Violence detection operation manual (version 2)

0. System requirement

Operation system: Ubuntu 14.04 or Ubuntu 16.04
GPU memory: >8GB
Storage: >80GB

1. Installation

a). all you need
You can download all the code from Lisa Anne's Github:
https://github.com/LisaAnne/lisa-caffe-public
Datasets UCF101 and basic instructions are given in this website:
https://people.eecs.berkeley.edu/~lisa_anne/LRCN_video

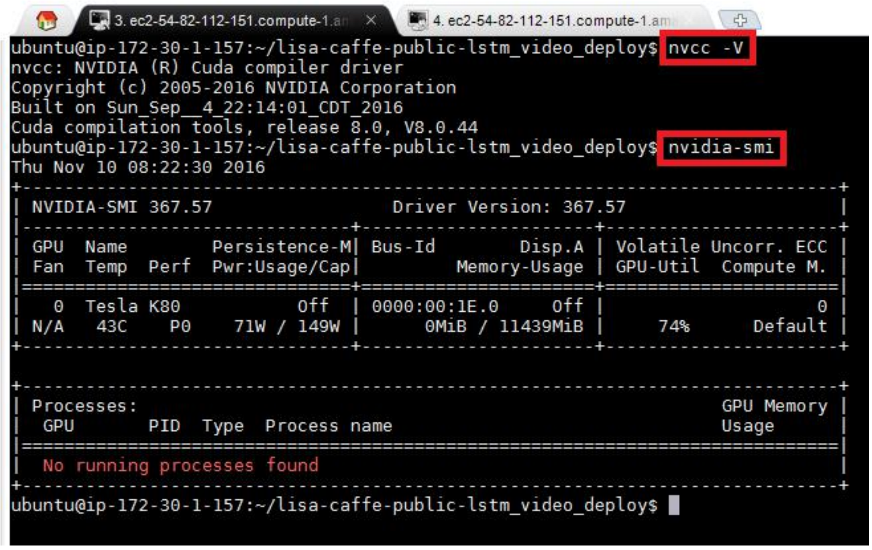
b). dependencies
The whole structure is based on caffe. To install caffe, you need to install the following dependencies as suggested here:
http://caffe.berkeleyvision.org/installation.html

-CUDA

CUDA 8.0 is available for Ubuntu 14.04 and 16.04. Here we take Ubuntu 16.04 as an example.
Download runfile(local) CUDA8.0 from:
https://developer.nvidia.com/cuda-downloads
Then follow the instruction on blog:
http://www.52nlp.cn/%E6%B7%B1%E5%BA%A6%E5%AD%A6%E4%B9%A0%E4%B8%BB%E6%9C%BA%E7%8E%AF%E5%A2%83%E9%85%8D%E7%BD%AE-ubuntu-16-04-nvidia-gtx-1080-cuda-8

After installation, you may use these command line to verify whether your CUDA is ready:
nvcc -V
Nvidia-smi

Your installation is successful while you see this:



-CUDNN

The accelerator of CUDA, it's good to have it but not necessarily.

-BLAS

There are many versions of BLAS. The default by Makefile.config is atlas.

-OpenCV

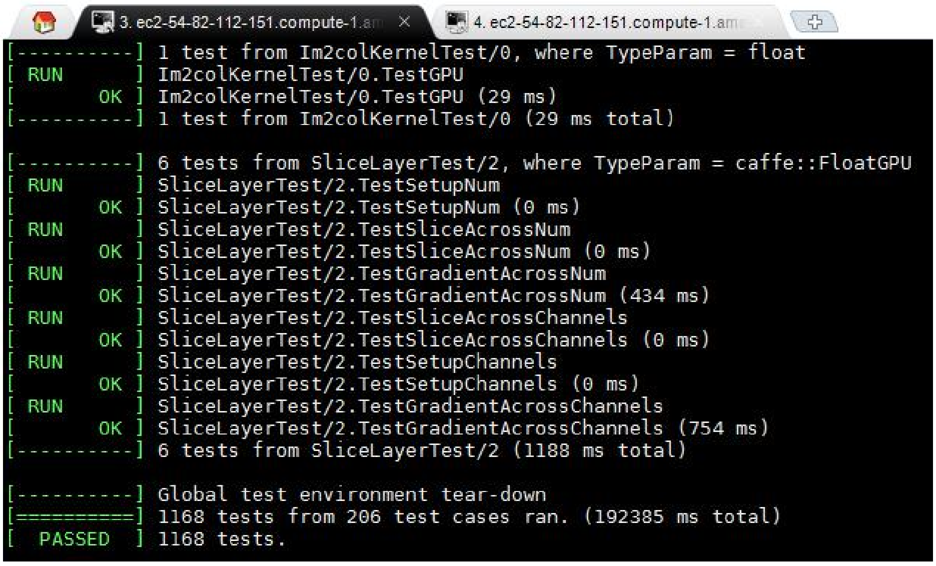
The OpenCV is usually installed previously. To check the version that you have, you may use the following line. But if it is not installed, just follow the instructions online.
python
>>> import cv2
>>> cv2.__version__
>>> exit()

-Python2.7
Python 2 and python 3 both works for this project. Here we use python 2.7.

-ffmpeg
This is to extract the frames from the video clips.

c). compilation
The instruction is here:
http://caffe.berkeleyvision.org/installation.html

More specific:
cp Makefile.config.example Makefile.config
adjust Makefile.config settings
make all
make pycaffe
make test
make runtest
No need to do CMake Build. When no error occurs during the whole procedure, and the make runtest gives all green OK, congratulations! Your system is ready for training now. But you will find the compilation frustrating and time consuming. Just be patient and learn to use google to shoot problems. 99% of the cases happened before.



```
[----------] 1 test from Im2colKernelTest/0, where TypeParam = float
[ RUN      ] Im2colKernelTest/0.TestGPU
[       OK ] Im2colKernelTest/0.TestGPU (29 ms)
[----------] 1 test from Im2colKernelTest/0 (29 ms total)

[----------] 6 tests from SliceLayerTest/2, where TypeParam = caffe::FloatGPU
[ RUN      ] SliceLayerTest/2.TestSetupNum
[       OK ] SliceLayerTest/2.TestSetupNum (0 ms)
[ RUN      ] SliceLayerTest/2.TestSliceAcrossNum
[       OK ] SliceLayerTest/2.TestSliceAcrossNum (0 ms)
[ RUN      ] SliceLayerTest/2.TestGradientAcrossNum
[       OK ] SliceLayerTest/2.TestGradientAcrossNum (434 ms)
[ RUN      ] SliceLayerTest/2.TestSliceAcrossChannels
[       OK ] SliceLayerTest/2.TestSliceAcrossChannels (0 ms)
[ RUN      ] SliceLayerTest/2.TestSetupChannels
[       OK ] SliceLayerTest/2.TestSetupChannels (0 ms)
[ RUN      ] SliceLayerTest/2.TestGradientAcrossChannels
[       OK ] SliceLayerTest/2.TestGradientAcrossChannels (754 ms)
[----------] 6 tests from SliceLayerTest/2 (1188 ms total)

[----------] Global test environment tear-down
[==========] 1168 tests from 206 test cases ran. (192385 ms total)
[  PASSED  ] 1168 tests.
```

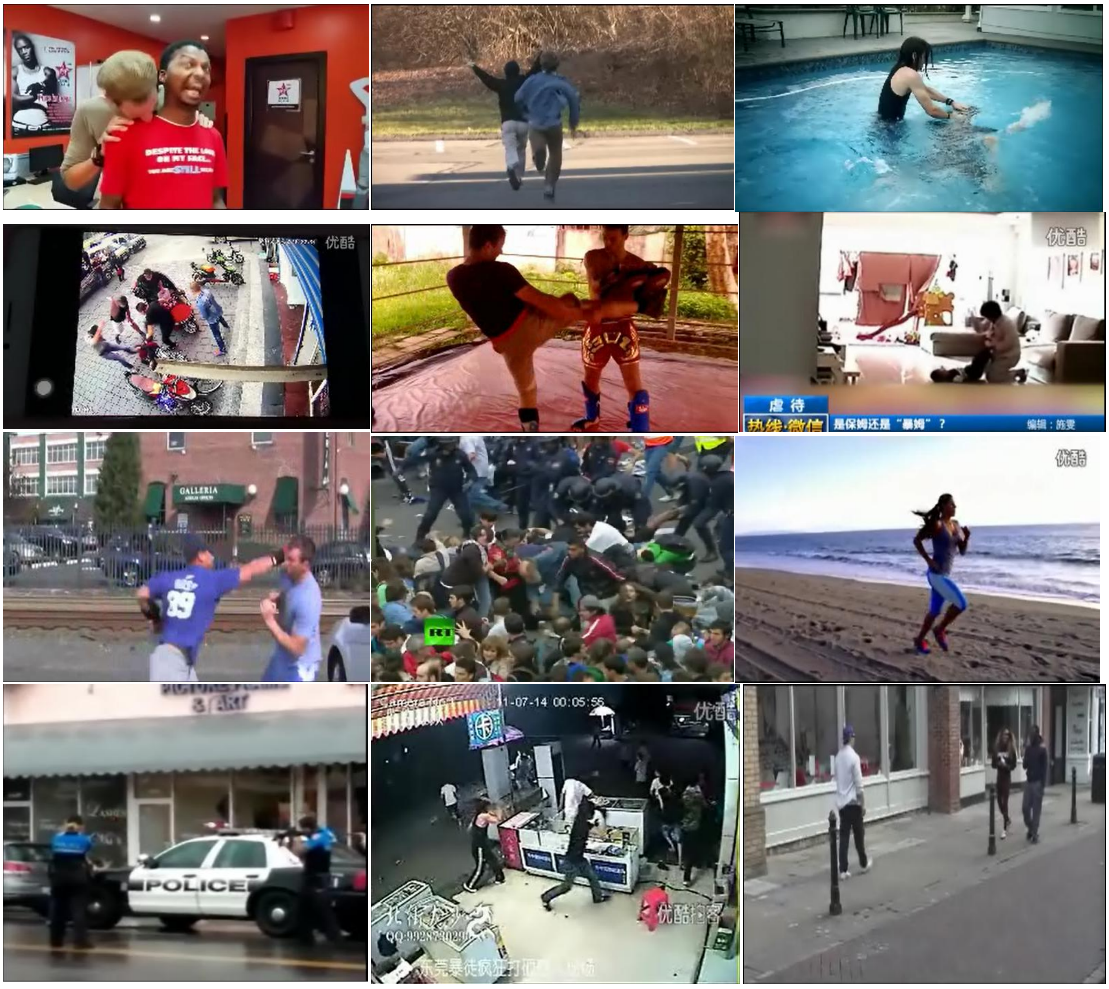
2. data description

The original training is done with the dataset UCF101, which includes 101 labeled human activities. Each label contains around 100 clips, each clip is about 5-10 second long. The format of the clip is avi, with the size of 320x240 pixel. 70% of the clips will be used as training, while 30% will be used as testing.

Frames will be extracted from the clips. Current extract rate is 30 frames/second. After extraction, we will calculate the optical flow images from the frames. Finally these two folders will be taken as training and testing material, frames and optical flow images.

For our newly released human aggression dataset, there are 4 labels: punch, wrestling, group fighting, riot, and 2 reference labels: walking, running. Each label contains 60 clips, each clip is about 3-10 seconds long. The format and training deployment is the same as original.

Frames:



Optical flow:

3. Deployment

a). To deploy the system and to fit the training environment that you desire, please understand the function of each file under folder:
~/lisa-caffe-public-lstm_video_deploy/examples/LRCN_activity_recognition

> **-action_hash_rev.p**
> Mapping file, each item corresponding to one label of the activities. Change the name and label number if needed.

-caffe_imagenet_hyb2_wr_rc_solver_sqrt_iter_310000
Pretrained model, it is a hybrid between the reference Caffe net and the network used by Zeiler & Fergus. You can either find it in local folder or download it here:
https://people.eecs.berkeley.edu/~lisa_anne/LRCN_video_weights.html

-classify_video.py
This is the cold to classify video after training.

-create_flow_images_LRCN.m
Matlab file to generate flow images from frames. The matlab can't run on EC2 instance, so for this step we need to run locally.

-deploy_lstm.prototxt deploy_singleFrame.prototxt
Files to deploy layers. Modify it if you wish to change the layer feature.

-extract_frames.sh
To extract frames from video clips.

-lstm_solver_flow.prototxt lstm_solver_RGB.prototxt singleFrame_solver_flow.prototxt singleFrame_solver_RGB.prototxt
These files deploy the setting of training, eg maximum iteration, step size.

-run_lstm_flow.sh run_lstm_RGB.sh run_singleFrame_flow.sh run_singleFrame_RGB.sh
Bash file to start training. Use the following command line to start:
chmod +x run_lstm_flow.sh
./run_sltm_flow.sh

-sequence_input_layer.py
Data layer for video. Change flow_frames and RGB_frames to be the path to the flow and RGB frames.

-train_test_lstm_flow.prototxt train_test_lstm_RGB.prototxt train_test_singleFrame_flow.prototxt train_test_singleFrame_RGB.prototxt
Change the batch size here if it is too large for the GPU memory.

-ucf101_singleFrame_flow_test_split1.txt ucf101_singleFrame_flow_train_split1.txt ucf101_singleFrame_RGB_test_split1.txt ucf101_singleFrame_RGB_train_split1.txt
Namelist of the frames and flow images, error occurs when the program can not find corresponding image.

-ucf101_split1_testVideos.txt ucf101_split1_trainVideos.txt
Namelist of the videos, error occurs when the program can not find corresponding video clip.

b). file preparation
Most of the files are listed enclose the package. In case you want to involve your own data, here are the files you need to modify.

-action_hash_rev.p
Change the name and label number if needed.

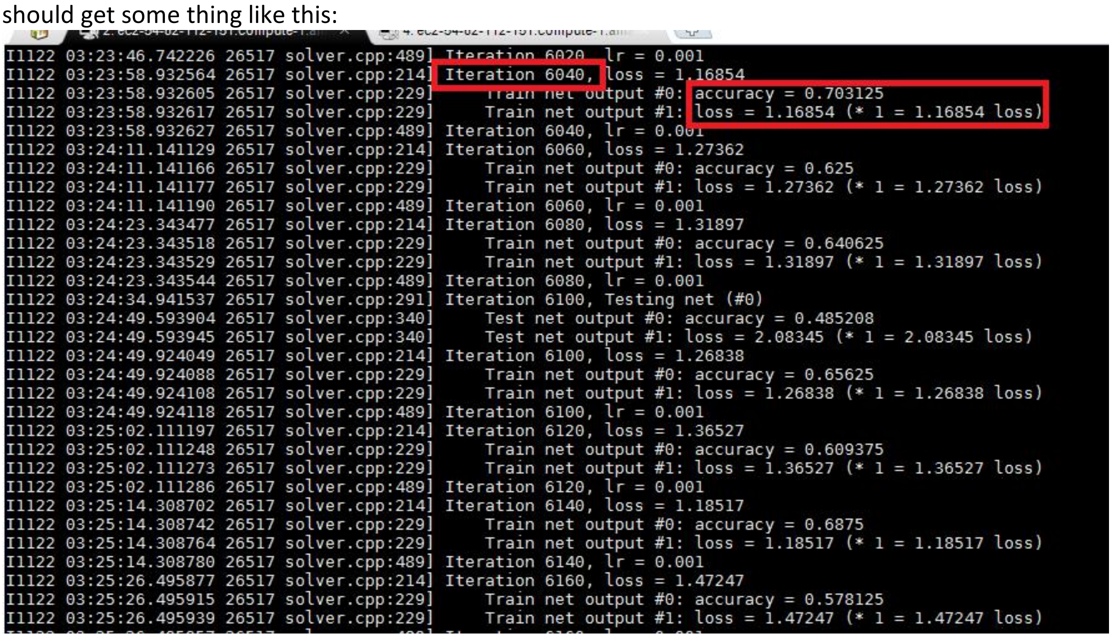
-create_flow_images_LRCN.m
The mex files are missing in the original package but I have put it in mine. There is another method to create the flow image by python, it code is also enclosed. Play with the parameters to get the best performance.

-ucf101_singleFrame_flow_test_split1.txt ucf101_singleFrame_flow_train_split1.txt ucf101_singleFrame_RGB_test_split1.txt ucf101_singleFrame_RGB_train_split1.txt &ucf101_split1_testVideos.txt ucf101_split1_trainVideos.txt

Caution! Collecting the dataset and make the namelist is the most frustrating part of work which will drive you crazy. Take a cup of coffee and a deep breath, this work can take you up to days.
If you have a new dataset, list all the names as in example:
v_Rowing_g20_c03/flow_image_v_Rowing_g20_c03.0014.jpg 75
v_Rowing_g20_c03/ is the name of the folder
flow_image_v_Rowing_g20_c03.0014.jpg is the image name
75 is the label number
The namelist file is huge that sometimes it exceed the capacity of excel. Highly recommended to separate it into 2 to avoid lost of data.
The name needs to be totally well mixed, or the training performance will be poor.
You need at lease 3 labels to launch the training.

4. training

Now you are good to train the model. By running ./run_singleFrame_flow.sh, after 5 minutes you should get some thing like this:



```
I1122 03:23:46.742226 26517 solver.cpp:489] Iteration 6020, lr = 0.001
I1122 03:23:58.932564 26517 solver.cpp:214] Iteration 6040, loss = 1.16854
I1122 03:23:58.932605 26517 solver.cpp:229]     Train net output #0: accuracy = 0.703125
I1122 03:23:58.932617 26517 solver.cpp:229]     Train net output #1: loss = 1.16854 (* 1 = 1.16854 loss)
I1122 03:23:58.932627 26517 solver.cpp:489] Iteration 6040, lr = 0.001
I1122 03:24:11.141129 26517 solver.cpp:214] Iteration 6060, loss = 1.27362
I1122 03:24:11.141166 26517 solver.cpp:229]     Train net output #0: accuracy = 0.625
I1122 03:24:11.141177 26517 solver.cpp:229]     Train net output #1: loss = 1.27362 (* 1 = 1.27362 loss)
I1122 03:24:11.141190 26517 solver.cpp:489] Iteration 6060, lr = 0.001
I1122 03:24:23.343477 26517 solver.cpp:214] Iteration 6080, loss = 1.31897
I1122 03:24:23.343518 26517 solver.cpp:229]     Train net output #0: accuracy = 0.640625
I1122 03:24:23.343529 26517 solver.cpp:229]     Train net output #1: loss = 1.31897 (* 1 = 1.31897 loss)
I1122 03:24:23.343544 26517 solver.cpp:489] Iteration 6080, lr = 0.001
I1122 03:24:34.941537 26517 solver.cpp:291] Iteration 6100, Testing net (#0)
I1122 03:24:49.593904 26517 solver.cpp:340]     Test net output #0: accuracy = 0.485208
I1122 03:24:49.593945 26517 solver.cpp:340]     Test net output #1: loss = 2.08345 (* 1 = 2.08345 loss)
I1122 03:24:49.924049 26517 solver.cpp:214] Iteration 6100, loss = 1.26838
I1122 03:24:49.924088 26517 solver.cpp:229]     Train net output #0: accuracy = 0.65625
I1122 03:24:49.924108 26517 solver.cpp:229]     Train net output #1: loss = 1.26838 (* 1 = 1.26838 loss)
I1122 03:24:49.924118 26517 solver.cpp:489] Iteration 6100, lr = 0.001
I1122 03:25:02.111197 26517 solver.cpp:214] Iteration 6120, loss = 1.36527
I1122 03:25:02.111248 26517 solver.cpp:229]     Train net output #0: accuracy = 0.609375
I1122 03:25:02.111273 26517 solver.cpp:229]     Train net output #1: loss = 1.36527 (* 1 = 1.36527 loss)
I1122 03:25:02.111286 26517 solver.cpp:489] Iteration 6120, lr = 0.001
I1122 03:25:14.308702 26517 solver.cpp:214] Iteration 6140, loss = 1.18517
I1122 03:25:14.308742 26517 solver.cpp:229]     Train net output #0: accuracy = 0.6875
I1122 03:25:14.308764 26517 solver.cpp:229]     Train net output #1: loss = 1.18517 (* 1 = 1.18517 loss)
I1122 03:25:14.308780 26517 solver.cpp:489] Iteration 6140, lr = 0.001
I1122 03:25:26.495877 26517 solver.cpp:214] Iteration 6160, loss = 1.47247
I1122 03:25:26.495915 26517 solver.cpp:229]     Train net output #0: accuracy = 0.578125
I1122 03:25:26.495939 26517 solver.cpp:229]     Train net output #1: loss = 1.47247 (* 1 = 1.47247 loss)
```

Iteration indicates the progress of your training. Accuracy indicates the current accuracy and loss is an important index of how well the training goes. Usually accuracy is low and loss is high at beginning, but it evolves after more iteration. If your number is always poor, there is something wrong.

The training takes 20+ hours for UCF101, and for a small dataset with 3 labels it takes about 2 hours.

5. Classifying

Run classify_video.py, there are 4 models running and each will give their own opinion. Play with the weight and you can adjust to the best performance.

6 Happy coding!

Appendix: The results of experiment 20.11.2016

a). Project description:

Dataset is UCF101 combined with self collected dataset. The replacement is as follows:

bandMarching -> riot	5
WalkingWithDog -> running	97
BoxingPunchingBag -> groupfighting	16
BoxingSpeedBag -> Punching	17
Punch -> Walking	70

Flow images are trained on instance *new-deep-surveillance,* and RGB images are trained on local GTX1070.

b). Results

Single model:

Model 1: single frame flow
Iteration: 50000
Loss: 1.9244
Accuracy: 0.55

Model 2: lstm flow
Iteration: 30000
Loss: 1.64313
Accuracy: 0.58

Model 3: single frame RGB
Iteration: 7800
Loss: 1.46
Accuracy: 0.65

Model 4: lstm RGB
Iteration: 30000
Loss: 1.29
Accuracy: 0.66

All model combined:

Accuracy as in classifying 101 categories: 76%
Accuracy as in classifying violence&non-violence: 90%

You can find all the files under the path:

Instance: new-deep-surveillance
/home/ubuntu/lisa-caffe-public-lstm_video_deploy/examples/LRCN_activity_recognition

Local: GTX10701
/home/lisa_code/lisa-caffe-public-lstm_video_deploy/examples/LRCN_activity_recognition

Dataset here:
/2.0TB Volume/home/trainer/Ting/flow_images/
/2.0TB Volume/home/trainer/Ting/frames/