Delisio: A Multimodal State Machine for Personalized Recipe Generation

Peyman Khodabandehlouei
peymankhodabandehlouei@gmail.com

Abstract

This paper presents Delisio (delisio.app) , an intelligent solution for generating personalized recipes, allowing users to enjoy a wide range of meals while maintaining their dietary goals. A Convolutional Neural Network (CNN) with transfer learning is used to accurately identify the dish name from an image (e.g., Paella, Sushi), and the recognized name is passed to a Large Language Model (LLM) to generate a personalized recipe based on the diet, nutritional goal, allergies, cooking level, and equipments. The CNN model achieves an accuracy of 92.49%, ensuring reliable food recognition, while the LLM generates high-quality, context-aware recipes.

1. Introduction

In recent years, LLMs have revolutionized our daily lives. Their advanced language understanding capabilities facilitate the development of agents that can interact with the world and successfully achieve a complex goal. When language understanding is combined with other capabilities, such as vision, these models become even more powerful. Among the many domains that intelligent agents can transform, food is particularly significant. As a fundamental aspect of human life, food is deeply connected to health, culture, and personal preferences.

Existing recipe applications and websites provide generic recipes that do not consider individuals' dietary needs and restrictions. Users often spend significant time searching for recipes that match their preferences, and even when they find suitable options, they may struggle to adapt them to their diet. This lack of personalization limits the usefulness of these platforms. While Large Language Models can generate recipes, they require effective instructions for personalizing recipe generation and cannot recognize dishes from images. oppositely, Convolutional Neural Networks can classify food images but cannot generate text-based recipes.

Delisio introduces a multimodal state machine architecture that seamlessly combines a Convolutional Neural Network for visual food recognition and a Large Language Model for personalized recipe generation. The state machine design ensures a structured transition from image recognition to personalized recipe creation.

This paper presents a Minimum Viable Product (MVP) implementation of the system, focusing on 10 diverse dish types to validate the concept.

2. Related Work

Food image classification has received growing attention in recent years, particularly with the rise of deep learning. CNNs have shown strong performance in classifying food items across datasets such as Food-101 and UECFOOD-256. Works like DeepFood [1] and FoodAI [2] demonstrate the effectiveness of transfer learning and fine-tuning pre-trained CNN architectures (e.g., ResNet, MobileNet) for recognizing food images in real-world environments. Despite these successes, most food classification models are limited to assigning class labels and do not extend to recipe generation or dietary personalization.

2.1. Recipe Generation with Language Models

Recent advances in Natural Language Processing (NLP), particularly the development of LLMs, have enabled models to generate natural-sounding recipes from structured prompts. Systems like GPT-3 have shown promise in generating ingredient lists and instructions based on user-defined dish names or ingredients. However, these models rely solely on text input and cannot process visual data, such as food images. Furthermore, they often require carefully crafted prompts and lack built-in understanding of user-specific dietary needs, such as allergies or nutritional goals.

2.2. Multimodal Approaches in Food AI

Some recent research attempts to bridge vision and language in the food domain using multimodal models. For instance, image-to-recipe models such as Recipe1M [3] and TastyNet explore matching visual inputs to a structured recipe database. However, these models often lack personalization and depend on static recipe retrieval rather than dynamic generation. They also do not leverage LLMs' ability to personalize outputs based on user context. To the best of my knowledge, there is no existing work that integrates

a CNN for visual dish recognition with an LLM for personalized recipe generation in a modular, state-machine architecture.

3. Methodology

This section describes the architecture and workflow of Delisio. The system is designed to seamlessly transition from image recognition to recipe generation, providing personalized culinary experiences for users.

3.1. System Overview

The workflow of Delisio, illustrated in Figure 1, consists of six main stages:

- **Image Input:** Users upload an image of a dish through the user interface.
- **Food Recognition (CNN):** The uploaded image is processed by a CNN to identify the dish name.
- **Dish Name Extraction:** The recognized dish name is extracted as text.
- **User Preferences Query:** The system retrieves the user's dietary preferences, allergies, and nutritional goals from the user database.
- **Recipe Generation (LLM):** The dish name and user preferences are passed to gpt-3.5-turbo, which generates a personalized recipe based on the user's profile.
- **Recipe Output:** The generated personalized recipe is returned to the user.

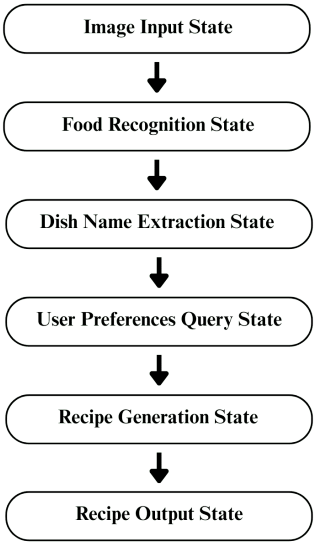

Figure 1. System workflow

3.2. Food Image Classification with CNN

For the food classification model, transfer learning is used. Several pre-trained CNN architectures are evaluated to determine the best model for food recognition. Each model is initialized with ImageNet weights.

3.2.1. Base Models Evaluated

The following base models were tested for performance:

- **MobileNetV2**
- **EfficientNetB0**
- **ResNet50**
- **InceptionV3**
- **DenseNet121**

The following layers were added as the classifier for the base models:

- **Global Average Pooling:** To reduce spatial dimensions.
- **Dense Layer:** 128 units with ReLU activation.
- **Dropout:** 0.5 to prevent overfitting.
- **Output Layer:** Softmax activation for multi-class classification.

All five models were trained for 5 epochs using the same training dataset and hyperparameters. The training accuracy of each model is presented in Figure 2, showing the comparative performance of the models.

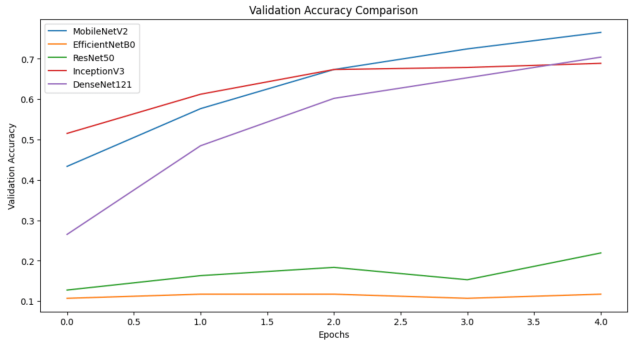

Figure 2. Validation accuracy of the base models over 5 epochs.

Table 1. Validation Accuracy of Base Models (5 Epochs)

Model	Validation Accuracy (%)
MobileNetV2	76.53
EfficientNetB0	11.73
ResNet50	21.94
InceptionV3	68.88
DenseNet121	70.41

As shown in Table 1, the best performing base model was MobileNetV2 on this dataset. An important observation was the poor performance of EfficientNetB0. This can occur for various reasons, such as the dense layer being too simple for this complex feature extractor. Further investigation and more extensive tests are needed for production-level decision making. However, since this paper aims to

validate the concept with an MVP, MobileNetV2 was chosen as the base model.

3.3. Hyperparameter Optimization

To further optimize the performance of the MobileNetV2-based model, hyperparameter tuning was performed using Keras Tuner with a Random Search strategy. This approach allows for systematic exploration of various hyperparameters to identify the best configuration for the model. The following hyperparameters were considered:

- **Number of Dense Layers:** Ranging from 1 to 3.
- **Units per Dense Layer:** 64 to 512 units, in steps of 64.
- **Batch Normalization:** Optional (Enabled or Disabled).
- **Dropout Rate:** 0.2 to 0.6, in steps of 0.1.
- **Learning Rate:** Log-scaled between 10^{-5} and 10^{-3}.

The Keras Tuner was configured to perform a Random Search with the following settings:

- **Search Algorithm:** Random Search.
- **Objective:** Validation Accuracy.
- **Maximum Trials:** 25.
- **Executions per Trial:** 1.
- **Training Epochs per Trial:** 30.

The training process was managed with the following callbacks:

- **Early Stopping:** Stopped training if validation accuracy did not improve for 5 epochs.
- **ReduceLROnPlateau:** Reduced the learning rate if validation loss plateaued.

3.3.1. Hyperparameter Search Results

After 25 trials, the best model configuration achieved a validation accuracy of **91.33%**. The optimal hyperparameters were as follows:

- **Number of Dense Layers:** 1
- **Units in Dense Layer:** 256
- **Dropout Rate:** 0.5
- **Learning Rate:** 0.000369
- **Batch Normalization:** Disabled

Table 2. Hyperparameter Tuning Results with Keras Tuner

Hyperparameter	Best Value
Number of Dense Layers	1
Units per Dense Layer	256
Dropout Rate	0.5
Learning Rate	0.000369
Batch Normalization	Disabled
Best Validation Accuracy	91.33%

3.4. Model Training and Evaluation

After identifying the optimal hyperparameters through Keras Tuner, the final model was trained using the MobileNetV2 architecture with the best configuration. The base model was initialized with ImageNet weights, and the top layers were replaced with a custom classifier. The final model architecture is shown below:

- **Base Model:** MobileNetV2 (pre-trained on ImageNet).
- **Feature Extractor:** Frozen base model to retain pre-trained knowledge.
- **Global Average Pooling:** To reduce spatial dimensions.
- **Dense Layer:** 256 units with ReLU activation.
- **Dropout:** 0.5 for regularization.
- **Output Layer:** 10 units with Softmax activation for multi-class classification.
- **Optimizer:** Adam with a learning rate of 0.000369.
- **Loss Function:** Categorical Cross-Entropy.
- **Metrics:** Validation Accuracy.

The training process was monitored using both training and validation loss, as shown in Figure 3. The model demonstrated stable training, with both training and validation losses decreasing consistently. No significant overfitting was observed, as the validation loss remained close to the training loss.

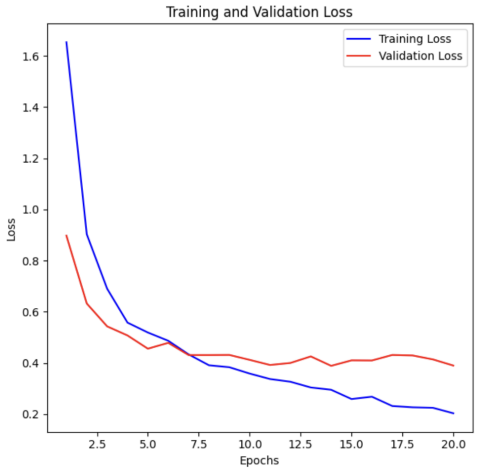

Figure 3. Training and Validation Loss of the Model.

The training and validation loss curves in Figure 3 show a stable learning process, with the validation loss converging close to the training loss. This indicates that the model has effectively learned without severe overfitting. The use of early stopping ensured that the training process was terminated at the optimal point.

To further evaluate the model's performance, a confusion matrix was generated on the validation set, as shown in Figure 4. The confusion matrix provides a detailed view of the model's performance across all 10 food classes. Each cell

indicates the number of correct and incorrect predictions for each class.

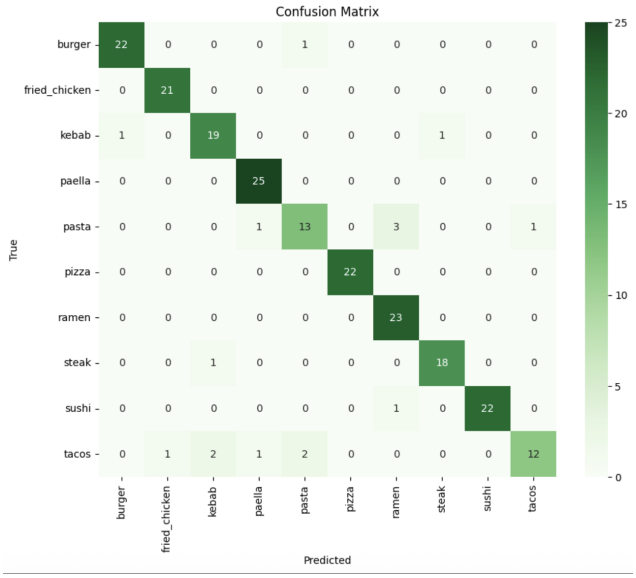

Figure 4. Confusion Matrix of the Model on the Validation Set.

The confusion matrix in Figure 4 reveals the model's performance across all classes:

- **High Accuracy Classes:** The model achieved strong performance in classes such as *Burger*, *Paella*, and *Ramen*, with high true positive values.
- **Misclassified Classes:** Some confusion was observed in visually similar classes, such as *Kebab* and *Tacos*. This may be due to the visual similarity of these dishes.
- **Balanced Performance:** The model demonstrated a balanced performance across the majority of the 10 classes, with no severe class imbalance.

Overall, the final model achieved a strong validation accuracy, and the confusion matrix provides insights into areas for potential improvement. The model can be further enhanced by fine-tuning the base model.

3.5. Fine-Tuning

After obtaining the best frozen model configuration, further fine-tuning was conducted by selectively unfreezing different numbers of layers in the base model. The objective was to identify the optimal number of unfrozen layers that maximized validation accuracy without overfitting. This approach leverages the pre-trained knowledge of the model while allowing it to adapt to the specific food classification task.

3.5.1. Fine-Tuning Configuration

The fine-tuning process was conducted as follows:

- **Base Model:** MobileNetV2 (pre-trained on ImageNet).
- **Range of Unfrozen Layers:** From 10 to 100 layers, in increments of 10.
- **Learning Rate:** 1×10^{-6} (low learning rate for stable fine-tuning).
- **Training Epochs:** 15 epochs for each configuration, with early stopping applied.
- **Early Stopping:** Stopped training if validation accuracy did not improve for 5 consecutive epochs.
- **Best Model Save:** The best model for each configuration was saved as "fine_tuned_[layers]_layers.h5".

3.5.2. Fine-Tuning Results

The validation accuracy achieved for each fine-tuning configuration is presented in Figure 5. The model achieved the highest validation accuracy of **89.8%** when the last **70 layers** of the base model were unfrozen. This result indicates that partial fine-tuning of the model significantly improves performance, likely because the upper layers of the base model can adapt to the specific food classification task.

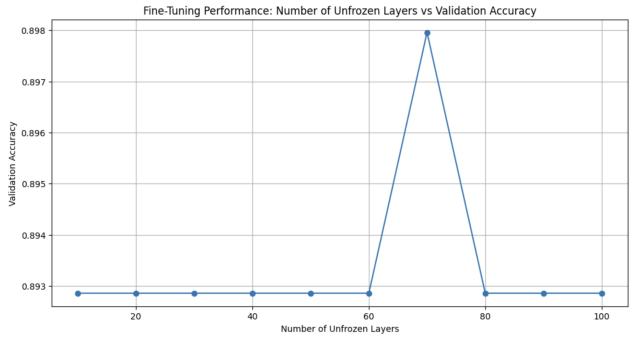

Figure 5. Fine-Tuning Performance: Validation Accuracy vs. Number of Unfrozen Layers.

3.5.3. Analysis and Discussion

The fine-tuning results reveal several important insights about the model's behavior:

- **Optimal Configuration (70 Unfrozen Layers):** The model achieved the highest validation accuracy of 89.8% when the last 70 layers were unfrozen. This suggests that selectively unfreezing a portion of the model's layers allows it to adapt effectively to the specific food classification task without disrupting the pre-trained knowledge.
- **Diminishing Returns Beyond 70 Layers:** Unfreezing more than 70 layers led to a decline in validation accuracy. This decline is likely due to overfitting, where the model begins to lose the generalizable features learned during pre-training and instead focuses excessively on the training data.
- **Stable Performance with Limited Fine-Tuning:** For configurations with fewer unfrozen layers (10 to 60), the model demonstrated stable performance, maintaining moderate validation accuracy. This stability indicates that

the pre-trained knowledge is effectively retained when the majority of the base model is frozen.

- **Effectiveness of Low Learning Rate:** The use of a low learning rate (1×10^{-6}) was crucial in maintaining stable fine-tuning, preventing the model from making large, unstable updates to the weights.
- **Final Model Performance Metrics:** The model was evaluated on the validation set, achieving the following classification performance:

Table 3. Final Model Performance Metrics on Validation Set

Class	Precision	Recall	F1-Score
Burger	0.9565	0.9565	0.9565
Fried Chicken	0.9545	1.0000	0.9767
Kebab	0.8636	0.9048	0.8837
Paella	0.9259	1.0000	0.9615
Pasta	0.8125	0.7222	0.7647
Pizza	1.0000	1.0000	1.0000
Ramen	0.8519	1.0000	0.9200
Steak	0.9474	0.9474	0.9474
Sushi	1.0000	0.9565	0.9778
Tacos	0.9231	0.6667	0.7742
Overall Accuracy		0.9249	
Macro Avg	0.9235	0.9154	0.9163
Weighted Avg	0.9256	0.9249	0.9223

These results demonstrate that the model achieves strong classification performance across most classes, with an overall accuracy of 92.49%. Notably:

- **High-Performing Classes:** The model achieved perfect precision and recall for *Pizza* and strong performance for *Sushi* and *Fried Chicken*.
- **Challenging Classes:** Slightly lower performance was observed for *Pasta* and *Tacos*, likely due to the visual similarity of these dishes with other classes.
- **Balanced Model Performance:** The model maintains a high weighted average for precision, recall, and F1-Score, indicating that it generalizes well to the entire dataset.

3.6. LangGraph State Machine

One of the core innovations of Delisio is the use of a state machine for managing the recipe generation process, implemented using the LangGraph library. LangGraph is a flexible Python library that enables the creation of scalable, maintainable state machines, making it an ideal choice for this task. The state machine ensures that the process of generating personalized recipes is well-organized, reliable, and easy to extend.

3.6.1. State Machine Design

The state machine used in Delisio is composed of six clearly defined states, as illustrated in Figure 1. Each state represents a specific stage in the recipe generation process:

- **Image Input State:** The user uploads an image of a dish through the user interface. This image is received by the system, and the state machine is triggered.
- **Food Recognition State:** The uploaded image is processed by the Convolutional Neural Network (CNN), which identifies the dish name (e.g., Paella, Sushi). The recognized dish name is extracted as text.
- **Dish Name Extraction State:** The extracted dish name is stored, and the state machine transitions to the next state. This ensures that the recognized dish name is reliably passed to the next stage.
- **User Preferences Query State:** The system queries the user database to retrieve the user's dietary preferences, allergies, and nutritional goals. These user-specific preferences are used to personalize the recipe.
- **Recipe Generation State:** The dish name and user preferences are passed to the Large Language Model (LLM) with a structured prompt. The LLM generates a personalized recipe that matches the user's dietary goals and restrictions.
- **Recipe Output State:** The generated recipe is formatted and returned to the user in a clear, user-friendly format. This marks the end of the state machine process.

3.6.2. LangGraph Implementation

Each of the six states was defined as a node in the LangGraph graph, with transitions specifying the conditions under which the system moves from one state to another. This modular approach ensures that the system is easy to understand, modify, and scale.

Core LangGraph Concepts Used:

- **State Nodes:** Each stage of the recipe generation process is defined as a separate node in the graph, making the workflow clear and modular.
- **Transition Conditions:** The state machine transitions from one state to the next based on successful operations (e.g., image recognized, user preferences retrieved).
- **Error Handling:** The state machine can gracefully handle errors, such as failed image recognition or LLM generation errors, by transitioning to appropriate error states.

3.6.3. Prompt Design for Recipe Generation

A critical aspect of the Recipe Generation State is the prompt design for the Large Language Model (LLM). The prompt is dynamically generated based on the user's preferences, ensuring that the generated recipe is highly personalized. The structured prompt follows this format:

> "You are a professional chef specialized in creating personalized recipes. Your task is to generate a healthy, easy-to-follow recipe that matches the user's dietary preferences, allergies, and nutritional goals. Generate a personalized recipe for [Dish Name] considering a [Diet Type] diet that

follows [Nutritional Goal] goal. Avoid the following ingredients: [Allergies]."

This approach ensures that the LLM receives clear instructions, resulting in accurate and user-specific recipes. If the LLM fails to generate a recipe (e.g., due to a network error), the state machine can re-prompt the LLM or return a user-friendly error message.

3.6.4. Scalability and Future Improvements

The use of LangGraph makes the Delisio state machine highly scalable. Additional states (such as advanced user settings, ingredient substitution, or multi-step recipe generation) can be easily added without disrupting the existing workflow. The modular design also ensures that new LLM models or image recognition models can be integrated with minimal modification.

4. Data Description

High-quality data is the most critical component of this project, directly impacting the model's performance and generalization. Although there are several existing open-source food image datasets, such as Food-101, they often lack the necessary image quality and diversity required for this project. To overcome this challenge, a custom high-quality dataset was created by scraping images using Google and Bing APIs including 10 classes:

- **Burger**
- **Fried Chicken**
- **Kebab**
- **Paella**
- **Pasta**
- **Pizza**
- **Ramen**
- **Steak**
- **Sushi**
- **Tacos**

Each image was manually checked for quality, ensuring that only clear, high-resolution images were included. This careful monitoring significantly improved the model's training quality as eventually model's performance.

4.1. Data Preprocessing

All collected images were preprocessed using the following steps:

- **Image Resizing:** All images were resized to (224, 224) pixels.
- **Train-Validation-Test Split:** The dataset was manually divided into three subsets, with 80% of the images used for training, 10% for validation, and 10% for testing. This balanced split ensures that the model is trained on a sufficient amount of data while maintaining separate validation and test sets for unbiased performance evaluation.
- **Data Augmentation:** During training, real-time data augmentation was applied to enhance model generalization and prevent overfitting. The following augmentation techniques were used:
  - **Rotation:** Images were randomly rotated within a range of 0-20 degrees, introducing variety in image orientation.
  - **Width and Height Shift:** Images were randomly shifted horizontally and vertically by up to 20%, simulating slight changes in the position of the food within the image.
  - **Shear Transformation:** A shear range of 20% was applied to introduce minor perspective distortion.
  - **Zooming:** Random zooming (up to 20%) was applied to simulate varying distances of the camera from the food.
  - **Horizontal Flipping:** Images were randomly flipped horizontally, adding further variety.
  - **Pixel Rescaling:** All pixel values were scaled between 0 and 1 for normalization.
  - **Fill Mode:** Any empty pixels generated by augmentation were filled using the "nearest" mode, preserving image integrity.

4.2. Dataset Availability

The final custom dataset has been made publicly available to promote transparency and reproducibility. It can be accessed through the following Google drive:

https://drive.google.com/drive/folders/1o-WyAllNMNSCVfOZKQbcVazKbmeqSp_T?usp=sharing

4.3. Class Distribution and Sample Images

The dataset consists of 10 food classes, each with a balanced number of high-quality images. Figure 6 shows sample images from each class, providing a visual representation of the data diversity.

Figure 6. Sample Images from Each Class in the Custom Dataset.

5. Conclusion

This paper presented Delisio, an intelligent solution for personalized recipe generation, leveraging a multimodal state machine that combines a Convolutional Neural Network (CNN) for food recognition and a Large Language Model (LLM) for recipe generation. The use of a state machine with the LangGraph library ensured a clear, modular, and scalable workflow, making the system both flexible and maintainable.

The model demonstrated strong performance, achieving an overall accuracy of 92.49% for food classification across 10 diverse dish types. Fine-tuning the base model further enhanced performance, with careful selection of the number of unfrozen layers. The final model achieved balanced performance across most classes, and the use of user-specific preferences allowed for highly personalized recipe generation.

The creation of a custom high-quality dataset was a critical factor in real-world accuracy of this project, providing clean, diverse images that ensured effective training. Data augmentation and hyperparameter tuning further optimized the model, making it robust and efficient.

Future work may focus on expanding the number of dish classes, improving the LLM's ability to generate more complex and culturally diverse recipes, and enhancing user interaction by incorporating additional customization options, such as cooking methods or dietary preferences.

Delisio represents a scalable, adaptable solution that bridges computer vision and natural language processing for a truly personalized culinary experience.

6. Supplementary Material

This section provides additional resources and materials that support the Delisio project. These resources ensure transparency, reproducibility, and further exploration for interested readers.

6.1. Code and Data Availability

The full source code for the Delisio project, including the model training, state machine implementation, and recipe generation code, is available on GitHub. The custom high-quality food image dataset used for training and evaluation is also publicly accessible.

- **Code Repository:** https://github.com/PeymanKh/delisio_mvp
- **Dataset Repository:** https://drive.google.com/drive/folders/1o-WyAllNMNSCVfOZKQbcVazKbmeqSp_T?usp=sharing

6.2. Delisio Marketing Page

For trying latest Delisio version in production, including its features and user experience, please visit the official marketing page: https://delisio.app

References

[1] Chang Liu, Yu Cao, Yan Luo, Guanling Chen, Vinod Vokkarane, and Yunsheng Ma. Deepfood: Deep learning-based food image recognition for computer-aided dietary assessment. *Proceedings of ICOST*, 2016. 1

[2] Doyen Sahoo, Wang Hao, Ke Shu, Wu Xiongwei, Palakorn Achananuparp, Ee-Peng Lim, and Steven CH Hoi. Foodai: Food image recognition via deep learning for smart food logging. *arXiv preprint arXiv:1909.11946*, 2019. 1

[3] Amaia Salvador, Michal Drozdzal, Xavier Giro-i Nieto, Adriana Romero, Piotr Bojanowski, and Aaron Courville. Learning cross-modal embeddings for cooking recipes and food images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3020–3028, 2017. 1