TDT4900-Master-Thesis

1.0

Generated by Doxygen 1.8.18

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 beam Namespace Reference

Classes

- class Beam

5.2 build_features Namespace Reference

Variables

- ROOT_PATH = Path(__file__).absolute().parents[2]
- tuple DIMENSIONS = (299, 299, 3)
- parser = argparse.ArgumentParser()
- type
- bool
- default
- help
- nargs
- str
- int
- args = vars(parser.parse_args())
- dataset_ = args['dataset']
- new_dims_ = args['new_image_size']
- tuple dims = DIMENSIONS[:2]
- raw_path = ROOT_PATH.joinpath('data', 'raw')
- interim_path = ROOT_PATH.joinpath('data', 'interim')
- processed_path = ROOT_PATH.joinpath('data', 'processed')
- image_path_ = raw_path.joinpath(dataset_, 'Images')
- save_path_ = interim_path.joinpath(dataset_, 'Images')
- output_layer_dim_ = args['output_layer_idx']
- vis_att_ = args['visual_attention']
- split_set_path_ = interim_path.joinpath(dataset_, dataset_ + '_full.csv')
- vis_att

5.2.1 Variable Documentation

5.2.1.1 args

```
build_features.args = vars(parser.parse_args())
```

Definition at line 37 of file build_features.py.

5.2.1.2 bool

```
build_features.bool
```

Definition at line 20 of file build_features.py.

5.2.1.3 dataset_

```
build_features.dataset_ = args['dataset']
```

Definition at line 39 of file build_features.py.

5.2.1.4 default

```
build_features.default
```

Definition at line 20 of file build_features.py.

5.2.1.5 DIMENSIONS

```
tuple build_features.DIMENSIONS = (299, 299, 3)
```

Definition at line 11 of file build_features.py.

5.2.1.6 dims

```
build_features.dims = DIMENSIONS[:2]
```

Definition at line 41 of file build_features.py.

5.2.1.7 help

```
build_features.help
```

Definition at line 21 of file build_features.py.

5.2.1.8 image_path_

```
build_features.image_path_ = raw_path.joinpath(dataset_, 'Images')
```

Definition at line 53 of file build_features.py.

5.2.1.9 int

```
build_features.int
```

Definition at line 35 of file build_features.py.

5.2.1.10 interim_path

```
build_features.interim_path = ROOT_PATH.joinpath('data', 'interim')
```

Definition at line 50 of file build_features.py.

5.2.1.11 nargs

```
build_features.nargs
```

Definition at line 24 of file build_features.py.

5.2.1.12 new_dims_

```
build_features.new_dims_ = args['new_image_size']
```

Definition at line 40 of file build_features.py.

5.2.1.13 output_layer_dim_

```
build_features.output_layer_dim_ = args['output_layer_idx']
```

Definition at line 58 of file build_features.py.

5.2.1.14 parser

```
build_features.parser = argparse.ArgumentParser()
```

Definition at line 19 of file build_features.py.

5.2.1.15 processed_path

```
build_features.processed_path = ROOT_PATH.joinpath('data', 'processed')
```

Definition at line 51 of file build_features.py.

5.2.1.16 raw_path

```
build_features.raw_path = ROOT_PATH.joinpath('data', 'raw')
```

Definition at line 49 of file build_features.py.

5.2.1.17 ROOT_PATH

```
build_features.ROOT_PATH = Path(__file__).absolute().parents[2]
```

Definition at line 8 of file build_features.py.

5.2.1.18 save_path_

```
build_features.save_path_ = interim_path.joinpath(dataset_, 'Images')
```

Definition at line 54 of file build_features.py.

5.2.1.19 split_set_path_

```
build_features.split_set_path_ = interim_path.joinpath(dataset_, dataset_ + '_full.csv')
```

Definition at line 65 of file build_features.py.

5.2.1.20 str

```
build_features.str
```

Definition at line 27 of file build_features.py.

5.2.1.21 type

```
build_features.type
```

Definition at line 20 of file build_features.py.

5.2.1.22 vis_att

```
build_features.vis_att
```

Definition at line 71 of file build_features.py.

5.2.1.23 vis_att_

```
build_features.vis_att_ = args['visual_attention']
```

Definition at line 59 of file build_features.py.

5.3 custom_layers Namespace Reference

Classes

- class AttentionLayer
- class ImageEncoder
- class MultimodalDecoder
- class SentinelLSTM

5.4 data_cleaning Namespace Reference

Functions

- def basic_data_cleaning (df_path, save_path, voc_save_path)
- def replace_uncommon_words (caption_df, corpus)
- def remove_too_many_unk (caption_df)
- def remove_bad_captions (caption_df)
- def is_all_one_letter (caption, letter)
- def number_to_word (word)

Variables

- ROOT_PATH = Path(__file__).absolute().parents[2]
- int THRESHOLD = 3
- float UNK_PERCENTAGE = 0.4

5.4.1 Function Documentation

5.4.1.1 basic_data_cleaning()

```
def data_cleaning.basic_data_cleaning (
              df_path,
              save_path,
              voc_save_path )

Basic data cleaning entails:
- Converting to lower.
- Removing punctuation.
- Converting numbers to number words and removing large numbers.
- Removing one letter words that are not 'a'.
- Replacing uncommon words with UNK token.
- Removing captions with too many UNK tokens.
- Removing weired mmmmm mm mmm captions.

Parameters
----------
df_path : Path or str. Where the .csv file containing unprocessed
    cases is located.
save_path : Path or str. Where the new .csv file containing
    pre-processed cases will be saved.
voc_save_path : Path or str. Where the vocabulary will be saved.

Returns
-------
Saves cleaned dataset at save_path. saves vocabulary at
voc_save_path.
```

Definition at line 12 of file data_cleaning.py.

```
12 def basic_data_cleaning(df_path, save_path, voc_save_path):
13     """
14     Basic data cleaning entails:
15     - Converting to lower.
16     - Removing punctuation.
17     - Converting numbers to number words and removing large numbers.
18     - Removing one letter words that are not 'a'.
19     - Replacing uncommon words with UNK token.
20     - Removing captions with too many UNK tokens.
21     - Removing weired mmmmm mm mmm captions.
22 
23     Parameters
24     ----------
25     df_path : Path or str. Where the .csv file containing unprocessed
26         cases is located.
27     save_path : Path or str. Where the new .csv file containing
28         pre-processed cases will be saved.
29     voc_save_path : Path or str. Where the vocabulary will be saved.
30 
31     Returns
32     -------
33     Saves cleaned dataset at save_path. saves vocabulary at
34     voc_save_path.
35     """
36     df_path = Path(df_path)
37     save_path = Path(save_path)
38     voc_save_path = Path(voc_save_path)
39     caption_df = pd.read_csv(df_path)
40     # make a new column for the cleaned version of the caption
41     # I do this because I want to keep the original version
42     caption_df['clean_caption'] = ''
43     # corpus of words, {word: occurrence in corpus}
44     corpus = {}
45     table = str.maketrans('', '', string.punctuation)
46     for i in range(len(caption_df)):
47         cap_tokens = caption_df.loc[i, 'caption'].split()
48         # convert to lower
49         cap_tokens = [word.lower() for word in cap_tokens]
50         # remove punctuation from each token
51         cap_tokens = [word.translate(table) for word in cap_tokens]
52         # converting numbers to number words and remove large numbers
53         cap_tokens = [word if word.isalpha() else number_to_word(word)
54                       for word in cap_tokens]
55 
56         # remove hanging 's' and other possible weired one letter words
57         cap_tokens = [word for word in cap_tokens
58                       if len(word) > 1 or word == 'a']
59         # add words to corpus
60         for word in cap_tokens:
61             if word not in corpus.keys():
62                 corpus[word] = 1
63             else:
64                 corpus[word] += 1
65 
66         # add the cleaned caption to df
67         caption_df.at[i, 'clean_caption'] = ' '.join(cap_tokens)
68     print("Full vocabulary size:", len(corpus))
69 
70     replace_corpus, caption_df = replace_uncommon_words(caption_df, corpus)
71 
72     # remove captions with more than 40% UNK tokens in captions
73     count_before = len(caption_df)
74     caption_df, remove_caps_len = remove_too_many_unk(caption_df)
75     count_after = len(caption_df)
76     if count_after != count_before - remove_caps_len:
77         print("Did not remove 40% UNK captions!!"
78               "\nOr something else went wrong.")
79 
80     # remove bad mm mmmmm mmmm captions if they still exist
81     count_before = count_after
82     caption_df, remove_caps_len = remove_bad_captions(caption_df)
83     count_after = len(caption_df)
84     if count_after != count_before - remove_caps_len:
85         print("Did not remove bad mmmm captions!!"
86               "\nOr something else went wrong.")
87 
88     print('vocabulary size that will be used:',
89           len(corpus) - len(replace_corpus))
90 
91     # check save path
92     if not save_path.parent.is_dir():
93         save_path.parent.mkdir(parents=True)
94 
95     # save captions
96     caption_df.to_csv(save_path)
97 
```

```
98      # vocabulary stuff
99      vocabulary = [key for key in corpus.keys() if corpus[key] >= THRESHOLD]
100      vocabulary.append('endseq')
101      # add startseq last so that we can remove it easily from models vocabulary
102      vocabulary.append('startseq')
103
104      # check voc save path
105      if not voc_save_path.parent.is_dir():
106          voc_save_path.parent.mkdir(parents=True)
107
108      with open(voc_save_path, 'w') as voc_file:
109          for word in vocabulary:
110              voc_file.write(word + '\n')
111
112
```

5.4.1.2 is_all_one_letter()

```
def data_cleaning.is_all_one_letter (
                caption,
                letter )
```

Definition at line 181 of file data_cleaning.py.

```
181 def is_all_one_letter(caption, letter):
182     tmp_caption = caption[1:len(caption) - 1]
183     for word in tmp_caption:
184         for char in word:
185             if char != letter:
186                 return False
187     return True
188
189
```

5.4.1.3 number_to_word()

```
def data_cleaning.number_to_word (
                word )
```

Definition at line 190 of file data_cleaning.py.

```
190 def number_to_word(word):
191     numbers = {'0': 'zero', '1': 'one', '2': 'two', '3': 'three', '4': 'four',
192                '5': 'five', '6': 'six', '7': 'seven', '8': 'eight',
193                '9': 'nine'}
194     if word in numbers.keys():
195         return numbers[word]
196     return ''
```

5.4.1.4 remove_bad_captions()

```
def data_cleaning.remove_bad_captions (
                caption_df )
```

```
Remove objectively bad captions that do not contain real words.
Here we only remove captions that only consists of the letter m,
since our exploration found that such cases exist in MS COCO.

Parameters
----------
caption_df : DataFrame. contains pre-processed cases.

Returns
-------
caption_df : DataFrame. Where bad captions have been removed.
remove_caps : list. caption_ids of the captions that were removed.
```

Definition at line 154 of file data_cleaning.py.

```
154 def remove_bad_captions(caption_df):
155     """
156     Remove objectively bad captions that do not contain real words.
157     Here we only remove captions that only consists of the letter m,
158     since our exploration found that such cases exist in MS COCO.
159
160     Parameters
161     ----------
162     caption_df : DataFrame. contains pre-processed cases.
163
164     Returns
165     -------
166     caption_df : DataFrame. Where bad captions have been removed.
167     remove_caps : list. caption_ids of the captions that were removed.
168     """
169     remove_caps = []
170     for i in range(len(caption_df)):
171         caption = caption_df.loc[i, 'clean_caption'].split()
172         if is_all_one_letter(caption, 'm'):
173             print("adds mmmmm mmmm to remove")
174             remove_caps.append(caption_df.loc[i, 'caption_id'])
175     caption_df = caption_df.loc[
176                 ~caption_df.loc[:, 'caption_id'].isin(remove_caps), :]
177     caption_df = caption_df.reset_index(drop=True)
178     return caption_df, len(remove_caps)
179
180
```

5.4.1.5 remove_too_many_unk()

```
def data_cleaning.remove_too_many_unk (
              caption_df )
```

```
Remove captions with too many UNK tokens in the caption.

Parameters
----------
caption_df : DataFrame. contains pre-processed cases.

Returns
-------
caption_df : DataFrame. Where captions with too many UNKs have
    been removed.
remove_caps : list. caption_ids of the captions that were removed.
```

Definition at line 126 of file data_cleaning.py.

```
126 def remove_too_many_unk(caption_df):
127     """
128     Remove captions with too many UNK tokens in the caption.
129
130     Parameters
131     ----------
132     caption_df : DataFrame. contains pre-processed cases.
133
134     Returns
```

```
135     -------
136     caption_df : DataFrame. Where captions with too many UNKs have
137         been removed.
138     remove_caps : list. caption_ids of the captions that were removed.
139     """
140     remove_caps = []
141     for i in range(len(caption_df)):
142         caption = caption_df.loc[i, 'clean_caption'].split()
143         length = len(caption)
144         unks = sum([1 if w == 'UNK' else 0 for w in caption])
145         if unks / length > UNK_PERCENTAGE:
146             print("removes too many UNKs")
147             remove_caps.append(caption_df.loc[i, 'caption_id'])
148     caption_df = caption_df.loc[
149                  ~caption_df.loc[:, 'caption_id'].isin(remove_caps), :]
150     caption_df = caption_df.reset_index(drop=True)
151     return caption_df, len(remove_caps)
152
153
```

5.4.1.6 replace_uncommon_words()

```
def data_cleaning.replace_uncommon_words (
                caption_df,
                corpus )
```

Definition at line 113 of file data_cleaning.py.

```
113 def replace_uncommon_words(caption_df, corpus):
114     # replace words with less than THRESHOLD occurrences with an UNK
115     # token
116     replace_corpus = set([key for key in corpus if corpus[key] < THRESHOLD])
117     for i in range(len(caption_df)):
118         cap_tokens = caption_df.loc[i, 'clean_caption'].split()
119         cap_tokens = [word if word not in replace_corpus else 'UNK'
120                       for word in cap_tokens]
121         cap_tokens = ['startseq'] + cap_tokens + ['endseq']
122         caption_df.at[i, 'clean_caption'] = ' '.join(cap_tokens)
123     return replace_corpus, caption_df
124
125
```

5.4.2 Variable Documentation

5.4.2.1 ROOT_PATH

```
data_cleaning.ROOT_PATH = Path(__file__).absolute().parents[2]
```

Definition at line 6 of file data_cleaning.py.

5.4.2.2 THRESHOLD

```
int data_cleaning.THRESHOLD = 3
```

Definition at line 8 of file data_cleaning.py.

5.4.2.3 UNK_PERCENTAGE

```
float data_cleaning.UNK_PERCENTAGE = 0.4
```

Definition at line 9 of file data_cleaning.py.

5.5 data_generator Namespace Reference

Functions

- def data_generator (data_df, batch_size, steps_per_epoch, wordtoix, features, seed=2222)
- def get_image (visual_features, data_df, i)
- def get_caption (data_df, i)
- def to_categorical (y, num_classes=None, dtype='float32')
- def pad_sequences (sequences, maxlen)

Variables

- ROOT_PATH = Path(__file__).absolute().parents[2]

5.5.1 Function Documentation

5.5.1.1 data_generator()

```
def data_generator.data_generator (
               data_df,
               batch_size,
               steps_per_epoch,
               wordtoix,
               features,
               seed = 2222 )
```

```
outputs data in batches
```

Definition at line 10 of file data_generator.py.

```
10 def data_generator(data_df, batch_size, steps_per_epoch,
11                     wordtoix, features, seed=2222):
12     """
13     outputs data in batches
14     """
15     # TODO: order data to create batches with captions
16     #  of roughly the same length
17     r.seed(seed)
18
19     # load visual features
20     max_length = max([len(c.split())
21                       for c in set(data_df.loc[:, 'clean_caption'])])
22     # infinite loop
23     while True:
24         # new Epoch have started
25         # new shuffle state for next epoch
26         shuffle_state = r.randint(0, 10000)
```

```
27         # shuffle df
28         data_df = shuffle(data_df, random_state=shuffle_state)
29         data_df = data_df.reset_index(drop=True)
30         for step in range(steps_per_epoch):
31             # create a new batch
32             x1 = []
33             x2 = []
34             caption_lengths = []
35             # Steps per epoch is equal to floor of
36             # total_samples/batch_size
37             # TODO: make steps per epoch equal to ceiling of
38             # TODO (continued): total_samples/batch_size
39             for i in range(batch_size * step, batch_size * (step + 1)):
40                 image = get_image(features, data_df, i)
41                 caption = get_caption(data_df, i)
42                 # create caption number sequence
43                 seq = [wordtoix[word] for word in caption.split(' ')
44                        if word in wordtoix]
45                 caption_lengths.append(len(seq))
46
47                 x1.append(image)
48                 x2.append(torch.tensor(seq))
49
50             # pad input sequence
51             x2 = pad_sequences(x2, max_length)  # output is a tensor
52
53             x1 = torch.tensor(x1)  # convert to tensor
54             caption_lengths = np.array(caption_lengths)  # convert to array
55             yield [[x1, x2], caption_lengths]
56
57
```

5.5.1.2 get_caption()

```
def data_generator.get_caption (
                data_df,
                i )
```

Definition at line 63 of file data_generator.py.

```
63 def get_caption(data_df, i):
64     return data_df.loc[i, 'clean_caption']
65
66
```

5.5.1.3 get_image()

```
def data_generator.get_image (
                visual_features,
                data_df,
                i )
```

Definition at line 58 of file data_generator.py.

```
58 def get_image(visual_features, data_df, i):
59     image_name = data_df.loc[i, 'image_name']
60     return visual_features[image_name]
61
62
```

5.5.1.4 pad_sequences()

```
def data_generator.pad_sequences (
                sequences,
                maxlen )
```

Definition at line 84 of file data_generator.py.

```
84 def pad_sequences(sequences, maxlen):
85     num = len(sequences)
86     out_dims = (num, maxlen)
87     out_tensor = sequences[0].data.new(*out_dims).fill_(0)
88     for i, tensor in enumerate(sequences):
89         length = tensor.size(0)
90         out_tensor[i, :length] = tensor
91     return out_tensor
92
```

5.5.1.5 to_categorical()

```
def data_generator.to_categorical (
                y,
                num_classes = None,
                dtype = 'float32' )
```

Definition at line 67 of file data_generator.py.

```
67 def to_categorical(y, num_classes=None, dtype='float32'):
68     # copied from keras for convenience
69     y = np.array(y, dtype='int')
70     input_shape = y.shape
71     if input_shape and input_shape[-1] == 1 and len(input_shape) > 1:
72         input_shape = tuple(input_shape[:-1])
73     y = y.ravel()
74     if not num_classes:
75         num_classes = np.max(y) + 1
76     n = y.shape[0]
77     categorical = np.zeros((n, num_classes), dtype=dtype)
78     categorical[np.arange(n), y] = 1
79     output_shape = input_shape + (num_classes,)
80     categorical = np.reshape(categorical, output_shape)
81     return categorical
82
83
```

5.5.2 Variable Documentation

5.5.2.1 ROOT_PATH

```
data_generator.ROOT_PATH = Path(__file__).absolute().parents[2]
```

Definition at line 7 of file data_generator.py.

5.6 generator_framework Namespace Reference

Classes

- class Generator

Functions

- def loss_switcher (loss_string)
- def optimizer_switcher (optimizer_string)

Variables

- ROOT_PATH = Path(__file__).absolute().parents[2]

5.6.1 Function Documentation

5.6.1.1 loss_switcher()

```
def generator_framework.loss_switcher (
              loss_string )
```

Definition at line 28 of file generator_framework.py.

```
28 def loss_switcher(loss_string):
29     loss_string = loss_string.lower()
30     switcher = {
31         'cross_entropy': nn.CrossEntropyLoss,
32         'mse': nn.MSELoss
33     }
34
35     return switcher.get(loss_string, nn.CrossEntropyLoss)
36
37
```

5.6.1.2 optimizer_switcher()

```
def generator_framework.optimizer_switcher (
              optimizer_string )
```

Definition at line 38 of file generator_framework.py.

```
38 def optimizer_switcher(optimizer_string):
39     optimizer_string = optimizer_string.lower()
40     switcher = {
41         'adam': optim.Adam,
42         'sgd': optim.SGD
43     }
44     return switcher.get(optimizer_string, optim.SGD)
45
46
```

5.6.2 Variable Documentation

5.6.2.1 ROOT_PATH

```
generator_framework.ROOT_PATH = Path(__file__).absolute().parents[2]
```

Definition at line 25 of file generator_framework.py.

5.7 glove_embeddings Namespace Reference

Functions

- def load_glove_vectors (glove_path)
- def embeddings_matrix (vocab_size, wordtoix, embeddings_index, embedding_dim)

5.7.1 Function Documentation

5.7.1.1 embeddings_matrix()

```
def glove_embeddings.embeddings_matrix (
                vocab_size,
                wordtoix,
                embeddings_index,
                embedding_dim )
```

Definition at line 16 of file glove_embeddings.py.

```
16 def embeddings_matrix(vocab_size, wordtoix, embeddings_index, embedding_dim):
17     embedding_matrix = np.zeros((vocab_size, embedding_dim))
18     for word, i in wordtoix.items():
19         # if i < max_words:
20         embedding_vector = embeddings_index.get(word)
21         if embedding_vector is not None:
22             # Words not found in the embedding index will be all zeros
23             embedding_matrix[i] = embedding_vector
24     return embedding_matrix
```

5.7.1.2 load_glove_vectors()

```
def glove_embeddings.load_glove_vectors (
                glove_path )
```

Definition at line 4 of file glove_embeddings.py.

```
4 def load_glove_vectors(glove_path):
5     # Load Glove vectors
6     embeddings_index = {}  # empty dictionary
7     with open(glove_path, encoding='utf-8') as f:
8         for line in f:
9             values = line.split()
10             word = values[0]
11             coefs = np.asarray(values[1:], dtype='float32')
12             embeddings_index[word] = coefs
13     return embeddings_index
14
15
```

5.8 handle_karpathy_split Namespace Reference

Functions

- def order_raw_data_and_move_to_interim (data_path, dataset, ann_path)
- def initialize_full_dict ()
- def initialize_ann_dict ()

Variables

- ROOT_PATH = Path(__file__).absolute().parents[2]

5.8.1 Function Documentation

5.8.1.1 initialize_ann_dict()

```
def handle_karpathy_split.initialize_ann_dict ( )
```

Definition at line 134 of file handle_karpathy_split.py.

```
134 def initialize_ann_dict():
135     init_dict = {
136         "info": {
137             "description": "description",
138             "url": "www",
139             "version": 1.0,
140             "year": 2020,
141             "contributor": "contributor",
142             "date_created": "yyyy-mm-dd hh:mm:ss"
143         },
144         "images": [],
145         "licenses": [],
146         "type": "captions",
147         "annotations": []
148     }
149     ann_dict = {
150         "test": {},
151         "val": {},
152         "train": {}
153     }
154     for key in ann_dict:
155         ann_dict[key] = deepcopy(init_dict)
156     return ann_dict
```

5.8.1.2 initialize_full_dict()

```
def handle_karpathy_split.initialize_full_dict ( )
```

Definition at line 115 of file handle_karpathy_split.py.

```
115 def initialize_full_dict():
116     train_dict = {}
117     # val and test does not need caption_id nor caption columns
118     test_dict = {
119         'image_id': [],
120         'image_name': [],
121     }
122     val_dict = {
123         'image_id': [],
124         'image_name': [],
125     }
126     columns = ['image_id', 'image_name', 'caption_id', 'caption']
127     test_columns = columns[:2]
128     for col in columns:
129         train_dict[col] = []
130     full_dict = {'train': train_dict, 'val': val_dict, 'test': test_dict}
131     return full_dict, columns, test_columns
132
133
```

5.8.1.3 order_raw_data_and_move_to_interim()

```
def handle_karpathy_split.order_raw_data_and_move_to_interim (
                data_path,
                dataset,
                ann_path )
```

Definition at line 9 of file handle_karpathy_split.py.

```
9 def order_raw_data_and_move_to_interim(data_path, dataset, ann_path):
10     data_path = Path(data_path)
11     ann_path = Path(ann_path)
12     with open(data_path, 'r') as json_file:
13         data_dict = json.load(json_file)
14
15     full_dict, columns, test_columns = initialize_full_dict()
16
17     ann_dict = initialize_ann_dict()
18
19     ann_images = {
20         "test": [],
21         "val": [],
22         "train": []
23     }
24     ann_annotations = {
25         "test": [],
26         "val": [],
27         "train": []
28     }
29
30     images = data_dict['images']
31
32     for image in images:
33         ann_image_obj = {
34             "license": 0,
35             "url": "www",
36             "width": 0,
37             "height": 0,
38             "date_captured": "yyyy-mm-dd hh:mm:ss"
39         }
40
41         # get image name
42         image_name = image['filename']
43         image_id = image['imgid']
44
45         # add info to ann_image_obj
46         ann_image_obj['file_name'] = image_name
47         ann_image_obj['id'] = image_id
48
49         # get split
50         split = image['split']
51
52         if split in {"test", "val"}:
53             # save ann_image_obj
54             ann_images[split].append(ann_image_obj)
55             # save image info to .csv files
56             full_dict[split]['image_id'].append(image_id)
57             full_dict[split]['image_name'].append(image_name)
58         # go through captions and add them to dict split
59         captions = image['sentences']
60         sentids = image['sentids']
61         for sentid, caption in zip(sentids, captions):
62             ann_cap_obj = {
63                 "id": sentid,
64                 "image_id": image_id,
65                 "caption": caption['raw']
66             }
67
68             raw_caption = caption['raw']
69             caption_id = image_name[:-4] + '#' + str(sentid)
70             # add info to dicts
71             if split in {'restval', 'train'}:
72                 full_dict['train']['image_id'].append(image_id)
73                 full_dict['train']['image_name'].append(image_name)
74                 full_dict['train']['caption_id'].append(caption_id)
75                 full_dict['train']['caption'].append(raw_caption)
76                 # save ann_cap_obj
77                 ann_annotations['train'].append(ann_cap_obj)
78             else:
79                 # save ann_cap_obj
80                 ann_annotations[split].append(ann_cap_obj)
81
82     if not ann_path.is_dir():
83         # if this is not a directory then make it, including every
```

```
84          # parent that may be missing
85          ann_path.mkdir(parents=True)
86
87      for s in ["test", "val", "train"]:
88          ann_dict[s]['images'] = ann_images[s]
89          ann_dict[s]['annotations'] = ann_annotations[s]
90          # save dict split as .json file
91          with open(ann_path.joinpath(dataset + '_' + s + '.json'), 'w') \
92                  as ann_file:
93              json.dump(ann_dict[s], ann_file)
94
95      train_df = pd.DataFrame(full_dict['train'], columns=columns)
96      test_df = pd.DataFrame(full_dict['test'], columns=test_columns)
97      val_df = pd.DataFrame(full_dict['val'], columns=test_columns)
98
99      save_path = ROOT_PATH.joinpath('data', 'interim', 'karpathy_split')
100      if not save_path.is_dir():
101          # if dir does not exist create dir
102          # create parents if they do not exist either
103          save_path.mkdir(parents=True)
104
105      train_file = save_path.joinpath(dataset + '_train.csv')
106      test_file = save_path.joinpath(dataset + '_test.csv')
107      val_file = save_path.joinpath(dataset + '_val.csv')
108
109      train_df.to_csv(train_file)
110      test_df.to_csv(test_file)
111      val_df.to_csv(val_file)
112      print("finished job!!")
113
114
```

5.8.2 Variable Documentation

5.8.2.1 ROOT_PATH

```
handle_karpathy_split.ROOT_PATH = Path(__file__).absolute().parents[2]
```

Definition at line 6 of file handle_karpathy_split.py.

5.9 make_dataset Namespace Reference

Variables

- ROOT_PATH = Path(__file__).absolute().parents[2]
- parser = argparse.ArgumentParser()
- type
- str
- default
- help
- bool
- args = vars(parser.parse_args())
- raw_path = ROOT_PATH.joinpath('data', 'raw')
- interim_path = ROOT_PATH.joinpath('data', 'interim')
- processed_path = ROOT_PATH.joinpath('data', 'processed')
- ann_path_ = processed_path.joinpath('annotations')
- dataset_ = args['dataset']
- data_path_ = raw_path.joinpath('dataset_' + dataset_ + '.json')
- output_path_ = interim_path.joinpath(dataset_)
- list splits_ = ['train', 'val']
- df_path_ = interim_path.joinpath(dataset_ + '_train.csv')
- save_path_ = interim_path.joinpath(dataset_ + '_train_clean.csv')
- voc_save_path_ = interim_path.joinpath(dataset_ + '_vocabulary.csv')

5.9.1 Variable Documentation

5.9.1.1 ann_path_

```
make_dataset.ann_path_ = processed_path.joinpath('annotations')
```

Definition at line 29 of file make_dataset.py.

5.9.1.2 args

```
make_dataset.args = vars(parser.parse_args())
```

Definition at line 23 of file make_dataset.py.

5.9.1.3 bool

```
make_dataset.bool
```

Definition at line 20 of file make_dataset.py.

5.9.1.4 data_path_

```
make_dataset.data_path_ = raw_path.joinpath('dataset_' + dataset_ + '.json')
```

Definition at line 35 of file make_dataset.py.

5.9.1.5 dataset_

```
make_dataset.dataset_ = args['dataset']
```

Definition at line 30 of file make_dataset.py.

5.9.1.6 default

```
make_dataset.default
```

Definition at line 17 of file make_dataset.py.

5.9.1.7 df_path_

```
make_dataset.df_path_ = interim_path.joinpath(dataset_ + '_train.csv')
```

Definition at line 46 of file make_dataset.py.

5.9.1.8 help

```
make_dataset.help
```

Definition at line 18 of file make_dataset.py.

5.9.1.9 interim_path

```
make_dataset.interim_path = ROOT_PATH.joinpath('data', 'interim')
```

Definition at line 27 of file make_dataset.py.

5.9.1.10 output_path_

```
make_dataset.output_path_ = interim_path.joinpath(dataset_)
```

Definition at line 40 of file make_dataset.py.

5.9.1.11 parser

```
make_dataset.parser = argparse.ArgumentParser()
```

Definition at line 16 of file make_dataset.py.

5.9.1.12 processed_path

```
make_dataset.processed_path = ROOT_PATH.joinpath('data', 'processed')
```

Definition at line 28 of file make_dataset.py.

5.9.1.13 raw_path

```
make_dataset.raw_path = ROOT_PATH.joinpath('data', 'raw')
```

Definition at line 26 of file make_dataset.py.

5.9.1.14 ROOT_PATH

```
make_dataset.ROOT_PATH = Path(__file__).absolute().parents[2]
```

Definition at line 8 of file make_dataset.py.

5.9.1.15 save_path_

```
make_dataset.save_path_ = interim_path.joinpath(dataset_ + '_train_clean.csv')
```

Definition at line 47 of file make_dataset.py.

5.9.1.16 splits_

```
list make_dataset.splits_ = ['train', 'val']
```

Definition at line 41 of file make_dataset.py.

5.9.1.17 str

```
make_dataset.str
```

Definition at line 17 of file make_dataset.py.

5.9.1.18 type

```
make_dataset.type
```

Definition at line 17 of file make_dataset.py.

5.9.1.19 voc_save_path_

```
make_dataset.voc_save_path_ = interim_path.joinpath(dataset_ + '_vocabulary.csv')
```

Definition at line 48 of file make_dataset.py.

5.10 predict_model Namespace Reference

Variables

- ROOT_PATH = Path(__file__).absolute().parents[2]
- parser = argparse.ArgumentParser()
- type
- bool
- default
- help
- str
- required
- int
- args = vars(parser.parse_args())
- interim_path = ROOT_PATH.joinpath('data', 'interim')
- processed_path = ROOT_PATH.joinpath('data', 'processed')
- ann_path = processed_path.joinpath('annotations')
- models_path = ROOT_PATH.joinpath('models')
- model_dir = models_path.joinpath(args['model'])
- split_ = args['split']
- dataset_ = args['dataset']
- train_path = interim_path.joinpath(dataset_, dataset_ + '_train_clean.csv')
- test_path
- voc_path_ = interim_path.joinpath(dataset_, dataset_ + '_vocabulary.csv')
- feature_path_
- saved_model_path_ = model_dir.joinpath('BEST_checkpoint.pth.tar')
- model_name_ = args['model_name']
- save_path_ = models_path
- int em_dim = 300
- int hidden_shape_ = 50
- string loss_function_ = 'cross_entropy'
- string opt = 'adam'
- float lr_ = 0.0001
- int seed_ = 222
- max_length = max_length_caption(train_path)
- list input_shape_ = [[8, 8, 1536], max_length]
- generator
- beam_size_ = args['beam_size']
- val_batch_size = args['val_batch_size']
- res_file = model_dir.joinpath('TEST_' + split_ + '_result.json')
- eval_file = model_dir.joinpath('TEST_' + split_ + '_eval.json')
- annFile = ann_path.joinpath(dataset_ + '_' + split_ + '.json')
- result

5.10.1 Variable Documentation

5.10.1.1 ann_path

```
predict_model.ann_path = processed_path.joinpath('annotations')
```

Definition at line 43 of file predict_model.py.

5.10.1.2 annFile

```
predict_model.annFile = ann_path.joinpath(dataset_ + '_' + split_ + '.json')
```

Definition at line 98 of file predict_model.py.

5.10.1.3 args

```
predict_model.args = vars(parser.parse_args())
```

Definition at line 39 of file predict_model.py.

5.10.1.4 beam_size_

```
predict_model.beam_size_ = args['beam_size']
```

Definition at line 89 of file predict_model.py.

5.10.1.5 bool

```
predict_model.bool
```

Definition at line 15 of file predict_model.py.

5.10.1.6 dataset_

```
predict_model.dataset_ = args['dataset']
```

Definition at line 52 of file predict_model.py.

5.10.1.7 default

```
predict_model.default
```

Definition at line 15 of file predict_model.py.

5.10.1.8 em_dim

```
int predict_model.em_dim = 300
```

Definition at line 68 of file predict_model.py.

5.10.1.9 eval_file

```
predict_model.eval_file = model_dir.joinpath('TEST_' + split_ + '_eval.json')
```

Definition at line 94 of file predict_model.py.

5.10.1.10 feature_path_

```
predict_model.feature_path_
```

Initial value:

```
1 =  processed_path.joinpath(
2         dataset_, 'Images', 'encoded_visual_attention_full.pkl')
```

Definition at line 60 of file predict_model.py.

5.10.1.11 generator

```
predict_model.generator
```

Initial value:

```
1 =  Generator(model_name_, input_shape_, hidden_shape_,
2                           voc_path_, feature_path_,
3                           save_path_,
4                           loss_function=loss_function_,
5                           optimizer=opt, lr=lr_,
6                           embedding_size=em_dim,
7                           seed=seed_)
```

Definition at line 79 of file predict_model.py.

5.10.1.12 help

```
predict_model.help
```

Definition at line 16 of file predict_model.py.

5.10.1.13 hidden_shape_

```
int predict_model.hidden_shape_ = 50
```

Definition at line 69 of file predict_model.py.

5.10.1.14 input_shape_

```
list predict_model.input_shape_ = [[8, 8, 1536], max_length]
```

Definition at line 77 of file predict_model.py.

5.10.1.15 int

```
predict_model.int
```

Definition at line 30 of file predict_model.py.

5.10.1.16 interim_path

```
predict_model.interim_path = ROOT_PATH.joinpath('data', 'interim')
```

Definition at line 41 of file predict_model.py.

5.10.1.17 loss_function_

```
string predict_model.loss_function_ = 'cross_entropy'
```

Definition at line 70 of file predict_model.py.

5.10.1.18 lr_

```
float predict_model.lr_ = 0.0001
```

Definition at line 72 of file predict_model.py.

5.10.1.19 max_length

```
predict_model.max_length = max_length_caption(train_path)
```

Definition at line 76 of file predict_model.py.

5.10.1.20 model_dir

```
predict_model.model_dir = models_path.joinpath(args['model'])
```

Definition at line 45 of file predict_model.py.

5.10.1.21 model_name_

```
predict_model.model_name_ = args['model_name']
```

Definition at line 64 of file predict_model.py.

5.10.1.22 models_path

```
predict_model.models_path = ROOT_PATH.joinpath('models')
```

Definition at line 44 of file predict_model.py.

5.10.1.23 opt

```
string predict_model.opt = 'adam'
```

Definition at line 71 of file predict_model.py.

5.10.1.24 parser

```
predict_model.parser = argparse.ArgumentParser()
```

Definition at line 14 of file predict_model.py.

5.10.1.25 processed_path

```
predict_model.processed_path = ROOT_PATH.joinpath('data', 'processed')
```

Definition at line 42 of file predict_model.py.

5.10.1.26 required

```
predict_model.required
```

Definition at line 26 of file predict_model.py.

5.10.1.27 res_file

```
predict_model.res_file = model_dir.joinpath('TEST_' + split_ + '_result.json')
```

Definition at line 93 of file predict_model.py.

5.10.1.28 result

```
predict_model.result
```

Initial value:

```
1 =  generator.evaluate(test_path,
2                                annFile,
3                                res_file,
4                                eval_file,
5                                batch_size=val_batch_size,
6                                beam_size=beam_size_)
```

Definition at line 101 of file predict_model.py.

5.10.1.29 ROOT_PATH

```
predict_model.ROOT_PATH = Path(__file__).absolute().parents[2]
```

Definition at line 7 of file predict_model.py.

5.10.1.30 save_path_

```
predict_model.save_path_ = models_path
```

Definition at line 67 of file predict_model.py.

5.10.1.31 saved_model_path_

```
predict_model.saved_model_path_ = model_dir.joinpath('BEST_checkpoint.pth.tar')
```

Definition at line 62 of file predict_model.py.

5.10.1.32 seed_

```
int predict_model.seed_ = 222
```

Definition at line 73 of file predict_model.py.

5.10.1.33 split_

```
string predict_model.split_ = args['split']
```

Definition at line 46 of file predict_model.py.

5.10.1.34 str

```
predict_model.str
```

Definition at line 18 of file predict_model.py.

5.10.1.35 test_path

```
predict_model.test_path
```

Initial value:

```
1 =  interim_path.joinpath(dataset_,
2                                    dataset_ + '_' + split_ + '.csv')
```

Definition at line 57 of file predict_model.py.

5.10.1.36 train_path

```
predict_model.train_path = interim_path.joinpath(dataset_, dataset_ + '_train_clean.csv')
```

Definition at line 56 of file predict_model.py.

5.10.1.37 type

```
predict_model.type
```

Definition at line 15 of file predict_model.py.

5.10.1.38 val_batch_size

```
predict_model.val_batch_size = args['val_batch_size']
```

Definition at line 90 of file predict_model.py.

5.10.1.39 voc_path_

```
predict_model.voc_path_ = interim_path.joinpath(dataset_, dataset_ + '_vocabulary.csv')
```

Definition at line 59 of file predict_model.py.

5.11 preprocess_coco Namespace Reference

Functions

- def find_captions (captions, image_id)
- def make_dataframe (data_path)
- def preprocess_coco (data_path, output_path, splits)

Variables

- ROOT_PATH = Path(__file__).absolute().parents[2]

5.11.1 Function Documentation

5.11.1.1 find_captions()

```
def preprocess_coco.find_captions (
              captions,
              image_id )
```

Definition at line 8 of file preprocess_coco.py.

```
8 def find_captions(captions, image_id):
9     cap_ids = []
10     caps = []
11     remove_objects = []
12     for capobj in captions:
13         if capobj['image_id'] == image_id:
14             cap_ids.append(capobj['id'])
15             caps.append(capobj['caption'])
16             remove_objects.append(capobj)
17     # remove used captions
18     for c in remove_objects:
19         captions.remove(c)
20     # print('cap_size', len(captions))
21     return cap_ids, caps
22
23
```

5.11.1.2 make_dataframe()

```
def preprocess_coco.make_dataframe (
              data_path )
```

Definition at line 24 of file preprocess_coco.py.

```
24 def make_dataframe(data_path):
25     with open(data_path, 'r') as json_file:
26         data_dict = json.load(json_file)
27 
28     out_dict = {
29         'image_id': [],
30         'image_name': [],
31         'caption_id': [],
32         'caption': []
33     }
34 
35     images = data_dict['images']
36     captions = data_dict['annotations']
37     cap_counter = 0
38     for imgobj in images:
39         im_id = imgobj['id']
40         im_name = imgobj['file_name']
41         cap_ids, caps = find_captions(captions, imgobj['id'])
42         for c in range(len(cap_ids)):
43             cap_id = im_name + "#" + str(cap_ids[c])
44             caption = caps[c]
45             out_dict['image_id'].append(im_id)
46             out_dict['image_name'].append(im_name)
47             out_dict['caption_id'].append(im_name + "#" + cap_id)
48             out_dict['caption'].append(caption)
49             cap_counter += 1
50             if cap_counter % 1000 == 0:
51                 print(cap_counter)
52 
53     data_df = pd.DataFrame(data=out_dict, columns=out_dict.keys())
54     return data_df
55 
56 
```

5.11.1.3 preprocess_coco()

```
def preprocess_coco.preprocess_coco (
              data_path,
              output_path,
              splits )
```

Definition at line 57 of file preprocess_coco.py.

```
57 def preprocess_coco(data_path, output_path, splits):
58     # TODO: modify to create annotation files too
59     for split in splits:
60         d_path = data_path.joinpath('captions_' + split + '2014.json')
61         split_df = make_dataframe(d_path)
62         split_df.to_csv(output_path.joinpath('coco_' + split + '.csv'))
63 
64 
```

5.11.2 Variable Documentation

5.11.2.1 ROOT_PATH

```
preprocess_coco.ROOT_PATH = Path(__file__).absolute().parents[2]
```

Definition at line 5 of file preprocess_coco.py.

5.12 reduce_images_in_dataset Namespace Reference

5.12.1 Detailed Description

```
main file for creating splits of the training set where the number of images
are reduced and not the number of captions per image.
```

5.13 resize_images Namespace Reference

Functions

- def resize_images (image_path, save_path, new_dims)

Variables

- ROOT_PATH = Path(__file__).absolute().parents[2]

5.13.1 Function Documentation

5.13.1.1 resize_images()

```
def resize_images.resize_images (
                image_path,
                save_path,
                new_dims )
```

Definition at line 9 of file resize_images.py.

```
9 def resize_images(image_path, save_path, new_dims):
10     image_path = Path(image_path)
11     save_path = Path(save_path)
12     directory = save_path.joinpath(str(new_dims[0]) + 'x' + str(new_dims[1]))
13     if not directory.is_dir():
14         directory.mkdir(parents=True)
15     for im_file in image_path.glob('*.jpg'):
16         p = len(str(im_file.parent)) + 1
17         image_name = str(im_file)[p:]
18         image = imread(str(im_file))
19         image = resize(image, new_dims)
20         imsave(directory.joinpath(image_name), image)
```

5.13.2 Variable Documentation

5.13.2.1 ROOT_PATH

```
resize_images.ROOT_PATH = Path(__file__).absolute().parents[2]
```

Definition at line 6 of file resize_images.py.

5.14 Resnet_features Namespace Reference

Functions

- def load_pre_trained_model (output_layer_idx)
- def load_inception ()
- def encode (image, model)
- def encode_vis_att (image, model)
- def extract_image_features (image_path, save_path, split_set_path, output_layer_idx, vis_att=True)
- def load_visual_features (feature_path)

Variables

- ROOT_PATH = Path(__file__).absolute().parents[2]

5.14.1 Function Documentation

5.14.1.1 encode()

```
def Resnet_features.encode (
                image,
                model )
```

Definition at line 30 of file Resnet_features.py.

```
30 def encode(image, model):
31     from keras.applications.inception_resnet_v2 import preprocess_input
32     # add one more dimension
33     image = np.expand_dims(image, axis=0)
34     # preprocess the image
35     image = preprocess_input(image)
36     # Get the encoding vector for the image
37     fea_vec = model.predict(image)
38     # reshape from (1, 2048) to (2048, )
39     fea_vec = np.reshape(fea_vec, fea_vec.shape[1])
40     return fea_vec
41
42
```

5.14.1.2 encode_vis_att()

```
def Resnet_features.encode_vis_att (
                image,
                model )
```

Definition at line 43 of file Resnet_features.py.

```
43 def encode_vis_att(image, model):
44     from keras.applications.inception_resnet_v2 import preprocess_input
45     # add one more dimension
46     image = np.expand_dims(image, axis=0)
47     # preprocess the image
48     image = preprocess_input(image)
49     # Get the encoding vector for the image
50     fea_vec = model.predict(image)
51     # reshape from (1, 8, 8, 1536) to (8, 8, 1536)
52     fea_vec = np.reshape(fea_vec, fea_vec.shape[1:])
53     return fea_vec
54
55
```

5.14.1.3 extract_image_features()

```
def Resnet_features.extract_image_features (
                image_path,
                save_path,
                split_set_path,
                output_layer_idx,
                vis_att = True )
```

Definition at line 56 of file Resnet_features.py.

```
56 def extract_image_features(image_path,
57                            save_path,
58                            split_set_path,
59                            output_layer_idx,
60                            vis_att=True):
61     # consider splitting the process up in parts and then
62     # combining the parts at the end, to reduce the amount of images
63     # in memory at any time
64     image_path = Path(image_path)
65     save_path = Path(save_path)
66
67     if not save_path.is_dir():
68         save_path.mkdir(parents=True)
69
70     model = load_pre_trained_model(output_layer_idx)
71     data_df = pd.read_csv(split_set_path)
72     image_split = set(data_df.loc[:, 'image_id'])
73     start = time()
74     encoding_data = {}
75     n = len(image_split)
76     count = 0
77     for im_file in image_path.glob('*.jpg'):
78         p = len(str(im_file.parent)) + 1
79         im_file = str(im_file)
80         image_name = im_file[p:]
81         if image_name in image_split:
82             count += 1
83             if vis_att:
84                 encoding_data[image_name] = encode_vis_att(imread(im_file),
85                                                            model)
86             else:
87                 encoding_data[image_name] = encode(imread(im_file), model)
88             print(str(count) + ' / ' + str(n))
89     print("Time taken in seconds =", time() - start)
90     # Save the bottleneck train features to disk
91     with open(save_path, "wb") as encoded_pickle:
92         dump(encoding_data, encoded_pickle)
93
94
```

5.14.1.4 load_inception()

```
def Resnet_features.load_inception ( )
```

Definition at line 21 of file Resnet_features.py.

```
21 def load_inception():
22     from keras.applications.inception_v3 import InceptionV3
23     from keras.models import Model
24     model = InceptionV3(weights='imagenet')
25     new_model = Model(model.input, model.layers[-2].output)
26     new_model.summary()
27     return new_model
28
29
```

5.14.1.5 load_pre_trained_model()

```
def Resnet_features.load_pre_trained_model (
              output_layer_idx )
```

Definition at line 12 of file Resnet_features.py.

```
12 def load_pre_trained_model(output_layer_idx):
13     from keras.applications.inception_resnet_v2 import InceptionResNetV2
14     from keras.models import Model
15     model = InceptionResNetV2(weights='imagenet')
16     model_new = Model(model.input, model.layers[output_layer_idx].output)
17     model_new.summary()
18     return model_new
19
20
```

5.14.1.6 load_visual_features()

```
def Resnet_features.load_visual_features (
              feature_path )
```

Definition at line 95 of file Resnet_features.py.

```
95 def load_visual_features(feature_path):
96     with open(feature_path, 'rb') as file:
97         data_features = load(file)
98     print('Photos: %d' % len(data_features))
99     return data_features
```

5.14.2 Variable Documentation

5.14.2.1 ROOT_PATH

```
Resnet_features.ROOT_PATH = Path(__file__).absolute().parents[2]
```

Definition at line 9 of file Resnet_features.py.

5.15 split_flickr8k Namespace Reference

Functions

- def make_train_val_test_split (df_path, split_paths, save_path)

Variables

- ROOT_PATH = Path(__file__).absolute().parents[2]

5.15.1 Function Documentation

5.15.1.1 make_train_val_test_split()

```
def split_flickr8k.make_train_val_test_split (
              df_path,
              split_paths,
              save_path )
```

Definition at line 7 of file split_flickr8k.py.

```
7 def make_train_val_test_split(df_path, split_paths, save_path):
8     # get df
9     cap_df = pd.read_csv(df_path)
10     train_path = split_paths[0]
11     val_path = split_paths[1]
12     test_path = split_paths[2]
13
14     # read train val test files
15     with open(train_path, 'r') as train_file:
16         train_images = [im_id.strip() for im_id in train_file.readlines()
17                         if len(im_id) > 0]
18     with open(val_path, 'r') as val_file:
19         val_images = [im_id.strip() for im_id in val_file.readlines()
20                       if len(im_id) > 0]
21     with open(test_path, 'r') as test_file:
22         test_images = [im_id.strip() for im_id in test_file.readlines()
23                        if len(im_id) > 0]
24     train_df = cap_df.loc[cap_df.loc[:, 'image_name'].isin(train_images), :]
25     val_df = cap_df.loc[cap_df.loc[:, 'image_name'].isin(val_images), :]
26     test_df = cap_df.loc[cap_df.loc[:, 'image_name'].isin(test_images), :]
27     # make a merged version
28     full_df = train_df.copy()
29     full_df = full_df.append(val_df)
30     full_df = full_df.append(test_df)
31
32     # save splits
33     train_df.to_csv(save_path.joinpath('flickr8k_train.csv'))
34     val_df.to_csv(save_path.joinpath('flickr8k_val.csv'))
35     test_df.to_csv(save_path.joinpath('flickr8k_test.csv'))
36     # save full set
37     full_df.to_csv(save_path.joinpath('flickr8k_full.csv'))
38     print("Finished making splits!")
```

5.15.2 Variable Documentation

5.15.2.1 ROOT_PATH

```
split_flickr8k.ROOT_PATH = Path(__file__).absolute().parents[2]
```

Definition at line 4 of file split_flickr8k.py.

5.16 src Namespace Reference

5.17 subset_splits Namespace Reference

5.17.1 Detailed Description

```
Main file for creating the dataset splits, for the data exploration.
```

5.18 text_to_csv Namespace Reference

Functions

- def text_to_csv (file_path, save_path)

Variables

- ROOT_PATH = Path(__file__).absolute().parents[2]

5.18.1 Function Documentation

5.18.1.1 text_to_csv()

```
def text_to_csv.text_to_csv (
              file_path,
              save_path )
```

convert info from txt fil to csv. .csv file should be saved in data/interim folder.

Definition at line 17 of file text_to_csv.py.

```
17 def text_to_csv(file_path, save_path):
18     """
19     convert info from txt fil to csv. .csv file should be saved in
20     data/interim folder.
21     """
22     file_path = Path(file_path)
23     save_path = Path(save_path)
24
25     # set up caption dictionary
26     captions = {}
27     labels = ['image_id', 'image_name', 'caption_id', 'caption']
28     for l in labels:
29         captions[l] = []
30
31     image_name2ids = {}
32     counter = 0
33
34     # read txt file an extract info
35     with open(file_path, 'r') as file:
36         doc = file.read()
37     for line in doc.split('\n'):
38         # split line by white space
39         tokens = line.split()
40         if len(tokens) == 0:
41             continue
42         # take the first token as image id, the rest as description
43         caption_id, caption_tokens = tokens[0], tokens[1:]
44
45         # extract .jpg filename from image id
46         image_name = caption_id.split('#')[0]
47
48         if image_name not in image_name2ids:
49             image_name2ids[image_name] = counter
50             counter += 1
51
52         # convert description tokens back to string
53         caption = ' '.join(caption_tokens)
54
55         # add all info to the caption dictionary
56         captions['image_id'].append(image_name2ids[image_name])
57         captions['image_name'].append(image_name)
```

```
58          captions['caption_id'].append(caption_id)
59          captions['caption'].append(caption)
60
61      parent = save_path.parent
62      if not parent.is_dir():
63          # if the directory of the save path is not a directory
64          # then make it a directory along with any of its parents
65          parent.mkdir(parents=True)
66
67      # convert dict to DataFrame
68      cap_df = pd.DataFrame(data=captions, columns=labels)
69      cap_df.to_csv(save_path)
```

5.18.2 Variable Documentation

5.18.2.1 ROOT_PATH

```
text_to_csv.ROOT_PATH = Path(__file__).absolute().parents[2]
```

Definition at line 5 of file text_to_csv.py.

5.19 torch_generators Namespace Reference

Classes

- class AdaptiveDecoder
- class AdaptiveModel

Functions

- def model_switcher (model_str)

5.19.1 Function Documentation

5.19.1.1 model_switcher()

```
def torch_generators.model_switcher (
                model_str )
```

Definition at line 11 of file torch_generators.py.

```
11 def model_switcher(model_str):
12     model_str = model_str.lower()
13     switcher = {
14         'adaptive': AdaptiveModel,
15         'adaptive_decoder': [AdaptiveDecoder, ImageEncoder]
16     }
17     return switcher.get(model_str, AdaptiveModel)
18
19
```

5.20 train_model Namespace Reference

Variables

- ROOT_PATH = Path(__file__).absolute().parents[2]
- parser = argparse.ArgumentParser()
- type
- int
- default
- help
- str
- float
- bool
- nargs
- required
- args = vars(parser.parse_args())
- interim_path
- processed_path
- dataset = args['dataset']
- annFile
- train_path = interim_path.joinpath(dataset + '_train_clean.csv')
- val_path = interim_path.joinpath(dataset + '_val.csv')
- voc_path_ = interim_path.joinpath(dataset + '_vocabulary.csv')
- feature_path_
- save_path_ = ROOT_PATH.joinpath('models')
- model_name_ = args['model']
- batch_size = args['batch_size']
- val_batch_size = args['val_batch_size']
- beam_size = args['beam_size']
- epochs = args['epochs']
- em_dim = args['embedding_size']
- hidden_size_ = args['hidden_size']
- loss_function_ = args['loss_function']
- opt = args['optimizer']
- lr_ = args['lr']
- seed_ = args['seed']
- max_length = max_length_caption(train_path)
- image_feature_size = args['image_feature_size']
- list input_shape_ = [image_feature_size, max_length]
- generator

5.20.1 Variable Documentation

5.20.1.1 annFile

```
train_model.annFile
```

Initial value:

```
1 =  processed_path.joinpath('annotations',
2                                          'karpathy_split',
3                                          dataset + '_val.json')
```

Definition at line 68 of file train_model.py.

5.20.1.2 args

```
train_model.args = vars(parser.parse_args())
```

Definition at line 58 of file train_model.py.

5.20.1.3 batch_size

```
train_model.batch_size = args['batch_size']
```

Definition at line 82 of file train_model.py.

5.20.1.4 beam_size

```
train_model.beam_size = args['beam_size']
```

Definition at line 84 of file train_model.py.

5.20.1.5 bool

```
train_model.bool
```

Definition at line 46 of file train_model.py.

5.20.1.6 dataset

```
train_model.dataset = args['dataset']
```

Definition at line 64 of file train_model.py.

5.20.1.7 default

```
train_model.default
```

Definition at line 17 of file train_model.py.

5.20.1.8 em_dim

```
train_model.em_dim = args['embedding_size']
```

Definition at line 87 of file train_model.py.

5.20.1.9 epochs

```
train_model.epochs = args['epochs']
```

Definition at line 86 of file train_model.py.

5.20.1.10 feature_path_

```
train_model.feature_path_
```

Initial value:

```
1 =  processed_path.joinpath(
2         dataset, 'Images', 'encoded_visual_attention_full.pkl')
```

Definition at line 75 of file train_model.py.

5.20.1.11 float

```
train_model.float
```

Definition at line 40 of file train_model.py.

5.20.1.12 generator

```
train_model.generator
```

Initial value:

```
1 =  Generator(model_name_, input_shape_, hidden_size_,
2                           voc_path_, feature_path_,
3                           save_path_,
4                           loss_function=loss_function_,
5                           optimizer=opt, lr=lr_,
6                           embedding_size=em_dim,
7                           seed=seed_)
```

Definition at line 103 of file train_model.py.

5.20.1.13 help

```
train_model.help
```

Definition at line 18 of file train_model.py.

5.20.1.14 hidden_size_

```
train_model.hidden_size_ = args['hidden_size']
```

Definition at line 88 of file train_model.py.

5.20.1.15 image_feature_size

```
train_model.image_feature_size = args['image_feature_size']
```

Definition at line 96 of file train_model.py.

5.20.1.16 input_shape_

```
list train_model.input_shape_ = [image_feature_size, max_length]
```

Definition at line 101 of file train_model.py.

5.20.1.17 int

```
train_model.int
```

Definition at line 17 of file train_model.py.

5.20.1.18 interim_path

```
train_model.interim_path
```

Initial value:

```
1 =  ROOT_PATH.joinpath('data',
2                                    'interim')
```

Definition at line 60 of file train_model.py.

5.20.1.19 loss_function_

```
train_model.loss_function_ = args['loss_function']
```

Definition at line 89 of file train_model.py.

5.20.1.20 lr_

```
train_model.lr_ = args['lr']
```

Definition at line 91 of file train_model.py.

5.20.1.21 max_length

```
train_model.max_length = max_length_caption(train_path)
```

Definition at line 94 of file train_model.py.

5.20.1.22 model_name_

```
train_model.model_name_ = args['model']
```

Definition at line 80 of file train_model.py.

5.20.1.23 nargs

```
train_model.nargs
```

Definition at line 53 of file train_model.py.

5.20.1.24 opt

```
train_model.opt = args['optimizer']
```

Definition at line 90 of file train_model.py.

5.20.1.25 parser

```
train_model.parser = argparse.ArgumentParser()
```

Definition at line 16 of file train_model.py.

5.20.1.26 processed_path

```
train_model.processed_path
```

Initial value:

```
1 =  ROOT_PATH.joinpath('data',
2                                    'processed')
```

Definition at line 62 of file train_model.py.

5.20.1.27 required

```
train_model.required
```

Definition at line 53 of file train_model.py.

5.20.1.28 ROOT_PATH

```
train_model.ROOT_PATH = Path(__file__).absolute().parents[2]
```

Definition at line 7 of file train_model.py.

5.20.1.29 save_path_

```
train_model.save_path_ = ROOT_PATH.joinpath('models')
```

Definition at line 78 of file train_model.py.

5.20.1.30 seed_

```
train_model.seed_ = args['seed']
```

Definition at line 92 of file train_model.py.

5.20.1.31 str

```
train_model.str
```

Definition at line 36 of file train_model.py.

5.20.1.32 train_path

```
train_model.train_path = interim_path.joinpath(dataset + '_train_clean.csv')
```

Definition at line 72 of file train_model.py.

5.20.1.33 type

```
train_model.type
```

Definition at line 17 of file train_model.py.

5.20.1.34 val_batch_size

```
train_model.val_batch_size = args['val_batch_size']
```

Definition at line 83 of file train_model.py.

5.20.1.35 val_path

```
train_model.val_path = interim_path.joinpath(dataset + '_val.csv')
```

Definition at line 73 of file train_model.py.

5.20.1.36 voc_path_

```
train_model.voc_path_ = interim_path.joinpath(dataset + '_vocabulary.csv')
```

Definition at line 74 of file train_model.py.

5.21 utils Namespace Reference

Functions

- def max_length_caption (df_path)
- def load_vocabulary (voc_path)
- def save_checkpoint (directory, epoch, epochs_since_improvement, encoder, decoder, enc_optimizer, dec↩ _optimizer, cider, is_best)
- def save_training_log (path, training_history)

Variables

- ROOT_PATH = Path(__file__).absolute().parents[2]

5.21.1 Function Documentation

5.21.1.1 load_vocabulary()

```
def utils.load_vocabulary (
               voc_path )
```

Definition at line 16 of file utils.py.

```
16 def load_vocabulary(voc_path):
17     # vocabulary need to be consistent whenever this function is called
18     # vocabulary must therefore be loaded as a list
19     with open(voc_path, 'r') as voc_file:
20         vocabulary = [word.strip() for word in voc_file.readlines()
21                       if len(word) > 0]
22     vocabulary.insert(0, 'UNK')
23     wordtoix = {}
24     ixtoword = {}
25     for i, word in enumerate(vocabulary):
26         wordtoix[word] = i
27         ixtoword[i] = word
28     return wordtoix, ixtoword
```

5.21.1.2 max_length_caption()

```
def utils.max_length_caption (
               df_path )
```

Definition at line 7 of file utils.py.

```
7 def max_length_caption(df_path):
8     df = pd.read_csv(df_path)
9     captions = set(df.loc[:, 'clean_caption'])
10     length = max(len(c.split()) for c in captions)
11     print('DATAFRAME PATH', df_path)
12     print('MAX LENGTH OF CAPTION', length)
13     return length
14
15
```

5.21.1.3 save_checkpoint()

```
def utils.save_checkpoint (
                directory,
                epoch,
                epochs_since_improvement,
                encoder,
                decoder,
                enc_optimizer,
                dec_optimizer,
                cider,
                is_best )
```

Definition at line 7 of file utils.py.

```
7 def save_checkpoint(directory,
8                     epoch,
9                     epochs_since_improvement,
10                     encoder,
11                     decoder,
12                     enc_optimizer,
13                     dec_optimizer,
14                     cider,
15                     is_best):
16     directory = Path(directory)
17
18     # remove worse checkpoints
19     for file in directory.glob('checkpoint_*'):
20         file.unlink()
21
22     state = {
23         'epoch': epoch,
24         'epochs_since_improvement': epochs_since_improvement,
25         'cider': cider,
26         'encoder': encoder,
27         'decoder': decoder,
28         'enc_optimizer': enc_optimizer,
29         'dec_optimizer': dec_optimizer
30     }
31
32     filename = directory.joinpath('checkpoint_' + str(epoch) + '.pth.tar')
33     torch.save(state, filename)
34     # If this checkpoint is the best so far,
35     # store a copy so it doesn't get overwritten by a worse checkpoint
36     if is_best:
37         # remove last best checkpoint
38         for file in directory.glob('BEST_*'):
39             file.unlink()
40
41         filename = directory.joinpath('BEST_checkpoint.pth.tar')
42         torch.save(state, filename)
43         return filename
44     return None
45
46
```

5.21.1.4 save_training_log()

```
def utils.save_training_log (
                path,
                training_history )
```

Definition at line 47 of file utils.py.

```
47 def save_training_log(path, training_history):
48     # I do not care that the function is static
49     with open(path, 'w') as train_log:
50         train_log.write('################## '
51                         'LOG FILE '
52                         '##################\n\n')
53         train_log.write('DATA and FEATURES\n')
54         train_log.write('Training data path: ' +
```

```
55                      training_history['train_path'] + '\n')
56          train_log.write('Feature path: ' + training_history['feature_path']
57                      + '\n')
58          train_log.write('Vocabulary path: ' + training_history['voc_path']
59                      + '\n')
60          train_log.write('Vocabulary size: ' + training_history['voc_size']
61                      + '\n\n')
62
63          train_log.write('## CONFIGS / HYPERPARAMETERS ##\n')
64          train_log.write('Optimizer: ' + training_history['optimizer'] +
65                      '\n')
66          train_log.write('Learning rate: ' + training_history['lr']
67                      + '\n\n')
68
69          train_log.write('####### '
70                      'MODEL '
71                      '#######\n')
72          train_log.write('Model name: ' + training_history['model_name']
73                      + '\n\n')
74          train_log.write(training_history['encoder'] + '\n')
75          train_log.write(training_history['decoder'] + '\n')
76          train_log.write('Trainable parameters: ' +
77                      training_history['trainable_parameters'] + '\n')
78          train_log.write('Model save path: ' +
79                      training_history['model_save_path'] + '\n\n')
80
81          train_log.write('## Training Configs ##\n')
82          train_log.write('Epochs: ' + training_history['epochs'] + '\n')
83          train_log.write('Batch size: ' + training_history['batch_size']
84                      + '\n')
85          train_log.write('Training time: ' +
86                      training_history['training_time'] + '\n')
87          train_log.write('Loss function: ' + training_history['loss']
88                      + '\n\n')
89          train_log.write('## Train log!\n')
90          # Lastly write the training log
91          for loss in training_history['history']:
92              # TODO: add val score.... zip?
93              train_log.write(str(round(loss, 5)) + '\n')
```

5.21.2 Variable Documentation

5.21.2.1 ROOT_PATH

```
utils.ROOT_PATH = Path(__file__).absolute().parents[2]
```

Definition at line 4 of file utils.py.

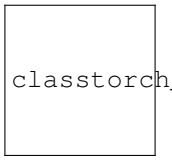
5.22 visualize Namespace Reference

Chapter 6

Class Documentation

6.1 torch_generators.AdaptiveDecoder Class Reference

Inheritance diagram for torch_generators.AdaptiveDecoder:



Public Member Functions

- def __init__ (self, input_shape, hidden_size, vocab_size, device, num_lstms=0, embedding_size=300, seed=222)
- def forward (self, x, states)
- def initialize_variables (self, batch_size)

Public Attributes

- input_shape
- visual_feature_shape
- max_len
- hidden_size
- vocab_size
- em_size
- num_lstms
- random_seed
- device
- embedding
- sentinel_lstm
- attention_block
- decoder

6.1.1 Detailed Description

```
Adaptive Decoder.

This class will not do any encoding of the images, but expects an
encoded image as input. This generator does not output full sequences,
instead it only outputs the predictions at a timestep.
```

Definition at line 115 of file torch_generators.py.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 __init__()

```
def torch_generators.AdaptiveDecoder.__init__ (
              self,
              input_shape,
              hidden_size,
              vocab_size,
              device,
              num_lstms = 0,
              embedding_size = 300,
              seed = 222 )
```

Definition at line 125 of file torch_generators.py.

```
125     def __init__(self,
126                  input_shape,
127                  hidden_size,
128                  vocab_size,
129                  device,
130                  num_lstms=0,
131                  embedding_size=300,
132                  seed=222):
133         super(AdaptiveDecoder, self).__init__()
134
135         self.input_shape = input_shape
136         self.visual_feature_shape = input_shape[0]
137         self.max_len = input_shape[1]
138         self.hidden_size = hidden_size
139
140         self.vocab_size = vocab_size
141         self.em_size = embedding_size
142         self.num_lstms = num_lstms
143         self.random_seed = seed
144
145         self.device = device
146
147         # layers
148         self.embedding = nn.Embedding(self.vocab_size, self.em_size)
149         self.sentinel_lstm = SentinelLSTM(self.em_size * 2,
150                                           self.hidden_size,
151                                           n=self.num_lstms)
152         self.attention_block = AttentionLayer(self.hidden_size,
153                                               self.hidden_size)
154         self.decoder = MultimodalDecoder(self.hidden_size,
155                                          self.vocab_size, n=1)
156
```

6.1.3 Member Function Documentation

6.1.3.1 forward()

```
def torch_generators.AdaptiveDecoder.forward (
                self,
                x,
                states )
```

Definition at line 157 of file torch_generators.py.

```
157     def forward(self, x, states):
158         # unpack input
159         input_img, input_w = x
160         global_image, encoded_images = input_img
161         # global (batch_size, embedding_size)
162         # encoded (batch_size, 64, hidden_size)
163 
164         # embed word
165         embedded_w = self.embedding(input_w)
166         # (batch_size, embedding_size)
167 
168         # cat input w with v_avg
169         x_t = torch.cat((embedded_w, global_image), dim=1)
170         # (batch_size, embedding_size*2)
171 
172         # get states
173         h_tm1, c_tm1 = states
174 
175         # decoding
176         h_t, c_t, h_top, s_t = self.sentinel_lstm(x_t, (h_tm1, c_tm1))
177         z_t = self.attention_block([encoded_images, s_t, h_top])
178         pt = self.decoder(z_t)
179 
180         return pt, h_t, c_t
181 
```

6.1.3.2 initialize_variables()

```
def torch_generators.AdaptiveDecoder.initialize_variables (
                self,
                batch_size )
```

Definition at line 182 of file torch_generators.py.

```
182     def initialize_variables(self, batch_size):
183         # initialize h and c as zeros
184         hs = torch.zeros(self.num_lstms + 1, batch_size, self.hidden_size) \
185             .to(self.device)
186         cs = torch.zeros(self.num_lstms + 1, batch_size, self.hidden_size) \
187             .to(self.device)
188         return hs, cs
189 
190 
191 
```

6.1.4 Member Data Documentation

6.1.4.1 attention_block

```
torch_generators.AdaptiveDecoder.attention_block
```

Definition at line 145 of file torch_generators.py.

6.1.4.2 decoder

`torch_generators.AdaptiveDecoder.decoder`

Definition at line 147 of file torch_generators.py.

6.1.4.3 device

`torch_generators.AdaptiveDecoder.device`

Definition at line 138 of file torch_generators.py.

6.1.4.4 em_size

`torch_generators.AdaptiveDecoder.em_size`

Definition at line 134 of file torch_generators.py.

6.1.4.5 embedding

`torch_generators.AdaptiveDecoder.embedding`

Definition at line 141 of file torch_generators.py.

6.1.4.6 hidden_size

`torch_generators.AdaptiveDecoder.hidden_size`

Definition at line 131 of file torch_generators.py.

6.1.4.7 input_shape

`torch_generators.AdaptiveDecoder.input_shape`

Definition at line 128 of file torch_generators.py.

6.1.4.8 max_len

`torch_generators.AdaptiveDecoder.max_len`

Definition at line 130 of file torch_generators.py.

6.1.4.9 num_lstms

`torch_generators.AdaptiveDecoder.num_lstms`

Definition at line 135 of file torch_generators.py.

6.1.4.10 random_seed

`torch_generators.AdaptiveDecoder.random_seed`

Definition at line 136 of file torch_generators.py.

6.1.4.11 sentinel_lstm

`torch_generators.AdaptiveDecoder.sentinel_lstm`

Definition at line 142 of file torch_generators.py.

6.1.4.12 visual_feature_shape

`torch_generators.AdaptiveDecoder.visual_feature_shape`

Definition at line 129 of file torch_generators.py.

6.1.4.13 vocab_size

`torch_generators.AdaptiveDecoder.vocab_size`

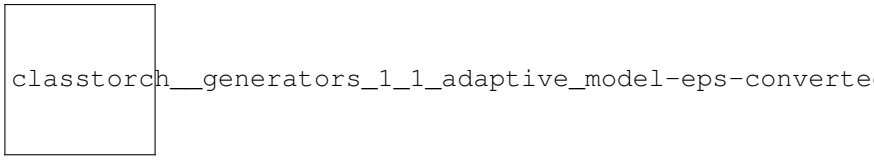
Definition at line 133 of file torch_generators.py.

The documentation for this class was generated from the following file:

- src/models/torch_generators.py

6.2 torch_generators.AdaptiveModel Class Reference

Inheritance diagram for torch_generators.AdaptiveModel:



Public Member Functions

- def __init__ (self, input_shape, hidden_size, vocab_size, device, num_lstms=0, embedding_size=300, seed=222)
- def initialize_variables (self, batch_size)
- def forward (self, x, caption_lengths, has_end_seq_token=True)

Public Attributes

- input_shape
- visual_feature_shape
- max_len
- hidden_size
- vocab_size
- em_size
- num_lstms
- random_seed
- device
- image_encoder
- decoder

6.2.1 Detailed Description

Definition at line 20 of file torch_generators.py.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 __init__()

```
def torch_generators.AdaptiveModel.__init__ (
                self,
                input_shape,
                hidden_size,
                vocab_size,
                device,
                num_lstms = 0,
                embedding_size = 300,
                seed = 222 )
```

Definition at line 22 of file torch_generators.py.

```
22     def __init__(self,
23                  input_shape,
24                  hidden_size,
25                  vocab_size,
26                  device,
27                  num_lstms=0,
28                  embedding_size=300,
29                  seed=222):
30         super(AdaptiveModel, self).__init__()
31         self.input_shape = input_shape
32         self.visual_feature_shape = input_shape[0]
33         self.max_len = input_shape[1]
34         self.hidden_size = hidden_size
35 
36         self.vocab_size = vocab_size
37         self.em_size = embedding_size
38         self.num_lstms = num_lstms
39         self.random_seed = seed
40 
41         self.device = device
42 
43         # layers
44         # encoder
45         self.image_encoder = ImageEncoder(self.visual_feature_shape,
46                                           self.hidden_size,
47                                           self.em_size)
48         # decoder
49         self.decoder = AdaptiveDecoder(self.input_shape,
50                                        self.hidden_size,
51                                        self.vocab_size,
52                                        self.device,
53                                        num_lstms=self.num_lstms,
54                                        embedding_size=self.em_size,
55                                        seed=self.random_seed)
56 
```

6.2.3 Member Function Documentation

6.2.3.1 forward()

```
def torch_generators.AdaptiveModel.forward (
                self,
                x,
                caption_lengths,
                has_end_seq_token = True )
```

Definition at line 65 of file torch_generators.py.

```
65     def forward(self, x, caption_lengths, has_end_seq_token=True):
66         # visual features (batch_size, 8 ,8, 1536)
67         # batch_size is equal to the number of captions
68         im_input = x[0].to(self.device)
69         w_input = x[1].to(self.device)
70 
```

```
71          global_images, encoded_images = self.image_encoder(im_input)
72          # (batch_size, embedding_size) (batch, 512) global_images
73          # (batch_size, region_size, hidden_size) (batch, 64, 512) encoded_imgs
74
75          # sort batches by caption length descending, this way the whole
76          # batch_size_t will be correct
77          # convert to tensor
78          caption_lengths = torch.from_numpy(caption_lengths)
79          caption_lengths, sort_idx = caption_lengths.sort(dim=0,
80                                                           descending=True)
81          w_input = w_input[sort_idx]  # (batch_size, max_len)
82          global_images = global_images[sort_idx]  # (batch_size, embedding_size)
83          encoded_images = encoded_images[sort_idx]  # (batch_size, 64, 1536)
84
85          batch_size = encoded_images.size()[0]
86
87          decoding_lengths = np.copy(caption_lengths)
88          if has_end_seq_token:
89              decoding_lengths = (decoding_lengths - 1)
90          batch_max_length = max(decoding_lengths)
91
92          predictions = torch.zeros(batch_size,
93                                    self.max_len,
94                                    self.vocab_size)
95          # initialize h and c
96          h_t, c_t = self.decoder.initialize_variables(batch_size)
97
98          for timestep in range(batch_max_length):
99              batch_size_t = sum([lens > timestep for lens in decoding_lengths])
100             # x: [input_img, input_w]
101             # input_img: [global_image, encoded_image]
102             # image features does not vary over time
103             input_image_t = [global_images[:batch_size_t],
104                              encoded_images[:batch_size_t]]
105             x_t = [input_image_t, w_input[:batch_size_t, timestep]]
106
107             pt, h_t, c_t = self.decoder(x_t,
108                                         (h_t[:, :batch_size_t],
109                                          c_t[:, :batch_size_t]))
110             predictions[:batch_size_t, timestep, :] = pt
111
112         return predictions, decoding_lengths
113
114
```

6.2.3.2 initialize_variables()

```
def torch_generators.AdaptiveModel.initialize_variables (
                self,
                batch_size )
```

Definition at line 57 of file torch_generators.py.

```
57      def initialize_variables(self, batch_size):
58          # initialize h and c as zeros
59          hs = torch.zeros(self.num_lstms + 1, batch_size, self.hidden_size)\
60              .to(self.device)
61          cs = torch.zeros(self.num_lstms + 1, batch_size, self.hidden_size)\
62              .to(self.device)
63          return hs, cs
64
```

6.2.4 Member Data Documentation

6.2.4.1 decoder

```
torch_generators.AdaptiveModel.decoder
```

Definition at line 42 of file torch_generators.py.

6.2.4.2 device

`torch_generators.AdaptiveModel.device`

Definition at line 34 of file torch_generators.py.

6.2.4.3 em_size

`torch_generators.AdaptiveModel.em_size`

Definition at line 30 of file torch_generators.py.

6.2.4.4 hidden_size

`torch_generators.AdaptiveModel.hidden_size`

Definition at line 27 of file torch_generators.py.

6.2.4.5 image_encoder

`torch_generators.AdaptiveModel.image_encoder`

Definition at line 38 of file torch_generators.py.

6.2.4.6 input_shape

`torch_generators.AdaptiveModel.input_shape`

Definition at line 24 of file torch_generators.py.

6.2.4.7 max_len

`torch_generators.AdaptiveModel.max_len`

Definition at line 26 of file torch_generators.py.

6.2.4.8 num_lstms

`torch_generators.AdaptiveModel.num_lstms`

Definition at line 31 of file torch_generators.py.

6.2.4.9 random_seed

`torch_generators.AdaptiveModel.random_seed`

Definition at line 32 of file torch_generators.py.

6.2.4.10 visual_feature_shape

`torch_generators.AdaptiveModel.visual_feature_shape`

Definition at line 25 of file torch_generators.py.

6.2.4.11 vocab_size

`torch_generators.AdaptiveModel.vocab_size`

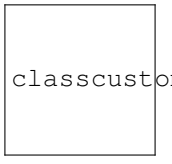
Definition at line 29 of file torch_generators.py.

The documentation for this class was generated from the following file:

- src/models/torch_generators.py

6.3 custom_layers.AttentionLayer Class Reference

Inheritance diagram for custom_layers.AttentionLayer:



Public Member Functions

- def __init__ (self, input_size, hidden_size)
- def forward (self, x)

Public Attributes

- v_att
- s_proj
- s_att
- h_proj
- h_att
- alpha_layer
- context_proj

6.3.1 Detailed Description

Definition at line 112 of file custom_layers.py.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 __init__()

```
def custom_layers.AttentionLayer.__init__ (
              self,
              input_size,
              hidden_size )
```

Definition at line 114 of file custom_layers.py.

```
114     def __init__(self, input_size, hidden_size):
115         super(AttentionLayer, self).__init__()
116         # input_size 512
117         # hidden_size 512
118 
119         self.v_att = nn.Linear(input_size, hidden_size)
120 
121         self.s_proj = nn.Linear(input_size, input_size)
122         self.s_att = nn.Linear(input_size, hidden_size)
123 
124         self.h_proj = nn.Linear(input_size, input_size)
125         self.h_att = nn.Linear(input_size, hidden_size)
126 
127         self.alpha_layer = nn.Linear(hidden_size, 1)
128         # might move this outside
129         self.context_proj = nn.Linear(input_size, input_size)
130 
```

6.3.3 Member Function Documentation

6.3.3.1 forward()

```
def custom_layers.AttentionLayer.forward (
              self,
              x )
```

Definition at line 131 of file custom_layers.py.

```
131     def forward(self, x):
132         # x : [V, s_t, h_t]
133         # V = [v1, v2, ..., vk]
134         # c_t is context vector: sum of alphas*v
135         # output should be beta*s_t + (1-beta)*c_t
136         # print('attention layer')
137         V = x[0]  # (batch_size, 8x8, hidden_size)
138         s_t = x[1]  # (batch_size, hidden_size)
139         h_t = x[2]  # (batch_size, hidden_size)
140 
141         # embed visual features
142         v_embed = F.relu(self.v_att(V))  # (batch_size, 64, hidden_size)
143 
144         # s_t embedding
145         s_proj = F.relu(self.s_proj(s_t))  # (batch_size, hidden_size)
146         s_att = self.s_att(s_proj)  # (batch_size, hidden_size)
147 
148         # h_t embedding
149         h_proj = torch.tanh(self.h_proj(h_t))  # (batch_size, hidden_size)
150         h_att = self.h_att(h_proj)  # (batch_size, hidden_size)
151 
152         # make s_proj the same dimension as V
153         s_proj = s_proj.unsqueeze(1)  # (batch_size, 1, hidden_size)
154 
155         # make s_att the same dimension as v_att
156         s_att = s_att.unsqueeze(1)  # (batch_size, 1, hidden_size)
157 
158         # make h_att the same dimension as regions_att
159         h_att = h_att.unsqueeze(1).expand(h_att.size()[0],
160                                           V.size()[1] + 1,
161                                           h_att.size()[1])
162         # (batch_size, 64 + 1, hidden_size)
163 
164         # concatenations
165         regions = torch.cat((V, s_proj), dim=1)
166         # (batch_size, 64 +1, hidden_size)
167         regions_att = torch.cat((v_embed, s_att), dim=1)
168         # (batch_size, 64 +1, hidden_size)
169 
170         # add h_t to regions_att
171         alpha_input = F.tanh(regions_att + h_att)
172         # (batch_size, 64 +1, hidden_size)
173 
174         # compute alphas + beta
175         alpha = F.softmax(self.alpha_layer(alpha_input).squeeze(2), dim=1)
176         # (batch_size, 64 + 1)
177         alpha = alpha.unsqueeze(2)  # (batch_size, 64 +1, 1)
178 
179         # multiply with regions
180         context_vector = (alpha * regions).sum(dim=1)  # the actual z_t
181         # (batch_size, hidden_size)
182 
183         z_t = torch.tanh(self.context_proj(context_vector + h_proj))
184         # (batch_size, hidden_size)
185 
186         return z_t
```

6.3.4 Member Data Documentation

6.3.4.1 alpha_layer

```
custom_layers.AttentionLayer.alpha_layer
```

Definition at line 127 of file custom_layers.py.

6.3.4.2 context_proj

`custom_layers.AttentionLayer.context_proj`

Definition at line 129 of file custom_layers.py.

6.3.4.3 h_att

`custom_layers.AttentionLayer.h_att`

Definition at line 125 of file custom_layers.py.

6.3.4.4 h_proj

`custom_layers.AttentionLayer.h_proj`

Definition at line 124 of file custom_layers.py.

6.3.4.5 s_att

`custom_layers.AttentionLayer.s_att`

Definition at line 122 of file custom_layers.py.

6.3.4.6 s_proj

`custom_layers.AttentionLayer.s_proj`

Definition at line 121 of file custom_layers.py.

6.3.4.7 v_att

`custom_layers.AttentionLayer.v_att`

Definition at line 119 of file custom_layers.py.

The documentation for this class was generated from the following file:

- src/models/custom_layers.py

6.4 beam.Beam Class Reference

Public Member Functions

- def __init__ (self, image, states, beam_size, input_token, eos, max_len, vocab_size, device, beam_id=-1)
- def update (self, predictions, h, c)
- def find_best_comlpete_sequence (self)
- def get_sequences (self)
- def get_encoded_image (self)
- def get_global_image (self)
- def get_hidden_states (self)
- def get_cell_states (self)
- def has_best_sequence (self)
- def get_best_sequence (self)

Public Attributes

- id
- encoded_image
- c
- beam_size
- num_unfinished
- EOS
- max_len
- vocab_size
- device
- captions
- previous_words
- top_scores
- finished_caps
- finished_caps_scores
- optimality_certificate
- h

6.4.1 Detailed Description

Definition at line 4 of file beam.py.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 __init__()

```
def beam.Beam.__init__ (
                self,
                image,
                states,
                beam_size,
                input_token,
                eos,
                max_len,
                vocab_size,
                device,
                beam_id = -1 )
```

Definition at line 6 of file beam.py.

```
6     def __init__(self,
7                  image,
8                  states,
9                  beam_size,
10                  input_token,
11                  eos,
12                  max_len,
13                  vocab_size,
14                  device,
15                  beam_id=-1):
16         # misc
17         self.id = beam_id
18         self.global_image, self.encoded_image = image
19         self.h, self.c = states
20         self.beam_size = beam_size
21         self.num_unfinished = beam_size
22         self.EOS = eos
23         self.max_len = max_len
24         self.vocab_size = vocab_size
25         self.device = device
26         # initialize captions in beam
27         # unfinished captions
28         init_cap = torch.tensor(input_token).to(self.device)
29         self.captions = init_cap.unsqueeze(0).expand(self.beam_size, -1)
30         init_prev_w = torch.tensor(input_token).to(self.device)
31         init_prev_w = init_prev_w.unsqueeze(0).expand(self.beam_size, -1)
32         self.previous_words = init_prev_w.squeeze(1)
33         self.top_scores = torch.zeros(self.beam_size, 1).to(self.device)
34         # finished captions
35         self.finished_caps = []  # these will never be sorted
36         self.finished_caps_scores = []  # will never be sorted
37         # Early stopping
38         self.optimality_certificate = False
39
```

6.4.3 Member Function Documentation

6.4.3.1 find_best_comlpete_sequence()

```
def beam.Beam.find_best_comlpete_sequence (
                self )
```

Definition at line 114 of file beam.py.

```
114     def find_best_comlpete_sequence(self):
115         probs = torch.tensor(self.finished_caps_scores).to(self.device)
116         _, idx = probs.topk(1, dim=0)
117         return int(idx)
118
```

6.4.3.2 get_best_sequence()

```
def beam.Beam.get_best_sequence (
              self )
```

```
Returns
-------
A list of sequence tokens.
```

Definition at line 140 of file beam.py.

```
140     def get_best_sequence(self):
141         """
142         Returns
143         -------
144         A list of sequence tokens.
145         """
146         assert self.has_best_sequence(), "Beam search incomplete"
147         idx = self.find_best_comlpete_sequence()
148         return self.finished_caps[idx]
```

6.4.3.3 get_cell_states()

```
def beam.Beam.get_cell_states (
              self )
```

Definition at line 133 of file beam.py.

```
133     def get_cell_states(self):
134         return self.c
135
```

6.4.3.4 get_encoded_image()

```
def beam.Beam.get_encoded_image (
              self )
```

Definition at line 123 of file beam.py.

```
123     def get_encoded_image(self):
124         return self.encoded_image.unsqueeze(0).expand(self.num_unfinished,
125                                                       -1, -1)
126
```

6.4.3.5 get_global_image()

```
def beam.Beam.get_global_image (
              self )
```

Definition at line 127 of file beam.py.

```
127     def get_global_image(self):
128         return self.global_image.unsqueeze(0).expand(self.num_unfinished, -1)
129
```

6.4.3.6 get_hidden_states()

```
def beam.Beam.get_hidden_states (
                self )
```

Definition at line 130 of file beam.py.

```
130     def get_hidden_states(self):
131         return self.h
132
```

6.4.3.7 get_sequences()

```
def beam.Beam.get_sequences (
                self )
```

Definition at line 119 of file beam.py.

```
119     def get_sequences(self):
120         # only return unfinished
121         return self.previous_words
122
```

6.4.3.8 has_best_sequence()

```
def beam.Beam.has_best_sequence (
                self )
```

Definition at line 136 of file beam.py.

```
136     def has_best_sequence(self):
137         return len(self.finished_caps) == self.beam_size or \
138                 self.optimality_certificate
139
```

6.4.3.9 update()

```
def beam.Beam.update (
                self,
                predictions,
                h,
                c )
```

Definition at line 40 of file beam.py.

```
40     def update(self, predictions, h, c):
41         # h, c: (n, num_unfinished, hidden_size)
42         # predictions (num_unfinished, vocab_size)
43         if self.num_unfinished == 0 or self.optimality_certificate:
44             # Do nothing
45             return
46
47         # add probabilities
48         # (beam_size, 1) --> (beam_size, vocab_size)
49         scores = self.top_scores.expand_as(predictions)
50         # (beam_size, v) + (num_unfinished, v) --> (num_unfinished, v)
51         predictions = scores + predictions
52
53         # flatten predictions and find top k
```

```
54          top_probs, top_words = predictions.view(-1).topk(self.num_unfinished,
55                                                           dim=0,
56                                                           largest=True,
57                                                           sorted=True)
58          # top_probs, top_words: (num_unfinished)
59          prev_word_idx = top_words / self.vocab_size  # previous index
60          next_word_idx = top_words % self.vocab_size  # word predicted
61
62          # add predicted words to caption
63          self.captions = torch.cat([self.captions[prev_word_idx],
64                                     next_word_idx.unsqueeze(1)],
65                                    dim=1)
66          unfinished_idx = [idx for idx, next_word in enumerate(next_word_idx)
67                            if next_word != self.EOS]
68          finished_idx = list(set(range(len(next_word_idx))) -
69                              set(unfinished_idx))
70
71          # sort h, c, top_scores, previous_words
72          # only keep the unfinished ones
73          self.h = h[:, prev_word_idx[unfinished_idx]]
74          self.c = c[:, prev_word_idx[unfinished_idx]]
75          self.top_scores = top_probs[unfinished_idx].unsqueeze(1)
76          self.previous_words = next_word_idx[unfinished_idx]
77
78          # update finished captions
79          beam_reduce_num = min(len(finished_idx), self.num_unfinished)
80          if beam_reduce_num > 0:
81              # add finished captions to finished
82              self.finished_caps.extend(self.captions[finished_idx].tolist())
83              self.finished_caps_scores.extend(top_probs[finished_idx])
84              self.num_unfinished -= beam_reduce_num
85
86              # find best complete caption
87              best_fin_idx = self.find_best_comlpete_sequence()
88              # check whether optimality certificate is achieved
89              if len(self.finished_caps) < self.beam_size:
90                  # only check if there still are unfinished caps left
91                  # if there are none in unfinished then there is no
92                  # point to check whether the optimality certificate is obtained
93                  best_unfin_prob, _ = self.top_scores.topk(1, dim=0)
94                  self.optimality_certificate = \
95                      self.finished_caps_scores[best_fin_idx] > best_unfin_prob
96
97              if self.optimality_certificate:
98                  # no need for further calculations
99                  self.num_unfinished = 0
100
101          # sort captions, only keep the unfinished ones
102          # could not do this until after finished was updated
103          self.captions = self.captions[unfinished_idx]
104
105          if self.captions.size(0):
106              # there are still more captions left
107              # move captions to finished if length too long
108              if self.captions.size(1) >= self.max_len:
109                  self.finished_caps.extend(self.captions.tolist())
110                  self.finished_caps_scores.extend(self.top_scores)
111                  self.captions = torch.empty(self.beam_size)
112                  self.num_unfinished = 0
113
```

6.4.4 Member Data Documentation

6.4.4.1 beam_size

`beam.Beam.beam_size`

Definition at line 11 of file beam.py.

6.4.4.2 c

beam.Beam.c

Definition at line 10 of file beam.py.

6.4.4.3 captions

beam.Beam.captions

Definition at line 20 of file beam.py.

6.4.4.4 device

beam.Beam.device

Definition at line 16 of file beam.py.

6.4.4.5 encoded_image

beam.Beam.encoded_image

Definition at line 9 of file beam.py.

6.4.4.6 EOS

beam.Beam.EOS

Definition at line 13 of file beam.py.

6.4.4.7 finished_caps

beam.Beam.finished_caps

Definition at line 26 of file beam.py.

6.4.4.8 finished_caps_scores

`beam.Beam.finished_caps_scores`

Definition at line 27 of file beam.py.

6.4.4.9 h

`beam.Beam.h`

Definition at line 73 of file beam.py.

6.4.4.10 id

`beam.Beam.id`

Definition at line 8 of file beam.py.

6.4.4.11 max_len

`beam.Beam.max_len`

Definition at line 14 of file beam.py.

6.4.4.12 num_unfinished

`beam.Beam.num_unfinished`

Definition at line 12 of file beam.py.

6.4.4.13 optimality_certificate

`beam.Beam.optimality_certificate`

Definition at line 29 of file beam.py.

6.4.4.14 previous_words

beam.Beam.previous_words

Definition at line 23 of file beam.py.

6.4.4.15 top_scores

beam.Beam.top_scores

Definition at line 24 of file beam.py.

6.4.4.16 vocab_size

beam.Beam.vocab_size

Definition at line 15 of file beam.py.

The documentation for this class was generated from the following file:

- src/models/beam.py

6.5 generator_framework.Generator Class Reference

Public Member Functions

- def __init__ (self, model_name, input_shape, hidden_size, voc_path, feature_path, save_path, loss_↩ function='cross_entropy', optimizer='adam', lr=0.0001, embedding_size=300, seed=222)
- def compile (self)
- def initialize_optimizer (self)
- def train (self, data_path, validation_path, ann_path, epochs, batch_size, early_stopping_freq=6, val_batch↩ _size=1, beam_size=1, validation_metric='CIDEr')
- def train_on_batch (self, x, caption_lengths)
- def predict (self, data_path, batch_size=1, beam_size=1)
- def beam_search (self, batch, beam_size=1)
- def evaluate (self, data_path, ann_path, res_path, eval_path, batch_size=1, beam_size=1, metric='CIDEr')
- def load_model (self, path)
- def save_model (self, path)

Static Public Member Functions

- def post_process_predictions (predictions)

Public Attributes

- input_shape
- max_length
- embedding_size
- hidden_size
- save_path
- random_seed
- ixtoword
- vocab_path
- vocab_size
- feature_path
- encoded_features
- encoder
- decoder
- train_params
- model_name
- loss_string
- criterion
- optimizer_string
- encoder_optimizer
- decoder_optimizer
- lr
- framework_name
- device

6.5.1 Detailed Description

Definition at line 47 of file generator_framework.py.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 __init__()

```
def generator_framework.Generator.__init__ (
              self,
              model_name,
              input_shape,
              hidden_size,
              voc_path,
              feature_path,
              save_path,
              loss_function = 'cross_entropy',
              optimizer = 'adam',
              lr = 0.0001,
              embedding_size = 300,
              seed = 222 )
```

Definition at line 49 of file generator_framework.py.

```
49     def __init__(self,
```

```
50                 model_name,
51                 input_shape,
52                 hidden_size,
53                 voc_path,
54                 feature_path,
55                 save_path,
56                 loss_function='cross_entropy',
57                 optimizer='adam',
58                 lr=0.0001,
59                 embedding_size=300,
60                 seed=222):
61         # delete if not used in this class
62         self.input_shape = input_shape
63         self.max_length = self.input_shape[1]
64
65         self.embedding_size = embedding_size
66         self.hidden_size = hidden_size
67
68         self.save_path = save_path
69         self.random_seed = seed
70
71         self.wordtoix, self.ixtoword = load_vocabulary(voc_path)
72         self.vocab_path = voc_path
73         self.vocab_size = len(self.wordtoix)
74         self.feature_path = feature_path
75         self.encoded_features = load_visual_features(feature_path)
76
77         # initialize model as None
78         self.encoder = None
79         self.decoder = None
80         self.train_params = 0
81
82         self.model_name = model_name
83
84         # initialize loss function
85         self.loss_string = loss_function
86         self.criterion = loss_switcher(self.loss_string)()
87
88         # set up optimizer
89         self.optimizer_string = optimizer
90         self.encoder_optimizer = None  # not initialized
91         self.decoder_optimizer = None
92         self.lr = lr
93
94         # misc
95         self.framework_name = 'CaptionGeneratorFramework'
96         # gpu
97         self.device = torch.device("cuda:0"
98                                    if torch.cuda.is_available() else "cpu")
99         print('Device:', self.device)
100
```

6.5.3 Member Function Documentation

6.5.3.1 beam_search()

```
def generator_framework.Generator.beam_search (
              self,
              batch,
              beam_size = 1 )
```

```
Preform the beam search algorithm on a batch of images.

Parameters
----------
batch : list.
    List of encoded images.
beam_size : int.
    Size of beam. Default is 1.

Returns
-------
predictions : dict.
    key: image index, value: predicted caption.
```

Definition at line 468 of file generator_framework.py.

```
468     def beam_search(self, batch, beam_size=1):
469         """
470         Preform the beam search algorithm on a batch of images.
471 
472         Parameters
473         ----------
474         batch : list.
475             List of encoded images.
476         beam_size : int.
477             Size of beam. Default is 1.
478 
479         Returns
480         -------
481         predictions : dict.
482             key: image index, value: predicted caption.
483         """
484         # consider if it is possible to handle more than one sample at a time
485         # for instance more images, and/or predict on the entire beam
486         # initialization
487         batch_size = len(batch)
488         batch = torch.tensor(batch).to(self.device)
489 
490         global_images, encoded_images = self.encoder(batch)
491 
492         h_t, c_t = self.decoder.initialize_variables(batch_size * beam_size)
493 
494         # initialize beams as containing 1 caption
495         # need beams to keep track of original indices
496         beams = [Beam([g_image, enc_image],
497                       states=[h_t[:, i*beam_size: (i+1)*beam_size],
498                               c_t[:, i*beam_size: (i+1)*beam_size]],
499                       beam_size=beam_size,
500                       input_token=[self.wordtoix['startseq']],
501                       eos=self.wordtoix['endseq'],
502                       max_len=self.max_length,
503                       vocab_size=self.vocab_size,
504                       device=self.device,
505                       beam_id=i)
506                  for i, (g_image, enc_image) in
507                  enumerate(zip(global_images, encoded_images))]
508 
509         working_beams_idx = set([i for i in range(batch_size)])
510 
511         predictions = defaultdict(str)  # key: batch index, val: caption
512 
513         while True:
514             # find current batch_size aka number of beams
515             # counts unfinished beams
516             batch_size_t = sum(b.num_unfinished > 0 for b in beams)
517             if batch_size_t == 0:
518                 # all beams are done
519                 break
520 
521             # get sequences
522             sequences = torch.cat([b.get_sequences() for b in beams], dim=0)
523             # get images
524             global_images = torch.cat([b.get_global_image() for b in beams],
525                                       dim=0)
526             encoded_images = torch.cat([b.get_encoded_image() for b in beams],
527                                        dim=0)
528             images = [global_images, encoded_images]
529             # get states
530             h_t = torch.cat([b.get_hidden_states() for b in beams], dim=1)
531             c_t = torch.cat([b.get_cell_states() for b in beams], dim=1)
532 
533             # get predictions
534             x = [images, sequences]
535             y_predictions, h_t, c_t = self.decoder(x, (h_t, c_t))
536             # y_predictions (M*N, voc_size)
537             y_predictions = torch.log_softmax(y_predictions, dim=1)
538             # higher log_prob --> higher pob
539 
540             remove_idx = set()
541             for i in range(batch_size):
542                 start_idx = i * beam_size
543                 if i in working_beams_idx:
544                     # feed right predictions to right beams
545                     b = beams[i]
546                     end_idx = start_idx + b.num_unfinished
547                     preds = y_predictions[start_idx: end_idx]
548                     # update beam with predictions
549                     b.update(preds,
550                              h_t[:, start_idx: end_idx],
551                              c_t[:, start_idx: end_idx])
552 
553                     if b.has_best_sequence():
```

```
554                         # add to finished predictions
555                         predictions[i] = \
556                             ' '.join([self.ixtoword[w]
557                                       for w in b.get_best_sequence()])
558                         remove_idx.add(i)
559             # remove idx of finished beams
560             working_beams_idx = set(idx for idx in working_beams_idx
561                                     if idx not in remove_idx)
562 
563         # should be removed when finished
564         assert len(predictions) == batch_size, \
565             "The number of predictions does not match the number of images"
566         return predictions
567 
```

6.5.3.2 compile()

```
def generator_framework.Generator.compile (
              self )
```

Builds the model.

Definition at line 101 of file generator_framework.py.

```
101     def compile(self):
102         """
103         Builds the model.
104         """
105         # initialize model
106         self.decoder, self.encoder = model_switcher(self.model_name)
107         self.encoder = self.encoder(self.input_shape[0],
108                                     self.hidden_size,
109                                     self.embedding_size)
110         self.decoder = self.decoder(self.input_shape,
111                                     self.hidden_size,
112                                     self.vocab_size,
113                                     self.device,
114                                     embedding_size=self.embedding_size,
115                                     seed=self.random_seed)
116         print(self.encoder)
117         print(self.decoder)
118         self.train_params += sum(p.numel() for p in self.decoder.parameters()
119                                  if p.requires_grad)
120         self.train_params += sum(p.numel() for p in self.encoder.parameters()
121                                  if p.requires_grad)
122         self.encoder.to(self.device)
123         self.decoder.to(self.device)
124         print('Trainable Parameters:', self.train_params, '\n\n\n\n')
125         self.initialize_optimizer()  # initialize optimizer
126 
```

6.5.3.3 evaluate()

```
def generator_framework.Generator.evaluate (
              self,
              data_path,
              ann_path,
              res_path,
              eval_path,
              batch_size = 1,
              beam_size = 1,
              metric = 'CIDEr' )
```

```
Function to evaluate model on data from data_path
using evaluation metric metric.

Parameters
----------
data_path : Path or str.
    Validation or test set.
ann_path : Path or str.
    Location of annotation file corresponding to val or test set.
res_path : Path or str.
    File that the results will be saved in.
eval_path : Path or str.
    File where all metric scores will be saved in.
batch_size : int.
    The number of images to perform inference on simultaneously.
beam_size : int.
    Size of beam.
metric : str.
    Automatic evaluation metric. Compatible metrics are
    {Bleu_1, Bleu2, Bleu_3, Bleu_4, METEOR, ROUGE_L, CIDEr, SPICE}.

Returns
-------
Automatic evaluation score.
```

Definition at line 568 of file generator_framework.py.

```
568     def evaluate(self, data_path, ann_path, res_path, eval_path, batch_size=1,
569                  beam_size=1, metric='CIDEr'):
570         """
571         Function to evaluate model on data from data_path
572         using evaluation metric metric.
573 
574         Parameters
575         ----------
576         data_path : Path or str.
577             Validation or test set.
578         ann_path : Path or str.
579             Location of annotation file corresponding to val or test set.
580         res_path : Path or str.
581             File that the results will be saved in.
582         eval_path : Path or str.
583             File where all metric scores will be saved in.
584         batch_size : int.
585             The number of images to perform inference on simultaneously.
586         beam_size : int.
587             Size of beam.
588         metric : str.
589             Automatic evaluation metric. Compatible metrics are
590             {Bleu_1, Bleu2, Bleu_3, Bleu_4, METEOR, ROUGE_L, CIDEr, SPICE}.
591 
592         Returns
593         -------
594         Automatic evaluation score.
595         """
596         data_path = Path(data_path)
597         ann_path = Path(ann_path)
598         res_path = Path(res_path)
599         eval_path = Path(eval_path)
600         print("Evaluating model ...")
601         # get models predictions
602         predictions = self.predict(data_path,
603                                    batch_size=batch_size,
604                                    beam_size=beam_size)
605         print("Finished predicting")
606         # save predictions to res_path which is .json
607         with open(res_path, 'w') as res_file:
608             json.dump(predictions, res_file)
609 
610         coco = COCO(str(ann_path))
611         coco_res = coco.loadRes(str(res_path))
612         coco_eval = COCOEvalCap(coco, coco_res)
613         coco_eval.params['image_id'] = coco_res.getImgIds()
614         coco_eval.evaluate()
615 
616         # save evaluations to eval_path which is .json
617         with open(eval_path, 'w') as eval_file:
618             json.dump(coco_eval.eval, eval_file)
619 
620         return coco_eval.eval[metric]
621 
```

6.5.3.4 initialize_optimizer()

```
def generator_framework.Generator.initialize_optimizer (
              self )
```

```
Initializes the optimizer.
After this is called, optimizer will no longer be None.
```

Definition at line 127 of file generator_framework.py.

```
127     def initialize_optimizer(self):
128         """
129         Initializes the optimizer.
130         After this is called, optimizer will no longer be None.
131         """
132         self.encoder_optimizer = optimizer_switcher(self.optimizer_string)(
133             self.encoder.parameters(), self.lr)
134         self.decoder_optimizer = optimizer_switcher(self.optimizer_string)(
135             self.decoder.parameters(), self.lr)
136
```

6.5.3.5 load_model()

```
def generator_framework.Generator.load_model (
              self,
              path )
```

Definition at line 622 of file generator_framework.py.

```
622     def load_model(self, path):
623         checkpoint = torch.load(path)
624         self.encoder = checkpoint['encoder']
625         self.decoder = checkpoint['decoder']
626         self.encoder_optimizer = checkpoint['enc_optimizer']
627         self.decoder_optimizer = checkpoint['dec_optimizer']
628         self.encoder.eval()
629         self.decoder.eval()
630         print('Loaded checkpoint at:', path)
631         print(self.encoder)
632         print(self.decoder)
633
```

6.5.3.6 post_process_predictions()

```
def generator_framework.Generator.post_process_predictions (
              predictions )  [static]
```

Definition at line 457 of file generator_framework.py.

```
457     def post_process_predictions(predictions):
458         # helper function, consider moving to utils
459         # remove startseq and endseq token from sequence
460         processed = []
461         for prediction in predictions.values():
462             prediction = prediction.replace('startseq', '')
463             prediction = prediction.replace('endseq', '')
464             prediction = prediction.strip()
465             processed.append(prediction)
466         return processed
467
```

6.5.3.7 predict()

```
def generator_framework.Generator.predict (
                self,
                data_path,
                batch_size = 1,
                beam_size = 1 )
```

```
Function to make self.model make predictions given some data.

Parameters
----------
data_path : Path or str.
    Path to csv file containing the test set.
batch_size : int.
    The number of images to predict on simultaneously. Default 1.
beam_size : int.
    Default is 1, which is the same as doing greedy inference.

Returns
-------
predicted_captions : list.
    Dictionary image_name as keys and predicted captions through
    beam search are the values.
```

Definition at line 389 of file generator_framework.py.

```
389     def predict(self, data_path, batch_size=1, beam_size=1):
390         """
391         Function to make self.model make predictions given some data.
392 
393         Parameters
394         ----------
395         data_path : Path or str.
396             Path to csv file containing the test set.
397         batch_size : int.
398             The number of images to predict on simultaneously. Default 1.
399         beam_size : int.
400             Default is 1, which is the same as doing greedy inference.
401 
402         Returns
403         -------
404         predicted_captions : list.
405             Dictionary image_name as keys and predicted captions through
406             beam search are the values.
407         """
408         data_path = Path(data_path)
409         self.encoder.eval()  # put model in evaluation mode
410         self.decoder.eval()
411 
412         data_df = pd.read_csv(data_path)
413         data_df = data_df.reset_index(drop=True)
414 
415         predicted_captions = []
416 
417         num_images = len(data_df)
418         if batch_size > num_images:
419             batch_size = num_images
420         steps = int(np.ceil(num_images / batch_size))
421         prev_batch_idx = 0
422         for i in range(steps):
423             # create input batch
424             end_batch_idx = min(prev_batch_idx + batch_size, num_images)
425             batch_df = data_df.iloc[prev_batch_idx: end_batch_idx, :]
426             image_ids = batch_df.loc[:, 'image_id'].to_numpy()
427             image_names = batch_df.loc[:, 'image_name'].to_numpy()
428 
429             enc_images = []
430             for image_id, image_name in zip(image_ids, image_names):
431                 # get encoded features for this batch
432                 pred_dict = {
433                     "image_id": int(image_id),
434                     "caption": ""
435                 }
436                 predicted_captions.append(pred_dict)
437                 enc_images.append(self.encoded_features[image_name])
438 
439             # get full sentence predictions from beam_search algorithm
```

```
440             predictions = self.beam_search(enc_images, beam_size=beam_size)
441             predictions = self.post_process_predictions(predictions)
442
443             # put predictions in the right pred_dict
444             counter = 0
445             for idx in range(prev_batch_idx, end_batch_idx):
446                 predicted_captions[idx]["caption"] = predictions[counter]
447                 counter += 1
448
449             # verbose
450             print('Batch step', i + 1)
451             # update prev_batch_idx
452             prev_batch_idx = end_batch_idx
453
454         return predicted_captions
455
```

6.5.3.8 save_model()

```
def generator_framework.Generator.save_model (
                self,
                path )
```

Definition at line 634 of file generator_framework.py.

```
634     def save_model(self, path):
635         path = Path(path)
636         assert path.is_dir()
637         path1 = path.joinpath(self.model_name + '_decoder.pth')
638         torch.save(self.decoder.state_dict(), path1)
639         path2 = path.joinpath(self.model_name + '_encoder.pth')
640         torch.save(self.encoder.state_dict(), path2)
```

6.5.3.9 train()

```
def generator_framework.Generator.train (
                self,
                data_path,
                validation_path,
                ann_path,
                epochs,
                batch_size,
                early_stopping_freq = 6,
                val_batch_size = 1,
                beam_size = 1,
                validation_metric = 'CIDEr' )
```

```
Method for training the model.

Parameters
----------
data_path : Path or str.
    Path to trainingset file (*.csv).
validation_path : Path or str.
    Path to validationset file (*.csv).
ann_path : Path or str.
    Path to the validation annotation file (*.json)
epochs : int.
    Max number of epochs to continue training the model for.
batch_size : int.
    Mini batch size.
early_stopping_freq : int.
```

```
    If no improvements over this number of epochs then stop training.
val_batch_size : int.
    The number of images to do inference on simultaneously.
    Default is 1.
beam_size : int.
    Beam size for validation. Default is 1.
validation_metric : str.
    Which automatic text evaluation metric to use for validation.
    Metrics = {'CIDEr', 'METEOR', 'SPICE', 'ROUGE_L',
    'Bleu_1', 'Bleu_2', 'Bleu_3', 'Bleu_4'}.
    Defualt value is 'CIDEr'.

Returns
-------
Saves the model and checkpoints to its own folder. Then writes a log
with all necessary information about the model and training.
```

Definition at line 137 of file generator_framework.py.

```
137     def train(self,
138               data_path,
139               validation_path,
140               ann_path,
141               epochs,
142               batch_size,
143               early_stopping_freq=6,
144               val_batch_size=1,
145               beam_size=1,
146               validation_metric='CIDEr'):
147         """
148         Method for training the model.
149 
150         Parameters
151         ----------
152         data_path : Path or str.
153             Path to trainingset file (*.csv).
154         validation_path : Path or str.
155             Path to validationset file (*.csv).
156         ann_path : Path or str.
157             Path to the validation annotation file (*.json)
158         epochs : int.
159             Max number of epochs to continue training the model for.
160         batch_size : int.
161             Mini batch size.
162         early_stopping_freq : int.
163             If no improvements over this number of epochs then stop training.
164         val_batch_size : int.
165             The number of images to do inference on simultaneously.
166             Default is 1.
167         beam_size : int.
168             Beam size for validation. Default is 1.
169         validation_metric : str.
170             Which automatic text evaluation metric to use for validation.
171             Metrics = {'CIDEr', 'METEOR', 'SPICE', 'ROUGE_L',
172             'Bleu_1', 'Bleu_2', 'Bleu_3', 'Bleu_4'}.
173             Defualt value is 'CIDEr'.
174 
175         Returns
176         -------
177         Saves the model and checkpoints to its own folder. Then writes a log
178         with all necessary information about the model and training.
179         """
180         data_path = Path(data_path)
181         validation_path = Path(validation_path)
182         ann_path = Path(ann_path)
183         train_df = pd.read_csv(data_path)
184 
185         training_history = {
186             'encoder': str(self.encoder),
187             'decoder': str(self.decoder),
188             'trainable_parameters': str(self.train_params),
189             'lr': str(self.lr),
190             'optimizer': self.optimizer_string,
191             'loss': self.loss_string,
192             'model_name': self.model_name,
193             'history': [],
194             'voc_size': str(self.vocab_size),
195             'voc_path': str(self.vocab_path),
196             'feature_path': str(self.feature_path),
197             'train_path': str(data_path),
198             'epochs': str(epochs),
199             'batch_size': str(batch_size),
200             'training_time': str(0),
```

```
201             'model_save_path': ''
202         }
203 
204         steps_per_epoch = len(train_df) // batch_size
205 
206         train_generator = data_generator(train_df, batch_size,
207                                          steps_per_epoch,
208                                          self.wordtoix,
209                                          self.encoded_features,
210                                          seed=self.random_seed)
211 
212         start_time = time()
213 
214         date_time_obj = datetime.now()
215         timestamp_str = date_time_obj.strftime("%d-%b-%Y_(%H:%M:%S)")
216         directory = self.save_path.joinpath(self.model_name + '_'
217                                             + timestamp_str)
218         # check that directory is a Directory if not make it one
219         if not directory.is_dir():
220             directory.mkdir()
221 
222         best_val_score = -1  # all metric give positive scores
223         best_path = None
224         epochs_since_improvement = 0
225 
226         for e in range(1, epochs + 1):
227             # early stopping
228             if epochs_since_improvement == early_stopping_freq:
229                 print('Training TERMINATED!\nNo Improvements for '
230                       + str(early_stopping_freq) + ' epochs.')
231                 break
232 
233             self.encoder.train()  # put model in train mode
234             self.decoder.train()
235 
236             print('Epoch: #' + str(e))
237             batch_history = []
238             for s in range(1, steps_per_epoch + 1):
239                 print('Step: #' + str(s) + '/' + str(steps_per_epoch))
240                 # zero the gradient buffers
241                 self.encoder_optimizer.zero_grad()
242                 self.decoder_optimizer.zero_grad()
243 
244                 # get minibatch from data generator
245                 x, caption_lengths = next(train_generator)
246 
247                 loss_num = self.train_on_batch(x, caption_lengths)
248                 batch_history.append(loss_num)
249 
250             # add the mean loss of the epoch to the training history
251             training_history['history'].append(np.mean(
252                 np.array(batch_history)))
253 
254             # validation here
255             # consider just overwriting the same file
256             eval_path = directory.joinpath('captions_eval_' + str(e) + '.json')
257             res_path = directory.joinpath('captions_result_' + str(e)
258                                           + '.json')
259             metric_score = self.evaluate(validation_path,
260                                          ann_path,
261                                          res_path,
262                                          eval_path,
263                                          batch_size=val_batch_size,
264                                          beam_size=beam_size,
265                                          metric=validation_metric)
266             # save model checkpoint
267             is_best = metric_score > best_val_score
268             best_val_score = max(metric_score, best_val_score)
269             tmp_model_path = save_checkpoint(directory,
270                                              epoch=e,
271                                              epochs_since_improvement=
272                                              epochs_since_improvement,
273                                              encoder=self.encoder,
274                                              decoder=self.decoder,
275                                              enc_optimizer=
276                                              self.encoder_optimizer,
277                                              dec_optimizer=
278                                              self.decoder_optimizer,
279                                              cider=metric_score,
280                                              is_best=is_best)
281             if tmp_model_path:
282                 best_path = tmp_model_path
283 
284             if is_best:
285                 epochs_since_improvement = 0
286             else:
287                 epochs_since_improvement += 1
```

```
288
289         # end of training
290         training_time = timedelta(seconds=int(time() - start_time))  # seconds
291         d = datetime(1, 1, 1) + training_time
292         training_time = "%d:%d:%d:%d" % (d.day-1, d.hour, d.minute, d.second)
293         training_history['training_time'] = str(training_time)
294         training_history['model_save_path'] = str(best_path)
295         train_path = directory.joinpath(self.model_name + '_log.txt')
296         # save model to file
297         self.save_model(directory)
298         # save log to file
299         save_training_log(train_path, training_history)
300
```

6.5.3.10 train_on_batch()

```
def generator_framework.Generator.train_on_batch (
                self,
                x,
                caption_lengths )


Parameters
----------
x : list.
    batch of images and captions.
target : tensor.

caption_lengths

Returns
-------
```

Definition at line 301 of file generator_framework.py.

```
301     def train_on_batch(self, x, caption_lengths):
302         """
303
304         Parameters
305         ----------
306         x : list.
307             batch of images and captions.
308         target : tensor.
309
310         caption_lengths
311
312         Returns
313         -------
314
315         """
316         # unpack batch
317         input_img, input_w = x
318         # move to device
319         input_img = input_img.to(self.device)
320         input_w = input_w.to(self.device)
321
322         # encode images
323         global_images, enc_images = self.encoder(input_img)
324         # (batch_size, embedding_size) (batch, 512) global_images
325         # (batch_size, region_size, hidden_size) (batch, 64, 512) encoded_imgs
326
327         # sort batches by caption length descending, this way the whole
328         # batch_size_t will be correct
329         # convert to tensor
330         caption_lengths = torch.from_numpy(caption_lengths)
331         caption_lengths, sort_idx = caption_lengths.sort(dim=0,
332                                                          descending=True)
333         input_w = input_w[sort_idx]  # (batch_size, max_len)
334         global_images = global_images[sort_idx]  # (batch_size, embedding_size)
335         enc_images = enc_images[sort_idx]  # (batch_size, 64, 1536)
336
337         target = input_w[:, 1:]  # sorted targets
338         target = target.to(self.device)
339
340         batch_size = enc_images.size()[0]
```

```
341
342         # we do not want to predict the last endseq token
343         decoding_lengths = np.copy(caption_lengths)
344         decoding_lengths = (decoding_lengths - 1)
345         max_batch_length = max(decoding_lengths)
346
347         predictions = torch.zeros(batch_size,
348                                   self.max_length,
349                                   self.vocab_size)
350
351         h_t, c_t = self.decoder.initialize_variables(batch_size)
352
353         for timestep in range(max_batch_length):
354             batch_size_t = sum([lens > timestep for lens in decoding_lengths])
355             # x: [input_img, input_w]
356             # input_img: [global_image, encoded_image]
357             # image features does not vary over time
358             input_image_t = [global_images[:batch_size_t],
359                              enc_images[:batch_size_t]]
360             x_t = [input_image_t, input_w[:batch_size_t, timestep]]
361
362             pt, h_t, c_t = self.decoder(x_t,
363                                         (h_t[:, :batch_size_t],
364                                          c_t[:, :batch_size_t]))
365             predictions[:batch_size_t, timestep, :] = pt
366
367         # loop finished
368         # pack padded sequences
369         output = pack_padded_sequence(predictions,
370                                       decoding_lengths,
371                                       batch_first=True)[0]
372         target = pack_padded_sequence(target,
373                                       decoding_lengths,
374                                       batch_first=True)[0]
375
376         # get loss
377         loss = self.criterion(output, target)
378         loss_num = loss.item()
379         print('loss', '(' + self.optimizer_string + '):', loss_num)
380
381         # backpropagate
382         loss.backward()
383         # update weights
384         self.encoder_optimizer.step()
385         self.decoder_optimizer.step()
386
387         return loss_num
388
```

6.5.4 Member Data Documentation

6.5.4.1 criterion

`generator_framework.Generator.criterion`

Definition at line 75 of file generator_framework.py.

6.5.4.2 decoder

`generator_framework.Generator.decoder`

Definition at line 68 of file generator_framework.py.

6.5.4.3 decoder_optimizer

`generator_framework.Generator.decoder_optimizer`

Definition at line 80 of file generator_framework.py.

6.5.4.4 device

`generator_framework.Generator.device`

Definition at line 86 of file generator_framework.py.

6.5.4.5 embedding_size

`generator_framework.Generator.embedding_size`

Definition at line 54 of file generator_framework.py.

6.5.4.6 encoded_features

`generator_framework.Generator.encoded_features`

Definition at line 64 of file generator_framework.py.

6.5.4.7 encoder

`generator_framework.Generator.encoder`

Definition at line 67 of file generator_framework.py.

6.5.4.8 encoder_optimizer

`generator_framework.Generator.encoder_optimizer`

Definition at line 79 of file generator_framework.py.

6.5.4.9 feature_path

`generator_framework.Generator.feature_path`

Definition at line 63 of file generator_framework.py.

6.5.4.10 framework_name

`generator_framework.Generator.framework_name`

Definition at line 84 of file generator_framework.py.

6.5.4.11 hidden_size

`generator_framework.Generator.hidden_size`

Definition at line 55 of file generator_framework.py.

6.5.4.12 input_shape

`generator_framework.Generator.input_shape`

Definition at line 51 of file generator_framework.py.

6.5.4.13 ixtoword

`generator_framework.Generator.ixtoword`

Definition at line 60 of file generator_framework.py.

6.5.4.14 loss_string

`generator_framework.Generator.loss_string`

Definition at line 74 of file generator_framework.py.

6.5.4.15 lr

generator_framework.Generator.lr

Definition at line 81 of file generator_framework.py.

6.5.4.16 max_length

generator_framework.Generator.max_length

Definition at line 52 of file generator_framework.py.

6.5.4.17 model_name

generator_framework.Generator.model_name

Definition at line 71 of file generator_framework.py.

6.5.4.18 optimizer_string

generator_framework.Generator.optimizer_string

Definition at line 78 of file generator_framework.py.

6.5.4.19 random_seed

generator_framework.Generator.random_seed

Definition at line 58 of file generator_framework.py.

6.5.4.20 save_path

generator_framework.Generator.save_path

Definition at line 57 of file generator_framework.py.

6.5.4.21 train_params

`generator_framework.Generator.train_params`

Definition at line 69 of file generator_framework.py.

6.5.4.22 vocab_path

`generator_framework.Generator.vocab_path`

Definition at line 61 of file generator_framework.py.

6.5.4.23 vocab_size

`generator_framework.Generator.vocab_size`

Definition at line 62 of file generator_framework.py.

The documentation for this class was generated from the following file:

- src/models/generator_framework.py

6.6 custom_layers.ImageEncoder Class Reference

Inheritance diagram for custom_layers.ImageEncoder:

Public Member Functions

- def __init__ (self, input_shape, hidden_size, embedding_size)
- def forward (self, x)

Public Attributes

- average_pool
- v_affine
- global_affine

6.6.1 Detailed Description

Definition at line 56 of file custom_layers.py.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 __init__()

```
def custom_layers.ImageEncoder.__init__ (
              self,
              input_shape,
              hidden_size,
              embedding_size )
```

Definition at line 58 of file custom_layers.py.

```
58     def __init__(self, input_shape, hidden_size, embedding_size):
59         super(ImageEncoder, self).__init__()
60         self.average_pool = nn.AvgPool2d(input_shape[0])
61         # affine transformation of attention features
62         self.v_affine = nn.Linear(input_shape[2], hidden_size)
63         # affine transformation of global image features
64         self.global_affine = nn.Linear(input_shape[2], embedding_size)
65
```

6.6.3 Member Function Documentation

6.6.3.1 forward()

```
def custom_layers.ImageEncoder.forward (
              self,
              x )
```

Definition at line 66 of file custom_layers.py.

```
66     def forward(self, x):
67         # x = V, (batch_size, 8, 8, 1536)
68         # print('Image Encoder')
69         input_shape = x.size()
70         pixels = input_shape[1] * input_shape[2]  # 8x8 = 64
71         global_image = self.average_pool(x).view(input_shape[0], -1)
72         inputs = x.view(input_shape[0], pixels, input_shape[3])
73
74         # transform
75         global_image = self.global_affine(global_image)
76         inputs = self.v_affine(inputs)
77
78         return global_image, inputs
79
80
```

6.6.4 Member Data Documentation

6.6.4.1 average_pool

`custom_layers.ImageEncoder.average_pool`

Definition at line 60 of file custom_layers.py.

6.6.4.2 global_affine

`custom_layers.ImageEncoder.global_affine`

Definition at line 64 of file custom_layers.py.

6.6.4.3 v_affine

`custom_layers.ImageEncoder.v_affine`

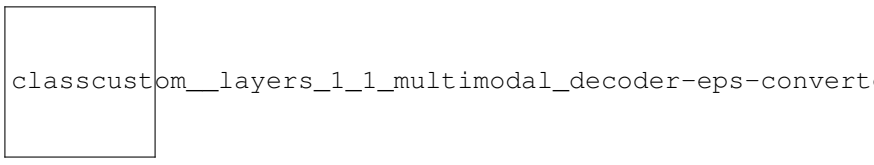
Definition at line 62 of file custom_layers.py.

The documentation for this class was generated from the following file:

- src/models/custom_layers.py

6.7 custom_layers.MultimodalDecoder Class Reference

Inheritance diagram for custom_layers.MultimodalDecoder:



Public Member Functions

- def __init__ (self, input_shape, hidden_size, n=0)
- def forward (self, x)

Public Attributes

- layers
- output_layer

6.7.1 Detailed Description

Definition at line 81 of file custom_layers.py.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 __init__()

```
def custom_layers.MultimodalDecoder.__init__ (
                self,
                input_shape,
                hidden_size,
                n = 0 )
```

Definition at line 83 of file custom_layers.py.

```
83     def __init__(self, input_shape, hidden_size, n=0):
84         super(MultimodalDecoder, self).__init__()
85         self.layers = []
86         if n:
87             new_input_shape = input_shape*2
88             self.layers.append(nn.Linear(input_shape, input_shape))
89             input_shape = new_input_shape
90         else:
91             n = 1
92 
93         for _ in range(n - 1):
94             self.layers.append(nn.Linear(input_shape, input_shape))
95 
96         # output layer, this is the only layer if n=0
97         self.output_layer = nn.Linear(input_shape, hidden_size)
98 
```

6.7.3 Member Function Documentation

6.7.3.1 forward()

```
def custom_layers.MultimodalDecoder.forward (
                self,
                x )
```

Definition at line 99 of file custom_layers.py.

```
99     def forward(self, x):
100         # x: context_vector + h_t (batch_size, hidden_size)
101         # print('Multimodal Decoder')
102         concat = x
103         for layer in self.layers:
104             y = F.relu(layer(concat))
105             concat = torch.cat((concat, y), dim=1)
106 
107         # softmax on output
108         y = F.softmax(self.output_layer(concat))
109         return y
110 
111 
```

6.7.4 Member Data Documentation

6.7.4.1 layers

`custom_layers.MultimodalDecoder.layers`

Definition at line 85 of file custom_layers.py.

6.7.4.2 output_layer

`custom_layers.MultimodalDecoder.output_layer`

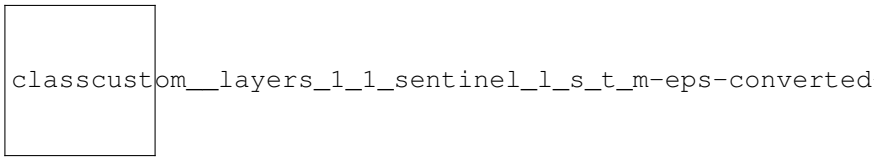
Definition at line 97 of file custom_layers.py.

The documentation for this class was generated from the following file:

- src/models/custom_layers.py

6.8 custom_layers.SentinelLSTM Class Reference

Inheritance diagram for custom_layers.SentinelLSTM:



Public Member Functions

- def __init__ (self, input_size, hidden_size, n=0)
- def forward (self, x, states)

Public Attributes

- n
- lstm_cells
- lstm_kernel
- x_gate
- h_gate

6.8.1 Detailed Description

Definition at line 6 of file custom_layers.py.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 __init__()

```
def custom_layers.SentinelLSTM.__init__ (
              self,
              input_size,
              hidden_size,
              n = 0 )
```

Definition at line 8 of file custom_layers.py.

```
8     def __init__(self, input_size, hidden_size, n=0):
9         super(SentinelLSTM, self).__init__()
10         # NB! there is a difference between LSTMCell and LSTM.
11         # LSTM is notably much quicker
12         self.n = n
13         self.lstm_cells = []
14         if n:
15             inp_size = hidden_size
16         else:
17             inp_size = input_size
18         self.lstm_kernel = nn.LSTMCell(inp_size, hidden_size)
19         self.x_gate = nn.Linear(inp_size, hidden_size)
20         self.h_gate = nn.Linear(hidden_size, hidden_size)
21
22         inp_size = input_size
23         hid_size = hidden_size
24         for _ in range(self.n):
25             self.lstm_cells.append(nn.LSTMCell(inp_size, hid_size))
26             inp_size = hid_size
27
```

6.8.3 Member Function Documentation

6.8.3.1 forward()

```
def custom_layers.SentinelLSTM.forward (
              self,
              x,
              states )
```

Definition at line 28 of file custom_layers.py.

```
28     def forward(self, x, states):
29         # print('Sentinel LSTM')
30         # remember old states
31         h_tm1, c_tm1 = states
32         # new states lists
33         hs = torch.zeros(h_tm1.size())
34         cs = torch.zeros(c_tm1.size())
35         inputs = x
36         for i in range(self.n):
37             # feed layers the correct h and c states
38             h, c = self.lstm_cells[i](inputs, (h_tm1[i], c_tm1[i]))
```

```
39             hs[i] = h
40             cs[i] = c
41             # add residual
42             inputs = h + inputs
43
44         # get new states
45         h_t, c_t = self.lstm_kernel(inputs, (h_tm1[-1], c_tm1[-1]))
46         hs[-1] = h_t
47         cs[-1] = c_t
48
49         # compute sentinel vector
50         # could either concat h_tm1 with x or have to gates
51         sv = torch.sigmoid(self.h_gate(h_tm1[-1]) + self.x_gate(inputs))
52         s_t = sv * torch.tanh(c_t)
53         return hs, cs, h_t, s_t
54
55
```

6.8.4 Member Data Documentation

6.8.4.1 h_gate

`custom_layers.SentinelLSTM.h_gate`

Definition at line 20 of file custom_layers.py.

6.8.4.2 lstm_cells

`custom_layers.SentinelLSTM.lstm_cells`

Definition at line 13 of file custom_layers.py.

6.8.4.3 lstm_kernel

`custom_layers.SentinelLSTM.lstm_kernel`

Definition at line 18 of file custom_layers.py.

6.8.4.4 n

`custom_layers.SentinelLSTM.n`

Definition at line 12 of file custom_layers.py.

6.8.4.5 x_gate

`custom_layers.SentinelLSTM.x_gate`

Definition at line 19 of file custom_layers.py.

The documentation for this class was generated from the following file:

- src/models/custom_layers.py

Chapter 7

File Documentation

7.1 src/__init__.py File Reference

Namespaces

- src

7.2 src/data/data_cleaning.py File Reference

Namespaces

- data_cleaning

Functions

- def data_cleaning.basic_data_cleaning (df_path, save_path, voc_save_path)
- def data_cleaning.replace_uncommon_words (caption_df, corpus)
- def data_cleaning.remove_too_many_unk (caption_df)
- def data_cleaning.remove_bad_captions (caption_df)
- def data_cleaning.is_all_one_letter (caption, letter)
- def data_cleaning.number_to_word (word)

Variables

- data_cleaning.ROOT_PATH = Path(__file__).absolute().parents[2]
- int data_cleaning.THRESHOLD = 3
- float data_cleaning.UNK_PERCENTAGE = 0.4

7.3 src/data/data_generator.py File Reference

Namespaces

- data_generator

Functions

- def data_generator.data_generator (data_df, batch_size, steps_per_epoch, wordtoix, features, seed=2222)
- def data_generator.get_image (visual_features, data_df, i)
- def data_generator.get_caption (data_df, i)
- def data_generator.to_categorical (y, num_classes=None, dtype='float32')
- def data_generator.pad_sequences (sequences, maxlen)

Variables

- data_generator.ROOT_PATH = Path(__file__).absolute().parents[2]

7.4 src/data/handle_karpathy_split.py File Reference

Namespaces

- handle_karpathy_split

Functions

- def handle_karpathy_split.order_raw_data_and_move_to_interim (data_path, dataset, ann_path)
- def handle_karpathy_split.initialize_full_dict ()
- def handle_karpathy_split.initialize_ann_dict ()

Variables

- handle_karpathy_split.ROOT_PATH = Path(__file__).absolute().parents[2]

7.5 src/data/make_dataset.py File Reference

Namespaces

- make_dataset

Variables

- make_dataset.ROOT_PATH = Path(__file__).absolute().parents[2]
- make_dataset.parser = argparse.ArgumentParser()
- make_dataset.type
- make_dataset.str
- make_dataset.default
- make_dataset.help
- make_dataset.bool
- make_dataset.args = vars(parser.parse_args())
- make_dataset.raw_path = ROOT_PATH.joinpath('data', 'raw')
- make_dataset.interim_path = ROOT_PATH.joinpath('data', 'interim')
- make_dataset.processed_path = ROOT_PATH.joinpath('data', 'processed')
- make_dataset.ann_path_ = processed_path.joinpath('annotations')
- make_dataset.dataset_ = args['dataset']
- make_dataset.data_path_ = raw_path.joinpath('dataset_' + dataset_ + '.json')
- make_dataset.output_path_ = interim_path.joinpath(dataset_)
- list make_dataset.splits_ = ['train', 'val']
- make_dataset.df_path_ = interim_path.joinpath(dataset_ + '_train.csv')
- make_dataset.save_path_ = interim_path.joinpath(dataset_ + '_train_clean.csv')
- make_dataset.voc_save_path_ = interim_path.joinpath(dataset_ + '_vocabulary.csv')

7.6 src/data/preprocess_coco.py File Reference

Namespaces

- preprocess_coco

Functions

- def preprocess_coco.find_captions (captions, image_id)
- def preprocess_coco.make_dataframe (data_path)
- def preprocess_coco.preprocess_coco (data_path, output_path, splits)

Variables

- preprocess_coco.ROOT_PATH = Path(__file__).absolute().parents[2]

7.7 src/data/reduce_images_in_dataset.py File Reference

Namespaces

- reduce_images_in_dataset

7.8 src/data/split_flickr8k.py File Reference

Namespaces

- split_flickr8k

Functions

- def split_flickr8k.make_train_val_test_split (df_path, split_paths, save_path)

Variables

- split_flickr8k.ROOT_PATH = Path(__file__).absolute().parents[2]

7.9 src/data/subset_splits.py File Reference

Namespaces

- subset_splits

7.10 src/data/text_to_csv.py File Reference

Namespaces

- text_to_csv

Functions

- def text_to_csv.text_to_csv (file_path, save_path)

Variables

- text_to_csv.ROOT_PATH = Path(__file__).absolute().parents[2]

7.11 src/data/utils.py File Reference

Namespaces

- utils

Functions

- def utils.max_length_caption (df_path)
- def utils.load_vocabulary (voc_path)

Variables

- utils.ROOT_PATH = Path(__file__).absolute().parents[2]

7.12 src/models/utils.py File Reference

Namespaces

- utils

Functions

- def utils.save_checkpoint (directory, epoch, epochs_since_improvement, encoder, decoder, enc_optimizer, dec_optimizer, cider, is_best)
- def utils.save_training_log (path, training_history)

7.13 src/features/build_features.py File Reference

Namespaces

- build_features

Variables

- build_features.ROOT_PATH = Path(__file__).absolute().parents[2]
- tuple build_features.DIMENSIONS = (299, 299, 3)
- build_features.parser = argparse.ArgumentParser()
- build_features.type
- build_features.bool
- build_features.default
- build_features.help
- build_features.nargs
- build_features.str
- build_features.int
- build_features.args = vars(parser.parse_args())
- build_features.dataset_ = args['dataset']
- build_features.new_dims_ = args['new_image_size']
- tuple build_features.dims = DIMENSIONS[:2]
- build_features.raw_path = ROOT_PATH.joinpath('data', 'raw')
- build_features.interim_path = ROOT_PATH.joinpath('data', 'interim')
- build_features.processed_path = ROOT_PATH.joinpath('data', 'processed')
- build_features.image_path_ = raw_path.joinpath(dataset_, 'Images')
- build_features.save_path_ = interim_path.joinpath(dataset_, 'Images')
- build_features.output_layer_dim_ = args['output_layer_idx']
- build_features.vis_att_ = args['visual_attention']
- build_features.split_set_path_ = interim_path.joinpath(dataset_, dataset_ + '_full.csv')
- build_features.vis_att

7.14 src/features/glove_embeddings.py File Reference

Namespaces

- glove_embeddings

Functions

- def glove_embeddings.load_glove_vectors (glove_path)
- def glove_embeddings.embeddings_matrix (vocab_size, wordtoix, embeddings_index, embedding_dim)

7.15 src/features/resize_images.py File Reference

Namespaces

- resize_images

Functions

- def resize_images.resize_images (image_path, save_path, new_dims)

Variables

- resize_images.ROOT_PATH = Path(__file__).absolute().parents[2]

7.16 src/features/Resnet_features.py File Reference

Namespaces

- Resnet_features

Functions

- def Resnet_features.load_pre_trained_model (output_layer_idx)
- def Resnet_features.load_inception ()
- def Resnet_features.encode (image, model)
- def Resnet_features.encode_vis_att (image, model)
- def Resnet_features.extract_image_features (image_path, save_path, split_set_path, output_layer_idx, vis↩ _att=True)
- def Resnet_features.load_visual_features (feature_path)

Variables

- Resnet_features.ROOT_PATH = Path(__file__).absolute().parents[2]

7.17 src/models/beam.py File Reference

Classes

- class beam.Beam

Namespaces

- beam

7.18 src/models/custom_layers.py File Reference

Classes

- class custom_layers.SentinelLSTM
- class custom_layers.ImageEncoder
- class custom_layers.MultimodalDecoder
- class custom_layers.AttentionLayer

Namespaces

- custom_layers

7.19 src/models/generator_framework.py File Reference

Classes

- class generator_framework.Generator

Namespaces

- generator_framework

Functions

- def generator_framework.loss_switcher (loss_string)
- def generator_framework.optimizer_switcher (optimizer_string)

Variables

- generator_framework.ROOT_PATH = Path(__file__).absolute().parents[2]

7.20 src/models/predict_model.py File Reference

Namespaces

- predict_model

Variables

- predict_model.ROOT_PATH = Path(__file__).absolute().parents[2]
- predict_model.parser = argparse.ArgumentParser()
- predict_model.type
- predict_model.bool
- predict_model.default
- predict_model.help
- predict_model.str
- predict_model.required
- predict_model.int
- predict_model.args = vars(parser.parse_args())
- predict_model.interim_path = ROOT_PATH.joinpath('data', 'interim')
- predict_model.processed_path = ROOT_PATH.joinpath('data', 'processed')
- predict_model.ann_path = processed_path.joinpath('annotations')
- predict_model.models_path = ROOT_PATH.joinpath('models')
- predict_model.model_dir = models_path.joinpath(args['model'])

- predict_model.split_ = args['split']
- predict_model.dataset_ = args['dataset']
- predict_model.train_path = interim_path.joinpath(dataset_, dataset_ + '_train_clean.csv')
- predict_model.test_path
- predict_model.voc_path_ = interim_path.joinpath(dataset_, dataset_ + '_vocabulary.csv')
- predict_model.feature_path_
- predict_model.saved_model_path_ = model_dir.joinpath('BEST_checkpoint.pth.tar')
- predict_model.model_name_ = args['model_name']
- predict_model.save_path_ = models_path
- int predict_model.em_dim = 300
- int predict_model.hidden_shape_ = 50
- string predict_model.loss_function_ = 'cross_entropy'
- string predict_model.opt = 'adam'
- float predict_model.lr_ = 0.0001
- int predict_model.seed_ = 222
- predict_model.max_length = max_length_caption(train_path)
- list predict_model.input_shape_ = [[8, 8, 1536], max_length]
- predict_model.generator
- predict_model.beam_size_ = args['beam_size']
- predict_model.val_batch_size = args['val_batch_size']
- predict_model.res_file = model_dir.joinpath('TEST_' + split_ + '_result.json')
- predict_model.eval_file = model_dir.joinpath('TEST_' + split_ + '_eval.json')
- predict_model.annFile = ann_path.joinpath(dataset_ + '_' + split_ + '.json')
- predict_model.result

7.21 src/models/torch_generators.py File Reference

Classes

- class torch_generators.AdaptiveModel
- class torch_generators.AdaptiveDecoder

Namespaces

- torch_generators

Functions

- def torch_generators.model_switcher (model_str)

7.22 src/models/train_model.py File Reference

Namespaces

- train_model

Variables

- train_model.ROOT_PATH = Path(__file__).absolute().parents[2]
- train_model.parser = argparse.ArgumentParser()
- train_model.type
- train_model.int
- train_model.default
- train_model.help
- train_model.str
- train_model.float
- train_model.bool
- train_model.nargs
- train_model.required
- train_model.args = vars(parser.parse_args())
- train_model.interim_path
- train_model.processed_path
- train_model.dataset = args['dataset']
- train_model.annFile
- train_model.train_path = interim_path.joinpath(dataset + '_train_clean.csv')
- train_model.val_path = interim_path.joinpath(dataset + '_val.csv')
- train_model.voc_path_ = interim_path.joinpath(dataset + '_vocabulary.csv')
- train_model.feature_path_
- train_model.save_path_ = ROOT_PATH.joinpath('models')
- train_model.model_name_ = args['model']
- train_model.batch_size = args['batch_size']
- train_model.val_batch_size = args['val_batch_size']
- train_model.beam_size = args['beam_size']
- train_model.epochs = args['epochs']
- train_model.em_dim = args['embedding_size']
- train_model.hidden_size_ = args['hidden_size']
- train_model.loss_function_ = args['loss_function']
- train_model.opt = args['optimizer']
- train_model.lr_ = args['lr']
- train_model.seed_ = args['seed']
- train_model.max_length = max_length_caption(train_path)
- train_model.image_feature_size = args['image_feature_size']
- list train_model.input_shape_ = [image_feature_size, max_length]
- train_model.generator

7.23 src/visualization/visualize.py File Reference

Namespaces

- visualize