Folders and Files:

1 data

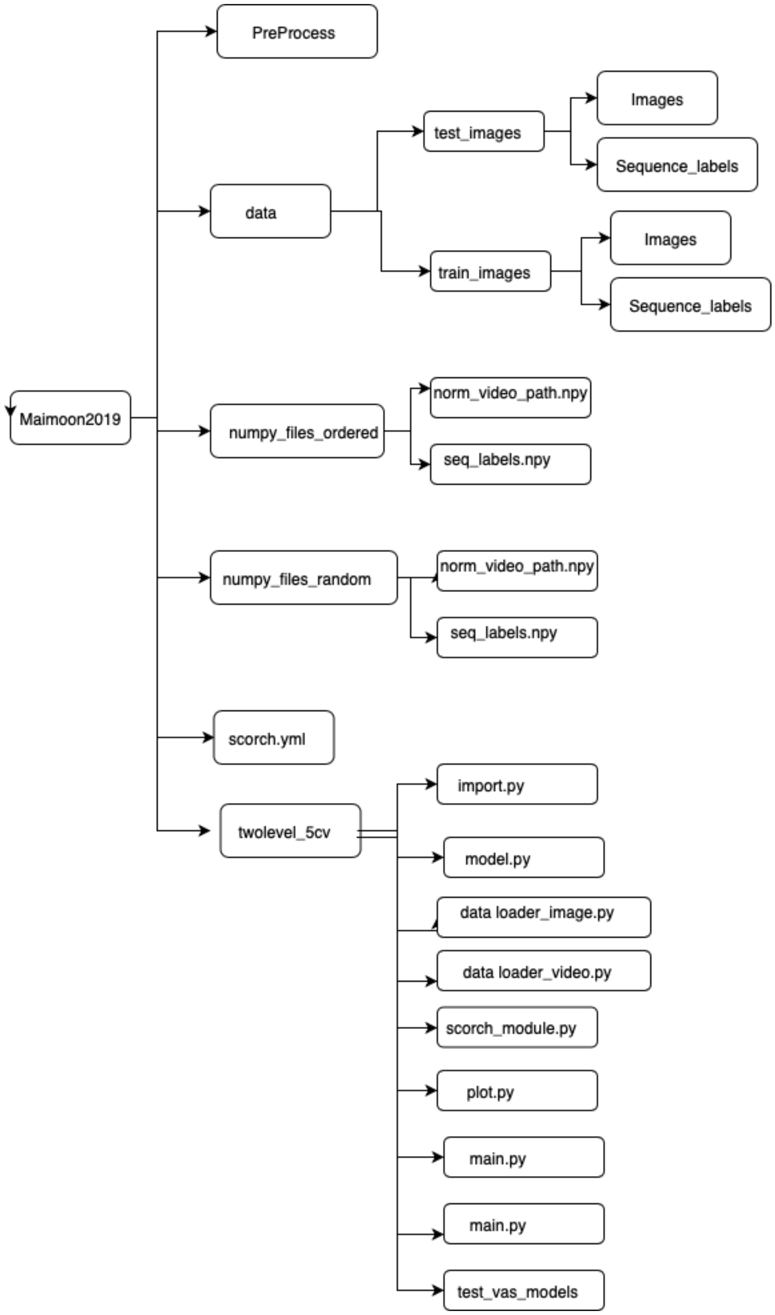
2 numpy_files_ordered

2 numpy_files_random

3 two_level_5cv

4 skorch.yml

Directory Tree:



Requirements:
0) Needs Python 3
1) Make sure skorch library is installed in python
skorch: *https://skorch.readthedocs.io/en/stable/*
2) Make sure mathplotlib is installed
python3 -m pip install matplotlib/sudo apt-get python3-matplotlib

Note an easier way would be to set up the conda environment: skorch.yml
Given if conda is installed:

1) conda env create -f skorch.yml
2) source activate skorch

1 Data:

The data folder contains tracked, normalised facial images
Test images = Few dummy set for testing
Train images = All 25 participants

Each has following directories
1)Images = Images /<ParticipantID>/<SequenceNumber>/<Frames>
2)Sequence_Labels = Sequence_Labels/AFF, OPR, VAS, SEN/>/<SequenceTxt>

Ensure this is how your data is stored.

2 numpy_files_test/ordered/random:

The folders test_numpy_files, numpy_files_ordered and numpu_files_random contain 2 NumPy files:

1) norm_video_paths.npy
2) seq_labels.npy

How to generate NumPy files for testing or training?
With the help of the file **numpy_files_generator.py**

Given the folloing directory of the data: ./data/test_images/
The numpy files would be saved in the folder: './test_numpy_files'

$python numpy_files_generator.py

3 two_level_5cv:

The File Directory:

1) imports.py
2) model.py
3) dataloader_image.py/dataloader_video.py
4) skcorch_module.py
5) plot.py
6) main.py
7) test.py
7)models_test

Maimoon

imports.py = imports all the required libraries and initializes all the CNN and RNN parameters in the parameters dictionary.

Functions

All the parameters and hyperparameters that can be changed have a #Change before them

The following are the parameters that can be changed:

```
#Change the dataloader to load mp4 videos or frames
from dataloader_image import PainDataset
or
from dataloader_video import PainDataset

#Change the number of cv to be done
•   folds = 5

#Change The path for models to be saved
•   MODEL_PATH_DIR = './models_VAS'

#Change to type of experiment for train and test result graphs
•   result_path = "./"+timestamp+"_FoldResult_VAS_TwoLevel_Random"
```

```
if running test.py:
#Change the path for models to be tested
•   TEST_MODEL_PATH_DIR = './models_vas_test'
```

```
#Change the path for the testing Images
•   global_dict['TEST_DATA_PATH'] = "../data/test_images/"

#Change the Numpy files for testing
•   files= {
  "seq_labels" : '../test_numpy_files/seq_labels.npy',
  "video_paths" : '../test_numpy_files/norm_video_paths.npy',}
```

```
#Change the Numpy files for training
•   files= {
Use ordered or random as required
  #"seq_labels" : '../numpy_files_ordered/seq_labels.npy',
  #"video_paths" : '../numpy_files_ordered/norm_video_paths.npy',
  "seq_labels" : '../numpy_files_random/seq_labels.npy',
  "video_paths" : '../numpy_files_random/norm_video_paths.npy',
 }
```

Experiments

Random Distribution:

In order to run data loaded randomly, make sure that the following numpy files are used:

1) np.load('../../numpy_files_random/seq_labels.npy')
2) np.load('../../numpy_files_random/norm_video_paths.npy')

Stratified Distribution:

In order to run data loaded randomly, make sure that the following numpy files are used:

1) np.load('../../numpy_files_ordered/seq_labels.npy')
2) np.load('../../numpy_files_ordered/norm_video_paths.npy')

```
#Change the path for training images
•   DATA_PATH = "../data/train_images/"
```

```
#Change different train parameters
 train_params = {
    #"cuda": False,
    #"device": 'cpu',
    "device": 'cuda',
    "cuda": True,
    "seed": 1,
    "batch_size": 1,
    "test_batch_size": 1,
    "epochs": 15,
    "lr": [0.0001,0.0001,0.0001,0.0001],
    "weight_decay": 0,
    "console_logs": 200,
    "training_loss_func": 'mse', (or ‘custom’)
```

Loss Experiment:

The training loss needs to be changed accordingly in the function to:
1)MSE
3)Custom Loss

```
    "regularization": True, (change to False if regularization not required)
    "custom_loss_alpha": 0.7,
}
```

Note: Custom Loss is not defined for single labels since there is one label and no concept of variance.

```
 #Change the weight vector of the pain labels
  w = torch.FloatTensor([<weight tensor>])

  #Change the pain label experiment details
  labels_dict = {
    "w" : w,
    "idx":<ordered index of laebls>
    "number":<number of labels>,
    "label”:<ordered label titles>}
```

For example running OPR and VAS:

```
  w = torch.FloatTensor([5,10]

  labels_dict = {
    "w" : w,
    "idx":[1:3]
    "number":2
    "label":[‘OPR’,'VAS']}
```

All fields have to be ordered as following: [AFF,OPR,VAS,SEN]

```
  #Change different network parameters
  network_params = {
     "pre_trained": True, (downloads the pretrained weights of AlexNet)
     "input_size": 4096,
     "hidden_size": 1024,
     "num_layers": [2,2,2,2], (number of layers for inner 4 cv)
     "nonlinearity": 'tanh',
     "bias": True,
     "batch_first": True,
     "dropout": [0.1,0.1,0.1,0.1], (dropout for inner 4 cv)
     "bidirectional": False,
     "lambda": [0.000001,0.000001,0.000001,0.000001], (regularizer rater for inner 4cv)
  }
```

Diyala

model.py = contains the ‘model’ class for the architecture used.

Contains two classes:

1) class AlexNet(nn.Module)
2) class CnnRnn(nn.Module)

The CNN used is AlexNet. When run for the first time it downloads the pre trained AlexNet from the model_url = 'https://download.pytorch.org/models/alexnet-owt-4df8aa71.pth'

dataloader.py = loads the dataset and data loader, pytorch requirement

Takes in the numpy file and generates a dataset with the following keys:

1) sample['clip'],
2) sample['label']

dataloader_video.py: takes data as video clips (.mp4) files

dataloader_image.py: takes data as frames

Maimoon

skorch_module.py = overwrites the training loss function in skorch library used.

class NeuralNetRegressorNet(NeuralNetRegressor):

def get_loss(self, y_pred, y_true, *args, **kwargs): the training loss function for back propagation

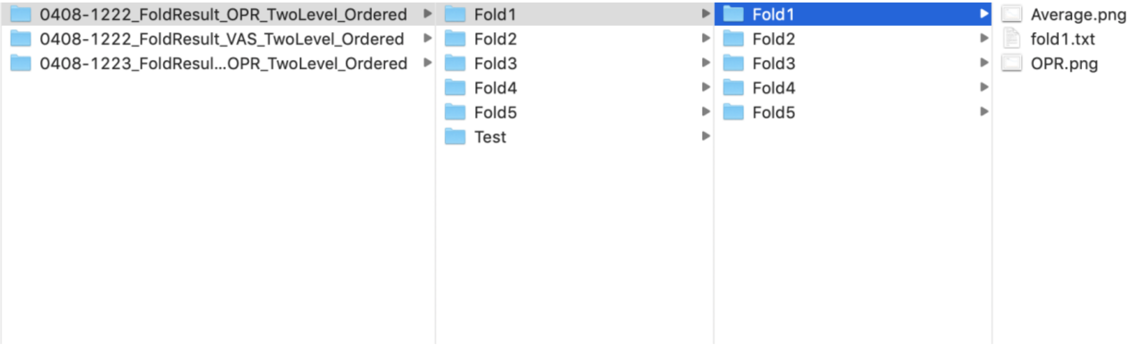
class EpochTimer(Callback): function called at the end of every epoch, used for storing the loss per label for train and validation

Maimoon

plot.py = for plotting the training/testing at the end of the 5 folds.(put ref to paper)

Contains the following function:

1) **def generate_results_dir(result_path,folds,MODEL_PATH_DIR) :** generates all the nested directories to store the training result of the 2 level 5 fold cv.
Output example= 1008-1405_FoldResult_VAS_TwoLevel_Random
The first part of the name is the timestamp associated with the folder directory that was generated.



0408-1222_FoldResult_OPR_TwoLevel_Ordered
0408-1222_FoldResult_VAS_TwoLevel_Ordered
0408-1223_FoldResul...OPR_TwoLevel_Ordered

Fold1
Fold2
Fold3
Fold4
Fold5
Test

Fold1
Fold2
Fold3
Fold4
Fold5

Average.png
fold1.txt
OPR.png

The second columns shows directory of the outer fold, and the inner directory shows directory of the inner fold. There are 5 folds in the outer level, and 4 folds in each of the outer 5 folds Each of the inner fold contains the average train loss vs epoch graph, scaled train loss vs epoch graph, and a txt file storing all the required information.

2) **def plot_func(history,num_epochs,path):** Draws the training loss graphs at the end of every fold inner.
The graph drawn is the training loss vs epochs.

3) **def write_history(file,net,num_epochs):** writes the information to a txt file at the end of training for each inner fold.
The information stored is as follows:
1) Train loss at every epoch

4) **def plot_test(folds,test_fold,test_loss_tensor,path,min_param_Array,pred_true_array,typ)):** Draws the testing error graphs for all the outer folds at the end.

The information stored in txt files as follows:
1) Average Test error for each outer fold.
2) Test error scaled in original range for each outer fold.
3) Parameters index which gave the minimum validation loss(note the parameter corresponding to the index can be found in the imports.py file)
4) Average loss across all folds
5) Actual, predicted labels

Maimoon

main.py

Contains the following function

1)def test_loss_fn(net, ds, y=None): Accuracy function for testing, called for one outer fold

2) def initialise_model(val_dataset,idx): For initialisng a new model for each fold

3) main function:

Performs 2 level cross validation with nested loops

How to run main for training?

Run main.py as following:

```
 $CUDA_VISIBLE_DEVICES="SPECIFY GPU NUMBER" python main.py
```

If need to turn gpu off, simply set cuda to false in imports.py

Maimoon

test.py

Look at imports.py to change relevant variable names

Run test.py as following

```
$CUDA_VISIBLE_DEVICES="SPECIFY GPU NUMBER" python test.py
```

Note numpy files need to be generated for different dataset, or different subsets for testing