

Acknowledgement

The satisfaction that accompanies the completion of any task would be incomplete without the mention of the people who made it possible, and whose constant guidance and encouragement helped us in completing the project successfully. We consider it a privilege to express our gratitude and respect to all those who guided us through the course of the project.

We would like to extend our gratitude and thankfulness to Dr. Ramesh Jain and Dr. Siripen Pongpaichet, whose knowledge and research materials on Future Health and Computer Vision were the key for us to crack all the dead ends that we came across.

We would also like to extend our sincere gratitude to Neil Jain, Asquith Bailey and Pinaki Sinha of Krumbs Inc. for guiding us throughout the course of the project and providing us with useful resources including Krumbs SDK.

We extend our sincere thanks to Dr. Dinakar Sitaram, Professor, for his constant guidance, encouragement, support and invaluable advice, without which this project would not be where it is today.

We extend our gratitude to Prof. V. R. Badri Prasad, Associate Professor, for his overall co-ordination of our project and also for helping us in stages where we desperately needed such guidance.

We express our gratitude to Prof. Nitin V. Pujari, Professor and Head of Department of Computer Science and Engineering, PES Institute of Technology whose guidance and support has been invaluable.

We would like to express our heartfelt thanks to Prof. M. R. Doreswamy, founder, Prof. D. Jawahar, CEO, Dr. K. N. B. Murthy, Vice Chancellor and Dr. K. S. Sridhar, Principal, for providing us with the congenial environment for presenting this project.

Abstract

In the modern era where there is an enormous degree of Human-Computer-Interaction, people resort to technology to advise them on their food habits and maintaining wellness. Although many vendors deliver restaurant information to users at their fingertips, there is a huge divide as to the person's expectations and restaurant search engines.

Existing apps take simple manually entered attributes like location, price etc. into consideration while endorsing restaurants. The user has to enter textual data regarding all the dishes consumed so that fitness apps make dietary recommendations. However, learning people's taste preferences and accurately predicting their dining interests cognitively bridges the gap between "Needs" and "Resources".

We propose SpoonSnap, an intelligent food logging framework designed to meet a foodie's palate preferences, travel constraints, and fine-grained cuisine predisposition. SpoonSnap enables a simple and accurate food preference profiling procedure by leveraging image classifiers powered by Deep Learning and an Indian Food Database filled with images that we scraped. The framework also provides means to generate accurate personalized dish or restaurant recommendations based on the food habits of the user.

Table of Contents

1. Introduction.....	7
1.1 Problem Definition.....	8
1.2 Generic Proposed Solution.....	8
1.2.1 Using Feature Extraction from Images	8
1.2.2 Using Deep Neural Networks	9
1.3 Acknowledgement.....	10
2. Literature Survey	11
3. System Requirement Specification	13
3.1 High Level Block Diagram.....	13
3.2 Environment Used	13
3.2.1 Hardware Required	13
3.2.2 Software Required	13
3.2.3 Project Requirements	14
3.2.4 Constraints and Dependencies	15
3.2.5 Assumptions.....	15
3.2.6 Use Case Diagram	16
3.2.7 Requirement Traceability Matrix.....	17
4. Schedule.....	18
4.1 Gantt Chart.....	18
5. System Design	19
5.1 Architectural Diagram	19
5.2 Sequence Diagram / Interaction Diagram.....	20
5.3 User Interface Design	20
5.3.1 User Login and Register	21
5.3.2 Image Upload and Reporting	22
5.3.4 Time Separated Logger.....	24
5.4 Updated RTM	25
6. Design	26
6.1. Indian Food Data Scraper	26
Our model has multiple output layers unlike a traditional Neural Network:.....	31
6.6 Updated RTM	33

7. Implementation	34
7.1 Pseudo Code / Algorithms	34
7.1.1 Android Front-end	34
7.2 Codebase Structure	38
7.3 Coding Guidelines Used	40
7.3.1 Code Indentation.....	40
7.3.2 Lines.....	40
7.3.3 Imports	40
7.3.4 String Quotes	41
7.3.5 Whitespaces in expressions and statements.....	41
7.4 Code Snippets	42
7.4.1 StarterApplication.....	42
7.4.2 Fetching Daily Uploads of a User.....	43
7.4.3 Extracting Dish Features.....	44
7.4.4 Login Preference Function.....	45
7.4.5 Logger Dashboard Fragment Layout.....	46
7.4.6 Connecting and Listening to Amazon SQS	47
7.4.7 Sending JSON to Remote Server.....	47
7.4.8 General Feature Extraction from Clarifai	48
7.4.9 Food Feature Extraction from Custom Trained Model.....	48
7.4.10 Storing New User Details on the Server	49
7.4.11 Concept Search	49
7.4.12 Image Scraper Snippet.....	50
7.5 Unit Test Cases	51
7.6 Metrics for Unit Test Cases	52
7.7 UPDATED RTM	52
8. Testing	54
8.1 System/Function test Specifications for the Project	54
8.1.1 Test Dataset Distribution	54
8.1.2 Test UI	54
8.2 Test Environment.....	54
8.2.1 Software Environment	54
8.2.1 Hardware Environment.....	55
8.3 Test Procedure	55
8.4 Example Test Result	56

8.5 Test Metrics	57
8.5.1 Test Statistics	57
8.5.2 Effort Variance	58
8.6 Updated RTM	58
9. Results and Discussion	60
10. Retrospective	61
10.1 What went well	61
10.2 What did not go well.....	61
11 References.....	62

List of Figures

Figure No.	Title	Page No.
1	Generic Proposed solution using feature Extraction	8
2	Generic Proposed solution using Deep Learning	9
3	High Level Architecture	13
4	Use Case Diagram	16
5	Estimated Gantt Chart	18
6	Architecture Diagram	19
7	Sequential Diagram for the Framework	20
8	Login and Registration Screens	21
9	Image Reporting	22
10	Speech Input and Feature Extraction	23
11	Logger with Navigation Drawer	24
12	ASP 106 Images	26
13	Neo4j Visualisation of ASP 106 dataset	27
14	Deep Learning vs. Machine Learning Feature Engineering	28
15	Database Schema	29
16	AI Model Overview	30
17	Clarifai Deconvet Architecture for enhancing feature Enginnering	31
18	mediaJSON structure	34
19	Java Codebase Structure	38
20	XML Codebase Structure	39
21	Concepts Detected in the Image	56
22	Caltech 256 classification accuracies	57
23	Effort Variance	58

1. Introduction

Increasingly, culinary artists as well as healthcare professionals are looking at technology to enhance food which in turn will enhance quality of life. Culinary artists are interested in making our multi-sensory food experience more exciting and pleasant. Healthcare experts are interested in helping us enjoy our culinary experience as long as possible by increasing our life healthy as well as long.

In recent years, much has been written about the Internet of Things and its potential to revolutionise the way businesses and public services operate. The vision, fast becoming reality, is for networks of connected sensors that gather data from factories, vehicles, hospitals, homes, shops and supply chains across the globe. Human Computer Interaction is being hailed as the technology that will enable everything from smart cities that can, for example, optimise traffic flow, energy usage and signage, to systems that predict earthquakes. The food industry is smartening up. With the help of the IoT, food processors and suppliers are working together to reduce maintenance costs, uncover opportunities, influence productivity, and transparently track the supply chain. Connecting to the cloud improves global manufacturing processes and shows where business stands to grow. Food suppliers are offering smart ways to stay connected and spot opportunities using state-of-the-art recommendation systems. By connecting people, processes, and data together, the software makes decision making easier and expedites the product to market process.

We want to apply technology to help people enjoy food by selecting right food, at right time, at right place, in right company. We see role of technology at various stages ranging from finding what to eat, to eat balanced nutritious food, be sensitive to their personal health situation, and satisfy their personal culinary tastes. There is very little to help healthy foodies currently. We want to change this situation by developing technology to collect, manage, and use big food data to assist individuals manage their culinary and health experience using modern wearable and smartphones.

1.1 Problem Definition

The advent of smart-citizens has begun. Foodies have access to their directory of eating places by manually using search engines on their phones. What they do not have access to, is the power of eating places automatically recommended to them according to their personal preferences. There is no means for personal devices of people, for example Foodies, to explore interesting places to eat autonomously. Present day wellness applications have minimal functionality for log-based analysis. Wearable devices don't fully understand the wearer's palate and we aim to provide that power to these devices.

1.2 Generic Proposed Solution

We aim to use our Food Logger to build a human model, a foodie who enjoys eating at different places. Acceptable to his/her personal whims, we let him explore places without having him use technology for propagating his thought. We eliminate the task for the foodie to explain his/her taste preferences and detect his/her consumption patterns for making eating-place recommendations. Foodies love to judge and rate places and dishes they eat and look back to all the exquisite foods eaten and we provide them a means for them to do so using our intelligent logger.

1.2.1 Using Feature Extraction from Images

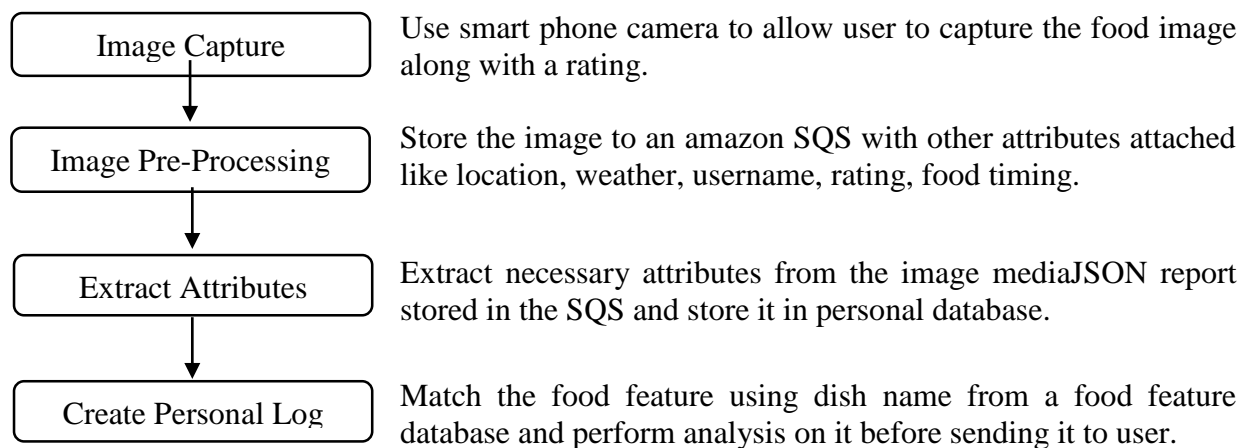


Figure 1: Generic Proposed Solution using Feature Extraction

1.2.2 Using Deep Neural Networks

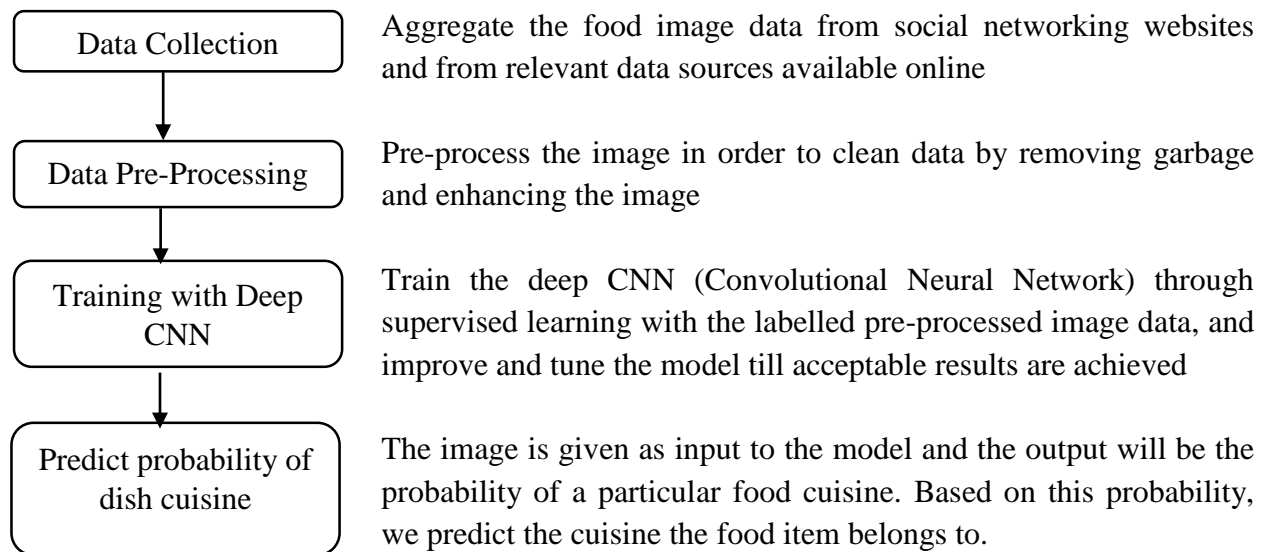


Figure 2: Generic Proposed Solution using Deep Learning

1.3 Acknowledgement



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING PES INSTITUTE OF TECHNOLOGY

(An Autonomous Institute under VTU, Belgaum)

100 Feet Ring Road, BSK- III Stage, Bangalore – 560 085

Project ID: PW066

Project Title: SpoonSnap: The Intelligent Food Framework using Deep Learning

Project Team:

Animesh Sahay	1PI13CS027
Syed Munawwar Quadri	1PI13CS174
Prashanth TK	1PI13CS175

This project report was submitted for review on _____. I acknowledge that the project team has implemented all recommended changes in the project report.

Guide signature with date:

Guide Name: Dr. Dinakar Sitaram

2. Literature Survey

The following are the major applications which are most similar to our framework with respect to motivation and functionalities.

1. MyFitnessPal (MyFitnessPal, 2016):

This is a mobile phone app that possesses a gigantic food database. The app monitors sodium, vitamin, cholesterol intake and other nutritional information that can give you a better idea of what you're eating beyond a simple daily calorie limit. Primarily a nutrition tracking database and vibrant social community that just wants to help you get a handle on the foods you eat. Since it is aimed at maintaining personal wellness by monitoring

2. Healthify (LTD, 2016):

It provides the following features -:

- 1) Calories consumed in a day
- 2) Amount of water consumed in a day
- 3) Connects to a smart tracker
- 4) Calories burnt in a day
- 5) Set a weight loss goal

It also provides personal trainer by paid services, who will we look into your goals and decide the food you consume and suggest you the activities which is good for you. It analyses your food log and suggests future meals based on the nutrition. The shortcomings are the same as MyFitnessPal.

3. NutritionIX (Nutritionix, 2015):

NutritionIX is a popular food logging and fitness recommendation application which is popular because of its huge DB of restaurants and their menu including the nutritional values of the product. The following are its features -

Full feature dashboard that has different meals of the day like Breakfast, lunch, Dinner and the respective calorie intake in them, this info is organised on the basis of per day limit.

The calorie intake is given by either searching the food item or scanning the barcode.

One interesting feature is we can select the restaurants we have been to and can add whatever we ate from there and its respective calorie intake.

Detailed stats about the calorie intake categorised on the basis of day, week, and month.

Push notifications every day at 9 to remind to log food.

4. Yum-me:

Yum-me is a Cornell Tech project centred on a personalized nutrient-based meal recommender system designed to meet individual's nutritional expectations, dietary restrictions, and fine-grained food preferences [2]. Yum-me enables a simple and accurate food preference profiling procedure via a visual quiz-based user interface, and projects the learned profile into the domain of nutritionally appropriate food options to find ones that will appeal to the user. Although they also “learn about the user”, they offer recipe recommendations after making the user enter his/her preferences manually, which can be quite tedious in comparison to the leading commercial mobile applications. Our approach is more marketable due to the enhanced user experience and reduced app usage time. By providing restaurant recommendations, we act like the middle-man between Zomato and the user. The difference is that instead of the user manually searching for preferable places to dine in, we will bring the results to him without him having to do any work.

[URL: <https://arxiv.org/abs/1605.07722>]

3. System Requirement Specification

3.1 High Level Block Diagram

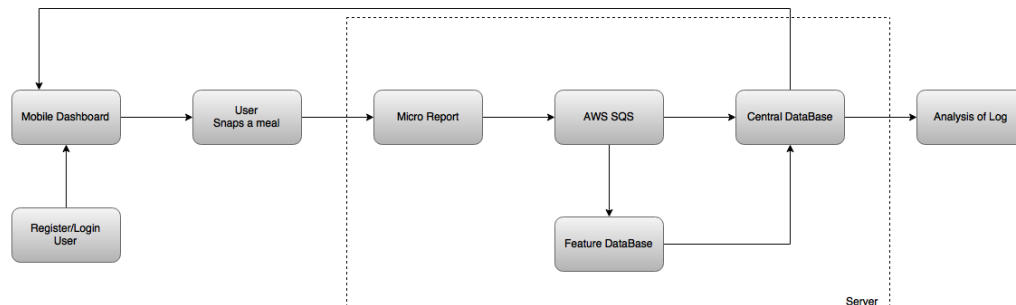


Figure 3: High Level Architecture

3.2 Environment Used

The hardware and software, and project requirements are stated in the subsections that follow:

3.2.1 Hardware Required

The framework requires faster computation, so the following hardware is required -

1. **Processor** : Intel, 1.2Ghz or higher
2. **RAM** : 4GB or more
3. **Hard Drive** : 500 GB or more
4. **Servers** : Storage of 250 GB with 8 GB RAM
5. **Smart Phones** : Android phones with 2 GB RAM and 50 MB storage

3.2.2 Software Required

1. Windows 7+ OS
2. Android Studio 2.2.1
3. Krumbs Starter Application/SDK
4. Clarifai API
5. MySQL
6. Apache
7. Wampp Server
8. Google Drive, docs, slides
9. Sublime Text Editor
10. Node.js Runtime

3.2.3 Project Requirements

3.2.3.1 Functional Requirements

F1	The user should be able to use the camera in their smartphones to capture image of the food item along with the rating associated.
F2	The system should accept image uploads along with option of recording audio and video.
F3	The system should predict using the CNN with confidence of prediction.
F4	The system should be able to ask user for entering the food item name using Google voice and speech API.
F5	The system should query the database with the food item name and display the features of the food consumed.
F6	The system should send the features of the food item along with the image to user's log.
F7	The system should analyse this data and give user the detailed report about what they eat by day, week and month.

3.2.3.2 User Interface Requirements

U1	Log In or Register	The user should be able to Log in to the system using the correct credentials or create a new account if a new user.
U2	Capturing Image	Using the smart phone's camera the user should be able to capture the food item image along with their rating associated with it.
U3	CNN Output	The system should predict the cuisine of the food item by using the clarifai CNN and should display back the prediction to user.
U4	Food Feature and Log	The system should take a speech input from user to confirm the food item name and provide all the food features stored in a database along with the food image, time and location.

3.2.3.3 Non-Functional Requirements

NF1	Usability	The system should have well designed interface which can facilitate new users to operate the system.
NF2	Security	The system should secure the identity of the users by maintaining anonymity of their personal and food data.
NF3	Testability	The Classifier should be able to give most accurate cuisine recommendations.
NF4	Performance	The system should have a good response time.

NF5	Compatibility	The system should work on all modern android devices and OSs.
NF6	Extensibility	The system can be extended to predict more food items.

3.2.4 Constraints and Dependencies

1. The images should be of good quality so that the classifier can be trained more accurately.
2. The amount of data available for training our system is limited i.e. for 106 Indian food items as there is no food image dataset available for Indian dishes.
3. A labelled dataset is required to train the model.
4. A general knowledge of basic computer and smart phone skills is required to use the application.

3.2.5 Assumptions

1. The user hasn't finished eating his food and captures the best possible image of the food item.
2. All the images input into the system are of accepted quality to train as well as to detect the cuisine.
3. The serving of the food item is assumed to be one single and thus the volume is also standard.
4. The food item is prepared in a standard way with the basic ingredients.

3.2.6 Use Case Diagram

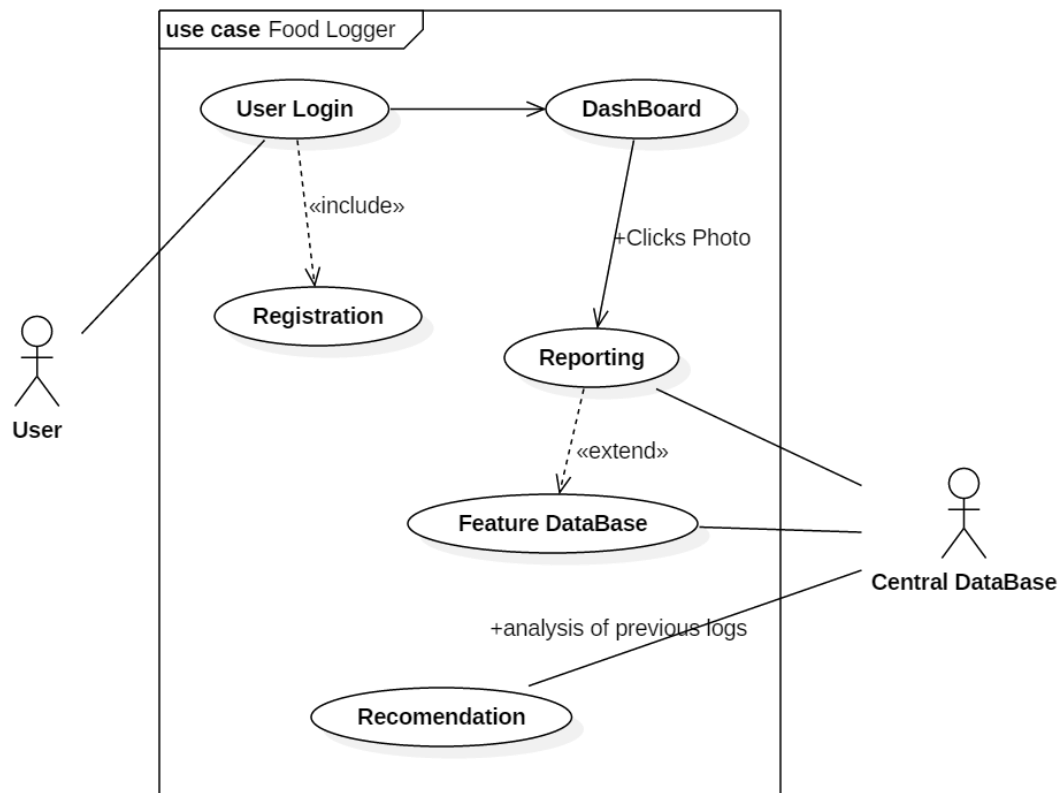


Figure 4: Use Case Diagram

Use Case Description -

Description - A system which allows user to take a photograph of a food item, log it and receive smart recommendations on the basis of the food choices.

Primary Actors - User, System framework.

Secondary Actors - System administrators.

Pre-Conditions - User must have an android device and internet connectivity.

Post Conditions - User is recommended.

3.2.7 Requirement Traceability Matrix

ID	Name	Requirement	Implementation	Testing
F1	User Login			
F2	Intent Capture			
F3	Image Upload to server			
F4	SQS listener			
F5	Clarifai Prediction			
F6	Feature Extraction			
F7	Dashboard			

4. Schedule

4.1 Gantt Chart

SpoonSnap

Gantt Chart for the period jan - may 2017, assuming numbers as dates.

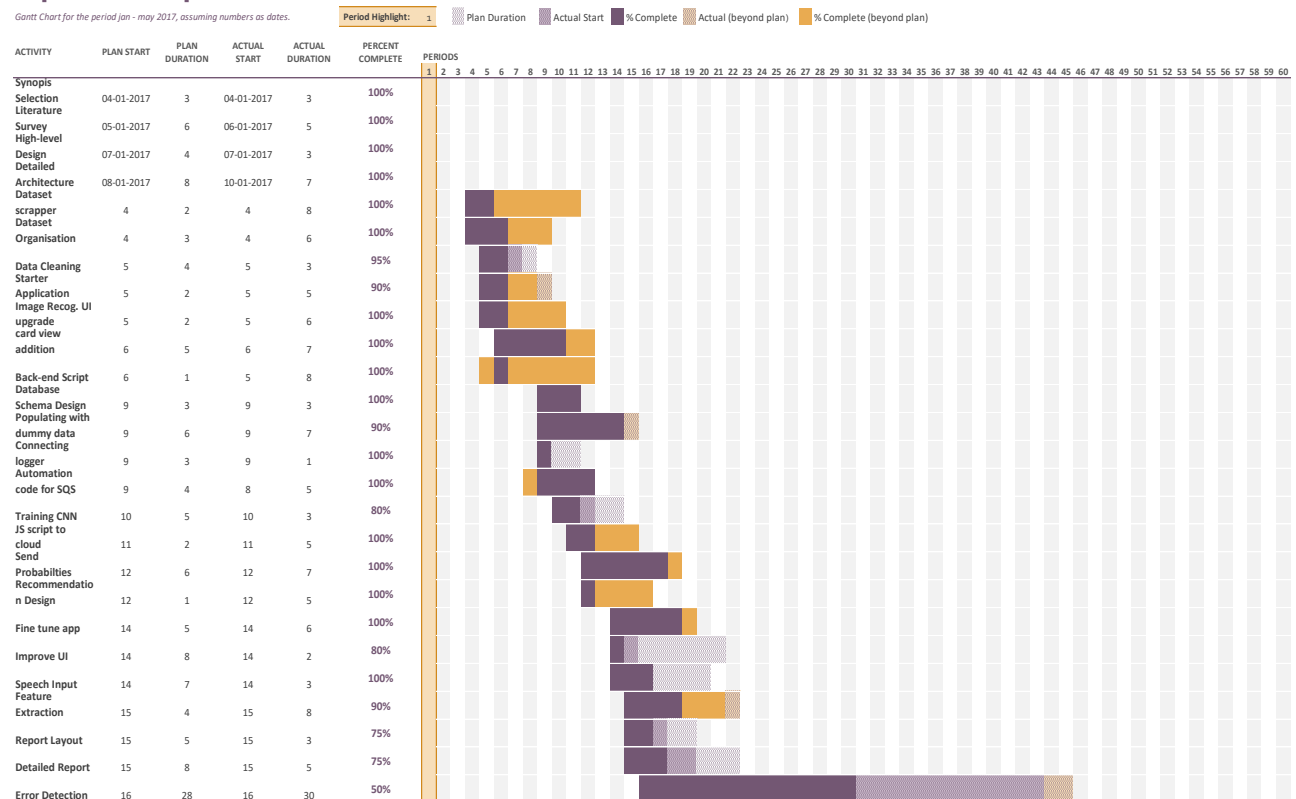


Figure 5: Estimated Gantt chart

5. System Design

5.1 Architectural Diagram

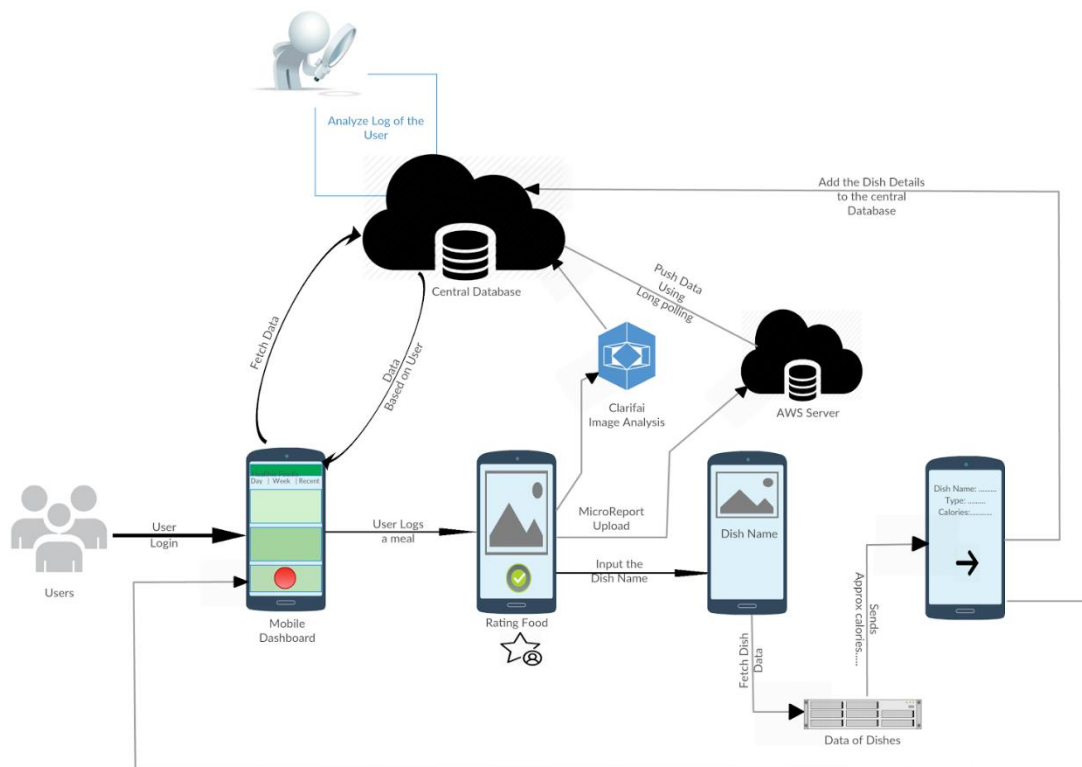


Figure 6: Architecture Diagram

5.2 Sequence Diagram / Interaction Diagram

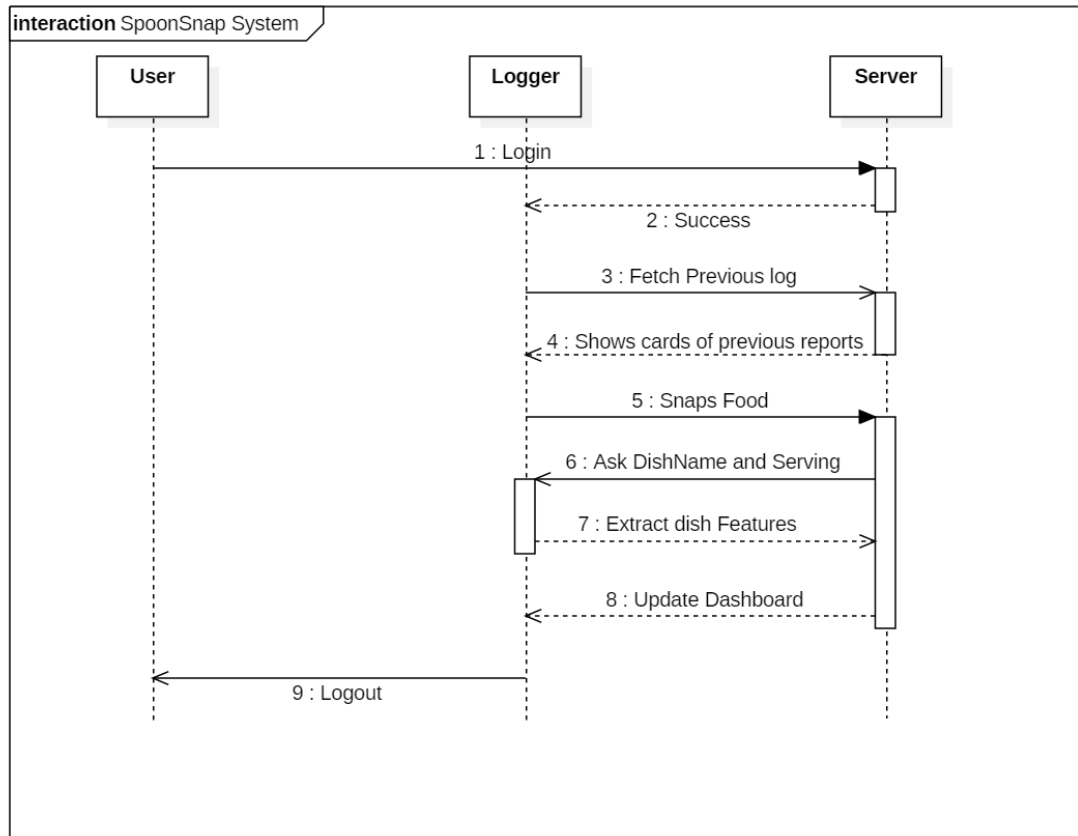


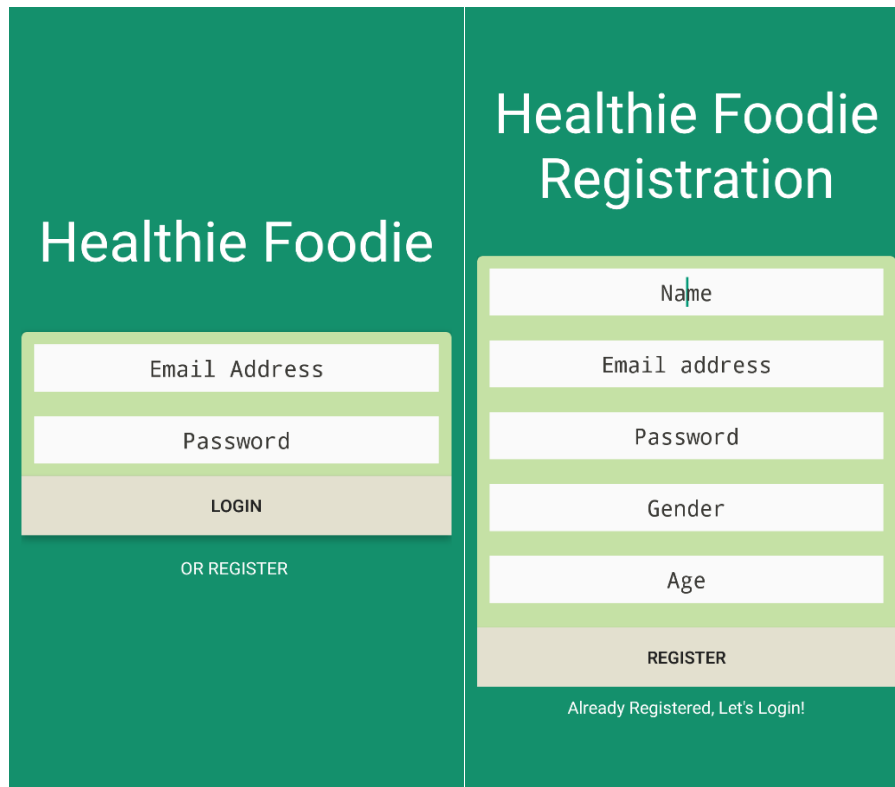
Figure 7: Sequence diagram for the framework

5.3 User Interface Design

The front-end application will be an android application with advanced material design and latest android developer's templates. The basic layout will consist of a camera activity in the background which will be used to capture images. Krumbs, a micro-report capturing platform is used to convert images to micro reports and is used to communicate. The other activity is the recommendation, in this a card view will be designed which will be presented to the user after all the computation. The neighbour to recommendation activity is the logger activity, this will be used to log all the food items of the user and will have different filters like sorting by month, day and week. The android application's UI design is explained in detail in further sections.

5.3.1 User Login and Register

All the users have to login to the application using their correct credentials, these credentials like username, password are useful for maintaining individual log of user and also improves security of the application. If there is a new user he/she has to register to use the application.



The figure displays two mobile application screens for 'Healthie Foodie'.

Left Screen (Login):

- Title: Healthie Foodie
- Input fields: Email Address, Password
- Button: LOGIN
- Link: OR REGISTER

Right Screen (Registration):

- Title: Healthie Foodie Registration
- Input fields: Name, Email address, Password, Gender, Age
- Button: REGISTER
- Link: Already Registered, Let's Login!

Figure 8: Login and Register Screens

5.3.2 Image Upload and Reporting

The next activity allows the user to capture an image by a single click of a button. The section is divided into 4 categories naming breakfast, lunch, dinner and others with each category having a rating from *divine* to *yuck*. This helps our system to capture a person's individual rating for the food item. Also other attributes like username, location, time etc. is captured in this phase.

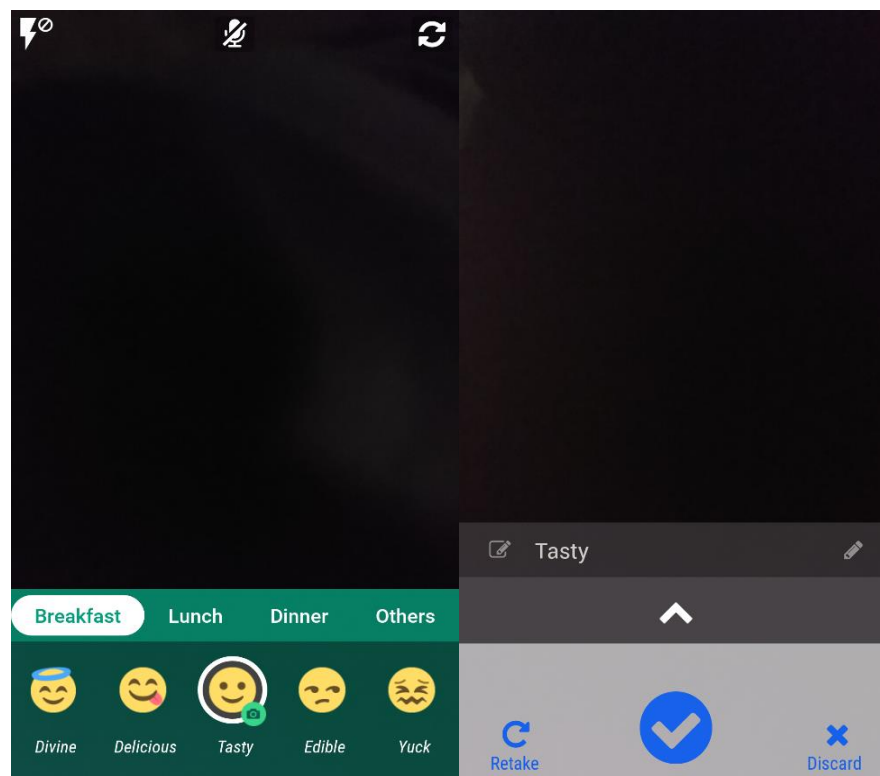
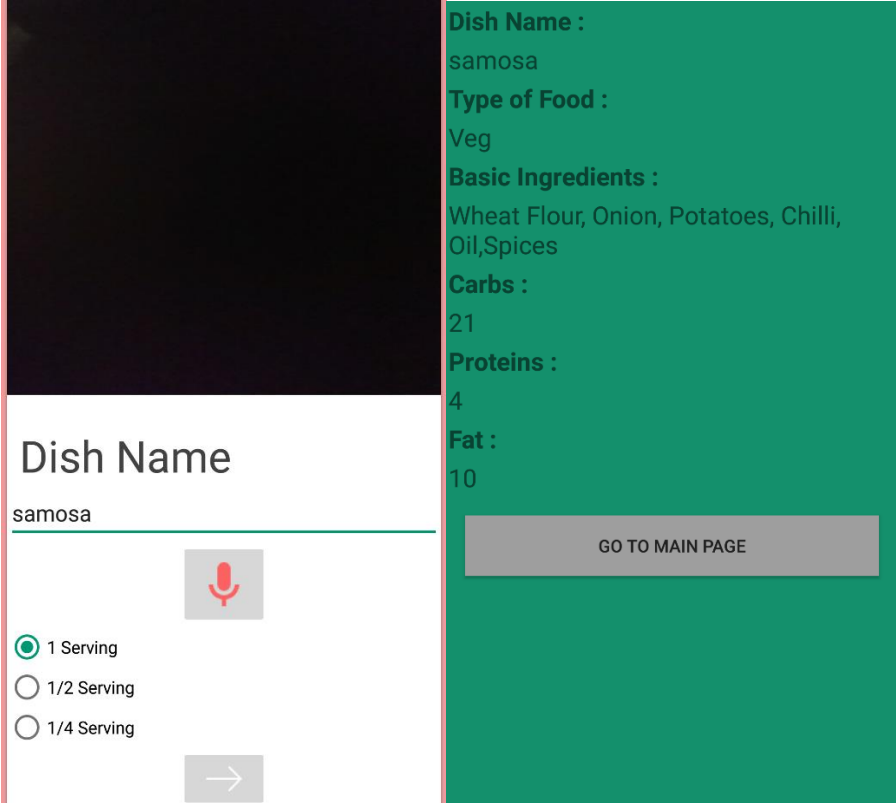


Figure 9: Image Reporting

5.3.3 Speech Input and Feature Extraction

The next activity asks user for input to annotate the food item captured. This is done by using Google speech to text API. His input is then sent to a food feature database to extract various features of the food item and display it to the user.



The screenshot displays the SpoonSnap app interface. On the left, a dark image placeholder is at the top. Below it, the text 'Dish Name' is followed by the input 'samosa'. A microphone icon is to the right of the input. Below the input, three radio buttons are shown: '1 Serving' (selected), '1/2 Serving', and '1/4 Serving'. A right arrow button is at the bottom. On the right, a green panel displays the extracted features: 'Dish Name : samosa', 'Type of Food : Veg', 'Basic Ingredients : Wheat Flour, Onion, Potatoes, Chilli, Oil, Spices', 'Carbs : 21', 'Proteins : 4', and 'Fat : 10'. A 'GO TO MAIN PAGE' button is at the bottom of the green panel.

Feature	Value
Dish Name	samosa
Type of Food	Veg
Basic Ingredients	Wheat Flour, Onion, Potatoes, Chilli, Oil, Spices
Carbs	21
Proteins	4
Fat	10

Figure 10: Speech Input and Feature Extraction

5.3.4 Time Separated Logger

The data from the previous activity and user's inputs are sent to a database where they are stored and analysed. After this a log is created for the user which includes information like food image, location, dish name and an id. This information is displayed to the user in a dynamic card view where the size of card view depends upon the size of image. Also the logger is divided into three sections namely Day, Week and Last 5 uploads which are controlled using an advanced navigation drawer.

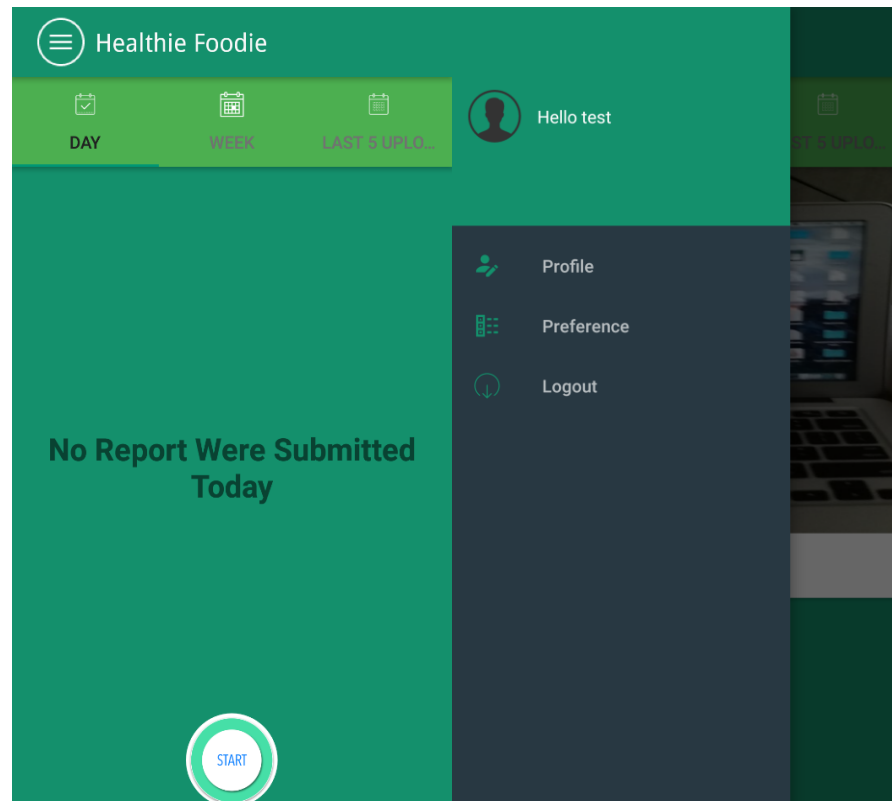


Figure 11: Logger with navigation drawer

5.4 Updated RTM

ID	NAME	Requirement	IMPLEMENTATION	TESTING
F1	User Login	User must be able to Login and register through the application		
F2	Intent Capture	Should be able to use Mobile camera for image capture		
F3	Image Upload to server	Image captured should be sent to the server on the cloud		
F4	SQS listener	Image when uploaded to AWS SQS the report should be removed from the queue and inserted to our central database		
F5	Clarifai Prediction	Given the image as input, predict the probability of the cuisines		
F6	Feature Extraction	Given Dish Name Extract the Features and approximate calories for the same		
F7	Dashboard	It should have all the previous logs of the user filtered based on time		

6. Design

Our entire food framework is divided into following parts:

1. Indian Food Data Scrapper
2. The Trainer
3. User-Interface - Mobile Application
4. Logger Server
5. Food Recognition Module

6.1. Indian Food Data Scrapper

As there were no database available for Indian Food Dish. We have to build a scraper which could scrape the data from the Internet. We used JavaScript and DownThemAll Tool for Scraping from different websites like Instagram, Facebook, many more social networking sites and Different search engines. We were able successfully scraper 106 most photograph Indian food dishes. After lot of cleaning the Database has 106 Indian Dishes with each dish having 700-1100 images each.

6.1.1 Indian Food Dataset (ASP 106)

After cleaning the scraped data, there are totally 106 Indian Food Dishes which we used for training our clarifai model. The Dataset Directory had 106 folders, with each folder representing Food dish name. Each Dish Directory had on an average of 700-1100 image named with unique id. Below Figure shows a screenshot of the image dataset.

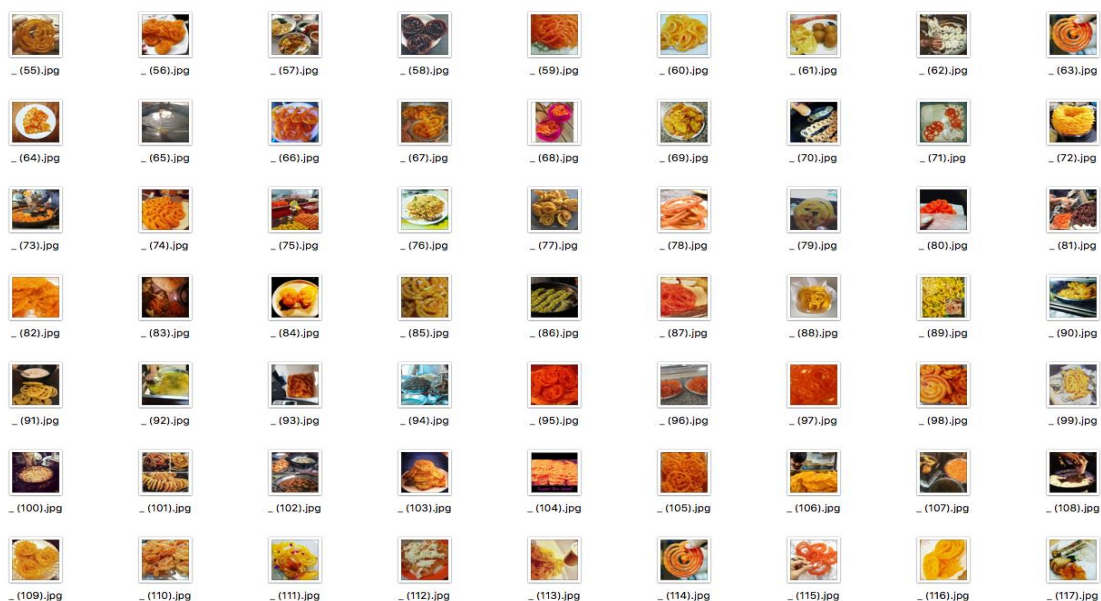


Figure 12: ASP 106 Images

We have created a visualization of the food dishes using Neo4j for getting an overview of the dishes and their categories. Neo4j is a graph database management system with native graph storage and processing. Below Figure shows the screenshot of the visualization.

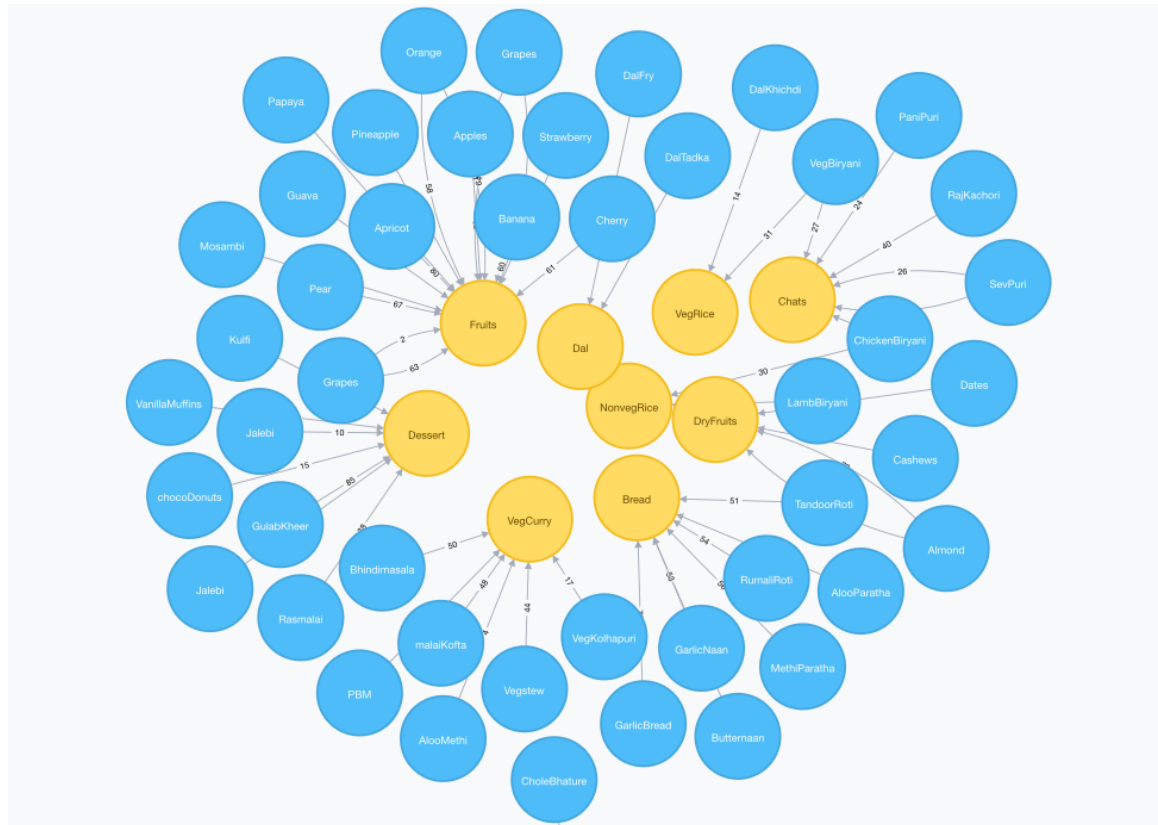


Figure 13: Neo4j visualisation of ASP 106 dataset

6.2 The Trainer

The Trainer module performs the following tasks not in order:

- Clean the dataset
- Organizing the data into classes/folders
- Pre-processing the Label data
- Accurately mapping the output classes to labels
- Converting image data into base64 strings for uploading into our cloud
- Batch training

The depth of the neural net allows it to construct a feature hierarchy of increasing abstraction, with each subsequent layer acting as a filter for more and more complex features that combine those of the previous layer. This feature hierarchy and the filters

which model significance in the data, is created automatically when deep nets learn to reconstruct unsupervised data. Because they can work with unsupervised data, which constitutes the majority of data in the world, deep nets can become more accurate than traditional Machine Learning algorithms that are unable to handle unsupervised data.

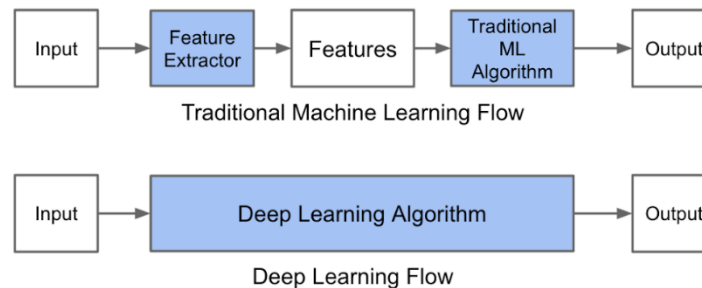


Figure 14: Deep learning vs. Machine Learning: Feature Engineering

Deep learning maps inputs to outputs. It finds correlations. It is known as a “universal approximator”, because it can learn to approximate the function $f(x) = y$ between any input x and any output y , assuming they are related through correlation or causation at all.

Therefore the Trainer Module inherently handles the task of feature engineering which would otherwise be a rather tedious task considering the size and complexity of our image dataset.

6.3 User-Interface – Mobile Application

The UI is implemented in android and uses the advanced material design features like dynamic card view, multiple tab navigation drawer and app bar. The details of UI part of the application is described in section 5.4.

6.4 Logger Server

Server of the logger was hosted on cloud Based web hosting service provider. We used phpMyAdmin for hosting our SQL Database online. PhpMyAdmin is a free and open source tool written in PHP intended to handle the administration of MySQL. Our entire server-side code is written in PHP and Node.js. The Node.js environment was setup on the CCBD Server and is running as a daemon process to handle continuous requests.

Since we Log the food habits of the person it's important for us to know who is the user and few of his personal details. Hence to store the details of the user we have created a table called User in our database which stores the personal data of the user. Our Database

has three tables one is user which stores the details of the user, Second Micro-Report table which is used to store the data which is been logged by the user, Third Main-table which is used to store the features of the dish. The schema for the tables is attached in the Figure below.

<p>u324452936_db.microreports</p> <ul style="list-style-type: none"> id : int(11) messageid : varchar(50) dish_id : int(11) report_time : datetime latitude : varchar(20) longitude : varchar(20) restaurant_name : varchar(50) revgeo_full : varchar(200) food_timing : varchar(10) tastiness : varchar(10) rating : int(11) username : varchar(100) first_name : varchar(30) imgurl : varchar(150) weather : varchar(50) North_Indian : decimal(10,8) South_Indian : decimal(10,8) Chinese : decimal(10,8) Mughlai : decimal(10,8) Fast_Food : decimal(10,8) Desserts : decimal(10,8) Continental : decimal(10,8) East_Indian : decimal(10,8) West_Indian : decimal(10,8) serving : decimal(10,0) Dish_Name : varchar(50) 	<p>u324452936_db.users</p> <ul style="list-style-type: none"> id : int(11) unique_id : varchar(23) name : varchar(50) email : varchar(100) gender : text age : int(3) encrypted_password : varchar(80) salt : varchar(10) created_at : datetime updated_at : datetime 	<p>u324452936_db.main_table</p> <ul style="list-style-type: none"> Classes : varchar(255) Location : int(11) Time : int(11) Veg/Non Veg : varchar(30) Basic Ingredients : text Calorie Carb(g) : float Proteins : float Fat : float Taste : varchar(50) Meal : varchar(50)
---	---	--

Figure 15: Database Schema

Once User Snaps a Food item, the photo is sent to Amazon Simple Queue Service (Amazon SQS), which is a distributed message queuing service. So every report which is captured will be sent to the queue. Then the server script will pick the report, process the report by sending imageURL to our clarifai image classifier model and get the concepts, then all the Data is sent to the Micro report Table in our Database. Once the micro report is successfully submitted Dish name, an activity is opened where user is asked to input the dish name and serving. Here another server call is made where features of that dish is extracted from table and based on serving, approximate calories are also given to the user. Once we have the dish name and serving we update the micro report of the user and send it to the logger.

6.5 Food Recognition Module

Our reason behind choosing Clarifai for performing our Food Recognition is based on multiple reasons. Clarifai was proven to be one of the foremost Deep Learning experts in the field in terms of food image analysis. Their deep convolutional network has outperformed the best machine learning models in the ImageNet challenge [1]. By using their platform, we are now able to store our model on the cloud. We will be able to communicate with it using simple protocols such as HTTP (REST API) and retrieve outputs at negligible time-delays, thus greatly enhancing the user experience. Our whole Food Recognition Model can be clearly divided into 3 sub modules:

6.5.1 Input Connector

This connector is an interface for our Deep Learning model to our Logger. We use JavaScript to design the connection by using the required credentials offered to us. The credentials required are a Client ID and a Client Secret which are unique to the developer. The credentials are required for each call made to the Food Recognition Module. The Trainer module uses this port initially for training our model using our ASP 106 Image Dataset.

6.5.2 Deep Learning Model

We first pre-process the data collected using the scraper and store it in a local repository. The image data needs to associate with the correct annotations and tags for the Food Recognition Module to learn properly. For this, further data processing was done where the images needed to be mapped to the exact tag-vectors for the categories we use for learning.

We adopt the architecture described by Matthew Zeiler in his ECCV 2014 paper on Convolutional Networks [1] for our deep learning model.

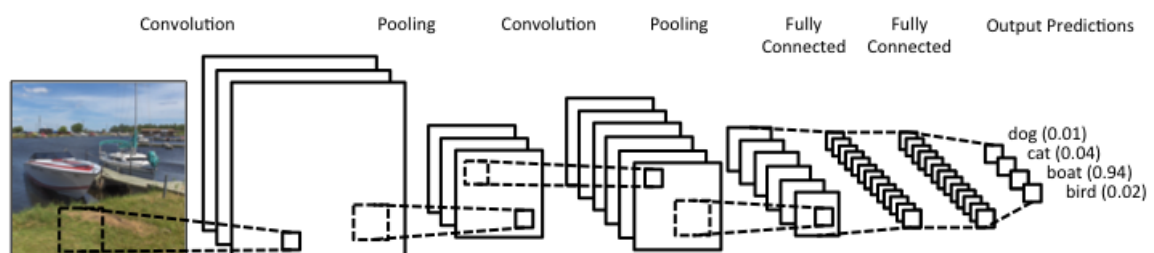


Figure 16: AI Model Overview

They use a deconvnet to visualize the activations which cause different features to get picked up by the network. While visualization of a trained model gives insight into its operation, it can also assist with selecting good architectures in the first place. By visualizing the first and second layers of Krizhevsky et al.'s architecture, various problems are apparent. The first layer filters are a mix of extremely high and low frequency information, with little coverage of the mid frequencies. Additionally, the 2nd layer visualization shows aliasing artefacts caused by the large stride 4 used in the 1st layer convolutions. To remedy these problems, (i) the 1st layer filter size was reduced from 11x11 to 7x7 and (ii) the stride of the convolution was made 2, rather than 4. This new architecture retains much more information in the 1st and 2nd layer features, as shown in Fig. 5(b) & (d) in [1].

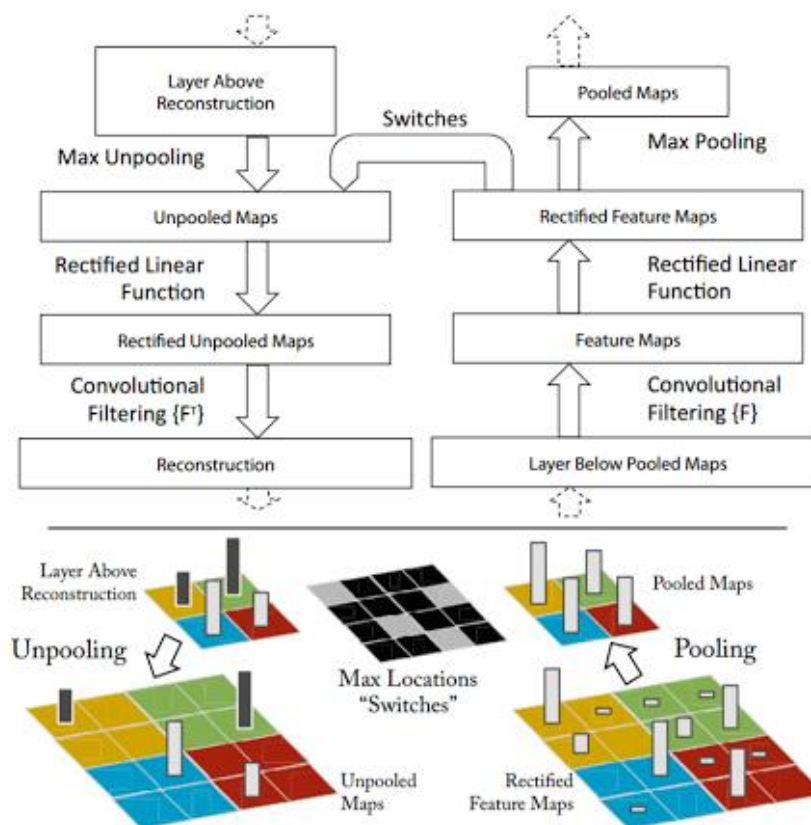


Figure 17: Clarifai's Deconvnet Architecture for enhancing feature engineering

Our model has multiple output layers unlike a traditional Neural Network:

- *The Primary Output Layer:* This layer outputs the main classification. It takes a dish image as input and outputs a Concept Vector. This vector consists of the different confidence values for each of our categories. Each output category in the Primary Layer corresponds to a restaurant class used by restaurant search engines.

The confidence values are probabilities which state the probability of the foodie's dish belonging to one of the following Food Categories:

- North Indian
- South Indian
- Chinese
- Mughalai
- Fast Food
- Desserts
- Continental
- East Indian
- West Indian

These categories were chosen because while performing domain research, all major metropolitan cities in India had the maximum number of restaurant listings for the above 9 categories. Other categories possessed miniscule numbers of listings.

- *The Generic Food Output Layer:* This layer behaves like the output layer of a generic deep learning model which has been trained on food items all over the world.
- *The Auxiliary Output Layer:* This layer behaves like the output layer of a generic deep learning model which has been trained on all kinds of things from our world. We plan to use this layer for spam filtering in the future.

6.5.3 Output Connector

This connector is an interface to the front-end endpoint. It encodes and transmits Concept Vectors from either of the 3 layers into the user interface.

6.6 Updated RTM

ID	NAME	Requirement	IMPLEMENTATION	TESTING
F1	User Login	User must be able to Login and register through the application		
F2	Intent Capture	Should be able to use Mobile camera for image capture		
F3	Image Upload to server	Image captured should be sent to the server on the cloud		
F4	SQS listener	Image when uploaded to AWS SQS the report should be removed from the queue and inserted to our central database		
F5	Clarifai Prediction	Given the image as input, predict the probability of the cuisines		
F6	Feature Extraction	Given Dish Name Extract the Features and approximate calories for the same		
F7	Dashboard	It should have all the previous logs of the user filtered based on time		

7. Implementation

7.1 Pseudo Code / Algorithms

7.1.1 Android Front-end

The application has an android front-end, the first activity uses the login feature along with a PHP script in the background to validate the credentials for the user. The similar script is used for registering the new user.

This is followed by the micro-report capturing phase which allows user to take a photograph of the image and creates a mediaJSON file which has other details like location, username, food_timing, user_ratings attached to it. The snippet of the mediaJSON file is attached below.



Figure 18: mediaJSON structure

As shown there are different sub-categories to get maximum attributes from the picture. When sub-category in the mediaJSON gives the time when the report is captured, where

gives the location of the user which is used to get the restaurant name, why gives the intent of the user and also the rating he/she gives to the food item. This file is later stored on amazon SQS and is retrieved by our scripts.

7.1.2 Food Recognition Module

Our food recognition model is built and trained on Clarifai cloud platform and follows the following algorithm:

For training images from Dish number K onwards

- **Step 1:** Organize all images of dish_no > K into the correct folder structure
- **Step 2:** Clean folders of unnecessary data like gifs, html files etc.
- **Step 3:** Allocate batch size to 128
- **Step 4:** Edit Tempsheet.csv so that the second row is dish_no = K+1 till dish_no =N for the label_vectors and preprocess all label_vectors to interpolate with categorical 0s and 1s.
- **Step 5:** Generate array of arrays containing < Dish_no, Dish_name, Label_Vector>.
- **Step 6:** Traverse directory structure
 - Create separate object called dish for Dish class
 - Attach Label_Vector to dish->concepts
 - Encode all images in the dish folder to base64 strings
 - Create and attach array of base64 strings to dish
 - Write dish object to json file for temporary storage
- **Step 7:** Initialize ClarifaiClient using ClientID and ClientSecret
- **Step 8:** Iterate through all dish json files
 - For each dish:
 - Generate model_input object

```

model_input= {
                                base64:element.imgarray[dish_no],
                                concepts:label_vector,
                                }
          
```

 - Push model_input object to model_input_list array
 - Create ClarifaiClient.model.inputs using model_input_list
- **Step 9:** For callback fired on Step 8, train current model.

For testing an image on the custom Model

- **Step 1:** Obtain image_url from Automation module.
- **Step 2:** Model_Name:= SpoonSnap
- **Step 3:** Test model via image_url method
- **Step 4:** Log the response object and return to the automation module

7.1.3 Automation Script and Database

The micro-report described in section 7.1.1 is sent to an amazon SQS queue. *Amazon Simple Queue Service (Amazon SQS)* is a distributed message queuing service introduced by Amazon.com in late 2004. It supports programmatic sending of messages via web service applications as a way to communicate over the Internet. SQS is intended to provide a highly scalable hosted message queue that resolves issues arising from the common producer-consumer problem or connectivity between producer and consumer.

From SQS the micro-report to our local database hosted on hostinger. The pseudo code for this automation is shown below:

For connecting to SQS

- **Step 1:** Initialise Secret passkey, code and region in a constructor.
- **Step 2:** Initialise parameters like *waittimesecods*, *numberofmessages*.
- **Step 3:** Enable long polling by setting *ReceiveMessageWaitTimeSeconds* on the *SetQueueAttributesRequest* before calling the Amazon SQS class's *setQueueAttribute* method.
- **Step 4:** Use *ReceiveMessageRequest* to get the mediaJSON file to a variable.

For sending the report to our database

- **Step 1:** Parse the JSON received in the previous step.
- **Step 2:** Scrape the necessary attributes like *latitude*, *username* from this JSON response.
- **Step 3:** Create a new JSON object using all the attributes.
- **Step 4:** Use node.js's *request* function to send the JSON object created in step 3 to a PHP script hosted remotely.
- **Step 5:** Use the *file_get_contents* function to get the JSON file in the receiving PHP script.
- **Step 6:** Connect to the correct database using username and password.
- **Step 7:** Use *\$input* function with appropriate database fields to insert and update in the database.
- **Step 8:** Close the connection.

7.2 Codebase Structure

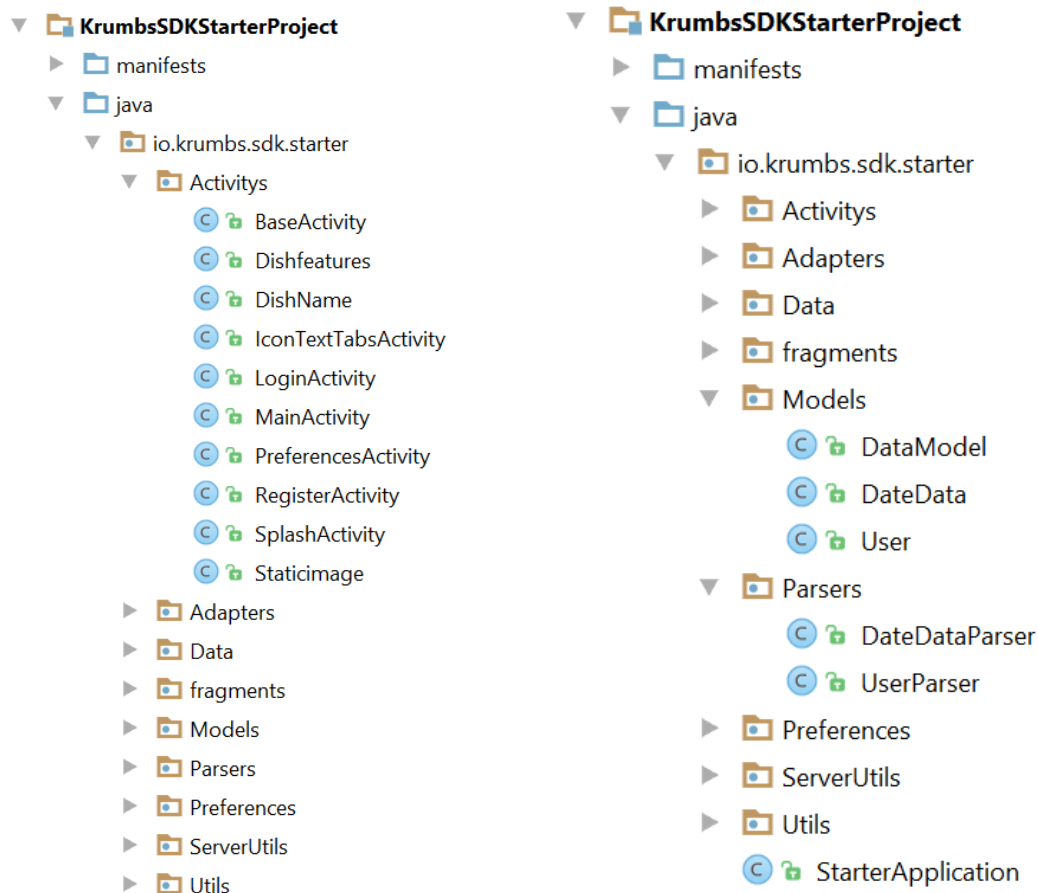


Figure 19: Java Codebase Structure

The java codebase structure of the android application is shown in figure 19. The activity folder contains all the java activities, *MainActivity* contains the cardview, navigation drawer, app bar views and connections among them.

LoginActivity, *RegisterActivity* and *SplashActivity* are used for creating user login form, registration form and attaching splash screen in the application respectively.

Dishfeatures and *DishName* activities are used for extracting the dish features from the database and display them in a proper layout in a new activity.

IconTextTabsActivity creating different fragments in the navigation drawer naming day, week and last 5 activities of the user.

The *parsers* directory uses scripts to parse response coming from the server, *DateDataParser* is used to parse the response containing dates and *UserParser* parses the user data.

Fragments, *Data* and *Adapter* directories are used to fragment and adapter extensions respectively.

Utils directory contains the *StarterApplication* class which is used to run the micro-report capture activity.

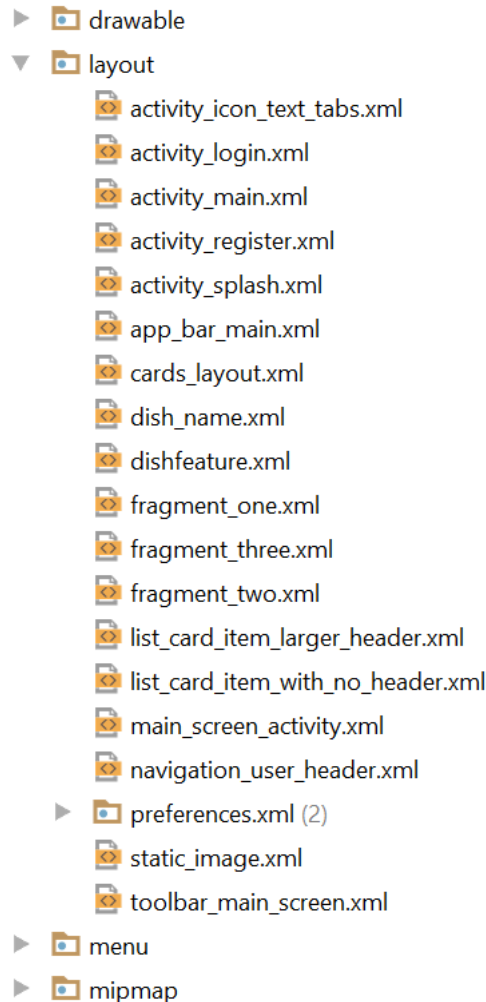


Figure 20: XML Codebase Structure

The subsequent XML codebase structure for the application is shown in Figure 20. The *layout* directory contains all the XML files/layouts used in the application. The *drawable* directory has all the images used in the application.

The *activity_login* and *activity_register* files are used to create layout for user login and register respectively.

Fragment_one, *fragment_two* and *fragment_three* layouts are responsible for creating the screen for navigation drawers and enable side-scrolling amongst them.

Main_screen_activity is the dashboard screen for the logger and shows the card view of reports submitted by the user.

The *preferences.xml* directory contains the layout for the left navigation drawer which has fields namely preferences, logout.

The *menu* directory contains various menu layouts used in the application.

7.3 Coding Guidelines Used

7.3.1 Code Indentation

1. Tabs were used for indentation. The standard java coding guideline allows a tab width of 4 spaces. We use the same in all the sections of our code.
2. In the case that the line overflows, vertical alignment of components was ensured.
3. Implicit vertical alignment is favoured over hanging indentation unless further indentation is required to make segments distinguishable.
4. Node.js follows tab length of 2 spaces, which was followed in all the sections.

7.3.2 Lines

1. Lines must be a maximum of 80 characters in length to avoid scrolling even if the project needs to be edited with the editor opened split screen alongside several windows.
2. Backslashes ('\') are to be used to indicate wrapping of a line greater than 80 characters only when implicit wrapping (line continuation inside parentheses, brackets and braces) is not applicable.
3. In multiple places, blank lines are included to improve code readability in accordance with the following rules :
 - a. Top-level function and class-definitions are surrounded with two blank lines.
 - b. Inside a class, method definitions are surrounded by a single blank line.
 - c. Additional blank lines are used to separate groups of related functions.
 - d. Blank lines are used at times in functions to indicate logical sections.

7.3.3 Imports

1. Imports should be on separate lines on top of the entire code. Android studio allows us to encapsulate all the import statements under one segment.
2. Multiple modules can be imported from the same package in a single line, but multiple packages should not be imported on a single line.

3. Imports should be grouped in the following order with a blank line between each group
 - a. Standard Library imports
 - b. Related third party imports
 - c. Local application/library specific imports
 - d. Absolute imports are recommended, as they are usually more readable and tend to be better behaved
4. Wildcard imports should be avoided.

7.3.4 String Quotes

1. Double-Quotes (“”) are used for strings.
2. Backslashes (\) are avoided to improve readability.
3. Node.js advises to use single-quotes unless we are dealing with JSON.

7.3.5 Whitespaces in expressions and statements

1. Avoid whitespaces in the following cases
 - a. Immediately inside braces, brackets or parentheses
 - b. Immediately before a comma, colon or semicolon
 - c. Between a trailing comma and a following close parentheses
 - d. More than one space around an assignment operator in order to align it with another
2. Binary operators are surrounded with a single space on either side
3. If operators with different priorities are used, consider adding whitespace around the operators with the lowest priority

7.4 Code Snippets

7.4.1 StarterApplication

```
KMediaUploadListener kMediaUploadListener = new KMediaUploadListener() {
    // onMediaUpload listens to various status of media upload to the cloud.
    @Override
    public void onMediaUpload(String id, KMediaUploadListener.MediaUploadStatus
mediaUploadStatus,
                               Media.MediaType mediaType, URL mediaUrl) {
        Log.e("url", String.valueOf(mediaUrl));

        if (mediaUploadStatus != null) {
            Intent intent1 = new Intent(StarterApplication.this,
Staticimage.class);
            intent1.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            startActivity(intent1);
            Log.i("KRUMBS Status", mediaUploadStatus.toString());
            if (mediaUploadStatus ==
KMediaUploadListener.MediaUploadStatus.UPLOAD_SUCCESS) {
                if (mediaType != null && mediaUrl != null) {
                    Log.i("KRUMBS Media, Type: ", mediaType + ": ID:" + id + ",
URL:" + mediaUrl);
                    Toast.makeText(StarterApplication.this,
mediaUploadStatus.toString(), Toast.LENGTH_LONG).show();
                    Intent intent = new Intent(StarterApplication.this,
DishName.class);
                    intent.putExtra("imageUrl", mediaUrl.toString());
                    intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                    startActivity(intent);
                }
            }
        }
    }
};
```

7.4.2 Fetching Daily Uploads of a User

```
private void getDayData() {
    new AsyncTask<Void, Void, JSONArray>(){

        DialogUtils dialogUtils = new
DialogUtils(getActivity(),DialogUtils.Type.PROGRESS_DIALOG);
        @Override
        protected void onPreExecute() {
            super.onPreExecute();
            dialogUtils.showProgressDialog("Loading Data","please wait...",false);
        }

        @Override
        protected JSONArray doInBackground(Void... params) {
            return new ServerRequest().loadDayData(getContext(),getTodaysData());
        }

        @Override
        protected void onPostExecute(JSONArray response) {
            super.onPostExecute(response);
            dialogUtils.dismissProgressDialog();
            if (response != null) {
                data = new ArrayList<DataModel>();
                try {
                    for (int i = 0; i < response.length(); i++) {
                        JSONObject object = response.getJSONObject(i);
                        data.add(new DataModel(
                            object.getString("restaurant_name"),
                            object.getString("id"),
                            object.getString("tastiness"),
                            object.getString("imgurl")
                        ));
                    }
                    adapter = new CustomAdapterRecyclerView(getContext(), data);
                    recyclerView.setHasFixedSize(true);
                    layoutManager = new LinearLayoutManager(getContext());
                    recyclerView.setLayoutManager(layoutManager);
                    recyclerView.setItemAnimator(new DefaultItemAnimator());
                    recyclerView.setAdapter(adapter);
                    adapter.notifyDataSetChanged();
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            } else {
                set_day.setText("No Report Were Submitted Today");
                Log.e("DateData","FAILED");
            }
        }
    }.execute();
}
```

7.4.3 Extracting Dish Features

```
private void ExtractFeature(final String dishname) {
    new AsyncTask<Void, Void, JSONObject>() {
        DialogUtils progressDialog = new DialogUtils(Dishfeatures.this,
DialogUtils.Type.PROGRESS_DIALOG);

        @Override
        protected void onPreExecute() {
            super.onPreExecute();
            progressDialog.showProgressDialog("Extracting Features", "please
wait..", false);
            //progress
        }

        @Override
        protected JSONObject doInBackground(Void... params) {
            return new ServerRequest().ExtractDishFeatures(Dishfeatures.this,
dishname);
        }

        @Override
        protected void onPostExecute(JSONObject response) {
            super.onPostExecute(response);
            progressDialog.dismissProgressDialog();
            if (response != null) {
                //Dish_Features_text.setText(response.toString());
                try {

Feature_Dish_name.setText(response.getString("Classes").toString());
                Feature_Type.setText(response.getString("Veg/Non Veg"));
                Feature_Ingredients.setText(response.getString("Basic
Ingredients").toString());
                Feature_carbs.setText(response.getString("Calorie
Carb(g)").toString());

Feature_proteins.setText(response.getString("Proteins").toString());
                Feature_Fat.setText(response.getString("Fat").toString());

                } catch (JSONException e) {
                    e.printStackTrace();
                }

                //Toast.makeText(Dishfeatures.this, "Hello" + response.toString(),
Toast.LENGTH_LONG).show();
            } else {
                Toast.makeText(Dishfeatures.this, "oops!..something went wrong",
Toast.LENGTH_LONG).show();
            }
        }
    }.execute();
}
```

7.4.4 Login Preference Function

```
public LoginPreferences(Context context){
    preferences = context.getSharedPreferences(USERTABLE, context.MODE_PRIVATE);
    this.context = context;
}

public void loginUser(User user){
    SharedPreferences.Editor editor = preferences.edit();
    editor.putString(USERID,user.getUid());
    editor.putString(USERNAME,user.getUsername());
    editor.putString(EMAIL,user.getEmail());
    editor.putInt(AGE, user.getAge());
    editor.putString(GENDER, user.getGender());
    editor.putBoolean(IS_LOGGED_IN,true);
    editor.apply();
}

public void logoutUser(){
    if(preferences.getBoolean(IS_LOGGED_IN,false)){
        SharedPreferences.Editor editor = preferences.edit();
        editor.clear();
        editor.apply();
    }
}

public User getUser() {
    String userId = preferences.getString(USERID, "");
    String Name = preferences.getString(USERNAME, "");
    String email = preferences.getString(EMAIL, "");
    int age = preferences.getInt(AGE, -1);
    String gender = preferences.getString(GENDER, "");
    return new User(userId,Name,email,gender,age);
}
```

7.4.5 Logger Dashboard Fragment Layout

```

<android.support.design.widget.AppBarLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    >

    <android.support.design.widget.TabLayout
        android:id="@+id/tabs"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:tabMode="fixed"
        app:tabGravity="fill"
        android:background="@color/color_16"
        />
</android.support.design.widget.AppBarLayout>

<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"

    >

    <android.support.v4.view.ViewPager

        android:id="@+id/viewpager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior"
        android:paddingTop="40sp"
        />

        <ImageButton
            android:id="@+id/start_report_button"
            android:layout_width="@dimen/report_button_dim"
            android:layout_height="@dimen/report_button_dim"
            android:src="@drawable/start_report_button"
            android:layout_centerHorizontal="true"
            android:layout_centerVertical="false"
            android:layout_alignParentBottom="true"
            android:layout_margin="@dimen/report_button_layout_margin"
            android:scaleType="fitCenter"
            android:background="@drawable/start_button_background" />

```

7.4.6 Connecting and Listening to Amazon SQS

```
awscred.load( {filename:"./keys/krumbs-sdk-client.credentials"}, function(err,
data) {
if (err) throw err
var sqs = new AWS.SQS({region: 'us-west-2',
    accessKeyId:data.credentials.accessKeyId,
    secretAccessKey: data.credentials.secretAccessKey
});
var testQUrl = "https://sqs.us-west-2.amazonaws.com/636414996916/io-krumbs-sdk-
mediajson_spoonsnap";
var params = {QueueUrl: testQUrl, /* required */
    MaxNumberOfMessages: 1
};
sqs.receiveMessage(params, function(err, data) {
    if (err)
console.log(err, err.stack); // an error occurred
    else {
        var messages = data.Messages;
        if(messages && messages.length > 0){
            var tempJson = JSON.parse(JSON.stringify(data));
            var second=JSON.parse(tempJson.Messages[0].Body);
            //console.log(second.media[0].media_source.default_src);
            console.log(PreviousURL);
            if(PreviousURL!=second.media[0].media_source.default_src){
PreviousURL=second.media[0].media_source.default_src;
console.log("\nhello\n");
console.log(JSON.stringify(second));
insertReportToServer(second); // successfully inserted into server
            console.log("removing messages:"+JSON.stringify(messages));
            _.each(messages, function(msg) {
var delParams = {
    QueueUrl: testQUrl,
    ReceiptHandle: msg.ReceiptHandle
};
};
};
};
```

7.4.7 Sending JSON to Remote Server

```
function requestinsert(objfinal){
console.log(objfinal);
request({
    url: "http://foodlogging.pe.hu/insert_report.php",
    method: "POST",
    json: true, // <--Very important!!!
    body: objfinal
}, function (error, response, body){
    console.log(body);
});
}
```

7.4.8 General Feature Extraction from Clarifai

```
ClarifaiClient.models.predict(Clarifai.FOOD_MODEL,  
"http://www.vegrecipesofindia.com/wp-content/uploads/2011/10/medu-vada.jpg").then(  
function(response) {  
    console.log("Vada");  
    console.log(response.outputs[0].data.concepts);  
  
},  
function(err) {  
    console.error("Bad scene bro"+err);  
    console.log(err);  
})
```

7.4.9 Food Feature Extraction from Custom Trained Model

```
app.models.predict("SpoonSnap", url).then(  
    function(response) {  
        var res = response.outputs[0].data.concepts;  
        console.log(res);  
        var result = [];  
        for(var i = 0; i < res.length; i++){  
            result[res[i].id] = res[i].value;  
        }  
  
        South_Indian=result['South Indian'];  
        Fast_Food=result['Fast Food'];  
        Mughlai=result['Mughlai'];  
        Desserts=result['Desserts'];  
        West_Indian=result['West Indian'];  
        Chinese=result['Chinese'];  
        Continental=result['Continental'];  
        North_Indian=result['North Indian'];  
        East_Indian=result['East Indian'];  
    },  
    function(err) {  
        process.stdout.write("Some Error is Detected");  
    }  
)
```


7.4.10 Storing New User Details on the Server

```
public function storeUser($name, $email, $password, $gender, $age) {
    $uuid = uniqid('', true);
    $hash = $this->hashSHA($password);
    $encrypted_password = $hash["encrypted"]; // encrypted password
    $salt = $hash["salt"]; // salt

    $stmt = $this->conn->prepare("INSERT INTO users(unique_id, name, email,
gender, age, encrypted_password, salt, created_at) VALUES(?, ?, ?, ?, ?, ?, ?,
NOW())");
    $stmt->bind_param("ssssiss", $uuid, $name, $email, $gender, $age,
$encrypted_password, $salt);
    $result = $stmt->execute();
    $stmt->close();

    // check for successful store
    if ($result) {
        $stmt = $this->conn->prepare("SELECT * FROM users WHERE email = ?");
        $stmt->bind_param("s", $email);
        $stmt->execute();

        $stmt->bind_result($id, $unique_id, $name, $email, $gender, $age,
$encrypted_password, $salt, $created_at, $updated_at);
        $stmt->fetch();
        $user =
array('id'=>$id, 'unique_id'=>$unique_id, 'name'=>$name, 'email'=>$email,
'gender'=>$gender, 'age'=>$age, 'encrypted_password'=>$encrypted_password, 'salt' =>
$salt, 'created_at' => $created_at, 'updated_at' =>$updated_at);

        //$user = $stmt->get_result()->fetch_assoc();
        $stmt->close();

        return $user;
    } else {
        return false;
    }
}
```

7.4.11 Concept Search

```
function Concept_Searcher (event){
    //console.log("Why ")
    var query=$("#concept_query").value
    ClarifaiClient.inputs.search({concept: {name:"East Indian"}}).then(
    {
        function (response)
        {
            console.log("Image Url:")
            console.log(response.hits[0].input.data.image.url)
        }
    });
}
inputbtn.on("click", ImageUploader);
```

7.4.12 Image Scraper Snippet

```

var fs = require('fs');
browser.ignoreSynchronization = true;
var outputFilename = 'Output.txt';
describe('Google Demo', function() {
  it('Should Search', function() {
    browser.driver.get('https://www.facebook.com/');

    browser.driver.findElement(by.xpath('//*[@id="email"]')).sendKeys('test123@gmail.co
m');

    browser.driver.findElement(by.xpath('//*[@id="pass"]')).sendKeys('24jhj45hj45');




    browser.driver.findElement(by.xpath('//input[contains(@value,"Log
In")]')).click();

    browser.driver.get('https://www.facebook.com/search/str/%2523samosa/photos-
keyword');
    browser.driver.sleep(2000);
    browser.driver.sleep(10000);
    //scroll
    //scrollPage();
    //var someimage =
    browser.driver.findElement(by.xpath('//div[contains(@id,"pagelet_loader_initial_bro
wse_result")]'));
    //var myimg = browser.driver.findElement(by.tagName('img'))[0];
    element.all(by.tagName('img')).each(function (element)
    {
      var linkTarget = element.getAttribute('src').then(function(attr)
      {
        var checkStringStartWithFacebook =
        attr.startsWith("https://www.facebook.com/");
        if(!checkStringStartWithFacebook){
          console.log(attr);
        }
      });
    });
  });
});

```

7.5 Unit Test Cases

The various units were tested on different types of images, i.e., dishes belonging to different categories from all regions of India. The Image classifier outputs confidence values for each of the concepts triggered on performing forward propagation on the image on the Convolutional Neural Network.

Dish Name	Image	Confidence Values Top 2	Result
Jalebi		<pre> ▼ concepts: Array(9) ▼ 0: Object app_id: "ad31d95a8ee348" id: "North Indian" name: "North Indian" value: 0.9021083 ► __proto__: Object ▼ 1: Object app_id: "ad31d95a8ee348" id: "Desserts" name: "Desserts" value: 0.85608506 </pre>	Positive
Donut		<pre> id: "Desserts" name: "Desserts" value: 0.9874079 ► __proto__: Object ▼ 1: Object app_id: "ad31d95a8ee348" id: "Continental" name: "Continental" value: 0.9560407 </pre>	Positive
Vada		<pre> ▼ concepts: Array(9) ▼ 0: Object app_id: "ad31d95a8ee34858a7" id: "Fast Food" name: "Fast Food" value: 0.52996033 ► __proto__: Object ▼ 1: Object app_id: "ad31d95a8ee34858a7" id: "South Indian" name: "South Indian" value: 0.5007343 </pre>	Positive

Gulab Jamun		<pre> id: "North Indian" name: "North Indian" value: 0.76861894 __proto__: Object 1: Object app_id: "ad31d95a8ee34858" id: "East Indian" name: "East Indian" value: 0.47648132 __proto__: Object 2: Object app_id: "ad31d95a8ee34858" id: "Desserts" name: "Desserts" value: 0.26771516 </pre>	Negative
-------------	---	--	----------

7.6 Metrics for Unit Test Cases

We used accuracy as the metrics for unit test cases. We checked each module with different types of images and fine-tuned the thresholds so that it gave acceptable results with all of them.

7.7 UPDATED RTM

ID	NAME	Requirement	IMPLEMENTATION	TESTING
F1	User Login	User must be able to Login and register through the application	The user was validated from email id and password. we used okhttp3 of server connection	
F2	Intent Capture	Should be able to use Mobile camera for image capture	Used Camera.open() to get an instance of the camera object and build custom view on camera intent	
F3	Image Upload to server	Image captured should be sent to the server on the cloud	every time user clicks a photo we sent the data to the AWS SQS	
F4	SQS listener	Image when uploaded to AWS SQS the report should be removed from the queue and inserted to our central database	On the server side we enabled the long polling of listening any new reports that were submitted. The script was run as a daemon process	

F5	Clarifai Prediction	Given the image as input, predict the probability of the cuisines	By means of Deep learning, the CNN computes the probability of the cuisines. The image is sent to an Inception-v3 CNN which has 22 layers that predicts the cuisines	
F6	Feature Extraction	Given Dish Name Extract the Features and approximate calories for the same	when we get the dish name we extract the features from the database and approximate calories for the same and send it to the user	
F7	Dashboard	It should have all the previous logs of the user filtered based on time	once report is successfully submitted we retrieve the report based on the user and show his/her log	

8. Testing

8.1 System/Function test Specifications for the Project

8.1.1 Test Dataset Distribution

We performed testing on the trained classifier using ASP-106 dataset. It included 106 classes with an average of 700 images for each class. Testing was performed using these classes with 30-40 images each.

8.1.2 Test UI

The UI was tested on various android powered smart phones with android version ranging from 21 to 26. We found that there were slight changes in the material design of the UI in different versions, but the overall functionality worked on all of them.

8.2 Test Environment

8.2.1 Software Environment

Operating system version	Microsoft Windows 10
Android Studio Version	2.3.1
Krums SDK Snapshot	2.4.0
Build Tools Version	25.0.0
Okhttp3 Version	3.5.0

Picasso Version	2.5.2
Phpmyadmin Version	3.5.2.2
Node.js Version	6.5.0
Npm Version	3.10.3
AWS SDK Version	2.0

8.2.1 Hardware Environment

RAM	16GB
Processor	Intel i7
Clock Speed	2.30 GHz
GPU	GTX 580

8.3 Test Procedure

The Image classifier tests were conducted by manually inputting images to the classifier while measuring:

- Response time
- comparing the top 2 or 3 concepts with the expected values

All tests were run by using the web framework on the browser because of the reliability of such a setup. We created a web interface for inserting images into the module, training and testing.

The classifier was tested using different datasets during the development stage and the results can be seen in [1]. For the Caltech-256 data, 15, 30, 45, or 60 training images per class were selected, reporting the average of the per-class accuracies in Table 4. Our ImageNet – pre-trained model beats the current state-of-the-art results obtained by Bo et al. [3] by a significant margin: 74.2% vs. 55.2% for 60 training images/class. However, as with the Caltech-101 dataset, the model trained from scratch does poorly. With our pre trained model, just 6 Caltech-256 training images are needed to beat the leading method using 10 times as many images.

8.4 Example Test Result

We choose a highly photographed fast food in India called “Pani Puri” and observed the confidence values for Fast Food because that was the expected category of food. Below is the triggered concepts output

Dish: Pani Puri

URL: <https://qph.ec.quoracdn.net/main-qimg-14f6bc4a3b89ae33a33107a9b56b38e7>

Pani Puri

```
▼ Object {status: Object, outputs: Array(1), rawData: Object}
  ▼ outputs: Array(1)
    ▼ 0: Object
      created_at: "2017-04-18T22:50:14.163008Z"
      data: Object
        concepts: Array(9)
          ▼ 0: Object
            app_id: "ad31d95a8ee34858a7cb23f8be75392e"
            id: "Fast Food"
            name: "Fast Food"
            value: 0.9178843
            ► __proto__: Object
          ▼ 1: Object
            app_id: "ad31d95a8ee34858a7cb23f8be75392e"
            id: "North Indian"
            name: "North Indian"
            value: 0.4171964
            ► __proto__: Object
          ► 2: Object
          ► 3: Object
```

Figure 21: Concepts Detected in the Image

8.5 Test Metrics

The accuracies of the best CNN models from the ILSVRC Imagenet challenge were compared with Clarifai's CNNs. As seen, with smaller sizes of inputs our network does not perform on par with Bo et al and Sohn et al [1].

8.5.1 Test Statistics

The accuracies of the rival networks are tabulated as below. The input data size of images is in thousands.

# Train	Acc % 15/class	Acc % 30/class	Acc % 45/class	Acc % 60/class
Sohn <i>et al.</i> [24]	35.1	42.1	45.7	47.9
Bo <i>et al.</i> [3]	40.5 ± 0.4	48.0 ± 0.2	51.9 ± 0.2	55.2 ± 0.3
Non-pretr.	9.0 ± 1.4	22.5 ± 0.7	31.2 ± 0.5	38.8 ± 1.4
ImageNet-pretr.	65.7 ± 0.2	70.6 ± 0.2	72.7 ± 0.4	74.2 ± 0.3

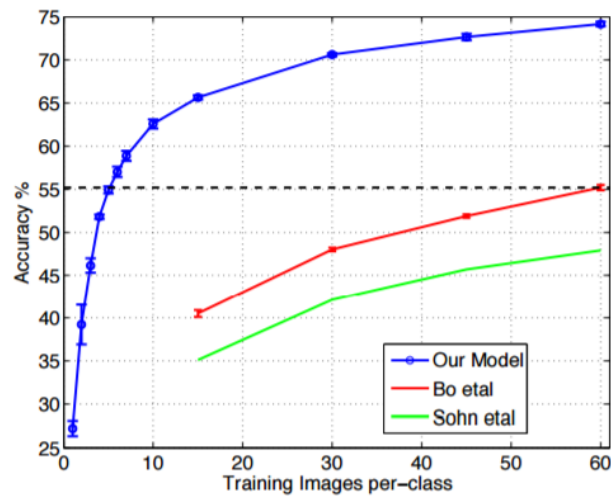


Figure 22: Caltech 256 classification accuracies

8.5.2 Effort Variance

The Histogram for Effort variance is shown below in figure 23.

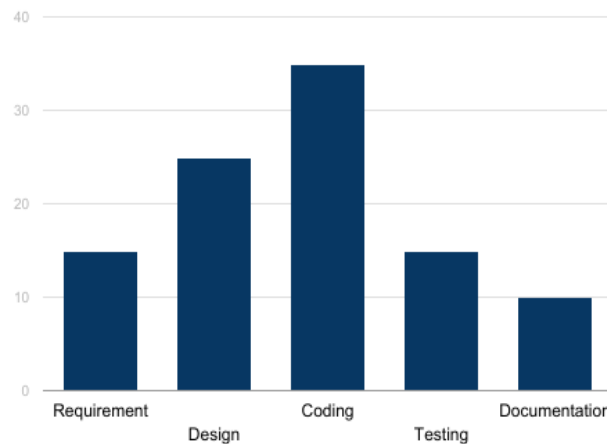


Figure 23: Effort Variance

8.6 Updated RTM

ID	NAME	Requirement	IMPLEMENTATION	TESTING
F1	User Login	User must be able to Login and register through the application	The user was validated from email id and password. we used okhttp3 of server connection	Different test case were given for testing the login functionality and passed for the same
F2	Intent Capture	Should be able to use Mobile camera for image capture	Used Camera.open() to get an instance of the camera object and build custom view on camera intent	The usability of the code was tested and checked that code doesn't break any point of the time.
F3	Image Upload to server	Image captured should be sent to the server on the cloud	every time user clicks a photo we sent the data to the AWS SQS	Multiple images were uploaded together to check scalability of the module

F4	SQS listener	Image when uploaded to AWS SQS the report should be removed from the queue and inserted to our central database	On the server side we enabled the long polling for listening any new reports that were submitted. The script was run as a daemon process	Submitted multiple reports at once and fetched them individually
F5	Clarifai Prediction	Given the image as input, predict the probability of the cuisines	By means of Deep learning, the CNN computes the probability of the cuisines. The image is sent to an Inception-v3 CNN which has 22 layers that predicts the cuisines	It was tested on clarifai's commercial graded GPUs
F6	Feature Extraction	Given Dish Name Extract the Features and approximate calories for the same	when we get the dish name we extract the features from the database and approximate calories for the same and send it to the user	By providing Different test case and handling them were taken care off
F7	Dashboard	It should have all the previous logs of the user filtered based on time	once report is successfully submitted we retrieve the report based on the user and show his/her log	Tested Dynamic card view from the data based on the User.

9. Results and Discussion

Our final system is a full end to end working food framework with an android application in the user end, amazon SQS and hostinger in the server end and clarifai custom trained model as the deep learning back-end.

The android application was tested on a number of different smart phones and the APK is released to our classmates and collaborators. The server works well with handling of multiple users and multiple requests at the same time.

We were able to work with 15-20 users being online simultaneously and our application was able to handle the traffic. Also users still upload food images on a day to day basis on the application. Android versions from Kit Kat to Nugget is supported where Nugget displaying the most advanced material design patterns.

Our CNN gives an accuracy of above 80% for the small number of test cases that we tried. We were unable to do bulk testing because of the API restrictions imposed by Clarifai on the number of operations per month. The confidence probabilities were compared with the expected categories and we obtained accurate results for the same.

With our framework we aim to start a revolution in the field of future health and food. Currently there is very little work done to help healthy foodies. We are trying to change this situation by developing technology to collect, manage, and use big food data to assist individuals manage their culinary and health experience using modern wearable and smart phones. Our application SpoonSnap is the first step toward this, we are working with people around the world to make SpoonSnap one of the unique and best applications for food logging and personalized recommendation.

10. Retrospective

10.1 What went well

For us, this project was not just any final year project but one which enabled us to develop as programmers, managers, and entrepreneurs and even exposed us to a great extent to the field of future health and lifestyle.

Since we all are fond of food and technology, it didn't take time for us to finalise on a project in this domain. We had similar interests for the application and the design. After doing some literature survey we realised that there is not much work done in the field of Indian food items or Indian food community. Since India is a diverse country, the food habits are also diverse which makes collection of data a very complicated and time taking task. We targeted this domain and created food image dataset exclusively for Indian dishes. Our dataset ASP-106 is currently the biggest Indian food image dataset, it is hosted online and is available for use.

This project was a great illustration of the importance of technology in field of food. We got to interact directly with professors and foodies around the world and discuss domain of health with them. We were exposed to huge amount of data and critical information. This opportunity helped us improve our management and entrepreneurial skills as we were handed the task of selling our idea to people from different domains and lifestyles. It was a great experience for everyone.

10.2 What did not go well

Since there was not much work done in the field of Indian food and cuisine it took us lot of time to collect data for the same. We used up lot of our time in data collection and cleaning which delayed our work in design and implementation sections. Also, Indian culture has thousands of food items and different ways of cooking, hence we decide on 106 most photographed Indian dishes and scrapped their data from various food items. We faced small problems while hosting our server scripts because there were no free hosting websites/platform. Everything else was sorted and smooth.

11 References

- [1]Zeiler M.D., Fergus R. (2014) Visualizing and Understanding Convolutional Networks. In: Fleet D., Pajdla T., Schiele B., Tuytelaars T. (eds) Computer Vision – ECCV 2014. ECCV 2014. Lecture Notes in Computer Science, vol 8689. Springer, Cham
- [2] YANG, L., HSIEH, C. K., YANG, H., POLLAK, J. P., DELL, N., BELONGIE, S., ... & ESTRIN, D. Yum-me: A Personalized Nutrient-based Meal Recommender System. LTD, H. W. (2016, February). *HealthifyMe: Mobile Weight loss Coach*. Retrieved from HealthifyMe: <https://healthifyme.com/>
- MyFitnessPal, I. (2016). *Free calorie counter, exercise calculator*. Retrieved from MyfitnessPal: <https://www.myfitnesspal.com/>
- Nutritionix. (2015, May). *Nutritionix - Largest Verified Nutrition Database*. Retrieved from Nutritionix - Largest Verified Nutrition Database: <https://www.nutritionix.com/>