# CPSC 542 - Assignment 2

Spencer Au

## Problem Statement

The goal of this project is to use a U-Net Model in order to perform image segmentation on a dataset of various images of food. The dataset contains images of food, and the goal is to be able to segment the images of food into the pixels that are the food in question or the pixels that are just the "background". The relevance of this problem is that it can be used to help automate the process of identifying and classifying food in images, which can be useful for a variety of applications such as food recognition, dietary analysis, and food waste reduction. The task of segmentating (food) images involves understanding complex patterns, which traditional machine learning models may struggle to handle efficiently due to their limited capacity for feature extraction, etc. Deep learning models, particularly convolutional neural networks (CNNs) and specifically the U-Net architecture, excel in image segmentation tasks across many datasets and benchmarks. This capability to learn directly from data and improve with scale makes deep learning the de facto standard for image segmentation problems. The plan to solve the problem is to first preprocess the data, and then train a U-Net model on the dataset. The U-Net model is a popular architecture for image segmentation tasks, and is known for its ability to effectively segment images of varying sizes and shapes. The U-Net model consists of a contracting path, which captures context, and an expansive path, which enables precise localization. The model is trained using a combination of the binary crossentropy loss function and the Adam optimizer. The model is trained on the Chapman University DGX Cluster, which is a high performance computing cluster with 8 NVIDIA A100 GPUs. The model is trained for 100 epochs, and is evaluated on a test set to determine its performance. The model is then used to segment images of food into different categories, and the results are analyzed to determine the model's effectiveness.

## Data

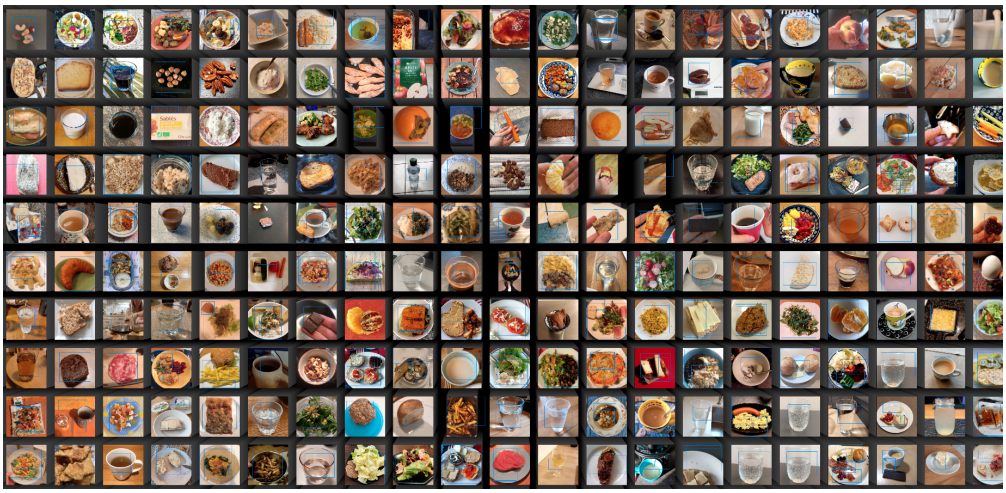

Figure 1: Visualized Data

The data used in this project was obtained from the Food Recognition 2022 Dataset from Kaggle. The dataset contains 39,962 RGB images and 76,491 annotations in teh training class, split into 498 classes. The validation set consists of 1000 RGB images, with 1830 annotations and 498 classes. The test set is the Food Recognition Benchmark. The dataset is formatted in MS-COCO format. Since we are only performing binary segmentation, namely seeing which pixels belong to the object being detected and which are background pixels, we only need the images and the annotations. The images are in the form of .jpg files, and the annotations are in the form of .json files. The images are of varying sizes, and the annotations are in the form of polygons. The dataset is quite large, and contains a wide variety of food images, which will help the model generalize to new images of food. The dataset is also quite diverse, containing images of food from different cultures, with different colors, and different backgrounds, which will help the model learn to classify images of food that may be occluded, be of different colors, have different backgrounds, or lighting conditions that cannot be effectively controlled for via image augmentation. The dataset is also quite balanced, with a large number of images for each class, which will help the model learn to classify images of food that may be occluded, be of different colors, have different backgrounds, or lighting conditions that cannot be effectively controlled for via image augmentation. The dataset is also quite balanced, with a large number of images for each class, which will help the model learn to classify images of food that may be occluded, be of different colors, have different backgrounds, or lighting conditions that cannot be effectively controlled for via image augmentation.

# Methods

## Preprocessing

In terms of preprocessing, I used a custom HubSegmentationDataGenerator class which extends the Keras Sequence class in order to load the data, and perform image augmentation. The class is a subclass of the Keras Sequence class, and is used to load the images and annotations, and perform image augmentation. The class takes in the image and annotation datasets, the batch size, image dimensions, and a shuffle flag to determine if the data should be shuffled at the end of each epoch. The **len** method computes the number of batches per epoch, while **getitem** fetches a specific batch, applying necessary preprocessing such as resizing and datatype conversion. Images are resized to the target dimensions and preprocessed for neural network compatibility, and masks are converted from boolean to uint8, resized to match the images, and ensured to be in a single-channel format. If any mask has more than one channel, only the first channel is used, and any extra dimensions are removed to maintain a consistent input shape for the model. This generator thereby streamlines the process of preparing image-mask pairs for efficient training of segmentation models. **on_epoch_end** is automatically called at the end of each epoch during the model's training. Its primary function is to update the indices of the dataset. **data_generation** generates a batch of data by processing the corresponding images and masks for training. For each index in the current batch, it loads the image from the Hub dataset, resizes it to the desired dimensions, and applies any necessary preprocessing steps such as normalizing pixel values. Concurrently, it processes the associated masks by resizing them and converting their data type to ensure consistency, and, if needed, adds a channel dimension to match the model's expected input shape. The processed images and masks are then compiled into arrays which are subsequently returned to the model for training on that particular batch.

## Model

In terms of the model, I used a U-Net model style architecture, which is a popular architecture for image segmentation tasks. The U-Net model consists of a encoding path and a decoding path, which are connected by skip connections. The encoding and decoding path are also connected by a convolutional block as a bridge between the two paths. Each encoder block consists of a convolutional block followed by a 2x2 max pooling layer. Each decoder block consists of an transposed convolutional layer with a 3x3 kernel and a stride of 2 for upsampling followed by a concatenation with the corresponding encoder block to facilitate skip connections, and then 2 subsequent convolutional layers using a 3x3 kernel, ReLu activation, and same padding. The convolutional blocks consist of a convolutional layer with a 3x3 kernel, ReLu activation, same padding, and l2 regularization followed by a Dropout layer with a dropout rate of 0.4, and then follwoed by another convolutional layer with a 3x3 kernel, ReLu activation, same padding, and l2 regularization.

The encoder part of the U-Net architecture consists of 4 encoder blocks, with 64, 128, 256, and 512 filters respectively. The bridge consists of a convolutional block with 1024 filters. The decoder part of the U-Net architecture consists of 4 decoder blocks, with 512, 256, 128, and 64 filters respectively. The output layer consists of a convolutional layer with 1 filter and a sigmoid activation function. The model is trained using the binary crossentropy loss function and the Adam optimizer with a learning rate of 0.0001. The model is trained for 200 epochs utilizing early stopping to prevent overfitting wiht a patience of 15 epochs. The model was evauluated using accuracy and the average IoU as the metrics, and the results were analyzed to determine the model's effectiveness.

## Results and Discussion

| Metric | Train | Val |
|---|---|---|
| Loss | 0.2906 | 0.2894 |
| Accuracy | 0.8686 | 0.8700 |
| IoU | 0.6134 | 0.6149 |

Looking at the data and metrics, both the training and validation set performed reasonably well, as the loss for both is relatively low at 0.2906 and 0.2894 for the training and validation sets respectively. In addition, the accuracy metric is also prety good on both the training and validation sets at 0.8686 and 0.8700 respectively. The IoU, or Intersection over Union, is decent at 0.6134 and 0.6149 respectively. An IoU of 0.6134 means that, on average, the overlap area between the predicted masks and the actual masks is 61.34% of their union area for that evaluation set. Similarly, an IoU of 0.6149 indicates a 61.49% overlap in that evaluation set. These scores suggest that the model has moderate to good performance in terms of accurately segmenting the objects of interest compared to the actual masks. Below are some example segmentation images to visualize how the model has performed. Overall the model seems to be doing a pretty solid job of segmentating images, as the predicted masks vs the actual masks seem to be pretty close.

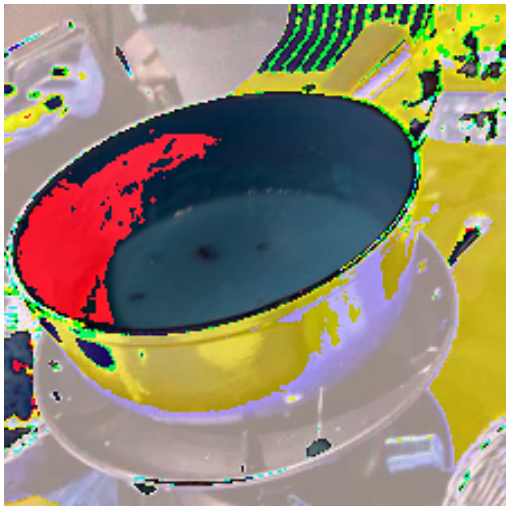

Figure 2: Image 2

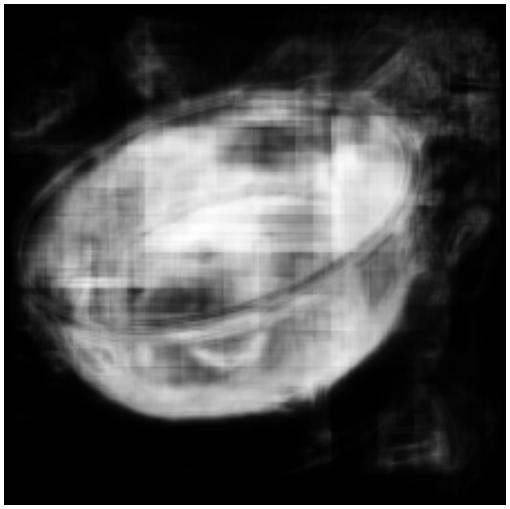

Figure 3: Image 2 Predicted Mask

Figure 4: Image 2 Actual Mask

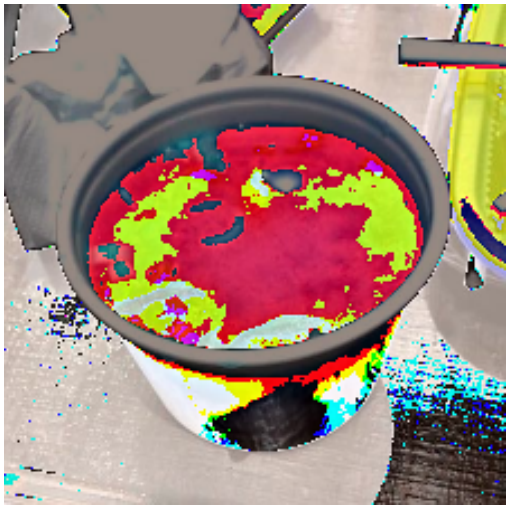

Figure 5: Image 3

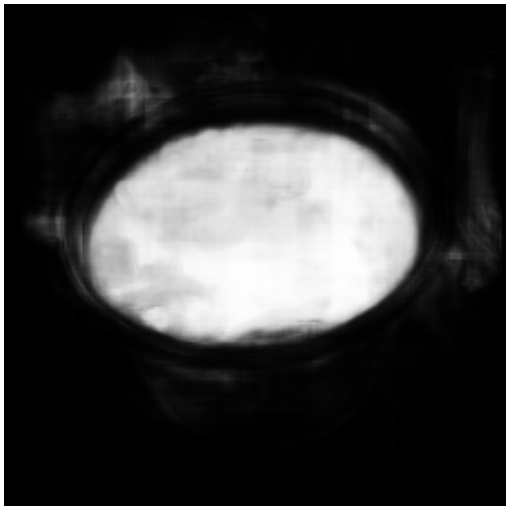

Figure 6: Image 3 Predicted Mask

Figure 7: Image 3 Actual Mask

Figure 8: Image 4

Figure 9: Image 4 Predicted Mask

Figure 10: Image 4 Actual Mask