Anaconda for Windows installation:

i. Go to the following link: Anaconda.com/downloads.

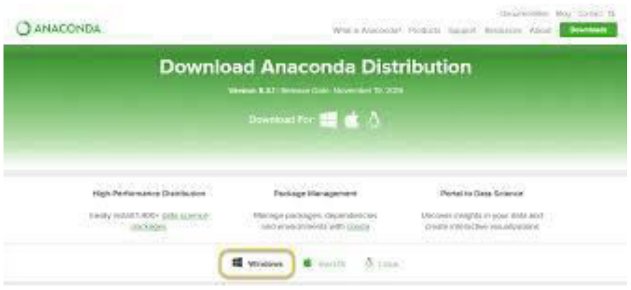
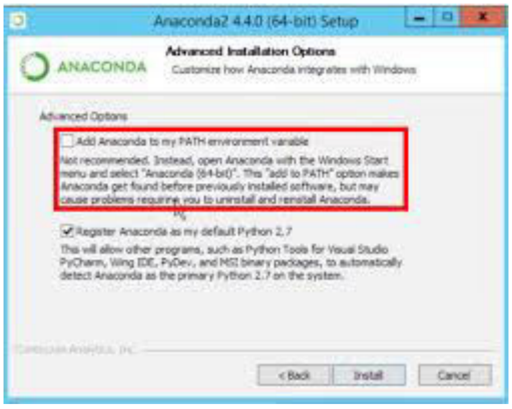


Fig3.2.2Anaconda installation

ii. Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer)

iii. Select path (i.e. add anaconda to path & register anaconda as default python 3.4)



Anaconda2 4.4.0 (64-bit) Setup

Advanced Installation Options

Customize how Anaconda integrates with Windows

Advanced Options

☐ Add Anaconda to my PATH environment variable

Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.

☑ Register Anaconda as my default Python 2.7

This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 2.7 on the system.

Fig 3.2.2.1 Anaconda path

iv. Click finish.

v. Open jupyter notebook.

3.3. Features:

- **Readable:** Python is a very readable language.
- **Easy to Learn:** Learning python is easy as this is an expressive and high level programming language, which means it is easy to understand the language and thus easy to learn.
- **Cross platform:** Python is available and can run on various operating systems such as

Mac, Windows, Linux, Unix etc. This makes it a cross platform and portable language.

- **Open Source:** Python is an open source programming language.
- **Large standard library:** Python comes with a large standard library that has some handy codes and functions which we can use while writing code in Python.
- **Free:** Python is free to download and use. This means you can download it for free and use it in your application. Python is an example of a FLOSS (Free/Libre Open Source Software), which means you can freely distribute copies of this software, read its source code and modify it.
- **Supports exception handling:** If you are new, you may wonder what is an exception? An exception is an event that can occur during program execution and can disrupt the normal flow of program. Python supports exception handling which means we can write less error prone code and can test various scenarios that can cause an exception later on.
- **Advanced features:** Supports generators and list comprehensions.
- **Automatic memory management:** Python supports automatic memory management which means the memory is cleared and freed automatically. You do not have to bother clearing the memory.

3.4. 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. Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Python has five standard data types –

- Numbers
- Strings
- Lists
- Tuples
- Dictionary

3.4.1. Python Numbers:

Number data types store numeric values. They are immutable data types, means that changing the value of a number data type results in a newly allocated object. Python supports four different numerical types −

- int (signed integers) − They are often called just integers or int, are positive or negative whole numbers with no decimal point.
- long (long integers) − Also called longs, they are integers of unlimited size, written like integers and followed by an uppercase or lowercase L.
- float (floating point real values) − Also called floats, they represent real numbers and are written with a decimal point dividing the integer and fractional parts. Floats may also be in scientific notation, with E or e indicating the power of 10 (2.5e2 = 2.5 x 102 = 250).

3.4.2. Python Strings:

In Python, Strings can be created by simply enclosing characters in quotes. Python does not support character types. These are treated as length-one strings and are also considered as substrings. Substrings are immutable and cannot be changed once created. Strings are the ordered blocks of text that are enclosed in single or double quotations. Thus, whatever is written in quotes, is considered as string. Though it can be written in single or double quotations, double quotation marks allow the user to extend strings over multiple lines without backslashes, which is usually the signal of continuation of an expression, e.g., ‘abc’, “ABC”.

3.4.3. Python Lists:

List is a collection data type in python. It is ordered and allows duplicate entries as well. Lists in python need not be homogeneous, which means it can contain different data types like integers, strings and other collection data types. It is mutable in nature and allows indexing to access the members in a list.

- To declare a list, we use the square brackets.
- List is like any other array that we declare in other programming languages. Lists in python are often used to implement stacks and queues. The lists are mutable in nature.

Therefore, the values can be changed even after a list is declared.

3.4.4. Python Tuples:

A tuple is a collection of objects which ordered and immutable. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets. Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also.

3.4.5. Python Dictionary:

It is a collection data type just like a list or a set, but there are certain features that make python dictionary unique. A dictionary in python is not ordered and is changeable as well. We can make changes in a dictionary unlike sets or strings which are immutable in nature. Dictionary contains key-value pairs like a map that we have in other programming languages. A dictionary has indexes. Since the value of the keys we declare in a dictionary are always unique, we can use them as indexes to access the elements in a dictionary.

3.5 Functions:

3.5.1. Defining a function:

- Function blocks begin with the keyword def followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or docstring.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

3.5.2. 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.

3.6. OOPs Concepts:

3.6.1 Class:

Python is an object-oriented programming language. Unlike procedure-oriented programming, where the main emphasis is on functions, object-oriented programming stresses on objects.

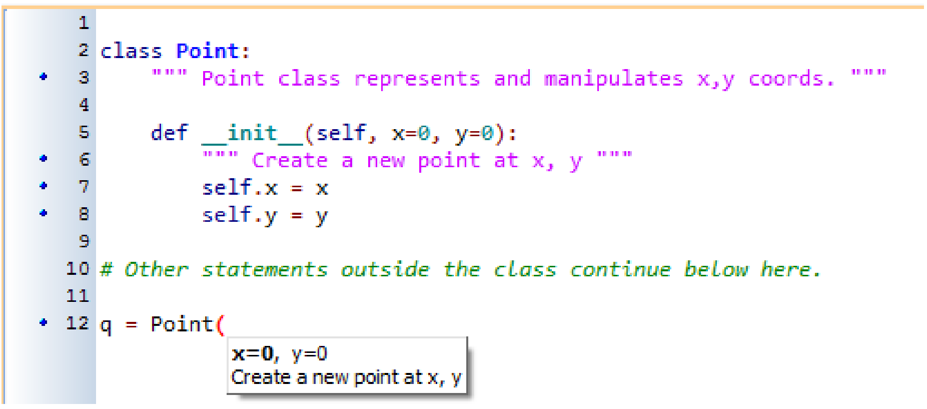
● An object is simply a collection of data (variables) and methods (functions) that act on those data. Similarly, a class is a blueprint for that object.

● We can think of class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house. House is the object.

● As many houses can be made from a house's blueprint, we can create many objects from a class. An object is also called an instance of a class and the process of creating this object is called instantiation.

● Like function definitions begin with the def keyword in Python, class definitions begin with a class keyword.

● The first string inside the class is called docstring and has a brief description about the class



```

class Point:
    """ Point class represents and manipulates x,y coords. """

    def __init__(self, x=0, y=0):
        """ Create a new point at x, y """
        self.x = x
        self.y = y

# Other statements outside the class continue below here.

q = Point(
```

```
x=0, y=0
Create a new point at x, y
```

Fig 3.6.1 Python class

4. Predicting Ships from Satellite Images

4.1. Project Requirements

4.1.1. Packages used:

- **NumPy:** NumPy is a python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it for free. NumPy stands for Numerical Python. In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

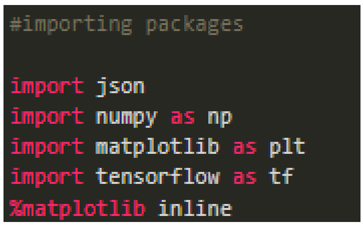
- **Pandas:** Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

- **Matplotlib:** Matplotlib is one of the most popular Python packages used for data visualization. It is a cross-platform library for making 2D plots from data in arrays. Matplotlib is written in Python and makes use of NumPy, the numerical mathematics extension of Python. It provides an object-oriented API that helps in embedding plots in applications using Python GUI toolkits such as PyQt, WxPythonotTkinter. It can be used in Python and I Python shells, Jupyter notebook and web application servers also. Matplotlib has a procedural interface named the Pylab, which is designed to resemble MATLAB, a proprietary programming language developed by MathWorks. Matplotlib along with NumPy can be considered as the open source equivalent of MATLAB.

- **TensorFlow:** TensorFlow is a free software library focused on machine learning created by Google. Initially released as part of the Apache 2.0 open-source license, TensorFlow was originally developed by engineers and researchers of the Google Brain Team, mainly for internal use. TensorFlow is considered the successor of the closed-source application Disbelief and is presently used by Google for research and production purposes. TensorFlow is considered the first serious implementation of a framework focused on deep learning. TensorFlow is also known as Google TensorFlow.

4.1.2. Versions of packages:

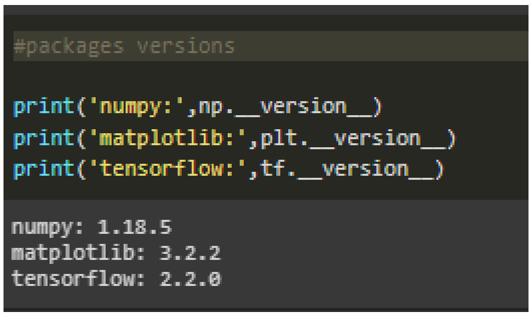
Loading required packages:



```
#importing packages

import json
import numpy as np
import matplotlib as plt
import tensorflow as tf
%matplotlib inline
```

Fig 4.1.2 Importing packages

Packages versions:



```
#packages versions

print('numpy:',np.__version__)
print('matplotlib:',plt.__version__)
print('tensorflow:',tf.__version__)
```

```
numpy: 1.18.5
matplotlib: 3.2.2
tensorflow: 2.2.0
```

Fig 4.1.2.1 Packages versions

4.1.3. Algorithms used:

- Convolution Neural Networks (CNN).

4.2. Project Statement:

The aim of this dataset is to help address the difficult task of detecting the location of large ships in satellite images. Automating this process can be applied to many issues including monitoring port activity levels and supply chain analysis.

4.3. Dataset Description:

Dataset contains of 3 distinct files. They are:

i) Shipsnet
   a. Xshipsnet
ii) Scenes
   a. Xscenes
iii) Shipsnet.json

Xshipsnet and xscenes contains images 4000 and 8 images, respectively. Whereas json file contains data about the images. JSON file has longitude and latitude of each ship, label, scenes_ids present in the data. The dataset includes 4000 80 x 80 RGB images labeled as either ships (1) or no ship (0) classification. Images were derived from planet scope full frame visual scenes products, which are orthorectified to 3-meter pixel size.

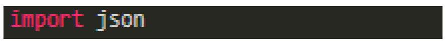
4.4. Objective of the Case Study:

Objective of problem is to identify ships from satellite images accurately.

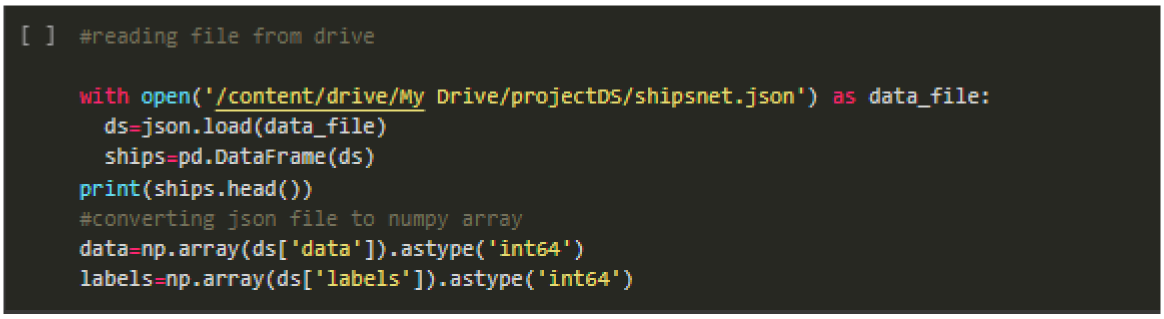
5. DATA PREPROCESSING/FEATURE ENGINEERING AND EDA

5.1. Image Data Analysis:

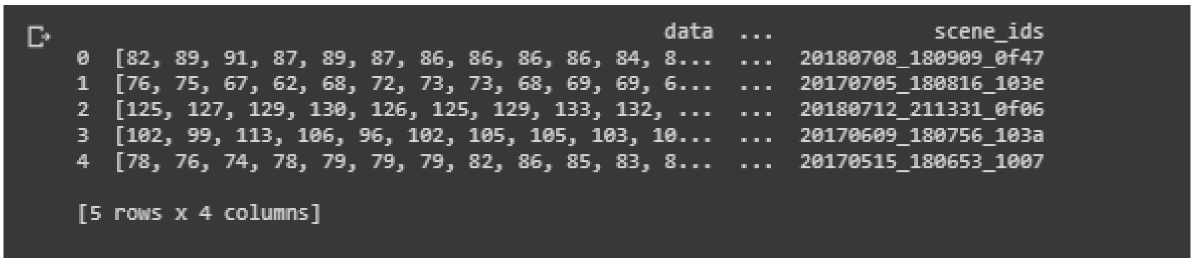
As the data is in json format we have to import json package and read the data in a specific manner that helps python read and print data that is in json format.



```
import json
```



```
[ ] #reading file from drive

    with open('/content/drive/My Drive/projectDS/shipsnet.json') as data_file:
      ds=json.load(data_file)
      ships=pd.DataFrame(ds)
    print(ships.head())
    #converting json file to numpy array
    data=np.array(ds['data']).astype('int64')
    labels=np.array(ds['labels']).astype('int64')
```



```
                                                data  ...              scene_ids
0  [82, 89, 91, 87, 89, 87, 86, 86, 86, 86, 84, 8...  ...  20180708_180909_0f47
1  [76, 75, 67, 62, 68, 72, 73, 73, 68, 69, 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

[5 rows x 4 columns]
```

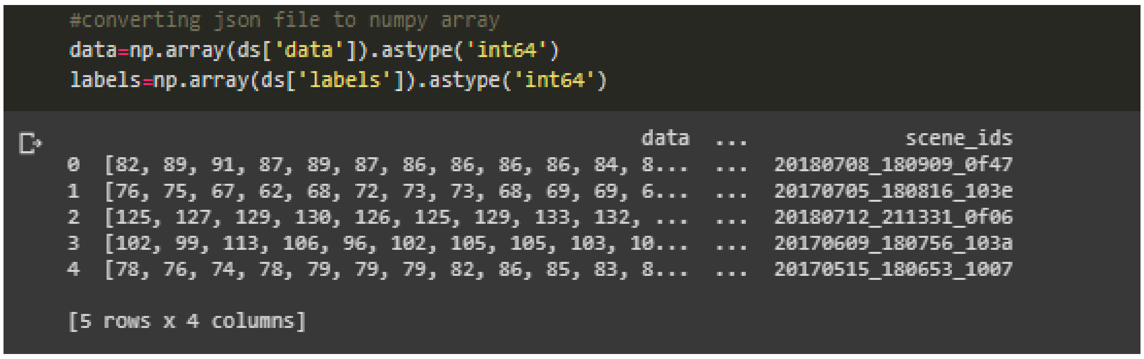
Fig 5.1.1 Reading and converting of data

- Json module makes it easier to parse JSON strings and files containing JSON objects.
- You can parse a json sting using “json.loads( )” method. This method returns a dictionary.
- You can use json.load( ) method to read a file containing JSON objects. Suppose we have a file x.json which contains json objects we can parse the file by using
“with open(‘path_to _file/x.json’) as data_file:”
“Data = json.load(data_file)”
- Here, we use open( ) function to read the json file. Then, the file is parsed using json.load( ) method which gives us a dictionary.

5.2. Data Type Conversion:

Converting JSON file to NumPy array:

We can convert the json object to NumPy array using np.array() function. Here "DataFrame.astype()" method is used to cast a pandas object to a specified dtype.astype() function also provides the capability to convert any suitable existing column to categorical type.



```
#converting json file to numpy array
data=np.array(ds['data']).astype('int64')
labels=np.array(ds['labels']).astype('int64')
```

```
                                        data  ...            scene_ids
0  [82, 89, 91, 87, 89, 87, 86, 86, 86, 86, 84, 8...  ...  20180708_180909_0f47
1  [76, 75, 67, 62, 68, 72, 73, 73, 68, 69, 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

[5 rows x 4 columns]
```

Fig 5.2.1 Converting of data

DataFrame.astype() function comes very handy when we want to case a particular column data type to another data type. Not only that but we can also use a Python dictionary input to change more than one column type at once. The key label in dictionary is corresponding to the column name and the values label in the dictionary is corresponding to the new data types we want the columns to be of.

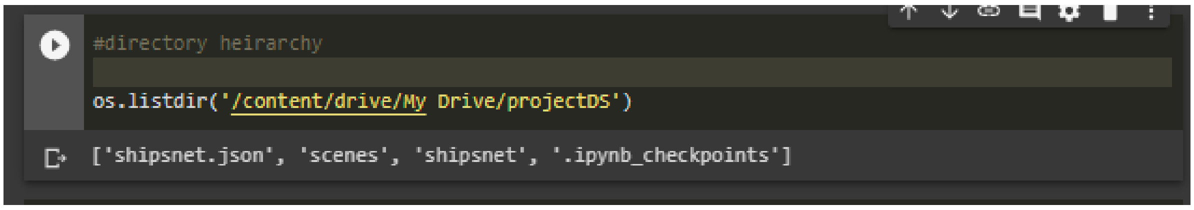
5.3. Handling Missing Values:

There are a number of schemes that have been developed to indicate the presence of missing data in a table or Data Frame. Generally, they revolve around one of two strategies: using a mask that globally indicates missing values or choosing a sentinel value that indicates a missing entry. In the masking approach, the mask might be an entirely separate Boolean array, or it may involve appropriation of one bit in the data representation to locally indicate the null status of a value. In the sentinel approach, the sentinel value could be some data-specific convention, such as indicating a missing integer value with -9999 or some rare bit pattern, or it could be a more global convention, such as indicating a missing floating-point value with NaN (Not a Number), a special value which is part of the IEEE floating-point specification.

- As this is an image dataset there are no missing values.

6. Visualization of Data

6.1. visualizing directory hierarchy:

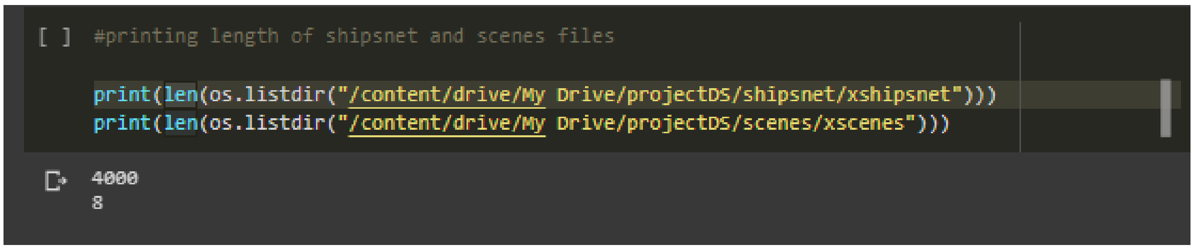


```
#directory heirarchy

os.listdir('/content/drive/My Drive/projectDS')

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

Fig 6.1. Directory hierarchy

As we can see in the directory hierarchy there are 3 datafiles.

6.2. Visualizing length of data files:

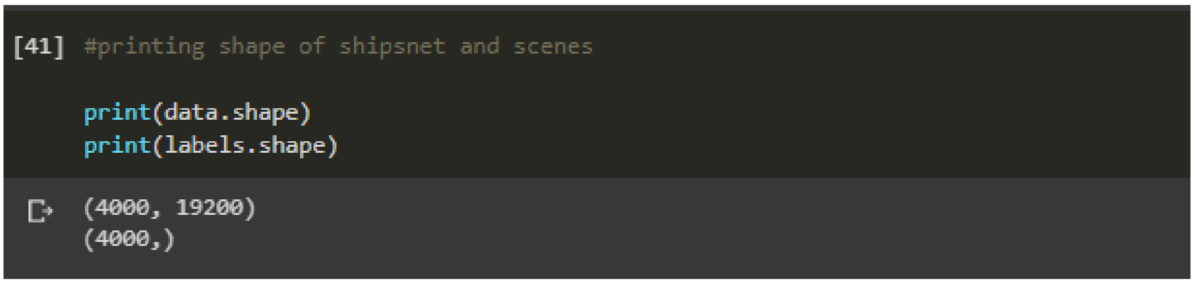


```
[ ] #printing length of shipsnet and scenes files

    print(len(os.listdir("/content/drive/My Drive/projectDS/shipsnet/xshipsnet")))
    print(len(os.listdir("/content/drive/My Drive/projectDS/scenes/xscenes")))

    4000
    8
```

6.2.0 Visualizing length of data files

Here we are printing the length of both the shipsnet and scenes directories. As we can see xshipsnet has 4000 images and xscenes has 8 images.

6.2.1. Shape of the data:



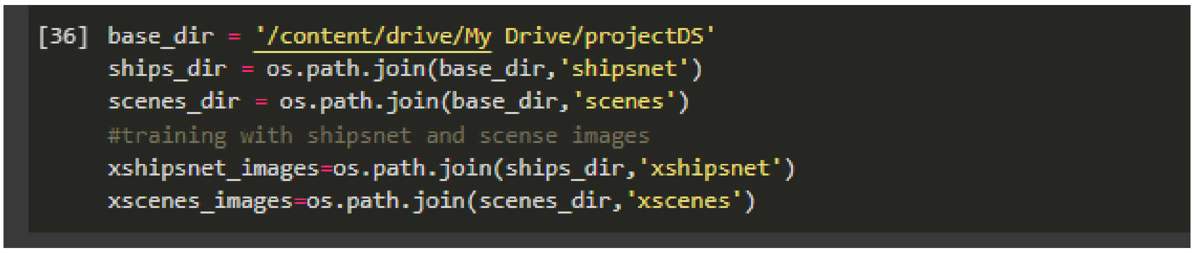
```
[41] #printing shape of shipsnet and scenes

     print(data.shape)
     print(labels.shape)

     (4000, 19200)
     (4000,)
```

Fig 6.2.1 Shape of data

- The dataset contains 4000 images. One image is represented as a vector of length 19200 elements.

- Labels contains vector of length 4000 elements.

6.3. Setting up base directories:



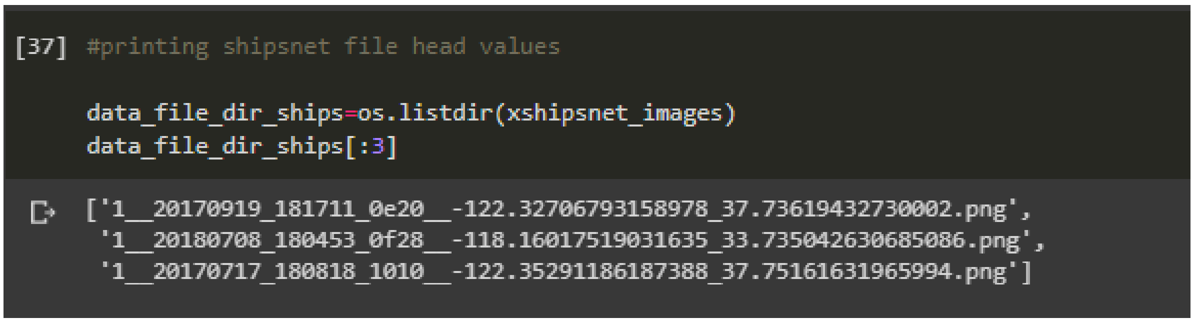
```
[36] base_dir = '/content/drive/My Drive/projectDS'
     ships_dir = os.path.join(base_dir,'shipsnet')
     scenes_dir = os.path.join(base_dir,'scenes')
     #training with shipsnet and scense images
     xshipsnet_images=os.path.join(ships_dir,'xshipsnet')
     xscenes_images=os.path.join(scenes_dir,'xscenes')
```

Fig 6.3.1 Setting up directories

As we visualized in the directory hierarchy, we have 2 distinct image datafile shipsnet and scenes. Each of these files have another file called as xshipsnet and xscenes which contains images. So, we are setting up directory paths for all the files using os.path.join( ) method.

6.4. Visualizing data from Shipsnet dataset:

6.4.1. visualizing files in dataset:

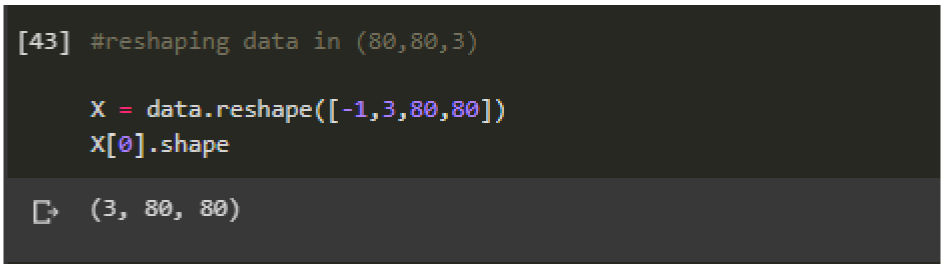


```
[37] #printing shipsnet file head values

     data_file_dir_ships=os.listdir(xshipsnet_images)
     data_file_dir_ships[:3]

     ['1__20170919_181711_0e20__-122.32706793158978_37.73619432730002.png',
      '1__20180708_180453_0f28__-118.16017519031635_33.735042630685086.png',
      '1__20170717_180818_1010__-122.35291186187388_37.75161631965994.png']
```

Fig 6.4.1 Visualizing shipsnet data

In the above image we can see the image names. We can observe that the data available in xshipsnet is images because the data is in “.png” format which is image format.

6.4.2. Reshaping of Shipsnet data:

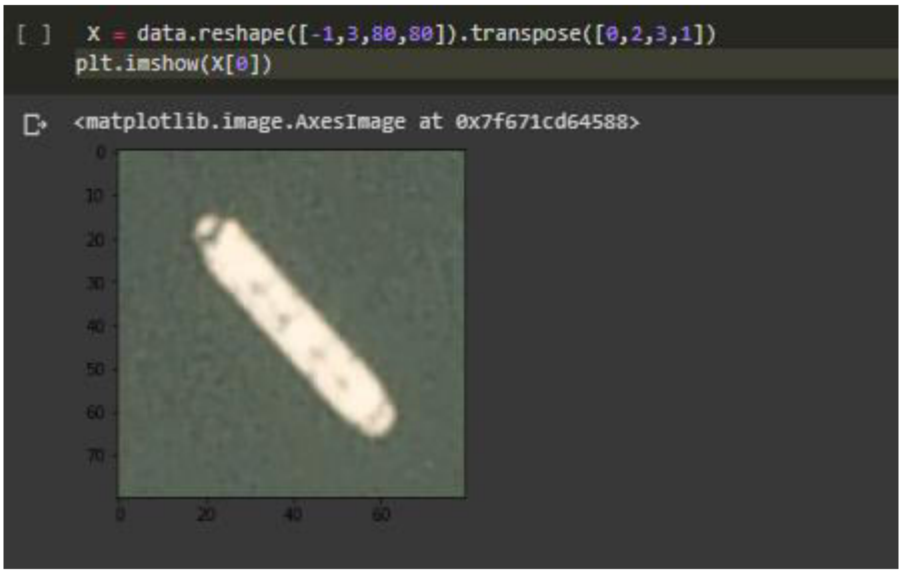


```
[43] #reshaping data in (80,80,3)

     X = data.reshape([-1,3,80,80])
     X[0].shape

     (3, 80, 80)
```

Fig 6.4.2 Reshaping of data

Since we need data to be in (80,80,3) format we need to reshape the data using reshape( ) function.

- This process is called as reshaping of data. The data is initially is in the shape (3,80,80) but the shape required by the user to perform is (80,80,3) so we can reshape the data into desired shape.
- Reshape( ) function is used to give a new shape to an array without changing its data. Reshaped array will be compatible with the original shape.



```
[ ]  X = data.reshape([-1,3,80,80]).transpose([0,2,3,1])
    plt.imshow(X[0])

    <matplotlib.image.AxesImage at 0x7f671cd64588>
```

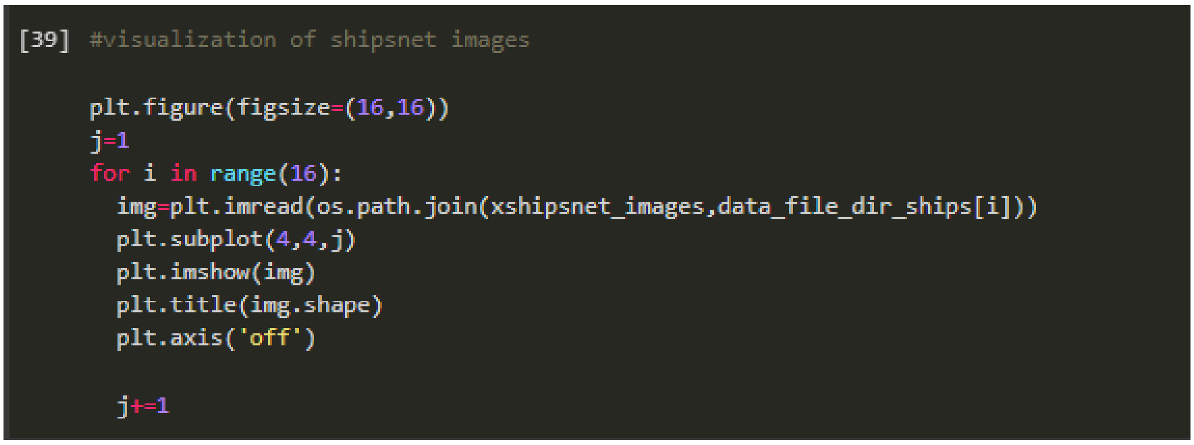
Fig 6.4.2.1 Shipsnet image transpose visualization

As we can see in the above image data has been reshaped into (80,80,3) by using reshape( ) and transpose( ) functions.

- The reshape( ) function is used to give a new shape to an array without changing its data.

- The transpose() function Reverse or permute the axes of an array, returns the modified array. In layman's terms transpose is simply reversing of the data from left to right.
- We can see that transpose( ) and reshape( ) are working as expected as we checked it by printing out an image.

6.4.3. Visualizing images from Shipsnet dataset:



```
[39] #visualization of shipsnet images

     plt.figure(figsize=(16,16))
     j=1
     for i in range(16):
       img=plt.imread(os.path.join(xshipsnet_images,data_file_dir_ships[i]))
       plt.subplot(4,4,j)
       plt.imshow(img)
       plt.title(img.shape)
       plt.axis('off')

       j+=1
```

Fig 6.4.3 Shipsnet image visualization

We are visualizing the images from xshipsnet by using a for loop and we are reading the images with the directory we set up. As we are using a (4,4) subplot we get 16 images and after every image is read j value increases by 1 and the loop continues.

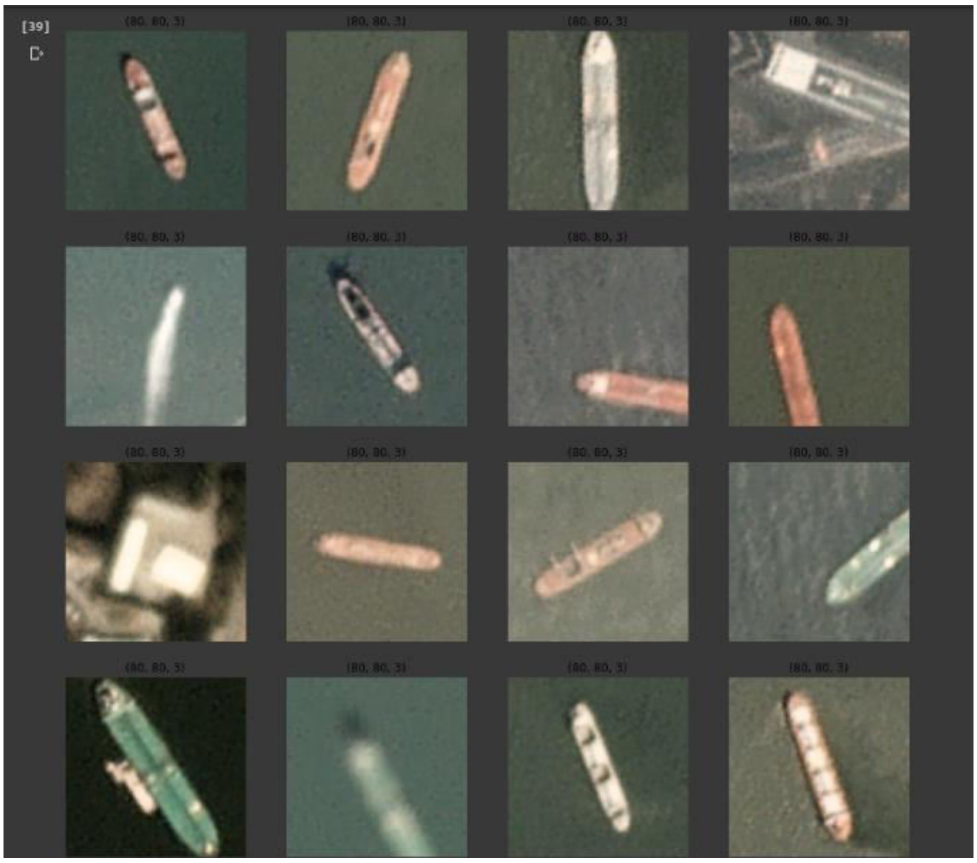
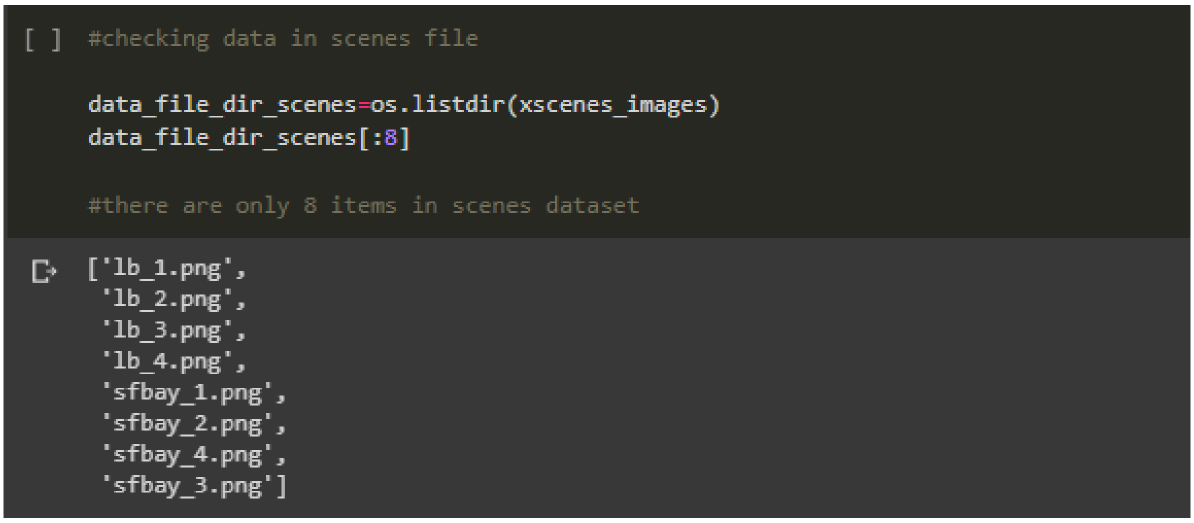


Fig 6.4.4 Shipsnet images

In the below figure we can see the image visualization. And we can see the shape of the image which is (80,80,3) as we reshaped it from (3,80,80) to (80,80,3).

6.5. Visualizing data from Scenes dataset:



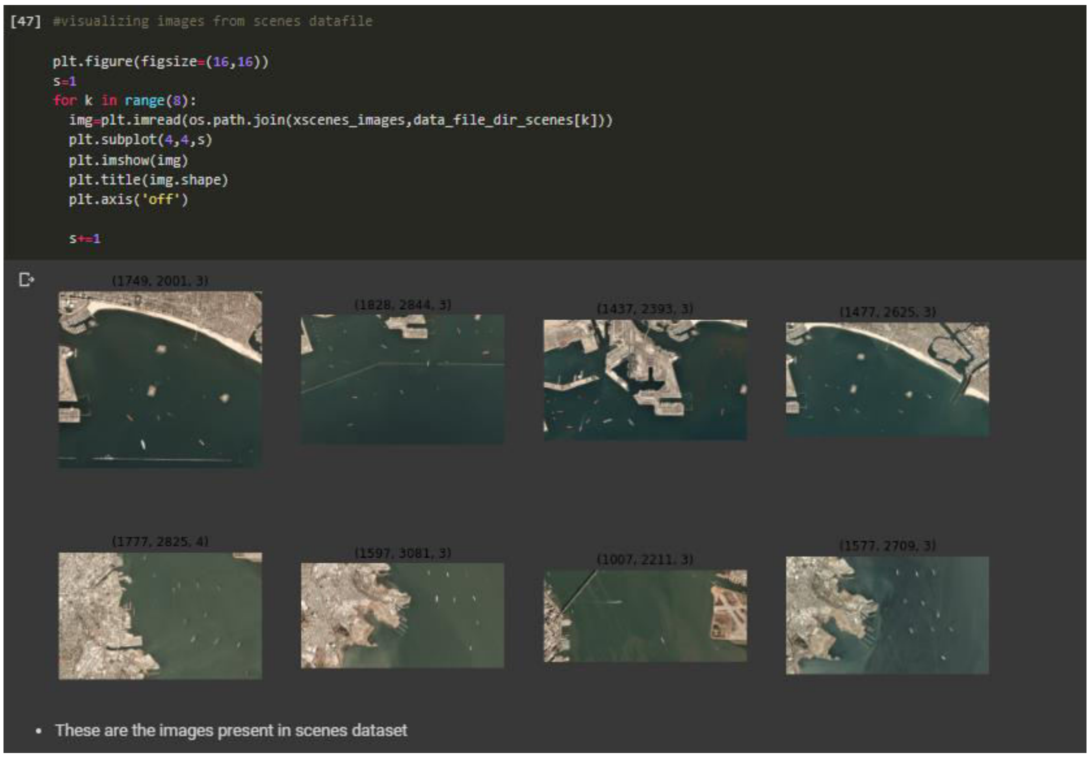
```
[ ]  #checking data in scenes file

     data_file_dir_scenes=os.listdir(xscenes_images)
     data_file_dir_scenes[:8]

     #there are only 8 items in scenes dataset
```

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

Fig 6.5.0 Scenes data

As we have seen earlier there are only 8 images in scenes datafile. And the data in scenes dataset are images because the files are in .png format.

6.5.1. Visualizing images from Scenes dataset:



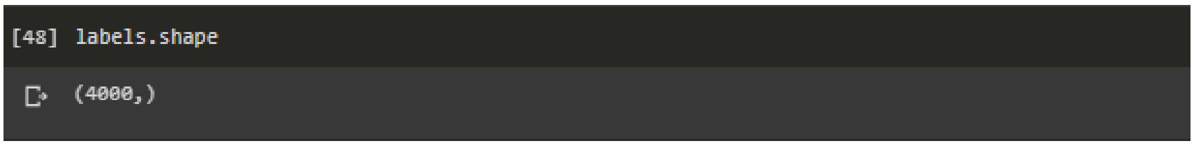
```
[47] #visualizing images from scenes datafile

     plt.figure(figsize=(16,16))
     s=1
     for k in range(8):
       img=plt.imread(os.path.join(xscenes_images,data_file_dir_scenes[k]))
       plt.subplot(4,4,s)
       plt.imshow(img)
       plt.title(img.shape)
       plt.axis('off')

       s+=1
```

- These are the images present in scenes dataset

Fig 6.5.1 Scenes images

These are the images present in scenes dataset. We can see the shape of these images as the shape is not (80,80,3) we have to reshape the data into desired shape while performing operations on the data.

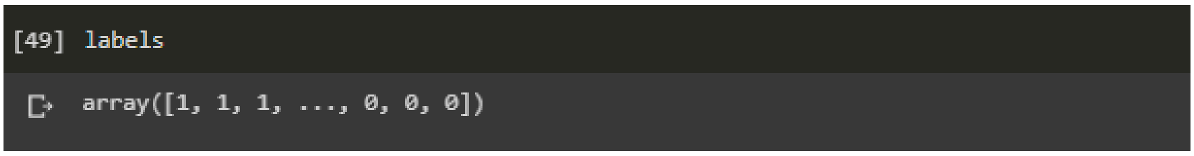
6.6. Bincount of data:



```
[48] labels.shape

     (4000,)
```

Fig 6.6.1 Shape of label

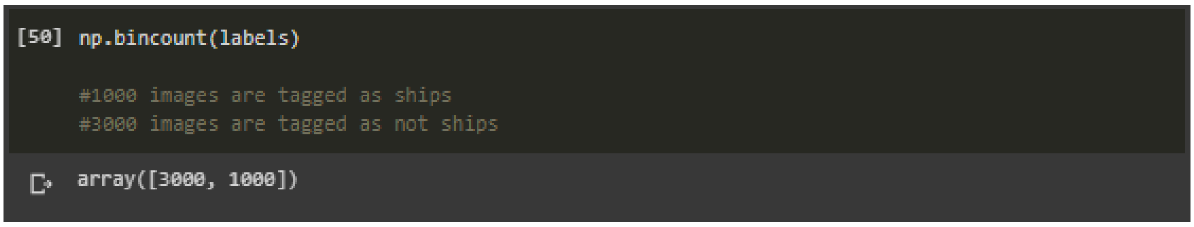
Labels is the output vector. Labels contains 4000 elements.



```
[49] labels

array([1, 1, 1, ..., 0, 0, 0])
```

Fig 6.6.2 Labels array

As we can see here labels data is in only 0's and 1's because the images are either tagged as ships which is represented by 1 or not ships which is represented as 0. So, the data array contains only 0's and 1's. The data is in array format as we converted it into NumPy array.



```
[50] np.bincount(labels)

     #1000 images are tagged as ships
     #3000 images are tagged as not ships

array([3000, 1000])
```

Fig 6.6.3 Bincount

numpy.bincount(arr, weights = None, min_len = 0) : In an array of positive integers, it counts the occurrence of each element. Each bin value is the occurrence of its index. One can also set the bin size accordingly.

So, by using bincount we get the exact number of images that are tagged as ships and not ships. In 4000 images 1000 images are ships and 3000 are not ships.

6.7. Train test split:

One of the first decisions to make when starting a modeling project is how to utilize the existing data. One common technique is to split the data into two groups typically referred to as the training and testing sets. The training set is used to develop models and feature sets; they are the substrate for estimating parameters, comparing models, and all of the other activities required to reach a final model. The test set is used only at the conclusion of these activities for estimating a final, unbiased assessment of the model's performance. It is critical that the test set not be used prior to this point. Looking at the test sets results would bias the outcomes since the testing data will have become part of the model development process. There are a number of ways to split the data into training and testing sets. The most common approach is to use some version of random sampling.

Completely random sampling is a straightforward strategy to implement and usually protects the process from being biased towards any characteristic of the data. However, this approach can be problematic when the response is not evenly distributed across the outcome. A less risky splitting strategy would be to use a stratified random sample based on the outcome. For classification models, this is accomplished by selecting samples at random within each class. This approach ensures that the frequency distribution of the outcome is approximately equal within the training and test sets. When the outcome is numeric, artificial strata can be constructed based on the quartiles of the data.

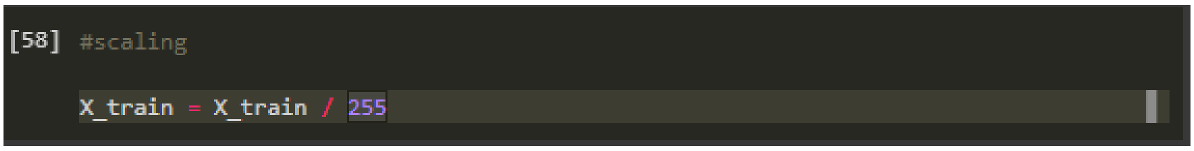
We are using Shipsnet datafile as our testing data and in this shipsnet data we are taking 0.2 that is 20 percent as validation split which means it is an 80-20 split. In validation dataset we check if the train data is working as expected and giving us accurate predictions. We are using shipsnet and scenes datafiles as our testing dataset.

6.8. Scaling of data:

- Images are comprised of matrices of pixel values.
- Black and white images are single matrix of pixels, whereas color images have a separate array of pixel values for each color channel, such as red, green, and blue.
- Pixel values are often unsigned integers in the range between 0 and 255. Although these pixel values can be presented directly to neural network models in their raw format, this can result in challenges during modeling, such as in the slower than expected training of the model.
- Instead, there can be great benefit in preparing the image pixel values prior to modeling, such as simply scaling pixel values to the range 0-1 to centering and even standardizing the values.
- For most image data, the pixel values are integers with values between 0 and 255.
- Neural networks process inputs using small weight values, and inputs with large integer values can disrupt or slow down the learning process. As such it is good

practice to normalize the pixel values so that each pixel value has a value between 0 and 1.

- It is valid for images to have pixel values in the range 0-1 and images can be viewed normally.
- This can be achieved by dividing all pixel's values by the largest pixel value; that is 255. This is performed across all channels, regardless of the actual range of pixel values that are present in the image.



```
[58] #scaling

     X_train = X_train / 255
```

Fig 6.8 Scaling of data

7. MODEL BUILDING AND EVALUATION

7.1. Brief about the algorithms used:

- **Convolutional neural network:**

In neural networks, Convolutional neural network (ConvNets or CNNs) is one of the main categories to do image recognition, image classification. Object detection, recognition of faces etc., are some of the areas where CNN is widely used.

CNN image classification take an input image, processes it and classifies it under certain categorie (E.g., Dog, Cat, Tiger, Lion). Computers sees an input image as array of pixels, and it depends on the image resolution. Based on the image resolution, it will see h x w x d(h = Height, w = Width, d = Dimension). E.g., An image of 6 x 6 x 3 array of matrix of RGB (3 refers to RGB values) and an image of 4 x 4 x 1 array of matrix of grayscale image.

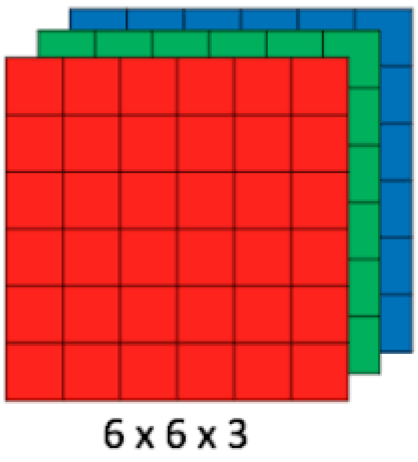


Fig 7.1.1 Array of matrix of rgb

Technically, deep learning is used for CNN models to train and test, each input image will pass through a series of convolution layers with filters (Kernels), Pooling, fully connected layers (FC) and SoftMax function is applied to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.

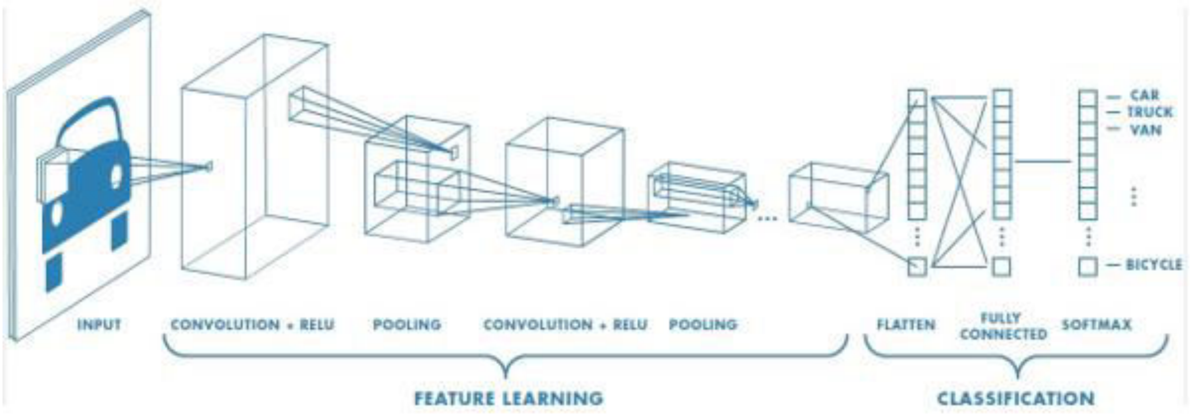
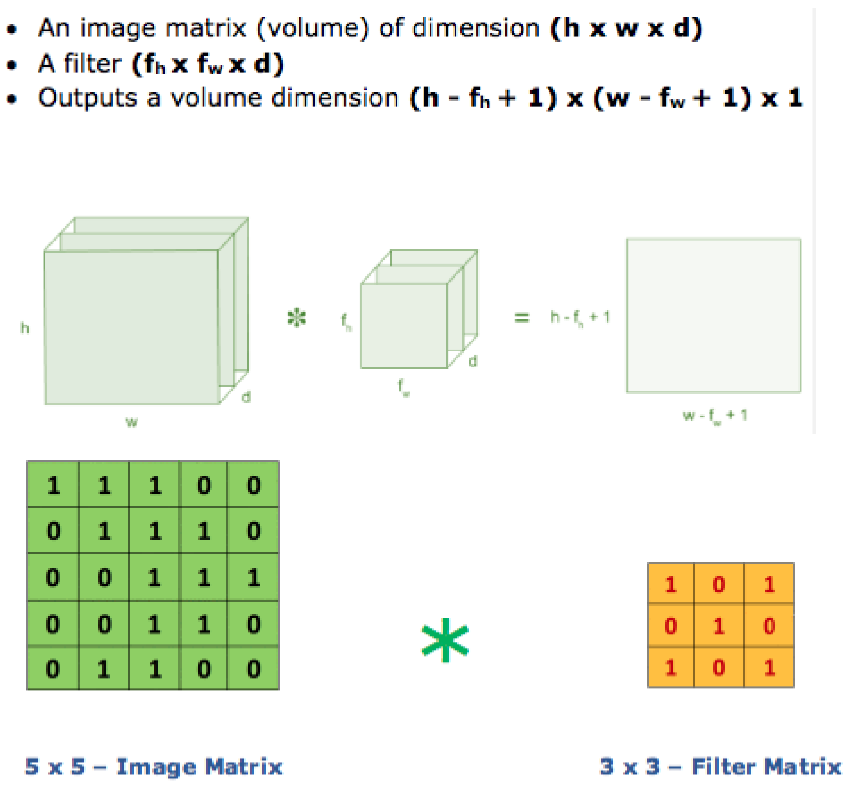


Fig 7.1.2 Complete flow of CNN

- **Convolution Layer**

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.



- An image matrix (volume) of dimension **(h x w x d)**
- A filter **(f_h x f_w x d)**
- Outputs a volume dimension **(h - f_h + 1) x (w - f_w + 1) x 1**

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

5 x 5 – Image Matrix

1	0	1
0	1	0
1	0	1

3 x 3 – Filter Matrix

Fig 7.1.3 Convolution layer

Consider a 5 x 5 image whose pixel values are 0,1 and filter matrix 3 x3 as shown below then the convolution of 5 x 5 image matrix multiplies with 3 x 3 filter matrix which is called a feature map. Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters.

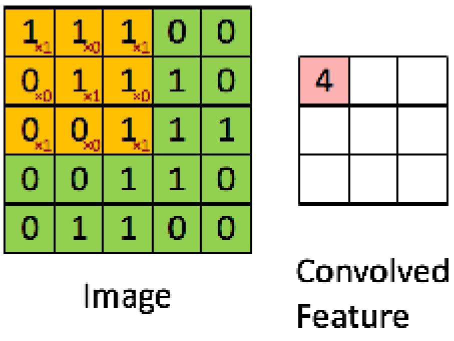


Fig 7.1.4 Feature map

- **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.

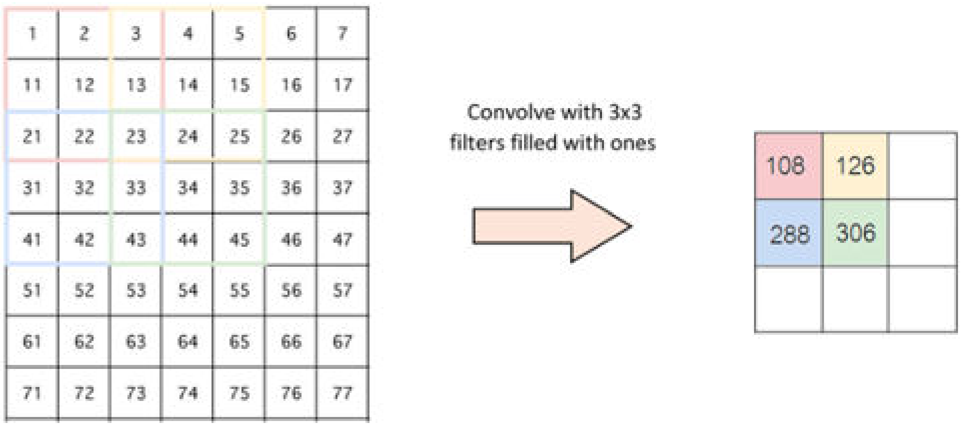


Fig 7.1.5 Convolution working

- **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.

- **Non-Linearity (ReLU)**

- ReLU stands for Rectified Linear Unit for a non-linear operation. The output is ***f(x) = max(0, x).***
- **Why ReLU is important:** ReLU's purpose is to introduce non-linearity in our ConvNet. Since the real-world data would want our ConvNet to learn would be non-negative linear values.
- There are other nonlinear functions such as tanh or sigmoid that can also be used instead of ReLU. Most of the data scientists use ReLU since performance wise ReLU is better than the other two.

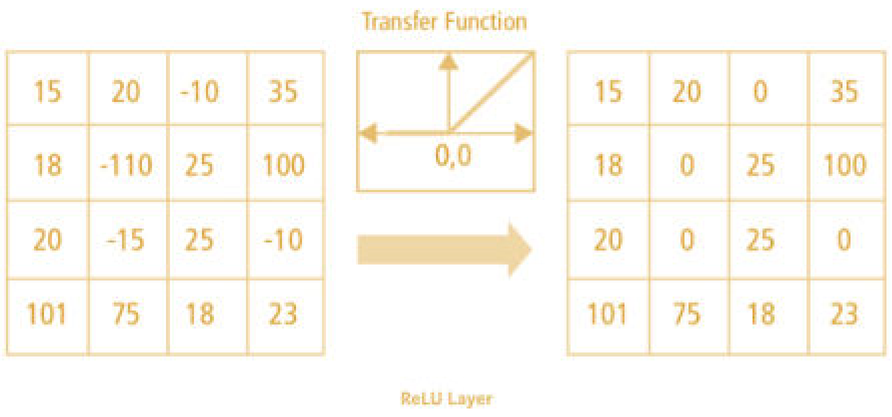


Fig 7.1.6 ReLU layer

- **Pooling Layer**

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called sub-sampling or down sampling which reduces the dimensionality of each map but retains important information. Spatial pooling can be of different types:

- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling takes the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.

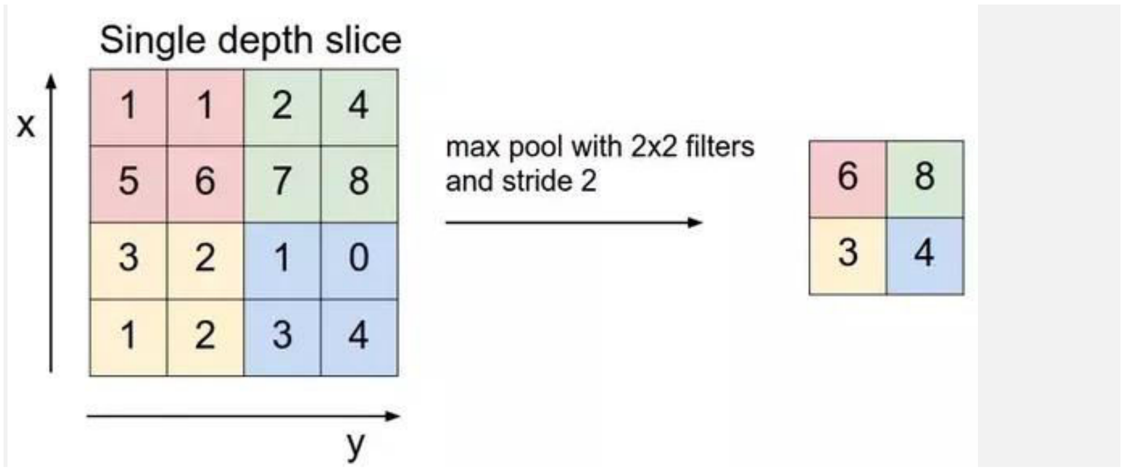


Fig 7.1.7 MaxPooling

- **Fully Connected Layer**

The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like a neural network.

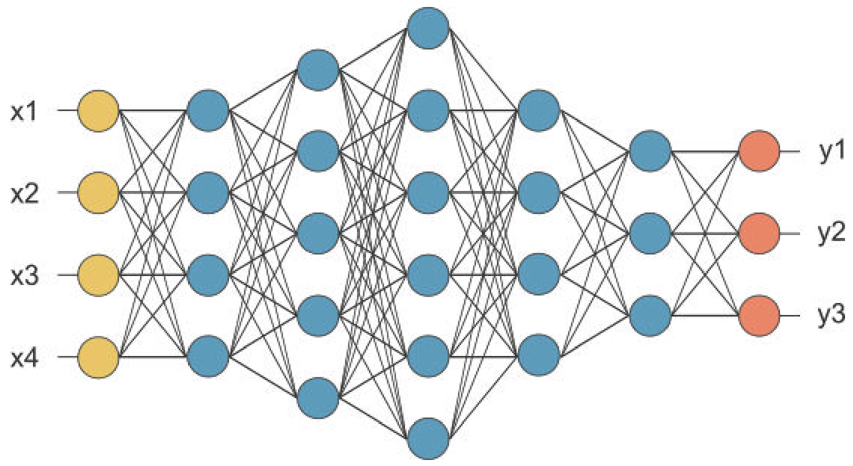


Fig 7.1.8 Fully connected layer

In the above diagram, the feature map matrix will be converted as vector (x1, x2, x3,...). With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as SoftMax or sigmoid to classify the outputs as cat, dog, car, truck etc.,

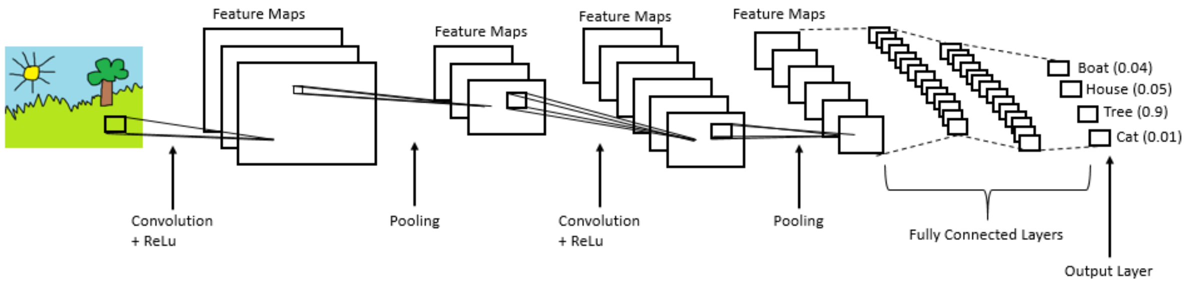


Fig 7.1.9 CNN working

7.2. Train the Models:

Splitting the data: After the preprocessing is done then the data is split into train and test sets.

● In Machine Learning in order to access the performance of the classifier. You train the classifier using 'training set' and then test the performance of your classifier on unseen 'test set’. An important point to note is that during training the classifier only uses the training set. The test set must not be used during training the classifier. The test set will only be available during testing the classifier.

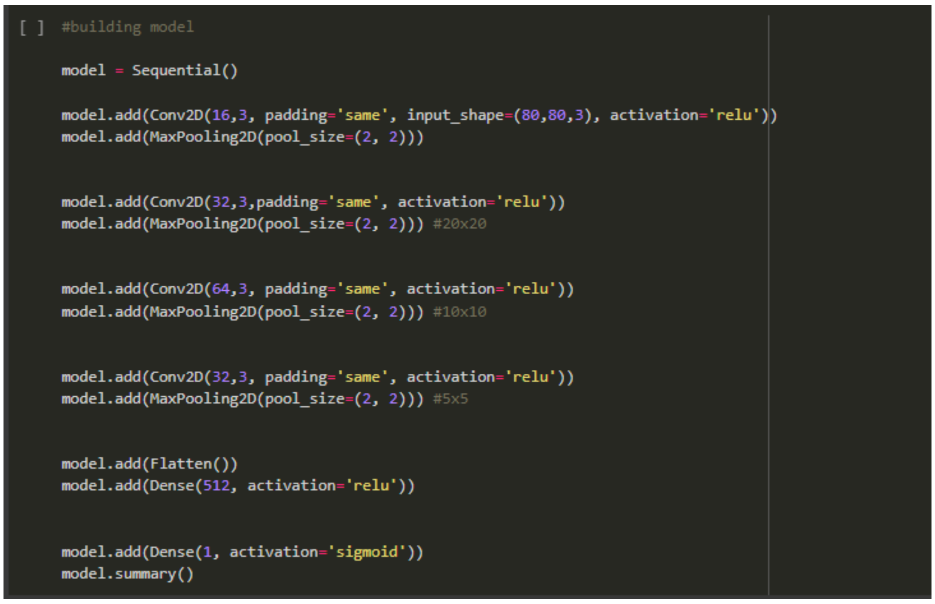
● Training set - A subset to train a model. (Model learns patterns between Input and

Output)

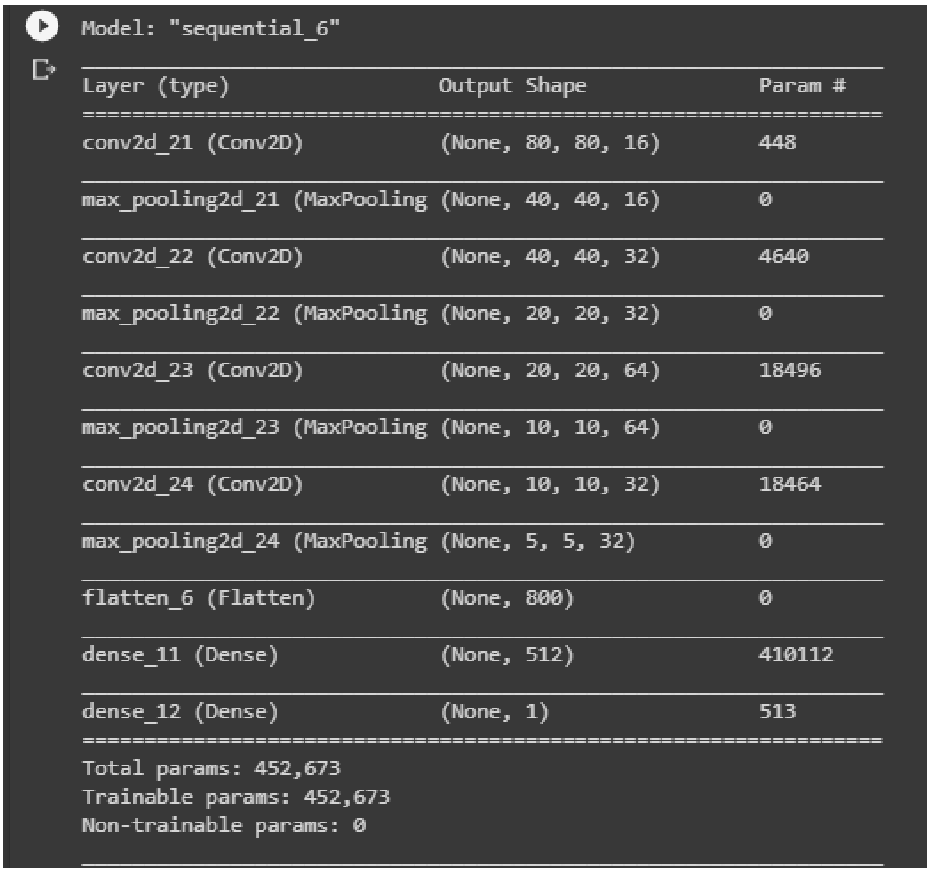
● Test set - A subset to test the trained model. (To test whether the model has correctly learnt).

●The amount or percentage of Splitting can be taken as specified.

● First, we need to identify the input and output variables and we need to separate the input set and output set.



```
#building model

model = Sequential()

model.add(Conv2D(16,3, padding='same', input_shape=(80,80,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))


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())
model.add(Dense(512, activation='relu'))


model.add(Dense(1, activation='sigmoid'))
model.summary()
```



```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_21 (Conv2D)           (None, 80, 80, 16)        448
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 40, 40, 16)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 40, 40, 32)        4640
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 20, 20, 32)        0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 20, 20, 64)        18496
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 10, 10, 32)        18464
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 5, 5, 32)          0
_________________________________________________________________
flatten_6 (Flatten)          (None, 800)               0
_________________________________________________________________
dense_11 (Dense)             (None, 512)               410112
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 513
=================================================================
Total params: 452,673
Trainable params: 452,673
Non-trainable params: 0
_________________________________________________________________
```

Fig 7.2.1 model building

7.3. Building the Model:

- **The sequential model**

The most common type of model is the sequential model, which is a linear stack of layers. You can create a sequential model by passing a list of layers to the sequential() function.

- **Functional model**

Another way to create a Layers model is via the tf.model() function. The key difference between tf.model() and tf.sequential() is that tf.model() allows you to create an arbitrary graph of layers, as long as they do not have cycles.

- **Dense Layer**

A dense layer will consider the entire image. It will look at all the pixels and use that information that information to generate some output.

- **Convolutional Layer**

The convolutional layer will look at the specific parts of the image.

- **MaxPooling2D Layer**

Down samples the input representation by taking the maximum value over the window defined by pool_size for each dimension along the feature's axis. The window is shifted by strides in each dimension. The resulting output when using "valid" padding option has a shape(number of rows or columns) of output_shape = (input_shape - pool_size + 1) / strides)

The resulting output shape when using the "same" padding option is output_shape = input_shape / strides

Arguments:

- **pool_size**: integer or tuple of 2 integers, window size over which to take the maximum. (2, 2) will take the max value over a 2x2 pooling window. If only one integer is specified, the same window length will be used for both dimensions.
- **strides**: Integer, tuple of 2 integers, or None. Strides values. Specifies how far the pooling window moves for each pooling step. If None, it will default to pool_size.
- **padding**: One of "valid" or "same" (case-insensitive). "valid" means no padding. "same" results in padding evenly to the left/right or up/down of the input such that output has the same height/width dimension as the input.
- **data_format**: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

- **Flatten Layer**

  Flattens the input. Does not affect the batch size.

  Note: If inputs are shaped (batch,) without a feature axis, then flattening adds an extra channel dimension and output shape is (batch, 1).

- **Model summary**

  It gives us the summary of the model that we built.

  Call model.summary() to print a useful summary of the model, which includes:

- Name and type of all layers in the model.
- Output shape for each layer.
- Number of weight parameters of each layer.
- If the model has general topology (discussed below), the inputs each layer receives
- The total number of trainable and non-trainable parameters of the model.

- **Activation function**

Activation function decides whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

We know, neural network has neurons that work in correspondence of weight, bias and their respective activation function. In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as back-propagation. Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.

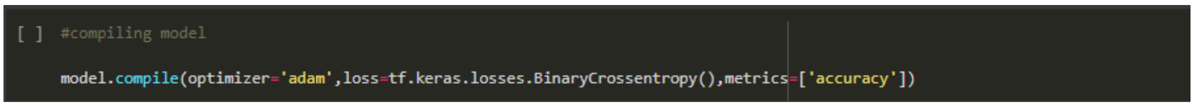
- **RELU:**

Stands for Rectified linear unit. It is the most widely used activation function. Chiefly implemented in hidden layers of Neural network.

- **Equation:** ***A(x) = max (0, x)***. It gives an output x if x is positive and 0 otherwise.
- **Value Range:** [0, inf)
- **Nature:** non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
- **Uses:** ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

- **Sigmoid Function:**
- It is a function which is plotted as **'S'** shaped graph.
- **Equation:**
  $A = 1/(1 + e^{-x})$
- **Nature:** Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- **Value Range:** 0 to 1
- **Uses:** Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be ***1*** if value is greater than **0.5** and ***0*** otherwise.

7.4. Compile the Model



```
[ ]  #compiling model
     model.compile(optimizer='adam',loss=tf.keras.losses.BinaryCrossentropy(),metrics=['accuracy'])
```

Fig 7.4.1 Compiling model

The last step in building the model is to define the loss function, optimizer and metrics we would like to track.

To train a model with fit(), you need to specify a loss function, an optimizer, and optionally, some metrics to monitor.

You pass these to the model as arguments to the compile() method.

Optimizers:

- SGD() (with or without momentum)
- RMSprop()
- Adam()
- etc.

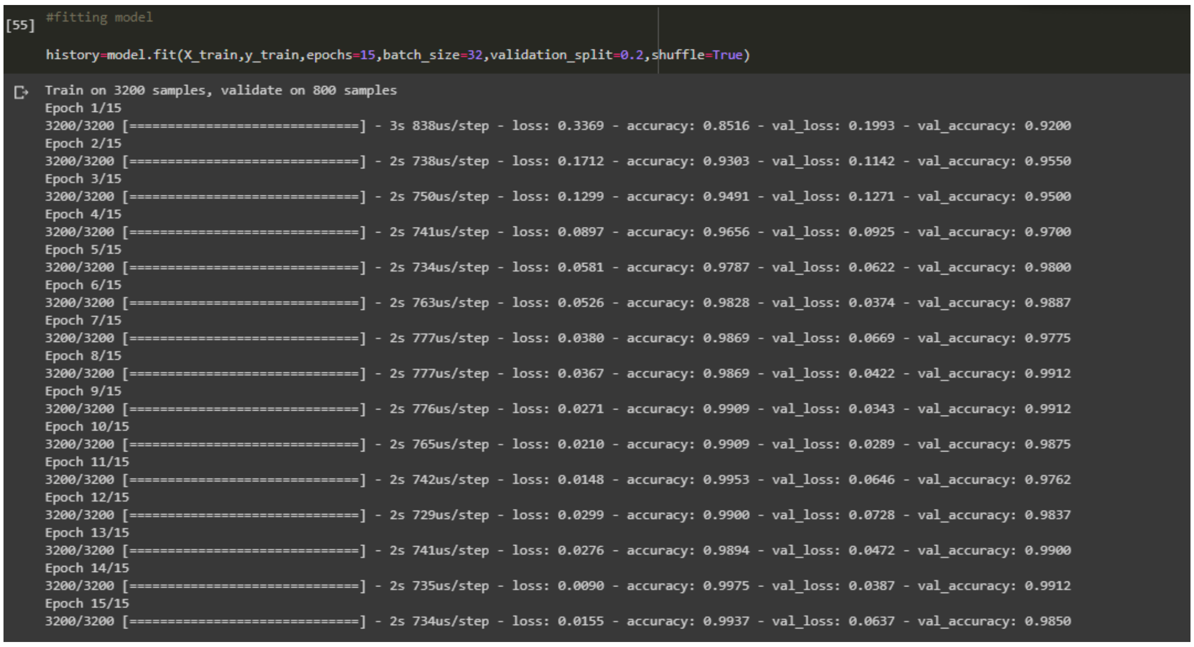
Losses:

- MeanSquaredError()
- KLDivergence()
- CosineSimilarity()
- etc.

Metrics:

- AUC()
- Precision()
- Recall()
- etc.

7.5. Fitting the model



```
[55] #fitting model

     history=model.fit(X_train,y_train,epochs=15,batch_size=32,validation_split=0.2,shuffle=True)

     Train on 3200 samples, validate on 800 samples
     Epoch 1/15
     3200/3200 [==============================] - 3s 838us/step - loss: 0.3369 - accuracy: 0.8516 - val_loss: 0.1993 - val_accuracy: 0.9200
     Epoch 2/15
     3200/3200 [==============================] - 2s 738us/step - loss: 0.1712 - accuracy: 0.9303 - val_loss: 0.1142 - val_accuracy: 0.9550
     Epoch 3/15
     3200/3200 [==============================] - 2s 750us/step - loss: 0.1299 - accuracy: 0.9491 - val_loss: 0.1271 - val_accuracy: 0.9500
     Epoch 4/15
     3200/3200 [==============================] - 2s 741us/step - loss: 0.0897 - accuracy: 0.9656 - val_loss: 0.0925 - val_accuracy: 0.9700
     Epoch 5/15
     3200/3200 [==============================] - 2s 734us/step - loss: 0.0581 - accuracy: 0.9787 - val_loss: 0.0622 - val_accuracy: 0.9800
     Epoch 6/15
     3200/3200 [==============================] - 2s 763us/step - loss: 0.0526 - accuracy: 0.9828 - val_loss: 0.0374 - val_accuracy: 0.9887
     Epoch 7/15
     3200/3200 [==============================] - 2s 777us/step - loss: 0.0380 - accuracy: 0.9869 - val_loss: 0.0669 - val_accuracy: 0.9775
     Epoch 8/15
     3200/3200 [==============================] - 2s 777us/step - loss: 0.0367 - accuracy: 0.9869 - val_loss: 0.0422 - val_accuracy: 0.9912
     Epoch 9/15
     3200/3200 [==============================] - 2s 776us/step - loss: 0.0271 - accuracy: 0.9909 - val_loss: 0.0343 - val_accuracy: 0.9912
     Epoch 10/15
     3200/3200 [==============================] - 2s 765us/step - loss: 0.0210 - accuracy: 0.9909 - val_loss: 0.0289 - val_accuracy: 0.9875
     Epoch 11/15
     3200/3200 [==============================] - 2s 742us/step - loss: 0.0148 - accuracy: 0.9953 - val_loss: 0.0646 - val_accuracy: 0.9762
     Epoch 12/15
     3200/3200 [==============================] - 2s 729us/step - loss: 0.0299 - accuracy: 0.9900 - val_loss: 0.0728 - val_accuracy: 0.9837
     Epoch 13/15
     3200/3200 [==============================] - 2s 741us/step - loss: 0.0276 - accuracy: 0.9894 - val_loss: 0.0472 - val_accuracy: 0.9900
     Epoch 14/15
     3200/3200 [==============================] - 2s 735us/step - loss: 0.0090 - accuracy: 0.9975 - val_loss: 0.0387 - val_accuracy: 0.9912
     Epoch 15/15
     3200/3200 [==============================] - 2s 734us/step - loss: 0.0155 - accuracy: 0.9937 - val_loss: 0.0637 - val_accuracy: 0.9850
```

Fig 7.5.1 Fitting the model

- **Epochs**

In terms of artificial neural networks, an epoch refers to one cycle through the full training dataset. Usually, training a neural network takes more than a few epochs. In other words, if we feed a neural network the training data for more than one epoch in different patterns, we hope for a better generalization when given a new "unseen" input (test data). An epoch is often mixed up with an iteration. Iterations is the number of batches or steps through partitioned packets of the training data, needed to complete one epoch. Heuristically, one motivation is that (especially for large but finite training sets) it gives the network a chance to see the previous data to readjust the model parameters so that the model is not biased towards the last few data points during training.

Arguments

x: Input data. It could be:

- A NumPy array (or array-like), or a list of arrays (in case the model has multiple inputs).
- A TensorFlow tensor, or a list of tensors (in case the model has multiple inputs).

- A dict mapping input names to the corresponding array/tensors if the model has named inputs.
- A tf.data dataset. Should return a tuple of either (inputs, targets) or (inputs, targets, sample_weights).
- A generator or keras.utils.Sequence returning (inputs, targets) or (inputs, targets, sample_weights). A more detailed description of unpacking behavior for iterator types (Dataset, generator, Sequence) is given below.

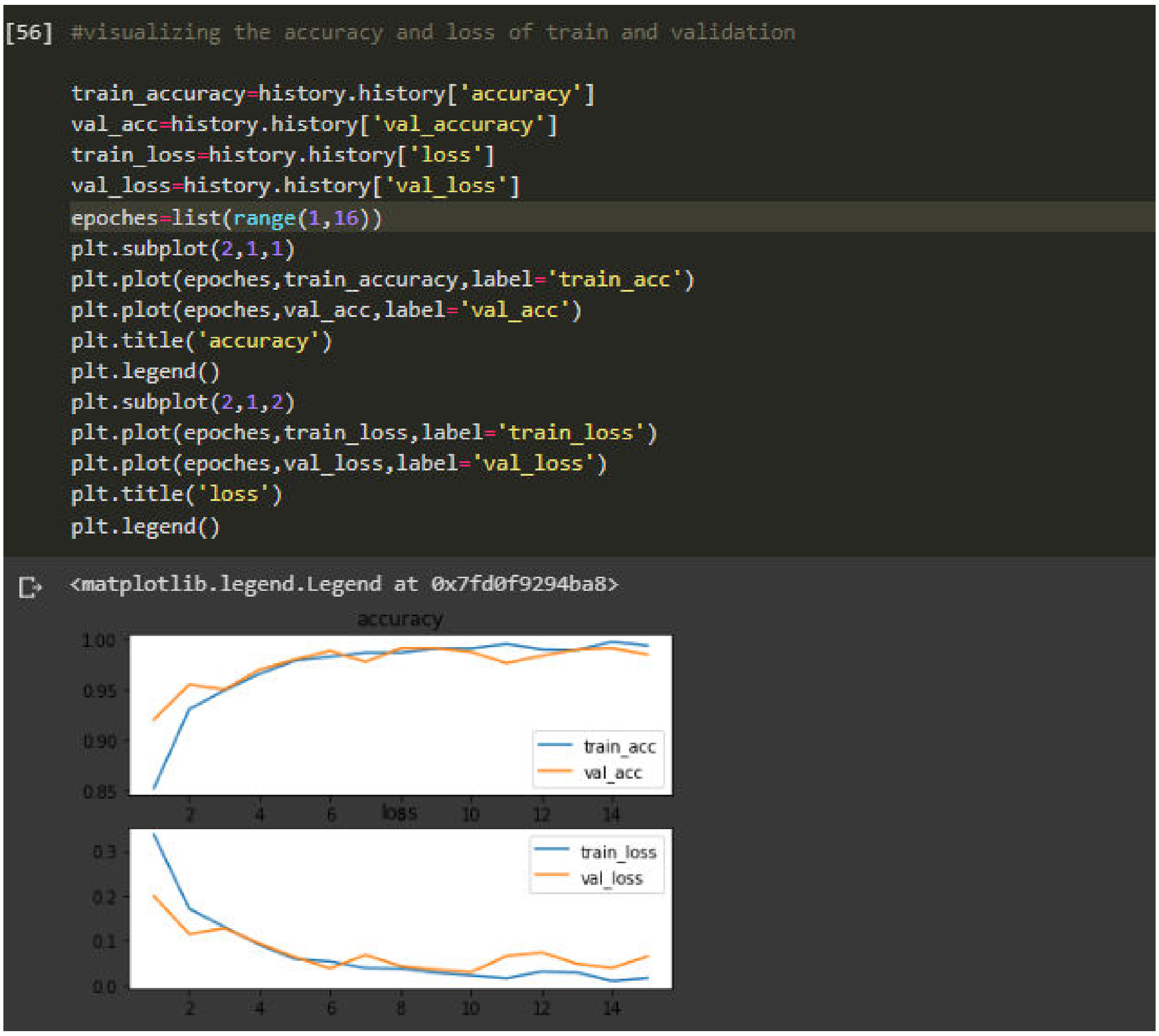
y: Target data. Like the input data x, it could be either NumPy array(s) or TensorFlow tensor(s). It should be consistent with x (you cannot have NumPy inputs and tensor targets, or inversely). If x is a dataset, generator, or keras.utils.Sequence instance, y should not be specified (since targets will be obtained from x).

batch_size: Integer or None. Number of samples per gradient update. If unspecified, batch_size will default to 32. Do not specify the batch_size if your data is in the form of datasets, generators, or keras.utils.Sequence instances (since they generate batches).

shuffle: Boolean (whether to shuffle the training data before each epoch) or str (for 'batch'). This argument is ignored when x is a generator. 'batch' is a special option for dealing with the limitations of HDF5 data; it shuffles in batch-sized chunks. Has no effect when steps_per_epoch is not None.

validation_data: Data on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be trained on this data. Thus, note the fact that the validation loss of data provided using validation_split or validation_data is not affected by regularization layers like noise and dropuout. validation_data will override validation_split. validation_data could be: - tuple (x_val, y_val) of NumPy arrays or tensors - tuple (x_val, y_val, val_sample_weights) of Numpy arrays - dataset For the first two cases, batch_size must be provided. For the last case, validation_steps could be provided. Note that validation_data does not support all the data types that are supported in x, e.g., dict, generator or keras.utils.Sequence.

7.6. Visualizing loss and accuracy



```
[56] #visualizing the accuracy and loss of train and validation

    train_accuracy=history.history['accuracy']
    val_acc=history.history['val_accuracy']
    train_loss=history.history['loss']
    val_loss=history.history['val_loss']
    epoches=list(range(1,16))
    plt.subplot(2,1,1)
    plt.plot(epoches,train_accuracy,label='train_acc')
    plt.plot(epoches,val_acc,label='val_acc')
    plt.title('accuracy')
    plt.legend()
    plt.subplot(2,1,2)
    plt.plot(epoches,train_loss,label='train_loss')
    plt.plot(epoches,val_loss,label='val_loss')
    plt.title('loss')
    plt.legend()

<matplotlib.legend.Legend at 0x7fd0f9294ba8>
```

Fig 7.6.1 Visualizing loss and accuracy

7.6.1. Model Overfitting

Overfitting refers to a model that models the training data too well. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the model’s ability to generalize.

Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when learning a target function. As such, many nonparametric machine learning algorithms also include parameters or techniques to limit and constrain how much detail the model learns.

7.6.2. Model Underfitting

Underfitting refers to a model that can neither model the training data nor generalize to new data.

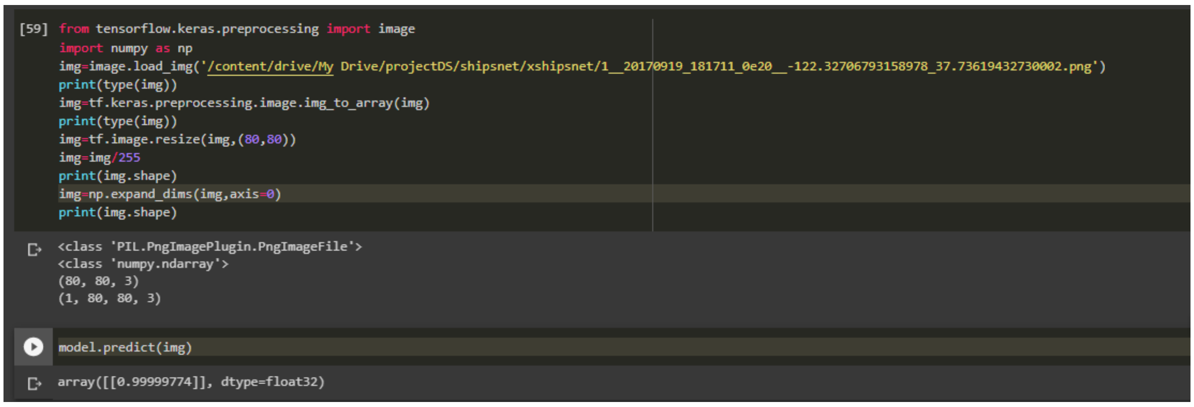
An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data.

Underfitting is often not discussed as it is easy to detect given a good performance metric. The remedy is to move on and try alternate machine learning algorithms. Nevertheless, it does provide a good contrast to the problem of overfitting.

- So, as we can see from the graph above our model is working as expected without overfitting or underfitting the data.

7.7. Predicting from raw data

7.7.1. Predicting from Shipsnet data

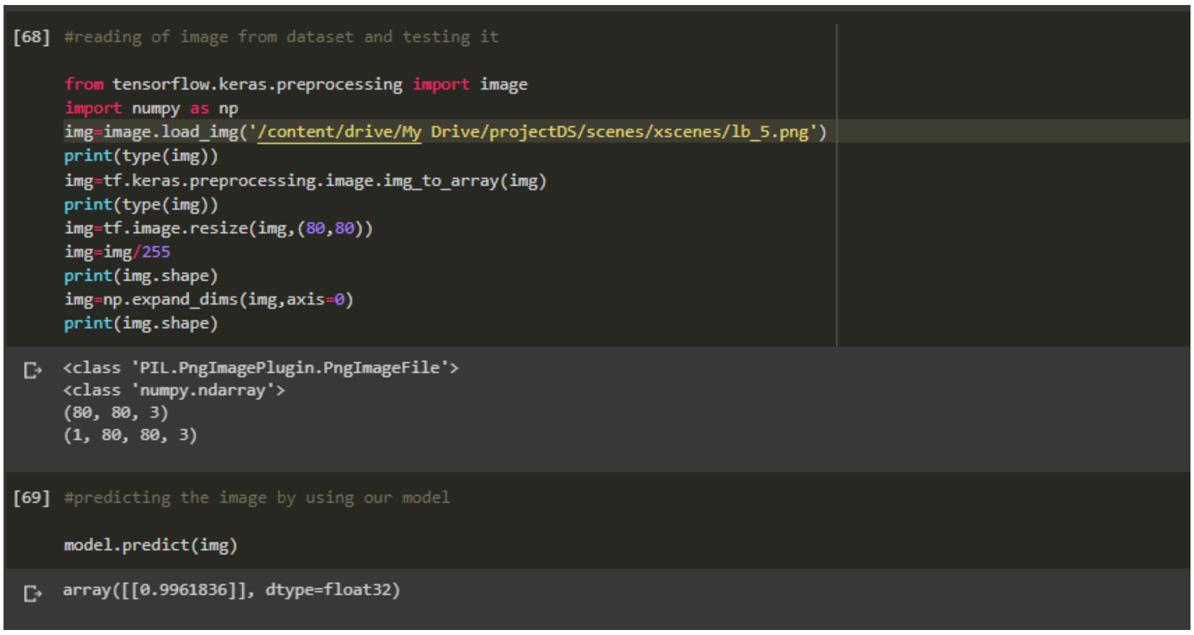


```
[59] from tensorflow.keras.preprocessing import image
     import numpy as np
     img=image.load_img('/content/drive/My Drive/projectDS/shipsnet/xshipsnet/1__20170919_181711_0e20__-122.32706793158978_37.73619432730002.png')
     print(type(img))
     img=tf.keras.preprocessing.image.img_to_array(img)
     print(type(img))
     img=tf.image.resize(img,(80,80))
     img=img/255
     print(img.shape)
     img=np.expand_dims(img,axis=0)
     print(img.shape)

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

     model.predict(img)

     array([[0.99999774]], dtype=float32)
```

Fig 7.7.1 Prediction of shipsnet data

Our model is working as expected as it is giving us a predict value of 0.99.

7.7.2. Predicting from Scenes data



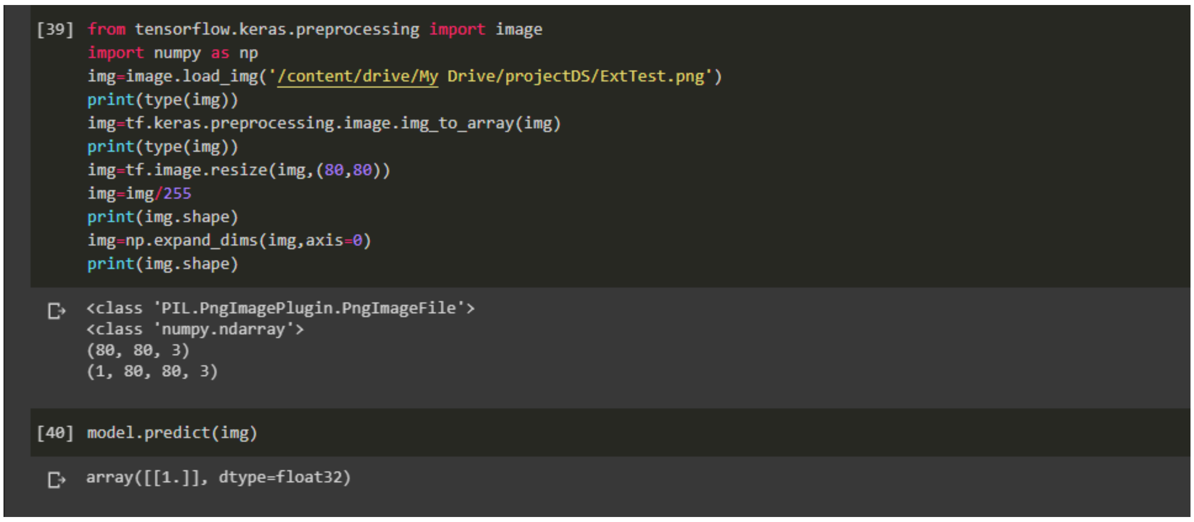
```
[68] #reading of image from dataset and testing it

     from tensorflow.keras.preprocessing import image
     import numpy as np
     img=image.load_img('/content/drive/My Drive/projectDS/scenes/xscenes/lb_5.png')
     print(type(img))
     img=tf.keras.preprocessing.image.img_to_array(img)
     print(type(img))
     img=tf.image.resize(img,(80,80))
     img=img/255
     print(img.shape)
     img=np.expand_dims(img,axis=0)
     print(img.shape)

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

[69] #predicting the image by using our model

     model.predict(img)

     array([[0.9961836]], dtype=float32)
```

Fig 7.7.2 Predicting from scenes data

When we are predicting from scenes dataset, we are successfully converting the image into required shape and predicting it.

We are getting a predict value of 0.996 which means our model is able to identify ships accurately.

7.7.3. Predicting from External image

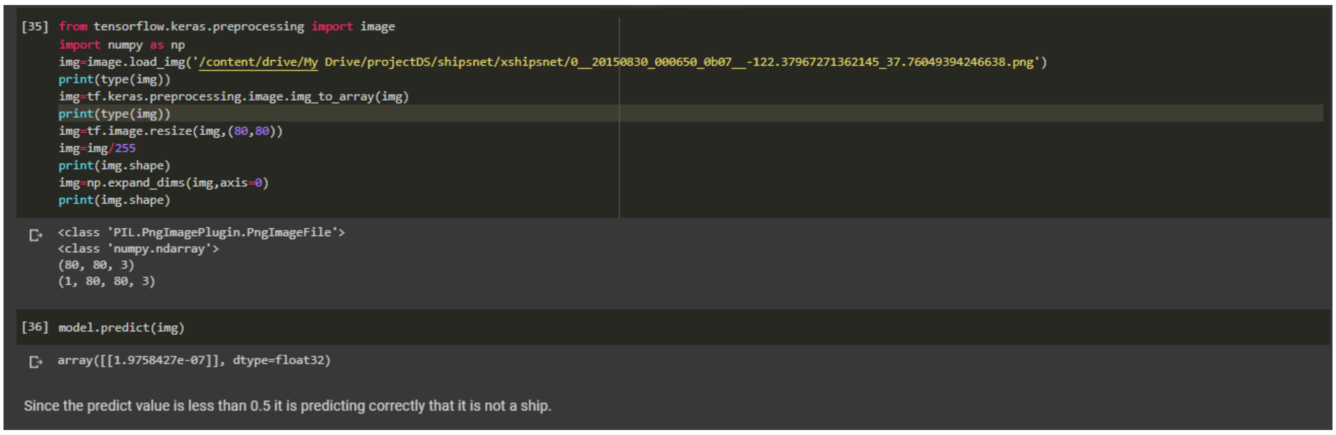


```
[39] from tensorflow.keras.preprocessing import image
     import numpy as np
     img=image.load_img('/content/drive/My Drive/projectDS/ExtTest.png')
     print(type(img))
     img=tf.keras.preprocessing.image.img_to_array(img)
     print(type(img))
     img=tf.image.resize(img,(80,80))
     img=img/255
     print(img.shape)
     img=np.expand_dims(img,axis=0)
     print(img.shape)

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

[40] model.predict(img)

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

Fig 7.7.3 Predicting from external image

When we are giving an external image to predict if it is a ship or not it is able to predict it with utmost accuracy of 1. So, we can say that our model is working as expected.

7.7.4. Predicting using not a ship image:



```
[35] from tensorflow.keras.preprocessing import image
     import numpy as np
     img=image.load_img('/content/drive/My Drive/projectDS/shipsnet/xshipsnet/0__20150830_000650_0b07__-122.37967271362145_37.76049394246638.png')
     print(type(img))
     img=tf.keras.preprocessing.image.img_to_array(img)
     print(type(img))
     img=tf.image.resize(img,(80,80))
     img=img/255
     print(img.shape)
     img=np.expand_dims(img,axis=0)
     print(img.shape)

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

[36] model.predict(img)

     array([[1.9758427e-07]], dtype=float32)
```

Since the predict value is less than 0.5 it is predicting correctly that it is not a ship.

Fig 7.7.4 Predicting using image tagged as not a ship

As we can see here, the predict value is less than 0.5 so we can categorize this image into not a ship. If the predict value is in range of 0.5 to 1 we consider it as a ship and 0 to 0.5 we consider it as not a ship.

8.Conclusion

It is concluded that the model is computed to get good accuracy for unknown data that detects ship images and non-ship images correctly which leads to clear understanding of how the model parameters should be defined to get best accuracy. We can see that CNN algorithm has been successfully implemented in traversing the images and differentiating the images which are tagged as ships and not ships with utmost accuracy of 1. We have also used various optimizers, activation functions and implemented many more techniques to train our model. We have successfully trained and tested our model by using vast amounts of data and managed to get accurate results with respect to the model built.

9. References

- https://www.kaggle.com/rhammell/ships-in-satellite-imagery/home
- https://www.outsource2india.com/software/articles/machine-learning-applications-how-it-works-who-uses-it.asp
- https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148
- https://www.tensorflow.org/js/guide/models_and_layers
- https://www.tensorflow.org/guide/keras/sequential_model
- https://keras.io/api/models/model_training_apis/
- **GitHub link :** https://github.com/Dheeraj0725/AIMLproject