YOLACT Training README

San Jose State University, Department of Computer Engineering

Author: Adam Goldstein
Project Advisor: Dr. Harry Li

03/16/2023	Created Document	Adam Goldstein
03/21/2023	Added Real-Time Inference with VirtualHome Instructions	Adam Goldstein

Github Source Link: https://github.com/haotian-liu/yolact_edge
Github Fork:
https://github.com/adkap2/Instance-Segmentation-with-Unity-Interactive-Augmented-Reality

Table of Contents

I. Introduction

Computer vision has become popular in fields such as virtual reality (VR) and augmented reality (AR). Within computer vision, real-time instance segmentation is used to detect objects on a pixel level basis. Real-time instance segmentation is a challenging problem that requires the detection and segmentation of each object instance in an image. YolactEdge is a deep learning model that performs instance segmentation in real-time and runs accurately on edge devices at real-time speeds. YolactEdge leverages TensorRT, a high-performance deep learning inference library, to achieve fast and accurate inference on NVIDIA GPUs.

In this use case, a YolactEdge model is trained to provide real-time instance segmentation inference within the Unity virtualhome environment. To achieve real-time inference, the power of TensorRT, a high-performance deep learning inference library, was leveraged. Additionally, the NetMQ library was used to perform TCP/IP socket communication between UnityVirtualHome and the YolactEdge inference model.

The training data used to develop the YolactEdge model was synthetically generated from Unity, simulating a diverse range of real-world scenarios. To generate the ground truth annotations for this data, the household assets available in the Unity SyntheticHome application were utilized, which automatically generated accurate annotations for each object in the scene.

The integration of NetMQ allowed for a reliable and fast communication channel between Unity and YolactEdge. Every frame generated in Unity is compressed and sent to YolactEdge for prediction with minimal delay. This entire pipeline runs at around 10-15 fps on my system, which is suitable for many real-time applications.

The following sections will discuss the details of the communication pipeline, including the preprocessing of the input frames, the transmission of data using NetMQ, and the postprocessing of the inference results. An evaluation of the performance of the pipeline will be provided, including the accuracy of the predictions and the latency of the system. Finally, the potential applications of this pipeline in various VR/AR environments will be discussed.

The readme is comprised of the following parts

1. Installation Instructions
2. Configuring the model to train with a custom dataset
3. Training the model
4. Evaluation and Inference with YolactEdge
5. Real-Time Inference with Unity VirtualHome

II. Installation

Note: My computer specifications and installation dependencies are as follows
Computer Specifications

- Intel i5 9600k
- GTX 1080 ti (11gb ram)
- 32gb memory
- 200gb ssd

Operating System

- Ubuntu 20.04

Installation Dependencies

- Python=3.9.13
- nvidia-cuda-nvrtc-cu11==11.7.99
- nvidia-cuda-runtime-cu11==11.7.99
- nvidia-cudnn-cu11==8.5.0.96
- opencv-python==4.5.1.48
- pycocotools==2.0.6
- tensorrt==8.5.3.1
- torch==1.13.1
- torch2trt==0.4.0
- torchvision==0.14.1

1. Clone the github repository
   ```
   git clone https://github.com/haotian-liu/yolact_edge.git
   ```
2. cd into directory
   ```
   cd yolact_edge
   ```
3. Create a new conda environment
   ```
   conda create -n yolactedge_env
   conda activate yolactedge_env
   ```
   # Note: If dependencies in environment.yml break, try installing each dependency manually based on my personal environment
4. Download pretrained weights file such as 'yolact_base_54_800000.pth'
5. Make new directory 'weights'
   ```
   mkdir weights
   ```
6. Place weight file inside weights directory
7. Place example image inside directory and name it "input_image.png'
8. Test Yolact by evaluating it on a single image

```
# Process an image and save it to another file.
python eval.py --trained_model=weights/yolact_base_54_800000.pth
--score_threshold=0.15 --top_k=15 --image=input_image.png:output_image.png
```

III. Configuring the Model

To prepare YolactEdge for training with a custom dataset, modifications are necessary to the ‘config.py’ file and ‘yolact.py’. YolactEdge must be properly configured to train on the custom object classes associated with the indoor environment.

1. Modify the config.py file
   # Note: config.py is located inside the ‘data’ directory from within the yolact repository

```
tree
├── backbone.py
├── CHANGELOG.md
├── data
│   ├── coco.py
│   ├── config.py
│   ├── grid.npy
```

   a. Open ‘config.py’

```
code config.py
```

   b. Create a new dataset derived from dataset base and name it SyntheticHome
      # Note: DATASETS begin on line 106 in the config.py file

```
SyntheticHome = dataset_base.copy({
    'name': 'SyntheticHome'
})
```

   c. Within SyntheticHome, add the file path to the train and validation images and labels

```
SyntheticHome = dataset_base.copy({
   'name': 'SyntheticHome',
   'train_images': '/path/to/training/images/',
   'train_info':   '/path/to/training/labels/',
   'valid_images': '/path/to/validation/images/',
   'valid_info':   '/path/to/validation/labels',
})
```

   d. Within SyntheticHome, add a label map offsetting the index from 0 to 1 with length equal to the number of class labels

```
SyntheticHome = dataset_base.copy({
   'name': 'SyntheticHome',
   'train_images': 'train_images': '/path/to/training/images/',
   'train_info':   '/path/to/training/labels/',
   'valid_images': '/path/to/validation/images/',
   'valid_info':   '/path/to/validation/labels',
   'label_map': {0:1, 1:2, 2:3, 3:4, 4:5, 5:6, 6:7, 7:8},
```

```
})
```

e. Within SyntheticHome, add alphabetically ordered ‘class_names’ derived from the annotation file

```
SyntheticHome = dataset_base.copy({
   'name': 'SyntheticHome',
   'train_images': 'train_images': '/path/to/training/images/',
   'train_info':   '/path/to/training/labels/',
   'valid_images': '/path/to/validation/images/',
   'valid_info':   '/path/to/validation/labels',
   'label_map': {0:1, 1:2, 2:3, 3:4, 4:5, 5:6, 6:7, 7:8},
'class_names': ("Bed", "Bench", "Chair", "Door", "Microwave", "Refrigerator", "Sofa",
"Table")
})
```

f. Under CONFIG DEFAULTS, create a new configuration object named ‘synthetic_home_config’
Note: CONFIG DEFAULTS begin around line 430

```
 synthetic_home_config = yolact_base_config.copy({
   'name': 'synthetic_home',
})
```

g. Under synthetic_home_config specify the SyntheticHome dataset and class length

```
synthetic_home_config = yolact_base_config.copy({
   'name': 'synthetic_home',
   # Dataset stuff
   'dataset': SyntheticHome,
   # Class length is + 1 to contain background class
   'num_classes': len(SyntheticHome.class_names) + 1,
})
```

h. Under synthetic_home_config, decrease the learning rate for optimal performance on custom dataset

```
synthetic_home_config = yolact_base_config.copy({
   'name': 'synthetic_home',
   # Dataset stuff
   'dataset': SyntheticHome,
   # Class length is + 1 to contain background class
   'num_classes': len(SyntheticHome.class_names) + 1,
   # Slow the learning rate
   'lr': 1e-4,
})
```

i. Decrease the max number of iterations to 40,000 due to training on custom dataset

```
    'max_iter': 40000, # Smaller dataset
```

j. Modify the ‘cfg’ variable to read the ‘synthetic_home_config’ object
Note ‘cfg’ should be located around line 1000

```
cfg = synthetic_home_config.copy()
```

2. Modify the yolact.py file
Note yolact.py is located within the base yolact_edge directory
If performing transfer learning from the pretrained YolactBase model, freeze all trained layers except for the prediction layer.

```
tree
├── [ 17K]  backbone.py
├── [2.9K]  CHANGELOG.md
├── [4.0K]  data
├── [ 928]  environment.yml
├── [ 46K]  eval.py
├── [4.0K]  external
├── [4.0K]  layers
├── [1.0K]  LICENSE
├── [4.0K]  logs
├── [4.0K]  __pycache__
├── [ 16K]  README.md
├── [4.0K]  results
├── [1.4K]  run_coco_eval.py
├── [4.0K]  scripts
├── [ 21K]  train.py
├── [4.0K]  utils
├── [4.0K]  web
├── [4.0K]  weights
└── [ 31K]  yolact.py
```

a. Modify ‘pred_x’ to ‘pred_x.detach()’
Note: this is necessary for training on a custom dataset with transfer learning
Note: p = pred_layer(pred_x) is located around line 1770

```
                # p = pred_layer(pred_x)
                p = pred_layer(pred_x.detach())
```

b. Place the state_dict loader in a try-except block
Note: this is located around line 1260

```
        try:
            self.load_state_dict(state_dict)
        except RuntimeError as e:
            print('Ignoring " ' + str(e)+ '"')
```

IV. Training the Model

There are two approaches to training the YolactEdge model on the Unity Synthetic Home generated dataset.

1. Training the model from scratch
2. Applying transfer learning from a Yolact pretrained model

Training the model from scratch using only the Synthetic Home Generated dataset may result in better overall evaluation performance. However, this approach requires significantly more training time and may produce a less versatile model.

To train the model on the custom SyntheticHomes dataset with transfer learning, you must select a pre-trained weights file such as 'yolact_base_800000.pth'. This approach reduces the required training time from 7 days to 3 days.

1. From the base yolactedge directory, call ‘train.py’ with the following parameters

Note: the --resume parameter specifies the weights file to begin training from if using transfer learning
Note: --start_iter=-1 sets the first iteration to be what is pre-specified in the associated weights file
Note: --batch_size=5 modifies the number of epochs required for training, the default batch size of 8 requires more Vram usage.

```
python train.py --config=synthetic_home_config
--resume=weights/yolact_base_54_800000.pth --start_iter=0 --batch_size=5
```

2. View the training and compare the first epoch time estimation and mAP values to those below

```
[18182]  800010 || B: 1.577 | C: 10.912 | M: 2.176 | S: 5.294 | T: 19.960 || ETA: 5 days, 15:44:24
|| timer: 0.422
[18182]  800020 || B: 1.440 | C: 10.894 | M: 2.292 | S: 5.199 | T: 19.825 || ETA: 4 days, 0:28:17 ||
timer: 0.433
[18182]  800030 || B: 1.663 | C: 10.810 | M: 2.395 | S: 5.101 | T: 19.969 || ETA: 3 days, 11:38:53
|| timer: 0.421
[18182]  800040 || B: 1.699 | C: 10.742 | M: 2.466 | S: 5.000 | T: 19.907 || ETA: 3 days, 5:14:58 ||
timer: 0.420
[18182]  800050 || B: 1.716 | C: 10.654 | M: 2.509 | S: 4.926 | T: 19.806 || ETA: 3 days, 1:14:28 ||
timer: 0.421

Computing validation mAP (this may take a while)...


Calculating mAP...

       |  all  |  .50  |  .55  |  .60  |  .65  |  .70  |  .75  |  .80  |  .85  |  .90  |  .95  |
-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
```

```
   box |  0.12 |  0.17 |  0.17 |  0.17 |  0.17 |  0.17 |  0.17 |  0.17 |  0.00 |  0.00 |  0.00 |
  mask |  0.08 |  0.17 |  0.17 |  0.17 |  0.17 |  0.17 |  0.00 |  0.00 |  0.00 |  0.00 |  0.00 |
-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
```

3. Continue training the model until completion

V. Evaluation and Inference with YolactEdge

1. Evaluate the yolact model trained on the custom indoor home dataset
    a. cd into the weights file directory

```
cd weights
```

    b. Locate the most recently trained weights file

```
ls -l
```

    c. Copy the filename

```
yolact_base_20454_900000.pth
```

    d. From yolact_edge base dir, call eval.py to evaluate the model with the specified weights file

```
python eval.py --config=synthetic_home_config
--trained_model=weights/yolact_base_20454_900000.pth --score_threshold=0.15 --top_k=15
```

    e. View the evaluation metrics and compare the mAP values to those shown below

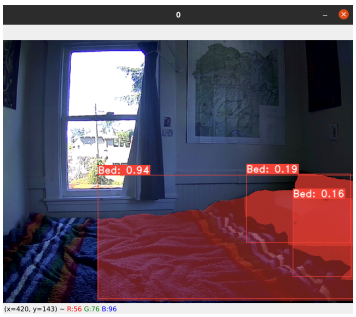
```
Processing Images [                                                  ]     55 /     55 (100.00%)     5.23 fps
Saving data...
Calculating mAP...

       |  all  |  .50  |  .55  |  .60  |  .65  |  .70  |  .75  |  .80  |  .85  |  .90  |  .95  |
-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
   box | 29.14 | 38.92 | 38.07 | 37.57 | 35.50 | 34.62 | 30.69 | 29.18 | 24.66 | 16.42 |  5.81 |
  mask | 30.56 | 38.40 | 37.96 | 37.46 | 36.37 | 35.64 | 33.72 | 31.65 | 24.50 | 21.84 |  8.10 |
-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
```

2. Run inference from a live video camera
    a. From the yolact_edge base directory, call

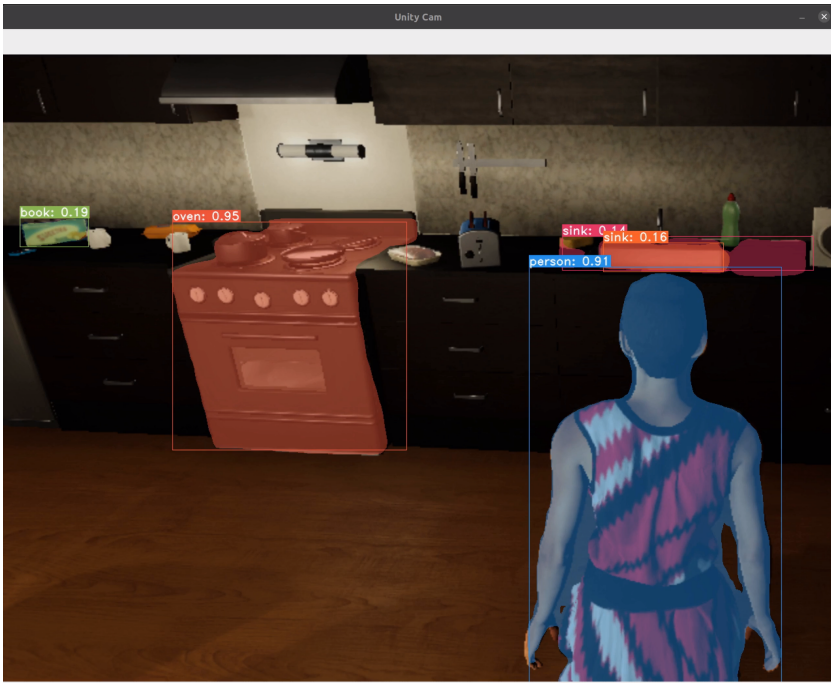
```
python eval.py --config=synthetic_home_config
--trained_model=weights/yolact_base_20454_900000.pth --score_threshold=0.15
--top_k=15 --video_multiframe=4 --video=0
```

    b. View the live videocam feed



VI. Real-Time Inference with Unity VirtualHome

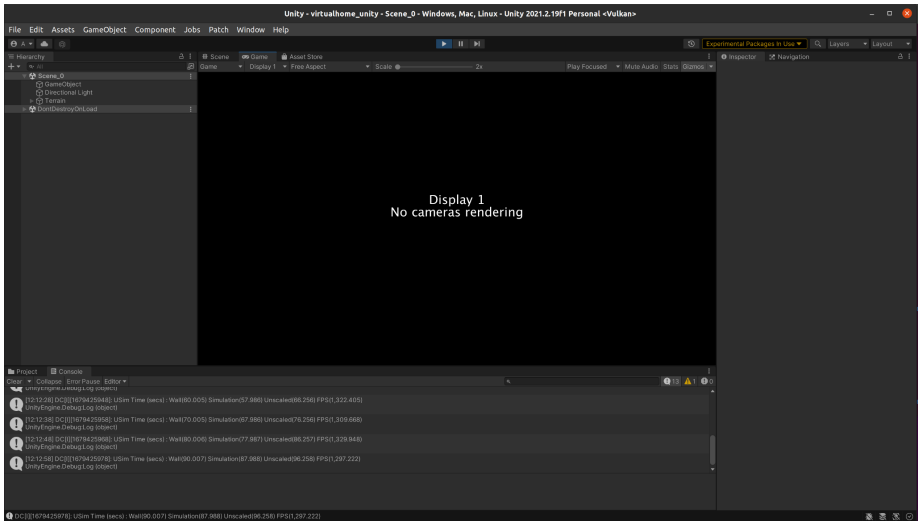
Performing Real-Time instance segmentation with Unity VirtualHomes should result in the successful detection and classification of household objects from Unity. Using TCP/IP Socket Communication, each Unity frame is continuously streamed to python where YolactEdge runs real-time inference.



1. Ensure Unity VirtualHomes is installed and running based on the instructions provided in this README document https://github.com/adkap2/Instance-Segmentation-with-Unity-Interactive-Augmented-Reality/blob/main/documentation/Unity_VirtualHome_README.pdf

Note: Steps 3 and 4 must be run simultaneously

2. Ensure VirtualHome in Unity is in play mode



3. Run real_time_inference.py to
    A. Cd into yolact_edge directory

```
cd yolact_edge
```

    B. Run real_time_inference.py

```
python real_time_inference_.py
```

    C. Wait for 3-5 minutes for model to load and TCP/IP Socket communication to begin

```
Configuring YOLACT edge...
Loading YOLACT edge model...
Loading weights from 'weights/yolact_edge_54_800000.pth'...
```

    - Loads the YolactEdge trained instance segmentation model with the specified config parameters
    - Sets up a ZeroMQ socket to receive images delivered from VirtualHome
    - Displays the detections continuously using OpenCV
4. Run virtualhome_running.py to load and run the Unity VirtualHome agent activity scripts and scene
    A. In a new terminal, activate the conda environment created from Unity VirtualHome installation

```
conda activate unityvhome
```

    B. Enter the base project directory

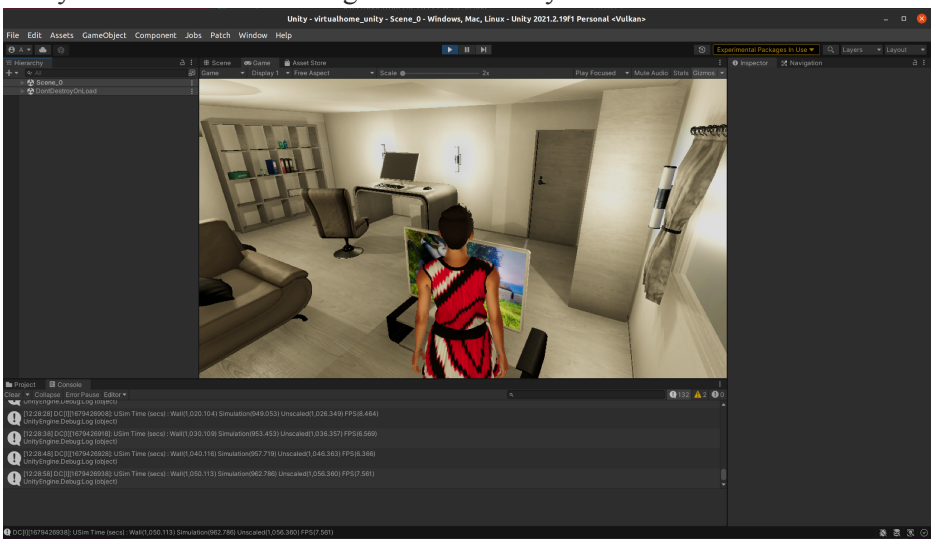
```
cd Instance-Segmentation-with-Unity-Interactive-Augmented-Reality
```

    C. Run the code blocks in virtual_home_runner.ipynb

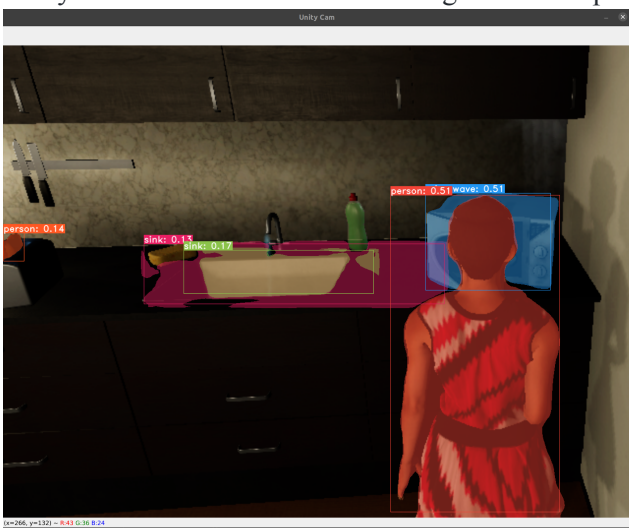
```
virtual_home_runner.ipynb
```

5. View the real-time Instance Segmentation Inference

A. Verify VirtualHome is running the task in Unity



B. Verify the real-time instance is running from the OpenCV window



C. Monitor the FPS from `real_time_inference.py`

```
(yolactedge_env)
adam@adam-Z390-M-GAMING:~/Desktop/CMPE295/Instance-Segmentation-with-Unity-
Interactive-Augmented-Reality/yolact_edge$ python real_time_inference.py
Configuring YOLACT edge...
Loading YOLACT edge model...
Loading weights from 'weights/yolact_edge_54_800000.pth'...
Warning: Encountered known unsupported method torch.zeros
Model ready for inference...
Starting up on localhost port 5555
Waiting for messages...
Benchmarking performance...
Average 18.4940562277329 FPS
Average 14.906720688061982 FPS
```

Citation

[1] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, “YOLACT: Real-Time Instance Segmentation,” in 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019.

[2] H. Liu, R. A. R. Soto, F. Xiao, and Y. J. Lee, “YolactEdge: Real-time instance segmentation on the edge,” arXiv [cs.CV], 2020.

[3] “VirtualHome,” Virtual-home.org. [Online]. Available: http://virtual-home.org/. [Accessed: 21-Mar-2023].