CookBook Report

Project Team Albertosaurus:

Bhairaw Aryan

Jeremy Ng

Rushil Punchhi

Ting Xu

Submitted to:

Professor Mieszko Lis

CPEN 291

Abstract

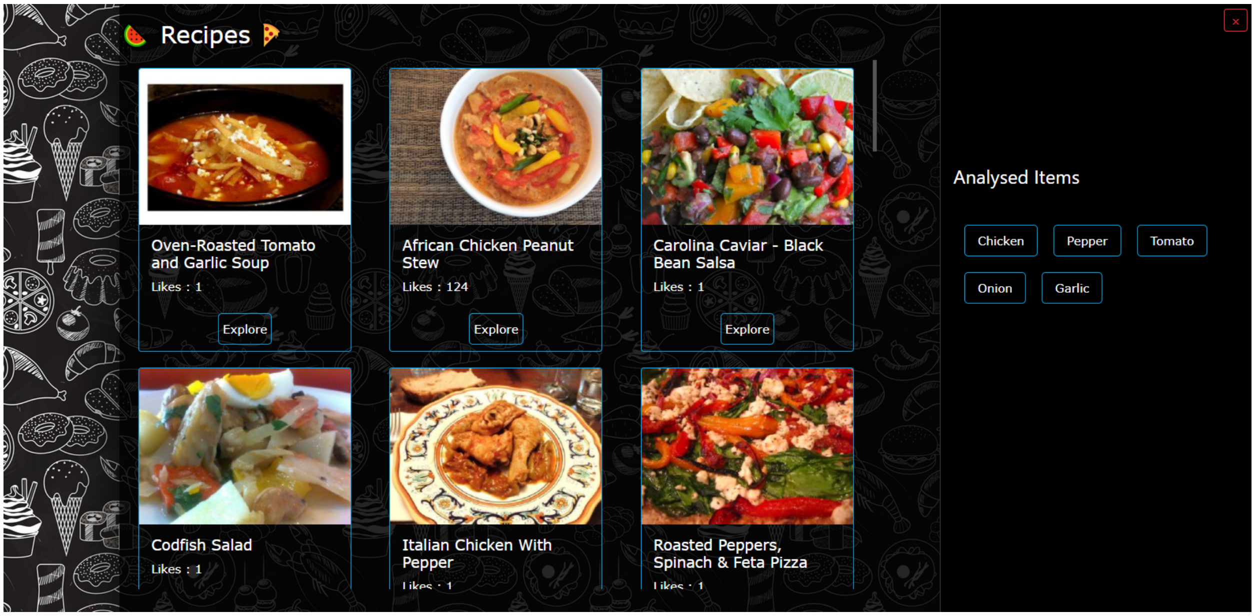
CookBook is a web and android app that takes in images of fruits and vegetables, uses a machine learning model to recognize them in the images, and then outputs various recipes that use them as ingredients. Users can either upload images from their computer, take an image straight from their camera, or type in the name of an ingredient. Each recipe's anatomy can further be fetched to display the ingredients, instructions, prep time, serving size, wine-pairing, cuisine type and a link to the original source of the recipe. The latter is achieved by using the Spoonacular client — an API that connects over 360,000 recipes as well as an open source recipe database for complex food ontology.

Walkthrough

When first connecting to CookBook via the web app, users will see a home screen with a 'Get Started!' button. The button pops an interface, with the option to either upload images of the ingredients through camera(if present)/regular file upload or manually type in the name of the ingredients. In the image below, we have typed in 'Pepper', 'Chicken' and uploaded images of garlic, onions, and tomatoes.

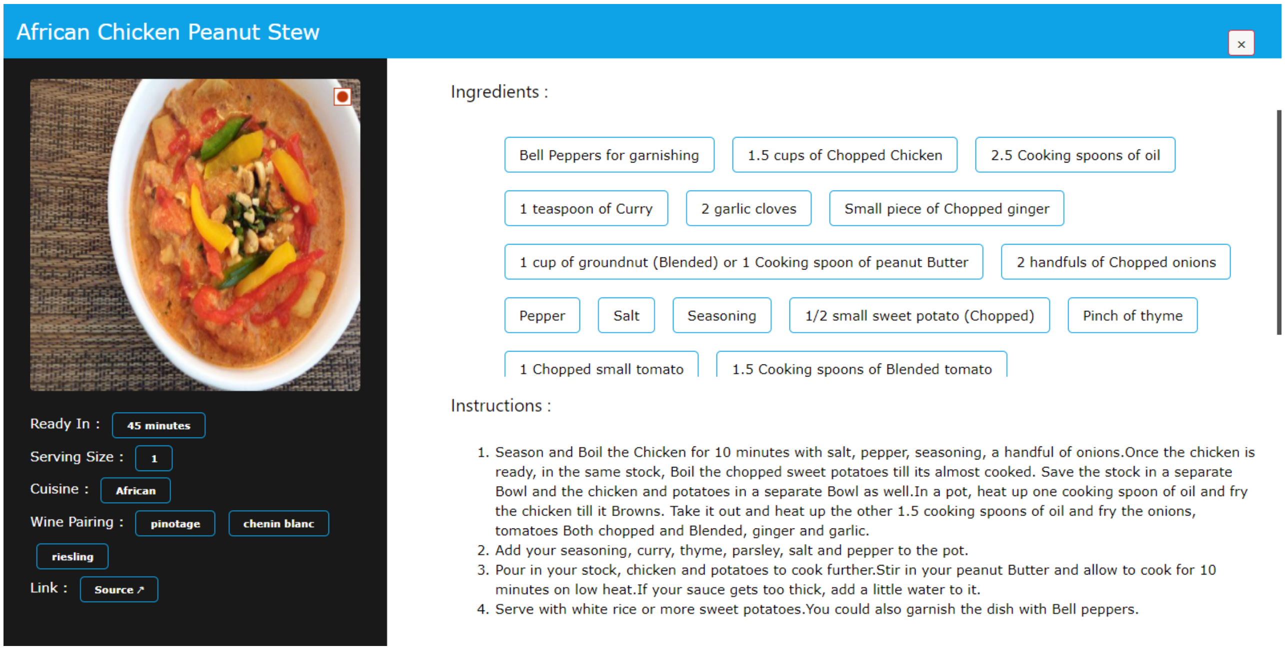
Main page of CookBook.

After pressing upload, CookBook will use its machine learning model to recognize the fruits or vegetables in the image, and then use the Spoonacular API to fetch suitable recipes. In the image below, the identified ingredients will show up on the right, and a list of recipes will be available for users to scroll through. To learn more about a specific recipe, users can press the 'Explore' button beneath the desired recipe.



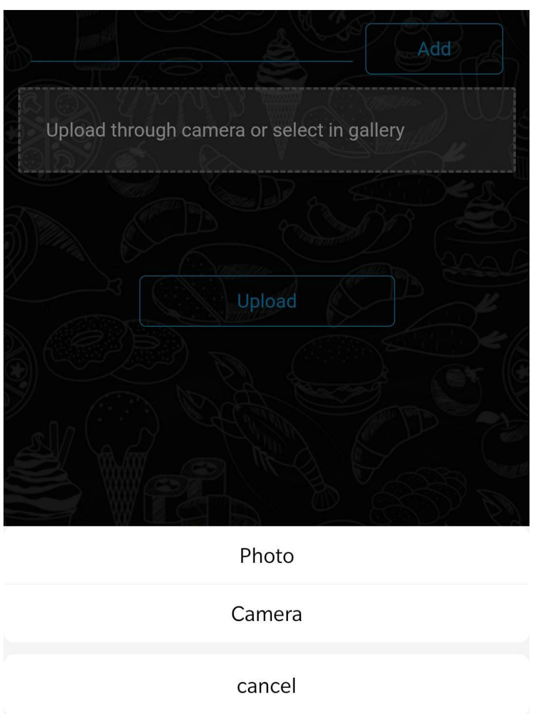
Recipes displayed after inputting chicken, pepper, garlic, onion, and tomato.

Upon clicking into a recipe, detailed instructions will be shown, along with an ingredients list, serving size, prep time, wine-pairing, cuisine, and the source of the recipe. Let's click on the most liked recipe, 'African Chicken Peanut Stew,' and we can see a modal opens up as shown in the image below.



Recipe page with ingredients, instructions, prep time, serving size, cuisine-type, wine-pairing and a link to source.

Users accessing CookBook via the iOS or Android app will be able to directly upload images from their camera or their photo library, as shown in the images below.



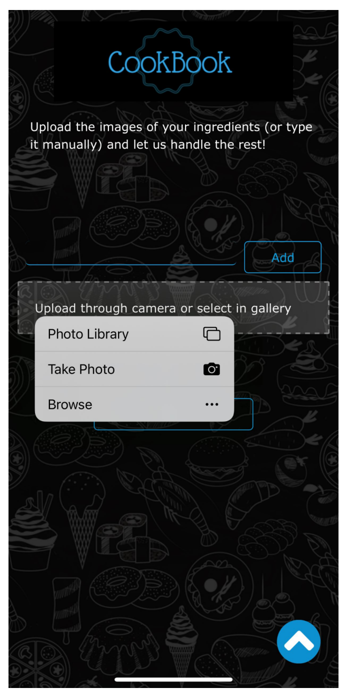


Image input on the Android app (left) and the iOS app (right).

Machine Learning Model

We used MobileNetV2 as the architecture for the machine learning model. MobileNets are characterized by their fast inference since they are based on a streamlined architecture, using depth-wise separable convolutions to build light weight deep neural networks [1]. This feature provides the basis for very efficient mobile-oriented models, allowing these models to serve as the base for many visual recognition tasks. Because we wanted a mobile version of our web app, we decided to use MobileNets. MobileNetV2 features an inverted residual structure with a bottleneck between layers, which also removes non-linearities in the narrow layers [2]. It also utilizes a novel framework known as SSDLite, which results in a prediction speed increase of up to 35% in comparison to previous versions [2].

We trained two separate implementations of a model on a dataset that consists of datasets from Kaggle (Fruits360, a dataset of images of fruits taken from all around it, and Fruit and Vegetable Image Recognition, a dataset include 36 species, each with around 100 images), and images from Bing Image Search and Google Image Search. After gathering all of the images, we further increased the number of images by applying data augmentations to them (random and rule cropping, flipping, rotation, adding noise using USM (unsharp masking), using histogram equalization, adjusting contrast and brightness). Then, each image class was split into training and testing datasets in a ratio of 7:3 respectively. Finally, the model with the highest accuracy was picked for the final version of CookBook. The diagram below shows a list of the supported ingredients.

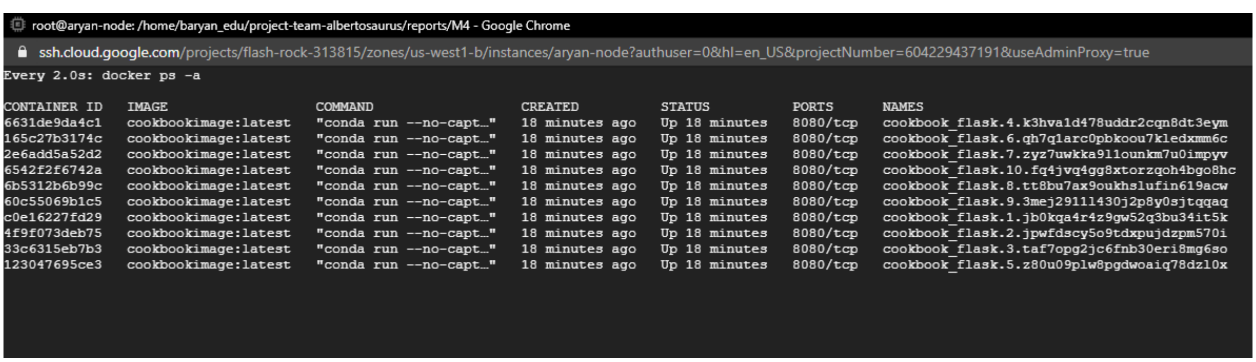
Apple	Clementine	Kaki	Nut Forest	Pomelo
Apricot	Cocos	Kiwi	Nut Pecan	Potato
Avocado	Corn	Kohlrabi	Onion	Quince
Banana	Corn Husk	Kumquats	Orange	Raddish
Beetroot	Cucumber	Lemon	Papaya	Rambutan
Blueberry	Dates	Lettuce	Passion Fruit	Raspberry
Cabbage	Eggplant	Limes	Pea	Redcurrant
Cactus fruit	Fig	Lychee	Peach	Salak
Cantaloupe	Garlic	Mandarine	Pear	Soy Beans
Capsicum	Ginger	Mango	Pepino	Spinach
Carambula	Granadilla	Mangostan	Pepper	Strawberry
Carrot	Grape	Maracuja	Physalis	Tamarillo
Cauliflower	Grapefruit	Melon Piel de Sapo	Physalis with Husk	Tangelo
Cherry	Guava	Mulberry	Pineapple	Tomato
Chestnut	Hazelnut	Napa Cabbage	Pitahaya Red	Turnip
Chilli Pepper	Huckleberry	Nectarine	Plum	Walnut
Choy	Jalepeno	Nectarine Flat	Pomegranate	Watermelon

Ingredients supported by the machine learning model.

Backend Flask

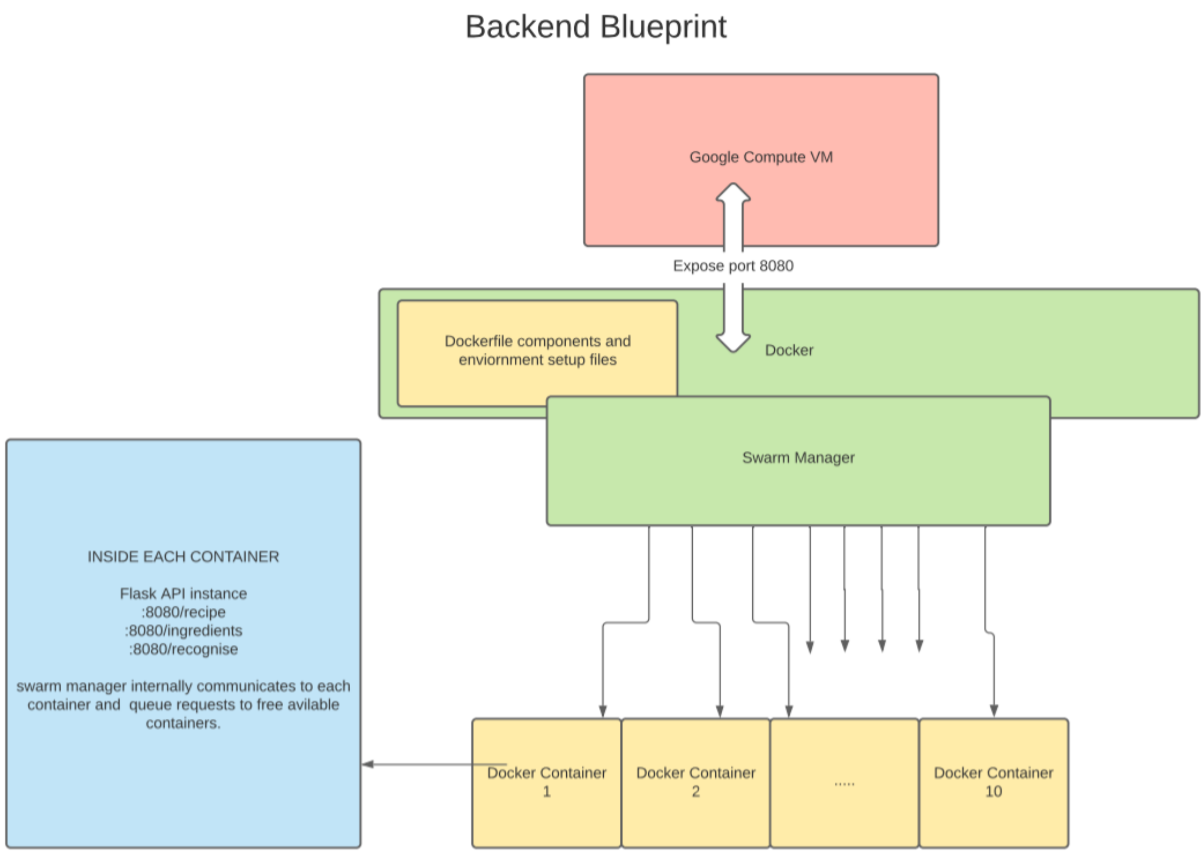
The backend has been implemented using Flask and is responsible for handling all the requests sent to it via the frontend. It listens on port 8080 and has 3 major functionalities — load the model.pht file and predict incoming images (:8080/recognise), create a spoonacular client instance to fetch lists of recipes using the ingredients (:8080/ingredients), and finally make a call to retrieve the complete info about a specific recipe (:8080/recipe). Since our ML model accepts one request at a time and predicts sequentially, multiple subsequent requests to the flask API slowed the system and in some cases even crashed or timed-out the server. Hence we came up with the idea to parallelize the task. We used docker and docker swarm to create multiple instances of the ML model with their own separate containers in the conda environment that could be scaled from 0 to even 1000 (needs more RAM on google compute engine as we keep on increasing dockers) at one click. This solved two major problems:

1. It can handle up to 10 simultaneous requests at once (current number of dockers as per optimal server configuration). This significantly reduces the time to respond to concurrent API calls by 90%.
2. Whenever a docker container crashes it doesn't affect Google's VM or other dockers. Instead, the manager node automatically transfers the workload to an available docker and in the meantime spins off a new docker with the same model instance to be used again. This reduces the downtime of our server by 99% in case of crashes or timeout error.



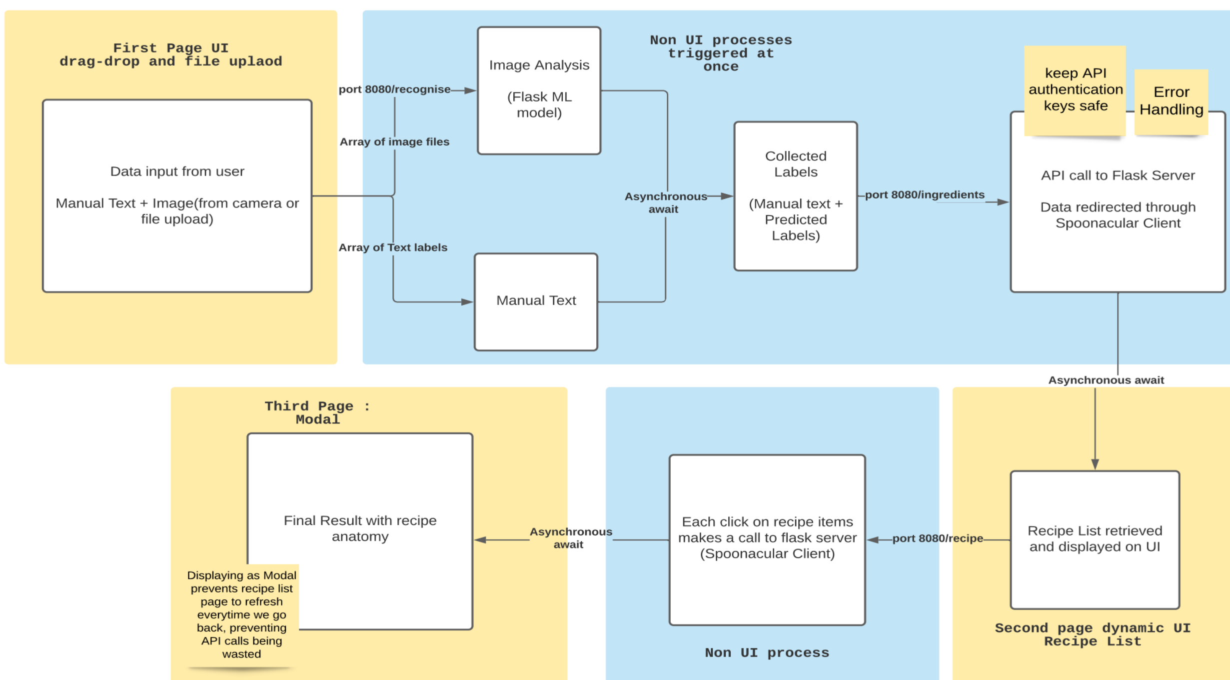
To build our app to be more robust and efficient, we thought of a way to increase the Spoonacular quota. As each authentication key is allowed for only 150 requests per day, each team member created a free account to total up our daily quota to 600 requests/day. Whenever one of the key's quota is exhausted, the server automatically switches to the next available API key and begins retrieving data, thereby increasing our available resources by 4 times. The advantages of using a Spoonacular client in the Flask instead of directly making requests through the frontend was:

- Keep our authentication key safe and secure
- Easily handle switching
- Keep everything organised.



Web App

The web app was created using ReactJS and various npm libraries. Animations and page transitions have been used to improve user experience. All the components are functional components and use hooks to change and render states and the items are dynamically loaded using states and props. The complete structure of the frontend and how images get processed into ingredients and further into recipes can be deduced from the blueprint [Fig 2]. Most tasks have been automated using multiple asynchronous calls and await functions. The image and manual feeds are segregated. The image files are uploaded to the server encoded in jsonified formdata through POST requests. The returned labels then combine with manual feed and a simultaneous request is made to flask for fetching recipes. The returned array of jsonified recipes are displayed dynamically as cards on the recipe list page each having a related explore button. Each 'explore' sends a POST request with an id to fetch info about the chosen recipe which is finally displayed as modal. The clever part about implementing modals is that they prevented refreshing of the recipe-list page whenever it was closed or open. As, fetching a list of 30 recipes at once was an expensive task regarding exhausting the spoonacular API limit, it helped reduce the cost by around 70%. The frontend has been deployed as a single instance on google cloud server and its default ip and port has been linked to a free domain name.



Android App

The Android app for CookBook can be installed by downloading CookBook.apk (found in M4 of the GitHub repository). In contrast to the web app, the Android app allows users to take photos directly from their camera and upload them. The app uses WebView to load the URL address of the web app and change the display to be more mobile-friendly. We first set the URL to the web address with webView.loadUrl (http://www.cookbookubc.ml), and then set up the WebViewClient and overrided the shouldOverrideUrlLoading method to ensure that the page would be loaded by the Android app instead of opening the web app on a browser. The next step was to set up the WebChromeClient, which intercepts HTML5 methods of calling to files and allows the Android operating system to handle the blocked files on its own. Then, the class PopupWindow was used to set up the popup layout and the popup events. To enable camera access, we had the app request for camera and photo album permissions. Additionally, to confirm whether an image was successfully obtained, a prompt message was implemented. If the fetching is successful, "Successful Access!' will be displayed. Otherwise, "Get failed" will be shown. Finally, the results were sent back to the web app in the form of an array of URIs.

iOS App

The iOS app for CookBook was successfully tested but cannot be installed by users due to restrictions on Apple developer accounts. The development process for the iOS app is similar to that of the Android app. We first set the fixed NSURLaddress as http://www.cookbookubc.ml, then built the NSURLRequest object and filled it in with the NSURL object. The third step was to initialize the WKWebView configuration and set up the layout. Then we called the loadRequest method of the WKWebView in the viewDidLoad method, which is the life cycle behavior declaration of ViewController, and filled it with the aforementioned NSURLRequest object.

Contributions

Jeremy created a machine learning model that was ultimately not picked for the final rendition of CookBook. He then focused on the secretarial side of things, such as the final report and the demo.

Aryan built the web app using ReactJS to analyze images with the embedded model in Flask and send and receive data using the Spoonacular API.

Ting created the final version of the machine learning model, embedded the trained model into Flask for the backend portion of the web app, completed both of the mobile apps, and helped explain her respective parts for the final report.

Rushil worked on an earlier implementation of the Android app and helped with the final report.

References

[1] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 2017. [Online]. Available: arXiv:1704.04861v1.

[2] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 2019. [Online]. Available: arXiv:1801.04381v4.