staged_keras_wrapper Documentation

Release 0.55

Marc Bolaños

Nov 30, 2016

CONTENTS

Contents:

CHAPTER

ONE

AVAILABLE MODULES

Documentation for the keras_wrapper module.

See code examples in demo.ipynb and test.py

dataset.py

class keras_wrapper.dataset.**Data_Batch_Generator**(*set_split*, *net*, *dataset*, *num_iterations*, *batch_size=50*, *normalization=False*, *data_augmentation=True*, *mean_substraction=True*, *predict=False*, *random_samples=-1*, *shuffle=True*)

Batch generator class. Retrieves batches of data.

generator()

Gets and processes the data :return: generator with the data

class keras_wrapper.dataset.**Dataset**(*name*, *path*, *silence=False*)

Class for defining instances of databases adapted for Keras. It includes several utility functions for easily managing data splits, image loading, mean calculation, etc.

build_vocabulary(*captions*, *id*, *tokfun*, *do_split*, *min_occ=0*, *n_words=0*)

Vocabulary builder for data of type 'text'

Parameters

- **captions** – Corpus sentences
- **id** – Dataset id of the text
- **tokfun** – Tokenization function. (used?)
- **do_split** – Split sentence by words or use the full sentence as a class.
- **min_occ** – Minimum occurrences of each word to be included in the dictionary.
- **n_words** – Maximum number of words to include in the dictionary.

Returns None.

calculateTrainMean(*id*)

Calculates the mean of the data belonging to the training set split in each channel.

getClassID(*class_name*, *id*)

Returns the class id (int) for a given class string.

getX(*set_name*, *init*, *final*, *normalization_type='0-1'*, *normalization=False*, *meanSubstraction=True*, *dataAugmentation=True*, *debug=False*)

Gets all the data samples stored between the positions init to final

Parameters

- **set_name** – 'train', 'val' or 'test' set
- **init** – initial position in the corresponding set split. Must be bigger or equal than 0 and smaller than final.
- **final** – final position in the corresponding set split.
- **debug** – if True all data will be returned without preprocessing

'raw-image', 'video', 'image-features' and 'video-features'-related parameters

Parameters **normalization** – indicates if we want to normalize the data.

'image-features' and 'video-features'-related parameters

Parameters **normalization_type** – indicates the type of normalization applied. See available types in self.__available_norm_im_vid for 'raw-image' and 'video' and self.__available_norm_feat for 'image-features' and 'video-features'.

'raw-image' and 'video'-related parameters

Parameters

- **meanSubstraction** – indicates if we want to substract the training mean from the returned images (only applicable if normalization=True)
- **dataAugmentation** – indicates if we want to apply data augmentation to the loaded images (random flip and cropping)

Returns X, list of input data variables from sample 'init' to 'final' belonging to the chosen 'set_name'

getXY(*set_name*, *k*, *normalization_type='0-1'*, *normalization=False*, *meanSubstraction=True*, *dataAugmentation=True*, *debug=False*)

Gets the [X,Y] pairs for the next 'k' samples in the desired set.

Parameters

- **set_name** – 'train', 'val' or 'test' set
- **k** – number of consecutive samples retrieved from the corresponding set.
- **sorted_batches** – If True, it will pick data of the same size
- **debug** – if True all data will be returned without preprocessing

'raw-image', 'video', 'image-features' and 'video-features'-related parameters

Parameters **normalization** – indicates if we want to normalize the data.

'image-features' and 'video-features'-related parameters

Parameters **normalization_type** – indicates the type of normalization applied. See available types in self.__available_norm_im_vid for 'image' and 'video' and self.__available_norm_feat for 'image-features' and 'video-features'.

'raw-image' and 'video'-related parameters

Parameters

- **meanSubstraction** – indicates if we want to substract the training mean from the returned images (only applicable if normalization=True)

- **dataAugmentation** – indicates if we want to apply data augmentation to the loaded images (random flip and cropping)

Returns [X,Y], list of input and output data variables of the next 'k' consecutive samples belonging to the chosen 'set_name'

Returns [X, Y, [new_last, last, surpassed]] if debug==True

getXY_FromIndices(*set_name*, *k*, *normalization_type='0-1'*, *normalization=False*, *meanSubstraction=True*, *dataAugmentation=True*, *debug=False*)

Gets the [X,Y] pairs for the samples in positions 'k' in the desired set.

Parameters

- **set_name** – 'train', 'val' or 'test' set
- **k** – positions of the desired samples
- **sorted_batches** – If True, it will pick data of the same size
- **debug** – if True all data will be returned without preprocessing

'raw-image', 'video', 'image-features' and 'video-features'-related parameters

Parameters **normalization** – indicates if we want to normalize the data.

'image-features' and 'video-features'-related parameters

Parameters **normalization_type** – indicates the type of normalization applied. See available types in self.__available_norm_im_vid for 'raw-image' and 'video' and self.__available_norm_feat for 'image-features' and 'video-features'.

'raw-image' and 'video'-related parameters

Parameters

- **meanSubstraction** – indicates if we want to substract the training mean from the returned images (only applicable if normalization=True)
- **dataAugmentation** – indicates if we want to apply data augmentation to the loaded images (random flip and cropping)

Returns [X,Y], list of input and output data variables of the samples identified by the indices in 'k' samples belonging to the chosen 'set_name'

Returns [X, Y, [new_last, last, surpassed]] if debug==True

getY(*set_name*, *init*, *final*, *normalization_type='0-1'*, *normalization=False*, *meanSubstraction=True*, *dataAugmentation=True*, *debug=False*)

Gets the [Y] samples for the FULL dataset

Parameters

- **set_name** – 'train', 'val' or 'test' set
- **init** – initial position in the corresponding set split. Must be bigger or equal than 0 and smaller than final.
- **final** – final position in the corresponding set split.
- **debug** – if True all data will be returned without preprocessing

'raw-image', 'video', 'image-features' and 'video-features'-related parameters

Parameters

- **normalization** – indicates if we want to normalize the data.

- **normalization_type** – indicates the type of normalization applied. See available types in self.__available_norm_im_vid for 'raw-image' and 'video' and self.__available_norm_feat for 'image-features' and 'video-features'.

'raw-image' and 'video'-related parameters

Parameters

- **meanSubstraction** – indicates if we want to substract the training mean from the returned images (only applicable if normalization=True)
- **dataAugmentation** – indicates if we want to apply data augmentation to the loaded images (random flip and cropping)

Returns Y, list of output data variables from sample 'init' to 'final' belonging to the chosen 'set_name'

loadFeatures(*X*, *feat_len*, *normalization_type='L2'*, *normalization=False*, *loaded=False*, *external=False*, *data_augmentation=True*)

Loads and normalizes features.

Parameters

- **X** – Features to load.
- **feat_len** – Length of the features.
- **normalization_type** – Normalization to perform to the features (see: self.__available_norm_feat)
- **normalization** – Whether to normalize or not the features.
- **loaded** – Flag that indicates if these features have been already loaded.
- **external** – Boolean indicating if the paths provided in 'X' are absolute paths to external images
- **data_augmentation** – Perform data augmentation (with mean=0.0, std_dev=0.01)

Returns Loaded features as numpy array

loadImages(*images*, *id*, *normalization_type='0-1'*, *normalization=False*, *meanSubstraction=True*, *dataAugmentation=True*, *external=False*, *loaded=False*, *prob_flip_horizontal=0.5*, *prob_flip_vertical=0.0*)

Loads a set of images from disk.

:param images : list of image string names or list of matrices representing images :param normalization_type: type of normalization applied :param normalization : whether we applying a 0-1 normalization to the images :param meanSubstraction : whether we are removing the training mean :param dataAugmentation : whether we are applying dataAugmentatino (random cropping and horizontal flip) :param external : if True the images will be loaded from an external database, in this case the list of images must be absolute paths :param loaded : set this option to True if images is a list of matricies instead of a list of strings :param prob_flip_horizontal: probability of horizontal image flip if applying dataAugmentation :param prob_flip_vertical: probability of vertical image flip if applying dataAugmentation

loadText(*X*, *vocabularies*, *max_len*, *offset*, *fill*, *pad_on_batch*, *words_so_far*)

Text encoder: Transforms samples from a text representation into a numerical one. It also masks the text.

Parameters

- **X** – Text to encode.
- **vocabularies** – Mapping word -> index
- **max_len** – Maximum length of the text.

- **offset** – Shifts the text to the right, adding null symbol at the start
- **fill** – 'start': the resulting vector will be filled with 0s at the beginning,

'end': it will be filled with 0s at the end. :param pad_on_batch: Whether we get sentences with length of the maximum length of the minibatch or sentences with a fixed (max_text_length) length. :param words_so_far: Experimental feature. Use with caution. :return: Text as sequence of number. Mask for each sentence.

loadVideos(*n_frames*, *id*, *last*, *set_name*, *max_len*, *normalization_type*, *normalization*, *meanSubstraction*, *dataAugmentation*)

Loads a set of videos from disk. (Untested!)

Parameters

- **n_frames** – Number of frames per video
- **id** – Id to load
- **last** – Last video loaded
- **set_name** – 'train', 'val', 'test'
- **max_len** – Maximum length of videos
- **normalization_type** – Type of normalization applied
- **normalization** – Whether we apply a 0-1 normalization to the images
- **meanSubstraction** – Whether we are removing the training mean
- **dataAugmentation** – Whether we are applying dataAugmentatino (random cropping and horizontal flip)

Returns

preprocessBinary(*labels_list*)

Preprocesses binary classes. :param labels_list: Binary label list given as an instance of the class list. :return: Preprocessed labels.

preprocessCategorical(*labels_list*)

Preprocesses categorical data. :param labels_list: Label list. Given as a path to a file or as an instance of the class list. :return: Preprocessed labels.

preprocessFeatures(*path_list*, *id*, *set_name*, *feat_len*)

Preprocesses features. We should give a path to a text file where each line must contain a path to a .npy file storing a feature vector. Alternatively "path_list" can be an instance of the class list. :param path_list: Path to a text file where each line must contain a path to a .npy file storing a feature vector. Alternatively, instance of the class list. :param id: Dataset id :param set_name: Used? :param feat_len: Length of features. If all features have the same length, given as a number. Otherwise, list. :return: Preprocessed features

preprocessReal(*labels_list*)

Preprocesses real classes. :param labels_list: Label list. Given as a path to a file or as an instance of the class list. :return: Preprocessed labels.

preprocessText(*annotations_list*, *id*, *set_name*, *tokenization*, *build_vocabulary*, *max_text_len*, *max_words*, *offset*, *fill*, *min_occ*, *pad_on_batch*, *words_so_far*)

Preprocess 'text' data type: Builds vocabulary (if necessary) and preprocesses the sentences. Also sets Dataset parameters. :param annotations_list: Path to the sentences to process. :param id: Dataset id of the data. :param set_name: Name of the current set ('train', 'val', 'test') :param tokenization: Tokenization to perform. :param build_vocabulary: Whether we should build a vocabulary for this text or not.

:param max_text_len: Maximum length of the text. If max_text_len == 0, we treat the full sentence as a class. :param max_words: Maximum number of words to include in the dictionary. :param offset: Text shifting. :param fill: Whether we path with zeros at the beginning or at the end of the sentences. :param min_occ: Minimum occurrences of each word to be included in the dictionary. :param pad_on_batch: Whether we get sentences with length of the maximum length of the minibatch or sentences with a fixed (max_text_length) length. :param words_so_far: Experimental feature. Should be ignored. :return: Preprocessed sentences.

resetCounters(*set_name='all'*)
Resets some basic counter indices for the next samples to read.

setClasses(*path_classes*, *id*)
Loads the list of classes of the dataset. Each line must contain a unique identifier of the class. :param path_classes: Path to a text file with the classes or an instance of the class list. :param id: Dataset id :return: None

setInput(*path_list, set_name, type='raw-image', id='image', repeat_set=1, required=True, img_size=[256, 256, 3], img_size_crop=[227, 227, 3], use_RGB=True, max_text_len=35, tokenization='tokenize_basic', offset=0, fill='end', min_occ=0, pad_on_batch=True, build_vocabulary=False, max_words=0, words_so_far=False, feat_len=1024, max_video_len=26*)
Loads a list of samples which can contain all samples from the 'train', 'val', or 'test' sets (specified by set_name).

General parameters

Parameters

- **path_list** – can either be a path to a text file containing the paths to the images or a python list of paths
- **set_name** – identifier of the set split loaded ('train', 'val' or 'test')
- **type** – identifier of the type of input we are loading (accepted types can be seen in self.__accepted_types_inputs)
- **id** – identifier of the input data loaded
- **repeat_set** – repeats the inputs given (useful when we have more outputs than inputs). Int or array of ints.
- **required** – flag for optional inputs

'raw-image'-related parameters

Parameters

- **img_size** – size of the input images (any input image will be resized to this)
- **img_size_crop** – size of the cropped zone (when dataAugmentation=False the central crop will be used)

'text'-related parameters

Parameters

- **tokenization** – type of tokenization applied (must be declared as a method of this class) (only applicable when type=='text').
- **build_vocabulary** – whether a new vocabulary will be built from the loaded data or not (only applicable when type=='text').
- **max_text_len** – maximum text length, the rest of the data will be padded with 0s (only applicable if the output data is of type 'text').

- **max_words** – a maximum of 'max_words' words from the whole vocabulary will be chosen by number or occurrences
- **offset** – number of timesteps that the text is shifted to the right (for sequential conditional models, which take as input the previous output)
- **fill** – select whether padding before or after the sequence
- **min_occ** – minimum number of occurrences allowed for the words in the vocabulary. (default = 0)
- **pad_on_batch** – the batch timesteps size will be set to the length of the largest sample +1 if True, max_len will be used as the fixed length otherwise
- **words_so_far** – if True, each sample will be represented as the complete set of words until the point defined by the timestep dimension (e.g. t=0 'a', t=1 'a dog', t=2 'a dog is', etc.)

'image-features' and 'video-features'- related parameters

Parameters **feat_len** – size of the feature vectors for each dimension. We must provide a list if the features are not vectors.

'video'-related parameters

Parameters **max_video_len** – maximum video length, the rest of the data will be padded with 0s (only applicable if the input data is of type 'video' or video-features').

setInputGeneral(*path_list, split=[0.8, 0.1, 0.1], shuffle=True, type='raw-image', id='image'*)
DEPRECATED

Loads a single list of samples from which train/val/test divisions will be applied.

Parameters

- **path_list** – path to the text file with the list of images.
- **split** – percentage of images used for [training, validation, test].
- **shuffle** – whether we are randomly shuffling the input samples or not.
- **type** – identifier of the type of input we are loading (accepted types can be seen in self.__accepted_types_inputs)
- **id** – identifier of the input data loaded

setLabels(*labels_list*, *set_name*, *type='categorical'*, *id='label'*)
DEPRECATED

setList(*path_list*, *set_name*, *type='raw-image'*, *id='image'*)
DEPRECATED

setListGeneral(*path_list, split=[0.8, 0.1, 0.1], shuffle=True, type='raw-image', id='image'*)
Deprecated

setOutput(*path_list*, *set_name*, *type='categorical'*, *id='label'*, *repeat_set=1*, *tokenization='tokenize_basic'*, *max_text_len=0*, *offset=0*, *fill='end'*, *min_occ=0*, *pad_on_batch=True*, *words_so_far=False*, *build_vocabulary=False*, *max_words=0*, *sample_weights=False*)
Loads a set of output data, usually (type=='categorical') referencing values in self.classes (starting from 0)

General parameters

Parameters

- **path_list** – can either be a path to a text file containing the labels or a python list of labels.
- **set_name** – identifier of the set split loaded ('train', 'val' or 'test').
- **type** – identifier of the type of input we are loading (accepted types can be seen in self.__accepted_types_outputs).
- **id** – identifier of the input data loaded.
- **repeat_set** – repeats the outputs given (useful when we have more inputs than outputs). Int or array of ints.

'text'-related parameters

Parameters

- **tokenization** – type of tokenization applied (must be declared as a method of this class) (only applicable when type=='text').
- **build_vocabulary** – whether a new vocabulary will be built from the loaded data or not (only applicable when type=='text').
- **max_text_len** – maximum text length, the rest of the data will be padded with 0s (only applicable if the output data is of type 'text') Set to 0 if the whole sentence will be used as an output class.
- **max_words** – a maximum of 'max_words' words from the whole vocabulary will be chosen by number or occurrences
- **offset** – number of timesteps that the text is shifted to the right (for sequential conditional models, which take as input the previous output)
- **fill** – select whether padding before or after the sequence
- **min_occ** – minimum number of occurrences allowed for the words in the vocabulary. (default = 0)
- **pad_on_batch** – the batch timesteps size will be set to the length of the largest sample +1 if True, max_len will be used as the fixed length otherwise
- **words_so_far** – if True, each sample will be represented as the complete set of words until the point defined by the timestep dimension (e.g. t=0 'a', t=1 'a dog', t=2 'a dog is', etc.)

setSilence(*silence*)
Changes the silence mode of the 'Dataset' instance.

setTrainMean(*mean_image*, *id*, *normalization=False*)
Loads a pre-calculated training mean image, 'mean_image' can either be:

•numpy.array (complete image)

•list with a value per channel

•string with the path to the stored image.

Parameters **id** – identifier of the type of input whose train mean is being introduced.

shuffleTraining()
Applies a random shuffling to the training samples.

tokenize_aggressive(*caption*, *lowercase=True*)

Aggressive tokenizer for the input/output data of type 'text':

- Removes punctuation
- Optional lowercasing

Parameters

- **caption** – String to tokenize
- **lowercase** – Whether to lowercase the caption or not

Returns Tokenized version of caption

tokenize_basic(*caption*, *lowercase=True*)

Basic tokenizer for the input/output data of type 'text':

- Splits punctuation
- Optional lowercasing

Parameters

- **caption** – String to tokenize
- **lowercase** – Whether to lowercase the caption or not

Returns Tokenized version of caption

tokenize_icann(*caption*)

Tokenization used for the icann paper:

- Removes some punctuation (. , ")
- Lowercasing

Parameters **caption** – String to tokenize

Returns Tokenized version of caption

tokenize_montreal(*caption*)

Similar to tokenize_icann

- Removes some punctuation
- Lowercase

Parameters **caption** – String to tokenize

Returns Tokenized version of caption

tokenize_none(*caption*)

Does not tokenizes the sentences. Only performs a stripping

Parameters **caption** – String to tokenize

Returns Tokenized version of caption

tokenize_none_char(*caption*)

Character-level tokenization. Respects all symbols. Separates chars. Inserts <space> sybmol for spaces. If found an escaped char, "'" symbol, it is converted to the original one # List of escaped chars (by moses tokenizer) & -> & | -> | < -> < > -> > ' -> ' " -> " [-> [ ] ->] :param caption: String to tokenize :return: Tokenized version of caption

tokenize_questions(*caption*)

Basic tokenizer for VQA questions:

- Lowercasing
- Splits contractions
- Removes punctuation
- Numbers to digits

Parameters **caption** – String to tokenize

Returns Tokenized version of caption

tokenize_soft(*caption*, *lowercase=True*)

Tokenization used for the icann paper:

- Removes very little punctuation
- Lowercase

Parameters

- **caption** – String to tokenize
- **lowercase** – Whether to lowercase the caption or not

Returns Tokenized version of caption

class keras_wrapper.dataset.**Homogeneous_Data_Batch_Generator**(*set_split*, *net*, *dataset*, *num_iterations*, *batch_size=50*, *maxlen=100*, *normalization=False*, *data_augmentation=True*, *mean_substraction=True*, *predict=False*)

Retrieves batches of the same length. Parts of the code borrowed from https://github.com/kelvinxu/arctic-captions/blob/master/homogeneous_data.py

keras_wrapper.dataset.**create_dir_if_not_exists**(*directory*)

Creates a directory if it doen't exist

Parameters **directory** – Directory to create

Returns None

keras_wrapper.dataset.**loadDataset**(*dataset_path*)

Loads a previously saved Dataset object.

Parameters **dataset_path** – Path to the stored Dataset to load

Returns Loaded Dataset object

keras_wrapper.dataset.**saveDataset**(*dataset*, *store_path*)

Saves a backup of the current Dataset object.

Parameters

- **dataset** – Dataset object to save
- **store_path** – Saving path

Returns None

cnn_model.py

keras_wrapper.cnn_model.**CNN_Model**
alias of *Model_Wrapper*

class keras_wrapper.cnn_model.**Model_Wrapper**(*nOutput=1000, type='basic_model', silence=False, input_shape=[256, 256, 3], structure_path=None, weights_path=None, seq_to_functional=False, model_name=None, plots_path=None, models_path=None, inheritance=False*)

Wrapper for Keras' models. It provides the following utilities:

- Training visualization module.
- Set of already implemented CNNs for quick definition.
- Easy layers re-definition for finetuning.
- Model backups.
- Easy to use training and test methods.

BeamSearchNet(*ds*, *parameters*)

Approximates by beam search the best predictions of the net on the dataset splits chosen. :param batch_size: size of the batch :param n_parallel_loaders: number of parallel data batch loaders :param normalization: apply data normalization on images/features or not (only if using images/features as input) :param mean_substraction: apply mean data normalization on images or not (only if using images as input) :param predict_on_sets: list of set splits for which we want to extract the predictions ['train', 'val', 'test'] :param optimized_search: boolean indicating if the used model has the optimized Beam Search implemented (separate self.model_init and self.model_next models for reusing the information from previous timesteps). The following attributes must be inserted to the model when building an optimized search model:

•ids_inputs_init: list of input variables to model_init (must match inputs to conventional model)

•ids_outputs_init: list of output variables of model_init (model probs must be the first output)

•ids_inputs_next: list of input variables to model_next (previous word must be the first input)

•ids_outputs_next: list of output variables of model_next (model probs must be the first output and the number of out variables must match the number of in variables)

•matchings_init_to_next: dictionary from 'ids_outputs_init' to 'ids_inputs_next'

•matchings_next_to_next: dictionary from 'ids_outputs_next' to 'ids_inputs_next'

Returns predictions dictionary with set splits as keys and matrices of predictions as values.

Empty(*nOutput*, *input*)

Creates an empty Model_Wrapper (can be externally defined)

GAP(*nOutput*, *input*)

Creates a GAP network for object localization as described in the paper Zhou B, Khosla A, Lapedriza A, Oliva A, Torralba A. Learning Deep Features for Discriminative Localization. arXiv preprint arXiv:1512.04150. 2015 Dec 14.

Outputs: 'GAP/softmax' output of the final softmax classification 'GAP/conv' output of the generated convolutional maps.

Identity_Layer(*nOutput*, *input*)
Builds an dummy Identity_Layer, which should give as output the same as the input. Only used for passing the output from a previous stage to the next (see Staged_Network).

One_vs_One(*nOutput*, *input*)
Builds a simple One_vs_One network with 3 convolutional layers (useful for ECOC models).

One_vs_One_Inception(*nOutput=2, input=[224, 224, 3]*)
Builds a simple One_vs_One_Inception network with 2 inception layers (useful for ECOC models).

One_vs_One_Inception_v2(*nOutput=2, input=[224, 224, 3]*)
Builds a simple One_vs_One_Inception_v2 network with 2 inception layers (useful for ECOC models).

Union_Layer(*nOutput*, *input*)
Network with just a dropout and a softmax layers which is intended to serve as the final layer for an ECOC model

VGG_16(*nOutput*, *input*)
Builds a VGG model with 16 layers.

VGG_16_FunctionalAPI(*nOutput*, *input*)
16-layered VGG model implemented in Keras' Functional API

VGG_16_PReLU(*nOutput*, *input*)
Builds a VGG model with 16 layers and with PReLU activations.

add_One_vs_One_Inception(*input*, *input_shape*, *id_branch*, *nOutput=2*, *activation='softmax'*)
Builds a simple One_vs_One_Inception network with 2 inception layers on the top of the current model (useful for ECOC_loss models).

add_One_vs_One_Inception_Functional(*input*, *input_shape*, *id_branch*, *nOutput=2*, *activation='softmax'*)
Builds a simple One_vs_One_Inception network with 2 inception layers on the top of the current model (useful for ECOC_loss models).

add_One_vs_One_Inception_v2(*input*, *input_shape*, *id_branch*, *nOutput=2*, *activation='softmax'*)
Builds a simple One_vs_One_Inception_v2 network with 2 inception layers on the top of the current model (useful for ECOC_loss models).

basic_model(*nOutput*, *input*)
Builds a basic CNN model.

beam_search(*X*, *params*, *null_sym=2*)
Beam search method for Cond models. (https://en.wikibooks.org/wiki/Artificial_Intelligence/Search/Heuristic_search/Beam_search) The algorithm in a nutshell does the following:

1.k = beam_size

2.open_nodes = [[]] * k

3.while k > 0:

3.1. Given the inputs, get (log) probabilities for the outputs. 3.2. Expand each open node with all possible output. 3.3. Prune and keep the k best nodes. 3.4. If a sample has reached the <eos> symbol:

3.4.1. Mark it as final sample. 3.4.2. k -= 1

3.5. Build new inputs (state_below) and go to 1.

4.return final_samples, final_scores

Parameters

- **X** – Model inputs
- **params** – Search parameters
- **null_sym** – <null> symbol

Returns UNSORTED list of [k_best_samples, k_best_scores] (k: beam size)

checkParameters(*input_params*, *default_params*)
Validates a set of input parameters and uses the default ones if not specified.

decode_predictions(*preds*, *temperature*, *index2word*, *sampling_type*, *verbose=0*)
Decodes predictions :param preds: Predictions codified as the output of a softmax activation function. :param temperature: Temperature for sampling. :param index2word: Mapping from word indices into word characters. :param sampling_type: 'max_likelihood' or 'multinomial'. :param verbose: Verbosity level, by default 0. :return: List of decoded predictions.

decode_predictions_beam_search(*preds*, *index2word*, *pad_sequences=False*, *verbose=0*)
Decodes predictions from the BeamSearch method. :param preds: Predictions codified as word indices. :param index2word: Mapping from word indices into word characters. :param pad_sequences: Whether we should make a zero-pad on the input sequence. :param verbose: Verbosity level, by default 0. :return: List of decoded predictions

decode_predictions_one_hot(*preds*, *index2word*, *verbose=0*)
Decodes predictions following a one-hot codification. :param preds: Predictions codified as one-hot vectors. :param index2word: Mapping from word indices into word characters. :param verbose: Verbosity level, by default 0. :return: List of decoded predictions

ended_training()
Indicates if the model has early stopped.

log(*mode*, *data_type*, *value*)
Stores the train and val information for plotting the training progress.

Parameters

- **mode** – 'train', or 'val'
- **data_type** – 'iteration', 'loss' or 'accuracy'
- **value** – numerical value taken by the data_type

plot()
Plots the training progress information.

predictNet(*ds*, *parameters*, *out_name=None*)
Returns the predictions of the net on the dataset splits chosen. The input 'parameters' is a dict() which may contain the following parameters:

Parameters

- **batch_size** – size of the batch
- **n_parallel_loaders** – number of parallel data batch loaders
- **normalize** – apply data normalization on images/features or not (only if using images/features as input)
- **mean_substraction** – apply mean data normalization on images or not (only if using images as input)

- **predict_on_sets** – list of set splits for which we want to extract the predictions ['train', 'val', 'test']

Returns predictions dictionary with set splits as keys and matrices of predictions as values.

predictOnBatch(*X*, *in_name=None*, *out_name=None*, *expand=False*)
Applies a forward pass and returns the predicted values.

predict_cond(*X*, *states_below*, *params*, *ii*, *optimized_search=False*, *prev_out=None*)
Returns predictions on batch given the (static) input X and the current history (states_below) at time-step ii. WARNING!: It's assumed that the current history (state_below) is the last input of the model! See Dataset class for more information :param X: Input context :param states_below: Batch of partial hypotheses :param params: Decoding parameters :param ii: Decoding time-step :param optimized_search: indicates if we are using optimized search (only applicable if beam search specific models self.model_init and self.model_next models are defined) :param prev_out: output from the previous timestep, which will be reused by self.model_next (only applicable if beam search specific models self.model_init and self.model_next models are defined) :return: Network predictions at time-step ii

prepareData(*X_batch*, *Y_batch=None*)
Prepares the data for the model, depending on its type (Sequential, Model, Graph). :param X_batch: Batch of input data. :param Y_batch: Batch output data. :return: Prepared data.

removeInputs(*inputs_names*)
Removes the list of inputs whose names are passed by parameter from the current network. This function is only valid for Graph models.

removeLayers(*layers_names*)
Removes the list of layers whose names are passed by parameter from the current network. Function only valid for Graph models. Use self.replaceLastLayers(...) for Sequential models.

removeOutputs(*outputs_names*)
Removes the list of outputs whose names are passed by parameter from the current network. This function is only valid for Graph models.

replaceLastLayers(*num_remove*, *new_layers*)
Replaces the last 'num_remove' layers in the model by the newly defined in 'new_layers'. Function only valid for Sequential models. Use self.removeLayers(...) for Graph models.

resumeTrainNet(*ds*, *parameters*, *out_name=None*)
DEPRECATED

Resumes the last training state of a stored model keeping also its training parameters. If we introduce any parameter through the argument 'parameters', it will be replaced by the old one.

Parameters **out_name** – name of the output node that will be used to evaluate the network accuracy. Only applicable for Graph models.

sample(*a*, *temperature=1.0*)
Helper function to sample an index from a probability array :param a: Probability array :param temperature: The higher, the flatter probabilities. Hence more random outputs. :return:

sampling(*scores*, *sampling_type='max_likelihood'*, *temperature=1.0*)
Sampling words (each sample is drawn from a categorical distribution). Or picks up words that maximize the likelihood. :param scores: array of size #samples x #classes; every entry determines a score for sample i having class j :param sampling_type: :param temperature: Temperature for the predictions. The higher, the flatter probabilities. Hence more random outputs. :return: set of indices chosen as output, a vector of size #samples

setInputsMapping(*inputsMapping*)
Sets the mapping of the inputs from the format given by the dataset to the format received by the model.

Parameters **inputsMapping** – dictionary with the model inputs' identifiers as keys and the dataset inputs' identifiers as values. If the current model is Sequential then keys must be ints with the desired input order (starting from 0). If it is Graph then keys must be str.

setName(*model_name*, *plots_path=None*, *models_path=None*, *clear_dirs=True*)
Changes the name (identifier) of the Model_Wrapper instance.

setOptimizer(*lr=None*, *momentum=None*, *loss=None*, *metrics=None*)
Sets a new optimizer for the CNN model.

Parameters

- **lr** – learning rate of the network
- **momentum** – momentum of the network (if None, then momentum = 1-lr)
- **loss** – loss function applied for optimization

setOutputsMapping(*outputsMapping*, *acc_output=None*)
Sets the mapping of the outputs from the format given by the dataset to the format received by the model.

Parameters

- **outputsMapping** – dictionary with the model outputs' identifiers as keys and the dataset outputs' identifiers as values. If the current model is Sequential then keys must be ints with the desired output order (in this case only one value can be provided). If it is Graph then keys must be str.
- **acc_output** – name of the model's output that will be used for calculating the accuracy of the model (only needed for Graph models)

testNetSamples(*X*, *batch_size=50*)
Applies a forward pass on the samples provided and returns the predicted classes and probabilities.

testNet_deprecated(*ds*, *parameters*, *out_name=None*)
Applies a complete round of tests using the test set in the provided Dataset instance.

Parameters **out_name** – name of the output node that will be used to evaluate the network accuracy. Only applicable for Graph models.

The available (optional) testing parameters are the following ones:

Parameters **batch_size** – size of the batch (number of images) applied on each interation

Data processing parameters

Parameters

- **n_parallel_loaders** – number of parallel data loaders allowed to work at the same time
- **normalization** – boolean indicating if we want to 0-1 normalize the image pixel values
- **mean_substraction** – boolean indicating if we want to substract the training mean

testOnBatch(*X*, *Y*, *accuracy=True*, *out_name=None*)
Applies a test on the samples provided and returns the resulting loss and accuracy (if True).

Parameters **out_name** – name of the output node that will be used to evaluate the network accuracy. Only applicable for Graph models.

trainNet(*ds*, *parameters*, *out_name=None*)
Trains the network on the given dataset 'ds'.

Parameters **out_name** – name of the output node that will be used to evaluate the network accuracy. Only applicable to Graph models.

The input 'parameters' is a dict() which may contain the following (optional) training parameters:

Visualization parameters

Parameters

- **report_iter** – number of iterations between each loss report
- **iter_for_val** – number of interations between each validation test
- **num_iterations_val** – number of iterations applied on the validation dataset for computing the average performance (if None then all the validation data will be tested)

Learning parameters

Parameters

- **n_epochs** – number of epochs that will be applied during training
- **batch_size** – size of the batch (number of images) applied on each interation by the SGD optimization
- **lr_decay** – number of iterations passed for decreasing the learning rate
- **lr_gamma** – proportion of learning rate kept at each decrease. It can also be a set of rules defined by a list, e.g. lr_gamma = [[3000, 0.9], ..., [None, 0.8]] means 0.9 until iteration 3000, ..., 0.8 until the end.

Data processing parameters

Parameters

- **n_parallel_loaders** – number of parallel data loaders allowed to work at the same time
- **normalize** – boolean indicating if we want to 0-1 normalize the image pixel values
- **mean_substraction** – boolean indicating if we want to substract the training mean
- **data_augmentation** – boolean indicating if we want to perform data augmentation (always False on validation)
- **shuffle** – apply shuffling on training data at the beginning of each epoch.

Other parameters

Parameters **save_model** – number of iterations between each model backup

keras_wrapper.cnn_model.**loadModel**(*model_path*, *iter*)

Loads a previously saved Model_Wrapper object.

Parameters

- **model_path** – Path to the Model_Wrapper object to load
- **iter** – Number of iterations

Returns Loadaed Model_Wrapper

keras_wrapper.cnn_model.**saveModel**(*model_wrapper*, *iter*, *path=None*)

Saves a backup of the current Model_Wrapper object after being trained for 'iter' iterations.

Parameters

- **model_wrapper** – Object to save

- **iter** – Number of iterations
- **path** – Oath to save

Returns None

staged_network.py

class keras_wrapper.staged_network.**Staged_Network**(*silence=False*, *model_name=None*, *plots_path=None*, *models_path=None*)

Builds and manages a set of CNN_Model instances that will be trained and executed in ordered stages.

addBranch(*branch*, *stage_id*, *axis=0*, *out_name=None*, *in_name=None*, *training_is_enabled=True*, *expand_dimensions=True*, *balanced=True*)

Adds a new branch to a specific stage.

Parameters

- **stage_id** – id position of the stage where the new branch will be added.
- **axis** – indicates the axis where we want to apply the join of the outputs.
- **out_name** – name of the output node used for evaluation. If stage is a list, then we must provide a list of output identifiers. Only has to be specified for the Graph stages
- **in_name** – name of the previous stage output that will be used for propagation to the next stage. If stage is a list, then we must provide a list of output identifiers. Only has to be specified for the Graph stages
- **training_is_enabled** – indicates if the training is enabled for the input branches. If stage is a list, then input a list of booleans if only some branches' training will be enabled.
- **expand_dimensions** – indicates if we want to expand the dimensions (to 4) of the inputted data to this stage. If stage is a list, then input a list of booleans if only some branches' input dimensions will be expanded.
- **balanced** – indicates if we want to perform a balanced training (equal number of samples per class). If stage is a list, then input a list of booleans if only some branches will be trained in a balanced manner.

addStage(*stage*, *axis=0*, *out_name=None*, *in_name=None*, *reloading_model=False*, *training_is_enabled=True*, *expand_dimensions=True*, *balanced=True*)

Add either a Stage or CNN_Model object to the list of stages. If the current stage is a parallel stage, it must be a list of Stages and the parameter axis must be specified.

Parameters

- **stage** – single Stage instance or multi-stage (parallalel) list of Stage instances.
- **axis** – indicates the axis where we want to apply the join of the outputs of all the branches in this stage. Only has to be specified for the parallel stages.
- **out_name** – name of the output node used for evaluation. If stage is a list, then we must provide a list of output identifiers. Only has to be specified for the Graph stages
- **in_name** – name of the previous stage output that will be used for propagation to the next stage. If stage is a list, then we must provide a list of output identifiers. Only has to be specified for the Graph stages
- **reloading_model** – only should be set to 'True' when the Staged_Network model is being reloaded from memory

- **training_is_enabled** – indicates if the training is enabled for the current stage. If stage is a list, then input a list of booleans if only some branches' training will be enabled.
- **expand_dimensions** – indicates if we want to expand the dimensions (to 4) of the inputted data to this stage. If stage is a list, then input a list of booleans if only some branches' input dimensions will be expanded.
- **balanced** – indicates if we want to perform a balanced training (equal number of samples per class). If stage is a list, then input a list of booleans if only some branches will be trained in a balanced manner.

changeAllNames(*model_name*, *clear_dirs=True*)
Changes all the model names and plot/save locations from all the stages and braches in the current Staged_Network. Only the base 'model_name' must be given, the standard stage/branch names, plot and model locations will be used.

checkParameters(*input_params*, *default_params*)
Validates a set of input parameters and uses the default ones if not specified.

enableTraining(*stage_id*, *training_is_enabled*)
Replaces the trainingIsEnabled list from the selected stage.

forwardUntilStage(*X*, *stage_id*)
Applies a forward pass on all the stages until 'stage_id' (not included).

getJoinOnAxis(*position*)
Returns the axis on a certain position.

getNumStages()
Gets the current number of defined Stages.

getStage(*position*)
Returns the Stage object on a certain position.

popStage()
Removes the last stage on a Staged_Network

predictClassesOnBatch(*X*, *topN=5*)
Applies a forward pass along all the Staged_Network and returns the topN predicted classes sorted.

predictOnBatch(*X*)
Applies a forward pass along all the Staged_Network and returns the predicted values.

removeBranches(*branch_ids*, *stage_id*)
Removes a specific set of branches from a specific stage.

removeWorseClassifiers(*ds*, *stage_id*, *min_accuracy=0.7*, *parameters={}*)
Applies a complete round of tests using the validation ('val') set in the provided Dataset instance for finding the set of recently trained classifiers in 'stage_id' that have not been properly trained. If they do not accomplish a minimum accuracy 'min_accuracy' they are removed.

Parameters **batch_size** – size of the batch (number of images) applied on each interation

Data processing parameters

Parameters

- **n_parallel_loaders** – number of parallel data loaders allowed to work at the same time
- **normalize_images** – boolean indicating if we want to 0-1 normalize the image pixel values
- **mean_substraction** – boolean indicating if we want to substract the training mean

replaceStage (*stage*, *position*)
Replaces a Stage object on a certain position

resumeTrainNet (*ds*, *stage_id*, *parameters={}*)
Resumes the last training state of a stored stage keeping also its training parameters. If we introduce any parameter through the argument 'parameters', it will be replaced by the old one.

setName (*model_name*, *plots_path=None*, *models_path=None*, *clear_dirs=True*)
Changes the name (identifier) of the Staged_Network instance.

testNet (*ds*, *stage_id=None*, *parameters={}*)
Applies a complete round of tests using the test set in the provided Dataset instance. If stage_id=None the test will be applied on the whole staged network. The available (optional) testing parameters are the following ones:

Parameters **batch_size** – size of the batch (number of images) applied on each interation

Data processing parameters

Parameters

- **n_parallel_loaders** – number of parallel data loaders allowed to work at the same time
- **normalize_images** – boolean indicating if we want to 0-1 normalize the image pixel values
- **mean_substraction** – boolean indicating if we want to substract the training mean

trainNet (*ds*, *stage_id*, *parameters={}*)
Trains stage defined by stage_id on the given dataset 'ds'. The available (optional) training parameters are the following ones:

Visualization parameters

Parameters

- **report_iter** – number of iterations between each loss report
- **iter_for_val** – number of interations between each validation test
- **num_iterations_val** – number of iterations applied on the validation dataset for computing the average performance (if None then all the validation data will be tested)

Learning parameters

Parameters

- **n_epochs** – number of epochs that will be applied during training
- **batch_size** – size of the batch (number of images) applied on each interation by the SGD optimization
- **lr_decay** – number of iterations passed for decreasing the learning rate
- **lr_gamma** – proportion of learning rate kept at each decrease. It can also be a set of rules defined by a list, e.g. lr_gamma = [[3000, 0.9], ..., [None, 0.8]] means 0.9 until iteration 3000, ..., 0.8 until the end.

Data processing parameters

Parameters

- **n_parallel_loaders** – number of parallel data loaders allowed to work at the same time

- **normalize_images** – boolean indicating if we want to 0-1 normalize the image pixel values
- **mean_substraction** – boolean indicating if we want to substract the training mean
- **data_augmentation** – boolean indicating if we want to perform data augmentation (always False on validation)

Other parameters

Parameters **save_model** – number of iterations between each model backup

valWorsePairs(*ds*, *n_pairs=5*, *parameters={}*, *avoid_pairs=[]*)

Applies a complete round of tests using the validation ('val') set in the provided Dataset instance for finding the set of 'n_pairs' pairs of classes that have a higher intra-error. The set of pairs of classes provided as tuples (#classA, #classB) will not be selected. The available (optional) testing parameters are the following ones:

Parameters **batch_size** – size of the batch (number of images) applied on each interation

Data processing parameters

Parameters

- **n_parallel_loaders** – number of parallel data loaders allowed to work at the same time
- **normalize_images** – boolean indicating if we want to 0-1 normalize the image pixel values
- **mean_substraction** – boolean indicating if we want to substract the training mean

keras_wrapper.staged_network.**loadStagedModel**(*model_path*)

Loads a previously saved Staged_Network object.

keras_wrapper.staged_network.**saveStagedModel**(*staged_network*, *path=None*)

Saves a backup of the current Staged_Network object.

stage.py

class keras_wrapper.stage.**Stage**(*nInput*, *nOutput*, *input_shape*, *output_shape*, *type='basic_model'*, *silence=False*, *structure_path=None*, *weights_path=None*, *model_name=None*, *plots_path=None*, *models_path=None*)

Class for defining a single stage from a Staged_Network. This class is only intended to be used in conjunction with the Staged_Network class.

applyClassMapping(*Y*)

Returns the corresponding integer identifiers for the current Stage's mapping given a set of categorical arrays Y.

applyMask(*prediction*)

Returns a prediction matrix after applying the defined output mask.

defineClassMapping(*mapping*)

Defines an input mapping from all the classes available in a Staged_Network to the ones used in this particular Stage instance.

Parameters **mapping** – dictionary with all the classes in the Staged_Network as 'keys' and the corresponding mapped inputs to this Stage as 'values'. If some 'key' is not used, its 'value' should be set to None.

defineOutputMask(*mask*)
Defines an output mask for redirecting this stage's output to the following stage once this stage has been trained (on test mode).

Parameters **mask** – dictionary with all the Stage's output values indices as 'keys' and the corresponding mapped indices on the final test's output as 'values'. If some 'key' is not used, its 'value' should be set to None. E.g. if we have output_shape = [1,1] then mask = {'[0]': [0], '[1]': None} If we want to disable the mask, we can set it to None. The mask is disabled by default.

isReadyToTrainOnBatch(*batch_size*, *balanced*)
Checks if the input samples lists have enough samples to train on a batch. In this case returns the first 'batch_size' samples, else returns False.

Parameters

- **batch_size** – number of samples retrieved for the next training batch
- **balanced** – wether we want a set of balanced samples or not

predictOnBatch(*X*, *in_name=None*, *out_name=None*, *expand=False*)
Applies a forward pass and returns the predicted values after being processed by the output mask.

Parameters

- **in_name** – name of the input we are asking for (from the previous stage). Only applicable to Graph models.
- **out_name** – name of the output used for prediction (from the current stage).
- **expand** – indicates if we want to expand the input dimensions to 4

testOnBatch(*X*, *Y*, *accuracy=True*, *out_name=None*)
Applies a test on the samples provided and returns the resulting loss and accuracy (if True). It selects only the samples that are valid for the current Stage (self.mapping[labels] != None).

Parameters **out_name** – name of the output we are asking for. Only applicable to Graph models.

trainOnBatch(*X*, *Y*, *batch_size*, *balanced=True*, *out_name=None*)
Trains the current stage on the last 'batch_size' samples received. It makes sure the mapping of the introduced samples is balanced among all the defined classes. If it isn't it keeps storing them for a latter forward-backward pass.

Parameters

- **X** – numpy array with a set of loaded images ready to use for Keras
- **Y** – numpy array with a set of categorical labels ready to use for keras
- **batch_size** – number of samples that we want to use to train on the next batch
- **balanced** – indicates if we want to train on a set of balanced samples (same number of samples from each class)
- **out_name** – name of the output node that will be used to evaluate the network accuracy. Only applicable for Graph models.

trainOnBatch_DEPRECATED_class_weight(*X*, *Y*, *batch_size*, *balanced=True*, *out_name=None*)
Trains the current stage on the last 'batch_size' samples received. It makes sure the mapping of the introduced samples is balanced among all the defined classes. If it isn't it keeps storing them for a latter forward-backward pass.

Parameters

- **X** – numpy array with a set of loaded images ready to use for Keras
- **Y** – numpy array with a set of categorical labels ready to use for keras
- **batch_size** – number of samples that we want to use to train on the next batch
- **balanced** – indicates if we want to train on a set of balanced samples (same number of samples from each class)
- **out_name** – name of the output node that will be used to evaluate the network accuracy. Only applicable for Graph models.

trainOnBatch_DEPRECATED_lists(*X*, *Y*, *batch_size*, *balanced=True*, *out_name=None*)

Trains the current stage on the last 'batch_size' samples received. It makes sure the mapping of the introduced samples is balanced among all the defined classes. If it isn't it keeps storing them for a latter forward-backward pass.

Parameters

- **X** – numpy array with a set of loaded images ready to use for Keras
- **Y** – numpy array with a set of categorical labels ready to use for keras
- **batch_size** – number of samples that we want to use to train on the next batch
- **balanced** – indicates if we want to train on a set of balanced samples (same number of samples from each class)
- **out_name** – name of the output node that will be used to evaluate the network accuracy. Only applicable for Graph models.

ecoc_classifier.py

class keras_wrapper.ecoc_classifier.**ECOC_Classifier**(*table*, *distance='hamming'*)

Class for defining a classical ECOC classifier.

append(*rows*)

Appends new rows to the ECOC table which will define the code for new classifiers.

dist2sim(*dist*)

Converts a set of distances into similarities

hammingDistance(*X*)

Returns a matrix of dimensions (n_samples, n_classes) with the distance of each sample to each class.

predictOnBatch(*X*, *in_name=None*, *out_name=None*, *expand=False*)

Predicts the classes for a set of samples.

setDistance(*distance*)

Changes the kind of distance used for computing the predictions.

softmax(*similarities*)

Applies the softmax function to a set of similarities for obtaining a normalized vector of probabilities.

testOnBatch(*X*, *Y*, *accuracy=True*, *out_name=None*)

Applies a test on the samples provided and returns the resulting loss and accuracy (if True).

utils.py

keras_wrapper.utils.**prepareGoogleNet_Food101**(*model_wrapper*)
Prepares the GoogleNet model after its conversion from Caffe

keras_wrapper.utils.**prepareGoogleNet_Food101_ECOC_loss**(*model_wrapper*)
Prepares the GoogleNet model for inserting an ECOC structure after removing the last part of the net

keras_wrapper.utils.**prepareGoogleNet_Food101_Stage1**(*model_wrapper*)
Prepares the GoogleNet model for serving as the first Stage of a Staged_Netork

keras_wrapper.utils.**prepareGoogleNet_Stage2**(*stage1*, *stage2*)
Removes the second part of the GoogleNet for inserting it into the second stage.

thread_loader.py

class keras_wrapper.thread_loader.**ThreadDataLoader**(*target*, **args*)
Data loader based on threads (parallel execution)

class keras_wrapper.thread_loader.**ThreadModelLoader**(*target*, **args*)
Model loader based on threads (parallel execution)

keras_wrapper.thread_loader.**retrieveModel**(*path_json*, *path_h5*, *path_pkl*)
Loads a model using a parallel thread.

keras_wrapper.thread_loader.**retrieveXY**(*dataset*, *set_name*, *batchSize*, *normalization*, *meanSubstraction*, *dataAugmentation*)
Retrieves a set of samples from the given dataset and the given set name

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

k

A

B

C

D

E

F

G

S

T

U

V