Pop'n Food: 3D Food Model Estimation System from a Single Image

Shu Naritomi Keiji Yanai
Department of Informatics, The University of Electro-Communications, Tokyo, Japan
{naritomi-s, yanai}@mm.inf.uec.ac.jp

***Abstract*—Dietary calorie management has been an important topic in recent years, and various methods and applications on image-based food calorie estimation have been published in the multimedia community. Most of the existing methods of estimating food calorie amounts use 2D-based image recognition. However, since actual food is a 3D object, there is a limit to the accuracy of calorie estimation using 2D-based methods. Therefore, in our previous work,we proposed a method to reconstruct the 3D shape of the dish (food and plate) and a plate (without foods) from a single 2D image and estimate a more accurate food volume. Such researches on 3D reconstruction have been active recently, and it is necessary to qualitatively evaluate what kind of 3D shape has been reconstructed. However, checking a large number of 3D models reconstructed from a large number of images requires many steps and is tedious. Against this background, this demo paper introduces an application named "Pop'n Food" which has the following two functions: (1) A web application for visualizing a large number of images to check the learning results and the 3D model generated from them. (2) A web application that selects an image from a browser and generates and visualizes a 3D model in real-time. This demo system is based on our previous work named Hungry Networks. Demo video: https://youtu.be/YyIu8bL65EE**

***Index Terms*—Food, 3D reconstruction, web application, WebGL**

I. Introduction

Dietary calorie management has been an important topic in recent years, and various methods and applications on image-based food calorie estimation have been published in the multimedia community. Most of the existing methods of estimating food calorie amounts use 2D-based image recognition [1], [2], [3], [4], [5], [6]. However, since actual foods are 3D objects, the accuracy of calorie estimation using 2D-based methods is limited. 3D-based methods have been explored so far as well. Some works tried to estimate the 3D volume of foods from a single image using depth estimation [7], [8] or using a depth camera [9], [10]. These studies had restrictions such as the requirement for a depth camera and the need for food to be on a flat plate. On the other hand, in our previous work [11], we have achieved a highly accurate 3D reconstruction of a dish (food and plate) and a dish (without foods) from a single dish RGB image for more accurate volume estimation. This method allows us to maintain the consistency of the dish portion of the estimated meal and dish 3D Volume. The difference in volume between the two shapes of dish and plate reveals

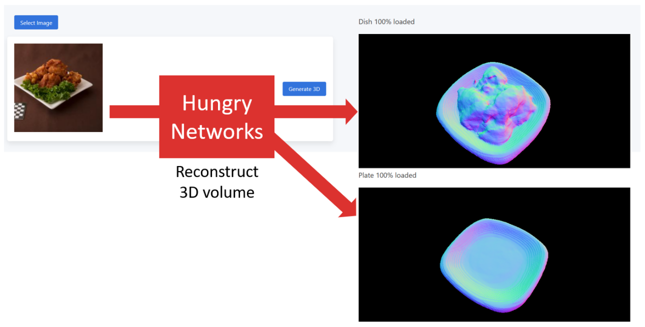

Fig. 1. User interface of "Pop'n Food". left is input image and right is 3D view.

the volume of the food area, which is useful for estimating calorie content. Again, the method proposed in the previous study generated two 3D models from a single image. So, the qualitative evaluation is a very hard task. We explain how hard it is to do. Generally, to see a 3D model, software tools for visualizing and editing 3D models such as Blender and MeshLab are used. These applications read local files, so if deep neural networks are executed on a pubic cloud or remote machine, we will need to download 3D models to the local machine. Downloading a large number of 3D models to the local machine can be very time-consuming. Next, we enter the 3D model into the 3D model visualization tool, but we have to check each time from which image the 3D model was generated. Also, the Hungry Networks proposed in our previous work is even harder because it generates two 3D models. What we want to do is see what 3D models a network produces from the images. We do not want to download or check the correspondence between 3D models and images. Therefore, we created a web application named "Pop'n Food" that displays an image list on a web browser and allows us to view two 3D models at the same time by clicking the button corresponding to the image. The UI looks like Figure 1.

This application eliminates the need to hang out in front of a computer to download a large number of models, or to have human check the correspondence between images and 3D models. The main use case of this application is to easily view the results of 3D models that have been reconstructed with the evaluation data set using a method that achieves 3D reconstruction in advance. However, there is also a need to generate and check 3D models using images other than those in the evaluation dataset, so we also created a model that generates and displays two 3D models in a real-time way using images as input from a browser.

In this paper, we first introduce our previous research, Hungry networks, which reconstructs the 3D shape of dish and plate from a single image. Next, we will explain a system named "Pop'n Food" that generates a 3D model using Hungry networks from a single image in real-time and visualizes the generated 3D model.

II. Hungry Networks

The Hungry Network is a deep neural network that reconstructs two 3D shapes of a dish (food and plate) and a plate (without foods) from a single food image. The network consists of one encoder and two decoders as shown in Figure 2. The encoder takes an image as input and extracts image features. The decoder takes the image features and the coordinates $x \in \mathbb{R}^3$ as input and outputs occupancy $o \in \mathbb{R}$. The occupancy indicates whether the coordinate $x \in \mathbb{R}^3$ is inside or outside the model. It is 1 if it is inside the model and 0 if it is outside. Of the two decoders, one is trained to output occupancy to generate a model of the dish. The other decoder is trained to output occupancy to reconstruct only the plate. The training of occupancy takes as input a non-discretized (continuous) coordinate x, so it is possible to create 3D models with infinite resolution (in practice a resolution of 128x128x128 is sufficient). From the resulting occupancy filed, a 3D mesh is generated using a marching cube [12]. The algorithm based on Occupancy Networks [13] for extracting mesh is as follows. First, we infer the occupancy at the initial resolution of $32 \times 32 \times 32$. Next, we infer the occupancy again by increasing the resolution of only the boundary of the object to be generated. Since we do not need to compute anything other than the boundary surface, this can be done quickly. In each iteration, we increase the resolution, divide the grid into eight parts, and increase the resolution as $32 \times 32 \times 32 \Rightarrow 64 \times 64 \times 64 \Rightarrow 128 \times 128 \times 128$.

The occupancy field obtained at high resolution is used as the input of the Marching cubes algorithm, and the isosurface is extracted as the mesh. Since this algorithm can always generate a mesh that is watertight and has no self-intersection, it is convenient to consider the volume of food.

A. *Training Hungry Networks*

We explain how to train the network. The loss function used to train the network consists of three terms. The first is a loss that learns the occupancy to represent the 3D volume of the dish. The second is the loss to learn the occupancy to represent the 3D volume of the plate, and the third is the loss to match the shape of the plate part of the two 3D volumes of the dish and the plate. Since occupancy is expressed as a real number between 0 and 1, it is equivalent to binary classification. Therefore, we use BCE loss as shown in Eq.1.

$$\mathcal{L}_{\mathcal{O}}(f_d(x,p), o(p)) = \mathcal{L}_{bce}(f_d(x,p), o(p)) \qquad (1)$$

Here, $p \in \mathbb{R}^3$ is the input point and x is the feature of an input image. f_d is the decoder. Also, the occupancy rate corresponding to the point p is expressed as $o(p) \in \mathbb{R}$. Next, we introduce "plate consistency loss", which is a loss function for matching the plate parts of the two 3D volumes output. This is because if this loss function is not included, the plate shapes of the two output 3D volumes will be different.

Table I shows the pattern of occupancy of Decoder1 and Decoder2. There is no problem if the same point p is used as input and the inferred occupancy is the same. The condition where the occupancy of a dish is 1 and the occupancy of a plate is 0 is not a problem because it corresponds to the food part of the dish model. However, when the occupancy rate of the dish is 0 and the occupancy rate of the plate is 1, it means that the plate parts of the 3D volume of the dish and the plate do not match, which is problematic. Therefore, we solve this problem by introducing the following loss. As can be seen from the table, we use the fact that $f_{d2}(p) - f_{d1}(p)$ is greater than 0 only when the occupancy of the meal is 0 and the occupancy of the tableware is 1.

$$\mathcal{L}_{\mathcal{C}}(f_{d1}(p), f_{d2}(p)) = \max(f_{d2}(p) - f_{d1}(p)\ , 0) \qquad (2)$$

The above two formulas (Eq.1, Eq.2) are put together to determine the loss $\mathcal{L}_{\mathcal{B}}$ for each mini-batch of the entire learning. Here, $\mathcal{B}$ is the sampled mini-batch, I_i is the i-th image of the batch, and K points in total from the i-th batch

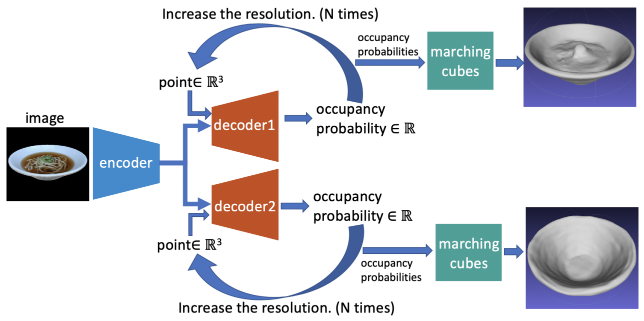

Fig. 2. The overview of "Hungry Networks."[11]

TABLE I
PATTERNS OF OCCUPANCY

dish occupancy ($f_{d1}(p)$)	plate occupancy ($f_{d2}(p)$)	$f_{d2}(p) - f_{d1}(p)$
0	0	0
1	0	-1
0	1	1
1	1	0

are sampled, and $p_{i,j}$ represents the sampled j-th point of the i-th image. It is assumed that f_e is the encoder that output image features, and f_{d1} and f_{d2} are decoder outputs that output food and plate occupancy rates, respectively.

$$x_i = f_e(I_i) \tag{3}$$

$$y1_{i,j} = f_{d1}(x_i, p_{i,j}) \tag{4}$$

$$y2_{i,j} = f_{d2}(x_i, p_{i,j}) \tag{5}$$

$$\begin{aligned}\mathcal{L}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|}\sum_{i=1}^{|\mathcal{B}|}\sum_{j=1}^{K}\Big(&\lambda_1\mathcal{L}_{\mathcal{O}}(y1_{i,j}, o1_i(p_{i,j})) \\ &+ \lambda_2\mathcal{L}_{\mathcal{O}}(y2_{i,j}, o2_i(p_{i,j})) \\ &+ \lambda_3\mathcal{L}_{\mathcal{C}}(y1_{i,j}, y2_{i,j})\Big)\end{aligned} \tag{6}$$

B. Training Results

We created our dataset containing 3D models of dish and plate and used it to train the network. The images for training were created by rendering the 3D models. The non-dish portions of the images were synthesized from various background images. The reconstruction results of the network trained on these datasets using the actual meal image as input instead of the rendered image are shown in Figure 3. It can be seen that even when the network was trained with rendered images, it shows correct 3D reconstruction results with real meal images as input.

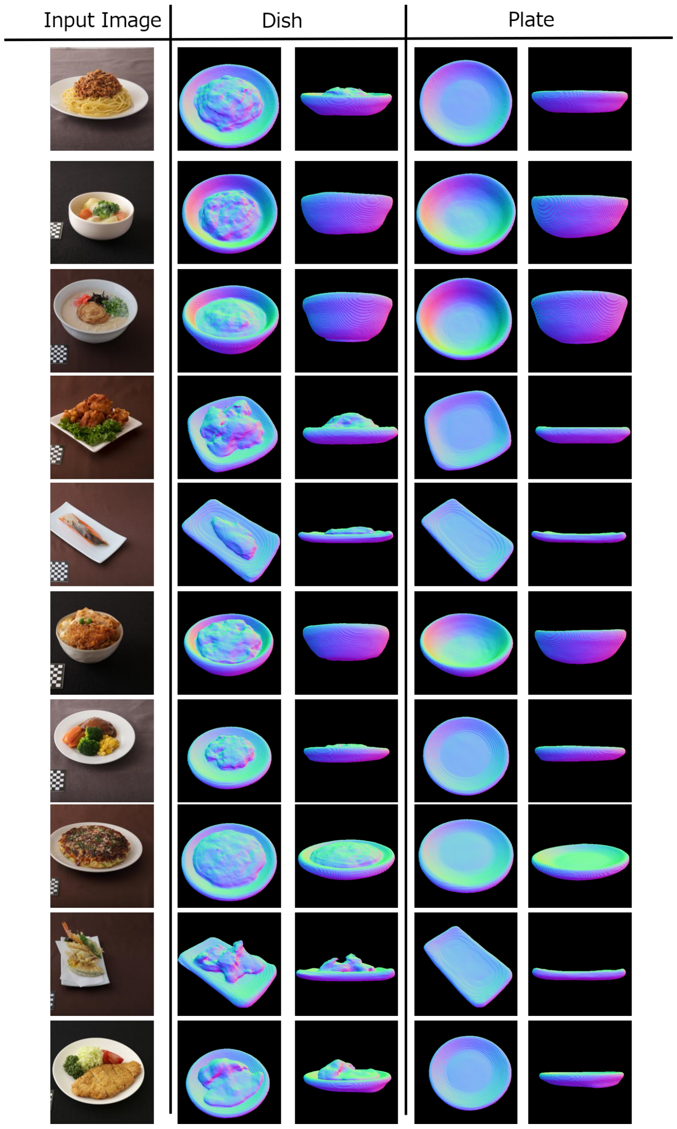

Fig. 3. Hungry Networks reconfiguration results.

III. Pop'n Food

The user interface of the application is shown in Figure 5. There is a list of images on the left and two canvases on the right. The 3D model of the dish is displayed on the upper side of the canvas, and the 3D model of the plate is displayed on the lower side. By clicking the button next to the image, the user can see the 3D model reconstructed from the clicked image. In the works on 3D reconstruction, we often see videos created from images of models rendered from different angles and displayed in a browser. This application, on the other hand, uses WebGL via a JavaScript library called three.js, which allows users to interactively view the 3D model from all angles. The application we created this time has two modes. One is a mode to view a 3D model generated in advance from the image of the data set for evaluation. The other is a mode in which the user can input arbitrary images one by one, generate a 3D model in real-time, and view it. Since the mode in which the model is generated in advance omits some functions of the real-time mode, the operation of the real-time mode will be described here. The architecture for generating 3D models in real-time is shown in Figure 4. First, the user inputs the image into the browser. When the browser loads the image, the image entered in the list of images is displayed on the left side of the UI. Each element of the list of images has a button labeled "Generate 3D." By clicking this button, the image will be sent to the web server. When the web server receives an input image, it throws the image to the API Server equipped with GPU. On this API server, two 3D models of a dish and a plate are generated from the input image using Hungry Networks [11]. After generating the 3D model, the system saves it with the input image in the server-side storage. At this time, a unique model ID such as GUID is also generated at the same time so that the generated 3D model can be searched from the unique model ID. The API Server returns the generated unique model ID to the Web Server, and the Web Server returns it to the browser side. Based on the returned unique model ID, the browser requests the 3D model from the web server, deserializes the 3D model returned as a response and displays it using Three.js. This is the overall flow of real-time mode. In the mode of viewing pre-generated 3D models in the evaluation dataset, a reconstructed 3D model

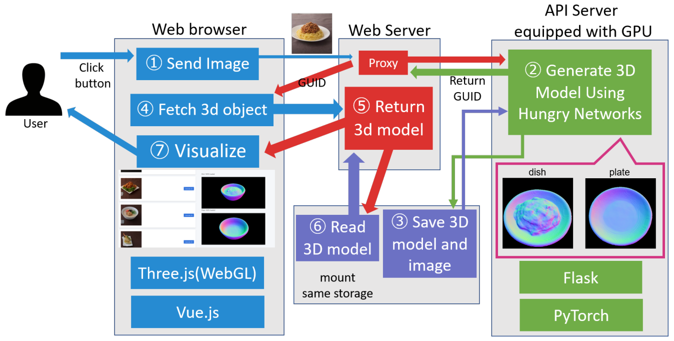

Fig. 4. demo system overview

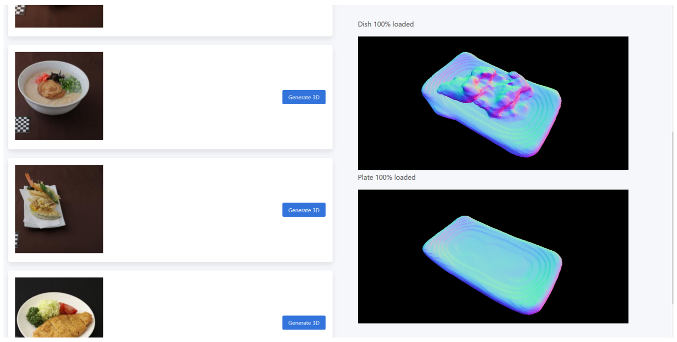

Fig. 5. Web Interface. There is a list of images on the left and two canvases on the right. The 3D model of the dish is displayed on the upper side of the canvas, and the 3D model of the plate is displayed on the lower side. By clicking the button next to the image, the user can see the 3D model reconstructed from the clicked dish image. The 3D model is also rendered in real-time, so you can interact with it (not a video).

and its associated unique model ID are generated in advance. By embedding the unique model ID into html, the 3D model can be easily displayed without a GPU-equipped API Server.

Introducing the libraries and frameworks used for implementation. On the browser side, Three.js and Vue.js are mainly used. Three.js is a library for easily handling WebGL etc. with JavaScript and easily creating 3D CG applications, and Vue.js is a reactive JavaScript framework for creating UI. API Server is implemented in Python, and Web Server is set up using Flask. Flask is a micro web framework. Hungry Networks, which runs internally, is implemented in PyTorch. The GPU equipped in API Server is GTX 1060. By loading Hungry Networks when the web server starts and deploying it on the GPU, the response to the request is within about 2 seconds.

IV. Conclusion

In this paper, we introduced a system that can reconstruct and visualize two 3D models of a dish and a plate from a single dish image in a real-time way on a web browser. By utilizing WebGL, this application enables users to interactively check 3D models. Thanks to such an application, even a generated 3D model can be reconstructed very easily, which is useful for qualitative evaluation and so on.

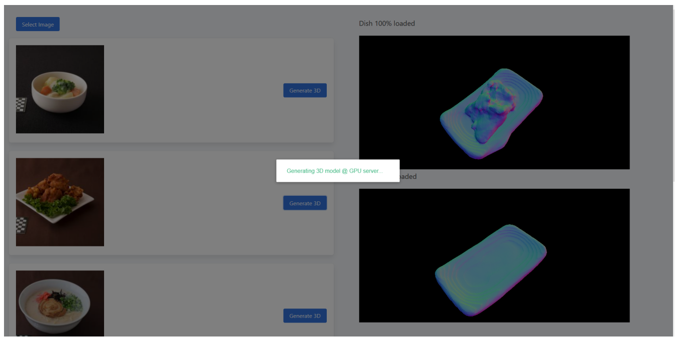

Fig. 6. By clicking the 3D generation button next to the image, the user can generate a 3D model in real-time by inputting an image other than the data set for evaluation and see the result. The generation time per image is 2 seconds.

Acknowledgements: This work was supported by JSPS KAKENHI Grant Number 17H06100.

References

[1] T. Ege and K Yanai. Image-based food calorie estimation using recipe information. *IEICE Transactions on Information and Systems*, E101-D(5):1333–1341, 2018.
[2] T. Ege and K. Yanai. Imag-based food calorie estimation using knowledge on food categories, ingredients and cooking directions. In *Proc. of ACM Multimedia Thematic Workshop*, 2017.
[3] T. Ege and K. Yanai. Estimating food calories for multiple-dish food photos. In *Proc. of Asian Conference on Pattern Recognition*, 2017.
[4] T. Ege and K Yanai. Multi-task learning of dish detection and calorie estimation. In *Proc. of IJCAI and ECAI Workshop on Multimedia Assisted Dietary Management*, 2018.
[5] S. Naritomi and K. Yanai. CalorieCaptorGlass: Food calorie estimation based on actual size using hololens and deep learning. In *Proc. of IEEE Conference on Virtual Reality and 3D User Interfaces*, 2020.
[6] R. Tanno, T. Ege, and K. Yanai. AR DeepCalorieCam V2: food calorie estimation with cnn and ar-based actual size estimation. In *Proc. of the 24th ACM Symposium on Virtual Reality Software and Technology*, pages 1–2, 2018.
[7] A. Meyers, N. Johnston, V. Rathod, A. Korattikara, A. Gorban, N. Silberman, S. Guadarrama, G. Papandreou, J. Huang, and K. P. Murphy. Im2Calories: towards an automated mobile vision food diary. In *Proc. of the IEEE International Conference on Computer Vision*, pages 1233–1241, 2015.
[8] Y. Lu, D. Allegra, M. Anthimopoulos, F. Stanco, G. M. Farinella, and S. Mougiakakou. A multi-task learning approach for meal assessment. In *Proc. of the Joint Workshop on Multimedia for Cooking and Eating Activities and Multimedia Assisted Dietary Management*, pages 46–52, 2018.
[9] Y. Ando, T. Ege, J. Cho, and K. Yanai. DepthCalorieCam: A mobile application for volume-based foodcalorie estimation using depth cameras. In *Proc. of the 5th International Workshop on Multimedia Assisted Dietary Management*, pages 76–81, 2019.
[10] F. P. . Lo, Y. Sun, J. Qiu, and B. P. L. Lo. Point2volume: A vision-based dietary assessment approach using view synthesis. *IEEE Transactions on Industrial Informatics*, 16(1):577–586, 2020.
[11] S. Naritomi and K Yanai. Hungry networks: 3d mesh reconstruction of a dish and a plate from a single dish image for estimating food volume. In *Proc. of ACM Multimedia Asia*, 2020.
[12] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.
[13] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy Networks: Learning 3d reconstruction in function space. In *Proc. of IEEE Computer Vision and Pattern Recognition*, pages 4460–4470, 2019.