



[Click to Take the FREE Computer Vision Crash-Course](#)

Search...



## How to Normalize, Center, and Standardize Image Pixels in Keras

by **Jason Brownlee** on April 3, 2019 in **Deep Learning for Computer Vision**

Tweet

Share

Share

Last Updated on July 5, 2019

The pixel values in images must be scaled prior to providing the images as input to a deep learning neural network model during the training or evaluation of the model.

Traditionally, the images would have to be scaled prior to the development of the model and stored in memory or on disk in the scaled format.

An alternative approach is to scale the images using a preferred scaling technique just-in-time during the training or model evaluation process. Keras supports this type of data preparation for image data via the ImageDataGenerator class and API.

In this tutorial, you will discover how to use the ImageDataGenerator class to scale pixel data just-in-time when fitting and evaluating deep learning neural network models.

After completing this tutorial, you will know:

- How to configure and use the ImageDataGenerator to scale pixel data.
- How to use the ImageDataGenerator to normalize pixel data.
- How to use the ImageDataGenerator to center and standardize pixel data.

Discover how to build models for photo classification, [new computer vision book](#), with 30 step-by-step tutorials.

Let's get started.

### Start Machine Learning

You can master applied Machine Learning **without math or fancy degrees.** Find out how in this *free* and *practical* course.

Email Address

START MY EMAIL COURSE



How to Normalize, Center, and Standardize Images With the ImageDataGenerator in Keras  
Photo by [Sagar](#), some rights reserved.

## Tutorial Overview

This tutorial is divided into five parts; they are:

1. MNIST Handwritten Image Classification Dataset
2. ImageDataGenerator class for Pixel Scaling
3. How to Normalize Images With ImageDataGenerator
4. How to Center Images With ImageDataGenerator
5. How to Standardize Image With ImageDataGenerator

## MNIST Handwritten Image Classification Dataset

Before we dive into the usage of the ImageDataGenerator class for preparing image data, we must select an image dataset on which to test the generator.

The [MNIST problem](#), is an image classification problem.

The goal of the problem is to classify a given image of handwritten digits. In other words, such, it is a multiclass image classification problem.

This dataset is provided as part of the Keras library and is loaded into memory by a call to the `keras.datasets.mnist` function.

The function returns two tuples: one for the training data and one for the test data. For example:

### Start Machine Learning

You can master applied Machine Learning **without math or fancy degrees**. Find out how in this *free* and *practical* course.

START MY EMAIL COURSE

```
1 # example of loading the MNIST dataset
2 from keras.datasets import mnist
3 (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

We can load the MNIST dataset and summarize the dataset. The complete example is listed below.

```
1 # load and summarize the MNIST dataset
2 from keras.datasets import mnist
3 # load dataset
4 (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
5 # summarize dataset shape
6 print('Train', train_images.shape, train_labels.shape)
7 print('Test', (test_images.shape, test_labels.shape))
8 # summarize pixel values
9 print('Train', train_images.min(), train_images.max(), train_images.mean(), train_images.std())
10 print('Test', test_images.min(), test_images.max(), test_images.mean(), test_images.std())
```

Running the example first loads the dataset into memory. Then the shape of the train and test datasets is reported.

We can see that all images are 28 by 28 pixels with a single channel for black-and-white images. There are 60,000 images for the training dataset and 10,000 for the test dataset.

We can also see that pixel values are integer values between 0 and 255 and that the mean and standard deviation of the pixel values are similar between the two datasets.

```
1 Train (60000, 28, 28) (60000,)
2 Test ((10000, 28, 28), (10000,))
3 Train 0 255 33.318421449829934 78.56748998339798
4 Test 0 255 33.791224489795916 79.17246322228644
```

We will use this dataset to explore different pixel scaling methods using the ImageDataGenerator class in Keras.

## Want Results with Deep Learning for Computer Vision?

Take my free 7-day email crash course now (with sample code).

Click to sign-up and also get a free PDF Ebook version of the course.

Download Your Free PDF

### Start Machine Learning

You can master applied Machine Learning **without math or fancy degrees**. Find out how in this *free* and *practical* course.

START MY EMAIL COURSE

## ImageDataGenerator Class for

The ImageDataGenerator class in Keras provides a suite of methods to generate an image dataset prior to modeling.

The class will wrap your image dataset, then when requested, it will return images in batches to the algorithm during training, validation, or evaluation and apply the scaling operations just-in-time. This provides an efficient and convenient approach to scaling image data when modeling with neural networks.

The usage of the ImageDataGenerator class is as follows.

- 1. Load your dataset.
- 2. Configure the ImageDataGenerator (e.g. construct an instance).
- 3. Calculate image statistics (e.g. call the *fit()* function).
- 4. Use the generator to fit the model (e.g. pass the instance to the *fit\_generator()* function).
- 5. Use the generator to evaluate the model (e.g. pass the instance to the *evaluate\_generator()* function).

The ImageDataGenerator class supports a number of pixel scaling methods, as well as a range of [data augmentation techniques](#). We will focus on the pixel scaling techniques and leave the data augmentation methods to a later discussion.

The three main types of pixel scaling techniques supported by the ImageDataGenerator class are as follows:

- **Pixel Normalization:** scale pixel values to the range 0-1.
- **Pixel Centering:** scale pixel values to have a zero mean.
- **Pixel Standardization:** scale pixel values to have a zero mean and unit variance.

The pixel standardization is supported at two levels: either per-image (called sample-wise) or per-dataset (called feature-wise). Specifically, the mean and/or mean and standard deviation statistics required to standardize pixel values can be calculated from the pixel values in each image only (sample-wise) or across the entire training dataset (feature-wise).

Other pixel scaling methods are supported, such as ZCA, brightening, and more, but we will focus on these three most common methods.

The choice of pixel scaling is selected by specifying arguments to the ImageDataGenerator when an instance is constructed; for example:

```
1 # create and configure the data generator
2 datagen = ImageDataGenerator(...)
```

Next, if the chosen scaling method requires that statistics can be calculated and stored by calling

When evaluating and selecting a model, it is common and then apply them to the validation and test dataset

```
1 # calculate scaling statistics on the training
2 datagen.fit(trainX)
```

## Start Machine Learning

You can master applied Machine Learning **without math or fancy degrees**. Find out how in this *free* and *practical* course.

START MY EMAIL COURSE

Once prepared, the data generator can be used to fit a neural network model by calling the `flow()` function to retrieve an iterator that returns batches of samples and passing it to the `fit_generator()` function.

```
1 # get batch iterator
2 train_iterator = datagen.flow(trainX, trainy)
3 # fit model
4 model.fit_generator(train_iterator, ...)
```

If a validation dataset is required, a separate batch iterator can be created from the same data generator that will perform the same pixel scaling operations and use any required statistics calculated on the training dataset.

```
1 # get batch iterator for training
2 train_iterator = datagen.flow(trainX, trainy)
3 # get batch iterator for validation
4 val_iterator = datagen.flow(valX, valy)
5 # fit model
6 model.fit_generator(train_iterator, validation_data=val_iterator, ...)
```

Once fit, the model can be evaluated by creating a batch iterator for the test dataset and calling the `evaluate_generator()` function on the model.

Again, the same pixel scaling operations will be performed and any statistics calculated on the training dataset will be used, if needed.

```
1 # get batch iterator for testing
2 test_iterator = datagen.flow(testX, testy)
3 # evaluate model loss on test dataset
4 loss = model.evaluate_generator(test_iterator, ...)
```

Now that we are familiar with how to use the `ImageDataGenerator` class for scaling pixel values, let's look at some specific examples.

## How to Normalize Images With ImageDataGenerator

The `ImageDataGenerator` class can be used to rescale pixel values from the range of 0-255 to the range 0-1 preferred for neural network models.

Scaling data to the range of 0-1 is traditionally referred to as normalization.

This can be achieved by setting the `rescale` argument to `1./255` to achieve the desired range.

In this case, the ratio is  $1/255$  or about 0.0039. For example:

```
1 # create generator (1.0/255.0 = 0.0039215686274509803)
2 datagen = ImageDataGenerator(rescale=1.0/255.0)
```

The `ImageDataGenerator` does not need to be fit in the training process; the scaling factor can be calculated once and used for all data.

### Start Machine Learning

You can master applied Machine Learning **without math or fancy degrees.**  
Find out how in this *free* and *practical* course.

START MY EMAIL COURSE



Next, iterators can be created using the generator for both the train and test datasets. We will use a batch size of 64. This means that each of the train and test datasets of images are divided into groups of 64 images that will then be scaled when returned from the iterator.

We can see how many batches there will be in one epoch, e.g. one pass through the training dataset, by printing the length of each iterator.

```
1 # prepare an iterators to scale images
2 train_iterator = datagen.flow(trainX, trainY, batch_size=64)
3 test_iterator = datagen.flow(testX, testY, batch_size=64)
4 print('Batches train=%d, test=%d' % (len(train_iterator), len(test_iterator)))
```

We can then confirm that the pixel normalization has been performed as expected by retrieving the first batch of scaled images and inspecting the min and max pixel values.

```
1 # confirm the scaling works
2 batchX, batchy = train_iterator.next()
3 print('Batch shape=%s, min=%.3f, max=%.3f' % (batchX.shape, batchX.min(), batchX.max()))
```

Next, we can use the data generator to fit and evaluate a model. We will define a simple convolutional neural network model and fit it on the *train\_iterator* for five epochs with 60,000 samples divided by 64 samples per batch, or about 938 batches per epoch.

```
1 # fit model with generator
2 model.fit_generator(train_iterator, steps_per_epoch=len(train_iterator), epochs=5)
```

Once fit, we will evaluate the model on the test dataset, with about 10,000 images divided by 64 samples per batch, or about 157 steps in a single epoch.

```
1 _, acc = model.evaluate_generator(test_iterator, steps=len(test_iterator), verbose=0)
2 print('Test Accuracy: %.3f' % (acc * 100))
```

We can tie all of this together; the complete example is listed below.

```
1 # example of using ImageDataGenerator to normalize images
2 from keras.datasets import mnist
3 from keras.utils import to_categorical
4 from keras.models import Sequential
5 from keras.layers import Conv2D
6 from keras.layers import MaxPooling2D
7 from keras.layers import Dense
8 from keras.layers import Flatten
9 from keras.preprocessing.image import ImageDataGenerator
10 # load dataset
11 (trainX, trainY), (testX, testY) = mnist.load_data()
12 # reshape dataset to have a single channel
13 width, height, channels = trainX.shape[1], trainX.shape[2], trainX.shape[3]
14 trainX = trainX.reshape((trainX.shape[0], width, height, channels))
15 testX = testX.reshape((testX.shape[0], width, height, channels))
16 # one hot encode target values
17 trainY = to_categorical(trainY)
18 testY = to_categorical(testY)
19 # confirm scale of pixels
20 print('Train min=%.3f, max=%.3f' % (trainX.min(), trainX.max()))
21 print('Test min=%.3f, max=%.3f' % (testX.min(), testX.max()))
22 # create generator (1.0/255.0 = 0.00392156862745098)
23 datagen = ImageDataGenerator(rescale=1.0/255.0)
24 # prepare an iterators to scale images
```

## Start Machine Learning ×

You can master applied Machine Learning  
**without math or fancy degrees.**  
Find out how in this *free* and *practical* course.

START MY EMAIL COURSE

```

25 train_iterator = datagen.flow(trainX, trainY, batch_size=64)
26 test_iterator = datagen.flow(testX, testY, batch_size=64)
27 print('Batches train=%d, test=%d' % (len(train_iterator), len(test_iterator)))
28 # confirm the scaling works
29 batchX, batchy = train_iterator.next()
30 print('Batch shape=%s, min=%.3f, max=%.3f' % (batchX.shape, batchX.min(), batchX.max()))
31 # define model
32 model = Sequential()
33 model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(width, height, channels)))
34 model.add(MaxPooling2D((2, 2)))
35 model.add(Conv2D(64, (3, 3), activation='relu'))
36 model.add(MaxPooling2D((2, 2)))
37 model.add(Flatten())
38 model.add(Dense(64, activation='relu'))
39 model.add(Dense(10, activation='softmax'))
40 # compile model
41 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
42 # fit model with generator
43 model.fit_generator(train_iterator, steps_per_epoch=len(train_iterator), epochs=5)
44 # evaluate model
45 _, acc = model.evaluate_generator(test_iterator, steps=len(test_iterator), verbose=0)
46 print('Test Accuracy: %.3f' % (acc * 100))

```

Running the example first reports the min and max pixel values on the train and test sets. This confirms that indeed the raw data has pixel values in the range 0-255.

Next, the data generator is created and the iterators are prepared. We can see that we have 938 batches per epoch with the training dataset and 157 batches per epoch with the test dataset.

We retrieve the first batch from the dataset and confirm that it contains 64 images with the height and width (rows and columns) of 28 pixels and 1 channel, and that the new minimum and maximum pixel values are 0 and 1 respectively. This confirms that the normalization has had the desired effect.

```

1 Train min=0.000, max=255.000
2 Test min=0.000, max=255.000
3 Batches train=938, test=157
4 Batch shape=(64, 28, 28, 1), min=0.000, max=1.000

```

The model is then fit on the normalized image data. Training does not take long on the CPU. Finally, the model is evaluated in the test dataset, applying the same normalization.

```

1 Epoch 1/5
2 938/938 [=====] - 12s 13ms/step - loss: 0.1841 - acc: 0.9448
3 Epoch 2/5
4 938/938 [=====] - 12s
5 Epoch 3/5
6 938/938 [=====] - 12s
7 Epoch 4/5
8 938/938 [=====] - 12s
9 Epoch 5/5
10 938/938 [=====] - 12s
11 Test Accuracy: 99.050

```

Now that we are familiar with how to use the ImageDataGenerator for image normalization, let's look at examples of pixel centering

## How to Center Images With ImageDataGenerator

### Start Machine Learning

You can master applied Machine Learning **without math or fancy degrees.** Find out how in this *free* and *practical* course.

START MY EMAIL COURSE

Another popular pixel scaling method is to calculate the mean pixel value across the entire training dataset, then subtract it from each image.

This is called centering and has the effect of centering the distribution of pixel values on zero: that is, the mean pixel value for centered images will be zero.

The ImageDataGenerator class refers to centering that uses the mean calculated on the training dataset as feature-wise centering. It requires that the statistic is calculated on the training dataset prior to scaling.

```
1 # create generator that centers pixel values
2 datagen = ImageDataGenerator(featurewise_center=True)
3 # calculate the mean on the training dataset
4 datagen.fit(trainX)
```

It is different to calculating of the mean pixel value for each image, which Keras refers to as sample-wise centering and does not require any statistics to be calculated on the training dataset.

```
1 # create generator that centers pixel values
2 datagen = ImageDataGenerator(samplewise_center=True)
```

We will demonstrate feature-wise centering in this section. Once the statistic is calculated on the training dataset, we can confirm the value by accessing and printing it; for example:

```
1 # print the mean calculated on the training dataset.
2 print(datagen.mean)
```

We can also confirm that the scaling procedure has had the desired effect by calculating the mean of a batch of images returned from the batch iterator. We would expect the mean to be a small value close to zero, but not zero because of the small number of images in the batch.

```
1 # get a batch
2 batchX, batchy = iterator.next()
3 # mean pixel value in the batch
4 print(batchX.shape, batchX.mean())
```

A better check would be to set the batch size to the size of the training dataset (e.g. 60,000 samples), retrieve one batch, then calculate the mean. It should be a very small value close to zero.

```
1 # try to flow the entire training dataset
2 iterator = datagen.flow(trainX, trainy, batch_size=len(trainX), shuffle=False)
3 # get a batch
4 batchX, batchy = iterator.next()
5 # mean pixel value in the batch
6 print(batchX.shape, batchX.mean())
```

The complete example is listed below.

```
1 # example of centering a image dataset
2 from keras.datasets import mnist
3 from keras.preprocessing.image import ImageDataGenerator
4 # load dataset
5 (trainX, trainy), (testX, testy) = mnist.load_data()
6 # reshape dataset to have a single channel
7 width, height, channels = trainX.shape[1], trainX.shape[2], trainX.shape[3]
8 trainX = trainX.reshape((trainX.shape[0], width, height, channels))
9 testX = testX.reshape((testX.shape[0], width, height, channels))
```

## Start Machine Learning

You can master applied Machine Learning  
**without math or fancy degrees.**  
Find out how in this *free* and *practical* course.

START MY EMAIL COURSE



```

10 # report per-image mean
11 print('Means train=%.3f, test=%.3f' % (trainX.mean(), testX.mean()))
12 # create generator that centers pixel values
13 datagen = ImageDataGenerator(featurewise_center=True)
14 # calculate the mean on the training dataset
15 datagen.fit(trainX)
16 print('Data Generator Mean: %.3f' % datagen.mean)
17 # demonstrate effect on a single batch of samples
18 iterator = datagen.flow(trainX, trainy, batch_size=64)
19 # get a batch
20 batchX, batchy = iterator.next()
21 # mean pixel value in the batch
22 print(batchX.shape, batchX.mean())
23 # demonstrate effect on entire training dataset
24 iterator = datagen.flow(trainX, trainy, batch_size=len(trainX), shuffle=False)
25 # get a batch
26 batchX, batchy = iterator.next()
27 # mean pixel value in the batch
28 print(batchX.shape, batchX.mean())

```

Running the example first reports the mean pixel value for the train and test datasets.

The MNIST dataset only has a single channel because the images are black and white (grayscale), but if the images were color, the mean pixel values would be calculated across all channels in all images in the training dataset, i.e. there would not be a separate mean value for each channel.

The ImageDataGenerator is fit on the training dataset and we can confirm that the mean pixel value matches our own manual calculation.

A single batch of centered images is retrieved and we can confirm that the mean pixel value is a small-ish value close to zero. The test is repeated using the entire training dataset as a the batch size, and in this case, the mean pixel value for the scaled dataset is a number very close to zero, confirming that centering is having the desired effect.

```

1 Means train=33.318, test=33.791
2 Data Generator Mean: 33.318
3 (64, 28, 28, 1) 0.09971977
4 (60000, 28, 28, 1) -1.9512918e-05

```

We can demonstrate centering with our convolutional neural network developed in the previous section.

The complete example with feature-wise centering is listed below.

```

1 # example of using ImageDataGenerator to center
2 from keras.datasets import mnist
3 from keras.utils import to_categorical
4 from keras.models import Sequential
5 from keras.layers import Conv2D
6 from keras.layers import MaxPooling2D
7 from keras.layers import Dense
8 from keras.layers import Flatten
9 from keras.preprocessing.image import ImageDataGenerator
10 # load dataset
11 (trainX, trainY), (testX, testY) = mnist.load_data()
12 # reshape dataset to have a single channel
13 width, height, channels = trainX.shape[1], trainX.shape[2], trainX.shape[3]
14 trainX = trainX.reshape((trainX.shape[0], width, height, channels))
15 testX = testX.reshape((testX.shape[0], width, height, channels))

```

## Start Machine Learning ×

You can master applied Machine Learning  
**without math or fancy degrees.**  
Find out how in this *free* and *practical* course.

START MY EMAIL COURSE

```

16 # one hot encode target values
17 trainY = to_categorical(trainY)
18 testY = to_categorical(testY)
19 # create generator to center images
20 datagen = ImageDataGenerator(featurewise_center=True)
21 # calculate mean on training dataset
22 datagen.fit(trainX)
23 # prepare an iterators to scale images
24 train_iterator = datagen.flow(trainX, trainY, batch_size=64)
25 test_iterator = datagen.flow(testX, testY, batch_size=64)
26 print('Batches train=%d, test=%d' % (len(train_iterator), len(test_iterator)))
27 # define model
28 model = Sequential()
29 model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(width, height, channels)))
30 model.add(MaxPooling2D((2, 2)))
31 model.add(Conv2D(64, (3, 3), activation='relu'))
32 model.add(MaxPooling2D((2, 2)))
33 model.add(Flatten())
34 model.add(Dense(64, activation='relu'))
35 model.add(Dense(10, activation='softmax'))
36 # compile model
37 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
38 # fit model with generator
39 model.fit_generator(train_iterator, steps_per_epoch=len(train_iterator), epochs=5)
40 # evaluate model
41 _, acc = model.evaluate_generator(test_iterator, steps=len(test_iterator), verbose=0)
42 print('Test Accuracy: %.3f' % (acc * 100))

```

Running the example prepares the ImageDataGenerator, centering images using statistics calculated on the training dataset.

We can see that performance starts off poor but does improve. The centered pixel values will have a range of about -227 to 227, and neural networks often train more efficiently with small inputs. Normalizing followed by centering would be a better approach in practice.

Importantly, the model is evaluated on the test dataset, where the images in the test dataset were centered using the mean value calculated on the training dataset. This is to avoid any data leakage.

```

1 Batches train=938, test=157
2 Epoch 1/5
3 938/938 [=====] - 12s 13ms/step - loss: 12.8824 - acc: 0.2001
4 Epoch 2/5
5 938/938 [=====] - 12s 13ms/step - loss: 6.1425 - acc: 0.5958
6 Epoch 3/5
7 938/938 [=====] - 12s 13ms/step - loss: 0.0678 - acc: 0.9796
8 Epoch 4/5
9 938/938 [=====] - 12s 13ms/step - loss: 0.0678 - acc: 0.9796
10 Epoch 5/5
11 938/938 [=====] - 12s 13ms/step - loss: 0.0678 - acc: 0.9796
12 Test Accuracy: 98.540

```

## How to Standardize Image With

Standardization is a data scaling technique that assures the distribution of the data to have a mean of zero and a standard deviation of one.

Data with this distribution is referred to as a standard normal distribution. Neural networks as the dataset sums to zero and the inputs are scaled to have a standard deviation of one.

### Start Machine Learning

You can master applied Machine Learning **without math or fancy degrees.**  
Find out how in this *free* and *practical* course.

START MY EMAIL COURSE

3.0 (e.g. 99.7 of the values will fall within three standard deviations of the mean).

Standardization of images is achieved by subtracting the mean pixel value and dividing the result by the standard deviation of the pixel values.

The mean and standard deviation statistics can be calculated on the training dataset, and as discussed in the previous section, Keras refers to this as feature-wise.

```
1 # feature-wise generator
2 datagen = ImageDataGenerator(featurewise_center=True, featurewise_std_normalization=True)
3 # calculate mean and standard deviation on the training dataset
4 datagen.fit(trainX)
```

The statistics can also be calculated then used to standardize each image separately, and Keras refers to this as sample-wise standardization.

```
1 # sample-wise standardization
2 datagen = ImageDataGenerator(samplewise_center=True, samplewise_std_normalization=True)
```

We will demonstrate the former or feature-wise approach to image standardization in this section. The effect will be batches of images with an approximate mean of zero and a standard deviation of one.

As with the previous section, we can confirm this with some simple experiments. The complete example is listed below.

```
1 # example of standardizing a image dataset
2 from keras.datasets import mnist
3 from keras.preprocessing.image import ImageDataGenerator
4 # load dataset
5 (trainX, trainy), (testX, testy) = mnist.load_data()
6 # reshape dataset to have a single channel
7 width, height, channels = trainX.shape[1], trainX.shape[2], 1
8 trainX = trainX.reshape((trainX.shape[0], width, height, channels))
9 testX = testX.reshape((testX.shape[0], width, height, channels))
10 # report pixel means and standard deviations
11 print('Statistics train=%.3f (%.3f), test=%.3f (%.3f)' % (trainX.mean(), trainX.std(), testX.me
12 # create generator that centers pixel values
13 datagen = ImageDataGenerator(featurewise_center=True, featurewise_std_normalization=True)
14 # calculate the mean on the training dataset
15 datagen.fit(trainX)
16 print('Data Generator mean=%.3f, std=%.3f' % (datagen.mean(), datagen.std()))
17 # demonstrate effect on a single batch of samples
18 iterator = datagen.flow(trainX, trainy, batch_size=64)
19 # get a batch
20 batchX, batchy = iterator.next()
21 # pixel stats in the batch
22 print(batchX.shape, batchX.mean(), batchX.std())
23 # demonstrate effect on entire training dataset
24 iterator = datagen.flow(trainX, trainy, batch_size=64)
25 # get a batch
26 batchX, batchy = iterator.next()
27 # pixel stats in the batch
28 print(batchX.shape, batchX.mean(), batchX.std())
```

Running the example first reports the mean and standard deviation of the training and test datasets.

## Start Machine Learning

You can master applied Machine Learning  
**without math or fancy degrees.**  
Find out how in this *free* and *practical* course.

START MY EMAIL COURSE

The data generator is then configured for feature-wise standardization and the statistics are calculated on the training dataset, matching what we would expect when the statistics are calculated manually.

A single batch of 64 standardized images is then retrieved and we can confirm that the mean and standard deviation of this small sample is close to the expected standard Gaussian.

The test is then repeated on the entire training dataset and we can confirm that the mean is indeed a very small value close to 0.0 and the standard deviation is a value very close to 1.0.

```
1 Statistics train=33.318 (78.567), test=33.791 (79.172)
2 Data Generator mean=33.318, std=78.567
3 (64, 28, 28, 1) 0.010656365 1.0107679
4 (60000, 28, 28, 1) -3.4560264e-07 0.9999998
```

Now that we have confirmed that the standardization of pixel values is being performed as we expect, we can apply the pixel scaling while fitting and evaluating a convolutional neural network model.

The complete example is listed below.

```
1 # example of using ImageDataGenerator to standardize images
2 from keras.datasets import mnist
3 from keras.utils import to_categorical
4 from keras.models import Sequential
5 from keras.layers import Conv2D
6 from keras.layers import MaxPooling2D
7 from keras.layers import Dense
8 from keras.layers import Flatten
9 from keras.preprocessing.image import ImageDataGenerator
10 # load dataset
11 (trainX, trainY), (testX, testY) = mnist.load_data()
12 # reshape dataset to have a single channel
13 width, height, channels = trainX.shape[1], trainX.shape[2], 1
14 trainX = trainX.reshape((trainX.shape[0], width, height, channels))
15 testX = testX.reshape((testX.shape[0], width, height, channels))
16 # one hot encode target values
17 trainY = to_categorical(trainY)
18 testY = to_categorical(testY)
19 # create generator to standardize images
20 datagen = ImageDataGenerator(featurewise_center=True, featurewise_std_normalization=True)
21 # calculate mean on training dataset
22 datagen.fit(trainX)
23 # prepare an iterators to scale images
24 train_iterator = datagen.flow(trainX, trainY, batch_size=64)
25 test_iterator = datagen.flow(testX, testY, batch_size=64)
26 print('Batches train=%d, test=%d' % (len(train_iterator), len(test_iterator)))
27 # define model
28 model = Sequential()
29 model.add(Conv2D(32, (3, 3), activation='relu'))
30 model.add(MaxPooling2D((2, 2)))
31 model.add(Conv2D(64, (3, 3), activation='relu'))
32 model.add(MaxPooling2D((2, 2)))
33 model.add(Flatten())
34 model.add(Dense(64, activation='relu'))
35 model.add(Dense(10, activation='softmax'))
36 # compile model
37 model.compile(optimizer='adam', loss='categorical_crossentropy')
38 # fit model with generator
39 model.fit_generator(train_iterator, steps_per_epoch=1000, epochs=10)
40 # evaluate model
41 _, acc = model.evaluate_generator(test_iterator, 1000)
```

## Start Machine Learning

You can master applied Machine Learning  
**without math or fancy degrees.**  
Find out how in this *free* and *practical* course.

START MY EMAIL COURSE

```
42 print('Test Accuracy: %.3f' % (acc * 100))
```

Running the example configures the ImageDataGenerator class to standardize images, calculates the required statistics on the training set only, then prepares the train and test iterators for fitting and evaluating the model respectively.

```
1 Epoch 1/5
2 938/938 [=====] - 12s 13ms/step - loss: 0.1342 - acc: 0.9592
3 Epoch 2/5
4 938/938 [=====] - 12s 13ms/step - loss: 0.0451 - acc: 0.9859
5 Epoch 3/5
6 938/938 [=====] - 12s 13ms/step - loss: 0.0309 - acc: 0.9906
7 Epoch 4/5
8 938/938 [=====] - 13s 13ms/step - loss: 0.0230 - acc: 0.9924
9 Epoch 5/5
10 938/938 [=====] - 13s 14ms/step - loss: 0.0182 - acc: 0.9941
11 Test Accuracy: 99.120
```

## Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- **Color.** Update an example to use an image dataset with color images and confirm that scaling is performed across the entire image rather than per-channel.
- **Sample-Wise.** Demonstrate an example of sample-wise centering or standardization of pixel images.
- **ZCA Whitening.** Demonstrate an example of using the ZCA approach to image data preparation.

If you explore any of these extensions, I'd love to know.

Post your findings in the comments below.

## Further Reading

This section provides more resources on the topic if you are looking to go deeper.

### API

- [MNIST database of handwritten digits, Keras API.](#)
- [Image Preprocessing Keras API](#)
- [Sequential Model Keras API](#)

### Articles

- [MNIST database, Wikipedia.](#)
- [68–95–99.7 rule, Wikipedia.](#)

## Summary

In this tutorial, you discovered how to use the ImageDataGenerator class to standardize images, calculate the required statistics on the training set only, then prepare the train and test iterators for fitting and evaluating the model respectively.

### Start Machine Learning

You can master applied Machine Learning **without math or fancy degrees.**  
Find out how in this *free* and *practical* course.

START MY EMAIL COURSE



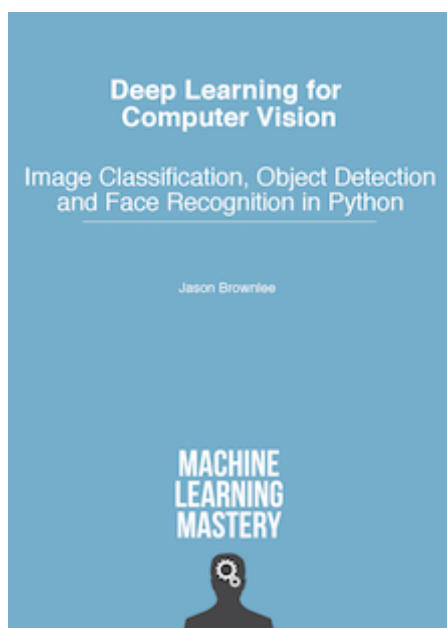
Specifically, you learned:

- How to configure and use the ImageDataGenerator class for train, validation, and test datasets of images.
- How to use the ImageDataGenerator to normalize pixel values when fitting and evaluating a convolutional neural network model.
- How to use the ImageDataGenerator to center and standardize pixel values when fitting and evaluating a convolutional neural network model.

Do you have any questions?

Ask your questions in the comments below and I will do my best to answer.

## Develop Deep Learning Models for Vision Today!



### Develop Your Own Vision Models in Minutes

...with just a few lines of python code

Discover how in my new Ebook:  
[Deep Learning for Computer Vision](#)

It provides **self-study tutorials** on topics like:  
*classification, object detection (yolo and rcnn), face recognition (vggface and facenet), data preparation* and much more...

### Finally Bring Deep Learning to your Vision Projects

Skip the Academics. Just Results.

[SEE WHAT'S INSIDE](#)

Tweet

Share

Share



#### About Jason Brownlee

Jason Brownlee, PhD is a machine learning expert with modern machine learning methods via Python.  
[View all posts by Jason Brownlee →](#)

### Start Machine Learning

You can master applied Machine Learning **without math or fancy degrees.**  
Find out how in this *free* and *practical* course.

[START MY EMAIL COURSE](#)

< [How to Load, Convert, and Save Images With the Keras API](#)

[A Gentle Introduction to](#)

[ats >](#)

## 32 Responses to *How to Normalize, Center, and Standardize Image Pixels in Keras*



**Julian Loaiza** April 3, 2019 at 2:17 pm #

REPLY ↩

Thanks for the article. First question, what is the impact in the model using sample-wise or feature-wise? Second question, If I'm going to use my model for production do I need to save the mean and standard deviation when I was trained my model with feature-wise method?



**Jason Brownlee** April 3, 2019 at 4:14 pm #

REPLY ↩

From the post:

“per-image (called sample-wise) or per-dataset (called feature-wise)”



**JG** April 3, 2019 at 8:59 pm #

REPLY ↩

Magisterial post Jason !. thanks.

Anyways, reading your post you say “...The pixel standardization is supported at two levels: either per-image (called sample-wise) or per-dataset (called feature-wise)...” the opposite of your previous answer!

My question is, I though ImageDataGenerator apply for data augmentation, that is to say to simulate we have more data (image) to train, and apply normalization, centering and standardization (rescaling) are parts of this more general “augmentation” methods. Anyway, I am confused about if data validation or test, have to be preprocessed in the same way as training data. i.e. It is clearly necessary to fit training data (with data augmentation) , but I though were no necessary for validation and test data ...am I right? or in which way we can maintain apart validation and test data ). Thanks



**Jason Brownlee** April 4, 2019 at 7:54 am #

Fixed, thanks.

It can be used for data prep and for data aug, and former case in this post.

If a statistic is calculated across samples (images) set and used on the val/test sets.

This can be done a few ways, but perhaps the eas but fit the instance (calculate stats) using the same

### Start Machine Learning



You can master applied Machine Learning **without math or fancy degrees.**  
Find out how in this *free* and *practical* course.

START MY EMAIL COURSE

the relevant `flow()` function.



**Jose** April 24, 2019 at 11:02 pm #

REPLY ↩

Awesome post!

It is clear that `flow_from_directory` handles the data from the 'directory', so in this case, the RAM is not overwhelmed. This is good when utilizing huge datasets.

Now, standardize with image generator requires to fit the data before, this implies that the data must be loaded in memory, so:

How about handling huge image datasets which cannot be contained in RAM?

How can one calculate mean and standard deviation on the huge training dataset in this case?



**Jason Brownlee** April 25, 2019 at 8:18 am #

REPLY ↩

There are many solutions:

- estimate stats from a smaller sample
- estimate stats using progressive loading
- use scaling that does not require global stats



**Jose** April 30, 2019 at 7:34 pm #

REPLY ↩

Thanks for the answer! ☐ Would you mind to point out where to find some “how to ... in keras” regarding the mentioned solutions?



**Jason Brownlee** May 1, 2019 at 7:01 am #

REPLY ↩

You can use the tutorial as a starting point and add in the additional config and test.



**Lau** June 12, 2019 at 1:18 pm #

Hi, thanks for sharing the valueable insights. I am looking for data augmentation using with ImageGenerator. May I know how to do this etc. with the Normalize, Center, and Standardize methods?

Once the `.fit()` has been applied on the training set, may I know how to Standardize on the validation set?

Thank you

## Start Machine Learning



You can master applied Machine Learning **without math or fancy degrees.** Find out how in this *free* and *practical* course.

START MY EMAIL COURSE



**Jason Brownlee** June 12, 2019 at 2:26 pm #

REPLY ↩

Yes, you can specify the augmentation directly, here's an example:  
<https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/>



**najeh** February 18, 2020 at 10:26 pm #

REPLY ↩

to create a grid of 3x3 images, we use "subplot(row, column , index)"  
but in your example you have used "subplot(330 +1 +i)"  
what do you mean by this code?



**Jason Brownlee** February 19, 2020 at 8:04 am #

REPLY ↩

That is the older API. Both do the same thing.



**Anthony** February 24, 2020 at 6:34 am #

REPLY ↩

Nice article, thank you!

One question – if I center the data whilst training, I assume that I have to do the same to an image at prediction time. If so, what is the best way to do that?

I have seen in other posts that a mean value is subtracted from the predicted image, and that mean value is the one calculated over the training data set. If that is the case, how can I obtain the mean (and possibly standard deviation) values for the training dataset?



**Jason Brownlee** February 24, 2020 at 7:49 am #

Yes, see best practices here:  
<https://machinelearningmastery.com/best-practices-for-convolutional-neural-networks/>



**Anthony** February 24, 2020 at 9:39 am #

Nice! Appreciated; thank you!

## Start Machine Learning



You can master applied Machine Learning  
**without math or fancy degrees.**  
Find out how in this *free* and *practical* course.

START MY EMAIL COURSE



**Ravi Theja** February 25, 2020 at 7:10 am #

REPLY ↩

Hey Jason,

Thanks for the great article. I'm trying to understand if there is any reason to use one technique over the other. Could you give me some pointers? I'm working on gray-scale data. Thank you!



**Jason Brownlee** February 25, 2020 at 7:53 am #

REPLY ↩

It's tough to justify. Often we follow what others have done and achieved good results or we test each method and use what gets the best results.

See this:

<https://machinelearningmastery.com/best-practices-for-preparing-and-augmenting-image-data-for-convolutional-neural-networks/>



**mohmaya** March 4, 2020 at 11:27 am #

REPLY ↩

Can we do both rescale, and then (samplewise\_center=True, samplewise\_std\_normalization=True) for an image? which techniques are mutually exclusive? What does it mean if I normalize an image and then do these two samplewise techniques, too.



**Jason Brownlee** March 4, 2020 at 1:34 pm #

REPLY ↩

Yes, but it might be odd.

It's a good question. It might be easier to start with a thesis/idea and test whether it improves modeling, rather than enumerating all scaling methods.



**Saar** April 27, 2020 at 11:05 pm #

Thank you for writing, it has been very educational.

One question though – after the normalization your code you said in the article, most values now lay in  $[-3,3]$ , while the input it got. Isn't tanh a better suited function after the

Kind Regards,  
Saar.

## Start Machine Learning



You can master applied Machine Learning **without math or fancy degrees.**  
Find out how in this *free* and *practical* course.

START MY EMAIL COURSE





**Jason Brownlee** April 28, 2020 at 6:46 am #

REPLY ↩

You're welcome.

Typically it isn't better in practice. Try and see for yourself. Remember, relu operates on the weighted sum, not raw inputs.



**Lars** May 6, 2020 at 5:58 pm #

REPLY ↩

Thanks for that nice tutorial!

I was wondering how I know if I should use standardization or not. I am working on a dataset for emotion recognition with a small amount of faces but many samples using VGG-Face.



**Jason Brownlee** May 7, 2020 at 6:42 am #

REPLY ↩

My best advice is to evaluate the model with and without the scaling operation and compare the results.

Use it if it results in a model with better skill.



**Nina** May 12, 2020 at 1:09 pm #

REPLY ↩

How to Setup a flow for validation data, assume that we can fit all images into CPU memory



**Jason Brownlee** May 12, 2020 at 1:32 pm #

REPLY ↩

Perhaps this will help:

<https://machinelearningmastery.com/how-to-load-large-datasets-from-directories-for-deep-learning-with-keras/>



**Nina** May 12, 2020 at 5:48 pm #

but the image not in the directory, I lo



**Jason Brownlee** May 13, 2020 at 6

## Start Machine Learning ×

You can master applied Machine Learning **without math or fancy degrees.** Find out how in this *free* and *practical* course.

**START MY EMAIL COURSE**

If you are using cifar10, you can load all images into memory and split the dataset or use a percentage for validation.

See this:

<https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/>



**Nina** May 12, 2020 at 5:46 pm #

REPLY ↩

How can I determine the number of node for intermediate dance layer, and final one, I try to understand the relation between the number of input neuron and output.



**Jason Brownlee** May 13, 2020 at 6:29 am #

REPLY ↩

Use trial and error:

<https://machinelearningmastery.com/faq/single-faq/how-many-layers-and-nodes-do-i-need-in-my-neural-network>



**Nina** May 13, 2020 at 1:54 pm #

REPLY ↩

Thanks, A lot for your replay, It's really helping ☐



**nkm** May 25, 2020 at 4:01 am #

REPLY ↩

Hi Mr Jason,

I get this error while using flow\_from\_directory:

“ImageDataGenerator specifies `featurewise_std_normalization`, but it hasn't been fit on any training data.”

there is a step of `train_datagen.fit(x_train)`, which requi  
can I implement `featurewise_std_normalization` augme



**Jason Brownlee** May 25, 2020 at 5:56 am #

Sorry to hear that, this will help:

<https://machinelearningmastery.com/faq/single-faq>

## Start Machine Learning



You can master applied Machine Learning  
**without math or fancy degrees.**  
Find out how in this *free* and *practical* course.

START MY EMAIL COURSE

# Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT






**Welcome!**  
My name is *Jason Brownlee* PhD, and I **help developers** get results with **machine learning**.  
[Read more](#)

## Never miss a tutorial:



## Picked for you:

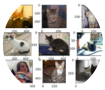
- [How to Train an Object Detection Model with Keras](#)
- [How to Develop a Face Recognition System Using](#)
- [How to Perform Object Detection With YOLOv3 in K](#)

Start Machine Learning

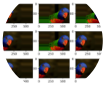
You can master applied Machine Learning **without math or fancy degrees.** Find out how in this *free* and *practical* course.

Email Address

START MY EMAIL COURSE



How to Classify Photos of Dogs and Cats (with 97% accuracy)



How to Configure Image Data Augmentation in Keras

## Loving the Tutorials?

The Deep Learning for Computer Vision EBook is where I keep the ***Really Good*** stuff.

SEE WHAT'S INSIDE

---

© 2020 Machine Learning Mastery Pty. Ltd. All Rights Reserved.

Address: PO Box 206, Vermont Victoria 3133, Australia. | ACN: 626 223 336.

[LinkedIn](#) | [Twitter](#) | [Facebook](#) | [Newsletter](#) | [RSS](#)

[Privacy](#) | [Disclaimer](#) | [Terms](#) | [Contact](#) | [Sitemap](#) | [Search](#)

## Start Machine Learning ×

You can master applied Machine Learning  
**without math or fancy degrees.**  
Find out how in this *free* and *practical* course.

START MY EMAIL COURSE