Food Volume Estimation

Using monocular depth estimation to estimate food volume in an input image.

Method

Depth Network Training

A monocular depth estimation network is trained using monocular video sequences, as suggested by Godard et al. . The sequences used for this purpose are obtained from the EPIC-Kitchens dataset, which includes many hours of egocentric, food handling videos.

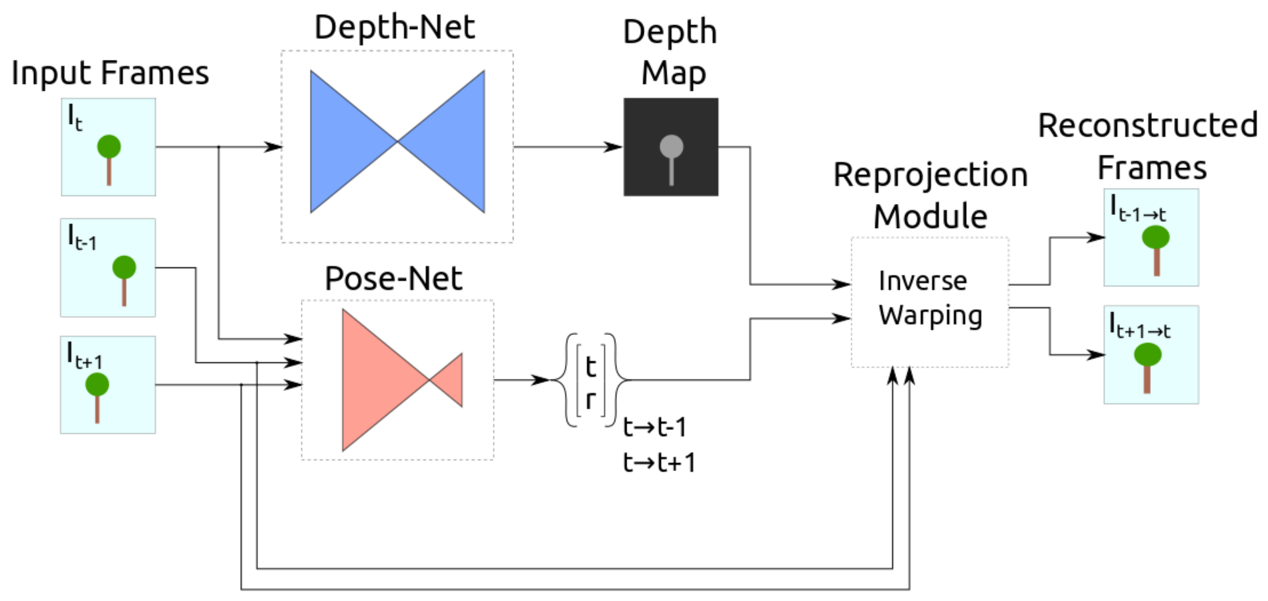


Figure 1: Depth Network Training

Volume Estimation

The food input image is passed through the trained depth network and segmentation module to predict the depth map and food object mask. These outputs, along with the camera intrinsics, generate a point cloud on which the volume estimation is perfomed.

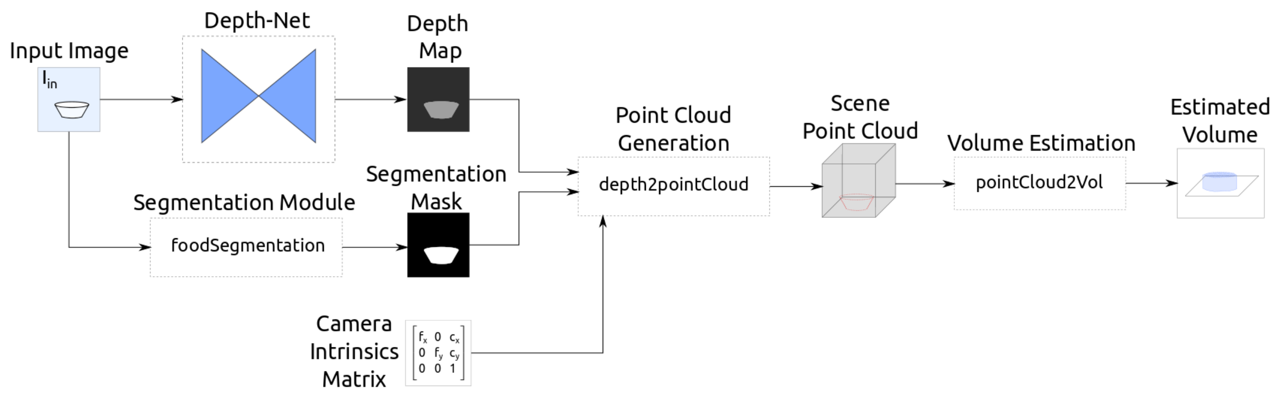


Figure 2: Volume Estimation

Requirements

The code is written and tested in `python 3.6`. The required pip packages for running the volume estimation script are:

```
numpy==1.16.3
pandas=0.24.2
opencv-python==4.1.0.25
scipy==1.2.1
scikit-learn==0.21.1
tensorflow==1.13.1
keras==2.2.4
h5py==2.9.0
matplotlib==3.0.3
```

To train the model you also need `image-classifiers==0.2.1` for importing the required Resnet18 model and weights.

Training

To train the depth estimation network use the `monovideo.py` script as:

```
monovideo.py --train --train_dataframe dataFrame.csv --config config.json
   --batch_size B --training_epochs E --model_name name --save_per S
   --starting_weights initial_weights.h5
```

The required arguments include a Pandas dataFrame (`dataframe.csv`) containing paths to frame triplets:

curr_frame	prev_frame	next_frame
path_to_frame_t	path_to_frame_t-1	path_to_frame_t+1
path_to_frame_t+1	path_to_frame_t	path_to_frame_t+2
...	...	...

and a JSON file (`config.json`) that describes various training parameters:

```
{
  "name": "epic-kitchens",
  "img_size": [128, 224, 3],
  "intrinsics": [[1564.51, 0, 960], [0, 1564.51, 540], [0, 0, 1]],
  "depth_range": [0.01, 10]
}
```

The model architecture is saved in `name.json` when the model is instantiated whereas the model weights are saved in `name_weights_[epoch_e/final].h5` every `S` epochs and when training is complete (H5 format). All outputs are stored in the `trained_models` directory.
The triplet-defining dataFrame can be created using the `data_utils.py` script as:

```
data_utils.py --create_set_df --data_source data_sources --save_target df.csv --stride S
```

where the `data_sources` file contains the directories in which the images are saved. For example:

```
/home/usr/food_volume_estimation/datasets/EPIC_KITCHENS_2018/frames/rgb/train/P01/P03_3/
/home/usr/food_volume_estimation/datasets/EPIC_KITCHENS_2018/frames/rgb/train/P01/P05_1/
```

You can also create a training set from multiple EPIC-Kitchens source directories, resizing the images and applying optical flow filtering (proposed by Zhou et al.) to reduce overall training costs:

```
data_utils.py --create_EPIC_set --data_source data_sources --save_target save_dir
  --target_width W --target_height H --interp [nearest/bilinear/cubic] --stride S
```

To avoid redefining the data sources after creating and saving a training set of images, use the `create_dir_df` flag:

```
data_utils.py --create_dir_df --data_source img_dir --save_target df.csv --stride S
```

The recommended stride value for the EPIC-Kitchens dataset is 10.

Testing

The `model_tests.py` script offers testing of either all network outputs or the full-scale predicted depth:

```
model_tests.py --test_outputs --test_dataframe test_df.csv --config config.json
   --model_architecture model_name.json --model_weights model_name_weights.h5 --n_tests 5
```

```
model_tests.py --infer_depth --test_dataframe test_df.csv --config config.json
   --model_architecture model_name.json --model_weights model_name_weights.h5 --n_tests 5
```

Again, a Pandas dataFrame defining the frame triplets is required, since the all-outputs test generates the source to target frame reconstructions. All tests are performed without data augmentation.

Volume Estimation

To estimate the food volume in an input image use the `estimate_volume.py` script as:

```
estimate_volume.py --input_image img_path --depth_model_architecture model_name.json
   --depth_model_weights model_name_weights.h5 --segmentation_model [GAP/GMAP/GMP] --fov D
   --focal_length F --depth_rescaling R --min_depth min_d --max_depth max_d
   [--plot_results]
```

The model architecture and weights are generated by the training script, as discussed above. If either the camera field of view (FoV) or focal length is given, the intrinsics matrix is generated during runtime. Otherwise, a default 60 degrees FoV is used to generate the matrix. The depth rescaling, min depth and max depth parameters are model-dependent and should not be changed unless the model has been retrained with the new values.
If you wish to visualize the volume estimation pipeline, run the example notebook `visualize_volume_estimation.ipynb`. Point cloud plots are dependent on the PyntCloud library.

Models

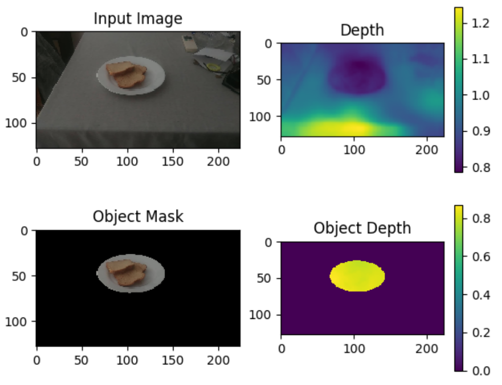
Download links for the pre-trained models:

- Low-res model:
  - → Architecture: https://drive.google.com/open?id=1IJ4k1TtFpConpkJVsGf37F-WWJ1rmP4Y
  - → Weights: https://drive.google.com/open?id=1mFvc20GbzUGyo9xl401BNxGNSLiKEewr

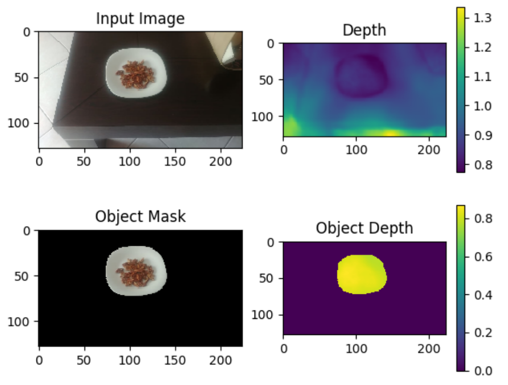
Todo

- ✓ Trained low-res model (224x128 inputs) and achieved promising results.
- ✓ Trained the high-res model (448x256 inputs). No obvious advantages observed.
- ☐ Calibrate depth predictions.
- ☐ Fitting volume primitives based on food detection could improve volume estimation.

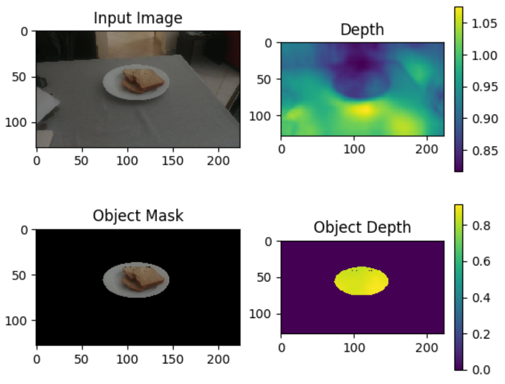
Low-res model examples:



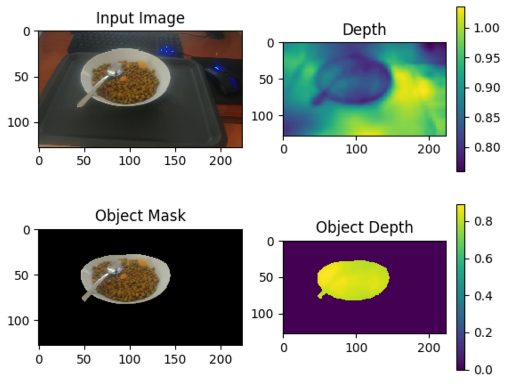
(a)



(b)



(c)



(d)

Figure 3: (a) **Example 1**: Estimated volume 0.30L (b) **Example 2**: Estimated volume 0.66L (c) **Example 3**: Estimated volume 0.28L (d) **Example 4**: Estimated volume 1.21L