Purdue University

VIP IPAA I

VIPER VIP Final Report

Nutrition Change Estimation Through Video

Team Members

William Tao

Piotr Nabrzyski

Donny Weintz

Benjamin Nguyen

Instructor: Prof. Zhu, Prof. Zoltowski, Prof. Delp, Prof. Dinani

December 13, 2025

Contents

1 Abstract

Tracking the different ingredients and their caloric content in a meal is essential for personal health and dietary management. Having the ability to identify and estimate the caloric value of an item when picked up can be extremely beneficial when making real-time decisions for meal prepping. It can also reduce the need to manually measure the amount of food you're using because it's all digital. By utilizing an object detection model trained on datasets containing ingredients and foods from around the world, detecting ingredients from a video becomes an easy task. However, mapping each item to an accurate calorie value is challenging. Our project addresses this problem by leveraging a YOLO [2] (You Only Look Once) object detection model for ingredient identification, utilizing the FNDDS dataset [6] for calorie mapping, and tracking the current and cumulative calorie values in a video.

2 Introduction

Computer Vision is an extremely important field in machine learning with many different real-world applications, such as self-driving cars, surveillance, medical assistance, and more. In this paper, we aim to apply computer vision for nutrition estimation in videos, specifically, identifying ingredients, tracking calories, and mapping them to recipes found online. Our solution utilizes a three-stage pipeline. First, we scan the kitchen environment and leverage hand and cookware detection for a concentrated area for the next step, ingredient detection. We used a pretrained YOLOv8 [2] model as our object detection model to classify the ingredients, and fine-tuned on the HD-EPIC [4] recipe preparation videos and Viper Food Net (VFN) dataset [3]. Finally, we mapped the identified ingredients to the FNDDS dataset [6] to retrieve the energy (caloric) values. For real-world applicability, we added recipe matching with Recipe NLG [1] for reconstruction and potential sharing in the future.

3 Datasets

3.1 HD-EPIC

The HD-EPIC Dataset is a collection of recordings that capture a broader range of real-world cooking activities including fetching, prepping, and weighing/adding ingredients through a first-person view via Project Aria glasses [4] thanks to 9 participants. Furthermore, the dataset also contains a wide variety of annotations consisting of narrations, parsing, recipe, sound, action boundaries, object motion, object itinerary, and object priming. In total, there are 2.3 TB of videos and video annotations.

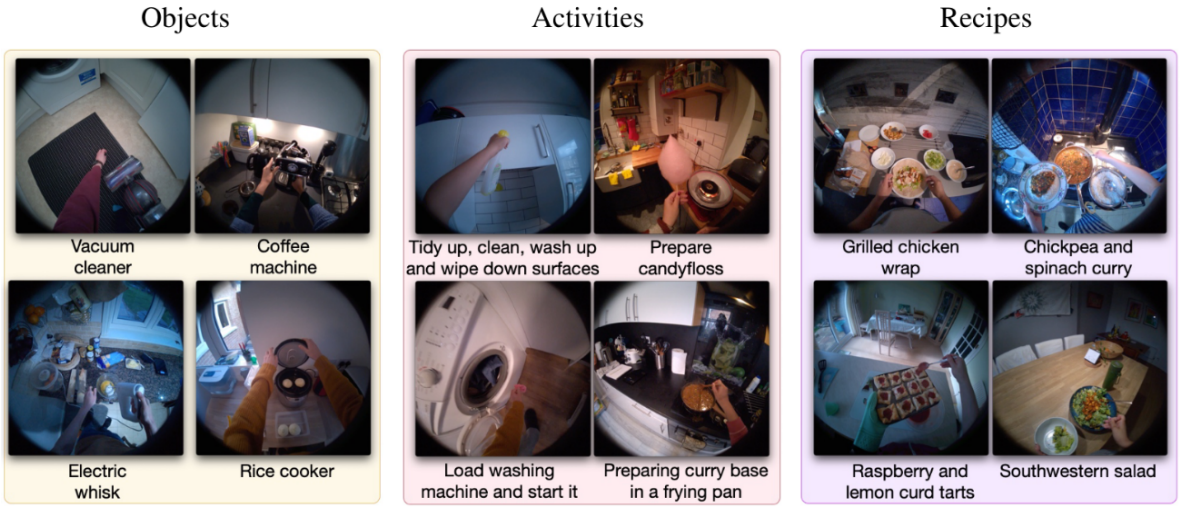


Figure 1: Dataset Description from HD-Epic Website [4]

In our project, video samples from this dataset were used to test our model specifically in comparison with the object and recipe annotations shown in Figure 1.

3.2 Viper Food Net

The Viper Food Net or VFN Dataset [3] is a selection of food images along with bounding box coordinates to locate the food item and a corresponding food category from the What We Eat In America (WWEIA) database. The content of this dataset is made up of 14991 images, 22423 bounding boxes, and 82 respective categories from the WWEIA index.

Figure 2: Sample Image from VFN Database [3]

Figure 3: Sample Image from VFN with Bounding Box [3]

The use of this dataset in our project was an addition of training data for our model to recognize where objects are and what those objects are. In terms of object detection, the location of the object is compared to the ground truth bounding box as shown between Figures 2 and 3

3.3 Food and Nutrient Database for Dietary Studies

The Food and Nutrient Database for Dietary Studies or FNDDS [6] gives nutrition values for corresponding foods or drinks from the WWEIA index as well as one of 5,432 unique food codes within a .xlsx file. Additionally included in this file are a description of the item, its categorizing number from the WWEIA index, and a description of its WWEIA category.

FNDDS Nutrient Values

2021-2023 Food and Nutrient Database for Dietary Studies - At A Glance

Food cod	Main food description	WWEIA Category numb	WWEIA Category description	Energy (kcal)
42401010	Coconut milk, used in cooking	1902	Plant-based milk	230
42401100	Yogurt, coconut milk	1904	Plant-based yogurt	108
42401200	Yogurt, almond milk	1904	Plant-based yogurt	128
42402010	Coconut cream, canned, sweetened	1902	Plant-based milk	357
42403010	Coconut water, unsweetened	7204	Fruit drinks	18
42404010	Coconut water, sweetened	7204	Fruit drinks	37
42500000	Trail mix, NFS	2804	Nuts and seeds	454
42500100	Trail mix with nuts	2804	Nuts and seeds	604
42501000	Trail mix with nuts and fruit	2804	Nuts and seeds	454
42501500	Trail mix with chocolate	2804	Nuts and seeds	503
42502100	Trail mix with pretzels, cereal, or granola	2804	Nuts and seeds	476

Figure 4: Sample from FNDDS Nutrient Values .xlsx File [6]

This is the dataset that we've integrated and mapped to in our project in order to return calorie values from the foods that are recognized.

3.4 RecipeNLG

The RecipeNLG dataset contains 1,312,871 unique cooking recipes with 2,226,362 unique ingredients alongside the recipe instructions [1]. This dataset was used for recipe mapping, with ingredients and preparation tools we detected in the HD-EPIC videos.

title	ingredients	directions	link	source	NER
1312871 unique values	**2226362** unique values	**2211644** unique values	**2231142** unique values	Gathered 74% Recipes1M 26%	**2133496** unique values
No-Bake Nut Cookies	["1 c. firmly packed brown sugar", "1/2 c. evaporated milk", "1/2 tsp. vanilla", "1/2 c. broken nuts...	["In a heavy 2-quart saucepan, mix brown sugar, nuts, evaporated milk and butter or margarine.", "St...	www.cookbooks.com/Recipe-Details.aspx?id=44874	Gathered	["brown sugar", "milk", "vanilla", "nuts", "butter", "bite size shredded rice biscuits"]
Jewell Ball'S Chicken	["1 small jar chipped beef, cut up", "4 boned chicken breasts", "1 can cream of mushroom soup", "1 c...	["Place chipped beef on bottom of baking dish.", "Place chicken on top of beef.", "Mix soup and crea...	www.cookbooks.com/Recipe-Details.aspx?id=699419	Gathered	["beef", "chicken breasts", "cream of mushroom soup", "sour cream"]
Creamy Corn	["2 (16 oz.) pkg. frozen corn", "1 (8 oz.) pkg. cream cheese, cubed", "1/3 c. butter, cubed", "1/2 t...	["In a slow cooker, combine all ingredients. Cover and cook on low for 4 hours or until heated throu...	www.cookbooks.com/Recipe-Details.aspx?id=10570	Gathered	["frozen corn", "cream cheese", "butter", "garlic powder", "salt", "pepper"]

Figure 5: Sample from RecipeNLG Dataset [1]

4 Detection Methods

We combine two different detection models for our nutrition estimation system. For ingredients and preparation tools, we use YOLOv8 [2], a YOLO [5] (You Only Look Once) series model. For hand detection, we use MediaPipe Hands [7]. Together, they help us identify ingredients in each frame and fixate on the relevant objects by narrowing the detection region around the hands.

4.1 You Only Look Once

4.1.1 YOLO Overview

YOLO [5] (You Only Look Once) is a popular object detection model that excels in real-time applications while maintaining high accuracy. It predicts bounding boxes and class probabilities in a single forward pass, making it suitable for our application. In this project, we used the pretrained Ultralytics YOLOv8 model [2], and fine-tuned it on HD-EPIC [4] and VFN [3] data for increased performance on ingredient detection and HD-EPIC evaluation.

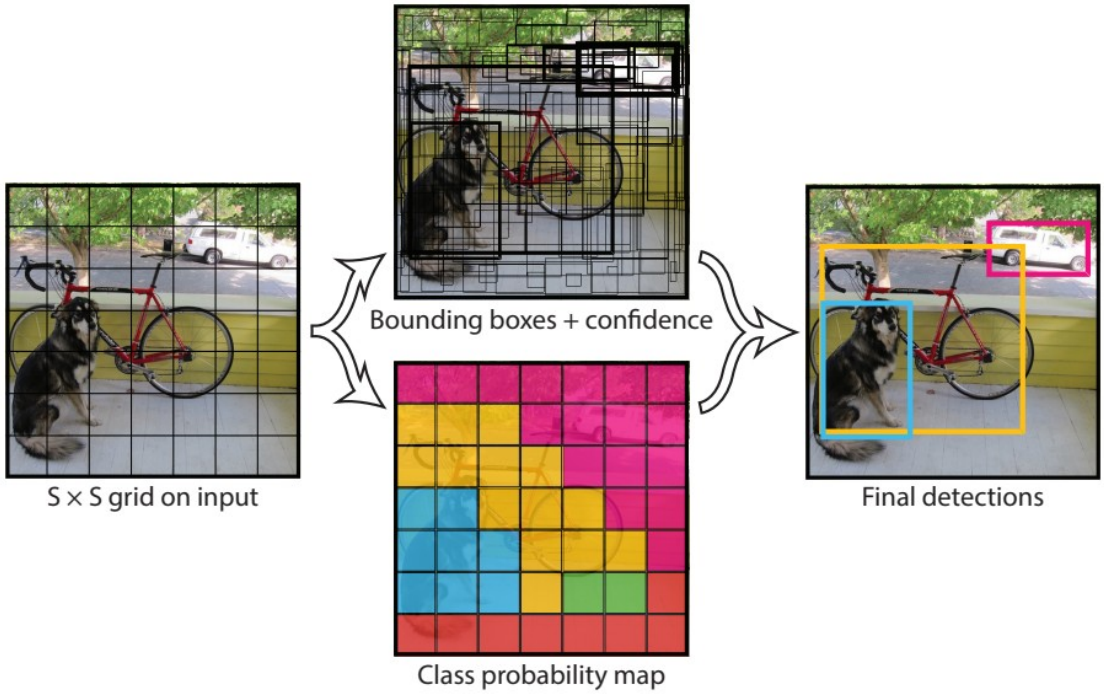


Figure 6: YOLO pipeline. Reproduced from Redmon *et al.* [5]

The model consists of a CNN backbone, which excels in feature extraction, especially in images. First, the image is divided into an $S \times S$ grid (original paper uses 7×7), where each cell is responsible for detecting objects. Each cell predicts bounding box coordinates (x, y, w, h), an objectness confidence score (how likely an object is present), and class probabilities. From there, Non-Maximum Suppression (NMS) is used to remove duplicate bounding boxes if there are overlaps.

4.1.2 YOLO Loss

Not only does the single forward pass, the basis of YOLO, allow for real-time processing and a quicker training framework, but the structure of the loss function, as seen below

in equation 1, also helps with efficiency.

$$
\begin{aligned}
L = & \sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \\
& + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
& + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
& + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2
\end{aligned}
\tag{1}
$$

To understand this huge loss function, we can break it down into two components – object class probability loss and localization loss:

$$L = L_{cls} + L_{loc} \tag{2}$$

The class loss, L_{cls}, can be then expanded as the first line of 1:

$$L_{cls} = \sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \tag{3}$$

Where S^2 is the number of grid cells in the image, $\mathbb{1}_i^{obj}$ is an indicator (1 if an object is present in cell i, and otherwise 0), c is the class index from the set of possible classes, and $p_i(c)$ and $\hat{p}_i(c)$ are, respectively, the ground truth and predicted probabilty that cell i belong to class c. The purpose of the indicator is to only count those grid cells where there is an object in that grid cell.

The localization loss is fairly longer than the class probability loss, so it can be broken down as confidence loss and coordinate loss:

$$L = L_{conf} + L_{coord} \tag{4}$$

Starting with the coordinate loss, weighed by a factor, λ_{coord}, is calculated using the following equation:

$$L_{coord} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \, l \tag{5}$$

With:

$$l = \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 + (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \tag{6}$$

Where x and y are the coordinates and w and h are the dimensions of bounding box i. The square roots of the widths and heights are there to prevent the limitation where, without taking the square root, the same error in a small box would be worse than that in a larger box. So, in essence, as the box size scales linearly, the square roots of the width and height scale sublinearly, which effectively suppresses the differences between large and small boxes.

The last component of the loss function is the confidence loss, L_{conf}, as seen below:

$$L_{conf} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \left[\mathbb{1}_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2 \right] + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \left[\mathbb{1}_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2 \right] \tag{7}$$

which is separated into confidence loss where an object is present, and where an object is not present, where the latter is scaled down by a factor, λ_{noobj}. This is to prevent the limitations of training confidence scores being pushed to zero, as well as the coordinate loss contribution being low as many grid cells may not contain an image.

With these individual loss components, the total loss function as seen in 1 is constructed. Though the total loss function may appear daunting, we can see that each individual component is actually a simple sum of squared errors loss function, making the total loss nothing but a sum of squared errors loss, and effectively enabling YOLO to be a regression problem.

4.2 MediaPipe Hands

MediaPipe Hands [7] is a real-time hand tracking system that uses a two stage detection pipeline. The first stage is a single-shot palm detector model that identifies palms in each video frame. The second stage runs a landmark detector in the palm detection region to identify 21 hand landmarks. The model architecture for each stage is shown in Figures 7 and 8.

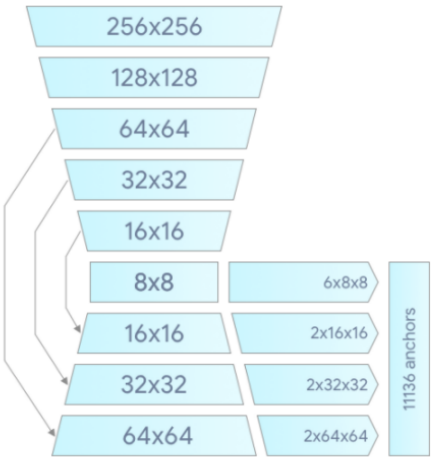


Figure 7: Palm Detector. Reproduced from Zhang *et al.* [7]

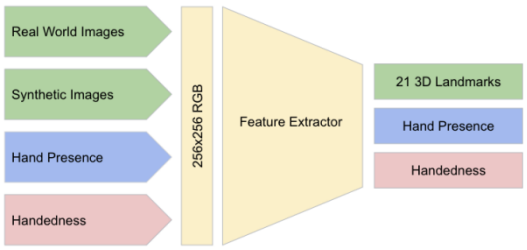


Figure 8: Landmark Detector. Reproduced from Zhang *et al.* [7]

In our nutrition-estimation pipeline, we leverage MediaPipe Hands to identify where interaction with ingredients is likely to occur. Since all ingredients added to a recipe must be touched by the participant, palm locations provide reliable cues for where food items will appear in the image. For each detected palm, we extract a small rectangular region around the detection and run our ingredient detection model within this region. This approach reduces unnecessary computation and improves ingredient detection accuracy by focusing inference on the most relevant image regions.

4.3 Region-based Convolutional Neural Network

In the original YOLO paper, Redmon proposed that potentially combining YOLO with a Fast Region-based Convolutional Neural Network (R-CNN) could decrease localiza-

tion and background errors leading to a general boost in performance [5].

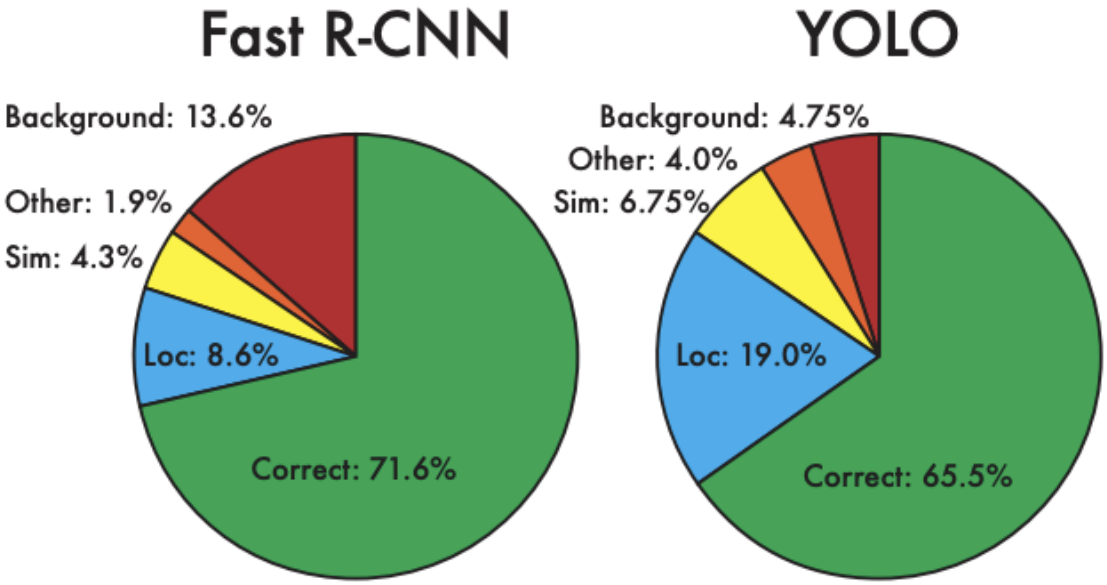


Figure 9: Fast R-CNN vs. YOLO Error Analysis. Reproduced from Redmon *et al.* [5]

We can see in Figure 9, that Fast R-CNN has significantly lower localization errors with more correct detections, while YOLO offers significantly fewer background errors. The general idea Redmon proposed was to make predictions with both R-CNN and YOLO, and if both models predict a similar box, then that prediction is boosted. We attempted to implement Faster R-CNN alongside YOLO, but found that the overall results showed a worse performance than solely YOLO, as seen in Figure 10, while also significantly increasing the inference time, resulting in less than real-time inference. We also experimented with slightly changing this combination so that YOLO still acted as a detector and classifier, but Faster R-CNN acted as purely a binary detector that detected whether any ingredient was present or not. Although achieving better results with this new application of Faster R-CNN, we still found that the overall performance of the combination did not have significant enough improvement to make up for the longer inference time, as seen in Figure 11. Due to this, our final model omits R-CNN completely, and this section is more of a note on R-CNN.

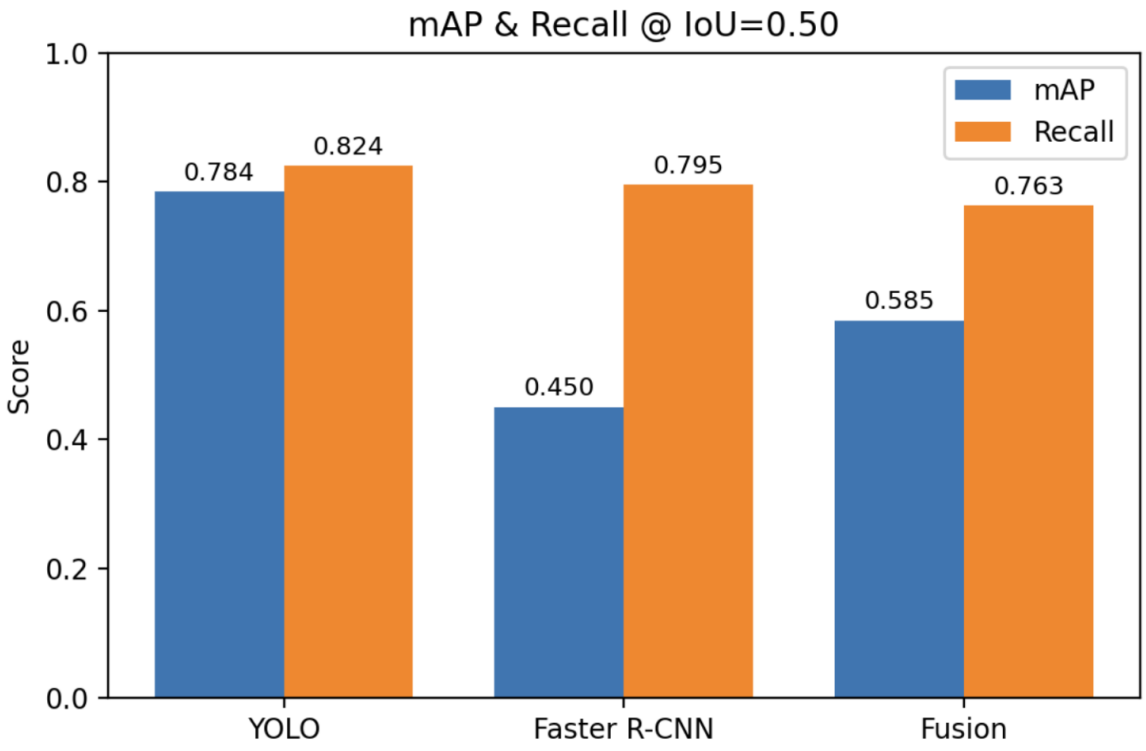


Figure 10: YOLO + Faster R-CNN as a Classifier

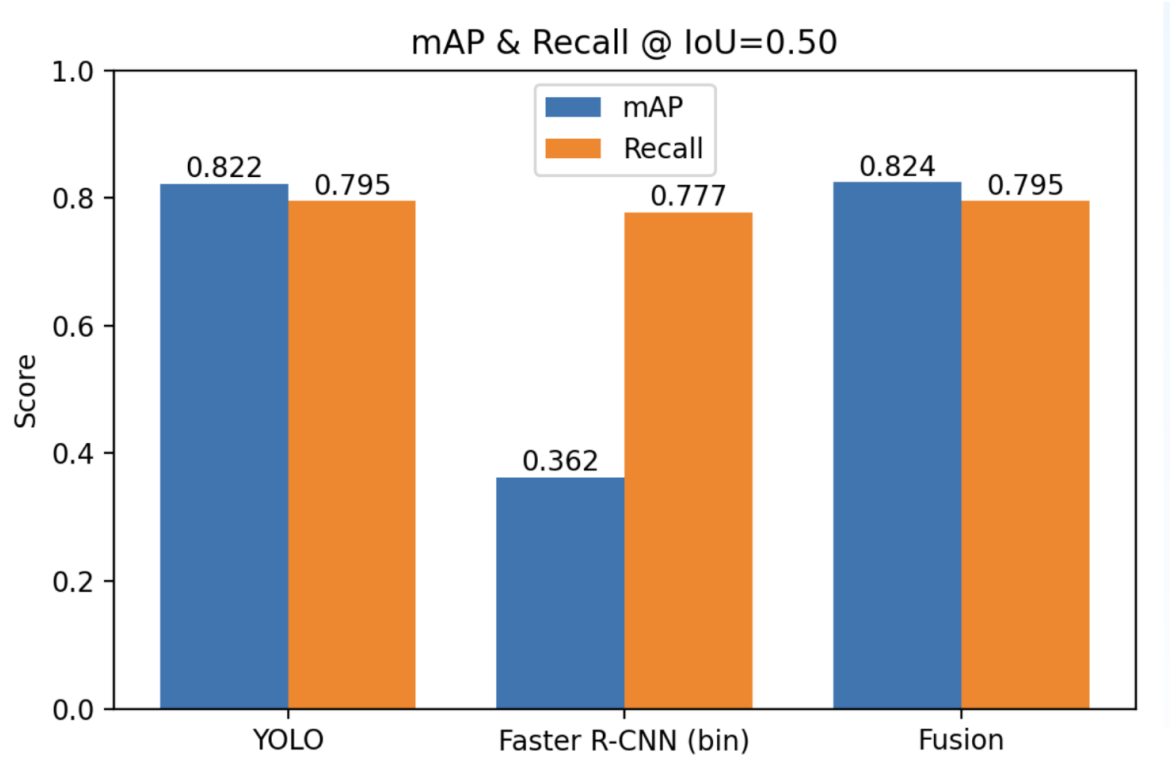


Figure 11: YOLO + Faster R-CNN as a Binary Detector

5 Calorie Mapping

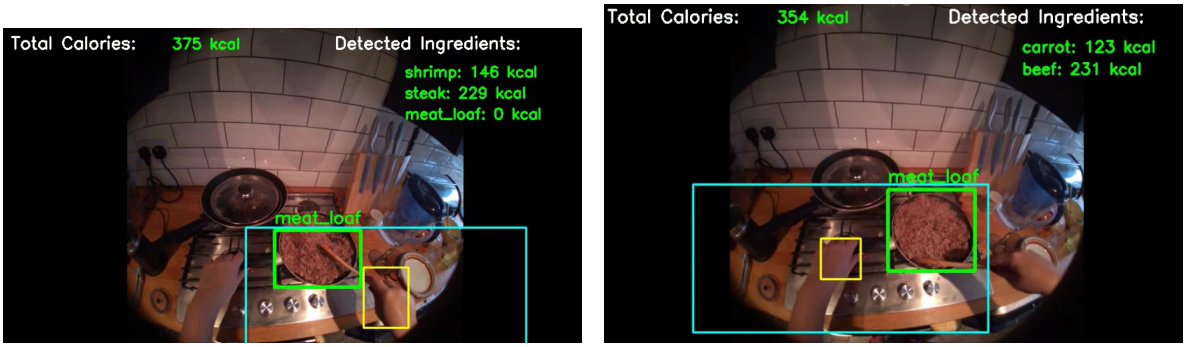
To translate our ingredient classifications into something more meaningful, we used the FNDDS dataset [6], discussed in more detail in Section 3.3, to retrieve the corresponding calorie values. Integrating FNDDS into our detection pipeline came with some obstacles, namely, how we should be processing the detections. Because our model was trained on multiple datasets without proper labels, there were instances where classifications came with errors like "meat_loaf" instead of "meat loaf". In the FNDDS dataset, there are many similar entries with minor differences. An example can be seen in Figure 12. To combat this, we normalize the classifications by removing underscores, capitalization, and extra spaces that may cause mapping errors in the dataset.

Main food description	WWEIA Category numbe	WWEIA Category description	Energy (kcal)
Cheese, NFS	1602	Cheese	381
Cheese, Blue or Roquefort	1602	Cheese	353
Cheese, Brick	1602	Cheese	371
Cheese, Camembert	1602	Cheese	300
Cheese, Brie	1602	Cheese	334
Cheese, Cheddar	1602	Cheese	409
Cheese, Cheddar, reduced fat	1602	Cheese	316
Cheese, Cheddar, nonfat or fat free	1602	Cheese	157
Cheese, Colby	1602	Cheese	394
Cheese, Colby Jack	1602	Cheese	393
Cheese, Feta	1602	Cheese	273
Cheese, Fontina	1602	Cheese	389
Cheese, goat	1602	Cheese	364
Cheese, Gouda or Edam	1602	Cheese	356
Cheese, Gruyere	1602	Cheese	413
Cheese, Limburger	1602	Cheese	327
Cheese, Monterey	1602	Cheese	392
Cheese, Monterey, reduced fat	1602	Cheese	313
Cheese, Mozzarella, NFS	1602	Cheese	296
Cheese, Mozzarella, part skim	1602	Cheese	296
Cheese, Mozzarella, reduced sodium	1602	Cheese	280
Cheese, Mozzarella, nonfat or fat free	1602	Cheese	141

Figure 12: Display of similar entries with varying energy values in the FNDDS dataset [6]

To match cheese to a correct energy value, we take the *NFS* entry, standing for "Not Further Specified", as a general entry. When experimenting with our model on the HD-EPIC videos [4], we encountered another problem where our model was double-

counting the same object as two different classes. In Figure 13, we can see the same object being classified as meat_loaf and steak. To prevent cases like this, we manually created a food aggregation list to move specific classifications into a broader category. In this case, we decided that meatloaf and steak fall into the beef category.



Double (incorrect) classifications Generalized classification

Figure 13: Change of ingredient classification method

This strategy is extremely important in detection cases like these, as we don't want to double-count objects and cause confusion for users. Also, for foods that look similar and can be confusing to the ingredient detector, this simplifies classifications. The aggregation list has its flaws, as it was created manually, and could potentially miss other foods that could benefit from generalization. Some examples from the aggregation list are shown in Table 1.

Ingredient	Generalized Category
steak	beef
meat loaf	beef
chicken breast	chicken
grilled chicken	chicken
pork chop	pork

Table 1: Some examples of generalized ingredients

Despite these attempts at improving the calorie mapping, there's still a fundamental problem with our current system. The calorie values in FNDDS [6] are **per 100 grams** of the respective food. As of now, we can't quantify the amount of food that's present in the video, leading to misconceptions in the calorie amounts that are shown in the results. We can only provide an estimation of how much food is present. Future work will focus on adding quantity estimations for more accurate calorie tracking for a more balanced tool.

6 Recipe Mapping

As we lack the ability to currently estimate how much of an ingredient was added to a given recipe in our videos, as highlighted in Section 5, for more immediate real world applicability, we integrated recipe mapping by utilizing the RecipeNLG dataset [1].

6.1 Ingredient Matching

The ingredient mapping was performed by calculating the intersection over union (IoU), or Jaccard index, between our detected ingredients and the ingredients in each recipe from RecipeNLG. These were almost direct one-to-one matches as the ingredient in RecipeNLG are listed separately in addition to within the instructions, which made the lookup fairly straightforward. However, as also discussed in Section 5, not all ingredient names in our detection datasets match the corresponding ingredient names in RecipeNLG. This meant that we had to introduce manual canonicalization. The most relevant example of this was in the HD-EPIC video where the participant was cooking Cacio e Pepe. Due to the natural canonicalization of the term "pasta" in our detections, there was a disparity where the RecipeNLG ingredient called for "penne". This meant that for this specific example, we had to canonicalize the ingredients both from our detection model and from the RecipeNLG dataset to the same word. This is again, only specific to this example; for each recipe that uses a non-canonical word such as "parmesan", we have to write a manual canonicalization mapping for all different types of cheeses to be "cheese". This in effect prevents us from the ability to automate and generalize this to all recipes and cooking videos as it is very difficult and time-consuming to manually write a canonicalization for every potential ingredient we may encounter.

6.2 Preparation Tool Matching

To match the preparation tools between our detections and RecipeNLG, canonicalization was not as big of an issue, as generally things like "pot" are already canonical in all recipes. We did, however, not have access to the preparation tools in the same manner as ingredients as they were not listed separately from the instructions. This meant that we had to scrape through the instructions to find all the words that were within our classes for preparation tools. Once we retrieved all of the relevant preparation tools from the recipes, we then performed the same process as with the ingredients, and calculated the IoU to find the score of matched ingredients.

6.3 Scoring Recipes

We then combined the two IoUs by placing more emphasis on the ingredient score, as people may use different preparation tools to perform the same tasks, and matched the top scoring recipes. This helps with the issue in calorie mapping where we cannot detect how much of an ingredient was added, by mapping a video to a recipe, and finding out how many calories are per serving from that recipe.

7 Nutrition Change Estimation Pipeline

To integrate all components of our system into a cohesive solution, we designed a three-stage processing pipeline. The workflow is organized into the following phases: scanning the kitchen environment, detecting ingredients, and estimating nutrition information. An overview of this pipeline is illustrated in Figure 14. Each stage is described in detail in the following sections.

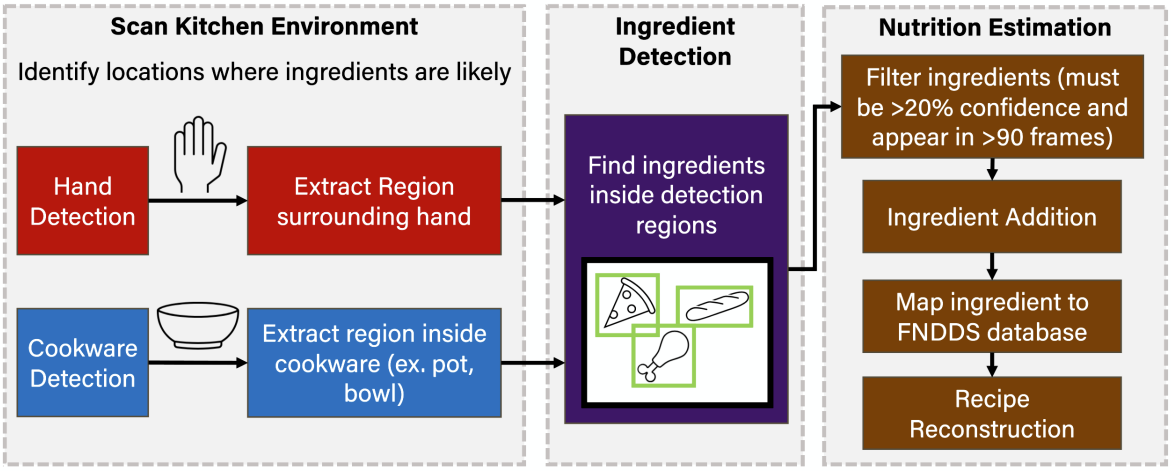


Figure 14: Nutrition Change Estimation Pipeline.

7.1 Scanning Kitchen Environment

Scanning the kitchen environment is a critical first step in our pipeline. Its primary purpose is to localize regions within each frame where ingredients are most likely to appear. To achieve this, we run MediaPipe Hands as described in Section 4.2. For each detected hand, we generate a small region of interest (ROI) and apply our ingredient detection model within that region. This enables us to identify ingredients that the user is actively interacting with. Figure 15 illustrates a sample hand detection (yellow) and the corresponding ingredient prediction ROI (cyan).

In addition to hand tracking, we incorporate a preparation tool detection model that identifies cookware such as pots, pans, bowls, and plates. Since these objects are commonly involved in food preparation, we also run our ingredient detection model

within their bounding boxes. Figure 15 shows an example of cookware detection in pink. Together, these targeted ROIs allow us to focus on meaningful regions of the scene and substantially reduce false positives.

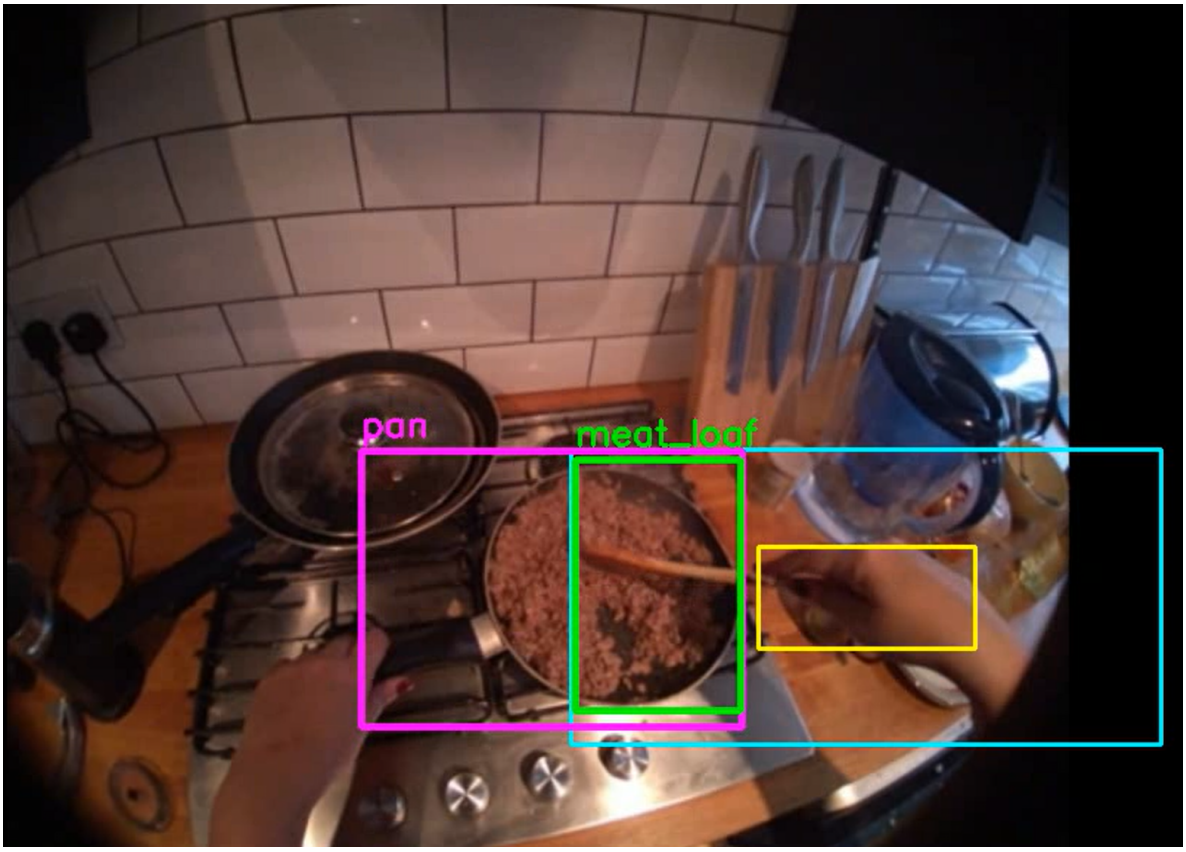


Figure 15: Sample frame of detected hand region, preparation tool, and ingredient.

7.2 Ingredient Detection

The second stage of our pipeline is ingredient detection. As described in Section 7.1, we first extract regions of interest where ingredients are likely to appear. Within these ROIs, we apply our ingredient detection model detailed in Section 4.1, which identifies candidate ingredients being added to the recipe. A sample ingredient detection is shown in green in Figure 15.

7.3 Nutrition Estimation

The final stage of our pipeline is estimating the nutritional value of the detected ingredients. We begin by filtering ingredient predictions to reduce noise and eliminate false positives. An ingredient is only considered valid if it appears in at least 90 frames with a confidence of at least 20%, and these frames do not need to be consecutive. For a typical 60 fps video, this corresponds to roughly 1.5 seconds of visibility. Once an ingredient satisfies these criteria, it is treated as having been added to the recipe.

We then map each validated ingredient to its corresponding entry in the FNDDS database, as described in Section 5. After processing all ingredients, we attempt to match the resulting ingredient sequence to a known recipe using the method outlined in Section 6.

8 Results

8.1 Ingredient Detection

As mentioned in 4.1, we employ a YOLO model for ingredient detection. Starting from the pretrained Ultralytics YOLOv8 model, we fine tuned it on data from the VFN [3] dataset. We preprocessed the VFN dataset and removed any food classes that were not relevant for ingredient detection. This narrowed the dataset from 82 classes to 58. The final class distribution is shown in Figure 16.

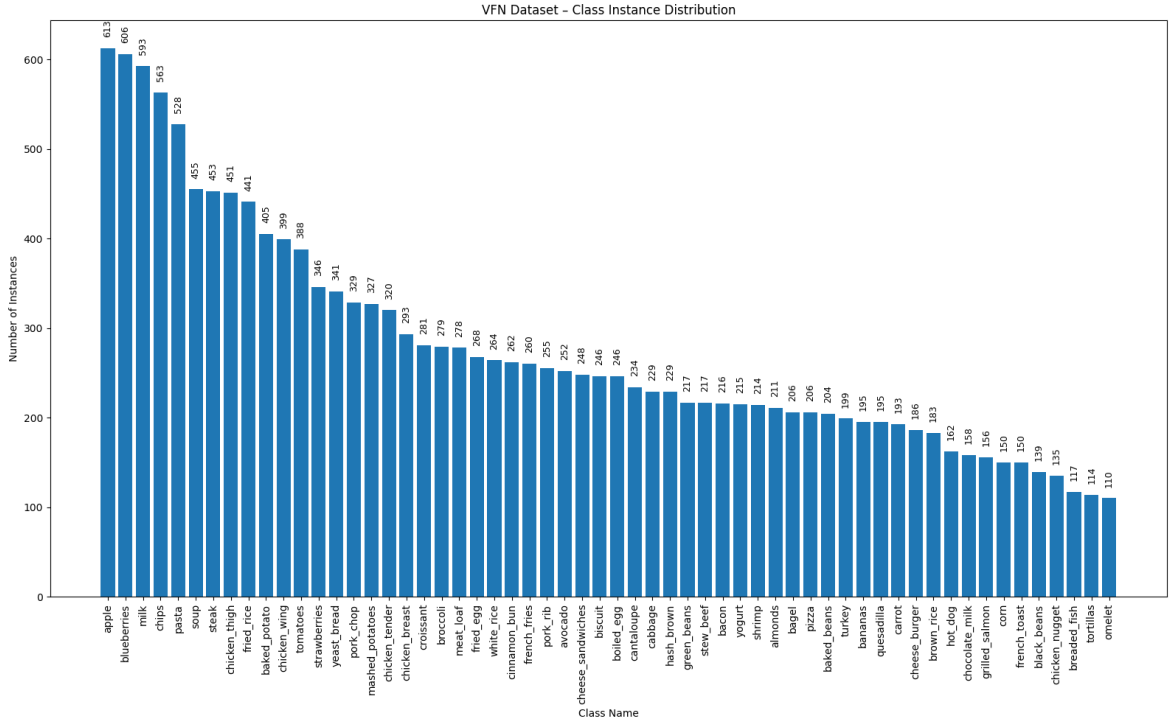


Figure 16: Class distribution for processed VFN dataset.

After processing, we trained the model for 100 epochs on the remaining images. The results of the training are shown in Figure 17. The confusion matrix for the expected vs. predicted ingredients is shown in Figure 18. Relevant metrics are listed in Table 2. A sample batch with ground truth and predicted labels are show in Figures 19 and 20 respectively.

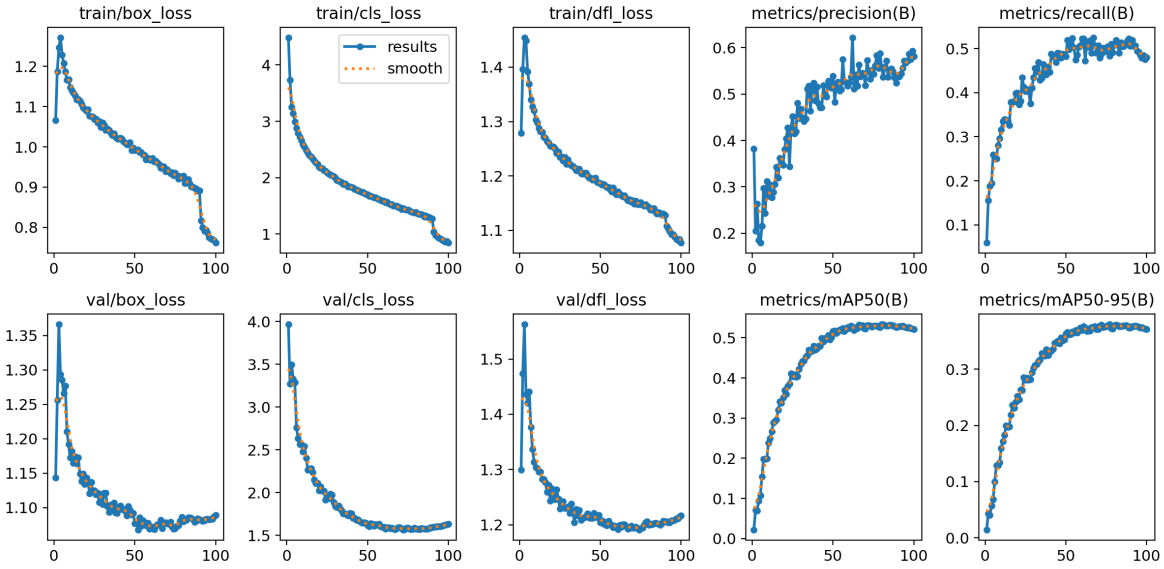


Figure 17: Results after 100 epochs of training on VFN.

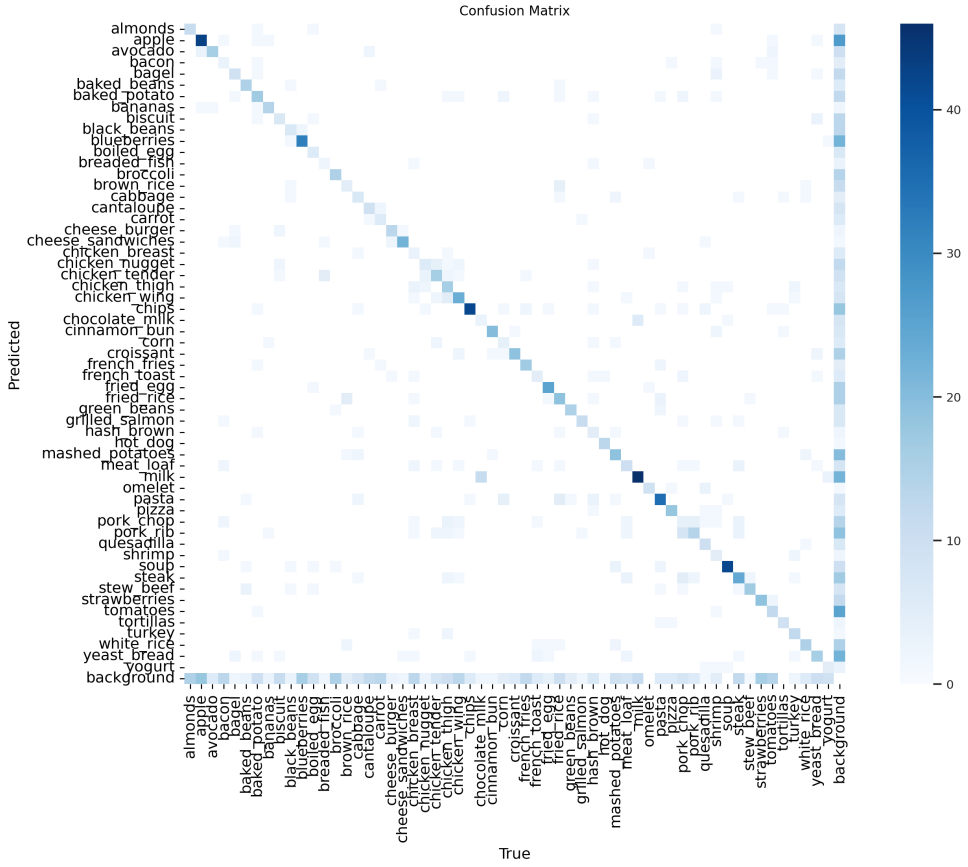


Figure 18: Ingredient detection confusion matrix.

Metric	Value at Epoch 100
Precision	58%
Recall	48%
mAP50	52%

Table 2: Training performance at epoch 100.

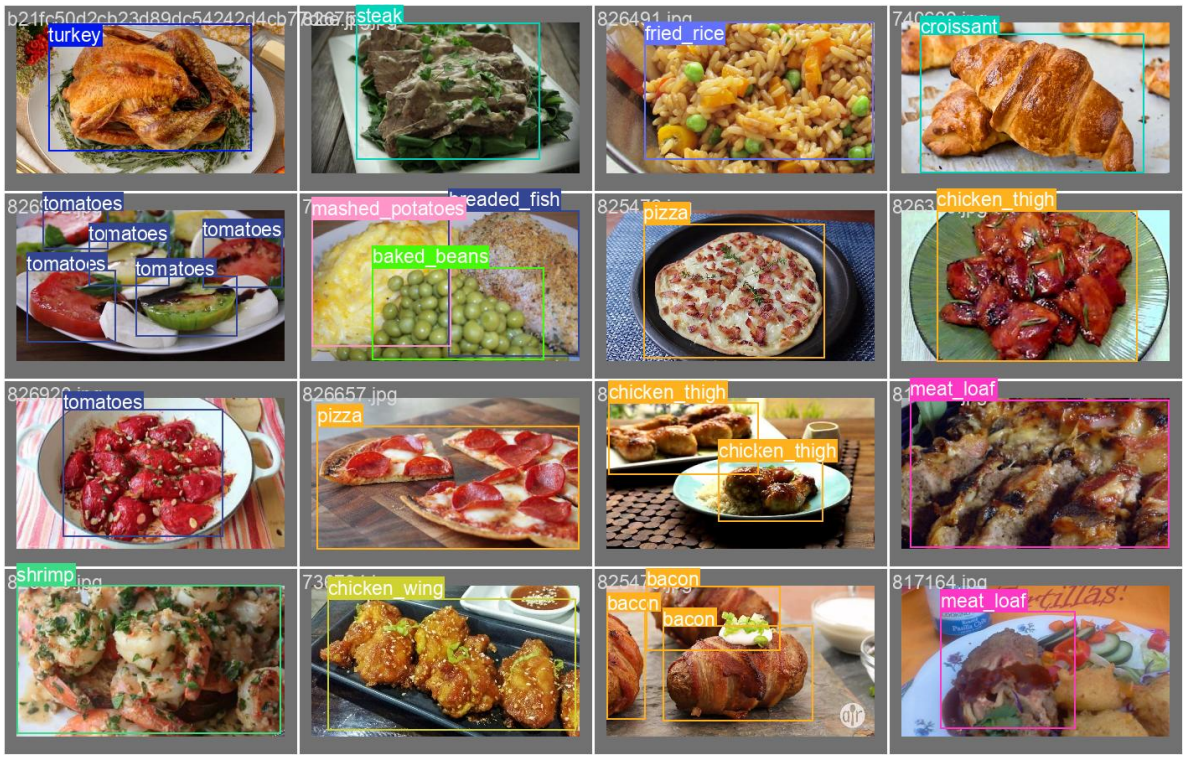


Figure 19: VFN sample batch ground truth.

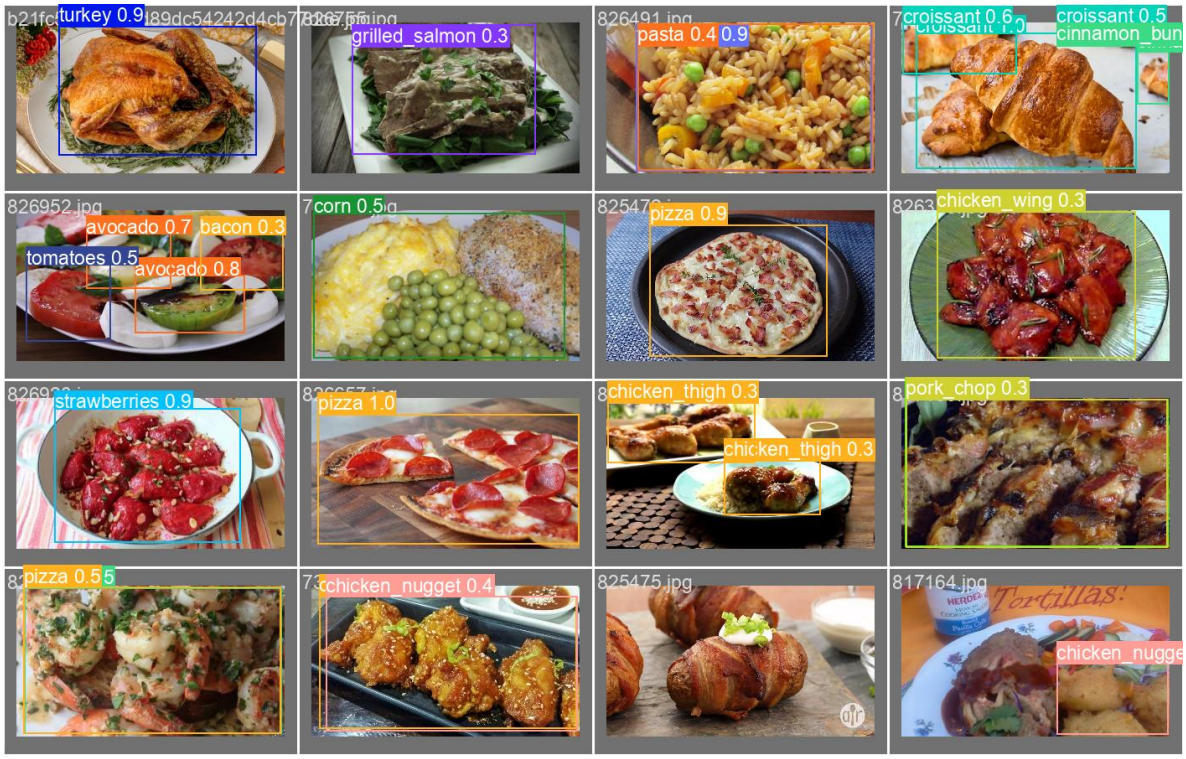


Figure 20: VFN sample batch predicted labels.

Overall, the model achieved moderate performance, and the results highlight some of the challenges involved with the task. As evidenced in Figure 16, the class distribution is unbalanced. Many ingredients appear only a few times, limiting the ability of the model to generalize. This is only further exacerbated by the domain shift from VFN to HD-EPIC. In the VFN dataset, most images contain food items that are large relative to the total image size. Additionally, most images contain only one class instance. In the HD-EPIC dataset, the images have multiple food items that are much smaller relative to the frame size, and they are subjected to a variety of environments and lighting conditions. This makes it difficult to run an inference on the HD-EPIC data, as the model was not trained in the same domain. We analyzed the performance of the model as compared to the number of instances of each class, and, as seen in Figure 21, found that as classes increase in instance count, their precision becomes more consistent. With additional data, class balancing, and domain-specific augmentations, we expect performance could improve significantly.

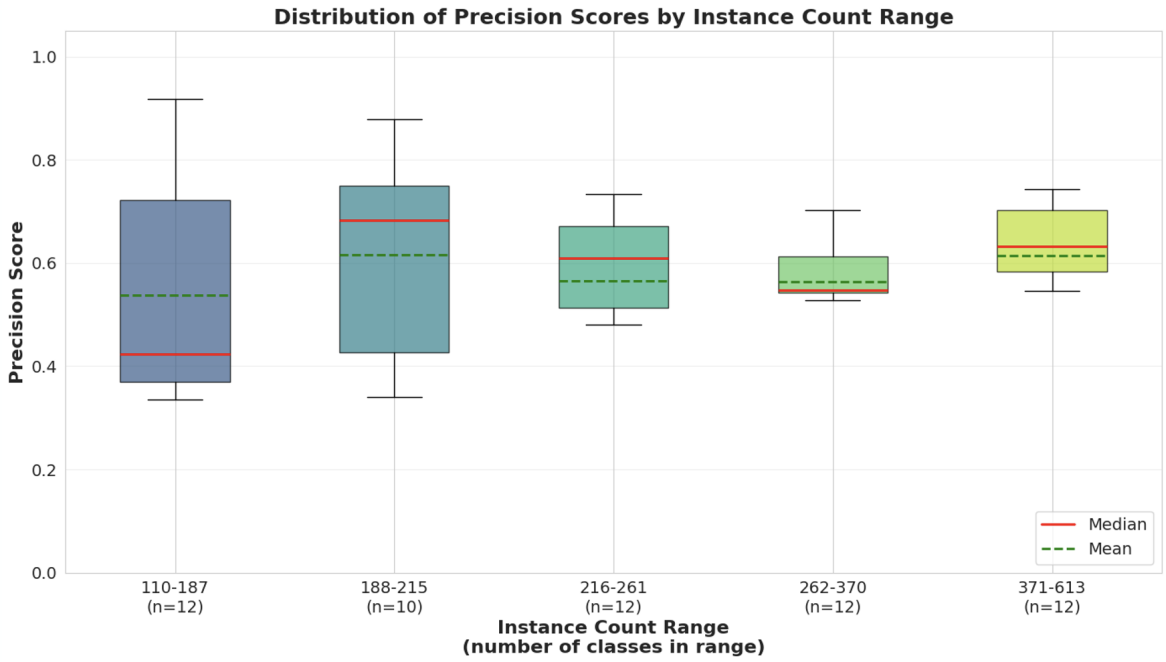


Figure 21: Precision vs. Instances Plot for Classes – Grouped by Instances

8.2 Preparation Tool Detection

For the preparation tool detection, similarly to the ingredient detection, we used the pretrained YOLOv8 model from Ultralytics as a starting point and fine tuned it. We used the preparation tools from the HD-EPIC dataset and trained on those frames where they were present, with a total of 23 classes. The model was trained for 100 epochs, with results of the training and confusion matrix for the expected vs. predicted preparation tools shown in Figures 22 and 23, respectively. Relevant metrics are listed in Table 3, and a sample batch from validation is shown in Figure 24.

Metric	Value at Epoch 100
Precision	59%
Recall	56%
mAP50	53%

Table 3: Preparation Tool Training Performance at epoch 100

This model achieved a similar performance to the ingredient detection model. As we trained on HD-EPIC, there was no issue with the domain shift we saw in ingredient

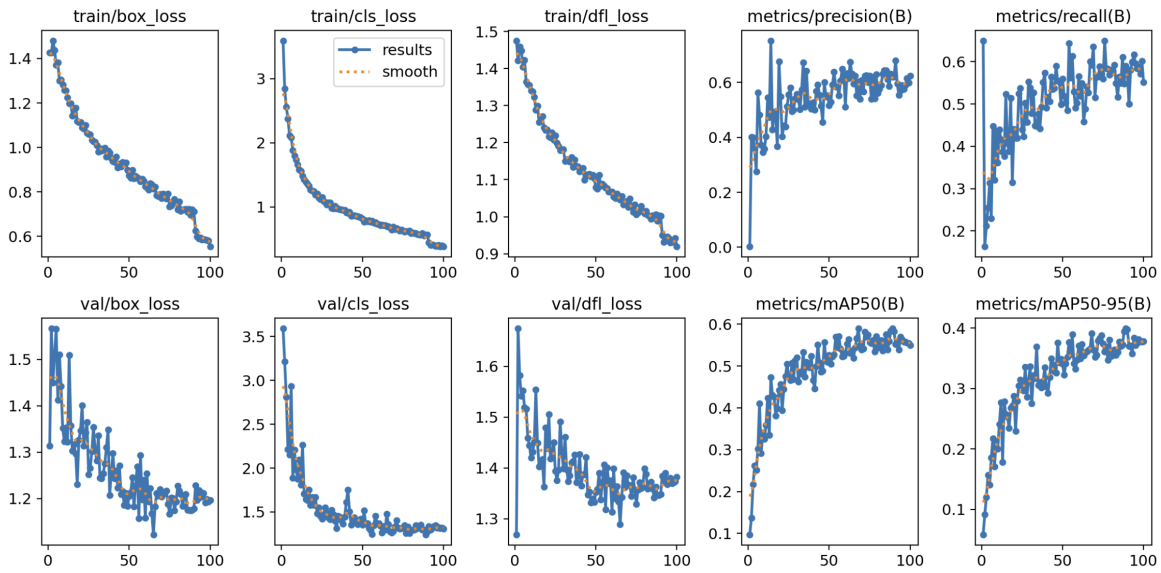


Figure 22: Preparation Tool Results after 100 epochs of training on HD-EPIC

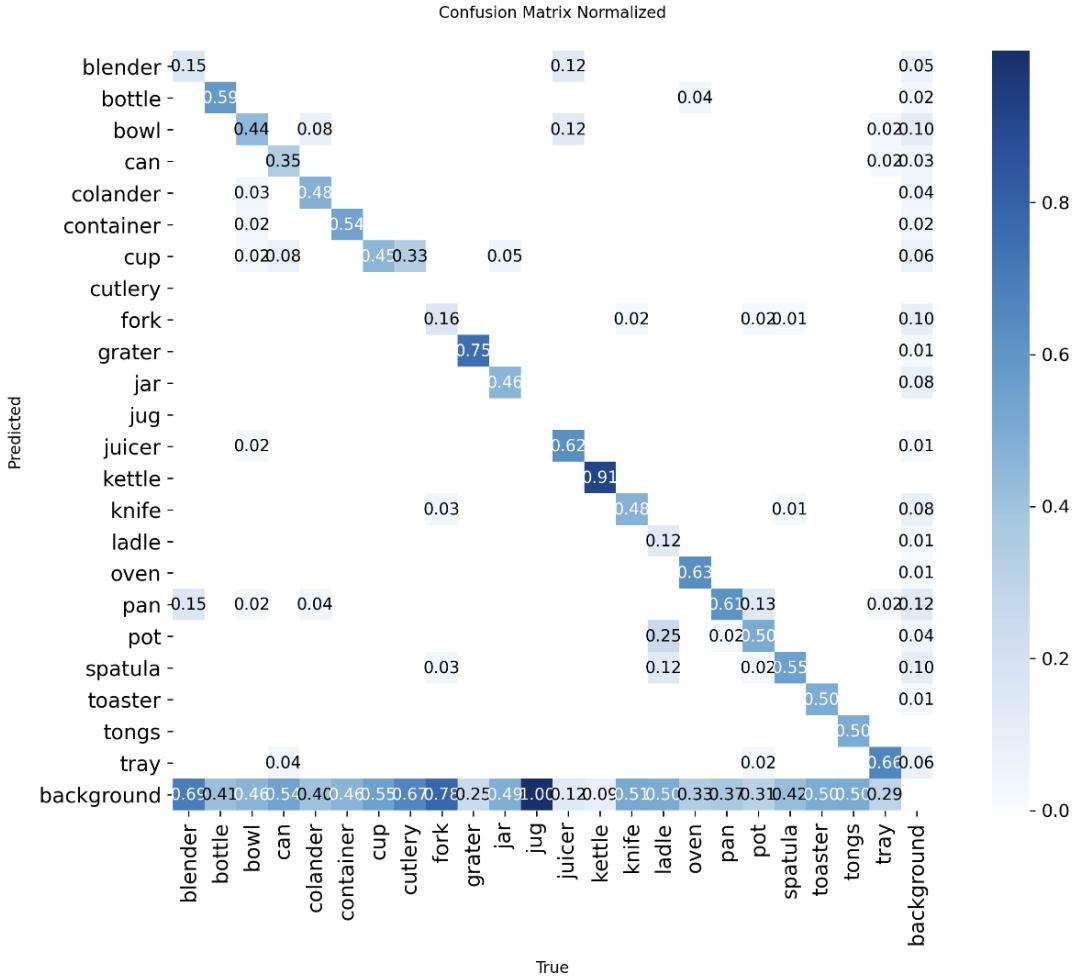


Figure 23: Preparation Tool Confusion Matrix

Figure 24: Preparation Tool Sample Batch Predicted Labels

detection, so our real-time inference of preparation tools was more consistent.

8.3 Recipe Mapping

Analyzing whether our recipe mapping works proved difficult. There was no consistent way to numerically analyze how correct our prediction was, as there are many similar recipes, and our process is based on probability matching. Another issue in preventing a complete analysis was the structure of many HD-EPIC videos. The majority of videos captured the participant cooking more than one recipe at the same time, resulting in more than hour-long videos. This prevented us from analyzing the entire video to extract one recipe, as it would be tainted by the others being cooked. Another portion of the videos, the participant prepared "recipes" which were not in RecipeNLG, such as cereal or orange juice. The remaining videos were few, however, we managed to find one cooking Cacio e Pepe and mapped it to RecipeNLG. This yielded the top matched recipe being Macaroni and Cheese with a score of 0.360, as seen in Table 4. This also

Top Recipe Matches	Recipe	Score	Matched Ingredient	Matched Tools
1	Macaroni And Cheese	0.360	cheese, pasta, water	pot
2	Macaroni	0.338	cheese, pasta, water	none
3	Kindergarten Spaghetti	0.332	meat, pasta	none

Table 4: Top Recipe Matches for Cacio e Pepe

	Original Recipe	Matched Recipe
Total kcal	756	7353
Servings	1	10
kcal / serving	756	735.3

Table 5: Calories Comparison Between Original and Matched Recipe

yielded the calorie values found in Table 5, which, despite not being the same recipe, were similar between the two. This is likely due to the fact that Cacio e Pepe is different from Macaroni And Cheese mainly in the addition of black peppercorns, which we were not detecting. Overall, this proof of concept could have real-world applicability if there was more concrete canonicalization.

9 Evaluation and Future Work

9.1 Evaluation

We evaluated our system both qualitatively and quantitatively across the different components of our pipeline. Overall, the system demonstrated strengths in hand detection and detection region localization. However, the system also has several areas where performance was limited by dataset constraints and other challenges.

9.1.1 Strengths

Our system was able to achieve fairly consistent object detection for both cookware and ingredients using our YOLO models. Although ingredient detection accuracy varied across classes, performance for distinct ingredients was reliable. Similarly, MediaPipe Hands provided stable hand tracking, which allowed us to identify ROIs for ingredient detection.

9.1.2 Weaknesses

The biggest limitations of our system stem from insufficient training data. Our ingredient detection model can only predict 58 classes from the VFN dataset, which limits our ability to detect a wide variety of ingredients. Domain shift also proved to be a hindrance to model performance, as translating from the VFN dataset to the HD-EPIC domain decreased the accuracy of predictions. Additionally, we lack the ability to estimate ingredient quantities, which leads to an incomplete analysis of nutrition information.

9.2 Future Work

Several extensions and improvements to our system could improve the accuracy of our ingredient predictions and calorie estimations.

9.2.1 Ingredient Quantity Estimation

Our current pipeline assumes a unit quantity for every detected ingredient, which limits the accuracy of calorie estimation. To improve this, we can investigate methods of quantity estimation:

- Using depth cues or relative size analysis to estimate the volume/scale of a food. For example, we could use the average size of a human hand as a reference for scale. From there, we could estimate the relative size of food items based on bounding box size.
- For certain ingredients we could detect when they are poured or scooped. Using the amount of time spent scooping or pouring, we could estimate the quantity of the ingredient. However, this also depends on the relative rate of pouring/scooping.

9.2.2 Dataset Expansion and Domain Adaptation

Ingredient detection performance could be enhanced significantly through multiple improvements:

- Adding more training data and ingredient classes. We have already experimented with this using supplemental data from the FoodSeg dataset. With proper preprocessing, we believe that this could make our ingredient detector more accurate and generalizable.
- Using data augmentation techniques to alter the scaling, lighting, and relative size of ingredients could help reduce the issues associated with domain shift between our training and testing data.

10 Conclusion

In this project, we developed a system for estimating nutritional information from egocentric cooking videos. Our three stage pipeline: environment scanning, ingredient detection, and nutrition estimation, integrates multiple computer vision components to identify ingredients, track user interactions, and infer recipe-level nutritional changes.

Using MediaPipe Hands and preparation tool detection, we localized meaningful ingredient prediction regions within each frame, significantly reducing noise and improving the reliability of ingredient detections. Our YOLO-based ingredient detection model, fine tuned on the VFN dataset, achieved moderate performance despite challenges such as class imbalance and domain shift. Our nutrition estimation module incorporated ingredient filtering, FNDDS calorie mapping, and recipe reconstruction to generate calorie estimates. However, the system currently assumes unit quantity additions, leaving quantity estimation as an important area for future development.

Overall, this project demonstrates the feasibility of using computer vision to automatically infer nutritional information from real-world cooking videos. While there is still substantial room for improvement, particularly in ingredient detection accuracy and quantity estimation, our system provides a strong foundation for future work in automated dietary analysis and nutrition tracking.

References

[1] Michał Bień, Michał Gilski, Martyna Maciejewska, Wojciech Taisner, Dawid Wisniewski, and Agnieszka Lawrynowicz. RecipeNLG: A cooking recipes dataset for semi-structured text generation. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 22–28, Dublin, Ireland, December 2020. Association for Computational Linguistics.

[2] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8, 2023.

[3] Runyu Mao, Jiangpeng He, Zeman Shao, Sri Kalyan Yarlagadda, and Fengqing Zhu. Visual aware hierarchy based food recognition. *ICPR Workshops*, 2020.

[4] Toby Perrett, Ahmad Darkhalil, Saptarshi Sinha, Omar Emara, Sam Pollard, Kranti Parida, Kaiting Liu, Prajwal Gatti, Siddhant Bansal, Kevin Flanagan, Jacob Chalk, Zhifan Zhu, Rhodri Guerrier, Fahd Abdelazim, Bin Zhu, Davide Moltisanti, Michael Wray, Hazel Doughty, and Dima Damen. Hd-epic: A highly-detailed egocentric video dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2025.

[5] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[6] U.S. Department of Agriculture, Agricultural Research Service. USDA Food and Nutrient Database for Dietary Studies 2021-2023. Food Surveys Research Group Home Page, 2021-2023.

[7] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking, 2020.

11 Appendix

11.1 William Tao

My contributions to the team and project include organizing team meetings, researching CNNs and YOLO, implementing my custom and hybrid YOLO and RCNN model in hopes of better performance, and calorie mappings with the FNDDS dataset. In the report wrote the abstract, introduction, YOLO, and calorie mapping sections. Most of my contributions in GOGS include the custom YOLO model, which never ended up being used because of poor performance, and the calorie mapping logic, a core component in our nutrition estimation tool. In my calorie mapping script, I process through the FNDDS Excel sheet and model classifications, making sure they can correctly map to the corresponding calorie values. I also made a food aggregation list to generalize common classifications from our model, removing potential double classifications of the same object. Overall, I believe I was a great team member in this group and made positive contributions to the project.

11.2 Piotr Nabrzyski

This semester, I contributed to this project in many ways. At the beginning, when we first took on this project, I helped conceptualize a plethora of the potential roadblocks we would be met with over the course of working with the HD-EPIC dataset, which helped us foresee and prevent some of these when developing our project. I developed methods to extract ingredients and preparation tools from the HD-EPIC dataset by sifting through the categories of noun classes for all words that would be applicable to either category. I kept track of our timeline throughout the semester. I helped develop our current working YOLO model, as well as trying to build off of it with combining Faster R-CNN, though this did not improve our performance as outlined in Section 4.3. I also adapted our YOLO model to detect cookware and preparation tools, as well as developed the RecipeNLG mapping with the ingredient and preparation tool classes. In this report, I wrote the sections on YOLO loss, Recipe Mapping, R-CNN, RecipeNLG Dataset, Preparation Tool Detection Results, and Recipe Mapping Results. I also contributed to editing the revising our final report.

11.3 Donny Weintz

Throughout the project, I contributed to group communication, planning, and coordination across all system components. My primary technical focus was developing a fast and accurate ingredient detection model. I designed and refined several data extraction pipelines, working with datasets such as HD-EPIC, VFN, FoodSeg, and multiple Hugging Face datasets to prepare clean and usable training data. The results of my best performing model, trained on the processed VFN dataset, is detailed in Section 8. I also integrated MediaPipe hand tracking into the system to isolate ingredient prediction regions, which helped reduce false positives and improve overall detection performance. In addition to detection, I contributed to the design and implementation of the final nutrition estimation pipeline, ensuring that ingredient detection, calorie mapping, and recipe matching all functioned cohesively. The pipeline is described in detail in Section 7. Finally, I created the project demo system, integrating all modules into a video demonstration that showcases the end-to-end functionality of our system. My contributions to this report include Sections 4.2, 7, 8.1, 9, and 10. Overall, I learned a lot through this project and believe I made significant contributions to its success.

11.4 Benjamin Nguyen

Over the project timeline, I documented project functions for the understanding of those without computer vision background, developed a frame preprocessing module for extreme conditions such as darkness, and worked alongside all teammates during team meetings as well as lab sessions wherever able. Due to the limited amount of frames that contained extreme conditions due to the optimization of the data collected, the preprocessing module wasn't needed nor implemented for the sake of simplicity and left in GOGS. Furthermore, I documented the dataset descriptions within the report in order to establish what we're using to accomplish this project. The contribution of my work to the project itself is weak, but I have been able to learn a significant amount from my group that I believe wouldn't have stuck with me in a standard class setting in data implementation, computer vision, and general python. I am eager to use this defining experience for future applications of coding. Thank you all.