Faculty of Engineering - Cairo University
Credit Hour System Programs

Computer and Communications Engineering

CCE-C

Graduation Project Report
Spring 2021

Eye Food

Prepared by:

Amir Salah El-Ahmady AbdulRaheem
Karim Ibrahim Amin Wahba
Mostafa Sherif Ahmed Mourad
Youssed Sayed Mostafa Mohamed

Supervised by:

Prof. Ahmed Darwish

Cairo University
Faculty of Engineering
Credit Hour Programs

جــــامـعـة الـقـــاهـرة
كـلـيــــة الـهـنـدســـة
برامـــج الساعــات المعتمـدة

Eye-Food

A Graduation Project Report Submitted
to
Faculty of Engineering, Cairo University
in Partial Fulfillment of the requirements of the degree
of
Bachelor of Science in Computer Engineering.

Presented by

Amir Salah El-Ahmady
Karim Ibrahim Amin
Mostafa Sherif Mourad
Youssef Sayed Mostafa

Supervised by

Prof. Ahmed Darwish

9/7/2021

Cairo University

Credit Hour Systems Programs

Directory of B.Sc. Graduation Projects

Faculty of Engineering

Computer and Communications Engineering (CCE-C)

Project Code	**CMPN481**	
Project Title	**Eye-Food**	
Keywords	**Computer Vision, Smart Glasses, Food Recognition, Visual Impairments, Object Detection, Image Processing**	
Students	**Name: Amir Salah El-Ahmadi** **E-mail: ameer.alahmedy@gmail.com** **Phone: +20 102 399 1616** **Address: Cairo, New Cairo, Fifth Settlement, Region 1, District 2, 23.**	**Name: Karim Ibrahim Amin** **E-mail: karim.ibrahim.amin@gmail.com** **Phone: +20 111 109 9679** **Address: 13 Dar El Baydaa' st. Hawamdiyah, Giza.**
	Name: Mostafa Sherif Mourad **E-mail: mostafamsa1998@gmail.com** **Phone: +20 100 559 4862** **Address: 44 Mohy El Deen Abo El Ezz, Dokki, Giza**	**Name: Youssef Sayed Mostafa** **E-mail: youssefsayed20@gmail.com** **Phone: +20 122 570 5598** **Address: 3 Al Nile Al abyad st. Giza.**
Supervisor	**Name: Prof. Ahmed Darwish**	**E-mail: darwish@ieee.org**
	Signature:	**Phone: +20 122 224 7381**
Project Summary	**Eye Food is aimed to help Visually Impaired Individuals to be able to Recognize Food on their own. Visual Impaired individuals in Egypt have very few products that ease their everyday lives. Our product is hardware-optimized and ready to be implemented in smart glasses.**	

Abstract

Visually impaired people usually need the guidance of another individual to help them simply recognize a meal on a table. Specifically, in Egypt and the region, solutions directed to aid this community with this problem are very few and global solutions may not be helpful in recognizing oriental food. Our objective is to approach and tackle the problem by developing a solution that is optimized and ready to be implemented in a hardware smart glasses. Throughout this document we will discuss our approach used. Such a problem of detecting and recognizing the served food can be solved by applying state-of-the-art machine learning in computer vision whereas the hardware captures an image of a dining table as an input image, multiple plates are detected by another object detection model that leverages deep learning and the cropped images of the food are further passed in the pipeline to a classification module responsible for the classification of the food which includes oriental and local food categories. This approach involves gathering images of local food to create a dataset that can be used for training the classifier. The output of our product takes the form of an audio statement that informs the user of the food plate(s) on the dining table and their respective locations.

Testing the integrated pipeline on a raspberry pi 4 showed reasonable results with plenty of space for speed improvements. Accuracies compared to previous works in the field were excellent. Finally, this work is technically supported by *Valeo*.

الملخص

كثيرا ما يحتاج ضعاف البصر إلى من يرشدهم لمساعدتهم في معرفة الطعام على الطاولة أمامهم. خاصة في مصر والمنطقة ، الحلول الموجهة لمجابهة هذه المشكلة نادرة جدا ، كما أن الحلول لها حول العالم لا تساعد في التعرف على الأكل الشرقي. مهمتنا هي مواجهة هذه المشكلة من خلال تطوير حل محسن ومجهز ليعمل في نظارات ذكية. على مدار هذه الوثيقة سنناقش نهجنا المستخدم. مشكلة كالتحري والتعرف على الطعام المقدم يمكن حلها من خلال تطبيق تقنيات تعلم الآلة الحديثة في مجال رؤية الكمبيوتر. تلتقط المعدات صورة لطاولة الطعام ، يتم اكتشاف عدة أطباق من خلال نموذج للكشف عن الكائن والذي يستخدم التعلم العميق ، تاليا يتم إدخال الصور المكتشفة في وحدة التصنيف المسئولة عن تصنيف الطعام والذي يضم الطعام الشرقي والمحلي. هذا النهج ينطوي على تجميع صور للطعام المحلي لبناء مجموعة للبيانات والتي تستخدم لتدريب المصنف. مخرج المنتج عبارة عن جملة صوتية تعلم المستخدم بطsبيعة الطعام أمامه وأماكنه.

اختبار مراحل العمل على الرازبيري باي 4 أظهرت نتائج معقولة بكثير من التحسينات الممكنة. مقارنة بالأعمال السابقة في هذا المجال كانت الدقة ممتازة. ختاما ، كان هذا العمل مدعوما تقنيا من خلال *Valeo*.

ACKNOWLEDGMENT

Throughout the writing of this dissertation, we have received a great deal of support and assistance.

We would first like to thank our supervisor, Professor Ahmed Darwish, whose expertise was invaluable in formulating the research questions and methodology. Your insightful feedback, mentorship and patience pushed us to sharpen our thinking and brought our work to a higher level.

We would like to acknowledge all our extinguished professors in the computer engineering department. It’s the knowledge we gained that helped us the most in the last four years.

Finally, we could not have completed this dissertation without the support of our friends and families who provided stimulating discussions as well as happy distractions to rest our mind outside of our research.

Table of Contents

List of Figures

List of Tables

List of Abbreviation

ADAM	Adaptive Moment Estimation
AI	Artificial Intelligence
AP	Average Precision
API	Application Programming Interface
CNN	Convolutional Neural Network
CovNet	Convolutional Neural Network
FC	Fully Connected Layer
FDA	Food and Drug Association
FPS	Frames Per Second
GPU	Graphics Processing Unit
GUI	Graphical User Interface
MAC	Multiply Accumulate Operations
mAP	Mean Average Precision
POOL	Pooling Layer
RCNN	Region Based Convolutional Neural Networks
RELU	Relu Activation Function
RNN	Recurrent Neural Network
RPN	Region Proposal Network
RoI	Region of Interest
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
OCR	Optical Character Recognition
SVM	Support Vector Machine
ViT	Google’s Vision Transformer
WHO	World Health Organization

List of Symbols

ρ	Rho
θ	Theta

Contacts

Team Members

Name	Email	Phone Number
Amir Salah Al-Ahmedy AbdulRaheem	ameer.alahmedy@gmail.com	+2 01023991616
Karim Ibrahim Amin Wahba	karim.ibrahim.amin@gmail.com	+2 01111099679
Mostafa Sherif Ahmed Mourad	Mostafamsa1998@gmail.com	+20 1005594862
Youssef Sayed Mostafa Mohamed	youssefsayed20@gmail.com	+2 01225705598

Supervisor

Name	Email	Number
Dr. Ahmed Darwish	darwish@ieee.org	+20 1222247381

This page is left intentionally empty

Chapter 1: Introduction

The objective of the project is to have a prototype software module that is optimized to run on smart glasses/mobile to solve the problem of detecting the food plates and classifying the food inside for impaired vision people or who are totally blind.

1.1. Motivation and Justification

According to the World Health Organization W-H-O, there are more than 2.2 million people with visual impairment in Egypt 900,000 of which are totally blind. These people with visual disabilities living in Egypt still have hope in challenging their physical disabilities and living an independent life. The question here is “Will the status of visually disabled people improves?”. Our project main idea focuses on helping the visually disabled people to live a better and easier life to feel like a normal human living his life without any assistance from others

The first aspect of the project is to have a mechanism that can solve the problem of recognizing dishes(food) and their ingredients for visually impaired/totally blind. Despite the dozens of applications, algorithms and systems available the problem has not been fully addressed by the machine learning and computer vision communities specially in Egypt and the region. This is due to the lack of distinctive and spatial layout of food images, typically found in images depicting scenes or objects. For example, an image of an outdoor scene can be typically decomposed into (a) a ground place, (b) a horizon, (c) a forest, and (d) the sky. Patterns like these, cannot be found in food images. Food ingredients, like those found in a salad, are mixtures that frequently come in different shapes and sizes, depending much on regional practices and cultural habits. It should be highlighted that the nature of the dishes is often defined by the different colors, shapes and textures of the various ingredients.

Nevertheless, the kind of features that describe food in most cases are easily recognizable by humans in a single image, regardless of the geometrical variations of the ingredients. Hence, it could be considered that food recognition is a specific and a very complex classification problem demanding the development of models that are able to exploit local information (features) within images. Food recognition is one of the most important components in image-based dietary assessment. However, due to the different complexity level of food images and inter-class similarity of food categories, it is challenging for an image-based food recognition system to achieve high accuracy for a variety of publicly available datasets. In this work, we propose a new two-step food recognition system that includes food detection and hierarchical food classification using Convolutional Neural Networks (CNNs).

1.2. The Essential Question

The essential question is: Can we use computer vision techniques to recognize food photos. Specifically, how to build an optimized app that fits in a mobile's processor or a microprocessor/micro controller of a smart glasses that recognizes food with high accuracies with a reasonably low runtime? Our project revolves around three main questions; How can we help the visually impaired people to detect the food on the table? How can we inform them of the type of the food is in that dishes? How fast can we achieve both?

Our mission is to provide an optimized app that can run on glasses/mobile that helps in recognizing dishes and food in it. We are targeting methods and techniques that can detect food and classify it with high accuracy at a low latency. The project allows us to learn and apply the concepts of machine learning. It gives us a chance where we leverage what we have learnt and apply it to a real-life problem and get to experience the real engineering market orientation.

This project fits into the faculty's mission and vision respectively are:

"Achieve academic excellence to graduate competitive engineers, academically, professionally and ethically, capable of continuous learning, in line with international innovations and effective contribution to sustainable development in Egypt"

"Excellence and leadership in engineering education nationally, regionally and internationally to better serve individuals, society and environment."

1.3. Project Objectives and Problem Definition

Starting by restating what the main objective, helping out visually impaired individuals recognizing food. This can be practically translated into a couple of sub objectives. Firstly, a food recognition module. That can detect the number of plates and their positions on a dining table. Secondly, a food classification module with added support of oriental food. Through what we've learnt/ are learning, there are plenty of functional ways this can be done. Comparing and eliminating the least to the less performant approaches, we decided to go with the Deep Learning approach for classification for many reasons that will be further discussed in the documents. Thirdly, a hardware gadget that can run in almost-real time performance detecting the food the client/user is looking at. This is through running the aforementioned module on the chip/processor of the gadget. Fourthly, Integration of both modules and optimizations. Perhaps the hardest of the objectives. Managing to run and optimize a good reasonably accurate machine learning model to a relatively small processing power chip in an almost real time will be quite a challenge.

Food classification is a challenging problem due to the large number of food categories, since the visual similarity between different food categories is high. The goal is to build a model to predict the different food plates in one image including the fine-grained food-category. One of the goals of this proposed project is to contribute to the computer vision with a robust and effective food recognition system. However, the main motive remains at helping visually impaired people with recognizing the food.

Our system uses an assumption in order to solve the problem efficiently. This assumption is made based on the segments we have done surveying on. Namely, the assumption is:

- The user is most likely using the glasses to detect a plate he’s holding or a number of plates on a dining table.
- The dish is going to have one item of food only and this will also facilitate the eating process for the disabled man as he will be knowing what he is going to eat from each dish.
- The lights are fair enough to be detected by a 5MP raspberry pi camera module
- No two plates intersect. No plates are stacked.

Having the objectives and limitations clearly defined allows us to define the problem concretely.

1.4. Project Outcomes

The planned outcome is to create a light software module that is optimized enough and suitable for running on small devices such as a smart glass, a mobile phone, or a microprocessor like Raspberry Pi. This program will help the visually impaired people in detecting the number of food plates on a dining table and their relative positions. It wil allow them to detect the food type and it will include oriental food. This project to us is out first step in developing a fully functional smart glasses for visually impaired with plenty of features that can achieve independence for the user.

1.5. Document Organization

In this report, we present different aspects of our project. In addition to all the research we have conducted in the computer vision field specifically topics related to the food recognition problem. We also lay out distinctively chosen approaches for detection and classification of food. Moreover, we propose a simple prototype in addition to a basic software application.

The report is divided into the following chapters: Chapter 2 in which we describe our market visibility study by identifying the targeted customers and discussing our market survey. In Chapter 3, we illustrate our literature survey describing various techniques and approaches we inspected. Chapter 4 involves the complete system design and architecture. We describe in detail the two modules that constitute our project. Chapter 5 delves into testing and verification, in it we illustrate our obtained accuracies and compare it with previous work results. Finally, in chapter 6 we conclude our work and present the possible future work.

Chapter 2: Market Visibility Study

In this chapter, we will analyze the market, state the targeted customers, the market competitors and clarify our goals. This will make it easier for us to proceed knowing the struggles we might face and how others dealt with the same situation.

2.1. Targeted Customers

The intended customers of this project are blind or visually impaired people. With the aid of this food recognition application, they will be able to recognize the food served to them and find its position to reach for it.

According to the World Health Organization, there are more than 2.2 million people with visual impairment in Egypt. From those 2.2 million, 900,000 are totally blind.

This is also targeting the big companies which manufacture glasses that help the blind or visually impaired people. It will be a great feature to add to the current glasses, that we will see some of in the next section.

2.2. Market Survey

In this section, we review the market to grasp the competitors and the vendors providing similar solutions. This is done to visualize a market gap that can be filled and to have insights of the challenges we are going to encounter.

The first two competitors are glasses that help the blind or visually impaired people generally, then the next four are applications and services available for food recognition.

2.2.1. IrisVision

IrisVision electronic glasses are a highly innovative assistive technology solution, which is registered with the FDA (Food and Drug Administration) as a Class-1 medical device and is redefining the concept of wearable low vision aids. It is a combination a Samsung’s VR headset and a smartphone.

It comes with a lot of features including an advanced OCR program with text to speech capability, a personal virtual assistant and many other features to help you visualize the world better.

Figure 2.1 IrisVision Glasses

Price: $2,950

2.2.2. NuEyes Pro

NuEyes Pro -is a head-worn lightweight and wireless pair of smart glasses, which can be controlled either through a wireless handheld controller or a set of voice commands. A camera on the front of the glasses captures the image and displays it magnified inside of the lenses. You can get up to 12X magnified images. Bar code and QR scanning are one of the features also.

Price: $5,995

Figure 2.2 NuEyes Pro Glasses

2.2.3. Calorie Mama

An instant food recognition app. It is a smart camera app that uses deep learning to track nutrition from food images.

The application is mostly free (including the food recognition part), but if you want the premium features, you will have to pay 210 EGP per month or 630 EGP annually. The premium features include full access of the detailed nutrition information, custom home workout creator and meal plans.

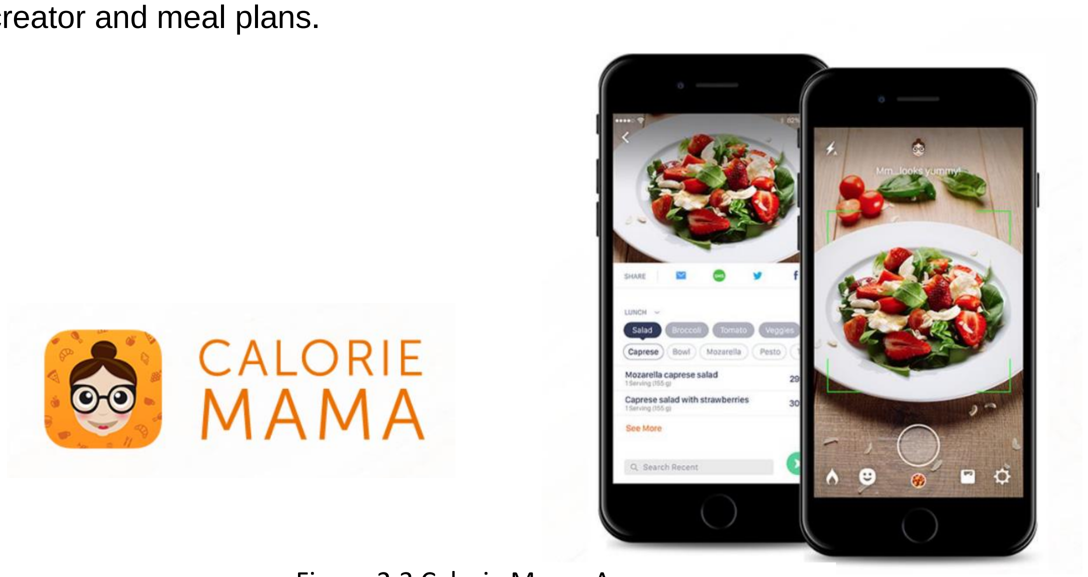


Figure 2.3 Calorie Mama App

2.2.4. Foodvisor

Foodvisor is a nutrition coach that will help you eat healthier and reach your goal. Just snap a picture of your meal and you will receive its nutrition facts.

It estimates the nutrition facts after asking you to enter the quantity of each food type.

If you want a nutrition program adapted to your needs and some other premium features, you will have to pay subscription fees 420 EGP for 3 months or 860 EGP for a year.

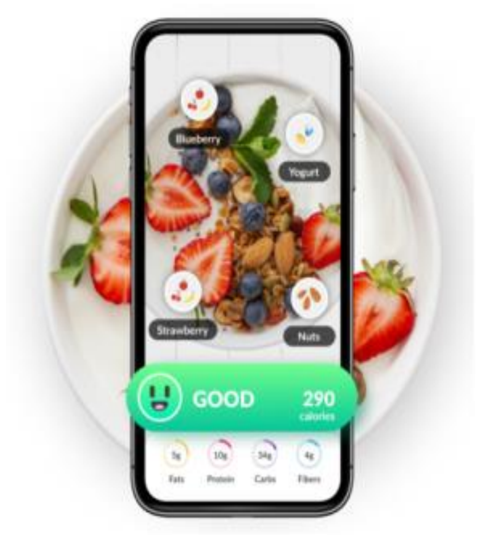


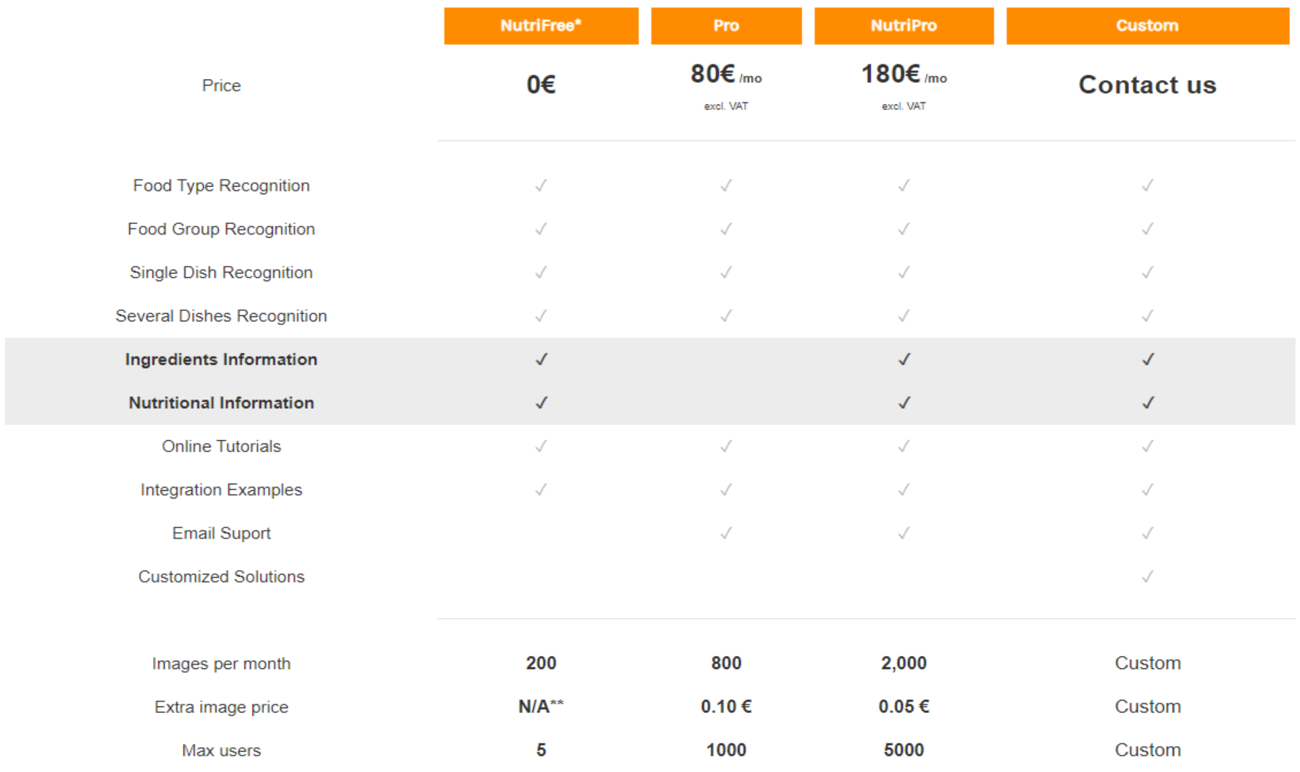
Figure 2.4 Foodvisor App

2.2.5. LogMeal

It is an advanced API for food AI and food tracking. LogMeal API offers a set of cloud-based Machine and Deep Learning algorithms for automated food detection.

It offers different services which are: food type detection, food group detection, single dish recognition, several dishes recognition, ingredients information, nutritional information and customized services.

The following image shows the prices of different packages per month.



	NutriFree*	Pro	NutriPro	Custom
Price	0€	80€ /mo excl. VAT	180€ /mo excl. VAT	Contact us
Food Type Recognition	✓	✓	✓	✓
Food Group Recognition	✓	✓	✓	✓
Single Dish Recognition	✓	✓	✓	✓
Several Dishes Recognition	✓	✓	✓	✓
Ingredients Information	✓		✓	✓
Nutritional Information	✓		✓	✓
Online Tutorials	✓	✓	✓	✓
Integration Examples	✓	✓	✓	✓
Email Suport		✓	✓	✓
Customized Solutions				✓
Images per month	200	800	2,000	Custom
Extra image price	N/A**	0.10 €	0.05 €	Custom
Max users	5	1000	5000	Custom

Figure 2.5 LogMeal Packages and Prices

2.2.6. FoodAI

Smart Food Recognition with the state-of-the-art Visual Recognition technology. It offers food image recognition technologies for advancing AI in food and healthcare, particularly for Singapore local food. It covers 756 visual food classes.

FoodAI™ is developed by the R&D team led by Prof Steven HOI at the School of Computing and Information Systems (SCIS), Singapore Management University (SMU), Singapore.

2.3. Business Case and Financial Analysis

Table 2.1 CapEX First Year

		Jan-21	Feb-21	Mar-21	Apr-21	May-21	Jun-21	Jul-21	Aug-21	Sep-21	Oct-21	Nov-21	Dec-21
CapEx													
	Laptops	700	700	700	700	700	700	700	700	700	700	700	700
	Development Tools	50	50	50	50	50	50	50	50	50	50	50	50
	Raspberry pi								13500	13500	13500	13500	13500
	Camera								1500	1500	1500	1500	1500
	Glasses Frame								250	250	250	250	250
	Assemble for production								100	100	100	100	100

Table 2.2 CapEx Second Year

		Jan-22	Feb-22	Mar-22	Apr-22	May-22	Jun-22	Jul-22	Aug-22	Sep-22	Oct-22	Nov-22	Dec-22
CapEx		700	700	700	700								
	Laptops	50	50	50	50								
	Development Tools	13500	13500	13500	13500	13500	13500	13500	13500	13500	13500		
	Raspberry pi	1500	1500	1500	1500	1500	1500	1500	1500	1500	1500		
	Camera	250	250	250	250	250	250	250	250	250	250		
	Glasses Frame	100	100	100	100	100	100	100	100	100	100		
	Assemble for production	700	700	700	700								

Table 2.3 OpEx First Year

		Jan-21	Feb-21	Mar-21	Apr-21	May-21	Jun-21	Jul-21	Aug-21	Sep-21	Oct-21	Nov-21	Dec-21
OpEx													
	Salaries								8000	8000	8000	10000	10000
	Contracts					800	800	800	800	800	800	800	800
	Rent				2500	2500	2500	2500	2500	2500	2500	2500	2500
	Bills			400	750	750	750	750	750	750	750	800	800
	Marketing				1000	1000	1000	1000	1000	1000	1000	1000	1000
	Sales								1000	1000	1000	1000	1000

Table 2.4 OpEx Second Year

		Jan-22	Feb-22	Mar-22	Apr-22	May-22	Jun-22	Jul-22	Aug-22	Sep-22	Oct-22	Nov-22	Dec-22
OpEx													
	Salaries	10000	10000	10000	10000	10000	15000	15000	15000	15000	15000	15000	15000
	Contracts	800	800	800	800	800	800	800	800	800	800	800	800
	Rent	2500	2500	2500	2500	2500	2500	2500	2500	2500	2500	2500	2500
	Bills	800	800	800	800	800	800	800	800	800	800	800	800
	Marketing	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
	Sales	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000

The next 2 tables are for the Subtotal for the **CapEx** and **OpEx:**

Table 2.5 CapEx and OpEx Subtotal First Year

		Jan-21	Feb-21	Mar-21	Apr-21	May-21	Jun-21	Jul-21	Aug-21	Sep-21	Oct-21	Nov-21	Dec-21
SubTotal		750	750	1150	5000	5800	5800	5800	30150	30150	30150	32200	32200
Accumulative		750	1500	2650	7650	13450	19250	25050	55200	85350	115500	147700	179900

Table 2.6 CapEx and Opex Subtotal Second Year

		Jan-22	Feb-22	Mar-22	Apr-22	May-22	Jun-22	Jul-22	Aug-22	Sep-22	Oct-22	Nov-22	Dec-22
SubTotal		16100	16100	16100	16100	16100	21100	21100	21100	21100	36450	21100	21100
Accumulative		196000	212100	228200	244300	260400	281500	302600	323700	344800	381250	402350	423450

Table 2.7 Revenue First Year

		Jan-21	Feb-21	Mar-21	Apr-21	May-21	Jun-21	Jul-21	Aug-21	Sep-21	Oct-21	Nov-21	Dec-21
Revenue													
	Unit Sales								9000	9000	9000	13500	13500
	Sponsors										2500	2500	2500
SubTotal		0	0	0	0	0	0	0	9000	9000	11500	16000	16000
Accumulative		0	0	0	0	0	0	0	9000	18000	29500	45500	61500

Table 2.8 Revenue Second Year

		Jan-22	Feb-22	Mar-22	Apr-22	May-22	Jun-22	Jul-22	Aug-22	Sep-22	Oct-22	Nov-22	Dec-22
Reveneu		13500	22500	22500	36000	36000	36000	31500	31500	31500	31500	22500	22500
	Unit Sales	2500	2500	2500	2500	2500	2500	2500	2500	2500	5000	5000	5000
	Sponsors												
		16000	25000	25000	38500	38500	38500	34000	34000	34000	36500	27500	27500
SubTotal		77500	102500	127500	166000	204500	243000	277000	311000	345000	381500	409000	436500
Accumulative		13500	22500	22500	36000	36000	36000	31500	31500	31500	31500	22500	22500

Table 2.9 Profit First Year

		Jan-21	Feb-21	Mar-21	Apr-21	May-21	Jun-21	Jul-21	Aug-21	Sep-21	Oct-21	Nov-21	Dec-21
Profit		(750)	(1500)	(2650)	(7650)	(13450)	(19250)	(25050)	(46200)	(67350)	(86000)	(102200)	(118400)
Taxes													
Net Profit													

Table 2.10 Profit Second Year

		Jan-22	Feb-22	Mar-22	Apr-22	May-22	Jun-22	Jul-22	Aug-22	Sep-22	Oct-22	Nov-22	Dec-22
Profit		(118500)	(109600)	(100700)	(78300)	(55900)	(38500)	(25600)	(12700)	200	250	6650	13050
Taxes										28			
Net Profit										172			

2.3.1. Business Case

Based on the above market survey and after analyzing the market, we have deduced the price of our product to be 4500 Egyptian pounds. Moreover, we will be producing 70 units to reach our break-even point by September 2022 and after this date we will be having the Net profit for our project.

2.3.2. Financial Analysis

All the cash flow table is calculated in **Egyptian pound**, it is calculated until the break-even point in September 2022 (After 19 Months from beginning of our project) based on 70 units sold, then we will start calculating profits.

2.3.2.1. Capital Expenditure

After we have worked on the implemented approach, our target for the capital expenditure is that producing 70 units of our product to reach the break-even point. Also, given that we started the development in Jan 2021. We've been working on the software development for a few months. Then after we are done with implementation, we are going to produce the hardware in August 2021 and the expenses for the laptops and its depreciation will be divided to end by April 2022, so all of these expenses are including in the first two tables.

2.3.2.2. Operational Expenses:

For the first 3 months we were working from home so we didn't need rent and the salaries weren't peaking as it's a growing idea and a startup. As we don't have a high capital, our expected launch in the market is in in August 2021, we will start to get some revenue in August so we will be hiring and giving out salaries and contracts to people who will be working in assembling our product as well as the cost of running the development team. This will definitely create an upwards trend in the bills.

Chapter 3: Literature Survey

In this section we will discuss the necessary background for different machine learning models used in the image classification and detection problems. There are basically three main popular approaches that Classification problems rely on; namely, Convolutional Neural Networks, Vision Transformers and Feature Engineering. In the following we present each in some details. We follow that by giving necessary background about Object Detection Approaches. We end this chapter by a Comparative Study of Previous Work and justification of why we chose the Implemented Approach.

3.1. Convolutional Neural Networks

Convolutional Neural Networks. Abbreviated as CNNs or sometimes CovNets, is a multi-layered neural network designed to complex features of the data at each layer to correctly anticipate the classification. Before we get into in depth details of how they work, let's revise a core concept in CNNs which is convolution.

3.1.1. Convolution

The convolution of two function f(t) and g(t) is:

$$[f * g](t) \equiv \int_0^t f(\tau)\, g(t-\tau)\, d\tau,$$

Which is basically the multiplication then summation or integration.

In CNNs we use a simple convolution between the matrices. Let's say we have a 3x3 matrix and a 5x5 one. A simple convolution would be:

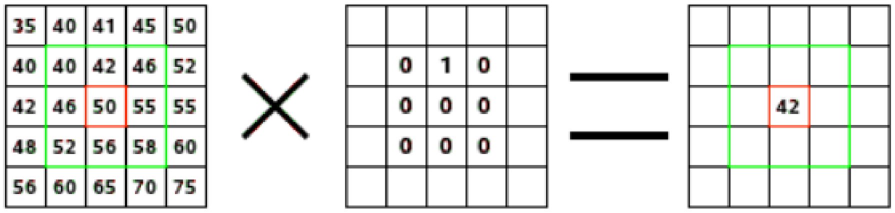


Figure 3.1 Convolution Operation

However, this is only one frame of what’s happening behind the scenes. This smaller matrix has to be shifted all across the 5x5 matrix. Notice that the resulting matrix, as expected, is of a smaller size.

3.1.2. Convolutional Neural Network Architecture

A simple Convolutional Neural Network architecture would look like:

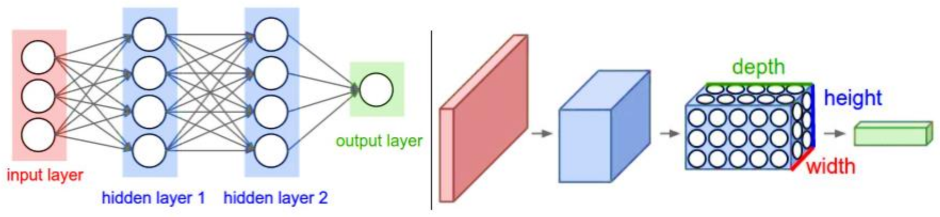


Figure 3.2 CNN Simple Example

Digging Deeper, let’s call the 3x3 matrix a filter and the 5x5 matrix the image. This is simply what happens inside a convolutional layer in a neural network. Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. (Note that the word depth here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.) For example, the input images in the famous dataset CIFAR-10 are an input volume of activations, and the volume has dimensions 32x32x3 (width, height, depth respectively). The neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would for CIFAR-10 have dimensions 1x1x10, because by the end of the ConvNet architecture we will reduce the full image into a single vector of class scores, arranged along the depth dimension.

An architecture for a convolutional Neural Net would typically follow the following formula:

```
INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
```

Hence, we use three main types of layers to build Convolutional Network architectures: **Convolutional Layer** [CONV], **Pooling Layer** [POOL], and **Fully-Connected Layer** [FC] (exactly as seen in regular Neural Networks). We will stack these layers to form a full Convolutional Network architecture.

3.1.3. Convolutional Layer

Now let’s dig deeper, the input for a CNN is called a tensor. A tensor is a multidimensional array containing data for the validation of training of the CNN. It can be represented by (d,h,w,c). Where d is the depth of the tensor i.e., the number of images, h and w are the height and the weight respectively and c is the number of channels in an image. The feature detector (filter) is a small array with fixed values. Convolution of this array with the input tensor reveals visual features. The kernel moves with specific steps (stride) to cover the entire surface of the image. The output of the convolution is named feature map and is representing the features extracted by the filter from the input. The output depends mainly on the input, kernel size and stride. Here’s a figure showing a simple convolutional layer input and output.

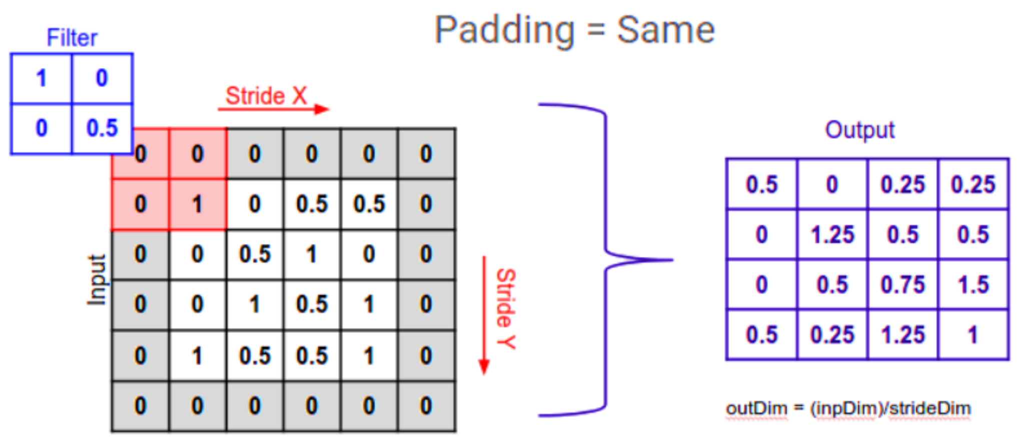


Figure 3.3 Convolution Example

Accepts a volume of size ***W1×H1×D1***
Produces a volume of size ***W2×H2×D2W2×H2×D2*** where:

W2=(W1−F+2P)/S+1W2=(W1−F+2P)/S+1

H2=(H1−F+2P)/S+1H2=(H1−F+2P)/S+1

D2=K

3.1.4. Pooling Layer

To understand what a pooling layer does we need to understand a very small limitation in a convolutional layer. A limitation of the feature map output of convolutional layers is that they record the precise position of features in the input. This means that small movements in the position of the feature in the input image will result in a different feature map. This can happen with re-cropping, rotation, shifting, and other minor changes to the input image.

A common approach to addressing this problem from signal processing is called down sampling. This is where a lower resolution version of an input signal is created that still contains the large or important structural elements, without the fine detail that may not be as useful to the task. Down sampling can be achieved with convolutional layers by changing the stride of the convolution across the image.

A more robust and common approach is to use a pooling layer. A pooling layer is a layer added after the convolutional layer. Specifically, after a nonlinearity (e.g., ReLU) has been applied to the feature maps output by a convolutional layer. A schematic representation of convolutional-and pooling layer in a CNN would look very similar to this:

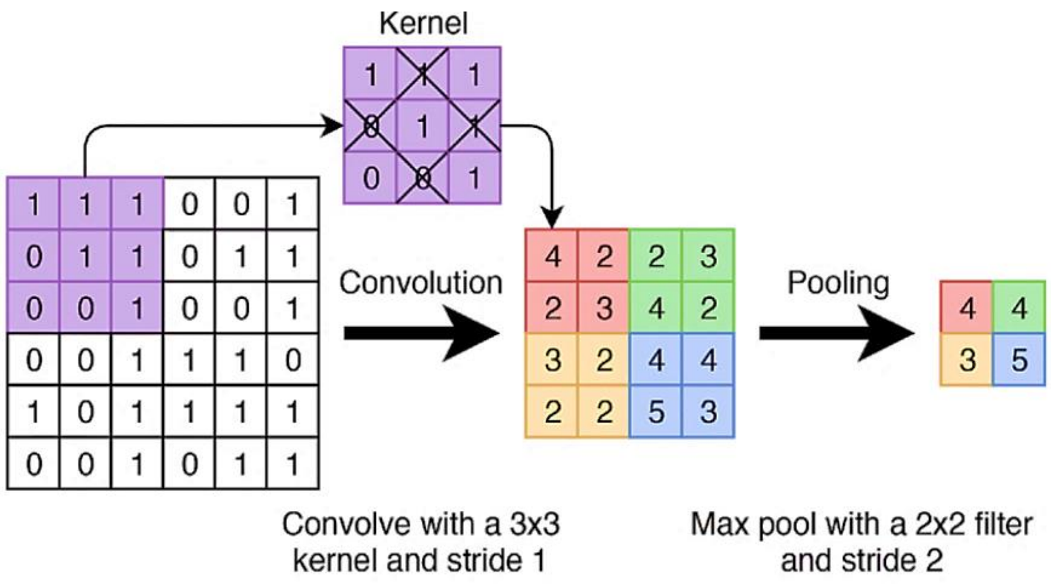

Kernel
Convolution
Pooling
Convolve with a 3x3 kernel and stride 1
Max pool with a 2x2 filter and stride 2

Figure 3.4 Convolution with Different Strides

3.1.5. Fully-Connected Layer

A fully Connected Layer is simply a multi-layer perceptron. The intuition behind a neural network comes from our system of neurons. It is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. An artificial neuron that receives a signal then processes it and can signal neurons connected to it. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. A typical one hidden layer architecture would look like:

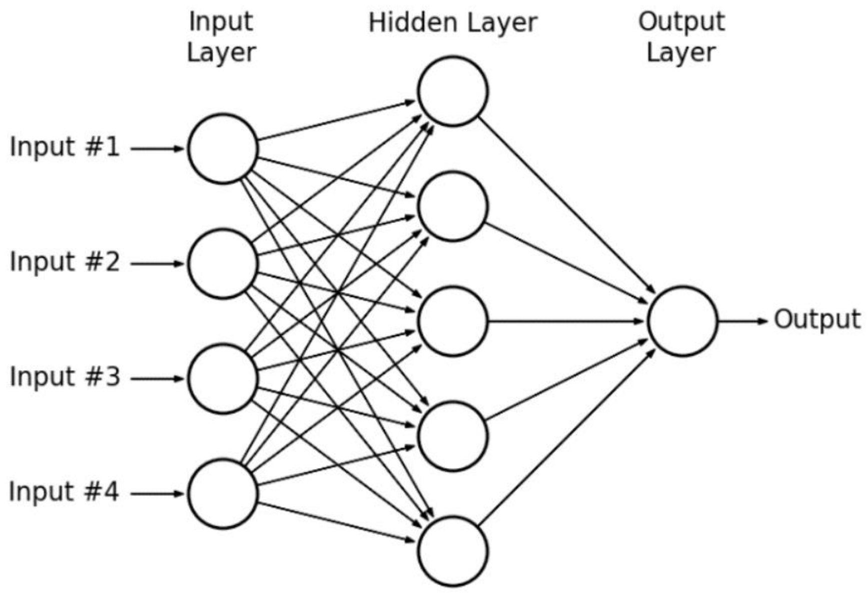


Figure 3.5 Fully Connected Layer

In the hidden layer, the result of the input multiplication with the weights is added to a bias term and then passed to the activation function. The result of the activation function is the input used to predict the output.

In the context of deep learning, Neurons have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

Fully-connected layers come in an architecture, In the context of computer vision and convolutional neural networks, as a top layer for classification. In other conventional classification algorithms, like SVMs, we used to extract features from the data to make the classification work. The convolutional layers are serving the same purpose of feature extraction. CNNs capture better representation of data and hence we don’t need to do feature engineering. After feature extraction we need to classify the data into various classes, this can be done using a fully connected neural network. In place of fully connected layers, we can also use a conventional classifier like SVM. But we generally end up adding FC layers to make the model end-to-end trainable. The fully connected layers learn a (possibly non-linear) function between the high- level features given as an output from the convolutional layers. Hence, in most popular machine learning models, the last few layers are full connected layers which compiles the data extracted by previous layers to form the final output. It is the second most time consuming layer second to Convolution Layer.

It is worth noting that the only difference between FC and CONV layers is that the neurons in the CONV layer are connected only to a local region in the input, and that many of the neurons in a CONV volume share parameters. However, the neurons in both layers still compute dot products, so their functional form is identical. This arouses an important question, if fully connected layer can basically do what a convolutional layer does then why aren’t all layers fully connected? The answer is simply: Fully connected layers can be seen as a brute force approach whereas there are approaches like the convolutional layer which reduces the input to concerned features only. It would be computationally very expensive.

3.2. Vision Transformers

Transformers in the computer vision is the new “state of the art” replacing the convolutional neural networks era. In this subsection we will be introducing all the required knowledge for grasping the main ideas of how a transformer works on images. Firstly, let us remember the concept of attention.

3.2.1. Attention

In NLP, transformers and attention have been utilized successfully in a plethora of tasks including reading comprehension, abstractive summarization, word completion, and others. Attention became popular in the general task of dealing with sequences.

The attention was first introduced in the paper “Neural Machine Translation by Jointly Learning to Align and Translate” [1]. What attention basically does is that it allows the model to focus on the relevant parts of the input sequence as needed. It does so by a scoring exercise. Let there be a sequence-to-sequence model with attention for the task of text translation from French to English, the architecture would look similar to this:

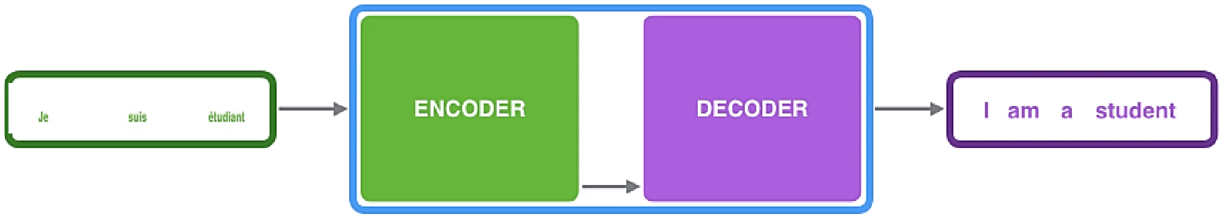


Figure 3.6 Sequence to Sequence Translation Model

Briefly, the encoder and the decoder are usually Recurrent Neural Networks that maintain a hidden state. The encoder outs the hidden states vectors to the decoder (The arrow between encoder and decoder) and hence the attention mechanism takes the following order to execute:

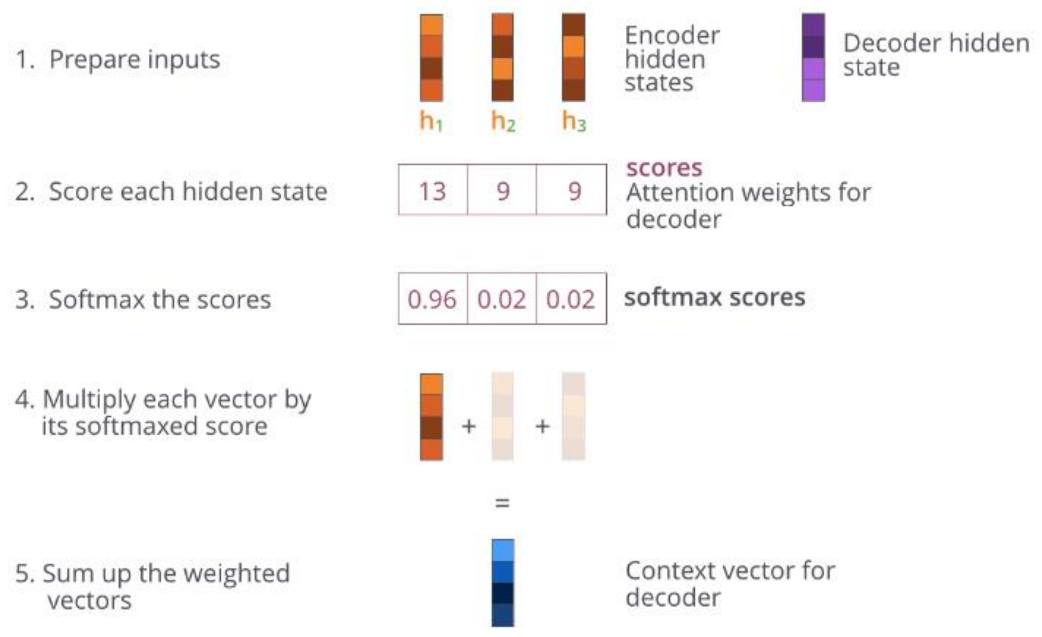

1. Prepare inputs
h1
h2
h3
Encoder hidden states
Decoder hidden state
2. Score each hidden state
13
9
9
scores
Attention weights for decoder
3. Softmax the scores
0.96
0.02
0.02
softmax scores
4. Multiply each vector by its softmaxed score
+
+
=
5. Sum up the weighted vectors
Context vector for decoder

Figure 3.7 Attention Mechanism

This scoring exercise is done at each time step on the decoder side. The output context vector is the scored output of the attention which indicates which words in English attends to which words in French.

Next, we'll be discussing a transformer model and how it utilizes the attention mechanisms stated here.

3.2.2. Transformers

Transformers mainly rely on the discussed technique called attention. The main two differences between attention and convolution are that convolution operates on a fixed sized window and has fixed weights, while it's not the case with attention. Comprehending the concepts of the transformers will be easier following the machine translation example proposed in the attention subsection.

Similar to a typical sequence model, transformers are constructed of an encoder and a decoder blocks. In transformers, usually the encoder/decoder block will consist of many encoders/decoders components that are identical but not always share weights. A single encoder to decoder architecture will look similar to this:

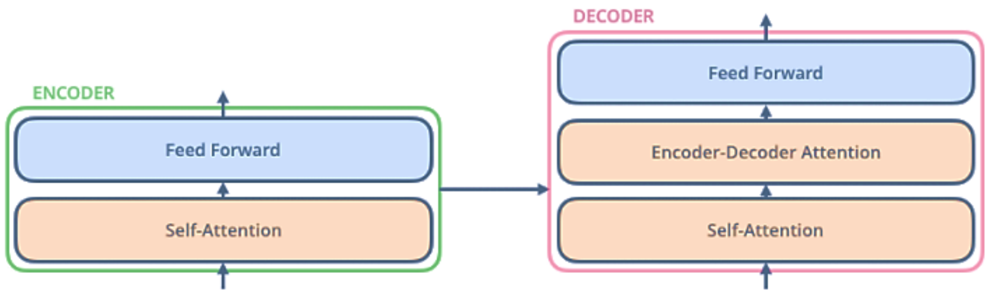


Figure 3.8 Transformer Encoder Decoder

The encoder's inputs first flow through a self-attention layer – a layer that helps the encoder look at other words in the input sentence as it encodes a specific word. The outputs of the self-attention layer are fed to a feed-forward neural network (Fully connected network). The exact same feed-forward network is independently applied to each position. The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence (similar what attention does in sequence to sequence models discussed above).

The Self-Attention in the encoder is the method the Transformer uses to bake the "understanding" of other relevant words into the one we're currently processing, similar to how maintaining a hidden state allows an RNN to incorporate its representation of previous words/vectors it has processed with the current one it's processing. For illustration, let us say for example, we have a sentence to translate: "The animal didn't cross the street because it was too tired". A self-attention mechanism would map the word "it" to the following words.

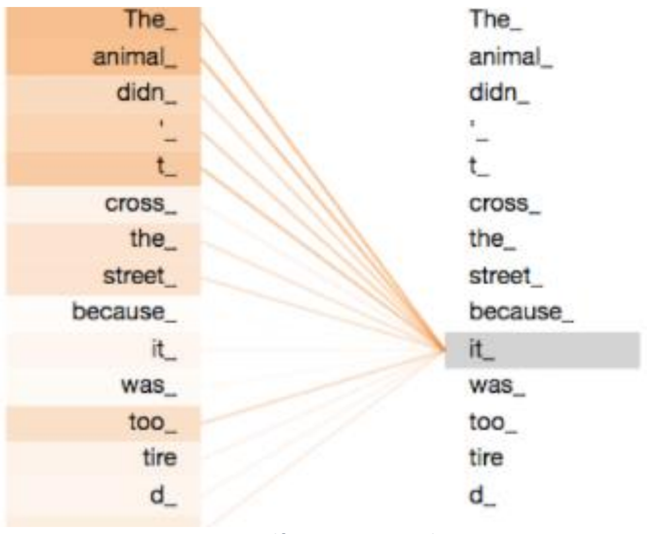


Figure 3.9 Self Attention Mechanism

Throughout this subsection we've been illustrating on text translation examples for delivering concepts easily. It's about time to take what we have learned so far and apply it to an image. In vision transformers, an architecture will usually consist of a mid –scale deep convolutional neural network that acts as a backbone for feature extraction and outputs a tensor which is then introduced to the transformer as inputs. Other architectures would propose dividing the image into a grid of 3x3,4x4,16x16..etc images and directly introduce them to the transformers.

Here is an example of self-attention applied on image. Let us say we have and image that was divided to small 2x4 regions. The small matrix, we will call it X, is introduced to the encoder layer in the transformer and encounters the self-attention block.

The first step is to calculate the Query, Key, and Value matrices. We do that by multiplying X by the trainable weights matrices (WQ, WK, WV).

Next, we multiply the calculated Query and Key matrices and divide them by the square root of the dimensions of K.

Passing the results to a Softmax normalizes the scores so they're all positive and add up to 1. Then, we multiply each Value matrix by the softmax score (in preparation to sum them up). The intuition here is to keep intact the values of the images we want to focus on, and drown-out irrelevant ones.

Last step is to sum up the weighted value tensors. This produces the output of the self-attention layer at this position (for the first image).

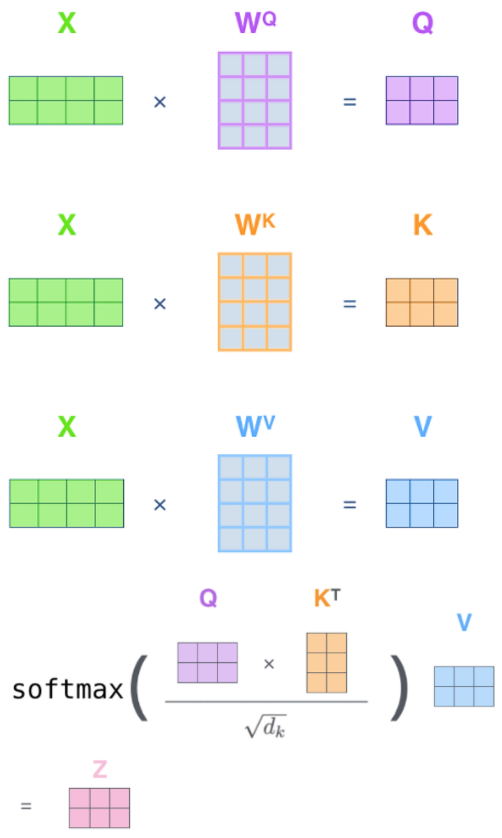


Figure 3.10 Self Attention Calculation

To sum up, Attention alleviates the shortcomings of convolution by having changing (dynamic) weights, and operating on variable sized windows. In addition, CNNs are trivial to parallelize and exploit local dependencies. These are one of many comparisons that transformers will ace. Nevertheless, transformers few limitations can be bottlenecking. They are usually data hungry to train and take far long than any CNN, which let us anticipate that CNNs would perform better on low to mid-scale datasets.

3.3. Classical Feature Engineering

An essential step in solving any object classification task is to represent the visual information of the object. This is commonly known as feature extraction. Features need to be robust to image perturbations, such as viewpoint and illumination changes. Given the labeled pixels of an image after segmentation, we need to extract visual characteristics from each region (food item) to help train the classifier to identify each food item. Therefore, proper selection of features is the key to correct classification.

Certain criteria are necessary when selecting the features: the features should contain enough information to uniquely describe each food item yet not be domain-specific; they should be easy to compute and relate well with the human visual system. Based on these criteria, four types of features can be extracted for each segmented food region, which are: color, size, shape and texture.

3.3.1. Color Image Quantization

Color image quantization (clustering) is a process that reduces the number of distinct colors used in an image, usually with the intention that the new image should be as visually similar as possible to the original image. This figure shows colorful images reduced to four colors by clustering.

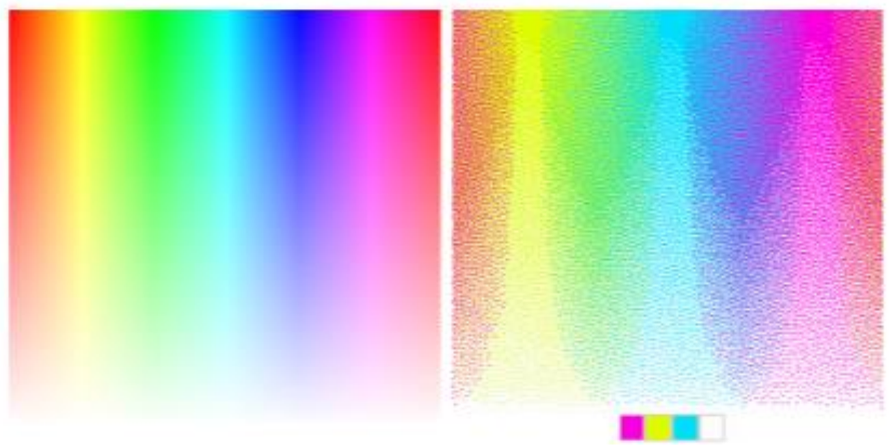


Figure 3.11 Color Clustering

For color feature extraction, color mean shift segmentation followed by color-clustering can be applied to the image.

3.3.2. Texture Feature Extraction

Texture provides information in the spatial arrangement of colors or intensities in an image. Texture is characterized by the spatial distribution of intensity levels in a neighborhood. It is basically a repeating pattern of local variations in image intensity and so it cannot be defined for a pixel/point. Upon the methods used for texture analysis and feature extraction is Gabor filters**.**

3.3.2.1. Gabor Filters

In the spatial domain, a 2-D Gabor filter is a Gaussian kernel function modulated by a sinusoidal plane wave. It essentially analyzes whether there is any specific frequency content in the image in specific directions in a localized region around the point or region of analysis.

Frequency and orientation representations of Gabor filters are claimed by many contemporary vision scientists to be similar to those of the human visual system. They have been found to be particularly appropriate for texture representation and discrimination.

This figure shows a satellite image of the map of Valencia in Italy before and after applying the Gabor filters. [2]

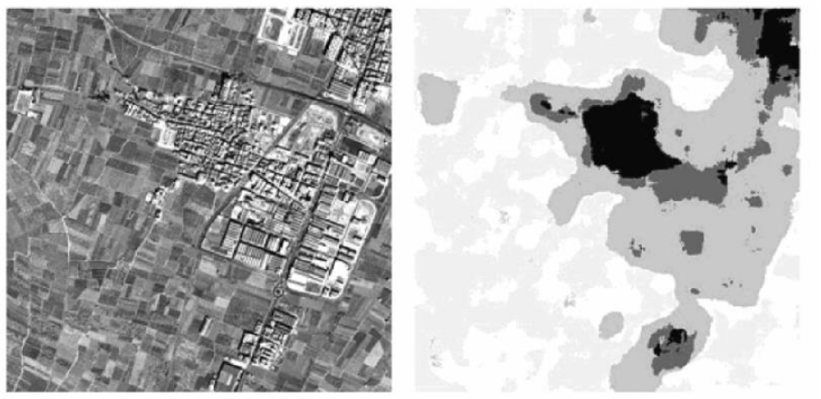


Figure 3.12 Gabor Filter Example

3.3.3. Shape Feature Extraction

Shape feature learning methods extract individual shape features from images. Shape features are the features of objects constructed by a set of elements like points, lines and curves. These features can be corner features or edge features which can be detected by feature detection methods. Edge detection and counter definition can be engaged to find the shape of food portions.

3.3.3.1. Hough Transform

Among the very famous line and shape detection techniques is the Hough transform for ellipse detection [3]. The Hough transform in its simplest form is a method to detect straight lines but it can also be used to detect circles or ellipses.

For the Hough Transform algorithm, it is crucial to perform edge detection first to produce an edge image which will then be used as input into the algorithm. An edge detection algorithm detects edges in an image by determining where the brightness/intensity of an image changes drastically. Examples of edge detection algorithms are Canny [4] and Sobel [5].

The algorithm assumes that the edge is detected and it is robust against noise or missing points.

The algorithm represents a straight line by a line called the normal line that passes through the origin and perpendicular to that straight line. The form of the normal line is ρ = x cos(θ) + y sin(θ) where ρ is the length of the normal line and θ is the angle between the normal line and the x axis.

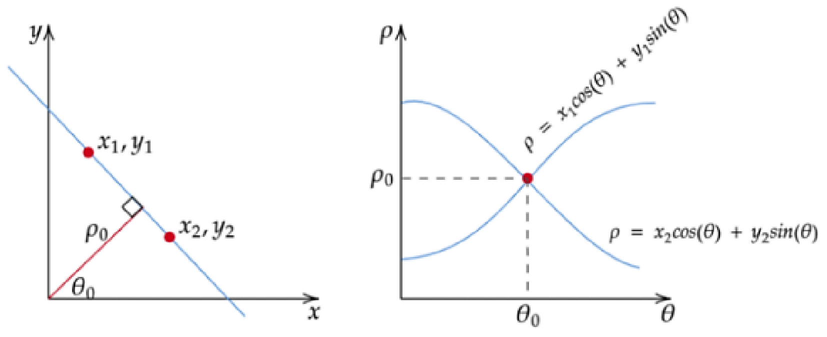

y
x_1, y_1
x_2, y_2
ρ_0
θ_0
x
ρ
$\rho = x_1 cos(\theta) + y_1 sin(\theta)$
ρ_0
$\rho = x_2 cos(\theta) + y_2 sin(\theta)$
θ_0
θ

Figure 3.13 Hough Space

Hence, mapping all the edge points from an edge image onto the Hough Space will generate a lot of cosine curves. If two edge points lay on the same line, their corresponding cosine curves will intersect each other on a specific (ρ, θ) pair. Thus, the Hough Transform algorithm detects lines by finding the (ρ, θ) pairs that has a number of intersections larger than a certain threshold.

A small modification can be used for ellipse detection. The algorithm takes two different points belonging to the ellipse. It assumes that it is the main axis. A loop on all the other points determines how much an ellipse passes to them. A good match corresponds to high accumulator values. The figure shows the Hough transform applied on a cup of coffee.

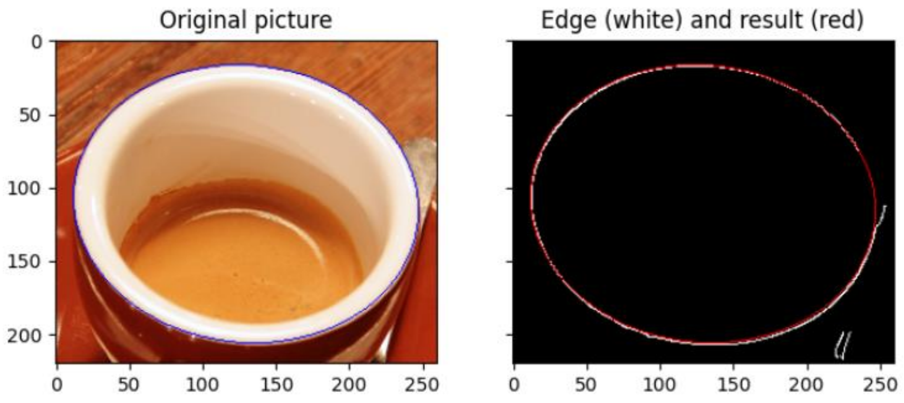


Figure 3.14 Hough Transform Example

3.4. Object Detection Networks and Metrics

3.4.1. Precision and Recall

Precision measures how accurate your predictions are i.e., the percentage of your predictions that are correct. Recall measures how good you find all the positives.

3.4.2. IoU

Intersection over union measures the overlap between two boundaries. It is used to measure how much the predicted boundary overlaps with the ground truth.

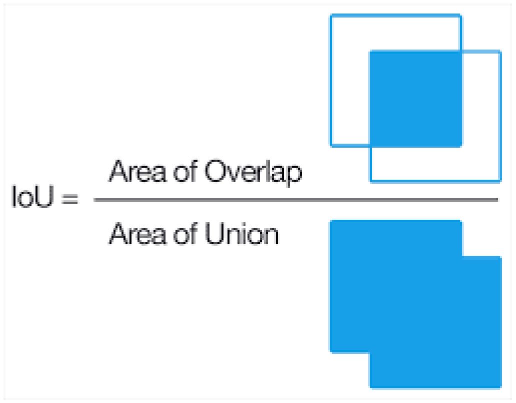


Figure 3.15 IoU Example

3.4.3. Average Precision

The general definition for the average precision (AP) is finding the area under the precision-recall curve. Precision and recall are always between 0 and 1. Therefore, AP falls within 0 and 1 also. Average precision computes the average precision value for recall value over 0 to 1.

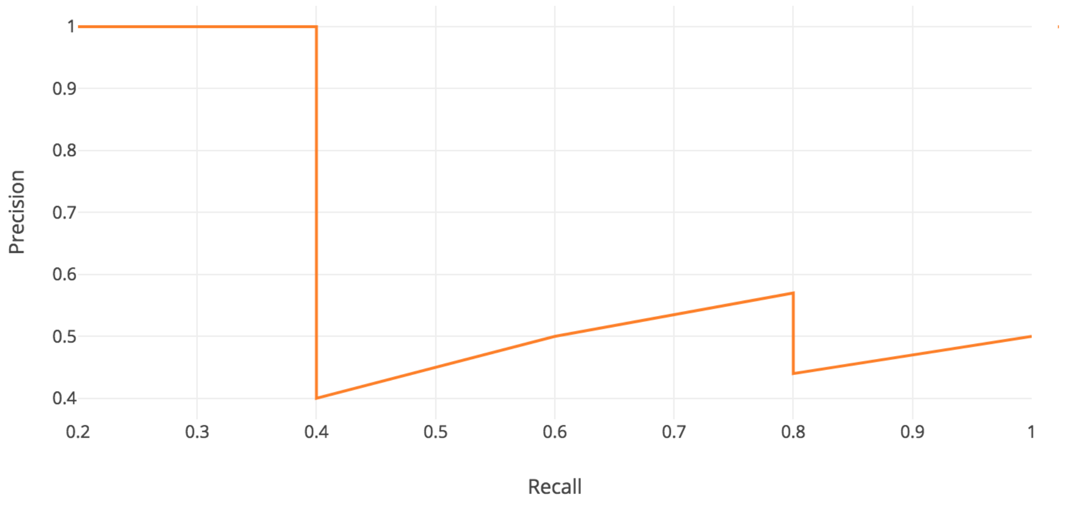


Figure 3.16 Precision-Recall Curve

$$AP@\alpha = \int_0^1 p(r)\,dr$$

3.4.4. Mean Average Precision (mAp)

$$mAP = \frac{1}{n}\sum_{k=1}^{k=n} AP_k$$
$$AP_k = \text{the AP of class } k$$
$$n = \text{the number of classes}$$

The mean Average Precision or mAP score is calculated by taking the mean AP over all classes and/or overall IoU thresholds, depending on different detection challenges that exist.

3.4.5. RCNN

Region proposal methods and region-based convolutional neural networks (R-CNNs) drive the recent advances in object detection. Although region-based CNNs were computationally expensive their cost has been drastically reduced thanks to sharing convolutions across proposals.

R-CNN was proposed by Ross Girshick et al. in 2014 [6] to deal with the problem of efficient object localization in object detection. Previous methods used exhaustive search techniques which used sliding windows of different scales on image to propose region proposals Instead, this paper uses the Selective search algorithm which takes advantage of segmentation of objects and Exhaustive search to efficiently determine the region proposals.

This selective search algorithm proposes approximately 2000 region proposals per image.

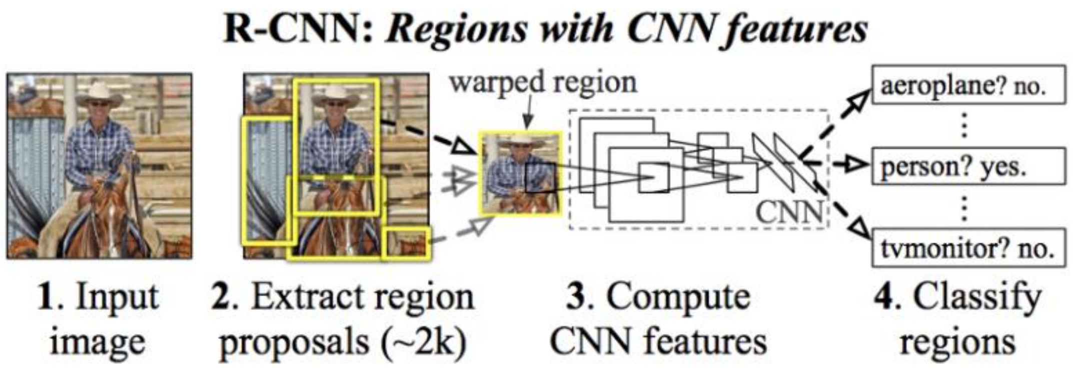


Figure 3.17 RCNN [6]

Problems with the RCNN is that it is slow that it takes about 49 seconds to detect objects in an image on GPU. In addition to that, each image needs to classify 2000 region proposals. So, it takes a lot of time to train the network. Moreover, lots of disk space is required to store the feature maps.

3.4.6. Fast RCNN

In R-CNN we passed every region proposal individually in the CNN architecture and selective search produced around 2000 region proposals for a picture. Along these lines, it is computationally costly to prepare and even test the picture utilizing R-CNN.

To manage this issue Fast R-CNN was proposed [7], it takes the entire picture and region proposals as contribution to its CNN architecture in one forward proliferation. It additionally consolidates various pieces of architecture, (for example, ConvNet, RoI pooling, and arrangement layer) in one complete architecture. That additionally eliminates the necessity to store a feature map and saves disk space. It likewise utilizes the softmax layer rather than SVM in its classification of region proposals which end up being quicker and produce preferred accuracy over SVM.

The architecture of the fast RCNN looks as follows:

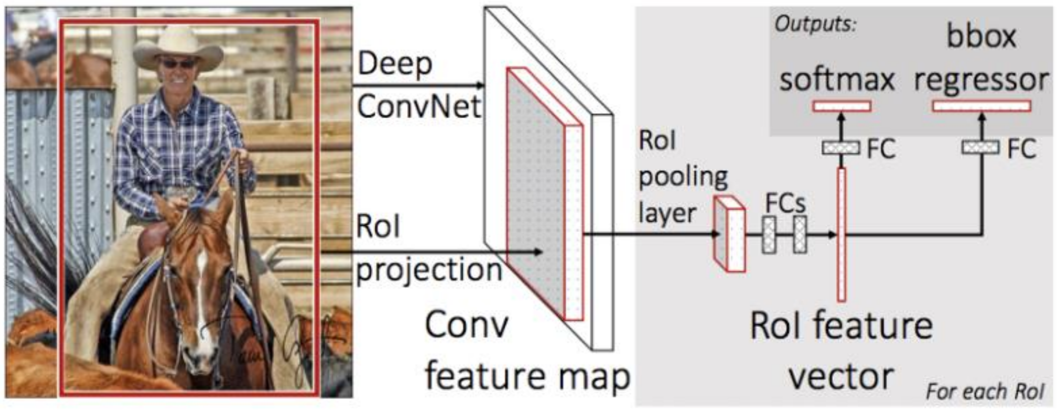


Figure 3.18 Fast RCNN [7]

3.4.7. Faster RCNN

Both of the above algorithms (R-CNN & Fast R-CNN) uses selective search to find out the region proposals. Selective search is a slow and time-consuming process affecting the performance of the network. Therefore, Shaoqing Ren et al. came up with an object detection algorithm that eliminates the selective search algorithm and lets the network learn the region proposals.

Similar to Fast R-CNN, the image is provided as an input to a convolutional network which provides a convolutional feature map. Instead of using selective search algorithm on the feature map to identify the region proposals, a separate network is used to predict the region proposals. The predicted region proposals are then reshaped using a RoI pooling layer which is then used to classify the image within the proposed region and predict the offset values for the bounding boxes.

The figure shows the architecture of the Faster R-CNN. [8]

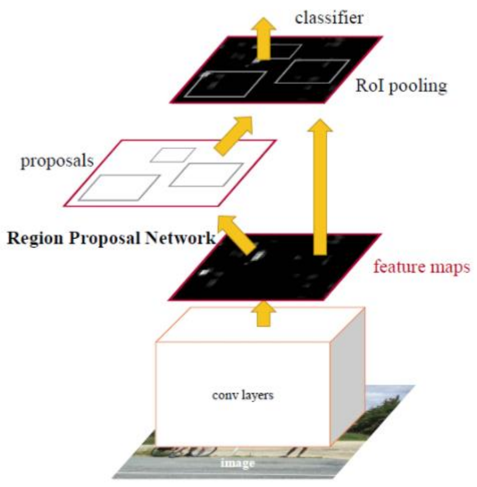


Figure 3.19 Faster RCNN [8]

3.4.7.1. Region Proposal Network (RPN)

The goal of RPN is to output a set of proposals, each of which has a score of its probability of being an object and also the class/label of the object. RPN can take any sized input to achieve this task. These proposals are further refined by feeding to 2 siblings fully connected layers-one for bounding box regression and the other for box classification (i.e. is the object foreground or background.)

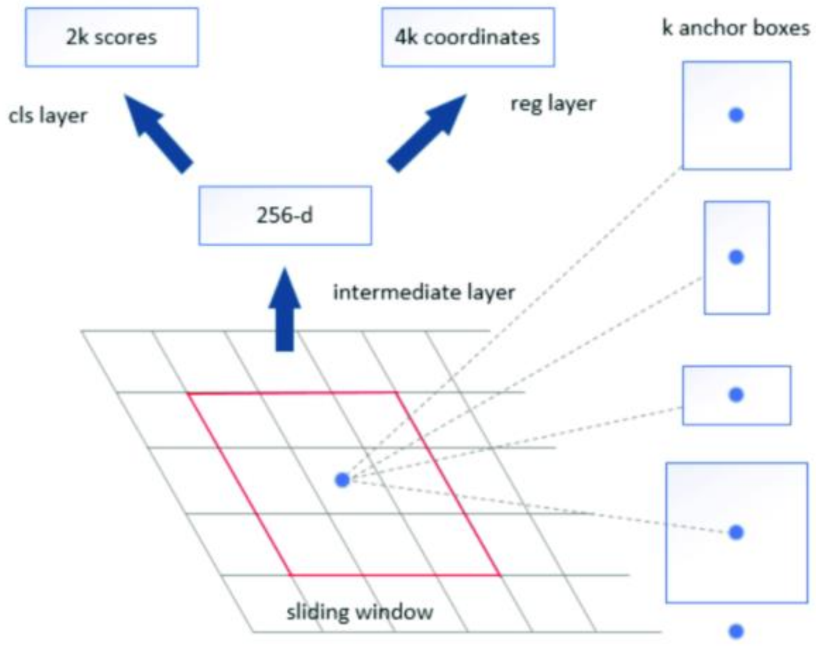


Figure 3.20 Region Proposal Network [8]

The RPN that generates the proposals slide a small network over the output of the last layer of the feature map. There are 2k classification scores and 4k box coordinates generated by RPN where k is the maximum possible number of region proposals.

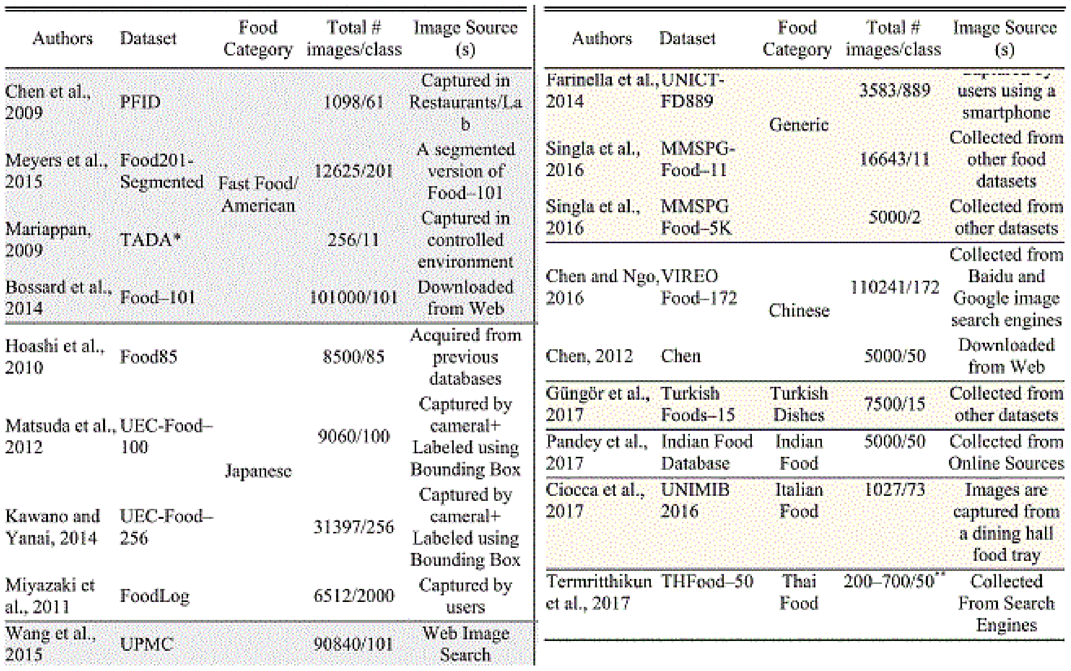
3.5. Comparative Study of Previous Work

In this section we will give a comparative, classified short literature review of the latest publications related to our project within the recent years. During our time spent researching the different approaches we came across a numerous number of papers for the datasets and for the techniques and approaches used.

3.5.1. Datasets

We will start by comparing the different food datasets. The figure below shows a comparison between the different datasets in terms of the food categories, total number of images/class and the image sources.

Table 3.1 Datasets Comparison [9]



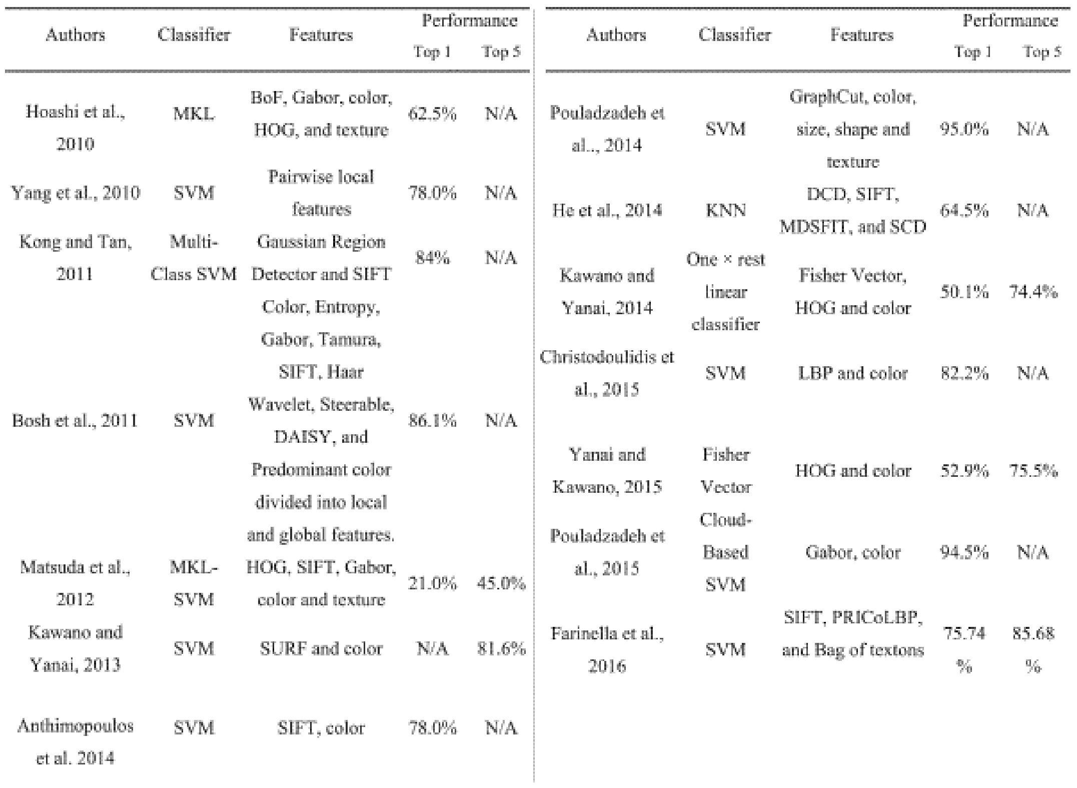
Authors	Dataset	Food Category	Total # images/class	Image Source (s)
Chen et al., 2009	PFID	Fast Food/ American	1098/61	Captured in Restaurants/Lab
Meyers et al., 2015	Food201-Segmented		12625/201	A segmented version of Food–101
Mariappan, 2009	TADA*		256/11	Captured in controlled environment
Bossard et al., 2014	Food–101		101000/101	Downloaded from Web
Hoashi et al., 2010	Food85	Japanese	8500/85	Acquired from previous databases
Matsuda et al., 2012	UEC-Food–100		9060/100	Captured by cameral+ Labeled using Bounding Box
Kawano and Yanai, 2014	UEC-Food–256		31397/256	Captured by cameral+ Labeled using Bounding Box
Miyazaki et al., 2011	FoodLog		6512/2000	Captured by users
Wang et al., 2015	UPMC		90840/101	Web Image Search
Farinella et al., 2014	UNICT-FD889	Generic	3583/889	[illegible] users using a smartphone
Singla et al., 2016	MMSPG-Food–11		16643/11	Collected from other food datasets
Singla et al., 2016	MMSPG Food–5K		5000/2	Collected from other datasets
Chen and Ngo, 2016	VIREO Food–172	Chinese	110241/172	Collected from Baidu and Google image search engines
Chen, 2012	Chen		5000/50	Downloaded from Web
Güngör et al., 2017	Turkish Foods–15	Turkish Dishes	7500/15	Collected from other datasets
Pandey et al., 2017	Indian Food Database	Indian Food	5000/50	Collected from Online Sources
Ciocca et al., 2017	UNIMIB 2016	Italian Food	1027/73	Images are captured from a dining hall food tray
Termritthikun et al., 2017	THFood–50	Thai Food	200–700/50**	Collected From Search Engines

The datasets shown above is in varying forms and genres. We used the genre criteria to filter out what is not very similar to the Egyptian food. Added to that, among the other criteria that are not mentioned above, is the availability of bounding boxes and the presence of multiple plates/dining table images. In the next chapters we will discuss how we even filtered the chosen datasets even more to keep it strictly to local food.

3.5.2. Traditional Techniques

Traditional Image processing techniques include classical classification and detection using algorithms that were discussed in the previous section. Here we spent our time researching for the methods that worked with our previously defined problem of food detection and classification. Here's a summary of the papers we found most helpful with their reported accuracies.

Table 3.2 Classical Approaches Survey [9]



Authors	Classifier	Features	Performance	
			Top 1	Top 5
Hoashi et al., 2010	MKL	BoF, Gabor, color, HOG, and texture	62.5%	N/A
Yang et al., 2010	SVM	Pairwise local features	78.0%	N/A
Kong and Tan, 2011	Multi-Class SVM	Gaussian Region Detector and SIFT	84%	N/A
Bosh et al., 2011	SVM	Color, Entropy, Gabor, Tamura, SIFT, Haar Wavelet, Steerable, DAISY, and Predominant color divided into local and global features.	86.1%	N/A
Matsuda et al., 2012	MKL-SVM	HOG, SIFT, Gabor, color and texture	21.0%	45.0%
Kawano and Yanai, 2013	SVM	SURF and color	N/A	81.6%
Anthimopoulos et al. 2014	SVM	SIFT, color	78.0%	N/A
Pouladzadeh et al.., 2014	SVM	GraphCut, color, size, shape and texture	95.0%	N/A
He et al., 2014	KNN	DCD, SIFT, MDSFIT, and SCD	64.5%	N/A
Kawano and Yanai, 2014	One × rest linear classifier	Fisher Vector, HOG and color	50.1%	74.4%
Christodoulidis et al., 2015	SVM	LBP and color	82.2%	N/A
Yanai and Kawano, 2015	Fisher Vector	HOG and color	52.9%	75.5%
Pouladzadeh et al., 2015	Cloud-Based SVM	Gabor, color	94.5%	N/A
Farinella et al., 2016	SVM	SIFT, PRICoLBP, and Bag of textons	75.74 %	85.68 %

From this survey we can notice that the trend of using traditional approaches was decreasing. The most recent paper using traditional techniques was in 2016.

Moreover, the highest accuracy reported on top 1 performance is 95% from the paper by P. Pouladzadeh, S. Shirmohammadi, and A. Yassine [10]. This paper proposed two traditional methods, one for segmentation and the other is for classification. The figure shown is the proposed system from the paper.

It is basically divided into four main blocks: Image Processing, Segmentation, Classification and the Feature Extraction. Classification uses trivial SVM. We'll focus on the Segmentation and Feature Extraction blocks.

The Segmentation used was Graph-cut followed by a color clustering and a texture segmentation technique that we explained in the

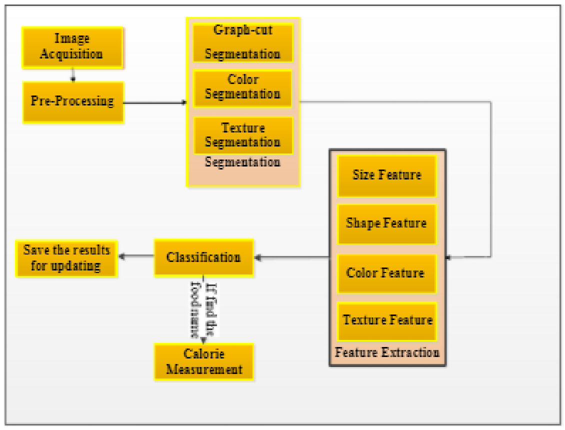
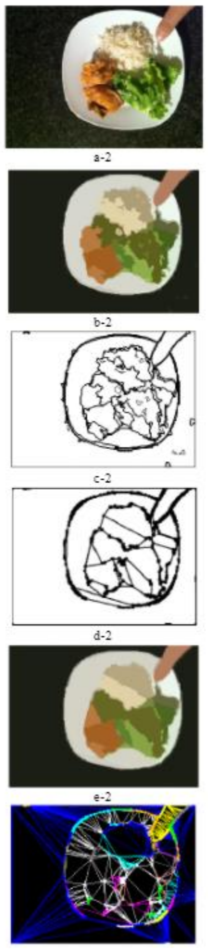


Figure 3.21 Traditional Food Detection Architecture [10]

previous chapter. The paper was oriented towards the segmentation as it improved the overall accuracies by 30% from the authors' previous attempts. As for the classification, the authors used the size, shape, color and texture features stated in the previous section. It is worth mentioning

that their dataset consisted of only 30 classes and they were mainly fruits/vegetables. They were mainly assuming multiple food per dish and a single dish per image. Lastly, all their dataset images were very neatly top viewed with a black background. A list of their food classes with a sample image and the recognition rate are found below.

Table 3.3 Traditional Food Detection Architecture [10]



No.	*Food items*	*Recognition Rate (%)*	
		Using color-texture segmentation	*Using graph-cut, color-texture segmentation*
1	Red Apple	97.64	100
2	Orange	95.59	97.5
3	Corn	94.85	96
4	Tomato	89.56	95
5	Carrot	99.79	100
6	Bread	98.39	99
7	Pasta	94.75	98
8	Sauce	88.78	92
9	Chicken	86.55	89
10	Egg	77.53	83
11	Cheese	97.47	97
12	Meat	95.73	96
13	Onion	89.99	93
14	Beans	98.68	98
15	Fish	77.7	85
16	Banana	97.65	97
17	Green Apple	97.99	97
18	Cucumber	97.65	98
19	Lettuce	77.55	85
20	Grapes	95.7	95
21	Potato	88.56	89
22	Tangerine	97.59	99
23	Chocolate Cake	88.19	85
24	Caramel Cake	85.29	85
25	Rice	94.85	94
26	Green Pepper	97.99	98
27	Strawberry	83.47	98
28	Cooked Vegetable	92.62	96
29	Cabbage	77.55	100
30	Blueberry	83.47	95
	Total average	**92.21**	**95**

3.5.3. Deep Learning Approaches

Deep learning approaches were promising in both the segmentation and the classification. To tackle the problem of classification the approaches were mainly focused on two technologies, Convolutional Neural Networks and Vision Transformers. Meanwhile in Segmentation and Object Detection, Convolutional Neural Nets and Transformers were either involved as a backbone for a new network architecture or as the main solution with modifications. In this subsection we will divide the paper to classification only and Object Detection and Classification.

3.5.3.1. Classification

3.5.3.2. Deep Convolutional Neural Networks

Upon our research we came across a 2020 paper comparing the classification accuracies of different deep convolutional networks on the dataset food-101 mentioned earlier.

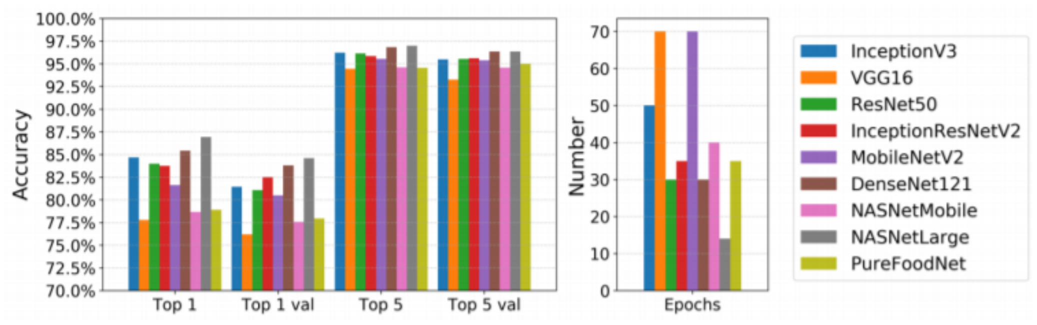


Figure 3.22 DCNN Comparison on Food-101 [11]

The graph above shows the comparison of the CNNs in terms of Top 1 accuracy, Top 5 accuracy and the number of epochs used to fine tune the networks. All the networks were pre-trained on ImageNet Dataset and then with transfer Learning, the authors replaced the necessary top layers with the ones needed to classify the 101 class of food-101, the dataset.

The paper also proposed its own architecture named “PureFoodNet” which showed great results on the charts but neither the implementation nor the architecture was clear in the paper (i.e: number of Layers weren’t stated..etc.) Comparing the previous results to the those from traditional approaches, the gap in accuracies is clear as expected.

3.5.3.3. Vision Transformer (ViT)

Now going further, from the paper “Transformers in Vision: A survey” [12] we were introduced to Google’s new Vision Transformer abbreviated “ViT”. It showed promising results. Here is the architecture as per the paper:

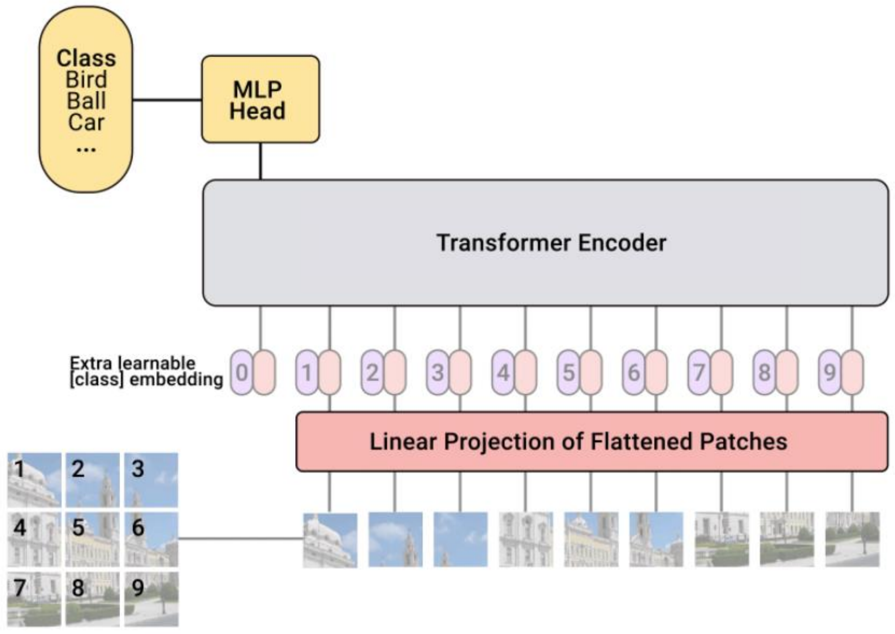

Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Extra learnable [class] embedding
0
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches
1
2
3
4
5
6
7
8
9

Figure 3.23 ViT Architecture [13]

As per the paper, what is happening is as illustrated in the Vision Transformers section, the image is split into 16x16 patches. The image above shows only 9 patches but in reality they used 16.
Second step is the linear projection and flattening of the patches. Then these patches are passed to a network that produces lower-dimensional linear embeddings from the flattened patches. Third step is adding positional encodings to the embeddings. Unlike Recurrent Neural Networks that naturally contain the positional information in the sequence, transformers have to be adjusted to include such information. Fourth, step is feeding the input sequence to a typical transformer encoder of the following architecture:

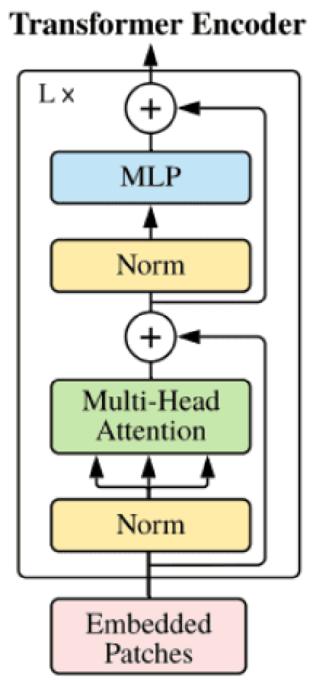


Figure 3.24 ViT Encoder [13]

Next is pre-training the model with image labels (fully supervised on a huge dataset). By huge datasets, the paper means in orders of millions. Datasets like JFT-300 which is Google's 300 million images dataset. Last step comes in transferring the learning and fine tuning on "relatively" small scale datasets like ImageNet.

The figure below shows a comparison of the different Versions and a ResNet Convolutional Network (BiT) on the different datasets and during training.

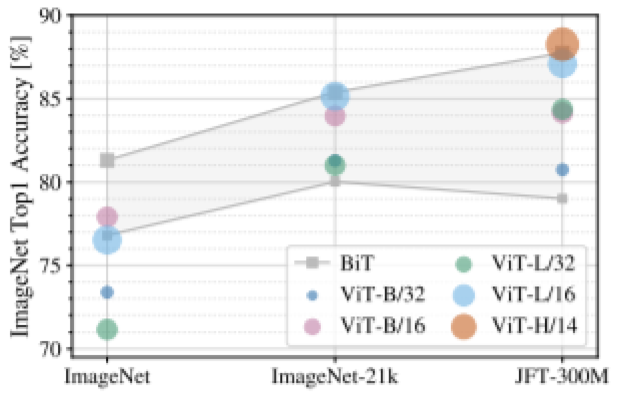


Figure 3.25 Different Versions of Transformers Comparison [13]

Very Important conclusions can be drawn here. As shown, the ResNet Convolutional Neural Network happens to be superior on the low-scale dataset “ImageNet” which supports what is commonly known about transformers being data hungry.

3.5.3.1. Object Detection

3.5.3.1.1. Fast RCNN vs RCNN

The comparison between RCNN and Fast RCNN is not surprising at all. Nevertheless, as the names state, RCNN is the slowest in training time as we have explained earlier, you have to train 2000 region proposals per image. In literature comparisons, it shows that it’s not just the training. Test time figures show a huge difference too.

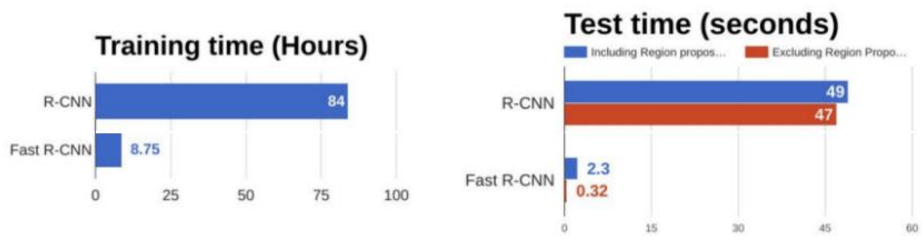


Figure 2.26 RCNN vs Fast RCNN

The reason “Fast R-CNN” is faster than R-CNN is because you don’t have to feed 2000 region proposals to the convolutional neural network every time. Instead, the convolution operation is done only once per image and a feature map is generated from it.

3.5.3.1.2. Faster RCNN vs Fast RCNN

One reason for the name Faster RCNN comes from the fact that the aforementioned algorithms (RCNN and Fast RCNN) use selective search to find out the region proposals. Selective search is a slow and time-consuming process affecting the performance of the network. The chart below shows the test time of Faster RCNN against its predecessors.

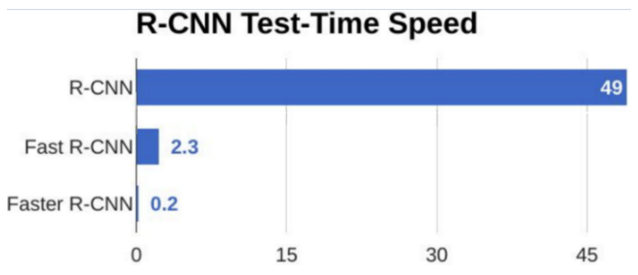


Figure 3.27 RCNN vs Fast RCNN vs Faster RCNN

3.6. Implemented Approach

In this section we will closely consider the different approaches proposed previously and eliminate the approaches that do not fit our problem description.

3.6.1. Dataset Choice

Firstly, considering the Datasets, we mainly filtered by the genre of the dataset food. Upon inspection of classes and images, Japanese, America, Thai, Turkish, Italian and Indian Food Datasets mentioned above. The most fitted and the reasonably similar to the Egyptian Local food was Food-101 [14]. Moreover, Food-101 contains 101 classes with 101K images, hence, it is a balanced dataset. Such attribute of a dataset will make our evaluations easier to calculate and inspection of the miss-classified would be more straight forward.

Although we choose the dataset that we thought it have the most similar classes, we needed to manipulate and further filter the datasets for the odd classes. We ended up with a 54 Class of food-101 including 8 classes of oriental food that we manually collected. Those classes were [Molokheya, Mahshi, Foul Medames, Koshary, Kebda, Golash, Sambosa, Hummus].

A list of the 101 classes is found below and the list of the 54 classes of the final dataset will be available in the next chapter.

Table 3.4 Food-101 Classes

Apple pie	Chicken wings	French fries	Lobster roll sandwich	Ramen
Baby back ribs	Chocolate cake	French onion soup	Macaroni and cheese	Ravioli
Baklava	Chocolate mousse	French toast	Macarons	Red velvet cake
Beef carpaccio	Churros	Fried calamari	Miso soup	Risotto
Beef tartare	Clam chowder	Fried rice	Mussels	Samosa
Beet salad	Club sandwich	Frozen yogurt	Nachos	Sashimi
Beignets	Crab cakes	Garlic bread	Omelette	Scallops
Bibimbap	Creme brulee	Gnocchi	Onion rings	Seaweed salad
Bread pudding	Croque madame	Greek salad	Oysters	Shrimp and grits
Breakfast burrito	Cup cakes	Grilled cheese sandwich	Pad thai	Spaghetti bolognese
Bruschetta	Deviled eggs	Grilled salmon	Paella	Spaghetti carbonara
Caesar salad	Donuts	Guacamole	Pancakes	Spring rolls

Cannoli	Dumplings	Gyoza	Panna cotta	Steak
Caprese salad	Edamame	Hamburger	Peking duck	Strawberry shortcake
Carrot cake	Eggs benedict	Hot and sour soup	Pho	Sushi
Ceviche	Escargots	Hot dog	Pizza	Tacos
Cheesecake	Falafel	Huevos rancheros	Pork chop	Takoyaki
Cheese plate	Filet mignon	Hummus	Poutine	Tiramisu
Chicken curry	Fish and chips	Ice cream	Prime rib	Tuna tartare
Chicken quesadilla	Foie gras	Lasagna	Pulled pork sandwich	Waffles
		Lobster bisque		

One problem with this dataset is that it only contains one plate/dish per image. In our defined problem, a visually impaired individual will definitely have more than a plate in front of him. For this cause and for improving the modularity of the components in our system, we proposed using another dataset which has bounding boxes for the sole purpose of training the object detection model to be able to detect Food or No-Food (i.e.: Food or background). Therefore, we will use two datasets, one for training the classifier (Food-101) and another for training the object detection module.

The dataset chosen for the detection module is UECFOOOD100 [15]. It is an unbalanced dataset that The dataset contains 100-kind food photos. Each food photo has a bounding box indicating the location of the food item in the photo. In our approach for using the dataset we do not really care about the classes as they will all be translated to one class (Food) in our object detection training pipeline. Samples of the UECFOOD100 are presented in the System Testing and Verification Chapter.

3.6.2. Classification Model Choice

In this subsection we will narrow our options to the most prominent models and closely consider each. In the Previous section, we discussed the performance of the traditional image processing, the deep convolutional networks and the vision transformers.

Starting with the traditional image processing techniques, the paper we mentioned earlier had drawn some assumptions that are not valid in our problem. Firstly, their dataset consisted of only 30 classes and they were mainly fruits/vegetables. Of course, this can't be the case in our problem. Also, it was only a single dish per image. This assumption was not directly stated but their sample image was always a single plate. They used the segmentation only for multiple food per dish. Here are some sample images from their dataset:

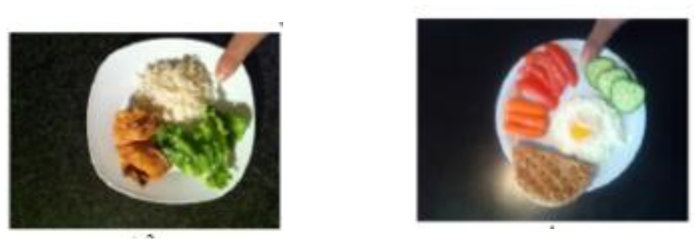


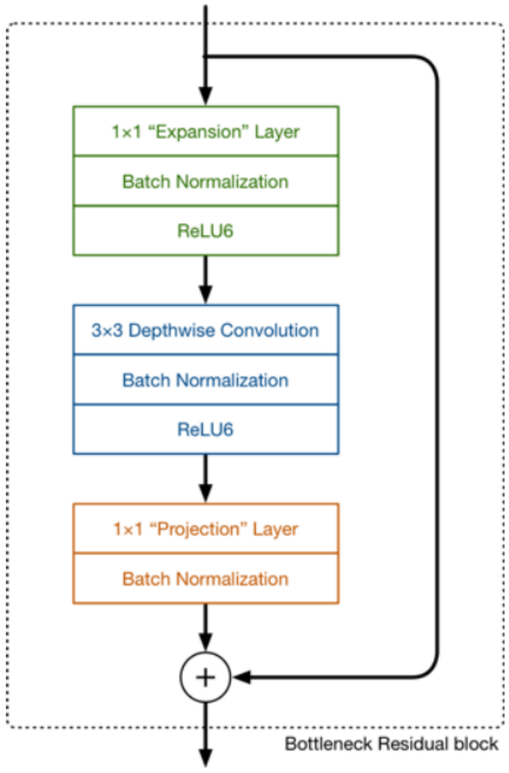
Figure 3.28 Sample Photos from the traditional approach paper [10]

Lastly, all their dataset images were very neatly top viewed with a dark background. Looking into the other traditional approaches were either of low accuracies or high accuracies on very limited datasets with plenty of drawn assumptions. For this reason and the aforementioned ones, we didn't pursue with the traditional approach.

This leave us to the Deep Convolutional Networks and the Transformers. The later showed very promising results. However, from this ViT's chart shown in figure 3.25, the conclusion drawn is that the performance of the transformers is lower than that of the Deep Convolutional Networks (BiT) if it was only trained on a relatively small dataset in this case ImageNet. That forces us to retrieve the pre-trained weights of the transformers trained on the larger datasets. Given that this paper is very recent, it was published officially on the third of June this year, ImageNet weights were the only weights available at the time we started implementing the models. ViT was not supported yet as a pre-trained model in the machine learning frameworks we use for training. Added to that, transformers are data hungry by nature, fine tuning a vision transformer needs computational resources and time that we don't have.
For those two reasons, we weren't lucky enough to live up to the promise and use the anticipated new computer vision "State of the art" in our project.

Hence, after closely analyzing, trying and researching we decided to use a deep convolutional network for the classification. We have seen a comparison between the different convolutional networks in the previous section. Among the very performant networks is MobileNetV2 [16].

MobileNetV2 is a convolutional neural network architecture that seeks to perform well on mobile devices. As a whole, the big idea behind MobileNet is that convolutional layers are quite expensive to compute, can be replaced by so-called depth wise separable convolutions. The job of the convolution layer is split into three subtasks: first there is a depth wise convolution layer that filters the input, followed by a 1×1 (or pointwise) convolution layer that combines these filtered values to create new features and the last layer known as the projection layer, projects data with a high number of dimensions (channels) into a tensor with a much lower number of dimensions. Hence it makes the number of channels smaller. The expansion layer simply does the exact opposite of the projection layer. This kind of layer is also called a bottleneck layer because it reduces the amount of data that flows through the network. Hence, the main building block is as shown in the figure.
The next figure shows an example of the building block in action:



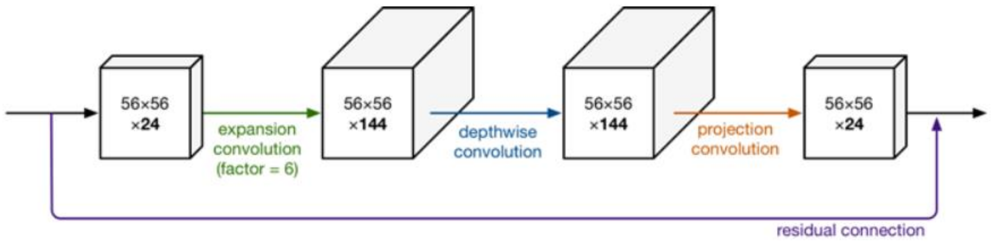
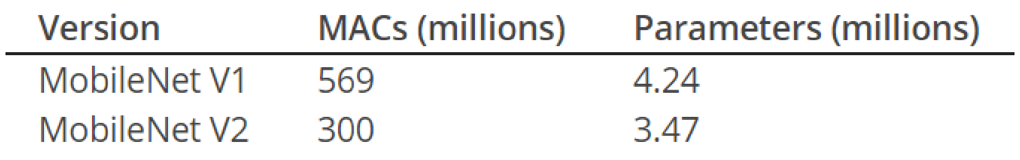


Figure 3.29 MobileNetV2 Architecture

Going to back to the main reason behind choosing MobileNet, here's a comparison between MobileNetV2 and its older sibling MobileNetV1 (only difference is that MobileNetV1 didn't use residual connections and projection layers).

Table 3.5 MobileNet Versions Comparison



Version	MACs (millions)	Parameters (millions)
MobileNet V1	569	4.24
MobileNet V2	300	3.47

"MACs" are multiply-accumulate operations. This measures how many calculations are needed to perform inference on a single 224×224 RGB image. (The larger the image, the more MACs are needed.) So the lower numbers in the table, the better. Also, the fact that is uses fewer parameters is pleasing because that's where most of the speed gains come from on mobile devices.

Hence, for the aforementioned reasons we chose to pursue with a pre-trained MobileNetV2 on ImageNet and fine tuning/training it on our custom dataset. This model will be to classify the photos from the food images out.

3.6.3. Object Detection Model Choice

From the previous section, we can very confidently draw the conclusion that Faster RCNN is the most performant in terms of accuracy (mAp) and speed (Frames Per second, FPS). Therefore, we proceeded with training a pre-trained Faster RCNN and for detecting the different plates in the image. Later, these plates will be cropped and introduced to the classification module which will then classify the classes into the appropriate food class. In the next chapter more implementation details will be provided for the two modules.

Chapter 4: System Design and Architecture

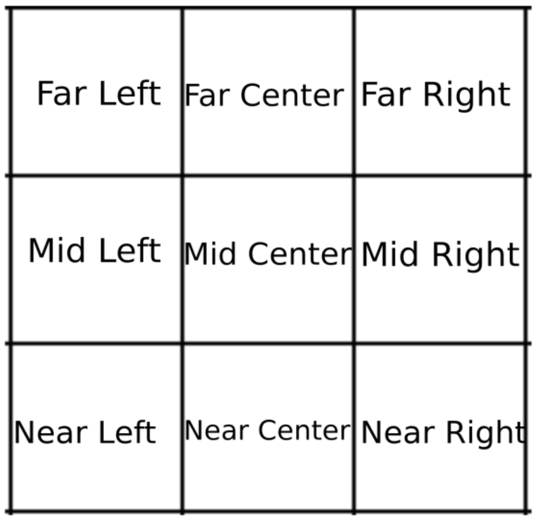
4.1. Overview and Assumptions

In this section, we introduce the chosen underlying architecture and the detailed system design. We start with describing the detection module followed by intermediate processing and ending with the classification module. In this respect we also describe the datasets used for each stage and the models which constitute each module.

Our system uses an assumption in order to solve the problem efficiently. This assumption is made based on the segments we have done surveying on. Namely, the assumption is:

- The user is most likely using the glasses to detect a plate he's holding or a number of plates on a dining table.
- The dish is going to have one item of food only and this will also facilitate the eating process for the disabled man as he will be knowing what he is going to eat from each dish.
- The lights are fair enough to be detected by a 5MP raspberry pi camera module
- No two plates intersect. No plates are stacked.

It is worth mentioning that we presume the following assumption for the input image: the position of a plate in the picture, we assume it is one of 9 positions as shown in the below diagram.



4.2. System Architecture

4.2.1. Block Diagram

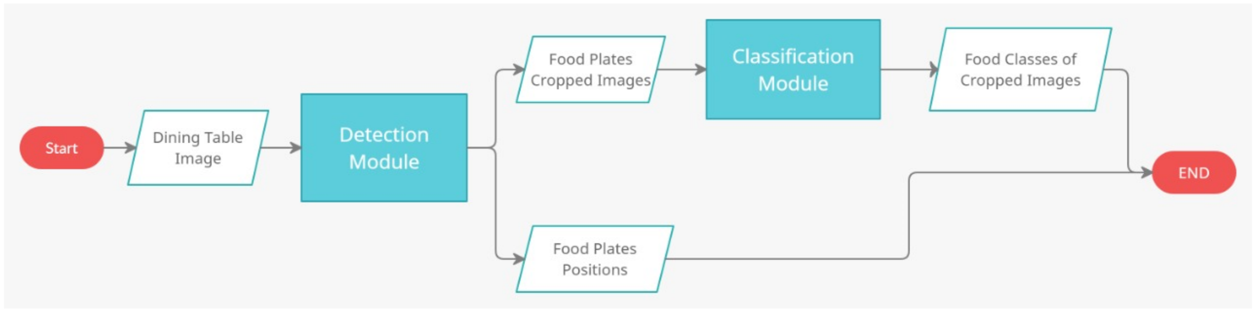


Figure 4.1 Block Diagram

As shown in the block diagram above, a picture is captured of the dining table to be inserted to the detection module, then, the positions of the plates are determined and each plate is cropped to be fed into the classification module. Each image is classified into one of fifty-four food classes.

4.3. Food Detection Module

In this section, we will explain the functionality of the food detection module. We will further demonstrate how this module was implemented and lay out its pipeline.

4.3.1 Functional Description

An input image is fed to the module consisting of several plates of food to be recognized. The module is a FasterRCNN neural network with a ResNet as its backbone.

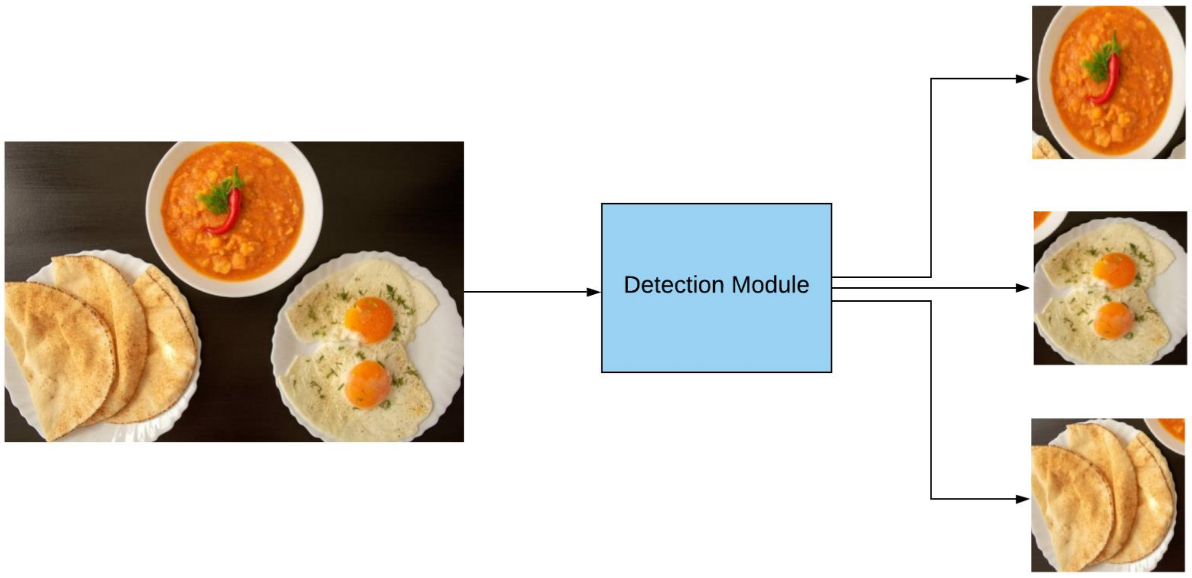


Figure 4.2 Detection Module Block Diagram

4.3.2 Modular Decomposition

4.3.2.1 Datasets

The detection model was trained to detect food images using UECFOOD100 which consists of 100 different food classes. Each food photo has a bounding box indicating the location of the food item in the photo. Most of the food categories in this dataset are popular foods in Japan. It consists of roughly 14.4 thousand images. It is important to note that the dataset we use is unbalanced, i.e., classes have different numbers of pictures.

4.3.2.2 Classical Model

At the beginning of our work on the project, we tried detecting ellipses as most of the plates are circular or oval shaped. But we soon found that it is an impractical solution to the problem of food detection for two reasons:

The picture has to be captured from a top view, which is not the case for a person sitting on the table.

The food has to be in an elliptical plate, which is not always the case.

The image below shows the initial work we have done using the mentioned approach. We used Hough transform to detect circular plates.

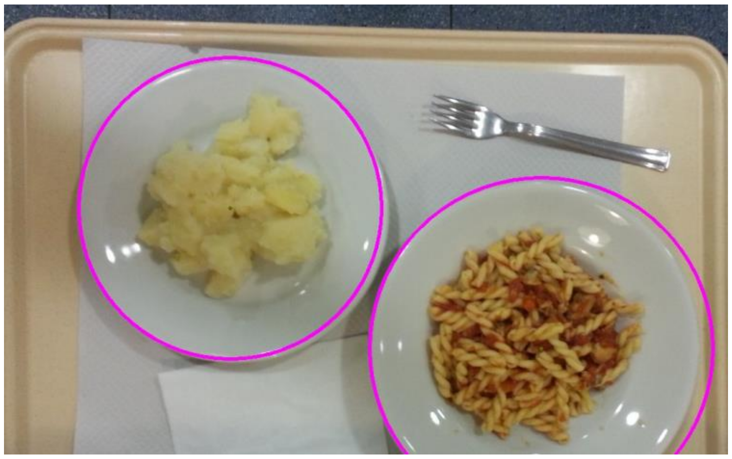


Figure 4.3 Hough Transform Output

4.3.2.3 Deep Learning Model

We built the detection model using the FasterRCNN neural network with residual neural network as its backbone.

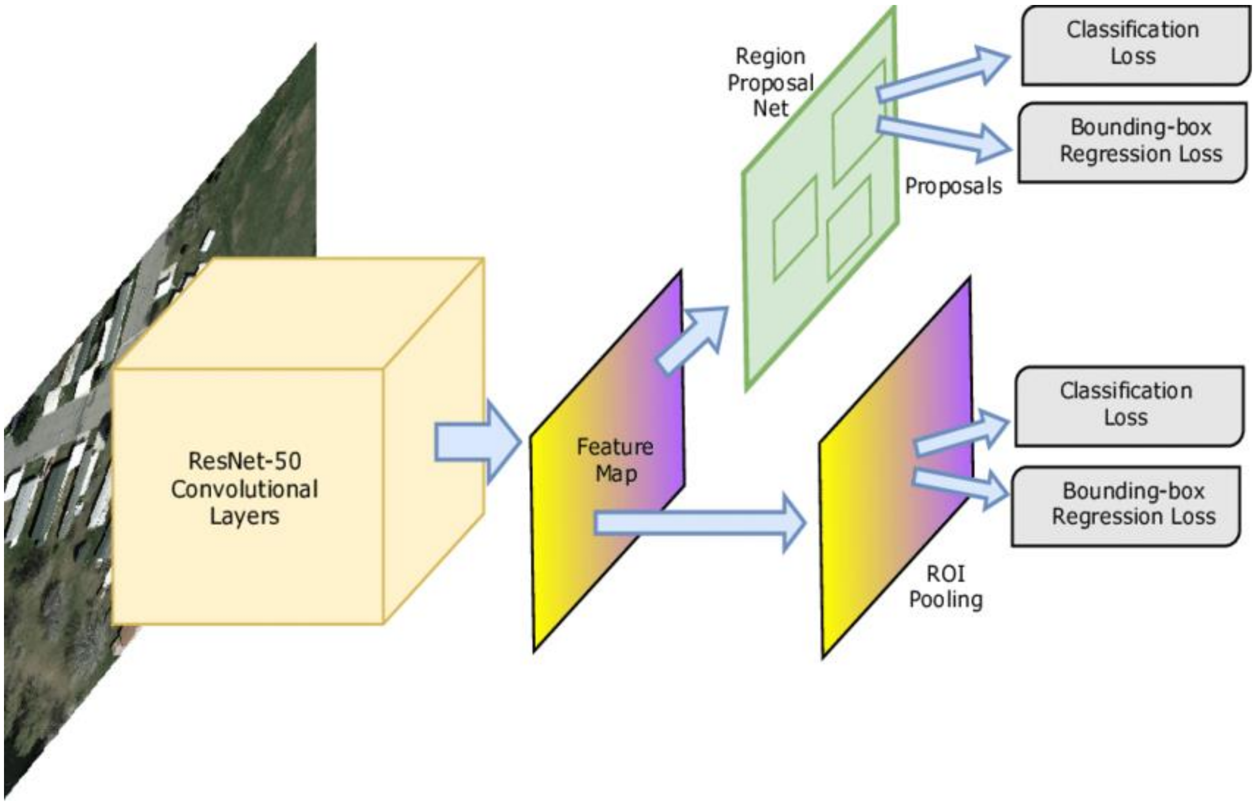
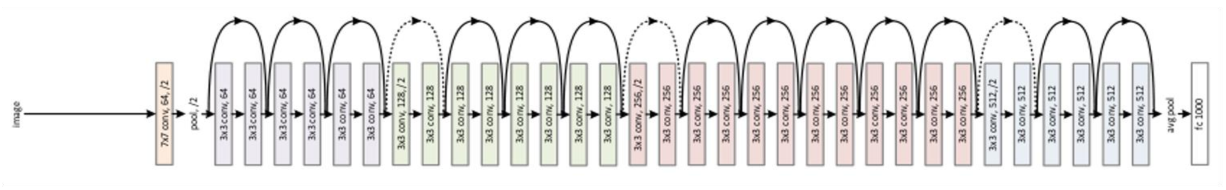


Figure 4.4 ResNet-50 Faster RCNN [8]

Faster RCNN

Faster R-CNN was introduced in 2015 by k He et al.Our object detection uses the Faster RCNN network, which is composed of two modules The first module is a deep fully convolutional network that proposes regions, and the second module is the Fast R-CNN detector that uses the proposed regions.

Residual Networks(ResNets)



The backbone of the fasterRCNN we used is a ResNet. Residual neural networks utilize skip connections, or shortcuts to jump over some layers. Adding skip connections is beneficial for two main reasons: to avoid vanishing gradients, or to reduce the accuracy saturation problem; where adding more layers to a suitably deep model leads to higher training error. ResNet50, which we are using, is a variant of the ResNet model which has 48 Convolution layers along with 1 MaxPool and 1 Average Pool layer.

4.3.2.4 Detection Pipeline

The training phase of the detection model began with loading the food data from the datasets together with the bounding boxes and labels. Secondly, we split the datasets into training and test sets. Hence, we train the neural network. We consider having two classes; 1 indicating food, the other indicates non-food image. The model is trained with the following configurations: 16 batches, 27 epochs, and optimized with the ADAM algorithm.

Non-maximum suppression

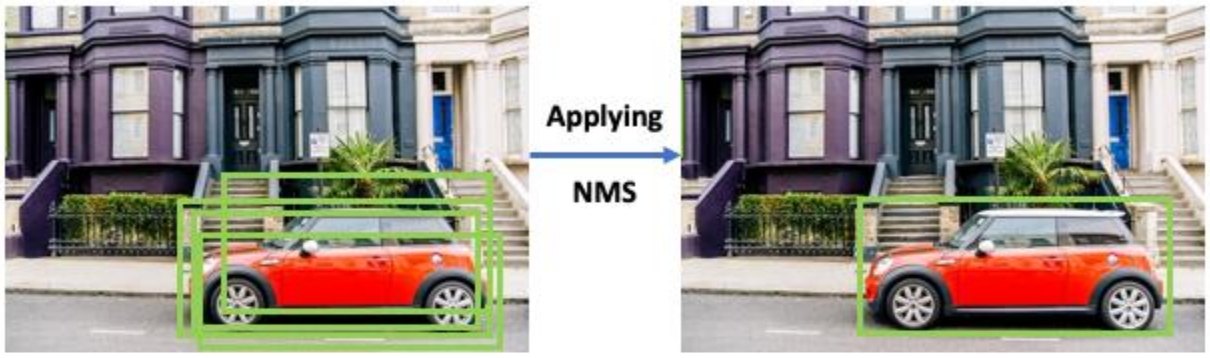


Figure 4.5 NMS Example

Non maximum suppression is used in numerous computer vision tasks. In our case, we use it after the detection module has finished detecting and produced the candidate bounding boxes. The inputs to the non-maximum suppression function are the bounding boxes, the score of each bounding box prediction, and the intersection over union threshold, in our case it is 0.08. The output is a single bounding box that completely covers the span of the detected object.

Eventually, the location of the food plate is detected to be in one of 9 position grids. We divide the picture into the following boxes: far left, far center, far right, mid left, mid center, mid right, near left, near center, and near left. The object (food plate) detected is assigned the position closest to it.

4.4. Food Classification Module

In this section, we will explain the functionality of the food classification module. We will further demonstrate how this module was implemented and what were the design constraints.

4.4.1. Functional Description

This module takes a food plate image as the input and the model's functionality is to predict the type of the food in that image. This procedure can be easily demonstrated by the next figure.

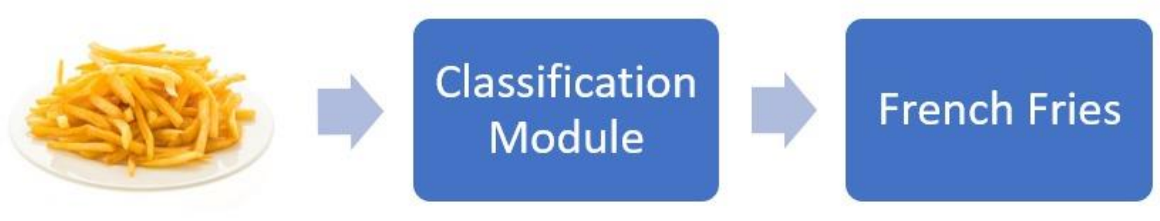


Figure 4.6 Classification Module Block Diagram

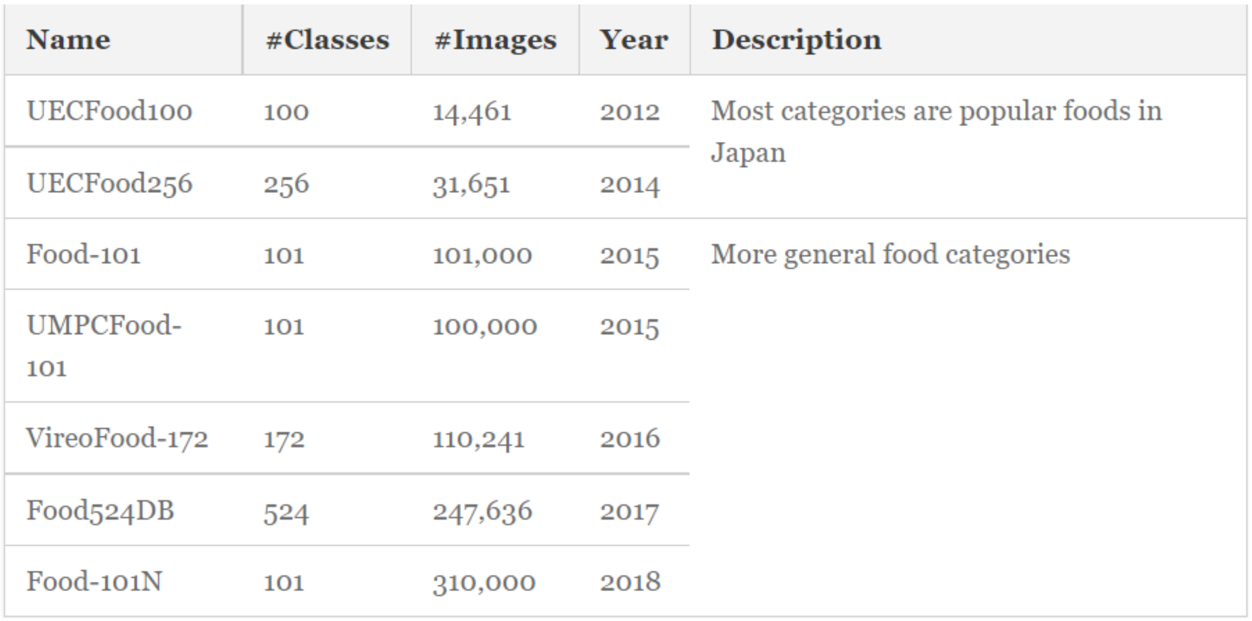
The idea is simple but how are we going to do this? That is the important question.

4.4.2. Modular Decomposition

4.4.2.1. Dataset

The first step in this model is choosing or gathering the right dataset. There are a lot of food datasets you can find publicly.

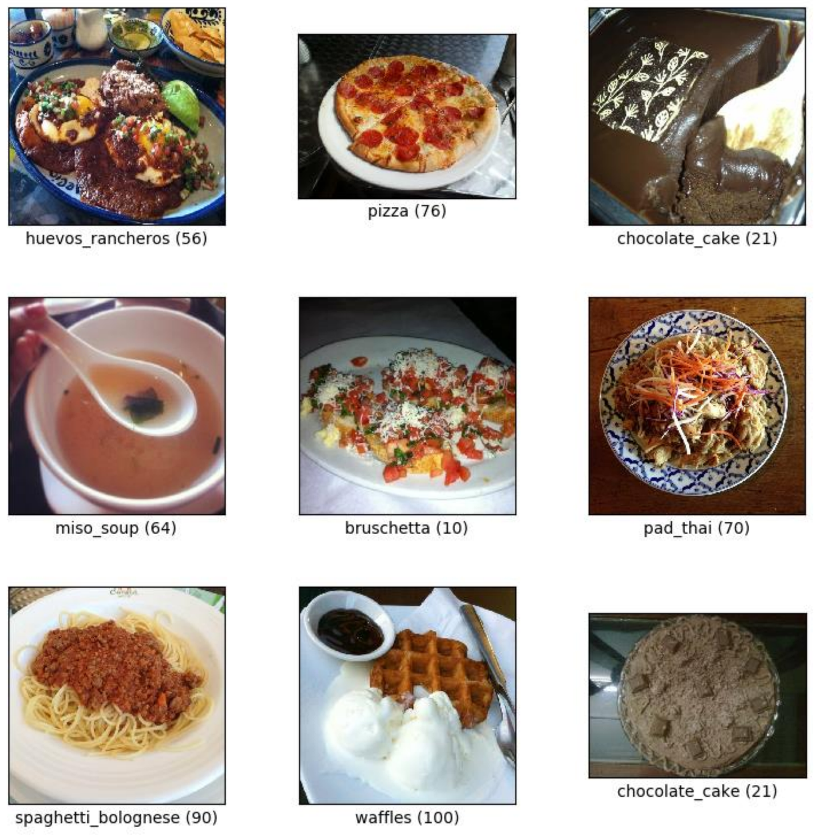
Table 4.1 Datasets Comparison



Name	#Classes	#Images	Year	Description
UECFood100	100	14,461	2012	Most categories are popular foods in Japan
UECFood256	256	31,651	2014	
Food-101	101	101,000	2015	More general food categories
UMPCFood-101	101	100,000	2015	
VireoFood-172	172	110,241	2016	
Food524DB	524	247,636	2017	
Food-101N	101	310,000	2018	

These are some of the many food datasets we found. We needed a balanced dataset which means that all the classes have the same number of images. In the classification module we wanted a dataset that is more general and not constrained to a specific culture.

Food-101 was our choice as it contained a lot of food types we know, and this was a great start for us. Food-101 contains 101 classes of food, each class contains exactly 1000 images which was what we were looking for. The next figures will show an example of images you can find in this dataset.



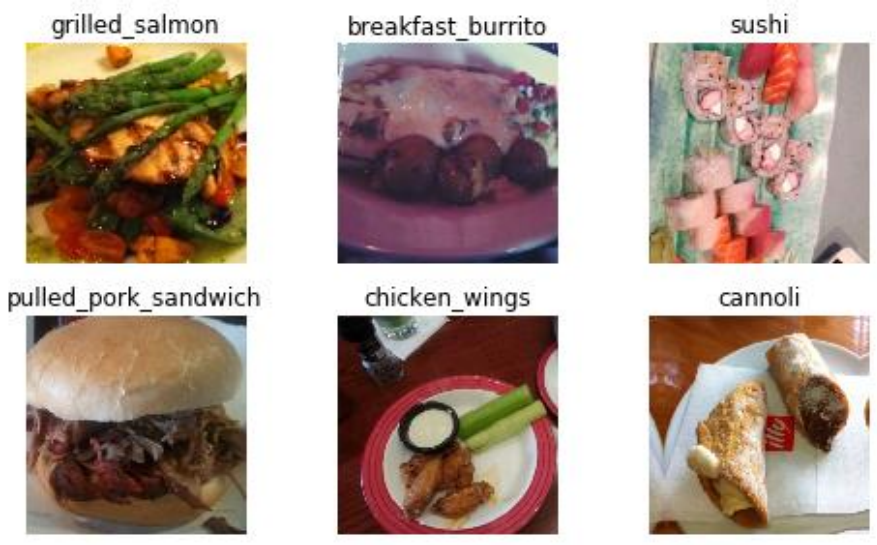


Figure 4.7 Food-101 Sample Photos 2

The dataset needed some work also, as mentioned before it contains 101 classes, we needed to remove the unneeded classes. This process is necessary because the more classes the harder it will be to get a high accuracy. So, the logical solution is to remove the classes we do not need.

We also needed to add oriental food to be more conventional in our region, but this was not as easy as it sounds. Each class contains 1000 images, so we needed to get the same number of images or close to it, in order to keep the dataset balanced.

The table below lists all the 54 classes we used to train the classification model on. It consists of oriental food but mostly from famous food around the world such as pizza and ice cream.

Table 4.2 Our Chosen Classes

Caesar salad	Guacamole	Pizza
Cheesecake	Hamburger	Prime rib
Cheese plate	Hot dog	Ramen
Chicken wings	Hummus	Red velvet cake
Chocolate cake	Ice cream	Rice
Creme caramel	Kebda	Risotto
Cup cakes	Koshary	Sambosak
Donuts	Lasagna	Shrimp and grits
Egg plate	Macaroni and cheese	Spaghetti bolognese
Falafel	Macarons	Spaghetti carbonara
Foul heraty	Mahshy	Spring rolls
Foul medames	Molokheya	Steak
French fries	Mussels	Strawberry shortcake
Fried calamari	Omelette	Sushi
Fried fish	Onion rings	Tacos
Garlic bread	Oysters	Tiramisu
Golash	Pancakes	Tomato salad
Greek salad	Peking duck	Waffles

4.4.2.2. Deep Learning Model

We wanted to make a classical model at first, but approximately all the papers were against this idea. It was used in the old days but they were not that effective. We tried a couple of approaches but most of them lead to a dead end. Even the approaches we completed to the end were time consuming and their predictions were lower than we expected.

After the research done and documented in the literature survey chapter, we now need to choose a model. Our main priority is speed but of course accuracy is very important. Now we need a backbone model for feature extraction.

We decided to use **MobileNetV2**, because of its high reliability and its speed, as it is considered a light weight model. Before we explain the architecture of MobileNetV2, we need to have a look at MobileNetV1 first.

As you can see in the next figure, this is the convolutional blocks of MobileNetV1.

Here there are 2 layers, the first layer is depth wise convolution. It is used to perform a lightweight filtering by applying a convolutional filter for each input channel. The second layer is a 1x1 convolution. This layer is responsible for building new features by computing linear combinations of the input channels.

Relu6 is used due to its robustness when used with low-precision computation. It is defined by the function **min(max(x, 0), 6)**.

Now let us see the difference in these convolutional blocks that were done in MobileNetV2.

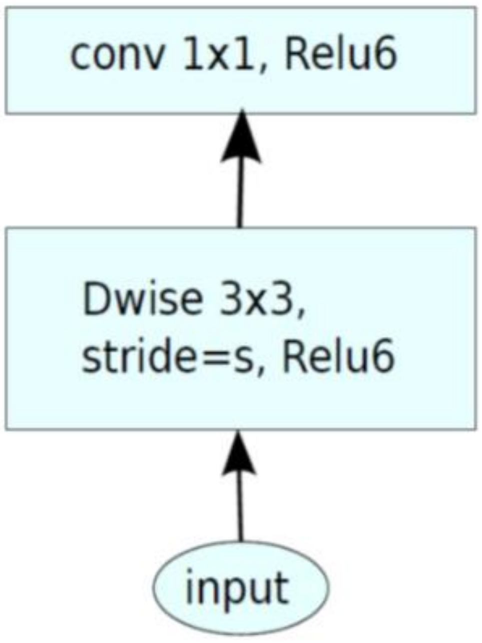


Figure 4.8 MobileNetV1 Convolution Block [16]

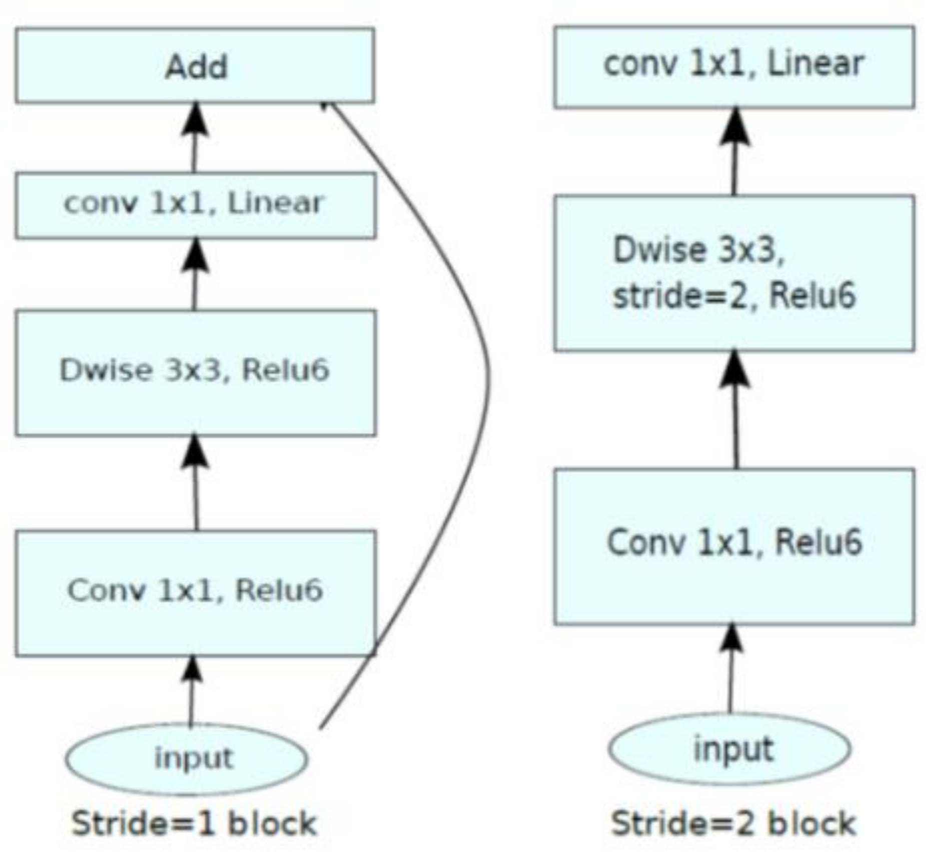


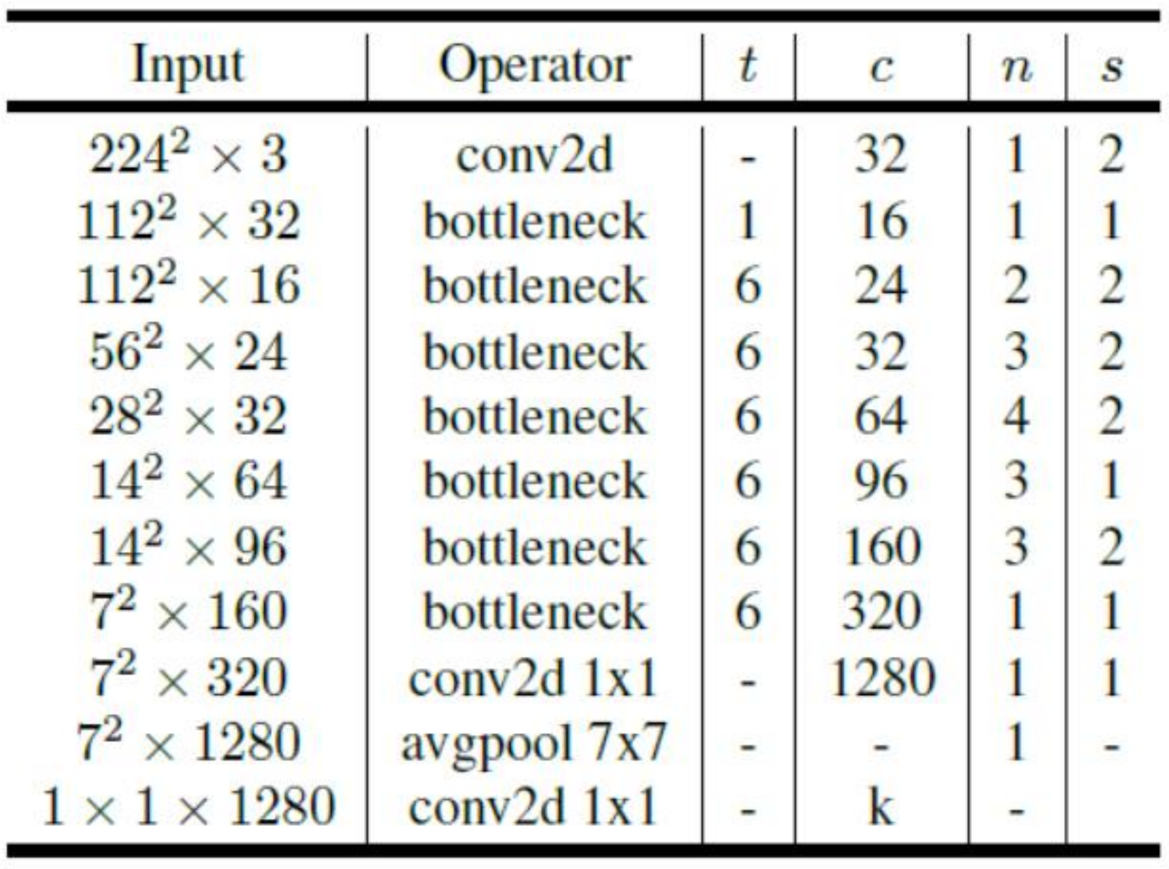
Figure 4.9 MobileNetV2 Convolution Blocks [16]

In MobileNetV2, there are two different types of blocks. The first one is a residual block with stride of 1, and the other is a block with stride of 2 used for downsizing. As we can see in the above figure, there are 3 layers for the two types of blocks.

The first layer is a 1x1 convolution with Relu6. The second one is the depth wise convolution with Relu6. The third layer is another 1x1 convolution, but without any non-linearity.

The overall architecture of MobileNetV2 is shown in the next figure, where t is the expansion factor, c is the number of output channels, n is the number of repeatability and s is the stride.

Table 4.3 MobileNetV2 Architecture



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

The bottleneck is the convolutional blocks explained above. The input image is a 224x224x3. This deep neural network has 3.4 million parameters, which is why it is nearly impossible for us to train it from the beginning. So, we needed to get a pretrained model.

We used a pretrained model on **ImageNet**. This is an image classification challenge. The goal of this image classification challenge is to train a model that can correctly classify an input image into 1,000 separate object categories. Models are trained on 1.2 million training images with another 50,000 images for validation and 100,000 images for testing.

Now that we got the perfect pretrained model for us, the next step is the transfer learning process. The model already has some knowledge in image classification, but the parameters in the latter layers need to be tuned to be fit our food classification problem. Using this technique saves time that would have been wasted in training.

We removed the last few layers in the model and added our layers with the number of classes we needed. Then, we started training for 50 epochs as recommended by the papers.

4.4.3. Design Constraints

One of the design constraints that faced us in this module was the speed vs accuracy problem. The speed is very important for us as we want the classification to be instant. But the accuracy cannot be ignored. What is the use of giving fast wrong answers?

We made our priority the speed but accuracy is still as important to us. Thus, we were looking to the best speed possible given that the accuracy does not go under the minimum we set.

Another design constraint was the accuracy vs number of classes or types of food. The more types you include in your dataset the more valuable this module gets. Here comes another tradeoff. You want your food menu to be big and at the same time the accuracy of the classification to be very high. Increasing one variable will decrease the other logically.

Our solution to this problem was to focus on the important food types in our menu and remove the food we rarely see here in Egypt. Now our menu should not be enormous, so our accuracy is supposedly high.

4.4.4. Other Description of the Classification Module

To ensure that this module is clear, it is better to see the bigger picture of the whole process. We already saw how the detection module works. Briefly the detection module is used to get the boundary boxes of the food plates. Now, we can crop these boxes to get the food plates.

Here comes the classification module's turn. The input for this module is the images coming from the detection module. The expected output from this module is the classification of each image. In other words, predicting the food type of each plate.

4.5. Graphical User Interface

4.5.1 Functional Description

This module is mostly used to visualize our work. It helps us to show the outputs of the program as a whole. The GUI allows you to add an input image, then this image is passed through the detection model and the classification after that. The output is then displayed to the user.

4.5.2 Modular Decomposition

4.5.2.1 Desktop GUI

We started with a desktop GUI using Kivy. We chose Kivy because we wanted both backend and frontend to be in Python. With that approach we will not need an API in our application.

The GUI is simple, first it will ask you to choose your input image from either an already saved image or from your camera. But the camera is not much used in this desktop version, as it opens the camera of your laptop. This will be explained and shown in the user guide section in the appendix.

4.5.2.2 Mobile Application

We exploited the fact that Kivy can create cross-platforms. You can run the same code with some small modifications on android. We tried this approach using Buildozer to generate our .apk application, but some packages were not supported and the app crashes instantly after opening.

We tried another approach which is making the frontend with Kivy and a backend with Flask and connect them using an API. This solution worked fine at first. We had problems deploying this application on different android versions. We needed to try another approach.

Our final solution was to use **Flutter** as the frontend, and use the **Flask** backend implemented in the previous approach.

4.6. Hardware

4.6.1. Prototype Functionality

On running the program, the classification and detection modules are loaded and the program waits for the button to be pressed. On pressing the button, the camera module captures a photo, which is then fed to the two models to recognize food in the picture. On finishing the first picture, the program waits for another keypress and so on.

4.6.2. Modular Decomposition

4.6.2.1. Raspberry pi 4

We used the raspberry pi 4 as a prototype for our project. We bought one with the following specifications:

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 8GB LPDDR4-3200 SDRAM
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2 USB 3.0 ports; 2 USB 2.0 ports.
- Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
- 2 × micro-HDMI ports (up to 4kp60 supported)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
- OpenGL ES 3.0 graphics
- Micro-SD card slot for loading operating system and data storage
- 5V DC via USB-C connector (minimum 3A*)
- 5V DC via GPIO header (minimum 3A*)
- Power over Ethernet (PoE) enabled (requires separate PoE HAT)

4.6.2.2. Camera Module

In addition to the raspberry pi we bought a camera module to capture images with the following specifications:

Table 4.4 Camera Module Specs

Size	Around 25 × 24 × 9 mm
Weight	3g
Still resolution	5 Megapixels
Video modes	1080p30, 720p60 and 640 × 480p60/90
Linux integration	V4L2 driver available
C programming API	OpenMAX IL and others available
Sensor	OmniVision OV5647
Sensor resolution	2592 × 1944 pixels
Sensor image area	3.76 × 2.74 mm
Pixel size	1.4 µm × 1.4 µm

Optical size	¼”
Full-frame SLR lens equivalent	35 mm
S/N ratio	36 dB
Dynamic range	67 dB @ 8x gain
Sensitivity	680 mV/lux-sec
Dark current	16 mV/sec @ 60 C
Well capacity	4.3 Ke-
Fixed focus	1 m to infinity
Focal length	3.60 mm +/- 0.01
Horizontal field of view	53.50 +/- 0.13 degrees
Vertical field of view	41.41 +/- 0.11 degrees
Focal ratio (F-Stop)	2.9

4.6.2.3. Other Components

- A button to control the camera.
- A headphone to listen to the result of the recognition.
- Male/Male and Female/Female jumpers.
- A small breadboard.

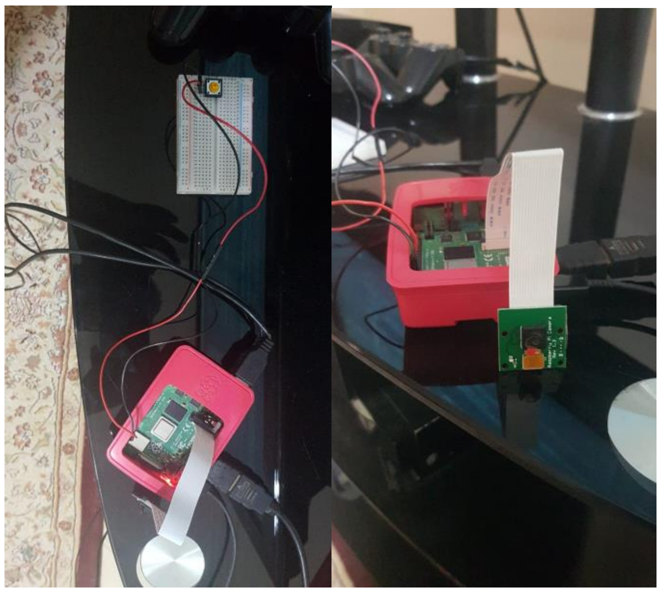


Figure 4.10 Hardware Photos

Chapter 5: System Testing and Verification

In the software development cycle, testing is an essential step. It checks whether the actual software product matches expected requirements and to ensure that software product is defect free. Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss. We detail in the following sections how we tested the 'Eyefood' system.

5.1. Testing Setup

We tested the two modules i.e. the detection and classification modules each with a different dataset as each problem demands. The detection module was tested with the UECfood100 dataset which consists of pictures containing multiple plates as shown in the below figures.

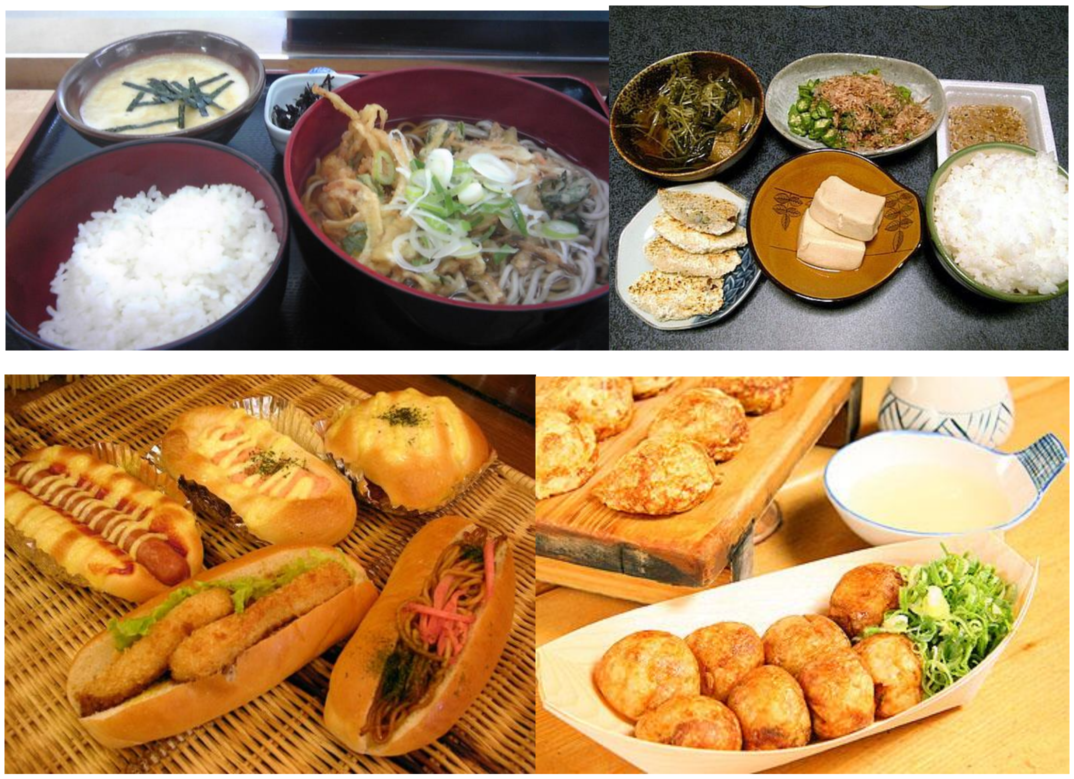


Figure 5.1 UECFOOD-100 Sample Photos 2

5.2. Testing Plan and Strategy

Initially, in our testing plan, we had to find a suitable dataset to train and test our program on, luckily UECFood100 is a suitable open-source dataset that is available online. On the other hand, we had to build part of the dataset used for classification in order to include oriental food like molekhya, and mahshy. We started with the Food101 dataset which included 101 classes each class had 1000 images. We eliminated irrelevant classes and added the data we gathered to end up with a modified Food101 dataset that consists of 54 classes each class with 1000 images and included oriental and Egyptian food.

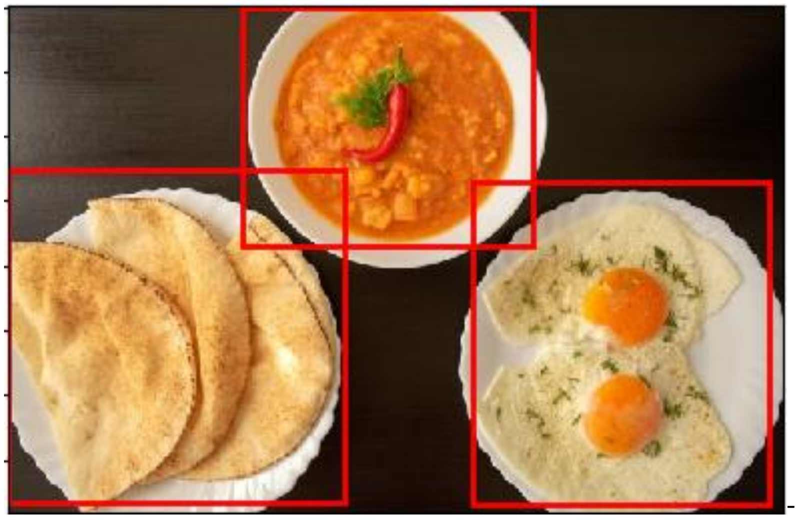
5.2.1. Module Testing

5.2.1.1. Detection Module

Calculating the accuracy of such a module is quite difficult, while the model manages to detect all the plates in a picture it might fail to capture every plate in another picture. In most cases, most of the plates are detected successfully.

Instead, we use the mean average precision(mAP) which is a popular metric in measuring the accuracy of object detectors like Faster RCNN. Before delving into the workings of mAP, we used the precision, recall, and intersection over union (IoU) previously mentioned in the Literature Survey.

Here's some insights from the Inspection of the detected plates



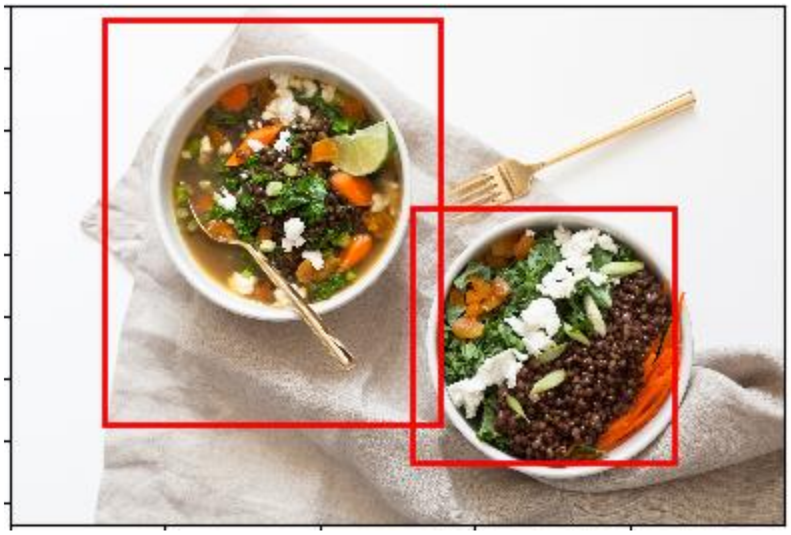


Figure 5.2 Detection Module Testing

5.2.1.2. Classification Module

We began with the classification module at the start of our work on the project using the Food101 dataset without any modifications, later we performed the previously mentioned insertions and deletions on the dataset, and splitted each class to 750 images for training and 250 for testing. The following figure shows the confusion matrix of the model on the 101 classes of Food101.

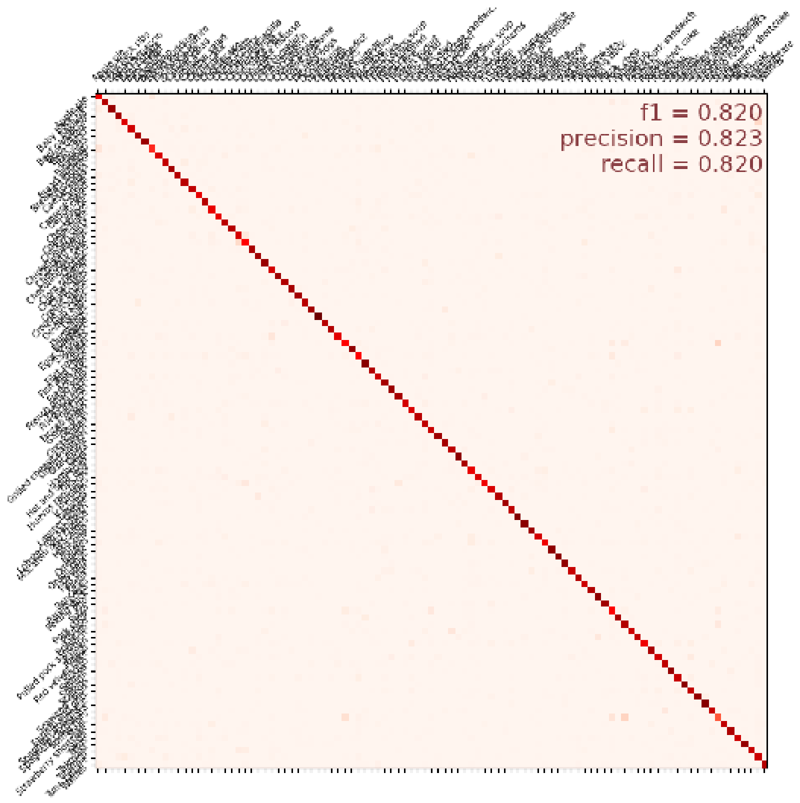


Figure 5.3 Confusion Matrix on 101 Classes

For the 54 classes modified-Food101, here is the confusion matrix and classification report, we omitted the f1 score and the support since the dataset is balanced.

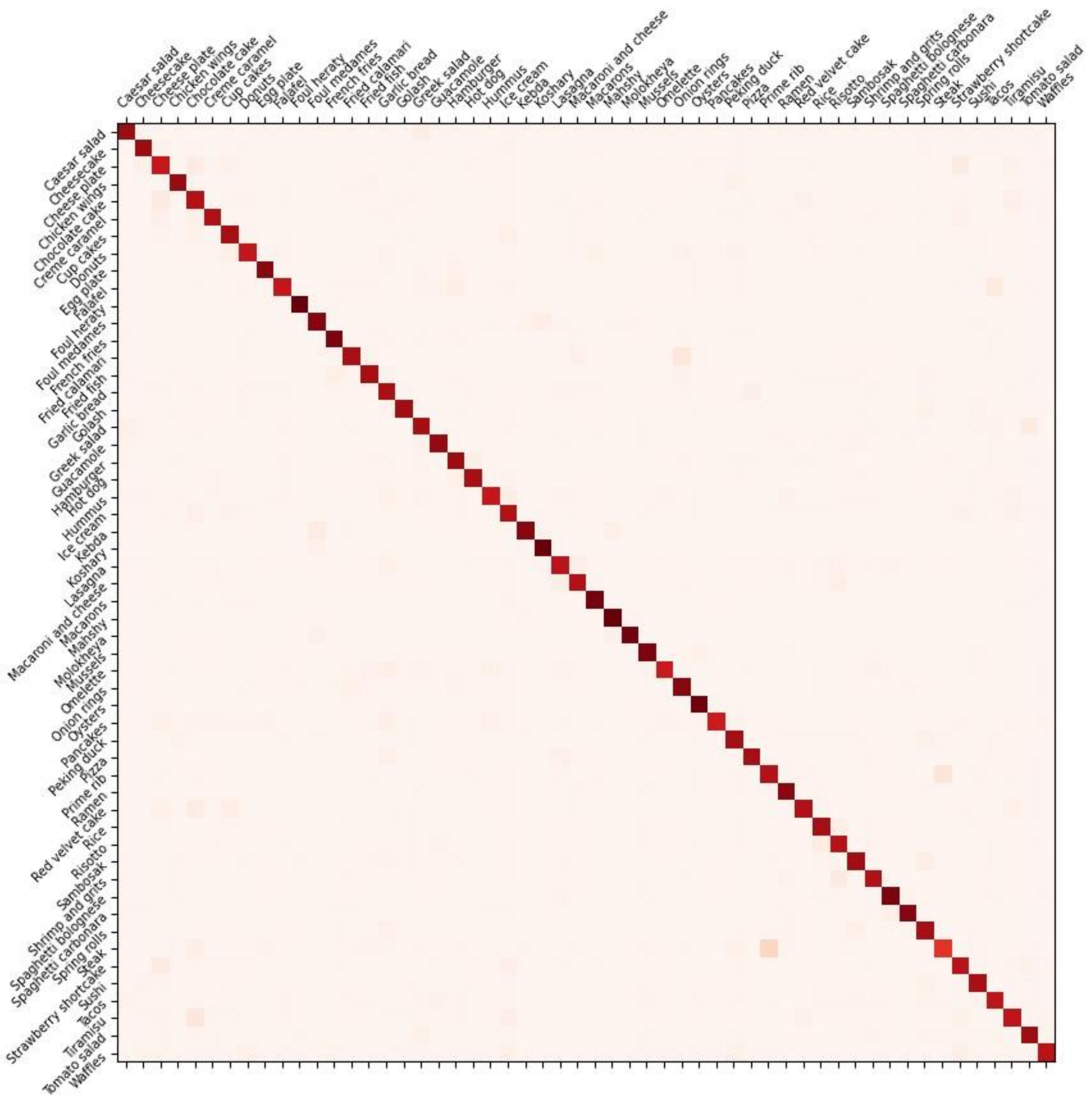


Figure 5.4 Confusion Matrix for 54 Classes

Table 5.1 Precision and Recall for 54 Classes

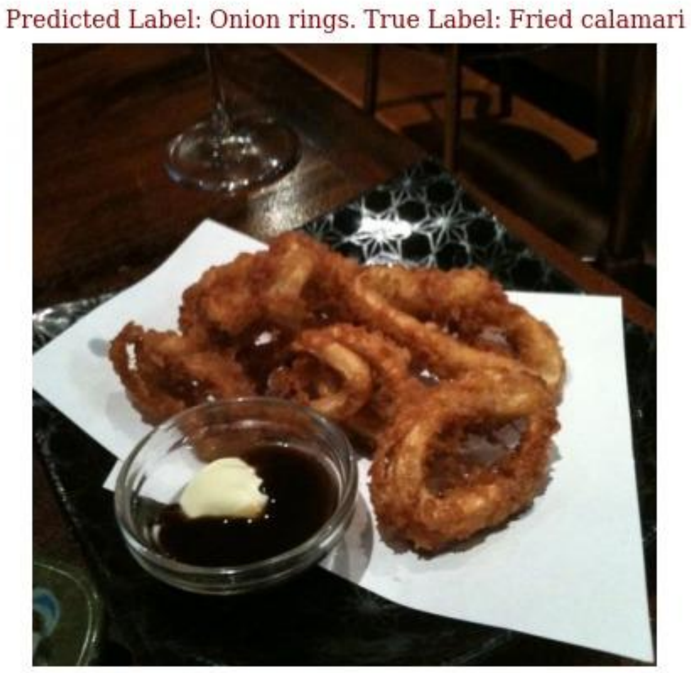
Class	**precision**	**recall**
0	0.88	0.87
1	0.81	0.88
2	0.72	0.74
3	0.91	0.88
4	0.65	0.8
5	0.9	0.82
6	0.83	0.85
7	0.85	0.75
8	0.9	0.9
9	0.87	0.75
10	0.98	0.98
11	0.91	0.92
12	0.89	0.94
13	0.85	0.84
14	0.83	0.84
15	0.74	0.84

16	0.89	0.86
17	0.82	0.85
18	0.88	0.89
19	0.81	0.88
20	0.87	0.84
21	0.81	0.74
22	0.68	0.81
23	0.95	0.91
24	0.93	0.97
25	0.8	0.78
26	0.84	0.8
27	0.88	0.95
28	0.95	0.98
29	0.98	0.95
30	0.89	0.93
31	0.82	0.73
32	0.84	0.91
33	0.89	0.96

34	0.9	0.72
35	0.79	0.86
36	0.92	0.85
37	0.79	0.81
38	0.86	0.91
39	0.9	0.81
40	0.85	0.86
41	0.76	0.8
42	0.85	0.86
43	0.85	0.82
44	0.91	0.94
45	0.95	0.92
46	0.81	0.85
47	0.8	0.64
48	0.78	0.79
49	0.86	0.84
50	0.81	0.77
51	0.78	0.78

52	0.81	0.87
53	0.91	0.79

The next figures show some of the misclassified images in classification. Each figure also show the predicted and true labels of the image. You can see that some of them are even difficult for us to know.



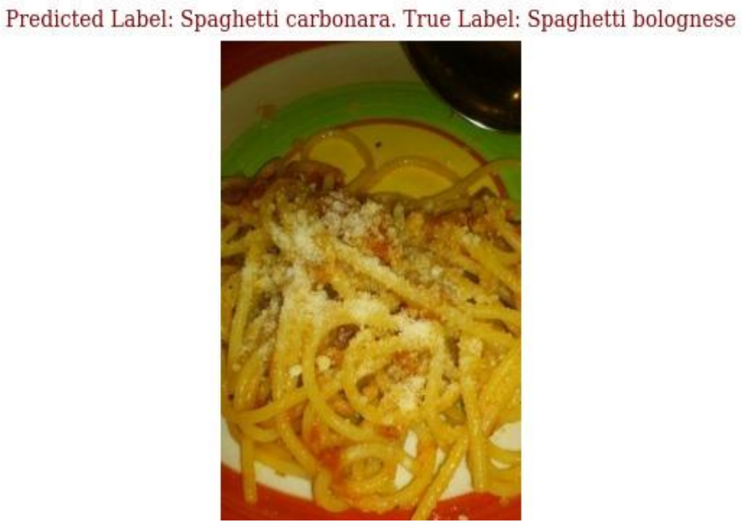

Predicted Label: Spaghetti carbonara. True Label: Spaghetti bolognese

Figure 5.8 Misclassified 2

Figure 5.5 Misclassified 1

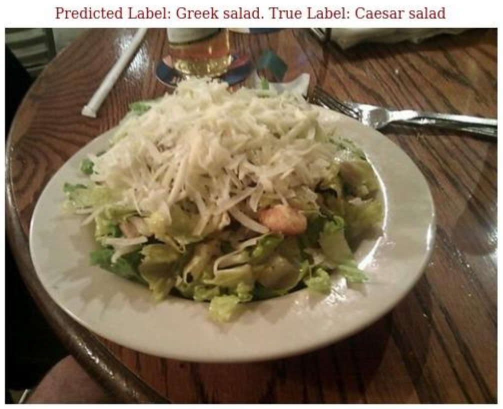


Figure 5.7 Misclassified 3

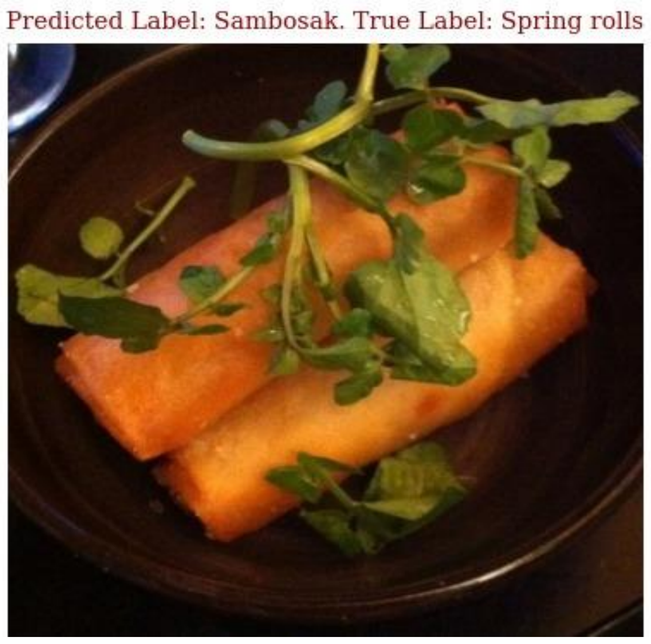


Figure 5.9 Misclassified 4

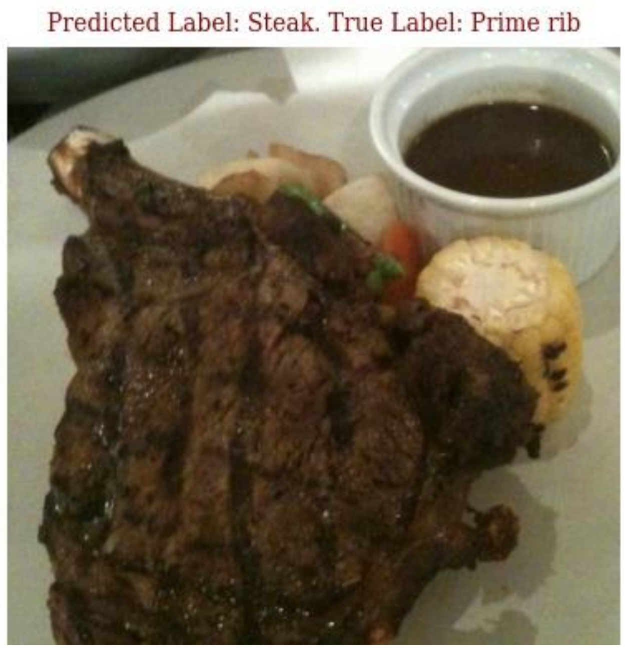


Figure 5.10 Misclassified 5

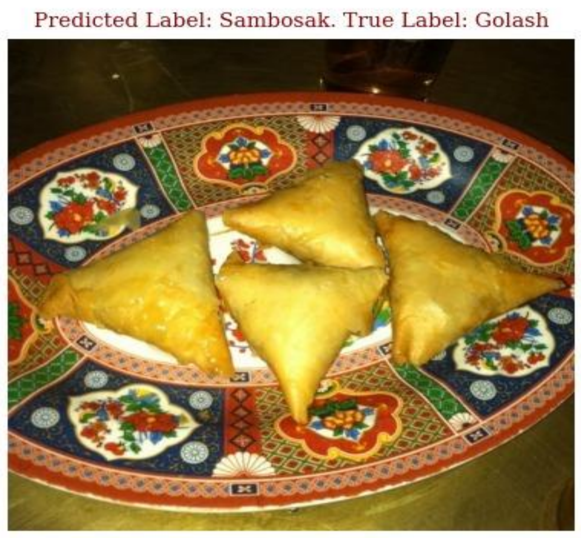


Figure 5.11 Misclassified 7

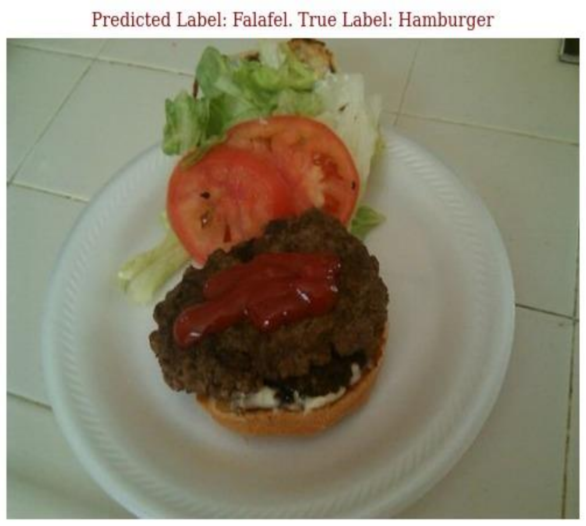


Figure 5.12 Misclassified 8

5.2.2. Integration Testing

After validating that each module is working properly on its own, we started integrating the two modules together. Integrating the two modules was fairly a simple operation as the outputs of the detection module are the inputs of the classification module without the need of any processing to be performed on the images.

5.3. Testing Schedule

The first couple of months in the project were spent on learning about the different machine learning and deep learning techniques that can be used along with researching the previous work in the field of food recognition. The below figure shows the timeline of the implementation and testing phases. The implementation phase started in January.

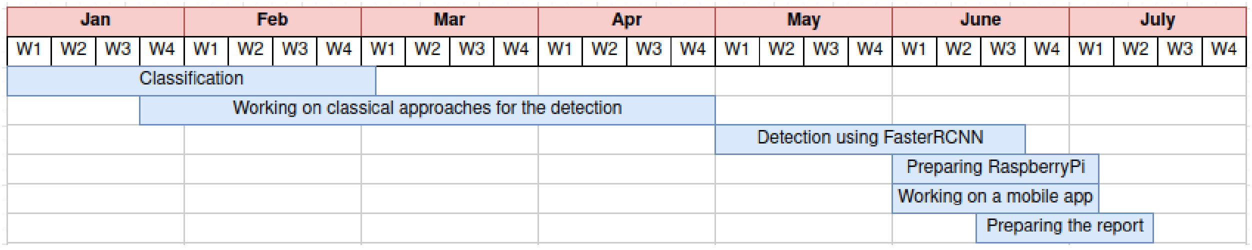


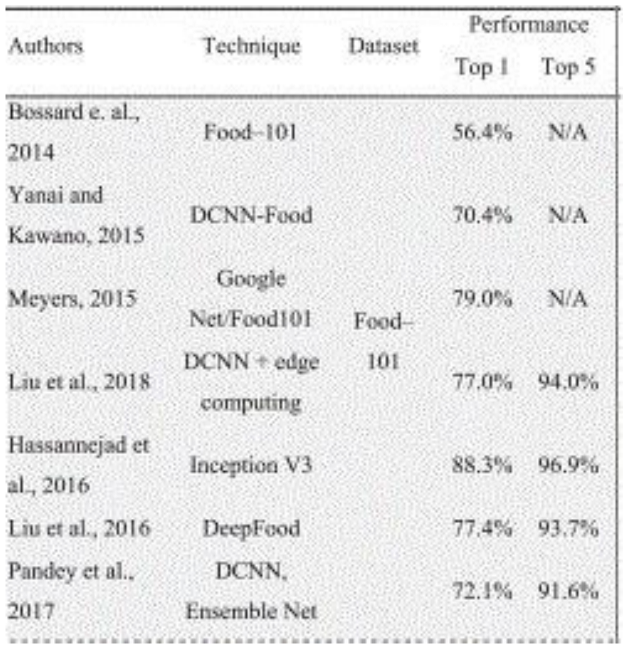
Figure 5.13 Testing Schedule

5.4. Comparative Results to Previous Work

Since our project consists of just two modules: classification and detection, we will compare our results to the results of previous work concerning the two problems.

5.4.1. Food Classification

In the food classification module, we managed to achieve accuracy around 85%. The below table shows different classification approaches on the food 101 dataset. Some exceed our accuracy such as Pouladzadeh et al which reaches 99%, and Hassannejad et al which reaches 88%. This drop in accuracy is due to several reasons, firstly, we trained using a low batch size due to capabilities of the GPU, secondly, we trained it for a shorter period for the same issue related to the GPU.



Authors	Technique	Dataset	Performance	
			Top 1	Top 5
Bossard e. al., 2014	Food–101		56.4%	N/A
Yanai and Kawano, 2015	DCNN-Food		70.4%	N/A
Meyers, 2015	Google Net/Food101	Food–101	79.0%	N/A
Liu et al., 2018	DCNN + edge computing		77.0%	94.0%
Hassannejad et al., 2016	Inception V3		88.3%	96.9%
Liu et al., 2016	DeepFood		77.4%	93.7%
Pandey et al., 2017	DCNN, Ensemble Net		72.1%	91.6%

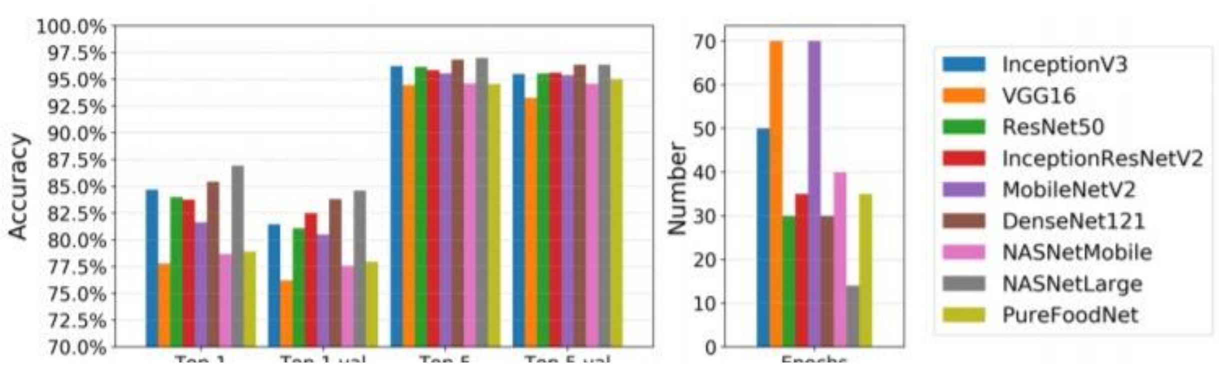
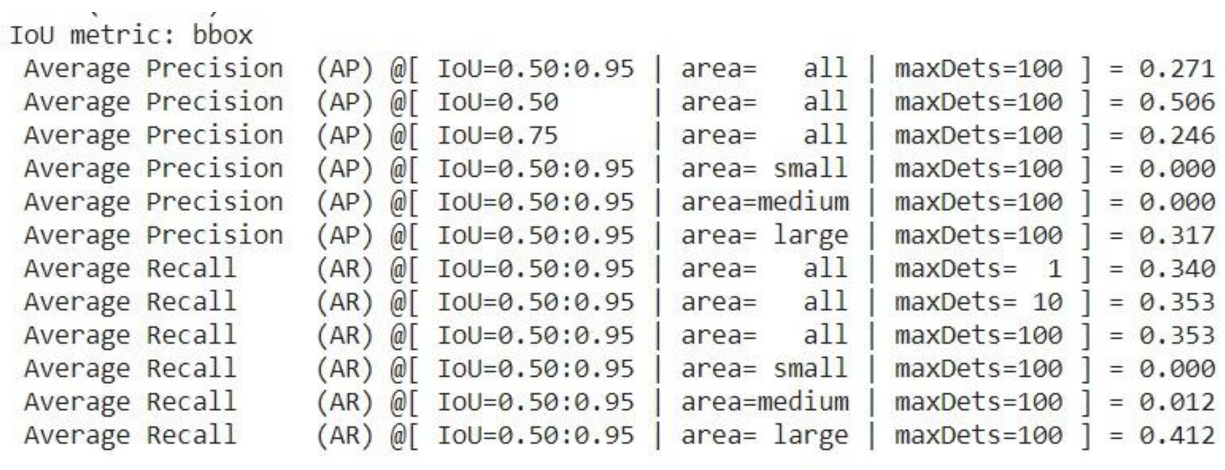


Figure 5.14 Comparative Classification Figure [11]

The diagram above shows different performances of different deep neural networks on the food101 dataset. We conclude from it that MobileNetV2 is among the best architectures when it comes to food classification. Although, it needs a large number of epochs, around 70 epochs. Its accuracy reaches almost 95% in top 5.

5.4.2. Food Detection

For the food detection module, we tested its performance by feeding into it images of dining tables and images with more than one plate and ensured that the generated bounding boxes are correctly plotted on the image. Several images were tested, images from the UECfood100 dataset and images that we gathered from our own dining tables.



```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.271
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.506
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.246
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.317
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.340
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.353
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.353
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.012
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.412
```

Figure 5.15 Detection Module Testing

The figure above shows the average precision and recall values for different intersection-over-union (IOU) values. The smallest IOU value in the figure is 0.5. After experimenting we found that an IOU value of 0.08 gives the best results. Yunsheng Shi et al. proposed a Unified Message Passing model (UniMP) that has a testing accuracy varying from 73% to 86% depending on the used dataset.

Chapter 6: Conclusions and Future Work

This chapter will state the summary of our project and we will discuss the challenges that faced us and the experiences and skills we gained throughout the project. Also, we will suggest some future work to improve our project and make it faster and flexible.

6.1. Faced challenges

One of the main challenges that faced us was finding the dataset. We could not find datasets that have a dining table and most of the datasets had food images randomly distributed without organizing them in categories. Hence we settled for using the dataset Food 101 that had only one dish per image which was better in many aspects but did not include oriental food.

We added our own oriental food to be able to have a product that serves our local customers. We ended up deleting some classes that did not or rarely exist. Unfortunately, this had to be done manually which introduced new challenges so it consumed time and effort but we succeeded to collect many classes for the oriental food and our dataset will be an extremely valuable resource for future researchers.

Another challenge we faced was the hardware as our GPU memory was not large enough to run the software during training. We had to limit the batch size. This unfortunately affected the accuracy. However, left us with a place for improvement.

Most importantly a challenge we faced and are still facing is the hardware computability of the raspberry pi/ any replacement. The processor is relatively weaker which led to a slow performance. We were able to optimize in most of the modules and the code boosted the performance by twice the speed. However, it is still not close to real-time. Other optimization solutions suggested migrating to different software tools that are lighter on the mobile devices but we didn’t have the time.

6.2. Gained Experience

In the Project we gained loads of practical experience which complemented our academic background among them to mention a sample not listing all:

- New AI approaches and how to deal with deep learning techniques.
  We have dealt with many possible challenges and problems that an AI engineer could face in real life and implemented powerful solutions and models to solve these challenges.

- Setting up hardware and using raspberry pi in any project that needs it.
  We worked with new advanced tools that were never used before, platforms, libraries in developing efficient machine and deep learning and we learned how to host machine and deep learning models and integrating them with mobile apps and hosting it on a hardware.

- Food detection techniques knowledge on which is faster and offer higher accuracy.
  We learned how to search for, approach and read state of the art papers in deep learning field and we managed to understand and implement them correctly.

- How to work on real time projects

- How to find many options for a solution how to compare and select the optimum one.

- Teamwork and how the tasks should be distributed.

6.3. Conclusions

In this report we presented a project that focused mainly on helping the visually impaired people and how to facilitate their lives. We concentrated on food recognition being a feature that does not yet exist in commercially available glasses for the blind.

We have presented several approaches to tackle and solve our problem in terms of classification and detection. Some showed great results other were good but slow. Eventually, we have chosen the deep convolutional networks for classification of the food plates and object detection of the multiple dishes per image because experimental results showed it is a good compromise for accuracy (performance) vs speed (processing time)

A prototype was implemented and testing results on a collection of 31 images containing at mostly two plates and it achieved an accuracy of 87% end-to-end including classification, detection and the hardware module.

6.4. Future Work

We recommend the continuation of the following to make the product more commercially viable:

- Optimization for the detection module as its size was a bit high compared to the other modules so this will make the detection time more less and faster than it is, In addition to the size of the memory on the hardware and software app as people always like applications with low memory high speed and performance.
- Increase the number of classes for the oriental food and make a variety of it to make the model able to detect all kinds of Egyptian food.
- Add features that will make the product more friendly and add diet options and features. This will suit people who are obsessed with fitness and healthy nutrition. One idea will be to track calories consumed throughout the day.
  This could be done via adding a look up table for each food in the module to be able to count the number of calories and macros for the food and make it more convenient.
- Deploy the application on IOS platforms to make it more usable for a more than one platform.
- Adding depth estimation in our app to estimate the quantity of the given image and be able to give a good calorie estimation for the dish. Such feature can help living a healthy lifestyle and add value to everybody’s life not just visually impaired

References

[1] Dzmitry Bahdanau, Kyunghyun Cho & Yoshua Bengio (2016). Neural Machine Translation by Jointly Learning to Align and Translate.

[2] Recio, Jorge & Fernández, Luis & Fernandez, Alfonso. (2005). Use of Gabor Filters for texture classification of digital images. Física de la tierra, ISSN 0214-4557, Nº 17, 2005, pags. 47-59.

[3] Yuen, H. & Illingworth, John & Kittler, J.. (1988). Ellipse Detection using the Hough Transform. 10.5244/C.2.41.

[4] Rebecca J. Danos, Robert H. Brandenberger (2008) Canny Algorithm, Cosmic Strings and the Cosmic Microwave Background

[5] N. Kanopoulos, N. Vasanthavada and R. L. Baker, "Design of an image edge detection filter using the Sobel operator," in IEEE Journal of Solid-State Circuits, vol. 23, no. 2, pp. 358-367, April 1988, doi: 10.1109/4.996.

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik (2013 Rich feature hierarchies for accurate object detection and semantic segmentation

[7] Ross B. Girshick, (2015) Fast R-CNN

[8] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun (2016) Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

[9] MOHAMMED AHMED SUBHI 1, (Member, IEEE), SAWAL HAMID ALI1, (Member, IEEE),AND MOHAMMED ABULAMEER MOHAMMED2 (2019) " Vision-Based Approaches for Automatic Food Recognition and Dietary Assessment: A Survey"

[10] P. Pouladzadeh, S. Shirmohammadi, and A. Yassine, ''Using graph cut segmentation for food calorie measurement,'' in Proc. MeMeA, Jun. 2014, pp. 1–6.

[11] Kiourt, C., Pavlidis, G. and Markantonatou, S., (2020), Deep learning approaches in food recognition, MACHINE LEARNING PARADIGMS - Advances in Theory and Applications of Deep Learning, Springer

[12] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah (2021) "Transformers in Vision: A Survey"

[13] Alexey Dosovitskiy∗,† , Lucas Beyer∗ , Alexander Kolesnikov∗ , Dirk Weissenborn∗ , Xiaohua Zhai∗ , Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby∗,"An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale"

[14] Bossard, Lukas and Guillaumin, Matthieu and Van Gool, Luc (2014) "Food-101 -- Mining Discriminative Components with Random Forests"

[15] Yuji Matsuda and Keiji Yanai: Recognition of Multiple Food Images by Detecting Candidate Regions, Proc. of http://foodcam.mobi/dataset.htmlIEEE International Conference on Multimedia and Expo (ICME), (2012)

[16] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen (2018) MobileNetV2: Inverted Residuals and Linear Bottlenecks

Appendix A: Development Platforms and Tools

A.1 Software Tools

1. TensorFlow

TensorFlow is an end-to-end machine learning open source platform. It has a comprehensive and flexible tools, libraries and resources that helps researchers push the state-of-the-art in Machine learning and makes it easily to develop, build and deploy ML powered applications using any platform or language. Speed or performance is not an issue in building and training state-of-the-art models as tensorflow gives the flexibility and control of features like the Keras Functional API and model subclassing API for creation of complex topologies. Moreover, it provides easy prototyping and fast debugging and execution.

2. Keras

Keras is an API not designed for machines but for human beings. Keras helps reducing cognitive load and offers consistent and simple APIs. Moreover, it decreases the number of user actions needed for common use cases, and it gives understandable and actionable error messages. Furthermore, it also has extended and detailed documentation and developer guides.

3. OpenCV

Open Source Computer Vision Library is an open source library for computer vision and machine learning. OpenCV works as a common infrastructure for computer vision applications and speeds up the use of machine perception commercially. The library has optimized algorithms which exceeds 2500, which consists of comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. Opencv helps to detect the faces and the objects, classify human actions of, track moving objects, extract 3D models of objects, stitch images together to form an entire scene of high resolution image, retrieve similar images from database, remove red eyes from images taken using flash, follow movement of eyes, etc. OpenCV has been used tremendously worldwide as it has more than 47 thousand users and number of downloads which exceeds 18 million.

4. Scikit-learn

Scikit-learn is a free software machine learning library for the Python programming language.It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

5. Pytorch

PyTorch is an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing,primarily developed by Facebook's AI Research lab (FAIR). It is free and open-source software released under the Modified BSD license. Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ interface.

6. Kivy

It is an open source python library for fast development of mobile apps as it makes use of innovative user interface. It is a cross platform framework which can run on linux, windows, Android and IOS. Moreover, it is stable, has a well-documented API and a programming guide to help beginners use kivy easily and smoothly. The toolkit comes with highly extensible widgets which are 20 widgets and more. Also many parts are written in C using Python and tested with regression test.

7. NumPy

Numpy is the crucial package for scientific computing using Python as It offers numerical computing tools such as comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and much more. Also NumPy supports a wide range of hardware and computing platforms, and plays well with GPU, and sparse array libraries. Moreover, Numpy is a high level syntax which makes it accessible, easier and productive for programmers considering any background or experience level.

8. Flutter

Flutter is an open-source UI software development kit created by Google. It is used to develop cross platform applications for Android, iOS, Linux, Mac, Windows, Google Fuchsia, and the web from a single codebase.

A.2 Hardware Tools

1. Raspberry pi

Raspberry Pi OS (formerly Raspbian) is a Debian-based operating system for Raspberry Pi. Since 2015, it has been officially provided by the Raspberry Pi Foundation as the primary operating system for the Raspberry Pi family of compact single-board computers.The first version of Raspbian was created by Mike Thompson and Peter Green as an independent project.The initial build was completed in June 2012.

2. CUDA

Cuda (an acronym for Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model created by Nvidia. It allows software developers and software engineers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing – an approach termed GPGPU (general-purpose computing on graphics processing units). The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels. The CUDA platform is designed to work with programming languages such as C, C++, and Fortran. This accessibility makes it easier for specialists in parallel programming to use GPU resources, in contrast to prior APIs like Direct3D and OpenGL, which required advanced skills in graphics programming. CUDA-powered GPUs also support programming frameworks such as OpenMP, OpenACC and OpenCL

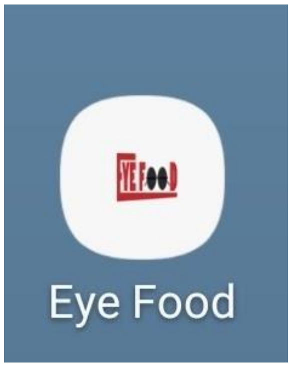
3. Google Colab

Google Colab is a free and open source groupware suite. It consists of the Colab server and a wide variety of Colab clients, including KDE PIM-Suite Kontact, Round cube web frontend, Mozilla Thunderbird and Mozilla Lightning with sync Colab extension and Microsoft Outlook with proprietary Colab Connector Plugins. The special idea behind colab is the use of IMAP as an underlying protocol not only for email, but also for contact and calendar entries. These entries are saved in special IMAP-folders, utilising the Colab XML-format and the storage and access rights are controlled by IMAP-server. LDAP takes care of the configuration and maintenance of Colab.

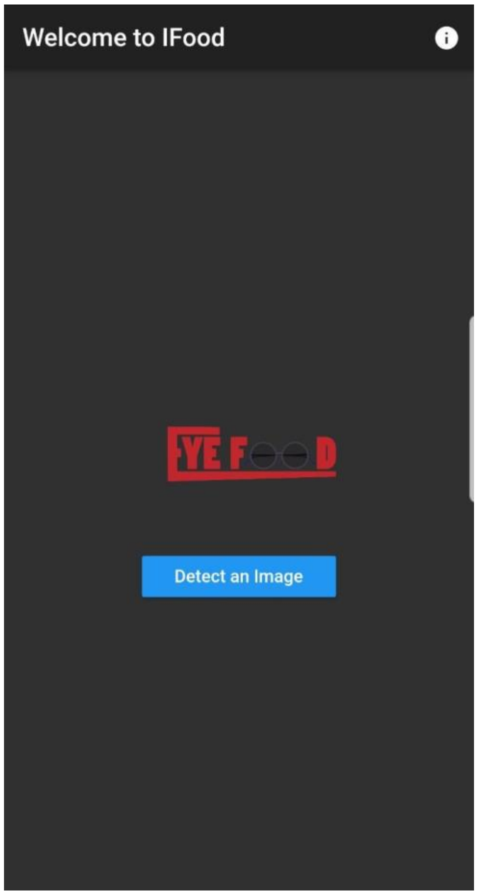
Appendix B: User Guide

For the android app, After installing the APK file, this part explains how to use Eyefood:

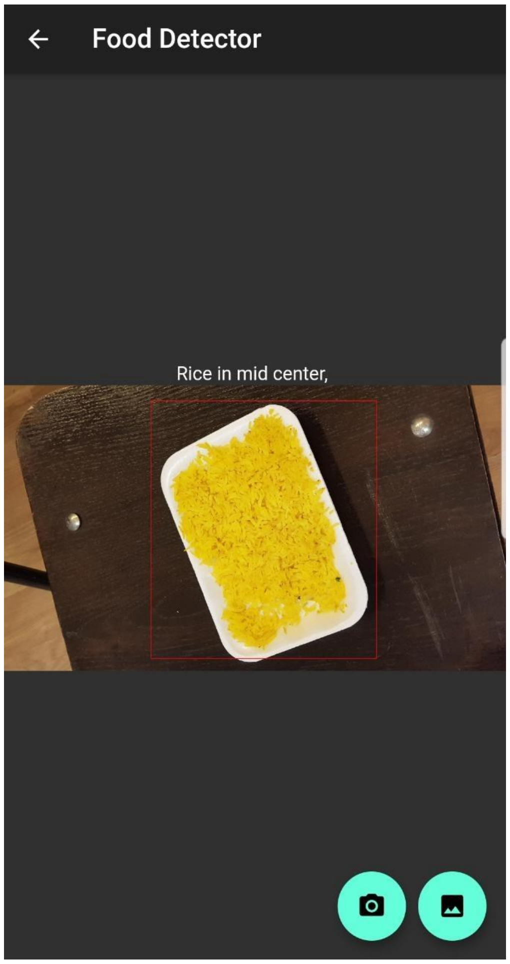
1. The figure below shows the application appearance on the device:



2. The first screen shows the logo with a button below it to detect images:

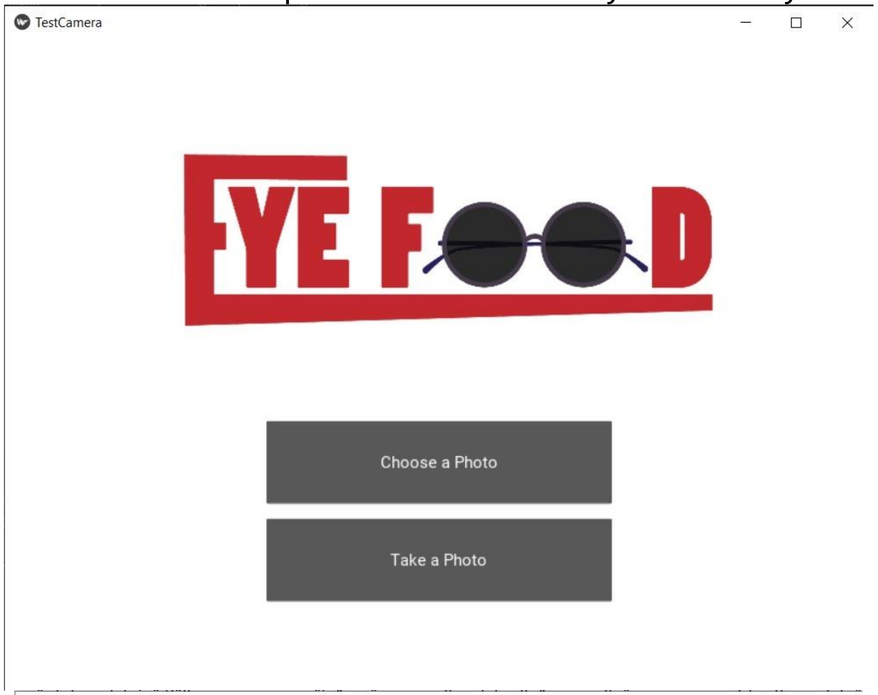


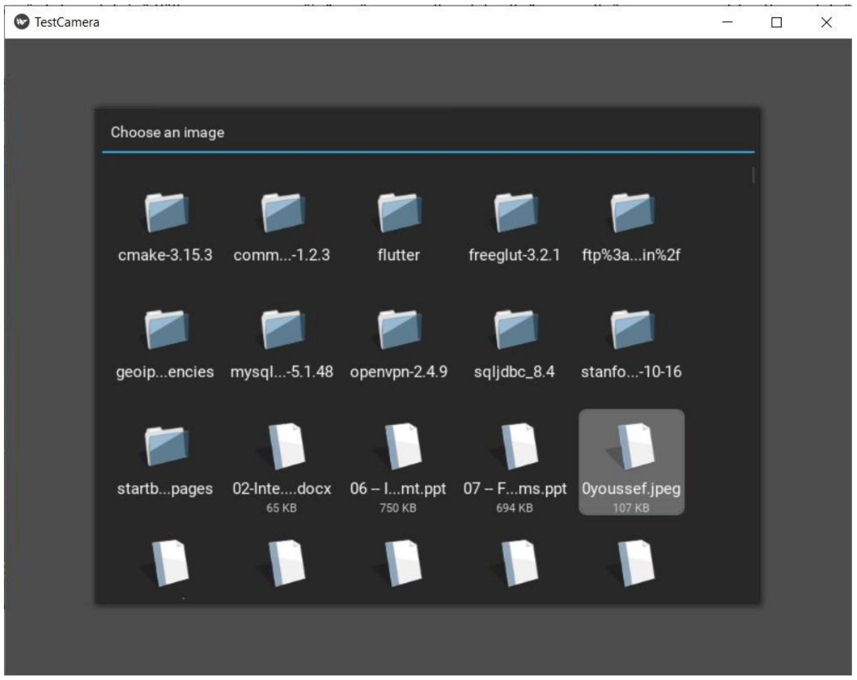
3. After choosing the image or capturing it wait for moment for the server to respond.



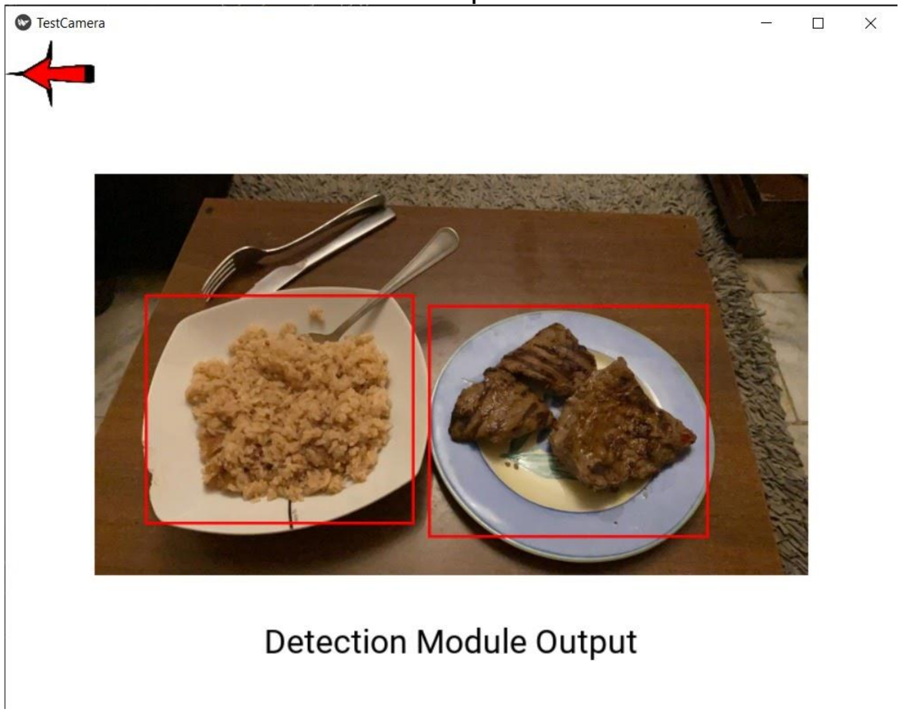
For the desktop app,

1. Choose whether to take a photo or use one from your directory.

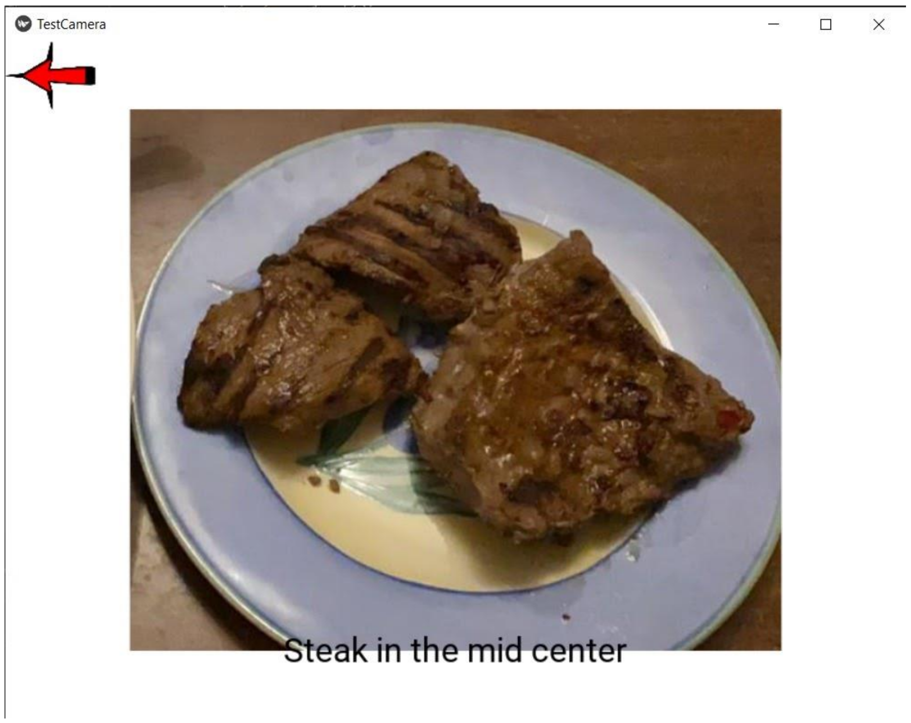


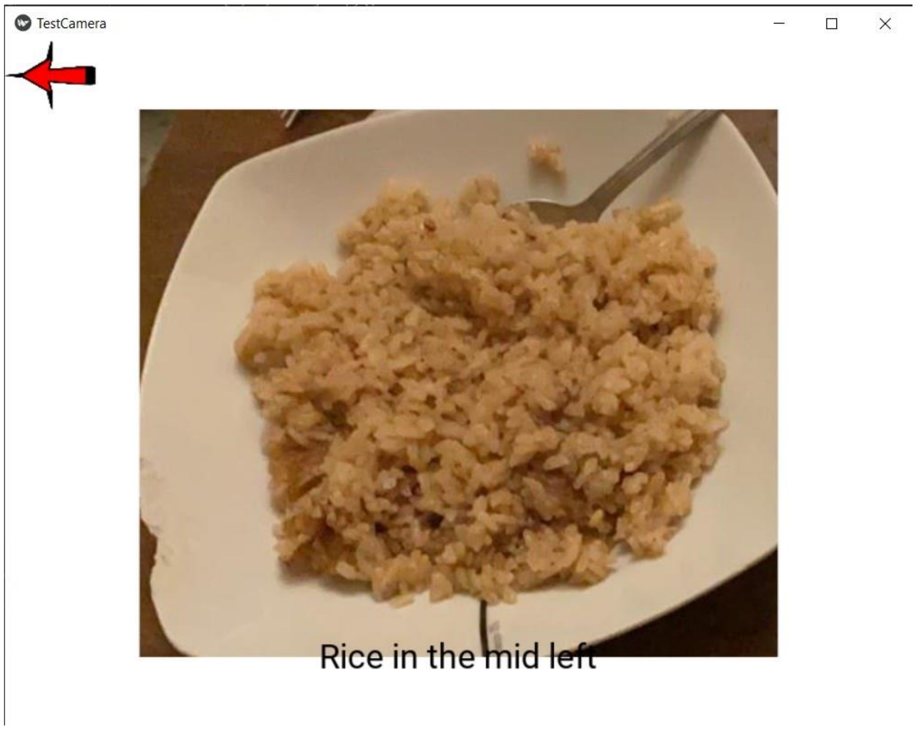


2. Wait for a moment for the server to respond.



3. Swipe right to see classification output.




TestCamera
Rice in the mid left

Appendix C: Feasibility Study

A. Financial feasibility

Hardware applications like **Eye food** need capital cost for its materials. The hardware consists of a glasses frame which will need to be modified accordingly to each user. One of many proposed solutions is making a deal with an outsourcing company specialized in glasses frames like Maghrabi which will allow the user to have multiple styles and will also support our product and market it. Other hardware requirements include an 0.8W mini speaker that will be attached on the glasses frame. In addition to that, we will need to hire employees that will assemble and work on testing the production line. Moreover, the business team has a strategy of advertisement that will need financial aid to support the ads. Most of these expenses will make some loss at the early stages of the project but then we will start to sale the product to reach our target.

B. Technical feasibility

Eye Food is hardware product that offers food detection feature which allows you to detect the dish in front of you and its detailed location. It consists of the main hardware unit and the software module. The hardware choice as a proof of concept is the Raspberry Pi 4B. Raspberry pi has been in the market for almost a decade now and they introduce various solutions that are versatile enough to be used in almost any small project. The problem with the Pi 4B is solely its size. PI introduces a smaller sibling names Raspberry Pi Zero with the dimensions that can fit very well in a smart glasses frame. As for the software module, we developed our models using tensorflow, tensorflowlite and pytorch. The latter is supported by Facebook AI and has a wide community of users that share their knowledge and faced challenges. Tensorflow and TensorflowLite are developed and maintained by Google. These credible providers add credibility to our solution and ensure the maintainability and the support on the long run.

C. Legal Feasibility

In Eye-Food since there is a camera module, we take our privacy measures very seriously. Eye-food doesn't save the images on the device nor it keeps any privacy risking details in the log files. Our project is legally acceptable and the name meets all of the requirements, laws and regulations needed. It is worth mentioning that the libraries we use are free and open source.

D. SWOT Analysis

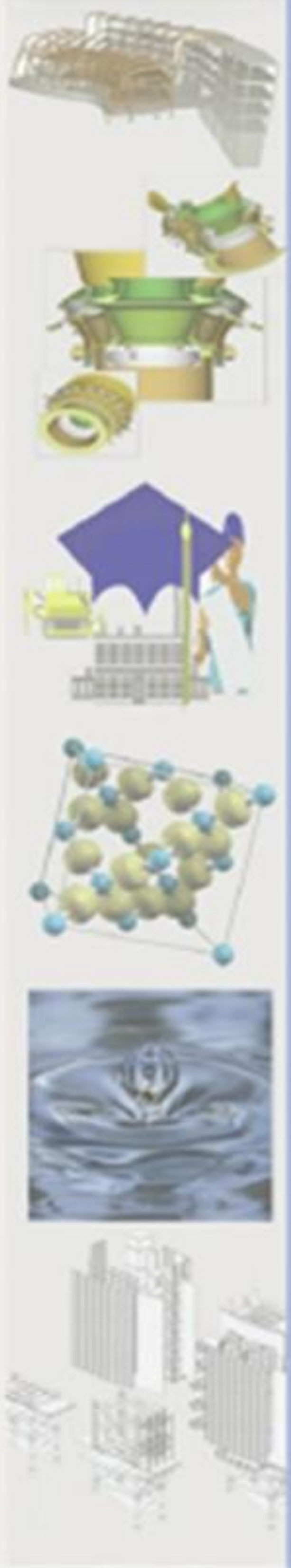
The table A.1 shows the Analysis for Eye Food project from the point of view of its strengths, weaknesses, opportunities and threats.

Table A.1

Strength	**Weaknesses**
• The regional market doesn't have strong competitors. • International competitors sell the same product with very high prices.	• It may be damaged easily if it falls. • Currently, the hardware is relatively slow • It can misclassify a class causing disturbance for the user.
Opportunities	**Threats**
• Visually impaired people in Egypt are not well taken care of by the government. • Restaurants welcome the idea and start doing marketing and offers for visually impaired people. • Restaurants may help as gather more oriental food images to enrichen the dataset and add support for more local food.	• Sponsors terminate their contracts with us. • Not having a pro marketing strategy.

Sponsor

Valeo



AET Architectural Engineering and Technology

CCE Communication and Computer Engineering

CEM Construction Engineering and Management

MDE Mechanical Design Engineering

PPC Petro Chemical Engineering

STE Structural Engineering

WEE Water Engineering and Environment