shapenet Documentation

Release 0.1.0

Justus Schock

May 09, 2019

Contents:

`shapenet` provides a PyTorch Implementation of `Super-Realtime Facial Landmark Detection by Deep Regression of Shape Model Parameters`

CHAPTER 1

ShapeNet

The `ShapeNet` package provides a PyTorch implementation of our paper "Super-Realtime Facial Landmark Detection and Shape Fitting by Deep Regression of Shape Model Parameters".

It cointains:

1.1 JIT

The `jit` subpackage contains simplified implementations of some modules (networks and layers) to export them via `torch.jit`

1.1.1 Layers

The following layers have been simplified for `torch.jit`:

1.1.1.1 HomogeneousShapeLayer

class HomogeneousShapeLayer(*shapes*, *n_dims*, *use_cpp=False*)

Bases: `sphinx.ext.autodoc.importer._MockObject`

Module to Perform a Shape Prediction (including a global homogeneous transformation)

forward(*params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e088472e8>*)

Performs the actual prediction

Parameters **params** (`torch.Tensor`) – input parameters

Returns predicted shape

Return type `torch.Tensor`

num_params

Property to access these layer's number of parameters

Returns number of parameters

Return type int

1.1.1.2 HomogeneousTransformationLayer

class HomogeneousTransformationLayer(*n_dims: int*, *use_cpp=False*)
Bases: sphinx.ext.autodoc.importer._MockObject

Wrapper Class to Wrap the Python and C++ API into a combined python API

forward(*shapes: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e088bb6a0>*, *params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e088bb4e0>*)
Selects individual parameters from `params` and forwards them through the actual layer implementation

Parameters

- **shapes** (`torch.Tensor`) – shapes to transform
- **params** (`torch.Tensor`) – parameters specifying the affine transformation

Returns the transformed shapes in cartesian coordinates

Return type `torch.Tensor`

num_params

1.1.1.3 _HomogeneousTransformationLayerPy

class _HomogeneousTransformationLayerPy(*n_dims*)
Bases: sphinx.ext.autodoc.importer._MockObject

Module to perform homogeneous transformations in 2D and 3D (Implemented in Python)

_ensemble_2d_matrix(*rotation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e088bba90>*, *translation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e088bb780>*, *scale_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e088bb710>*)
ensembles the homogeneous transformation matrix for 2D

Parameters

- **rotation_params** (`torch.Tensor`) – parameters specifying the rotation (one parameter)
- **translation_params** (`torch.Tensor`) – parameters specifying the translation (two parameters)
- **scale_params** (`torch.Tensor`) – parameter specifying the global scaling factor (one parameter) (currently only isotropic scaling supported)

Returns 2D transformation matrix

Return type `torch.Tensor`

_ensemble_3d_matrix(*rotation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e088bb6d8>*, *translation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e088bb5f8>*, *scale_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e088bbfd0>*)
ensembles the homogeneous transformation matrix for 3D

Parameters

- **rotation_params** (torch.Tensor) – parameters specifying the rotation (three parameters)
- **translation_params** (torch.Tensor) – parameters specifying the translation (three parameters)
- **scale_params** (torch.Tensor) – parameter specifying the global scaling factor (one parameter) (currently only isotropic scaling supported)

Returns 3D transformation matrix

Return type torch.Tensor

_ensemble_trafo(*rotation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e088bbd30>, translation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e088bbb00>, scale_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e088bbac8>*)

ensembles the transformation matrix in 2D and 3D

Parameters

- **rotation_params** (torch.Tensor) – parameters specifying the rotation (one per DoF)
- **translation_params** (torch.Tensor) – parameters specifying the translation (one per dimension)
- **scale_params** (torch.Tensor) – parameter specifying the global scaling factor (currently only isotropic scaling supported)

Returns transformation matrix

Return type torch.Tensor

forward(*shapes: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e088bb5c0>, rotation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e088bbf28>, translation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e088bbe80>, scale_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e088bbe10>*)

ensembles the homogeneous transformation matrix and applies it to the shape tensor

Parameters

- **shapes** (torch.Tensor) – shapes to transform
- **rotation_params** (torch.Tensor) – parameters specifying the rotation (one per DoF)
- **translation_params** (torch.Tensor) – parameters specifying the translation (one per dimension)
- **scale_params** (torch.Tensor) – parameter specifying the global scaling factor (currently only isotropic scaling supported)

Returns the transformed shapes in cartesian coordinates

Return type torch.Tensor

1.1.1.4 ShapeLayer

class ShapeLayer(*shapes*, *use_cpp=False*)

Bases: sphinx.ext.autodoc.importer._MockObject

forward(*shape_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e087475f8>*)

num_params

1.1.2 AbstractShapeNetwork

class AbstractShapeNetwork(***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Abstract JIT Network

static norm_type_to_class(*norm_type*)

1.1.3 AbstractFeatureExtractor

class AbstractFeatureExtractor(*in_channels*, *out_params*, *norm_class*, *p_dropout=0*)

Bases: sphinx.ext.autodoc.importer._MockObject

Abstract Feature Extractor Class all further feature extractors should be derived from

static _build_model(*in_channels*, *out_features*, *norm_class*, *p_dropout*)

Build the actual model structure

Parameters

- **in_channels** (*int*) – number of input channels
- **out_features** (*int*) – number of outputs
- **norm_class** (*Any*) – class implementing a normalization
- **p_dropout** (*float*) – dropout probability

Returns ensembled model

Return type torch.jit.ScriptModule

forward(*input_batch*)

Feed batch through network

Parameters **input_batch** (torch.Tensor) – batch to feed through network

Returns extracted features

Return type torch.Tensor

1.1.4 Conv2dRelu

class Conv2dRelu(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Block holding one Conv2d and one ReLU layer

forward(*input_batch*)

Forward batch though layers

Parameters **input_batch** (class:*torch.Tensor*) – input batch

Returns **class** – result

Return type *torch.Tensor*

1.1.5 Img224x224Kernel7x7SeparatedDims

class Img224x224Kernel7x7SeparatedDims(*in_channels*, *out_params*, *norm_class*, *p_dropout=0*)

Bases: *shapenet.jit.abstract_network.AbstractFeatureExtractor*

static _build_model(*in_channels*, *out_params*, *norm_class*, *p_dropout*)

Build the actual model structure

Parameters

- **in_channels** (*int*) – number of input channels
- **out_params** (*int*) – number of outputs
- **norm_class** (*Any*) – class implementing a normalization
- **p_dropout** (*float*) – dropout probability

Returns ensembled model

Return type `torch.jit.ScriptModule`

forward(*input_batch*)

Feed batch through network

Parameters **input_batch** (`torch.Tensor`) – batch to feed through network

Returns extracted features

Return type `torch.Tensor`

1.1.6 ShapeNetwork

class ShapeNetwork(*layer_cls*, *layer_kwargs*, *in_channels=1*, *norm_type='instance'*, *img_size=224*, *tiny=False*, *feature_extractor=None*, ***kwargs*)

Bases: *shapenet.jit.abstract_network.AbstractShapeNetwork*

Network to Predict a single shape

forward(*input_images*)

Forward input batch through network and shape layer

Parameters **input_images** (`torch.Tensor`) – input batch

Returns predicted shapes

Return type `torch.Tensor`

model

static norm_type_to_class(*norm_type*)

1.2 Layer

The `layer` subpackage contains implementations of all provided layers

1.2.1 ShapeLayer

class ShapeLayer(*shapes*, *use_cpp=False*)
Bases: sphinx.ext.autodoc.importer._MockObject

Wrapper to compine Python and C++ Implementation under Single API

forward(*shape_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740208>*)
Forwards parameters to Python or C++ Implementation

Parameters **shape_params** (torch.Tensor) – parameters for shape ensembling

Returns Ensempled Shape

Return type torch.Tensor

num_params
Property to access these layer's parameters

Returns number of parameters

Return type int

1.2.2 _ShapeLayerPy

class _ShapeLayerPy(*shapes*)
Bases: sphinx.ext.autodoc.importer._MockObject

Python Implementation of Shape Layer

forward(*shape_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e087402b0>*)
Ensemble shape from parameters

Parameters **shape_params** (torch.Tensor) – shape parameters

Returns ensembled shape

Return type torch.Tensor

num_params
Property to access these layer's parameters

Returns number of parameters

Return type int

1.2.3 _ShapeLayerCpp

class _ShapeLayerCpp(*shapes*, *verbose=True*)
Bases: sphinx.ext.autodoc.importer._MockObject

C++ Implementation of Shape Layer

forward(*shape_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740358>*)
Ensemble shape from parameters

Parameters **shape_params** (torch.Tensor) – shape parameters

Returns ensembled shape

Return type torch.Tensor

num_params
Property to access these layer's parameters

Returns number of parameters

Return type int

1.2.4 HomogeneousTransformationLayer

class HomogeneousTransformationLayer(*n_dims: int*, *use_cpp=False*)
Bases: sphinx.ext.autodoc.importer._MockObject

Wrapper Class to Wrap the Python and C++ API into a combined python API

forward(*shapes: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740748>*, *params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740780>*)
Actual prediction

Parameters

- **shapes** (torch.Tensor) – shapes before applied global transformation
- **params** (torch.Tensor) – parameters specifying the global transformation

Returns Transformed shapes

Return type torch.Tensor

num_params

1.2.5 _HomogeneousTransformationLayerPy

class _HomogeneousTransformationLayerPy(*n_dims*)
Bases: sphinx.ext.autodoc.importer._MockObject

Module to perform homogeneous transformations in 2D and 3D (Implemented in Python)

_ensemble_2d_matrix(*rotation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740b00>*, *translation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740b38>*, *scale_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740b70>*)
ensembles the homogeneous transformation matrix for 2D

Parameters

- **rotation_params** (torch.Tensor) – parameters specifying the rotation (one parameter)
- **translation_params** (torch.Tensor) – parameters specifying the translation (two parameters)
- **scale_params** (torch.Tensor) – parameter specifying the global scaling factor (one parameter) (currently only isotropic scaling supported)

Returns 2D transformation matrix

Return type torch.Tensor

_ensemble_3d_matrix(*rotation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740ba8>, translation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740be0>, scale_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740c18>*)

ensembles the homogeneous transformation matrix for 3D

Parameters

- **rotation_params** (torch.Tensor) – parameters specifying the rotation (three parameters)
- **translation_params** (torch.Tensor) – parameters specifying the translation (three parameters)
- **scale_params** (torch.Tensor) – parameter specifying the global scaling factor (one parameter) (currently only isotropic scaling supported)

Returns 3D transformation matrix

Return type torch.Tensor

_ensemble_trafo(*rotation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740a58>, translation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740a90>, scale_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740ac8>*)

ensembles the transformation matrix in 2D and 3D

Parameters

- **rotation_params** (torch.Tensor) – parameters specifying the rotation (one per DoF)
- **translation_params** (torch.Tensor) – parameters specifying the translation (one per dimension)
- **scale_params** (torch.Tensor) – parameter specifying the global scaling factor (currently only isotropic scaling supported)

Returns transformation matrix

Return type torch.Tensor

forward(*shapes: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740978>, rotation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e087409b0>, translation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e087409e8>, scale_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740a20>*)

ensembles the homogeneous transformation matrix and applies it to the shape tensor

Parameters

- **shapes** (torch.Tensor) – shapes to transform
- **rotation_params** (torch.Tensor) – parameters specifying the rotation (one per DoF)
- **translation_params** (torch.Tensor) – parameters specifying the translation (one per dimension)
- **scale_params** (torch.Tensor) – parameter specifying the global scaling factor (currently only isotropic scaling supported)

Returns the transformed shapes in cartesian coordinates

Return type `torch.Tensor`

1.2.6 _HomogeneousTransformationLayerCpp

class **_HomogeneousTransformationLayerCpp**(*n_dims*, *verbose=True*)
Bases: `sphinx.ext.autodoc.importer._MockObject`

Module to perform homogeneous transformations in 2D and 3D (Implemented in C++)

forward(*shapes: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740828>*, *rotation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740860>*, *translation_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e08740898>*, *scale_params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e087408d0>*)
ensembles the homogeneous transformation matrix and applies it to the shape tensor

Parameters

- **shapes** (`torch.Tensor`) – shapes to transform
- **rotation_params** (`torch.Tensor`) – parameters specifying the rotation (one per DoF)
- **translation_params** (`torch.Tensor`) – parameters specifying the translation (one per dimension)
- **scale_params** (`torch.Tensor`) – parameter specifying the global scaling factor (currently only isotropic scaling supported)

Returns the transformed shapes in cartesian coordinates

Return type `torch.Tensor`

1.2.7 HomogeneousShapeLayer

class **HomogeneousShapeLayer**(*shapes*, *n_dims*, *use_cpp=False*)
Bases: `sphinx.ext.autodoc.importer._MockObject`

Module to Perform a Shape Prediction (including a global homogeneous transformation)

forward(*params: <sphinx.ext.autodoc.importer._MockObject object at 0x7f0e087b7a58>*)
Performs the actual prediction

Parameters **params** (`torch.Tensor`) – input parameters

Returns predicted shape

Return type `torch.Tensor`

num_params
Property to access these layer's number of parameters

Returns number of parameters

Return type int

1.3 Networks

The `networks` subpackage contains implementations of all provided networks

1.3.1 AbstractShapeNetwork

class AbstractShapeNetwork(***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

Abstract base Class to provide a convenient norm_class_mapping

static norm_type_to_class(*norm_type*)
helper function to map a string to an actual normalization class

Parameters **norm_type** (*str*) – string specifying the normalization class

Returns Normalization Class (subclass of torch.nn.Module)

Return type type

1.3.2 AbstractFeatureExtractor

class AbstractFeatureExtractor(*in_channels*, *out_params*, *norm_class*, *p_dropout=0*)
Bases: sphinx.ext.autodoc.importer._MockObject

Abstract Feature Extractor Class all further feature extracotrs should be derived from

static _build_model(*in_channels*, *out_features*, *norm_class*, *p_dropout*)
Build the actual model structure

Parameters

- **in_channels** (*int*) – number of input channels
- **out_features** (*int*) – number of outputs
- **norm_class** (*Any*) – class implementing a normalization
- **p_dropout** (*float*) – dropout probability

Returns ensembled model

Return type torch.nn.Module

forward(*input_batch*)
Feed batch through network

Parameters **input_batch** (torch.Tensor) – batch to feed through network

Returns exracted features

Return type torch.Tensor

1.3.3 Conv2dRelu

class Conv2dRelu(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

Block holding one Conv2d and one ReLU layer

forward(*input_batch*)
Forward batch though layers

Parameters **input_batch** (torch.Tensor) – input batch

Returns result

Return type torch.Tensor

1.3.4 Img224x224Kernel7x7SeparatedDims

class Img224x224Kernel7x7SeparatedDims(*in_channels*, *out_params*, *norm_class*, *p_dropout=0*)

Bases: *shapenet.networks.abstract_network.AbstractFeatureExtractor*

static _build_model(*in_channels*, *out_params*, *norm_class*, *p_dropout*)

Build the actual model structure

Parameters

- **in_channels** (*int*) – number of input channels
- **out_params** (*int*) – number of outputs
- **norm_class** (*Any*) – class implementing a normalization
- **p_dropout** (*float*) – dropout probability

Returns ensembled model

Return type `torch.nn.Module`

forward(*input_batch*)

Feed batch through network

Parameters **input_batch** (`torch.Tensor`) – batch to feed through network

Returns exracted features

Return type `torch.Tensor`

1.3.5 ShapeNetwork

class ShapeNetwork(*layer_cls*, *layer_kwargs*, *in_channels=1*, *norm_type='instance'*, *img_size=224*, *feature_extractor=None*, ***kwargs*)

Bases: *shapenet.networks.abstract_network.AbstractShapeNetwork*

Network to Predict a single shape

static closure(*model*, *data_dict: dict*, *optimizers: dict*, *criterions={}*, *metrics={}*, *fold=0*, ***kwargs*)

closure method to do a single backpropagation step

Parameters

- **model** (*ShapeNetwork*) – trainable model
- **data_dict** (*dict*) – dictionary containing the data
- **optimizers** (*dict*) – dictionary of optimizers to optimize model's parameters
- **criterions** (*dict*) – dict holding the criterions to calculate errors (gradients from different criterions will be accumulated)
- **metrics** (*dict*) – dict holding the metrics to calculate
- **fold** (*int*) – Current Fold in Crossvalidation (default: 0)
- ****kwargs** – additional keyword arguments

Returns

- *dict* – Metric values (with same keys as input dict metrics)
- *dict* – Loss values (with same keys as input dict criterions)

- *list* – Arbitrary number of predictions as `torch.Tensor`

Raises `AssertionError` – if optimizers or criterions are empty or the optimizers are not specified

forward(*input_images*)
Forward input batch through network and shape layer

Parameters **input_images** (`torch.Tensor`) – input batch

Returns predicted shapes

Return type `torch.Tensor`

model

static norm_type_to_class(*norm_type*)
helper function to map a string to an actual normalization class

Parameters **norm_type** (*str*) – string specifying the normalization class

Returns Normalization Class (subclass of `torch.nn.Module`)

Return type type

1.3.6 CustomGroupNorm

class CustomGroupNorm(*n_features*, *n_groups=2*)
Bases: `sphinx.ext.autodoc.importer._MockObject`

Custom Group Norm which adds n_groups=2 as default parameter

forward(*x*)
Forward batch through network

Parameters **x** (`torch.Tensor`) – batch to forward

Returns normalized results

Return type `torch.Tensor`

1.4 Scripts

The `scripts` subpackage contains basic scripts for:

1.4.1 prepare_all_data

prepare_all_data()
Prepares all Datasets from commandline arguments

See also:

_prepare_ibug_dset(), *_prepare_cats()*

1.4.2 prepare_helen_dset

prepare_helen_dset()
Prepares the HELEN Dataset from commandline arguments

See also:

_prepare_ibug_dset() iBug Datasets HELEN Dataset

1.4.3 prepare_lfpw_dset

prepare_lfpw_dset()
Prepares the LFPW Dataset from commandline arguments

See also:

_prepare_ibug_dset() iBug Datasets LFPW Dataset

1.4.4 prepare_cat_dset

prepare_cat_dset()
Prepares the Cat Dataset from commandline arguments

See also:

_prepare_cats() Cat Dataset

1.4.5 _prepare_ibug_dset

_prepare_ibug_dset(*zip_file*, *dset_name*, *out_dir*, *remove_zip=False*, *normalize_pca_rot=True*)
Prepares an ibug dataset (from a given zipfile)

Parameters

- **zip_file** (*str*) – the zip archive containing the data
- **dset_name** (*str*) – the dataset's name
- **out_dir** (*str*) – the output directory
- **remove_zip** (*bool, optional*) – whether or not to remove the ZIP file after finishing the preparation
- **normalize_pca_rot** (*bool, optional*) – whether or not to normalize the data's rotation during PCA

See also:

iBug Datasets

1.4.6 _prepare_cats

_prepare_cats(*out_dir*, *remove_zip=False*, *normalize_pca_rot=False*, ***split_options*)
Prepares the cat dataset (with multiprocessing)

Parameters

- **out_dir** (*str*) – the output directory

- **remove_zip** (*bool, optional*) – whether or not to remove the ZIP file after finishing the preparation
- **normalize_pca_rot** (*bool, optional*) – whether or not to normalize the data's rotation during PCA

See also:

Cat Dataset

1.4.7 _process_single_cat_file

_process_single_cat_file(*file*, *target_dir*)
Processes a single file of the cat dataset

Parameters

- **file** (*str*) – the file to process
- **target_dir** (*str*) – the target directory

1.4.8 _make_pca

_make_pca(*data_dir*, *out_file*, *normalize_rot=False*, *rotation_idxs=()*)
Creates a PCA from data in a given directory

Parameters

- **data_dir** (*str*) – directory containing the image and landmark files
- **out_file** (*str*) – file the pca will be saved to
- **normalize_rot** (*bool, optional*) – whether or not to normalize the data's rotation
- **rotation_idxs** (*tuple, optional*) – indices for rotation normalization, msut be specified if `normalize_rot=True`

1.4.9 train_shapenet

train_shapenet()
Trains a single shapenet with config file from comandline arguments

See also:

`delira.training.PyTorchNetworkTrainer`

1.4.10 predict

predict()
Predicts file directory with network specified by files to output path

1.4.11 create_jit_net_from_config_and_weight

create_jit_net_from_config_and_weight(*config_dict*, *weight_file*)
Creates a JIT Network from config dict and weight file

Parameters

- **config_dict** (*dict*) – dict containing network configuration
- **weight_file** (*str*) – path to file containing weights

Returns jitted network

Return type torch.jit.ScriptModule

1.4.12 main

main()

1.5 Utils

The utils subpackage contains general utilities, whcih did not fit into any other subpackage

1.5.1 Config

class Config(*verbose=False*)

Bases: object

Implements parser for configuration files

1.5.2 now

now()

Return current time as YYYY-MM-DD_HH-MM-SS

Returns current time

Return type string

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index

Symbols

A

C

F

H

I

M

N

P

S

T