RECIPE RECOMMENDATION WEBAPP AND IMAGE CLASSIFICATION

A project report submitted in partial fulfillment of the requirements for the degree of

Bachelor of Engineering

By

Aarti Bandekar (8296)

Anjana Singh (8304)

Mounita Bhagat (8453)

Under the guidance of

Prof. Swapnali Makdey

DEPARTMENT OF ELECTRONICS ENGINEERING

Fr. Conceicao Rodrigues College of Engineering

Bandra (W), Mumbai - 400050

University of Mumbai

May 10th , 2021

Certificate

This is to certify that the project entitled **"RECIPE RECOMMENDATION WEB APP AND IMAGE CLASSIFICATION"** is a bonafide work of **Aarti Bandekar (8296), Anjana Singh (8304), Mounita Bhagat (8453)** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of **Undergraduate** in **Electronics Engineering**.

Prof. Swapnali Makdey

Supervisor/ Guide

Dr. Sapna Prabhu

Head of Department

Dr. S. Unnikrishnan

Principal

PROJECT REPORT APPROVAL

This project report entitled ***RECIPE RECOMMENDATION WEB APP AND IMAGE CLASSIFICATION*** *by*

Aarti Bandekar (Roll No: 8296)

Anjana Singh (Roll No: 8304)

Mounita Bhagat (Roll No: 8453)

is approved for the degree of Bachelor of Engineering.

Examiners

1.----------

2.----------

Date:

Place:

Declaration

We declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Aarti Bandekar (Roll No: 8296): _____

Anjana Singh (Roll No: 8304): _____

Mounita Bhagat (Roll No: 8453): _____

Date:

Abstract

A recommendation system is useful in filtering a wide range of information by narrowing down the amount of information to only the relevant content. The main objective of this proposed application is to suggest to the user a preferred recipe using an ingredient-based filtering algorithm. The web application provides the user with a list of recipes based on their choice of ingredients which gives users nutritious recipe choices that will also satisfy their personal taste. This application helps to provide Customers with a User-friendly application which helps them to select the required options according to their preferences. The process of identifying recipes from an image is quite an interesting field with various applications. Since food monitoring plays a leading role in health-related problems, it is becoming more essential in our day-to-day lives. In this project, an approach has been presented to classify images of food using convolutional neural networks. Unlike the traditional artificial neural networks, convolutional neural networks have the capability of estimating the score function directly from image pixels. In our project we have made use of two models: Xception and MobileNet, which gave us the accuracy of 98.88% and 94.13% for the 5 classes of food101 dataset. We have used mx pooling function for the data, and the features extracted from this function are used to train the network.

Acknowledgments

We have great pleasure in presenting the report on "RECIPE RECOMMENDATION WEB APP AND IMAGE CLASSIFICATION". We take this opportunity to express our sincere thanks to the guide Prof. Swapnali Makdey, C.R.C.E, Bandra (W), Mumbai, for providing the technical guidelines, and suggestions regarding the line of this work. We enjoyed discussing our project's progress with ma'am. We thank Dr. Sapna Prabhu, head of Department (Electronics Engineering), our principal, and the management staff of C.R.C.E., Mumbai for encouragement and providing the necessary infrastructure for pursuing the project. We also thank all the non-teaching staff for their valuable support, help us complete our project.

Aarti Bandekar (8296)

Anjana Singh (8304)

Mounita Bhagat (8453)

Date: May 10th , 2021

Table of Contents

List of Figures and Tables

Chapter 1
Introduction

In modern times, heavy workloads and busy lives often cause people to neglect their health by being careless with what they eat. People are faced with a limitless selection of meals which is often a constraint in searching for recipes especially with students and working class adults. This happens because preparing meals requires people to search, think, prepare, and even learn new recipes, but they're still at a risk of being frustrated as the time consumed in this entire process is spent being unproductive, especially considering the time constraints that these individuals face. People also seem to have varied cuisine preferences, therefore, it would be commendable to have a system that recommends people personalized meals based on the ingredients they have at their disposal.

Moreover, being in the middle of a deadly global pandemic has rendered even simple errands like fetching groceries and pantry ingredients difficult and dangerous. It has become imperative that we put the things we already have in our kitchens to use and make healthy, sustainable meals out of them. Our web application aims to help solve this problem by providing a functionality that lets users enter the ingredients they have in a search bar and get recipe results based on them. In addition to that, we have also trained and tested a machine learning model that classifies food images into five different categories, based on the customized Food101 dataset. We have used Convolution Neural Network. CNNs are used for image classification and recognition because of its high accuracy. The CNN follows a hierarchical model which works on building a network, like a funnel, and finally gives out a fully-connected layer where all the neurons are connected to each other and the output is processed. CNNs are fully connected feed forward neural networks. CNNs are very effective in reducing the number of parameters without losing on the quality of models. Images have high dimensionality (as each pixel is considered as a feature) which suits the above described abilities of CNNs. Our model gives us high accuracy and precise results. Food images dominate across social media platforms and drive restaurant selection and travel, but are still fairly unorganized due to the sheer volume of images. Utilized correctly, food image classification can improve food experiences across the board, such as to recommend dishes and new eateries, improve cuisine lookup, and help people make the right food choices for their diets. In this project we explore the problem of food image classification through training convolutional neural networks, both from scratch and with pre-trained weights learned on a larger image dataset (transfer learning), achieving an accuracy of 98.88% and accuracy of 94.13%.

Chapter 2

Literature Survey

Table 2.1: Personalized web-based application for movie recommendations

Title of Papers	Journal Publications	Description	Reference
Personalized web-based application for movie recommendations	Killian Duay Bachelor's Thesis Degree program in Business Information Technology 2019	The aim of this thesis was first to study Machine Learning and the ways it can be used in the case of a movie recommender system. Then the aim was to implement one of those solutions in a project of a web-based application for movie recommendations. The main goal of the project was to develop a fully functional web application that allows users to get recommendations on all existing movies in the world. The application will be tested by real users. The application will have a ReactJS frontend (web-based and responsive design), a Python backend, and a Firebase Database and Authentication. The movies and their information will be fetched on the TMDB API that provides data on all existing movies and which provides a limited number of data for free and the remaining data is paid in case the user wants to make a project with a huge amount of data in their data set.	https://www.theseus.fi/bitstream/handle/10024/172583/KillianDuay_thesis.pdf?sequence=2&isAllowed=y

Table 2.2: Recipe Recommendation System using Machine Learning Models

Title of Papers	Journal Publications	Description	Reference
Recipe Recommendation System using Machine Learning Models	1 Suyash Maheshwari and 2 Manas Chourey. Student (B.E.), Information Technology, Sinhgad College Of Engineering, Pune, India. 2nd Student (B.E.), Information Technology, Sinhgad College Of Engineering, Pune, India	Here, they have used two machine learning models-vector space model and the Word2Vec model to find top ingredient pairs from different cuisines and to suggest alternate ingredients. The focus is on Indian cuisine. Indian cuisine is very vast and diverse and hence it is difficult to find patterns and generate pairs. Due to the increasing availability of data in online databases, data mining and machine learning methods are starting to play a prominent role in food consumption analysis and food preference modeling. It also helps people who are allergic to certain ingredients by recommending alternate ingredients. Our project is using different technologies for the front end and as well as back end. And the sorting system is based only on ingredients and not on the different types of cuisines.	https://www.irjet.net/archives/V6/i9/IRJET-V6I946.pdf

Table 2.3: Discovery of Recipes Based on Ingredients using Machine Learning

Title of Papers	Journal Publications	Description	Reference
Discovery of Recipes Based on Ingredients using Machine Learning	S. Praveen 1, M.V. Prithvi Raj 2, R. Poovar San3, V. Thiruvengadam 4, M. Kevin Kumar 5 1,2,3,4B.Tech, Dept. Of Information Technology, Valliammai Engineering College, Chennai, Tamil Nadu 5Asst. Prof, Dept. Of Information Technology, Valliammai Engineering College, Chennai, Tamil Nadu	The problem is, the user cannot identify what dishes can be cooked by using the ingredients available by the user. To overcome these problems, a Machine Learning approach was used in this project which enables the user to suggest the recipes based on the available ingredients by the user. This way of searching makes the user, the selection of recipes in a smarter way, and makes the household food maker easier. The objective is to reduce the selection of recipes in an easier way by using ingredients as input.	https://www.irjet.net/archives/V6/i3/IRJET-V6I328.pdf

Table 2.4: Food Detection and Recipe Recommendation

Title of Papers	Journal Publications	Description	Reference
Food Detection and Recipe Recommendation	Jay Kukreja1 , Kirtishil Patil2 , Aditi Navgire3 , Kiran Yele4 Of Computer Engg Department, VIIT, Pune, India And Prof . Aniket Katade 5 Of Professor, Computer Engg Department, VIIT, Pune, India5	In the given paper they have designed a mobile recipe recommendation system using image processing for various recipes and display the name of the recipe and the list of ingredients needed for cooking the recipe . The proposed system carries out recipe detection on the given dish or fruits and other recipes in a realtime way on an Android-based smart phone, and recommends cooking recipes and detects ingredients related to the recognized food recipe. By only pointing the native camera of a mobile device to the food which is placed on the dish, the user can obtain a recipe and ingredient list instantly. After detection of food, the user gets the details and facts of ingredients and recipe procedure as per the food image. Tensor-flow library is used to detect the recipe as well as the ingredients using Tensor-flow Lite library.	http://www.ijasret.com /VolumeArticles/FullTextPDF/457_3.FOOD_DETECTION_AND_RECIPE_RECOMMENDATION.pdf

Table 2.5: Recipe Recommendation System

Title of Papers	Journal Publications	Description	Reference
Image to Recipe translation with Deep convolutional Neural Networks	Muriz Serifovic	In this article we will look at how to train deep convolutional neural networks with Keras to *classify images into food categories* and to *output a matching recipe*. The dataset contains >800'000 food images and >300'000 recipes from chefkoch.de.Hardly any other area affects human well-being to a similar extent as nutrition. Every day countless of food pictures are published from users on social networks; from the first home-made cake to the top Michelin dish, the joy is shared with you in case a dish was successfully cooked.It is a fact that no matter how different you may be from each other, good food is appreciated by everyone.Advances in the classification of individual cooking ingredients are sparse. The problem is that there are almost no public edited records available. This work deals with the problem of automated recognition of a photographed cooking dish and the subsequent output of the appropriate recipe. The distinction between the difficulty of the chosen problem and previous supervised classification problems is that there are large overlaps in food dishes (aka high intra-class similarity), as dishes of different categories may look very similar only in terms of image information.	https://towardsdatascience.com/this-ai-is-hungry-b2a8655528be

Table 2.6: Image to Recipe translation with Deep Convolutional Neural Networks

Title of Papers	Journal Publications	Description	Reference
A Cooking Recipe Recommendation System with Visual Recognition of Food Ingredients	Keiji Yanai, Takuma Maruyama and Yoshiyuki Kawano The University of Electro-Communications, Tokyo, Japan	In this paper, a cooking recipe recommendation system is proposed which runs on a consumer smartphone as an interactive mobile application. The proposed system employs real-time visual object recognition of food ingredients, and recommends cooking recipes related to the recognized food ingredients. Because of visual recognition, by only pointing a built-in camera on a smartphone to food ingredients, a user can get to know related cooking recipes instantly. The objective of the proposed system is to assist people who cook to decide a cooking recipe at grocery stores or at a kitchen. In the current implementation, the system can recognize 30 kinds of food ingredients in 0.15 seconds, and it has achieved the 83.93% recognition rate within the top six candidates. By the user study, we confirmed the effectiveness of the proposed system.	https://online-journals.org/index.php/i-jim/article/view/3623

Table 2.7: A Cooking Recipe Recommendation System with Visual Recognition of Food Ingredients

Title of Papers	Journal Publications	Description	Reference
A Cooking Recipe Recommendation System with Visual Recognition of Food Ingredients	Keiji Yanai, Takuma Maruyama and Yoshiyuki Kawano The University of Electro-Communications, Tokyo, Japan	In this paper, a cooking recipe recommendation system is proposed which runs on a consumer smartphone as an interactive mobile application. The proposed system employs real-time visual object recognition of food ingredients, and recommends cooking recipes related to the recognized food ingredients. Because of visual recognition, by only pointing a built-in camera on a smartphone to food ingredients, a user can get to know related cooking recipes instantly. The objective of the proposed system is to assist people who cook to decide a cooking recipe at grocery stores or at a kitchen. In the current implementation, the system can recognize 30 kinds of food ingredients in 0.15 seconds, and it has achieved the 83.93% recognition rate within the top six candidates. By the user study, we confirmed the effectiveness of the proposed system.	https://online-journals.org/index.php/i-jim/article/view/3623

Chapter 3

Recipe Recommendation Web Application

3.1 Block Diagram

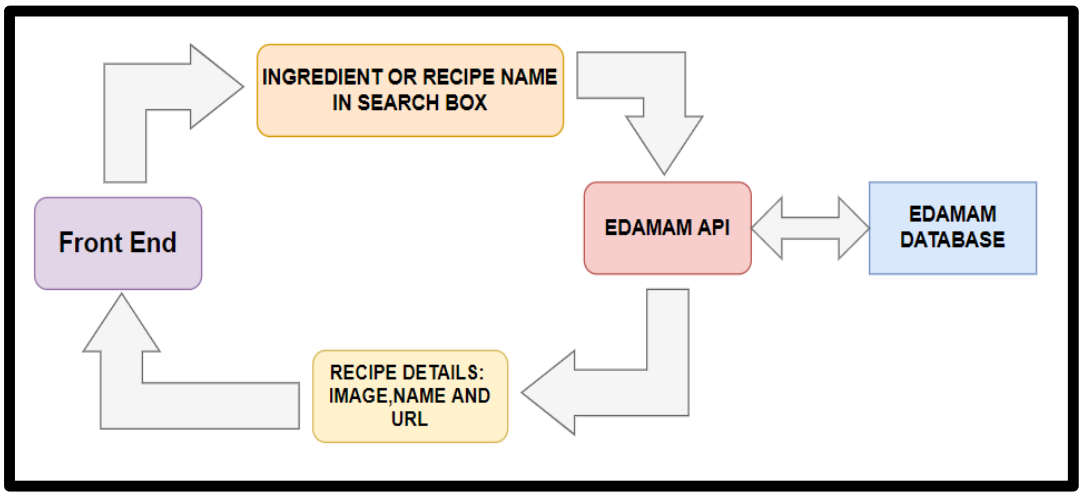


Fig 3.1: Web Application Block Diagram

3.2 Working

The technologies used to make this project are the following:

FrontEnd - Bootstrap

Bootstrap: It is used to Quickly design and customize responsive mobile-first sites with Bootstrap, the world's most popular front-end open source toolkit, featuring Sass variables and mixins, responsive grid system, extensive pre-built components, and powerful JavaScript plugins Bootstrap is used to design the FrontEnd of the application to provide a clean and sophisticated user interface on the Client side. The Recipes or Ingredients entered in the SearchBox are sent to the BackEnd which then forwards it to the Edamam Database through the Edamam API. It then fetches the relevant recipes back for us and displays the output on the FrontEnd. The Output consists of the Recipe Image, Title and URL.

BackEnd - JQuery, JavaScript ES6, Axios, Edamam API

Javascript ES6: JavaScript ES6 brings new syntax and features to make the code more modern and more readable. It allows you to write less code and do more. ES6 introduces us to many great features like arrow functions, template strings, class destruction, Modules, Arrow functions, template literals, array and object destruction.

jQuery**:** jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

Axios: Promise based HTTP client for the browser and node.js. Features include:

1. Make XMLHttpRequests from the browser
2. Make http requests from node.js
3. Supports the Promise API
4. Intercept request and response
5. Transform request and response data
6. Cancel requests
7. Automatic transforms for JSON data
8. Client side support for protecting against XSRF

Promise based: Returns promises rather than accepting callbacks.

Promise is a JavaScript object that links producing and consuming code. It supports two properties called state and result. While a promise object is "pending", the result is undefined, when fulfilled; the result is a value, when rejected; the result is an error object.

Promises make error handling across multiple asynchronous calls much more effortless than using callbacks. This in turn makes the code look much cleaner.

Benefits of Promises:

1. Improves Code readability
2. Better handling of asynchronous operations
3. Better flow of control definition in asynchronous logic

CallBack:

A CallBack is a function passed on as an argument to another function.

CallBacks are best used in asynchronous functions, where one function has to wait for another function (for instance, waiting for a file to load).

Recipe Search API by Edamam: The Edamam B2B API is accessed by sending HTTPS requests on specific URLs as described below. The base URL is https://api.edamam.com, and you obtain the full URL by appending request's path to the base URL, for example, https://api.edamam.com/search.On success, the API returns HTTP code 200 OK and the body contains the result of the query in JSON format. In case of errors, the API returns an error code (e.g. 404 NOT FOUND).

We use the EDAMAM API to access their database of recipes.

An **API** is a set of programming code that enables data transmission between one software product and another. It also contains the terms of this data exchange.

Application programming interfaces consist of two components:

1. Technical specification describing the data exchange options between solutions with the specification done in the form of a request for processing and data delivery protocols.
2. Software interface written to the specification that represents it.

The software that needs to access information (i.e., X hotel room rates for certain dates) or functionality (i.e., a route from point A to point B on a map based on a user's location) from another software, calls its API while specifying the requirements of how data/functionality must be provided. The other software returns data/functionality requested by the former application.

And the interface by which these two applications communicate is what the API specifies.

Each API contains and is implemented by **function calls** – language statements that request software to perform particular actions and services. Function calls are phrases composed of verbs and nouns, for example:

1. Start or finish a session
2. Get amenities for a single room type
3. Restore or retrieve objects from a server.

Function calls are described in the API documentation. APIs serve numerous purposes. Generally, they can simplify and speed up software development. Developers can add functionality (i.e., recommender engine, accommodation booking, image recognition, payment

processing) from other providers to existing solutions or build new applications using services by third-party providers. In all these cases, specialists don't have to deal with source code, trying to understand how the other solution works. They simply connect their software to another one. In other words, APIs serve as an abstraction layer between two systems, hiding the complexity and working details of the latter.

3.3 Code Explanation

Index.html

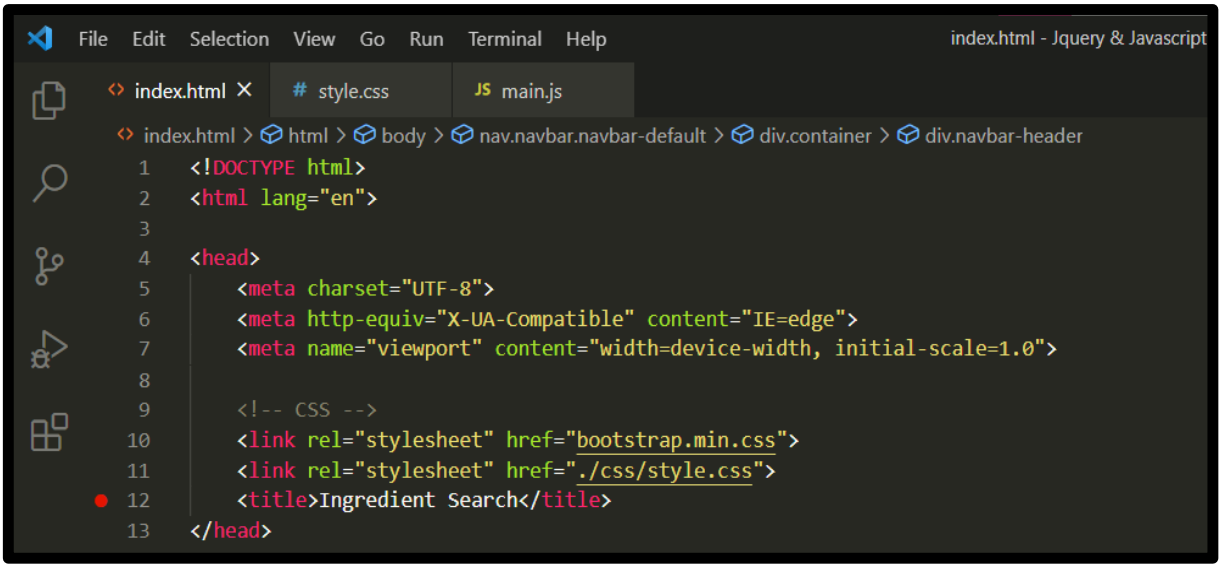


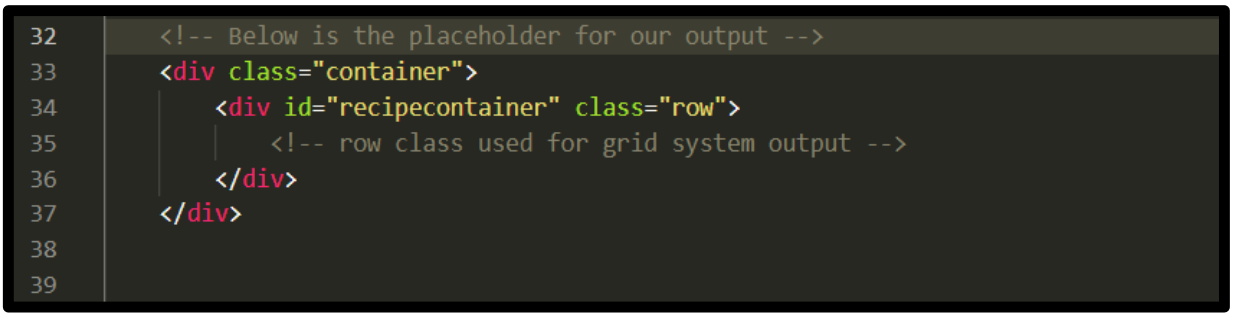
Fig 3.2: Index.html1

1. We set the standard language to English for the browser to recognize it.
2. In the Header Tag, we also link our bootstrap link and our stylesheet for custom edits. We use a customized Bootstrap Theme from Bootswatch for our User Interface.

```
<body>
    <nav class="navbar navbar-default">
        <div class="container">
            <div class="navbar-header">
                <a class="navbar-brand" href="index.html">Ingredient Search</a>
            </div>
        </div>
    </nav>

    <div class="container">
        <div class="jumbotron">
            <h3 class="text-center">Search For Recipes</h3>
            <form id="searchForm">
                <input type="text" class="form-control" id="searchText" placeholder="Search Recipes">
            </form>
        </div>
    </div>
```

Fig 3.3: Index.html2

3. We define the classes **navbar**, **navbar-default**(for the mint green shade in UI) ,**navbar-header**(For the name our webapp) and **navbar-brand** to style our NavBar, these Bootstrap themed classes are included with Bootswatch for ease of styling
4. The class named **Text-Center**, centers the title in our Jumbotron.A **jumbotron** indicates a big box for calling extra attention to some special content or information. A **jumbotron** is displayed as a grey box with rounded corners. It also enlarges the font sizes of the text inside it.
5. We set the ID to **searchForm** to connect our form to our Javascript file.
6. We set the ID **searchText** to extract the search terms entered by the user in the search box to send it further through the Edamam API.
7. The **placeholder** attribute specifies a short hint that describes the expected value of an input field (e.g. a sample value or a short description of the expected format). The short hint is displayed in the input field before the user enters a value.



```
    <!-- Below is the placeholder for our output -->
    <div class="container">
        <div id="recipecontainer" class="row">
            <!-- row class used for grid system output -->
        </div>
    </div>
```

Fig 3.4: Index.html3

8. This final container of our HTML file is used to display the final output on the screen, it is connected to our Javascript file through the ID **recipecontainer**.
9. The class **row** ensures that the output images are displayed in a row that fits the overall container dimensions.

```
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"
        integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4=" crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
    <script src="js/main.js"></script>
</body>

</html>
```

Fig 3.5: Index.html4

10. The scripts are always linked in the end to make sure they run after the elements have all loaded. Here, we link jQuery, axios and our own custom main.js file.

Main.js

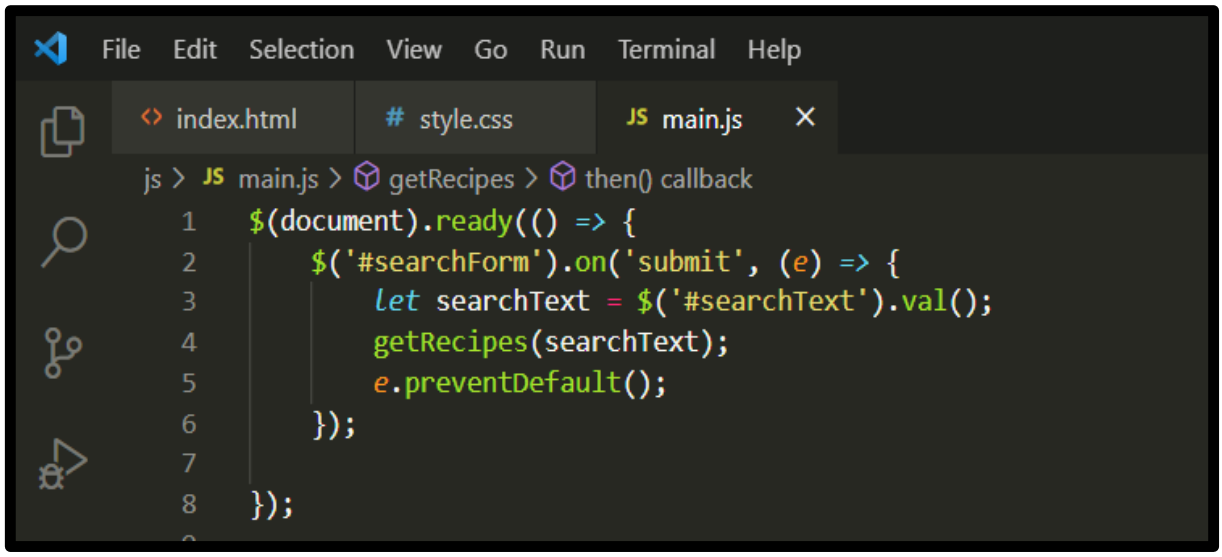


Fig 3.6: Main.js1

1. **$(document).ready()**: jQuery detects the state of readiness for us. This line of code will only run once the page DOM(Document Object Model) is ready for JS Code to execute. This will prevent any jQuery code from running before the document has finished loading.

2. On ID **searchForm** we're listening for an event 'submit' then we will have a callback and prevent the Default event to stop from submitting the form. It will stop the form from actually submitting to a file and prevent a link from submitting the following URL.
3. The value entered in the searchBox is sent as a request to the API, which then brings back recipes that match that query. The text entered in the searchBox is stored in the variable named **searchText**, we then call the function **getRecipes** and pass that value.

```
function getRecipes(searchText) {
    axios.get('https://api.edamam.com/search?app_id=fe3fdc46&app_key=1000e1ecdac75f004873cf6bfacb27e7&q=' + searchText)
        .then((response) => {
            console.log(response);
            let recipes = response.data.hits;
            let output = '';
            $.each(recipes, (index, recipehit) => {
                output += `
                <div class="col-md-3">
                    <div class="well text-center">
                    <img src="${recipehit.recipe.image}">
                    <h5>${recipehit.recipe.label}</h5>
                    <a class="btn btn-primary" href="${recipehit.recipe.url}"> Recipe Details</a>
                    </div>
                </div>
                `;
            });
            // <a onclick="recipeSelected('${recipehit.recipe.url}')" class="btn btn-primary" href="#"> Recipe Details</a>
            //
            $('#recipecontainer').html(output);
        })
        .catch((err) => {
            console.log(err);
        });
}
```

Fig 3.7: Main.js2

4. Let us now define the function getRecipes, where the variable searchText is passed as an argument.
5. A **GET request** can be made with Axios to "get" data from a server. The HTTP get request is performed by calling **axios.get()**.
6. The get() method requires two parameters to be supplied to it. First, it needs the URI of the service endpoint. Second, it should be passed an object that contains the properties we want to send to our server. The promise-based version of this is used in our code.
7. Axios requests are promises, which means they have a then() function for promise chaining, and a catch() function for handling errors. Axios' catch() behaves exactly the

same as the promise catch() function. So, you can use promise chaining, and add a catch() at the end to handle any errors that occur in the promise chain.

8. **$.each(recipes , (index, recipehit)):** We need to loop through the **hits** array and then append each recipe onto the **output** variable, we will then output it all onto the screen hence a jQuery each loop is used here to iterate through each recipe.
9. The **Title: ${recipehit.recipe.label}** and **Image: <img src="${recipehit.recipe.image}">** of each recipe is accessed from the API.
10. An **anchor tag** with an onclick event is made, when the button is clicked, it is going to redirect us to the web page the URL points to.
11. **$('#recipecontainer').html(output)**: The output will now be displayed on screen using the ID **recipecontainer**.

Style.css

Fig 3.8: Style.css

1. The output images are set with a **width:100%** to contain it within its element.
2. **@media(min-width:960px):** A **media query** is also defined for making our web app responsive for mobile view. When the **min-width** goes below **960px** the height of the container and image are resized to fit the mobile view.

Postman: Postman is a popular API client that makes it easy for developers to create, share, test and document APIs. This is done by allowing users to create and save simple and complex HTTP/s requests, as well as read their responses. The result – more efficient and less tedious work.

Postman is very convenient when it comes to executing APIs. Once you've entered and saved them, you can simply use them over and over again, without having to remember the exact endpoint, headers, API keys, etc.

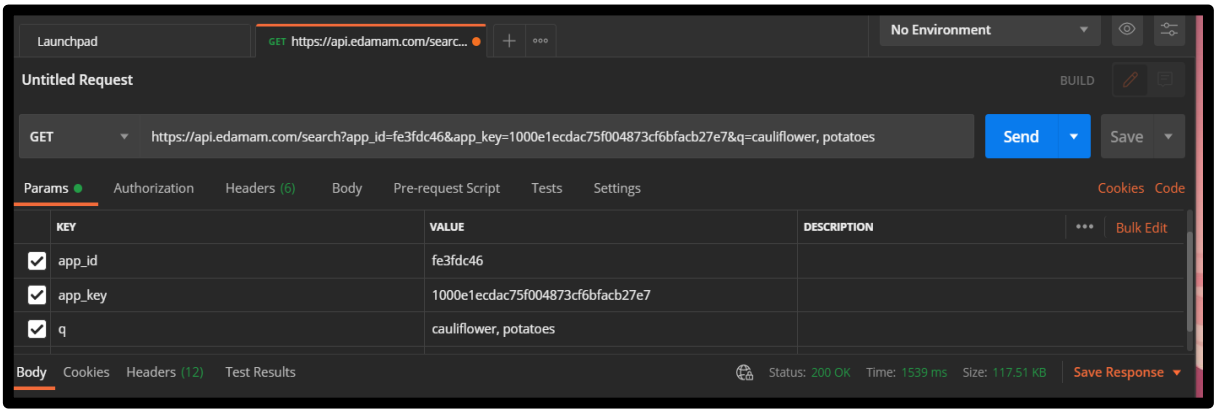


Fig 3.9: Postman1

1. Enter the API endpoint where it says 'Enter request URL' and select the method (the action type) on the left of that field. The default method is GET but we will use POST in the example below.
2. Add authorization tokens/credentials according to the server side requirements.
3. Enter headers as required. In our case that is the **app_id**, **app_key** and **q**.
4. We can execute this API now, hit the 'Send' button, which is located to the right of the API request field. You can also click on the 'Save' button beside it to save that API request to your library.

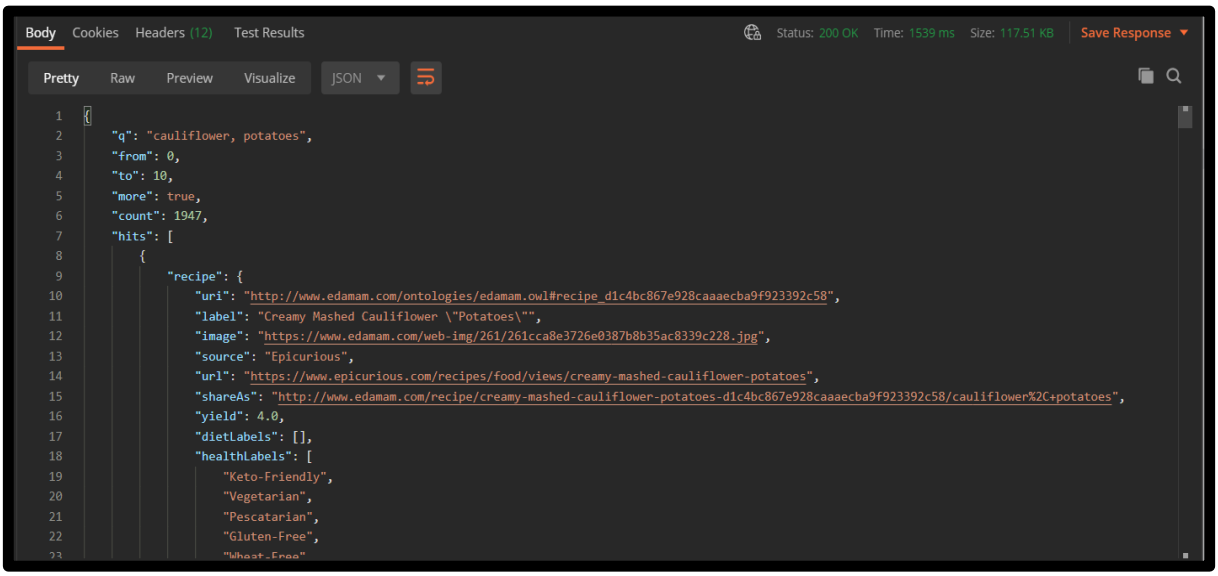


Fig 3.10: Postman2

5. Accessibility – To use the Postman tool, one would just need to log-in to their own accounts making it easy to access files anytime, anywhere as long as a Postman application is installed on the computer.
6. Use of Collections – Postman lets users create collections for their Postman API calls. Each collection can create subfolders and multiple requests. This helps in organizing your test suites.
7. Collaboration – Collections and environments can be imported or exported making it easy to share files. A direct link can also be used to share collections.
8. Creating Environments – Having multiple environments aids in less repetition of tests as one can use the same collection but for a different environment. This is where parameterization will take place which we will discuss in further lessons.
9. Creation of Tests – Test checkpoints such as verifying for successful HTTP response status can be added to each Postman API calls which help ensure test coverage.
10. Debugging – Postman console helps to check what data has been retrieved making it easy to debug tests.
11. Continuous Integration – With its ability to support continuous integration, development practices are maintained.
12. Data Parameterization is one of the most useful features of Postman. Instead of creating the same requests with different data, you can use variables with parameters. These data can be from a data file or an environment variable. Parameterization helps to avoid repetition of the same tests and iterations can be used for automation testing.

3.4 Software Used

1. VISUAL CODE STUDIO

2. GOOGLE CHROME

3. POSTMAN

3.5 Results

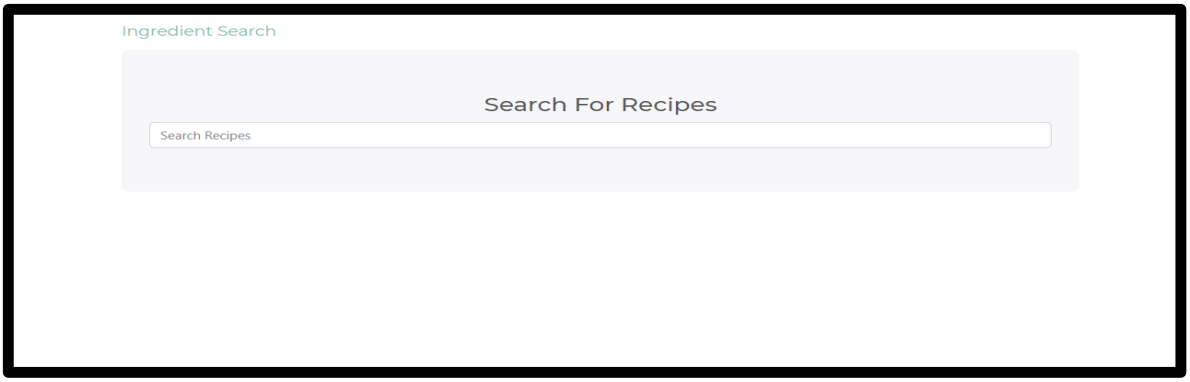


Fig 3.11: FrontEnd UI

1. Enter the list of Ingredients and press **ENTER**

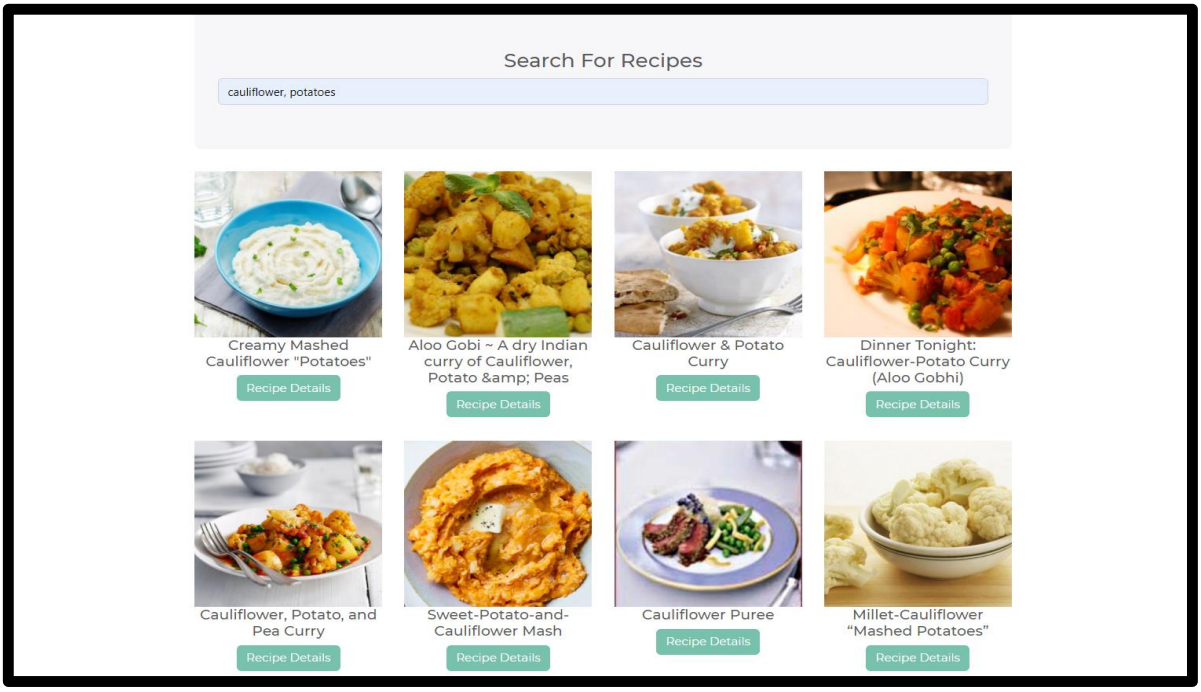


Fig 3.12: FrontEnd UI with Output

2. The Output with relevant Search results are displayed on the screen. It is displayed in a row format.
3. The buttons, which are anchor tags on the BackEnd are enabled with on-click event listeners that redirect you to the URL with the recipe instructions.

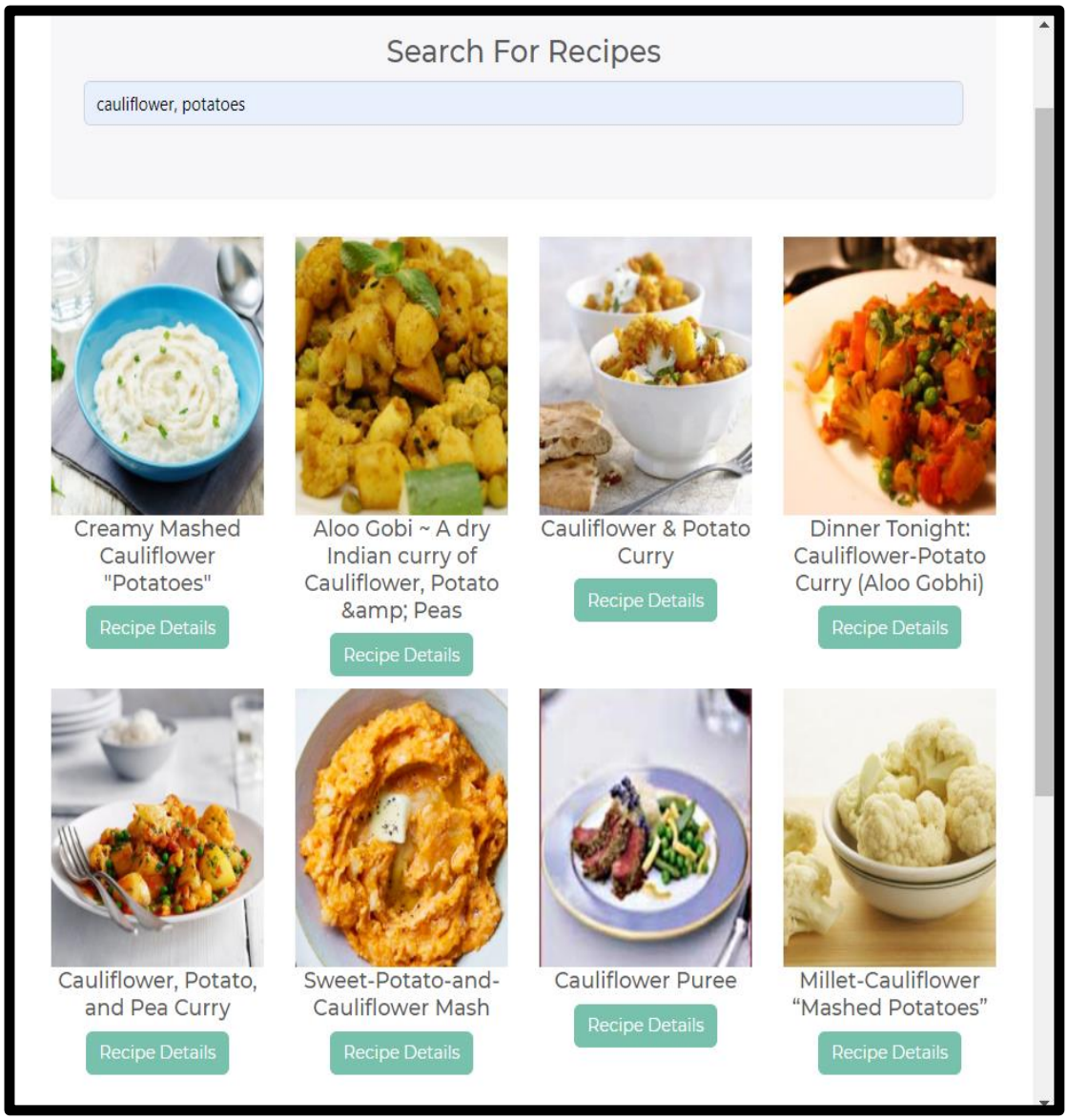


Fig 3.13: FrontEnd UI with Output on a Responsive Mobile View

4. When the width of the screen goes below 960px, the height and width of each individual image element is resized for a better interface.

Chapter 4

Recipe Image Classification

4.1 Flowchart

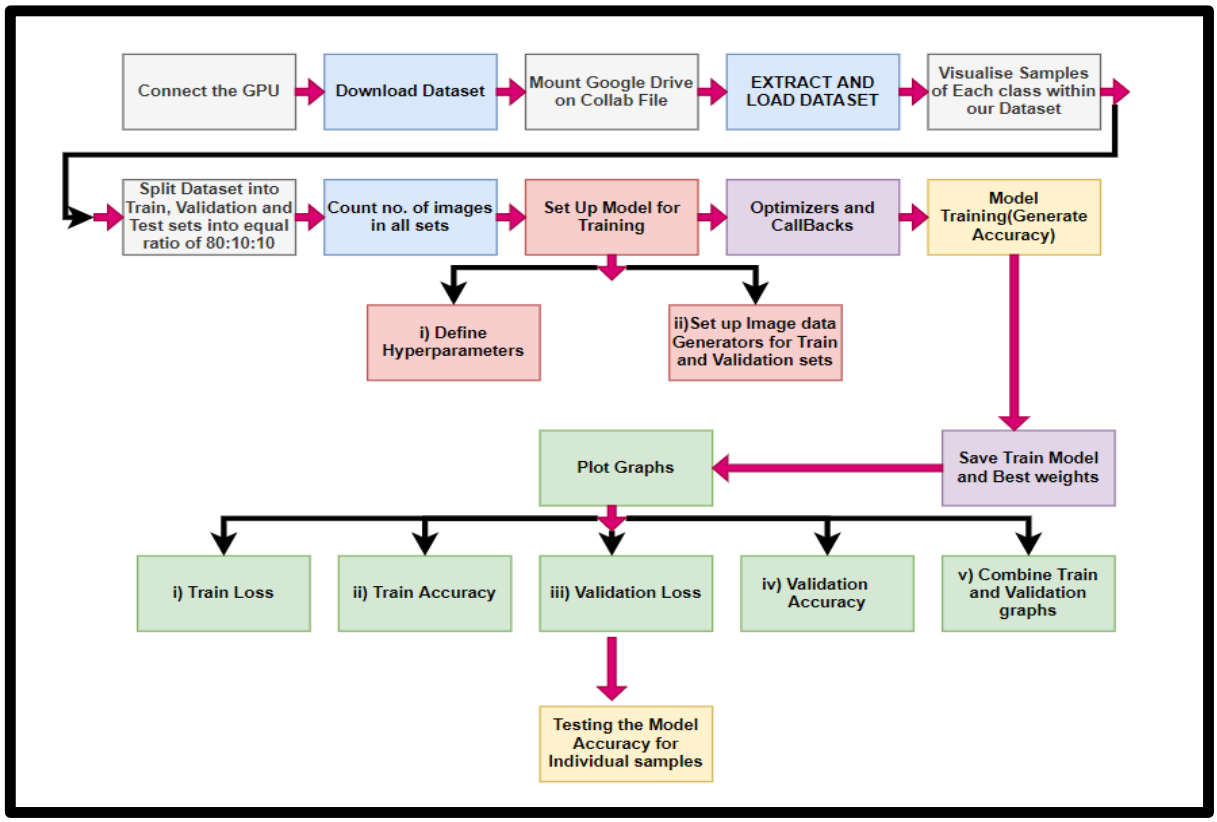


Fig 4.1: Image Classification Flowchart

4.2 Drawbacks and Errors faced while working with various Datasets

- ❑ Indian food dataset consisting of 20 classes gave 17. 48% accuracy and got terminated after the 12th epoch even after reducing the classes to 6.
- ❑ Food 101 dataset started with 17.89% accuracy and reached 63.46% in its last epoch.
- ❑ The 6 classes used were chocolate cake, cupcake, French fries, ice cream, pizza and samosa.
- ❑ 3 models were trained with different datasets and classes

Food101 having 101 classes was reduced to food 12 which gave 63.46% accuracy.

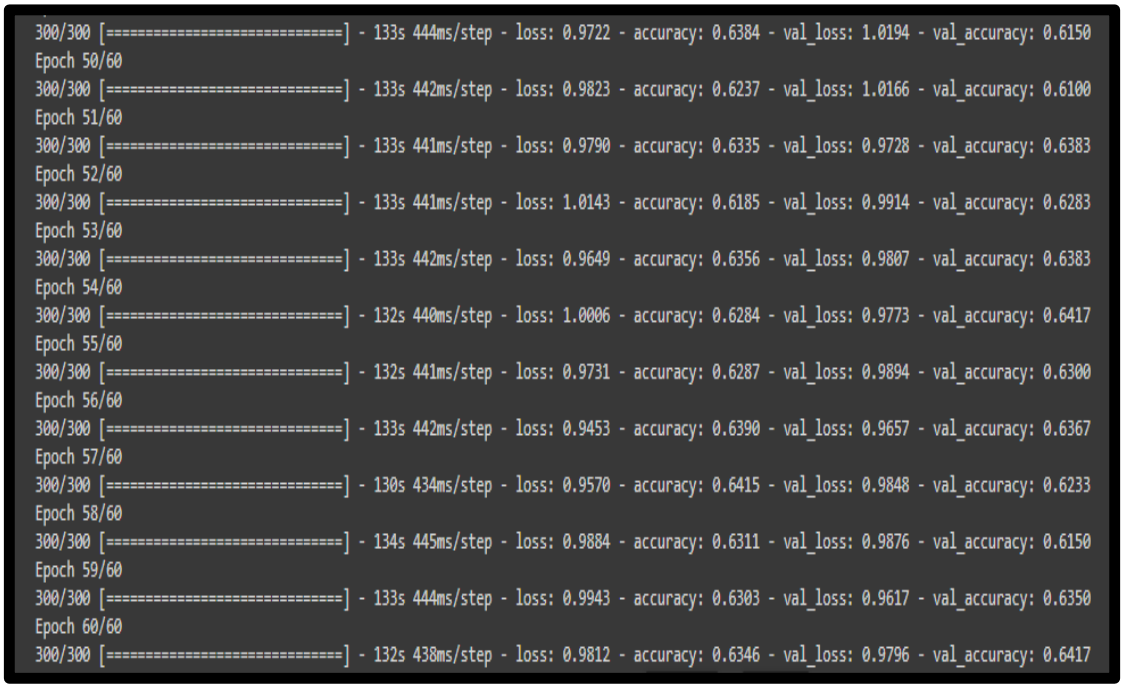


Fig 4.2: Food101 dataset result with 63.46% accuracy

Food 11 having 11 classes gave 8% accuracy

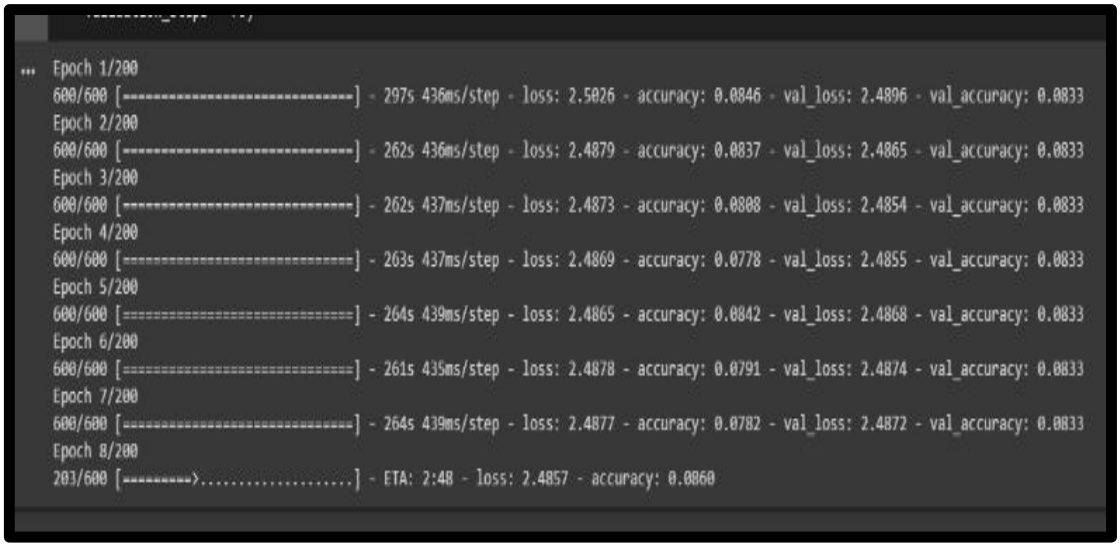


Fig 4.3: Food11 dataset result with 8% accuracy

Indian food dataset which gave 17.48% accuracy

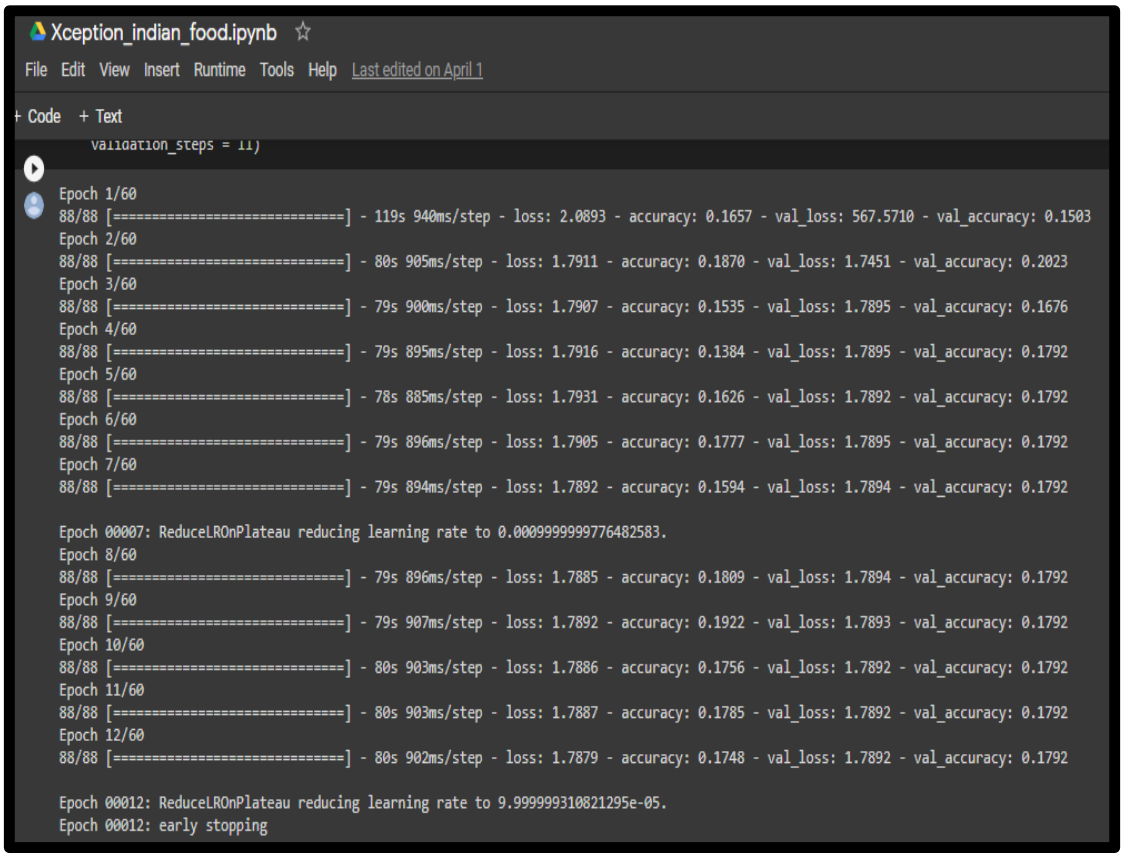


Fig 4.4: Indian food dataset result with 17.48% accuracy

4.3 Explanation and working

4.3.1 What is CNN

Image classification:

Image classification is the process of segmenting images into different categories based on their features. A feature could be the edges in an image, the pixel intensity, the change in pixel values, and many more. Consider three images belonging to the same individual, however they vary when compared across features like the color of the image, position of the face, the background color, color of the shirt, and many more. The biggest challenge when working with images is the uncertainty of these features. To the human eye, it looks all the same, however, when converted to data you may not find a specific pattern across these images easily.

An image consists of the smallest indivisible segments called pixels and every pixel has a strength often known as the pixel intensity. Whenever we study a digital image, it usually comes with three color channels, i.e. the Red-Green-Blue channels, popularly known as the "RGB" values. Why RGB? Because it has been seen that a combination of these three can produce all possible color pallets. Whenever working with a color image, the image is made up of multiple pixels with every pixel consisting of three different values for the RGB channels.

Now if there are multiple such images and try to label them as different individuals we can do it by analyzing the pixel values and looking for patterns in them. However, the challenge here is that since the background, the color scale, the clothing, etc. vary from image to image, it is hard to find patterns by analyzing the pixel values alone.

Hence it requires a more advanced technique that can detect these edges or find the underlying pattern of different features in the face using which these images can be labeled or classified. This is where a more advanced technique like CNN comes into the picture.

CNN:

CNN or the convolutional neural network (CNN) is a class of deep learning neural networks. In short, think of CNN as a machine learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other.

CNN works by extracting features from the images. Any CNN consists of the following:

1. The input layer which is a grayscale image
2. The Output layer which is a binary or multi-class labels
3. Hidden layers consisting of convolution layers, ReLU (rectified linear unit) layers, the pooling layers, and a fully connected Neural Network

It is very important to understand that ANN or Artificial Neural Networks, made up of multiple neurons, are not capable of extracting features from the image. This is where a combination of convolution and pooling layers comes into the picture. Similarly, the convolution and pooling layers can't perform classification hence we need a fully connected Neural Network.

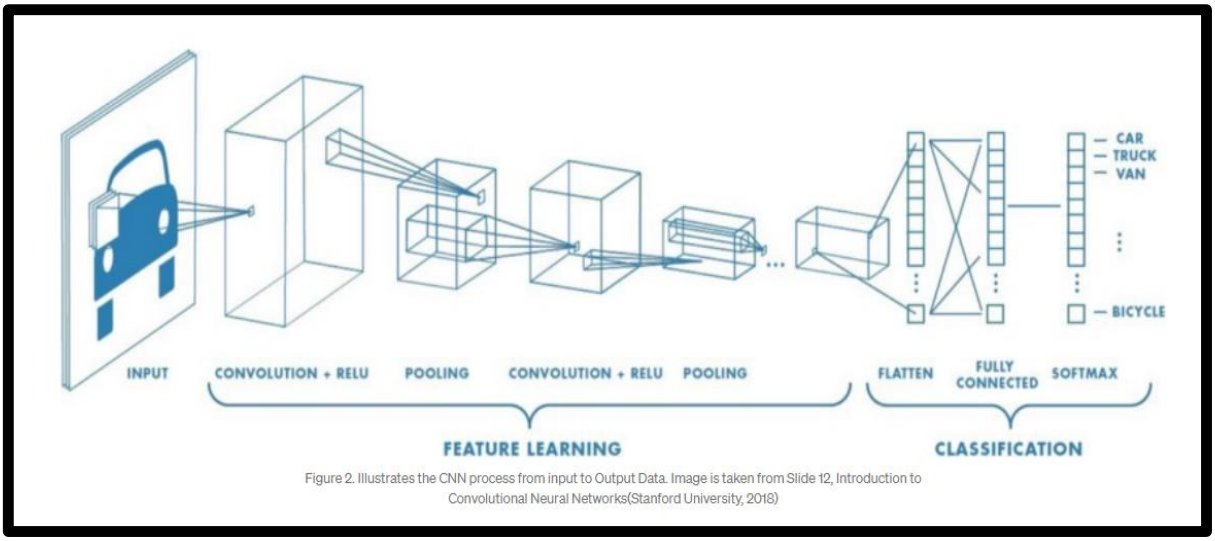


Fig 4.5: CNN process flow

A CNN has

1. Convolutional layers
2. ReLU layers
3. Pooling layers
4. A Fully connected layer

A classic CNN architecture would look something like this:

Input -> Convolution -> ReLU -> Convolution -> ReLU -> Pooling ->

ReLU -> Convolution -> ReLU -> Pooling -> Fully Connected

A CNN **convolves** (not convolutes…) learned features with input data and uses 2D convolutional layers. This means that this type of network is ideal for processing 2D images. Compared to other image classification algorithms, CNNs actually use very little preprocessing. This means that they can **learn** the filters that have to be hand-made in other algorithms. CNNs can be used in tons of applications from image and video recognition, image classification, and recommender systems to natural language processing and medical image analysis. CNNs have an input layer, an output layer, and hidden layers.

The hidden layers usually consist of convolutional layers, ReLU layers, pooling layers, and fully connected layers.

a. Convolutional layers apply a convolution operation to the input. This passes the information on to the next layer.
b. Pooling combines the outputs of clusters of neurons into a single neuron in the next layer.
c. Fully connected layers connect every neuron in one layer to every neuron in the next layer.

4.3.2 Convolution layer

An image to be classified is provided to the input layer and output is the predicted class label computed using extracted features from the image. An individual neuron in the next layer is connected to some neurons in the previous layer, this local correlation is termed as receptive field. The local features from the input image are extracted using a receptive field. The receptive field of a neuron associated to a particular region in the previous layer forms a weight vector, which remains equal at all points on the plane, where the plane refers to the neurons in the next layer. As the neurons in the plane share the same weights, thus the similar features occurring at different locations in the input data can be detected. The weight vector, also known as filter or kernel, slides over the input vector to generate the feature map. This method of sliding the filter horizontally as well as vertically is called convolution operation. This operation extracts N number of features from the input image in a single layer representing distinct features, leading to N filters and N feature maps.

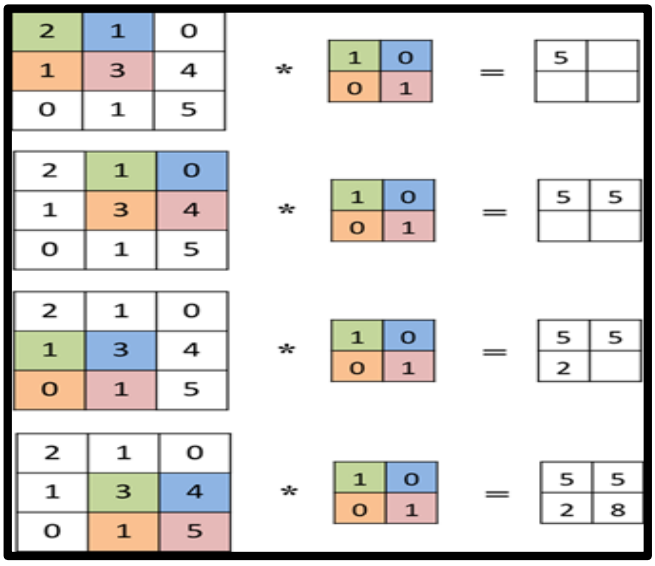


Fig 4.6: Convolution Operation by choosing a 2 x 2 Window

4.3.3 ReLu

After each conv layer, it is convention to apply a nonlinear layer (or activation layer) immediately afterward. The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers (just element wise multiplications and summations).In the past, nonlinear functions like tanh and sigmoid were used, but researchers found out that ReLU layers work far better because the network is able to train a lot faster (because of the computational efficiency) without making a significant difference to the accuracy. It also helps to alleviate the vanishing gradient problem, which is the issue where the lower layers of the network train very slowly because the gradient decreases exponentially through the layers (Explaining this might be out of the scope of this post, but see here and here for good descriptions). The ReLU layer applies the function f(x) = max(0, x) to all of the values in the input volume. In basic terms, this layer just changes all the negative activations to 0.This layer increases the nonlinear properties of the model and the overall network without affecting the receptive fields of the conv layer.

Let's consider that we have access to multiple images of different vehicles, each labeled into a truck, car, van, bicycle, etc. Now the idea is to take these pre-label/classified images and develop a machine learning algorithm that is capable of accepting a new vehicle image and classifying it into its correct category or label. Now before we start building a neural network we need to understand that most of the images are converted into a grayscale form before they are processed.

Depending on the weights associated with a filter, the features are detected from the image. Notice when an image is passed through a convolution layer, it tries to identify the features by analyzing the change in neighboring pixel intensities. If the image has similar pixel intensity throughout, then no edges are detected. It is only when the pixels change intensity the edges are visible. Hence ReLu is used.

Why ReLU?

ReLU or rectified linear unit is a process of applying an activation function to increase the non-linearity of the network without affecting the receptive fields of convolution layers. ReLU allows faster training of the data, whereas Leaky ReLU can be used to handle the problem of vanishing gradients. Some of the other activation functions include Leaky ReLU, Randomized Leaky ReLU, Parameterized ReLU Exponential Linear Units (ELU), Scaled Exponential Linear Units Tanh, hardtanh, softtanh, softsign, softmax, and softplus.

The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. The rectified linear activation function overcomes the vanishing gradient problem, allowing models to learn faster and perform better.

The purpose of applying the rectifier function is to increase the non-linearity in our images. The reason we want to do that is that images are naturally non-linear. When you look at any image, you'll find it contains a lot of non-linear features (e.g. the transition between pixels, the borders, the colors, etc.)

The general objective of the convolution operation is to extract high-level features from the image. We can always add more than one convolution layer when building the neural network, where the first Convolution Layer is responsible for capturing gradients whereas the second layer captures the edges. The addition of layers depends on the complexity of the image hence there are no magic numbers on how many layers to add. Note application of a 3 x 3 filter results in the original image results in a 3 x 3 convolved feature, hence to maintain the original dimension often the image is padded with values on both ends.

4.3.4 Pooling layer

The exact location of a feature becomes less significant once it has been detected. Hence, the convolution layer is followed by the pooling or sub-sampling layer. The major advantage of using pooling technique is that it remarkably reduces the number of trainable parameters and introduces translation invariance. To perform pooling operation, a window is selected and the input elements lying in that window are passed through a pooling function as shown in figure below.

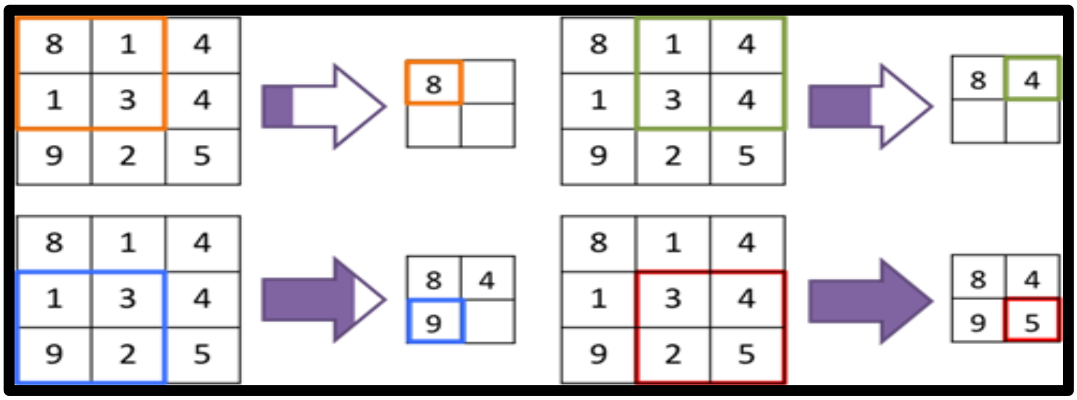


Fig 4.7: Pooling Operation by choosing a 2 x 2 Window

4.3.5 Fully Connected Layer

Fully connected layer is similar to the fully connected network in the conventional models. The output of the first phase (includes convolution and pooling repetitively) is fed into the fully connected layer, and the dot product of weight vector and input vector is computed in order to obtain final output. This component is used to either make a more abstract representation of the inputs by further processing of the features or classify the inputs based on the features extracted by preceding layers.

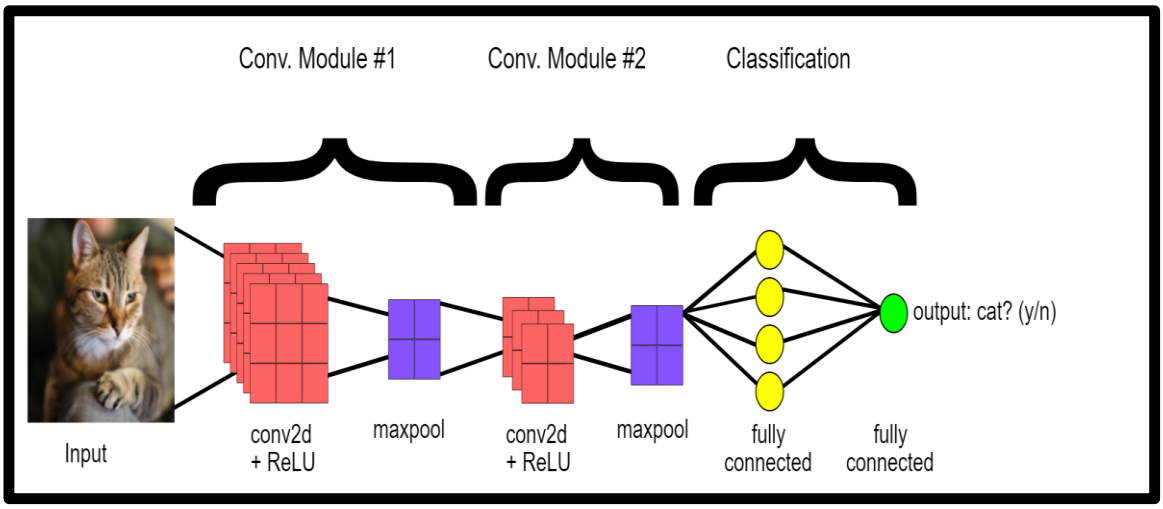


Fig 4.8: Convolution

Role of the Pooling Layer

The pooling layer applies a non-linear down-sampling on the convolved feature often referred to as the activation map. This is mainly to reduce the computational complexity required to process the huge volume of data linked to an image. Pooling is not compulsory and is often avoided. Usually, there are two types of pooling,

With the introduction of the Deep Convolutional Neural Network we are getting state of the art results on problems such as image classification and image recognition. So, over the years, researchers tend to make deeper Neural Networks(adding more layers) to solve such complex tasks and to also improve the classification/recognition accuracy. But, it has been seen that as we go on adding more layers to the neural network, it becomes difficult to train them and the accuracy starts saturating and then degrades also. Here ResNet comes to the rescue and helps solve this problem. In this article, we shall know more about ResNet and its architecture.

4.3.6 Resnet

ResNet, short for Residual Network is a specific type of neural network that was introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun in their paper “Deep Residual Learning for Image Recognition”.The ResNet models were extremely successful which you can guess from the following:

- Won 1st place in the ILSVRC 2015 classification competition with a top-5 error rate of 3.57% (An ensemble model)
- Won 1st place in ILSVRC and COCO 2015 competition in ImageNet Detection, ImageNet localization, Coco detection and Coco segmentation.
- Replacing VGG-16 layers in Faster R-CNN with ResNet-101. They observed relative improvements of 28%
- Efficiently trained networks with 100 layers and 1000 layers also.

Need for ResNet

Mostly in order to solve a complex problem, we stack some additional layers in the Deep Neural Networks which results in improved accuracy and performance. The intuition behind adding more layers is that these layers progressively learn more complex features. For example, in the case of recognizing images, the first layer may learn to detect edges, the second layer may learn to identify textures and similarly the third layer can learn to detect objects and so on.

Residual Block

This problem of training very deep networks has been alleviated with the introduction of ResNet or residual networks and these Resnets are made up from Residual Blocks.

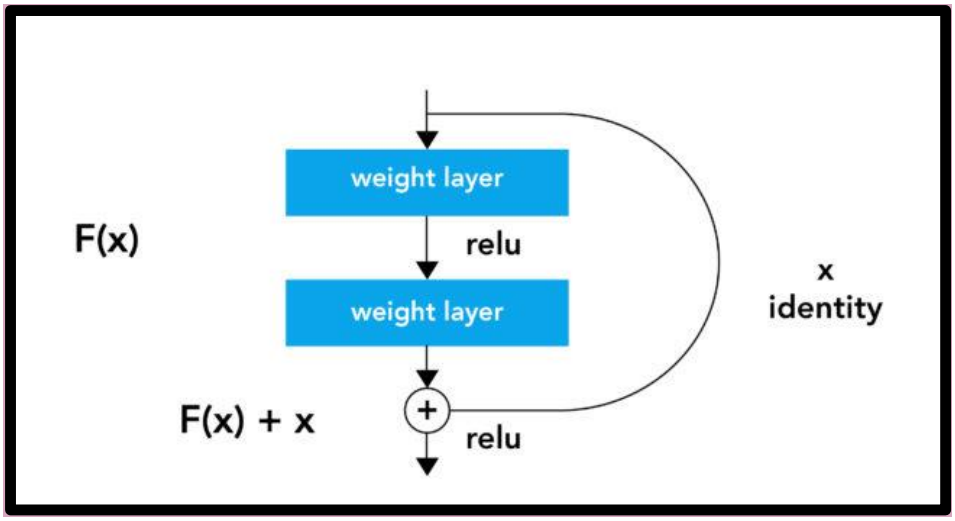


Fig 4.9: Residual learning- a building block

The very first thing we notice to be different is that there is a direct connection which skips some layers (may vary in different models) in between. This connection is called 'skip connection' and is the core of residual blocks. Due to this skip connection, the output of the layer is not the same now. Without using this skip connection, the input 'x' gets multiplied by the weights of the layer followed by adding a bias term.

How ResNet helps

The skip connections in ResNet solve the problem of vanishing gradients in deep neural networks by allowing this alternate shortcut path for the gradient to flow through. The other way that these connections help is by allowing the model to learn the identity functions which ensures that the higher layer will perform at least as good as the lower layer, and not worse. Let me explain this further.

Say we have a shallow network and a deep network that maps an input 'x' to output 'y' by using the function H(x). We want the deep network to perform at least as good as the shallow network and not degrade the performance as we saw in case of plain neural networks (without residual blocks). One way of achieving so is if the additional layers in a deep network learn the identity function and thus their output equals inputs which do not allow them to degrade the performance even with extra layers.

ResNet architecture

The ResNet network uses a 34-layer plain network architecture inspired by VGG-19 in which then the shortcut connection is added. These shortcut connections then convert the architecture into the residual network.

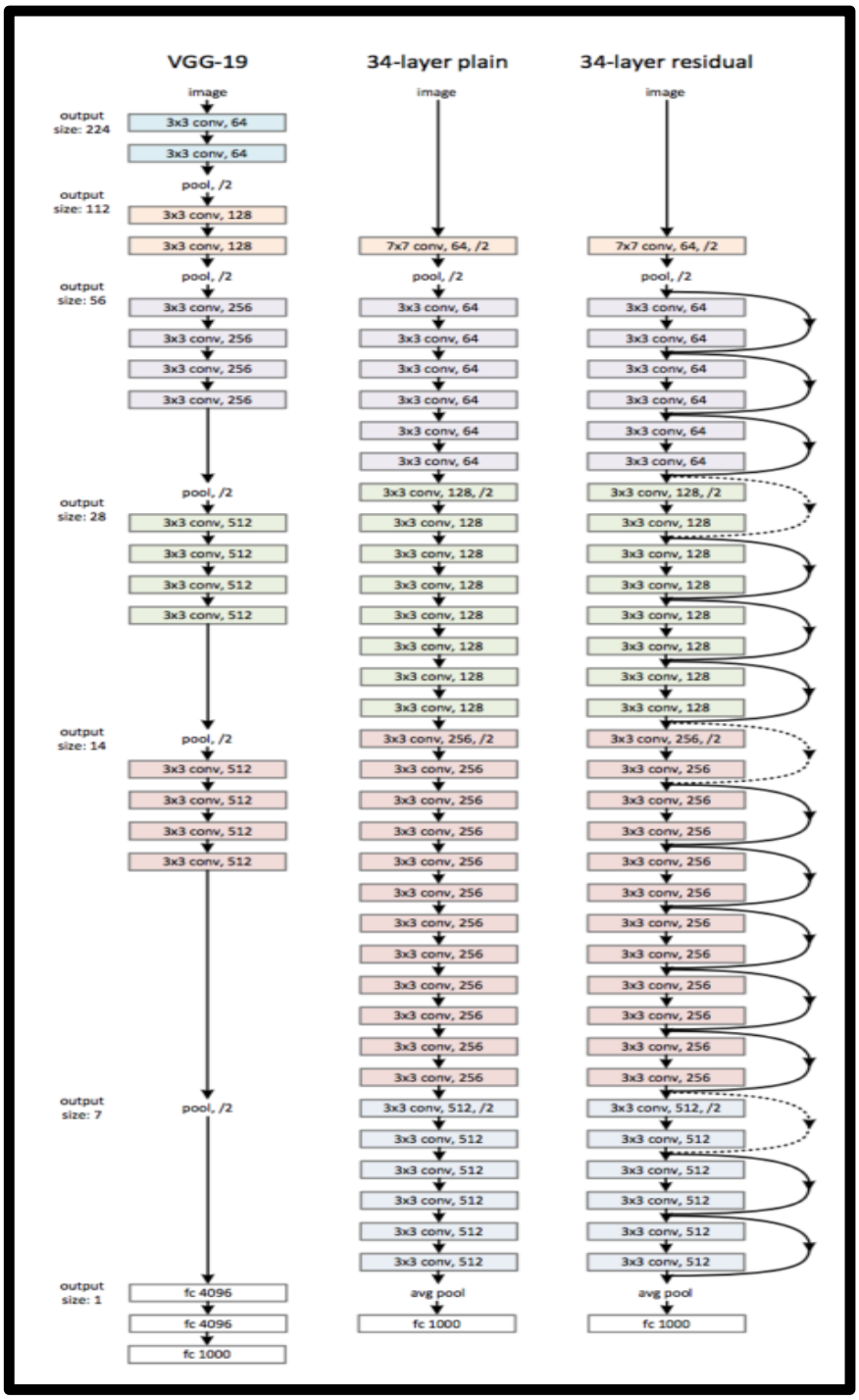


Fig 4.10: ResNet architecture

4.4 Machine Learning Recipe Classification- Xception Model

Step 1 - Upload the Food5 dataset on the google drive - Open google colab - Change the Runtime type to GPU.

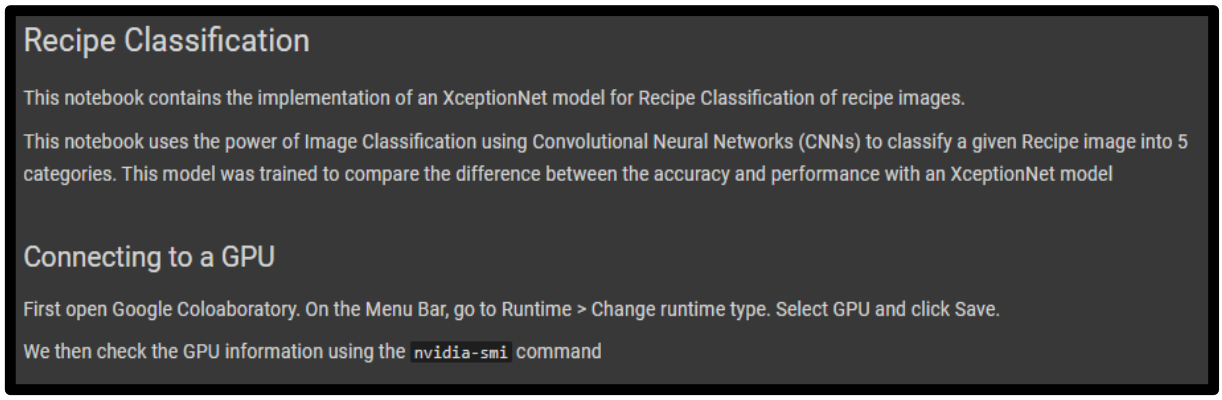


Fig 4.11: Xception- mounting to GPU1

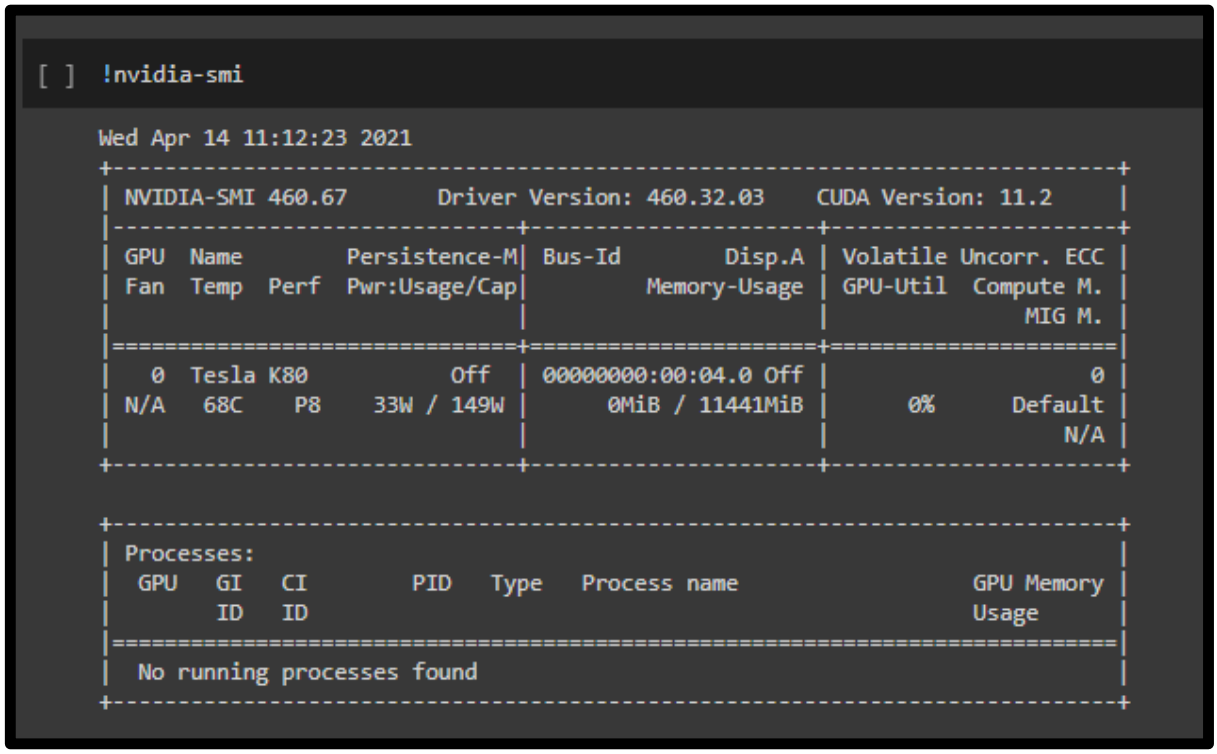


Fig 4.11: Xception- mounting to GPU2

Step 2 - Dataset used is Food101 from Kaggle.
The dataset has been reduced to 5 classes instead of 101 classes in order to obtain higher accuracy.
The detailed procedure is listed down below in the images.
Then we mount the google drive to google colab.

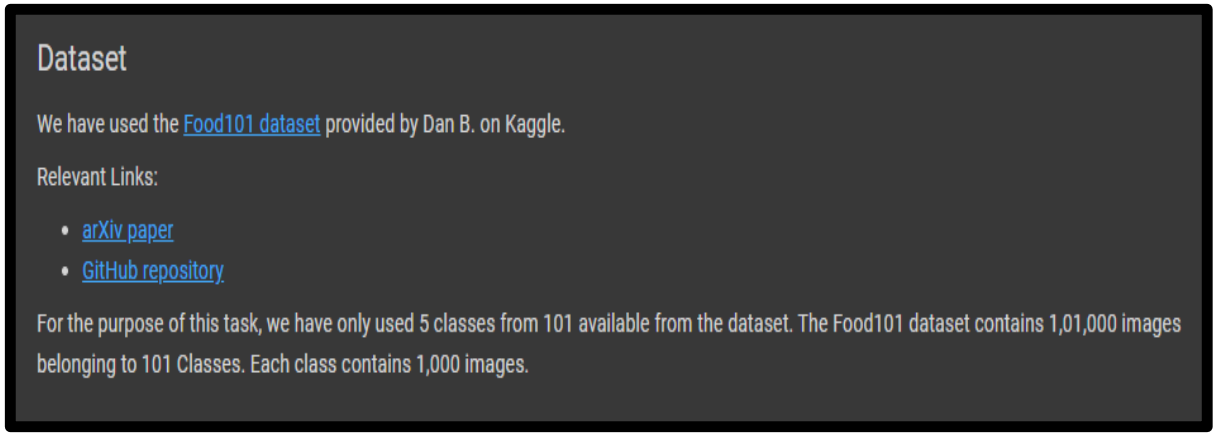


Fig 4.13: Xception- Dataset

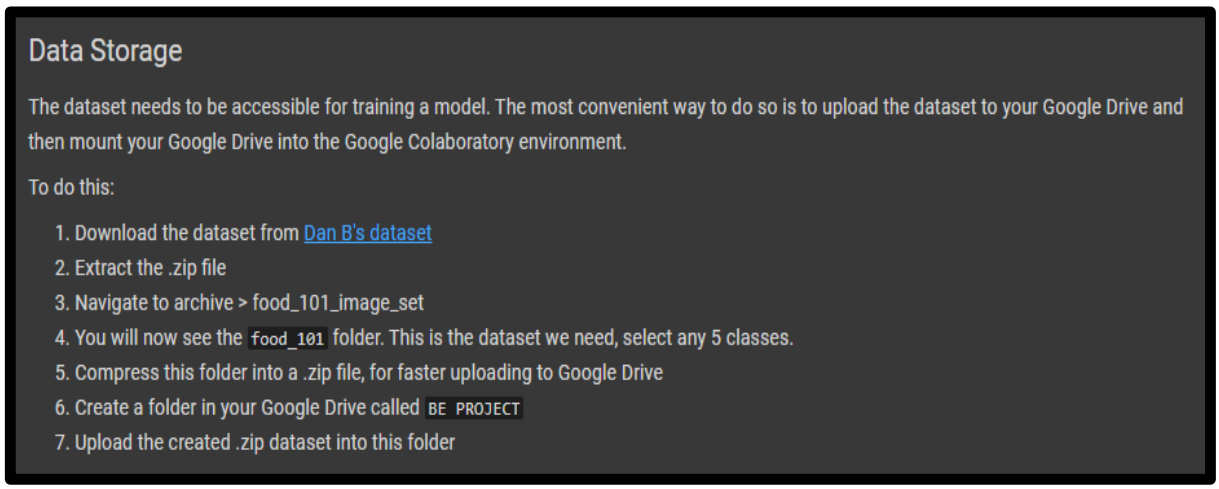


Fig 4.14: Xception- Data Storage

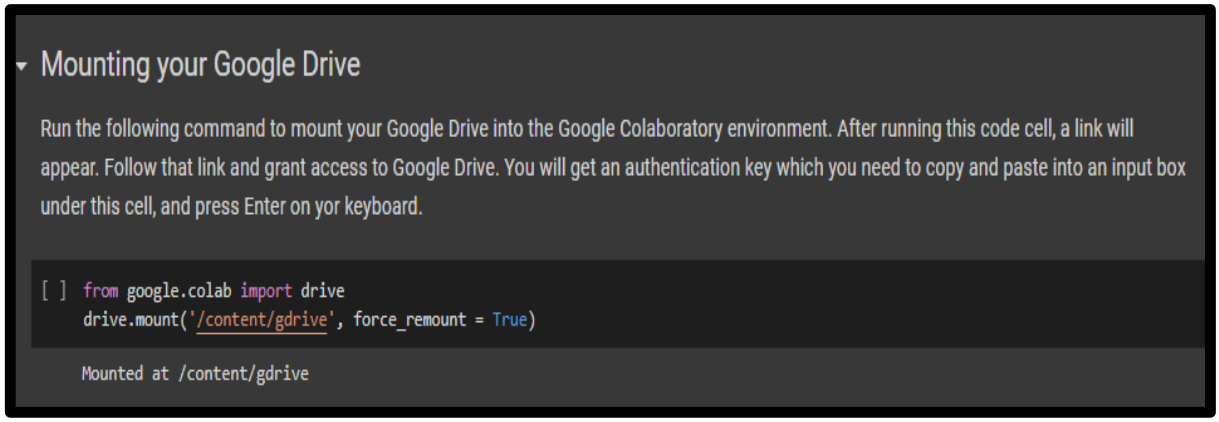


Fig 4.15: Xception- Mounting Google Drive

Step 3 - After mounting the google drive we import the necessary Libraries.

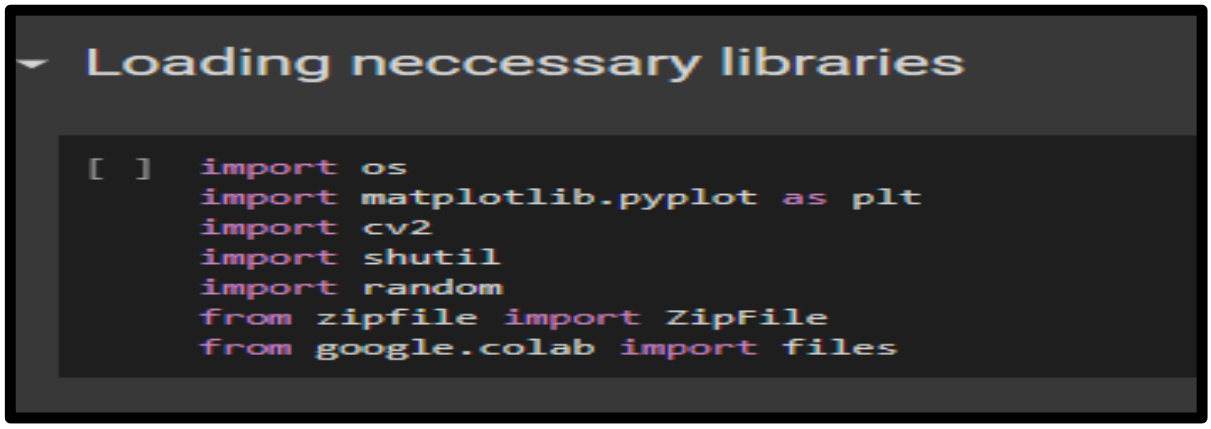


Fig 4.16: Xception- Loading necessary libraries

Step 4 - Next Step is to extract the dataset from the google drive where it was first uploaded after that we define the paths to the 5 class folders.

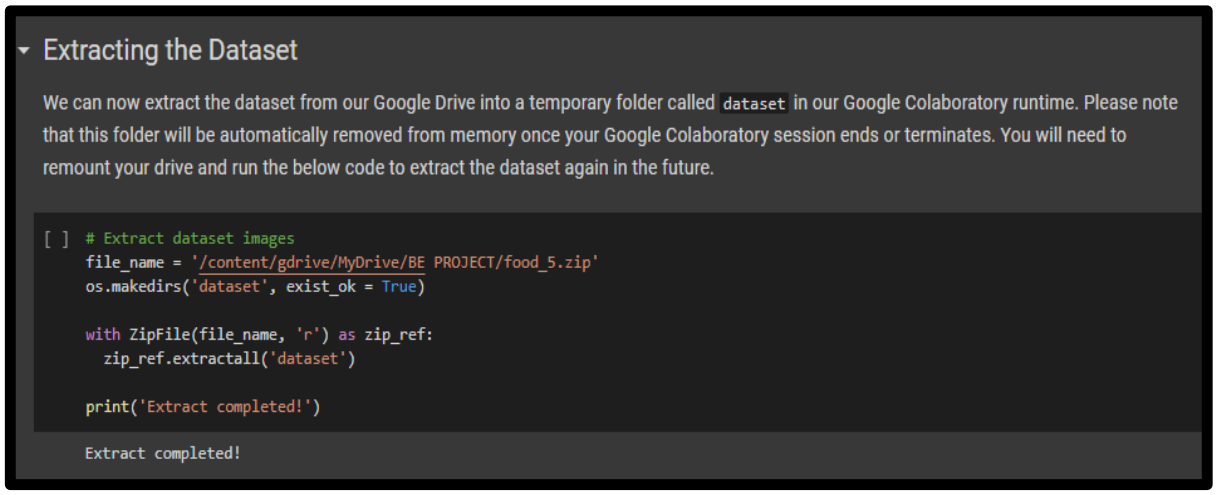


Fig 4.17: Xception- Extracting the dataset

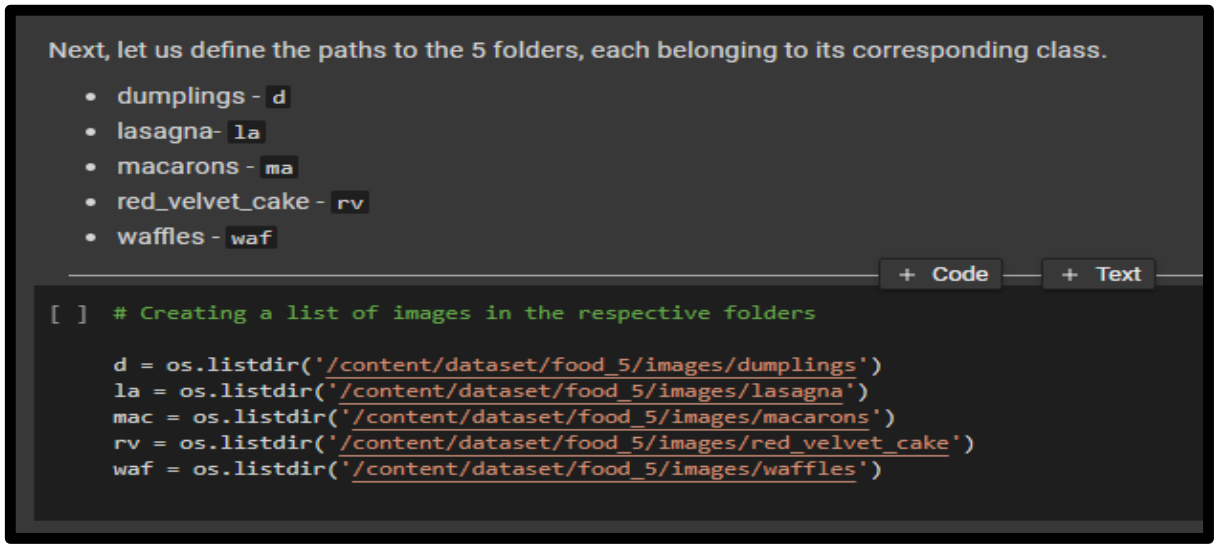


Fig 4.18: Xception- Defining paths

Step 5 - Visualizing the individual dumpling samples from dataset

Dataset Visualization

Let us view some samples of each class within our dataset.

Visualizing some Dumplings samples from the dataset

```
# Visualizing some Dumplings samples from the dataset
plt.figure(figsize = (15,15))

for i in range(12):
    plt.subplot(3, 4, i + 1)
    x = random.randint(0, len(d))
    path = '/content/dataset/food_5/images/dumplings/' + d[x]
    img = cv2.imread(path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.imshow(img)
    plt.title('dumplings')

plt.tight_layout()
plt.show()
```

Fig 4.19: Xception- Dumplings dataset visualization

Fig 4.20: Xception- Dumplings dataset visualization output

Step 6 - Visualizing Lasagna Samples from the dataset.

Visualizing some Lasagna samples from the dataset

```
# Visualizing some Lasagna samples from the dataset
plt.figure(figsize = (15,15))

for i in range(12):
    plt.subplot(3, 4, i + 1)
    x = random.randint(0, len(la))
    path = '/content/dataset/food_5/images/lasagna/' + la[x]
    img = cv2.imread(path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.imshow(img)
    plt.title('lasagna')

plt.tight_layout()
plt.show()
```

Fig 4.21: Xception- Lasagna dataset visualization

Fig 4.22: Xception- Lasagna dataset visualization output

Step 7 - Visualizing Macaron samples from the dataset.

Visualizing some Macarons samples from the dataset

```
# Visualizing some Macarons samples from the dataset
plt.figure(figsize = (15,15))

for i in range(12):
    plt.subplot(3, 4, i + 1)
    x = random.randint(0, len(mac))
    path = '/content/dataset/food_5/images/macarons/' + mac[x]
    img = cv2.imread(path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.imshow(img)
    plt.title('macarons')

plt.tight_layout()
plt.show()
```

Fig 4.23: Xception- Macron dataset visualization

Fig 4.24: Xception- Macron dataset visualization output

Step 8 - Visualizing some red_velvet_cake samples from the dataset.

Visualizing some red_velvet_cake samples from the dataset

```
# Visualizing some red_velvet_cake samples from the dataset
plt.figure(figsize = (15,15))

for i in range(12):
    plt.subplot(3, 4, i + 1)
    x = random.randint(0, len(rv))
    path = '/content/dataset/food_5/images/red_velvet_cake/' + rv[x]
    img = cv2.imread(path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.imshow(img)
    plt.title('red_velvet_cake')

plt.tight_layout()
plt.show()
```

Fig 4.25: Xception- Red velvet cake dataset visualization

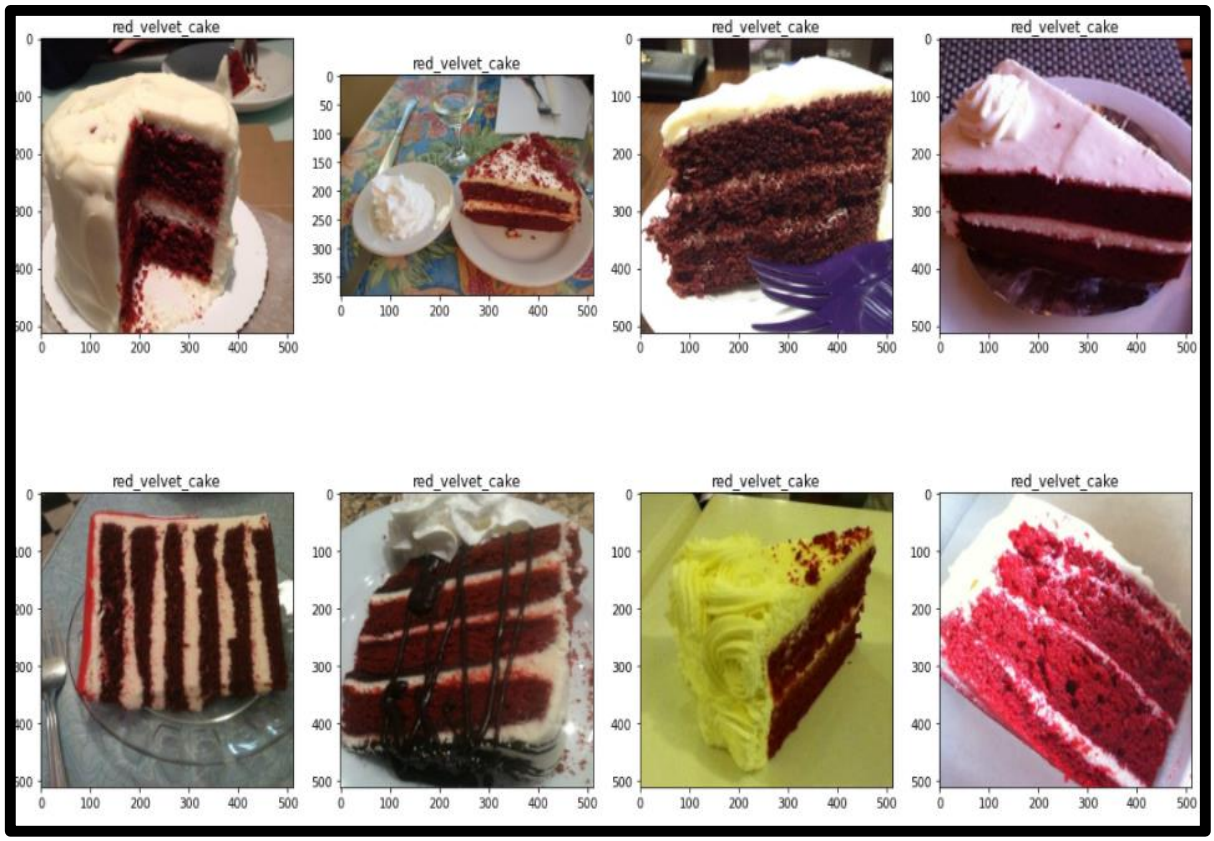


Fig 4.26: Xception- Red velvet cake dataset visualization output

Step 9 - Visualizing Waffle food samples from the dataset.

Visualizing some Waffles food samples from the dataset

```
# Visualizing some Waffles samples from the dataset
plt.figure(figsize = (15,15))

for i in range(12):
    plt.subplot(3, 4, i + 1)
    x = random.randint(0, len(waf))
    path = '/content/dataset/food_5/images/waffles/' + waf[x]
    img = cv2.imread(path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.imshow(img)
    plt.title('waffles')

plt.tight_layout()
plt.show()
```

Fig 4.27: Xception- Waffles dataset visualization

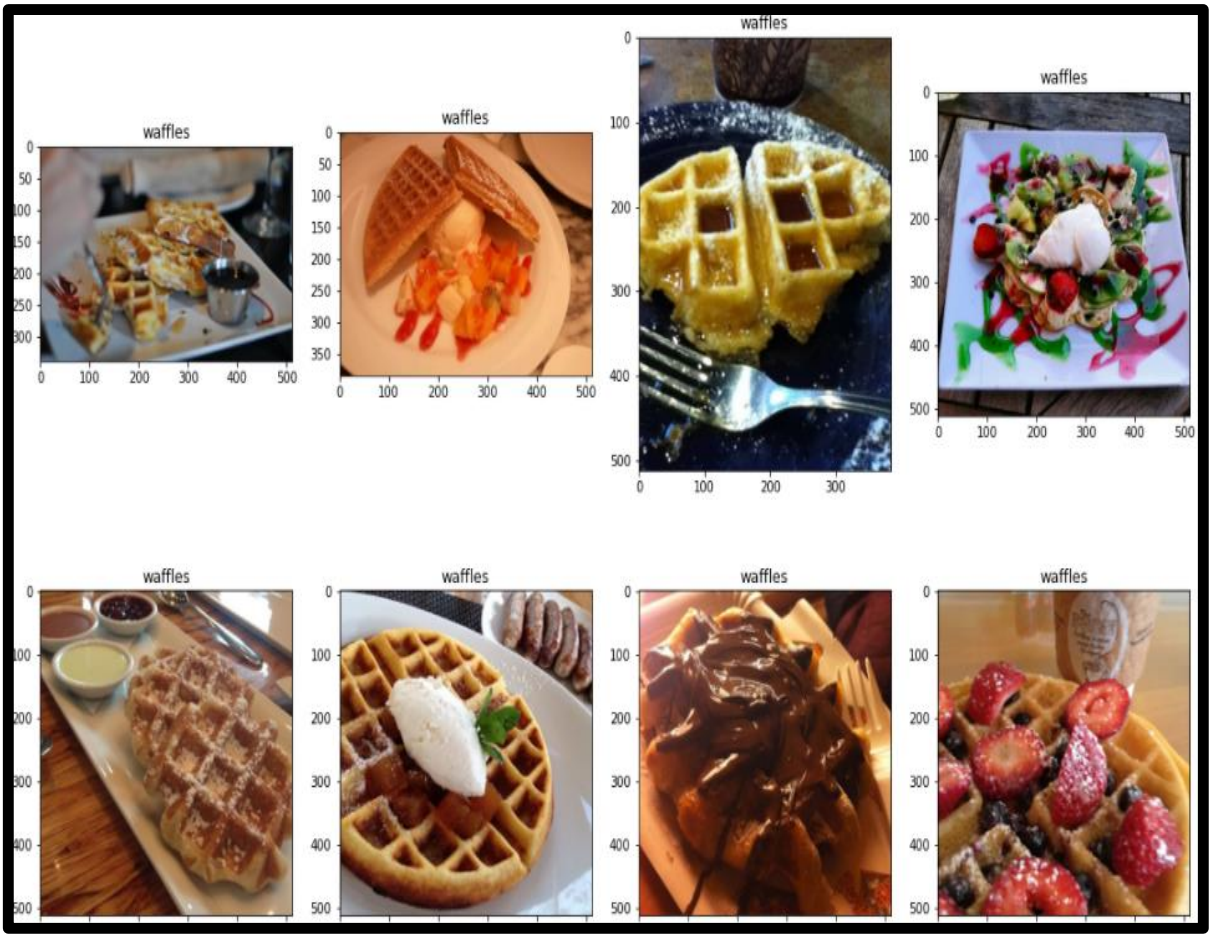


Fig 4.28: Xception- Waffles dataset visualization output

Step 10 - Splitting the dataset into train, validation and test sets.
Split-folders library is used to split the dataset into train, validation and test sets.

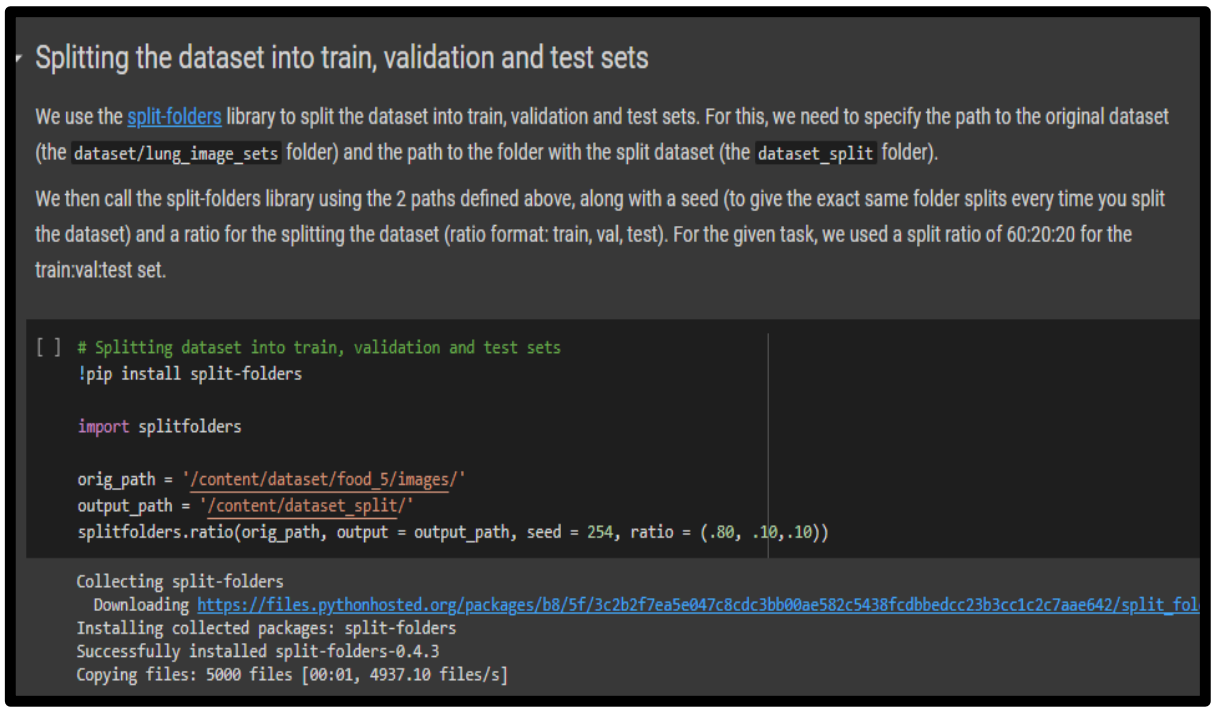


Fig 4.29: Xception- Splitting the dataset

Then count and check the number of images in each of the sets we created.

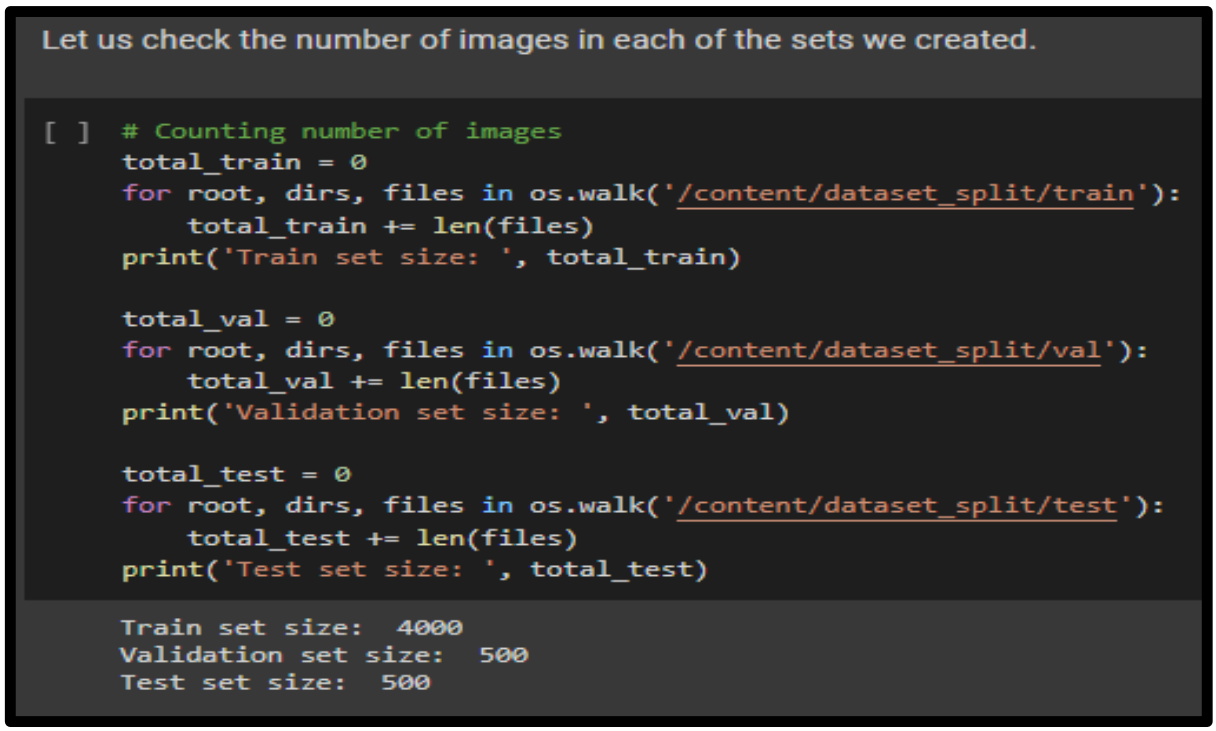


Fig 4.30: Xception- Count the number of images

Xception model:

Step 11 - Setting up the Model training where we use an XceptionNet model from Keras Applications through keras.applications.xception.

Start with importing the required libraries.

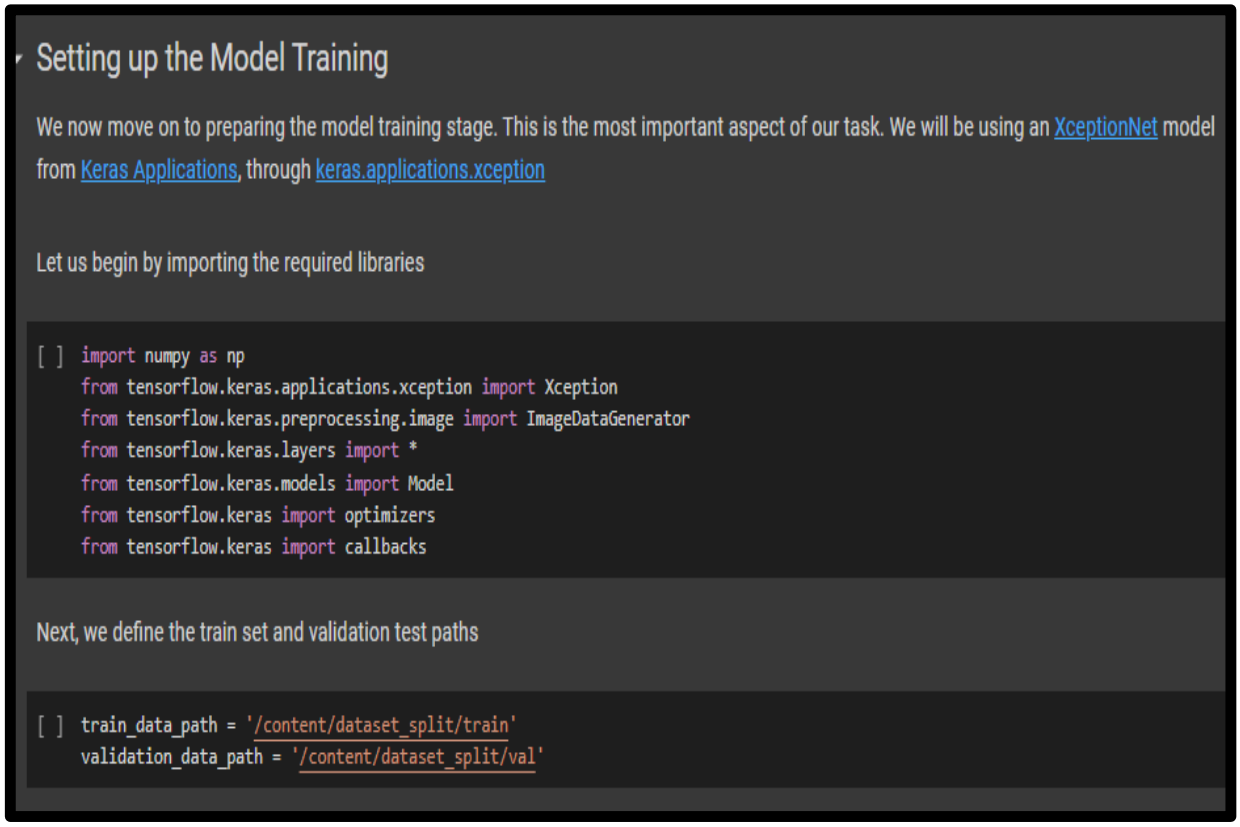


Fig 4.31: Xception- Setting up model training

We then define the train set and validation test paths and then the hyper parameters like image width, image height, channels, batch size, validation steps, number of classes, learning rate and epochs.

The next step is to define the training and validation generators using the defined hyper parameters.
ImageDataGenerator: Generate batches of tensor image data with real-time data augmentation. Read more here.

train_generator: Takes the path to a directory & generates batches of augmented data for training, based on the train image data generator.

validation_generator: Takes the path to a directory & generates batches of augmented data for validation, based on the validation image data generator.

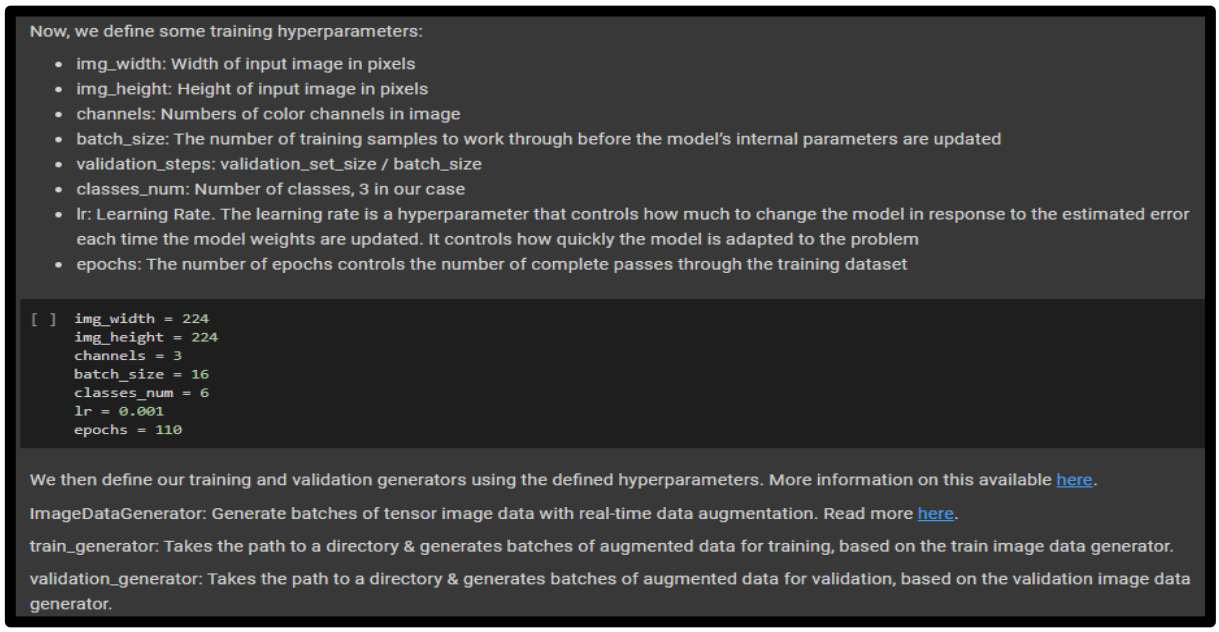


Fig 4.32: Xception- Defining hyperparameters1

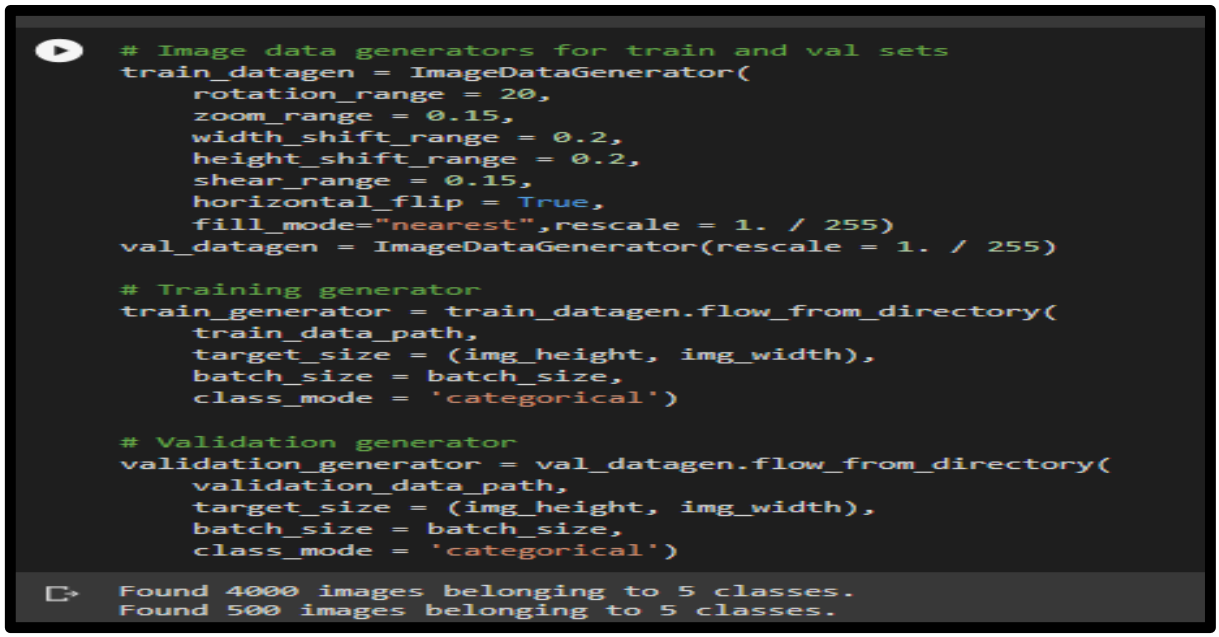


Fig 4.33: Xception- Defining hyperparameters2

Step 12 - Define the Xception net model. The input uses the same image height and width which were defined earlier.

Ingredient weights are used as starting weights to train the model for our custom number of classes.

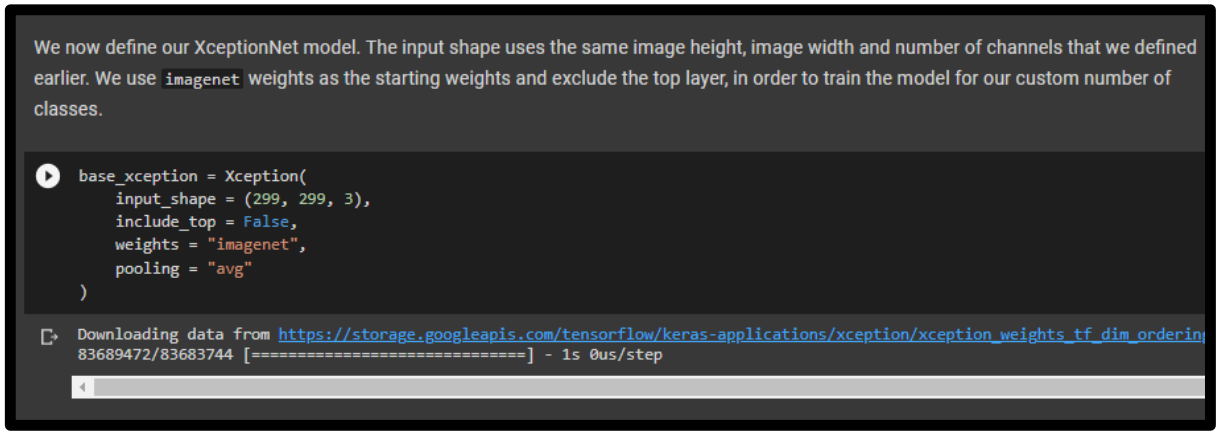


Fig 4.34: Xception- Defining XceptionNet model

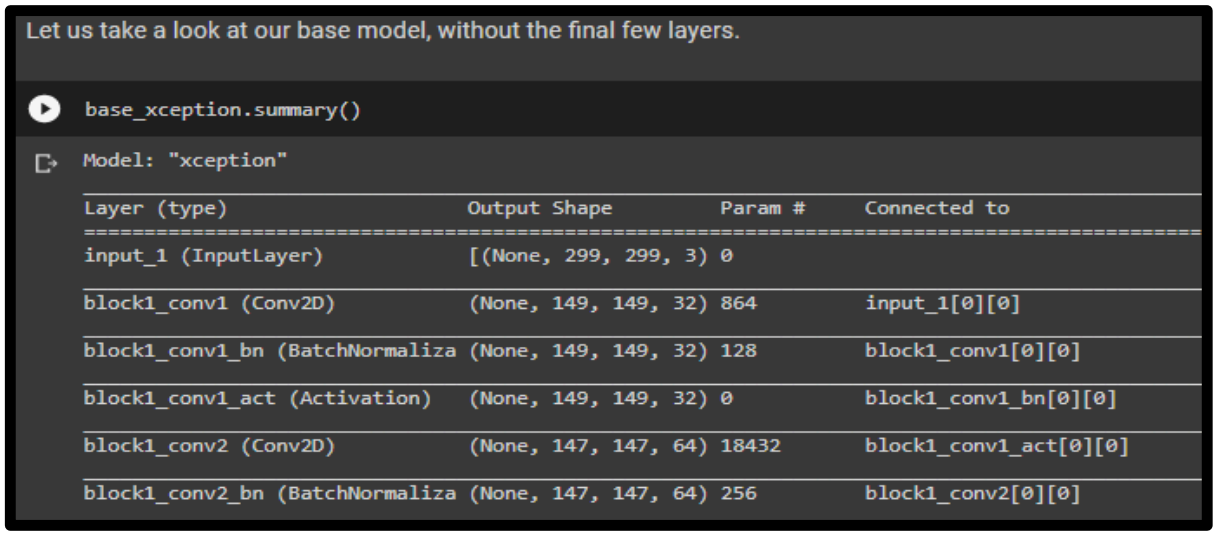


Fig 4.35: Xception- Base model summary

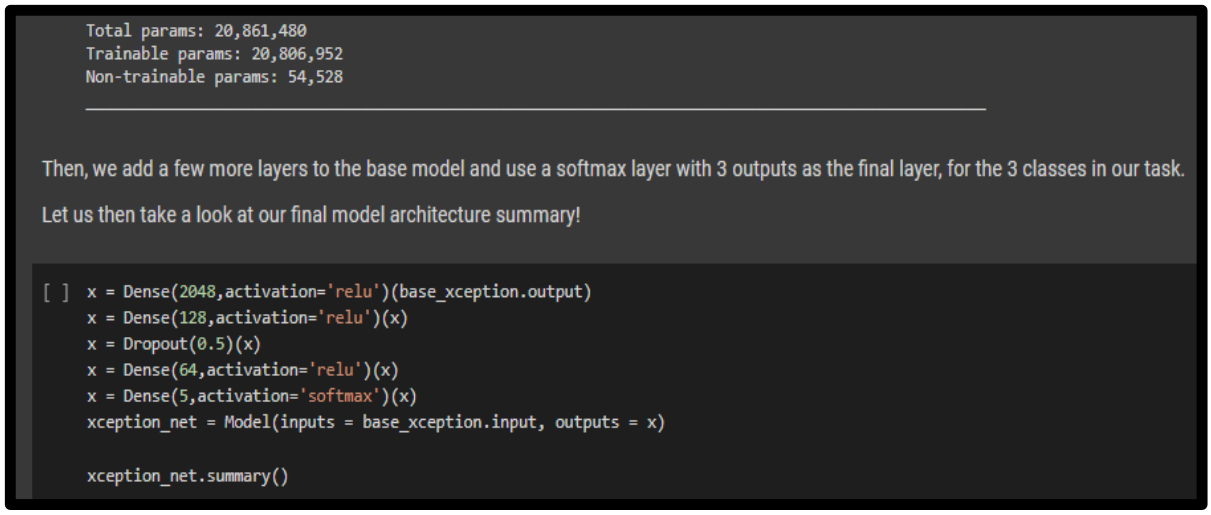


Fig 4.36: Xception- XceptionNet summary

Step 13 - Optimizer and Callbacks.

An optimizer is a method or algorithm to update the various parameters that can reduce the loss in much less effort. One of the most commonly used and best optimizers is Adam.

```
[ ] adam = optimizers.Adam(lr = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e-5)

    xception_net.compile(loss = 'categorical_crossentropy',
                optimizer = adam,
                metrics = ['accuracy'])
```

Fig 4.37: Xception- Optimizer and callback

```
[ ]  target_dir = './snapshots2'

     if not os.path.exists(target_dir):
       os.mkdir(target_dir)
```

Fig 4.38: Xception- Folder to store model weights

Next, we create a directory to save our model weights. We create a snapshots2 folder in the Google Colaboratory environment for this purpose.

However, please note that like the extracted dataset, these weights will be deleted once your session terminates.

Thus, if you have ample space in your Google Drive, you can temporarily save your model weights there. After the training is complete, you can select the best weights and discard the others.

For our task, we created a snapshots2 folder within the BE PROJECT folder we created in our Google Drive earlier, which we used to store our dataset in.

In case you do not have enough space in Google Drive, you can proceed with storing your model weights in the Google Collaboratory environment and then copy the best weights to your Google Drive, after the training process.

```
# Directory to store training logs
log_dir = './tf-log/'

# Path tp store trained weights. Uncomment the needed one and comment out the other
### Google Drive
file_path = '/content/gdrive/MyDrive/BE PROJECT/snapshots2/best_weight{epoch:03d}.h5'

'''
### Google Colaboratory environment
file_path = '/content/snapshots2/best_weight{epoch:03d}.h5'
'''

# Checkpoints
checkpoints = callbacks.ModelCheckpoint(file_path, monitor = "val_loss", verbose = 0, save_best_only = True, mode = 'auto')

# Reduce learning rate on plateau
lr_op = callbacks.ReduceLROnPlateau(monitor = "val_loss", factor = 0.1, patience = 5,
  verbose = 1, mode = "auto", cooldown = 0,
  min_lr = 1e-30)

# Early stopping
early_stop = callbacks.EarlyStopping(
    monitor = "val_loss",
    min_delta = 0,
    patience = 10,
    verbose = 1,
    mode = "auto",
    baseline = None,
    restore_best_weights = False,
)

# Add all the created callbacks
cbks = [checkpoints, lr_op, early_stop]
```

Fig 4.39: Xception- Directory to store training logs

Model Training

Finally, we can begin with our model trainig. We call the fit() method for this purpose. We pass the train and validation generators, number of epochs, callbacks, steps_per_epochs (train_set_size / batch_size), validation_steps(validation_set_size / batch_size).

```
# Model training
history = xception_net.fit(
    train_generator,
    steps_per_epoch = 250,
    epochs = 110,
    validation_data = validation_generator,
    callbacks = cbks,
    validation_steps = 32)
```

Fig 4.40: Xception- Model training

```
Epoch 00025: ReduceLROnPlateau reducing learning rate to 1.0000000474974514e-05.
Epoch 26/110
250/250 [==============================] - 73s 293ms/step - loss: 0.0350 - accuracy: 0.9890 - val_loss: 0.1824 - val_accuracy: 0.9680
Epoch 27/110
250/250 [==============================] - 73s 293ms/step - loss: 0.0400 - accuracy: 0.9889 - val_loss: 0.1822 - val_accuracy: 0.9700
Epoch 28/110
250/250 [==============================] - 73s 293ms/step - loss: 0.0317 - accuracy: 0.9886 - val_loss: 0.1915 - val_accuracy: 0.9700
Epoch 29/110
250/250 [==============================] - 73s 293ms/step - loss: 0.0369 - accuracy: 0.9878 - val_loss: 0.1851 - val_accuracy: 0.9720
Epoch 30/110
250/250 [==============================] - 74s 294ms/step - loss: 0.0439 - accuracy: 0.9888 - val_loss: 0.1833 - val_accuracy: 0.9700

Epoch 00030: ReduceLROnPlateau reducing learning rate to 1.0000000656873453e-06.
Epoch 00030: early stopping
```

+ Code + Text

Fig 4.41: Xception- Results

Training Statistics

We have now trained our model and have our weights saved in the selected `snapshots` folder. Kindly store these weights safely, as you will need them in the future to perform inferences.

Let us see the training statistics. This will help us select the best model. The validation loss is a good metric to select the best weights. Since we have checkpointed the weights based on the best validation loss, the last saved weights will most likely be the best one. In this case, it was the weights of the 45th epoch.

```
[ ] training_stats = history.history
    print(training_stats)
```

```
{'loss': [0.9760598540306091, 0.7416197061538696, 0.5999800562858582, 0.4745841324329376, 0.4488547146320343, 0.3705286383628845,
```

Best Model Weights Statistics

- Train Loss: 0.0360
- Train Accuracy: 0.9868
- Val Loss: 0.0486
- Val Accuracy: 0.9857

Fig 4.42: Xception- Training Statistics

Plotting the Graphs

For Visualizing the graphs for train loss, train accuracy, validation loss, validation accuracy.

The loss graphs should ideally show a gradual decrease, while the accuracy graphs should ideally show a gradual increase**.**

Train Loss

We can see the train loss starting at a very high value initially. As the no. of epochs increase, the model learns and the train loss decreases after some initial instability.

We can see the initial train loss around/between 0.03-0.04 towards the final few epochs**.**

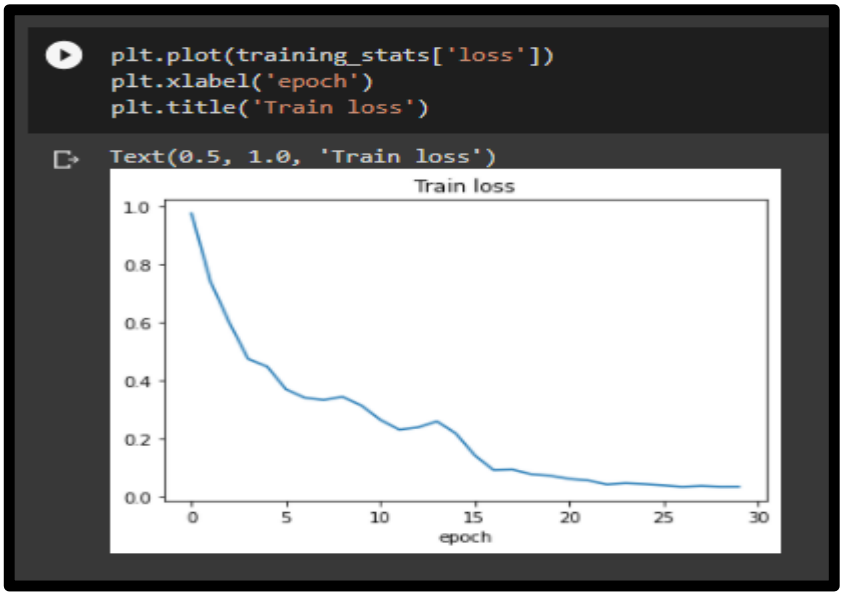


Fig 4.43: Xception- Train loss graph

Train Accuracy

We can see the train accuracy starting at a very low value initially. As the number of epochs increases, the model learns and the train accuracy increases after some initial instability. We can see that the train accuracy is around 98-99% towards the final few epochs.

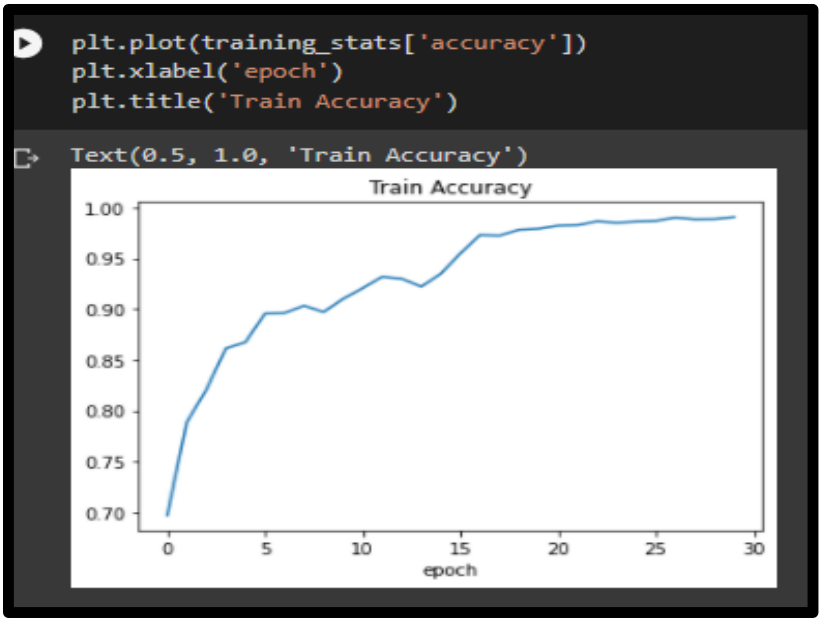


Fig 4.44: Xception- Train accuracy graph

Validation Loss

We can see the validation loss being very unstable and exploding to a very high value initially. As the number of epochs increases, the model learns and the validation loss decreases after some initial instability. We can see that the train loss is around 0.04-0.05 towards the final few epochs.

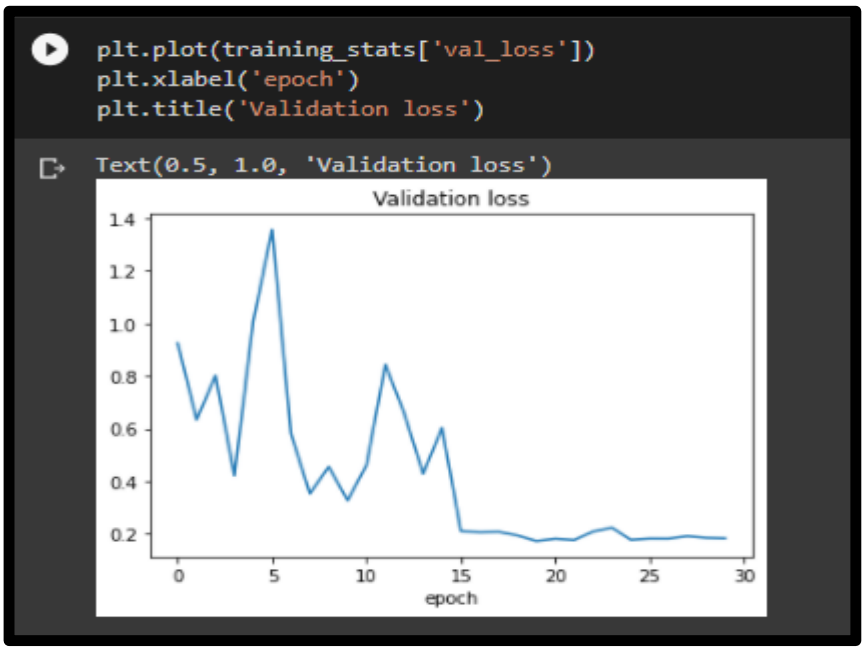


Fig 4.45: Xception- Validation Loss graph

Validation Accuracy

We can see the validation accuracy starting at a very low value initially and acting very unstable. As the number of epochs increases, the model learns and the validation accuracy increases after some initial instability. We can see that the validation accuracy is around 98-99% towards the final few epochs

```
plt.plot(training_stats['val_accuracy'])
plt.xlabel('epoch')
plt.title('Validation Accuracy')
```

```
Text(0.5, 1.0, 'Validation Accuracy')
```

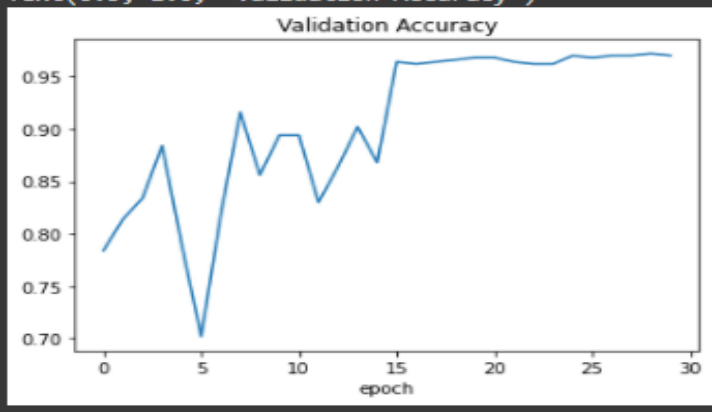


Fig 4.46: Xception- Validation accuracy graph

Combining train and validation graphs

Let us visualize the train and validation graphs together for comparison. The blue line represents the train metric and the orange line represents the validation metric.

```
fig, ax = plt.subplots(1, 2, figsize=(30, 10))
ax = ax.ravel()

for i, metric in enumerate(["accuracy", "loss"]):
    ax[i].plot(xception_net.history.history[metric][:])
    ax[i].plot(xception_net.history.history["val_" + metric][:])
    ax[i].set_title("Model {}".format(metric))
    ax[i].set_xlabel("epochs")
    ax[i].set_ylabel(metric)
    ax[i].legend(["train", "val"])
```

Fig 4.47: Xception- Train and validation comparison

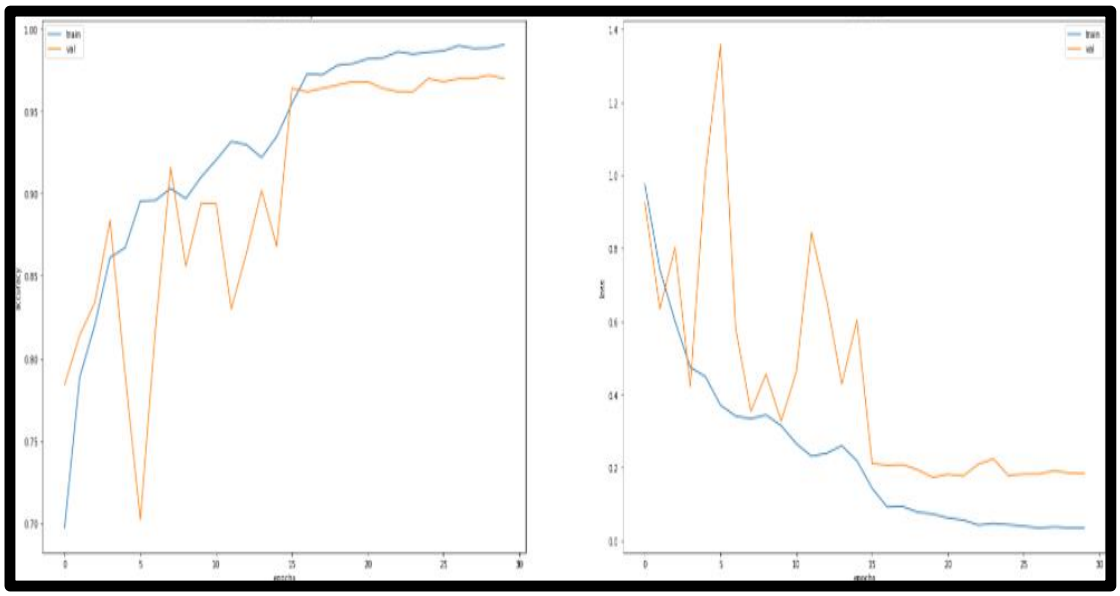


Fig 4.48: Xception- Train and validation comparison graph

Testing the model

It is now time to test our trained model using our test set! We will also plot the incorrectly classified images to get an idea of where and why the model might be misclassifying.

As always, we first import the needed packages.

Next, we load the model from our best weights. In this case, the weights of the 45th epoch were the best. They are stored in the `snapshots 2` directory within the `BE Project` folder on our mounted Google Drive.

```
# Load model
model = load_model('/content/gdrive/MyDrive/BE PROJECT/snapshots2/best_weight020.h5')
```

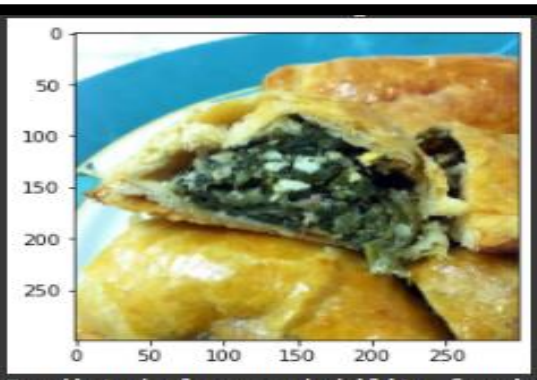
Fig 4.49: Xception- load model

Testing for individual Samples: Here we test the individual samples for their accuracy and if the images are correctly classified or not.

Testing for Dumplings samples

```
# Testing for Dumpling samples
class_dict = {0: 'dumplings', 1: 'lasagna', 2: 'macarons', 3: 'red_velvet_cake', 4: 'waffles'}
dumplings_count = 0

food_d = os.listdir('dataset_split/test/dumplings')

for img in food_d:
    img_path = 'dataset_split/test/dumplings/' + img
    img = load_img(img_path, target_size = (299,299))
    img_tensor = img_to_array(img)
    img_tensor = np.expand_dims(img_tensor, axis=0)
    img_tensor /= 255.

    ### predict() takes image tensor of (batchsize,heighht,width,channels)
    ### Returns an array for all images in batch with probability for each class
    pred = model.predict(img_tensor)

    classes = np.argmax(pred)

    if classes == 0:
      dumplings_count += 1
    else:
      plt.imshow(img_tensor[0])
      plt.show()
      print('Predicted class probability for image: ', pred[0][classes])

      ### Class number with highest probablity (Classification result of CNN)
      print('Predicted Class :', class_dict[classes])
```

Fig 4.50: Xception- Test for dumplings




```
Predicted class probability for image:  0.5297894
Predicted Class : waffles
```

Let us calculate the test accuracy for dumplings samples.

```
print('Test size for dumplings: ', len(food_d))
print('Correctly classifed as dumplings:', dumplings_count)
print('Incorrectly classified images:', len(food_d) - dumplings_count)
print('Accuracy:', (dumplings_count / len(food_d)) * 100)
```

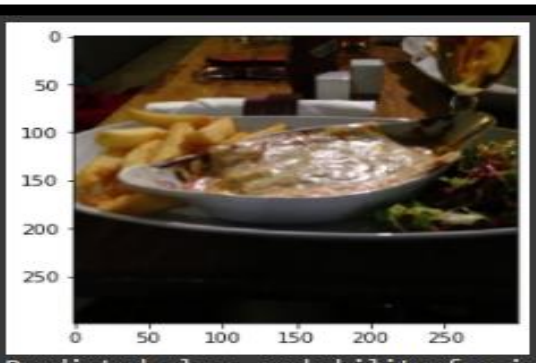
```
Test size for dumplings:  100
Correctly classifed as dumplings: 94
Incorrectly classified images: 6
Accuracy: 94.0
```

Fig 4.51: Xception- Test for dumplings output

Testing for Lasagna

```
# Testing for Lasagna samples
class_dict = {0: 'dumplings', 1: 'lasagna', 2: 'macarons', 3: 'red_velvet_cake', 4: 'waffles'}
lasagna_count = 0

food_la = os.listdir('dataset_split/test/lasagna')

for img in food_la:
    img_path = 'dataset_split/test/lasagna/' + img
    img = load_img(img_path, target_size = (299,299))
    img_tensor = img_to_array(img)
    img_tensor = np.expand_dims(img_tensor, axis=0)
    img_tensor /= 255.

    ### predict() takes image tensor of (batchsize,heighht,width,channels)
    ### Returns an array for all images in batch with probability for each class
    pred = model.predict(img_tensor)

    classes = np.argmax(pred)

    if classes == 1:
        lasagna_count += 1
    else:
      plt.imshow(img_tensor[0])
      plt.show()
      print('Predicted class probability for image: ', pred[0][classes])

      ### Class number with highest probablity (Classification result of CNN)
      print('Predicted Class :', class_dict[classes])
```

Fig 4.52: Xception- Test for lasagna




```
Predicted class probability for image:  0.69060963
Predicted Class : waffles
```

Let us calculate the test accuracy for Lasagna samples.

```
print('Test size for lasagna: ', len(food_la))
print('Correctly classifed as lasagna:', lasagna_count)
print('Incorrectly classified images:', len(food_la) - lasagna_count)
print('Accuracy:', (lasagna_count / len(food_la)) * 100)
```

```
Test size for lasagna:  100
Correctly classifed as lasagna: 99
Incorrectly classified images: 1
Accuracy: 99.0
```

Fig 4.53: Xception- Test for lasagna output

Testing for Macarons samples

```
# Testing for Macaron samples
class_dict = {0: 'dumplings', 1: 'lasagna', 2: 'macarons', 3: 'red_velvet_cake', 4: 'waffles'}
macarons_count = 0

food_mac = os.listdir('dataset_split/test/macarons')

for img in food_mac:
    img_path = 'dataset_split/test/macarons/' + img
    img = load_img(img_path, target_size = (299,299))
    img_tensor = img_to_array(img)
    img_tensor = np.expand_dims(img_tensor, axis=0)
    img_tensor /= 255.

    ### predict() takes image tensor of (batchsize,heighht,width,channels)
    ### Returns an array for all images in batch with probability for each class
    pred = model.predict(img_tensor)

    classes = np.argmax(pred)

    if classes == 2:
        macarons_count += 1
    else:
      plt.imshow(img_tensor[0])
      plt.show()
      print('Predicted class probability for image: ', pred[0][classes])

      ### Class number with highest probablity (Classification result of CNN)
      print('Predicted Class :', class_dict[classes])
```

Fig 4.54: Xception- Test for macron

```
Predicted class probability for image:  0.8927565
Predicted Class : red_velvet_cake
```


```
Predicted class probability for image:  0.40367597
Predicted Class : dumplings
```

Let us calculate the test accuracy for Macarons samples.

```
[ ]  print('Test size for macarons: ', len(food_mac))
     print('Correctly classifed as macarons:', macarons_count)
     print('Incorrectly classified images:', len(food_mac) - macarons_count)
     print('Accuracy:', (macarons_count / len(food_d)) * 100)

     Test size for macarons:  100
     Correctly classifed as macarons: 95
     Incorrectly classified images: 5
     Accuracy: 95.0
```

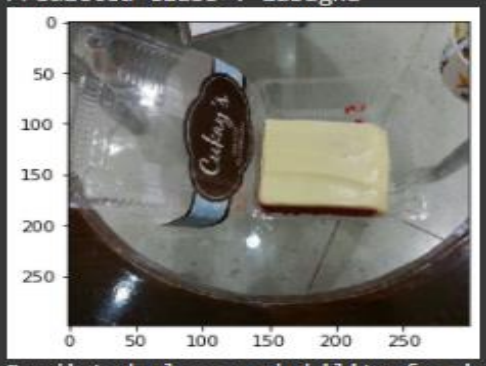
Fig 4.55: Xception- Test for macron output

Testing for red_velvet_cake samples

```
# Testing for red_velvet_cake samples
class_dict = {0: 'dumplings', 1: 'lasagna', 2: 'macarons', 3: 'red_velvet_cake', 4: 'waffles'}
red_velvet_cake_count = 0

food_rv = os.listdir('dataset_split/test/red_velvet_cake')

for img in food_rv:
    img_path = 'dataset_split/test/red_velvet_cake/' + img
    img = load_img(img_path, target_size = (299,299))
    img_tensor = img_to_array(img)
    img_tensor = np.expand_dims(img_tensor, axis=0)
    img_tensor /= 255.

    ### predict() takes image tensor of (batchsize,heighht,width,channels)
    ### Returns an array for all images in batch with probability for each class
    pred = model.predict(img_tensor)

    classes = np.argmax(pred)

    if classes == 3:
        red_velvet_cake_count += 1
    else:
      plt.imshow(img_tensor[0])
      plt.show()
      print('Predicted class probability for image: ', pred[0][classes])

      ### Class number with highest probablity (Classification result of CNN)
      print('Predicted Class :', class_dict[classes])
```

Fig 4.56: Xception- Test for red velvet cake

```
Predicted class probability for image:  0.3337038
Predicted Class : lasagna
```




```
Predicted class probability for image:  0.5437461
Predicted Class : macarons
```

Calculating the accuracy for red_velvet_cake

```
[ ] print('Test size for red_velvet_cake: ', len(food_rv))
    print('Correctly classifed as red_velvet_cake:', red_velvet_cake_count)
    print('Incorrectly classified images:', len(food_rv) - red_velvet_cake_count)
    print('Accuracy:', (red_velvet_cake_count / len(food_rv)) * 100)

    Test size for red_velvet_cake:  100
    Correctly classifed as red_velvet_cake: 96
    Incorrectly classified images: 4
    Accuracy: 96.0
```

Fig 4.57: Xception- Test for red velvet cake output

Testing for Waffles Samples

```
class_dict = {0: 'dumplings', 1: 'lasagna', 2: 'macarons', 3: 'red_velvet_cake', 4: 'waffles'}
waffles_count = 0

food_waf = os.listdir('dataset_split/test/waffles')

for img in food_waf:
    img_path = 'dataset_split/test/waffles/' + img
    img = load_img(img_path, target_size = (299,299))
    img_tensor = img_to_array(img)
    img_tensor = np.expand_dims(img_tensor, axis=0)
    img_tensor /= 255.

    ### predict() takes image tensor of (batchsize,heighht,width,channels)
    ### Returns an array for all images in batch with probability for each class
    pred = model.predict(img_tensor)

    classes = np.argmax(pred)

    if classes == 4:
      waffles_count += 1
    else:
      plt.imshow(img_tensor[0])
      plt.show()
      print('Predicted class probability for image: ', pred[0][classes])

      ### Class number with highest probablity (Classification result of CNN)
      print('Predicted Class :', class_dict[classes])
```

Fig 4.58: Xception- Test for waffle

```
Predicted Class : red_velvet_cake
```


```
Predicted class probability for image:  0.29154128
Predicted Class : red_velvet_cake
```

Calculating the accuracy for Waffles

```
print('Test size for waffles: ', len(food_waf))
print('Correctly classifed as waffles:', waffles_count)
print('Incorrectly classified images:', len(food_waf) - waffles_count)
print('Accuracy:', (waffles_count / len(food_waf)) * 100)
```

```
Test size for waffles:  100
Correctly classifed as waffles: 91
Incorrectly classified images: 9
Accuracy: 91.0
```

Fig 4.59: Xception- Test for waffle output

4.5 Machine Learning Recipe Classification- MobileNet Model

Step 1 - Upload the Food5 dataset on the google drive - Open google colab - Change the Runtime type to GPU.

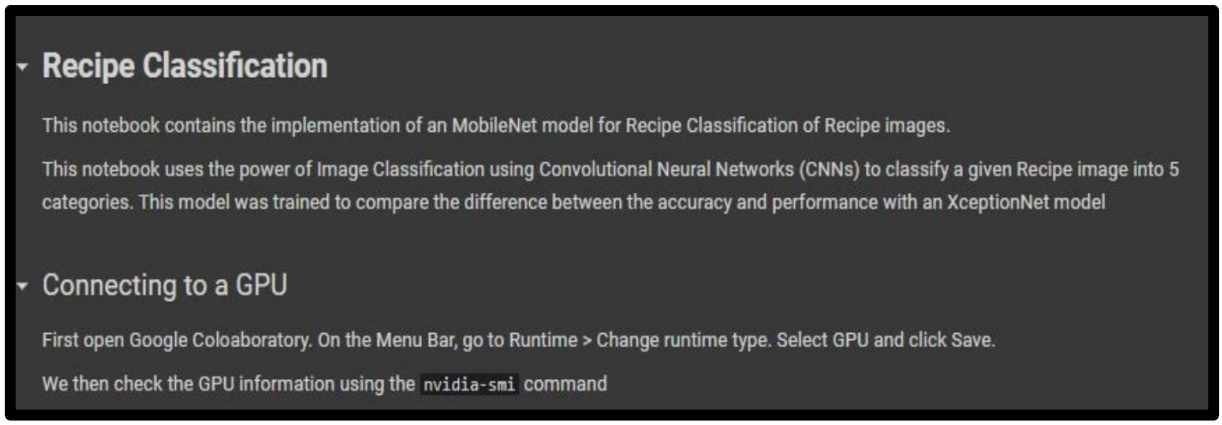


Fig 4.60: MobileNet- Mounting to GPU1

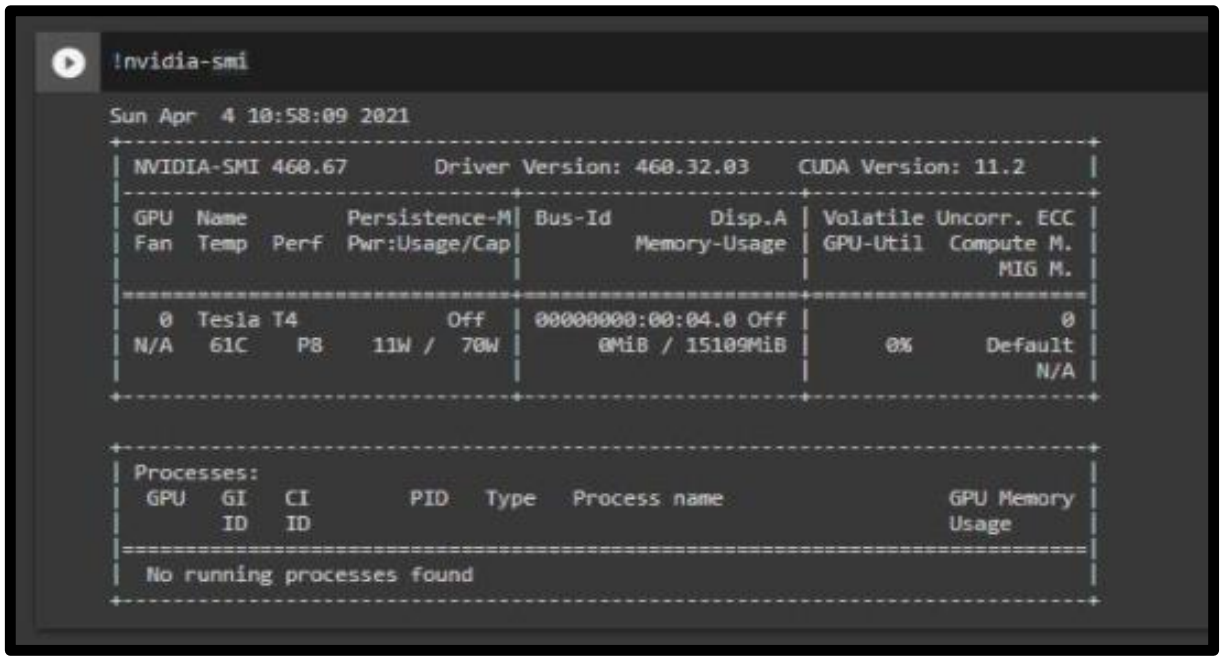


Fig 4.61: MobileNet- Mounting to GPU2

Step 2: Dataset used is Food101 from Kaggle. The dataset has been reduced to 5 classes instead of 101 classes in order to obtain higher accuracy.

The detailed procedure is listed down below in the images.

Then we mount the google drive to google colab.

Fig 4.62: MobileNet- Dataset

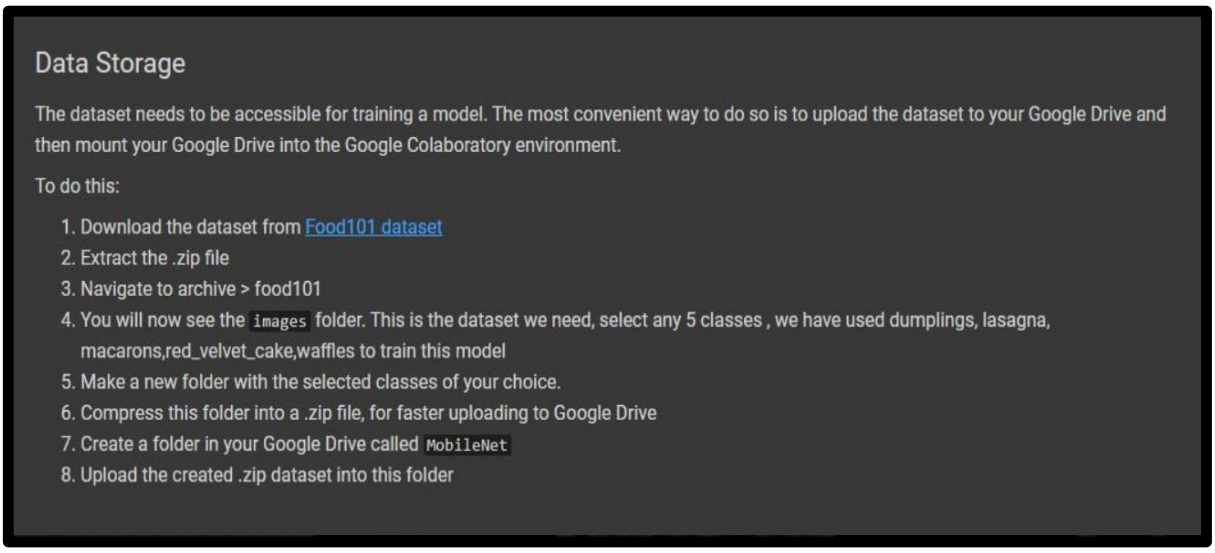


Fig 4.63: MobileNet- Dataset storage

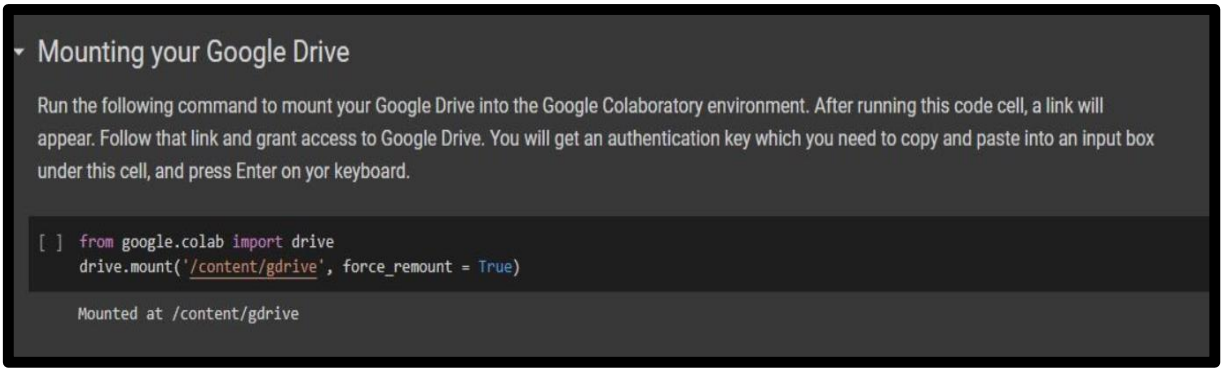


Fig 4.64: MobileNet- mounting google drive

Step3: After mounting the google drive we import the necessary Libraries.

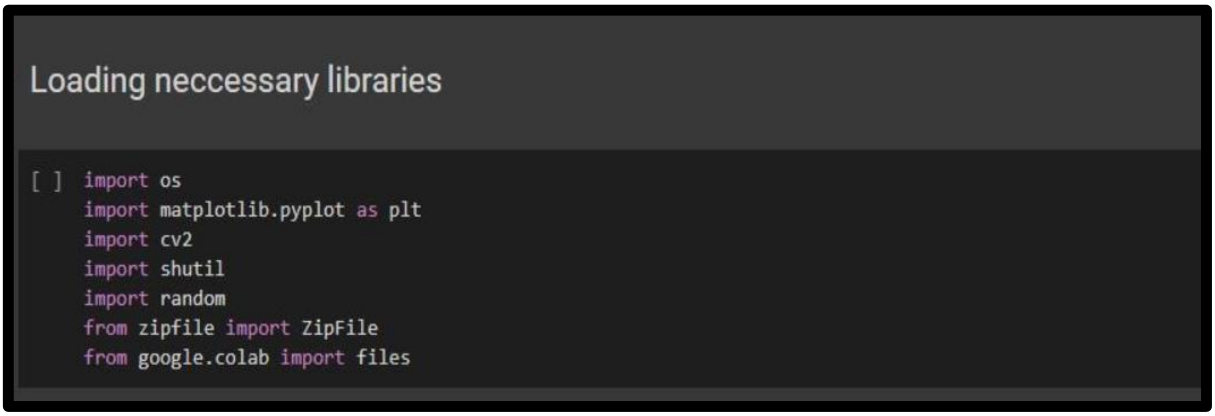


Fig 4.65: MobileNet- loading necessary libraries

Step 4: Next Step is to extract the dataset from the google drive where it was first uploaded after that we define the paths to the 5 class folders.

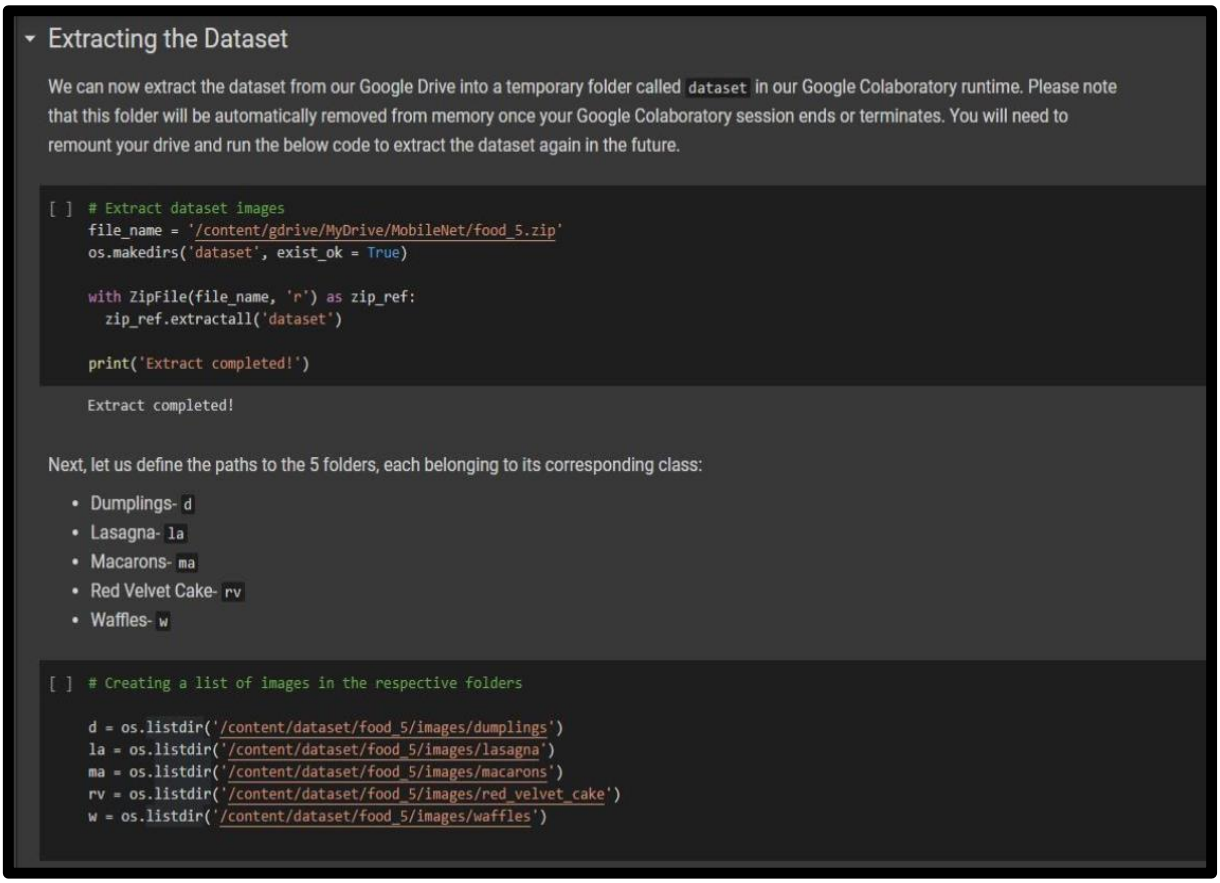


Fig 4.66: MobileNet- Extracting the dataset

Step 5 - Visualizing Dumpling food samples from the dataset.

Dataset Visualization

Let us view some samples of each class within our dataset.

Visualizing Dumpling samples from the dataset

```
[ ]  # Visualizing Dumpling samples from the dataset
     plt.figure(figsize = (15,15))

     for i in range(12):
         plt.subplot(3, 4, i + 1)
         x = random.randint(0, len(d))
         path = '/content/dataset/food_5/images/dumplings/' + d[x]
         img = cv2.imread(path)
         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
         plt.imshow(img)
         plt.title('Dumplings')

     plt.tight_layout()
     plt.show()
```

Fig 4.67: MobileNet- Dumpling dataset visualization

Fig 4.68: MobileNet- Dumpling dataset visualization output

Step 6 - Visualizing Lasagna food samples from the dataset.

Visualizing Lasagna samples from the dataset

```
[ ]  # Visualizing Lasagna samples from the dataset
     plt.figure(figsize = (15,15))

     for i in range(12):
         plt.subplot(3, 4, i + 1)
         x = random.randint(0, len(la))
         path = '/content/dataset/food_5/images/lasagna/' + la[x]
         img = cv2.imread(path)
         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
         plt.imshow(img)
         plt.title('Lasagna')

     plt.tight_layout()
     plt.show()
```

Fig 4.69: MobileNet- Lasagna dataset visualization

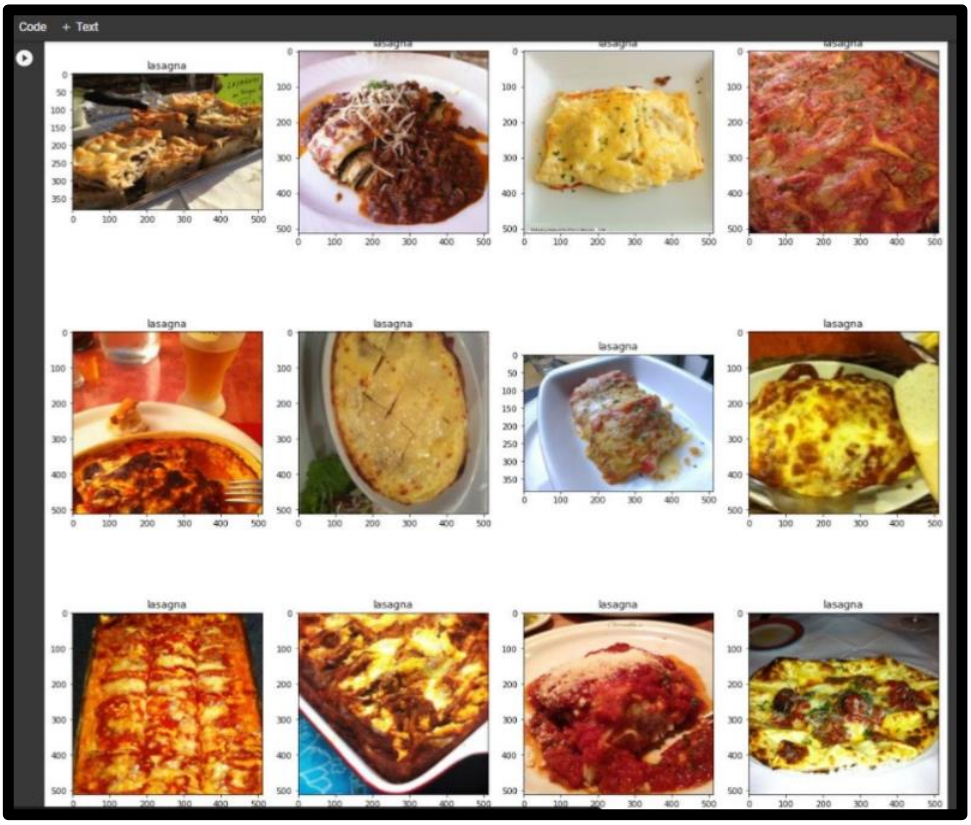


Fig 4.70: MobileNet- Lasagna dataset visualization output

Step 7 - Visualizing Macaron food samples from the dataset.

Visualizing Macarons samples from the dataset

```
[ ]  # Visualizing Macaron samples from the dataset
     plt.figure(figsize = (15,15))

     for i in range(12):
         plt.subplot(3, 4, i + 1)
         x = random.randint(0, len(ma))
         path = '/content/dataset/food_5/images/macarons/' + ma[x]
         img = cv2.imread(path)
         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
         plt.imshow(img)
         plt.title('Macarons')

     plt.tight_layout()
     plt.show()
```

Fig 4.71: MobileNet- Macron dataset visualization

Fig 4.72: MobileNet- Macron dataset visualization output

Step 8 - Visualizing Red Velvet food samples from the dataset.

Visualizing some Red Velvet Cake samples from the dataset

```
[ ]  # Visualizing Red Velvet Cake samples from the dataset
     plt.figure(figsize = (15,15))

     for i in range(12):
         plt.subplot(3, 4, i + 1)
         x = random.randint(0, len(rv))
         path = '/content/dataset/food_5/images/red_velvet_cake/' + rv[x]
         img = cv2.imread(path)
         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
         plt.imshow(img)
         plt.title('Red Velvet Cake')

     plt.tight_layout()
     plt.show()
```

Fig 4.73: MobileNet- Red velvet cake dataset visualization

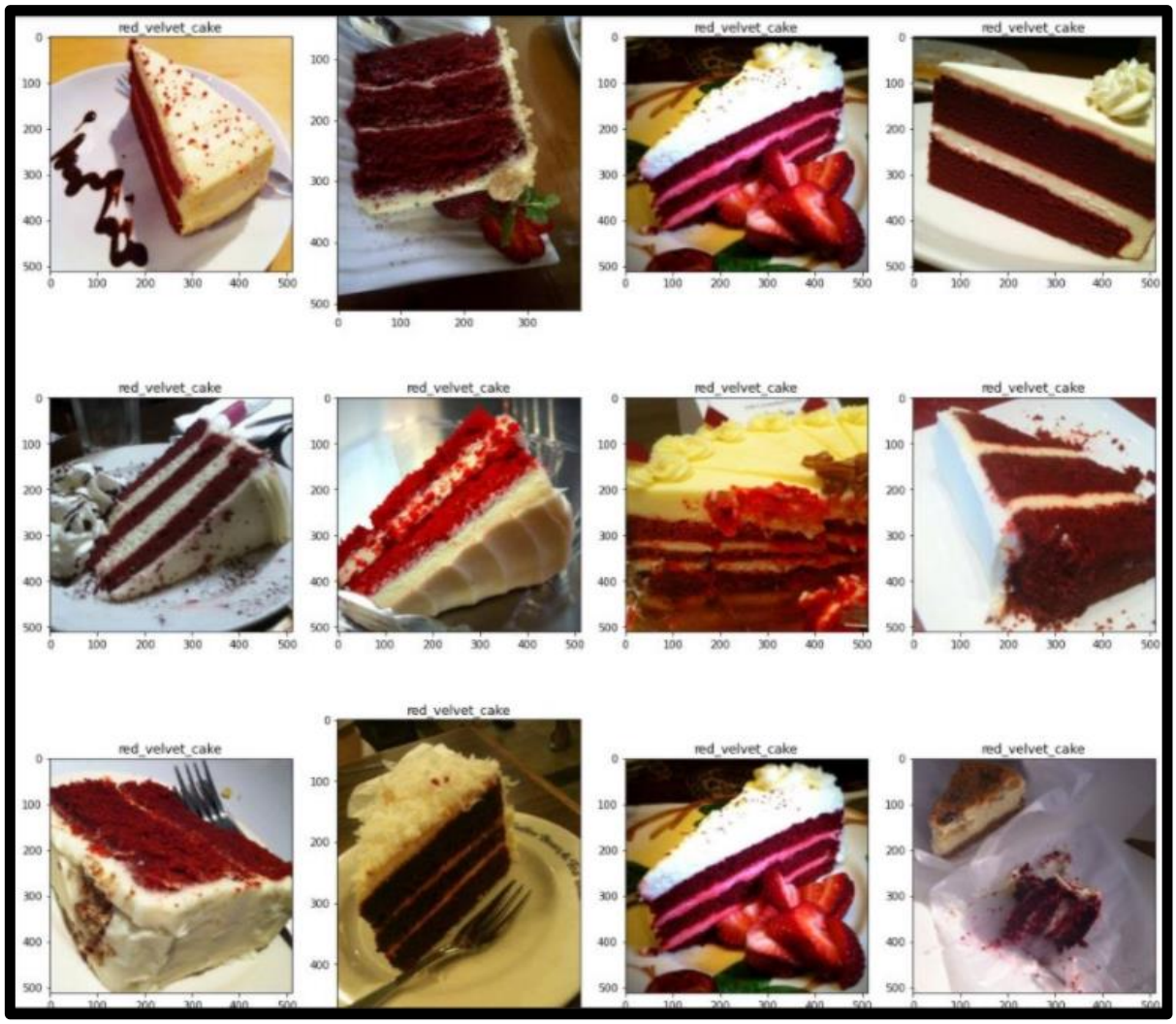


Fig 4.74: MobileNet- Red velvet cake dataset visualization output

Step 9 - Visualizing Waffle food samples from the dataset.

Visualizing Waffles samples from the dataset

```
[ ]  # Visualizing Waffles samples from the dataset
     plt.figure(figsize = (15,15))

     for i in range(12):
         plt.subplot(3, 4, i + 1)
         x = random.randint(0, len(w))
         path = '/content/dataset/food_5/images/waffles/' + w[x]
         img = cv2.imread(path)
         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
         plt.imshow(img)
         plt.title('waffles')

     plt.tight_layout()
     plt.show()
```

Fig 4.75: MobileNet- Waffle dataset visualization

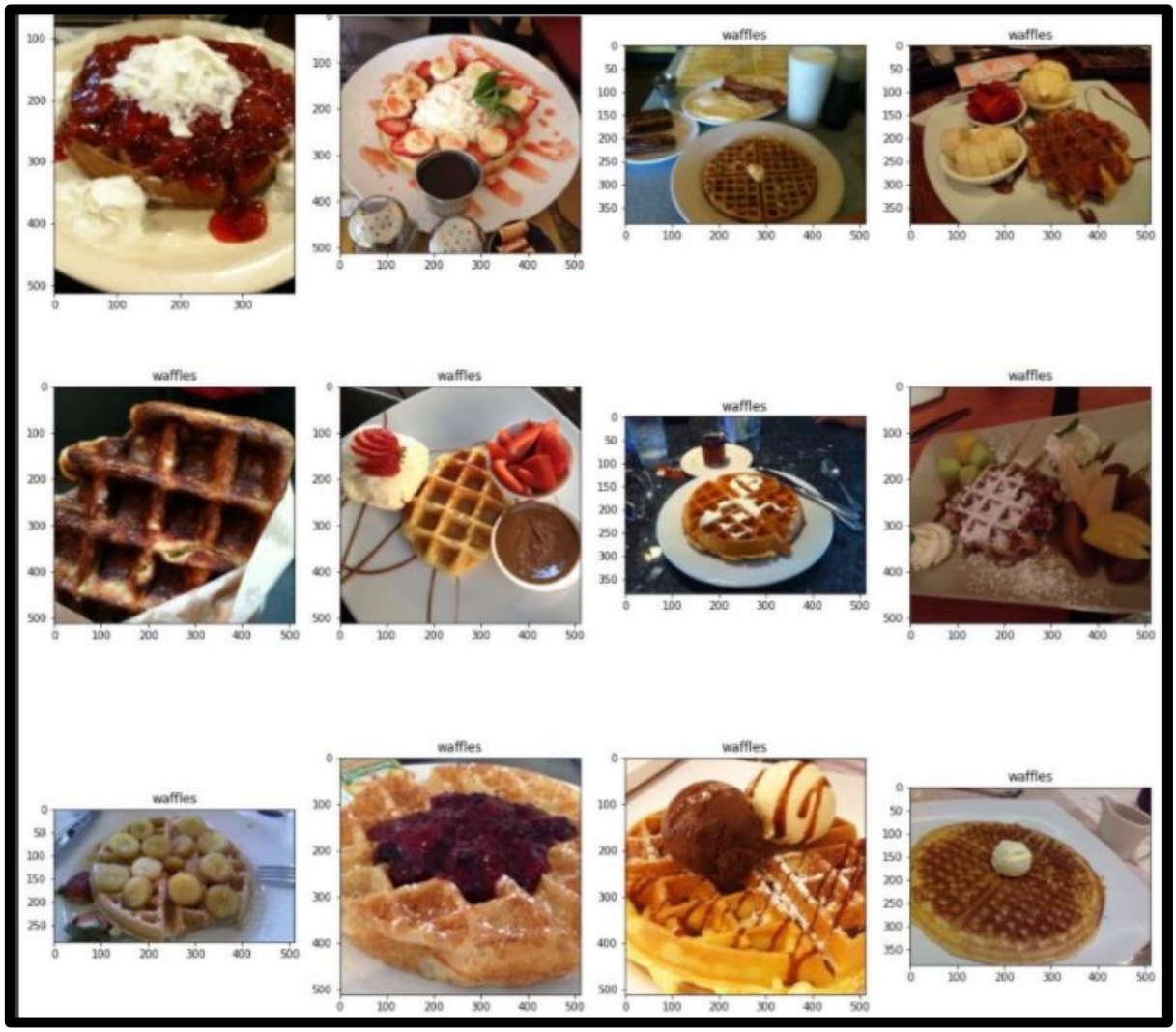


Fig 4.76: MobileNet- Waffle dataset visualization output

Step 10 - Splitting the dataset into train, validation and test sets.
Split-folders library is used to split the dataset into train, validation and test sets.

Fig 4.77: MobileNet- Splitting the dataset

Then count and check the number of images in each of the sets we created.

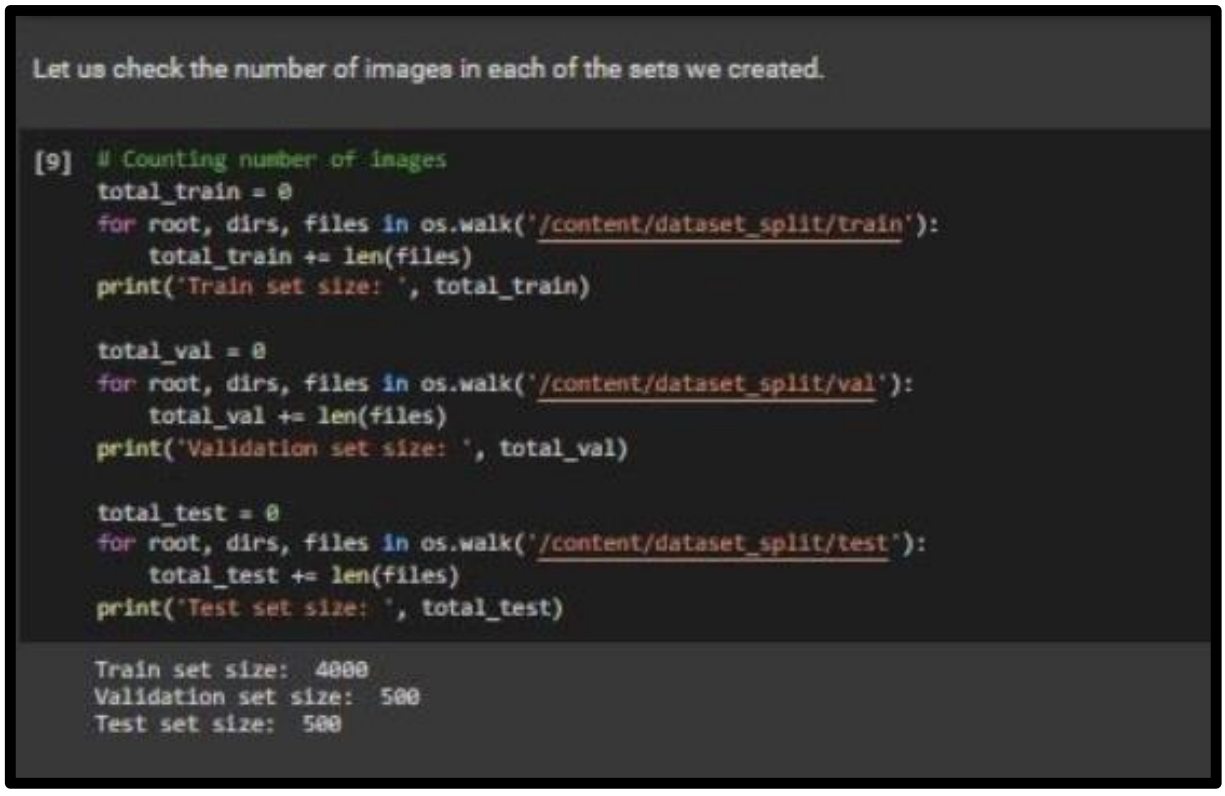


Fig 4.78: MobileNet- Counting no of images

MobileNet Model:

Step 11 - Setting up the Model training where we use a MobileNet model from Keras Applications through Keras.applications. mobilenet.

Start with importing the required libraries.

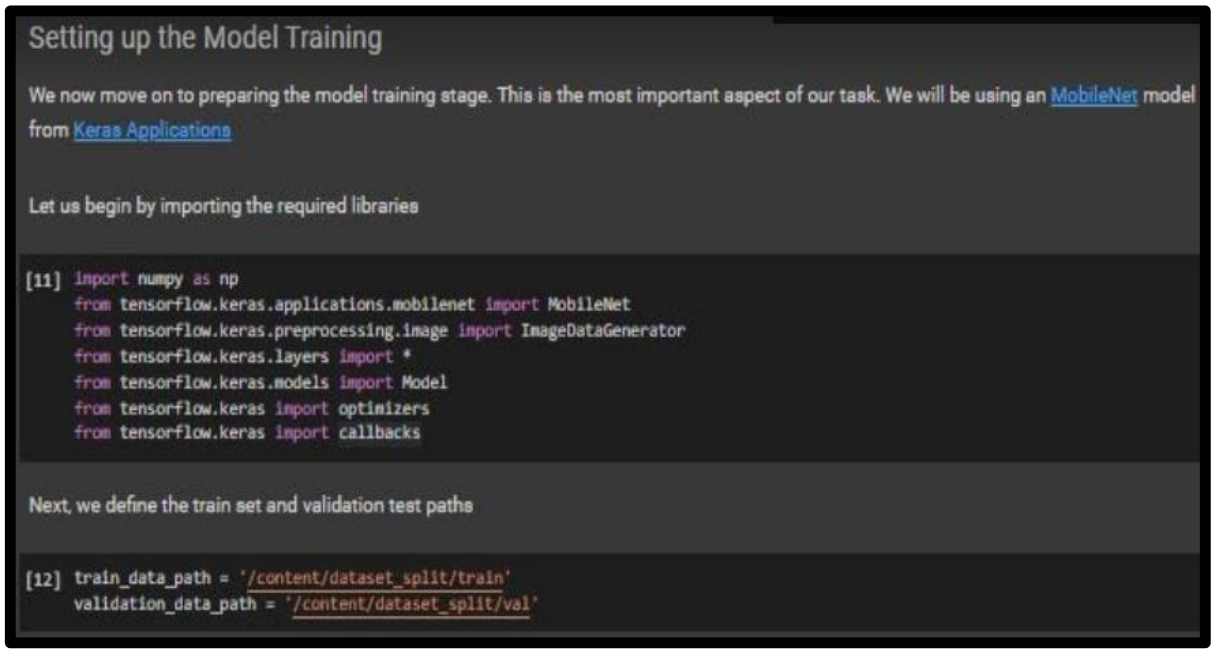


Fig 4.79: MobileNet- Setting up model training

We then define the train set and validation test paths and then the hyper parameters like image width, image height, channels, batch size, validation steps, number of classes, learning rate and epochs.

The next step is to define the training and validation generators using the defined hyper parameters.

ImageDataGenerator: Generate batches of tensor image data with real-time data augmentation. Read more here.

train_generator: Takes the path to a directory & generates batches of augmented data for training, based on the train image data generator.

validation_generator: Takes the path to a directory & generates batches of augmented data for validation, based on the validation image data generator.

Now, we define some training hyperparameters:

- img_width: Width of input image in pixels
- img_height: Height of input image in pixels
- channels: Numbers of color channels in image
- batch_size: The number of training samples to work through before the model's internal parameters are updated
- validation_steps: validation_set_size / batch_size
- classes_num: Number of classes, 3 in our case
- lr: Learning Rate. The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. It controls how quickly the model is adapted to the problem
- epochs: The number of epochs controls the number of complete passes through the training dataset

```
[13] img_width = 224
     img_height = 224
     channels = 3
     batch_size = 16
     classes_num = 5
     epochs = 50
```

We then define our training and validation generators using the defined hyperparameters. More information on this available here.

ImageDataGenerator: Generate batches of tensor image data with real-time data augmentation. Read more here.

train_generator: Takes the path to a directory & generates batches of augmented data for training, based on the train image data generator.

validation_generator: Takes the path to a directory & generates batches of augmented data for validation, based on the validation image data generator.

Fig 4.80: MobileNet- Defining hyperparameters1

```
[14] # Image data generators for train and val sets
     train_datagen = ImageDataGenerator(
         rotation_range = 20,
         zoom_range = 0.15,
         width_shift_range = 0.2,
         height_shift_range = 0.2,
         shear_range = 0.15,
         horizontal_flip = True,
         fill_mode="nearest",rescale = 1. / 255)
     val_datagen = ImageDataGenerator(rescale = 1. / 255)

     # Training generator
     train_generator = train_datagen.flow_from_directory(
         train_data_path,
         target_size = (img_height, img_width),
         batch_size = batch_size,
         class_mode = 'categorical')

     # Validation generator
     validation_generator = val_datagen.flow_from_directory(
         validation_data_path,
         target_size = (img_height, img_width),
         batch_size = batch_size,
         class_mode = 'categorical')

     Found 4000 images belonging to 5 classes.
     Found 500 images belonging to 5 classes.
```

Fig 4.81: MobileNet- Defining hyperparameters2

Step 12 - Define the MobileNet model. The input uses the same image height and width which were defined earlier.

Ingredient weights are used as starting weights to train the model for our custom number of classes.

```
[18] base_mobilenet = MobileNet(
        input_shape=None,
        alpha=1.0,
        depth_multiplier=1,
        dropout=0.001,
        include_top=True,
        weights="imagenet",
        input_tensor=None,
        pooling=None,
        classes=1000,
        classifier_activation="softmax"
    )
```

Fig 4.82: MobileNet- Define MobileNet model

Let us take a look at our base model, without the final few layers.

```
[19] base_mobilenet.summary()
```

```
Model: "mobilenet_1.00_224"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
conv1 (Conv2D)               (None, 112, 112, 32)      864
_________________________________________________________________
conv1_bn (BatchNormalization (None, 112, 112, 32)      128
_________________________________________________________________
conv1_relu (ReLU)            (None, 112, 112, 32)      0
_________________________________________________________________
conv_dw_1 (DepthwiseConv2D)  (None, 112, 112, 32)      288
_________________________________________________________________
conv_dw_1_bn (BatchNormaliza (None, 112, 112, 32)      128
_________________________________________________________________
conv_dw_1_relu (ReLU)        (None, 112, 112, 32)      0
_________________________________________________________________
conv_pw_1 (Conv2D)           (None, 112, 112, 64)      2048
_________________________________________________________________
```

Fig 4.83: MobileNet- Base model summary

Then, we add a few more layers to the base model and use a softmax layer with 5 outputs as the final layer, for the 5 classes in our task.

Let us then take a look at our final model architecture summary!

```
x = Dense(2048,activation='relu')(base_mobilenet.output)
x = Dense(128,activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(64,activation='relu')(x)
x = Dense(5,activation='softmax')(x)
mobilenet_net = Model(inputs = base_mobilenet.input, outputs = x)

mobilenet_net.summary()
```

Fig 4.84: MobileNet- MobileNet summary

Step 13 - Optimizer and Callbacks.

An optimizer is a method or algorithm to update the various parameters that can reduce the loss in much less effort. One of the most commonly used and best optimizers is Adam.

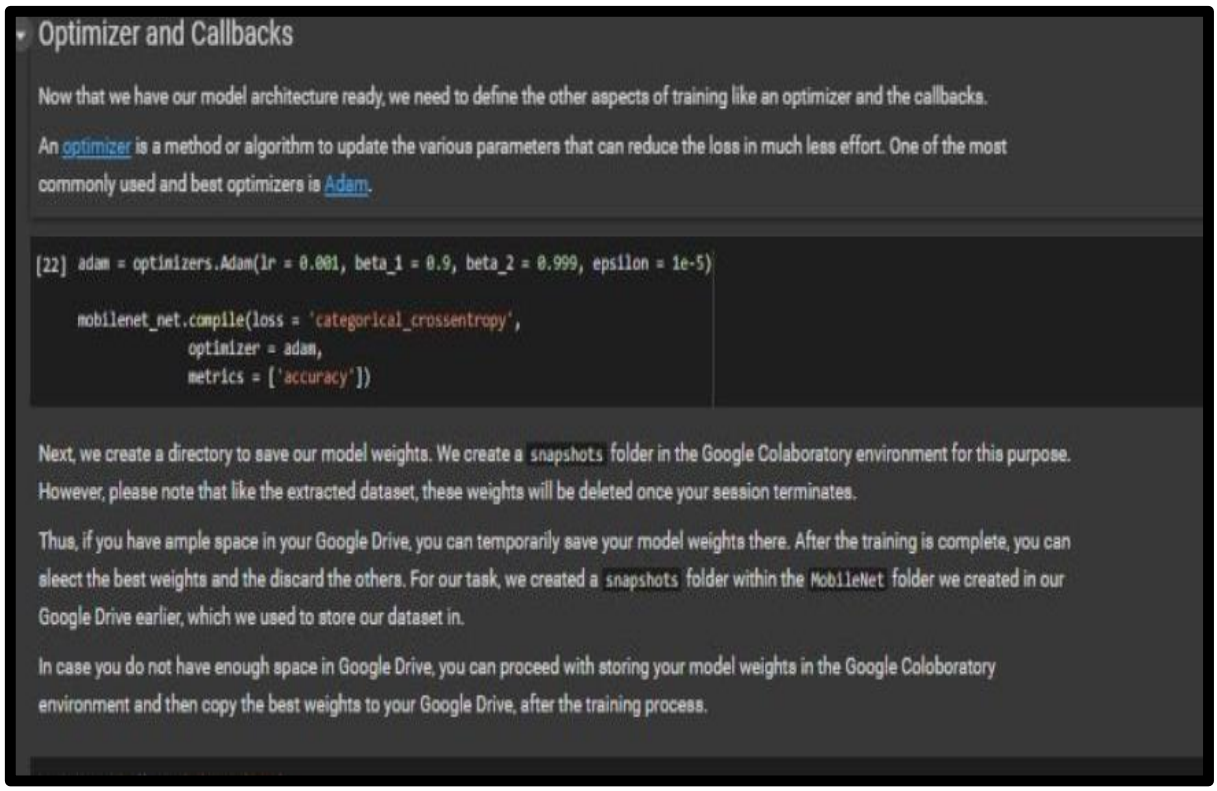


Fig 4.85: MobileNet- Optimizer and Callbacks

Fig 4.86: MobileNet- Folder to store model weights

Next, we create a directory to save our model weights. We create a snapshots folder in the Google Colaboratory environment for this purpose.

However, please note that like the extracted dataset, these weights will be deleted once your session terminates.

Thus, if you have ample space in your Google Drive, you can temporarily save your model weights there. After the training is complete, you can select the best weights and discard the others.

For our task, we created a snapshots folder within the mobilenet folder we created in our Google Drive earlier, which we used to store our dataset in.

In case you do not have enough space in Google Drive, you can proceed with storing your model weights in the Google Collaboratory environment and then copy the best weights to your Google Drive, after the training process.

```
[24] # Directory to store training logs
     log_dir = './tf-log/'

     # Path tp store trained weights. Uncomment the needed one and comment out the other
     ### Google Drive
     file_path = '/content/gdrive/MyDrive/MobileNet/snapshots/best_weight{epoch:03d}.h5'

     '''
     ### Google Colaboratory environment
     file_path = '/content/snapshots/best_weight{epoch:03d}.h5'
     '''

     # Checkpoints
     checkpoints = callbacks.ModelCheckpoint(file_path, monitor = "val_loss", verbose = 0, save_best_only = True, mode = 'auto')

     # Reduce learning rate on plateau
     lr_op = callbacks.ReduceLROnPlateau(monitor = "val_loss", factor = 0.1, patience = 5,
       verbose = 1, mode = "auto", cooldown = 0,
       min_lr = 1e-30)

     # Early stopping
     early_stop = callbacks.EarlyStopping(
         monitor = "val_loss",
         min_delta = 0,
         patience = 10,
         verbose = 1,
         mode = "auto",
         baseline = None,
         restore_best_weights = False,
     )

     # Add all the created callbacks
     cbks = [checkpoints, lr_op, early_stop]
```

Fig 4.87: MobileNet- Directory to store training logs

Model Training

Finally, we can begin with our model trainig. We call the fit() method for this purpose. We pass the train and validation generators, number of epochs, callbacks, steps_per_epochs (train_set_size / batch_size), validation_steps(validation_set_size / batch_size).

```
[ ] # Model training
    history = mobilenet_net.fit(
        train_generator,
        steps_per_epoch = 250,
        epochs = 110,
        validation_data = validation_generator,
        callbacks = cbks,
        validation_steps = 32)
```

Fig 4.88: MobileNet- Model training

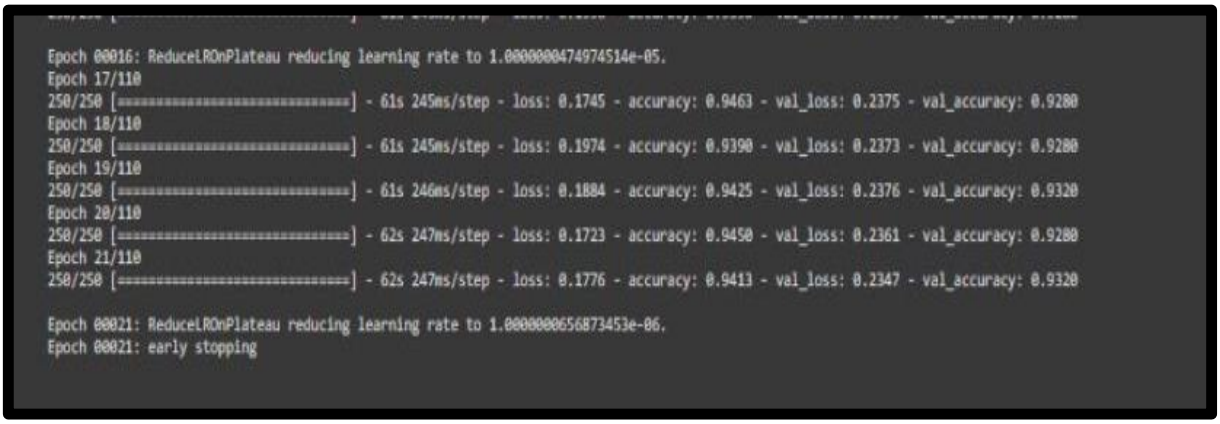


Fig 4.89: MobileNet- Results

Training statistics

Fig 4.90: MobileNet- Training statistics

Plotting the Graphs

For Visualizing the graphs for train loss, train accuracy, validation loss, validation accuracy.

The loss graphs should ideally show a gradual decrease, while the accuracy graphs should ideally show a gradual increase.

Train Loss

We can see the train loss starting at a very high value initially. As the no. of epochs increase, the model learns and the train loss decreases after some initial instability.

We can see the initial train loss around/between 0.15-0.18 towards the final few epochs.

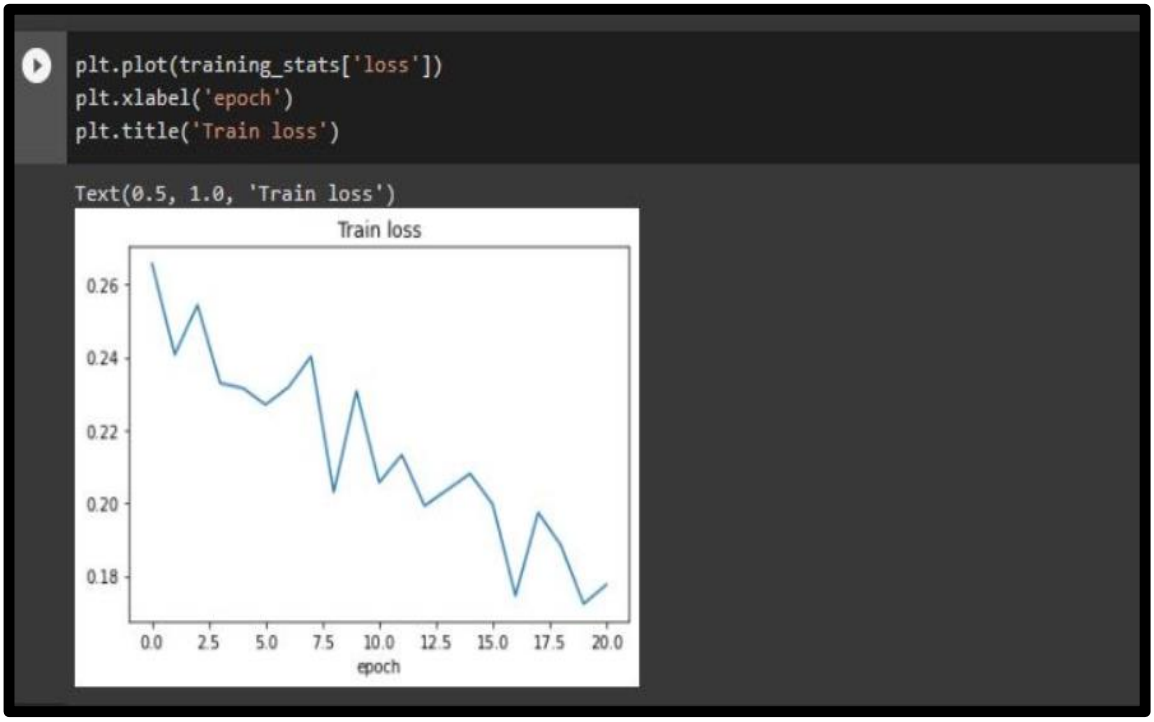


Fig 4.91: MobileNet- Train loss graph

Train Accuracy

We can see the train accuracy starting at a very low value initially. As the number of epochs increases, the model learns and the train accuracy increases after some initial instability. We can see that the train accuracy is around 94% towards the final few epochs.

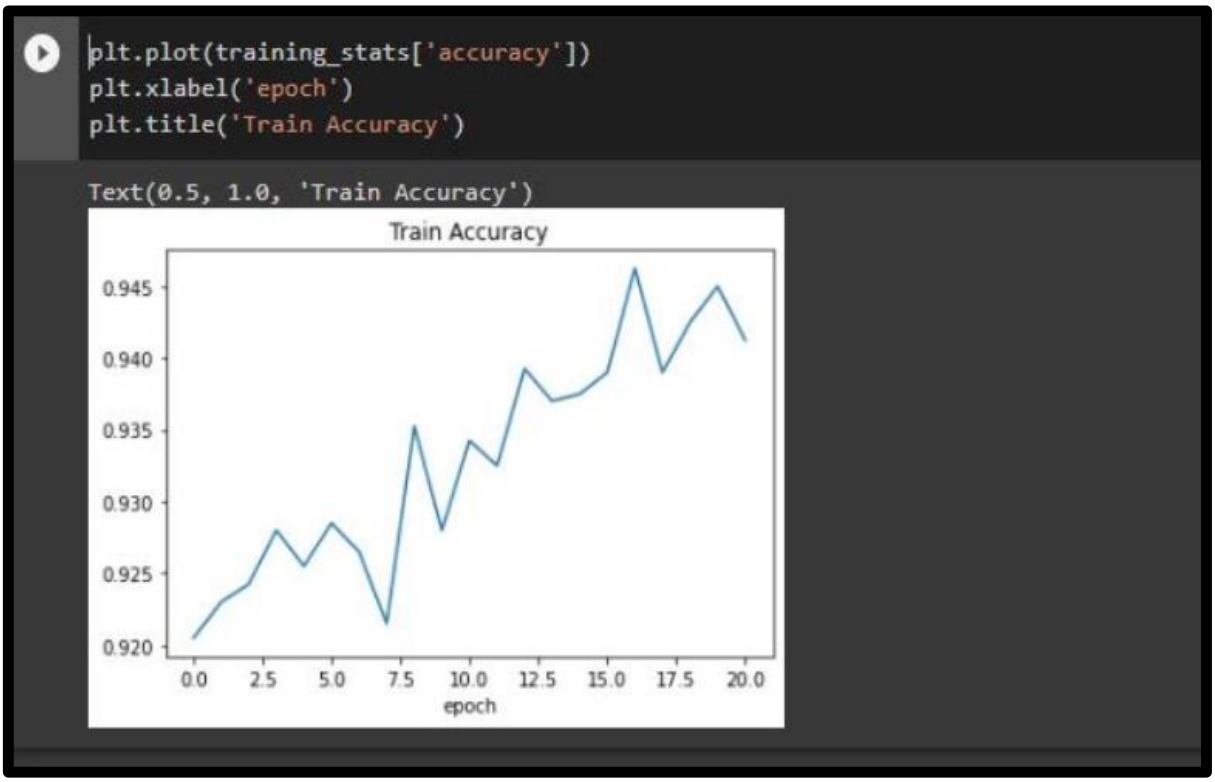


Fig 4.92: MobileNet- Train accuracy graph

Validation Loss

We can see the validation loss being very unstable and exploding to a very high value initially. As the number of epochs increases, the model learns and the validation loss decreases after some initial instability. We can see that the train loss is around 0.235 towards the final few epochs.

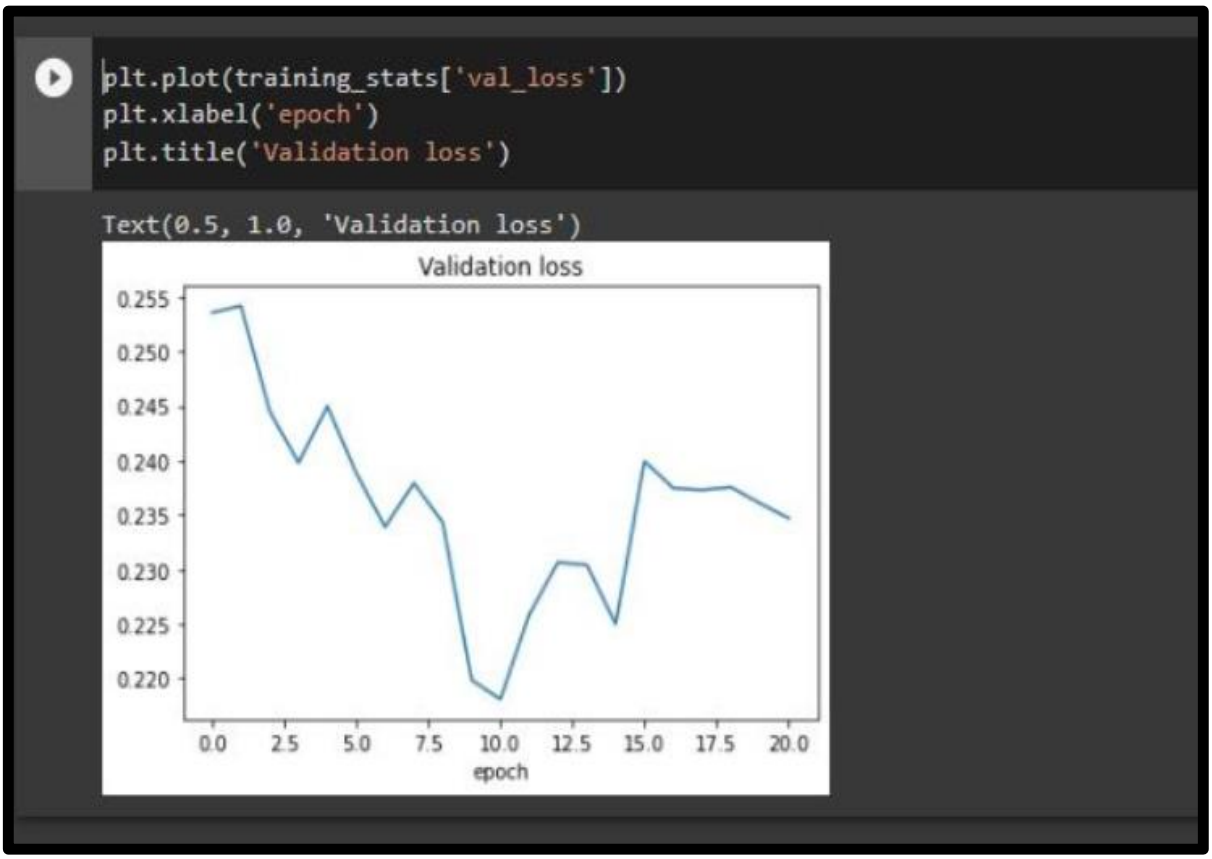


Fig 4.93: MobileNet- Validation loss graph

Validation Accuracy

We can see the validation accuracy starting at a very low value initially and acting very unstable. As the number of epochs increases, the model learns and the validation accuracy increases after some initial instability. We can see that the validation accuracy is around 93% towards the final few epochs.

```
plt.plot(training_stats['val_accuracy'])
plt.xlabel('epoch')
plt.title('Validation Accuracy')
```

```
Text(0.5, 1.0, 'Validation Accuracy')
```

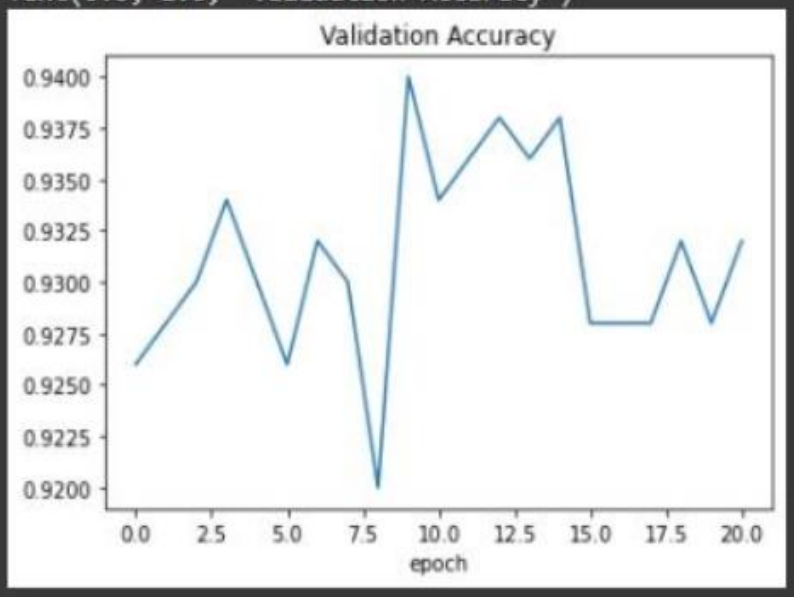


Fig 4.94: MobileNet- Validation accuracy graph

Combining train and validation graphs

Let us visualize the train and validation graphs together for comparison. The blue line represents the train metric and the orange line represents the validation metric.

```
fig, ax = plt.subplots(1, 2, figsize=(30, 10))
ax = ax.ravel()

for i, metric in enumerate(["accuracy", "loss"]):
    ax[i].plot(mobilenet_net.history.history[metric][:])
    ax[i].plot(mobilenet_net.history.history["val_" + metric][:])
    ax[i].set_title("Model {}".format(metric))
    ax[i].set_xlabel("epochs")
    ax[i].set_ylabel(metric)
    ax[i].legend(["train", "val"])
```

Fig 4.95: MobileNet- Train and validation comparison

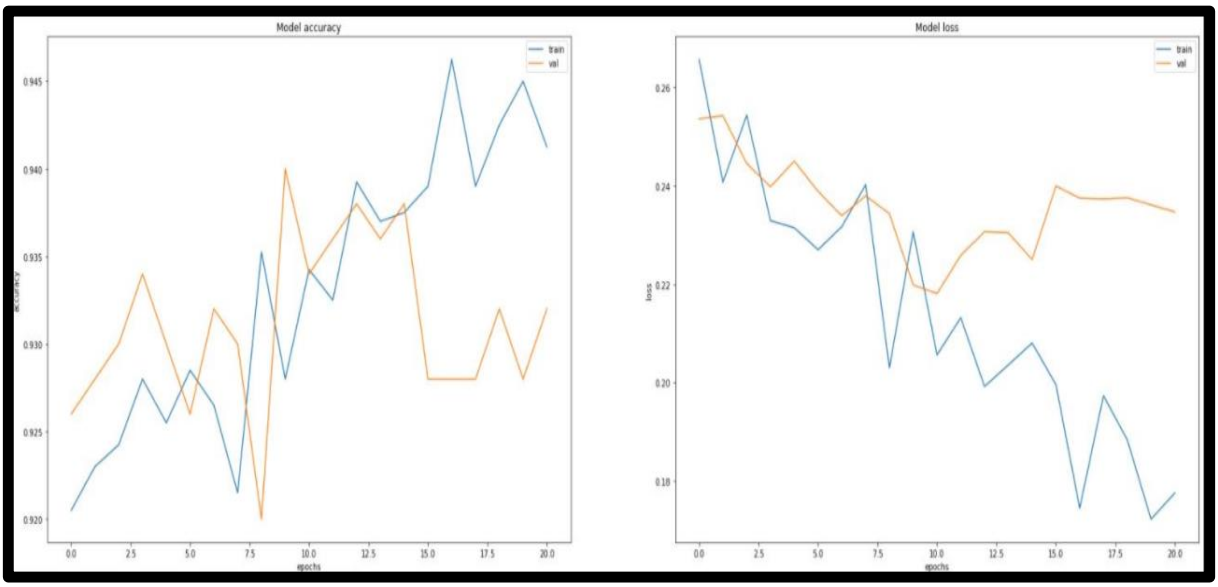


Fig 4.96: MobileNet- Train and validation comparison graph

Testing the model

It is now time to test our trained model using our test set! We will also plot the incorrectly classified images to get an idea of where and why the model might be misclassifying.

As always, we first import the needed packages.

Testing for individual Samples: Here we test the individual samples for their accuracy and if the images are correctly classified or not.

Testing for Dumpling samples

```
[28] # Testing for Dumpling samples
     class_dict = {0: 'dumplings', 1: 'lasagna', 2: 'macarons', 3: 'red_velvet_cake', 4: 'waffles'}
     dumplings_count = 0

     food_d = os.listdir('dataset_split/test/dumplings')

     for img in food_d:
         img_path = 'dataset_split/test/dumplings/' + img
         img = load_img(img_path, target_size = (224,224))
         img_tensor = img_to_array(img)
         img_tensor = np.expand_dims(img_tensor, axis=0)
         img_tensor /= 255.

         ### predict() takes image tensor of (batchsize,heighht,width,channels)
         ### Returns an array for all images in batch with probability for each class
         pred = model.predict(img_tensor)

         classes = np.argmax(pred)

         if classes == 0:
           dumplings_count += 1
         else:
           plt.imshow(img_tensor[0])
           plt.show()
           print('Predicted class probability for image: ', pred[0][classes])

           ### Class number with highest probablity (Classification result of CNN)
           print('Predicted Class :', class_dict[classes])
```

Fig 4.97: Testing for dumpling sample

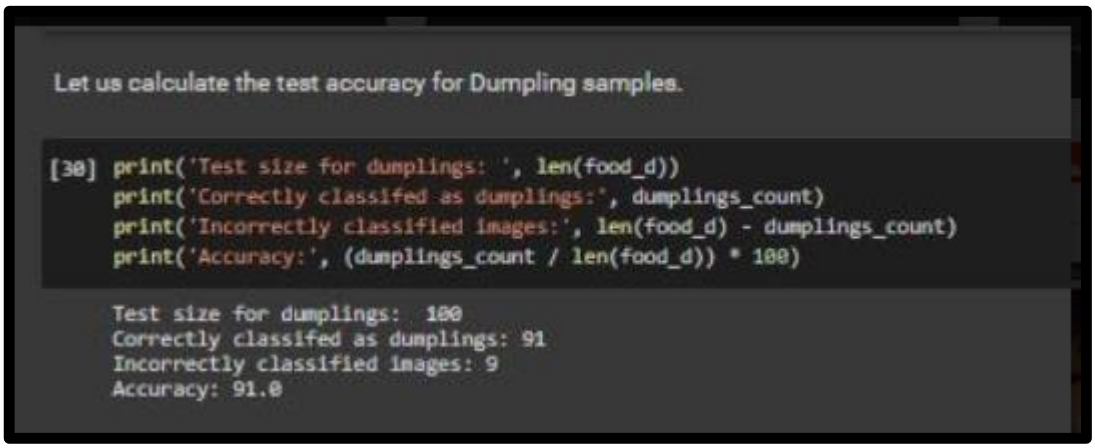
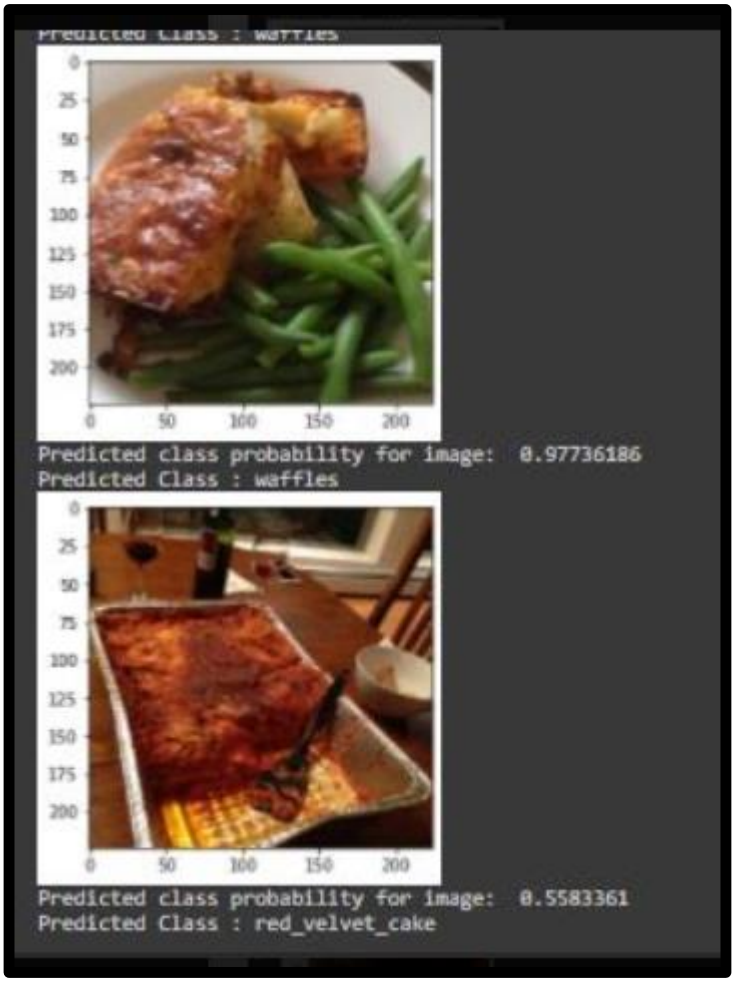


Fig 4.98: Testing for dumpling sample output

Testing for Lasagna samples

```
# Testing for lasagna samples
class_dict = {0: 'dumplings', 1: 'lasagna', 2: 'macarons', 3: 'red_velvet_cake', 4: 'waffles'}
lasagna_count = 0

food_la = os.listdir('dataset_split/test/lasagna')

for img in food_la:
    img_path = 'dataset_split/test/lasagna/' + img
    img = load_img(img_path, target_size = (224,224))
    img_tensor = img_to_array(img)
    img_tensor = np.expand_dims(img_tensor, axis=0)
    img_tensor /= 255.

    ### predict() takes image tensor of (batchsize,heighht,width,channels)
    ### Returns an array for all images in batch with probability for each class
    pred = model.predict(img_tensor)

    classes = np.argmax(pred)

    if classes == 1:
      lasagna_count += 1
    else:
      plt.imshow(img_tensor[0])
      plt.show()
      print('Predicted class probability for image: ', pred[0][classes])

      ### Class number with highest probablity (Classification result of CNN)
      print('Predicted Class :', class_dict[classes])
```

Fig 4.99: Testing for lasagna sample



Let us calculate the test accuracy for Lasagna samples.

```
print('Test size for lasagna: ', len(food_la))
print('Correctly classifed as lasagna:', lasagna_count)
print('Incorrectly classified images:', len(food_la) - lasagna_count)
print('Accuracy:', (lasagna_count / len(food_la)) * 100)
```

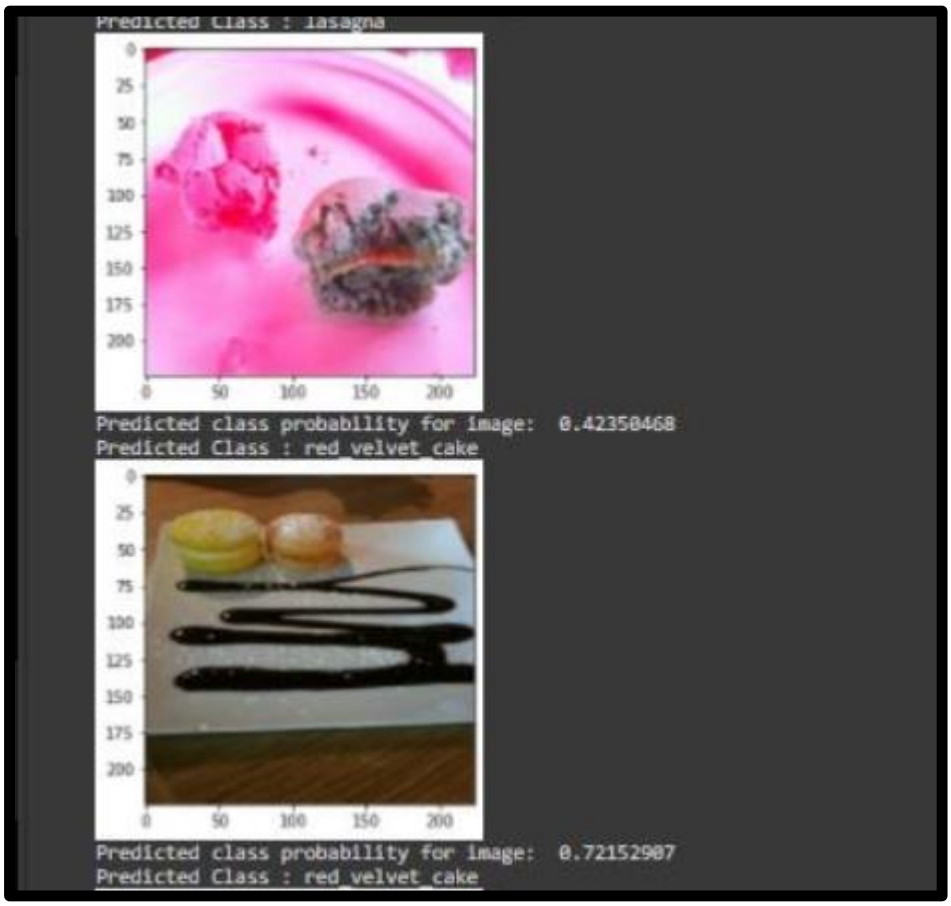
```
Test size for lasagna:  100
Correctly classifed as lasagna: 92
Incorrectly classified images: 8
Accuracy: 92.0
```

Fig 4.100: Testing for lasagna sample output

Testing for Macarons samples

```
[44] # Testing for Macarons samples
     class_dict = {0: 'dumplings', 1: 'lasagna', 2: 'macarons', 3: 'red_velvet_cake', 4: 'waffles'}
     macarons_count = 0

     food_ma = os.listdir('dataset_split/test/macarons')

     for img in food_ma:
         img_path = 'dataset_split/test/macarons/' + img
         img = load_img(img_path, target_size = (224,224))
         img_tensor = img_to_array(img)
         img_tensor = np.expand_dims(img_tensor, axis=0)
         img_tensor /= 255.

         ### predict() takes image tensor of (batchsize,heighht,width,channels)
         ### Returns an array for all images in batch with probability for each class
         pred = model.predict(img_tensor)

         classes = np.argmax(pred)

         if classes == 2:
           macarons_count += 1
         else:
           plt.imshow(img_tensor[0])
           plt.show()
           print('Predicted class probability for image: ', pred[0][classes])

           ### Class number with highest probablity (Classification result of CNN)
           print('Predicted Class :', class_dict[classes])
```

Fig 4.101: Testing for macron sample



Let us calculate the test accuracy for Macaron samples.

```
[45] print('Test size for macarons: ', len(food_ma))
     print('Correctly classifed as macarons:', macarons_count)
     print('Incorrectly classified images:', len(food_ma) - macarons_count)
     print('Accuracy:', (macarons_count / len(food_ma)) * 100)
```

```
Test size for macarons:  100
Correctly classifed as macarons: 94
Incorrectly classified images: 6
Accuracy: 94.0
```

Double-click (or enter) to edit

Fig 4.102: Testing for macron sample output

Testing for Red Velvet Cake samples

```
[46] # Testing for Red Velvet Cake samples
     class_dict = {0: 'dumplings', 1: 'lasagna', 2: 'macarons', 3: 'red_velvet_cake', 4: 'waffles'}
     red_velvet_cake_count = 0

     food_rv = os.listdir('dataset_split/test/red_velvet_cake')

     for img in food_rv:
         img_path = 'dataset_split/test/red_velvet_cake/' + img
         img = load_img(img_path, target_size = (224,224))
         img_tensor = img_to_array(img)
         img_tensor = np.expand_dims(img_tensor, axis=0)
         img_tensor /= 255.

         ### predict() takes image tensor of (batchsize,heighht,width,channels)
         ### Returns an array for all images in batch with probability for each class
         pred = model.predict(img_tensor)

         classes = np.argmax(pred)

         if classes == 3:
           red_velvet_cake_count += 1
         else:
           plt.imshow(img_tensor[0])
           plt.show()
           print('Predicted class probability for image: ', pred[0][classes])

           ### Class number with highest probablity (Classification result of CNN)
           print('Predicted Class :', class_dict[classes])
```

Fig 4.103: Testing for red velvet cake sample

Let us calculate the test accuracy for Red Velvet Cake samples.

```
[47] print('Test size for Red Velvet Cake: ', len(food_rv))
     print('Correctly classifed as Red Velvet Cake:', red_velvet_cake_count)
     print('Incorrectly classified images:', len(food_rv) - red_velvet_cake_count)
     print('Accuracy:', (red_velvet_cake_count / len(food_rv)) * 100)

     Test size for Red Velvet Cake:  100
     Correctly classifed as Red Velvet Cake: 95
     Incorrectly classified images: 5
     Accuracy: 95.0
```

Fig 4.104: Testing for red velvet cake sample output

Testing for Waffles samples

```
# Testing for Waffle samples
class_dict = {0: 'dumplings', 1: 'lasagna', 2: 'macarons', 3: 'red_velvet_cake', 4: 'waffles'}
waffles_count = 0

food_w = os.listdir('dataset_split/test/waffles')

for img in food_w:
    img_path = 'dataset_split/test/waffles/' + img
    img = load_img(img_path, target_size = (224,224))
    img_tensor = img_to_array(img)
    img_tensor = np.expand_dims(img_tensor, axis=0)
    img_tensor /= 255.

    ### predict() takes image tensor of (batchsize,heighht,width,channels)
    ### Returns an array for all images in batch with probability for each class
    pred = model.predict(img_tensor)

    classes = np.argmax(pred)

    if classes == 4:
      waffles_count += 1
    else:
      plt.imshow(img_tensor[0])
      plt.show()
      print('Predicted class probability for image: ', pred[0][classes])

      ### Class number with highest probablity (Classification result of CNN)
      print('Predicted Class :', class_dict[classes])
```

Fig 4.105: Testing for waffle sample

Let us calculate the test accuracy for Waffle samples.

```
[49] print('Test size for Waffles: ', len(food_w))
     print('Correctly classifed as Waffles:', waffles_count)
     print('Incorrectly classified images:', len(food_w) - waffles_count)
     print('Accuracy:', (waffles_count / len(food_w)) * 100)
```

```
Test size for Waffles:  100
Correctly classifed as Waffles: 88
Incorrectly classified images: 12
Accuracy: 88.0
```

Fig 4.106: Testing for waffle sample output

4.6 Comparison between MobileNet and Xception model

We have trained two machine learning models i.e. Xception and MobileNet models in order to compare the difference between the accuracy and performance of both models. Both the models use the Food101 dataset obtained from Kaggle. Although the dataset originally contained 101 classes, we have reduced it to just 5 classes for the sake of our project.

Xception:

Xception stands for "extreme inception." Xception is an extension of the inception Architecture which replaces the standard Inception modules with depthwise Separable Convolutions.It reframes the way we look at neural nets — conv nets in particular. And, as the name suggests, it takes the principles of Inception to an extreme.

MobileNet:

MobileNet uses depth wise separable convolutions. This convolution block was at first introduced by Xception. A depthwise separable convolution is made of two operations: a depthwise convolution and a pointwise convolution.

Xception and MobileNet both use depthwise separable convolution, but the purpose of the two is different. Xception uses depthwise separable convolution while increasing the amount of network parameters to compare the effect, mainly to investigate the effectiveness of this structure. MobileNet uses depthwise separable convolution to compress and speed up, and the amount of parameters is significantly reduced. The purpose is not to improve performance but speed.

Comparing the training and epochs of the two models for the same dataset:

In both cases the dataset and the number of classes are the same.

Xception: For this model, we have set the number of epochs for training as 110. However, the model training gave us its highest accuracy of 98.68% at the 30th epoch and therefore the training stopped there.

```
# Model training
history = xception_net.fit(
    train_generator,
    steps_per_epoch = 250,
    epochs = 110,
    validation_data = validation_generator,
    callbacks = cbks,
    validation_steps = 32)
```

```
Epoch 00025: ReduceLROnPlateau reducing learning rate to 1.0000000474974514e-05.
Epoch 26/110
250/250 [==============================] - 73s 293ms/step - loss: 0.0350 - accuracy: 0.9890 - val_loss: 0.1824 - val_accuracy: 0.9680
Epoch 27/110
250/250 [==============================] - 73s 293ms/step - loss: 0.0400 - accuracy: 0.9889 - val_loss: 0.1822 - val_accuracy: 0.9700
Epoch 28/110
250/250 [==============================] - 73s 293ms/step - loss: 0.0317 - accuracy: 0.9886 - val_loss: 0.1915 - val_accuracy: 0.9700
Epoch 29/110
250/250 [==============================] - 73s 293ms/step - loss: 0.0369 - accuracy: 0.9878 - val_loss: 0.1851 - val_accuracy: 0.9720
Epoch 30/110
250/250 [==============================] - 74s 294ms/step - loss: 0.0439 - accuracy: 0.9888 - val_loss: 0.1833 - val_accuracy: 0.9700

Epoch 00030: ReduceLROnPlateau reducing learning rate to 1.0000000656873453e-06.
Epoch 00030: early stopping
```

Fig 4.107: Results for Xception training

The training statistics for the Xception model are as follows:

```
[ ] training_stats = history.history
    print(training_stats)

    {'loss': [0.9760598540306091, 0.7416197061538696, 0.5999800562858582, 0.4745841324329376, 0.4488547146320343, 0.3705286383628845, 0.3408872485166
```

Best Model Weights Statistics

- Train Loss: 0.0360
- Train Accuracy: 0.9868
- Val Loss: 0.0486
- Val Accuracy: 0.9857

Fig 4.108: Statistics for Xception training

MobileNet: The number of epochs we set for this model was the same as that of the Xception model i.e. 110. However, the MobileNet model gave us its highest accuracy in the 21st epoch and therefore the training stopped there. The accuracy it gave is 94.13%.

```
# Model training
history = mobilenet_net.fit(
    train_generator,
    steps_per_epoch = 250,
    epochs = 110,
    validation_data = validation_generator,
    callbacks = cbks,
    validation_steps = 32)
```

```
Epoch 00016: ReduceLROnPlateau reducing learning rate to 1.0000000474974514e-05.
Epoch 17/110
250/250 [==============================] - 61s 245ms/step - loss: 0.1745 - accuracy: 0.9463 - val_loss: 0.2375 - val_accuracy: 0.9280
Epoch 18/110
250/250 [==============================] - 61s 245ms/step - loss: 0.1974 - accuracy: 0.9390 - val_loss: 0.2373 - val_accuracy: 0.9280
Epoch 19/110
250/250 [==============================] - 61s 246ms/step - loss: 0.1884 - accuracy: 0.9425 - val_loss: 0.2376 - val_accuracy: 0.9320
Epoch 20/110
250/250 [==============================] - 62s 247ms/step - loss: 0.1723 - accuracy: 0.9450 - val_loss: 0.2361 - val_accuracy: 0.9280
Epoch 21/110
250/250 [==============================] - 62s 247ms/step - loss: 0.1776 - accuracy: 0.9413 - val_loss: 0.2347 - val_accuracy: 0.9320

Epoch 00021: ReduceLROnPlateau reducing learning rate to 1.0000000656873453e-06.
Epoch 00021: early stopping
```

Fig 4.109: Results for MobileNet training

The training statistics for the MobileNet model are as follows:

```
[ ] training_stats = history.history
    print(training_stats)
```

Best Model Weights Statistics

- Train Loss: 0.1776
- Train Accuracy: 0.9413
- Val Loss: 0.2347
- Val Accuracy: 0.9320

Fig 4.110: Statistics for MobileNet training

Comparing graphs for both the models:

Train loss: Train Loss is the value of the objective function that we are minimizing. Basically it is a number indicating how bad the model's prediction was on a single example. The goal of training a model is to find a set of weights and biases that have low loss, on average, across all examples.

Train loss for Xception model-

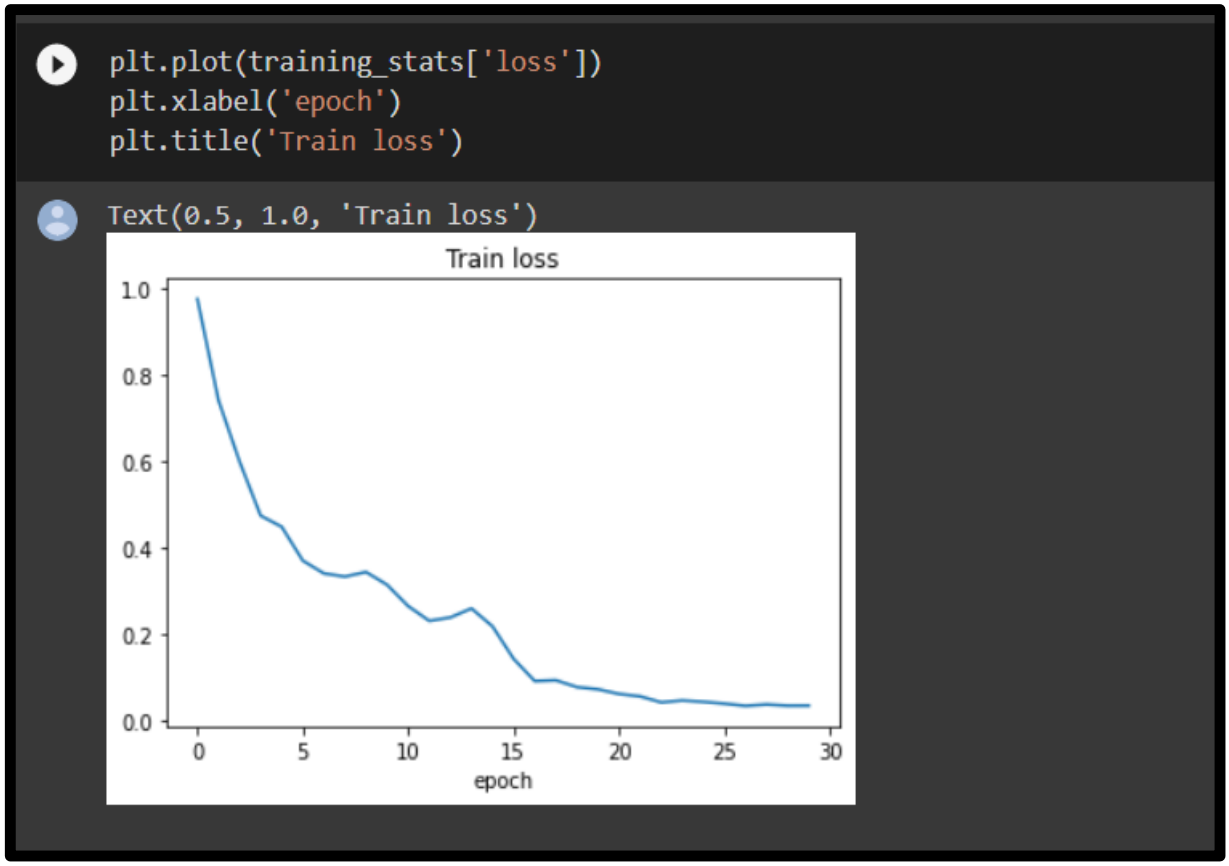


Fig 4.111: Train loss for Xception model

We can see the train loss starting at a very high value initially. As the number of epochs increase, the model learns and the train loss decreases after some initial instability. We can see that the train loss is around 0.03-0.04 towards the final few epochs.

Train loss for MobileNet model-

```
plt.plot(training_stats['loss'])
plt.xlabel('epoch')
plt.title('Train loss')
```

```
Text(0.5, 1.0, 'Train loss')
```

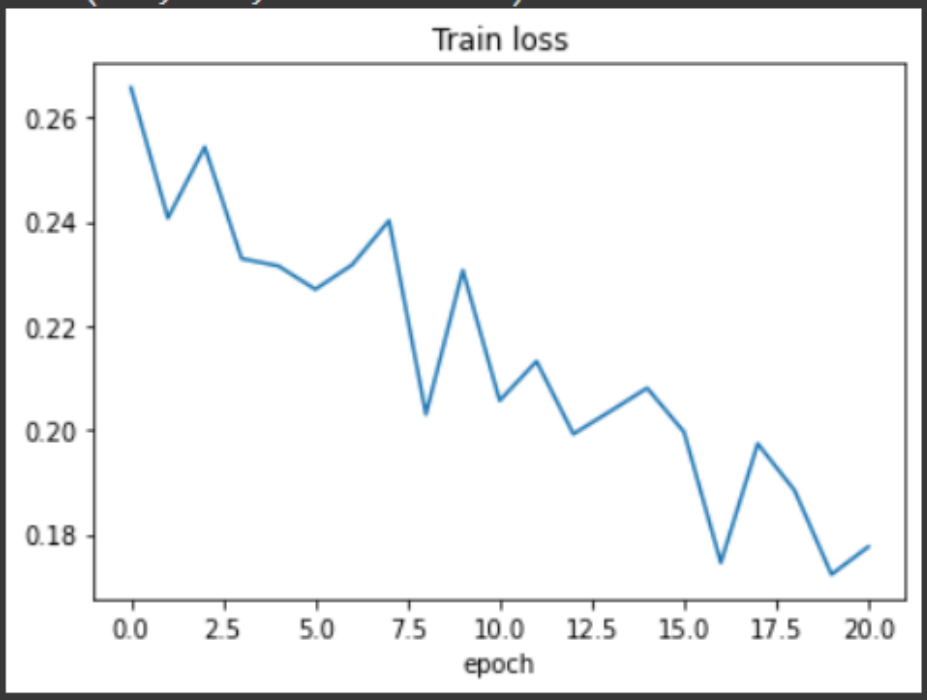


Fig 4.112: Train loss for MobileNet model

We can see the train loss starting at a very high value initially. As the number of epochs increase, the model learns and the train loss decreases after some initial instability. Towards the final epochs, it gives us a loss of 0.15-0.18.

Train Accuracy: It is the accuracy of a model on examples it was constructed on. Training accuracy is usually the accuracy you get if you apply the model on the training data. The aim is to get as high a train accuracy for a model.

Train Accuracy for Xception model-

```
[ ]  plt.plot(training_stats['accuracy'])
     plt.xlabel('epoch')
     plt.title('Train Accuracy')

     Text(0.5, 1.0, 'Train Accuracy')
```

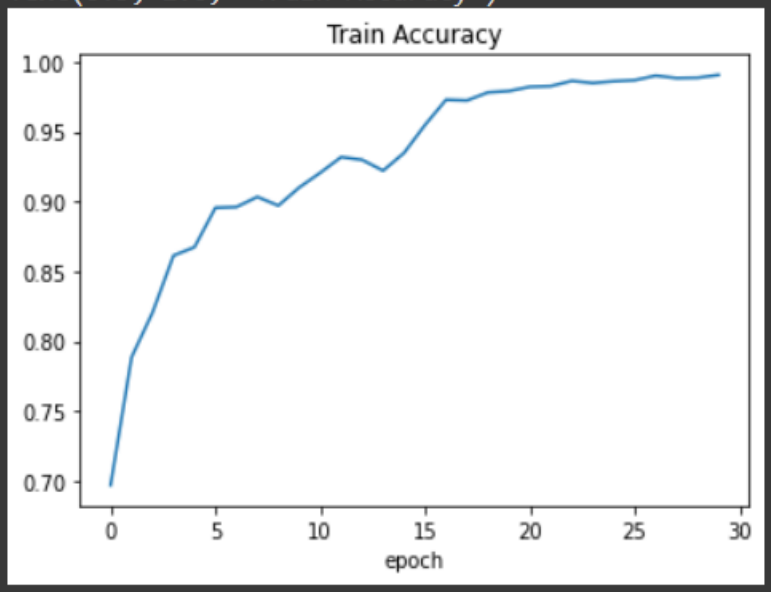


Fig 4.113: Train accuracy for Xception model

We can see the train accuracy starting at a very low value initially. As the number of epochs increase, the model learns and the train accuracy increases after some initial instability. We can see that the train accuracy is around 98-99% towards the final few epochs.

Train Accuracy for MobileNet model-

```
[ ]  plt.plot(training_stats['accuracy'])
     plt.xlabel('epoch')
     plt.title('Train Accuracy')

     Text(0.5, 1.0, 'Train Accuracy')
```

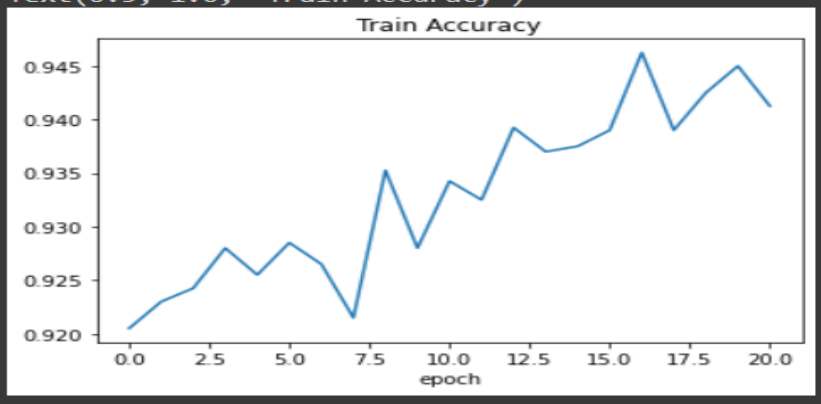


Fig 4.114: Train accuracy for MobileNet model

We can see the train accuracy starting at a very low value initially. As the number of epochs increases, the model learns and the train accuracy increases after some initial instability. We can see that the train accuracy is around 94% towards the final few epochs.

Validation loss: Validation loss should always be more than the training loss. Our aim is to make the validation loss as low as possible with each epoch.

Validation loss for Xception model-

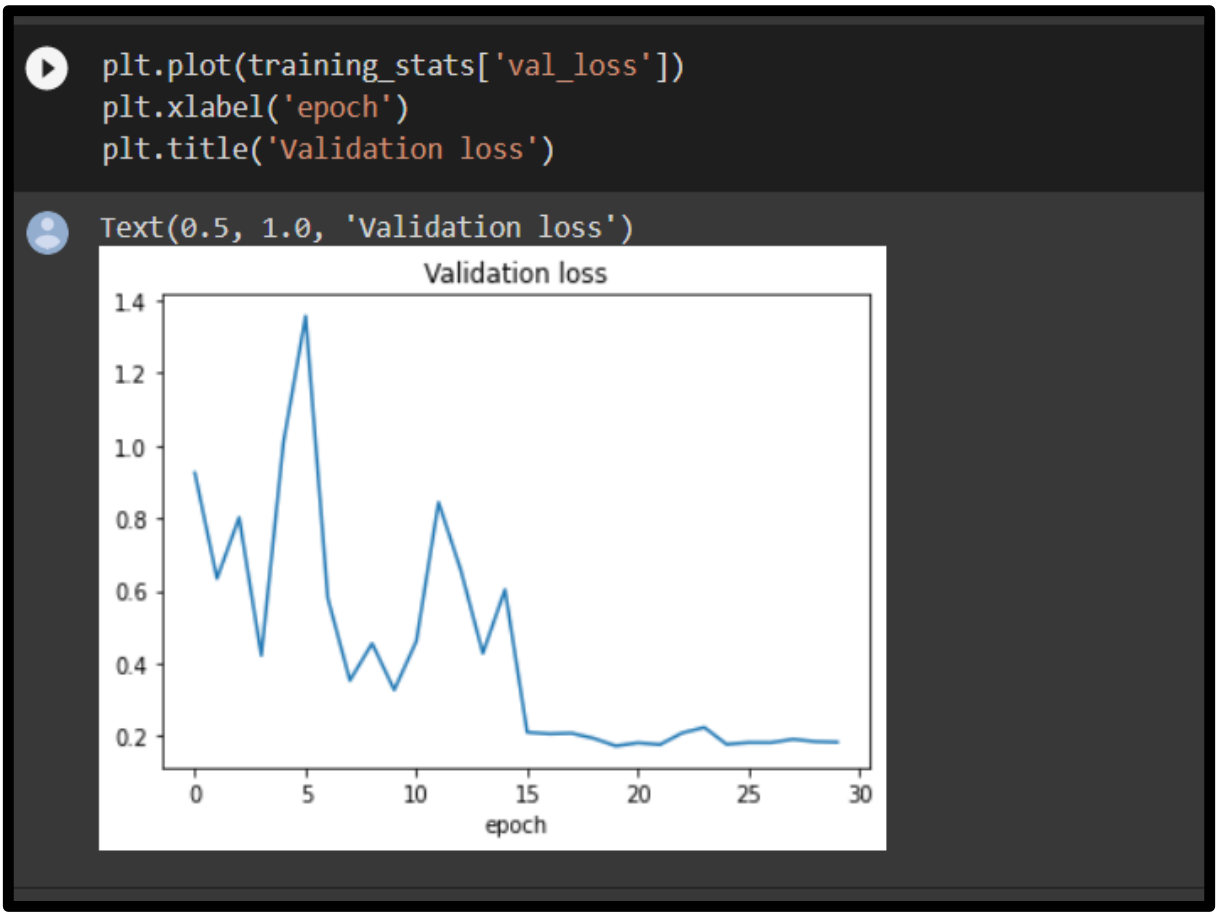


Fig 4.115: Validation loss for Xception model

We can see the validation loss being very unstable and exploding to a very high value initially. As the number of epochs increases, the model learns and the validation loss decreases after some initial instability. We can see that the train loss is around 0.04-0.05 towards the final few epochs.

Validation loss for MobileNet model-

```
plt.plot(training_stats['val_loss'])
plt.xlabel('epoch')
plt.title('Validation loss')
```

```
Text(0.5, 1.0, 'Validation loss')
```

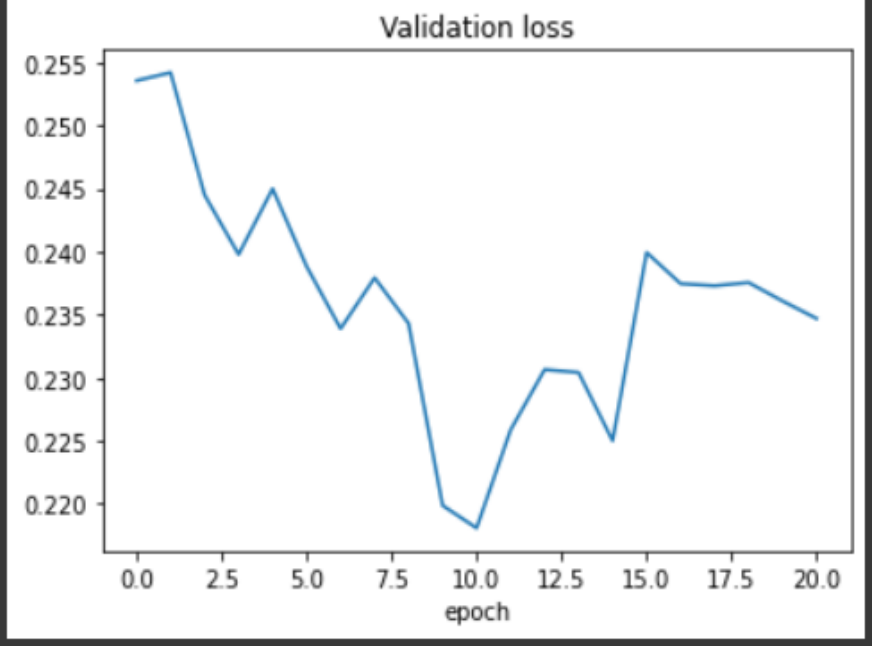


Fig 4.116: Validation loss for MobileNet model

We can see the validation loss being very unstable and exploding to a very high value initially. As the number of epochs increases, the model learns and the validation loss decreases after some initial instability. We can see that the train loss is around 0.235 towards the final few epochs.

Validation Accuracy: The validation set is used to evaluate the models' performance and we aim to get as high an accuracy as possible.

Validation accuracy for Xception model-

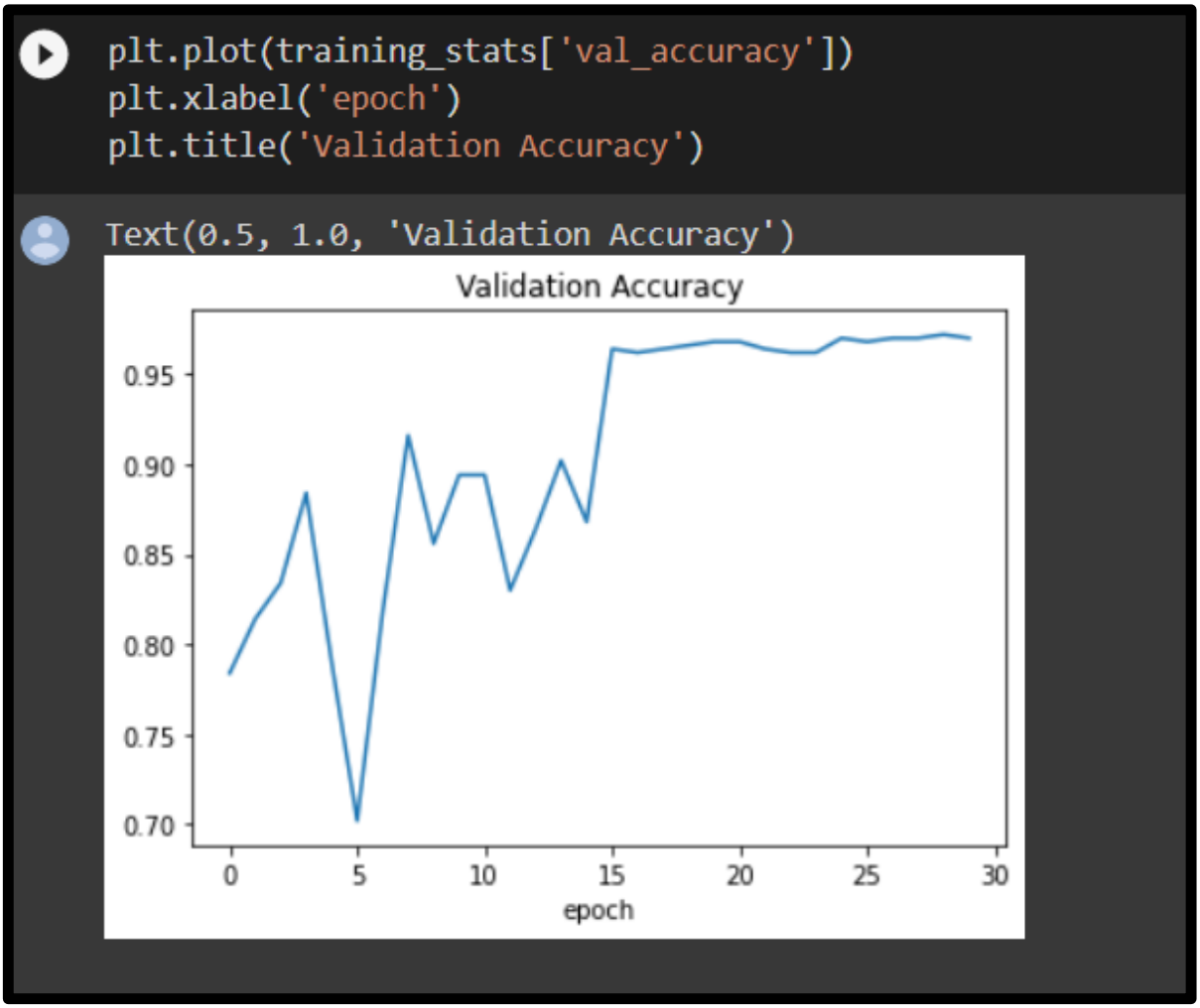


Fig 4.117: Validation accuracy for Xception model

We can see the validation accuracy starting at a very low value initially and acting very unstable. As the number of epochs increases, the model learns and the validation accuracy increases after some initial instability. We can see that the validation accuracy is around 98-99% towards the final few epochs.

Validation accuracy for MobileNet model-

```
[ ]  plt.plot(training_stats['val_accuracy'])
     plt.xlabel('epoch')
     plt.title('Validation Accuracy')

     Text(0.5, 1.0, 'Validation Accuracy')
```

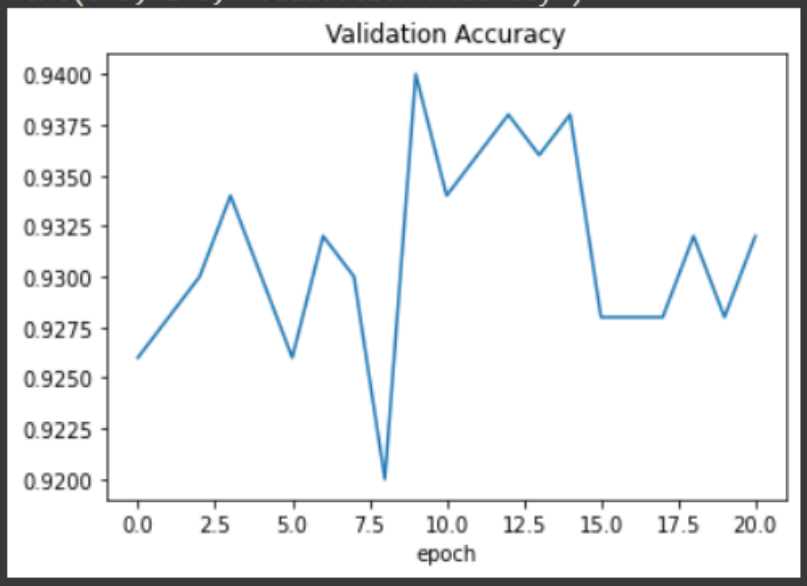


Fig 4.118: Validation accuracy for MobileNet model

We can see the validation accuracy starting at a very low value initially and acting very unstable. As the number of epochs increases, the model learns and the validation accuracy increases after some initial instability. We can see that the validation accuracy is around 93% towards the final few epochs.

Combined train and validation graphs:

We compare the train and validation graphs for both models. The blue line represents the train metric and the orange line represents the validation metric.

Xception model-

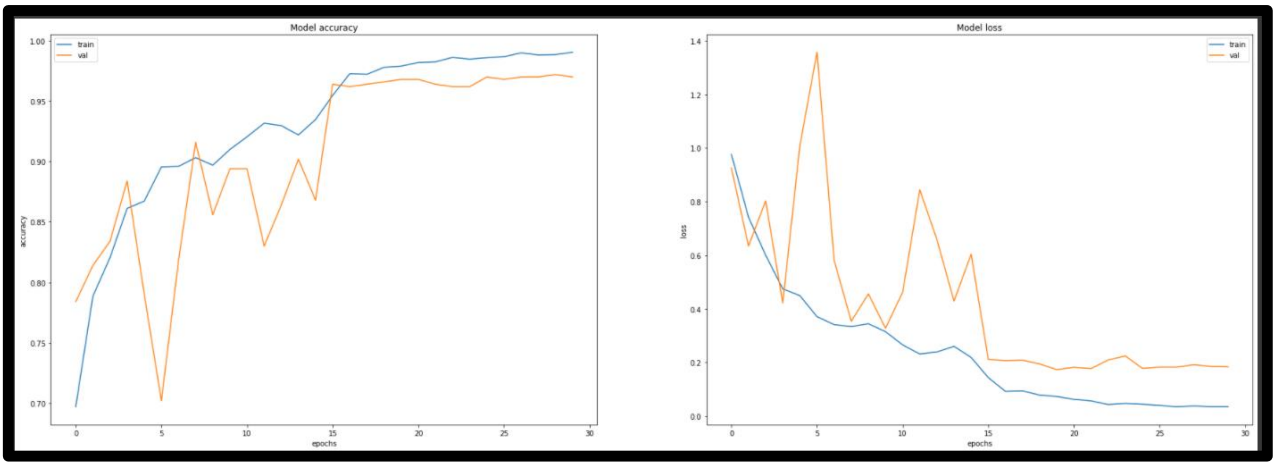


Fig 4.119: Combined train and validation graphs for Xception model

MobileNet model-

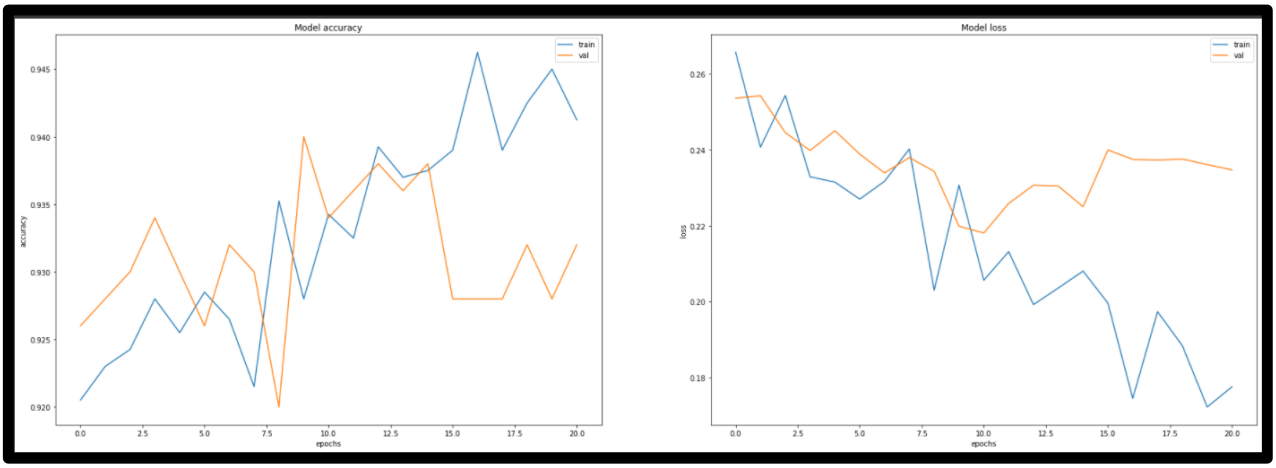


Fig 4.120: Combined train and validation graphs for MobileNet model

Comparing the testing accuracy of both models for each class:

Table 4.1: Comparison of testing accuracy of each class

Class	**Xception model accuracy**	**MobileNet model accuracy**
Dumpling	94%	91%
Lasagna	99%	92%
Macarons	95%	94%
Red Velvet Cake	96%	95%
Waffle	91%	88%

Chapter 5

Conclusion & future scope

5.1 Conclusion

The idea behind this project is to encourage users to adopt a healthy alternative to the conventional dining out system, which is especially observed among working-class adults and students, that arise because it is easier for people to order out than to decide on what to cook, however, such practices, in the long run, can not only lead to severe health complications but are also not budget-friendly.

Foodstagramming is so popular because people believe they are interesting and that, of course, that they get a lot of joy from beautiful, tasty food. It turns out that cooking, especially when done for others, comes with some positive psychological benefits. Posting aesthetic snapshots of gourmet dishes is also a part of our visual self-presentation: polished photos show us in a positive light to others.

Since food photos are huge on social media and are easy-to-produce and relevant to everyone, we built a deep learning model to identify a food image and provide the recipe for people to cook.

Our app will cater to the following problems:

1. Helps Controlling Temptations: If you can find healthy alternatives for junk food that have a smaller amount of calories, it will get easier for you to avoid chips.
2. Users can prepare Enjoyable meals: Instead of just settling for a regular cereal, you can treat yourself with an easy to make home-cooked meal.
3. Helps Tracking progress: Preparing meals at home helps you have control over the quality and proportion of meals you are consuming which in turn helps you achieve your fitness goals
4. Make the most of available ingredients without compromising on your creativity.

5. We have studied the performance of two models on the same dataset.
6. The MobileNet model will help when we want to deploy the application on mobile devices.

5.2 Future Scope

Some ideas for the future scope could include the following:

1. Recipes can be classified into various cuisines.
2. Login and Signup feature for personalized content and to save their favorite recipes.
3. Calorie Calculator for mindful consumption of calories that match the User's health goals.
4. Filter recipes based on meal time, for instance as, Breakfast, Lunch, Snacks, Drinks and Dinner and also classify them as Vegetarian or Non- Vegetarian.
5. Special Dish recommendations based on Festivities and Seasons.
6. Recipe segregation based on cooking time(20-30 mins) for fast paced individuals looking for a quick meal.
7. Offer more recipe classification classes.
8. Identify ingredients from Images
9. Accept Ingredient images and offer relevant recipes.

Chapter 6

References

[1] Janik Kluake- How to Build a Machine Learning API with Python and Flask
https://www.statworx.com/ch/blog/how-to-build-a-machine-learning-api-with-python-and-flask/

[2] Machine learning web application using React and Flask
https://school.geekwall.in/p/rJaqp8EqN/machine-learning-web-application-using-react-and-flask

[3] Aakanksha NS- Creating a web application powered by a fastai model
https://medium.com/usf-msds/creating-a-web-application-powered-by-a-fastai-model-d5ee560d5207

[4] Kevin Liao- Prototyping a Recommender System Step by Step Part 1: KNN Item-Based Collaborative Filtering
https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea

[5] Andrew Ng, Stanford University- Machine Learning
https://www.coursera.org/learn/machine-learning/home/welcome

[6] Saeed Aghabozorgi, Machine Learning using Python
https://www.coursera.org/learn/machine-learning-with-python/home/info

[7] Sayak Paul- Hyperparameter Optimization in Machine Learning Models
https://www.datacamp.com/community/tutorials/parameter-optimization-machine-learning-models

[8] Pier Paolo Ippolito- Hyperparameter Optimization
https://towardsdatascience.com/hyperparameters-optimization-526348bb8e2d

[9] GR Chandrashekhar- Regularization In Machine Learning – A Detailed Guide
https://analyticsindiamag.com/regularization-in-machine-learning-a-detailed-guide/

[10] Kiprono Elijah Koech- Softmax Activation Function — How It Actually Works
https://towardsdatascience.com/softmax-activation-function-how-it-actually-works-d292d335bd78

[11] RoseMarieNeilson-Diced-A-Recipe-Recommender
https://github.com/RoseMarieNeilson/Diced-A-Recipe-Recommender

[12] Rachel Zhiqing- Recipe-Reco
https://github.com/ZeeTsing/Recipe_reco/blob/master/README.md

[13] Madasamy M- Introduction to recommendation systems and How to design Recommendation system, that resembling the Amazon
https://madasamy.medium.com/introduction-to-recommendation-systems-and-how-to-design-recommendation-system-that-resembling-the-9ac167e30e95

[14] Xingyu Liu- Recipe-Recommendation
https://github.com/Flourishlove/Recipe-Recommendation/blob/master/Proposal.pdf

[15] Liev Garcia-Build A Recommendation Engine in One Day
https://blog.dataiku.com/build-a-recommendation-engine-in-one-day

[16] Google- Recommendation Systems
https://developers.google.com/machine-learning/recommendation

[17] Rachel Zhiqing Zheng-How to build personalized recommendation from scratch: recipes from Food.com
https://medium.com/analytics-vidhya/how-to-build-personalized-recommendation-from-scratch-recipes-from-food-com-c7da4507f98

[18] Killian Duay "Personalized web based application for movie recommendations" Thesis submitted for completion Bachelor of Science at Haaga-Helia University of Applied Sciences, 2019