

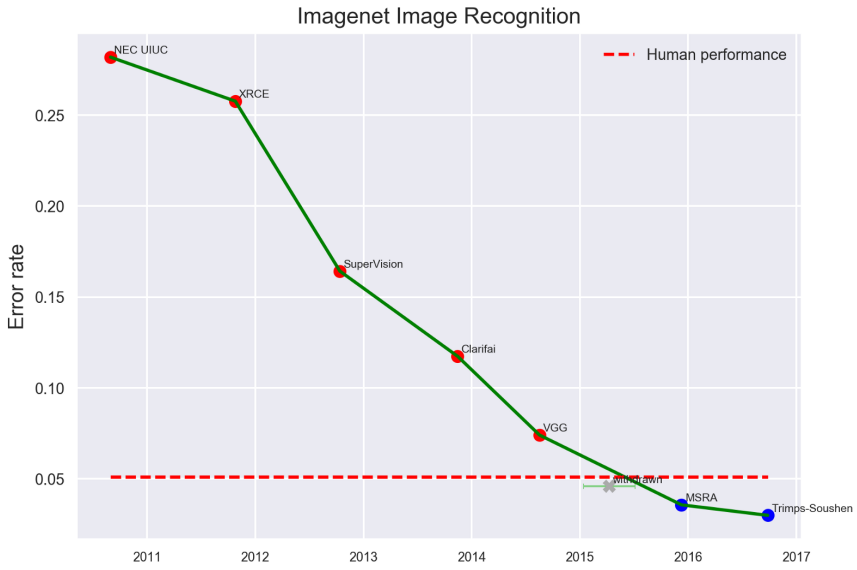
# Image Augmentation for Deep Learning using Keras and Histogram Equalization


In this post we'll go over:

- Image Augmentation: What is it? Why is it important?
- Keras: How to use it for basic Image Augmentation.
- Histogram Equalization: What is it? How is it useful?
- Implementing Histogram Equalization Techniques: one way to modify the keras.preprocessing image.py file.

## Image Augmentation: What is it? Why is it important?

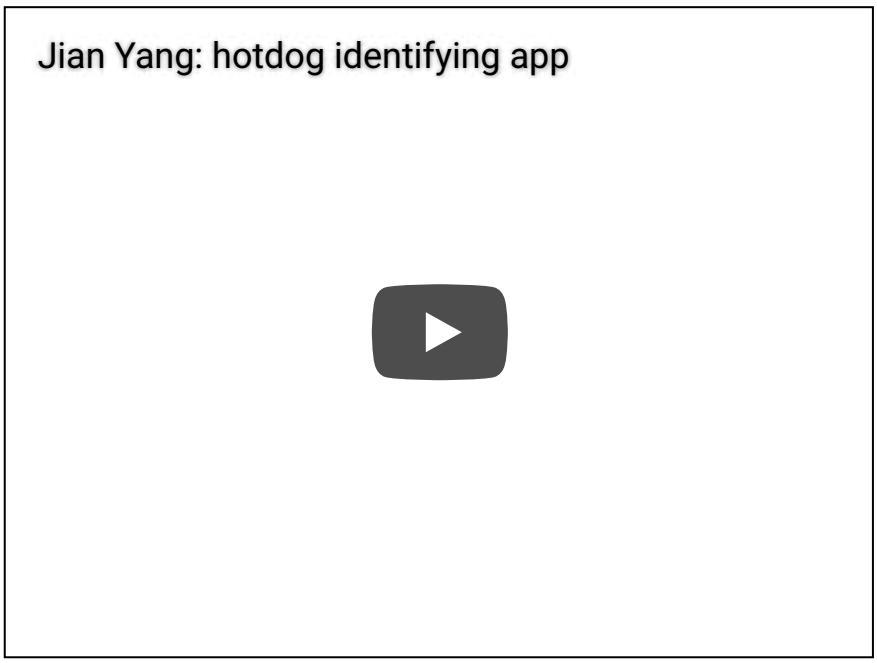
Deep Neural Networks, particularly Convolutional Neural Networks (CNNs), are particularly proficient at image classification tasks. State-of-the-art CNNs have even been shown to exceed human performance in image recognition.




https://www.eff.org/ai/metrics

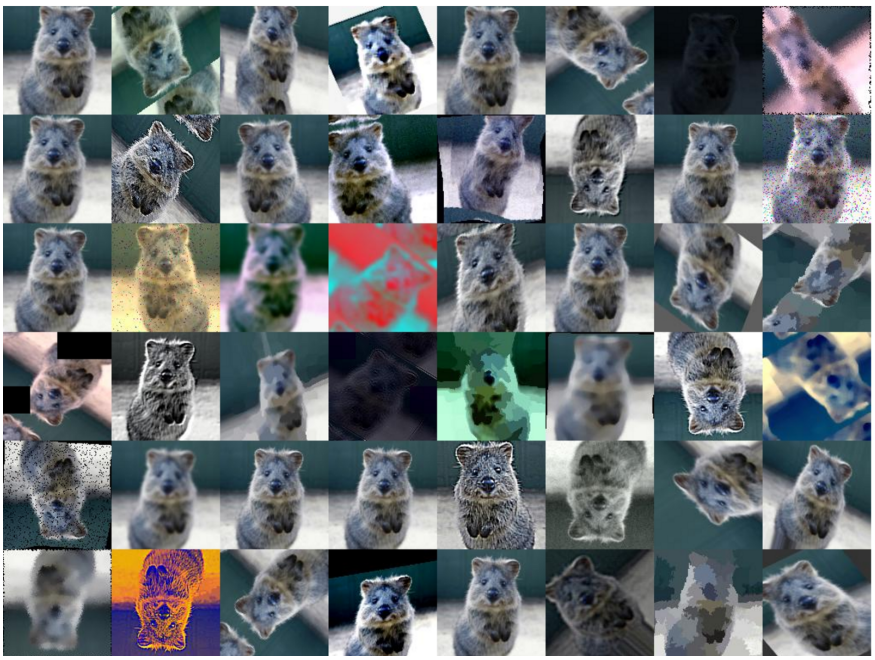
However, as we learn from Mr. Jian-Yang's "Hot Dog, Not Hot Dog" food recognition app in the popular TV show, *Silicon Valley*, (the app is now available on the app store) gathering images as training data can be both costly and time consuming.

If you aren't familiar with the TV show Silicon Valley, be warned that the language in the following clip is NSFW:




In order to combat the high expense of collecting thousands of training images, image augmentation has been developed in order to generate training data from an existing dataset. Image Augmentation is the

process of taking images that are already in a training dataset and manipulating them to create many altered versions of the same image. This both provides more images to train on, but can also help expose our classifier to a wider variety of lighting and coloring situations so as to make our classifier more robust. Here are some examples of different augmentations from the imgaug library.



https://github.com/aleju/imgaug

## Using Keras for Basic Image Augmentation

There are many ways to pre-process images. In this post we will go over some of the most common out-of-the-box methods that the keras deep learning library provides for augmenting images, then we will show how to alter the keras.preprocessing image.py file in order to enable histogram equalization methods. We will use the cifar10 dataset that comes with keras. However, we will only be using the images of cats and dogs from the dataset in order to keep the task small enough to be performed on a CPU—in case you want to follow along. You can view an IPython notebook of the source code from this post.

### Loading and Formatting the Data

The first thing that we will do is load the cifar10 dataset and format the images to prepare them for the CNN. We'll also take a peek at a few of the images just to make sure the data has loaded properly.

```
from __future__ import print_function
import keras
from keras.datasets import cifar10
from keras import backend as K
import matplotlib
from matplotlib import pyplot as plt
import numpy as np
```

```
#Input image dimensions
img_rows, img_cols = 32, 32
```

```
#The data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
#Only look at cats [=3] and dogs [=5]
train_picks = np.ravel(np.logical_or(y_train==3,y_train==5))
test_picks = np.ravel(np.logical_or(y_test==3,y_test==5))
```

```
y_train = np.array(y_train[train_picks]==5,dtype=int)
y_test = np.array(y_test[test_picks]==5,dtype=int)
```

```
x_train = x_train[train_picks]
x_test = x_test[test_picks]
```

```
if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 3, img_rows,
img_cols)
    x_test = x_test.reshape(x_test.shape[0], 3, img_rows,
img_cols)
    input_shape = (3, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows,
img_cols, 3)
    x_test = x_test.reshape(x_test.shape[0], img_rows,
img_cols, 3)
    input_shape = (img_rows, img_cols, 3)
```

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

```
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
#Convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(np.ravel(y_train),
num_classes)
y_test = keras.utils.to_categorical(np.ravel(y_test),
num_classes)
```

```
#Look at the first 9 images from the dataset
```

```
images = range(0,9)
```

```
for i in images:
    plt.subplot(330 + 1 + i)
    plt.imshow(x_train[i], cmap=pyplot.get_cmap('gray'))
#Show the plot
plt.show()
```

The cifar10 images are only 32 x 32 pixels, so they look grainy when magnified here, but the CNN doesn't know it's grainy, all it sees is DATA.

## Create an image generator from ImageDataGenerator()

Augmenting our image data with keras is dead simple. A shoutout to Jason Brownlee who provides a great tutorial on this. First we need to create an image generator by calling the `ImageDataGenerator()` function and pass it a list of parameters describing the alterations that we want it to perform on the images. We will then call the `fit()` function on our image generator which will apply the changes to the images batch by batch. By default, the modifications will be applied randomly, so not every image will be changed every time. You can also use `keras.preprocessing` to export augmented image files to a folder in order to build up a giant dataset of altered images should you desire to do so.

We'll look at some of the more visually interesting augmentations here. A description of the all of the possible `ImageDataGenerator()`

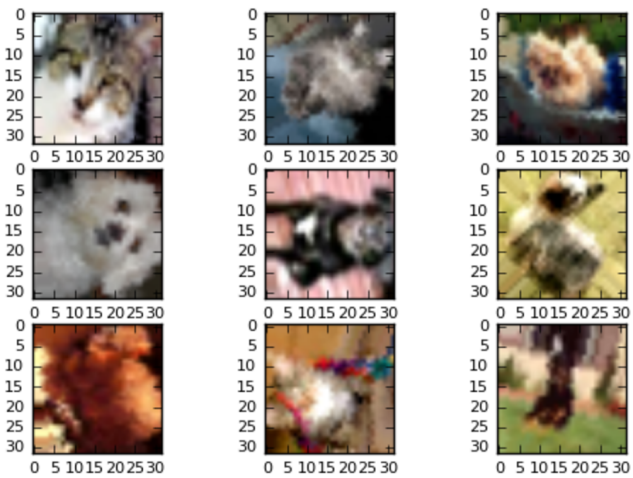
parameters as well as a list of the other methods available in `keras.preprocessing` can be seen in the keras documentation.

## Randomly Rotate Images

```
# Rotate images by 90 degrees
datagen = ImageDataGenerator(rotation_range=90)
```

```
# fit parameters from data
datagen.fit(x_train)
```

```
# Configure batch size and retrieve one batch of images
for X_batch, y_batch in datagen.flow(x_train, y_train,
batch_size=9):
    # Show 9 images
    for i in range(0, 9):
        pyplot.subplot(330 + 1 + i)
        pyplot.imshow(X_batch[i].reshape(img_rows, img_cols,
3))
    # show the plot
    pyplot.show()
    break
```



## Flip Images Vertically

```
# Flip images vertically
datagen = ImageDataGenerator(vertical_flip=True)
```

```
# fit parameters from data
datagen.fit(x_train)
```

```
# Configure batch size and retrieve one batch of images
for X_batch, y_batch in datagen.flow(x_train, y_train,
batch_size=9):
    # Show 9 images
    for i in range(0, 9):
        pyplot.subplot(330 + 1 + i)
        pyplot.imshow(X_batch[i].reshape(img_rows, img_cols,
3))
    # show the plot
    pyplot.show()
    break
```

Flipping images horizontally is also one of the classic ways of generating more data for a classifier. It is just as easy to do and probably makes more sense with this dataset, however, I've left out the code and images because there's no way of knowing whether a dog or cat image has been flipped horizontally without seeing the original.

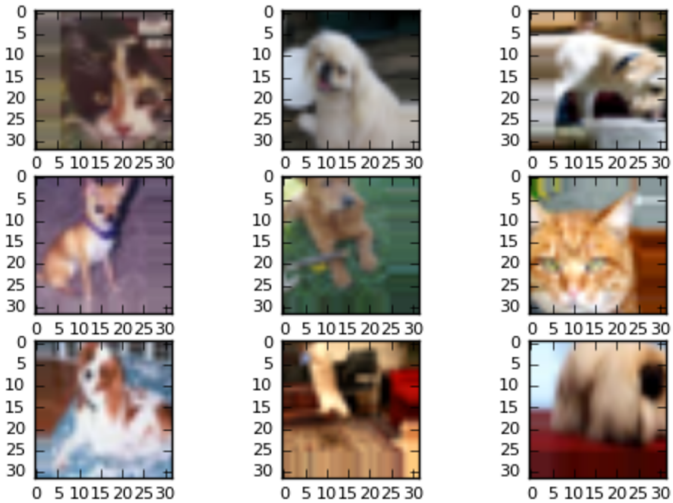
## Shift Images Vertically or Horizontally by 20%

```
# Shift images vertically or horizontally
# Fill missing pixels with the color of the nearest pixel
datagen = ImageDataGenerator(width_shift_range=.2,
                             height_shift_range=.2,
                             fill_mode='nearest')
```

```
# fit parameters from data
datagen.fit(x_train)
```

```
# Configure batch size and retrieve one batch of images
for X_batch, y_batch in datagen.flow(x_train, y_train,
batch_size=9):
    # Show 9 images
    for i in range(0, 9):
        pyplot.subplot(330 + 1 + i)
```

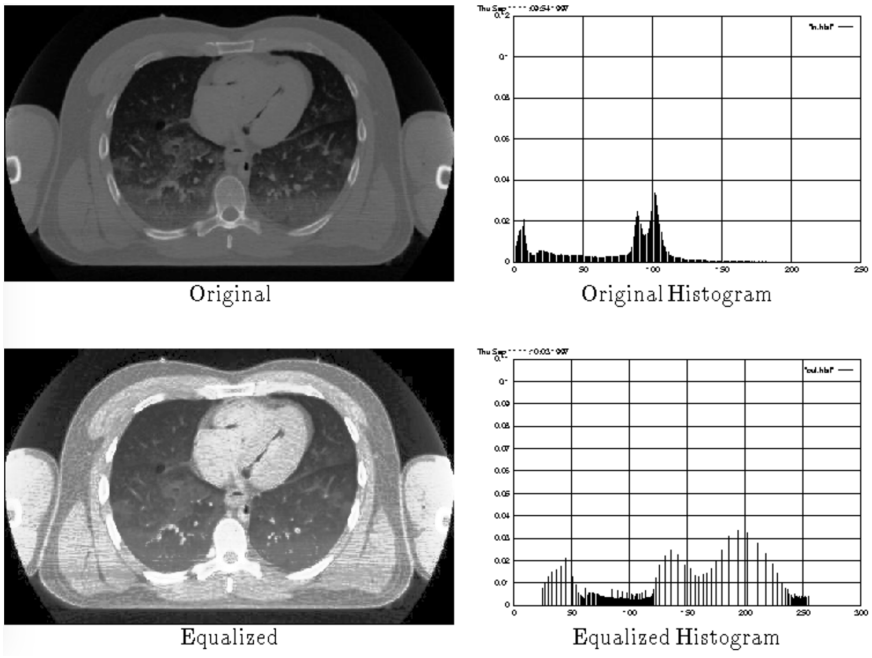
```
        pyplot.imshow(X_batch[i].reshape(img_rows, img_cols,
3))
    # show the plot
    pyplot.show()
    break
```



## Histogram Equalization Techniques

Histogram Equalization is the process taking a low contrast image and increasing the contrast between the image's relative highs and lows in order to bring out subtle differences in shade and create a higher contrast image. The results can be striking, especially for grayscale images. Here are some examples:

https://www.bruzed.com/2009/10/contrast-stretching-and-histogram-equalization/




http://www-classes.usc.edu/engr/ee-s/569/qa2/Histogram%20Equalization.htm


https://studentathome.wordpress.com/2013/03/27/local-histogram-equalization/

In this post we'll be looking at three image augmentation techniques for improving contrast in images. These approaches are sometimes also referred to as "Histogram Stretching" because they take the distribution of pixel intensities and stretch the distribution to fit a wider range of values thereby increasing the level of contrast between the lightest and darkest portions of an image.

## Histogram Equalization

Histogram Equalization increases contrast in images by detecting the distribution of pixel densities in an image and plotting these pixel densities on a histogram. The distribution of this histogram is then analyzed and if there are ranges of pixel brightnesses that aren't currently being utilized, the histogram is then "stretched" to cover

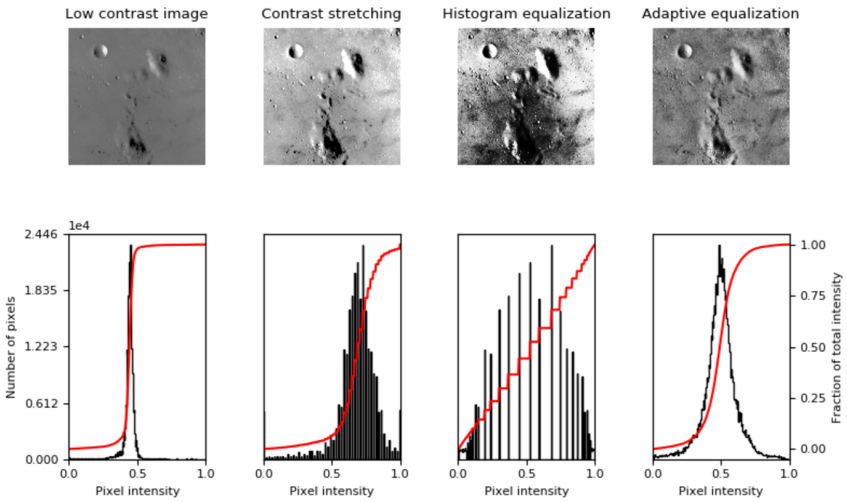
those ranges, and then is "back projected" onto the image to increase the overall contrast of the image.

## Contrast Stretching

Contrast Stretching takes the approach of analyzing the distribution of pixel densities in an image and then "rescales the image to include all intensities that fall within the 2nd and 98th percentiles."

## Adaptive Equalization

Adaptive Equalization differs from regular histogram equalization in that several different histograms are computed, each corresponding to a different section of the image; however, it has a tendency to over-amplify noise in otherwise uninteresting sections.




The following code comes from the sci-kit image library's docs and has been altered to perform the three above augmentations on the first image of our cifar10 dataset. First we will import the necessary modules from the sci-kit image (skimage) library and then modify the code from the sci-kit image documentation to view the augmentations on the first image of our dataset.

```
# Import skimage modules
from skimage import data, img_as_float
from skimage import exposure
```

```
# Lets try augmenting a cifar10 image using these techniques
from skimage import data, img_as_float
from skimage import exposure
```

```
# Load an example image from cifar10 dataset
img = images[0]
```

```
# Set font size for images
matplotlib.rcParams['font.size'] = 8
```

```
# Contrast stretching
p2, p98 = np.percentile(img, (2, 98))
img_rescale = exposure.rescale_intensity(img, in_range=(p2,
p98))
```

```
# Histogram Equalization
img_eq = exposure.equalize_hist(img)
```

```
# Adaptive Equalization
img_adapteq = exposure.equalize_adapthist(img,
clip_limit=0.03)
```

```
#### Everything below here is just to create the plot/graphs
####
```

```
# Display results
fig = plt.figure(figsize=(8, 5))
axes = np.zeros((2, 4), dtype=np.object)
axes[0, 0] = fig.add_subplot(2, 4, 1)
for i in range(1, 4):
    axes[0, i] = fig.add_subplot(2, 4, 1+i,
sharex=axes[0,0], sharey=axes[0,0])
for i in range(0, 4):
    axes[1, i] = fig.add_subplot(2, 4, 5+i)
```

```
ax_img, ax_hist, ax_cdf = plot_img_and_hist(img, axes[:, 0])
ax_img.set_title('Low contrast image')
```

```
y_min, y_max = ax_hist.get_ylim()
ax_hist.set_ylabel('Number of pixels')
ax_hist.set_yticks(np.linspace(0, y_max, 5))
```

```
ax_img, ax_hist, ax_cdf = plot_img_and_hist(img_rescale,
axes[:, 1])
ax_img.set_title('Contrast stretching')
```

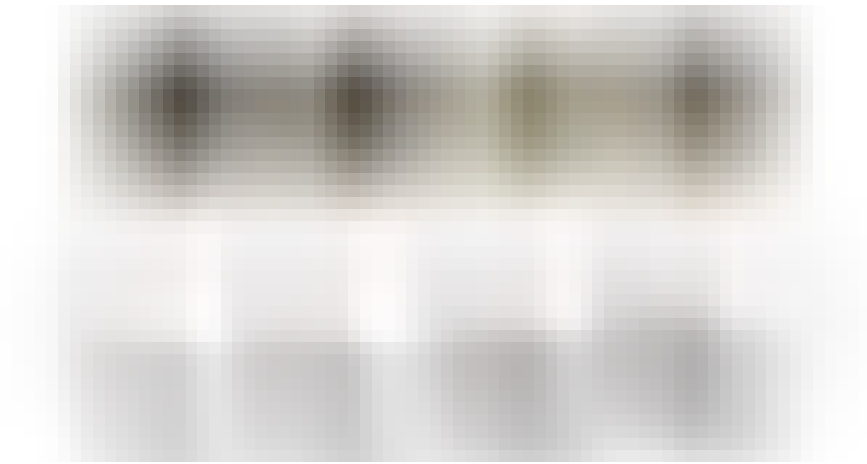
```
ax_img, ax_hist, ax_cdf = plot_img_and_hist(img_eq, axes[:,
2])
ax_img.set_title('Histogram equalization')
```

```
ax_img, ax_hist, ax_cdf = plot_img_and_hist(img_adapteq,
axes[:, 3])
ax_img.set_title('Adaptive equalization')
```

```
ax_cdf.set_ylabel('Fraction of total intensity')
ax_cdf.set_yticks(np.linspace(0, 1, 5))
```

```
# prevent overlap of y-axis labels
fig.tight_layout()
plt.show()
```

Here are the modified images of a low contrast cat from the cifar10 dataset. As you can see, the results are not as striking as they might be with a low contrast grayscale image, but still help improve the quality of the images.



## Modifying keras.preprocessing to enable Histogram Equalization techniques.

Now that we have successfully modified one image from the cifar10 dataset, we will demonstrate how to alter the keras.preprocessing image.py file in order to execute these different histogram modification techniques just as we did the out-of-the-box keras augmentations using `ImageDataGenerator()`. Here are the general steps that we will follow in order to implement this functionality:

### Overview

- Find the keras.preprocessing image.py file on your own machine.
- Copy the image.py file into your file or notebook.
- Add one attribute for each equalization technique to the DataImageGenerator() init function.
- Add IF statement clauses to the random_transform method so that augmentations get implemented when we call `datagen.fit()`.

One of the simplest ways to make alterations to keras.preprocessing's `image.py` file is simply to copy and paste its contents into our code. This will then remove the need to import it. You can view the contents of the `image.py` file on github here. However, in order to be sure that

you're grabbing the same version of the file that you were importing previously, it's better to grab the `image.py` file that is already on your machine. Running `print(keras.__file__)` will print out the path to the keras library that is on your machine. The path (for mac users) may look something like:

```
/usr/local/lib/python3.5/dist-packages/keras/__init__.pyc
```

This gives us the path to keras on our local machine. Go ahead and navigate there and then into the `preprocessing` folder. Inside `preprocessing` you will see the `image.py` file. You can then copy its contents into your code. The file is long, but for beginners this is probably one of the easiest ways to go about making alterations to it.

## Editing image.py

At the top of image.py you can comment out the line: `from ..import backend as K` if you have already included it above.

At this point, also double-check to make sure that you're importing the necessary scikit-image modules so that the copied `image.py` can see them.

```
from skimage import data, img_as_float
from skimage import exposure
```

We now need to add six lines to the ImageDataGenerator class's `__init__` method so that it has the three properties that represent the types of augmentation that we're going to be adding. The code below is copied from my current image.py. The lines with `#####` to the side are lines that I have added.

```
def __init__(self,
 contrast_stretching=False, #####
 histogram_equalization=False,#####
 adaptive_equalization=False, #####
 featurewise_center=False,
 samplewise_center=False,
 featurewise_std_normalization=False,
 samplewise_std_normalization=False,
 zca_whitening=False,
 rotation_range=0.,
```

```
 width_shift_range=0.,
 height_shift_range=0.,
 shear_range=0.,
 zoom_range=0.,
 channel_shift_range=0.,
 fill_mode='nearest',
 cval=0.,
 horizontal_flip=False,
 vertical_flip=False,
 rescale=None,
 preprocessing_function=None,
 data_format=None):
 if data_format is None:
 data_format = K.image_data_format()
 self.counter = 0
 self.contrast_stretching = contrast_stretching, #####
 self.adaptive_equalization = adaptive_equalization #####
 self.histogram_equalization = histogram_equalization #####
 self.featurewise_center = featurewise_center
 self.samplewise_center = samplewise_center
 self.featurewise_std_normalization =
featurewise_std_normalization
 self.samplewise_std_normalization =
samplewise_std_normalization
 self.zca_whitening = zca_whitening
 self.rotation_range = rotation_range
 self.width_shift_range = width_shift_range
 self.height_shift_range = height_shift_range
 self.shear_range = shear_range
 self.zoom_range = zoom_range
 self.channel_shift_range = channel_shift_range
 self.fill_mode = fill_mode
 self.cval = cval
 self.horizontal_flip = horizontal_flip
 self.vertical_flip = vertical_flip
 self.rescale = rescale
 self.preprocessing_function = preprocessing_function
```

The `random_transform()` function (below) responds to the arguments we have been passed into the `ImageDataGenerator()` function. If we have set the `contrast_stretching`, `adaptive_equalization`, or `histogram_equalization` parameters to `True`, when we call `ImageDataGenerator()`, (just like we would for the other image augmentations) `random_transform()` will then apply the desired image augmentation.

```
def random_transform(self, x):
```

```
  img_row_axis = self.row_axis - 1
  img_col_axis = self.col_axis - 1
  img_channel_axis = self.channel_axis - 1
```

```
# use composition of homographies
# to generate final transform that needs to be applied
  if self.rotation_range:
```

```
      theta = np.pi / 180 * np.random.uniform(-
self.rotation_range, self.rotation_range)
  else:
      theta = 0


  if self.height_shift_range:
      tx = np.random.uniform(-self.height_shift_range,
self.height_shift_range) * x.shape[img_row_axis]
  else:
      tx = 0


  if self.width_shift_range:
      ty = np.random.uniform(-self.width_shift_range,
self.width_shift_range) * x.shape[img_col_axis]
  else:
      ty = 0


  if self.shear_range:
      shear = np.random.uniform(-self.shear_range,
self.shear_range)
  else:
      shear = 0


  if self.zoom_range[0] == 1 and self.zoom_range[1] == 1:
      zx, zy = 1, 1
  else:
      zx, zy = np.random.uniform(self.zoom_range[0],
self.zoom_range[1], 2)


transform_matrix = None
   if theta != 0:
       rotation_matrix = np.array([[np.cos(theta), -
np.sin(theta), 0],
                                   [np.sin(theta),
np.cos(theta), 0],
                                   [0, 0, 1]])
       transform_matrix = rotation_matrix


  if tx != 0 or ty != 0:
    shift_matrix = np.array([[1, 0, tx],
                             [0, 1, ty],
                             [0, 0, 1]])
    transform_matrix = shift_matrix if transform_matrix is
None else np.dot(transform_matrix, shift_matrix)


  if shear != 0:
      shear_matrix = np.array([[1, -np.sin(shear), 0],
                               [0, np.cos(shear), 0],
                               [0, 0, 1]])
      transform_matrix = shear_matrix if transform_matrix is
None else np.dot(transform_matrix, shear_matrix)


  if zx != 1 or zy != 1:
      zoom_matrix = np.array([[zx, 0, 0],
                              [0, zy, 0],
                              [0, 0, 1]])
      transform_matrix = zoom_matrix if transform_matrix is
None else np.dot(transform_matrix, zoom_matrix)
```

```
  if transform_matrix is not None:
     h, w = x.shape[img_row_axis], x.shape[img_col_axis]
           transform_matrix =
transform_matrix_offset_center(transform_matrix, h, w)
     x = apply_transform(x, transform_matrix,
img_channel_axis,
                                 fill_mode=self.fill_mode,
cval=self.cval)
```

```
  if self.channel_shift_range != 0:
     x = random_channel_shift(x, self.channel_shift_range,
img_channel_axis)
```

```
  if self.horizontal_flip:
     if np.random.random() < 0.5:
        x = flip_axis(x, img_col_axis)
```

```
  if self.vertical_flip:
     if np.random.random() < 0.5:
        x = flip_axis(x, img_row_axis)

  if self.contrast_stretching: #####
     if np.random.random() < 0.5: #####
        p2, p98 = np.percentile(x, (2, 98)) #####
        x = exposure.rescale_intensity(x, in_range=(p2,
p98)) #####

  if self.adaptive_equalization: #####
     if np.random.random() < 0.5: #####
        x = exposure.equalize_adapthist(x, clip_limit=0.03)
#####

  if self.histogram_equalization: #####
     if np.random.random() < 0.5: #####
        x = exposure.equalize_hist(x) #####

  return x
```

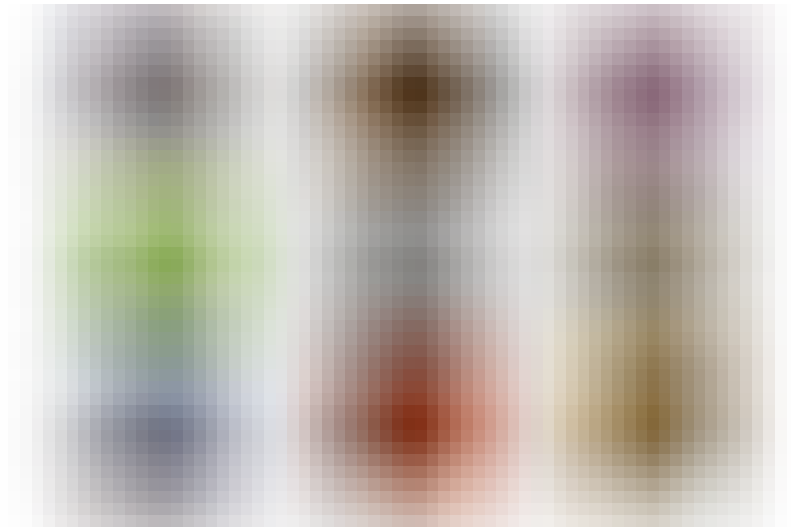
Now we have all of the necessary code in place and can call ImageDataGenerator() to perform our histogram modification techniques. Here's what a few images look like if we set all three of the values to `True`.

```
# Initialize Generator
datagen = ImageDataGenerator(contrast_stretching=True,
adaptive_equalization=True, histogram_equalization=True)
```

```
# fit parameters from data
datagen.fit(x_train)
```

```
# Configure batch size and retrieve one batch of images
for x_batch, y_batch in datagen.flow(x_train, y_train,
batch_size=9):
    # Show the first 9 images
    for i in range(0, 9):
```

```
        pyplot.subplot(330 + 1 + i)
        pyplot.imshow(x_batch[i].reshape(img_rows, img_cols,
3))
    # show the plot
    pyplot.show()
    break
```



I don't recommend setting more than one of them to `True` for any given dataset. Make sure you experiment with your particular dataset in order to see what helps improve your classifier's accuracy. For colored images, I've found that contrast stretching often obtains better results than histogram modification or adaptive equalization.

## Train and Validate your Keras CNN

The final step is to train our CNN and validate the model using `model.fit_generator()` in order to train and validate our neural network on the augmented images.

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
```

```
batch_size = 64
num_classes = 2
epochs = 10
```

```
model = Sequential()
model.add(Conv2D(4, kernel_size=(3,
3),activation='relu',input_shape=input_shape))
model.add(Conv2D(8, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
```

```
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
             metrics=['accuracy'])
```

```
datagen.fit(x_train)
history = model.fit_generator(datagen.flow(x_train, y_train,
batch_size=batch_size),
                        steps_per_epoch=x_train.shape[0] //
batch_size,
                        epochs=20,
                        validation_data=(x_test, y_test))
```