datasets.py: used for create the dataset object for training. In this file we add a class for our dataset "SocialAI"

train.py: code for training that can be used also for performing fine tuning.

test.py: test the trained model.

leave_one_out_eval.py: type of valildation, google search

utils: some functions used in different files

model.py: class that define the l2cs model

demo.py: original demo from l2cs github code

demo_local.py: prediction of l2cs on given images

demo_pepper.py: real time mirror experiment

demo_ft_lin: use finetuned l2cs model to predict gaze

dataset_local_lin: use new dataset

datasets_local_no_par1315: use new dataset without participant 13 and 15

demo_local_folder: run demo in folder

demo_pepper_lin: gaze following real time experiment changed code

demo_pepper2: gaze following real time experiment version2

look_robot_aoi_action: real time experiment : robot detect which AOI of robot body I am looking at then do some action

look_robot_aoi: robot detect which AOI of robot body I am looking at then say name

look_robot_or_not: robot detect if I am looking at it

test_local_lin

train_local_lin_newdata_l

train_local_lin_newdata

train_local_lin

epoch_result: result of train

COMMANDS to run l2cs pretrained

demo with l2cs trained on Gaze360:

python3 demo_local.py --snapshot models/Gaze360-20220914T091057Z-001/Gaze360/L2CSNet_gaze360.pkl

NOTE: when you train with MPII remember to change number of bins and pitch yaw prediction transformation

demo with l2cs trained on MPIIGaze:

python3 demo_local.py --snapshot models/MPIIGaze-20220914T091058Z-001/MPIIGaze/fold1.pkl

demo on pepper streming data with l2cs trained on Gaze360:

python3 demo_pepper.py --ip=10.15.3.25 --port=12345 --cam_id=4 --snapshot models/Gaze360-20220914T091057Z-001/Gaze360/L2CSNet_gaze360.pkl

demo on pepper streming data with l2cs trained on MPIIGaze:

python3 demo_pepper.py --ip=192.168.0.167 --port=12345 --cam_id=4 --snapshot models/MPIIGaze-20220914T091058Z-001/MPIIGaze/fold1.pkl

GPU - CPU changes: map_location, cuda, gpu_id, .cpu, batch_size

-------- how to do fine tuning ----------

python train_local.py --dataset socialai

-------- time of execution ----------

[video_1, MPIIGaze, CPU] = 52.0 s

[video_1, MPIIGaze, GPU, batch_size 16] = 44.7

[video_1, MPIIGaze, GPU, batch_size 32] = 44.5

fine-tuning

Dataset composition

We trained the models using two different datasets: the first one contains only standard low-quality pictures while the second with both 4k images and low-quality pictures from the webcam. From this training set, a subset was extracted and employed as a validation set. For the test set instead, we decided to use only pictures from the Pepper monocular camera. 4K images are not used at test time since at deployment time (at least in our experiment) the model will predict gazes using its built-in monocular camera. Thus, using in the test set just images from the built-in camera we can have a better estimation of the real-time performance

of the model. On the other hand, incorporating the 4k images in the training set can be beneficial to increase the dataset size.

Furthermore, higher-quality images can potentially offer more informative cues than low-quality ones at training time allowing to generate more robust and accurate models.
The dataset for the training consist in three set: the training, the validation and the test set. The training set contains the data of the participant 1,2,4,5,7,8,9,14,15,18,19,20,21. the validation set instead constains the data of just two participant 12 and 13. The test instead contains the data of the participant 3,6,10,11,16,17. It's important to notice that the test set contains always only low quality images while training and validation set have both low and high. The idea behind is that the low-quality images are the one that the robot and the models will see at deploy time, since it will use the built in monocular camera of the robot.

Updates on the model

1 Freeze all the network and train only the last two fully connected layers;
2 We freeze the first convolutional layer and we train the last two fully connected layers and four central convolutional layers;

Moreover, exploit drop-out layers as regularization techniques to preserve the generalization capability of the model. Particularly, we added four drop-out layers after each of the convolutional layer of the L2CS model as a way to avoid the network from overfitting.