INSTACHEF - SMART RECIPE GENERATOR

ABSTRACT

In an age where convenience, nutrition, and sustainability are paramount, InstaChef - Smart Recipe Generator offers a transformative solution for home cooks. This innovative application leverages advanced image recognition technology to allow users to snap pictures of the ingredients they have on hand, generating personalized recipe suggestions based on those items.

The core functionality of InstaChef involves identifying various food items from user-submitted photos. Once the ingredients are recognized, the app accesses a vast recipe database to curate tailored suggestions that minimize the need for additional grocery shopping. This feature not only enhances meal preparation but also addresses food waste, encouraging users to make the most of their existing supplies.

In addition to recipe generation, InstaChef provides detailed nutritional information, including calorie counts and macronutrient breakdowns. This empowers users to make informed dietary choices, accommodating specific preferences such as vegan, gluten-free, or low-carb diets. The app's user-friendly interface allows for easy navigation through recipes, with step-by-step instructions and the ability to save favorite dishes for future reference.

Beyond individual users, InstaChef serves families aiming to promote healthier eating habits. By making cooking more accessible and enjoyable, the app inspires users to engage with their food and understand nutrition better. This aligns with broader trends toward healthier lifestyles and sustainable practices, encouraging users to choose home-cooked meals over takeout or processed options.

In conclusion, InstaChef stands as a vital tool for modern cooks, seamlessly merging technology with culinary creativity. By empowering users to transform their ingredients into delicious meals, InstaChef is set to redefine the cooking experience, making it more accessible, informative, and enjoyable for everyone.

TABLE CONTENTS

CHAPTER NO	**TITLE**	**PAGE NO**
	ABSTRACT	i
	LIST OF TABLES	ii
	LIST OF FIGURES	iii
1.	**INTRODUCTION**	01
	1.1 Overview	01
	1.2 Problem Definition	02
2.	**SYSTEM ANALYSIS**	03
	2.1 Existing System	03
	2.2 Proposed System	04
	2.3 Development Environment	05
3.	**SYSTEM DESIGN**	06
	3.1 UML Diagrams	06
	3.2 Data Dictionary	11
	3.3 ER Diagram	13
	3.4 Data Flow Diagram	14
4.	**SYSTEM ARCHITECTURE**	17
	4.1 Architecture Overview	17
	4.2 Technologies and Related Work	18
5.	**SYSTEM IMPLEMENTATION**	20

	5.1 Data Acquisition	20
	5.2 Image Labeling	23
	5.3 Image Augmentation	23
	5.4 Model Training	24
6.	**DEPLOYMENT**	32
	Deploy on GUI (Graphical User Interface)	32
7.	**CONCLUSION**	34
	Conclusion	34
	Future Enhancements	34
8.	**APPENDIX**	35
9.	**REFERENCES**	36

LIST OF TABLES

TABLE NO	TABLE DESCRIPTION	PAGE NO
5.2.5	mAP Tabulation	28

LIST OF FIGURES

FIG NO	FIGURE DESCRIPTION	PAGE NO
3.1.1	Use Case Diagram for InstaChef - Smart Recipe Generator	06
3.1.2	Class Diagram for InstaChef - Smart Recipe Generator	07
3.1.3	Sequence Diagram for InstaChef - Smart Recipe Generator	08
3.1.4	Collaboration Diagram for InstaChef - Smart Recipe Generator	09
3.1.5	Activity Diagram for InstaChef - Smart Recipe Generator	10
3.3	ER Diagram for InstaChef - Smart Recipe Generator	13
3.4.1	Dataflow Diagram level-0	14
3.4.2	Dataflow Diagram level-1	15
3.4.3	Dataflow Diagram level-2	16
4.1	Architecture Diagram for InstaChef - Smart Recipe Generator	17
4.2.1	YoloV4 Object Detection Model	18
4.2.2	Yolo V4 MatLab and Simulink	18
5.1.1	Iteration 1: Tomato with and without bag	20
5.1.2	Iteration 2: 3 Class datasets with and without bag	21
5.1.3	Setup for Data Acquisition	21
5.1.4	Iteration 3: Classes with and without bag	22
5.2.1	Iteration 3: Labeling Interface	23
5.2.2	Various Transformation on the image	24
5.2.3	IoU (Intersection of Union)	26
5.2.4	IoU (Intersection of Union) in Real	26
5.2.5	P-R Curve	27
5.2.6	mAP vs Loss over Iteration	29
5.2.7	Earlier Model Detection vs Recent Model Detection	30
6.1.1	GUI of Fruits and Vegetable Detection POS	33

CHAPTER 1

INTRODUCTION

OVERVIEW

InstaChef - Smart Recipe Generator is a groundbreaking app designed to simplify home cooking by allowing users to snap photos of their available ingredients. Utilizing advanced image recognition technology, the app identifies food items and generates personalized recipe suggestions from a vast database. This feature not only streamlines meal preparation but also encourages users to make the most of what they already have, reducing food waste.

In addition to offering tailored recipes, InstaChef provides detailed nutritional information, including calorie counts and macronutrient breakdowns, catering to various dietary preferences such as vegan, gluten-free, or low-carb. The user-friendly interface ensures a seamless cooking experience, complete with step-by-step instructions and options to save favorite dishes.

By making cooking more engaging and accessible, InstaChef promotes healthier eating habits and inspires users to embrace their culinary creativity, redefining the way we approach meal preparation at home.

PROBLEM DEFINITION

This project centers on the development and implementation of InstaChef - Smart Recipe Generator, an application aimed at enhancing the cooking experience for users of all ages. The app addresses several key challenges in meal preparation, including:

- Ingredient Utilization: Many individuals struggle to make the most of their available ingredients, often leading to food waste. InstaChef empowers users to create meals based on what they already have.

- Nutritional Awareness: With increasing focus on health, users often seek nutritional information about their meals. The app provides detailed insights into calorie counts and macronutrient breakdowns, promoting informed dietary choices.

- Culinary Inspiration: Home cooks can sometimes feel uninspired or unsure about what to prepare. InstaChef offers personalized recipe suggestions, encouraging creativity and exploration in the kitchen.

The project encompasses the integration of image recognition technology for ingredient identification, along with data collection and visualization to enhance user engagement and understanding. By demonstrating how cooking through technology can enrich primary education and daily life, InstaChef aims to transform the culinary landscape, making cooking enjoyable and sustainable for everyone.

CHAPTER 2

SYSTEM ANALYSIS

EXISTING SYSTEM

Existing System

The current landscape of cooking applications primarily focuses on recipe databases and meal planning without addressing the specific needs of users based on their available ingredients. These existing systems typically include:

- **Recipe Search Functionality:** Users can search for recipes based on specific ingredients, but the process requires prior knowledge of what to look for.
- **Nutritional Information:** Some apps provide nutritional details, but they often lack personalization based on users' unique dietary needs or preferences.
- **User Engagement:** Many existing cooking apps fail to create an engaging user experience, leaving users feeling uninspired or disconnected from the cooking process.
- **Data Visualization:** While some platforms offer nutritional tracking, they do not visualize this data in an intuitive way, making it challenging for users to understand their eating habits.

Disadvantages:

- **Lack of Ingredient Recognition:** Most systems do not incorporate image recognition, requiring users to manually input ingredients, which can be time-consuming.
- **Limited Target Audience:** Existing applications often cater to a broad audience without focusing on specific demographics, such as families or students.
- **No Feedback Mechanism:** Many apps do not provide user feedback or generate reports that help users track their cooking habits or dietary choices over time

PROPOSED SYSTEM

The proposed system, InstaChef - Smart Recipe Generator, is a digital platform designed to transform the cooking experience for users of all ages by integrating educational content with engaging technology. The app specifically targets home cooks, allowing them to learn and explore culinary skills in an interactive manner.

Key features of the system include:

- **Ingredient Recognition:** Users can take photos of their available ingredients, which the app will analyze to generate personalized recipe suggestions.
- **Nutritional Education:** The app provides detailed nutritional information, helping users understand the health benefits of their meals and make informed dietary choices.
- **User Engagement:** By gamifying the cooking process, InstaChef encourages users to experiment with different recipes and cooking techniques, fostering a love for cooking.
- **Feedback and Reporting:** After each cooking session, the app generates a report summarizing the nutritional content and cooking habits, which users can track over time. This feature promotes self-assessment and learning.
- **Interactive Learning:** The system emphasizes critical thinking and problem-solving skills by challenging users to utilize available ingredients creatively, encouraging them to think outside the box.

Overall, InstaChef offers a more engaging and interactive learning experience compared to traditional cooking methods. By making cooking fun and educational, the app aims to enhance users' culinary skills and promote healthier eating habits, making it a valuable tool for individuals and families alike.

DEVELOPMENT ENVIRONMENT

SOFTWARE REQUIREMENT

- Windows 11
- Visual Studio

HARDWARE REQUIREMENT

- Processor: 64-bit, four-core,2 GHz minimum per core
- RAM: 4GB for execution
- Hard disk: 2GB for installation
- Proper internet Connectivity

CHAPTER 3

SYSTEM DESIGN

UML DIAGRAMS

Use Case Diagram

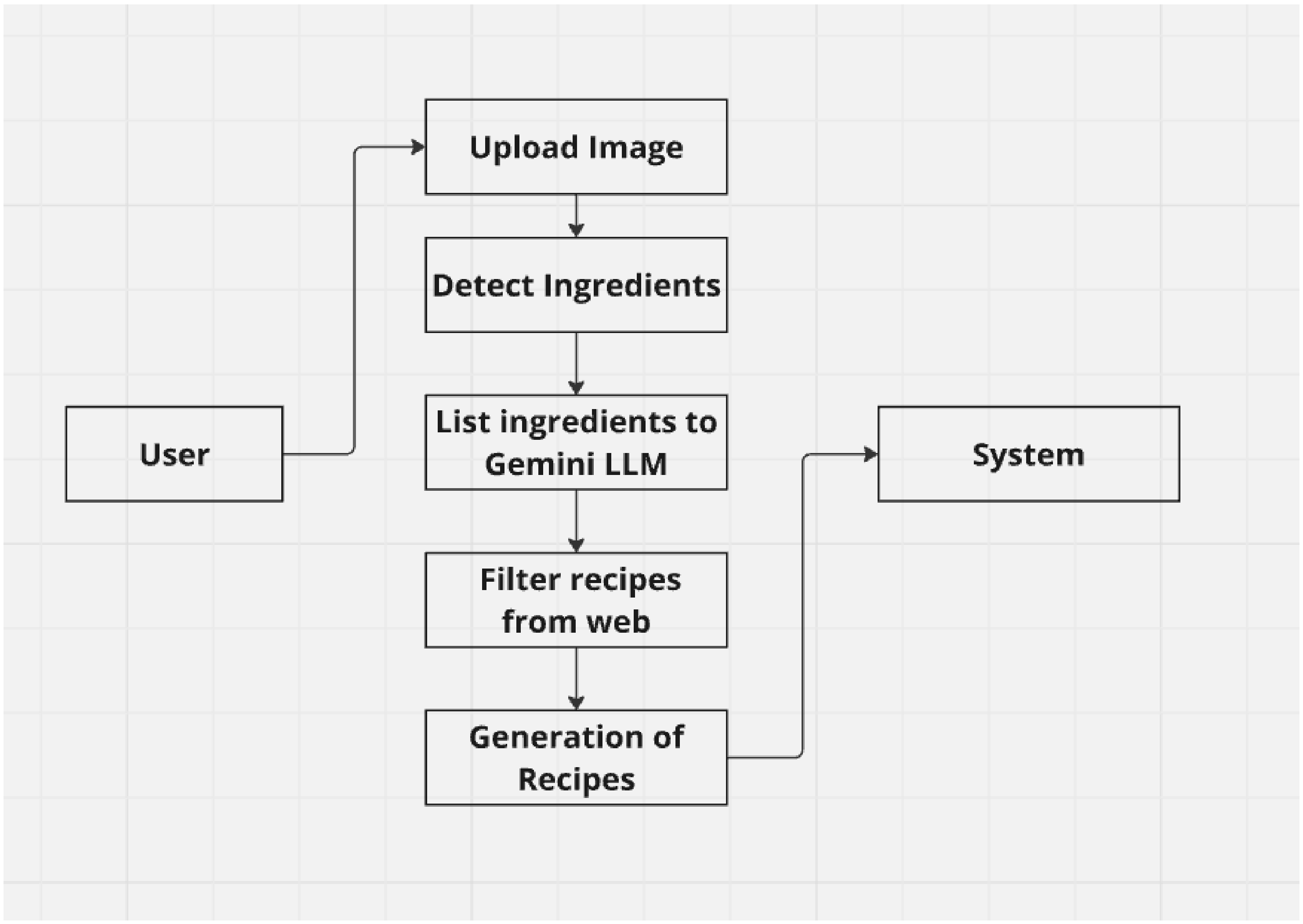


Fig 3.1.1 Use case diagram for InstaChef - Smart Recipe Generator

Class Diagram:

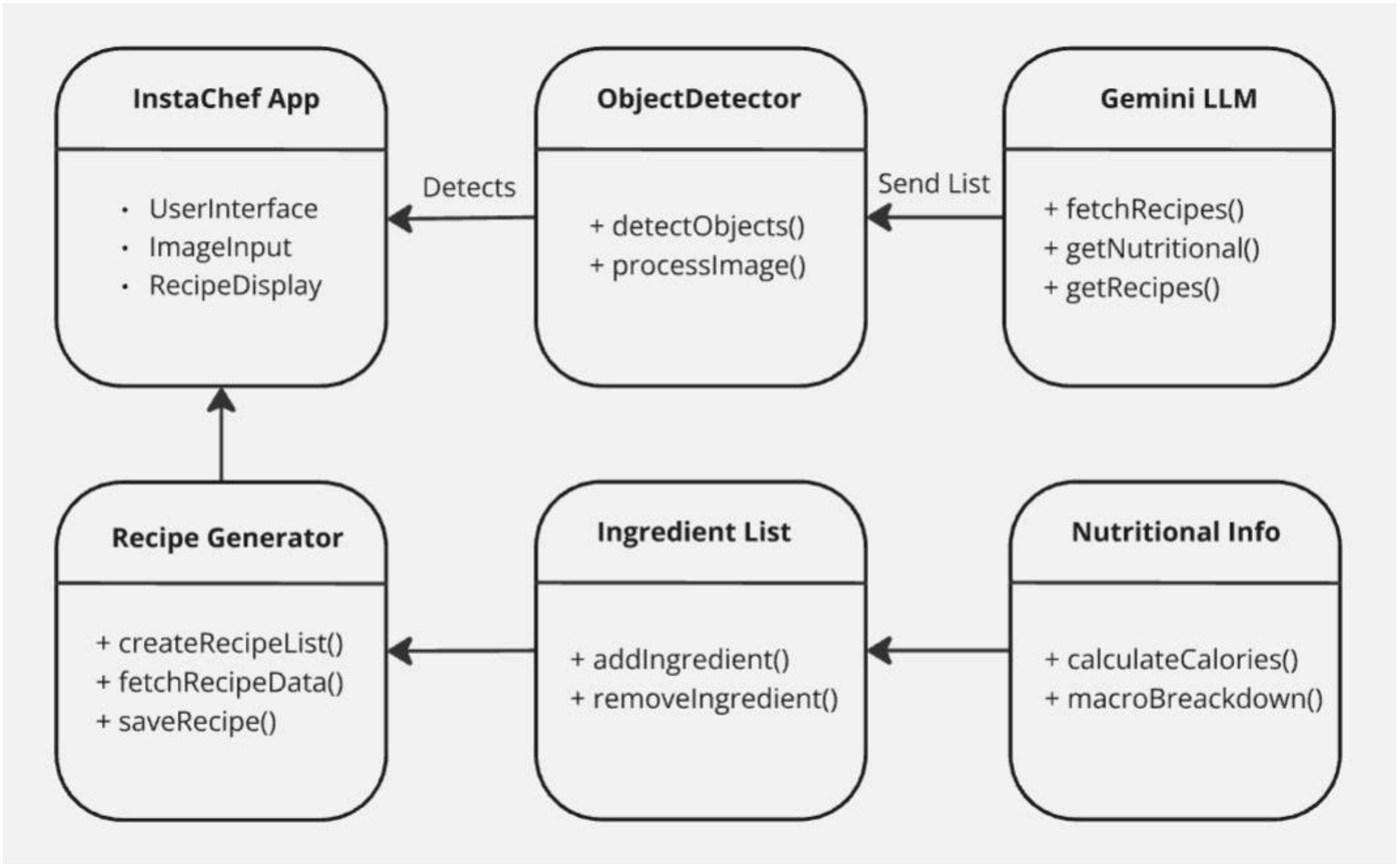


Fig 3.1.2 Class Diagram for InstaChef - Smart Recipe Generator

Sequence Diagram:

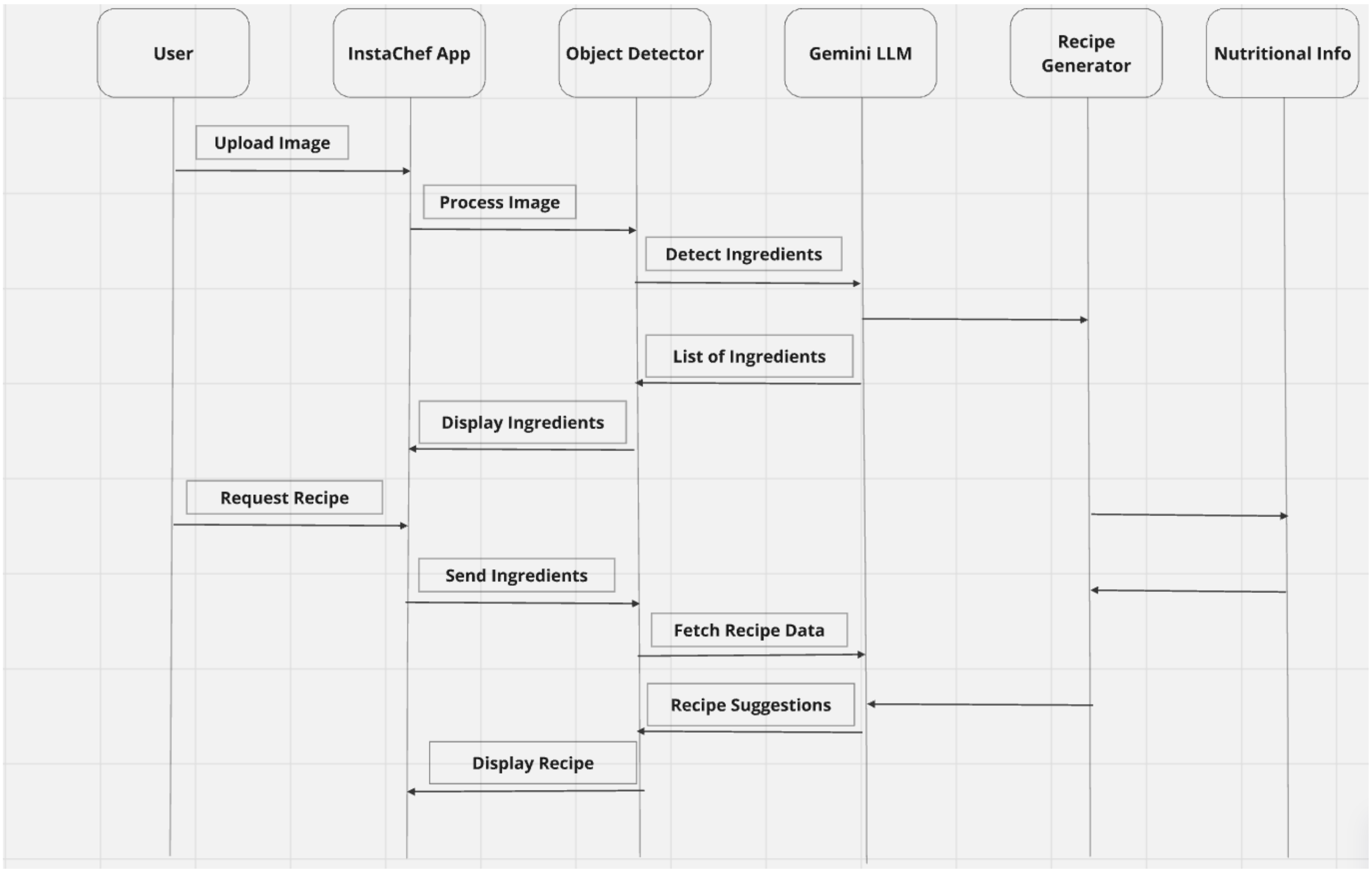


Fig 3.1.3 Sequence Diagram for InstaChef - Smart Recipe Generator

Collaboration diagram:

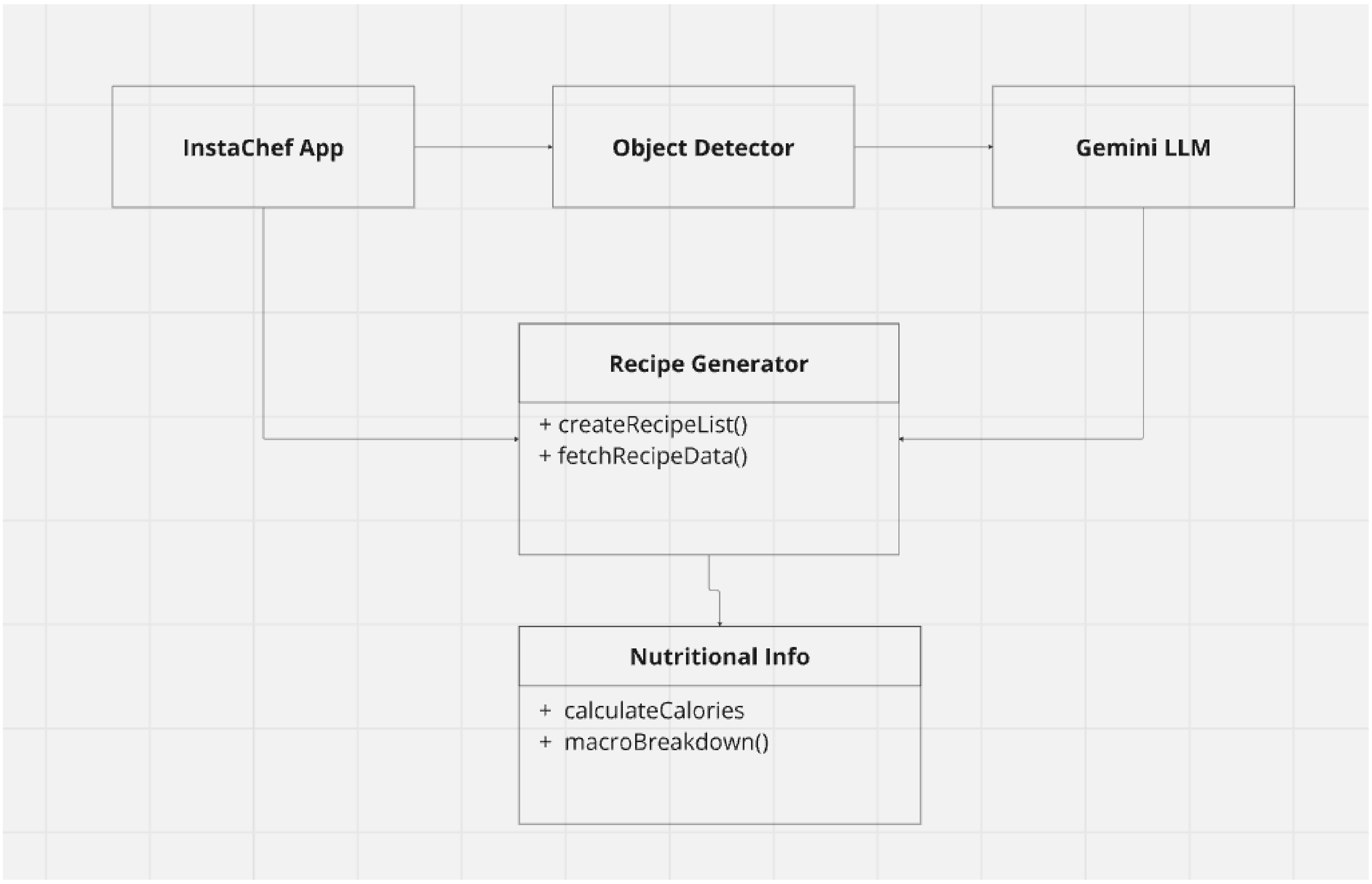


Fig 3.1.4 Collaboration diagram for InstaChef - Smart Recipe Generator

Activity Diagram :

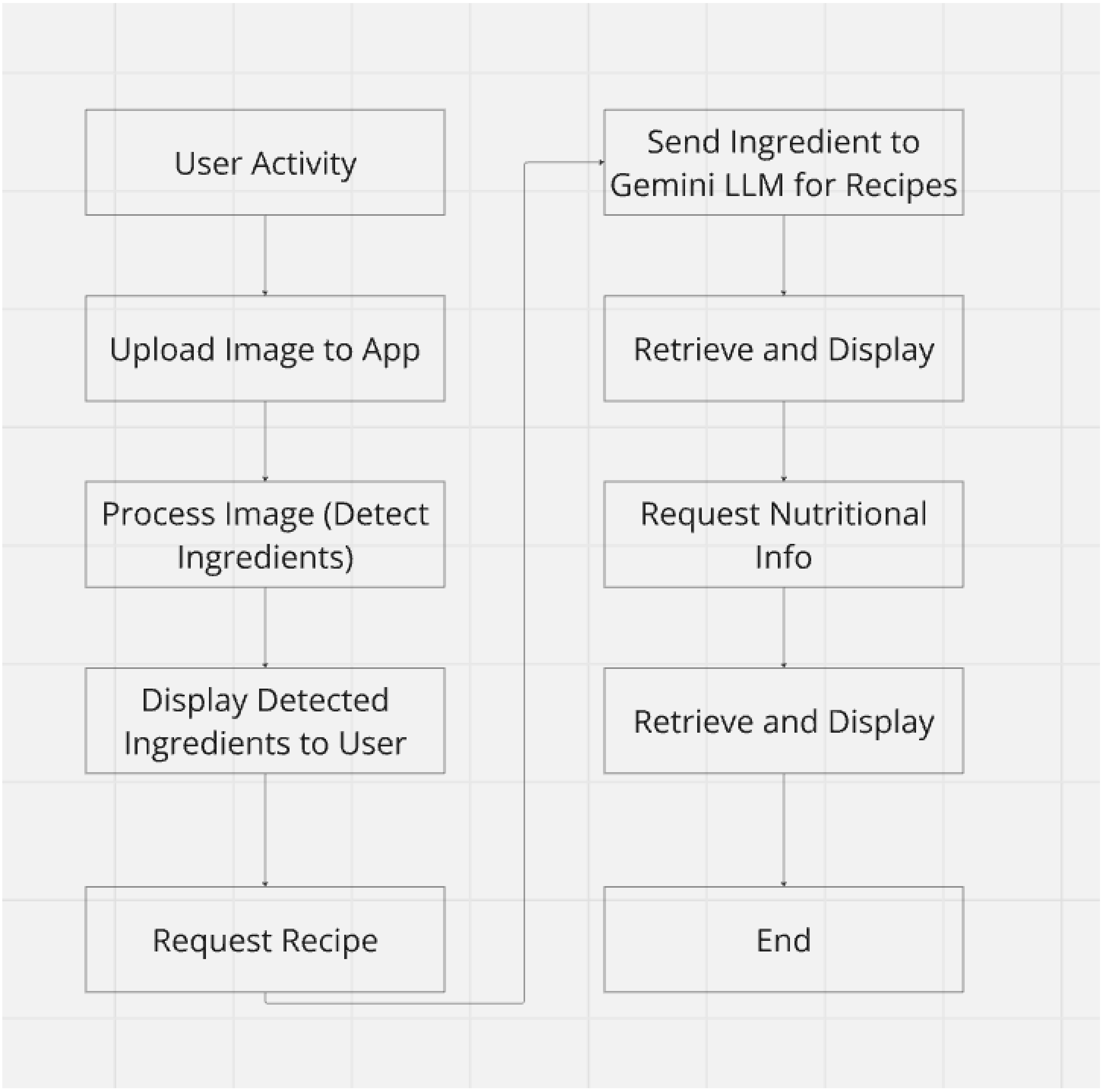


Fig 3.1.5 Activity Diagram for InstaChef - Smart Recipe Generator

ER DIAGRAM

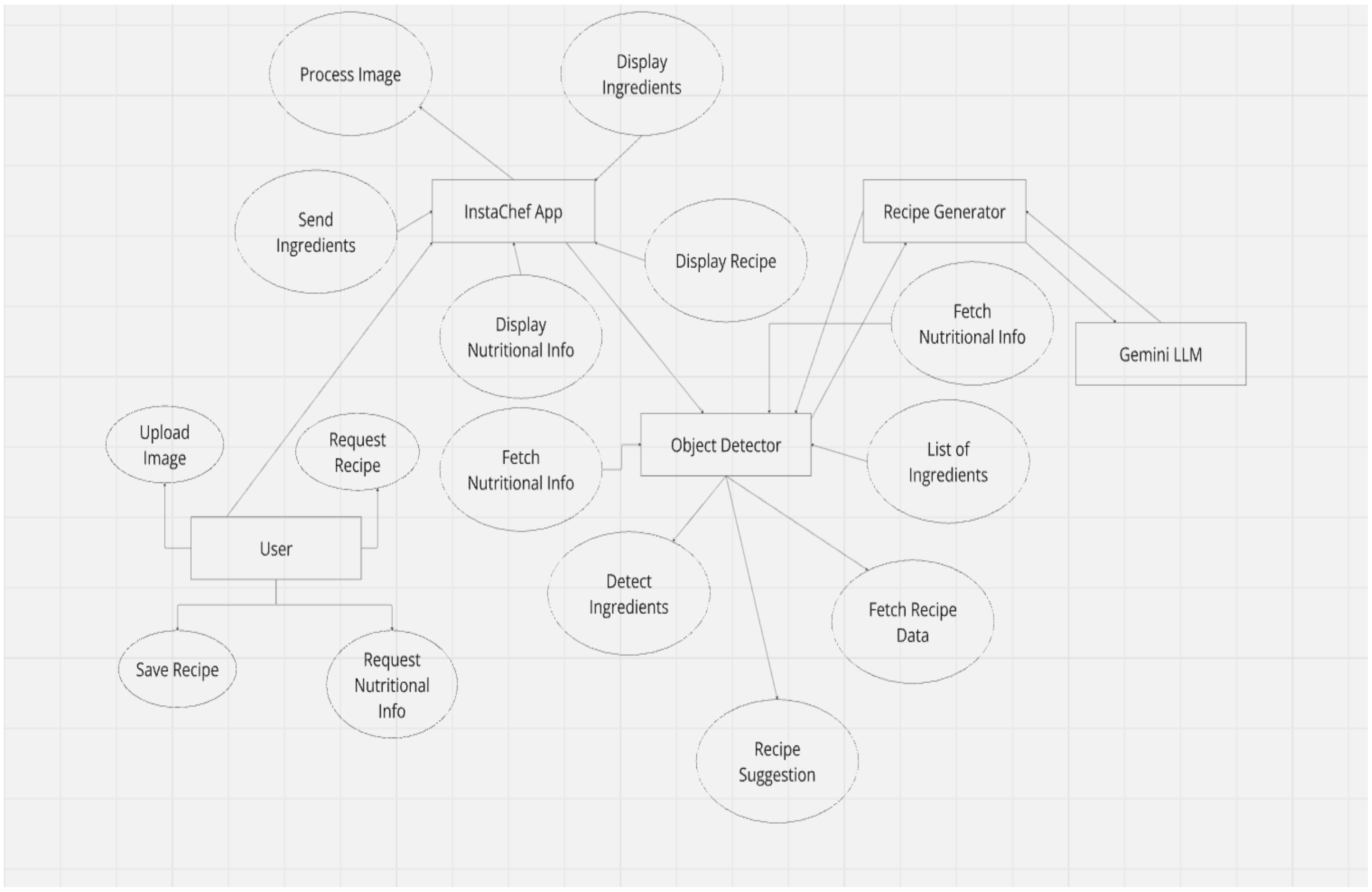


Fig 3.3 ER Diagram for InstaChef - Smart Recipe Generator

DATA FLOW DIAGRAM

0 LEVEL DFD

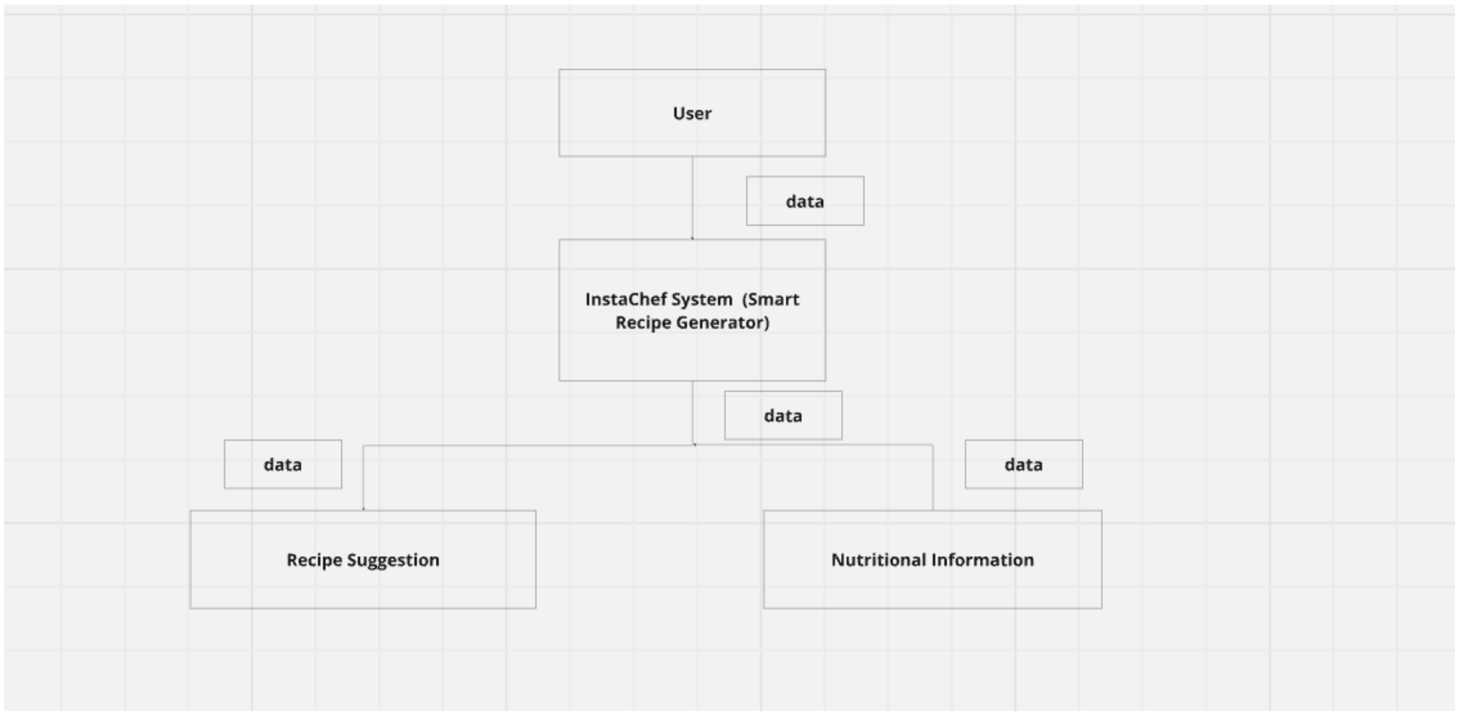


Fig 3.4.1 Dataflow diagram level 0

FIRST LEVEL DFD

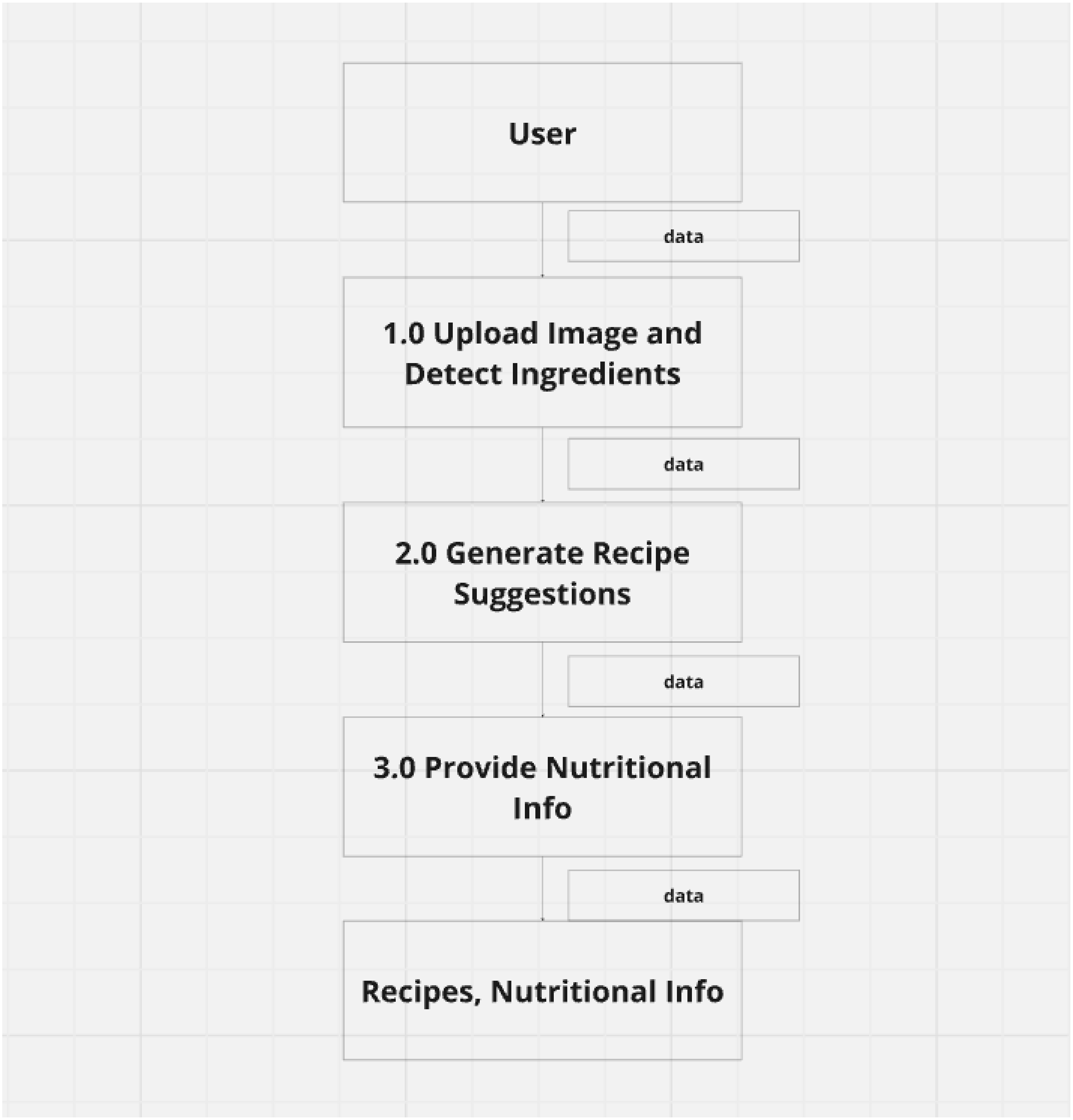

User
data
1.0 Upload Image and Detect Ingredients
data
2.0 Generate Recipe Suggestions
data
3.0 Provide Nutritional Info
data
Recipes, Nutritional Info

Fig 3.4.2 Dataflow diagram level 1

SECOND LEVEL DFD

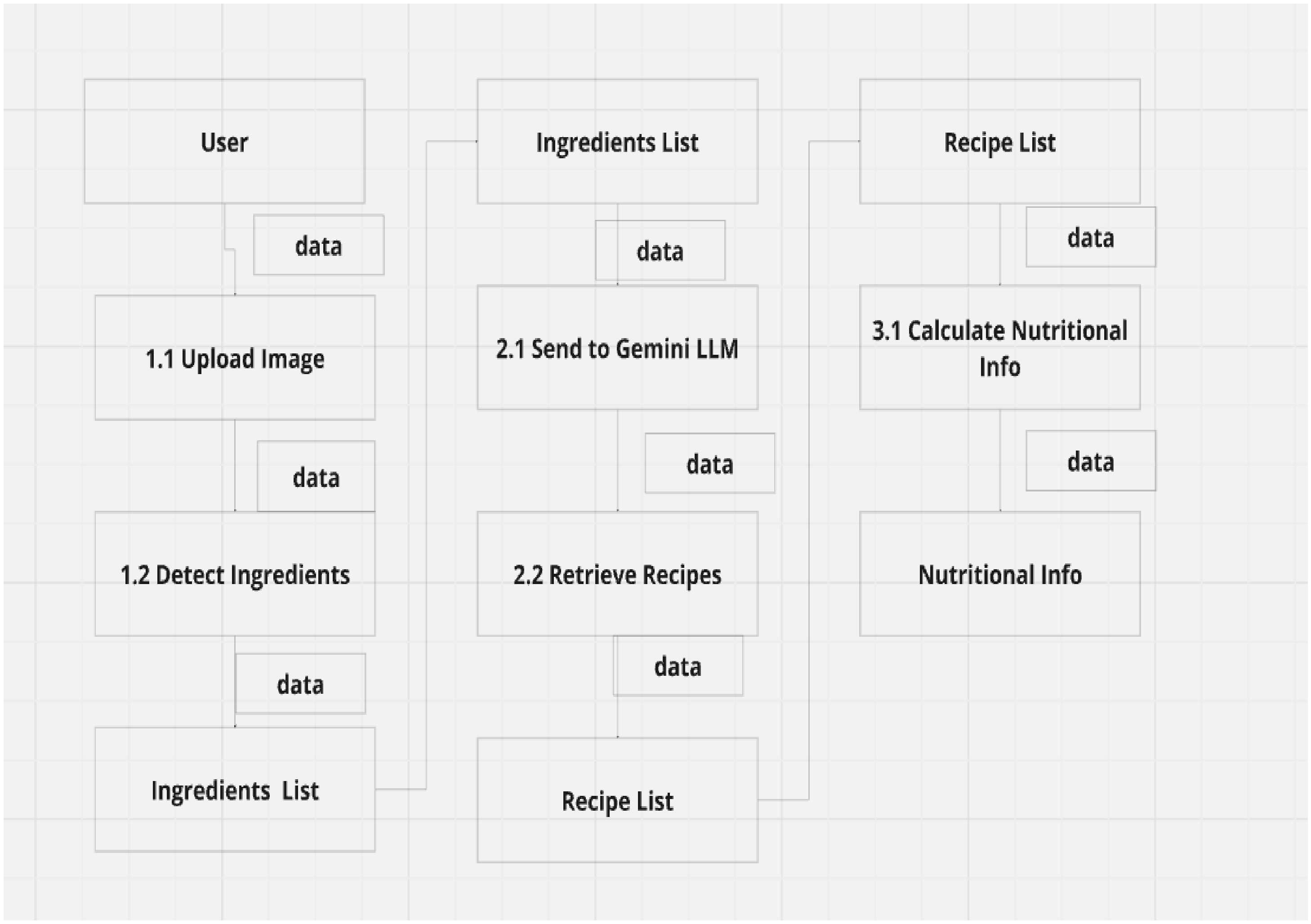


Fig 3.4.3 Dataflow diagram level 2

CHAPTER 4

SYSTEM ARCHITECTURE

ARCHITECTURE OVERVIEW

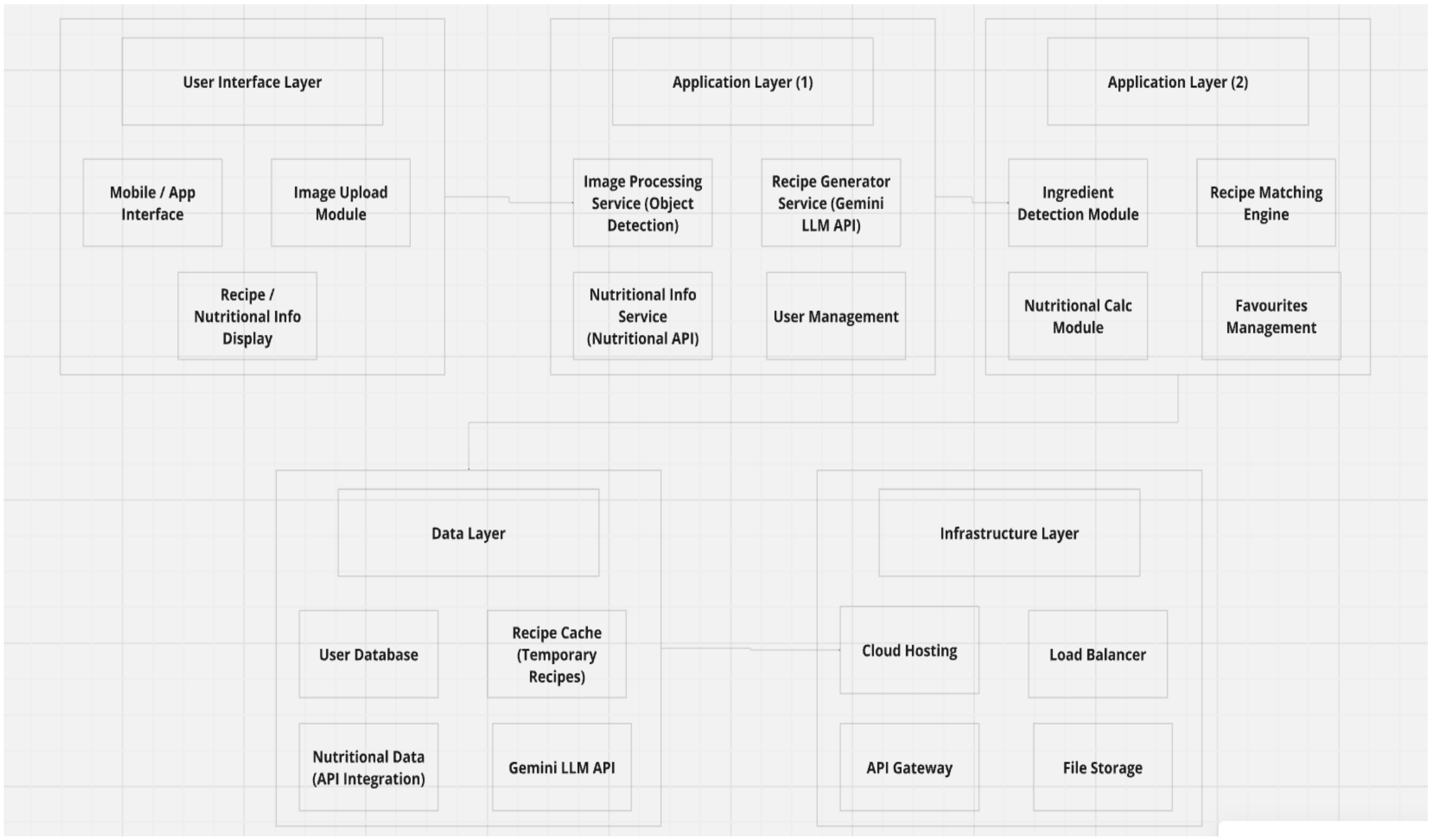


Fig 4.1 Architecture diagram for InstaChef - Smart Recipe Generator

The **InstaChef architecture** is organized into four main layers: **User Interface Layer**, **Application Layer**, **Data Layer**, and **Infrastructure Layer**, ensuring an efficient, scalable, and user-friendly platform for generating recipes from ingredient images.

1. User Interface Layer
This layer represents the front-end, where users interact with the system through a mobile app or web app.

- **Mobile/Web App**: Users can upload images of ingredients, view suggested recipes, check nutritional information, and save favorites. The app is built with frameworks like **React** (for web) or **Flutter** (for mobile), ensuring an intuitive and responsive user experience.
- **Image Upload Module**: Allows users to upload photos of their ingredients, which are then sent to the backend for processing.
- **Recipe and Nutritional Info Display**: Presents users with recipe suggestions and a detailed nutritional breakdown.

2. Application Layer
The **Application Layer** forms the core logic of the system, handling image processing, recipe generation, and nutritional calculation.

- **Image Processing Service**: Utilizes an object detection model (like YOLO or custom models) to detect ingredients in the uploaded image.
- **Recipe Generator**: Sends the list of detected ingredients to the **Gemini LLM** API to fetch relevant recipe suggestions from online databases. The recipes are tailored to user preferences (e.g., dietary restrictions).
- **Nutritional Information Service**: Fetches and calculates the nutritional value of each ingredient or recipe, displaying calorie counts and macronutrient data.
- **Favorites Management**: Enables users to save and retrieve favorite recipes for future use.

3. Data Layer
This layer is responsible for storing and managing data.

- **User Database**: Contains user profiles, saved recipes, and preferences.
- **Recipe Cache**: Temporarily stores recipes fetched from Gemini LLM for quick access.
- **External APIs**: Gemini LLM and nutritional databases provide recipe data and nutritional information.

4. Infrastructure Layer
The system is hosted on cloud services like **AWS**, **Azure**, or **Google Cloud**, ensuring scalability, security, and availability.

- **Load Balancer**: Manages incoming traffic to distribute workload across multiple servers.

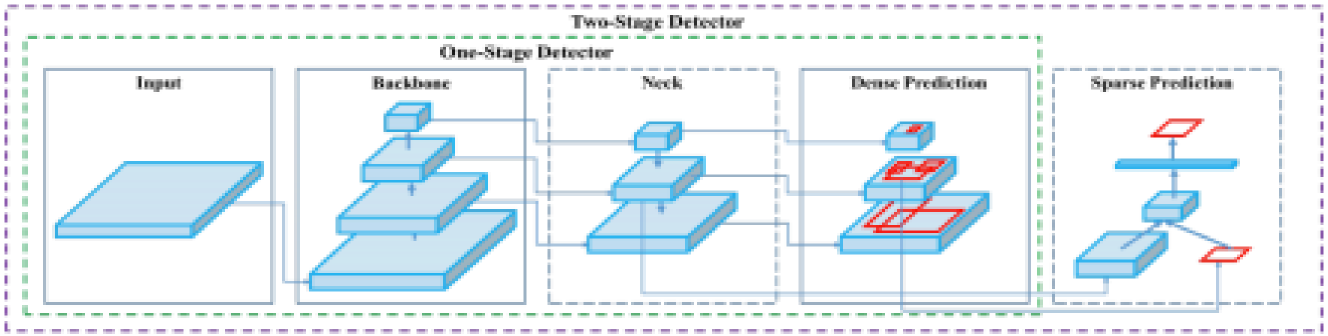
- **API Gateway**: Handles secure communication between the user interface and backend services.
- **Cloud Storage**: Stores uploaded images and cached recipe data.

This architecture ensures efficient image recognition, recipe generation, and nutritional data processing while providing a smooth, scalable user experience.

TECHNOLOGIES AND RELATED WORK

Technologies: Introduction of machine learning model in use

There are Different Object detection models available to use like R-CNN, YOLO, Faster R-CNN, etc. Among them, the YOLO object detection model is giving more accuracy with great speed as it uses Single-shot detection for speed and two-shot detection for better accuracy. For better speed YOLO object detection model is a good fit for the proposed project. YOLO object detection You only look once (YOLO) is a state-of-the-art, real-time object detection system. YOLO object detection model is very well known for its speed, accuracy also flexibility.The first three YOLO versions have been released in 2016, 2017, and 2018 respectively. YOLOv4 is related to it's previous YOLO version. It was released in early 2020. YOLOv5 was introduced in 2020 after a few months of YOLOv4 but, it is controversial and, there is no such difference between YOLOv4 and YOLOv5. Here, I worked YOLOv4 for this project as it is mature enough and resources are available to do some extra work.



Input: { Image, Patches, Image Pyramid, … }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], … }

Neck: { FPN [44], PANet [49], Bi-FPN [77], … }

Head:

Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], … }

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], … }

Fig 4.2.1 YoloV4 Object Detection Model

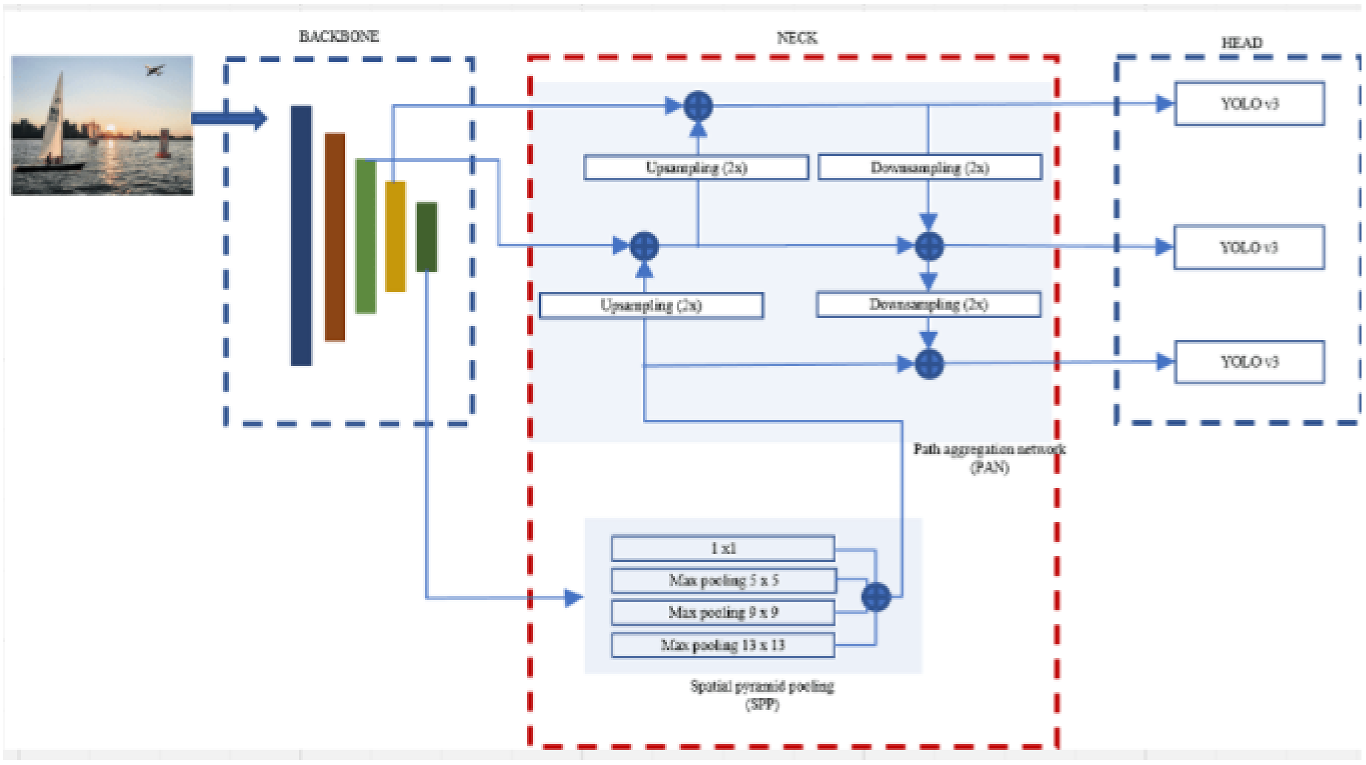


Fig 4.2.1 YoloV4 MatLab and Simulink

The YOLO v4 Object detection model is based on its prior object detection model, which is YOLOv3.

Layers

Input: Here, it takes Input as an image.

Backbone: YOLO uses different convolution neural networks as a backbone for ex VGG16 for classification, 50 layers of ResNet neural network, etc.

Neck: This layer used to predict object on a different scale by using future pyramid networks like FPN, Bi-FPN

Head: This layer is used to predict the dense area where two objects are very close and, for that Single-Shot Detector (SSD), Region Proposal Network (RPN), etc.

For the objects are on some distance in those case for the prediction Faster R-CNN used for prediction in YOLOv4. The sparse prediction uses a regional method for detection, so it also considers as second shot detection.

Related Work

Making Custom Data set and Training Object detection is still new and, It is not a mature thing. But the use case of this technology is limitless.
For specific context related to self-checkout stations, The work done by Fruits and Vegetable Classification from the live video Lukas Danev. They Make a micro-app and Make a custom dataset which separates by with bag and without the bag. And with Segmentation and Gaussian
Bayes Classifier, They get good accuracy on their model 85% for the top three predictions, Which is great at those time 2017. Another study is Tunnel type Image-based system . Which is considered as the same context, but it includes hardware to implement. Tiliter vision They do the same context, where they make a self-checkout machine which takes an image as input and scale it and give the price for that. Their proposed system works well with the fruits and detection without a bag. It cannot show bag detection.
It is not the optimal solution, to buy fruits or vegetables without a bag if we buy it in bulk. Another Study Conduct on the topic Simplifying Grocery shopping with Deep Learning by Jing Ning, Yongfeng Li, Ajay. Ramesh Where they mask image labels by pixel-level annotation (It fills out the image by the border of an object). They use the R-CNN model for the training and achieved 84% of mAP(mean average precision) for a default threshold of 50%. Their main focus on all items of the grocery store that includes items with the barcode as well.
The Research-based on fruit verity classification is similar research as they classify fruits and their verity as well. For the classification problem, they used a double-track method where they used two 9-layer CNN one is used for the RoI detection and the other is used for classifying fruit category with the detected RoI. They got a good classification speed which is an avg 350ms, It is slightly more than the YOLOv3 Realtime object detection .

CHAPTER 5

SYSTEM IMPLEMENTATION

DATA ACQUISITION

The most important part of the Dataset Acquisition is how to capture the images, which can give you a good result after model training. We start from the basic where we make small datasets through the big datasets. All of the images were captured, With Default settings in the camera of iPhone 11.
• To train an object detection model, the datasets play an important role. There are different

datasets available on Kaggle, Open Image Dataset By google, and Some other websites.
• During the initial stage of my project, I reviewed many data set from them Fruit 360 one of them, but the limitation of that dataset is it is specifically developed to reticular fruit detection by segmented portion.
• For the YOLO object detection, we need to label a bunch of images with the background.
• Although Fruit 360 dataset works with R-CNN as per research, it will struggle with seeing through the semi-transparent plastic bag, which is the most crucial part of this project.
• To Conclude All this Scenario, I decided to make a custom dataset for my custom model.

Iteration 1:

• First Iteration contains only 1 class with 69 photos of the tomatoes with a bag and without a bag, which is such a small dataset for any object detection model.
• It is developed for the experimental purpose for checks how it can behave with the YOLO object detection model.

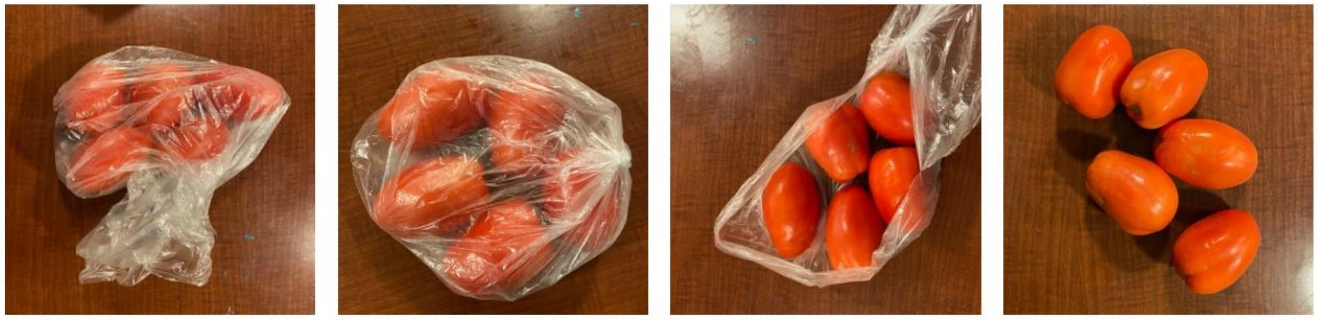


Fig 5.1.1 Iteration 1: Tomato with bag and without bag

Iteration 2:

• The second iteration contains three different classes, which include lemons, chilies, apples. These classes are with and without a bag.
• Here, there were 165 images taken during the 2nd iteration. It is randomly captured images without worrying about the background and other things.

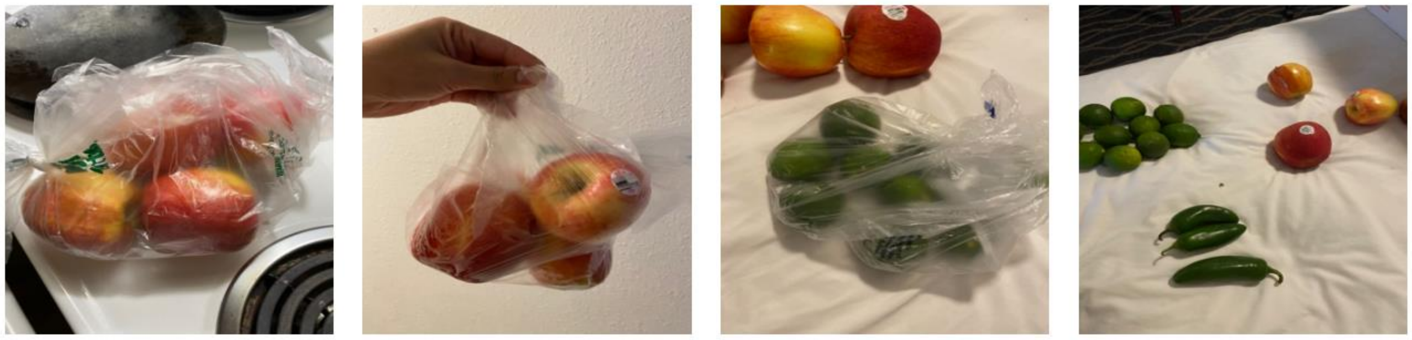


Fig 5.1.2 Iteration 2: Class Datasets with and without bag

Iteration 3:

• During the 3rd Iteration, we got exposure to dataset creation. Here we decided to make a

platform that depicts the environment at the self-checkout stations.
• To depict the same environment as self-checkout stations, we put the chrome plate as a background, and for steadiness in all photos, we set up the phone in one place until we got the entire dataset.
• Here is a little look at the DIY-home setup of our platform.

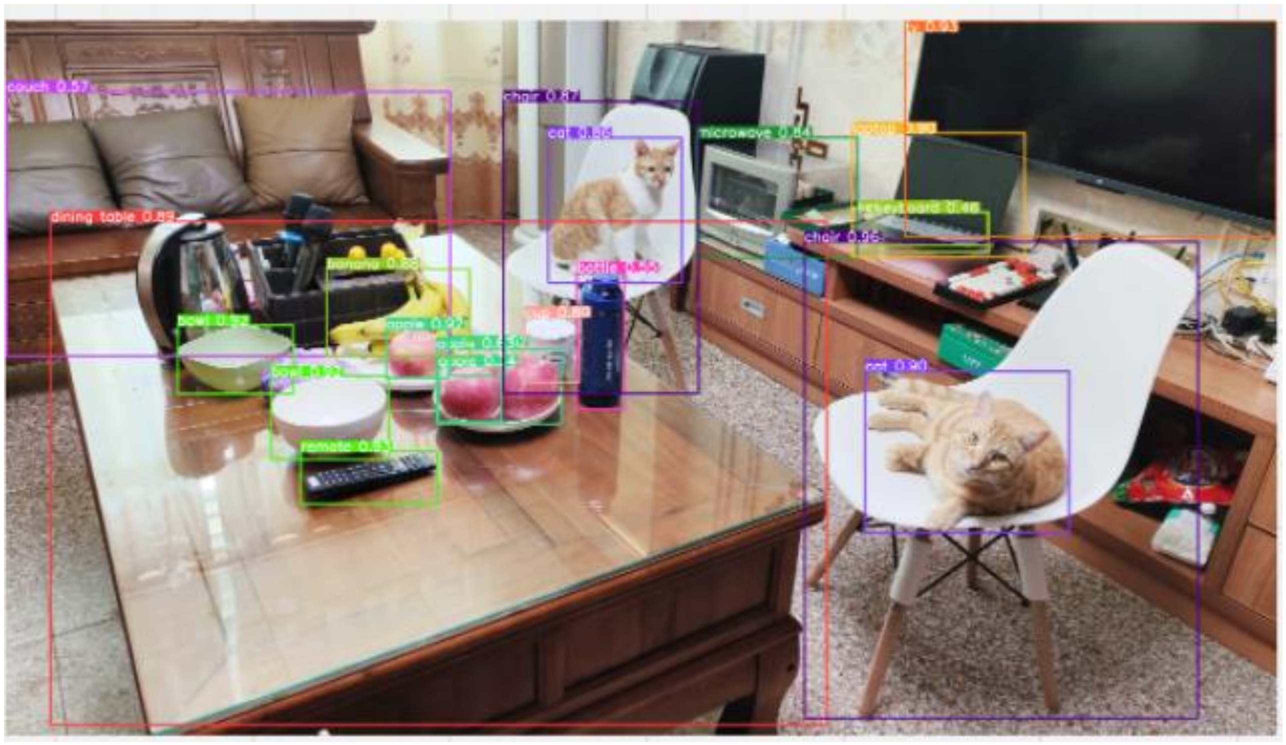


Fig 5.1.3 Setup for Data Acquisition

• Here, during the 3rd iteration, we create 14 separate classes in which we took approx 50 images of every individual class.
• 14 class are banana-bag, banana, blackberries, raspberry, lemon-bag, lemon, grapes-bag, grapes, tomato-bag, tomato, apple-bag, apple, chili-bag, chili
• For proper lighting, we set up a lamp backside of the phone stand.

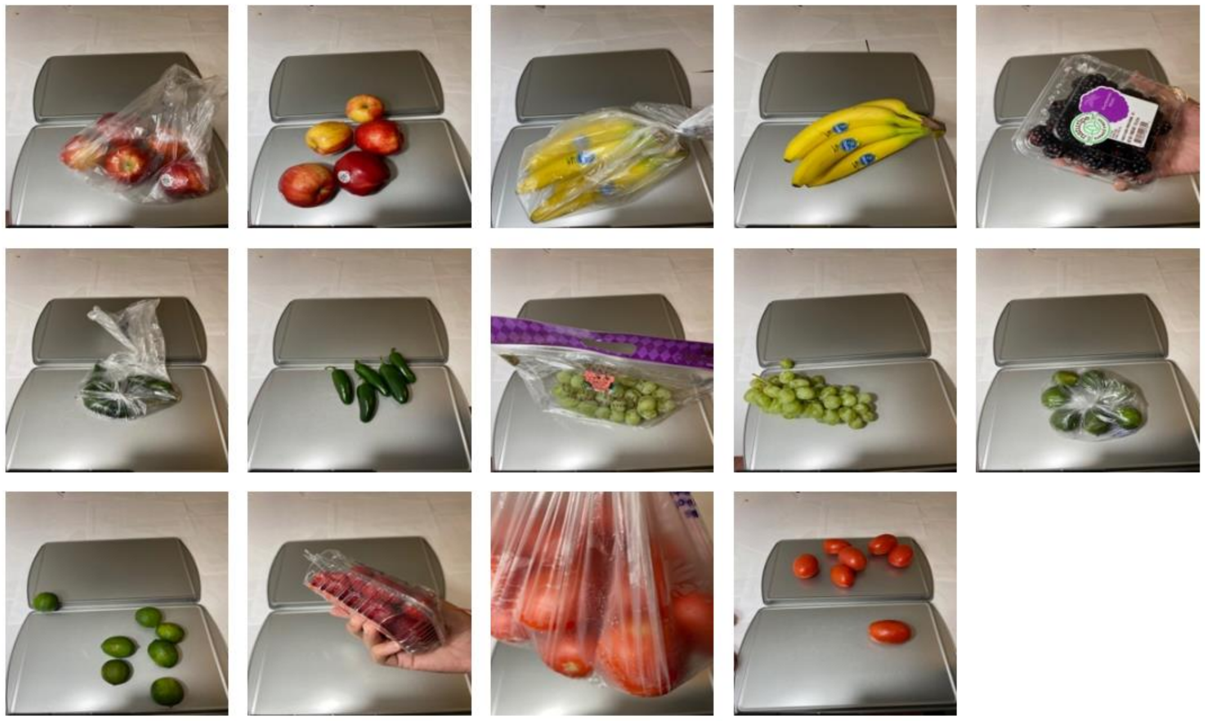


Fig 5.1.4 Iteration 3: Classes with and without bag

• Here, we took photos by putting an object near and far to the camera. That way our model can learn for all of the scenarios.
• Here, after Iteration 3, we took a total number of 656 images with a different angle and different distance from the camera.
• To give a feel like a self-checkout station we put chrome plate so when it will get to the production model have a little more accuracy due to the identical background.
• In some of the images, we intentionally keep our hand in the images. It will give the feel of a customer's hand during the detection of the item.
• Some of the images we keep half in the bag half outside of the bag. That way we can assume if some of the items keep outside of the bag then it can still perform the detection.

IMAGE LABELING

• With the dataset labels for every image, the deep learning model can recognize an object and learn a particular object for a certain class.
• There are plenty of Image labeling tools available e.g., LabelImg, VGG Image Annotation, Labelbox.
• For the YOLO object detection, each label needs to be labelled an object in a rectangular shape on its class. LabelImg is the best suitable, popular, and easily available python tool.
• We manually labeled 656 images, which generate 656 .xml label files for each image.

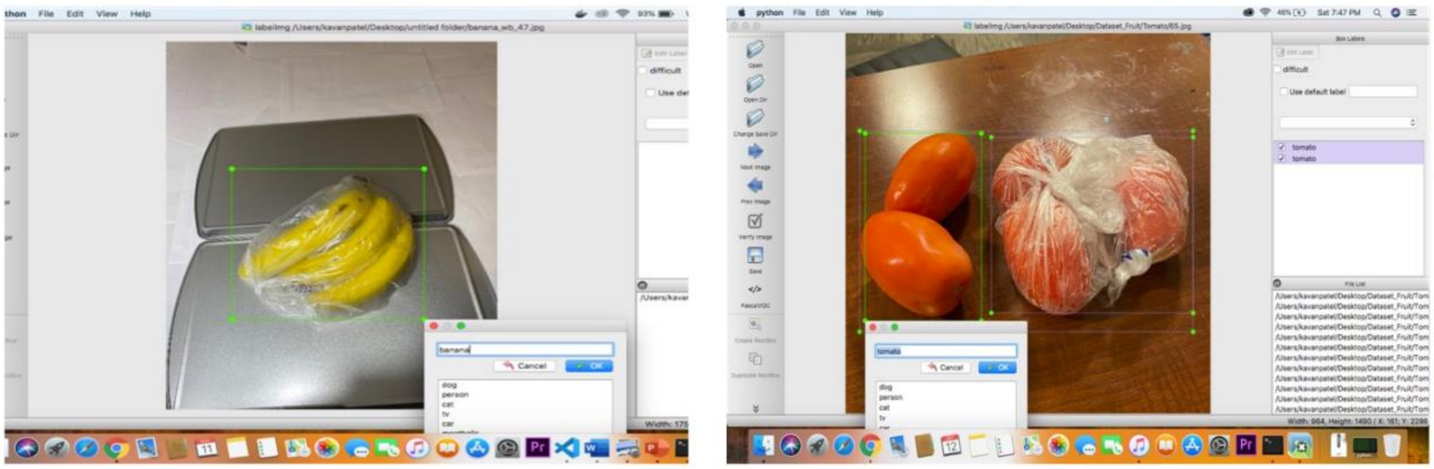


Fig 5.2.1 Iteration 3: Labelling Interface

DATA AUGMENTATION

• Data Augmentation is a step to increase the size of the data set by doing some transformation on the images for e.g., Rotating an image by some angle, adding noise to an image, shifting image, mirroring the image, Adding filters to the image, etc.
• For Image augmentation, there are plenty of resources are available but image augmentation with Yolo labels CloDSA is one of the base open-source programs with using that we can do our necessary transformation to the image as well as labels [12]. Here, we used a notebook provided by CloDSA to Do augmentation on our dataset.
• CloDSA accepts image files and .txt formatted labels. During data labeling, we created labels in the .xml file. We used XmlToTxt Project by Isabek to convert all .xml file to .txt file [13].
• For our data augmentation, we do not use all transformations of CloDSA for e.g., adding filters can change the color of the fruits or vegetable that can lead to our model in a different direction. So, we used Vertical flip, Horizontal flip, 90-degree rotation, average blurring, Raise Hue, Horizontal and vertical flip, and keep one copy of the original image.
• Here we did 6 types of transformation, so our dataset size increased from 656 to 4592 images. If we consider storage it increases from approx. 5Gb to 15Gb.

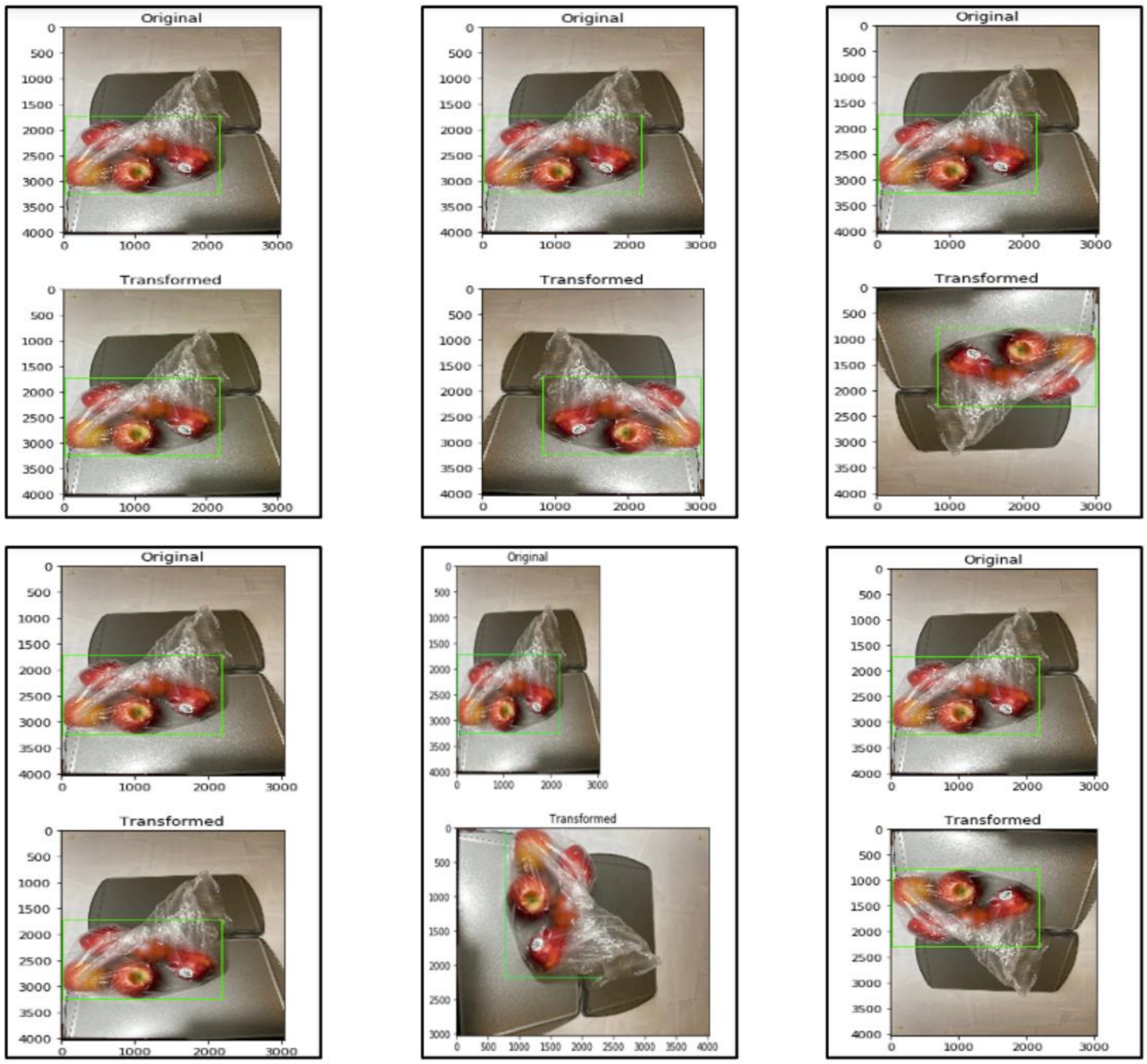


Fig 5.2.2 Various Transformation on the image

MODEL TRAINING

The ultimate goal for the training model is to get a good result with minimum loss. To train any Deep Learning model requires powerful GPU Hardware. Here we are training the YOLOv4 Custom object detection model. As this model deals with the images instead of only variables powerful GPU are necessary. Google Colab already provides a free Powerful GPU to use which is Tesla K8 up to 12Gb. For that, we move all our datasets to Google drive to use them.

These are the Steps involved to Training to get the actual model:

Cloning & Building Darknet on Colab Virtual machine:18

Darknet is an open-source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation [14].
Building Darknet on our virtual environment is in just two steps
1) Clone darknet repository from git hub [15].
2) Use !make command to build it.

1. Mounting Google drive with Colab:

We are used google drive as a storage for our dataset. The mounting drive is just a need drive.mount('/content/gdrive') command.
Before the training, we move our custom dataset stored on google drive, to google Colab's virtual machine. The reason to use google drive is to move the data to Colab with using the drive is too fast.

2. Attach Configuration files for training:

Cfg file is attached to tell the model that what is the max batch size, step size, number of classes we are training for, filters, etc.
We got a sample yolov4-custom.cfg file from the darknet repository of AlexeyAB [15].

3. Downloading yolov4 weights to make training process faster:

By downloading pre-trained yolov4 weights makes the training process more accurate and faster.
It can use their pre-trained model's weight and filter in our training, which helps to learn our model faster in a short period.

4. Training:

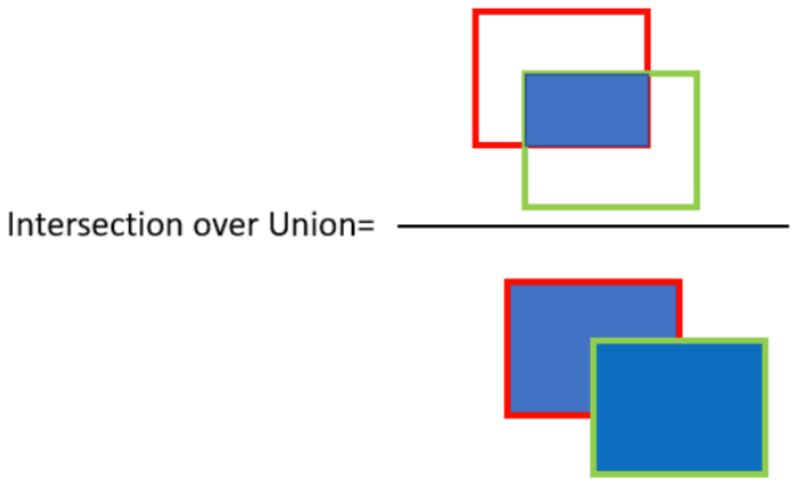
Yolov4 custom object detection traing is started by following line.

!./darknet detector train data/obj.data cfg/yolov4-obj.cfg yolov4.conv.137 -dont_show -map[15].

Where first it selects train command of detector class from darknet environment.
Then we attached obj.data file which describes different paths:
The number of classes, training & validation files path in .txt format, a path for obj.names, a path for the backup folder. 19
Here obj.names file describe all class names, in our case banana, apple, banana bag, chili, etc.

mAP (Mean Average Precision) Calculation

the mAP is the most common term in deep learning to evaluate models. It calculates based on how well your prediction is. Calculation of the mAP is based on Precision, Recall, IoU (Intersection Over Union).**IoU:**
IoU says how good your prediction bounding boxes are.



- Red-ground truth bounding box;
- Green-predicted bounding box.

Fig 5.2.3 IoU (Intersection of Union)

Here Red boxes are actual label created during the labeling process and green box are the predicted boxes so IoU says how well our prediction fit to the actual image label.

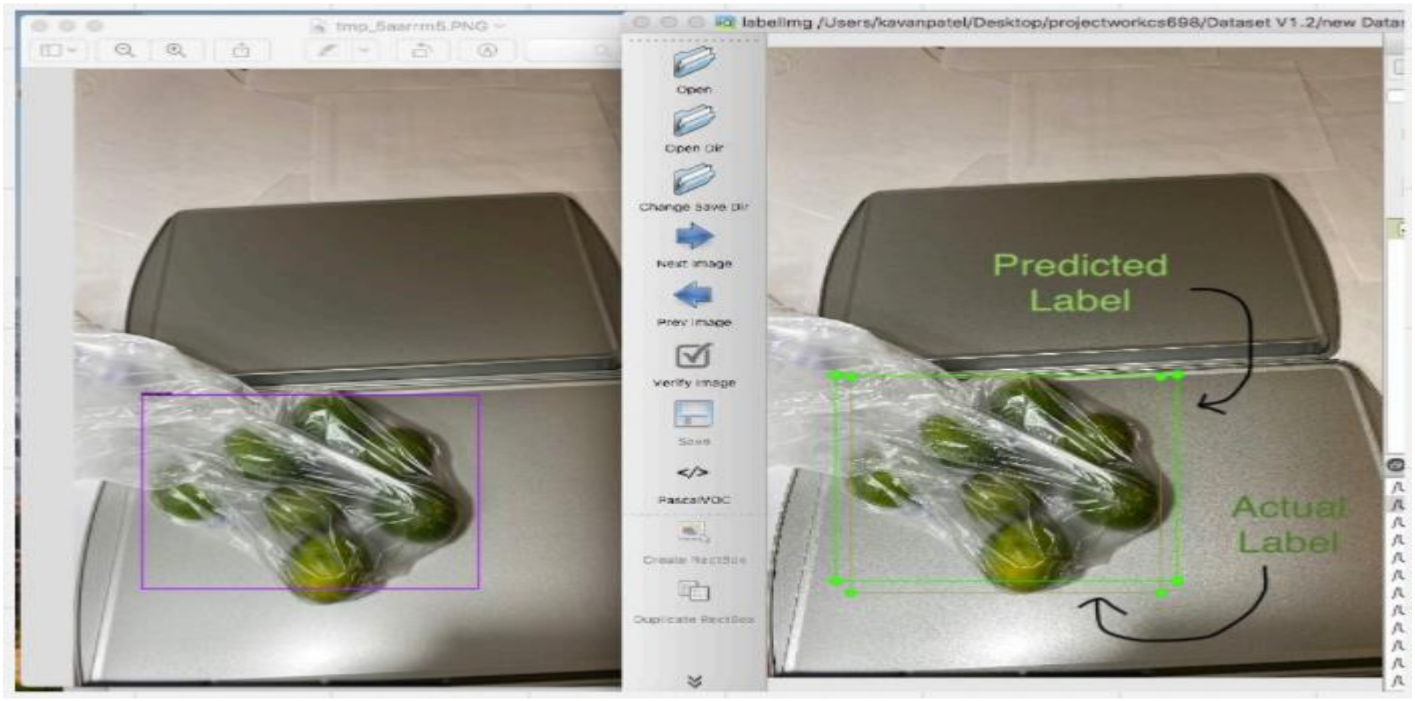


Fig 5.2.4 IoU (Intersection of Union) in Real

Here, in figure 4.1.4.2 the dark green(Actual Label) marked label boxes are actual labels made during the dataset labeling process. And Light Green (Predicted Label) marked label boxes are predicted labels by our custom trained label.

Precision calculation:

 if IoU is >= 0.5, it classifies as True Positive (TP) [16].
 if IoU is <0.5, it classifies as False Positive (FP) [16].
 When the label is present in the image and the model cannot be able to predict then it classifies as False Negative (FN) [16].
Precision P= TP / FP + TP
Recall R= TP / FP + FN
mAP= 1 / ∑ APi where N is total number of images.
AP (Average Precision): It is an average value of precision at 11 points on PR curve. 21
PR-Curve is made by Precision and Recall values. Where AP is calculated by average of maximum precision value at 11 recall point ranging from (0.0, 0.1, 0.2,…,1.0)

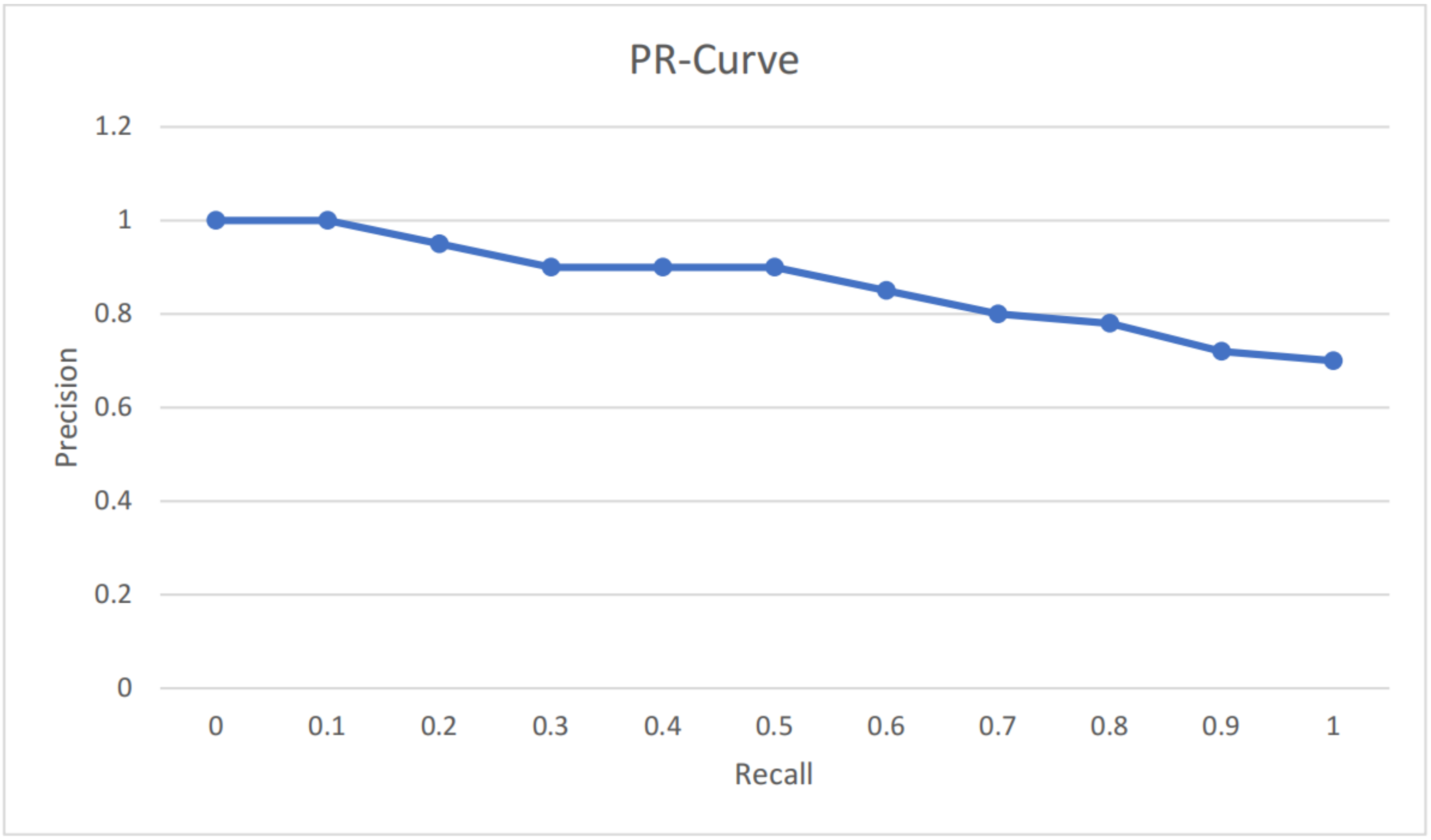


Fig 5.2.5 P-R Curve

To calculate AP (Average Precision) of above graph = 1/11 [1+1+0.95+0.9+0.9+0.9+0.85+0.8+0.78+0.72+0.7] = 0.86 (86%)
In mAP calculation the threshold value for IoU is by default 0.5, It can be changed upon a model.
The mAP is the most important part to check any model's accuracy. During Yolov4 custom object detection model training, it saves our model every 1000 iteration on our defined path

(usually in our google drive).
The ideal time for the model training is 6000 steps for better accuracy as per the documentation of yolov4 [15]. We can stop training at 2000 steps it will give fair results. Here we start training our model on Colab and let them run overnight around 12 hrs. and we go done until 2000 steps. After testing our model, we thought it needs to get 6000 steps. To get more accuracy. So we kick start our training from the last saved .weight file and do this process two to three more times because colab provides free GPU for a limited time. After our training until 6000 steps, we calculated mAP for each of the saved models, and results from our mAP calculation are perfectly aligned with the YOLOv4 official document and we got proper accuracy on actual detection which we needed so we stopped our training at 6000 steps and move forward to next model conversion and deployment process.

Iteration	1000	2000	3000	4000	5000	6000	Best weight (6000 +)
mAP	0.7782 (77.82%)	0.9736 (97.36%)	0.9914 (99.14%)	0.9918 (99.18%)	0.9939 (99.39%)	0.9924 (99.24%)	0.9952 (99.52%)

Fig 5.2.5 mAP Tabulation

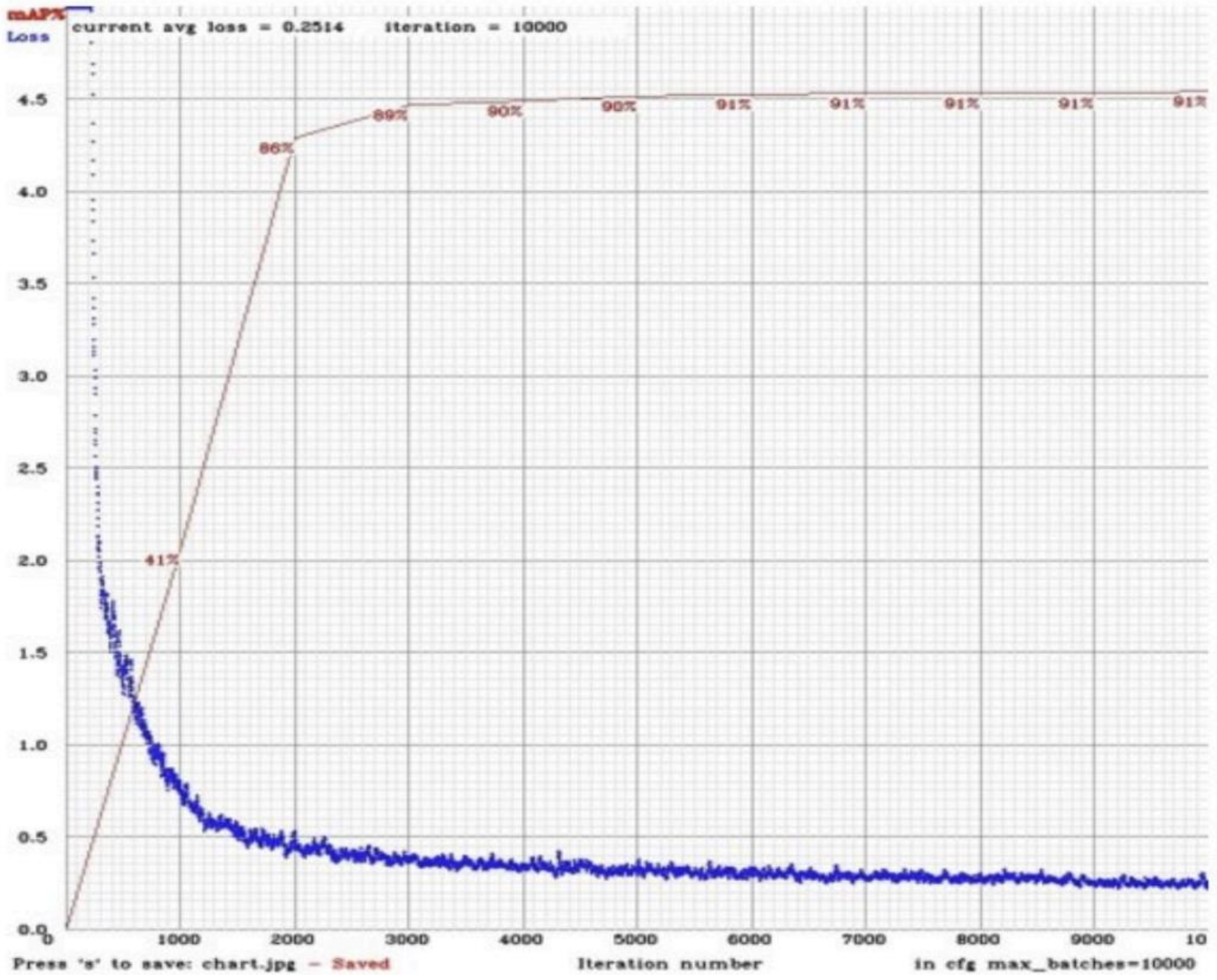
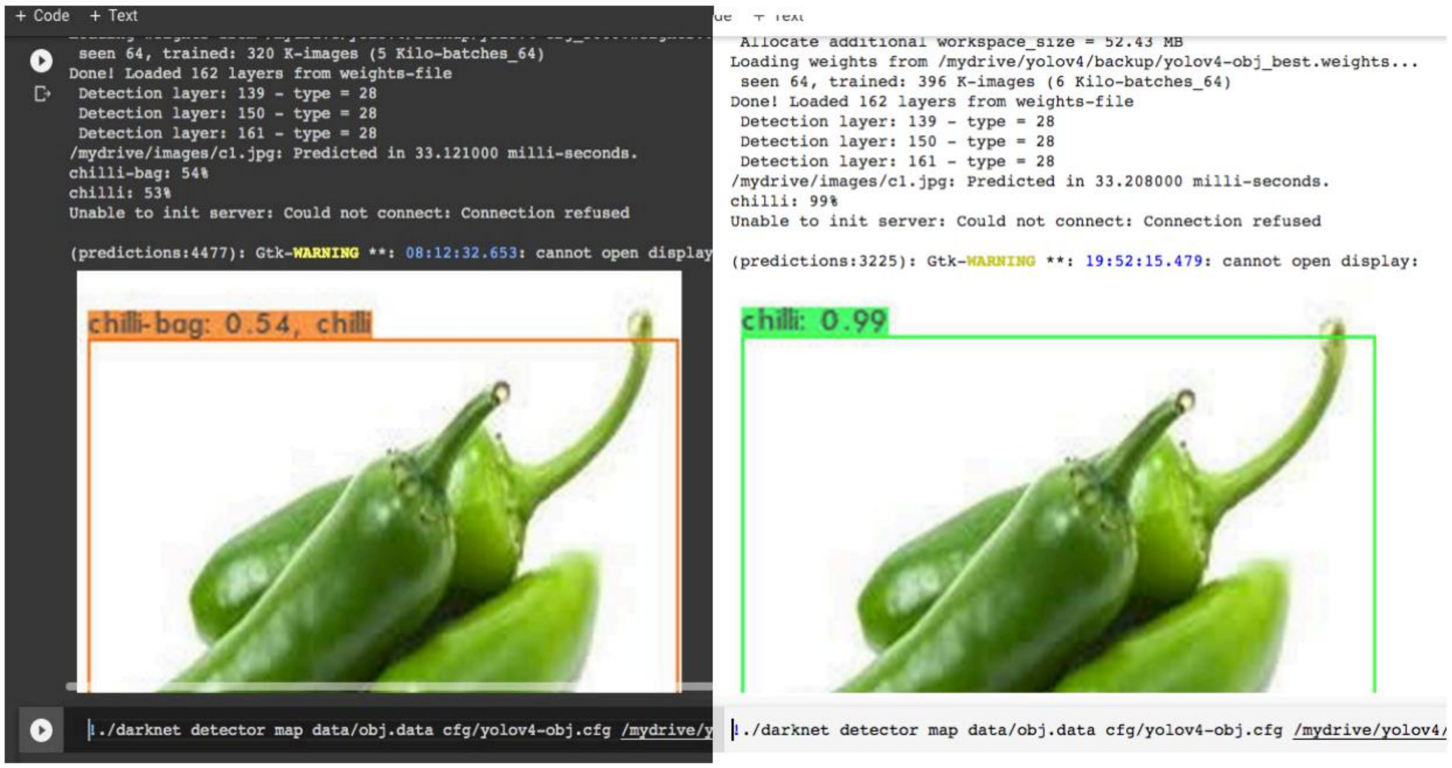


Fig 5.2.6 mAP vs Loss over Iteration

Here above table shows mAP calculation by our generated model and figure 4.1.3 shows mAP vs. Loss over iteration provided by official documentation if we match our table number with the graph it completely aligns with the graph for every 1000 iteration.
Here is the sample mAP Calculation from actual result at 6000+ iteration:
IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP @ 0.50) = 0.995275, pr 99.53 %
Total Detection Time: 109 Seconds

Weight files are the model file which consists of weights for the detection. During the training, we got a weight file for every 1000 iteration and we test those files on images to see the results



```
 seen 64, trained: 320 K-images (5 Kilo-batches_64)
Done! Loaded 162 layers from weights-file
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
/mydrive/images/c1.jpg: Predicted in 33.121000 milli-seconds.
chilli-bag: 54%
chilli: 53%
Unable to init server: Could not connect: Connection refused

(predictions:4477): Gtk-WARNING **: 08:12:32.653: cannot open display
```

```
!./darknet detector map data/obj.data cfg/yolov4-obj.cfg /mydrive/y
```

```
 Allocate additional workspace_size = 52.43 MB
Loading weights from /mydrive/yolov4/backup/yolov4-obj_best.weights...
 seen 64, trained: 396 K-images (6 Kilo-batches_64)
Done! Loaded 162 layers from weights-file
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
/mydrive/images/c1.jpg: Predicted in 33.208000 milli-seconds.
chilli: 99%
Unable to init server: Could not connect: Connection refused

(predictions:3225): Gtk-WARNING **: 19:52:15.479: cannot open display:
```

```
!./darknet detector map data/obj.data cfg/yolov4-obj.cfg /mydrive/yolov4/
```

Fig 5.2.7 Earlier Model Detection vs Recent Model Detection

It is a comparison of the model detection at 2000 iteration vs model detection at 6000 iteration and we observed a huge difference in detection in the early stages.

Yolov4 Custom Object detection model trained entirely on Darknet environment. It can perform detection on the darknet environment during the coding process but to use our model for real prediction It needs to be deployed on GUI.
TensorFlow provides many different methods to deploy our model into actual production. There is plenty of TensorFlow API used to convert our model as a TensorFlow model like TFJS, TFlite.

TFJS: It is the TensorFlow version of the deep learning model which is very lightweight and can be served through the web.

TFlite: It is the TensorFlow version of the deep learning model which is lightweight and used for mobile and IoT(Internet Of Things) Devices.
TensorFlow: It is a general-purpose TensorFlow version of the deep learning model.
Here, we used TensorFlow-Yolov4-Tflite API to convert our darknet Yolov4 model to the TensorFlow model. This conversion generates our TensorFlow version of the model with .pb (protobuf file)

Protobuf file is developed by google which contains graph information and weights of the model. Directory artifact of the YOLOv4 model After Converted to TensorFlow:

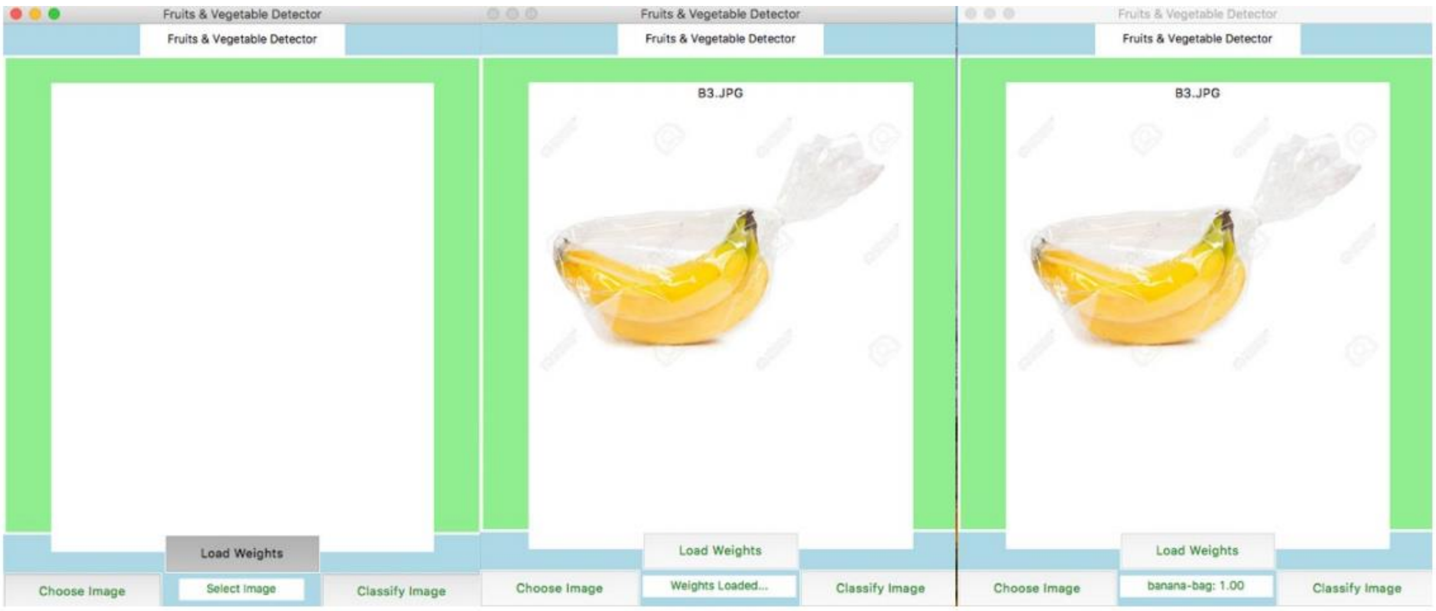
```
Yolov4-416
|
------/assets
------/variables
|
------ variables.data-00000-of-00001
------ variables.index
------saved_model.pb
```

CHAPTER 6

DEPLOYMENT

DEPLOY ON GUI (GRAPHICAL USER INTERFACE)

Deployment is a step to serve the model available to others. The deployment model is simple as it takes input and gives an output (in our case image with prediction).
According to Study in data science study, 87% of the data science project never got into production. Lake of deployment can be one of the reasons where the project is developed but it never actually out from the console.
Deployment can be done in various ways on various platforms. For example, Google provides google cloud, Amazon provides an amazon sage maker for cloud-based machine learning model deployment.
Here for our project, local deployment is enough for our need as it reduces the cost of the cloud, so it makes no extra cost project.
Here, we used the Tkinter library of python, which is Great for GUI development.
We try to keep it simple, so there is a load weight button to load our model weight (model weights need to be load only once). Then load the image button to load our image and the Classify image button to predict it.
On press of the classify, it can request the TensorFlow version of the YOLOv4 model and get the prediction.



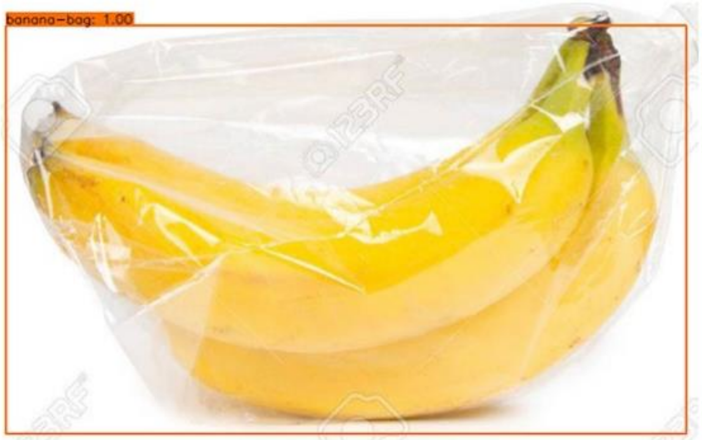


Fig 6.1.1 GUI of Fruits and Vegetable Detection POS

CHAPTER 7

CONCLUSION

CONCLUSION

By model analysis and accuracy performance, we can say that the generated model is giving enough results, for our fruits & vegetable detection POS purpose. mAP of the model is 0.94 which also says about model accuracy.
To load the weight in the application, it takes up to 30 sec on very less powerful CPU, which can be improved with the good CPU or GPU
During actual testing, for some of the images, we still got false prediction or no prediction in the case of the multiple objects. In the self-checkout station, there is a need to be only one item placed on the counter, so we can ignore the problems with the multiple objects, where the problem of the false prediction can be solved by post processing output.

FUTURE ENHANCEMENTS

Here, we trained our model for only 14 different classes, upon the success of our model it can be trained mode class as per need by repeating the same process of custom model training.
Prediction from the live video is always an exciting part of any object detection model. Our model can do that but, it requires solid hardware and few ticks.
Sometimes as 5 to 10 % of times, it predicts as a false class. It can be eliminated by feeding more than one image in a system, during image feeding from the live video (e.g., we can capture five different images during image acquisition on the interval of 0.5 sec) make five different predictions and use the max function to choose output. That can improve real time model accuracy more.
Developing a good GUI for better results is still part of future work.

CHAPTER 8

APPENDIX

Tools & Libraries Used:

- Python 3.7
- Tkinter: For making GUI Using python
- OpenCV: an open-source library for Computer Vision
- Tensorflow-Yolov4-Tflite: Converting YOLOv4 Model to TensorFlow Lite Model
- NumPy: Math Operation (Operation with tensor)
- TensorFlow: an open-source machine learning library
- LabelImg: A tool to Label Customize dataset
- Colab: A Cloud IDE for developing code on Cloud and use powerful GPU
- Google Drive: To store custom dataset for the training
- JPEG mini: Tool to compress the image size without loosing quality
- PIL(Pillow): A Python free Open-source image Library

CHAPTER 9

REFERENCES

1) AI Involvement in Supermarket: Food west problem
Blog (https://www.futurithmic.com/2019/06/10/forget-self-checkout supermarkets-future-will-rely-on-ai/)
2) Cheating By changing Fruits or Vegetable name at self-checkout
Blog (https://www.dailymail.co.uk/news/article-5774293/Brits-using-self checkout-machines-steal-expensive-fruit-veg-saying-theyre-buying-carrots.html)
3) Cheating By changing Barcodes at self-checkout
Blog (https://www.theatlantic.com/magazine/archive/2018/03/stealing-from-self checkout/550940/)
4) YOLO: Real Time Object detection:
Blog (https://pjreddie.com/darknet/yolo/)
5) YOLO Versions & History:
Data Science (https://towardsdatascience.com/yolo-v4-or-yolo-v5-or-pp-yolo dad8e40f7109)
6) Bochkovskiy, Alexey & Wang, Chien-Yao & Liao, Hong-yuan. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection.
Research Gate (https://www.researchgate.net/publication/340883401_YOLOv4_Optimal_Speed_and_Accuracy_of_Object_Detection)
7) Fruits and Vegetable Classification From Live Video
Danev, L. (2017). Fruit and vegetable classification from live video. **Semantic Scholar (https://www.semanticscholar.org/paper/Fruit-and-vegetable-classification from-live-video-Danev/e3c565f3b6b90f91017a844d0c664e85f6be52ca?p2df)**
8) Tunnel-type digital imaging-based system for use in automated self-checkout and cashier-assisted checkout operations in retail store environments
Patents (https://patents.google.com/patent/US20090134221A1/en)
9) Tiliter Vision: AI Self-checkout
Blog (https://www.tiliter.com/blog/blog-post-title-two-kd2tg)