## Image Data Generator

We need a sample image to demonstrate standard data augmentation techniques.

In this tutorial, we will use a photograph of a bird titled “[Feathered Friend](https://www.flickr.com/photos/thenovys/3854468621/)” by AndYaDontStop, released under a permissive license.

Download the image and save it in your current working directory with the filename ‘bird.jpg ‘.



Feathered Friend, taken by AndYaDontStop.  
Some rights reserved.

The Keras deep learning library provides the ability to use data augmentation automatically when training a model.

This is achieved by using the [ImageDataGenerator class](https://keras.io/preprocessing/image/).

First, the class may be instantiated and the configuration for the types of data augmentation are specified by arguments to the class constructor.

A range of techniques are supported, as well as pixel scaling methods. We will focus on five main types of data augmentation techniques for image data; specifically:

* Image shifts via the *width\_shift\_range* and *height\_shift\_range* arguments.
* Image flips via the *horizontal\_flip* and *vertical\_flip* arguments.
* Image rotations via the *rotation\_range* argument
* Image brightness via the *brightness\_range* argument.
* Image zoom via the *zoom\_range* argument.

For example, an instance of the ImageDataGenerator class can be constructed.

|  |  |
| --- | --- |
| 1  2  3 | ...  # create data generator  datagen = ImageDataGenerator() |

Once constructed, an iterator can be created for an image dataset.

The iterator will return one batch of augmented images for each iteration.

An iterator can be created from an image dataset loaded in memory via the *flow ()* function; for example:

|  |  |
| --- | --- |
| 1  2  3  4  5 | ...  # load image dataset  X, y = ...  # create iterator  it = datagen.flow(X, y) |

Alternately, an iterator can be created for an image dataset located on disk in a specified directory, where images in that directory are organized into subdirectories according to their class.

|  |  |
| --- | --- |
| 1  2  3 | ...  # create iterator  it = datagen.flow\_from\_directory(X, y, ...) |

Once the iterator is created, it can be used to train a neural network model by calling the *fit\_generator ()* function.

The *steps\_per\_epoch* argument must specify the number of batches of samples comprising one epoch. For example, if your original dataset has 10,000 images and your batch size is 32, then a reasonable value for *steps\_per\_epoch* when fitting a model on the augmented data might be *ceil (10,000/32)*, or 313 batches.

|  |  |
| --- | --- |
| 1  2  3  4 | # define model  model = ...  # fit model on the augmented dataset  model.fit\_generator(it, steps\_per\_epoch=313, ...) |

The images in the dataset are not used directly. Instead, only augmented images are provided to the model. Because the augmentations are performed randomly, this allows both modified images and close facsimiles of the original images (e.g. almost no augmentation) to be generated and used during training.

A data generator can also be used to specify the validation dataset and the test dataset. Often, a separate *ImageDataGenerator* instance is used that may have the same pixel scaling configuration (not covered in this tutorial) as the *ImageDataGenerator* instance used for the training dataset, but would not use data augmentation. This is because data augmentation is only used as a technique for artificially extending the training dataset in order to improve model performance on an unaugmented dataset.

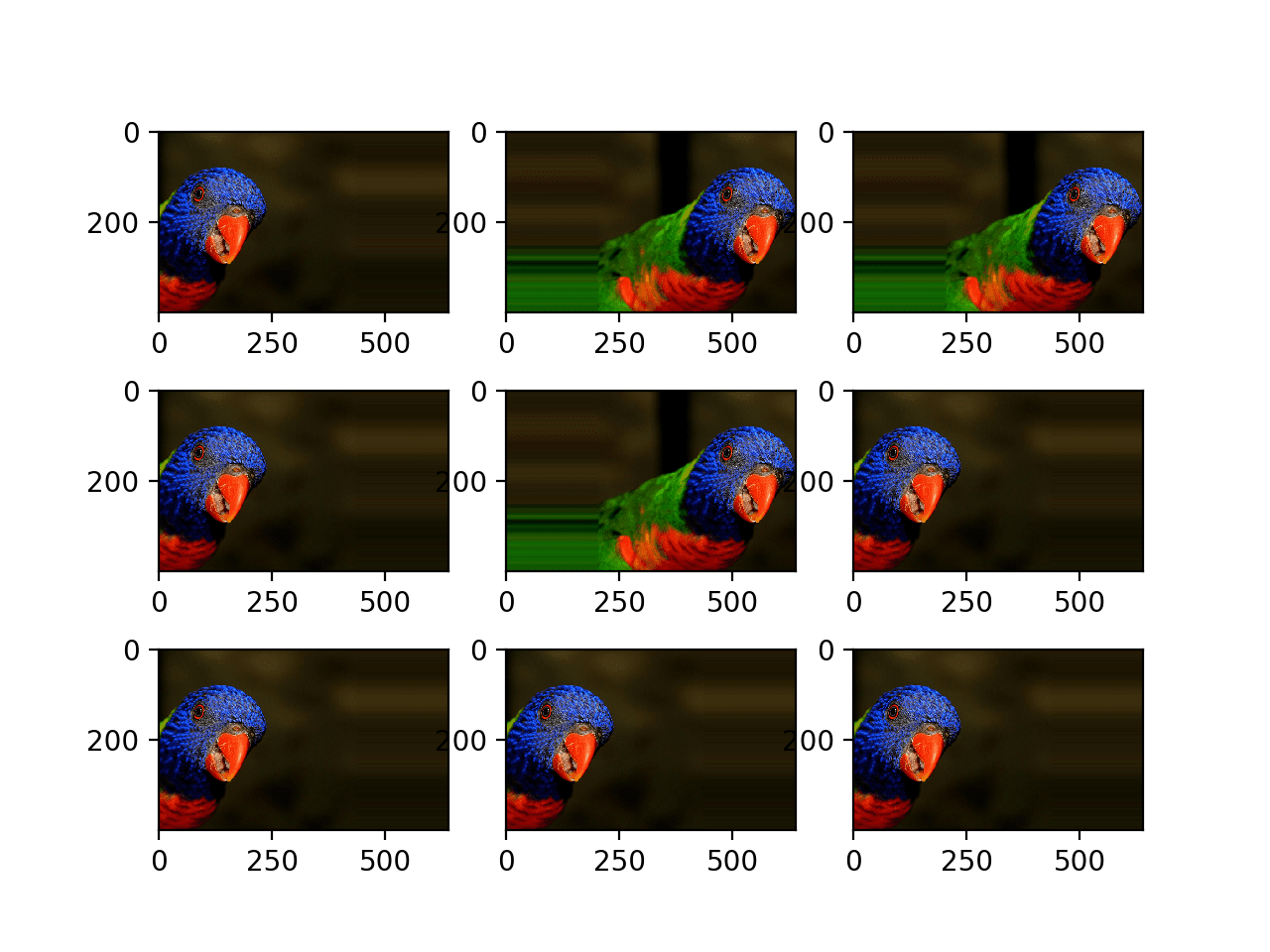
Now that we are familiar with how to use the *ImageDataGenerator*, let’s look at some specific data augmentation techniques for image data.

We will demonstrate each technique standalone by reviewing examples of images after they have been augmented. This is a good practice and is recommended when configuring your data augmentation. It is also common to use a range of augmentation techniques at the same time when training. We have isolated the techniques to one per section for demonstration purposes only.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | # example of horizontal shift image augmentation  from numpy import expand\_dims  from keras. preprocessing. image import load\_img  from keras. preprocessing. image import img\_to\_array  from keras. preprocessing. image import ImageDataGenerator  from matplotlib import pyplot  # load the image  img = load\_img('bird.jpg')  # convert to numpy array  data = img\_to\_array(img)  # expand dimension to one sample  samples = expand\_dims (data, 0)  # create image data augmentation generator  datagen = Image Data Generator (width\_shift\_range= [-200,200])  # prepare iterator  it = datagen. Flow (samples, batch\_size=1)  # generate samples and plot  for i in range (9):  # define subplot  Pyplot.subplot(330 + 1 + i)  # generate batch of images  batch = it. Next ()  # convert to unsigned integers for viewing  image = batch [0]. astype('uint8')  # plot raw pixel data  pyplot. imshow(image)  # show the figure  pyplot.show() |

Running the example creates the instance of *ImageDataGenerator* configured for image augmentation, then creates the iterator. The iterator is then called nine times in a loop and each augmented image is plotted.

We can see in the plot of the result that a range of different randomly selected positive and negative horizontal shifts was performed and the pixel values at the edge of the image are duplicated to fill in the empty part of the image created by the shift.



Plot of Augmented Generated with a Random Horizontal Shift

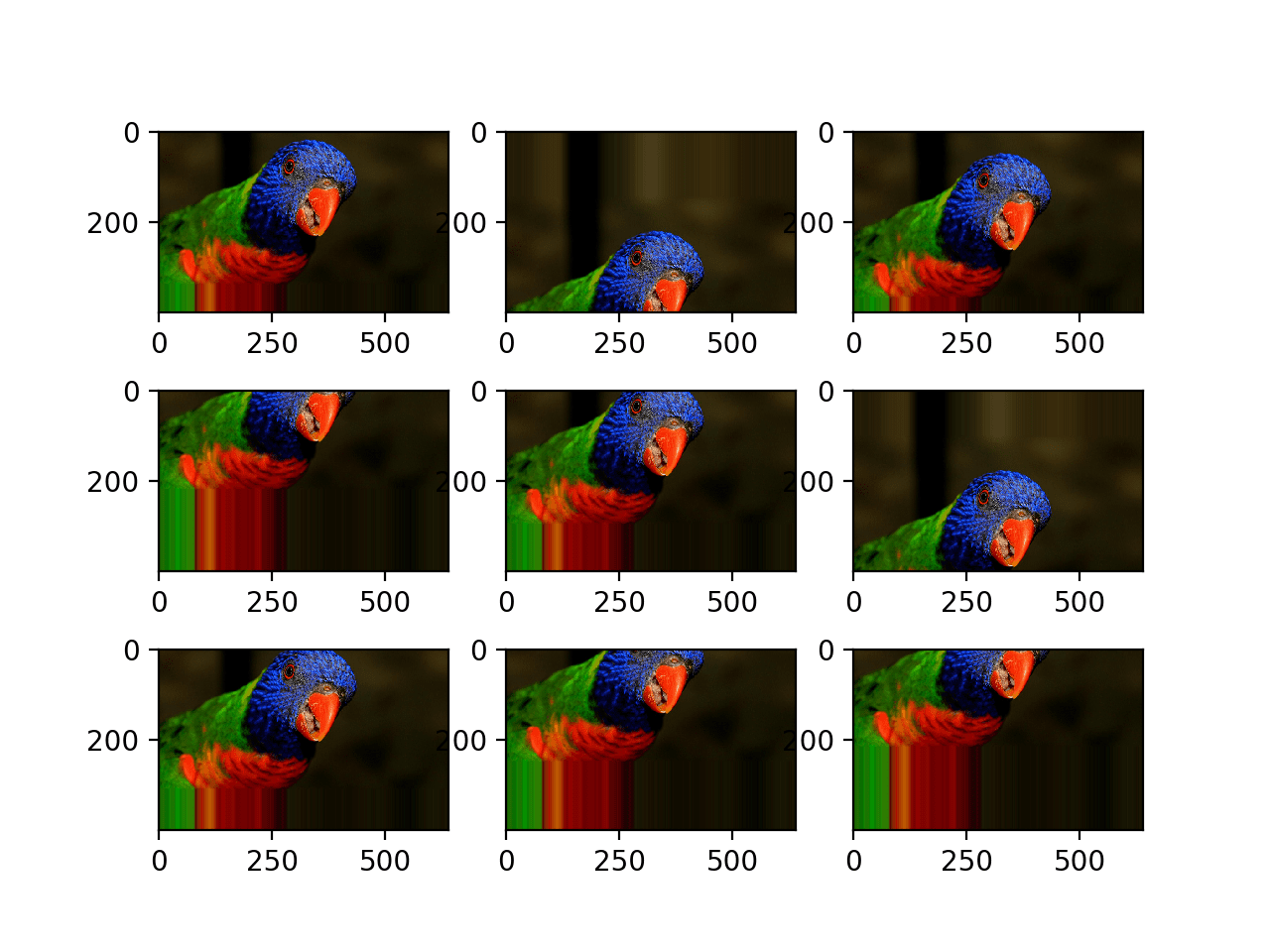
Below is the same example updated to perform vertical shifts of the image via the *height\_shift\_range* argument, in this case specifying the percentage of the image to shift as 0.5 the height of the image.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | # example of vertical shift image augmentation  from numpy import expand\_dims  from keraspreprocess ing. image import load\_img  from keras. preprocessing. image import img\_to\_array  from keras. preprocessing. image import ImageDataGenerator  from matplotlib import pyplot  # load the image  img = load\_img('bird.jpg')  # convert to numpy array  data = img\_to\_array(img)  # expand dimension to one sample  samples = expand\_dims (data, 0)  # create image data augmentation generator  datagen = ImageDataGenerator(height\_shift\_range=0.5)  # prepare iterator  it = datagen. Flow (samples, batch\_size=1)  # generate samples and plot  for i in range (9):  # define subplot  pyplot. Subplott (330 + 1 + i)  # generate batch of images  batch = it. Next ()  # convert to unsigned integers for viewing  image = batch [0]. astype ('uint8')  # plot raw pixel data  Pyplot. imshow(image)  # show the figure  pyplot.show() |

**Running the example creates a plot of images augmented with random positive and negative vertical shifts.**

**We can see that both horizontal and vertical positive and negative shifts probably make sense for the chosen photograph, but in some cases, the replicated pixels at the edge of the image may not make sense to a model.**

**Note that other fill modes can be specified via “*fill\_mode*” argument.**



Plot of Augmented Images with a Random Vertical Shift

**Horizontal and Vertical Flip Augmentation**

An image flip means reversing the rows or columns of pixels in the case of a vertical or horizontal flip respectively.

The flip augmentation is specified by a boolean *horizontal\_flip* or *vertical\_flip* argument to the *ImageDataGenerator* class constructor. For photographs like the bird photograph used in this tutorial, horizontal flips may make sense, but vertical flips would not.

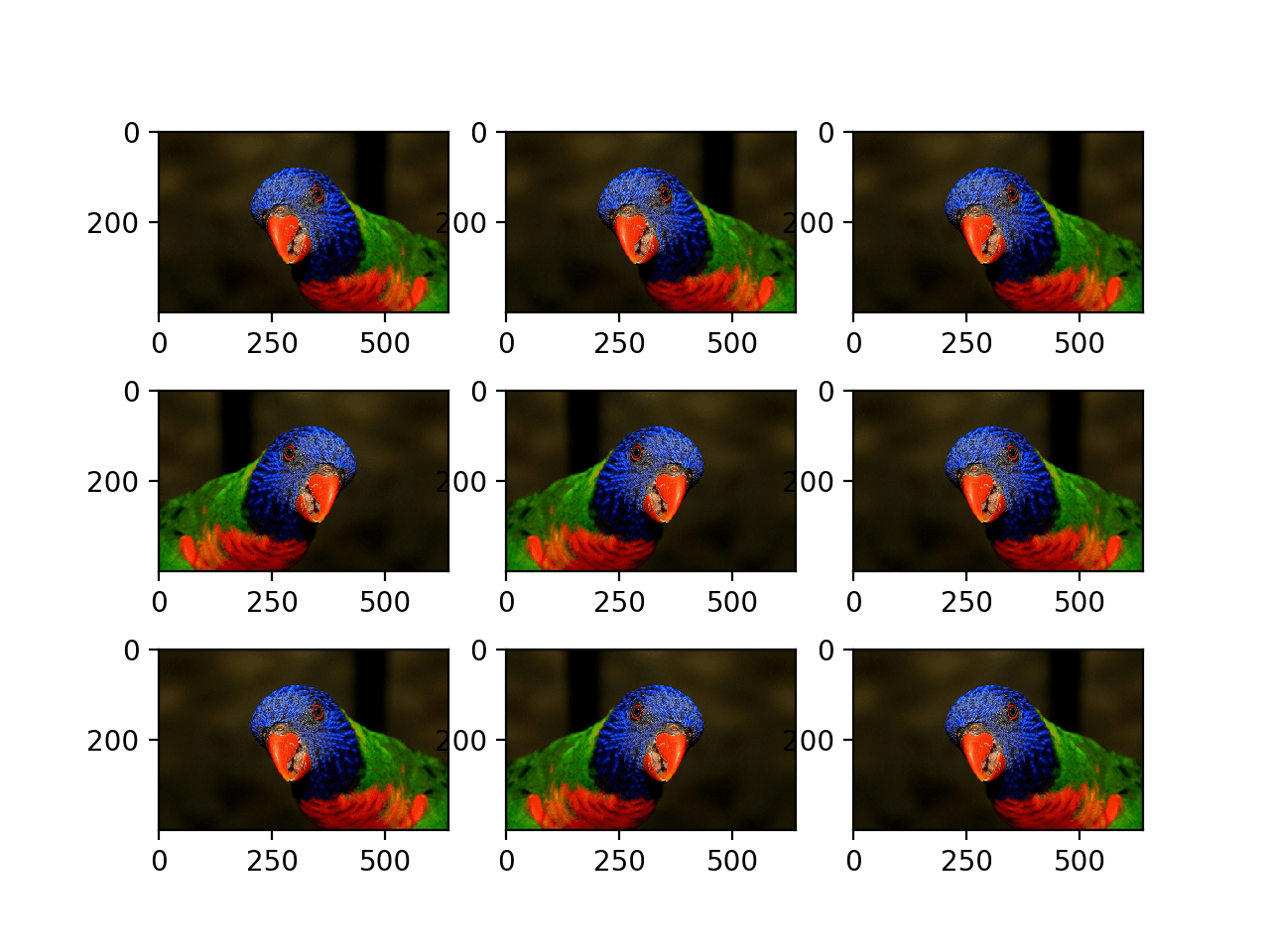
For other types of images, such as aerial photographs, cosmology photographs, and microscopic photographs, perhaps vertical flips make sense.

The example below demonstrates augmenting the chosen photograph with horizontal flips via the *horizontal\_flip* argument.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | # example of horizontal flip image augmentation  from numpy import expand\_dims  from keras. preprocessing. image import load\_img  from keras. preprocessing. image import img\_to\_array  from keras. preprocessing. image import ImageDataGenerator  from matplotlib import pyplot  # load the image  img = load\_img('bird.jpg')  # convert to numpy array  data = img\_to\_array(img)  # expand dimension to one sample  samples = expand\_dims (data, 0)  # create image data augmentation generator  datagen = ImageDataGenerator(horizontal\_flip=True)  # prepare iterator  it = datagen. Flow (samples, batch\_size=1)  # generate samples and plot  for i in range (9):  # define subplot  Pyplot. Subplot (330 + 1 + i)  # generate batch of images  batch = it. next ()  # convert to unsigned integers for viewing  image = batch [0]. Astype ('uint8')  # plot raw pixel data  pyplot. imshow (image)  # show the figure  pyplot.show() |

**Running the example creates a plot of nine augmented images.**

**We can see that the horizontal flip is applied randomly to some images and not others.**



Plot of Augmented Images with a Random Horizontal Flip

**Random Rotation Augmentation**

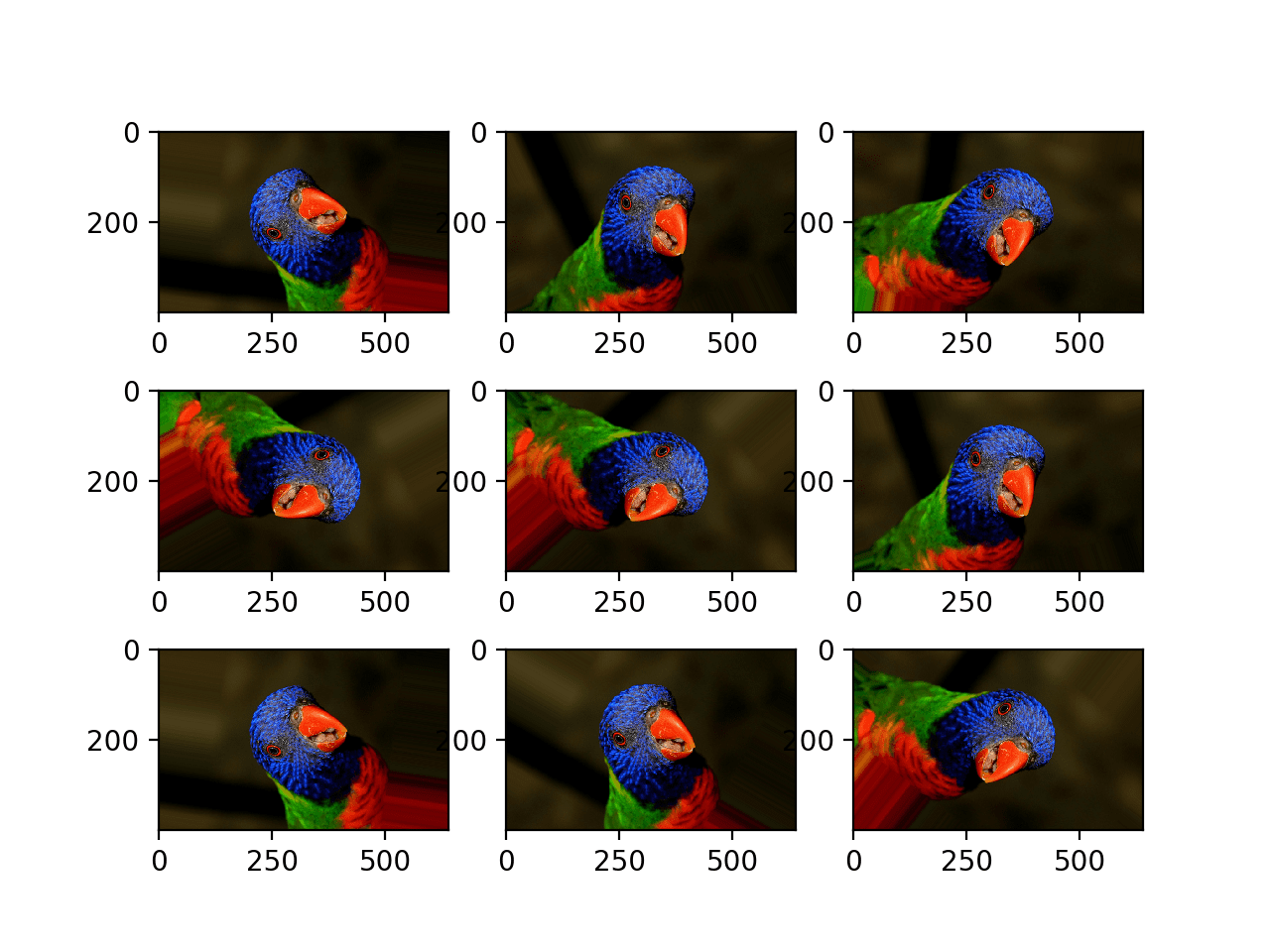
A rotation augmentation randomly rotates the image clockwise by a given number of degrees from 0 to 360.

The rotation will likely rotate pixels out of the image frame and leave areas of the frame with no pixel data that must be filled in.

The example below demonstrates random rotations via the *rotation\_range* argument, with rotations to the image between 0 and 90 degrees.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | # example of random rotation image augmentation  from numpy import expand\_dims  from keras. preprocessing. image import load\_img  from keras. preprocessing. image import img\_to\_array  from keras. preprocessing. image import ImageDataGenerator  from matplotlib import pyplot  # load the image  img = load\_img('bird.jpg')  # convert to numpy array  data = img\_to\_array(img)  # expand dimension to one sample  samples = expand\_dims (data, 0)  # create image data augmentation generator  datagen = ImageDataGenerator(rotation\_range=90)  # prepare iterator  it = datagen. Flow (samples, batch\_size=1)  # generate samples and plot  for i in range (9):  # define subplot  pyplot. Subplot (330 + 1 + i)  # generate batch of images  batch = it. Next ()  # convert to unsigned integers for viewing  image = batch [0]. astype('uint8')  # plot raw pixel data  pyplot. imshow (image)  # show the figure  pyplot.show() |

Running the example generates examples of the rotated image, showing in some cases pixels rotated out of the frame and the nearest-neighbor fill.



Plot of Images Generated with a Random Rotation Augmentation

**Random Brightness Augmentation**

The brightness of the image can be augmented by either randomly darkening images, brightening images, or both.

The intent is to allow a model to generalize across images trained on different lighting levels.

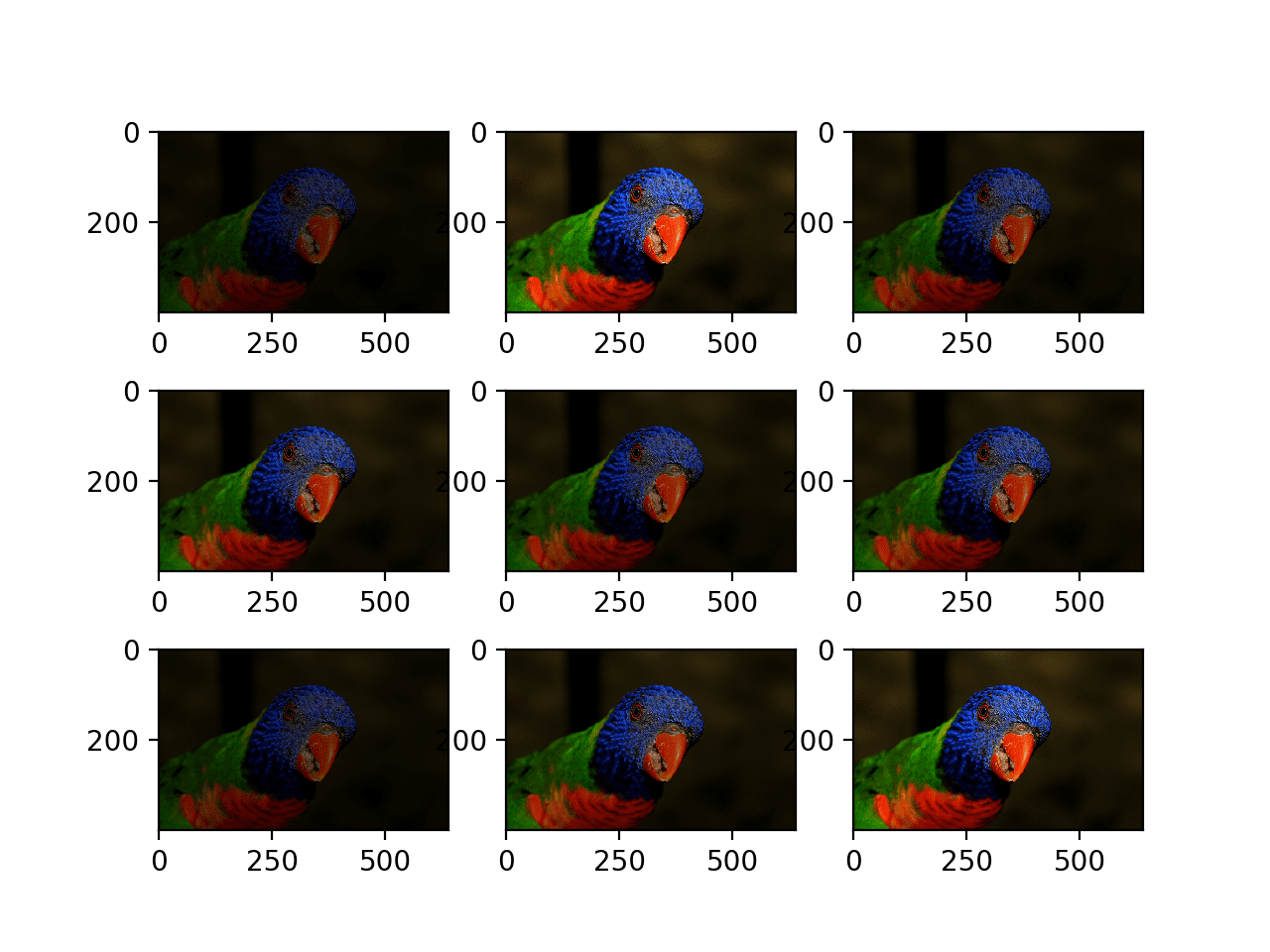
This can be achieved by specifying the *brightness\_range* argument to the *ImageDataGenerator ()* constructor that specifies min and max range as a float representing a percentage for selecting a brightening amount.

Values less than 1.0 darken the image, e.g. [0.5, 1.0], whereas values larger than 1.0 brighten the image, e.g. [1.0, 1.5], where 1.0 has no effect on brightness.

The example below demonstrates a brightness image augmentation, allowing the generator to randomly darken the image between 1.0 (no change) and 0.2 or 20%.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | # example of brighting image augmentation  from numpy import expand\_dims  from keras. Preprocessing. image import load\_img  from keras. preprocessing. image import img\_to\_array  from keras. preprocessing. image import ImageDataGenerator  from matplotlib import pyplot  # load the image  img = load\_img('bird.jpg')  # convert to numpy array  data = img\_to\_array(img)  # expand dimension to one sample  samples = expand\_dims (data, 0)  # create image data augmentation generator  datagen = ImageDataGenerator (brightness\_range= [0.2,1.0])  # prepare iterator  it = datagen. Flow (samples, batch\_size=1)  # generate samples and plot  for i in range (9):  # define subplot  pyplot. Subplot (330 + 1 + i)  # generate batch of images  batch = it. Next ()  # convert to unsigned integers for viewing  image = batch [0]. Astype ('uint8')  # plot raw pixel data  pyplot. imshow (image)  # show the figure  pyplot.show() |

**Running the example shows the augmented images with varying amounts of darkening applied.**



Plot of Images Generated with a Random Brightness Augmentation

**Random Zoom Augmentation**

A zoom augmentation randomly zooms the image in and either adds new pixel values around the image or interpolates pixel values respectively.

Image zooming can be configured by the *zoom\_range* argument to the *ImageDataGenerator* constructor. You can specify the percentage of the zoom as a single float or a range as an array or tuple.

If a float is specified, then the range for the zoom will be [1-value, 1+value]. For example, if you specify 0.3, then the range will be [0.7, 1.3], or between 70% (zoom in) and 130% (zoom out).

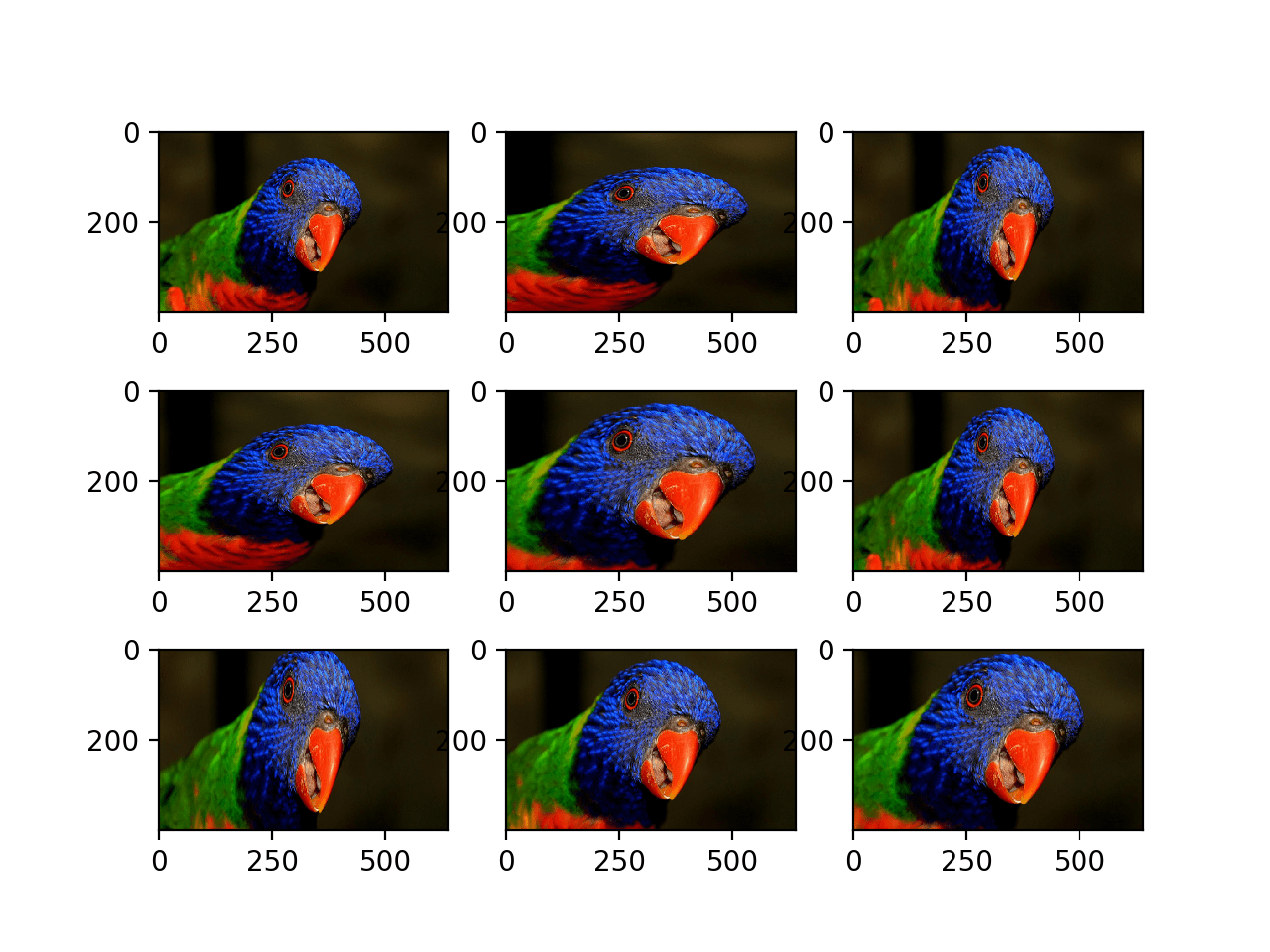
The zoom amount is uniformly randomly sampled from the zoom region for each dimension (width, height) separately.

The zoom may not feel intuitive. Note that zoom values less than 1.0 will zoom the image in, e.g. [0.5,0.5] makes the object in the image 50% larger or closer, and values larger than 1.0 will zoom the image out by 50%, e.g. [1.5, 1.5] makes the object in the image smaller or further away. A zoom of [1.0,1.0] has no effect.

The example below demonstrates zooming the image in, e.g. making the object in the photograph larger.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | # example of zoom image augmentation  from numpy import expand\_dims  from keras.preprocessing.image import load\_img  from keras. preprocessing.image import img\_to\_array  from keras. preprocessing. image import ImageDataGenerator  from matplotlib import pyplot  # load the image  img = load\_img('bird.jpg')  # convert to numpy array  data = img\_to\_array(img)  # expand dimension to one sample  samples = expand\_dims (data, 0)  # create image data augmentation generator  datagen = ImageDataGenerator (zoom\_range= [0.5,1.0])  # prepare iterator  it = datagen. Flow (samples, batch\_size=1)  # generate samples and plot  for i in range (9):  # define subplot  Pyplot.subplot (330 + 1 + i)  # generate batch of images  batch = it. next ()  # convert to unsigned integers for viewing  image = batch [0]. astype('uint8')  # plot raw pixel data  pyplot. imshow(image)  # show the figure  pyplot.show() |

Running the example generates examples of the zoomed image, showing a random zoom in that is different on both the width and height dimensions that also randomly changes the aspect ratio of the object in the image.



Plot of Images Generated with a Random Zoom Augmentation