DIME

Release 1.0

Felix Tempel, Pål Haugen

Aug 18, 2025

CONTENTS:

The repository to unite research and development.

> **Note**
>
> All data remains safely at your computer during use.

CHAPTER

ONE

USAGE

1.1 Installation

You have to be member of the DeepInMotion Github project. If that applies to you you can download the project via:

```
$ git clone https://github.com/DeepInMotion/DIME.git
```

1.2 Python

To use DIME, first install the required packages using pip within a virtual environment:

```
(.venv) $ pip install -r requirements.txt
```

Now you can start the application as:

```
(.venv) $ python -m shiny run --host 0.0.0.0 --port 8000 ./src/dashboard/app.py
```

1.3 Docker

To use DIME within a Docker container make sure you have installed Docker on your machine. You can download Docker here.

Afterwards you can build the docker image with:

```
$ docker compose build dashboard
```

If you want to rebuild it use:

```
$ docker compose build --no-cache
```

After the docker image is build you can start the application via:

```
$ docker compose up dashboard
```

Now you can access the application at port 8000 in your browser under http://localhost:8000/

1.4 Basic Usage

The user can upload a video on the left side and insert basic information about the infant. When a video is uploaded the analysis can be initiated with the "Run" button. After the analysis is finished the results will be displayed in the different panels.

The first panel shows just the basic information for the patient and the satus of the CP risk score, this is ment as an overview only.

The second tab "Analysis Results" shows the video of the infant with the skeleton graph superimposed. In adition detail rearding the risk scores for the entire video is displayed and compared to distributions from videos of typical CP positive and typical CP negative videos.

The third tab "XAI Visualization" shows a video of the infant with the skeleton graph superimposed. The bodyparts that the model focuses on are highlighted in red. At the same time the cp risk score for the time window is displayed below the video.

The fourth tab "Original Video" shows the original video without any superimposed information.

The fith tab "Additional Infos" describes the scientific background for the model, and key metrics for training and testing.

1.5 Future Improvements

This application needs future maintenance. The JAMA code from Gross et. al is not designed in an object orientated way and is hard to reuse. The improvement of this codebase is out of scope for the Deep In Motion research project. To make future integrations easier, as a first step the tracking functionality has been modularised. This makes it possible to integrate other models and the SHAP explanation, such as the on from Tempel et. al.

CHAPTER

TWO

API REFERENCE

This page contains auto-generated API reference documentation[1].

2.1 src

2.1.1 Submodules

2.1.1.1 src.cp_prediction

2.1.1.1.1 Submodules

src.cp_prediction.models

Submodules

src.cp_prediction.models.gcn_search_model_tc_cam

Classes

Swish	Swish activation function.
SELayer	SELayer.
Channel_Att	Channel Attention Module.
Frame_Att	Frame Attention Module.
Joint_Att	Joint Attention Module.
GraphConv	Basic Graph Convolution module.
Spatial_Bottleneck_Block	Spatial Bottleneck Block
Spatial_MBConv_Block	Spatial MBConv Block.
Temporal_Bottleneck_Block	Temporal Bottleneck Block.
Temporal_MBConv_Block	Temporal Convolution Block using residual block.
st_gcn	Spatial-Temporal Graph Convolution module.
Model	Spatial-Temporal Graph Convolutional Network (ST-GCN).

Module Contents

class `src.cp_prediction.models.gcn_search_model_tc_cam.`**`Swish`**(**args*, ***kwargs*)

Bases: `torch.nn.Module`

Swish activation function. Adopted from: # https://github.com/narumiruna/efficientnet-pytorch

[1] Created with sphinx-autoapi

forward(*x*)

class src.cp_prediction.models.gcn_search_model_tc_cam.**SELayer**(*channel*, *reduction=4*, *swish_nonlinearity=False*)

Bases: torch.nn.Module

SELayer. Adopted from: https://github.com/moskomule/senet.pytorch/blob/master/senet/se_module.py

avg_pool

forward(*x*)

class src.cp_prediction.models.gcn_search_model_tc_cam.**Channel_Att**(*channels*, *reduction=4*, *swish_nonlinearity=False*, ***kwargs*)

Bases: torch.nn.Module

Channel Attention Module. Adopted from: https://github.com/yfsong0709/ResGCNv1/blob/master/src/model/attentions.py

bn

forward(*x*)

class src.cp_prediction.models.gcn_search_model_tc_cam.**Frame_Att**(*channels*, *kernel_size*, *swish_nonlinearity=False*, ***kwargs*)

Bases: torch.nn.Module

Frame Attention Module. Adopted from https://github.com/yfsong0709/ResGCNv1/blob/master/src/model/attentions.py

avg_pool

max_pool

conv

bn

forward(*x*)

class src.cp_prediction.models.gcn_search_model_tc_cam.**Joint_Att**(*channels*, *num_joints*, *swish_nonlinearity=False*, ***kwargs*)

Bases: torch.nn.Module

Joint Attention Module. Adopted from´: https://github.com/yfsong0709/ResGCNv1/blob/master/src/model/attentions.py

bn

forward(*x*)

class src.cp_prediction.models.gcn_search_model_tc_cam.**GraphConv**(*in_channels*, *out_channels*, *kernel_size*, *t_kernel_size=1*, *t_stride=1*, *t_padding=0*, *t_dilation=1*, *bias=True*, *groups=1*)

Bases: `torch.nn.Module`

Basic Graph Convolution module.

Adapted from: https://github.com/yysijie/st-gcn/blob/master/net/utils/tgcn.py

Parameters

- **in_channels** (*int*) – Number of channels in the input sequence data.
- **out_channels** (*int*) – Number of channels produced by the convolution.
- **kernel_size** (*int*) – Size of the graph convolution kernel.
- **t_kernel_size** (*int*) – Size of the temporal convolution kernel.
- **t_stride** (*int, optional*) – Stride of the temporal convolution. Default is 1.
- **t_padding** (*int, optional*) – Temporal zero-padding added to both sides of the input. Default is 0.
- **t_dilation** (*int, optional*) – Spacing between temporal kernel elements. Default is 1.
- **bias** (*bool, optional*) – If True, adds a learnable bias to the output. Default is True.
- **groups** (*int, optional*) – Number of groups in the convolution. Default is 1.

Shape:

- Input[0]: Input graph sequence tensor of shape (N, C_{in}, T_{in}, V)
- Input[1]: Adjacency matrix tensor of shape (K, V, V)
- Output[0]: Output graph sequence tensor of shape (N, C_{out}, T_{out}, V)
- Output[1]: Adjacency matrix for output of shape (K, V, V)

Where:

- N is the batch size
- C_{in} / C_{out} are input/output channels
- T_{in} / T_{out} are input/output sequence lengths
- V is the number of graph nodes
- $K = kernel\_size$ is the spatial kernel size

kernel_size

conv

forward(x, A)

class `src.cp_prediction.models.gcn_search_model_tc_cam.`**`Spatial_Bottleneck_Block`**(*in_channels*, *out_channels*, *kernel_size*, *residual=False*, *reduction=4*, *basic=False*, *se_ratio=0*, *swish_nonlinearity=False*, ***kwargs*)

Bases: `torch.nn.Module`

Spatial Bottleneck Block https://github.com/yfsong0709/ResGCNv1/blob/master/src/model/blocks.py

`basic = False`

`se_ratio = 0`

`conv_down`

`bn_down`

`bn`

`conv_up`

`bn_up`

`forward`(*x*, *A*)

class `src.cp_prediction.models.gcn_search_model_tc_cam.`**`Spatial_MBConv_Block`**(*in_channels*, *out_channels*, *kernel_size*, *residual=False*, *expansion=4*, *se_ratio=0*, *swish_nonlinearity=False*, ***kwargs*)

Bases: `torch.nn.Module`

Spatial MBConv Block. Adopted from: https://github.com/lukemelas/EfficientNet-PyTorch/blob/master/efficientnet_pytorch/model.py

`expansion = 4`

`se_ratio = 0`

`conv_up`

`bn_up`

`depthwise_conv`

`bn`

`conv_down`

bn_down

forward(*x*, *A*)

class src.cp_prediction.models.gcn_search_model_tc_cam.**Temporal_Bottleneck_Block**(*channels*, *kernel_size*, *stride=1*, *residual=False*, *reduction=4*, *dropout_factor=0.0*, *basic=False*, *inner_se_ratio=0*, *outer_se_ratio=0*, *scales=1*, *swish_nonlinearity=False*, ***kwargs*)

Bases: `torch.nn.Module`

Temporal Bottleneck Block. Adopted from: https://github.com/yfsong0709/ResGCNv1/blob/master/src/model/blocks.py

basic = False

inner_se_ratio = 0

outer_se_ratio = 0

temporal_branches

conv

bn

forward(*x*, *res_module*)

class src.cp_prediction.models.gcn_search_model_tc_cam.**Temporal_MBConv_Block**(*channels*, *kernel_size*, *stride=1*, *residual=False*, *expansion=4*, *dropout_factor=0.0*, *inner_se_ratio=0*, *outer_se_ratio=0*, *scales=1*, *swish_nonlinearity=False*, ***kwargs*)

Bases: `torch.nn.Module`

Temporal Convolution Block using residual block. Adopted from: https://github.com/lukemelas/EfficientNet-PyTorch/blob/master/efficientnet_pytorch/model.py

expansion = 4

inner_se_ratio = 0

outer_se_ratio = 0

temporal_branches

conv

bn

forward(*x*, *res_module*)

class src.cp_prediction.models.gcn_search_model_tc_cam.**st_gcn**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *dropout=0.0*, *reduction=4*, *expansion=4*, *block_type='basic'*, *inner_se_ratio=0*, *outer_se_ratio=0*, *temporal_scales=1*, *attention='null'*, *num_joints=19*, *residual='dense'*, *swish_nonlinearity=False*)

Bases: `torch.nn.Module`

Spatial-Temporal Graph Convolution module.

Adapted from: https://github.com/yysijie/st-gcn/blob/master/net/st_gcn.py

Parameters

- **in_channels** (*int*) – Number of channels in the input sequence data.
- **out_channels** (*int*) – Number of channels produced by the convolution.
- **kernel_size** (*tuple*) – Size of the temporal and graph convolution kernels.
- **stride** (*int, optional*) – Stride of the temporal convolution. Default is 1.
- **dropout** (*float, optional*) – Dropout rate. Default is 0.0.
- **reduction** (*int, optional*) – Compression factor in the bottleneck convolution. Default is 4.
- **expansion** (*int, optional*) – Expansion factor in the inverted bottleneck convolution. Default is 6.
- **block_type** (*str, optional*) – Type of micro-block architecture to use. Default is 'basic'.
- **inner_se_ratio** (*int, optional*) – Reduction ratio for inner Squeeze-and-Excitation (SE) block. If 0, SE is not applied. Default is 0.
- **outer_se_ratio** (*int, optional*) – Reduction ratio for outer Squeeze-and-Excitation (SE) block. If 0, SE is not applied. Default is 0.
- **temporal_scales** (*int, optional*) – Number of scales in multi-scale temporal convolution. Default is 1.
- **attention** (*str, optional*) – Type of attention mechanism to apply in the GCN module. Default is 'null'.

- **num_joints** (*int, optional*) – Number of joints in the skeleton. Default is 19 (In-Motion skeleton).
- **residual** (*str, optional*) – Type of residual connection to apply. Default is 'dense'.
- **swish_nonlinearity** (*bool, optional*) – If True, uses the Swish activation function throughout the network; otherwise, uses ReLU. Default is False.

Shape:

- Input[0]: Graph sequence tensor of shape (N, C_{in}, T_{in}, V)
- Input[1]: Adjacency matrix of shape (K, V, V)
- Output[0]: Output feature tensor of shape (N, C_{out}, T_{out}, V)
- Output[1]: Adjacency matrix for output of shape (K, V, V)

Where:

- N is the batch size
- C_{in} / C_{out} are input/output channels
- T_{in} / T_{out} are input/output sequence lengths
- V is the number of graph nodes (joints)
- $K = kernel\_size[1]$ is the spatial kernel size

`attention = None`

`forward`(x, A)

class `src.cp_prediction.models.gcn_search_model_tc_cam.Model`(*num_class*, *graphs*, *in_channels=6*, *edge_importance_weighting=True*, *dropout=0.0*, *num_input_branches=3*, *attention='null'*, *spatial_pool=False*, *se_outer=False*, *se_inner=False*, *initial_residual='null'*, *residual='dense'*, *initial_block_type='basic'*, *block_type='basic'*, *input_width=16*, *initial_main_width=32*, *temporal_kernel_size=9*, *num_input_modules=3*, *num_main_levels=2*, *num_main_level_modules=2*, *input_temporal_scales=[1, 1, 1]*, *main_temporal_scales=[1, 1]*, *bottleneck_factor=4*, *se_ratio=4*, *relative_se=False*, *swish_nonlinearity=False*, *top_block=[]*, ***kwargs*)

Bases: `torch.nn.Module`

Spatial-Temporal Graph Convolutional Network (ST-GCN). Based on: https://github.com/yysijie/st-gcn/blob/master/net/st_gcn.py

Parameters

- **num_class** (*int*) – Number of output classes for the classification task.
- **graphs** (*list*) – List of graphs used for spatial graph convolutions.
- **in_channels** (*int*) – Number of input channels.
- **edge_importance_weighting** (*bool*) – If True, adds learnable weights to the edges of the graph.
- **dropout** (*float, optional*) – Dropout rate. Default: 0.0.
- **num_input_branches** (*int, optional*) – Number of input modalities (branches). Default: 3.
- **attention** (*str, optional*) – Type of attention mechanism used in the GCN module. Default: 'null'.
- **spatial_pool** (*bool, optional*) – If True, applies global spatial pooling before classification. Default: False.
- **se_outer** (*bool, optional*) – If True, applies outer Squeeze-and-Excitation. Default: False.
- **se_inner** (*bool, optional*) – If True, applies inner Squeeze-and-Excitation. Default: False.
- **initial_residual** (*str, optional*) – Residual type for the initial layer. Default: 'null'.
- **residual** (*str, optional*) – Residual type used throughout the network. Default: 'dense'.
- **initial_block_type** (*str, optional*) – Block architecture used in the initial layer. Default: 'basic'.
- **block_type** (*str, optional*) – Block architecture used throughout the network. Default: 'basic'.
- **input_width** (*int, optional*) – Number of channels in the first input convolutional layer. Default: 16.
- **initial_main_width** (*int, optional*) – Number of channels in the first convolution of the main branch. Default: 32.
- **temporal_kernel_size** (*int, optional*) – Kernel size used for temporal convolutions. Default: 9.
- **num_input_modules** (*int, optional*) – Number of ST-GCN modules in each input branch. Default: 3.
- **num_main_levels** (*int, optional*) – Number of abstraction levels in the main branch. Default: 2.
- **num_main_level_modules** (*int, optional*) – Modules per abstraction level in the main branch. Default: 2.
- **input_temporal_scales** (*list of int, optional*) – Temporal scales for multi-scale convolution in each input block. Default: [1, 1].
- **main_temporal_scales** (*list of int, optional*) – Temporal scales for multi-scale convolution in the main branch. Default: [1, 1, 1].
- **bottleneck_factor** (*int, optional*) – Factor for channel reduction/expansion in bottleneck or MBConv blocks. Default: 4.

- **se_ratio** (*int, optional*) – Downsampling factor in Squeeze-and-Excitation layers. Default: 4.
- **relative_se** (*bool, optional*) – If True, scales SE downsampling ratio relative to layer width. Default: False.
- **swish_nonlinearity** (*bool, optional*) – If True, uses Swish activation instead of ReLU. Default: False.
- ****kwargs** – Additional keyword arguments for graph convolution modules.

Shape:

- **Input:** (N, C_{in}, T, V, M) **where:**
  - N is the batch size,
  - C_{in} is the number of input channels,
  - T is the temporal length (frames),
  - V is the number of graph nodes (joints),
  - M is the number of instances/persons in each frame.
- Output: $(N, numclass)$

in_channels_per_branch = 2

spatial_pool = False

swish_nonlinearity = False

num_input_branches = 3

num_input_modules = 3

graphs

graph_input

num_joints

data_bn

st_gcn_input

bottleneck_bn

st_gcn_main

fcn

forward(x)

src.cp_prediction.predict

Attributes

logger
BASE_OUTPUT_DIR
OUTPUT_DIR
args

Functions

config_logger()	Logging configuration.
is_docker()	Check if docker is used.
predict(tracking_coords, body_parts, frame_rate[, ...])	Main callable for prediction.
infer_predict(video_path[, store, visualize, group, ...])	Perform CP prediction inference on supplied video.
predict_video(→ bool)	Predict CP risk on supplied video.
main(file_path, visualize, store, tracked, group, ...)	Main program for performing tracking and prediction of cerebral palsy from video of infant spontaneous movements.

Module Contents

src.cp_prediction.predict.**config_logger**()

Logging configuration.

Returns
None

src.cp_prediction.predict.**logger**

src.cp_prediction.predict.**is_docker**()

Check if docker is used.

Returns
bool

src.cp_prediction.predict.**BASE_OUTPUT_DIR**

src.cp_prediction.predict.**OUTPUT_DIR**

src.cp_prediction.predict.**predict**(*tracking_coords: numpy.ndarray*, *body_parts: list*, *frame_rate: float*, *pred_frame_rate: float = 30.0*, *window_stride: int = 2*, *num_models: int = 10*, *num_portions: int = 7*, *prediction_threshold: float = 0.350307*, *xai_technique: str = 'cam'*)

Main callable for prediction.

Parameters

- `tracking_coords`
- `body_parts`
- `frame_rate`
- `pred_frame_rate`
- `window_stride`
- `num_models`
- `num_portions`
- `prediction_threshold`
- `xai_technique`

Returns:

`src.cp_prediction.predict.`**`infer_predict`**(*video_path*, *store=False*, *visualize=False*, *group=False*, *binary=False*, *overlay=False*, *color_scheme='GYOR'*, *xai_technique='cam'*, *mask=False*, *mask_expansion=0.5*, *output_dir=OUTPUT_DIR*)

Perform CP prediction inference on supplied video. :param video_path: System path of video to analyze :param store: boolean

Flag to create CSV file with predicted CP risk values

Parameters

- **`visualize`** – boolean Flag to create visualization of CAM
- **`group`** – boolean Flag for grouping body keypoints in CAM visualization
- **`binary`** – boolean Flag for two-color visualization of CAM
- **`overlay`** – boolean Flag for overlay with window CP risk and uncertainty in visualization
- **`color_scheme`** – string Combination of colors to use for CAM visualization (e.g., 'GYOR' for 'G' = green, 'Y' = yellow, 'O' = orange and 'R' = red)
- **`xai_technique`** – string Explainable AI (XAI) technique for estimating contribution of body keypoints ('cam' or 'gradcam')
- **`mask`** – boolean Flag for generating separate video with face masked
- **`mask_expansion`** – float Expansion/padding value to set size of face mask

Returns

CP risk values of the supplied video.

`src.cp_prediction.predict.`**`predict_video`**(*file_path*, *store=False*, *visualize=True*, *group=False*, *binary=False*, *overlay=False*, *color_scheme='GYOR'*, *xai_technique='cam'*, *mask=False*, *mask_expansion=0.5*, *output_dir=OUTPUT_DIR*) → bool

Predict CP risk on supplied video.

Parameters

- **`file_path`** – path System path of video to analyze
- **`store`** – boolean Flag to create CSV file with predicted CP risk values
- **`visualize`** – boolean Flag to create visualization of CAM

- **group** – boolean Flag for grouping body keypoints in CAM visualization
- **binary** – boolean Flag for two-color visualization of CAM
- **overlay** – boolean Flag for overlay with window CP risk and uncertainty in visualization
- **color_scheme** – string Combination of colors to use for CAM visualization (e.g., 'GYOR')
- **xai_technique** – string Explainable AI (XAI) technique ('cam' or 'gradcam')
- **mask** – boolean Flag for generating separate video with face masked
- **mask_expansion** – float Expansion/padding value to set size of face mask
- **output_dir** – path (optional) Directory to store output files

Returns
True if successful, False if not

`src.cp_prediction.predict.`**`main`**`(`*file_path*, *visualize*, *store*, *tracked*, *group*, *binary*, *overlay*, *color_scheme*, *xai_technique*, *mask*, *mask_expansion*`)`

Main program for performing tracking and prediction of cerebral palsy from video of infant spontaneous movements. :param file_path: :param visualize: :param store: :param tracked: :param group: :param binary: :param overlay: :param color_scheme: :param xai_technique: :param mask: :param mask_expansion:

Returns
bool if successful, False if not

`src.cp_prediction.predict.`**`args`**

src.cp_prediction.utils

Module for useful utility objects.

Submodules

src.cp_prediction.utils.feeder

Classes

EvalFeeder	An abstract class representing a `Dataset`.

Module Contents

class `src.cp_prediction.utils.feeder.`**`EvalFeeder`**`(`*data*, *graph*, *window_size=150*, *parts_distance=75*, *standardize_rotation=True*, *absolute=True*, *relative=False*, *motion1=True*, *motion2=False*, *bone=True*, *bone_angle=False*, *debug=False*`)`

Bases: `torch.utils.data.Dataset`

An abstract class representing a `Dataset`.

All datasets that represent a map from keys to data samples should subclass it. All subclasses should overwrite *`__getitem__()`*, supporting fetching a data sample for a given key. Subclasses could also optionally overwrite *`__len__()`*, which is expected to return the size of the dataset by many `Sampler` implementations and the default options of `DataLoader`. Subclasses could also optionally implement `__getitems__()`, for speedup batched samples loading. This method accepts list of indices of samples of batch and returns list of samples.

> **Note**
>
> `DataLoader` by default constructs an index sampler that yields integral indices. To make it work with a map-style dataset with non-integral indices/keys, a custom sampler must be provided.

data

graph

window_size = 150

parts_distance = 75

standardize_rotation = True

debug_slice = None

absolute = True

relative = False

motion1 = True

motion2 = False

bone = True

bone_angle = False

load_data()

__len__()

__iter__()

__getitem__(*index*)

src.cp_prediction.utils.graph

Classes

Graph	Graph module for modeling skeletons extracted by OpenPose.

Functions

`get_hop_distance(num_nodes, edge[, max_hop])`	
`normalize_digraph(A)`	
`normalize_undigraph(A)`	
`get_adjacency_matrix(edges, num_nodes)`	
`k_adjacency(A, k[, with_self, self_factor])`	
`normalize_adjacency_matrix(A)`	

Module Contents

class `src.cp_prediction.utils.graph.`**`Graph`**(*layout='in-motion'*, *strategy='spatial'*, *max_hop=1*, *dilation=1*, *disentangled_num_scales=7*, *use_mask=True*)

Graph module for modeling skeletons extracted by OpenPose.

Parameters

- **strategy** (*str*) – Partitioning strategy. Must be one of the following: - 'uniform': Uniform Labeling - 'distance': Distance Partitioning - 'spatial': Spatial Configuration

  For more information, see the 'Partition Strategies' section in our paper: https://arxiv.org/abs/1801.07455

- **layout** (*str*) – Skeleton layout. Must be one of the following: - 'in-motion': Consists of 19 joints:

  'head_top', 'nose', 'right_ear', 'left_ear', 'upper_neck', 'right_shoulder', 'right_elbow', 'right_wrist', 'thorax', 'left_shoulder', 'left_elbow', 'left_wrist', 'pelvis', 'right_hip', 'right_knee', 'right_ankle', 'left_hip', 'left_knee', 'left_ankle'.

- **max_hop** (*int*) – Maximum distance between two connected nodes.
- **dilation** (*int*) – Spacing between kernel points.
- **disentangled_num_scales** (*int*) – Number of disentangled adjacency matrices, covering hop distances from 0 to `disentangled_num_scales - 1`.
- **use_mask** (*bool*) – If True, adds a residual mask to the edges of the graph.

max_hop = 1

dilation = 1

layout = 'in-motion'

strategy = 'spatial'

disentangled_num_scales = 7

use_mask = True

num_nodes = None

self_link = None

neighbor_link = None

edge_link = None

center = None

edge = None

hop_dis

__str__()

get_edge(*layout*)

get_adjacency(*strategy*)

get_neck_hip_indexes()

src.cp_prediction.utils.graph.**get_hop_distance**(*num_nodes*, *edge*, *max_hop=1*)

src.cp_prediction.utils.graph.**normalize_digraph**(*A*)

src.cp_prediction.utils.graph.**normalize_undigraph**(*A*)

src.cp_prediction.utils.graph.**get_adjacency_matrix**(*edges*, *num_nodes*)

src.cp_prediction.utils.graph.**k_adjacency**(*A*, *k*, *with_self=False*, *self_factor=1*)

src.cp_prediction.utils.graph.**normalize_adjacency_matrix**(*A*)

src.cp_prediction.utils.predict_helpers

Functions

median_filter(→ numpy.ndarray)	Median filtering on a window of size window_stride.
coords_raw_to_norm(→ numpy.ndarray)	Transform raw coords to normalized coords.
get_rotation_angle(sample_data, graph)	Get rotation angle from sample_data
rotate(→ numpy.ndarray)	Rotate sample_data by the given angle
create_bone_motion_features(→ numpy.ndarray)	Create motion bone features like in Pa-RES-GCN
init_seed(→ None)	Initialize random seed
load_data(→ torch.utils.data.DataLoader)	Load data and initialize Feeders for PyTorch.
load_model(weights_path, args, graph_input[, ...])	Initialize CP PyTorch model and move onto GPU if available.
infer(data, weights_path, args[, xai_technique, debug])	Inference.
get_video_metadata(raw_video_path, str, int)	Gets metadata from a video file.
convert_to_float(frac_str)	
read_csv_to_array(→ List[List[float]])	Helper function that reads the rows of a csv into an array
create_new_csv(csv_name, header)	Creates a new csv file with the given name and header
add_row_csv(row, file_path)	Add a row to the CSV file.
display_body_parts_cam(image, coordinates, cams, groups)	Draw markers on predicted body part locations.
display_segments_cam(image, coordinates[, ...])	Draw segments between body parts according to predicted body part locations.
display_mask(image, coordinates[, image_height, ...])	Draw segments between body parts according to predicted body part locations.
create_synchronized_visualization(video_path, ...)	Creates a synchronized visualization of the video with body keypoints/CAM

Module Contents

src.cp_prediction.utils.predict_helpers.**median_filter**(*resampled_coords*, *window_stride*) → numpy.ndarray

Median filtering on a window of size window_stride. :param resampled_coords: :param window_stride:

Return type
filtered_coords

src.cp_prediction.utils.predict_helpers.**coords_raw_to_norm**(*raw_coords*, *median_pelvis_x*, *median_pelvis_y*, *median_trunk_length*, *num_trunk_lengths=2*) → numpy.ndarray

Transform raw coords to normalized coords. :param raw_coords: :param median_pelvis_x: :param median_pelvis_y: :param median_trunk_length: :param num_trunk_lengths:

Returns:

src.cp_prediction.utils.predict_helpers.**get_rotation_angle**(*sample_data*, *graph*)

Get rotation angle from sample_data We calculate the vertical line (spine) of the body by finding the normal vector from the neck onto the line between the hips This is used to find the degrees to rotate the body into a vertical position :param sample_data: :param graph:

Returns
angle

`src.cp_prediction.utils.predict_helpers.`**`rotate`**(*sample_data*, *angle*) → numpy.ndarray

Rotate sample_data by the given angle :param sample_data: :param angle:

Return type
rotated_sample_data

`src.cp_prediction.utils.predict_helpers.`**`create_bone_motion_features`**(*data*, *conn*, *center_joint=1*) → numpy.ndarray

Create motion bone features like in Pa-RES-GCN https://github.com/yfsong0709/ResGCNv1/tree/master/src/dataset/data_utils.py :param data: :param conn: :param center_joint:

Return type
features

`src.cp_prediction.utils.predict_helpers.`**`init_seed`**(*seed=1*) → None

Initialize random seed :param seed:

Returns
None

`src.cp_prediction.utils.predict_helpers.`**`load_data`**(*data*, *args*, *graph_input*, *debug=False*) → torch.utils.data.DataLoader

Load data and initialize Feeders for PyTorch. :param data: :param args: :param graph_input: :param debug:

Return type
dataloader

`src.cp_prediction.utils.predict_helpers.`**`load_model`**(*weights_path*, *args*, *graph_input*, *graph_main=None*, *output_device=0*)

Initialize CP PyTorch model and move onto GPU if available. :param weights_path: :param args: :param graph_input: :param graph_main: :param output_device:

Returns
model

`src.cp_prediction.utils.predict_helpers.`**`infer`**(*data*, *weights_path*, *args*, *xai_technique='cam'*, *debug=False*)

Inference. :param data: :param weights_path: :param args: :param xai_technique: :param debug:

Returns:

`src.cp_prediction.utils.predict_helpers.`**`get_video_metadata`**(*raw_video_path: str*)

Gets metadata from a video file. :param raw_video_path:

Returns:

`src.cp_prediction.utils.predict_helpers.`**`convert_to_float`**(*frac_str*)

`src.cp_prediction.utils.predict_helpers.`**`read_csv_to_array`**(*file_path: str*) → List[List[float]]

Helper function that reads the rows of a csv into an array :param file_path: path to the csv file

Returns
array of csv data

`src.cp_prediction.utils.predict_helpers.`**`create_new_csv`**(*csv_name: str*, *header: [str]*)

Creates a new csv file with the given name and header :param csv_name: Full path to the CSV file. :param header: List of column names for the CSV.

Returns:

src.cp_prediction.utils.predict_helpers.**add_row_csv**(*row: List[Any]*, *file_path: str*)

Add a row to the CSV file. :param row: list of values to add to the csv file :param file_path:

Returns:

src.cp_prediction.utils.predict_helpers.**display_body_parts_cam**(*image*, *coordinates*, *cams*, *groups*, *image_height=1024*, *image_width=1024*, *marker_radius=5*, *cam_threshold=0.299173*, *binary=False*)

Draw markers on predicted body part locations.

Parameters

- **image** – PIL Image The loaded image the coordinate predictions are inferred for
- **image_draw** – PIL ImageDraw module Module for performing drawing operations
- **coordinates** – Numpy array Predicted body part coordinates in image
- **cams** – Numpy array Predicted body part contribution (CAM) in image
- **groups** – Array Body keypoint indices associated with groups of body keypoints
- **image_height** – int Height of image
- **image_width** – int Width of image
- **marker_radius** – int Radius of marker
- **cam_threshold** – float Threshold value of CAM for body part to contribute towards prediction of CP
- **binary** – float Flag for two-color visualization

Returns

openCV image

src.cp_prediction.utils.predict_helpers.**display_segments_cam**(*image*, *coordinates*, *image_height=1024*, *image_width=1024*, *segment_width=3*)

Draw segments between body parts according to predicted body part locations.

Parameters

- **image** – opencv frame The loaded image the coordinate predictions are inferred for
- **coordinates** – Numpy array Predicted body part coordinates in image
- **image_height** – int Height of image
- **image_width** – int Width of image
- **segment_width** – int Width of association line between markers

Returns

image

src.cp_prediction.utils.predict_helpers.**display_mask**(*image*, *coordinates*, *image_height=1024*, *image_width=1024*, *intensity=10*, *expansion=0.5*)

Draw segments between body parts according to predicted body part locations.

Parameters

- **image** – frame opencv The loaded image the coordinate predictions are inferred for
- **coordinates** – List Predicted body part coordinates in image
- **image_height** – int Height of image
- **image_width** – int Width of image
- **intensity** – int Intensity of blurring
- **expansion** – float Value indicating how much the face mask is expanded beyond face coordinates identified by tracker

Returns
mask

src.cp_prediction.utils.predict_helpers.**create_synchronized_visualization**(*video_path*, *risk_csv_path*, *output_path*, *logger*)

Creates a synchronized visualization of the video with body keypoints/CAM and the CP risk chart below it.

Parameters

- **video_path**
- **risk_csv_path**
- **output_path**
- **logger**

Returns
None

Functions

add_row_csv(row, file_path)	Add a row to the CSV file.
coords_raw_to_norm(→ numpy.ndarray)	Transform raw coords to normalized coords.
create_new_csv(csv_name, header)	Creates a new csv file with the given name and header
display_body_parts_cam(image, coordinates, cams, groups)	Draw markers on predicted body part locations.
display_mask(image, coordinates[, image_height, ...])	Draw segments between body parts according to predicted body part locations.
display_segments_cam(image, coordinates[, ...])	Draw segments between body parts according to predicted body part locations.
get_video_metadata(raw_video_path, str, int)	Gets metadata from a video file.
infer(data, weights_path, args[, xai_technique, debug])	Inference.
median_filter(→ numpy.ndarray)	Median filtering on a window of size window_stride.
read_csv_to_array(→ List[List[float]])	Helper function that reads the rows of a csv into an array

Package Contents

src.cp_prediction.utils.**add_row_csv**(*row: List[Any]*, *file_path: str*)

Add a row to the CSV file. :param row: list of values to add to the csv file :param file_path:

Returns:

`src.cp_prediction.utils.`**`coords_raw_to_norm`**(*raw_coords*, *median_pelvis_x*, *median_pelvis_y*, *median_trunk_length*, *num_trunk_lengths=2*) → numpy.ndarray

Transform raw coords to normalized coords. :param raw_coords: :param median_pelvis_x: :param median_pelvis_y: :param median_trunk_length: :param num_trunk_lengths:

Returns:

`src.cp_prediction.utils.`**`create_new_csv`**(*csv_name: str*, *header: [str]*)

Creates a new csv file with the given name and header :param csv_name: Full path to the CSV file. :param header: List of column names for the CSV.

Returns:

`src.cp_prediction.utils.`**`display_body_parts_cam`**(*image*, *coordinates*, *cams*, *groups*, *image_height=1024*, *image_width=1024*, *marker_radius=5*, *cam_threshold=0.299173*, *binary=False*)

Draw markers on predicted body part locations.

Parameters

- **`image`** – PIL Image The loaded image the coordinate predictions are inferred for
- **`image_draw`** – PIL ImageDraw module Module for performing drawing operations
- **`coordinates`** – Numpy array Predicted body part coordinates in image
- **`cams`** – Numpy array Predicted body part contribution (CAM) in image
- **`groups`** – Array Body keypoint indices associated with groups of body keypoints
- **`image_height`** – int Height of image
- **`image_width`** – int Width of image
- **`marker_radius`** – int Radius of marker
- **`cam_threshold`** – float Threshold value of CAM for body part to contribute towards prediction of CP
- **`binary`** – float Flag for two-color visualization

Returns

openCV image

`src.cp_prediction.utils.`**`display_mask`**(*image*, *coordinates*, *image_height=1024*, *image_width=1024*, *intensity=10*, *expansion=0.5*)

Draw segments between body parts according to predicted body part locations.

Parameters

- **`image`** – frame opencv The loaded image the coordinate predictions are inferred for
- **`coordinates`** – List Predicted body part coordinates in image
- **`image_height`** – int Height of image
- **`image_width`** – int Width of image
- **`intensity`** – int Intensity of blurring
- **`expansion`** – float Value indicating how much the face mask is expanded beyond face coordinates identified by tracker

Returns
mask

`src.cp_prediction.utils.`**`display_segments_cam`**(*image*, *coordinates*, *image_height=1024*, *image_width=1024*, *segment_width=3*)

Draw segments between body parts according to predicted body part locations.

Parameters

- **image** – opencv frame The loaded image the coordinate predictions are inferred for
- **coordinates** – Numpy array Predicted body part coordinates in image
- **image_height** – int Height of image
- **image_width** – int Width of image
- **segment_width** – int Width of association line between markers

Returns
image

`src.cp_prediction.utils.`**`get_video_metadata`**(*raw_video_path: str*)

Gets metadata from a video file. :param raw_video_path:

Returns:

`src.cp_prediction.utils.`**`infer`**(*data*, *weights_path*, *args*, *xai_technique='cam'*, *debug=False*)

Inference. :param data: :param weights_path: :param args: :param xai_technique: :param debug:

Returns:

`src.cp_prediction.utils.`**`median_filter`**(*resampled_coords*, *window_stride*) → numpy.ndarray

Median filtering on a window of size window_stride. :param resampled_coords: :param window_stride:

Return type
filtered_coords

`src.cp_prediction.utils.`**`read_csv_to_array`**(*file_path: str*) → List[List[float]]

Helper function that reads the rows of a csv into an array :param file_path: path to the csv file

Returns
array of csv data

2.1.1.2 src.dashboard

2.1.1.2.1 Submodules

src.dashboard.app

Attributes

static_dir	
app	

Module Contents

src.dashboard.app.**static_dir**

src.dashboard.app.**app**

src.dashboard.datahandler

Attributes

folder_path
video_dest_folder

Functions

extract_unique_ids_from_folder(→ list)	Function to extract unique IDs from filenames
get_tracked_video_file_path(→ str | None)	Function to get tracked video file path
read_kde_profiles([cp_type])	Function to read KDE profiles
get_xai_video_file_path(→ str | None)	Function to get XAI video file path
get_original_video_file_path(→ str | None)	Function to get original video file path
read_cp_file(→ pandas.DataFrame)	Function to read CP file
read_ensamble_preds(patient_id)	Function to read Ensamble predictions

Module Contents

src.dashboard.datahandler.**folder_path**

src.dashboard.datahandler.**video_dest_folder**

src.dashboard.datahandler.**extract_unique_ids_from_folder**(*path: os.PathLike*) → list

Function to extract unique IDs from filenames :param path:

Returns
Ids as list

src.dashboard.datahandler.**get_tracked_video_file_path**(*patient_id: str*) → str | None

Function to get tracked video file path :param patient_id: identifier of patient

Returns
relative path of tracked video file or None

src.dashboard.datahandler.**read_kde_profiles**(*cp_type='yes'*)

Function to read KDE profiles :param cp_type:

Returns:

src.dashboard.datahandler.**get_xai_video_file_path**(*patient_id: str*) → str | None

Function to get XAI video file path :param patient_id: identifier of patient

Returns
relative path of XAI video file or None

`src.dashboard.datahandler.get_original_video_file_path(patient_id: str)` → str | None

Function to get original video file path :param patient_id: identifier of patient

Returns
relative path of original video file or None

`src.dashboard.datahandler.read_cp_file(patient_id: str)` → pandas.DataFrame

Function to read CP file :param patient_id: identifier of patient

Returns
pandas dataframe with cp score

`src.dashboard.datahandler.read_ensamble_preds(patient_id: str)`

Function to read Ensamble predictions :param patient_id: identifier of patient

Returns
pdf dataframe with ensamble predictions or empty dataframe with zeros

src.dashboard.plots

Attributes

blue_theme
custom_blue

Functions

score_plot(→ matplotlib.pyplot)	Plots the CP risk scores over time with confidence intervals and thresholds.
plot_gauge_with_needle(value[, title, min_val, ...])	Plot gauge with needle plot
gradient_plot(→ matplotlib.pyplot)	Gradient plot
kde_plot(df, cp_yes, cp_no, name)	Plots KDE of CP risk scores with shaded area and median line.

Module Contents

`src.dashboard.plots.score_plot(df: pandas.DataFrame, name: str)` → matplotlib.pyplot

Plots the CP risk scores over time with confidence intervals and thresholds. :param df: DataFrame containing the CP risk scores data. :param name: Name of the file being plotted.

Returns
matplotlib.figure.Figure containing the plot.

`src.dashboard.plots.plot_gauge_with_needle(value, title='', min_val=0, max_val=1, cut_off=0.35)`

Plot gauge with needle plot :param value: :param title: :param min_val: :param max_val: :param cut_off:

Returns:

`src.dashboard.plots.gradient_plot(df: pandas.DataFrame)` → matplotlib.pyplot

Gradient plot :param df:

Returns:

`src.dashboard.plots.`**`kde_plot`**(*df*, *cp_yes*, *cp_no*, *name*)

Plots KDE of CP risk scores with shaded area and median line.

Parameters

- **df** (*pd.DataFrame*) – DataFrame containing 'window_cp_risk'.
- **name** (*str*) – Name of the file being plotted.

Returns

KDE plot with median.

Return type

matplotlib.figure.Figure

`src.dashboard.plots.`**`blue_theme`**

`src.dashboard.plots.`**`custom_blue = '#007bc2'`**

src.dashboard.server

Functions

server(input, output, session)	

Module Contents

`src.dashboard.server.`**`server`**(*input*, *output*, *session*)

src.dashboard.shared

Attributes

app_dir
project_root

Module Contents

`src.dashboard.shared.`**`app_dir`**

`src.dashboard.shared.`**`project_root = b'.'`**

src.dashboard.ui

Attributes

app_ui

Module Contents

`src.dashboard.ui.app_ui`

2.1.1.3 src.tracker

2.1.1.3.1 Submodules

src.tracker.tracking

Classes

Tracker	Tracker class based on EfficientposeIII.

Module Contents

class `src.tracker.tracking.`**`Tracker`**(*resolution: int*, *framework: str*, *logger*)

Tracker class based on EfficientposeIII.

Parameters

- **resolution** – tracker resolution
- **framework** – (onnx, tf)
- **logger** – logger

resolution

framework

logger

batch_size = 20

part_size = 10

total_batches = None

num_video_frames = None

model = None

frame_width = None

frame_height = None

fps = None

segments = [(0, 1), (1, 2), (1, 3), (1, 4), (4, 8), (8, 5), (5, 6), (6, 7), (8, 9), (9, 10), (10, 11), (8,...

segment_colors = [(34, 34, 34), (34, 34, 34), (34, 34, 34), (34, 34, 34), (34, 34, 34), (34, 34, 34), (34, 34,...

track(*cap: cv2.VideoCapture*) → numpy.array

Main callable for tracking. Tracking model has to be loaded before calling this function. :param cap:

Returns:

load_video(*file_path: str*) → cv2.VideoCapture | bool

Load the video with opencv and extract the metadata. :param file_path:

Returns:

__preprocess(*batch: numpy.array*, *old=False*) → numpy.array

Preprocess Numpy array according to model preferences.

Parameters

- **old**
- **batch** – ndarray Numpy array of shape (n, h, w, 3)
- **resolution** – int Input height and width of model to utilize

Returns

Preprocessed Numpy array of shape (n, resolution, resolution, 3).

__infer_track(*batch: numpy.array*) → numpy.array

Infer the tracking model based on onnx. :param batch:

Returns: tracked batch

get_track_model(*script_dir: str*) → None

Load the tracking model from onnx.

Parameters

script_dir – path of the script directory to find models.

Returns

None

extract_coordinates(*frame_output: numpy.array*, *real_time=False*)

Extract coordinates from supplied confidence maps.

Parameters

- **frame_output** – ndarray Numpy array of shape (h, w, c)
- **frame_height** – int Height of relevant frame
- **frame_width** – int Width of relevant frame
- **real-time** – boolean Defines if processing is performed in real-time

Returns

List of predicted coordinates for all c body parts in the frame the outputs are computed from.

static **resize**(*source_array: numpy.array*, *target_height: int*, *target_width: int*) → numpy.array

Resizes an image or image-like Numpy array to be no larger than (target_height, target_width) or (target_height, target_width, c).

Parameters

- **source_array** – ndarray Numpy array of shape (h, w) or (h, w, 3)
- **target_height** – int Desired maximum height
- **target_width** – int Desired maximum width

Returns

Resized Numpy array.

`pad`(*source_array: numpy.array*, *target_height: int*, *target_width: int*) → numpy.array

Pads an image or image-like Numpy array with zeros to fit the target-size.

Parameters

- **source_array** – ndarray Numpy array of shape (h, w) or (h, w, 3)
- **target_height** – int Height of padded image
- **target_width** – int Width of padded image

Returns

Zero-padded Numpy array of shape (target_height, target_width) or (target_height, target_width, c).

`extract_rotation`(*video_metadata: dict*) → int

Extract the rotation from the video metadata

Parameters

video_metadata

Returns

rotation

`annotate_video`(*file_path: str*, *coordinates: list*, *output_dir*) → None

Annotates supplied video from predicted coordinates with openCV.

Parameters

- **file_path** – path System path of video to annotate
- **coordinates** – list Predicted body part coordinates for each frame in the video
- **output_dir** – path (optional) Directory to store the output video

Returns

None

`display_body_parts`(*image*, *coordinates*, *image_height=1024*, *image_width=1024*, *marker_radius=5*)

Draw markers on predicted body part locations.

Parameters

- **image** – PIL Image The loaded image the coordinate predictions are inferred for
- **image_draw** – PIL ImageDraw module Module for performing drawing operations
- **coordinates** – List Predicted body part coordinates in image
- **image_height** – int Height of image
- **image_width** – int Width of image
- **marker_radius** – int Radius of marker

Returns

Instance of PIL image with annotated body part predictions.

`display_segments`(*image*, *coordinates*, *image_height=1024*, *image_width=1024*, *segment_width=5*)

Draw segments between body parts according to predicted body part locations.

Parameters

- **image** – PIL Image The loaded image the coordinate predictions are inferred for
- **image_draw** – PIL ImageDraw module Module for performing drawing operations

- **coordinates** – List Predicted body part coordinates in image
- **image_height** – int Height of image
- **image_width** – int Width of image
- **segment_width** – int Width of association line between markers

Returns
Instance of PIL image with annotated body part segments.

static **track_save**(*file_path: str*, *coordinates: list*, *output_dir*)
Saves predicted coordinates as CSV.

Parameters

- **file_path** – path System path of video/image to annotate
- **coordinates** – list Predicted body part coordinates for video/image
- **output_dir** – path (optional) Directory to store the output CSV

PYTHON MODULE INDEX

S