

# **Model-Based AI Testing and Automation**

A Project Report  
Presented to  
The Faculty of the College of  
Engineering  
San Jose State University  
In Partial Fulfillment  
Of the Requirements for the Degree  
**Master of Science in Software Engineering**

By  
**Khajanchi Deep** ([deep.khajanchi@sjsu.edu](mailto:deep.khajanchi@sjsu.edu))  
**Kim Ilsoo** ([ilsoo.kim@sjsu.edu](mailto:ilsoo.kim@sjsu.edu))  
**Tran Chi** ([chi.t.tran@sjsu.edu](mailto:chi.t.tran@sjsu.edu))  
**Tran Hieu** ([hieu.v.tran@sjsu.edu](mailto:hieu.v.tran@sjsu.edu))  
12/2020

Copyright © 2020

**Khajanchi Deep**

**Kim Ilsoo**

**Tran Chi**

**Tran Hieu**

ALL RIGHTS RESERVED

**APPROVED**

---

Gao Jerry, Project Advisor

---

Daniel Harkey, Director, MS Software Engineering

---

Xiao Su, Department Chair

## ABSTRACT

### **Model-Based AI Testing and Automation**

By

**Khajanchi Deep**

**Kim Ilsoo**

**Tran Chi**

**Tran Hieu**

Artificial Intelligence (AI) is rapidly gaining more and more popularity in the modern world in different applications and businesses. Almost every smartphone currently has one or more AI-enabled personal assistants that are embedded in its operation system. Machine learning that applies neural networks is widely used in major business domains, such as social networks, robotics, and data science. By utilizing neural networks, people can manage time and the cost of operating big data-based applications as well as improve the accuracy of data analytics.

However, the current AI system validation lacks well-defined and experience-approved AI system validation models, quality assurance standards, and assessment methods for machine learning-based AI systems that are based on big data. Moreover, it also lacks efficient and cost-effective automatic quality validation tools. In addition, the current AI system validation comes up with several challenges. The first challenge is how to establish the quality assurance and testing coverage criteria for AI systems that use machine learning methods based on big data. The second challenge is how to prepare and conduct a clear and effective problem report and analysis for AI system developers.

In this project, the authors propose a model-based white-box testing methodology to investigate the relationship between given machine learning models and their model test coverage and accuracy coverage. The proposed solution includes visualizing neural networks' traverse inside each model, displaying linkages between nodes, and showing detailed information of each neural network layer. The test and accuracy coverage matrix also can be displayed to support users track, control, and enhance applications' quality.

## **Acknowledgments**

The authors are deeply indebted to our parents, relatives, and friends who always encourage us to pursue higher education to get higher levels on our career ladder.

## Table of Contents

<b>Chapter 1. Project Overview</b>	<b>1</b>
Introduction	1
Importance of Software testing	1
Disadvantages of Conventional testing?	1
Why Model based testing?	1
AI system, neural networks and model-based AI testing	2
Proposed Areas of Study and Academic Contribution	5
Current State of the Art	6
<b>Chapter 2. Project Architecture</b>	<b>7</b>
Introduction	7
Overview	7
Entire System Diagram	7
Model Testing	7
Tracking and Monitoring	8
Trace Analysis	8
Test Runner	8
Test Control	8
Test Result Validation	8
Architecture Subsystems	8
Tracking and monitoring diagram	8
Test Runner diagram	8
<b>Chapter 3. Technology Descriptions</b>	<b>9</b>
Client Technologies	9
Python	9
Tensorflow	9
Keras	9
ipywidgets	9
Neptune	9
Typescript	9
Web deployment	9
Middle-Tier Technologies	10
Jupyter Notebook	10
Anaconda	10
Nodejs	10
Data-Tier Technologies	10
H5 format	10
PostgreSQL	10
Cloud-based technology	10
Amazon Sagemaker	10
Amazon Elastic Inference	11

<b>Chapter 4. Project Design</b>	<b>12</b>
Client Design	12
Overview	12
UI Design	12
Activity Diagram	12
Middle-Tier Design	12
Overview	12
Tracking and monitoring Design	12
Test Runner Diagram	12
AI Tracking Library Diagram	12
Data-Tier Design	12
Overview	12
Test Data Set database schema.	12
Model Tracking information database schema.	13
Script of database schema.	13
<b>Chapter 5. Project Implementation</b>	<b>14</b>
Client Implementation	14
User interface as a web application	14
Visualization	18
Middle-Tier Implementation	18
Django framework	18
node.js server	19
Data-Tier Implementation	19
<b>Chapter 6. Testing and Verification</b>	<b>25</b>
Unit Testing	25
Overview	25
Scope	25
Test Execution Approach	25
Test Preparation	26
Test Deliverables	26
Assumptions, Dependencies & Risks	27
System Integration Testing	27
Overview	27
Scope	27
Test Execution Approach	28
Test Preparation	28
Test cases development:	29
SIT environment setup:	29
Test data preparation	29
Test Deliverables	30
Assumptions, Dependencies & Risks	30
Performance Testing	30

Overview	30
Scope	31
Test Preparation	31
Test Deliverables	33
Assumptions, Dependencies & Risks	33
<b>Chapter 7. Performance and Benchmarks</b>	<b>35</b>
Performance and benchmarking criteria	35
Benchmarking results	
<b>Chapter 8. Deployment, Operations, Maintenance</b>	<b>39</b>
Deployment strategies	39
Operational needs	39
Maintenance requirements	4
<b>Chapter 9. Summary, Conclusions, and Recommendations</b>	<b>41</b>
Summary	41
Conclusions	41
Recommendations for Further Research	41
AI White-box testing with more diverse models	41
Generating improved real-time testing reports while train the model	41
<b>Glossary</b>	<b>42</b>
<b>References</b>	<b>43</b>
<b>Appendices</b>	<b>47</b>
Appendix A.	



## List of Figures

Figure 1. Global test automation market revenue, 2015 – 2021.....	2
Figure 2. White box AI testing bed.....	5
Figure 3. System Diagram.....	7
Figure 4. Client implementation.....	14
Figure 5. User Login page screenshot.....	14
Figure 6. User registration screenshot.....	15
Figure 7. File input screenshot.....	16
Figure 8. User account access screenshot.....	16
Figure 9. User account access - visualizing various insights screenshot.....	17
Figure 10. User account access - accessing user's history screenshot.....	18
Figure 11. AWS PostgreSQL database connection screenshot.....	20
Figure 12. AWS PostgreSQL database connection.....	20
Figure 13. PostgreSQL database connected to the host from local screenshot.....	21
Figure 14. Diagram of Project database structure.....	22
Figure 15. Models.py in Django framework screenshot.....	23
Figure 16. Models.py (1) in Django framework screenshot .....	23
Figure 17. Models.py (2) in Django framework screenshot.....	24
Figure 18. Disease classification in a model.....	35
Figure 19. Image sorting based on classification.....	36
Figure 20. TensorFlow data visualization.....	37
Figure 21. Hidden layer representation.....	37
Figure 22. Collect loss and accuracy per batch.....	38
Figure 23. Collection of weights of each layer.....	38

## List of Tables

Table 1. Comparisons between neural network model applications.....	3
Table 2. Test data preparation.....	29
Table 3. Load testing.....	31
Table 4. Performance testing.....	31
Table 5. Testing tools.....	33
Table 6. Assumptions.....	33

## **Chapter 1. Project Overview**

### **Introduction**

#### ***Importance of Software testing***

Software testing is an important stage in the Software Development Life Cycle. It is the process of checking both the internal coding logics (white-box testing) and the external behaviors (black-box testing) of the software to assure it follows requirement specifications and designs. Software developers create white-box test scripts with multiple conditions to run on each branch of code to assure each node and edge are visited at least one time. Black box testing checks that behaviors of the system follow users' requirements. Software testing is a way to allow end users to implement testing standalone to understand potential risks when launching the application [1].

#### ***Disadvantages of Conventional testing?***

Conventional testing is known as the set of testing tasks that verify the developed application follows requirement specifications [2]. This approach may bring problems to testing teams when the requirements are not clear enough to verify and the scope of work is huge with a lot of test data, or when the requirements are changed regularly. This leads to the scope of regression testing being very difficult to define and may affect performance of the testing team.

#### ***Why Model based testing?***

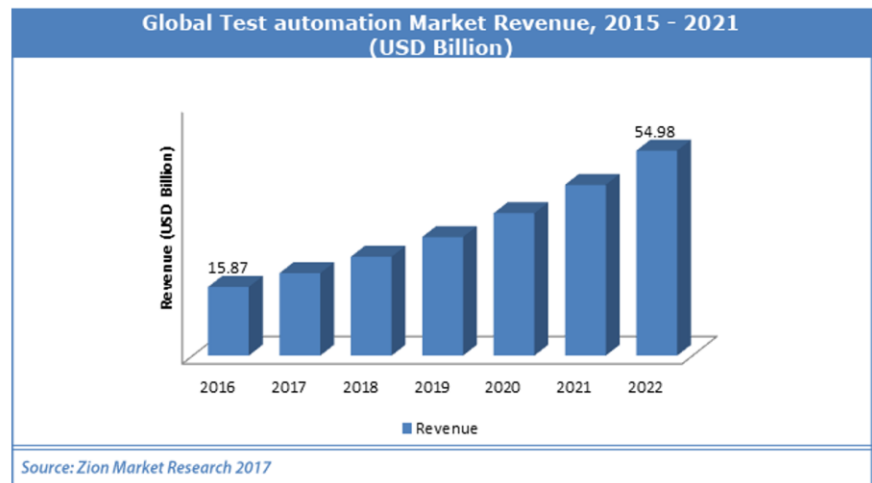
To bring more stable, maintainable, and effective testing methods to test development teams, model-based testing was introduced. The behaviors of the software systems are described by a model [3]. The models can be implemented in any stage of the Software Development Life Cycle, such as requirement specification, code automatic generation, risk analysis and test document generation [3]. Models are not only used for getting and analyzing requirements but also for developing software systems by using many different rules [3]. Models are implemented in many fields including biology and aircraft to get requirements and provide a reusable framework for product creation [3]. They are now considered as an object-oriented analysis and design solution in the software engineering process [3].

Modeling can cover the big picture of systems that includes requirement, design, testing, and deployment. When the system is enhanced, updated, or restructured, the model can be maintained and reused. These characteristics are very important and valuable for testing teams. Each time the system needs to be changed, test engineers may not have to re-do all testing tasks on the whole system or struggle with specifying scope of regression testing. Larry Apfelbaum and John Doyle said in their paper that: "By constructing a model of a system that defines the systems desired behavior for specified inputs to it, a team now has a mechanism for a structured analysis of the system" [3].

### *AI system, neural networks, and model-based AI testing*

Artificial Intelligence (AI) is a core part of building intelligent systems. There are two major directions of AI study: Humanistic AI (HAI) and Rationalistic AI (RAI) [4]. Of which, HAI is applied into machines to allow them to think and make actions like humans. In the other direction, RAI studies the effectiveness of human intelligence simulation on machines [4].

With many technological innovations and achievements in AI and big data analytics, humans are now approaching and using diverse AI software, AI applications, and smart devices. Machine learning and deep learning techniques are applied on large-scale data to train machines to create diversified AI features and capabilities [12]. Testing techniques now must be changed to adapt to AI software testing. Moreover, conventional testing cannot cover AI-based software and applications effectively [13]. It lacks well-defined and experience-approved AI system validation models and methods, as well as well-defined quality assurance standards and assessment methods for machine learning based AI systems that are based on big data [13]. Besides that, automation tests currently play an important role in validating AI products. According to Zion Market Research 2017, the global test automation market is expected to grow from 15.87 USD Billion in 2016 to 54.98 USD Billion by 2022.



*Figure 1 - Global test automation market revenue, 2015 – 2021.*

According to [5], Machine Intelligence consists of AI and Computational Intelligence. Neural networks are belonging to soft computing that is under Computational Intelligence [5], and they can detect noisy and unstable information [5]. Technically, they are a set of algorithms simulating the human brain. Hence, neural networks contain not only complex interconnected structures but also parallel computational parts that are responsible for processing simple single tasks [5]. Neural networks (NN) interpret sensory data that is perceived by machines, labelling or clustering raw data. They are designed to recognize patterns that are numerical and contained in vectors. These patterns afterwards will be translated to real-world data, such as images, text, speech or time series [6].

There are two kinds of neutral networks: feed forward neural network and recurrent neural network (RNN). A feed forward neural network is an artificial NN in which all connections

between nodes are straight forward without making any cycles. By contrast, an RNN forms a directed graph between nodes' connections following a temporal sequence. Below is the introduction of Convolutional Neural Network-CNN (as a feed forward neural network) and Long-Short Term Neural Network-LSTM (as a recurrent neural network)

- a. **LSTM:** LSTM has the ability to remember information for long periods of time. In this model, each data object, such as image, text, and speech is processed as many vectors representing objects or words inside the data object. Each vector will be recognized by analyzing four neural network layers. Each layer contains cell state, forget gate, input gate and output gate. Output gate of the previous layer contains input data for the next layer. There are 3 steps of the LSTM process: i) forget irrelevant parts of the previous state; ii) update cell state value and iii) select what parts of the current cell state for the output gate.
- b. **CNN:** Convolutional neural network is popular when it comes to computer vision tasks such as image recognition and object detection. Typically, a CNN has an input layer, several convolutional / pooling layers, an output layer, and multiple hidden layers. When adding a new layer, the matrix size is decreased.

LSTM can be combined with convolutional neural networks (CNNs) to improve quality of automatic image captioning.

There was a group of LSTMs based models used for sequence tagging that included: LSTM networks, bidirectional LSTM (BI-LSTM) networks, LSTM with a Conditional Random Field (CRF) layer (LSTM-CRF) and bidirectional LSTM with a CRF layer (BI-LSTM-CRF) [7]. Convolutional LSTM was introduced by extending the fully-connected LSTM (integrated CNN and LSTM) to forecast precipitation [8]. BI-LSTM was applied in text-to-speech systems to distinguish the relationship between any two objects when translating from text to speech [10]. Their experimental results of combining both lower and upper hidden layers that used feed forward NN and Bi-LSTM correspondingly would bring a better performance when compared to the conventional, decision tree-based, Hidden Markov Model or a Deep Neural Network Text-To-Speech system [10].

There is diverse previous research that is related to Neural Networks models. Below table is an example of the related work.

Title of research	Purpose of research	Type of model	Datasets	Results
An acoustic signature based neural network model for type recognition of two-wheelers [14]	Recognition of different types of bikes and scooters.	Average Zero Crossing rate (ZCR), Root Mean Square (RMS), and Short Time Energy (STE),	118 different models and manufacturers.	73.33% of Accuracy

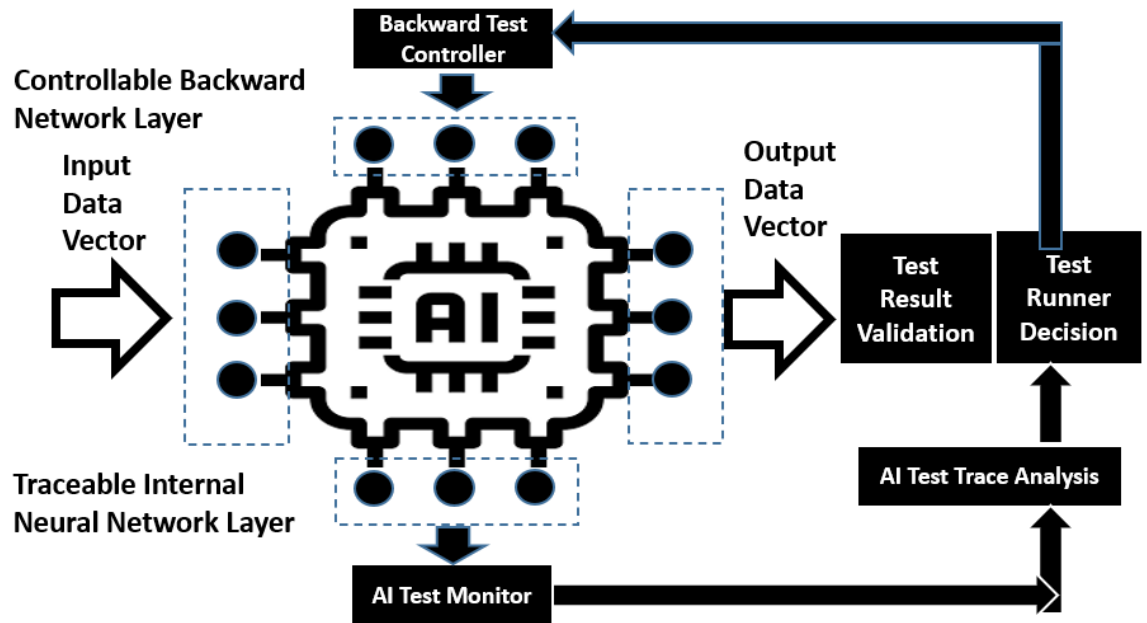
		Mean and Standard Deviation of Spectrum Centroid (CMEAN and CSD)		
Integrating a Piecewise Linear Representation Method and a Neural Network Model for Stock Trading Points Prediction [15]	Predict stock market based on history of 4 years S&P 500 index data	Back-propagation neural network (BPN)	Historical stock data (2000-2003, the S&P 500 index data)	16.2 % improved by using piecewise linear representation (PLR) and BPN compared to using only PLR.
An artificial neural network model with Yager composition theory for transformer state assessment [16]	Artificial neural network models have a better effect than traditional modes for Oil-immersed transformers.	Artificial neural network model with Yager composition theory	Different type of Power Transformer	Getting a result with 110kV oil-immersed transformer. Uncertainty factor is reduced to 0.027.
Neural Network Model on Basin Flood Prevention Effect Assessment [17]	Prevent Basin Flood with NN.	The radial basis function artificial neural network (RBF-ANN)	The assessment index system including 15 indexes and the standard including 3 levels are constructed for the Liaohe basin.	Effective
A Deep Neural Network Model for Target-based	Target-based sentiment analysis.	A deep neural network model combining convolutional	English and Chinese datasets with all 3 datasets:	CNN-RLSTM model performs better than existing

Sentiment Analysis [18]		neural network and regional long short-term memory (CNN-RLSTM)	REST, LAPT and AUTO	methods like SVM.
-------------------------	--	--	---------------------	-------------------

*Table 1 - Comparisons between neural network model applications.*

### Proposed Areas of Study and Academic Contribution

As mentioned in the table above, the achieved accuracy is different between models and datasets. A big question that needs to be answered is how to trace the models' traverse to detect data loss and defects to improve accuracy and test coverage. The authors propose a solution that allows users to evaluate test coverage and accuracy coverage by accessing all layers inside neural networks to collect metrics, such as activation, weights, bias, loss, and accuracy. The proposed solution also provides visualization of neural networks by displaying all layers, all nodes in each layer, and linkages between nodes. Users can interact with layers by clicking on each one and going to the final node. The diagram below shows how the solution works to generate final reports.



*Figure 2 - White-box AI testing bed.*

As presented in Figure 2, the input data vector is loaded into the models and will be processed through backward network layers. These layers will be controlled and traced under an AI test monitor to extract all necessary metrics to generate test analysis reports, test result validations, and test runner decisions. These tracings will be executed in loops correspondingly

with feedback network layers. The proposed solution will be implemented on machine learning and deep learning projects that utilize LSTM and CNN models.

### **Current State of the Art**

Even though white box testing for neuron networks is a new research topic, there has been interesting research coming up with innovative approaches. Despite different methodologies, the basic concepts centralize analyzing the coverage of a given model. In the notable paper on test RNN [19], Wei and his colleagues introduce three testing metrics, including cell coverage, gate coverage and sequence coverage, to analyze the behaviors of an LSTM model when dealing with various inputs. Meaningful test suits can be generated given a “set of Euclidean norm balls with the seed inputs as centers” used by an oracle when it is combined with mutation-based fuzzing [19]. Despite the great result, Wei’s approach does not hold much information on the uncovered cells / gates / sequences for a given input making it harder to compute the overall model coverage. Another outstanding work on the topic is the RNN-test [20], which was developed by Jianmin and his colleagues. The approach in this paper uses hidden state coverage, cell state coverage, gate coverage, and DX coverage. Compared to Wei’s methodology, Jianmin’s method is not restricted to just the LSTM model. The RNN-test “is scalable for variants of RNNs without the limit of the application contexts and supports multiple combinations of orientations and coverage metrics freely” [20]. When tested on the PTB language model, the method decreases the model’s valid perplexity by 1.159% and its test perplexity by 12.582% after retraining with adjusted inputs [20]. However, the downside to this approach is, again, like Wei’s, which is missing the inactivated neurons. In our paper, we propose an improved approach that captures additional coverage data giving more insights on the relation between model coverage and model accuracy.



## Chapter 2. Project Architecture

### Introduction

#### Overview

Here, the author describes our entire project architecture and explains about main components for AI model testing.

#### Entire System Diagram

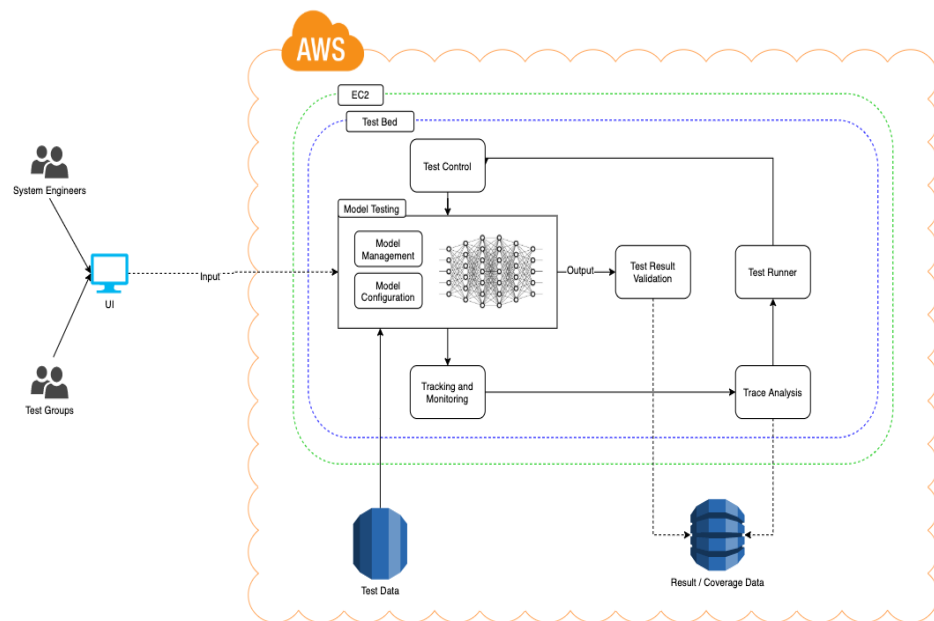


Figure 3. System Diagram

Entire system diagram will be updated more clearly after the development. There will be some minor changes.

The test bed consists of several core modules:

#### Model Testing

The module performs predefined test scripts on the selected model and test data. When a request arrives, the Model Management submodule supplies the model, and the Model Configuration unit tweaks the model setting according to the request parameters. Initially, the models used in the project are neuron networks which have built-in feedback loops. Finally, the model testing unit pulls test data from an external NoSQL database and executes the scripts.

### ***Tracking and Monitoring***

This component collects trace data from trackers embedded in used models. Even though not every node in the network has a tracker installed, the tracked neurons should be distributed within the input layer, hidden layer and output layer.

### ***Trace Analysis***

The collected trace data are analyzed and aggregated in this module. The computed coverage data is written to an external SQL database.

### ***Test Runner***

The component decides the series of test scripts to run. These scripts can be predefined by system engineers or generated programmatically on-the-fly when needed.

### ***Test Control***

The test execution can be controlled any time with this unit. In addition to the basic functionalities such as scheduling / starting / pausing / stopping tests, the test control unit has to provide the ability to swap / insert scripts.

### ***Test Result Validation***

The output of the model is validated here. The results are written to the same SQL database that stores trace data. Various materialized views are created based on these two inserted data for faster querying.

## **Architecture Subsystems**

### ***Tracking and monitoring diagram***

Tracking Library architecture diagram and monitoring sequence diagram will be updated.

### ***Test Runner diagram***

Test Runner diagram will be updated.

### **Chapter 3. Technology Descriptions**

In this project, the authors mainly focus on using machine learning and deep learning technologies, as well as web and cloud frameworks. Below is the description of technology that is divided into sections of client, middle-tier, data-tier, and cloud layer.

#### **Client Technologies**

##### ***Python***

Python is a programming language that is used to create a framework including all common libraries used by all models. These common functions are used to build dataset, manage test sets, and track quality of training models.

##### ***TensorFlow***

Open source machine learning platform that supports basic and powerful libraries to train models, collect data, build dataset, and visualize training results.

##### ***Keras***

Keras is an open-source library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

##### ***ipywidgets***

ipywidgets are interactive HTML widgets for Jupyter notebooks, JupyterLab and the IPython kernel. Ipywidgets supports users to build a simple web application implementing machine learning features.

##### ***Neptune***

Neptune is a lightweight experiment management tool that helps users keep track of machine learning experiments. With Neptune, users can easily visualize training results, such as accuracy, loss, f1-score, precision on each batch/epoch.

##### ***JavaScript***

JavaScript language is used as a core language for frontend and visualization in the project. Data is brought from machine learning model files and prediction to Django framework integrated PostgreSQL dataset. And from those backend API, it is served to JavaScript based frontend frameworks and multiple modules.

### ***ReactJS***

ReactJS is a frontend framework of JavaScript for implementing rich UI and showcases the dashboard including graphs, charts, layers movements and various comparisons among hidden layers' attributes.

### ***Typescript***

TypeScript is an open-source language which builds on JavaScript. Using Typescript to visualize neuron networks of each machine learning model.

### ***Web deployment***

- a. GitHub: all source code is managed by GitHub to check-in and checkout. GitHub also is a repository to store all trained models that will be deployed to Binder to make them sharable and being accessed in the cloud.
- b. Binder: is an open community that makes it possible to create shareable, interactive, and reproducible environments. Using Binder and GitHub to deploy all machine learning jobs in the cloud.

## **Middle-Tier Technologies**

### ***Jupyter Notebook***

The Jupyter Notebook is an open-source web application that allows users to create and share documents that contain live code, equations, visualizations, and narrative text. All programming will be run on Jupyter Notebook.

### ***Anaconda***

Anaconda distribution is a free and open-source container of Python and R programming language for computing scientists. The distribution includes data-science packages suitable for Windows, Linux, and macOS. All necessary Python packages for the project will be installed using the Anaconda environment.

### ***Nodejs***

This platform helps to run Typescript packages that are developed to visualize neuron networks inside machine learning models.

### ***Django framework***

The main purpose to implement white box evaluation in the Django framework is to fluent workflow and smooth coordination between machine learning model in python and visualization

in the frontend platform of ReactJS. Django framework is built upon a python programming language which fully supports this projects' requirements and gets compatible with extended requirements too.

For Data Science and Machine Learning models, this could be a good reference for converting their simple .py model files into a much more dynamic and powerful web application that can accept inputs from a user and generate a prediction. And from predictions and training dataset, it becomes less tough after utilizing Django framework supported for REST APIs in python language.

## **Data-Tier Technologies**

### ***H5 format***

Keras models are saved and loaded as h5 format.

### ***PostgreSQL***

The database management system that is used to store model information, links, nodes, and trained models. PostgreSQL supports inserting very large data, such as more than 100MB trained models.

## **Cloud-based technology**

### ***Amazon Sage maker***

This project will be deployed in the cloud using Amazon Sage maker. This service is the cloud machine-learning platform that enables developers to create, train, and deploy machine-learning models in the cloud.

### ***Amazon Elastic Inference***

Attach low-cost GPU-powered acceleration to Amazon EC2 and sage maker instances to reduce the cost of running deep learning inference.

## **Chapter 4. Project Design**

Author describes the project design in this chapter, it includes client UI design and back-end Middle-Tier Design.

### **Client Design**

#### ***Overview***

#### ***UI Design***

UI Screenshot will be attached.

#### ***Activity Diagram***

Activity Diagram for users will be attached.

### **Middle-Tier Design**

#### ***Overview***

#### ***Tracking and monitoring Design***

Tracking and monitoring is a main function of the middle-tier. Tracking component diagram and monitoring component diagram will be attached.

#### ***Test Runner Diagram***

Test Runner sequence diagram will be attached.

#### ***AI Tracking Library Diagram***

AI Tracking Library class diagram will be attached.

### **Data-Tier Design**

#### ***Overview***

#### ***Test Data Set database schema.***

Test Data Set will be managed on the database. The database schema for the management test data set will be attached in this part.

#### ***Model Tracking information database schema.***

It provides how we collect machine learning results data and tracking data to the database. Model tracking information database schema and diagram will be attached.

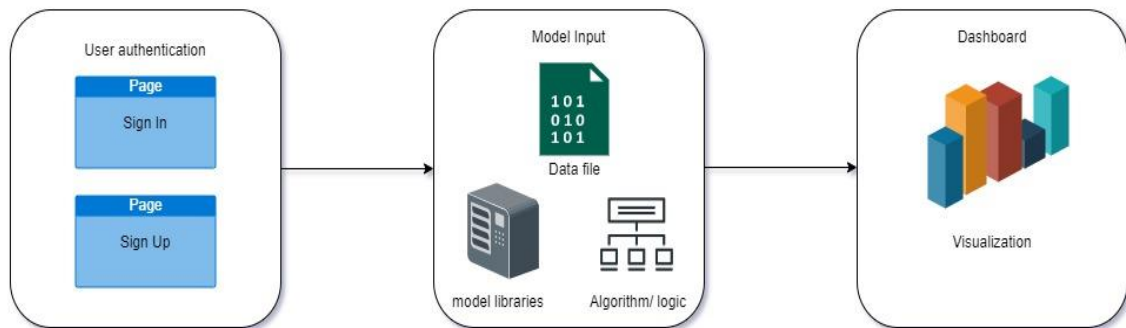
***Script of database schema.***

Script of database will be generated and attached.

## Chapter 5. Project Implementation

To make projects on white box model-based AI testing available and accessible in the real world, the author has implemented projects in various modules as in machine learning model implementation, database implementation, web application implementation with framework, and user interface implementation.

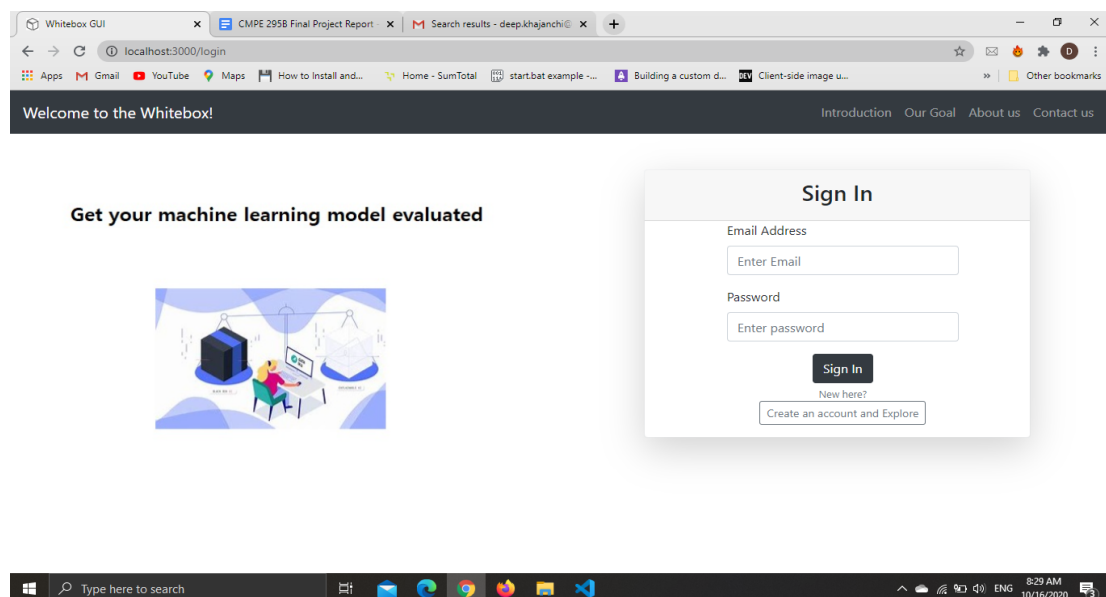
### *Client Implementation*



*Figure 4. client implementation*

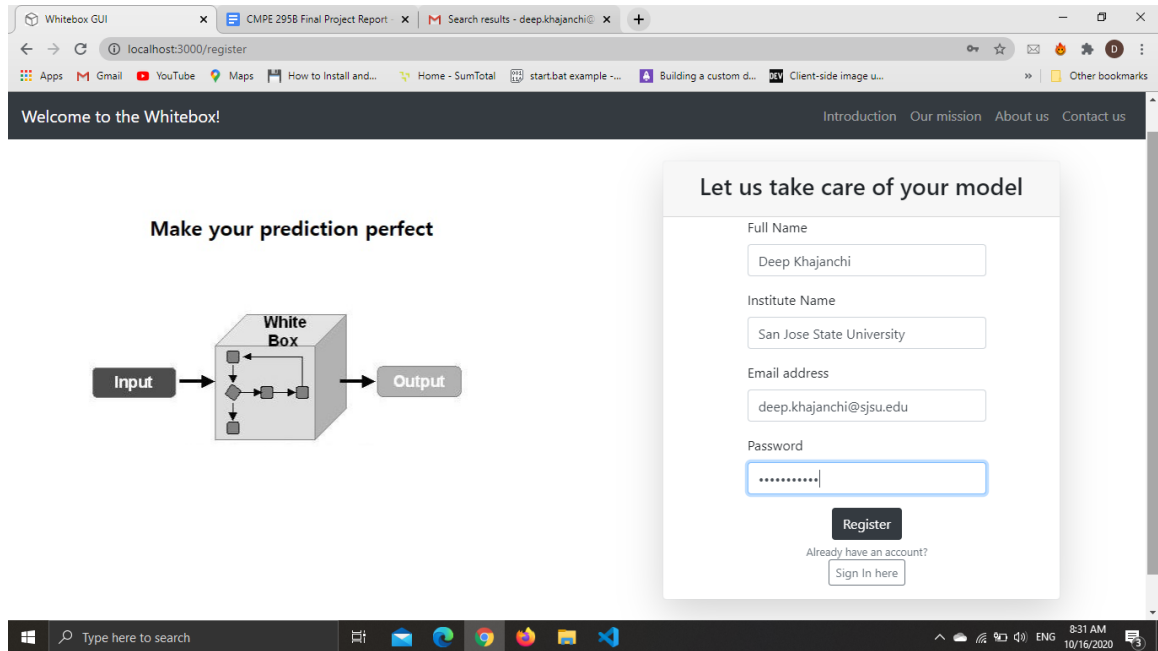
- User Interface as a web application

Author has provided a full stack web application with various components in react.js. It has the first module of user authentication with registration and login activities.



*Figure 5. User Login page Screenshot*





*Figure 6. User registration Screenshot*

Users can submit their data file either in .csv, .zip, .sql, .psql format to feed the data for the machine learning model as an initial stage. Furthermore, users need to feed library files and logic/algorithm as a core of machine learning model input. After submitting these, it passes to Django middleware before processing into the Jupyter notebook test bed for white box testing.

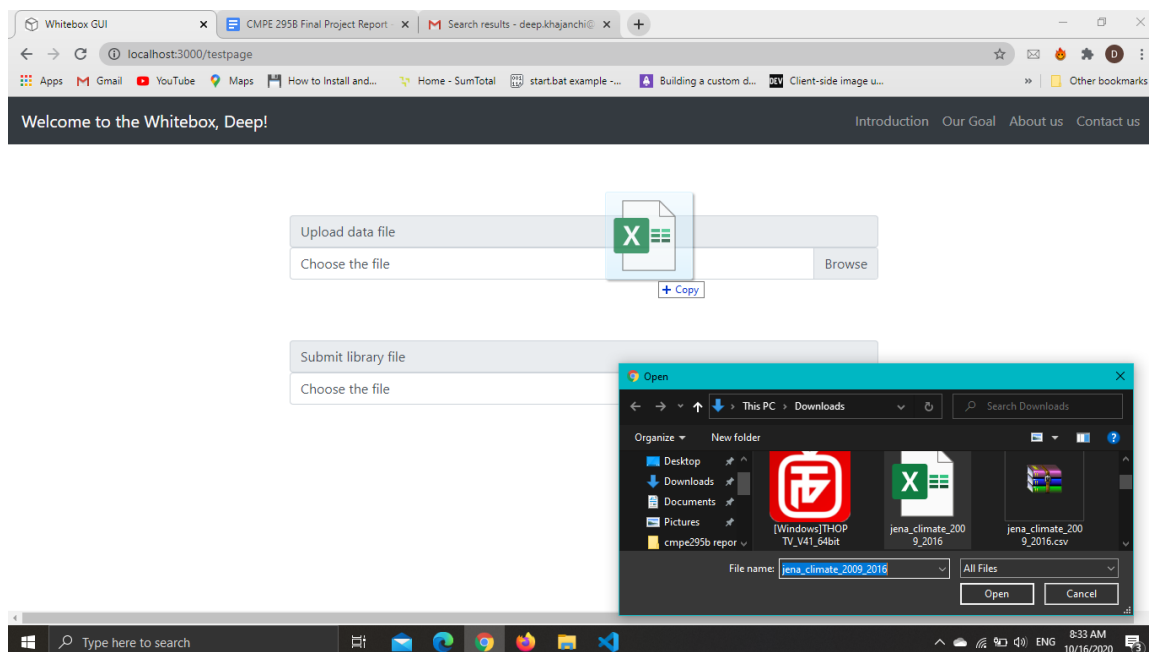


Figure 7. File input Screenshot

## • Visualization

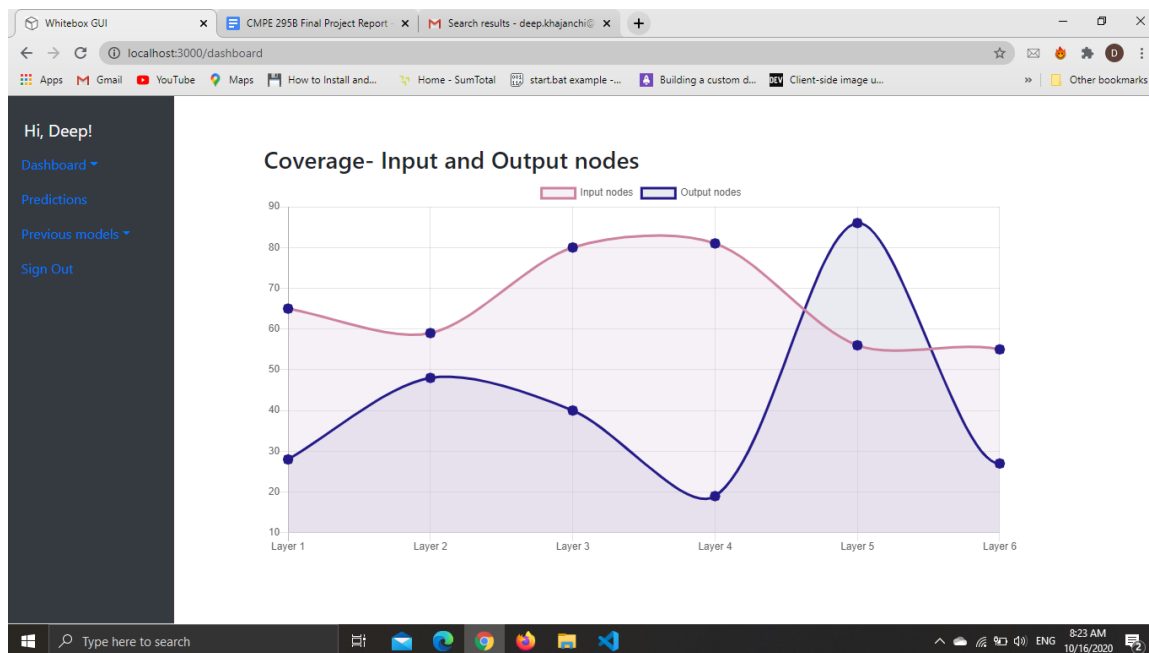
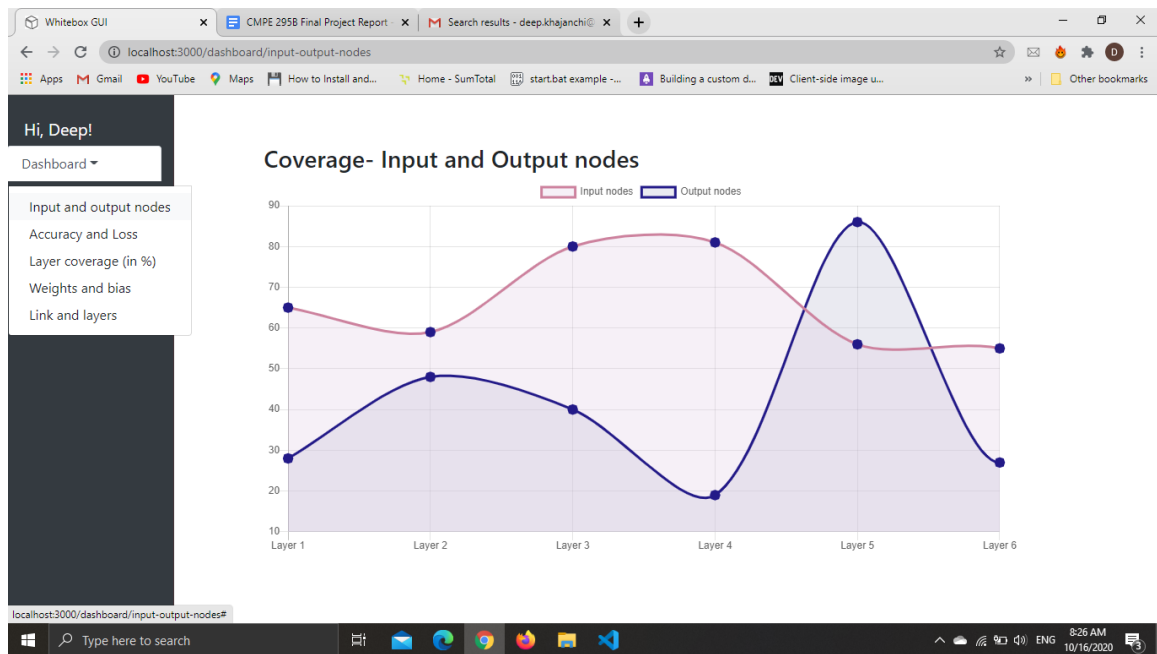


Figure 8. User account access - visualization

As a result of data process looping and filtering out the best possible results, the author is showing five types of outcomes from the test bed as in white box testing attributes.

1. Coverage comparison - input and output nodes
2. Accuracy and loss of data prediction
3. layer coverage by applied algorithm
4. Weights and biases in hidden layer
5. Relation between link and layers



*Figure 9. User account access - visualizing various insights screenshot*

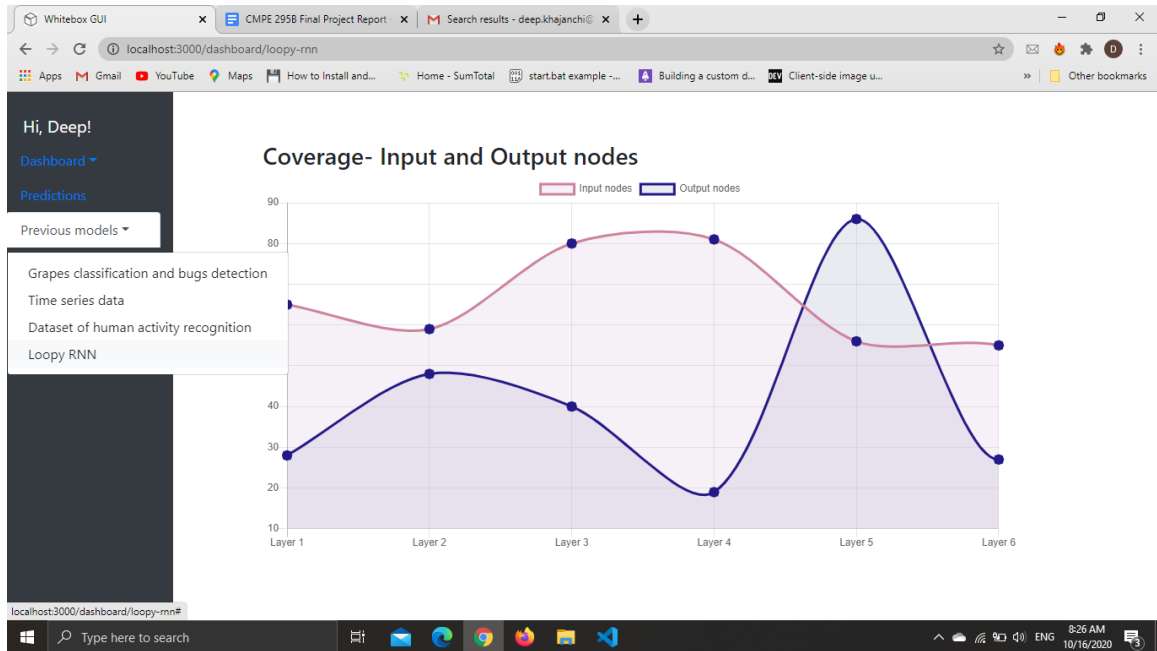


Figure 10. User account access - accessing user's history screenshot

## Middle-Tier Implementation

### *Django framework*

Author has implemented Whitebox testing in the Django framework for immense amount of dataset to be easily loaded and frequently updated with migration facility in Django.

To install Django:

*pip install django*

Inside the directory where routing file `urls.py` is located, author has created a new python file '`views.py`'. This views file would be responsible for getting the input from any user using a web page, using the same input for generating predictions and then showing this prediction back to the user using a web page.

Now inside the `urls.py` file add the below code for configuring the urls for our website (different urls for different web pages). We have first imported our views file inside the `urls.py` file and then inside the `urlpatterns` list added path for our home page (the default page whenever we open our web application instead of django default view) and the result page (for displaying the results to user) along with there respective names (using which they can be referred inside our html files).

Now we need to define these two functions inside our views.py file. Along with these two, we will also create a getPredictions function that would be able to generate predictions by loading our pre-trained model.

There are couple of things we need to understand here before moving forward.

1. Forms action parameter → View that collects the form data, generate predictions using this data and then redirects to the webpage with result. For django we have a specific way of doing this- ***action*** = “{% ***urls 'name\_of\_url'*** %}”. In our case the name of our url is result (as we have already provided this in our urls.py file).
2. Also, along with this we need to provide a {% ***csrf\_token*** %} (i.e. cross site reference forgery token) which makes data handling in django more secure. This part is mandatory and hence it should always be the first line inside your form tags.

Once we fill in the respective data and submit the form, we will be redirected to the home.js which is a react file, it will display the prediction for input data.

### ***node.js server***

Author has utilized node.js server as middleware for visualization playground in typescript scripting language and trained dataset at the back of machine learning model. This node.js server is chosen mainly because of data frequency from raw dataset and train data set after the evaluation of models.

### **Data-Tier Implementation**

While implementing the machine learning model especially in testing strategy-white box testing, it is always in the top priority to evaluate database. In fact, database is core foundation for any project. But when it comes to handle database to store as permanent data and ephemeral data, it remains brainstorming for developer.

Author has chosen PostgreSQL database as the primary database for this project. It is hosted on the Amazon Web Service to connect with local client of PostgreSQL.

The screenshot shows the 'Create database' page in the AWS Management Console. The 'Choose a database creation method' section has 'Standard Create' selected. The 'Engine options' section shows 'PostgreSQL' selected as the engine type, with version 'PostgreSQL 12.3-R1' chosen from the dropdown. Other engines like Amazon Aurora, MySQL, MariaDB, Oracle, and Microsoft SQL Server are also visible but not selected. The 'Templates' section is at the bottom.

*Figure 11. AWS PostgreSQL database connection screenshot*

The screenshot shows the 'Settings' page for a PostgreSQL database instance. The 'DB instance identifier' is 'cmpe295ab'. The 'Credentials Settings' section shows the 'Master username' as 'postgres' and the 'Master password' as '\*\*\*\*\*'. The 'DB instance size' section shows the 'DB instance class' as 'db.t2.micro'.

*Figure 12. AWS PostgreSQL database connection screenshot*

The screenshot shows a Windows terminal window titled "SQL Shell (psql)". The prompt is "psql (12.4)". A warning message is displayed: "WARNING: Console code page (437) differs from Windows code page (1252). 8-bit characters might not work correctly. See psql reference page 'Notes for Windows users' for details. Type 'help' for help." The user enters the command "project-> \l", which lists the databases. The output is a table with columns: Name, Owner, Encoding, Collate, Ctype, and Access privileges. The databases listed are postgres, project, template0, and template1. The user then enters "project-> \di", which shows no large objects. Next, the user enters "project-> \c", which connects to the "project" database as user "cmpe295ab". Finally, the user enters "project-> \du", which lists the roles. The output is a table with columns: Role name, Attributes, and Member of. The roles listed are cmpe295ab and postgres.

```

SQL Shell (psql)
psql (12.4)
WARNING: Console code page (437) differs from Windows code page (1252)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
Type "help" for help.

project-> \l
      List of databases
  Name | Owner | Encoding | Collate | Ctype | Access privileges
-----+-----+-----+-----+-----+-----
 postgres | postgres | UTF8 | English_United_States.1252 | English_United_States.1252 | 
 project | cmpe295ab | UTF8 | English_United_States.1252 | English_United_States.1252 | 
 template0 | postgres | UTF8 | English_United_States.1252 | English_United_States.1252 | =c/postgres/postgres+
 template1 | postgres | UTF8 | English_United_States.1252 | English_United_States.1252 | =c/postgres/postgres+
(4 rows)

project-> \di
      Large objects
 ID | Owner | Description
----+-----+-----
(0 rows)

project-> \c
You are now connected to database "project" as user "cmpe295ab".
project-> \du
      List of roles
 Role name | Attributes | Member of
-----+-----+-----
 cmpe295ab | Create DB, Replication | {}
          | 10 connections | 
          | Password valid until 2022-12-31 21:44:35-08 | 
 postgres | Superuser, Create role, Create DB, Replication, Bypass RLS | {}

project->

```

Figure 13. PostgreSQL database connected to the host from local screenshot



*Figure 14. Diagram. Of Project database structure*

Under the PostgreSQL database instance “cmpe295ab”, author has created multiple tables to capacitate project’s data storage requirement.

Tables are:

- (1) Account
- (2) Model\_info
- (3) Layer\_info
- (4) Data\_process
- (5) model\_classification
- (6) node\_information
- (7) weight\_information
- (8) model\_test
- (9) result



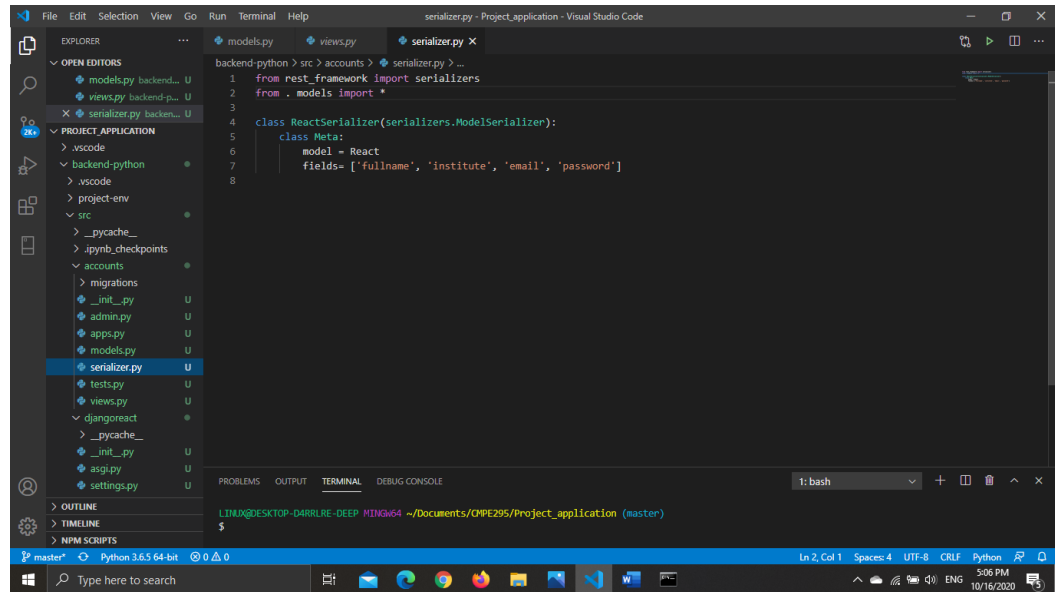


Figure 15. Models.py in Django framework screenshot

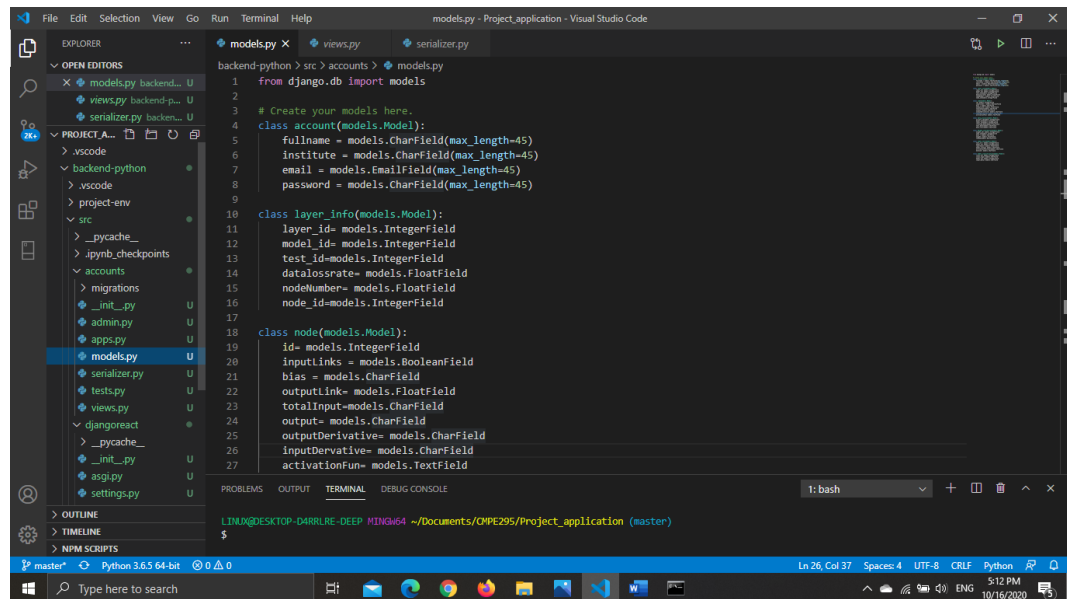
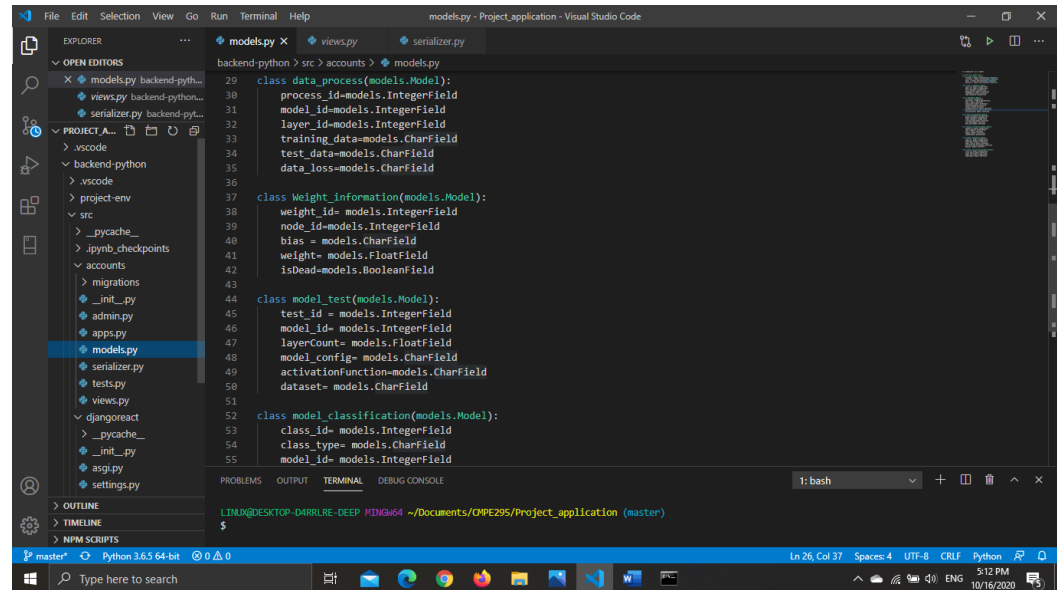


Figure 16. Models.py (1) in Django framework screenshot



*Figure 17. Models.py (2) in Django framework screenshot*

h5 format:

- To store Keras models and its large amount of dataset, author has used Hierarchical data format (HDF), which is a set of file formats (HDF4, HDF5) designed to store and organized large sort of data.

## Chapter 6. Testing and Verification

Describe your test strategy, process, and results for verifying the functionality of your project.

### Unit Testing

#### *Overview*

Each member will create unit test cases/unit test scripts and execute unit testing on their own models. This test assures all sub-components are checked at module level to qualify all requirements. Each developer should conduct a unit test report with test results and test evidence. Test cases/test scripts must cover all main business requirements of the function.

#### *Scope*

#### **Functional Testing:**

Coding logic of the four models will be implemented follow essential requirements in each module:

- Model selection.
- Model implementation.
- Model checking and monitoring.
- Report generation.

#### ***Test Execution Approach***

##### **1. UT script and unit test cases creation**

Each member will create their own unit test cases/unit test scripts that are used in their models. UT cases/UT scripts may be updated between releases to accommodate the requirements. UT cases/UT scripts should be managed by each developer.

##### **2. Execute test script and log defects**

UT is executed by developers on their own module and defects should be logged into Jira for tracking and monitoring. Test results should be recorded for evaluation and make decisions if needed.

##### **3. Fix bugs and re-test**

All defects will be fixed at this stage or some of minor ones can be accepted to keep until the final stage or approved for go live without being fixed. After the defects are being fixed, the developers will execute re-testing their module to assure no affection for their features.

##### **4. UT report**

UT cases with test results should be created to show testing progress and performance of each developer. Test report also is used to decide next actions.

### ***Test Preparation***

- Test cases/test scripts creation.
- Development environment.
- Test data preparation.

### ***Test cases/Test scripts development:***

Key principles should be used for test cases/test scripts development including:

- Test cases/Test scripts are created based on the main business requirements.
- Test cases/Test scripts can be tested with simulated data or real data.
- Test cases/Test scripts will cover positive and negative tests.

### ***UT environment setup:***

UT environment is the development environment on each machine. UT environment setup should cover below components:

- Infrastructure: personal laptop/desktop, Storage, Network Connectivity.
- Software:
  - OS, Database, support software.
- Computer for UT:
  - HPC.
  - Developers' machines.
- Test Management & Support Tools
  - Source code management tool (Github).
  - Defect tracking tool (Jira).

### ***Test data preparation***

Each member will collect data for their own module.

### ***Test Deliverables***

#### ***Manual Test deliverables:***

- Unit test cases/test scripts with test results will be documented.
- Test evidence for each test script of the high priority test cases and all the failed test cases.
- Unit test status - Weekly test report.
- Defect reports to show defects identified in UT and solutions to fix the defects.

***Assumptions, Dependencies & Risks***

***Assumptions:***

- Test data is prepared partly or fully.
- Network connectivity is stable and strong to implement neural network models and big data.

***Dependencies & Risks***

- Network slows down.
- Resources for testing not available, such as members get sick, no machine for working, HPC shutdown.

## **System Integration Testing**

### ***Overview***

Systems Integrated Test (SIT) is to confirm that the working applications or groups of components can be fully integrated to satisfy functional and technical requirements. It verifies system integrity, data, transaction flows, and connectivity and delivery of integrated business functionality across all applications, functions, and interfaces.

### ***Scope***

**Manual/Functional Testing:**

- ***System/Functional testing:***
  - Functional Testing – Manual.
  - User Interface Testing.
  - Regression Testing.
  - Re-Testing.
- ***Integration testing:***
  - Data, data conversion and database integrity Testing.
  - Regression Testing.
  - Defect Testing.

### ***Test Execution Approach***

The test execution approach for SIT will follow 4 steps as below.

#### **1. Execution of test cases and analyzing the defects**

SIT test cases will be created by the project manager for all models. All engineers execute testing and record test results for later evaluation. If any errors occurred, the detailed description of error should be logged into the defect tracking system (Jira). Person in charge of the defect will analyze the root causes, fix the defect, re-test, do a regression test, and log the detailed solution of the fixing.

#### **2. Review Test Execution Progress**

The actual test execution progress will be monitored by the Project manager against the plan and appropriate corrective action will be performed based on the progress. Execution and Defect Status review meetings will be conducted during test execution.

#### **3. Retesting of Defects**

The failed test cases shall be re-executed in the same round once the defect fix has been provided and the same has been released into the respective environment by the team as per the release frequency agreed between members.

#### **4. Test Cycle Closure**

Each cycle will be closed when the quality gate / exit criteria is met. Test report shall be prepared for every round of testing and discussed in the test closure meeting. Completeness of the respective round will be tracked in the test report and the recommendation to progress or not to the next cycle will also be embedded in the test report.

**All the release will have 3 rounds of testing as below:**

- Round 1: Functional testing (Without integration).
- Round 2: End to end testing with integration.
- Round 3: Retesting and regression testing.

### ***Test Preparation***

All necessary preparations must be done before starting SIT execution. The SIT preparation will cover:

- Test case development.
- SIT environment setup.
- Test data preparation.

***Test cases development:***

Key principles should be used for test case development including:

- Tests cases are based on the business process and can be combined to form end to end business scenarios.
- Test cases should be tested with real data.
- Test cases will cover positive and negative tests.
- Test cases will cover UI/UX usability.

***SIT environment setup:***

The SIT environment must be prepared and made ready before starting SIT execution. SIT environment setup should cover below components:

- Infrastructure: Server, Storage, Network Connectivity.
- Software:
  - OS, Database, support software.
  - Integration between members.
- Computer for SIT:
  - HPC
  - AWS Cloud
  - Developers' machines
- Test Management & Support Tools
  - Source code management tool (Github)
  - Defect tracking tool (Jira)

***Test data preparation***

Test Data Requirement	Responsibility
Grapes classification and bugs detection	Chi Tran
Time series data	Deep Khajanchi
Dataset of human activity recognition	Ilsoo Kim

Loopy RNN	Hieu Tran
-----------	-----------

*Table 2. test data preparation*

### **Test Deliverables**

#### ***Manual Test deliverables:***

- SIT test cases with test results are documented.
- Test evidence for each high priority test case and all the failed test cases will be recorded.
- SIT test status - Weekly report.
- Defect reports to show defects identified in SIT and the resolution/fix applied to resolve the defects.

#### ***Assumptions, Dependencies & Risks***

##### ***Assumptions:***

- The SIT environment will be available by the start date given in the schedule for executing the test scripts.
- Source code will be fully unit and integration tested and made available on the Test environment by the date(s) given in the schedule for executing the test scripts.
- Necessary connectivity/access to the team (if required) will be provided (such as access to HPC).
- Full and successful test execution will be considered when all the test cases have been executed and passed.

##### ***Dependencies & Risks***

- Network slows down.
- Resources for testing not available, such as members get sick, no machine for working, AWS shutdown.

### **Performance Testing**

#### ***Overview***

Software Performance testing is a type of testing performed to determine the performance of an application/product to obtain, validate or verify quality attributes of the system like responsiveness, speed, scalability, stability under a variety of load conditions.

The performance testing activity is expected to be carried over spread across key phases as mentioned below.

- Application Assessment.



- Application walkthrough.
- Performance Test Planning.
- Performance test data and scripts creation.
- Environment setup for testing.
- Test scenario will be prepared.
- Performance Testing Execution.
- Performance Analysis and recommendations.
- Performance Final Report.

### ***Scope***

Scope of performance testing will cover the following types of test.

Test Type	Description	Release 3	
		R1	R2
Load Testing	Verify performance of the system under normal and peak access.	Yes	Yes

*Table 3. load testing*

Below are the scenarios has been identified for performance testing

Events	Note	Concurrent Users (Number of Request)
Login	Login screen.	3000
Run model	Implement models.	1000
Model tracking and monitoring	Visualize and track nodes and edge traverse.	1000
Generate report	Generate test report to PDF/doc files.	1000

*Table 4. performance testing*

### ***Test Preparation***

All necessary preparations must be done before starting Performance test execution. The performance test preparation will cover:

- Identify the main scenarios.
- Performance test scripts development.
- Performance test environment setup.
- Test data preparation.
- Defined the SLA.

#### **Performance test checklist:**

The high-level activities before, during and after any performance test execution are listed below. The task list can be updated during the test.

#### ***Before the Test***

- Backup databases.
- Restart Application server and Database (if required).
- Verify all scripts, scenarios, and connectivity.
- Verify test data.
- Manually verify the application (Smoke Test).
- Start Monitoring.

#### ***During the test***

- Observe testing process.
- Monitor the application on AWS and database health.

#### ***After the test***

- Collect performance test results.
- Initiate Database refresh from backup.
- Collect performance monitoring data.
- Analyze test results.
- Send a report.

#### ***Test data preparation:***

Test data will be collected from all members' data.

#### ***Testing Tools:***

Tool Name	Tool Type	Usage/Purpose of the Tool
Jira	Test and Defect Management	Test plan, test result and defect management.
Jmeter	Performance test tool	Performance test scripts, execution and report.

AWS	Monitoring tool	To monitor the server's health
-----	-----------------	--------------------------------

*Table 5. testing tools*

Performance metrics:

- **Response and Elapsed Time**
- **TPH (Transaction per Hour):** represents the number of transactions executed in 1-hour duration of the test.
- **Throughput / Hits**

Throughput is defined as the number of requests sent in a unit of time.

### **Server-side monitoring**

Resource utilization metrics (CPU, Memory, Costly SQL's, Application logs) will be monitored and measured during all the performance tests to ensure that there are no capacity bottlenecks.

### **Test Deliverables**

- Performance test scripts.
- Performance test designed scenarios.
- Comparison between target performance state (SLA defined in test plan) and achieved performance test result.
- Final performance assessment report.

### **Assumptions, Dependencies & Risks**

#### **Assumptions:**

Assumption Head	Assumptions
Application	<ul style="list-style-type: none"> <li>● All features must be completed for development and SIT.</li> <li>● All major defects should be fixed and re-tested.</li> </ul>
Data and Database	<ul style="list-style-type: none"> <li>● Data must be real data and fully collected from all members.</li> </ul>
Environment	<ul style="list-style-type: none"> <li>● Performance test tools must be available before testing. And the environment should be monitored by a member regularly.</li> </ul>

*Table 6. Assumptions*

### **Dependencies & Risks**

- Skill of developers.
- Performance of resources.

- Resources' availability.
- Project timeline.

## Chapter 7. Performance and Benchmarks

Performance and benchmarking criteria:

- Performance evaluation in the test based
  - 1) Based on data loss and accuracy of the data
  - 2) Based on number of hidden layers
  - 3) Based on positive and negative bias
  - 4) Depending on the link and layer coordination in the hidden layer
- Performance based on data reach:
  - 1) Raw database graph after plotting the graph
  - 2) Trained dataset after looping through the hidden layer
  - 3) Various test dataset based on model's data variation
  - 4) Fetch data generation in the hidden layer

Benchmarking results:

Since authors have been working of total of four various kinds of machine learning models to test major aspects of all the models, there are the benchmarking results for couple of models such as in Grapes classification and bugs detection model, and the model of human activity recognition.

In the Grape Classification and bugs detection model, author was able to be classified various class within a model based on disease in a leaf, which should be measured as the sorting of dataset. From this sort, training and test data division is made very easy for prediction and in loop generation.

```
num_b_measles = len(os.listdir(b_measles_fold))
num_b_rot = len(os.listdir(b_rot_fold))
num_leaf_blight = len(os.listdir(leaf_blight_fold))
num_healthy = len(os.listdir(healthy))
num_phylloxera = len(os.listdir(phylloxera_fold))
num_mealybug = len(os.listdir(mealybug_fold))
total_images = num_b_measles + num_b_rot + num_leaf_blight + num_healthy + num_phylloxera + num_mealybug

print('Total Images: {}'.format(total_images))
print('There are {} images in black measles class'.format(num_b_measles))
print('There are {} images in black rot class'.format(num_b_rot))
print('There are {} images in leaf blight class'.format(num_leaf_blight))
print('There are {} images in healthy class'.format(num_healthy))
print('There are {} images in phylloxera class'.format(num_phylloxera))
print('There are {} images in mealybug class'.format(num_mealybug))

Total Images: 4796
There are 1383 images in black measles class
There are 1180 images in black rot class
There are 1076 images in leaf blight class
There are 423 images in healthy class
There are 200 images in phylloxera class
There are 534 images in mealybug class
```

*Figure 18. . disease classification in a model*

- Since the target is to track each details of the test bed of the model, node and link coverage made very easy to evaluate.
- Based on the data, shuffling, and splitting the data into train, validation and test folder seems viable for remaining future implementation.

```

Total images: 4796
Total black measles: 1383
Total black rot: 1180
Total leaf blight: 1076
Total healthy: 423
Total phylloxera: 200
Total mealybug: 534

Number of training black measles images: 886
Number of training black rot images: 756
Number of training leaf blight images: 689
Number of training healthy images: 272
Number of training phylloxera images: 128
Number of training mealybug images: 343

Number of validation black measles images: 221
Number of validation black rot images: 188
Number of validation leaf blight images: 172
Number of validation healthy images: 67
Number of validation phylloxera images: 32
Number of validation mealybug images: 85

Number of test black measles images: 276
Number of test black rot images: 236
Number of test leaf blight images: 215
Number of test healthy images: 84

```

*Figure 19. Image sorting based on classification*

One major benchmarking result is to able to load tensor board's graph which shows accuracy and loss of training and validation on tensor board:

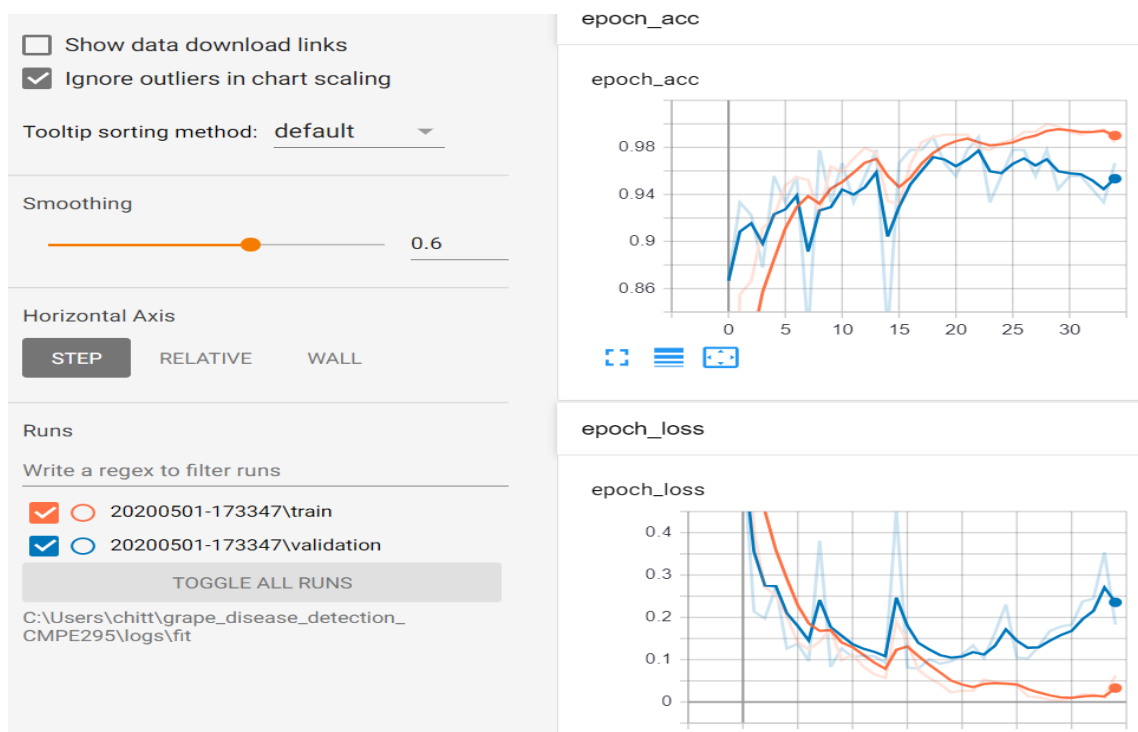


Figure 20. TensorFlow data visualization

Collecting all the hidden layers:

```
In [82]: # print all hidden layers
tf.compat.v1.disable_eager_execution()
for layer in model.layers:
    hidden_layer = layer.output

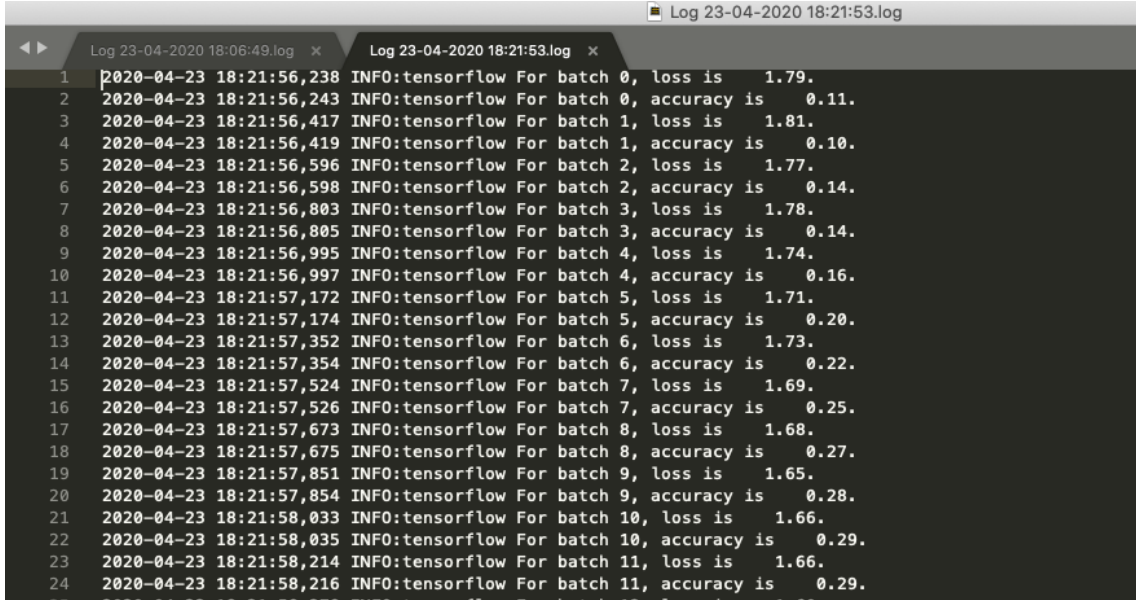
    print(hidden_layer)
```

Tensor("conv2d\_4/Identity:0", shape=(None, 254, 254, 64), dtype=float32)  
Tensor("max\_pooling2d\_4/Identity:0", shape=(None, 127, 127, 64), dtype=float32)  
Tensor("conv2d\_5/Identity:0", shape=(None, 125, 125, 64), dtype=float32)  
Tensor("max\_pooling2d\_5/Identity:0", shape=(None, 62, 62, 64), dtype=float32)  
Tensor("conv2d\_6/Identity:0", shape=(None, 60, 60, 128), dtype=float32)  
Tensor("max\_pooling2d\_6/Identity:0", shape=(None, 30, 30, 128), dtype=float32)  
Tensor("conv2d\_7/Identity:0", shape=(None, 28, 28, 128), dtype=float32)  
Tensor("max\_pooling2d\_7/Identity:0", shape=(None, 14, 14, 128), dtype=float32)  
Tensor("flatten\_1/Identity:0", shape=(None, 25088), dtype=float32)  
Tensor("dense\_2\_1/Identity:0", shape=(None, 512), dtype=float32)  
Tensor("dense\_3/Identity:0", shape=(None, 6), dtype=float32)

Figure 21. Hidden layer representation

In the Human activity recognition model, logs were the key feature to collect and to proceed further to cover possible nodes.

authors were able to collect logs based on the loss and accuracy per batch and eventually for each node in each layer.

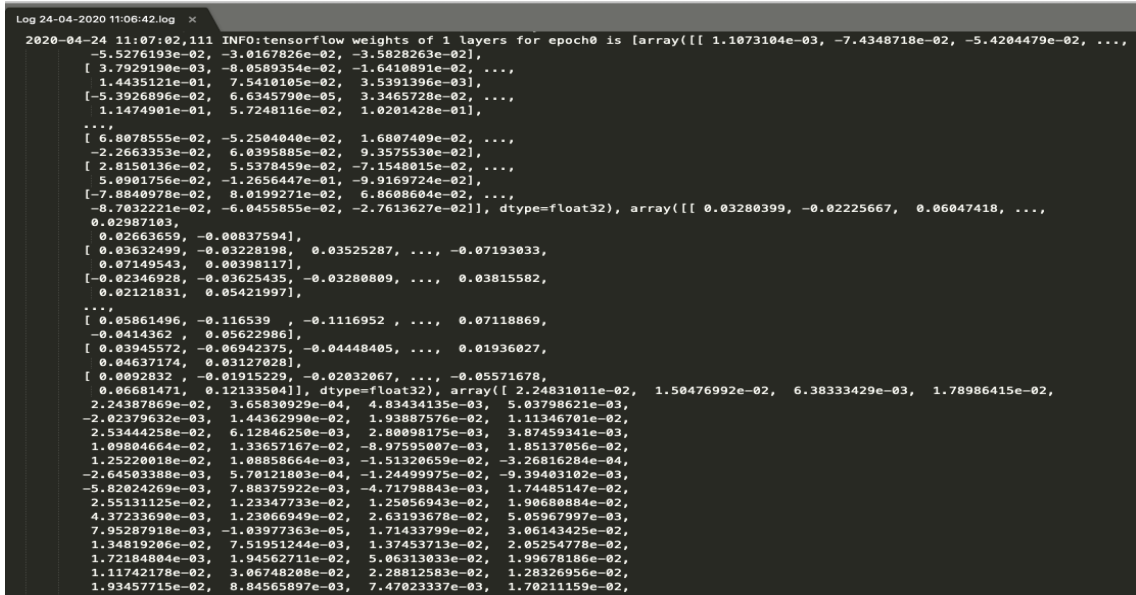


```

Log 23-04-2020 18:21:53.log
1 2020-04-23 18:21:56,238 INFO:tensorflow For batch 0, loss is 1.79.
2 2020-04-23 18:21:56,243 INFO:tensorflow For batch 0, accuracy is 0.11.
3 2020-04-23 18:21:56,417 INFO:tensorflow For batch 1, loss is 1.81.
4 2020-04-23 18:21:56,419 INFO:tensorflow For batch 1, accuracy is 0.10.
5 2020-04-23 18:21:56,596 INFO:tensorflow For batch 2, loss is 1.77.
6 2020-04-23 18:21:56,598 INFO:tensorflow For batch 2, accuracy is 0.14.
7 2020-04-23 18:21:56,803 INFO:tensorflow For batch 3, loss is 1.78.
8 2020-04-23 18:21:56,805 INFO:tensorflow For batch 3, accuracy is 0.14.
9 2020-04-23 18:21:56,995 INFO:tensorflow For batch 4, loss is 1.74.
10 2020-04-23 18:21:56,997 INFO:tensorflow For batch 4, accuracy is 0.16.
11 2020-04-23 18:21:57,172 INFO:tensorflow For batch 5, loss is 1.71.
12 2020-04-23 18:21:57,174 INFO:tensorflow For batch 5, accuracy is 0.20.
13 2020-04-23 18:21:57,352 INFO:tensorflow For batch 6, loss is 1.73.
14 2020-04-23 18:21:57,354 INFO:tensorflow For batch 6, accuracy is 0.22.
15 2020-04-23 18:21:57,524 INFO:tensorflow For batch 7, loss is 1.69.
16 2020-04-23 18:21:57,526 INFO:tensorflow For batch 7, accuracy is 0.25.
17 2020-04-23 18:21:57,673 INFO:tensorflow For batch 8, loss is 1.68.
18 2020-04-23 18:21:57,675 INFO:tensorflow For batch 8, accuracy is 0.27.
19 2020-04-23 18:21:57,851 INFO:tensorflow For batch 9, loss is 1.65.
20 2020-04-23 18:21:57,854 INFO:tensorflow For batch 9, accuracy is 0.28.
21 2020-04-23 18:21:58,033 INFO:tensorflow For batch 10, loss is 1.66.
22 2020-04-23 18:21:58,035 INFO:tensorflow For batch 10, accuracy is 0.29.
23 2020-04-23 18:21:58,214 INFO:tensorflow For batch 11, loss is 1.66.
24 2020-04-23 18:21:58,216 INFO:tensorflow For batch 11, accuracy is 0.29.

```

Figure 22. Collect loss and accuracy per batch



```

Log 24-04-2020 11:06:42.log
2020-04-24 11:07:02,111 INFO:tensorflow weights of 1 layers for epoch0 is [array([[ 1.1073104e-03, -7.4348718e-02, -5.4204479e-02, ...,
-5.5276193e-02, -3.0167826e-02, -3.5828263e-02],
[ 3.7929190e-03, -8.0589354e-02, -1.6410891e-02, ...,
1.4435121e-01, 7.5410105e-02, 3.5391396e-03],
[-5.3926896e-02, 6.6345790e-05, 3.3465728e-02, ...,
1.1474901e-01, 5.7248116e-02, 1.0201428e-01],
...,
[ 6.8078555e-02, -5.2504040e-02, 1.6807409e-02, ...,
-2.2663353e-02, 6.0395885e-02, 9.3575530e-02],
[ 2.8150136e-02, 5.5378459e-02, -7.1548015e-02, ...,
5.0901756e-02, -1.2656447e-01, -9.9189724e-02],
[-7.8840070e-02, 8.0199271e-02, 6.8608604e-02, ...,
-8.7032221e-02, -6.0455855e-02, -2.7613627e-02]], dtype=float32), array([[ 0.03280399, -0.02225667, 0.06047418, ...,
0.02987103,
0.02663659, -0.00837594],
[ 0.03632499, -0.03228198, 0.03525287, ..., -0.07193033,
0.07149543, 0.00398117],
[-0.02346928, -0.03625435, -0.03280809, ..., 0.03815582,
0.02121831, 0.05421997],
...,
[ 0.05861496, -0.116539, -0.1116952, ..., 0.07118869,
-0.0414362, 0.05622986],
[ 0.03945572, -0.06942375, -0.04448405, ..., 0.01936027,
0.04637174, 0.03127028],
[ 0.0092832, -0.01915229, -0.02032067, ..., -0.05571678,
0.06681471, 0.12133504]], dtype=float32), array([[ 2.24831011e-02, 1.50476992e-02, 6.38333429e-03, 1.78986415e-02,
2.24307869e-02, 3.65830929e-04, 4.83424135e-03, 5.03798621e-03,
-2.02379632e-03, 1.44362990e-02, 1.93887576e-02, 1.11346701e-02,
2.53444258e-02, 6.12846250e-03, 2.80098175e-03, 3.87459341e-03,
1.09804664e-02, 1.33657167e-02, -8.97595007e-03, 1.85137056e-02,
1.25220018e-02, 1.08858664e-03, -1.51320659e-02, -3.26816284e-04,
-2.64503388e-03, 5.70121803e-04, -1.24499975e-02, -9.39403102e-03,
-5.8024269e-03, 7.88375922e-03, -4.71798843e-03, 1.74485147e-02,
2.55131125e-02, 1.23347733e-02, 1.25056943e-02, 1.90680884e-02,
4.37233690e-03, 1.23066949e-02, 2.63193678e-02, 5.05967997e-03,
7.95207010e-03, -1.03977363e-05, 1.71433709e-02, 3.06143425e-02,
1.34819206e-02, 7.51951244e-03, 1.37453713e-02, 2.05254778e-02,
1.72184804e-03, 1.94562711e-02, 5.06313033e-02, 1.99678186e-02,
1.11742178e-02, 3.06748208e-02, 2.28812583e-02, 1.28326956e-02,
1.93457715e-02, 8.84565897e-03, 7.47023337e-03, 1.70211159e-02,

```

Figure 23. Collection of weights of each layer



## Chapter 8. Deployment, Operations, Maintenance

Describe any deployment strategies, operational needs, and maintenance required for your project.

### *Deployment strategies:*

#### **AWS EC2 instance**

In this project, multiple servers are required to run at a same time. There are at least three servers require to continue execute tasks:

- 1) Frontend server for visualization
  - 2) Django server to carry data from dataset
  - 3) Jupyter notebook server to run machine learning model
- To handle those servers smoothly, AWS EC2 seems optimal to spin up. All the servers' IP addresses are supposed to store in cross-origin regions as in Axios.
  - After storing cross servers' IP address, they are called in each other's entry point files to run at the same time.

#### **Amazon Sage maker**

- To store latest model's data, track logs of the white box testing, store upgraded model after testbed evaluation, Amazon Sage maker is proven best service to store those kinds of machine learning model details.

#### **AWS RDS**

- It is one of the main components for the project, PostgreSQL has enabled server on the AWS relational database service. Through the single host name, desired amount of developer can access the database to perform key functions with database.

#### **Jupyter notebook deployment**

- Two ways are there to deploy Jupyter notebook, one is on the AWS EC2 and another is to integrate it into Django framework. Since Django is the main handler of REST APIs, second option seems more doable for less load on the server.

### *Operational needs*

#### **Computational power**

- Minimum memory of 8 GB RAM for possible smooth functionalities
- Minimum storage capacity of 1 TB for downloading and installing various software
- Network adapter – 802.11ac 2.4/5 GHz wireless adapter
- Processor- minimum intel core i5(6<sup>th</sup> generation) or equivalent
- Operation system – Windows, Linux or MacOS

***Maintenance requirement:***

- Upgradation of modules, packages, and library for JavaScript frameworks for the frontend. Frontend technology is the most rapidly changing so they get deprecated in less amount of time.
- Issuing of latest license from Amazon AWS to continue cloud services and dependencies.
- Storage and cache management in cloud storage.
- Billing cycle on AWS.
- Django framework file structure changes so upgradation for that is mandatory too.

## **Chapter 9. Summary, Conclusions, and Recommendations**

### **Summary**

Summary of our project will be attached.

### **Conclusions**

Conclusions of our project will be attached.

### **Recommendations for Further Research**

#### ***AI White-box testing with more diverse models***

We covered most of the AI models in our project. However, since users can generate or develop their own model by themselves, adding a diverse testing model would be great further research.

#### ***Generating improved real-time testing reports while training the model***

If users or developers can get testing reports in real-time, it would be helpful for choosing the model and designing the model.

## **Glossary**

## References

S.S.Riaz Ahamed, “Studying the Feasibility and Importance of Software Testing: An Analysis” in *International Journal of Engineering Science and Technology*, vol.1(3), 2009, 119-128.

This paper showed what is software testing and how it performs. Software testing is for detecting errors and reliability estimation. The important part of software testing is to find the failure with all possible inputs.

[2] <http://softwaretestingguide.blogspot.com/2011/12/what-is-difference-between-conventional.html>

This website presents the difference between Conventional Testing and Unconventional Testing. The conventional testing is used for testing developed applications by test engineers. However, the unconventional testing is a verification method that is applied throughout the entire software process by the quality assurance team to check whether it follows guidelines or not.

[3] Larry Apfelbaum, John Doyle, “Model Based Testing” at *Software Quality Week Conference* in May, 1997.

This paper introduced the objective of Model Based Testing. Model based testing is used in diverse industries. It provides a reusable framework for checking quality of products. It mentions that most processes in testing do not have quantitative metrics and it is difficult to reuse.

[4] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, New Jersey, 1995.

This book introduced a modern approach to Artificial Intelligence. The authors talked about neural studying and reinforcement learning.

[5] Imre J. Rudas, János Fodor, “Intelligent Systems”, *Int. J. of Computers, Communications & Control*, ISSN 1841-9836, E-ISSN 1841-9844 Vol. III (2008), Suppl. Issue: Proceedings of ICCCC 2008, pp. 132-138.

This paper showed what is an intelligence system and what is the difference between artificial intelligence and computational intelligence. Artificial intelligence is related to hard computing, but computational intelligence comes from soft computing which has genetic algorithms and neural networks.

[6] R.C. Eberhart and Y. Shui, *Computational Intelligence – Concepts to Implementations*, Elsevier, 2007.

This paper explained computational intelligence tools and techniques. It showed the Neural Network History, components and terminology. It also explains how the Neural Network solves real problems.

[7] <https://pathmind.com/wiki/neural-network>

This web page introduces some basic knowledge of Neural Network and Deep learning.

[8] Huang, Zhiheng, Wei Xu, and Kai Yu. "Bidirectional LSTM-CRF Models for Sequence Tagging." *arXiv preprint arXiv:1508.01991* (2015).

This paper presented the application of Long Short-Term Memory based models. In this paper, they applied bidirectional Long Short-Term Memory on Conditional Random Field layer to improve quality of sequence tagging.

[9] Xingjian Shi Zhourong Chen Hao Wang Dit-Yan Yeung, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting." *Advances in neural information processing systems*. 2015.

This paper proposed using convolutional LSTM Network models to forecast weather. This LSTM Network model helped to improve the ROVER algorithm.

[10] Agarap, Abien Fred M. "A neural network architecture combining gated recurrent unit (GRU) and support vector machine (SVM) for intrusion detection in network traffic data." *Proceedings of the 2018 10<sup>th</sup> International Conference on Machine Learning and Computing*. 2018.

In this paper, the authors applied linear support vector machines (SVM) with GRU model which is called GRU-SVM model. It provided higher results than the conventional GRU-Softwaremax model. The training accuracy was ~81.54%.

[11] Fan, Yuchen / Qian, Yao / Xie, Feng-Long / Soong, Frank K. (2014): "TTS synthesis with bidirectional LSTM based recurrent neural networks", *In INTERSPEECH-2014*, 1964-1968.

This paper introduced applying Bidirectional Long Short-Term Memory to text-to-speech (TTS) systems. This neural network model provided better performance than conventional text-to-speech systems.

[12] Simon Andermatt, Simon Pezold, Philippe Cattin, "Multi-dimensional Gated Recurrent Units for the Segmentation of Biomedical 3D-Data", *Deep Learning and Data Labeling for Medical Applications*, 2016, Volume 10008 ISBN: 978-3-319-46975-1.

This paper mentioned that multidimensional Gated Recurrent Unit (MD-GRU) gave great results for the Segmentation of Biomedical 3D-Data. It took less time for running than multidimensional Long Short-Term Memory (MD-LSTM) with the same input condition.

[13] J. Gao, C. Tao, D. Jie and S. Lu, "Invited Paper: What is AI Software Testing? and Why," *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, San Francisco East Bay, CA, USA, 2019, pp. 27-2709.

This paper introduced what is AI testing, AI testing scopes and techniques, the importance of AI testing for AI-based software and applications.

[14] C. Tao, J. Gao and T. Wang, "Testing and Quality Validation for AI Software—Perspectives, Issues, and Practices," in *IEEE Access*, vol. 7, pp. 120164-120175, 2019.

This paper talked about limitations of conventional testing for AI-based software and applications. It also introduced new testing features, requirement analysis, testing techniques and testing scope for AI software.

[15] B. S. Anami and V. B. Pagi, "An acoustic signature based neural network model for type recognition of two-wheelers," *2009 International Multimedia, Signal Processing and Communication Technologies, Aligarh*, 2009, pp. 28-31.

[16] P. Chang, C. Fan and C. Liu, "Integrating a Piecewise Linear Representation Method and a Neural Network Model for Stock Trading Points Prediction," in *IEEE Transactions on Systems, Man, and Cybernetics*, Part C (Applications and Reviews), vol. 39, no. 1, pp. 80-92, Jan. 2009.

[17] Z. Lin, S. Tang, G. Peng, Y. Zhang and Z. Zhong, "An artificial neural network model with Yager composition theory for transformer state assessment," *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, Chongqing, 2017, pp. 652-655.

[18] Y. Shi, y. Zheng and M. Li, "Neural Network Model on Basin Flood Prevention Effect Assessment," *2009 Second International Conference on Intelligent Computation Technology and Automation, Changsha, Hunan*, 2009, pp. 83-86.

[19] S. Chen, C. Peng, L. Cai and L. Guo, "A Deep Neural Network Model for Target-based Sentiment Analysis," *2018 International Joint Conference on Neural Networks (IJCNN)*, Rio de Janeiro, 2018, pp. 1-7.

[20] Huang, Wei et al. "testRNN: Coverage-guided Testing on Recurrent Neural Networks." *ArXiv abs/1906.08557 (2019): n. pag.*

[21] Jianmin Guo\* , Yue Zhao\* , Xueying Han† , Yu Jiang\* and Jiaguang Sun, "RNN-Test: Adversarial Testing Framework for Recurrent Neural Network Systems", *arXiv:1911.06155*, 2019

[22] <https://towardsdatascience.com/creating-a-machine-learning-based-web-application-using-django-5444e0053a09>

[23] <https://biodatamining.biomedcentral.com/articles/10.1186/s13040-017-0154-4#:~:text=Introduction,been%20applied%20as%20'standards'>



## **Appendices**

### **Appendix A.**