multimodal Documentation

Corentin Dancette

Feb 25, 2022

DATASETS

multimodal is a python library providing tools for vision and language research. It provides visual features commonly used for Captionning and Visual Question Answering tasks, as well as datasets such as VQA.

CHAPTER

ONE

VISUAL QUESTION ANSWERING

Visual Question Answering datasets are available in **multimodal**. Annotations data are automatically downloaded and processed when the class is instanciated. Note that the pre-processing can take several minutes.

class multimodal.datasets.**VQA**(**args: Any*, ***kwargs: Any*)

Pytorch Dataset implementation for the VQA v1 dataset (visual question answering). See https://visualqa.org/ for more details about it.

When this class is instanciated, data will be downloaded in the directory specified by the `dir_data` parameter. Pre-processing of questions and answers will take several minutes.

When the `features` argument is specified, visual features will be downloaded as well. About 60Go will be necessary for downloading and extracting features.

Parameters

- **dir_data** (*str*) – dir for the multimodal cache (data will be downloaded in a vqa2/ folder inside this directory
- **features** (*str|object*) – which visual features should be used. Choices: `coco-bottomup` or `coco-bottomup-36` You can also give directly the feature instance.
- **split** (*str*) – Which t [`train`, `val`, `test`]
- **dir_features** (*str*) – directory to download features. If None, defaults to $dir_data/features
- **label** (*str*) – either *multilabel*, or *best*. For *multilabel*, GT scores for questions are given by the score they are assigned by the VQA evaluation. If *best*, GT is the label of the top answer.
- **tokenize_questions** (*bool*) – If True, preprocessing will tokenize questions into tokens. The tokens are stored in item["question_tokens"].
- **load** (*bool*) – default *True*. If false, then the questions annotations and questions will not be loaded in memory. This is useful if you want only to download and process the data.

__getitem__(*index*)

Returns a dictionnary with the following keys

```
{
    'image_id',
    'question_id',
    'question',
    'answer_type',
    'multiple_choice_answer',
    'answers',
```

(continues on next page)

(continued from previous page)

```
    'image_id',
    'question_type',
    'question_id',
    'scores',
    'label'   # ground truth label to be used for the loss
}
```

Aditionnaly, if visual features are used, keys from the features will be added.

static collate_fn(*batch*)
Use this method to collate batches of data.

evaluate(*predictions*)
Evaluates a list of predictions, according to the VQA evaluation protocol. See https://visualqa.org/evaluation.html.

Parameters **predictions** (*list*) – List of dictionnaries containing `question_id` and `answer` keys. The answer must be specified as a string.

Returns A dict of floats containing scores for "overall", "yes/no", number", and "other" questions.

class multimodal.datasets.**VQA2**(**args: Any*, ***kwargs: Any*)
Pytorch Dataset implementation for the VQA v2 dataset (visual question answering). See https://visualqa.org/ for more details about it.

When this class is instanciated, data will be downloaded in the directory specified by the `dir_data` parameter. Pre-processing of questions and answers will take several minutes.

When the `features` argument is specified, visual features will be downloaded as well. About 60Go will be necessary for downloading and extracting features.

Parameters

- **dir_data** (*str*) – dir for the multimodal cache (data will be downloaded in a vqa2/ folder inside this directory
- **features** (*str*) – which visual features should be used. Choices: `coco-bottomup` or `coco-bottomup-36`
- **split** (*str*) – Which t [`train`, `val`, `test`]
- **dir_features** (*str*) – directory to download features. If None, defaults to $dir_data/features
- **label** (*str*) – either *multilabel*, or *best*. For *multilabel*, GT scores for questions are given by the score they are assigned by the VQA evaluation. If *best*, GT is the label of the top answer.
- **tokenize_questions** (*bool*) – If True, preprocessing will tokenize questions into tokens. The tokens are stored in item["question_tokens"].

__getitem__(*index*)
Returns a dictionnary with the following keys

```
{
    'image_id',
    'question_id',
    'question',
    'answer_type',
```

(continues on next page)

(continued from previous page)

```
    'multiple_choice_answer',
    'answers',
    'image_id',
    'question_type',
    'question_id',
    'scores',
    'label'   # ground truth label to be used for the loss
}
```

Aditionnaly, if visual features are used, keys from the features will be added.

static collate_fn(*batch*)
Use this method to collate batches of data.

evaluate(*predictions*)
Evaluates a list of predictions, according to the VQA evaluation protocol. See https://visualqa.org/evaluation.html.

Parameters **predictions** (*list*) – List of dictionnaries containing `question_id` and `answer` keys. The answer must be specified as a string.

Returns A dict of floats containing scores for "overall", "yes/no", number", and "other" questions.

class multimodal.datasets.**VQACP**(**args: Any*, ***kwargs: Any*)
Pytorch Dataset implementation for the VQA-CP v1 dataset (visual question answering). See https://www.cc.gatech.edu/grads/a/aagrawal307/vqa-cp/ for more details about it.

When this class is instanciated, data will be downloaded in the directory specified by the `dir_data` parameter. Pre-processing of questions and answers will take several minutes.

When the `features` argument is specified, visual features will be downloaded as well. About 60Go will be necessary for downloading and extracting features.

Parameters

- **dir_data** (*str*) – dir for the multimodal cache (data will be downloaded in a vqa2/ folder inside this directory
- **features** (*str*) – which visual features should be used. Choices: `coco-bottomup` or `coco-bottomup-36`
- **split** (*str*) – Which t [`train`, `val`, `test`]
- **dir_features** (*str*) – directory to download features. If None, defaults to $dir_data/features
- **label** (*str*) – either *multilabel*, or *best*. For *multilabel*, GT scores for questions are given by the score they are assigned by the VQA evaluation. If *best*, GT is the label of the top answer.
- **tokenize_questions** (*bool*) – If True, preprocessing will tokenize questions into tokens. The tokens are stored in item["question_tokens"].

__getitem__(*index*)
Returns a dictionnary with the following keys

```
{
    'image_id',
    'question_id',
```

(continues on next page)

(continued from previous page)

```
    'question',
    'answer_type',
    'multiple_choice_answer',
    'answers',
    'image_id',
    'question_type',
    'question_id',
    'scores',
    'label'   # ground truth label to be used for the loss
}
```

Aditionnaly, if visual features are used, keys from the features will be added.

static collate_fn(*batch*)
Use this method to collate batches of data.

evaluate(*predictions*)
Evaluates a list of predictions, according to the VQA evaluation protocol. See https://visualqa.org/evaluation.html.

Parameters **predictions** (*list*) – List of dictionnaries containing `question_id` and `answer` keys. The answer must be specified as a string.

Returns A dict of floats containing scores for "overall", "yes/no", number", and "other" questions.

class multimodal.datasets.**VQACP2**(**args: Any*, ***kwargs: Any*)
Pytorch Dataset implementation for the VQA-CP v2 dataset (visual question answering). See https://www.cc.gatech.edu/grads/a/aagrawal307/vqa-cp/ for more details about it.

When this class is instanciated, data will be downloaded in the directory specified by the `dir_data` parameter. Pre-processing of questions and answers will take several minutes.

When the `features` argument is specified, visual features will be downloaded as well. About 60Go will be necessary for downloading and extracting features.

Parameters

- **dir_data** (*str*) – dir for the multimodal cache (data will be downloaded in a vqa2/ folder inside this directory
- **features** (*str*) – which visual features should be used. Choices: `coco-bottomup` or `coco-bottomup-36`
- **split** (*str*) – Which t [`train`, `val`, `test`]
- **dir_features** (*str*) – directory to download features. If None, defaults to $dir_data/features
- **label** (*str*) – either *multilabel*, or *best*. For *multilabel*, GT scores for questions are given by the score they are assigned by the VQA evaluation. If *best*, GT is the label of the top answer.
- **tokenize_questions** (*bool*) – If True, preprocessing will tokenize questions into tokens. The tokens are stored in item["question_tokens"].

__getitem__(*index*)
Returns a dictionnary with the following keys

```
{
    'image_id',
    'question_id',
    'question',
    'answer_type',
    'multiple_choice_answer',
    'answers',
    'image_id',
    'question_type',
    'question_id',
    'scores',
    'label'   # ground truth label to be used for the loss
}
```

Aditionnaly, if visual features are used, keys from the features will be added.

static collate_fn(*batch*)

Use this method to collate batches of data.

evaluate(*predictions*)

Evaluates a list of predictions, according to the VQA evaluation protocol. See https://visualqa.org/evaluation.html.

Parameters **predictions** (*list*) – List of dictionnaries containing `question_id` and `answer` keys. The answer must be specified as a string.

Returns A dict of floats containing scores for "overall", "yes/no", number", and "other" questions.

CHAPTER

TWO

CLEVR

https://cs.stanford.edu/people/jcjohns/clevr/

class multimodal.datasets.**CLEVR**(**args: Any*, ***kwargs: Any*)

CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning.

See https://cs.stanford.edu/people/jcjohns/clevr/

Warning: instanciating this class will download a 18Gb file to the multimodal data directory (by default in your applications data). You can specify the multimodal data directory by specifying the `dir_data` argument, or specifying it in your path.

Parameters

- **dir_data** (*str*) – dir for the multimodal cache (data will be downloaded in a clevr/ folder inside this directory
- **split** (*str*) – either train, val or test
- **transform** – torchvision transform applied to images. By default, only ToTensor.

__getitem__(*index: int*)

Returns a dictionnary with the following keys:

```
{
    "index",
    "question",
    "answer":,
    "question_family_index":,
    "image_filename":,
    "image_index":,
    "image"
    "label",
}
```

Note that you can recover the program for an example by using the index:

```
index = item["index"][0]  #  first item of batch
program = clevr.questions[index]["program"]
```

CHAPTER

THREE

TOKENIZERS

3.1 Base class

We implement a basic tokenizer class, built on torchtext basic_english tokenizer. Its purpose is to transform text into a series of token_ids (integers), that will be fed to a word vector.

class multimodal.text.**BasicTokenizer**(*tokens: List[str] = []*, *sentences: List[str] = []*, *name: Optional[str] = None*, *pad_token='<pad>'*, *unk_token='<unk>'*, *padding_side='right'*, *dir_data: Optional[str] = None*)

This class maps word tokens to token_ids. In case of unknown token ids, the It will also pad the data.

Parameters

- **tokens** (*list*) – Tokens to add in the dictionnary. Those can be tokens from pretrain vectors.
- **sentences** (*list*) – List of sentences that need to be tokenized first, before building the vocab. Tokens from those sentences will be added to the vocabulary if they were not in it already.
- **name** (*str*) – name which will be used to save the tokenizer. Use a different name when changing the tokens.
- **pad_token** (*str*) – token used to pad the data.
- **unk_token** (*str*) – token used for unknown words. The id is saved in the attribute unk_token_id.
- **pad_side** (*str*) – either "left" or "right". The pad_token_id attribute will save the position.
- **dir_data** (*str*) – directory to save multimodal data.

tokenize(*s*, *replace_unk=True*, *padding=True*)

This function will return the tokenized representation of the input. Example: tokenize("Hello there") will return ["hello", "there"], assuming both words are in the vocabulary.

In case a list of strings is given as input, this function will add padding tokens to ensure that all outputs have the same length.

Parameters

- **s** (*str | List[str]*) – Either a string or a list of string, to be tokenized.
- **keep_unk** (*bool*) – If true, then the tokenizes will not replace unknown words with the UNK token. Default: false
- **padding** (*bool*) – Whether to add the padding token or not.

convert_tokens_to_ids(*tokens*)
Converts tokenized representations :param tokens: List of string tokens that will be converted to their token ids. :type tokens: list :param If a token is missing from the vocabulary: :param it will be converted to self.unk_token_id.: :param Padding tokens will be converted to self.pad_token_id.:

3.2 VQA v2

The pretrained tokenizer for VQA v2 is called `pretrained-vqa2`.

```
from multimodal.text import BasicTokenizer
tokenizer = BasicTokenizer.from_pretrained("pretrained-vqa2")
tokens = tokenizer("What color is the car?")
# feed tokens to model
```

CHAPTER

FOUR

BOTTOM-UP TOP-DOWN OBJECT FEATURES

Those visual features were introduced by Anderson et. al. in the paper Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering.

They are extracted with a Faster R-CNN, and trained on the Visual Genome dataset to detect objects and their attributes (shapes, colors...).

multimodal provides a class to download and use those features extracted on the COCO image dataset. They can be used for most Visual Question Answering and Captionning that use the this dataset for images.

```
from multimodal.features import COCOBottomUpFeatures

bottomup = COCOBottomUpFeatures(
    features="coco-bottomup-36",
    dir_data="/data/multimodal",
)
image_id = 13455
feats = bottomup[image_id]
print(feats.keys())
# ['image_w', 'image_h', 'num_boxes', 'boxes', 'features']
print(feats["features"].shape)  # numpy array
# (36, 2048)
```

class multimodal.features.**COCOBottomUpFeatures**(*features: str*, *dir_data: Optional[str] = None*)

Bottom up features for the COCO dataset

Parameters

- **features** (*str*) – one of [trainval2014_36, trainval2014, test2014_36, test2014, test2015-36, test2015]. Specifies the split, and the number of detected objects. _36 means 36 objetcs are detected in every image, and otherwise, the number is based on a detection threshold, between 10 and 100 objects.
- **dir_data** (*str*) – Directory where multimodal data will be downloaded. You need at least 60Go for downloading and extracting the features.

__getitem__(*image_id: int*)

Get the features.

Parameters **image_id** (*str|int*) – The id of the image in COCO dataset.

Returns

A dictionnary containing the following keys:

```
{
    'image_id',
    'image_h': height
    'image_w': width
    'num_boxes': number of objects
    'boxes': Numpy array of shape (N, 4) containing bounding box␣
↪coordinates
    'features': Numpy array of shape (N, 2048) containing features.
}
```

keys()

Returns List of all keys

Return type list

CHAPTER

FIVE

MODELS

5.1 UpDown

The Bottom-Up and Top-Down Attention for VQA model is available in multimodal.

You can either train it directly (you will need to clone the repository), or import it in your code.

CHAPTER

SIX

COMMAND LINE INTERFACE

All commands start with `python -m multimodal <subcommand>`

The subcommands available are listed here:

6.1 VQA Evaluation: `vqa-eval`

6.1.1 Description

Run the evaluation, following the VQA evaluation metric, taking into account answers from multiple humans.

```
python -m multimodal vqa-eval -p <predictions-path> -s <split> --dir_data <multimodal_
↪dir_data>
```

6.1.2 Options

-p <path>, **--predictions** <path>
: path to predictions, should follow the official VQA evaluation format (see https://visualqa.org/evaluation.html)

-s <split>, **--split** <split>
: VQA split, either `train`, `val` or `test` depending on the dataset (in VQA-CP, there are only train and test).

--dir_data <dir_data> (optional)
: path where data will be downloaded if necessary. By default in appdata.

Example

```
$ python -m multimodal vqa-eval -s val -p logs/updown/predictions.json
Loading questions
Loading annotations
Loading aid_to_ans
{'overall': 0.6346422273435531, 'yes/no': 0.8100979625284017, 'number': 0.
↪42431932892585483, 'other': 0.5569148080507953}
```

6.2 Data Download: download

6.2.1 Description

Download and process data.

```
python -m multimodal download <dataset> --dir_data <dir_data>
```

6.2.2 Options

--dir_data <dir_data> (optional)
: path where data will be downloaded if necessary. By default in appdata.

dataset
: Name of the dataset to download. Can be either VQA, VQA2, VQACP, VQACP2, coco-bottom-up, coco-bottomup-36.

This library was developped by Corentin Dancette. If you have any new feature request or want to report a bug, please open an issue on the github tracker, or submit a Pull Request.

INDEX

Symbols

B

C

D

E

K

P

T

V