

IEEE Video & Image Processing Cup 2017

Code Documentation

Team Markovians

Bangladesh University of Engineering and Technology

Supervisor:

Prof Mohammad Ariful **Haque**

Postgraduate Member:

Sayeed Shafayet **Chowdhury**

Undergraduate Team:

Ahmed **Maksud**

Nahian Ibn **Hassan**

Muhammad Suhail **Najeeb**

Jubaer **Hossain**

Shahruk **Hossain**

SM Raiyan **Chowdhury**

Shakib **Zaman**

Roknuzzaman **Rokon**

Kinjol **Barua**

Contents

1.	Training the System	
1.1.	Frame Extraction	3
1.2.	Challenge Type Classifier	
1.2.1.	Extracting Features	4
1.2.2.	Training RCNN	8
1.3.	Bounding Box Detector	
1.3.1.	Enlarging Ground Truth Boxes	9
1.3.2.	Training F-RCNN	10
1.4.	Sign Type Classifier	
1.4.1.	Extracting ROIs	14
1.4.2.	Extracting Features	16
1.4.3.	Training CNN	18
2.	Testing the System	
2.1.	System Summary	22
2.2.	Result Generation	25
3.	Dependencies	26

1.Training the System

The following sections details how the system was trained and how one can reproduce the same results. The system has three trainable neural networks:

- **Challenge Type Classifier** : A *Recurrent Convolutional Neural Network* (RCNN)
- **Bounding Box Detector** : An implementation of the *Faster Region-based Convolutional Neural Network combined with a Region Proposal Network* (F-RCNN with RPN)
- **Sign Type Classifier** : A custom *Convolutional Neural Network* (CNN)

Each of these networks were trained separately using extracted frames from videos belonging to the trainset.

(*Note : We opted to extract the frames and save them to disk instead of feeding videos directly into the code mainly because initially we had some problems reading the videos with our installation of openCV. We managed to fix the issue later but stuck with training the system using frames instead of video. This also had the added benefit of allowing us to more easily sample and process frames instead of having to load entire videos at a time.*)

The code files required for training can be found in the folder labelled 1.Training.

1.1 Frame Extraction

Inside the 1.Training folder, the python script labelled *extractFrames.py* was used to extract frames from all of the videos and save them to disk. It uses the Py-Av library to read the videos and save the frames to disk. The dependencies for this script are listed in the **Dependencies** section.

The directory from where videos will be read, and where the extracted frames will be saved must be edited in the script. The section marked as *Settings* has these parameters. (see fig.1)

The *loc* variable needs to be set to the directory containing the videos of the dataset, while the *dst* variable may be set to any other location where the frames will be saved. The parameters in lines 54 to 57 are used to select which videos to extract frames from. The default settings as shown in fig.1 will extract frames from all the videos in the dataset. The code uses multi-threading, and on a Intel Core i7 (4th gen and up) , using 7 threads, frame extraction can be done in about 3-4 hours at the most.

```
50 #----- Settings -----
51
52 max_threads = 7 # number of maximum active threads at a given time
53
54 N = [1,49]      # number of videos to process , 1-49
55 effects = [0,12] # effect types 0-12 -> [0,0] implies no challenge
56 levels = [1,5]  # effect levels 1-5
57 syn = 1         # set to 1 to include Synthesized videos
58
59 # directory where videos are kept
60 loc = "/media/shahruk/Terra 2.0C/VIP Cup 2017 Data/Videos/"
61
62 # directory where frames will be saved
63 dst = "../Frames/"
64
```

fig.1 : Settings for Frame Extraction

1.2 Challenge Type Classifier

The subfolder labelled *Challenge Type Classifier* inside 1.Training folder contains the files needed to train the network. The feature files which were fed to the training script are found inside the Features folder, and the corresponding weights obtained are inside the Models folder. To reproduce the latter , one needs to do the following.

1.2.1 Feature Extraction

First the training features must be extracted using the script labelled *extractFeatures.py*. Like before, the directory where the extracted frames are saved must be set as the *loc* variable (fig. 2). The features will be saved within the Features directory by default.

For training we extracted two sets of features:

Feature Set B

This set contained samples from all of the video sequences. The No Challenge videos were included and Level 2 & 3 of all the challenge types. The values and explanation for the variables in line 20 to 24 to produce this set are as follows (also shown in fig.2) :

Including all trainset videos	:	$N_vid = [0,0]$
All challenges + No challenge	:	$effects = [0,12]$
Challenge levels 2 and 3	:	$levels = [2,3]$
Including Synthesized Videos	:	$syn = 1$
Not excluding Read Videos	:	$only_syn = 0$
Resizing image 125 x 125 px	:	$img_rows = 125, img_cols = 125$

```
7  # ----- Settings -----
8
9  # location of extracted Frames
10 loc = '../Frames/'
11
12 # save directory for features
13 feat_dir = './Features/'
14
15 # Video Selection
16 # Set N_vid = [0,0] to process only train videos, and N_vid = [0,1] to process only test videos
17 # if you want to list particular videos, put N_vid[0] = -1 and the follow with video # for rest of the elements
18 # else if you want a particular range, just enter the range in N_vid; e.g [1,9] will process video # 1 to video # 9
19
20 N_vid = [0,0]      # video sequence
21 effects = [0,12]   # challenge type 1-12 (1-11 for syn automatically adjusted )
22 levels = [2,3]     # challenge level 1-5
23 syn = 1            # make 1 to include Synthesized videos
24 only_syn = 0       # make 1 and syn = 1 to include only Synthesized videos
25
26 # name of feature set, and optional description
27 name = 'B'
28 desc = ''
29
30 # resize parameters
31 img_rows = 125
32 img_cols = 125
33
```

fig.2 : Settings for extraction of feature set B used to train RCNN

Running the script with this settings should produce a 2.3 GB *npz* file inside the Features directory labelled *B_train_125_125.npz* .

Feature Set C

This set also contained samples from all of the video sequences. But the No Challenge videos were excluded. All challenges were included but the challenge level were 4 & 5.

The values for the variables in line 20 to 24 to produce this set are as follows (also shown in fig.3) :

Including all trainset videos	:	$N_{vid} = [0,0]$
Only Challenges	:	$effects = [1,12]$
Challenge levels 4 and 5	:	$levels = [4,5]$
Including Synthesized Videos	:	$syn = 1$
Not excluding Read Videos	:	$only_syn = 0$
Resizing image 125 x 125 px	:	$img_rows = 125, img_cols = 125$

```

7  # ----- Settings -----
8
9  # location of extracted Frames
10 loc = '../Frames/'
11
12 # save directory for features
13 feat_dir = './Features/'
14
15 # Video Selection
16 # Set N_vid = [0,0] to process only train videos, and N_vid = [0,1] to process only test videos
17 # if you want to list particular videos, put N_vid[0] = -1 and the follow with video # for rest of the elements
18 # else if you want a particular range, just enter the range in N_vid; e.g [1,9] will process video # 1 to video # 9
19
20 N_vid = [0,0]      # video sequence
21 effects = [1,12]   # challenge type 1-12 (1-11 for syn automatically adjusted )
22 levels = [4,5]     # challenge level 1-5
23 syn = 1           # make 1 to include Synthesized videos
24 only_syn = 0       # make 1 and syn = 1 to include only Synthesized videos
25
26 # name of feature set, and optional description
27 name = 'C'
28 desc = ''
29
30 # resize parameters
31 img_rows = 125
32 img_cols = 125
33

```

fig.3 : Settings for extraction of feature set C used to train RCNN

Running the script with this settings should produce a 2.2 GB *npz* file inside the Features directory labelled *C_train_125_125.npz* .

For validation during training, we produced a further two feature sets using the test videos. The settings were the same as for feature set B and feature set C, except that the variable N_{Vid} was set to $[0,1]$ to use testset videos. (see fig.4 and fig.5)

```

7 # ----- Settings -----
8
9 # location of extracted Frames
10 loc = '../Frames/'
11
12 # save directory for features
13 feat_dir = './Features/'
14
15 # Video Selection
16 # Set N_vid = [0,0] to process only train videos, and N_vid = [0,1] to process only test videos
17 # if you want to list particular videos, put N_vid[0] = -1 and the follow with video # for rest of the elements
18 # else if you want a particular range, just enter the range in N_vid; e.g [1,9] will process video # 1 to video # 9
19
20 N_vid = [0,1]      # video sequence
21 effects = [0,12]   # challenge type 1-12 (1-11 for syn automatically adjusted )
22 levels = [2,3]     # challenge level 1-5
23 syn = 1           # make 1 to include Synthesized videos
24 only_syn = 0      # make 1 and syn = 1 to include only Synthesized videos
25
26 # name of feature set, and optional description
27 name = 'B'
28 desc = ''
29
30 # resize parameters
31 img_rows = 125
32 img_cols = 125

```

fig.4 : Settings for extraction of feature set B used to test RCNN (validation set)

```

7 # ----- Settings -----
8
9 # location of extracted Frames
10 loc = '../Frames/'
11
12 # save directory for features
13 feat_dir = './Features/'
14
15 # Video Selection
16 # Set N_vid = [0,0] to process only train videos, and N_vid = [0,1] to process only test videos
17 # if you want to list particular videos, put N_vid[0] = -1 and the follow with video # for rest of the elements
18 # else if you want a particular range, just enter the range in N_vid; e.g [1,9] will process video # 1 to video # 9
19
20 N_vid = [0,1]      # video sequence
21 effects = [1,12]   # challenge type 1-12 (1-11 for syn automatically adjusted )
22 levels = [4,5]     # challenge level 1-5
23 syn = 1           # make 1 to include Synthesized videos
24 only_syn = 0      # make 1 and syn = 1 to include only Synthesized videos
25
26 # name of feature set, and optional description
27 name = 'C'
28 desc = ''
29
30 # resize parameters
31 img_rows = 125
32 img_cols = 125
33

```

fig.5 : Settings for extraction of feature set C used to test RCNN (validation set)

Running the script with each of these settings should produce two more npz files labelled *B_test_125_125.npz* and *C_test_125_125.npz*

1.2.2 Training RCNN

Now you can commence training using the *trainRCNN.py* file inside the *Challenge Type Classifier* directory. The script file has already been set with parameters we used for training as shown in fig.6

The names of the features extracted in section 1.3.1 are placed in the *train_feat_name* and *test_feat_name* lists. The program will train the model using the features given in the *train_feat_name* list, and then validate against those given in the *test_feat_name*. The callbacks are set so that after every epoch, the weights are saved as well the weight with the best validation accuracy is saved as *best.hdf5* inside the subfolder of the model within the Models directory (see lines 124-127)

```
13  |#----- Settings -----|
14  |# folder where features from extractFeatures.py are saved|
15  |feat_dir = './Features/'|
16  |model_dir = './Models/'|
17
18  |# load/train old model|
19  |load_model = 0|
20  |continue_training = 0|
21  |evaluate_on_test_data = 0|
22
23  |# model description|
24  |model_name = 'BC'|
25  |load_model_name = ''|
26
27  |# name of features to load|
28  |train_feat_name = ['B_train_125_125','C_train_125_125']|    # used for training
29  |test_feat_name = ['B_test_125_125','C_test_125_125']|      # used for validation
30
31  |# Training Parameters|
32  |batch_size = 50|
33  |epochs = 50|
34  |patience = 15|        # early stopping parameter
35  |min_delta = 0.01|     # minimum change required to prevent stopping
36
37  |# RCNN Parameters|
38  |nbRCL = 6|            # 6 recurrent layers
39  |nbFilters= 128|       # 128 filters in conv layer
40  |filtersize = 3|       # 3x3 kernel size
41
42  |# image parameters|
43  |nbChannels = 3|
44  |shapel = 125|         # model expects 125x125 images
45  |shape2 = 125|
46  |nbClasses = 11|      # 11 challenge type classes (see extractFeatures.py)
47
```

fig.6 : Settings for training RCNN

The training will commence for 50 epochs. For our run, the validation accuracy reached its peak of 99.4% at the 20th epoch. The weights of different epochs are also saved within the *weights* subfolder inside the *BC* subfolder within the Models directory. The best weight (in terms of validation accuracy) in any case is saved as *best.hdf5* inside the *BC* subfolder as stated before.

This best weight must be copied to the *effectClass* subfolder inside the 2.Testing directory. That is the <....*/2.Testing/effectClass/*> directory.

With that, the training for the Challenge Type classifier is complete.

1.3 Bounding Box Detector

The folder labelled *Bounding Box Detector* inside the *1.Training* folder contains the code required to train this network. The training of this network is slightly more complicated than the other networks. Training time is also the longest for this; on our computer system, it took 5 days to complete training. The steps involved are detailed in the following sections.

1.3.1 Enlarging Ground Truth Boxes

The original implementation of the F-RCNN was to use to detect bigger bounding boxes compared to the given dataset. In order to improve detection performance we enlarged the given ground truth boxes to include not just the sign but the background around the sign, the context the object of interest was in, so to speak.

The code to this is given in the *enlargeROI.py* script. The directory variables already set and do not need to be modified. We enlarged the boxes by 2.5 times; the *zoom_factor* in line 6 of the code is hence set at 2.5.

The original label files provided to us by the organizers should also be inside the directory in the *labels* folder. Running the script should generate the .csv files found inside the *labels2_5* folder. The *2_5* implies that the ground truth boxes were scaled by 2.5 times along its length and width.

(Note : the code only modifies the coordinates of two opposite corners, [*llx*, *lly*, *urx*, *ury*] , since this is all that is required for training. The other coordinates are not modified and kept as is)

1.3.2 Training F-RCNN

The training is initiated by running the *trainFRCNN.py* script. But first, the directory of the extracted frames location must be set as the *loc* variable inside the script. The other parameters are already set to the values we used for our training as shown in fig.7a ,fig.7b and fig.7c (*jump to pg 12 if you want to skip overview of the parameters*)

```
7 # ----- Dataset Settings -----
8
9 # Directory containing extracted frames
10 loc = "../Frames/"
11
12 label_dir = './labels2_5/'      # Labels directory (boxes enlarged by 2.5 times in x and y)
13
14 # Video Selection
15 # Set N_vid = [0,0] to process only train videos
16 # if you want to list particular videos, put N_vid[0] = -1 and the follow with video # for rest of the elements
17 # else if you want a particular range, just enter the range in N_vid; e.g [1,9] will process video # 1 to video # 9
18
19 N_vid = [0,0]
20 effects = [0,12]    # [0,0] for no challenge
21 levels = [1,5]      # [1,5] for all levels
22 syn = 1             # 1 to include Synthesized videos
23 only_syn = 0        # 1 to extract ONLY Synthesized videos
24
25 # for training the FRCNN, we didn't take all the frames to save on training time
26 # these parameters dictate the skipping order
27 frame_skipR = 3      # 1 implies no skipping; else take every nth frame
28 frame_skipS = 4
29 skip_level_seq = [[1,2],[2,3],[1,3]] # these levels will be skipped for consecutive videos in this order
30
```

fig.7a : Settings for dataset generation to train F-RCNN

As before, the parameters in lines 19 to 23 deal with selecting the video to use for training. Parameters in lines 27 to 29 determine the way frames are sampled from the chosen videos. Since the training time using all the training frames would be quite long, we included this sampling feature to cut down on training time. From a given video, we took every 3rd frame that had at least one ROI (for real videos) and every 4th frame that had at least one ROI (for synthesized videos). We also skipped some two challenge levels from different video sequences. The skipping was done in such way so that a skipped level in one video sequence is included in another video sequence. A similar scheme was also used for the skipped frames by varying the starting frame from which every 3rd or 4th frame was taken.

```

32 # ----- Training Settings -----
33 class options:
34     def __init__(self):
35
36         # model_name (L implies left, and R implies right)
37         self.name = 'All_7class_[50,150]_660_R'
38         self.load_name = ''
39
40         # load previous weights to continue Training
41         self.load_weights = False
42
43         # load data file already generated previously
44         self.load_data = False
45
46         # number of epochs
47         self.num_epochs = 40
48         # the number of images iterated over per epoch
49         self.epoch_length = 12000
50
51         # resize the smallest side of input image to this value
52         self.im_size = 660
53         # the amount of overlap between the right and left half of the frame
54         self.overlap = 70
55
56         # threshold for asserting positive samples
57         self.rpn_max_overlap_threshold = 0.7
58
59         # window sizes and ratios used by RPN
60         self.anchor_box_scales = [50,150]
61         self.anchor_box_ratios = [[1, 1]]
62         self.rpn_stride = 16
63
64         if self.name[-1] == 'R':
65             self.cutImage = [0,660,814-self.overlap/2,1627]      #(y1,y2,x1,x2)
66         elif self.name[-1] == 'L':
67             self.cutImage = [0,660,0,814+self.overlap/2]          #(y1,y2,x1,x2)
68         else:
69             self.cutImage = [0,660,0,1628]                        #(y1,y2,x1,x2)
70

```

fig.7b : Settings for F-RCNN parameters to use in training

The class shown above contains some parameters that are used by the different parts of the F-RCNN system during training & testing. We split the chosen frames into left and right halves, and trained separate models on different PCs, again to cut down on training time. We also discarded the bottom half of the images as traffic signs appeared usually above the horizon line. The cropping happens in lines 65-69 depending on whether we are training the right side or left side model.

```

71 # batch size for classifier and ROI pooling
72 self.num_rois = 30
73
74 # location of config file to load settings from (default = 'config.pickle')
75 self.config_filename = './' + self.name + '/' + self.name + '_config.pickle'
76
77 # location of loading/saving model weights
78 self.input_weight_path = './' + self.load_name + '/' + self.load_name + '_model_frcnn.hdf5'
79 self.output_weight_path = './' + self.name + '/' + self.name + '_model_frcnn.hdf5'
80
81 # getting background samples
82 self.getBGsamples = False
83
84 # options for applying augmentations to train data
85 self.horizontal_flips = False
86 self.vertical_flips = False
87 self.rot_90 = False
88
89 # parameters for trainset list processing
90 self.train_path = 'trainset.txt'
91
92 # remapping of sign class in labels
93 self.num_frames = None
94 self.balanced_classes = True
95
96 # the 14 different type of signs were classed into these categories
97 self.class_map = { 1:'ABC', 2:'whiteMiddle',
98                   3:'whiteMiddle', 4:'diagonal',
99                   5:'diagonal', 6:'stop',
100                  7:'bike', 8:'triangle',
101                  9:'diagonal', 10:'diagonal',
102                  11:'whiteMiddle', 12:'stop',
103                  13:'triangle', 14:'ABC',
104                  15:'bg' }
105

```

fig.7c : Settings for F-RCNN parameters to use in training continued

The options for real time augmentation of the images during training was included, but ended up not being used to save on time. The different sign classes were grouped together differently to be used by the F-RCNN classifier to differentiate between proposed boxes with object of interest and one without. The new sign groups are self explanatory.

To start training, the *trainFRCNN.py* script must be executed from the terminal. The code will first generate a text file containing the annotations for the different frames, which will be then automatically be parsed and used to train by the function in the *train_frcnn.py* script.

The code should generate a folder called *All_7class_[50,150]_660_R* inside the *Bounding Box Detector* directory. The configuration files and other settings, as well as the weights of training are saved to this folder.

After every epoch, the code should print the losses and the average number of proposed boxes that overlapped with the ground truth per frame. The code saves the weights after every epoch as well as saves the weights with the best loss separately. But we took the weight that had the highest number of boxes overlapping with ground truth on top of having a minimized loss. Our training outputs to terminal are given in the text file labelled *Training Run 1.txt* in the *Bounding Box Detector* directory.

Even though the loss continued to decrease ever so slightly every epoch, so did the average rpn overlap. So we opted to take the weights generated after the 25th epoch (refer to the *Training Run 1.txt* and fig.8) as it was a compromise between the average rpn overlap and loss.

```

Elapsed time: 8400.40735579

Loss did not improve
-----
Epoch 25/40
12000/12000 [=====] - 8490s - rpn_cls: 0.0790 - rpn_regr: 0.0607 - detector_cls: 0.1705 - rpn_overlap: 7.0971
-----
Mean number of bounding boxes from RPN overlapping ground truth boxes: 7.38671875
Classifier accuracy for bounding boxes from RPN: 0.9396124999
Loss RPN classifier: 0.0805845538504
Loss RPN regression: 0.0605819203838
Loss Detector classifier: 0.170907719047
Loss Detector regression: 0.11304403908
Elapsed time: 8490.89565516

Total loss decreased from 0.42793696293 to 0.425118232361
-----
Epoch 26/40
12000/12000 [=====] - 8311s - rpn_cls: 0.0772 - rpn_regr: 0.0601 - detector_cls: 0.1764 - rpn_overlap: 7.1098
-----
Mean number of bounding boxes from RPN overlapping ground truth boxes: 7.32418496341
Classifier accuracy for bounding boxes from RPN: 0.93941458317
Loss RPN classifier: 0.0768598426614

```

fig.8 : Expected output to terminal during training

Delete all other weights except the weight of the 25th epoch. It should be labelled *All_7class_[50,150]_660_R_model_frcnn_025_7.39*.

(note : the value at the end may be different from 7.39 as we did not set the numpy random seed to a particular value before training. Hence differences in shuffling and sampling may vary the rpn overlap slightly)

Rename the weight to : *All_7class_[50,150]_660_R_model_frcnn* and then copy the entire folder called *All_7class_[50,150]_660_R* to the *frcnn* folder inside the *2.Testing* directory; that is the *<.../2.Testing/frcnn/>* directory.

At the time of writing there was a known bug in the training code, where the error below would appear for a couple of particular frames used in training. However this has been handled so that training continues with the rest of the frames.

```
Epoch 17/40
9849/12000 [=====>.....] - ETA: 1448s - rpn_cls: 0.0839 - rpn_regr: 0.0636 -
detector_cls: 0.1831 - rpn_overlap: 6.9642

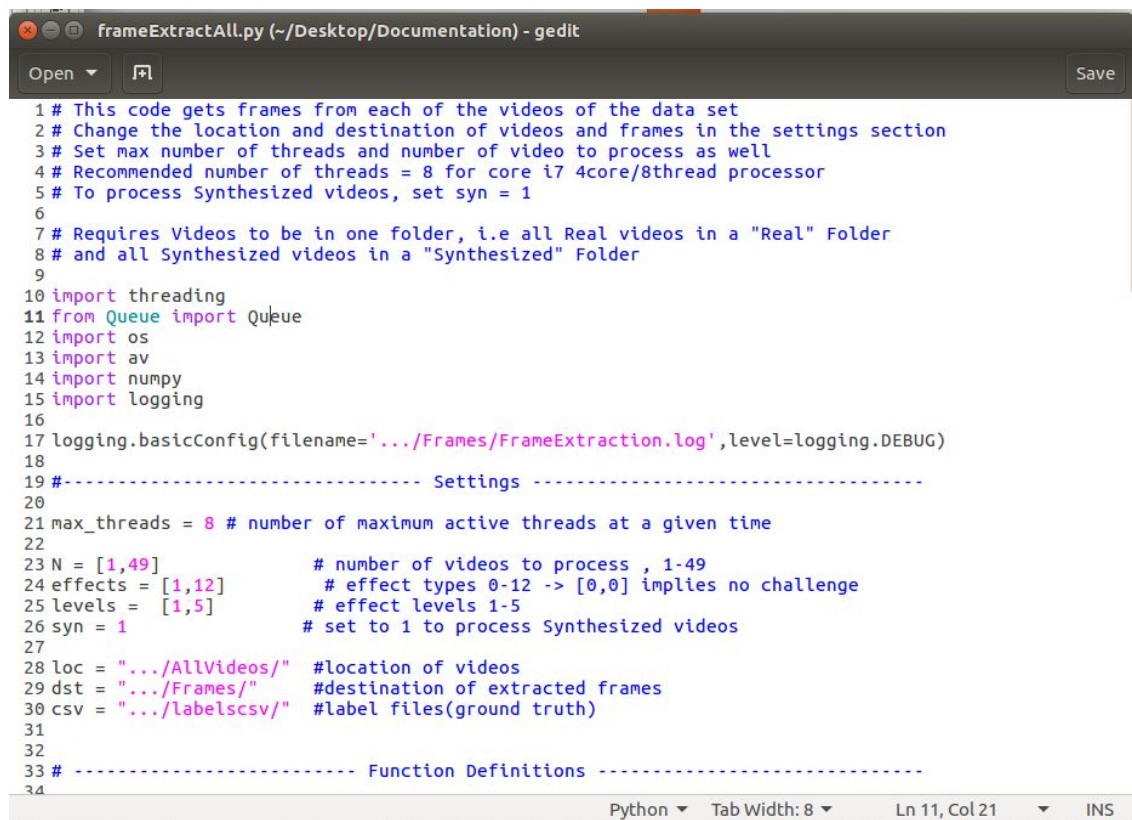
/media/smlab/E/FRCNN/CODES/Keras_Shahruk/FRCNN v3/keras_frcnn/roi_helpers.py:139: RuntimeWarning:
overflow encountered in exp
  h1 = np.exp(th) * h
/media/smlab/E/FRCNN/CODES/Keras_Shahruk/FRCNN v3/keras_frcnn/roi_helpers.py:266: RuntimeWarning:
invalid value encountered in add
  A[3, :, :, curr_layer] += A[1, :, :, curr_layer]
/media/smlab/E/FRCNN/CODES/Keras_Shahruk/FRCNN v3/keras_frcnn/roi_helpers.py:283: RuntimeWarning:
invalid value encountered in greater_equal
  idxs = np.where((x1 - x2 >= 0) | (y1 - y2 >= 0))
```

fig.9 : known bug for some particular frames.

1.4 Sign Type Classifier

1.4.1: Extracting The frames from videos:

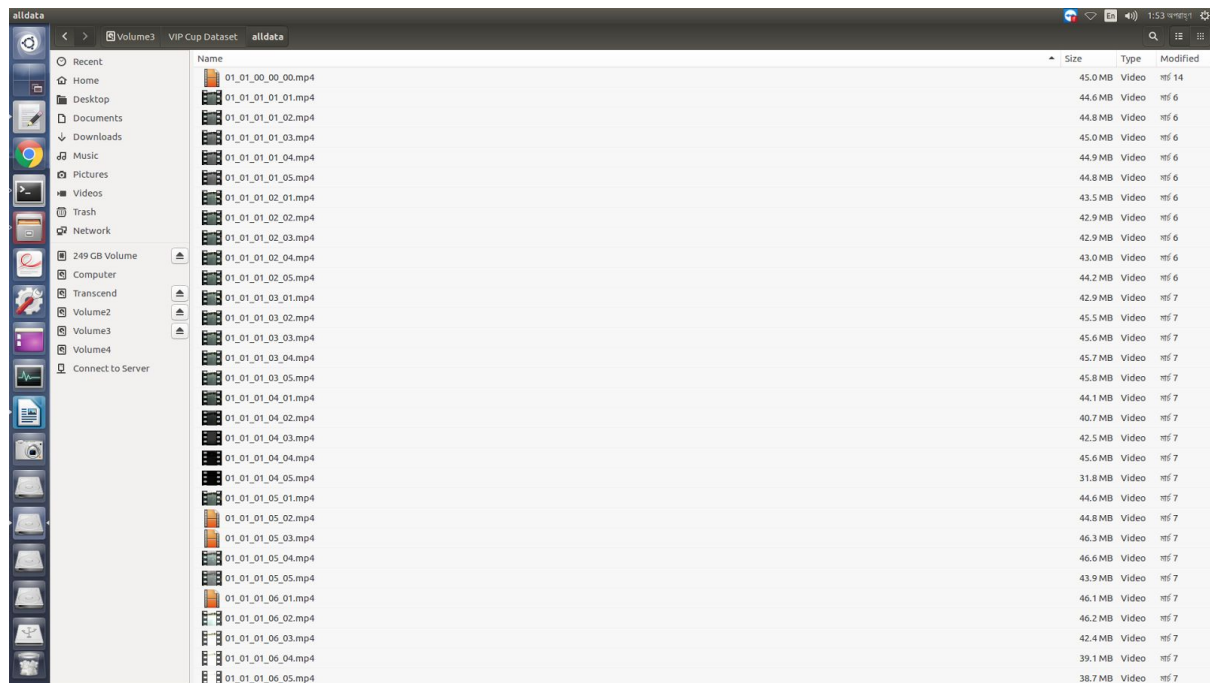
In order to extract frames from videos we use the program 'FrameExtractAll.py'



```
frameExtractAll.py (~/Desktop/Documentation) - gedit
Open Save
1 # This code gets frames from each of the videos of the data set
2 # Change the location and destination of videos and frames in the settings section
3 # Set max number of threads and number of video to process as well
4 # Recommended number of threads = 8 for core i7 4core/8thread processor
5 # To process Synthesized videos, set syn = 1
6
7 # Requires Videos to be in one folder, i.e all Real videos in a "Real" Folder
8 # and all Synthesized videos in a "Synthesized" Folder
9
10 import threading
11 from Queue import Queue
12 import os
13 import av
14 import numpy
15 import logging
16
17 logging.basicConfig(filename='../Frames/FrameExtraction.log', level=logging.DEBUG)
18
19 #----- Settings -----
20
21 max_threads = 8 # number of maximum active threads at a given time
22
23 N = [1,49] # number of videos to process , 1-49
24 effects = [1,12] # effect types 0-12 -> [0,0] implies no challenge
25 levels = [1,5] # effect levels 1-5
26 syn = 1 # set to 1 to process Synthesized videos
27
28 loc = "../AllVideos/" #location of videos
29 dst = "../Frames/" #destination of extracted frames
30 csv = "../labelscsv/" #label files(ground truth)
31
32
33 # ----- Function Definitions -----
34
```

It is required to run this code setting the parameters: **loc** (location of videos), **dst** (destination where to extract frames), **csv**(the ground truth files, converted to csv from text, provided with the code in the folder 'labelscsv') and Filename in the logging.basicconfig for logging info.

** It is required that all the video files are contained in one single directory for the frame extract code to run. i.e Like in this directory:



It is required to do two runs to extract all the frames. First for the No challenge cases with these settings:

```
N = [1,49]
effects = [0,0]
levels = [0,0]
syn = 1
```

Then for All challenge cases with these settings:

```
N = [1,49]
effects = [1,12]
levels = [1,5]
syn = 1
```

Running the program twice with these settings will produce frames from the videos in the destination folder.

1.4.2: Extracting ROIs from ground truth

We use two settings for extracting ROIs from ground truth. The program *'roiExtract_v1.py'* is used for this purpose. The first setting is with the ground truth and the second is from ground truth plus a zoom factor of 2.5x

Setting up directories:

```
# Directories
csv_dir = '.../labelscsv/'          # location of ROI labels
sav_dir = '.../ROIs/NoZoom/'        # save location for extracted ROIs
frame_dir = '.../Frames/'           # location of extracted frames
```

a: No Zoom:

ROI extraction is performed with **zoom_factor** set to 0 in these two sets to extract all the ROIs.

```
N = [1,49]
effects = [0,0]
levels = [0,0]
syn = 1
```

```
N = [1,49]
effects = [1,12]
levels = [1,5]
syn = 1
```

b. 2.5x Zoom:

ROI extraction is performed with **zoom_factor** set to 2.5x like the previous step to extract ROIs with a factor of **2.5x** enlargement.

1.4.3: Manual Training Data:

Since the FRCNN outputs a great amount of unwanted frames we use a set of manually trained csv files. These need ROI and feature Extraction as well which are performed with *'roiExtract_v2.py'*

a. For Real Videos:

**it is suggested to follow the numbers of the array N since not all Video sequences were not chosen for manual training.

No challenge settings:

```
csv_dir = '.../csvTrained/Real/'
sav_dir = '.../ManualTrain/Real/Cropped/'
frame_dir = '.../Frames/Real/'

V = [1]
N = [1,2,3,9,10,11,12,13,14,15,16,17,20,22,23,25,27,28,29,30,32,33]

E = [0] #range(1,13)
L = [0] #range(1,6)
```

Challenge Video settings:

```
csv_dir = '.../csvTrained/Real/'
sav_dir = '.../ManualTrain/Real/Cropped/'
frame_dir = '.../Frames/Real/'

V = [1]
N = [1,2,3,9,10,11,12,13,14,15,16,17,20,22,23,25,27,28,29,30,32,33]

E = [0] #range(1,13)
L = [0] #range(1,6)
```

b. For Synthesized Videos:

No challenge settings:

```
csv_dir = '.../csvTrained/Synthesized/'
sav_dir = '.../ManualTrain/Synthesized/Cropped/'
frame_dir = '.../Frames/Synthesized/'

V = [2]
N = [1,3,5,7,8,11,14,15,19,21,25,29,35,38,42,45,47,49]

E = [0] #range(1,13)
L = [0] #range(1,6)
```

Challenge Video Settings:

```

csv_dir = '.../csvTrained/Synthesized/'
sav_dir = '.../ManualTrain/Synthesized/Cropped/'
frame_dir = '.../Frames/Synthesized/'

V = [2]
N = [1,3,5,7,8,11,14,15,19,21,25,29,35,38,42,45,47,49]

E = range(1,13)
L = range(1,6)

```

1.4.4: Feature Extraction

The following steps describe the process of training the classifier. We use the program *'featureExtract.py'* for this purpose.

For example if we want to extract features for Challenge 1 i.e. Discoloration

a. Feature Extraction (Training Set):

The program *featureExtract.py* is used to extract features and save them into npz files. For challenge 1, we need to extract a total of three Training Feature sets with the following settings:

Settings :

```

N_vid = [0,0]
effects = [1,1]
levels =[1,5]
syn = 1
only_syn = 0

```

The directory of extracted features and the log file should be defined in the following lines: (line 51,130)

```

train_feat = '../Features/' + name + '_' + typ + '_' + str(img_rows) + '_' + str(img_cols)

# log file
logging.basicConfig(filename='../Features/' + name + '_' + typ + '_' + str(img_rows) + '_' + str(
img_cols) + '.log',level=logging.INFO,filemode='w')

```

Three sets of npz files are created for a challenge.

i. NoZoom (ground truth ROIs):

Directories: Location where the NoZoom ROIs are extracted:

```

# Directories
locR = '.../ROIs/NoZoom/Real/Cropped/'
locS = '.../ROIs/NoZoom/Synthesized/Cropped/'

```

Also provide name of the npz file set and optionally description might be set here:

```
# Data type = train/test
name = 'NoZoomCh1'
desc = 'Test Dataset comprised of No Zoom Challenge 1'
```

ii. 2.5x Zoom Out (ground truth with 2.5 times enlarged ROIs)

```
# Directories
locR = '.../ROIs/ZoomedOut_2.5/Real/Cropped/'
locS = '.../ROIs/ZoomedOut_2.5/Synthesized/Cropped/'
|
```

Name also required for npz file.

iii. Manual Training Input

```
# Directories
locR = '.../ROIs/ManualTrain/Real/Cropped/'
locS = '.../ROIs/ManualTrain/Synthesized/Cropped/'
|
```

Name also required for npz file.

b. Feature Extraction (Validation Set)

We use 2.5 Times enlarged ROIs directories and setting **N_vid** to **[0,1]** to generate an npz file for validation.

```
N_vid = [0,1]
effects = [1,1]
levels = [1,5]
syn = 1
only_syn = 0

# Directories
locR = '.../ROIs/ZoomedOut_2.5/Real/Cropped/'
locS = '.../ROIs/ZoomedOut_2.5/Synthesized/Cropped/'
|
```

name required for npz file.

**** for no challenge there is an exception where settings should be as follows:**

```
N_vid = [0,0]
effects = [0,0]
levels = [0,0]
syn = 1
only_syn = 0
```

Repeat For all challenges:

This process is repeated for All challenges which provides us with a set of npz files to work our classifier upon.

For example, for challenge type 'i' only change the corresponding setting **effects = [i,i]** and the corresponding name as accordingly.

1.4.5: Training Classifier

a. Combining multiple npz files:

The necessary npz file names (without the '_train_64_64.npz') are entered in the array named feats and the name which the npz files will be stored is also provided. Feature directories are also set up. Table 1.4.5 is followed to combine npz files.

```
#directories
feat_dir = '../Features/'           #directory for source features
save_dir = '../Features_Split/'     #directory for storing shuffled and split features

#feature names which need to be shuffled and split
feats= ['ZoomCh1','NoZoomCh1','ManCh1',.....'', '', '']
save_name = 'Model1' #feature save name
```

b. Training The classifier:

The classifier is located in the Folder '**Classifier**'. To configure the classifier it is necessary to configure the parameters in the file: '**tscParams.py**'

The parameters **name_train** will have the names of the npz files prepared by **combineNPZ.py** and **name_test** will only require the validation sets.

The **Batch_size** will vary with the GPU used in training. It is recommended to use 32/64/128 batch size which suits the GPU. .

Model_name is necessary for training. Model wise training sets are discussed in the following table.

After setting the options run the program '**Classifier.py**' in order to train the classifier. Weights are stored in the folder model_dir. The best weight will be found in the folder as **best.hdf5** for using in the final detection.

Model:	Train Set (Goes into Combine npz) No Zoom + 2.5x Zoom + Manual Train Set For each challenge	Validation set 2.5x Zoom Test Set
No Challenge + Decolour + Codec Error	No Challenge + Challenge 1+ Challenge 3	No Challenge + Challenge 1+ Challenge 3
Blurs	Challenge 2 + Challenge 7	Challenge 2 + Challenge 7
Rain + Noise	Challenge 8 + Challenge 9	Challenge 8 + Challenge 9
Darkening	Challenge 4	Challenge 4
Dirty Lens	Challenge 5	Challenge 5
Exposure	Challenge 6	Challenge 6
Shadow	Challenge 10	Challenge 10
Snow	Challenge 11	Challenge 11
Haze	Challenge 12	Challenge 12
Default	** See below	No Challenge

Table 1.4.5

i. e. for No Challenge + Decolour + Codec error we take the following as **name_train** entries:

feats = ['NoZoomNC', 'ZoomNC', 'ManNC', 'NoZoomCh1', 'ZoomCh1', 'ManCh1', 'NoZoomCh3', 'ZoomCh3', 'ManCh3']

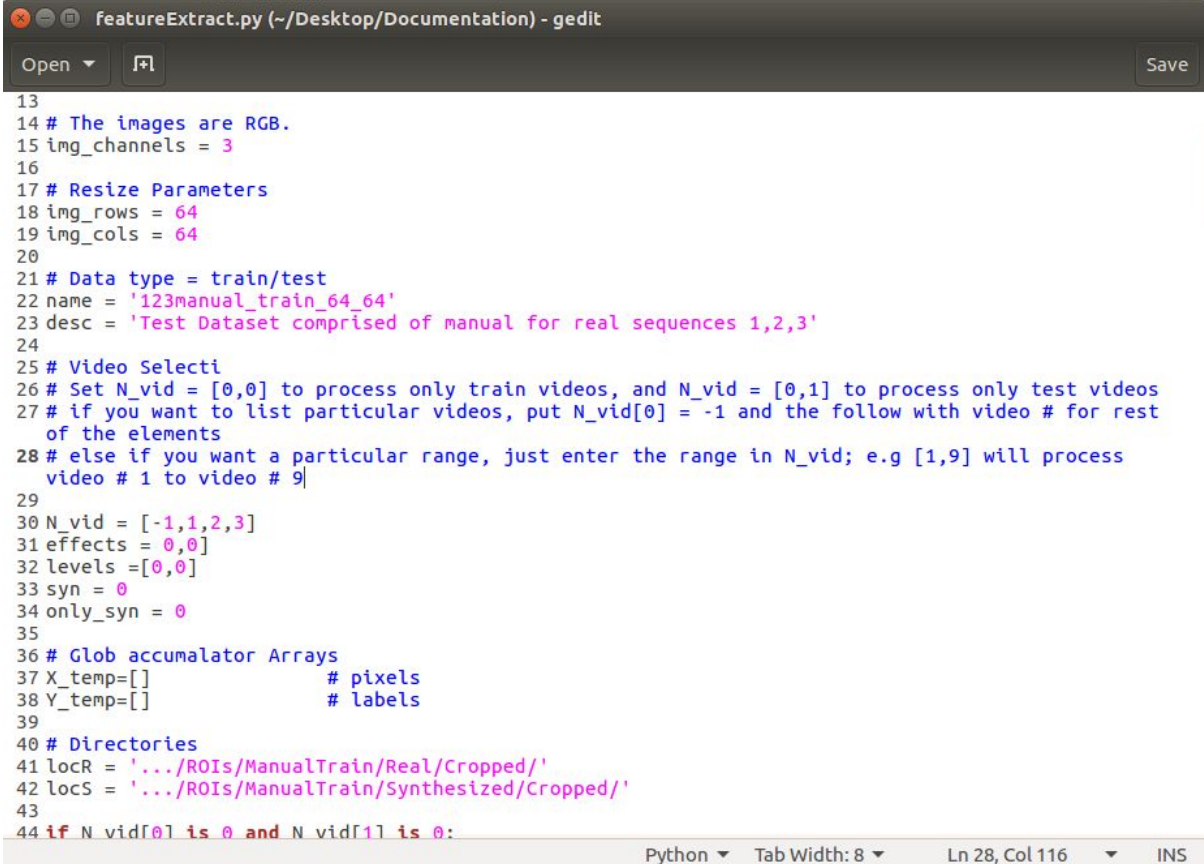
Then combineNPZ.py provides output as files:

['model1_00', 'model1_01', 'model1_02', 'model1_03', 'model1_04', 'model1_05', 'model1_06', 'model1_07', 'model1_08']

Which we then enter to the **name_train** variable into the **tscOptions.py** file.

The validation set does not require a shuffle by *combineNPZ.py*.

The **Default** model was trained using No Challenge Data at 2.5 zoom and Manual Training Data from video sequences 1,2 and 3 only. In order to produce the manual part of the dataset, **featureExtract.py** needs to be run with the following parameters:



```
13
14 # The images are RGB.
15 img_channels = 3
16
17 # Resize Parameters
18 img_rows = 64
19 img_cols = 64
20
21 # Data type = train/test
22 name = '123manual_train_64_64'
23 desc = 'Test Dataset comprised of manual for real sequences 1,2,3'
24
25 # Video Selecti
26 # Set N_vid = [0,0] to process only train videos, and N_vid = [0,1] to process only test videos
27 # if you want to list particular videos, put N_vid[0] = -1 and the follow with video # for rest
  of the elements
28 # else if you want a particular range, just enter the range in N_vid; e.g [1,9] will process
  video # 1 to video # 9|
29
30 N_vid = [-1,1,2,3]
31 effects = [0,0]
32 levels =[0,0]
33 syn = 0
34 only_syn = 0
35
36 # Glob accumulator Arrays
37 X_temp=[]          # pixels
38 Y_temp=[]          # labels
39
40 # Directories
41 locR = '.../ROIs/ManualTrain/Real/Cropped/'
42 locS = '.../ROIs/ManualTrain/Synthesized/Cropped/'
43
44 if N_vid[0] is 0 and N_vid[1] is 0:
```

Afterwards, **123manual_train_64_64.npz** and **ZoomNC_train_64_64.npz** are used in combineNPZ.py to train the classifier.

Thus 10 sets of classifiers are produces according to the table.

2. Testing the System

2.1 System Summary

The system consists of two blocks :

- a. *Frame Generation & Challenge Type Detection*
- b. *ROI detection, tracking and classification*

Part (a) is handled by the script *frameExtractLib.py*, from which functions are called by the *populateQ* function from the *main.py* script. There are two frame generators, #1 uses video as input from which frames are obtained, #2 used frames directly as input. We included the latter in the event that the function we used to read videos *cv2.VideoCapture()* does not work on the PC where the code will be run.

Inside these frame generators, first the obtained frame is passed to the challenge type classifier and then to the respective *Queues* for the rest of the system. The classifier's output is not actioned upon until the first 7 frames are processed. Once this is done, the type with the most number of detections is chosen as the challenge type and the following actions are taken :

i. Update the tracker mode

(Uses Lucas-Kanade by default, switches to Kalman for challenge type 5,6 and 10)

ii. Update the sign classifier weights

(Uses separate set of weights for each challenge type)

This update process is then continued for all successive frames.

The *frameExtractLib.py* script also contains some pre-processing functions which are called based on the challenge type. But they don't modify the frames in any way and returns whatever image was passed to the function. Due to time constraints, we couldn't process all the frames and train the system with them and so opted to not use them.

The frames are put into 5 queues by the *populateQ* function :

- I. *imgQ* : Queue used by Sign Classifier to Crop ROI from
- II. *frCNNQL* : Queue for Left side of frame to be used by frCNN
- III. *frCNNQR* : Queue for Right side of frame to be used by frCNN
- IV. *LKTrackQ* : Queue for tracker module
- V. *imgVisualizeQ*: Queue for visualizing detected boxes (disabled by default)

At most there will be 300 frames in the Queue and the queues will not populate until a slot is freed by consumption. The system uses only the current frame for its detection process, the queued frames are just kept in buffer.

The FRCNN takes a frame and proposes ROI boxes for it, which are then passed to the tracker module.

The tracker module uses its own prediction based on either optical flow (Lucas Kanade) or Kalman Filtering, and compares it with the FRCNN output. It determines which co-ordinates are the best to take based on the fact that a box will move by only so much between frames.

Besides keeping track of all the boxes, it uses feedback from the sign classifier to determine whether the boxes are valid. If the ROI box does not contain a traffic sign listed in the given labels, those boxes are discarded, and boxes in that same region are not accepted from FRCNN for 2 consecutive frames.

The tracker system switches between Lucas-Kanade and Kalman dynamically based on the information from the challenge type classifier. For challenges where there are static objects on the frames , such as Dirty Lens (challenge 5) or Shadows(challenge 10), optical flow is not a suitable method for tracking and so we switch to Kalman Filter.

The tracker module then passes the boxes in its possession to the Sign Type Classifier, which classifies the signs into one of 22 classes (14 being traffic signs, and 8 extra classes of non sign objects we manually added). If the output class is among the traffic signs, it's fed back to the tracker module which revitalizes the corresponding ROI box. If it's not the box is " killed off " and boxes in that regions from FRCNN are not accepted for consecutive two frames.

The output from the sign type classifier is then written to the detection file.

2.2 Result Generation

To generate the detection files, all the weights must be in place at their respective directories inside the *2.Testing* folder. Once this is done, the *main.py* script can be run from the the terminal. (*More details on the flags inside the README FILES directory*)

To generate the detection files, you need to run the main.py file and pass the directory of the videos with the flag -p

```
$ python main.py -p '/media/drive/vip/videos/'
```

We have also kept the provision to split the testing process into 6 parts, where each part consists of 5 testset video sequences and all their challenge types and levels. This way the code can be run on 6 different pcs/gpus simultaneously to generate the detection files faster. To do this, you need to pass the --part flag followed by an integer between 1 to 6 to denote the part to process.

```
$ python main.py -p '/media/drive/vip/videos/' --part 1
```

```
$ python main.py -p '/media/drive/vip/videos/' --part 6
```

If at any time the result generation is stopped due to power failure or some other error, the generation can be resumed from the point where it stopped by passing the --continue flag

```
$ python main.py -p '/media/drive/vip/videos/' --part 1 --continue
```

```
$ python main.py -p '/media/drive/vip/videos/' --continue
```

3.Dependencies

We developed our code to run on Ubuntu 16.04 using python 2.7.13

The following modules are required for the running our system

1. OpenCV version 3.2
2. Anaconda version 1.6.3
3. Keras version (2.x.x) with TensorFlow backend
4. Scipy
5. Scikit-image
6. Numpy
7. Progressbar 2 (run at the terminal: *conda install progressbar2*)
8. PyAV (follow directions here: <https://mikeboers.github.io/PyAV/installation.html>)
9. Built-in python libraries such as threading, os, glob, logging etc.