DeepMinds Senior Final Report

Matthew Munoz

Matthew Nieves

Tajwar Rahman

Date: 12/7/2024

Table of Contents

Abstract.....2

Introduction.....3

Literature Review.....4

The Data.....6
- Recipe Recommendation:.....6
- Food-based Image Recognition:.....7

Models.....11
- Recipe Recommendation:.....11
- Food-based Image Recognition:.....12

Results.....14
- Recipe Recommendation:.....14
- Food-based Image Recognition:.....17

Application.....23

Conclusion.....27

References.....28

Abstract:

The DeepMinds Recipe Application is a cutting-edge resource for home cooks of all levels. It leverages machine learning to personalize recipe recommendations and identify dishes from uploaded images. This report details the datasets, methodologies, and performance of the implemented models:

- **Recommendation System:** A collaborative filtering approach analyzes user reviews and keywords to recommend recipes similar to those users enjoy.

- **Image Recognition:** A deep neural network trained on the FoodX251 dataset classifies food images into various categories, enabling recipe suggestions based on the identified dish.

The application is built with modern frameworks like Next.js and FastAPI and utilizes AWS services for scalability and security. Key features include a dynamic search engine, user profiles, and social interaction functionalities. Although both models require further refinement, the application successfully bridges the gap between technology and culinary exploration. Plans involve enhancing model performance, promoting user engagement, and leveraging AWS for efficient model hosting.

Introduction:

DeepMinds recipe application is a cutting-edge resource for anyone who enjoys cooking, whether they are a culinary master or simply a beginner. The app leverages machine learning methodologies to implement a personalized recommendation system and an advanced food image recognition system to bring the culinary world to the modern age. Specifically, the two technologies are as described:

- **Collaborative filtering for Personalized Recommendations:** A collaborative filtering algorithm analyzes user preferences based on reviews to personalize recommendations to similar users. It also uses a similar framework to create recipe similarities based on keywords, allowing users to find recipes similar to recipes they already love. This recommender gives users a unique experience within the app, allowing them to create a culinary profile that fits their tastes.

- **Deep Neural Network for Image Recognition:** To further enhance the user experience, the app incorporates an advanced image recognition system powered by a deep neural network. This system allows users to take a photo of a dish simply, and the app will analyze the image to identify potential recipes for making at home. This innovative feature inspires users and simplifies the process of finding recipes for dishes they encounter in real life, such as restaurant meals or dishes shared by friends.

The application built using Amazon Web Services(AWS) creates a user-friendly interface that anyone can use. Its beautiful interface allows users to interact with powerful technologies and advance their culinary skills. The following report will delve deeply into the datasets used, methodologies, and performance of machine learning technologies. The goal is to demonstrate how integrating these cutting-edge technologies can significantly improve the user experience of a recipe application, making it a more engaging, personalized, and helpful tool for home cooks of all levels.

Literature Review:

The primary source, *Deep Learning in Food Category Recognition*, explores the use of deep learning in enhancing food recognition tasks such as identifying food types, ingredients, quality, and quantity. It reviews key aspects like datasets, data augmentation, feature extraction, and machine learning algorithms, emphasizing CNNs, transfer learning, and semi-supervised learning. This work aims to highlight current methodologies and inspire further advancements in food category recognition for research and industrial applications. The Conference on Computer Vision and Pattern Recognition iFood 2019 Fine-Grained Visual Categorization Challenge also introduced the FoodX251 dataset, offering a fine-grained benchmark for food image classification. While few studies specifically explore testing on FoodX251, existing benchmarks primarily rely on pre-trained models such as ResNet50, ResNet101, and DenseNet201. These sources provided valuable insights into designing and improving the model. For instance, the breakdown of densely connected architectures shown below in Figure 1 inspired the decision to construct a custom DenseNet and compare its performance with DenseNet201 pre-trained on ImageNet, as FoodX251's fine-grained nature presents challenges that benefit from deeper, well-connected networks.

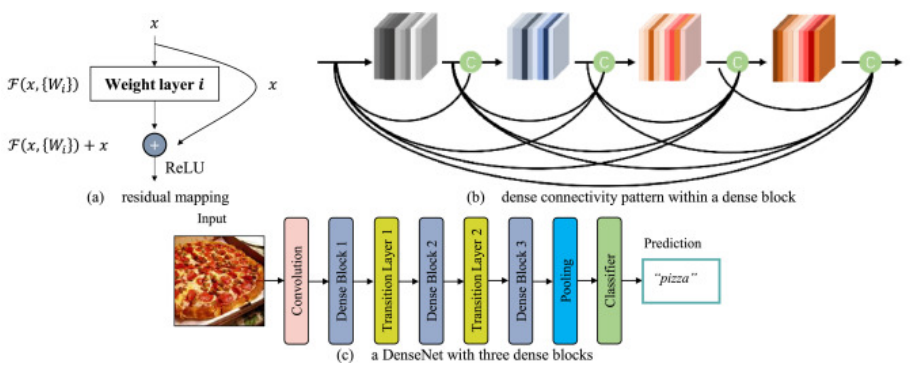


Figure 1. Structure of Dense CNN

DenseNets are designed for efficient feature reuse and improved gradient flow. In a DenseNet, dense blocks connect each layer to every preceding layer, ensuring that earlier features are directly accessible to later layers. Each layer produces a fixed number of new feature maps, determined by the growth rate. Between dense blocks are transition layers, which reduce spatial dimensions and the number of feature maps through 1x1 convolutions and pooling. The

reduction factor in transition layers determines how much the feature maps are compressed. As seen in Figure 1, DenseNets processes input images through an initial convolution, passes them through dense and transition layers, and then uses a final classifier to make predictions, such as identifying the "pizza" in the example.

The Data:

Recommender

The recommendation system's dataset was obtained from the Kaggle.*com—Recipe and Reviews* dataset. Specifically, the data includes two separate datasets: one for the recipes and the other for the reviews. The recipe dataset consists of 522,517 recipes with 12 observations, including name, RedipeId, and keywords used for the collaborative-filtering recommender. Figure 2 shows some key insights that were used for the implementation of the recommendation system. In particular, the high number of reviewers with a low number of reviews given would make it difficult to find similarities. Additionally, the correlation between user review count and high reviews would make it challenging to allow nuanced opinions with similar correlations to find matches. These insights allowed for consideration of where to filter the recipe and choose a better recommendation methodology for implementation. The keywords were also of particular interest to allow recipes of similar cuisines or ideas to be matched together for recommendation when a user wants a similar recipe.

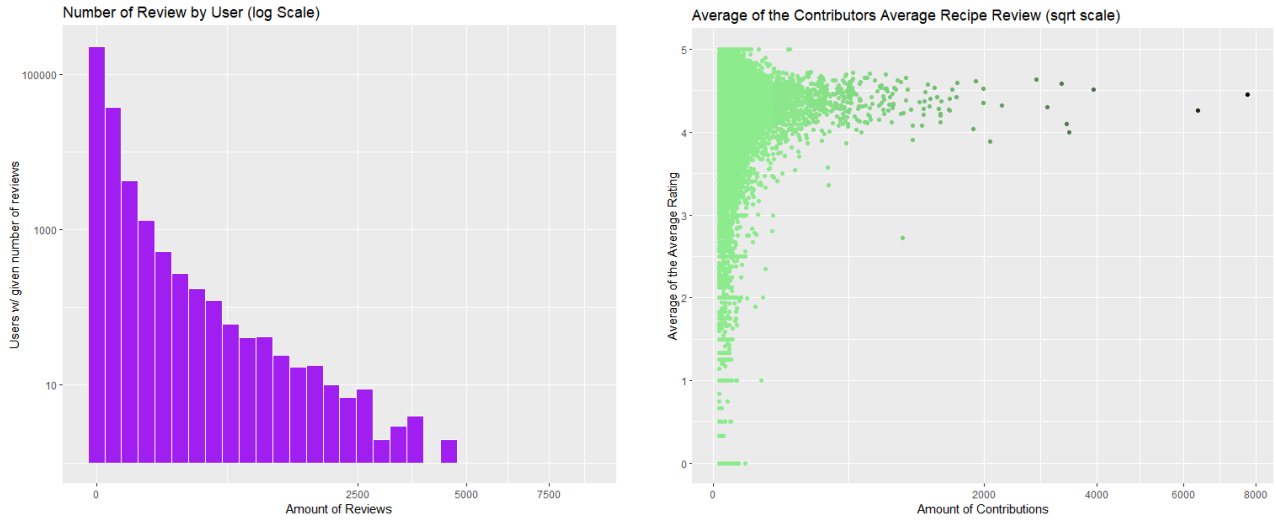


Figure 2. Recipe Information

The other dataset includes user interactions, which simply hold information on users and the ratings they gave to the various recipes. This dataset contains 1,401,982 from 271,907 users,

with reviews ranging from 1 to 5. Figure 3 shows some visualizations to help you understand the data better.

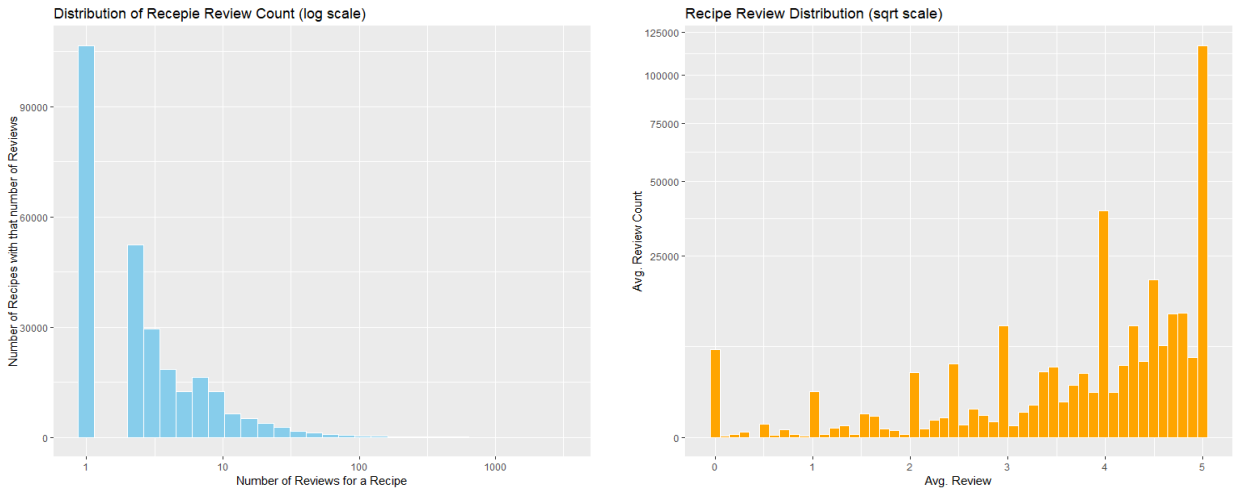


Figure 3. Review Information

The central insight gained from this data analysis is that the average recipe has fewer reviews. This insight confirms that users tend to give higher ratings, with most being averse to low ratings. Upon further analysis, a cutoff of recipes with at least 60 reviews and personalized recommendations to users who have given at least 10 reviews would allow a more accurate system. This cutoff also enables addressing the cold-start problem that usually plagues recommender systems, which is a system that does not have much information to allow a recommendation to be made to a user, leading to poor performance.

Image Classification

Dataset Overview

The FoodX251 dataset, which consists of 251 fine-grained food labels, was selected for the food image classification model. The dataset was originally part of the iFood 2019 Fine-Grained Visual Categorization challenge, offering a more complex and realistic challenge than cleaner

datasets. Unlike datasets with perfectly organized, unambiguous labels and no noise, FoodX251 contains a mix of fine-grained classes and potential ambiguities, making it more reflective of real-world scenarios.

Data Splits

The dataset came pre-split into distinct subsets:

- Training Set: 75% of data (118,475 images)
- Validation Set: 7.5% of data (11,994 images)
- Testing Set: 17.5% of the data (28,377 images)

Figure 4 shows a small visualization of some of the food classes in the dataset.

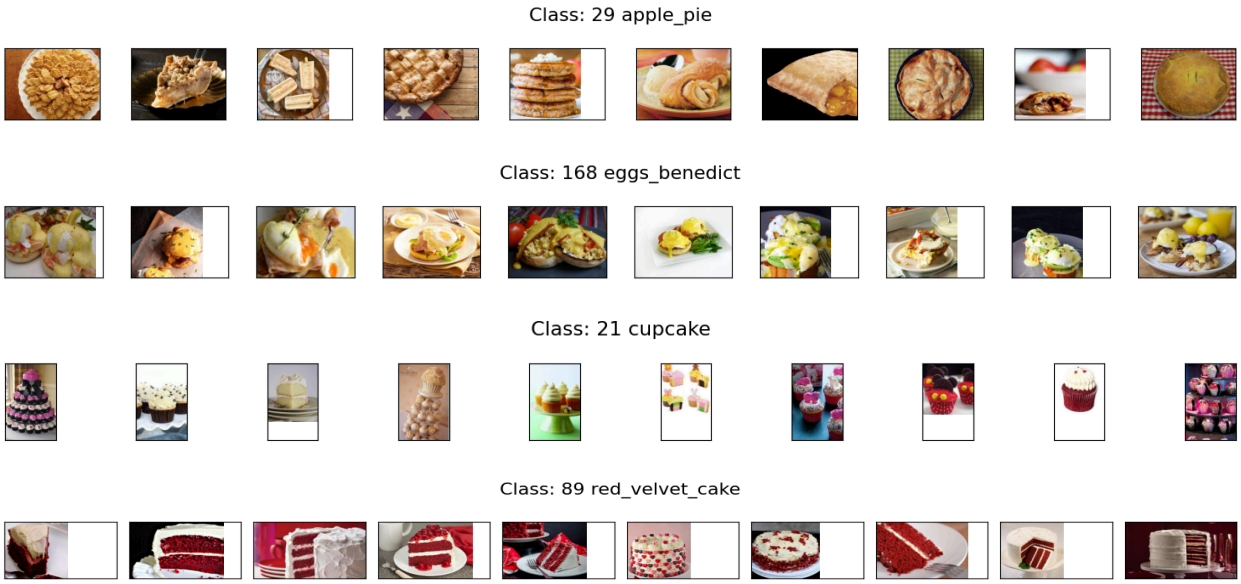


Figure 4. FoodX251 Data Visualization

Motivation for Using FoodX251

One of the primary reasons for choosing the FoodX251 dataset was the limited number of available studies and associated benchmarks. While datasets like Food-101 have been extensively studied, with numerous high-performing models and results published, FoodX251 has yet to be explored beyond the iFood 2019 challenge. The most notable resources found were

a research paper from the contest winners and a few Kaggle notebooks created by participants. This scarcity of existing work made the dataset an appealing challenge for the project, as it allowed for work in an area with fewer established benchmarks.

Image Preprocessing and Transformations

Before training any models, a series of image preprocessing and augmentation techniques to enhance the dataset's diversity and improve the model's generalization ability. These transformations were applied to the training set and included:

- Shear transformation by 20%
- Rotation up to 30º
- Width shift by 20%
- Height shift by 20%
- Zooming in or out by 20%
- Horizontal flipping

These augmentations simulate real-world variations in food images, such as different angles, positions, and scales. This preprocessing step ensures that the model learns robust features that are less sensitive to input data variations.

Challenges of Class Imbalance

One significant issue with the dataset is the imbalance in class representation. Across the 251 food labels, the number of images per class varies greatly, ranging from as few as 20 images to over 650. This imbalance can lead to biased model performance, where the classifier disproportionately favors overrepresented classes, neglecting minority classes entirely. This behavior is detrimental to multi-class classification tasks, particularly in real-world scenarios. Class weights were implemented using the sklearn library to balance class representation during training to address this issue. This approach provides a simpler alternative to more complex techniques like oversampling or synthetic data generation. Figure 5 below illustrates the distribution of images per class, highlighting the need for class weights to mitigate the effects of class imbalance.

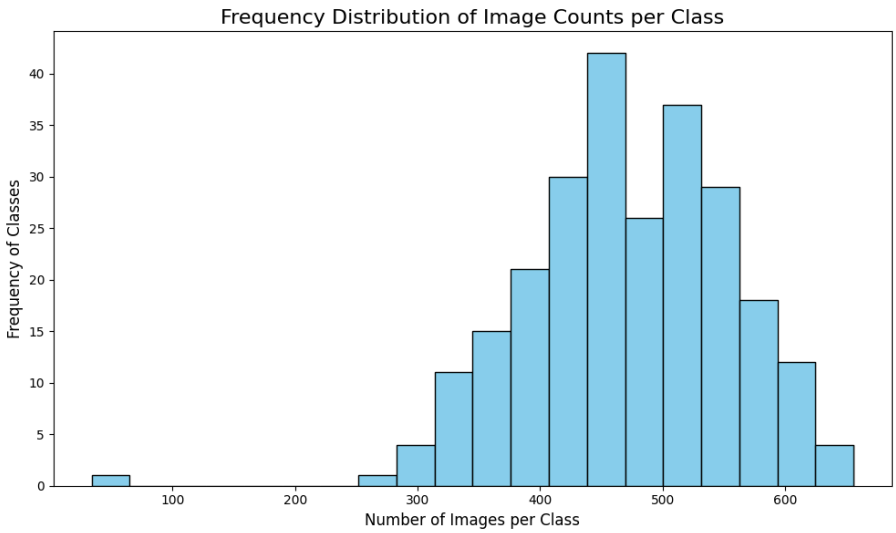


Figure 5. Food Class Representation

Models:

Recommender

Several recommender methodologies were considered, including cosine similarity, Kendall's, tau, and Pearson correlation coefficients. However, the simplicity and power of cosine similarity allowed the limited information on users and recipes to be used to create the recommender. It's a similarity measure between two vectors of an inner product space representing the cosine of the angle between them. In the context of recommendation systems, these vectors often represent user preferences or item characteristics. Cosine similarity works as follows:

- **Vector Representation:** Each user is represented as a vector, where each dimension corresponds to a recipe, and each value in that dimension corresponds to the reviews given to the recipes. Similarly, each recipe is represented as a vector, and the dimension values are the keywords that describe that recipe.

- **Similarity Calculations:** The cosine similarity is computed by using the equation below.

$$CS(A,\ B)\ = \frac{A \bullet B}{||A||\ ||B||}$$

- **Recommendation Generation:** The computation is performed on every user for every other user and every recipe for every other recipe. This information is then placed in a matrix for matrix factorization that includes all the information, giving every computation a value between -1 and 1, where higher values indicate highly similar instances. More specifically, vectors that point in similar directions are considered more similar without regard to the values given. This allows the system to find nuances in the data and not simply return recommendations for users with exact data but for users who exhibit similar trends.

A visual representation of cosine similarity is shown below.

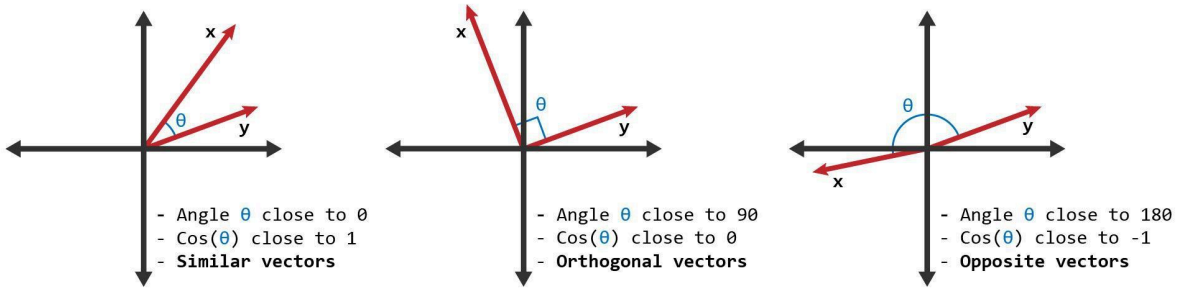


Figure 6. Depiction of Cosine Similarity[3]

Cosine similarity is a valuable technique for building recommendation systems, especially when dealing with sparse data and high-dimensional spaces. Effectively measuring the similarity between users and items enables personalized recommendations that cater to individual preferences.

Image Classification

Introduction to Pre-Trained Models for Food Image Classification

For the image classification model on the FoodX251 dataset, consideration of the well-known models recognized for their strong performance on food datasets, such as ResNet50, DenseNet201, MobileNetV2, and VGG19, was first performed. These models have demonstrated exceptional results on datasets like Food-101, which contain broad categories with minimal fine-grained distinctions. However, FoodX251 presented a more challenging problem due to its fine-grained nature, featuring subtle similarities between many classes. This distinction made it evident that the approach would require more robust feature extraction and handling of noisy data compared to datasets like Food-101, where such complications are minimal.

Initial Attempts with MobileNetV2 and Custom Models

Our initial approach involved using MobileNetV2, which is known for its lightweight architecture and computational efficiency. To evaluate performance, an attempt to design a small sequential model composed of convolutional, max-pooling, and dropout layers. Unfortunately, this custom model struggled significantly on the noisy, fine-grained dataset, achieving less than

1% accuracy during early tests. This emphasized the complexities of handling a large dataset like FoodX251 and the need for a deeper model. Using MobileNetV2, preliminary tests were conducted to observe how a pre-trained model performed on the dataset. Using a simple setup with the Adam optimizer, categorical cross-entropy loss, and basic metrics tracking across 20 epochs, the model achieved around 30% accuracy. This test served as a valuable benchmark for subsequent models.

Addressing Computational Limitations

Initially, training was conducted on a MacBook Pro with an Intel i9 CPU, which imposed significant computational limitations. To address this, Kaggle's free 30hrs/wk GPU quota (P100 GPU) and purchased Google Colab Pro for additional GPU access, including T4 and L4 GPUs, were utilized. These resources provided sufficient computing power to train larger models and conduct tests without session crashes or time constraints.

Choosing DenseNet201 as the Baseline Model

After considering ResNet50 and DenseNet201 for the baseline, it was determined that DenseNet201 was better suited for our needs due to its ability to encourage feature reuse while having fewer parameters than ResNet50, resulting in lower computational costs. DenseNet201 also showed faster training times per epoch (28-30 minutes) than ResNet50. Implementation of callbacks to log metrics such as accuracy, loss, precision, and recall, ensuring all training data was preserved for further analysis and visualization, was also performed. These callbacks were particularly useful on Colab, where session interruptions could occur.

CreatingtheOwn DenseNet

DeepMinds decided to create a DenseNet for the model based on the structure shown in Figure 1 and the specific makeup discussed earlier. For the DenseNet, a growth rate of 32, a reduction of 0.5, and four dense blocks with three transition layers. Each dense block consists of batch normalization, ReLu activation, and convolution repetitions. The transition layers contain normalization, 1x1 convolution, and average pooling for downsampling.

Results:

Recommender

This section evaluates the performance of the implemented recommender system, which uses both the user-user and the item-item collaborative filtering techniques.

Unfortunately, appropriate testing for recommender systems requires measuring click-through rates, conversion rates, and user satisfaction, among other post-app deployment methods. So, to accomplish testing for the user-user collaborative filtering, a pseudo-user was created, imitating a real live user to simulate whether user similarities are being identified accurately. For the item-item collaborative filtering, several recipes were looked into after similarity computation to see if they have similar attributes.

Recipe	**Rating**
10457	5
26499	5
27733	5
16831	5
37252	5
21812	5
10554	5
12027	5
22815	2
100688	5

Recipe	**Recipe**
10457	5
46877	5
60085	5
57549	5
2137	5
63869	5
27141	5
48582	5
110139	5
41707	5

Figure 7. User 1533 Reviews

The user-user similarity performed well, with the model accurately determining similar users and recommending recipes between them. Initially, identification of a user with many recipe reviews identified by the number 1533, which has 21 reviews. A new pseudo-user was then created to mimic 10 of 1533's preferences, although with a lower score than what 1533 rated each recipe. The cousin similarity was then computed on all users, with the recommender identifying users 1 and 1533 as having a similarity of 68%, the highest for these two users. It did this knowing one of the users, user no. 1, is more critical of the recipe reviews. A depiction of the test can be seen in Figure 7 and Figure 8.

Recipe	**Rating**
10457	4
26499	4
27733	4
16831	4
37252	4
21812	4
10554	4
12027	4
22815	1
100688	4

Recipe
2137
14953
27141
41707
48582
57549
60085
63869
110139
132916

Figure 8. User 1 Reviews(left), Recommendations(right)

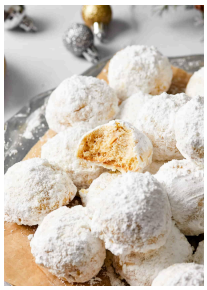
The item-item recipe similarity was tested by examining several recipes and identifying similar recipes to be returned. Two of these recipes can be seen in Figures 9 and 10. The two shown are

the recipes “Biscotti di Prato” and “Prawn Mexican” and their identified similar recipes. For the former, the system identified that the recipe has the keywords Weeknight, <4 Hours, Europen, Oven, and Cookie & Brownie, with the returned recipes holding all of these keywords and several others. The latter of the two recipes performed similarly, with the keyword being held by

Biscotti Di Prato

Chocolate and Vanilla Biscotti

Gourmet Cappuccino Biscotti



Kourabiedes (Greek Cookies)

Italian Wine Biscuits

Spritzgeback (Spritz Cookies)

Figure 9. Biscotti Di Prato’s five most similar recipes

Prawn Mexicana

Barbecued Shrimp

Firecracker Shrimp

Boiled Shrimp With Spicy Butter

Barbecued Cajun Shrimp

Rejennos Jaibas

Figure 10. Prawn Mexicana's five most similar recipes

Overall, the recommendation system performed well and is a baseline for user satisfaction and retirement. It allows users to identify pertinent recipes and personalize them to their liking, keeping them satisfied and returning for a great user experience.

Image Classification

Performance of Pre-trained DenseNet201

DenseNet201 was trained for 40 epochs using the Adam optimizer with a learning rate of 0.0001, using a learning rate scheduler, checkpoints, early stopping, and data logging. The training was conducted on an L4 GPU via Google Colab, with categorical cross-entropy loss. The model demonstrated steady improvement after the first five epochs, with the most significant increase in accuracy occurring between epochs 5–10. Metrics stabilized in the latter epochs while precision and recall improved. The best-performing model, restored from epoch 20 (lowest validation

loss), achieved an accuracy of 63.56%, loss of 1.506, precision of 0.7658, and recall of 0.55, as shown below in Figure 11.

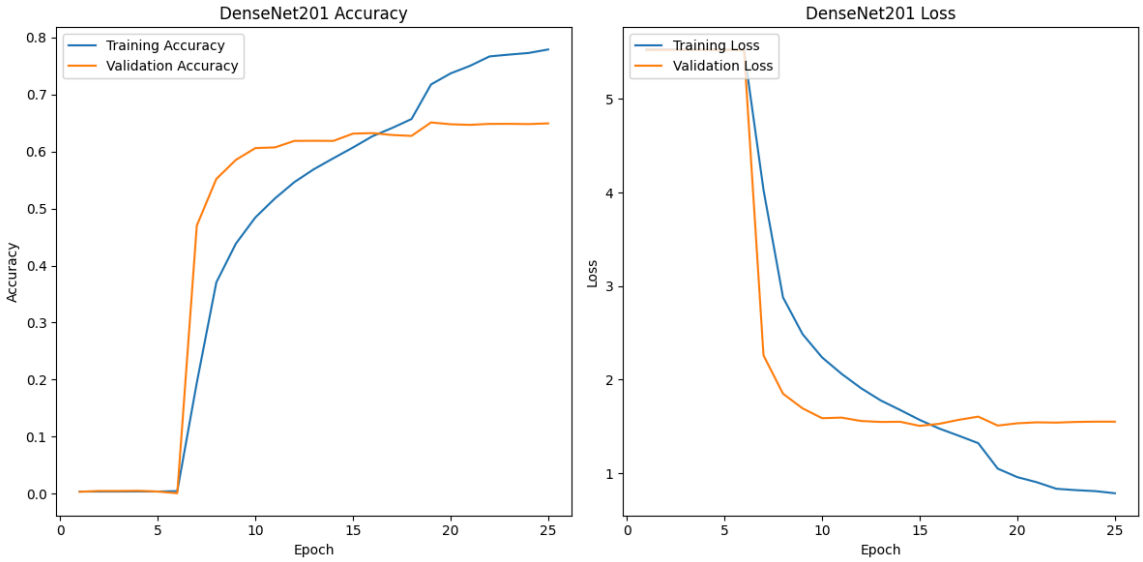


Figure 11. DenseNet 201 Training Results

Performance of the Custom DenseNet

Our custom DenseNet model was trained for 45 epochs using the same optimizer, loss function, and callbacks. The loss and accuracy curves indicate a steady learning process, with performance peaking around epoch 20 and fluctuating slightly before recovering and improving near epoch 45. While the model learned effectively, adjustments to the patience parameter of the early stopping callback could have allowed further improvements over additional epochs. Figure 12 below shows the following metrics: the best model was saved at epoch 20 with an accuracy of 55.18%, loss of 1.99, precision of 0.6848, and recall of 0.4818.

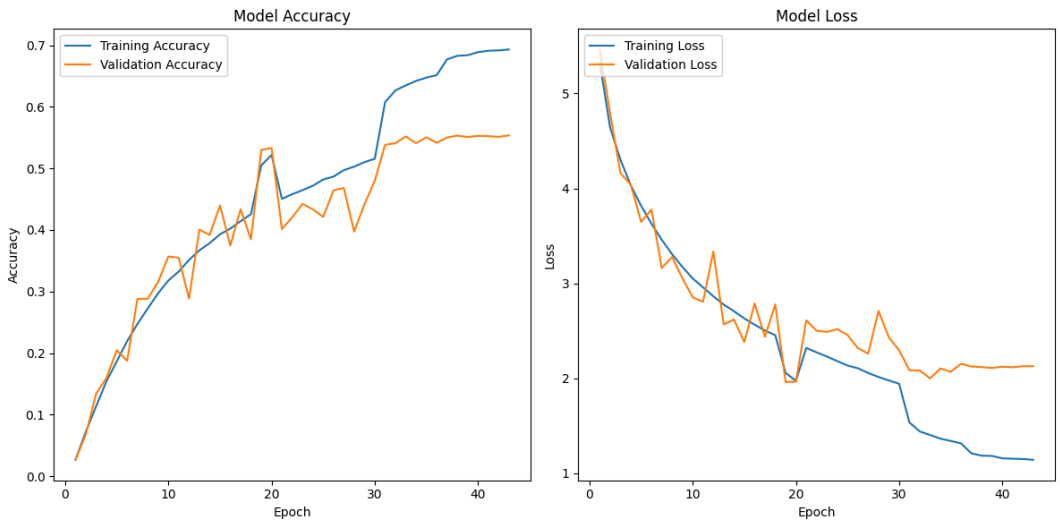


Figure 12. Custom DenseNet Results

Comparison of DenseNet201 and Custom DenseNet

A comparison of the two models' metrics (Figure 13) reveals that DenseNet201 consistently outperformed the custom model in accuracy, precision, and recall. This outcome aligns with expectations, given that DenseNet201 is a larger, pre-trained model fine-tuned on the ImageNet dataset. Despite this, the custom model showed promising results, suggesting its potential as a lightweight alternative.

	Accuracy	Loss	Precision	Recall
Our Model	0.5518	1.99	0.6848	0.4818
Densenet 201	0.6356	1.506	0.7658	0.55

Figure 13. Comparison of Metrics

Analysis oftheCustom DenseNet's Metrics

Below, in Figures 14 and 15, the top 10 and bottom 10 performing classes are shown by F1 Score. F1 Score was selected as the evaluation metric due to its ability to balance precision and recall, making it particularly effective for capturing both false positives and false negatives.

The results revealed no clear correlation between the number of training samples and model performance. While some well-represented classes performed well, others, like churros with over 600 training images, ranked among the lowest-performing categories. Conversely, certain underrepresented classes achieved high F1-scores, suggesting that factors beyond sample size—such as the uniqueness of class features or overlap with other categories—play a significant role in influencing the model's performance.

Class Number	Food Name	precision	recall	f1-score	support	Num Images
150	bruschetta	0.900000	0.775862	0.833333	58.0	552
195	oyster	0.851852	0.793103	0.821429	58.0	445
239	rock_cake	0.787234	0.840909	0.813187	44.0	179
169	bread_pudding	0.767857	0.860000	0.811321	50.0	586
163	filet_mignon	0.916667	0.721311	0.807339	61.0	489
1	beignet	0.827586	0.786885	0.806723	61.0	531
191	mussel	0.853659	0.760870	0.804598	46.0	496
77	coq_au_vin	0.833333	0.769231	0.800000	65.0	460
8	dumpling	0.880952	0.725490	0.795699	51.0	585
166	tuna_tartare	0.803571	0.762712	0.782609	59.0	421

Figure 14. Top 10 Class Performance by F1-Score

Class Number	Food Name	precision	recall	f1-score	support	Num Images
155	churro	0.050000	0.090909	0.064516	11.0	629
54	farfalle	0.192308	0.100000	0.131579	50.0	433
161	coconut_cake	0.140351	0.222222	0.172043	36.0	559
71	sukiyaki	0.833333	0.102041	0.181818	49.0	405
53	falafel	0.205128	0.190476	0.197531	42.0	487
86	pancake	0.228571	0.190476	0.207792	42.0	359
144	flan	0.280000	0.166667	0.208955	42.0	454
234	pirogi	0.186441	0.275000	0.222222	40.0	374
109	strawberry_shortcake	0.209302	0.243243	0.225000	37.0	338
128	schnitzel	0.206897	0.260870	0.230769	23.0	369

Figure 15. Bottom 10 Class Performance by F1-Score

Analysis of Common Misclassifications

It was observed that the most common misclassifications are between visually similar foods. Some visually similar foods that were misclassified include: barbecued spare ribs and baby back ribs, linguine and fettuccine, and spring rolls and egg rolls. There was also ambiguous labeling; beef tartare was misclassified as steak tartare 14 times, though these foods are the same and should not be under different labels. These foods that were misclassified are visually similar. At the same time, their differences lie in subtle features like the inside contents or the outside texture and thickness of the wrapper for egg rolls and spring rolls, while linguine and fettuccine are both nearly flat pasta but have different widths.

Figure 16. Sample Misclassifications

Effects of Class Weights on Performance

The custom DenseNetwas was trained without applying class weights to evaluate their impact. The results indicated a noticeable drop in performance, particularly in recall and accuracy, but minimal improvement in precision. Incorporating class weights improved balance across all metrics, demonstrating their effectiveness in addressing imbalances within the dataset.

Accuracy	Loss	Precision	Recall
52.18	2.003	0.7070	0.4074

Table 1. Model Results Without Class Weights

A side-by-side comparison of performance in terms of F1-Score by the number of images a class is represented by can also be seen. In Figure 17, shows the F1-Score by the number of images in a class when class weights are implemented and when it isn't. The plots show similar results,

where both performances are close with and without weights, but performance is slightly more balanced when class weights are implemented. Looking at both ends of the plots, it can be determined that the class with the most images performed worse with weights than without. However, classes with the least images performed better, so although performance was balanced, the weights negatively affected some classes.

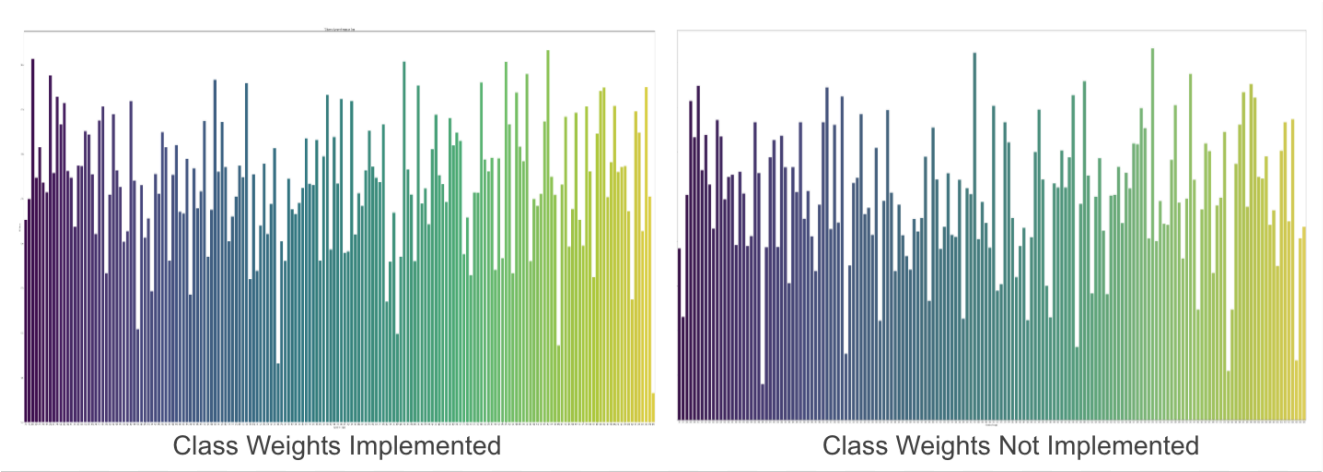


Figure 17. F1-Score Distribution by Number of Images in a Class

Future Plans

To improve the model's performance, the dataset will be refined by resolving ambiguous labels, such as merging or removing overlapping classes like beef tartare and steak tartare. The aim is also to address noisy data by cleaning low-quality images that hinder learning. Additionally, the model will be enhanced by incorporating kernel regularizers to improve generalization. Finally, the model will explore advanced class balancing techniques, such as oversampling, to address dataset imbalances effectively.

The image recognition model delivers accurate results for several classes and meets the primary objective. Additionally, the model has been integrated into the application's search bar, enabling an effective image search option. While satisfied with the model's current functionality, it is acknowledged that there is significant room for improvement, providing opportunities for future refinement and enhancement.

Application:

Frameworks and Hosting

The DeepMinds Recipe Application leverages a modern and robust technological stack to deliver an optimized user experience. The front end, built using Next.js, incorporates dynamic routing and server-side rendering (SSR) to enhance performance and ensure smooth interactions. SSR ensures that pages are pre-rendered on the server, providing faster load times and better SEO. Vercel, the hosting platform, allows seamless integration with the Next.js framework, ensuring scalability, global content delivery, and reduced latency.

The backend is developed using FastAPI, a high-performance web framework ideal for building APIs. FastAPI's asynchronous capabilities and built-in support for data validation make it efficient and developer-friendly. This backend is deployed on Heroku, which provides a reliable platform for managing API services. Heroku's features, such as continuous deployment from GitHub and built-in monitoring, simplified the development lifecycle and ensured stability during deployment.

AWS Integration

The application integrates seamlessly with AWS services, ensuring scalability and security. The integration includes:

- AWS DynamoDB: A NoSQL database that efficiently stores recipes, user profiles, and interactions. Its flexibility allows quick querying and updates, even with complex data models.
- AWS Cognito: A comprehensive service for user authentication and profile management. Cognito enables secure sign-ins, social media login integrations, and token-based authentication, ensuring user data is protected.

These services allow the application to handle large-scale operations, such as processing multiple recipe uploads, user interactions, and real-time personalized data retrieval.

Integrated Models

Recommendation System:

The recommendation system is embedded into multiple pages, including the '/popular' section. It uses collaborative filtering to analyze user behavior and generate personalized suggestions. The system processes user ratings and activity data to identify patterns, providing a tailored experience for each user.

The algorithm leverages similarity metrics such as cosine similarity to identify user trends and shared preferences. Recipes are dynamically ranked based on their popularity and relevance to individual users.

Image Recognition:

A state-of-the-art deep learning model underpins the image recognition feature. This model is integrated into the search engine, allowing users to upload images of dishes. The backend processes these images using convolutional neural networks (CNNs) trained on diverse datasets, enabling accurate classification and recipe matching. This feature particularly benefits users encountering new dishes, as it bridges the gap between visual observation and recipe discovery.

Key Features

Dynamic Search Engine:

The search engine enables users to find recipes using various inputs, such as textual queries, ingredient lists, or uploaded images. Integrating full-text search and machine learning ensures that results are relevant and personalized.

User Engagement:

Interactive features such as ratings, comments, and the ability to follow other users create a vibrant community. Users can engage with content through social interactions, share feedback, and discover new recipes. A leaderboard highlights trending recipes and top contributors, fostering a sense of achievement and competition among users.

Profile Management:

The profile management system allows users to upload recipes, save favorites, and customize their culinary preferences. Each profile includes a history of interactions, providing users with insights into their activity and preferences.

Challenges During Deployment

The deployment process presented several challenges. Initially, AWS Elastic Beanstalk was considered for hosting the backend. However, recent changes in AWS configurations made this option less feasible. An alternative approach using AWS API Gateway and AWS Lambda was explored. While effective, this approach required manual updates, as the API had to be zipped and uploaded after each iteration. This process took time and hindered rapid development.

Ultimately, Heroku was selected as the backend hosting platform due to its streamlined deployment workflow and robust feature set. Heroku's compatibility with FastAPI and its automated deployment pipelines resolved earlier challenges, enabling quicker iterations and reliable service.

Future Plans

Hosting Models:

The team plans to upload machine learning models to AWS S3, enabling direct calls from the FastAPI backend. This will enhance the scalability and efficiency of the application's inference processes, allowing faster response times for recommendations and image recognition.

User Activity Features:

Future updates will include functionality for users to search for and follow others. This feature will promote community building and provide recommendations based on similar user profiles. Enhanced analytics will also track user engagement, offering insights for further personalization.

By combining advanced machine learning models, dynamic routing, and server-side rendering, the DeepMinds Recipe Application bridges the gap between cutting-edge technology and

culinary exploration. This innovative platform is positioned for continuous evolution, ensuring a premium user experience and fostering a thriving culinary community.

Conclusion:

DeepMinds created a complete, functional app for food lovers to enjoy and bring their hobby into the modern age. This culinary sidekick will allow users to find and make delicious recipes by implementing a recommendation system and a state-of-the-art image recognition model. This report has detailed the datasets, methodologies, and performance metrics, along with the application architecture and key features.

The results demonstrate the effect of the machine learning models on the implemented systems. The recommendation system successfully implemented two collaborative filtering models using cosine similarity to determine similar users for recommendation and recipes for more straightforward lookup and finding. Meanwhile, image recognition is able to classify food images into various different classes for recipe recommendation. While both models have room for improvement, the recipe app provides the experience DeepMinds set out to create from its inception.

Looking ahead, the team plans to refine the application further by enhancing model performance, fostering user engagement through community features, and leveraging AWS services for efficient model hosting. These enhancements will improve the user experience while encouraging more experimentation and discourse in the culinary scene.

References:

Alvin. "Food.Com - Recipes and Reviews." *Kaggle*, 28 Dec. 2020, www.kaggle.com/datasets/irkaal/foodcom-recipes-and-reviews.

Fkih, Fethi Fkih. "Similarity Measures for Collaborative Filtering-Based Recommender Systems: Review and Experimental Comparison." *Journal of King Saud University - Computer and Information Sciences*, Elsevier, 16 Sept. 2021, www.sciencedirect.com/science/article/pii/S1319157821002652.

Karabiber, Fatih. "Cosine Similarity." *Learn Data Science - Tutorials, Books, Courses, and More*, www.learndatasci.com/glossary/cosine-similarity/. Accessed 13 Dec. 2024.

Salim, Farsana, et al. "DenseNet-201 and xception pre-trained deep learning models for fruit recognition." Electronics, vol. 12, no. 14, 19 July 2023, p. 3132, https://doi.org/10.3390/electronics12143132.

Sikka, Karan, and Maggie. "IFood - 2019 at FGVC6." *Kaggle*, 22 Apr. 2019, www.kaggle.com/competitions/ifood-2019-fgvc6/overview.

Volety, Rohan. "Food Recognition Model with Deep Learning Techniques [Updated]." *Labellerr*, Labellerr, 9 Dec. 2024, www.labellerr.com/blog/food-recognition-and-classification-using-deep-learning/.

Zhang, Yudong, et al. "Deep learning in food category recognition." *Information Fusion*, vol. 98, Oct. 2023, p. 101859, https://doi.org/10.1016/j.inffus.2023.101859.