Augmented Autoencoders

Implicit 3D Orientation Learning for 6D Object Detection from RGB Images

Martin Sundermeyer, Zoltan-Csaba Marton, Maximilian Durner, Manuel Brucker, Rudolph Triebel
Best Paper Award, ECCV 2018.

paper, supplement, oral

Citation

If you find Augmented Autoencoders useful for your research, please consider citing:

```
@InProceedings{Sundermeyer_2018_ECCV,
author = {Sundermeyer, Martin and Marton, Zoltan-Csaba and Durner, Maximilian and Brucker, Manuel and Triebel, Rudolph},
title = {Implicit 3D Orientation Learning for 6D Object Detection from RGB Images},
booktitle = {The European Conference on Computer Vision (ECCV)},
month = {September},
year = {2018}
}
```

Multi-path Learning for Object Pose Estimation Across Domains

Martin Sundermeyer, Maximilian Durner, En Yen Puang, Zoltan-Csaba Marton, Narunas Vaskevicius, Kai O. Arras, Rudolph Triebel
CVPR 2020
The code of this work can be found here

Overview

We propose a real-time RGB-based pipeline for object detection and 6D pose estimation. Our novel 3D orientation estimation is based on a variant of the Denoising Autoencoder that is trained on simulated views of a 3D model using Domain Randomization. This so-called Augmented Autoencoder has several advantages over existing methods: It does not require real, pose-annotated training data, generalizes to various test sensors and inherently handles object and view symmetries.

<

p align="center">

<

p>

1.) Train the Augmented Autoencoder(s) using only a 3D model to predict 3D Object Orientations from RGB image crops \ 2.) For full RGB-based 6D pose estimation, also train a 2D Object Detector (e.g. https://github.com/fizyr/keras-retinanet) \ 3.) Optionally, use our standard depth-based ICP to refine the 6D Pose

Requirements: Hardware

For Training

Nvidia GPU with >4GB memory (or adjust the batch size)
RAM >8GB
Duration depending on Configuration and Hardware: ~3h per Object

Requirements: Software

Linux, Python 2.7 / Python 3 (experimental)

GLFW for OpenGL:

```
sudo apt-get install libglfw3-dev libglfw3
```

Assimp:

```
sudo apt-get install libassimp-dev
```

Tensorflow >= 1.6
OpenCV >= 3.1

```
pip install --user --pre --upgrade PyOpenGL PyOpenGL_accelerate
pip install --user cython
pip install --user cyglfw3
pip install --user pyassimp==3.3
pip install --user imgaug
pip install --user progressbar
```

Headless Rendering

Please note that we use the GLFW context as default which does not support headless rendering. To allow for both, onscreen rendering & headless rendering on a remote server, set the context to EGL:

```
export PYOPENGL_PLATFORM='egl'
```

In order to make the EGL context work, you might need to change PyOpenGL like here

Preparatory Steps

1. Pip installation

```
pip install --user .
```

2. Set Workspace path, consider to put this into your bash profile, will always be required

```
export AE_WORKSPACE_PATH=/path/to/autoencoder_ws
```

3. Create Workspace, Init Workspace (if installed locally, make sure .local/bin/ is in your PATH)

```
mkdir $AE_WORKSPACE_PATH
cd $AE_WORKSPACE_PATH
ae_init_workspace
```

Train an Augmented Autoencoder

1. Create the training config file. Insert the paths to your 3D model and background images.

```
mkdir $AE_WORKSPACE_PATH/cfg/exp_group
cp $AE_WORKSPACE_PATH/cfg/train_template.cfg $AE_WORKSPACE_PATH/cfg/exp_group/my_autoencoder.cfg
gedit $AE_WORKSPACE_PATH/cfg/exp_group/my_autoencoder.cfg
```

2. Generate and check training data. The object views should be strongly augmented but identifiable.

(Press *ESC* to close the window.)

```
ae_train exp_group/my_autoencoder -d
```

This command does not start training and should be run on a PC with a display connected.

Output:

3. Train the model (See the Headless Rendering section if you want to train directly on a server without display)

```
ae_train exp_group/my_autoencoder
```

```
$AE_WORKSPACE_PATH/experiments/exp_group/my_autoencoder/train_figures/training_images_29999.png
```

Middle part should show reconstructions of the input object (if all black, set higher bootstrap_ratio / auxilliary_mask in training config)

4. Create the embedding

```
ae_embed exp_group/my_autoencoder
```

Testing

Augmented Autoencoder only

have a look at /auto_pose/test/

Feed one or more object crops from disk into AAE and predict 3D Orientation

```
python aae_image.py exp_group/my_autoencoder -f /path/to/image/file/or/folder
```

The same with a webcam input stream

```
python aae_webcam.py exp_group/my_autoencoder
```

Multi-object RGB-based 6D Object Detection from a Webcam stream

Option 1: Train a RetinaNet Model from https://github.com/fizyr/keras-retinanet

adapt $AE_WORKSPACE_PATH/eval_cfg/aae_retina_webcam.cfg

```
python auto_pose/test/aae_retina_webcam_pose.py -test_config aae_retina_webcam.cfg -vis
```

Option 2: Using the Google Detection API with Fixes

Train a 2D detector following https://github.com/naisy/train_ssd_mobilenet
adapt /auto_pose/test/googledet_utils/googledet_config.yml

```
python auto_pose/test/aae_googledet_webcam_multi.py exp_group/my_autoencoder exp_group/my_autoencoder2 exp_group/my_autoencoder3
```

Evaluate a model

For the evaluation you will also need https://github.com/thodan/sixd_toolkit + our extensions, see sixd_toolkit_extension/help.txt

Create the evaluation config file

```
mkdir $AE_WORKSPACE_PATH/cfg_eval/eval_group
cp $AE_WORKSPACE_PATH/cfg_eval/eval_template.cfg $AE_WORKSPACE_PATH/cfg_eval/eval_group/eval_my_autoencoder.cfg
gedit $AE_WORKSPACE_PATH/cfg_eval/eval_group/eval_my_autoencoder.cfg
```

Evaluate and visualize 6D pose estimation of AAE with ground truth bounding boxes

Set estimate_bbs=False in the evaluation config

```
ae_eval exp_group/my_autoencoder name_of_evaluation --eval_cfg eval_group/eval_my_autoencoder.cfg
e.g.
ae_eval tless_nobn/obj5 eval_name --eval_cfg tless/5.cfg
```

Evaluate 6D Object Detection with a 2D Object Detector

Set estimate_bbs=True in the evaluation config

Generate a training dataset for T-Less using detection_utils/generate_sixd_train.py

```
python detection_utils/generate_sixd_train.py
```

Train https://github.com/fizyr/keras-retinanet or https://github.com/balancap/SSD-Tensorflow

```
ae_eval exp_group/my_autoencoder name_of_evaluation --eval_cfg eval_group/eval_my_autoencoder.cfg
e.g.
ae_eval tless_nobn/obj5 eval_name --eval_cfg tless/5.cfg
```

Config file parameters

```yaml [Paths]

# Path to the model file. All formats supported by assimp should work.
```

Tested with ply files.

MODEL_PATH: /path/to/my_3d_model.ply

Path to some background image folder. Should contain a * as a placeholder for the image name.

BACKGROUND_IMAGES_GLOB: /path/to/VOCdevkit/VOC2012/JPEGImages/*.jpg

[Dataset]

cad or reconst (with texture)

MODEL: reconst

Height of the AE input layer

H: 128

Width of the AE input layer

W: 128

Channels of the AE input layer (default BGR)

C: 3

Distance from Camera to the object in mm for synthetic training images

RADIUS: 700

Dimensions of the renderered image, it will be cropped and rescaled to H, W later.

RENDER_DIMS: (720, 540)

Camera matrix used for rendering and optionally for estimating depth from RGB

K: [1075.65, 0, 720/2, 0, 1073.90, 540/2, 0, 0, 1]

Vertex scale. Vertices need to be scaled to mm

VERTEX_SCALE: 1

Antialiasing factor used for rendering

ANTIALIASING: 8

Padding rendered object images and potentially bounding box detections

PAD_FACTOR: 1.2

Near plane

CLIP_NEAR: 10

Far plane

CLIP_FAR: 10000

Number of training images rendered uniformly at random from SO(3)

NOOF_TRAINING_IMGS: 10000

Number of background images that simulate clutter

NOOF_BG_IMGS: 10000

[Augmentation]

Using real object masks for occlusion (not really necessary)

REALISTIC_OCCLUSION: False

Maximum relative translational offset of input views, sampled uniformly

MAX_REL_OFFSET: 0.20

Random augmentations at random strengths from imgaug library

CODE: Sequential([#Sometimes(0.5, PerspectiveTransform(0.05)), #Sometimes(0.5, CropAndPad(percent=(-0.05, 0.1))), Sometimes(0.5, Affine(scale=(1.0, 1.2))), Sometimes(0.5, CoarseDropout(p=0.2, size_percent=0.05)), Sometimes(0.5, GaussianBlur(1.2*np.random.rand())), Sometimes(0.5, Add((-25, 25), per_channel=0.3)), Sometimes(0.3, Invert(0.2, per_channel=True)), Sometimes(0.5, Multiply((0.6, 1.4), per_channel=0.5)), Sometimes(0.5, Multiply((0.6, 1.4))), Sometimes(0.5, ContrastNormalization((0.5, 2.2), per_channel=0.3))], random_order=False)

[Embedding]

for every rotation save rendered bounding box diagonal for projective distance estimation

EMBED_BB: True

minimum number of equidistant views rendered from a view-sphere

MIN_N_VIEWS: 2562

for each view generate a number of in-plane rotations to cover full SO(3)

NUM_CYCLO: 36

[Network]

additionally reconstruct segmentation mask, helps when AAE decodes pure blackness

AUXILIARY_MASK: False

Variational Autoencoder, factor in front of KL-Divergence loss

VARIATIONAL: 0

Reconstruction error metric

LOSS: L2

Only evaluate 1/BOOTSTRAP_RATIO of the pixels with highest errors, produces sharper edges

BOOTSTRAP_RATIO: 4

regularize norm of latent variables

NORM_REGULARIZE: 0

size of the latent space

LATENT_SPACE_SIZE: 128

number of filters in every Conv layer (decoder mirrored)

NUM_FILTER: [128, 256, 512, 512]

stride for encoder layers, nearest neighbor upsampling for decoder layers

STRIDES: [2, 2, 2, 2]

filter size encoder

KERNEL_SIZE_ENCODER: 5

filter size decoder

KERNEL_SIZE_DECODER: 5

[Training] OPTIMIZER: Adam NUM_ITER: 30000 BATCH_SIZE: 64 LEARNING_RATE: 1e-4 SAVE_INTERVAL: 5000

[Queue]

number of threads for producing augmented training data (online)

NUM_THREADS: 10

preprocessing queue size in number of batches

QUEUE_SIZE: 50