DEEP LEARNING

(Malaria Cells Image Classification)

Summer Internship Report Submitted in partial fulfillment

of the requirement for undergraduate degree of

Bachelor of Technology

In

Computer Science and Engineering

By

M.Sai Srinivas

221710305032

Under the Guidance of

Mr. M.

Assistant Professor

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 **“Malaria Cells Image Classification”** 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 M.Sai Srinivas

Date:20-07-20 221710305032

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled **“Malaria Cells Image Classification”** is being submitted by M.Sai Srinivas (221710305032) 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 2018-19

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

Mr.
Assistant Professor
Department of CSE

Dr.S.Phani Kumar
Professor and HOD
Department of CSE

ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful 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 respected **Dr.S.Phani Kumar,** Head of the Department of Electronics and Communication Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a 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.

Muriki Sai Srinivas

221710305032

ABSTRACT

Deep learning algorithms are used to predict the values from the Image data set by splitting the data set in to train , validation and test and building Deep learning algorithms models of higher accuracy to predict whether infected or uninfected by Malaria is the primary task to be performed on Images of Cells dataset.

Malaria caused by the Plasmodium parasites, is a blood disorder, which is transmitted through the bite of a woman Anopheles mosquito. With almost 240 million cases mentioned each year, the sickness puts nearly forty percentage of the global populace at danger. Macroscopic usually take a look at thick and thin blood smears to identify a disease or a cause and figure it out what weakens them. However, the accuracy depends upon smear quality and awareness in classifying and counting parasite and non-parasite cells. Manual evaluation, which is the gold standard for diagnosis requires various steps to be performed. Moreover, this process leads to overdue and misguided analysis, even when it comes to the hands of expertise. In our project, we aim at building a robust, minimized reliance of humans, sensitive model for automated analysis of Malaria. A category of deep learning models, namely Convolutional Neural Networks, guarantee especially versatile and advanced outcome with end-to-cease attribute extraction and categorization. The precision, unwavering quality, velocity and cost of the methods utilized for malaria examination are key to the diseases' cure and ultimate eradication. In this study, we compare the overall performance of pre- trained CNN primarily based DL model as characteristic extractors closer to classifying parasite and non-parasite cells to aid in progressed sickness screening. The highest quality model layers for attribute extraction from the underlying records, is determined experimentally. The dataset has a variety of Parasite and Non-Parasite pictures of blood samples. To achieve accurate outcome, we have selected certain dominating features such as size, color, shape and cell count from the images which will help in the categorization process. Pre-trained CNNs are used as a promising tool for attribute extraction; this can be determined by the outcome of its statistical validation. Given these developments, automated microscopy could be a very good deal in the chase towards a low-priced, effortless, and dependable method for diagnosing malaria.

Table of Contents:

Contents

LIST OF FIGURES:

1. MACHINE LEARNING:

1.1.INTRODUCTION

Machine Learning (ML) is the scientific study of algorithms and statistical models that computer systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence (AI).

Human designers often produce machines that do not work as well as desired in the environments in which they are used. In fact, certain characteristics of the working environment might not be completely known at design time. Machine learning methods can be used for on-the-job improvement of existing machine designs. The amount of knowledge available about certain tasks might be too large for explicit encoding by humans. Machines that learn this knowledge gradually might be able to capture more of it than humans would want to write down. Environments change over time. Machines that can adapt to a changing environment would reduce the need for constant redesign. New knowledge about tasks is constantly being discovered by humans. Vocabulary changes. There is a constant stream of new events in the world. Continuing redesign of AI systems to conform to new knowledge is impractical, but machine learning methods might be able to track much of it.

1.2.IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and “more items to consider” and “get yourself a little something” on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today’s data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that's in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques.

The process flow depicted here represents how machine learning works

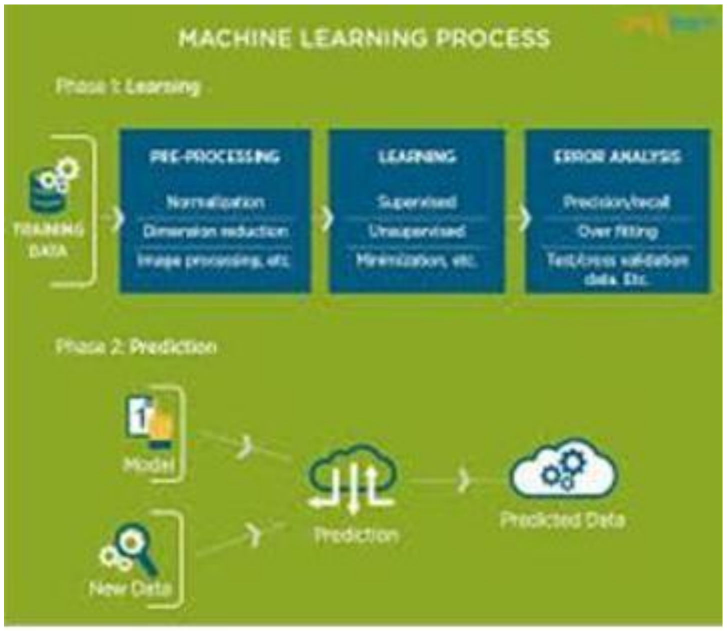


Fig 1.2.1 The Process Flow

1.3.USES OF MACHINE LEARNING:

Artificial Intelligence (AI) is everywhere. Possibility is that you are using it in one way or the other and you don't even know about it. One of the popular applications of AI is Machine Learning (ML), in which computers, software, and devices perform via cognition (very similar to human brain). Herein, we share few examples of machine learning that we use every day and perhaps have no idea that they are driven by ML. These are some the uses and applications of ML

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 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

Smartphones: Samsung Bixby on Samsung S8

Mobile Apps: Google Allo

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.

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 is 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.

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. 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 helps them to compare millions of transactions taking place and distinguish between legitimate or illegitimate transactions taking place between the buyers and sellers**.**

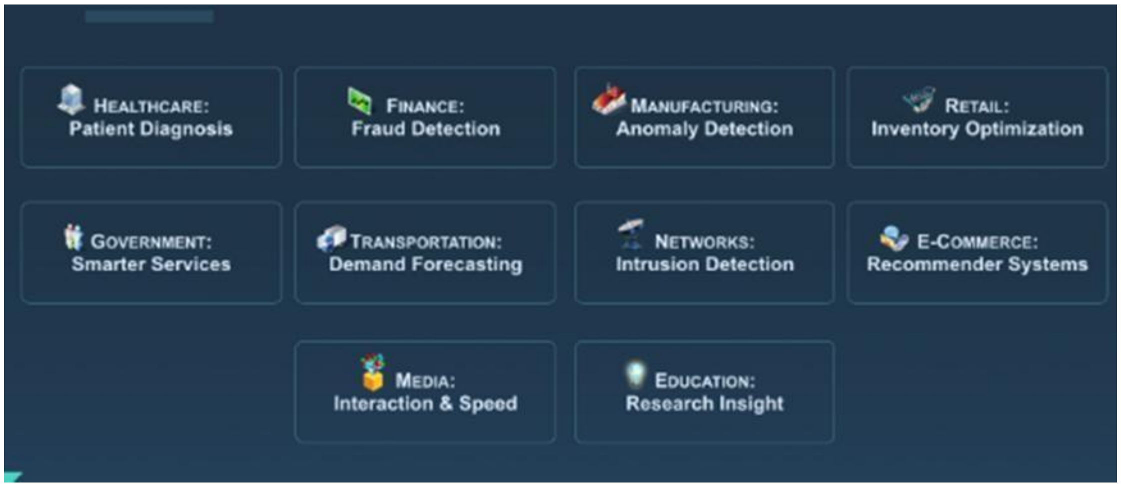


Fig 1.3.1 Uses of Machine learning

1.4.TYPES OF LEARNING ALGORITHMS:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

1.4.1. SUPERVISED LEARNING:

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning. Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data. Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign. Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

UNSUPERVISED LEARNING:

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

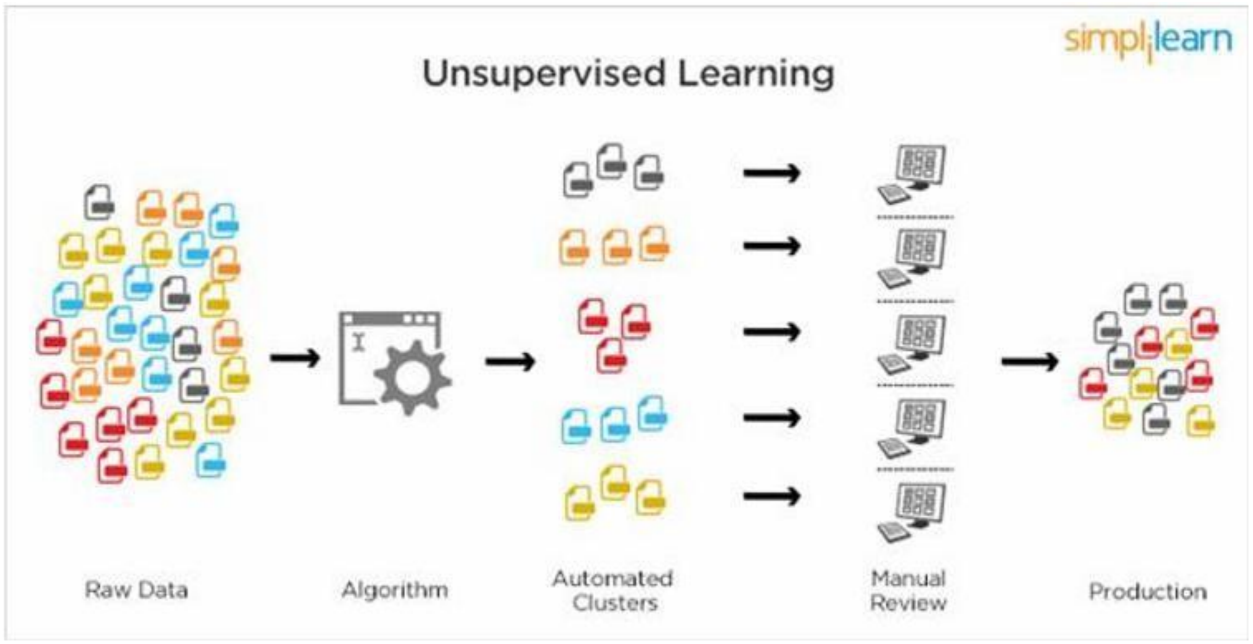


Fig 1.4.2.1 Unsupervised Learning.

Popular techniques where unsupervised learning is used also include self-organizing maps , nearest neighbor mapping , singular value decomposition , and k-means clustering. Basically, online recommendations , identification of data outliers , and segment text topics are all examples of unsupervised learning.

1.4.2. SEMI SUPERVISED LEARNING:

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

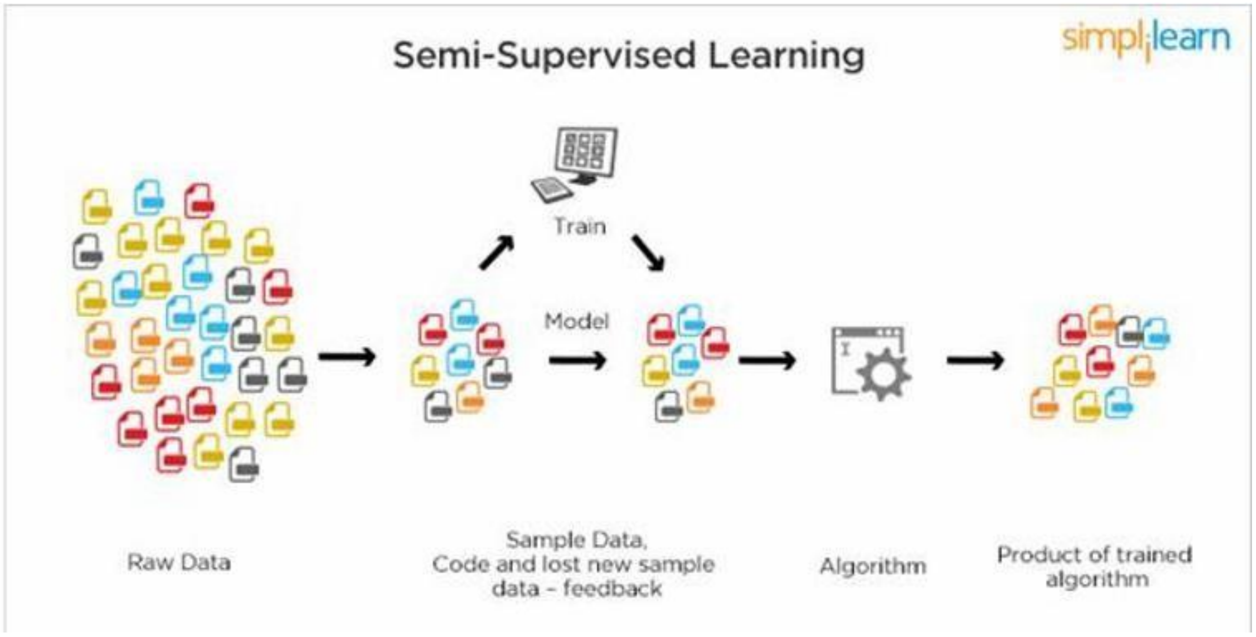


Fig 1.4.3.1 Semi Supervised Learning

1.5.RELATION BETWEEN DATA MINING,MACHINE LEARNING AND DEEP LEARNING:

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

2. DEEP LEARNING:

Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network.

2.1.KEY TAKEAWAYS:

Deep learning is an AI function that mimics the workings of the human brain in processing data for use in detecting objects, recognizing speech, translating languages, and making decisions.

Deep learning AI is able to learn without human supervision, drawing from data that is both unstructured and unlabeled.

Deep learning, a form of machine learning, can be used to help detect fraud or money laundering, among other functions.

2.2.HOW DEEP LEARNING WORKS:

One of the most common AI techniques used for processing big data is machine learning, a self-adaptive algorithm that gets increasingly better analysis and patterns with experience or with newly added data.

If a digital payments company wanted to detect the occurrence or potential for fraud in its system, it could employ machine learning tools for this purpose. The computational algorithm built into a computer model will process all transactions happening on the digital platform, find patterns in the data set, and point out any anomaly detected by the pattern.

Deep learning, a subset of machine learning, utilizes a hierarchical level of artificial neural networks to carry out the process of machine learning. The artificial neural networks are built like the human brain, with neuron nodes connected together like a web. While traditional programs build analysis with data in a linear way, the hierarchical function of deep learning systems enables machines to process data with a nonlinear approach.

2.3. USES:

Deep learning models are widely used in extracting high-level abstract features, providing improved performance over the traditional models, increasing interpretability and also for understanding and processing biological data. To predict splicing action of exons, a fully connected feedforward neural network was designed by Xiong et al. In recent years, CNNs were applied on the DNA dataset directly without the requirement of defining features a priority. Compared to a fully connected network, CNNs use less parameters by applying a convolution operation on the input data space and also parameters are shared between the regions. Hence, large DNA sequence data can be trained using these models and also improved pattern detection accuracy can be obtained. Deepbind, a deep architecture based on CNNs, was proposed by Alipanathi et al. which predicts specificities of DNA and RNA binding proteins. CNNs were also used for predicting chromatin marks from a DNA sequence. Angermueller et al. have incorporated CNNs for predicting DNA methylation states. Like CNNs, other deep architectures were also applied for extracting features from raw DNA sequence data and for processing the data.

2.4.ALGORITHMS IN DEEP LEARNING:

Deep Learning is a new area of Machine Learning research, which has been introduced with the objective of moving Machine Learning closer to one of its original goals: Artificial Intelligence. See these course notes for a brief intro of ml of ai and an introduction to deep learning algorithms

Deep Learning is about learning multiple levels of representation and abstraction that help to make sense of data such as images, sound, and text. For more about deep learning algorithms, see for example:

Things presented here will introduce you to some of the most important deep learning algorithms and will also show you how to run them using Theano. Theano is a python library that makes writing deep learning models easy, and gives the option of training them on a GPU.

The algorithm tutorials have some prerequisites. You should know some python, and be familiar with NumPy. Since this tutorial is about using Theano.

The purely supervised learning algorithms are meant to be read in order:

1. Logistic Regression - using Theano for something simple
2. Multilayer perception - introduction to layers
3. Deep Convolution Network- a simplified version of LeNet5

The unsupervised and semi-supervised learning algorithms can be read in any order (the auto-encoders can be read independently of the RBM/DBN thread):

- Auto Encoders, Denoising Autoencoders - description of autoencoders
- Stacked Denoising Auto-Encoders - easy steps into unsupervised pre-training for deep nets
- Restricted Boltzmann Machines - single layer generative RBM model
- Deep Belief Networks - unsupervised generative pre-training of stacked RBMs followed by supervised fine-tuning

Building towards including the mcRBM model, we have a new tutorial on sampling from energy models:

- HMC Sampling - hybrid (aka Hamiltonian) Monte-Carlo sampling with scan()

Building towards including the Contractive auto-encoders tutorial, we have the code for now:

- Contractive auto-encoders code - There is some basic doc in the code.

Recurrent neural networks with word embeddings and context window:

- Semantic Parsing of Speech using Recurrent Net

LSTM network for sentiment analysis:

- LSTM network

Energy-based recurrent neural network (RNN-RBM):

- Modeling and generating sequences of polyphonic music

Segmentation for medical imagery (meant to be read in order):

- Fully Convolutional Networks (FCN) for 2D segmentation
- U-Net

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.

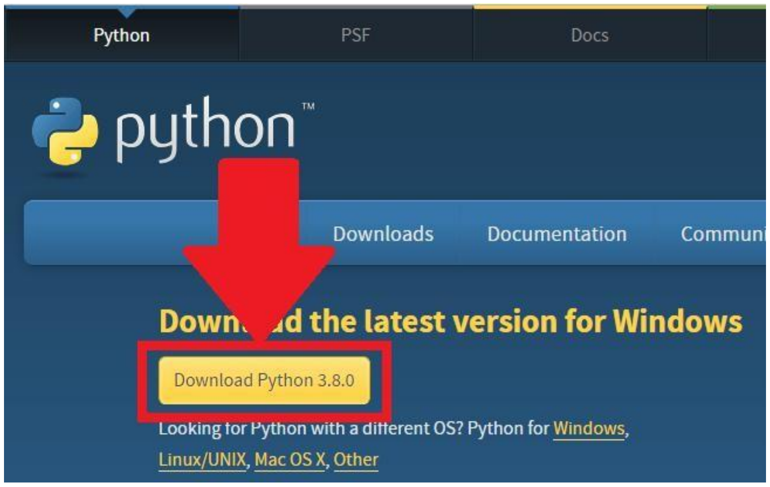
Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

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(USING PYTHON IDLE):

- 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 also.

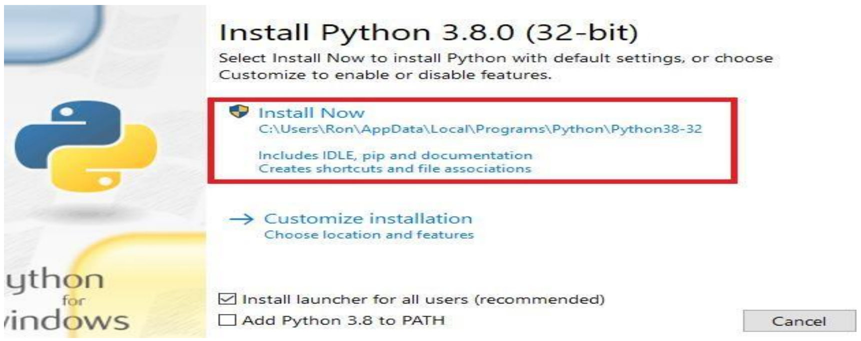


Down ad the latest version for Windows

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Fig 3.2.1.1 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**



Install Python 3.8.0 (32-bit)

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

Install Now
C:\Users\Ron\AppData\Local\Programs\Python\Python38-32

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☐ Add Python 3.8 to PATH

downloading

Fig 3.2.1.2 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.

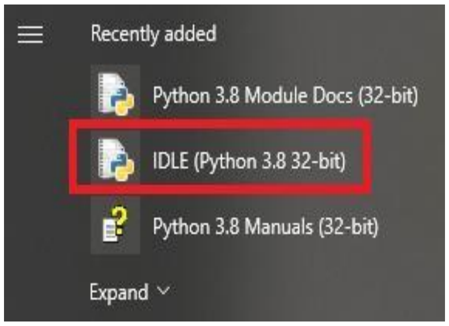
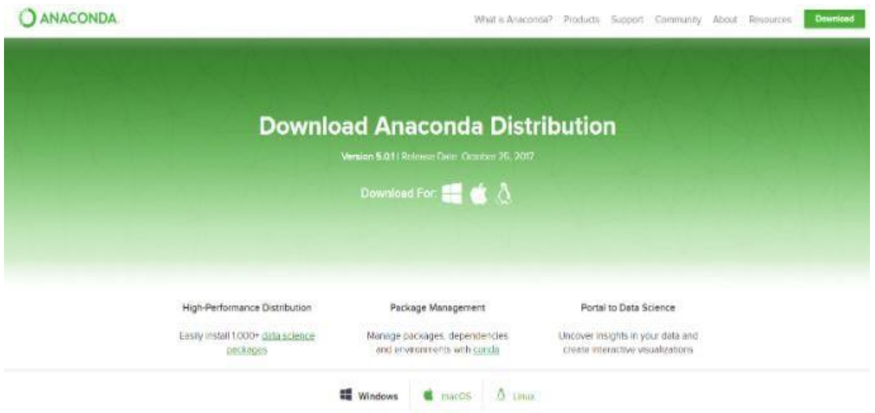


Fig 3.2.1.3 IDLE

2.2.2 Python Installation 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



ANACONDA

What is Anaconda? Products Support Community About Resources Download

Download Anaconda Distribution

Version 5.0.1 | Release Date: October 26, 2017

Download For:

High-Performance Distribution

Easily install 1,000+ data science packages

Package Management

Manage packages, dependencies and environments with conda

Portal to Data Science

Uncover insights in your data and create interactive visualizations

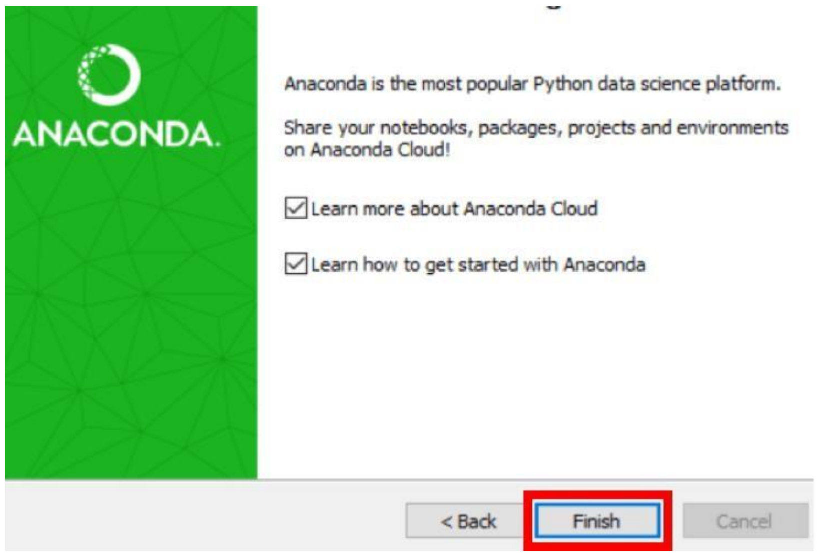
Windows macOS Linux

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)

iv. Click finish

v. Open Jupyter notebook



ANACONDA.

Anaconda is the most popular Python data science platform.

Share your notebooks, packages, projects and environments on Anaconda Cloud!

☑ Learn more about Anaconda Cloud

☑ Learn how to get started with Anaconda

< Back Finish Cancel

Fig 3.2.2.1 After installation

Fig 3.2.2.2 Jupyter notebook

3.3.FEATURES:

i. **Readable:** Python is a very readable language.

ii. **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

iii. **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.

iv. **Open Source:** Python is an open source programming language.

v. **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.

vi. **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.

vii. **Supports exception handling:** If you are new, you may wonder what is an exception? An exception is an event that can occur during program exception 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.

viii. **Advanced features:** Supports generators and list comprehensions. We will cover these features later.

ix. **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 NUMBER:

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

Python supports four different numerical types −

- **int (signed integers)** − They are often called just integers or ints, 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 10^2 = 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 can't 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 returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

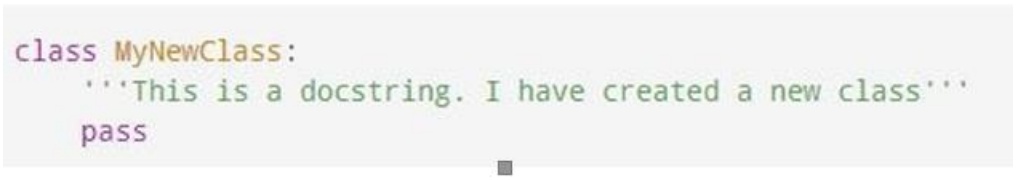
3.5.2. CALLING A FUNCTION:

- 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.OOP'S CONCEPT:

3.6.1. CLASS:

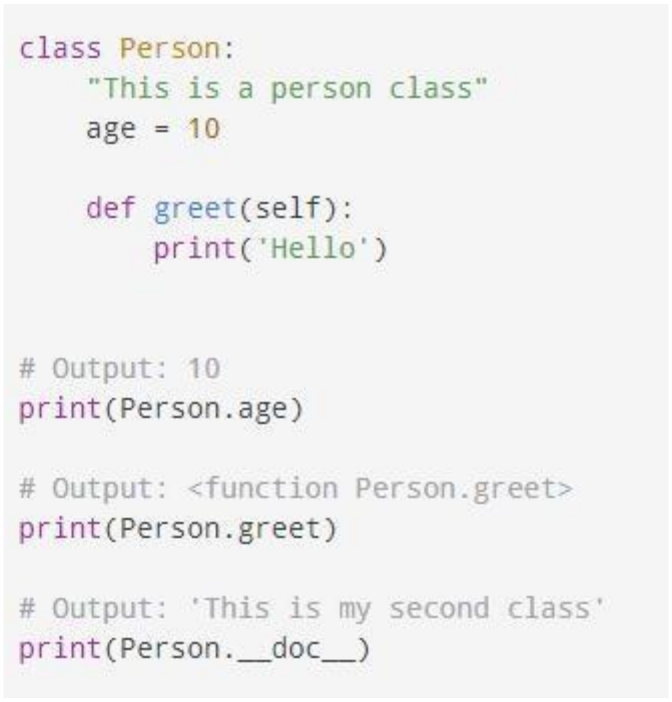
- Python is an object-oriented programming language. Unlike procedure-oriented programming, where the main emphasis is on functions, object-oriented programming stresses on objects.
- An object is simply a collection of data (variables) and methods (functions) that act on those data. Similarly, a class 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 MyNewClass:
    '''This is a docstring. I have created a new class'''
    pass
```

Fig 3.6.1.1 Class defining

- As soon as we define a class, a new class object is created with the same name. This class object allows us to access the different attributes as well as to instantiate new objects of that class



```
class Person:
    "This is a person class"
    age = 10

    def greet(self):
        print('Hello')


# Output: 10
print(Person.age)

# Output: <function Person.greet>
print(Person.greet)

# Output: 'This is my second class'
print(Person.__doc__)
```

Fig 3.6.1.2 Example of class

4. CASE STUDY:

4.1.PROBLEM STATEMENT:

Create the model to train and test the classification of parasitized and uninfected Malaria Cells images

4.2.DATA SET:

The dataset is taken from the Kaggle. The dataset contains several images of the parasitized and uninfected cell images of malaria

Link for the dataset https://www.kaggle.com/iarunava/cell-images-for-detecting-malaria

The dataset contains 2 folders

- Parasitized
- Uninfected

And there are 27,558 images in both Parasitized and Uninfected folders

Kaggle API command to download the dataset is "kaggle datasets download -d iarunava/cell-images-for-detecting-malaria"

4.3.OBJECTIVE OF THE CASE STUDY:

Objective of problem is to classify the images of malaria cells i.e.; cells are parasitized or uninfected

4.4.PROJECT REQUIREMENTS:

4.4.1. PACKAGES USED:

- **NumPy:**

  NumPy is a python library used for working with arrays. 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 ndata, 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.

- **OS:**

  The OS module in python provides functions for interacting with the operating system. OS, comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality. The *os* and *os. path* modules include many functions to interact with the file system.

- **ZipFile:**

  The ZIP file format is a common archive and compression standard. This module provides tools to create, read, write, append, and list a ZIP file. Any advanced use of this module will require an understanding of the format, as defined in PKZIP Application Note. This module does not currently handle multi-disk ZIP files.

- **Seaborn:**

  Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

- **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 IPython 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 library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. Since neural networks, but also other ML algorithms, mostly work with matrix multiplications, it is much quicker to run them on Graphical Processing Units (GPUs), rather than on standard Central Processing Units.

5. MODEL BUILDING:

5.1.PREPROCESSING OF THE DATA:

Preprocessing of the data actually involves the following steps:

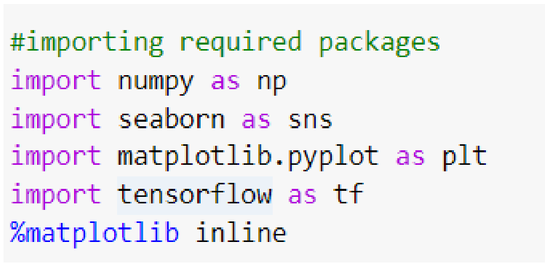
5.2.GETTING THE DATASET:

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

5.3.IMPORTING THE LIBRARIES:

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

LOADING THE REQUIRED PACKAGES

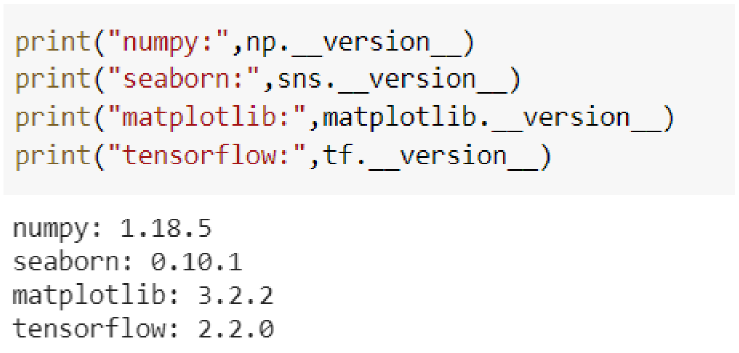


```
#importing required packages
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
%matplotlib inline
```

Fig 5.3.1 loading packages

5.4.VERSIONS OF THE PACKAGES:

The versions of the packages are found by following command



```
print("numpy:",np.__version__)
print("seaborn:",sns.__version__)
print("matplotlib:",matplotlib.__version__)
print("tensorflow:",tf.__version__)
```

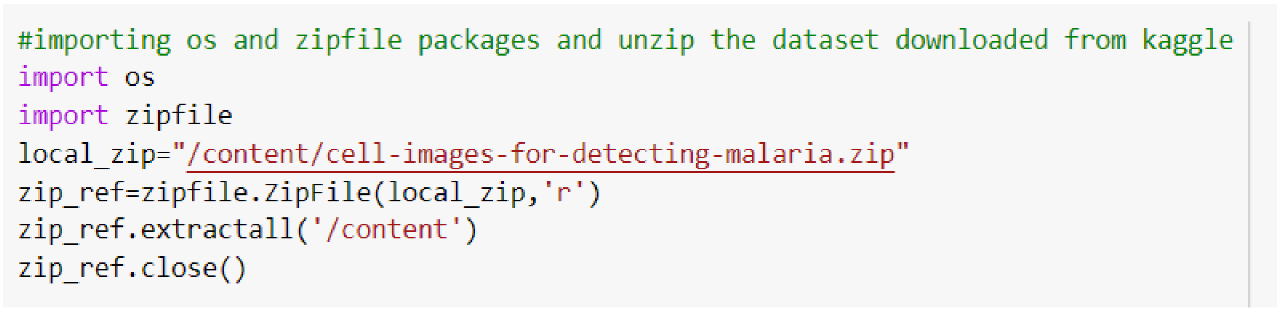
```
numpy: 1.18.5
seaborn: 0.10.1
matplotlib: 3.2.2
tensorflow: 2.2.0
```

Fig 5.4.1 versions of packages

5.5.IMPORTING AND UNZIPPING DATA SET ZIP FOLDER:

OS in python is used to access and peek into the files or folders at the different location using the location path.

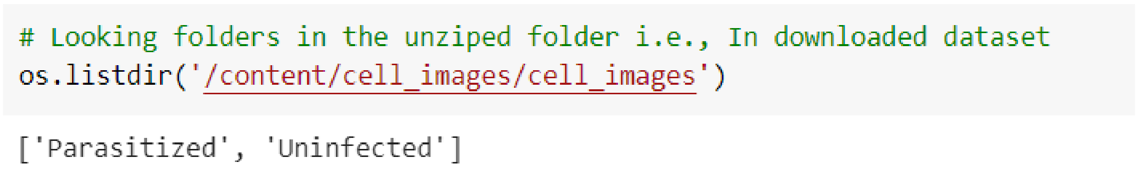
Zipfile in python is used to unzip the zip folders and this unzip is done using ZipFile and extractall function and we can specify the unzip will be read or write.



```
#importing os and zipfile packages and unzip the dataset downloaded from kaggle
import os
import zipfile
local_zip="/content/cell-images-for-detecting-malaria.zip"
zip_ref=zipfile.ZipFile(local_zip,'r')
zip_ref.extractall('/content')
zip_ref.close()
```

Fig 5.5.1 Importing and Unziping the dataset folder

Folders in the Zipped folders

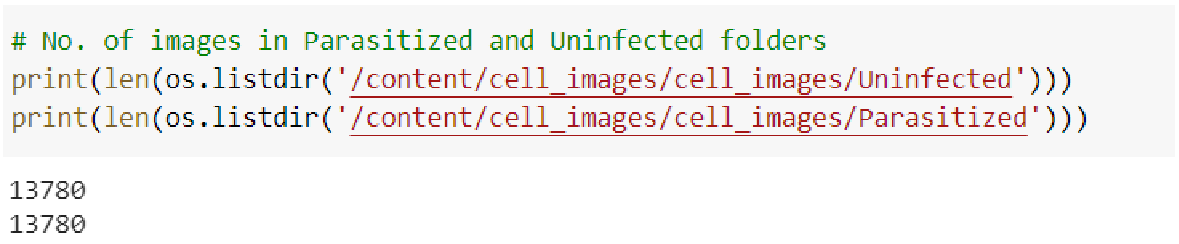


```
# Looking folders in the unziped folder i.e., In downloaded dataset
os.listdir('/content/cell_images/cell_images')
```

```
['Parasitized', 'Uninfected']
```

Fig 5.5.2 Folders in the unzipped dataset

No. of Images in the Dataset

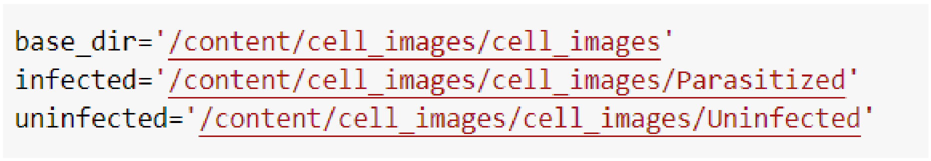


```
# No. of images in Parasitized and Uninfected folders
print(len(os.listdir('/content/cell_images/cell_images/Uninfected')))
print(len(os.listdir('/content/cell_images/cell_images/Parasitized')))
```

```
13780
13780
```

Fig 5.5.3 No. of Images in the Parasitized and Uninfected folders

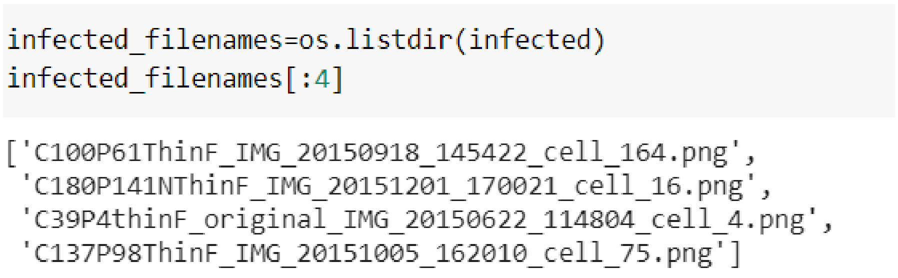
Creating a variables for the base , infected and uninfected paths



```
base_dir='/content/cell_images/cell_images'
infected='/content/cell_images/cell_images/Parasitized'
uninfected='/content/cell_images/cell_images/Uninfected'
```

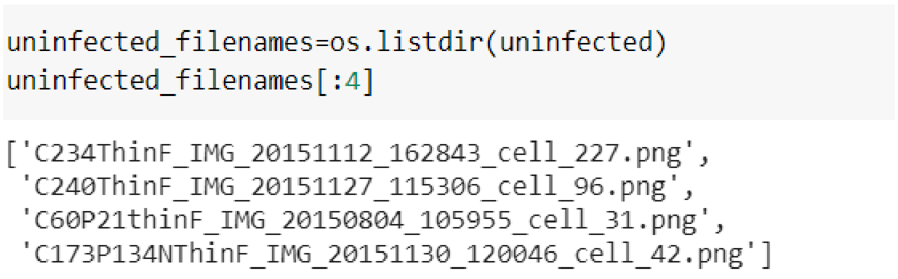
Fig 5.5.4 Variables of the base , infected and uninfected paths

Creating the list of (image) filenames of the infected and uninfected images



```
infected_filenames=os.listdir(infected)
infected_filenames[:4]
```

```
['C100P61ThinF_IMG_20150918_145422_cell_164.png',
 'C180P141NThinF_IMG_20151201_170021_cell_16.png',
 'C39P4thinF_original_IMG_20150622_114804_cell_4.png',
 'C137P98ThinF_IMG_20151005_162010_cell_75.png']
```



```
uninfected_filenames=os.listdir(uninfected)
uninfected_filenames[:4]
```

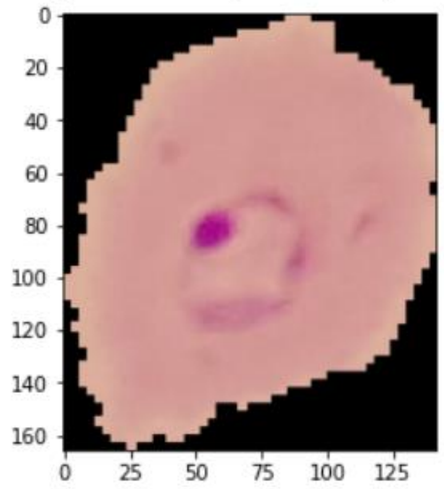
```
['C234ThinF_IMG_20151112_162843_cell_227.png',
 'C240ThinF_IMG_20151127_115306_cell_96.png',
 'C60P21thinF_IMG_20150804_105955_cell_31.png',
 'C173P134NThinF_IMG_20151130_120046_cell_42.png']
```

Fig 5.5.5 List first four filenames of images in infected and uninfected

Displaying first images from the Infected and Uninfected folders

```
plt.imshow(plt.imread(os.path.join(infected,infected_filenames[0])))
```

```
<matplotlib.image.AxesImage at 0x7f77e333ffd0>
```



```
plt.imshow(plt.imread(os.path.join(uninfected,uninfected_filenames[0])))
```

```
<matplotlib.image.AxesImage at 0x7f77e325eb38>
```

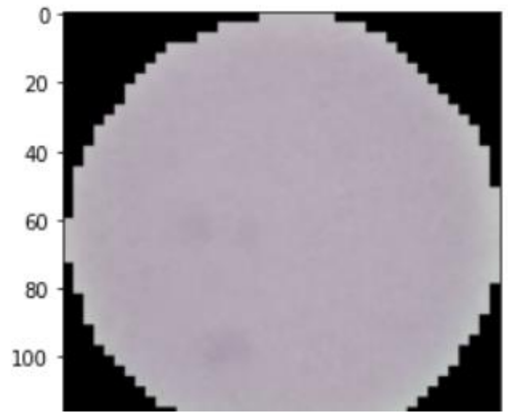


Fig 5.5.6 First Images from the Infected and Uninfected

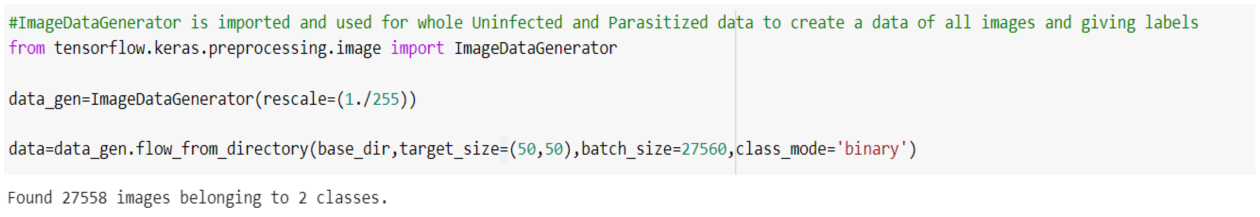
6. DATA PREPROCESSING/FEATURE ENGINEERING AND EDA:

6.1.IMAGEDATAGENERATOR:

TensorFlow in python provides ImageDataGenerator which helps in rescaling the images and creating a batch of image with the labeling

ImageDataGenerator is imported from the tensorflow.keras.preprocessing.image. Created an object for the ImageDataGenerator with the rescale attribute where the image is rescaled to 1 or 255

This ImageDataGenerator has flow_from_directory function which will help in creating a data by using base_dir , target_size , batch_size and class_mode as its attributes. My base directory is base_dir , target_size is (50,50) , batch_size is 27560 and class_mode is binary. So , it will consider Uninfected and Parasitized as 2 different class since my class mode is binary and had 2 different class it will consider 1st folder as 1.0 and 2nd as 0.0 and the image size is (50,50) and there will be 27558 images in the data.

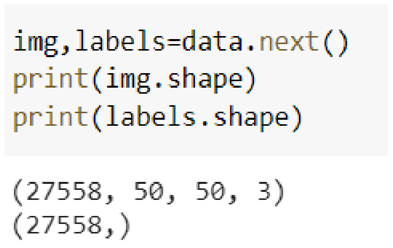


```
#ImageDataGenerator is imported and used for whole Uninfected and Parasitized data to create a data of all images and giving labels
from tensorflow.keras.preprocessing.image import ImageDataGenerator

data_gen=ImageDataGenerator(rescale=(1./255))

data=data_gen.flow_from_directory(base_dir,target_size=(50,50),batch_size=27560,class_mode='binary')
```

```
Found 27558 images belonging to 2 classes.
```

Fig 6.1.1 Creating a data using ImageDataGenerator

Shapes of the data Created:

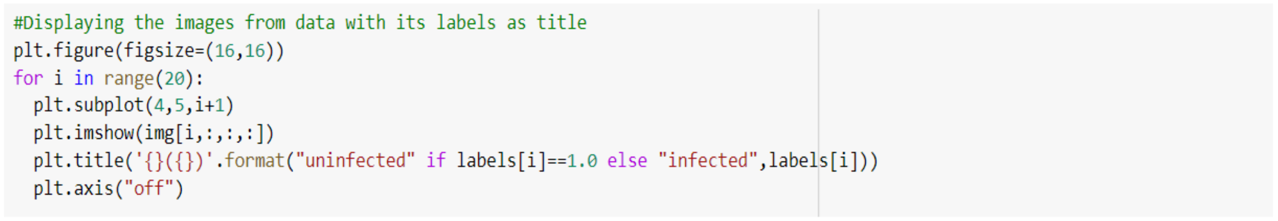


```
img,labels=data.next()
print(img.shape)
print(labels.shape)
```

```
(27558, 50, 50, 3)
(27558,)
```

Fig 6.1.2 Shape of the created data

Displaying the Images from the data:



```
#Displaying the images from data with its labels as title
plt.figure(figsize=(16,16))
for i in range(20):
  plt.subplot(4,5,i+1)
  plt.imshow(img[i,:,:,:])
  plt.title('{}({})'.format("uninfected" if labels[i]==1.0 else "infected",labels[i]))
  plt.axis("off")
```

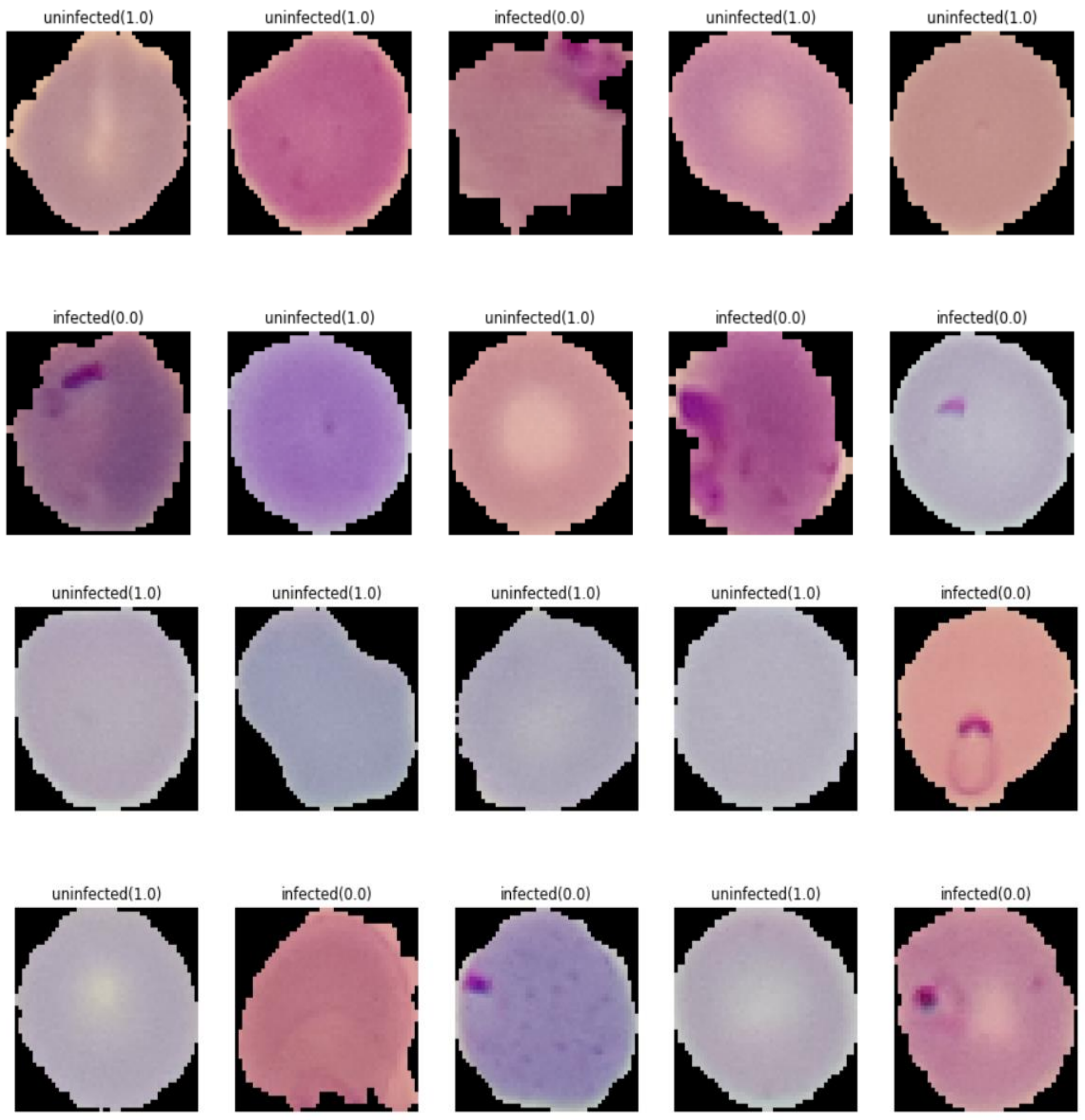
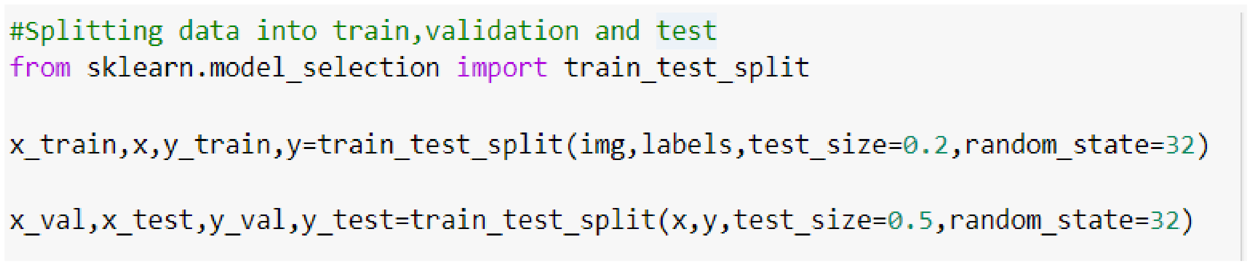


Fig 6.1.3 Images from the data

6.2.SPLITTING DATA INTO TRAIN , VALIDATION AND TEST:

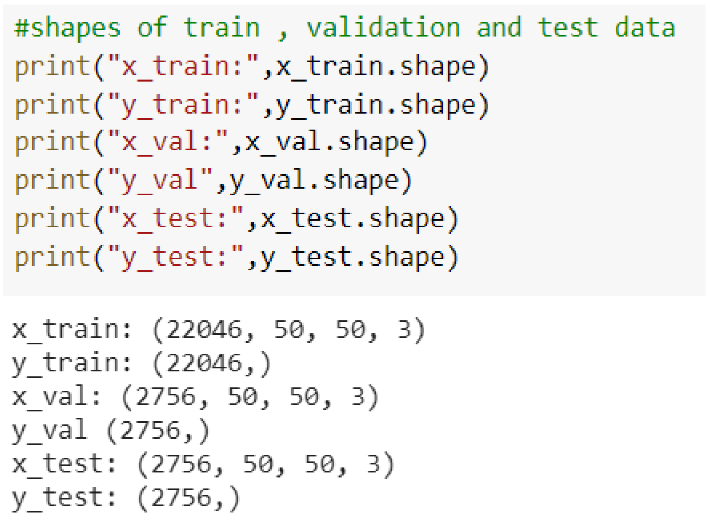
Data Created by the ImageDataGenerator is divided into 3 parts one is train , another is Validation and finally one is Test. 80% of the data is taken as training data and rest of 20% is data is validation and test data. In that 20% data, validation data is 10% and remaining 10% is test data.

Data is divided by using the train_test_split and it is imported from the sklearn.model_selection.



```
#Splitting data into train,validation and test
from sklearn.model_selection import train_test_split

x_train,x,y_train,y=train_test_split(img,labels,test_size=0.2,random_state=32)

x_val,x_test,y_val,y_test=train_test_split(x,y,test_size=0.5,random_state=32)
```

Fig 6.2.1 Splitting Data into train , validation and test



```
#shapes of train , validation and test data
print("x_train:",x_train.shape)
print("y_train:",y_train.shape)
print("x_val:",x_val.shape)
print("y_val",y_val.shape)
print("x_test:",x_test.shape)
print("y_test:",y_test.shape)
```

```
x_train: (22046, 50, 50, 3)
y_train: (22046,)
x_val: (2756, 50, 50, 3)
y_val (2756,)
x_test: (2756, 50, 50, 3)
y_test: (2756,)
```

Fig 6.2.2 Shapes of the train , validation and test

7. MODEL BUILDING AND EVALUATION:

7.1.BRIEF DESCRIPTION ABOUT THE CNN MODEL:

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction.

A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

Keras Conv2D is a 2D Convolution Layer, this layer creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs. Conv2D filters extend through the three channels in an image (Red, Green, and Blue). After the convolutions are performed individually for each channel, they are added up to get the final convoluted image. The output of a filter after a convolution operation is called a feature map.

Kernel: In image processing kernel is a convolution matrix or masks which can be used for blurring, sharpening, embossing, edge detection, and more by doing a convolution between a kernel and an image.

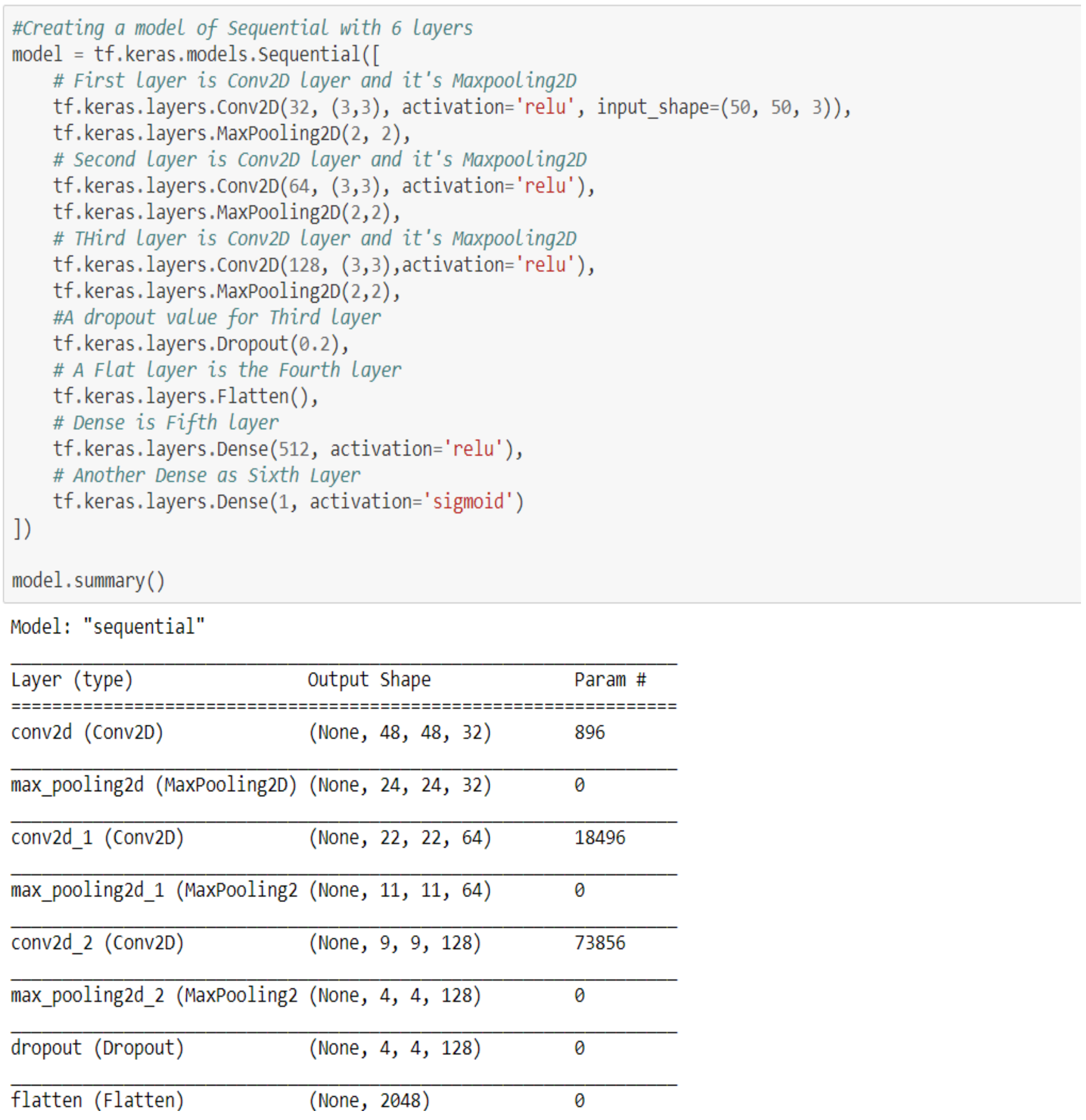
Maximum pooling, or max pooling, is a pooling operation that calculates the maximum, or largest, value in each patch of each feature map. The results are down sampled or pooled feature maps that highlight the most present feature in the patch, not the average presence of the feature in the case of average pooling.

Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped-out” randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

A dense layer is just a regular layer of neurons in a neural network. Each neuron receives input from all the neurons in the previous layer, thus densely connected.

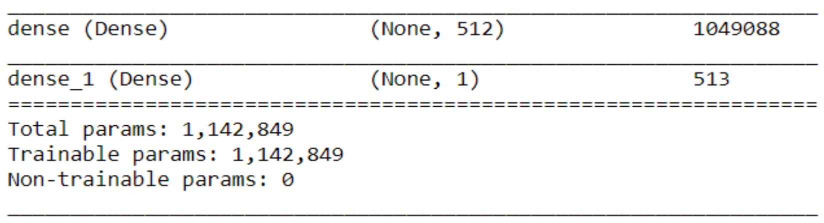
7.2.BUILDING THE CNN MODEL:

In our CNN model we used conv2d , maxpooling2d , dropout , flatten and dense function as the layers. 6 layers are used to build the model. In That first 3 layers are conv2d with maxpooling2d and relu(Rectified Linear Unit) is the activation and finally used 32 , 64 , 128 filters respective layers and added dropout layer of 0.2 for the last layer. And then one layer is flatten layer , another is dense layer with 512 filters and relu activation and final layer is also a dense layer with 1 output filter and sigmoid activation.



```
#Creating a model of Sequential with 6 layers
model = tf.keras.models.Sequential([
    # First layer is Conv2D layer and it's Maxpooling2D
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(50, 50, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    # Second layer is Conv2D layer and it's Maxpooling2D
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # THird layer is Conv2D layer and it's Maxpooling2D
    tf.keras.layers.Conv2D(128, (3,3),activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    #A dropout value for Third layer
    tf.keras.layers.Dropout(0.2),
    # A Flat layer is the Fourth layer
    tf.keras.layers.Flatten(),
    # Dense is Fifth layer
    tf.keras.layers.Dense(512, activation='relu'),
    # Another Dense as Sixth Layer
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 48, 48, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 24, 24, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 22, 22, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 11, 11, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 9, 9, 128)         73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout (Dropout)            (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
```

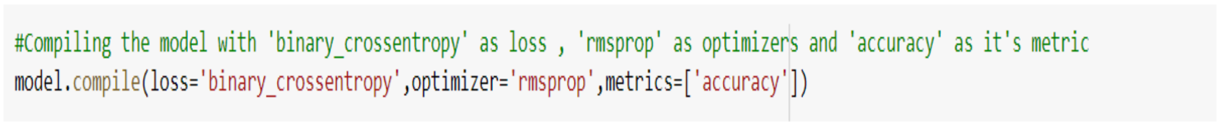


```
_________________________________________________________________
dense (Dense)                (None, 512)               1049088
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 1,142,849
Trainable params: 1,142,849
Non-trainable params: 0
_________________________________________________________________
```

Fig 7.2.1 Creating model and its summary

7.3.TRAINING THE DATA:

Model is compiled by using compile function from model object of CNN model and this compile will binary_crossentropy as loss attribute , rmsprop as optimizer attribute and accuracy as metric attribute

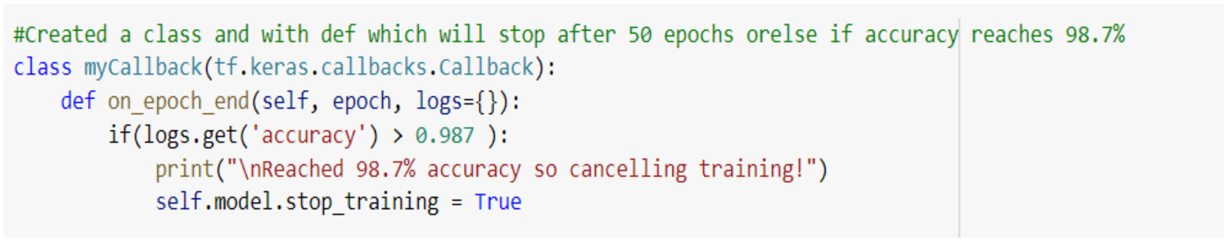


```
#Compiling the model with 'binary_crossentropy' as loss , 'rmsprop' as optimizers and 'accuracy' as it's metric
model.compile(loss='binary_crossentropy',optimizer='rmsprop',metrics=['accuracy'])
```

Fig 7.3.1 Compiling the model created

Created a Class with mycallback as its name and a function in it with on_epoch_end as name.

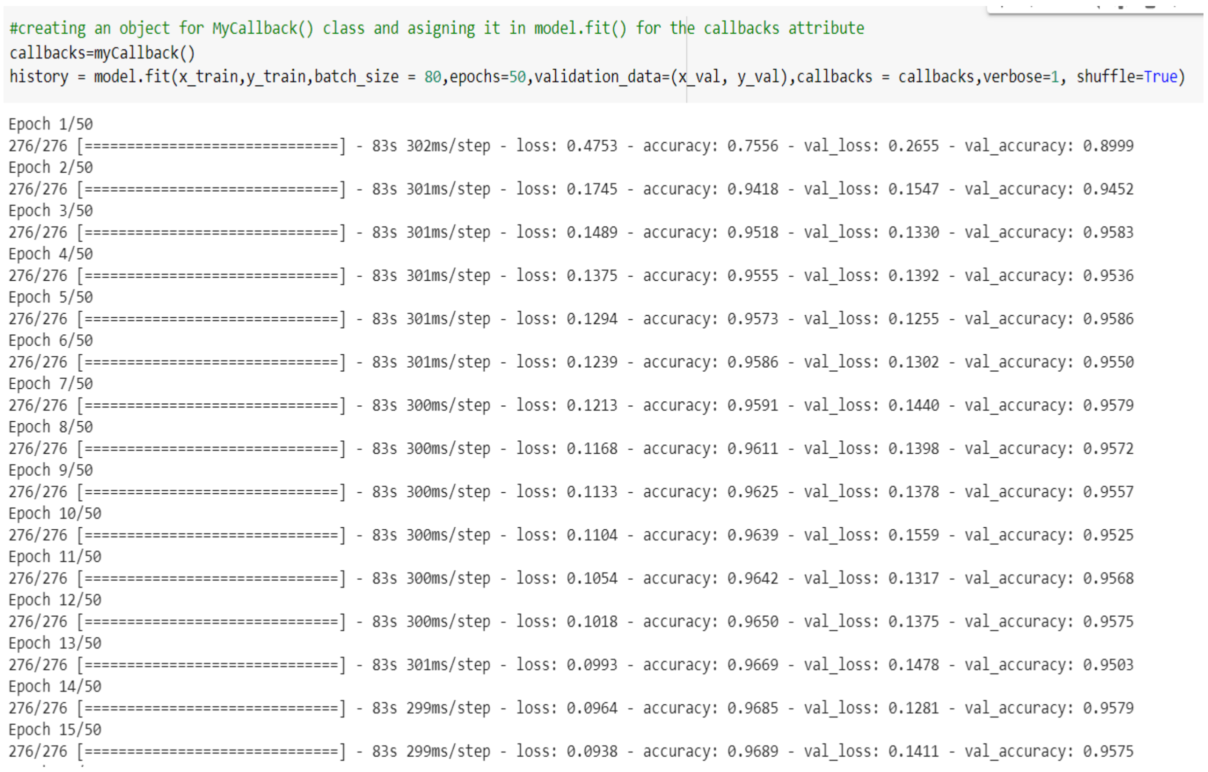
This function will make sure that training will be stopped if model achieves 98.7% accuracy before the number of epochs completed.



```
#Created a class and with def which will stop after 50 epochs orelse if accuracy reaches 98.7%
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('accuracy') > 0.987 ):
            print("\nReached 98.7% accuracy so cancelling training!")
            self.model.stop_training = True
```

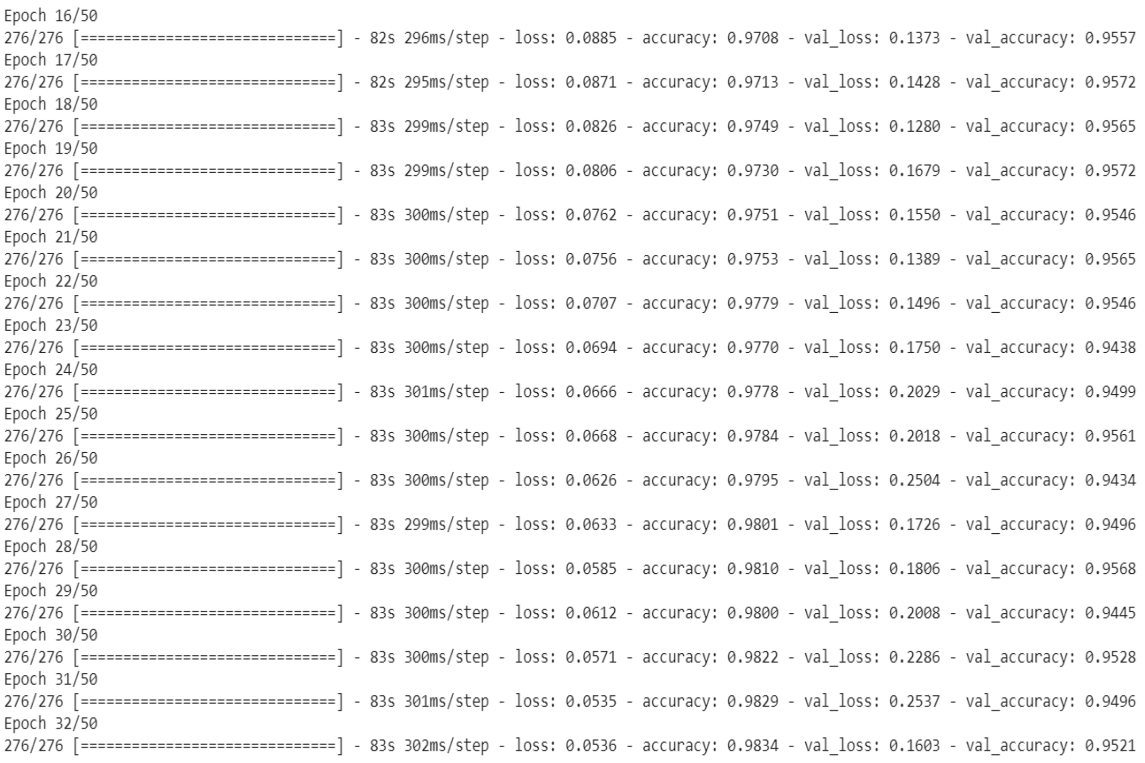
Fig 7.3.2 mycallback class is created

Training the model using train data and validation data. These train and validation data are divided from the data in preprocessing of data. Training the model is done by using fit function in the model object and it takes x_train , y_train , batch_size , epochs , x_val , y_val , callback , verbose and shuffle as input. And our batch_size is 80 , epochs are 50 , callback takes object of mycallback object , verbose is 1 and shuffle is True.

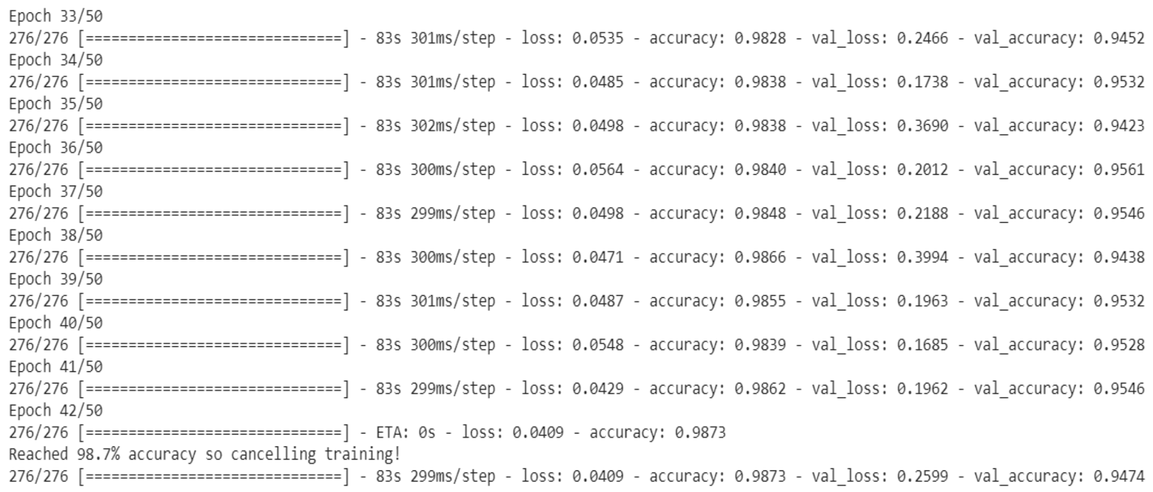


```
#creating an object for MyCallback() class and asigning it in model.fit() for the callbacks attribute
callbacks=myCallback()
history = model.fit(x_train,y_train,batch_size = 80,epochs=50,validation_data=(x_val, y_val),callbacks = callbacks,verbose=1, shuffle=True)
```

```
Epoch 1/50
276/276 [==============================] - 83s 302ms/step - loss: 0.4753 - accuracy: 0.7556 - val_loss: 0.2655 - val_accuracy: 0.8999
Epoch 2/50
276/276 [==============================] - 83s 301ms/step - loss: 0.1745 - accuracy: 0.9418 - val_loss: 0.1547 - val_accuracy: 0.9452
Epoch 3/50
276/276 [==============================] - 83s 301ms/step - loss: 0.1489 - accuracy: 0.9518 - val_loss: 0.1330 - val_accuracy: 0.9583
Epoch 4/50
276/276 [==============================] - 83s 301ms/step - loss: 0.1375 - accuracy: 0.9555 - val_loss: 0.1392 - val_accuracy: 0.9536
Epoch 5/50
276/276 [==============================] - 83s 301ms/step - loss: 0.1294 - accuracy: 0.9573 - val_loss: 0.1255 - val_accuracy: 0.9586
Epoch 6/50
276/276 [==============================] - 83s 301ms/step - loss: 0.1239 - accuracy: 0.9586 - val_loss: 0.1302 - val_accuracy: 0.9550
Epoch 7/50
276/276 [==============================] - 83s 300ms/step - loss: 0.1213 - accuracy: 0.9591 - val_loss: 0.1440 - val_accuracy: 0.9579
Epoch 8/50
276/276 [==============================] - 83s 300ms/step - loss: 0.1168 - accuracy: 0.9611 - val_loss: 0.1398 - val_accuracy: 0.9572
Epoch 9/50
276/276 [==============================] - 83s 300ms/step - loss: 0.1133 - accuracy: 0.9625 - val_loss: 0.1378 - val_accuracy: 0.9557
Epoch 10/50
276/276 [==============================] - 83s 300ms/step - loss: 0.1104 - accuracy: 0.9639 - val_loss: 0.1559 - val_accuracy: 0.9525
Epoch 11/50
276/276 [==============================] - 83s 300ms/step - loss: 0.1054 - accuracy: 0.9642 - val_loss: 0.1317 - val_accuracy: 0.9568
Epoch 12/50
276/276 [==============================] - 83s 300ms/step - loss: 0.1018 - accuracy: 0.9650 - val_loss: 0.1375 - val_accuracy: 0.9575
Epoch 13/50
276/276 [==============================] - 83s 301ms/step - loss: 0.0993 - accuracy: 0.9669 - val_loss: 0.1478 - val_accuracy: 0.9503
Epoch 14/50
276/276 [==============================] - 83s 299ms/step - loss: 0.0964 - accuracy: 0.9685 - val_loss: 0.1281 - val_accuracy: 0.9579
Epoch 15/50
276/276 [==============================] - 83s 299ms/step - loss: 0.0938 - accuracy: 0.9689 - val_loss: 0.1411 - val_accuracy: 0.9575
```



```
Epoch 16/50
276/276 [==============================] - 82s 296ms/step - loss: 0.0885 - accuracy: 0.9708 - val_loss: 0.1373 - val_accuracy: 0.9557
Epoch 17/50
276/276 [==============================] - 82s 295ms/step - loss: 0.0871 - accuracy: 0.9713 - val_loss: 0.1428 - val_accuracy: 0.9572
Epoch 18/50
276/276 [==============================] - 83s 299ms/step - loss: 0.0826 - accuracy: 0.9749 - val_loss: 0.1280 - val_accuracy: 0.9565
Epoch 19/50
276/276 [==============================] - 83s 299ms/step - loss: 0.0806 - accuracy: 0.9730 - val_loss: 0.1679 - val_accuracy: 0.9572
Epoch 20/50
276/276 [==============================] - 83s 300ms/step - loss: 0.0762 - accuracy: 0.9751 - val_loss: 0.1550 - val_accuracy: 0.9546
Epoch 21/50
276/276 [==============================] - 83s 300ms/step - loss: 0.0756 - accuracy: 0.9753 - val_loss: 0.1389 - val_accuracy: 0.9565
Epoch 22/50
276/276 [==============================] - 83s 300ms/step - loss: 0.0707 - accuracy: 0.9779 - val_loss: 0.1496 - val_accuracy: 0.9546
Epoch 23/50
276/276 [==============================] - 83s 300ms/step - loss: 0.0694 - accuracy: 0.9770 - val_loss: 0.1750 - val_accuracy: 0.9438
Epoch 24/50
276/276 [==============================] - 83s 301ms/step - loss: 0.0666 - accuracy: 0.9778 - val_loss: 0.2029 - val_accuracy: 0.9499
Epoch 25/50
276/276 [==============================] - 83s 300ms/step - loss: 0.0668 - accuracy: 0.9784 - val_loss: 0.2018 - val_accuracy: 0.9561
Epoch 26/50
276/276 [==============================] - 83s 300ms/step - loss: 0.0626 - accuracy: 0.9795 - val_loss: 0.2504 - val_accuracy: 0.9434
Epoch 27/50
276/276 [==============================] - 83s 299ms/step - loss: 0.0633 - accuracy: 0.9801 - val_loss: 0.1726 - val_accuracy: 0.9496
Epoch 28/50
276/276 [==============================] - 83s 300ms/step - loss: 0.0585 - accuracy: 0.9810 - val_loss: 0.1806 - val_accuracy: 0.9568
Epoch 29/50
276/276 [==============================] - 83s 300ms/step - loss: 0.0612 - accuracy: 0.9800 - val_loss: 0.2008 - val_accuracy: 0.9445
Epoch 30/50
276/276 [==============================] - 83s 300ms/step - loss: 0.0571 - accuracy: 0.9822 - val_loss: 0.2286 - val_accuracy: 0.9528
Epoch 31/50
276/276 [==============================] - 83s 301ms/step - loss: 0.0535 - accuracy: 0.9829 - val_loss: 0.2537 - val_accuracy: 0.9496
Epoch 32/50
276/276 [==============================] - 83s 302ms/step - loss: 0.0536 - accuracy: 0.9834 - val_loss: 0.1603 - val_accuracy: 0.9521
```

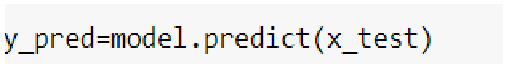


```
Epoch 33/50
276/276 [==============================] - 83s 301ms/step - loss: 0.0535 - accuracy: 0.9828 - val_loss: 0.2466 - val_accuracy: 0.9452
Epoch 34/50
276/276 [==============================] - 83s 301ms/step - loss: 0.0485 - accuracy: 0.9838 - val_loss: 0.1738 - val_accuracy: 0.9532
Epoch 35/50
276/276 [==============================] - 83s 302ms/step - loss: 0.0498 - accuracy: 0.9838 - val_loss: 0.3690 - val_accuracy: 0.9423
Epoch 36/50
276/276 [==============================] - 83s 300ms/step - loss: 0.0564 - accuracy: 0.9840 - val_loss: 0.2012 - val_accuracy: 0.9561
Epoch 37/50
276/276 [==============================] - 83s 299ms/step - loss: 0.0498 - accuracy: 0.9848 - val_loss: 0.2188 - val_accuracy: 0.9546
Epoch 38/50
276/276 [==============================] - 83s 300ms/step - loss: 0.0471 - accuracy: 0.9866 - val_loss: 0.3994 - val_accuracy: 0.9438
Epoch 39/50
276/276 [==============================] - 83s 301ms/step - loss: 0.0487 - accuracy: 0.9855 - val_loss: 0.1963 - val_accuracy: 0.9532
Epoch 40/50
276/276 [==============================] - 83s 300ms/step - loss: 0.0548 - accuracy: 0.9839 - val_loss: 0.1685 - val_accuracy: 0.9528
Epoch 41/50
276/276 [==============================] - 83s 299ms/step - loss: 0.0429 - accuracy: 0.9862 - val_loss: 0.1962 - val_accuracy: 0.9546
Epoch 42/50
276/276 [==============================] - ETA: 0s - loss: 0.0409 - accuracy: 0.9873
Reached 98.7% accuracy so cancelling training!
276/276 [==============================] - 83s 299ms/step - loss: 0.0409 - accuracy: 0.9873 - val_loss: 0.2599 - val_accuracy: 0.9474
```

Fig 7.3.3 Training the model

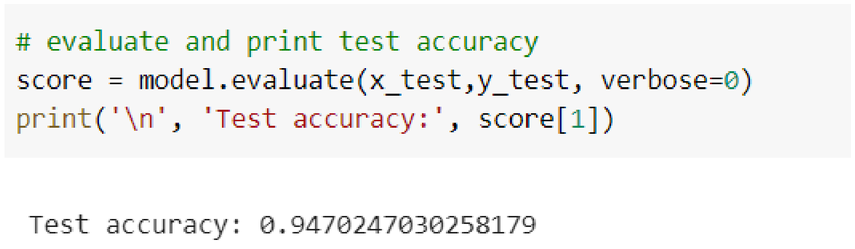
7.4.PREDICTING AND SUMMERIZING THE MODEL LOSS:

Predicting the data for the x_test data using predict function and evaluating the predicted data using evaluate function and finally summarizing the data model loss from train and test.



```
y_pred=model.predict(x_test)
```

Fig 7.4.1 Classifying the x_test data i.e., predicting



```
# evaluate and print test accuracy
score = model.evaluate(x_test,y_test, verbose=0)
print('\n', 'Test accuracy:', score[1])
```

```
 Test accuracy: 0.9470247030258179
```

Fig 7.4.2 Evaluating the model on test data

```
# summarize history for accuracy
plt.subplot(2,1,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')

 # summarize history for loss
plt.tight_layout(pad=3.0)

plt.subplot(2,1,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

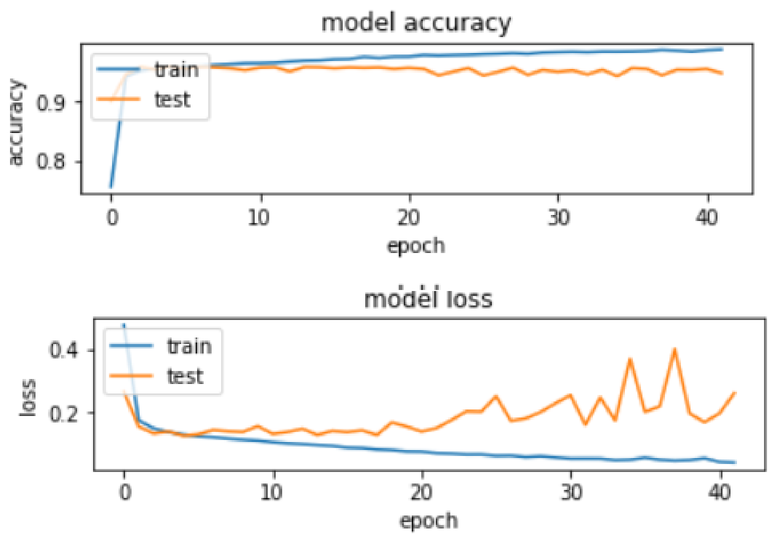
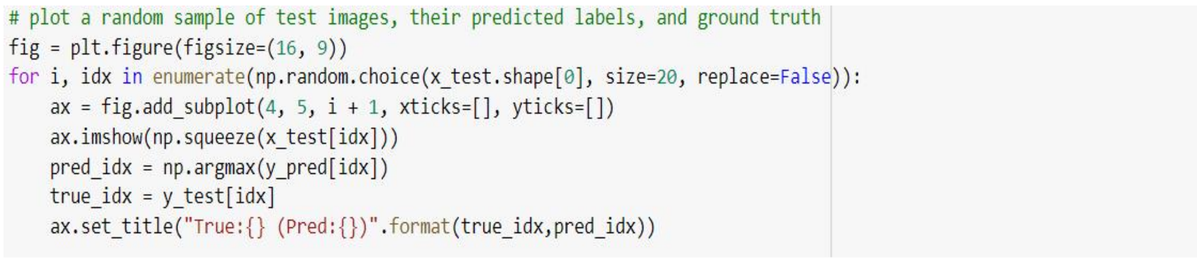


Fig 7.4.3 Summarizing the model loss for train and test data

Visualing the images with true values and predicted values



```
# plot a random sample of test images, their predicted labels, and ground truth
fig = plt.figure(figsize=(16, 9))
for i, idx in enumerate(np.random.choice(x_test.shape[0], size=20, replace=False)):
    ax = fig.add_subplot(4, 5, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_test[idx]))
    pred_idx = np.argmax(y_pred[idx])
    true_idx = y_test[idx]
    ax.set_title("True:{} (Pred:{})".format(true_idx,pred_idx))
```

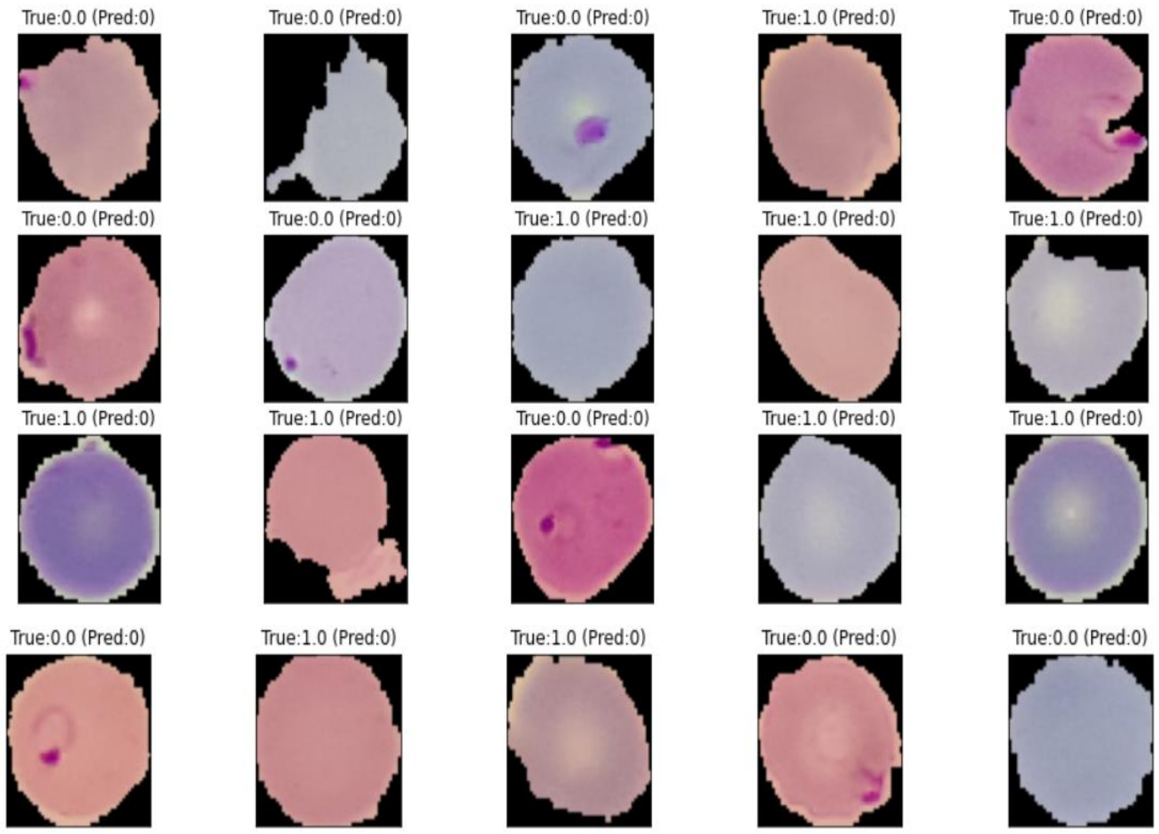


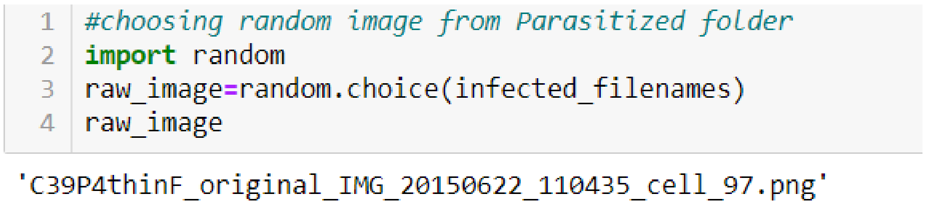
Fig 7.4.2 Visualizing the data with true and predicted values

7.5.PREDICTING THE CLASSES FOR SINGLE IMAGES:

Preprocessing

For the single images firstly we convert into array form that means reading RGB values in the array form and then resize into the (50,50,3) shape using resize function because of our model input and then rescaling the image by dividing the array with 255 and finally reshaping into [1,50,50,3] using reshape since our model will accept 4 dimensioned input. This all comes under the preprocessing of the image.

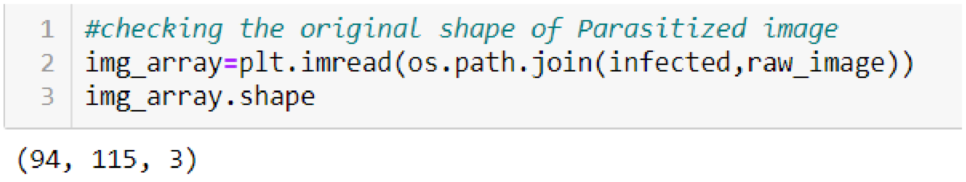
Parasitized image



```python
#choosing random image from Parasitized folder
import random
raw_image=random.choice(infected_filenames)
raw_image
```

```
'C39P4thinF_original_IMG_20150622_110435_cell_97.png'
```

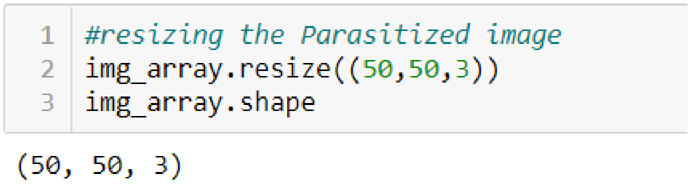
Fig 7.5.1 Choosing random image from the Parasitized folder



```python
#checking the original shape of Parasitized image
img_array=plt.imread(os.path.join(infected,raw_image))
img_array.shape
```

```
(94, 115, 3)
```

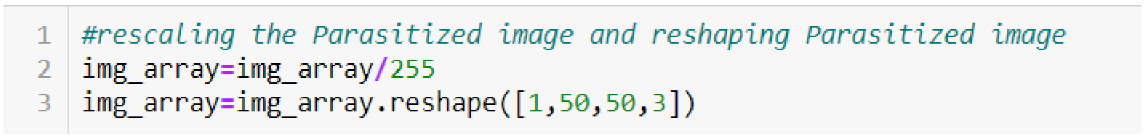
Fig 7.5.2 Reading the RGB in array format and checking the shape of the Parasitized image



```python
#resizing the Parasitized image
img_array.resize((50,50,3))
img_array.shape
```

```
(50, 50, 3)
```

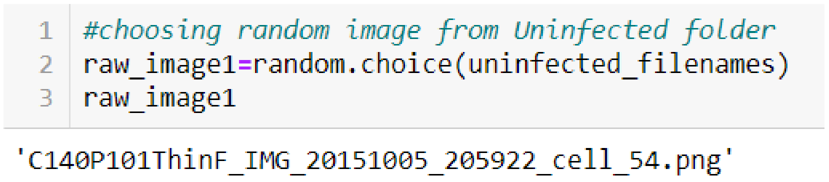
Fig 7.5.3 Resizing the Parasitized image and checking the shape of the Parasitized image



```python
#rescaling the Parasitized image and reshaping Parasitized image
img_array=img_array/255
img_array=img_array.reshape([1,50,50,3])
```

Fig 7.5.4 Rescaling and reshaping the resized Parasitized image

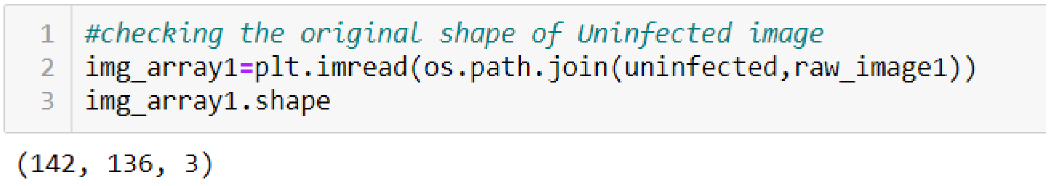
Uninfected image



```python
#choosing random image from Uninfected folder
raw_image1=random.choice(uninfected_filenames)
raw_image1
```

```
'C140P101ThinF_IMG_20151005_205922_cell_54.png'
```

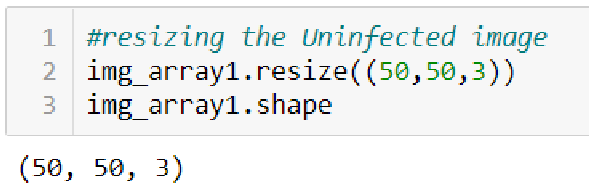
Fig 7.5.5 Choosing random image from the Uninfected folder



```python
#checking the original shape of Uninfected image
img_array1=plt.imread(os.path.join(uninfected,raw_image1))
img_array1.shape
```

```
(142, 136, 3)
```

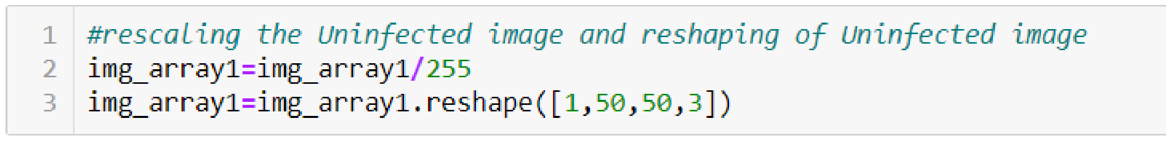
Fig 7.5.6 Reading the RGB in array format and checking the shape of the Uninfected image



```
#resizing the Uninfected image
img_array1.resize((50,50,3))
img_array1.shape
```

```
(50, 50, 3)
```

Fig 7.5.7 Resizing the Uninfected image and checking the shape of the Uninfected image



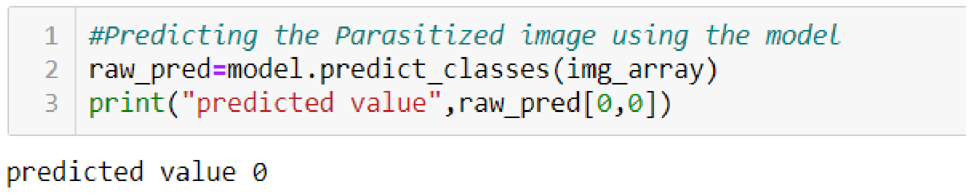
```
#rescaling the Uninfected image and reshaping of Uninfected image
img_array1=img_array1/255
img_array1=img_array1.reshape([1,50,50,3])
```

Fig 7.5.8 Rescaling and reshaping the resized Uninfected image

Predicting the image using model

We will predict the preprocessed image using the model with predict_classes and printing the predicted value

Parasitized image

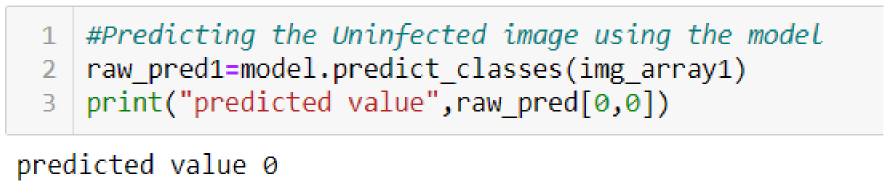


```
#Predicting the Parasitized image using the model
raw_pred=model.predict_classes(img_array)
print("predicted value",raw_pred[0,0])
```

```
predicted value 0
```

Fig 7.5.9 Predicting the class of the Parasitized image

Uninfected image



```
#Predicting the Uninfected image using the model
raw_pred1=model.predict_classes(img_array1)
print("predicted value",raw_pred[0,0])
```

```
predicted value 0
```

Fig 7.5.10 Predicting the class of the Uninfected image

8. CONCLUSION:

Malaria is a widespread disease that has claimed millions of lives all over the world. Automation of the diagnosis process will provide accurate diagnosis of the disease, which will benefit health-care to resources-scarce areas. We showed that the deep convolutional network based on conv2d was capable of achieving 98.7% on test data.

9. REFERENCE:

- https://en.wikipedia.org/wiki/Machine_learning
- https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148
- https://www.kaggle.com/iarunava/cell-images-for-detecting-malaria/kernels