SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.035 Computer Vision
1D Project Report

Table of Contents

1. Introduction

Github Link: https://github.com/hengjiaming/computer_vision_app.git

1.1 Background

Singapore faces a growing health crisis, with obesity rates rising from 21.1% in 1992 to 36.2% in 2017, alongside increasing cases of diabetes, high blood pressure, and high cholesterol. These conditions strain public health and impose significant economic costs, such as the $1 billion annually projected for diabetes-related healthcare and productivity losses. If unchecked, these issues can lead to severe complications like cardiovascular disease and kidney failure.

Government initiatives like the National Steps Challenge and Eat, Drink, Shop Healthy Challenge have encouraged healthier habits, but their reliance on extrinsic rewards limits sustained behavior change. Misconceptions about healthy eating persist, further complicating efforts to improve public health.

This project aims to address these gaps by developing a system that uses computer vision to provide real-time nutritional insights from food images. By educating users and offering personalized feedback, the solution empowers individuals to make informed dietary choices and sustain healthier behaviors.

1.2 Problem Framing

Opportunities for Innovation

Growing awareness of health and nutrition, alongside government initiatives to promote healthier lifestyles, has created demand for effective diet management tools. However, current dietary tracking systems often fall short due to manual logging, unintuitive interfaces, and lack of personalization, discouraging sustained use.

Leveraging advanced computer vision technology for accurate food recognition and portion estimation can address these gaps. A user-friendly system providing personalized feedback can empower individuals to make informed dietary choices, complement existing health tools, and support overall well-being.

Stakeholder Analysis

General Consumers: Everyday users need a simple, reliable system with minimal manual input and accurate feedback to encourage sustained engagement and foster healthier eating habits.

Fitness Enthusiasts and Athletes: This group values precision in tracking nutrient intake and benefits from features like meal suggestions, alerts for nutritional targets, and seamless

integration with fitness tools.

Patients and Healthcare Providers: Individuals with dietary restrictions and healthcare professionals require scientifically validated data, privacy compliance, and integration with healthcare systems for effective monitoring and guidance.

1.3 UN SDG Contributions

Our project contributes to the **UN's Sustainable Development Goals** by addressing key global challenges.

It directly aligns with **SDG 3: Good Health and Well-being**, which aims to ensure healthy lives and promote well-being for all. By leveraging computer vision to provide real-time nutritional insights from images of food dishes, our project empowers users to make informed dietary choices and promotes preventive healthcare. By addressing the rising prevalence of diet-related diseases in Singapore, the system complements national health initiatives and fosters healthier behaviors through personalized feedback.

Additionally, it supports **SDG 9: Industry, Innovation, and Infrastructure**, as our project fosters innovation by integrating advanced AI models like EfficientNet for food classification, regression for nutrient breakdown, and depth estimation for portion sizing, showcasing how technology can advance health solutions.

Furthermore, it aligns with **SDG 12: Responsible Consumption and Production**, which promotes sustainable consumption and waste reduction. By raising awareness of portion sizes and nutrient intake, our project helps individuals consume mindfully, reducing overconsumption and food wastage.

Ultimately, our work contributes to a healthier, more informed population, bridging gaps in dietary management and health awareness through accessible, cutting-edge technology.

1.4 Data

During our research on the topic we managed to source a dataset called Nutrition-5k which is a dataset that contains visual and nutritional data of around 5,006 plates of food captured from Google cafeterias.(https://github.com/google-research-datasets/Nutrition5k)
The Data Consists of;

4 Rotating Side-Angle Videos: Captured from 4 Raspberry Pi cameras at varying angles, allowing 360-degree views of each dish. Frames can be extracted from the videos using provided scripts.

Overhead RGB-D Images: Contains RGB, raw depth, and colorized depth images for each dish, showing depth variations with visual encoding (blue for close, red for far).

Detailed Ingredient Information:Each dish includes a list of ingredients with per-ingredient mass, total dish mass, calories, and macronutrients (fat, protein, carbohydrates).

The Ingredient Metadata is provided as a CSV file with ingredients IDs, names and nutritional information per gram (based on USDA data). The Dish metadata is in a separate CSV file for each cafeteria and contains information like dish_id, total_calories, total_mass as well as per-ingredient nutritional values. Also provided are Pre-structured splits of data via dish IDs into training and testing splits to ensure no overlap between splits as well as an evaluation script for calculating absolute and percentage mean average error on nutritional predictions, allowing comparison with results in the Nutrition 5k paper.

Considering Nutrition-5k as our dataset of use, we were faced with the challenge of handling the video data. It is crucial to note that preprocessing of the data in the Nutrition-5k dataset for frame extraction required significant compute resources. Hence, it was vital for us to source for the alternative image datasets to overcome the issue of lack of resources. We ended up using an open-source dataset of extracted frames from the 4 Rotating Side-Angle Videos of the Nutrition-5k dataset.

2. Methodology

We began with identifying some key objectives that we needed to achieve namely;
To be able to recognise food items and portion sizes and integrate with nutrition databases(e.g. USDA database)
To estimate caloric content and macronutrient distribution(fat, protein, carbohydrates) per dish in real-time.
As well as having a basic interface to present nutritional information clearly.
We split the taskings into different sections which we will go through later which are Food identification, Depth portion size estimation and our StreamLit section.

2.1 Food Identification

The food identification portion began in our minds as an object classification problem but quickly pivoted to an image classification problem as we realised that Nutrition-5k lacked bounding box information for us to train for object detection models such as YOLOv5.

2.1.1 Model Exploration

We began this section of training the food identification model by experimenting with different models to choose which was most suitable for fine-tuning.

We started by earmarking several models for testing namely EfficientNetV2_S, ResNet 101, ResNet 50 v2. We began training with the initially selected models (YOLOv5, YOLOv8, Faster RCNN with ResNet50 or ResNet101 backbone) on the Nutrition-5k dataset for food recognition, with a focus on setting a strong baseline with default hyperparameters.

Experimentation / Initial Exploration

Independent variables:

- Base model architecture
- EfficientNetV2_S, ResNet 101, ResNet 50 v2.
- Hyperparameters (fine-tuning)

Dependent variables:

- Accuracy
- Precision
- Recall
- F-Score

Testing Details

We standardised our testing details to the following parameters, prioritising speed over coverage in our exploration phase.

Epochs: 5
Dataset size: 20%
Batch Size: 8
Dropout: none
Regularization: none
Learning rate: 1e-3
Optimizer: Adam
Multi-label Classification Loss: Binary Cross Entropy

2.1.2 Exploration Results

Model	Task	Precision	Classification Accuracy	Recall
EfficientNetV2_S	Image Classification	0.6448	0.3710	0.6041
ResNet 101	Image Classification	0.26134	0.09434	0.18939

ResNet 50 v2	Image Classification	0.6634	0.2444	0.0457

After experimenting with the different models, we decided that since EfficientNetV2 had the best classification accuracy and recall scores, we decided that EfficientNetV2 would be the model we would be moving forward with.

2.1.3 Hyperparameter Tuning

Taking the EfficientNetV2 model we performed Hyper-parameter tuning using scikit-learn's GridSearchCV function.

The parameter grid was defined as:
Learning rate: 1e-5, 1e-4, 1e-3
Batch size: 4, 6, 8
Optimizer class: Adam optimizer, Stochastic Gradient Descent(SGD) Optimizer

The Optimiser ran through a total of 54 different variations of fits resulting in the best params estimation of;
batch_size: 8,
learning rate: 1e-3,
optimizer class : Adam optimizer

2.1.4 Full Training

After completing finding the best hyperparameters, we trained the model on the full dataset with 20 epochs.

Model	Task	Precision	Classification Accuracy	Recall
EfficientNetV2_S	Image Classification	0.6448	0.3710	0.6041

Final Model Evaluation scores:
Accuracy: 0.603
Precision: 0.786
Recall: 0.766

2.2 Nutritional Estimation

2.2.1 Regression Model

The regression model was built using EfficientNetV2-S as the base feature extractor. The classification head was removed, retaining the pre-trained feature extraction layers to output a 1280-dimensional feature vector. On top of this, four regression branches were added, each predicting one nutritional target: protein, fat, carbohydrates, and mass. Each branch consisted of linear layers with ReLU activation, dropout for regularization, and a single continuous output node.

Data Preparation and Normalization
The dataset contained images paired with ground truth values for each nutritional target. These values were normalized using MinMaxScaler to scale them between 0 and 1. The scalers were fitted on the training data and saved for inverse transformation during inference. Images were preprocessed through resizing and normalization to match the input requirements of EfficientNet.

Training Process
The model was trained using Mean Squared Error (MSE) as the loss function for each target. The total loss was computed as the sum of individual losses. The Adam optimizer was used for weight updates. During training, each image passed through the EfficientNet feature extractor, and the resulting features were fed into the regression branches for predictions. The model's weights were adjusted to minimize the total loss.

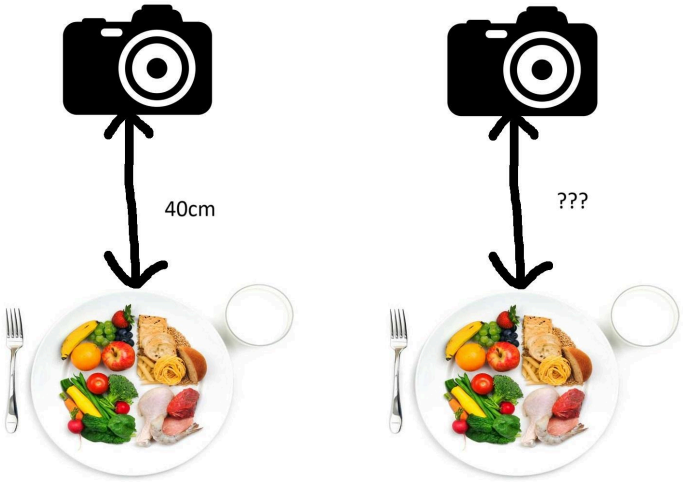
Training Hyperparameters
Epochs: 10
Batch Size: 16
Dropout Rate: 0.3
Learning rate: 1e-4
Optimizer: Adam
Loss Function: Mean Squared Error (MSE)

Training Results:
Protein MAE: 0.037357
Fat MAE: 0.047500
Carbs MAE: 0.030630
Mass MAE: 0.049011

2.3 Depth Portion Size estimation

Another issue we had with using the Nutrition 5K model stems from its training set, which consists of videos taken 40 cm away from the plate of food. However, on our end, when users upload their

image, it would be impractical to expect them to take their images 40cm away from the plate of food. Thus, we should expect our user's images to be from different distances away from the plate and thus adapt our solutions to fit this criteria, utilizing some sort of scaling to do so.



Nutrition5k User's Input

2.3.1 Reference Object Solution

For our first iteration, we attempted to introduce scaling through a reference object.

For the reference object model, we first have to start off with selecting a reference object with a **known size.** In our case, we selected a coin to be a reference object due to its similarity in shape with a dish. In every image that we wish to process, we need to ensure that our reference object appears in the image. The image processing procedure is as follows:

1) After uploading the image, the model performs object detection, to detect the plate and the coin, drawing bounding boxes around both.
2) The model will then output the dimensions of the bounding boxes, which will be in image pixels.
3) We can then calculate the size of the plate using the following formula:
    a) $size\ of\ user\ plate\ =\ \frac{pixel\ size\ of\ plate}{pixel\ size\ of\ coin}\ *\ known\ size\ of\ reference\ object$
4) From the get go, we took the nutritional content output of the Nutrition5K dataset to be based on a reference plate of size 10.5 inches.
5) Using the assumed plate size, we can calculate the scaled nutritional value using:
    a) $Final\ nutritional\ output\ =\ \frac{size\ of\ user\ plate}{size\ of\ reference\ plate}\ *\ nutrition\ output\ of\ reference\ plate$

Limitations

For this depth estimation strategy to work, the user has to be in possession of a particular coin wherever they want to use the app to track their nutritional requirements. This would be

inconvenient for many users, since we live in a cashless society where most prefer to pay through digital services like PayLah/PayNow or credit card, all of which can be utilized using their mobile devices. Furthermore, the assumption of the reference plate is merely an estimation based on common plate sizes and may not be the truth and accuracy is limited in poor lighting conditions.

2.3.2 Final Solution

Recognizing the limitations of the reference object solution, we pivoted to leverage depth estimation as a scalable and user-friendly alternative. This approach removes the dependency on a physical reference object by employing advanced computer vision techniques to determine the metric dimensions of objects directly from a single image.

Metric3D, a depth estimation model was used. Metric3D performs zero-shot metric depth estimation and surface normal prediction, generating a depth map. This depth map can then be used in conjunction with camera intrinsics (focal length, sensor size) along with image width and height to assist us in scaling our nutritional content, providing the final output result. The process is as follows:

1) User uploads an image of a plate of food into the model to perform depth estimation.
2) The metadata in the image is extracted to retrieve camera intrinsics such as focal length and sensor width.
3) Model outputs the depth estimation map and depth estimation by considering the camera intrinsics and image dimensions.
4) With this depth estimation, we receive the distance of the camera from the plate.
5) We can utilize the fact that Nutrition-5k works with training data 40cm away from the plate, to come up with scaling factor:
   $$\frac{distance\ estimation\ in\ depth\ estimation}{distance\ estimation\ in\ Nutrition5k} = \frac{distance\ estimation\ in\ depth\ estimation}{40cm}$$
6) We can then multiply this scaling factor with our nutrition output from Nutrition-5k to get out final nutrition output:
   $$Final\ nutrition\ output\ = \frac{distance\ estimation\ in\ depth\ estimation}{40cm}\ *\ nutrition\ output\ from\ Nutrition5k$$

Limitation

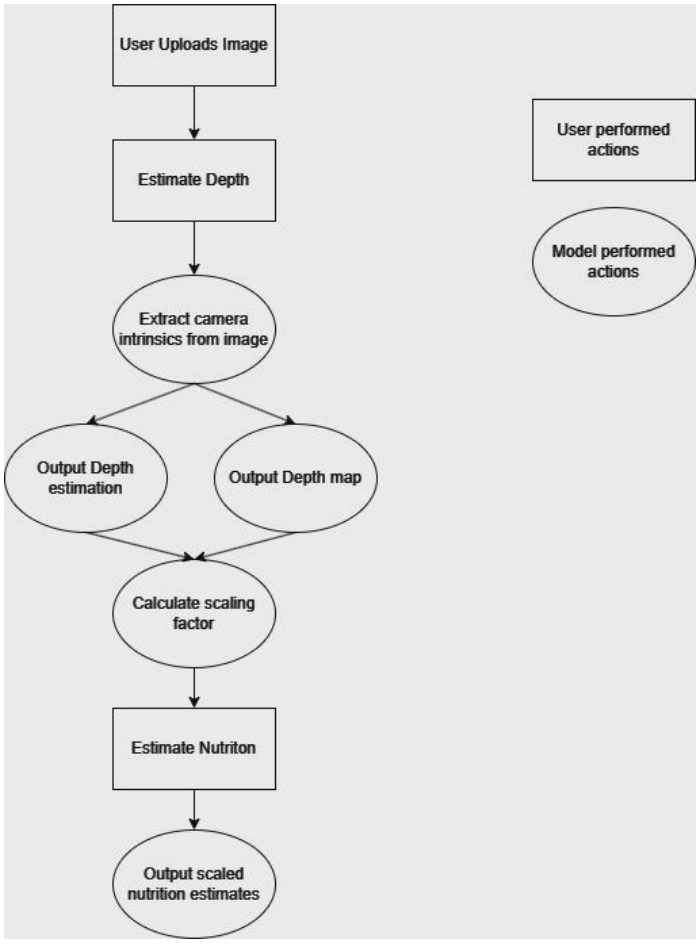
There are instances where the image uploaded by users do not contain any metadata, thus we would not be able to perform the scaling properly. However, a quick fix would be to allow the user to specify the model of the device he is currently using if we are not able to extract the necessary camera intrinsics.

2.4 StreamLit Framework

We utilize the Streamlit framework (streamlit.io) to create an intuitive, user-friendly interface and integrate the different components of our project. The interface allows users to upload an image of a food dish, which is processed in real-time to provide nutritional insights such as protein, fat, and carbohydrate content using our EfficientNet model for classification and regression for

nutrient breakdown. Additionally, the depth estimation output is displayed to estimate portion size.

The following diagram depicts the end to end flow of the application:



2.5 Connecting the Components

The integration of multiple machine learning models and components was a critical aspect of this project, combining depth estimation, regression-based nutritional analysis, and a user-facing application.

Integration of Depth and Nutritional Models

The scaling factor from the depth model was applied to the regression model's predictions, linking depth information with nutritional estimates. This ensured predictions were accurate regardless of image distance or camera setup.

Streamlit App

A user-friendly Streamlit app was built for interactive input and output where users can upload images for analysis. Depth maps and scaled nutritional predictions are displayed in real-time.

For unknown cameras, users can select their device, and predefined focal lengths are used to compute scaling.

Evaluation
The evaluation script used known parameters for the Raspberry Pi camera to compute scaling factors since we knew that the Nutrition-5k data was taken using a Raspberry Pi camera.
Predictions were compared to ground truth values, and Mean Absolute Error (MAE) was calculated for accuracy.

3. Results

Final Model Evaluation Results:
Protein MAE: 4.50
Fat MAE: 3.23
Carbs MAE: 4.61
Mass MAE: 29.85

Protein (MAE: 4.50)
The model achieves a relatively low mean absolute error (MAE) for protein predictions. This suggests that the model effectively captures features correlated with protein content in the images.

Fat (MAE: 3.23)
The lowest MAE is observed for fat predictions, indicating that the model performs exceptionally well in this area. This could be attributed to the distinct visual characteristics of fatty foods, such as oils, visible fats, or textures, which are effectively learned by the regression model.

Carbohydrates (MAE: 4.61)
Carbohydrate predictions show a slightly higher error compared to protein and fat but remain within an acceptable range. This could be due to the diverse appearances of carbohydrate-rich foods such as bread, rice and noodles, which may pose a greater challenge for the model to generalize accurately.

Mass (MAE: 29.85)
The largest MAE is observed for mass predictions. This is expected, as estimating the mass of food items directly from images is inherently more challenging. Factors such as varying portion sizes, perspective distortions, and the lack of a direct relationship between visual appearance and weight contribute to this higher error. However, the integration of depth-based scaling has likely mitigated this error to some extent, as it adjusts predictions based on real-world object size.

Overall Performance

The model's performance demonstrates a moderate ability to predict nutritional content, with particularly good results for fat and protein. The slightly higher errors for carbohydrates and mass reflect the inherent challenges of these tasks but remain within acceptable bounds for practical applications. The performance may also have been hampered by the inaccuracy of the focal length used since we were not able to confirm the exact model of the Raspberry Pi camera used in the Nutrition-5k data, leading to inaccurate scaling of depth.

5. Conclusion and Future Work

One improvement we aim to implement is transitioning from multi-class classification to object detection for identifying food on a plate. The current multi-class classification approach provides the models confidence in all classes for the image provided. Object detection, on the other hand, identifies both the class and location of each food item through bounding boxes, which enables precise differentiation in complex arrangements. Furthermore, the presence of bounding boxes and location allows for more precise analysis to infer portion sizes or relative quantities of food items. Leveraging modern frameworks like YOLO or Faster R-CNN, this shift promises to improve the accuracy and versatility of our system.

Another future improvement is transitioning from metadata-based depth estimation to utilizing RGB-D data. Our current method estimates depth indirectly using metadata like focal length, which introduces limitations in precision and consistency. By adopting RGB-D data, we can directly incorporate pixel-wise depth measurements, offering richer spatial information and simplifying depth perception tasks. This change would enhance the system's ability to handle diverse conditions, such as varying lighting or texture, while improving overall accuracy and

robustness. To achieve this, we plan to source for RGB-D datasets, modify model architectures, and adopt depth-aware preprocessing techniques, ensuring a more reliable and efficient approach to depth estimation.

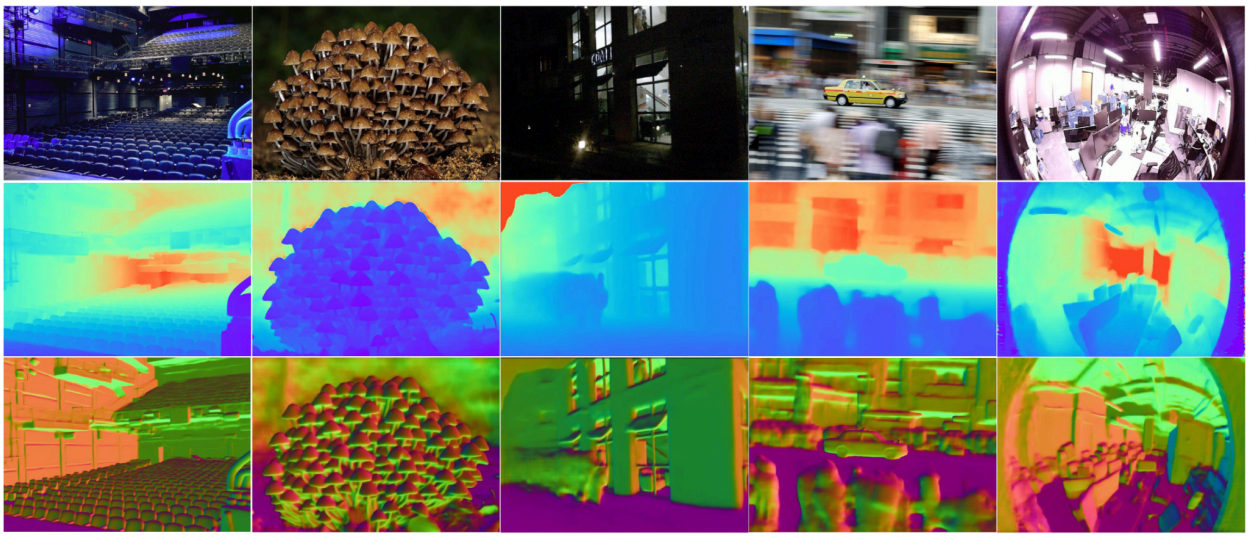
To address the limitations of relying on image metadata for focal length, we plan to implement a solution that reduces this dependency. Focal length data is often unavailable in image metadata, creating challenges for depth estimation. As a workaround, we propose identifying the device used to capture the image, as each device typically has a specific focal length. To facilitate this, we will enable users to select their mobile device through our web application. Additionally, we will scrape publicly available data to build a comprehensive mapping of mobile devices to their corresponding focal lengths. This mapped focal length can then be used as input to our model, ensuring consistent and accurate depth estimation even when metadata is missing.

Appendices

Github Link: https://github.com/hengjiaming/computer_vision_app.git

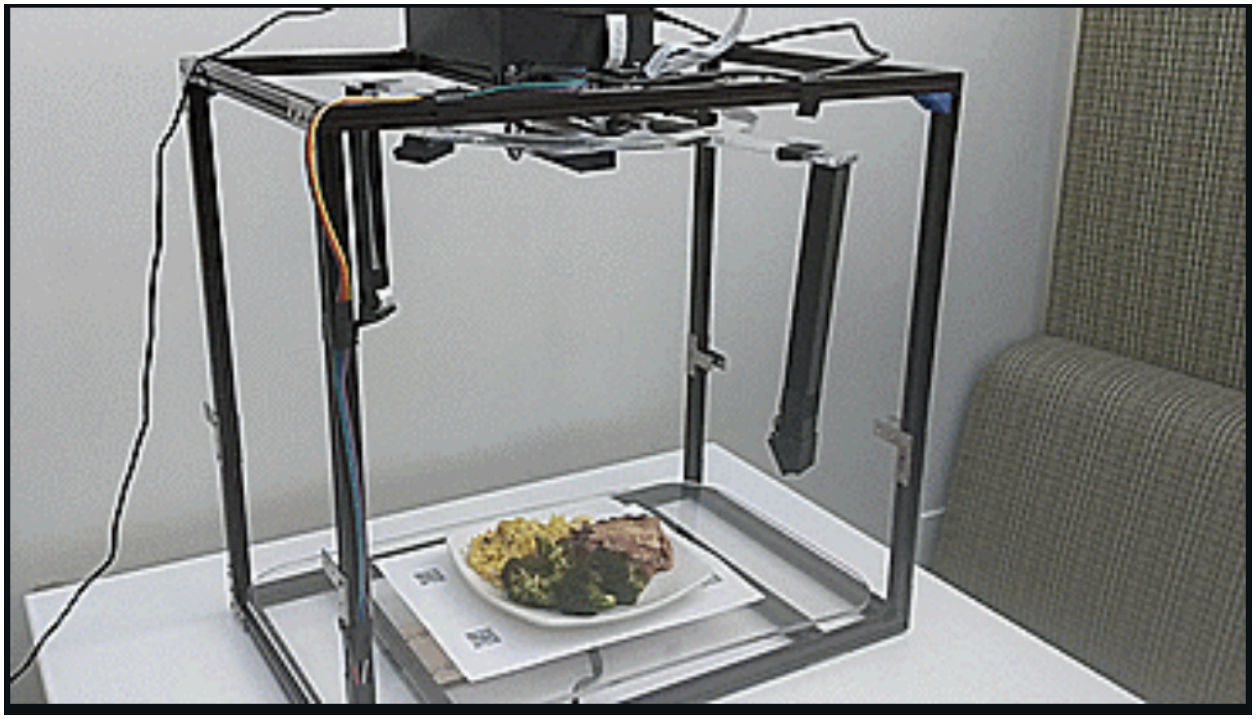
Appendix A - Dataset

Monocular depth estimation with Metric3D:



Nutrition-5k dataset output example:

Nutrition-5k setup with two cameras rotating around the dish, 40cm depth:



Appendix B - Models and Training

Multi-class Classification EfficientNet_V2_S architecture:

```python
import torch.nn as nn
import torch.optim as optim
from torchvision.models import efficientnet_v2_s

# Define the model
class MultiLabelEfficientNet(nn.Module):
    def __init__(self, num_classes):
        super(MultiLabelEfficientNet, self).__init__()
        self.base_model = efficientnet_v2_s(weights="IMAGENET1K_V1")
        self.base_model.features[0][0].stride = (1, 1)
        self.base_model.classifier[1] = nn.Linear(self.base_model.classifier[1].in_features, num_classes)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.base_model(x)
        return self.sigmoid(x)
```

Regression model architecture:

```
class EfficientNetBase(nn.Module):
    def __init__(self):
        super(EfficientNetBase, self).__init__()
        self.base_model = efficientnet_v2_s(weights=None)
        self.base_model.classifier = nn.Identity()

    def forward(self, x):
        return self.base_model(x)

class MultiTaskModel(nn.Module):
    def __init__(self, base_model):
        super(MultiTaskModel, self).__init__()
        self.base_model = base_model
        in_features = 1280
        self.protein_branch = self._create_branch(in_features)
        self.fat_branch = self._create_branch(in_features)
        self.carbs_branch = self._create_branch(in_features)
        self.mass_branch = self._create_branch(in_features)
```

Appendix C - Product Interface

Screen capture of app interface:

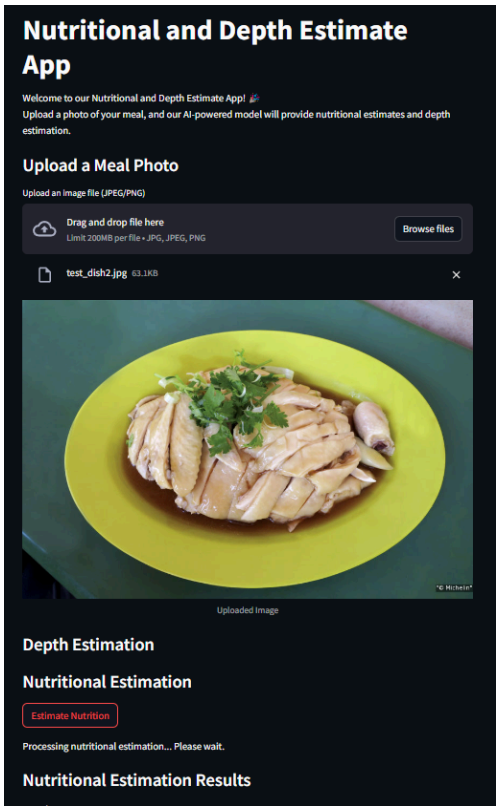
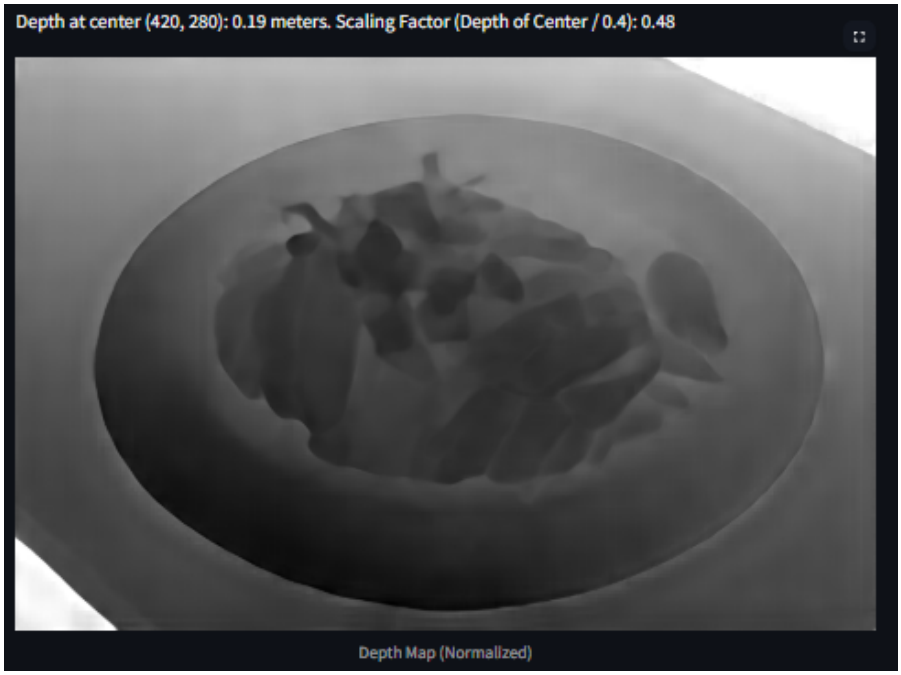


Image of depth map from depth estimation model output:



depth map

Numerical output of our final estimation: