### An Industry Oriented Mini Project

on

# “STOCK TRADING BOT”

*Submitted*

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

### Bachelor of Technology

in

### Computer Science and Engineering

by

**MR. G. JASHWANTH REDDY**

**(18261A0522)**

Under the guidance of

### Dr. V. SUBBA RAMAIAH

**(Assistant Professor)**



### DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING MAHATMA GANDHI INSTITUTE OF TECHNOLOGY

(Affiliated to Jawaharlal Nehru Technological University Hyderabad) Kokapet (V), Gandipet (M), Hyderabad.

Telangana -500 075. (India)

### 2020-2021

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY

(Affiliated to Jawaharlal Nehru Technological University Hyderabad) GANDIPET, HYDERABAD – 500 075. Telangana

**CERTIFICATE**



This is to certify that the Industrial Mini Project entitled “**Stock Trading Bot”** is being submitted by **G. Jashwanth Reddy** bearing **Roll No: 18261A0522** in partial fulfillment for the award of **B.Tech** in **Computer Science and Engineering** to **Jawaharlal Nehru Technological University Hyderabad** is a record of bonafide work carried out under the supervision of Dr. **V Subba Ramaiah, Assistant Professor, Department of CSE.**

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

Project Guide Head of the Department

### Dr. V. Subba Ramaiah Dr. C. R. K. Reddy

Asst. Professor, Dept. of CSE Professor, Dept. of CSE

### External Examiner

is

2

# DECLARATION

This is to certify that the work reported in this project titled “**STOCK TRADING BOT”** is a record of work done by me in the Department of Computer Science and Engineering, Mahatma Gandhi Institute of Technology, Hyderabad.

No part of the work is copied from books/journals/internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the work done entirely by me and not copied from any other source.

G. JASHWANTH REDDY

(18261A0522)

# ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the project.

I would like to express my sincere thanks to **Dr. K. Jaya Sankar, Principal MGIT**,

for providing the work facilities in the college.

I am also thankful to **Dr. C.R.K Reddy, Professor & Head of Department**, Dept. of Computer Science and Engineering for providing excellent infrastructure and a conducive atmosphere for completing this project successfully.

I am also extremely thankful to my Project Coordinators, **Dr. C.R.K Reddy, Professor & Head of Department, Dept. of CSE, Dr. B Prasanthi, Assistant Professor, Dept. of CSE and Ms. D Deepika, Assistant Professor, Dept. of CSE**, for their valuable suggestions and interest throughout the course of this project.

I would like to express my sincere gratitude and indebtedness to my Project Guide, **Dr. V Subba Ramaiah, Assistant Professor, Dept. of CSE** who has supported me throughout the project with patience and knowledge.

I convey my heartfelt thanks to the lab staff for allowing me to use the required equipment whenever needed.

Finally, I would like to take this opportunity to thank my family for their support throughout the work. I sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of this work.

**G. JASHWANTH REDDY (18261A0522)**

# TABLE OF CONTENTS

|  |  |
| --- | --- |
| Certificate | i |
| Declaration | ii |
| Acknowledgements | iii |
| List of Figures | vi |
| List of Tables | vii |
| Abstract | viii |
| **1. Introduction** | 1 |
| 1.1 Problem Definition | 2 |
| 1.2 Existing System | 2 |
| 1.3 Proposed System | 2 |
| 1.4 Requirements Specification | 3 |
| 1.4.1 Hardware Requirements | 3 |
| 1.4.2 Software Requirements | 3 |
| **2. Literature Survey** | 4 |
| **3. Methodology** | 9 |
| 3.1 Architecture | 9 |
| 3.1.1 VGG Classifier Pre-processing | 9 |
| 3.1.2 Dataset Description | 11 |
|  |  |
| 3.1.3 Neural Networks | 13 |
| 3.1.4 Convolutional Neural Network | 14 |
| 3.2 Modules | 15 |

|  |  |
| --- | --- |
| 3.3 Diagrammatic Representation | 16 |
| 3.3.1 Use Case Diagram | 17 |
|  |  |
| 3.3.2 Sequence Diagram | 18 |
| **4. Testing and Results** | 19 |
| 4.1 Evaluation | 19 |
| 4.2 Results | 20 |
| 4.2.1 jupyter notebook | 20 |
| 4.2.2 Output | 21 |
| **5. Conclusion and Future Scope** | 23 |
| 5.1 Conclusion | 23 |
| 5.2 Future Scope | 23 |
| **Bibliography** | 24 |
| **Appendix** | 26 |

v

6

# LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| Figure 3.1 | General Training Work-Flow | 9 |
| Figure 3.2 | Model Design | 11 |
| Figure 3.3 | Images in Rice leaf disease Dataset | 12 |
| Figure 3.4 | One Neuron in a Neural Network | 13 |
| Figure 3.6 | Convolutional Neural Network | 14 |
| Figure 3.7 | Use Case Diagram | 17 |
| Figure 3.8 | Sequence Diagram | 18 |
| Figure 4.1 | Juypter notebook environment | 20 |
| Figure 4.2 | Result-1 | 21 |
| Figure 4.3 | Result-2 | 21 |
| Figure 4.4 | Result-3 | 22 |
| Figure 4.5 | Result-4 | 22 |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| Table 2.1 | Literature Survey for Rice Leaf Detection using VGG Classifier. | 7 |

# ABSTRACT

Stock trading bot allows traders to establish specific rules for both trade entries and exits that, once programmed, can be automatically executed via a computer. Traders and investors can turn precise entry, exit, and money management rules into automated trading systems that allow computers to execute and monitor the trades. The trade entry and exit rules can be based on simple conditions such as a moving average crossover or they can be complicated strategies that require a comprehensive understanding of the programming language specific to the user's trading platform. This also helps to understand the accuracy of a trading algorithm. The bot can generate revenue at an inhuman and enhanced speed and frequency. The characterized sets of trading guidelines that are passed on to the bot are reliant upon timing, value, amount, or any mathematical model. Aside from profitable openings for the trader, algo-trading renders the market more liquid and trading more precise by precluding the effect of human feelings on trading.

This project uses a simple cross-over strategy which takes into account the closed average price of a stock over a number of days to decide whether to buy or sell a particular stock. It aims to further this revolution in the markets of tomorrow by providing an effective and efficient solution to overcome the drawbacks faced due to manual trading by building a Trading Bot which will automatically trade user strategies alongside its own algorithms for day-to-day trading based on different market conditions and user approach, and throughout the course of the day invest and trade with continuous modifications to ensure the best trade turnover for the day while reducing the transaction cost, hence enabling huge profits for concerned users be it Organizations or individuals.

viii

9

# INTRODUCTION

Stock trading using a bot is a technique for executing orders utilizing mechanized pre-modified trading guidelines representing factors like time, cost, and volume. This kind of trading endeavors to use the speed and computational assets of PCs comparative with human brokers. This kind trading improves these chances through better technique configuration, testing, and execution.

There is a long list of advantages to having a computer monitor the markets for trading opportunities and execute the trades, including:

**Minimizing Emotions:**

Automated trading systems minimize emotions throughout the trading process. By keeping emotions in check, traders typically have an easier time sticking to the plan. Since trade orders are executed automatically once the trade rules have been met, traders will not be able to hesitate or question the trade.

**Backtesting:**

Backtesting applies trading rules to historical market data to determine the viability of the idea. When designing a system for automated trading, all rules need to be absolute, with no room for interpretation. The computer cannot make guesses and it has to be told exactly what to do. Traders can take these precise sets of rules and test them on historical data before risking money in live trading. Careful backtesting allows traders to evaluate and fine-tune a trading idea, and to determine the system's expectancy – i.e., the average amount a trader can expect to win (or lose) per unit of risk.

**Preserving Discipline:**

Because trade rules are established and trade execution is performed automatically, discipline is preserved even in volatile markets. Discipline is often lost due to emotional factors such as fear of taking a loss, or the desire to eke out a little more profit from a trade. Automated trading helps ensure discipline is maintained because the trading plan will be followed exactly.

**Improving Order Entry Speed:**

Since computers respond immediately to changing market conditions, automated systems are able to generate orders as soon as trade criteria are met. Getting in or out of a trade a few seconds earlier can make a big difference in the trade's outcome. As soon as a position is entered, all other orders are automatically generated, including protective stop losses and profit targets.

**Diversifying Trading:**

Automated trading systems permit the user to trade multiple accounts or various strategies at one time. This has the potential to spread risk over various instruments while creating a hedge against losing positions.

## 1.1 Problem Definition

The problem is to develop a bot that can buy and sell stocks using a trading algorithm. We then have to interface our bot with a brokerage company that manages our transactions.

## 1.2 Existing System

In the existing model, machine learning models are used to predict the stock prices. The modelled bot can buy and sell stocks, but machine learning in stock trading is quite risky. There’s no chance to test the model in real time before investing.

The given model only predicts the changes in stocks, but it won’t interact with any brokerages to do the trading on behalf of humans. So, it can help in research although its not effective for trading.

## 1.3 Proposed System

In the proposed model, we can use any trading algorithm. We are interfacing our bot with a virtual stock trading brokerage app. This helps us to test various trading algorithms. Based on their performance, we can in fact deploy our bot into real time trading.

## 1.4 Requirements Specification

### 1.4.1 Hardware Requirements

* + - * Processor : Intel core (i5 or higher)
      * RAM : 4 GB
      * ROM : 8 GB

### 1.4.2 Software Requirements

* + - * Operating System : Windows/ MacOS
      * Programming Language : Python
      * Integrated Development Environment (IDE) : Juypter notebook
      * Subscription to a brokerage : Alpaca trading API

# LITERATURE SURVEY

In literature survey I have investigated various researches on this particular domain and some of them are as follows:

**1. Algorithmic Trading Bot**

Algorithmic trading uses algorithms that follow a trend and defined set of instructions to perform a trade. The trade can generate revenue at an inhuman and enhanced speed and frequency. The characterized sets of trading guidelines that are passed on to the program are reliant upon timing, value, amount, or any mathematical model. Aside from profitable openings for the trader, algo-trading renders the market more liquid and trading more precise by precluding the effect of human feelings on trading. Our project aims to further this revolution in the markets of tomorrow by providing an effective and efficient solution to overcome the drawbacks faced due to manual trading by building an Algorithmic Trading Bot which will automatically trade user strategies alongside its own algorithms for day-to-day trading based on different market conditions and user approach, and throughout the course of the day invest and trade with continuous modifications to ensure the best trade turnover for the day while reducing the transaction cost, hence enabling huge profits for concerned users be it Organizations or individuals.

**2. Stock Trading Bot Using Deep Reinforcement Learning**

This paper proposes automating swing trading using deep reinforcement learning. The deep deterministic policy gradient-based neural network model trains to choose an action to sell, buy, or hold the stocks to maximize the gain in asset value. The paper also acknowledges the need for a system that predicts the trend in stock value to work along with the reinforcement learning algorithm. We implement a sentiment analysis model using a recurrent convolutional neural network to predict the stock trend from the financial news. The objective of this paper is not to build a better trading bot, but to prove that reinforcement learning is capable of learning the tricks of stock trading.

**3. Algo-Trading Strategy for Intraweek Foreign Exchange Speculation Based on Random Forest and Probit Regression**

In the Forex market, the price of the currencies increases and decreases rapidly based on many economic and political factors such as commercial balance, the growth index, the inflation rate, and the employment indicators. Having a good strategy to buy and sell can make a profit from the above changes. A successful strategy in Forex should take into consideration the relation between benefits and risks. In this work, we propose an intraweek foreign exchange speculation strategy for currency markets based on a combination of technical indicators. This system has a two-level decision and is composed of the Probit regression model and rules discovery using Random Forest. There are two minimum requirements for a trading strategy: a rule to enter the market and a rule to exit it. Our proposed system, to enter the currency market, should validate two conditions. First, it should validate Random Forest access rules over the following week while in the second one the predicted value of the next day using Probit should be positive. To exit the currency market just one negative warning from Probit or Random Forest is enough. This system was used to develop dynamic portfolio trading systems. The profitability of the model was examined for USD/ (EUR, JYN, BRP) variation within the period from January 2014 to January 2016. The proposed system allows improving the prediction accuracy. This indicates a good prediction of the behaviour market and it helps to identify the good times to enter it or to leave it.

**4. Rules Based Policy for Stock Trading: A New Deep Reinforcement Learning Method**

Automated trading is fully represented as an online decision-making problem, where agents desire to sell it at a higher price to buy at a low one. In financial theory, financial markets trading produces a noisy and random behaviour involving highly imperfect information. Therefore, developing a profitable strategy is very complicated in dynamic and complex stock market environments. This paper introduces a new deep reinforcement learning (DRL) method based on the encouragement window policy for automatic stock trading. Motivated by the advantage function, the proposed approach trains a DRL agent to handle the trading environment's dynamicity and generate huge profits. On the one hand, the advantage function tries to estimate the relative value of the current state's selected actions. It consists of the discounted sum of rewards and the baseline estimate. On the other hand, the encouragement window is based only on the last rewards, providing a dense synthesized experience instead of a noisy signal. This process has progressively improved actions' quality by balancing the action selection versus states' uncertainty. The self-learned rules drive the agent's policy to choose productive actions that produce a high achievement across the environment. Experimental results on four real-world stocks have proven the proposed system's efficiency. Precisely, it has produced outstanding performances, executed more creative trades by a small number of transactions, and outperformed different baselines.

**5. Predicting the direction of stock market prices using tree-based classifiers**

Predicting returns in the stock market is usually posed as a forecasting problem where prices are predicted. Intrinsic volatility in the stock market across the globe makes the task of prediction challenging. Consequently, forecasting and diffusion modeling undermines a diverse range of problems encountered in predicting trends in the stock market. Minimizing forecasting error would minimize investment risk. In the current work, we pose the problem as a direction-predicting exercise signifying gains and losses. We develop an experimental framework for the classification problem which predicts whether stock prices will increase or decrease with respect to the price prevailing n days earlier. Two algorithms, random forests, and gradient boosted decision trees (using XGBoost) facilitate this connection by using ensembles of decision trees. We test our approach and report the accuracies for a variety of companies as improvement over existing predictions. A novelty of the current work is about the selection of technical indicators and their use as features, with high accuracy for medium to long-run prediction of stock price direction.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S.no** | **Author** | **Title** | **Year** | **Techniques** | **Advantages** | **Disadvantages** |
| **1.** | Medha Mathur, Satyam Mhadalekar, Sahil Mhatre, Vanita Mane | Algorithmic Trading Bot | 2021 | In this model, machine learning models are used to predict the stock prices. The modelled bot can buy and sell stocks | This model is useful for different markets as it is flexible. | Machine learning in stock trading is quite risky. There’s no chance to test the model in real time before investing. |
| **2.** | Akhil Raj Azhikodan, Anvitha G. K. Bhat and Mamatha V. Jadhav | Stock Trading Bot Using Deep Reinforcement Learning | 2019 | The deep deterministic policy gradient-based neural network model trains to choose an action to sell, buy, or hold the stocks to maximize the gain in asset value. | The paper acknowledges the need for a system that predicts the trend in stock value to work along with the reinforcement learning algorithm. | Although this project shows that reinforcement learning is useful in trading, it provides less accuracy. |
| **3.** | Zineb Bousbaa, Marouane Chihab, Omar Bencharef, and Soumia Ziti | Algo-Trading Strategy for Intraweek Foreign Exchange Speculation Based on Random Forest and Probit Regression | 2019 | This system has a two-level decision and is composed of the Probit regression model and rules discovery using Random Forest. | This indicates a good prediction of the behavior market and it helps to identify the good times to enter it or to leave it. | It only provides prediction on when to enter and when to exit. It doesn’t provide information for the actions in between. |
| **4.** | Hirchoua Badr, Brahim Ouhbi, Bouchra Frikh | Rules Based Policy for Stock Trading: A New Deep Reinforcement Learning Method | 2019 | Deep reinforcement learning method. | The self-learned rules drive the agent's policy to choose productive actions that produce a high achievement across the environment. | This can’t trade on its own. |
| **5.** | Suryoday Basaka, Saibal Karbc, Snehanshu Sahaa, Luckyson Khaidema, Sudeepa RoyDeya | Predicting the direction of stock market prices using tree-based classifiers | 2019 | Random forests, and gradient boosted decision trees (using XGBoost) | A novelty of this work is about the selection of technical indicators and their use as features, with high accuracy for medium to long-run prediction of stock price direction. | Machine learning in stock trading is quite risky. There’s no chance to test the model in real time before investing. |

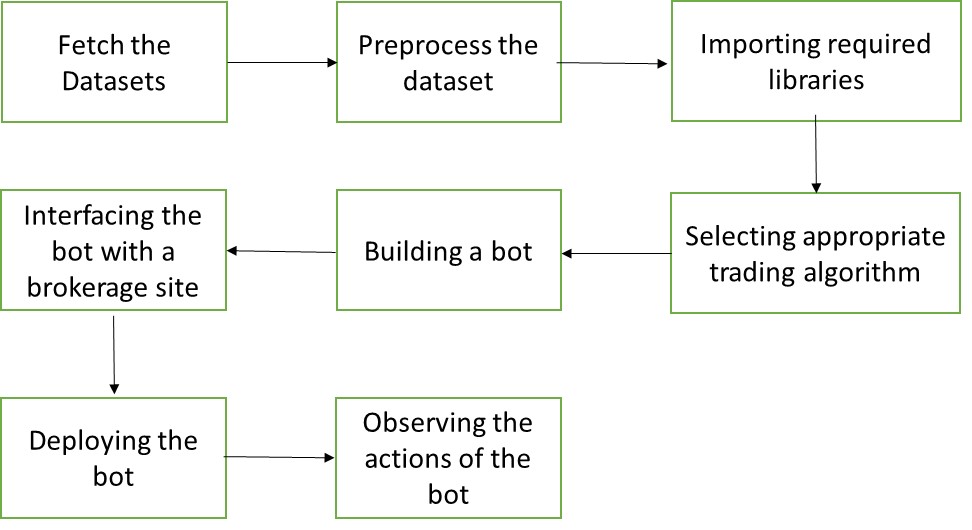
**Table 2.1**: Literature survey for Stock Trading Bot

# METHODOLOGY

## 3.1 ARCHITECTURE

The objective is to build a fully functioning bot that can buy and sell stocks by interacting with a brokerage API. The decisions of the bot are based on the cross-over trading algorithm. This algorithm calculates whether a stock is profitable or not based on its price from the past few days. The basic flow of operation can be observed in Figure 3.1.

**SYSTEM ARCHITECTURE**



**Fig. 3.1**. General work-flow

## 3.1.1 Cross over trading strategy:

The crossover is a point on the trading chart in which a security's price and a technical indicator line intersect, or when two indicators themselves cross. Crossovers are used to estimate the performance of a financial instrument and to predict coming changes in trend, such as reversals or breakouts.

A crossover is used by a technical analyst to forecast how a stock will perform in the near future. For most models, the crossover signals that it’s time to either buy or sell the underlying asset. Investors use crossovers along with other indicators to track things like turning points, price trends and money flow.

Crossovers indicating a moving average are generally the cause of breakouts and breakdowns. Moving averages can determine a change in the price trend based on the crossover. For example, a technique for trend reversal is using a five-period simple moving average along with a 15-period simple moving average (SMA). A crossover between the two will signal a reversal in trend, or a breakout or breakdown.

A breakout would be indicated by the five-period moving average crossing up through the 15-period. This is also indicative of an uptrend, which is made of higher highs and lows. A breakdown would be indicated by the five-period moving average crossing down through the 15-period. This is also indicative of a downtrend, composed of lower highs and lows.

Longer time frames result in stronger signals. For example, a daily chart carries more weight than a one-minute chart. Conversely, the shorter time frames give earlier indicators, but they are also susceptible to false signals as well.

When the faster moving average goes above the slower moving average this is an indication that the price will go up and is an optimal time to buy the stock, this type of indicator is known as a “Golden Cross”. When the faster moving average goes below the slower moving average this is a great indication of when the price will fall and is an optimal time to sell, this is known as a “Death Cross”.

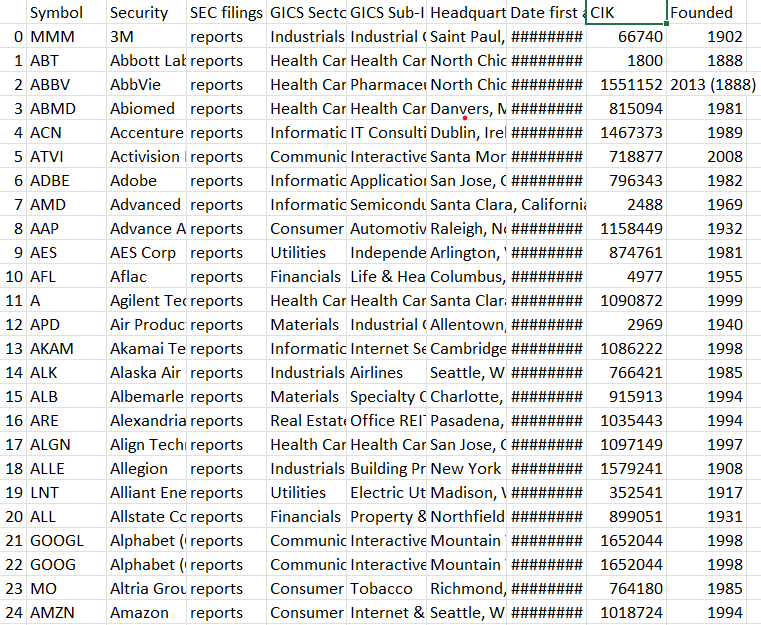
A moving average simplifies price data by smoothing it out and creating one flowing line. This makes seeing the trend easier. Exponential moving averages react quicker to price changes than simple moving averages. As we can see from the figure 3.2, this is how cross over strategy works. Once, the blue-line overlaps the yellow line and moves above, the bot sells the stocks, anticipating a loss.



**Fig. 3.2.**  A demonstration of cross-over trading strategy

### 3.1.2 Dataset description

The data that we will be getting will be from the S&P 500 index. The S&P 500 index, or Standard & Poor’s 500, is a very important index that tracks the performance of the stocks of 500 large-cap companies in the U.S. The ticker symbol for the S&P 500 index is ^GSPC. The series of letters represents the performance of the 500 stocks listed on the S&P. However, it must be remembered that ^GSPC is a price index and is not tradeable. It only shows the movement of stock prices in the S&P 500 index. The code uses pandas to scrape the ticker labels of the S&P 500 index off of Wikipedia and formats the tickers into a csv file.

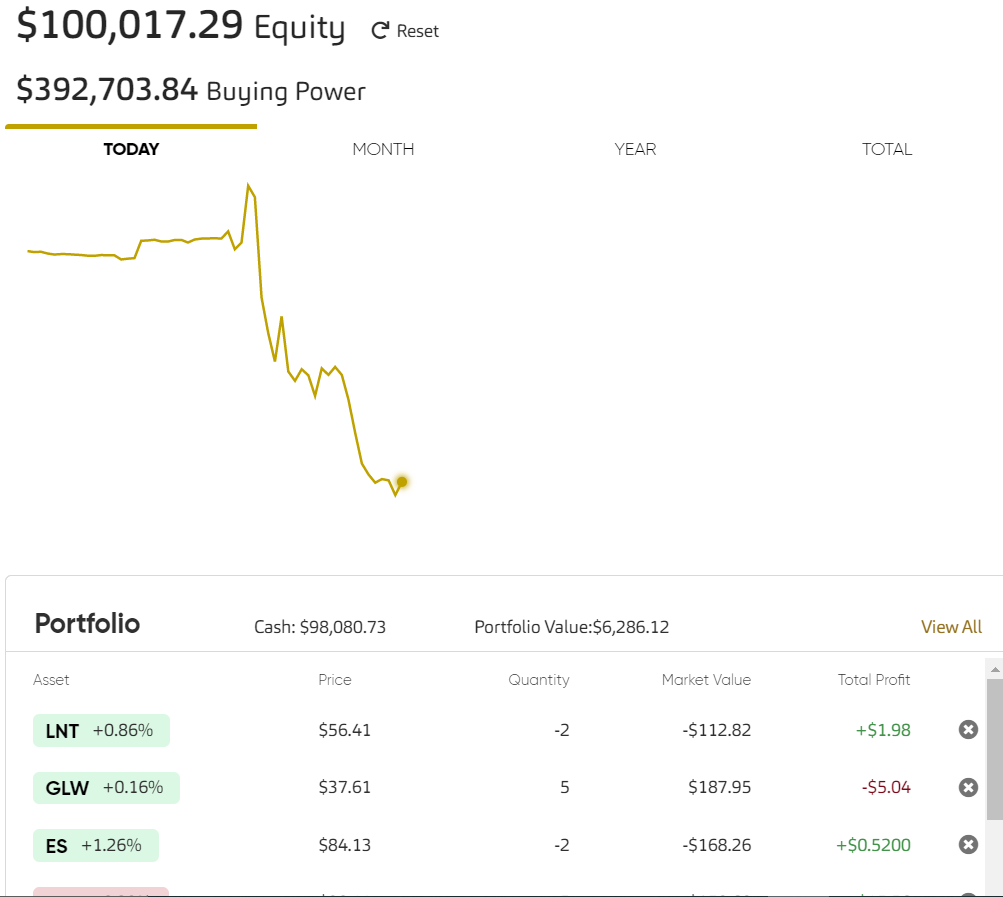


**Fig. 3.3**. A glimpse of the S&P 500 index dataset

### 3.1.3 Selecting and Interacting with a Brokerage

For this project we will be using Alpaca API to place and execute trades. There are many different brokerages to choose from when writing your algorithm/bot, but the advantages that are included in this brokerage are the commission free trades (being able to trade without a fee), the ability to simulate trades without using actual money (paper trading), and most importantly it is designed to developers and their trading bots. There are ways to make bots for Robinhood, Webull, or other brokers using an unofficial API, but it is unstable and there is no guarantee that this will work in the near future.

The actions of the bot will be reflected in the brokerage as observed in the figure 3.4.



**Fig. 3.4** A snapshot of the brokerage API

### 3.1.4 Coding the bot

The bot is coded in python language. Every hour from when the stock market opens to when the market closes this code will check to see if the moving averages cross over each other for each of the tickers we scrapped. The variable “result” is a Boolean value that will determine what type of trade we will executed. When the variable “position” is true we execute a buy order and when the variable is false, we execute a sell order. In the case where it is neither of this value, we will have a None variable and the stock will be skipped.

### 3.2 Modules

* **NumPy:**

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. NumPy is open- source software and has many contributors.

### Pillow:

PIL is the Python Imaging Library by Fredrik and Contributors. The Python Imaging Library adds image processing capabilities to your Python interpreter. This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.

### Matplotlib:

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is a plotting library for the Python programming language and its numerical mathematics extension NumPy. Matplotlib can be used in Python scripts, the Python and IPython shell, web application servers, and various graphical user interface toolkits.

### Torch:

Torch is an open-source machine learning library, a scientific computing framework, and a script language based on the Lua programming language. It provides a wide range of algorithms for deep learning, and uses the scripting language LuaJIT, and an underlying C implementation. The Tensor also supports mathematical operations like max, min, sum, statistical distributions like uniform, normal and multinomial, and BLAS operations like dot product, matrix-vector multiplication etc.

### TorchVision:

TorchVision is PyTorch’s own computer vision library which contains many important and useful datasets as well as models and transformation operations that are often used in the area of computer vision. The torchvision package consists of popular datasets, model architectures, and common image transformations for computer vision.

### 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.

## 3.3 Diagrammatic Representation

### UML Diagrams:

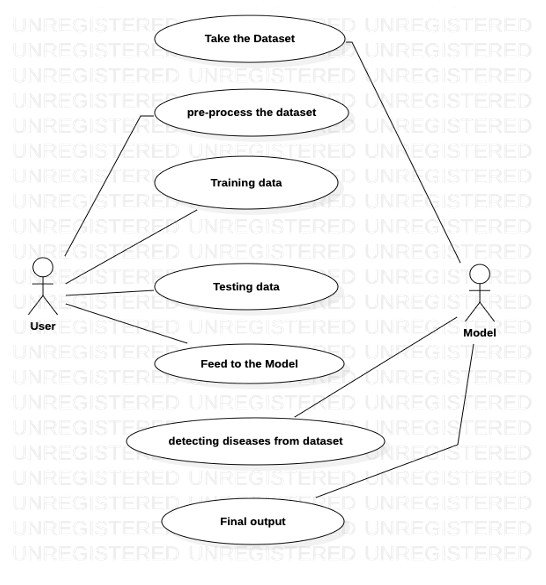
The Unified Modelling Language (UML) is a standard language for writing software blueprints. The UML is a language for:

* **Specifying**: It is just like a blueprint created by an architect prior to the construction.
* **Visualizing**: Visualizing is concerned with deep analysis of system to be constructed.
* **Constructing**: Modelling also provide us mechanism which are essential while constructing a system.
* **Documenting**: Finally, modelling justifies its importance by applying all its credentials to be bounded in a piece of paper referred as document.

The UML is a language which provides vocabulary and the rules for combining words in that vocabulary for the purpose of communication. A modelling language is a language whose vocabulary and the rules focus on the conceptual and physical representation of a system. Modelling yields an understanding of a system

### 3.3.1 Use Case Diagram

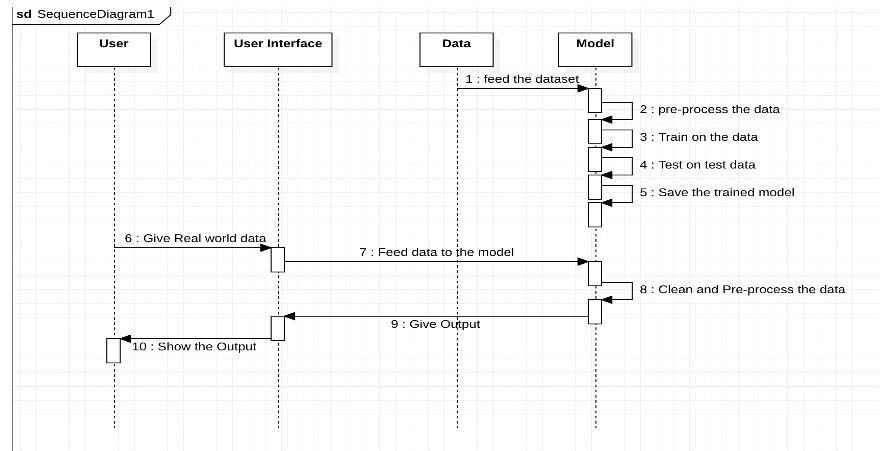
Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So, when a system is analyzed together its functionalities use cases are prepared and actors are identified.



**Fig. 3.6.** Use Case Diagram of Rice Leaf Disease Detection using VGG Classifier

In Figure 3.6, the user loads the dataset and preprocesses the image data. While, the system applies transfer learning, builds a VGG model, trains and tests the model to generate results.

### 3.3.3 Sequence Diagram

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

**Fig. 3.9.** Sequence Diagram of Rice Leaf Disease Detection using VGG Classifier

The Figure 3.9 shows the sequence followed in the project, where the user loads the dataset and preprocesses it. The system applies transfer learning, builds a VGG model, applies an activation function, Trains and tests the model. The results are generated back to the user interface.

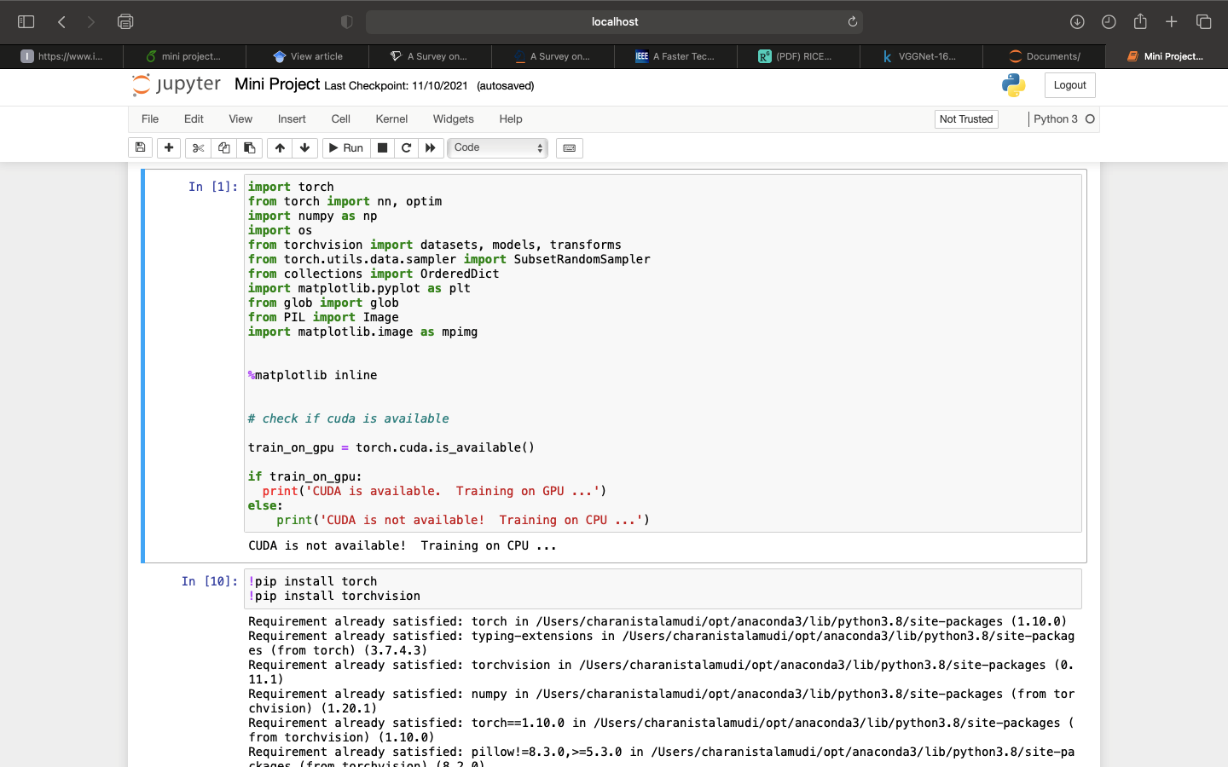
# TESTING AND RESULTS

## 4.1 Evaluation

* Data-preprocessing: The images in the data set are resized to 224 X 224.
* Data set is divided into 80 % for training and remaining 20% for testing purposes.
* Now the Data Augmentation process is applied to avoid overfitting.
* The convolution layers of the model extracts the features in order to classify the images.
* Now with those training set features the model is trained.
* The trained model is validated/tested using the testing data set.
* If tuning is necessary then we can perform tuning by changing the number of epochs or by changing activations functions or by changing batch size accordingly.
* This model is saved so that it can be used for new data ie. For uploaded image.
* The server contains the trained model and by using this model it predicts whether it is healthy or not. If it is not healthy then it gives the disease name as output.
* This result is sent as a response back to the user.

## 4.2 Results

### 4.2.1Jupyter notebook



**Fig.4.1**. jupyter notebook Environment

The code for the “Rice leaf disease detection using VGG Classifier” is written here. The code is written in Python language.

As shown in the above figure 4.1, it shows the environment for the implementation of the working code.

### 4.2.2 Output

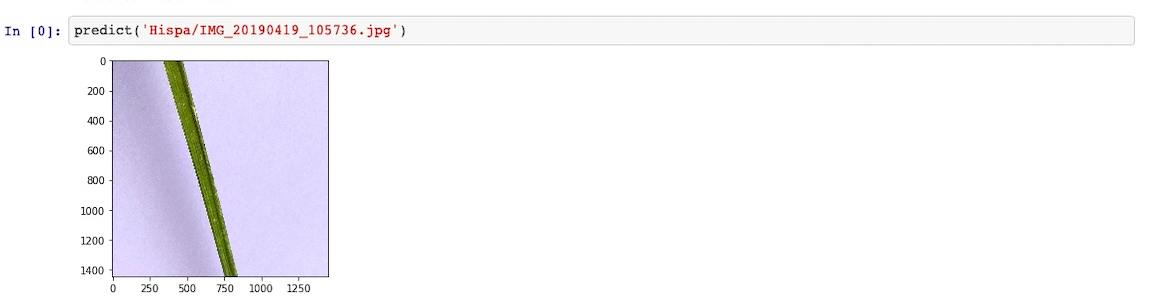
After training the model on 7 epochs, the test images are used to produce results where wthen the user the gives the image pathname to the model, it shows the output as image of that pathname along with it’s disease.



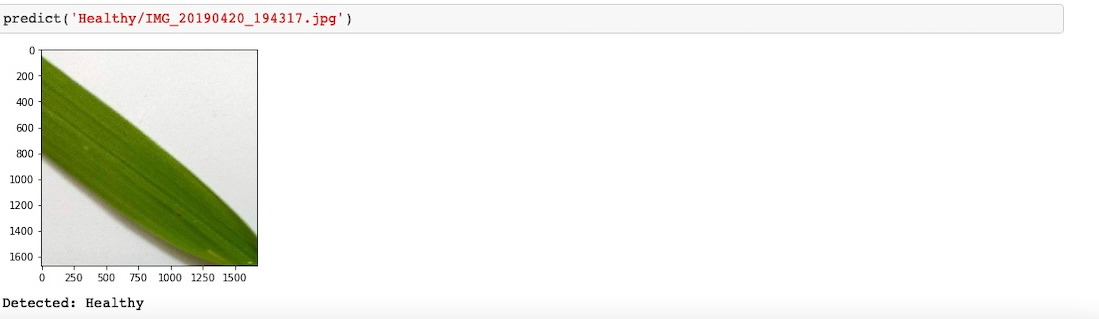
**Fig.4.2.** Result-1



**Fig.4.3.** Result-2



**Fig.4.4**. Result-3



**Fig. 4.5** Result-4

Figures 4.2, 4.3, 4.4, 4.5 display the final output on the range of different type of diseases that a rice leaf affected with. So, as by concluding the outputs we can say that model is working more accurately compared to other classifiers with VGG Classifier.

# CONCLUSION AND FUTURE SCOPE

## 5.1 CONCLUSION:

There are many methods in automated or computer vision plant disease detection and classification processes, but still, this research field is lacking. In addition, there are still no commercial solutions on the market, except those dealing with plant species recognition based on the leaves images.

Here a new approach of using deep learning methods was explored in order to automatically classify and detect plant diseases from leaf images with an accuracy of 96% training and 94% validation.

## 5.2 FUTURE SCOPE:

The developed model was able to detect leaves between healthy leaves and different diseases, which can be visually diagnosed. The model is able to detect only a few diseases. In future we will extend to this model to detect more number of diseases. As this is a small a website containing a server with trained model and frontend for the users. In future we will extend this with APIs and in further we can use drones which will captures the leaves and send those images to the server where it predicts where they are healthy or not and will maintain a record of statistical data (percentage of healthy leaves and defected leaves at a particular location).

# BIBLIOGRAPHY

1. Pothen, M. E., & Pai, M. L. (2020, March). Detection of Rice Leaf Diseases Using Image Processing.

In 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC) (pp. 424-430). IEEE..

[2] Rani, B., Ojha, S., Kaushal, H., Kumar, S., Roy Chowdhury, T., Karmakar, S., & Jana, B. (2020). A Study on Detection of Brown Spot Disease in Rice Plant Using Machine Learning Technique: International Conference on Recent Trends in

Artificial Intelligence, IoT, Smart Cities & Application (ICAISC 2020). IoT, Smart Cities & Application (ICAISC 2020)(July 9, 2020).

[3] Bera, T., Das, A., Sil, J., & Das, A. K. (2019). A survey on rice plant disease identification using image processing and data mining techniques. In Emerging Technologies in Data Mining and Information Security (pp. 365-376). Springer, Singapore.

[4] Islam, T., Sah, M., Baral, S., & Choudhury, R. R. (2018, April). A faster technique on rice disease detectionusing image processing of affected area in agro-field. In 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT) (pp. 62-66). IEEE.

[5] Suresha, M., Shreekanth, K. N., & Thirumalesh, B. V. (2017, April). Recognition of diseases in paddy leaves using knn classifier. In 2017 2nd International Conference for Convergence in Technology (I2CT) (pp. 663-666). IEEE.

[6] Fakhri, A., Nasir, A., Nordin, M., Rahman, A., Mamat, A.R.: A study of image processing in agriculture application under high performance computing environment. Int. J. Comput. Sci. Telecommun. **3**, 16–24 (2012)

[7] Phadikar, S., Sil, J., Das, A.K.: Classification of rice leaf diseases based on morphological changes. Int. J. Inf. Electron. Eng. **2**(3) (2012)

[8] Anitha, A.: A predictive modeling approach for improving paddy crop productivity using data mining techniques. Turk. J. Electr. Eng. Comput. Sci. **25**, 4777–4787 (2017)

# APPENDIX

#In[1]

# Download and unzip (2.2GB)

# !pip install -q kaggle

# !mkdir -p ~/.kaggle

# !cp kaggle.json ~/.kaggle/

# !chmod 600 /root/.kaggle/kaggle.json

# !kaggle datasets download -d minhhuy2810/rice-diseases-image-dataset

#In[2]

# Import Libraries

import torch

from torch import nn, optim

import numpy as np

import os

from torchvision import datasets, models, transforms

from torch.utils.data.sampler import SubsetRandomSampler

from collections import OrderedDict

import matplotlib.pyplot as plt

from glob import glob

from PIL import Image

import matplotlib.image as mpimg

%matplotlib inline

#In[3]

# Check if CUDA is available

train\_on\_gpu = torch.cuda.is\_available()

if train\_on\_gpu:

print('CUDA is available. Training on GPU ...')

else:

print('CUDA is not available! Training on CPU ...')

#In[4]

# Install Libraries

!pip install torch

!pip install torchvision

#In[5]

#Dataset

data\_dir = '/Users/charanistalamudi/Downloads/archive/LabelledRice/Labelled'

images = glob(os.path.join(data\_dir, '\*/\*.jpg'))

total\_images = len(images)

print('Total images:', total\_images)

#In[6]

#number of images per class

image\_count = []

class\_names = []

for folder in os.listdir(os.path.join(data\_dir)):

folder\_num = len(os.listdir(os.path.join(data\_dir, folder)))

image\_count.append(folder\_num)

class\_names.append(folder)

print('{:20s}'.format(folder), end=' ')

print(folder\_num)

data\_transforms = {

'train': transforms.Compose([

transforms.RandomRotation(45),

transforms.RandomResizedCrop(224),

transforms.RandomHorizontalFlip(),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406],

[0.229, 0.224, 0.225])

]),

'valid': transforms.Compose([

transforms.Resize(224 + 32),

transforms.CenterCrop(224),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406],

[0.229, 0.224, 0.225])

])

}

image\_datasets=dict()

image\_datasets['train']=datasets.ImageFolder(data\_dir,transform=data\_transforms['train'])

image\_datasets['valid']=datasets.ImageFolder(data\_dir,transform=data\_transforms['valid'])

# number of subprocesses to use for data loading

num\_workers = 0

# how many samples per batch to load

batch\_size = 20

# percentage of training set to use as validation

valid\_size = 0.2

#In[7]

# obtain training indices to use for validation

num\_train = len(image\_datasets['train'])

indices = list(range(num\_train))

np.random.shuffle(indices)

split = int(np.floor(valid\_size \* num\_train))

train\_idx, valid\_idx = indices[split:], indices[:split]

# define samplers for obtaining training and validation batches

train\_sampler = SubsetRandomSampler(train\_idx)

valid\_sampler = SubsetRandomSampler(valid\_idx)

# create dataloaders

train\_loader = torch.utils.data.DataLoader(image\_datasets['train'], batch\_size=batch\_size, sampler=train\_sampler, num\_workers=num\_workers)

valid\_loader = torch.utils.data.DataLoader(image\_datasets['valid'], batch\_size=batch\_size, sampler=valid\_sampler, num\_workers=num\_workers)

dataloaders = {}

dataloaders['train'] = train\_loader

dataloaders['valid'] = valid\_loader

dataset\_sizes = {x: len(image\_datasets[x]) for x in ['train', 'valid']}

# specify class names

class\_names = image\_datasets['train'].classes

# helper function to un-normalize and display image

def imshow(img):

img = img / 2 + 0.5

plt.imshow(np.transpose(img, (1, 2, 0))) #convert from tensor image

# obtain one batch of training images

dataiter = iter(train\_loader)

images, labels = dataiter.next()

images = images.numpy()

# plot the images in batch with their labels

fig = plt.figure(figsize=(20, 3))

# display 20 images

for idx in np.arange(20):

ax = fig.add\_subplot(2, 20/2, idx+1, xticks=[], yticks=[])

imshow(images[idx])

ax.set\_title(class\_names[labels[idx]])

#In[8]

# download the pretrained model

model\_vgg = models.vgg16(pretrained=True)

for param in model\_vgg.features.parameters():

param.requires\_grad = False

n\_inputs = model\_vgg.classifier[6].in\_features

last\_layer = nn.Sequential(nn.Linear(n\_inputs, 512),

nn.ReLU(True),

nn.BatchNorm1d(512),

nn.Dropout(0.5),

nn.Linear(512, 4))

model\_vgg.classifier[6] = last\_layer

if train\_on\_gpu:

model\_vgg = model\_vgg.cuda()

model\_vgg

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model\_vgg.classifier.parameters(), lr=0.001)

device = "cuda" if torch.cuda.is\_available() else "cpu"

model\_vgg.to(device);

#In[9]

def train(n\_epochs, loaders, model, optimizer, criterion, use\_cuda, save\_path):

"""returns trained model"""

# initialize tracker for minimum validation loss

valid\_loss\_min = np.Inf

for epoch in range(1, n\_epochs+1):

# initialize variables to monitor training and validation loss

train\_loss = 0.0

valid\_loss = 0.0

# train the model

model.train()

for batch\_idx, (data, target) in enumerate(loaders['train']):

# move to GPU

if use\_cuda:

data, target = data.cuda(), target.cuda()

## find the loss and update the model parameters accordingly

optimizer.zero\_grad()

output = model(data)

loss = criterion(output, target)

loss.backward()

optimizer.step()

## record the average training loss

train\_loss = train\_loss + ((1 / (batch\_idx + 1)) \* (loss.data - train\_loss))

# print training/validation statistics

print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(

epoch,

train\_loss,

valid\_loss

))

## save the model if validation loss has decreased

if valid\_loss <= valid\_loss\_min:

print('Validation loss decreased ({:.6f} --> {:.6f}). Saving model..'.format(valid\_loss\_min, valid\_loss))

torch.save(model.state\_dict(), save\_path)

valid\_loss\_min = valid\_loss

return model

model\_vgg = train(10, dataloaders, model\_vgg, optimizer,

criterion, train\_on\_gpu, 'model.pt')

#In[10]

# load the model that got the best validation accuracy ny7

model\_vgg.load\_state\_dict(torch.load('model.pt'))

def predict(img\_path):

transform = transforms.Compose([transforms.RandomResizedCrop(224),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406],

[0.229, 0.224, 0.225])])

img = Image.open(img\_path)

img = transform(img)

img = img.unsqueeze(0)

prediction = model\_vgg(img)

prediction = prediction.cpu().data.numpy().argmax()

img=mpimg.imread(img\_path)

imgplot = plt.imshow(img)

plt.show()

print('Detected: {}'.format(class\_names[prediction]))

os.chdir("/Users/charanistalamudi/Downloads/archive/LabelledRice/Labelled")

predict('Healthy/IMG\_20190419\_135724.jpg')

predict('BrownSpot/IMG\_20190420\_195201.jpg')

# Extract midlevel features from ResNet-gray self.midlevel\_resnet = nn.Sequential(\*list(resnet.children())[0:6]) ## Second half: Upsampling

self.upsample = nn.Sequential(

nn.Conv2d(MIDLEVEL\_FEATURE\_SIZE, 128, kernel\_size=3, stride=1, padding=1), nn.BatchNorm2d(128),

nn.ReLU(), nn.Upsample(scale\_factor=2),

nn.Conv2d(128, 64, kernel\_size=3, stride=1, padding=1), nn.BatchNorm2d(64),

nn.ReLU(),

nn.Conv2d(64, 64, kernel\_size=3, stride=1, padding=1), nn.BatchNorm2d(64),

nn.ReLU(), nn.Upsample(scale\_factor=2),

nn.Conv2d(64, 32, kernel\_size=3, stride=1, padding=1), nn.BatchNorm2d(32),

nn.ReLU(),

nn.Conv2d(32, 2, kernel\_size=3, stride=1, padding=1), nn.Upsample(scale\_factor=2) )

def forward(self, input):

# Pass input through ResNet-gray to extract features midlevel\_features = self.midlevel\_resnet(input)

# Upsample to get colors

output = self.upsample(midlevel\_features) return output

#In[8]

model = ColorizationNet()

#In[9]

criterion = nn.MSELoss()

#In[9]

optimizer = torch.optim.Adam(model.parameters(), lr=1e-2, weight\_decay=0.0)

#In[10]

#Loading the data

class GrayscaleImageFolder(datasets.ImageFolder):

'''Custom images folder, which converts images to grayscale before loading''' def getitem (self, index):

path, target = self.imgs[index] img = self.loader(path)

if self.transform is not None:

img\_original = self.transform(img) img\_original = np.asarray(img\_original) img\_lab = rgb2lab(img\_original) img\_lab = (img\_lab + 128) / 255 img\_ab = img\_lab[:, :, 1:3]

img\_ab = torch.from\_numpy(img\_ab.transpose((2, 0, 1))).float() img\_original = rgb2gray(img\_original)

img\_original = torch.from\_numpy(img\_original).unsqueeze(0).float() if self.target\_transform is not None:

target = self.target\_transform(target) return img\_original, img\_ab, target

#In[11]

#Transforms for training and validation data # Training

train\_transforms = transforms.Compose([transforms.RandomResizedCrop(224), transforms.RandomHorizontalFlip()])

train\_imagefolder = GrayscaleImageFolder('images/train', train\_transforms)

train\_loader = torch.utils.data.DataLoader(train\_imagefolder, batch\_size=64, shuffle=True) # Validation

val\_transforms = transforms.Compose([transforms.Resize(256), transforms.CenterCrop(224)])

val\_imagefolder = GrayscaleImageFolder('images/val' , val\_transforms)

val\_loader = torch.utils.data.DataLoader(val\_imagefolder, batch\_size=64, shuffle=False)

#In[12]

# Before we train, we define helper functions for tracking the training loss and converting images back to RGB.

class AverageMeter(object):

'''A handy class from the PyTorch ImageNet tutorial''' def init (self):

self.reset() def reset(self):

self.val, self.avg, self.sum, self.count = 0, 0, 0, 0 def update(self, val, n=1):

self.val = val

self.sum += val \* n self.count += n

self.avg = self.sum / self.count

def to\_rgb(grayscale\_input, ab\_input, save\_path=None, save\_name=None): '''Show/save rgb image from grayscale and ab channels

Input save\_path in the form {'grayscale': '/path/', 'colorized': '/path/'}''' plt.clf() # clear matplotlib

color\_image = torch.cat((grayscale\_input, ab\_input), 0).numpy() # combine channels color\_image = color\_image.transpose((1, 2, 0)) # rescale for matplotlib color\_image[:, :, 0:1] = color\_image[:, :, 0:1] \* 100

color\_image[:, :, 1:3] = color\_image[:, :, 1:3] \* 255 - 128 color\_image = lab2rgb(color\_image.astype(np.float64)) grayscale\_input = grayscale\_input.squeeze().numpy()

if save\_path is not None and save\_name is not None:

plt.imsave(arr=grayscale\_input, fname='{}{}'.format(save\_path['grayscale'], save\_name), cmap='gray')

plt.imsave(arr=color\_image, fname='{}{}'.format(save\_path['colorized'], save\_name))

#In[13]

#Validation

def validate(val\_loader, model, criterion, save\_images, epoch): model.eval()

# Prepare value counters and timers

batch\_time, data\_time, losses = AverageMeter(), AverageMeter(), AverageMeter() end = time.time()

already\_saved\_images = False

for i, (input\_gray, input\_ab, target) in enumerate(val\_loader): data\_time.update(time.time() - end)

# Use GPU

if use\_gpu: input\_gray, input\_ab, target = input\_gray.cuda(), input\_ab.cuda(), target.cuda() # Run model and record loss

output\_ab = model(input\_gray) # throw away class predictions loss = criterion(output\_ab, input\_ab)

losses.update(loss.item(), input\_gray.size(0)) # Save images to file

if save\_images and not already\_saved\_images: already\_saved\_images = True

for j in range(min(len(output\_ab), 10)): # save at most 5 images save\_path = {'grayscale': 'outputs/gray/', 'colorized': 'outputs/color/'}

save\_name = 'img-{}-epoch-{}.jpg'.format(i \* val\_loader.batch\_size + j, epoch)

to\_rgb(input\_gray[j].cpu(), ab\_input=output\_ab[j].detach().cpu(), save\_path=save\_path, save\_name=save\_name)

# Record time to do forward passes and save images batch\_time.update(time.time() - end)

end = time.time()

# Print model accuracy -- in the code below, val refers to both value and validation if i % 25 == 0:

print('Validate: [{0}/{1}]\t'

'Time {batch\_time.val:.3f} ({batch\_time.avg:.3f})\t' 'Loss {loss.val:.4f} ({loss.avg:.4f})\t'.format(

i, len(val\_loader), batch\_time=batch\_time, loss=losses)) print('Finished validation.')

return losses.avg

#In[14]

#Training

def train(train\_loader, model, criterion, optimizer, epoch):

print('Starting training epoch {}'.format(epoch)) model.train()

# Prepare value counters and timers

batch\_time, data\_time, losses = AverageMeter(), AverageMeter(), AverageMeter() end = time.time()

for i, (input\_gray, input\_ab, target) in enumerate(train\_loader):

# Use GPU if available

if use\_gpu: input\_gray, input\_ab, target = input\_gray.cuda(), input\_ab.cuda(), target.cuda() # Record time to load data (above)

data\_time.update(time.time() - end) # Run forward pass

output\_ab = model(input\_gray)

loss = criterion(output\_ab, input\_ab) losses.update(loss.item(), input\_gray.size(0)) # Compute gradient and optimize optimizer.zero\_grad()

loss.backward() optimizer.step()

# Record time to do forward and backward passes batch\_time.update(time.time() - end)

end = time.time()

# Print model accuracy -- in the code below, val refers to value, not validation if i % 25 == 0:

print('Epoch: [{0}][{1}/{2}]\t'

'Time {batch\_time.val:.3f} ({batch\_time.avg:.3f})\t' 'Data {data\_time.val:.3f} ({data\_time.avg:.3f})\t' 'Loss {loss.val:.4f} ({loss.avg:.4f})\t'.format(

epoch, i, len(train\_loader), batch\_time=batch\_time, data\_time=data\_time, loss=losses))

print('Finished training epoch {}'.format(epoch))

#In[15]

# Move model and loss function to GPU if use\_gpu:

criterion = criterion.cuda() model = model.cuda()

#In[16]

# Make folders and set parameters

#we define a training loop and we train for 40 epochs os.makedirs('outputs/color', exist\_ok=True) os.makedirs('outputs/gray', exist\_ok=True) os.makedirs('checkpoints', exist\_ok=True) save\_images = True

best\_losses = 1e10 epochs = 40

#In[17]

# Train model

for epoch in range(epochs):

train(train\_loader, model, criterion, optimizer, epoch) with torch.no\_grad():

losses = validate(val\_loader, model, criterion, save\_images, epoch)

# Save checkpoint and replace old best model if current model is better if losses < best\_losses:

best\_losses = losses

torch.save(model.state\_dict(), 'checkpoints/model-epoch-{}-losses-

{:.3f}.pth'.format(epoch+1,losses))

#In[18]

# For displaying RESULTS import matplotlib.image as mpimg

image\_pairs = [('outputs/color/img-2-epoch-0.jpg', 'outputs/gray/img-2-epoch-0.jpg'), ('outputs/color/img-7-epoch-0.jpg', 'outputs/gray/img-7-epoch-0.jpg')]

for c, g in image\_pairs:

color = mpimg.imread(c) gray = mpimg.imread(g) f, axarr = plt.subplots(1, 2) f.set\_size\_inches(15, 15)

axarr[0].imshow(gray, cmap='gray') axarr[1].imshow(color) axarr[0].axis('off'), axarr[1].axis('off') plt.show()