mindpose

mindpose contributors

Apr 28, 2023

APIS

CHAPTER

ONE

MINDPOSE.DATA

mindpose.data.**create_dataset**(*image_root*, *annotation_file=None*, *dataset_format='coco_topdown'*, *is_train=True*, *device_num=None*, *rank_id=None*, *num_workers=1*, *config=None*, ***kwargs*)

Create dataset for training or evaluation.

Parameters

- **image_root** (str) – The path of the directory storing images
- **annotation_file** (Optional[str]) – The path of the annotation file. Default: None
- **dataset_format** (str) – The dataset format. Different format yield different final output. Default: *coco_topdown*
- **is_train** (bool) – Wether this dataset is used for training/testing: Default: True
- **device_num** (Optional[int]) – Number of devices (e.g. GPU). Default: None
- **rank_id** (Optional[int]) – Current process's rank id. Default: None
- **num_workers** (int) – Number of workers in reading data. Default: 1
- **config** (Optional[Dict[str, Any]]) – Dataset-specific configuration
- **use_gt_bbox_for_val** – Use GT bbox instead of detection result during evaluation. Default: False
- **detection_file** – Path of the detection result. Default: None

Return type
GeneratorDataset

Returns
Dataset for training or evaluation

mindpose.data.**create_pipeline**(*dataset*, *transforms*, *method='topdown'*, *batch_size=1*, *is_train=True*, *normalize=True*, *normalize_mean=[0.485, 0.456, 0.406]*, *normalize_std=[0.229, 0.224, 0.255]*, *hwc_to_chw=True*, *num_workers=1*, *config=None*)

Create dataset tranform pipeline. The returned datatset is transformed sequentially based on the given list of transforms.

Parameters

- **dataset** (Dataset) – Dataset to perform transformations
- **transforms** (List[Union[str, Dict[str, Any]]]) – List of transformations
- **method** (str) – The method to use. Default: "topdown"

- **batch_size** (int) – Batch size. Default: 1
- **is_train** (bool) – Whether the transformation is for training/testing. Default: True
- **normalize** (bool) – Perform normalization. Default: True
- **normalize_mean** (List[float]) – Mean of the normalization: Default: [0.485, 0.456, 0.406]
- **normalize_std** (List[float]) – Std of the normalization: Default: [0.229, 0.224, 0.255]
- **hwc_to_chw** (bool) – Wwap height x width x channel to channel x height x width. Default: True
- **num_workers** (int) – Number of workers in processing data. Default: 1
- **config** (Optional[Dict[str, Any]]) – Transform-specific configuration

Return type
Dataset

Returns
The transformed dataset

CHAPTER

TWO

MINDPOSE.DATA.DATASET

class mindpose.data.dataset.**BottomUpDataset**(*image_root*, *annotation_file=None*, *is_train=False*, *num_joints=17*, *config=None*)

Bases: object

Create an iterator for ButtomUp dataset, return the tuple with (image, boxes, keypoints, target, mask, tag_ind) for training; return the tuple with (image, mask, center, scale, image_file, image_shape) for evaluation.

Parameters

- **image_root** (str) – The path of the directory storing images
- **annotation_file** (Optional[str]) – The path of the annotation file. Default: None
- **is_train** (bool) – Wether this dataset is used for training/testing. Default: False
- **num_joints** (int) – Number of joints in the dataset. Default: 17
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None

Items in iterator:

image: Encoded data for image file
keypoints: Keypoints in (x, y, visibility)
mask: Mask of the image showing the valid annotations
target: A placeholder for later pipline using
tag_ind: A placeholder of later pipline using
image_file: Path of the image file
boxes: Bounding box coordinate (x0, y0), (x1, y1)

Note: This is an abstract class, child class must implement *load_dataset_cfg* and *load_dataset* method.

load_dataset()

Loading the dataset, where the returned record should contain the following key

Keys:

image_file: Path of the image file.
keypoints (For training only): Keypoints in (x, y, visibility).
boxes (For training only): Bounding box coordinate (x0, y0), (x1, y1).
mask_info (For training only): The mask info of crowed or zero keypoints instances.

Return type
List[Dict[str, Any]]

Returns
A list of records of groundtruth or predictions

`load_dataset_cfg()`

Loading the dataset config, where the returned config must be a dictionary which stores the configuration of the dataset, such as the image_size, etc.

Return type
Dict[str, Any]

Returns
Dataset configurations

`class mindpose.data.dataset.COCOBottomUpDataset(image_root, annotation_file=None, is_train=False, num_joints=17, config=None)`

Bases: *BottomUpDataset*

Create an iterator for ButtomUp dataset, return the tuple with (image, boxes, keypoints, mask, target, tag_ind) for training; return the tuple with (image, mask, center, scale, image_file, image_shape) for evaluation.

Parameters

- **image_root** (str) – The path of the directory storing images
- **annotation_file** (Optional[str]) – The path of the annotation file. Default: None
- **is_train** (bool) – Wether this dataset is used for training/testing. Default: False
- **num_joints** (int) – Number of joints in the dataset. Default: 17
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None

Items in iterator:

image: Encoded data for image file
keypoints: Keypoints in (x, y, visibility)
mask: Mask of the image showing the valid annotations
target: A placeholder for later pipline using
keypoints_coordinate: A placeholder of later pipline using
image_file: Path of the image file
boxes: Bounding box coordinate (x0, y0), (x1, y1)

`load_dataset()`

Loading the dataset, where the returned record should contain the following key

Keys:

image_file: Path of the image file.
keypoints (For training only): Keypoints in (x, y, visibility).
boxes (For training only): Bounding box coordinate (x0, y0), (x1, y1).
mask_info (For training only): The mask info of crowed or zero keypoints instances.

Return type
List[Dict[str, Any]]

Returns
A list of records of groundtruth or predictions

`load_dataset_cfg()`

Loading the dataset config, where the returned config must be a dictionary which stores the configuration of the dataset, such as the image_size, etc.

Return type
`Dict[str, Any]`

Returns
Dataset configurations

class mindpose.data.dataset.**COCOTopDownDataset**(*image_root*, *annotation_file=None*, *is_train=False*, *num_joints=17*, *use_gt_bbox_for_val=False*, *detection_file=None*, *config=None*)

Bases: *TopDownDataset*

Create an iterator for TopDown dataset based COCO annotation format. return the tuple with (image, center, scale, keypoints, rotation, target, target_weight) for training; return the tuple with (image, center, scale, rotation, image_file, boxes, bbox_ids, bbox_score) for evaluation.

Parameters

- **image_root** (`str`) – The path of the directory storing images
- **annotation_file** (`Optional[str]`) – The path of the annotation file. Default: None
- **is_train** (`bool`) – Wether this dataset is used for training/testing. Default: False
- **num_joints** (`int`) – Number of joints in the dataset. Default: 17
- **use_gt_bbox_for_val** (`bool`) – Use GT bbox instead of detection result during evaluation. Default: False
- **detection_file** (`Optional[str]`) – Path of the detection result. Defaul: None
- **config** (`Optional[Dict[str, Any]]`) – Method-specific configuration. Default: None

Item in iterator:

image: Encoded data for image file
center: A placeholder for later pipline using
scale: A placeholder of later pipline using
keypoints: Keypoints in (x, y, visibility)
rotation: Rotatated degree
target: A placeholder for later pipline using
target_weight: A placeholder of later pipline using
image_file: Path of the image file
boxes: Bounding box coordinate (x, y, w, h)
bbox_id: Bounding box id for each single image
bbox_score: Bounding box score, 1 for ground truth

`load_dataset()`

Loading the dataset, where the returned record should contain the following key

Keys:

image_file: Path of the image file
bbox: Bounding box coordinate (x, y, w, h)
keypoints: Keypoints in [K, 3(x, y, visibility)]
bbox_score: Bounding box score, 1 for ground truth
bbox_id: Bounding box id for each single image

Return type
List[Dict[str, Any]]

Returns
A list of records of groundtruth or predictions

`load_dataset_cfg()`

Loading the dataset config, where the returned config must be a dictionary which stores the configuration of the dataset, such as the image_size, etc.

Return type
Dict[str, Any]

Returns
Dataset configurations

`class mindpose.data.dataset.ImageFolderBottomUpDataset(image_root, annotation_file=None, is_train=False, num_joints=17, config=None)`

Bases: *BottomUpDataset*

Create an iterator for ButtomUp dataset based on image folder. It is usually used for demo usage. Return the tuple with (image, mask, center, scale, image_file, image_shape)

Parameters

- **image_root** (str) – The path of the directory storing images
- **annotation_file** (Optional[str]) – The path of the annotation file. Default: None
- **is_train** (bool) – Wether this dataset is used for training/testing. Default: False
- **num_joints** (int) – Number of joints in the dataset. Default: 17
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None

`load_dataset()`

Loading the dataset, where the returned record should contain the following key

Keys:

image_file: Path of the image file.

Return type
List[Dict[str, Any]]

Returns
A list of records of groundtruth or predictions

`load_dataset_cfg()`

Loading the dataset config, where the returned config must be a dictionary which stores the configuration of the dataset, such as the image_size, etc.

Return type
Dict[str, Any]

Returns
Dataset configurations

class mindpose.data.dataset.**TopDownDataset**(*image_root*, *annotation_file=None*, *is_train=False*, *num_joints=17*, *use_gt_bbox_for_val=False*, *detection_file=None*, *config=None*)

Bases: object

Create an iterator for TopDown dataset, return the tuple with (image, center, scale, keypoints, rotation, target, target_weight) for training; return the tuple with (image, center, scale, rotation, image_file, boxes, bbox_ids, bbox_score) for evaluation.

Parameters

- **image_root** (str) – The path of the directory storing images
- **annotation_file** (Optional[str]) – The path of the annotation file. Default: None
- **is_train** (bool) – Wether this dataset is used for training/testing. Default: False
- **num_joints** (int) – Number of joints in the dataset. Default: 17
- **use_gt_bbox_for_val** (bool) – Use GT bbox instead of detection result during evaluation. Default: False
- **detection_file** (Optional[str]) – Path of the detection result. Default: None
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None

Item in iterator:

image: Encoded data for image file
center: A placeholder for later pipline using
scale: A placeholder of later pipline using
keypoints: Keypoints in [K, 3(x, y, visibility)]
rotation: Rotatated degree
target: A placeholder for later pipline using
target_weight: A placeholder of later pipline using
image_file: Path of the image file
bbox: Bounding box coordinate (x, y, w, h)
bbox_id: Bounding box id for each single image
bbox_score: Bounding box score, 1 for ground truth

Note: This is an abstract class, child class must implement *load_dataset_cfg* and *load_dataset* method.

---

load_dataset()

Loading the dataset, where the returned record should contain the following key

Keys:

image_file: Path of the image file
bbox: Bounding box coordinate (x, y, w, h)

keypoints: Keypoints in [K, 3(x, y, visibility)]
bbox_score: Bounding box score, 1 for ground truth
bbox_id: Bounding box id for each single image

Return type
`List[Dict[str, Any]]`

Returns
A list of records of groundtruth or predictions

`load_dataset_cfg()`

Loading the dataset config, where the returned config must be a dictionary which stores the configuration of the dataset, such as the image_size, etc.

Return type
`Dict[str, Any]`

Returns
Dataset configurations

CHAPTER

THREE

MINDPOSE.DATA.TRANSFORM

class mindpose.data.transform.**BottomUpGenerateTarget**(*is_train=True*, *config=None*, *sigma=2.0*, *max_num=30*)

Bases: *BottomUpTransform*

Generate heatmap with the keypoint coordinatess with multiple scales.

Parameters

- **is_train** (bool) – Whether the transformation is for training/testing. Default: True
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None
- **sigma** (float) – The sigmal size of gausian distribution. Default: 2.0
- **max_num** (int) – Maximum number of instances within the image. Default: 30

Inputs:

data: Data tuples need to be transformed

Outputs:

result: Transformed data tuples

transform(*state*)

Transform the state into the transformed state. state is a dictionay storing the informaton of the image and labels, the returned states is the updated dictionary storing the updated image and labels.

Parameters
state (Dict[str, Any]) – Stored information of image and labels

Return type
Dict[str, Any]

Returns
Updated inforamtion of image and labels based on the transformation

---

Note:

Required *keys* for transform: keypoints
Returned *keys* after transform: target, tag_ind

---

class mindpose.data.transform.**BottomUpHorizontalRandomFlip**(*is_train=True*, *config=None*, *flip_prob=0.5*)

Bases: *BottomUpTransform*

Perform randomly horizontal flip in bottomup approach.

Parameters

- **is_train** (bool) – Whether the transformation is for training/testing. Default: True
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None
- **flip_prob** (float) – Probability of performing a horizontal flip. Default: 0.5

transform(*state*)

Transform the state into the transformed state. state is a dictionay storing the informaton of the image and labels, the returned states is the updated dictionary storing the updated image and labels.

Parameters
state (Dict[str, Any]) – Stored information of image and labels

Return type
Dict[str, Any]

Returns
Updated inforamtion of image and labels based on the transformation

---

Note:

Required *keys* for transform: image, mask, keypoints

Returned *keys* after transform: image, mask, keypoints

---

class mindpose.data.transform.**BottomUpPad**(*is_train=True*, *config=None*)

Bases: *BottomUpTransform*

Padding the image to the *max_image_size*.

Parameters

- **is_train** (bool) – Whether the transformation is for training/testing. Default: True
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None

transform(*state*)

Transform the state into the transformed state. state is a dictionay storing the informaton of the image and labels, the returned states is the updated dictionary storing the updated image and labels.

Parameters
state (Dict[str, Any]) – Stored information of image and labels

Return type
Dict[str, Any]

Returns
Updated inforamtion of image and labels based on the transformation

---

Note:

Required *keys* for transform: image

Returned *keys* after transform: image, mask

---

class mindpose.data.transform.**BottomUpRandomAffine**(*is_train=True*, *config=None*, *rot_factor=30.0*, *scale_factor=(0.75, 1.5)*, *scale_type='short'*, *trans_factor=40.0*)

Bases: *BottomUpTransform*

Random affine transform the image. The mask and keypoints will be rescaled to the heatmap sizes after the transformation.

Parameters

- **is_train** (bool) – Whether the transformation is for training/testing. Default: True
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None
- **rot_factor** (float) – Randomly rotated in [-rotation_factor, rotation_factor]. Default: 30.
- **scale_factor** (Tuple[float, float]) – Randomly Randomly scaled in [scale_factor[0], scale_factor[1]]. Default: (0.75, 1.5)
- **scale_type** (str) – Scaling with the long / short length of the image. Default: short
- **trans_factor** (float) – Translation factor. Default: 40.

transform(*state*)

Transform the state into the transformed state. state is a dictionay storing the informaton of the image and labels, the returned states is the updated dictionary storing the updated image and labels.

Parameters
state (Dict[str, Any]) – Stored information of image and labels

Return type
Dict[str, Any]

Returns
Updated inforamtion of image and labels based on the transformation

---

Note:

Required *keys* for transform: image, mask, keypoints

Returned *keys* after transform: image, mask, keypoints

---

class mindpose.data.transform.**BottomUpRescale**(*is_train=True*, *config=None*)

Bases: *BottomUpTransform*

Rescaling the image to the *max_image_size* without change the aspect ratio.

Parameters

- **is_train** (bool) – Whether the transformation is for training/testing. Default: True
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None

transform(*state*)

Transform the state into the transformed state. state is a dictionay storing the informaton of the image and labels, the returned states is the updated dictionary storing the updated image and labels.

Parameters
state (Dict[str, Any]) – Stored information of image and labels

Return type
Dict[str, Any]

Returns
Updated inforamtion of image and labels based on the transformation

---

Note:

Required *keys* for transform: image
Returned *keys* after transform: image, center, scale, image_shape

---

class mindpose.data.transform.**BottomUpResize**(*is_train=True*, *config=None*, *size=512*, *base_length=64*)

Bases: *BottomUpTransform*

Resize the image without change the aspect ratio. The length of the short side of the image will be equal to the input *size*.

Parameters

- **is_train** (bool) – Whether the transformation is for training/testing. Default: True
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None
- **size** (int) – The target size of the short side of the image. Default: 512
- **base_length** (int) – The minimum size the image. Default: 64

transform(*state*)

Transform the state into the transformed state. state is a dictionay storing the informaton of the image and labels, the returned states is the updated dictionary storing the updated image and labels.

Parameters
state (Dict[str, Any]) – Stored information of image and labels

Return type
Dict[str, Any]

Returns
Updated inforamtion of image and labels based on the transformation

---

Note:

Required *keys* for transform: image
Returned *keys* after transform: image, mask, center, scale, image_shape

---

class mindpose.data.transform.**BottomUpTransform**(*is_train=True*, *config=None*)

Bases: *Transform*

Transform the input data into the output data based on bottom-up approach.

Parameters

- **is_train** (bool) – Whether the transformation is for training/testing. Default: True
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None

Inputs:
data: Data tuples need to be transformed

Outputs:
result: Transformed data tuples

Note: This is an abstract class, child class must implement *transform* method.

`load_transform_cfg()`

Loading the transform config, where the returned the config must be a dictionary which stores the configuration of this transformation, such as the transformed image size, etc.

Return type
`Dict[str, Any]`

Returns
Transform configuration

`setup_required_field()`

Get the required columns names used for this transformation. The columns names will be later used with Minspore Dataset *map* func.

Return type
`List[str]`

Returns
The column names

`class mindpose.data.transform.TopDownAffine(is_train=True, config=None, use_udp=False)`

Bases: *TopDownTransform*

Affine transform the image, and the transform image will contain single instance only.

Parameters

- **is_train** (`bool`) – Whether the transformation is for training/testing. Default: True
- **config** (`Optional[Dict[str, Any]]`) – Method-specific configuration. Default: None
- **use_udp** (`bool`) – Use Unbiased Data Processing (UDP) affine transform. Default: False

Inputs:

data: Data tuples need to be transformed

Outputs:

result: Transformed data tuples

`transform(state)`

Transform the state into the transformed state. state is a dictionay storing the informaton of the image and labels, the returned states is the updated dictionary storing the updated image and labels.

Parameters
state (`Dict[str, Any]`) – Stored information of image and labels

Return type
`Dict[str, Any]`

Returns
Updated inforamtion of image and labels based on the transformation

Note:

Required *keys* for transform: image, center, scale, rotation, keypoints (optional)
Returned *keys* after transform: image, keypoints (optional)

---

class mindpose.data.transform.**TopDownBoxToCenterScale**(*is_train=True*, *config=None*)

Bases: *TopDownTransform*

Convert the box coordinate to center and scale. If *is_train* is True, the center will be randomly shifted by a small amount.

Parameters

- **is_train** (bool) – Whether the transformation is for training/testing. Default: True
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None

Inputs:

data: Data tuples need to be transformed

Outputs:

result: Transformed data tuples

transform(*state*)

Transform the state into the transformed state. state is a dictionay storing the informaton of the image and labels, the returned states is the updated dictionary storing the updated image and labels.

Parameters
state (Dict[str, Any]) – Stored information of image and labels

Return type
Dict[str, Any]

Returns
Updated inforamtion of image and labels based on the transformation

---

Note:

Required *keys* for transform: boxes
Returned *keys* after transform: center, scale

---

class mindpose.data.transform.**TopDownGenerateTarget**(*is_train=True*, *config=None*, *sigma=2.0*, *use_different_joint_weights=False*, *use_udp=False*)

Bases: *TopDownTransform*

Generate heatmap from the coordinates.

Parameters

- **is_train** (bool) – Whether the transformation is for training/testing. Default: True
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None
- **sigma** (float) – The sigmal size of gausian distribution. Default: 2.0
- **use_different_joint_weights** (bool) – Use extra joint weight in target weight calculation. Default: False

- **use_udp** (bool) – Use Unbiased Data Processing (UDP) encoding. Default: False

Inputs:

data: Data tuples need to be transformed

Outputs:

result: Transformed data tuples

transform(*state*)

Transform the state into the transformed state. state is a dictionay storing the informaton of the image and labels, the returned states is the updated dictionary storing the updated image and labels.

Parameters
state (Dict[str, Any]) – Stored information of image and labels

Return type
Dict[str, Any]

Returns
Updated inforamtion of image and labels based on the transformation

---

Note:

Required *keys* for transform: keypoints
Returned *keys* after transform: target, target_weight

---

class mindpose.data.transform.**TopDownHalfBodyTransform**(*is_train=True*, *config=None*, *num_joints_half_body=8*, *prob_half_body=0.3*, *scale_padding=1.5*)

Bases: *TopDownTransform*

Perform half-body transform. Keep only the upper body or the lower body at random.

Parameters

- **is_train** (bool) – Whether the transformation is for training/testing. Default: True
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None
- **num_joints_half_body** (int) – Threshold number of performing half-body transform. Default: 8
- **prob_half_body** (float) – Probability of performing half-body transform. Default: 0.3
- **scale_padding** (float) – Extra scale padding multiplier in generating the cropped images. Default: 1.5

Inputs:

data: Data tuples need to be transformed

Outputs:

result: Transformed data tuples

transform(*state*)

Transform the state into the transformed state. state is a dictionay storing the informaton of the image and labels, the returned states is the updated dictionary storing the updated image and labels.

Parameters
state (Dict[str, Any]) – Stored information of image and labels

Return type
Dict[str, Any]

Returns
Updated inforamtion of image and labels based on the transformation

---

Note:

Required *keys* for transform: keypoints
Returned *keys* after transform: center, scale

---

class mindpose.data.transform.**TopDownHorizontalRandomFlip**(*is_train=True*, *config=None*, *flip_prob=0.5*)

Bases: *TopDownTransform*

Perform randomly horizontal flip in topdown approach.

Parameters

- **is_train** (bool) – Whether the transformation is for training/testing. Default: True
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None
- **flip_prob** (float) – Probability of performing a horizontal flip. Default: 0.5

Inputs:

data: Data tuples need to be transformed

Outputs:

result: Transformed data tuples

transform(*state*)

Transform the state into the transformed state. state is a dictionay storing the informaton of the image and labels, the returned states is the updated dictionary storing the updated image and labels.

Parameters
state (Dict[str, Any]) – Stored information of image and labels

Return type
Dict[str, Any]

Returns
Updated inforamtion of image and labels based on the transformation

---

Note:

Required *keys* for transform: image, keypoints, center
Returned *keys* after transform: image, keypoints, center

class mindpose.data.transform.**TopDownRandomScaleRotation**(*is_train=True*, *config=None*, *rot_factor=40.0*, *scale_factor=0.5*, *rot_prob=0.6*)

Bases: *TopDownTransform*

Perform random scaling and rotation.

Parameters

- **is_train** (bool) – Whether the transformation is for training/testing. Default: True
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None
- **rot_factor** (float) – Std of rotation degree. Default: 40.
- **scale_factor** (float) – Std of scaling value. Default: 0.5
- **rot_prob** (float) – Probability of performing rotation. Default: 0.6

Inputs:

data: Data tuples need to be transformed

Outputs:

result: Transformed data tuples

transform(*state*)

Transform the state into the transformed state. state is a dictionay storing the informaton of the image and labels, the returned states is the updated dictionary storing the updated image and labels.

Parameters
state (Dict[str, Any]) – Stored information of image and labels

Return type
Dict[str, Any]

Returns
Updated inforamtion of image and labels based on the transformation

Note:

Required *keys* for transform: scale
Returned *keys* after transform: scale, rotation

class mindpose.data.transform.**TopDownTransform**(*is_train=True*, *config=None*)

Bases: *Transform*

Transform the input data into the output data based on top-down approach.

Parameters

- **is_train** (bool) – Whether the transformation is for training/testing. Default: True
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None

Inputs:
data: Data tuples need to be transformed

Outputs:
result: Transformed data tuples

Note: This is an abstract class, child class must implement *transform* method.

`load_transform_cfg()`

Loading the transform config, where the returned the config must be a dictionary which stores the configuration of this transformation, such as the transformed image size, etc.

Return type
`Dict[str, Any]`

Returns
Transform configuration

`setup_required_field()`

Get the required columns names used for this transformation. The columns names will be later used with Minspore Dataset *map* func.

Return type
`List[str]`

Returns
The column names

`class mindpose.data.transform.Transform(is_train=True, config=None)`

Bases: `object`

Transform the input data into the output data.

Parameters

- **is_train** (`bool`) – Whether the transformation is for training/testing. Default: True
- **config** (`Optional[Dict[str, Any]]`) – Method-specific configuration. Default: None

Inputs:
data: Data tuples need to be transformed

Outputs:
result: Transformed data tuples

Note: This is an abstract class, child class must implement *load_transform_cfg*, *transform* and *setup_required_field* method.

`load_transform_cfg()`

Loading the transform config, where the returned the config must be a dictionary which stores the configuration of this transformation, such as the transformed image size, etc.

Return type
`Dict[str, Any]`

Returns
Transform configuration

`setup_required_field()`

Get the required columns names used for this transformation. The columns names will be later used with Minspore Dataset *map* func.

Return type
`List[str]`

Returns
The column names

`transform(state)`

Transform the state into the transformed state. state is a dictionay storing the informaton of the image and labels, the returned states is the updated dictionary storing the updated image and labels.

Parameters
state (`Dict[str, Any]`) – Stored information of image and labels

Return type
`Dict[str, Any]`

Returns
Updated inforamtion of image and labels based on the transformation

CHAPTER

FOUR

MINDPOSE.MODELS

class mindpose.models.**EvalNet**(*net*, *decoder*, *output_raw=True*)

Bases: Cell

Create network for forward propagate and decoding only.

Parameters

- **net** (*Net*) – Network used for foward and backward propagate
- **decoder** (*Decoder*) – Decoder
- **output_raw** (bool) – Return extra net's ouput. Default: True

Inputs:

inputs: List of tensors

Outputs

result: Decoded result
raw_result (optional): Raw result if output_raw is true

class mindpose.models.**Net**(*backbone*, *head*, *neck=None*)

Bases: Cell

Create network for foward and backward propagate.

Parameters

- **backbone** (*Backbone*) – Model backbone
- **head** (*Head*) – Model head
- **neck** (Optional[*Neck*]) – Model neck. Default: None

Inputs:

x: Tensor

Outputs:

result: Tensor

class mindpose.models.**NetWithLoss**(*net*, *loss*, *has_extra_inputs=False*)

Bases: Cell

Create network with loss.

Parameters

- **net** (*Net*) – Network used for foward and backward propagate
- **loss** (*Loss*) – Loss cell
- **has_extra_inputs** (bool) – Has Extra inputs in the loss calculation. Default: False

Inputs:

data: Tensor feed into network
label: Tensor of label
extra_inputs: List of extra tensors used in loss calculation

Outputs:

loss: Loss value

mindpose.models.**create_backbone**(*name*, *pretrained=False*, *ckpt_url=''*, *in_channels=3*, ***kwargs*)

Create model backbone.

Parameters

- **name** (str) – Name of the backbone
- **pretrained** (bool) – Whether the backbone is pretrained. Default: False
- **ckpt_url** (str) – Url of the pretrain check point. Default: None
- **in_channels** (int) – Number of channels in the input data. Default: 3
- ****kwargs** (Any) – Arguments which feed into the backbone

Return type
Backbone

Returns
Model backbone

mindpose.models.**create_decoder**(*name*, ***kwargs*)

Create model decoder.

Parameters

- **name** (str) – Name of the decoder
- ****kwargs** (Any) – Arguments which feed into the decoder

Return type
Decoder

Returns
Model decoder

mindpose.models.**create_eval_network**(*net*, *decoder*, *output_raw=True*)

Create network for inferencing or evaluation.

Parameters

- **net** (*Net*) – Network used for foward and backward propagate
- **decoder** (*Decoder*) – Decoder
- **output_raw** (bool) – Return extra net's ouput. Default: True

Return type
EvalNet

Returns
Network for inferencing or evaluation

mindpose.models.**create_head**(*name*, *in_channels*, *num_joints=17*, ***kwargs*)

Create model head.

Parameters

- **name** (str) – Name of the head
- **in_channels** – Number of channels in the input tensor
- **num_joints** (int) – Number of joints. Default: 17
- ****kwargs** (Any) – Arguments which feed into the head

Return type
Head

Returns
Model head

mindpose.models.**create_loss**(*name*, ***kwargs*)

Create model loss.

Parameters

- **name** (str) – Name of the loss
- ****kwargs** (Any) – Arguments which feed into the loss

Return type
Loss

Returns
Loss

mindpose.models.**create_neck**(*name*, *in_channels*, *out_channels*, ***kwargs*)

Create model neck.

Parameters

- **name** (str) – Name of the neck
- **in_channels** – Number of channels in the input tensor
- **out_channels** – Number of channels in the output tensor
- ****kwargs** (Any) – Arguments which feed into the neck

Return type
Neck

Returns
Model neck

mindpose.models.**create_network**(*backbone_name*, *head_name*, *neck_name=''*, *backbone_pretrained=False*, *backbone_ckpt_url=''*, *in_channels=3*, *neck_out_channels=256*, *num_joints=17*, *backbone_args=None*, *neck_args=None*, *head_args=None*)

Create network for training.

Parameters

- **backbone_name** (str) – Backbone name
- **head_name** (str) – Head name
- **neck_name** (str) – Neck name. Default: ""
- **backbone_pretrained** (bool) – Whether backbone is pretrained. Default: False
- **backbone_ckpt_url** (str) – Url of backbone's pretrained checkpoint. Default: ""
- **in_channels** (int) – Number of channels in the input data. Default: 3
- **neck_out_channels** (int) – Number of output channels in the neck. Default: 256
- **num_joints** (int) – Number of joints in the output. Default: 17
- **backbone_args** (Optional[Dict[str, Any]]) – Arguments for backbone. Defauult: None
- **neck_args** (Optional[Dict[str, Any]]) – Arguments for neck. Default: None
- **head_args** (Optional[Dict[str, Any]]) – Arguments for head: Default: None

Return type
Net

Returns
Network for training

mindpose.models.**create_network_with_loss**(*net*, *loss*, *has_extra_inputs=False*)

Create network with loss for training.

Parameters

- **net** (*Net*) – Network used for foward and backward propagate
- **loss** (*Loss*) – Loss cell
- **has_extra_inputs** (bool) – Has Extra inputs in the loss calculation. Default: False

Return type
NetWithLoss

Returns
Network with loss for training

CHAPTER

FIVE

MINDPOSE.MODELS.BACKBONES

class mindpose.models.backbones.**Backbone**(*auto_prefix=True*, *flags=None*)

Bases: Cell

Abstract class for all backbones.

Note: Child class must implement *foward_feature* and *out_channels* method.

forward_feature(*x*)

Perform the feature extraction.

Parameters
x (Tensor) – Tensor

Return type
Tensor

Returns
Extracted feature

property out_channels: Union[List[int], int]

Get number of output channels.

Returns
Output channels.

class mindpose.models.backbones.**HRNet**(*stage_cfg*, *in_channels=3*)

Bases: *Backbone*

HRNet Backbone, based on "Deep High-Resolution Representation Learning for Human Pose Estimation".

Parameters

- **stage_cfg** (Dict[str, Dict[str, int]]) – Configuration of the extra blocks. It accepts a dictionay storing the detail config of each block. which include *num_modules*, *num_branches*, *block*, *num_blocks*, *num_channels* and *multiscale_output*. For detail example, please check the implementation of *hrnet_w32* and *hrnet_w48*
- **in_channels** (int) – Number the channels of the input. Default: 3

Inputs:

x: Input Tensor

Outputs:

feature: Feature Tensor

forward_feature(*x*)

Perform the feature extraction.

Parameters
x (Tensor) – Tensor

Return type
Tensor

Returns
Extracted feature

property out_channels: int

Get number of output channels.

Returns
Output channels.

class mindpose.models.backbones.**ResNet**(*block*, *layers*, *in_channels=3*, *groups=1*, *base_width=64*, *norm=None*)

Bases: *Backbone*

ResNet model class, based on "Deep Residual Learning for Image Recognition".

Parameters

- **block** (Type[Union[BasicBlock, Bottleneck]]) – Block of resnet
- **layers** (List[int]) – Number of layers of each stage
- **in_channels** (int) – Number the channels of the input. Default: 3
- **groups** (int) – Number of groups for group conv in blocks. Default: 1
- **base_width** (int) – Base width of pre group hidden channel in blocks. Default: 64
- **norm** (Optional[Cell]) – Normalization layer in blocks. Default: None

Inputs:

x: Input Tensor

Outputs:

feature: Feature Tensor

forward_feature(*x*)

Perform the feature extraction.

Parameters
x (Tensor) – Tensor

Return type
Tensor

Returns
Extracted feature

property out_channels: int

Get number of output channels.

Returns
Output channels.

mindpose.models.backbones.**hrnet_w32**(*pretrained=False*, *ckpt_url=''*, *in_channels=3*)

Get HRNet with width=32 model.

Parameters

- **pretrained** (bool) – Whether the model is pretrained. Default: False
- **ckpt_url** (str) – Url of the pretrained weight. Default: “”
- **in_channels** (int) – Number of input channels. Default: 3

Return type
HRNet

Returns
HRNet model

mindpose.models.backbones.**hrnet_w48**(*pretrained=False*, *ckpt_url=''*, *in_channels=3*)

Get HRNet with width=48 model.

Parameters

- **pretrained** (bool) – Whether the model is pretrained. Default: False
- **ckpt_url** (str) – Url of the pretrained weight. Default: “”
- **in_channels** (int) – Number of input channels. Default: 3

Return type
HRNet

Returns
HRNet model

mindpose.models.backbones.**resnet101**(*pretrained=False*, *ckpt_url=''*, *in_channels=3*, ***kwargs*)

Get 101 layers ResNet model.

Parameters

- **pretrained** (bool) – Whether the model is pretrained. Default: False
- **ckpt_url** (str) – Url of the pretrained weight. Default: “”
- **in_channels** (int) – Number of input channels. Default: 3
- **kwargs** – Arguments which feed into Resnet class

Return type
ResNet

Returns
Resnet model

mindpose.models.backbones.**resnet152**(*pretrained=False*, *ckpt_url=''*, *in_channels=3*, ***kwargs*)

Get 152 layers ResNet model.

Parameters

- **pretrained** (bool) – Whether the model is pretrained. Default: False

- **ckpt_url** (str) – Url of the pretrained weight. Default: “”
- **in_channels** (int) – Number of input channels. Default: 3
- **kwargs** – Arguments which feed into Resnet class

Return type
ResNet

Returns
Resnet model

mindpose.models.backbones.**resnet50**(*pretrained=False*, *ckpt_url=''*, *in_channels=3*, ***kwargs*)

Get 50 layers ResNet model.

Parameters

- **pretrained** (bool) – Whether the model is pretrained. Default: False
- **ckpt_url** (str) – Url of the pretrained weight. Default: “”
- **in_channels** (int) – Number of input channels. Default: 3
- **kwargs** – Arguments which feed into Resnet class

Return type
ResNet

Returns
Resnet model

CHAPTER

SIX

MINDPOSE.MODELS.NECKS

class mindpose.models.necks.**Neck**(*auto_prefix=True*, *flags=None*)

Bases: Cell

Abstract class for all necks. Child class must implement *construct* and *out_channels* method.

property out_channels: Union[List[int], int]

Get number of output channels.

Returns

Output channels.

CHAPTER

SEVEN

MINDPOSE.MODELS.HEADS

class mindpose.models.heads.**HRNetHead**(*in_channels=32*, *num_joints=17*, *final_conv_kernel_size=1*)

Bases: *Head*

HRNet Head, based on "Deep High-Resolution Representation Learning for Human Pose Estimation". It is a 1x1 convoultion layer using the feature ouptput.

Parameters

- **in_channels** (int) – Number the channels of the input. Default: 32.
- **num_joints** (int) – Number of joints in the final output. Default: 17
- **final_conv_kernel_size** (int) – The kernel size in the final convolution layer. Default: 1

Inputs:

x: Input Tensor

Outputs:

result: Result Tensor

class mindpose.models.heads.**Head**(*auto_prefix=True*, *flags=None*)

Bases: Cell

Abstract class for all heads.

class mindpose.models.heads.**HigherHRNetHead**(*in_channels=32*, *num_joints=17*, *with_ae_loss=[True, False]*, *tag_per_joint=True*, *final_conv_kernel_size=1*, *num_deconv_layers=1*, *num_deconv_filters=[32]*, *num_deconv_kernels=[4]*, *cat_outputs=[True]*, *num_basic_blocks=4*)

Bases: *Head*

HigherHRNet Head, based on "HigherHRNet: Scale-Aware Representation Learning for Bottom-Up Human Pose Estimation".

Parameters

- **in_channels** (int) – Number the channels of the input. Default: 32.
- **num_joints** (int) – Number of joints in the final output. Default: 17
- **with_ae_loss** (List[bool]) – Output the associated embedding for each resolution. Default: [True, False]

- **tag_per_joint** (bool) – Whether each of the joint has its own coordinate encoding. Default: True
- **final_conv_kernel_size** (int) – The kernel size in the final convolution layer. Default: 1
- **num_deconv_layers** (int) – Number of deconvolution layers. Default: 1
- **num_deconv_filters** (List[int]) – Number of filters in each deconvolution layer. Default: [32]
- **num_deconv_kernels** (List[int]) – Kernel size in each deconvolution layer. Default: [4]
- **cat_outputs** (List[bool]) – Whether to concate the feature before deconvolution layer at each resoluton. Default: [True]
- **num_basic_blocks** (int) – Number of basic blocks after deconvolution. Default: 4

Inputs:

x: Input Tensor

Outputs:

result: Tuples of Tensor at different resolution

class mindpose.models.heads.**SimpleBaselineHead**(*num_deconv_layers=3*, *num_deconv_filters=[256, 256, 256]*, *num_deconv_kernels=[4, 4, 4]*, *in_channels=2048*, *num_joints=17*, *final_conv_kernel_size=1*)

Bases: *Head*

SimpleBaseline Head, based on "Simple Baselines for Human Pose Estimation and Tracking". It contains few number of deconvolution layers following by a 1x1 convolution layer.

Parameters

- **num_deconv_layers** (int) – Number of deconvolution layers. Default: 3
- **num_deconv_filters** (List[int]) – Number of filters in each deconvolution layer. Default: [256, 256, 256]
- **num_deconv_kernels** (List[int]) – Kernel size in each deconvolution layer. Default: [4, 4, 4]
- **in_channels** (int) – number the channels of the input. Default: 2048.
- **num_joints** (int) – Number of joints in the final output. Default: 17
- **final_conv_kernel_size** (int) – The kernel size in the final convolution layer. Default: 1

Inputs:

x: Input Tensor

Outputs:

result: Result Tensor

CHAPTER

EIGHT

MINDPOSE.MODELS.DECODERS

class mindpose.models.decoders.**BottomUpHeatMapAEDecoder**(*num_joints=17*, *num_stages=2*, *with_ae_loss=[True, False]*, *use_nms=False*, *nms_kernel=5*, *max_num=30*, *tag_per_joint=True*, *shift_coordinate=False*)

Bases: *Decoder*

Decode the heatmaps with associativa embedding into coordinates

Parameters

- **num_joints** (int) – Number of joints. Default: 17
- **num_stages** (int) – Number of resolution in the heatmap outputs. If it is larger than one, then heatmap aggregation is performed. Default: 2
- **with_ae_loss** (List[bool]) – Output the associated embedding for each resolution. Default: [True, False]
- **use_nms** (bool) – Apply NMS for the heatmap output. Default: False
- **nms_kernel** (int) – NMS kerrnel size. Default: 5
- **max_num** (int) – Maximum number (K) of instances in the image. Default: 30
- **tag_per_joint** (bool) – Whether each of the joint has its own coordinate encoding. Default: True
- **shift_coordinate** (bool) – Perform a +-0.25 pixel coordinate shift based on heatmap value. Default: False

Inputs:

model_output: Model output. It is a list of Tensors with the length equal to the num_stages.
mask: Heatmap mask of the valid region.

Outputs:

val_k, tag_k, ind_k: Tuples contains the maximum value of the heatmap for each joint with the corresponding tag value and location.

class mindpose.models.decoders.**Decoder**(*auto_prefix=True*, *flags=None*)

Bases: Cell

Abstract class for all decoders.

class mindpose.models.decoders.**TopDownHeatMapDecoder**(*pixel_std=200.0*, *to_original=True*, *shift_coordinate=False*, *use_udp=False*, *dark_udp_refine=False*, *kernel_size=11*)

Bases: *Decoder*

Decode the heatmaps into coordinates with bounding boxes.

Parameters

- **pixel_std** (float) – The scaling factor using in decoding. Default: 200.
- **to_original** (bool) – Convert the coordinate into the raw image. Default: True
- **shift_coordinate** (bool) – Perform a +-0.25 pixel coordinate shift based on heatmap value. Default: False
- **use_udp** (bool) – Use Unbiased Data Processing (UDP) decoding. Default: False
- **dark_udp_refine** (bool) – Use post-refinement based on DARK / UDP. It cannot be use with *shift_coordinate* in the same time. Default: False
- **kernel_size** (int) – Gaussian kernel size for UDP post-refinement, it should match the heatmap gaussian simg in training. K=17 for sigma=3 and K=11 for sigma=2. Default: 11

Inputs:

heatmap: The ordinary output based on heatmap-based model, in shape [N, C, H, W]
center: Center of the bounding box (x, y) in raw image, in shape [N, C, 2]
scale: Scale of the bounding box with respect to the raw image, in shape [N, C, 2]
score: Score of the bounding box, in shape [N, C, 1]

Outputs:

coordinate: The coordindate of C joints, in shape [N, C, 3(x_coord, y_coord, score)]
boxes: The coor bounding boxes, in shape [N, 6(center_x, center_y, scale_x, scale_y, area, bounding_box_score)]

CHAPTER

NINE

MINDPOSE.MODELS.LOSS

class mindpose.models.loss.**AELoss**(*tag_per_joint=True*, *reduction='mean'*)

Bases: *Loss*

Associative embedding loss. Or called *Grouping loss*. Based on "End-to-End Learning for Joint Detection and Grouping".

Parameters

- **tag_per_joint** (bool) – Whether each of the joint has its own coordinate encoding. Default: True
- **reduction** (Optional[str]) – Type of the reduction to be applied to loss. The optional value are "mean", "sum" and "none". Default: "mean"

Inputs:

pred: Predicted tags. In shape [N, K, H, W] if tag_per_joint is True; in shape [N, H, W] otherwise. Where K stands for the number of joints.

target: Ground truth of tag mask. In shape [N, M, K, 2] if tag_per_joint is True; in shape [N, M, 2] otherwise. Where M stands for number of instances.

Outputs:

loss: Loss tensor contains the push loss and the pull loss.

class mindpose.models.loss.**AEMultiLoss**(*num_joints=17*, *num_stages=2*, *stage_sizes=[(128, 128), (256, 256)]*, *mse_loss_factor=[1.0, 1.0]*, *ae_loss_factor=[0.001, 0.001]*, *with_mse_loss=[True, True]*, *with_ae_loss=[True, False]*, *tag_per_joint=True*)

Bases: *Loss*

Combined loss of MSE and AE for multi levels of resolutions

Parameters

- **num_joints** (int) – Number of joints. Default: 17
- **num_stages** (int) – Number of resolution levels. Default: 2
- **stage_sizes** (List[Tuple[int, int]]) – The sizes in each stage. Default: [(128, 128), (256, 256)]
- **mse_loss_factor** (List[float]) – Weighting for MSE loss at each level. Default: [1.0, 1.0]

- **ae_loss_factor** (List[float]) – Weighting for Associative embedding loss at each level. Default: [0.001, 0.001]
- **with_mse_loss** (List[bool]) – Whether to calculate MSE loss at each level. Default: [True, False]
- **with_ae_loss** (List[bool]) – Whether to calculate AE loss at each level. Default: [True, False]
- **tag_per_joint** (bool) – Whether each of the joint has its own coordinate encoding. Default: True

Inputs:

pred: List of prediction result at each resolution level. In shape [N, aK, H, W]. Where K stands for the number of joints. a=2 if the correspoinding with_ae_loss is True

target: Ground truth of heatmap. In shape [N, S, K, H, W]. Where S stands for the number of resolution levels.

mask: Ground truth of the heatmap mask. In shape [N, S, H, W].

tag_ind: Ground truth of tag position. In shape [N, S, M, K, 2]. Where M stands for number of instances.

Outputs:

loss: Single Loss value

class mindpose.models.loss.**JointsMSELoss**(*use_target_weight=False*, *reduction='mean'*)

Bases: *Loss*

Joint Mean square error loss. It is the MSE loss of heatmaps with extra weight for different channel.

Parameters

- **use_target_weight** (bool) – Use extra weight in loss calculation. Default: False
- **reduction** (Optional[str]) – Type of the reduction to be applied to loss. The optional value are “mean”, “sum” and “none”. Default: “mean”

Inputs:

pred: Predictions, in shape [N, K, H, W]

target: Ground truth, in shape [N, K, H, W]

target_weight: Loss weight, in shape [N, K]

Outputs:

loss: Loss value

class mindpose.models.loss.**JointsMSELossWithMask**(*reduction='mean'*)

Bases: *Loss*

Joint Mean square error loss with mask. Mask-out position will not contribute to the loss.

Parameters

reduction (Optional[str]) – Type of the reduction to be applied to loss. The optional value are “mean”, “sum” and “none”. Default: “mean”

Inputs:

pred: Predictions, in shape [N, K, H, W]
target: Ground truth, in shape [N, K, H, W]
mask: Ground truth Mask, in shape [N, H, W]

Outputs:

loss: Loss value

`class` mindpose.models.loss.**Loss**(*reduction='mean'*)

Bases: LossBase

Abstract class for all losses.

CHAPTER

TEN

MINDPOSE.ENGINE

mindpose.engine.**create_evaluator**(*annotation_file*, *name='topdown'*, *metric='AP'*, *config=None*, *dataset_config=None*, ***kwargs*)

Create evaluator engine. Evaluator engine is used to provide metric performance based on the provided prediction result.

Parameters

- **annotation_file** (str) – Path of the annotation file. It only supports COCO-format now.
- **name** (str) – Name of the evaluation method. Default: "topdown"
- **metric** (Union[str, List[str]]) – Supported metrics. Default: "AP"
- **config** (Optional[Dict[str, Any]]) – Evaluaton config. Default: None
- **dataset_config** (Optional[Dict[str, Any]]) – Dataset config. Since the evaluation method sometimes relies on the dataset info. Default: None
- ****kwargs** (Any) – Arguments which feed into the evaluator

Return type
Evaluator

Returns
Evaluator engine for evaluation

mindpose.engine.**create_inferencer**(*net*, *name='topdown_heatmap'*, *config=None*, *dataset_config=None*, ***kwargs*)

Create inference engine. Inference engine is used to perform model inference on the entire dataset based on the given method name.

Parameters

- **net** (*EvalNet*) – Network for evaluation
- **name** (str) – Name of the inference method. Default: "topdown_heatmap"
- **config** (Optional[Dict[str, Any]]) – Inference config. Default: None
- **dataset_config** (Optional[Dict[str, Any]]) – Dataset config. Since the inference method sometimes relies on the dataset info. Default: None
- ****kwargs** (Any) – Arguments which feed into the inferencer

Return type
Inferencer

Returns
Inference engine for inferencing

CHAPTER

ELEVEN

MINDPOSE.ENGINE.INFERENCER

class mindpose.engine.inferencer.**BottomUpHeatMapAEInferencer**(*net*, *config=None*, *progress_bar=False*, *decoder=None*)

Bases: *Inferencer*

Create an inference engine for bottom-up heatmap with associative embedding based method. It runs the inference on the entire dataset and outputs a list of records.

Parameters

- **net** (*EvalNet*) – Network for evaluation
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None
- **progress_bar** (bool) – Display the progress bar during inferencing. Default: False
- **decoder** (Optional[*BottomUpHeatMapAEDecoder*]) – Decoder cell. It is used for hflip TTA. Default: None

Inputs:

dataset: Dataset

Outputs:

records: List of inference records.

infer(*dataset*)

Running the inference on the dataset. And return a list of records. Normally, in order to be compatible with the evaluator engine, each record should contains the following keys:

Keys:

pred: The predicted coordindate, in shape [M, 3(x_coord, y_coord, score)]

box: The coor bounding boxes, each record contains (center_x, center_y, scale_x, scale_y, area, bounding box score)

image_path: The path of the image

bbox_id: Bounding box ID

Parameters

dataset (Dataset) – Dataset for inferencing

Return type
List[Dict[str, Any]]

Returns
List of inference results

`load_inference_cfg()`

Loading the inference config, where the returned config must be a dictionary which stores the configuration of the engine, such as the using TTA, etc.

Return type
Dict[str, Any]

Returns
Inference configurations

`class mindpose.engine.inferencer.Inferencer(net, config=None)`

Bases: object

Create an inference engine. It runs the inference on the entire dataset and outputs a list of records.

Parameters

- **net** (*EvalNet*) – Network for inference
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration for inference. Default: None

Inputs:

dataset: Dataset for inferencing

Outputs:

records: List of inference records

Note: This is an abstract class, child class must implement *load_inference_cfg* method.

`infer(dataset)`

Running the inference on the dataset. And return a list of records. Normally, in order to be compatible with the evaluator engine, each record should contains the following keys:

Keys:

pred: The predicted coordindate, in shape [C, 3(x_coord, y_coord, score)]

box: The coor bounding boxes, each record contains (center_x, center_y, scale_x, scale_y, area, bounding box score)

image_path: The path of the image

bbox_id: Bounding box ID

Parameters
dataset (Dataset) – Dataset for inferencing

Return type
List[Dict[str, Any]]

Returns
List of inference results

`load_inference_cfg()`

Loading the inference config, where the returned config must be a dictionary which stores the configuration of the engine, such as the using TTA, etc.

Return type
`Dict[str, Any]`

Returns
Inference configurations

`class mindpose.engine.inferencer.TopDownHeatMapInferencer(net, config=None, progress_bar=False, decoder=None)`

Bases: *Inferencer*

Create an inference engine for Topdown heatmap based method. It runs the inference on the entire dataset and outputs a list of records.

Parameters

- **net** (*EvalNet*) – Network for evaluation
- **config** (`Optional[Dict[str, Any]]`) – Method-specific configuration. Default: None
- **progress_bar** (`bool`) – Display the progress bar during inferencing. Default: False
- **decoder** (`Optional[TopDownHeatMapDecoder]`) – Decoder cell. It is used for hflip TTA. Default: None

Inputs:

dataset: Dataset

Outputs:

records: List of inference records.

`infer(dataset)`

Running the inference on the dataset. And return a list of records. Normally, in order to be compatible with the evaluator engine, each record should contains the following keys:

Keys:

pred: The predicted coordindate, in shape [M, 3(x_coord, y_coord, score)]

box: The coor bounding boxes, each record contains (center_x, center_y, scale_x, scale_y, area, bounding box score)

image_path: The path of the image

bbox_id: Bounding box ID

Parameters
dataset (`Dataset`) – Dataset for inferencing

Return type
`List[Dict[str, Any]]`

Returns
List of inference results

`load_inference_cfg()`

Loading the inference config, where the returned config must be a dictionary which stores the configuration of the engine, such as the using TTA, etc.

Return type
`Dict[str, Any]`

Returns
Inference configurations

CHAPTER

TWELVE

MINDPOSE.ENGINE.EVALUATOR

class mindpose.engine.evaluator.**BottomUpEvaluator**(*annotation_file*, *metric='AP'*, *num_joints=17*, *config=None*, *remove_result_file=True*, *result_path='./result_keypoints.json'*)

Bases: *Evaluator*

Create an evaluator based on BottomUp method. It performs the model evaluation based on the inference result (a list of records), and outputs with the metirc result.

Parameters

- **annotation_file** (str) – Path of the annotation file. It only supports COCO-format.
- **metric** (Union[str, List[str]]) – Supported metrics. Default: "AP"
- **num_joints** (int) – Number of joints. Default: 17
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None
- **remove_result_file** (bool) – Remove the cached result file after evaluation. Default: True
- **result_path** (str) – Path of the result file. Default: "./result_keypoints.json"

Inputs:

inference_result: Inference result from inference engine

Outputs:

evaluation_result: Evaluation result based on the metric

eval(*inference_result*)

Running the evaluation base on the inference result. Output the metric result.

Parameters
inference_result (Dict[str, Any]) – List of inference records

Return type
Dict[str, Any]

Returns
metric result. Such as AP.5, etc.

load_evaluation_cfg()

Loading the evaluation config, where the returned config must be a dictionary which stores the configuration of the engine, such as the using soft-nms, etc.

Return type
Dict[str, Any]

Returns
Evaluation configurations

class mindpose.engine.evaluator.**Evaluator**(*annotation_file*, *metric='AP'*, *num_joints=17*, *config=None*)

Bases: object

Create an evaluator engine. It performs the model evaluation based on the inference result (a list of records), and outputs with the metirc result.

Parameters

- **annotation_file** (str) – Path of the annotation file. It only supports COCO-format now.
- **metric** (Union[str, List[str]]) – Supported metrics. Default: "AP"
- **num_joints** (int) – Number of joints. Default: 17
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None

Inputs:
inference_result: Inference result from inference engine

Outputs:
evaluation_result: Evaluation result based on the metric

Note: This is an abstract class, child class must implement *load_evaluation_cfg* method.

---

eval(*inference_result*)

Running the evaluation base on the inference result. Output the metric result.

Parameters
inference_result (Dict[str, Any]) – List of inference records

Return type
Dict[str, Any]

Returns
metric result. Such as AP.5, etc.

load_evaluation_cfg()

Loading the evaluation config, where the returned config must be a dictionary which stores the configuration of the engine, such as the using soft-nms, etc.

Return type
Dict[str, Any]

Returns
Evaluation configurations

property metrics: Set[str]

Returns the metrics used in evaluation.

class mindpose.engine.evaluator.**TopDownEvaluator**(*annotation_file*, *metric='AP'*, *num_joints=17*, *config=None*, *remove_result_file=True*, *result_path='./result_keypoints.json'*)

Bases: *Evaluator*

Create an evaluator based on Topdown method. It performs the model evaluation based on the inference result (a list of records), and outputs with the metirc result.

Parameters

- **annotation_file** (str) – Path of the annotation file. It only supports COCO-format.
- **metric** (Union[str, List[str]]) – Supported metrics. Default: "AP"
- **num_joints** (int) – Number of joints. Default: 17
- **config** (Optional[Dict[str, Any]]) – Method-specific configuration. Default: None
- **remove_result_file** (bool) – Remove the cached result file after evaluation. Default: True
- **result_path** (str) – Path of the result file. Default: "./result_keypoints.json"

Inputs:

inference_result: Inference result from inference engine

Outputs:

evaluation_result: Evaluation result based on the metric

eval(*inference_result*)

Running the evaluation base on the inference result. Output the metric result.

Parameters
inference_result (Dict[str, Any]) – List of inference records

Return type
Dict[str, Any]

Returns
metric result. Such as AP.5, etc.

load_evaluation_cfg()

Loading the evaluation config, where the returned config must be a dictionary which stores the configuration of the engine, such as the using soft-nms, etc.

Return type
Dict[str, Any]

Returns
Evaluation configurations

CHAPTER

THIRTEEN

MINDPOSE.OPTIM

mindpose.optim.**create_optimizer**(*params*, *name='adam'*, *learning_rate=0.001*, *weight_decay=0.0*, *filter_bias_and_bn=True*, *loss_scale=1.0*, ***kwargs*)

Create optimizer.

Parameters

- **params** (List[Any]) – Netowrk parameters
- **name** (str) – Optimizer Name. Default: adam
- **learning_rate** (Union[float, LearningRateSchedule]) – Learning rate. Accept constant learning rate or a Learning Rate Scheduler. Default: 0.001
- **weight_decay** (float) – L2 weight decay. Default: 0.
- **filter_bias_and_bn** (bool) – whether to filter batch norm paramters and bias from weight decay. If True, weight decay will not apply on BN parameters and bias in Conv or Dense layers. Default: True.
- **loss_scale** (float) – Loss scale in mix-precision training. Default: 1.0
- ****kwargs** (Any) – Arguments feeding to the optimizer

Return type
Optimizer

Returns
Optimizer

CHAPTER

FOURTEEN

MINDPOSE.SCHEDULER

class mindpose.scheduler.**WarmupCosineDecayLR**(*lr*, *total_epochs*, *steps_per_epoch*, *warmup=0*, *min_lr=0.0*)

Bases: LearningRateSchedule

CosineDecayLR with warmup.

Parameters

- **lr** (float) – initial learning rate.
- **total_epochs** (int) – The number of total epochs of learning rate.
- **steps_per_epoch** (int) – The number of steps per epoch.
- **warmup** (Union[int, float]) – If it is a interger, it means the number of warm up steps of learning rate. If it is a decimal number, it means the fraction of total steps to warm up. Default = 0
- **min_lr** (float) – Lower lr bound. Default = 0

Inputs:

global_step: Global step

Outpus:

lr: Learning rate at that step

class mindpose.scheduler.**WarmupMultiStepDecayLR**(*lr*, *total_epochs*, *steps_per_epoch*, *milestones*, *decay_rate=0.1*, *warmup=0*)

Bases: LearningRateSchedule

Multi-step decay with warmup.

Parameters

- **lr** (float) – initial learning rate.
- **total_epochs** (int) – The number of total epochs of learning rate.
- **steps_per_epoch** (int) – The number of steps per epoch.
- **milestones** (List[int]) – The epoch number where the learning rate dacay by one time
- **decay_rate** (float) – Decay rate. Default = 0.1

- **warmup** (Union[int, float]) – If it is a interger, it means the number of warm up steps of learning rate. If it is a decimal number, it means the fraction of total steps to warm up. Default = 0

Inputs:

global_step: Global step

Outpus:

lr: Learning rate at that step

mindpose.scheduler.**create_lr_scheduler**(*name*, *lr*, *total_epochs*, *steps_per_epoch*, *warmup=0*, ***kwargs*)

Create learning rate scheduler.

Parameters

- **name** (str) – Name of the scheduler. Default: warmup_cosine_decay
- **lr** (float) – initial learning rate.
- **total_epochs** (int) – The number of total epochs of learning rate.
- **steps_per_epoch** (int) – The number of steps per epoch.
- **warmup** (Union[int, float]) – If it is a interger, it means the number of warm up steps of learning rate. If it is a decimal number, it means the fraction of total steps to warm up. Default = 0
- ****kwargs** (Any) – Arguments feed into the corresponding scheduler

Return type
LearningRateSchedule

Returns
Learning rate scheduler

CHAPTER

FIFTEEN

MINDPOSE.CALLBACKS

class mindpose.callbacks.**EvalCallback**(*inferencer=None*, *evaluator=None*, *dataset=None*, *interval=1*, *max_epoch=1*, *save_best=False*, *save_last=False*, *best_ckpt_path='./best.ckpt'*, *last_ckpt_path='./last.ckpt'*, *target_metric_name='AP'*, *summary_dir='.'*, *rank_id=None*, *device_num=None*)

Bases: Callback

Running evaluation during training. The training, evaluation result will be saved in summary record format for visualization. The best and last checkpoint can be saved after each training epoch.

Parameters

- **inferencer** (Optional[*Inferencer*]) – Inferencer for running inference on the dataset. Default: None
- **evaluator** (Optional[*Evaluator*]) – Evaluator for running evaluation. Default: None
- **dataset** (Optional[Dataset]) – The dataset used for running inference. Default: None
- **interval** (int) – The interval of running evaluation, in epoch. Default: 1
- **max_epoch** (int) – Total number of epochs for training. Default: 1
- **save_best** (bool) – Saving the best model based on the result of the target metric performance. Default: False
- **save_last** (bool) – Saving the last model. Default: False
- **best_ckpt_path** (str) – Path of the best checkpoint file. Default: "./best.ckpt"
- **last_ckpt_path** (str) – Path of the last checkpoint file. Default: "./last.ckpt"
- **target_metric_name** (str) – The metric name deciding the best model to save. Default: "AP"
- **summary_dir** (str) – The directory storing the summary record. Default: "."
- **rank_id** (Optional[int]) – Rank id. Default: None
- **device_num** (Optional[int]) – Number of devices. Default: None

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

N

O

R

S

T

W