Predictive Frame Detection

Salman Alfarisyi (salmanalfarisyi087@gmail.com)

The following software contains two different algorithms from two different research groups, a STMFANet for camera frame prediction and a Deep-SAD for anomaly detection. The point of this is to create a camera fault detection and prediction system which can be used to detect and predict camera faults or obstructions such as snow, fog, or rain. The idea is for the STMFANet to predict the next few frames of the camera based on 8-10 previous frames and then detect obstructions within the predicted frames using the Deep-SAD network. Currently, both algorithms are still separate, but the intended goal is to combine the two.

Prerequisites:

Install the following libraries to use both algorithms:

- Python 3.7
- Imageio 2.9.0
- Opencv 3.4.2
- Matplotlib 3.1.2
- Numpy 1.17.0
- Pandas 1.2.4
- Pillow 6.0.0
- Pytorch 1.7.1
- Pytorch-wavelets 1.3.0
- Scikit-learn 0.21.2
- Scipy 1.6.2
- Seaborn 0.9.0
- Tensorboardx 2.2
- Torchaudio 0.7.2
- Torchvision 0.8.2
- Tqdm 4.62.0

Usage

STMFANet

There are 2 folders for the STMFANet frame prediction training: STFMANet-Color and STFMANet-Gray. The color folder contains a network that has been optimized for color images while the gray folder is for grayscale images. STMFANet folders contain training that has been optimized to work on two GPUs. This is because with the A2D2 dataset we are using, the amount of memory required to train this network is so much that it requires more than 1 GPU's worth of memory. Even then, the frames must be downscaled to almost 1/8th the original resolution just to be able to train in color. Hence there is a copy of this code in the server00 in the CAV lab under "sda/Salman/STMFANet/" where the network is trained.

As of now the data sets are fully completed and ready to be used for training, testing, and validation. They are in "sda/Sampo/datasets/A2D2/". This folder contains 3 camera and lidar datasets: Galmerscheim, Ingolstadt, and Munich. These 3 datasets correspond to 3 different cities from the A2D2 dataset. The one used for training is the Munich dataset and the Ingolstadt and Galmerscheim datasets are to be used for testing and validation respectively. Do note that these datasets are huge, and I recommend to not bother moving them or messing with them.

The “sda/Salman/STMFANet/” as well as within the “Predictive-Frame-Detection-main/Implementation” contains the tools needed to modify and create the datasets required for training, testing, and validation. However, I have gone through doing it all beforehand so there should be no need to do this step again. To learn more about how to create datasets and how to modify the configurations inside the network, I highly recommend reading the documentation written by Alireza in the “Predictive-Frame-Detection-main/Implementation” path of the repository.

To start training, if you are using the CAV server, please go to the Nvidia-X-server settings to see if anyone is using the GPUs for training and avoid using any GPUs that other people may be using. Take note of the GPU numbers that are free. Then in the train_a2d2.py file on either the color or grayscale version of the STMFANet, locate the line “os.environ["CUDA_VISIBLE_DEVICES"] = '1,2'”. This is an environment variable that sets what GPUs are visible to the code when it is run. Write the 2 free GPUs inside the string with a comma to separate them. Do note that the GPU number in python is different to the GPU numbers in Nvidia-X-server settings. GPU 0 in Nvidia-X-server settings is GPU 1 in Python and so on.

Once the GPUs are set, then find the train_options.py file in the options folder inside the STFMANet-Color/gray folder. inside contains a plethora of options that control the STFMANet prediction network. Feel free to change the arguments inside this file to whatever suits the experiment you are doing. They have helpful notes beside them to describe the function of each setting.

Additionally, in this setting, there is a way to set the network to run on pure CPU. By uncommenting “#self.opt.gpu_ids = []”, we can force the code to run on CPU only. Though I have never tried this after I have purposely split the model to run on 2 GPUs.

“self.opt.video_list = 'train_data_list.txt'” specifies the file which contains a list of paths that correspond to frames that are used to train the model. If you want to run the other datasets from other cities, I have provided a test_data_list.txt and validation_data_list.txt for Ingolstadt and Galmerscheim respectively. Just change the value of this variable to the name of the file you wish to use and make sure the list text files are in the data folder as well.

Once done setting things up, training can begin by executing the train_a2d2.py file. To monitor the progress of training, use the console to open Tensorboard and point to the folder with the same name as you experiment in the “tb” folder. Tensorboard should give you an IP address with a port number which you can copy to the web browser and monitor the loss graphs and the images from there. Tensorboard shouldn’t update automatically all the time but rather update every few minutes or so. How fast it updates can be changed in the train_options.py file.

For more information on how the algorithm works as well as other settings that you might be interested in, read Alireza’s documentation called “STFMANet document.pdf” which is in the “Predictive-Frame-Detection-main/Implementation” folder and the paper by Beibei Jin et. al. which can be found in this link (https://arxiv.org/pdf/2002.09905.pdf).

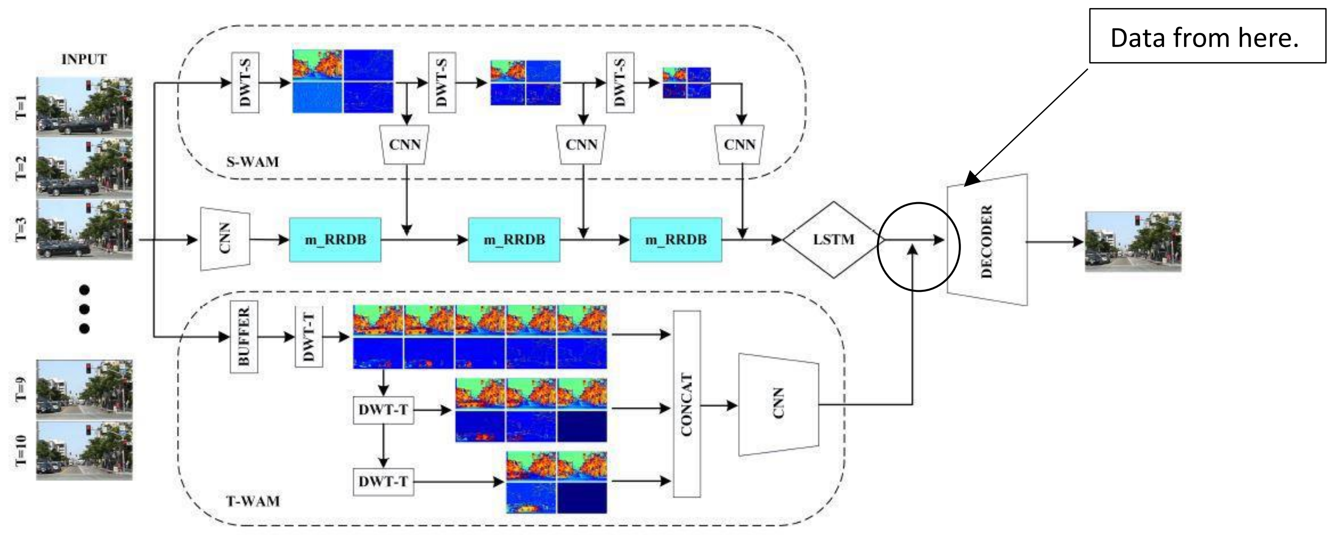
Deep-SAD

Deep-SAD is a neural network designed to detect anomalies in datasets and images. This is done by training the model in a semi-supervised manner where the model’s results are compared to a hypersphere center. The farther the distance from the result to the hypersphere center the more likely the data is anomalous. The model then creates a threshold where the data is declared as anomalous. To learn more about Deep-SAD, I recommend reading Ruff et. al. paper which can be found at this link (https://arxiv.org/pdf/1906.02694.pdf).

The code within this repository is directly cloned from Lukas Ruff's repository from the Deep-SAD paper which can be found at this link (https://github.com/lukasruff/Deep-SAD-PyTorch). The readme inside the Deep-SAD repository contains instructions on how to install and run experiments. To run the A2D2 dataset, simply run copy this line into the command line:

```
"python main.py a2d2 a2d2Detector ../log/DeepSAD/a2d2 ../data --ratio_known_outlier 0.01 --
ratio_pollution 0.1 --lr 0.0001 --n_epochs 150 --lr_milestone 50 --batch_size 128 --weight_decay
0.5e-6 --pretrain False --ae_lr 0.0001 --ae_n_epochs 150 --ae_batch_size 128 --ae_weight_decay
0.5e-3 --normal_class 0 --known_outlier_class 1 --n_known_outlier_classes 1 --seed 10"
```

The train and test datasets I have already created are not made from image frames. Rather this is made from the data created right after the LSTM and before the decoder step of the STFMANet as seen in the image below. This is to increase the amount of data available for the detector network and allow it to detect anomalies much easier.



As a result, creating this dataset and creating a compatible neural network for this task is not trivial and while I have figured out how to run their code to run our experiments, it is not fully finished yet. This is where you may have to experiment with it. The source code of the Deep-SAD folder contains two folders called networks and datasets that contain two files: a2d2detector.py and a2d2dataset.py respectively. A2d2detector.py contains a definition of a convolutional neural network that is used for anomaly detection. A2d2dataset.py contains the code needed to import the dataset to the training program so it can be used to train and test the model. Feel free to play around with the settings in these two files as well as playing around with the other arguments in the network.

If you have any questions, feel free to contact me.