

---

# Smile Recognition

*Release 1.0*

**Benedykt Pawlus**

**Sep 04, 2023**



**CONTENTS:**

<b>1</b>	<b>smilerecognition</b>	<b>1</b>
1.1	concat_test module . . . . .	1
1.2	constants module . . . . .	1
1.3	dataset module . . . . .	1
1.4	extract_movie_frames module . . . . .	4
1.5	get_best_models module . . . . .	5
1.6	model module . . . . .	5
1.7	net module . . . . .	8
1.8	split_labels module . . . . .	15
1.9	start module . . . . .	15
	<b>Python Module Index</b>	<b>17</b>
	<b>Index</b>	<b>19</b>



## SMILERECOGNITION

## 1.1 concat\_test module

`concat_test.main()`

Performs calculation of best models to concatenate with starting model

## 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_au(files, features_data, au_feature_name, videos_frequency, sigmoid_mul, sigmoid_add, video_max_len)`

Loads AU based features to be used in model

### Parameters

- **key** – Requested name
- **au\_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. Additionally 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 models and outputs saved parameters for best 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\_aud\_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, 'calcmimmax_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*, *calcmimmax\_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

**calcmixmap\_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 optimizer

**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()**

Splits predicted labels between spontaneous and deliberate ones

## 1.9 start module

**start.main()**

Starts training and validation processes of single and concatenation models

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



## PYTHON MODULE INDEX

### C

`concat_test`, [1](#)

### g

`get_best_models`, [5](#)

### S

`split_labels`, [15](#)

`start`, [15](#)



## Symbols

`_NonLocalBlockND` (class in model), 8  
`__calc_and_out_total_accloss()` (net.UVANEMO method), 8  
`__calc_loss()` (net.UVANEMO method), 8  
`__calc_total_labels()` (net.UVANEMO method), 9  
`__copy_loaders()` (net.UVANEMO method), 9  
`__debug_params()` (net.UVANEMO method), 9  
`__determine_value()` (net.UVANEMO method), 9  
`__dict__` (net.UVANEMO attribute), 9  
`__fit()` (net.UVANEMO method), 10  
`__forward_a_u_features()` (model.DeepSmileNet method), 6  
`__getitem__()` (dataset.UVANEMODataGenerator method), 1  
`__init__()` (dataset.UVANEMODataGenerator method), 1  
`__init__()` (model.ConvLSTM method), 5  
`__init__()` (model.ConvLSTMCell method), 5  
`__init__()` (model.Convbn method), 6  
`__init__()` (model.DeepSmileNet method), 6  
`__init__()` (model.MultipleDeepSmileNet method), 7  
`__init__()` (model.NONLocalBlock2D method), 7  
`__init__()` (model.TemporalAttension method), 7  
`__init__()` (model.\_NonLocalBlockND method), 8  
`__init__()` (net.UVANEMO method), 10  
`__init_features_data()` (dataset.UVANEMODataGenerator method), 1  
`__init_loaders()` (net.UVANEMO method), 10  
`__init_outputs()` (net.UVANEMO method), 11  
`__init_video_data()` (dataset.UVANEMODataGenerator method), 1  
`__init_whole_dataset()` (dataset.UVANEMODataGenerator method), 1  
`__len__()` (dataset.UVANEMODataGenerator method), 1  
`__module__` (dataset.UVANEMODataGenerator attribute), 1  
`__module__` (model.ConvLSTM attribute), 5  
`__module__` (model.ConvLSTMCell attribute), 5  
`__module__` (model.Convbn attribute), 6  
`__module__` (model.DeepSmileNet attribute), 6  
`__module__` (model.MultipleDeepSmileNet attribute), 7  
`__module__` (model.NONLocalBlock2D attribute), 7  
`__module__` (model.TemporalAttension attribute), 7  
`__module__` (model.\_NonLocalBlockND attribute), 8  
`__module__` (net.UVANEMO attribute), 11  
`__multi_evaluate_forward()` (net.UVANEMO method), 11  
`__multi_forward()` (net.UVANEMO method), 11  
`__orig_bases__` (dataset.UVANEMODataGenerator attribute), 1  
`__out__()` (net.UVANEMO method), 12  
`__out_to_verbose()` (net.UVANEMO method), 12  
`__out_verbose` (net.UVANEMO attribute), 12  
`__prepare_loaders()` (net.UVANEMO method), 12  
`__prepare_output_models()` (net.UVANEMO method), 12  
`__prepare_training_statistics()` (net.UVANEMO method), 12  
`__process_loader_data()` (net.UVANEMO method), 12  
`__save_model()` (net.UVANEMO method), 12  
`__single_evaluate_forward()` (net.UVANEMO method), 13  
`__single_forward()` (net.UVANEMO method), 13  
`__train_prepare()` (net.UVANEMO method), 13  
`__update_csv_labels()` (net.UVANEMO method), 13  
`__update_csv_labels_data()` (net.UVANEMO method), 14  
`__update_csv_lossacc_data()` (net.UVANEMO method), 14  
`__update_training_diagram()` (net.UVANEMO method), 14  
`__weakref__` (net.UVANEMO attribute), 14  
`_fpn_layers()` (model.DeepSmileNet method), 6  
`_is_full_backward_hook` (model.ConvLSTM attribute), 5  
`_is_full_backward_hook` (model.ConvLSTMCell attribute), 5  
`_is_full_backward_hook` (model.Convbn attribute), 6  
`_is_full_backward_hook` (model.DeepSmileNet attribute), 6  
`_is_full_backward_hook`

*(model.MultipleDeepSmileNet attribute), 7*  
*\_is\_full\_backward\_hook (model.NONLocalBlock2D attribute), 7*  
*\_is\_full\_backward\_hook (model.TemporalAttension attribute), 7*  
*\_is\_full\_backward\_hook (model.\_NonLocalBlockND attribute), 8*

## B

*batch\_size\_train (net.UVANEMO attribute), 14*  
*batch\_size\_valtest (net.UVANEMO attribute), 14*

## C

*calcmixmap\_features (net.UVANEMO attribute), 14*  
*concat\_test module, 1*  
*constants module, 1*  
*Convbn (class in model), 6*  
*ConvLSTM (class in model), 5*  
*ConvLSTMCell (class in model), 5*

## D

*dataset module, 1*  
*DeepSmileNet (class in model), 6*

## E

*epochs (net.UVANEMO attribute), 14*  
*extract\_frames\_from\_database() (in module extract\_movie\_frames), 4*  
*extract\_movie\_frames module, 4*

## F

*features\_path (net.UVANEMO attribute), 14*  
*features\_zip\_to\_dict() (in module dataset), 1*  
*folds\_path (net.UVANEMO attribute), 14*  
*forward() (model.\_NonLocalBlockND method), 8*  
*forward() (model.Convbn method), 6*  
*forward() (model.ConvLSTM method), 5*  
*forward() (model.ConvLSTMCell method), 5*  
*forward() (model.DeepSmileNet method), 6*  
*forward() (model.MultipleDeepSmileNet method), 7*  
*forward() (model.TemporalAttension method), 7*

## G

*get\_best\_models module, 5*  
*get\_face\_from\_image() (in module extract\_movie\_frames), 4*

## I

*init\_hidden() (model.ConvLSTMCell method), 5*

## L

*load\_features\_au() (in module dataset), 1*  
*load\_features\_auwise() (in module dataset), 2*  
*load\_features\_crossau() (in module dataset), 2*  
*load\_features\_si() (in module dataset), 2*  
*load\_frames() (in module dataset), 3*  
*lr (net.UVANEMO attribute), 14*

## M

*main() (in module concat\_test), 1*  
*main() (in module get\_best\_models), 5*  
*main() (in module split\_labels), 15*  
*main() (in module start), 15*  
*model module, 5*  
*module concat\_test, 1*  
*constants, 1*  
*dataset, 1*  
*extract\_movie\_frames, 4*  
*get\_best\_models, 5*  
*model, 5*  
*net, 8*  
*split\_labels, 15*  
*start, 15*  
*multi\_train() (net.UVANEMO method), 14*  
*MultipleDeepSmileNet (class in model), 6*

## N

*net module, 8*  
*NONLocalBlock2D (class in model), 7*

## P

*prepare\_data\_concat() (net.UVANEMO method), 14*  
*prepare\_data\_single() (net.UVANEMO method), 15*

## R

*read\_au\_txt() (in module dataset), 3*  
*read\_auwise\_txt() (in module dataset), 3*  
*read\_image() (in module dataset), 3*  
*read\_si\_txt() (in module dataset), 3*

## S

*single\_train() (net.UVANEMO method), 15*  
*split() (net.UVANEMO method), 15*  
*split\_labels module, 15*  
*start module, 15*



## T

TemporalAttension (*class in model*), 7  
 training (*model.\_NonLocalBlockND attribute*), 8  
 training (*model.Convbn attribute*), 6  
 training (*model.ConvLSTM attribute*), 5  
 training (*model.ConvLSTMCell attribute*), 5  
 training (*model.DeepSmileNet attribute*), 6  
 training (*model.MultipleDeepSmileNet attribute*), 7  
 training (*model.NONLocalBlock2D attribute*), 7  
 training (*model.TemporalAttension attribute*), 7

## U

UVANEMO (*class in net*), 8  
 UVANEMODataGenerator (*class in dataset*), 1

## V

videos\_frequency (*net.UVANEMO attribute*), 15  
 videos\_path (*net.UVANEMO attribute*), 15  
 videos\_zip\_to\_dict() (*in module dataset*), 4

## W

write\_movie\_frames() (*in module extract\_movie\_frames*), 4