

981 lines (890 sloc)

42.8 KB

...

```
1  """Utilities for real-time data augmen
2  """
3  import warnings
4
5  import numpy as np
6
7  from .affine_transformations import (a
8                                     a
9                                     a
10 from .dataframe_iterator import DataFr
11 from .directory_iterator import Direct
12 from .numpy_array_iterator import Nump
13
14
15 class ImageDataGenerator(object):
16     """Generate batches of tensor imag
17     The data will be looped over (in
18
19     # Arguments
20         featurewise_center: Boolean.
21             Set input mean to 0 over t
22         samplewise_center: Boolean. Se
23         featurewise_std_normalization:
24             Divide inputs by std of th
25         samplewise_std_normalization:
26         zca_whitening: Boolean. Apply
27         zca_epsilon: epsilon for ZCA w
28         rotation_range: Int. Degree ra
```

```

25     samplewise_std_normalization:
26     zca_whitening: Boolean. Apply
27     zca_epsilon: epsilon for ZCA w
28     rotation_range: Int. Degree ra
29     width_shift_range: Float, 1-D
30         - float: fraction of total
31         - 1-D array-like: random e
32         - int: integer number of p
33           `(-width_shift_range,
34         - With `width_shift_range=
35           are integers `[-1, 0,
36           same as with `width_sh
37           while with `width_shif
38           in the interval `[-1.0
39     height_shift_range: Float, 1-D
40         - float: fraction of total
41         - 1-D array-like: random e
42         - int: integer number of p
43           `(-height_shift_range,
44         - With `height_shift_range
45           are integers `[-1, 0,
46           same as with `height_s
47           while with `height_shi
48           in the interval `[-1.0
49     brightness_range: Tuple or lis
50         a brightness shift value f
51     shear_range: Float. Shear Inte
52         (Shear angle in counter-cl
53     zoom_range: Float or [lower, u
54         If a float, `[lower, upper
55     channel_shift_range: Float. Ra
56     fill_mode: One of {"constant",
57         Default is 'nearest'.
58         Points outside the boundar

```

```

86         If you never set it, then
87         validation_split: Float. Fract
88         (strictly between 0 and 1)
89         interpolation_order: int, orde
90         the spline interpolation.
91         dtype: Dtype to use for the ge
92
93     # Examples
94     Example of using `.flow(x, y)`:
95
96     ```python
97     (x_train, y_train), (x_test, y_tes
98     y_train = np_utils.to_categorical(
99     y_test = np_utils.to_categorical(y
100
101     datagen = ImageDataGenerator(
102         featurewise_center=True,
103         featurewise_std_normalization=
104         rotation_range=20,
105         width_shift_range=0.2,
106         height_shift_range=0.2,
107         horizontal_flip=True)
108
109     # compute quantities required for
110     # (std, mean, and principal compor
111     datagen.fit(x_train)
112
113     # fits the model on batches with r
114     model.fit_generator(datagen.flow(>
115         steps_per_epoc
116
117     # here's a more "manual" example
118     for e in range(epochs):
119         print('Epoch', e)

```

```

54         If a float, [lower, upper]
55         channel_shift_range: Float. Random
56         fill_mode: One of {"constant",
57             Default is 'nearest'.
58             Points outside the boundaries
59             according to the given mode
60             - 'constant': kkkkkkkk|abc
61             - 'nearest':  aaaaaaaa|abc
62             - 'reflect':  abcd dcba|abc
63             - 'wrap':    abcdabcd|abcd|a
64         cval: Float or Int.
65             Value used for points outside
66             when `fill_mode = "constant"
67         horizontal_flip: Boolean. Randomly
68         vertical_flip: Boolean. Randomly
69         rescale: rescaling factor. Defaults
70             If None or 0, no rescaling
71             otherwise we multiply the
72             (after applying all other
73         preprocessing_function: function
74             The function will run after
75             The function should take as
76             one image (NumPy tensor) and
77             and should output a NumPy
78         data_format: Image data format
79             either "channels_first" or
80             "channels_last" mode means
81             `(samples, height, width,
82             "channels_first" mode means
83             `(samples, channels, height,
84             It defaults to the `image_data
85             Keras config file at `~/keras
86             If you never set it, then
87         validation_split: Float. Fraction

```



```

114     model.fit_generator(datagen.flow(x
115                             steps_per_epoc
116
117     # here's a more "manual" example
118     for e in range(epochs):
119         print('Epoch', e)
120         batches = 0
121         for x_batch, y_batch in datage
122             model.fit(x_batch, y_batch
123             batches += 1
124             if batches >= len(x_train)
125                 # we need to break the
126                 # the generator loops
127                 break
128     ...
129     Example of using `.flow_from_directories`
130
131     ```python
132     train_datagen = ImageDataGenerator(
133         rescale=1./255,
134         shear_range=0.2,
135         zoom_range=0.2,
136         horizontal_flip=True)
137
138     test_datagen = ImageDataGenerator(
139
140     train_generator = train_datagen.fl
141         'data/train',
142         target_size=(150, 150),
143         batch_size=32,
144         class_mode='binary')
145
146     validation_generator = test_datage
147         'data/validation',

```

```

175         image_datagen.fit(images, augment=
176         mask_datagen.fit(masks, augment=Tr
177
178         image_generator = image_datagen.fl
179             'data/images',
180             class_mode=None,
181             seed=seed)
182
183         mask_generator = mask_datagen.flow
184             'data/masks',
185             class_mode=None,
186             seed=seed)
187
188         # combine generators into one whic
189         train_generator = zip(image_genera
190
191         model.fit_generator(
192             train_generator,
193             steps_per_epoch=2000,
194             epochs=50)
195         ```
196
197         Example of using ``.flow_from_dat
198
199
200         ```python
201
202         train_df = pandas.read_csv("./trai
203         valid_df = pandas.read_csv("./vali
204
205         train_datagen = ImageDataGenerator
206             rescale=1./255,
207             shear_range=0.2,
208             zoom_range=0.2,

```

```

240         def __init__(self,
241                         featurewise_center=False,
242                         samplewise_center=False,
243                         featurewise_std_normalization=True,
244                         samplewise_std_normalization=True,
245                         zca_whitening=False,
246                         zca_epsilon=1e-6,
247                         rotation_range=0,
248                         width_shift_range=0.,
249                         height_shift_range=0.,
250                         brightness_range=None,
251                         shear_range=0.,
252                         zoom_range=0.,
253                         channel_shift_range=0,
254                         fill_mode='nearest',
255                         cval=0.,
256                         horizontal_flip=False,
257                         vertical_flip=False,
258                         rescale=None,
259                         preprocessing_function=None,
260                         data_format='channels',
261                         validation_split=0.0,
262                         interpolation_order=1,
263                         dtype='float32'):
264
265         self.featurewise_center = featurewise_center
266         self.samplewise_center = samplewise_center
267         self.featurewise_std_normalization = featurewise_std_normalization
268         self.samplewise_std_normalization = samplewise_std_normalization
269         self.zca_whitening = zca_whitening
270         self.zca_epsilon = zca_epsilon
271         self.rotation_range = rotation_range
272         self.width_shift_range = width_shift_range
273         self.height_shift_range = height_shift_range

```

```
210
211     test_datagen = ImageDataGenerator(
212
213     train_generator = train_datagen.flow(
214         dataframe=train_df,
215         directory='data/train',
216         x_col="filename",
217         y_col="class",
218         target_size=(150, 150),
219         batch_size=32,
220         class_mode='binary')
221
222     validation_generator = test_datagen(
223         dataframe=valid_df,
224         directory='data/validation',
225         x_col="filename",
226         y_col="class",
227         target_size=(150, 150),
228         batch_size=32,
229         class_mode='binary')
230
231     model.fit_generator(
232         train_generator,
233         steps_per_epoch=2000,
234         epochs=50,
235         validation_data=validation_generator,
236         validation_steps=800)
237     ...
238     """
239
240     def __init__(self,
241                 featurewise_center=False,
242                 samplewise_center=False,
243                 featurewise_std_normalization=False,
```



```
299         self.row_axis = 1
300         self.col_axis = 2
301         if validation_split and not 0:
302             raise ValueError(
303                 '`validation_split` must be
304                 ' Received: %s' % validation_split)
305         self._validation_split = validation_split
306
307         self.mean = None
308         self.std = None
309         self.zca_whitening_matrix = None
310
311         if isinstance(zoom_range, (float, int)):
312             self.zoom_range = [1 - zoom_range, 1 + zoom_range]
313         elif (len(zoom_range) == 2 and
314               all(isinstance(val, (float, int)) for val in zoom_range)):
315             self.zoom_range = zoom_range
316         else:
317             raise ValueError('`zoom_range` should be a tuple
318                               ' of two floats. Received %s' % zoom_range)
319
320         if zca_whitening:
321             if not featurewise_center:
322                 self.featurewise_center = featurewise_center
323                 warnings.warn('This ImageDataGenerator does not
324                               ' support zca whitening when
325                               ' setting featurewise_center to
326                               ' False')
327             if featurewise_std_normalization:
328                 self.featurewise_std_normalization = featurewise_std_normalization
329                 warnings.warn('This ImageDataGenerator does not
330                               ' support zca whitening when
331                               ' setting featurewise_std_normalization
332                               ' to True')
333             if featurewise_std_normalization and not featurewise_center:
```

```
328         warnings.warn('This Im
329             ``zca_wh
330             'which o
331             ``featur
332     if featurewise_std_normalizati
333         if not featurewise_center:
334             self.featurewise_cente
335             warnings.warn('This Im
336             ``featur
337             'which o
338             ``featur
339     if samplewise_std_normalizatio
340         if not samplewise_center:
341             self.samplewise_center
342             warnings.warn('This Im
343             ``sample
344             'which o
345             ``sample
346     if brightness_range is not Non
347         if (not isinstance(brightn
348             len(brightness_ran
349             raise ValueError(
350                 ``brightness_range
351                 'Received: %s' % (
352     self.brightness_range = bright
353
354     def flow(self,
355         x,
356         y=None,
357         batch_size=32,
358         shuffle=True,
359         sample_weight=None,
360         seed=None,
361         save_to_dir=None,
```

```
391         Prefix to use for file
392         (only relevant if `save
393         save_format: one of "png",
394         (only relevant if `save
395         ignore_class_split: Boolean
396         in number of classes i
397         split (useful for non-
398         subset: Subset of data (`
399         `validation_split` is
400
401     # Returns
402     An `Iterator` yielding tup
403     where `x` is a NumPy a
404     (in the case of a sing
405     of NumPy arrays (in th
406     additional inputs) and
407     of corresponding label
408     the yielded tuples are
409     If `y` is None, only t
410     """
411     return NumpyArrayIterator(
412         x,
413         y,
414         self,
415         batch_size=batch_size,
416         shuffle=shuffle,
417         sample_weight=sample_weigh
418         seed=seed,
419         data_format=self.data_form
420         save_to_dir=save_to_dir,
421         save_prefix=save_prefix,
422         save_format=save_format,
423         ignore_class_split=ignore_
424         subset=subset,
```

```

422         save_format=save_format,
423         ignore_class_split=ignore_
424         subset=subset,
425         dtype=self.dtype
426     )
427
428     def flow_from_directory(self,
429                             directory,
430                             target_size
431                             color_mode
432                             classes=None
433                             class_mode
434                             batch_size
435                             shuffle=True
436                             seed=None,
437                             save_to_disk
438                             save_prefix
439                             save_format
440                             follow_links
441                             subset=None
442                             interpolation
443                             keep_aspect_ratio
444         """Takes the path to a directory
445
446         # Arguments
447         directory: string, path to
448             It should contain one
449             Any PNG, JPG, BMP, PPM
450             inside each of the sub
451             will be included in the
452             See [this script](
453             https://gist.github.com
454             for more details.
455         target_size: Tuple of integers

```


453 <https://gist.github.com>
454 for more details.
455 target_size: Tuple of inte
456 default: `(256, 256)`.
457 The dimensions to whic
458 color_mode: One of "grayscale"
459 Whether the images will
460 have 1, 3, or 4 channels
461 classes: Optional list of
462 (e.g. `['dogs', 'cats']`
463 If not provided, the labels
464 inferred from the subdirectories
465 under `directory`, where
466 be treated as a different
467 (and the order of the
468 indices, will be alphabetical)
469 The dictionary containing
470 indices can be obtained
471 class_mode: One of "categorical",
472 "input", or None. Default
473 Determines the type of
474 - "categorical" will be used
475 - "binary" will be 1D
476 "sparse" will be 1D
477 - "input" will be images
478 to input images (must be
479 - If None, no labels are
480 (the generator will
481 which is useful to use
482 Please note that in
483 the data still needs
484 of `directory` for images
485 batch_size: Size of the batch
486 shuffle: Whether to shuffle

```

543         dataframe,
544         directory=
545         x_col="fil
546         y_col="cla
547         weight_col
548         target_siz
549         color_mode
550         classes=No
551         class_mode
552         batch_size
553         shuffle=Tr
554         seed=None,
555         save_to_di
556         save_prefi
557         save_forma
558         subset=Non
559         interpolat
560         validate_f
561         **kwargs):
562         """Takes the dataframe and the
563         and generates batches of augm
564
565         **A simple tutorial can be fou
566         ht
567
568         # Arguments
569         dataframe: Pandas dataframe
570         `directory` (or absolu
571         images in a string col
572         depending on the `clas
573         - if `class_mode` is `
574         include the `y_col
575         Values in column c
576         or list/tuple if m

```

```

513         the center with target
514
515     # Returns
516         A `DirectoryIterator` yield
517         where `x` is a NumPy array
518         of images with shape `x`
519         and `y` is a NumPy array
520         of labels
521     """
522     return DirectoryIterator(
523         directory,
524         self,
525         target_size=target_size,
526         keep_aspect_ratio=keep_aspect_ratio,
527         color_mode=color_mode,
528         classes=classes,
529         class_mode=class_mode,
530         data_format=self.data_format,
531         batch_size=batch_size,
532         shuffle=shuffle,
533         seed=seed,
534         save_to_dir=save_to_dir,
535         save_prefix=save_prefix,
536         save_format=save_format,
537         follow_links=follow_links,
538         subset=subset,
539         interpolation=interpolation,
540         dtype=self.dtype
541     )
542
543     def flow_from_dataframe(self,
544                           dataframe,
545                           directory=".",
546                           x_col="file_name",
547                           y_col="class_name",
548                           class_mode="classes",
549                           data_format="channels_last",
550                           target_size=(256, 256),
551                           batch_size=32,
552                           shuffle=True,
553                           seed=None,
554                           save_to_dir=None,
555                           save_prefix="flow_from_dataframe_",
556                           save_format="png",
557                           follow_links=False,
558                           subset=False,
559                           interpolation="nearest",
560                           dtype=None):

```

```
575         Values in column c
576         or list/tuple if n
577         - if `class_mode` is `
578         the given `y_col`
579         - if `class_mode` is `
580         the columns specified
581         - if `class_mode` is `
582         directory: string, path to
583         data in `x_col` column
584         x_col: string, column in `
585         absolute paths if `dir
586         y_col: string or list, col
587         weight_col: string, column
588         weights. Default: `Non
589         target_size: tuple of inte
590         The dimensions to whic
591         color_mode: one of "grayscale"
592         Whether the images wil
593         classes: optional list of
594         Default: None. If not
595         automatically inferred
596         which will map to the
597         The dictionary contain
598         indices can be obtaine
599         class_mode: one of "binary
600         "raw", sparse" or None
601         Mode for yielding the
602         - `"binary"`: 1D NumPy
603         - `"categorical"`: 2D
604         Supports multi-label
605         - `"input"`: images id
606         work with autoenco
607         - `"multi_output"`: li
608         - `"raw"`: NumPy array
```



```
645         if 'has_ext' in kwargs:
646             warnings.warn('has_ext is
647                             'to match th
648                             DeprecationW
649         if 'sort' in kwargs:
650             warnings.warn('sort is dep
651                             'same order
652                             'is set to F
653         if class_mode == 'other':
654             warnings.warn('`class_mode
655                             `class_mode
656                             class_mode = 'raw'
657         if 'drop_duplicates' in kwargs
658             warnings.warn('drop_duplic
659                             'by using th
660                             DeprecationW
661
662         return DataFrameIterator(
663             dataframe,
664             directory,
665             self,
666             x_col=x_col,
667             y_col=y_col,
668             weight_col=weight_col,
669             target_size=target_size,
670             color_mode=color_mode,
671             classes=classes,
672             class_mode=class_mode,
673             data_format=self.data_form
674             batch_size=batch_size,
675             shuffle=shuffle,
676             seed=seed,
677             save_to_dir=save_to_dir,
678             save_prefix=save_prefix,
```

```

715         else:
716             warnings.warn('This Im
717                             ``featur
718                             'been fi
719                             'first b
720         if self.featurewise_std_normal
721             if self.std is not None:
722                 x /= (self.std + 1e-6)
723             else:
724                 warnings.warn('This Im
725                             ``featur
726                             'but it
727                             'been fi
728                             'first b
729         if self.zca_whitening:
730             if self.zca_whitening_matr
731                 flat_x = x.reshape(-1,
732                 white_x = flat_x @ sel
733                 x = np.reshape(white_x
734             else:
735                 warnings.warn('This Im
736                             ``zca_wh
737                             'been fi
738                             'first b
739         return x
740
741     def get_random_transform(self, img
742         """Generates random parameters
743
744         # Arguments
745             seed: Random seed.
746             img_shape: Tuple of intege
747                     Shape of the image tha
748

```

```
796
797     if self.zoom_range[0] == 1 and
798         zx, zy = 1, 1
799     else:
800         zx, zy = np.random.uniform
801             self.zoom_range[0],
802             self.zoom_range[1],
803             2)
804
805     flip_horizontal = (np.random.r
806     flip_vertical = (np.random.ran
807
808     channel_shift_intensity = None
809     if self.channel_shift_range !=
810         channel_shift_intensity =
811
812
813     brightness = None
814     if self.brightness_range is no
815         brightness = np.random.uni
816
817
818     transform_parameters = {'theta
819                             'tx':
820                             'ty':
821                             'shear
822                             'zx':
823                             'zy':
824                             'flip_
825                             'flip_
826                             'chann
827                             'brigh
828
829     return transform_parameters
```

```

832         Applies a transformation to
833
834     # Arguments
835         x: 3D tensor, single image
836         transform_parameters: Dict
837             describing the transfo
838             Currently, the followi
839             from the dictionary ar
840         - ``theta``: Float. Ro
841         - ``tx``: Float. Shift
842         - ``ty``: Float. Shift
843         - ``shear``: Float. Sh
844         - ``zx``: Float. Zoom
845         - ``zy``: Float. Zoom
846         - ``flip_horizontal``:
847         - ``flip_vertical``: B
848         - ``channel_shift_inte
849         - ``brightness``: Floa
850
851     # Returns
852         A transformed version of t
853     """
854     # x is a single image, so it d
855     img_row_axis = self.row_axis -
856     img_col_axis = self.col_axis -
857     img_channel_axis = self.channe
858
859     x = apply_affine_transform(x,
860                                tra
861                                tra
862                                tra
863                                tra
864                                tra
865                                row
866                                col

```



```

950         x *= self.rescale
951
952     if augment:
953         ax = np.zeros(
954             tuple([rounds * x.shape
955                   dtype=self.dtype])
956         for r in range(rounds):
957             for i in range(x.shape
958                 ax[i + r * x.shape
959         x = ax
960
961     if self.featurewise_center:
962         self.mean = np.mean(x, axis=
963         broadcast_shape = [1, 1, 1
964         broadcast_shape[self.chann
965         self.mean = np.reshape(sel
966         x -= self.mean
967
968     if self.featurewise_std_normal
969         self.std = np.std(x, axis=
970         broadcast_shape = [1, 1, 1
971         broadcast_shape[self.chann
972         self.std = np.reshape(self
973         x /= (self.std + 1e-6)
974
975     if self.zca_whitening:
976         n = len(x)
977         flat_x = np.reshape(x, (n,
978
979         u, s, _ = np.linalg.svd(fl
980         s_inv = np.sqrt(n) / (s +
981         self.zca_whitening_matrix

```

```

145
146     validation_generator = test_datagen
147         'data/validation',
148         target_size=(150, 150),
149         batch_size=32,
150         class_mode='binary')
151
152     model.fit_generator(
153         train_generator,
154         steps_per_epoch=2000,
155         epochs=50,
156         validation_data=validation
157         validation_steps=800)
158     ...
159

```

160 Example of transforming images and

```

161
162     ```python
163     # we create two instances with the
164     data_gen_args = dict(featurewise_c
165                           featurewise_s
166                           rotation_rang
167                           width_shift_r
168                           height_shift_
169                           zoom_range=0.
170     image_datagen = ImageDataGenerator
171     mask_datagen = ImageDataGenerator(
172
173     # Provide the same seed and keywor
174     seed = 1
175     image_datagen.fit(images, augment=
176     mask_datagen.fit(masks, augment=Tr
177
178     image_generator = image_datagen.fl

```

```
269         self.zca_whitening = zca_white
270         self.zca_epsilon = zca_epsilon
271         self.rotation_range = rotation
272         self.width_shift_range = width
273         self.height_shift_range = height
274         self.shear_range = shear_range
275         self.zoom_range = zoom_range
276         self.channel_shift_range = channel_shift_range
277         self.fill_mode = fill_mode
278         self.cval = cval
279         self.horizontal_flip = horizontal_flip
280         self.vertical_flip = vertical_flip
281         self.rescale = rescale
282         self.preprocessing_function = preprocessing_function
283         self.dtype = dtype
284         self.interpolation_order = interpolation_order
285
286         if data_format not in {'channels_first', 'channels_last'}:
287             raise ValueError(
288                 '`data_format` should be either "channels_first" or "channels_last". '
289                 'Received: %s' % data_format)
290         self.data_format = data_format
291         if data_format == 'channels_first':
292             self.channel_axis = 1
293             self.row_axis = 2
294             self.col_axis = 3
295         if data_format == 'channels_last':
296             self.channel_axis = 3
297             self.row_axis = 1
298             self.col_axis = 2
```

```

359         sample_weight=None,
360         seed=None,
361         save_to_dir=None,
362         save_prefix='',
363         save_format='png',
364         ignore_class_split=False,
365         subset=None):
366     """Takes data & label arrays,
367
368     # Arguments
369     x: Input data. NumPy array
370         If tuple, the first el
371         should contain the ima
372         another NumPy array or
373         that gets passed to th
374         without any modificati
375         Can be used to feed th
376         along with the images.
377         In case of grayscale d
378         should have value 1, i
379         of RGB data, it should
380         of RGBA data, it shoul
381     y: Labels.
382     batch_size: Int (default:
383     shuffle: Boolean (default:
384     sample_weight: Sample weig
385     seed: Int (default: None).
386     save_to_dir: None or str (
387         This allows you to opt
388         to which to save the a
389         (useful for visualizin
390     save_prefix: Str (default:
391         Prefix to use for file
392         (only relevant if `sav

```



```

482         Please note that in
483         the data still needs
484         of `directory` for i
485         batch_size: Size of the ba
486         shuffle: Whether to shuffl
487         If set to False, sorts
488         seed: Optional random seed
489         save_to_dir: None or str (
490         This allows you to opt
491         a directory to which t
492         the augmented pictures
493         (useful for visualizin
494         save_prefix: Str. Prefix t
495         (only relevant if `sav
496         save_format: One of "png",
497         (only relevant if `sav
498         follow_links: Whether to f
499         class subdirectories (
500         subset: Subset of data (`"
501         `validation_split` is
502         interpolation: Interpolati
503         resample the image if
504         target size is differe
505         Supported methods are
506         and `"bicubic"`.
507         If PIL version 1.1.3 o
508         supported. If PIL vers
509         `"box"` and `"hamming"
510         By default, `"nearest"
511         keep_aspect_ratio: Boolean
512         size without aspect ra
513         the center with target
514
515         # Returns
516         A `DirectoryIterator` w

```

```

609         - "sparse": 1D NumPy
610         - `None`, no targets a
611         batches of image d
612         `model.predict_gen
613     batch_size: size of the ba
614     shuffle: whether to shuffl
615     seed: optional random seed
616     save_to_dir: None or str (
617         This allows you to opt
618         to which to save the a
619         (useful for visualizin
620     save_prefix: str. Prefix t
621         (only relevant if `sav
622     save_format: one of "png",
623         (only relevant if `sav
624     follow_links: whether to f
625         (default: False).
626     subset: Subset of data (`"
627         `validation_split` is
628     interpolation: Interpolati
629         target size is differe
630         Supported methods are
631         If PIL version 1.1.3 o
632         supported. If PIL vers
633         `"hamming"` are also s
634     validate_filenames: Boolea
635         `x_col`. If `True`, in
636         option can lead to spe
637         Default: `True`.
638
639     # Returns
640     A `DataFrameIterator` yiel
641     where `x` is a NumPy array
642     of images with shape `(bat

```

```

676         seed=seed,
677         save_to_dir=save_to_dir,
678         save_prefix=save_prefix,
679         save_format=save_format,
680         subset=subset,
681         interpolation=interpolatio
682         validate_filenames=validat
683         dtype=self.dtype
684     )
685
686     def standardize(self, x):
687         """Applies the normalization c
688
689         `x` is changed in-place since
690         to standardize images and feed
691         would be created instead it wo
692         If you want to apply this meth
693         you can call the method creati
694
695         standardize(np.copy(x))
696
697         # Arguments
698             x: Batch of inputs to be n
699
700         # Returns
701             The inputs, normalized.
702         """
703         if self.preprocessing_function:
704             x = self.preprocessing_fun
705         if self.rescale:
706             x *= self.rescale
707         if self.samplewise_center:
708             x -= np.mean(x, keepdims=T
709         if self.samplewise_std_normali

```

```
754         img_col_axis = self.col_axis -
755
756         if seed is not None:
757             np.random.seed(seed)
758
759         if self.rotation_range:
760             theta = np.random.uniform(
761                 -self.rotation_range,
762                 self.rotation_range)
763         else:
764             theta = 0
765
766         if self.height_shift_range:
767             try: # 1-D array-like or
768                 tx = np.random.choice(
769                     tx *= np.random.choice
770             except ValueError: # floa
771                 tx = np.random.uniform
772
773                 if np.max(self.height_shif
774                     tx *= img_shape[img_ro
775         else:
776             tx = 0
777
778         if self.width_shift_range:
779             try: # 1-D array-like or
780                 ty = np.random.choice(
781                     ty *= np.random.choice
782             except ValueError: # floa
783                 ty = np.random.uniform
784
785                 if np.max(self.width_shift
786                     ty *= img_shape[img_co
787         else:
788             ty = 0
```



```

888     def random_transform(self, x, seed
889         """Applies a random transforma
890
891         # Arguments
892             x: 3D tensor, single image
893             seed: Random seed.
894
895         # Returns
896             A randomly transformed ver
897         """
898         params = self.get_random_trans
899         return self.apply_transform(x,
900
901     def fit(self, x,
902             augment=False,
903             rounds=1,
904             seed=None):
905         """Fits the data generator to
906
907         This computes the internal dat
908         data-dependent transformations
909
910         Only required if `featurewise_
911         `featurewise_std_normalization
912
913         When `rescale` is set to a val
914         sample data before computing t
915
916         # Arguments
917             x: Sample data. Should hav
918             In case of grayscale data
919             the channels axis should
920             of RGB data, it should ha
921             of RGBA data, it should h

```

```

920         of RGB data, it should ha
921         of RGBA data, it should h
922         augment: Boolean (default:
923         Whether to fit on rand
924         rounds: Int (default: 1).
925         If using data augmenta
926         this is how many augme
927         seed: Int (default: None).
928     """
929     x = np.asarray(x, dtype=self.d
930     if x.ndim != 4:
931         raise ValueError('Input to
932                             'Got arra
933     if x.shape[self.channel_axis]
934         warnings.warn(
935             'Expected input to be
936             'following the data fo
937     self.data_format + '
938     str(self.channel_axis)
939     'either 1, 3 or 4 chan
940     str(self.channel_axis)
941     'However, it was passe
942     str(x.shape) + ' (' +
943     ' channels).')
944
945     if seed is not None:
946         np.random.seed(seed)
947
948     x = np.copy(x)
949     if self.rescale:
950         x *= self.rescale

```