Smile Recognition

Release 1.0

Benedykt Pawlus

Sep 01, 2023

CONTENTS:

CHAPTER

ONE

SMILERECOGNITION

1.1 concat_test module

concat_test.**main**()

Performs calculation of best models to concatenate with starting features

1.2 constants module

1.3 dataset module

class dataset.**UVANEMODataGenerator**(**args: Any*, ***kwargs: Any*)

Bases: Dataset

Dataset of UVA-Nemo and AUDA features

__getitem__(*idx*)

__init__(*label_path*, *videos_path*, *videos_frequency*, *features_path*, *feature_list*, *test=False*, *calcminmax_features=False*)

__init_features_data()

__init_video_data()

__init_whole_dataset()

__len__()

__module__ = 'dataset'

__orig_bases__ = (torch.utils.data.Dataset,)

dataset.**features_zip_to_dict**(*data*, *calcminmax_features*)

Prepares dictionary containing names of all AUDA package features for a video sequence

Parameters

data – Zip file to read from

Returns

Dictionary containing names of all AUDA package features for a video sequence and length of longest video

dataset.**load_features_aus**(*files*, *features_data*, *aus_feature_name*, *videos_frequency*, *sigmoid_mul*, *sigmoid_add*, *video_max_len*)

Loads AU based features to be used in model

Parameters

- **key** – Requested name
- **aus_feature_name** – Requested AU based feature name
- **features_data** – Zip file to read from
- **files** – Dictionary containing names of all AUDA package features for a video sequence
- **videos_frequency** – FPS used in videos
- **video_max_len** – Length of longest video
- **sigmoid_mul** – Sigmoid data normalization modifier -> narrows or expands the curve
- **sigmoid_add** – Sigmoid data normalization modifier -> offsets the curve

Returns

AU based features and their length

dataset.**load_features_auwise**(*files*, *features_data*, *sigmoid_mul*, *sigmoid_add*)

Loads AU-wise features to be used in model

Parameters

- **features_data** – Zip file to read from
- **files** – Dictionary containing names of all AUDA package features for a video sequence
- **sigmoid_mul** – Sigmoid data normalization modifier -> narrows or expands the curve
- **sigmoid_add** – Sigmoid data normalization modifier -> offsets the curve

Returns

AU-wise features

dataset.**load_features_crossau**(*files*, *features_data*)

Loads cross-AU features to be used in model

Parameters

- **features_data** – Zip file to read from
- **files** – Dictionary containing names of all AUDA package features for a video sequence

Returns

Cross-AU features

dataset.**load_features_si**(*files*, *features_data*, *key*, *videos_frequency*, *video_max_len*)

Loads smile intensities features to be used in model

Parameters

- **key** – Requested name
- **features_data** – Zip file to read from
- **files** – Dictionary containing names of all AUDA package features for a video sequence
- **videos_frequency** – FPS used in videos
- **video_max_len** – Length of longest video

Returns
Smile intensities features and their length

dataset.**load_frames**(*videos_data_names*, *videos_data*, *name*, *videos_frequency*, *video_max_len*)

Loads video frames to be used in model

Parameters

- **name** – Requested name
- **videos_data** – Zip file to read from
- **videos_data_names** – Dictionary containing names of all video frames for a video sequence
- **videos_frequency** – FPS used in videos
- **video_max_len** – Length of longest video

Returns
Video frames data and their length

dataset.**read_au_txt**(*zip*, *name*)

Reads AUDA sequential features (AU based) from zip file

Parameters

- **zip** – Zip file to read from
- **name** – Name of file to read

Returns
AU based features data

dataset.**read_auwise_txt**(*zip*, *name*)

Reads AUDA features (AU-wise and cross-AU) from zip file

Parameters

- **zip** – Zip file to read from
- **name** – Name of file to read

Returns
AUDA features data

dataset.**read_image**(*zip*, *name*)

Reads video frame from zip file

Parameters

- **zip** – Zip file to read from
- **name** – Name of file to read

Returns
Video frame data

dataset.**read_si_txt**(*zip*, *name*)

Reads AUDA sequential features (smile intensities) from zip file

Parameters

- **zip** – Zip file to read from
- **name** – Name of file to read

Returns
Smile intensities features data

dataset.**videos_zip_to_dict**(*data*)

Prepares dictionary containing names of all video frames for a video sequence

Parameters
data – Zip file to read from

Returns
Dictionary containing names of all video frames for a video sequence and length of longest video

1.4 extract_movie_frames module

extract_movie_frames.**extract_frames_from_database**(*input_videos_dir*, *output_zip_dir*, *output_json_dir*, *input_landmarks_dir=None*)

Builds zip with face-focused video frames in form of png files, from UVA-NEMO database that contains zip with mp4 files. Additionaly builds json file with all read videos

Parameters

- **input_videos_dir** – Original UVA-NEMO database
- **output_zip_dir** – Output zip directory
- **output_json_dir** – Output json file with all read videos
- **input_landmarks_dir** – DLIB landmark detector

extract_movie_frames.**get_face_from_image**(*img*, *alignment=False*)

Gets face focus from an image

Parameters
img – Original image

Returns
Faced focused image

extract_movie_frames.**write_movie_frames**(*flag*, *path*, *video_name*, *output_zip_name*)

Writes frame images from video

Parameters

- **path** – Path to video inside zip file
- **video_name** – Name of the video
- **output_zip_name** – Output zip with face-focused video frames

1.5 get_best_models module

get_best_models.**main**()

Summarizes trained single models and outputs saved parameters for best single models

1.6 model module

class model.**ConvLSTM**(*input_dim*, *hidden_dim*)

Bases: Module

PyTorch module for ConvLSTM

__init__(*input_dim*, *hidden_dim*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

__module__ = 'model'

_is_full_backward_hook: bool | None

forward(*input_tensor*, *time=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class model.**ConvLSTMCell**(*input_dim*, *hidden_dim*)

Bases: Module

PyTorch module for ConvLSTM Cell

__init__(*input_dim*, *hidden_dim*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

__module__ = 'model'

_is_full_backward_hook: bool | None

forward(*input_tensor*, *cur_state*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

init_hidden(*batch_size*, *height*, *width*)

training: bool

class model.**Convbn**(*ins*, *ous*, *kernel*, *padding=0*)

Bases: Module

PyTorch module for Convolutional layer+Batch Normalization with ReLU

__init__(*ins*, *ous*, *kernel*, *padding=0*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

__module__ = 'model'

_is_full_backward_hook: bool | None

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class model.**DeepSmileNet**(*f*)

Bases: Module

PyTorch module for RealSmileNet and AUDA features classification

__forward_au_features(*aus*, *aus_len*, *lstm_layer*, *cls_layer*)

Forwards AUDA Package's sequential features

__init__(*f*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

__module__ = 'model'

_fpn_layers(*cfg*, *in_channels=3*)

Creates module for RealSmileNet's FPN Block

_is_full_backward_hook: bool | None

forward(*x_videos*, *s*, *x_df_dict*, *frames_len*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class model.**MultipleDeepSmileNet**(*deepSmileNets*, *variant*)

Bases: Module

PyTorch module for multiple DeepSmileNet modules (aka. concatenation models)

__init__(*deepSmileNets*, *variant*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

__module__ = 'model'

_is_full_backward_hook: bool | None

forward(*x_videos*, *s*, *x_df_dict*, *frames_len*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class model.**NONLocalBlock2D**(*in_channels*, *inter_channels=None*, *sub_sample=True*, *bn_layer=True*)

Bases: _NonLocalBlockND

PyTorch module for NonLocalBlock2D

__init__(*in_channels*, *inter_channels=None*, *sub_sample=True*, *bn_layer=True*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

__module__ = 'model'

_is_full_backward_hook: bool | None

training: bool

class model.**TemporalAttension**(*channels*)

Bases: Module

PyTorch module for RealSmileNet's TSA Block

__init__(*channels*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

__module__ = 'model'

_is_full_backward_hook: bool | None

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class model.**_NonLocalBlockND**(*in_channels*, *inter_channels=None*, *dimension=3*, *sub_sample=True*, *bn_layer=True*)

Bases: Module

PyTorch module for NonLocalBlock

__init__(*in_channels*, *inter_channels=None*, *dimension=3*, *sub_sample=True*, *bn_layer=True*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

__module__ = 'model'

_is_full_backward_hook: bool | None

forward(*x*, *return_nl_map=False*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

1.7 net module

class net.**UVANEMO**(*epochs*, *lr*, *folds_path*, *videos_path*, *videos_frequency*, *features_path*, *batch_size_train*, *batch_size_valtest*, *calcminmax_features*)

Bases: object

Class used for model training on UVA-NEMO database

__calc_and_out_total_accloss(*prefix*, *epoch*, *pred_label*, *true_label*, *loss*, *accs_arr*, *losses_arr*)

Calculates and outputs total loss and accuracy in epoch

Parameters

- **prefix** – Prefix in output (whether it is training, validation or test set)
- **epoch** – Epoch index
- **pred_label** – Predicted labels
- **true_label** – Ground true labels
- **loss** – Total loss in epoch
- **accs_arr** – Array where accuracies should be stored
- **losses_arr** – Array where losses should be stored

Returns

Accuracy and loss in epoch

__calc_loss(*train*, *loss_func*, *pred*, *y*, *optimizer*)

Calculates loss and optimizes the model in batch

Parameters

- **pred** – Predicted labels from a batch
- **y** – Ground true labels
- **train** – Should optimization take place
- **optimizer** – Model optimizer
- **loss_func** – Used loss function

__calc_total_labels(*names*, *pred_label*, *labels_arr*)

Calculates predicted labels in epoch

Parameters

- **names** – Names of sequences
- **pred_label** – Predicted labels
- **labels_arr** – Array where predicted labels should be stored

__copy_loaders(*loader_data_path*, *target_path*)

Copies fold's labels to model training output path

Parameters

- **loader_data_path** – Directory with fold's sets
- **target_path** – Model training output path

__debug_params(*optimizer*, *prefix*)

Debugs model's parameters. Used to check if model properly freezes params

Parameters

- **optimizer** – Model optimizer
- **prefix** – Special file to debug this data

__determine_value(*name*)

Determines numerical variable for class name

Parameters

name – Class name

Returns

Numerical variable for class name

__dict__ = mappingproxy({'__module__': 'net', '__doc__': 'Class used for model training on UVA-NEMO database', 'epochs': None, 'lr': None, 'batch_size_train': None, 'batch_size_valtest': None, 'videos_frequency': None, 'calcminmax_features': None, 'folds_path': None, 'videos_path': None, 'features_path': None, '_UVANEMO__out_verbose': None, '__init__': <function UVANEMO.__init__>, '_UVANEMO__out': <function UVANEMO.__out>, '_UVANEMO__out_to_verbose': <function UVANEMO.__out_to_verbose>, '_UVANEMO__determine_value': <function UVANEMO.__determine_value>, 'split': <function UVANEMO.split>, '_UVANEMO__prepare_loaders': <function UVANEMO.__prepare_loaders>, '_UVANEMO__copy_loaders': <function UVANEMO.__copy_loaders>, '_UVANEMO__prepare_output_models': <function UVANEMO.__prepare_output_models>, '_UVANEMO__init_loaders': <function UVANEMO.__init_loaders>, '_UVANEMO__init_outputs': <function UVANEMO.__init_outputs>, 'prepare_data_single': <function UVANEMO.prepare_data_single>, 'prepare_data_concat': <function UVANEMO.prepare_data_concat>, '_UVANEMO__prepare_training_statistics': <function UVANEMO.__prepare_training_statistics>, '_UVANEMO__save_model': <function UVANEMO.__save_model>, '_UVANEMO__update_training_diagram': <function UVANEMO.__update_training_diagram>, '_UVANEMO__update_csv_labels': <function UVANEMO.__update_csv_labels>, '_UVANEMO__update_csv_lossacc_data': <function UVANEMO.__update_csv_lossacc_data>, '_UVANEMO__update_csv_labels_data': <function UVANEMO.__update_csv_labels_data>, '_UVANEMO__train_prepare': <function UVANEMO.__train_prepare>, '_UVANEMO__debug_params': <function UVANEMO.__debug_params>, '_UVANEMO__process_loader_data': <function UVANEMO.__process_loader_data>, '_UVANEMO__fit': <function UVANEMO.__fit>, '_UVANEMO__calc_loss': <function UVANEMO.__calc_loss>, '_UVANEMO__calc_and_out_total_accloss': <function UVANEMO.__calc_and_out_total_accloss>, '_UVANEMO__calc_total_labels': <function UVANEMO.__calc_total_labels>, '_UVANEMO__single_forward': <function UVANEMO.__single_forward>, '_UVANEMO__single_evaluate_forward': <function UVANEMO.__single_evaluate_forward>, 'single_train': <function UVANEMO.single_train>, '_UVANEMO__multi_forward': <function UVANEMO.__multi_forward>, '_UVANEMO__multi_evaluate_forward': <function UVANEMO.__multi_evaluate_forward>, 'multi_train': <function UVANEMO.multi_train>, '__dict__': <attribute '__dict__' of 'UVANEMO' objects>, '__weakref__': <attribute '__weakref__' of 'UVANEMO' objects>, '__annotations__': {}})

__fit(*x_video*, *s*, *x_df_dict*, *frames_len*, *device*)

Forwards data to model

Parameters

- **x_video** – Input features - Compressed video frames
- **x_df_dict** – Input features - AUDA Package features
- **device** – Device where model is stored
- **frames_len** – Lengths of sequence
- **s** – 'Reversed' lengths of sequence (element-wise MaxLengthInBatch-frames_len) (used in ConvLSTM)

Returns

Sigmoid output and predicted labels from a batch

__init__(*epochs*, *lr*, *folds_path*, *videos_path*, *videos_frequency*, *features_path*, *batch_size_train*, *batch_size_valtest*, *calcminmax_features*)

Class constructor

__init_loaders(*loader_data_path*, *folder_path*, *loader_features*)

Prepares loaders for training and copies fold's labels to model training output path

Parameters

- **loader_data_path** – Directory with fold's sets
- **folder_path** – Model training output path
- **loader_features** – Features for dataset that model will require

__init_outputs(*folder_path*)

Copies fold's labels to model training output path and initializes default verbose file

Parameters
folder_path – Model training output path

__module__ = 'net'

__multi_evaluate_forward(*epoch*, *prefix*, *accs_arr*, *losses_arr*, *labels_arr*, *data_loader*, *train*, *device*, *optimizer*, *loss_func*, *loss_penalty*)

Forwards and calculates data to model for a single epoch (concatenation model)

Parameters

- **epoch** – Epoch index
- **prefix** – Prefix in output (whether it is training, validation or test set)
- **accs_arr** – Array where accuracies should be stored
- **losses_arr** – Array where losses should be stored
- **labels_arr** – Array where predicted labels should be stored
- **device** – Device where model is stored
- **data_loader** – Data loader for either a training,validate or test set
- **train** – Should optimization take place in each batch
- **optimizer** – Model optimizer
- **loss_func** – Used loss function
- **loss_penalty** – Multiplier of loss function (for validate and tests in order to compare to training set)

Returns
Accuracy and loss in epoch

__multi_forward(*device*, *data_loader*, *train*, *optimizer*, *loss_func*)

Forwards data to model for a single epoch (concatenation model)

Parameters

- **device** – Device where model is stored
- **data_loader** – Data loader for either a training,validate or test set
- **train** – Should optimization take place in each batch
- **optimizer** – Model optimizer
- **loss_func** – Used loss function

Returns
Total loss, predicted labels, true labels

`__out`(*info*, *file*)

Writes display info to console and extra file

Parameters

- **info** – Information to display
- **file** – Extra text file to export information

`__out_to_verbose`(*info*)

Writes display info to console and default verbose file

Parameters
info – Information to display

`__out_verbose = None`

Default verbose file path

`__prepare_loaders`(*loader_data_path*, *features*)

Prepares loaders for training

Parameters

- **loader_data_path** – Directory with fold's sets
- **features** – Features for dataset that model will require

`__prepare_output_models`(*target_path*)

Copies fold's labels to model training output path

Parameters

- **loader_data_path** – Directory with fold's sets
- **target_path** – Model training output path

`__prepare_training_statistics`()

Prepares statistics of training process (accuracies, losses, predicted labels etc.

`__process_loader_data`(*x_video*, *y*, *s*, *x_df_dict*, *device*)

Process data that comes from one of loader's datasets

Parameters

- **x_video** – Input features - Compressed video frames
- **x_df_dict** – Input features - AUDA Package features
- **device** – Device where model is stored
- **y** – Ground true labels
- **s** – 'Reversed' lengths of sequence (element-wise MaxLengthInBatch-frames_len)

Returns
Processed data

`__save_model`(*e*, *va*, *vl*, *ta*, *tl*)

Saves model's hyperparameters

Parameters

- **e** – Epoch used in file name

- **va** – Validate set accuracy used in file name
- **vl** – Validate set loss used in file name
- **ta** – Test set accuracy used in file name
- **tl** – Test set loss used in file name

__single_evaluate_forward(*epoch*, *prefix*, *accs_arr*, *losses_arr*, *labels_arr*, *data_loader*, *train*, *device*, *optimizer*, *loss_func*, *loss_penalty*)

Forwards and calculates data to model for a single epoch (single model)

Parameters

- **epoch** – Epoch index
- **prefix** – Prefix in output (whether it is training, validation or test set)
- **accs_arr** – Array where accuracies should be stored
- **losses_arr** – Array where losses should be stored
- **labels_arr** – Array where predicted labels should be stored
- **device** – Device where model is stored
- **data_loader** – Data loader for either a training,validate or test set
- **train** – Should optimization take place in each batch
- **optimizer** – Model optimizer
- **loss_func** – Used loss function
- **loss_penalty** – Multiplier of loss function (for validate and tests in order to compare to training set)

Returns

Accuracy and loss in epoch

__single_forward(*device*, *data_loader*, *train*, *optimizer*, *loss_func*)

Forwards data to model for a single epoch (single model)

Parameters

- **device** – Device where model is stored
- **data_loader** – Data loader for either a training,validate or test set
- **train** – Should optimization take place in each batch
- **optimizer** – Model optimizer
- **loss_func** – Used loss function

Returns

Total loss, predicted labels, true labels

__train_prepare(*idx*)

Prepares training process (model initialization, GPU usage, optimizer)

Parameters

idx – Index of fold cycle used in training

Returns

Device where model is stored, optimizer, loss function used

__update_csv_labels(*filename*, *pred_labels*, *true_labels*)

Updates predicted labels data to csv file

Parameters

- **filename** – Output csv name
- **pred_labels** – Data containing predicted labels
- **true_labels** – Data containing ground true labels

__update_csv_labels_data()

Updates predicted labels data to csv files

__update_csv_lossacc_data()

Updates learning curves data to csv file

__update_training_diagram()

Updates learning curves.

__weakref__

list of weak references to the object (if defined)

batch_size_train = None

Size of minibatches in training set

batch_size_valtest = None

Size of minibatches in validate and testing set

calcminmax_features = None

Debugging variable - used to display AUDA statistics

epochs = None

Number of epochs used in model training

features_path = None

Path to zip with UVA-NEMO based AUDA features

folds_path = None

Path to cross-validation method folds' labels

lr = None

Learning rate used in model optizmizer

multi_train(*idx*)

Performs training of concatenation model

Parameters

idx – Index of fold cycle used in training

prepare_data_concat(*idx*, *working_dir*, *state_dir*, *variant*, *ignore*)

Prepares concatenation model for training

Parameters

- **idx** – Index of fold cycle used in training
- **working_dir** – Model training output path
- **state_dir** – Directory containing pretrained models
- **variant** – Variant of concatenation

- **ignore** – By default all pretrained models are used in training. This parameter adds option to ignore some of them

prepare_data_single(*idx*, *working_dir*, *features*)

Prepares single model for training

Parameters

- **idx** – Index of fold cycle used in training
- **working_dir** – Model training output path
- **features** – Features for dataset that model will require

single_train(*idx*)

Performs training of single model

Parameters

idx – Index of fold cycle used in training

split(*label_path*, *folds_path*)

Changes 10 folds containing video sequences to 10 cross-validation cycles training,validation and testing sets

Parameters

- **label_path** – Directory with 10 folds
- **folds_path** – Save directory for cycles sets

videos_frequency = None

FPS used in videos

videos_path = None

Path to zip with UVA-NEMO videos

1.8 split_labels module

split_labels.**main**()

In single trained single models splits predicted labels between spontaneus and deliberate ones

1.9 start module

start.**main**()

Starts process of single and concatenation models training and validation

- **ref**
  genindex
- **ref**
  modindex
- **ref**
  search

PYTHON MODULE INDEX

S

INDEX

Symbols

T

U

V

W