NvGaze Training Guide

NVIDIA CORPORATION

1. License Information for NvGaze Training Code

2. Pipeline description

- Create dataset
    - Capture real data and create labels (including gaze/pupil information)
    - Render synthetic data (lit images + region maps)
- Process data to generate h5 file including gaze/pupil information
- Run training (pytorch) with augmentation to increase variability in data
- Run inference (pytorch) to evaluate model
- serialize pytorch model to binary TensorRT model
- Potentially reduce accuracy of model (fp32 to fp16 or int8)
- Run accelerated CUDA-based inference on TensorRT model

3. Install Dependencies

- Python > 3.7, tested Python 3.9.12 / 3.10.4

- Install CUDA 11.3
- Dependencies

```
pip install PyQt5
pip install numpy
pip install opencv-python
pip install h5py
pip install matplotlib
pip install scipy
pip install pandas
pip install psutil
pip install imageio
pip install torch torchvision torchaudio --extra-index-url
https://download.pytorch.org/whl/cu113
pip install tensorboard
pip install --upgrade tensorflow
```

check https://www.tensorflow.org/install/pip

We assume that all scripts used in this tutorial are located in C:\NvGaze\Training. For other paths please change the directories where appropriate.

4. Dataset Generation

- Render dataset using NvGaze model
  - Generates images with labels for gaze direction and eye ball position and camera position
- Create pupil / iris labels for rendered images without such annotations
  - Could be potentially be rendered using emissive points at target location
  - Alternatively use script to write pupil/iris/eyeball info
    `python convert_raw_to_pupil_iris_eyeball.py`

- Generate h5 file from rendered data + labels for one or many datasets
  - `python createDataset_sample.py`

For the example dataset you should see the following output in case of success

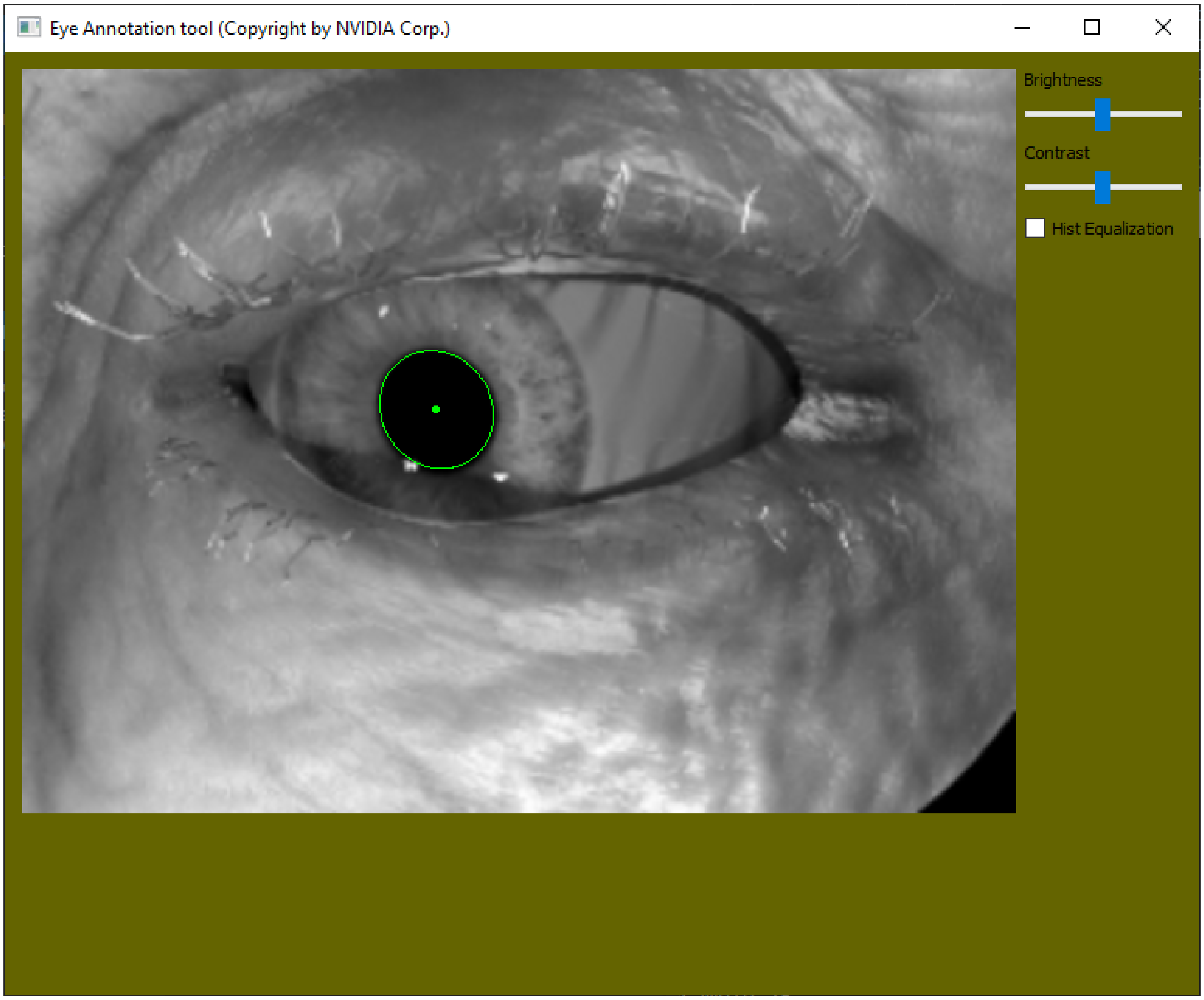
…

```
   image 1595 / 1600
   image 1596 / 1600
   image 1597 / 1600
   image 1598 / 1600
   image 1599 / 1600
   image 1600 / 1600
number of frames written for dataset :  1600
```

```
Writing label info for selected samples...
writing label  archive
writing label  image_L_0
writing label  regionmask_withskin_L_0
writing label  regionmask_withoutskin_L_0
writing label  string_eye
writing label  string_original_eye_x
writing label  string_original_eye_y
writing label  string_original_eye_z
writing label  float_blink
writing label  float_gaze_x_degree
writing label  float_gaze_y_degree
writing label  float_pupil_size
writing label  float_slippage_x
writing label  float_slippage_y
writing label  float_slippage_z
writing label  float_pupil_x
writing label  float_pupil_y
writing label  float_pupilellipse_major
writing label  float_pupilellipse_minor
writing label  float_pupilellipse_angle
writing label  float_iris_x
writing label  float_iris_y
writing label  float_irisellipse_major
writing label  float_irisellipse_minor
writing label  float_irisellipse_angle
writing label  float_eyeball_x
writing label  float_eyeball_y
writing label  float_eyeball_diameter
done with h5 file : …/Datasets/active\small_example_dataset.h5

H5 file created in  19  seconds
```

You can use the `h5Explorer.py` or the `eyeAnnoationTool.py` to inspect the newly generated h5 dataset. The example dataset includes accurate pupil positions for the synthetic eye model.



- Generate h5 file from recorded videos/images
    - convertVideoToH5Dataset.py
    - then add labels using dataset annotation tool

- Create combined data for synthetic + real data
    - mergeH5Files.py

- Create reflection dataset for augmentation
    - Use “create_reflection_dataset.py”
        - We suggest using a dataset like http://web.mit.edu/torralba/www/indoor.html
        - The python script allows you to limit the number of used images (for example 500)
        - scripts generates a file ‘reflectiondataset.h5’ that is used later for augmentation during training

- You can inspect the images in the dataset using the `H5Explorer.py`

5. Dataset Annotation

Start eye annotation application

```
python eyeAnnotationTool.py
```

The application loads the sample dataset (sampleData_offaxis.h5) by default. You can load any valid h5 container but dragging the h5 file into the application GUI.
The h5 container is assumed to use the following labels for the pupil.

float_pupil_x
float_pupil_y
float_pupilellipse_major
float_pupilellipse_minor
float_pupilellipse_angle

The annotation works as follows:

- activate one of the annotation modes: pupil, iris, eyeball
- iterate through the individual samples using W,S,A,D
- create/remove/retarget fit points for ellipse fit
- Perform ellipse fit
- Mark samples as valid and invalid/rejected
- Save dataset at anytime or at application shutdown

Valid samples are highlighted green, rejected samples are red, unprocessed samples are brown.
The application output is written to the console.
There are GUI elements for brightness and contrast adjustment and histogram equalization for better visibility.

The application uses the following interaction methods.

Mouse interaction

- Left click: create fitting point on target contour
- Left drag: move existing fitting point
- Right click: remove fitting point
- Shift + Left mouse drag: move fitting points group

Keyboard interaction

- Shift: fit/refit ellipse for existing fitting points
- 1: Pupil annotation mode
- 2: Iris annotation mode
- 3: Eyeball annotation mode
- R: Reject sample
- T: Take (Accept) sample
- A: jump to previous sample
- D: jump to next sample
- W: jump forward 50 samples in dataset
- S: jump backwards 50 samples in dataset

6. Training

Start by creating a training specification script. We provide an example script called `sample_pupil_synthetic.py`

The training script specifies

- the training data to train and test on
- image input resolution
- network type
- output label description
- loss function
- dataset loader script
- learning function specs

We also provide a sample Dataset class that includes the training data loading logic and data augmentation and is configured by the previously described training specification script. The provided sample dataset is contained in the file `sample_pupil_synthetic.py`

Finally the module `trainPupil.py` includes the training logic and is started.by the training run script `runTraining_sample.py`.

After configuring the required scripts start the training procedure by running

```
python sample_runTrainingPupil.py -c
../experiments/sample_exp_offaxis_synthetic.py -e
exp_sample_pupilcenter -m local
```

The script will start the job and if successful start reporting the loss per epoch:

```
2022-03-29 20:12:32.735011: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 21654 MB memory:
-> device: 0, name: NVIDIA GeForce RTX 3090, pci bus id:
0000:01:00.0, compute capability: 8.6
  Training Epoch 1:   Running Time- 28.49    Training Loss (RMSE)-
120.5326    Test Loss (RMSE)- 16.8105
  Training Epoch 2:   Running Time- 23.78    Training Loss (RMSE)-
45.0638    Test Loss (RMSE)- 21.0772
  Training Epoch 3:   Running Time- 25.19    Training Loss (RMSE)-
38.2578    Test Loss (RMSE)- 19.6023
```

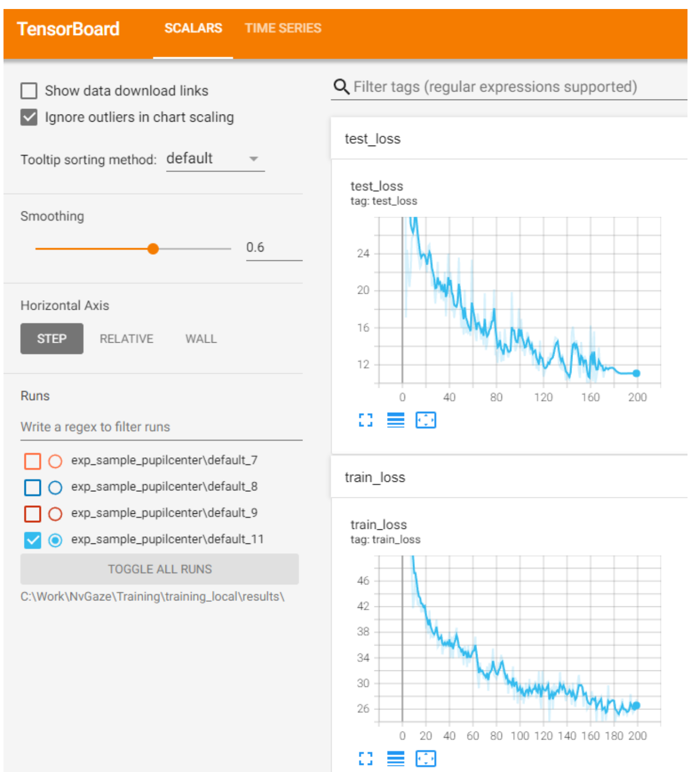
```
  Training Epoch 4:   Running Time- 27.41    Training Loss (RMSE)-
36.3651    Test Loss (RMSE)- 18.1385
  Training Epoch 5:   Running Time- 24.87    Training Loss (RMSE)-
35.9118    Test Loss (RMSE)- 23.6875
  Training Epoch 6:   Running Time- 29.58    Training Loss (RMSE)-
33.9929    Test Loss (RMSE)- 15.9358
  Training Epoch 7:   Running Time- 28.38    Training Loss (RMSE)-
33.3854    Test Loss (RMSE)- 21.3046
```

Then run Tensorboard to enable live training progress and analysis

`tensorboard --logdir=C:\NvGaze\results\`

Open Tensorboard live view in browser

http://localhost:6006/



At the start of the training process all required scripts are collectively archived and copied into a zip file into the training output folder (specified in the training run script) and a tensorboard log file is created.

During training the process writes out the latest and in addition the best weights with respect to the loss function into .pth files located in the training output folder.

It is possible to change training script parameters from the command line.

This is helpful to start many training runs for hyperparameter optimization.
-m, --mode, default = 'local' : 'local' runs training locally (this is the default). 'remote' submits training job(s) to a cluster.

-v, --var, nargs='*', action='append' : A variable and value pair or variable and range for multiple submits

-e, --experiment_name, default = 'test' : Describe the experiment. If not given, it will assume you are testing your code and will put the dump files under [your network drive]/results/test/

-l, --experiment_detail, default = '' : Experiment specifier for run description

-c, --config_path, default='config.py' : Path to the config file to be used.

-j, --job_name, default= None : The name of the job to be submitted.

The following call overrides the variables `strides`, `kernel_sizes`, `output_channel_counts` and `input_resolution` using the `-v` parameter.

```
python sample_runTrainingPupil.py -c
../experiments/sample_exp_offaxis_synthetic.py -e
exp_sample_pupilcenter -m local -v strides 2 2 2 2 2 2 -v
kernel_sizes 3 3 3 3 3 3 -v output_channel_counts 32 48 72 104 160
248 -v input_resolution 127 127
```